Course Project: Big Data Concepts and Implementations

Aditya Shekhar Camarushy – adcama@iu.edu – 2000772747

I. INTRODUCTION

For my course project I opted for a Hands-On Project in which I leverage the Google Cloud Platform as my Choice of cloud platform to implement a Movie Recommender system for a given user using PySpark.

In addition to the recommender system, I also performed some data transformations, computation of summary stats and visualizations in a jupyter notebook on the Dataproc cluster.

The dataset I used is the movielens dataset by Grouplens, I used the MovieLens 1M Dataset which is a stable benchmark dataset consisting of 1 million movie ratings from 6000 users on 4000 movies, it was released in 2003.

II. BACKGROUND

I picked out the topic of recommender systems as it was one of the first things that sparked my interest in the field of Data science, especially given its ubiquity in our daily lives, we see it on Netflix for movie / TV show recommendations, we also use it on various e-commerce platforms such as Amazon.

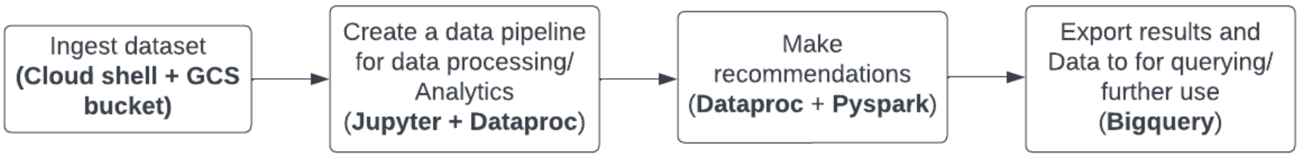
Furthermore, the scale of the data used on these platforms is massive (often in petabytes) and I saw it as a perfect opportunity to build a recommender system on the cloud as it can scale and has the tools to work with Big data. In fact, while researching on these platforms I found that they had a very similar tech stack to what I planned to use, for instance Netflix used a combination of Hadoop + Spark, MySQL and AWS while I was making use of similar technologies on the Google Cloud Platform.

As an avid movie/show aficionado it was also interesting for me to create pipeline and uncover trends through summary statistics and visualizations on the Movielens dataset.

In conclusion, I was motivated to work on this project as it would help my gain real world experience on an industry standard cloud platform while being able to meet my current course learning goals and also leveraging Machine Learning, Data mining skills from my past courses.

III. METHODOLODY

Before diving right into the methodology, I used and the steps to create a solution I would like to discuss the architecture / program flow that I had in mind to accomplish this project.



The flow of the program is as shown above –

1) The first step is to ingest the dataset from the external source i.e. Movielens in this case. I accomplished this using cloud shell as my staging area where I used the dataset download url and ran a curl command to download the zip file. The next this is to unzip the files in cloud shell and then move it to a google cloud storage bucket using a gsutil command. With this we have successfully ingested the dataset.

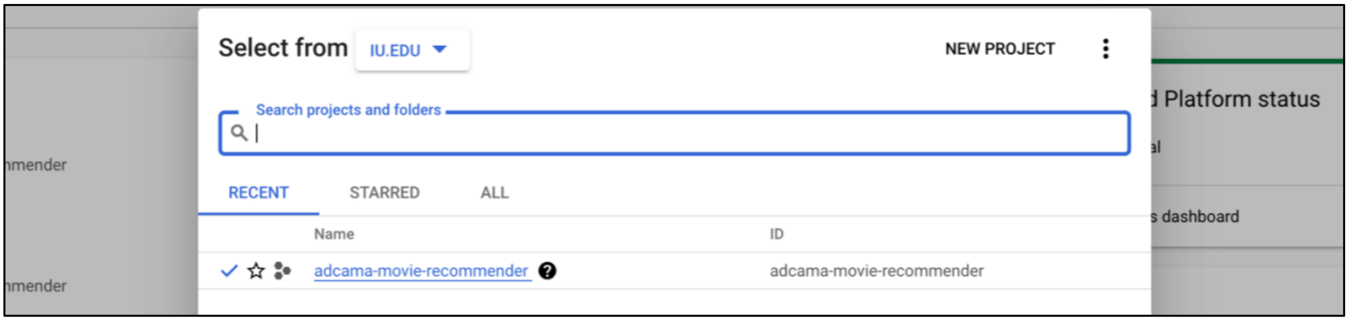
2) The next step to accomplish is to then load the datasets from the gcp buckets and perform some analysis on them. I do this by creating a Jupyter notebook instance in my dataproc cluster. In my analysis I initially transform the data in order to make analysis easier, then create some summary statistics on the dataset and finally produce visualizations using matplotlib that help us understand the dataset better.

3) Once the preliminary analysis is done, I then run a spark job on my dataproc cluster which accomplishes 2 things, first one is to generate the top n movie recommendations (I have chosen n = 3) for every user in the dataset. The second thing this spark job does is migrate the all the data (The original dataset along with the predictions) into a Bigquery dataset in the form of tables in order to create a data warehouse.

4) Finally we can navigate to big query and query the dataset based on our requirements, create views and perform other database operations.

We will now discuss the end-to-end steps I used to create the technological set up and steps taken to build the solution on Google Cloud Platform –

1) Create the project under **"FA22-BL-INFO-I535-ONLINE"** folder that falls under the IU.EDU organization.



2) Activate the cloud shell and create a Google cloud storage bucket (distributed file storage) to store the dataset. But before doing so it is crucial to **enable billing for your project.**

Here we use the gsutil mb command to make bucket.

```
adcama@cloudshell:~ (adcama-movie-recommender)$ gsutil mb -p adcama-movie-recommender -c STANDARD -l US-EAST5 -b on gs:
//adcama-movie-recommender-data
Creating gs://adcama-movie-recommender-data/...
adcama@cloudshell:~ (adcama-movie-recommender)$
```

3) Download the data into your cloud shell and move it to your gcp bucket. As seen in the image below we use the curl command to download the file, unzip the file where we are able to see our dataset that consists of 3 .data files corresponding to the movie ratings.

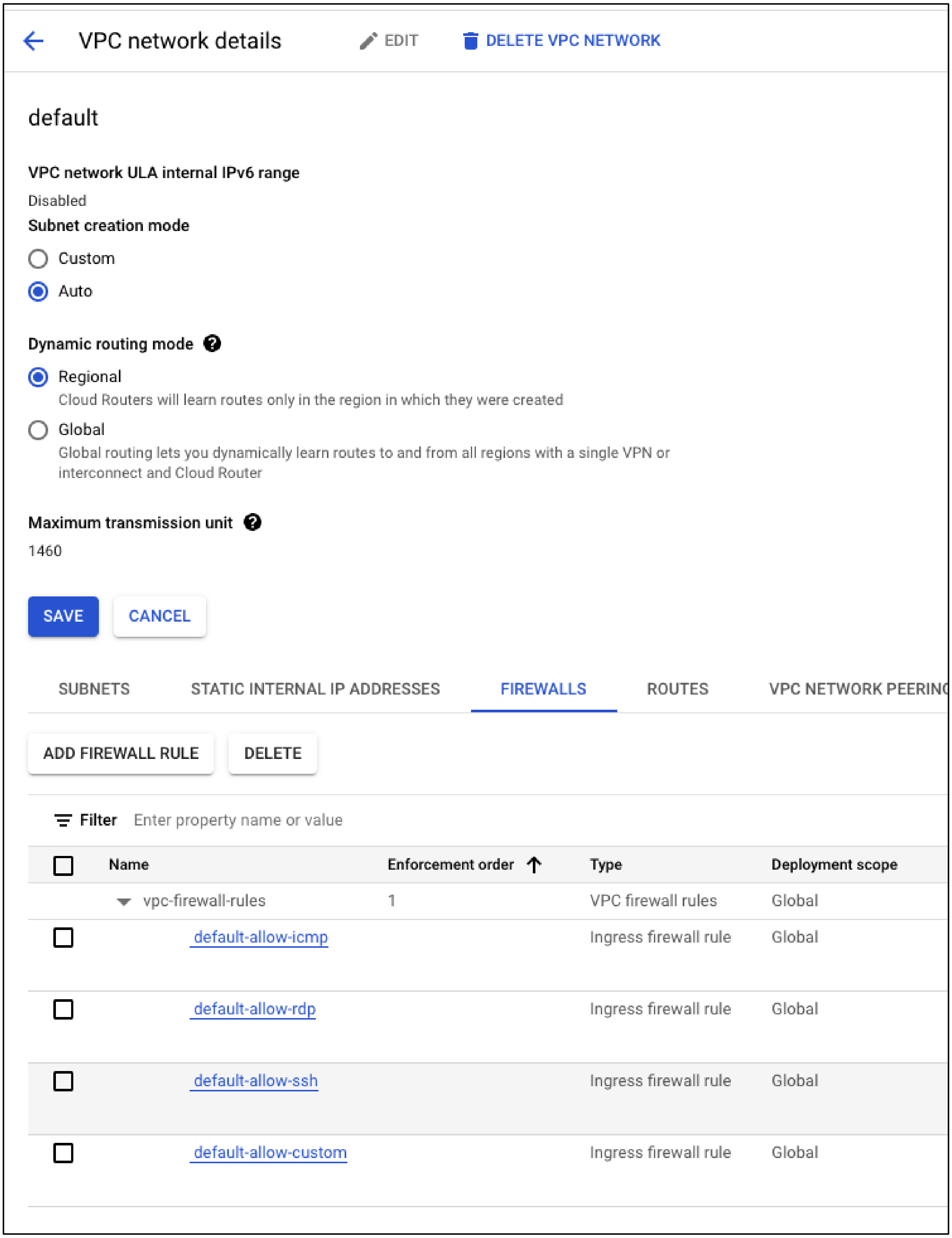
```
adcama@cloudshell:~ (adcama-movie-recommender)$ curl https://files.grouplens.o
rg/datasets/movielens/ml-1m.zip -o movielens1m.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
t
                                 Dload  Upload   Total   Spent    Left  Speed
100 5778k  100 5778k    0     0  16.5M      0 --:--:-- --:--:-- --:--:-- 16.5M
adcama@cloudshell:~ (adcama-movie-recommender)$ ls
movielens1m.zip  README-cloudshell.txt
adcama@cloudshell:~ (adcama-movie-recommender)$ unzip movielens1m.zip
Archive:  movielens1m.zip
   creating: ml-1m/
  inflating: ml-1m/movies.dat
  inflating: ml-1m/ratings.dat
  inflating: ml-1m/README
  inflating: ml-1m/users.dat
adcama@cloudshell:~ (adcama-movie-recommender)$ ls
ml-1m  movielens1m.zip  README-cloudshell.txt
adcama@cloudshell:~ (adcama-movie-recommender)$ cd ml-1m/
adcama@cloudshell:~/ml-1m (adcama-movie-recommender)$ ls
movies.dat  ratings.dat  README  users.dat
```

