

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.

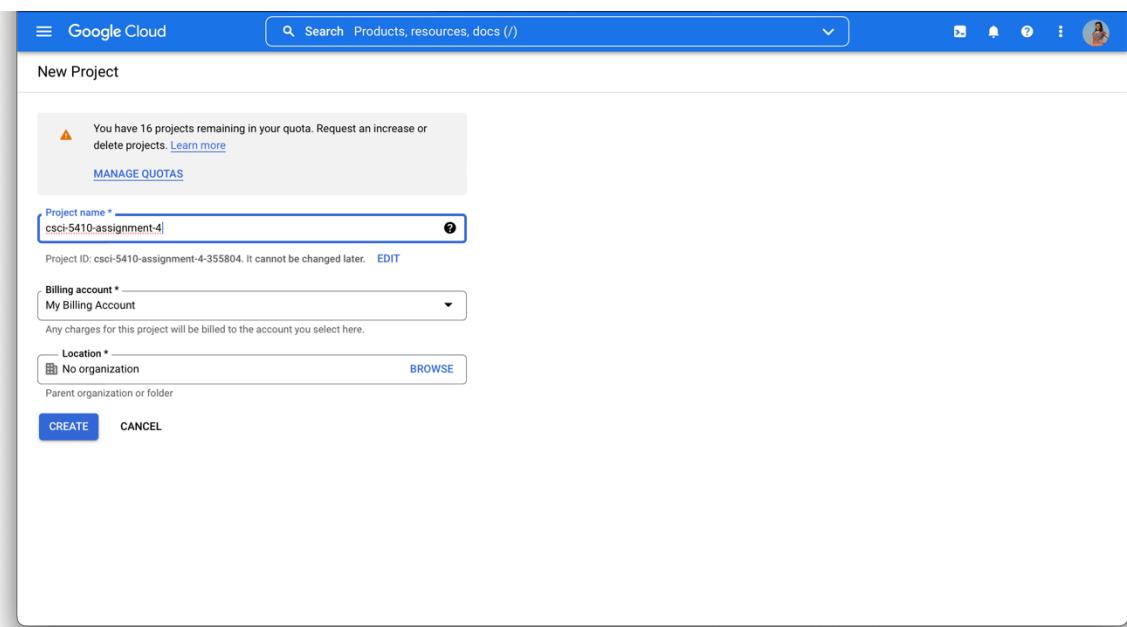


Figure 1. Screenshot of creating project in Google Cloud Console.

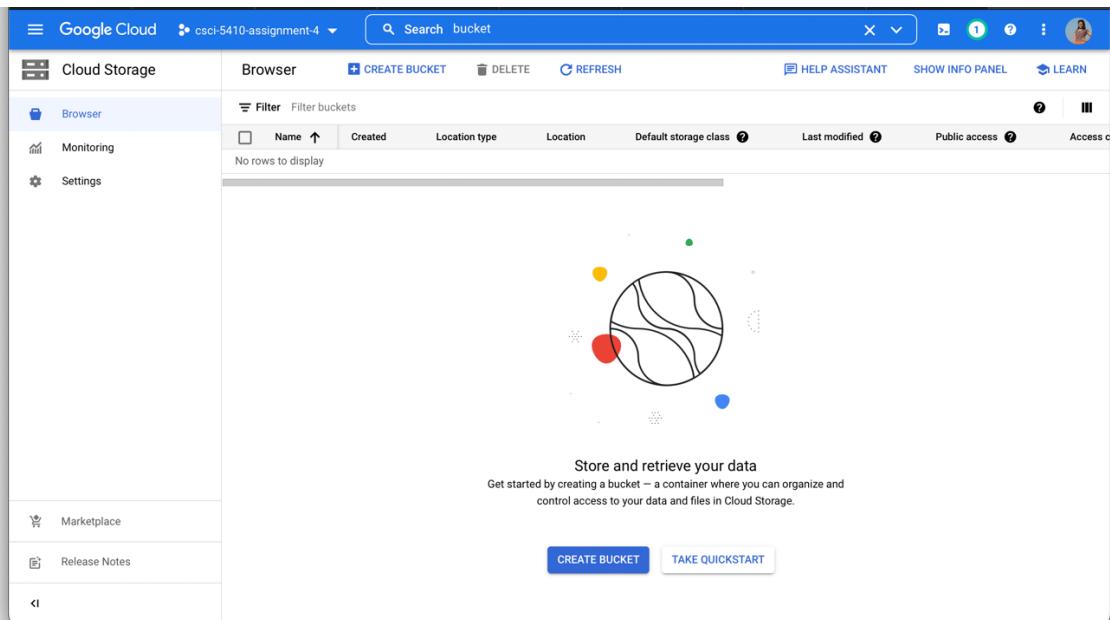


Figure 2. Screenshot of empty Cloud Storage[1] in Google Cloud Console.

