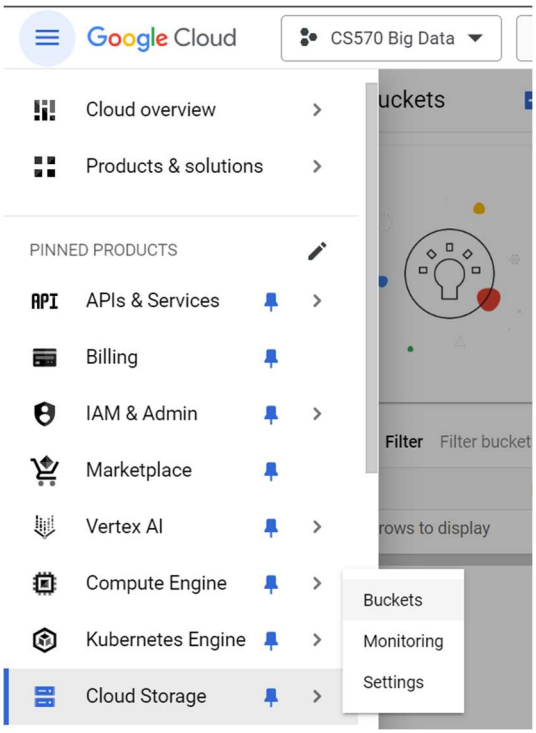
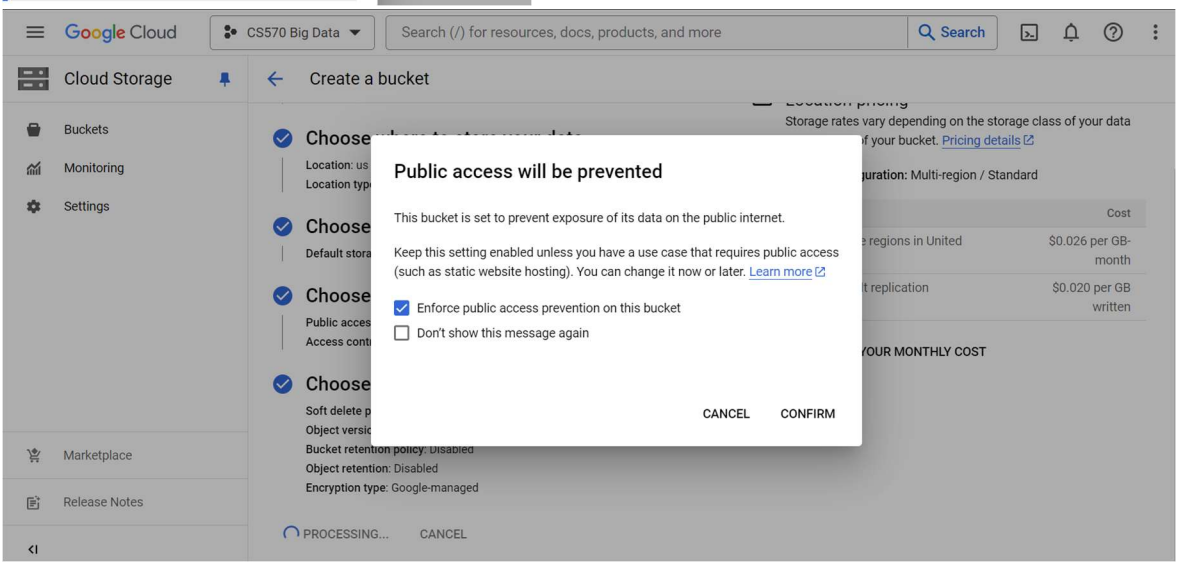
Week 8: Homework 1: Project: Movie Recommendation with MLlib - Collaborative Filtering (implementaiton 2)

Steps to complete the project:

