databricks BDM Assignment

(https://databricks.com)

Question 1

```
fileloc = "/FileStore/tables/users.csv"
filetype ="csv"

infer_schema ="false"
firstrow_header ="false"
delimiter =","

df =
spark.read.format(filetype).option("inferSchema",infer_schema).option("header",firstrow_header).option("sep",delimiter
).load(fileloc)

display(df)


dff_user =
df.withColumnRenamed("_c0","UserID").withColumnRenamed("_c1","EmailID").withColumnRenamed("_c2","NativeLanguage").with
ColumnRenamed("_c3","Location")


dff_user.show()
```

```
fileloc = "/FileStore/tables/transactions.csv"
filetype ="csv"

infer_schema ="false"
firstrow_header ="false"
delimiter =","

df_Transaction =
spark.read.format(filetype).option("inferSchema",infer_schema).option("header",firstrow_header).option("sep",delimiter
).load(fileloc)

dff_Transaction =
df_Transaction.withColumnRenamed("_c0","Transaction_ID").withColumnRenamed("_c1","Product_ID").withColumnRenamed("_c2"
,"UserID").withColumnRenamed("_c3","Price").withColumnRenamed("_c4","Product_Description")


dff_Transaction.show()
```

```
dff_user.createOrReplaceTempView("User_Master")

sqlDF_USER = spark.sql("SELECT * FROM User_Master")
#sqlDF_USER.show()

dff_Transaction.createOrReplaceTempView("Transaction_Data")

sqlDF_Transaction = spark.sql("SELECT * FROM Transaction_Data")
sqlDF_Transaction.show()
```

Question 1a

```
sqlContext.sql("SELECT count( distinct UM.Location) as CountLocation FROM Transaction_Data TD Inner join User_Master
UM on TD.UserID = UM.USERID ").show()
```

Question 1b

```
sqlContext.sql("SELECT UserID as User ,Product_Description FROM Transaction_Data  order by cast(UserID as int) asc
").show()
```

Question 1C

```
# 1C

sqlContext.sql("SELECT UserID as User ,Product_Description , sum(Price) as Price_perProduct FROM Transaction_Data
group by UserID ,Product_Description   order by cast(UserID as int) asc ").show()
```

