Learning Google Cloud Vertex AI

Build, deploy, and manage machine learning models with Vertex AI

Hemanth Kumar K

bpb

Learning Google Cloud Vertex AI

Build, deploy, and manage machine learning models with Vertex AI

Hemanth Kumar K

www.bpbonline.com

First published: 2024

Published by BPB Online
WeWork
119 Marylebone Road
London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55515-353

www.bpbonline.com

Dedicated to

All the people who are changing the world
through technology

About the Author

Hemanth Kumar K has more than decade of experience in the corporate industry, started his career in telecom domain worked on multiple projects and clients across the globe to plan and optimize mobile networks. Got fascinated towards data & machine learning changed his focus and career path towards data science & analytics. He has worked on many data analytics projects to solve business problems in various domains such as insurance, telecom, media, health, engineering analytics. In addition to GCP he has also worked on Azure platform.

About the Reviewers

- **Dr. Muthu Kumaraswamy B** is an Associate Director at Searce where he leads a team of 100 + Machine learning engineers and delivers custom machine learning solutions. He has over 15 years of experience in the IT industry, and has worked on a variety of projects in the areas of software development, cloud computing, and artificial intelligence. He has a Ph.D. from the University Madras.

  Dr. Muthu Kumaraswamy is an expert in a variety of programming languages, including JavaScript, Python, and C++. He is also proficient in a variety of cloud computing platforms, including Google Cloud Platform and Amazon Web Services. He is a strong advocate for using artificial intelligence to solve real-world problems, and has developed several AI-powered solutions for his clients.

  Dr. Muthu Kumaraswamy is a frequent speaker at industry conferences and events. He is also a regular contributor to technical blogs and publications. He is passionate about sharing his knowledge and expertise with others, and is always looking for new ways to help people learn and grow.

  Here are some of his notable achievements:

  - Developed a cloud-based AI solution that helped a large manufacturing company improve its production efficiency by 15%.
  - Led a team of engineers in the development of a new Machine learning platform that was used by a financial services company to automate its risk management processes.
  - Published several articles in technical journals and magazines on the topics of software development, cloud computing, and artificial intelligence.
  - Presented at several industry conferences on the use of AI to solve real-world problems.
  - Dr. Muthu Kumaraswamy is a highly skilled and experienced Machine learning engineer with a proven track record of success in delivering high-quality solutions to his clients. He is a valuable asset to any team and would be a great addition to your company

❖ **Abdul Hannan Siddiqui** is a Passionate and Innovative Machine Learning Engineer with over 5 years of experience in MLOps, Time Series, Predictive Modeling, Computer Vision, and Natural Language Processing. He is a Google Certified Machine Learning Engineer with multiple Google Cloud Platform certifications, specializing in Data Engineering using Python, ETL, and effectively handling large-scale migration processes to the cloud, as well as data analytics on Google BigQuery.

Abdul Hannan Siddiqui is currently working as a Senior Machine Learning Engineer at Argility. Prior to that, he worked as a Machine Learning Engineer at Quantiphi and as a Data Engineer at Accenture. His expertise lies in MLOps, Time Series, Predictive Modeling, Computer Vision, and Natural Language Processing, and he is highly skilled in using Python for Data Engineering and ETL processes. With his extensive experience and Google Cloud Platform certifications, Abdul Hannan Siddiqui is dedicated to leveraging his skills to drive successful machine learning and data science projects, ensuring optimal outcomes for his clients or organization.

Acknowledgements

I want to sincerely thank my family and friends for their continuous support and inspiration as I wrote this book.

I'm also appreciative of BPB Publications for their advice and know-how in helping me finish this book. This book had a lengthy revision process that benefited greatly from the participation and cooperation of editors, technical specialists, and reviewers.

Throughout my many years in the corporate industry, I would also like to recognize the important contributions of my coworkers and colleagues, who have taught me a lot and given me insightful criticism of my work.

Finally, I want to express my gratitude to all the readers who have shown interest in and support for my work. Your words of support have been

Preface

In today's world data enthusiast are expected to have expertise in one more cloud platforms. Solving business problems through data on cloud requires comprehensive understanding not just of algorithms but also cloud platforms and its components.

This book aims in helping you understand Vertex AI component of Google Cloud Platform (GCP) with real time examples. Before diving into Vertex AI, book gives you fair idea about google cloud platform and its components and also the positioning of Vertex AI. For all the examples used in this book for model training data is either uploaded to google cloud storage (GCS) or big query, so we will start with GCS, big query and also about the Identity access and management (IAM). Vertex AI has lot of components starting from datasets, Workbench, Pipelines, training, experiments, models, end points and features. The primary objective here is to learn these features of Vertex AI concepts with real-time use cases.

Chapter 1: Basics of Google Cloud Platform: starts with a fundamental understanding of Google Cloud Platform in this chapter. GCP's footprint, hierarchy, and platform, as well as how we can interact with GCP's components. GCP's several cloud service models (IAAS, PAAS, SAAS). A high-level overview of GCP's various components. We also take a look at GCP's storage capabilities, including how to set up Google Cloud storage and upload files to it. How to use Google's Big Query. We will comprehend the necessity and significance Incident and Access Management (IAM) in the platform.

Chapter 2: Introduction to Vertex AI and AutoML Tabular: In this chapter, the GCP Vertex AI component will be discussed. The integration of automl and custom model features into a single GCP component. We also comprehend the key distinctions between automl and custom models. We'll start with a walkthrough of how to create a dataset for structural data and how to train an automl model for it. We'll look at how to deploy the model and get batch and online predictions.

Chapter 3: AutoML Image, Text, and Pre-built Models: We'll work with unstructured data in this chapter. How do we make datasets out of unstructured data? (images and text data). How can an automl model be trained for picture and text classification tasks? How to use the trained model to make predictions in batches and in real time. We'll start to see how we can use GCP's pre-trained models to solve our problem.

Chapter 4: Vertex AI Workbench and Custom Model Training: We'll begin working on how to use Vertex AI for custom model training. We'll begin with a basic model training example then progress to a more complex example. We'll work on the Vertex AI Workbench component, evaluate its various possibilities, and develop a single workbench for custom model training. We'll introduce you to the principles of Container, Docker, and Kubernetes before we go into custom Vertex AI models.

Chapter 5: Vertex AI Custom Model Hyperparameters and Deployment: We started with a modest model training exercise on Vertex AI to get a feel for custom model training. In this chapter, we'll look at a more difficult task: training a model with hyperparameter adjustment. We'll discuss how to deploy the custom model for serving predictions once it's been trained.

Chapter 6: Introduction to Pipelines and Kubeflow: In this chapter we will introduce you to the concept of MLOPS. Understanding machine learning life cycle. We will see how to build a pipeline using automl of Vertex AI with an example.

Chapter 7: Pipelines using Kubeflow for Custom Models: This chapter starts with the introduction of Kubeflow. How can we use Kubeflow components to build pipelines? How to create jobs for model training using container. CI/CD for Kubeflow pipelines.

Chapter 8: Pipelines using TensorFlow Extended: The introduction to TFX is the first part of this chapter. How can we construct pipelines using TFX components? How to use a container to construct jobs for model training. For TFX pipelines, CI/CD is used.

Chapter 9: Vertex AI Feature Store: This chapter is based on the Vertex AI Feature Store. We'll begin with an overview of the Feature Store. What is the purpose of feature stores? The fundamentals of feature stores are covered first, followed by a practical part on feature stores.

Chapter 10: Explainable AI: With Explainable AI, we'll begin investigating machine learning models in this chapter. How can we use explainable AI to better understand ML models for image categorization and how can we use explainable AI to better identify bias in ML models?

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/of74aa7

The code bundle for the book is also hosted on GitHub at **https://github.com/bpbpublications/Learning-Google-Cloud-Vertex-AI**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Table of Contents

1. Basics of Google Cloud Platform 1
Introduction 1
Structure 1
Objectives 2
Introduction to Cloud 2
Advantages of Cloud 3
Importance of Cloud for data scientist 5
Types of Cloud 6
Introduction to Google Cloud Platform 7
Account creation on Google Cloud Platform 8
Footprint of Google Cloud Platform 11
Cloud Service Model 12
Services offered by GCP 14
Hierarchy of GCP 17
New Project Creation 18
Deletion of Project 20
Interacting with GCP services 21
Google Cloud Platform Console 21
Command Line Interface 22
APIs 23
Storage 24
Working with Google Cloud Storage 27
Deletion of bucket 34
Compute 34
Summary of Compute services of GCP 38
Creation of compute Engine (VM instances) 38
Accessing the VM instance 44
Deletion of VM instance 46
BigQuery 47
Working with BigQuery 48

Identity and access management........55
Conclusion........58
Questions........58

2. Introduction to Vertex AI and AutoML Tabular........59
Introduction........59
Structure........59
Objectives........60
Introduction to Vertex AI........60
Key features........61
Vertex AI prediction service........62
Working with Vertex AI........62
Vertex AI AutoML........65
Creation of tabular datasets........66
Model training........71
Model evaluation........76
Batch predictions........80
Model deployment for online predictions........81
Service account creation........86
Serving online predictions........90
Model un-deployment and deleting end point........92
Model deletion........94
Dataset deletion........94
Conclusion........95
Questions........95

3. AutoML Image, Text, and Pre-built Models........97
Introduction........97
Structure........97
Objectives........98
Vertex AI AutoML for image data........98
Image dataset creation........101
Model training image........105

Model evaluation image 109
Batch Predictions image 111
Model deployment for online predictions image 114
Serving online predictions image 116
Vertex AI AutoML for text data 119
Text dataset creation 121
Model training text 124
Model evaluation text 127
Batch Predictions text 128
Model deployment for online predictions text 131
Serving online predictions text 133
Pre-built models in GCP 135
Benefits of AutoML 138
Limitations of AutoML 138
Conclusion 139
Questions 139

4. Vertex AI Workbench and Custom Model Training 141
Introduction 141
Structure 141
Objectives 142
Vertex AI workbench 142
Vertex AI Workbench creation and working 143
Data for building custom model 147
Introduction to Containers and Dockers 148
Creation of Dockerfile 149
Model building 154
Image creation 158
Pushing image to container registry 161
Submitting the custom model training job 162
Completion of custom model training job 167
Deletion of resources 168
Conclusion 169

Questions....169

5. Vertex AI Custom Model Hyperparameter and Deployment....171

Introduction....171

Structure....171

Objectives....172

Hyperparameter in machine learning....172

Working of hyperparameters tuning....173

Vertex AI Vizier....173

Data for building custom model....174

Creation of workbench....175

Creation of Dockerfile....176

Model building code....177

Image creation....182

Submitting the custom model training job....183

Completion of custom model training job....189

Model importing....190

Model deployment and predictions....193

Submitting training job with Python SDK....198

Deletion of resources....201

Conclusion....201

Questions....201

6. Introduction to Pipelines and Kubeflow....203

Introduction....203

Structure....203

Objectives....204

What is machine learning pipeline....204

Vertex AI pipelines....204

Benefits of machine learning pipelines....205

Execution....205

Model versioning and tracking....205

Troubleshooting....205

Resource utilization....205

Introduction to Kubeflow 205
Components of Kubeflow 206
Tasks of Kubeflow 206
Data for model training 207
API enablement 208
Additional permission for compute engine 210
Pipeline code walk through 212
Execution of Pipeline 221
Deleting resources 225
Conclusion 225
Questions 225

7. Pipelines using Kubeflow for Custom Models 227
Introduction 227
Structure 227
Objectives 228
Data for model training 228
Additional permissions 229
Creation of Workbench 231
Pipeline code walk through 232
Pipeline 243
Pipeline comparison 247
Deletion of resources 249
Differences between Vertex AI and Kubeflow pipelines 249
Conclusion 249
Questions 249

8. Pipelines using TensorFlow Extended 251
Introduction 251
Structure 251
Objectives 252
What is TensorFlow Extended 252
TFX Pipelines 252

Components of TFX *253*
Types of custom components *254*
Functionalities of custom components *254*
Data for pipeline building 256
Pipeline code walk through 257
Deletion of resources 272
Conclusion 273
Questions 273

9. Vertex AI Feature Store 275
Introduction 275
Structure 275
Objectives 276
Knowing Vertex AI feature store 276
Hierarchy of feature store 276
Advantages of feature store 277
Disadvantages of feature store 278
Data for feature store exercise *279*
Working on feature store using GUI 279
Working on feature store using Python 287
Deleting resources 297
Best practices for feature store 297
Conclusion 298
Questions 298

10. Explainable AI 299
Introduction 299
Structure 299
Objectives 300
What is Explainable AI 300
Need of Explainable AI 300
XAI on Vertex AI 302
Example-based explanations *302*

Feature-based explanations *303*
Feature attribution methods *303*
Data for Explainable AI exercise 304
Model training for image data 305
Image classification model deployment 310
Explanations for image classification 312
Tabular classification model deployment 315
Explanations for tabular data (classification) 318
Deletion of resources 320
Limitations of Explainable AI 321
Conclusion 321
Questions 322

Index 323-328

Chapter 1
Basics of Google Cloud Platform

Introduction

You will learn about the Google cloud platform in this chapter, as well as its benefits and the role it plays in today's digital revolution. Basic knowledge of cloud computing, including cloud service models, GCP account creation, footprint, range of services, and GCP hierarchy. This chapter will also introduce a few key GCP services, including storage, computation, google BigQuery and identity and access management, is then provided.

Structure

In this chapter, we will cover the following topics:

- Introduction and basics of Cloud platform
- Advantages of Cloud
- Importance of Cloud for data scientists
- Types of Cloud
- Introduction to Google Cloud platform
- Footprint of Google Cloud

- Cloud service model
- Services of GCP
- Hierarchy of GCP
- Interacting with GCP services
- Storage in GCP
- Compute in GCP
- BigQuery
- Identity and Access Management

Objectives

Before diving into the Vertex AI of the Google Cloud platform, it is very essential to grasp a few significant principles and vital services of the cloud platform. Users will have a solid understanding of the GCP components and services by the time this chapter ends. Detailed instructions for using GCP's storage, compute, and BigQuery services are included.

Introduction to Cloud

The term *Cloud* describes the applications and databases that run on servers that can be accessed over the Internet. Data centers across the globe host the cloud servers. Organizations can avoid managing physical servers or running software on their own computers by utilizing cloud computing. The cloud enables users to access the same files and applications from almost any device, because the computing and storage takes place on servers in a data center, instead of locally on the user device.

For businesses, switching to cloud computing removes some IT costs and overhead: for instance, they no longer need to update and maintain their own servers, as the cloud vendor they are using will do that.

Advantages of Cloud

There are various advantages of cloud as shown in *Figure 1.1*, and mentioned as follows:

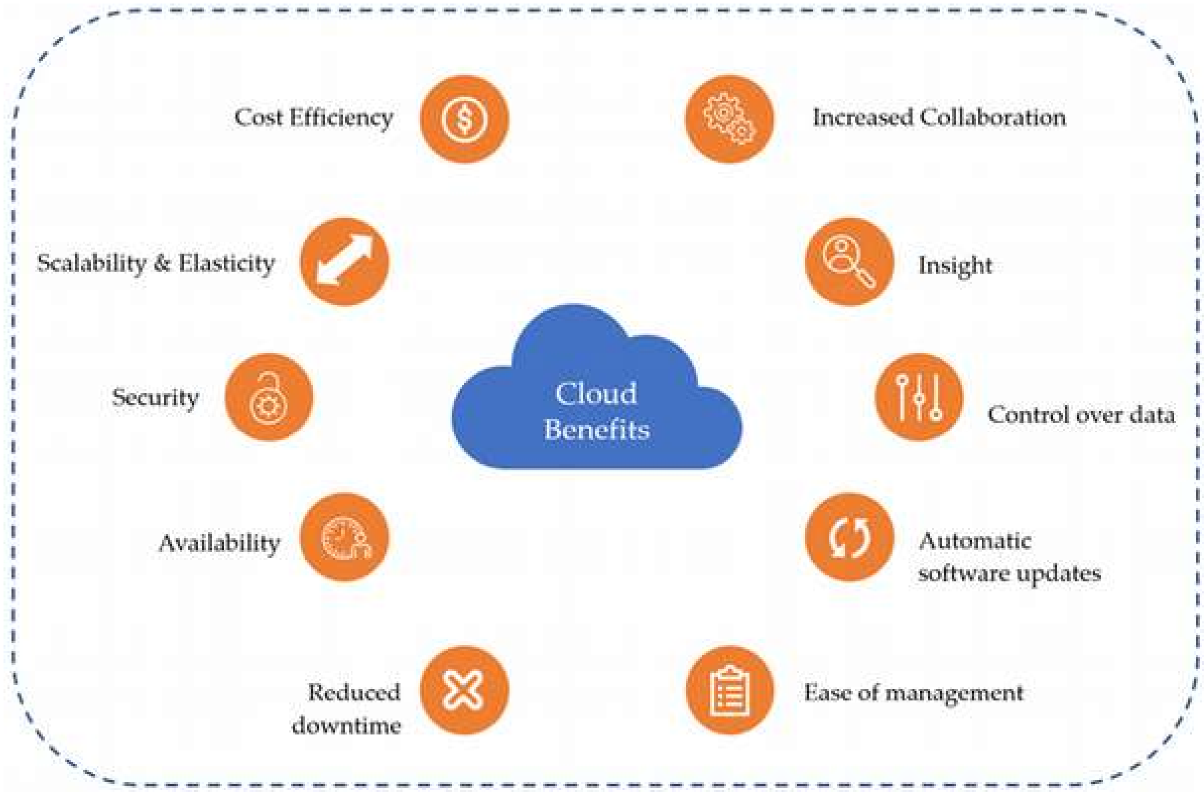


***Figure 1.1**: Advantages of Cloud platform*

- **Cost efficiency**: In terms of IT infrastructure management, cloud computing is undoubtedly the most cost-effective option. It is incredibly affordable for organizations of any size to transition from on-premises hardware to the cloud thanks to a variety of pay-as-you-go and other scalable choices. Using cloud resources instead of purchasing costly server equipment and PCs that need a lot of time to set up and maintain, such as long hours of setup and maintenance. Cloud also helps in reduced spending on compute, storage, network, operational and upgrade expenses.
- **Scalability and elasticity**: Overall, cloud hosting is more flexible than hosting on a local machine. You do not have to undertake a costly (and time-consuming) upgrade to your IT infrastructure if you need more bandwidth. This increased degree of latitude and adaptability may have a major impact on productivity.

Elasticity is only employed for a short amount of time to deal with rapid shifts in workload. This is a short-term strategy used to meet spikes in demand, whether they are unanticipated or seasonal. The static increase in workload is met through scalability. To cope with an anticipated rise in demand, a long-term approach to scalability is used.

- **Security**: Cloud platform provides a multitude of cutting-edge security measures, which ensure the safe storage and management of any data. Granular permissions and access control using federated roles are two examples of features that may help limit access to sensitive data to just those workers who have a legitimate need for it. This helps reduce the attack surface that is available to hostile actors. Authentication, access control, and encryption are some of the fundamental safeguards that providers of cloud storage put in place to secure their platforms and the data that is processed on those platforms. After that, users can implement additional security measures of their own, in addition to these precautions, to further strengthen cloud data protection and restrict access to sensitive information stored in the cloud.
- **Availability**: The vast majority of cloud service providers are quite dependable in terms of the provision of their services; in fact, the vast majority of them maintain an uptime of 99.9 percent. Moving to the cloud should be done with the intention of achieving high availability. The goal is to make your company's goods, services, and tools accessible to your clients and workers at any time of day and from any location in the world using any device that can connect to the internet.
- **Reduced downtime**: Cloud based solutions provide the ability to operate critical systems and data directly from the cloud or to restore them to any location. During a catastrophic event involving information technology, they make it easier for you to get these systems back online, reducing the amount of manual work required by conventional recovery techniques.
- **Increased Collaboration**: Developers, QA, operations, security, and product architects are all exposed to the same infrastructure and may work concurrently without tripping on one another's toes in cloud settings. To minimize disputes and misunderstanding, cloud roles and permissions provide more visibility and monitoring of who performed what and when. Different cloud environments, such as staging, QA, demo, and pre-production, may be created for specialized reasons. The cloud makes transparent collaboration simpler and promotes it.

- **Insight**: A bird's-eye perspective of your data is also provided through the integrated cloud analytics that are offered by cloud platforms. When your data is kept in the cloud, it is much simpler to put in place, monitoring systems and create individualized reports for doing information analysis throughout the whole organization. You will be able to improve efficiency and construct action plans based on these insights, which will allow your organization to fulfil its objectives.
- **Control over data**: Cloud provides you total visibility and control over your data. You have complete control over which users are granted access to which levels of specified data. This not only gives you control, but also helps simplify work by ensuring that staff members are aware of the tasks they have been allocated. Additionally, it will make working together much simpler. Because several users may make edits to the same copy of the text at the same time, there is no need that multiple copies of the document be distributed to the public.
- **Automatic software updates**: There is nothing more cumbersome than being required to wait for the installation of system upgrades, especially for those who already have a lot on their plates. Applications that are hosted in the cloud instantly refresh and update themselves, eliminating the need for an IT personnel to carry out manual updates for the whole organization. This saves critical time and money that would have been spent on consulting from other sources.
- **Ease of managing**: The use of cloud can streamline and improve IT maintenance and management capabilities through the use of agreements supported by SLA, centralized resource administration, and managed infrastructure. Users can take advantage of a simple user interface without having to worry about installing anything. In addition, users are provided with management, maintenance, and delivery of the IT services.

Importance of Cloud for data scientist

Since the beginning of the previous decade, the expansion of data has followed an exponential pattern, and this trend is expected to continue. The safe and secure storage of data should be one of the top priorities of every company. The cloud is usually the top option when it comes to storing and processing the enormous quantity of data since it has all of the advantages that were discussed above. As a consequence of this, a data scientist in today's world has to have experience with cloud computing in addition to their expertise in statistics, machine learning algorithms, and other areas.

However, due to the low processing capacity of their CPU, they are unable to carry out these responsibilities in a timely way, assuming that they are even capable of doing so at all. In addition, the memory of the machine is often incapable of storing massive datasets because of their size. It determines how quickly the assignment is performed and how well it was accomplished overall. Data scientists are now able to investigate more extensive collections of data without being constrained by the capabilities of their local workstations thanks to the cloud. Utilizing the cloud might result in a decrease in the cost of infrastructure since it eliminates the requirement for a physical server. In addition, depending on the cloud for data storage can lead to a reduction in the cost of infrastructure. In addition to offering data storage services, many cloud platforms including google cloud platform also has other services caterings to data ingestion, data processing, analytics, AI and data visualization.

Types of Cloud

There are three types of cloud based on different capabilities:

- Public Cloud
- Private Cloud
- Hybrid Cloud

Public Cloud: The public cloud is a massive collection of readily available computing resources, including networking, memory, processing elements, and storage. Users can rent these resources, which are housed in one of the public cloud vendors globally dispersed and fully managed datacenters, to create your IT architecture. Using a web browser, users have access to your resources in this form of cloud. Google Cloud Platform is an example for Public Cloud.

A major advantage of the public cloud is that the underlying hardware and logic are hosted, owned, and maintained by each vendor. Customers are not responsible for purchasing or maintaining the physical components that comprise their public cloud IT solutions. In addition, **Service Level Agreements** (**SLAs**) bind each provider to a monthly uptime percentage and security guarantee in accordance with regulations.

Private Cloud: Unlike public clouds, private clouds are owned and operated only by a single organization. They have usually been housed in the company's datacenter and run on the organization 's own equipment. To host their private cloud on their equipment, however, an organization may use a third-party supplier. Even if the resources are housed in a remotely managed datacenter, private cloud has certain characteristics with public cloud in this case. They may be able to provide certain administrative services but they would not be able to offer the full range of public cloud services.

If the private cloud is housed in your own datacenter, organization have complete control over the whole system. A self-hosted private cloud may help to comply with some of the stricter security and compliance regulations.

Hybrid Cloud: This kind of cloud computing is a blend and integration of both public and private clouds, as the name of this form of cloud computing indicates. In this manner, it will be able to provide you with the advantages associated with a variety of cloud kinds when it comes to cloud computing. It enables a larger degree of flexibility in terms of the transmission of data and expands the alternatives available to a company for its adoption. This guarantees a high level of control as well as an easy transition while giving everything at rates that are more economical.

Introduction to Google Cloud Platform

Google Cloud Platform is one of the hyper scale infrastructure providers in the industry. It is a collection of cloud computing services that are offered by Google. These services operate on the same infrastructure that Google employs for its end-user products, including YouTube, Gmail, and a number of other offerings. The Google Cloud Platform provides a wide range of services, such as computing, storage, and networking, among other things.

Google Cloud Platform was first launched in 2008, and as of now, it is the third cloud platform that sees the most widespread use. Additionally, there is a growing need for platforms that are hosted on the cloud.

The Google cloud gives us a service-centric perspective of all our environments in addition to providing a standard platform and data analysis for deployments, regardless of where they are physically located. Using the capabilities of sophisticated analytics and machine learning offered by Google Cloud, we can extract the most useful insights from our data. Users will be able to automate procedures, generate predictions, and simplify administration and operations with the support of Google's serverless data analytics and machine learning platform. The services provided by Google Cloud encrypt data while it is stored, while it is being sent, and while it is being used. Advanced security mechanisms protect the privacy of data.

Account creation on Google Cloud Platform

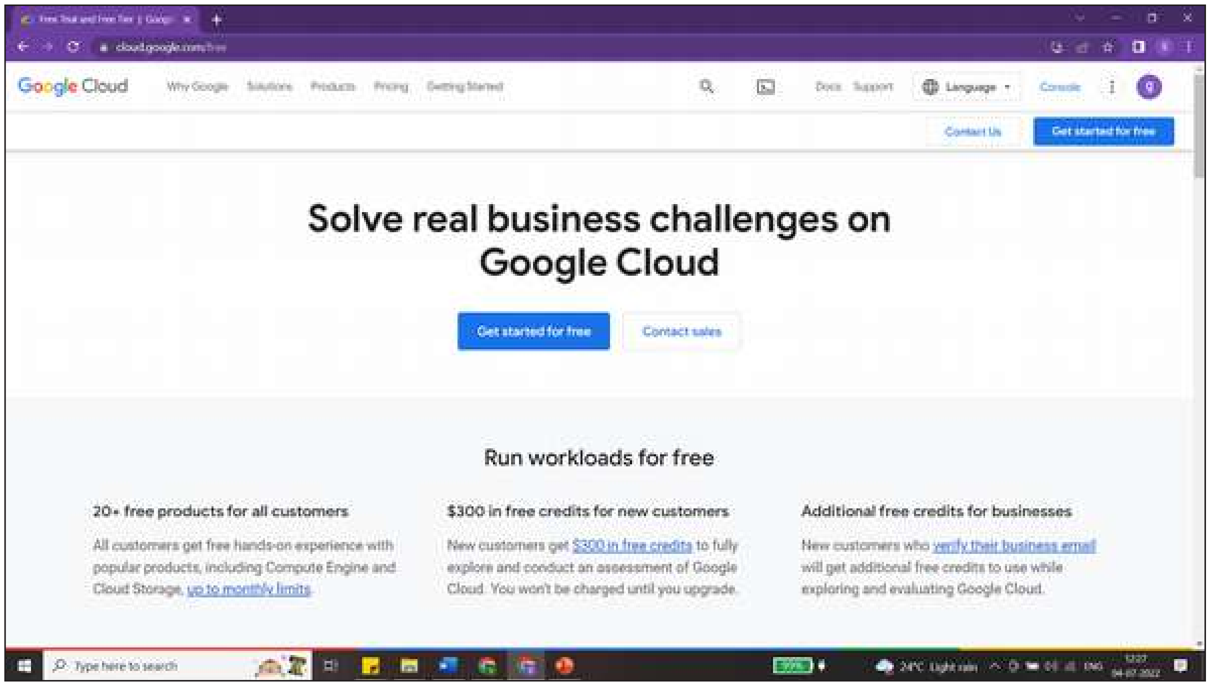
Users can create free GCP account from the link **https://cloud.google.com/free**.

Free account provides 300$ credit for a period of 90 days.

Steps for creating a free account are as follows:

1. Open **https://cloud.google.com/free**.
2. Click on **Get started for free**.

   The opening screen looks like *Figure 1.2*:



Solve real business challenges on Google Cloud

Run workloads for free

20+ free products for all customers

All customers get free hands-on experience with popular products, including Compute Engine and Cloud Storage, up to monthly limits.

$300 in free credits for new customers

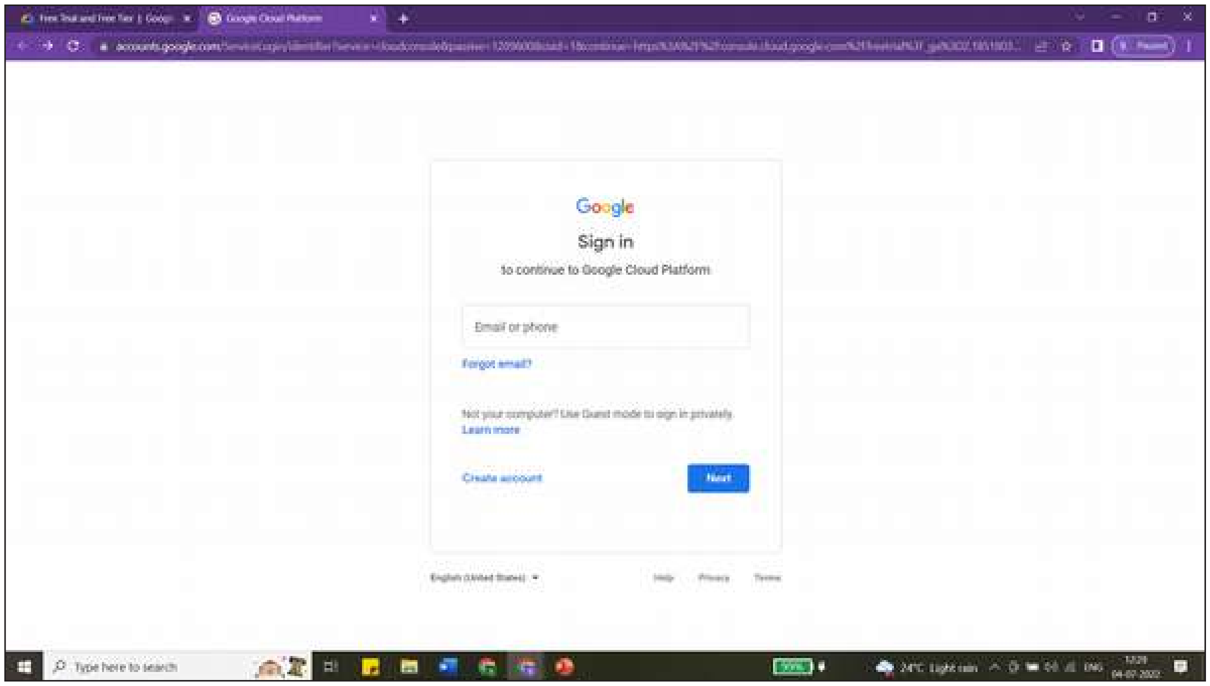
New customers get $300 in free credits to fully explore and conduct an assessment of Google Cloud. You won't be charged until you upgrade.

Additional free credits for businesses

New customers who verify their business email will get additional free credits to use while exploring and evaluating Google Cloud.

***Figure 1.2**: GCP account creation*

3. Login with your Gmail credentials, create one if you do not have. This can be seen illustrated in *Figure 1.3*:


Google

Sign in

to continue to Google Cloud Platform

Email or phone

Forgot email?

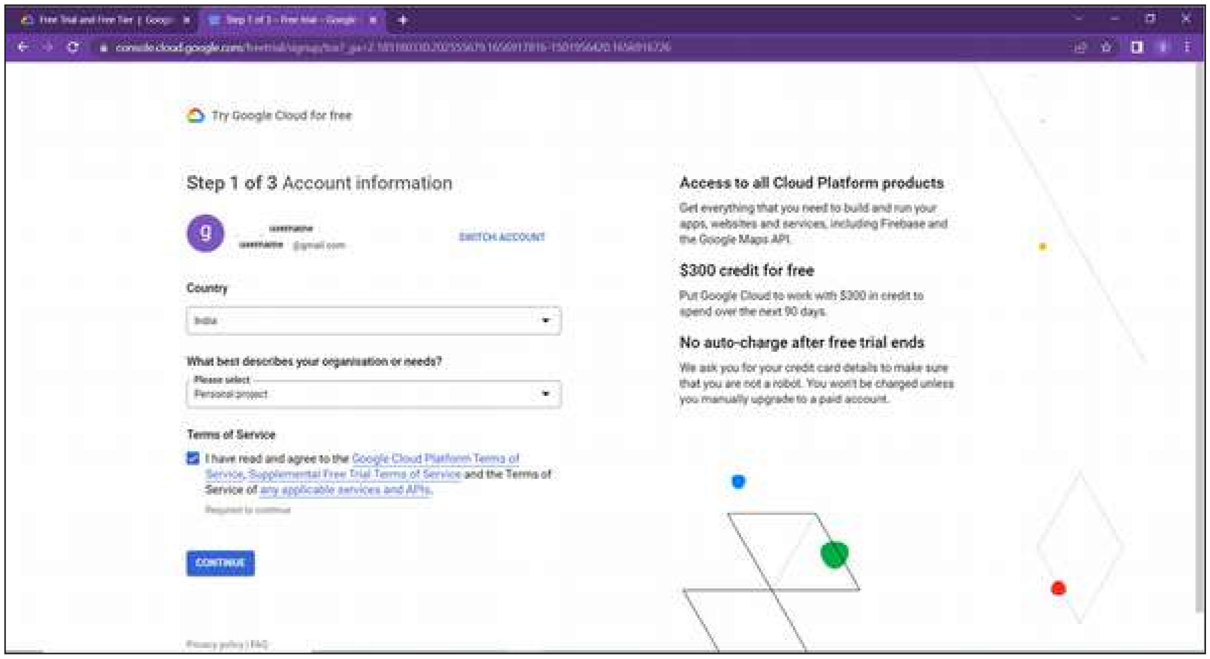
Not your computer? Use Guest mode to sign in privately.
Learn more

Create account Next

English (United States) Help Privacy Terms

***Figure 1.3**: GCP account creation enter valid mail address*

4. Selection of **COUNTRY** and needs:


Try Google Cloud for free

Step 1 of 3 Account information

username
username @gmail.com SWITCH ACCOUNT

Country

India

What best describes your organisation or needs?

Please select
Personal project

Terms of Service

☑ I have read and agree to the Google Cloud Platform Terms of Service, Supplemental Free Trial Terms of Service and the Terms of Service of any applicable services and APIs.

Required to continue

CONTINUE

Privacy policy | FAQ

Access to all Cloud Platform products

Get everything that you need to build and run your apps, websites and services, including Firebase and the Google Maps API.

$300 credit for free

Put Google Cloud to work with $300 in credit to spend over the next 90 days.

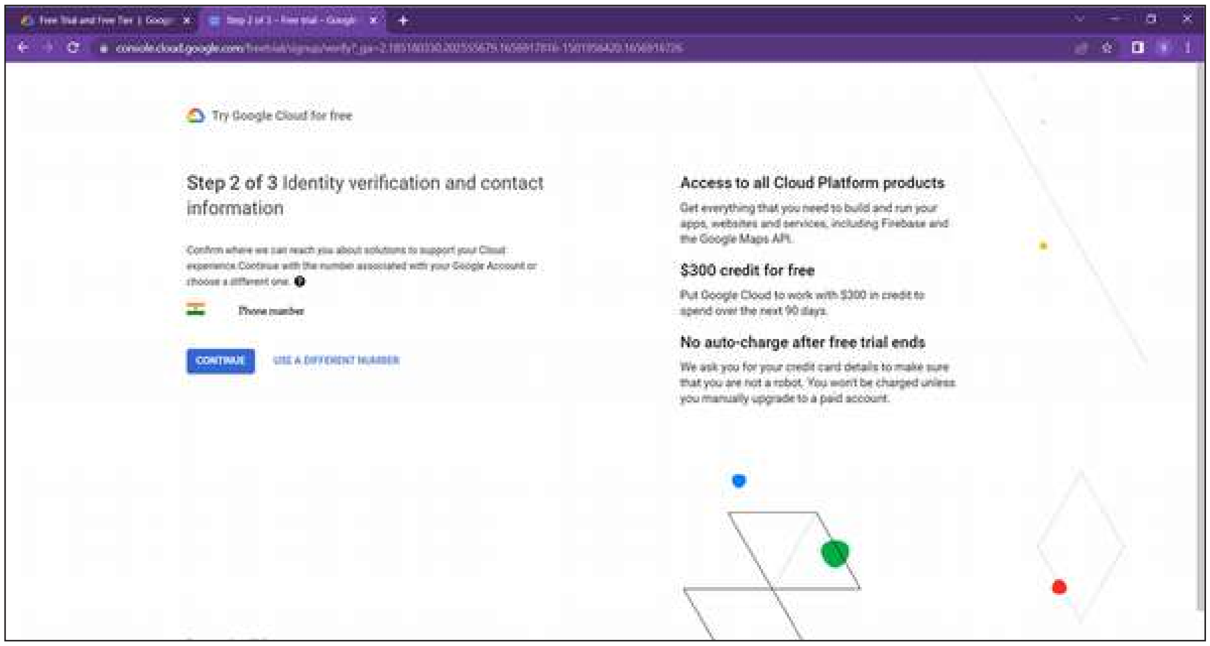
No auto-charge after free trial ends

We ask you for your credit card details to make sure that you are not a robot. You won't be charged unless you manually upgrade to a paid account.

***Figure 1.4**: GCP account creation country selection*

5. Select the **Country** and project. Check the Terms of service and click on **CONTINUE**.

6. Provide phone number for the identity verification as shown in *Figure 1.5*:


Try Google Cloud for free

Step 2 of 3 Identity verification and contact information

Confirm where we can reach you about solutions to support your Cloud experience. Continue with the number associated with your Google Account or choose a different one.

Phone number

CONTINUE USE A DIFFERENT NUMBER

Access to all Cloud Platform products

Get everything that you need to build and run your apps, websites and services, including Firebase and the Google Maps API.

$300 credit for free

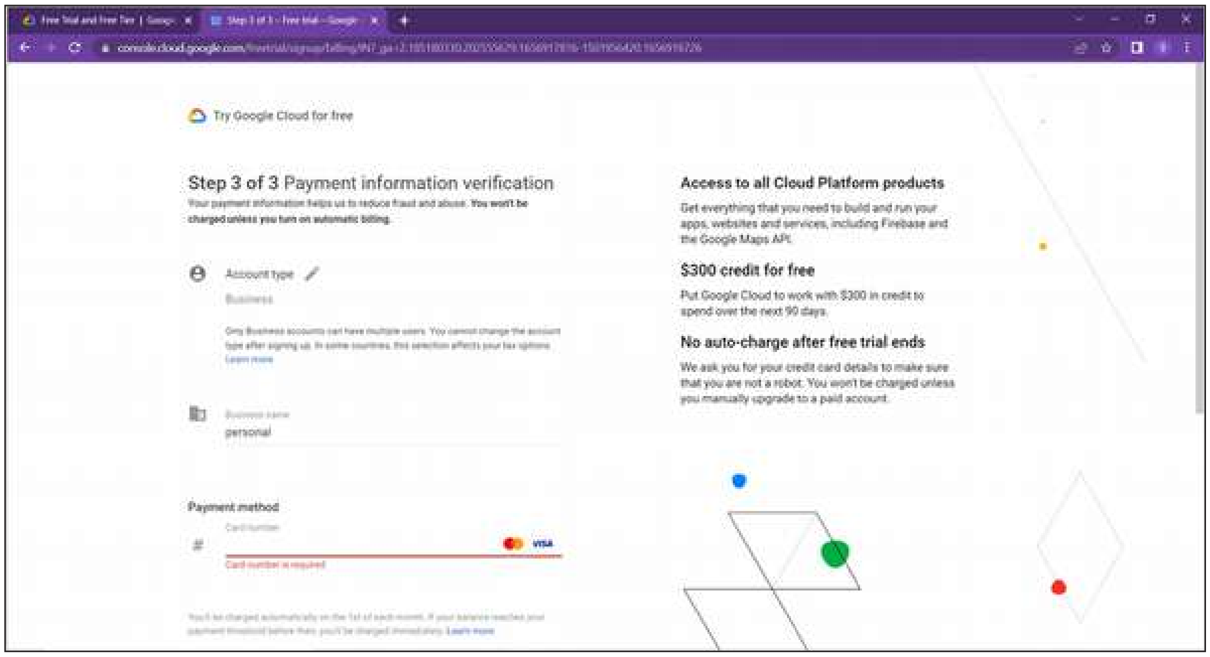
Put Google Cloud to work with $300 in credit to spend over the next 90 days.

No auto-charge after free trial ends

We ask you for your credit card details to make sure that you are not a robot. You won't be charged unless you manually upgrade to a paid account.

***Figure 1.5**: GCP account creation enter phone number*

7. Free accounts require a credit card. Verification costs Rs 2. Addresses must be provided. Click on **START MY FREE TRIAL** on this page:


Try Google Cloud for free

Step 3 of 3 Payment information verification

Your payment information helps us to reduce fraud and abuse. **You won't be charged unless you turn on automatic billing.**

Account type

Business

Only Business accounts can have multiple users. You cannot change the account type after signing up. In some countries, this selection affects your tax options. Learn more

Business name

personal

Payment method

Card number

Card number is required

You'll be charged automatically on the 1st of each month. If your balance reaches your payment threshold before then, you'll be charged immediately. Learn more

Access to all Cloud Platform products

Get everything that you need to build and run your apps, websites and services, including Firebase and the Google Maps API.

$300 credit for free

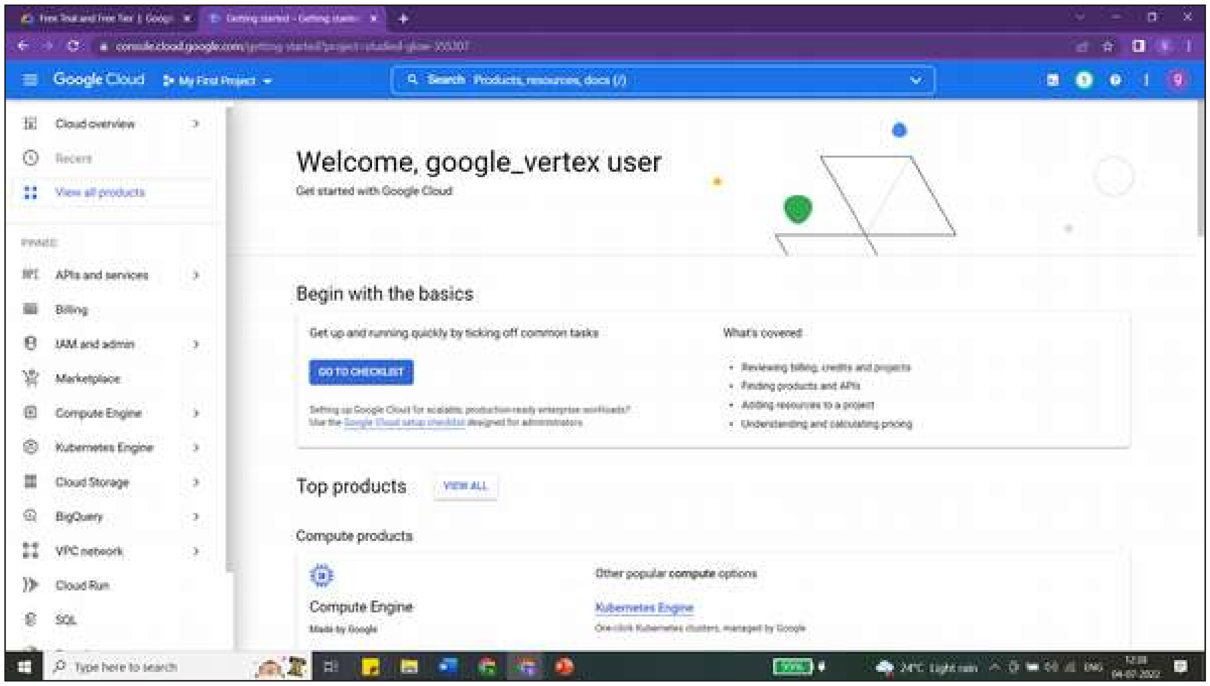
Put Google Cloud to work with $300 in credit to spend over the next 90 days.

No auto-charge after free trial ends

We ask you for your credit card details to make sure that you are not a robot. You won't be charged unless you manually upgrade to a paid account.

***Figure 1.6**: GCP account creation enter valid credit card details*

8. Users will land into this page once the free trail has started. The welcome page can be seen in *Figure 1.7*:



Welcome, google_vertex user

Get started with Google Cloud

Begin with the basics

Get up and running quickly by ticking off common tasks

Setting up Google Cloud for scalable, production-ready enterprise workloads? Use the Google Cloud setup checklist designed for administrators

What's covered

- Reviewing billing, credits and projects
- Finding products and APIs
- Adding resources to a project
- Understanding and calculating pricing

Top products

Compute products

Compute Engine

Made by Google

Other popular compute options

Kubernetes Engine

One-click Kubernetes clusters, managed by Google

Figure 1.7*: Landing page of GCP*

Footprint of Google Cloud Platform

Independent geographical areas are known as regions, while zones make up regions. Zones and regions are logical abstractions of the underlying physical resources that are offered in one or more datacenters physically located throughout the world. Within a region, the Google Cloud resources are deployed to specific locations referred to as zones. It is important that zones are seen as a single failure area within a region. *Figure 1.8* shows the footprint of GCP:

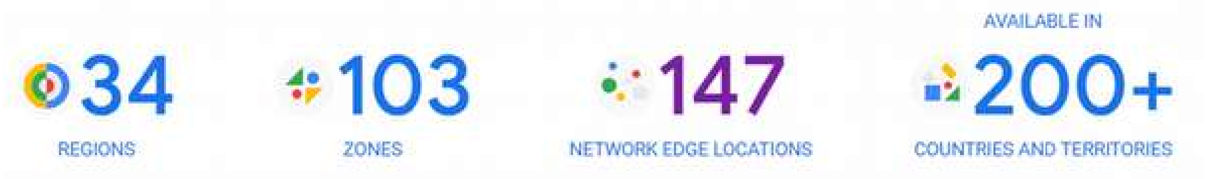


Figure 1.8*: Footprint of GCP*

The time this book was written there were about 34 regions, 103 zones and 147 network edge location across 200+ countries. GCP is constantly increasing its presence across the globe, please check the link mentioned below to get the latest numbers.

Image source: **https://cloud.google.com/about/locations**

The services and resources offered by Google Cloud may either be handled on a zonal or regional level, or they can be managed centrally by Google across various regions.:

- **Zonal resources**: The resources in a zone only work in that zone. When a zone goes down, some or all of the resources in that zone can be affected.
- **Regional resources**: They are spread across multiple zones in a region to make sure they are always available.
- **Multiregional resources**: Google manages a number of Google Cloud services to be redundant and spread both inside and between regions. These services improve resource efficiency, performance, and availability.
- **Global resources**: Any resource within the same project has access to global resources from any zone. There is no requirement to specify a scope when creating a global resource.

Network edge locations are helpful for hosting static material that is well-liked by the user base of the hosting service. The material is temporarily cached on these edge nodes, which enables users to get the information from a place that is much closer to where they are located. Users will have a more positive experience as a result of this.

There are few benefits associated with the GCP's regions and zones. When it comes to ensuring high availability, high redundancy, and high dependability, the notion of regions and zones is helpful. Obey the laws and regulations that have been established by the government. Data rules might vary greatly from one nation to the next.

Cloud Service Model

The cloud platform offers a variety of services, all of which may be roughly placed into one of three distinct categories:

- **Infrastructure as a service (IAAS)**
- **Platform as a service (PAAS)**
- **Software as a service (SAAS)**

The difference between cloud service models is illustrated in the *Figure 1.9*

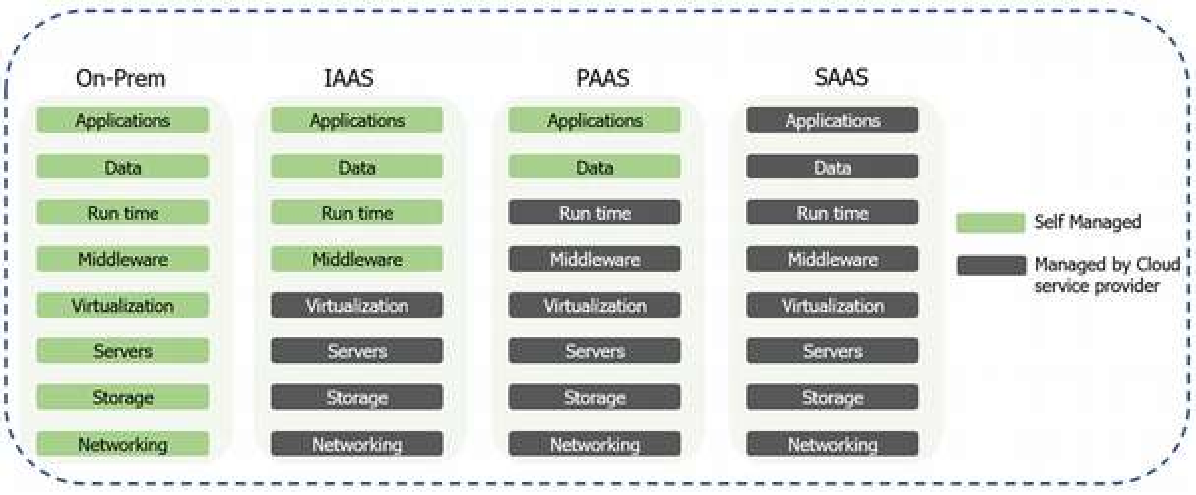


***Figure 1.9**: Cloud Service Model*

Let us imagine we are working on an application and hosting it at the same time on a server that is located on our premises. In this particular circumstance, it is our obligation to own and maintain the appropriate infrastructure, as well as the appropriate platforms, and of course, our application.

- **Infrastructure as a service**: In the case of IAAS, it will be the cloud's obligation to provide the necessary infrastructure, which may include virtual machines, networking, and storage devices. We are still responsible for ensuring that we have the appropriate platform for development and deployment. We have no other option for exercising control over the underlying infrastructure but to make use of it. One example of an infrastructure as a service provided by Google with its compute engine and Kubernetes engine.
- **Platform as a service**: In the case of PAAS, the responsibility for providing the appropriate platform for development and deployment, such as an operating system and tools for the environment in which programming languages are used, lies with the cloud service provider. This responsibility exists in addition to the infrastructure responsibility to provide such things. One example of a PAAS platform is Google App Engine.
- **Software as a service**: In the instance of SAAS, a cloud service provider will rent out to their customers apps that are theirs to run on their infrastructure. The maintenance of the software applications will also fall within the purview of the cloud service provider, in addition to the platform and the underlying infrastructure. These software programs are accessible to us on

whatever device we choose by way of web browsers, app browsers, and so on. Email (Gmail) and cloud storage (Google Drive) are two excellent instances of SAAS.

- **Data as a service (DAAS)**: DAAS is a service that is now starting to gain broad use, in contrast to the three service models that were mentioned before, which have been popular for more than a decade. This is partly owing to the fact that general cloud computing services were not originally built for the management of enormous data workloads; rather, they catered to the hosting of applications and basic data storage needs (as opposed to data integration, analytics, and processing).

SaaS eliminates the need to install and administer software on a local computer. Similarly, Data-as-a-Service methodology centers on the on-demand delivery of data from a number of sources using **application programming interfaces** (**APIs**). It is intended to make data access more straightforward, and provide curated datasets or streams of data that can be consumed in a variety of forms. These formats are often unified via the use of data virtualization. In fact, a DaaS architecture may consist of a wide variety of data management technologies such as data virtualization, data services, self-service analytics.

In its most basic form, DaaS enables organizations to have access to the ever-growing quantity and sophistication of the data sources at their disposal in order to give consumers with the most relevant insights. The democratization of data is something that is absolutely necessary for every company that wants to get actual value from its data. It gives a significant potential to monetize an organization's data and acquire a competitive edge by adopting a more data-centric approach to the operations and procedures.

Services offered by GCP

Users may make use of a comprehensive selection of services provided by Google Cloud Platform. Every one of the services may be placed into one of the categories that are shown in the *Figure 1.10*:

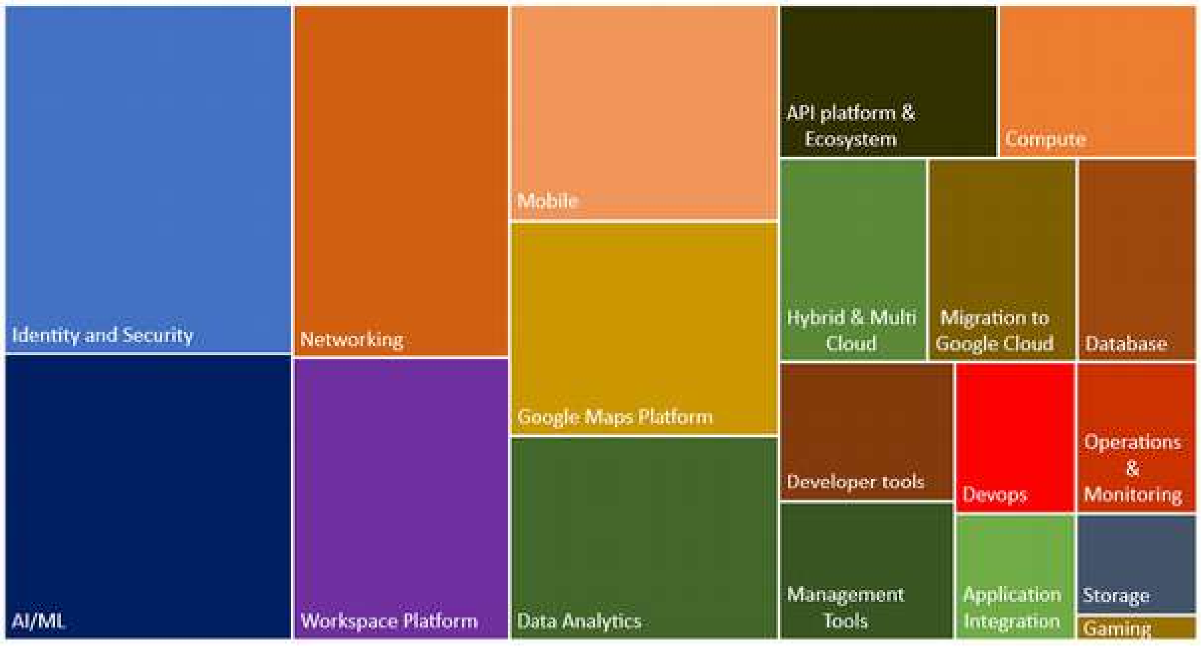


***Figure 1.10**: Services of GCP*

- Google offers Cloud Storage for storing unstructured items, Filestore for sharing files in the traditional manner, and persistent disk for virtual machines in the storage space. Compute Engine, App Engine, Cloud Run, Kubernetes Engine, and Cloud Functions are the core computing services that Google Cloud Platform provides.
- Cloud SQL, supports MySQL, PostgreSQL, and Microsoft SQL Server; and Cloud Spanner, which is a massively scalable database that is capable of running on a global scale. These are the relational database services that GCP provides.
- Bigtable, Firestore, Memorystore, and Firebase Realtime Database are different NoSQL services that Google provides. When dealing with massive amounts of analytical work, Bigtable is the most effective solution. Firestore is well suited for use in the construction of client-side web and mobile apps. Firebase Realtime Database is ideal for real-time data synchronization between users, such as is required for collaborative app development. Memorystore is a kind of datastore that operates entirely inside memory and is generally used to accelerate application performance by caching data that is frequently accessed.
- BigQuery is the name of the data warehouse service offered by Google.

- A **Virtual Private Cloud** (**VPC**) is an on-premises network on GCP. By using VPC Network Peering, virtual private clouds may be linked to one another. Users may utilize Cloud VPN, which operates over the internet, to establish a protected connection between a VPC and an on-premises network. Alternatively, users can establish a dedicated, private connection by using either Cloud Interconnect or Peering. To facilitate the migration of applications and data sets to its platform, the platform provides a diverse set of options. Offers Anthos as an alternative for hybrid cloud deployments.
- The field of data analytics is one in which Google excels in particular. Pub/Sub is used as a buffer for services that may not be able to deal with large surges in the amount of data coming in. Dataproc is a Hadoop and Spark implementation that is controlled by Dataproc. Apache Beam is the underlying technology that powers Dataflow, a managed implementation. You can do data processing using Dataprep even if you do not know how to write code, and it leverages Dataflow behind the scenes. Users may use google looker studio to visualize or show your data using graphs, charts, and other such graphical representations.
- Platform provides AI and ML services for a diverse group of customers. Vertex AI provides AUTOML option for the novices, for more experienced users, it provides trained models that make predictions via API and also provides various options for the advanced AI practitioners.
- Cloud Build enables you to develop continuous integration / continuous deployment pipelines. Private Git repositories that are hosted on GCP are known as Cloud Source Repositories. Artifact Registry expands on the capabilities of Container Registry and is the recommended container registry for Google Cloud. It provides a single location for storing and managing your language packages and Docker container images.
- **IAM** stands for **Identity and Access Management**, and it enables users and apps to have roles assigned to them. Everything you store in the GCP is by default encrypted. Companies now have the ability to control their encryption keys thanks to Cloud Key Management. Your API keys, passwords, certificates, and any other sensitive information may be safely stored in the Secret Manager.
- The Monitoring, Logging, Error Reporting, Trace, Debugger, and Profiler functions are all included in the Cloud Operations suite. The Active Security Threats and Vulnerabilities, as well as Compliance Infractions, are Presented to You by the Security Command Center. The development of Google Cloud Platform resources may be automated with the help of Cloud Deployment Manager.

Hierarchy of GCP

The term *resource* is used to describe anything that is put to use on Google Cloud Platform. Everything in the Google cloud has a clear hierarchy that resembles a parent-child connection. Hierarchy followed in Google Cloud Platform is as shown in *Figure 1.11*:

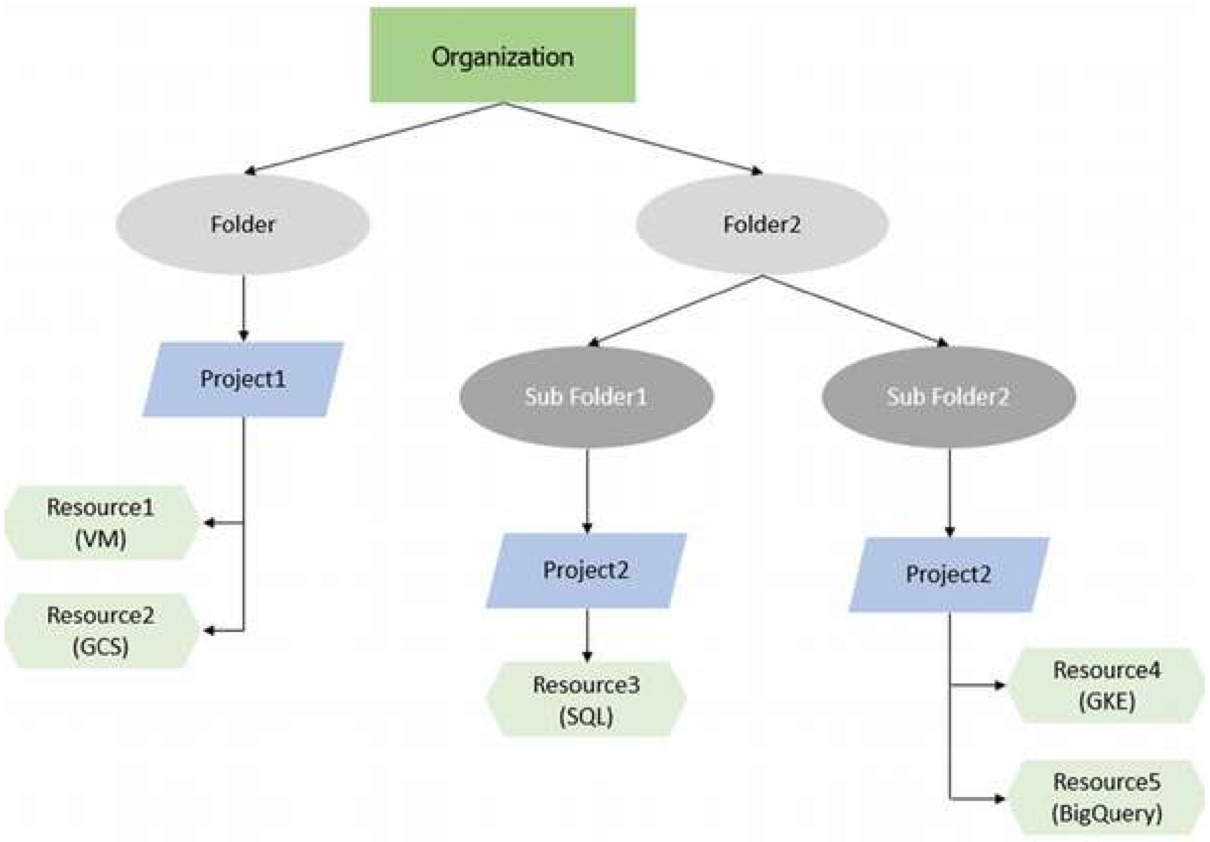


***Figure 1.11**: Hierarchy of GCP*

The Organization node serves as the starting point for the GCP Hierarchy and may stand for either an organization or a firm. The organization is the progenitor of both the folder and the project, as well as their respective resources. The rules for controlling access that have been implemented on the organization are applicable to all of the projects and resources that are affiliated with it.

But, if we establish an account with the personal mail ID as we did in the previous section, we would not be able to view the organization. On the other hand, if we login with our Google Workspace account and then start a project, the organization will be provided for us immediately. In addition, without organization, only a small number of the functions of the resource manager will be available.

Under organization we have folders. We are able to have an extra grouping mechanism at our disposal with the assistance of folders, and we may conceptualize this as a hierarchy of sub-organizations contained inside the larger organization. It is possible for a folder to have extra subfolders included inside it. You have the option of granting rights to access the project and all of its resources, either completely or partially, depending on the folder in question.

A project is an entity that exists at the most fundamental level. It is possible to have many projects nested inside of organization's and folders. The project is absolutely necessary in order to make use of GCP resources, and it serves as the foundation for making use of cloud services, managing APIs, and enabling billing. A project has two different IDs connected with it. The first of these is the project ID, which is a one-of-a-kind identification for the project. And the second one is the project number, which is automatically issued whenever a project is created, and we are unable to modify it in any way.

The term *resources* refers to the components that make up Google Cloud Platform. Resources include things like cloud storage, databases, virtual machines, and so on. Each time we establish a cloud storage bucket or deploy a virtual machine, we link those resources to the appropriate project.

New Project Creation

Google Cloud projects serve as the foundation for the creation, activation, and use of all Google Cloud services, such as managing APIs, enabling invoicing, adding and deleting collaborators, and managing permissions for Google Cloud resources.

Project can be created by following the steps in web console:

1. Navigation from **IAM and admin | Manage resources**. The sequence can be seen in *Figure 1.12*:

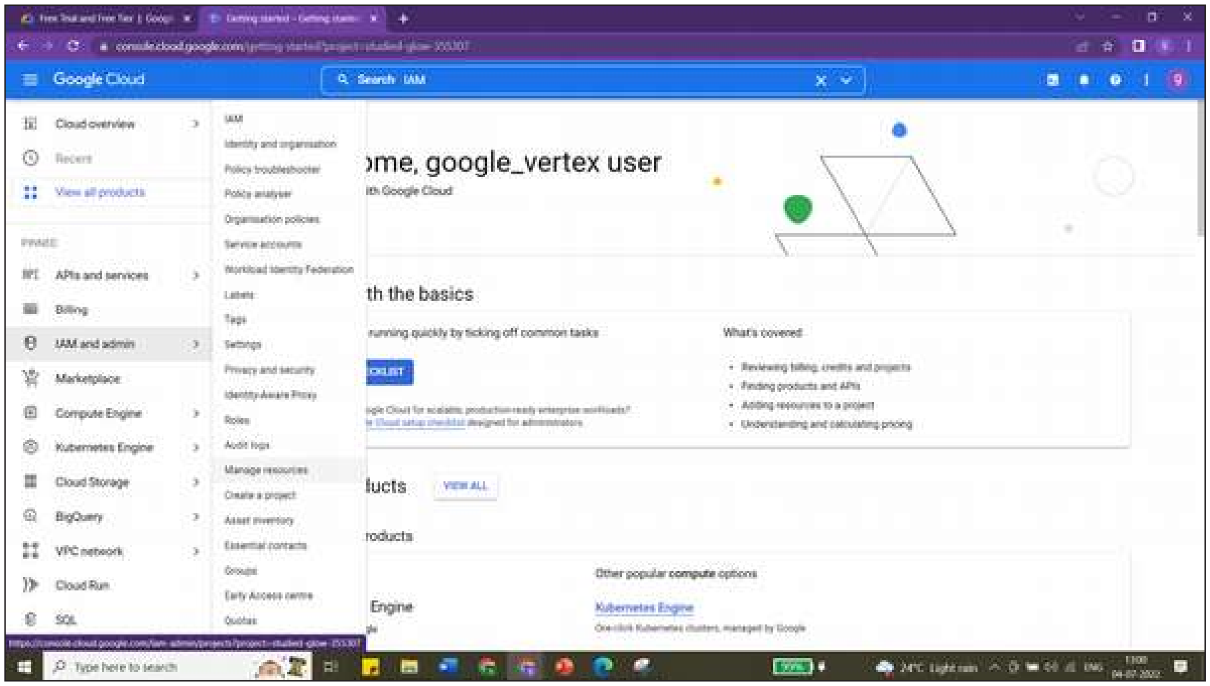


***Figure 1.12**: Project Creation*

2. Click on **CREATE PROJECT**.

   The project creation can be seen in the following figure:

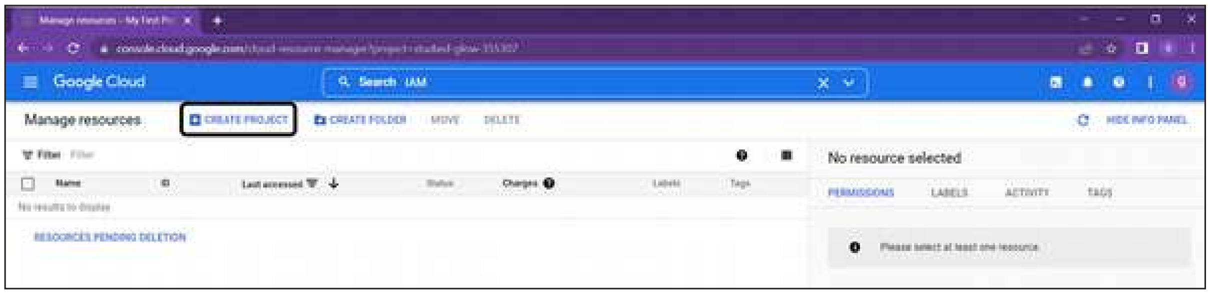


***Figure 1.13**: Project Creation*

3. Users to provide name for the project, follow the steps as shown in *Figure 1.14*:

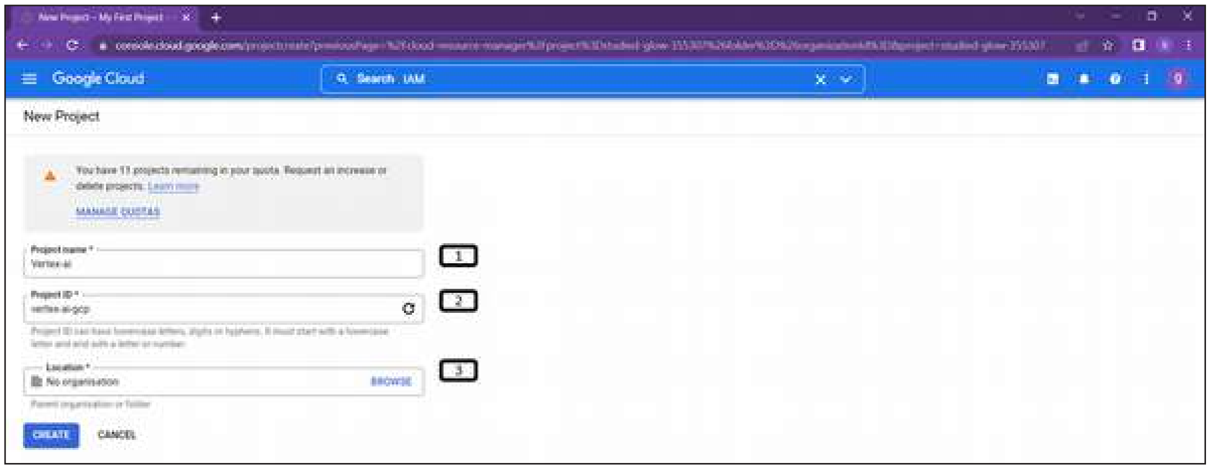

New Project – My First Project
Google Cloud
Search IAM
New Project
You have 11 projects remaining in your quota. Request an increase or delete projects. Learn more
MANAGE QUOTAS
Project name *
Vertex-ai
Project ID *
vertex-ai-gcp
Project ID can have lowercase letters, digits or hyphens. It must start with a lowercase letter and end with a letter or number.
Location *
No organisation
BROWSE
Parent organisation or folder
CREATE
CANCEL
1
2
3

Figure 1.14: *Project Creation*

1. Provide **Project name**.
2. **Project ID** will be automatically populated, users can edit it while creation of the project. If users need to access the resources under Project through SDK or APIs project-ID is needed. Once project is created users cannot change the Project-ID
3. If users are creating a project under **Organization**, select the organization. Users with free account cannot create organization or folder. All the projects will be created under **No Organization**.

NOTE: Users who are accessing through free account will be given limited amount of project creation.

Deletion of Project

To delete any project that is active:

1. Select the project that needs to be deleted.
2. Click on **DELETE**, users will be prompted for confirmation.

This can be seen illustrated in Figure 1.15:

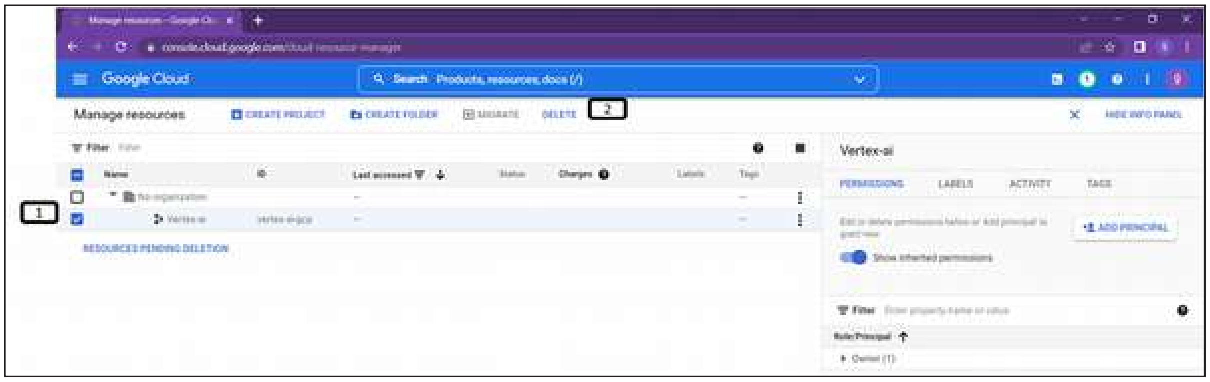


***Figure 1.15**: Project deletion*

Once the users will confirm the deletion of project, it will be marked for deletion and will be in same state for 30 days. Users can restore the project within a period of 30 days, post that project and the resources associated under that project will be deleted permanently and cannot be recovered back. Also, project which is marked under deletion is not usable.

Interacting with GCP services

When we talk about resources, let us discuss how we can work with them. GCP offers three basic ways to interact with the services and resources:

Google Cloud Platform Console

When working with Google Cloud, the Console or Web User Interface is the most common method of communication. At the same time, it delivers a wealth of functionality and an easy-to-use interface for the users who are just getting started with GCP.

Cloud console can be accessed with the link **https://console.cloud.google.com**.

Landing page of the google cloud console is as shown in *Figure 1.16*:

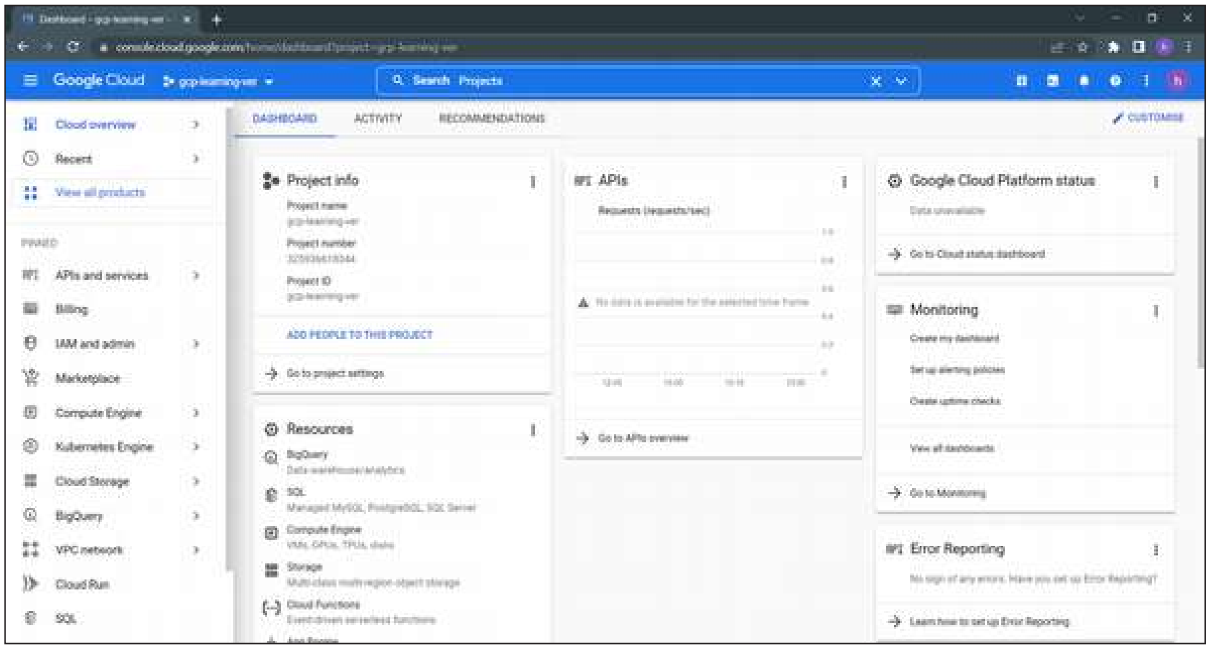


Figure 1.16: *GCP Console*

Command Line Interface

You may control your development process and your GCP resources using the gcloud command-line tool provided by the Google Cloud SDK, if you are a developer. To get command-line access to Google Cloud's computational resources, you may use the Cloud Shell. With a 5-GB home directory, Cloud Shell is a Debian-based virtual computer from which you can easily manage your GCP projects and resources. Cloud Shell is pre-installed with the gcloud command-line tool and other necessary tools, allowing you to get up and running fast. To use cloud shell, follow the steps below:

Activate Cloud shell as shown in *Figure 1.17*:

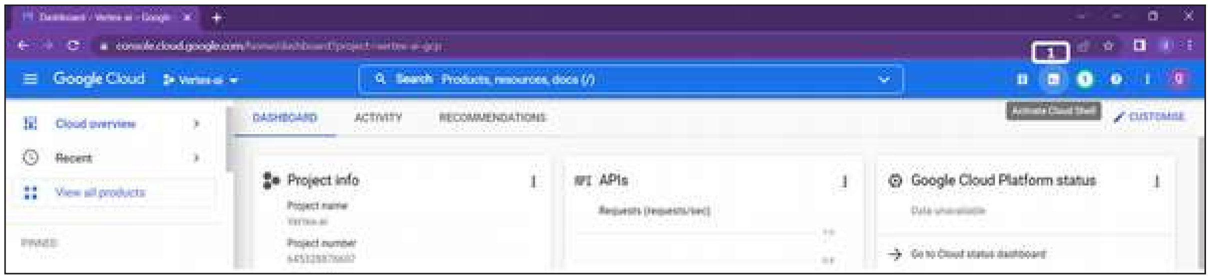
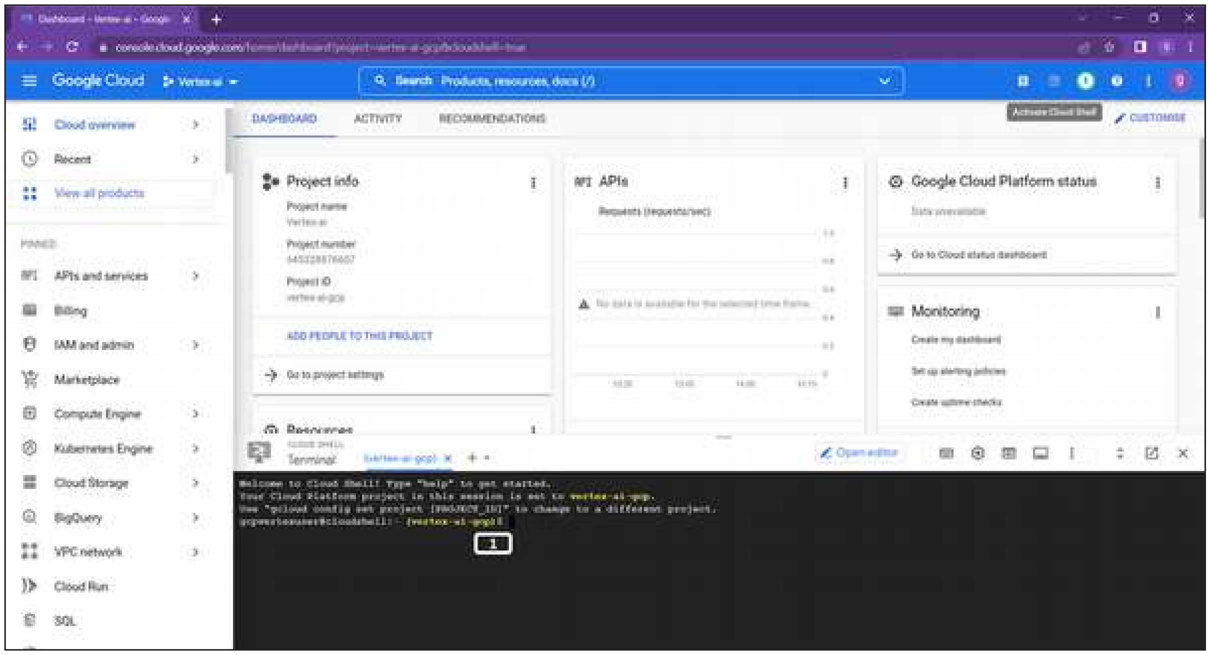


Figure 1.17: *GCP CLI*

Click on **Activate cloud shell**.It will take few mins to open the Cloud shell command window

Once the cloud shell is activated, black space appears at the bottom of the screen to type commands as shown in *Figure 1.18*:



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to vertex-ai-gcp.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
gcpvertexuser@cloudshell:~ (vertex-ai-gcp)$
```

***Figure 1.18**: GCP CLI*

1. Type commands:

   `gcloud projects create project_id - For Project creation`

   `gcloud projects delete project_id - For project deletion`

APIs

It is common for apps to communicate with Google Cloud via **Software Development Kit** (**SDK**). Go, Python, and node.js are few of the many programming languages for which Google Cloud SDKs are available.

NOTE: We will use this method while getting predictions from the deployed models.

Storage

Along with computing and network, storage is considered to be one of the fundamental building components. Applications benefit from storage services' increased levels of persistence and durability. These services are located deep inside the platform and serve as the foundation for the vast majority of Google Cloud's services as well as the systems that you construct on top of it. They are the platform's central pillars.

Three types of storage options are provided by Google Cloud:

- Persistent Disks
- Filestore
- **Cloud Storage** (**GCS**)

They are explained as follows:

- **Persistent Disks**: Block storage is provided by a Google Cloud Persistent Disk, which is used by virtual machine hosted on Google Cloud (Google Cloud Compute Engine). Imagine those Persistent Disks as simple USB sticks; this will help you comprehend it far better than any other method. They may be connected to virtual machines or detached from them. They allow you to construct data persistence for your services whether virtual machines are started, paused, or terminated. These Persistent Disks are put to use to power not just the virtual machines that are hosted on Google Cloud Compute Engine, but also the Google Kubernetes Engine service.

  A Google Cloud Persistent Disk operates similarly to a virtual disc on your local PC. Persistent Disk can either be HDD or SSD, with the latter offering superior I/O performance. In addition, there is the choice of where they are placed as well as the sort of availability that is required, which may be either regional, zonal, or local.

  Other capabilities of Google Cloud Persistent Disks that are lesser known but are prove useful include automatic encryption, the ability to resize the disc while it is being used, and a snapshot capability that can be used for both backing up data and creating images for virtual machines. Read and write access can be configured for multiple VMs. One VM can have write access and all other VMs can have read access for a Persistent disk.

- **Filestore**: Filestore is a network file storage service that is provided by Google Cloud. The idea of network file storage has been around for quite some time, and similar to block storage, it can also be found in the on-premises data centers that most businesses use. You should be comfortable with the notion if you are used to dealing with NAS, which stands for network-attached storage. In response to the dearth of services that are compatible with **network-attached storage** (**NAS**), Google has expanded its offerings to include a cloud file storage service.

  The primary distinction between Persistent disk and network file storage, as the name suggests, provides a disk storage over the network. This makes it possible to create systems with many parallel services that are able to read and write files from the same disc storage that is mounted across the network. These systems may be developed with the help of this capability.

  The following are some examples of uses for Filestore:

  - o Majority of the on-premises applications needs a file system interface, Filestore makes it easy to migrate these kind of enterprise applications to the cloud
  - o It is used in the rendering of media in order to decrease latency.

- **Cloud Storage**: The service for storing objects that is provided by Google Cloud is known as Google Cloud Storage. It offers a number of extremely intriguing features that are pre-installed, such as object versioning and fine-grained permissions (either by object or bucket), both of which have the potential to simplify the development process and cut down on operational overhead. The Google Cloud Storage platform is used as the basis for a variety of different services.

  Having this kind of storage is not at all usual in ordinary on-premises systems, which often have a capacity that is more restricted and connection that is both quick and exclusive. Object storage, on the other hand, has a very user-friendly interface in terms of how it works. In layman's words, its value proposition is such that you are able to acquire and put whatever file you want using a REST API; in addition, this may extend forever with each object expanding up to the terabyte scale; and last, its value proposition is such that it is possible to store any amount of data. Buckets are the *namespaces* that are used in Cloud Storage to organize the many items that are stored there. Even while a bucket has the capacity to carry a number of items, each individual item will only ever belong to a single bucket.

The inexpensive cost of this storage type (cents per GB), along with its serverless approach and its ease of use, has contributed to its widespread adoption in cloud-native system architectures. The cloud service provider is then responsible for handling the laborious tasks of data replication, availability, integrity checks, capacity planning, and so on. APIs make it possible for applications to both save and retrieve items.

Based on factors like cost, availability, and frequency of access, cloud storage has four different storage classes. They are **Standard**, **Nearline**, **Coldline**, and **Archive** as shown in *Figure 1.19*:

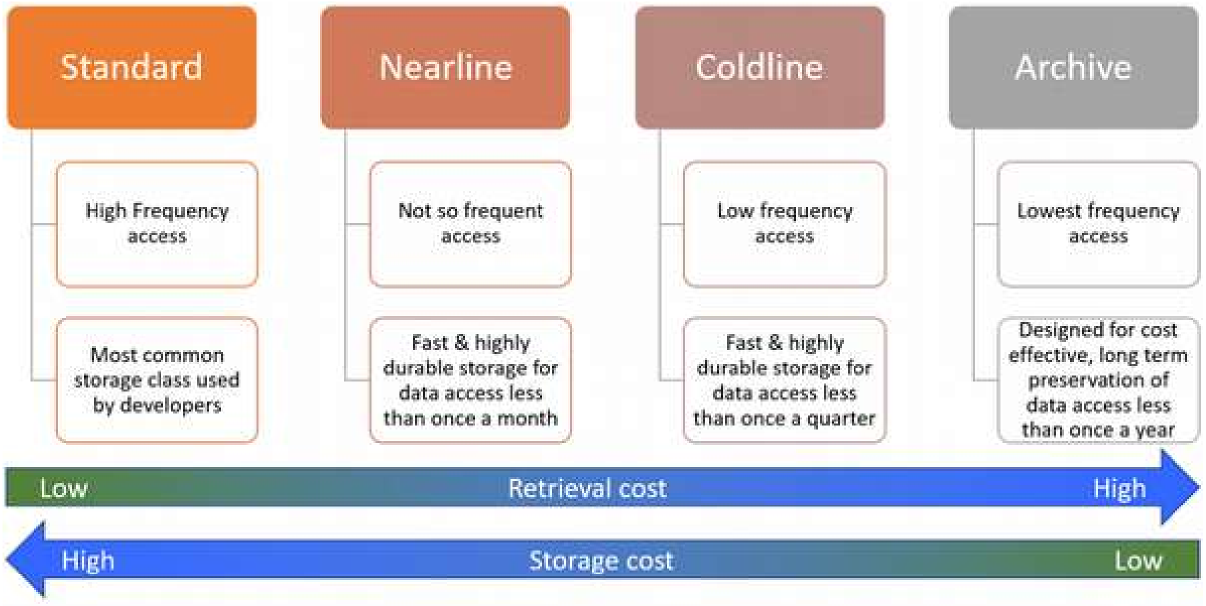

Standard
High Frequency access
Most common storage class used by developers
Nearline
Not so frequent access
Fast & highly durable storage for data access less than once a month
Coldline
Low frequency access
Fast & highly durable storage for data access less than once a quarter
Archive
Lowest frequency access
Designed for cost effective, long term preservation of data access less than once a year
Low
Retrieval cost
High
High
Storage cost
Low

***Figure 1.19**: GCP Storage*

- **Standard class**: This class of storage allows for high frequency access and is the type of storage that is most often used by software developers.
- **Nearline storage class**: This class is used for data that are not accessed very regularly, generally for data that are not accessed more than once a month. Generally speaking, nearline storage is used for data.
- **Lowline storage class**: This class is used for records that are normally accessed not more often than once every three months.
- **Archive storage class**: This class is used for data that is accessed with the lowest frequency and is often used for the long-term preservation of data.

Working with Google Cloud Storage

Majority of the exercises covered in this book inputs the data from the cloud storage, so let us see how we can create cloud storage bucket and upload files to the bucket.

Step 1: Open cloud storage:

Follow the steps described in the *Figure 1.20* to open the cloud storage:

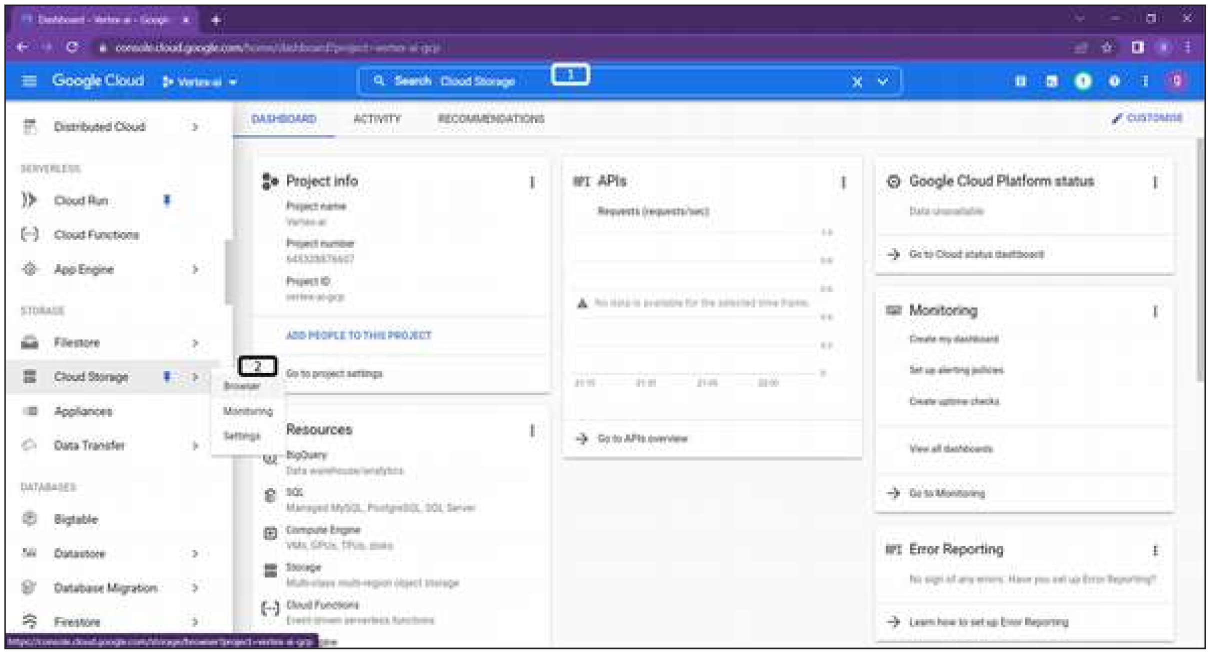


***Figure 1.20**: Cloud Storage Creation*

1. Users can type cloud storage in the search box.
2. Alternatively, users can navigate under **STORAGE to Cloud Storage | Browser**.

Step 2: Bucket creation:

Users can upload files or folders into cloud storage after creating a bucket. Initiate the bucket creation process as shown in *Figure 1.21*:

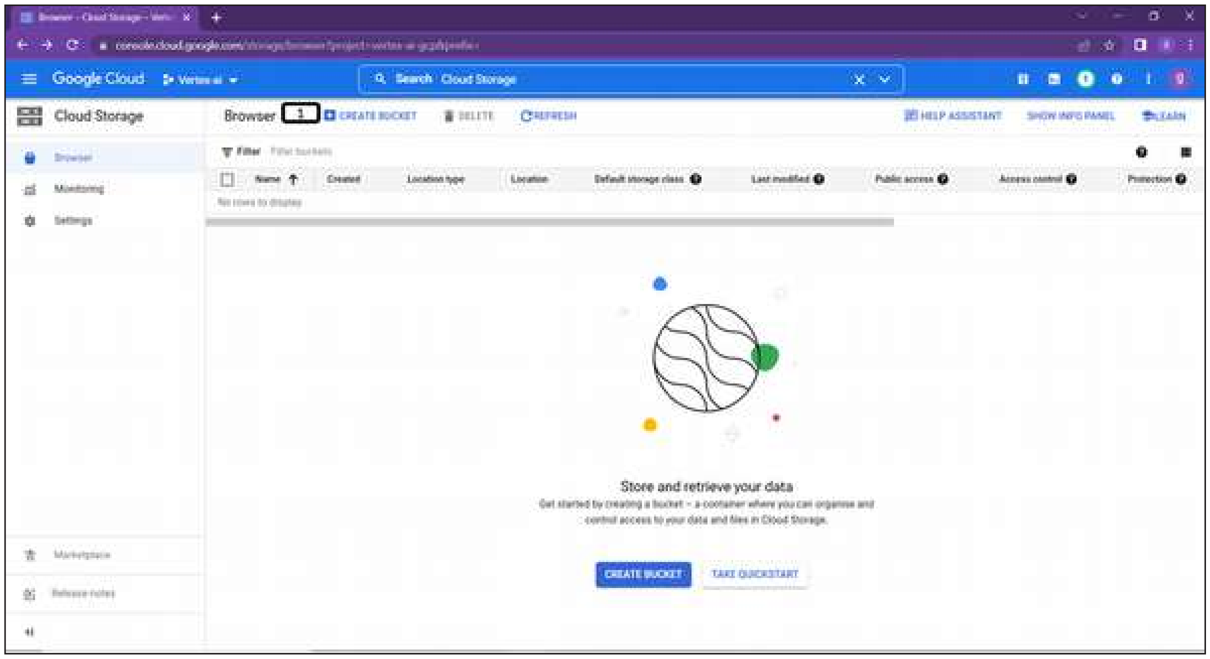


Figure 1.21: New bucket creation

1. Click on **CREATE BUCKET**.

Step 3: Provide name for the bucket:

Follow steps shown in *Figure 1.22* to add names, labels to bucket:

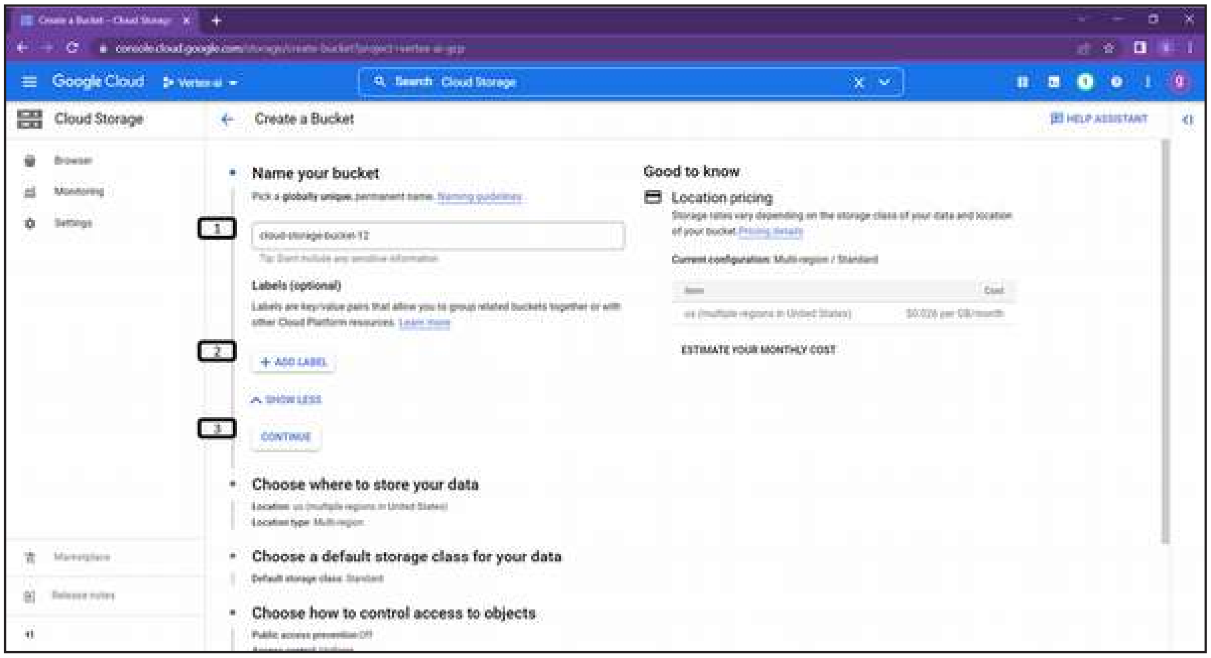
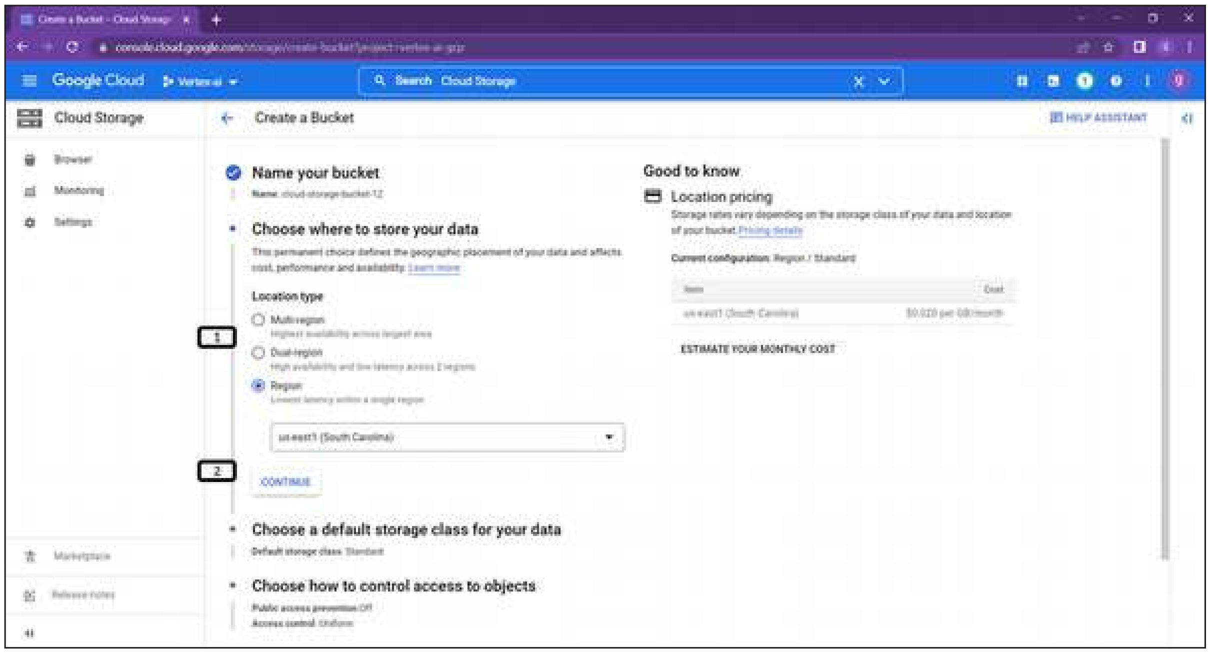

Google Cloud
Vertex ai
Search Cloud Storage
Cloud Storage
Create a Bucket
HELP ASSISTANT
Browser
Monitoring
Settings
Name your bucket
Pick a globally unique, permanent name. Naming guidelines
1
cloud-storage-bucket-12
Labels (optional)
Labels are key/value pairs that allow you to group related buckets together or with other Cloud Platform resources. Learn more
2
+ ADD LABEL
SHOW LESS
3
CONTINUE
Choose where to store your data
Location: us (multiple regions in United States)
Location type: Multi-region
Choose a default storage class for your data
Default storage class: Standard
Choose how to control access to objects
Public access prevention Off
Good to know
Location pricing
Storage rates vary depending on the storage class of your data and location of your bucket. Pricing details
Current configuration: Multi-region / Standard
Item
Cost
us (multiple regions in United States)
$0.026 per GB/month
ESTIMATE YOUR MONTHLY COST
Marketplace
Release notes

Figure 1.22: Bucket name

1. Users need to provide **globally unique** name for the bucket.
2. Labels are optional, it provides key/value pairs to group buckets (or even services) together.
3. Click on **Continue**.

Step 4: Choosing location for bucket:

Follow steps in *Figure 1.23* to choose the location of the bucket:


Create a Bucket

Name your bucket
Name: cloud-storage-bucket-12

Choose where to store your data
This permanent choice defines the geographic placement of your data and affects cost, performance and availability. Learn more

Location type
- Multi-region — Highest availability across largest area
- Dual-region — High availability and low latency across 2 regions
- [✓ Region] — Lowest latency within a single region

us-east1 (South Carolina)

CONTINUE

Choose a default storage class for your data
Default storage class: Standard

Choose how to control access to objects
Public access prevention: Off
Access control: Uniform

Good to know
Location pricing
Storage rates vary depending on the storage class of your data and location of your bucket. Pricing details
Current configuration: Region / Standard

Item	Cost
us-east1 (South Carolina)	$0.020 per GB/month

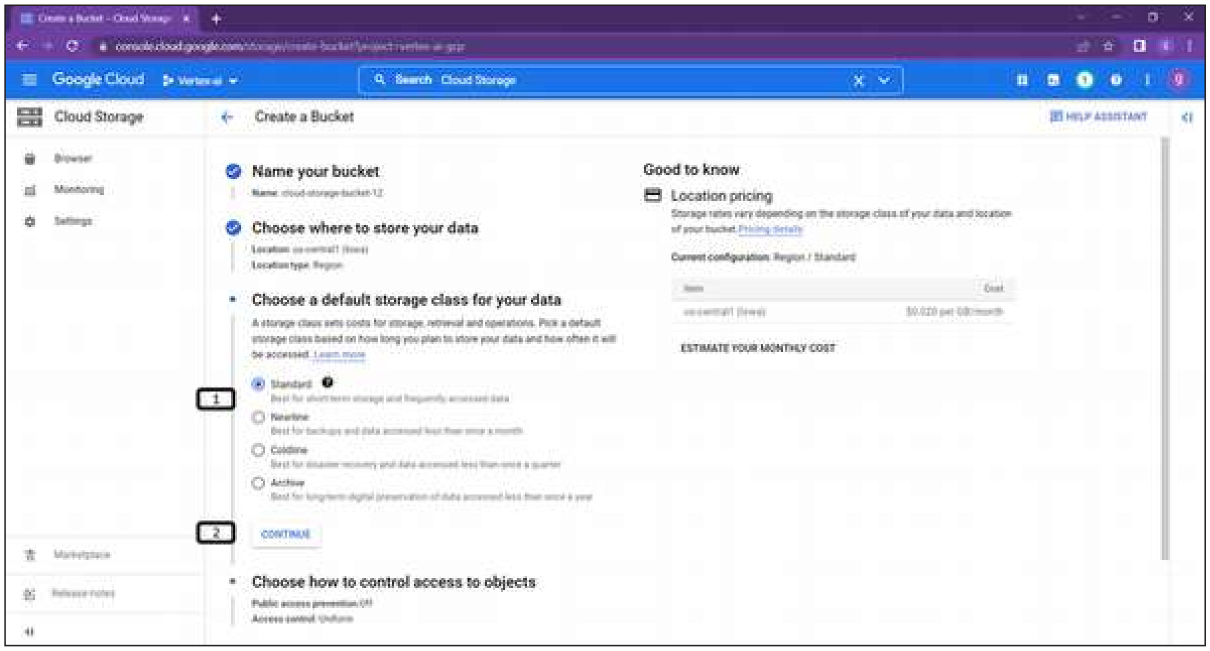
ESTIMATE YOUR MONTHLY COST

Figure 1.23: Location for the bucket

1. Users can select location type. **Multi-region** gives options to choose multiple regions of America or Europe or Asia. **Dual-region** provides options to choose two regions belonging to same continent (America, Europe and Asia-pacific). **Region** provides options to choose one region from the drop-down list. For majority of the exercises which this book covers we will have the data uploaded to bucket belonging to single region.
2. Click **Continue**.

Step 5: Selecting storage class

Follow the steps shown in *Figure 1.24* to select the class of storage:


Create a Bucket

Name your bucket
Name: cloud-storage-bucket-12

Choose where to store your data
Location: us-central1 (Iowa)
Location type: Region

Choose a default storage class for your data
A storage class sets costs for storage, retrieval and operations. Pick a default storage class based on how long you plan to store your data and how often it will be accessed. Learn more

- [✓ Standard] Best for short-term storage and frequently accessed data
- Nearline: Best for backups and data accessed less than once a month
- Coldline: Best for disaster recovery and data accessed less than once a quarter
- Archive: Best for long-term digital preservation of data accessed less than once a year

CONTINUE

Choose how to control access to objects
Public access prevention: Off
Access control: Uniform

Good to know
Location pricing
Storage rates vary depending on the storage class of your data and location of your bucket. Pricing details

Current configuration: Region / Standard

Item	Cost
us-central1 (Iowa)	$0.020 per GB/month

ESTIMATE YOUR MONTHLY COST

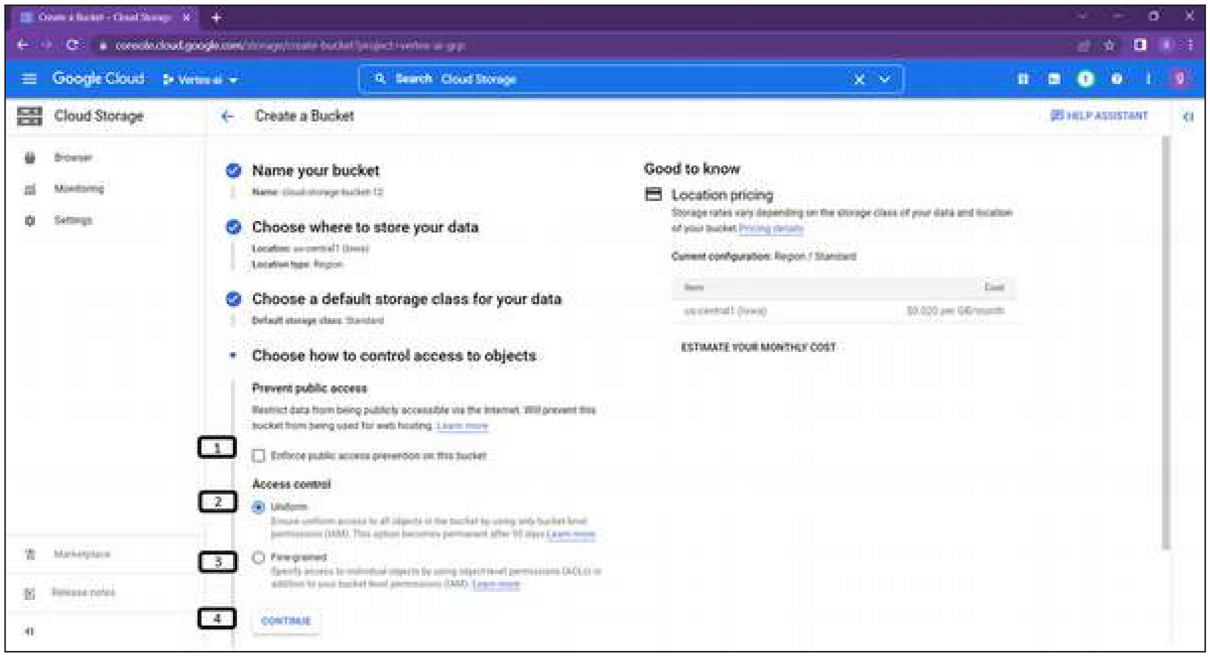
***Figure 1.24**: Storage class for the bucket*

1. Choose **Standard** (Exercises which this book covers we will create standard bucket).
2. Click on **Continue**.

NOTE: Try providing different options with location and storage class and observe the variation in the price estimates.

Step 6: Access control for buckets:

Follow the steps described in the *Figure 1.25* to configure access control for the bucket:



Create a Bucket

Name your bucket
Name: cloud-storage-bucket-12

Choose where to store your data
Location: us-central1 (Iowa)
Location type: Region

Choose a default storage class for your data
Default storage class: Standard

Choose how to control access to objects

Prevent public access
Restrict data from being publicly accessible via the internet. Will prevent this bucket from being used for web hosting. Learn more

☐ Enforce public access prevention on this bucket

Access control
● Uniform
Ensure uniform access to all objects in the bucket by using only bucket-level permissions (IAM). This option becomes permanent after 90 days Learn more
○ Fine-grained
Specify access to individual objects by using object-level permissions (ACLs) in addition to your bucket-level permissions (IAM). Learn more

CONTINUE

Good to know
Location pricing
Storage rates vary depending on the storage class of your data and location of your bucket. Pricing details

Current configuration: Region / Standard

Item	Cost
us-central1 (Iowa)	$0.020 per GB/month

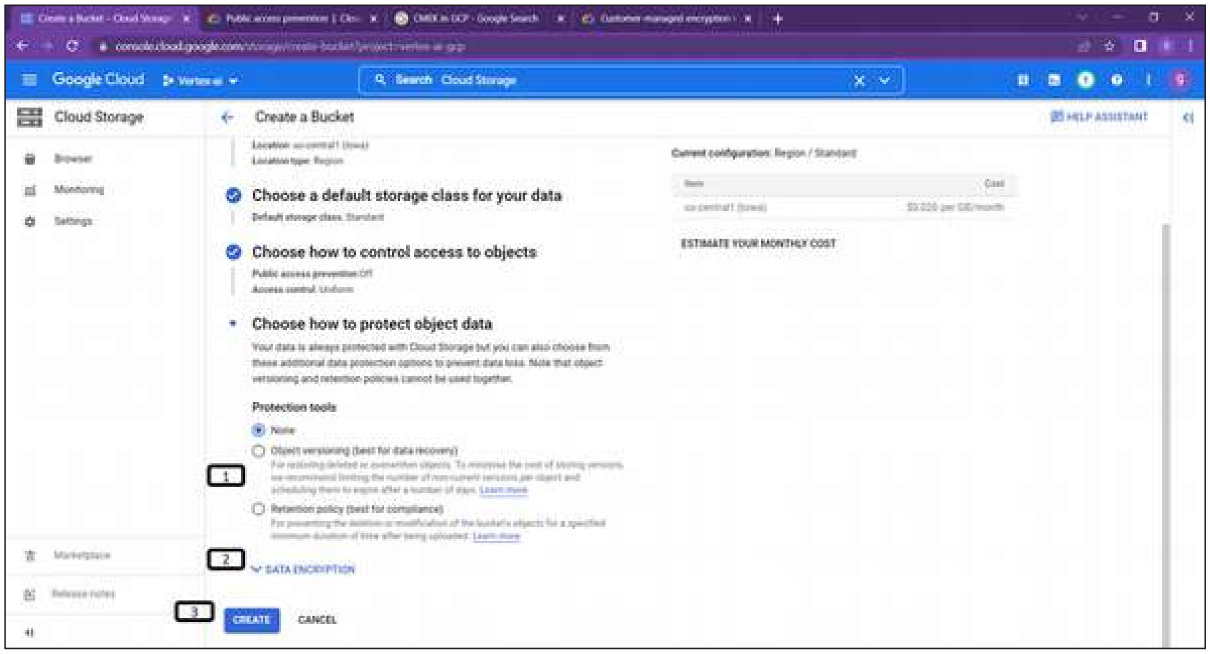
ESTIMATE YOUR MONTHLY COST

Figure 1.25: Access control for the bucket

1. Selection of the box prevents public access from the internet, if this option is chosen it will not be possible to provide public access through IAM policies.
2. Uniform access controls apply same IAM policies for all the folders and files belonging to the bucket (select **Uniform**)
3. Opposite to **Uniform** access controls, **Fine-grained** specifies policies to individual files and folders.
4. Click **Continue**.

Step 7: Data protection in bucket:

Follow the steps described in *Figure 1.26* for data protection:



Create a Bucket

Location: us-central1 (Iowa)
Location type: Region

Choose a default storage class for your data
Default storage class: Standard

Choose how to control access to objects
Public access prevention: Off
Access control: Uniform

Choose how to protect object data
Your data is always protected with Cloud Storage but you can also choose from these additional data protection options to prevent data loss. Note that object versioning and retention policies cannot be used together.

Protection tools
- (●) None
- ( ) Object versioning (best for data recovery)
  For restoring deleted or overwritten objects. To minimize the cost of storing versions, we recommend limiting the number of non-current versions per object and scheduling them to expire after a number of days. Learn more
- ( ) Retention policy (best for compliance)
  For preventing the deletion or modification of the bucket's objects for a specified minimum duration of time after being uploaded. Learn more

Current configuration: Region / Standard

Item	Cost
us-central1 (Iowa)	$0.020 per GB/month

Figure 1.26: Data protection for bucket

1. GCP provides additional options for data protection, object versioning helps users for data recovery and retention policy helps for compliance (data cannot be deleted for a minimum period of time once it is uploaded).
2. All the data uploaded to GCP will be encrypted by google managed encryption key, if users can choose **customer managed encryption keys (CMEK)** for more control.
3. Click on **Create**.

Step 8: Uploading files to the bucket:

Follow the steps described in *Figure 1.27* to upload files to cloud storage:

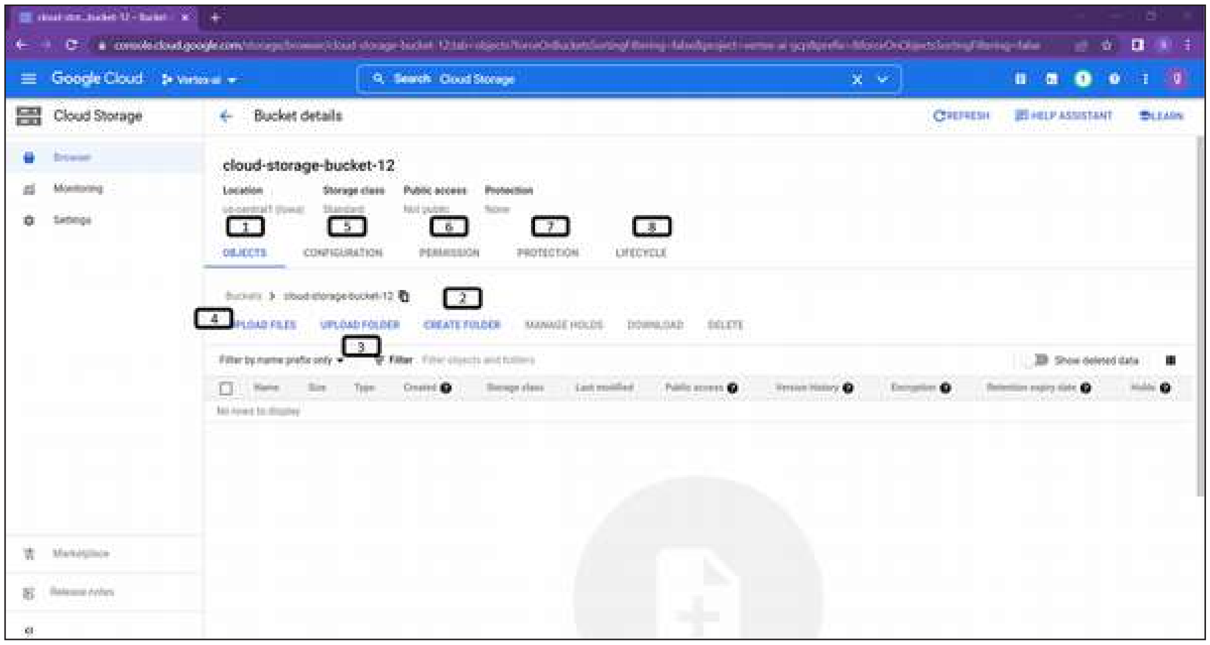
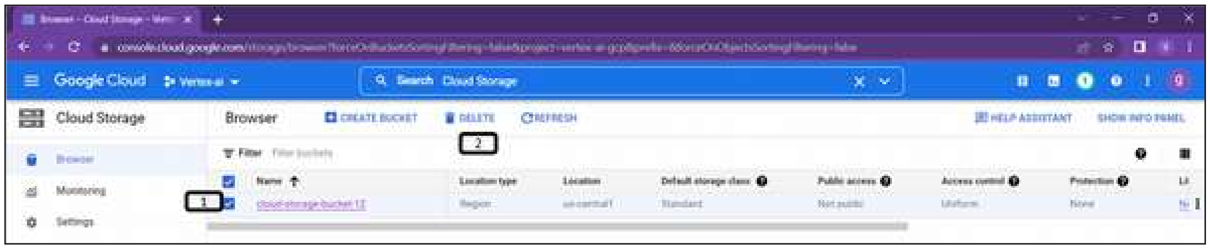


***Figure 1.27**: File upload to bucket*

1. **OBJECTS** provide users options to upload the data.
2. **CREATE FOLDER** inside the bucket.
3. Directly **UPLOAD FOLDER** from the local system.
4. Upload files.
5. All the options which users have chosen during the process of bucket creation will be listed under configuration.
6. **PERMISSIONS** provides options to enable prevent public access, or change the uniform access/fine-grained access. Also, it provides options to add users for accessing the bucket.
7. **PROTECTION** tab provides options to enable or disable object versioning and retention policy.
8. When specific criteria are satisfied, **LIFECYCLE** rules allow you to perform operations on the items in a bucket.

Deletion of bucket

Follow steps described in *Figure 1.28* to delete the bucket:



Name	Location type	Location	Default storage class	Public access	Access control	Protection
cloud-storage-bucket-12	Region	us-central1	Standard	Not public	Uniform	None

***Figure 1.28**: Bucket deletion*

1. Select the bucket which needs to be deleted.
2. Click on **DELETE**, you will be prompted with a pop-up where user need to type delete.

Compute

The provision of compute services is an essential component of any cloud services offering. One of the most important products that Google provides to companies who want to host their applications on the cloud is called Google Compute Services. The umbrella term "computing" covers a wide variety of specialized services, configuration choices, and products that are all accessible for your consumption.

Users have access to the computing and hosting facilities provided by Google Cloud, where they may choose from the following available options:

- **Compute Engine**: Users may construct and utilize virtual machines in the cloud as a server resource via the usage of Google Compute Engine, which is an **IAAS** platform. Users do not need to acquire or manage physical server hardware. Virtual Machines are servers that are hosted in the cloud, and Google's data centers will be the locations where they are operated. They provide configurable choices for the **central processing unit** (**CPU**) and **graphics processing unit** (**GPU**), memory, and storage, in addition to several options for operating systems. Both a **command line interface** (**CLI**) and a web console are available for use when accessing virtual machines. When the compute engine is decommissioned, all the data will be erased. Therefore, the persistence of data may be ensured by using either traditional or solid-state drives.

  The Hypervisor is what runs Virtual Machines. The hypervisor is a piece of software that is installed on host operating systems and is responsible for the creation and operation of many virtual machines. The **hypervisor** on the host computer makes it possible to run many virtual machines by allowing them to share the host's resources. *Figure* 1.29 shows the relationship between VMs, hypervisor and physical infrastructure.

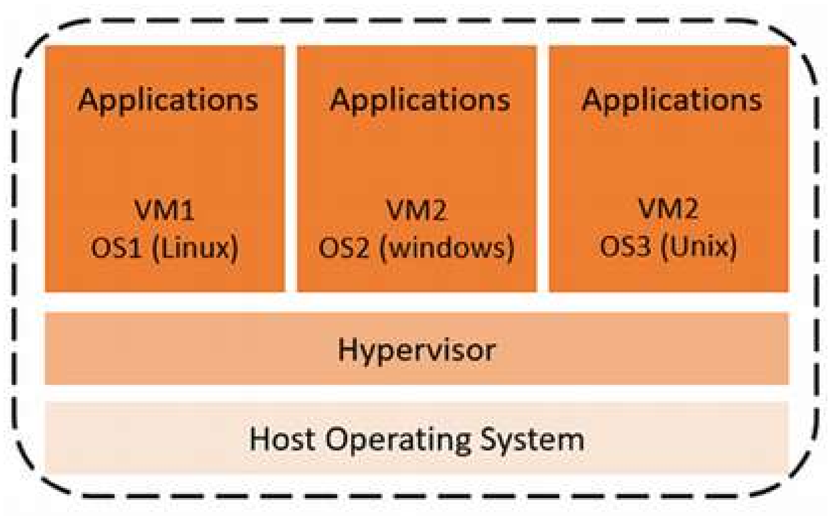


***Figure 1.29**: Compute Engine*

When it comes to pricing, stability, backups, scalability, and security, using Google Compute Engine is a good choice. It is a cost-efficient solution since consumers only need to pay for the amount of time that they utilize the resources. It allows live migration of VMs from one host to another, which helps to assure the system's reliability. In addition to that, it has a backup mechanism that is reliable, built-in, and redundant. It does this by making reservations in order to assist and guarantee that applications have the capacity they need as they expand. Additional security will be provided by the compute engine for the applications running on it.

Compute Engine is a good solution for migrating established systems or fine-grained management of the operating system or other operational features.

- **Kubernetes Engine**: The **Google Kubernetes Engine**, sometimes known as **GKE**, is a container as a service of GCP. When installing containerized apps, clusters are managed via the open-source Kubernetes cluster management system. Kubernetes makes it possible for users to communicate with container clusters by providing the necessary capabilities. Both a control plane and a worker node are components of the Kubernetes system. Worker nodes are supplied to function as compute engines.

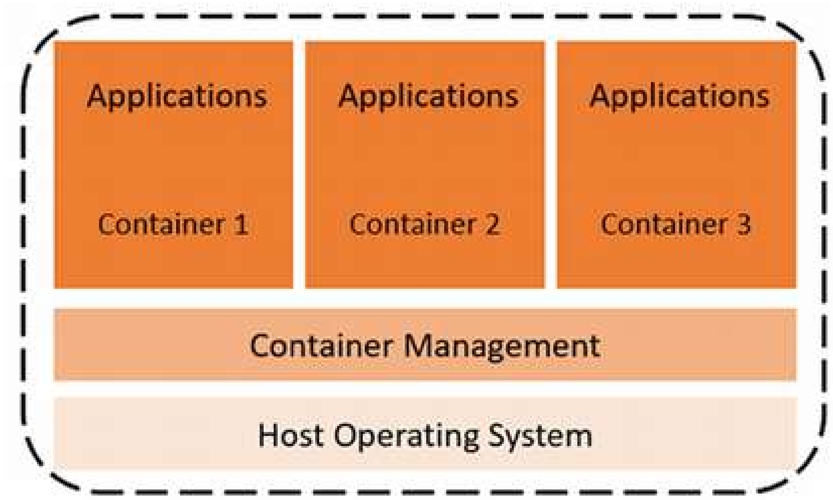


***Figure 1.30**: Kubernetes Engine*

VM operates on Guest operating systems and hypervisors, however with Kubernetes Engine, compute resources are segregated in containers, and both the **container manager** and the **Host operating system** are responsible for their management.

Unique features include auto upgrades, node auto maintenance and auto scaling (Additional resources will be allotted by Kubernetes Engine in the event that the strain is placed on the applications).

- o Advanced cluster administration functions, such as load balancing, node pools, logging and monitoring, will be made available to the users as a bonus.
- o Compute Engine instance provides load balancing and distribution
- o Within a cluster, node pools are used to identify subsets of nodes in order to provide greater flexibility.
- o Logging and monitoring in conjunction with Cloud Monitoring so that you can see within your cluster.

- **Google APP Engine**: GCP's provision of a platform as a service for the development and deployment of scalable apps is known as Google APP Engine. It is a kind of computing known as serverless computing, and gives the users the ability to execute their code in a computing environment that does not involve the setting up of virtual machines or Kubernetes clusters.

  It is compatible with a variety of programming languages, including Java, Python, Node.js, and Go. Users have the option of developing their apps in any of the supported languages. The Google App Engine is equipped with a number of APIs and services that make it possible for developers to create applications that are powerful and packed with features. These characteristics are as follows:

- o Access to the application log
- o Blobstore, serve large data objects

Other important characteristics include a pay-as-you-go strategy, which means that you only pay for the resources employed. When there is a spike in the number of users using an application, the app engine will immediately increase the number of available resources, and vice versa.

Effective Diagnostic Services include Cloud Monitoring and Cloud Logging, which assist in running app scans to locate faults in the application. The app reporting document provides developers with assistance in immediately fixing any faults they find.

As a component of A/B testing, *Traffic Splitting* is a feature that allows app engine to automatically direct different versions of incoming traffic to various app iterations. Users are able to plan the subsequent increments depending on which version of the software functions the most effectively.

There are two distinct kinds of app engines:

- Standard APP Engine Applications are completely separate from the operating system of the server and any other applications that may be executing on that server at the same time. Along with the application code, there is no need for any operating system packages or other built software to be installed.
- The second kind is called Flexible APP Engine.

Docker containers are executed by users inside the App Engine environment. Additionally, it is necessary to install libraries or other third-party software in order to execute application code.

- **Google Cloud Functions**: The Google Cloud Function is a lightweight serverless computing solution that is used for event-driven processing. It is a **function as a service** (**FAAS**) product of GCP. Through the use of Cloud Functions, the time-consuming tasks of maintaining servers, setting software, upgrading frameworks, and patching operating systems are eliminated.

  The user has to provide code for Cloud Functions to start running in response to an event since GCP completely manages both the software and the infrastructure. Cloud events are occurrences that take place inside a cloud computing environment, such as changes to the data stored in a database, additions of new files to a storage system, and even the construction of new virtual machines are all instances of these kinds of operations. A trigger is a declaration that you are interested in a certain event or set of events. Binding a function to a trigger allows user to capture and act on events. Event data is the data that is passed to Cloud Function when the event trigger results in function execution:

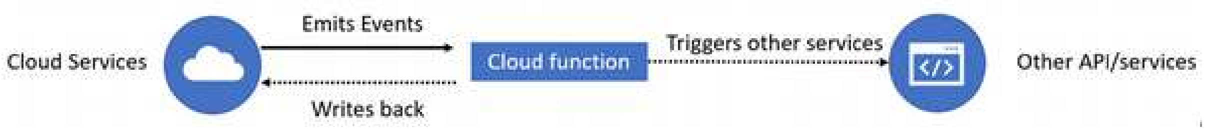


***Figure 1.31**: Cloud Function*

Summary of Compute services of GCP

The summary of compute services of GCP can be seen in *Figure 1.32*:

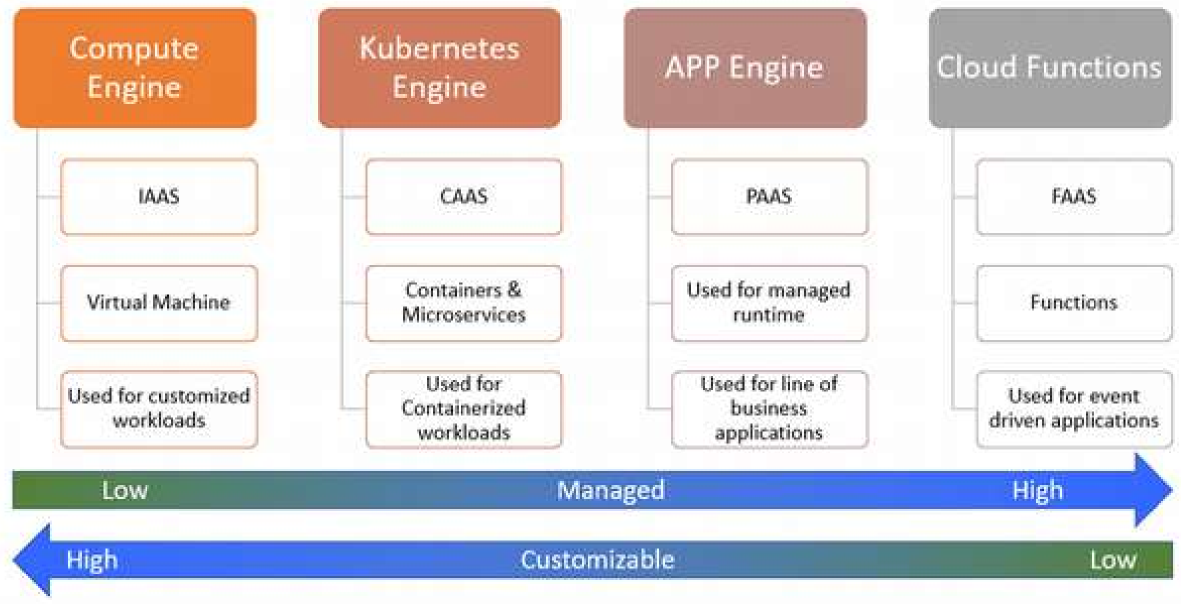

Compute Engine
IAAS
Virtual Machine
Used for customized workloads
Kubernetes Engine
CAAS
Containers & Microservices
Used for Containerized workloads
APP Engine
PAAS
Used for managed runtime
Used for line of business applications
Cloud Functions
FAAS
Functions
Used for event driven applications
Low
Managed
High
High
Customizable
Low

***Figure 1.32**: Summary Compute Service*

Creation of compute Engine (VM instances)

Compute Engine instances may run Google's public Linux and Windows Server images and private custom images. Docker containers may also be deployed on Container-Optimized OS public image instances. Users' needs to specify zone, OS, and machine configuration type (number of virtual CPUs and RAM) while creation. Each Compute Engine instance has a small OS-containing persistent boot drive and more storage can be added. VM instance default time is **Coordinated Universal**

Time (**UTC**). Follow the steps below to create a virtual machine instance:

Step 1: Open compute engine:

Follow steps described in *Figure 1.33* to open the compute engine of the platform:

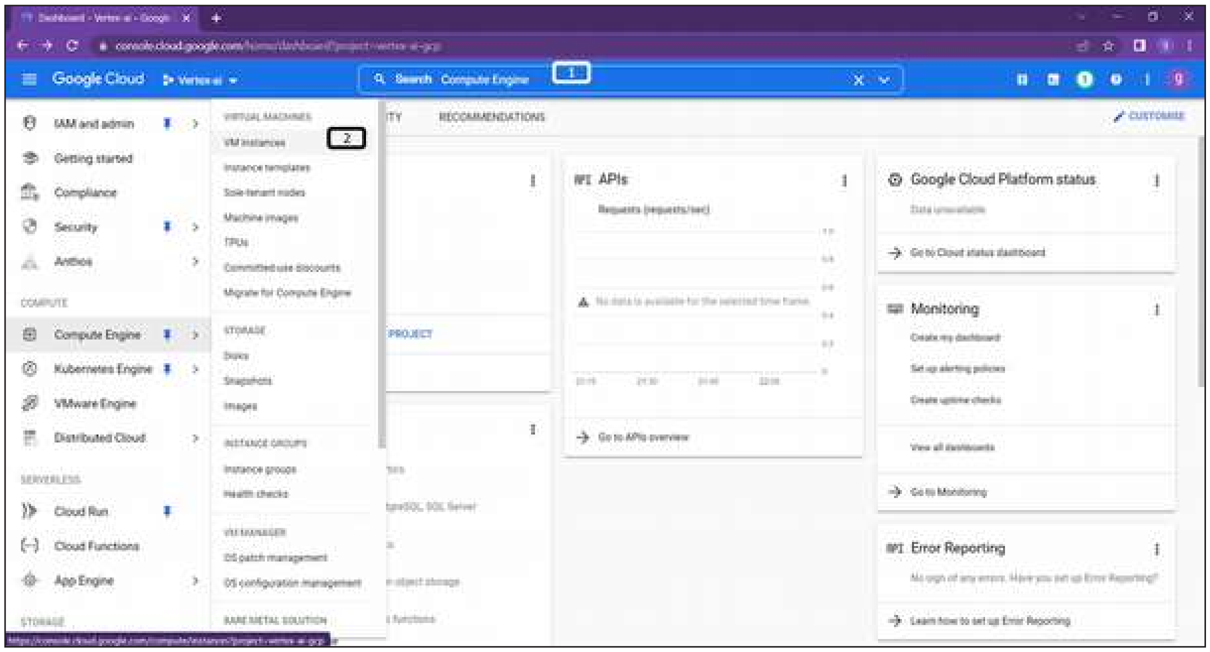


***Figure 1.33**: VM Creation*

1. Users can type Compute Engine in the search box.
2. Alternatively, users can navigate under **Compute-to-Compute Engine | VM instances**.

Step 2: Enable API for compute engine:

Users will be prompted to enable APIs when the resources are accessed for the first time as shown in *Figure 1.34*:

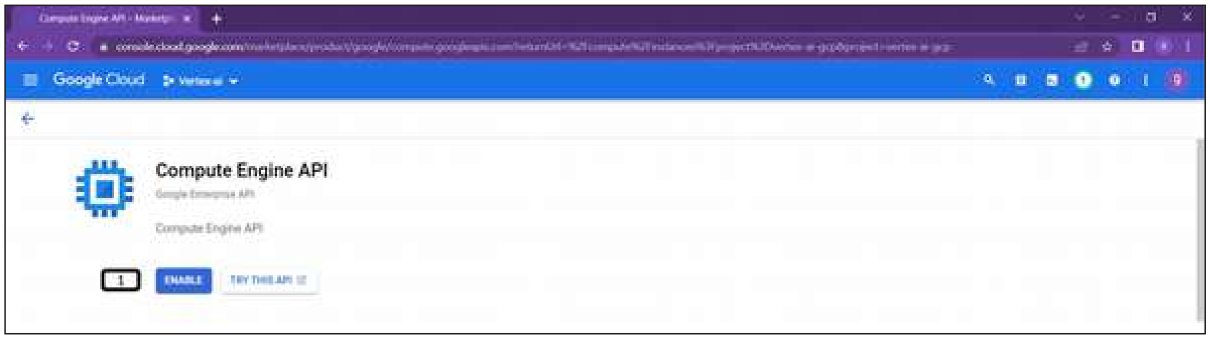


***Figure 1.34:** VM Creation API enablement*

1. If promoted for API enablement, then enable the API.

Step 3: VM instance selection:

Follow the steps described in *Figure 1.35* to select the VM:

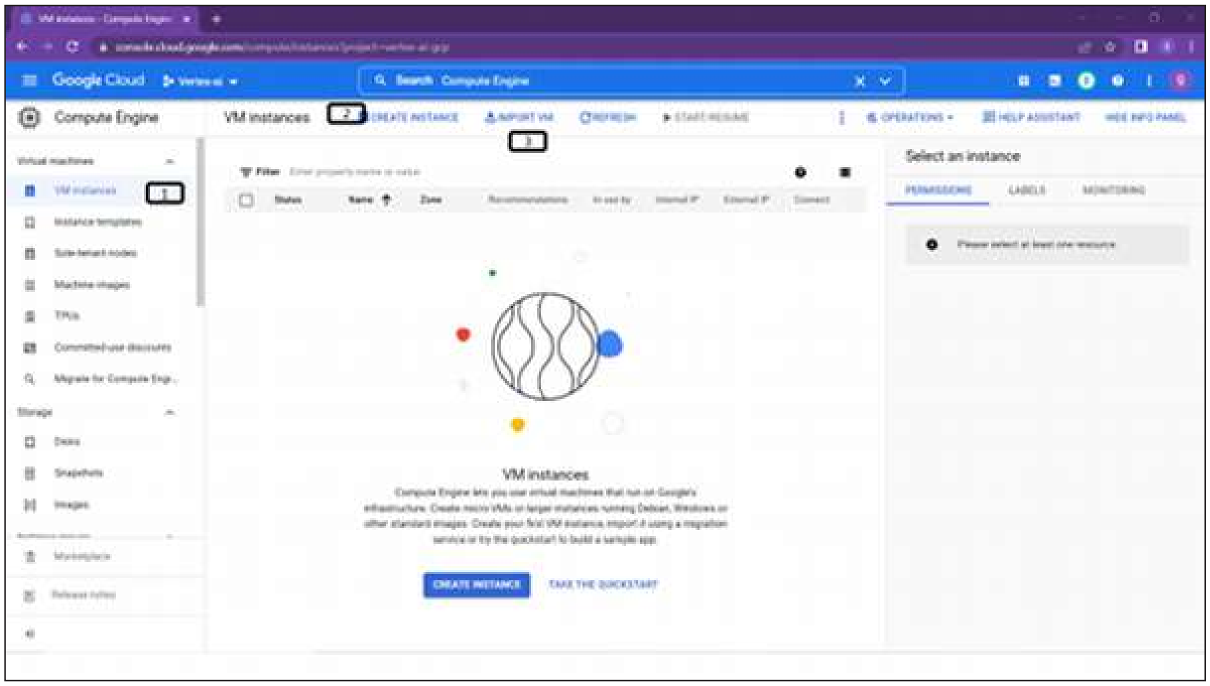
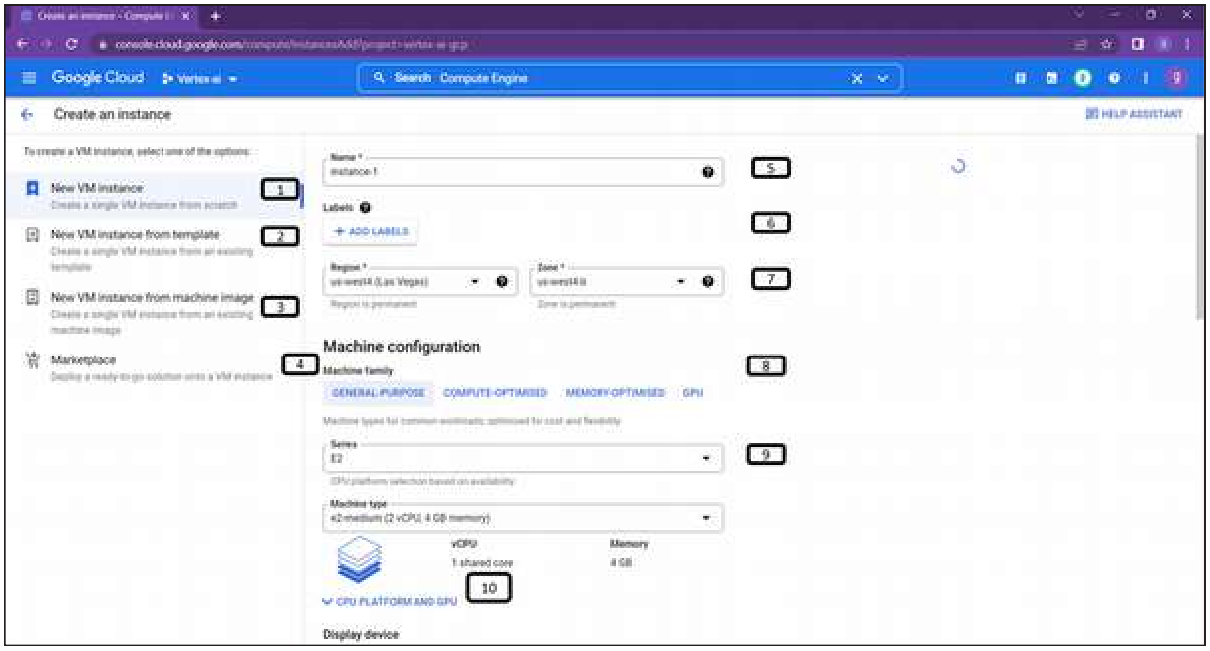


Figure 1.35: *VM Creation*

1. Select **VM instances**.
2. Click on **CREATE INSTANCE** to begin the creation process.
3. GCP provides option of **IMPORT VM** for the migration of compute engine.

Step 4: VM instance creation:

Follow the steps described in *Figure 1.36* to select location and machine type for the VM instance:


Create an instance

To create a VM instance, select one of the options:

- New VM instance — Create a single VM instance from scratch
- New VM instance from template — Create a single VM instance from an existing template
- New VM instance from machine image — Create a single VM instance from an existing machine image
- Marketplace — Deploy a ready-to-go solution onto a VM instance

Name *: instance-1

Labels: + ADD LABELS

Region *: us-west4 (Las Vegas) — Region is permanent

Zone *: us-west4-b — Zone is permanent

Machine configuration

Machine family: GENERAL-PURPOSE | COMPUTE-OPTIMISED | MEMORY-OPTIMISED | GPU

Machine types for common workloads, optimised for cost and flexibility

Series: E2 — CPU platform selection based on availability

Machine type: e2-medium (2 vCPU, 4 GB memory)

vCPU	Memory
1 shared core	4 GB

CPU PLATFORM AND GPU

Display device

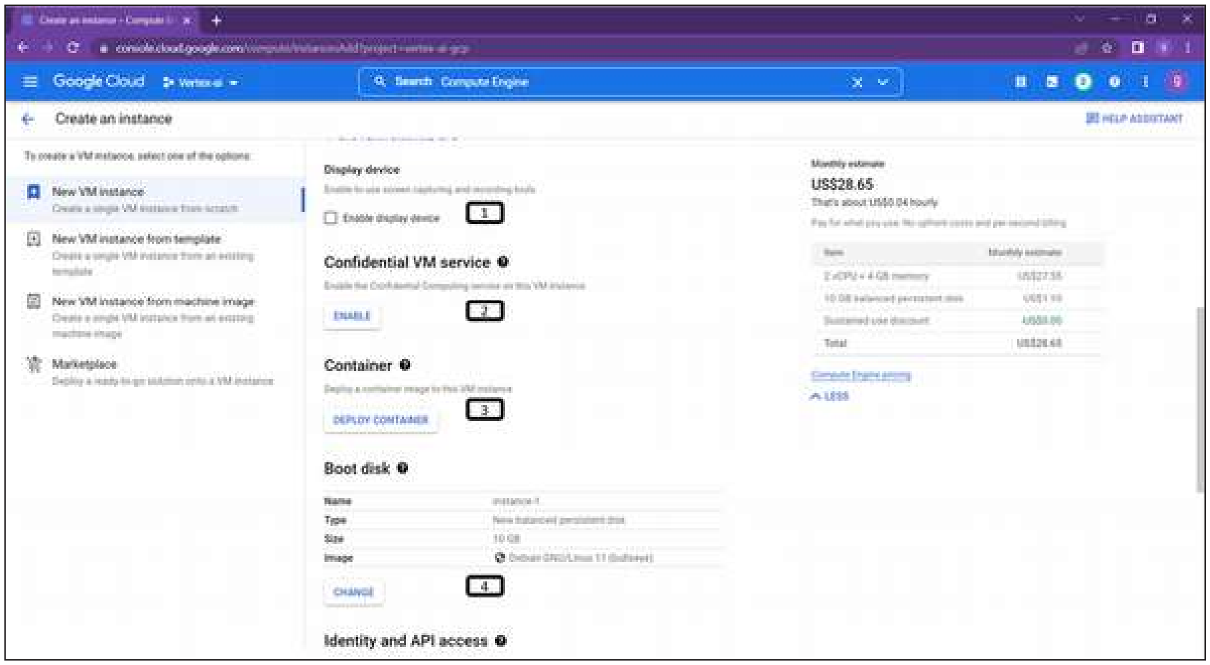
***Figure 1.36**: Location, machine selection for VM instance*

1. GCP provides multiple options for VM creation, first one for creating VM from scratch. We will create a **New VM instance**.
2. VM's can be created from template. Users can create a template and use it for future purpose.
3. VM's can be created from machine images, machine images is resource that stores configuration, metadata and other information to create a VM.
4. GCP also provides option to choose deploy ready to go solution on to VM instance.
5. Instance name has to be given by the user.
6. Labels are optional, they provides key/value pairs to group VMs together.
7. Select region and zone.
8. GCP provides a wide range of options for users in machine configurations starting from **GENERAL-PURPOSE**, **COMPUTE-OPTIMIZED**, **MEMORY-OPTIMIZED** AND **GPU** based.
9. Under each machine family configuration, GCP provides few series machines and machine type (to choose between CPU and memory).
10. Users can choose CPU platform and GPU, if they want to select **vCPU**s to core ratio and visible core count.

NOTE: Try selecting different machine configurations and observe the variation in the price estimates.

Step 5: VM instance disk and container selection:

Follow the steps described in *Figure 1.37* to select boot disks for the instance:


Create an instance

To create a VM instance, select one of the options:

New VM instance
Create a single VM instance from scratch

New VM instance from template
Create a single VM instance from an existing template

New VM instance from machine image
Create a single VM instance from an existing machine image

Marketplace
Deploy a ready-to-go solution onto a VM instance

Display device
Enable to use screen capturing and recording tools
Enable display device

Confidential VM service
Enable the Confidential Computing service on this VM instance
ENABLE

Container
Deploy a container image to this VM instance
DEPLOY CONTAINER

Boot disk

Name	instance-1
Type	New balanced persistent disk
Size	10 GB
Image	Debian GNU/Linux 11 (bullseye)

CHANGE

Identity and API access

Monthly estimate
US$28.65
That's about US$0.04 hourly
Pay for what you use: No upfront costs and per second billing

Item	Monthly estimate
2 vCPU + 4 GB memory	US$27.55
10 GB balanced persistent disk	US$1.10
Sustained use discount	US$0.00
Total	US$28.65

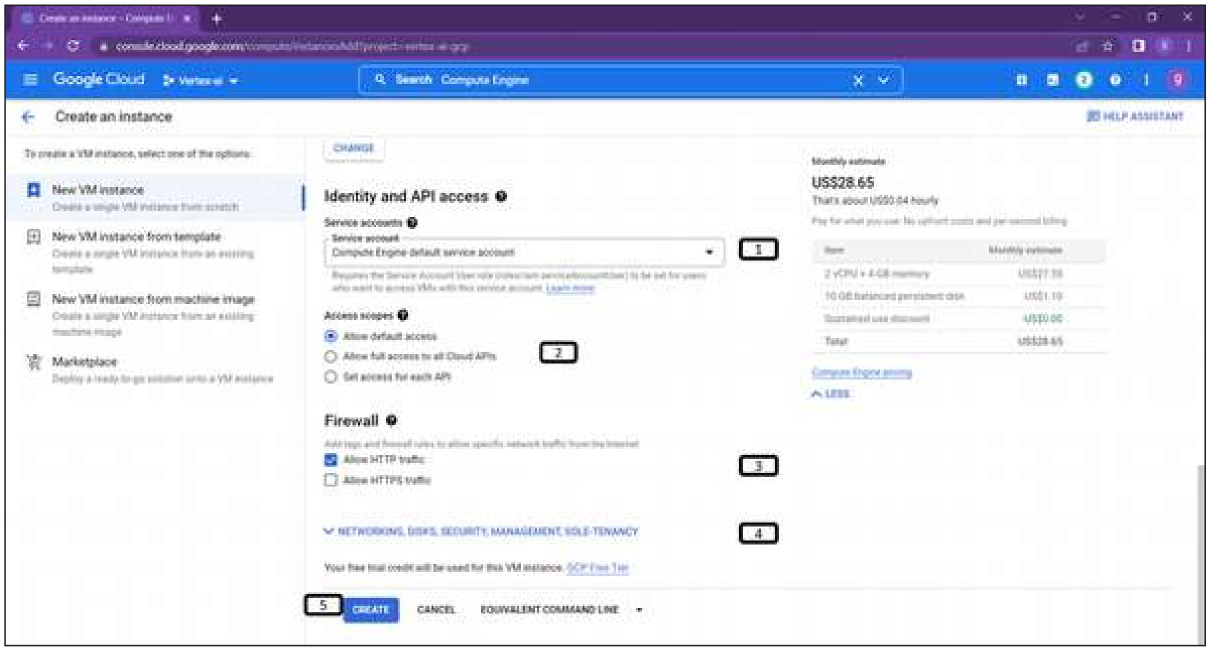
Compute Engine pricing
LESS

Figure 1.37: *Boot disk selection for VM instance*

1. **Enable display tools**, enable use of screen recording tools.
2. Adds protection to your data in use by keeping memory of this VM encrypted.
3. Deploy Container option is helpful when there is a need to deploy a container to VM instance by using a container-optimized OS image.
4. Users can change the operating system, size and type the hard disk (HDD/SDD) by clicking on change.

Step 6: VM instance creation access settings:

Follow the steps described in *Figure 1.38* to configure access:


Create an instance

To create a VM instance, select one of the options:

- New VM instance — Create a single VM instance from scratch
- New VM instance from template — Create a single VM instance from an existing template
- New VM instance from machine image — Create a single VM instance from an existing machine image
- Marketplace — Deploy a ready-to-go solution onto a VM instance

Identity and API access

Service accounts
Service account: Compute Engine default service account

Access scopes
- [✓ Allow default access]
- Allow full access to all Cloud APIs
- Set access for each API

Firewall
Add tags and firewall rules to allow specific network traffic from the Internet
- [✓ Allow HTTP traffic]
- Allow HTTPS traffic

NETWORKING, DISKS, SECURITY, MANAGEMENT, SOLE-TENANCY

Your free trial credit will be used for this VM instance. GCP Free Tier

Monthly estimate
US$28.65
That's about US$0.04 hourly
Pay for what you use: No upfront costs and per second billing

Item	Monthly estimate
2 vCPU + 4 GB memory	US$27.55
10 GB balanced persistent disk	US$1.10
Sustained use discount	US$0.00
Total	US$28.65

Figure 1.38: access control for VM instance

1. Choose to set the default service account associated with the VM instance or users can create a service account and use the same for the compute engine.
2. Users' needs select how VM needs to be accessed, they can choose allow default access, or allow all the APIs or only few APIs to access the VM
3. By default, all the internet traffic will be blocked, we need to enable them.
4. Additional options such as disk protection, reservations, network tags, change host name, delete or retain boot disk when instance is deleted etc., are provided under networking, disks, security and management.
5. Click on **Create**.

Accessing the VM instance

Once the virtual machine is created, it will be listed under VM instances as shown in *Figure 1.39*:

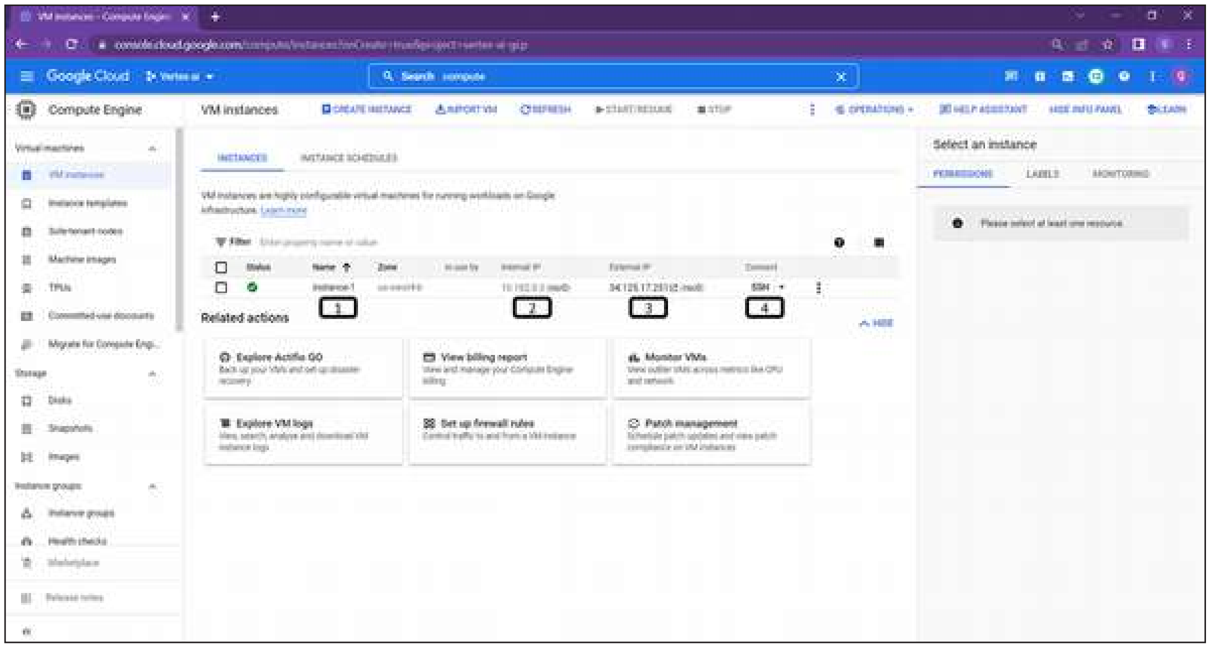


***Figure 1.39**: VM instance created and listed under VM instances*

1. Click on the instance type to see the details of the VM (including region, zone of VM, machine configuration, load monitoring for the CPU utilization, memory utilization, and so on).
2. **Internal IP** address is the subnet IP address.
3. **External IP** address to communicate with external devices
4. **SSH** for connecting to the VM.

Step 1: Accessing the VM

Follow the steps described in *Figure 1.40* and *Figure 1.41* to access the created VM instance:

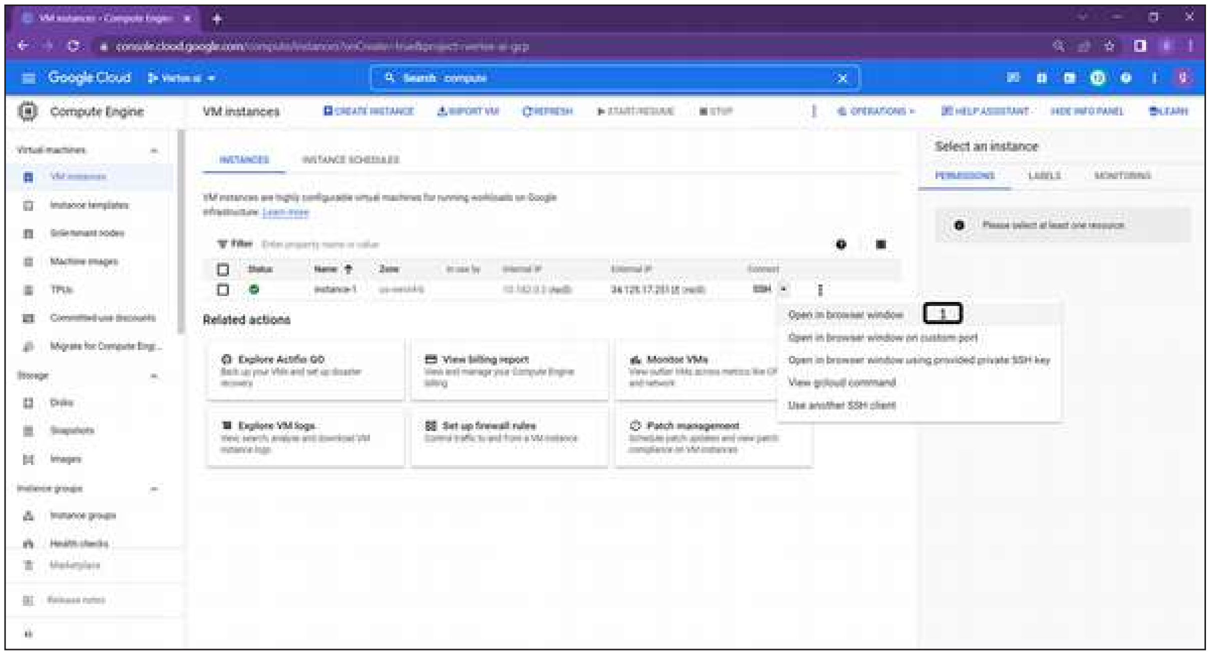
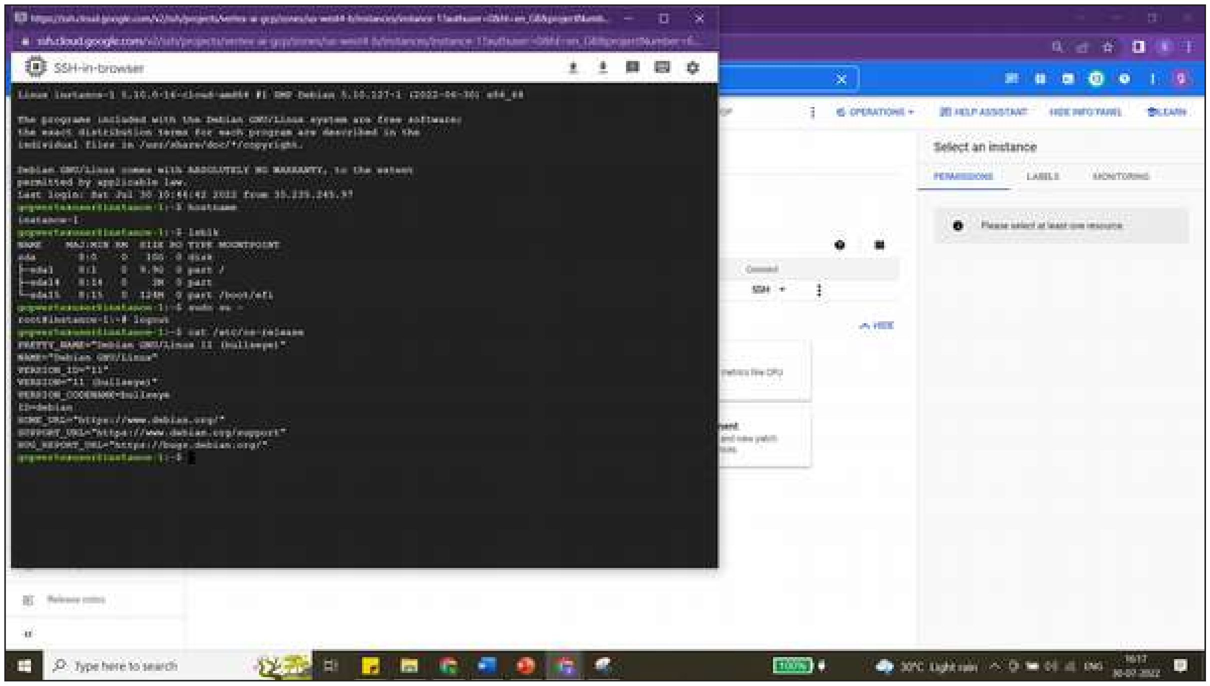


***Figure 1.40**: Accessing the VM*

1. Click on the **SSH options** and select open in browser window.

New window will open, SSH keys will be transferred and connection will be established as shown in *Figure 1.41*:



```
Linux instance-1 5.10.0-16-cloud-amd64 #1 SMP Debian 5.10.127-1 (2022-06-30) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Jul 30 10:46:42 2022 from 35.235.245.97
[illegible]@instance-1:~$ hostname
instance-1
[illegible]@instance-1:~$ lsblk
NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda       8:0    0   10G  0 disk
├─sda1    8:1    0  9.9G  0 part /
├─sda14   8:14   0    3M  0 part
└─sda15   8:15   0  124M  0 part /boot/efi
[illegible]@instance-1:~$ sudo su -
root@instance-1:~# logout
[illegible]@instance-1:~$ cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 11 (bullseye)"
NAME="Debian GNU/Linux"
VERSION_ID="11"
VERSION="11 (bullseye)"
VERSION_CODENAME=bullseye
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
[illegible]@instance-1:~$
```

***Figure 1.41**: Accessing the VM*

Try these commands in the window:

a. **`Hostname`** (to get the name of VM instance)

b. **`Lsblk`** (for the disk information)

c. **`Sudo su`** – (to check the root access)

d. **`Logout`** (to exit route access)

e. **`Cat /etc/os-release`** (to see the Operating system details)

f. **`Logout`** (to close the connection)

Deletion of VM instance

Follow the steps described in *Figure 1.42* to delete the VM instance:

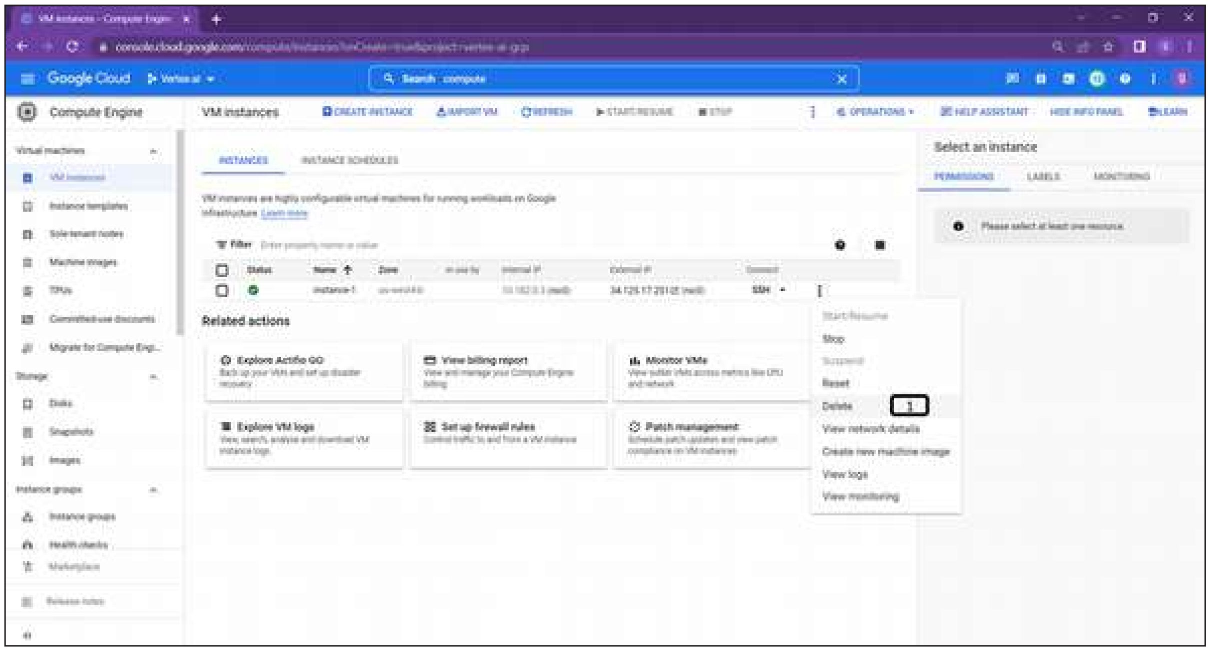


Figure 1.42*: Deletion of VM*

1. Click on three dots and select **Delete**.
2. The pop-up window will ask for the confirmation.

NOTE: All the compute resources are expensive in any platforms, make sure to decommission them after use.

BigQuery

Anyone has the power to analyze terabytes of data in the matter of seconds using BigQuery, which is a fully managed and serverless data warehouse solution offered in the Google Cloud Platform. This facilitates the management and analysis of data via the use of built-in capabilities such as machine learning, geospatial analysis, and business intelligence. The serverless architecture of BigQuery employs SQL queries to provide organizations with answers to their most pressing problems while requiring no administration of their underlying infrastructure.

The Google BigQuery architecture is built on Dremel, a distributed system designed by Google to query massive databases. The execution of the query is broken up into slots in Dremel, which enables fairness even when numerous users are concurrently searching the database. Under the hood, Dremel uses Jupiter, Google's internal data center network, to access the data storage on the distributed file system that is nicknamed Colossus. Jupiter is also the backbone of Google Cloud Storage. Data replication, data recovery, and management of data dissemination are all handled by Colossus. BigQuery utilizes a columnar storage structure for its data, which results in a high compression ratio and a high scan throughput. On the other hand, you may also utilize BigQuery with data that is stored in other Google Cloud services, such as Bigtable, Cloud Storage, Cloud SQL, and Google Drive.

Because of its architecture that is specifically designed to handle large amounts of data, BigQuery performs at its peak when it is given several petabytes of information to evaluate. BigQuery is best suited for use cases in which people need to make interactive ad-hoc queries of read-only datasets. In most cases, complicated analytical queries to a relational database take several seconds to run. This is an ideal situation for using BigQuery, which is why it is often utilized towards the end of the Big Data ETL pipeline, on top of processed data. BigQuery performs very well in circumstances in which the data does not undergo frequent changes because it comes with its own cache.

There is no need to install, setup, or maintain any underlying infrastructure in order for it to function properly because it is a completely managed service. Customers are only paid for the amount of data they keep and the number of queries they run against the database.

Working with BigQuery

Step 1: Opening BigQuery:

Follow the steps described in *Figure 1.43* to open BigQuery:

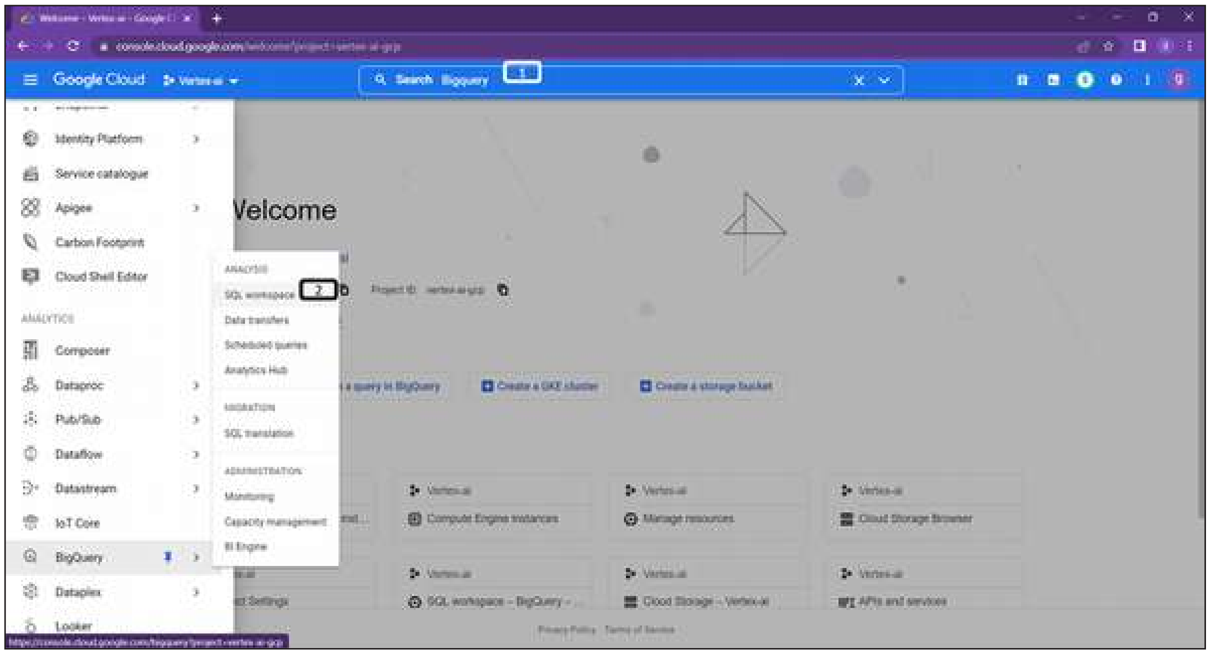


***Figure 1.43**: BigQuery*

1. Users can type **`BigQuery`** in the search box.
2. Alternatively, they can navigate under **Analytics to BigQuery | SQL workspace**.

Step 2: BigQuery landing page:

Landing page of BigQuery is as shown in *Figure 1.44*:

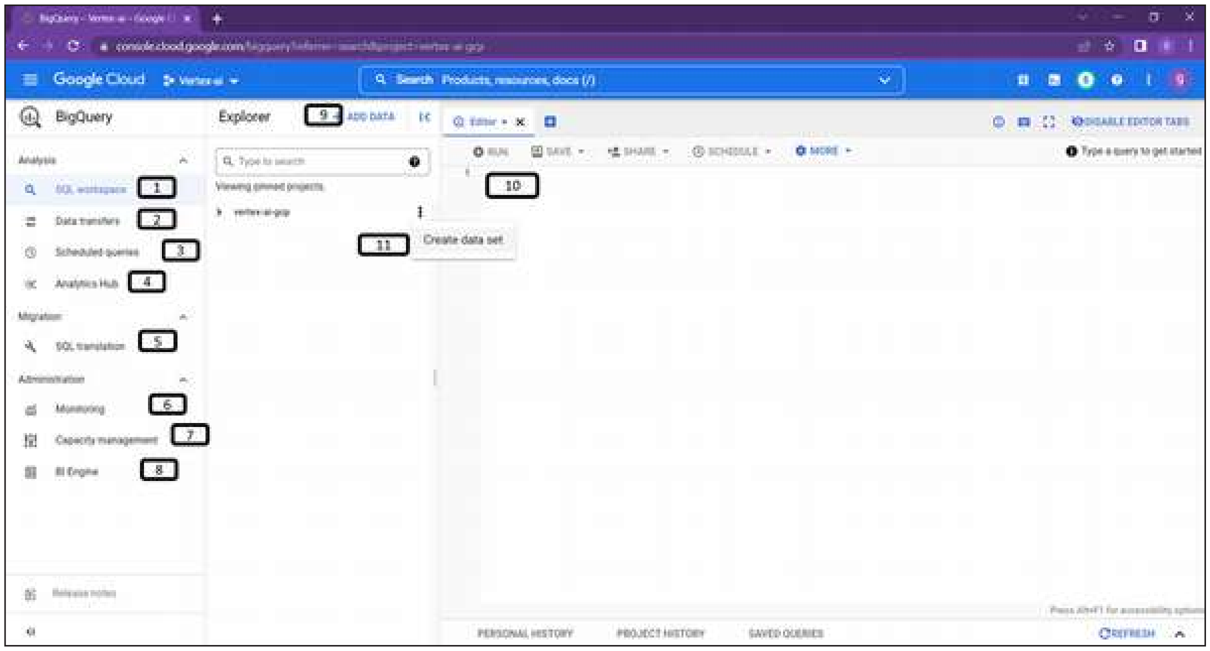


Figure 1.44: Landing page of BigQuery

1. Click on **SQL workspace** to create and run queries, work with tables & also for Job history.
2. The BigQuery **Data transfers** Service automates the scheduled, controlled data transfer into BigQuery.
3. **Scheduled queries** option is used to run queries on a recurring basis. Queries need to be in standard SQL.
4. **Analytics Hub** helps to transmit data assets across organizations effectively and securely.
5. **SQL translation** helps user to translate SQL dialect into a BigQuery Standard SQL query.
6. . Performance monitoring information and log data of BigQuery are available under **Monitoring** section.
7. **Capacity management** helps in capacity planning based on historical utilization information.

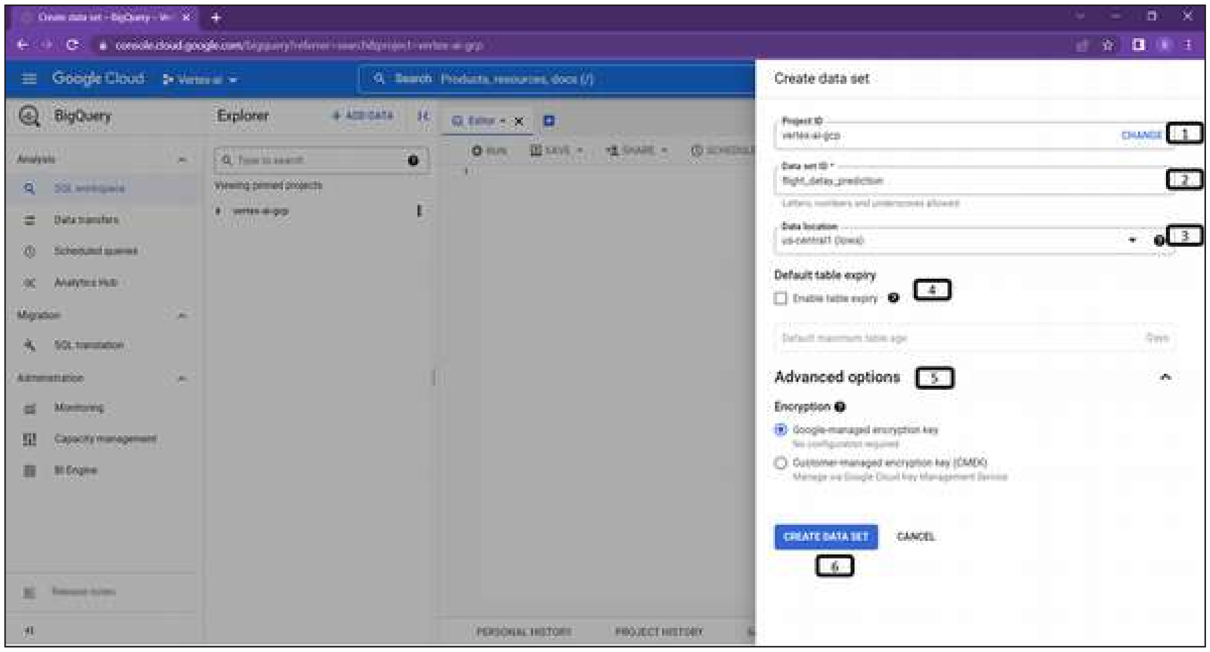
8. **BI Engine** is an in-memory analysis service.
9. **ADD DATA** provides options to import data from various sources, including public and private datasets.
10. **Editor window** to write queries to fetch the data.
11. Projects that are created are listed (Project ID will be displayed not the project Name). Click on three dots and select **Create data set**.

Step 3: BigQuery dataset creation:

We are using dataset from Kaggle which is listed under the public domain license. You can download the data in csv format to your local machine.

https://www.kaggle.com/datasets/jimschacko/airlines-dataset-to-predict-a-delay

Follow the steps described in *Figure 1.45* to create dataset in BigQuery:



Create data set

Project ID: vertex-ai-gcp

Data set ID *: flight_delay_prediction

Letters, numbers and underscores allowed

Data location: us-central1 (Iowa)

Default table expiry

Enable table expiry

Default maximum table age: Days

Advanced options

Encryption

Google-managed encryption key (No configuration required) [✓]

Customer-managed encryption key (CMEK) (Manage via Google Cloud Key Management Service)

***Figure 1.45**: BigQuery dataset creation*

1. Select the project. Dataset will be created under the selected project.
2. **Data set ID** : Name for the dataset.
3. **Data location** : Select the location.

4. Enable table expiry : If this option is selected then any table that will be created will be deleted after number of days mentioned by the user in the box.
5. **Advanced options** : For encryption to choose between Google managed or customer managed encryption.
6. Click on **CREATE DATA SET**.

Step 4: Table creation

Follow the steps described in *Figure 1.46* to create table under the dataset:

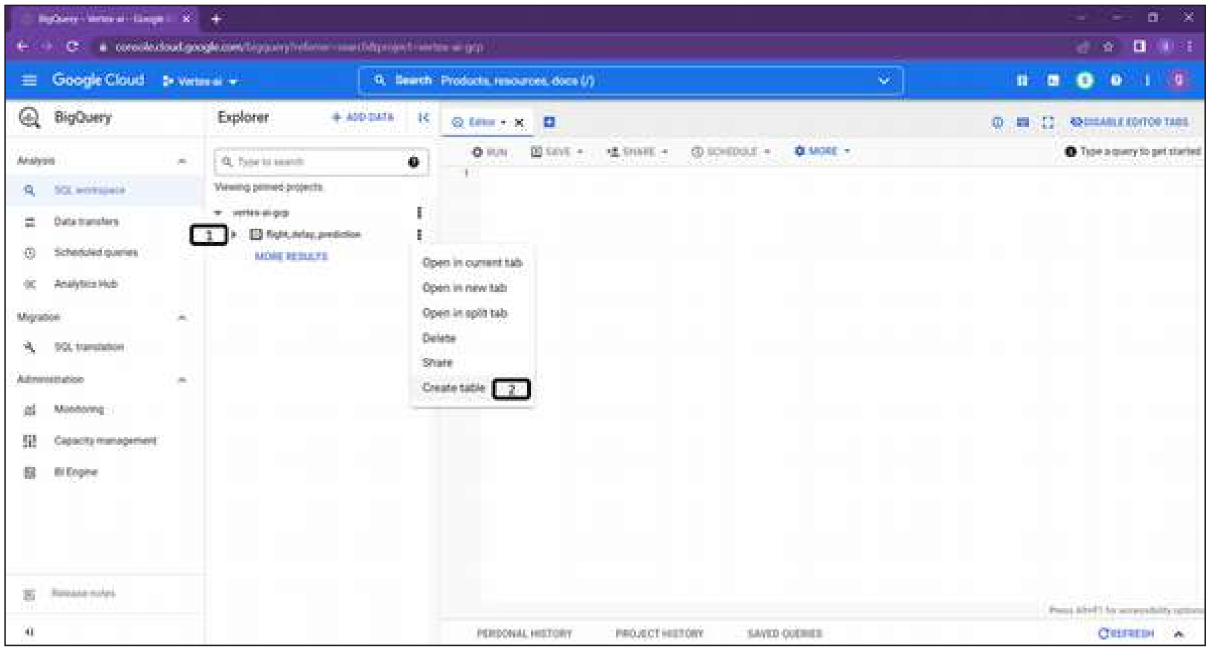
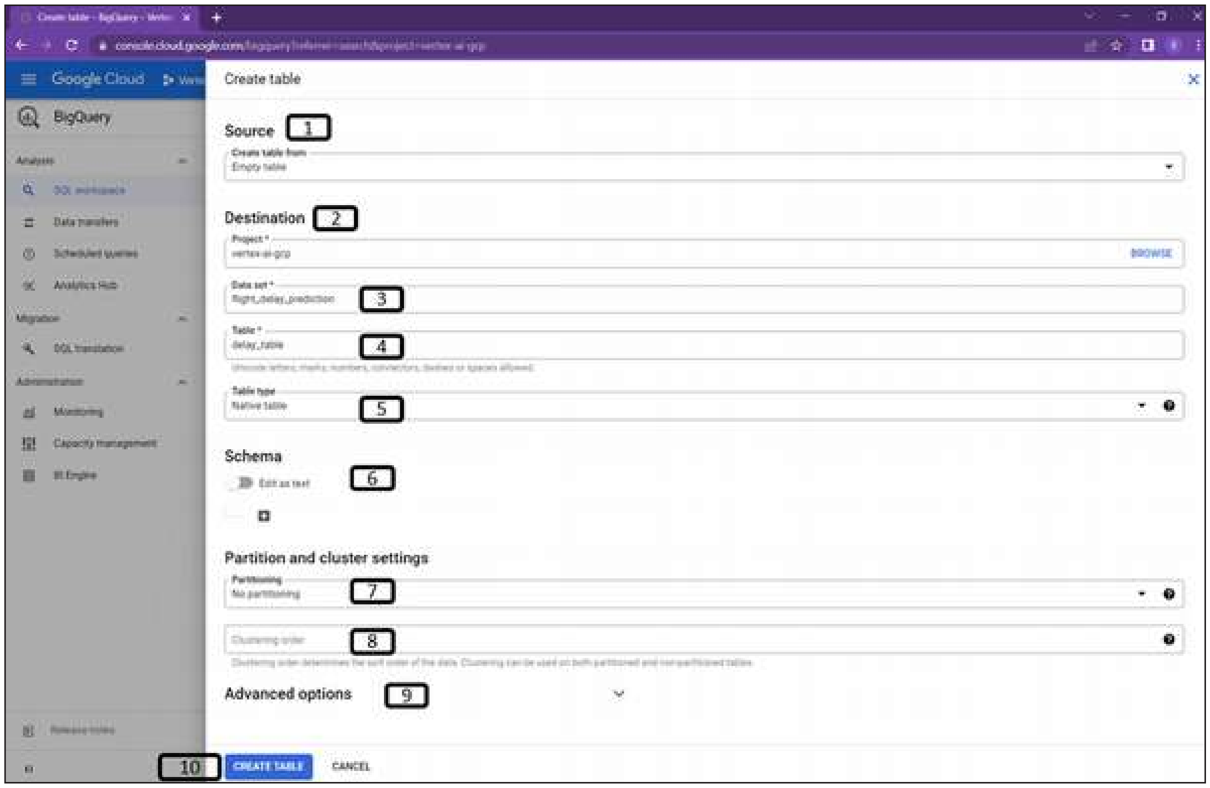


Figure 1.46: *Table creation in a dataset*

1. Created dataset will be listed under the selected project.
2. Select **Create table** to import the data.

Step 5: Data selection for table

Follow the steps described in *Figure 1.47* to upload data to BigQuery:


Create table

Source 1

Create table from: Empty table

Destination 2

Project *: vertex-ai-gcp

Data set *: flight_delay_prediction 3

Table *: delay_table 4

Table type: Native table 5

Schema

Edit as text 6

Partition and cluster settings

Partitioning: No partitioning 7

Clustering order 8

Advanced options 9

10 CREATE TABLE CANCEL

Figure 1.47: *Uploading data to BigQuery table*

1. Table can be created by uploading data from local machine, from google cloud storage, google drive, google bigtable, Amazon S3 bucket, Azure blob storage.

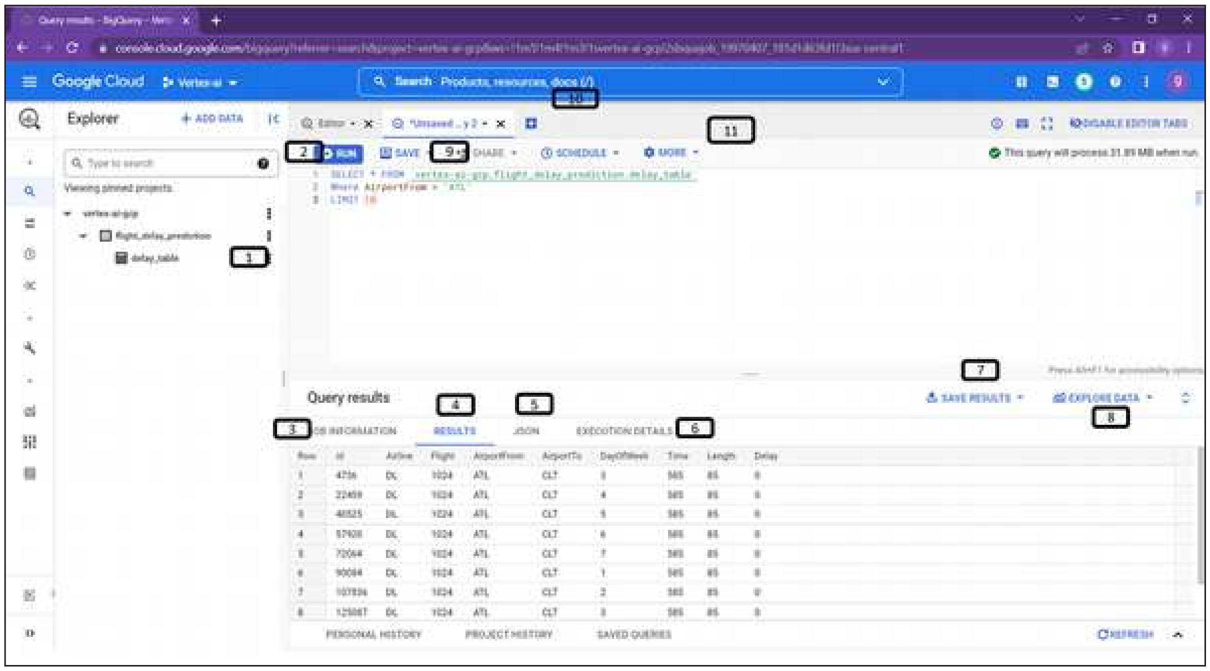
For this example, data is downloaded to local machine and choose **upload** option that data is picked from local machine. Once the upload option is selected, BigQuery provides option to select the and the format of the file (csv).

2. **Project** : Project under which dataset was created.
3. **Dataset Name** : Under which table needs to be created.
4. Enter the name of the table that needs to be created.
5. **Table type** : Native type is the only option; **Native table**s are tables backed by native BigQuery storage.
6. Edit **schema** : Users can enable this option to manually enter the schema of the data. (Once data is chosen, auto-detect option will be enabled).
7. **Partition** : Partitioning divides table into smaller segments. Portioning column will be created, filter conditions can be used on the Portioning column to reduce the time required to scan data.

8. Clustering organizes data based on the contents of specified columns in the schema.
9. Encryption options are provided under the **`Advanced options`**.
10. Click on **create Table**.

Step 6: Running query on BigQuery table:

Follow the steps described in *Figure 1.48* to run SQL query on the uploaded data:



```
SELECT * FROM `vertex-ai-gcp.flight_delay_prediction.delay_table`
Where AirportFrom = "ATL"
LIMIT 10
```

Row	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay
1	4736	DL	1024	ATL	CLT	3	585	85	0
2	22459	DL	1024	ATL	CLT	4	585	85	0
3	40525	DL	1024	ATL	CLT	5	585	85	0
4	57908	DL	1024	ATL	CLT	6	585	85	0
5	72064	DL	1024	ATL	CLT	7	585	85	0
6	90084	DL	1024	ATL	CLT	1	585	85	0
7	107836	DL	1024	ATL	CLT	2	585	85	0
8	125087	DL	1024	ATL	CLT	3	585	85	0

Figure 1.48: *SQL query to fetch data*

1. Created table name will be displayed under the selected dataset.
2. In the SQL editor, users can write the query and click on **Run** for execution. Below Query is used for the example:

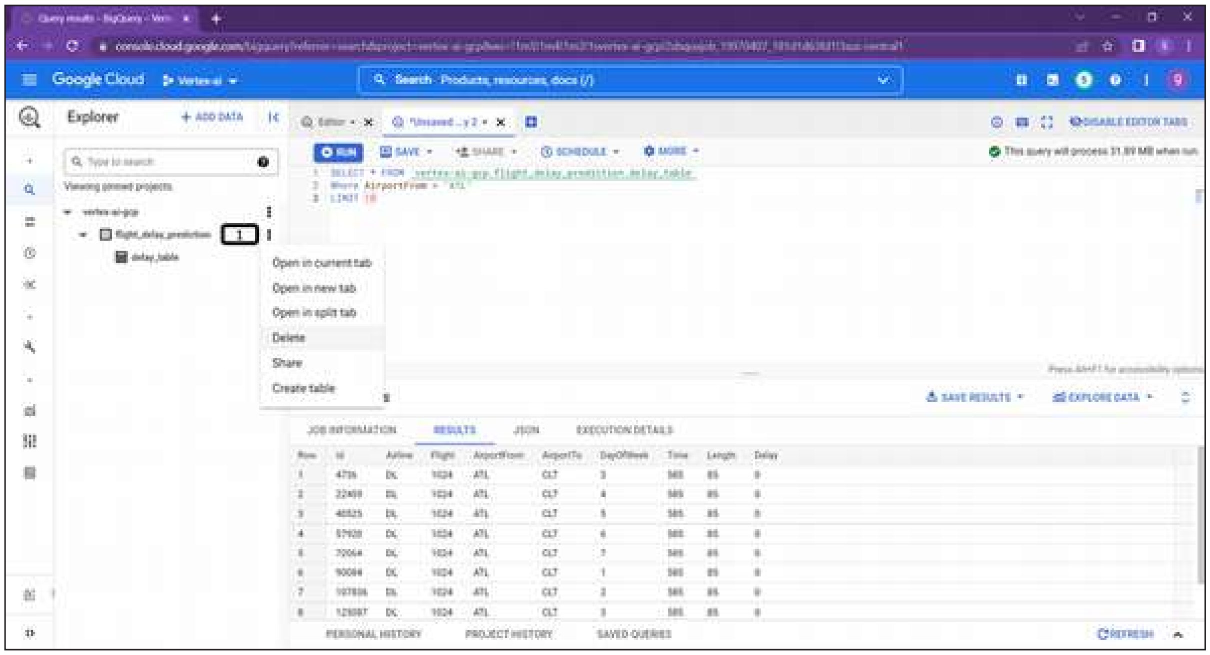
   ```
   Select * from 'vertex-ai-gcp.flight_delay_prediction.delay_table'

   Where AirpottFrom = "ATL"

   LIMIT 10
   ```

   `vertex-ai-gcp` is the project name, **`flight_delay_prediction`** is the dataset name, **`delay_table`** is the table name.
3. Results of the query is displayed in the **Query results** section. **JOB INFORMATION** provides information about Job ID, time taken, location, and so on.

4. Results of the query are displayed in the table structure in the **RESULTS** section.
5. Results of the query are displayed in the JSON structure.
6. Execution details contains information about amount of data processed, slot time consumed for execution, and so on.
7. **SAVE RESULTS** provides options to save the results in CSV, JSON or in the BigQuery table.
8. **EXPLORE DATA** provides options for exploring with data studio (a visualization tool).
9. **SAVE** option provides option to save the query for the later use.
10. Schedule provides the option of scheduling the query.
11. **MORE** provides options of formatting query, SQLL translation, etc.,

Step 7: Dataset deletion:

Follow the steps described in *Figure 1.49* to delete the dataset:



```
SELECT * FROM `vertex-ai-gcp.flight_delay_prediction.delay_table`
Where AirportFrom = 'ATL'
LIMIT 10
```

This query will process 31.89 MB when run.

Row	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay
1	4736	DL	1024	ATL	CLT	3	585	85	0
2	22459	DL	1024	ATL	CLT	4	585	85	0
3	40525	DL	1024	ATL	CLT	5	585	85	0
4	57928	DL	1024	ATL	CLT	6	585	85	0
5	70064	DL	1024	ATL	CLT	7	585	85	0
6	90084	DL	1024	ATL	CLT	1	585	85	0
7	107806	DL	1024	ATL	CLT	2	585	85	0
8	125087	DL	1024	ATL	CLT	3	585	85	0

Figure 1.49: *Bigquery dataset deletion*

1. Click on three dots and select **Delete**. A pop-up window will ask for the confirmation.

Identity and access management

Imagine that you have just joined an organization. They provide you with an ID card as well as an access card. With this access card, you will be allowed to enter a certain number of locations, including your workplace, the parking lot, and the cafeteria, among other places. You need to seek special authorization in order to have access to the lab or document area where you have some confidential information if you want to see such information. In a manner similar to that of GCP, we have something called **identity and access management**, or **IAM** for short.

Identity and access management is one of the most crucially important precautions that can be taken to ensure data safety in cloud computing environments like **Google Cloud Platform** (**GCP**). Every action that is performed, such as adding or deletion of resources, or communication between different services of the platform happens through API. As a result of this, it is essential to pay close attention to the rights that enable access to the resources and to make ensure that the absolute bare minimum number of permissions required to carry out business processes are made available. IAM is developed from the least privilege principle, that is, users will be denied of the access by default. Access needs to be opened for the users explicitly.

For the purpose of assigning permissions, GCP uses a system known as **Role Based Access Control** (**RBAC**). RBAC denotes the practice of basing the assignment of permissions on the tasks or responsibilities that are associated with an identity.

A ROLE is a designated set of permissions that determines whether or not an individual has the authority to carry out certain activities on a resource. Within IAM, we are unable to directly provide permissions to users. Instead of giving them permissions, we give them roles, which are collections of one or more permissions. The format **`service.resource.verb`** is used to indicate permissions. One example of this format is the **`compute.instances.list`** permission. The kind of activities that may be performed on a resource are established by its permissions. A member receives all of the permissions associated with the position when the role is provided to them.

There are three types of roles in Google Cloud IAM:

- **Basic roles**
  - Includes Owner, Editor, and Viewer role
  - Provides broad level of permissions and it is not recommended

- **Predefined roles**
  - Google creates and updates roles as needed, for as when Google Cloud introduces new features or services.
  - Example: **roles/notebooks.admin ; roles/ml.modelUser**
- **Custom roles**
  - Provides granular access according to a user-defined list of permissions
  - We can create a custom IAM role with one or more permissions and then grant that custom role to users or groups.
  - GCP does not update custom roles when new permissions, features, or services are introduced to platform

Members or users are also referred to as IDENTITY in GCP. Identity can be as follows:

- **User account** : Users with GCP access will be treated as a Useraccount or sometimes called as google account.
- **Service account** : It is not associated with user, but it is meant for communication between the GCP resources. We can create service accounts but we cannot access the resources directly from service accounts. There are two types of service accounts.
  - **Default service accounts** : When you activate or utilize certain Google Cloud services, which allow the service to launch tasks that access other Google Cloud resources, Google Cloud establishes user-managed service accounts.
  - **User-managed service accounts**: IAM API, Cloud Console, or gcloud command-line tool are used to establish user-managed service accounts. Creator manages and secures accounts. We will create one in the next chapter.
- **Google group**: Groups helps to manage users at scale. It is a simple way to attach roles to users with the same job functions. Each member of a Google group inherits the IAM roles granted to that group. A user can belong to multiple groups.
- **G suite domain**: It is user account for the entire organization.
- **Cloud Identity domain**: If the organization is not using G suite domain, and wants to use the google cloud IAM then cloud Identity domain is used. However, Cloud Identity domain users don't have access to applications and features belonging to Google Workspace.

- **Alias**: Also referred as special identifier. Two special identifiers are listed below.
    - **`allauthenticatedUsers`**: is a special identifier that represents all service accounts and all users on the internet who have authenticated with a Google Account
    - **`allusers`**: is a special identifier that represents anyone who is on the internet, including authenticated and unauthenticated users.
- Policy is enforceable against Roles and Members. This policy is related to a resource and specifies and governs which members are given certain responsibilities. The resource's policy outlines who (member) has what kind of access (role) to it.
- IAM policy may be configured at any level of the resource hierarchy, including the organization, folder, project, and resource levels. Resources inherit all of their parent resources' policies since IAM policy inheritance is transitive. IAM determines if an activity is authorized by the resource's policy when an authenticated member tries to access it. *Figure 1.50* illustrates the relationship of role, members, resources and policy in IAM.

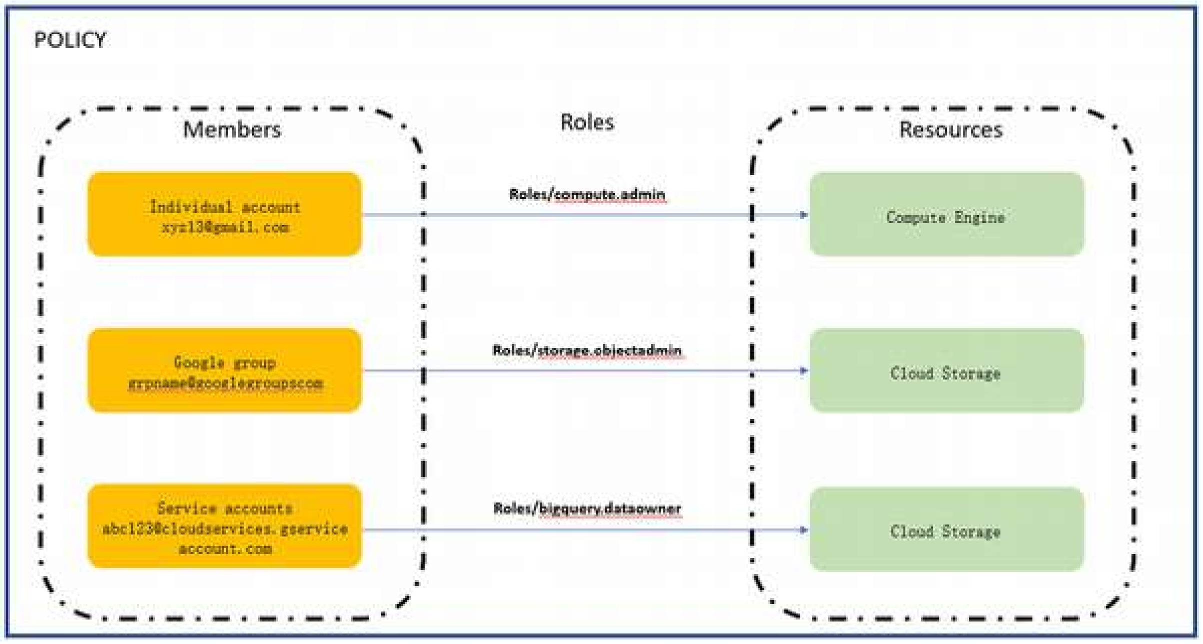

POLICY
Members
Roles
Resources
Individual account xyz13@gmail.com
Roles/compute.admin
Compute Engine
Google group grpname@googlegroupscom
Roles/storage.objectadmin
Cloud Storage
Service accounts abc123@cloudservices.gservice account.com
Roles/bigquery.dataowner
Cloud Storage

***Figure 1.50**: Identity and Access management*

Conclusion

In this chapter, we have learned what the cloud is, what are the benefits of using the cloud, why data scientists of today need to be aware of cloud platforms, and what the many types of service models that cloud platforms follow. We also discovered the hierarchy it adhere to (organizations, folders and projects). In addition to that, we focused on the more hands-on aspects of storage, computation, and BigQuery and why the platform requires identity and access management.

In the next chapter, we will start working on Vertex AI.

Questions

1. What exactly is a cloud?
2. Describe the many securities that are available through the cloud.
3. What are the options available to us if one of our projects is accidentally deleted?
4. What is Google BigQuery? What are the advantages of BigQuery?

CHAPTER 2

Introduction to Vertex AI and AutoML Tabular

Introduction

In the last chapter, we learned about the Google Cloud Platform. This chapter introduces the Vertex AI component of GCP. Vertex AI's major characteristics and what's new about Vertex AI's predictions. Vertex AI's AutoML section and its kinds. Followed by step-by-step instructions for Athe utoML tabular.

Structure

In this chapter we will cover the following topics:

- Introduction to Vertex AI
- Key features
- Vertex AI prediction service
- Working with Vertex AI
- Vertex AI auto ML
- Creation of tabular datasets
- Model training
- Model evaluation

- Batch predictions
- Model deployment for online predictions
- Service account creation
- Serving online predictions
- Model un-deployment and end-point deletion
- Model deletion
- Dataset deletion

Objectives

By the end of this chapter, you will be able to create a dataset for AutoML for tabular data, initiate the training, evaluate the model, and take the trained model to deployment and getting predictions from the deployed model and removing the model from deployment.

Introduction to Vertex AI

Google announced the launch of Vertex AI in mid of 2021. Vertex AI is an integrated platform for artificial intelligence that brings together all of Google's cloud services under one umbrella. Using the pre-trained and custom model building tools provided by Vertex AI, customers are able to construct machine learning models or simply deploy and scale such models. Users that create ML solutions on Vertex AI may take use of AutoML and other sophisticated ML components to significantly boost their productivity and scalability. In addition to this, Google placed a strong emphasis on making the Vertex AI platform user-friendly for novices and efficient for seasoned professionals. Because of this, it is simple to train models and involves 80 percent less lines of code than other methods. Vertex AI will evolve into a one-stop shop for all things relating to AI that are associated with GCP.

Because of all of these factors, putting into practice MLOps is now simpler and more readily available to machine learning teams. This indicates that users do not need to have GCP memorized in order to locate all of the MLOps tools that is dispersed across the GCP Console.

Even though many of the GCP services are built on open-source frameworks they are fully managed by platform, example vertex pipelines are built on top of Kubeflow pipelines and Tensorflow Extended, vertex Metadata is built on ML Metadata. While the fact that there is a high degree of cross communication between each component of Vertex is precisely what makes it so valuable for data science teams.

Key features

Vertex AI makes it simple to integrate video, translation, and natural language processing with current applications by providing access to pre-trained APIs for video, vision, and others. Engineers now have the ability to train models that are tailored to fit the specific requirements of their company with minimum effort and knowledge required.

Integration of AI and data across the whole process is provided by Vertex AI, which is natively integrated with Dataproc, Dataflow, and BigQuery by way of the Vertex AI Workbench. Either you can develop and run machine learning models in BigQuery or you can export data from BigQuery to Vertex AI Workbench and run ML models from there. Both options are available to you.

Vertex AI offers a single unified user interface and API for all AI-related Google Cloud services, bringing together the whole machine learning process under a single roof. For instance, you may utilize AutoML inside Vertex AI to train and compare models, and then save them all in a centralized repository for model storage.

Vertex AI integrates with open-source frameworks that are widely used, such as PyTorch and TensorFlow. Additionally, it supports alternative tools through the usage of custom containers and integrates with all open-source frameworks.

Other new features are as following:

- **Training reduction server**: AI Training Reduction Server is a new Vertex functionality. This program increases multisystem distributed training on Nvidia GPUs, according to Google. "Distributed training" refers to the practice of training a system over several computers, GPUs, CPUs, or specialized chips to save time and resources.

  This reduces training time for large language workloads like BERT and permits cost parity between approaches. When the training period is shortened, data scientists may increase a model's prediction performance within a deployment timeframe. This is helpful for mission-critical operations. Data scientists need not be required to have expertise in infrastructure engineering or operations engineering.

- **Tabular workflows**: Tabular workflows comprise a glassbox and a regulated AutoML pipeline, which allows user to observe and understand model generation and deployment. Data scientists can now train models on 1 TB datasets without accuracy loss. Users may choose which process elements to automate and which to manually engineer.

Tabular Workflows may be integrated into Vertex AI pipelines. Google incorporated additional management algorithms TabNet, model feature selection, and model distillation for advanced research models.

- **Serverless spark**: Google has launched the Serverless Spark tool in addition to collaborations with Neo4j and Labelbox in order to hasten the deployment of machine learning models into production and better integrate data modeling capabilities directly into the environment of data science. These collaborations will help ML model developers deal with unstructured, structured, and graph data. This will help Google speed up the deployment of machine learning models into production. Data scientists will be able to launch a serverless spark session on their notebooks and interactively write code for structured data using Google Serverless Spark.

Vertex AI prediction service

A machine learning model is put to use after users are ready to start responding to requests from it. To ensure safety and scalability, it is essential to establish total seamlessness. Vertex AI has a low total cost of ownership since it is a fully managed service. The absence of over-provisioning of hardware is due to smooth auto-scaling.

The most economical hardware for a given model may be chosen by developers thanks to Prediction Service and a variety of VM and GPU kinds. Contrary to open source, it also has several proprietary backend upgrades that further reduce expenses. Prebuilt components for request-response logging in BigQuery are included into Stackdriver's out-of-the-box logging capability to enable routine deployment of models from pipelines.

Built-in compliance and security: Users may utilize their own secure perimeter to install models. Your endpoints may be accessed by GCP's PCSA (pre-closure safety analysis) integration control tool, and your data is always secure.

Working with Vertex AI

With a good theoretical understanding of Vertex AI and its key features, now let's see how Vertex AI looks on the platform followed by working on AutoML.

Follow below steps to open Vertex AI on the platform:

Step 1:

1. Vertex AI can be found under Artificial Intelligence. **Vertex AI → Dashboard**.

2. Alternatively, Users can type Vertex AI in the search box as seen in *Figure 2.1*:

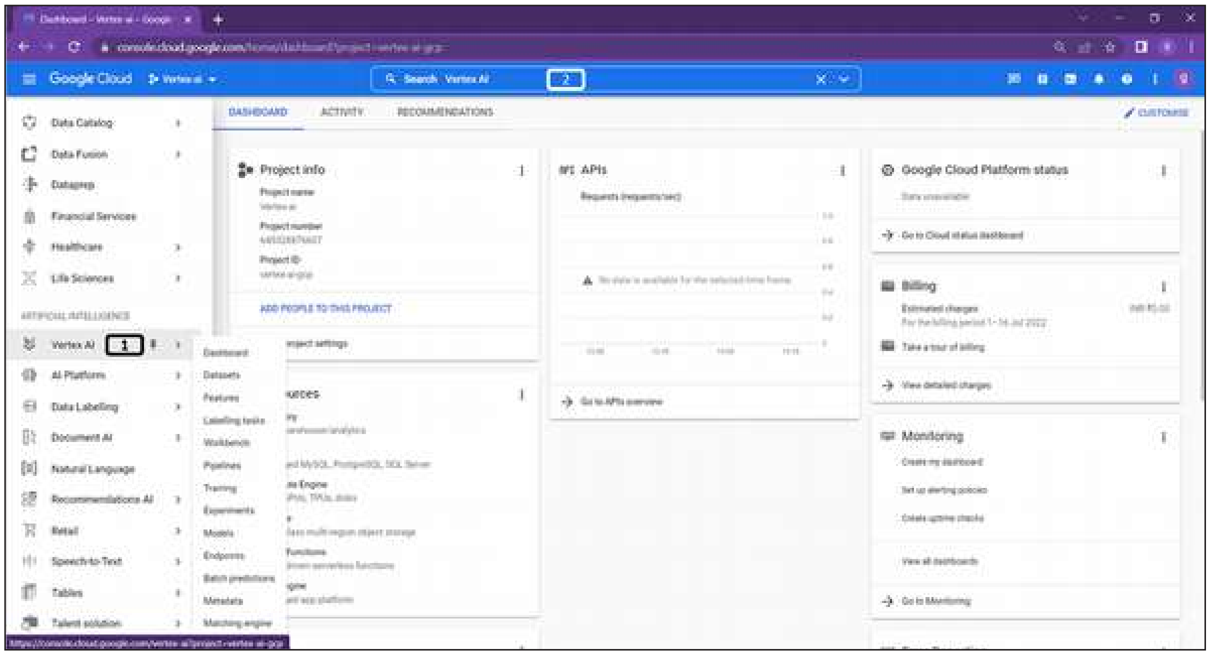


***Figure 2.1**: To open Vertex AI*

Step 2: Landing page of Vertex AI

The components on the landing page of Vertex AI can be seen in *Figure 2.2*:

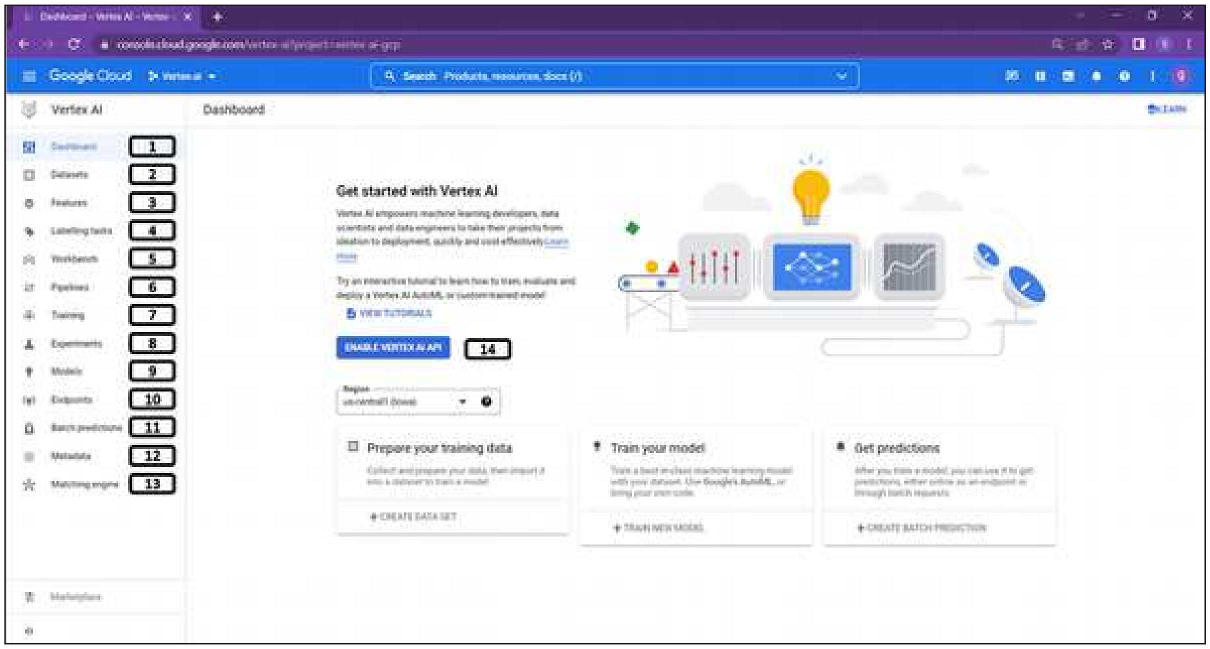


***Figure 2.2**: Landing page of Vertex AI*

Brief description of components on the landing page of Vertex AI are listed below:

1. **Vertex AI dashboard**: Landing page of Vertex AI.
2. **Datasets**: Datasets may be either organized or unstructured. It has controlled metadata, such as annotations (at present only Cloud Storage and BigQuery are supported).
3. **Feature Store**: A centralized repository for organizing, storing, and providing ML features will be provided via the feature store. This resolves the issues of redundant feature development and underuse of high-quality features. A centralized repository makes it apparent who is responsible for creating, storing, and calling features. Feature stores provide high-throughput batch requests and low-latency internet requests.
4. **Labels**: Labelling the data would be the first phase in the machine learning development cycle. Human labellers will assist in completing this labelling task.
5. **Workbench**: Vertex AI Workbench is a single development platform for the whole data science process. Vertex AI Workbench offers managed notebooks (Google managed environment, for easier operations) and user-managed notebooks (for users that need complete control over their environment).
6. **Pipelines**: By coordinating your machine learning process in a serverless fashion and preserving your workflow's artifacts using Vertex ML Metadata, Vertex AI Pipelines enables you to automate, monitor, and regulate your ML systems.
7. **Training**: Vertex AI's main model training approach is comprised of training pipelines.
8. **Experiments**: When creating a model for a problem, it is important to determine which model is suitable for that specific use case. Vertex AI Experiments gives you the ability to monitor, analyze, compare, and search across several ML Frameworks and training settings, including TensorFlow, PyTorch, and scikit-learn.
9. **Models**: ML models contains information that were either directly loaded or created through a training pipeline. A machine learning solution termed a model serves as a container for versions, or real implementations of models.
10. **Endpoints**: Predictions are served by deploying the trained model to an endpoint. Disambiguation is carried out depending on the request, and it may contain one or more models and variants of those models.
11. **Batch predictions**: For big datasets that would take too long to process using an online prediction method, Vertex AI Batch Prediction is an alternative. It

offers an effective, serverless, scalable solution for asynchronous responses. Users don't need to deploy the model to an endpoint in order to get batch predictions from the model resource.

12. **Metadata**: You may query metadata to help assess, debug, and audit the performance of your machine learning system and the artifacts that it generates. Metadata also allows you to record the metadata and artifacts that are created by your ML system.
13. **Matching engine**: Matching Engine offers the most advanced vector-similarity matching (also known as approximate closest neighbour) service in the market, as well as market-leading techniques to train semantic embeddings for similarity-matching use cases.

1. Enable the Vertex AI APIs to continue working.

This book covers various entities of Vertex AI with practical examples. We will start with AutoML of Vertex AI.

Vertex AI AutoML

The collection of machine learning solutions known as AutoML, makes it possible for software developers with less experience in machine learning to train high-quality models that are tailored specifically to the requirements of their company. Transfer learning and neural architecture search technologies developed by Google are essential for the operation of this system.

This environment is completely managed, which means that all of the operational procedures, infrastructure, and model management are taken care of for you. Using a simple graphical user interface, users may quickly train, test, and deploy machine learning models to address issues involving tabular data, vision, translation, and natural language. These tasks can be completed in as little as a few minutes. Vertex AI is by a wide margin the most comprehensive solution of automated machine learning currently available on the market.

Datasets are the first step of the machine learning lifecycle, and to get started you need data. Vertex AI currently supports managed datasets for four data types—tabular, image, text, and videos. Listed below are the tasks supported by AutoML for each of the data types:

- Tabular data:
  - Classification
  - Regression
  - Forecasting

- Image data:
    - o Classification (single label & multi-label)
    - o Image segmentation
    - o Image object detection
- Text data:
    - o Classification (single label & multi-label)
    - o Text entity extraction
    - o Text sentiment analysis
- Video data:
    - o Action recognition
    - o Classification
    - o Object tracking

Benefits of datasets:

- Manage your datasets in a central location.
- Create labels and other annotation sets with ease.
- Maintain a history of the iterative development and governance frameworks.
- By utilizing the same datasets to train AutoML and custom models, lets users to compare model performance.
- Create data visualizations and statistics.
- Divide data into training, test, and validation sets automatically.

Let us start with tabular data.

Creation of tabular datasets

Personal key indicators of heart disease data from Kaggle is used, and it is a classification task. Data can be downloaded from the below link.

https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease

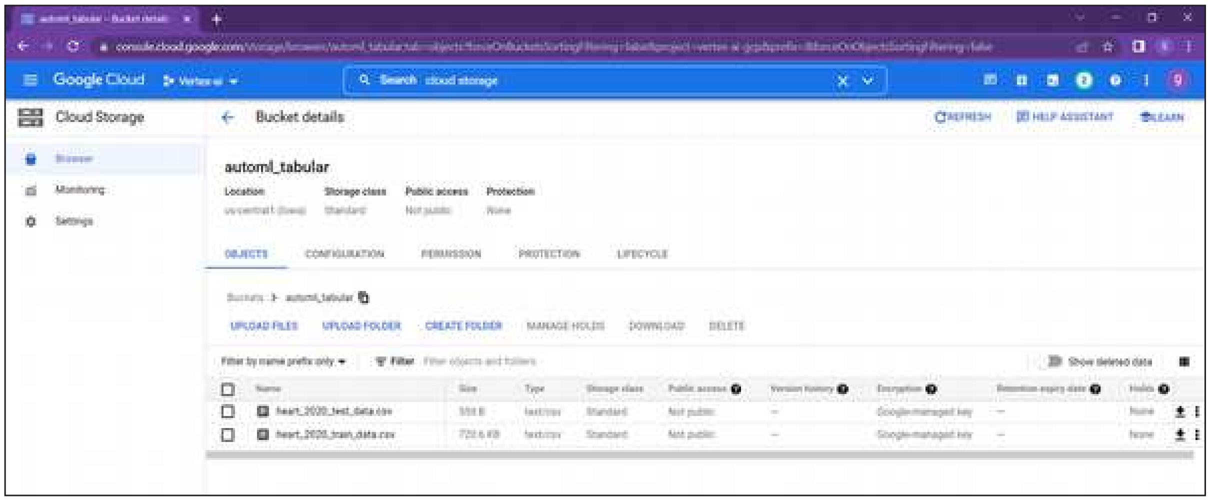
`automl_tabular` bucket is created under **`us-centra1`** (single region) as shown in the *Figure 2.3*. Data contains 300K+ records, however we are just using 10K for our practical. **`Train_data.csv`** is used for training and **`test_data.csv`** will be used for batch predictions (contains 5 random records excluding the target column).

Let us understand the data at a high level. Data contains about 18 columns. The **`HeartDisease`** column is the target column (binary) containing values of **Yes** and **No**.

Categorical columns: Smoking, alcoholdrinking, stroke, diffwalking, sex, agecategory, race, diabetic, physicalactivity, genhealth, asthma, kidneydisease, skincancer

Continuous columns: BMI, PhysicalHealth, MentalHealth, SleepTime

Figure 2.3 shows the data uploaded to cloud storage bucket:



Name	Size	Type	Storage class	Public access	Version history	Encryption	Retention expiry date	Holds
heart_2020_test_data.csv	559 B	text/csv	Standard	Not public	—	Google-managed key	—	None
heart_2020_train_data.csv	720.6 KB	text/csv	Standard	Not public	—	Google-managed key	—	None

***Figure 2.3**: Data uploaded to cloud storage bucket*

Follow the steps below to create a dataset:

Step 1: Select **Dataset** tab.

The dataset tab can be seen in *Figure 2.4*:

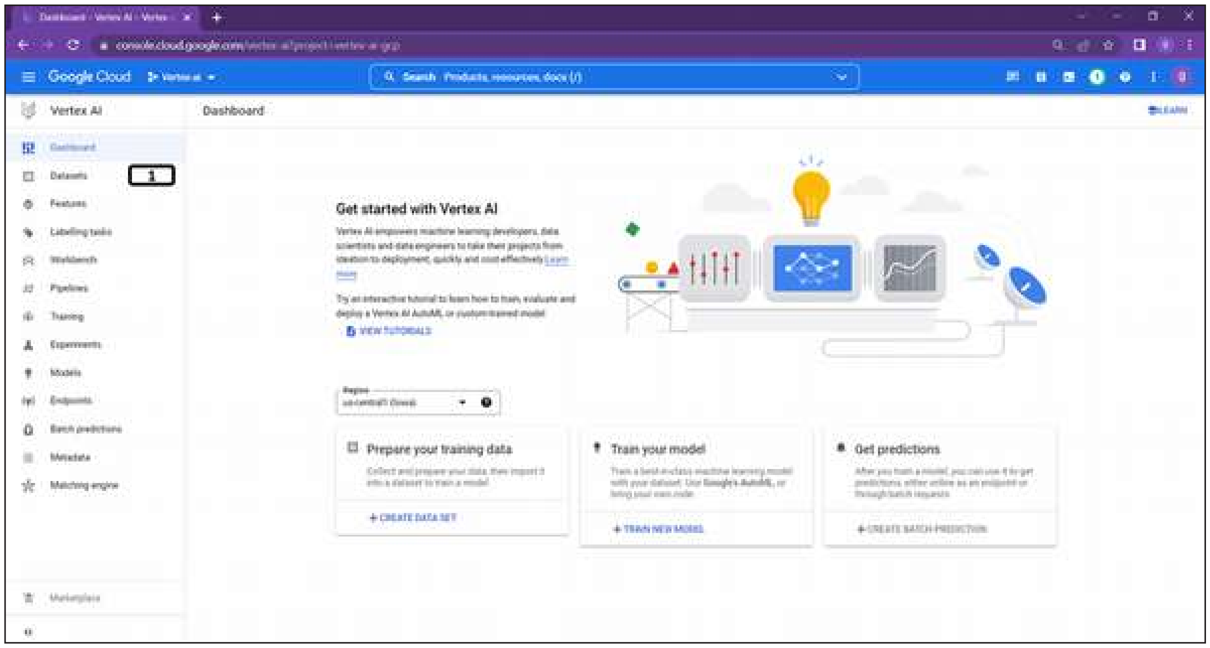
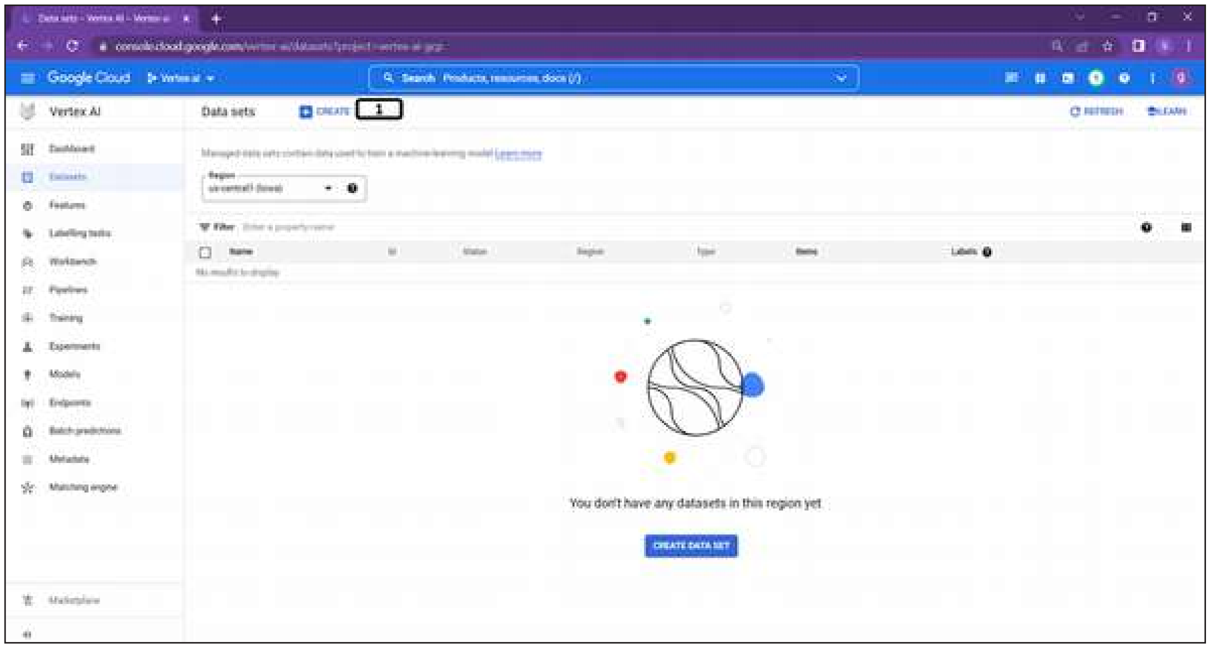


***Figure 2.4**: Selection of dataset tab*

Step 2: Creation of dataset: Click on **Create**.

Landing page of dataset is as shown in the *Figure 2.5*:



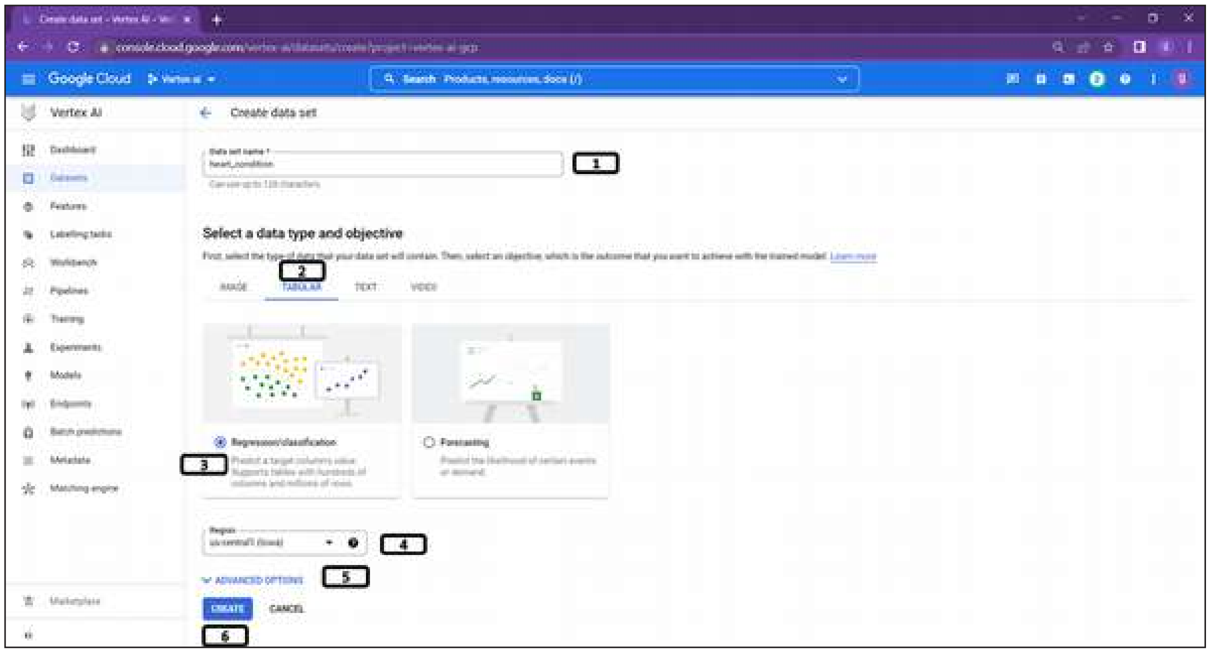
Region: us-central1 (Iowa)

You don't have any datasets in this region yet

Figure 2.5*: Dataset creation*

1. Details for dataset creation:

The details can be seen in *Figure 2.6*:



Data set name *: heart_condition

Region: us-central1 (Iowa)

Figure 2.6*: Tabular dataset*

1. Provide a name for the dataset.
2. Select **Tabular** in options.
3. Select classification (Since the target variable in the dataset is categorical).
4. Select the region to be **us-central1**.
5. Advanced options - For encryption.
6. Click **Create**.

Step 3: Select data for the dataset created

The screen in *Figure 2.7* will be automatically displayed once the dataset is created:

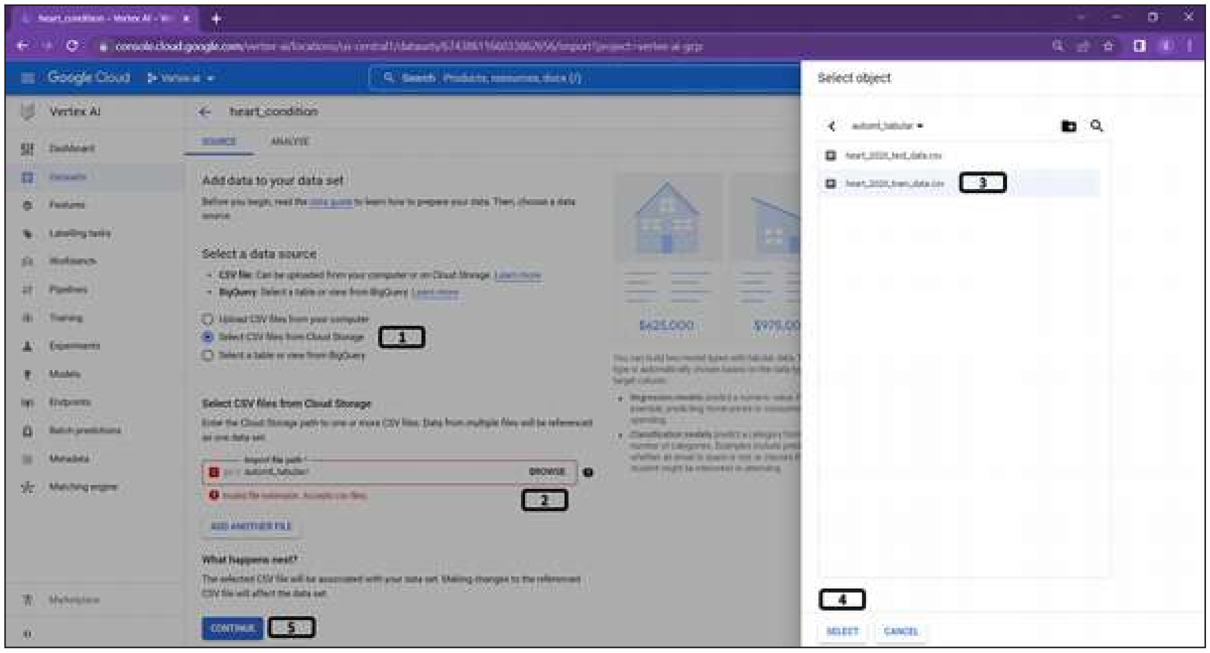
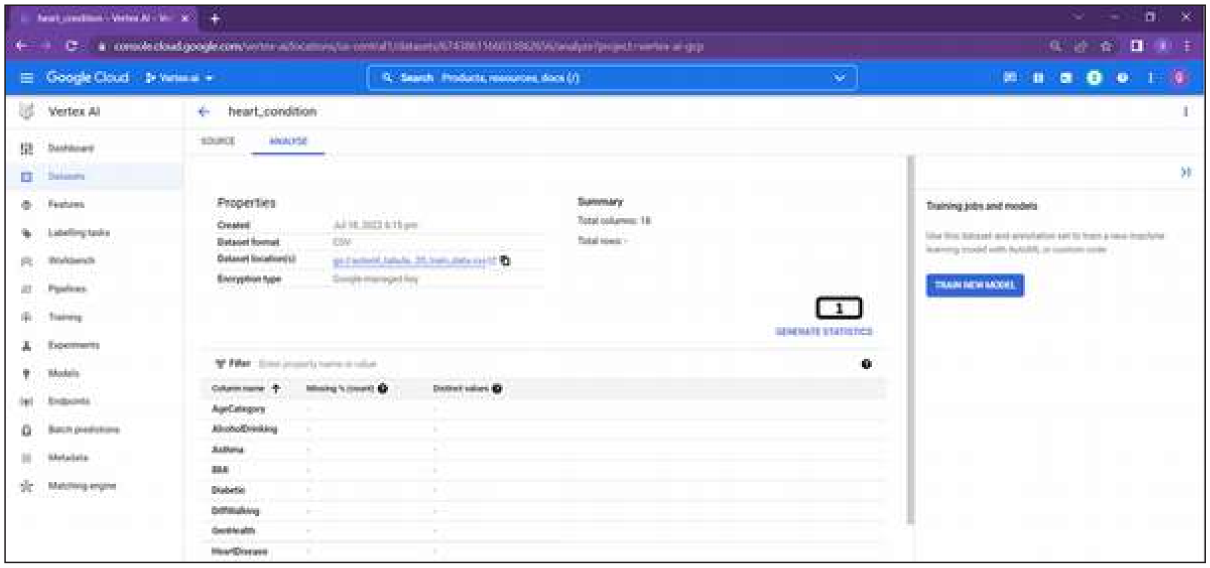


***Figure 2.7**: Data selection for dataset creation*

1. Select CSV file from cloud storage (select the `train.csv` and dataset can also be uploaded directly from local machine or from BigQuery).
2. Click **Browse** (Users can add multiple files).
3. Select the file needed to upload to the dataset.
4. Click **Select**.
5. Click **Continue**.
6. **Generation of statistics**

Step 3: Generation of statistics

Users will be taken to analyse page once the dataset is created, as shown in *Figure 2.8*:



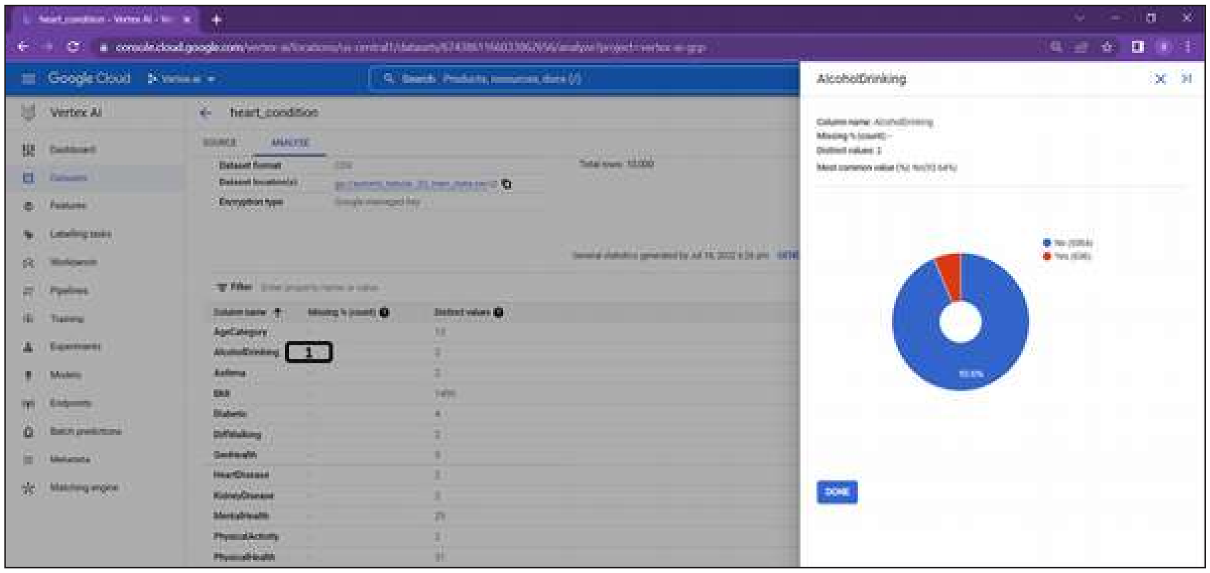
Column name	Missing % (count)	Distinct values
AgeCategory	-	-
AlcoholDrinking	-	-
Asthma	-	-
BMI	-	-
Diabetic	-	-
DiffWalking	-	-
GenHealth	-	-
HeartDisease	-	-

Figure 2.8*: Statistics generation of data*

1. Click on **Generate statistics** (Generate statistics helps AutoML to decide on the transformations to be applied on to the data during model training).

Step 3: Results of generate statistics

Generate statistics populates the details of missing value and the unique values for all the columns as shown in the *Figure 2.9*:



Column name	Missing % (count)	Distinct values
AgeCategory	-	13
AlcoholDrinking	-	2
Asthma	-	2
BMI	-	1490
Diabetic	-	4
DiffWalking	-	2
GenHealth	-	5
HeartDisease	-	2
KidneyDisease	-	2
MentalHealth	-	29
PhysicalActivity	-	2
PhysicalHealth	-	31

Figure 2.9*: Statistics generated*

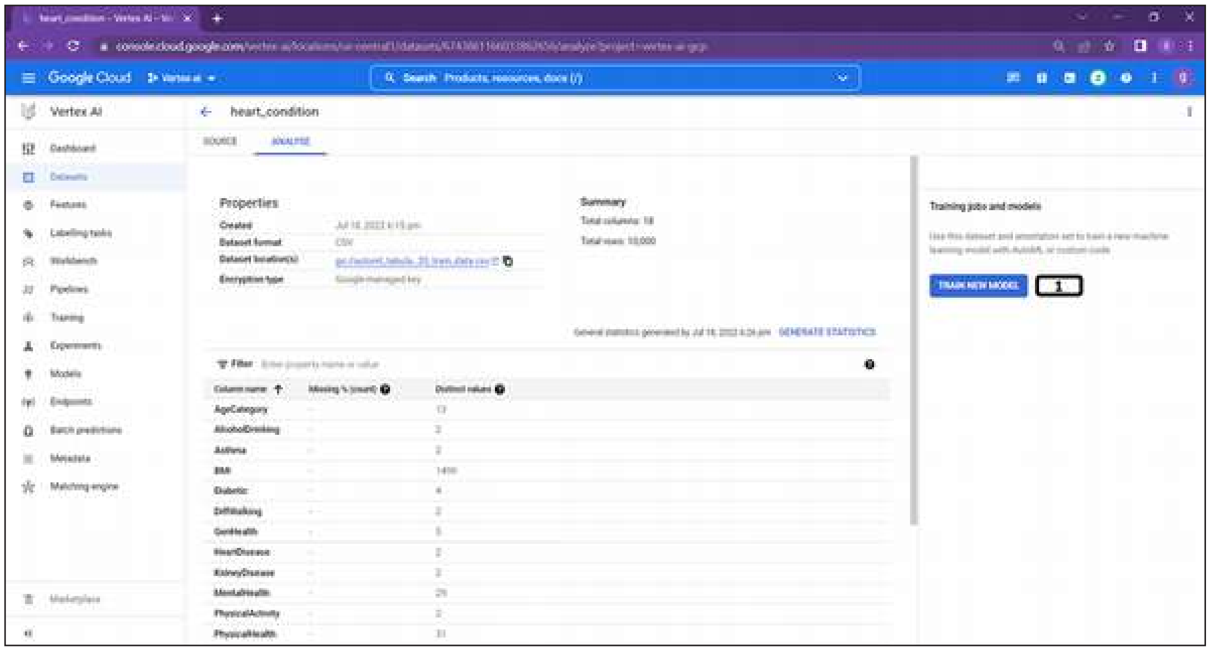
1. In addition to this, users can view the distribution for all the columns, by clicking on the columns.

Model training

Once the dataset is created, we can initiate the model training. In case model training is using AutoML, users do not need to worry about the algorithm selection. Follow below steps to start model training:

Step 1: Model training preparation

After checking the statistics of the data, users can click on done, as shown in the *Figure 2.9* to navigate to training phase as shown in *Figure 2.10*:



Properties

Created	Jul 18, 2023 4:15 pm
Dataset format	CSV
Dataset location(s)	[illegible]
Encryption type	Google-managed key

Summary
Total columns: 18
Total rows: 10,000

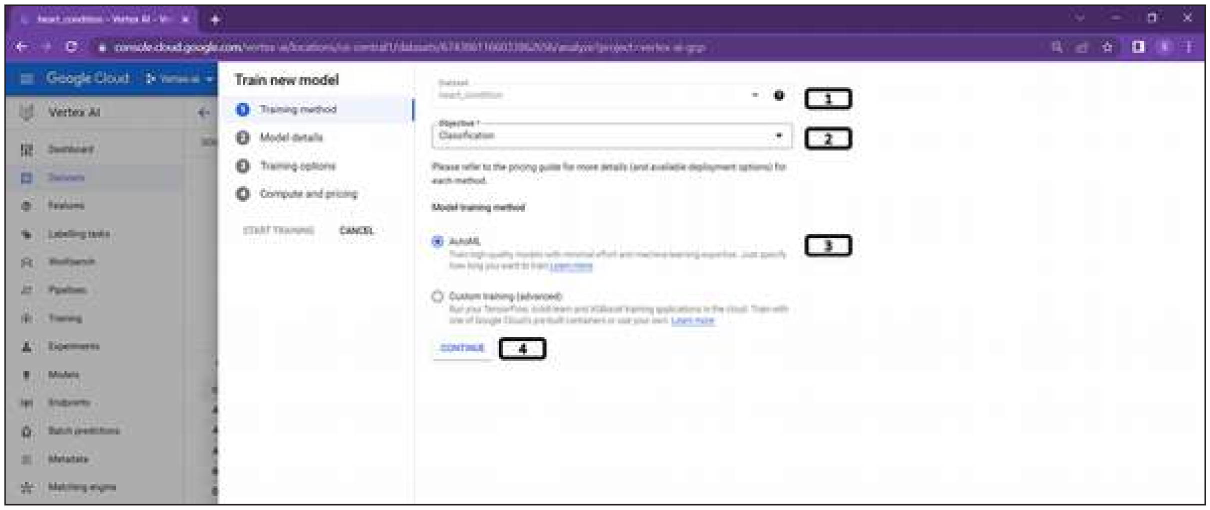
Column name	Missing % (count)	Distinct values
AgeCategory	-	13
AlcoholDrinking	-	2
Asthma	-	2
BMI	-	1450
Diabetic	-	4
DiffWalking	-	2
GenHealth	-	5
HeartDisease	-	2
KidneyDisease	-	2
MentalHealth	-	29
PhysicalActivity	-	2
PhysicalHealth	-	31

***Figure 2.10**: Tabular AutoML model training initiation*

1. Click on train new model.

Step 2: Training method selection.

First step in the model training is to choose the dataset and objective as shown in the *Figure 2.11*:



Dataset: heart_condition

Objective *: Classification

Please refer to the pricing guide for more details (and available deployment options) for each method.

Model training method

(●) AutoML

Train high-quality models with minimal effort and machine learning expertise. Just specify how long you want to train. Learn more

( ) Custom training (advanced)

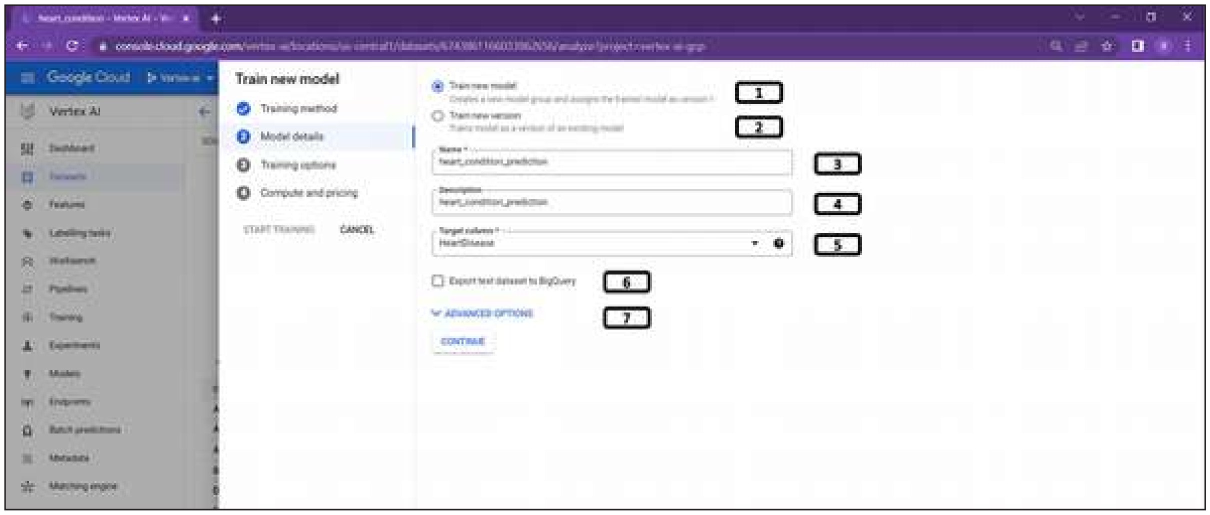
Run your TensorFlow, scikit-learn and XGBoost training applications in the cloud. Train with one of Google Cloud's prebuilt containers or use your own. Learn more

Figure 2.11: *Training method selection*

1. Select the dataset created.
2. Select the objective to be classification (since the target column is categorical).
3. Select **AutoML**.
4. Click **Continue**.

Step 3: Model details selection

Model details allows users choose between training new model or new version of the trained model as shown in *Figure 2.12*:



(●) Train new model

Creates a new model group and assigns the trained model as version 1

( ) Train new version

Trains model as a version of an existing model

Name *: heart_condition_prediction

Description: heart_condition_prediction

Target column *: HeartDisease

☐ Export test dataset to BigQuery

Figure 2.12: *Model selection details*

1. Select **Train new model**.
2. If a model is already trained and the users want to retrain the model due to change in the data, then this option lets them train model with new version.
3. Provide **Name** to the model.
4. Provide a **Description** to the model.
5. Select the **Target column**.
6. Test split of the dataset can be exported to BigQuery if the option is selected.
7. Clicking on **Advanced Options** lets users choose the data split.

Step 4: Data split for the training

Clicking on advanced options will give more options on the data split as shown in *Figure 2.13*:

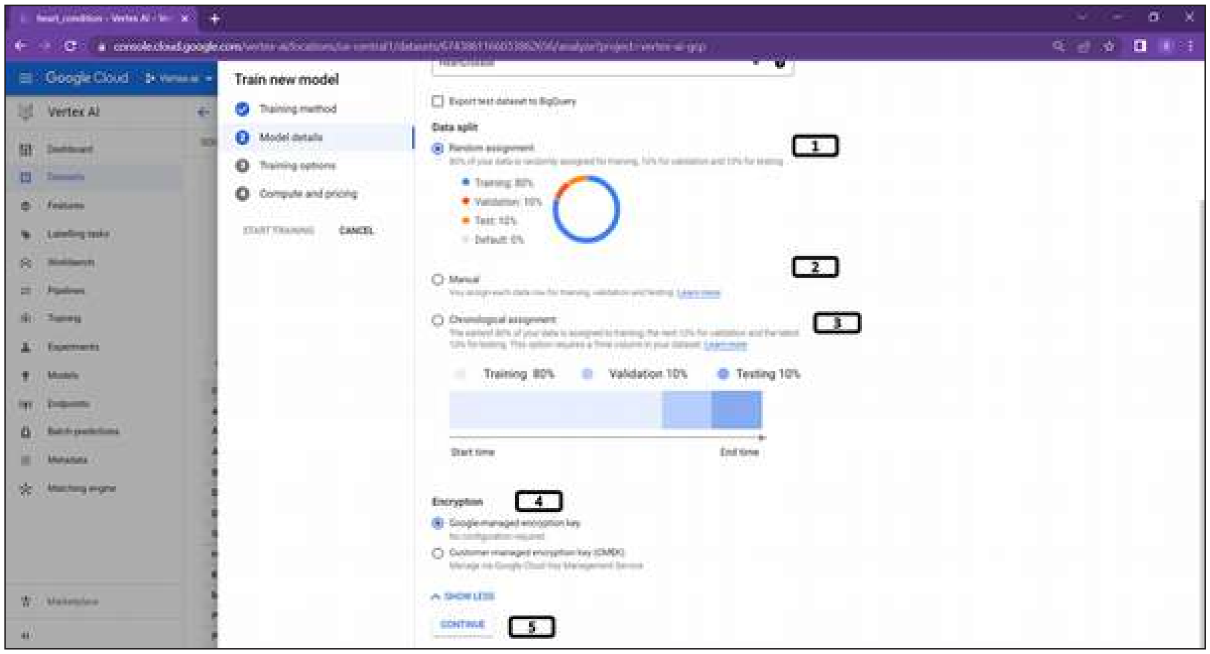


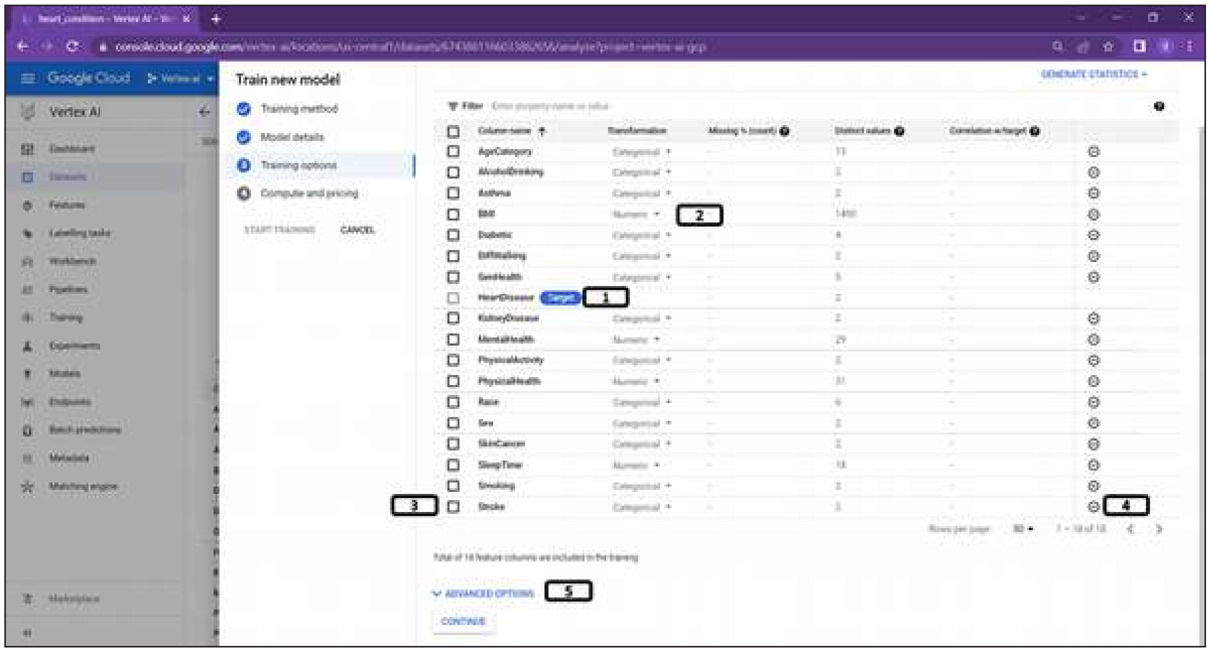
Figure 2.13: *Model selection details (data splitting)*

1. 80% of the data is used for training and 10% for validation and test respectively. And this split happens at random.
2. Users can also provide an additional column mentioning about the split for each record in the dataset.
3. If Chronological assignment is chosen, first 80% of the data will be used for training, next 10% is used for validation and the last 10% of the data is used for test.

4. Advanced options - For encryption.
5. Click **Continue**.

Step 5: Training options

Transformation options are provided for users as shown in *Figure 2.14*:



Column name	Transformation	Missing % (count)	Distinct values	Correlation w/target
AgeCategory	Categorical	-	13	-
AlcoholDrinking	Categorical	-	2	-
Asthma	Categorical	-	2	-
BMI	Numeric	-	1400	-
Diabetic	Categorical	-	4	-
DiffWalking	Categorical	-	2	-
GenHealth	Categorical	-	5	-
HeartDisease Target		-	2	-
KidneyDisease	Categorical	-	2	-
MentalHealth	Numeric	-	29	-
PhysicalActivity	Categorical	-	2	-
PhysicalHealth	Numeric	-	31	-
Race	Categorical	-	6	-
Sex	Categorical	-	2	-
SkinCancer	Categorical	-	2	-
SleepTime	Numeric	-	18	-
Smoking	Categorical	-	2	-
Stroke	Categorical	-	2	-

Total of 18 feature columns are included in the training

***Figure 2.14**: Training options (data transformation)*

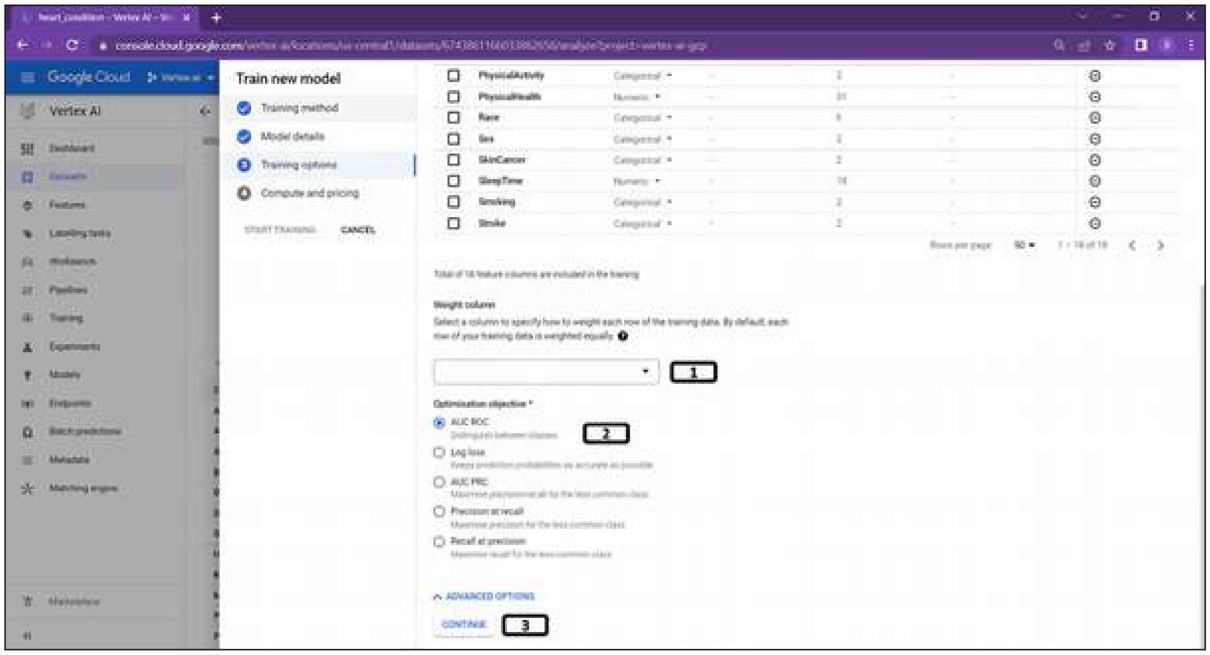
1. Target column is assigned.
2. AutoML does not always choose the right transformation to be applied. It is good to check manually and change the transformation type. Click on the arrow to change the transformation type.
3. If any column needs to be excluded from training, column needs to be selected.
4. Once the column is selected, click on "**-**".

(In our use case, we are considering all the columns for training, and there is no need to follow steps 3 & 4).

5. Click **Advanced options**.

Step 6: Select optimization objectives

Users will be provided with additional training settings options as shown in *Figure 2.15*:


Train new model

- Training method
- Model details
- Training options
- Compute and pricing

START TRAINING CANCEL

☐	PhysicalActivity	Categorical	2	
☐	PhysicalHealth	Numeric	31	
☐	Race	Categorical	6	
☐	Sex	Categorical	2	
☐	SkinCancer	Categorical	2	
☐	SleepTime	Numeric	19	
☐	Smoking	Categorical	2	
☐	Stroke	Categorical	2	

Total of 18 feature columns are included in the training

Weight column

Select a column to specify how to weight each row of the training data. By default, each row of your training data is weighted equally.

Optimization objective *

- [✓ AUC ROC] Distinguish between classes
- Log loss Keeps prediction probabilities as accurate as possible
- AUC PRC Maximize precision-recall for the less common class
- Precision at recall Maximize precision for the less common class
- Recall at precision Maximize recall for the less common class

ADVANCED OPTIONS

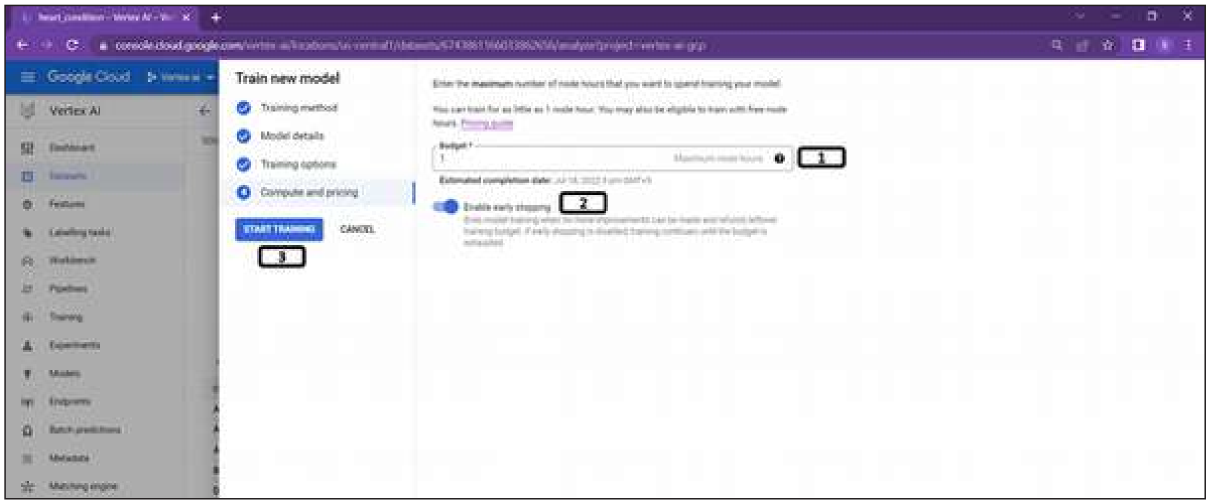
CONTINUE

Figure 2.15: *Training options (optimization objective)*

1. By default, AutoML considers all the columns to be equally important, however users are given an option to add more importance to the required columns.
2. Users are given options to choose the optimization objectives based on the use cases.
3. Click **Continue**.

Step 7: Compute and pricing

Final step for model training is as shown in the *Figure 2.16*:


Train new model

- Training method
- Model details
- Training options
- Compute and pricing

START TRAINING CANCEL

Enter the maximum number of node hours that you want to spend training your model.

You can train for as little as 1 node hour. You may also be eligible to train with free node hours. Pricing guide

Budget *: 1 Maximum node hours

Estimated completion date: Jul 18, 2022 3 pm GMT+5

Enable early stopping

Ends model training when no more improvements can be made and refunds leftover training budget. If early stopping is disabled, training continues until the budget is exhausted.

Figure 2.16: *Training initiation*

1. Users can specify the maximum node hours for training, the minimum value for tabular dataset is 1. A node hour represents the time a virtual machine spends on training the model.
2. Early stopping can be enabled to stop the training if model cannot be trained any further.
3. Click on **Start training**.

Tabular data structure recommended for AutoML (classification and regression):

- Data must be 100 GB or smaller
- Number columns must be greater than 2 and not more than 1000
- Number rows must be greater than 1000 and less than 100000000

Model evaluation

The method through which we measure the quality of a model's predictions is called model evaluation. To achieve this, AutoML evaluates the performance of the trained model on a test split of dataset, 10% of the training data will be used for evaluation (refer *Figure 2.13*).

Step 1: Checking model training

Status of training can be seen in *Figure 2.17*:

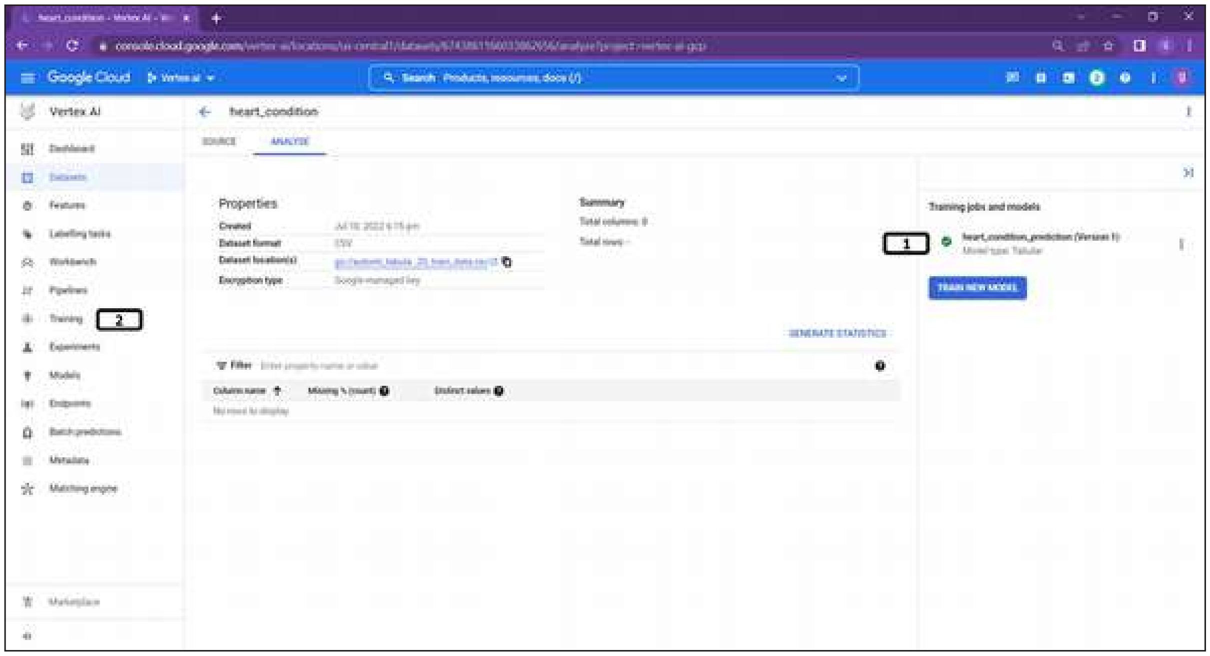
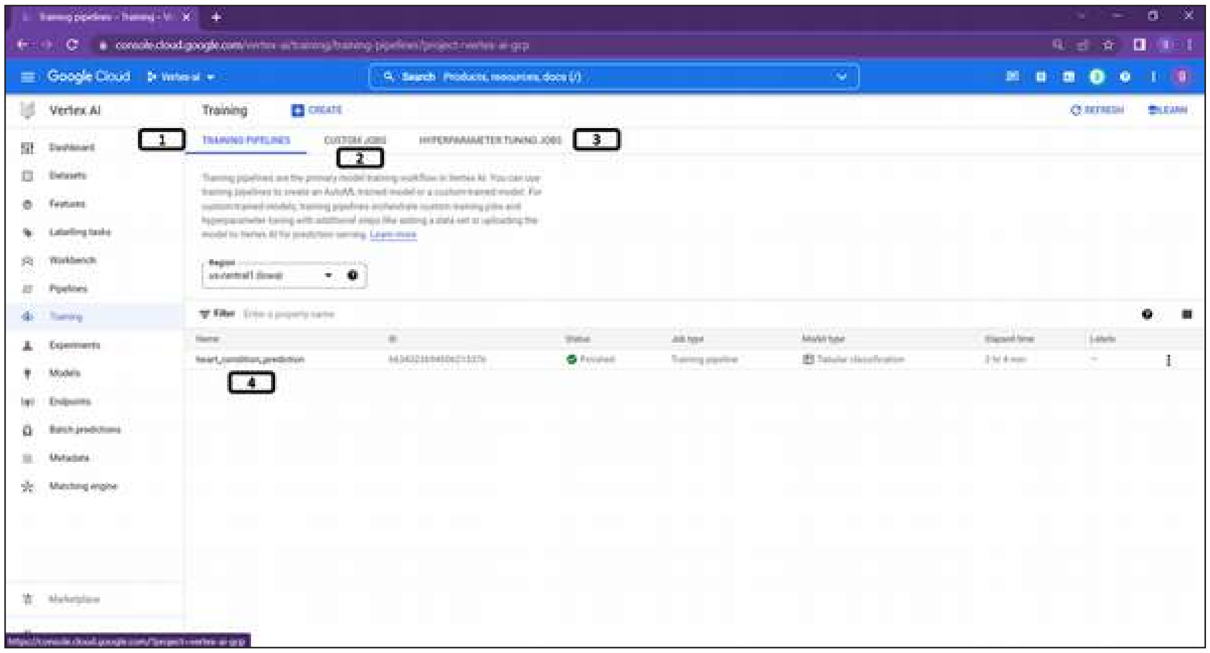


***Figure 2.17**: Training complete*

1. Once the model training is completed, it will be highlighted with a green tick.
2. Model training will also be listed in the training section of Vertex AI (click on it).

Step 2: Model training results

Training jobs will be listed under training section as shown in *Figure 2.18*:



Training pipelines are the primary model training workflow in Vertex AI. You can use training pipelines to create an AutoML trained model or a custom-trained model. For custom-trained models, training pipelines orchestrate custom training jobs and hyperparameter tuning with additional steps like adding a data set or uploading the model to Vertex AI for prediction serving. Learn more

Region: us-central1 (Iowa)

Name	ID	Status	Job type	Model type	Elapsed time	Labels
heart_condition_prediction	[illegible]	Finished	Training pipeline	Tabular classification	2 hr 4 min	-

***Figure 2.18**: Training job listed under training job*

1. Training pipelines will list training jobs of AutoML.
2. Custom Jobs will list training jobs of custom models.
3. Hyperparameter tuning jobs will list training jobs of custom models with hyperparameter tuning.
4. AutoML model trained in previous steps (click on it). Trained model will also be listed under model section of Vertex AI.

Step 3: Model evaluation

Clicking on the model will navigate to model section as shown in *Figure 2.19*:

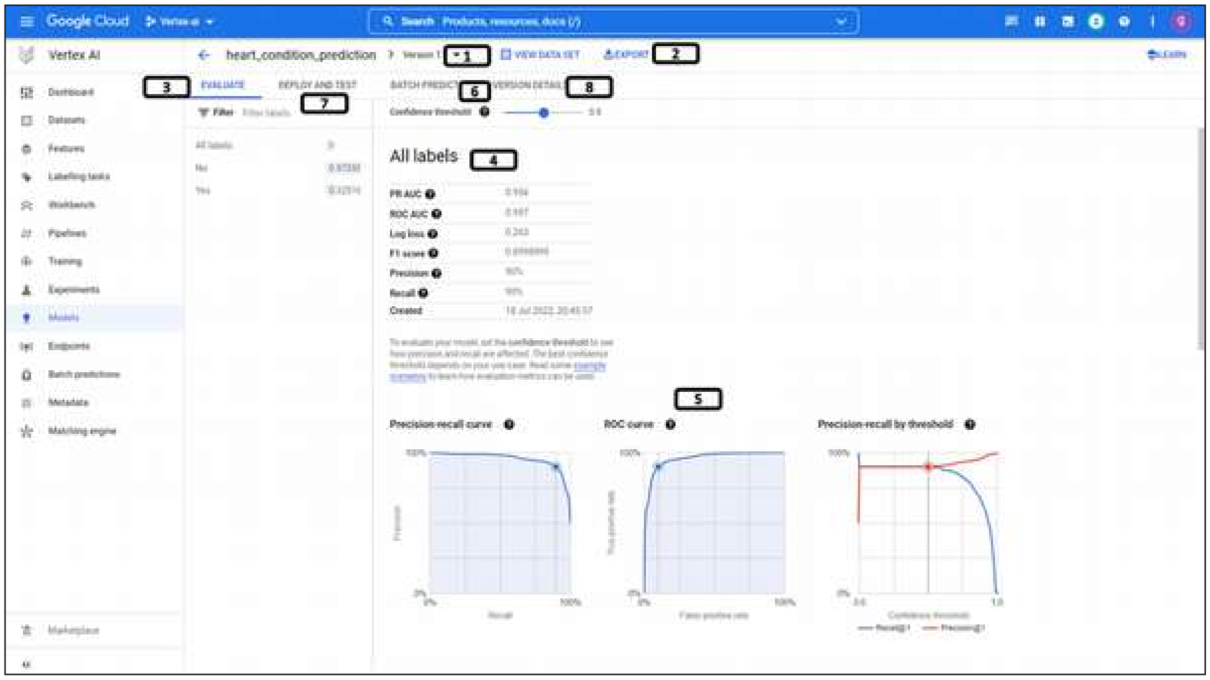


Figure 2.19: *Trained model evaluation*

1. When a model is trained, it will be the first version of the model. When multiple models are trained it provides options to select the specific version of the model.
2. Trained model can be exported as a Tensorflow package, and model can be exported to cloud storage.
3. Evaluate provides details about the class information about the target variable. Users can select any specific class to check the evaluation metrics for each of the classes separately.
4. Different evaluation metrics are listed along with the visual information about few metrics. Confidence threshold can be varied to inspect the graphs at different levels of confidence.
5. Graphical representation of the evaluation metrics (varies as per the confidence threshold).
6. Batch predict provides options for batch prediction.
7. Deploy and test provides options for the online prediction.
8. Version details provides various details of the model version (click on Version details).

Step 4: Confusion matrix and feature importance

Scrolling down will have information about other metrices as shown in *Figure 2.20*:

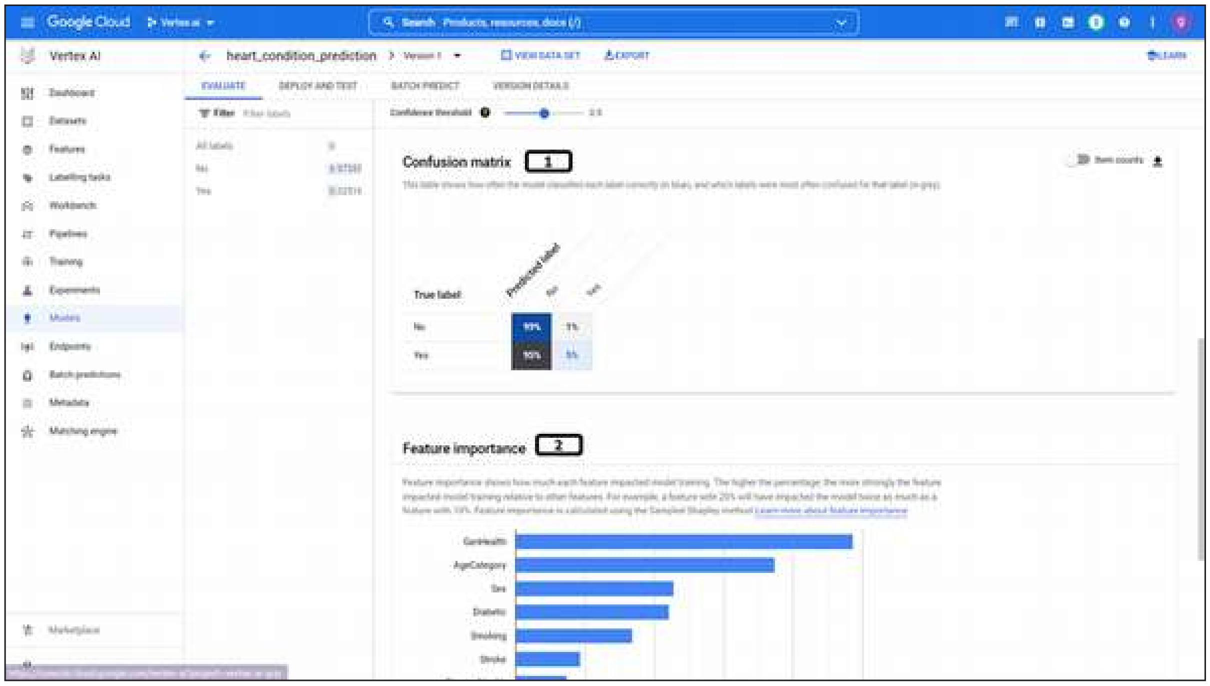
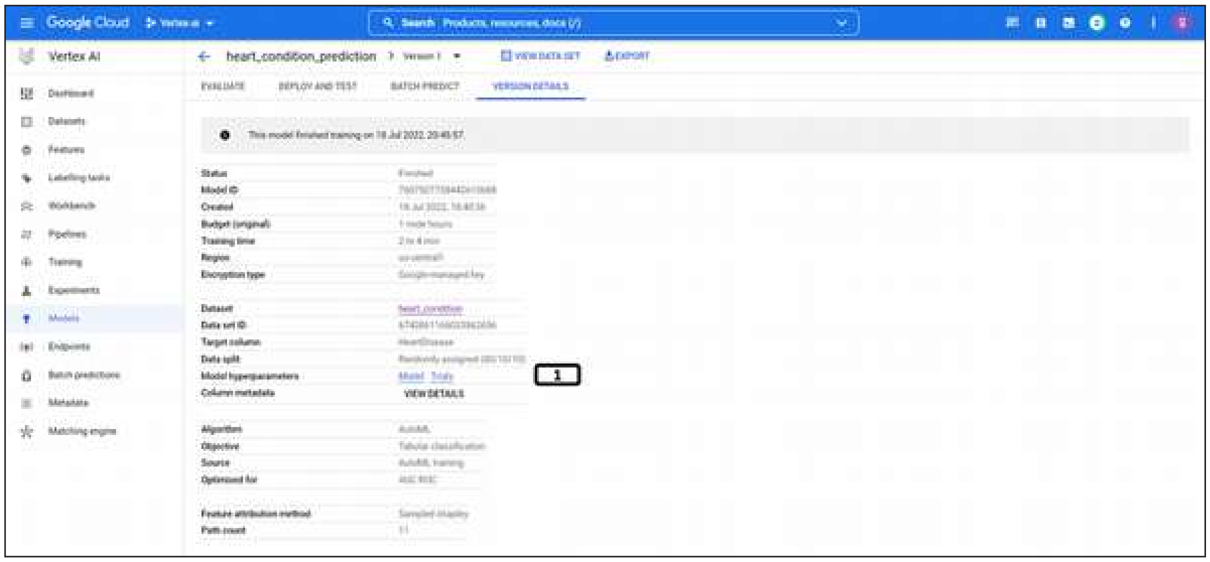


Figure 2.20: Trained model evaluation

1. Confusion matrix, click on the item counts to get the matrix with actual counts.
2. variable importance information (variable importance is calculated using sampled Shapley method).

Step 5: Model version details

More details on the model will be displayed as shown in *Figure 2.21*:



This model finished training on 18 Jul 2022, 20:45:57

Status	Finished
Model ID	[illegible]
Created	18 Jul 2022, 18:40:36
Budget (original)	1 node hours
Training time	2 hr 4 min
Region	us-central1
Encryption type	Google-managed key
Dataset	heart_condition
Data set ID	[illegible]
Target column	HeartDisease
Data split	Randomly assigned (80/10/10)
Model hyperparameters	Model Trials
Column metadata	VIEW DETAILS
Algorithm	AutoML
Objective	Tabular classification
Source	AutoML training
Optimized for	AUC ROC
Feature attribution method	Sampled shapley
Path count	11

Figure 2.21: Trained model details

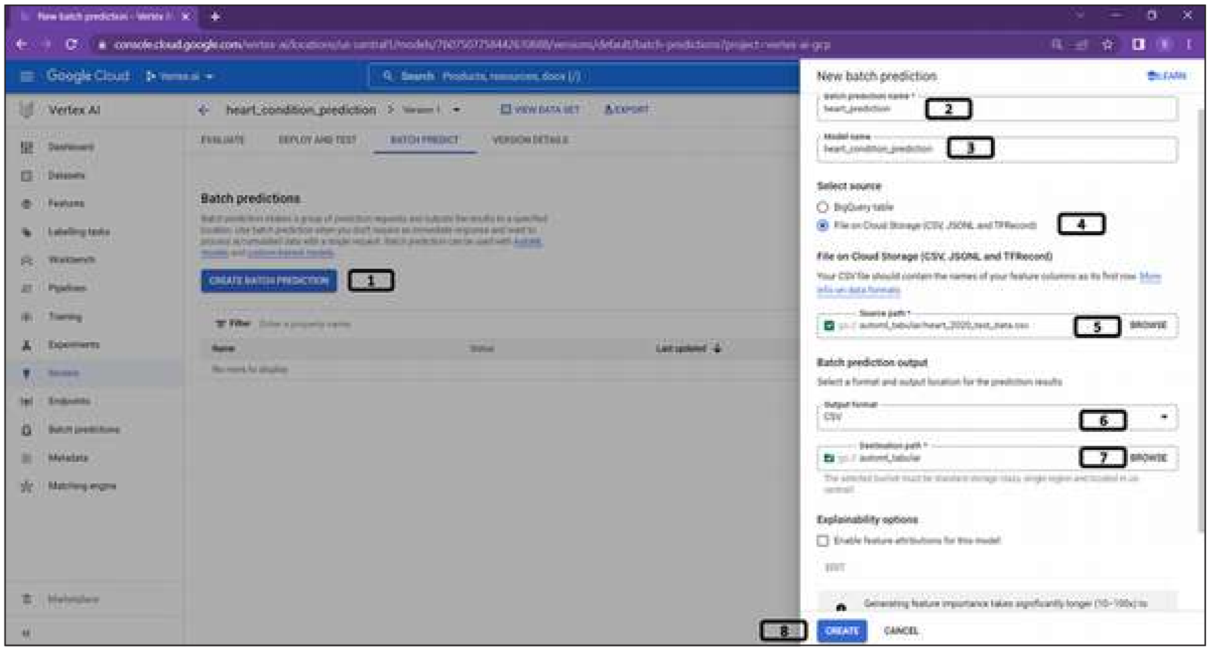
- Various details of the model are listed.
- Detailed log messages are available in the models and trials.

Batch predictions

A batch prediction request is asynchronous. Designed to manage a large number of instances in a single task and to execute more sophisticated models. Per request, may handle one or more instances. Predictions are written to output files at a place you choose in Cloud Storage. Follow below steps for batch predictions.:

Step 1: Creating batch predictions

While creating batch predictions, users need to enter all details as shown in *Figure 2.22*:


New batch prediction

Batch prediction name *: heart_prediction

Model name: heart_condition_prediction

Select source
- BigQuery table
- File on Cloud Storage (CSV, JSONL and TFRecord)

File on Cloud Storage (CSV, JSONL and TFRecord)

Your CSV file should contain the names of your feature columns as its first row. More info on data formats

Source path *: gs:// automl_tabular/heart_2020_test_data.csv

Batch prediction output

Select a format and output location for the prediction results

Output format: CSV

Destination path *: gs:// automl_tabular

The selected bucket must be standard storage class, single region and located in us-central1

Explainability options

Enable feature attributions for this model

Generating feature importance takes significantly longer (10–100x) to

Figure 2.22: *Batch predictions of the model*

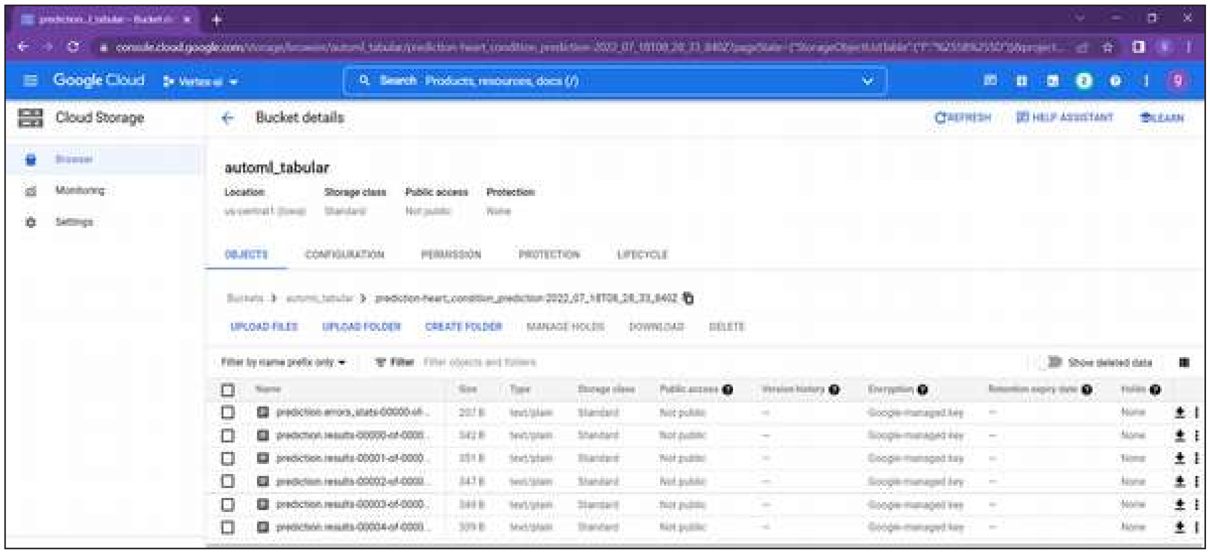
1. Click on **CREATE BATCH PREDICTION**.
2. Provide a name for the prediction.
3. Select model for prediction.
4. Data used for prediction can be sourced from cloud storage bucket or from bigquery.
5. Since we have chosen cloud storage select the file from the bucket.
6. Select the Output format, outputs can be directly dumped into bigquery table or it can be stored in cloud storage in csv/jsonl/TFrecord format.

7. Provide the output location.
8. Click on **Create**.

Batch predictions may take a few minutes to a few hours depending on the number of records. Importantly, users do not need to deploy the model for batch predictions.

Step 2: Output of batch predictions

After batch predictions, output will be stored to cloud storage as shown in *Figure 2.23*:


Bucket details

automl_tabular

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

Buckets › automl_tabular › prediction-heart_condition_prediction-2022_07_18T08_28_33_840Z

Name	Size	Type	Storage class	Public access	Version history	Encryption	Retention expiry date	Holds
prediction.errors_stats-00000-of-...	237 B	text/plain	Standard	Not public	—	Google-managed key	—	None
prediction.results-00000-of-0000...	342 B	text/plain	Standard	Not public	—	Google-managed key	—	None
prediction.results-00001-of-0000...	351 B	text/plain	Standard	Not public	—	Google-managed key	—	None
prediction.results-00002-of-0000...	347 B	text/plain	Standard	Not public	—	Google-managed key	—	None
prediction.results-00003-of-0000...	349 B	text/plain	Standard	Not public	—	Google-managed key	—	None
prediction.results-00004-of-0000...	309 B	text/plain	Standard	Not public	—	Google-managed key	—	None

***Figure 2.23**: Batch prediction output*

One csv file will be created for each record (prediction_results.csv). `Prediction.csv` contains the information of the input record along with the probability of scores for different class. `Error_stats.csv` will provide the information for the users if any errors are encountered during the prediction.

Model deployment for online predictions

The model package, which we will bundle into a docker image to deploy, comprises all the parts required to execute a model, including the run environment, ML framework, stored model file, and supported backend microservice. On the other hand, the endpoint is the place where the computer resources are located where your model is really processing prediction queries. For auto-scaling, users may choose the machine type and other factors.

The model must be deployed on an Endpoint before predictions based on it may be accepted. Follow below steps to deploy the model.

Step 1: Model deployment

Model deployment can be initiated under deploy and test as shown in *Figure 2.24*:

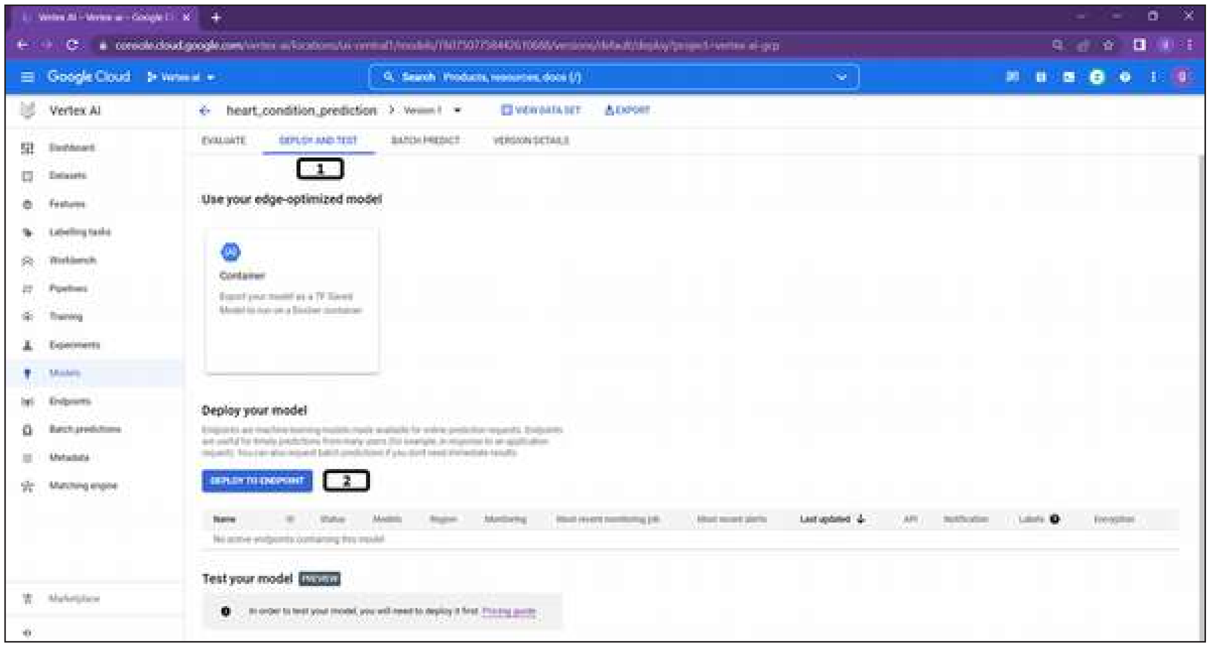


***Figure 2.24**: Model deployment*

1. Go to **Deploy and Test** tab.
2. Select **Deploy to endpoint**.

Step 2: Endpoint definition

The steps involved in Endpoint definition can be seen in *Figure 2.25*:

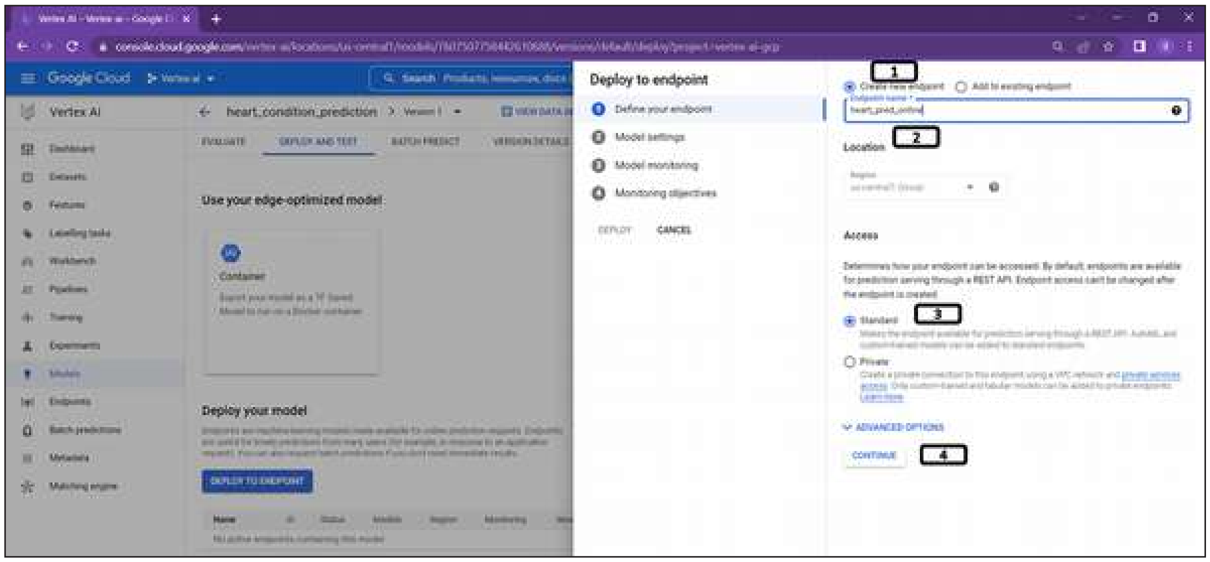
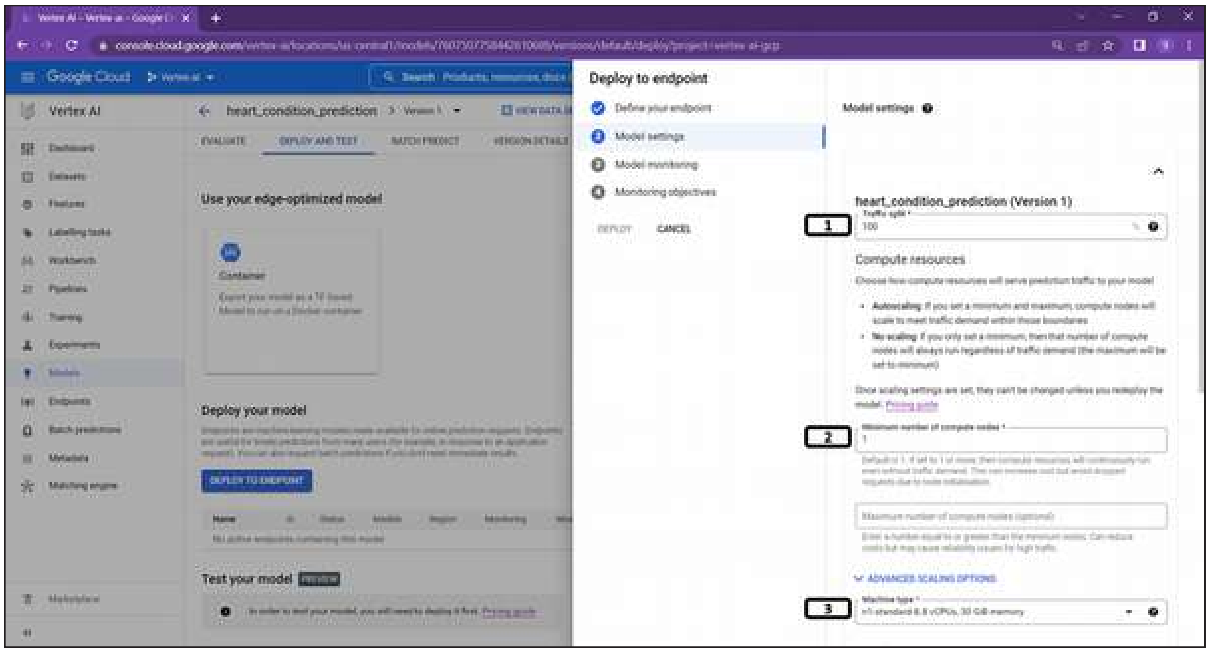


***Figure 2.25**: Endpoint creation*

1. Select **CREATE NEW ENDPOINT**, if model needs to be deployed to an existing endpoints users are provided with an option of Add to existing endpoints.
2. Provide the name for the endpoint.
3. Access can be standard unless it is for a private access
4. Click on **Continue**.

Step 3: Model settings

Traffic split, hardware needed for prediction can be chosen as shown in *Figure 2.26*:



heart_condition_prediction (Version 1)

Traffic split: 100

Minimum number of compute nodes: 1

Machine type: n1-standard-8, 8 vCPUs, 30 GiB memory

Figure 2.26: Model settings for deployment

1. If model is deployed in multiple endpoints, traffic split can be done between them. If the model is deployed in only one endpoint then it has to be set 100%.
2. Number of nodes required for prediction.
3. Select the machine type (if the model is very complicated then it is recommended to select machines with good configuration).

Scroll down to choose the options of logging and click **Continue**.

Step 4: Model monitoring

Follow below steps as shown in *Figure 2.27* to enable model monitoring in production:

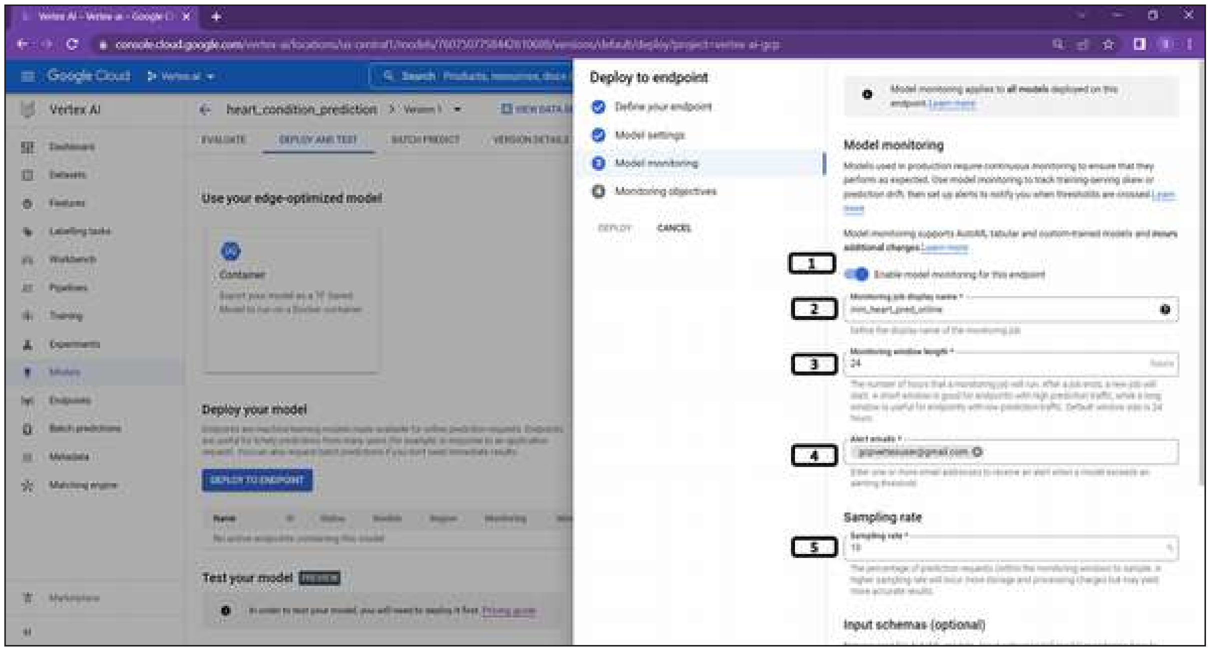


Figure 2.27: Model monitoring

1. Users can enable the model monitoring for the deployed model (model can be deployed without enabling the monitoring).
2. Provide name for the monitoring job.
3. Monitoring will be done by multiple jobs. Time length of the job needs to be provided. If the expected load is high then low number is recommended and vice versa.
4. Mail id for the alerts.
5. Percentage of request can be chosen within a monitoring window for sampling.

Input schema is optional for AutoML models and can be ignored. Scroll down and click **Continue**.

Step 5: Monitoring objectives

Models performance deteriorate over the time, follow steps shown in *Figure 2.28* to choose the model monitoring objectives:

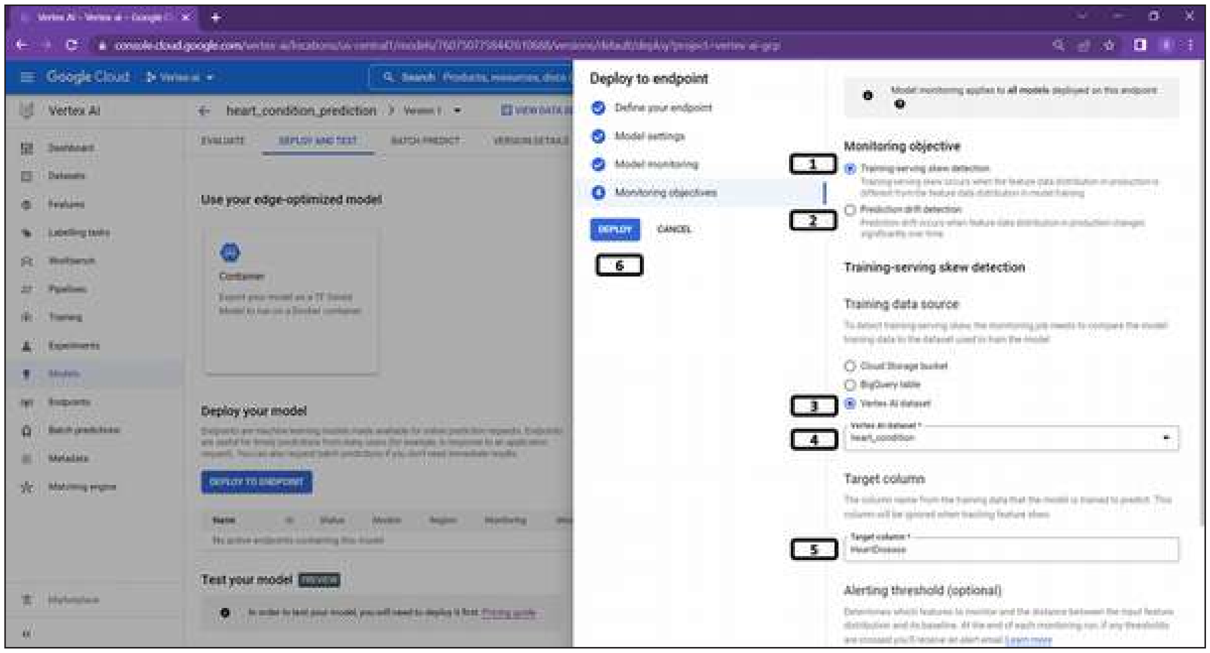


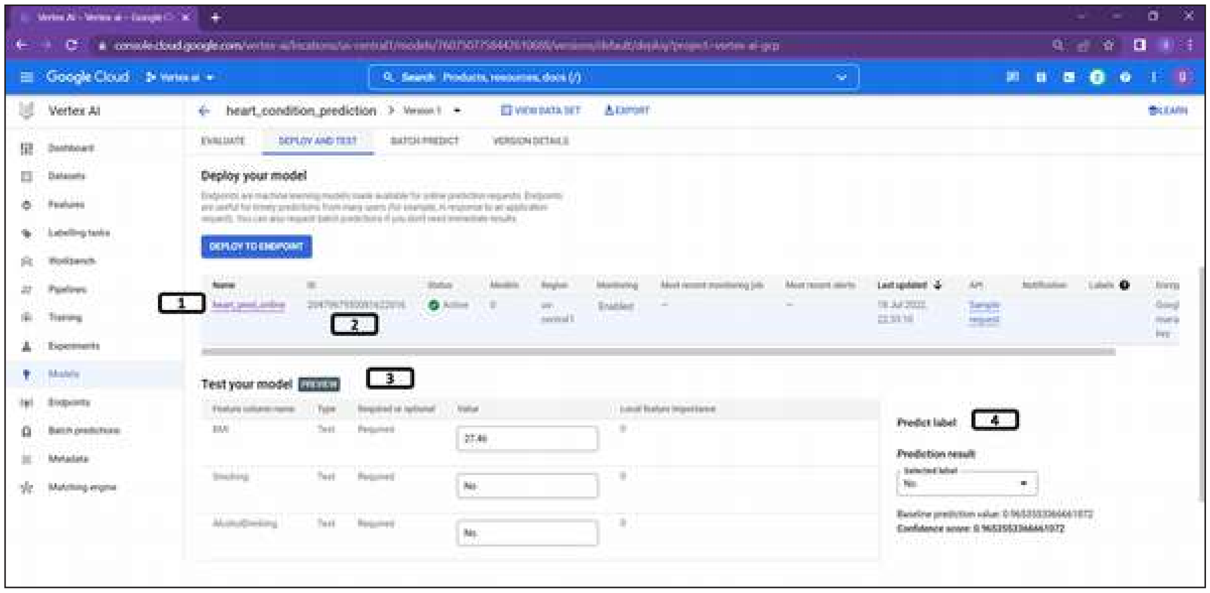
Figure 2.28: *Monitoring objective*

1. Prediction requests are logged in a BigQuery table in the same project. The input feature values contained in the logged requests are then analyzed for skew or drift (it requires training data for the comparison).
2. Model monitoring tracks changes in a feature's contributions to a model's predictions over time (no training data is required).
3. Since we have chosen skew detection, user needs to provide the training data options.
4. Chose the dataset which is used for training the AutoML model.
5. Provide the target column name.
6. Click **DEPLOY**.

It will take few mins to deploy the model to the endpoints.

Step 6: Testing deployed model

Follow the steps shown in *Figure 2.29* to test the deployed model on the platform before proceeding for online predictions:



Deploy your model

Name	ID	Status	Models	Region	Monitoring	Most recent monitoring job	Most recent alerts	Last updated	API
heart_pred_online	[illegible]	Active	0	us-central1	Enabled	–	–	18 Jul 2022, 22:33:16	Sample request

Test your model

Feature column name	Type	Required or optional	Value	Local feature importance
BMI	Text	Required	27.46	0
Smoking	Text	Required	No	0
AlcoholDrinking	Text	Required	No	0

Predict label

Prediction result

Selected label: No

Baseline prediction value: 0.9653553366461072

Confidence score: 0.9653553366461072

Figure 2.29: Online prediction testing

1. Model is deployed successfully.
2. ID is the model ID that is deployed. (One endpoints can deploy multiple models). Same ID is used for online predictions.
3. Even before users get the online predictions, users can test the model output on the platform by populating column values and clicking on predict at the bottom of the screen.
4. Output prediction will be populated with the probability scores.

Service account creation

Deployed model is ready to serve the request for predictions. While sending data and model details for predictions users should also be authenticated. In these scenarios user GCP account details cannot be used for authentication, service account is needed. Follow below steps for service account creation. Open IAM and admin and follow steps as shown in *Figure 2.30*:

Step 1: Service account creation

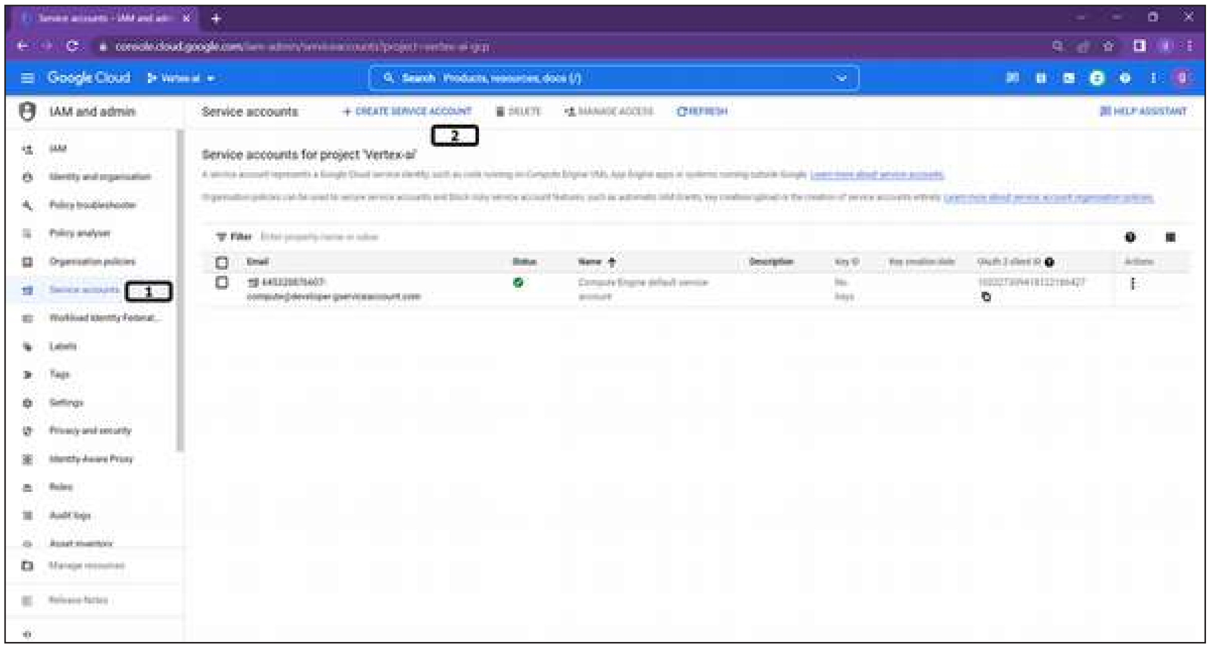
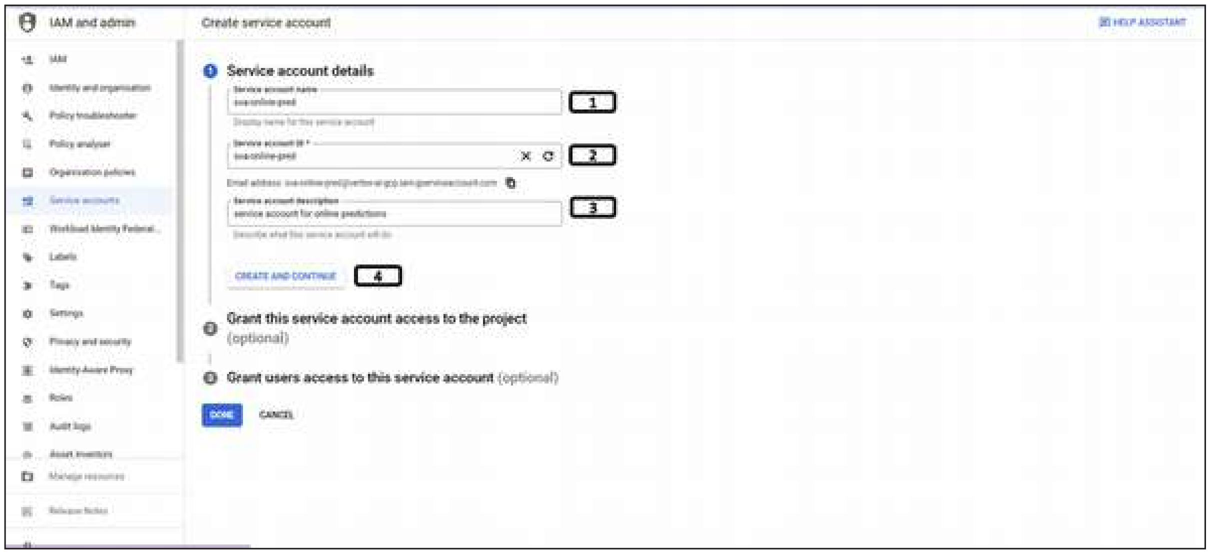


Figure 2.30: *Service account*

1. Select service accounts under IAM and admin section.
2. Click on **Create service account**.

Step 2: Service account details

Follow steps shown in *Figure 2.31* to add account details:



Service account name: sva-online-pred

Service account ID: sva-online-pred

Service account description: service account for online predictions

Figure 2.31: *Service account details*

1. Provide name for the service account.
2. Provide service account ID.
3. Provide description for the service account.
4. Click on **Create and continue**.

Changing the access to the service account is not required for this exercise.

Step 3: Assigning roles for the service account

During the process of service account creation let us also grant the access to certain roles needed for this service account. Granting these roles are important since we will use the same service account for custom model building, pipelines, and so on. Follow steps shown in the *Figure 2.32* to grant roles. (It is an optional step and can be done as and when needed as well).

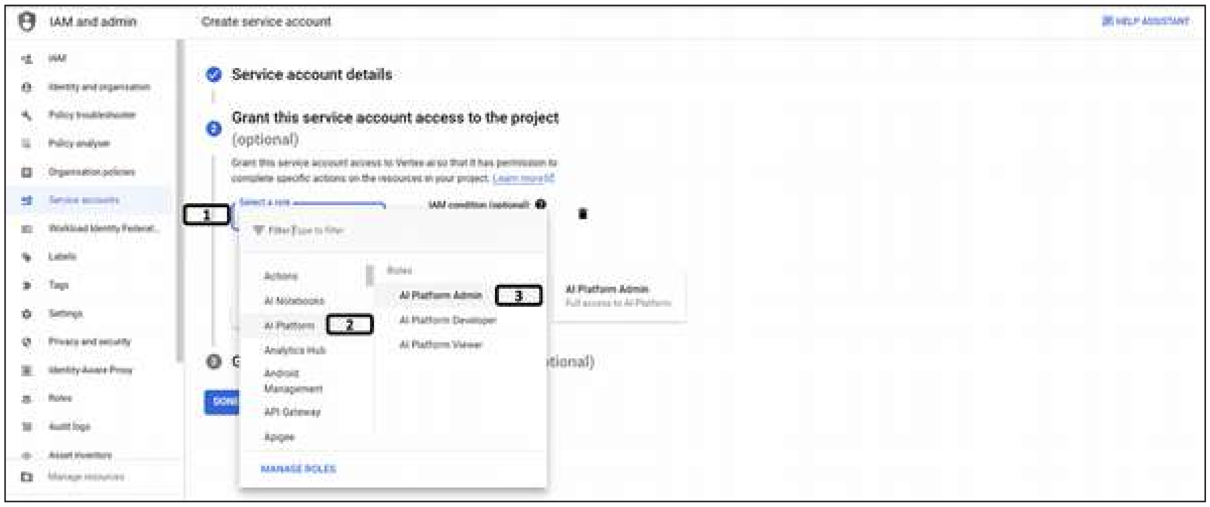


***Figure 2.32**: Grant AI platform access to service account*

1. Click on select role.
2. Select **AI Platform** under actions.
3. Select **AI platform admin**.

Similarly, add few other roles belonging to cloud storage and vertex AI as displayed in the *Figure 2.33*:

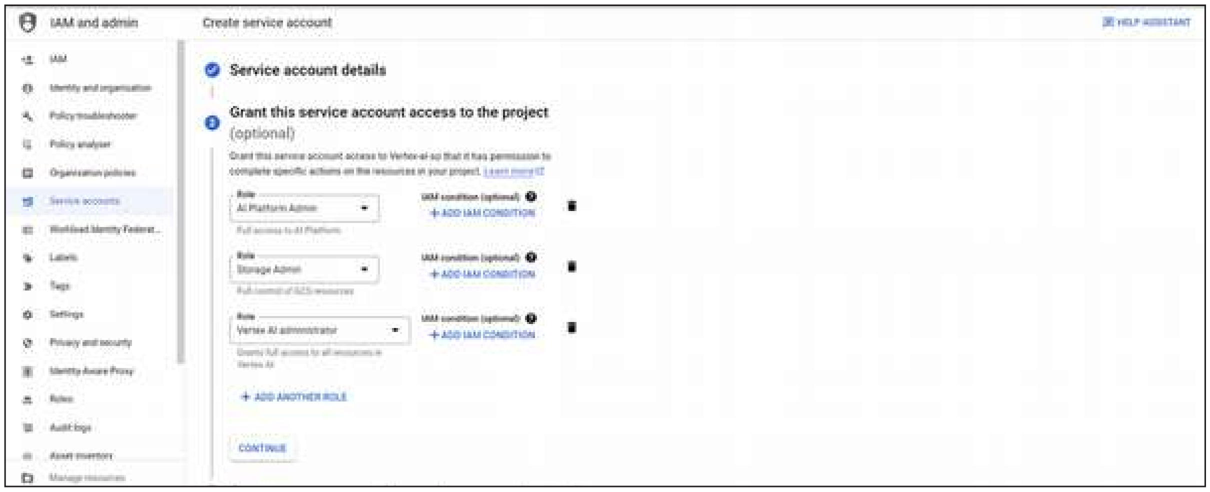


Figure 2.33: Grant other access to service account

Step 4: Service account manage keys

Once service account is created, users need to create keys. Follow steps shown in *Figure 2.34* and *Figure 2.35* to create keys:

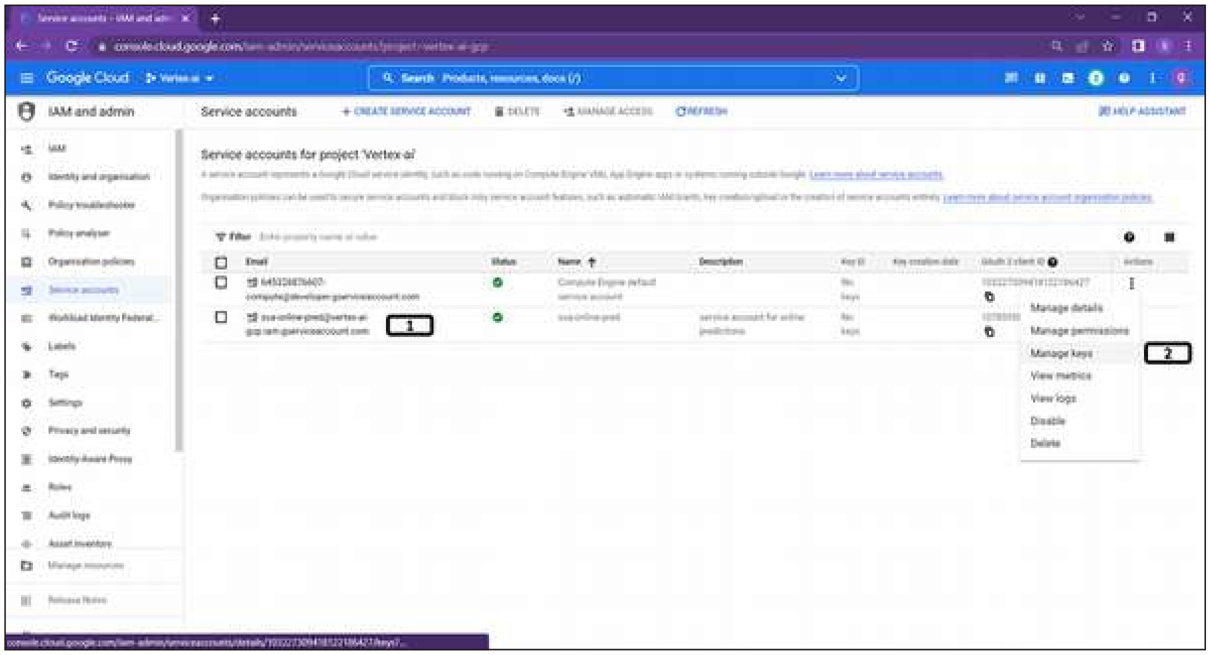
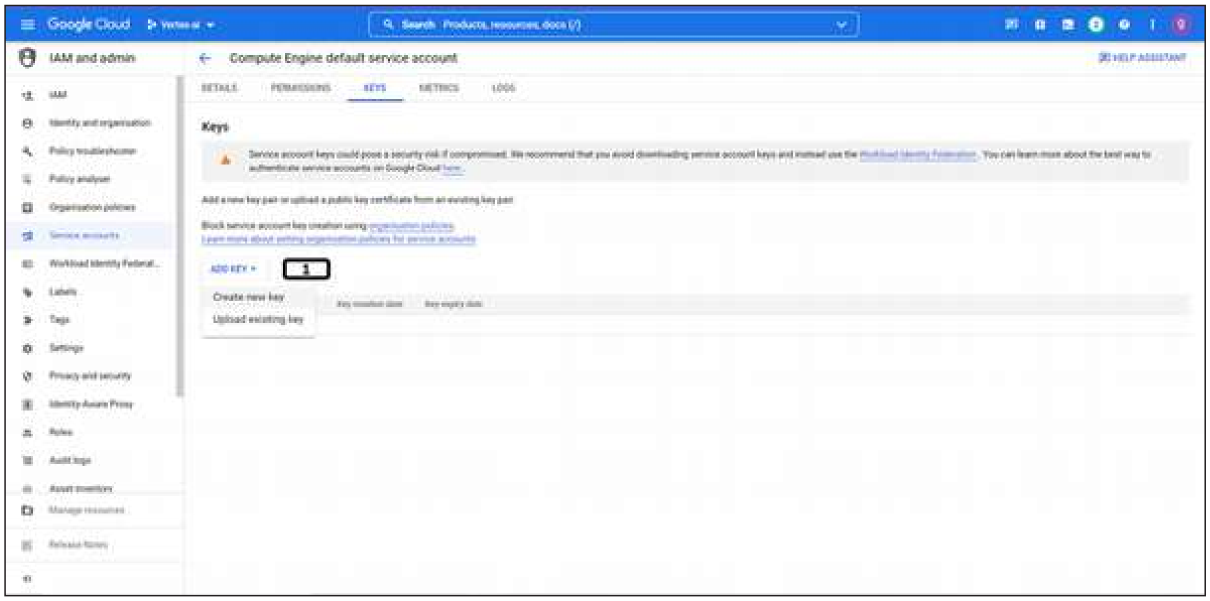

Service accounts – IAM and adm
console.cloud.google.com/iam-admin/serviceaccounts?project=vertex-ai-gcp
Google Cloud
Vertex-ai
Search Products, resources, docs (/)
IAM and admin
Service accounts
+ CREATE SERVICE ACCOUNT
DELETE
MANAGE ACCESS
REFRESH
HELP ASSISTANT
Service accounts for project 'Vertex-ai'
Filter
Email
Status
Name
Description
Key ID
Key creation date
OAuth 2 client ID
Actions
compute@developer.gserviceaccount.com
Compute Engine default service account
No keys
vca-online-pred@vertex-ai-gcp.iam.gserviceaccount.com
vca-online-pred
service account for online predictions
1
Manage details
Manage permissions
Manage keys
View metrics
View logs
Disable
Delete
2

Figure 2.34: Service account manage keys

1. Service account is now created.
2. Click on the actions of the created service account and click on manage keys.

Step 4: Service account add keys



Service account keys could pose a security risk if compromised. We recommend that you avoid downloading service account keys and instead use the Workload Identity Federation. You can learn more about the best way to authenticate service accounts on Google Cloud here.

Add a new key pair or upload a public key certificate from an existing key pair

Block service account key creation using organization policies.
Learn more about setting organization policies for service accounts

Figure 2.35: Service account key creation

1. Click on **ADD KEY** and choose create new key.

A Pop up will provide options to create key in json/P12 format. Choose json format and .json file will be downloaded.

NOTE: Users needs to provide full path of the json file for authentication in the python code. We can use the same key for next chapter also.

Serving online predictions

Model is deployed on to the GCP endpoint. Users can get the predictions from the deployed model using python on local machine.

Install `google-cloud-aiplatform` python package through the following command:

```
pip install google-cloud-aiplatform
```

More information on the package can be obtained from the following link:

https://pypi.org/project/google-cloud-aiplatform/

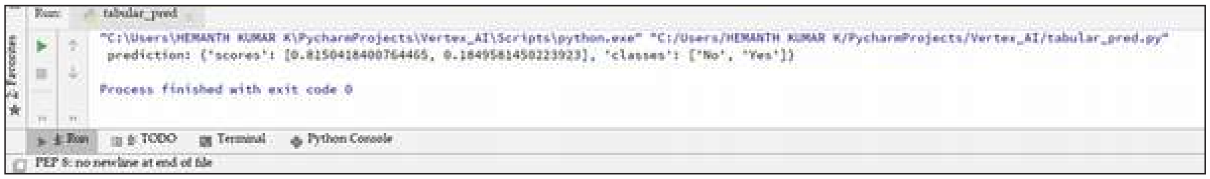
Python Code for predictions:

```
#Install google-cloud-aiplatform

from google.cloud import aiplatform
from google.protobuf import json_format

from google.protobuf.struct_pb2 import Value

import os


def     online_pred_tabular(project,model_id,instance_dict,location,api_
endpoint):

    client_options = {“api_endpoint”: api_endpoint}

    client = aiplatform.gapic.PredictionServiceClient(client_
    options=client_options)

    instance = json_format.ParseDict(instance_dict, Value())

    instances = [instance]

    parameters_dict = {}

    parameters = json_format.ParseDict(parameters_dict, Value())

    endpoint = client.endpoint_path(project=project, location=location,
    endpoint_id= model_id)

    response = client.predict(endpoint=endpoint, instances=instances,
    parameters =parameters)

    predictions = response.predictions

    for prediction in predictions:print(“ prediction:”, dict(prediction))

#For GCP authentication. Provide full path of the json file

os.environ[‘GOOGLE_APPLICATION_CREDENTIALS’] =””

project_id= “vertex-ai-gcp” #project_ID can be obtained from GCP dashboard
(refer figure 2.35)

model_id=”2047967950081622016” #Model_ID refer figure 2.29

location = “us-central1” #Location refer to figure 2.25

api_endpoint=”us-central1-aiplatform.googleapis.com”

#Inputs in Json format

inputs  =  {“BMI”:  “16.6”,  “Smoking”:  “Yes”,  “AlcoholDrinking”:  “No”,
“Stroke”: “No”, “PhysicalHealth”: “3”, “MentalHealth”: “30”, “DiffWalking”:
```

```
“No”, “Sex”: “Female”, “AgeCategory”: “55-59”, “Race”: “White”,Diabetic”:
“Yes”, “PhysicalActivity”: “Yes”, “GenHealth”: “Very good”, “SleepTime”:
“5”, “Asthma”: “Yes”, “KidneyDisease”: “No”, “SkinCancer”: “Yes”}

#Calling the function

online_pred_tabular(project_id, model_id,inputs,location,api_endpoint)
```

Output of the python code

The output of the python code can be seen in *Figure 2.36*:



```
"C:\Users\HEMANTH KUMAR K\PycharmProjects\Vertex_AI\Scripts\python.exe" "C:/Users/HEMANTH KUMAR K/PycharmProjects/Vertex_AI/tabular_pred.py"
 prediction: {'scores': [0.8150418400764465, 0.18495814502239231, 'classes': ['No', 'Yes']}

Process finished with exit code 0
```

***Figure 2.36**: Python code output*

Output contains probability scores for each of the classes.

> **NOTE: There are multiple ways of authentication, method used in the code is the simplest.**

Project_ID can be obtained from the dashboard of the GCP (landing page of the platform) as seen in the following screenshot:

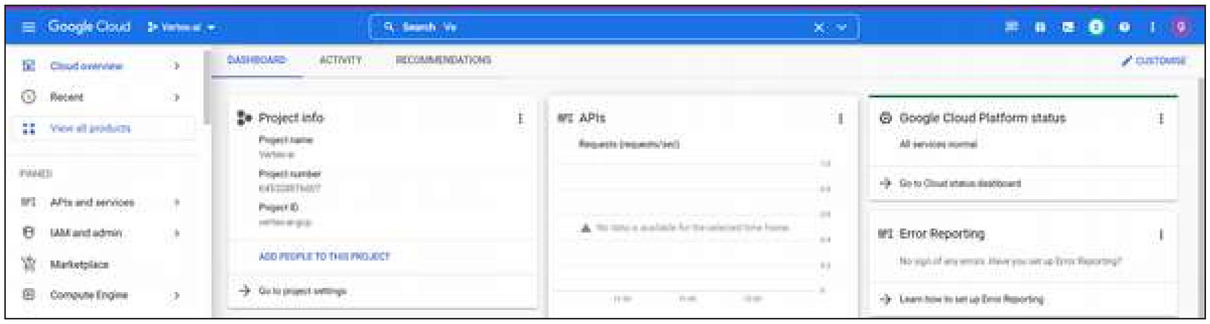


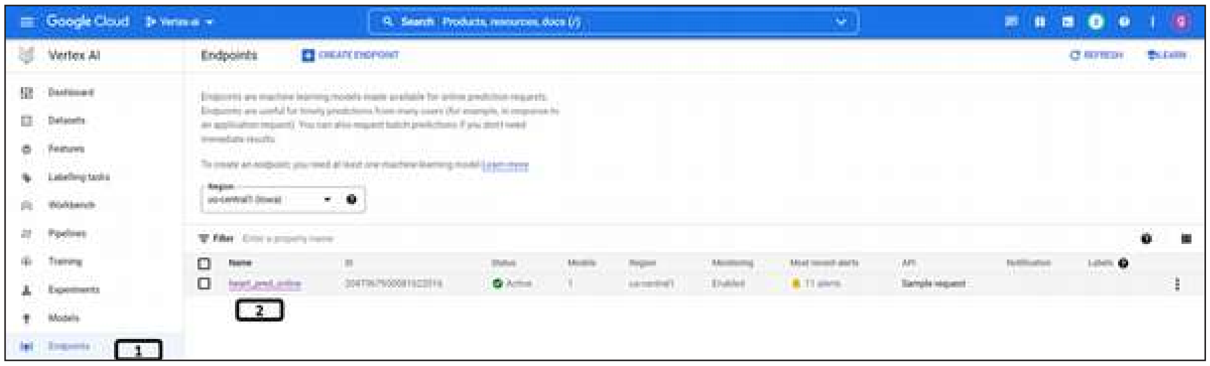
***Figure 2.37**: Project_ID from dashboard*

Model un-deployment and deleting end point

Endpoint cannot be deleted until the all the models are removed from deployment.

Step 1: Selecting the endpoint

Deployed model will be listed under endpoints as shown the *Figure 2.38*:

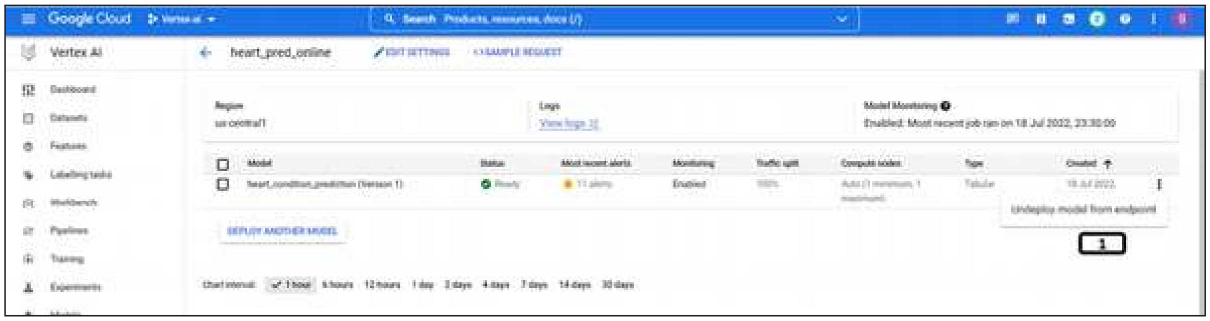


Name	ID	Status	Models	Region	Monitoring	Most recent alerts	API	Notification	Labels
heart_pred_online	2047967900081622016	Active	1	us-central1	Enabled	11 alerts	Sample request		

Figure 2.38: *Endpoint where model is deployed*

1. Navigate to endpoints.
2. Click on the endpoint that needs to be deleted.

Step 2: Removing model from deployment



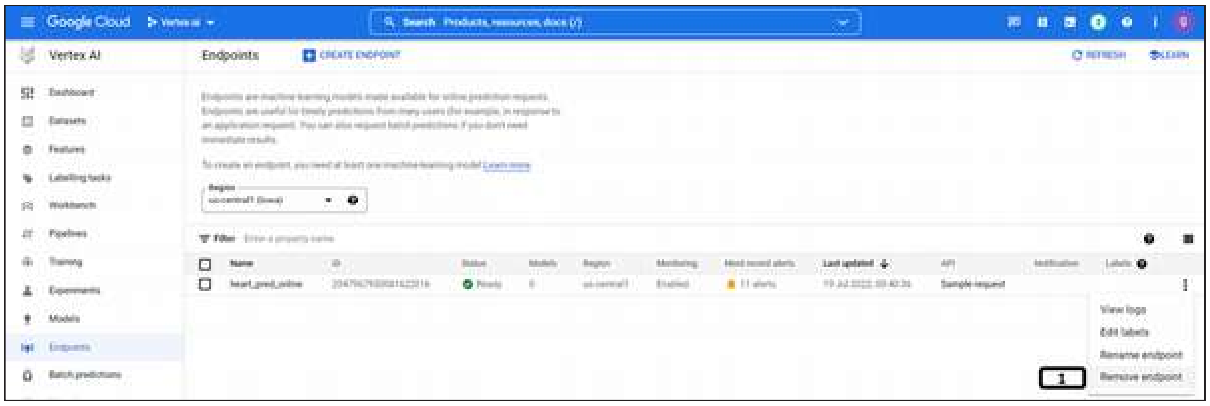
Region: us-central1
Logs: View logs
Model Monitoring: Enabled: Most recent job ran on 18 Jul 2022, 23:30:00

Model	Status	Most recent alerts	Monitoring	Traffic split	Compute nodes	Type	Created
heart_condition_prediction (Version 1)	Ready	11 alerts	Enabled	100%	Auto (1 minimum, 1 maximum)	Tabular	18 Jul 2022

Figure 2.39: *Removing model from deployment*

1. Select the model for removing and click on un-deploy model from endpoint.
2. A Popup will ask for confirmation.

Step 3: Endpoint deletion



Name	ID	Status	Models	Region	Monitoring	Most recent alerts	Last updated	API	Notification	Labels
heart_pred_online	2047967900081622016	Ready	0	us-central1	Enabled	11 alerts	19 Jul 2022, 00:40:36	Sample request		

Figure 2.40: *Endpoint deletion*

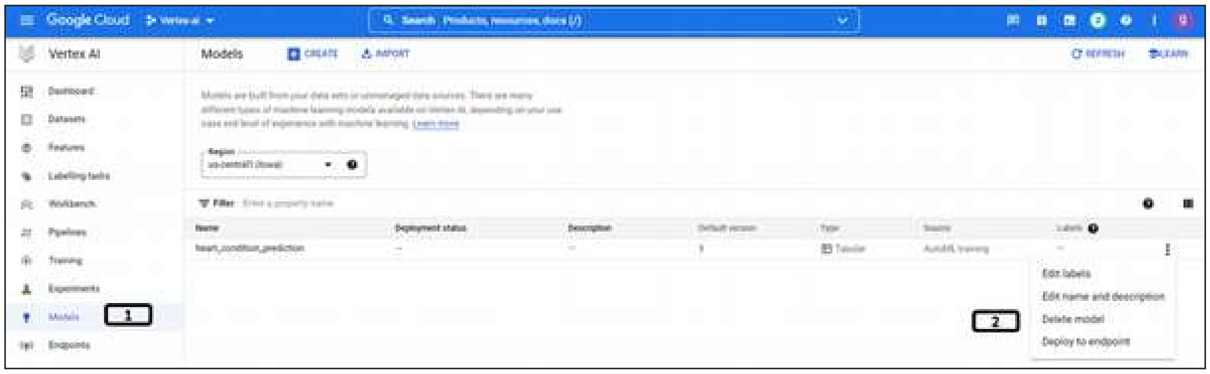
1. Select the endpoint to be deleted and click **Remove endpoint**.

NOTE: Deploying model is cost incurring, ensure to delete the model from deployment.

Model deletion

In the entire process of working on the AutoML with tabular data we have created dataset and models.

Please follow steps mentioned in the *Figure 2.41* to delete the trained model:



Models are built from your data sets or unmanaged data sources. There are many different types of machine learning models available on Vertex AI, depending on your use case and level of experience with machine learning. Learn more

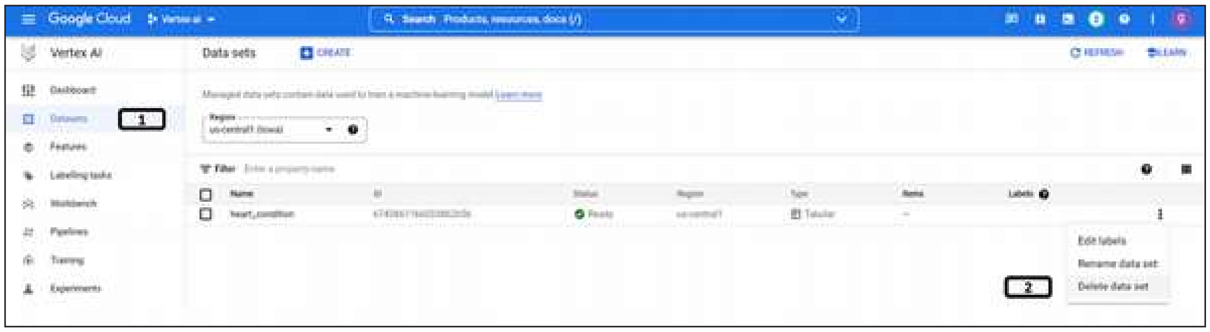
Name	Deployment status	Description	Default version	Type	Source	Labels
heart_condition_prediction	–	–	1	Tabular	AutoML training	–

Figure 2.41: Deletion of trained model

1. Navigate to models.
2. Select the trained model for deletion.

Dataset deletion

Please follow steps described in the *Figure 2.42* to delete the dataset created:



Managed data sets contain data used to train a machine learning model Learn more

Name	ID	Status	Region	Type	Items	Labels
heart_condition	[illegible]	Ready	us-central1	Tabular	–	

Figure 2.42: Dataset deletion

1. Navigate to datasets.
2. Select the dataset and click on **Delete dataset**.

NOTE: Note: batch predictions output are stored in the cloud storage, delete those files to complete clean up activity.

Conclusion

In this chapter we learnt about Vertex AI and key their features. We worked on the AutoML tabular, created dataset, trained the model and put it for deployment for predictions. In the next chapter we will work on AutoML for images and text data.

Questions

1. Can user train AutoML model without creating managed datasets?
2. Can multiple models be put into same endpoint for deployment?
3. Why is service account needed for getting online predictions?
4. What are the different tasks AutoML can handle for the tabular data?

Chapter 3
AutoML Image, Text, and Pre-built Models

Introduction

In the last chapter, we started working on AutoML for tabular data. This chapter continues with the AutoML section for images and text data, followed by step-by-step instructions for images and text data. GCP also provides pre-built models, and this chapter covers them in brief. This chapter also talks about the benefits and limitations of AutoML, in brief.

Structure

In this chapter, we will discuss the following topics

- Vertex AI AutoML for image data
- Image dataset creation
- Image model training
- Image model evaluation
- Batch prediction for image
- Model deployment for online predictions for image
- Serving online predictions image

- Vertex AI AutoML for text data
- Text dataset creation
- Text model training
- Text model evaluation
- Batch prediction for text
- Model deployment for online predictions for text
- Serving online predictions text
- Pre-built models in GCP
- Benefits of AutoML
- Limitations of AutoML

Objectives

By the end of this chapter, you will be able to create datasets, train the AutoML model, obtain batch predictions, and deploy the models for online predictions for images and text data. Also, get introduced to pre-built models of GCP.

Vertex AI AutoML for image data

AutoML for images can handle tasks of single-label, multi-label classification, object detection, and image segmentation. For the practical exercise of the AutoML for image single-label classification we are using boat classification image data which can be downloaded from below mentioned link:

https://www.kaggle.com/datasets/imsparsh/dockship-boat-type-classification

The dataset is listed under CC0: Public domain license and contains images belonging to nine categories. For our exercise, we are considering images belonging to cruise ships, ferry boats, and kayaks (50 images are chosen randomly from each category).

`AutoML_image_data` bucket is created under **`us-centra1`** (single region) and three folders containing images belonging to three categories are uploaded. An additional folder is created in the bucket by the name **`Data_for_batch_pred`** which contains a randomly chosen image for batch prediction. A snapshot of the cloud storage is shown in *Figure 3.1*:

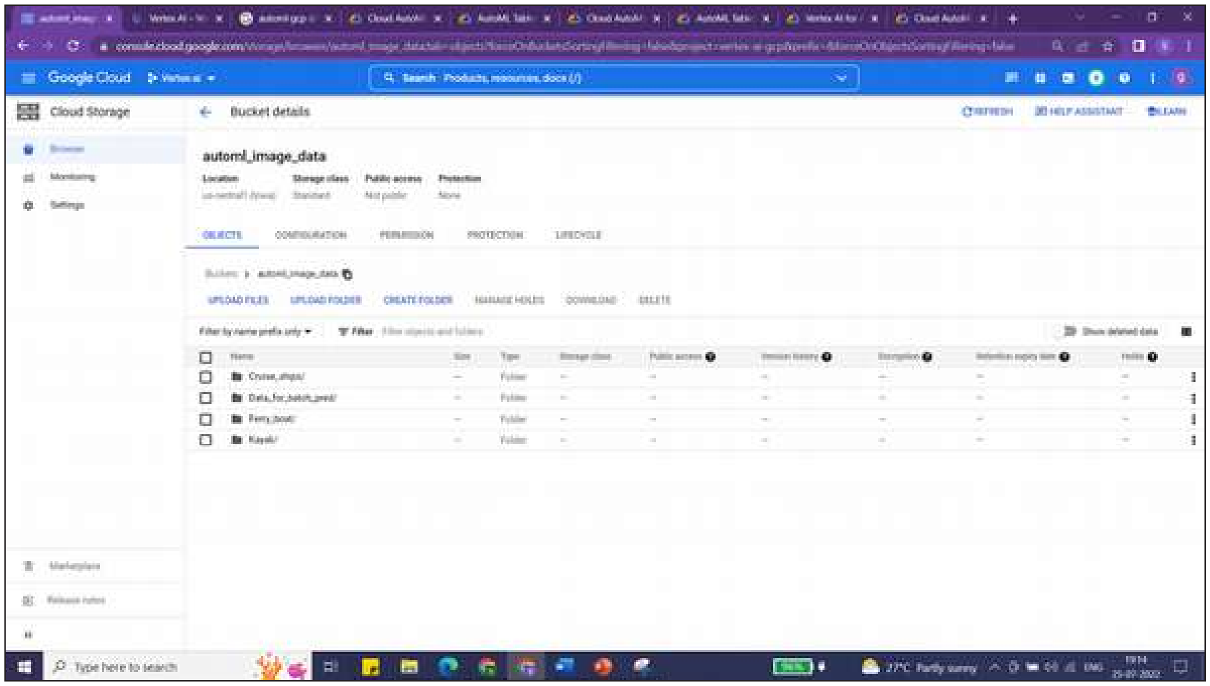


***Figure 3.1**: Image data uploaded to cloud storage*

The steps for working on the AutoML image data are like AutoML tabular data with few changes while creating the dataset and model training. In the case of tabular data, we were able to directly create a dataset with the '.csv' file uploaded to the cloud storage, but the in case of image data, users need to create a csv that contains information about the image belonging to the training/validation/test, the full path of the image and the category it belongs to.

While we are dealing with only 150 images belonging to three categories, it will be easy to create csv manually with the required information. But if we are dealing with many images, manually creating a csv is not a feasible solution. But a few lines of commands on the cloud shell will make it easier.

```
for f in $(gsutil ls gs://AutoML_image_data/Cruise_ships/); do echo
UNASSIGNED,$f,Cruise_ships;done>>class_labels.csv

for f in $(gsutil ls gs://AutoML_image_data/Ferry_boat/); do echo
UNASSIGNED,$f,Ferry_boat;done>>class_labels.csv

for f in $(gsutil ls gs://AutoML_image_data/Kayak/); do echo
UNASSIGNED,$f,Kayak;done>>class_labels.csv
```

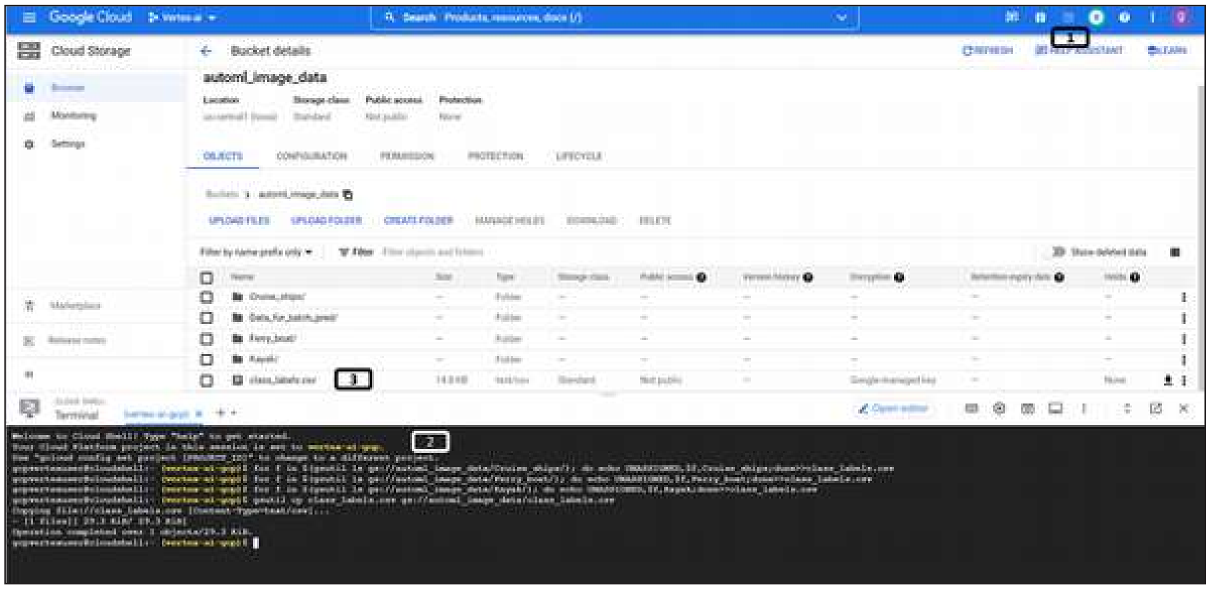
These commands will create a csv file that contains three columns:

- Setting an image to train/validation/test. In our case we have used the "UNASSIGNED", platform will choose images for training, validation, and testing.
- Full path of the images
- The category it belongs to.

Use the following command to push the CSV file to the cloud storage bucket:

`gsutil cp class_labels.csv gs://AutoML_image_data/class_labels.csv`

Open the command shell and type the commands as shown in *Figure 3.2*:

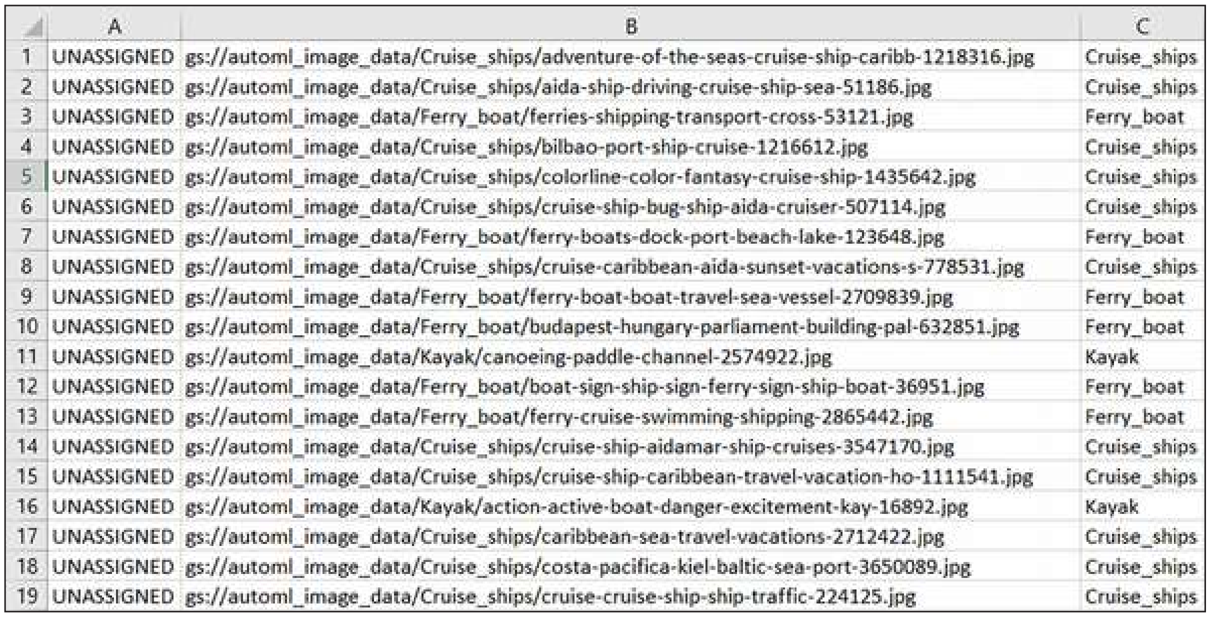


```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to vertex-ai-gcp.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
gcpvertexuser@cloudshell:~ (vertex-ai-gcp)$ for f in $(gsutil ls gs://automl_image_data/Cruise_ships/); do echo UNASSIGNED,$f,Cruise_ships;done>>class_labels.csv
gcpvertexuser@cloudshell:~ (vertex-ai-gcp)$ for f in $(gsutil ls gs://automl_image_data/Ferry_boat/); do echo UNASSIGNED,$f,Ferry_boat;done>>class_labels.csv
gcpvertexuser@cloudshell:~ (vertex-ai-gcp)$ for f in $(gsutil ls gs://automl_image_data/Kayak/); do echo UNASSIGNED,$f,Kayak;done>>class_labels.csv
gcpvertexuser@cloudshell:~ (vertex-ai-gcp)$ gsutil cp class_labels.csv gs://automl_image_data/class_labels.csv
Copying file://class_labels.csv [Content-Type=text/csv]...
- [1 files][ 29.3 KiB/ 29.3 KiB]
Operation completed over 1 objects/29.3 KiB.
gcpvertexuser@cloudshell:~ (vertex-ai-gcp)$
```

***Figure 3.2**: gs commands for csv creation (image data)*

1. Click on the icon to open the cloud shell.
2. Type the commands (type one command and press *Enter*).
3. New CSV is created and is available in the cloud storage.

A snapshot of the CSV file created is shown in *Figure 3.3*:



	A	B	C
1	UNASSIGNED	gs://automl_image_data/Cruise_ships/adventure-of-the-seas-cruise-ship-caribb-1218316.jpg	Cruise_ships
2	UNASSIGNED	gs://automl_image_data/Cruise_ships/aida-ship-driving-cruise-ship-sea-51186.jpg	Cruise_ships
3	UNASSIGNED	gs://automl_image_data/Ferry_boat/ferries-shipping-transport-cross-53121.jpg	Ferry_boat
4	UNASSIGNED	gs://automl_image_data/Cruise_ships/bilbao-port-ship-cruise-1216612.jpg	Cruise_ships
5	UNASSIGNED	gs://automl_image_data/Cruise_ships/colorline-color-fantasy-cruise-ship-1435642.jpg	Cruise_ships
6	UNASSIGNED	gs://automl_image_data/Cruise_ships/cruise-ship-bug-ship-aida-cruiser-507114.jpg	Cruise_ships
7	UNASSIGNED	gs://automl_image_data/Ferry_boat/ferry-boats-dock-port-beach-lake-123648.jpg	Ferry_boat
8	UNASSIGNED	gs://automl_image_data/Cruise_ships/cruise-caribbean-aida-sunset-vacations-s-778531.jpg	Cruise_ships
9	UNASSIGNED	gs://automl_image_data/Ferry_boat/ferry-boat-boat-travel-sea-vessel-2709839.jpg	Ferry_boat
10	UNASSIGNED	gs://automl_image_data/Ferry_boat/budapest-hungary-parliament-building-pal-632851.jpg	Ferry_boat
11	UNASSIGNED	gs://automl_image_data/Kayak/canoeing-paddle-channel-2574922.jpg	Kayak
12	UNASSIGNED	gs://automl_image_data/Ferry_boat/boat-sign-ship-sign-ferry-sign-ship-boat-36951.jpg	Ferry_boat
13	UNASSIGNED	gs://automl_image_data/Ferry_boat/ferry-cruise-swimming-shipping-2865442.jpg	Ferry_boat
14	UNASSIGNED	gs://automl_image_data/Cruise_ships/cruise-ship-aidamar-ship-cruises-3547170.jpg	Cruise_ships
15	UNASSIGNED	gs://automl_image_data/Cruise_ships/cruise-ship-caribbean-travel-vacation-ho-1111541.jpg	Cruise_ships
16	UNASSIGNED	gs://automl_image_data/Kayak/action-active-boat-danger-excitement-kay-16892.jpg	Kayak
17	UNASSIGNED	gs://automl_image_data/Cruise_ships/caribbean-sea-travel-vacations-2712422.jpg	Cruise_ships
18	UNASSIGNED	gs://automl_image_data/Cruise_ships/costa-pacifica-kiel-baltic-sea-port-3650089.jpg	Cruise_ships
19	UNASSIGNED	gs://automl_image_data/Cruise_ships/cruise-cruise-ship-ship-traffic-224125.jpg	Cruise_ships

***Figure 3.3**: Snapshot of csv created (image data)*

Image dataset creation

AutoML model training can be initiated only after creating the dataset. Follow the below steps to create an image dataset for image classification:

Step 1: Dataset creation

Navigate to the **Dataset** module of vertex AI as shown in *Figure 3.4*:

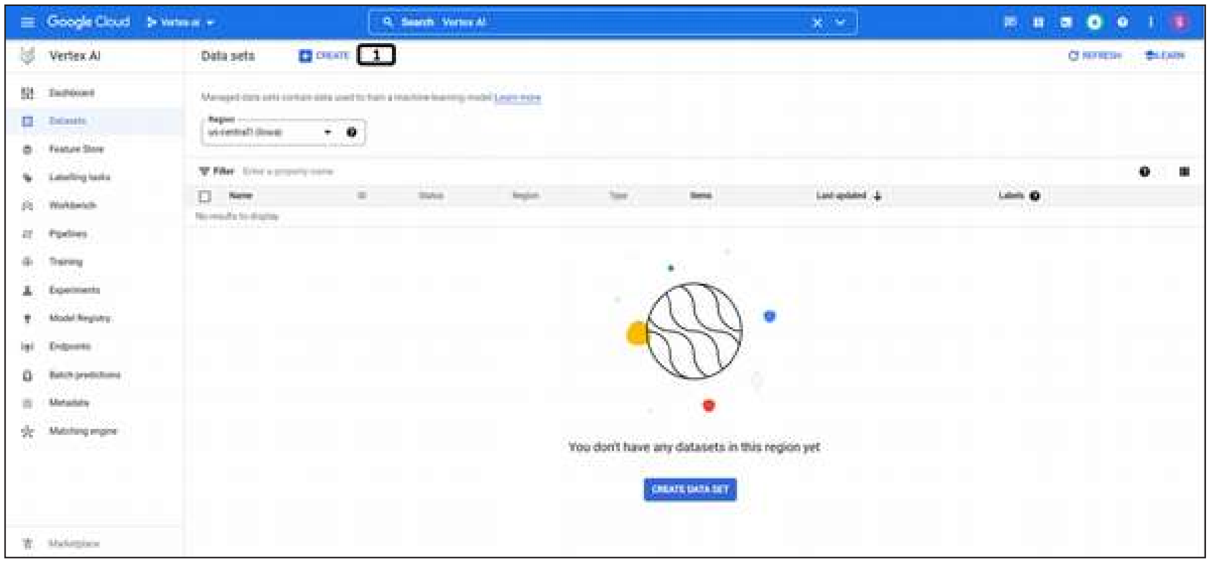
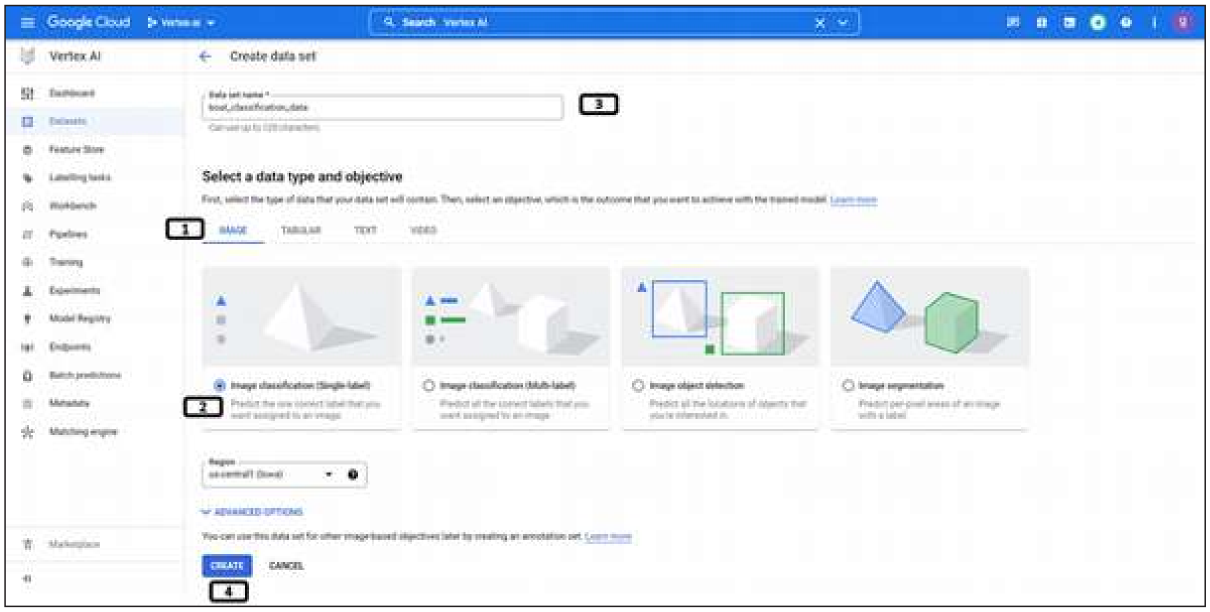


***Figure 3.4**: Landing page of the dataset*

1. Click on **CREATE**.

Step 2: Objective selection

The dataset creation tab will appear as shown in *Figure 3.5*:



Data set name *
boat_classification_data
Can use up to 128 characters

Select a data type and objective

First, select the type of data that your data set will contain. Then, select an objective, which is the outcome that you want to achieve with the trained model. Learn more

Image classification (Single-label): Predict the one correct label that you want assigned to an image

Image classification (Multi-label): Predict all the correct labels that you want assigned to an image

Image object detection: Predict all the locations of objects that you're interested in.

Image segmentation: Predict per-pixel areas of an image with a label

Region: us-central1 (Iowa)

You can use this data set for other image-based objectives later by creating an annotation set. Learn more

Figure 3.5: *Dataset selection for images*

1. Select the **IMAGE** tab.
2. Select **Image classification** (single label).
3. Provide a name for the dataset.
4. Click on **CREATE**.

Step 3: Data selection

Data can be selected from cloud storage or a local system. Follow the steps mentioned in *Figure 3.6* to upload images from the cloud storage:

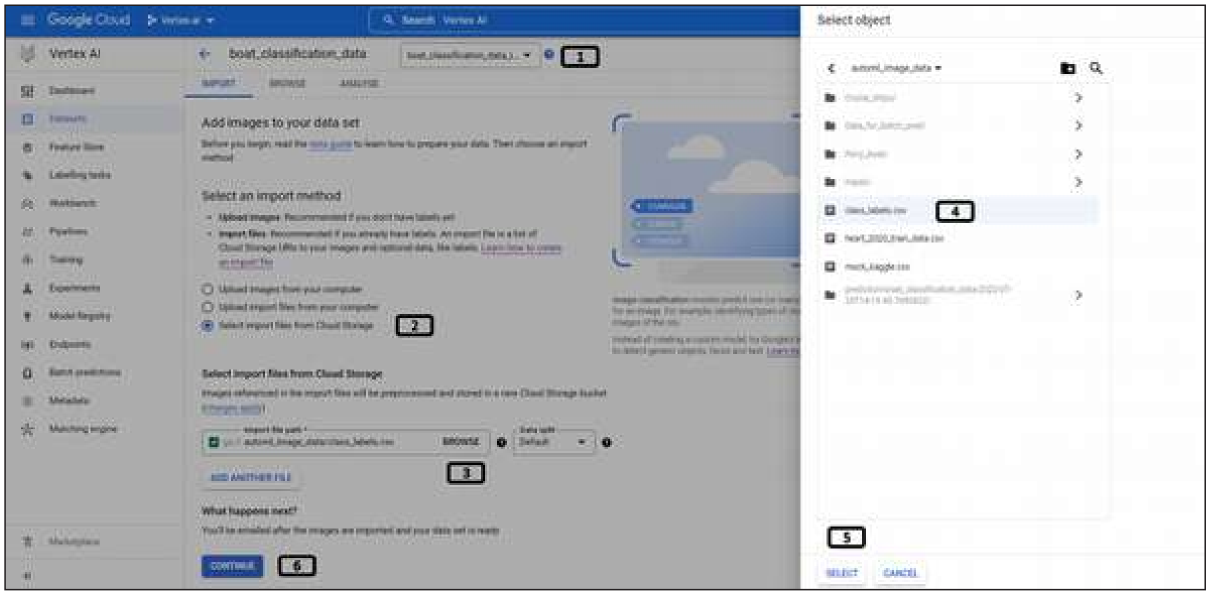


Figure 3.6: *Data selection for the image dataset*

1. Labels are stored in an annotation set which is associated with the model objective and will be automatically populated.
2. Select to import files from the cloud storage.
3. Click **BROWSE**.
4. Choose the CSV file created (refer to *Figure 3.6*).
5. Click **SELECT**.
6. Click **CONTINUE**.

Step 4: Data importing complete

Once the data importing is complete, data (images) will reflect as shown in *Figure 3.7*:

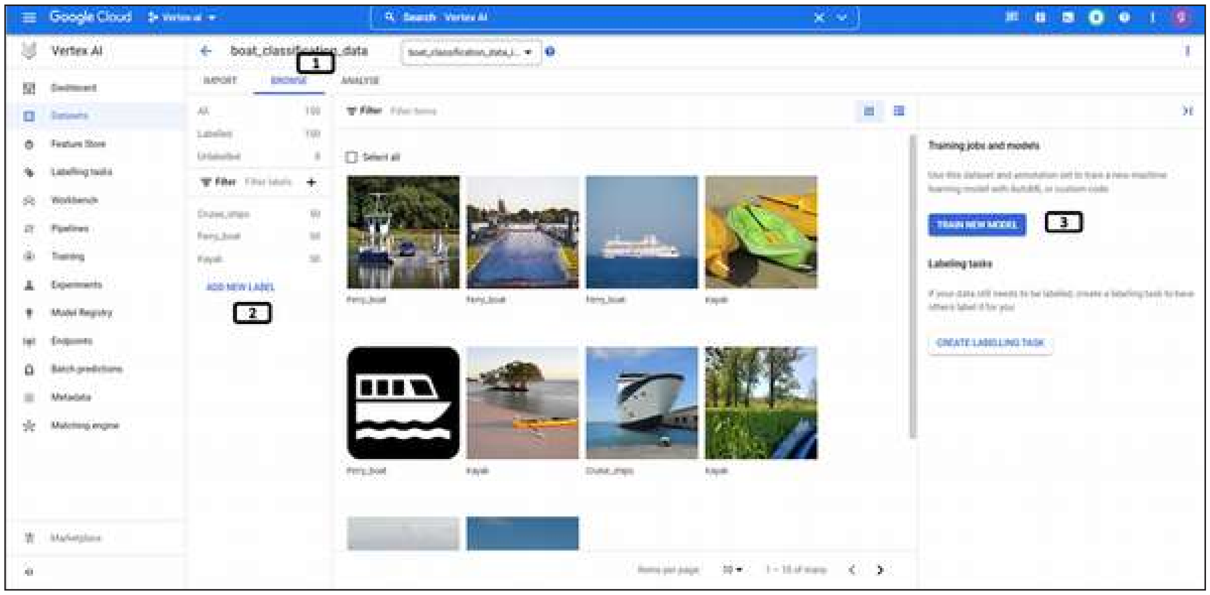
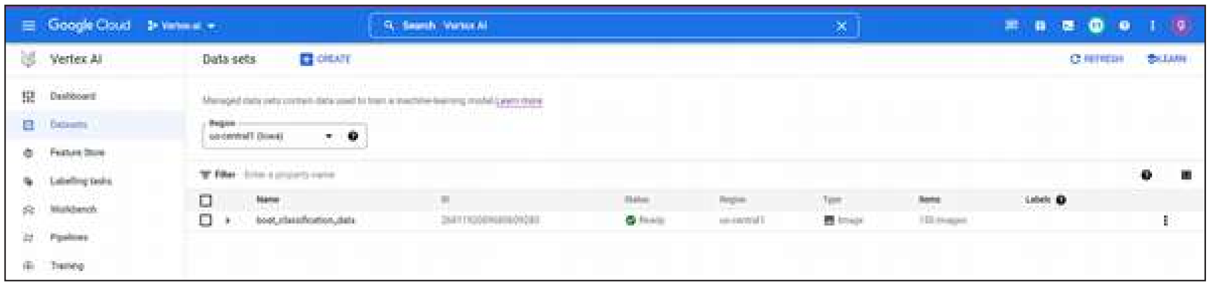


Figure 3.7: *Image dataset data import complete*

1. Navigate to the **BROWSE** tab.
2. Images uploaded are categorized as per the input CSV file.
3. Click on **TRAIN NEW MODEL** to start model training.

Once the dataset is created, it will be listed in the dataset module of the Vertex AI as shown in *Figure 3.8*:



Name	ID	Status	Region	Type	Items	Labels
boat_classification_data	[illegible]	Ready	us-central1	Image	150 images	

Figure 3.8: *Image dataset created under the dataset of Vertex AI*

NOTE:

- **GCP recommendation is to have at least 1000 images per label (minimum per label is 10.**
- **Maximum file size is 30MB.**

Model training image

Once the dataset is created and the data is uploaded to the dataset, we can start training the AutoML model. Follow the steps below to initiate the model training:

Step 1: Model Training method

Users will be navigated to the page as shown in *Figure 3.9* after clicking on **Train new model**:

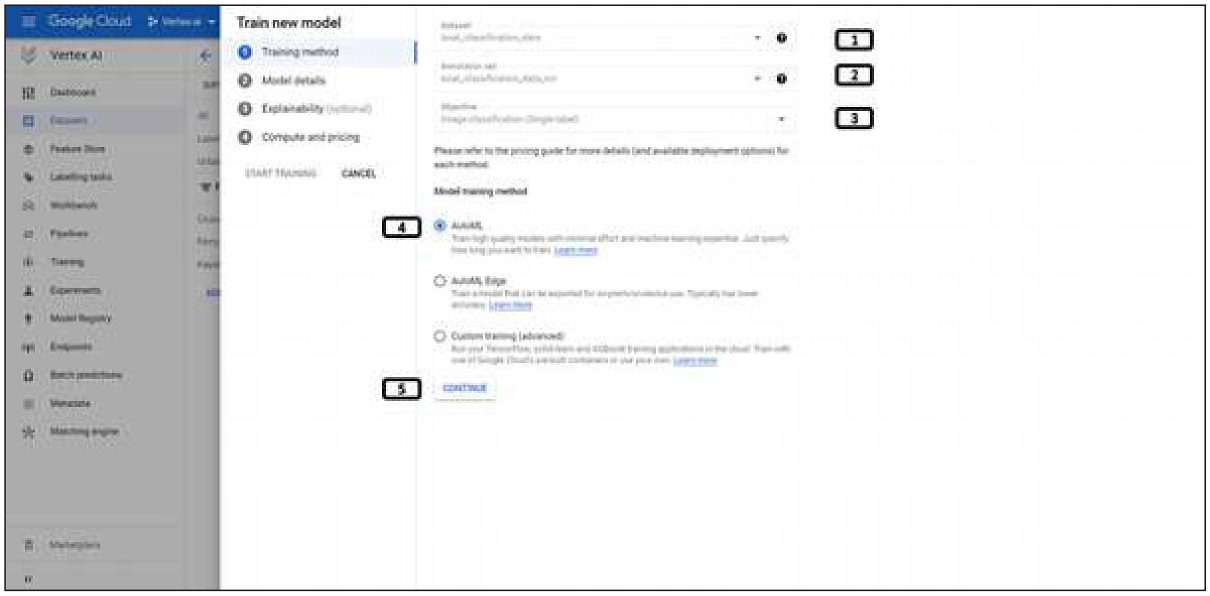
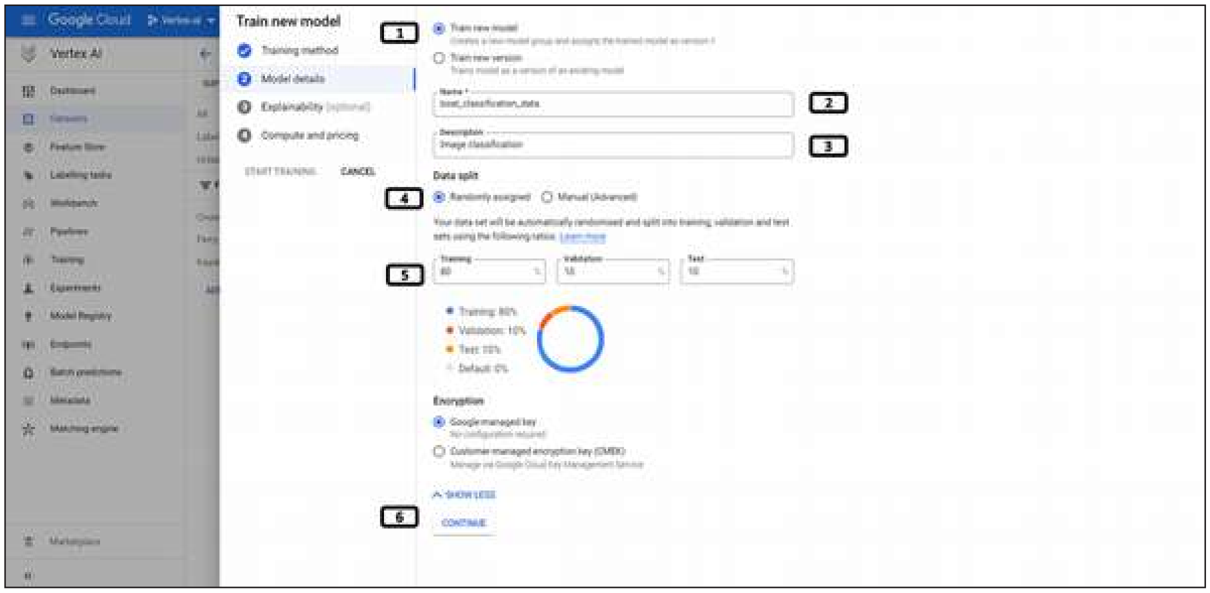


***Figure 3.9**: Training method selection for image*

1. Created dataset will automatically be populated.
2. **Annotation set** will also be automatically chosen.
3. **Objective** which was chosen while creating a dataset is automatically chosen.
4. Choose **AutoML** for training.
5. Click **CONTINUE**.

Step 2: Model details

The second step in model training is users providing **Model details**, follow the steps shown in *Figure 3.10*:



Train new model
Train new version

Name *
boat_classification_data

Description
Image classification

Data split
Randomly assigned
Manual (Advanced)

Your data set will be automatically randomized and split into training, validation and test sets using the following ratios. Learn more

Training 80 %
Validation 10 %
Test 10 %

Training: 80%
Validation: 10%
Test: 10%
Default: 0%

Encryption
Google-managed key
No configuration required
Customer-managed encryption key (CMEK)
Manage via Google Cloud Key Management Service

SHOW LESS

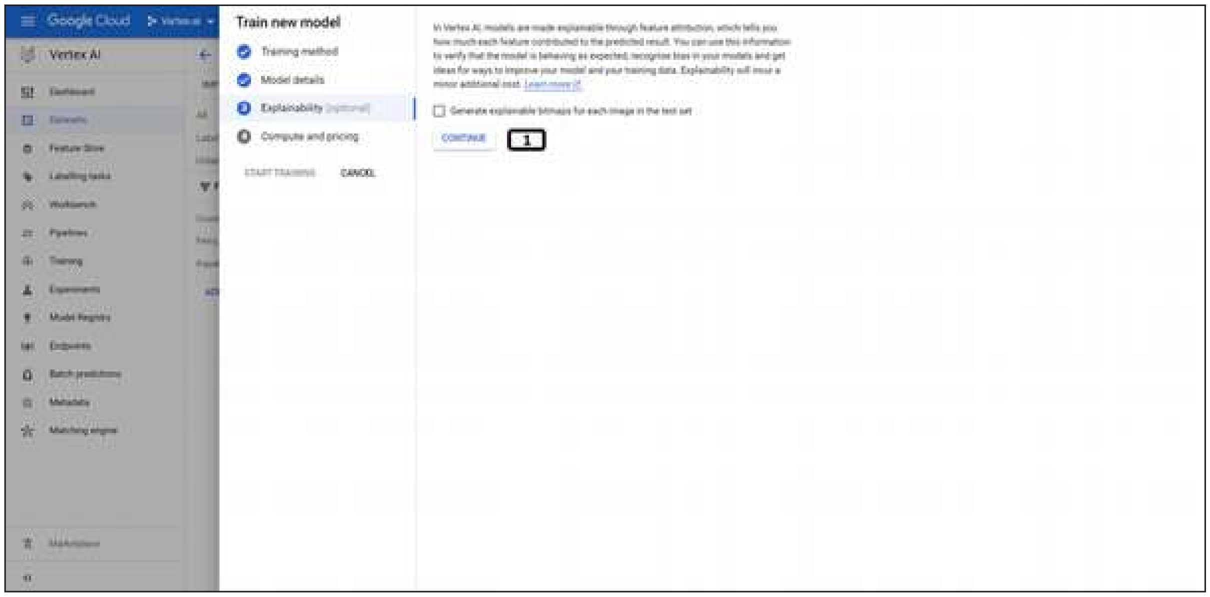
CONTINUE

Figure 3.10: *Model details for model training of image*

1. Select **Train new model** (if a new version of the existing model is to be trained choose the below option).
2. Provide the model's name.
3. Provide the model description.
4. Choose **Randomly assigned** (since we have given unassigned while creating the CSV file).
5. By default, 80% of the data will be used for training and 10% for validation and testing respectively. However, users can change as per their requirements.
6. Click **CONTINUE**.

Step 3: Explainability

Explainability of the model is an option that users can choose, for now, we will not choose that option. Explainability will be discussed in detail in the later part of the book. Users can skip the selection and choose **CONTINUE** as shown in *Figure 3.11*:



In Vertex AI, models are made explainable through feature attribution, which tells you how much each feature contributed to the predicted result. You can use this information to verify that the model is behaving as expected, recognize bias in your models and get ideas for ways to improve your model and your training data. Explainability will incur a minor additional cost. Learn more

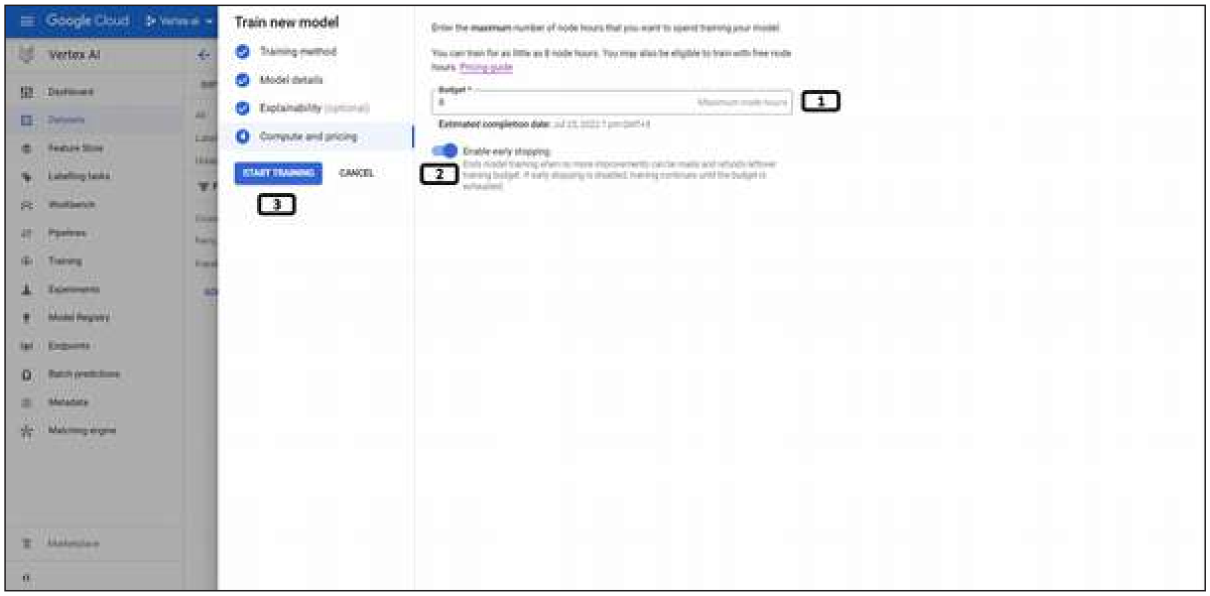
☐ Generate explainable bitmaps for each image in the test set

Figure 3.11: Model explainability for image

1. Click **CONTINUE**.

Step 4: Compute and pricing

Populate the compute hours for training as shown in *Figure 3.12*:



Enter the maximum number of node hours that you want to spend training your model.

You can train for as little as 8 node hours. You may also be eligible to train with free node hours. Pricing guide

Budget *: 8 Maximum node hours

Estimated completion date: [illegible]

Enable early stopping
Ends model training when no more improvements can be made and refunds leftover training budget. If early stopping is disabled, training continues until the budget is exhausted.

Figure 3.12: Compute & pricing details for model training of image

1. Specify the duration model training (for image data minimum training hours is 8 node hours).

2. Enable the early stopping (to stop the training if no further improvement is observed).
3. Click on **START TRAINING**.

Step 5: Model training complete

Once the model training is complete, the dataset page will be updated as shown in *Figure 3.13*:

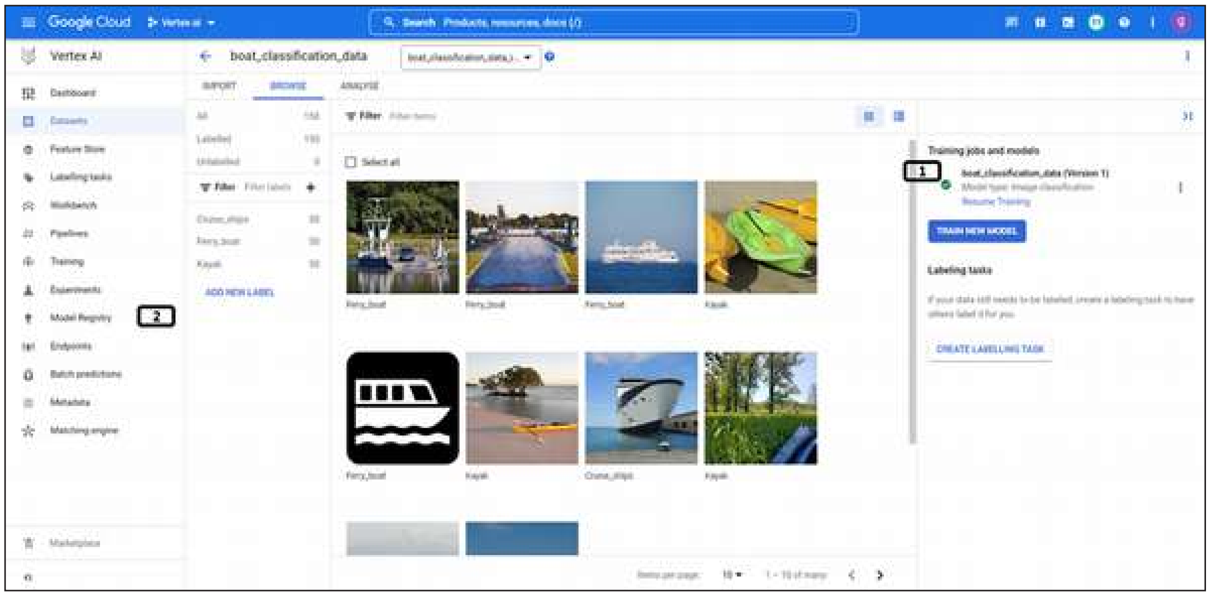


Figure 3.13: *Image model training status on the dataset page*

1. Green color tick indicates the completion of model training.
2. Click on **Model Registry**.

Step 6: Model registry

The trained model will be listed in the model registry as shown in *Figure 3.14*:

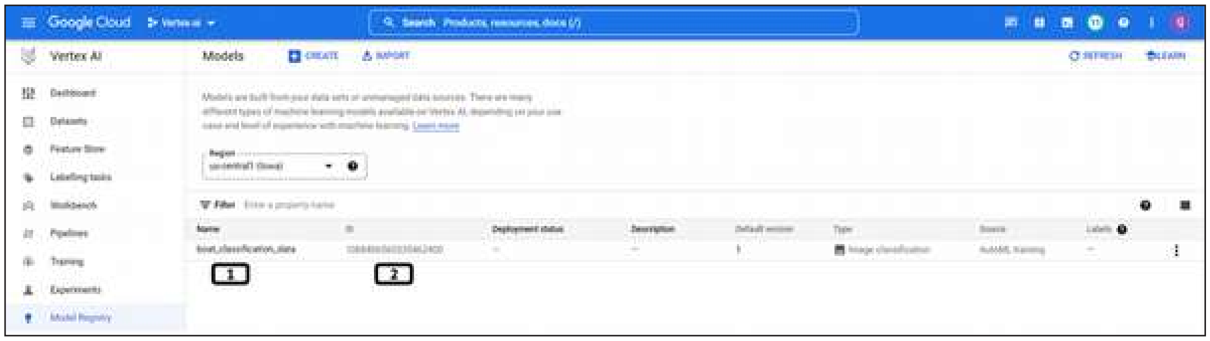


Figure 3.14: *Trained model of the image listed in the model registry*

1. Trained model is listed with a user-provided model name. Click on the model name.
2. Model ID, a unique ID for the model will be used for online predictions.

Step 7: Model version selection

Versions of the model are listed under the model registry as shown in *Figure 3.15*:

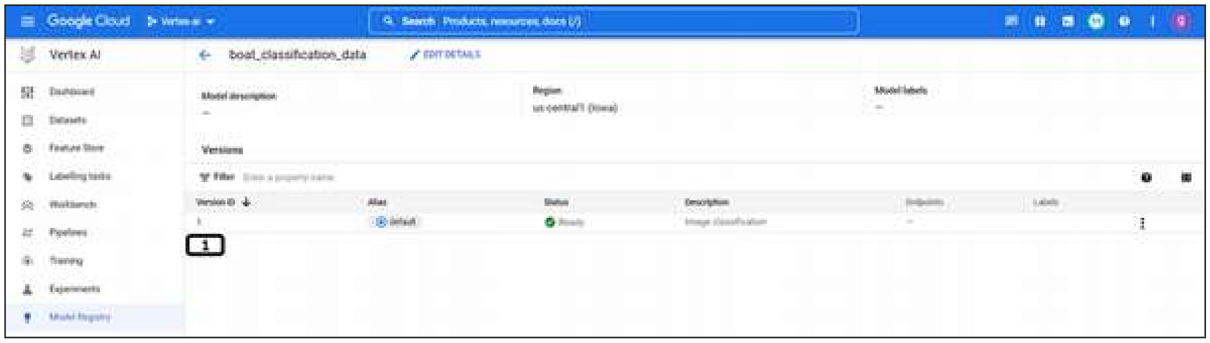


***Figure 3.15**: Image model versions on the model registry*

1. Select the version of the model (since we have trained a new model, one version is available). Click on **1**.

NOTE: Trained model will also be listed under the training module of Vertex AI as a training pipeline

Model evaluation image

In a way similar to AutoML tabular model, the AutoML image-trained model needs to be evaluated before proceeding further with batch/online predictions.

Step 1: Model performance

Trained model performance on test data is shown in *Figure 3.16*:

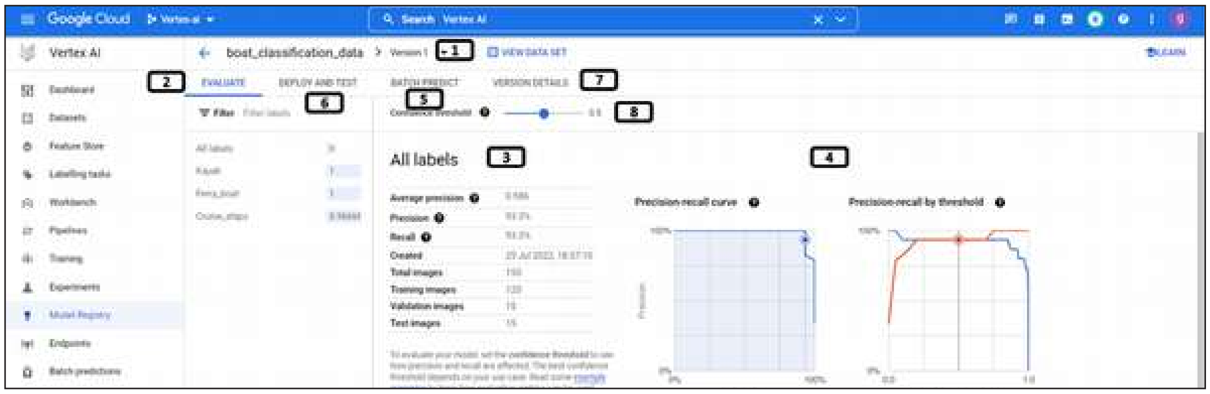


***Figure 3.16**: Model evaluation trained on an image dataset*

1. When a model is trained, it will be the first version of the model. When multiple models are trained it provides options to select the specific version of the model.
2. **EVALUATE** provides details about the class information about the target variable, users can select any specific class to check the evaluation metrics for each of the classes separately.
3. Different evaluation metrics are listed along with visual information about a few metrics. The confidence threshold can be varied to inspect the graphs at different levels of confidence.
4. Graphical representation of the evaluation metrics (varies as per the confidence threshold).
5. **BATCH PREDICT** provides options for batch prediction.
6. **DEPLOY AND TEST** provides options for online prediction.
7. **VERSION DETAILS** provides various details of the model version.

Step 2: Confusion matrix

Scroll down to get the confusion matrix as shown in *Figure 3.17*:

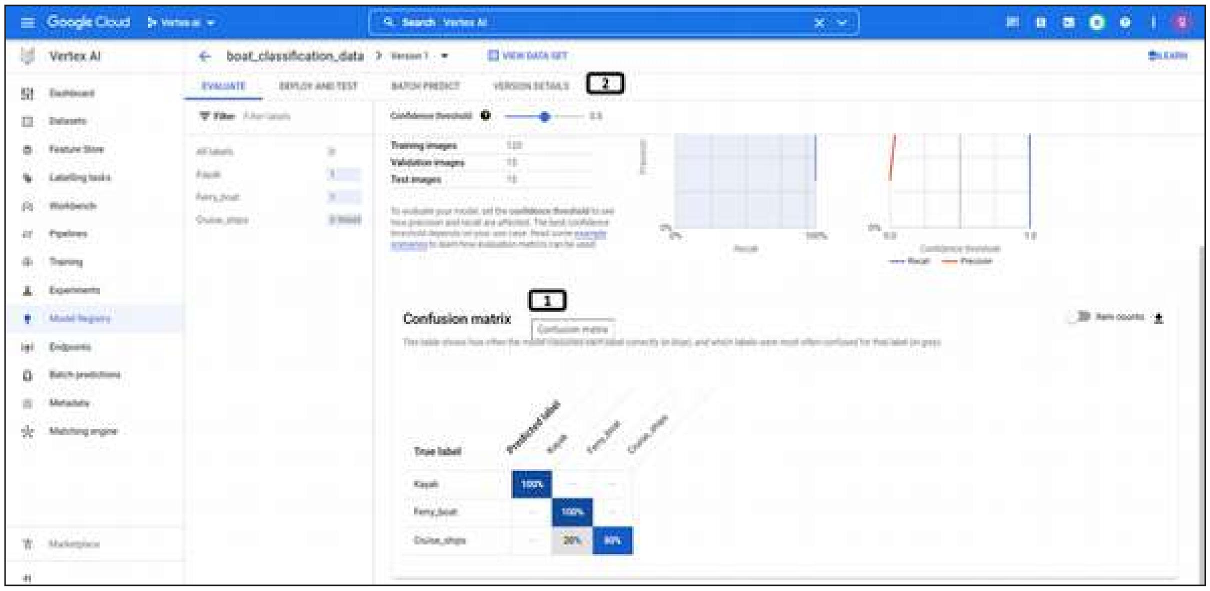


Figure 3.17: Confusion matrix of the trained model

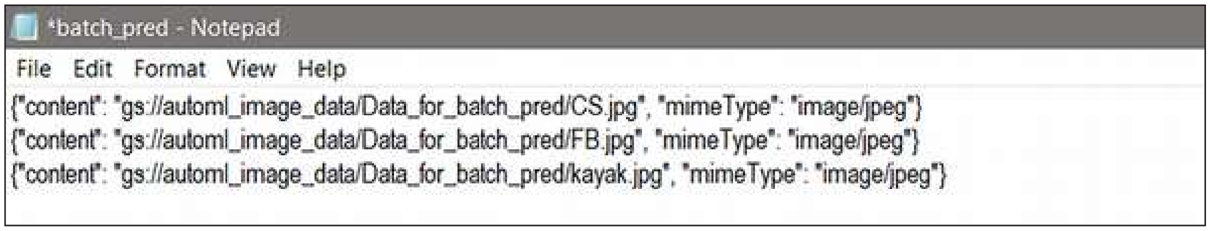
1. **Confusion matrix**.
2. **VERSION DETAILS** (Training time, training and test item details are available under the version details).

Batch Predictions image

Asynchronous prediction requests can be made without deploying the model to deployment.

Step 1: Preparing input for batch predictions

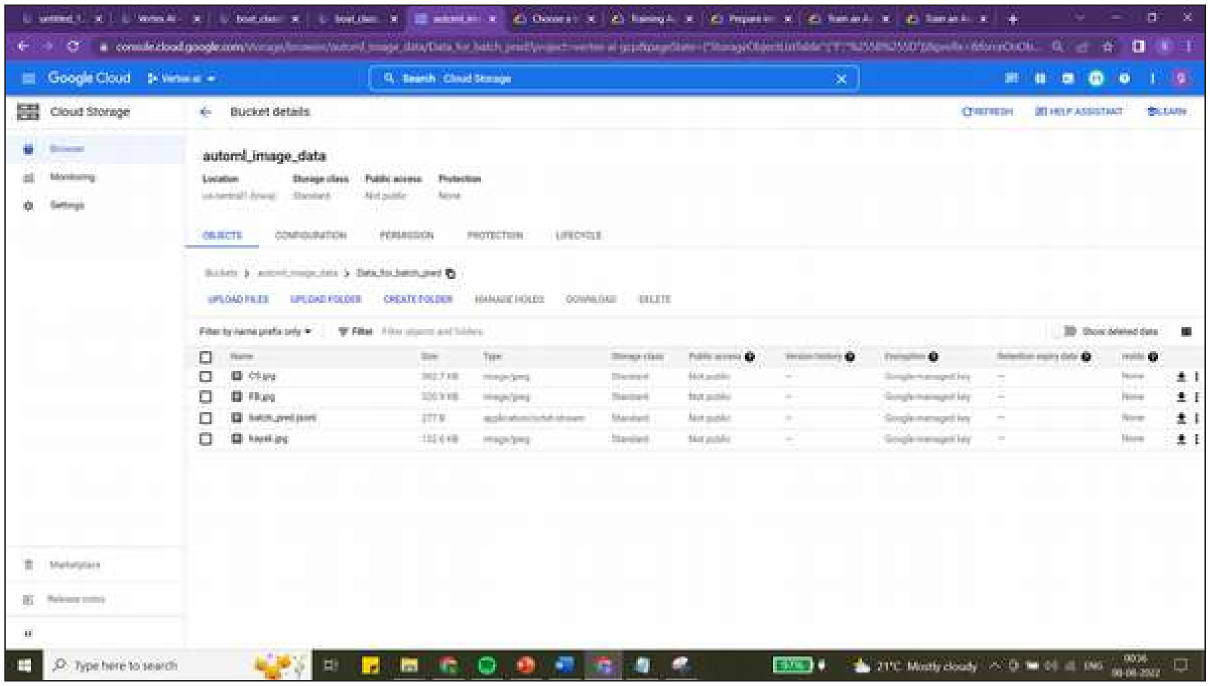
Image details and the full path of the images for batch predictions need to be prepared in the format shown in *Figure 3.18*. The file needs to be in JSONL format.