As seen in the image below we then proceed to move the 3 files to the bucket we created in step 2.

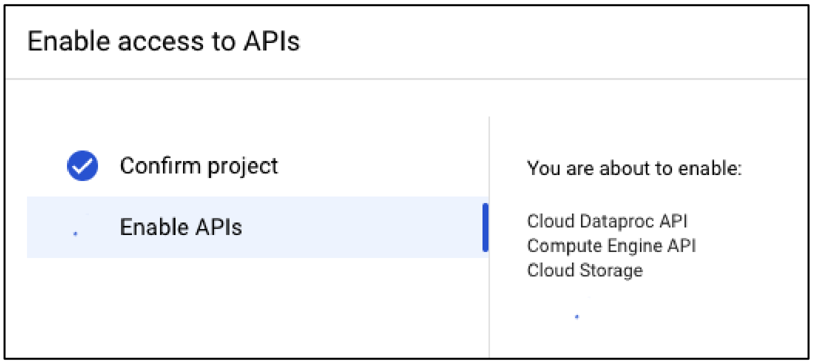
```
adcama@cloudshell:~/ml-1m (adcama-movie-recommender)$ gsutil cp -r movies.dat gs://adcama-movie-recommender-
data/data

Copying file://movies.dat [Content-Type=application/octet-stream]...
/ [1 files][167.3 KiB/167.3 KiB]
Operation completed over 1 objects/167.3 KiB.
```

Continued on next Page …….

4) The next and very important step is to create a VPC default network (which consists of subnets, ingress / egress rules), this network will also enable you to interact with Compute engine instances (Our dataproc cluster also runs on a compute Engine instance), other services in google cloud and is essentially the backbone of your cloud platform. Given below is the VPC config and firewalls rules I enabled, the subnet creation mode should be automatic.

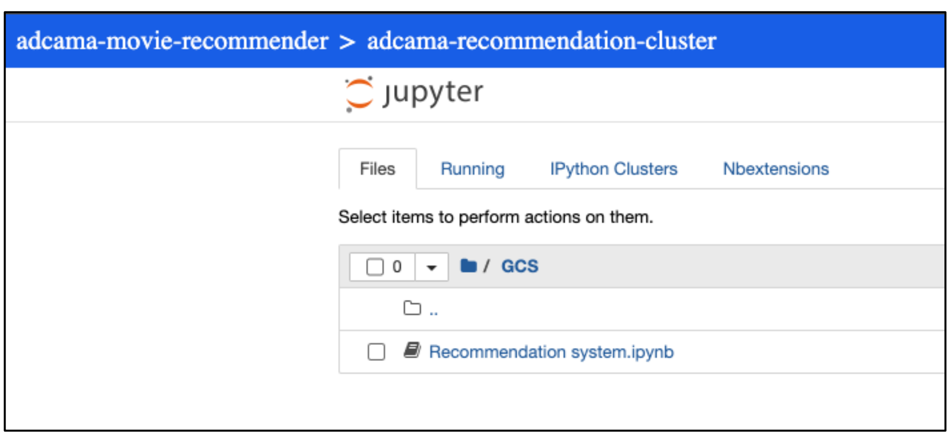

VPC network details
EDIT
DELETE VPC NETWORK
default
VPC network ULA internal IPv6 range
Disabled
Subnet creation mode
Custom
Auto
Dynamic routing mode
Regional
Cloud Routers will learn routes only in the region in which they were created
Global
Global routing lets you dynamically learn routes to and from all regions with a single VPN or interconnect and Cloud Router
Maximum transmission unit
1460
SAVE
CANCEL
SUBNETS
STATIC INTERNAL IP ADDRESSES
FIREWALLS
ROUTES
VPC NETWORK PEERING
ADD FIREWALL RULE
DELETE
Filter Enter property name or value
Name
Enforcement order
Type
Deployment scope
vpc-firewall-rules
1
VPC firewall rules
Global
default-allow-icmp
Ingress firewall rule
Global
default-allow-rdp
Ingress firewall rule
Global
default-allow-ssh
Ingress firewall rule
Global
default-allow-custom
Ingress firewall rule
Global

5) Create a Dataproc cluster on which we will run our spark jobs, for this we will have to enable the following APIs. Given below is the command a command to create a dataproc cluster and you will use something similar to create the cluster.

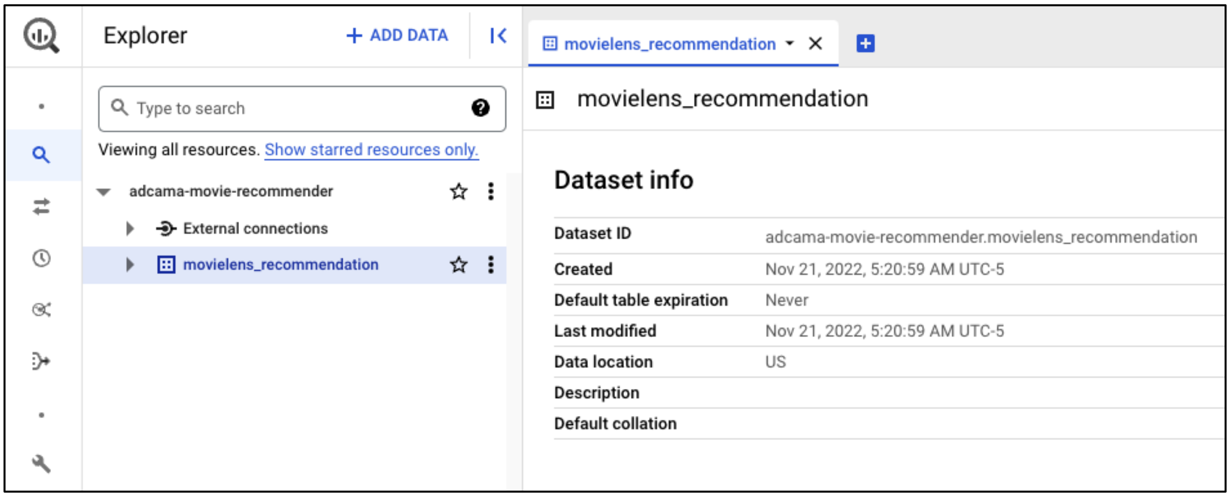



```
gcloud dataproc clusters create adcama-rec-cluster --region us-central1 --zone us-
central1-a --master-machine-type n1-standard-4 --master-boot-disk-size 500 --num-
workers 2 --worker-machine-type n1-standard-4 --worker-boot-disk-size 500 --image-
version 2.0-debian10 --optional-components JUPYTER --project adcama-movie-recommender
```

6) Navigate to the cluster instance (click on cluster name) and then click Web Interfaces > Jupyter and you will be greeted with a familiar Jupyter interface where you can run your code for analysis and visualizations.

