BDM1 – Assignment- AMPBA Batch 15

Submission By – Deep Kamal Singh

PGID – 12020053

Date – 08/March/2021

Table *of* Contents

Question 1

Consider the two data files (users.csv, transactions.csv). Users file has the following fields:

a) UserID
b) EmailID
c) NativeLanguage
d) Location

Transactions file has the following fields:

a) Transaction_ID
b) Product_ID
c) UserID
d) Price
e) Product_Description

By making use of Spark Core (i.e. without using Spark SQL) find out:

a) Count of unique locations where each product is sold.
b) Find out products bought by each user.
c) Total spending done by each user on each product.

Remember, you have to make use of Spark Core for this question. (15 Marks)

Answer to Q1

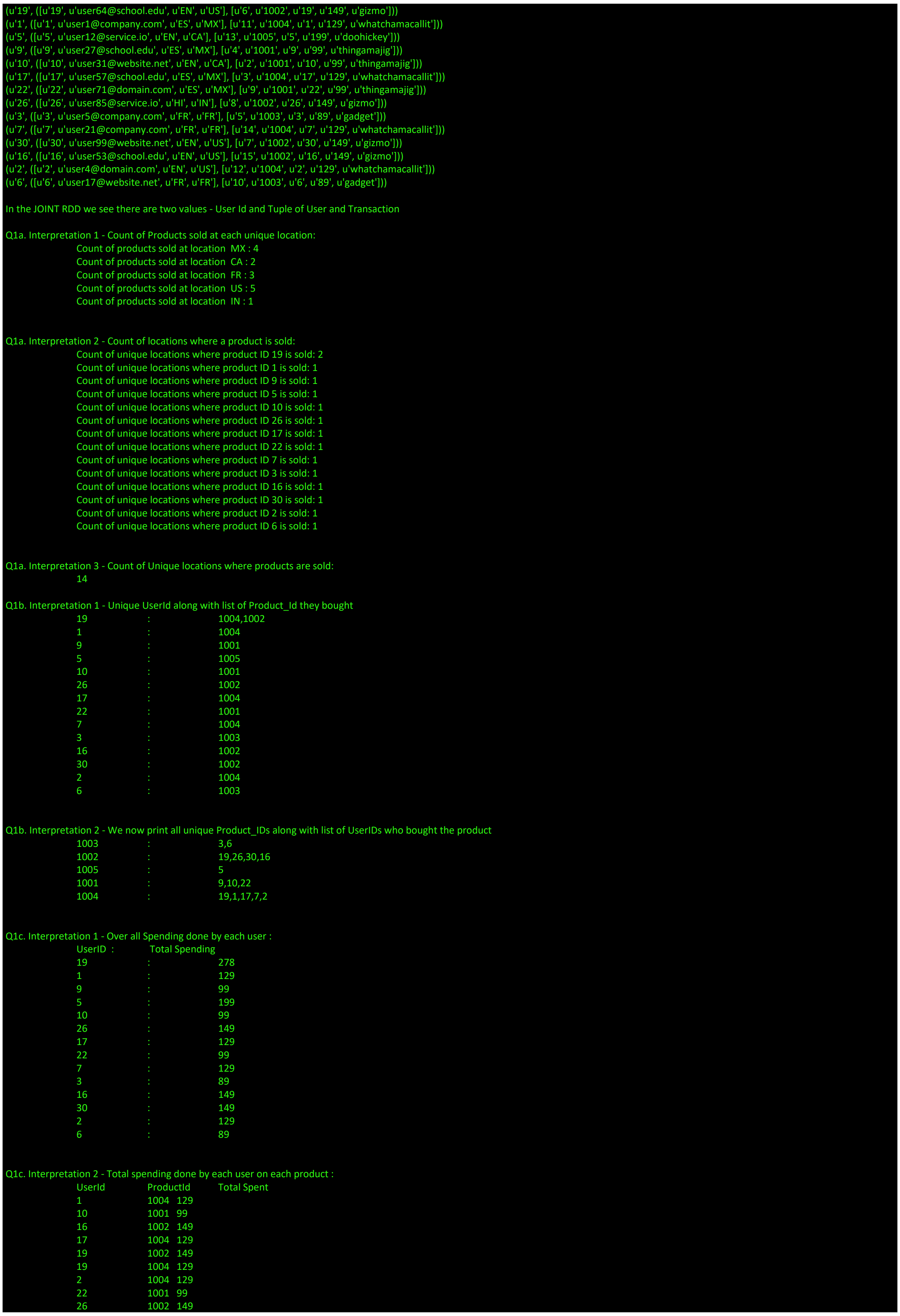
Code File name: BDM1_Assignment_Q1_Deepkamal_Singh.py

How to Execute:

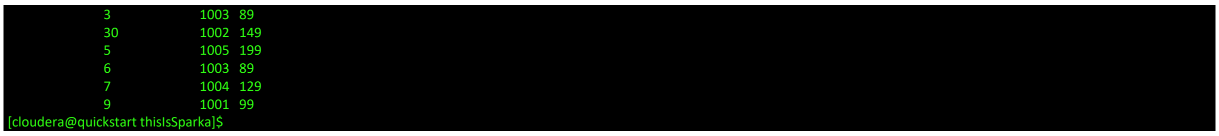
1. Logon to Cloudera VM – or any machine where:
    a. python 2.7.12 is installed
    b. spark-submit command works
    c. Files users.csv and transactions.csv must be placed in a directory, from where below commands will be run
2. Run command:
   `spark-submit BDM1_Assignment_Q1_Deepkamal_Singh.py`
3. Output of the execution will show up in the same console, added in the document

Q1 - Output of spark-submit BDM1_Assignment_Q1_Deepkamal_Singh.py

```
[cloudera@quickstart thisIsSparka]$ spark-submit BDM1_Assignment_Q1_Deepkamal_Singh.py
21/02/27 04:56:22 WARN Utils: Your hostname, quickstart.cloudera resolves to a loopback address: 127.0.0.1; using 172.16.3.2 instead (on interface eth3)
21/02/27 04:56:22 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address

21/02/27 04:56:22 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
21/02/27 04:56:23 INFO SparkContext: Running Spark version 3.0.1
21/02/27 04:56:23 INFO ResourceUtils: ==============================================================
21/02/27 04:56:23 INFO ResourceUtils: Resources for spark.driver:

21/02/27 04:56:23 INFO ResourceUtils: ==============================================================
21/02/27 04:56:23 INFO SparkContext: Submitted application: BDM1-Assignment-Deepkamal_Singh-12020053-Q1Q2
21/02/27 04:56:23 INFO SecurityManager: Changing view acls to: cloudera
21/02/27 04:56:23 INFO SecurityManager: Changing modify acls to: cloudera
21/02/27 04:56:23 INFO SecurityManager: Changing view acls groups to:
21/02/27 04:56:23 INFO SecurityManager: Changing modify acls groups to:
21/02/27 04:56:23 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(cloudera); groups with view permissions: Set(); users  with
modify permissions: Set(cloudera); groups with modify permissions: Set()
21/02/27 04:56:23 INFO Utils: Successfully started service 'sparkDriver' on port 38207.
21/02/27 04:56:23 INFO SparkEnv: Registering MapOutputTracker
21/02/27 04:56:23 INFO SparkEnv: Registering BlockManagerMaster
21/02/27 04:56:23 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
21/02/27 04:56:23 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
21/02/27 04:56:23 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
21/02/27 04:56:23 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-c6074bde-72de-4bcf-87d7-2c40a3890096
21/02/27 04:56:23 INFO MemoryStore: MemoryStore started with capacity 366.3 MiB
21/02/27 04:56:23 INFO SparkEnv: Registering OutputCommitCoordinator
21/02/27 04:56:23 INFO Utils: Successfully started service 'SparkUI' on port 4040.
21/02/27 04:56:23 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://172.16.3.2:4040
21/02/27 04:56:24 INFO Executor: Starting executor ID driver on host 172.16.3.2
21/02/27 04:56:24 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 47362.
21/02/27 04:56:24 INFO NettyBlockTransferService: Server created on 172.16.3.2:47362
21/02/27 04:56:24 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
21/02/27 04:56:24 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 172.16.3.2, 47362, None)
21/02/27 04:56:24 INFO BlockManagerMasterEndpoint: Registering block manager 172.16.3.2:47362 with 366.3 MiB RAM, BlockManagerId(driver, 172.16.3.2, 47362, None)
21/02/27 04:56:24 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 172.16.3.2, 47362, None)
21/02/27 04:56:24 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 172.16.3.2, 47362, None)
/home/cloudera/anaconda2/lib/python2.7/site-packages/pyspark/python/lib/pyspark.zip/pyspark/context.py:225: DeprecationWarning: Support for Python 2 and Python 3 prior to version 3.6 is
deprecated as of Spark 3.0. See also the plan for dropping Python 2 support at https://spark.apache.org/news/plan-for-dropping-python-2-support.html.
  DeprecationWarning)

users.csv is loaded ,now viewing the loaded data, taking 5:
1,user1@company.com,ES,MX
2,user4@domain.com,EN,US
3,user5@company.com,FR,FR
4,user9@site.org,HI,IN
5,user12@service.io,EN,CA

We find that values are loaded in single string for every entry - we will split it with separator, taking 5:
[u'1', u'user1@company.com', u'ES', u'MX']
[u'2', u'user4@domain.com', u'EN', u'US']
[u'3', u'user5@company.com', u'FR', u'FR']
[u'4', u'user9@site.org', u'HI', u'IN']
[u'5', u'user12@service.io', u'EN', u'CA']

Similarily we loaded transactions.csv, split with separator, now viewing transactions data, taking 5:
[u'1', u'1004', u'19', u'129', u'whatchamacallit']
[u'2', u'1001', u'10', u'99', u'thingamajig']
[u'3', u'1004', u'17', u'129', u'whatchamacallit']
[u'4', u'1001', u'9', u'99', u'thingamajig']
[u'5', u'1003', u'3', u'89', u'gadget']

Joining and viewing RDD - [0]'th index of user_rdd and [2]'nd index of txn_rdd is UserId
(u'19', ([u'19', u'user64@school.edu', u'EN', u'US'], [u'1', u'1004', u'19', u'129', u'whatchamacallit']))
```



```
(u'19', ([u'19', u'user64@school.edu', u'EN', u'US'], [u'6', u'1002', u'19', u'149', u'gizmo']))
(u'1', ([u'1', u'user1@company.com', u'ES', u'MX'], [u'11', u'1004', u'1', u'129', u'whatchamacallit']))
(u'5', ([u'5', u'user12@service.io', u'EN', u'CA'], [u'13', u'1005', u'5', u'199', u'doohickey']))
(u'9', ([u'9', u'user27@school.edu', u'ES', u'MX'], [u'4', u'1001', u'9', u'99', u'thingamajig']))
(u'10', ([u'10', u'user31@website.net', u'EN', u'CA'], [u'2', u'1001', u'10', u'99', u'thingamajig']))
(u'17', ([u'17', u'user57@school.edu', u'ES', u'MX'], [u'3', u'1004', u'17', u'129', u'whatchamacallit']))
(u'22', ([u'22', u'user71@domain.com', u'ES', u'MX'], [u'9', u'1001', u'22', u'99', u'thingamajig']))
(u'26', ([u'26', u'user85@service.io', u'HI', u'IN'], [u'8', u'1002', u'26', u'149', u'gizmo']))
(u'3', ([u'3', u'user5@company.com', u'FR', u'FR'], [u'5', u'1003', u'3', u'89', u'gadget']))
(u'7', ([u'7', u'user21@company.com', u'FR', u'FR'], [u'14', u'1004', u'7', u'129', u'whatchamacallit']))
(u'30', ([u'30', u'user99@website.net', u'EN', u'US'], [u'7', u'1002', u'30', u'149', u'gizmo']))
(u'16', ([u'16', u'user53@school.edu', u'EN', u'US'], [u'15', u'1002', u'16', u'149', u'gizmo']))
(u'2', ([u'2', u'user4@domain.com', u'EN', u'US'], [u'12', u'1004', u'2', u'129', u'whatchamacallit']))
(u'6', ([u'6', u'user17@website.net', u'FR', u'FR'], [u'10', u'1003', u'6', u'89', u'gadget']))

In the JOINT RDD we see there are two values - User Id and Tuple of User and Transaction

Q1a. Interpretation 1 - Count of Products sold at each unique location:
                Count of products sold at location  MX : 4
                Count of products sold at location  CA : 2
                Count of products sold at location  FR : 3
                Count of products sold at location  US : 5
                Count of products sold at location  IN : 1


Q1a. Interpretation 2 - Count of locations where a product is sold:
                Count of unique locations where product ID 19 is sold: 2
                Count of unique locations where product ID 1 is sold: 1
                Count of unique locations where product ID 9 is sold: 1
                Count of unique locations where product ID 5 is sold: 1
                Count of unique locations where product ID 10 is sold: 1
                Count of unique locations where product ID 26 is sold: 1
                Count of unique locations where product ID 17 is sold: 1
                Count of unique locations where product ID 22 is sold: 1
                Count of unique locations where product ID 7 is sold: 1
                Count of unique locations where product ID 3 is sold: 1
                Count of unique locations where product ID 16 is sold: 1
                Count of unique locations where product ID 30 is sold: 1
                Count of unique locations where product ID 2 is sold: 1
                Count of unique locations where product ID 6 is sold: 1


Q1a. Interpretation 3 - Count of Unique locations where products are sold:
                14

Q1b. Interpretation 1 - Unique UserId along with list of Product_Id they bought
                19              :               1004,1002
                1               :               1004
                9               :               1001
                5               :               1005
                10              :               1001
                26              :               1002
                17              :               1004
                22              :               1001
                7               :               1004
                3               :               1003
                16              :               1002
                30              :               1002
                2               :               1004
                6               :               1003


Q1b. Interpretation 2 - We now print all unique Product_IDs along with list of UserIDs who bought the product
                1003            :               3,6
                1002            :               19,26,30,16
                1005            :               5
                1001            :               9,10,22
                1004            :               19,1,17,7,2


Q1c. Interpretation 1 - Over all Spending done by each user :
                UserID  :       Total Spending
                19              :               278
                1               :               129
                9               :               99
                5               :               199
                10              :               99
                26              :               149
                17              :               129
                22              :               99
                7               :               129
                3               :               89
                16              :               149
                30              :               149
                2               :               129
                6               :               89


Q1c. Interpretation 2 - Total spending done by each user on each product :
                UserId          ProductId       Total Spent
                1               1004  129
                10              1001  99
                16              1002  149
                17              1004  129
                19              1002  149
                19              1004  129
                2               1004  129
                22              1001  99
                26              1002  149
```



```
        3           1003  89
        30          1002  149
        5           1005  199
        6           1003  89
        7           1004  129
        9           1001  99
[cloudera@quickstart thisIsSparka]$
```