```
{"content": "gs://automl_image_data/Data_for_batch_pred/CS.jpg", "mimeType": "image/jpeg"}
{"content": "gs://automl_image_data/Data_for_batch_pred/FB.jpg", "mimeType": "image/jpeg"}
{"content": "gs://automl_image_data/Data_for_batch_pred/kayak.jpg", "mimeType": "image/jpeg"}
```

***Figure 3.18**: JSONL file containing image data for batch prediction*

JSONL file along with the images used for batch predictions are uploaded to the same bucket under the folder `Data_for_batch_pred` as shown in *Figure 3.19*:

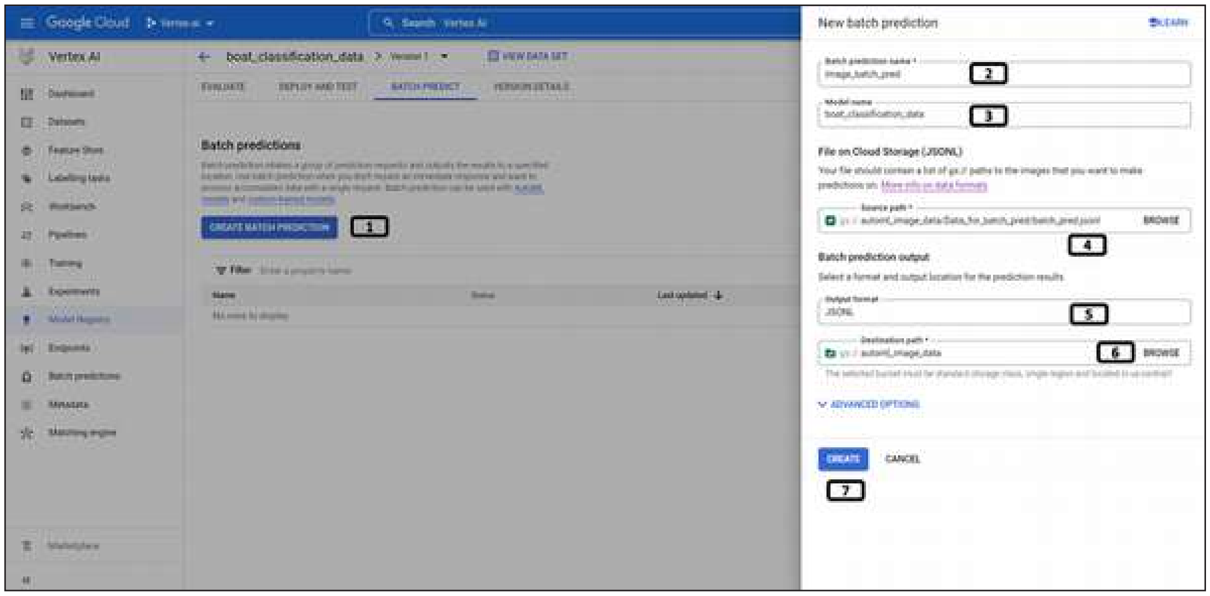


Name	Size	Type	Storage class	Public access
CS.jpg	362.7 KB	image/jpeg	Standard	Not public
FB.jpg	320.9 KB	image/jpeg	Standard	Not public
batch_pred.jsonl	277 B	application/octet-stream	Standard	Not public
kayak.jpg	152.6 KB	image/jpeg	Standard	Not public

***Figure 3.19**: JSONL file uploaded to cloud storage*

Step 2: Initializing batch prediction

Follow the steps shown in *Figure 3.20* to start the batch prediction:



New batch prediction

Batch prediction name *: image_batch_pred

Model name: boat_classification_data

File on Cloud Storage (JSONL)

Source path *: gs:// automl_image_data/Data_for_batch_pred/batch_pred.jsonl

Batch prediction output

Output format: JSONL

Destination path *: gs:// automl_image_data

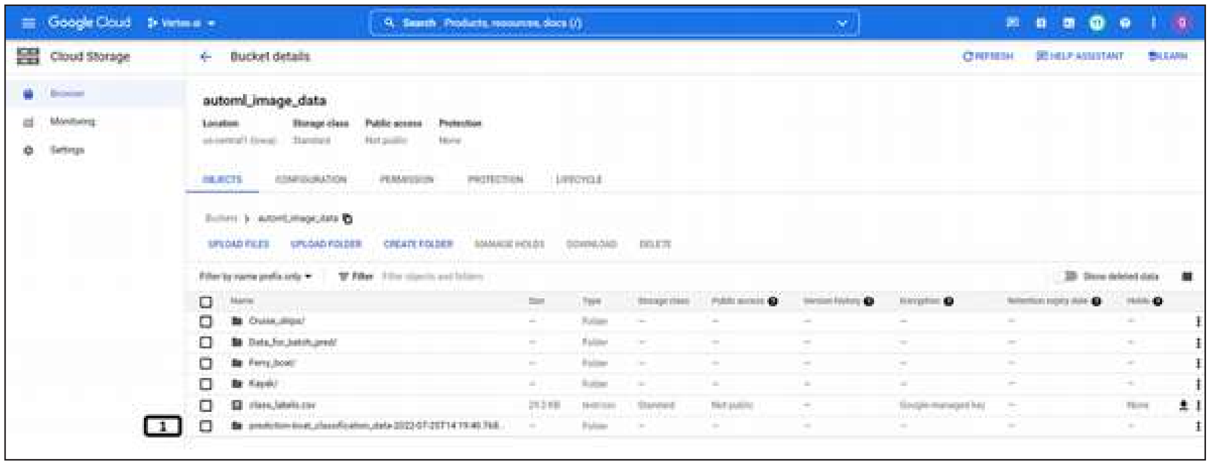
***Figure 3.20**: Batch prediction for image data*

1. Click on **CREATE BATCH PREDICTION**.
2. Provide a name for the prediction.
3. Select a model for prediction.
4. Select the JSONL file.
5. Select the **Output Format** to be JSONL.
6. Select the cloud storage bucket to store the output of the batch predictions.
7. Click on **CREATE**.

Batch predictions may take from a few minutes to a few hours depending on the number of images. Importantly users do not need to deploy the model for batch predictions.

Step 3: Output of batch predictions

The output of the batch predictions will be available on cloud storage as shown in *Figure 3.21*:



automl_image_data

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

Buckets › automl_image_data

Name	Size	Type	Storage class	Public access	Version history	Encryption	Retention expiry date	Holds
Cruise_ships/	—	Folder	—	—	—	—	—	—
Data_for_batch_pred/	—	Folder	—	—	—	—	—	—
Ferry_boat/	—	Folder	—	—	—	—	—	—
Kayak/	—	Folder	—	—	—	—	—	—
class_labels.csv	25.2 KB	text/csv	Standard	Not public	—	Google-managed key	—	None
prediction-boat_classification_data-2022-07-25T14:19:40.768...	—	Folder	—	—	—	—	—	—

***Figure 3.21**: Output for batch predictions stored back to cloud storage*

1. Folder containing a JSONL file will be created under the cloud storage bucket.

JSONL output can be accessed using a notepad, it contains the information regarding the input files and the confidence score of prediction for each category as shown below. Each line consists of input file information, class names, and the probability scores:

```
{"instance":{"content":"gs://AutoML_image_data/Data_for_batch_pred/FB.
jpg","mimeType":"image/jpeg"},"prediction":{"ids":["4888950415359475712
","2583107406145781760","7194793424573169664"],"displayNames":["Ferry_
boat","Kayak","Cruise_ships"],"confidences":[0.9992281,7.1564014E-4,5.623
7353E-5]}}

{"instance":{"content":"gs://AutoML_image_data/Data_for_batch_pred/
kayak.jpg","mimeType":"image/jpeg"},"prediction":{"ids":["2583107
406145781760","4888950415359475712","7194793424573169664"],"disp-
layNames":["Kayak","Ferry_boat","Cruise_ships"],"confidences":[0.9999988,
1.2133818E-6,5.55904E-9]}}

{"instance":{"content":"gs://AutoML_image_data/Data_for_batch_pred/CS.
jpg","mimeType":"image/jpeg"},"prediction":{"ids":["4888950415359475712
","7194793424573169664","2583107406145781760"],"displayNames":["Ferry_
boat","Cruise_ships","Kayak"],"confidences":[0.9019485,0.09799572,5.58072
6E-5]}}
```

Model deployment for online predictions image

Very similar to the AutoML tabular model, the AutoML image model can also be deployed to an endpoint for serving online predictions. Follow the steps below for the online predictions:

Step 1: Model Deployment

The landing page of model deployment is shown in *Figure 3.22*:

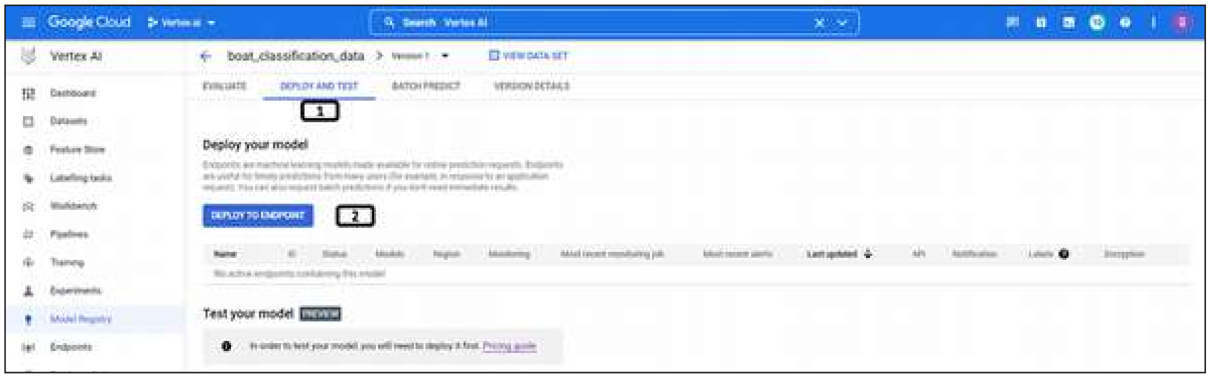


***Figure 3.22**: Model deployment page*

1. Go to the **DEPLOY AND TEST** tab.
2. Select **DEPLOY TO ENDPOINT**.

Step 2: Endpoint definition

Follow the steps shown in *Figure 3.23* to create an endpoint:

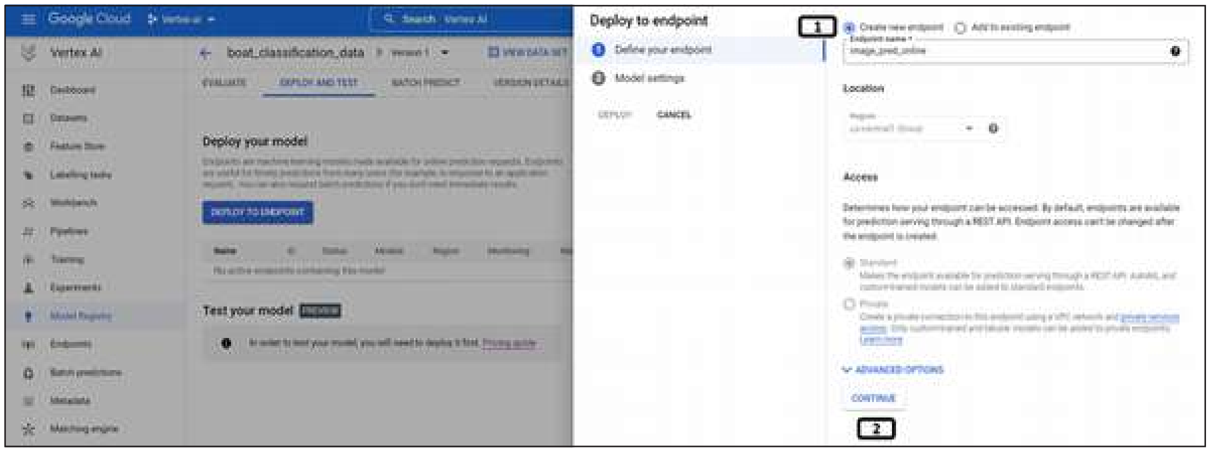


***Figure 3.23**: Endpoint creation for image classification model deployment*

1. Select **Create new endpoint**, if the model needs to be deployed on an existing endpoint users are provided with an option to **Add to existing endpoints**.
2. Click on **CONTINUE**.

Step 3: Model settings

Continue with the model settings to complete the model deployment as shown in *Figure 3.24*:

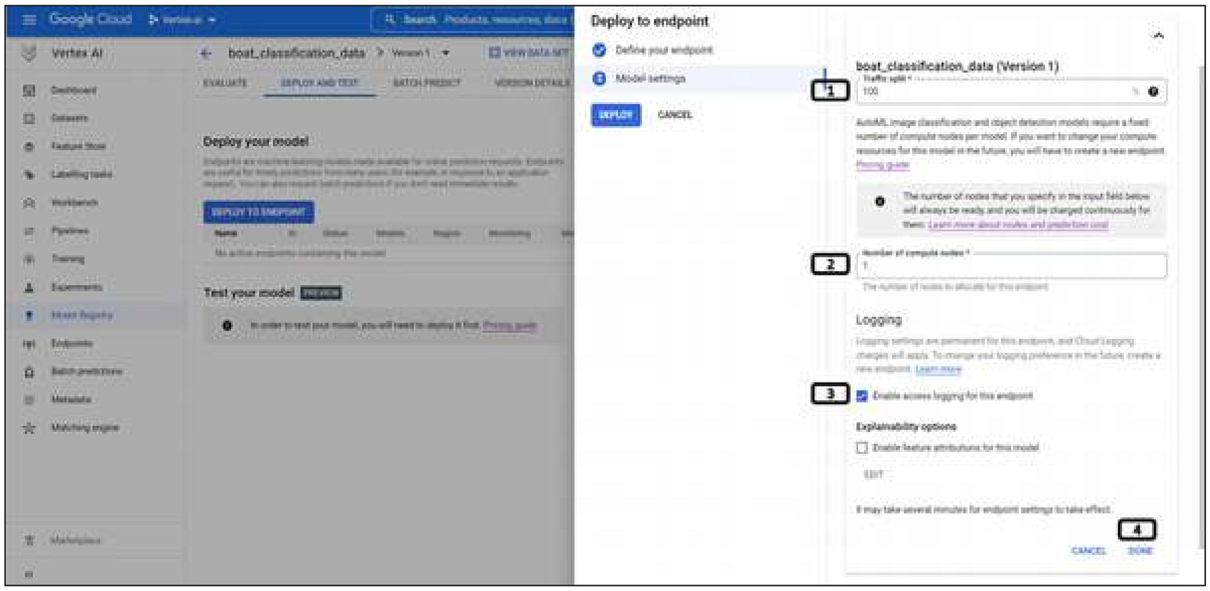


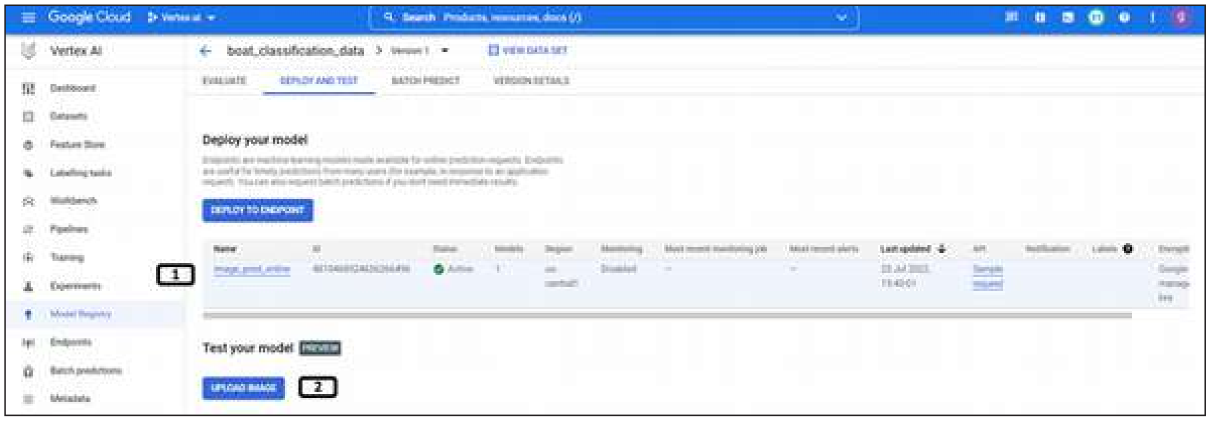
***Figure 3.24**: Deployment settings*

1. Populate **100** in the **Traffic Split** since we will be deploying the model only on 1 Node.
2. Number of nodes required for prediction – Populate **1**.
3. Enable logging for the endpoint.
4. Click on **DONE**.

Scroll down to choose the options of logging and click **CONTINUE**.

Step 4: Testing the deployed model

Follow the steps below to check the deployed model before proceeding with online predictions:

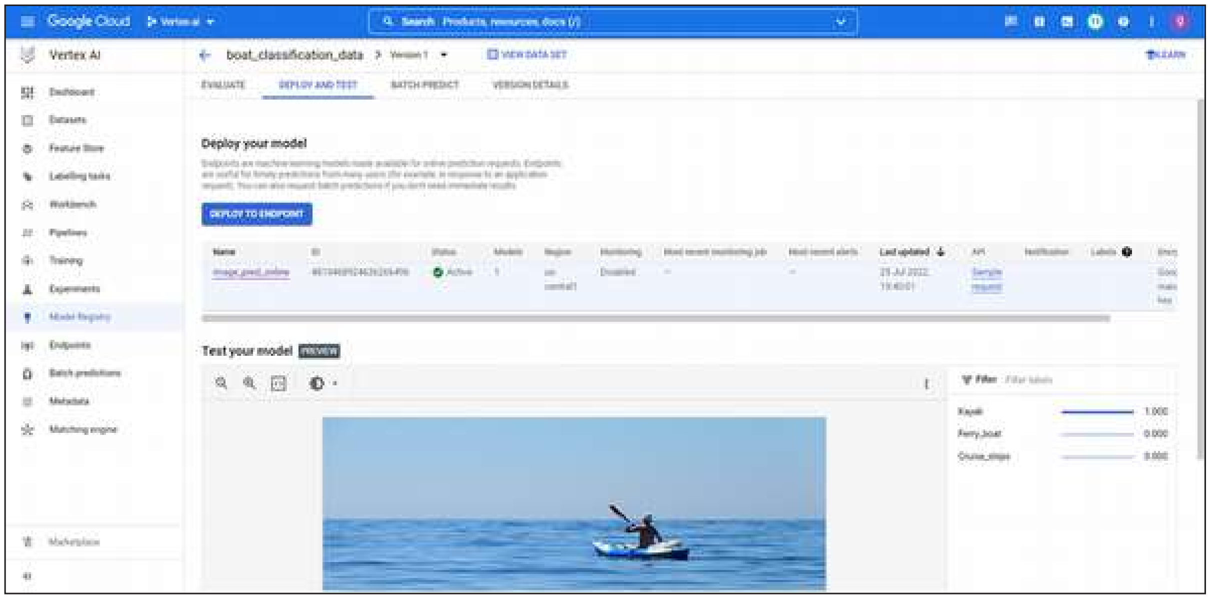


Endpoints are machine learning models made available for online prediction requests. Endpoints are useful for timely predictions from many users (for example, in response to an application request). You can also request batch predictions if you don't need immediate results.

Name	ID	Status	Models	Region	Monitoring	Most recent monitoring job	Most recent alerts	Last updated	API	Notification	Labels	Encrypt
image_pred_online	[illegible]	Active	1	us-central1	Disabled	–	–	25 Jul 2022, 11:40:01	Sample request			Google-managed key

Figure 3.25: *Image classification model deployment complete*

1. Once the endpoint is created, it will be listed in the **Model Registry** with the status active.
2. Users can upload an image to test the output. The uploaded image will be shown along with the probability score of each of the classes as shown in *Figure 3.26*:



Name	ID	Status	Models	Region	Monitoring	Most recent monitoring job	Most recent alerts	Last updated	API	Notification	Labels	Encrypt
image_pred_online	[illegible]	Active	1	us-central1	Disabled	–	–	25 Jul 2022, 11:40:01	Sample request			Google-managed key

Kayak	1.000
Ferry_boat	0.000
Cruise_ships	0.000

Figure 3.26: *Testing online prediction for image classification model*

Serving online predictions image

The model is deployed onto the GCP endpoint. Users can get the predictions from the deployed model using Python local machine. Use the same service account and keys which was created in the previous chapter.

Install the **`google-cloud-aiplatform`** python package through the following command:

```
pip install google-cloud-aiplatform
```

More information on the package can be obtained from the link below:

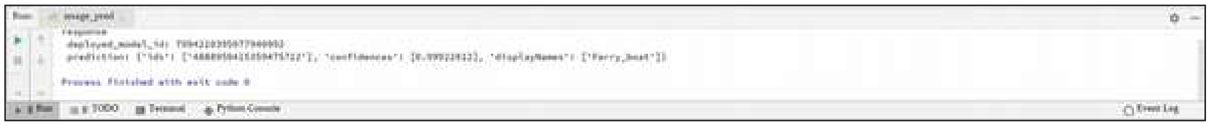
https://pypi.org/project/google-cloud-aiplatform/

Python Code for predictions

```
#Install google-cloud-aiplatform

import base64


from google.cloud import aiplatform

from google.cloud.aiplatform.gapic.schema import predict

import os

os.environ['GOOGLE_APPLICATION_CREDENTIALS'] =r"E:\service_account_json\
vertex-ai-gcp-0a71505b3eae.json"

project_ID= "vertex-ai-gcp"

model_ID= "4810468924636266496"

filename= r"E:\Publication\CHAPTERS\CHAPTER 3\DATA\Batch Pred\FB.jpg"

location= "us-central1"

api_endpoint= "us-central1-aiplatform.googleapis.com"


def predict_image_classification(

    project_ID= project_ID,

    model_ID= model_ID,

    filename= filename,

    location= location,

    api_endpoint= api_endpoint):

    client_options = {"api_endpoint": api_endpoint}

    client = aiplatform.gapic.PredictionServiceClient(client_
    options=client_options)

    with open(filename, “rb”) as f:

        file_content = f.read()

    content_encoded = base64.b64encode(file_content).decode(“utf-8”)

    instance = predict.instance.
    ImageClassificationPredictionInstance(content=content_encoded).to_
    value()

    instances = [instance]

    params= predict.params.ImageClassificationPredictionParams(confidence_
threshold=0.5, max_predictions=1).to_value()

    endpoint = client.endpoint_path(project=project_ID,
    location=location, endpoint=model_ID)

    response = client.predict(endpoint=endpoint, instances=instances,
    parameters=params)

    predictions = response.predictions

    for prediction in predictions:

        print(“ prediction:”, dict(prediction))


#Calling the function


predict_image_classification(project_ID,model_ID,filename,location,api_
endpoint)
```

Output of the Python code:

The output can be seen in *Figure 3.27*:



```
response
 deployed_model_id: [illegible]
 prediction: {'ids': ['[illegible]'], 'confidences': [[illegible]], 'displayNames': ['Ferry_boat']}

Process finished with exit code 0
```

***Figure 3.27**: Image classification model Python code output*

The output contains probability scores for each of the classes.

NOTE:

- **Project ID & other details can be obtained from the platform as explained in Chapter 2.**
- **Deploying model is cost incurring, ensure to delete the model from deployment.**
- **Users can delete the model and the dataset as described in Introduction to Vertex AI & auto ML Structure, Chapter 2.**

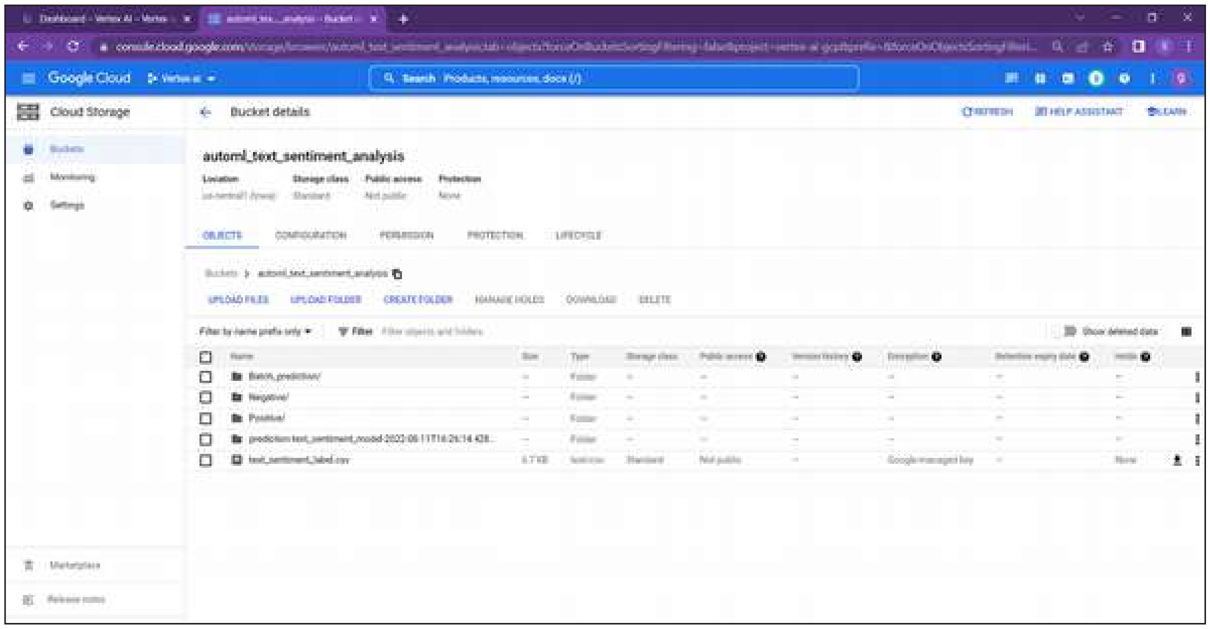
Vertex AI AutoML for text data

AutoML for text can handle tasks of single label, multi-label classification, text entity extraction, and text sentiment analysis. For the practical example, we are using sentiment analysis data which can be downloaded from below mentioned link:

https://www.kaggle.com/datasets/revanthrex/stanford-imdb-sentiment-analysis

Dataset is listed under CC0: Public domain license and it contains movie reviews of about 25K. Reviews are divided into positive and negative reviews.

`AutoML_text_sentiment_analysis` bucket is created under **`us-centra1`** (single region) and two folders containing 50 positive and negative reviews are uploaded. An additional folder is created in the bucket by the name **`Batch_prediction`** which contains randomly chosen text reviews. A snapshot of the cloud storage is shown in *Figure 3.28*:


Bucket details

automl_text_sentiment_analysis

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

Name	Size	Type	Storage class	Public access
Batch_prediction/	—	Folder	—	—
Negative/	—	Folder	—	—
Positive/	—	Folder	—	—
prediction-text_sentiment_model-2022-06-11T16:26:14.428...	—	Folder	—	—
text_sentiment_label.csv	4.7 KB	text/csv	Standard	Not public

Figure 3.28: Text data uploaded to cloud storage

Very similar to image dataset creation, even for sentiment analysis dataset we need to create a CSV file which contains which contains information about text belonging to training/validation/test, the full path of the image, and the category it belongs to:

```
for f in $(gsutil ls gs://AutoML_text_sentiment_analysis/Positive/); do
echo UNASSIGNED,$f,1,1;done>>text_sentiment_label.csv
```

```
for f in $(gsutil ls gs://AutoML_text_sentiment_analysis/Negative/); do
echo UNASSIGNED,$f,0,1;done>>text_sentiment_label.csv
```

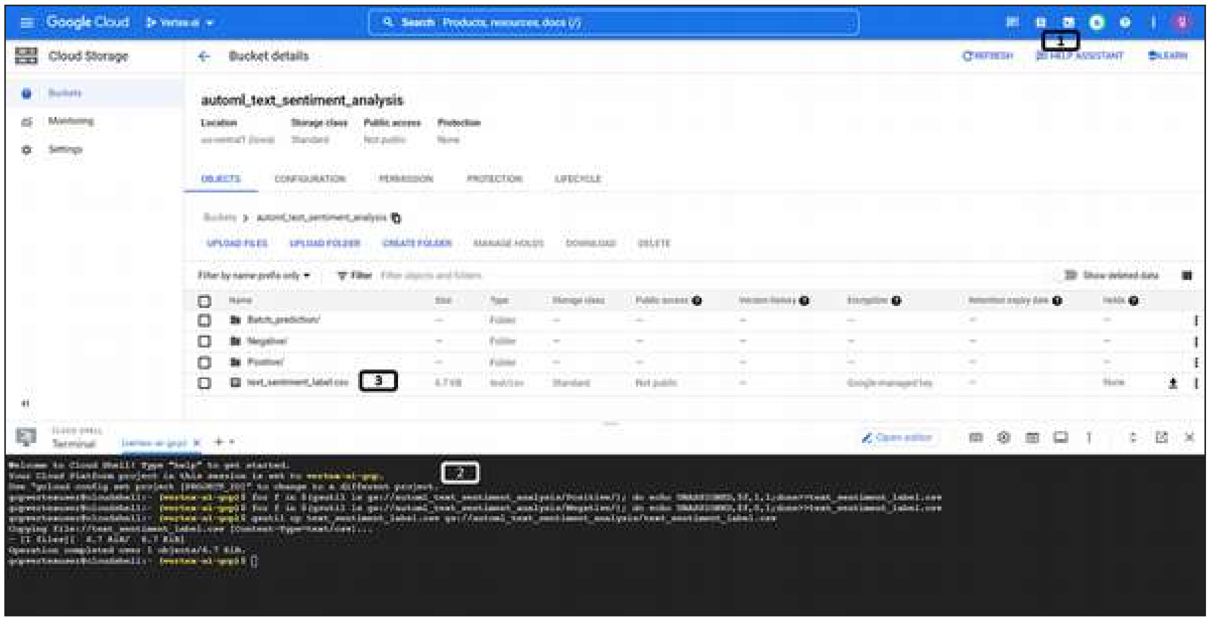
These commands will create a CSV file that contains three columns:

- Setting a text file to train/validation/test. In our case, we have used the "UNASSIGNED", platform that will choose images for training, validation, and testing.
- Full path of the text file.
- The category it belongs to.

Use the below command to push the CSV file to the cloud storage bucket.

```
Gsutil cp text_sentiment_label.csv gs://AutoML_text_sentiment_analysis/
text_sentiment_label.csv
```

Open the command shell and type the commands as shown in *Figure 3.29*:



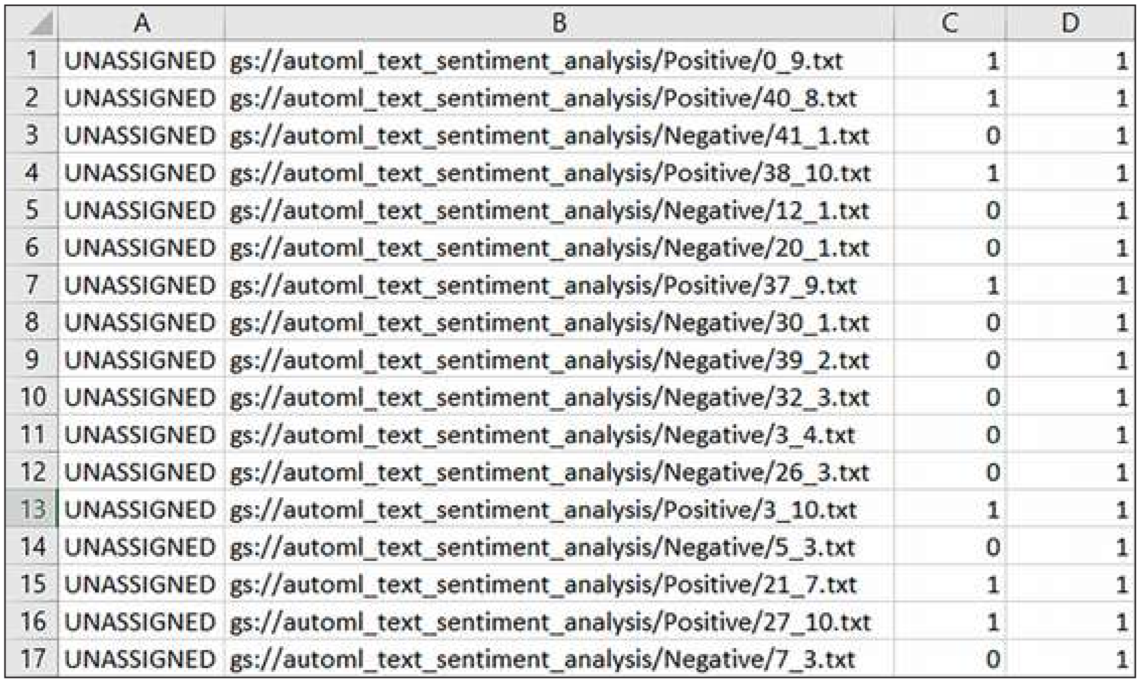
```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to vertex-ai-gcp.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
[illegible]@cloudshell:~ (vertex-ai-gcp)$ for f in $(gsutil ls gs://automl_text_sentiment_analysis/Positive/); do echo UNASSIGNED,$f,1,1;done>>text_sentiment_label.csv
[illegible]@cloudshell:~ (vertex-ai-gcp)$ for f in $(gsutil ls gs://automl_text_sentiment_analysis/Negative/); do echo UNASSIGNED,$f,0,1;done>>text_sentiment_label.csv
[illegible]@cloudshell:~ (vertex-ai-gcp)$ gsutil cp text_sentiment_label.csv gs://automl_text_sentiment_analysis/text_sentiment_label.csv
Copying file://text_sentiment_label.csv [Content-Type=text/csv]...
- [1 files][  4.7 KiB/  4.7 KiB]
Operation completed over 1 objects/4.7 KiB.
[illegible]@cloudshell:~ (vertex-ai-gcp)$
```

***Figure 3.29**: gs commands for csv creation (text data)*

1. Click on the icon to open the cloud shell.
2. Type the commands (type one command and press *Enter*).

3. New CSV is created and is available in the cloud storage.

A snapshot of the CSV file created is shown in *Figure 3.30*:



	A	B	C	D
1	UNASSIGNED	gs://automl_text_sentiment_analysis/Positive/0_9.txt	1	1
2	UNASSIGNED	gs://automl_text_sentiment_analysis/Positive/40_8.txt	1	1
3	UNASSIGNED	gs://automl_text_sentiment_analysis/Negative/41_1.txt	0	1
4	UNASSIGNED	gs://automl_text_sentiment_analysis/Positive/38_10.txt	1	1
5	UNASSIGNED	gs://automl_text_sentiment_analysis/Negative/12_1.txt	0	1
6	UNASSIGNED	gs://automl_text_sentiment_analysis/Negative/20_1.txt	0	1
7	UNASSIGNED	gs://automl_text_sentiment_analysis/Positive/37_9.txt	1	1
8	UNASSIGNED	gs://automl_text_sentiment_analysis/Negative/30_1.txt	0	1
9	UNASSIGNED	gs://automl_text_sentiment_analysis/Negative/39_2.txt	0	1
10	UNASSIGNED	gs://automl_text_sentiment_analysis/Negative/32_3.txt	0	1
11	UNASSIGNED	gs://automl_text_sentiment_analysis/Negative/3_4.txt	0	1
12	UNASSIGNED	gs://automl_text_sentiment_analysis/Negative/26_3.txt	0	1
13	UNASSIGNED	gs://automl_text_sentiment_analysis/Positive/3_10.txt	1	1
14	UNASSIGNED	gs://automl_text_sentiment_analysis/Negative/5_3.txt	0	1
15	UNASSIGNED	gs://automl_text_sentiment_analysis/Positive/21_7.txt	1	1
16	UNASSIGNED	gs://automl_text_sentiment_analysis/Positive/27_10.txt	1	1
17	UNASSIGNED	gs://automl_text_sentiment_analysis/Negative/7_3.txt	0	1

***Figure 3.30**: Snapshot of csv created (for text data)*

Column C contains value 0 or 1. 1 indicates a positive sentiment and 0 indicates negative sentiment. In the sentiment analysis of AutoML, Column C can integer value between 0 to 10 (the maximum sentiment value is the users' choice, in our case 1 is the maximum value). If users want to capture more granularity, such as 10 levels of sentiment, users can label the documents from 0 (most negative) to 9 (most positive).

Text dataset creation

AutoML model training for sentiment analysis can be initiated only after creating the dataset. Follow the below steps to create an image dataset for image classification:

Step 1: Dataset creation

Navigate to the dataset module of vertex AI as shown in *Figure 3.31*:

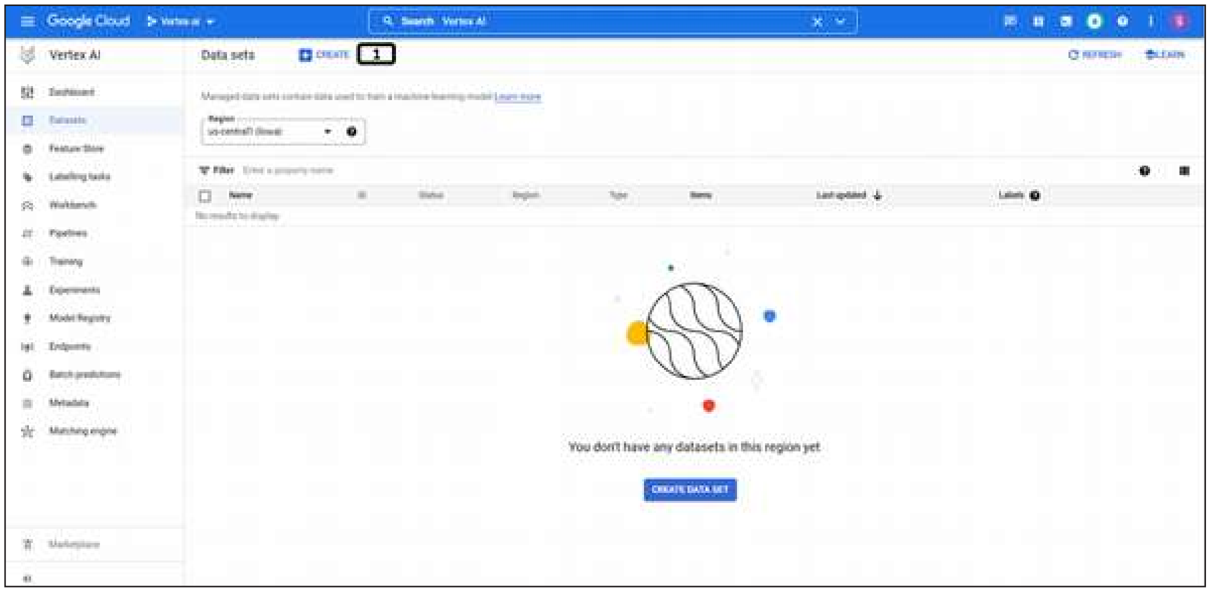


Figure 3.31: *Landing page of the dataset*

1. Click on **CREATE**.

Step 2: Objective selection

The dataset creation tab will appear as shown the *Figure 3.32*:

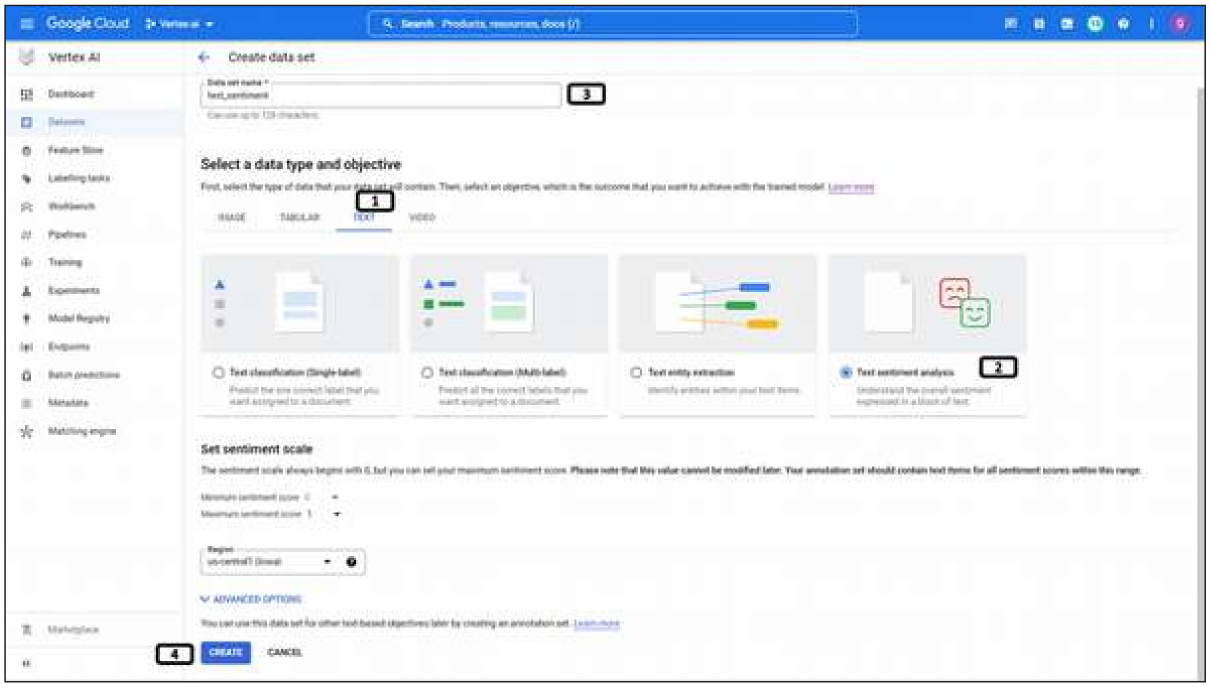


Figure 3.32: *Dataset selection for text*

1. Select the **Text** tab.

2. Select sentiment analysis.
3. Provide a name for the dataset.
4. Click on **CREATE**.

Step 3: Data selection

Data can be selected from cloud storage or local system. Follow the steps mentioned in *Figure 3.33* to upload images from the cloud storage:

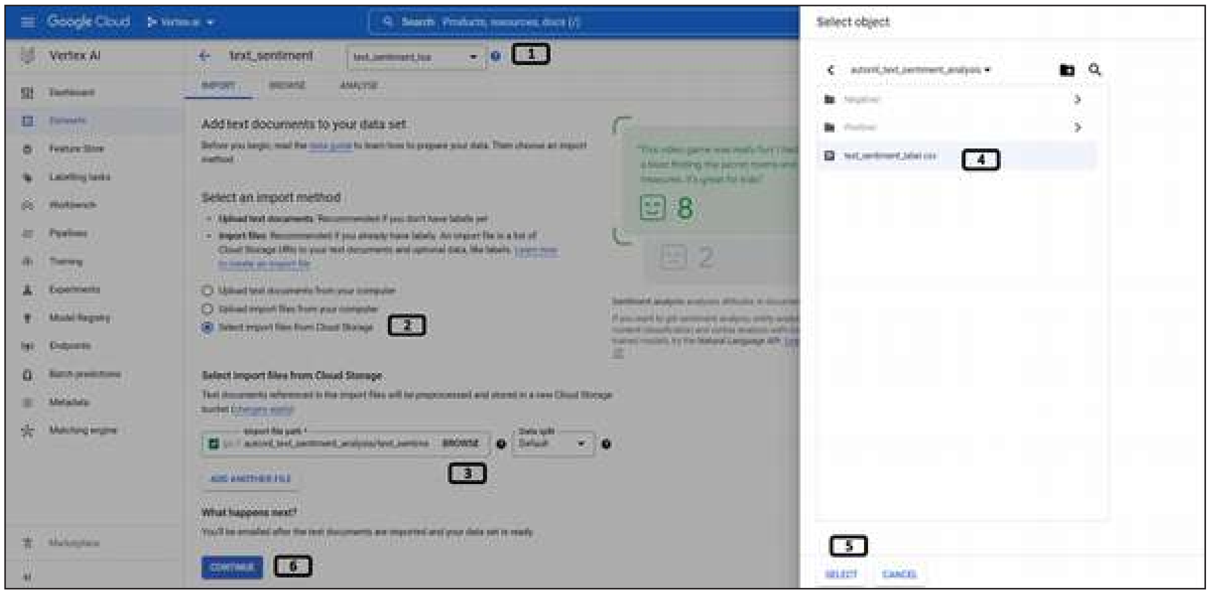


Figure 3.33: Data selection for text data

1. Labels are stored in an annotation set which is associated with the model objective and will be automatically populated.
2. Select to import files from the cloud storage.
3. Click **BROWSE**.
4. Choose the CSV file created (refer to *Figure 3.29*).
5. Click **SELECT**.
6. Click **CONTINUE**.

Step 4: Data importing complete

Once the data importing is complete, data (images) will reflect as shown in *Figure 3.34*:

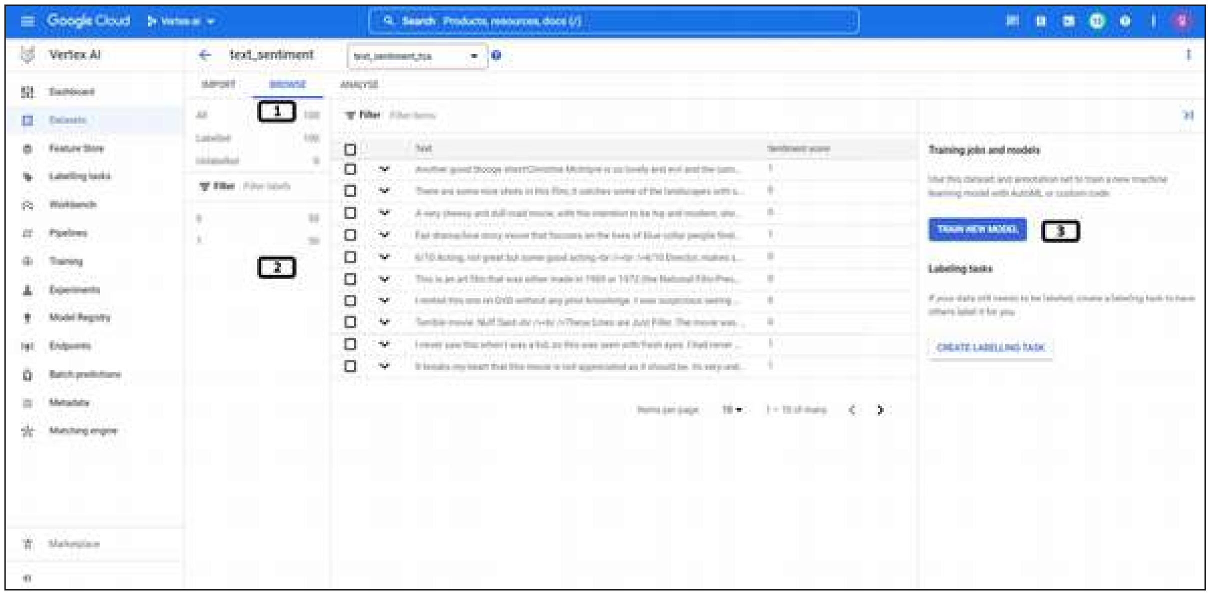


Figure 3.34: Text dataset data import complete

1. Navigate to the **BROWSE** tab.
2. Images uploaded are categorized as per the input CSV file.
3. Click **TRAIN NEW MODEL** to start model training.

Once the dataset is created, it will be listed in the dataset module of the Vertex AI.

NOTE:

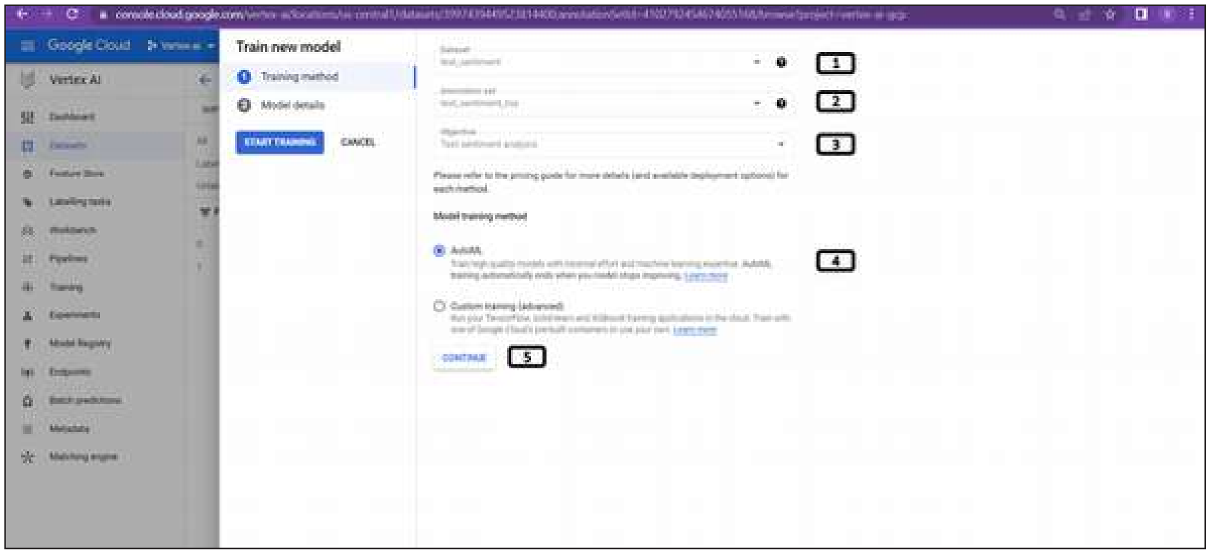
- **It is required to provide at least 10 and not exceeding 100,000 total training documents. The recommendation is to have at least 100 documents per sentiment value.**
- **Sentiment score values must be consecutive integers starting from zero.**
- **Equal proportion of documents for each sentiment score**

Model training text

Once the dataset is created and the data is uploaded to the dataset, we can start training the AutoML model. Follow the below steps to initiate the model training:

Step 1: Model training method

Users will be navigated to the page as shown in *Figure 3.35* after clicking on **Train new model**:



Dataset: text_sentiment

Annotation set: text_sentiment_ma

Objective: Text sentiment analysis

Please refer to the pricing guide for more details (and available deployment options) for each method.

Model training method

(●) AutoML
Train high-quality models with minimal effort and machine learning expertise. AutoML training automatically ends when you model stops improving. Learn more

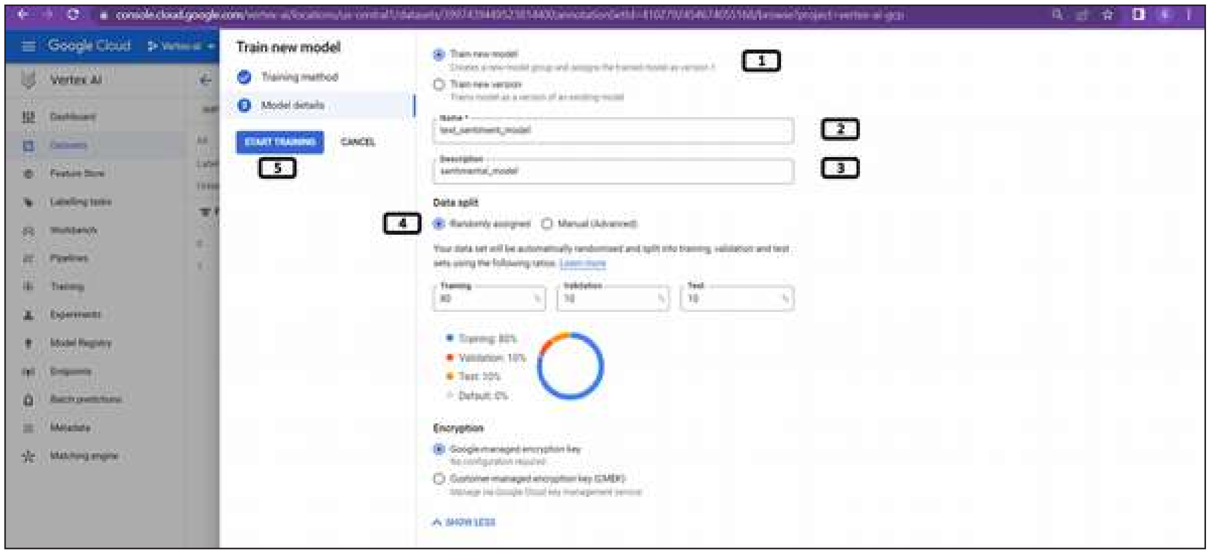
( ) Custom training (advanced)
Run your TensorFlow, scikit-learn and XGBoost training applications in the cloud. Train with one of Google Cloud's prebuilt containers or use your own. Learn more

Figure 3.35: *Training method selection for text dataset*

1. Created dataset will automatically be populated.
2. **Annotation set** will also be automatically chosen.
3. **Objective** which was chosen while creating a dataset is automatically chosen.
4. Choose **AutoML** for training.
5. Click **CONTINUE**.

Step 2: Model details

The next step in model training is users providing model details, follow the steps shown in *Figure 3.36*:



(●) Train new model
Creates a new model group and assigns the trained model as version 1
( ) Train new version
Trains model as a version of an existing model

Name *: text_sentiment_model

Description: sentimental_model

Data split
(●) Randomly assigned ( ) Manual (Advanced)

Your data set will be automatically randomized and split into training, validation and test sets using the following ratios. Learn more

Training: 80 %
Validation: 10 %
Test: 10 %

● Training: 80%
● Validation: 10%
● Test: 10%
● Default: 0%

Encryption
(●) Google-managed encryption key
No configuration required
( ) Customer-managed encryption key (CMEK)
Manage via Google Cloud key management service

Figure 3.36: *Model details for model training of text data*

1. Select **Train new model** (if a new version of the existing model is to be trained choose the below option).
2. Provide the model's name.
3. Provide the model description.
4. Choose randomly assigned (since we have given unassigned while creating CSV file). By default, 80% of the data will be used for training and 10% for validation and testing respectively. However, users can change as per their requirements.
5. Click on **START TRAINING**.

Step 3: Model training complete

Once the model training is complete, the dataset page will be updated as shown in *Figure 3.37*:

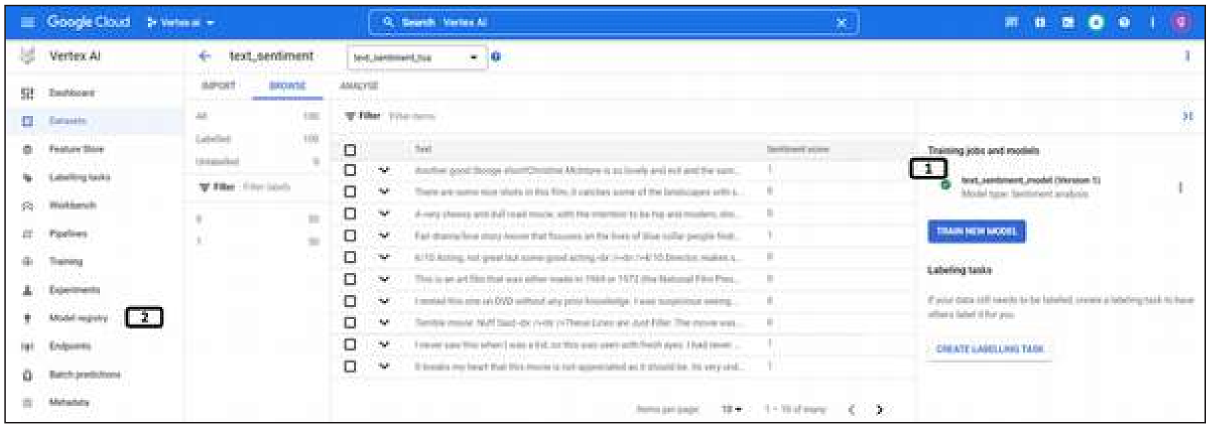


***Figure 3.37**: Model training status on the dataset page*

1. Green color tick indicates the completion of model training.
2. Click **Model registry**.

Step 4: Model registry

The trained model will be listed in the model registry as shown in *Figure 3.38*:

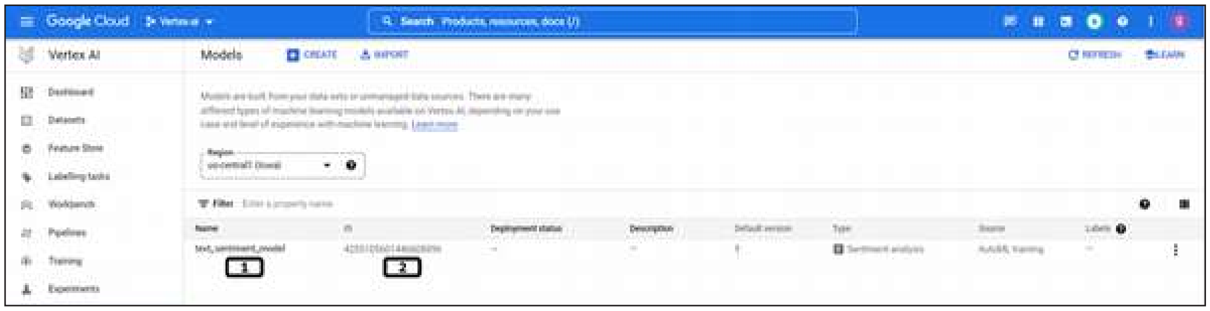


***Figure 3.38**: Trained model listed in model registry*

1. Trained model is listed with a user-provided model name. Click on the model name.
2. Model ID, a unique ID for the model (which will be used for online predictions).

Step 7: Model version selection

Versions of the model are listed under the model registry as shown in *Figure 3.39*:

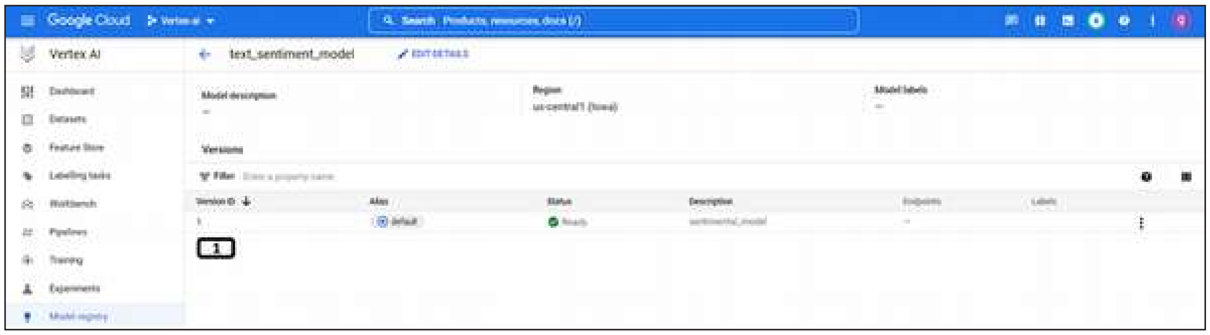


Figure 3.39*: Model versions on the model registry*

1. Select the version of the model (since we have trained a new model, one version is available). Click on **1**.

NOTE: Trained model will also be listed under the training module of Vertex AI as a training pipeline.

Model evaluation text

Any trained model needs to be evaluated before getting the predictions. Follow the below steps to evaluate the models:

Step 1: Trained model performance on test data is shown in *Figure 3.40*:

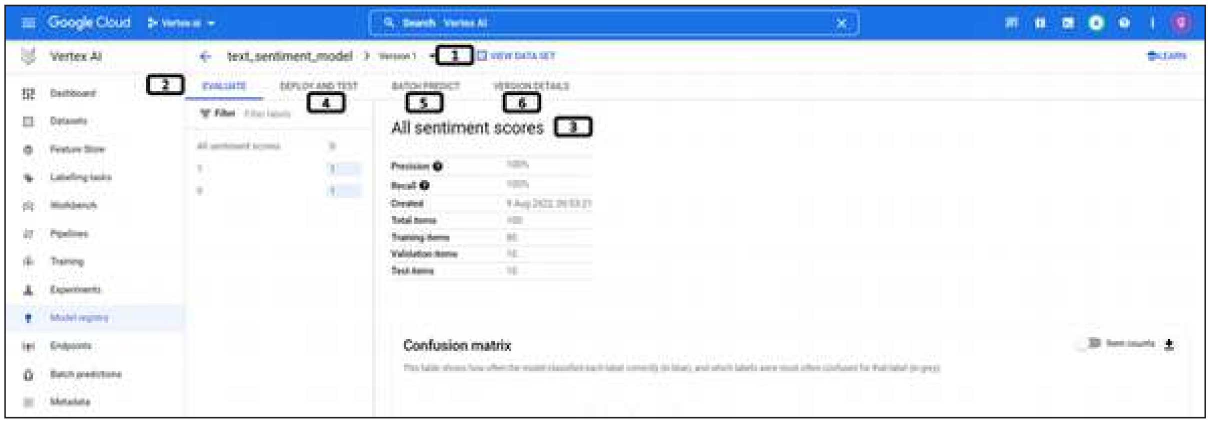


Figure 3.40*: Model evaluation trained on text dataset*

1. When a model is trained, it will be the first version of the model. When multiple models are trained, it provides options to select the specific version of the model.
2. **EVALUATE** provides details about the class information about the sentiment values, users can select any specific value to check the evaluation metrics for each of the classes separately.
3. Different evaluation metrics are listed along with visual information about a few metrics. The confidence threshold can be varied to inspect the graphs at different levels of confidence.
4. **DEPLOY AND TEST** provides options for online prediction.
5. **BATCH PREDICT** provides options for batch prediction.
6. **VERSION DETAILS** provides various details of the model version.

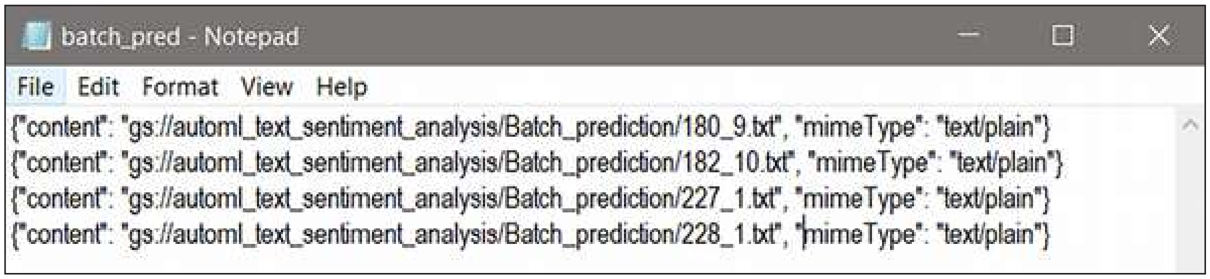
NOTE: Compare the evaluation for text, image, and tabular to see how differences in the evaluation metrics.

Batch Predictions text

It is very similar to trained models of tabular and images, text models can also be used for batch predictions without deploying them. Follow the below steps for batch predictions:

Step 1: Preparing input for batch predictions

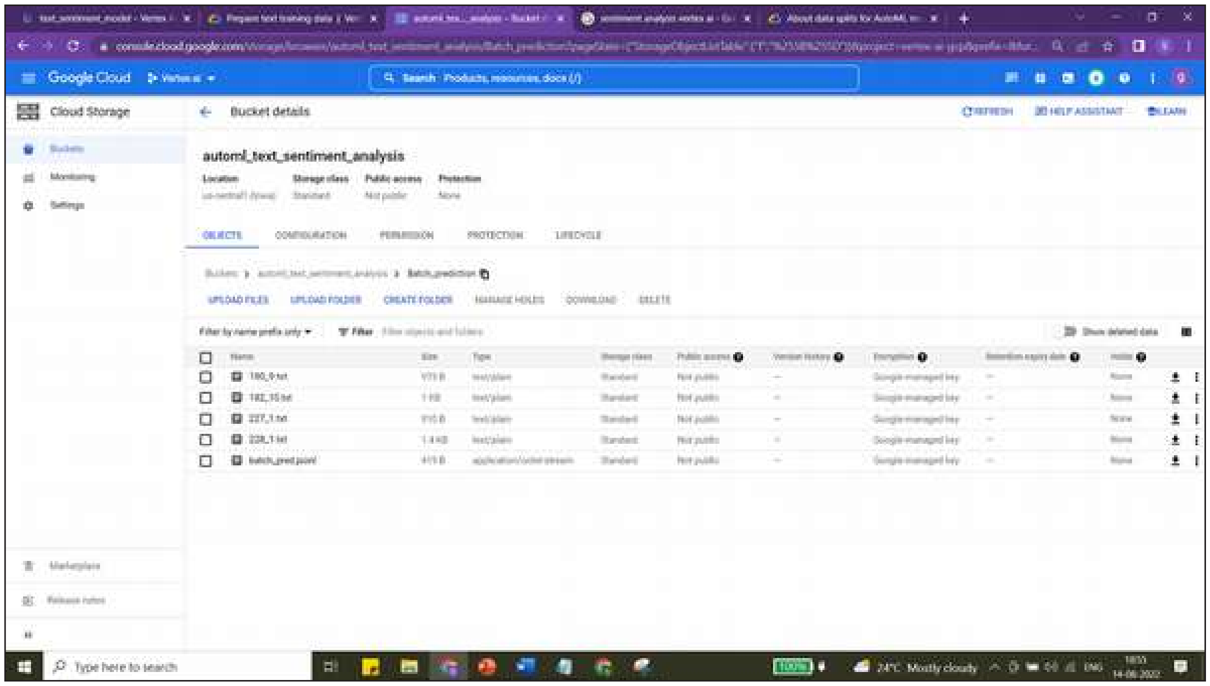
Image details and the full path of the images for batch predictions need to be prepared in the format shown in *Figure 3.41*. The file needs to be in JSONL format.



```
{"content": "gs://automl_text_sentiment_analysis/Batch_prediction/180_9.txt", "mimeType": "text/plain"}
{"content": "gs://automl_text_sentiment_analysis/Batch_prediction/182_10.txt", "mimeType": "text/plain"}
{"content": "gs://automl_text_sentiment_analysis/Batch_prediction/227_1.txt", "mimeType": "text/plain"}
{"content": "gs://automl_text_sentiment_analysis/Batch_prediction/228_1.txt", "mimeType": "text/plain"}
```

***Figure 3.41**: JSONL file containing text data for batch prediction*

Each line in the JSONL file represents a file for prediction. JSONL file along with the text files used for batch predictions are uploaded to the same bucket under the folder **`Batch_prediction`** as shown in *Figure 3.42*:



automl_text_sentiment_analysis

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

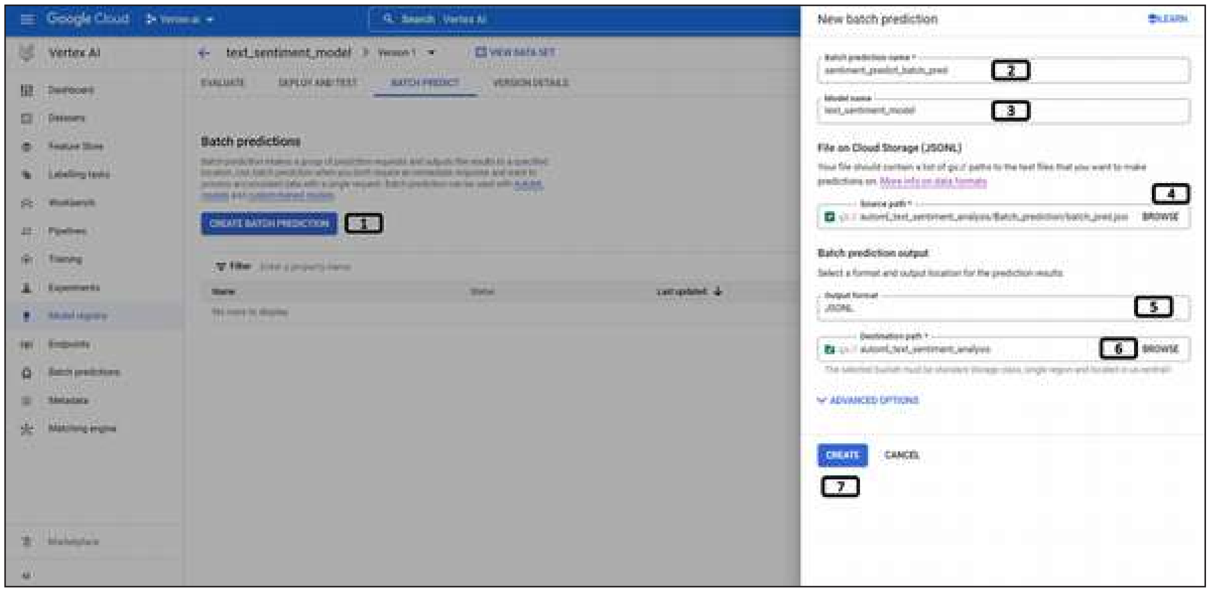
Buckets › automl_text_sentiment_analysis › Batch_prediction

Name	Size	Type	Storage class	Public access	Version history	Encryption	Retention expiry date	Holds
180_0.txt	173 B	text/plain	Standard	Not public	—	Google-managed key	—	None
182_15.txt	1 KB	text/plain	Standard	Not public	—	Google-managed key	—	None
227_1.txt	915 B	text/plain	Standard	Not public	—	Google-managed key	—	None
228_1.txt	1.4 KB	text/plain	Standard	Not public	—	Google-managed key	—	None
batch_pred.jsonl	415 B	application/octet-stream	Standard	Not public	—	Google-managed key	—	None

***Figure 3.42**: JSONL file uploaded to cloud storage*

Step 2: Initializing batch prediction

Follow the steps shown in *Figure 3.43* to start the batch prediction:



New batch prediction

Batch prediction name *: sentiment_predict_batch_pred

Model name: text_sentiment_model

File on Cloud Storage (JSONL)

Your file should contain a list of gs:// paths to the text files that you want to make predictions on. More info on data formats

Source path *: gs:// automl_text_sentiment_analysis/Batch_prediction/batch_pred.json

Batch prediction output

Select a format and output location for the prediction results

Output format: JSONL

Destination path *: gs:// automl_text_sentiment_analysis

The selected bucket must be standard storage class, single region and located in us-central1

***Figure 3.43**: Batch prediction for text data*

1. Click **CREATE BATCH PREDICTION**.
2. Provide a name for the prediction.
3. Select a model for prediction.

4. Select the JSONL file.
5. Select the output format to be JSONL.
6. Select the cloud storage bucket to store the output of the batch predictions.
7. Click **CREATE**.

Batch predictions may take from a few minutes to a few hours depending on the number of text files (and on the size of the text file).

Step 3: Output of batch predictions

The output of the batch predictions will be available on cloud storage as shown in *Figure 3.44*:

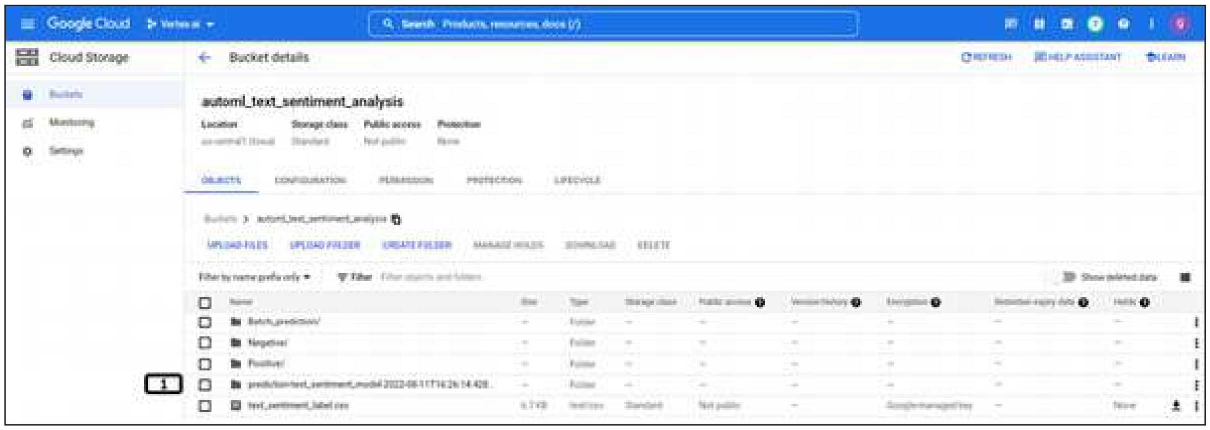


***Figure 3.44**: Output for batch predictions stored back to cloud storage*

1. Folder containing a JSONL file will be created under the cloud storage bucket.

JSONL output can be accessed using a notepad, it contains the information regarding the input files and the confidence score of prediction for each category as shown below:

```
{“instance”:{“content”:”gs://AutoML_text_sentiment_analysis/Batch_prediction/228_1.txt”,”mimeType”:”text/plain”},”prediction”:{“sentiment”:1}}

{“instance”:{“content”:”gs://AutoML_text_sentiment_analysis/Batch_prediction/227_1.txt”,”mimeType”:”text/plain”},”prediction”:{“sentiment”:1}}
```

```
{"instance":{"content":"gs://AutoML_text_sentiment_analysis/Batch_pre-
diction/182_10.txt","mimeType":"text/plain"},"prediction":{"senti-
ment":1}}
```

```
{"instance":{"content":"gs://AutoML_text_sentiment_analysis/Batch_pre-
diction/180_9.txt","mimeType":"text/plain"},"prediction":{"senti-
ment":0}}
```

NOTE: Unlike image classification, in case of sentiment analysis probability score will not be part of the output.

Model deployment for online predictions text

Deploy the model to get the online predictions. Follow the below steps to deploy the model onto the endpoint.

Step 1: Model deployment

The landing page of model deployment is shown in *Figure 3.45*:

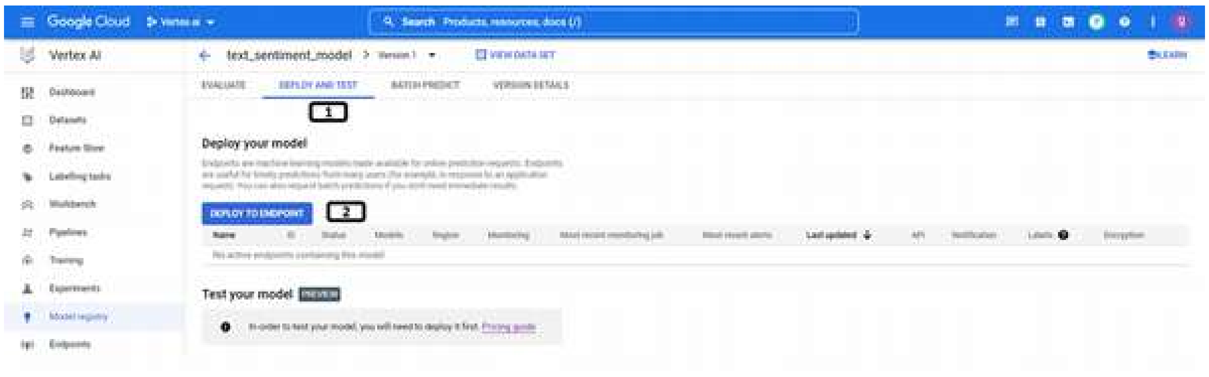


***Figure 3.45**: Model deployment page for text model*

1. Go to **DEPLOY AND TEST** tab.
2. Select **DEPLOY TO ENDPOINT**.

Step 2: Endpoint definition

Follow the steps shown in *Figure 3.46* to create an endpoint:

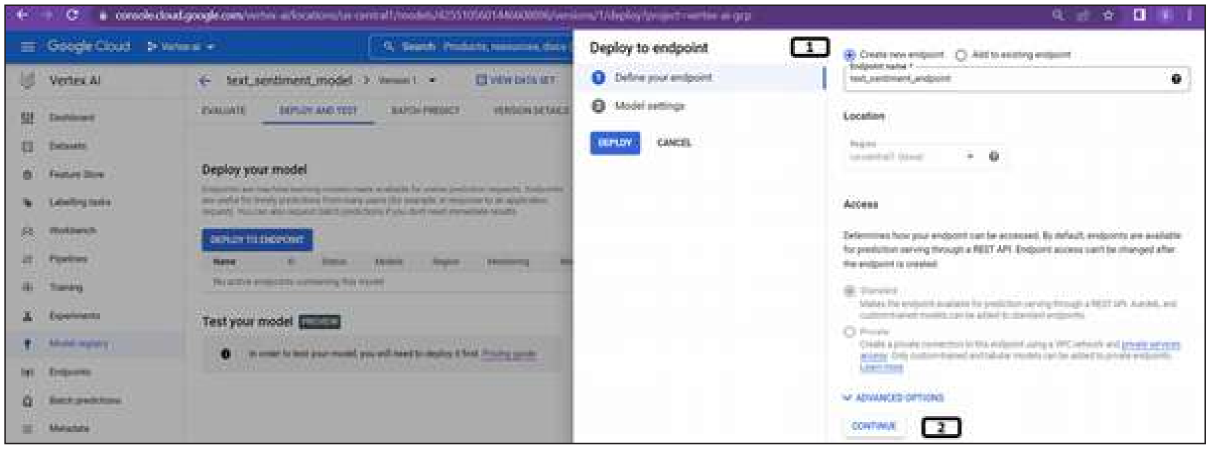


Figure 3.46: *Endpoint creation for sentiment model deployment*

1. Select **Create new endpoint**, if the model needs to be deployed to existing endpoints users are provided with an option of Add to existing endpoints.
2. Click **CONTINUE**.

Step 3: Model settings

Continue with the model settings to complete the model deployment as shown in *Figure 3.47*:

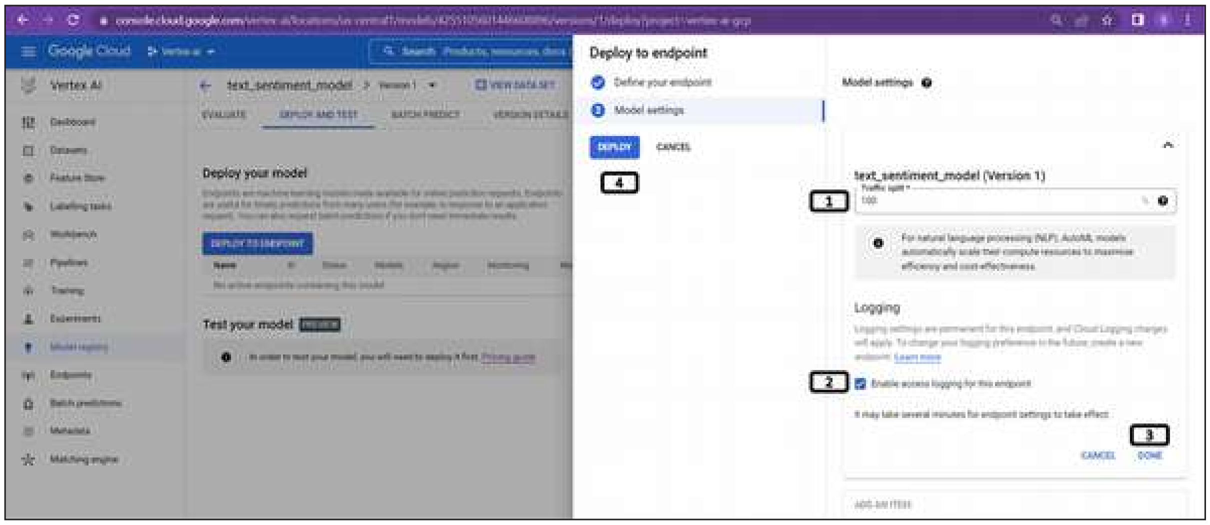
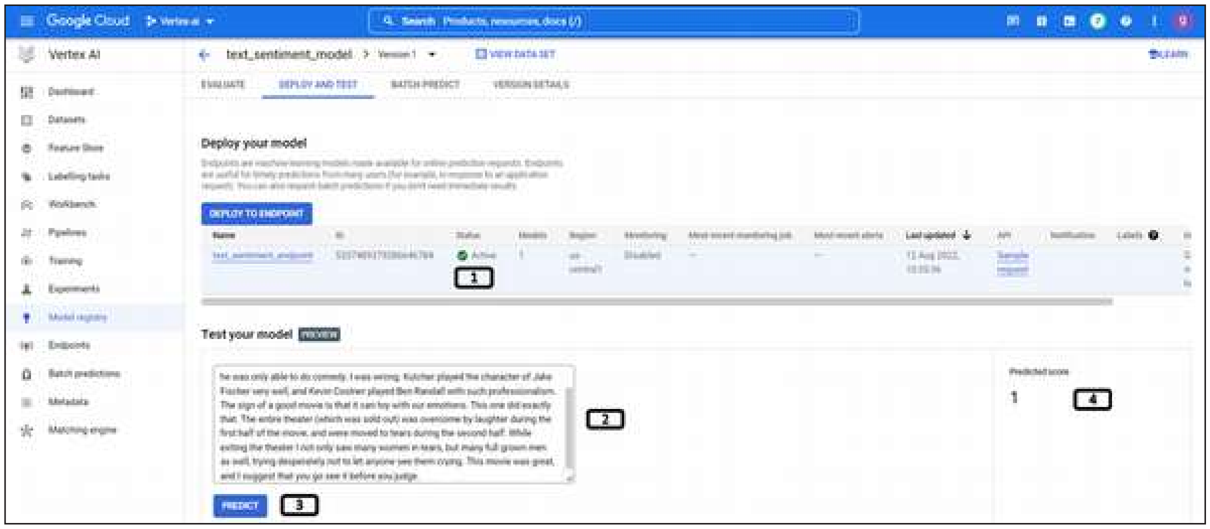


Figure 3.47: *Deployment settings*

1. Populate `100` in the traffic split since we will be deploying the model only on 1 Node.
2. Enable logging for the endpoint.
3. Click **DONE**.
4. Click **DEPLOY**.

Step 4: Testing the deployed model

Follow the below steps in *Figure 3.48* to check the deployed model before proceeding with online predictions:



Name	ID	Status	Models	Region	Monitoring	Most recent monitoring job	Most recent alerts	Last updated	API	Notification	Labels
text_sentiment_endpoint	[illegible]	Active	1	us-central1	Disabled	–	–	12 Aug 2022, [illegible]	Sample request		

he was only able to do comedy. I was wrong. Kutcher played the character of Jake Fischer very well, and Kevin Costner played Ben Randall with such professionalism. The sign of a good movie is that it can toy with our emotions. This one did exactly that. The entire theater (which was sold out) was overcome by laughter during the first half of the movie, and were moved to tears during the second half. While exiting the theater I not only saw many women in tears, but many full grown men as well, trying desperately not to let anyone see them crying. This movie was great, and I suggest that you go see it before you judge.

Figure 3.48: *Sentiment analysis model deployment complete*

1. Once the model is deployed, the status will be active.
2. Type the text in the box to check the prediction.
3. Click **PREDICT**.
4. Prediction score will be displayed.

Serving online predictions text

The model is deployed onto the GCP endpoint. Users can get the predictions from the deployed model using Python on the local machine. Use the same service account and keys which was created in the previous chapter.

Install the **`google-cloud-aiplatform`** python package through the below command.

```
pip install google-cloud-aiplatform
```

More information on the package can be obtained from the below link.

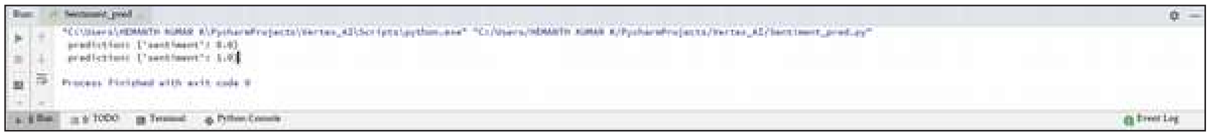
https://pypi.org/project/google-cloud-aiplatform/

Python Code for predictions

```
import base64

from google.cloud import aiplatform
from google.cloud.aiplatform.gapic.schema import predict

from google.protobuf import json_format

from google.protobuf.struct_pb2 import Value

import os

os.environ[‘GOOGLE_APPLICATION_CREDENTIALS’] =r”” # Full path of the
service account key (Json file)

project_ID= “vertex-ai-gcp”

endpoint_id= “5207489379286646784”

content_path= r”E:\Publication\CHAPTERS\CHAPTER 3\DATA\Sentiment
analysis\Test”

location= “us-central1”

api_endpoint= “us-central1-aiplatform.googleapis.com”


def predict_text_sentiment_analysis_sample(

    project_ID: str,

    endpoint_id: str,

    content_path: str,

    location: str = “us-central1”,

    api_endpoint: str = “us-central1-aiplatform.googleapis.com”):

    client_options = {“api_endpoint”: api_endpoint}

    client = aiplatform.gapic.PredictionServiceClient(client_
    options=client_options)

    for path in os.listdir(content_path):

        with open(os.path.join(content_path, path)) as f:

            content1=f.readlines()

        instance = predict.instance.
        TextSentimentPredictionInstance(content=str(content1[0]),).
        to_value()

        instances = [instance]
        parameters_dict = {}

        parameters = json_format.ParseDict(parameters_dict, Value())

        endpoint = client.endpoint_path(project=project_ID,
        location=location, endpoint=endpoint_id)

        response = client.predict(endpoint=endpoint,
        instances=instances, parameters=parameters)

        predictions = response.predictions

        for prediction in predictions:

            print(“ prediction:”, dict(prediction))

#Calling the function to predict the sentiment

predict_text_sentiment_analysis_sample(project_ID,endpoint_id,content_
path)
```

Output of the Python code

The output can be seen in *Figure 3.49*:



```
"C:\Users\HEMANTH KUMAR K\PycharmProjects\Vertex_AI\Scripts\python.exe" "C:/Users/HEMANTH KUMAR K/PycharmProjects/Vertex_AI/Sentiment_pred.py"
 prediction: {'sentiment': 0.0}
 prediction: {'sentiment': 1.0}

Process finished with exit code 0
```

***Figure 3.49**: Sentiment analysis model Python code output*

The output contains the sentiment class the text file belongs to.

NOTE: Project ID & other details can be obtained from the platform as explained in Chapter 2.

Deploying the model is cost incurring, ensure to delete the model from deployment.

Users can delete the model and the dataset as described in Chapter 2.

Pre-built models in GCP

In addition to the AutoML capabilities of the cloud platform, GCP also offers pre-trained models. Pre-trained models belonging to Natural language, vision, videos, translation, and so on, are available. Predictions from the pre-trained models can be obtained from the readily available APIs (without users deploying the models).

Under the Artificial Intelligence section of the navigation menu, these capabilities are listed as shown in *Figure 3.50*:

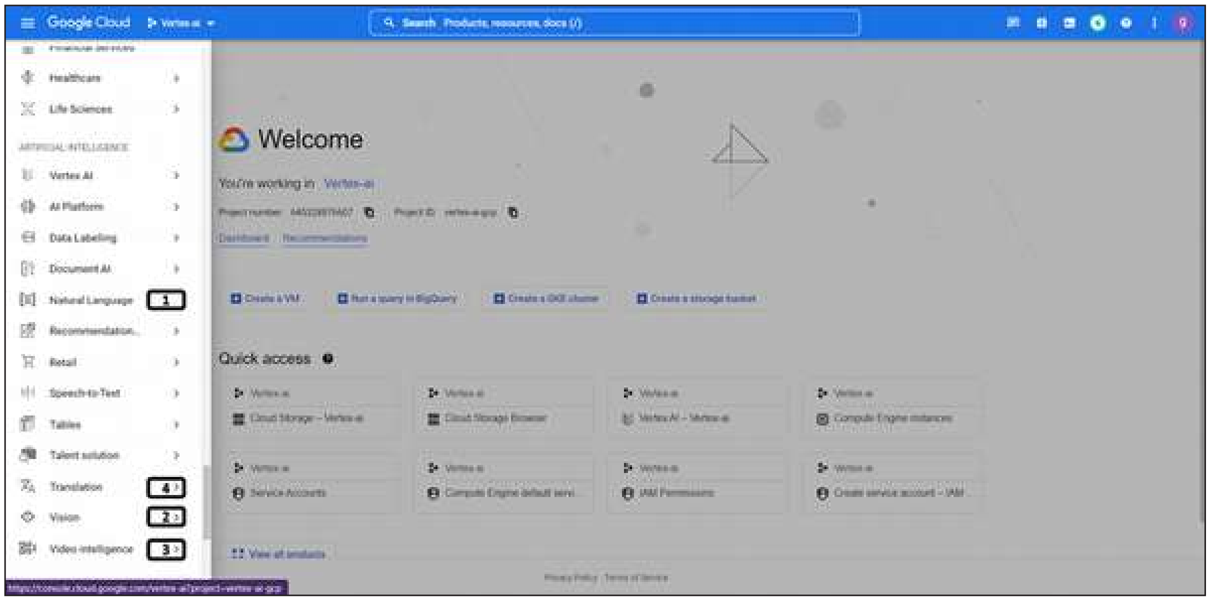
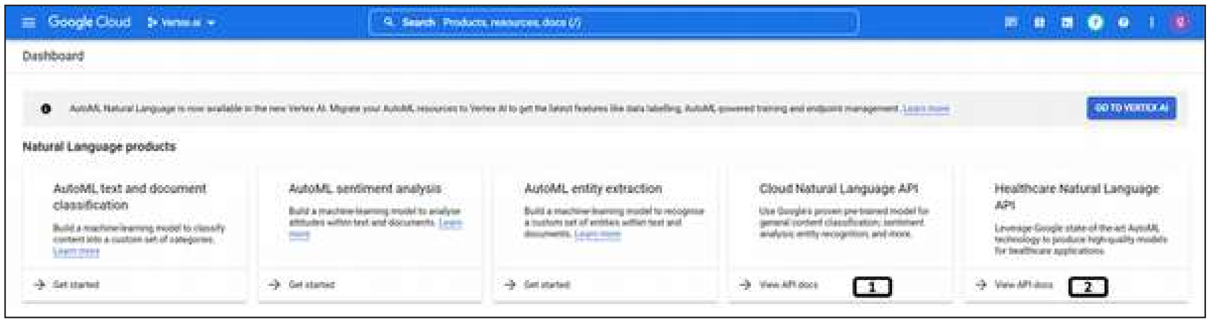


Figure 3.50: *Artificial Intelligence section in the navigation menu of GCP*

1. Pre-built models for **Natural Language**.
2. Pre-built models for images.
3. Pre-built models for videos.
4. Pre-built models for translation.

Clicking on the **Natural Language** will navigate to the page as shown in *Figure 3.51*:



AutoML Natural Language is now available in the new Vertex AI. Migrate your AutoML resources to Vertex AI to get the latest features like data labelling, AutoML-powered training and endpoint management. Learn more

AutoML text and document classification
Build a machine-learning model to classify content into a custom set of categories. Learn more

AutoML sentiment analysis
Build a machine-learning model to analyse attitudes within text and documents. Learn more

AutoML entity extraction
Build a machine-learning model to recognise a custom set of entities within text and documents. Learn more

Cloud Natural Language API
Use Google's proven pre-trained model for general content classification, sentiment analysis, entity recognition, and more.

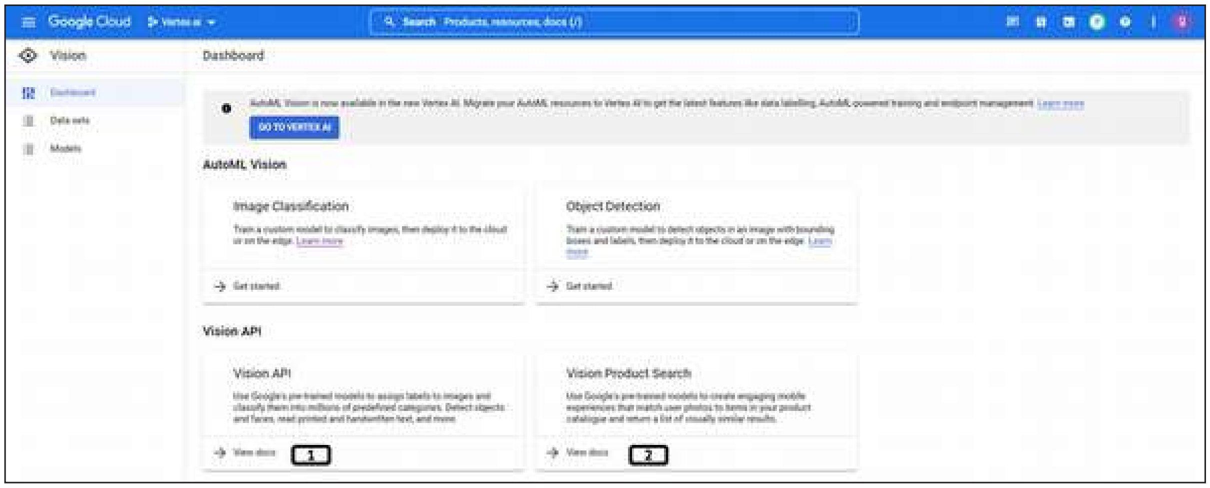
Healthcare Natural Language API
Leverage Google state-of-the-art AutoML technology to produce high-quality models for healthcare applications.

Figure 3.51: *Natural language landing page*

1. Natural language pre-trained models:
   a. They can be used for various tasks such as sentiment analysis, entity analysis, entity sentiment analysis, content classification, and syntax analysis.

2. **Healthcare Natural Language APIs** can be used for deriving insights from medical text.

Clicking on **Vision** will navigate to the page as shown in *Figure 3.52*:



AutoML Vision is now available in the new Vertex AI. Migrate your AutoML resources to Vertex AI to get the latest features like data labelling, AutoML-powered training and endpoint management. Learn more

Train a custom model to classify images, then deploy it to the cloud or on the edge. Learn more

Train a custom model to detect objects in an image with bounding boxes and labels, then deploy it to the cloud or on the edge. Learn more

Use Google's pre-trained models to assign labels to images and classify them into millions of predefined categories. Detect objects and faces, read printed and handwritten text, and more.

Use Google's pre-trained models to create engaging mobile experiences that match user photos to items in your product catalogue and return a list of visually similar results.

***Figure 3.52**: Vision landing page*

1. Vision pre-trained models:
   a. They can be used for mage labelling, face and landmark detection, optical character recognition, and so on.
2. Retailers can design products with reference photos that individually visually explain the product from a variety of angles using Vision API Product Search.

View API docs contain sample code in Python, Java, Go, and other languages which can be directly used to get the predictions from these pre-trained models. The sample code does not contain a piece of code for user authentication. Users can add the same line of code that has been used for authentication while getting the predictions from the AutoML models.

NOTE:

- **Users need to install google-cloud-language for natural language models and google-cloud-vision for vision models.**
- **Pre-trained models are trained with generic data and may not give accurate predictions for business-specific problems.**
- **Figures shown in the pre-trained models also contain the AutoML section, that is because before Vertex AI AutoML was a separate section in GCP.**

Benefits of AutoML

These are the benefits of AutoML:

- **Efficiency improvement**: The dramatic Increased productivity is one of the strongest arguments in favor of using AutoML tools. The more efficient you can be with your machine learning time, the more money you will save.
- **ML to Scale**: Instead of devoting all your time to repetitive modeling activities, you can use AutoML to focus on scaling up your machine learning applications. This allows you to address issues swiftly and effectively throughout your whole firm.
- **Bridges Skill Gaps**: This gives you a chance to fill vacuums in your organization's skill set. Given this, outsourcing your AI solutions, and making a one-time investment in an AutoML tool frequently makes more sense financially and logically than keeping an in-house staff of data scientists.
- **Promoting AI**: AutoML solutions assist business analysts and subject matter experts build guardrail-safe models. These roles no longer need model-building data scientists. AutoML can make data science more approachable and help mainstream AI.
- **Decrease in time to market**: Automating certain model development procedures may boost efficiency and reduce market time. AutoML models are not as excellent as manually customized ones, but they're no worse than basic ML models. It is also crucial to establish a footing swiftly and decisively while confronting severe competition.

Limitations of AutoML

There are a few limitations of AutoML:

- **Flexibility**: AutoML frameworks can handle most datasets, however, this generalization may leave out unusual datasets. Automated Machine Learning works well in most circumstances, but for other challenges, it may lack the accuracy and persistence of a human method.
- **Business challenges**: The goals for the present AutoML system optimization are set. Realistic challenges often include many goals, such as the requirement to draw fine distinctions between cost and decision-making. People have few opportunities to appraise this kind of multi-objective research adequately before the findings are known.

- **Processing Capacity**: AutoML tests almost every model before choosing the best method. This process utilizes several computing resources. In the manual version, the data scientist first eliminates incorrect algorithms.

Conclusion

We have covered the AutoML section of GCP in detail for images and text data in this chapter. We also talked about the benefits and limitations of AutoML. In the next chapter, we will start working on the custom models.

Questions

1. Can we have multiple objectives for model training in AutoML?
2. Can pre-built models of GCP used for all business use cases?
3. When do we choose AutoML over custom models?

Chapter 4
Vertex AI Workbench and Custom Model Training

Introduction

In the previous two chapters, we worked on AutoML for tabular, images and text data. In this chapter we will start working on custom model training. Custom model will provide more flexibility to data scientists while training the model unlike the AutoML. This chapter starts with creating a workbench on vertex AI followed by step-by-step instructions to work on custom models.

Structure

In this chapter we will discuss the following topics:

- Vertex AI workbench
- Vertex AI workbench creation and working
- Data for building custom model
- Creation of Dockerfile
- Model building
- Image creation
- Pushing image to container registry

- Submitting the custom model training job
- Completion of custom model training job
- Deletion of resources

Objectives

By the end of this chapter, you will be able to create workbench, create a Python code for custom model training, create Dockerfile, push the image to the container registry and initiate the custom training job.

Vertex AI workbench

The full data science process may be completed in a single development environment with the help of Vertex AI Workbench. Users may utilize the notebook-based environment provided by Vertex AI Workbench to do tasks such as querying and exploring data, developing and training a model, and running your code as part of a pipeline. Vertex AI workbench provides managed and user-managed notebooks.

Users may build an end-to-end notebook-based production environment with the assistance of Vertex AI Workbench's managed notebooks option, which comes with built-in integrations for your convenience. An option for user-managed notebooks is available inside Vertex AI Workbench for users that need complete command over their working environment.

Both of the available notebook alternatives come preconfigured with JupyterLab and have a whole suite of deep learning packages, including support for the TensorFlow and PyTorch frameworks. Users have the option of using instances with either a CPU or a GPU. The notebook instances provided by Vertex AI Workbench now interact with GitHub, allowing users to synchronize the notebook with a repository hosted on GitHub.

If users want to use a notebook for data exploration, analysis, or modeling, or want to utilize a notebook as a part of an end-to-end data science process, managed notebooks are often an excellent option to consider. Users can carry out workflow-oriented operations using managed notebooks instances without ever having to leave the JupyterLab interface. In addition to this, they also provide a wide variety of connectors and tools for the automation of data science process.

When users establish an instance of user-managed notebooks, users will be prompted to make some selections about the **Virtual Machine** (**VM**) that will be used. For example, when users create an instance of user-managed notebooks, users will be prompted to choose both the machine type and the framework that will be used. Users can make changes to the machine type of your instance after it has been created; however, doing so requires that your instance be restarted. Users have the ability to make manual changes, such as changing the software and package versions, on the instance of user-managed notebooks that you are using. Changing the framework that is being used on your instance is a procedure that requires additional effort.

Vertex AI Workbench creation and working

Follow these steps to create Vertex AI workbench:

Step 1: Landing page of Vertex AI

Workbench will be listed in the landing page of Vertex AI as shown in *Figure 4.1*:

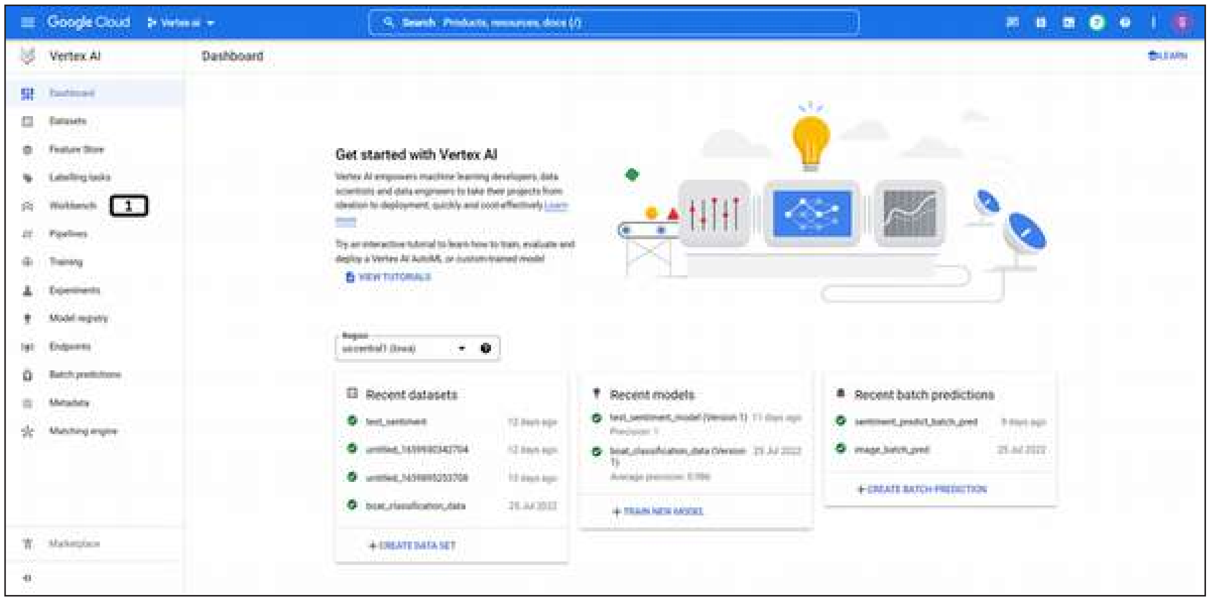
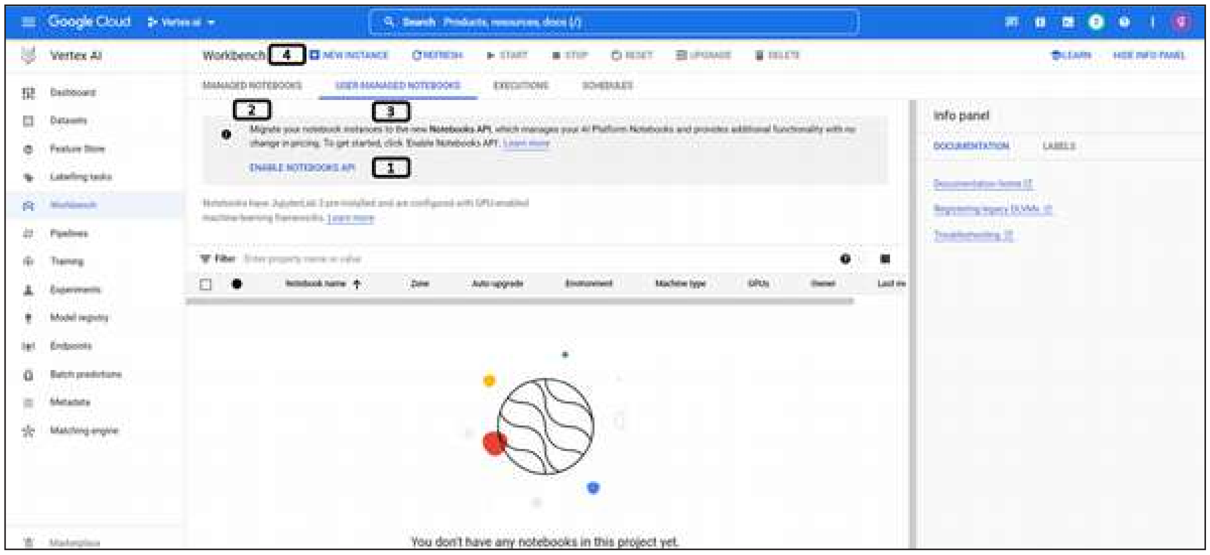


Figure 4.1*: Landing page of Vertex AI of GCP*

1. Click on **Workbench**.

Step 2: Landing page of workbench

The landing page of the workbench is shown in *Figure 4.2*:



Migrate your notebook instances to the new Notebooks API, which manages your AI Platform Notebooks and provides additional functionality with no change in pricing. To get started, click 'Enable Notebooks API'. Learn more

Notebooks have JupyterLab 3 pre-installed and are configured with GPU-enabled machine learning frameworks. Learn more

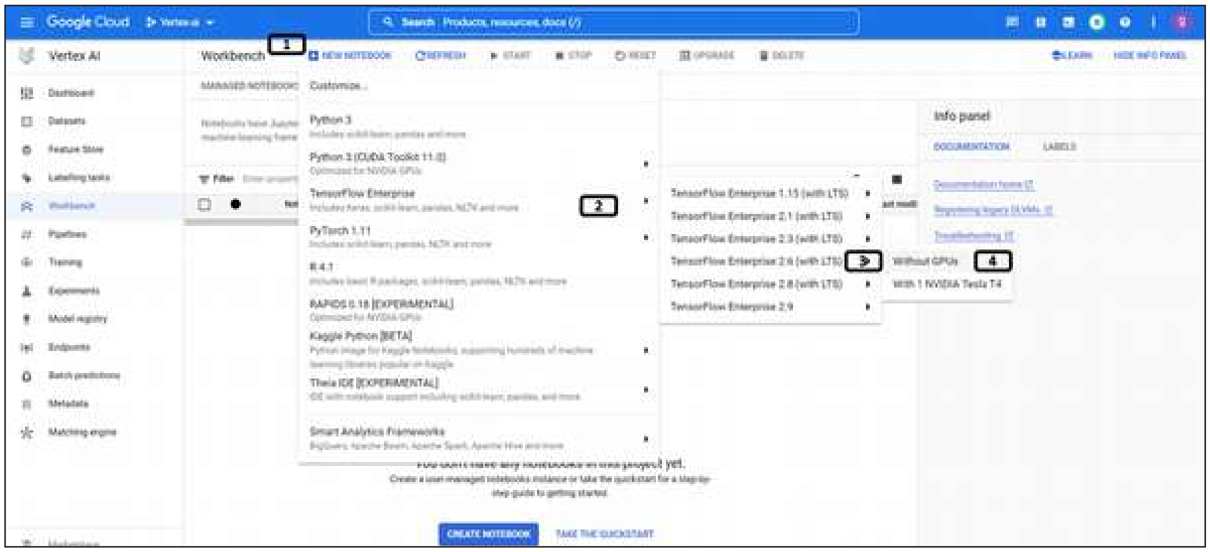
You don't have any notebooks in this project yet.

Figure 4.2: Landing page of workbench of vertex AI

1. Click **ENABLE NOTEBOOKS API** (notebook APIs need to be enabled once for every project. Notebook API can also be enabled from API and services of GCP.
2. Option to create **Managed Notebooks**.
3. Option to create **User-Managed Notebooks**, click on it (we will be using user-managed notebooks in majority of the exercises).
4. Option to create **New Instance**.

Step 3: New notebook selection

Users will be provided with options to create the instance as shown in *Figure 4.3*:


Google Cloud
Vertex AI
Workbench
1
NEW NOTEBOOK
REFRESH
START
STOP
RESET
UPGRADE
DELETE
LEARN
HIDE INFO PANEL
MANAGED NOTEBOOKS
Customize...
Python 3
Includes scikit-learn, pandas and more
Python 3 (CUDA Toolkit 11.0)
Optimized for NVIDIA GPUs
TensorFlow Enterprise
Includes Keras, scikit-learn, pandas, NLTK and more
2
PyTorch 1.11
Includes scikit-learn, pandas, NLTK and more
R 4.1
Includes basic R packages, scikit-learn, pandas, NLTK and more
RAPIDS 0.18 [EXPERIMENTAL]
Optimized for NVIDIA GPUs
Kaggle Python [BETA]
Python image for Kaggle Notebooks, supporting hundreds of machine learning libraries popular on Kaggle
Theia IDE [EXPERIMENTAL]
IDE with notebook support including scikit-learn, pandas, and more
Smart Analytics Frameworks
BigQuery, Apache Beam, Apache Spark, Apache Hive and more
TensorFlow Enterprise 1.15 (with LTS)
TensorFlow Enterprise 2.1 (with LTS)
TensorFlow Enterprise 2.3 (with LTS)
TensorFlow Enterprise 2.6 (with LTS)
3
TensorFlow Enterprise 2.8 (with LTS)
TensorFlow Enterprise 2.9
Without GPUs
4
With 1 NVIDIA Tesla T4
Dashboard
Datasets
Feature Store
Labeling tasks
Workbench
Pipelines
Training
Experiments
Model registry
Endpoints
Batch predictions
Metadata
Matching engine
Marketplace
Info panel
DOCUMENTATION
LABELS
Documentation home
Registering legacy DLVMs
Troubleshooting
CREATE NOTEBOOK
TAKE THE QUICKSTART

You don't have any notebooks in this project yet.

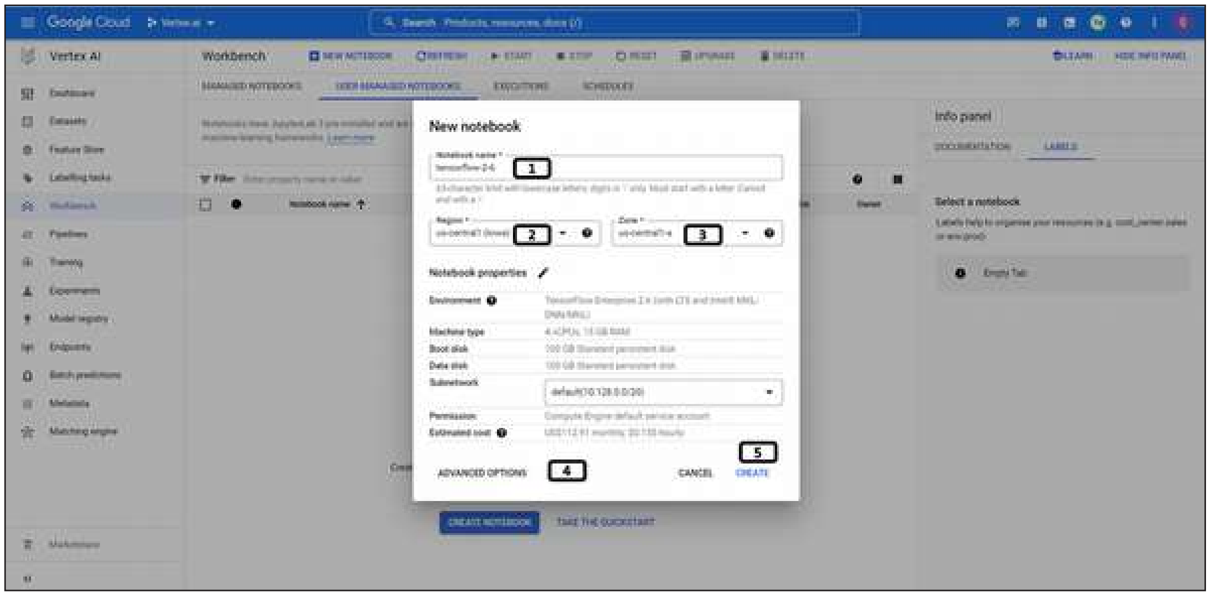
Create a user-managed notebooks instance or take the quickstart for a step-by-step guide to getting started.

Figure 4.3: New notebook selection (user managed)

1. Click **NEW NOTEBOOK**.
2. Select **Tensorflow Enterprise**.
3. Select **Tensorflow Enterprise 2.6** (we will be developing model using TensorFlow 2.6).
4. Select **Without GPUs**.

Step 4: TensorFlow notebook creation

After selecting TensorFlow New notebook popup will be shown as shown in *Figure 4.4*:



New notebook

Notebook name *: tensorflow-2-6

Region *: us-central1 (Iowa)

Zone *: us-central1-a

Notebook properties

Environment	TensorFlow Enterprise 2.6 (with LTS and Intel® MKL-DNN/MKL)
Machine type	4 vCPUs, 15 GB RAM
Boot disk	100 GB Standard persistent disk
Data disk	100 GB Standard persistent disk
Subnetwork	default(10.128.0.0/20)
Permission	Compute Engine default service account
Estimated cost	US$112.91 monthly, $0.155 hourly

ADVANCED OPTIONS CANCEL CREATE

Figure 4.4: Notebook creation

1. Provide **Notebook Name**.
2. Select **US-central** as **Region**.
3. Select **us-central1-a** as **Zone**.
4. **ADVANCED OPTIONS** provide users the choice to select the machine type, disk size, also the options on the networking (we can ignore for our case).
5. Click **CREATE**.

Step 5: Notebook creation complete

It will take a few minutes to complete the creation of notebooks and also the Jupyter lab instance. Once created notebook will be listed under the user-managed notebook section of workbench as shown in the *Figure 4.5*:

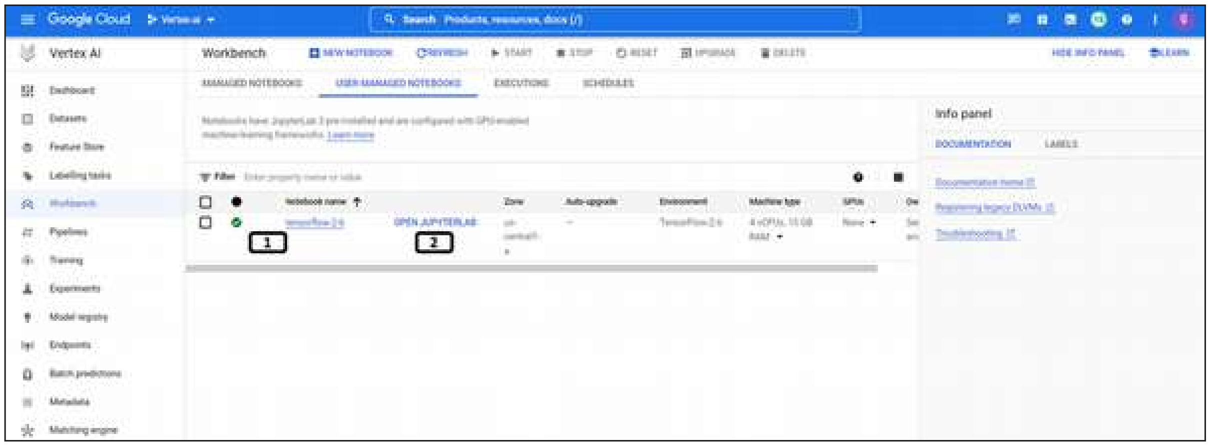


Figure 4.5: Notebook listed under workbench

1. Created notebook is listed.
2. Click **OPEN JUPYTERLAB**.

Step 6: Opening JupyterLab

After clicking on the JupyterLab, a session will be opened in a new tab as shown in *Figure 4.6*:

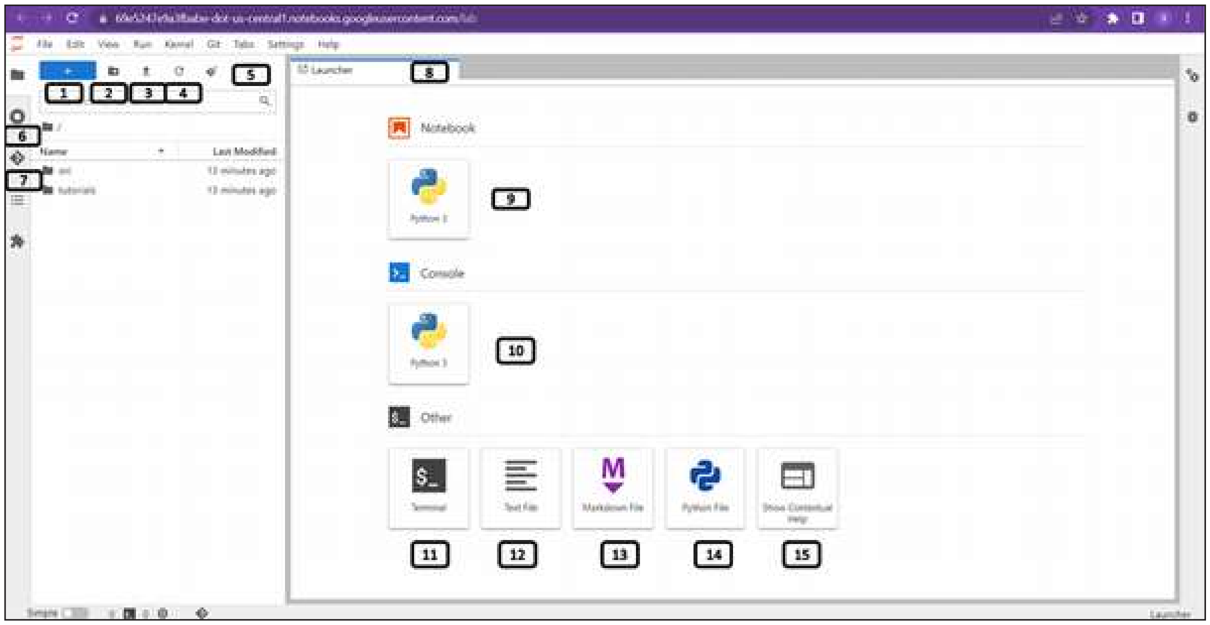


Figure 4.6: Landing page of JupyterLab

1. Button to open launcher (right side window in the figure).
2. Option to **Create folder**.
3. Option to **Upload files** from local system.
4. Option to **Refresh**.

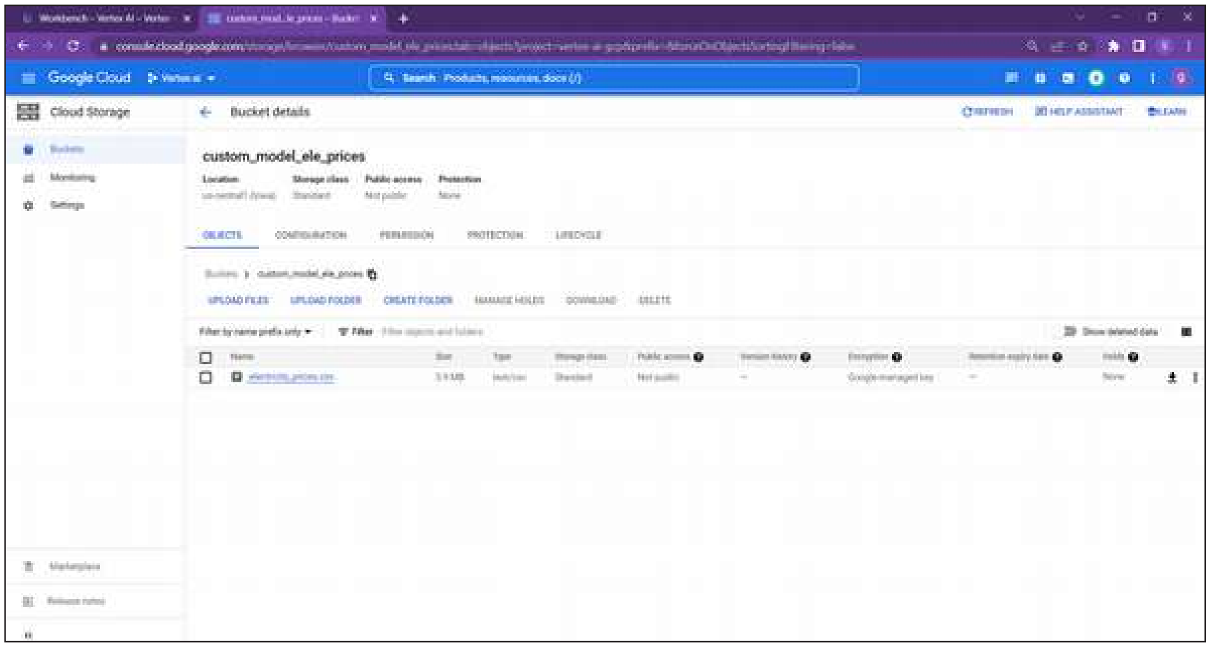
5. Option to **Clone git repository**.
6. Options to check **Kernels** and **Terminals**.
7. More options to connect to git repository.
8. Launcher using which new files can be created.
9. Option to create Python notebook.
10. Option to launch Python notebook console.
11. Option to launch **Terminal**.
12. Option to create **Text File**.
13. Option to create **Markdown File** (.md file).
14. Option to create **Python File** (.py file).
15. Options for **Show Contextual Help**.

Data for building custom model

For this exercise data is downloaded from Kaggle (link is provided below) and the dataset is listed under CC0:Public domain licenses. Data contains electricity prices for a data center and factors which may influence the price.

https://www.kaggle.com/datasets/salilchoubey/electrity-prices

`custom_model_ele_prices` bucket is created under `us-centra1` (single region) and .csv file is uploaded from to the bucket as shown in *Figure 4.7*:



Bucket details

custom_model_ele_prices

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

Name	Size	Type	Storage class	Public access	Version history	Encryption	Retention expiry date	Holds
electricity_prices.csv	5.1 MB	text/csv	Standard	Not public	–	Google-managed key	–	None

Figure 4.7*: CSV data uploaded to cloud storage*

Custom model building activity are done in multiple steps:

1. Create a Dockerfile.
2. Build the Python code for the ML model building.
3. Create a docker using Dockerfile and push the Python code to the docker.
4. Test the docker on the workbench environment.
5. Push the docker to the registry.
6. Submit the training job using created docker.

Introduction to Containers and Dockers

Containers virtualize the host OS (or kernel) and isolate an app's requirements from other containers. Before containers, having many apps on the same VM sometimes produced unusual behavior, therefore one application per VM was common. One app per VM addressed the conflicting dependency isolation issue, however it wasted resources (CPU and memory). A VM runs your program and a whole operating system, so your application has less resources.

Containers tackle this issue using a container engine and a container image, which is an application's dependencies. The container engine isolates applications by executing them in containers. This eliminates the need to run a distinct OS for each application, increasing resource usage and reducing expenses.

Docker, announced in 2013, revolutionized container technology by allowing users to quickly construct container images of programs. Docker is an open-source virtualization software to help developers. It is a PaaS software that isolates virtual environments to deploy, create, and test applications that are incompatible with the current OS.

Components of Docker:

- **Dockerfile**: Dockerfiles start each container. This text file includes instructions to construct a Docker image, including the OS, languages, environmental variables, file locations, network ports, and other components.
- **Dock image**: A Docker image is a portable, read-only executable file providing instructions for building a container and which software components it will execute.

- **Docker run**: Run in Docker runs a container. Multiple instances of the same image can run in a container.
- **DockerHub**: Docker Hub is a repository for container images. It's Docker's GitHub for containers.

Creation of Dockerfile

A Dockerfile is a text file that is read from top to bottom by Docker. It comprises of a series of instructions that instruct Docker how to build the Docker image. A Dockerfile is a recipe for creating Docker images and executing a separate build command that generates the Docker image from that recipe.

We need to enable the APIs of container registry and artifacts registry before we build the docker images. Follow below steps to enable them.

Step 1: APIs and services

All the APIs of the platform can be enabled or disabled through APIs and services. APIs and services can be opened navigation menu as shown in *Figure 4.8*:

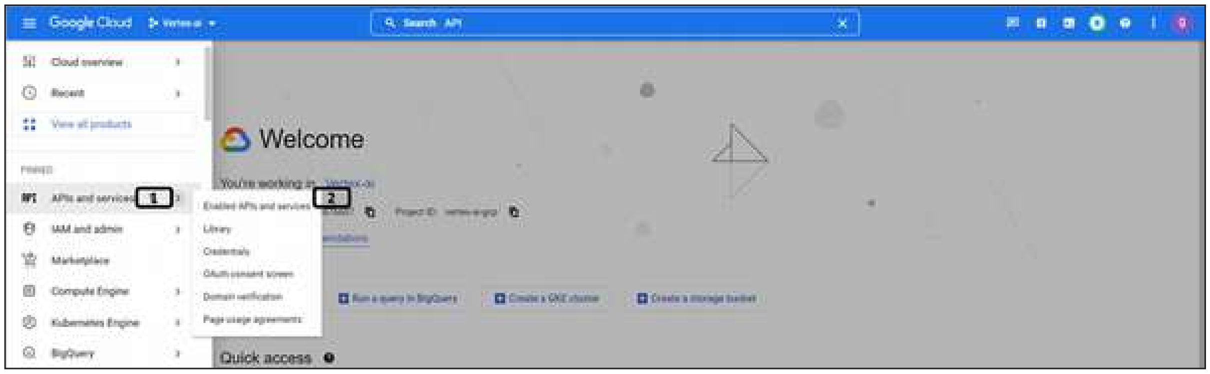


Figure 4.8: *Open APIs and services of platform*

1. Open the navigation menu and select **APIs and services**.
   a. Alternatively, users can search for APIs and services in the search box.
2. Click on **Enabled APIs and services**.

Step 2: Landing page of APIs and services

Landing page of APIs are services as shown in *Figure 4.9*:

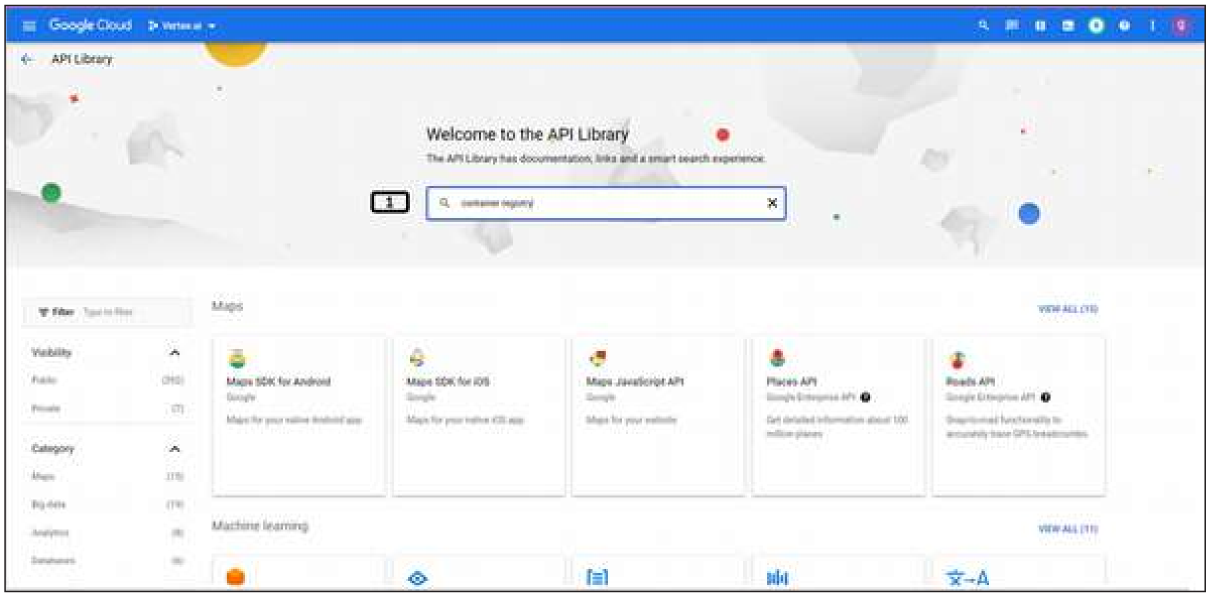
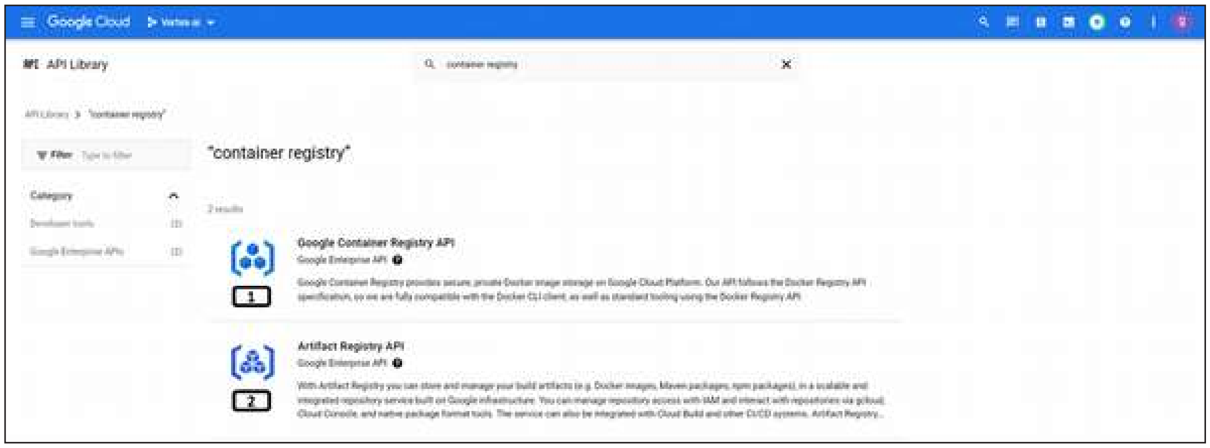


***Figure 4.9**: APIs and services of GCP*

1. Search for **`container registry`** in the search box.

Step 3: Container registry search results

Search results for container registry is as shown in *Figure 4.10*:



Google Container Registry API

Google Enterprise API

Google Container Registry provides secure, private Docker image storage on Google Cloud Platform. Our API follows the Docker Registry API specification, so we are fully compatible with the Docker CLI client, as well as standard tooling using the Docker Registry API.

Artifact Registry API

Google Enterprise API

With Artifact Registry you can store and manage your build artifacts (e.g. Docker images, Maven packages, npm packages), in a scalable and integrated repository service built on Google infrastructure. You can manage repository access with IAM and interact with repositories via gcloud, Cloud Console, and native package format tools. The service can also be integrated with Cloud Build and other CI/CD systems. Artifact Registry...

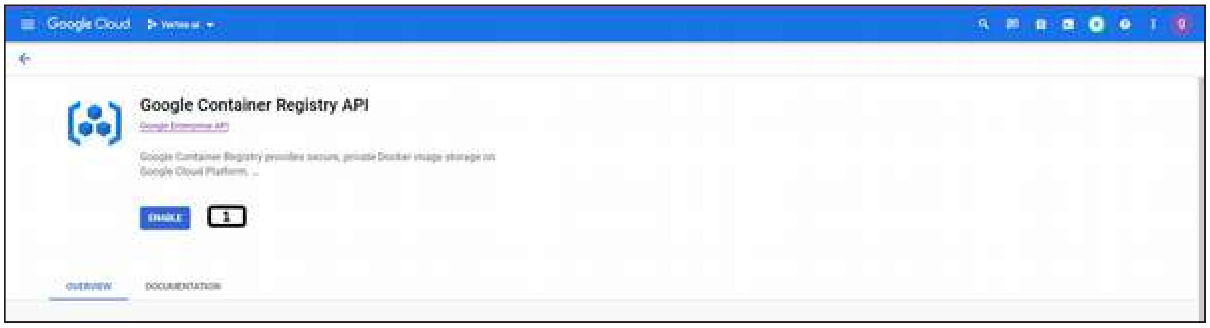
***Figure 4.10**: Search results of APIs*

1. Right click to enable the **Google Container Registry API**.
2. Right click on enable **Artifact Registry API**.

Step 4: Container registry API enablement

Container Registry is a centralized location where teams can maintain Docker images, perform vulnerability analysis, and implement fine-grained access control.

Follow the step in *Figure 4.11* to enable **Container registry API**:



Google Container Registry provides secure, private Docker image storage on Google Cloud Platform. ...

Figure 4.11: *Container registry API*

1. Click **ENABLE**.

Step 5: Artifacts registry API enablement

Artifact Registry supports container images and non-container artifacts. Artifact Registry enhances and expands Container Registry's features, such as customer-managed encryption keys, VPC-SC support, Pub/Sub notifications, delivering important security, scalability, and control advancements, and more.

Follow the action shown in *Figure 4.12* to enable container registry API:

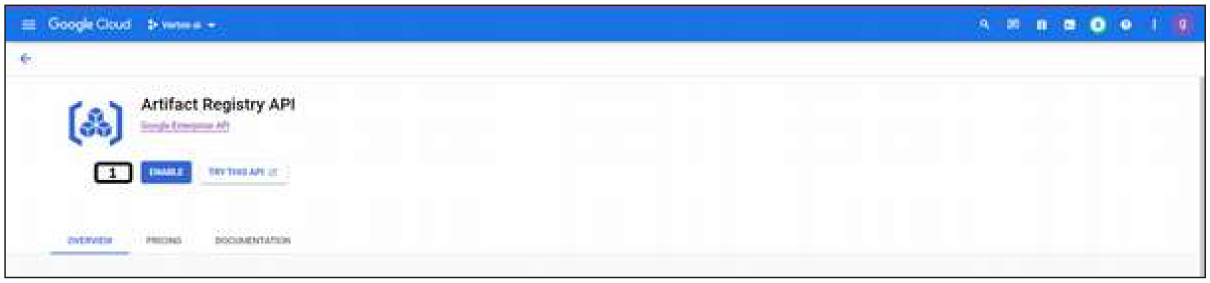


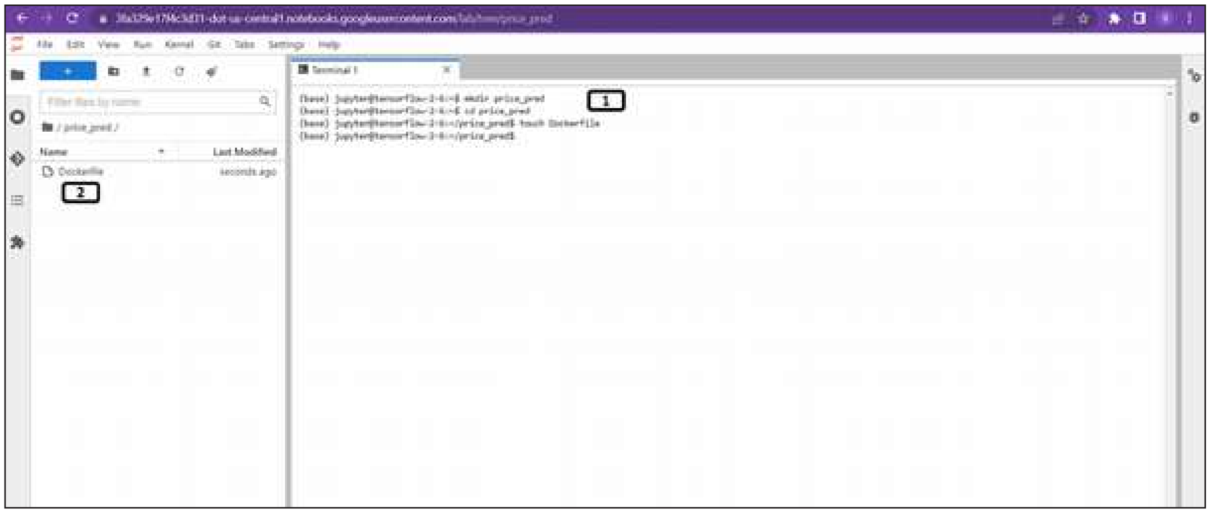
Figure 4.12: *Artifacts registry API*

1. **ENABLE** the **Artifact registry API**.

After enabling both the APIs, let us navigate back to the workbench and launch terminal. Refer to the following figure (double click on Terminal).

Step 6: Directory creation

Launch **terminal** (refer *Figure 4.6*, point no. 11) and follow the steps described in *Figure 4.13* to create directory in the workbench:



```
(base) jupyter@tensorflow-2-6:~$ mkdir price_pred
(base) jupyter@tensorflow-2-6:~$ cd price_pred
(base) jupyter@tensorflow-2-6:~/price_pred$ touch Dockerfile
(base) jupyter@tensorflow-2-6:~/price_pred$
```

Figure 4.13: *Directory creation in the terminal*

1. Type the following commands one after the other in the Terminal window.

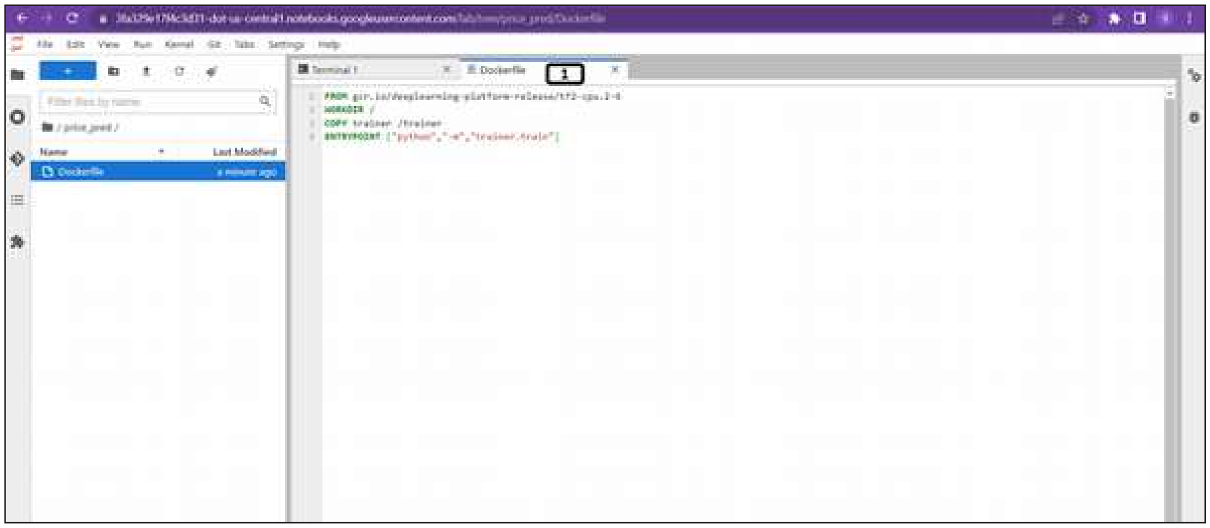
   We create new directory call **`price_pred`**, and create Dockerfile inside the **`price_pred`** directory:

   ```
   mkdir price_pred
   cd price_pred
   touch Dockerfile
   ```

2. **Dockerfile** will be created inside the **price_pred** folder (file will have no contents, double click on the **file** to open it).

Step 7: Edit Dockerfile

Enter the commands in the Dockerfile as shown in *Figure 4.14*:



```
FROM gcr.io/deeplearning-platform-release/tf2-cpu.2-6
WORKDIR /
COPY trainer /trainer
ENTRYPOINT ["python","-m","trainer.train"]
```

Figure 4.14: *Editing Dockerfile*

1. Type the following commands:

```
FROM gcr.io/deeplearning-platform-release/tf2-cpu.2-6
WORKDIR /
COPY trainer /trainer
ENTRYPOINT [“python”,”-m”,”trainer.train”]
```

Explanation for the commands is provided below:

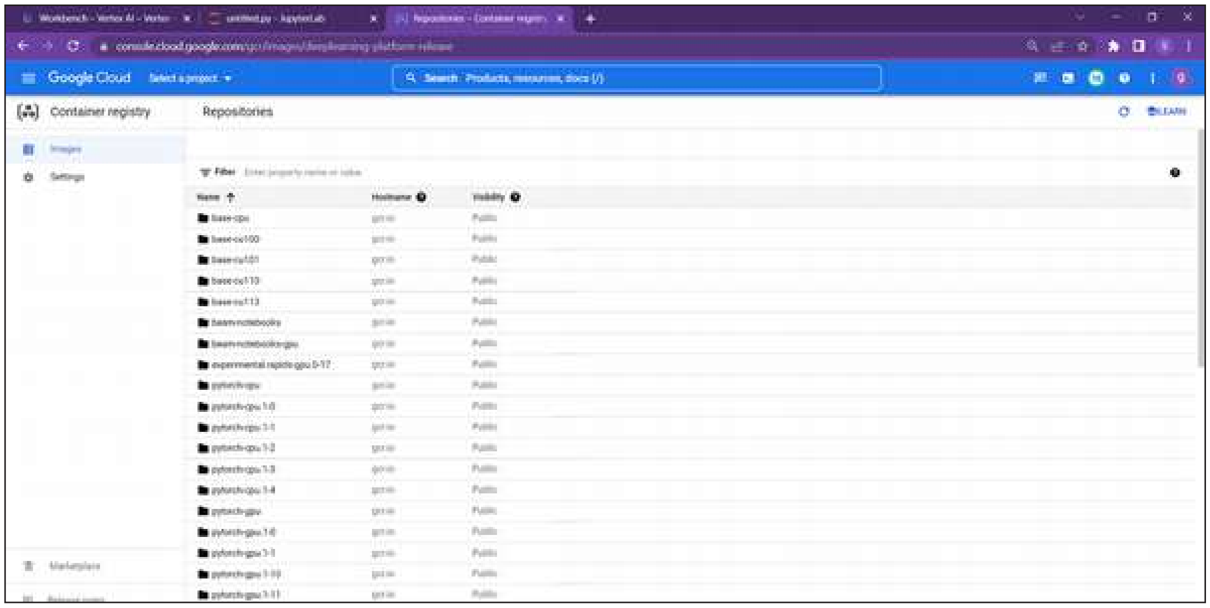
- Docker image will be built using the container images that is available in the **gcr.io**. (refer figure for more details).
- A working directory is set.
- It will copy the trainer folder to the working directory (**trainer** folder is yet to be created in the JupyterLab, trainer folder contains **train.py** file for model training)
- **ENTRYPOINT** point towards executables that execute when the container starts.

Save the Dockerfile (*Ctrl* + *S*) and close the file.

Container registry of GCP provides the repository of deep learning containers. Deep learning containers are Docker containers providing data science frameworks, libraries, and tools. These performance-optimized containers enable users to experiment and deploy processes rapidly. They are hosted at different data centers.

- **gcr.io** hosts images in U.S. data centers, however the location may vary.
- **us.gcr.io** hosts container images in EU region.
- **eu.gcr.io** hosts container images in US region.
- **asia.gcr.io** hosts container images in Asia region.

Figure 4.15 shows the repository for the **gcr.io**:



Name	Hostname	Visibility
base-cpu	gcr.io	Public
base-cu100	gcr.io	Public
base-cu101	gcr.io	Public
base-cu110	gcr.io	Public
base-cu113	gcr.io	Public
beam-notebooks	gcr.io	Public
beam-notebooks-gpu	gcr.io	Public
experimental.rapids-gpu.0-17	gcr.io	Public
pytorch-cpu	gcr.io	Public
pytorch-cpu.1-0	gcr.io	Public
pytorch-cpu.1-1	gcr.io	Public
pytorch-cpu.1-2	gcr.io	Public
pytorch-cpu.1-3	gcr.io	Public
pytorch-cpu.1-4	gcr.io	Public
pytorch-gpu	gcr.io	Public
pytorch-gpu.1-0	gcr.io	Public
pytorch-gpu.1-1	gcr.io	Public
pytorch-gpu.1-10	gcr.io	Public
pytorch-gpu.1-11	gcr.io	Public

Figure 4.15: Container registry for gcr.io

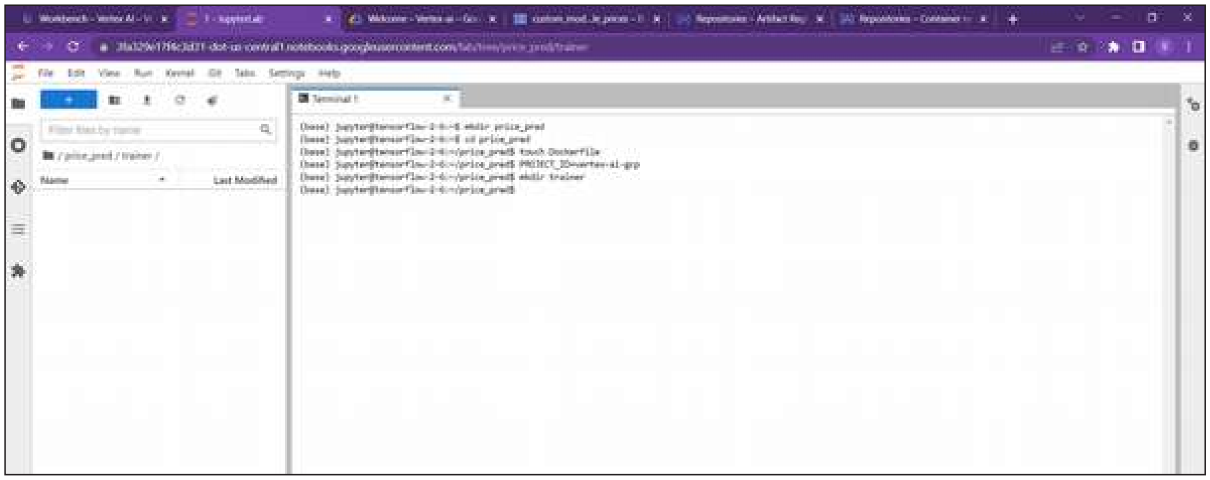
Model building

Once the Dockerfile is created, we need to work on the Python code for the model building. The task is to build a regression model to predict the electricity prices. Python code will be copied into the container and submitted for training job.

Let us go back to the terminal of the workbench and follow these steps to create the python code for model building:

Step 1: Create trainer folder

Type the commands in the terminal as shown in *Figure 4.16:*



```
(base) jupyter@tensorflow-2-6:~$ mkdir price_pred
(base) jupyter@tensorflow-2-6:~$ cd price_pred
(base) jupyter@tensorflow-2-6:~/price_pred$ touch Dockerfile
(base) jupyter@tensorflow-2-6:~/price_pred$ PROJECT_ID=vertex-ai-gcp
(base) jupyter@tensorflow-2-6:~/price_pred$ mkdir trainer
(base) jupyter@tensorflow-2-6:~/price_pred$
```

Figure 4.16: Trainer folder creation in terminal

```
PROJECT_ID=vertex-ai-gcp

mkdir trainer
```

In the first command, a variable called **PROJECT_ID** is set to the project ID (use project ID not the project name), we will use this command while creating a docker image. In the second line of command, we will create a new folder called **trainer**. Open the **trainer** folder.

Step 2: Create Python file

Follow the steps mentioned in *Figure 4.17* to create Python file:

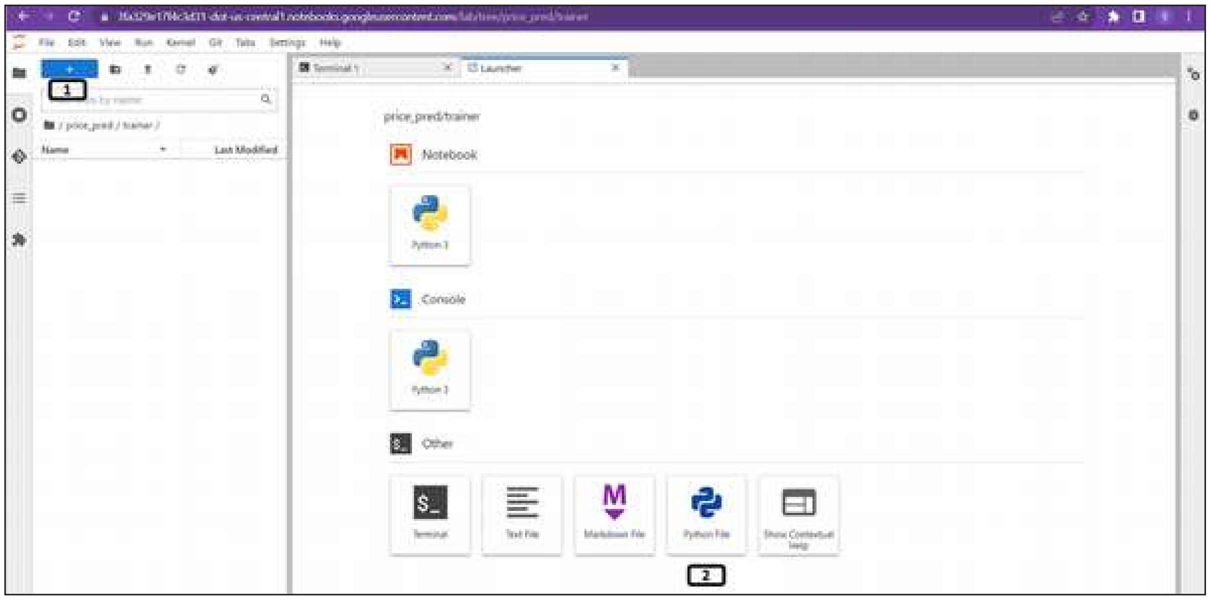


***Figure 4.17**: Python file creation on workbench*

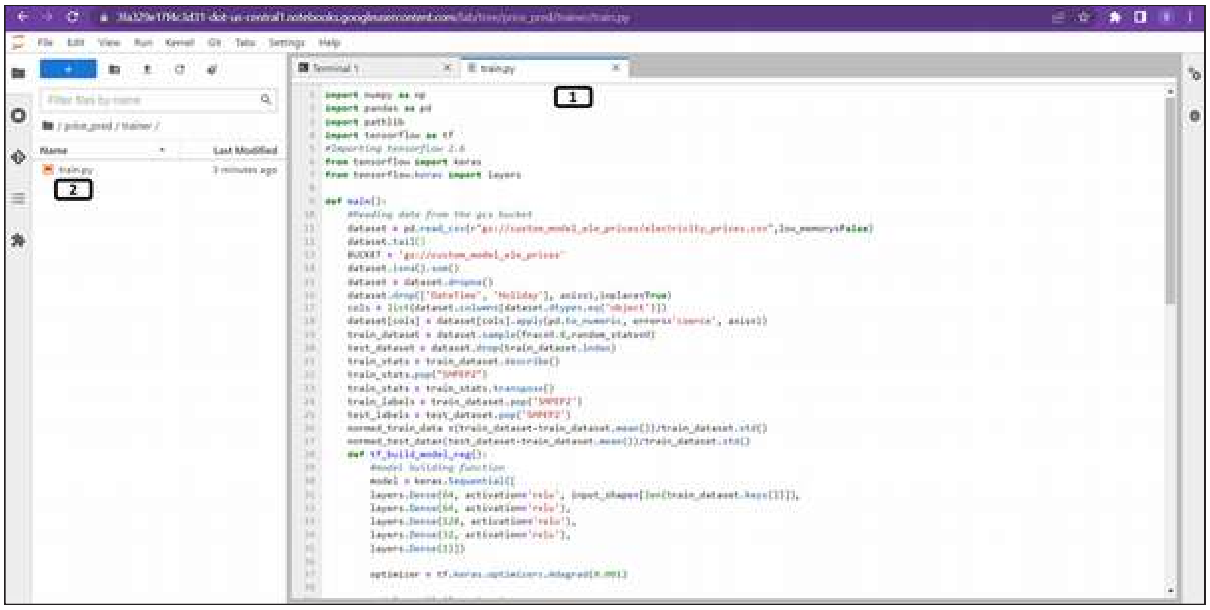
1. Click **New launcher**.
2. Double click on the **Python File**.

Step 3: Python code

Python file will be created inside the trainer folder, write the below given Python code in the py file as shown in the *Figure 4.18:*

```
import numpy as np

import pandas as pd

import pathlib

import tensorflow as tf

#Importing tensorflow 2.6
```

```
from tensorflow import keras

from tensorflow.keras import layers


def main():

    #Reading data from the gcs bucket

    dataset=pd.read_csv(r”gs://custom_model_ele_prices/electricity_
    prices.csv”, low_memory=False)

    dataset.tail()

    BUCKET = ‘gs://custom_model_ele_prices’

    dataset.isna().sum()

    dataset = dataset.dropna()

    dataset.drop([‘DateTime’, ‘Holiday’], axis=1,inplace=True)

    cols = list(dataset.columns[dataset.dtypes.eq(‘object’)])

    dataset[cols] = dataset[cols].apply(pd.to_numeric, errors=’coerce’,
    axis=1)

    train_dataset = dataset.sample(frac=0.8,random_state=0)

    test_dataset = dataset.drop(train_dataset.index)

    train_stats = train_dataset.describe()

    train_stats.pop(“SMPEP2”)

    train_stats = train_stats.transpose()

    train_labels = train_dataset.pop(‘SMPEP2’)

    test_labels = test_dataset.pop(‘SMPEP2’)

    normed_train_data =(train_dataset-train_dataset.mean())/train_
    dataset.std()

    normed_test_data=(test_dataset-train_dataset.mean())/train_dataset.
    std()
    def tf_build_model_reg():

        #model building function

        model = keras.Sequential([

        layers.Dense(64, activation='relu', input_shape=[len(train_
        dataset.keys())]),

        layers.Dense(64, activation='relu'),

        layers.Dense(128, activation='relu'),

        layers.Dense(32, activation='relu'),

        layers.Dense(1)])

        optimizer = tf.keras.optimizers.Adagrad(0.001)

        model.compile(loss='mse',optimizer=optimizer, metrics=['mae',
        'mse'])

        return model

    model = tf_build_model_reg()

    epochs = 10


    early_stop = keras.callbacks.EarlyStopping(monitor='val_loss',
    patience=5)


    early_history = model.fit(normed_train_data, train_
    labels,epochs=epochs, validation_split = 0.2,callbacks=[early_stop])

    model.save(BUCKET + '/model')


if __name__=="__main__":

    main()
```



```
import numpy as np
import pandas as pd
import pathlib
import tensorflow as tf
#Importing tensorflow 2.6
from tensorflow import keras
from tensorflow.keras import layers

def main():
    #Reading data from the gcs bucket
    dataset = pd.read_csv(r"gs://custom_model_ele_prices/electricity_prices.csv",low_memory=False)
    dataset.tail()
    BUCKET = 'gs://custom_model_ele_prices'
    dataset.isna().sum()
    dataset = dataset.dropna()
    dataset.drop(['DateTime', 'Holiday'], axis=1,inplace=True)
    cols = list(dataset.columns[dataset.dtypes.eq('object')])
    dataset[cols] = dataset[cols].apply(pd.to_numeric, errors='coerce', axis=1)
    train_dataset = dataset.sample(frac=0.8,random_state=0)
    test_dataset = dataset.drop(train_dataset.index)
    train_stats = train_dataset.describe()
    train_stats.pop("SMPEP2")
    train_stats = train_stats.transpose()
    train_labels = train_dataset.pop('SMPEP2')
    test_labels = test_dataset.pop('SMPEP2')
    normed_train_data =(train_dataset-train_dataset.mean())/train_dataset.std()
    normed_test_data=(test_dataset-train_dataset.mean())/train_dataset.std()
    def tf_build_model_reg():
        #model building function
        model = keras.Sequential([
        layers.Dense(64, activation='relu', input_shape=[len(train_dataset.keys())]),
        layers.Dense(64, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(1)])

        optimizer = tf.keras.optimizers.Adagrad(0.001)
```

***Figure 4.18**: Python code for model training*

1. Type the Python code.
2. Right click and rename the Python file name to **train.py**.

Save the Python file (*Ctrl* + *S*) and close the file.

Python code consists of code to import required packages. Main function reads the .csv file from the cloud storage bucket apply few data transformations, split the data for train and test. **`tf_build_model_reg`** function contains the skeleton for the TensorFlow model (model will be built using TensorFlow 2.6). At the last, trained model will be pushed into the cloud storage bucket.

NOTE: We have used Epochs to be 10. Increase it as per your need.

The code used for this exercise is not very detailed since it is only for illustrative purposes. Users can use more complex code for the model training.

Image creation

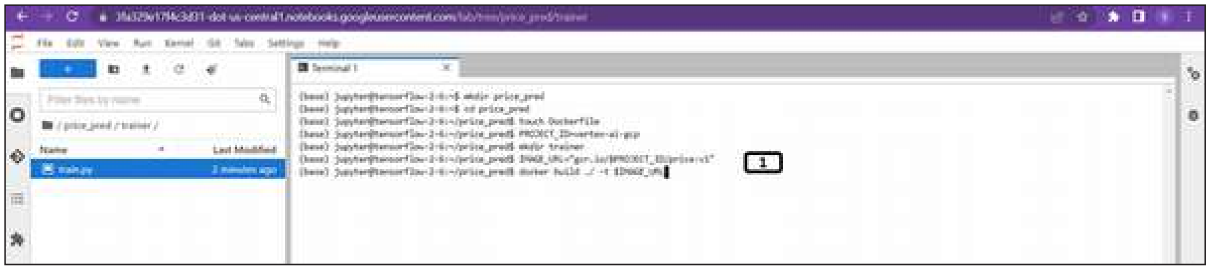
Now that we have completed the creation of Dockerfile and Python code for the model building. We can proceed with image creation.

Step 1: Image creation using docker build

Type the following commands as shown in *Figure 4.19* for image creation:

```
IMAGE_URL=”gcr.io/$PROJECT_ID/price:v1”
```

```
docker build ./ -t $IMAGE_URL
```

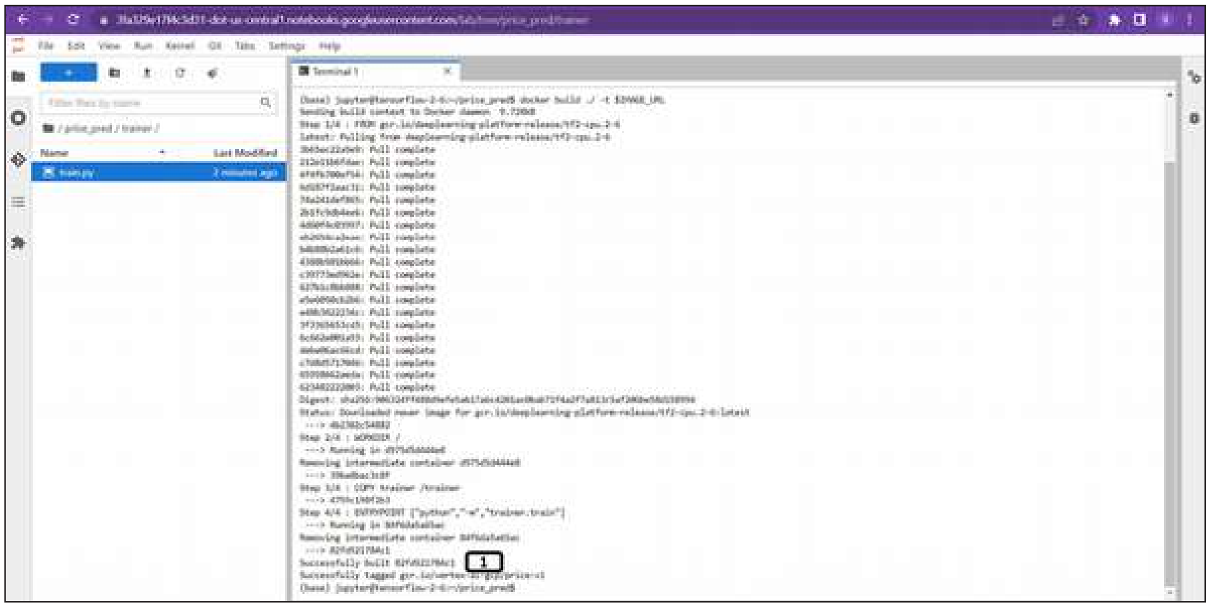


```
(base) jupyter@tensorflow-2-6:~$ mkdir price_pred
(base) jupyter@tensorflow-2-6:~$ cd price_pred
(base) jupyter@tensorflow-2-6:~/price_pred$ touch Dockerfile
(base) jupyter@tensorflow-2-6:~/price_pred$ PROJECT_ID=vertex-ai-gcp
(base) jupyter@tensorflow-2-6:~/price_pred$ mkdir trainer
(base) jupyter@tensorflow-2-6:~/price_pred$ IMAGE_URL="gcr.io/$PROJECT_ID/price:v1"
(base) jupyter@tensorflow-2-6:~/price_pred$ docker build ./ -t $IMAGE_URL
```

Figure 4.19: *Image creation process*

1. Type the commands in the Terminal.

It will take a few minutes to build the image from the Dockerfile. Once the build is complete, success message will be displayed in the terminal as shown in *Figure 4.20*:



```
(base) jupyter@tensorflow-2-6:~/price_pred$ docker build ./ -t $IMAGE_URL
Sending build context to Docker daemon  9.728kB
Step 1/4 : FROM gcr.io/deeplearning-platform-release/tf2-cpu.2-6
latest: Pulling from deeplearning-platform-release/tf2-cpu.2-6
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
Digest: sha256:[illegible]
Status: Downloaded newer image for gcr.io/deeplearning-platform-release/tf2-cpu.2-6:latest
 ---> [illegible]
Step 2/4 : WORKDIR /
 ---> Running in [illegible]
Removing intermediate container [illegible]
 ---> [illegible]
Step 3/4 : COPY trainer /trainer
 ---> [illegible]
Step 4/4 : ENTRYPOINT ["python","-m","trainer.train"]
 ---> Running in [illegible]
Removing intermediate container [illegible]
 ---> [illegible]
Successfully built [illegible]
Successfully tagged gcr.io/vertex-ai-gcp/price:v1
(base) jupyter@tensorflow-2-6:~/price_pred$
```

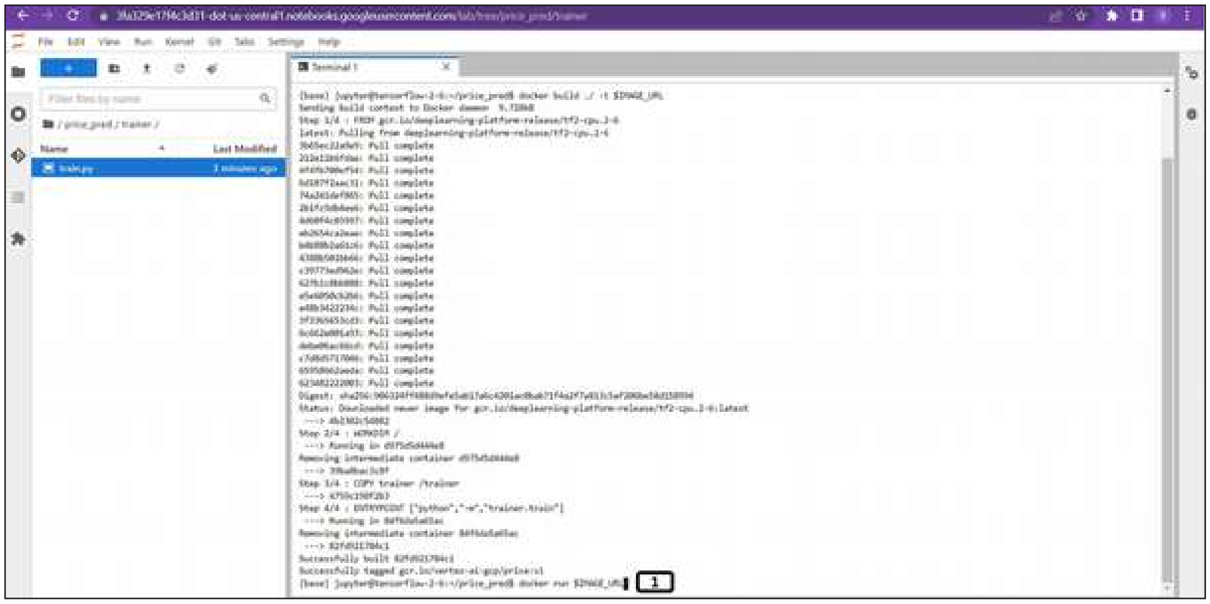
Figure 4.20: *Image creation complete*

1. Success message after completion of image creation.

Step 2: Checking the docker image

After creation of the image, let us check if we can run the python code from the image created. Type the command given below as shown in *Figure 4.21:*

```
docker run $IMAGE_URL
```

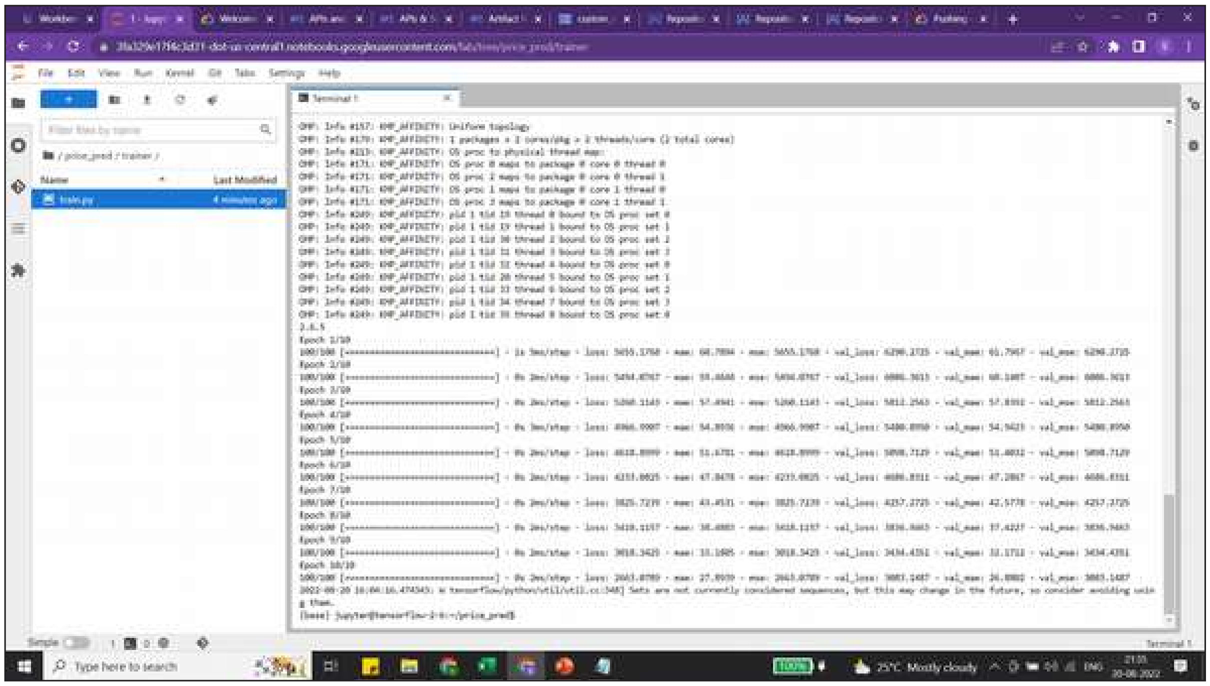


```
(base) jupyter@tensorflow-2-6:~/price_pred$ docker build ./ -t $IMAGE_URI
Sending build context to Docker daemon  9.728kB
Step 1/4 : FROM gcr.io/deeplearning-platform-release/tf2-cpu.2-6
latest: Pulling from deeplearning-platform-release/tf2-cpu.2-6
3b65ec22a6e9: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
Digest: sha256:[illegible]
Status: Downloaded newer image for gcr.io/deeplearning-platform-release/tf2-cpu.2-6:latest
 ---> [illegible]
Step 2/4 : WORKDIR /
 ---> Running in [illegible]
Removing intermediate container [illegible]
 ---> [illegible]
Step 3/4 : COPY trainer /trainer
 ---> [illegible]
Step 4/4 : ENTRYPOINT ["python","-m","trainer.train"]
 ---> Running in [illegible]
Removing intermediate container [illegible]
 ---> [illegible]
Successfully built [illegible]
Successfully tagged gcr.io/vertex-ai-gcp/price:v1
(base) jupyter@tensorflow-2-6:~/price_pred$ docker run $IMAGE_URI 1
```

Figure 4.21: Test run of python code on container

1. Type the command.

Python code has run successfully as shown in the following figure:



```
OMP: Info #157: KMP_AFFINITY: Uniform topology
OMP: Info #179: KMP_AFFINITY: 1 packages x 2 cores/pkg x 2 threads/core (2 total cores)
OMP: Info #213: KMP_AFFINITY: OS proc to physical thread map:
OMP: Info #171: KMP_AFFINITY: OS proc 0 maps to package 0 core 0 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 2 maps to package 0 core 0 thread 1
OMP: Info #171: KMP_AFFINITY: OS proc 1 maps to package 0 core 1 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 3 maps to package 0 core 1 thread 1
OMP: Info #249: KMP_AFFINITY: pid 1 tid 19 thread 0 bound to OS proc set 0
OMP: Info #249: KMP_AFFINITY: pid 1 tid 19 thread 1 bound to OS proc set 1
OMP: Info #249: KMP_AFFINITY: pid 1 tid 30 thread 2 bound to OS proc set 2
OMP: Info #249: KMP_AFFINITY: pid 1 tid 31 thread 3 bound to OS proc set 3
OMP: Info #249: KMP_AFFINITY: pid 1 tid 32 thread 4 bound to OS proc set 0
OMP: Info #249: KMP_AFFINITY: pid 1 tid 20 thread 5 bound to OS proc set 1
OMP: Info #249: KMP_AFFINITY: pid 1 tid 33 thread 6 bound to OS proc set 2
OMP: Info #249: KMP_AFFINITY: pid 1 tid 34 thread 7 bound to OS proc set 3
OMP: Info #249: KMP_AFFINITY: pid 1 tid 35 thread 8 bound to OS proc set 0
2.6.5
Epoch 1/10
100/100 [==============================] - 1s 5ms/step - loss: 5655.1768 - mae: 60.7994 - mse: 5655.1768 - val_loss: 6290.2725 - val_mae: 61.7967 - val_mse: 6290.2725
Epoch 2/10
100/100 [==============================] - 0s 2ms/step - loss: 5494.0767 - mae: 59.4668 - mse: 5494.0767 - val_loss: 6006.5613 - val_mae: 60.1407 - val_mse: 6006.5613
Epoch 3/10
100/100 [==============================] - 0s 2ms/step - loss: 5268.1143 - mae: 57.4941 - mse: 5268.1143 - val_loss: 5812.2563 - val_mae: 57.8392 - val_mse: 5812.2563
Epoch 4/10
100/100 [==============================] - 0s 3ms/step - loss: 4966.9987 - mae: 54.8936 - mse: 4966.9987 - val_loss: 5400.8950 - val_mae: 54.5623 - val_mse: 5400.8950
Epoch 5/10
100/100 [==============================] - 0s 2ms/step - loss: 4618.8999 - mae: 51.6781 - mse: 4618.8999 - val_loss: 5098.7129 - val_mae: 51.4032 - val_mse: 5098.7129
Epoch 6/10
100/100 [==============================] - 0s 2ms/step - loss: 4233.0025 - mae: 47.8478 - mse: 4233.0025 - val_loss: 4686.8311 - val_mae: 47.2867 - val_mse: 4686.8311
Epoch 7/10
100/100 [==============================] - 0s 2ms/step - loss: 3825.7239 - mae: 43.4531 - mse: 3825.7239 - val_loss: 4257.2725 - val_mae: 42.5778 - val_mse: 4257.2725
Epoch 8/10
100/100 [==============================] - 0s 2ms/step - loss: 3418.1157 - mae: 38.4883 - mse: 3418.1157 - val_loss: 3836.5663 - val_mae: 37.4227 - val_mse: 3836.5663
Epoch 9/10
100/100 [==============================] - 0s 2ms/step - loss: 3018.3425 - mae: 33.1805 - mse: 3018.3425 - val_loss: 3434.4351 - val_mae: 32.1712 - val_mse: 3434.4351
Epoch 10/10
100/100 [==============================] - 0s 2ms/step - loss: 2663.0789 - mae: 27.8939 - mse: 2663.0789 - val_loss: 3083.1487 - val_mae: 26.8802 - val_mse: 3083.1487
2022-08-20 16:04:16.474343: W tensorflow/python/util/util.cc:348] Sets are not currently considered sequences, but this may change in the future, so consider avoiding usin
g them.
(base) jupyter@tensorflow-2-6:~/price_pred$
```

Figure 4.22: Successful run of code

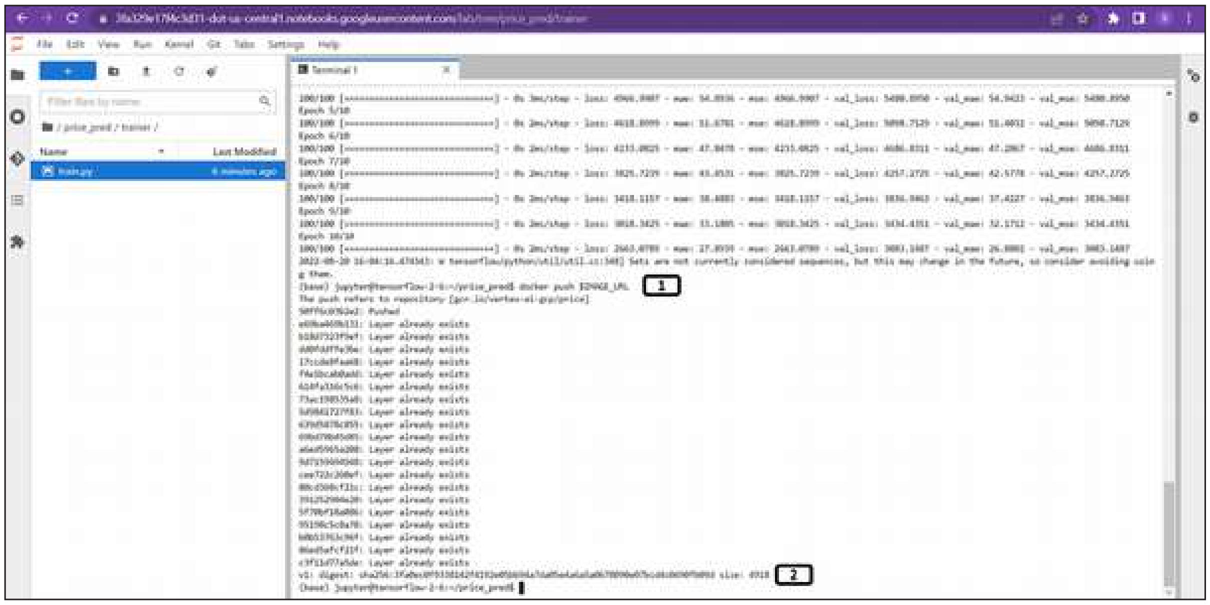
Pushing image to container registry

Now that we have created the image and tested if the code runs with proper output, we can now push the image to the container registry.

Step 1: Image push command

Type the following command in the Terminal as shown in *Figure 4.23*:

```
docker push $IMAGE_URL
```



```
Epoch 10/10
100/100 [==============================] - [illegible]
(base) jupyter@tensorflow-2-6:~/price_pred$ docker push $IMAGE_URL
The push refers to repository [gcr.io/vertex-ai-gcp/price]
[illegible]: Pushed
[illegible]: Layer already exists
[illegible]: Layer already exists
[illegible]: Layer already exists
v1: digest: sha256:[illegible] size: 4918
(base) jupyter@tensorflow-2-6:~/price_pred$
```

Figure 4.23*: Image to project container registry*

1. Type the command.
2. Image successfully pushed to the container registry.

Step 2: Check container registry

Open the **container registry** from the navigation menu of the platform (it will be listed under CI/CD). Alternatively, type container registry in the search box of the platform to open it. Landing page of container registry is as shown in *Figure 4.24:*

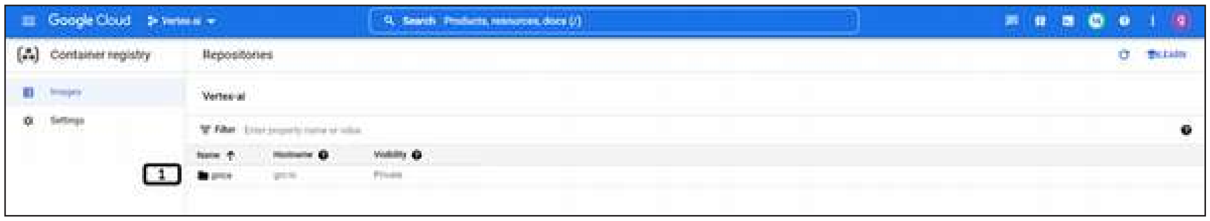


Figure 4.24: *Created price folder in container registry*

1. A folder called **price** will be created in the **container registry** (Refer to the image creation where we have specified price to be the name and v1 to be the tag). Click on **price**.

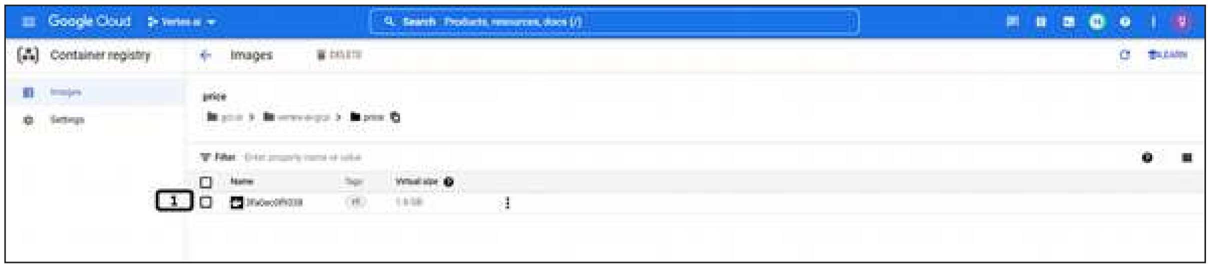


Figure 4.25: *Created image pushed to container registry*

1. Image will be available inside the folder and under **tag v1**.

Submitting the custom model training job

For the model to get trained, it picks the data from the .csv which is uploaded on the cloud storage bucket. By default, workbench resources will have a read and write access to the cloud storage. However, when we submit the job for training, training code will execute on one or more Google Cloud-managed VM instances. These VM will not have access to the cloud storage buckets and hence we will use the service account. Follow these steps to submit the training job:

Step 1: Training module of vertex AI

Open the training module of the vertex AI as shown in *Figure 4.26*:

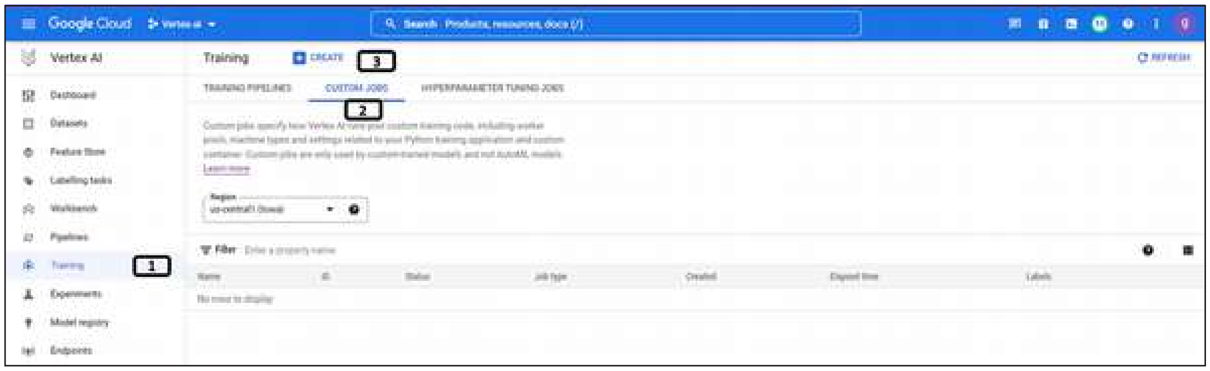


Figure 4.26: *Training module*

1. Select training module.
2. Select custom jobs.
3. Click **CREATE**.

Step 2: Selecting training method

Train new model page will appear as shown in *Figure 4.27*. Follow the steps mentioned in the figure to configure training method:

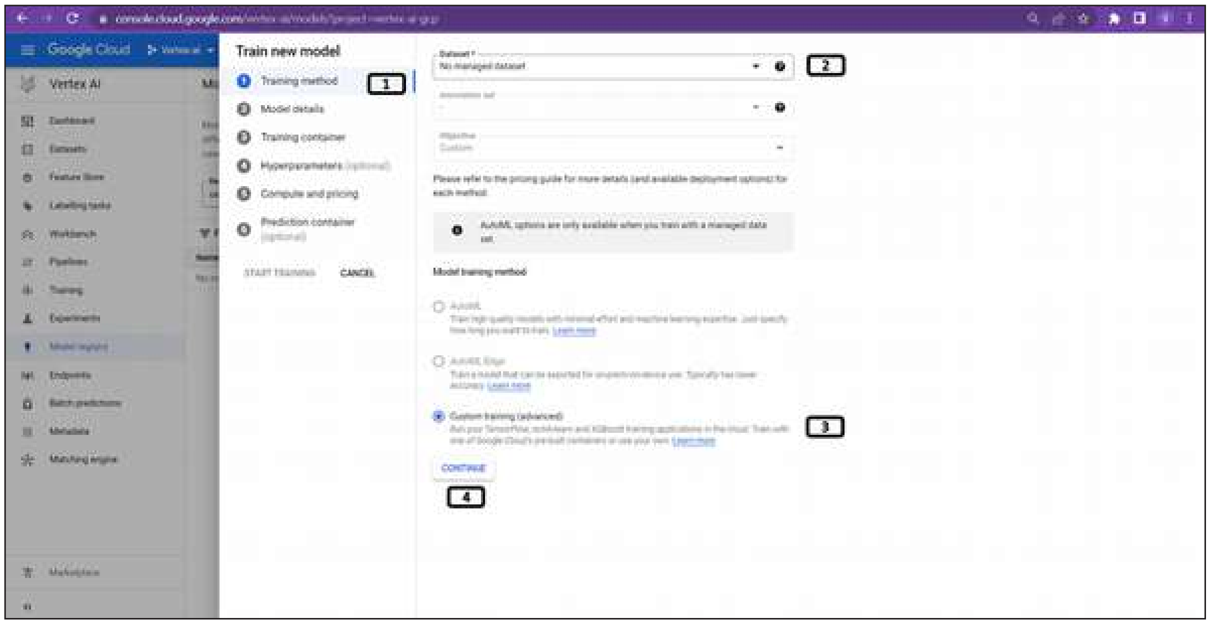


***Figure 4.27**: Training method selection for custom model*

1. Select the **Training method**.
2. Select **No managed dataset** under the **Dataset** selection.
3. Select custom training.
4. Click **CONTINUE**.

Step 3: Model details

Select the model details to train the model as shown in *Figure 4.28:*

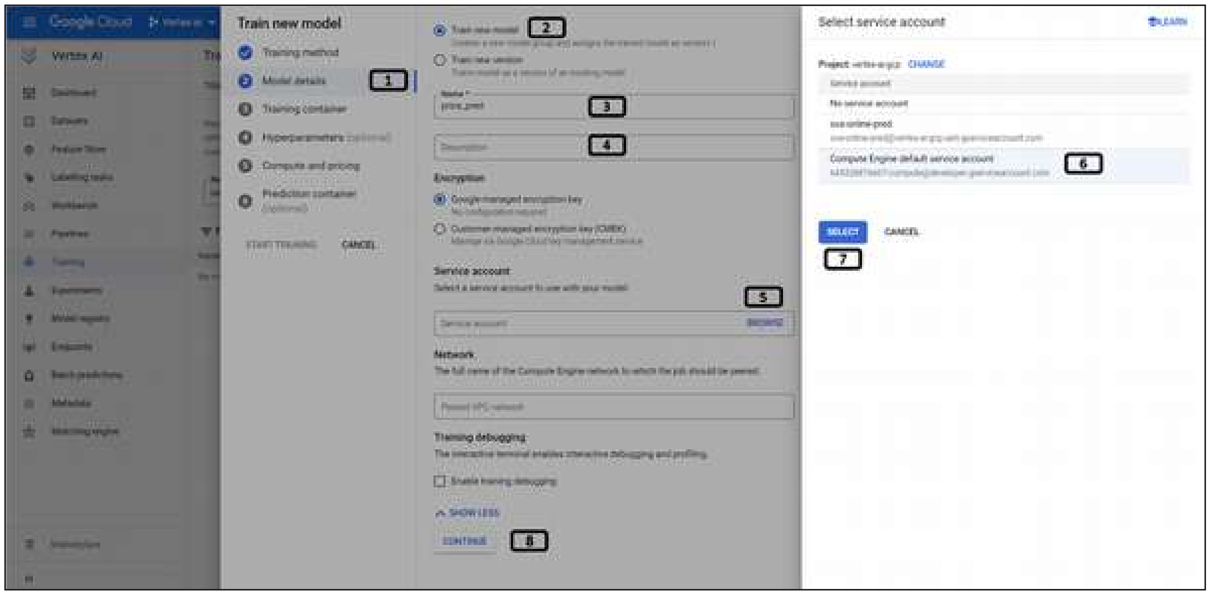


Figure 4.28: Model details for custom model

1. After training method selection, we have to enter the **Model details**.
2. Select **Train new model**.
3. Provide **Name** for the model.
4. Provide the **Model description**.
5. **Service account** must be chosen for the model training. Click **BROWSE**.
5. Since code will be pushed the Google cloud VM instance service account is needed for accessing data from cloud storage and to store back the model to the cloud storage.
6. Select service account associated with the compute (users can also choose service account which we had created in Chapter 2 but roles to access cloud storage needs to be added).
7. Click **SELECT**.
8. Click **CONTINUE**.

Step 4: Training container details

Follow the steps mentioned in *Figure 4.29* to configure training container:

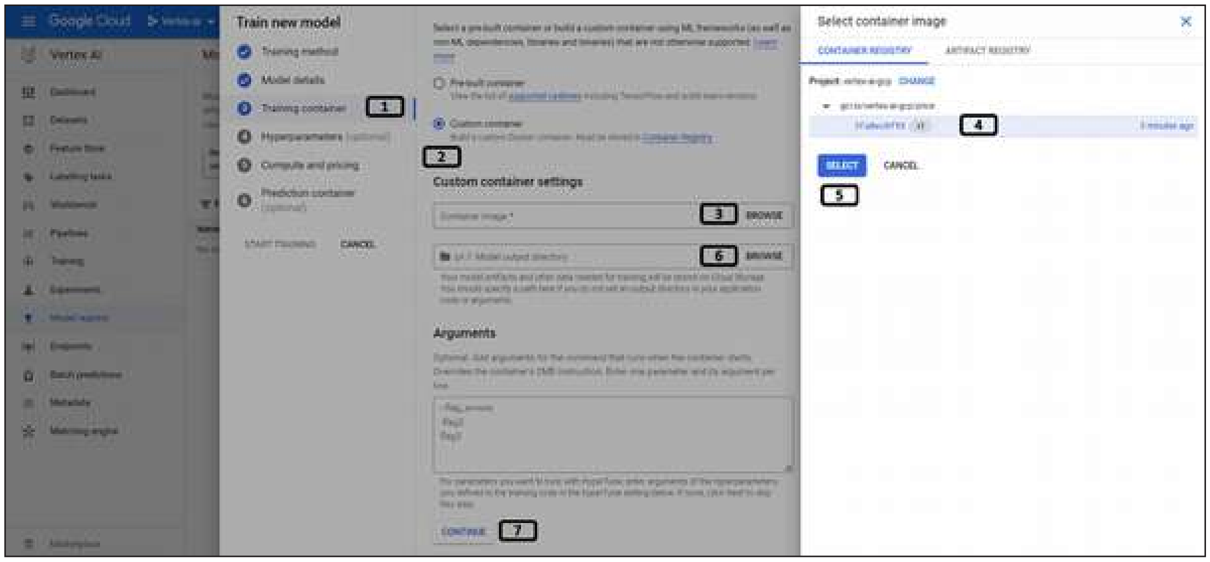


Figure 4.29: Container details for custom model

1. After **Model details** selection, we have to enter the training container details.
2. Select **Custom container**.
3. Click on **BROWSE** to select the image created in previous section.
4. Choose the image created.
5. Click on **SELECT**.
6. Select the cloud storage bucket to store model artifacts (since we have already chosen bucket in the code, this can be left bank).
7. Click on **CONTINUE**.

Step 5: Hyperparameter tuning

Next step is for hyperparameters. Hyperparameters will be covered in detail in the upcoming chapter. Follow the steps as shown in *Figure 4.30* to skip this for now:

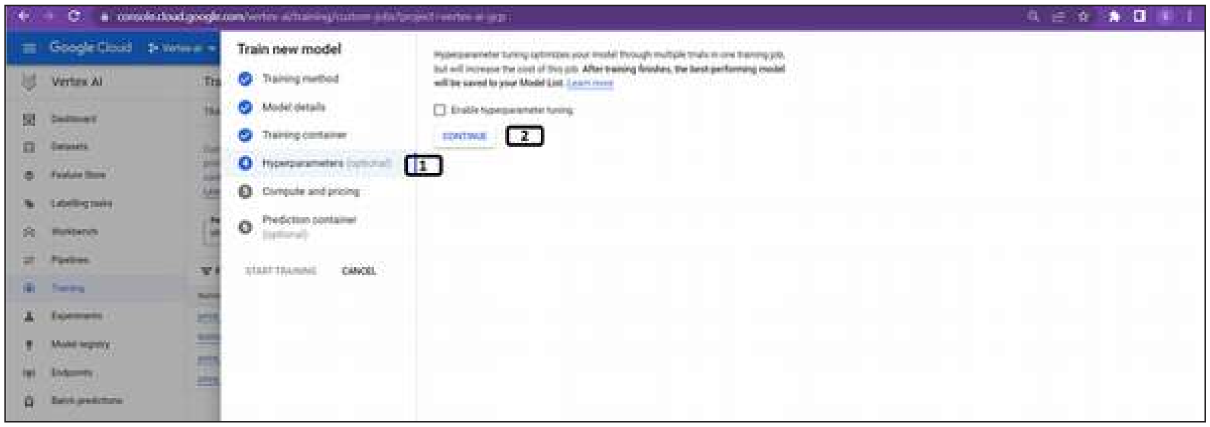
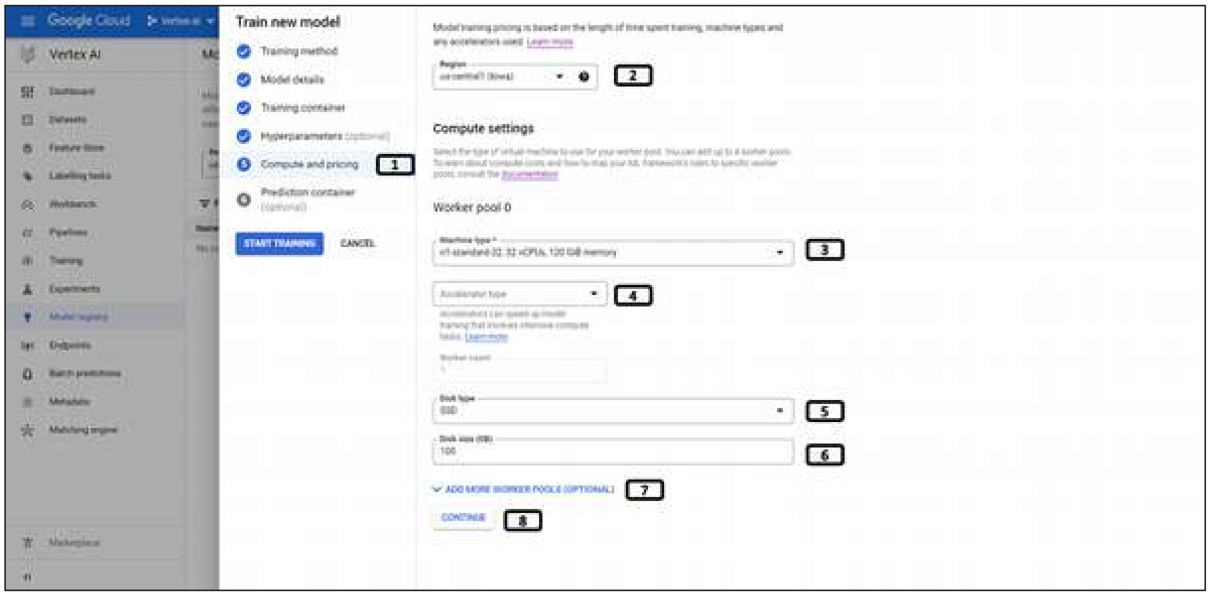


Figure 4.30: Hyperparameter tuning – optional

1. After **Training container** selection, **Hyperparameters** an optional setting can be configured.
2. Click **CONTINUE** (do not enable the hyperparameter tuning).

Step 6: Compute and pricing

Next step is to provide compute and pricing details. Follow the steps shown in *Figure 4.31* to select the machine type for training:



Region: us-central1 (Iowa)
Machine type: n1-standard-32, 32 vCPUs, 120 GiB memory
Accelerator type
Worker count: 1
Disk type: SSD
Disk size (GB): 100

Figure 4.31: *Compute and pricing for custom model*

1. Select **Compute and pricing**.
2. Select the **Region** where the model training is to happen.
3. Select the **Machine type** for the model training (choose the machine type based on how complicated your model training is).
4. **Acceleration type** is optional and can be chosen for complex model trainings.
5. **Disk type** can be chosen between standard and **SSD**.
6. **Disk size** of the VM has to be entered.
7. More **Worker nodes** can be added for the model training if the model is complex.
8. Click **CONTINUE**.

Step 7: Prediction container

Last step in initiating the custom model training job is to select the **Prediction container** and it is optional. Follow the steps in the figure to start the model training:

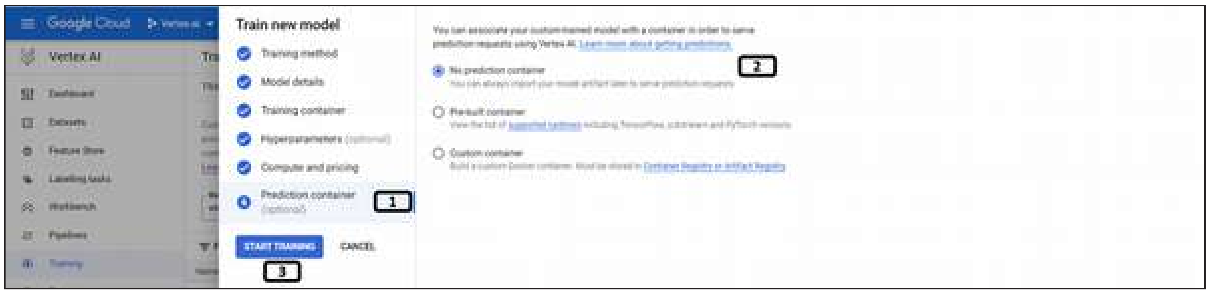


Figure 4.32: *Prediction container details – optional*

1. Select **Prediction container**.
2. Select no prediction container (this will not deploy the model on to the end point; however, users can deploy the model later as well).
3. Click on **START TRAINING**.

Completion of custom model training job

Based on the complexity of the model, model training might take few mins to few hours. Once the model training is completed it will be listed under the training section and also the model will be available in the model section.

Training job will be listed under training section of vertex AI as shown in *Figure 4.33*:

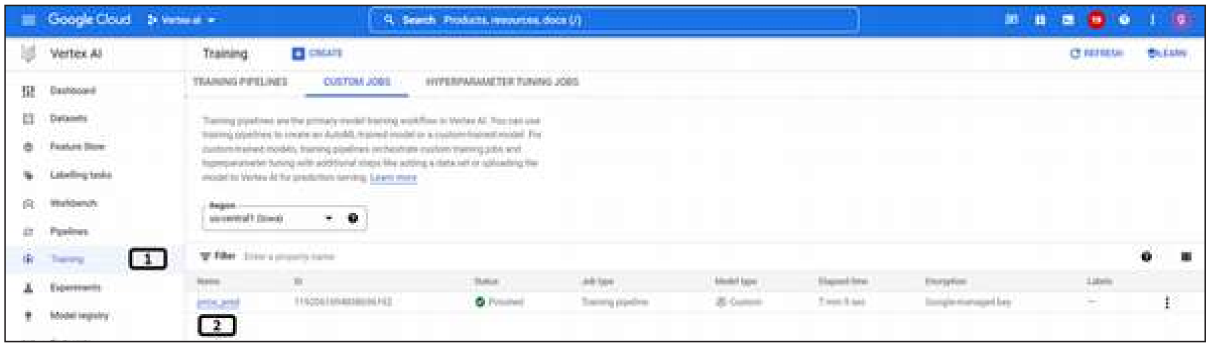
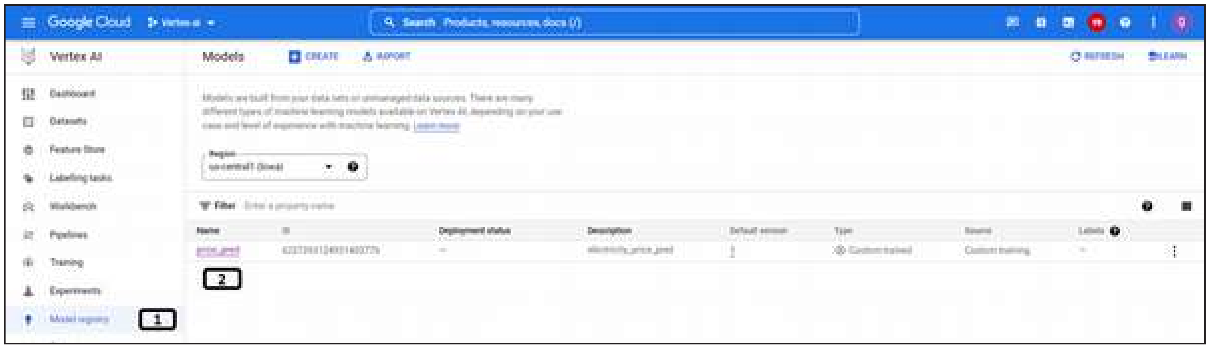


Figure 4.33*: Training module containing training jobs*

1. Select **Training** module under **Vertex AI**.
2. Training job will be listed under model type to be custom.

Trained model will be listed under model registry of vertex AI as shown in *Figure 4.34:*



Models are built from your data sets or unmanaged data sources. There are many different types of machine learning models available on Vertex AI, depending on your use case and level of experience with machine learning. Learn more

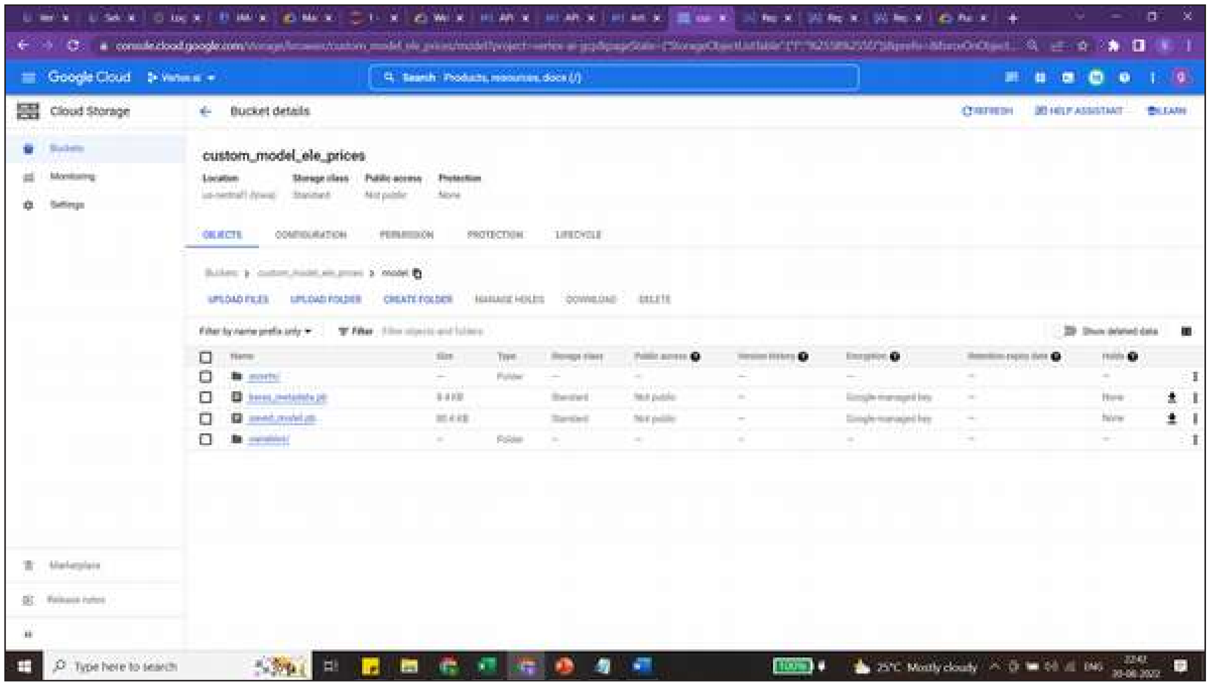
Region: us-central1 (Iowa)

Name	ID	Deployment status	Description	Default version	Type	Source	Labels
price_pred	6237393124931403776	—	electricity_price_pred	1	Custom trained	Custom training	—

Figure 4.34: *Model registry containing trained model*

1. Select **Model registry** of Vertex AI.
2. Trained model will be listed under custom trained type.

Model will be exported along with the artifacts to the cloud storage bucket as shown in *Figure 4.35* (bucket details were mentioned in the Python code):



custom_model_ele_prices

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

Buckets > custom_model_ele_prices > model

Name	Size	Type	Storage class	Public access	Version history	Encryption	Retention expiry date	Holds
assets/	—	Folder	—	—	—	—	—	—
keras_metadata.pb	8.4 KB		Standard	Not public	—	Google-managed key	—	None
saved_model.pb	80.4 KB		Standard	Not public	—	Google-managed key	—	None
variables/	—	Folder	—	—	—	—	—	—

Figure 4.35: *Cloud storage containing trained model and artifacts*

Deletion of resources

All the compute resources are expensive. Follow the steps as shown in Figure 4.36 to delete the notebook instance. Follow the steps in the previous chapter to delete the model from the model registry, training pipeline and model stored in the cloud storage:

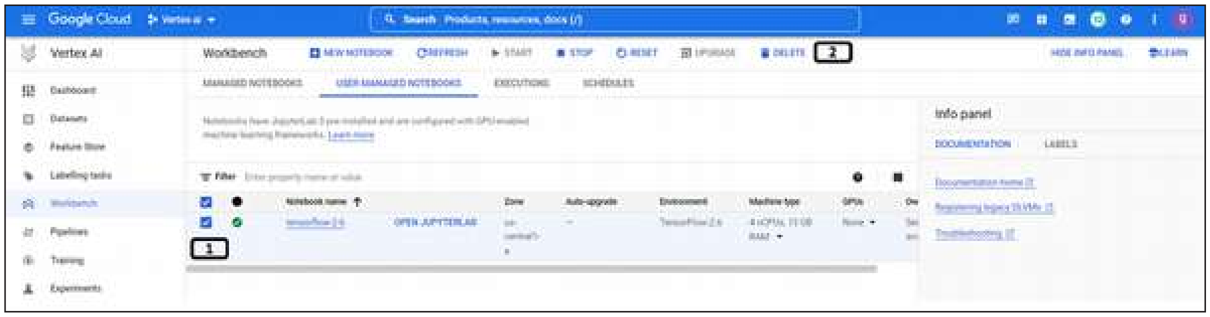


Figure 4.36: Deletion of notebook instance

1. Select the notebook name.
2. Click **DELETE** (A pop up will appear for users confirmation).

Conclusion

In this chapter we learnt about workbench in detail with various components of it. We have covered how we can create python files in the workbench, how to create image using Dockerfile, how to submit custom model training using vertex AI of GCP.

In the next chapter we will start understanding how hyperparameter tuning can be performed for the custom models.

Questions

1. What is the difference between managed notebooks and user managed notebooks?
2. Why is service account needed while submitting custom training job?
3. Can users take the custom trained models for deployment?

CHAPTER 5

Vertex AI Custom Model Hyperparameter and Deployment

Introduction

In the previous chapter, we started working on the custom model building on **Google Cloud Platform** (**GCP**) using Vertex AI components. In this chapter, we will see how to create a custom job with hyperparameter tuning, and how to initiate the training job using Python code.

Structure

In this chapter, we will cover the following topics:

- Hyperparameter in machine learning
- Working of hyperparameters tuning
- Vertex AI Vizier
- Data for building custom model
- Creation of workbench
- Creation of Dockerfile
- Model building code
- Image creation

- Submitting the custom model training job
- Completion of custom model training job
- Model importing
- Model deployment and predictions
- Submitting training job with Python SDK
- Deletion of resources

Objectives

By the end of this chapter, the users will be able to create custom jobs with hyperparameter tuning with **Graphical User Interface** (**GUI**) and using the Python SDK.

Hyperparameter in machine learning

The values of the hyperparameters regulate the learning process, while also deciding the values of the model parameters that a learning algorithm ultimately learns. The learning process and the model parameters that are produced as a consequence of it are both under the control of hyperparameters which are high-level parameters.

Before machine learning model training can even begin, data scientists must first choose and configure the hyperparameter values that the learning algorithm will utilize. This is done as part of the machine learning model training process. In this context, hyperparameters are considered to exist outside of the model because the model's values cannot be altered during the learning or training process.

The learning algorithm makes use of hyperparameters while it is learning, but they are not included in the model that is produced as a consequence of this learning. At the conclusion of the learning process, we will have the trained model parameters, which are, in a practical sense, what we mean when we talk about the model. The model does not include the hyperparameters that were adjusted throughout the training process. We are only aware of the model parameters that were learned about; we are unable to determine, for example, the hyperparameter values that were used to train a model based on the model itself.

Examples of hyperparameters include learning rate, number of hidden layers in the neural network and **`n_estimators`**, max depth, and max features for tree-based algorithms.

On the other hand, parameters are model-specific. They are learned or inferred solely from training data when the algorithm attempts to map input characteristics to labels or targets.

Model training begins with initializing parameters (random values or set to zeros). During training/learning, an optimization algorithm updates starting values (for example, gradient descent). As learning progresses, the learning algorithm updates parameter values but not hyperparameters.

Examples of parameters are weights and biases of a neural network and the cluster centroids in clustering.

Working of hyperparameters tuning

Running your training application several times with different values for your selected hyperparameters, all within the bounds you define, is how hyperparameter tuning works. Vertex AI remembers the outcomes of previous tests and uses that information to improve performance in new ones. Following the completion of the task, users will be provided with a summary of all trials and the optimal value configuration based on the criteria you set.

To tune hyperparameters, the Vertex AI must be in direct contact with the training program. The training application specifies all the information that your model requires. Users are expected to provide all the hyperparameters that need to be considered in the process of optimization and the outcome metrics such as accuracy, r2score, and so on, users wish optimize.

The Python package called **`cloudml-hypertune`**, aids in passing metrics to Vertex AI. More information about the **`cloudml-hypertune`** package is available in the following link:

https://pypi.org/project/cloudml-hypertune/

Vertex AI Vizier

You may fine-tune hyperparameters in complex machine learning models with the aid of Vertex AI Vizier, a black-box optimization tool. It may be challenging and time-consuming to manually adjust ML models when they contain several distinct hyperparameters. By adjusting the hyperparameters, Vertex AI Vizier maximizes the output of the model.

Users need to provide study configuration for Vizier to optimize machine learning models. Study configuration includes metrics that need to be optimized (such as accuracy, r2 score, and so on) and AI hyperparameters that will influence the metrics.

Study configuration is implemented and is then called a *Study*. A trial employs a study's aims (metrics) and input values (hyperparameters or parameters). A trial is a collection of inputs that produces a quantifiable output called measurement. A study continues until a specific number of trials is reached or is stopped. Importantly, Vertex AI Vizier proposes trial inputs but does not run them.

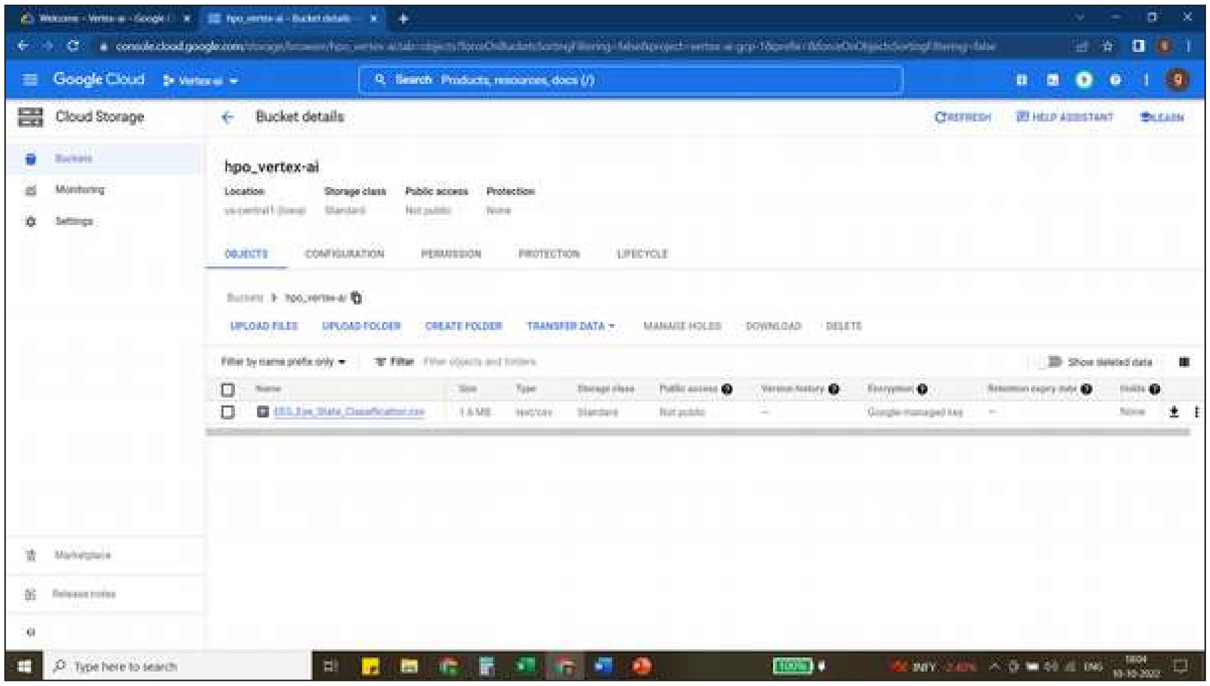
Hyperparameter tuning for custom training is a built-in feature that uses Vertex AI Vizier for training jobs.

Data for building custom model

For this exercise data is downloaded from Kaggle (link is provided further down) and the dataset is listed under CC0: Public domain licenses. Data contains various measurements from EEG and the state of the eye is captured via the camera. 1 indicates closed eye and 0 indicates open eye.

https://www.kaggle.com/datasets/robikscube/eye-state-classification-eeg-dataset

`hpo_vertex-ai` bucket is created under **`us-central1`** (single region) and .CSV file is uploaded from to the bucket as shown in *Figure 5.1*:



hpo_vertex-ai

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

Name	Size	Type	Storage class	Public access	Version history	Encryption	Retention expiry date	Holds
EEG_Eye_State_Classification.csv	1.6 MB	text/csv	Standard	Not public	–	Google-managed key	–	None

Figure 5.1*: Data uploaded to cloud storage*

Custom model building with hyperparameter tuning activity is done in multiple steps (very similar to the exercise of previous chapter with a few more additional steps). We will be developing a Random forest classifier using Sklearn package.

Creation of workbench

Follow the steps given in the previous chapter for creation of the workbench, but instead of choosing TensorFlow enterprise machine while creating, choose Python3 machine.

Step 1: Landing page of workbench

Landing page of the workbench as shown in the *Figure 5.2*:

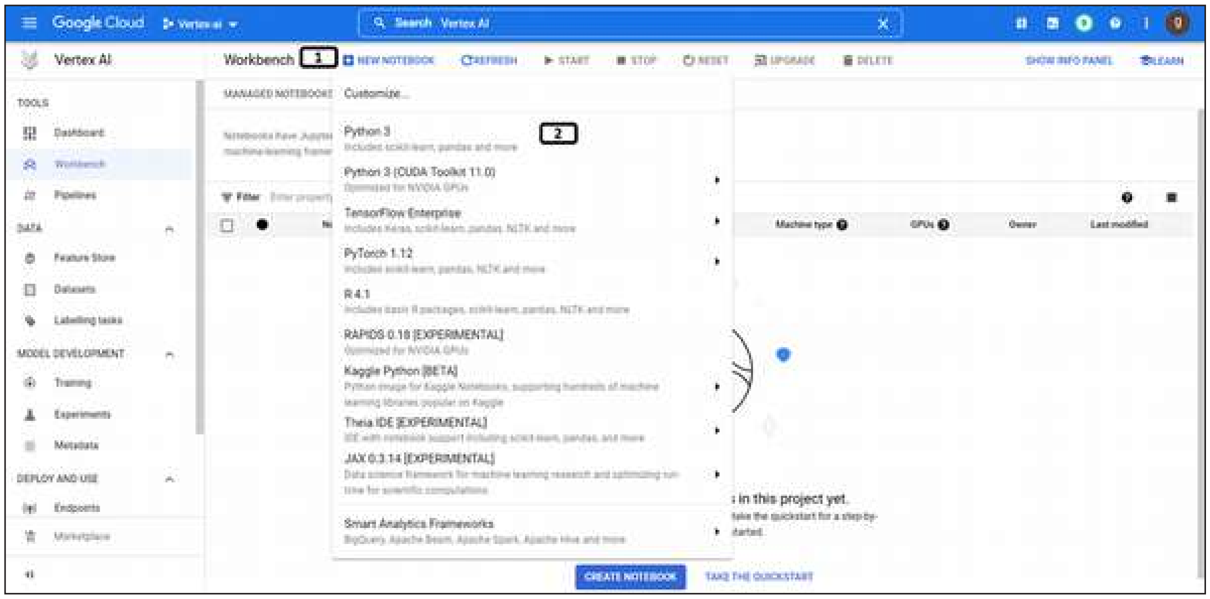


***Figure 5.2**: Creation of workbench*

Now follow the corresponding points:

1. Click **New Notebook**.
2. Select **Python 3**.
3. In the next pop up, provide the name for the workbench and select the region to be `us-central1` and click Create.

Step 2: Notebook creation complete

It will take few mins to complete the creation of notebook and AI the Jupyter lab instance. Once it loads, we will see the screen shown in *Figure 5.3*.

1. Click **OPEN JUPYTERLAB**:

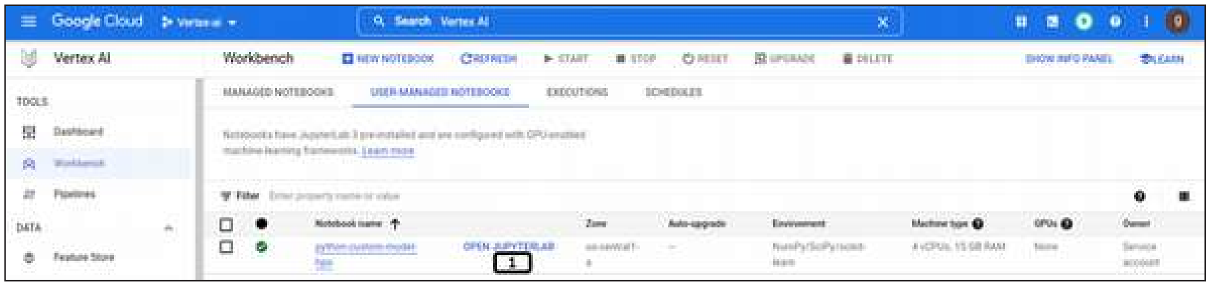


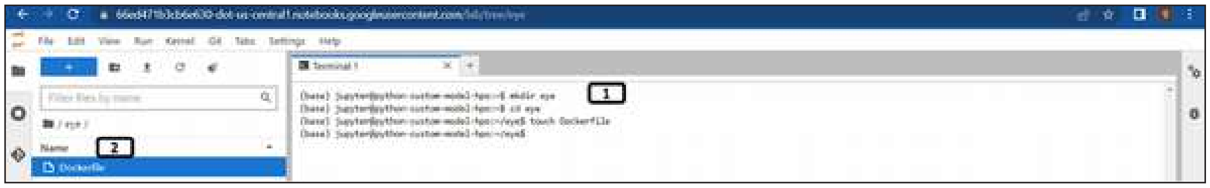
Figure 5.3: Workbench successfully created

Creation of Dockerfile

Launch the Terminal (refer to *Figure 4.6* in the previous chapter, point no. 11).

Step 1: Directory creation

Figure 5.4 illustrated creation of a directory in the workbench:



```
(base) jupyter@python-custom-model-tpo:~$ mkdir eye
(base) jupyter@python-custom-model-tpo:~$ cd eye
(base) jupyter@python-custom-model-tpo:~/eye$ touch Dockerfile
(base) jupyter@python-custom-model-tpo:~/eye$
```

Figure 5.4: Directory creation in the terminal

Follow the corresponding points:

1. Type the following commands one after the other in the Terminal window. We create a new directory called **eye**, and create **Dockerfile** inside the eye directory:

   ```
   mkdir eye
   cd eye
   touch Dockerfile
   ```

2. The **Dockerfile** will be created inside the **eye** folder. This file will have no contents; double click on the file to open it.

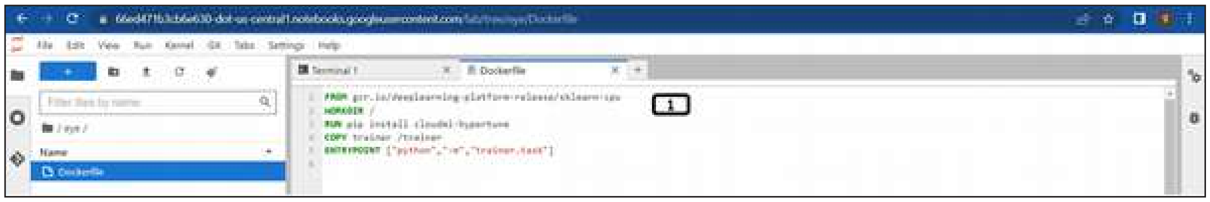
Step 2: Edit Dockerfile

To edit **Dockerfile**, follow the given steps:

1. Enter the following commands in the Dockerfile, as shown in the *Figure 5.5*:

   ```
   FROM gcr.io/deeplearning-platform-release/sklearn-cpu
   WORKDIR /
   ```

```
RUN pip install cloudml-hypertune
COPY trainer /trainer
ENTRYPOINT ["python","-m","trainer.task"]
```



```
1 FROM gcr.io/deeplearning-platform-release/sklearn-cpu
2 WORKDIR /
3 RUN pip install cloudml-hypertune
4 COPY trainer /trainer
5 ENTRYPOINT ["python","-m","trainer.task"]
6
```

***Figure 5.5**: Editing Dockerfile*

2. **`cloudml-hypertune`** package will not be available, and so it needs to be installed while creating the docker image, line number 3 in the former code installs the package while creating.
3. Save the **`Dockerfile`** and close the file.

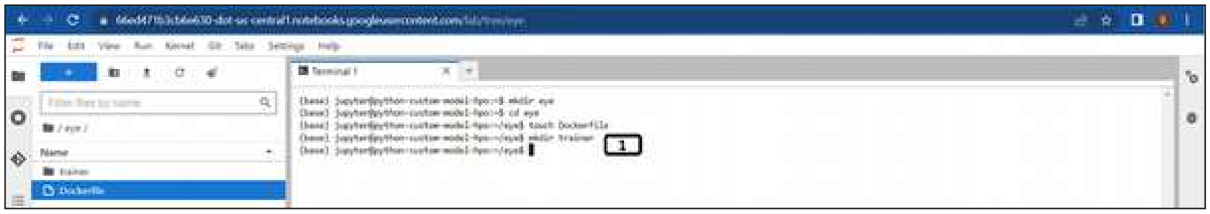
Model building code

Once the Dockerfile is created, we need to work on the python code for the model building. The task is to build a classification model. Python code will be copied into the container and submitted for training job.

Let us go back to the terminal of the workbench and follow the given steps to create the Python code for model building.

Step 1: Create a trainer folder

1. Type the commands in the Terminal as shown in the *Figure 5.6*:



```
(base) jupyter@python-custom-model-hpo:~$ mkdir eye
(base) jupyter@python-custom-model-hpo:~$ cd eye
(base) jupyter@python-custom-model-hpo:~/eye$ touch Dockerfile
(base) jupyter@python-custom-model-hpo:~/eye$ mkdir trainer
(base) jupyter@python-custom-model-hpo:~/eye$
```

***Figure 5.6**: Trainer folder creation in Terminal*

2. Then type the following command in the Terminal to create directory:

   `mkdir trainer`

   The command will create a new folder called **`Trainer`**. Open this folder.

Step 2: Create a Python file

Follow the given steps and refer to *Figure 5.7* to create a Python file (create the python file inside the trainer folder):

1. Click on New launcher, which is the **+** button.
2. Double click on the **Python** file.

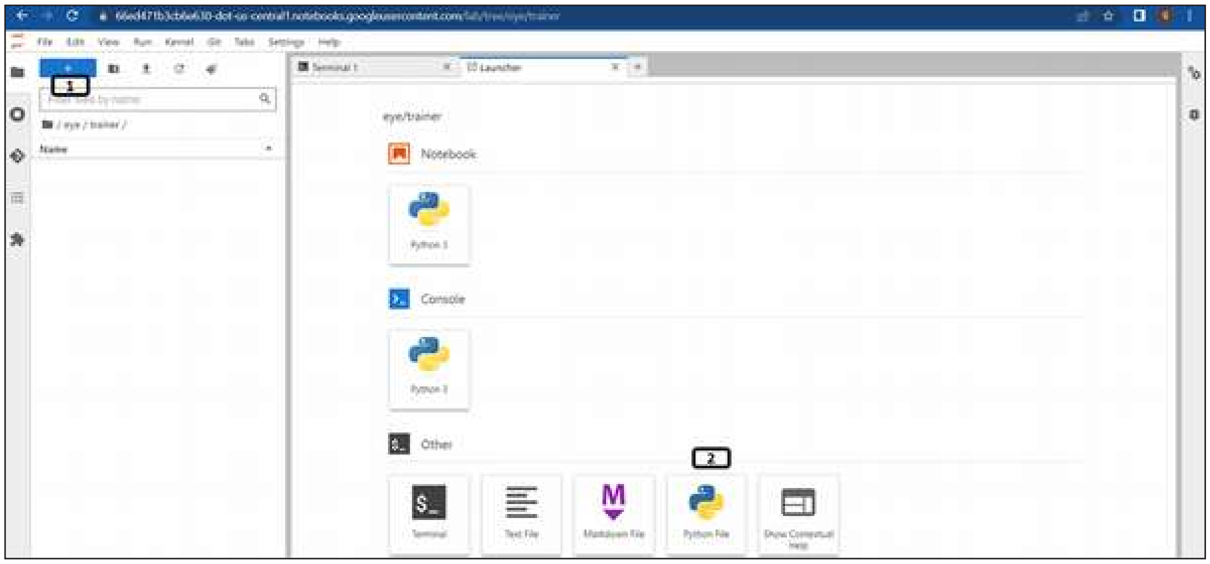


***Figure 5.7**: Python file creation on workbench for model building*

Step 3: Python code

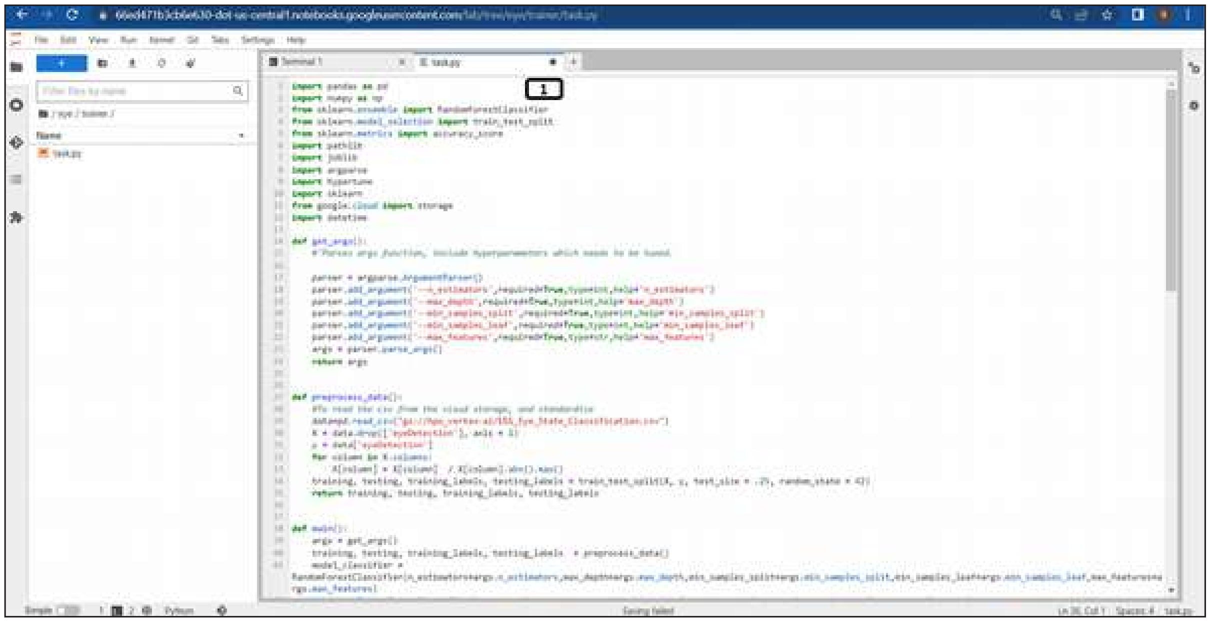
A Python file will be created inside the trainer folder. Write the following given Python code in the py file:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pathlib
import pickle
import argparse
import hypertune
import sklearn
from google.cloud import storage
```

```
import datetime

def get_args():

    #’Parses args function, include hyperparameters which needs to be 
    tuned.

    parser = argparse.ArgumentParser()

    parser.add_argument(‘--n_estimators’,required=True,type=int,help=’n_
    estimators’)

    parser.add_argument(‘--max_depth’,required=True,type=int,help=’max_
    depth’)

    parser.add_argument(‘--min_samples_
    split’,required=True,type=int,help=’min_samples_split’)

    parser.add_argument(‘--min_samples_
    leaf’,required=True,type=int,help=’min_samples_leaf’)

    parser.add_argument(‘--max_
    features’,required=True,type=str,help=’max_features’)

    args = parser.parse_args()

    return args

def preprocess_data():

    #To read the csv from the cloud storage, and standardize

    data=pd.read_csv(“gs://hpo_vertex-ai/EEG_Eye_State_Classification.
    csv”)

    X = data.drop([‘eyeDetection’], axis = 1)

    y = data[‘eyeDetection’]

    for column in X.columns:

        X[column] = X[column]  / X[column].abs().max()

    training, testing, training_labels, testing_labels = train_test_
    split(X, y, test_size = .25, random_state = 42)

    return training, testing, training_labels, testing_labels
```

```
def main():

    args = get_args()

   training, testing, training_labels, testing_labels = preprocess_data()

    model_classifier = RandomForestClassifier(n_estimators=args.n_
    estimators,max_depth=args.max_depth,min_samples_split=args.
    min_samples_split,min_samples_leaf=args.min_samples_leaf,max_
    features=args.max_features)

    model_classifier.fit(training,training_labels)

    y_pred = model_classifier.predict(testing)

    acc = accuracy_score(testing_labels, y_pred)

    hpt = hypertune.HyperTune()

    hpt.report_hyperparameter_tuning_metric(

        hyperparameter_metric_tag='accuracy',

        metric_value=acc)

    artifact_filename = "model.pkl"

    pickle.dump(model_classifier, open(artifact_filename, "wb"))

    BUCKET = 'hpo_vertex-ai'

    gcs = storage.Client(project="Vertex-ai")

    buck=gcs.bucket(BUCKET)

    blob = buck.blob(artifact_filename)

    blob.upload_from_filename(artifact_filename)

if __name__ == "__main__":

    main()
```

Refer to the following *Figure 5.8*:



```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pathlib
import joblib
import argparse
import hypertune
import sklearn
from google.cloud import storage
import datetime

def get_args():
    #'Parses args function, include hyperparameters which needs to be tuned

    parser = argparse.ArgumentParser()
    parser.add_argument('--n_estimators',required=True,type=int,help='n_estimators')
    parser.add_argument('--max_depth',required=True,type=int,help='max_depth')
    parser.add_argument('--min_samples_split',required=True,type=int,help='min_samples_split')
    parser.add_argument('--min_samples_leaf',required=True,type=int,help='min_samples_leaf')
    parser.add_argument('--max_features',required=True,type=str,help='max_features')
    args = parser.parse_args()
    return args


def preprocess_data():
    #To read the csv from the cloud storage, and standardize
    data=pd.read_csv("gs://hpo_vertex-ai/EEG_Eye_State_Classification.csv")
    X = data.drop(['eyeDetection'], axis = 1)
    y = data['eyeDetection']
    for column in X.columns:
        X[column] = X[column]  / X[column].abs().max()
    training, testing, training_labels, testing_labels = train_test_split(X, y, test_size = .25, random_state = 42)
    return training, testing, training_labels, testing_labels


def main():
    args = get_args()
    training, testing, training_labels, testing_labels  = preprocess_data()
    model_classifier =
RandomForestClassifier(n_estimators=args.n_estimators,max_depth=args.max_depth,min_samples_split=args.min_samples_split,min_samples_leaf=args.min_samples_leaf,max_features=a
rgs.max_features)
```

Figure 5.8: Python code for model building

Follow the corresponding points:

1. Type the Python code.
2. Right click and rename the Python file name to **`task.py`**.
3. Save the python file and close the file.

The Python code consists of codes, to import required packages. The function **`get args`** contains the hyperparameters that need to be tuned and hyperparameters will be passed as arguments. In our exercise, we are considering five hyperparameters for the tuning they are:

- **`n_estimator`**
- **`max_depth`**
- **`min_samples_split`**
- **`min_samples_leaf`**
- **`max_features`**

`max_features` takes a string value as input and all the other hyperparameters take integer value.

Function preprocess data, reads the file, applies basic transformations and returns the split data. The main function builds the model and initiates the hyperparameter tuning of the model and saves the model to cloud storage.

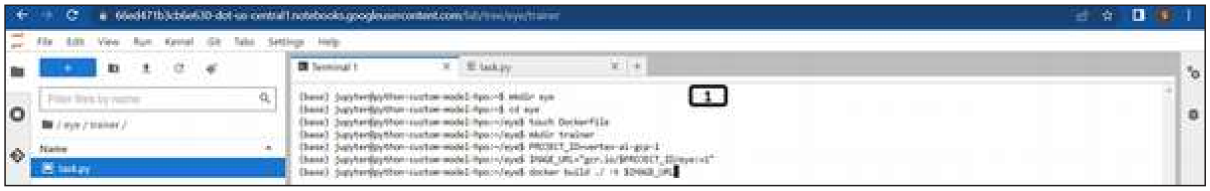
Image creation

Now that we have completed the creation of the Dockerfile and python code for the model building, we can proceed with image creation. Follow the given steps for the same.

Step 1: Image creation using docker build

Type the following commands in the terminal, as shown in *Figure 5.9*, for image creation:

```
PROJECT_ID=vertex-ai-gcp-1

IMAGE_URL=”gcr.io/$PROJECT_ID/eye:v1”

docker build ./ -t $IMAGE_URL
```



```
(base) jupyter@python-custom-model-hpo:~$ mkdir eye
(base) jupyter@python-custom-model-hpo:~$ cd eye
(base) jupyter@python-custom-model-hpo:~/eye$ touch Dockerfile
(base) jupyter@python-custom-model-hpo:~/eye$ mkdir trainer
(base) jupyter@python-custom-model-hpo:~/eye$ PROJECT_ID=vertex-ai-gcp-1
(base) jupyter@python-custom-model-hpo:~/eye$ IMAGE_URL="gcr.io/$PROJECT_ID/eye:v1"
(base) jupyter@python-custom-model-hpo:~/eye$ docker build ./ -t $IMAGE_URL
```

***Figure 5.9**: Image creation process*

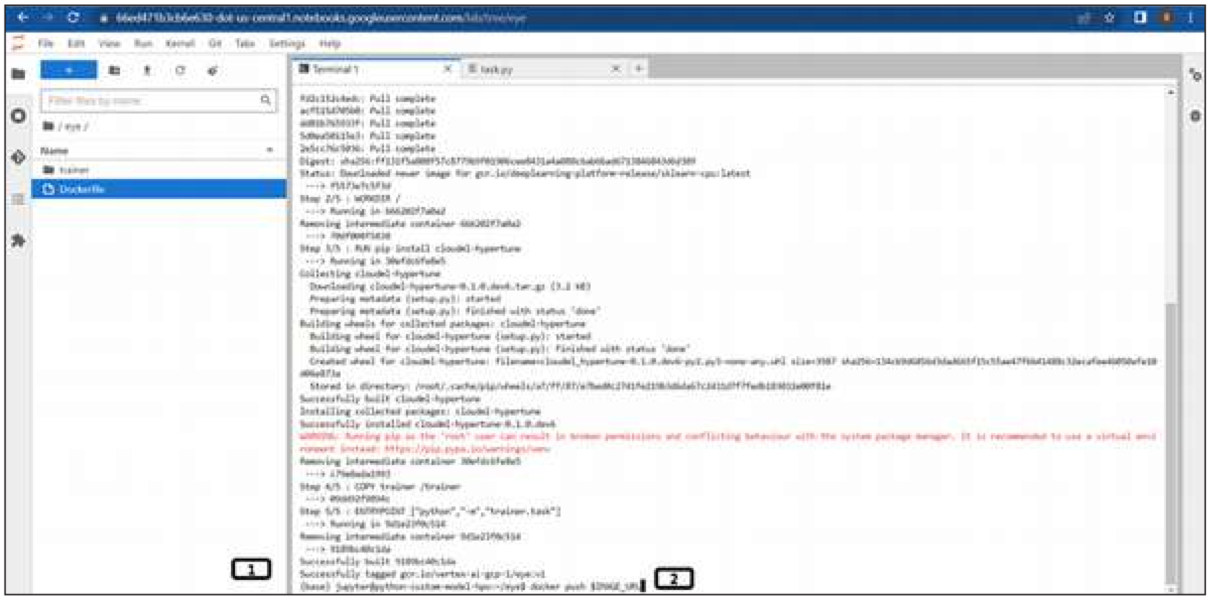
It will take few mins to build the image from the Dockerfile. Once build is complete, a success message will be displayed in the terminal as shown in *Figure 5.10.* (point number 1).

Step 2: Pushing the docker image to the container registry

Now that the image is created (the success message for image creation can be seen in *Figure 5.10*), type the following command to push the image to the container registry:

```
docker push $IMAGE_URL
```

Refer to the following figure:



```
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
[illegible]: Pull complete
Digest: sha256:[illegible]
Status: Downloaded newer image for gcr.io/deeplearning-platform-release/sklearn-cpu:latest
 ---> [illegible]
Step 2/5 : WORKDIR /
 ---> Running in [illegible]
Removing intermediate container [illegible]
 ---> [illegible]
Step 3/5 : RUN pip install cloudml-hypertune
 ---> Running in [illegible]
Collecting cloudml-hypertune
  Downloading cloudml-hypertune-0.1.0.dev6.tar.gz (3.2 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Building wheels for collected packages: cloudml-hypertune
  Building wheel for cloudml-hypertune (setup.py): started
  Building wheel for cloudml-hypertune (setup.py): finished with status 'done'
  Created wheel for cloudml-hypertune: filename=cloudml_hypertune-0.1.0.dev6-py2.py3-none-any.whl size=[illegible] sha256=[illegible]
[illegible]
  Stored in directory: /root/.cache/pip/wheels/[illegible]
Successfully built cloudml-hypertune
Installing collected packages: cloudml-hypertune
Successfully installed cloudml-hypertune-0.1.0.dev6
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual envi
ronment instead: https://pip.pypa.io/warnings/venv
Removing intermediate container [illegible]
 ---> [illegible]
Step 4/5 : COPY trainer /trainer
 ---> [illegible]
Step 5/5 : ENTRYPOINT ["python","-m","trainer.task"]
 ---> Running in [illegible]
Removing intermediate container [illegible]
 ---> [illegible]
Successfully built [illegible]
Successfully tagged gcr.io/vertex-ai-gcp-1/eye:v1
(base) jupyter@python-custom-model-hpo:~/eye$ docker push $IMAGE_URI
```

Figure 5.10: Image to project container registry

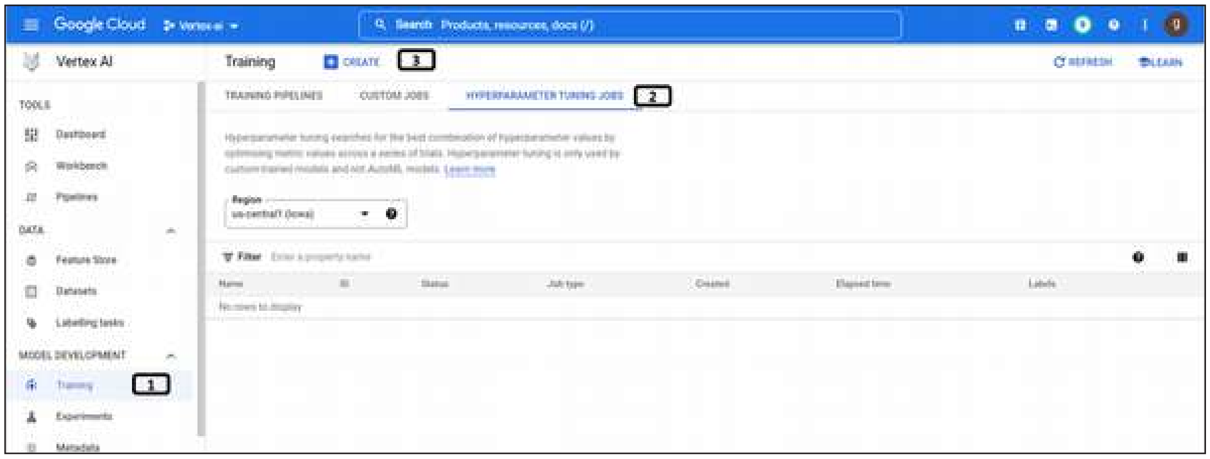
The image will be available in the container registry of the project (for checking, follow the steps described in the previous chapter).

Submitting the custom model training job

Steps for the custom model training with hyperparameter tuning are very similar to the ones described in the previous chapter, except that we will be enabling the hyperparameter tuning option. Follow the given steps to submit the training job.

Step 1: Training module of vertex AI

Open the training module of the vertex AI, as shown in *Figure 5.11*:



Hyperparameter tuning searches for the best combination of hyperparameter values by optimizing metric values across a series of trials. Hyperparameter tuning is only used by custom trained models and not AutoML models. Learn more

Region: us-central1 (Iowa)

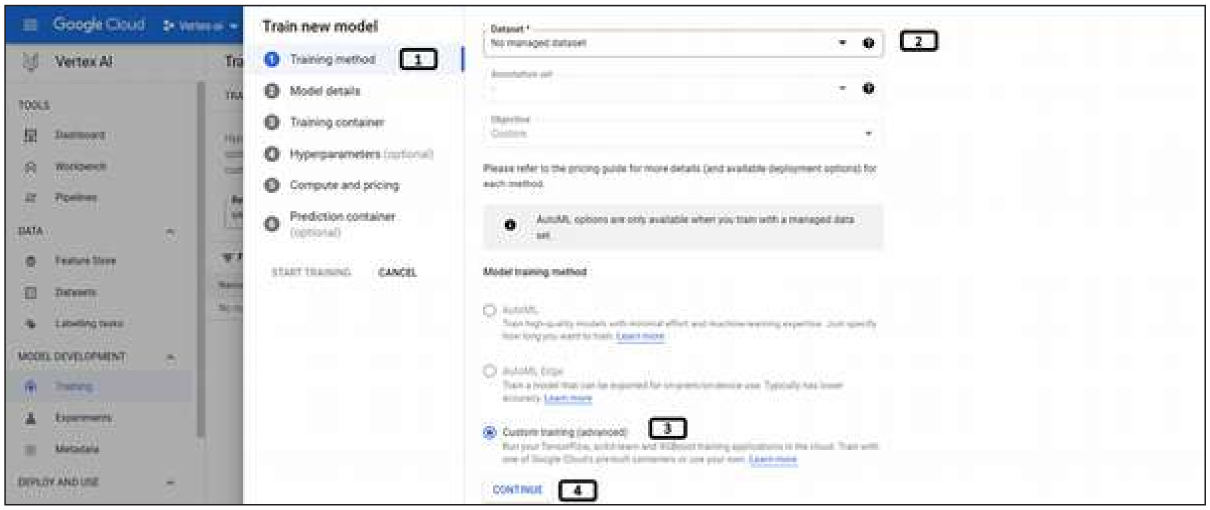
No rows to display

Figure 5.11: Training module of Vertex AI

1. Select the **Training** module.
2. Select **HYPERPARAMETER TUNING JOBS**.
3. Click on **CREATE**.

Step 2: Selecting training method

The **Train new model** page will appear, as shown *Figure 5.12*:



Dataset *
No managed dataset

Annotation set

Objective
Custom

Please refer to the pricing guide for more details (and available deployment options) for each method.

AutoML options are only available when you train with a managed data set.

Model training method

AutoML
Train high-quality models with minimal effort and machine learning expertise. Just specify how long you want to train. Learn more

AutoML Edge
Train a model that can be exported for on-prem/on-device use. Typically has lower accuracy. Learn more

[✓ Custom training (advanced)]
Run your TensorFlow, scikit-learn and XGBoost training applications in the cloud. Train with one of Google Cloud's prebuilt containers or use your own. Learn more

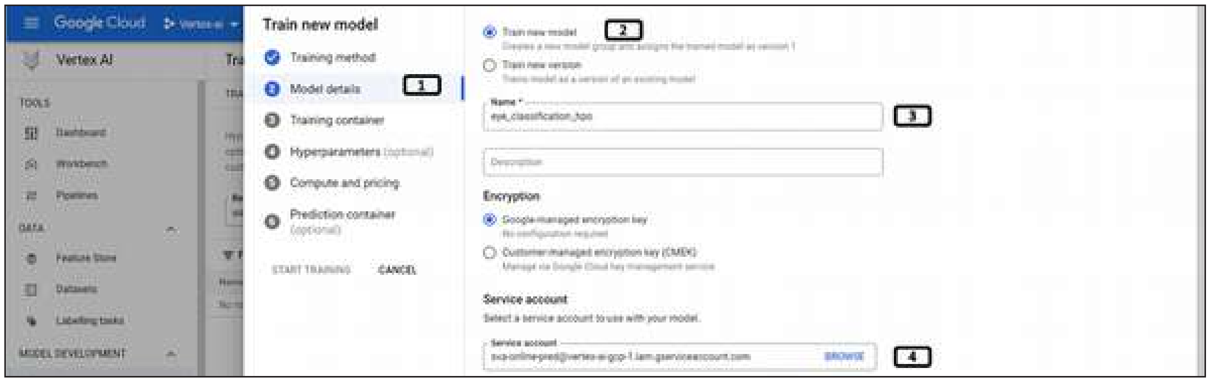
Figure 5.12: *Training method selection for custom model*

Follow the corresponding points to configure training method:

1. Select the **training method**.
2. Select **No managed dataset** under the **Dataset** selection.
3. Select **Custom Training**.
4. Click on **Continue**.

Step 3: Model details

Select the model details to train the model, as shown in the *Figure 5.13*:



[✓ Train new model]
Creates a new model group and assigns the trained model as version 1

Train new version
Trains model as a version of an existing model

Name *
eye_classification_hpo

Description

Encryption

[✓ Google-managed encryption key]
No configuration required

Customer-managed encryption key (CMEK)
Manage via Google Cloud key management service

Service account
Select a service account to use with your model.

Service account
[illegible]@vertex-ai-gcp-1.iam.gserviceaccount.com

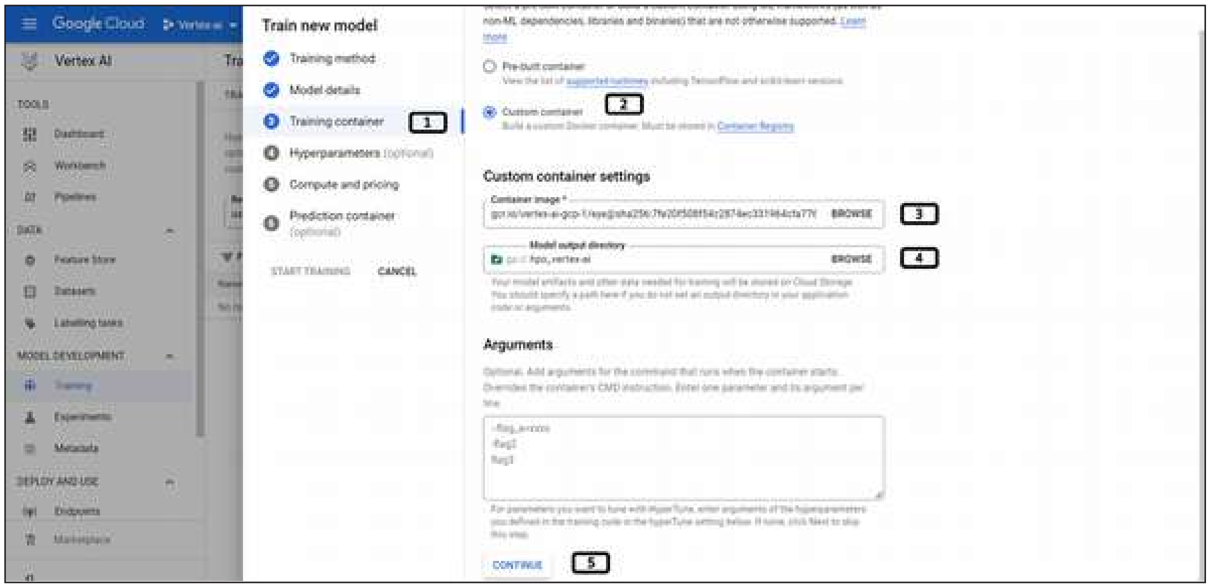
Figure 5.13: *Model details for custom model*

Follow the corresponding points:

1. After training method selection, we have to enter the **Model details**.
2. Select **Train new model**.
3. Provide a **Name** for the model.
4. Select the **Service account** associated with the compute (users can also choose service account which we had created in *Chapter 2, Introduction to Vertex AI and & AutoML Tabular*, but roles to access cloud storage need to be added.)
5. Scroll down and click on **Continue**.

Step 4: Training container details

Follow the numbers and their corresponding points in *Figure 5.14* to configure the training container:



Custom container settings

Container image *: gcr.io/vertex-ai-gcp-1/... [illegible]

Model output directory: gs:// hpo_vertex-ai

Arguments

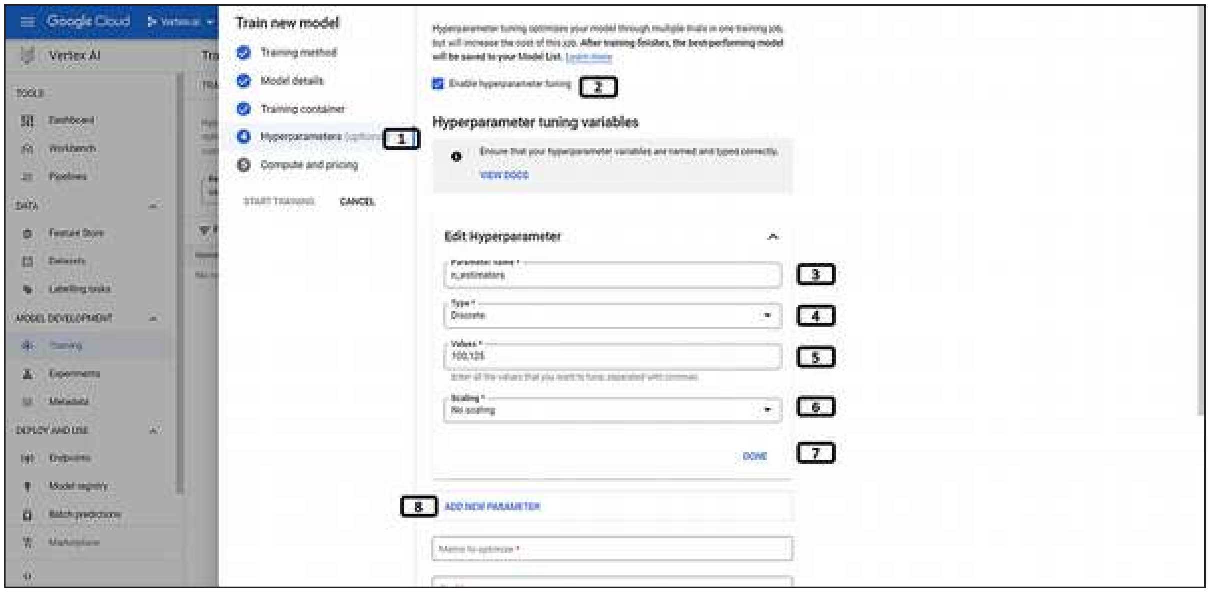
Figure 5.14: *Container details for custom model*

Follow the corresponding points:

1. After **Dodel details** selection, we have to enter the **Training container** details.
2. Select **Custom container**.
3. Browse for the **Container image** that has been pushed to container registry.
4. Select the cloud storage bucket to store model artifacts (since we have already chosen bucket in the code, this can be left bank.)
5. Click on **Continue**.

Step 5: Hyperparameter tuning

The next step is for hyperparameters. Follow the corresponding steps to enable and configure the hyperparameter tuning for model training, as shown in the *Figure 5.15*:



Hyperparameter tuning optimises your model through multiple trials in one training job, but will increase the cost of this job. After training finishes, the best-performing model will be saved to your Model List. Learn more

☑ Enable hyperparameter tuning

Hyperparameter tuning variables

Ensure that your hyperparameter variables are named and typed correctly.

Edit Hyperparameter

Parameter name *: n_estimators

Type *: Discrete

Values *: 100,125

Enter all the values that you want to tune, separated with commas.

Scaling *: No scaling

Metric to optimize *

Figure 5.15: Configuring hyperparameter tuning

Follow the corresponding points:

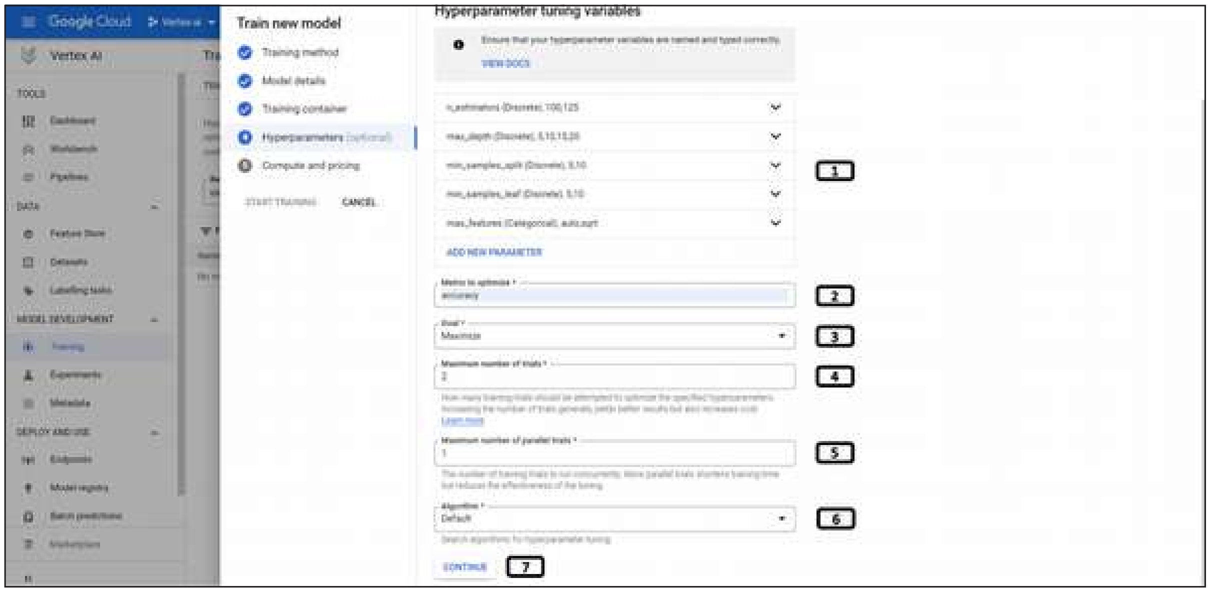
1. After **Training container** selection, an optional setting can be configured in **Hyperparameters**.
2. Select the **Enable hyperparameter tuning** option.
3. Provide the **Parameter name** (need to add the 5 parameters which we have included in the code, individually).
4. Select the **Type** of the input provided for the hyperparameter. The type of hyperparameters it supports are categorical, discrete, double and integer. Select integer for the `n_estimator`.
5. Provide the **Value**, as shown in the *Figure 5.15* (this option changes as per the type.)
6. Select **No scaling** option. (It supports linear, log and reverse log scaling).
7. Click on **Done**. (One of the five hyperparameter are added.)
8. Click on **Add new parameter** to add the details of the other parameters.

Refer to *Table 5.1* and add the details of remaining four parameters:

Parameter name	Type	Value	Scaling
`max_depth`	Discrete	5,10,15,20	No scaling
`min_samples_split`	Discrete	5,10	No scaling
`min_samples_leaf`	Discrete	5,10	No scaling
`max_features`	Categorical	auto, sqrt	N/A

***Table 5.1**: Hyperparameter list*

If needed, change the type of **`max_depth`**, **`min_samples_split`** and **`min_samples_leaf`** to integer for seeing other options. Refer to *Figure 5.16*:



n_estimators (Discrete) 100,125
max_depth (Discrete) 5,10,15,20
min_samples_split (Discrete) 5,10
min_samples_leaf (Discrete) 5,10
max_features (Categorical) auto,sqrt

Metric to optimize *: accuracy
Goal *: Maximize
Maximum number of trials *: 2
Maximum number of parallel trials *: 1
Algorithm *: Default

***Figure 5.16**: Configuring other parameters for hyperparameter tuning*

Refer to the corresponding steps to complete the configuration of the hyperparameter:

1. All the parameters that are added, will be listed.
2. In the code we are optimizing accuracy metrics (to maximize), provide accuracy as the input.
3. The **Goal** is to maximize the accuracy.
4. **Maximum number of trails** is kept as low as 2, since the model is very simple.
5. **Maximum number of parallel trails** is kept to 1 for the same reason.
6. Under **Algorithm**, choose **Default** (random search or grid search can also be chosen). Explanations of different algorithms are listed below

- Random search: A straightforward random search of the realizable region.
- Unspecified algorithm: Not specifying an algorithm. Vertex AI picks between Gaussian process bandits, linear combination search, or its versions.
- Grid search: A viable grid search. This is important if users wish to specify more trials than viable points. If no grid search is specified, Vertex AI may create duplicate options.

7. Click on **CONTINUE**.

NOTE: GCP has restricted training CPUs for the free accounts. Try to keep the maximum trails and parallel trails low. If you get error such as quota metrics exceeded, try reducing the value of these two parameters.

Step 6: Compute and pricing

The next step is to provide compute and pricing details. Refer to *Figure 5.17*:

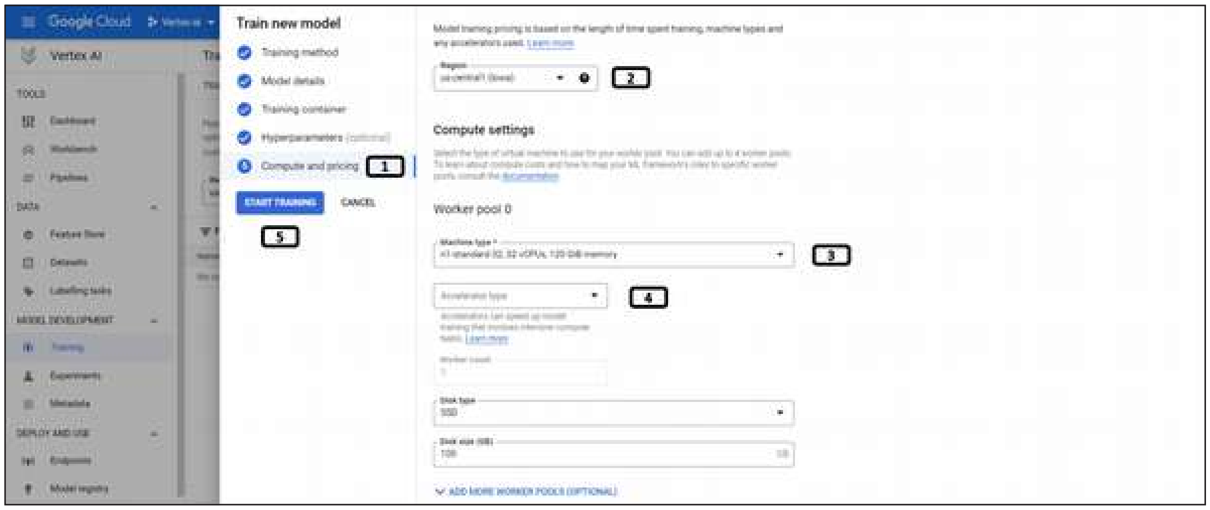


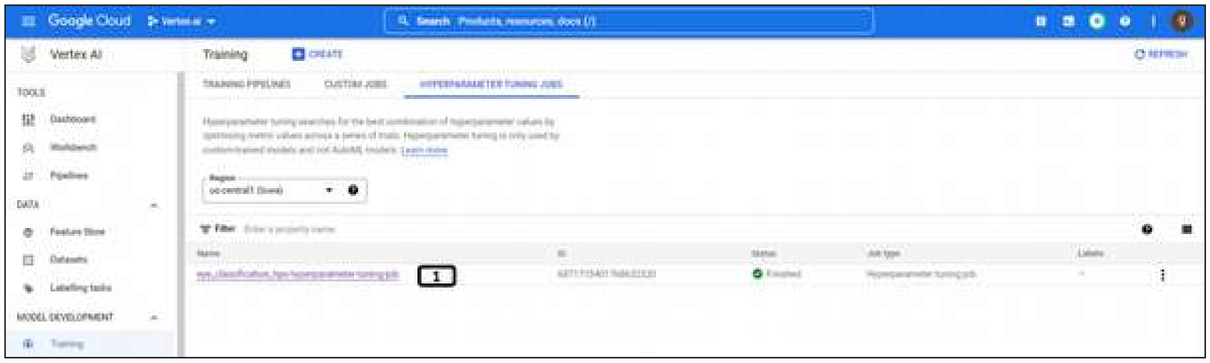
Figure 5.17: Compute and pricing for custom model

Follow the corresponding steps to select the machine type for training:

1. Select **Compute and pricing**.
2. Select the **Region** where the model training is to happen.
3. Select the **Machine type** for the model training (choose the machine type based on how complicated your model training is).
4. **Acceleration type** is optional and can be chosen for complex model trainings.
5. Click **START TRAINING**, to finally start the same.

