ASSIGNMENT 4: PART B

TASK A: Build an event-driven serverless application using GCP ML.

LINK TO GITLAB: https://git.cs.dal.ca/rtyagi/csci5410_b00872269_ruhirajnish_tyagi/-/tree/main/assignment-4

*Step 1: Create **sourcedatab00872269** bucket, programmatically upload files from train folder and create and show working of **generateVector** cloud function.*

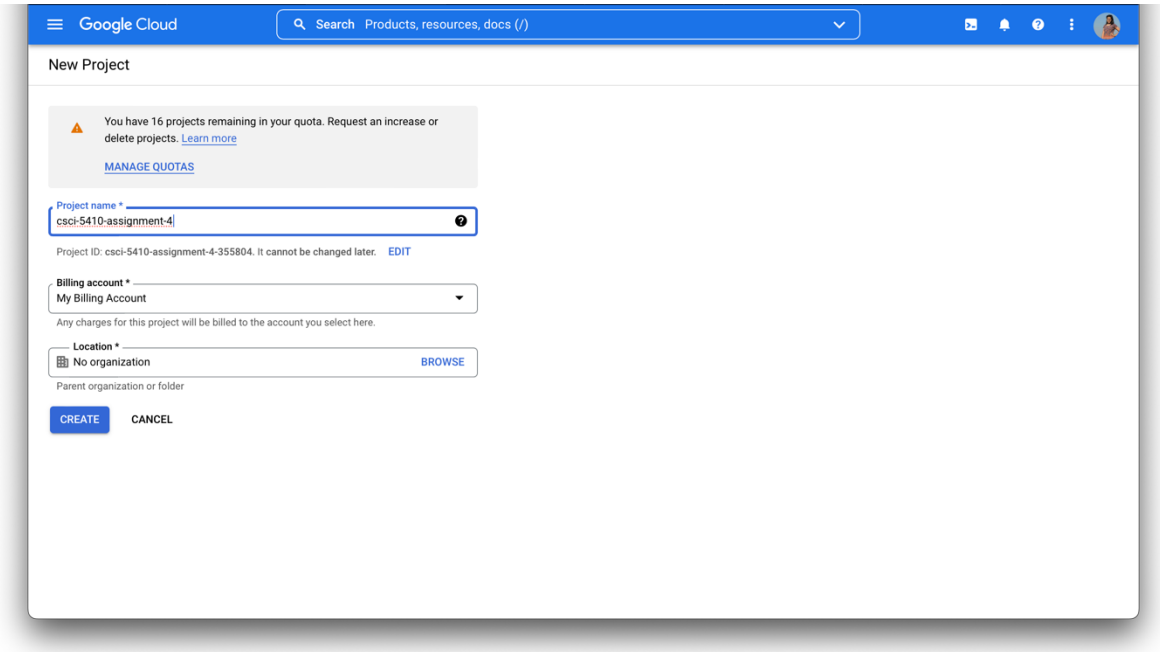


***Figure 1.** Screenshot of creating project in Google Cloud Console.*

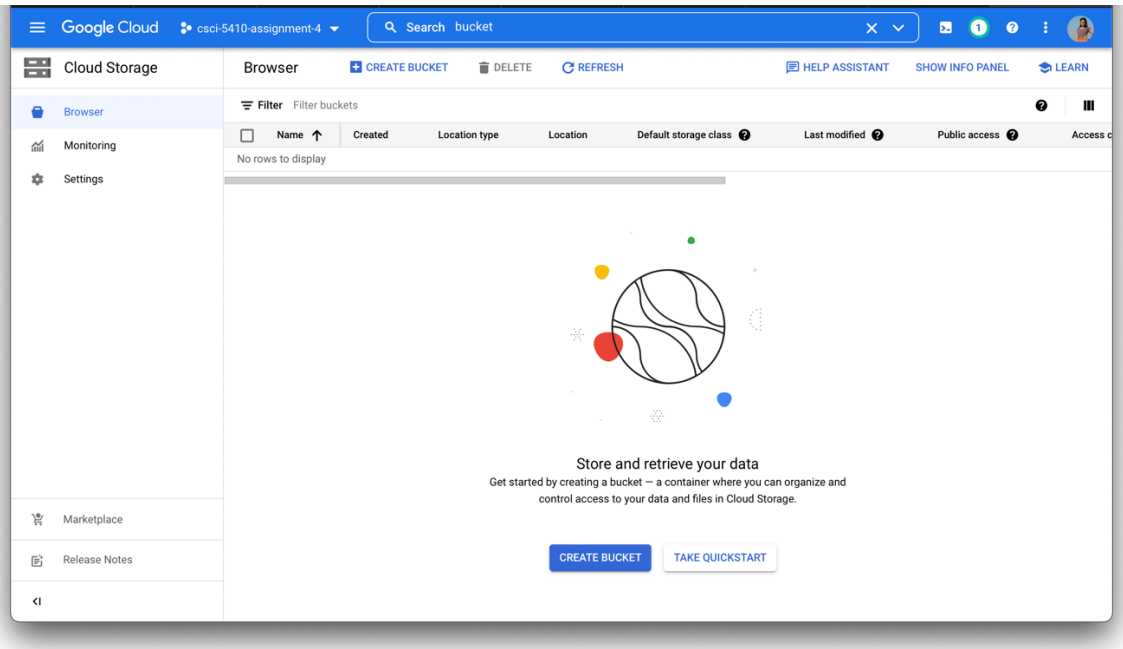


Figure 2. *Screenshot of empty Cloud Storage[1] in Google Cloud Console.*

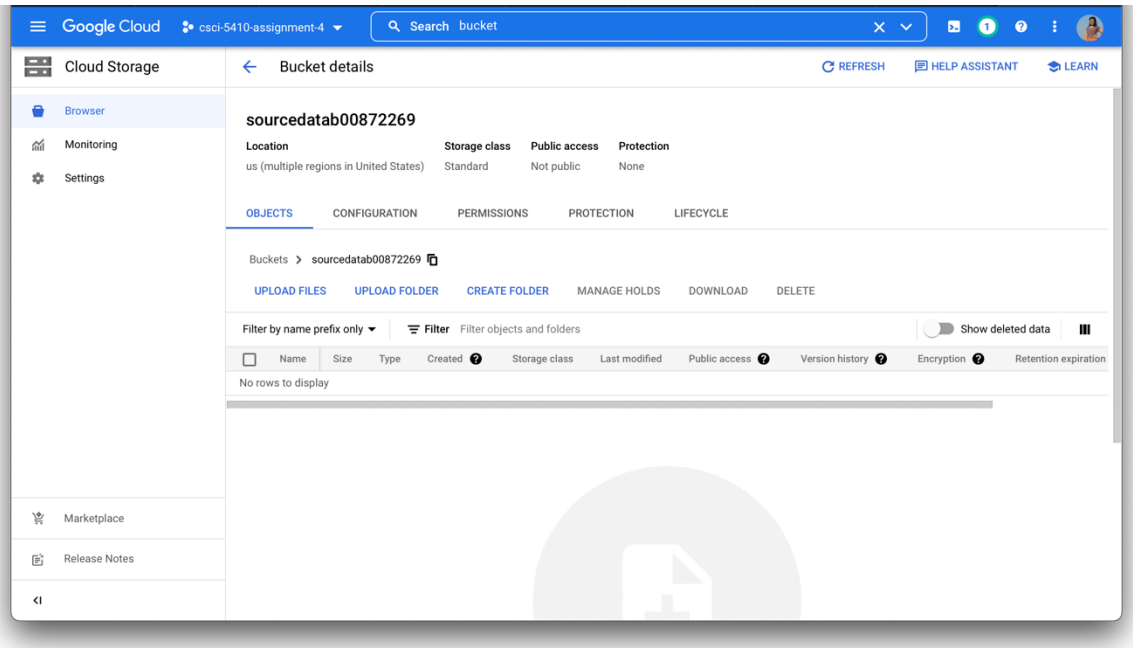


Figure 3. *Screenshot of empty sourcedatab00872269 bucket .*

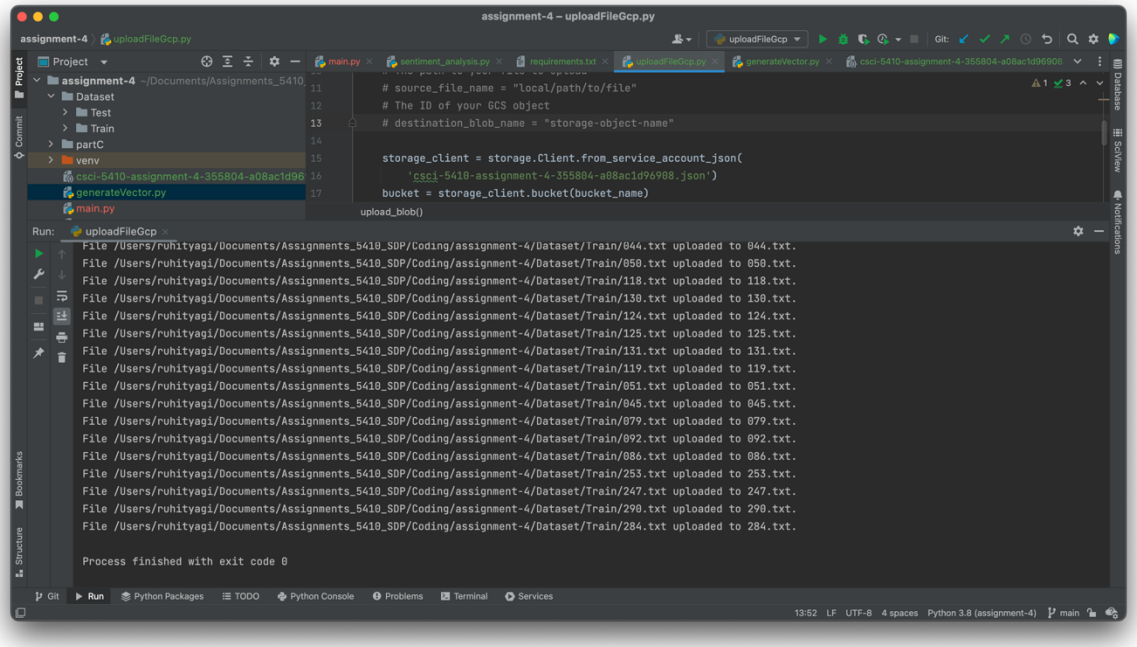


Figure 4. *Screenshot of programmatically[2] uploading files 001-299 to sourcedatab00872269 bucket.*