Question 2

Consider the dataset file Olympics.csv.

This file contains information about the Olympic games, players participating in the games, and details of medals won by them. Using Spark core and the data file, compute the following:

a) Total medals that each country won in a particular sport (such as Gymnastics).
b) In each Olympic games, how many medals has India won?
c) Compute top 3 countries in terms of total medals by each Olympic games year.

Olympics.csv Header:

1) Name
2) Age
3) Country
4) Year
5) Date
6) Sport Category
7) Number of gold medals
8) Number of silver medals
9) Number of bronze medals
10) Total number of medals

Remember, you have to make use of Spark Core for this question. (10 Marks).

Answer to Q2

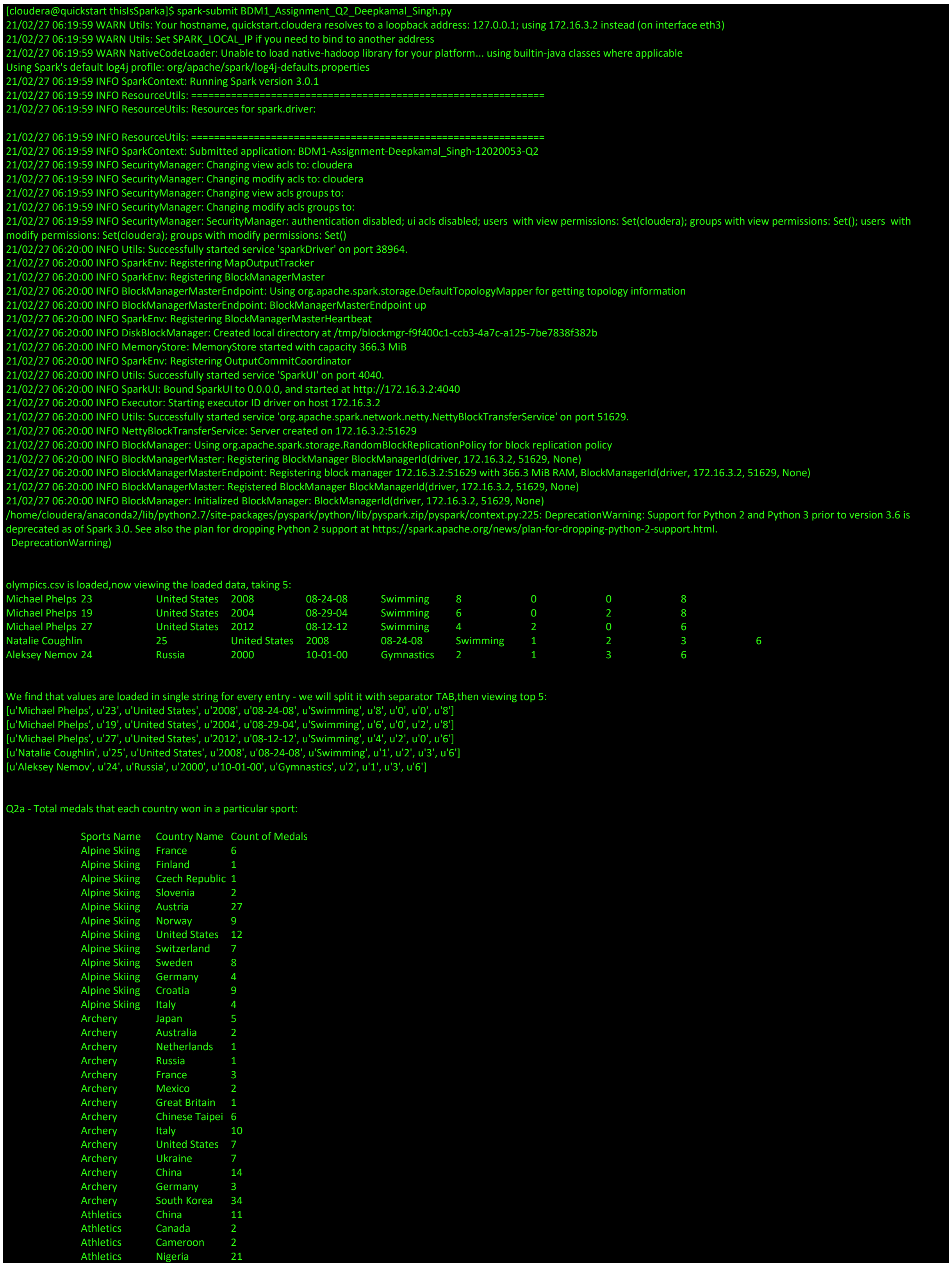
Code File name: BDM1_Assignment_Q2_Deepkamal_Singh.py

How to Execute:

1. Logon to Cloudera VM – or any machine where:
   a. python 2.7.12 is installed
   b. spark-submit command works
   c. File olympics.csv must be placed in a directory, from where below commands will be run
2. Run command:
   spark-submit BDM1_Assignment_Q2_Deepkamal_Singh.py

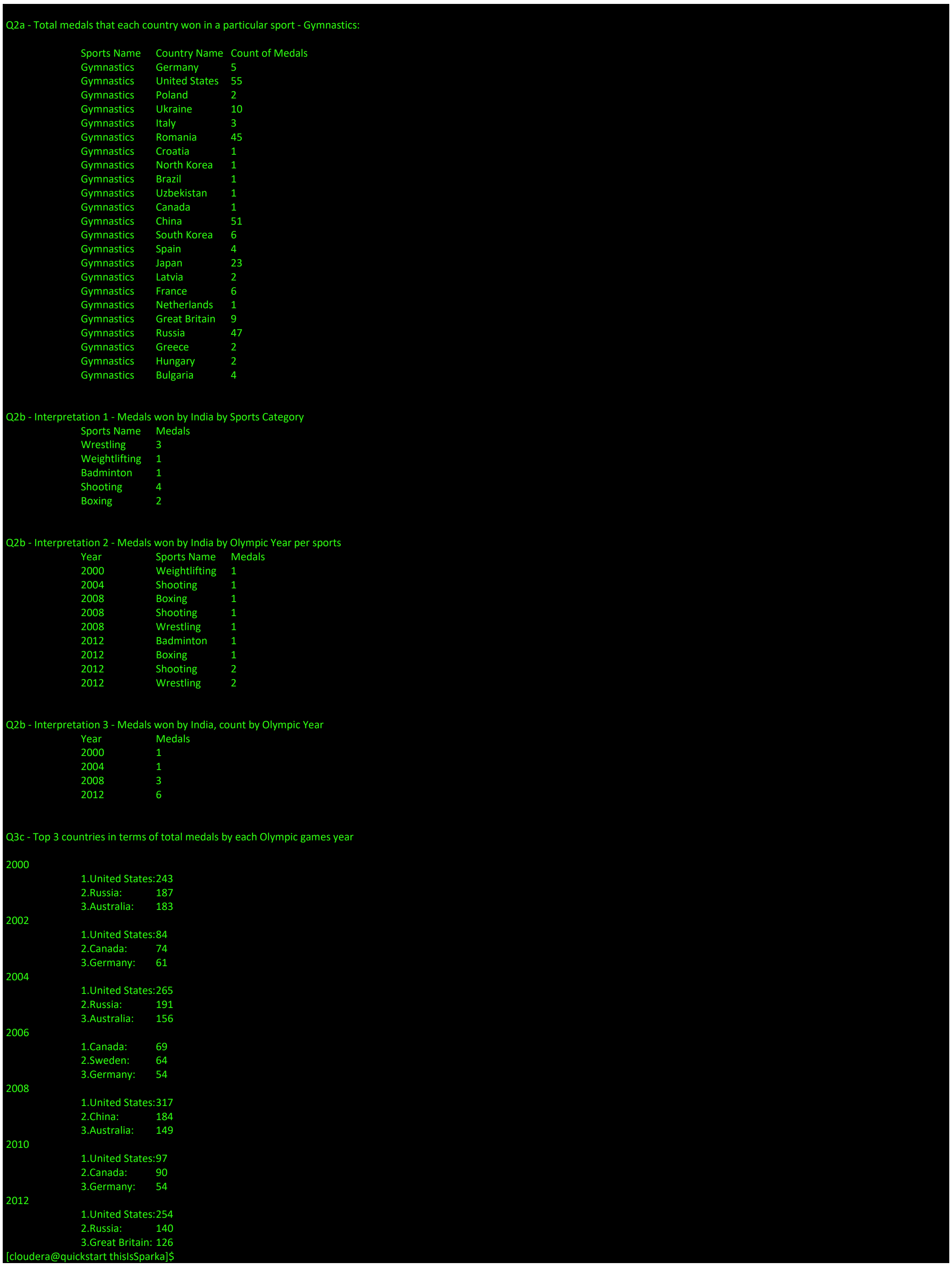
3. Output of the execution will show up in the same console, added in the document

