BIG DATA MANAGEMENT

GROUP ASSIGNMENT

MUHAMMAD IZZUDDIN BIN AHAMAD SFHAFI WQD170041

Contents

Amazon Prime Movie

Amazon Video is an Internet video on demand service that is developed, owned, and operated by Amazon. It offers television shows and films for rent or purchase and **Prime Video**, a selection of Amazon Studios original content and licenced acquisitions included in the Amazon's Prime subscription. In the United States, access to Prime Video is also available through a video-only membership, which does not require a full Prime subscription.[2]In countries like France and Italy, Rent or Buy and Prime Video are not available on the Amazon website and Prime Video content is only accessible through a dedicated website. In countries like the United States and the United Kingdom, Amazon Video additionally offers **Amazon Channels**, which allows viewers to subscribe to other suppliers' content, including HBO in the United States

1.1 Objective

In this Assignment. We will try to implement several Hadoop technology for our data analytic and recommendation system. We will use Hive and pig for our data analytic and exploration and Spark for our recommendation system.

1.2 Dataset

Link : *https://grouplens.org/datasets/movielens/*

Since we are not able to find Amazon Movie Dataset for data analytic and recommendation system. We will qcquire our dataset from MovieLens website. This dataset consist 100, 000 ratings applied to 9000 movies and 700 users. We will used 3 dataset from this source which:

u.data - Columns consist of user id | item id | rating | timestamp The time stamps are unix seconds since 1/1/1970 UTC

u.item – information for each movie which consist of movie id | movie title | release date | video release date IMDb URL | unknown | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western | The movie ids are the ones used in the u.data data set.

u.user – information about the user which consist of user id | age | gender | occupation | zip code

2.HIVE

Hadoop ecosystem allow user to work on large dataset. Hadoop apply Mapreduce to break the computation tasks into unit that can be distributed around cluster of computer and server thus providing cost-effective and horizontal scalability.

However, another challenge occurred on how to move the existing data infrastructure which made of traditional drelational databases and SQL?. This is where Hive come from. Hive provide an SQL dialect called Hive Querry Language (HiveQL) for querying data stored in Hadoop cluster (Jason Ruthergeln, 2015)

We use Hive for our data exploration to gain initial insight regarding the dataset. We perform join operation on the dataset u.data, u.item and u.user

2.1 Gender population

Using HiveQL, we will query the Hadoop cluster regarding the total number for Male and Female user of our user. From the data set , we create userinfo table from u.user, names table table from u.item and ratings table from u.data.

```
DROP VIEW topUserIDs;

CREATE VIEW topUserIDs AS
SELECT userID,rating
FROM ratings;


SELECT u.gender, COUNT(u.gender) as gendercount
FROM topUserIDs t JOIN userinfo u ON t.userID = u.userid
GROUP BY u.gender
ORDER BY gendercount DESC;
```

Figure 1:Gender Count Hive Script

u.gender	gendercount
M	74260
F	25740

Figure 2: Gender Population

Male made around 3 times the population of female user.

2.2 Average Rating Based on Gender

We try to find out rating given on average in total for both male and female using the Hive sciprt as below.

```
DROP VIEW topUserIDs;

CREATE VIEW topUserIDs AS
SELECT userID,rating
FROM ratings;


SELECT u.gender, AVG(rating) as avgrating
FROM topUserIDs t JOIN userinfo u ON t.userID = u.userid
GROUP BY u.gender
ORDER BY avgrating DESC;
```

Figure 3: Script for Average Rating Based on Gender Hive Script

u.gender	avgrating
F	3.5315073815073816
M	3.5292889846485322

Figure 4: average rating given based on gender result

On total average, both male and female rated the movie almost the same around 3.5 rating.

