# GCP steps:

[GCP steps:](#_d9o1vzqfi9kn)

[Steps 0: Create a Github repo](#_s91j8oa2fqez)

[Step 1: Create a Cloud account](#_xvl4vhx43eg)

[Step 2: Upload file to Cloud Storage and create service accounts](#_vy7a8efklyuu)

[Step 3: Python ETL to load data from Cloud Storage to Cloud SQL](#_7ycdv970bgzd)

[Step 4: Python to load data from Cloud SQL to BigQuery (as data will grow, they will need to be in Data Warehouse)](#_hp5fxqfpx2o8)

[Step 5: Terraform code to automate process from Storage to Cloud SQL uploading](#_3ksej1brz3wd)

[Step 6: Python code to use DLP (Data Loss prevention as this is sensitive data like gender, email, ip address)](#_f1hz23gw0bn)

[Step 7: ML model - Built on Vertex AI (Google Cloud)](#_o1r7e56i8lkf)

[K-Means clustering](#_agry9mujoaq)

[#add a new column in the dataset](#_gjdgxs)

[#Print the dataset with the predicted clusters](#_30j0zll)

[GMM](#_57awhexmuxk2)

[Step 8: Deploy the ML model as API endpoint - I used Vertex AI Platform prediction Service](#_f4hg1ghj73df)

# **Steps 0:** Create a Github repo

<https://github.com/cs-cloud-store/assessment>

# **Step 1:** Create a Cloud account

Google Cloud account with $300 free trial band set up billing.

# **Step 2:** Upload file to Cloud Storage and create service accounts

**Service accounts code:**

gcloud projects add-iam-policy-binding $GOOGLE\_CLOUD\_PROJECT \

--member serviceAccount:https-console-cloud-google-com@hale-brook-377621.iam.gserviceaccount.com \

--role roles/dlp.admin

gcloud projects add-iam-policy-binding $GOOGLE\_CLOUD\_PROJECT \

--member serviceAccount:https-console-cloud-google-com@hale-brook-377621.iam.gserviceaccount.com \

--role roles/dlp.serviceAgent

gcloud projects add-iam-policy-binding $GOOGLE\_CLOUD\_PROJECT \

--member serviceAccount:service-`gcloud projects list --filter="PROJECT\_ID:$GOOGLE\_CLOUD\_PROJECT" --format="value(PROJECT\_NUMBER)"`@dlp-api.iam.gserviceaccount.com \

--role roles/viewer

**Cloud Storage bucket**

# **Step 3:** Python ETL to load data from Cloud Storage to Cloud SQL

import pandas as pd

import mysql.connector

from google.cloud import storage

# Set up connection to Google Cloud Storage

client = storage.Client()

bucket = client.get\_bucket('your-bucket-name')

# Load data from CSV file in GCS into a pandas dataframe

blob = bucket.blob('your-file.csv')

data = blob.download\_as\_string()

df = pd.read\_csv(io.BytesIO(data))

# Transform data (if needed)

# For example, you could concatenate first\_name and last\_name into a new column called full\_name:

df['full\_name'] = df['first\_name'] + ' ' + df['last\_name']

# Set up connection to MySQL Cloud SQL

cnx = mysql.connector.connect(user='your-db-user', password='your-db-password',

host='your-db-host', database='your-db-name')

# Create cursor and table if not exists

cursor = cnx.cursor()

create\_table\_query = '''

CREATE TABLE IF NOT EXISTS customers (

id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(255),

gender VARCHAR(10),

ip\_address VARCHAR(50),

full\_name VARCHAR(100)

)

'''

cursor.execute(create\_table\_query)

cnx.commit()

# Load data into MySQL Cloud SQL

for index, row in df.iterrows():

insert\_query = '''

INSERT INTO customers (id, first\_name, last\_name, email, gender, ip\_address, full\_name)

VALUES (%s, %s, %s, %s, %s, %s, %s)

'''

values = (row['id'], row['first\_name'], row['last\_name'], row['email'], row['gender'], row['ip\_address'], row['full\_name'])

cursor.execute(insert\_query, values)

cnx.commit()

# Close cursor and connection

cursor.close()

cnx.close()