Q2 - Output of spark-submit BDM1_Assignment_Q2_Deepkamal_Singh.py



```
[cloudera@quickstart thisIsSparka]$ spark-submit BDM1_Assignment_Q2_Deepkamal_Singh.py
21/02/27 06:19:59 WARN Utils: Your hostname, quickstart.cloudera resolves to a loopback address: 127.0.0.1; using 172.16.3.2 instead (on interface eth3)
21/02/27 06:19:59 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/02/27 06:19:59 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
21/02/27 06:19:59 INFO SparkContext: Running Spark version 3.0.1
21/02/27 06:19:59 INFO ResourceUtils: ==============================================================
21/02/27 06:19:59 INFO ResourceUtils: Resources for spark.driver:

21/02/27 06:19:59 INFO ResourceUtils: ==============================================================
21/02/27 06:19:59 INFO SparkContext: Submitted application: BDM1-Assignment-Deepkamal_Singh-12020053-Q2
21/02/27 06:19:59 INFO SecurityManager: Changing view acls to: cloudera
21/02/27 06:19:59 INFO SecurityManager: Changing modify acls to: cloudera
21/02/27 06:19:59 INFO SecurityManager: Changing view acls groups to:
21/02/27 06:19:59 INFO SecurityManager: Changing modify acls groups to:
21/02/27 06:19:59 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(cloudera); groups with view permissions: Set(); users  with
modify permissions: Set(cloudera); groups with modify permissions: Set()
21/02/27 06:20:00 INFO Utils: Successfully started service 'sparkDriver' on port 38964.
21/02/27 06:20:00 INFO SparkEnv: Registering MapOutputTracker
21/02/27 06:20:00 INFO SparkEnv: Registering BlockManagerMaster
21/02/27 06:20:00 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
21/02/27 06:20:00 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
21/02/27 06:20:00 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
21/02/27 06:20:00 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-f9f400c1-ccb3-4a7c-a125-7be7838f382b
21/02/27 06:20:00 INFO MemoryStore: MemoryStore started with capacity 366.3 MiB
21/02/27 06:20:00 INFO SparkEnv: Registering OutputCommitCoordinator
21/02/27 06:20:00 INFO Utils: Successfully started service 'SparkUI' on port 4040.
21/02/27 06:20:00 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://172.16.3.2:4040
21/02/27 06:20:00 INFO Executor: Starting executor ID driver on host 172.16.3.2
21/02/27 06:20:00 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 51629.
21/02/27 06:20:00 INFO NettyBlockTransferService: Server created on 172.16.3.2:51629
21/02/27 06:20:00 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
21/02/27 06:20:00 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 172.16.3.2, 51629, None)
21/02/27 06:20:00 INFO BlockManagerMasterEndpoint: Registering block manager 172.16.3.2:51629 with 366.3 MiB RAM, BlockManagerId(driver, 172.16.3.2, 51629, None)
21/02/27 06:20:00 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 172.16.3.2, 51629, None)
21/02/27 06:20:00 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 172.16.3.2, 51629, None)
/home/cloudera/anaconda2/lib/python2.7/site-packages/pyspark/python/lib/pyspark.zip/pyspark/context.py:225: DeprecationWarning: Support for Python 2 and Python 3 prior to version 3.6 is
deprecated as of Spark 3.0. See also the plan for dropping Python 2 support at https://spark.apache.org/news/plan-for-dropping-python-2-support.html.
  DeprecationWarning)


olympics.csv is loaded,now viewing the loaded data, taking 5:
Michael Phelps  23          United States   2008        08-24-08    Swimming    8           0           0           8
Michael Phelps  19          United States   2004        08-29-04    Swimming    6           0           2           8
Michael Phelps  27          United States   2012        08-12-12    Swimming    4           2           0           6
Natalie Coughlin            25              United States   2008    08-24-08    Swimming    1           2           3           6
Aleksey Nemov 24            Russia          2000        10-01-00    Gymnastics  2           1           3           6


We find that values are loaded in single string for every entry - we will split it with separator TAB,then viewing top 5:
[u'Michael Phelps', u'23', u'United States', u'2008', u'08-24-08', u'Swimming', u'8', u'0', u'0', u'8']
[u'Michael Phelps', u'19', u'United States', u'2004', u'08-29-04', u'Swimming', u'6', u'0', u'2', u'8']
[u'Michael Phelps', u'27', u'United States', u'2012', u'08-12-12', u'Swimming', u'4', u'2', u'0', u'6']
[u'Natalie Coughlin', u'25', u'United States', u'2008', u'08-24-08', u'Swimming', u'1', u'2', u'3', u'6']
[u'Aleksey Nemov', u'24', u'Russia', u'2000', u'10-01-00', u'Gymnastics', u'2', u'1', u'3', u'6']


Q2a - Total medals that each country won in a particular sport:

                Sports Name     Country Name    Count of Medals
                Alpine Skiing   France          6
                Alpine Skiing   Finland         1
                Alpine Skiing   Czech Republic  1
                Alpine Skiing   Slovenia        2
                Alpine Skiing   Austria         27
                Alpine Skiing   Norway          9
                Alpine Skiing   United States   12
                Alpine Skiing   Switzerland     7
                Alpine Skiing   Sweden          8
                Alpine Skiing   Germany         4
                Alpine Skiing   Croatia         9
                Alpine Skiing   Italy           4
                Archery         Japan           5
                Archery         Australia       2
                Archery         Netherlands     1
                Archery         Russia          1
                Archery         France          3
                Archery         Mexico          2
                Archery         Great Britain   1
                Archery         Chinese Taipei  6
                Archery         Italy           10
                Archery         United States   7
                Archery         Ukraine         7
                Archery         China           14
                Archery         Germany         3
                Archery         South Korea     34
                Athletics       China           11
                Athletics       Canada          2
                Athletics       Cameroon        2
                Athletics       Nigeria         21
```

[Rest of output of Q2a is deleted intentionally]



```
Q2a - Total medals that each country won in a particular sport - Gymnastics:

                Sports Name   Country Name  Count of Medals
                Gymnastics    Germany       5
                Gymnastics    United States 55
                Gymnastics    Poland        2
                Gymnastics    Ukraine       10
                Gymnastics    Italy         3
                Gymnastics    Romania       45
                Gymnastics    Croatia       1
                Gymnastics    North Korea   1
                Gymnastics    Brazil        1
                Gymnastics    Uzbekistan    1
                Gymnastics    Canada        1
                Gymnastics    China         51
                Gymnastics    South Korea   6
                Gymnastics    Spain         4
                Gymnastics    Japan         23
                Gymnastics    Latvia        2
                Gymnastics    France        6
                Gymnastics    Netherlands   1
                Gymnastics    Great Britain 9
                Gymnastics    Russia        47
                Gymnastics    Greece        2
                Gymnastics    Hungary       2
                Gymnastics    Bulgaria      4


Q2b - Interpretation 1 - Medals won by India by Sports Category
                Sports Name   Medals
                Wrestling     3
                Weightlifting 1
                Badminton     1
                Shooting      4
                Boxing        2


Q2b - Interpretation 2 - Medals won by India by Olympic Year per sports
                Year          Sports Name   Medals
                2000          Weightlifting 1
                2004          Shooting      1
                2008          Boxing        1
                2008          Shooting      1
                2008          Wrestling     1
                2012          Badminton     1
                2012          Boxing        1
                2012          Shooting      2
                2012          Wrestling     2


Q2b - Interpretation 3 - Medals won by India, count by Olympic Year
                Year          Medals
                2000          1
                2004          1
                2008          3
                2012          6


Q3c - Top 3 countries in terms of total medals by each Olympic games year

2000
                1.United States:243
                2.Russia:       187
                3.Australia:    183
2002
                1.United States:84
                2.Canada:       74
                3.Germany:      61
2004
                1.United States:265
                2.Russia:       191
                3.Australia:    156
2006
                1.Canada:       69
                2.Sweden:       64
                3.Germany:      54
2008
                1.United States:317
                2.China:        184
                3.Australia:    149
2010
                1.United States:97
                2.Canada:       90
                3.Germany:      54
2012
                1.United States:254
                2.Russia:       140
                3.Great Britain: 126
[cloudera@quickstart thisIsSparka]$
```

Question 3a.

Consider the Movie Recommendation code and problem that was discussed during the class (Session 4 and 5).

a) Please provide a brief write-up on the problem, steps needed to arrive at the solution (recommendation system), and how exactly those steps are implemented in the code. You can make use of the PPT file that discusses the broad solution. While you are doing so, please also mention what each line of code does (It is not sufficient to mention what each block of code does, you would have to provide explanation for each line). (10 Marks)

b) Create a variation of the code by recommending only good movies (i.e. remove those movies that have bad ratings). (10 Marks)

c) Rather than using the cosine similarity score, make use of another correlation method (for example, Pearson) to compute similarity. Comment on whether your results are similar or very different. (5 Marks)

Problem Statement

Using available data and Apache Spark core, create a movie similarities finder process, such that when given a movie name, it will return list of similar movies.

Available Data

1. Ratings file – This file contains line separated list of Ratings for movies by different user
    a. Format of the file is RatingID::MovieID::Rating(1-5)::UserID
    b. We see users have rated multiple movies.
    c. And Movies are rated by multiple users.

2. Movies File – This file contains list of movies, each separated by a line
    a. Each movie contains these information:
        i. MovieID::Moviename (YEAR)::Genre1|Genre2|…
    b. MovieID is mapped with ratings in rating file

Solution Approach

Based on available data, the most straight forward approach to address problem statement is to correlate each movie with other movie based upon viewership and likeability. However we do not have any viewership data, except the ratings given by users for several movies,
So one suggested approach is to group movies by users who have rated all – such that every movie will be paired with another movie if it has been rated by same user,
To make a scalable linear process – we will perform self-join on Movie rating rdd so we get Movie1 vs Movie2 mapping with their ratings, structured way of doing it in Apache Spark is :

1. Read the data and map it to (user, (movie,rating))
2. Self-join for each user to get all combinations of movie ratings by the user:
   (user, ((movie,rating),(movie,rating))
   Every user rated at least 20 movies, so this blows up the data to huge scale
3. Remove duplicates from the self-join
4. Key by every combination of movies that were rated together:
   ((movie1,movie2),(rating1,rating2))
5. Group pairs of ratings by pairs of movies: ((m1,m2),((r1,r2),(r1,r2),…,(r1,r2)))
6. Calculate statistical similarity for each pair: ((m1,m2),(score,numPairs))
7. Filter and sort the top ten similarities to a movie requested by the user.