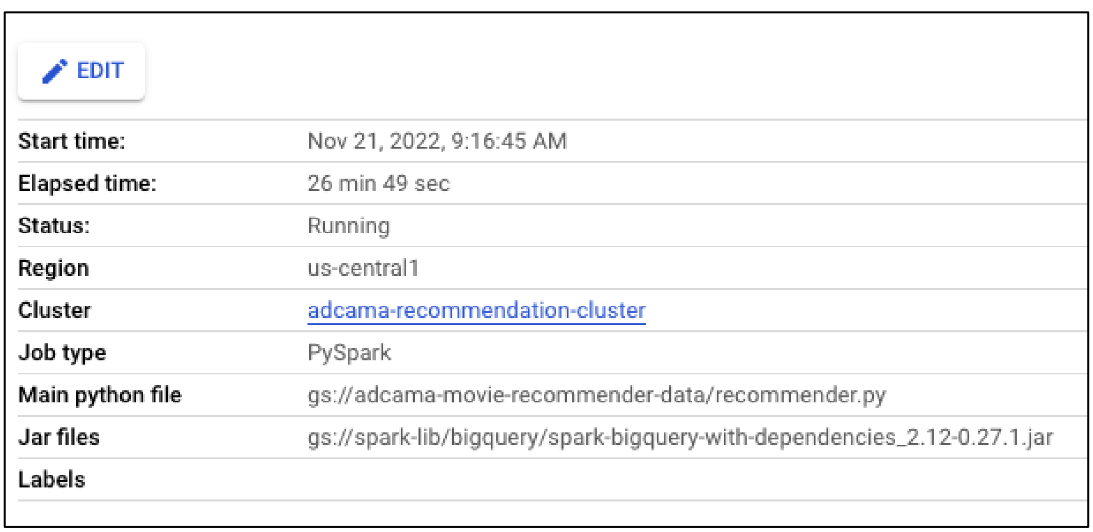


7) Our dataset consistes of 3 main .dat files, namely users, ratings and movies and so we have to create a Bigquery dataset where we have to store these tables (in addition to the recommendation table generated by our ML model), so we create a new dataset in Bigquery.



8) Create a python file that you will be using as the driver program to run the project and store it in your storage bucket. Use the file as a part of your parameters to submit a spark job. (Python file provided in the appendix)

Note the presence of a jar file, this file is a part of GCPs Spark-Bigquery connectors and is necessary for us to move the data ie. Our recommendations and dataset into Bigquery database.



9) The spark job generates the top 3 recommendations for each user and moves all the tables to bigquery (code will be discussed later) here is how the job output looks.

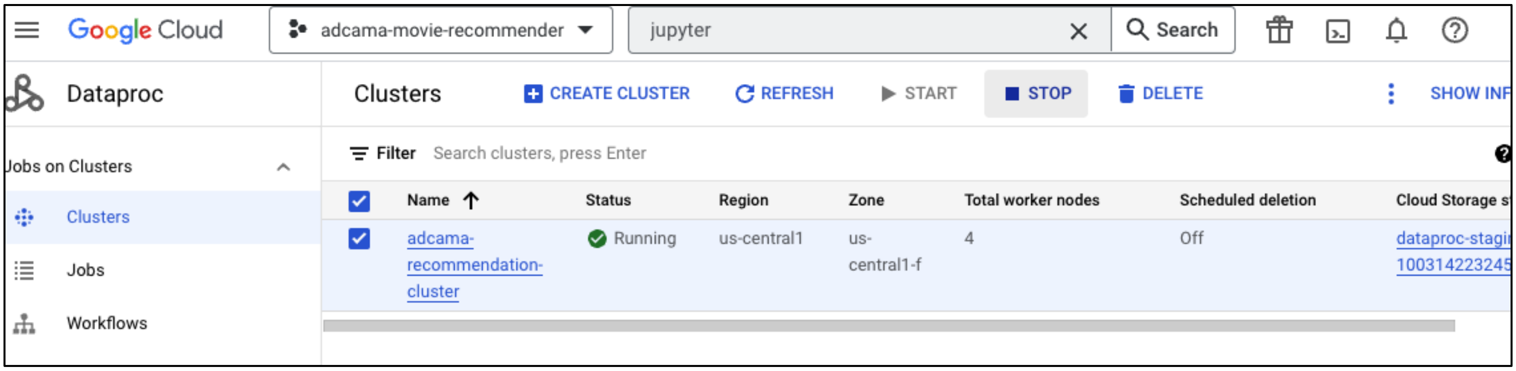
Job ID job-a05b7714

Output LINE WRAP: OFF

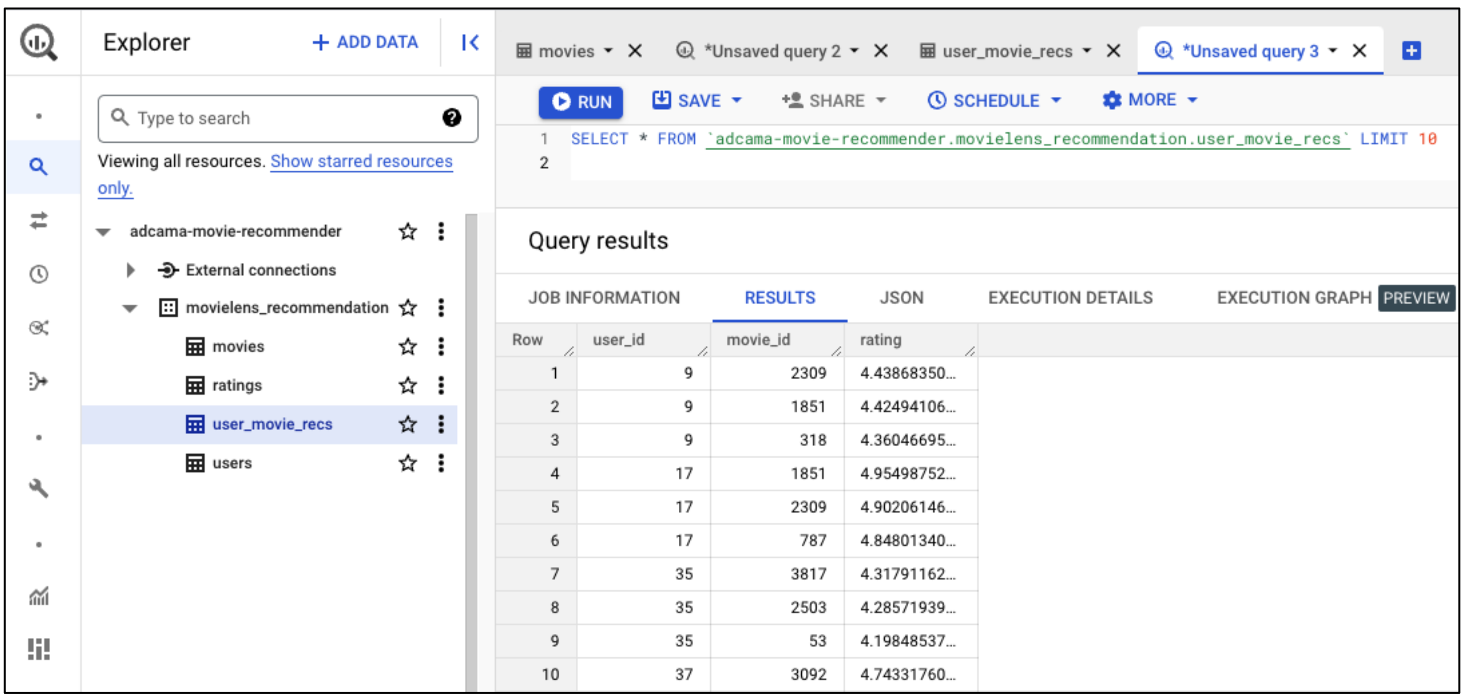
Spark jobs take ~60 seconds to initialize resources. DISM

```
22/11/21 14:16:52 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at adcama-recommendation-cluster-
22/11/21 14:16:52 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at adcama-recommendat
22/11/21 14:16:53 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
22/11/21 14:16:53 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
22/11/21 14:16:54 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1669037765104
22/11/21 14:16:55 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at adcama-recommendation-cluster-
22/11/21 14:16:57 INFO com.google.cloud.hadoop.repackaged.gcs.com.google.cloud.hadoop.gcsio.GoogleCloudStorageImpl: Ignoring
Total number of models to be tested:  18
Time taken for training 1819.3s
**Best Model**
  Rank: 150
  MaxIter: 5
  RegParam: 0.1
Root Mean Squared Error is 0.8828626206986543
+-------+-----------------------------------------------------------+
|user_id|recommendations                                            |
+-------+-----------------------------------------------------------+
|31     |[{1198, 4.3207555}, {3092, 4.293794}, {260, 4.264002}]     |
|34     |[{1851, 4.8325996}, {318, 4.7528214}, {1780, 4.595234}]    |
|53     |[{2309, 5.355816}, {53, 5.3002214}, {787, 5.290479}]       |
|65     |[{1741, 4.9415336}, {37, 4.929976}, {3147, 4.90405}]       |
|78     |[{1851, 4.402133}, {3817, 4.3339553}, {3092, 4.309911}]    |
|81     |[{1743, 4.9537096}, {1178, 4.8686466}, {858, 4.8638787}]   |
|85     |[{1851, 3.658333}, {1780, 3.6528223}, {2810, 3.6104143}]   |
|101    |[{3314, 5.570479}, {1780, 5.56808}, {1741, 5.5022225}]     |
|108    |[{1420, 4.235145}, {1743, 4.146439}, {2905, 4.1154037}]    |
|115    |[{2571, 4.7924914}, {1420, 4.7680593}, {1851, 4.750271}]   |
+-------+-----------------------------------------------------------+
only showing top 10 rows

Moving tables to bigquery ....
22/11/21 14:48:22 INFO com.google.cloud.spark.bigquery.write.context.BigQueryDirectDataSourceWriterContext: BigQuery DataSour
[BigQueryWriterCommitMessage{partitionId=0, taskId=0, epochId=1669042082770, tableId='projects/adcama-movie-recommender/datas
22/11/21 14:48:23 INFO com.google.cloud.spark.bigquery.write.context.BigQueryDirectDataSourceWriterContext: BigQuery DataSour
nanos: 487551000

22/11/21 14:48:27 INFO com.google.cloud.spark.bigquery.write.context.BigQueryDirectDataSourceWriterContext: BigQuery DataSour
[BigQueryWriterCommitMessage{partitionId=0, taskId=0, epochId=1669042104048, tableId='projects/adcama-movie-recommender/datas
22/11/21 14:48:27 INFO com.google.cloud.spark.bigquery.write.context.BigQueryDirectDataSourceWriterContext: BigQuery DataSour
nanos: 294431000

22/11/21 14:48:29 INFO com.google.cloud.spark.bigquery.write.context.BigQueryDirectDataSourceWriterContext: BigQuery DataSour
[BigQueryWriterCommitMessage{partitionId=0, taskId=0, epochId=1669042107786, tableId='projects/adcama-movie-recommender/datas
22/11/21 14:48:29 INFO com.google.cloud.spark.bigquery.write.context.BigQueryDirectDataSourceWriterContext: BigQuery DataSour
nanos: 566063000

22/11/21 14:48:33 INFO com.google.cloud.spark.bigquery.write.context.BigQueryDirectDataSourceWriterContext: BigQuery DataSour
[BigQueryWriterCommitMessage{partitionId=0, taskId=0, epochId=1669042110067, tableId='projects/adcama-movie-recommender/datas
22/11/21 14:48:33 INFO com.google.cloud.spark.bigquery.write.context.BigQueryDirectDataSourceWriterContext: BigQuery DataSour
nanos: 282463000

22/11/21 14:48:33 INFO org.sparkproject.jetty.server.AbstractConnector: Stopped Spark@5930504c{HTTP/1.1, (http/1.1)}{0.0.0.0:
```