# **Step 4:** Python to load data from Cloud SQL to BigQuery (as data will grow, they will need to be in Data Warehouse)

import pandas as pd

import mysql.connector

from google.cloud import bigquery

from google.cloud import storage

# Set up connection to Google Cloud Storage

client = storage.Client()

bucket = client.get\_bucket('your-bucket-name')

# Load data from CSV file in GCS into a pandas dataframe

blob = bucket.blob('your-file.csv')

data = blob.download\_as\_string()

df = pd.read\_csv(io.BytesIO(data))

# Transform data (if needed)

# For example, you could concatenate first\_name and last\_name into a new column called full\_name:

df['full\_name'] = df['first\_name'] + ' ' + df['last\_name']

# Set up connection to GCP Cloud SQL

cnx = mysql.connector.connect(user='your-db-user', password='your-db-password',

host='your-db-host', database='your-db-name')

# Load data into a pandas dataframe

df = pd.read\_sql('SELECT \* FROM customers', con=cnx)

# Set up connection to BigQuery

client = bigquery.Client()

table\_ref = client.dataset('your\_dataset\_name').table('your\_table\_name')

# Create job configuration and load data into BigQuery

job\_config = bigquery.LoadJobConfig()

job\_config.write\_disposition = bigquery.WriteDisposition.WRITE\_TRUNCATE

job\_config.source\_format = bigquery.SourceFormat.CSV

job\_config.schema = [

bigquery.SchemaField('id', 'INTEGER'),

bigquery.SchemaField('first\_name', 'STRING'),

bigquery.SchemaField('last\_name', 'STRING'),

bigquery.SchemaField('email', 'STRING'),

bigquery.SchemaField('gender', 'STRING'),

bigquery.SchemaField('ip\_address', 'STRING'),

bigquery.SchemaField('full\_name', 'STRING')

]

job = client.load\_table\_from\_dataframe(df, table\_ref, job\_config=job\_config)

job.result() # Wait for job to complete

# Close connection to GCP Cloud SQL

cnx.close()

# **Step 5:** Terraform code to automate process from Storage to Cloud SQL uploading

# Configure provider

provider "google" {

project = "hale-brook-377621"

region = "your-region"

zone = "your-zone"

}

# Configure storage bucket

resource "google\_storage\_bucket" "data\_bucket" {

name = "your-bucket-name"

}

# Configure Cloud SQL instance

resource "google\_sql\_database\_instance" "db\_instance" {

name = "your-instance-name"

region = "your-region"

database\_version = "MYSQL\_5\_7"

settings {

tier = "db-f1-micro"

}

}

# Configure Cloud SQL database and user

resource "google\_sql\_database" "db" {

name = "your-db-name"

instance = google\_sql\_database\_instance.db\_instance.name

}

resource "google\_sql\_user" "db\_user" {

name = "your-db-user"

password = "your-db-password"

instance = google\_sql\_database\_instance.db\_instance.name

host = "%"

}

# Configure Cloud Function to run ETL code

resource "google\_cloudfunctions\_function" "etl\_function" {

name = "your-function-name"

description = "ETL function that ingests, transforms, and loads data from GCS to Cloud SQL"

source\_archive\_bucket = google\_storage\_bucket.data\_bucket.name

source\_archive\_object = "etl\_function.zip"

entry\_point = "main"

runtime = "python37"

environment\_variables = {

SQL\_USER = google\_sql\_user.db\_user.name

SQL\_PASSWORD = google\_sql\_user.db\_user.password

SQL\_HOST = google\_sql\_database\_instance.db\_instance.ip\_address

SQL\_DATABASE = google\_sql\_database.db.name

GCS\_BUCKET = google\_storage\_bucket.data\_bucket.name

}

timeout = "180s"

available\_memory\_mb = 256

trigger\_http = true

}

# Configure Cloud Scheduler job to run Cloud Function daily

resource "google\_cloud\_scheduler\_job" "etl\_job" {

name = "your-job-name"

description = "Schedule ETL function to run daily at 7AM"

schedule = "0 7 \* \* \*"

target\_type = "google\_cloudfunctions\_function"

target\_http\_method = "POST"

target\_uri = google\_cloudfunctions\_function.etl\_function.https\_trigger\_url

}

# **Step 6:** Python code to use DLP (Data Loss prevention as this is sensitive data like gender, email, ip address)

import google.cloud.dlp

from google.cloud import dlp\_v2

# Instantiates a DLP client

dlp = dlp\_v2.DlpServiceClient()