Working Solution and Explaining the Code

A working solution was provided – code is as below, in line with code there are comments added to explain the functionality of the code,

The set of comments added already and newly added will be in below font and colour:

```
# Already added comment
```

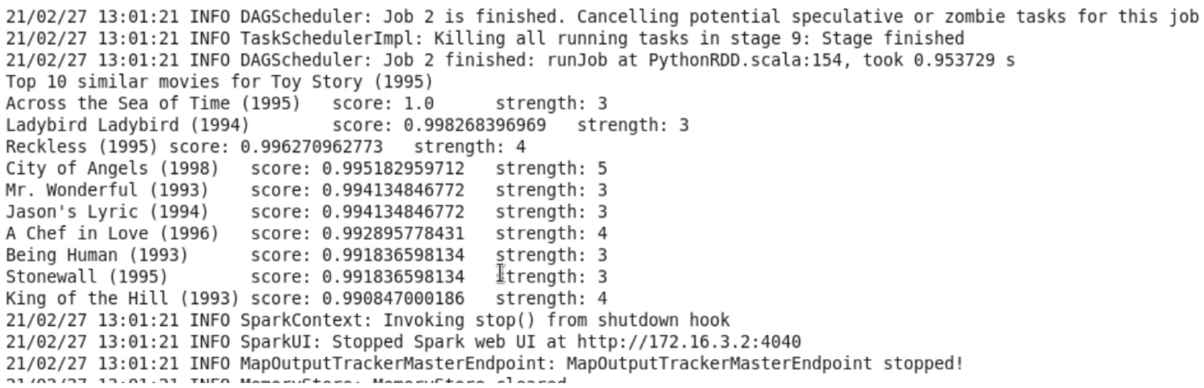
```
# Comments added to explain the code for this assignment
```

```
#------------------------- BDM1_Assignment_Q3_Deepkamal_Singh.py ---------------------#
import sys              # sys will be used for reading runtime arguments
# pyspark is library of Spark functions - this will be used extensively
from pyspark import SparkConf, SparkContext
from math import sqrt # math.sqrt is used in calculating cosine similarity

# We are reading movie data file - and converting them into a dict of
# movieId as key, and name as Value - this is to get O(1) retrieve for
# Movie Name based on Movie ID
def loadMovieNames():
    movieNames = {}
    with open("/home/cloudera/itemfile.txt") as f:
        for line in f:
            fields = line.split('|')
            movieNames[int(fields[0])] = fields[1].decode('ascii', 'ignore')
    return movieNames

# This method accepts a tuple of TWO values - user and ratings, while user
# is not of any significance, ratings is used to extract rating of a user
# for all movies rated by them
def makePairs((user, ratings)):
    (movie1, rating1) = ratings[0]
    (movie2, rating2) = ratings[1]
    return ((movie1, movie2), (rating1, rating2))

# This function simply compares movie IDs,
# it is used to find duplicate keys of rdd
# it will return true when movieID1 and movieID2 are not same,
# will return false if they are same
def filterDuplicates( (userID, ratings) ):
    (movie1, rating1) = ratings[0]
    (movie2, rating2) = ratings[1]
    return movie1 < movie2


# This function accepts list of Tuples with 2 numeric values
# it calculates a bivariate Cosine Similarity score of two values
# Thus it will be called with Movie Rating tuples and it will return a tuple
#                      (CosineScore, Total_number_of_Ratings)
def computeCosineSimilarity(ratingPairs):
    numPairs = 0
    sum_xx = sum_yy = sum_xy = 0
    for ratingX, ratingY in ratingPairs:
        sum_xx += ratingX * ratingX
        sum_yy += ratingY * ratingY
        sum_xy += ratingX * ratingY
        numPairs += 1

    numerator = sum_xy                #
    denominator = sqrt(sum_xx) * sqrt(sum_yy)

    score = 0
    if (denominator):
        score = (numerator / (float(denominator)))

    return (score, numPairs)

# Here we are setting basic configurations to prepare SparkContext,
# SparkMaster sets the master URL to connect to Spark , such as "local" to run
# locally with one thread, "local[4]" to run locally with 4 cores, or
# "spark://master:7077" to run on a Spark standalone cluster.
conf = SparkConf().setMaster("local[*]").setAppName("MovieSimilarities")
# A SparkContext represents the connection to a Spark cluster, and can be used to
# create RDDs, accumulators and broadcast variables on that cluster.
sc = SparkContext(conf = conf)

# nameDict will contain movie name mapped with movieID as key -
# nameDict will be used to print name of movie by Movie ID
print "\nLoading movie names..."
nameDict = loadMovieNames()

# loading datafile - containing ratings from local path
# note that file must exist at the path and is readable
data = sc.textFile("file:///home/cloudera/datafile2.txt")


# Map ratings to key / value pairs: user ID => movie ID, rating
# we are using RDD map function to split the value based on separator,
# and then assigning the key
ratings = data.map(lambda l: l.split()).map(lambda l: (int(l[0]), (int(l[1]), float(l[2]))))

# Emit every movie rated together by the same user.
# Self-join to find every combination.
# we are using RDD Join function to implement self join and create joined ratings
joinedRatings = ratings.join(ratings)
```

```
# At this point our RDD consists of userID => ((movieID, rating), (movieID, rating))

# Filter out duplicate pairs
# We are now applying filterDuplicates function created above to remove duplicate pairs
uniqueJoinedRatings = joinedRatings.filter(filterDuplicates)

# Now key by (movie1, movie2) pairs.
# we are now mapping and applying makePairs method
moviePairs = uniqueJoinedRatings.map(makePairs)

# We now have (movie1, movie2) => (rating1, rating2)
# Now collect all ratings for each movie pair and compute similarity
# we are grouping by the key – which will create list of ratings as value and movie pair tuple
# as key
moviePairRatings = moviePairs.groupByKey()

# We now have (movie1, movie2) = > (rating1, rating2), (rating1, rating2) ...
# Can now compute similarities.
# Here we are applying computeCosineSimilarity to find out ratings correlation between each
# movie
moviePairSimilarities = moviePairRatings.mapValues(computeCosineSimilarity).cache()

# Save the results if desired
#moviePairSimilarities.sortByKey()
#moviePairSimilarities.saveAsTextFile("movie-sims")

# Extract similarities for the movie we care about that are "good".
# We are checking how many commandline arguments were provided during runtime
# command-line argument at [0]'th index will be filename itself,
# on [1]'st we are expecting MovieID
if (len(sys.argv) > 1):

     # this is the minimum match score we are expecting – we are returning
     # cosineSimilarity outcome are neatly bound within [0,1] both inclusive,,
     # meaning the return score will variates between 0 to +1, thus we are marking
     # 0.20 as minimum acceptable similarity condition
    scoreThreshold = 0.20

     # coOccirenceThreshold is measure of total number of users who have given
     # ratings for both movies, higher it is more are the chances it will closer in
     # relevance and similarity, we are setting its value to 2 for test dataset, in
     # real world it should be higher than 400
    coOccurenceThreshold = 2

     # obtaining commandline argument and converting it to integer
     # setting it to movieID
    movieID = int(sys.argv[1])

    # Filter for movies with this sim that are "good" as defined by
    # our quality thresholds above
    # here we are checking that our movieID can be first or second in the pair
    # and applying the filter to return only the movie where similarity score is
    # higher than our set threshold, further we are also validating that number of
    # different ratings available for movie must be higher than set value of co-
    # occurence
    filteredResults = moviePairSimilarities.filter(lambda((pair,sim)): \
        (pair[0] == movieID or pair[1] == movieID) \
        and sim[0] > scoreThreshold and sim[1] > coOccurenceThreshold)

    # Sort by quality score.
    # we want to sort by descending order of score – so we can take out top 10
    results = filteredResults.map(lambda((pair,sim)): (sim, pair)).sortByKey(ascending = False).take(10)

# Printing Name of the movie – note its obsolete print syntax, will not work in python 3 and above
print "Top 10 similar movies for " + nameDict[movieID]
    for result in results:
        (sim, pair) = result
        # Display the similarity result that isn't the movie we're looking at
        similarMovieID = pair[0]    #assuming our movieID is at first position
        if (similarMovieID == movieID):    #if its not, we switch the assignment to second
            similarMovieID = pair[1]
        print nameDict[similarMovieID] + "\tscore: " + str(sim[0]) + "\tstrength: " + str(sim[1])

                        #------------------------- END ---------------------#
```