10) Remember to make sure that you stop your spark cluster once the job has been executed in order to prevent unnecessary billing charges to your GCP account.



11) The next step would be to explore and the make use of the data in Bigquery tables.



We will now briefly display the code that we used to run the spark job and what it does –

```
File: /Users/aditcam/Desktop/MGMT-ACC-BIGDATA/recommender.py
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row, SparkSession
```

```
from pyspark.sql.functions import explode
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.sql.functions import col
import time

spark = SparkSession.builder.appName('Spark_ALS_recommender').getOrCreate()

user_col = ['user_id', 'gender', 'age', 'occupation', 'zip']
user_df = spark.read.format("csv").option("delimiter", "::").option("inferSchema", "true").load("gs://adcama-movie-recommender-data/data/users.dat")
col_alias = user_df.columns
user_df = user_df.select([col(col_alias[i]).alias(user_col[i]) for i in range(len(user_col))])

rating_col = ['user_id', 'movie_id', 'rating', 'timestamp']
rating_df = spark.read.format("csv").option("delimiter", "::").option("inferSchema", "true").load("gs://adcama-movie-recommender-data/data/ratings.dat")
rating_df = rating_df.select([col(col_alias[i]).alias(rating_col[i]) for i in range(len(rating_col))])

movie_col = ['movie_id', 'title', 'genres']
movie_df = spark.read.format("csv").option("delimiter", "::").option("inferSchema", "true").load("gs://adcama-movie-recommender-data/data/movies.dat")
movie_df = movie_df.select([col(col_alias[i]).alias(movie_col[i]) for i in range(len(movie_col))])

(train, test) = rating_df.randomSplit([0.8, 0.2])

als = ALS(userCol="user_id", itemCol="movie_id", ratingCol="rating",coldStartStrategy="drop",nonnegative=True,implicitPrefs=False)

# Add hyperparameters and their respective values to param_grid
param_grid = ParamGridBuilder() \
            .addGrid(als.rank, [10,100,150]) \
            .addGrid(als.regParam, [.1, .15,.3]) \
            .addGrid(als.maxIter, [1, 5]) \
            .build()

# Define evaluator as RMSE and print length of evaluator
evaluator = RegressionEvaluator(
```

```
            metricName="rmse",
            labelCol="rating",
            predictionCol="prediction")
print ("Total number of models to be tested: ", len(param_grid))

# Build cross validation using CrossValidator
start = time.time()
cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=5)
cvModel = cv.fit(train)
best_model = cvModel.bestModel
end = time.time()
print('Time taken for training {}s'.format(round(end-start,2)))

print("**Best Model**")
# Print "Rank"
print("  Rank:", best_model._java_obj.parent().getRank())
# Print "MaxIter"
print("  MaxIter:", best_model._java_obj.parent().getMaxIter())
# Print "RegParam"
print("  RegParam:", best_model._java_obj.parent().getRegParam())

# View the predictions
test_predictions = best_model.transform(test)
RMSE = evaluator.evaluate(test_predictions)
print('Root Mean Squared Error is {}'.format(RMSE))

movieSubSetRecs = best_model.recommendForAllUsers(3)

movieSubSetRecs.show(10,False)

final_df = movieSubSetRecs \
.withColumn('recommendation',explode(movieSubSetRecs.recommendations)) \
.select(movieSubSetRecs.user_id,col('recommendation.movie_id'),col('recommendation.rating'))

print('Moving tables to bigquery .... ')

final_df.write.format('bigquery') \
```

```
74:   .option("writeMethod", "direct") \
75:   .mode("overwrite") \
76:   .save('movielens_recommendation.user_movie_recs')
77:
78: user_df.write.format('bigquery') \
79:   .option("writeMethod", "direct") \
80:   .mode("overwrite") \
81:   .save('movielens_recommendation.users')
82:
83: movie_df.write.format('bigquery') \
84:   .option("writeMethod", "direct") \
85:   .mode("overwrite") \
86:   .save('movielens_recommendation.movies')
87:
88: rating_df.write.format('bigquery') \
89:   .option("writeMethod", "direct") \
90:   .mode("overwrite") \
91:   .save('movielens_recommendation.ratings')
92:
93:
```

- In the code above, lines 1-7 are essentially import statements for all the libraries that are necessary.
- Line 9 creates a spark session / context.
- Line 11 – 22 basically gets the data from GCS storage bucket and converts it to a spark dataframe.
- In line 24 we perform a train test split on our ratings dataframe, preparing it for our ML model.
- Line 26 model creation.
- Line 29 – 40 building parameter grid, evaluator metric for 5 fold crossfold validation.
- Line 42 – 61 perform cross fold validation, obtain the best model parameters and then perform predictions, evaluate rmse.

- Line 63 recommend top 3 movie recommendationds for all users.
- Line 65 showing a sample of the output data.
- Line 67 – 91 moving the data to Big query dataset.

IV. RESULTS

First we will display the results obtained from the EDA analysis and note our observations.

1) In the image below we see summary statistics of the user, ratings and movie tables of the dataset.
   - There are 6040 users and their average age is 30 with a min of 1 (which is odd) and max of 56.
   - There are 1000209 ratings with an average rating of 3.58.
   - There are 3883 movies.
   - The predicted table consists of 6040 * 3 = 18120 records (3 movie rec / user). The mean rating is 5.26 but the max is 5, this is because the predictions have a RMSE of 0.81 thus accounting for the inaccuracy.