1. Create a bucket



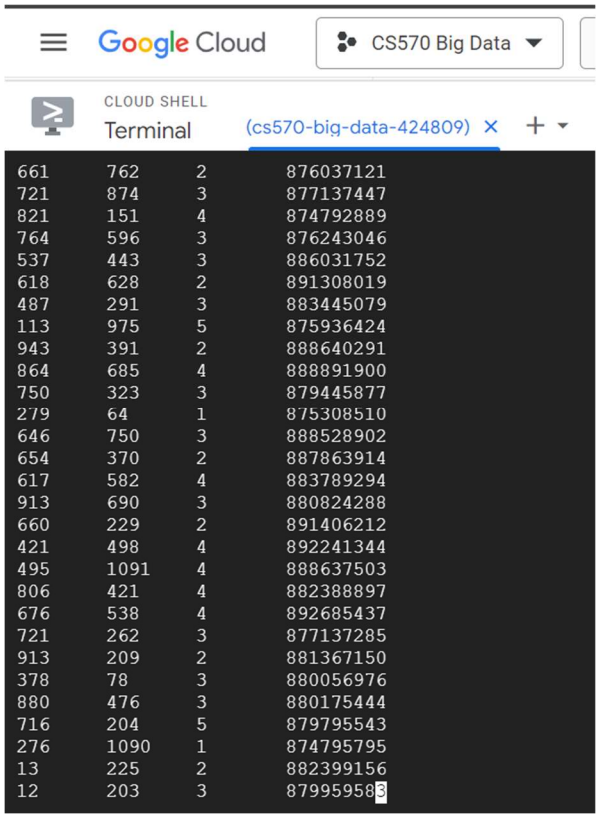


Step1: Preparing and Transform data

For this part of the project, we need to load the data we need for the recommendation system and save it in our cloud.

1. **Create a file named u.data** and load the rating data into this file. The data has an initial format of (UserID, MovieID, rating, Timestamp)

```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ vim u.data
shagos90499@cloudshell:~ (cs570-big-data-424809)$
```

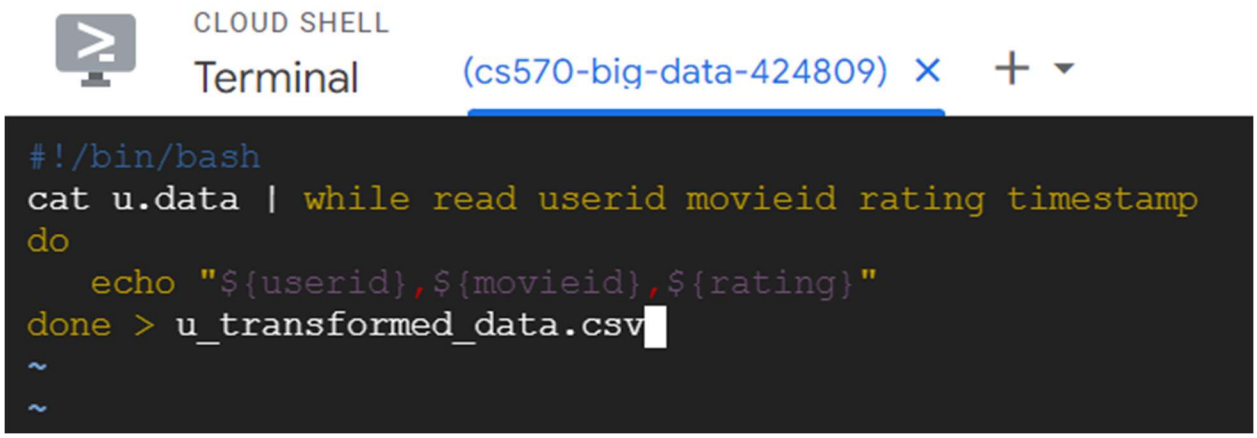


2. **Transform the data**: next we need to convert the initial format to (UserID, MovieID, rating) this format. So, we can use a bash script to do that.
The script can be formatted as follows:

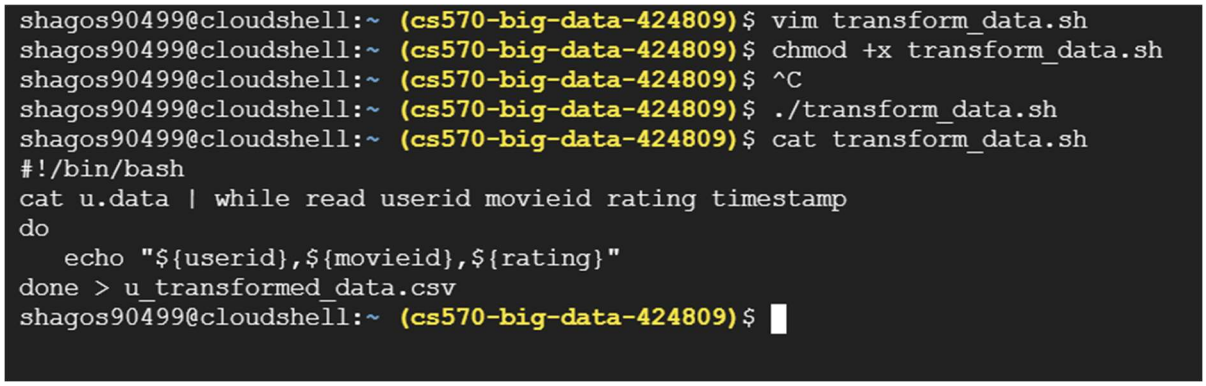
```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ vim transform_data.sh
```