Execution Output

Command – `spark-submit Movie-Similarities.py 1`



```
21/02/27 13:01:21 INFO DAGScheduler: Job 2 is finished. Cancelling potential speculative or zombie tasks for this job
21/02/27 13:01:21 INFO TaskSchedulerImpl: Killing all running tasks in stage 9: Stage finished
21/02/27 13:01:21 INFO DAGScheduler: Job 2 finished: runJob at PythonRDD.scala:154, took 0.953729 s
Top 10 similar movies for Toy Story (1995)
Across the Sea of Time (1995)   score: 1.0      strength: 3
Ladybird Ladybird (1994)        score: 0.998268396969   strength: 3
Reckless (1995) score: 0.996270962773   strength: 4
City of Angels (1998)   score: 0.995182959712   strength: 5
Mr. Wonderful (1993)    score: 0.994134846772   strength: 3
Jason's Lyric (1994)    score: 0.994134846772   strength: 3
A Chef in Love (1996)   score: 0.992895778431   strength: 4
Being Human (1993)      score: 0.991836598134   strength: 3
Stonewall (1995)        score: 0.991836598134   strength: 3
King of the Hill (1993) score: 0.990847000186   strength: 4
21/02/27 13:01:21 INFO SparkContext: Invoking stop() from shutdown hook
21/02/27 13:01:21 INFO SparkUI: Stopped Spark web UI at http://172.16.3.2:4040
21/02/27 13:01:21 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```

Improvements

1. We are here finding similarities by Cosine function – we can verify other correlations and check it with realistic data to define the right approach
    a. Refer to Answer to Q3c section for implementation of Pearson coefficient

2. We are only checking similarities by correlating User Ratings – however we have other information like movie release year and genre list also
    a. This can be used on top of Correlation similarity to increase relevance

3. Similar movie names can be a quick way to obtain movie sequel/prequels and put them in suggestions too

Question 3b

Create a variation of the code by recommending only good movies (i.e. remove those movies that have bad ratings). (10 Marks)

Answer to 3b

Code File name: BDM1_Assignment_Q3_Deepkamal_Singh.py

About the code

The code is a variation of provided python code Movie-Similarities.py

There are two new functions for similarity scoring are added by name

computeCosineSimilarityWithRatings
This function will also include rating in the return of movie pair, for Rating it will take simple average of all the available user ratings.
and

computePearsonWithRatings
This method calculates Pearson coefficient of all available rating pairs, it follows this formula

$$r_{xy} = \frac{cov(x, y)}{SD_x \times SD_y}$$

These methods are called to prepare similarities RDD from joined and unique rating RDD

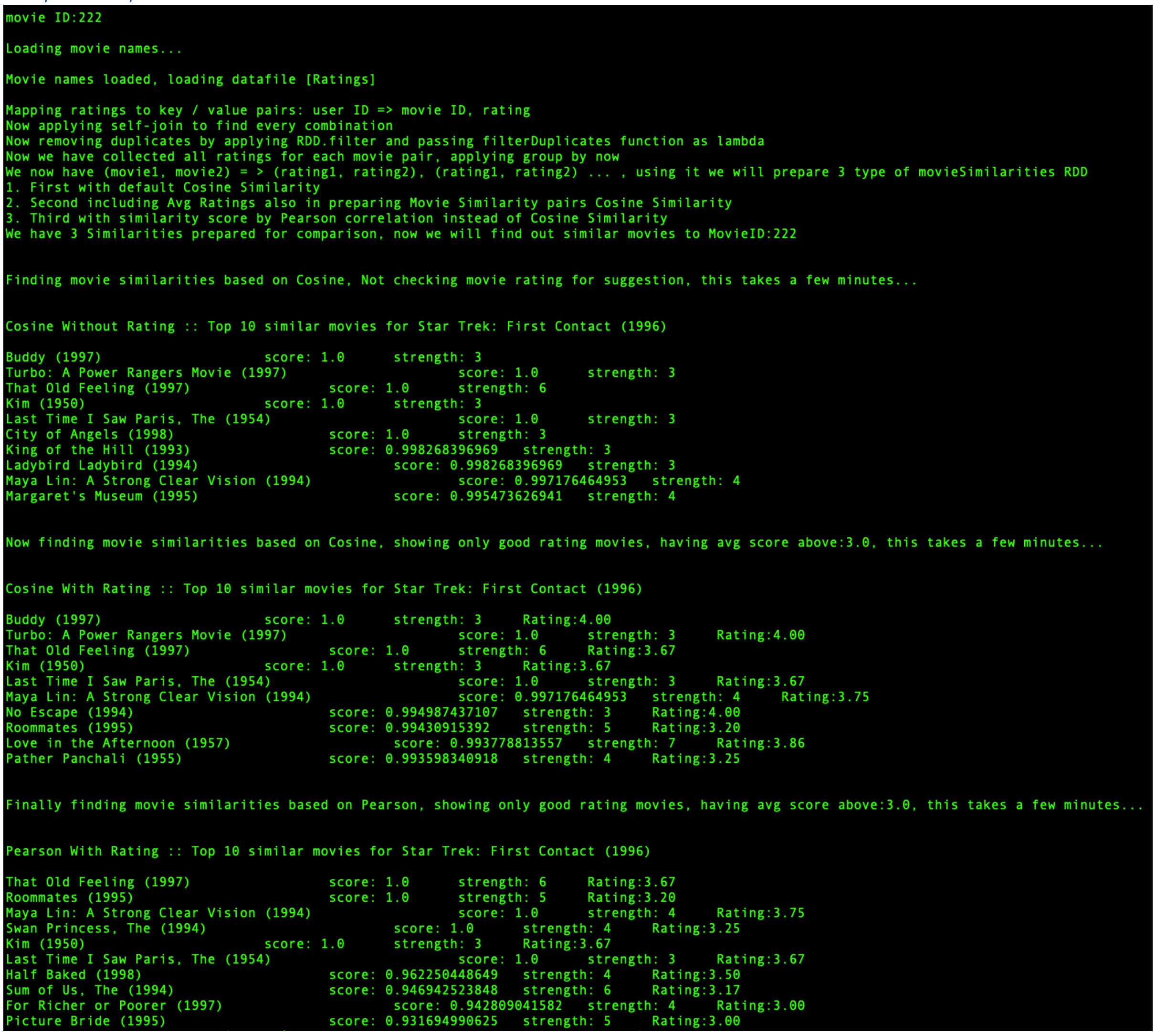
How to Execute:

1. Logon to Cloudera VM – or any machine where:
    a. python 2.7.12 is installed
    b. spark-submit command works
    c. File datafile.txt and itemfile.txt must be placed in a directory, from where below commands will be run
    d. The code is taken from Movie-Similarity.py and modifications are done in the execution behaviour
        i. Now another argument is optionally expected for Rating threshold, it accepts a numeric value which can have float value too.
    e. Code is added with 2 more functions explained in section below, once a movie ID is passed through execution, it shows movie suggestion computes cosineSimilarity without Ratings
2. Run command:
```
spark-submit BDM1_Assignment_Q3_Deepkamal_Singh.py <Movie ID> [<Rating Threshold>]
```
   Example:
```
spark-submit BDM1_Assignment_Q3_Deepkamal_Singh.py 1 3.5
```

3. Output of the execution will show up in the same console, added in the document
4. The output will be containing movie suggestions based on default cosine similarity, it will also show Rating based

Q3 - Output of spark-submit BDM1_Assignment_Q3_Deepkamal_Singh.py

Output Snapshot



```
movie ID:222

Loading movie names...

Movie names loaded, loading datafile [Ratings]

Mapping ratings to key / value pairs: user ID => movie ID, rating
Now applying self-join to find every combination
Now removing duplicates by applying RDD.filter and passing filterDuplicates function as lambda
Now we have collected all ratings for each movie pair, applying group by now
We now have (movie1, movie2) = > (rating1, rating2), (rating1, rating2) ... , using it we will prepare 3 type of movieSimilarities RDD
1. First with default Cosine Similarity
2. Second including Avg Ratings also in preparing Movie Similarity pairs Cosine Similarity
3. Third with similarity score by Pearson correlation instead of Cosine Similarity
We have 3 Similarities prepared for comparison, now we will find out similar movies to MovieID:222


Finding movie similarities based on Cosine, Not checking movie rating for suggestion, this takes a few minutes...


Cosine Without Rating :: Top 10 similar movies for Star Trek: First Contact (1996)

Buddy (1997)                    score: 1.0      strength: 3
Turbo: A Power Rangers Movie (1997)                     score: 1.0      strength: 3
That Old Feeling (1997)                 score: 1.0      strength: 6
Kim (1950)                      score: 1.0      strength: 3
Last Time I Saw Paris, The (1954)                       score: 1.0      strength: 3
City of Angels (1998)                   score: 1.0      strength: 3
King of the Hill (1993)                 score: 0.998268396969   strength: 3
Ladybird Ladybird (1994)                        score: 0.998268396969   strength: 3
Maya Lin: A Strong Clear Vision (1994)                  score: 0.997176464953   strength: 4
Margaret's Museum (1995)                        score: 0.995473626941   strength: 4


Now finding movie similarities based on Cosine, showing only good rating movies, having avg score above:3.0, this takes a few minutes...


Cosine With Rating :: Top 10 similar movies for Star Trek: First Contact (1996)

Buddy (1997)                    score: 1.0      strength: 3     Rating:4.00
Turbo: A Power Rangers Movie (1997)                     score: 1.0      strength: 3     Rating:4.00
That Old Feeling (1997)                 score: 1.0      strength: 6     Rating:3.67
Kim (1950)                      score: 1.0      strength: 3     Rating:3.67
Last Time I Saw Paris, The (1954)                       score: 1.0      strength: 3     Rating:3.67
Maya Lin: A Strong Clear Vision (1994)                  score: 0.997176464953   strength: 4     Rating:3.75
No Escape (1994)                        score: 0.994987437107   strength: 3     Rating:4.00
Roommates (1995)                        score: 0.99430915392    strength: 5     Rating:3.20
Love in the Afternoon (1957)                    score: 0.993778813557   strength: 7     Rating:3.86
Pather Panchali (1955)                  score: 0.993598340918   strength: 4     Rating:3.25


Finally finding movie similarities based on Pearson, showing only good rating movies, having avg score above:3.0, this takes a few minutes...


Pearson With Rating :: Top 10 similar movies for Star Trek: First Contact (1996)

That Old Feeling (1997)                 score: 1.0      strength: 6     Rating:3.67
Roommates (1995)                        score: 1.0      strength: 5     Rating:3.20
Maya Lin: A Strong Clear Vision (1994)                  score: 1.0      strength: 4     Rating:3.75
Swan Princess, The (1994)                       score: 1.0      strength: 4     Rating:3.25
Kim (1950)                      score: 1.0      strength: 3     Rating:3.67
Last Time I Saw Paris, The (1954)                       score: 1.0      strength: 3     Rating:3.67
Half Baked (1998)                       score: 0.962250448649   strength: 4     Rating:3.50
Sum of Us, The (1994)                   score: 0.946942523848   strength: 6     Rating:3.17
For Richer or Poorer (1997)                     score: 0.942809041582   strength: 4     Rating:3.00
Picture Bride (1995)                    score: 0.931694990625   strength: 5     Rating:3.00
```

Full Text Output

```
[cloudera@quickstart thisIsSparka]$ spark-submit BDM1_Assignment_Q3_Deepkamal_Singh.py 222 3
21/02/27 13:39:46 WARN Utils: Your hostname, quickstart.cloudera resolves to a loopback address: 127.0.0.1; using 172.16.3.2 instead (on interface eth3)
21/02/27 13:39:46 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/02/27 13:39:46 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
21/02/27 13:39:47 INFO SparkContext: Running Spark version 3.0.1
21/02/27 13:39:47 INFO ResourceUtils: ==============================================================
21/02/27 13:39:47 INFO ResourceUtils: Resources for spark.driver:

21/02/27 13:39:47 INFO ResourceUtils: ==============================================================
21/02/27 13:39:47 INFO SparkContext: Submitted application: BDM1-Assignment-Q3-Deepkamal_Singh-12020053
21/02/27 13:39:47 INFO SecurityManager: Changing view acls to: cloudera
21/02/27 13:39:47 INFO SecurityManager: Changing modify acls to: cloudera
21/02/27 13:39:47 INFO SecurityManager: Changing view acls groups to:
21/02/27 13:39:47 INFO SecurityManager: Changing modify acls groups to:
21/02/27 13:39:47 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(cloudera); groups with view permissions: Set(); users  with modify permissions:
Set(cloudera); groups with modify permissions: Set()
21/02/27 13:39:47 INFO Utils: Successfully started service 'sparkDriver' on port 37182.
21/02/27 13:39:47 INFO SparkEnv: Registering MapOutputTracker
21/02/27 13:39:47 INFO SparkEnv: Registering BlockManagerMaster
21/02/27 13:39:47 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
21/02/27 13:39:47 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
21/02/27 13:39:47 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
21/02/27 13:39:47 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-adc72014-ded3-46d3-9262-0b4721cb6d50
21/02/27 13:39:47 INFO MemoryStore: MemoryStore started with capacity 366.3 MiB
21/02/27 13:39:47 INFO SparkEnv: Registering OutputCommitCoordinator
21/02/27 13:39:47 INFO Utils: Successfully started service 'SparkUI' on port 4040.
21/02/27 13:39:47 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://172.16.3.2:4040
21/02/27 13:39:48 INFO Executor: Starting executor ID driver on host 172.16.3.2
21/02/27 13:39:48 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 43177.
21/02/27 13:39:48 INFO NettyBlockTransferService: Server created on 172.16.3.2:43177
21/02/27 13:39:48 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
21/02/27 13:39:48 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 172.16.3.2, 43177, None)
21/02/27 13:39:48 INFO BlockManagerMasterEndpoint: Registering block manager 172.16.3.2:43177 with 366.3 MiB RAM, BlockManagerId(driver, 172.16.3.2, 43177, None)
21/02/27 13:39:48 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 172.16.3.2, 43177, None)
21/02/27 13:39:48 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 172.16.3.2, 43177, None)
/home/cloudera/anaconda2/lib/python2.7/site-packages/pyspark/python/lib/pyspark.zip/pyspark/context.py:225: DeprecationWarning: Support for Python 2 and Python 3 prior to version 3.6 is deprecated as of Spark 3.0.
See also the plan for dropping Python 2 support at https://spark.apache.org/news/plan-for-dropping-python-2-support.html.
  DeprecationWarning)

movie ID:222

Loading movie names...