# Define the info types to be de-identified

info\_types = [{"name": "PERSON\_NAME"}, {"name": "EMAIL\_ADDRESS"}, {"name": "IP\_ADDRESS"}]

# Define the de-identification transformation

deidentify\_config = {

"record\_transformations": {

"field\_transformations": [

{

"fields": [{"name": "first\_name"}, {"name": "last\_name"}, {"name": "email"}],

"primitive\_transformation": {

"character\_mask\_config": {

"masking\_character": "\*",

"number\_to\_mask": 0.6,

"characters\_to\_ignore": [{"characters\_to\_skip": ".,;:-'\"()"}],

}

},

},

{

"fields": [{"name": "ip\_address"}],

"primitive\_transformation": {

"replace\_with\_info\_type\_config": {"info\_type": {"name": "IP\_ADDRESS"}},

},

},

]

},

"info\_type\_transformations": {"transformations": [{"info\_types": info\_types, "action": {"redact": {}}}]},

}

# Define the re-identification transformation

reidentify\_config = {

"record\_transformations": {

"field\_transformations": [

{

"fields": [{"name": "first\_name"}, {"name": "last\_name"}, {"name": "email"}],

"primitive\_transformation": {"replace\_with\_info\_type\_config": {"info\_type": {"name": "PERSON\_NAME"}}},

},

{"fields": [{"name": "ip\_address"}], "primitive\_transformation": {"replace\_with\_info\_type\_config": {"info\_type": {"name": "IP\_ADDRESS"}}}},

]

},

"info\_type\_transformations": {"transformations": [{"info\_types": info\_types, "action": {"replaceWithInfoTypeConfig": {"infoType": {"name": "PII"}}}}]},

}

# Define the dataset

dataset = [

{"id": 1, "first\_name": "John", "last\_name": "Doe", "email": "john.doe@example.com", "gender": "male", "ip\_address": "192.168.0.1"},

{"id": 2, "first\_name": "Jane", "last\_name": "Doe", "email": "jane.doe@example.com", "gender": "female", "ip\_address": "192.168.0.2"},

]

# De-identify the dataset

deidentify\_request = dlp\_v2.DeidentifyContentRequest(

parent=f"projects/{PROJECT\_ID}/locations/{LOCATION}",

deidentify\_config=deidentify\_config,

item={"table": {"headers": [{"name": "id"}, {"name": "first\_name"}, {"name": "last\_name"}, {"name": "email"}, {"name": "gender"}, {"name": "ip\_address"}], "rows": [{"values": [str(d[key]) for key in ["id", "first\_name", "last\_name", "email", "gender", "ip\_address"]]} for d in dataset]}},

)

deidentified\_response = dlp.deidentify\_content(deidentify\_request)

deidentified\_dataset = [{header["name"]: row["values"][i] for i, header in enumerate(deidentified\_response.item.table.headers)} for row in deidentified\_response.item.table.rows]

# Re-identify the dataset

reidentify\_request = dlp\_v2.ReidentifyContentRequest(

parent=f"projects/{PROJECT\_ID}/locations/{LOCATION}",

reidentify\_config=reidentify\_config,

inspect\_config={"info\_types": info\_types},

item={"table": {"headers": [{"name": "id"}, {"name": "first\_name"}, {"name": "last\_name"}, {"name": "email"}, {"name": "gender"}, {"name": "ip\_address"}], "rows": [{"values": [str(d[key]) for key in ["id", "first\_name", "last\_name", "email", "gender", "ip\_address"]]} for d in deidentified\_dataset]}},

)

reidentified\_response = dlp.reidentify\_content(reidentify\_request)

reidentified\_dataset = [{header["name"]: row["values"][i] for i, header in enumerate(reidentified\_response.item.table.headers)} for row in reidentified\_response.item.table.rows]

#Print the original and re-identified datasets to test the code

print("Original dataset:")

print(dataset)

print("Re-identified dataset:")

print(reidentified\_dataset)

css

# **Step 7:** ML model - Built on Vertex AI (Google Cloud)

Perform K-Means clustering and GMM clustering as the data looks like demographics data of people and contains network information (IP address). My thought process is that we can find patterns and outliers from the people using clustering algorithms. These are unsupervised type of machine learning.