- First copy this bash script into a script file called ***transform_data.sh***

```
#!/bin/bash
cat u.data | while read userid movieid rating timestamp
do
   echo "${userid},${movieid},${rating}"
done > u_transformed_data.csv
```



- Make the file executable:
  `chmod +x transform_data.sh`
- Execute the script
  `./transform_data.sh`



The shell scripts read each line of the u.data file, split the data by spaces, and reformat it to display only the UserID, MovieID, and rating fields separated by commas. This transformation removes the timestamp field and formats the data into CSV format, which is easier to work with for further processing. The transformed data will be saved in the file u_transformed_data.csv.

Step2: Uploading the data to cloud bucket

Next, we will upload the transformed data that we saved in the u_transformed_data.csv to our cloud storage bucket.

```
gsutil cp u_transformed_data.csv gs://big_data_movie_recommendation
```

```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ gsutil cp u_transformed_data.csv gs://big_data_movie_recommendation
Copying file://u_transformed_data.csv [Content-Type=text/csv]...
/ [1 files][956.2 KiB/956.2 KiB]
Operation completed over 1 objects/956.2 KiB.
shagos90499@cloudshell:~ (cs570-big-data-424809)$
```

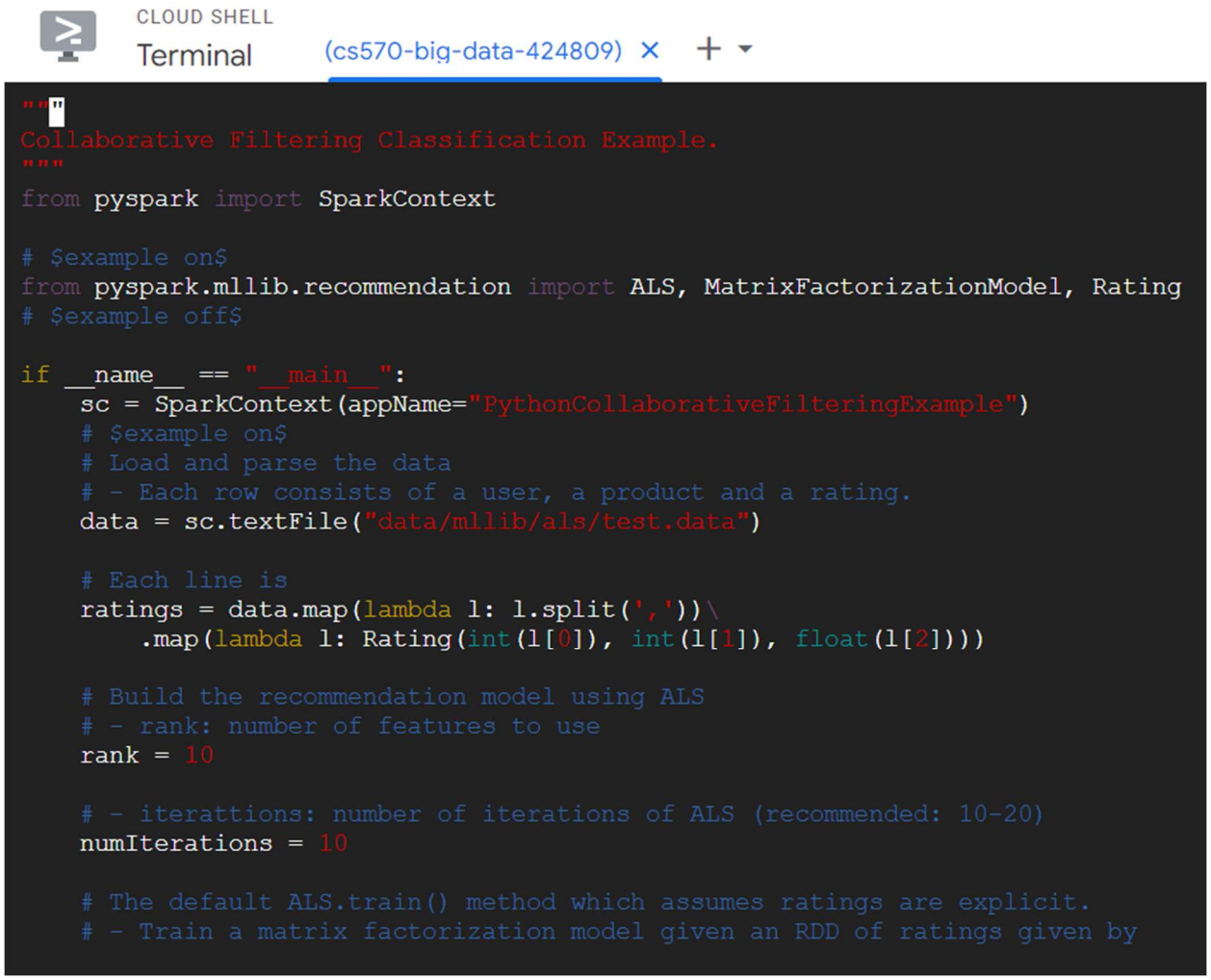
Step 3: Create and upload the PySpark script

