```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 20 12:55:24 2020

@author: anuju
"""
import os
os.environ['JAVA_HOME'] = 'C:/Program Files/Java/jre1.8.0_241'
os.environ['PYSPARK_SUBMIT_ARGS'] = "--master local[2] pyspark-shell"


# In[1]:

#Importing Necessary Libraries
from pyspark.sql import SparkSession
import numpy as np
import pandas as pd
from pyspark.sql.types import IntegerType, StructType, StructField, DateType, StringType
from pyspark.sql.functions import date_format, monotonically_increasing_id, row_number, when, col
import matplotlib.pyplot as plt
from collections import OrderedDict
from wordcloud import WordCloud
from pyspark.sql.window import Window
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
import seaborn as sns
from sklearn.metrics import confusion_matrix
from pyspark.sql.functions import lit


# In[2]:

# Building Spark Session
spark = SparkSession.builder.appName('collaborative filtering').getOrCreate()
SparkContext = spark.sparkContext
SparkContext.setSystemProperty('spark.driver.memory', '32g')
spark.conf.set("spark.sql.execution.arrow.enabled", True)
spark.conf.set("spark.sql.shuffle.partitions", 50)
spark.conf.set("spark.shuffle.memoryFraction","0.65")
spark.conf.set("spark.storage.memoryFraction","0.3")


# In[3]:

#Creating new text files from the given Customer data with 4 columns(MovieID, CustomerID, Rating, TimeStamp)
def netflix_file_massager(file_to_read, file_out_name):
    fout = open(file_out_name, "w")

    with open(file_to_read) as fp:
       line = fp.readline()

       while line:
           #print("{}: {}".format(cnt, line.strip()))

           if ':' in line:
               movie_id = line[0:-2]
               #print ("Movie ID", movie_id)


           line = fp.readline()
```

```
            if ':' in line:
                continue

            if (line):
                line_out = movie_id + "," + line
                #print (line_out)
                fout.write(line_out)

    fp.close()
    fout.close()


ratings1_raw = 'C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_1.txt'
ratings1_processed = 'C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_1_Processed.txt'

ratings2_raw = 'C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_2.txt'
ratings2_processed = 'C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_2_Processed.txt'

ratings3_raw = 'C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_3.txt'
ratings3_processed = 'C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_3_Processed.txt'

ratings4_raw = 'C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_4.txt'
ratings4_processed = 'C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_4_Processed.txt'

netflix_file_massager( ratings1_raw,ratings1_processed)
netflix_file_massager( ratings2_raw,ratings2_processed)
netflix_file_massager( ratings3_raw,ratings3_processed)
netflix_file_massager( ratings4_raw,ratings4_processed)

# In[4]
#Reading the Proessed Data Files to form spark dataframes
## Custom Schema for Customer Rating Data Reading

CustomerDBSchema = StructType([
    StructField("movie_id", IntegerType()),
    StructField("Customer_id", IntegerType()),
    StructField("Rating", IntegerType()),
    StructField("timestamp", DateType())
])

## Custom Schema for Movie Titles Data Reading
MovieDBSchema = StructType([
    StructField("movie_id", IntegerType()),
    StructField("Release_Year", IntegerType()),
    StructField("Movie_Title", StringType())
])

## Custom Schema for most frequently rated Movies
MovieRatingDBSchema = StructType([
    StructField("Rating_1", IntegerType()),
    StructField("Rating_2", IntegerType()),
    StructField("Rating_3", IntegerType()),
    StructField("Rating_4", IntegerType()),
    StructField("Rating_5", IntegerType())
])



# Loading preprocessed Customer data
ratings1 = spark.read.csv("C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_1_Processed.txt",
```

```
                           nanValue= True, sep = ',', schema=CustomerDBSchema)
ratings2 = spark.read.csv("C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_2_Processed.txt",
                           nanValue= True, sep = ',', schema=CustomerDBSchema)
ratings3 = spark.read.csv("C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_3_Processed.txt",
                           nanValue= True, sep = ',', schema=CustomerDBSchema)
ratings4 = spark.read.csv("C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/combined_data_4_Processed.txt",
                           nanValue= True, sep = ',', schema=CustomerDBSchema)


############Join all the 4 customer rating files:####################