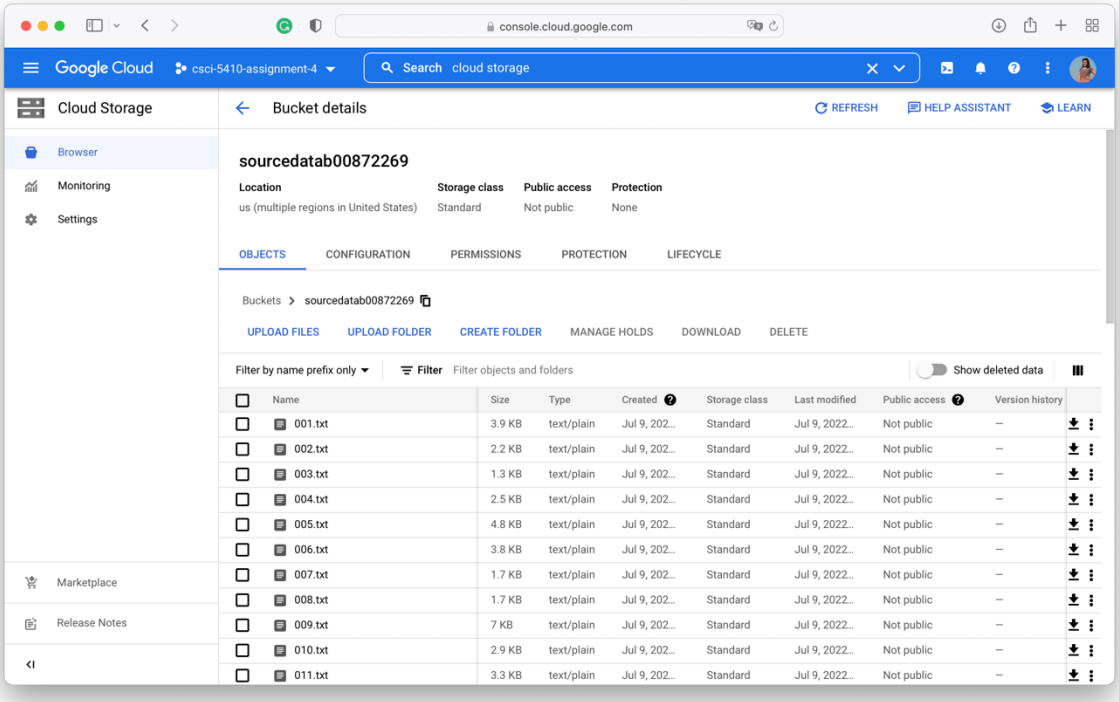


Figure 5(a). *Screenshot of SUCCESSFUL upload of files 001-299 to sourcedatab00872269 bucket.*

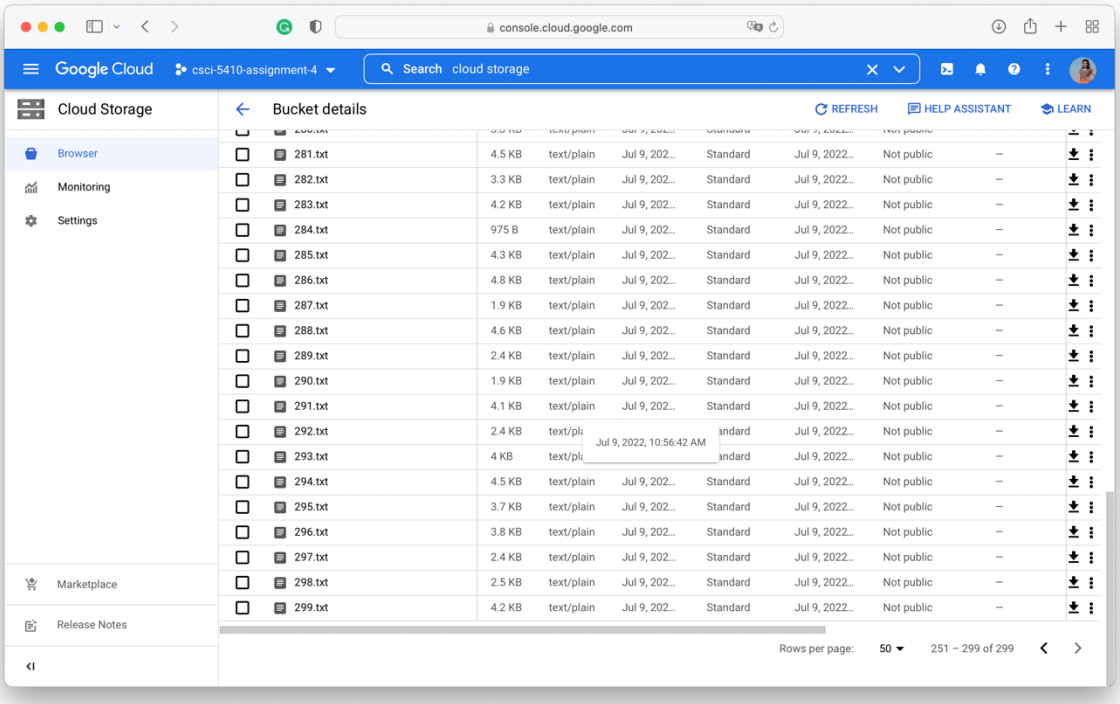


Figure 5(b). *Screenshot of SUCCESSFUL upload of files 001-299 to sourcedatab00872269 bucket.*