Summary Stats

In [10]:
```
user_df.describe().show()
```

summary	user_id	gender	age	occupation	zip
count	6040	6040	6040	6040	6040
mean	3020.5	null	30.639238410596025	8.146854304635761	87986.22464010713
stddev	1743.7421445462246	null	12.895961726906837	6.329511491401687	2499493.295731326
min	1	F	1	0	00231
max	6040	M	56	20	99945

In [11]:
```
rating_df.describe().show()
```

summary	user_id	movie_id	rating	timestamp
count	1000209	1000209	1000209	1000209
mean	3024.512347919285	1865.5398981612843	3.581564453029317	9.722436954046655E8
stddev	1728.4126948999715	1096.0406894572482	1.1171018453732606	1.2152558939916052E7
min	1	1	1	956703932
max	6040	3952	5	1046454590

In [12]:
```
movie_df.describe().show()
```

summary	movie_id	title	genres
count	3883	3883	3883
mean	1986.0494463044038	null	null
stddev	1146.7783494728876	null	null
min	1	$1,000,000 Duck (...	Action
max	3952	eXistenZ (1999)	Western

In [13]:
```
final_df.describe().show()
```

summary	user_id	movie_id	rating
count	18120	18120	18120
mean	3020.5	1773.130518763797	5.268932872767217
stddev	1743.6459035689982	1338.7664796694971	0.8100983781211466
min	1	37	1.3400229
max	6040	3542	8.630114

2) In the next image we see the most frequently rated movies

Most rated movies

In [15]:
```
most_rated = rating_df \
.groupBy("movie_id") \
.agg(count("user_id")) \
.withColumnRenamed("count(user_id)", "rating_cnt") \
.sort(desc("rating_cnt"))
```

In [16]:
```
most_rated.show(10,False)
```

```
+--------+----------+
|movie_id|rating_cnt|
+--------+----------+
|2858    |3428      |
|260     |2991      |
|1196    |2990      |
|1210    |2883      |
|480     |2672      |
|2028    |2653      |
|589     |2649      |
|2571    |2590      |
|1270    |2583      |
|593     |2578      |
+--------+----------+
only showing top 10 rows
```

In [17]:
```
most_rated_movies =  most_rated.join(movie_df,most_rated.movie_id == movie_df.movie_id)
```

In [18]:
```
most_rated_movies.show(25,False)
```

```
+--------+----------+--------+-------------------------------------------+--------------------------------+
|movie_id|rating_cnt|movie_id|title                                      |genres                          |
+--------+----------+--------+-------------------------------------------+--------------------------------+
|1580    |2538      |1580    |Men in Black (1997)                        |Action|Adventure|Comedy|Sci-Fi   |
|2366    |756       |2366    |King Kong (1933)                           |Action|Adventure|Horror         |
|1088    |687       |1088    |Dirty Dancing (1987)                       |Musical|Romance                 |
|1959    |626       |1959    |Out of Africa (1985)                       |Drama|Romance                   |
|3175    |1728      |3175    |Galaxy Quest (1999)                        |Adventure|Comedy|Sci-Fi         |
|1645    |826       |1645    |Devil's Advocate, The (1997)               |Crime|Horror|Mystery|Thriller   |
|496     |37        |496     |What Happened Was... (1994)                |Comedy|Drama|Romance            |
|2142    |201       |2142    |American Tail: Fievel Goes West, An (1991)|Animation|Children's|Comedy     |
|1591    |475       |1591    |Spawn (1997)                               |Action|Adventure|Sci-Fi|Thriller|
|2122    |233       |2122    |Children of the Corn (1984)                |Horror|Thriller                 |
|833     |78        |833     |High School High (1996)                    |Comedy                          |
|463     |47        |463     |Guilty as Sin (1993)                       |Crime|Drama|Thriller            |
|471     |599       |471     |Hudsucker Proxy, The (1994)                |Comedy|Romance                  |
|1342    |262       |1342    |Candyman (1992)                            |Horror                          |
|148     |23        |148     |Awfully Big Adventure, An (1995)           |Drama                           |
|3918    |167       |3918    |Hellbound: Hellraiser II (1988)            |Horror                          |
|3794    |121       |3794    |Chuck & Buck (2000)                        |Comedy|Drama                    |
|1238    |351       |1238    |Local Hero (1983)                          |Comedy                          |
|2866    |199       |2866    |Buddy Holly Story, The (1978)              |Drama                           |
|3749    |22        |3749    |Time Regained (Le Temps Retrouv�) (1999)   |Drama                           |
|2659    |46        |2659    |It Came from Hollywood (1982)              |Comedy|Documentary              |
|1829    |37        |1829    |Chinese Box (1997)                         |Drama|Romance                   |
|1721    |1546      |1721    |Titanic (1997)                             |Drama|Romance                   |
|1084    |686       |1084    |Bonnie and Clyde (1967)                    |Crime|Drama                     |
|1127    |1715      |1127    |Abyss, The (1989)                          |Action|Adventure|Sci-Fi|Thriller|
+--------+----------+--------+-------------------------------------------+--------------------------------+
only showing top 25 rows
```

3) Below we see the Highest Rated movies (not consider the count of ratings)

Highest rated movies

In [19]:
```
high_rated = rating_df \
.groupBy("movie_id") \
.agg(avg(col("rating")),count(col('movie_id'))) \
.withColumnRenamed("avg(rating)", "avg_rating") \
.withColumnRenamed("count(movie_id)", "rating_cnt") \
.sort(desc("avg_rating"),desc("rating_cnt"))
```

In [20]:
```
high_rated.show(10,False)
```

```
+--------+----------+----------+
|movie_id|avg_rating|rating_cnt|
+--------+----------+----------+
|787     |5.0       |3         |
|3233    |5.0       |2         |
|3280    |5.0       |1         |
|3881    |5.0       |1         |
|3607    |5.0       |1         |
|989     |5.0       |1         |
|3172    |5.0       |1         |
|3382    |5.0       |1         |
|3656    |5.0       |1         |
|1830    |5.0       |1         |
+--------+----------+----------+
only showing top 10 rows
```

In [21]:
```
high_rated.join(movie_df,high_rated.movie_id == movie_df.movie_id).sort(desc("avg_rating"),desc("rating_cnt")).show
```

```
+--------+----------+----------+--------+----------------------------------------+--------------------+
|movie_id|avg_rating|rating_cnt|movie_id|title                                   |genres              |
+--------+----------+----------+--------+----------------------------------------+--------------------+
|787     |5.0       |3         |787     |Gate of Heavenly Peace, The (1995)      |Documentary         |
|3233    |5.0       |2         |3233    |Smashing Time (1967)                    |Comedy              |
|3280    |5.0       |1         |3280    |Baby, The (1973)                        |Horror              |
|3881    |5.0       |1         |3881    |Bittersweet Motel (2000)                |Documentary         |
|3607    |5.0       |1         |3607    |One Little Indian (1973)                |Comedy|Drama|Western|
|989     |5.0       |1         |989     |Schlafes Bruder (Brother of Sleep) (1995)|Drama              |
|3172    |5.0       |1         |3172    |Ulysses (Ulisse) (1954)                 |Adventure           |
|3382    |5.0       |1         |3382    |Song of Freedom (1936)                  |Drama               |
|3656    |5.0       |1         |3656    |Lured (1947)                            |Crime               |
|1830    |5.0       |1         |1830    |Follow the Bitch (1998)                 |Comedy              |
+--------+----------+----------+--------+----------------------------------------+--------------------+
only showing top 10 rows
```

4) Below we see the highest rated movies with highest count

In [22]:

```
high_rated = rating_df \
.groupBy("movie_id") \
.agg(avg(col("rating")),count(col('movie_id'))) \
.withColumnRenamed("avg(rating)", "avg_rating") \
.withColumnRenamed("count(movie_id)", "rating_cnt") \
.sort(desc('rating_cnt'),desc("avg_rating"))
```

In [23]:

```
high_rated.join(movie_df,most_rated.movie_id == movie_df.movie_id).sort(desc('rating_cnt'),desc("avg_rating")).show
```

```
+--------+------------------+----------+--------+-----------------------------------------------------+--------------
---------------------+
|movie_id|avg_rating        |rating_cnt|movie_id|title                                                |genres
|
+--------+------------------+----------+--------+-----------------------------------------------------+--------------
---------------------+
|2858    |4.3173862310385065|3428      |2858    |American Beauty (1999)                               |Comedy|Drama
|
|260     |4.453694416583082 |2991      |260     |Star Wars: Episode IV - A New Hope (1977)            |Action|Adventu
re|Fantasy|Sci-Fi    |
|1196    |4.292976588628763 |2990      |1196    |Star Wars: Episode V - The Empire Strikes Back (1980)|Action|Adventu
re|Drama|Sci-Fi|War  |
|1210    |4.022892819979188 |2883      |1210    |Star Wars: Episode VI - Return of the Jedi (1983)    |Action|Adventu
re|Romance|Sci-Fi|War|
|480     |3.7638473053892216|2672      |480     |Jurassic Park (1993)                                 |Action|Adventu
re|Sci-Fi            |
|2028    |4.337353938937053 |2653      |2028    |Saving Private Ryan (1998)                           |Action|Drama|W
ar                   |
|589     |4.058512646281616 |2649      |589     |Terminator 2: Judgment Day (1991)                    |Action|Sci-Fi|
Thriller             |
|2571    |4.315830115830116 |2590      |2571    |Matrix, The (1999)                                   |Action|Sci-Fi|
Thriller             |
|1270    |3.9903213317847466|2583      |1270    |Back to the Future (1985)                            |Comedy|Sci-Fi
|
|593     |4.3518231186966645|2578      |593     |Silence of the Lambs, The (1991)                     |Drama|Thriller
|
+--------+------------------+----------+--------+-----------------------------------------------------+--------------
---------------------+
only showing top 10 rows
```

5) Below we see movies with highly polarized reviews.