temp = ratings1.union(ratings2)
temp = temp.union(ratings3)
customerdata_sp_df = temp.union(ratings4)
customerdata_sp_df.show()

""" Since the model crashes withlarge dataset, instead of combined customer data file, We tried using one file at a time,
and will compare the performance in each case"""

"""
+--------+-----------+------+----------+
|movie_id|Customer_id|Rating| timestamp|
+--------+-----------+------+----------+
|       1|    1488844|     3|2005-09-06|
|       1|     822109|     5|2005-05-13|
|       1|     885013|     4|2005-10-19|
|       1|      30878|     4|2005-12-26|
|       1|     823519|     3|2004-05-03|
|       1|     893988|     3|2005-11-17|
|       1|     124105|     4|2004-08-05|
|       1|    1248029|     3|2004-04-22|
|       1|    1842128|     4|2004-05-09|
|       1|    2238063|     3|2005-05-11|
|       1|    1503895|     4|2005-05-19|
|       1|    2207774|     5|2005-06-06|
|       1|    2590061|     3|2004-08-12|
|       1|       2442|     3|2004-04-14|
|       1|     543865|     4|2004-05-28|
|       1|    1209119|     4|2004-03-23|
|       1|     804919|     4|2004-06-10|
|       1|    1086807|     3|2004-12-28|
|       1|    1711859|     4|2005-05-08|
|       1|     372233|     5|2005-11-23|
+--------+-----------+------+----------+
only showing top 20 rows"""

# Loading preprocessed Movie data

movietitles_sp_df = spark.read.csv("C:/Users/anuju/Desktop/Group9_IoT_FinalProject/NetflixPrizeData/movie_titles.csv",
                                   nanValue= True, sep = ',', schema=MovieDBSchema)
movietitles_sp_df.show()

"""
+--------+------------+--------------------+
|movie_id|Release_Year|         Movie_Title|
+--------+------------+--------------------+
|       1|        2003|     Dinosaur Planet|
|       2|        2004|Isle of Man TT 20...|
|       3|        1997|           Character|
|       4|        1994|Paula Abdul's Get...|
|       5|        2004|The Rise and Fall...|
|       6|        1997|                Sick|
```

```
|       7|        1992|              8 Man|
|       8|        2004|What the #$*! Do ...|
|       9|        1991|Class of Nuke 'Em...|
|      10|        2001|             Fighter|
|      11|        1999|Full Frame: Docum...|
|      12|        1947|My Favorite Brunette|
|      13|        2003|Lord of the Rings...|
|      14|        1982|  Nature: Antarctica|
|      15|        1988|Neil Diamond: Gre...|
|      16|        1996|           Screamers|
|      17|        2005|           7 Seconds|
|      18|        1994|    Immortal Beloved|
|      19|        2000|By Dawn's Early L...|
|      20|        1972|     Seeta Aur Geeta|
+--------+------------+--------------------+
only showing top 20 rows
"""


# In[4]:

### Analysis of the data and data joining with Movie DB
## Joining the Movies and the Customer Databases

##Complete Customer Dataset Joining
CombinedData= customerdata_sp_df.join(movietitles_sp_df,on=['movie_id'], how='inner')
CombinedData.show()

"""
+--------+-----------+------+----------+------------+---------------+
|movie_id|Customer_id|Rating| timestamp|Release_Year|    Movie_Title|
+--------+-----------+------+----------+------------+---------------+
|       1|    1488844|     3|2005-09-06|        2003|Dinosaur Planet|
|       1|     822109|     5|2005-05-13|        2003|Dinosaur Planet|
|       1|     885013|     4|2005-10-19|        2003|Dinosaur Planet|
|       1|      30878|     4|2005-12-26|        2003|Dinosaur Planet|
|       1|     823519|     3|2004-05-03|        2003|Dinosaur Planet|
|       1|     893988|     3|2005-11-17|        2003|Dinosaur Planet|
|       1|     124105|     4|2004-08-05|        2003|Dinosaur Planet|
|       1|    1248029|     3|2004-04-22|        2003|Dinosaur Planet|
|       1|    1842128|     4|2004-05-09|        2003|Dinosaur Planet|
|       1|    2238063|     3|2005-05-11|        2003|Dinosaur Planet|
|       1|    1503895|     4|2005-05-19|        2003|Dinosaur Planet|
|       1|    2207774|     5|2005-06-06|        2003|Dinosaur Planet|
|       1|    2590061|     3|2004-08-12|        2003|Dinosaur Planet|
|       1|       2442|     3|2004-04-14|        2003|Dinosaur Planet|
|       1|     543865|     4|2004-05-28|        2003|Dinosaur Planet|
|       1|    1209119|     4|2004-03-23|        2003|Dinosaur Planet|
|       1|     804919|     4|2004-06-10|        2003|Dinosaur Planet|
|       1|    1086807|     3|2004-12-28|        2003|Dinosaur Planet|
|       1|    1711859|     4|2005-05-08|        2003|Dinosaur Planet|
|       1|     372233|     5|2005-11-23|        2003|Dinosaur Planet|