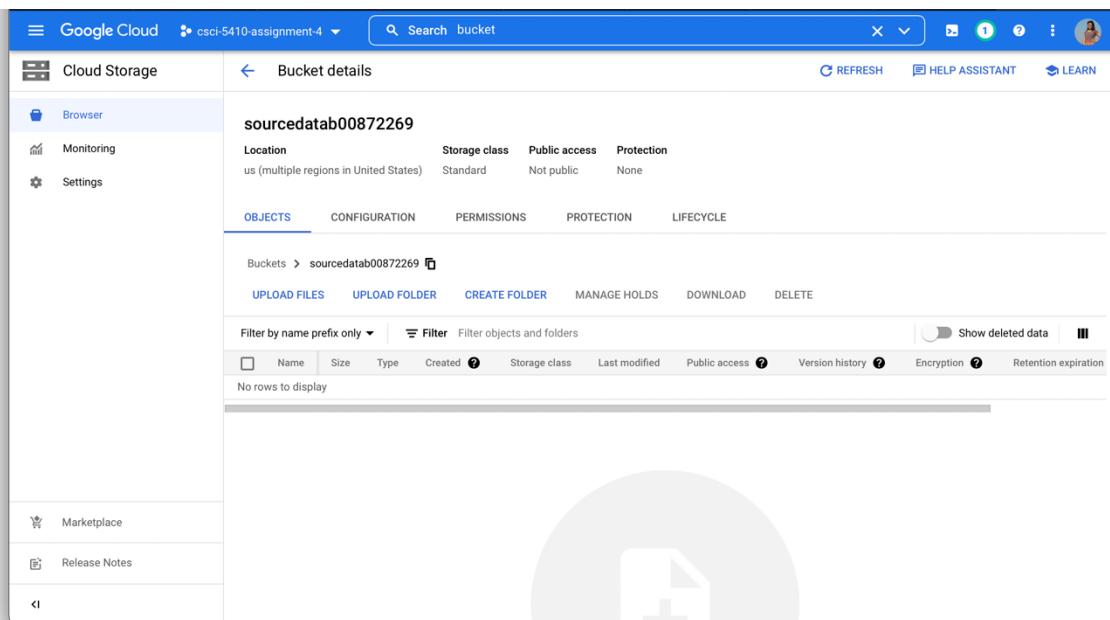
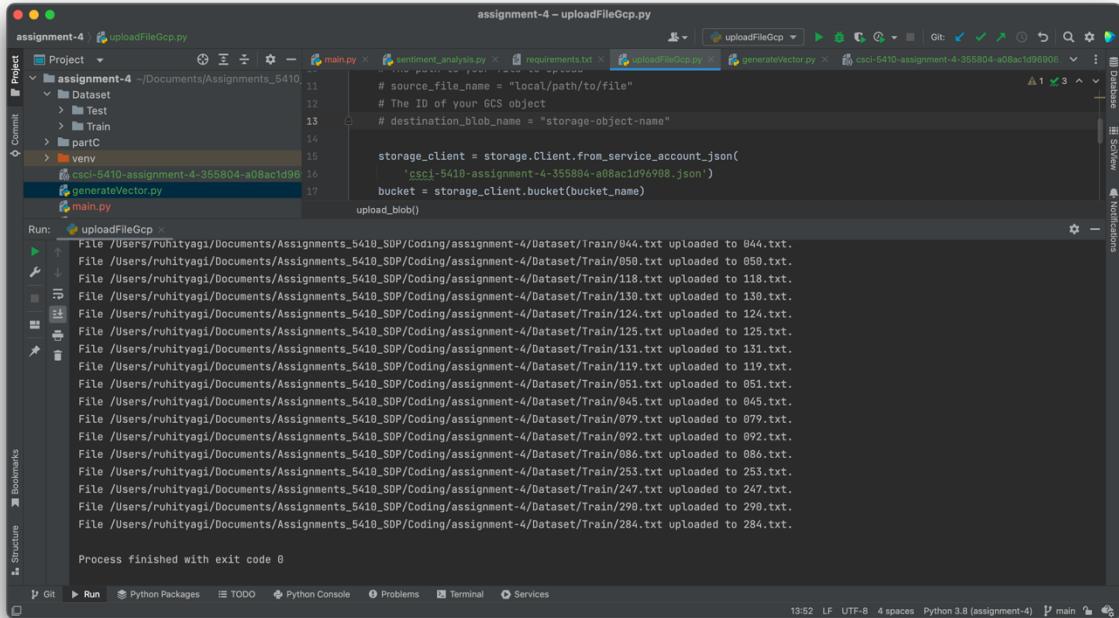


Figure 3. Screenshot of empty sourcedatab00872269 bucket .



The screenshot shows the PyCharm IDE interface with a project named 'assignment-4'. The 'uploadFileGcp.py' file is open, containing Python code for uploading files to Google Cloud Storage. The code uses the `storage.Client` class from the `google-cloud-storage` library. It specifies the source file name, destination blob name, and service account JSON key ('csci-5410-assignment-4-355804-a08acd1d9808.json'). The 'Run' tab shows the output of the upload process, listing 299 files from '001.txt' to '299.txt' being uploaded to the 'Train' directory in the 'sourcedatab00872269' bucket.

```

assignment-4 - uploadFileGcp.py
Project: assignment-4 ~/Documents/Assignments_5410
Commit: csci-5410-assignment-4-355804-a08acd1d9808
  Dataset
  Test
  Train
  venv
    csci-5410-assignment-4-355804-a08acd1d9808.json
    generateVector.py
  main.py

Run: uploadFileGcp
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/044.txt uploaded to 044.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/050.txt uploaded to 050.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/118.txt uploaded to 118.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/130.txt uploaded to 130.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/124.txt uploaded to 124.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/125.txt uploaded to 125.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/131.txt uploaded to 131.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/119.txt uploaded to 119.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/051.txt uploaded to 051.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/045.txt uploaded to 045.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/079.txt uploaded to 079.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/092.txt uploaded to 092.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/086.txt uploaded to 086.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/253.txt uploaded to 253.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/247.txt uploaded to 247.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/290.txt uploaded to 290.txt.
  File /Users/ruhirajyagi/Documents/Assignments_5410_SD/P/Coding/assignment-4/Dataset/Train/284.txt uploaded to 284.txt.

Process finished with exit code 0