Movie names loaded, loading datafile [Ratings]

Mapping ratings to key / value pairs: user ID => movie ID, rating
Now applying self-join to find every combination
Now removing duplicates by applying RDD.filter and passing filterDuplicates function as lambda
Now we have collected all ratings for each movie pair, applying group by now
We now have (movie1, movie2) = > (rating1, rating2), (rating1, rating2) ... , using it we will prepare 3 type of movieSimilarities RDD
1. First with default Cosine Similarity
2. Second including Avg Ratings also in preparing Movie Similarity pairs Cosine Similarity
3. Third with similarity score by Pearson correlation instead of Cosine Similarity
We have 3 Similarities prepared for comparison, now we will find out similar movies to MovieID:222


Finding movie similarities based on Cosine, Not checking movie rating for suggestion, this takes a few minutes...


Cosine Without Rating :: Top 10 similar movies for Star Trek: First Contact (1996)

Buddy (1997)                              score: 1.0    strength: 3
Turbo: A Power Rangers Movie (1997)                                   score: 1.0    strength: 3
That Old Feeling (1997)                                 score: 1.0    strength: 6
Kim (1950)                                score: 1.0    strength: 3
Last Time I Saw Paris, The (1954)                       score: 1.0    strength: 3
City of Angels (1998)                                   score: 1.0    strength: 3
King of the Hill (1993)                                 score: 0.998268396969       strength: 3
Ladybird Ladybird (1994)                                score: 0.998268396969       strength: 3
Maya Lin: A Strong Clear Vision (1994)                                score: 0.997176464953       strength: 4
Margaret's Museum (1995)                                score: 0.995473626941       strength: 4


Now finding movie similarities based on Cosine, showing only good rating movies, having avg score above:3.0, this takes a few minutes...


Cosine With Rating :: Top 10 similar movies for Star Trek: First Contact (1996)

Buddy (1997)                              score: 1.0    strength: 3   Rating:4.00
Turbo: A Power Rangers Movie (1997)                                   score: 1.0    strength: 3   Rating:4.00
That Old Feeling (1997)                                 score: 1.0    strength: 6   Rating:3.67
Kim (1950)                                score: 1.0    strength: 3   Rating:3.67
Last Time I Saw Paris, The (1954)                       score: 1.0    strength: 3   Rating:3.67
Maya Lin: A Strong Clear Vision (1994)                                score: 0.997176464953       strength: 4   Rating:3.75
No Escape (1994)                          score: 0.994987437107       strength: 3   Rating:4.00
Roommates (1995)                                        score: 0.99430915392        strength: 5   Rating:3.20
Love in the Afternoon (1957)                            score: 0.993778813557       strength: 7   Rating:3.86
Pather Panchali (1955)                                  score: 0.993598340918       strength: 4   Rating:3.25


Finally finding movie similarities based on Pearson, showing only good rating movies, having avg score above:3.0, this takes a few minutes...


Pearson With Rating :: Top 10 similar movies for Star Trek: First Contact (1996)

That Old Feeling (1997)                                 score: 1.0    strength: 6   Rating:3.67
Roommates (1995)                                        score: 1.0    strength: 5   Rating:3.20
Maya Lin: A Strong Clear Vision (1994)                                score: 1.0    strength: 4   Rating:3.75
Swan Princess, The (1994)                               score: 1.0    strength: 4   Rating:3.25
Kim (1950)                                score: 1.0    strength: 3   Rating:3.67
Last Time I Saw Paris, The (1954)                       score: 1.0    strength: 3   Rating:3.67
Half Baked (1998)                                       score: 0.962250448649       strength: 4   Rating:3.50
Sum of Us, The (1994)                                   score: 0.946942523848       strength: 6   Rating:3.17
For Richer or Poorer (1997)                             score: 0.942809041582       strength: 4   Rating:3.00
Picture Bride (1995)                                    score: 0.931694990625       strength: 5   Rating:3.00
```

Question 3c

Rather than using the cosine similarity score, make use of another correlation method (for example, Pearson) to compute similarity. Comment on whether your results are similar or very different. (5 Marks)

Answer to 3c

As explained in answer to Q3b, a new function is added by name computePearsonWithRatings

Similarity scoring by Pearson function

```
def computePearsonWithRatings(ratingPairs):
    # Expect a List of tuples - each tuple with 2 numeric values between 0 and 5
    # Form a series and find correlation
    sum_x = sum_y = numRecords = 0
    for xi, yi in ratingPairs:
        sum_x += xi
        sum_y += yi
        numRecords += 1
    x_bar = sum_x / numRecords
    y_bar = sum_y / numRecords
    moving_x_minus_xbar_sq = 0
    moving_y_minus_ybar_sq = 0
    numerator = 0
    for xi, yi in ratingPairs:
        x_tmp = xi - x_bar
        y_tmp = yi - y_bar
        moving_x_minus_xbar_sq += x_tmp ** 2
        moving_y_minus_ybar_sq += y_tmp ** 2
        numerator += (x_tmp * y_tmp)
    pear_coeff = 0
    if (moving_y_minus_ybar_sq != 0 and moving_x_minus_xbar_sq != 0):
        pear_coeff = numerator / sqrt(moving_x_minus_xbar_sq * moving_y_minus_ybar_sq)
    return (pear_coeff, numRecords, x_bar, y_bar)
```

Output Comparison

Execution for Movie ID 222, with Score rating threshold set at 3

```
spark-submit BDM1_Assignment_Q3_Deepkamal_Singh.py 222 3
```

- Cosine Similarity without ratings check output

```
Cosine Without Rating :: Top 10 similar movies for Star Trek: First Contact (1996)

Buddy (1997)                            score: 1.0      strength: 3
Turbo: A Power Rangers Movie (1997)                     score: 1.0      strength: 3
That Old Feeling (1997)                         score: 1.0      strength: 6
Kim (1950)                              score: 1.0      strength: 3
Last Time I Saw Paris, The (1954)                       score: 1.0      strength: 3
City of Angels (1998)                           score: 1.0      strength: 3
King of the Hill (1993)                         score: 0.998268396969   strength: 3
Ladybird Ladybird (1994)                        score: 0.998268396969   strength: 3
Maya Lin: A Strong Clear Vision (1994)                  score: 0.997176464953   strength: 4
Margaret's Museum (1995)                                score: 0.995473626941   strength: 4
```

- Cosine Similarity with Rating threshold set at 3.0

```
Cosine With Rating :: Top 10 similar movies for Star Trek: First Contact (1996)

Buddy (1997)                            score: 1.0      strength: 3     Rating:4.00
Turbo: A Power Rangers Movie (1997)                     score: 1.0      strength: 3     Rating:4.00
That Old Feeling (1997)                         score: 1.0      strength: 6     Rating:3.67
Kim (1950)                              score: 1.0      strength: 3     Rating:3.67
Last Time I Saw Paris, The (1954)                       score: 1.0      strength: 3     Rating:3.67
Maya Lin: A Strong Clear Vision (1994)          score: 0.997176464953   strength: 4     Rating:3.75
No Escape (1994)                                score: 0.994987437107   strength: 3     Rating:4.00
Roommates (1995)                                score: 0.99430915392    strength: 5     Rating:3.20
Love in the Afternoon (1957)                            score: 0.993778813557   strength: 7     Rating:3.86
Pather Panchali (1955)                          score: 0.993598340918   strength: 4     Rating:3.25
```

- Pearson Similarity with Rating threshold set at 3.0

```
Pearson With Rating :: Top 10 similar movies for Star Trek: First Contact (1996)

That Old Feeling (1997)                 score: 1.0      strength: 6     Rating:3.67
Roommates (1995)                                score: 1.0      strength: 5     Rating:3.20
Maya Lin: A Strong Clear Vision (1994)          score: 1.0      strength: 4     Rating:3.75
Swan Princess, The (1994)                       score: 1.0      strength: 4     Rating:3.25
Kim (1950)                              score: 1.0      strength: 3     Rating:3.67
Last Time I Saw Paris, The (1954)                       score: 1.0      strength: 3     Rating:3.67
Half Baked (1998)                               score: 0.962250448649   strength: 4     Rating:3.50
Sum of Us, The (1994)                           score: 0.946942523848   strength: 6     Rating:3.17
For Richer or Poorer (1997)                             score: 0.942809041582   strength: 4     Rating:3.00
Picture Bride (1995)                            score: 0.931694990625   strength: 5     Rating:3.00
```

- We conclude that Cosine and Pearson output, both with rating threshold are returning different list of movies
- The result list contains around 50-60% similar suggestions,
- Important to note that Cosine correlation function returns scores between 0 to 1, and Pearson varies from -1 to +1

However, we are using same 0.20 as threshold value for both – this can be improved after regression analysis

Appendix

Code for Q1

```
# Importing the required libraries
from pyspark import SparkConf, SparkContext

# Setting SparkConf, setting application name and SparkContext in local variable
# these steps are optional when Apache Spark is running on same machine (from where this code is run) and at
default configs
conf = SparkConf().setAppName("BDM1-Assignment-Deepkamal_Singh-12020053-Q1")
sc = SparkContext(conf=conf)

# We dont want to see DEBUG and INFO level messages generated by PySpark or spark-submit
sc.setLogLevel("ERROR")
'''
Q1. Consider the two data files (users.csv, transactions.csv). Users file has the following fields:
        a) UserID
        b) EmailID
        c) NativeLanguage
        d) Location
    Transactions file has the following fields:
        a) Transaction_ID
        b) Product_ID
        c) UserID
        d) Price
        e) Product_Description
    By making use of Spark Core (i.e. without using Spark SQL) find out:
        a) Count of unique locations where each product is sold.
        b) Find out products bought by each user.
        c) Total spending done by each user on each product.
    Remember, you have to make use of Spark Core for this question. (15 Marks)

Q2. Consider the dataset file Olympics.csv. This file contains information about the Olympic games,
    players participating in the games, and details of medals won by them. Using Spark core and the data file,
compute the following:
        a) Total medals that each country won in a particular sport (such as Gymnastics).
        b) In each Olympic games, how many medals has India won?
        c) Compute top 3 countries in terms of total medals by each Olympic games year.
    Remember, you have to make use of Spark Core for this question. (10 Marks).
'''

##################### Answer to Q1 ############################

# Loading the file `users.csv`
user_rdd = sc.textFile('users.csv')
# Viewing the loaded data
print("\n\nusers.csv is loaded ,now viewing the loaded data, taking 5:")
for aLine in user_rdd.take(5): print(aLine)
# We find that it is loaded as all values in single string for every entry - we will split it with separator

print("\n\nWe find that values are loaded in single string for every entry - we will split it with separator,
taking 5:")
user_rdd = user_rdd.map(lambda x: x.split(','))
for aLine in user_rdd.take(5): print(aLine)
# Similarily we load data from `transactions.csv` as well
txn_rdd = sc.textFile('transactions.csv')
txn_rdd = txn_rdd.map(lambda x: x.split(','))
print("\n\nSimilarily we loaded transactions.csv, split with separator, now viewing transactions data, taking 5:")
for aLine in txn_rdd.take(5): print(aLine)

# We have printed out and verified the user_rdd and txn_rdd RDDs

"""
Objective a) Count of unique locations where each product is sold.
Objective b) Find out products bought by each user.
Objective c) Total spending done by each user on each product.