Movies with highly conflicting/polarising reviews

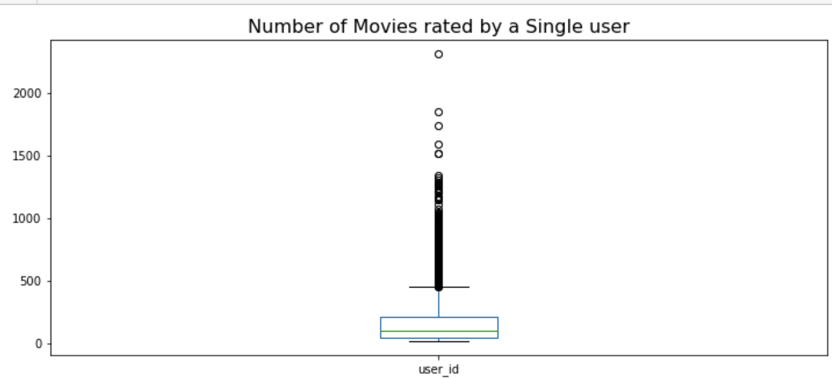
In [24]:
```
ratings_sd = rating_df\
.groupBy("movie_id")\
.agg(count("user_id").alias("rating_cnt"),
     avg(col("rating")).alias("avg_rating"),
     stddev(col("rating")).alias("sd_rating")
    )\
.where("rating_cnt > 50")
```

In [25]:
```
ratings_sd.join(movie_df,ratings_sd.movie_id == movie_df.movie_id).sort(desc("sd_rating")).show(10,False)
```

```
+--------+----------+------------------+------------------+--------+----------------------------------------------+--
---------------------+
|movie_id|rating_cnt|avg_rating        |sd_rating         |movie_id|title                                         |ge
nres                 |
+--------+----------+------------------+------------------+--------+----------------------------------------------+--
---------------------+
|1241    |70        |3.357142857142857 |1.4649104863162303|1241    |Braindead (1992)                              |Co
medy|Horror          |
|1924    |249       |2.6345381526104417|1.4559983991255796|1924    |Plan 9 from Outer Space (1958)                |Ho
rror|Sci-Fi          |
|2507    |54        |2.9814814814814814|1.4074028099524012|2507    |Breakfast of Champions (1999)                 |Co
medy                 |
|2275    |71        |3.3661971830985915|1.4065097289767452|2275    |Six-String Samurai (1998)                     |Ac
tion|Adventure|Sci-Fi|
|2362    |53        |2.849056603773585 |1.3922343195671307|2362    |Glen or Glenda (1953)                         |Dr
ama                  |
|3718    |61        |3.3114754098360657|1.3728435648506763|3718    |American Pimp (1999)                          |Do
cumentary            |
|2314    |104       |3.1346153846153846|1.3728129459672882|2314    |Beloved (1998)                                |Dr
ama                  |
|3864    |143       |2.6923076923076925|1.3646996740797475|3864    |Godzilla 2000 (Gojira ni-sen mireniamu) (1999)|Ac
tion|Adventure|Sci-Fi|
|3340    |71        |2.8028169014084505|1.3587780046777236|3340    |Bride of the Monster (1956)                   |Ho
rror|Sci-Fi          |
|2459    |247       |3.222672064777328 |1.3324484407683033|2459    |Texas Chainsaw Massacre, The (1974)           |Ho
rror                 |
+--------+----------+------------------+------------------+--------+----------------------------------------------+--
---------------------+
only showing top 10 rows
```

6) A boxplot of the number of ratings given by users.

In [32]:
```
pd_ratings_df.user_id.value_counts().plot.box(figsize=(12, 5))
plt.title("Number of Movies rated by a Single user", fontsize=16)
plt.show()
```

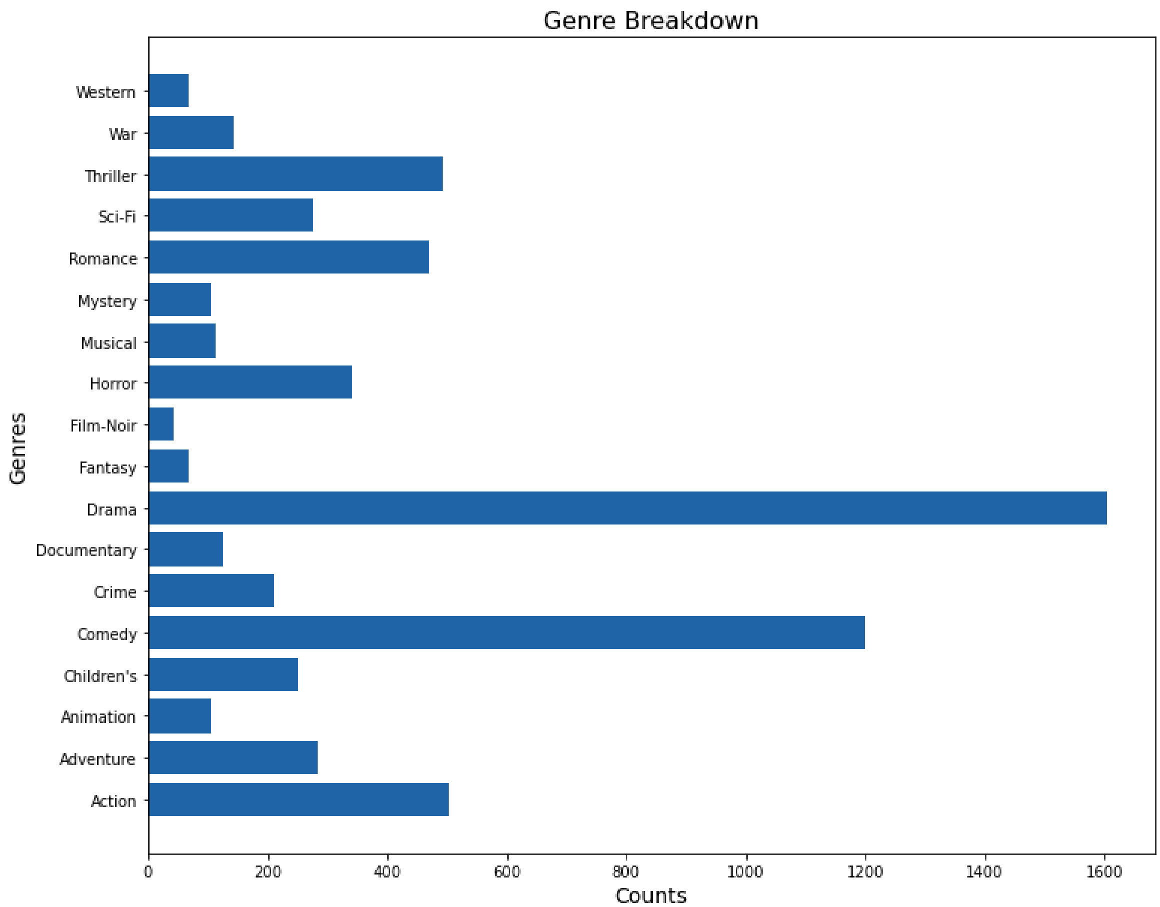



```
Max user movie ratings 2314
Min user ratings 20
```

7) Genre breakdown across dataset.