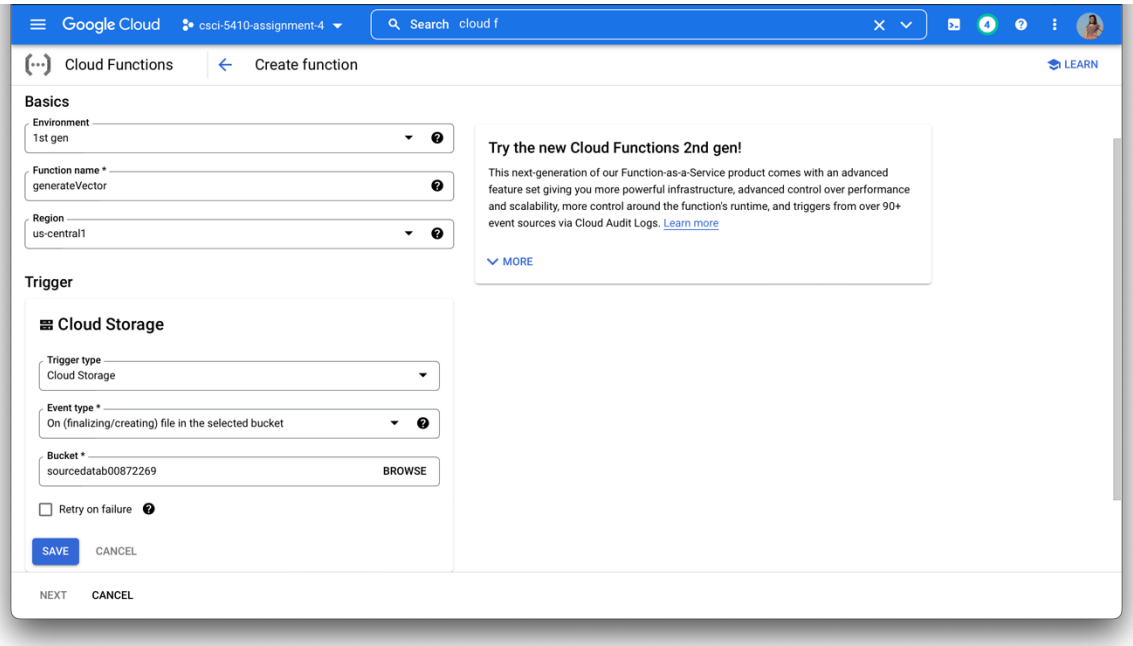


Figure 6. *Screenshot of creating generateVector cloud function[3].*

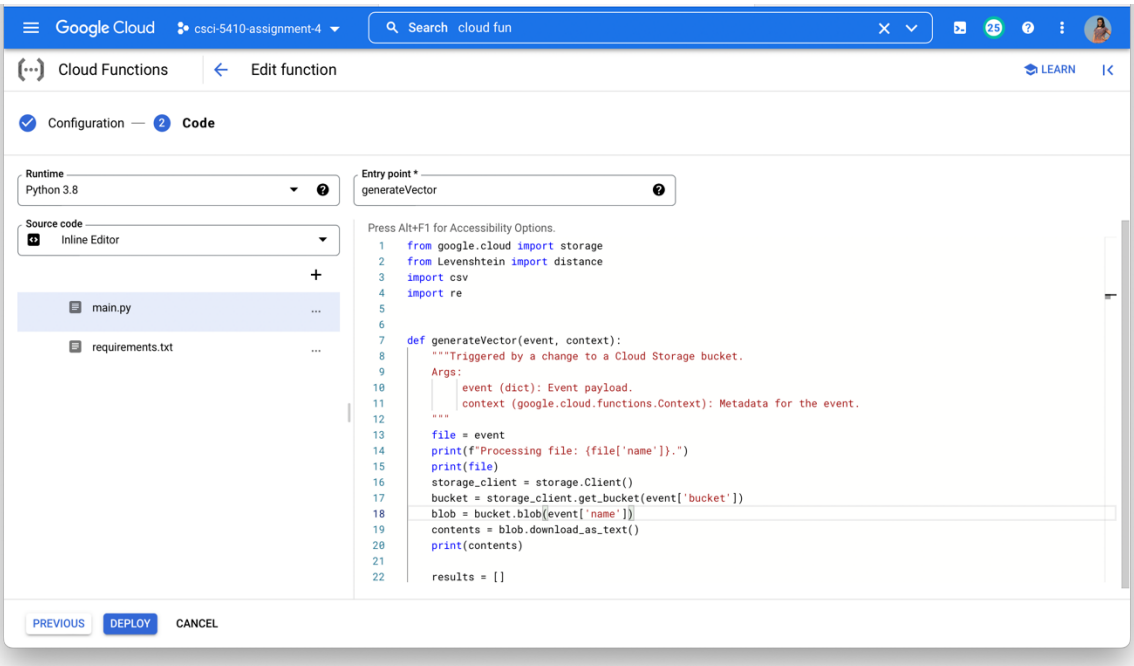


Figure 7. *Screenshot of created generateVector cloud function.*

*Step 2: Create **traindatab00872269** bucket and show working of **generateVector** cloud function to save **trainVector.csv** to the bucket.*