Question 2

Our motive is to recommend similar rated movies to each user.

```
import sys
from pyspark import SparkConf, SparkContext, SparkFiles
from math import sqrt
import os

#importing necessary python and pySpark modules such as sys (to access various functions and variables used in Python
runtime environment) , sparkConf ( Configuration for a Spark Application) ,SparkContext(Main Entry point for Spark
functionality) , SparkFiles( resolves path to files added through sc.addFile() ) , math ( to access square root
function from python math module ) , os (module that provides function to interact with operating system) for the
script to execute.
```

```
# This is  a user defined function which provides functionality to list movienames and its ID.

def loadMovieNames(): # Initialization or defination of function
    movieNames = {}  # Creation of an empty dictionary names movieNames
    spark.sparkContext.addFile("dbfs:/FileStore/shared_uploads/sbsnigdhac@gmail.com/itemfile.txt") # Using
SparkContext we are adding a file in Spark cluster
    with open(SparkFiles.get('itemfile.txt'),encoding="ISO-8859-1") as f: # Accessing the file in Spark Job using
SparkFile.get() and is being assigned to f
        for line in f:  # for loop is used to access each element in f and getting assigned to line variable.
            fields = line.split('|') # spliting each element of variable line using the pipe delimiter "|" and adding
it to the list fields.
            movieNames[int(fields[0])] = fields[1] # now movie id (0th index of list fields ) and movie name (1st
index of list fields) are being added into the dictionary as key (movie id) - value (movie name)
    return movieNames # this is the return statement of this function, which returns the dictionary moviewNames
consisting of movie id and movie name .

loadMovieNames() # calling the function to check the output
```

```
Out[43]: {1: 'Toy Story (1995)',
 2: 'GoldenEye (1995)',
 3: 'Four Rooms (1995)',
 4: 'Get Shorty (1995)',
 5: 'Copycat (1995)',
 6: 'Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)',
 7: 'Twelve Monkeys (1995)',
 8: 'Babe (1995)',
 9: 'Dead Man Walking (1995)',
 10: 'Richard III (1995)',
 11: 'Seven (Se7en) (1995)',
 12: 'Usual Suspects, The (1995)',
 13: 'Mighty Aphrodite (1995)',
 14: 'Postino, Il (1994)',
 15: "Mr. Holland's Opus (1995)",
 16: 'French Twist (Gazon maudit) (1995)',
 17: 'From Dusk Till Dawn (1996)',
 18: 'White Balloon, The (1995)',
 19: "Antonia's Line (1995)",
 20: 'Angels and Insects (1995)',
 21: 'Muppet Treasure Island (1996)',
```

```
# The defination of this function is to provide similiarities between pair of movies based on the ratings by each user.

def makePairs(user_ratings): # Initialization or a defination of a function . This is a parameterized function, which takes tuples as a parameter
    user, ratings=user_ratings #this line assigns the 1st element of the tuple to user variable and second element of the parameter tuple to ratings variable. By unpacking tuple, function has created created user ID and ratings list.
    (movie1, rating1) = ratings[0] # The ratings list contains two elements or tuple each providing movie name/movie ID and ratings of respective movie again by using tuple unpacking.
    (movie2, rating2) = ratings[1] # similarly the second element or tuple of the ratings list is assigned to movie2 and rating2 .
    return ((movie1, movie2), (rating1, rating2)) # finally it is returning the obtained pair of movie and ratings .

# To compute cosine similarity metrics on pair of movies , this functio has created tuple of movie pair and of their ratings.
# This user defined tuple parameterized function is used to filter out duplicates from the combinations created in the
above function code. the defination of the function is based on the logic to remove combinations such as (movie1,
movie1) , (movie2, movie2) and (movie2, movie1).

def filterDuplicates(userID_ratings): # This is the initialization or defination step of the function. Its a
parameterized function that accepts tuple as a parameter.
    userID, ratings = userID_ratings # this line unfolds the parameter tuple containing information about the userid
and ratings. It assign userid to the userid variable and ratings provided by the user to the ratings list
    (movie1, rating1) = ratings[0] # The ratings list contains two tuples each containg movieid and ratings , which
inturn is getting assigned to the tuple .
    (movie2, rating2) = ratings[1] # assigning second element of the ratings list to this tuple.
    return movie1 < movie2 # As the purpose of the function was to filter out the duplicates [(movie2, movie1) =
(movie1, movie2) , (movie1, movie1) , (movie2, movie2) ] , it is returning those combinations satisfying movie1 id <
movie2 id.
```

```
# The purpose of the function is to calculate cosine similarity score between two movies on the basis on ratings
provided by users.

def computeCosineSimilarity(ratingPairs): # This is the function defination and this functiona accepts tuple as
parameter. The paremeter is the two ratings Rating A and B given by two users for the same movie.
    numPairs = 0 # Initializing variable to zero
    sum_xx = sum_yy = sum_xy = 0 # Initializing variable to zero
    for ratingX, ratingY in ratingPairs: # This iterates the over the pair of ratings provided by the function
parameter
        sum_xx += ratingX * ratingX # It cumulatively adds square of one rating and assign to the variable.
        sum_yy += ratingY * ratingY # It cumulatively adds square of another rating and assign to the variable.
        sum_xy += ratingX * ratingY # It cumulatively adds  the product of both ratings and assign to the variable.
        numPairs += 1 # updates the number of pairs in each iteration.
    numerator = sum_xy # To calculate the cosine similarity score numerator (sum_xy) / denominator (sqrt (sum_xx)*
sqrt (sum_yy)) , it is defining numerator
    denominator = sqrt(sum_xx) * sqrt(sum_yy) # similarly defining denominator
    score = 0  # inittilaizing the score = 0
    if (denominator):
        score = (numerator / (float(denominator))) # This line says if denominator <> 0 then calculate this block.
    return (score, numPairs) # this returns the cosine similarity score and number of user pairs contributed in the
score
```

```
Loading movie names...
```

```
org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage 9.0 failed 1 times, most recent f
ailure: Lost task 0.0 in stage 9.0 (TID 18) (ip-10-172-193-42.us-west-2.compute.internal executor driver): org.apach
e.spark.api.python.PythonException: 'TypeError: filterDuplicates() missing 1 required positional argument: 'rating
s''. Full traceback below:
```