```

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

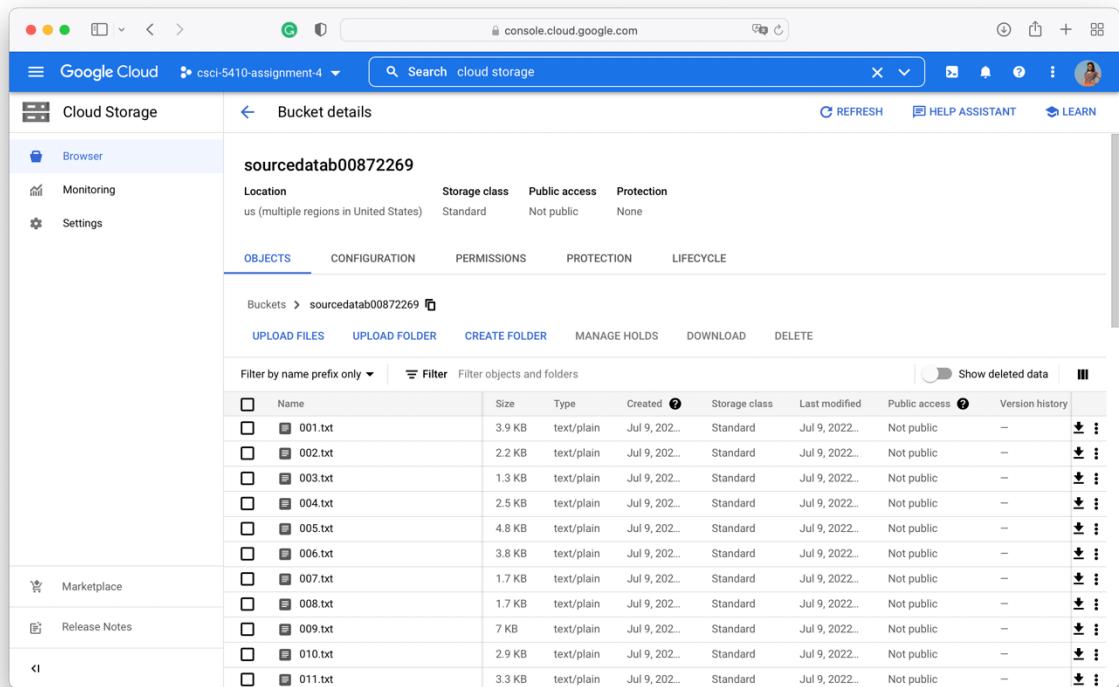
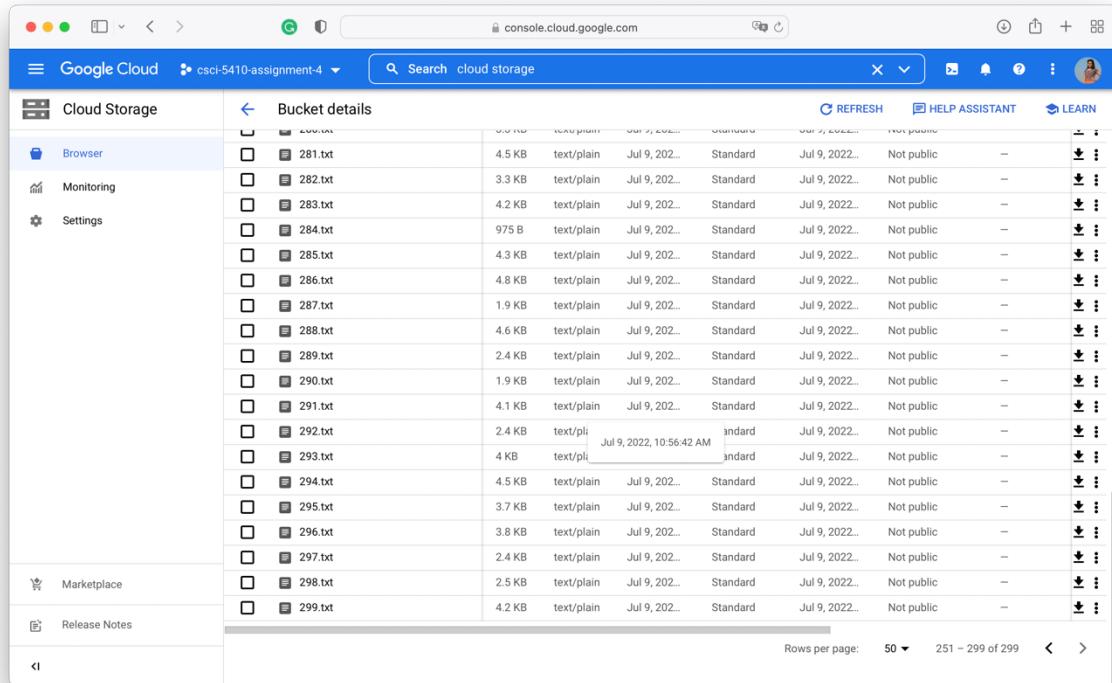


Figure 5(a). Screenshot of SUCCESSFUL upload of files 001-299 to sourcedatab00872269 bucket.

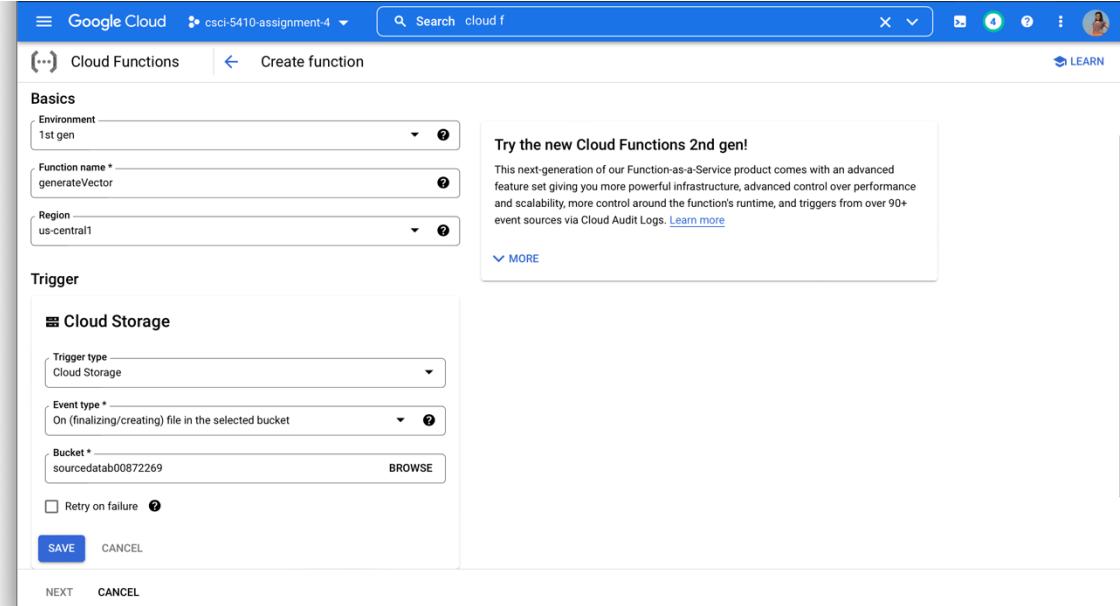


The screenshot shows the Google Cloud Storage interface. On the left, there's a sidebar with 'Cloud Storage', 'Browser', 'Monitoring', and 'Settings'. The main area is titled 'Bucket details' and shows a list of files in the '200.txt' folder. The files are listed as follows:

	File Name	Size	Type	Last Modified	Storage Class	Created	Access Control	Action
280.txt	280.txt	0.0 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	281.txt	4.5 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	282.txt	3.3 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	283.txt	4.2 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	284.txt	975 B	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	285.txt	4.3 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	286.txt	4.8 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	287.txt	1.9 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	288.txt	4.6 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	289.txt	2.4 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	290.txt	1.9 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	291.txt	4.1 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	292.txt	2.4 KB	text/plain	Jul 9, 2022, 10:56:42 AM	Standard	Jul 9, 2022...	Not public	⋮
	293.txt	4 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	294.txt	4.5 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	295.txt	3.7 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	296.txt	3.8 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	297.txt	2.4 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	298.txt	2.5 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮
	299.txt	4.2 KB	text/plain	Jul 9, 2022...	Standard	Jul 9, 2022...	Not public	⋮

At the bottom, it says 'Rows per page: 50 ▾ 251 – 299 of 299'.

