

MovieLens 100K Dataset Analysis with Cassandra and Spark

READY

MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota.

This data set consists of 100,000 ratings (1-5) from 943 users on 1682 movies and each user has rated at least 20 movies. It contains simple demographic info for the users (age, gender, occupation, zip)

The project aims to use Cassandra Query Language (CQL) and Spark2 Structured Query Language (SQL) to analyze the user info from the datasets to answer following questions:

1. Calculate the average rating for each movie.
2. Identify the top ten movies with the highest average ratings.
3. Find the users who have rated at least 50 movies and identify their favourite movie genres.
4. Find all the users who are less than 20 years old.
5. Find all the users whose occupation is "scientist" and whose age is between 30 and 40 years old.

This analysis uses Docker to run Zeppelin and Cassandra

1. Setting up the Docker

READY

- a. Create a Docker network

```
docker network create zeppelin-net
```

- b. Start Cassandra container

```
docker network create zeppelin-net
```

c. Start Zeppeling container

```
docker run -d --name zeppelin --network zeppelin-net -p 9090:8080 apache/zeppelin:0.12.0
```

d. Install Spark in Zeppelin container

```
docker exec -it zeppelin bash

cd /tmp

wget https://archive.apache.org/dist/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz

tar -xvzf spark-3.1.2-bin-hadoop3.2.tgz
```

e. Reconfigure the interpreter

```
SPARK_HOME -> /tmp/spark-3.1.2-bin-hadoop3.2
spark.master -> local[*]
spark.cassandra.connection.host -> Cassandra
spark.jars.packages -> com.datastax.spark:spark-cassandra-connector_2.12:3.1.0
zeppeline.spark.enableSupportVersionCheck -> false
```

READY

2. Uploading files from local into Zeppelin

```
docker cp "C:\Users\hajar\Downloads\ml-100k\u.user" zeppelin:/tmp/u.user  
  
docker cp "C:\Users\hajar\Downloads\Assignemnt_3\ml-100k\u.data" zeppelin:/tmp/u.data  
  
docker cp "C:\Users\hajar\Downloads\Assignment_3\ml-100k\u.item" zeppelin:/tmp/u.item
```

3. Create keyspaces and tables in Cassandra

READY

In Cassandra Shell

```
docker exec -it cassandra cqlsh
```

a) Create the keyspace

```
CREATE KEYSPACE movielens  
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};  
  
USE movielens;
```

b) Create the tables

```
CREATE TABLE users (  
    user_id int PRIMARY KEY,  
    age int,  
    gender text,  
    occupation text,  
    zip_code text  
);  
  
CREATE TABLE ratings (  
    user_id int,  
    movie_id int,  
    rating int,  
    timestamp bigint,  
    PRIMARY KEY (user_id, movie_id)  
);  
  
CREATE TABLE titles (  
    movie_id int PRIMARY KEY,  
    title text,  
    release_date text,  
    imdb_url text,  
    unknown int,  
    action int,  
    adventure int,  
    animation int,  
    children int,  
    comedy int,  
    crime int,  
    documentary int,  
    drama int,  
    fantasy int,  
    film_noir int,  
    horror int,  
    musical int,  
    mystery int,  
    romance int,  
    sci_fi int,  
    thriller int,  
    war int,
```

```
western int
);
```

```
%pyspark
#test Spark
spark.range(1).show()
```

```
+---+
| id|
+---+
|  0|
+---+
```

READY

4. Loading data from Zeppelin and write into Cassandra.

READY

Here we create RDD (Resilient Distributed Dataset) objects

```
%pyspark

#libraries

from pyspark.sql import Row
from pyspark.sql import functions as F
```

READY

User Data

READY

```
%pyspark

#to parse function
def parseInput(line):
    x = line.split('|') # the delimiter used is pipe
    return Row(user_id=int(x[0]), age=int(x[1]), gender=x[2], occupation=x[3], zip_code=x[4])
```