2.3 Average Rating(Descending) and Count Based on Occupation

Using Group By operation. We try to find out Highest average rating given the type of occupation of our user. We include count column to see the frequency of ratingfor each type of occupation.

```
DROP VIEW occupationRating;

CREATE VIEW IF NOT EXISTS occupationRating AS
SELECT userID,rating,movieID
FROM ratings;


SELECT u.occupation, AVG(rating) as avgrating,COUNT(u.occupation)
FROM occupationRating t JOIN userinfo u ON t.userID = u.userid
GROUP BY u.occupation
ORDER BY avgrating DESC;
```

Figure 5: Average Rating and Count based on Occupation Hive Script

u.occupation	avgrating	_c2
none	3.779134295227525	901
lawyer	3.7353159851301116	1345
doctor	3.688888888888889	540
educator	3.6706206312221985	9442
artist	3.653379549393414	2308
administrator	3.6356464768017114	7479
scientist	3.611273080660836	2058
salesman	3.582943925233645	856
programmer	3.5682604794257147	7801
librarian	3.560781338896264	5273
other	3.5523773797242804	10663
engineer	3.541406727828746	8175
technician	3.5322304620650313	3506
student	3.5151432345038027	21957
marketing	3.4856410256410255	1950
retired	3.4667495338719703	1609
entertainment	3.4410501193317424	2095

Figure 6: The average rating and count based on occupation

We found out that occupation none, lawyer and doctor tend to give higher rating to a movie. But based on the column, these population has lower frequency to rate a movie compare to other type of occupation.

2.4 Average Rating and Count(Descending) Based on Age

Using Group by operation on age, we try to find the average rating based on age. Compare to the previous script, we will arrange the order of our table based on descending total count of our user.

```
DROP VIEW ageRating;

CREATE VIEW IF NOT EXISTS ageRating AS
SELECT userID,rating,movieID
FROM ratings;


SELECT u.age, AVG(rating) as avgrating,COUNT(u.age) as countage
FROM ageRating a JOIN userinfo u ON a.userID = u.userid
GROUP BY u.age
ORDER BY countage DESC;
```

Figure 7: Average Rating and Count Based on user age Hive Script

u.age	avgrating	countage
27	3.525144013700763	6423
24	3.5419227392449515	4556
20	3.6705796038151135	4089
25	3.5778719162721155	4013
22	3.25332998240764	3979
30	3.43886230728336	3762
29	3.4484931506849317	3650
28	3.5418623929262227	3619
32	3.619398752127056	3526
19	3.4009675583380763	3514
26	3.213531258920925	3503
35	3.694320547130538	3363
21	3.46158940397351	3020
33	3.628785301122831	2939
31	3.3790351831701124	2757
23	3.3074916138650763	2683

Figure 8: Average Rating and Count Based on Age

Based on the findings, we found out that most of the user population consist of age 30 and below. With the average rating given around 3.5. This is tally with the finding under our Average Rating By gender section

2.5 Average Rating and Count(Descending) Based on ZipCode

We group by on zipcode to find the highest frequency of zipcode.

```
DROP VIEW ageRating;

CREATE VIEW IF NOT EXISTS ageRating AS
SELECT userID,rating,movieID
FROM ratings;


SELECT u.zipcode, AVG(rating) as avgrating,COUNT(u.zipcode) as countzipcode
FROM ageRating a JOIN userinfo u ON a.userID = u.userid
GROUP BY u.zipcode
ORDER BY countzipcode DESC;
```

Figure 9: Average Rating and Count Based on ZipCode Hive Script

u.zipcode	avgrating	countzipcode
55414	3.5222121486854037	1103
20009	3.489749430523918	878
10019	2.0447058823529414	850
22902	3.081730769230769	832
61820	3.576499388004896	817
48103	3.5777479892761392	746
10003	3.5625	736
60657	2.908029197080292	685
80525	3.4277286135693217	678
83702	3.544600938967136	639
29206	3.09748427672956	636
92626	3.8315602836879434	564
11758	3.8648148148148147	540
55105	3.2393320964749535	539
95064	3.465250965250965	518
55408	3.3517786561264824	506
21218	1.7038539553752536	493

Figure 10: Average Rating and Count based on Zipcode Result

2.6 Top Movie Average Score

Now we will try to find out the highest rating popular movie. We include WHERE ratingsCount>10 inside our script to imply popular movie.

```
CREATE VIEW IF NOT EXISTS movieRating AS
SELECT movieID, AVG(rating) as avgRating, COUNT(movieID) as ratingCount
FROM ratings
GROUP BY movieID
ORDER BY avgRating DESC;

SELECT n.title, avgRating
FROM movieRating m JOIN names n on m.movieID = n.movieID
WHERE ratingCount>10;
```