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



The screenshot shows the 'Create function' page in Google Cloud Functions. The 'Basics' section includes:

- Environment: 1st gen
- Function name: generateVector
- Region: us-central1

A callout box for 'Try the new Cloud Functions 2nd gen!' is present, mentioning the next-generation product's advanced features like powerful infrastructure, advanced control over performance and scalability, and triggers from over 90+ event sources via Cloud Audit Logs. A 'MORE' link is also shown.

The 'Trigger' section is set to 'Cloud Storage' with the following configuration:

- Trigger type: Cloud Storage
- Event type: On (finalizing/creating) file in the selected bucket
- Bucket: sourcedatab00872269
- Retry on failure:

At the bottom, there are 'SAVE' and 'CANCEL' buttons, along with 'NEXT' and 'CANCEL' links.

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

The screenshot shows the Google Cloud Functions interface. In the top navigation bar, it says "Google Cloud" and "csci-5410-assignment-4". A search bar contains "Search cloud fun". On the right, there are icons for "X", "25", "LEARN", and a user profile.

The main area is titled "Cloud Functions" and "Edit function". Below that, "Configuration" and "Code" tabs are shown, with "Code" being active.

Under "Runtime", "Python 3.8" is selected. The "Entry point" is set to "generateVector".

The "Source code" section shows "Inline Editor". The code editor contains the following Python script:

```
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']}.")

    storage_client = storage.Client()
    bucket = storage_client.get_bucket(event['bucket'])
    blob = bucket.blob(event['name'])
    contents = blob.download_as_text()
    print(contents)

    results = []
```

Below the code editor, there are buttons for "PREVIOUS", "DEPLOY", and "CANCEL".

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.

The screenshot shows the Google Cloud Storage interface for the bucket **traindatab00872269**. The bucket details page is displayed, showing the location as "us (multiple regions in United States)", storage class as "Standard", public access as "Not public", and protection as "None". The "OBJECTS" tab is selected, showing a table with columns: Name, Size, Type, Created, Storage class, Last modified, Public access, Version history, Encryption, and Retention expiration. A search bar at the top right allows filtering by name prefix. The message "No rows to display" is shown below the table.

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

The screenshot shows the Google Cloud Storage interface for the bucket **traindatab00872269**. The bucket details page is displayed, showing the location as "us (multiple regions in United States)", storage class as "Standard", public access as "Not public", and protection as "None". The "OBJECTS" tab is selected, showing a table with columns: Name, Size, Type, Created, Storage class, Last modified, Public access, Version history, Encryption, and Retention expiration. A search bar at the top right allows filtering by name prefix. The table now lists a single object named **trainVector.csv**, which is 237 B in size, a text/csv type, created on Jul 9, 2022, and last modified on Jul 9, 2022. The public access is set to "Not public". A note at the bottom left indicates an open link to the object's details page.

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

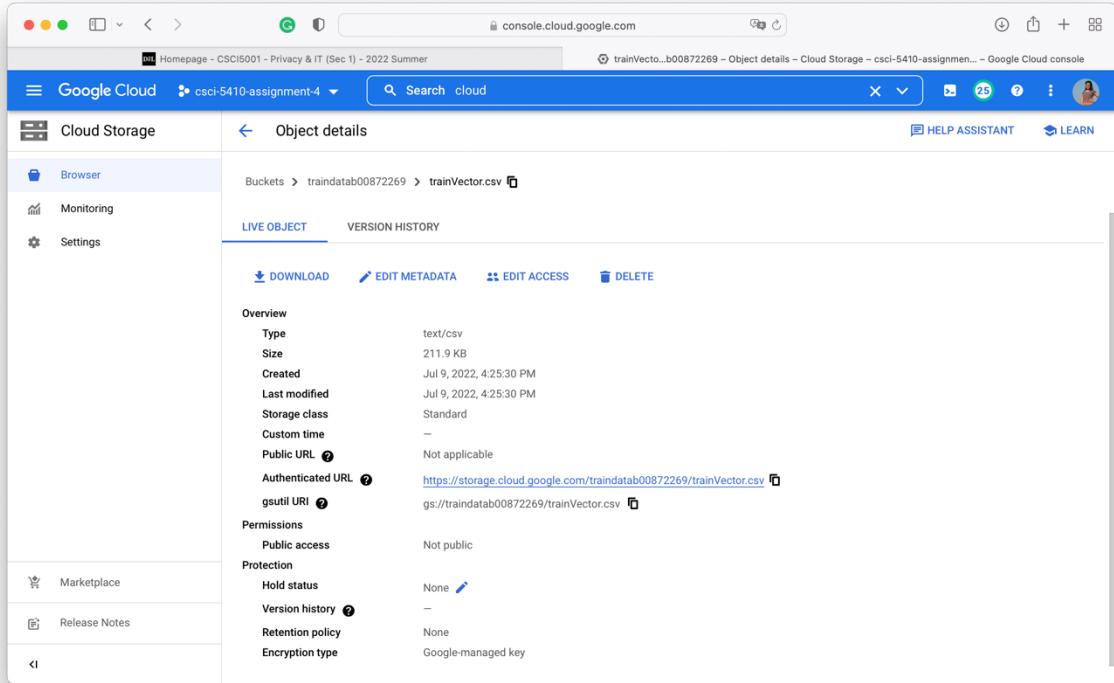


Figure 10. Screenshot of details of trainVector.csv in cloud storage.