## K-Means clustering

from google.cloud import aiplatform

from google.protobuf import json\_format

from google.protobuf.struct\_pb2 import Value

from google.cloud.aiplatform\_v1.types import (

DeployedModel,

ExplanationMetadata,

ExplanationParameters,

ExplanationSpec,

InputDataConfig,

Model,

ModelContainerSpec,

Port,

PredictRequest,

PredictionInput,

PredictionOutput,

)

import pandas as pd

from sklearn.preprocessing import StandardScaler

import numpy as np

# Set the project and location

PROJECT\_ID = "hale-brook-377621"

LOCATION = "us-central1"

# Set the name and location of the dataset file

DATASET\_PATH = "gs://your-bucket/dataset.csv"

# Create a Vertex AI client

client\_options = {"api\_endpoint": f"{LOCATION}-aiplatform.googleapis.com"}

client = aiplatform.gapic.JobServiceClient(client\_options=client\_options)

# Load the dataset

df = pd.read\_csv(DATASET\_PATH)

# Select features for clustering

X = df[["first\_name", "last\_name", "email", "gender", "ip\_address"]]

# Preprocess the features using standard scaler

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Set the number of clusters

num\_clusters = 3

# Set the container image URI

image\_uri = "us-docker.pkg.dev/vertex-ai/prediction/kmeans-cpu.1-0:latest"

# Create the model container specification

model\_container\_spec = ModelContainerSpec(image\_uri=image\_uri)

# Create the K-Means model

model = Model(

display\_name="kmeans-model",

container\_spec=model\_container\_spec,

metadata\_schema\_uri="gs://google-cloud-aiplatform/schema/prediction/classification\_1.0.0.yaml",

)

# Create the input data configuration

input\_data\_config = InputDataConfig(

instances\_format="json",

json\_instance\_schema\_uri="gs://google-cloud-aiplatform/schema/prediction/input/text\_classification\_example.json",

)

# Create the explanation spec

explanation\_metadata = ExplanationMetadata(

inputs={"features": {"input\_tensor\_name": "input"}}

)

explanation\_parameters = ExplanationParameters(

{"sampled\_shapley\_attribution": {"path\_count": 20}}

)

explanation\_spec = ExplanationSpec(

metadata=explanation\_metadata, parameters=explanation\_parameters

)

# Create the model deployment

deployed\_model = DeployedModel(model=model)

endpoint = aiplatform.Endpoint.create(

display\_name="kmeans-endpoint", explanation\_spec=explanation\_spec

)

endpoint.deploy(

deployed\_model=deployed\_model,

traffic\_percentage=100,

machine\_type="n1-standard-2",

)

# Create a prediction request for a sample dataset

input\_data = [{"first\_name": row[0], "last\_name": row[1], "email": row[2], "gender": row[3], "ip\_address": row[4]} for row in X.values]

predict\_request = PredictRequest(

endpoint=endpoint.resource\_name,

instances=[json\_format.Parse(json\_format.MessageToJson(PredictionInput(data=InputDataConfig.EncodedData(json\_format.Parse(json\_format.MessageToJson(Value(json\_value=bytes(json.dumps(instance), "utf-8"))))))), PredictRequest) for instance in input\_data],

)

# Make a prediction request

response = client.predict(predict\_request)

# Get the cluster assignments

cluster\_assignments = np.array([np.argmax(prediction.outputs["scores"].value) for prediction in response.predictions])

# #add a new column in the dataset

df["cluster"] = cluster\_assignments

# #Print the dataset with the predicted clusters

print(df.head())

## GMM

from google.cloud import aiplatform

from google.cloud.aiplatform.gapic.schema import predict as predict\_pb2

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.mixture import GaussianMixture

# Load the dataset

df = pd.read\_csv("dataset.csv")

# Select the features for clustering

X = df[["first\_name", "last\_name", "email", "gender", "ip\_address"]]

# Preprocess the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Set the number of clusters

num\_clusters = 3

# Set the container image URI for the GMM model

container\_image\_uri = "us-docker.pkg.dev/vertex-ai/prediction/gaussian-mixture-model:latest"