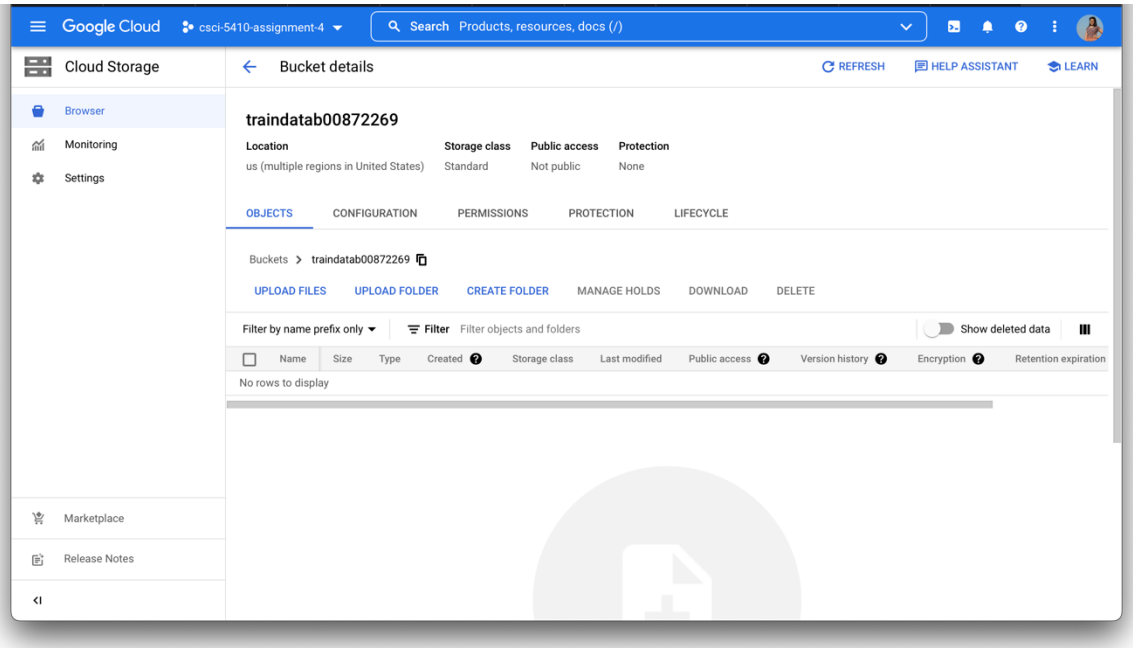


***Figure 8.** Screenshot of empty traindatab00872269 BEFORE running generateVector cloud function.*

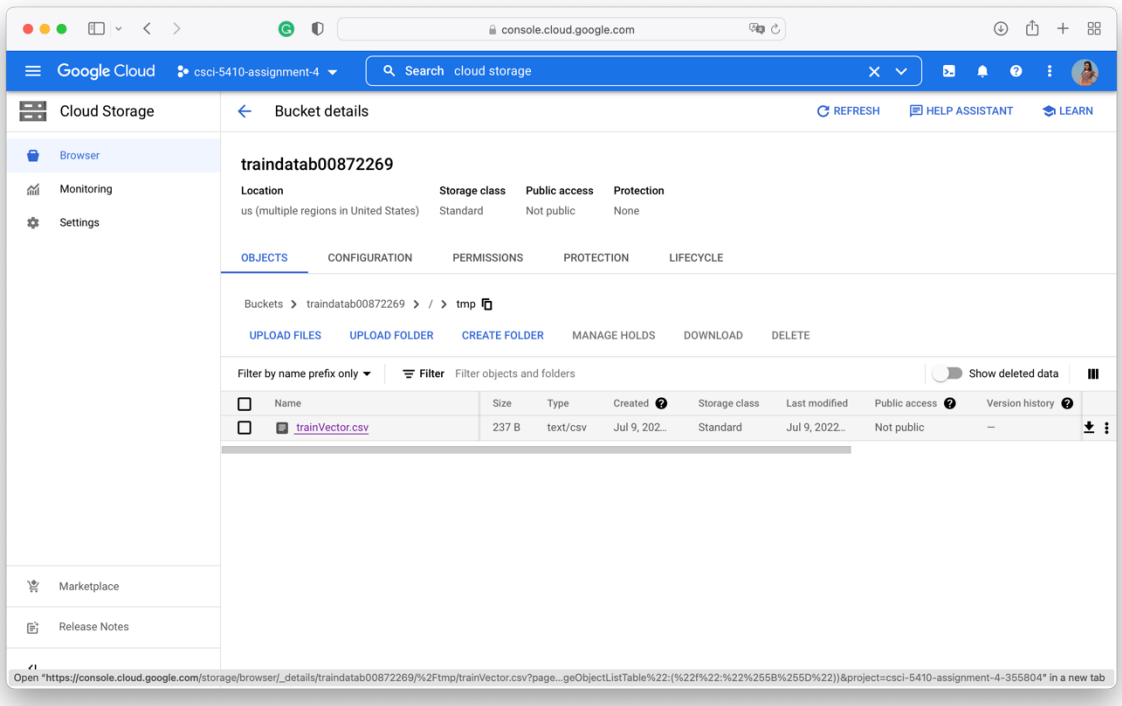


***Figure 9.** Screenshot of traindatab00872269 AFTER running generateVector cloud function.*

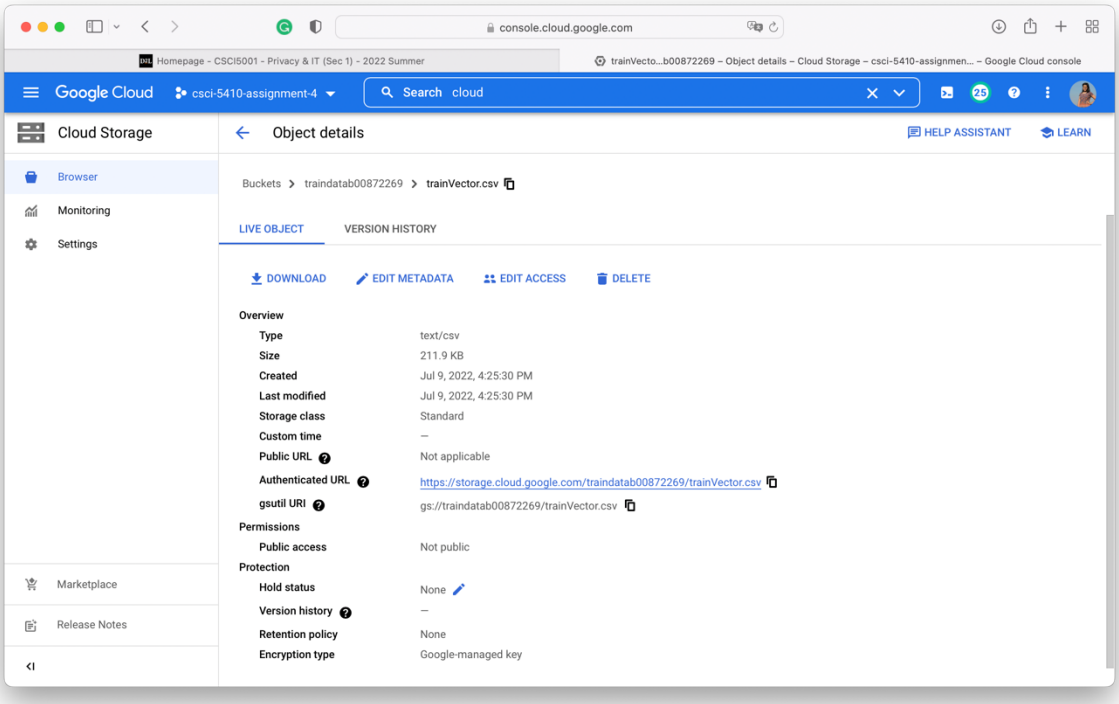
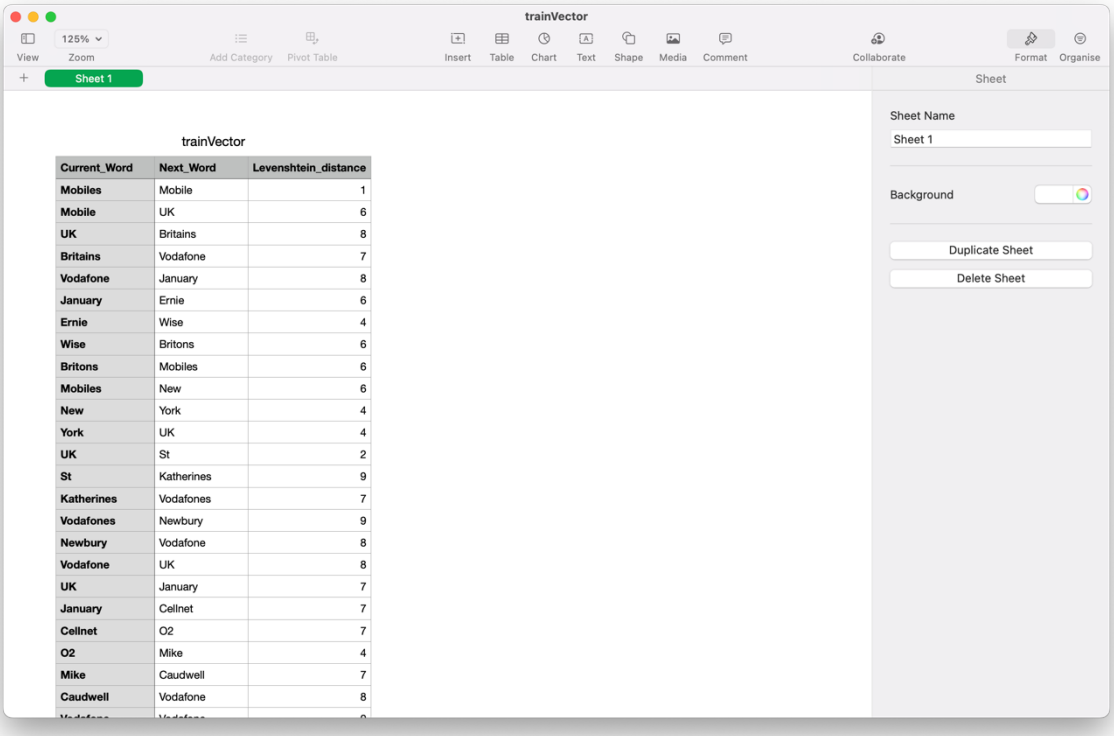


Figure 10. *Screenshot of details of trainVector.csv in cloud storage.*



trainVector

Current_Word	Next_Word	Levenshtein_distance
Mobiles	Mobile	1
Mobile	UK	6
UK	Britains	8
Britains	Vodafone	7
Vodafone	January	8
January	Ernie	6
Ernie	Wise	4
Wise	Britons	6
Britons	Mobiles	6
Mobiles	New	6
New	York	4
York	UK	4
UK	St	2
St	Katherines	9
Katherines	Vodafones	7
Vodafones	Newbury	9
Newbury	Vodafone	8
Vodafone	UK	8
UK	January	7
January	Cellnet	7
Cellnet	O2	7
O2	Mike	4
Mike	Caudwell	7
Caudwell	Vodafone	8

Figure 11. *Screenshot of trainVector.csv contents.*

*Step 3: Create **testdatab00872269** bucket and show working of **generateVector** cloud function to save **testVector.csv** to the bucket.*