This screenshot shows the contents of the 'trainVector' sheet in Google Sheets. The data is presented in a table with three columns: 'Current_Word', 'Next_Word', and 'Levenshtein_distance'. The table rows are as follows:

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	Vodafone	7
Vodafone	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
Mike	Vodafone	0

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.

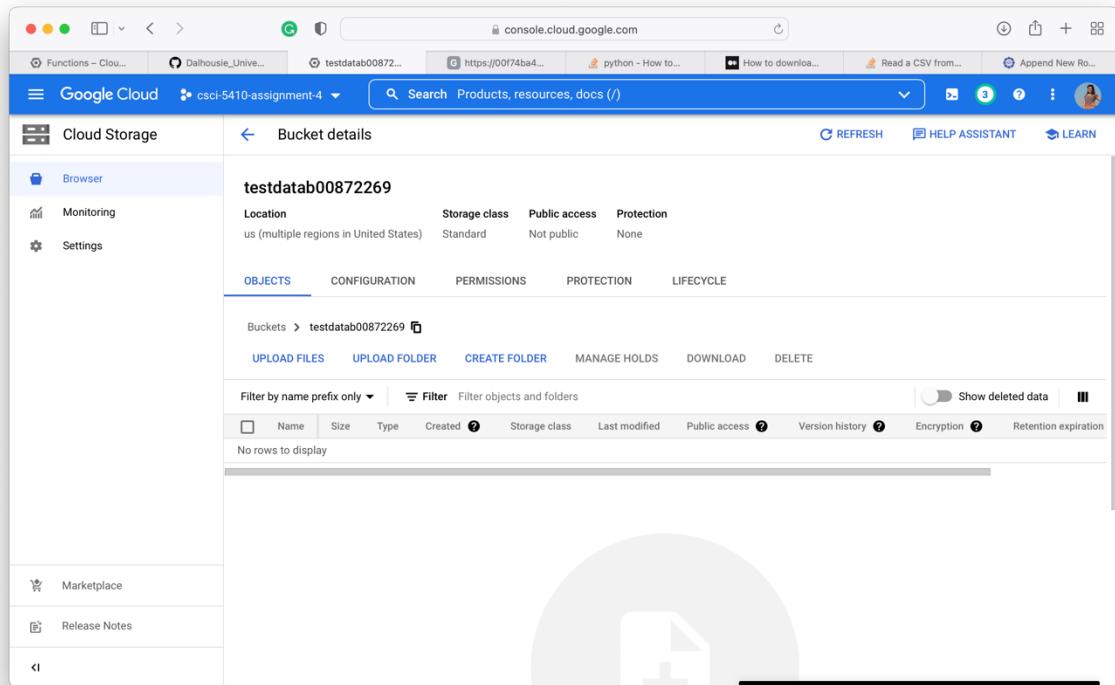


Figure 12. Screenshot of empty **testdatab00872269** BEFORE running **generateVector** cloud function.

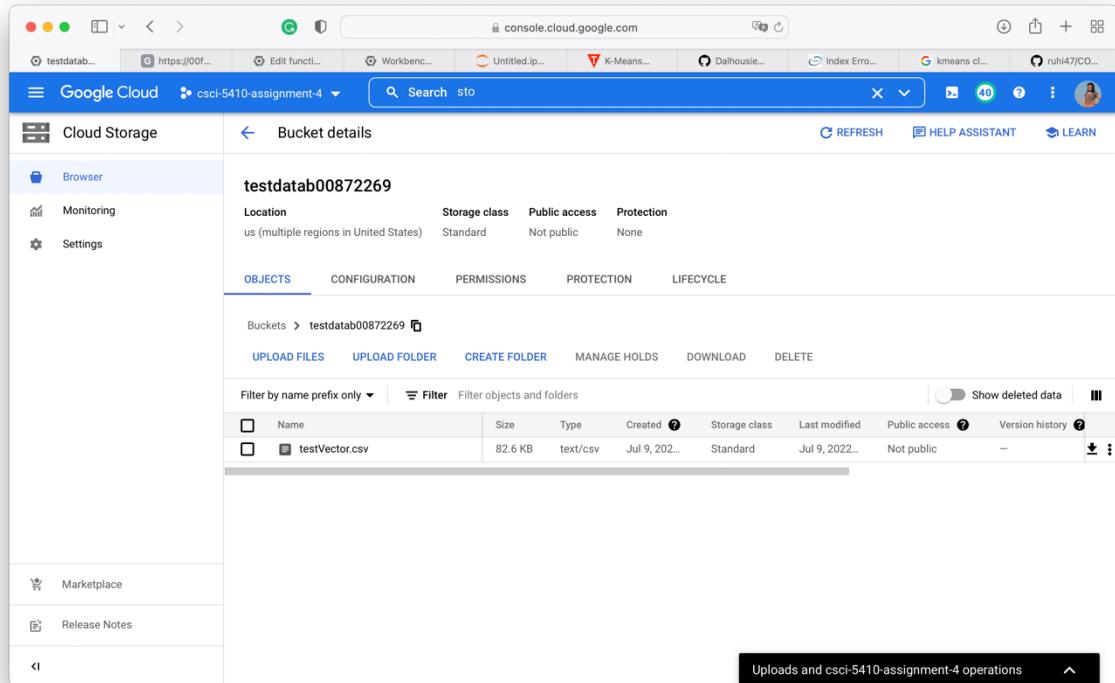


Figure 13. Screenshot of **testdatab00872269** AFTER running **generateVector** cloud function.