To obtain these, we need to join User and Transaction RDD on basis of UserID first

"""

# Joining and viewing RDD - [0]'th index of user_rdd and [2]'nd index of txn_rdd is UserId,
print("\n\nJoining and viewing RDD - [0]'th index of user_rdd and [2]'nd index of txn_rdd is UserId")
joint_rdd = user_rdd.map(lambda x: (x[0], x)).join(txn_rdd.map(lambda x: (x[2], x)))
for aLine in joint_rdd.collect(): print(aLine)

# In the JOINT RDD we see there are two values - User Id and Tuple of User and Transaction
#   User Tuple   (0:userID, 1:EmailID, 2:NativeLanguage,3:Location)
#   Transaction Tuple (0:TransactionID, 1:Product_ID, 2:UserID, 3:Price, 4:Product_Description)
print("\n\nIn the JOINT RDD we see there are two values - User Id and Tuple of User and Transaction")
print("\tUser Tuple   (0:userID, 1:EmailID, 2:NativeLanguage,3:Location)")
print("\tTransaction Tuple (0:TransactionID, 1:Product_ID, 2:UserID, 3:Price, 4:Product_Description)")
"""
Objective a) Count of unique locations where each product is sold.
```

```
We will extract location and product Id from Joint RDD, keeping location on 0 index - and then apply groupBy, and
count
"""
print("\n\n\nQ1a. Interpretation 1 - Count of Products sold at each unique location:")
# Count of Products sold at each unique location
for aLine in joint_rdd.map(lambda x: (x[1][0][3], x[1][1][2])). \
        groupByKey().map(lambda x: ("\tCount of products sold at location ", x[0], ":", str(len(x[1])))).collect():
    print(" ".join(aLine))

print("\n\n\nQ1a. Interpretation 2 - Count of locations where a product is sold:")
# Count of locations where a product is sold
for aLine in joint_rdd.map(lambda x: (x[1][1][2], x[1][0][3])). \
        groupByKey().map(lambda x: ("\tCount of unique locations where product ID", x[0], "is sold:",
                            str(len(x[1])))).collect(): print(" ".join(aLine))

print("\n\n\nQ1a. Interpretation 3 - Count of Unique locations where products are sold:")
# Count of Unique location where products are sold
print("\t" + str(joint_rdd.map(lambda x: (x[1][1][2], x[1][0][3])).groupByKey().map(lambda x: (x[0],
len(x[1]))).count()))


"""
Objective b) Find out products bought by each user.

For this, we will extract userId and productId from the joint_rdd

"""

print("\n\nQ1b. Interpretation 1 - Unique UserId along with list of Product_Id they bought")
# This prints all Unique UserId along with list of Product_Id they bought
for aLine in joint_rdd.map(lambda x: (x[0], x[1][1][1])).groupByKey(). \
        map(lambda x: (int(x[0]), list(x[1]))).collect():
    print("\t"+str(aLine[0]) + "\t:\t" + ",".join(aLine[1]))
# we find that only 1 user (UserId=19) has purchased more than 1 product

print("\n\nQ1b. Interpretation 2 - We now print all unique Product_IDs along with list of UserIDs who bought the
product")
# We now print all unique Product_IDs along with list of UserIDs who bought the product
for aLine in joint_rdd.map(lambda x: (x[1][1][1], x[0])).groupByKey(). \
        map(lambda x: (x[0], list(x[1]))).collect():
    # print(aLine)
    print("\t" + str(aLine[0]) + "\t:\t" + ",".join(aLine[1]))

# we find ProductID 1004 was bought by most 5 users, Product_ID 1005 is bought by only 1 user

"""

Objective c) Total spending done by each user on each product.

For this, we will extract UserId,Product_ID and Price from joint_rdd, followed by groupByKey and then summing up
all prices
"""

print("\n\nQ1c. Interpretation 1 - Over all Spending done by each user :")
# map on joint_rdd to extract UserId:x[0] , price:x[1][1][3] and we are converting Price to Integer by int(),
# reduceByKey function of rdd spreads the operation of each value ,
# its a transformation function used to merge the values
# of each key using an associative reduce function - in this case we are using sum operation
print("\tUserID  :\t Total Spending")
for aLine in joint_rdd.map(lambda x: (x[0], int(x[1][1][3]))).reduceByKey(lambda x, y: x + y).collect():
    print("\t" + str(aLine[0]) + "\t:\t" + str(aLine[1]))


print("\n\nQ1c. Interpretation 2 - Total spending done by each user on each product :")
print("\tUserId\tProductId\tTotal Spent")
for aLine in joint_rdd.map(lambda x: ((x[0], x[1][1][1]), int(x[1][1][3]))).reduceByKey(
        lambda x, y: x + y).sortByKey().collect():
    # print(aLine)
    print("\t" + str(aLine[0][0]) + "\t" + str(aLine[0][1])+"   "+str(aLine[1]))
```

Code for Q2

```
# Importing the required libraries
from pyspark import SparkConf, SparkContext

# Setting SparkConf, setting application name and SparkContext in local variable
# these steps are optional when Apache Spark is running on same machine (from where this code is run) and at
default configs
conf = SparkConf().setAppName("BDM1-Assignment-Deepkamal_Singh-12020053-Q2")
sc = SparkContext(conf=conf)

# We dont want to see DEBUG and INFO level messages generated by PySpark or spark-submit
sc.setLogLevel("ERROR")

"""
Q2. Consider the dataset file Olympics.csv. This file contains information about the Olympic games,
    players participating in the games, and details of medals won by them. Using Spark core and the data file,
compute the following:
        a) Total medals that each country won in a particular sport (such as Gymnastics).
        b) In each Olympic games, how many medals has India won?
        c) Compute top 3 countries in terms of total medals by each Olympic games year.
    Remember, you have to make use of Spark Core for this question. (10 Marks).

Olympics.csv Header:
1) Name
2) Age
3) Country
4) Year
5) Date
6) Sport Category
7) Number of gold medals
8) Number of silver medals
9) Number of bronze medals
10) Total number of medals

"""

##################### Answer to Q2 ############################

# Loading the file `olympics.csv`
olympics_rdd = sc.textFile('olympics.csv')

print("\n\nolympics.csv is loaded,now viewing the loaded data, taking 5:")

# Viewing the loaded data
for aLine in olympics_rdd.take(5): print(aLine)

print(
    "\n\nWe find that values are loaded in single string for every entry - we will split it with separator TAB,then
viewing top 5:")
# We find that it is loaded as all values in single string for every entry - we will split it with separator
olympics_rdd = olympics_rdd.map(lambda x: x.split('\t'))
for aLine in olympics_rdd.take(5): print(aLine)

# Now our data is ready for analysis and processing

"""
a) Total medals that each country won in a particular sport (such as Gymnastics).

For this, we will be extracting Country name (2'nd index), Sport (6'th index), and total medal(9th index) through
rdd.map function
"""
print("\n\nQ2a - Total medals that each country won in a particular sport:\n")
# We are applying map on olympics_rdd and selecting Country, Sports Category and Total_Medals,
# and also we are converting Country Name, Sports Category to String (from unicode),
# the Key is set as (Sports Category,Country Name), value is Total_Medal
# then we are applying reduceByKey - which aggregates the values based on lambda function - here we are using SUM
of
# values as aggregation - thus summing up values by key, finally we are applying SortBy() on Sports name
# final output will be line separated list of
#                    <Sports Name>     <Country Name>     <Count of Medals>
print("\tSports Name\tCountry Name\tCount of Medals")
for aLine in olympics_rdd.map(lambda l: ((str(l[2]), str(l[5])), int(l[9]))).reduceByKey(lambda v1, v2: v1 + v2). \
        map(lambda l: (l[0][1], l[0][0], l[1])).sortBy(lambda l: l[0]).collect(): print("\t" +
                                                                                         aLine[0] + "\t" + aLine[
                                                                                             1] + "\t" +
str(aLine[2]))

print("\n\nQ2a - Total medals that each country won in a particular sport - Gymnastics:\n")
print("\tSports Name\tCountry Name\tCount of Medals")
# For specific Sports Category, we apply filter
for aLine in olympics_rdd.map(lambda l: ((str(l[2]), str(l[5])), int(l[9]))).reduceByKey(lambda v1, v2: v1 + v2). \
        map(lambda l: (l[0][1], l[0][0], l[1])).sortBy(lambda l: l[0]).filter(lambda l: l[0] ==
"Gymnastics").collect():
    print("\t" + aLine[0] + "\t" + aLine[1] + "\t" + str(aLine[2]))
"""
    b) In each Olympic games, how many medals has India won?
```

```
For this , we will use filter again, this time filter will be applied on Country Name
"""
print("\n\nQ2b - Interpretation 1 - Medals won by India by Sports Category")
# final output will be line separated list of
#                    <Sports Name>     <Count of Medals>
print("\tSports Name\tMedals")

for aLine in olympics_rdd.filter(lambda l: l[2] == "India").map(
        lambda l: ((str(l[2]), str(l[5])), int(l[9]))).reduceByKey(lambda v1, v2: v1 + v2). \
        map(lambda l: (l[0][0], l[0][1], l[1])).collect(): print("\t" + aLine[1] + "\t" + str(aLine[2]))

print("\n\nQ2b - Interpretation 2 - Medals won by India by Olympic Year per sports")
# final output will be line separated list of
#                    <Year>     <Sports Name>     <Count of Medals>
print("\tYear\tSports Name\tMedals")
for aLine in olympics_rdd.filter(lambda l: l[2] == "India").map(
        lambda l: ((int(l[3]), str(l[5])), int(l[9]))).reduceByKey(lambda v1, v2: v1 + v2).sortByKey().collect():
print(
        "\t" + str(aLine[0][0]) + "\t" + aLine[0][1] + "\t" + str(aLine[1]))

print("\n\nQ2b - Interpretation 3 - Medals won by India, count by Olympic Year")
# final output will be line separated list of
#                    <Year>     <Sports Name>     <Count of Medals>
print("\tYear\tMedals")
for aLine in olympics_rdd.filter(lambda l: l[2] == "India").map(
        lambda l: (int(l[3]), int(l[9]))).reduceByKey(lambda v1, v2: v1 + v2).sortByKey().collect(): print(
        "\t" + str(aLine[0]) + "\t" + str(aLine[1]))

"""
    c) Compute top 3 countries in terms of total medals by each Olympic games year.