Completion of custom model training job

Based on the complexity of the model, model training might take anywhere between a few minutes to a few hours. Once the model training is completed, it will be listed under the training section, as shown in the *Figure 5.18*:

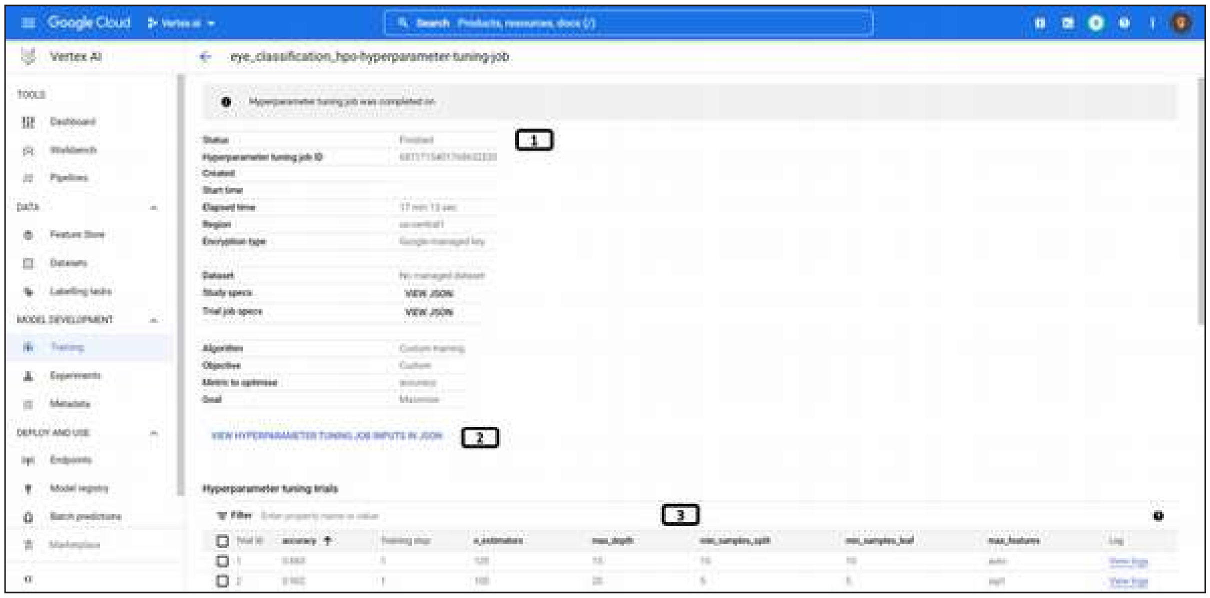


Name	ID	Status	Job type	Labels
eye_classification_hpo-hyperparameter-tuning-job	[illegible]	Finished	Hyperparameter tuning job	–

Figure 5.18: *Custom job listed under training module of Vertex AI*

Training job will be listed under hyperparameter tuning jobs with the status (click on it).

More details about the status, training logs will also be available, as shown in *Figure 5.19*:



eye_classification_hpo-hyperparameter-tuning-job

Hyperparameter tuning job was completed in

Status	Finished
Hyperparameter tuning job ID	[illegible]
Created	
Start time	
Elapsed time	17 min 13 sec
Region	us-central1
Encryption type	Google-managed key
Dataset	No managed dataset
Study specs	VIEW JSON
Trial job specs	VIEW JSON
Algorithm	Custom training
Objective	Custom
Metric to optimise	accuracy
Goal	Maximise

Hyperparameter tuning trials

Trial ID	accuracy	Training step	n_estimators	max_depth	min_samples_split	min_samples_leaf	max_features	Log
1	[illegible]	1	120	15	10	10	auto	View logs
2	[illegible]	1	100	20	5	5	sqrt	View logs

Figure 5.19: *Job details and log information of the training job*

Follow the corresponding steps:

1. Status of training, elapsed time and all other details are populated.

2. Click on **VIEW HYPERPARAMETER TUNING JOB INPUTS IN JSON**, to view the hyperparameters and other details of the training job.
3. Accuracy of various trails are populated in the section under **Hyperparameter tuning trails**, along with the detailed logs for each trail.

The trained model will be saved in the cloud storage bucket, as shown in *Figure 5.20*:

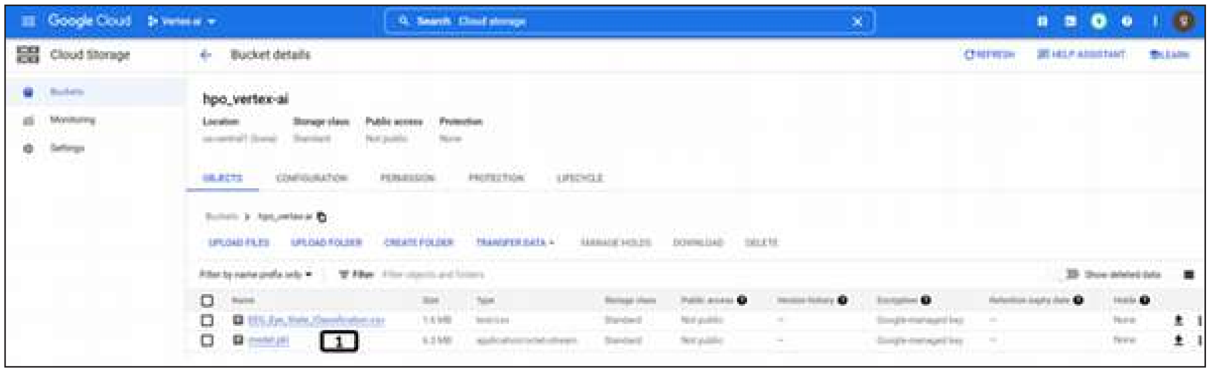


***Figure 5.20**: Model Trained model stored in cloud storage*

1. The trained model is exported as `pkl` file in cloud storage.

Model importing

Often, custom trained models will not be listed in the model registry. Follow the given steps to import model to the model registry.

Step 1: Select model registry

Select the model registry on vertex AI. Refer to *Figure 5.21*:

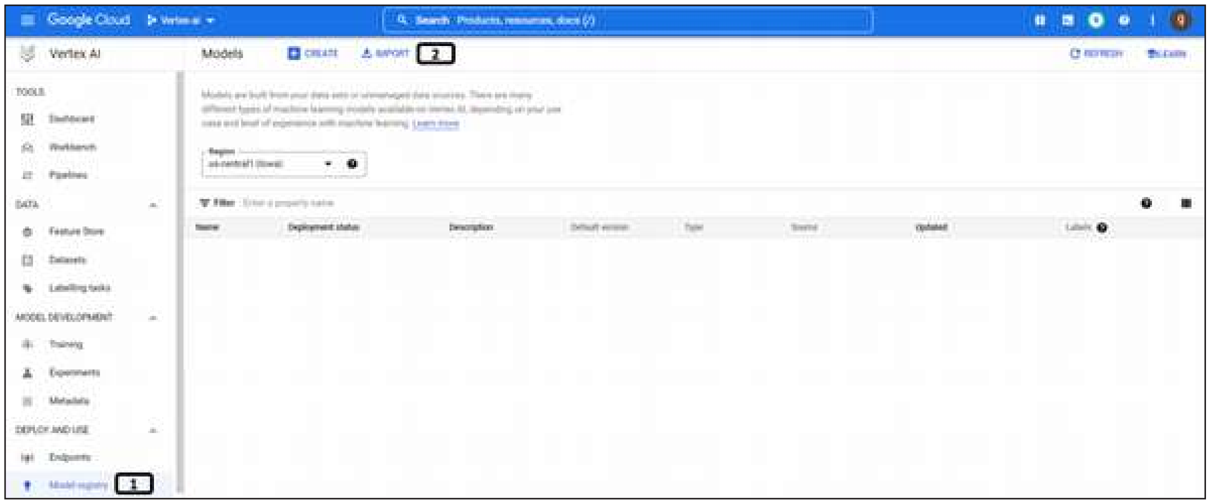


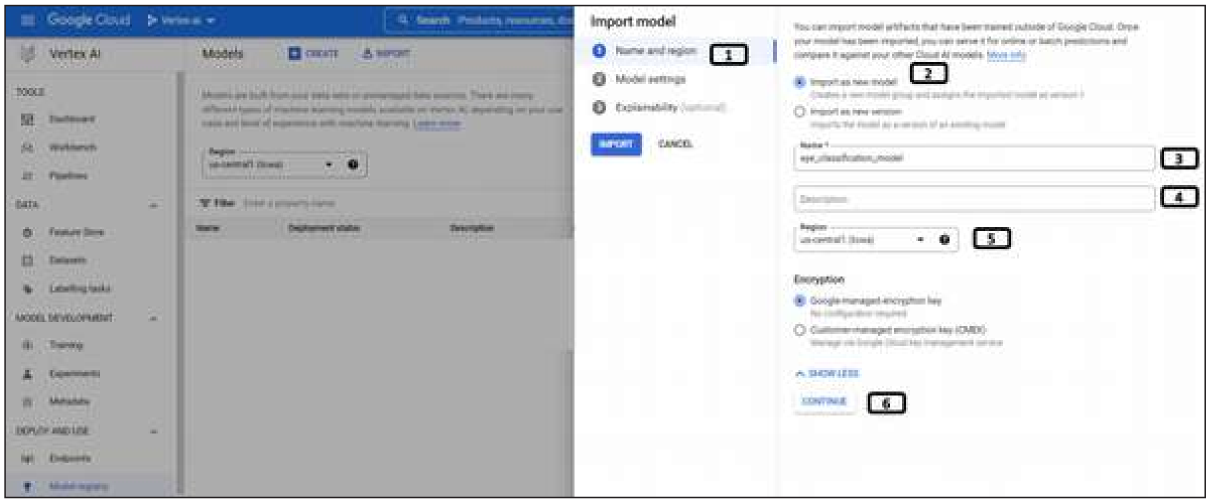
***Figure 5.21**: Model registry of Vertex AI*

Follow the corresponding points:

1. Select **Model registry**.
2. Click **IMPORT**.

Step 2: Selection of name and region for the model

Refer to the *Figure 5.22*:



You can import model artifacts that have been trained outside of Google Cloud. Once your model has been imported, you can serve it for online or batch predictions and compare it against your other Cloud AI models. More info

Import as new model — Creates a new model group and assigns the imported model as version 1

Import as new version — Imports the model as a version of an existing model

Name *: eye_classification_model

Description

Region: us-central1 (Iowa)

Google-managed encryption key — No configuration required

Customer-managed encryption key (CMEK) — Manage via Google Cloud key management service

***Figure 5.22**: Name and region information for Importing trained model*

Follow the corresponding points:

1. Select **Name and region**.
2. Select **Import as new model**.
3. Provide **Name** for the model.
4. Provide **Description** of the model.
5. Select the **Region** to save the model.
6. Click **CONTINUE**.

Step 3: Selection of model settings

Refer to *Figure 5.23*:

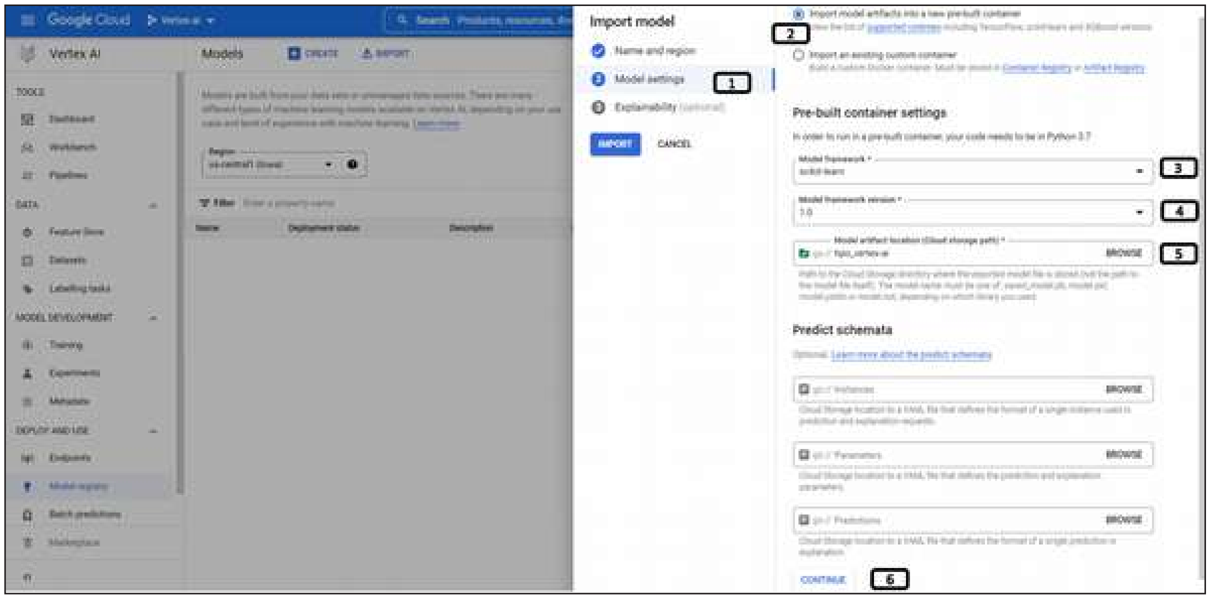


Figure 5.23: Model settings for Importing trained model

Follow the corresponding points to configure the model to import:

1. Select **Model settings**.
2. Select **Import model artifacts into a new pre-built container** (Pre-build container can be used only for models trained using TensorFlow, sklearn and XGboost). If model is trained using any other packages, select the **Import an existing custom container** option.
3. Select the framework on which model is trained.
4. Select the version of the package (sklearn).
5. Select the bucket which contains the exported model.
6. Click on **CONTINUE**.

Step 4: Explain ability (optional)

Refer to the following *Figure 5.24*:

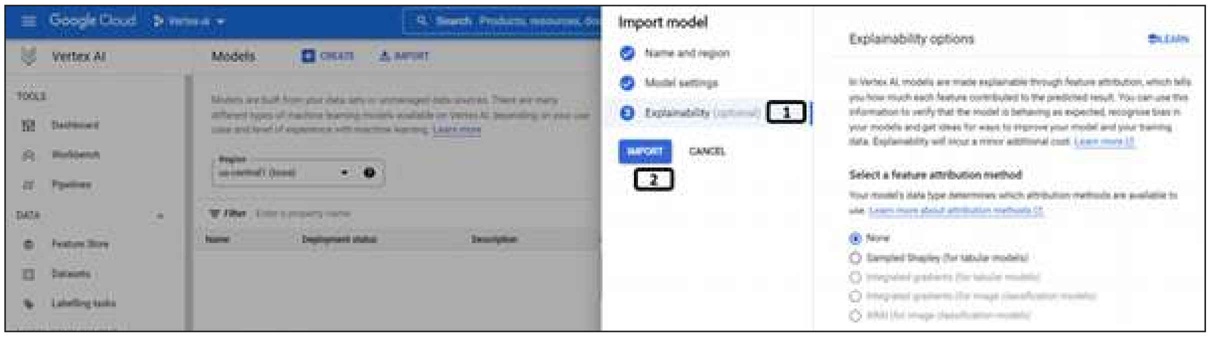

Google Cloud
Vertex AI
Models
CREATE
IMPORT
Import model
Name and region
Model settings
Explainability (optional)
IMPORT
CANCEL
Explainability options
Select a feature attribution method
None
Sampled Shapley (for tabular models)
Integrated gradients (for tabular models)
Integrated gradients (for image classification models)
XRAI (for image classification models)
1
2

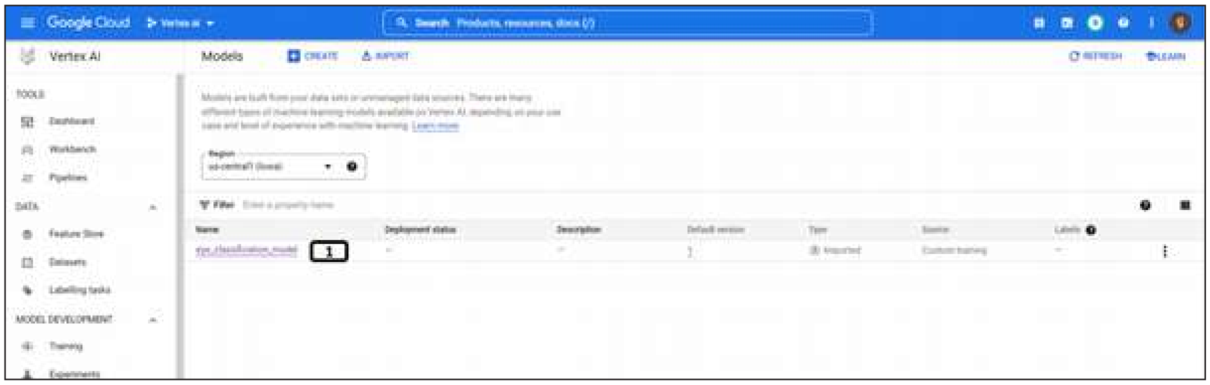
Figure 5.24: Setting explain ability of model

Follow the corresponding points to configure the model to import:

1. Enabling **Explainability** of the model is optional; we will be ignoring it in this chapter.
2. Click **IMPORT**.

Step 5: Model imported to model registry

The Imported model will be listed under model registry, as shown in the *Figure 5.25:*



Name	Deployment status	Description	Default version	Type	Source	Labels
eye_classification_model	–	–	1	Imported	Custom training	–

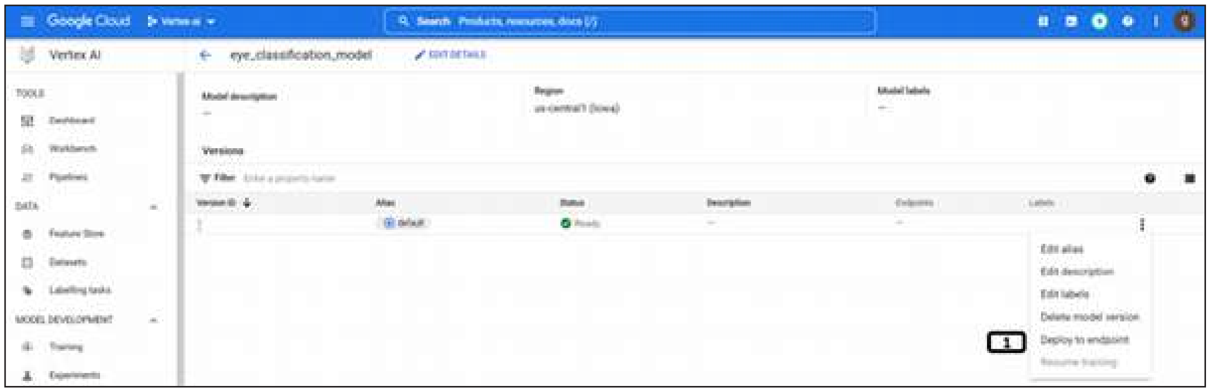
***Figure 5.25**: Model imported successfully and listed under model registry*

Follow the steps that has been described in the next section to deploy the model to endpoint and get online or batch predictions.

Model deployment and predictions

Click on the imported model to see versions of the model (multiple versions of the model can be imported under the same model). In our case version 1 will be the only version of the model and it will be default as shown in the *Figure 5.26* and follow the steps mentioned:

Step 1: Version selection for model deployment



Model description: –
Region: us-central1 (Iowa)
Model labels: –

Version ID	Alias	Status	Description	Endpoints	Labels
1	default	Ready	–	–	

***Figure 5.26**: Selection of model version for model deployment*

Follow the corresponding steps to create endpoint and deploy the model.

1. Click **Deploy to endpoint**.

Step 2: Endpoint creation

Refer the *Figure 5.27* and the corresponding steps to initiate the endpoint creation:

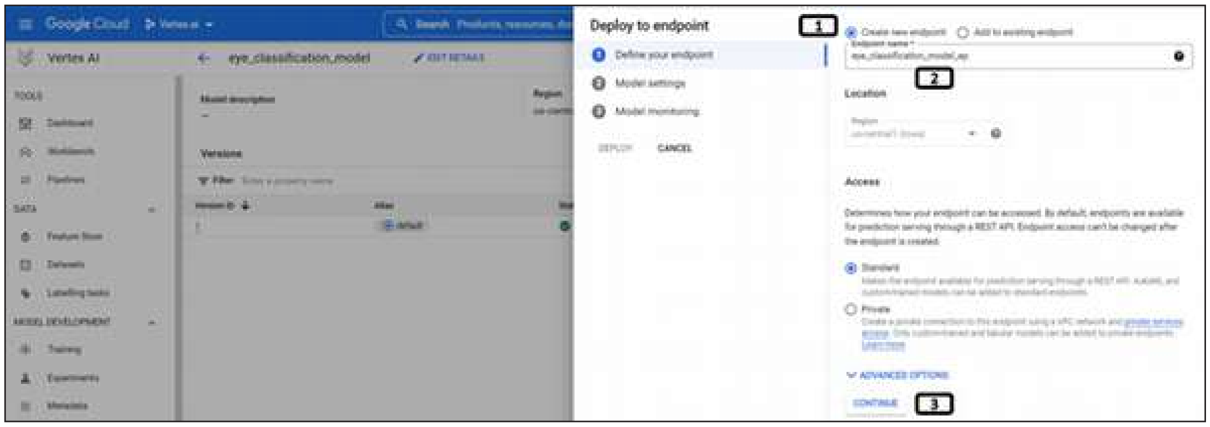


***Figure 5.27**: Endpoint creation*

Follow the corresponding points to create endpoint:

1. Select **Create new endpoint**
2. Provide **Endpoint name**
3. Click on **CONTINUE**.

Step 3: Model settings for deployment

Refer the *Figure 5.28* and the corresponding steps to complete the model settings:

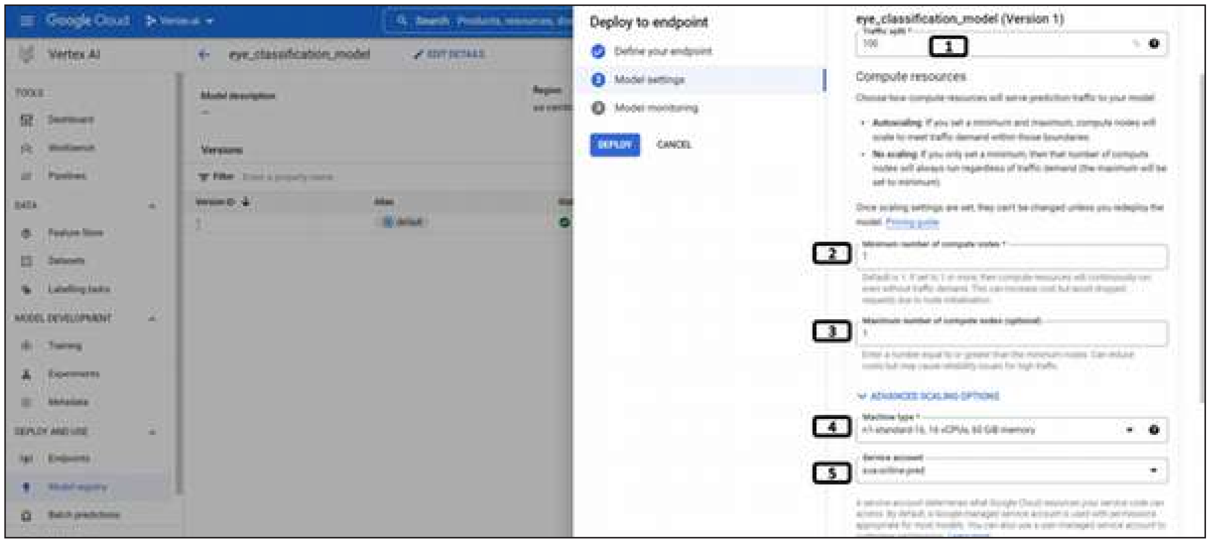


***Figure 5.28**: Model settings for deployment*

Follow the corresponding points to configure the model settings for deployment:

1. Select the **Traffic split** to **100**.
2. Provide **Minimum number of compute nodes** to be **1**.
3. Provide **Maximum number of compute nodes** to be **1**.
4. Select the **Machine type n1-standard-16** (lower configuration machine can also be selected since it is not a large model).
5. Select the **Service account**.
6. Scroll down to click **CONTINUE**.

Step 4: Model monitoring after deployment

Refer the *Figure 5.29* and the corresponding steps to complete the monitoring settings:

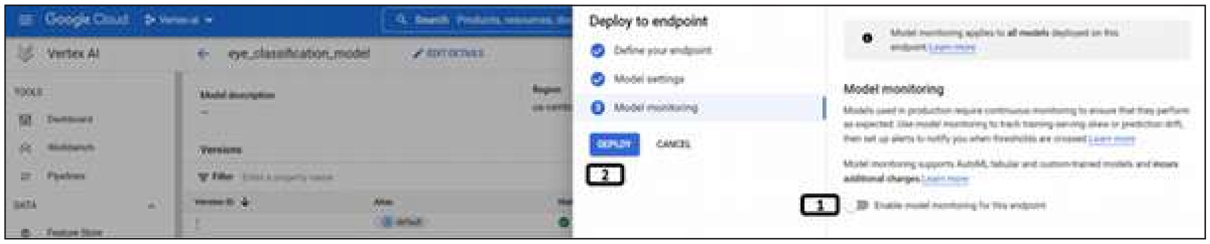


***Figure 5.29**: Model monitoring*

Follow the corresponding points to disable the model monitoring:

1. Disable the **Model monitoring** for the endpoint.
2. Click **DEPLOY**.

Step 5: Status of endpoint created

Refer the *Figure 5.30* to check the status of the model deployed. Status needs to be **Active** to obtain the predictions and endpoint ID needs to be provided in the Python code.

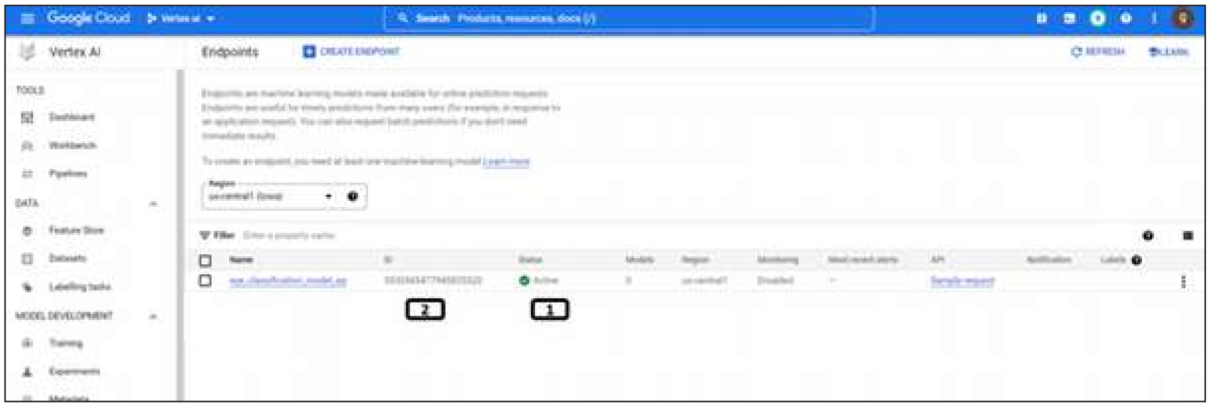


***Figure 5.30**: Model deployed to endpoint successfully*

Check below mentioned points:

1. Status of the endpoint needs to be **Active**.
2. **ID** needs to be provided as input in Python code to obtain the predictions.

Step 6: Online predictions from Python

Model is deployed on to the GCP endpoint. Users can get the predictions from the deployed model using python on local machine. (In our example we are using PyCharm IDE to get the predictions, predictions can be obtained from any IDE or same code can be used on the workbench for the predictions).

Code needs full path of service account, endpoint ID, location of the endpoint and api_endpoint to obtain the predictions. (Below code contains all the details, ensure to change the inputs as required before executing the code). Output of the code is shown in the *Figure 5.31*.

Install **`google-cloud-aiplatform`** Python package through the following command:

```
pip install google-cloud-aiplatform
```

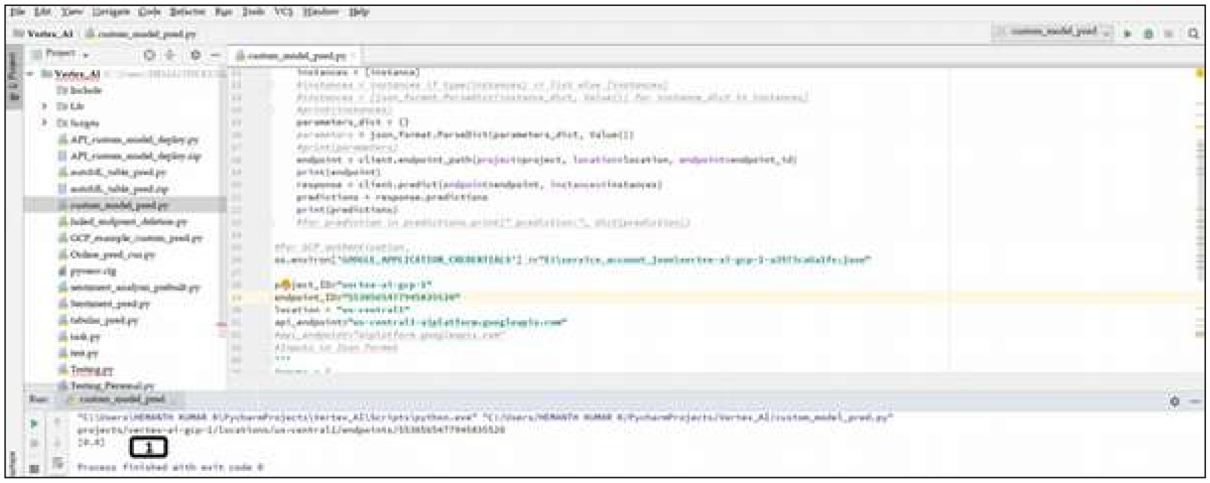
More information on the package can be obtained from below link.

https://pypi.org/project/google-cloud-aiplatform/

Python Code for predictions

```
#Install google-cloud-aiplatform

import os

from google.cloud import aiplatform

from google.protobuf import json_format

from google.protobuf.struct_pb2 import Value

#Working code for hyperparameters


def online_custom_models(project,endpoint_id,instance_dict,location,api_
endpoint):

    client_options = {"api_endpoint": api_endpoint}

    client = aiplatform.gapic.PredictionServiceClient(client_
    options=client_options)
    instance = json_format.ParseDict(instance_dict, Value())

    instances = [instance]

    parameters_dict = {}

    parameters = json_format.ParseDict(parameters_dict, Value())

     endpoint = client.endpoint_path(project=project, location=location, 
endpoint=endpoint_id)

    print(endpoint)

    response = client.predict(endpoint=endpoint, instances=instances)

    predictions = response.predictions

    print(predictions)


#For GCP authentication.

os.environ[‘GOOGLE_APPLICATION_CREDENTIALS’] =r”E:\service_account_json\
vertex-ai-gcp-1-a3973ca6a1fe.json”


project_ID=”vertex-ai-gcp-1”

endpoint_ID=”5530565477945835520”

location = “us-central1”

api_endpoint=”us-central1-aiplatform.googleapis.com”

#Calling the function


inputs=[4287.69,3997.44,4260,4121.03,4333.33,4616.41,4088.72,4638.46, 42
12.31,4226.67,4167.69,4274.36,4597.95,4350.77]

online_custom_models(project_ID,endpoint_ID,inputs,location,api_
endpoint)
```

Output:



```
    instances = [instance]
    [illegible]
    [illegible]
    [illegible]
    parameters_dict = {}
    parameters = json_format.ParseDict(parameters_dict, Value())
    #print(parameters)
    endpoint = client.endpoint_path(project=project, location=location, endpoint=endpoint_id)
    print(endpoint)
    response = client.predict(endpoint=endpoint, instances=instances)
    predictions = response.predictions
    print(predictions)
    [illegible]

[illegible]
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = [illegible]

project_ID="vertex-ai-gcp-1"
endpoint_ID=[illegible]
location = "us-central1"
api_endpoint="us-central1-aiplatform.googleapis.com"
[illegible]
```

```
[illegible]
projects/vertex-ai-gcp-1/locations/us-central1/endpoints/[illegible]
[0.4]

Process finished with exit code 0
```

Figure 5.31: *Prediction of the deployed model*

Submitting training job with Python SDK

In all the exercises that we have covered in the previous chapters, we have mainly worked on graphical user interface of the platform. Let us now see how to submit the job for training using Python code.

Step 1: Create a new python notebook on the Jupyter lab

Follow the given steps to create a new Python notebook file, as shown in the following figure:

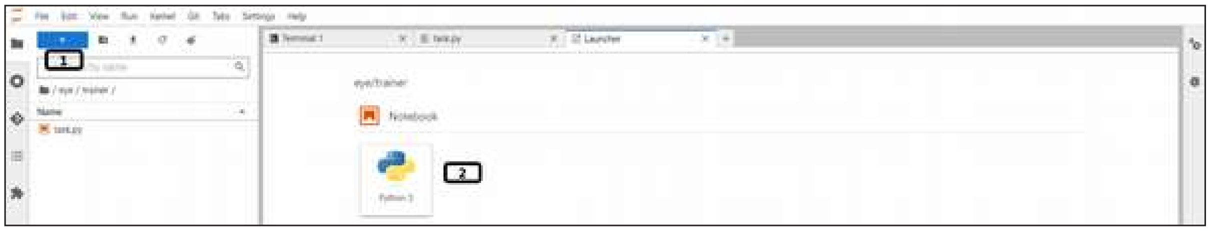


Figure 5.32: *Creation of Python notebook*

Follow the corresponding points:

1. Click on new launcher, that is, the **+** sign.
2. Click on Python notebook to create one.

Step 2: Python code for the training job initiation

Cell 1:

```
!pip3 install google-cloud-aiplatform --upgrade --user
```

```
from google.cloud import aiplatform

from google.cloud.aiplatform import hyperparameter_tuning as hpt
```

Cell 2:

```
worker_pool_details = [{

    “machine_spec”: {

        “machine_type”: “n1-standard-4”

    },

    “replica_count”: 1,

    “container_spec”: {

        “image_url”: “gcr.io/vertex-ai-gcp-1/eye:v1”

    }

}]

metric_spec={‘accuracy’:’maximize’} #Metric to be optimized

# Hyperparameters which needs to be optimized

parameter_spec_list = {“n_estimators”: hpt.DiscreteParameterSpec(val-
ues=[100,125], scale=None),

    “max_depth”: hpt.DiscreteParameterSpec(values=[5,10,15,20],
    scale=None),

    “min_samples_split”: hpt.DiscreteParameterSpec(values=[5,10],
    scale=None),

    “min_samples_leaf”:hpt.DiscreteParameterSpec(values=[5,10],
    scale=None),

    “max_features”:hpt.CategoricalParameterSpec(values=[“auto”,”sqrt”])}
```

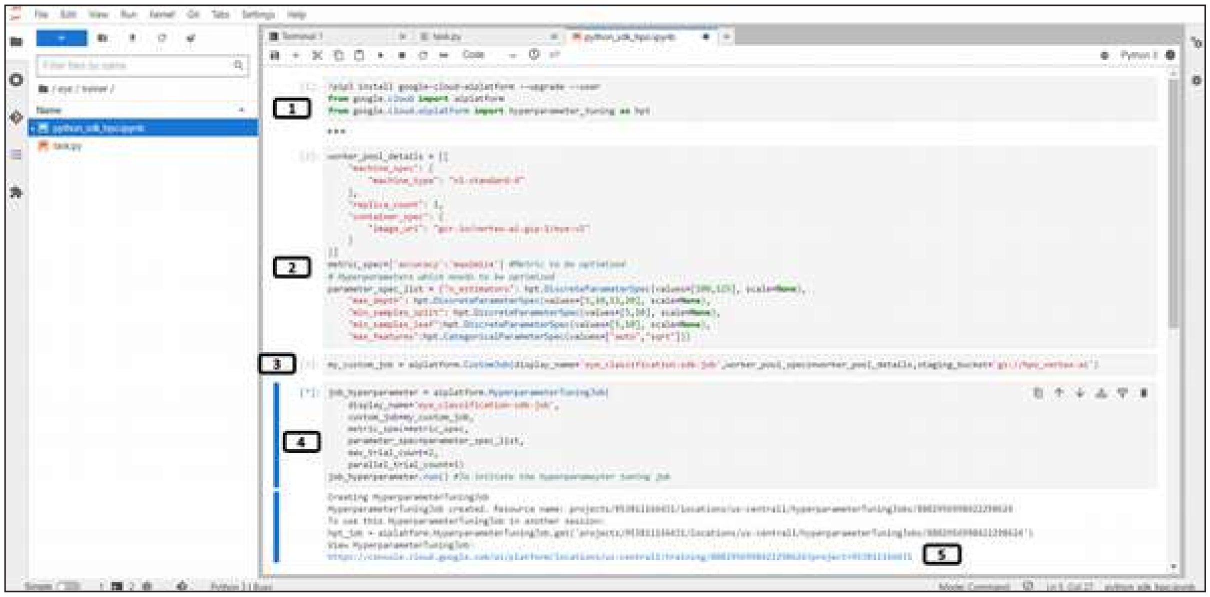
Cell 3:

```
my_custom_job = aiplatform.CustomJob(display_name=’eye_classification-
sdk-job’,worker_pool_specs=worker_pool_details,staging_bucket=’gs://hpo_
vertex-ai’)
```

Cell 4:

```
job_hyperparameter = aiplatform.HyperparameterTuningJob(
    display_name='eye_classification-sdk-job',
    custom_job=my_custom_job,
    metric_spec=metric_spec,
    parameter_spec=parameter_spec_list,
    max_trial_count=2,
    parallel_trial_count=1)
job_hyperparameter.run() #To initiate the hyperparameyter tuning job
```

Enter the code given previously, in different cells, as shown in the *Figure 5.33*:



```
!pip3 install google-cloud-aiplatform --upgrade --user
from google.cloud import aiplatform
from google.cloud.aiplatform import hyperparameter_tuning as hpt

worker_pool_details = [{
    "machine_spec": {
        "machine_type": "n1-standard-4"
    },
    "replica_count": 1,
    "container_spec": {
        "image_uri": "[illegible]"
    }
}]
metric_spec={'accuracy':'maximize'} #Metric to be optimized
# Hyperparameters which needs to be optimized
parameter_spec_list = {"n_estimators": hpt.DiscreteParameterSpec(values=[100,125], scale=None),
    "max_depth": hpt.DiscreteParameterSpec(values=[5,10,15,20], scale=None),
    "min_samples_split": hpt.DiscreteParameterSpec(values=[5,10], scale=None),
    "min_samples_leaf":hpt.DiscreteParameterSpec(values=[5,10], scale=None),
    "max_features":hpt.CategoricalParameterSpec(values=["auto","sqrt"])}

my_custom_job = aiplatform.CustomJob(display_name='eye_classification-sdk-job',worker_pool_specs=worker_pool_details,staging_bucket='gs://hpo_vertex_ai')

job_hyperparameter = aiplatform.HyperparameterTuningJob(
    display_name='eye_classification-sdk-job',
    custom_job=my_custom_job,
    metric_spec=metric_spec,
    parameter_spec=parameter_spec_list,
    max_trial_count=2,
    parallel_trial_count=1)
job_hyperparameter.run() #To initiate the hyperparameyter tuning job

Creating HyperparameterTuningJob
HyperparameterTuningJob created. Resource name: projects/[illegible]/locations/us-central1/hyperparameterTuningJobs/[illegible]
To use this HyperparameterTuningJob in another session:
hpt_job = aiplatform.HyperparameterTuningJob.get('projects/[illegible]/locations/us-central1/hyperparameterTuningJobs/[illegible]')
View HyperparameterTuningJob:
https://console.cloud.google.com/ai/platform/locations/us-central1/training/[illegible]?project=[illegible]
```

***Figure 5.33**: Python code for submitting training job*

Refer to the corresponding points for a better understanding of what each code does in each cell:

1. Installing and importing the required packages. Even though we are using notebook on GCP, **`google-cloud-ai`** platform will not be pre-installed.
2. Defining the hardware configuration, metrics to optimize and hyperparameter list.
3. Defining a custom job with display name and other details (if custom job needs to be initiated without hyperparameter optimization then add **`my_custom_job.run()`** to initiate the custom model training).
4. Adding hyperparameter step to the custom job training defined in the previous step.

5. Click on the link. It will take to the training page, as shown in *Figure 5.34*:

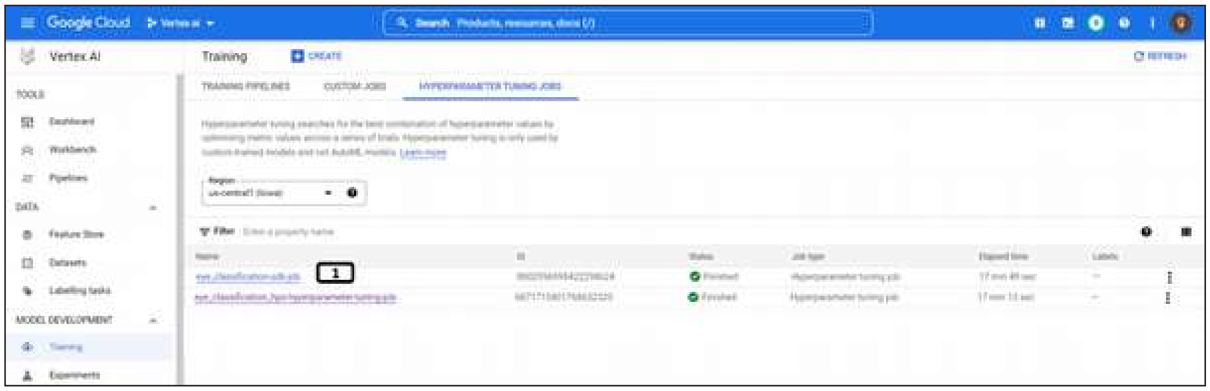


***Figure 5.34**: Training successfully completed*

Deletion of resources

All the compute resources are expensive. Follow the steps mentioned in chapter 2 under Model un-deployment and deleting end point to un-deploy model and endpoint deletions. Follow the steps mentioned in chapter 2 under Model deletion to delete model from registry. Follow the steps mentioned in chapter 4 under Deletion of resources to delete the workbench instance.

Conclusion

In this chapter, we learned about the custom model in detail, along with the hyper parameter tuning. We also covered how to submit the custom model for training using both GUI and SDK methods, and how to import the train models to model registry.

In the next chapter, we will start working on the pipelines of GCP.

Questions

1. How to pass the parameters for tuning in custom job?
2. During the initiation of training job including hyperparameter tuning from python SDK, why is defining custom training job needed?
3. If we are training model using PyTorch package, can we use pre-built containers for training?

Chapter 6
Introduction to Pipelines and Kubeflow

Introduction

In the previous chapters, we worked on workbench of Vertex AI to train custom models including hyperparameter tuning using Vizer. In this chapter, we will get started with the pipelines of Vertex AI. We will understand what pipeline is, what is Kubeflow, what are the components of the pipeline, and how to configure and run Vertex AI pipeline using Kubeflow.

Structure

In this chapter, we will discuss the following topics:

- What is machine learning pipeline
- Benefits of machine learning pipelines
- Introduction to Kubeflow
- Components of Kubeflow
- Tasks of Kubeflow
- Data for model training
- API enablement and additional permissions

- Pipeline code walk through
- Pipeline of vertex AI
- Deletion of resources

Objectives

By the end of this chapter, you will have a good understanding of Machine Learning (ML) pipelines, along with Kubeflow and its components. The readers will also learn how to construct pipelines using Kubeflow on Vertex AI.

What is machine learning pipeline

A machine learning pipeline is a comprehensive framework for managing input and output from a ML model (or set of multiple models). There are inputs of raw data, features, and outputs, as well as the machine learning model and its parameters, and predictions.

Another way of understanding ML pipelines is the process of breaking down large machine learning operations into smaller, reusable modules that may be used in a pipeline to produce models is another sort of ML pipeline. By eliminating unnecessary steps, an ML pipeline speeds up and simplifies the model-building process.

Vertex AI pipelines

Automating, monitoring, and governing your ML systems is made easier with Vertex AI pipelines, which orchestrates the ML workflow in a serverless fashion and stores the artifacts of that process in Vertex ML Metadata. By keeping track of your ML workflow's outputs in Vertex ML Metadata, users can trace back the steps that went into making each artifact, such as an ML model's training data, hyperparameters, and code.

Pipelines are used for applying methods from MLOps to streamline and keep tabs on your most-often-performed tasks. Try out various configurations of your machine learning process by playing with the hyperparameters, training steps, iterations, and so on. Re-use pipeline component or entire pipeline to train a new model. Pipelines that were built using Kubeflow SDK or TensorFlow extended can be executed using Vertex AI pipelines.

Benefits of machine learning pipelines

There are various benefits of machine learning pipelines:

Execution

The pipeline gives users the ability to program many phases to carry out in parallel in a dependable and unsupervised manner. This indicates that users are free to concentrate on other things concurrently while the process of data modelling and preparation is being carried out. Since each stage of the ML pipeline process runs as an independent component, the program can run more rapidly and provide an efficient and high-quality result.

Model versioning and tracking

Instead of manually keeping track of the data and outputs for each iteration, we may make use of a pipeline to clearly identify and version the data sources, as well as the inputs and outputs.

Troubleshooting

The pipeline approach divides each job into its components (such as different components for data pre-processing and model training). As a result, it is much simpler to debug the whole piece of code and pinpoint where the problems lie inside a certain component.

Resource utilization

The usage of several pipelines that are successfully coordinated across physically separate but similarly equipped computer systems and storage facilities is now possible. It enables effective use of hardware by distributing pipeline stages over several machines (CPUs, GPUs, Data Science VMs, and so on) depending on the computation complexity of each step.

Introduction to Kubeflow

The objective of the Kubeflow project is to simplify the deployment of ML workflows on Kubernetes while also enabling them to be portable and scalable. The aim of Kubeflow is to simplify the process of deploying open-source ML systems to a wide variety of infrastructures. Any environment in which Kubernetes is operational should also support the use of Kubeflow. The Kubeflow Pipelines SDK gives users access to a collection of Python packages that can be used to define and execute the ML workflows.

Components of Kubeflow

In **Kubeflow Pipelines** (**KFP**), the most fundamental building block of any piece of processing mechanism is known as a component. A component is a named template that specifies how to operate a container by making use of an image, a command, and several parameters. Because a component can also contain inputs and outputs, it can be thought of as a computational template in the same way that a function is. The dynamic data, known as component inputs, are utilized in either the container instructions or the arguments. Outputs from components might be artifacts of ML that are serializable.

There are three different ways to write components with these three characteristics:

- **Lightweight Python function-based components**: Lightweight Python function-based components are easiest to author (also known as a lightweight component). Lightweight components create a component that runs a single Python function in a container during execution.
- **Containerized Python components**: Containerized Python components let users bundle Python function-based components in containers. Containerized Python components allow authors to utilize extra source code outside the component's Python function description, including several files. This is the ideal way for publishing Python components that require more source code than can be contained in a single function or for reusing source code in several components.
- **Custom container components**: You may provide a container to execute as your component by using custom container components. Using an image, command, and args, you may specify a container using the ContainerSpec object of KFP.

Tasks of Kubeflow

An input-driven job executes a component, called Task. It is a component template instantiation. A pipeline consists of jobs that may or may not share data. One pipeline component can instantiate numerous jobs. Using loops, conditions, and exit handlers, tasks may be generated and run dynamically. Because tasks represent component runtime execution, you may configure environment variables, hardware resource requirements, and other task-level parameters.

There are three types of task dependencies:

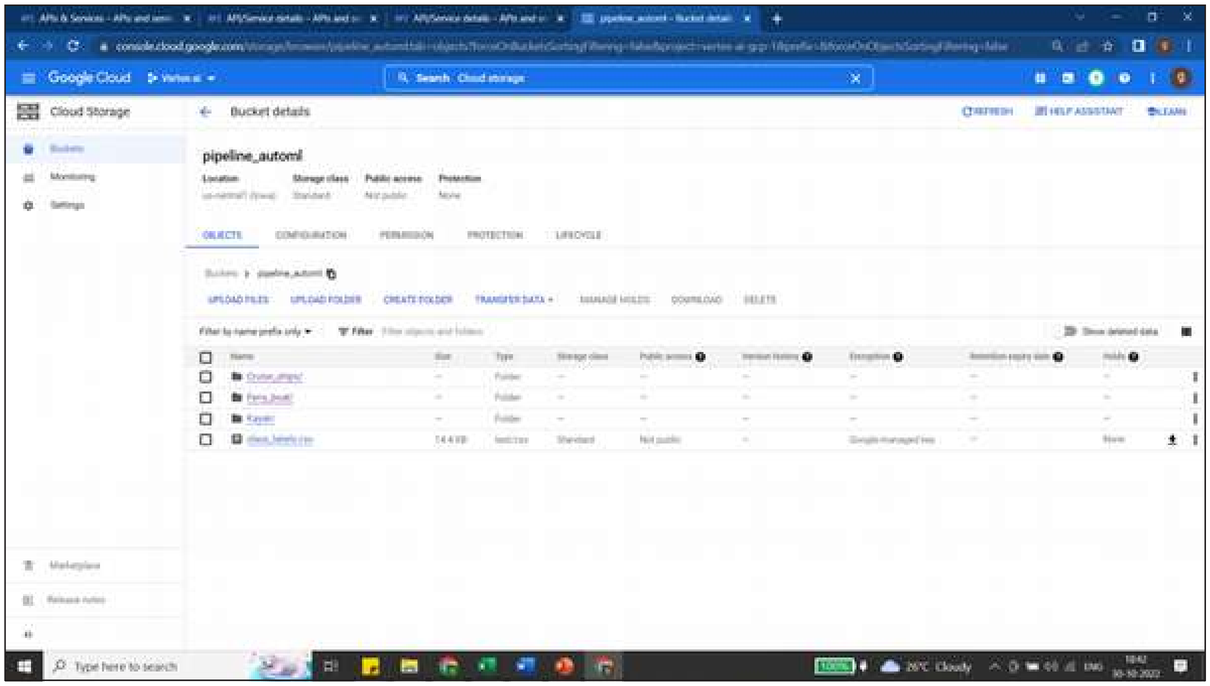
- **Independent tasks**: Depending on the circumstances, tasks may or may not be interdependent. If neither job depends on the results of the other, and vice versa, then we say that the two tasks are completely decoupled from one another. At pipeline runtime, two jobs can run simultaneously if they are independent.
- **Implicitly dependent tasks**: The creation of an implicit dependency between tasks occurs when the outcome of one job is the input to another. In this situation, the upper job will run before the latter, and the latter will get its results.
- **Explicitly dependent tasks**: It can be useful to control the order in which two activities run without sharing data between them. If this is the case, the subsequent job can be invoked by calling it. (.after() needs to be mentioned to create the sequence of execution between the tasks).

Data for model training

For this exercise, data is downloaded from Kaggle (link is provided below) and the dataset is listed under CC0:Public domain licenses (same dataset which we had used for AutoML for image data). For our exercise, we are considering images belonging to Cruise ships, ferry boat and Kayak (50 images are chosen randomly from each category).

https://www.kaggle.com/datasets/imsparsh/dockship-boat-type-classification

pipeline_automl bucket is created under **us-centra1** (single region) and three folders containing images belonging to three categories are uploaded. CSV file is created and uploaded to same bucket containing the information regarding the full path of the image files, and the category they belong to as shown in *Figure 6.1*: (Refer to Vertex AI AutoML for Image data section of *Chapter 6, AutoML Image, text and pre-built models to create the csv file*):


pipeline_automl

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

Name	Size	Type	Storage class	Public access	Version history	Encryption	Retention expiry date	Holds
Cruise_ships/	—	Folder	—	—	—	—	—	—
Ferry_boat/	—	Folder	—	—	—	—	—	—
Kayak/	—	Folder	—	—	—	—	—	—
class_labels.csv	14.4 KB	text/csv	Standard	Not public	—	Google-managed key	—	None

***Figure 6.1**: Data and csv uploaded to cloud storage bucket*

In this chapter, we will create pipeline on Vertex AI component of GCP using Kubeflow SDK. In the pipeline we will use the AutoML component of GCP for model training.

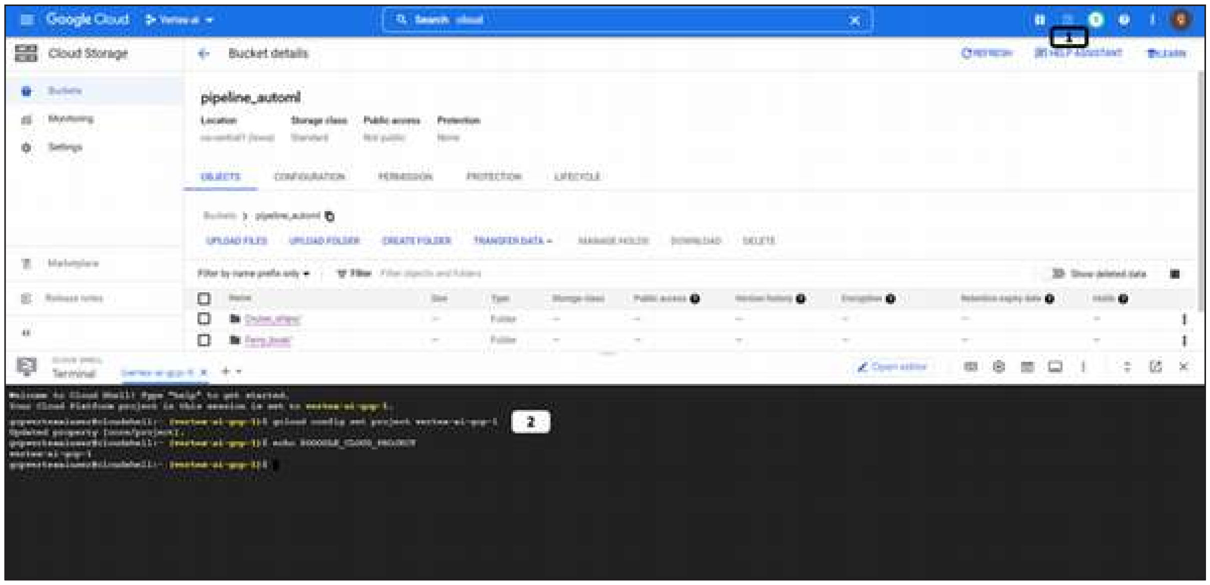
API enablement

We are enabling the APIs as and when it is required throughout various chapters. To work with vertex AI pipelines, we need to enable APIs in addition to the compute engine, container registry, aiplatform (which we have already enabled in previous chapters) like cloud functions, cloud build.

In the previous chapters we enabled the APIs through the user interface under APIs and services component of GCP. Let us see how we can enable multiple APIs in one go using cloud shell. The steps are as follows:

Step 1: Set the project

Follow the steps mentioned in *Figure 6.2* to activate the cloud shell and set the project:



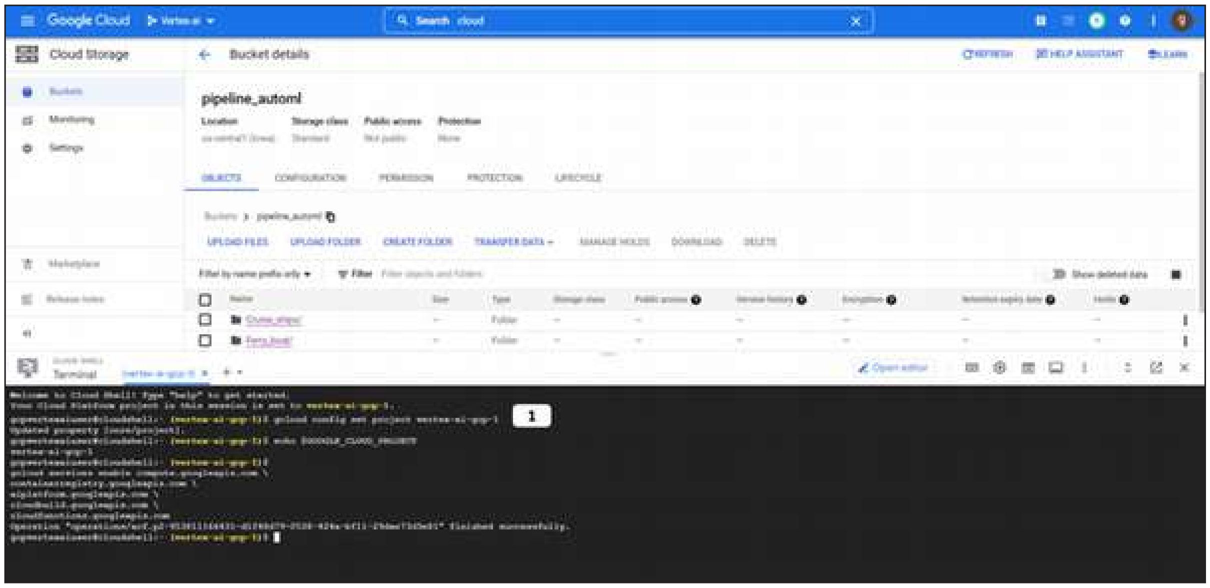
```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to vertex-ai-gcp-1.
[illegible]@cloudshell:~ (vertex-ai-gcp-1)$ gcloud config set project vertex-ai-gcp-1
Updated property [core/project].
[illegible]@cloudshell:~ (vertex-ai-gcp-1)$ echo $GOOGLE_CLOUD_PROJECT
vertex-ai-gcp-1
[illegible]@cloudshell:~ (vertex-ai-gcp-1)$
```

Figure 6.2: Activation of cloud shell and setting the project

1. Click on activate cloud shell.
2. Type the below commands:
   a. To set the project - **`gcloud config set project vertex-ai-gcp-1`**
   b. To check if the needed project is selected - **`echo $GOOGLE_CLOUD_PROJECT`**

Step 2: Enabling APIs

Follow the steps mentioned in *Figure 6.3* to enable multiple APIs of GCP through cloud shell:



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to vertex-ai-gcp-1.
[illegible]@cloudshell:~ (vertex-ai-gcp-1)$ gcloud config set project vertex-ai-gcp-1
Updated property [core/project].
[illegible]@cloudshell:~ (vertex-ai-gcp-1)$ echo $GOOGLE_CLOUD_PROJECT
vertex-ai-gcp-1
[illegible]@cloudshell:~ (vertex-ai-gcp-1)$
gcloud services enable compute.googleapis.com \
containerregistry.googleapis.com \
aiplatform.googleapis.com \
cloudbuild.googleapis.com \
cloudfunctions.googleapis.com
Operation "operations/acf.p2-[illegible]" finished successfully.
[illegible]@cloudshell:~ (vertex-ai-gcp-1)$
```

Figure 6.3: API enablement

1. Type the following commands to enable the APIs:

```
gcloud services enable compute.googleapis.com \
containerregistry.googleapis.com \
 aiplatform.googleapis.com \
cloudbuild.googleapis.com \
cloudfunctions.googleapis.com
```

Status message will be shown once the APIs are enabled.

Additional permission for compute engine

In addition to the API enablement, we also need to grant additional permission to the service account associated with the compute engine of GCP as a storage admin to access the source files from the cloud storage (By default notebooks will have access for the cloud storage but in case of pipeline, code to fetch data, model training will not be executed on the notebook and hence the additional permission is needed. Notebook is used only to define the pipeline components and to execute the pipeline). Follow these steps to grant the permission:

Step 1: Open IAM and admin section

Follow the steps mentioned in *Figure 6.4* to add roles to the service account associated with compute engine of GCP:

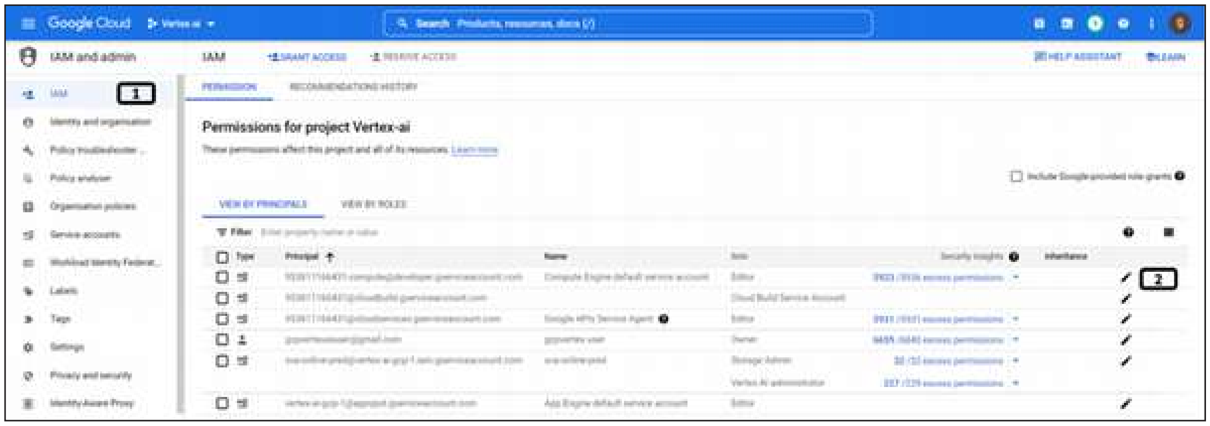


***Figure 6.4**: IAM of GCP*

1. Click **IAM**.
2. Click Edit principal button.

Step 2: Additional role

Follow the steps mentioned in *Figure 6.5, 6.6* and *6.7* to edit the roles of the service account of compute engine:

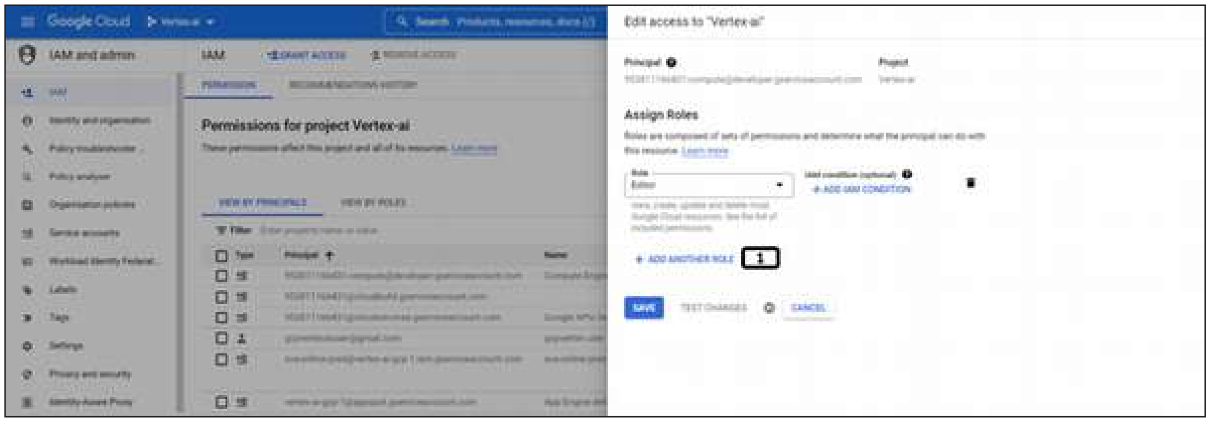


***Figure 6.5**: Edit roles of the service account*

1. Click **ADD ANOTHER ROLE**:

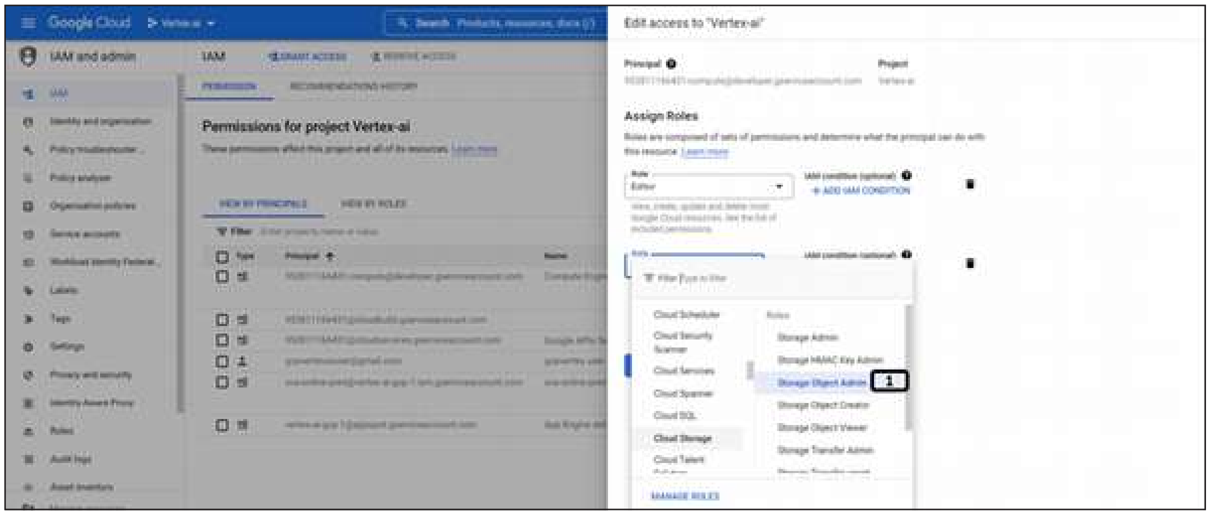


***Figure 6.6**: Adding storage object admin role*

2. Select **Storage Object Admin** role under the **Cloud Storage** filter.
3. Click **Save**. A new role will be added to the service account as shown in *Figure 6.7*:

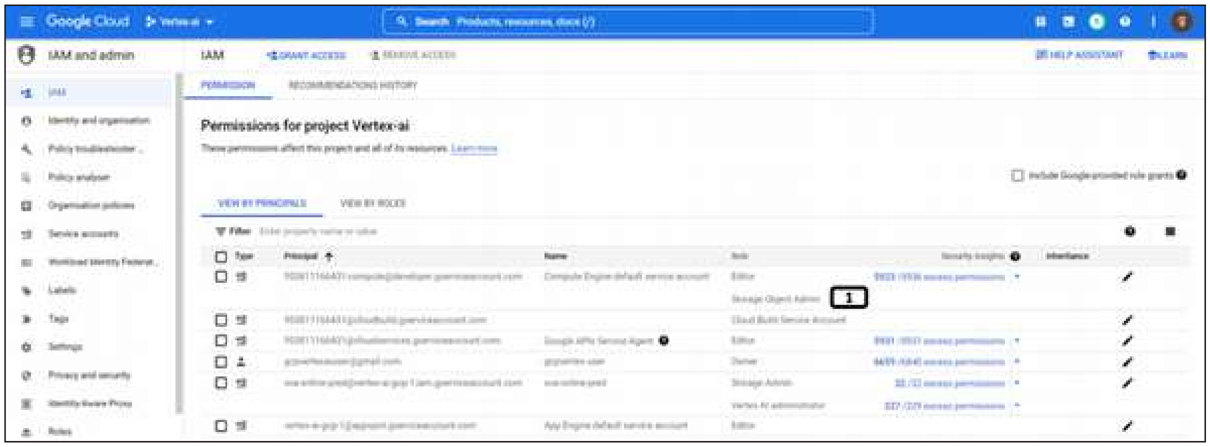


Figure 6.7: *New role added to service account is visible in the IAM*

1. New role will be visible for the service account associated with the compute engine.

Pipeline code walk through

Workbench needs to be created to run the pipeline code. Follow the steps followed in *Chapter 4, Vertex AI Workbench and custom model training* under the section *Vertex AI Workbench creation for creation of the workbench* (choose Python3 machine):

Step 1: Creating Python notebook file

Once the workbench is created, open the Jupyterlab and follow the steps mentioned in *Figure 6.8* to create Python notebook file:

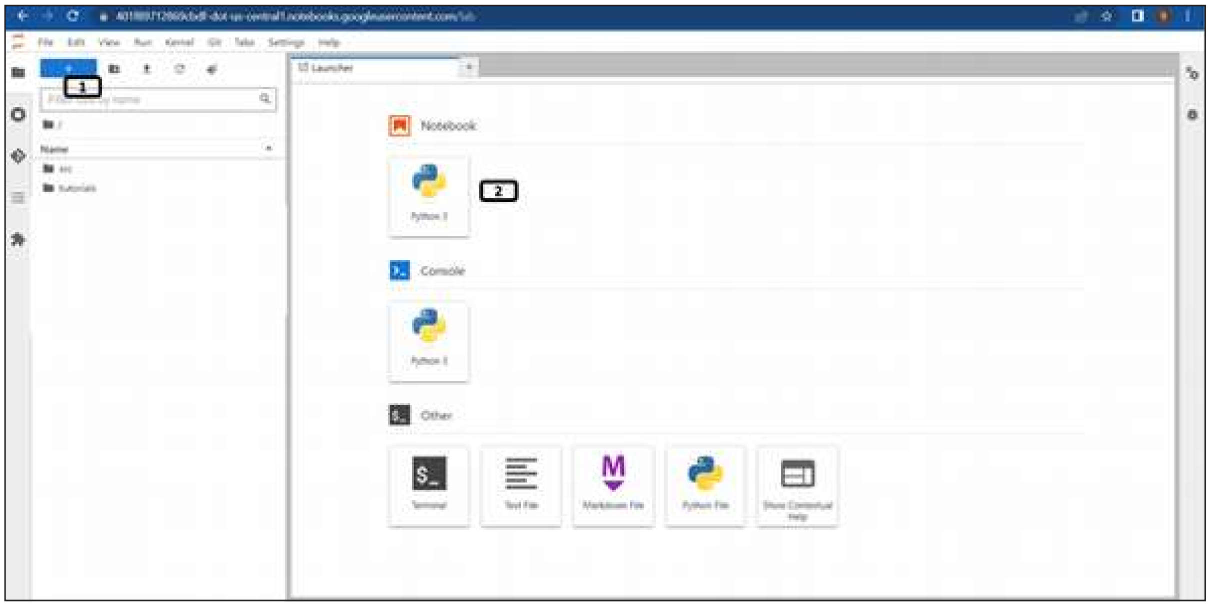


Figure 6.8: *New launcher window*

1. Click New launcher.
2. Double click on the **Python 3** notebook file to create one.

Step 2 onwards, run the codes given in separate cells.

Step 2: Package installation

Run the following commands to install the Kubeflow, google cloud pipeline and google cloud aiplatform package. (It will take few minutes to install the packages):

```
USER_FLAG = "–user"

!pip3 install {USER_FLAG} google-cloud-aiplatform==1.18.0 –upgrade

!pip3 install {USER_FLAG} kfp==1.8.10 google-cloud-pipeline-components==1.0
```

Step 3: Kernel restart

Type the following commands in the next cell, to restart the kernel. (Users can restart kernel from the GUI as well):

```
import os

if not os.getenv("IS_TESTING"):

    import IPython

    app = IPython.Application.instance()

    app.kernel.do_shutdown(True)
```

Step 4: Importing packages

Run the following mentioned line of codes in a new cell to import the required packages:

```
import kfp

from kfp.v2 import compiler, dsl

from kfp.v2.dsl import component, pipeline, Artifact, ClassificationMetrics, Input, Output, Model, Metrics

from typing import NamedTuple

from google_cloud_pipeline_components import aiplatform as gcc_aip

from google.cloud import aiplatform
```

Using the vertex AI pipeline, we are creating a dataset for the AutoML image from the data available in the cloud storage, training the AutoML image classification model, and evaluating the model to the thresholds. If the model performance is above the threshold, then the endpoint will be created and trained model will be deployed to the endpoint. The following figure has the flowchart of the steps carried out in the pipeline (different components of the pipeline). Except the model evaluation component, all other components are available in the **`google_cloud_pipeline_components`** package:

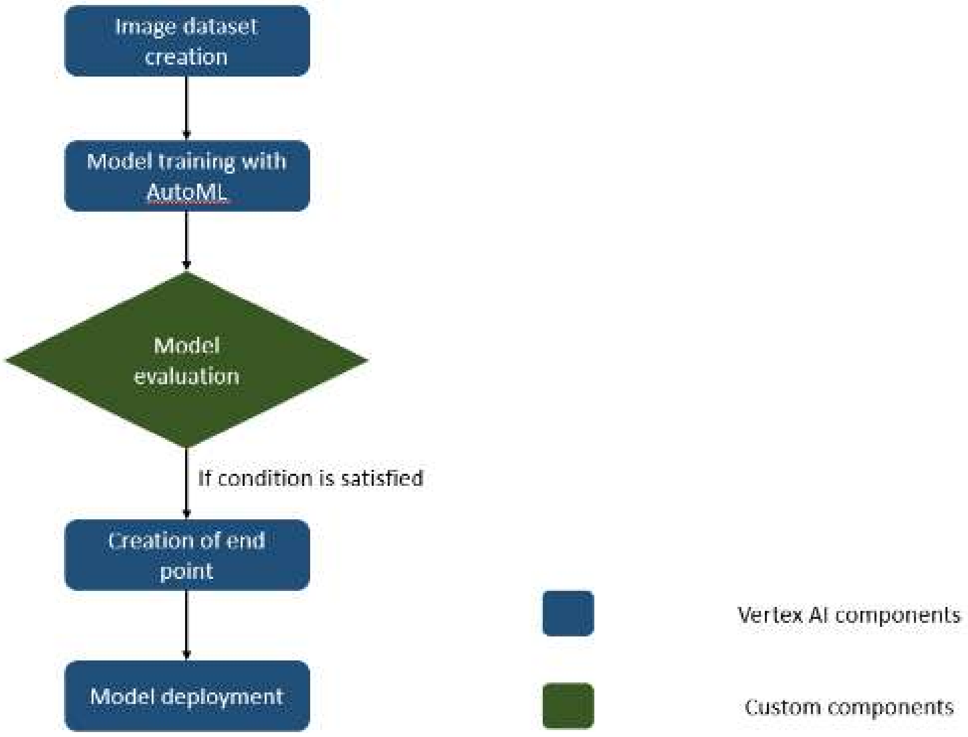


***Figure 6.9**: Components of pipeline*

Step 4: Bucket for storing artifacts of the pipeline

Run the following mentioned line of codes to set the location to store the artifacts. Bucket will be created during the pipeline run:

```
PROJECT_ID = “vertex-ai-gcp-1”

bucket_name_arti=”gs://” + PROJECT_ID + “-pipeline-automl-artifacts”

PATH=%env PATH

%env PATH={PATH}:/home/jupyter/.local/bin

REGION=”us-central1”
```

```
pipeline_folder = f”{bucket_name_arti}/pipeline_automl/”

print(pipeline_folder)
```

Step 5: Model evaluation component of the pipeline

Run the following mentioned codes to define the custom component for model evaluation and threshold comparison which will be included in the pipeline. Important points about the following code are:

- Decorator component is used to define this function as a component of the pipeline.
- Input for the component will be the trained model (and the artifacts) which will be collected from **`Input[Artifacts]`**.
- Function **`fetch_eval_info`** will fetch the evaluation data from the trained model, evaluation data will be parsed and passed to the **`metrics_log_check`** function to check if the model performance is above the threshold and return the value to be **`true`** (if the model performance is better than the threshold) or false (if other wise):

```
@component(base_image=”gcr.io/deeplearning-platform-release/tf2-
cpu.2-5:latest”,output_component_file=”model_eval_component.yaml”,
    packages_to_install=[“google-cloud-aiplatform”])
def image_classification_model_eval_metrics(
    project: str,
    location: str,
    api_endpoint: str,
    thresholds_dict_str: str,
    model: Input[Artifact],
    metrics: Output[Metrics],
    metrics_classification: Output[ClassificationMetrics],
) -> NamedTuple(“Outputs”, [(“dep_decision”, str)]):

    import json
    import logging
    from google.cloud import aiplatform as aip

    #  fetch_eval_info function fetches the evaluation
```

```
information from the trained model.
def fetch_eval_info(client_name, model_name):
    #Refer the repository for code block of the function

    return (model_eval.name,metrics_list_value,metrics_list_
    string)

def metrics_log_check(metrics_list_value, metrics_
classification,thresholds_dict_str):
    #Refer the repository for code block of the function

    return True

logging.getLogger().setLevel(logging.INFO)
aip.init(project=project)
# extract the model resource name from the input Model Artifact
model_resource_path = model.metadata[“resourceName”]
logging.info(“model path: %s”, model_resource_path)

client_options = {“api_endpoint”: api_endpoint}
# Initialize client that will be used to create and send
requests.
client = aip.gapic.ModelServiceClient(client_options=client_
options)
#To fetch the evaluation information for the specific models
eval_name, metrics_list_value, metrics_str_list = fetch_eval_
info(client, model_resource_path)
logging.info(“got evaluation name: %s”, eval_name)
logging.info(“got metrics list: %s”, metrics_list_value)
deploy = metrics_log_check(metrics_list_value, metrics_
classification,thresholds_dict_str)
if deploy: _decision = “true”
else: _decision = “false”
```

```
        logging.info("deployment decision is %s", dep_decision)
        return (dep_decision,)
```

Step 6: Pipeline construction

Run the following codes to define the pipeline with the custom components and the other GCP components. Important points about the following code are:

- Decorator **`kfp.dsl.pipeline`** is used to define this function as a pipeline.
- Input for the component will be the trained model (and the artifacts) which will be collected from **`Input[Artifacts]`**.
- Function **`fetch_eval_info`** will fetch the evaluation data from the trained model artifacts, which will be the passed to the **`metrics_log_check`** function to check if the model performance is above the threshold and retunes the value to be **`true`** (if the model performance is better than the threshold) or false (if other wise):

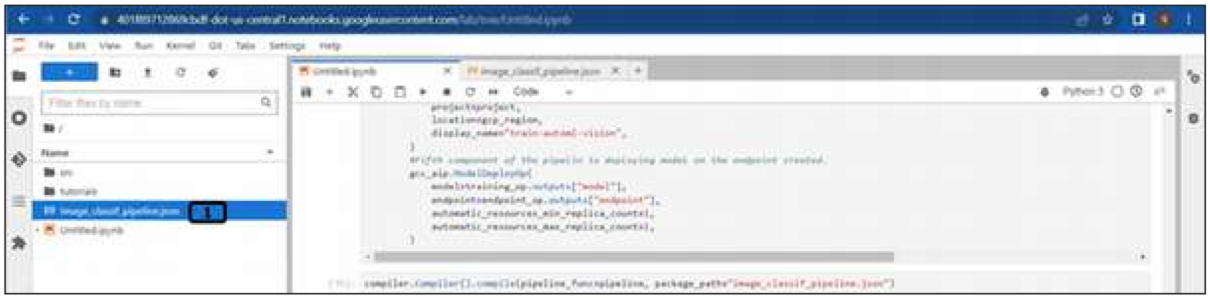
```
DISPLAY_NAME = 'image_boat_classification'
@kfp.dsl.pipeline(name="image-classification",pipeline_
root=pipeline_folder)
def pipeline(
    gcs_source: str = "gs://pipeline_automl/class_labels.csv",
    display_name: str = DISPLAY_NAME,
    project: str = PROJECT_ID,
    gcp_region: str = "us-central1",
    api_endpoint: str = "us-central1-aiplatform.googleapis.com",
    thresholds_dict_str: str = '{"auPrc": 0.60}',
):
    #First component
    dataset_create_op = gcc_aip.
    ImageDatasetCreateOp(project=project, display_name=display_
    name, gcs_source=gcs_source,import_schema_uri=aiplatform.
    schema.dataset.ioformat.image.single_label_classification)


    #Second component
    training_op = gcc_aip.AutoMLImageTrainingJobRunOp(
        project=project,
        display_name=display_name,
        prediction_type=”classification”,
        budget_milli_node_hours=8000,
        dataset=dataset_create_op.outputs[“dataset”],
    )
    #Third component
    model_eval_task = image_classification_model_eval_metrics(
        project,
        gcp_region,
        api_endpoint,
        thresholds_dict_str,
        training_op.outputs[“model”],
    )



    with dsl.Condition(
        model_eval_task.outputs[“dep_decision”] == “true”,
        name=”deploy_decision”,
    ):
        #Fourth component is end point creation only if the
        condition is met
        endpoint_op = gcc_aip.EndpointCreateOp(
            project=project,
            location=gcp_region,
            display_name=”train-automl-vision”,
        )
        #Fifth component of the pipeline is deploying model on
        the endpoint created.
        gcc_aip.ModelDeployOp(
            model=training_op.outputs[“model”],
            endpoint=endpoint_op.outputs[“endpoint”],
            automatic_resources_min_replica_count=1,
            automatic_resources_max_replica_count=1)
```

Step 7: Compile using kubeflow

Run the following codes to compile the pipeline code. The correct data type usage in pipelines is verified by the Kubeflow Pipelines SDK v2 compiler, as is the avoidance of inputs from parameters being used as outputs from artifacts and vice versa:

```
compiler.Compiler().compile(pipeline_func=pipeline, package_path="image_
classif_pipeline.json")
```

Once the code is executed, JSON file named as **`image_classif_pipeline`** will be created as shown in *Figure 6.10*:

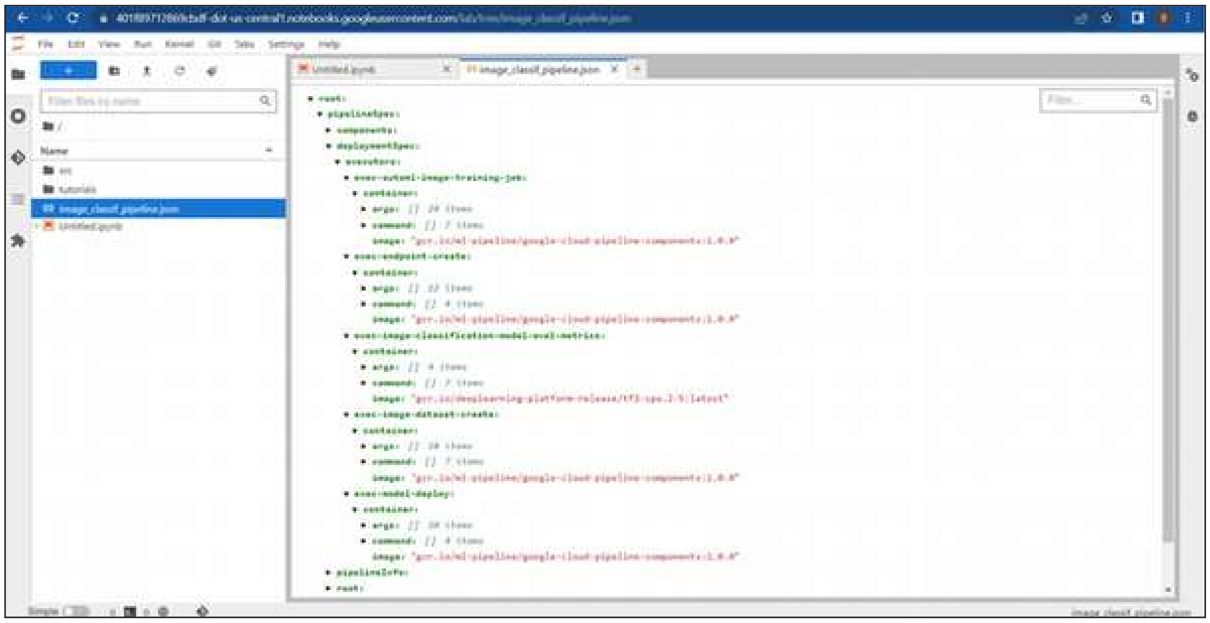


```
        project=project,
        location=gcp_region,
        display_name="train-automl-vision",
    )
    #final component of the pipeline to deploying model on the endpoint created.
    gcc_aip.ModelDeployOp(
        model=training_op.outputs["model"],
        endpoint=endpoint_op.outputs["endpoint"],
        automatic_resources_min_replica_count=1,
        automatic_resources_max_replica_count=1,
    )
```

```
compiler.Compiler().compile(pipeline_func=pipeline, package_path="image_classif_pipeline.json")
```

***Figure 6.10**: JSON file created for the custom model evaluation component*

1. Open the JSON file:



```
▾ root:
  ▾ pipelineSpec:
    ▸ components:
    ▾ deploymentSpec:
      ▾ executors:
        ▾ exec-automl-image-training-job:
          ▾ container:
            ▸ args: [] 20 items
            ▸ command: [] 7 items
              image: "gcr.io/ml-pipeline/google-cloud-pipeline-components:1.0.0"
        ▾ exec-endpoint-create:
          ▾ container:
            ▸ args: [] 12 items
            ▸ command: [] 4 items
              image: "gcr.io/ml-pipeline/google-cloud-pipeline-components:1.0.0"
        ▾ exec-image-classification-model-eval-metrics:
          ▾ container:
            ▸ args: [] 8 items
            ▸ command: [] 7 items
              image: "gcr.io/deeplearning-platform-release/tf2-cpu.2-5:latest"
        ▾ exec-image-dataset-create:
          ▾ container:
            ▸ args: [] 20 items
            ▸ command: [] 7 items
              image: "gcr.io/ml-pipeline/google-cloud-pipeline-components:1.0.0"
        ▾ exec-model-deploy:
          ▾ container:
            ▸ args: [] 20 items
            ▸ command: [] 4 items
              image: "gcr.io/ml-pipeline/google-cloud-pipeline-components:1.0.0"
    ▸ pipelineInfo:
    ▸ root:
```

***Figure 6.11**: JSON file contents*

JSON will have all the details regarding components of the pipeline, parameters of each component, run time configuration, parameters, and so on.

Except the model evaluation component, all other components will be executed on the ML-pipeline component. Only the model evaluation component will be executed on the deep-learning packages (which is mentioned in the code).

Step 8: Pipeline creation

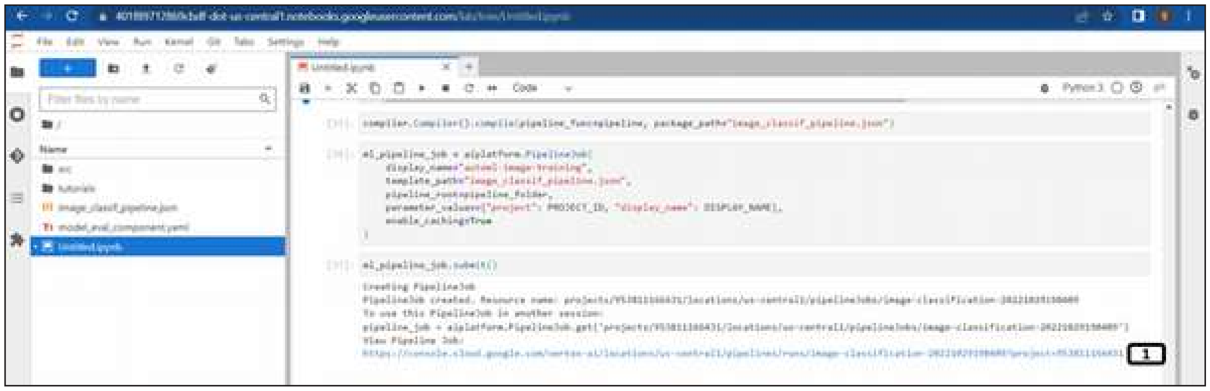
Run the following code to create the pipeline:

```
ml_pipeline_job = aiplatform.PipelineJob(

    display_name=”automl-image-training”,

    template_path=”image_classif_pipeline.json”,

    pipeline_root=pipeline_folder,

    parameter_values={“project”: PROJECT_ID, “display_name”: DISPLAY_
    NAME},

    enable_caching=True

)
```

Step 9: Submitting the Pipeline job

Run the following code to submit the pipeline job. The link will be provided once the code is executed for the pipeline as shown in *Figure 6.12*:

```
ml_pipeline_job.submit()
```



```
[ ]: compiler.Compiler().compile(pipeline_func=pipeline, package_path="image_classif_pipeline.json")

[ ]: ml_pipeline_job = aiplatform.PipelineJob(
         display_name="automl-image-training",
         template_path="image_classif_pipeline.json",
         pipeline_root=pipeline_folder,
         parameter_values={"project": PROJECT_ID, "display_name": DISPLAY_NAME},
         enable_caching=True
     )

[ ]: ml_pipeline_job.submit()

     Creating PipelineJob
     PipelineJob created. Resource name: projects/[illegible]/locations/us-central1/pipelineJobs/image-classification-[illegible]
     To use this PipelineJob in another session:
     pipeline_job = aiplatform.PipelineJob.get('projects/[illegible]/locations/us-central1/pipelineJobs/image-classification-[illegible]')
     View Pipeline Job:
     https://console.cloud.google.com/vertex-ai/locations/us-central1/pipelines/runs/image-classification-[illegible]?project=[illegible]
```

Figure 6.12: *Pipeline link in the notebook file*

1. Once the pipeline’ job is submitted, a link will be provided to check the status of the pipeline. Click on it.

Execution of Pipeline

Once the job is submitted for the execution, pipeline will be visible under the pipeline section of Vertex AI, status of each components will be displayed (yet to start, success, failure).Follow the below steps to check the status of the submitted pipeline.

Step 1: Pipeline of image classification

Open the link as shown in *Figure 6.13* to navigate to pipelines of Vertex AI. The pipeline will start executing and this exercise will take about 90 to 120 mins. All the five tasks in the exercise will be executed sequentially. Once the complete pipeline is executed, you will see a green colored tick mark on all the components as shown in *Figure 6.13*:

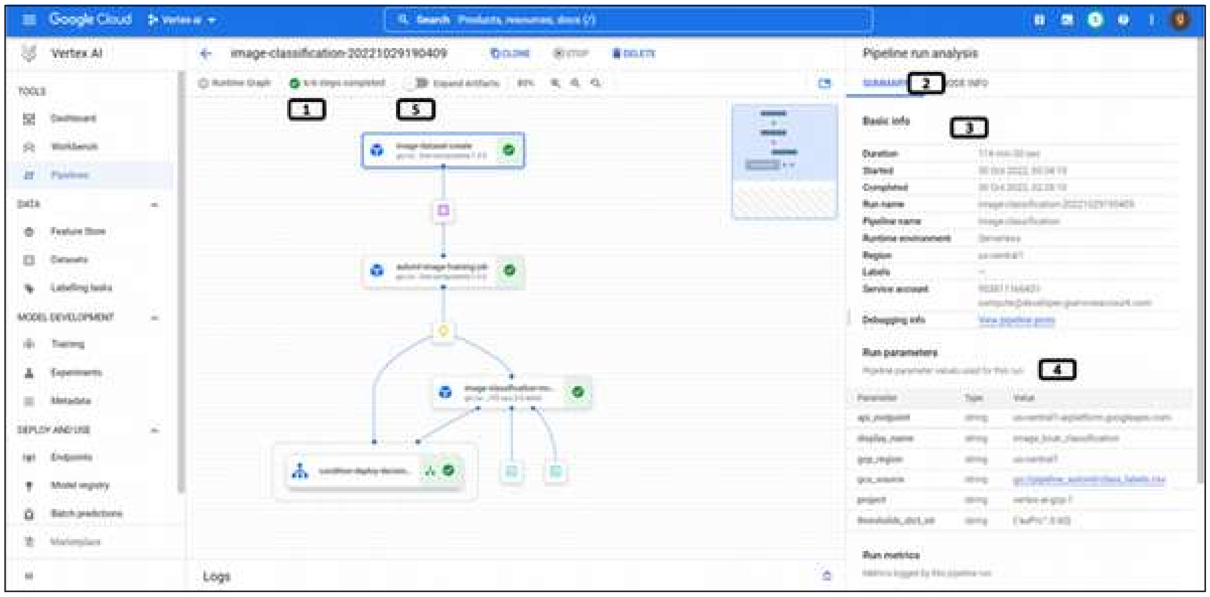


***Figure 6.13**: Pipeline created*

1. Indicating all the steps in the pipelines are executed successfully.
2. Click on **Summary** section of the pipeline.
3. **Basic info** provides the information of pipeline run, region, link to view the logs, and so on.
4. **Run parameter** provides information regarding the parameters that are used for the pipeline. Scroll down to get the run metrics in the *Summary* section.
5. Click the toggle button to expand the artifacts.

Step 2: Expand artifacts

Artifacts will be expanded in the pipeline as shown in *Figure 6.14*:

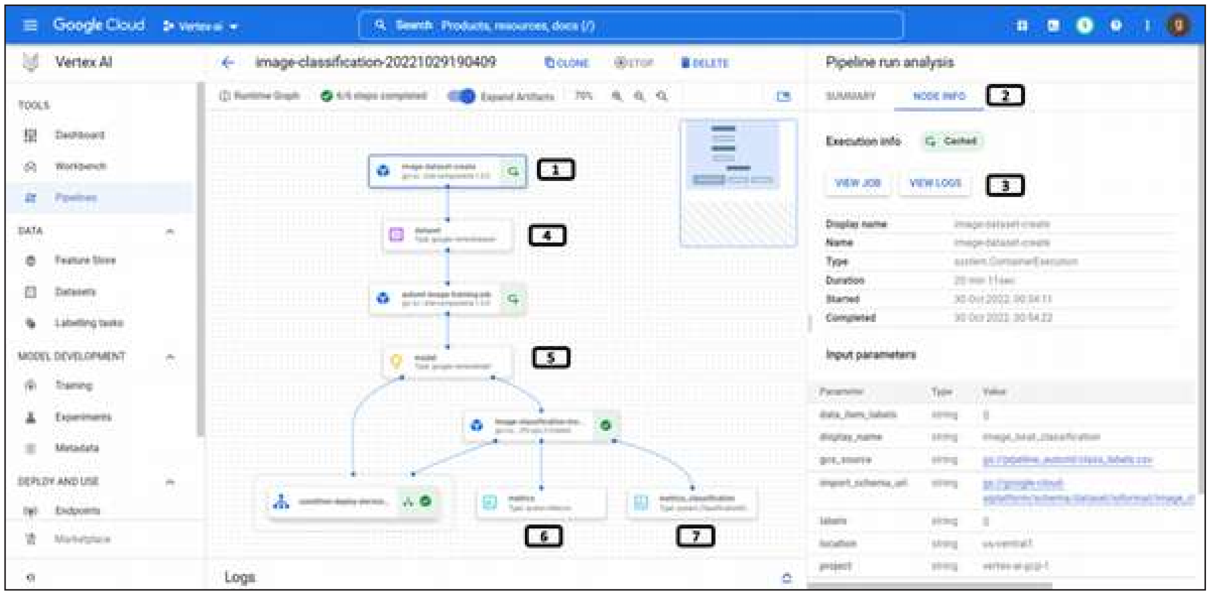
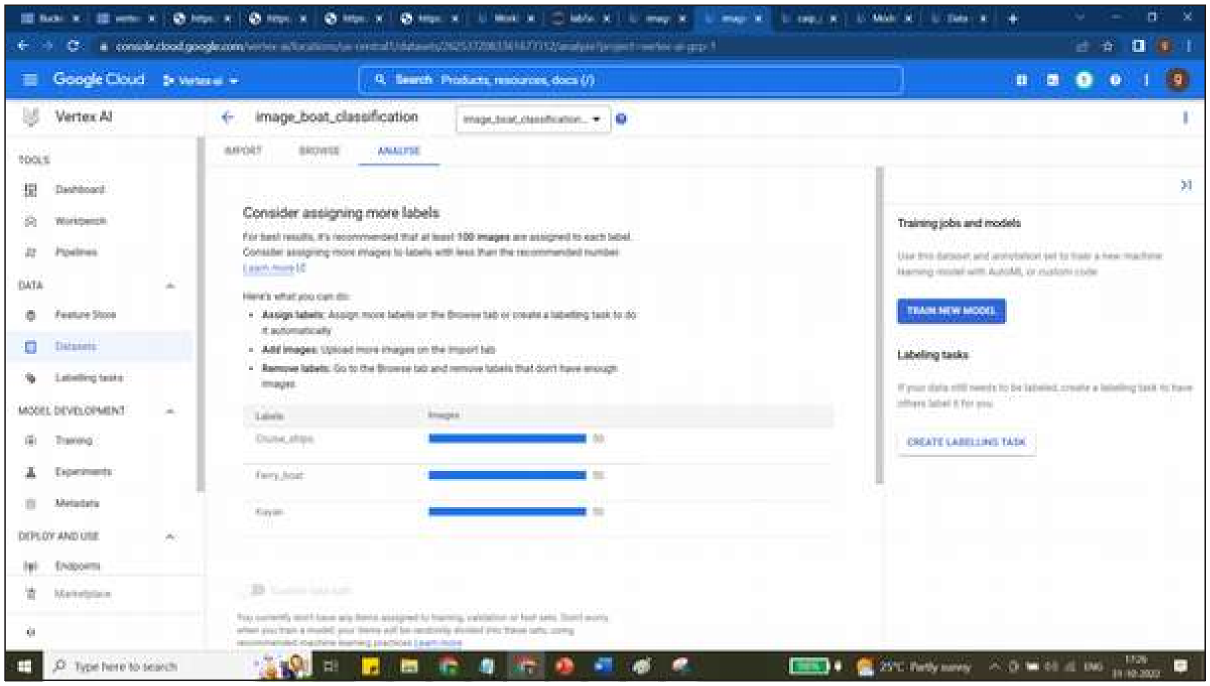


Figure 6.14: *Components and artifacts of the pipeline*

1. The first component in the pipeline is to create a dataset from the csv and image files from the cloud storage.
2. Click on the **Node Info** to get the basic info, input parameters and output parameters for first component.
3. View logs populates the log for this component at the bottom section.
4. Artifacts of the **dataset** created. Click on the artifact component in the node info link will be provided to the dataset created. Refer to *Figure 6.15*.
5. Artifacts of the trained **model**. Click on the artifact component in the node info link will be provided to the trained model is saved. Refer *Figure 6.16*.
6. Click on **metrics** to get the confusion matrix in the JSON format, and other model evaluation metrices.
7. Click on metrics to get the confusion matrix in the image format.

NOTE: If the pipeline fails due to errors in any components. Users can troubleshoot the component and re-run the pipeline. In this scenario the successfully executed components will not be executed again and the results will be picked from the cache.

The created dataset available under the dataset module of vertex AI is shown in *Figure 6.15*:



image_boat_classification

Consider assigning more labels

For best results, it's recommended that at least **100 images** are assigned to each label. Consider assigning more images to labels with less than the recommended number. Learn more

Here's what you can do:

- **Assign labels:** Assign more labels on the Browse tab or create a labelling task to do it automatically
- **Add images:** Upload more images on the import tab
- **Remove labels:** Go to the Browse tab and remove labels that don't have enough images

Labels	Images
Cruise_ships	50
Ferry_boat	50
Kayak	50

Training jobs and models

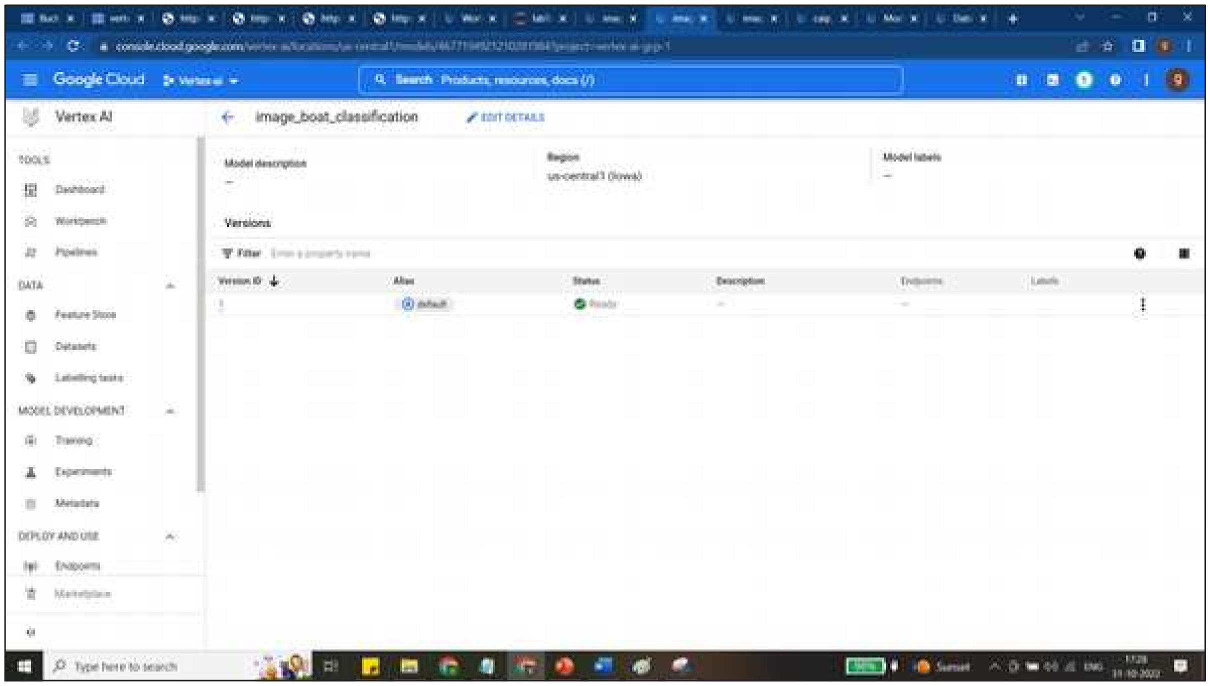
Use this dataset and annotation set to train a new machine learning model with AutoML or custom code

Labeling tasks

If your data still needs to be labeled, create a labeling task to have others label it for you

Figure 6.15: Dataset section of Vertex AI

Trained model will be available under the model section of Vertex AI as shown in *Figure 6.16*:



image_boat_classification

Model description: –

Region: us-central1 (Iowa)

Model labels: –

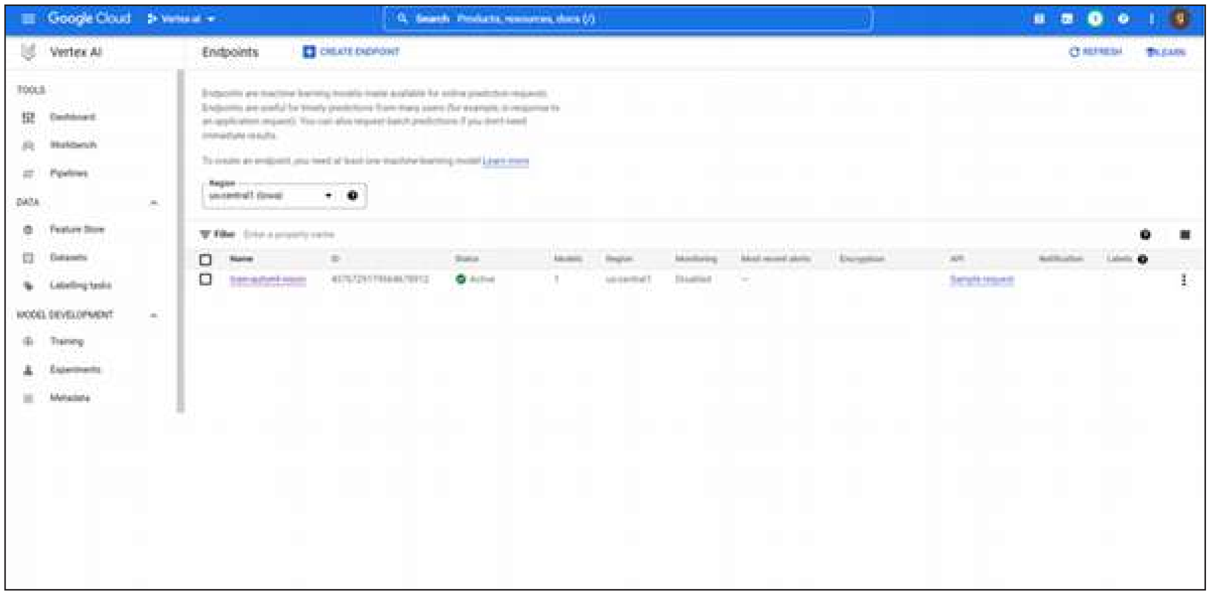
Versions

Version ID	Alias	Status	Description	Endpoints	Labels
1	default	Ready	–	–	

Figure 6.16: AutoML classification model listed under model registry of Vertex AI

Step 3: Model deployment

In this exercise, since the model performance was above the threshold, endpoint is created, and model is deployed to the end point as shown in the *Figure 6.17*:



Endpoints are machine learning models made available for online prediction requests. Endpoints are useful for timely predictions from many users (for example, in response to an application request). You can also request batch predictions if you don't need immediate results.

To create an endpoint, you need at least one machine learning model. Learn more

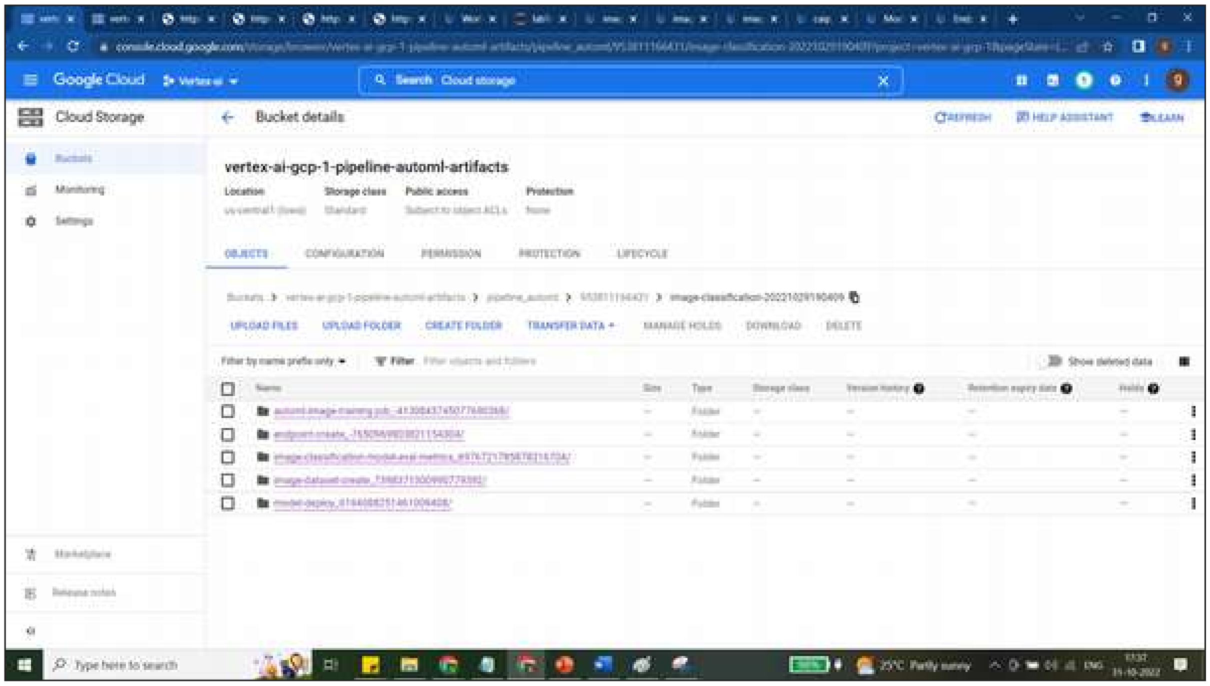
Region: us-central1 (Iowa)

Name	ID	Status	Models	Region	Monitoring	Most recent alerts	Encryption	API	Notification	Labels
[illegible]	[illegible]	Active	1	us-central1	Disabled	–		Sample request		

Figure 6.17: *Model deployed to endpoint*

Step 4: Pipeline artifacts stored in cloud storage

Pipeline artifacts are stored in the cloud storage as shown in *Figure 6.18*:



vertex-ai-gcp-1-pipeline-automl-artifacts

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Subject to object ACLs	None

Buckets > vertex-ai-gcp-1-pipeline-automl-artifacts > pipeline_automl > [illegible] > image-classification-[illegible]

Name	Size	Type	Storage class	Version history	Retention expiry date	Holds
automl-image-training-job_[illegible]/	–	Folder	–	–	–	–
endpoint-create_[illegible]/	–	Folder	–	–	–	–
image-classification-model-eval-metrics_[illegible]/	–	Folder	–	–	–	–
image-dataset-create_[illegible]/	–	Folder	–	–	–	–
model-deploy_[illegible]/	–	Folder	–	–	–	–

Figure 6.18: *Pipeline artifacts stored in the cloud storage*

Deleting resources

We have utilized workbench, cloud storage to store the images, model has been trained and deployed to the end point. You need to ensure to delete the workbench, clear the data stored in the cloud storage, delete the dataset created, remove the model from end point, and delete the end point.

Conclusion

In this chapter, we learnt about learnt about the pipelines, what is Kubeflow and its components. How to construct the pipeline using Kubeflow SDK and using AutoML component of GCP for model training. How to submit the pipeline for run and visually understand the pipeline and check the results.

In the next chapter, we will construct the pipeline using Kubeflow SDK for custom model training.

Questions

1. Pipelines built on what can be executed on the Vertex AI?
2. Why in the code, the base image information is mentioned only in the custom model evaluation function?
3. Why do we need to compile the pipeline before submitting it?
4. Will all the components will be executed from beginning if the pipeline gets executed after troubleshooting of errors?

Chapter 7
Pipelines using Kubeflow for Custom Models

Introduction

In the previous chapter, we worked on the pipelines of GCP using Kubeflow. We built the first pipeline using AutoML of the platform for model training. In this chapter, we will see how to build pipelines for custom models and compare the results of different pipelines. We will also understand a few differences between Vertex AI pipelines and Kubeflow pipelines.

Structure

In this chapter, we will cover the following topics:

- Data for model training
- Additional permissions
- Creation of workbench using cloud shell
- Pipeline code walk through
- Pipeline
- Pipeline comparison
- Deletion of resources

Objectives

By the end of this chapter, users will be able to build pipelines using Kubeflow for custom model building and compare the comparison of results of the different pipelines.

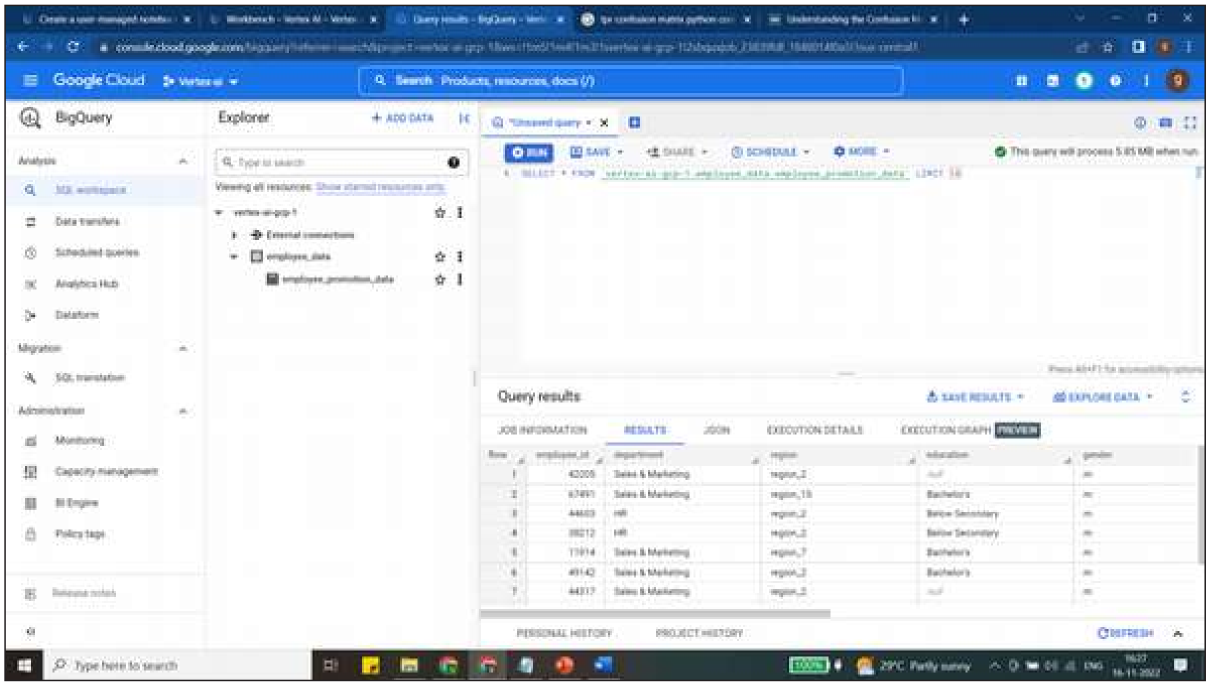
Data for model training

For this exercise, data is downloaded from Kaggle (link is provided below), and the dataset is listed under CC0:Public domain licenses. In previous chapters we have used cloud storage to upload the data to train the model, in this chapter data is being uploaded to BigQuery of GCP on which machine learning model is trained.

https://www.kaggle.com/datasets/arashnic/hr-ana

Refer to *Biggquery* section from *Chapter 1, Basics of Google Cloud Platform* to upload the data. *Figure 1.43* to *1.47* will help you upload the data to BigQuery. Refer to *Figure 1.48* to run a small query to check if the data upload is successful.

To upload data to BigQuery, create a dataset by the name **`employee_data`**. Under the dataset, create a table called **`employee_promotion_data`**. In the query section, use the query select **`* from ‘vertex-ai-gcp-1. employee_data. employee_promotion_data’ limit 10`** to check the data upload. *Figure 7.1* shows data that is uploaded to BigQuery:



```
SELECT * FROM `vertex-ai-gcp-1.employee_data.employee_promotion_data` LIMIT 10
```

Row	employee_id	department	region	education	gender
1	42205	Sales & Marketing	region_2	null	m
2	67491	Sales & Marketing	region_15	Bachelor's	m
3	44603	HR	region_2	Below Secondary	m
4	38212	HR	region_2	Below Secondary	m
5	11914	Sales & Marketing	region_7	Bachelor's	m
6	49142	Sales & Marketing	region_2	Bachelor's	m
7	44317	Sales & Marketing	region_2	null	m

Figure 7.1: *Data loaded into BigQuery table*

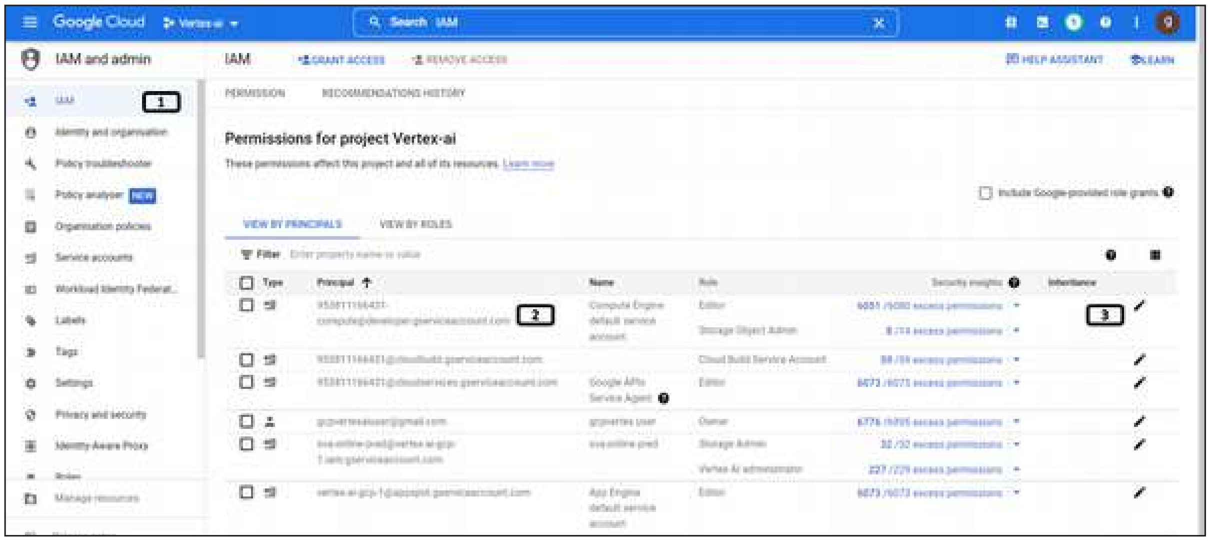
In this chapter we will create pipeline on Vertex AI component of GCP using Kubeflow SDK. In the pipeline we will use the AutoML component of GCP for model training.

Additional permissions

We also need to grant additional permission to the service account associated with the compute engine of GCP since we are fetching data from BigQuery. Follow these steps to grant the required permission.

Step 1: Open IAM and admin section

Follow the steps mentioned in *Figure 7.2* to add roles to the service account associated with compute engine of GCP:



Permissions for project Vertex-ai

These permissions affect this project and all of its resources. Learn more

Include Google-provided role grants

VIEW BY PRINCIPALS VIEW BY ROLES

Filter Enter property name or value

Type	Principal	Name	Role	Security insights	Inheritance
	953811166431-compute@developer.gserviceaccount.com	Compute Engine default service account	Editor	6051 /6080 excess permissions	
			Storage Object Admin	8 /14 excess permissions	
	953811166431@cloudbuild.gserviceaccount.com		Cloud Build Service Account	59 /59 excess permissions	
	953811166431@cloudservices.gserviceaccount.com	Google APIs Service Agent	Editor	6073 /6075 excess permissions	
	gcpvertexaiuser@gmail.com	gcpvertex user	Owner	6776 /6995 excess permissions	
	sva-online-pred@vertex-ai-gcp-1.iam.gserviceaccount.com	sva-online pred	Storage Admin	32 /32 excess permissions	
			Vertex AI administrator	227 /229 excess permissions	
	vertex-ai-gcp-1@appspot.gserviceaccount.com	App Engine default service account	Editor	6073 /6073 excess permissions	

***Figure 7.2**: IAM of GCP*

1. Click **IAM**.
2. Choose the service account associated with the project.
3. Click Edit **Principal**.

Step 2: Additional role

Follow the steps mentioned in *Figure 7.3, 7.4* and *7.5* to edit the roles of the service account of compute engine:

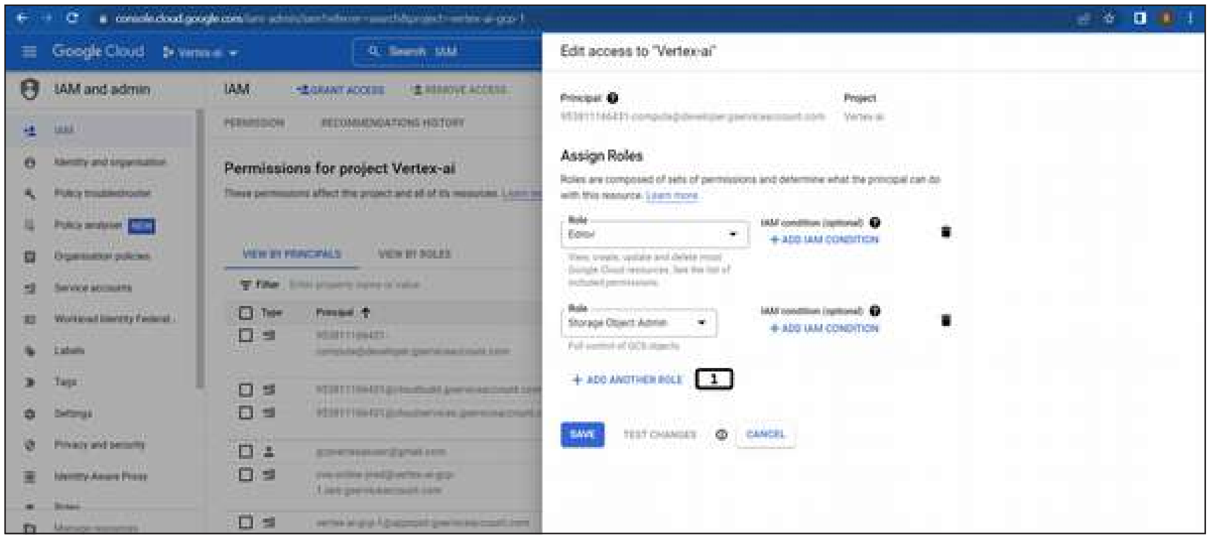

Edit access to "Vertex-ai"
Principal
Project
Assign Roles
Roles are composed of sets of permissions and determine what the principal can do with this resource. Learn more
Role
Editor
IAM condition (optional)
+ ADD IAM CONDITION
Storage Object Admin
Full control of GCS objects
+ ADD ANOTHER ROLE
1
SAVE
TEST CHANGES
CANCEL

Figure 7.3: *Edit roles of service account*

1. Click **+ Add Another Role** and follow the steps mentioned in the *Figure 7.4*:

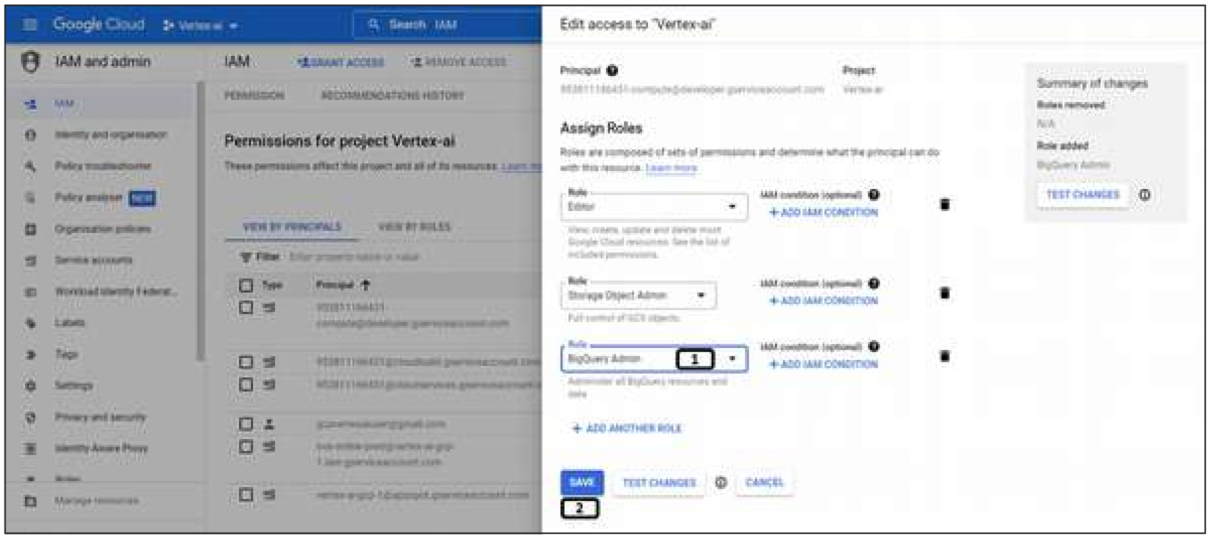
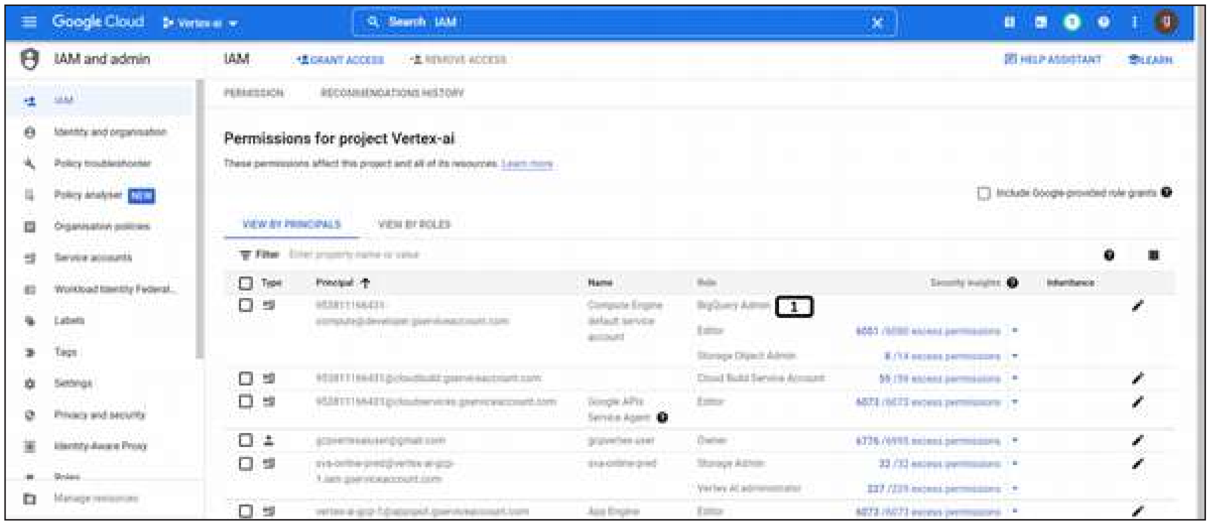


Figure 7.4: *Adding BigQuery admin role*

1. Select **BigQuery admin** role under the BigQuery product/service.
2. Click **SAVE**. A new role will be added to the service account as shown in the following figure:



Permissions for project Vertex-ai

Type	Principal	Name	Role	Security insights	Inheritance
	953811166431-compute@developer.gserviceaccount.com	Compute Engine default service account	BigQuery Admin 1		
			Editor	6051 /6080 excess permissions	
			Storage Object Admin	8 /14 excess permissions	
	953811166431@cloudbuild.gserviceaccount.com		Cloud Build Service Account	59 /59 excess permissions	
	953811166431@cloudservices.gserviceaccount.com	Google APIs Service Agent	Editor	6073 /6073 excess permissions	
	gcpvertexaiuser@gmail.com	gcpvertex user	Owner	6776 /6995 excess permissions	
	sva-online-pred@vertex-ai-gcp-1.iam.gserviceaccount.com	sva-online-pred	Storage Admin	32 /32 excess permissions	
			Vertex AI administrator	227 /235 excess permissions	
	vertex-ai-gcp-1@appspot.gserviceaccount.com	App Engine	Editor	6073 /6073 excess permissions	

Figure 7.5: New role added to service account is visible in the IAM

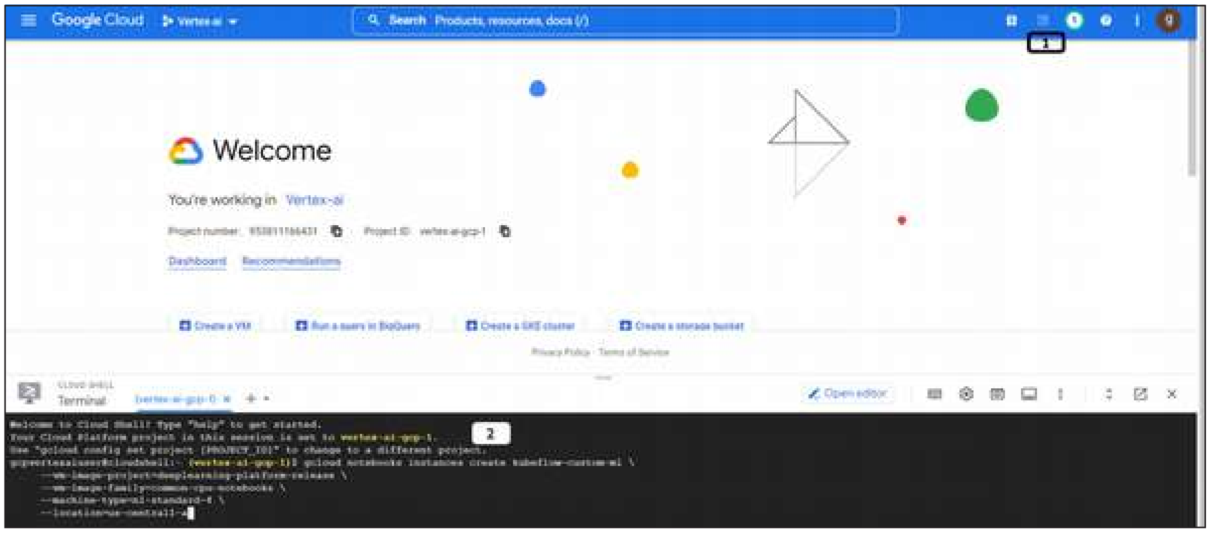
1. New role will be visible for the service account associated with the compute engine.

Creation of Workbench

In all the previous chapters we have worked on the workbench. Vertex AI Workbench and custom model training guides on creating a workbench using the GUI approach. Follow these steps to create Vertex AI workbench from the cloud shell:

Step 1: Activate cloud shell

Follow the steps shown in *Figure 7.6* to activate the cloud shell and type the following commands:



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to vertex-ai-gcp-1.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
gcpvertexaiuser@cloudshell:~ (vertex-ai-gcp-1)$ gcloud notebooks instances create kubeflow-custom-ml \
    --vm-image-project=deeplearning-platform-release \
    --vm-image-family=common-cpu-notebooks \
    --machine-type=n1-standard-4 \
    --location=us-central1-a
```

Figure 7.6: Activate cloud shell

1. Click Cloud shell to activate it
2. Type in the following mentioned commands in the cloud shell to create workbench:

```
gcloud notebooks instances create kubeflow-custom-ml \
      --vm-image-project=deeplearning-platform-release \
      --vm-image-family=common-cpu-notebooks \
      --machine-type=n1-standard-4 \
      --location=us-central1-a
```

Step 2: Created workbench

The created workbench will be listed under workbench section of Vertex AI as shown in *Figure 7.7*:

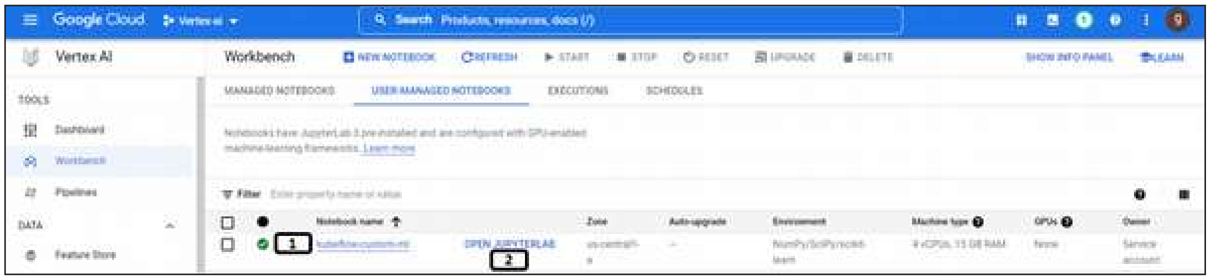


***Figure 7.7**: Workbench creation complete*

1. Newly created workbench.
2. Click on **OPEN JUPYTERLAB**.

Pipeline code walk through

We will be using Python 3 notebook file to type commands, create a pipeline, compile and to run it. Follow the following mentioned steps to create a Python file and type the Python codes given in this section.