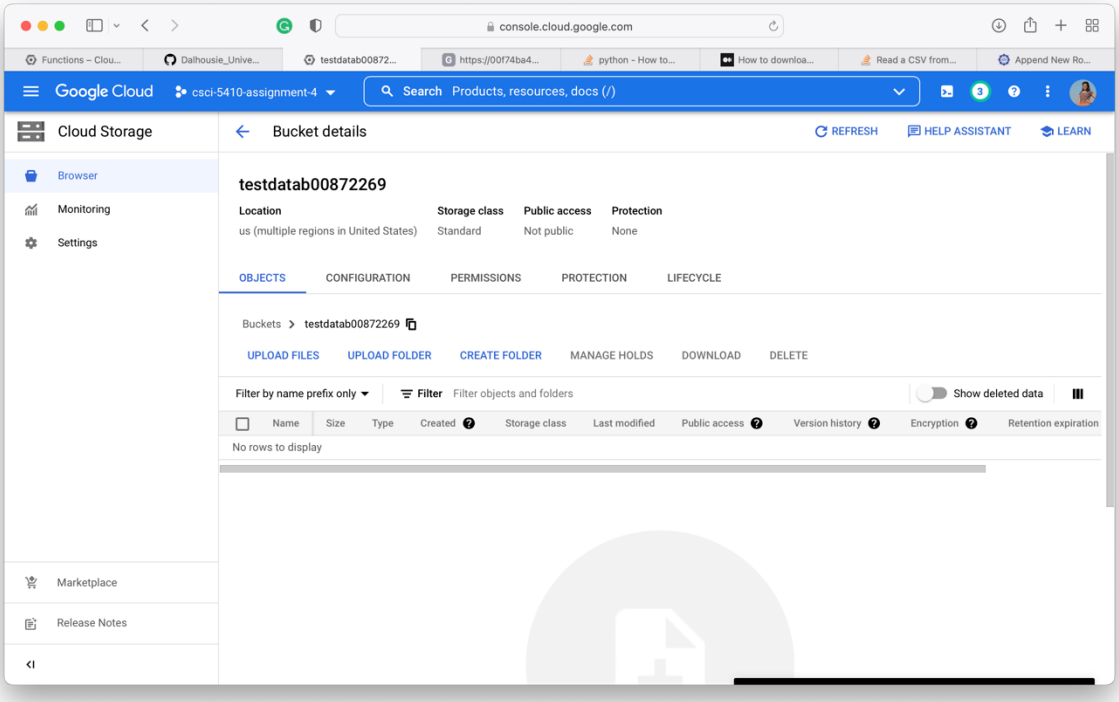


***Figure 12.** Screenshot of empty testdatab00872269 BEFORE running generateVector cloud function.*

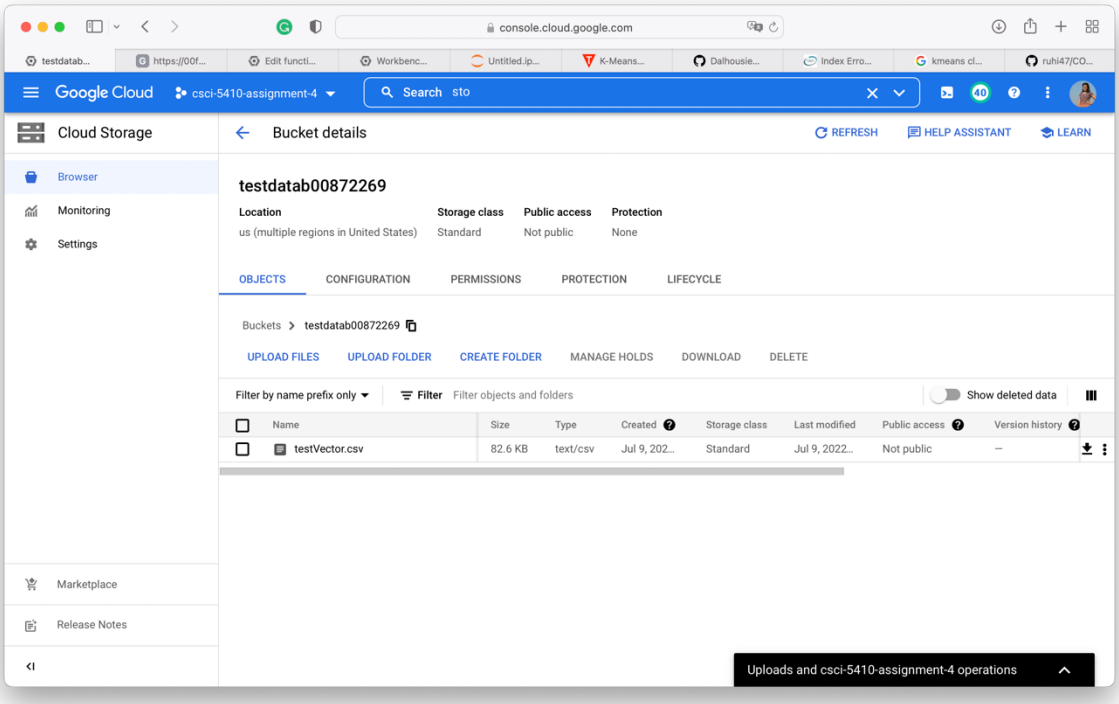


***Figure 13.** Screenshot of testdatab00872269 AFTER running generateVector cloud function.*

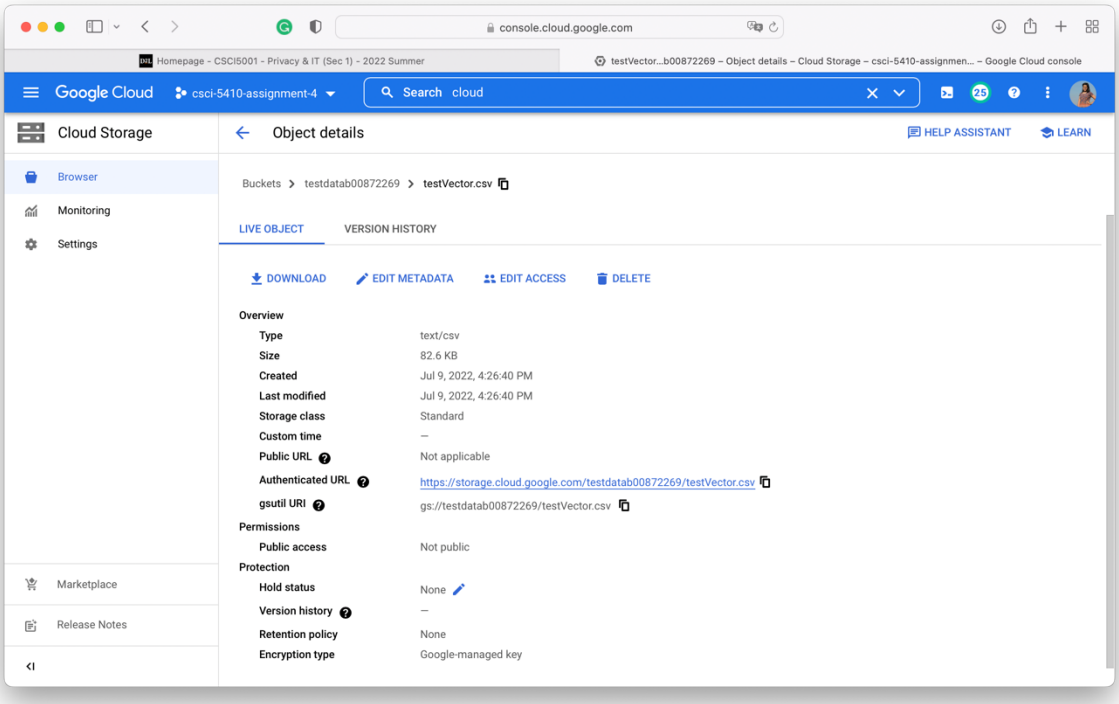
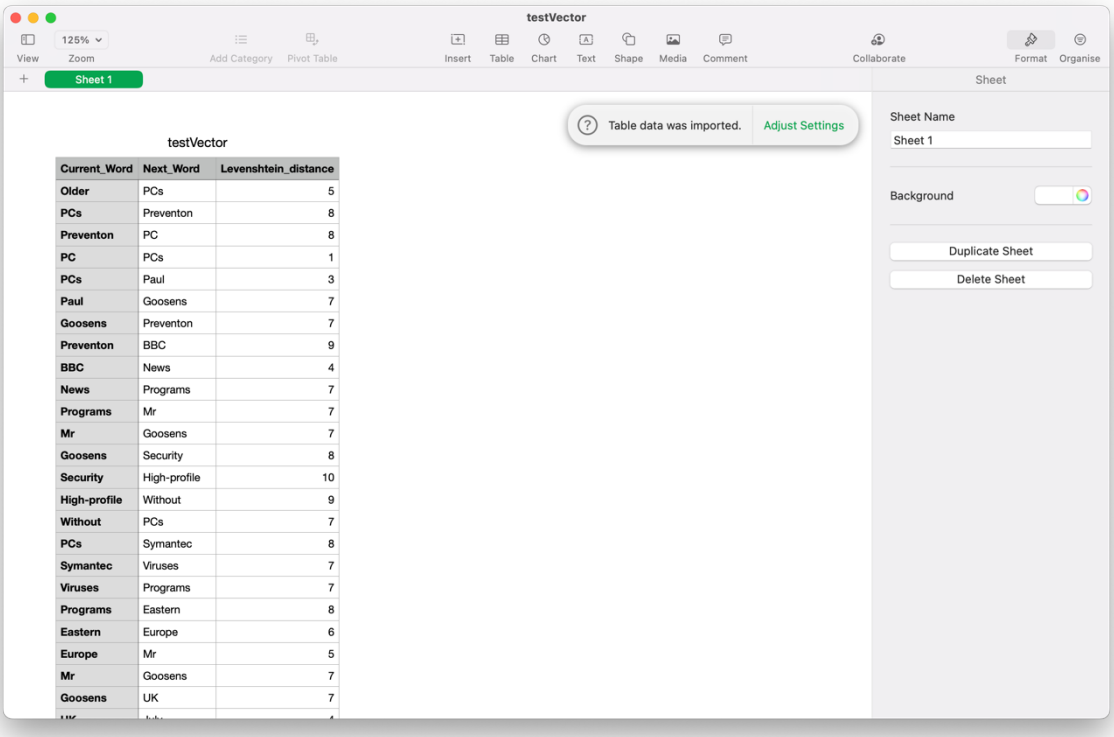


Figure 14. *Screenshot of details of testVector.csv in cloud storage.*



testVector

Current_Word	Next_Word	Levenshtein_distance
Older	PCs	5
PCs	Preventon	8
Preventon	PC	8
PC	PCs	1
PCs	Paul	3
Paul	Goosens	7
Goosens	Preventon	7
Preventon	BBC	9
BBC	News	4
News	Programs	7
Programs	Mr	7
Mr	Goosens	7
Goosens	Security	8
Security	High-profile	10
High-profile	Without	9
Without	PCs	7
PCs	Symantec	8
Symantec	Viruses	7
Viruses	Programs	7
Programs	Eastern	8
Eastern	Europe	6
Europe	Mr	5
Mr	Goosens	7
Goosens	UK	7

Figure 15. *Screenshot of testVector.csv contents.*

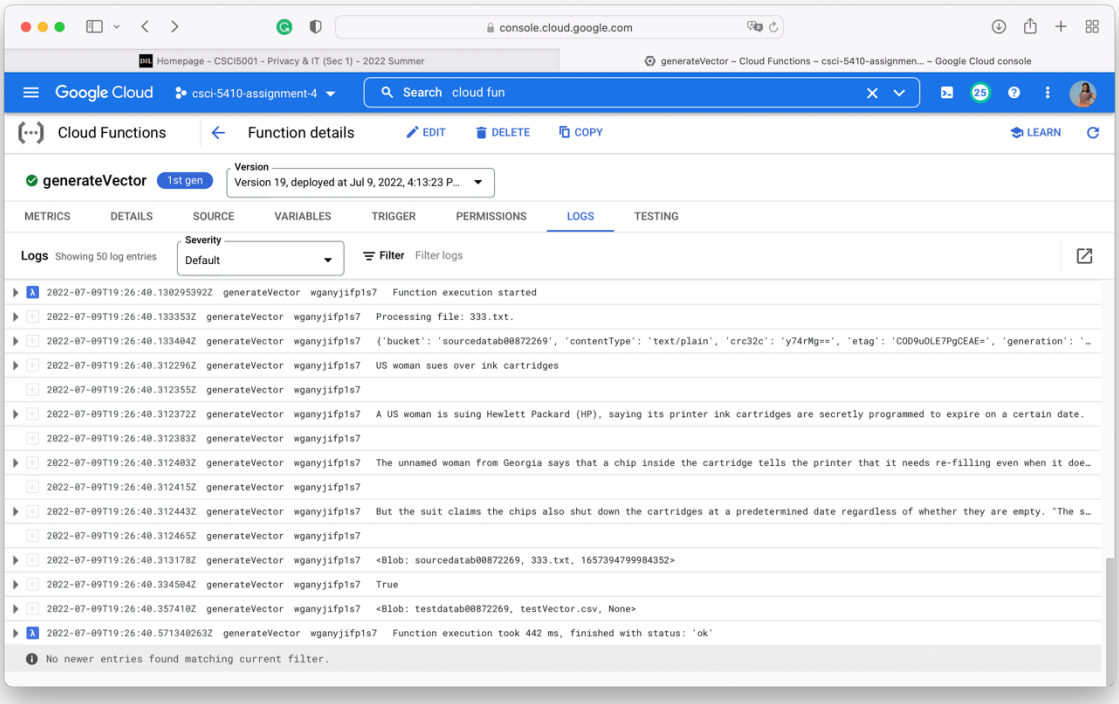


Figure 16. *Screenshot of logs of generateVector cloud function.*

*Step 4: Create **ml-test** GCP ML function in Vertex AI, train it using trainVector.csv, test it with testVector.csv and show the clusters of both data using **K-means Clustering algorithm[4]**.*