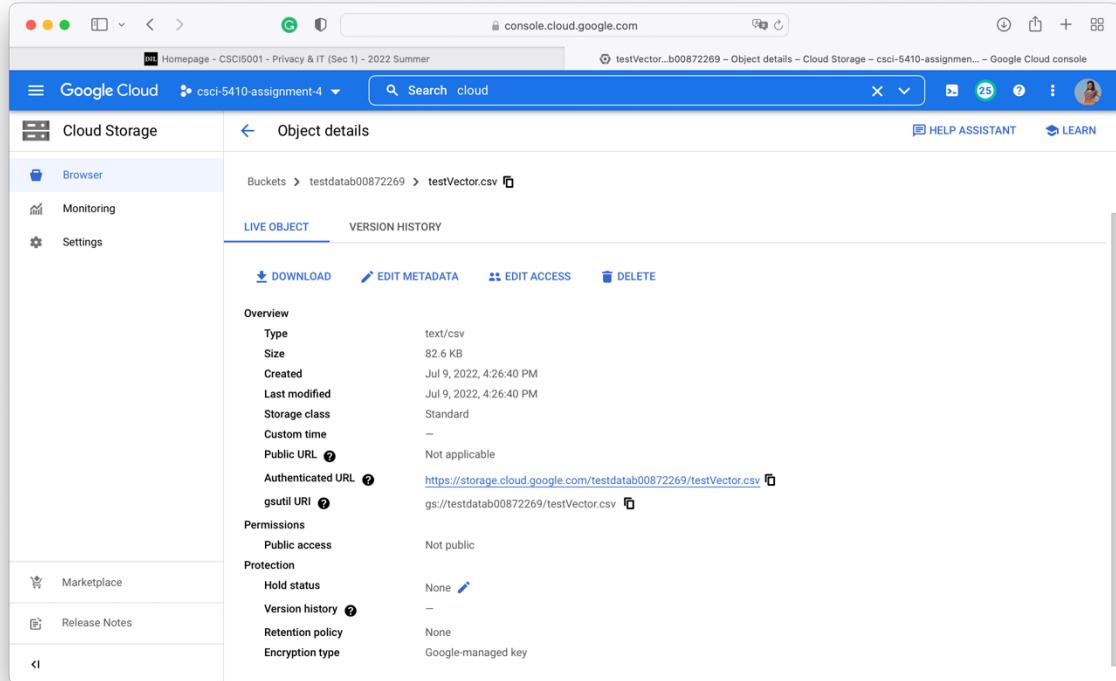
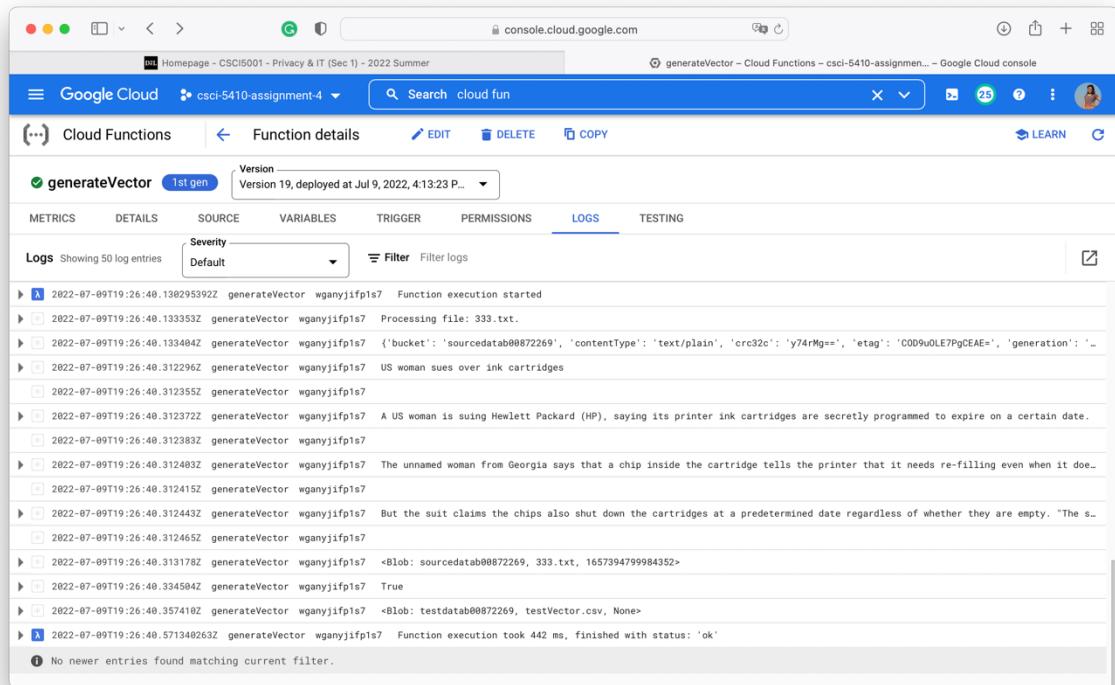


Figure 14. Screenshot of details of testVector.csv in cloud storage.

This screenshot shows the 'testVector' sheet in Google Sheets. The table has three columns: 'Current_Word', 'Next_Word', and 'Levenshtein_distance'. The data is as follows:

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
UK	UK	4

Figure 15. Screenshot of testVector.csv contents.



The screenshot shows the Google Cloud Functions interface for the 'generateVector' function. The 'LOGS' tab is selected, displaying 50 log entries. The logs show the execution of the function, starting with the file being processed ('Processing file: 333.txt') and ending with the successful completion ('Function execution took 442 ms, finished with status: 'ok''). The logs also mention a woman suing Hewlett Packard over printer ink cartridges.

```
2022-07-09T19:26:40.130295392Z generateVector wganyjifp1s7 Function execution started
2022-07-09T19:26:40.133353Z generateVector wganyjifp1s7 Processing file: 333.txt.
2022-07-09T19:26:40.134044Z generateVector wganyjifp1s7 {'bucket': 'sourcedatab00872269', 'contentType': 'text/plain', 'crc32c': 'y74rMg==', 'etag': 'C0D9uOLE7PgCEAE=', 'generation': '...
2022-07-09T19:26:40.312296Z generateVector wganyjifp1s7 US woman sues over ink cartridges
2022-07-09T19:26:40.312355Z generateVector wganyjifp1s7
2022-07-09T19:26:40.312372Z generateVector wganyjifp1s7 A US woman is suing Hewlett Packard (HP), saying its printer ink cartridges are secretly programmed to expire on a certain date.
2022-07-09T19:26:40.312383Z generateVector wganyjifp1s7
2022-07-09T19:26:40.312403Z generateVector wganyjifp1s7 The unnamed woman from Georgia says that a chip inside the cartridge tells the printer that it needs re-filling even when it doe...
2022-07-09T19:26:40.312415Z generateVector wganyjifp1s7
2022-07-09T19:26:40.312443Z generateVector wganyjifp1s7 But the suit claims the chips also shut down the cartridges at a predetermined date regardless of whether they are empty. "The s...
2022-07-09T19:26:40.312465Z generateVector wganyjifp1s7
2022-07-09T19:26:40.313178Z generateVector wganyjifp1s7 <Blob: sourcedatab00872269, 333.txt, 1657394799984352>
2022-07-09T19:26:40.334594Z generateVector wganyjifp1s7 True
2022-07-09T19:26:40.357410Z generateVector wganyjifp1s7 <Blob: testdatab00872269, testVector.csv, None>
2022-07-09T19:26:40.571340263Z generateVector wganyjifp1s7 Function execution took 442 ms, finished with status: 'ok'
No newer entries found matching current filter.
```

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].