Figure 11: Top Movie Average Score

n.title	avgrating
Close Shave, A (1995)	4.491071428571429
Schindler's List (1993)	4.466442953020135
Wrong Trousers, The (1993)	4.466101694915254
Casablanca (1942)	4.45679012345679
Wallace & Gromit: The Best of Aardman Animation (1996)	4.447761194029851
Shawshank Redemption, The (1994)	4.445229681978798
Rear Window (1954)	4.3875598086124405
Usual Suspects, The (1995)	4.385767790262173
Star Wars (1977)	4.3584905660377355

Figure 12: Top Movie Average Score

2.7 Top Popular Movie

We try to find out Most popular movie by ordering the rating count

```
DROP VIEW topMovieIDs;

CREATE VIEW topMovieIDs AS
SELECT movieID, count(movieID) as ratingCount
FROM ratings
GROUP BY movieID
ORDER BY ratingCount DESC;

SELECT n.title, ratingCount
FROM topMovieIDs t JOIN names n ON t.movieID =
```

Figure 13:Top Movie By Total Count Hive Script

n.title	ratingcount
Star Wars (1977)	583
Contact (1997)	509
Fargo (1996)	508
Return of the Jedi (1983)	507
Liar Liar (1997)	485
English Patient, The (1996)	481
Scream (1996)	478
Toy Story (1995)	452
Air Force One (1997)	431
Independence Day (ID4) (1996)	429
Raiders of the Lost Ark (1981)	420

Figure 14: Top Movie By Count

3. PIG

PIG possessed certain similarities with SQL. But there are more differences between these 2. For example, SQL allow user to query the database but not how they want it answered. But in Pig, the user describe exactly how to process the input data.

Another Major differences is that SQL revolve around answering one query at a time. This require user to write separate queries and store them into temporary table if they have multiple queries operation. Pig on the other hand, is designed to execute long series of data operations. (Dai, 2017)

SQL is design to handle relational database where the data is properly structured while Pig is designed for data processing environment where the scehma are sometimes unknown or inconsistence.

Compare to Hive Hive, we perform data exploration on the dataset using more complex manner. Below are diagram for the code and the result

3.1 Most Rated 1 star Movie

We perform the Pig script to find the lowest rated move. The implementation of the pig script can be seen as below.

```
ratings = LOAD '/user/maria_dev/ml-100k/u.data' AS (userID:int, movieID:int, rating:int, ratingTime:int);

metadata = LOAD '/user/maria_dev/ml-100k/u.item' USING PigStorage('|')
    AS (movieID:int, movieTitle:chararray, releaseDate: chararray, videoRelease: chararray, imdbLink: charar

nameLookup = FOREACH metadata GENERATE movieID, movieTitle;

groupedRatings = GROUP ratings BY movieID;

averageRatings = FOREACH groupedRatings GENERATE group AS movieID,
    AVG(ratings.rating) AS avgRating, COUNT(ratings.rating) AS numRatings;

 badMovies = FILTER averageRatings BY avgRating <2.0;

 nameBadMovies = JOIN badMovies BY movieID, nameLookup BY movieID;

 finalResults = FOREACH nameBadMovies GENERATE nameLookup::movieTitle AS movieName,
    badMovies::avgRating AS avgRating, badMovies::numRatings AS numRatings;

FinalResultsSorted = ORDER finalResults BY numRatings DESC;

DUMP FinalResultsSorted;
```

Figure 15: Most Rated 1 star Movie Pig Script

```
(Leave It to Beaver (1997),1.8409090909090908,44)
(Mortal Kombat: Annihilation (1997),1.9534883720930232,43)
(Crow: City of Angels, The (1996),1.9487179487179487,39)
(Bio-Dome (1996),1.903225806451613,31)
(Barb Wire (1996),1.9333333333333333,30)
(Free Willy 3: The Rescue (1997),1.7407407407407407,27)
(Showgirls (1995),1.9565217391304348,23)
(Lawnmower Man 2: Beyond Cyberspace (1996),1.7142857142857142,21)
(Children of the Corn: The Gathering (1996),1.3157894736842106,19)
(Home Alone 3 (1997),1.894736842105263,19)
(Ready to Wear (Pret-A-Porter) (1994),1.8333333333333333,18)
(Jaws 3-D (1983),1.9375,16)
(All Dogs Go to Heaven 2 (1996),1.8666666666666667,15)
(Amityville II: The Possession (1982),1.6428571428571428,14)
(Big Bully (1996),1.8571428571428572,14)
(Body Parts (1991),1.6153846153846154,13)
(Vampire in Brooklyn (1995),1.8333333333333333,12)
(Mr. Magoo (1997),1.9166666666666667,12)
(Solo (1996),1.8333333333333333,12)
(Gone Fishin' (1997),1.8181818181818181,11)
(Robocop 3 (1993),1.7272727272727273,11)
(Kazaam (1996),1.8,10)
(Bloodsport 2 (1995),1.7,10)
```