In this step, we will create the pyspark script that will perform the collaborative filtering and upload that file to the cloud storage bucket.

1. Create a file named **recommendation_example.py**
   `vim recommendation_example.py`
2. **Copy this script** in the file but replace the path to the text file since currently it is working from another link. Replace it with the correct path to your pyspark script in the cloud bucket.

```
"""
Collaborative Filtering Classification Example.
"""
from pyspark import SparkContext

# $example on$
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel,
Rating
# $example off$

if __name__ == "__main__":
    sc = SparkContext(appName="PythonCollaborativeFilteringExample")
    # $example on$
    # Load and parse the data
    # - Each row consists of a user, a product and a rating.
    data =
sc.textFile("gs://big_data_movie_recommendation/u_transformed_data.csv")

    # Each line is
    ratings = data.map(lambda l: l.split(','))\
        .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))

    # Build the recommendation model using ALS
    # - rank: number of features to use
    rank = 10

    # - iterattions: number of iterations of ALS (recommended: 10-20)
    numIterations = 10

    # The default ALS.train() method which assumes ratings are explicit.
    # - Train a matrix factorization model given an RDD of ratings given by
    #   users to some products, in the form of (userID, productID, rating)
pairs.
    # - We approximate the ratings matrix as the product of two lower-rank
    #   matrices of a given rank (number of features).
    #   + To solve for these features, we run a given number of
    #     iterations of ALS.
    #   + The level of parallelism is determined automatically based
    #     on the number of partitions in ratings.
    model = ALS.train(ratings, rank, numIterations)
```

```
    ############################################################
    # Evaluate the model on training data
    # - Evaluate the recommendation model on rating training data
    ############################################################
    testdata = ratings.map(lambda p: (p[0], p[1]))
    predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]),
r[2]))


    # Join input rating ((user, product), rate1) with predicted rating
    # ((user, product), rate2) to create ((user, product), (rate1, rate2))
    ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]),
r[2])).join(predictions)
    MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
    print("Mean Squared Error = " + str(MSE))

    # Save and load model
    model.save(sc, "target/tmp/myCollaborativeFilter")
    sameModel = MatrixFactorizationModel.load(sc,
"target/tmp/myCollaborativeFilter")
    # $example off$
```



3. **Upload the script to the bucket:**
   ```
   gsutil cp recommendation_example.py
   gs://big_data_movie_recommendation
   ```