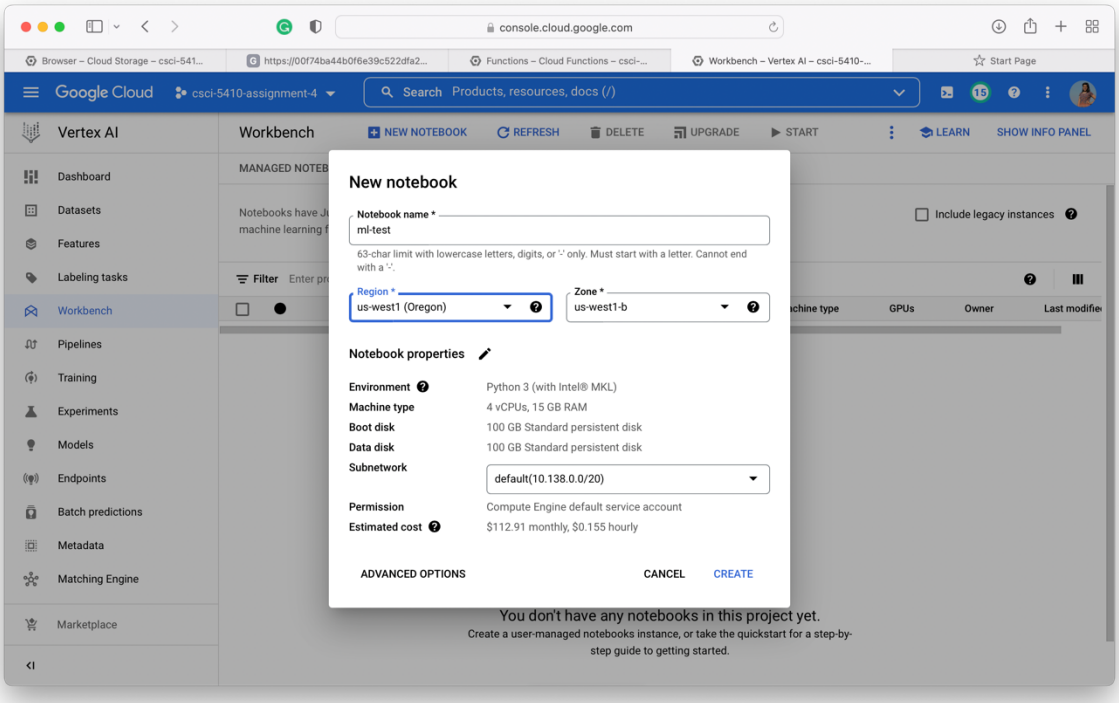


***Figure 17.** Screenshot of configuring jupyter notebook to perform GCP ML task in Vertex AI [5].*

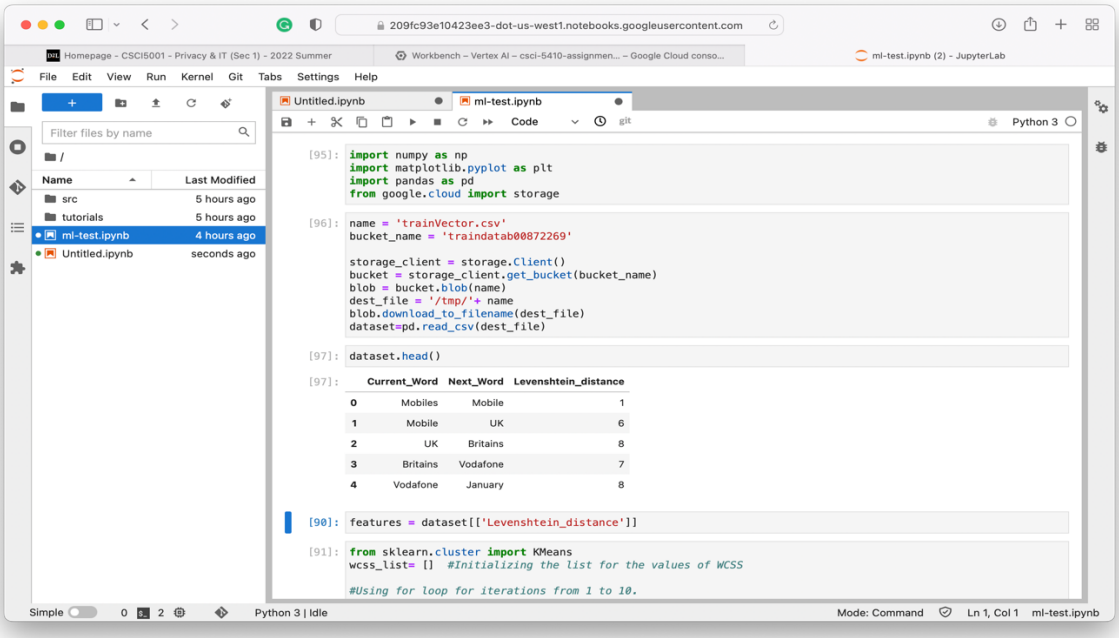


***Figure 18.** Screenshot of trainVector.csv data in notebook using pandas library.*

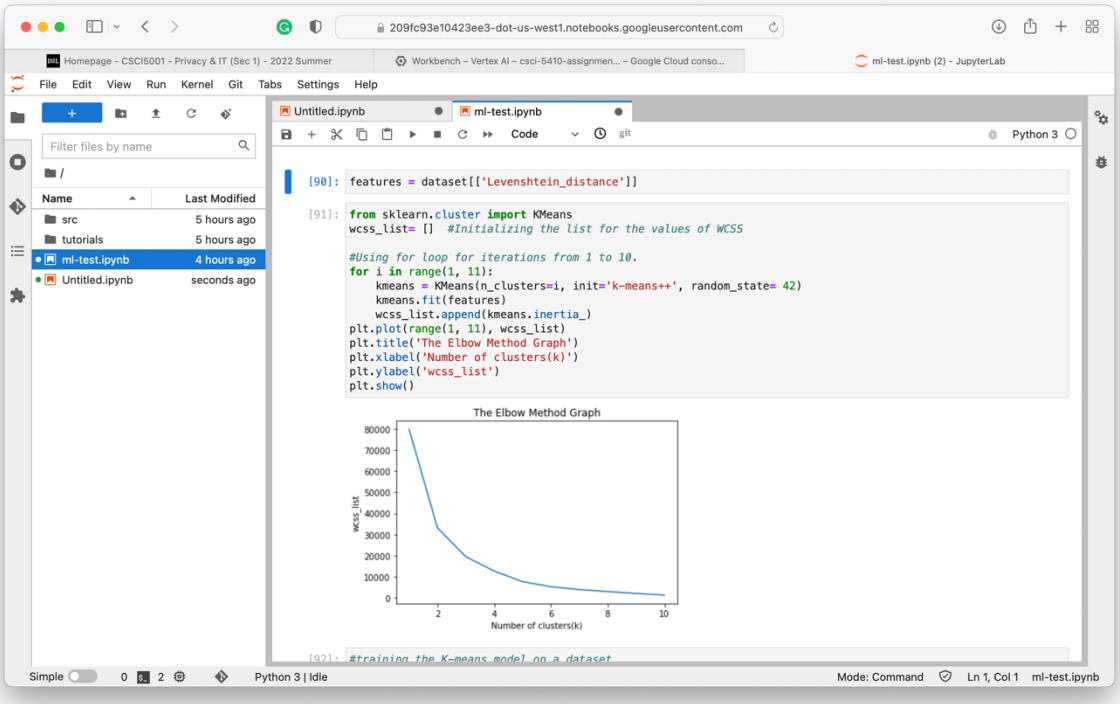


Figure 19(a). *Screenshot of elbow method graph bending at K=5.*

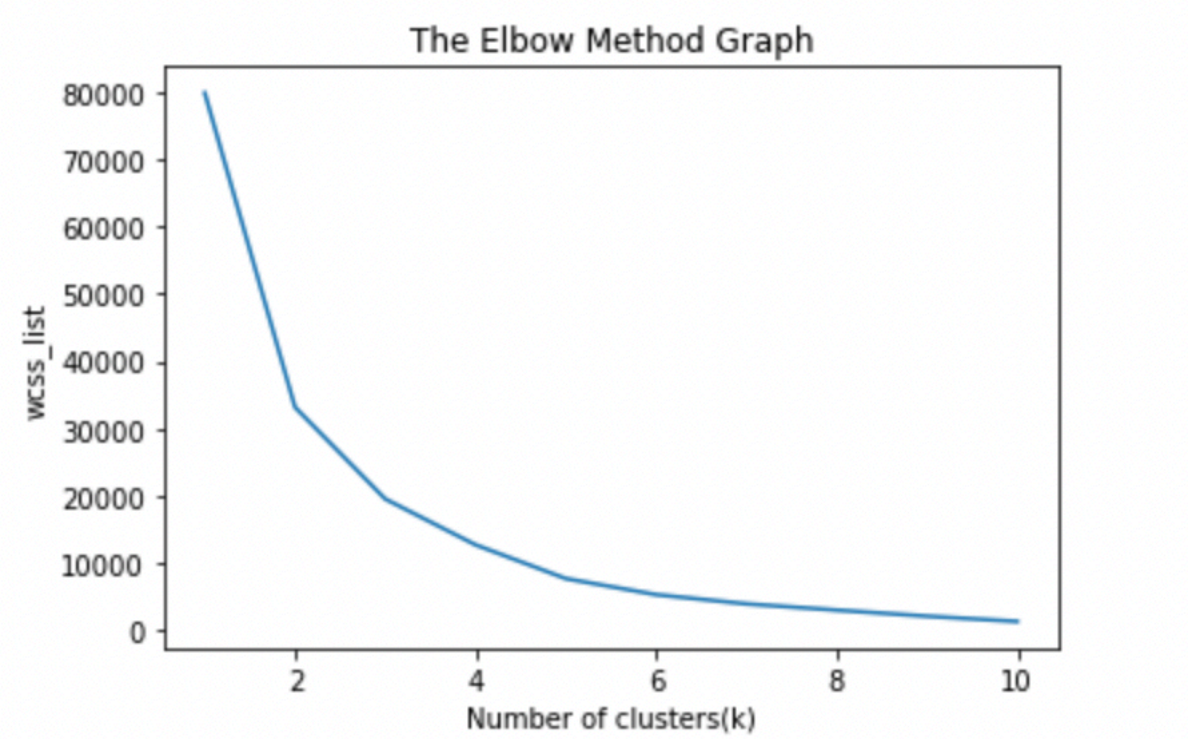


Figure 19(b). *Screenshot of elbow method graph bending at K=5.*