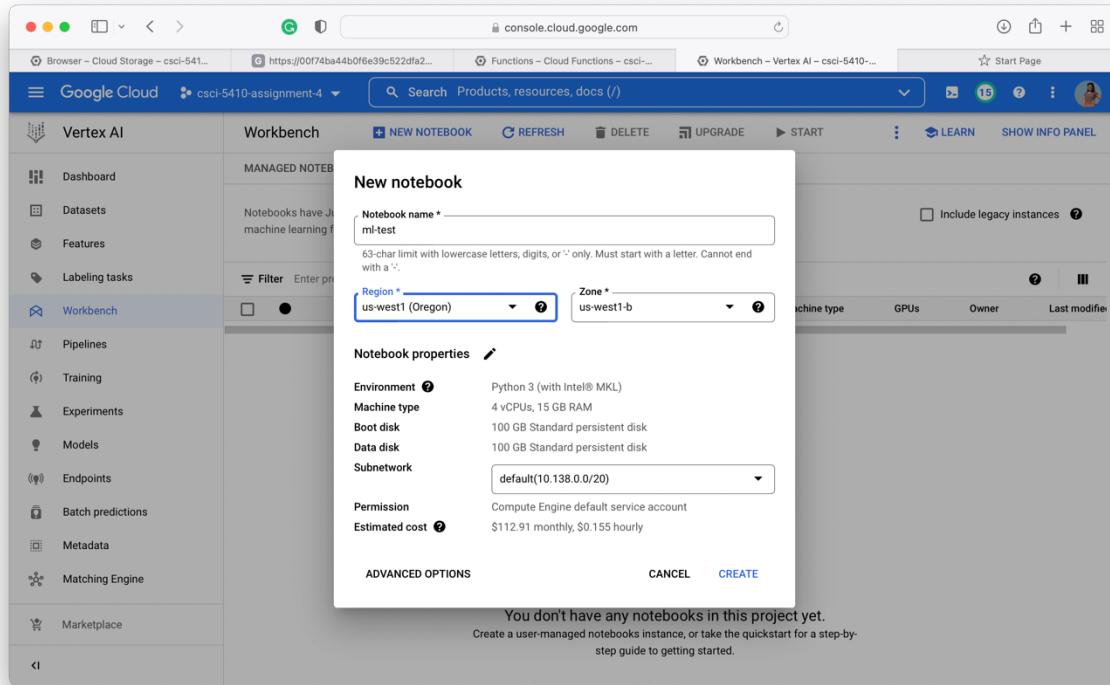


Figure 17. Screenshot of configuring jupyter notebook to perform GCP ML task in Vertex AI [5].

```

[95]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from google.cloud import storage

[96]: name = 'trainVector.csv'
bucket_name = 'traindataab00872269'

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)

[97]: dataset.head()

[97]:
  Current_Word  Next_Word  Levenshtein_distance
0        Mobile      Mobile                  1
1       Mobile        UK                   6
2         UK     Britains                  8
3    Britains   Vodafone                  7
4   Vodafone     January                  8

[98]: features = dataset[['Levenshtein_distance']]

[99]: from sklearn.cluster import KMeans
wcss_list= [] #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.

```

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

The screenshot shows a Jupyter Notebook interface with a Python code cell and its corresponding output plot. The code implements the K-Means clustering algorithm and plots the WCSS values for different numbers of clusters (from 1 to 10). The resulting graph, titled 'The Elbow Method Graph', shows a clear bend or 'elbow' at k=5, where the rate of decrease in WCSS begins to level off.

```
[90]: features = dataset[['Levenshtein_distance']]
[91]: from sklearn.cluster import KMeans
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()
```

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

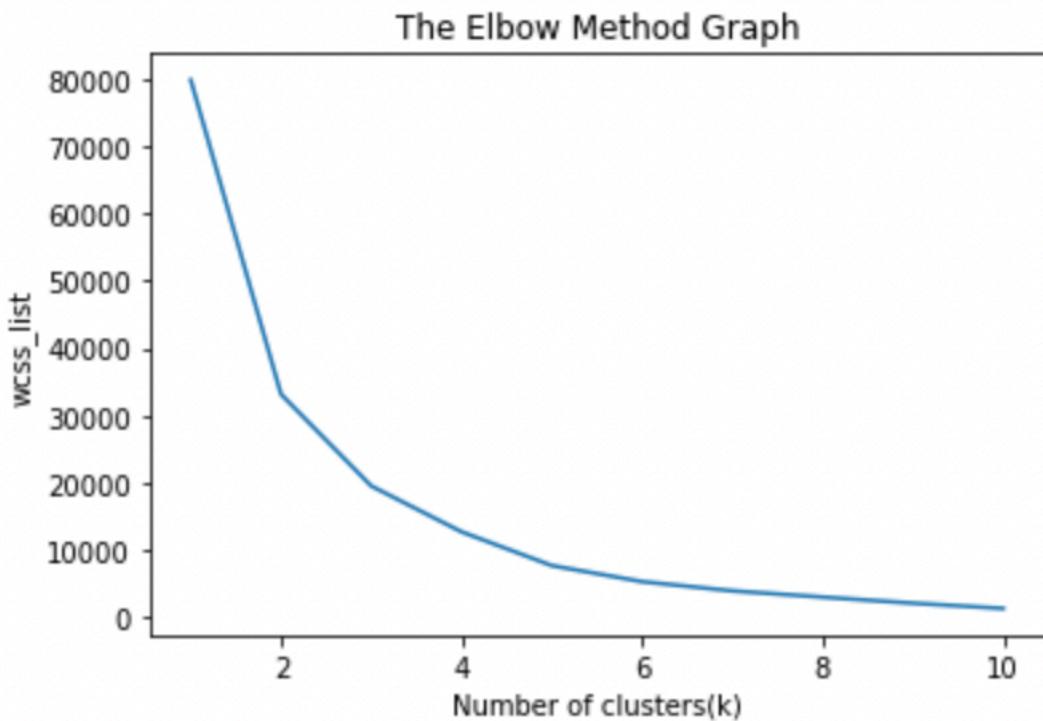


Figure 19(b). Screenshot of elbow method graph bending at $K=5$.