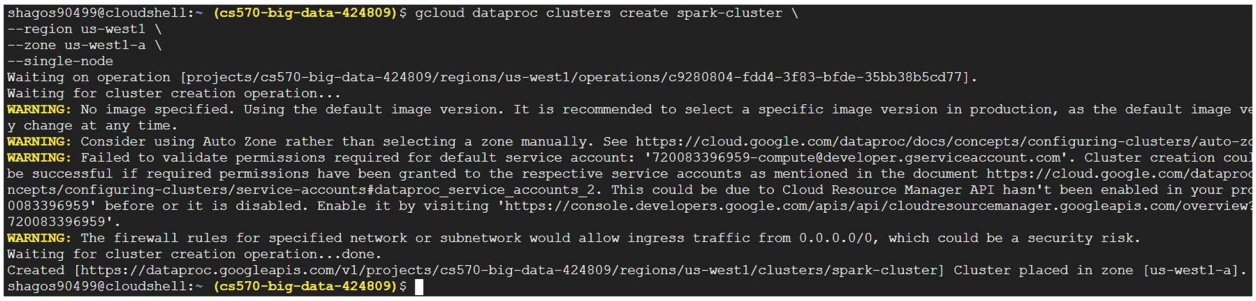
   This PySpark script implements collaborative filtering using the Alternating Least Squares (ALS) algorithm to build a movie recommendation system. It loads user-movie rating data, trains a recommendation model with the ALS algorithm, evaluates the model by calculating the Mean Squared Error (MSE) of predictions, and saves the trained model for future use.

Step 4: Submit the pyspark job to google Dataproc

To submit the pyspark script, first we need to create a Dataproc cluster and submit the script

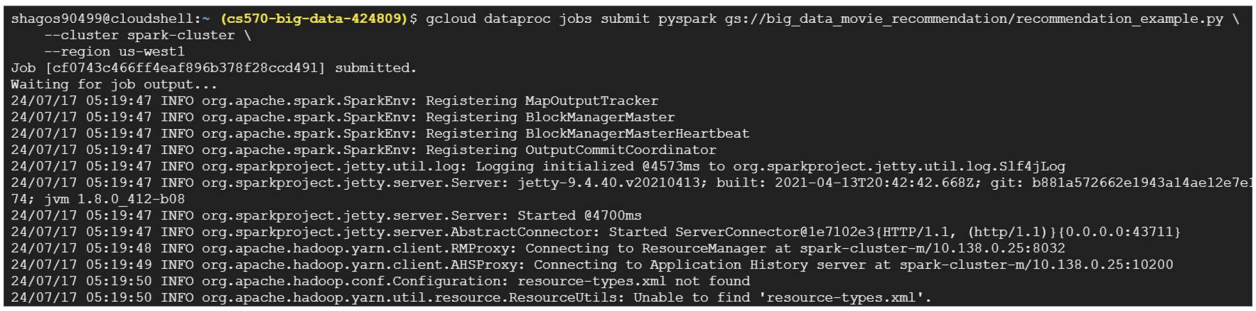
1. **Create a Dataproc cluster** with this command

```
gcloud dataproc clusters create spark-cluster \
--region us-west1 \
--zone us-west1-a \
--single-node
```



2. Next, submit the pyspark job to the Dataproc cluster

   ```
   gcloud dataproc jobs submit pyspark
   gs://big_data_ml_recommendation_sys/recommendation_example.py \
       --cluster spark-cluster \
       --region us-west1
   ```



```
driverOutputResourceUri: gs://dataproc-staging-us-west1-720083396959-ncaxf7jl/google-cloud-dataproc-m
69435d94d883933a2d1d56/driveroutput
jobUuid: bb828cfa-ea92-3b1a-bd8c-e9b1098ca204
placement:
  clusterName: spark-cluster
  clusterUuid: e25594a6-9c3d-4676-a348-e90edf4809ba
pysparkJob:
  mainPythonFileUri: gs://big_data_movie_recommendation/recommendation_example.py
reference:
  jobId: ba73e8893969435d94d883933a2d1d56
  projectId: cs570-big-data-424809
status:
  state: DONE
  stateStartTime: '2024-07-17T05:40:10.129484Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-07-17T05:39:04.600621Z'
- state: SETUP_DONE
  stateStartTime: '2024-07-17T05:39:04.628825Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-07-17T05:39:04.811591Z'
yarnApplications:
- name: PythonCollaborativeFilteringExample
  progress: 1.0
  state: FINISHED
  trackingUrl: http://spark-cluster-m:8088/proxy/application_1721193394462_0004/
```

```
Mean Squared Error = 0.48435603686025513
24/07/17 05:40:01 INFO org.apache.hadoop.m
```

3. The output of the pyspark job is as follows:
   The Mean Squared Error = 0.48435603686025513
   c

```
24/07/17 05:39:18 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
Mean Squared Error = 0.48435603686025513
24/07/17 05:40:01 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
24/07/17 05:40:01 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
24/07/17 05:40:01 WARN org.apache.hadoop.util.concurrent.ExecutorHelper: Thread (Thread[GetFileInf
java.lang.InterruptedException
```