Figure 16: Most Rated 1 star Movie result

3.2 Oldest Movie With 5 Star Rating

Now we will do data exploration to find the oldest movie with the highest rating

Note to the team: The descending order cannot to arrange the order from latest to oldest movie cannot be found. Hence why we start we oldest movie

```
ratings = LOAD '/user/maria_dev/ml-100k/u.data' AS (userID:int, movieID:int, rating:int, ratingTime:int);

metadata = LOAD '/user/maria_dev/ml-100k/u.item' USING PigStorage('|')
    AS (movieID:int, movieTitle:chararray, releaseDate: chararray, videoRelease: chararray, imdbLink: charar

nameLookup = FOREACH metadata GENERATE movieID, movieTitle,
    ToUnixTime(ToDate(releaseDate, 'dd-MMM-yyyy')) AS releaseTime;

ratingsByMovie = GROUP ratings By movieID;

avgRatings = FOREACH ratingsByMovie GENERATE group AS movieID, AVG(ratings.rating) AS avgRating;

fiveStarMovies = FILTER avgRatings BY avgRating > 4.0;

fiveStarsWithData = JOIN fiveStarMovies BY movieID, nameLookup BY movieID;

oldestFiveStarMovies = ORDER fiveStarsWithData BY nameLookup::releaseTime;

DUMP oldestFiveStarMovies;
```

Figure 17: Oldest Movie With 5 Star Rating Pig Script

Results

```
(493,4.15,493,Thin Man, The (1934),-1136073600)
(604,4.012345679012346,604,It Happened One Night (1934),-1136073600)
(615,4.0508474576271185,615,39 Steps, The (1935),-1104537600)
(1203,4.0476190476190474,1203,Top Hat (1935),-1104537600)
(613,4.037037037037037,613,My Man Godfrey (1936),-1073001600)
(633,4.057971014492754,633,Christmas Carol, A (1938),-1009843200)
(132,4.0772357723577235,132,Wizard of Oz, The (1939),-978307200)
(1122,5.0,1122,They Made Me a Criminal (1939),-978307200)
(136,4.123809523809523,136,Mr. Smith Goes to Washington (1939),-978307200)
(478,4.115384615384615,478,Philadelphia Story, The (1940),-946771200)
(524,4.021739130434782,524,Great Dictator, The (1940),-946771200)
(484,4.2101449275362315,484,Maltese Falcon, The (1941),-915148800)
(134,4.292929292929293,134,Citizen Kane (1941),-915148800)
(483,4.45679012345679,483,Casablanca (1942),-883612800)
(659,4.078260869565217,659,Arsenic and Old Lace (1944),-820540800)
(611,4.1,611,Laura (1944),-820540800)
(496,4.121212121212121,496,It's a Wonderful Life (1946),-757382400)
```

Figure 18: Oldest 5 Star Movie Result

4. SPARK

Spark is primarily written in Scala. It consist of a driver process and set of executor process. The driver process is responsible for 3 things which is maintaining information about spark application, responding to user's program and analysing, distributing and scheduling work across the executer

While executer is responsible for 2 things : executing code assign to it by the driver and report the state of computation back to the driver node.