Step 1: Create Python notebook file

Once the workbench is created, open the Jupyterlab and follow the steps mentioned in *Figure 7.8* to create Python notebook file:

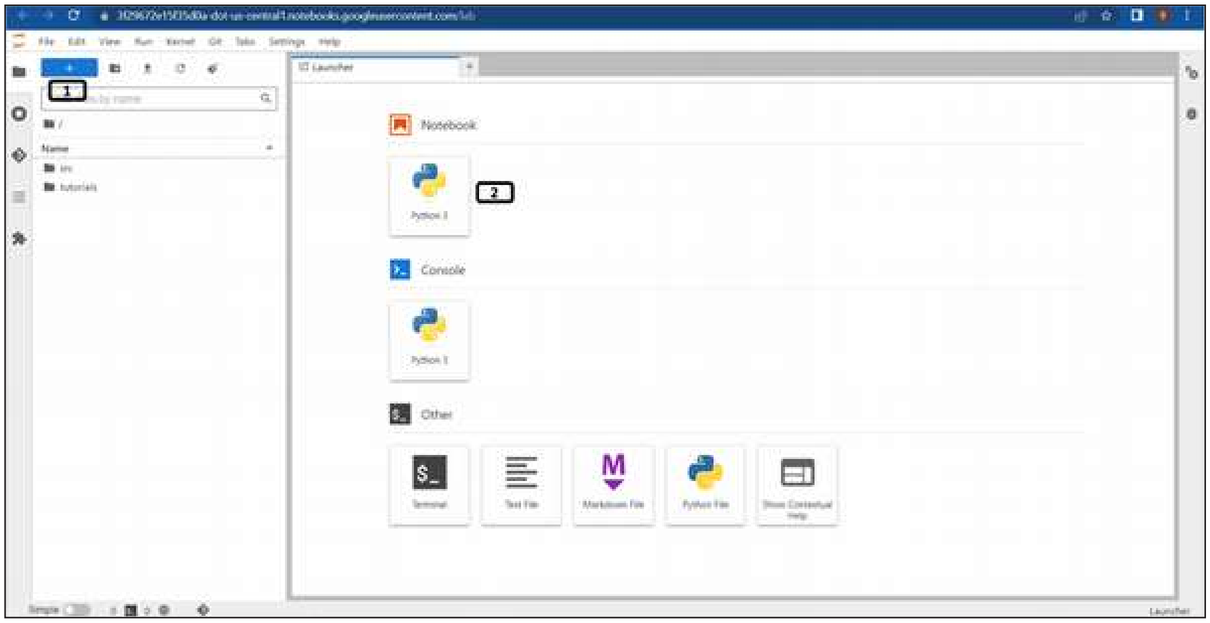


***Figure 7.8**: New launcher window*

1. Click New launcher.
2. Double click the **Python 3** notebook file to create one.

   Step 2 onwards, run the given codes in separate cells.

Step 2: Package installation

Run the following commands to install the Kubeflow, google cloud pipeline and google cloud aiplatform package. (It will take few minutes to install the packages):

```
USER_FLAG = "--user"

!pip3 install {USER_FLAG} google-cloud-aiplatform==1.3.0 --upgrade

!pip3 install {USER_FLAG} kfp --upgrade

!pip install google_cloud_pipeline_components
```

Step 3: Kernel restart

Type the following commands in the next cell, to restart the kernel. (Users can restart kernel from the GUI as well):

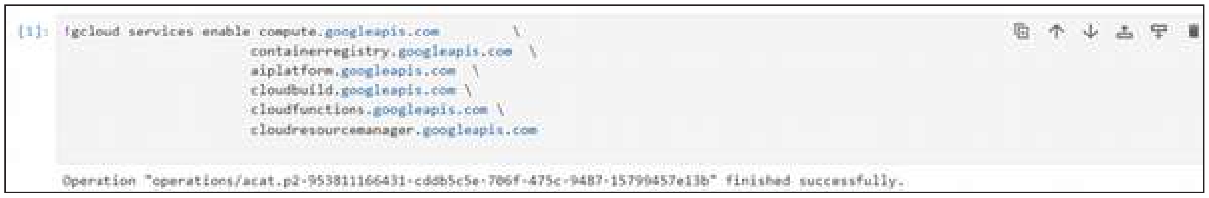
```
import os

if not os.getenv("IS_TESTING"):

    import IPython
    app = IPython.Application.instance()

    app.kernel.do_shutdown(True)
```

Step 4: Enabling APIs

Run the following mentioned lines of code to enable the APIs. We used to enable APIs from cloud shell. This is another way of enabling the APIs from the Python notebook (except cloud resource manager all other APIs are enabled in previous chapters):

```
!gcloud services enable compute.googleapis.com \

                        containerregistry.googleapis.com \

                        aiplatform.googleapis.com \

                        cloudbuild.googleapis.com \

                        cloudfunctions.googleapis.com \

                        cloudresourcemanager.googleapis.com
```

If all the APIs are enabled successfully, the success message will be printed as shown in *Figure 7.9*:



```
[1]: !gcloud services enable compute.googleapis.com          \
                         containerregistry.googleapis.com  \
                         aiplatform.googleapis.com  \
                         cloudbuild.googleapis.com \
                         cloudfunctions.googleapis.com \
                         cloudresourcemanager.googleapis.com

Operation "operations/acat.p2-953811166431-cddb5c5e-706f-475c-9487-15799457e13b" finished successfully.
```

Figure 7.9*: APIs enabled successfully*

Step 5: Importing packages

Run the following lines of codes in a new cell to import the required packages:

```
from typing import NamedTuple

from kfp.v2 import dsl

from kfp.v2.dsl import (Artifact, Dataset, Input, Model, Output,Metrics,
ClassificationMetrics, component, OutputPath, InputPath)

from kfp.v2 import compiler

from google.cloud import BigQuery

from google.cloud import aiplatform
from google.cloud.aiplatform import pipeline_jobs

from datetime import datetime
```

Using the vertex AI pipeline, we are creating a dataset from the data available in the BigQuery, apply minimal data transformation and split the data for the training and testing purpose, train the model with the training dataset and evaluate the model performance on the testing dataset. We are saving the artifacts (or intermediate outputs) at each stage of the pipeline to cloud storage.

Step 6: Bucket for storing artifacts of the pipeline

Run the following lines of codes to set the location to store the artifacts. Bucket will be created during the pipeline run:

```
USER_FLAG = “--user”

REGION=”us-central1”


#Setting the environment path

PATH=%env PATH

%env PATH={PATH}:/home/jupyter/.local/bin

# to get the projet name

shell_output=!gcloud config get-value project 2> /dev/null

PROJECT_ID=shell_output[0]


# Set bucket name to store the artifacts

BUCKET_NAME=”gs://”+PROJECT_ID+”-bucket-employee”


PIPELINE_ROOT = f”{BUCKET_NAME}/pipeline_root_employee/”

PIPELINE_ROOT
```

Step 7: First component of the pipeline (to fetch data from BigQuery)

Run the below mentioned codes to define the custom component for model evaluation and threshold comparison which will be included in the pipeline. Important points about the code are:

- Decorator component is used to define this function as a component of the pipeline.
- Input for the component will be location of the bigquery table from which data needs to be fetched:

  ```
  #First Component in the pipeline to fetch data from big query.
  @component(
      packages_to_install=[“google-cloud-bigquery==2.34.1”,
      “pandas”, “pyarrow”,”db-dtypes”,”scikit-learn”],
      base_image=”python:3.7”,
      output_component_file=”data_fetch_bigquery.yaml”
  )
  def get_data(bq_table : str, output_data_path: Output[Dataset]):
      from google.cloud import bigquery
      import pandas as pd
      bqclient = bigquery.Client()
      table = bigquery.TableReference.from_string(bq_table)
      rows = bqclient.list_rows(table)
      dataframe = rows.to_dataframe(create_bqstorage_client=True)
      dataframe.to_csv(output_data_path.path + “.csv” ,
      index=False, encoding=’utf-8-sig’)
  ```

Step 8: Second component of the pipeline (data transformation)

Run the following codes to define the second component of the pipeline. It will apply data transformation and split the data to train and test samples. Input for this component will be dataset from the previous component and the outputs will be two separate datasets.

```
#Second component in the pipeline to apply data transformation and split
the data
@component(
    packages_to_install=[“scikit-learn”, “pandas”],
    base_image=”python:3.7”,
    output_component_file=”data_transformation.yaml”,
)
```

```
def data_transformation(
    dataset: Input[Dataset],
    dataset_train: Output[Dataset],
    dataset_test: Output[Dataset]
):

    from sklearn.metrics import roc_curve
    from sklearn.model_selection import train_test_split
    from joblib import dump
    from sklearn.metrics import confusion_matrix
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier
    import pandas as pd

    data = pd.read_csv(dataset.path+".csv")
    data = data.copy()
    data = data.drop([‘region’,’employee_id’],axis=1)
    data = data.dropna(axis=0)
    data[‘gender’] = data[‘gender’].replace({‘f’:0, ‘m’:1})
    dummies = pd.get_dummies(data[‘department’])
    data = pd.concat([data,dummies],axis=1)
    data = data.drop(‘department’,axis=1)
    edu_ranking = {“Master’s & above” : 1, “Bachelor’s” : 2 , “Below
    Secondary” : 3}
    recruit_ranking = {‘sourcing’ : 1, ‘other’ : 2, ‘referred’ : 3}
    data[‘education’] = data[‘education’].map(edu_ranking)
    data[‘recruitment_channel’] = data[‘recruitment_channel’].
    map(recruit_ranking)

    train, test = train_test_split(data, test_size=0.3)

    train.to_csv(dataset_train.path + ".csv" , index=False,
    encoding='utf-8-sig')

    test.to_csv(dataset_test.path + ".csv" , index=False, encoding='utf-
    8-sig')
```

Step 9: Third component of the pipeline (Model training)

Run the following codes to define the third component of the pipeline. It will train the model (random forest classifier) on the training dataset. Input for this component will be training dataset from the second component, and the output will be the trained model:

```
#Third component Model training

@component(

    packages_to_install = [

        "pandas",

        "scikit-learn",

    ], base_image="python:3.9",

)

def emp_promotion_training(

    dataset:  Input[Dataset],

    model: Output[Model],

):


    from sklearn.ensemble import RandomForestClassifier

    import pandas as pd

    import pickle

    data = pd.read_csv(dataset.path+".csv")

    model_rf = RandomForestClassifier(n_estimators=10)
    model_dt.fit(
        data.drop(columns=[“is_promoted”]),
        data.is_promoted,
    )
    model.metadata[“framework”] = “decision tree”
    file_name = model.path + f”.pkl”
    with open(file_name, ‘wb’) as file:
        pickle.dump(model_rf, file)
```

Step 10: Fourth component of the pipeline (Model evaluation)

Run the following codes to define the fourth component of the pipeline. It will import the trained model from the third component and test dataset from the second component. Check the model performance on the test dataset and log the model metrices:

```
#Fourth component Model evaluation
@component(
    packages_to_install = [
        “pandas”,
        “scikit-learn”
    ], base_image=”python:3.9”,
)
def model_evaluation(
    test_set:  Input[Dataset],
    trained_model: Input[Model],
    metrics: Output[ClassificationMetrics],
    kpi: Output[Metrics]
):

    from sklearn.ensemble import RandomForestClassifier
```

```
import pandas as pd

import logging

import pickle

from sklearn.metrics import roc_curve, confusion_matrix, accuracy_
score, recall_score, precision_score

import json

import typing



data = pd.read_csv(test_set.path+”.csv”)

model = RandomForestClassifier()

file_name = trained_model.path + “.pkl”

with open(file_name, ‘rb’) as file:

    model = pickle.load(file)


y_test = data.drop(columns=[“is_promoted”])

y_target=data.is_promoted

y_pred = model.predict(y_test)

y_scores =  model.predict_proba(data.drop(columns=[“is_promoted”]))
[:, 1]

fpr, tpr, thresholds = roc_curve(

     y_true=data.is_promoted.to_numpy(), y_score=y_scores, pos_
     label=True

)

metrics.log_roc_curve(fpr.tolist(), tpr.tolist(), thresholds.
tolist())


metrics.log_confusion_matrix(

   [“False”, “True”],
   confusion_matrix(
       data.is_promoted, y_pred
   ).tolist(),
)
accuracy = accuracy_score(data.is_promoted, y_pred.round())
recall=recall_score(data.is_promoted, y_pred.round(), pos_label=1)
precision=precision_score(data.is_promoted, y_pred.round(), pos_
label=1)
trained_model.metadata[“accuracy”] = float(accuracy)
trained_model.metadata[“recall”] = float(recall)
trained_model.metadata[“precision”] = float(precision)
kpi.log_metric(“accuracy”, float(accuracy))
kpi.log_metric(“recall”, float(recall))
kpi.log_metric(“precision”, float(precision))
```

Step 11: Pipeline construction

Run the following codes to define the pipeline with the custom components:

- Decorator **`kfp.dsl.pipeline`** is used to define this function as a pipeline.
- Input for the component will be the trained model (and the artifacts) which will be collected from **`Input[Artifacts]`**.
- Function **`fetch_eval_info`** will fetch the evaluation data from the trained model artifacts, which will be the passed to the **`metrics_log_check`** function to check if the model performance is above the threshold and retunes the value to be **`true`** (if the model performance is better than the threshold) or **`false`** (if other wise):

```
@dsl.pipeline(
    # Default pipeline root. You can override it when submitting
the pipeline.
    pipeline_root=PIPELINE_ROOT,
    # A name for the pipeline. Use to determine the pipeline
    Context.
    name=”employee-pipeline-rf”,
)
def pipeline(
    bq_table: str = “vertex-ai-gcp-1.employee_data.employee_
    promotion_data”,
    project: str = PROJECT_ID,
    region: str = REGION,
    display_name: str = ‘pipeline-employeepromotion-rf-job{}’.
    format(datetime.now().strftime(“%Y%m%d%H%M%S”)),
    ):

    data = get_data(bq_table)
    data_transformed=data_transformation(data.outputs[“output_
    data_path”])
    train_model = emp_promotion_training(data_transformed.
    outputs[“dataset_train”])

    model_evaluation_op = model_evaluation(
        test_set=data_transformed.outputs[“dataset_test”],
        trained_model=train_model.outputs[“model”],
    )
```

Step 12: Compile using Kubeflow

Run the following codes to compile the pipeline code. The correct data type usage in pipelines is verified by the Kubeflow Pipelines SDK v2 compiler, as is the avoidance of inputs from parameters being used as outputs from artifacts and vice versa:

```
compiler.Compiler().compile(pipeline_func=pipeline,package_path=’employ-
ee_promotion_prediction_rf.json’)
```

Step 13: Pipeline creation

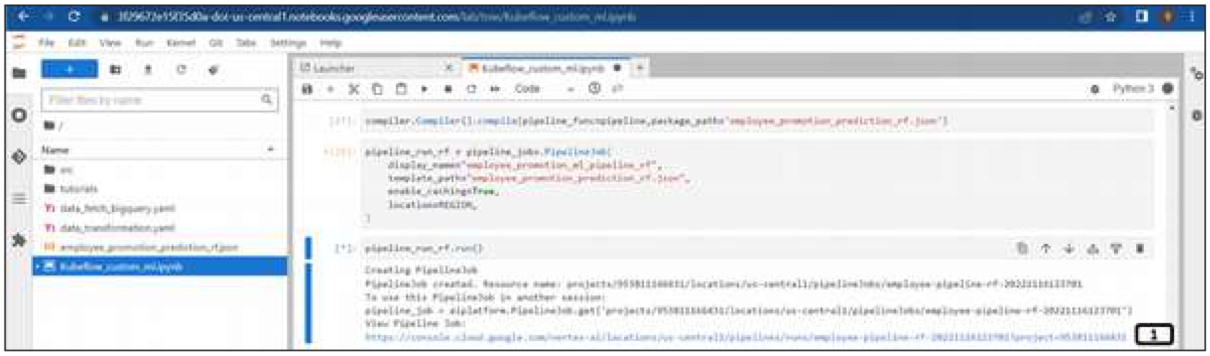
Run the following codes to create the pipeline:

```
pipeline_run_rf = pipeline_jobs.PipelineJob(
    display_name="employee_promotion_ml_pipeline_rf",

    template_path="employee_promotion_prediction_rf.json",

    enable_caching=True,

    location=REGION,

)
```

Step 14: Submitting the Pipeline job

Run the following code to submit the pipeline job. Link will be provided once the code is executed for the pipeline as shown in *Figure 7.10*:

```
pipeline_run_rf.run()
```



```
compiler.Compiler().compile(pipeline_func=pipeline,package_path="employee_promotion_prediction_rf.json")

pipeline_run_rf = pipeline_jobs.PipelineJob(
    display_name="employee_promotion_ml_pipeline_rf",
    template_path="employee_promotion_prediction_rf.json",
    enable_caching=True,
    location=REGION,
)

pipeline_run_rf.run()
Creating PipelineJob
PipelineJob created. Resource name: projects/[illegible]/locations/us-central1/pipelineJobs/employee-pipeline-rf-[illegible]
To use this PipelineJob in another session:
pipeline_job = aiplatform.PipelineJob.get('projects/[illegible]/locations/us-central1/pipelineJobs/employee-pipeline-rf-[illegible]')
View Pipeline Job:
https://console.cloud.google.com/vertex-ai/locations/us-central1/pipelines/runs/employee-pipeline-rf-[illegible]?project=[illegible]
```

***Figure 7.10**: Pipeline link in the notebook file*

1. Once the pipeline's job is submitted, a link will be provided to check the status of the pipeline. Click on it.

Pipeline

Follow these steps to analyze the status of the pipeline job, artifacts, lineage and output:

Step 1: Pipeline of custom model

Open the link as shown in *Figure 7.10* to navigate to the pipelines of Vertex AI. The pipeline will start executing and will take about 5 to 10 mins. All the four tasks in the exercise will be executed sequentially. Once the complete pipeline is executed, you will see green color tick mark on all the components as shown in *Figure 7.11*:

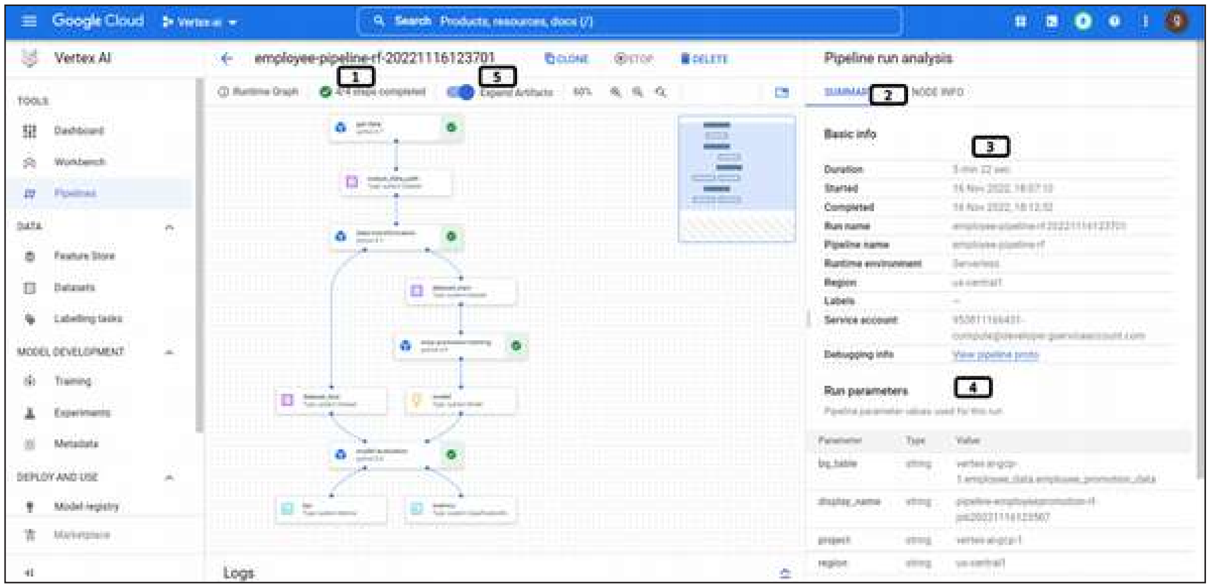


Figure 7.11: Pipeline created & executed successfully

1. Indicating all the steps in the pipelines are executed successfully.
2. Click **SUMMARY** section of the pipeline
3. Basic info provides the information of pipeline run, region, link to view the logs, and so on.
4. **Run parameters** provides information regarding the parameters that are used for the pipeline. Scroll down to get the run metrics in the summary section.
5. Click the toggle button to expand the artifacts

Step 2: Expand artifacts

Artifacts will be expanded in the pipeline as shown in *Figure 7.12*:

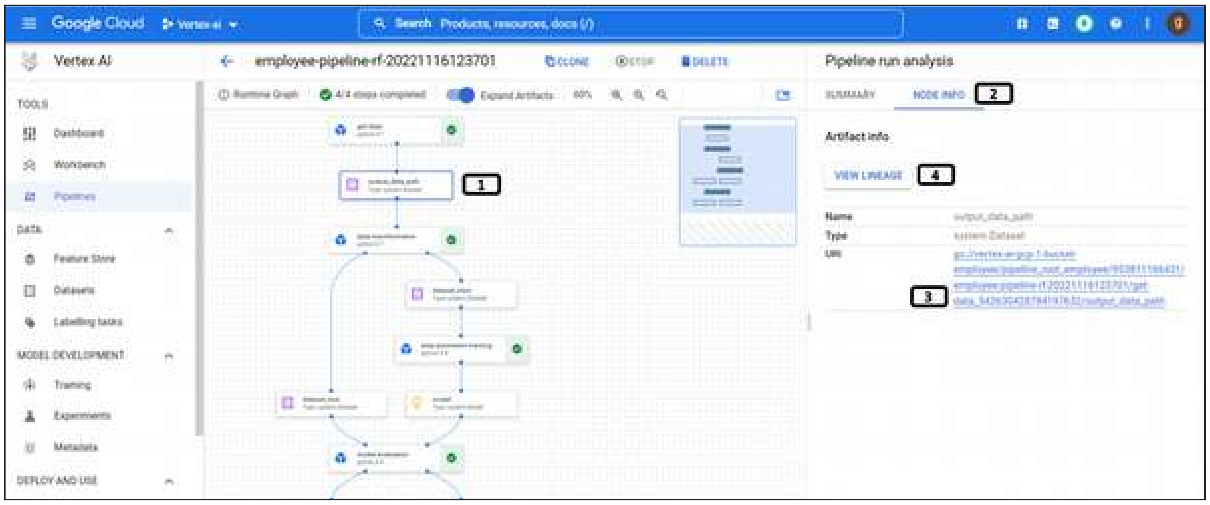


Figure 7.12: Components and artifacts of the pipeline

1. Boxes with a tick on it indicates the components of the pipeline. Others are the artifacts of the pipeline. Click on it.
2. Click **NODE INFO**.
3. Location of the cloud storage where the artifacts are stored.
4. Click **VIEW LINEAGE**.

Step 3: Lineage of the pipeline

The artifacts and parameters of a pipeline run are recorded in Vertex ML Metadata when you use Vertex AI Pipelines to execute the pipeline. By relieving users from the burden of maintaining pipeline's information, Vertex ML Metadata facilitates analysis of the items' provenance. Everything that went into making an artifact, as well as any derivative objects or information, are all part of its lineage.

Lineage for the pipeline will open in a separate tab as shown in *Figure 7.13*:

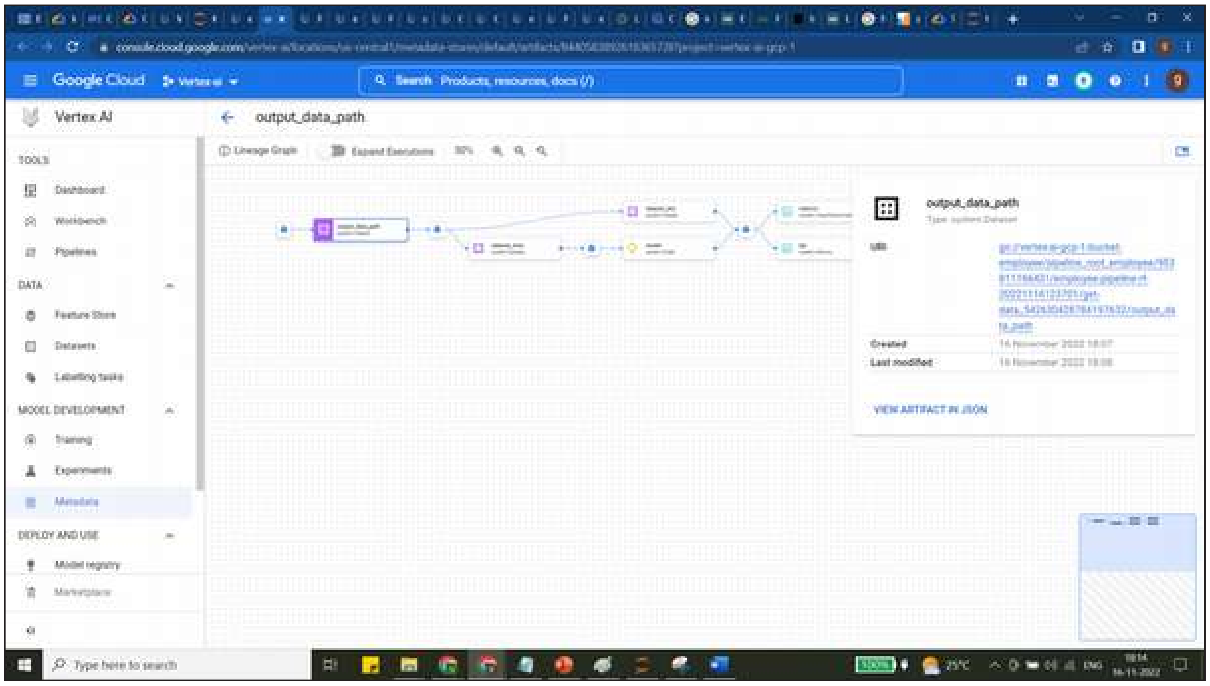


***Figure 7.13**: Lineage of the pipeline*

Step 4: Metrices and KPIs recorded in the pipeline

Metrices of trained model will be available in the artifacts as shown in *Figure 7.14*:

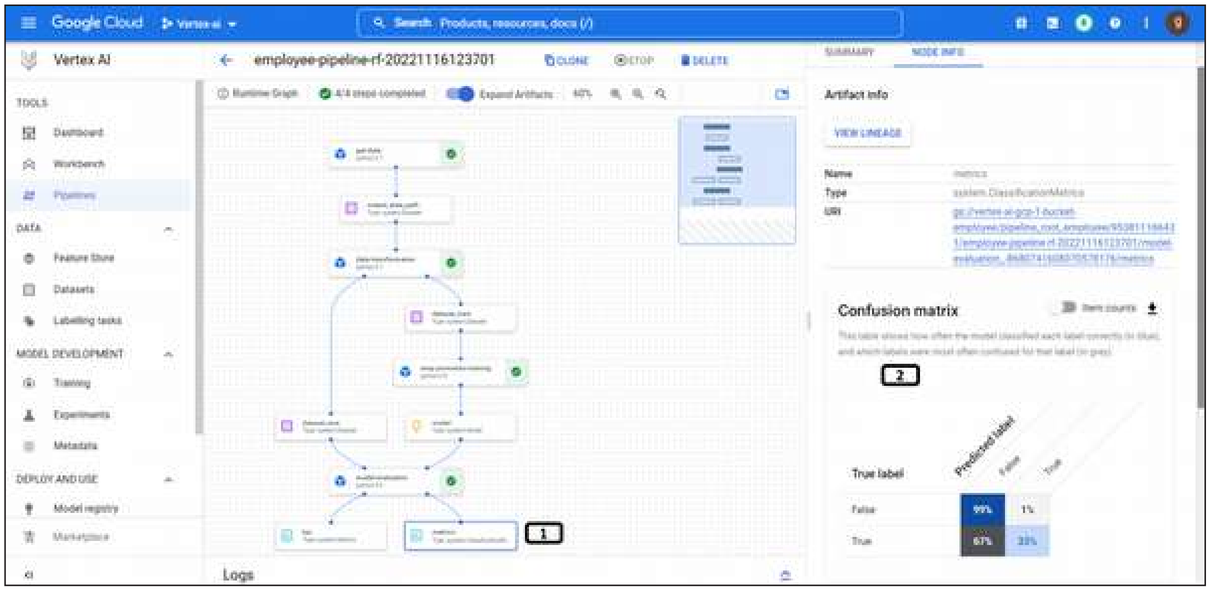


Figure 7.14: *Metrics of the model*

1. Click on the metrices artifacts of the pipeline.
2. In the last component of the pipeline, we have recorded **Confusion matrix** and ROC curve (scroll down to see it) are displayed.

KPIs of trained model will be available in the artifacts as shown in *Figure 7.15*:

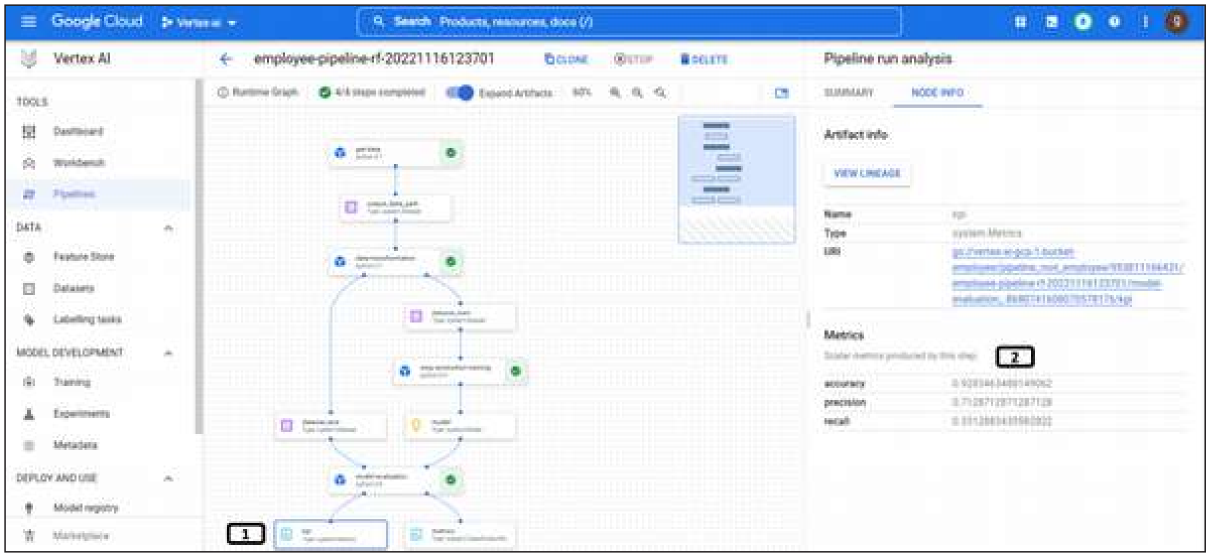
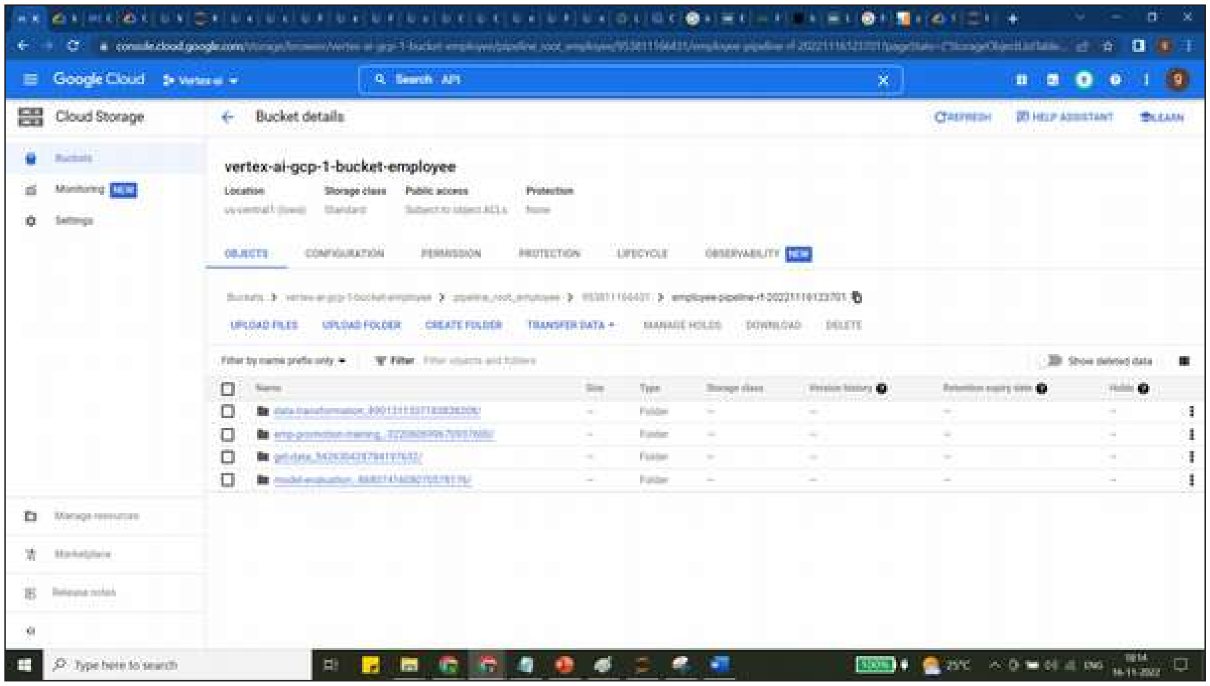


Figure 7.15: *KPIs of the model*

1. Click the **kpi** artifacts of the pipeline.
2. In the last component of the pipeline, we have recorded accuracy and other KPIs of the model which are displayed.

Step 5: Pipeline artifacts stored in cloud storage

Pipeline artifacts are stored in the cloud storage as shown in the following figure. Separate folders are created for the artifacts from different components of the pipeline:


Bucket details

vertex-ai-gcp-1-bucket-employee

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Subject to object ACLs	None

Buckets > vertex-ai-gcp-1-bucket-employee > pipeline_root_employee > [illegible] > employee-pipeline-rf-20221116123701

Name	Size	Type	Storage class	Version history	Retention expiry date	Holds
data-transformation_[illegible]/	—	Folder	—	—	—	—
emp-promotion-training_[illegible]/	—	Folder	—	—	—	—
get-data_[illegible]/	—	Folder	—	—	—	—
model-evaluation_[illegible]/	—	Folder	—	—	—	—

***Figure 7.16**: Artifacts stored in the cloud storage*

Pipeline comparison

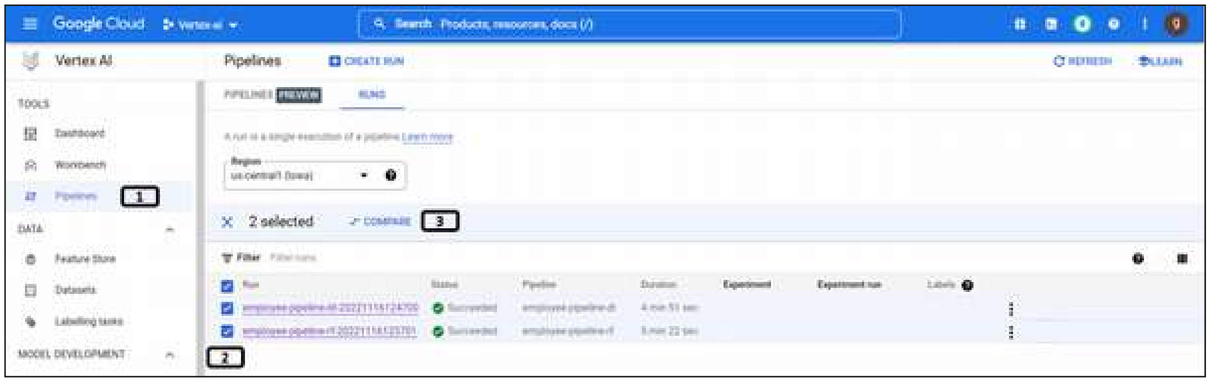
Users may run multiple pipelines with different models or with a different sample of data. GCP provides users an option to compare the performance of different pipelines.

In this exercise, we had built a pipeline to train random forest classifier model. Change random forest to any other model of your choice and run the pipeline (decision tree classifier can be used in this scenario with minimal changes in the code). Import the required package and make necessary changes in the code (at third and fourth component of the pipeline). Change the name of the JSON file, and the pipeline name. Compile and submit the pipeline job to execute.

Once the pipeline is executed successfully follow these steps to compare the pipelines.

Step 1: Pipelines module of Vertex AI

All the pipelines executed will be listed under pipeline section as shown in *Figure 7.17*:



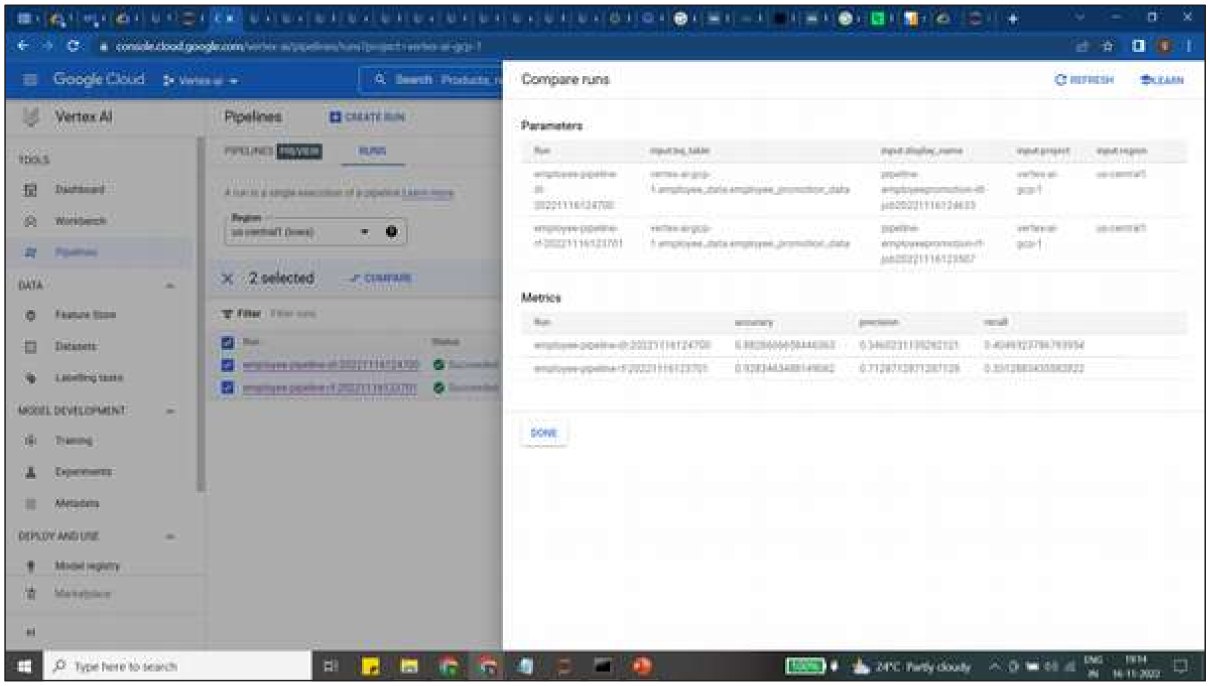
Run	Status	Pipeline	Duration	Experiment	Experiment run	Labels
employee-pipeline-dt-20221116124700	Succeeded	employee-pipeline-dt	4 min 51 sec			
employee-pipeline-rf-20221116123701	Succeeded	employee-pipeline-rf	5 min 22 sec			

Figure 7.17: *Pipeline module of Vertex AI*

1. Go to the **Pipelines** module of Vertex AI.
2. Select the pipeline to be compared.
3. Click **COMPARE**.

Step 2: Pipeline comparison

Using the pipeline comparison, users can compare the parameters of the pipeline used, the output metrics and KPIs of the pipelines as shown in *Figure 7.18*:



Parameters

Run	input.bq_table	input.display_name	input.project	input.region
employee-pipeline-dt-20221116124700	vertex-ai-gcp-1.employee_data.employee_promotion_data	pipeline-employeepromotion-dt-job20221116124633	vertex-ai-gcp-1	us-central1
employee-pipeline-rf-20221116123701	vertex-ai-gcp-1.employee_data.employee_promotion_data	pipeline-employeepromotion-rf-job20221116123507	vertex-ai-gcp-1	us-central1

Metrics

Run	accuracy	precision	recall
employee-pipeline-dt-20221116124700	0.8828666658446363	0.3460231139282121	0.4049323796793934
employee-pipeline-rf-20221116123701	0.9283443488149082	0.7128712871287128	0.3512883435582822

Figure 7.18: *Pipeline comparison*

Deletion of resources

We have utilized workbench, and BigQuery to store the data and cloud storage to store the artifacts of the pipeline. Ensure to delete the workbench, clear the data stored in the cloud storage and BigQuery.

Differences between Vertex AI and Kubeflow pipelines

Few important differences between Vertex AI pipelines and Kubeflow pipelines are listed below:

- **Domain-Specific Language (DSL) versions:** Pipelines created using TFX v0.30.0 or later and Kubeflow Pipelines SDK v2 domain-specific language (DSL) can be executed on the Vertex AI pipelines. Only the Kubeflow Pipelines SDK are executed using Kubeflow pipelines.
- **Storage:** Kubernetes resources like persistent volume claims can be used in Kubeflow Pipelines, whereas in Vertex AI cloud storage is used for data storage (Cloud Storage FUSE is used for mounting to the components)
- **Recursion:** Pipeline components that are called recursively are not supported by Vertex AI whereas Kubeflow supports it.

Conclusion

We learnt how to construct pipeline using custom model using Kubeflow SDK, compile and submit the pipeline job. The use of lineage in the pipeline, comparing different pipelines and its uses were also covered in this chapter.

In the next chapter, we will construct the pipeline using **TensorFlow Extended (TFX)** for custom model training.

Questions

1. What changes are needed if we have to execute the pipeline using xgboost/ pytorch /TensorFlow?
2. Why is Pipeline comparison helpful?
3. How are the artifacts and the parameters of the pipeline tracked?

CHAPTER 8
Pipelines using TensorFlow Extended

Introduction

In the previous chapter we worked on the pipelines of GCP using Kubeflow and built the first pipeline for the custom model training. In this chapter, we will start understanding the TFX and its components, and how to use these components for custom model training.

Structure

In this chapter, we will discuss the following topics:

- What is **TensorFlow Extended** (**TFX**)
- TFX Pipelines
- Components of TFX
- Data for pipeline building
- Pipeline code walk through
- Deletion of resources

Objectives

By the end of this chapter, users will be able to build TFX pipelines using standard components and execute the pipeline using Kubeflow.

What is TensorFlow Extended

It is undoubtedly challenging to put machine learning and deep learning models into production. Even though it produces additional value for a business, it is more time-consuming and prone to failure than the modeling itself. A model has to be maintained after it has been deployed. We must evaluate its functionality, the caliber of freshly produced data, and the suitability of the supporting infrastructure. The model should be retrained as necessary. To be long-term viable, the entire process must be automated with as little human involvement as feasible. Additionally, we must ensure that the model we are using is dependable, consistent, secure, and maybe scalable. Google officially launched its in-house platform TensorFlow Extended in early 2019 to assist businesses in implementing an industrial-grade end-to-end production system.

TFX is intended to be an end-to-end ML platform that is strong and versatile. It is built on the **TensorFlow** (**TF**) packages, which let Python programmers create custom functions. TFX encapsulates the whole TensorFlow ecosystem (from Keras to Data Validation to Serving) to enable users to construct and operate scalable and effective Machine Learning pipelines, as opposed to just allowing them to serve a model in a highly performant manner. It is structured as a series of adaptable components that need users to write less code.

TFX Pipelines

A TFX pipeline is an implementation of a ML process that is portable and can be performed on a variety of orchestrators, such as Kubeflow Pipelines, Apache beam, and Apache Airflow. The building blocks of a pipeline are called component and input parameters.

The outputs of component instances are artifacts, while the inputs of component instances often consist of artifacts that were made by upstream component instances. **Directed Acyclic Graphs** (**DAG**) are created using artifacts which determines the execution order of the pipeline.

Components of TFX

Pipelines break down the machine learning workflow into a series of components, with each component being responsible for a certain stage in the ML process. Standard components and custom components are both offered by TFX as separate groups. Users can construct pipelines with only a few standard components as well.

ML process may be expanded with the help of custom components. Constructing components are customized to match users' requirements, such as absorbing data from a closed-source system, applying data augmentation, sampling, integrating tools developed in languages other than Python into your machine learning process, such as data analysis with R, etc.

Components in TFX are made up of a component specification and an executor class, both of which are contained within a component interface class. The input and output contracts for a component are defined by the component's specification. This contract outlines the input and output artifacts of the component along with the parameters that are applied during the process of executing the component. The executor class of a component is responsible for implementing the tasks that the component is responsible for carrying out. Specification and the executor are combined into an interface class to use the component in the TFX pipelines.

The execution of the component takes place in three steps:

1. **Driver**: makes decisions about what needs to be done based on the metadata and coordinates the job execution
2. **Executor**: does the actual work to complete the job code contributed by users to complete the current project at hand
3. **Publisher**: is responsible for gathering the results of the executor and bringing the metadata database up to date.

If the standard functionality is all that is required, then we do not need to make any changes to the code that makes up the driver or the publisher. Simply extending the executor is all that is required of us to make modifications to the executor while maintaining the same inputs, outputs, and execution attributes. It is possible to write a completely bespoke executor to achieve an entirely distinct set of functionalities.

Types of custom components

There are three kinds of custom components: based on Python functions, based on containers, and the ones that are fully custom:

- **Python functions:** Building Python components that are based on functions is easier than building components that are based on containers or that are made from scratch. Type annotations show whether an argument is an input artifact, an output artifact, or a parameter. These annotations are in the Python function's arguments.
- **Container based:** Container-based components let you add code written in any language to your pipeline, as long as that code can be run in a Docker container.
- **Fully custom:** You can build fully custom components by defining the classes for the component specification, the component executor, and the component interface.

There are libraries and pipeline components in TFX. The following figure shows how TFX libraries and pipeline components relate to each other:

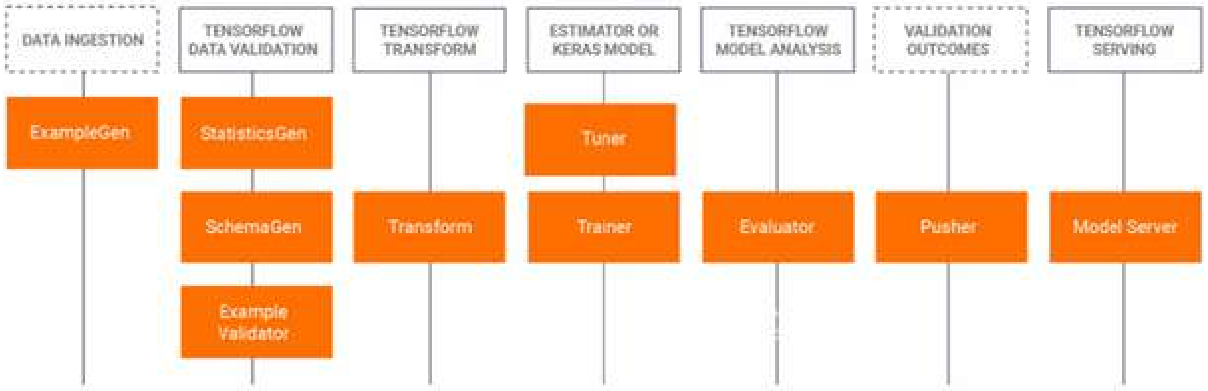


***Figure 8.1**: Libraries and components of TFX* [1]

Functionalities of custom components

Let us understand the functionalities of these components.

- **Examplegen**: The ExampleGen TFX pipeline component acts as the entry point and ingests data. ExampleGen scan automatically consumes external data sources, including CSV, TF Records, Avro, and Parquet, when used as inputs. ExampleGen generates TF examples, also known as TF sequence examples, as its outputs. These examples are particularly effective in

1 Image source: https://www.tensorflow.org/tfx/guide

performing data set representations and can be read reliably by subsequent components.

- **StatisticsGen**: Other pipeline components can make use of the statistics generated by the StatisticsGen TFX pipeline component. These statistics may be generated over both training data and serving data.
- **SchemaGen**: A description of the data that you enter, known as a schema, is used by the TFX components. An instance of the `schema.proto` class is being used here. It can specify the data types for feature values, the ranges of acceptable values, whether a feature must be present in all samples, and other features. In addition to this, a component of the SchemaGen pipeline will automatically construct a schema by deducing types, categories, and ranges from the training data.
- **ExampleValidator**: A component of the pipeline called ExampleValidator looks for irregularities in the data used for training and serving. It can identify several categories of abnormalities within the data. In addition to that, it can:
    - By comparing data statistics to a schema that details the user's expectations, validity checks can be carried out.
    - Compares the data from training and serving to look for any differences between the two.
    - Examining a data series can allow you to identify any instances of data drift.
- **Transform**: The ExampleGen component is responsible for emitting data artifacts, and the Transform TFX pipeline component is responsible for performing feature engineering on those artifacts. Transform component imports the data schema artifact from external sources or using the data schema artifact that was generated by SchemaGen.
- **Trainer**: A component of a pipeline called Trainer TFX is responsible for training a TensorFlow model. The trainer component generates at least one model for inference and stores it in a format that is compatible with TensorFlow saved models. A trustworthy model will include an entire TensorFlow program, with all its weights and computations included.
- **Tuner**: The Tuner component is responsible for fine-tuning the model's hyperparameters. The Tuner component is the most recent addition to the TFX effects component family and relies heavily on the Python Keras tuner API to tune hyperparameters. The transformed data is provided in the form of transform graph artifacts for the tuner component to take in as inputs, and the tuner component produces a hyperparameter artifact as an output.

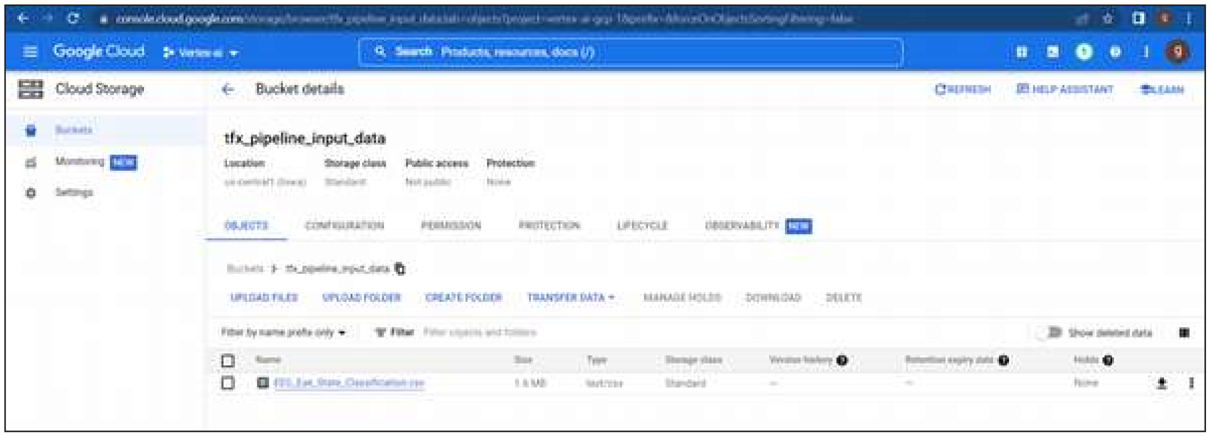
- **Evaluator**: The model that was developed by the trainer will be used by the evaluator component which will use the original input data artifact. In addition to this, it will conduct a comprehensive analysis by making use of the TensorFlow model analysis library.
- **InfraValidator**: Before sending a model into production, the InfraValidator, which is a TFX component, serves as an early warning layer to catch any potential issues. The validation of the model takes place within the actual infrastructure that serves the model, which is where the name InfraValidator comes from. If the evaluator guarantees that the model will perform as expected, then InfraValidator will also guarantee that the model will not have any mechanical issues.
- **Pusher**: During the process of model training or re-training, the Pusher component is used to send a model that has been validated to a deployment target.

Data for pipeline building

For this exercise data is downloaded from Kaggle (link is provided below) and the dataset is listed under CC0:Public domain licenses. The data contains various measurements from EEG and the state of the eye is captured via camera. 1 indicates closed eye and 0 indicates open eye.

https://www.kaggle.com/datasets/robikscube/eye-state-classification-eeg-dataset

tfx_pipeline_input_data bucket is created under **us-centra1** (single region) and csv file is uploaded from to the bucket as shown in *Figure 8.2*:


Bucket details

tfx_pipeline_input_data

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

Name	Size	Type	Storage class	Version history	Retention expiry date	Holds
EEG_Eye_State_Classification.csv	1.6 MB	text/csv	Standard	—	—	None

Figure 8.2: Data in GCS for pipeline construction

Pipeline code walk through

Workbench needs to be created for run the pipeline code. Follow the steps mentioned in the chapter Vertex AI workbench & custom model training., for creation of the workbench (choose **TensorFlow enterprise | TensorFlow Enterprise 2.9 | Without GPUs**, refer *Figure 8.3* for reference. All other steps will be the same as mentioned in the Vertex AI workbench and custom model training)

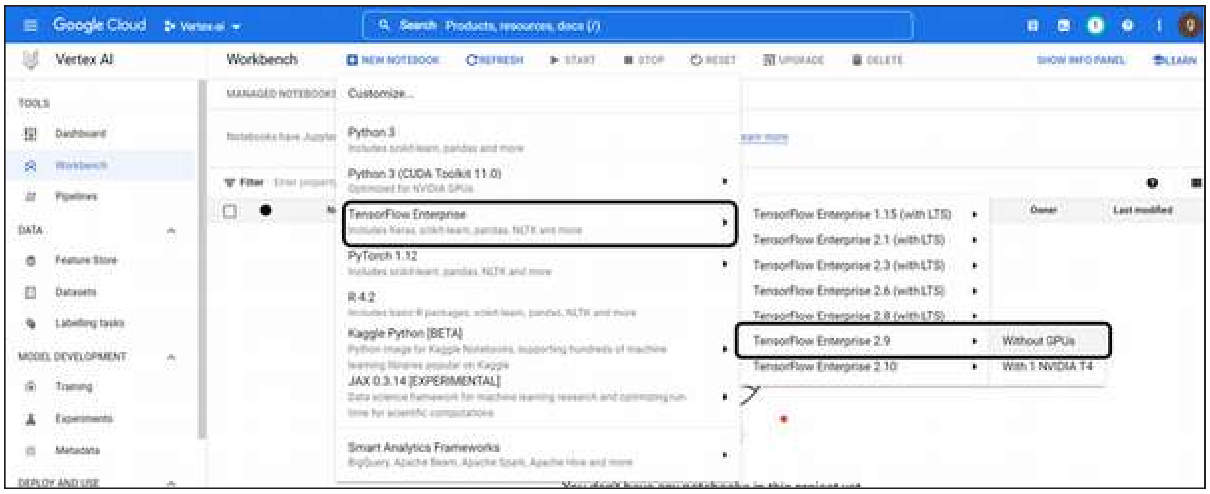

Google Cloud
Vertex AI
Workbench
NEW NOTEBOOK
REFRESH
START
STOP
RESET
UPGRADE
DELETE
SHOW INFO PANEL
LEARN
Customize...
Python 3
Python 3 (CUDA Toolkit 11.0)
TensorFlow Enterprise
PyTorch 1.12
R 4.2
Kaggle Python [BETA]
JAX 0.3.14 [EXPERIMENTAL]
Smart Analytics Frameworks
TensorFlow Enterprise 1.15 (with LTS)
TensorFlow Enterprise 2.1 (with LTS)
TensorFlow Enterprise 2.3 (with LTS)
TensorFlow Enterprise 2.6 (with LTS)
TensorFlow Enterprise 2.8 (with LTS)
TensorFlow Enterprise 2.9
TensorFlow Enterprise 2.10
Without GPUs
With 1 NVIDIA T4
Dashboard
Workbench
Pipelines
Feature Store
Datasets
Labelling tasks
Training
Experiments
Metadata

Figure 8.3: Workbench creation using TensorFlow enterprise

Step 1: Create Python notebook file

Once the workbench is created, open **Jupyterlab** and follow the steps mentioned in *Figure 8.4* to create Python notebook file:

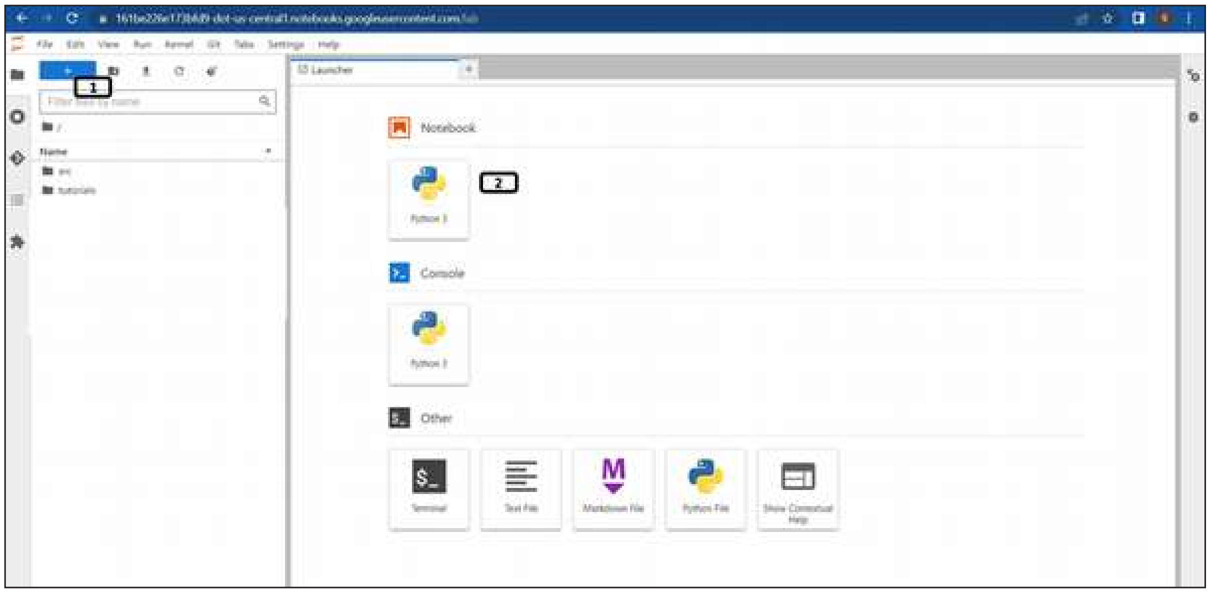


Figure 8.4: New launcher window

1. Click New launcher.
2. Double click on the **Python3 Notebook** file to create one.

Step 2 onwards, run the following codes in separate cells.

Step 2: Package installation

Run the following commands to install the Kubeflow, google cloud pipeline and google cloud aiplatform package. (It will take few minutes to install the packages):

```
USER_FLAG = "--user"

!pip install {USER_FLAG} --upgrade "tfx[kfp]<2"

!pip install {USER_FLAG} apache-beam[interactive]

!pip install python-snappy
```

Step 3: Kernel restart

Type the following commands in the next cell, to restart the kernel. (Users can restart kernel from the GUI as well):

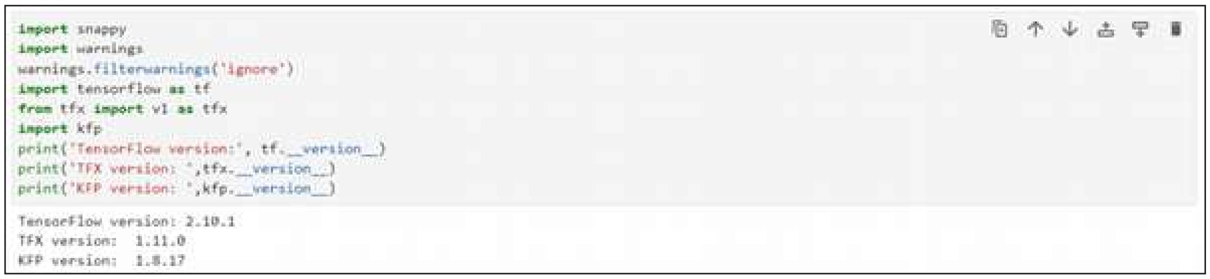
```
import os

import IPython

if not os.getenv(""):

    IPython.Application.instance().kernel.do_shutdown(True)
```

Step 4: Verify packages are installed

Run the following mentioned lines of code to check if the packages are installed (if the packages are installed properly try upgrading the pip package before installing **tfx** and **kfp** packages):

```
import snappy

import warnings

warnings.filterwarnings('ignore')

import tensorflow as tf

from tfx import v1 as tfx

import kfp

print('TensorFlow version:', tf.__version__)
print('TFX version: ',tfx.__version__)

print('KFP version: ',kfp.__version__)
```

If the packages are installed properly, you should see the versions of TensorFlow, TensorFlow extended and Kubeflow package versions as shown in *Figure 8.5*:



```
import snappy
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
from tfx import v1 as tfx
import kfp
print('TensorFlow version:', tf.__version__)
print('TFX version: ',tfx.__version__)
print('KFP version: ',kfp.__version__)
```

```
TensorFlow version: 2.10.1
TFX version:  1.11.0
KFP version:  1.8.17
```

***Figure 8.5**: Packages installed successfully*

Step 5: Setting up the project and other variables

Run the following mentioned line of codes in a new cell to set the project to the current one, also define variables to store the path for multiple purpose:

```
PROJECT_ID="vertex-ai-gcp-1"

!gcloud config set project {PROJECT_ID}

BUCKET_NAME="tfx_pipeline_demo"

NAME_PIPELINE = "tfx-pipeline"

ROOT_PIPELINE = f'gs://{BUCKET_NAME}/root/{NAME_PIPELINE}'

MODULE_FOLDER = f'gs://{BUCKET_NAME}/module/{NAME_PIPELINE}'

OUTPUT_MODEL_DIR=f'gs://{BUCKET_NAME}/output_model/{NAME_PIPELINE}'

INPUT_DATA_DIR = 'gs://tfx_pipeline_input_data'
```

`ROOT_PIPELINE` is used to store the artifacts of the pipeline, **`MODULE_FOLDER`** is used to store the **`.py`** file for the trainer component, **`OUTPUT_MODEL_DIR`** is used to store the trained model and **`INPUT_DATA_DIR`** is the GCS location where input data is located.

Step 6: Importing packages

Run the following mentioned line of codes to import required packages:

```
import tensorflow as tf

import tensorflow_transform as tft
```

```
from tensorflow import keras

from tensorflow_transform.tf_metadata import schema_utils

from tfx import v1 as tfx

from tfx_bsl.public import tfxio

from tfx.components.base import executor_spec

from      tfx.orchestration.experimental.interactive.interactive_context
import InteractiveContext

from tensorflow_metadata.proto.v0 import schema_pb2

import os

from typing import List
```

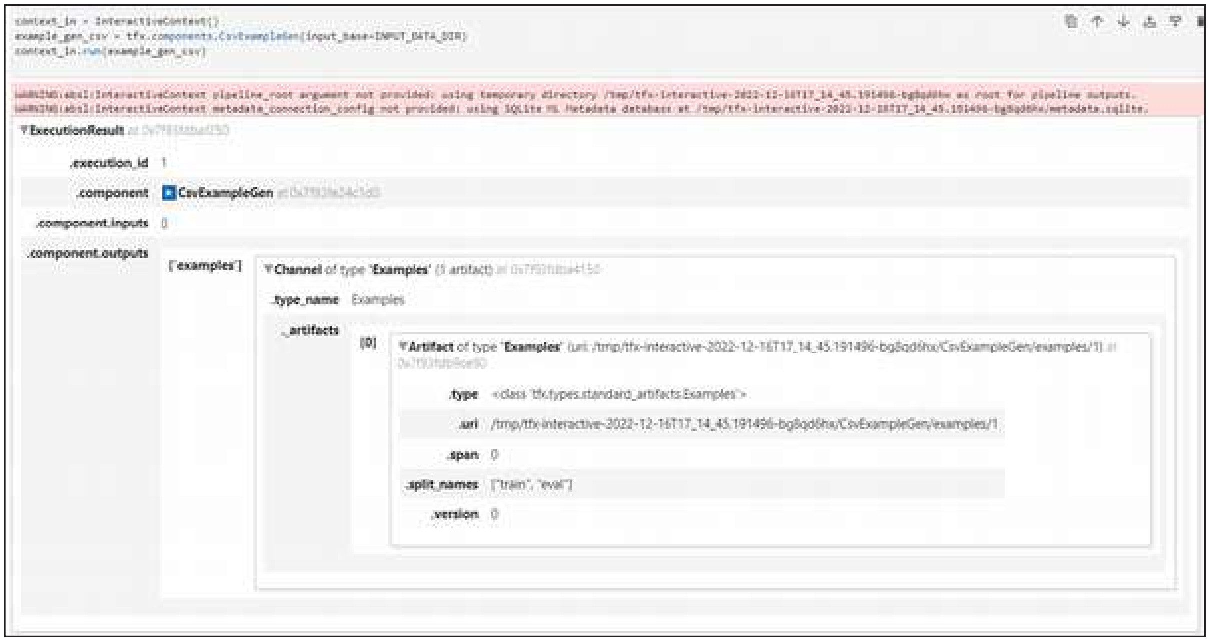
Step 7: Understanding a few TFX components from code

Even before jumping into pipeline creation and execution, let us try to understand how a few of the TFX components can be used individually and analyze the output of those components. Let us start with the data **`Examplegen`** component.

Examplegen: Run the following code in a new cell:

```
context_in = InteractiveContext()

example_gen_csv   =   tfx.components.CsvExampleGen(input_base=INPUT_DATA_
DIR)

context_in.run(example_gen_csv)
```

`Examplegen` component can read the data from various sources and data types such as CSV files, TRF records, and BigQuery. An interactive widget displaying the results of ExampleGen will appear on the notebook once run is complete as shown in *Figure 8.6*. ExampleGen typically generates two types of artifacts, which are known as training and evaluation examples. ExampleGen will divide the data into two thirds for the training set and one third for the evaluation set by default. Location where these artifacts are stored can also be viewed as shown:



```
context_in = InteractiveContext()
example_gen_csv = tfx.components.CsvExampleGen(input_base=INPUT_DATA_DIR)
context_in.run(example_gen_csv)
```

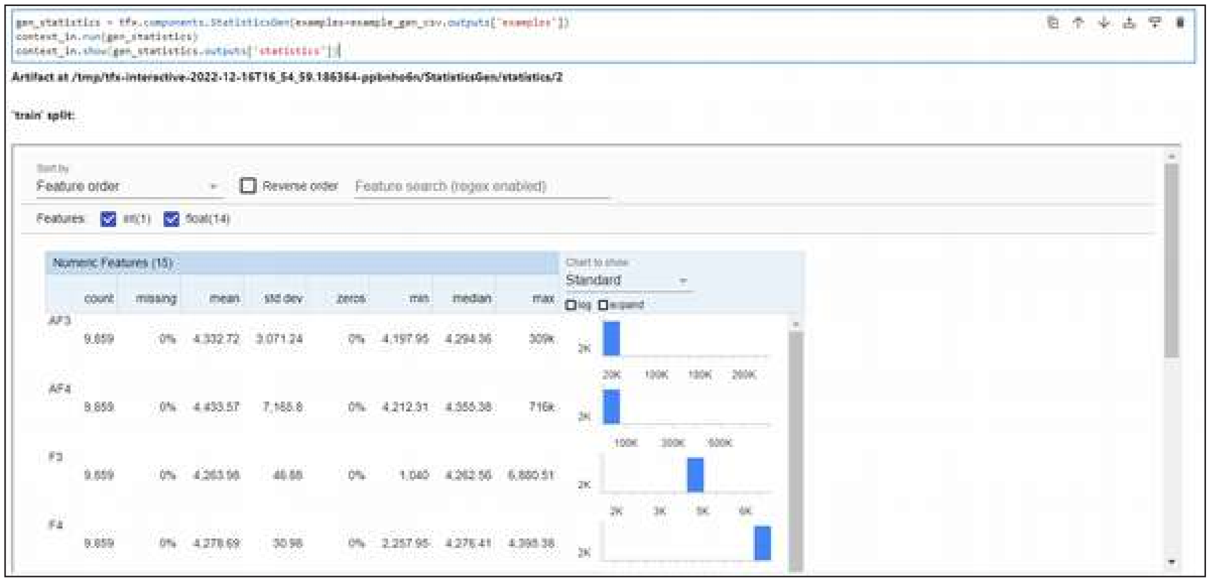
```
WARNING:absl:InteractiveContext pipeline_root argument not provided: using temporary directory /tmp/tfx-interactive-2022-12-16T17_14_45.191496-bg8qd6hx as root for pipeline outputs.
WARNING:absl:InteractiveContext metadata_connection_config not provided: using SQLite ML Metadata database at /tmp/tfx-interactive-2022-12-16T17_14_45.191496-bg8qd6hx/metadata.sqlite.
▼ExecutionResult
        .execution_id  1
           .component  CsvExampleGen
    .component.inputs  {}
   .component.outputs  ['examples']  ▼Channel of type 'Examples' (1 artifact)
                                       .type_name  Examples
                                       ._artifacts  [0]  ▼Artifact of type 'Examples' (uri: /tmp/tfx-interactive-2022-12-16T17_14_45.191496-bg8qd6hx/CsvExampleGen/examples/1)
                                                              .type  <class 'tfx.types.standard_artifacts.Examples'>
                                                               .uri  /tmp/tfx-interactive-2022-12-16T17_14_45.191496-bg8qd6hx/CsvExampleGen/examples/1
                                                              .span  0
                                                       .split_names  ["train", "eval"]
                                                           .version  0
```

***Figure 8.6**: Output for example_gen component*

StatisticsGen: Run the following code in a new cell:

```
gen_statistics = tfx.components.StatisticsGen(examples=example_gen_csv.
outputs['examples'])

context_in.run(gen_statistics)

context_in.show(gen_statistics.outputs['statistics'])
```

Statistics over your dataset are computed using the **`StatisticsGen`** component. These statistics provide a quick overview of your data, including details such as shape, features, and value distribution. You will use the output from the ExampleGen as input to compute statistics about the data. An interactive widget displaying the statistics of train and evaluation dataset separately appears once the run is complete as shown in *Figure 8.7*:



```
gen_statistics = tfx.components.StatisticsGen(examples=example_gen_csv.outputs['examples'])
context_in.run(gen_statistics)
context_in.show(gen_statistics.outputs['statistics'])
```

Artifact at /tmp/tfx-interactive-2022-12-16T16_54_59.186364-ppbnho6n/StatisticsGen/statistics/2

'train' split:

Sort by: Feature order ☐ Reverse order Feature search (regex enabled)

Features: ✓ int(1) ✓ float(14)

Numeric Features (15)

	count	missing	mean	std dev	zeros	min	median	max
AF3	9,859	0%	4,332.72	3,071.24	0%	4,197.95	4,294.36	309k
AF4	9,859	0%	4,433.57	7,165.8	0%	4,212.31	4,355.38	716k
F3	9,859	0%	4,263.96	46.88	0%	1,040	4,262.56	6,880.51
F4	9,859	0%	4,278.69	30.98	0%	2,257.95	4,276.41	4,398.38

Figure 8.7: Output for statsics_gen component

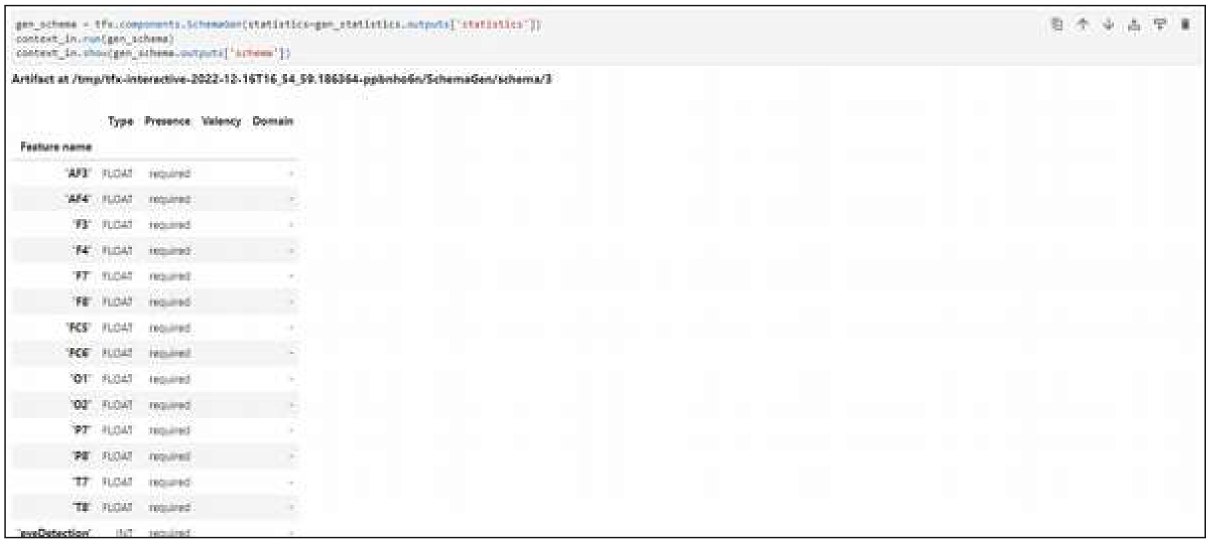
SchemaGen: Run the following code in a new cell:

gen_schema = tfx.components.SchemaGen(statistics=gen_statistics.outputs['statistics'])

context_in.run(gen_schema)

context_in.show(gen_schema.outputs['schema'])

From the statistics, the **SchemaGen** component will generate a schema for your data. A Schema is simply a data definition. It defines the data features' types, expected properties, bounds, and so on. Output of the **SchemaGen** is as shown in *Figure 8.8*:



```
gen_schema = tfx.components.SchemaGen(statistics=gen_statistics.outputs['statistics'])
context_in.run(gen_schema)
context_in.show(gen_schema.outputs['schema'])
```

Artifact at /tmp/tfx-interactive-2022-12-16T16_54_59.186364-ppbnho6n/SchemaGen/schema/3

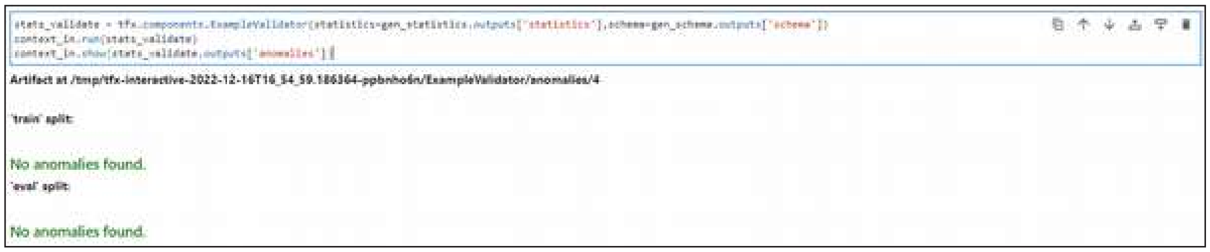
Feature name	Type	Presence	Valency	Domain
'AF3'	FLOAT	required		-
'AF4'	FLOAT	required		-
'F3'	FLOAT	required		-
'F4'	FLOAT	required		-
'F7'	FLOAT	required		-
'F8'	FLOAT	required		-
'FC5'	FLOAT	required		-
'FC6'	FLOAT	required		-
'O1'	FLOAT	required		-
'O2'	FLOAT	required		-
'P7'	FLOAT	required		-
'P8'	FLOAT	required		-
'T7'	FLOAT	required		-
'T8'	FLOAT	required		-
'eyeDetection'	INT	required		-

Figure 8.8: Output for schema_gen component

`ExampleValidator`: Run the following code in a new cell:

```
stats_validate = tfx.components.ExampleValidator(statistics=gen_
statistics.outputs['statistics'],schema=gen_schema.outputs['schema'])

context_in.run(stats_validate)

context_in.show(stats_validate.outputs['anomalies'])
```

Based on the defined schema, this component validates your data and detects anomalies. When in production, this can be used to validate any new data that enters your pipeline. It can detect drift, changes, and skew in new data, unexpected types, new column which was not in the schema. Output of **`ExampleValidator`** is as shown in *Figure 8.9*:



```
stats_validate = tfx.components.ExampleValidator(statistics=gen_statistics.outputs['statistics'],schema=gen_schema.outputs['schema'])
context_in.run(stats_validate)
context_in.show(stats_validate.outputs['anomalies'])
```

```
Artifact at /tmp/tfx-interactive-2022-12-16T16_54_59.186364-ppbnho6n/ExampleValidator/anomalies/4

'train' split:

No anomalies found.
'eval' split:

No anomalies found.
```

***Figure 8.9**: Output of example validator component*

Transform:

We will create **`file_transform.py`** which will contain information about the data labels and feature engineering steps:

- Run the following mentioned code in new cells to declare variable containing file name:

  ```
  TRANSFORM_MODULE_PATH = 'file_transform.py'
  ```

- Run the following mentioned codes in a new cell to create **`file_transform.py`** file. Preprocessing (**`preprocessing_fn`**) function is where the actual alteration of the dataset occurs. It receives and returns a tensor dictionary, where tensor refers to a Tensor or **`SparseTensor`**. In our example, we are not applying any transformations, code is just mapping to the output dictionary. (**`%%writefile`** command will create the **`.py`** file with the following code in it.):

  ```
  %%writefile {TRANSFORM_MODULE_PATH}


  import tensorflow as tf

  import tensorflow_transform as tft
  ```

```
NAMES = ['AF3','F7','F3','FC5','T7','P7','O1','O2','P8','T8','FC6
','F4','F8','AF4']
LABEL = 'eyeDetection'

def preprocessing_fn(raw_inputs):
    processed_data = dict()
    for items in NAMES:
        processed_data[items]=raw_inputs[items]
    processed_data[LABEL] = raw_inputs[LABEL]

    return processed_data
```

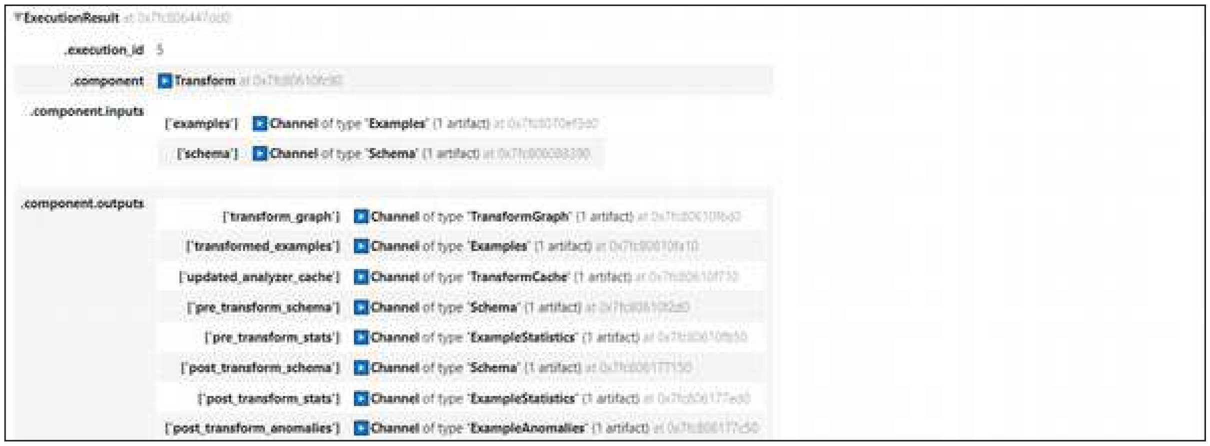
3. Files needs to copy into GCS bucket, run the following line of code to copy it:

```
!gsutil cp file_transform.py {MODULE_FOLDER}/
```

4. Run the following mentioned lines of codes in a new cell to configure transform component. Transform component is taking inputs from the **`example_gen`** and **`schema_gen`**:

```
transform_data = tfx.components.Transform(
    examples=example_gen_csv.outputs['examples'],schema=gen_
    schema.outputs['schema'],
    module_file=os.path.join(MODULE_FOLDER, TRANSFORM_MODULE_
    PATH))
context_in.run(transform_data, enable_cache=False)
```

Output of the transform component is as shown in *Figure 8.10*. Transform graph is one of the artifacts generated by the transform component and it will be used for the trainer module.



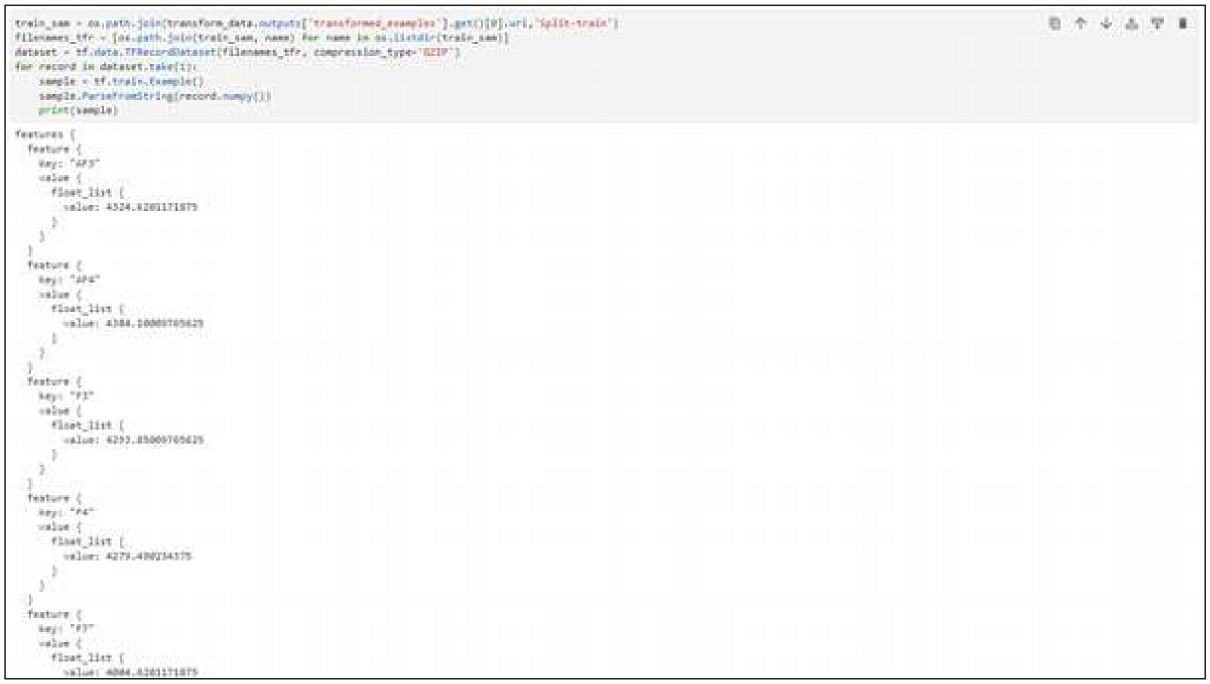
```
▾ExecutionResult at [illegible]
        .execution_id  5
          .component   Transform at [illegible]
   .component.inputs
                ['examples']   Channel of type 'Examples' (1 artifact) at [illegible]
                  ['schema']   Channel of type 'Schema' (1 artifact) at [illegible]

  .component.outputs
                 ['transform_graph']   Channel of type 'TransformGraph' (1 artifact) at [illegible]
            ['transformed_examples']   Channel of type 'Examples' (1 artifact) at [illegible]
           ['updated_analyzer_cache']  Channel of type 'TransformCache' (1 artifact) at [illegible]
            ['pre_transform_schema']   Channel of type 'Schema' (1 artifact) at [illegible]
             ['pre_transform_stats']   Channel of type 'ExampleStatistics' (1 artifact) at [illegible]
           ['post_transform_schema']   Channel of type 'Schema' (1 artifact) at [illegible]
            ['post_transform_stats']   Channel of type 'ExampleStatistics' (1 artifact) at [illegible]
        ['post_transform_anomalies']   Channel of type 'ExampleAnomalies' (1 artifact) at [illegible]
```

***Figure 8.10**: Output of transform component*

Run the following mentioned codes to check few records of the transformed data (This code will not be needed during the pipeline construction):

```
train_sam = os.path.join(transform_data.outputs['transformed_examples'].
get()[0].uri,'Split-train')

filenames_tfr = [os.path.join(train_sam, name) for name in os.listdir(train_
sam)]

dataset = tf.data.TFRecordDataset(filenames_tfr, compression_type='GZIP')

for record in dataset.take(1):

    sample = tf.train.Example()

    sample.ParseFromString(record.numpy())

    print(sample)
```

Output of 1 record will be as shown in *Figure 8.11*:



```
train_sam = os.path.join(transform_data.outputs['transformed_examples'].get()[0].uri,'Split-train')
filenames_tfr = [os.path.join(train_sam, name) for name in os.listdir(train_sam)]
dataset = tf.data.TFRecordDataset(filenames_tfr, compression_type='GZIP')
for record in dataset.take(1):
    sample = tf.train.Example()
    sample.ParseFromString(record.numpy())
    print(sample)
```

```
features {
  feature {
    key: "AF3"
    value {
      float_list {
        value: 4324.6201171875
      }
    }
  }
  feature {
    key: "AF4"
    value {
      float_list {
        value: 4384.10009765625
      }
    }
  }
  feature {
    key: "F3"
    value {
      float_list {
        value: 4293.85009765625
      }
    }
  }
  feature {
    key: "F4"
    value {
      float_list {
        value: 4279.490234375
      }
    }
  }
  feature {
    key: "F7"
    value {
      float_list {
        value: 4084.6201171875
```

Figure 8.11*: Transformed data example*

Trainer:

The Trainer component is in charge of preparing the input data and training the model. It requires the ExampleGen examples, the transform, and the training code. TensorFlow Estimators, Keras models, or custom training loops can be used in the training code. When compared to other components, trainer requires more modifications in the code:

- Run the following mentioned code in the new cell, it will generate **`trainer.py`** file (Complete code is available in the repository). **`Trainer.py`** (training code) file will be the input for the trainer module. Trainer component generates two artifacts model (trained model itself) and **`modelrun`** which can be used for storing logs, this can be seen in *Figure 8.12*. The **`Trainer.py`** file contains four functions; high level description of those functions is:
    - **`run_fn`** will be entry point to execute the training process
    - **`_input_fn`** generates features and labels for training
    - **`_get_serve_tf_examples_fn`** returns a function that parses a serialized **`tf.example`**
    - **`_make_keras_model`** creates and returns the model for classification

```
%%writefile trainer.py
from typing import List
from absl import logging
import tensorflow as tf
import tensorflow_transform as tft
from tensorflow import keras
from tensorflow_transform.tf_metadata import schema_utils
from tfx import v1 as tfx
from tfx_bsl.public import tfxio
from tensorflow_metadata.proto.v0 import schema_pb2
COL_NAMES=['AF3','F7','F3','FC5','T7','P7','O1','O2','P8','T8','FC6','F4','F8','AF4']
LABEL="eyeDetection"
BATCH_SIZE_TRAIN = 40
BATCH_SIZE_EVAL = 20
def _input_fn(files,accessor,transform_output,size) -> tf.data.Dataset:
    #Creates datasets and apply transformations on them and return. Refer the repository for code block of the function
    return dataset.map(apply_transform_fn).repeat()

def _get_serve_tf_examples_fn(model, transform_output):
    #To parse the serialized examples and return. Refer the
    repository for code block of the function

    return serve_tf_examples_fn

def _make_keras_model() -> tf.keras.Model:
    #Create model with layers, loss functions to be used and
    return. Refer the repository for code block of the function
    return model_classification

def run_fn(fn_args: tfx.components.FnArgs):
    tf_transform = tft.TFTransformOutput(fn_args.transform_output)
    train_samples = _input_fn(_args.train_files, _args.data_
    accessor,        tf_transform, =BATCH_SIZE_TRAIN)
    eval_samples = _input_fn(_args.eval_files, _args.data_
    accessor,        tf_transform, =BATCH_SIZE_EVAL)


    model_classification = _make_keras_model()
    model_classification.fit(
        train_samples,
        steps_per_epoch=fn_args.train_steps,
        validation_data=eval_samples,
        validation_steps=fn_args.eval_steps)
    sign = {
        "serving_default": _get_serve_tf_examples_fn(model_
        classification, tf_transform),
    }
    model_classification.save(fn_args.serving_model_dir, save_
    format='tf',signatures=sign)
```

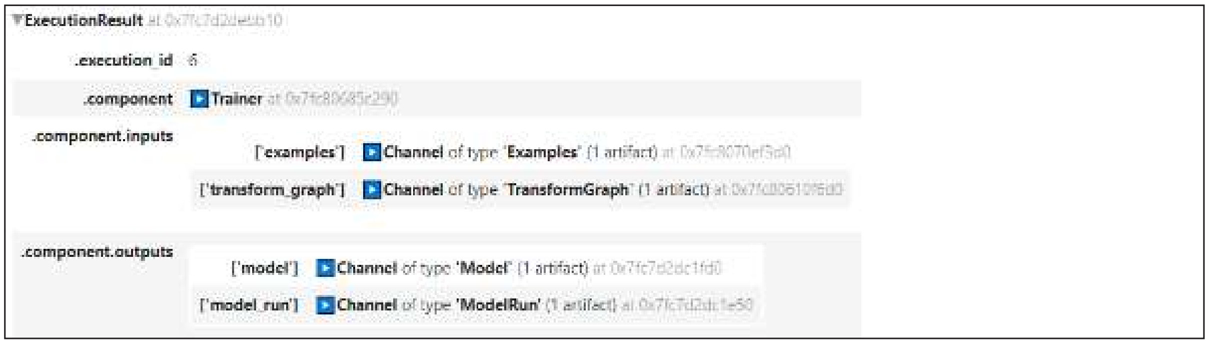
1. Run the following code to copy the trainer.py to GCS storage:

   ```
   !gsutil cp trainer.py {MODULE_FOLDER}/
   ```

2. Run the following code to initiate the trainer component. **`trainer_file="trainer.py"`**:

   ```
   trainer_file_path=os.path.join(MODULE_FOLDER, trainer_file)
   model_trainer = tfx.components.Trainer(
       examples=example_gen_csv.outputs["examples"],
       transform_graph=transform_data.outputs["transform_graph"],
       train_args=tfx.proto.TrainArgs(num_steps=200),
       eval_args=tfx.proto.EvalArgs(num_steps=10),
       module_file=trainer_file_path,
   )
   context_in.run(model_trainer, enable_cache=False)
   ```

The output of the trainer component is as shown in the following figure:



```
▼ExecutionResult at 0x7fc7d2de5b10
        .execution_id  6
           .component  Trainer at 0x7fc80685c290
    .component.inputs
                        ['examples']  Channel of type 'Examples' (1 artifact) at 0x7fc8070ef5d0
                 ['transform_graph']  Channel of type 'TransformGraph' (1 artifact) at 0x7fc80610f6d0
   .component.outputs
                           ['model']  Channel of type 'Model' (1 artifact) at 0x7fc7d2dc1fd0
                       ['model_run']  Channel of type 'ModelRun' (1 artifact) at 0x7fc7d2dc1e50
```

***Figure 8.12**: Output of the trainer component*

Step 6: Creation of pipeline

Till now we have understood and executed certain components of the **`tfx`** pipeline (not all the components are needed for the pipeline construction). Let's start constructing pipeline using **`example_gen`**, **`statistics_gen`**, **`schema_gen`**, **`transform`**, **`trainer`** and **`pusher`** components (Pusher component is used to push the trained model to a location, in this example trained model we will be pushed to the cloud storage):

- Define a function (**`_create_pipeline`**) to consisting of all the steps/components that needs to be included in the pipeline and returns the **`tfx`** pipeline. Run the below mentioned code in a new cell to define the pipeline function:

```
def _create_pipeline(pl_name, pipeline_root_folder, data_root,
                     module_file_transform, module_file_train,
                     model_dir_save,
                     ) -> tfx.dsl.Pipeline:

    example_gen_csv = tfx.components.CsvExampleGen(input_
    base=data_root)
    gen_statistics = tfx.components.
    StatisticsGen(examples=example_gen_csv.outputs['examples'])
    gen_schema = tfx.components.SchemaGen(statistics=gen_
    statistics.outputs['statistics'])
    transform_data = tfx.components.Transform(examples=example_
    gen_csv.outputs['examples'],schema=gen_schema.
    outputs['schema'],module_file=os.path.join(MODULE_FOLDER,
    TRANSFORM_MODULE_PATH))


    model_trainer = tfx.components.Trainer(
        module_file=module_file_train,
        examples=example_gen_csv.outputs['examples'],
       transform_graph=transform_data.outputs['transform_graph'],
        schema=gen_schema.outputs['schema'],
        train_args=tfx.proto.TrainArgs(num_steps=200),
        eval_args=tfx.proto.EvalArgs(num_steps=10))

    pusher = tfx.components.Pusher(
        model=model_trainer.outputs['model'],
        push_destination=tfx.proto.PushDestination(
        filesystem=tfx.proto.PushDestination.Filesystem(
        base_directory=model_dir_save)))

    return tfx.dsl.Pipeline(
        pipeline_name=pl_name,
        pipeline_root=pipeline_root_folder,
components=[example_gen_csv,gen_statistics,gen_schema,transform_
data,model_trainer,pusher])
```

Step 8: Defining a runner

As mentioned in the theory section, TFX is portable across environments and orchestration frameworks. TFX supports Airflow, Beam, and Kubeflow. It also provides flexibility for the developers to add their own orchestrators. Orchestrators must inherit TfxRunner. TFX orchestrators schedule pipeline components based on DAG dependencies using the logical pipeline object, which comprises pipeline args, components, and DAG. In our example we will be using Vertex Pipelines together with the Kubeflow V2 dag runner. In the code, we create runner using Kubeflow V2 dag and run it by passing all the pipeline parameters.

Run the following mentioned codes to define the runner:

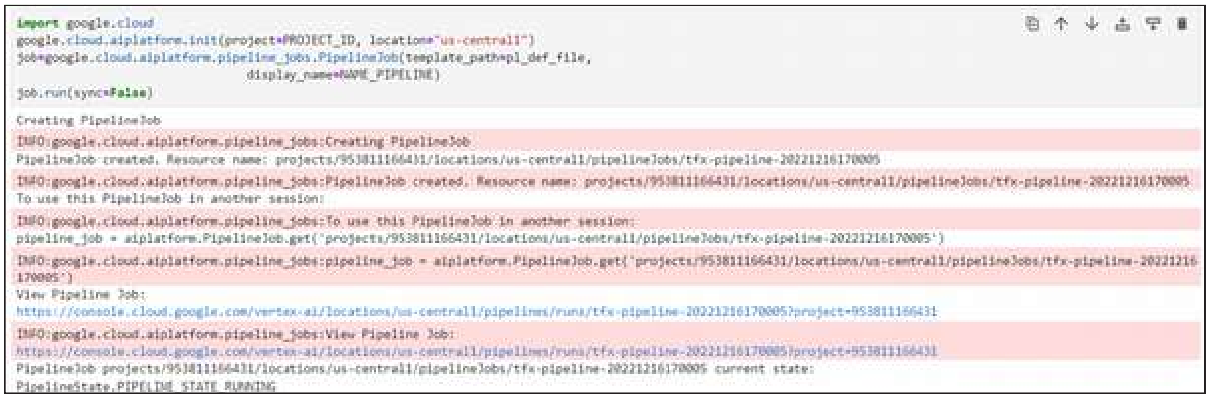
```
trainer_file=”trainer.py”

file_transform=os.path.join(MODULE_FOLDER, TRANSFORM_MODULE_PATH)

file_train=os.path.join(MODULE_FOLDER, trainer_file)

pl_def_file = NAME_PIPELINE + ‘.json’


pl_runner = tfx.orchestration.experimental.KubeflowV2DagRunner(

    config=tfx.orchestration.experimental.KubeflowV2DagRunnerConfig(),

    output_filename=pl_def_file)

_ = pl_runner.run(

    _create_pipeline(

        pl_name=NAME_PIPELINE,

        pipeline_root_folder=ROOT_PIPELINE,

        data_root=INPUT_DATA_DIR,

        module_file_transform=file_transform,

        module_file_train=file_train,

        model_dir_save=OUTPUT_MODEL_DIR))
```

Step 9: Pipeline execution

As the last step we need to execute the pipeline. Run the following mentioned codes in a new cell to start the pipeline execution:

```
import google.cloud

google.cloud.aiplatform.init(project=PROJECT_ID, location="us-central1")

job=google.cloud.aiplatform.pipeline_jobs.PipelineJob(template_path=pl_
def_file,

                                        display_name=NAME_PIPELINE)

job.run(sync=False)
```

Once the pipeline starts, users will be provided the link to check the status as shown in *Figure 8.13*:



```
import google.cloud
google.cloud.aiplatform.init(project=PROJECT_ID, location="us-central1")
job=google.cloud.aiplatform.pipeline_jobs.PipelineJob(template_path=pl_def_file,
                                    display_name=NAME_PIPELINE)
job.run(sync=False)
```

```
Creating PipelineJob
INFO:google.cloud.aiplatform.pipeline_jobs:Creating PipelineJob
PipelineJob created. Resource name: projects/953811166431/locations/us-central1/pipelineJobs/tfx-pipeline-20221216170005
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob created. Resource name: projects/953811166431/locations/us-central1/pipelineJobs/tfx-pipeline-20221216170005
To use this PipelineJob in another session:
INFO:google.cloud.aiplatform.pipeline_jobs:To use this PipelineJob in another session:
pipeline_job = aiplatform.PipelineJob.get('projects/953811166431/locations/us-central1/pipelineJobs/tfx-pipeline-20221216170005')
INFO:google.cloud.aiplatform.pipeline_jobs:pipeline_job = aiplatform.PipelineJob.get('projects/953811166431/locations/us-central1/pipelineJobs/tfx-pipeline-20221216
170005')
View Pipeline Job:
https://console.cloud.google.com/vertex-ai/locations/us-central1/pipelines/runs/tfx-pipeline-20221216170005?project=953811166431
INFO:google.cloud.aiplatform.pipeline_jobs:View Pipeline Job:
https://console.cloud.google.com/vertex-ai/locations/us-central1/pipelines/runs/tfx-pipeline-20221216170005?project=953811166431
PipelineJob projects/953811166431/locations/us-central1/pipelineJobs/tfx-pipeline-20221216170005 current state:
PipelineState.PIPELINE_STATE_RUNNING
```

***Figure 8.13**: Pipeline link*

1. Click on the link to check the status of the pipeline as shown in the following screenshot:

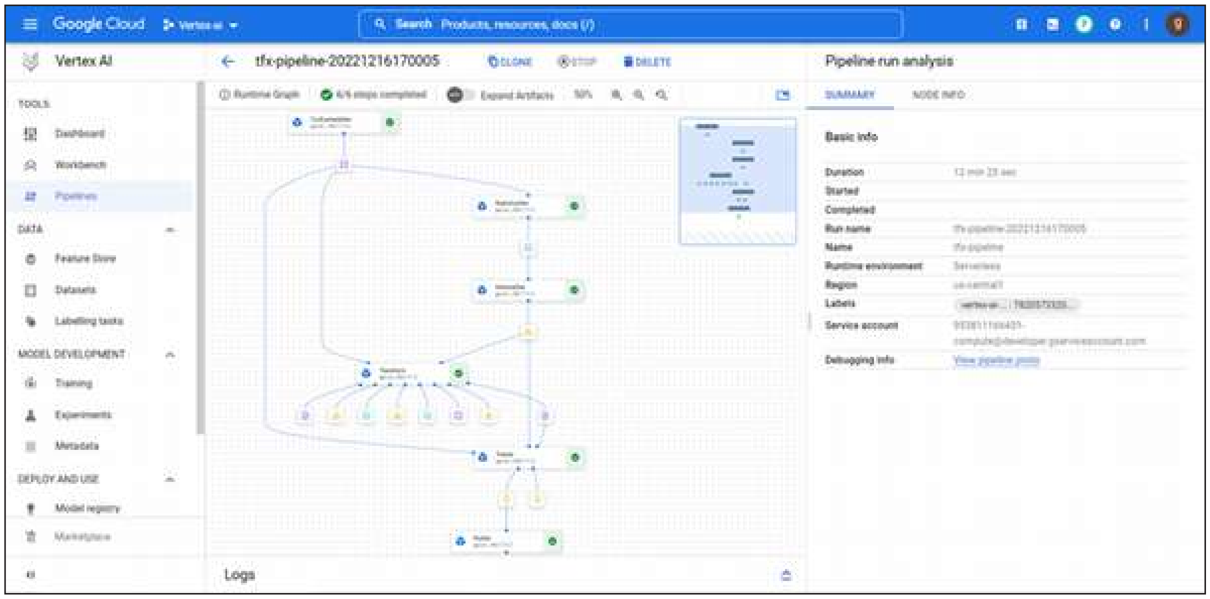


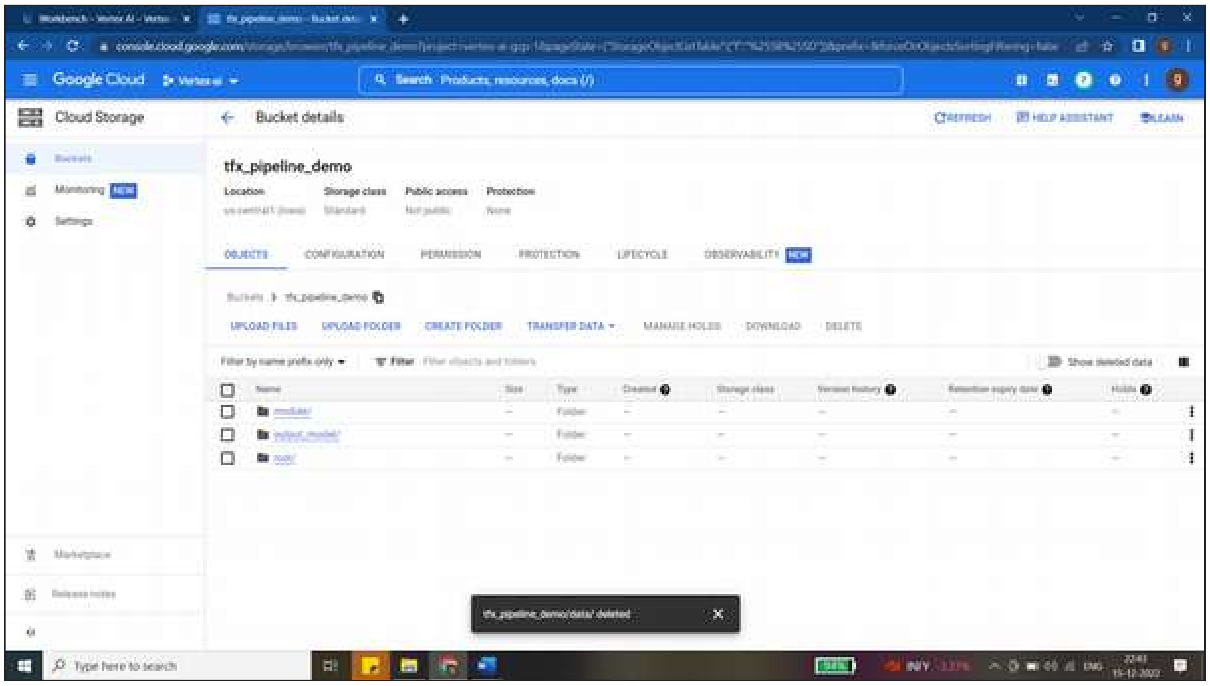
***Figure 8.14**: Pipeline executed successfully*

Pipeline will take a few mins to complete the training and push the trained model to the cloud storage. If the pipeline is constructed using tfx or Kubeflow, the user interface of the pipeline in GCP will remain same as shown in *Figure 8.14*. Spend some time to understand each of the components and its artifacts in detail as described in the previous chapter.

Step 10: Pipeline artifacts stored in cloud storage

Pipeline artifacts are stored in the cloud storage as shown in *Figure 8.15*:

- Module folder contains Python files which we had pushed while creating transform and trainer components.
- `Output_model` contains trained classification model.
- Root folder contains artifacts of each of the component of the pipeline:



tfx_pipeline_demo

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

Buckets > tfx_pipeline_demo

Name	Size	Type	Created	Storage class	Version history	Retention expiry date	Holds
module/	—	Folder	—	—	—	—	—
output_model/	—	Folder	—	—	—	—	—
root/	—	Folder	—	—	—	—	—

tfx_pipeline_demo/data/ deleted

Figure 8.15: Pipeline artifacts stored in the cloud storage

Deletion of resources

We have utilized workbench, cloud storage to store the data and the artifacts of the pipeline. For deletion of resources, ensure to delete the workbench, clear the data stored in the cloud storage.

Conclusion

We learnt about the TFX, a few of its components and constructed pipeline using some of the standard components. Also, we understood how to use Kubeflow for the orchestration of TFX pipeline on vertex AI.

In the next chapter, we will start understanding and working on feature store of the Vertex AI.

Questions

1. Which artifacts of the transform component is used in the training component of the pipeline?
2. What are the different orchestration options TFX supports?
3. Try using evaluator component between trainer and pusher component and re-construct the pipeline. (Use evaluator component to evaluate the model performance and push it only if the performance is good).

CHAPTER 9
Vertex AI Feature Store

Introduction

After learning about the pipelines of the platform, we will move to the feature store of GCP. In this chapter, we will start with an understanding of the feature store, and the advantages of features followed by a hands-on feature store.

Structure

In this chapter, we will cover the following topics:

- Knowing Vertex AI feature store
- Hierarchy of feature store
- Advantages of feature store
- Disadvantages of feature store
- Working on feature store using GUI
- Working on feature store using python
- Deleting resources
- Best practices for Feature store

Objectives

By the end of this chapter, users will have a good idea about the feature store, when to use it, and how to employ it with the web console of GCP and Python.

Knowing Vertex AI feature store

Vertex AI Feature Store is a centralized repository for managing and delivering machine learning features. To speed up the process of creating and delivering high-quality ML applications, many organizations are turning to centralized feature stores to facilitate the sharing, discovery, and re-use of ML features at scale.

The storage and processing power, as well as other components of the backend infrastructure, are handled by Vertex AI Feature Store, making it a fully managed solution. As a result of this strategy, data scientists may ignore the difficulties associated with delivering features into production and instead concentrate on the feature computation logic.

The feature store in Vertex AI is an integral aspect of the overall system. Use Vertex AI Feature Store on its own or include it in your existing Vertex AI workflows. For instance, the Vertex AI Feature Store may be queried for information to be used in the training of custom or AutoML models.

Hierarchy of feature store

The collection of entities for a certain entity time is stored in a feature store. Fields like entity ID, timestamp, and a series of attributes like feature 1, feature 2, and so on, are all defined for each entity type. The hierarchy of the feature store is described in *Figure 9.1*:

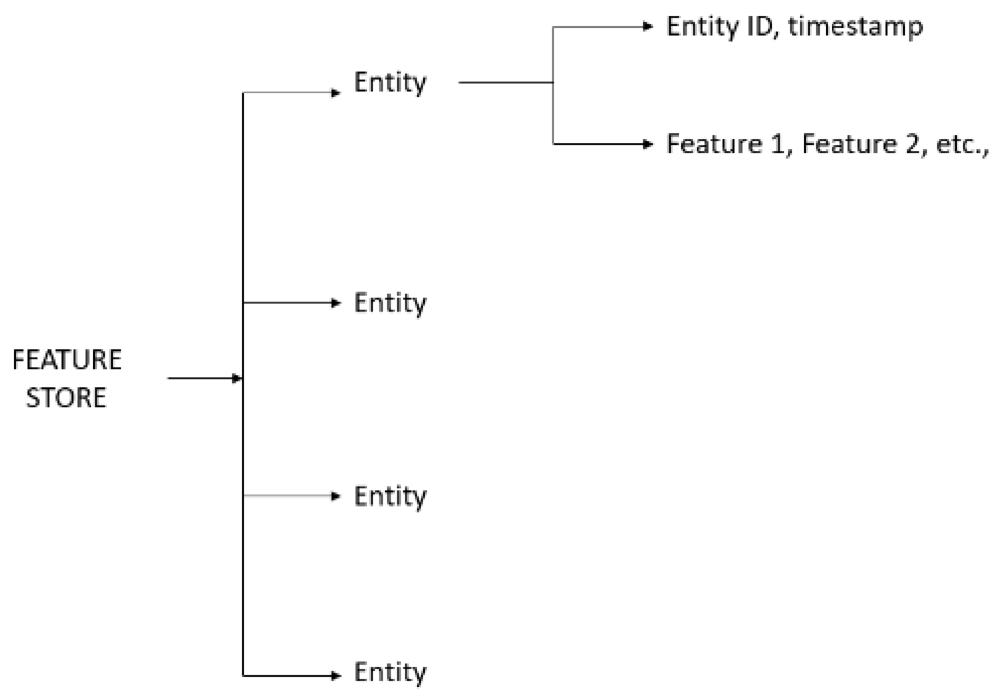


***Figure 9.1**: Hierarchy of feature store*

- **Feature store**: A top-level container for entity types, features, and their values.
- **Entity type**: A collection of semantically related features (real or virtual).
- **Entity**: An instance of the entity type.
- **Feature**: A measurable property or attribute of an entity type.
- **Feature values**: These contain values of the features at a specific point in time.

Advantages of feature store

These are the advantages of feature store:

- **Extend features company-wide:** Feature stores let you easily share features for training or serving. Different projects and use cases do not need feature re-engineering. Manage and deliver features from a central repository to preserve consistency throughout your business and prevent redundant efforts, especially for high-value features.

  Vertex AI Feature Store lets people find and reuse features using search and filtering. View feature metadata to assess quality and usage. For instance, you may check feature coverage and feature value distribution.

- **Serving at scale:** Online forecasts require low-latency feature serving, which Vertex AI Feature Store manages. Vertex AI Feature Store automatically builds and expands low-latency data serving infrastructure. You create features but outsource providing them. Data scientists may create new features without worrying about deployment using this management.
- **Reduce training-serving bias:** Training-serving skew happens when your production feature data distribution differs from the one used to train your model. This skew causes disparities between a model's training and production performance. Vertex AI Feature Store can handle training-serving bias with these examples:
    - Vertex AI Feature Store guarantees that feature values are ingested once and reused for training and serving. Without a feature store, training and serving features may use distinct code paths. Training and serving feature values may differ.
    - Vertex AI Feature Store offers previous data lookups for training. By collecting pre-prediction feature values, these lookups reduce data leakage.
- **Identify drift**: Vertex AI Feature Store detects drift in feature data distribution. Vertex AI Feature Store monitors feature value dispersion. Retrain models using impacted features as feature drift rises.
- **Retention**: The Vertex AI Feature Store preserves feature values for the allotted amount of time. This cap is determined by the feature values' timestamp, not the date and time the values were imported. Values with timestamps that go beyond the limit are scheduled for deletion by Vertex AI Feature Store.

Disadvantages of feature store

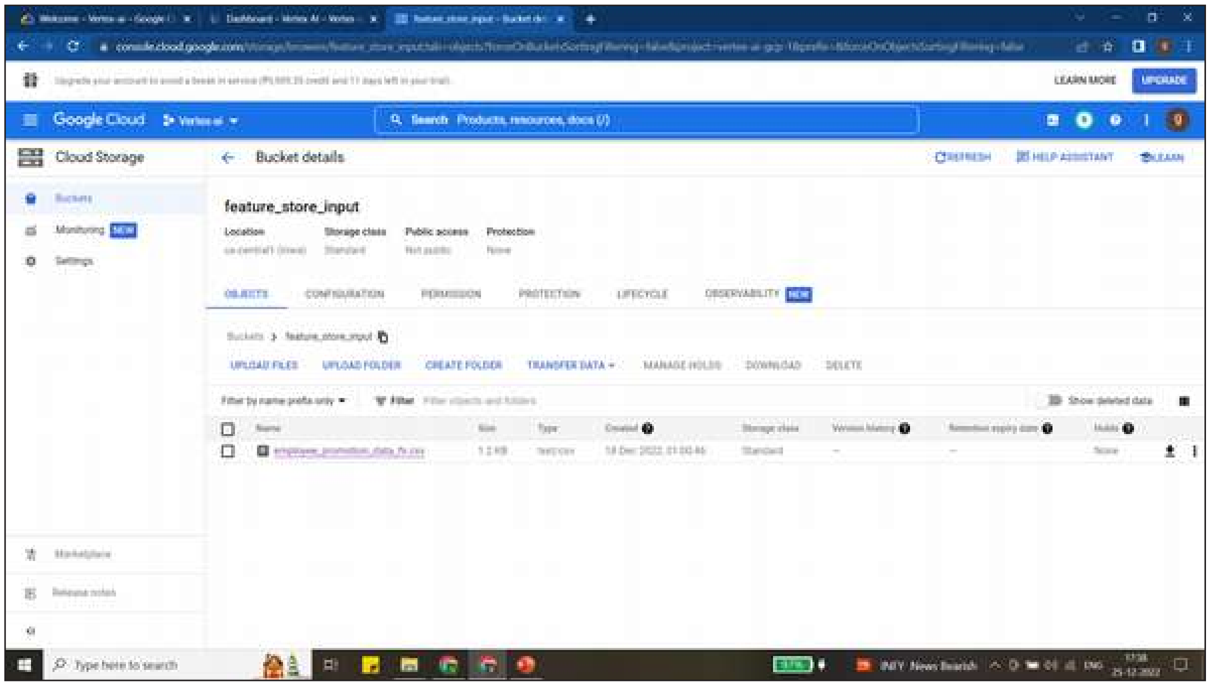
A feature store has overhead, which can make data science more complicated, especially for smaller projects. A feature store may complicate matters if a business has numerous little data sets. Feature stores are ineffective when the data is so diverse that no standard modeling approach will assist. Reusing features created on separate data sources and metadata is tough. Additionally, a feature store might not be the ideal choice when the features are not time-dependent or when features are needed only for batch predictions.

Data for feature store exercise

For this exercise, data is downloaded from Kaggle (link is provided below) and the dataset is listed under CC0 public domain licenses. Data contains information regarding employee promotion data. Since we are not building any model from the data, we are considering only 5 attributes (employee ID, education, gender, no. of trainings, and age), and only 50 samples are considered.

https://www.kaggle.com/datasets/arashnic/hr-ana

feature_store_input bucket is created under **us-centra1** (single region) and the CSV file is uploaded from the bucket as shown in Figure 9.2*:*



feature_store_input

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

Buckets > feature_store_input

Name	Size	Type	Created	Storage class	Version history	Retention expiry date	Holds
employee_promotion_data_fs.csv	1.2 KB	text/csv	18 Dec 2022, 21:00:46	Standard	–	–	None

Figure 9.2: Data stored in cloud storage

Working on feature store using GUI

Before we ingest data to the feature store or feature, we need to create a feature store, entity, and features. Resources of the feature store can be created using GUI or Python code. Follow the below-mentioned steps to create the feature store resources using GUI:

Step 1: Opening of feature store.

The landing page of the Vertex AI is shown in *Figure 9.3*:

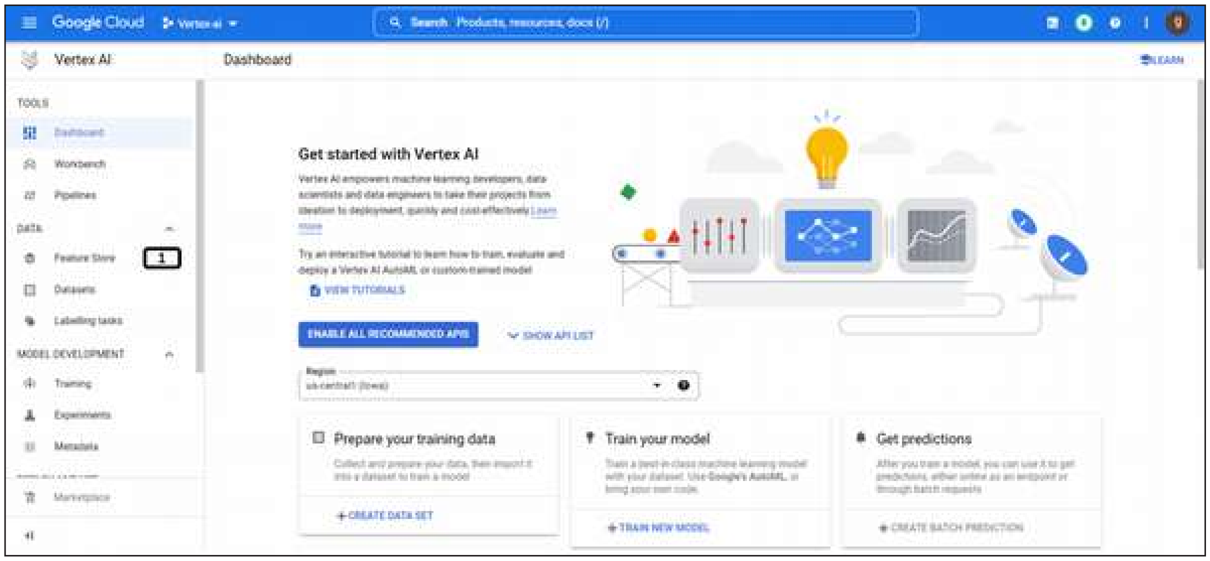


***Figure 9.3**: landing page of Vertex AI*

1. Click **Feature Store** to open.

Step 2: Landing page of feature store

Feature stores are region specific; the landing page provides information on the feature store under a specific region as shown in *Figure 9.4*:

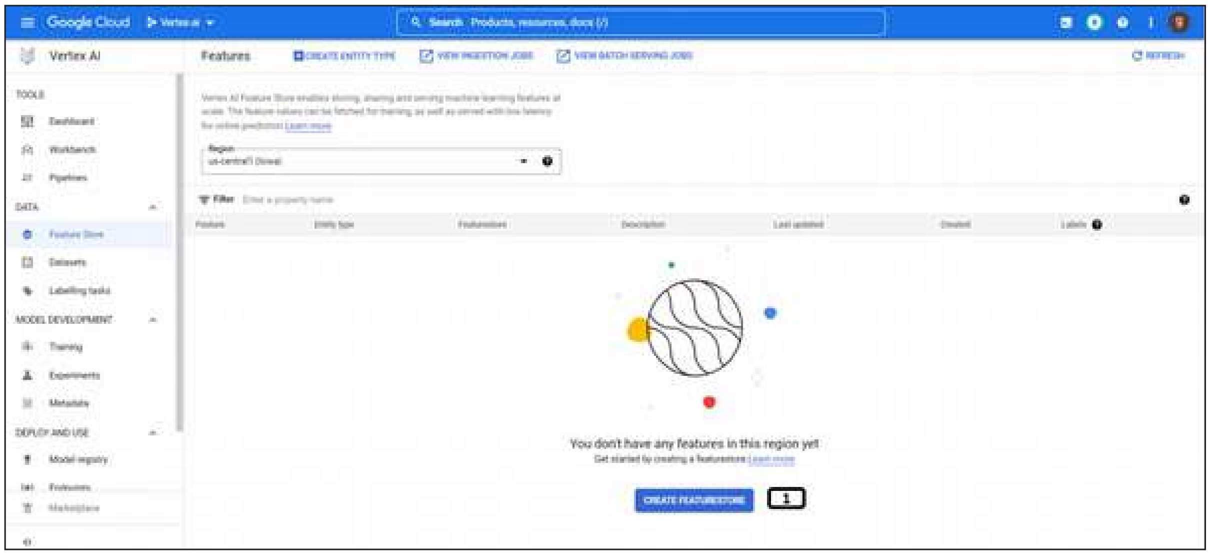


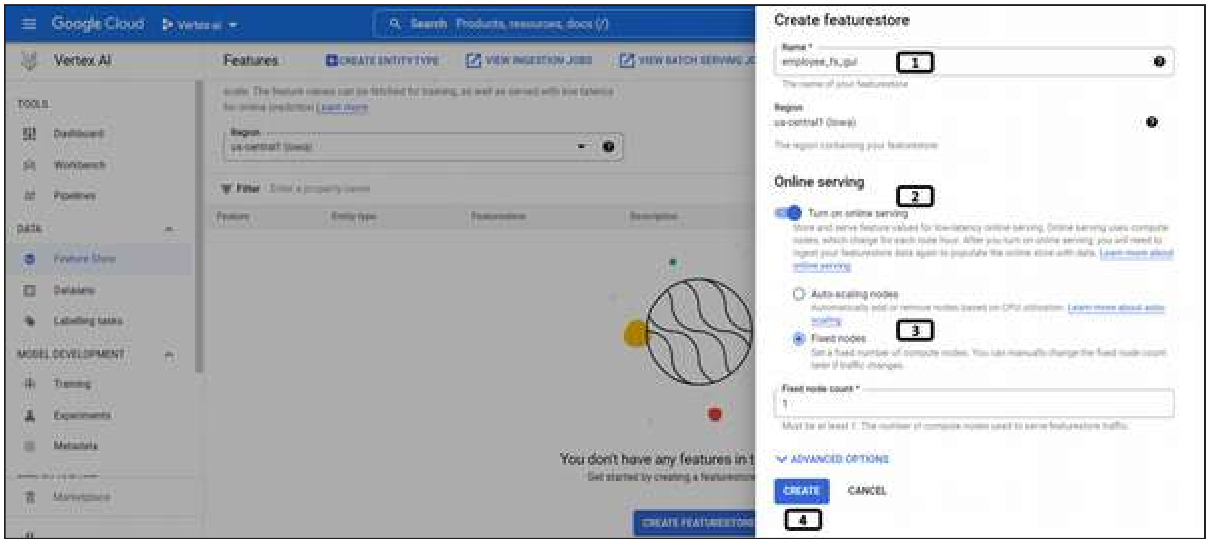
***Figure 9.4**: Landing page of feature store*

1. Click **CREATE FEATURESTORE**.

The region needs to be selected in this step (region cannot be changed post this step).

Step 3: Creation of feature store

Follow the steps mentioned in *Figure 9.5* to create a feature store:



Create featurestore

Name *: employee_fs_gui

The name of your featurestore

Region: us-central1 (Iowa)

The region containing your featurestore

Online serving

Turn on online serving

Store and serve feature values for low-latency online serving. Online serving uses compute nodes, which charge for each node hour. After you turn on online serving, you will need to ingest your featurestore data again to populate the online store with data. Learn more about online serving

Auto-scaling nodes

Automatically add or remove nodes based on CPU utilization. Learn more about auto scaling

Fixed nodes

Set a fixed number of compute nodes. You can manually change the fixed node count later if traffic changes.

Fixed node count *: 1

Must be at least 1. The number of compute nodes used to serve featurestore traffic.

Figure 9.5: *Creation of feature store*

1. Provide a name for the feature store.
2. Enable **Online Serving** if the features need to be made available for low-latency online serving.
3. Since the volume of data is small, select **Fixed Nodes** and provide the value of **1**.
4. Click on **CREATE**.

Step 4: Feature store created successfully

Once the feature store is created it will be displayed on the landing page as shown in *Figure 9.6*:

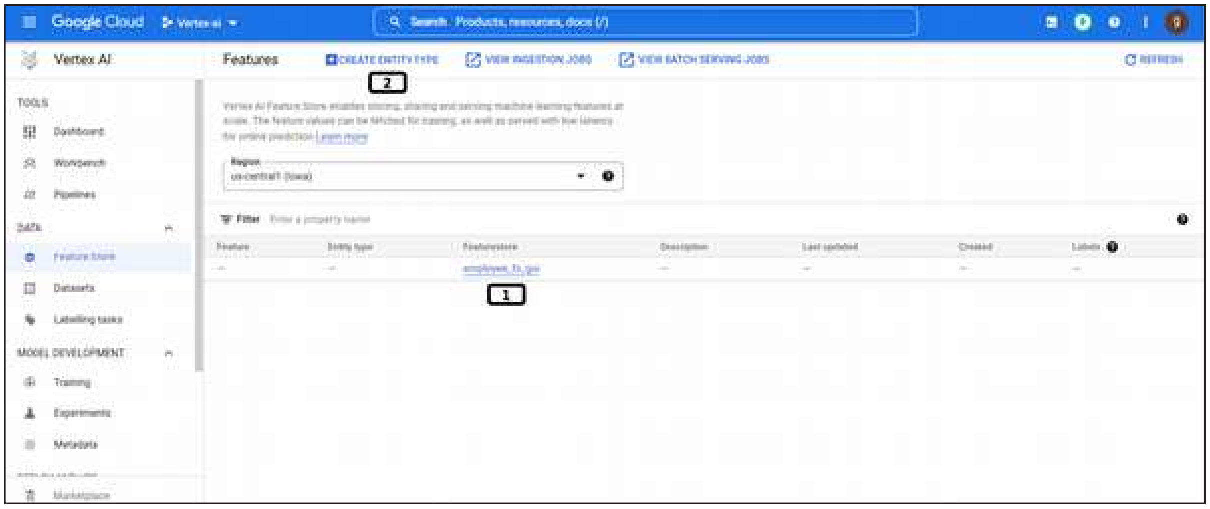


Figure 9.6: Feature store created and listed on the landing page

1. Newly created feature store.
2. Click on **Create Entity Type** for its creation.

Step 5: Creation of entity type

Follow the steps shown in *Figure 9.7* to create an entity type:

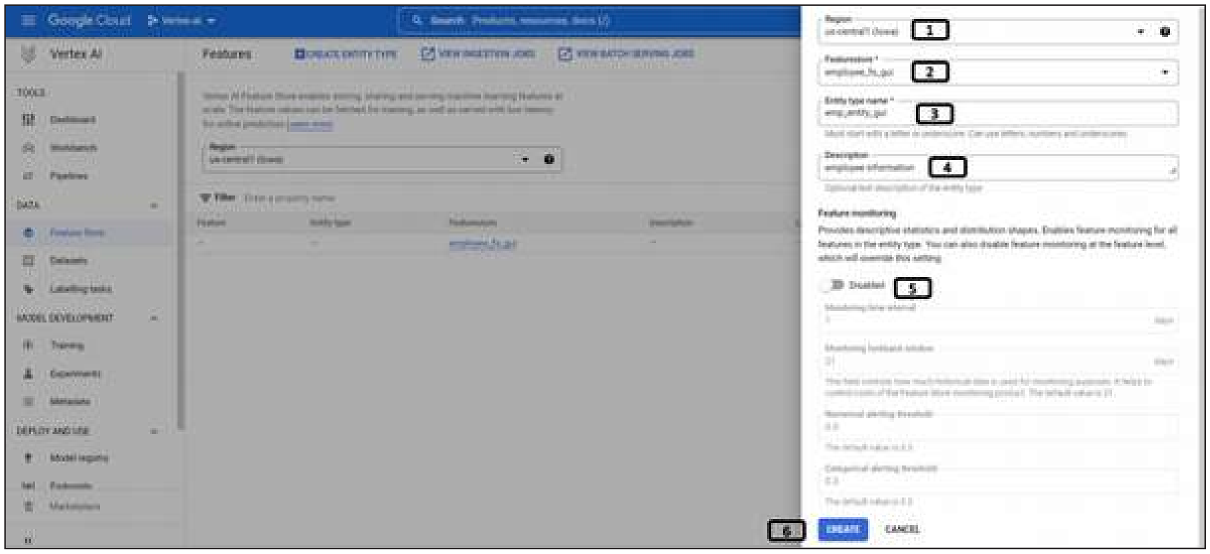


Figure 9.7: Creation of entity type

1. Select the **Region** under which the feature store is created.
2. All the feature stores under the region will be listed, select the newly created **Featurestore**.
3. Provide **Entity type** name.
4. Write the **Description** for the entity type.