For this we will do following steps:
    Step 1 - apply map on olympics_rdd and extract key as (Country Name, year) , and Value as Total medals
    Step 2 - apply reduceByKey to sum up total_medals based on country and year
    Step 3 - apply map to create new Key value pair - Key (Year, Medals), Value(Country name)
    Step 4 - apply sortBy on key (or sortByKey) , set ascending to false so we can get Descending order list
    Step 5 - remap and keep Year as key and Value as Medals and Country
    Step 6 - apply groupByKey - this will group descending order list of Year and Medal by Country Name
    Step 7 - Now we have RDD which has Key as year, and grouped list of country and medals
                  - we use LIST() to convert it to list,
                  - and then we access top3 values from list obtained from grouped values
                  - This will give us Top 3 Medal winners Countries for each year
"""
print("\n\nQ3c - Top 3 countries in terms of total medals by each Olympic games year\n")
#  for top 3 of every year
for aLine in olympics_rdd.map(lambda l: ((str(l[2]), int(l[3])), int(l[9]))).reduceByKey(lambda x, y: x + y). \
        map(lambda a: ((int(a[0][1]), int(a[1])), a[0][0])).sortBy(lambda l: l[0], ascending=False). \
        map(lambda l: (l[0][0], (l[1], l[0][1]))).groupByKey(). \
        map(lambda l: (l[0], list(l[1])[0:3])).collect():
    print(aLine[0])  # First printing the YEAR
    print("\t1." + aLine[1][0][0] + ":\t" + str(
        aLine[1][0][1]))  # Printing Topmost Medal Winner Country and medal count
    print("\t2." + aLine[1][1][0] + ":\t" + str(
        aLine[1][1][1]))  # Printing Second most Medal Winner Country and medal count
    print("\t3." + aLine[1][2][0] + ":\t" + str(
        aLine[1][2][1]))  # Printing Third most Medal Winner Country and medal count
```

Code for Q3

```
import sys
from math import sqrt

from pyspark import SparkConf, SparkContext

# Setting SparkConf and SparkContext in local variable
conf = SparkConf().setMaster("spark://172.16.3.2:7077").setAppName("BDM1-Assignment-Q3-Deepkamal_Singh-12020053")
sc = SparkContext(conf=conf)

# We dont want to see DEBUG and INFO level messages generated by PySpark or spark-submit
sc.setLogLevel("ERROR")


def loadMovieNames():
    movieNames = {}
    with open("itemfile.txt") as f:
        for line in f:
            fields = line.split('|')
            movieNames[int(fields[0])] = fields[1].decode('ascii', 'ignore')
    return movieNames


def makePairs((user, ratings)):
    (movie1, rating1) = ratings[0]
    (movie2, rating2) = ratings[1]
    return ((movie1, movie2), (rating1, rating2))


def filterDuplicates((userID, ratings)):
    (movie1, rating1) = ratings[0]
    (movie2, rating2) = ratings[1]
    return movie1 < movie2


'''
Method to read rating pairs of 2 movies and  calculate Pearson's correlation coefficient
'''


def computePearsonWithRatings(ratingPairs):
    # Expect a List of tuples - each tuple with 2 numeric values between 0 and 5
    # Form a series and find correlation
    sum_x = sum_y = numRecords = 0
    for xi, yi in ratingPairs:
        sum_x += xi
        sum_y += yi
        numRecords += 1
    x_bar = sum_x / numRecords
    y_bar = sum_y / numRecords
    moving_x_minus_xbar_sq = 0
    moving_y_minus_ybar_sq = 0
    numerator = 0
    for xi, yi in ratingPairs:
        x_tmp = xi - x_bar
        y_tmp = yi - y_bar
        moving_x_minus_xbar_sq += x_tmp ** 2
        moving_y_minus_ybar_sq += y_tmp ** 2
        numerator += (x_tmp * y_tmp)
    pear_coeff = 0
    if (moving_y_minus_ybar_sq != 0 and moving_x_minus_xbar_sq != 0):
        pear_coeff = numerator / sqrt(moving_x_minus_xbar_sq * moving_y_minus_ybar_sq)
    return (pear_coeff, numRecords, x_bar, y_bar)


def computeCosineSimilarityWithRatings(ratingPairs):
    # A sample ratingPair => [(5.0, 4.0), (4.0, 5.0), (3.0, 2.0), (5.0, 4.0), (3.0, 3.0), (4.0, 4.0), (2.0, 3.0)]
    # print(list(ratingPairs))
    # ratings = list(ratingPairs[1])
    numPairs = 0
    sum_xx = sum_yy = sum_xy = total_x = total_y = 0
    for ratingX, ratingY in ratingPairs:
        total_x += ratingX
        total_y += ratingY
        sum_xx += ratingX * ratingX
        sum_yy += ratingY * ratingY
        sum_xy += ratingX * ratingY
        numPairs += 1
    numerator = sum_xy
    denominator = sqrt(sum_xx) * sqrt(sum_yy)
    score = 0
    avg_ratingX = total_x / numPairs
    avg_ratingY = total_y / numPairs
    if (denominator):
        score = (numerator / (float(denominator)))
    return (score, numPairs, avg_ratingX, avg_ratingY)
```

```python
def computeCosineSimilarity(ratingPairs):
    # print(ratingPairs)
    numPairs = 0
    sum_xx = sum_yy = sum_xy = 0
    for ratingX, ratingY in ratingPairs:
        sum_xx += ratingX * ratingX
        sum_yy += ratingY * ratingY
        sum_xy += ratingX * ratingY
        numPairs += 1
    numerator = sum_xy
    denominator = sqrt(sum_xx) * sqrt(sum_yy)
    score = 0
    if (denominator):
        score = (numerator / (float(denominator)))
    return (score, numPairs)


# Save the results if desired
# moviePairSimilarities.sortByKey()
# moviePairSimilarities.saveAsTextFile("movie-sims")

def print_results(filteredResults, movieID, simAlgo, nameDict):
    results = filteredResults.map(lambda ((pair, sim)): (sim, pair)).sortByKey(ascending=False).take(10)
    print("\n\n" + simAlgo + " :: Top 10 similar movies for " + nameDict[movieID] + "\n")
    for (sim, pair) in results:
        # Display the similarity result that isn't the movie we're looking at
        # print(sim, pair)
        similarMovieID = pair[0]
        avg_rated = sim[3] if len(sim) > 2 else ""
        if (similarMovieID == movieID):
            similarMovieID = pair[1]
            avg_rated = sim[2] if len(sim) > 2 else ""
        print(nameDict[similarMovieID] + "\t\t\tscore: " + str(sim[0]) + "\tstrength: " + str(sim[1]) + (
            "\tRating:" + "{:.2f}".format(avg_rated) if len(sim) > 2 else ""))


if (len(sys.argv) > 1):
    scoreThreshold = 0.20
    coOccurenceThreshold = 2

    # Expect minimum avg user rating for good movie - if none provided set it to 3
    goodMovieRating = float(sys.argv[2]) if len(sys.argv) > 2 else 3
    movieID = int(sys.argv[1])

    print("\nmovie ID:" + str(movieID))

    print("\nLoading movie names...")
    nameDict = loadMovieNames()

    print("\nMovie names loaded, loading datafile [Ratings]")
    data = sc.textFile("datafile.txt")

    print("\nMapping ratings to key / value pairs: user ID => movie ID, rating")
    # Map ratings to key / value pairs: user ID => movie ID, rating
    ratings = data.map(lambda l: l.split('\t')).map(lambda l: (int(l[0]), (int(l[1]), float(l[2]))))

    # Emit every movie rated together by the same user.
    # Self-join to find every combination.
    print("Now applying self-join to find every combination")
    joinedRatings = ratings.join(ratings)

    # At this point our RDD consists of userID => ((movieID, rating), (movieID, rating))

    print("Now removing duplicates by applying RDD.filter and passing filterDuplicates function as lambda")
    # Filter out duplicate pairs
    uniqueJoinedRatings = joinedRatings.filter(filterDuplicates)

    # Now key by (movie1, movie2) pairs.
    moviePairs = uniqueJoinedRatings.map(makePairs)

    # We now have (movie1, movie2) => (rating1, rating2)
    # Now collect all ratings for each movie pair and compute similarity
    print("Now we have collected all ratings for each movie pair, "
          "applying group by now")

    moviePairRatings = moviePairs.groupByKey()

    # We now have (movie1, movie2) = > (rating1, rating2), (rating1, rating2) ...
    # Can now compute similarities.
    print("We now have (movie1, movie2) = > (rating1, rating2), (rating1, rating2) ..."
          " , using it we will prepare 3 type of movieSimilarities RDD")

    print("1. First with default Cosine Similarity")
    moviePairSimilarities = moviePairRatings.mapValues(computeCosineSimilarity).cache()

    print("2. Second including Avg Ratings also in preparing Movie Similarity pairs Cosine Similarity")

    movie_sim_with_ratings = moviePairRatings.mapValues(computeCosineSimilarityWithRatings).cache()
    print("3. Third with similarity score by Pearson correlation instead of Cosine Similarity")

    pearson_movies = moviePairRatings.mapValues(computePearsonWithRatings).cache()

    print("We have 3 Similarities prepared for comparison, now we will find out similar movies to MovieID:" + str(
        movieID))

    # Filter for movies with this sim
    ## Sample - (Pair: (197     , 1097), SIM: (0.9758729093599599, 7))
    filteredResultsWithoutRating = moviePairSimilarities.filter(
        lambda ((pair, sim)): (pair[0] == movieID or pair[1] == movieID) and sim[0] > scoreThreshold and sim[
            1] > coOccurenceThreshold)

    print(
        "\n\nFinding movie similarities based on Cosine, Not checking movie rating for suggestion, this takes a few
minutes...")
    print_results(filteredResultsWithoutRating, movieID, "Cosine Without Rating", nameDict)

    # Sample movie_sim_with_ratings::
    #   MovieID1 MovieId2  Score              NumPairs   AvgRating of MovieId1  AvgRating of MovieID2
    #  ((197     , 1097), (0.9758729093599599, 7,        3.7142857142857144, 3.5714285714285716))
    # suggest only if movie is of higher rating than goodMovieRating
    filteredResultsWithRating = movie_sim_with_ratings.filter(
        lambda ((pair, sim)): (pair[0] == movieID or pair[1] == movieID) and sim[0] > scoreThreshold and sim[
            1] > coOccurenceThreshold and ((pair[0] == movieID and sim[2] >= goodMovieRating) or (
                pair[1] == movieID and sim[3] >= goodMovieRating)))

    print("\n\nNow finding movie similarities based on Cosine, showing only good rating movies, "
          "having avg score above:" + str(goodMovieRating) + ", this takes a few minutes...")

    print_results(filteredResultsWithRating, movieID, "Cosine With Rating", nameDict)

    filteredResultsPearson = pearson_movies.filter(
        lambda ((pair, sim)): (pair[0] == movieID or pair[1] == movieID) and sim[0] > scoreThreshold and
                              sim[1] > coOccurenceThreshold and
                              (
                                      (pair[0] == movieID and sim[2] >= goodMovieRating) or
                                      (pair[1] == movieID and sim[3] >= goodMovieRating)
                              ))

    print("\n\nFinally finding movie similarities based on Pearson, showing only good rating movies, "
          "having avg score above:" + str(goodMovieRating) + ", this takes a few minutes...")
    print_results(filteredResultsPearson, movieID, "Pearson With Rating", nameDict)
else:
    print("Usage \n\tspark-submit BDM1_Assignment_Q3_Deepkamal_Singh.py <MovieID> [<Rating Threshold>]")
```

References

1. Spark documentation : https://spark.apache.org/docs/latest/
2. Statistical Co-relations : https://easystats.github.io/correlation/articles/types.html
3. Cosine Similarities : https://en.wikipedia.org/wiki/Cosine_similarity
4. LMS Portal – Lecture slides and videos - https://elearn.isb.edu/course/view.php?id=6522
5. LearningSpark2.0 : Jules S. Damji,Brooke Wenig,Tathagata Das & Denny Lee