READY

```
# Load raw u.user file
lines = spark.sparkContext.textFile("file:///tmp/u.user")

# Apply transformation
users = lines.map(parseInput)

# Create DataFrame
usersDataset = spark.createDataFrame(users)

# Write to Cassandra
usersDataset.write.format("org.apache.spark.sql.cassandra") \
    .mode("append") \
    .options(table="users", keyspace="movielens").save()

# Read back from Cassandra
readUsers = spark.read.format("org.apache.spark.sql.cassandra").options(table="users", keyspace="movielens").load()
```

```
+-----+---+-----+-----+
|user_id|age|gender|  occupation|zip_code|
+-----+---+-----+-----+
|   885| 30|    F|      other| 95316|
|   791| 31|    M|    educator| 20064|
|   176| 28|    M|   scientist| 07030|
|   393| 19|    M|    student| 83686|
|   582| 17|    M|    student| 93003|
|   371| 36|    M|   engineer| 99206|
|   561| 23|    M|   engineer| 60005|
|    89| 43|    F| administrator| 68106|
|    78| 26|    M| administrator| 61801|
|   265| 26|    M|   executive| 84601|
|   216| 22|    M|   engineer| 02215|
|   243| 33|    M|   educator| 60201|
|   674| 13|    F|    student| 55337|
|   111| 57|    M|   engineer| 90630|
|    72| 40|    F| administrator| 72024|
```

Ratings Data

READY

```
%pyspark

from pyspark.sql import Row
from pyspark.sql import functions as F
```

READY

```

# ParseInput Function
def parseInput2(line):
    x = line.split('\t') # the delimiter use is tab
    return Row(user_id=int(x[0]), movie_id=int(x[1]), rating=x[2], timestamp=x[3])

# Load raw u.data file
lines = spark.sparkContext.textFile("file:///tmp/u.data")

# mapping to row
ratings = lines.map(parseInput2)

# Create DataFrame
ratingsDataset = spark.createDataFrame(ratings)

# Write to Cassandra
ratingsDataset.write.format("org.apache.spark.sql.cassandra").mode("append").options(table="ratings", keyspace="movielens").save()

# Read back from Cassandra
readRatings = spark.read.format("org.apache.spark.sql.cassandra") \
    .options(table="ratings", keyspace="movielens").load()

readRatings.show()

```

```

+-----+-----+-----+-----+
|user_id|movie_id|rating|timestamp|
+-----+-----+-----+-----+
|    384|    258|     4|891273683|
|    384|    271|     4|891283502|
|    384|    272|     5|891273509|
|    384|    286|     4|891273649|
|    384|    289|     5|891283502|
|    384|    300|     4|891273809|
|    384|    302|     5|891273509|
|    384|    313|     5|891273683|
|    384|    316|     5|891274055|
|    384|    327|     4|891273761|
|    384|    328|     4|891274091|
|    384|    329|     3|891273761|
|    384|    333|     4|891273509|
|    384|    343|     3|891273716|
|    384|    347|     4|891273509|

```

```
%pyspark

from pyspark.sql import Row
from pyspark.sql import functions as F

# ParseInput Function
def parseInput3(line):
    fields = line.split('|') # the delimiter use is pipe
    return Row(
        movie_id=int(fields[0]),
        title=fields[1],
        release_date=fields[2],
        imdb_url=fields[4],
        unknown=int(fields[5]),
        action=int(fields[6]),
        adventure=int(fields[7]),
        animation=int(fields[8]),
        children=int(fields[9]),
        comedy=int(fields[10]),
        crime=int(fields[11]),
        documentary=int(fields[12]),
        drama=int(fields[13]),
        fantasy=int(fields[14]),
        film_noir=int(fields[15]),
        horror=int(fields[16]),
        musical=int(fields[17]),
        mystery=int(fields[18]),
        romance=int(fields[19]),
        sci_fi=int(fields[20]),
        thriller=int(fields[21]),
        war=int(fields[22]),
        western=int(fields[23])
    );

# Load raw u.item file
lines = spark.sparkContext.textFile("file:///tmp/u.item")

movies = lines.map(parseInput3)
moviesDataset = spark.createDataFrame(movies)

#write to cassandra
moviesDataset.write.format("org.apache.spark.sql.cassandra").mode("append").options(table="titles", keyspace="movielens").save()

#read to check
readMovies = spark.read.format("org.apache.spark.sql.cassandra").options(table="titles", keyspace="movielens").load()
```