5. Feature monitoring (enable if the features need to be monitored).

a. It enables the monitoring of feature stores and features. Feature store monitors CPU utilization, storage, and latency. Feature monitoring helps in monitoring changes in feature value distribution.

6. Click **CREATE**.

Step 6: Entity type created successfully

The entity type is created successfully as shown in *Figure 9.8* under the selected feature store:

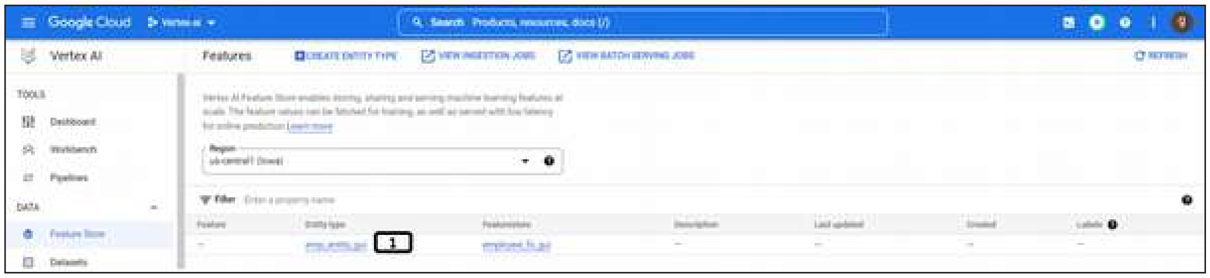


Figure 9.8: *Entity type created and listed on the landing page*

1. Click the newly created **Entity type**.

Step 7: Creation of features

Once the entity type is created, features need to be created before ingesting the values. Follow the steps mentioned in *Figure 9.9* to create features:

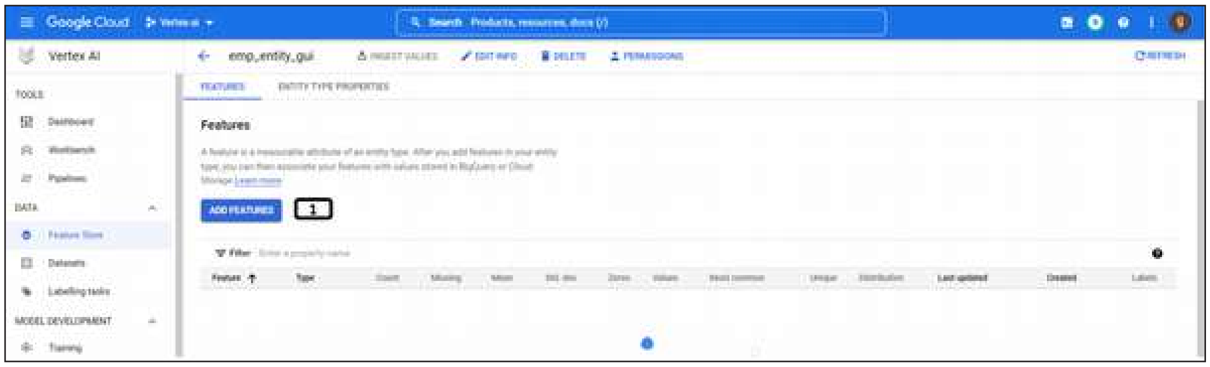
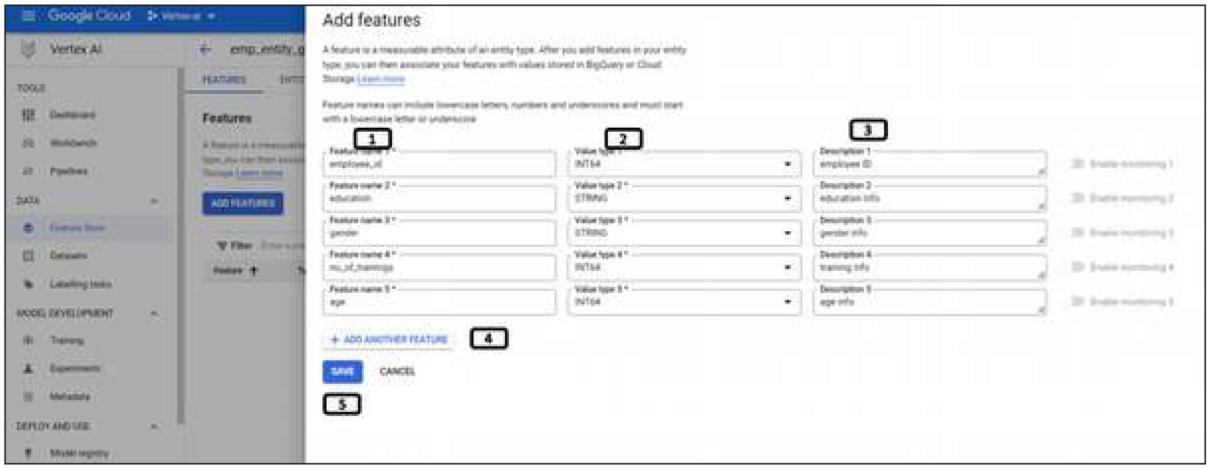


Figure 9.9*: Creation of features*

1. Click **ADD FEATURES**.

   A new side tab will pop out to enter the features as shown in *Figure 9.10*, follow the below steps to create the features:


Add features

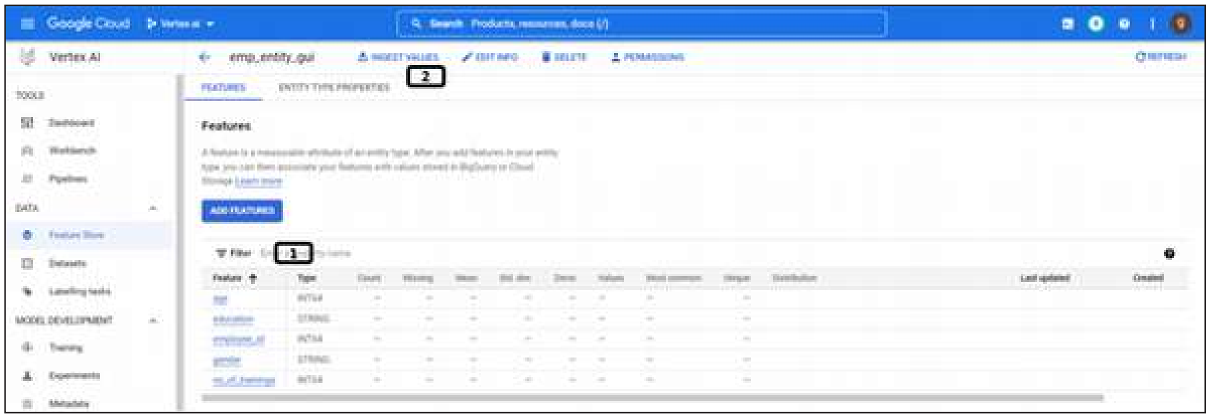
Feature name	Value type	Description
employee_id	INT64	employee ID
education	STRING	education info
gender	STRING	gender info
no_of_trainings	INT64	training info
age	INT64	age info

Figure 9.10: Adding user input for feature creation

2. Enter the **Feature name**.
3. Enter the **Value type** stored in that feature.
4. Enter the **Description** for the feature.
5. Click **Add Another Feature** to add new features.
6. Click **SAVE**, once all the features are added.

Step 8: Features created successfully

Once the features are created successfully, they are displayed on the entity type page as shown in *Figure 9.11*:


emp_entity_gui

Feature	Type
age	INT64
education	STRING
employee_id	INT64
gender	STRING
no_of_trainings	INT64

Figure 9.11: Features listed under the entity type

1. Newly created features are displayed in tabular format.
2. Click on **Ingest Values** to add the feature values.

Step 9: Ingesting feature values

Follow the steps mentioned in *Figure 9.12* to initiate the ingestion of feature values:

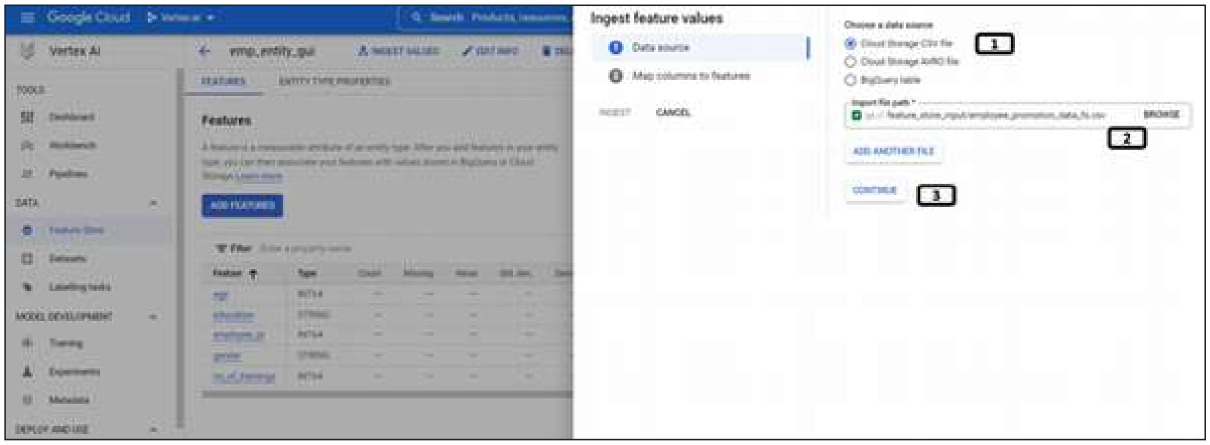


***Figure 9.12**: Importing data to features*

1. Data can be ingested from cloud storage or BigQuery. Select **Cloud Storage CSV file**.
2. Select the CSV file from the cloud storage by clicking **BROWSE**.
3. Click **CONTINUE** and follow the steps mentioned in *Figure 9.13*:

After selecting the data source, we need to map the columns of the data source to the features. Follow the steps mentioned in the *Figure 9.13* to map the features:

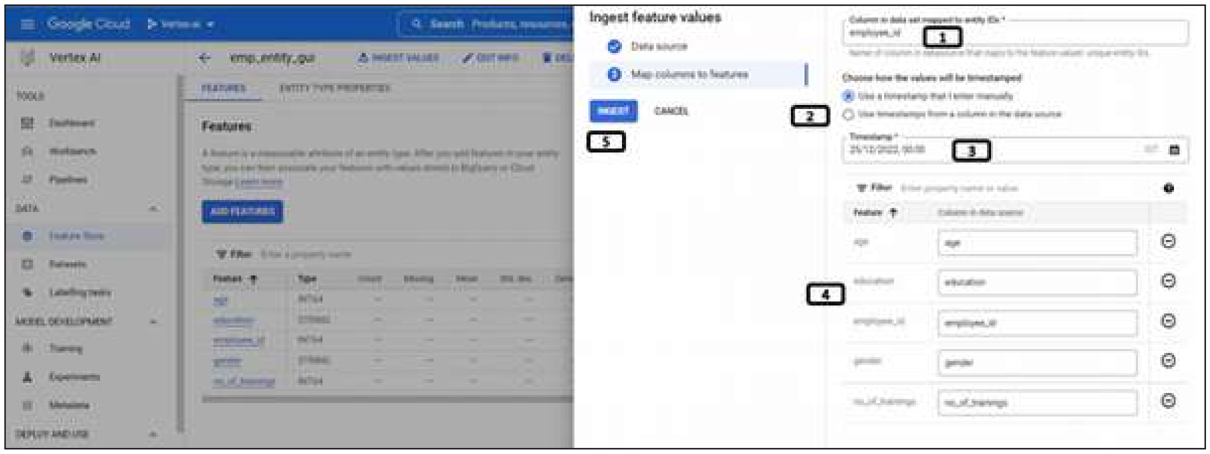


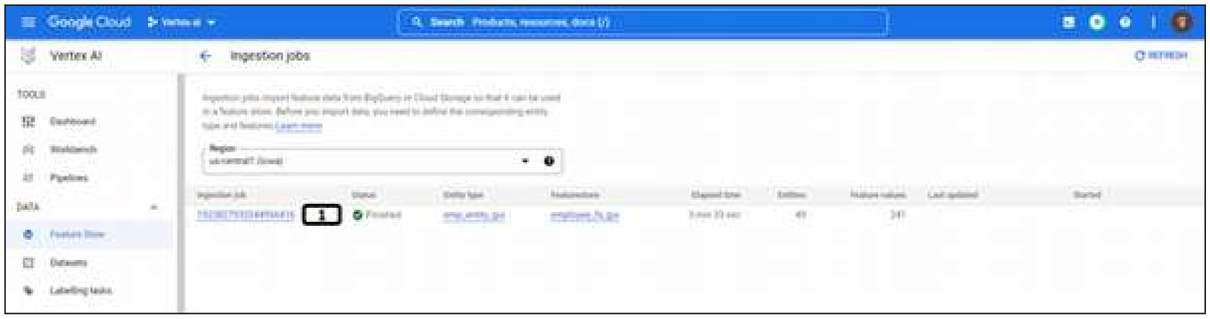
***Figure 9.13**: Mapping of columns to features*

1. Add **employee ID**, since that is the column which is containing unique values.
2. Select to enter the **Timestamp** manually. If data contains the timestamp values, the same column can be used here.
3. Select the date and time.

4. Map the column names in the CSV file to the features.
5. Click **INGEST** to initiate the ingestion job.

Step 10: Ingestion job successful

Once the feature values are ingested successfully, the ingestion job status will be updated as shown in *Figure 9.14*:



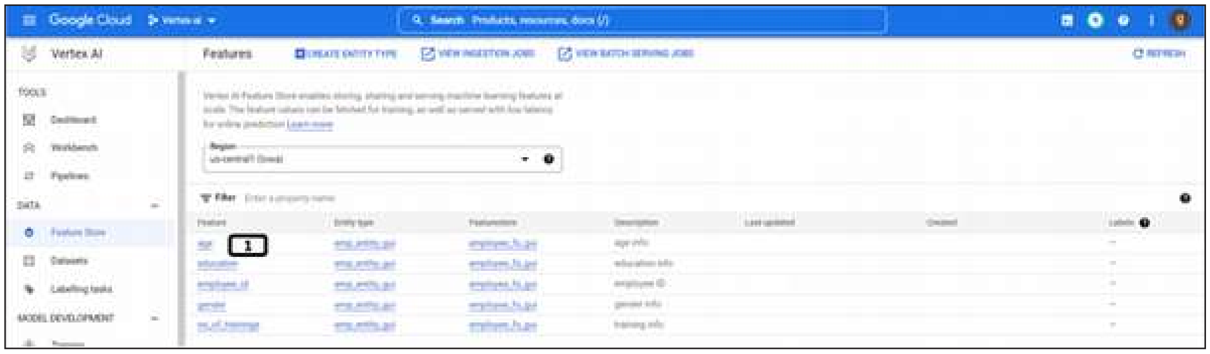
Ingestion job	Status	Entity type	Featurestore	Elapsed time	Entities	Feature values	Last updated	Started
[illegible]	Finished	emp_entity_gui	employee_fs_gui	3 min 33 sec	49	241		

Figure 9.14: *Ingestion jobs of feature store*

1. The ingestion job is completed successfully.

Step 11: Landing page of feature store after the creation of feature store, entity type, and features

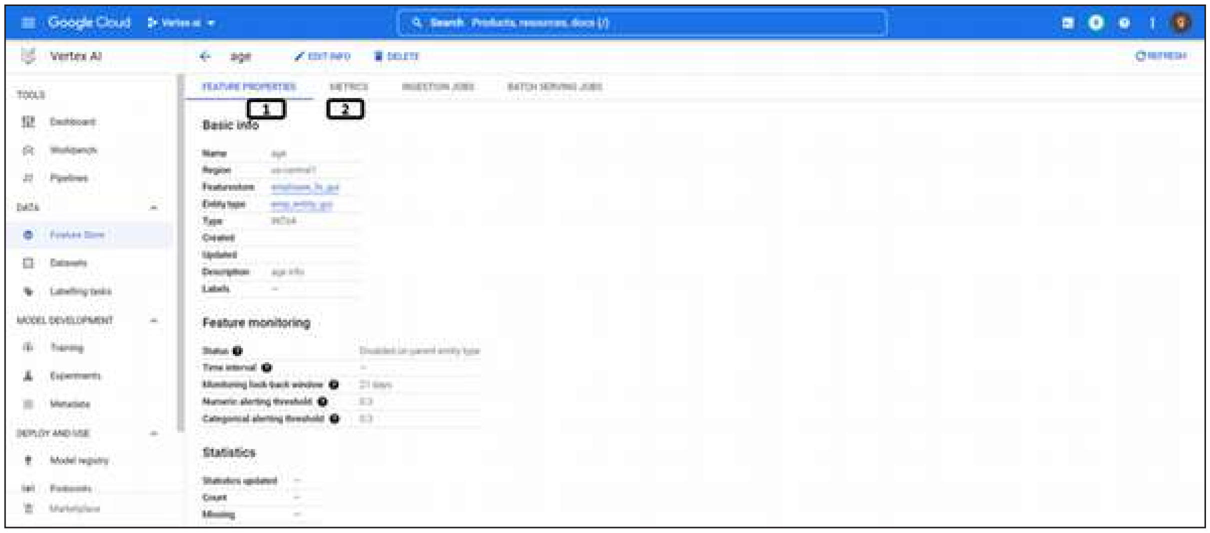
The landing page of the feature store is shown in *Figure 9.15*, all the features under entity type and feature store are listed and displayed in the tabular format:



Feature	Entity type	Featurestore	Description	Last updated	Created	Labels
age	emp_entity_gui	employee_fs_gui	age info			–
education	emp_entity_gui	employee_fs_gui	education info			–
employee_id	emp_entity_gui	employee_fs_gui	employee ID			–
gender	emp_entity_gui	employee_fs_gui	gender info			–
no_of_trainings	emp_entity_gui	employee_fs_gui	training info			–

Figure 9.15: *Landing page of feature store after the creation of features*

1. Click the **age** feature. The window will navigate to the properties of the feature as shown in *Figure 9.16*:



Basic info

Name age
Region us-central1
Featurestore employee_fs_gui
Entity type emp_entity_gui
Type INT64
Created
Updated
Description age info
Labels —

Feature monitoring

Status Disabled on parent entity type
Time interval —
Monitoring look back window 21 days
Numeric alerting threshold 0.3
Categorical alerting threshold 0.3

Statistics

Statistics updated —
Count —
Missing —

Figure 9.16: *Properties of feature*

1. For all the features, **Feature Properties** consisting of basic information and statistics are displayed.
2. **Metrics** are populated if the monitoring feature is enabled for the feature store and for that particular feature.

Working on feature store using Python

In the previous section, we worked on a feature store for the creation and uploading of feature values using the GUI approach. In this section, we shall create another feature store, ingest values, and also fetch the values from the feature store.

We will be using the **Python 3 notebook** file to type commands for working on the feature store. Follow the following-mentioned steps to create a Python file and type the Python codes given in this section.

Step 1: Create a Python notebook file

Once the workbench is created, open Jupyterlab and follow the steps mentioned in *Figure 9.17* to create a Python notebook file:

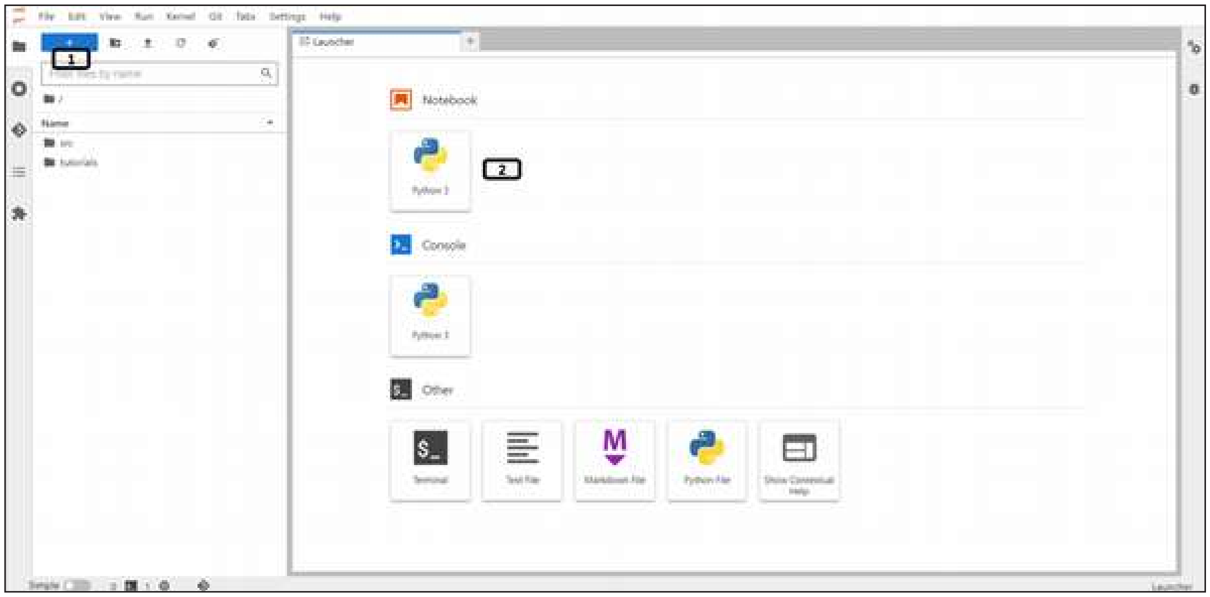


Figure 9.17: New launcher window of notebook

1. Click the new launcher.
2. Double-click the **Python 3** notebook file to create one.

Step 2: Package installation

Run the following commands to install the google cloud AI platform package. (It will take a few minutes to install the packages):

```
USER=”--user”

!pip install {USER} google-cloud-aiplatform
```

Step 3: Kernel restart

Type the following commands in the next cell, to restart the kernel. (users can restart the kernel from the GUI as well):

```
import os

import IPython

if not os.getenv(“”):

    IPython.Application.instance().kernel.do_shutdown(True)
```

Step 4: Importing the installed packages

Run the following-mentioned codes in a new cell to import the required packages:

```
import google.cloud.aiplatform_v1
```

```
from google.cloud.aiplatform_v1.types import featurestore_service as fs_s

from google.cloud.aiplatform_v1.types import featurestore as fs

from google.cloud.aiplatform_v1.types import feature

from google.cloud.aiplatform_v1.types import entity_type

from google.cloud.aiplatform_v1.types import io

from google.protobuf.timestamp_pb2 import Timestamp

from google.cloud.aiplatform_v1.types.featurestore_service import
ImportFeatureValuesRequest

from google.cloud.aiplatform_v1.types import FeatureSelector, IdMatcher

from google.cloud.aiplatform_v1.types import featurestore_online_service

import datetime
```

Step 5: Setting up the project and other variables

Run the following-mentioned line of codes in a new cell to set the project to the current one and also define variables to store the path for multiple purposes:

```
Project_id="vertex-ai-gcp-1"

featurestore_name="employee_fs_pysdk"

Entity_name="emp_entity_pysdk"

location = "us-central1"

endpoint = "us-central1-aiplatform.googleapis.com"
```

Step 6: Connecting to the feature store

Connection to the feature store is the first step to work on the feature store. We create a connection to the feature store through the service client to create and ingest values to it using **`FeaturestoreServiceClient`**.

`FeaturestoreOnlineServingServiceClient` is used to fetch the feature values from the feature store. Run the below-mentioned line of codes to complete the connection:

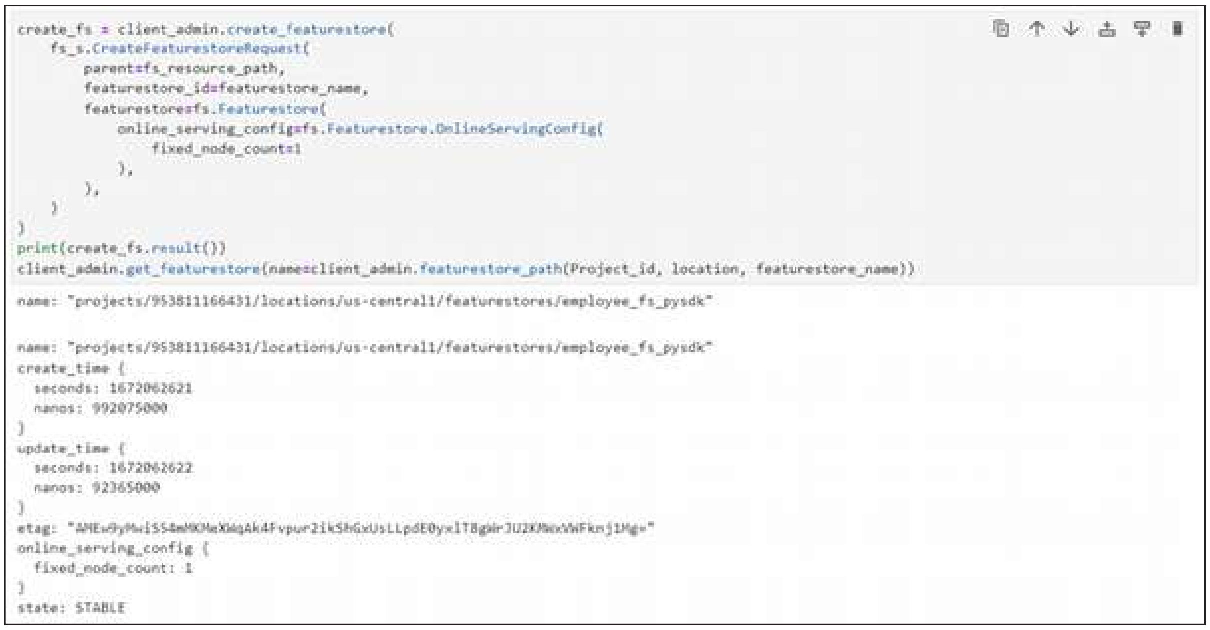
```
client_admin = google.cloud.aiplatform_v1.FeaturestoreServiceClient(cli-
ent_options={"api_endpoint": endpoint})

client_data  =  google.cloud.aiplatform_v1.FeaturestoreOnlineServingSer-
viceClient(client_options={"api_endpoint": endpoint})
```

```
fs_resource_path = client_admin.common_location_path(Project_id, loca-
tion)
```

Step 7: Creation of feature store

Instead of using the feature store that has been created from the GUI approach, we will create a new one. Feature store name, location, and project information are already in Step 5 (Setting up the project and other variables). Run the following code to create the feature store. The status of the feature store will be displayed in the results, as shown in *Figure 9.18*. The feature store creation procedure is a long-running operation, they are asynchronous jobs. except PI calls like updating or removing feature stores follow the same procedure.

```
create_fs = client_admin.create_featurestore(

    fs_s.CreateFeaturestoreRequest(

        parent=fs_resource_path,

        featurestore_id=featurestore_name,

        featurestore=fs.Featurestore(

            online_serving_config=fs.Featurestore.OnlineServingConfig(

                fixed_node_count=1

            ),

        ),

    )

)

print(create_fs.result())

client_admin.get_featurestore(name=client_admin.featurestore_path(Proj-
ect_id, location, featurestore_name))
```



```
create_fs = client_admin.create_featurestore(
    fs_s.CreateFeaturestoreRequest(
        parent=fs_resource_path,
        featurestore_id=featurestore_name,
        featurestore=fs.Featurestore(
            online_serving_config=fs.Featurestore.OnlineServingConfig(
                fixed_node_count=1
            ),
        ),
    )
)
print(create_fs.result())
client_admin.get_featurestore(name=client_admin.featurestore_path(Project_id, location, featurestore_name))
```

```
name: "projects/953811166431/locations/us-central1/featurestores/employee_fs_pysdk"

name: "projects/953811166431/locations/us-central1/featurestores/employee_fs_pysdk"
create_time {
  seconds: 1672062621
  nanos: 992075000
}
update_time {
  seconds: 1672062622
  nanos: 92365000
}
etag: "AMEw9yMwiS54mMKMeXWqAk4Fvpur2ikShGxUsLLpdE0yxlT8gWrJU2KMWxVWFknj1Mg="
online_serving_config {
  fixed_node_count: 1
}
state: STABLE
```

***Figure 9.18**: Feature store creation using Python*

Step 8: Creation of entity type

Entity type will be created under the newly created feature store using the **`create_entity_type`** method. Run the below-mentioned code in a new cell to create the entity type. The output of the last line in the code provides the path of the entity type created. Check the feature store landing page, the newly created feature store, and the entity type will be displayed:

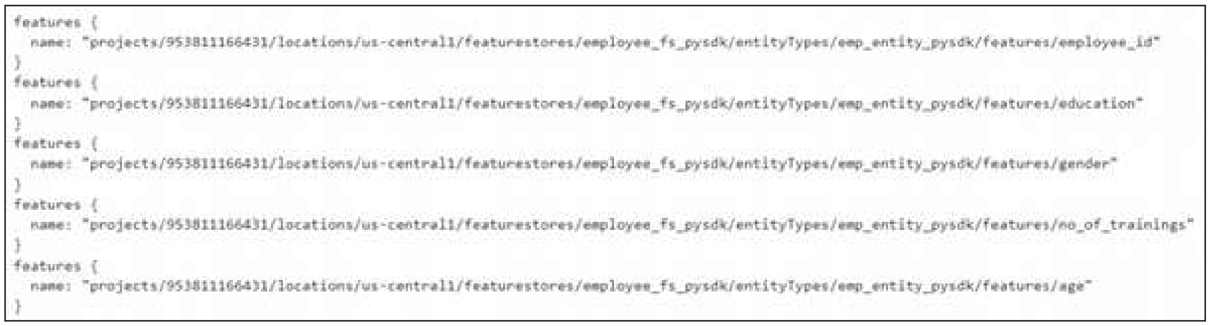
```
entity_creation = client_admin.create_entity_type(

    fs_s.CreateEntityTypeRequest(

        parent=client_admin.featurestore_path(Project_id, location,
        featurestore_name),

        entity_type_id=Entity_name,

        entity_type=entity_type.EntityType(

            description=”employee entity”,

        ),

    )

)

print(entity_creation.result())
```

Step 9: Creation of feature

Once the feature store and entity type are created, the feature needs to be created before ingesting the feature values. For each of the features, information on feature ID, type, and description is provided. Run the following-mentioned code in a new cell to add features:

```
client_admin.batch_create_features(

    parent=client_admin.entity_type_path(Project_id, location,
    featurestore_name, Entity_name),

    requests=[

        fs_s.CreateFeatureRequest(

            feature=feature.Feature(

                value_type=feature.Feature.ValueType.INT64,

                description=”employee id”,

            ),

            feature_id=”employee_id”,

        ),

        fs_s.CreateFeatureRequest(

            feature=feature.Feature(

                value_type=feature.Feature.ValueType.STRING,

                description=”education”,

            ),

            feature_id=”education”,

        ),

        fs_s.CreateFeatureRequest(

            feature=feature.Feature(

                value_type=feature.Feature.ValueType.STRING,

                description=”gender”,

            ),
            feature_id=”gender”,
        ),
        fs_s.CreateFeatureRequest(
            feature=feature.Feature(
                value_type=feature.Feature.ValueType.INT64,
                description=”no_of_trainings”,
            ),
            feature_id=”no_of_trainings”,
        ),
        fs_s.CreateFeatureRequest(
            feature=feature.Feature(
                value_type=feature.Feature.ValueType.INT64,
                description=”age”,
            ),
            feature_id=”age”,
        ),
    ],
).result()
```

Once the features are created, they are displayed in the output of the cell as shown in *Figure 9.19*:



```
features {
  name: "projects/953811166431/locations/us-central1/featurestores/employee_fs_pysdk/entityTypes/emp_entity_pysdk/features/employee_id"
}
features {
  name: "projects/953811166431/locations/us-central1/featurestores/employee_fs_pysdk/entityTypes/emp_entity_pysdk/features/education"
}
features {
  name: "projects/953811166431/locations/us-central1/featurestores/employee_fs_pysdk/entityTypes/emp_entity_pysdk/features/gender"
}
features {
  name: "projects/953811166431/locations/us-central1/featurestores/employee_fs_pysdk/entityTypes/emp_entity_pysdk/features/no_of_trainings"
}
features {
  name: "projects/953811166431/locations/us-central1/featurestores/employee_fs_pysdk/entityTypes/emp_entity_pysdk/features/age"
}
```

***Figure 9.19**: Addition of features to the feature store using Python*

Step 10: Define the ingestion job

As seen in the web console, feature values can be ingested from cloud storage or BigQuery. We shall use the same CSV file which has been uploaded to the cloud storage. Importantly, we should also supply the timestamp information while ingesting the values. Timestamps can be provided in the code or there can be a separate column in the data that contains timestamp information. Timestamp information must be in **`google.protobuf.Timestamp`** format. Run the following code in a new cell to define the ingestion job:

```
seconds = int(datetime.datetime.now().timestamp())

timestamp_input = Timestamp(seconds=seconds)

ingest_data_csv = fs_s.ImportFeatureValuesRequest(

    entity_type=client_admin.entity_type_path(

        Project_id, location, featurestore_name, Entity_name

    ),

    csv_source=io.CsvSource(

        gcs_source=io.GcsSource(

            uris=[

               "gs://feature_store_input/employee_promotion_data_fs.csv"

            ]

        )

    ),

    entity_id_field="employee_id",

    feature_specs=[

        ImportFeatureValuesRequest.FeatureSpec(id="employee_id"),

        ImportFeatureValuesRequest.FeatureSpec(id="education"),

        ImportFeatureValuesRequest.FeatureSpec(id="gender"),

        ImportFeatureValuesRequest.FeatureSpec(id="no_of_trainings"),

        ImportFeatureValuesRequest.FeatureSpec(id="age"),

    ],
    feature_time=timestamp_input,

    worker_count=1,

)
```

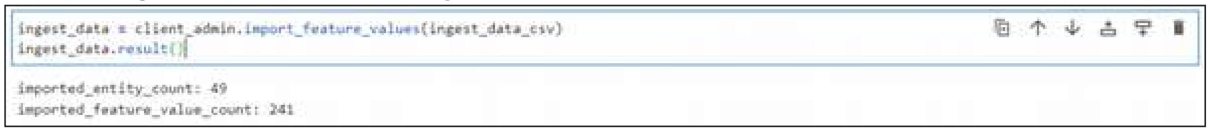
NOTE: If all feature values were generated at the same time, there is no need to have a timestamp column. Users can specify the timestamp as part of the ingestion request.

Step 11: Initiation of ingestion job

The ingestion job needs to be initiated after it is defined, run the following line of codes to begin the ingestion process:

```
ingest_data = client_admin.import_feature_values(ingest_data_csv)

ingest_data.result()
```

Once the ingestion is complete, it will provide information on the number of feature values ingested as shown in *Figure 9.20*:



```
ingest_data = client_admin.import_feature_values(ingest_data_csv)
ingest_data.result()

imported_entity_count: 49
imported_feature_value_count: 241
```

***Figure 9.20**: Ingestion of feature values using Python*

Step 12: Fetching feature values

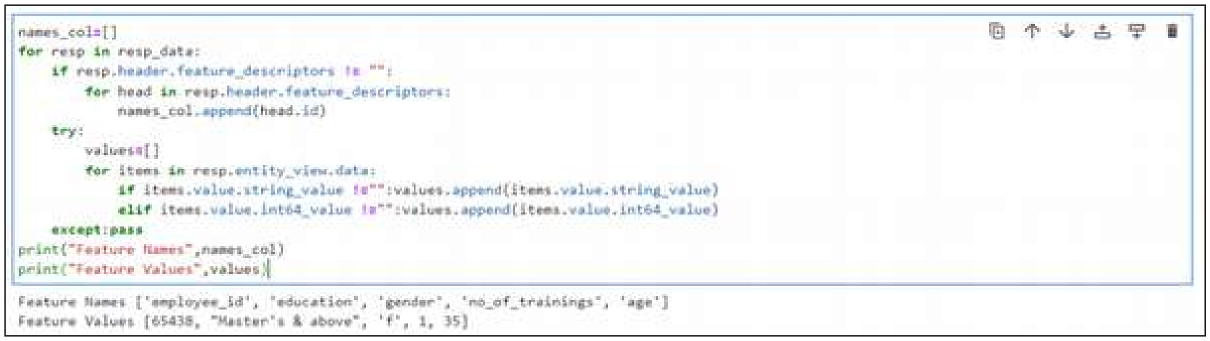
Feature values can be extracted from the feature store with the help of an online service client which has been created in Step 7 (Creation of feature store). Run the following lines of code to fetch data from the feature for a specific employee ID:

```
resp_data = client_data.streaming_read_feature_values(

    featurestore_online_service.StreamingReadFeatureValuesRequest(

        entity_type=client_admin.entity_type_path(

            Project_id, location, featurestore_name, Entity_name

        ),

        entity_ids=[“65438”],

        feature_selector=FeatureSelector(id_
        matcher=IdMatcher(ids=[“employee_id”,”education”,”gender”,”no_
        of_trainings”,”age”])),
    )
)
print(resp_data)
```

The output will be stored in the **`resp_data`** variable and it is an iterator. Run the following lines of code to extract and parse the data from the iterator:

```
names_col=[]
for resp in resp_data:
    if resp.header.feature_descriptors != “”:
        for head in resp.header.feature_descriptors:
            names_col.append(head.id)
    try:
        values=[]
        for items in resp.entity_view.data:
            if items.value.string_value !=””:values.append(items.value.
            string_value)
            elif items.value.int64_value !=””:values.append(items.value.
            int64_value)
    except:pass
print(“Feature Names”,names_col)
print(“Feature Values”,values)
```

The output of the cell is shown in *Figure 9.21*:



```
names_col=[]
for resp in resp_data:
    if resp.header.feature_descriptors != "":
        for head in resp.header.feature_descriptors:
            names_col.append(head.id)
    try:
        values=[]
        for items in resp.entity_view.data:
            if items.value.string_value !="":values.append(items.value.string_value)
            elif items.value.int64_value !="":values.append(items.value.int64_value)
    except:pass
print("Feature Names",names_col)
print("Feature Values",values)
```

```
Feature Names ['employee_id', 'education', 'gender', 'no_of_trainings', 'age']
Feature Values [65438, "Master's & above", 'f', 1, 35]
```

***Figure 9.21**: Data extracted from the feature store*

Deleting resources

We have utilized cloud storage to store the data and delete the CSV file from the cloud storage manually. Feature store is a cost-incurring resource on GCP, ensure to delete them. Also, the feature store cannot be deleted from the web console or GUI, we need to delete it through programming. Run the below-mentioned lines of code to delete the feature store (both the feature stores were created using GUI and Python). Check if the landing page of the feature store after running the code to ensure it is deleted (the landing page should look like the *Figure 9.4*):

```
client_admin.delete_featurestore(

    request=fs_s.DeleteFeaturestoreRequest(

        name=client_admin.featurestore_path(Project_id, location,
        featurestore_name),

        force=True,

    )

).result()

featurestore_name=employee_fs_gui

client_admin.delete_featurestore(

    request=fs_s.DeleteFeaturestoreRequest(

        name=client_admin.featurestore_path(Project_id, location,
        featurestore_name),

        force=True,

    )

).result()
```

Best practices for feature store

Below listed are few of the best practices for using Feature store of Vertex AI:

1. **Model features to multiple entities**: Some features might be used in multiple entities (like clicks per product at user level). In this kind of scenarios, it is best to create a separate entity to group shared features.
2. **Access control for multiple teams**: Multiple teams like data scientists, ML researchers, Devops, and so on, may require access to the same feature store

but with different level of permissions. Resource level IAM policies can be used to restrict the access to feature store or particular entity type.

3. **Ingesting historical data (backfilling)**: It is recommended to stop online serving while ingesting the historical data to prevent any changes to the online store.
4. **Cost optimization**:
    - **Autoscaling**: Instead of maintaining a high node count, autoscaling allows Vertex AI Feature Store to analyze traffic patterns and automatically modify the number of nodes up or down based on CPU consumption and also works better for cost optimization.
    - Recommended to provide a **`startTime`** in the **`batchReadFeatureValues`** or **`exportFeatureValues`** request to optimize offline storage costs during batch serving and batch export.

Conclusion

In this chapter, we learned about the feature store of Vertex AI, and worked on the creation of the feature store, entity type, adding features, and ingesting feature values using web console and Python.

In the next chapter, we will start understanding explainable AI, and how explainable AI works on Vertex AI.

Questions

1. What are the different input sources from which data can be ingested into a feature store?
2. Can feature stores have multiple entity types?
3. What are the scenarios in which using a feature store brings value?

Chapter 10
Explainable AI

Introduction

This last chapter of the book covers explainable AI. We will start with understanding what is explainable AI, its need, how explainable AI works on Vertex AI (for image and tabular data) and how to get the explanations from the deployed model.

Structure

In this chapter, we will discuss the following topics:

- What is Explainable AI
- Need of Explainable AI
- XAI on Vertex AI
- Data for Explainable AI exercise
- Model training for image data
- Image classification model deployment
- Explanations for image classification
- Tabular classification model deployment
- Explanations for tabular data

- Deletion of resources
- Limitations of Explainable AI

Objectives

By the end of this chapter, you will have a good idea about explainable AI and will know how to get the explanations from the deployed model in Vertex AI.

What is Explainable AI

Explainable AI (**XAI**) is a subfield of **Artificial Intelligence** (**AI**) that focuses on developing methods and strategies for using AI in a way that makes the outcomes of the solution understandable to human specialists. The mission of XAI is to ensure that AI systems be open and honest about not just the function they perform but also the purpose they serve. Interpretability is the broader umbrella under AI which includes explainable AI as one of its subcategories. Users can grasp what a model is learning, the additional information it must provide, and the reasoning behind its judgments concerning the problem that exists in the real world that we are seeking to solve, thanks to the model's interpretability.

Explainable AI is one of the core ideas that define trust in AI systems (along with accountability, reproducibility, lack of machine bias, and resiliency). The aim and ambition shared by data scientists and machine learning technologists is the development of AI that is explainable.

Need of Explainable AI

Artificial intelligence has the ability to automate judgments, and the outcomes of such decisions may have both beneficial and bad effects on businesses. It is essential to have an understanding of how AI comes to its conclusions, just as it is essential to have this understanding when recruiting decision is made for the business. A great number of companies are interested in using AI, but are hesitant to hand over decision-making authority to the model or AI simply because they do not yet trust the model. Explainability is beneficial in this regard since it offers insights into the decision-making process that models use. Explainable AI is a crucial component in the process of applying ethics to the usage of AI in business. Explainable AI is predicated on the notion that AI-based applications and technology should not be opaque "black box" models that are incomprehensible to regular people. *Figure 10.1* shows the difference between AI and Explainable AI:

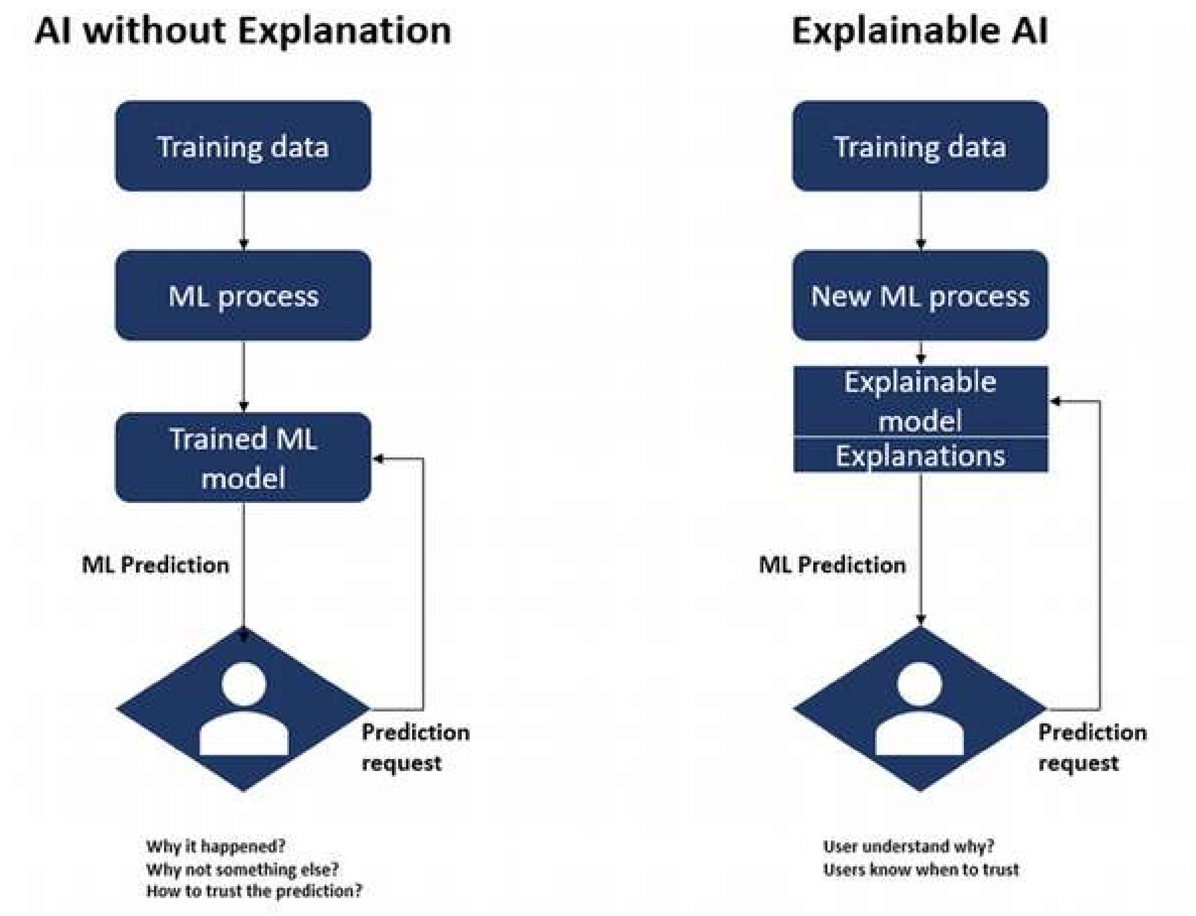


***Figure 10.1**: Explainable AI*

In the majority of the scenarios developing complex models is far easier than convincing stakeholders that the model is capable of producing decisions that are superior to those produced by humans. It is not the same thing as having greater accuracy scores or a lower RMSE to make a better judgment. A correct conclusion may be reached by providing accurate data as input. In many cases, the person making the choice is the one who needs to comprehend it. For them to feel at ease handing over the decision-making to the model, they need to understand how the model came to its conclusions.

Explainable AI is essential to the development of responsible AI because it offers an adequate amount of transparency and responsibility for the choices made by complicated AI systems. This is of utmost importance when it comes to artificial intelligence systems that have a substantial influence on the lives of people.

XAI on Vertex AI

Explainable AI of Vertex AI provides explanations that are either feature-based or example-based in order to give a better understanding of how models make decisions. Anyone who builds or uses machine learning will gain new abilities if they learn how a model behaves and how it is influenced by its training dataset. These new abilities will allow users to improve their models, increase their confidence in their predictions, and understand when and why things work.

Example-based explanations

In the case of explanations based on examples, Vertex AI makes use of the closest neighbor search to produce a list of instances (usually taken from the training set) that are most comparable to the input. These examples allow users to investigate and clarify the behavior of the model since users can reasonably anticipate that comparable inputs would result in similar predictions.

Consider the following scenario: users have a model that analyzes photos to determine whether they depict a bird or an aircraft; however, the model incorrectly identifies certain birds as planes. To figure out what is going on, we may extract other photos from the training set that is comparable to the one we are looking at and utilize example-based explanations to explain what is occurring. When we look at those instances, we see that many of the incorrectly identified birds and the training examples that are comparable to them are dark silhouettes and that most of the dark silhouettes that were aircraft were found in the training set. This suggests that users might potentially increase the quality of the model by including more silhouetted birds in the training set.

Explanations that are based on examples may also help identify confusing inputs that might be improved with human labelling. Models that provide embedding or latent representation for input variables are supported. Tree based models which do not provide embeddings for the inputs are not supported in examples-based explanations.

Feature-based explanations

Feature-based explanations is another way of explaining model output based on the features. The amount of contribution that each feature in the model made to the predictions that were made for a particular instance is shown by the feature attributions. When users make a request for predictions, they will get anticipated values that are suitable for the model you are using. Feature attribution info will be provided when users request for the explanations.

Feature attributions work on image and tabular data. They are supported for AutoML and custom trained models. (Classification models only for image data and classification/regression models for tabular data).

Feature attribution methods

Each approach for attributing features is based on Shapley values, which is an algorithm derived from cooperative game theory that gives credit for a given result to each participant in a game. When this concept is applied to models for machine learning, it indicates that each model feature is dealt with as if it were a "player" in the game. The Vertex Explainable AI provides a certain amount of credit to each individual characteristic based on its weight in the overall forecast:

- **Sampled Shapley method:** The sampled Shapley technique offers an estimate of the actual Shapley values via the use of sampling. Tabular models created using AutoML make use of the sampled Shapley approach to determine the relevance of features. For these models, which are meta-ensembles of tree and neural network structures, the Sampled Shapley method performs quite well.
- **Integrated gradients method:** Along an integral route, the integrated gradients approach calculates the gradient of the prediction output with respect to the characteristics of the input. This is done to get an accurate result. Calculations of the gradients are performed at various time intervals along a scaling parameter. Utilizing the Gaussian quadrature rule allows for the calculation of the size of each interval. (When dealing with picture data, think of this scaling option as a "slider" that sets all the image's pixels to a black value.) The integration of the gradients is done as follows:
  - An approximation of the integral may be found by using a weighted average.

  - Calculations are performed to get the element-wise product of the original input and the averaged gradients.
- **XRAI method:** To discover which parts of a picture that contribute the most to a certain prediction of class, the XRAI approach uses a combination of the integrated gradients method and some extra phases.
  - **Pixel-level attribution**: XRAI can do pixel-level attribution for the picture that is sent into it. In this stage of the process, XRAI makes use of the integrated gradients approach, applying it to both, a black and a white baseline.
  - **Over segmentation**: XRAI generates a tiny patch over the picture by over segmentation , which is done independently of pixel-level attribution. In order to construct the picture segments, XRAI takes advantage of *Felzenswalb*'s graph-based technique.
- **Region selection**: XRAI compiles the pixel-level attribution included inside each segment to calculate the attribution density of that segment. XRAI assigns a ranking to each segment based on these values, and then it arranges the segments from most positive to least positive. This identifies which parts of the picture contribute the most strongly to a certain class prediction, as well as which parts of the image are most prominent.

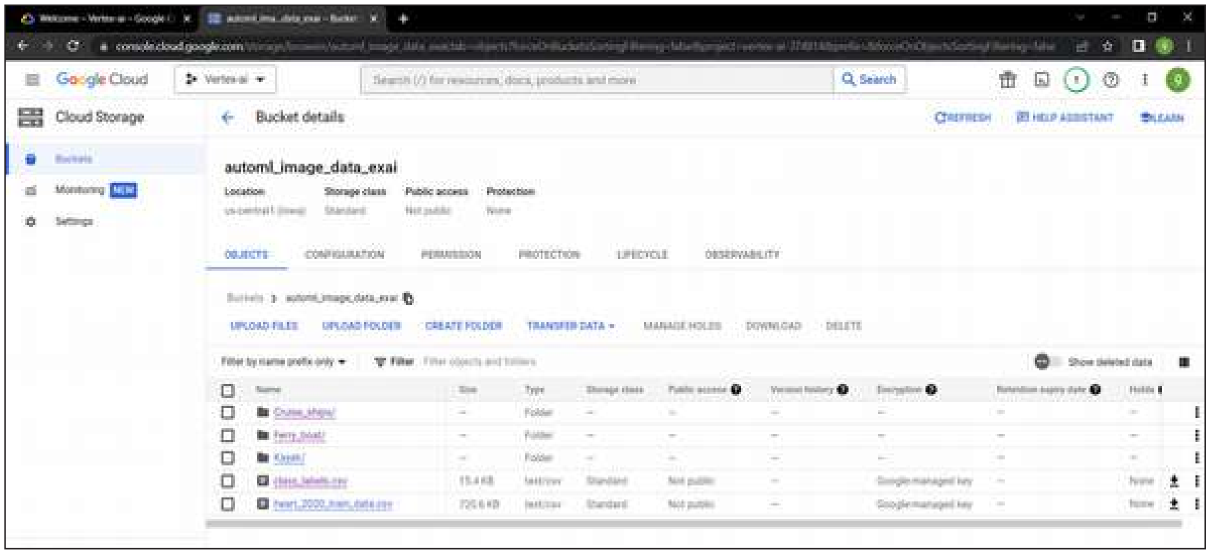
All types of models are supported for feature-based explanations. Classification models are supported for AutoML images and classification, and regression models are supported for AutoML tabular models.

Data for Explainable AI exercise

In this exercise, we will try to understand how explainable AI can be used to understand model prediction using image data and the tabular data with the help of AutoML of Vertex AI. The data used for AutoML tables and images will be used for this exercise as well. (Refer chapters Introduction to Vertex AI and AutoML Tabular and AutoML Image, text and pre-built models).

AutoML_image_data_exai bucket is created under **us-centra1** (single region).

Three folders containing image data (**Cise_ships**, **Ferry_boat** and **Kayak**) is uploaded and CSV file (**class_labels.csv**) is created as shown in *Chapter 3, AutoML Image, text and pre-built models* (refer *Figure 3.1* and *3.2* for csv creation). **heart_2020_train_data.csv** is uploaded to the same folder which will be used for AutoML tables. *Figure 10.2* shows the data uploaded to the cloud storage:



automl_image_data_exai

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	None

Buckets > automl_image_data_exai

Name	Size	Type	Storage class	Public access	Version history	Encryption	Retention expiry date	Holds
Cruise_ships/	—	Folder	—	—	—	—	—	—
Ferry_boat/	—	Folder	—	—	—	—	—	—
Kayak/	—	Folder	—	—	—	—	—	—
class_labels.csv	15.4 KB	text/csv	Standard	Not public	—	Google-managed key	—	None
heart_2020_train_data.csv	720.6 KB	text/csv	Standard	Not public	—	Google-managed key	—	None

***Figure 10.2**: Data uploaded to the cloud storage*

Model training for image data

The initial steps for dataset creation for the image data will have no changes. Refer Image dataset creation of *Chapter 3, AutoML Image, text and pre-built models* and follow the below mentioned steps for AutoML model training.

Step 1: Train new model

Newly created dataset will be listed under the dataset of the Vertex AI. Follow the below steps to initiate the model training as shown in the following screenshot:

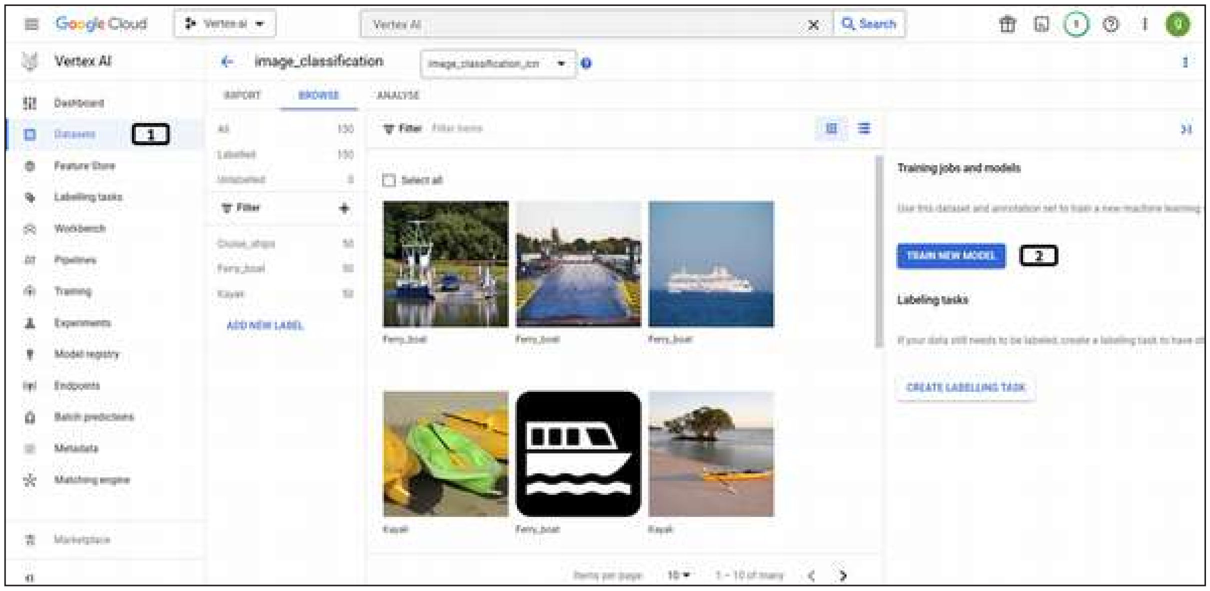


***Figure 10.3**: Image dataset created on Vertex AI*

1. Click on **Datasets** section of Vertex AI (open the newly created image dataset).
2. Click **TRAIN NEW MODEL**.

Step 2: Training method selection

Training method step does not have difference as mentioned in *Chapter 3, AutoML Image, text and pre-built models*. Follow these steps to set the training method:

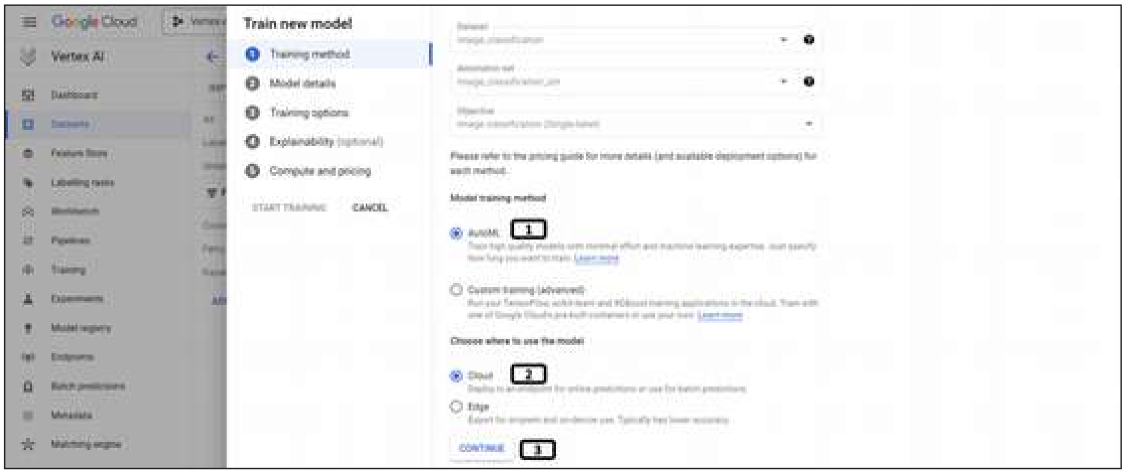
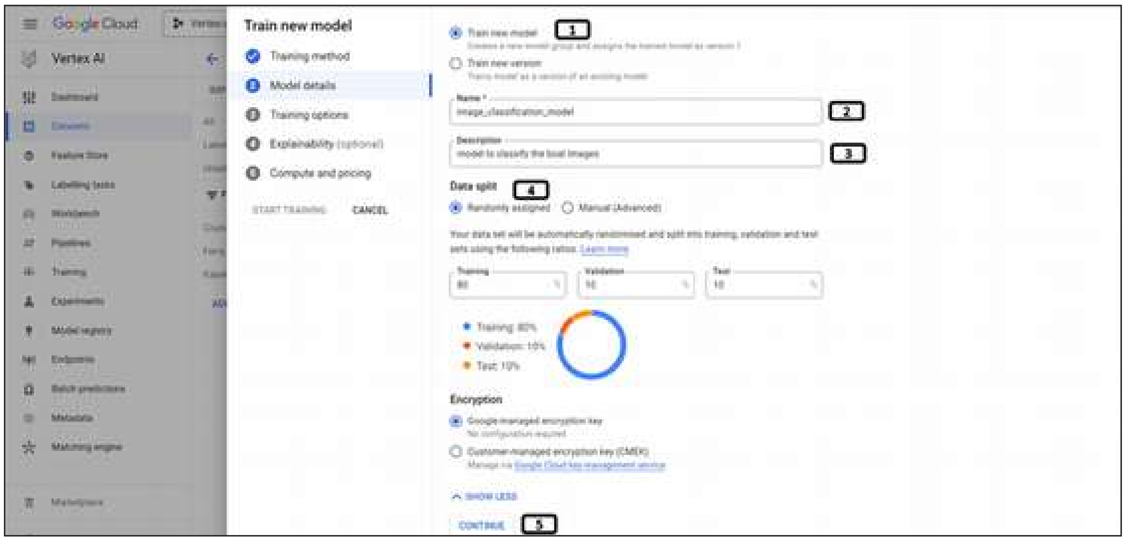


Figure 10.4: *Training method selection*

1. Select **AutoML**.
2. Select **Cloud** (we will deploy the model to get predictions and explanations).
3. Click on **CONTINUE**.

Step 3: Model details

Follow the steps mentioned below to set the model details:



Name *: image_classification_model

Description: model to classify the boat images

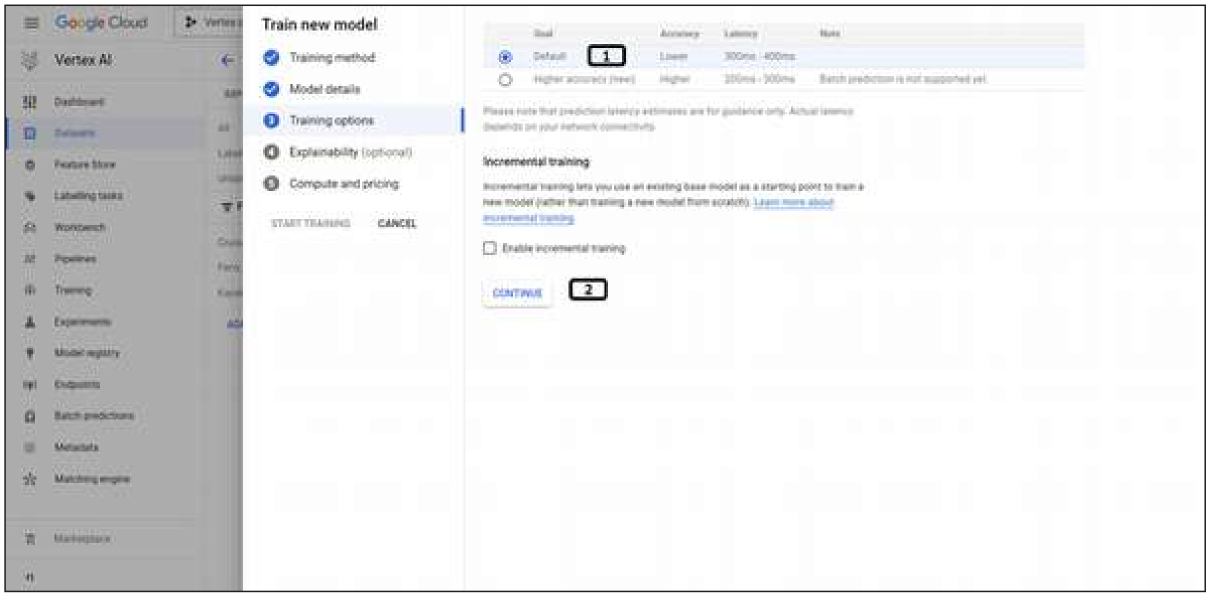
Training	Validation	Test
80	10	10

Figure 10.5: *Image classification model details*

1. Select **Train new model**.
2. Provide a **Name** for the model.
3. Provide **Description** for the model.
4. Under **Data split** select **Randomly assigned**.
5. Click **CONTINUE**.

Step 4: Training options

Follow the below mentioned steps for training options:



	Goal	Accuracy	Latency	Note
●	Default	Lower	300ms - 400ms	
○	Higher accuracy (new)	Higher	200ms - 300ms	Batch prediction is not supported yet

Please note that prediction latency estimates are for guidance only. Actual latency depends on your network connectivity.

Incremental training

Incremental training lets you use an existing base model as a starting point to train a new model (rather than training a new model from scratch). Learn more about incremental training

☐ Enable incremental training

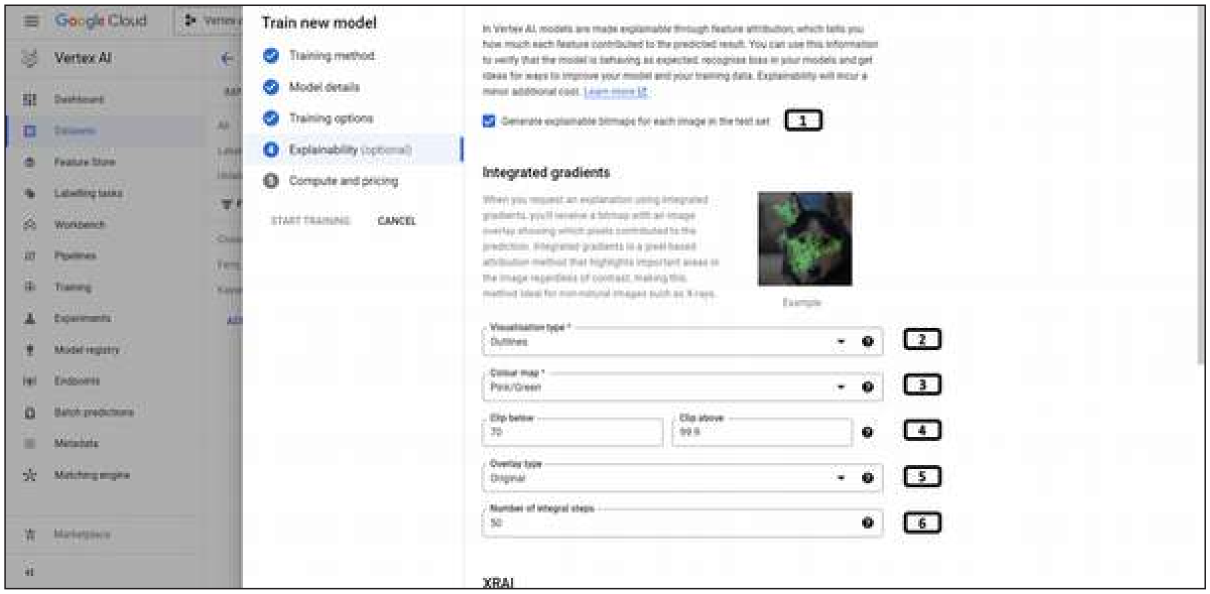
Figure 10.6: Image classification training options

1. Select **Default** training method.
2. Click **CONTINUE** (we do not have to enable incremental training for the explainable AI).

Step 5: Explain ability

Explain ability needs to be set in two places while working on AutoML images. The first one is during the training phase of the model and while deploying the model. Follow the below mentioned steps to configure explain ability of the model during training phase.

The steps shown below is for Integrated gradients method of **Explainability** as shown in the following screenshot:



In Vertex AI, models are made explainable through feature attribution, which tells you how much each feature contributed to the predicted result. You can use this information to verify that the model is behaving as expected, recognise bias in your models and get ideas for ways to improve your model and your training data. Explainability will incur a minor additional cost. Learn more

☑ Generate explainable bitmaps for each image in the test set

Integrated gradients

When you request an explanation using integrated gradients, you'll receive a bitmap with an image overlay showing which pixels contributed to the prediction. Integrated gradients is a pixel-based attribution method that highlights important areas in the image regardless of contrast, making this method ideal for non-natural images such as X-rays.

Visualisation type *: Outlines

Colour map *: Pink/Green

Clip below: 70 Clip above: 99.9

Overlay type: Original

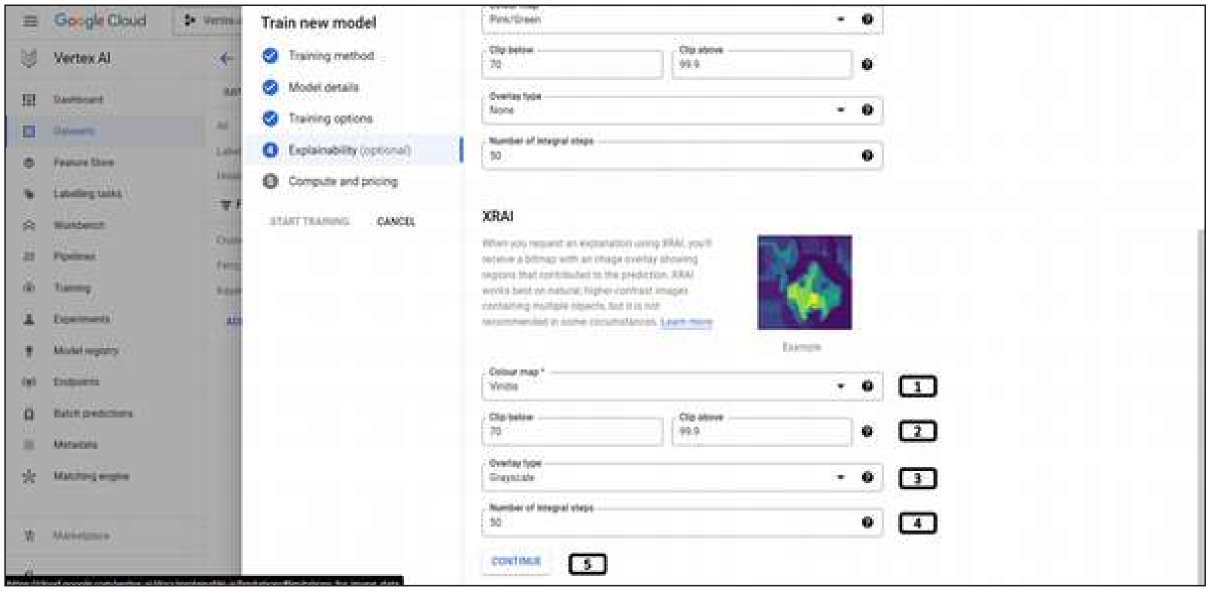
Number of integral steps: 50

XRAI

Figure 10.7: Explain ability of image classification model

1. Enable to **Generate explainable bitmaps**.
2. **Visualization type** set to **`Outlines`** (pixels is another option to understand which pixels are playing important role for the prediction)..
3. **Color map** select **Pink/Green** (Pink/Green color are used to highlight the areas on the image).
4. **Clip below** and **Clip above** parameters are used to reduce the noise. Enter **`70`** and **`99.9`** for click below and above respectively.
5. Select **`Original`** under **Overlay type** (pixels will be highlighted on top of the original image).
6. Enter **`50`** for the **Number of integral steps** (increasing this parameter will reduce the approximation error).

Scroll down and follow the steps mentioned in the following step to set the parameters for XRAI method:



Train new model

Training method

Model details

Training options

Explainability (optional)

Compute and pricing

START TRAINING CANCEL

Colour map: Pink/Green

Clip below: 70 Clip above: 99.9

Overlay type: None

Number of integral steps: 50

XRAI

When you request an explanation using XRAI, you'll receive a bitmap with an image overlay showing regions that contributed to the prediction. XRAI works best on natural, higher-contrast images containing multiple objects, but it is not recommended in some circumstances. Learn more

Example

Colour map *: Viridis 1

Clip below: 70 Clip above: 99.9 2

Overlay type: Grayscale 3

Number of integral steps: 50 4

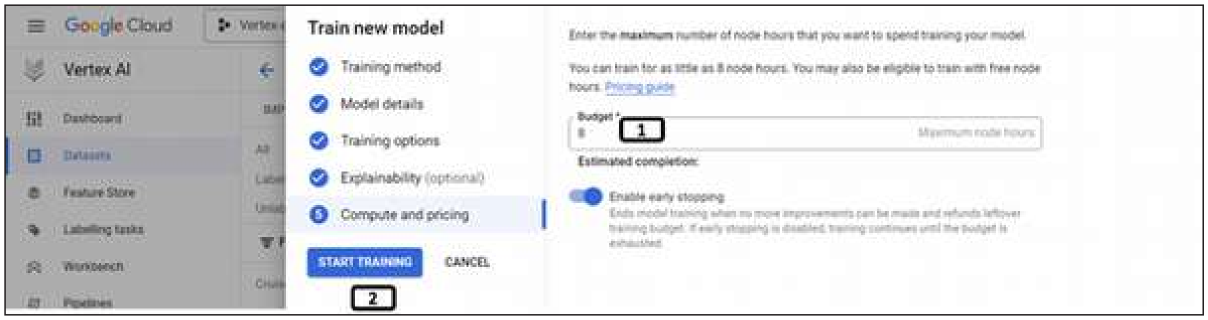
CONTINUE 5

Figure 10.8: *Explain ability of image classification model (XRAI)*

1. Choose the **Color map**.
2. **Clip below** and **Clip above** parameters are used to reduce the noise. Enter **70** and **99.9** for click below and above respectively.
3. Select **Original** under **Overlay type** (pixels will be highlighted on top of the original image).
4. Enter **50** for the **Number of Integral steps** (increasing this parameter will reduce the approximation error).
5. Click **CONTINUE**.

Step 6: Compute and pricing

Follow the below mentioned steps to configure the budget for the model training:



Train new model

Training method

Model details

Training options

Explainability (optional)

Compute and pricing

START TRAINING CANCEL 2

Enter the maximum number of node hours that you want to spend training your model.

You can train for as little as 8 node hours. You may also be eligible to train with free node hours. Pricing guide

Budget *: 8 1 Maximum node hours

Estimated completion:

Enable early stopping

Ends model training when no more improvements can be made and refunds leftover training budget. If early stopping is disabled, training continues until the budget is exhausted.

Figure 10.9: *Compute and training for image classification model*

1. Set the minimum node hours for **8** (it is the minimum value for image data).
2. Click **START TRAINING**.

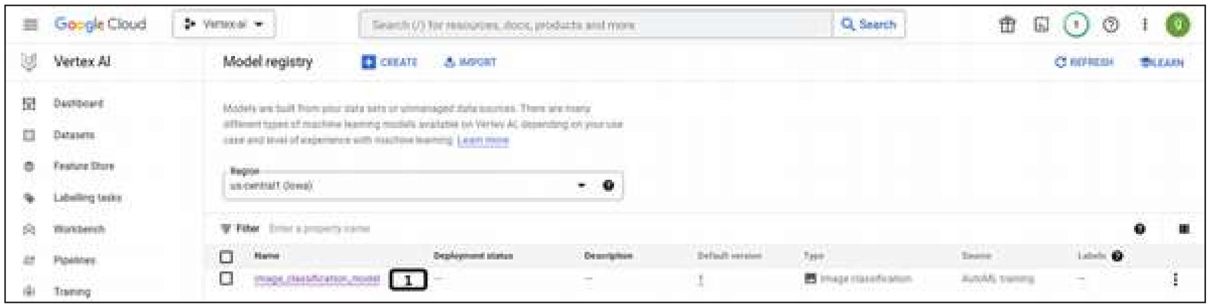
It will take a few hours to train the image classification model. Prediction and the explanation for the prediction can be obtained.

Image classification model deployment

Once the model is trained, it needs to be deployed to end point for online predictions. Also create a workbench to get the predictions (python workbench will suffice). Follow the steps mentioned in the chapter Vertex AI workbench & custom model training., for creation of the workbench (Python workbench will suffice). Follow the below mentioned steps for the deployment of the models.

Step 1: Trained model listed under Model registry

Follow the below mentioned step to deploy the trained model:



Models are built from your data sets or unmanaged data sources. There are many different types of machine learning models available on Vertex AI, depending on your use case and level of experience with machine learning. Learn more

Region: us-central1 (Iowa)

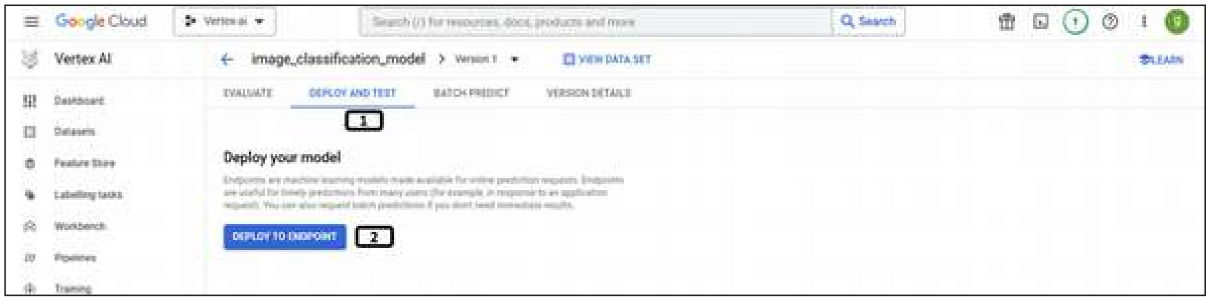
	Name	Deployment status	Description	Default version	Type	Source	Labels
☐	image_classification_model	–	–	1	Image classification	AutoML training	–

Figure 10.10: *Model registry*

1. Click the trained model and then click on the version 1 of the model.

Step 2: Deploy to end point

Once the model is selected (along with the version) users will get the option to evaluate the model, deploy and test the model, and so on. Follow the steps mentioned below to deploy the model:



Deploy your model

Endpoints are machine learning models made available for online prediction requests. Endpoints are useful for timely predictions from many users (for example, in response to an application request). You can also request batch predictions if you don't need immediate results.

Figure 10.11: *Image classification model deployment*

1. Click **DEPLOY AND TEST**.
2. Click **DEPLOY TO ENDPOINT**.

Step 3: Define end point

Follow the steps mentioned below to define the end point:

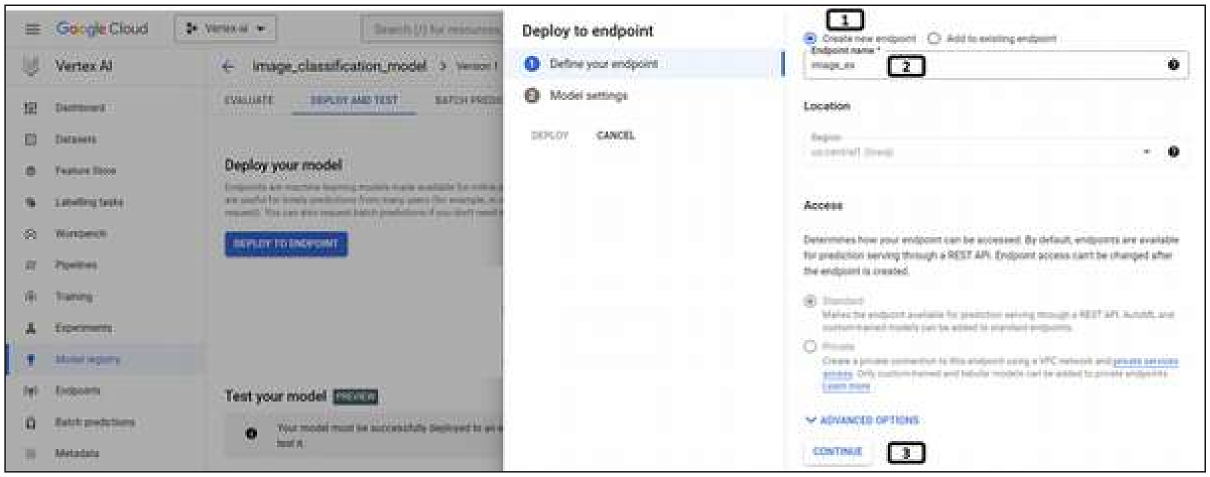


Figure 10.12: Image classification endpoint definition

1. Select **Create new endpoint**.
2. Provide the **Endpoint name**.
3. Click **CONTINUE**.

Step 4: Model settings

Follow the below mentioned steps to enable the explain ability of the model:

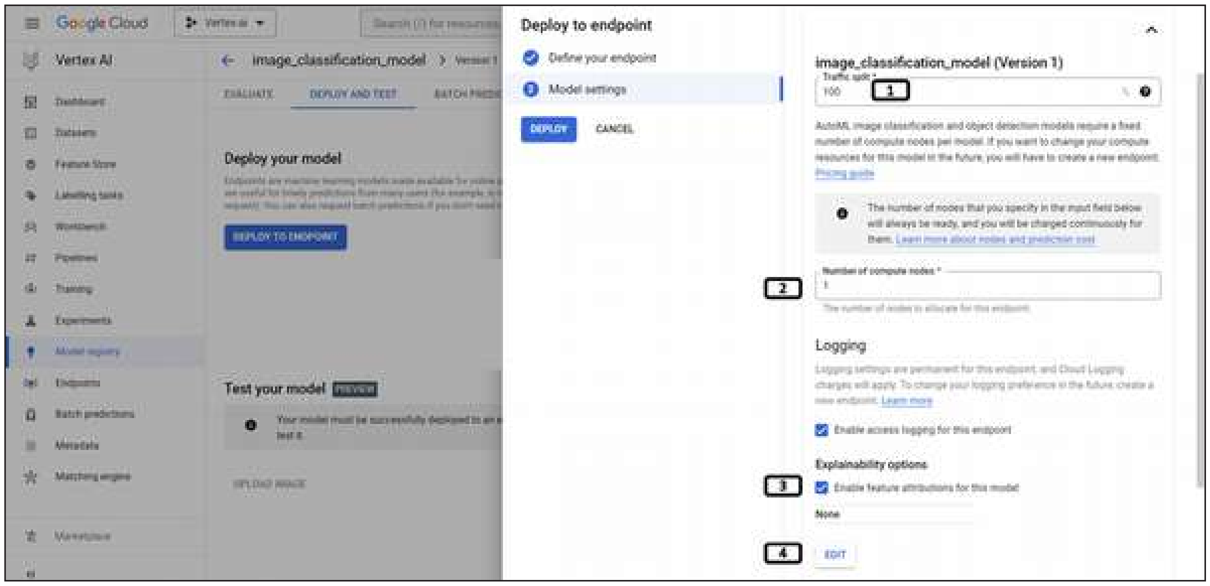


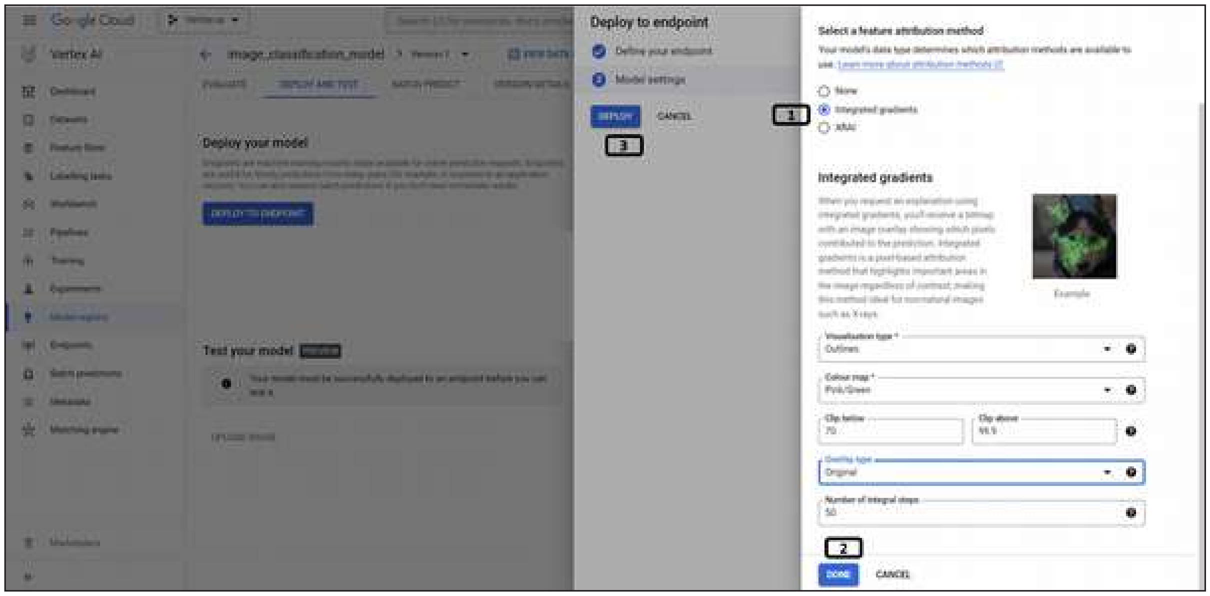
Figure 10.13: Image classification enabling explain ability

1. Set the **Traffic split** to **`100`**.
2. Set the **Number of compute nodes** for predictions to be **1**.

3. Enable the **Explainability options**.
4. Click **EDIT**.

Step 5: Feature attribution method selection

Follow the below mentioned steps to set the feature attribution method selection. In this example we are using Integrated gradients method for the explanations:



Visualisation type: Outlines
Colour map: Pink/Green
Clip below: 70
Clip above: 99.9
Overlay type: Original
Number of integral steps: 50

Figure 10.14: Image classification explain ability configuration

1. Select **Integrated gradients** method for **feature attribution method** (Keep the values same for the all the parameters, what was set during the training phase).
2. Click **DONE**.
3. Click **DEPLOY**.

Check if the model is deployed properly and then proceed with the python code to get predictions and explanations.

Explanations for image classification

Once the model is deployed successfully, open the Jupyter lab from the workbench created and enter the Python code given in the following steps:

Step 1: Install the required packages

Type the following Python code to install the required packages:

```
!pip install tensorflow
```

```
!pip install pip install google-cloud-aiplatform==1.12.1
```

Step 2: Kernel restart

Type following commands in the next cell, to restart the kernel: (Users can restart kernel from the GUI as well):

```
import os

import IPython

if not os.getenv(“”):

    Ipython.Application.instance().kernel.do_shutdown(True)
```

Step 3: Importing required packages

Once the kernel is restarted, run the following lines of codes to import the packages:

```
import base64

import tensorflow as tf

import google.cloud.aiplatform as gcai

import explainable_ai_sdk

import io

import matplotlib.image as mpimg

import matplotlib.pyplot as plt
```

Step 4: Input for prediction and explanation

Choose any image from the training set (stored in the cloud storage for the prediction) and provide the full path of the image chosen in the following code. Run the cell to read the image and covert the image to the required format:

```
img_input = tf.io.read_file(“gs://AutoML_image_data_exai/Kayak/adventure-
clear-water-exercise-1836601.jpg”)

b64str = base64.b64encode(img_input.numpy()).decode(“utf-8”)

instances_image = [{“content”: b64str}]
```

Step 5: Selection of the endpoint select

Run the following lines of code to select the endpoint where the model is deployed. In this method, we are using the display name of the endpoint (instead of the endpoint

ID). **`Image_ex`** is the endpoint name where the model is deployed. Full path of the endpoint (along with the endpoint ID) will be displayed in the output:

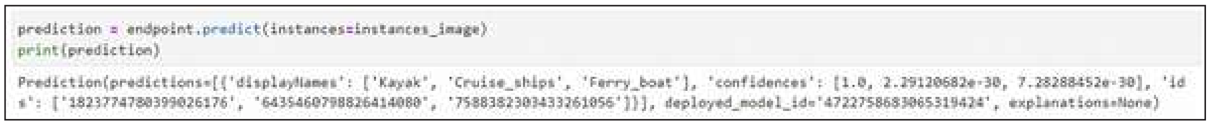
```
endpoint = gcai.Endpoint(gcai.Endpoint.list(

    filter=f'display_name={"image_ex"}',

    order_by='update_time')[-1].gca_resource.name)

print(endpoint)
```

Step 6: Image prediction

Run the following lines of code to get the prediction from the deployed model:

```
prediction = endpoint.predict(instances=instances_image)

print(prediction)
```

Prediction results will be displayed as shown in the following figure which contains display names and the probability of the classes:



```
prediction = endpoint.predict(instances=instances_image)
print(prediction)

Prediction(predictions=[{'displayNames': ['Kayak', 'Cruise_ships', 'Ferry_boat'], 'confidences': [1.0, 2.29120682e-30, 7.28288452e-30], 'id
s': ['1823774780399026176', '6435460798826414080', '7588382303433261056']}], deployed_model_id='4722758683065319424', explanations=None)
```

Figure 10.15: *Image classification prediction result*

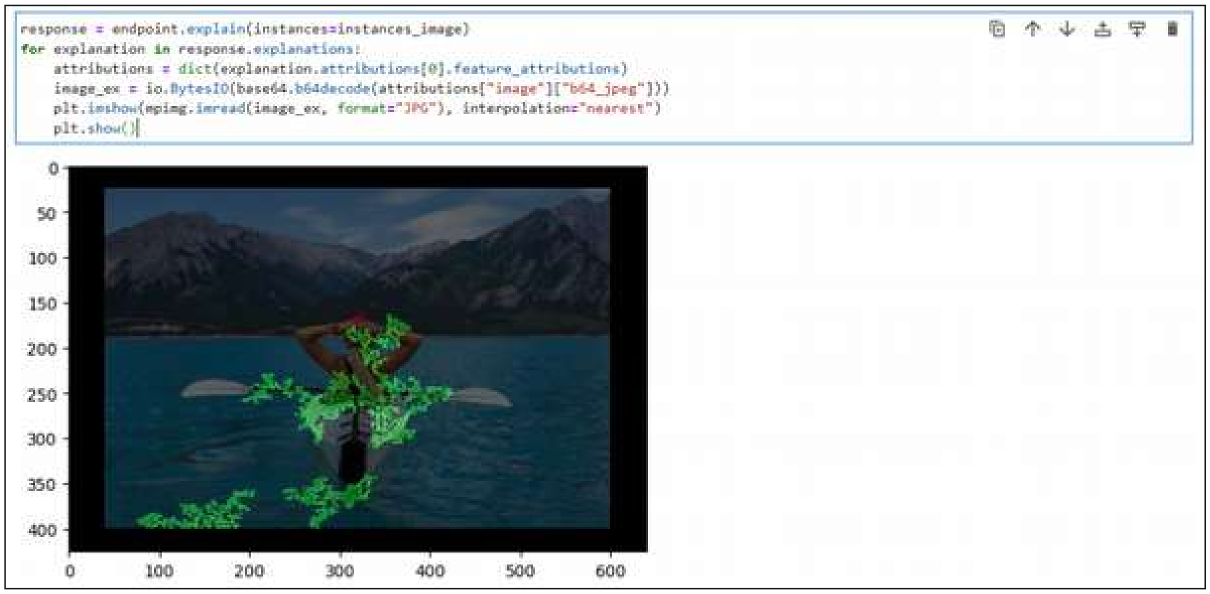
NOTE: Since we are running this code using the Vertex AI workbench, we are not using service account for authentication.

Step 7: Explanations

Run the following lines of codes to get the explanations for the input image:

```
response = endpoint.explain(instances=instances_image)

for explanation in response.explanations:

   attributions = dict(explanation.attributions[0].feature_attributions)

    image_ex = io.BytesIO(base64.b64decode(attributions["image"]["b64_
    jpeg"]))

    plt.imshow(mpimg.imread(image_ex, format="JPG"),
    interpolation="nearest")

    plt.show()
```

The output of the explanations is shown in the figure. Highlighted areas in green indicates areas/pixels which played important role for the prediction of the image:



```
response = endpoint.explain(instances=instances_image)
for explanation in response.explanations:
    attributions = dict(explanation.attributions[0].feature_attributions)
    image_ex = io.BytesIO(base64.b64decode(attributions["image"]["b64_jpeg"]))
    plt.imshow(mpimg.imread(image_ex, format="JPG"), interpolation="nearest")
    plt.show()
```

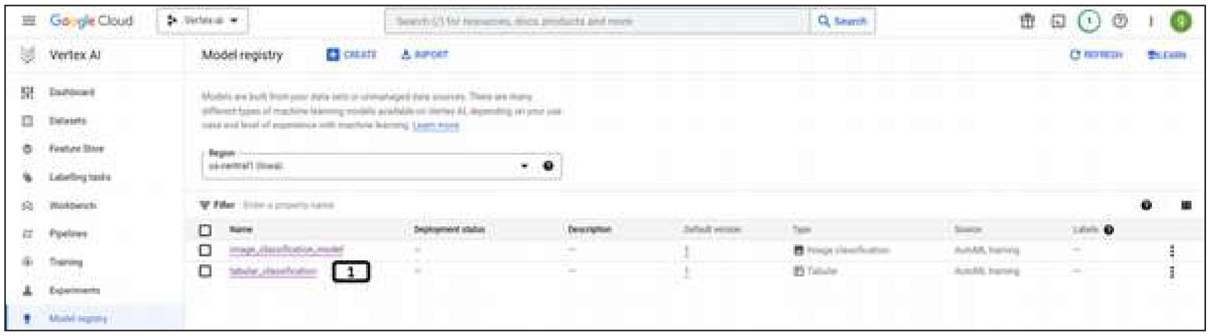
***Figure 10.16**: Image classification model explanation*

Tabular classification model deployment

In case of the image data, users had to configure explainable AI during the training and deployment phase, whereas, in case of tabular data, explainable AI needs to be configured only during the deployment phase (AutoML will enable the explainable AI by default during the training phase for the tabular data). Follow the steps mentioned in *Chapter 2, Introduction to Vertex AI & AutoML Tabular* for the tabular dataset creation and tabular AutoML model training. Follow the below mentioned steps for the model deployment of the trained model.

Step 1: Trained model in the model registry

Trained model will be listed in the model registry as shown in the following figure:



Name	Deployment status	Description	Default version	Type	Source	Labels
image_classification_model	–	–	1	Image classification	AutoML training	–
tabular_classification	–	–	1	Tabular	AutoML training	–

***Figure 10.17:** Model registry (tabular classification model)*

1. **tabular_classification** trained using AutoML. Click on the model and the version of the same.

Step 2: Deploy to end point

Once the model is selected (along with the version) users will get option to evaluate the model, deploy and test the model, and so on. Follow the steps mentioned below to deploy the model:

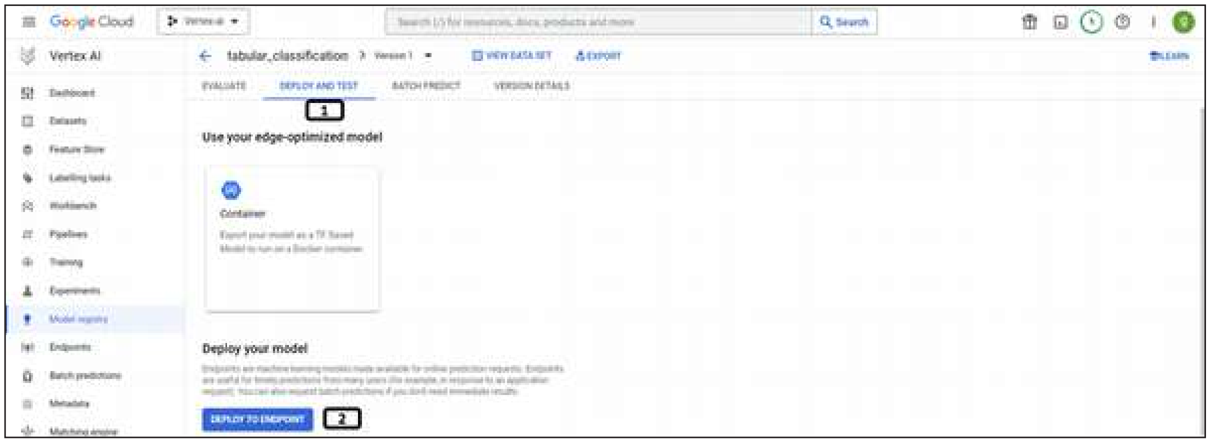


Figure 10.18: Trained tabular classification model

1. Select **DEPLOY AND TEST** tab.
2. Click **DEPLOY TO ENDPOINT**.

Step 3: Endpoint definition

Follow the steps mentioned below to define the endpoint:

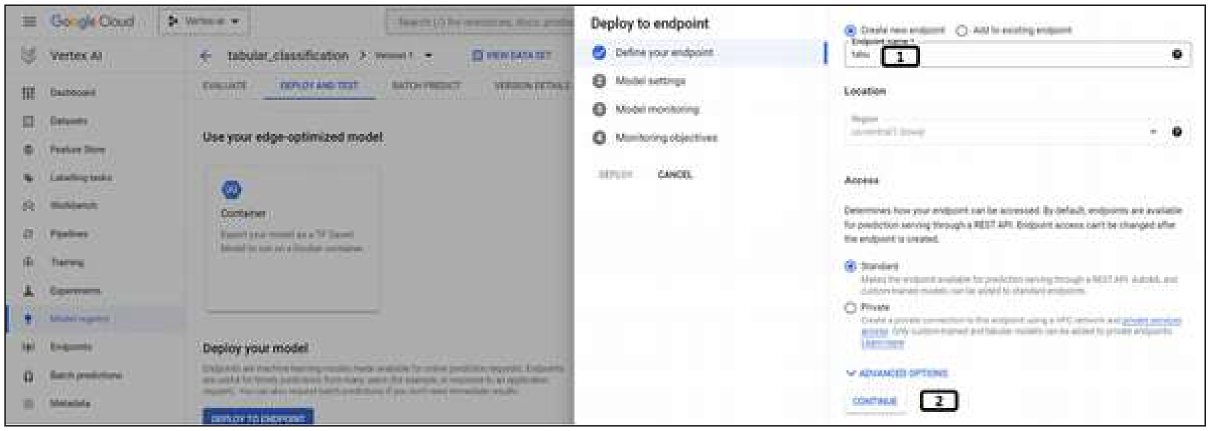
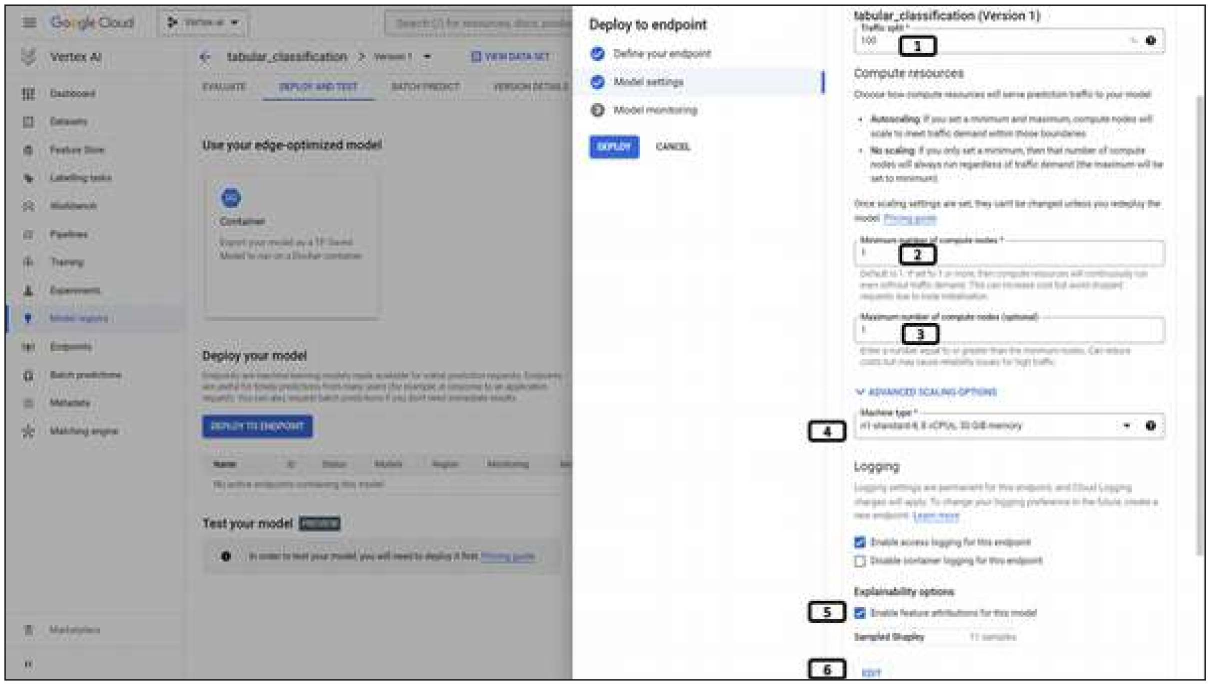


Figure 10.19: Endpoint definition tabular classification model

1. Provide **Endpoint name**.
2. Click **CONTINUE**.

Step 4: Model settings

Follow the steps mentioned below to configure the model settings and to enable the explain ability options:



tabular_classification (Version 1)

Traffic split: 100

Compute resources

Minimum number of compute nodes: 1

Maximum number of compute nodes (optional): 1

ADVANCED SCALING OPTIONS

Machine type: n1-standard-8, 8 vCPUs, 30 GiB memory

Logging

☑ Enable access logging for this endpoint

☐ Disable container logging for this endpoint

Explainability options

☑ Enable feature attributions for this model

Sampled Shapley 11 samples

Figure 10.20: Model settings (enabling explain ability)

1. Set the **Traffic split** to **`100`**.
2. Set the **Minimum number of compute nodes** to **`1`**.
3. Set the **Maximum number of compute nodes** to **`1`**.
4. Select **`n1-standard-8`** in **Machine type**.
5. Enable the **Explainability options**.
6. Click **EDIT**.

Step 5: Set the Explainability options

You can set the Explainability options by following the steps shown in the following figure:

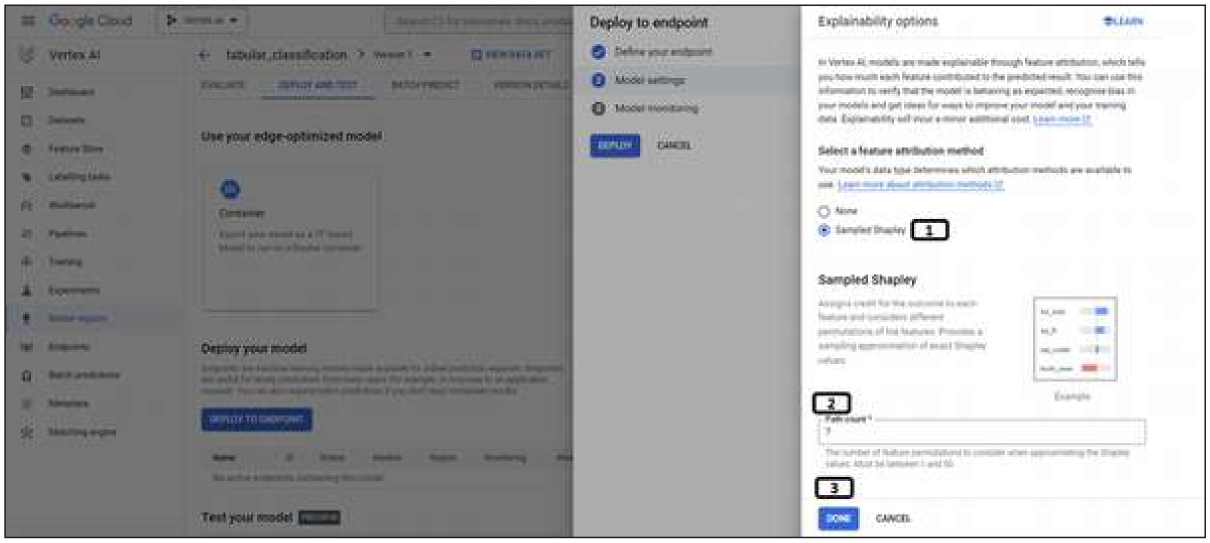


***Figure 10.21**: Sampled Shapley path count*

1. Select **Sampled Shapley** method.
2. Set the **Path count** to **7** (randomly chosen).
3. Click **DONE**.

Step 6: Model monitoring

Follow the below mentioned steps to disable the model monitoring (since it is not needed for the explanations):

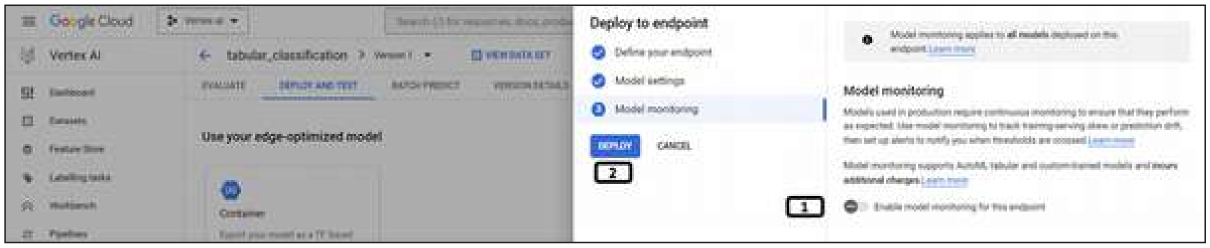


***Figure 10.22**: Model monitoring*

1. Disable **Model monitoring** options.
2. Click **DEPLOY**.

Explanations for tabular data (classification)

Once the model is deployed successfully, open the Jupyter lab from the workbench created and enter the Python code given in the below steps.

Step 1: Input for prediction and explanation

Select any record from the data. Modify it in the below mentioned format and run the cell:

```
instances_tabular=[{"BMI":"16.6","Smoking":"Yes","AlcoholDrinking":"-
No","Stroke":"No","PhysicalHealth":"3","MentalHealth":"30","Diff-
Walking":"No","Sex":"Female","AgeCategory":"55-59","Race":"White","-
Diabetic":"Yes","PhysicalActivity":"Yes","GenHealth":"Very
good","SleepTime":"5","Asthma":"Yes","KidneyDisease":"No","SkinCan-
cer":"Yes"}]
```

Step 2: Selection of the endpoint select

Run the below lines of code to select the endpoint where the model is deployed. In this method, we are using the display name of the endpoint (instead of the endpoint ID). **"tabu"** is the endpoint name where the model is deployed. Full path of the endpoint (along with the endpoint ID) will be displayed in the output:

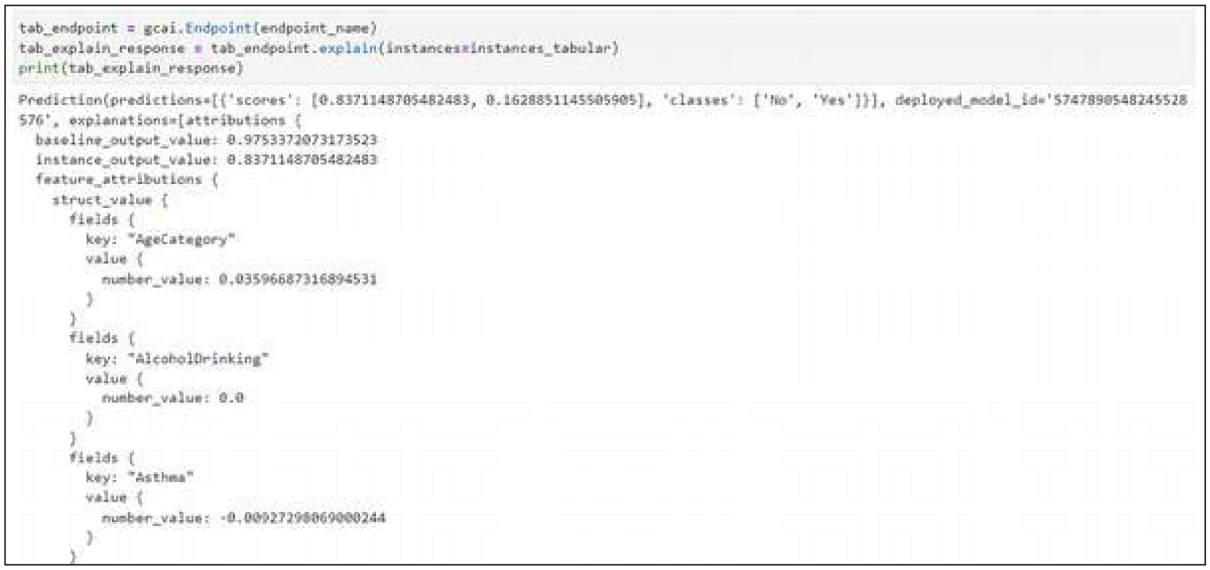
```
endpoint_tabular = gcai.Endpoint(gcai.Endpoint.list(
    filter=f'display_name={"tabu"}',
    order_by='update_time')[-1].gca_resource.name)
print(endpoint_tabular)
```

Step 3: Prediction

Run the following lines of code to get the prediction from the deployed model:

```
tab_endpoint = gcai.Endpoint(endpoint_name)
tab_explain_response = tab_endpoint.explain(instances=instances_tabular)
print(tab_explain_response)
```

Prediction results will be displayed as shown in the following figure which contains classes and the probability of the classes:



```
tab_endpoint = gcai.Endpoint(endpoint_name)
tab_explain_response = tab_endpoint.explain(instances=instances_tabular)
print(tab_explain_response)
```

```
Prediction(predictions=[{'scores': [0.8371148705482483, 0.1628851145505905], 'classes': ['No', 'Yes']}], deployed_model_id='5747890548245528
576', explanations=[attributions {
  baseline_output_value: 0.9753372073173523
  instance_output_value: 0.8371148705482483
  feature_attributions {
    struct_value {
      fields {
        key: "AgeCategory"
        value {
          number_value: 0.03596687316894531
        }
      }
      fields {
        key: "AlcoholDrinking"
        value {
          number_value: 0.0
        }
      }
      fields {
        key: "Asthma"
        value {
          number_value: -0.00927298069000244
        }
      }
```

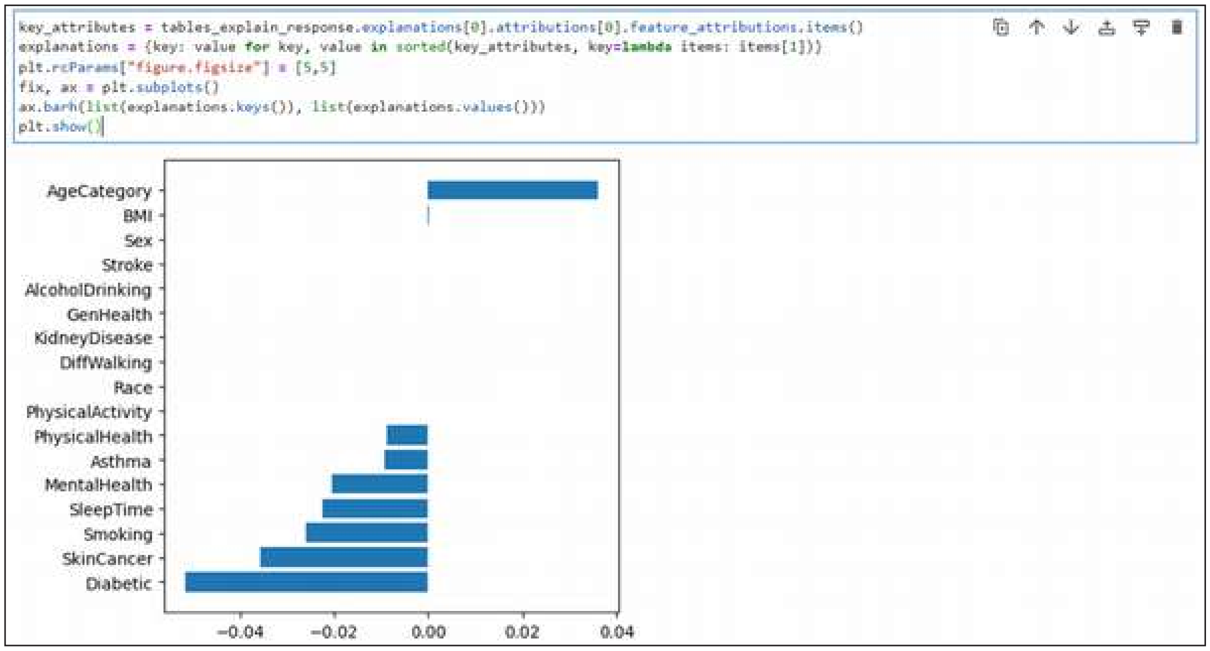
Figure 10.23: Predictions from deployed tabular classification model

Step 4: Explanations

Run the following lines of codes to get the explanations for the input record:

```
key_attributes = tables_explain_response.explanations[0].attributions[0].
feature_attributions.items()

explanations = {key: value for key, value in sorted(key_attributes,
key=lambda items: items[1])}

plt.rcParams["figure.figsize"] = [5,5]

fix, ax = plt.subplots()

ax.barh(list(explanations.keys()), list(explanations.values()))

plt.show()
```

Shapley value is provided in the explanations for each of the features and it is visualized as shown in the following figure:



```
key_attributes = tables_explain_response.explanations[0].attributions[0].feature_attributions.items()
explanations = {key: value for key, value in sorted(key_attributes, key=lambda items: items[1])}
plt.rcParams["figure.figsize"] = [5,5]
fix, ax = plt.subplots()
ax.barh(list(explanations.keys()), list(explanations.values()))
plt.show()
```

Figure 10.24: Explanations from deployed tabular classification model

Deletion of resources

We have utilized cloud storage to store the data, delete the files from the cloud storage manually. Dataset is created for image data and tabular data to delete them manually. Classification models for image and tabular are deployed to get the predictions and explanations, ensure to un-deploy the model from the endpoints and delete the endpoints (Refer *Chapter 2, Introduction to Vertex AI & AutoML Tabular* and *Chapter 3, AutoML Image, text and pre-built models*). Predictions are obtained using workbench, ensure to delete the workbench instance.

Limitations of Explainable AI

Following are some of the limitations of Explainable AI in Vertex AI:

- Each attribution merely displays how much the attribute influenced the forecast for that case. A single attribution may not represent model behavior. Aggregate attributions is preferred over a dataset to understand approximate model behavior.
- Model and data determine attributions. They can only show the model's data patterns, not any underlying linkages. The target's association with a feature does not depend on its strong attribution. The attribution indicates if the model predicts using the characteristic.
- Attributions alone cannot determine quality of the model; it is recommended to consider assessment of the training data and evaluation metrics of the model.
- Integrated gradients method works well for the differentiable models (where derivative of all the operations can be calculated in TensorFlow graph). Shapley method is used for the Non-differentiable models (non-differentiable operations in the TensorFlow network, such as rounding operations and decoding).

Conclusion

In this book, we started by understanding the cloud platform, a few important components of the cloud, and the advantages of the cloud platforms. We started working development of the machine learning models through Vertex AI AutoML for tabular, text, and image data, we deployed the trained models onto the endpoints for the online predictions. Even before entering into the complexity of the custom model building, we worked to understand how to leverage pre-build models of the platform to obtain predictions. For the custom models, we utilized a workbench for the code development for the model training and utilized docker images to submit the training jobs, also worked on the hyperparameter tuning to further enhance the model performance using Vizier. We worked on the pipeline components of the platform to train the model and evaluate and deploy the model for online predictions using both Kubeflow and TFX. We worked on creating a centralized repository for the features using the feature store of the Vertex AI. This is the last chapter of the book, where we learned about the explainable AI, need of it. We trained the AutoML classification model for image and tabular data for the explanations and obtained the explanations using the Python code. GCP is adding lot of new components and features to enhance its capability, check the platform (documentation of the platform) regularly to keep yourself updated.

Questions

1. Why explainable AI is important?
2. What are the different types of explanations supported by Vertex AI?
3. What is the difference between example based and feature based examples?

Index

A

Apache Beam 16
application programming interfaces (APIs) 14
Artifact Registry 151
AutoML
  benefits 138
  limitations 138, 139

B

batch prediction 80
  creating 80, 81
  output 81
batch predictions image
  initializing 112
  input, preparing 111
  output 112, 113
batch predictions text 128-130
BigQuery 15, 47
  data selection, for table 51-53
  dataset creation 50, 51
  dataset deletion 54
  landing page 48-50
  query, running on table 53, 54
  table creation 51
  working with 48
Bigtable 15

C

central processing unit (CPU) 34
cloud 2
  advantages 3-5
  hybrid cloud 7
  private cloud 6, 7
  public cloud 6
  significance, for data scientist 5, 6
cloud service models 12
  Data as a service (DaaS) 14
  Infrastructure as a service (IaaS) 13
  Platform as a service (PaaS) 13
  Software as a service (SaaS) 13
command line interface (CLI) 34
components, Kuberflow
  characteristics 206
  Containerized Python components 206
  custom container components 206
  Lightweight Python function-based components 206
components, TFX 253
Compute Engine
  API, enabling 39
  creating 38
  opening 39

VM instance creation 40, 41
VM instance creation access settings 43
VM instance disk and container selection 42
VM instance selection 40
compute services 34, 38
container registry 150
image, pushing to 161, 162
Containers 148
Coordinated Universal Time (UTC) 38
custom components, TFX
container-based components 254
fully custom components 254
functionalities 254-256
Python function based components 254
customer managed encryption keys (CMEK) 32
custom model training job
hyperparameter tuning 165

D

data analytics
Dataflow 16
Dataproc 16
Pub/Sub 16
Data as a service (DaaS) 14
database deletion 94
Dataflow 16
Dataprep 16
Dataproc 16
datasets 65
Directed Acyclic Graphs (DAG) 252
Docker 148
components 148
Dockerfile 148
creating 149-152
editing 152, 153
Docker Hub 149
Docker image 148
Docker run 149
Domain-Specific Language (DSL) 249

E

evaluator component 256
ExampleGen TFX pipeline component 254
ExampleValidator TFX pipeline component 255
Explainable AI (XAI) 299, 300
on Vertex AI 302
purpose 300, 301

F

feature attribution methods, XAI
integrated gradients method 303
sampled Shapely method 303
XRAI method 304
Filestore 25
Firebase Realtime Database 15
Firestore 15
function as a service (FaaS) 37

G

GCP CLI 22, 23
GCP Hierarchy 17, 18
project creation 18-20
project deletion 20, 21
GCP services 14
AI and ML services 16
APIs 23
BigQuery 15
Cloud Build 16
Cloud Operations suite 16
Cloud SQL 15
Cloud Storage 15
Command Line Interface 22, 23
data analytics 16
Google Cloud Platform Console 21, 22
Identity and Access Management 16
interacting with 21
NoSQL services 15
Virtual Private Cloud (VPC) 16
Google APP Engine 36

Flexible APP Engine 37
Standard APP Engine 37
Google BigQuery
architecture 47
Google Cloud Function 37, 38
Google Cloud Persistent Disk 24
Google Cloud Platform Console 21
Google Cloud Platform (GCP) 1, 7
account, creating 8-10
compute services 38
footprint 11, 12
global resources 12
hierarchy 17, 18
identity 56
multiregional resources 12
pre-built models 135-137
regional resources 12
services 14, 15
zonal resources 12
Google Cloud Storage 25, 26
access control, for buckets 31
Archive storage class 26
bucket creation 27, 28
bucket deletion 34
bucket location, selecting 29
bucket name, providing 28, 29
data protection, in bucket 32
files, uploading to bucket 33
Lowline storage class 26
Nearline storage class 26
opening 27
Standard class 26
storage class, selecting 30
working with 27
Google Compute Engine (GCE) 34
Google Compute Services 34
APP Engine 36, 37
Compute Engine 34, 35
Kubernetes Engine 35, 36
Google Kubernetes Engine (GKE) 35
graphics processing unit (GPU) 34

H

hybrid cloud 7
hyperparameter
in machine learning 172, 173
hyperparameters tuning
working 173
Hypervisor 34

I

IAM policy 57
Identity and Access Management (IAM) 16, 55
basic roles 55
custom roles 56
predefined roles 56
identity, GCP 56
Alias 57
Cloud Identity domain 56
default service accounts 56
Google group 56
G suite domain 56
service account 56
user account 56
user-managed service accounts 56
image dataset creation 101
data importing 103, 104
data selection 102, 103
objective selection 102
Infrastructure as a service (IaaS) 13
InfraValidator TFX component 256

K

Kubeflow 205
components 206
tasks 206, 207
Kubeflow Pipelines (KFP) 206, 249

L

landing page components, Vertex AI
batch predictions 64

Datasets 64
endpoints 64
Experiments 64
Feature Store 64
labels 64
Matching Engine 65
metadata 65
models 64
Pipelines 64
training 64
Vertex AI dashboard 64
Workbench 64

M

machine learning pipeline 204
benefits 205
execution 205
model versioning and tracking 205
resource utilization 205
troubleshooting 205
Memorystore 15
model deletion 94
model deployment
Endpoint definition 82, 83
for online predictions 81, 82
model monitoring 83, 84
model monitoring objectives 84, 85
model settings 83
testing 85, 86
model evaluation 76
confusion matrix and feature importance 78, 79
model training, checking 76, 77
model training results 77
model version details 79
performing 77, 78
model evaluation image 109
confusion matrix 110
model performance 109, 110
model evaluation text 127, 128
model training 71
compute and pricing 75, 76
data splitting 73
method selection, training 72
model details selection 72, 73
optimization objectives, selecting 74, 75
preparation 71
training options 74
model training image 105
completing 108
compute and pricing 107
explainability 106
model details 105, 106
model registry 108, 109
model training method 105
model version selection 109
model training text 124-127
model un-deployment
endpoint detection 93, 94
endpoint, selecting 92, 93
model, removing 93

N

network-attached storage (NAS) 25

O

online predictions
model deployment 81-86
python code output 92
serving 90
online predictions image
model deployment 114-116
serving 116-118
online predictions text
model deployment 131-133
serving 133, 135

P

Persistent Disks 24
pipelines for custom model, using Kubeflow
additional permissions, granting 229-231
code walk through 232-243

data, for model training 228
pipeline comparison 247, 248
pipeline job, analyzing 243-247
resources, deleting 249
workbench, creating 231, 232
Platform as a service (PaaS) 13
pre-built models
in GCP 135-137
private cloud 6, 7
public cloud 6
Pub/Sub 16
Pusher TFX component 256

R

regions 11
ROLE 55
Role Based Access Control (RBAC) 55

S

sampled Shapley technique 303
SchemaGen TFX pipeline component 255
Serverless Spark tool 62
service account creation 86-88
keys, adding 90
keys, managing 89
roles, assigning 88
Service Level Agreements (SLAs) 6
Software as a service (SaaS) 13
Software Development Kit (SDK) 23
StatisticsGen TFX pipeline component 255
storage 24
storage options, Google Cloud
Cloud Storage 25, 26
Filestore 25
Persistent Disks 24

T

tabular datasets
creating 66-69
statistics, generating 70
statistics generation results 70, 71
Tabular workflows 61
Task 206
tasks, Kuberflow 206
explicitly dependent tasks 207
implicitly dependent tasks 207
independent tasks 207
TensorFlow Extended 252
text dataset creation 121-124
TFX pipeline 252
code walk through 257-272
components 253
custom components 254-256
data for pipeline building 256
resources, deleting 272
Traffic Splitting 37
Trainer TFX component 255
Training Reduction Server 61
Transform TFX pipeline component 255
Tuner TFX component 255

V

Vertex AI 60
key features 61
landing page 63-65
prediction service 62
working with 62, 63
Vertex AI AutoML 65
datasets, benefits 66
data types 65, 66
for image data 98-101
for text data 119-121
Vertex AI Feature Store 276
advantages 277, 278
best practices 297, 298
data, for exercise 279
disadvantages 278
history 276, 277
Vertex AI Feature Store, with GUI
creating 281
entity type, creating 282, 283

features, creating 283, 284
feature values ingestion 285-287
landing page 280
working on 279, 280
Vertex AI Feature Store, with Python
connecting 289
creating 290
entity type, creating 291
feature, creating 292, 293
feature value ingestion 294, 295
feature values, fetching 295, 296
installed packages, importing 288
kernel restart 288
package installation 288
project, setting up 289
Python notebook file, creating 287
resources, deleting 297
working on 287
Vertex AI pipelines 204, 249
API enablement 208-210
code walk through 212-220
data, for model training 207, 208
executing 221-224
permission, for compute engine 210-212
resources, deleting 225
Vertex AI Vizier 173, 174
custom model training job completion 189, 190
custom model training job, submitting 183-188
data, for building custom model 174
Dockerfile, creating 176, 177
image creation 182, 183
model building code 177-181
model deployment 193-195
model importing 190-193
online predictions 196
resources, deleting 201
training job, submitting with Python SDK 198-200
workbench, creating 175
Vertex AI Workbench 142, 143
creating 143-147
custom model training job completion 167, 168
custom model training job, submitting 162-167
data, for building custom model 147, 148
Dockerfile, creating 149-153
image creation, with docker build 158-160
image, pushing to container registry 161, 162
model building 154-158
resources, deleting 168
Virtual Private Cloud (VPC) 16
VM instances 38
accessing 44-46
creating 38
deleting 46

X

XAI, on Vertex AI 302
data, for exercise 304
example-based explanations 302
feature attribution methods 303
feature-based explanations 303
image classification model deployment 310-315
limitations 321
model training, for image data 305-309
region selection 304
tabular classification model deployment 315-320
XRAI approach
over segmentation 304
pixel-level attribution 304

Learning Google Cloud Vertex AI

DESCRIPTION

Google Cloud Vertex AI is a platform for machine learning (ML) offered by Google Cloud, with the objective of making the creation, deployment, and administration of ML models on a large scale easier. If you are seeking a unified and collaborative environment for your ML projects, this book is a valuable resource for you.

This comprehensive guide is designed to help data enthusiasts effectively utilize Google Cloud Platform's Vertex AI for a wide range of machine learning operations. It covers the basics of the Google Cloud Platform, encompassing cloud storage, big query, and IAM. Subsequently, it delves into the specifics of Vertex AI, including AutoML, custom model training, model deployment on endpoints, development of Vertex AI pipelines, and the Explainable AI feature store.

By the time you finish reading this book, you will be able to navigate Vertex AI proficiently, even if you lack prior experience with cloud platforms. With the inclusion of numerous code examples throughout the book, you will be equipped with the necessary skills and confidence to create machine learning solutions using Vertex AI.

KEY FEATURES

- Harness the power of AutoML capabilities to build machine learning models.
- Learn how to train custom machine learning models on the Google Cloud Platform.
- Accelerate your career in data analytics by leveraging the capabilities of GCP.

WHAT YOU WILL LEARN

- Learn how to create projects, store data in GCP, and manage access permissions effectively.
- Discover how AutoML can be utilized for streamlining workflows.
- Learn how to construct pipelines using TFX (TensorFlow Extended) and Kubeflow components.
- Gain an overview of the purpose and significance of the Feature Store.
- Explore the concept of explainable AI and its role in understanding machine learning models.

WHO THIS BOOK IS FOR

This book is designed for data scientists and advanced AI practitioners who are interested in learning how to perform machine learning tasks on the Google Cloud Platform. Having prior knowledge of machine learning concepts and proficiency in Python programming would greatly benefit readers.

BPB PUBLICATIONS
www.bpbonline.com

ISBN 978-93-5551-535-3

9 789355 515353