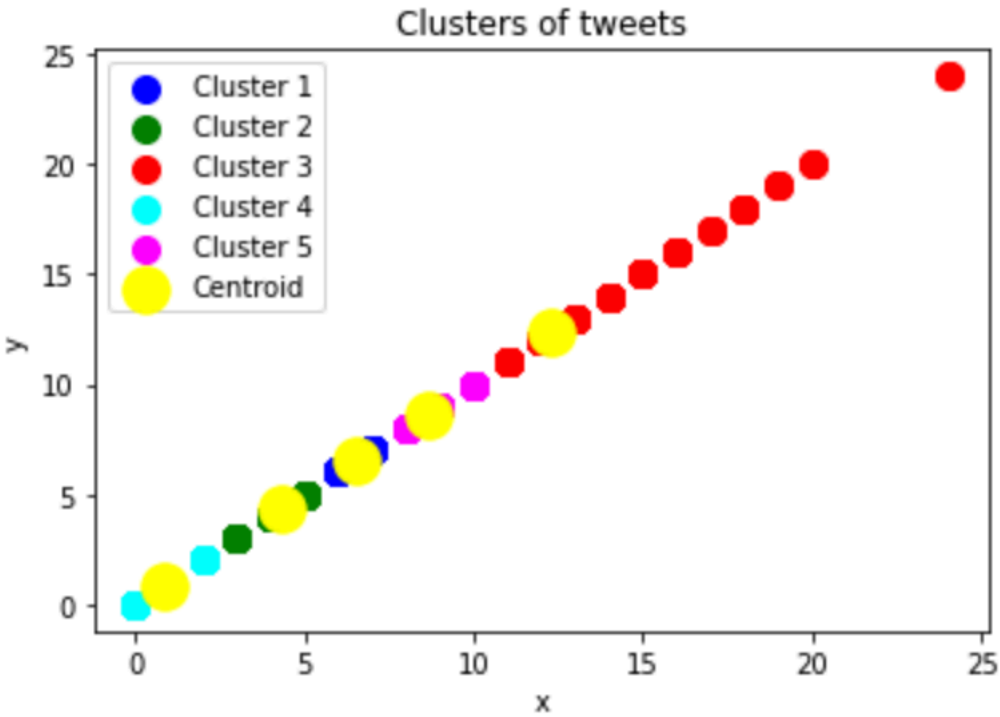
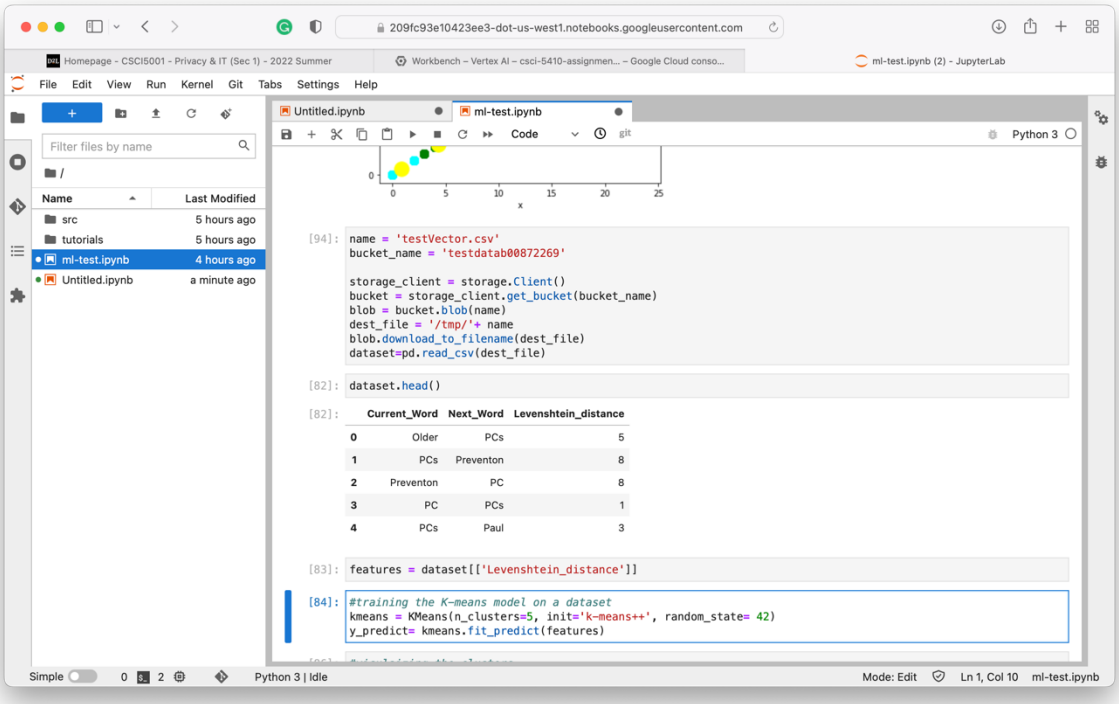


Figure 20. *Screenshot of clusters with centroid and outliers from trainVector.csv data.*



	Current_Word	Next_Word	Levenshtein_distance
0	Older	PCs	5
1	PCs	Preventon	8
2	Preventon	PC	8
3	PC	PCs	1
4	PCs	Paul	3

Figure 21. *Screenshot of testVector.csv data in notebook using pandas library.*

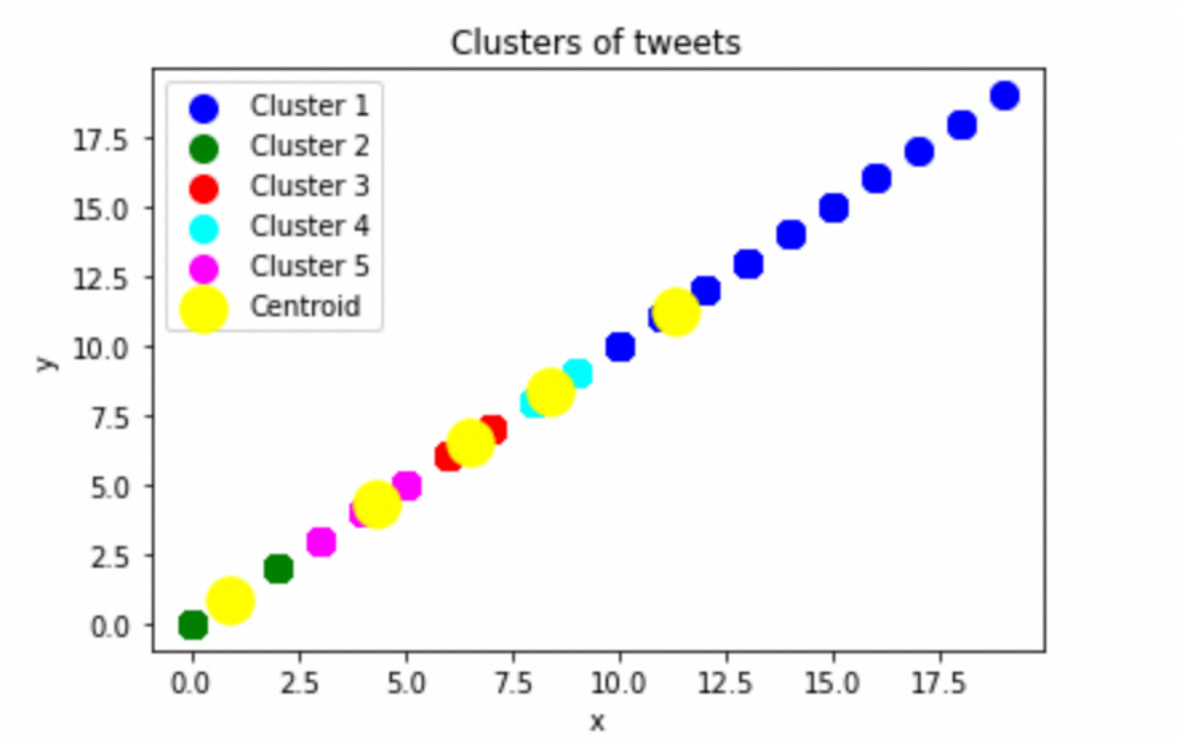


Figure 22. *Screenshot of clusters with centroid and outliers from testVector.csv data.*