# Create the model container specification

model\_container\_spec = {

"image\_uri": container\_image\_uri,

}

# Create the GMM model

model = aiplatform.Model(

display\_name="gmm-clustering-model",

container\_spec=model\_container\_spec,

predict\_schemata=predict\_pb2.ValueSpec(

encoded\_value=predict\_pb2.EncodedValueSpec(

tensor\_shape=predict\_pb2.TensorShape(dim=[predict\_pb2.FixedDim(size=5)])

)

),

)

# Create the input data configuration

input\_data\_config = {

"instances": [

{

"first\_name": x[0],

"last\_name": x[1],

"email": x[2],

"gender": x[3],

"ip\_address": x[4],

}

for x in X\_scaled

]

}

# Set the number of clusters in the explanation metadata

explanation\_metadata = {

"inputs": {

"first\_name": {"type": "CATEGORY"},

"last\_name": {"type": "CATEGORY"},

"email": {"type": "CATEGORY"},

"gender": {"type": "CATEGORY"},

"ip\_address": {"type": "CATEGORY"},

},

"outputs": {"cluster": {"type": "CATEGORY", "number\_of\_categories": num\_clusters}},

}

# Set the explanation spec

explanation\_spec = {"metadata": explanation\_metadata}

# Train the GMM model and deploy it to an endpoint

endpoint = model.deploy(

machine\_type="n1-standard-4",

min\_replica\_count=1,

max\_replica\_count=1,

)

# Create a prediction request for a sample dataset

sample\_dataset = [

{"first\_name": "Alice", "last\_name": "Smith", "email": "alice@example.com", "gender": "Female", "ip\_address": "123.45.67.89"},

{"first\_name": "Bob", "last\_name": "Jones", "email": "bob@example.com", "gender": "Male", "ip\_address": "98.76.54.32"},

{"first\_name": "Charlie", "last\_name": "Lee", "email": "charlie@example.com", "gender": "Male", "ip\_address": "12.34.56.78"},

]

sample\_dataset\_scaled = scaler.transform(pd.DataFrame(sample\_dataset)[["first\_name", "last\_name", "email", "gender", "ip\_address"]])

prediction\_request = predict\_pb2.PredictRequest(

instances=[{"values": [list(row)]} for row in sample\_dataset\_scaled],

parameters={"explanation\_spec": explanation\_spec},

)

# Make the prediction request using the Vertex AI Python SDK

response = endpoint.predict(prediction\_request)

# Get the cluster assignments from

# Extract the predicted clusters

predicted\_clusters = np.array(response.predictions[0].tables.values).flatten()

# Print the predicted clusters for each sample dataset

for i, cluster in enumerate(predicted\_clusters):

print(f"Sample dataset {i+1} is assigned to cluster {cluster}")

# **Step 8:** Deploy the ML model as API endpoint - I used Vertex AI Platform prediction Service

from googleapiclient import discovery

from google.oauth2 import service\_account

import numpy as np

# Specify the project ID and the name of the model to be deployed

PROJECT\_ID = 'hale-brook-377621'

MODEL\_NAME = 'your-model-name'

# Load the service account credentials

credentials = service\_account.Credentials.from\_service\_account\_file('path/to/your/credentials.json')

# Create a client object for the AI Platform Prediction service

ml = discovery.build('ml', 'v1', credentials=credentials)

# Define a function to preprocess the input data before making predictions

def preprocess(input\_data):

# Apply any necessary preprocessing steps here, such as scaling or one-hot encoding

return input\_data

# Define a function to make predictions using the deployed model

def predict(input\_data):

# Preprocess the input data

preprocessed\_data = preprocess(input\_data)

# Create a request body for the prediction request

request\_body = {"instances": preprocessed\_data.tolist()}

# Make the prediction request using the AI Platform Prediction API

parent = f'projects/{PROJECT\_ID}/models/{MODEL\_NAME}'

response = ml.projects().predict(name=parent, body=request\_body).execute()

# Extract the predicted labels from the response

predicted\_labels = np.array(response['predictions']).flatten()

# Return the predicted labels

return predicted\_labels

gcloud ai-platform models create your-model-name --regions=us-central1

gcloud ai-platform versions create your-version-name --model=your-model-name --runtime-version=2.5 --python-version=3.7 --framework=scikit-learn --origin=gs://your-bucket/path/to/model --project=hale-brook-377621