movie_id	action	adventure	animation	children	comedy	crime	documentary	drama	fantasy	film_noir	horror	imdb_url	musical	mystery	release_date	romance	sci_fi	thriller	title	unknown	war	western
1507	0	0	0	0	1	0	0	0	0	0	0	http://us.imdb.co...	0	0	11-Oct-1996	0	0	0	Three Lives and O...	0	0	0
596	0	0	0	1	1	0	0	0	0	0	0	http://us.imdb.co...	1	0	21-Jun-1996	0	0	0	Hunchback of Notr...	0	0	0
1145	0	0	0	0	0	0	0	1	0	0	0	http://us.imdb.co...	0	0	01-Jan-1994	0	0	0	Blue Chips (1994)	0	0	0
1609	0	0	0	0	0	1	0	0	0	0	0	http://us.imdb.co...	0	0	28-Mar-1997	0	0	0	B*A*P*S (1997)	0	0	0
1365	1	0	0	0	0	0	0	1	0	0	0	http://us.imdb.co...	0	0	01-Jan-1993	0	0	0	Johnny 100 Pesos ...	0	0	0
1364	1	0	0	0	0	0	0	0	0	0	0	http://us.imdb.co...	0	0	04-Oct-1996	0	0	0	Bird of Day (1996)	0	0	0

```
%pyspark
```

READY

```
#to allow DF view as SQL
usersDataset.createOrReplaceTempView("users")
ratingsDataset.createOrReplaceTempView("ratings")
moviesDataset.createOrReplaceTempView("movies")
```

4. Analysis

READY

a. Average rating for each movie

```
%pyspark
```

READY

```
from pyspark.sql.functions import avg

joined_df = readRatings.join(readMovies, on="movie_id")
avg_ratings = joined_df.groupBy("title").agg(avg("rating").alias("avg_rating"))

avg_ratings.limit(20).show()
```

```

+-----+-----+
|          title|      avg_rating|
+-----+-----+
|      Cosi (1996)|          4.0|
|      Psycho (1960)| 4.100418410041841|
| Three Wishes (1995)|3.222222222222223|
| If Lucy Fell (1996)|2.7586206896551726|
|When We Were King...| 4.045454545454546|
|   Annie Hall (1977)| 3.911111111111111|
|   Fair Game (1995)|2.1818181818181817|
|Heavenly Creature...|3.6714285714285713|
|Paris, France (1993)|2.3333333333333335|
|Snow White and th...|3.7093023255813953|
|Night of the Livi...|          3.421875|
|I'll Do Anything ...|          2.6|
|Spanking the Monk...| 3.074074074074074|
|      Mondo (1996)|          3.0|
|Threesome (1994)| 3.838700677410255|

```

b. Top ten movies with the highest average ratings

READY

```
%pyspark
```

```

top10_movies = avg_ratings.orderBy("avg_rating", ascending=False).limit(10)
top10_movies.show()

```

```

+-----+-----+
|          title|avg_rating|
+-----+-----+
|Someone Else's Am...|          5.0|
|Saint of Fort Was...|          5.0|
|Aiqing wansui (1994)|          5.0|
|Marlene Dietrich:...|          5.0|
|   Star Kid (1997)|          5.0|
|Great Day in Harl...|          5.0|
|Entertaining Ange...|          5.0|
|They Made Me a Cr...|          5.0|
|Santa with Muscle...|          5.0|
|  Prefontaine (1997)|          5.0|