Step 5: *Code samples* ***of generateVector.py, uploadFileGcp.py*** *and* ***ml-test.ipynb.***

1. **generateVector.py**

```
from google.cloud import storage
from Levenshtein import distance
import csv
import re


def generateVector(event, context):
    """Triggered by a change to a Cloud Storage bucket.
    Args:
         event (dict): Event payload.
         context (google.cloud.functions.Context): Metadata for the event.
    """
    file = event
    print(f"Processing file: {file['name']}.")
    print(file)
    storage_client = storage.Client()
    bucket = storage_client.get_bucket(event['bucket'])
    blob = bucket.blob(event['name'])
    contents = blob.download_as_text()
    print(contents)

    results = []
    stopwords = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours',
'ourselves', 'you', "you're", "you've", "you'll",
                 "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', "she's",
                 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
'they', 'them', 'their', 'theirs',
                 'themselves', 'what', 'which', 'who', 'whom', 'this',
'that', "that'll", 'these', 'those', 'am', 'is',
                 'are', 'was', 'were', 'be', 'been', 'being', 'have',
'has', 'had', 'having', 'do', 'does', 'did',
                 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
'because', 'as', 'until', 'while', 'of', 'at',
                 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after',
                 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again',
                 'further', 'then', 'once', 'here', 'there', 'when',
'where', 'why', 'how', 'all', 'any', 'both',
                 'each', 'few', 'more', 'most', 'other', 'some', 'such',
'no', 'nor', 'not', 'only', 'own', 'same',
                 'so', 'than', 'too', 'very', 's', 't', 'can', 'will',
'just', 'don', "don't", 'should', "should've",
                 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain',
'aren', "aren't", 'couldn', "couldn't", 'didn',
                 "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn',
"hasn't", 'haven', "haven't", 'isn', "isn't",
                 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn',
"needn't", 'shan', "shan't", 'shouldn',
                 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won',
"won't", 'wouldn', "wouldn't"]
```

```
    for word in contents.split():
        if word.lower() not in stopwords and word[0].isupper():
            w = re.sub("()\".,']", "", word)
            results.append(w)
    print(blob)
    if int(event['name'].split(".")[0]) < 300:
        name = 'trainVector.csv'
        bucket_name = 'traindatab00872269'
    else:
        name = 'testVector.csv'
        bucket_name = 'testdatab00872269'

    bucket = storage_client.bucket(bucket_name)
    stats = storage.Blob(bucket=bucket, name=name).exists(storage_client)
    print(stats)
    if not stats:
        with open('/tmp/' + name, 'w') as csvfile:
            spamwriter = csv.writer(csvfile, delimiter=',',
                                    quotechar='|',
quoting=csv.QUOTE_MINIMAL)
            spamwriter.writerow(['Current_Word', 'Next_Word',
'Levenshtein_distance'])

            for i in range(len(results) - 1):
                spamwriter.writerow([results[i], results[i + 1],
distance(results[i], results[i + 1])])
    else:
        bucket = storage_client.get_bucket(bucket_name)
        blob = bucket.blob(name)
        print(blob)
        dest_file = '/tmp/' + name
        blob.download_to_filename(dest_file)

        with open(dest_file, 'a') as f_object:
            writer_object = csv.writer(f_object, delimiter=',',
                                       quotechar='|',
quoting=csv.QUOTE_MINIMAL)
            for i in range(len(results) - 1):
                writer_object.writerow([results[i], results[i + 1],
distance(results[i], results[i + 1])])

    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(name)
    blob.upload_from_filename('/tmp/' + name)
```

2. **uploadFileGcp.py:**

```
from os import walk

from google.cloud import storage


def upload_blob(bucket_name, source_file_name, destination_blob_name):
    """Uploads a file to the bucket."""
    # The ID of your GCS bucket
    # bucket_name = "your-bucket-name"
    # The path to your file to upload
    # source_file_name = "local/path/to/file"
    # The ID of your GCS object
    # destination_blob_name = "storage-object-name"

    storage_client = storage.Client.from_service_account_json(
        'csci-5410-assignment-4-355804-a08ac1d96908.json')
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)

    blob.upload_from_filename(source_file_name)

    print(
        f"File {source_file_name} uploaded to {destination_blob_name}."
    )


if __name__ == '__main__':
    filenames = next(walk('Dataset/Train'), (None, None, []))[2]
    for files in filenames:
        upload_blob('sourcedatab00872269',

'/Users/ruhityagi/Documents/Assignments_5410_SDP/Coding/assignment-
4/Dataset/Train/'+files,
                    files)

    filenames = next(walk('Dataset/Test'), (None, None, []))[2]
    for files in filenames:
        upload_blob('sourcedatab00872269',

'/Users/ruhityagi/Documents/Assignments_5410_SDP/Coding/assignment-
4/Dataset/Test/'+files,
                    files)
```

3. **ml-test.ipynb:**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from google.cloud import storage
from sklearn.cluster import KMeans

name = 'trainVector.csv'
bucket_name = 'traindatab00872269'

storage_client = storage.Client()
bucket = storage_client.get_bucket(bucket_name)
blob = bucket.blob(name)
dest_file = '/tmp/'+ name
blob.download_to_filename(dest_file)
dataset=pd.read_csv(dest_file)
dataset.head()

features = dataset[['Levenshtein_distance']]
wcss_list = []  # Initializing the list for the values of WCSS

# Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(features)
    wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list)
plt.title('The Elbow Method Graph')
plt.xlabel('Number of clusters(k)')
plt.ylabel('wcss_list')
plt.show()

#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(features)
features=np.array(features)
plt.scatter(features[y_predict == 0], features[y_predict == 0], s = 100, c
= 'blue', label = 'Cluster 1') #for first cluster
plt.scatter(features[y_predict == 1], features[y_predict == 1], s = 100, c
= 'green', label = 'Cluster 2') #for second cluster
plt.scatter(features[y_predict== 2], features[y_predict == 2], s = 100, c =
'red', label = 'Cluster 3') #for third cluster
plt.scatter(features[y_predict == 3], features[y_predict == 3], s = 100, c
= 'cyan', label = 'Cluster 4') #for fourth cluster
plt.scatter(features[y_predict == 4], features[y_predict == 4], s = 100, c
= 'magenta', label = 'Cluster 5') #for fifth cluster
plt.scatter(kmeans.cluster_centers_[:], kmeans.cluster_centers_[:], s =
300, c = 'yellow', label = 'Centroid')
plt.title('Clusters of tweets')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()

name = 'testVector.csv'
bucket_name = 'testdatab00872269'

storage_client = storage.Client()
bucket = storage_client.get_bucket(bucket_name)
blob = bucket.blob(name)
```

```
dest_file = '/tmp/'+ name
blob.download_to_filename(dest_file)
dataset=pd.read_csv(dest_file)
dataset.head()

features = dataset[['Levenshtein_distance']]

#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(features)
#visulaizing the clusters
features=np.array(features)
plt.scatter(features[y_predict == 0], features[y_predict == 0], s = 100, c
= 'blue', label = 'Cluster 1') #for first cluster
plt.scatter(features[y_predict == 1], features[y_predict == 1], s = 100, c
= 'green', label = 'Cluster 2') #for second cluster
plt.scatter(features[y_predict== 2], features[y_predict == 2], s = 100, c =
'red', label = 'Cluster 3') #for third cluster
plt.scatter(features[y_predict == 3], features[y_predict == 3], s = 100, c
= 'cyan', label = 'Cluster 4') #for fourth cluster
plt.scatter(features[y_predict == 4], features[y_predict == 4], s = 100, c
= 'magenta', label = 'Cluster 5') #for fifth cluster
plt.scatter(kmeans.cluster_centers_[:], kmeans.cluster_centers_[:], s =
300, c = 'yellow', label = 'Centroid')
plt.title('Clusters of tweets')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

REFERENCES:

[1] "Cloud Storage documentation | Google Cloud", *Google Cloud*, 2022. [Online]. Available: https://cloud.google.com/storage/docs. [Accessed: 08- Jul- 2022]

[2] A. Chhabra, "Upload Files to Google Cloud Storage with Python - DZone Cloud", *dzone.com*, 2020. [Online]. Available: https://dzone.com/articles/upload-files-to-google-cloud. [Accessed: 08- Jul- 2022]

[3] "Cloud Functions | Google Cloud", *Google Cloud*, 2022. [Online]. Available: https://cloud.google.com/functions#section-4. [Accessed: 08- Jul- 2022]

[4] "K-Means Clustering Algorithm - Javatpoint", *www.javatpoint.com*, 2022. [Online]. Available: https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning. [Accessed: 08- Jul- 2022]

[5] "Vertex AI | Google Cloud", *Google Cloud*, 2022. [Online]. Available: https://cloud.google.com/vertex-ai?hl=en. [Accessed: 08- Jul- 2022]