Most Common genre in movies

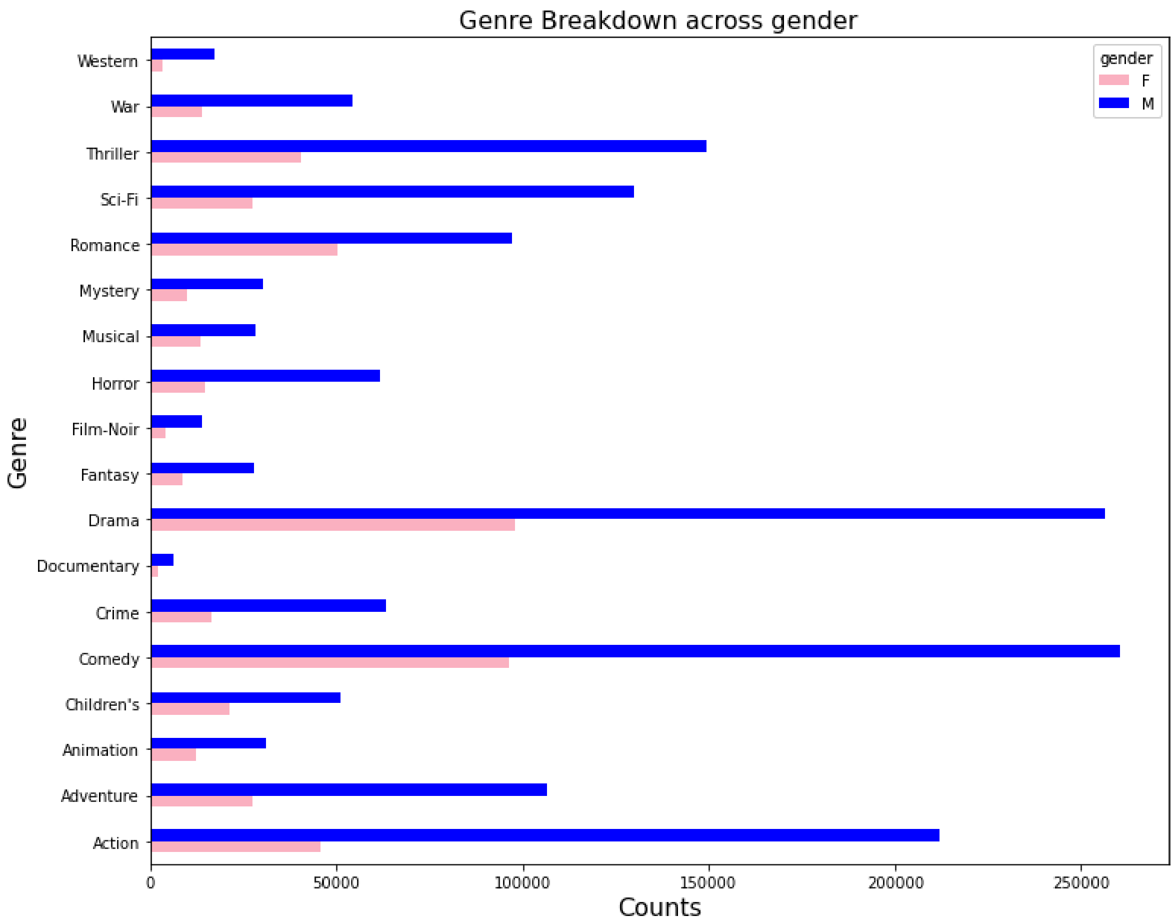
In [37]:
```
# get the genre names in the dataframe and their counts
temp = pd_movies_df[list(genres)]
label= temp.sum().index
label_counts= temp.sum().values
# plot a bar chart
plt.figure(figsize=(12, 10))
plt.barh(y= label, width= label_counts)
plt.title("Genre Breakdown", fontsize=16)
plt.ylabel("Genres", fontsize=14)
plt.xlabel("Counts", fontsize=14)
plt.show()
```



8) Genre breakdown across dataset with respect to genders.

Genre Breakdown across genders

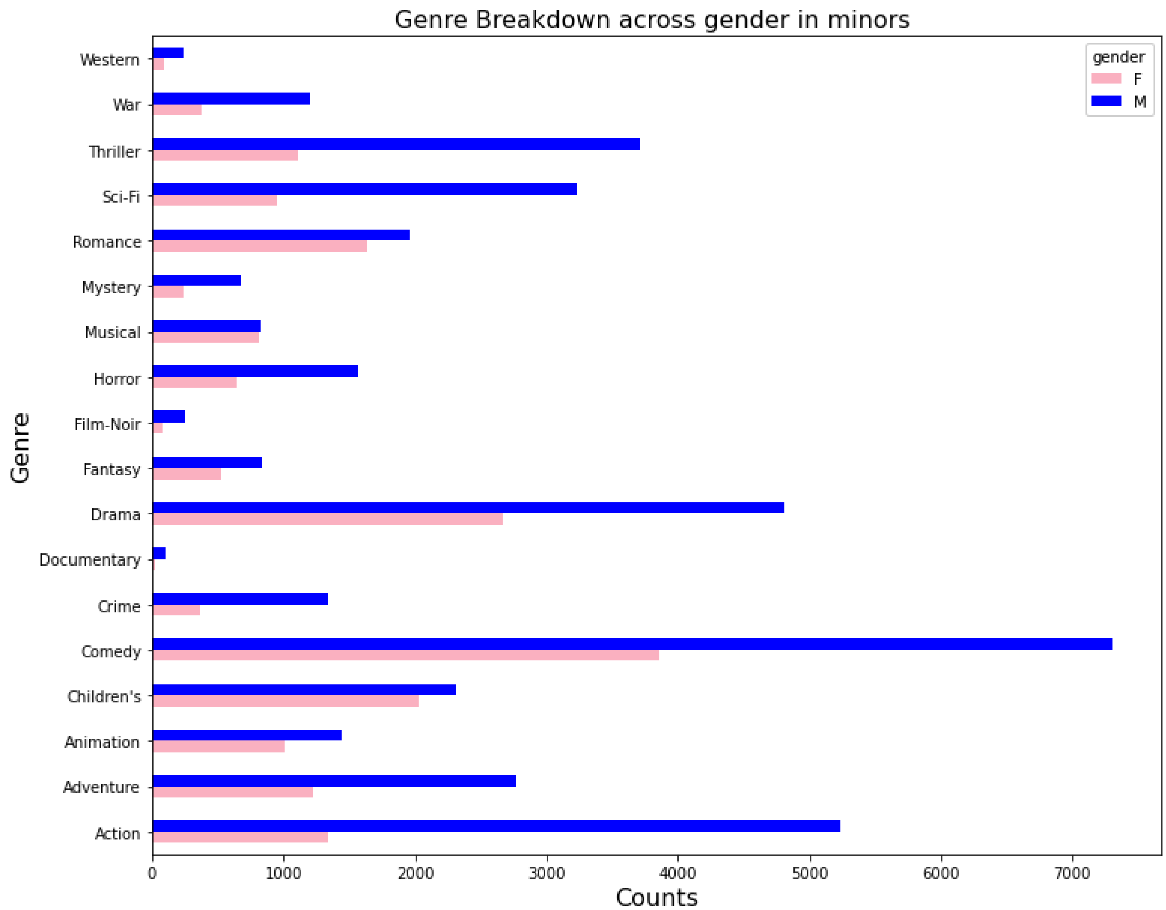
In [38]:
```
df_merged[list(genres)+["gender"]].groupby("gender").sum().T.plot(kind="barh", figsize=(12,10), color=["pink", "
plt.xlabel("Counts",fontsize=16)
plt.ylabel("Genre", fontsize=16)
plt.title("Genre Breakdown across gender", fontsize=16)
plt.show()
```



9) Genre breakdown across minors in the dataset with respect to genders.

Genre Breakdown across genders within children (i.e. age less than 18)

In [39]:
```
df_merged[df_merged['age']<18][list(genres)+["gender"]].groupby("gender").sum().T.plot(kind="barh", figsize=(12,1
plt.xlabel("Counts",fontsize=16)
plt.ylabel("Genre", fontsize=16)
plt.title("Genre Breakdown across gender in minors ", fontsize=16)
plt.show()
```



Finally, we look at the results of our predictions –

In the image below we see a sample of the predictions our program makes, it consists of the user for whom the prediction is, the respective movie id and the predicted rating for that user.