+--------+-----------+------+----------+------------+---------------+
only showing top 20 rows
"""

CombinedData.describe(['Rating']).show()
"""
+-------+------------------+
|summary|            Rating|
+-------+------------------+
```

```
|  count|         100480507|
|   mean| 3.604289964420661|
| stddev|1.0852185646295671|
|    min|                 1|
|    max|                 5|
+-------+------------------+
"""
##Joining Movie Titles dataframe with Customer Dataset 1
CombinedData_1= ratings1.join(movietitles_sp_df,on=['movie_id'], how='inner')
CombinedData_1.show()

##Joining Movie Titles dataframe with Customer Dataset 2
CombinedData_2= ratings2.join(movietitles_sp_df,on=['movie_id'], how='inner')
CombinedData_2.show()

##Joining Movie Titles dataframe with Customer Dataset 3
CombinedData_3= ratings3.join(movietitles_sp_df,on=['movie_id'], how='inner')
CombinedData_3.show()

##Joining Movie Titles dataframe with Customer Dataset 4
CombinedData_4= ratings4.join(movietitles_sp_df,on=['movie_id'], how='inner')
CombinedData_4.show()
# In[5]:

# ANALYZING DATA

## The RatingsperWeekday and RatingsperMonth graphs were created on single dataset, not the combined dataset

# Drawing WeekDay versus Number of Customers Rating to search for any relationship
# Drawing MonthOfYear versus Number of Customers also
Data = CombinedData_1

Day_Month = Data.select('timestamp', date_format('timestamp','E').alias('WeekDay'),
                                        date_format('timestamp','MMMM').alias('MonthOfYear'))
#Day_Month.show()

#add 'sequential' DUMMY index and join both dataframeS to get the final result
Data = Data.withColumn("row_idx", row_number().over(Window.orderBy(monotonically_increasing_id())))
Day_Month = Day_Month.withColumn("row_idx", row_number().over(Window.orderBy(monotonically_increasing_id())))

# after joining there will be 2 timestamp columns, renaming one and deleting it.
Data = Data.withColumnRenamed("timestamp", "timeDELETE")
Data = Data.join(Day_Month, on=['row_idx'], how='inner').drop('row_idx')
Data = Data.drop('timeDELETE')

# Create a TEMP view for running SQL Queries in PySPARK Session (for Ratings per WeekDay and Ratings per month )
Data.createOrReplaceTempView("RatingsPerWeekMonthView")
RatingsPerWeekDay = spark.sql ("SELECT WeekDay, COUNT(*) AS TotalRatingsPerWeekDay FROM \
                               RatingsPerWeekMonthView GROUP BY WeekDay \
                               ORDER BY \
                                   case WeekDay \
                                       when 'Mon' then 1 \
                                       when 'Tue' then 2 \
                                       when 'Wed' then 3 \
                                       when 'Thu' then 4 \
                                       when 'Fri' then 5 \
                                       when 'Sat' then 6 \
                                       when 'Sun' then 7 end")
RatingsPerWeekDay.show()
```

```
RatingsPerMonth = spark.sql ("SELECT MonthOfYear, COUNT(*) AS TotalRatingsPerMonth FROM \
                              RatingsPerWeekMonthView GROUP BY MonthOfYear \
                                ORDER BY \
                                    case MonthOfYear \
                                        when 'January' then 1 \
                                        when 'February' then 2 \
                                        when 'March' then 3 \
                                        when 'April' then 4 \
                                        when 'May' then 5 \
                                        when 'June' then 6 \
                                        when 'July' then 7 \
                                        when 'August' then 8 \
                                        when 'September' then 9 \
                                        when 'October' then 10 \
                                        when 'November' then 11 \
                                        when 'December' then 12 end")
RatingsPerMonth.show()

# Plot Total Ratings per Week Day to see customer rating behavior
#RatingsPerWeekDay_pd = RatingsPerWeekDay.toPandas()

RatingsPerWeekDay_pd= RatingsPerWeekDay.toPandas()
RatingsPerWeekDay_pd.plot(x='WeekDay', y = 'TotalRatingsPerWeekDay', kind = "bar")
plt.show()

# Plot Total Ratings per Month to see customer rating behavior
RatingsPerMonth_pd = RatingsPerMonth.toPandas()
RatingsPerMonth_pd.plot(x='MonthOfYear', y = 'TotalRatingsPerMonth', kind = "bar")
plt.show()
```

```
# In[6]:
###The Top Movie WordClouds are created on the Complete Dataset
###Creating WordCloud for Top 50 movies rated 1, 2, 3, 4 and 5.