The cluster manager of Spark controls physical machines and allocate resources to Spark application. This can be one of several core cluster manager, Spark standalone cluster manager, YARN or mesos. (Chambers, 2018)

We introduced 3 script under Spark. 2 of the script will be used to explore the dataset (LowestRatedMovie and LowestRatedPopularMovie). The differences between these 2 scripts is that for LowestRatedPopularMovie, the movie that has been rated more than 10 times will be selected (to imply popular). The third script will be used for our movie recommendation system based on targeted userID past preferences. For Spark, we will run the script through our Hadoop terminal

4.1 Lowest Rated Movie

We will run a python script under Spark to find the lowest rated moves for our dataset.

```
from pyspark import SparkConf, SparkContext

# This function just creates a Python "dictionary" we can later
# use to convert movie ID's to movie names while printing out
# the final results.
def loadMovieNames():
    movieNames = {}
    with open("ml-100k/u.item") as f:
        for line in f:
            fields = line.split('|')
            movieNames[int(fields[0])] = fields[1]
    return movieNames

# Take each line of u.data and convert it to (movieID, (rating, 1.0))
# This way we can then add up all the ratings for each movie, and
# the total number of ratings for each movie (which lets us compute the average)
def parseInput(line):
    fields = line.split()
    return (int(fields[1]), (float(fields[2]), 1.0))

if __name__ == "__main__":
    # The main script - create our SparkContext
    conf = SparkConf().setAppName("WorstMovies")
    sc = SparkContext(conf = conf)

    # Load up our movie ID -> movie name lookup table
    movieNames = loadMovieNames()

    # Load up the raw u.data file
    lines = sc.textFile("hdfs:///user/maria_dev/ml-100k/u.data")

    # Convert to (movieID, (rating, 1.0))
    movieRatings = lines.map(parseInput)

    # Reduce to (movieID, (sumOfRatings, totalRatings))
    ratingTotalsAndCount = movieRatings.reduceByKey(lambda movie1, movie2: ( movie1[0] + movie2[0], movie1[1] + movie2[1] ) )

    # Map to (movieID, averageRating)
    averageRatings = ratingTotalsAndCount.mapValues(lambda totalAndCount : totalAndCount[0] / totalAndCount[1])

    # Sort by average rating
    sortedMovies = averageRatings.sortBy(lambda x: x[1])

    # Take the top 10 results
    results = sortedMovies.take(10)

    # Print them out:
    for result in results:
        print(movieNames[result[0]], result[1])
```

Figure 19:Lowest Rated Movie script

```
SPARK_MAJOR_VERSION is set to 2, using Spark2
/usr/hdp/current/spark2-client/python/lib/pyspark.zip/pyspark/context.py:205: UserWarni
ng: Support for Python 2.6 is deprecated as of Spark 2.0.0
  warnings.warn("Support for Python 2.6 is deprecated as of Spark 2.0.0")
('3 Ninjas: High Noon At Mega Mountain (1998)', 1.0)
('Beyond Bedlam (1993)', 1.0)
('Power 98 (1995)', 1.0)
('Bloody Child, The (1996)', 1.0)
('Amityville: Dollhouse (1996)', 1.0)
('Babyfever (1994)', 1.0)
('Homage (1995)', 1.0)
('Somebody to Love (1994)', 1.0)
('Crude Oasis, The (1995)', 1.0)
('Every Other Weekend (1990)', 1.0)
[root@sandbox-hdp maria_dev]#
```

Figure 20: Lowest Rated Movie Result

4.2 Lowest Rated Popular Movie

Unlike Lowest Rated Movie, The Lowest Rated Popular Movie take number of time a particular movie has been rated as a sign of popular with 10 is the minimum threshold.

```python
from pyspark import SparkConf, SparkContext

# This function just creates a Python "dictionary" we can later
# use to convert movie ID's to movie names while printing out
# the final results.
def loadMovieNames():
    movieNames = {}
    with open("ml-100k/u.item") as f:
        for line in f:
            fields = line.split('|')
            movieNames[int(fields[0])] = fields[1]
    return movieNames

# Take each line of u.data and convert it to (movieID, (rating, 1.0))
# This way we can then add up all the ratings for each movie, and
# the total number of ratings for each movie (which lets us compute the average)
def parseInput(line):
    fields = line.split()
    return (int(fields[1]), (float(fields[2]), 1.0))

if __name__ == "__main__":
    # The main script - create our SparkContext
    conf = SparkConf().setAppName("WorstMovies")
    sc = SparkContext(conf = conf)

    # Load up our movie ID -> movie name lookup table
    movieNames = loadMovieNames()

    # Load up the raw u.data file
    lines = sc.textFile("hdfs:///user/maria_dev/ml-100k/u.data")

    # Convert to (movieID, (rating, 1.0))
    movieRatings = lines.map(parseInput)

    # Reduce to (movieID, (sumOfRatings, totalRatings))
    ratingTotalsAndCount = movieRatings.reduceByKey(lambda movie1, movie2: ( movie1[0] + movie2[0], movie1[1] + movie2[1] ) )

    # Filter out movies rated 10 or fewer times
    popularTotalsAndCount = ratingTotalsAndCount.filter(lambda x: x[1][1] > 10)

    # Map to (movieID, averageRating)
    averageRatings = popularTotalsAndCount.mapValues(lambda totalAndCount : totalAndCount[0] / totalAndCount[1])

    # Sort by average rating
    sortedMovies = averageRatings.sortBy(lambda x: x[1])

    # Take the top 10 results
    results = sortedMovies.take(10)

    # Print them out:
    for result in results:
        print(movieNames[result[0]], result[1])
```