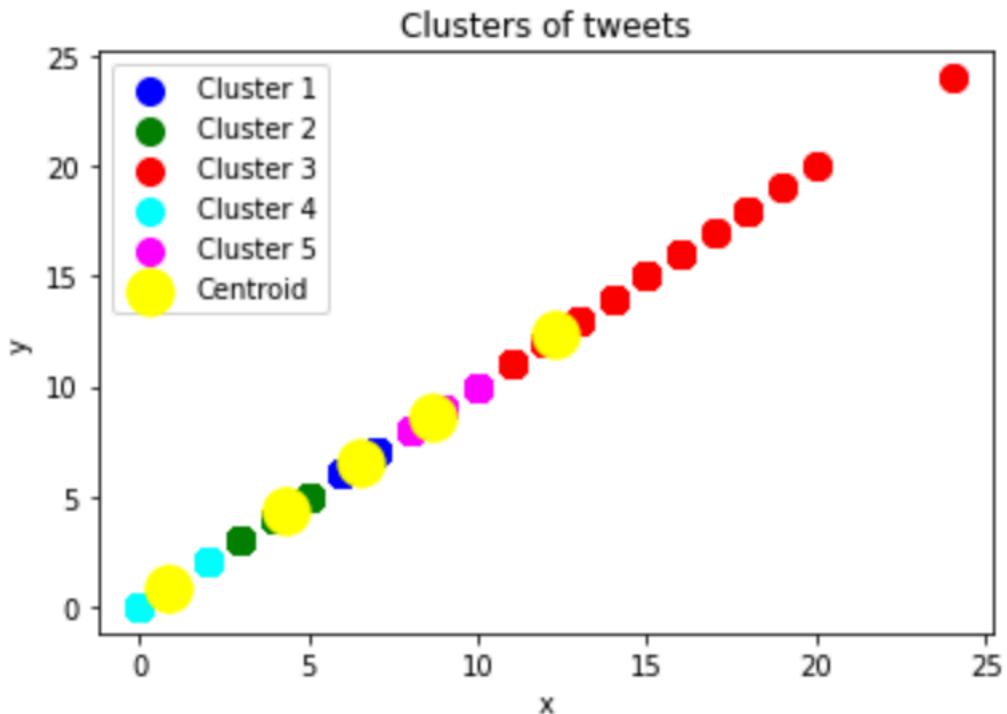


Figure 20. Screenshot of clusters with centroid and outliers from trainVector.csv data.

```

Untitled.ipynb ml-test.ipynb
File Edit View Run Kernel Git Tabs Settings Help
+ - Code git Python 3
Filter files by name
Name Last Modified
src 5 hours ago
tutorials 5 hours ago
ml-test.ipynb 4 hours ago
Untitled.ipynb a minute ago
[94]: name = 'testVector.csv'
bucket_name = 'testdata00872269'

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)

[82]: dataset.head()

[82]:


|   | 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                    |



[83]: features = dataset[['Levenshtein_distance']]

[84]: #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)

```

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

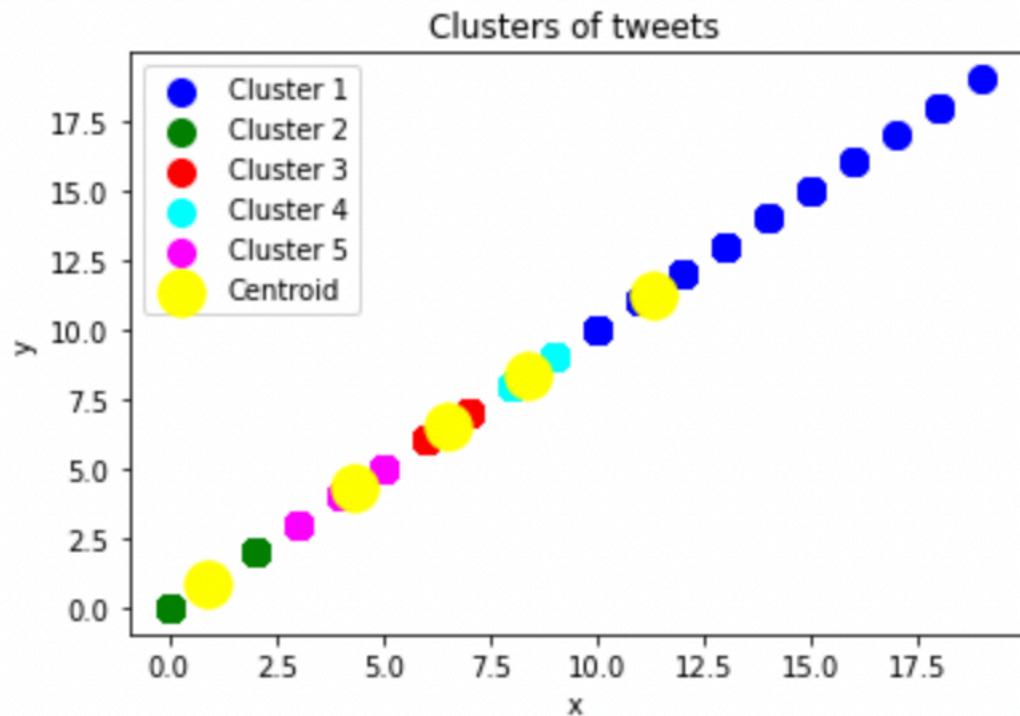


Figure 22. Screenshot of clusters with centroid and outliers from *testVector.csv* data.

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 = 'traindataab00872269'
else:
    name = 'testVector.csv'
    bucket_name = 'testdataab00872269'

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-a08acd96908.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 = 'traindataab00872269'

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_[:, 0], kmeans.cluster_centers_[:, 1], 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 = 'testdataab00872269'

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)
#visualizing 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_[:,0], kmeans.cluster_centers_[:,1], 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]