```

READY

+-----+-----+

c. Users who have rated at least 50 movies and what are their favourite movie genres

READY

```
%sql
WITH active_users AS ( -- filter users who have rated at least 50 movies
  SELECT user_id
  FROM ratings
  GROUP BY user_id
  HAVING COUNT(*) >= 50
),
user_genres AS ( -- joins ratings with movies to get genre for each movie
  SELECT r.user_id, m.*
  FROM ratings r
  JOIN movies m ON r.movie_id = m.movie_id
  WHERE r.user_id IN (SELECT user_id FROM active_users)
),
genre_counts AS (
  SELECT user_id, genre, COUNT(*) AS count
  FROM (
    SELECT user_id, stack(19, -- to unpivot genre columns to row
      'unknown', unknown, 'Action', action, 'Adventure', adventure, 'Animation', animation,
      'Children', children, 'Comedy', comedy, 'Crime', crime, 'Documentary', documentary,
      'Drama', drama, 'Fantasy', fantasy, 'Film-Noir', film_noir, 'Horror', horror,
      'Musical', musical, 'Mystery', mystery, 'Romance', romance, 'Sci-Fi', sci-fi,
      'Thriller', thriller, 'War', war, 'Western', western
    ) AS (genre, is_genre)
    FROM user_genres
  ) g
  WHERE is_genre = 1 -- the genre is belong to the original genre which 1
  GROUP BY user_id, genre
),
ranked_genres AS ( -- to rank genres based on count
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY count DESC) AS rank
  FROM genre_counts
)
SELECT user_id, genre, count
FROM ranked_genres
WHERE rank = 1 -- top genre for that user
ORDER BY count DESC
```

READY



user_id ▼ **genre** ▼ **count**

655	Drama	410
405	Drama	309
537	Drama	251
450	Drama	237
13	Drama	218
234	Drama	213
416	Drama	212
279	Comedy	211
201	Drama	196

d. Users who are less than 20 years old

READY

```
%pyspark
```

```
under20age = readUsers.filter("age < 20")  
under20age.show()
```

```
+-----+-----+-----+-----+  
|user_id|age|gender|occupation|zip_code|  
+-----+-----+-----+-----+  
|    262| 19|    F|    student| 78264|  
|    142| 13|    M|      other| 48118|  
|    223| 19|    F|    student| 47906|  
|    289| 11|    M|      none| 94619|  
|    618| 15|    F|    student| 44212|  
|    471| 10|    M|    student| 77459|  
|    281| 15|    F|    student| 06059|  
|    621| 17|    M|    student| 60402|  
|    887| 14|    F|    student| 27249|  
|    270| 18|    F|    student| 63119|
```

READY

```
| 36| 19| F| student| 93117|
| 347| 18| M| student| 90210|
| 674| 13| F| student| 55337|
| 375| 17| M| entertainment| 37777|
| 333| 13| M| student| 55337|
```

e. Users whose occupation is "scientist" and whose age is between 30 and 40 years old

READY

```
%sql
```

```
SELECT * FROM users
WHERE occupation = 'scientist' AND age BETWEEN 30 AND 40
```

READY



user_id	age	gender	occupation	zip_code
---------	-----	--------	------------	----------

40	38	M	scientist	27514
----	----	---	-----------	-------

71	39	M	scientist	98034
----	----	---	-----------	-------

74	39	M	scientist	T8H1N
----	----	---	-----------	-------

107	39	M	scientist	60466
-----	----	---	-----------	-------

183	33	M	scientist	27708
-----	----	---	-----------	-------

272	33	M	scientist	53706
-----	----	---	-----------	-------

309	40	M	scientist	70802
-----	----	---	-----------	-------

337	37	M	scientist	10522
-----	----	---	-----------	-------

430	38	M	scientist	98199
-----	----	---	-----------	-------

```
%pyspark
```

READY

```
sc.version
```

'3.1.2'