Movie recommendations

In [4]:
```
final_df = movieSubSetRecs \
.withColumn('recommendation',explode(movieSubSetRecs.recommendations)) \
.select(movieSubSetRecs.user_id,col('recommendation.movie_id'),col('recommendation.rating'))
```

In [5]:
```
final_df.show(10,False)
```

```
+-------+--------+---------+
|user_id|movie_id|rating   |
+-------+--------+---------+
|12     |3382    |5.854477 |
|12     |557     |4.7550344|
|12     |572     |4.5924554|
|26     |3382    |5.189043 |
|26     |572     |4.3411207|
|26     |1851    |3.9044237|
|27     |3382    |6.050927 |
|27     |557     |5.383823 |
|27     |3542    |5.0529943|
|28     |3382    |5.7740617|
+-------+--------+---------+
only showing top 10 rows
```

Let us now verify that our recommendations make sense. Consider a user with user_id = 790 let us see the predictions our program made for him (Using Bigquery) –

```
select u.user_id,u.gender,u.age,mr.rating,m.title,m.genres from
`adcama-movie-recommender.movielens_recommendation.users` as u
join
`adcama-movie-recommender.movielens_recommendation.user_movie_recs` as mr
on u.user_id = mr.user_id
join
`adcama-movie-recommender.movielens_recommendation.movies` as m
on mr.movie_id = m.movie_id
where u.user_id = 790;
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	user_id	gender	age	rating	title	genres
1	790	M	25	4.91...	Raiders of the Lost Ark (1981)	Action|Adventure
2	790	M	25	4.86...	Big Trees, The (1952)	Action|Drama
3	790	M	25	4.85...	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Fantasy|Sci-Fi

So we can see that our program recommends Action / Adventure / Sci-Fi / Drama genre movies.

Now, looking at the ratings originally given by the user it seems that they are highly interested in Action/ Sci-Fi/ Drama / Action /Thriller movies which aligns with our recommendation system so it looks like our recommendations are working well.

```
select u.user_id,u.gender,u.age,r.rating,m.title,m.genres from
`adcama-movie-recommender.movielens_recommendation.users` as u
join
`adcama-movie-recommender.movielens_recommendation.ratings` as r
on u.user_id = r.user_id
join
`adcama-movie-recommender.movielens_recommendation.movies` as m
on r.movie_id = m.movie_id
where u.user_id = 790;
```

Query results

SAVE RES

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS | EXECUTION GRAPH PREVIEW

Row	user_id	gender	age	rating	title	genres
1	790	M	25	5	X-Men (2000)	Action|Sci-Fi
2	790	M	25	5	Terminator 2: Judgment Day (1...	Action|Sci-Fi|Thriller
3	790	M	25	5	Back to the Future (1985)	Comedy|Sci-Fi
4	790	M	25	3	Ghost in the Shell (Kokaku kido...	Animation|Sci-Fi
5	790	M	25	5	Final Destination (2000)	Drama|Thriller
6	790	M	25	5	Star Wars: Episode IV - A New ...	Action|Adventure|Fantasy|Sci-Fi
7	790	M	25	4	Shanghai Noon (2000)	Action
8	790	M	25	3	Mission: Impossible (1996)	Action|Adventure|Mystery
9	790	M	25	5	Star Wars: Episode V - The Em...	Action|Adventure|Drama|Sci-Fi|...
10	790	M	25	4	Rosemary's Baby (1968)	Horror|Thriller
11	790	M	25	5	Pitch Black (2000)	Action|Sci-Fi
12	790	M	25	2	Police Academy 5: Assignment...	Comedy
13	790	M	25	3	Supernova (2000)	Adventure|Sci-Fi
14	790	M	25	4	Mission to Mars (2000)	Sci-Fi
15	790	M	25	5	Rules of Engagement (2000)	Drama|Thriller
16	790	M	25	5	Matrix, The (1999)	Action|Sci-Fi|Thriller
17	790	M	25	3	Forrest Gump (1994)	Comedy|Romance|War
18	790	M	25	4	Mrs. Doubtfire (1993)	Comedy
19	790	M	25	5	Star Wars: Episode VI - Return ...	Action|Adventure|Romance|Sci...
20	790	M	25	5	Patriot, The (2000)	Action|Drama|War
21	790	M	25	5	Alien (1979)	Action|Horror|Sci-Fi|Thriller
22	790	M	25	5	Blade Runner (1982)	Film-Noir|Sci-Fi
23	790	M	25	5	Terminator, The (1984)	Action|Sci-Fi|Thriller

V. RESULTS

- **An interpretation of results**

1. While most of the summary statistics are ordinary, one the stood out the the minimum age of a person participating in rating of movies is 1, and there are many such records, these might be misreported ages/ incorrect/incomplete datapoints
2. In our predictions we see quite a few individuals whose predicted ratings that exceed 5, but this should ideally not be the case as the ratings are out of 5, what we need to keep in mind is that while training our ML model the best model we obtained after crossfold validation had an Root mean squared error value of 0.8 which means that the ML algorithm can be inaccurate at times but the rating doesn't need to be perfect in this case because we are trying to recommend the best movies for a user.
3. Looking at results number 3 and 4 we can infer that the movies that have the highest average rating are not necessarily the best because only 2-3 people have rated it 5, on the contrary a movie that has been rated 4.5 by thousands will most certainly end up being better so we should be very careful when looking at the average rating for a movie/product .etc.
4. In result 5 we see movies that have the most polarized reviews i.e. a very high standard deviation in their rating which means people either hate it or love it. It is interesting to see that most of these movies belong to Horror/Sci-Fi which can indeed be hit or miss depending on the type of person watching.
5. In result 6 the boxplot is surprising in that there are certain people who are outliers and have rated 2000+ movies.
6. From the genre breakdown we can conclude that Drama, Comedy Action, thriller and Romance are the most frequent movie genres in our dataset.
7. Contrasting results 8 and 9 of gender vs genre breakdown across a variety of ages it is clear that the dataset is skewed toward men therefore making it harder to draw any interesting inferences.

- **How I employed technologies / skills learned from this course**

In an effort to cover a majority of the topics learnt during the course of the semester and make this project a true culmination of all my learnings from the GCP Qwiklabs, study materials, lectures and readings I tried to incorporate as much as I could in the following manner –

1. *Implement a pipeline (download - transform - summarize – visualize)*

Created a jupyter notebook on the dataproc cluster which I then used to transform, summarize and create a data processing pipeline with visualizations.

2. *Develop a storage model for the dataset (e.g., a data lake or a database)*

Leveraged Bigquery as a Data warehouse to store and query data and also used Google Cloud Storage buckets as a distributed file system.

3. *Run an algorithm using a parallel programming framework (using Hadoop or IU supercomputing resources)*

I ran a Spark job on Google Cloud's Dataproc cluster (Hadoop Based) using the Python Pyspark wrapper to provide movie recommendations using collaborative filtering with the Alternate Least Squares technique.

4. *Explore a big data cloud platform environment*

Through the course of this project I got the opportunity to explore a variety of Google cloud platform's services including but not limited to google cloud storage buckets, Cloud shell, Bigquery, Dataproc, Compute Engine, GKE, Cloud Endpoints, VPC Networking, IAM and billing and resource management.

- **Barriers or Failures encountered**

1. Lots of troubleshooting and set up steps : I was under the assumption that a VPC network would be set up in the environment but I had to end up configuring it manually, same goes for billing as well.
2. Jar files for spark jobs : When configuring the spark job to run my program failed to run many times as the version of Scala being used for my Spark was incompatible with the Spark bigquery connector jar file and therefore I had to perform a deep dive into google cloud docs to figure out the right version of the connector for my version of scala. (https://github.com/GoogleCloudDataproc/spark-bigquery-connector)
3. Keeping resources in check : We had a limited 25$ amount and I had to be very careful with how I used the compute resources, leaving an instance on by mistake could mean that I would have to pay out of pocket in order to complete my project.

VI. CONCLUSION

Through the course we of this project I was able to successfully implement a movie recommendation system and deploy it on the cloud using Pyspark.

From my experience this project was a fantastic learning experience that helped me understand the importance and impact of cloud in the world of Big data. There we so many new tools and technologies I was able to explore and it was a fun challenge setting things up from scratch, troubleshooting issues with VMs, networks definitely improved my knowledge.

This was one project that tested all the computer science skills that I have gathered over the last 6 years of my higher education and I feel satisfied having created a full fledged project end-to-end.

VII. REFERENCES

1. https://towardsdatascience.com/build-recommendation-system-with-pyspark-using-alternating-least-squares-als-matrix-factorisation-ebe1ad2e7679
2. https://spark.apache.org/docs/latest/ml-collaborative-filtering.html
3. https://grouplens.org/datasets/movielens/
4. https://cloud.google.com/dataproc/docs/quickstarts/create-cluster-console
5. https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.tuning.ParamGridBuilder.html
6. https://sparkbyexamples.com/pyspark/pyspark-explode-array-and-map-columns-to-rows/
7. https://cloud.google.com/storage/docs/creating-buckets#storage-create-bucket-gsutil
8. https://cloud.google.com/storage/docs/storage-classes
9. https://towardsdatascience.com/comprehensive-data-explorations-with-matplotlib-a388be12a355
10. https://stackoverflow.com/questions/62408093/one-hot-encoding-multiple-categorical-data-in-a-column
11. https://github.com/nadia1123/movielens-dataset-with-pyspark