Figure 21: Lowest Rated Popular Movie Script

```
[root@sandbox-hdp maria_dev]# spark-submit LowestRatedPopularMovieSpark.py
SPARK_MAJOR_VERSION is set to 2, using Spark2
/usr/hdp/current/spark2-client/python/lib/pyspark.zip/pyspark/context.py:205: UserWarni
ng: Support for Python 2.6 is deprecated as of Spark 2.0.0
  warnings.warn("Support for Python 2.6 is deprecated as of Spark 2.0.0")
('Children of the Corn: The Gathering (1996)', 1.3157894736842106)
('Body Parts (1991)', 1.6153846153846154)
('Amityville II: The Possession (1982)', 1.6428571428571428)
('Lawnmower Man 2: Beyond Cyberspace (1996)', 1.7142857142857142)
('Robocop 3 (1993)', 1.7272727272727273)
('Free Willy 3: The Rescue (1997)', 1.7407407407407407)
("Gone Fishin' (1997)", 1.8181818181818181)
('Solo (1996)', 1.8333333333333333)
('Ready to Wear (Pret-A-Porter) (1994)', 1.8333333333333333)
('Vampire in Brooklyn (1995)', 1.8333333333333333)
[root@sandbox-hdp maria_dev]#
```

Figure 22:Lowest Rated Popular Movie Result

4.3 Code for ALS Recommendation system

The ALS algorithm will be used to design our recommendation system under Spark

```
from pyspark.sql import SparkSession
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row
from pyspark.sql.functions import lit

# Load up movie ID -> movie name dictionary
def loadMovieNames():
    movieNames = {}
    with open("ml-100k/u.item") as f:
        for line in f:
            fields = line.split('|')
            movieNames[int(fields[0])] = fields[1].decode('ascii', 'ignore')
    return movieNames

# Convert u.data lines into (userID, movieID, rating) rows
def parseInput(line):
    fields = line.value.split()
    return Row(userID = int(fields[0]), movieID = int(fields[1]), rating = float(fields[2]))


if __name__ == "__main__":
    # Create a SparkSession
    spark = SparkSession.builder.appName("MovieRecs").getOrCreate()

    # Load up our movie ID -> name dictionary
    movieNames = loadMovieNames()

    # Get the raw data
    lines = spark.read.text("hdfs:///user/maria_dev/ml-100k/u.data").rdd

    # Convert it to a RDD of Row objects with (userID, movieID, rating)
    ratingsRDD = lines.map(parseInput)

    # Convert to a DataFrame and cache it
    ratings = spark.createDataFrame(ratingsRDD).cache()

    # Create an ALS collaborative filtering model from the complete data set
    als = ALS(maxIter=5, regParam=0.01, userCol="userID", itemCol="movieID", ratingCol="rating")
    model = als.fit(ratings)

    # Print out ratings from user 0:
    print("\nRatings for user ID 0:")
    userRatings = ratings.filter("userID = 0")
    for rating in userRatings.collect():
        print movieNames[rating['movieID']], rating['rating']

    print("\nTop 20 recommendations:")
    # Find movies rated more than 100 times
    ratingCounts = ratings.groupBy("movieID").count().filter("count > 100")
    # Construct a "test" dataframe for user 0 with every movie rated more than 100 times
    popularMovies = ratingCounts.select("movieID").withColumn('userID', lit(0))

    # Run our model on that list of popular movies for user ID 0
    recommendations = model.transform(popularMovies)

    # Get the top 20 movies with the highest predicted rating for this user
    topRecommendations = recommendations.sort(recommendations.prediction.desc()).take(20)

    for recommendation in topRecommendations:
        print (movieNames[recommendation['movieID']], recommendation['prediction'])

    spark.stop()

```

Figure 23: ALS recommendation system script

We select userID:10 to test our recommendation system. Based on the user profiling, user 10 like to watch epic movie with emphasize on adventure and plot-twist genre.

```
Ratings for user ID 10:
French Twist (Gazon maudit) (1995) 4.0
Sabrina (1954) 4.0
Brazil (1985) 3.0
Laura (1944) 5.0
Twelve Monkeys (1995) 4.0
Fargo (1996) 5.0
Smoke (1995) 3.0
Sunset Blvd. (1950) 5.0
Secrets & Lies (1996) 5.0
Bonnie and Clyde (1967) 5.0
Evita (1996) 4.0
Boogie Nights (1997) 4.0
Dial M for Murder (1954) 4.0
Notorious (1946) 4.0
Manchurian Candidate, The (1962) 4.0
Secret of Roan Inish, The (1994) 4.0
Stand by Me (1986) 5.0
Mother (1996) 4.0
Hoop Dreams (1994) 4.0
Unforgiven (1992) 4.0
Cape Fear (1991) 4.0
Lone Star (1996) 5.0
Emma (1996) 4.0
Get Shorty (1995) 4.0
House of the Spirits, The (1993) 3.0
Braveheart (1995) 5.0
Dirty Dancing (1987) 4.0
Liar Liar (1997) 3.0
M (1931) 5.0
Shawshank Redemption, The (1994) 4.0
Barcelona (1994) 3.0
39 Steps, The (1935) 4.0
Shining, The (1980) 5.0
Sling Blade (1996) 5.0
Glory (1989) 4.0
Substance of Fire, The (1996) 4.0
```

Figure 24: UserID:10 previous rating

The movie list above is the rating given by userID:10 on the movie that the user has watched.

```
Top 20 recommendations:
(u'Casablanca (1942)', 5.0283746719360352)
(u"Schindler's List (1993)", 4.9566831588745117)
(u'Wrong Trousers, The (1993)', 4.9534454345703125)
(u'Godfather, The (1972)', 4.8714971542358398)
(u'Star Wars (1977)', 4.8696041107177734)
(u'Raiders of the Lost Ark (1981)', 4.8536453247070312)
(u'Godfather: Part II, The (1974)', 4.8526945114135742)
(u'Shawshank Redemption, The (1994)', 4.8311910629272461)
(u'Citizen Kane (1941)', 4.8293161392211914)
(u'Bridge on the River Kwai, The (1957)', 4.8277926445007324)
(u'Amadeus (1984)', 4.8207082748413086)
(u'12 Angry Men (1957)', 4.8165550231933594)
(u'North by Northwest (1959)', 4.8111486434936523)
(u'Maltese Falcon, The (1941)', 4.7980451583862305)
(u'Lawrence of Arabia (1962)', 4.7931046485900879)
(u'Usual Suspects, The (1995)', 4.7915687561035156)
(u'Rear Window (1954)', 4.7852945327758789)
(u'Great Escape, The (1963)', 4.7689089775085449)
(u'African Queen, The (1951)', 4.7559232711791992)
(u'Sling Blade (1996)', 4.7467570304870605)
[root@sandbox-hdp maria_dev]#
```

Figure 25:Recommend Movie tp UserID:10 Based on ALS Algo

Above, The top 20 movie recommend to userID:10 produced by the ALS algorithm (Collaborative Recommendation). The average ratings are the right are the collective average rating given by the whole users in the database

References

Chambers, B. (2018). *Spark: The Definitive Guide.* O'Rilley Media.

Dai, A. G. (2017). *Programming Pig.* O'Reilley Media.

Jason Ruthergeln, D. W. (2015). *Prrogramming Hive.* O'Rilley Media.