CombinedData.createOrReplaceTempView("Netflix_TempView")
TopMovieCountWordCloud = 50

for iter in range(1, 6, 1):
    MovieRatingQuery = "SELECT Movie_Title, count(*) \
                        FROM Netflix_TempView where Rating == " + str(iter) + " group By Movie_Title"

   # print (SQLQuery)
    RatedMovies_sp_df= spark.sql(MovieRatingQuery)

    RatedMovieTitles_RowList= RatedMovies_sp_df.select("Movie_Title").collect()
    RatedMovieTitlesCounts_RowList= RatedMovies_sp_df.select("count(1)").collect()

    RatedMovieTitles_List = [str(row['Movie_Title']) for row in RatedMovieTitles_RowList]
    RatedMovieTitlesCounts_List = [int(row['count(1)']) for row in RatedMovieTitlesCounts_RowList]

    RatedMovies_dict= dict(zip(RatedMovieTitles_List, RatedMovieTitlesCounts_List))

    # Let us sort the freq dictionary
    RatedMovies_dict_sorted = OrderedDict(sorted(RatedMovies_dict.items(),
                                                 key=lambda x: x[1], reverse=True)[:TopMovieCountWordCloud])

    # make HD image of WordCloud
    wordcloud = WordCloud(collocations=False, max_font_size=300,
                          colormap='copper', background_color='white',
```

```
                          width=1920, height=1200 ).generate_from_frequencies(RatedMovies_dict_sorted)
    plt.figure()
    PlotTitle = "[ Top " + str(TopMovieCountWordCloud) + " movies with customer rating of " + str(iter) + " ]"
    plt.title(PlotTitle, fontname="Trebuchet MS", fontsize=50, color="red", pad=50)
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.show()


# In[7]:

# The Top 10 Most Frequently Rated Movies and their Rating Distribution Barplot is created on the complete dataset.
# Creating Multiple bar plot to see rating distribution for top 10 most frequently rated movies of all times.
###Frequently Rated Movie Segregation Plot (10 most frequently rated movies)

Top10TotalRatings = spark.sql("SELECT Movie_Title, count(*) \
                              AS TotalRatings FROM Netflix_TempView group By Movie_Title order by 2 DESC LIMIT 10")
Top10TotalRatings.show()

"""
+--------------------+------------+
|         Movie_Title|TotalRatings|
+--------------------+------------+
|   Miss Congeniality|      232944|
|    Independence Day|      216596|
|         The Patriot|      211764|
|       The Godfather|      206551|
|The Day After Tom...|      196397|
|Pirates of the Ca...|      193941|
|        Pretty Woman|      193295|
|             Twister|      187808|
|  Gone in 60 Seconds|      183590|
|        Forrest Gump|      181508|
+--------------------+------------+
"""

CombRatings_df = Top10TotalRatings
TopMovieCount = 10

# Let us make multiple bar plot using
for RatingIter in range(1, 6, 1):

    Rating_List = []

    for iter in range(0, TopMovieCount, 1):

        MovieLookup = Top10TotalRatings.take(TopMovieCount)[iter][0]

        RatingQuery = "SELECT count(*) AS RatingsCount FROM Netflix_TempView where \
                       Movie_Title == \"" + str(MovieLookup) + "\" and Rating == " + str(RatingIter)
        print (RatingQuery)

        RatingQueryOut = spark.sql(RatingQuery)
        RatingQueryOut_RowList= RatingQueryOut.select("RatingsCount").collect()
        Rating_List.append(RatingQueryOut_RowList[0].RatingsCount)

    # Get the Rating list and add it to its corresponding column in the Rating PySPARK data frame
    RatingList_df = spark.createDataFrame(Rating_List, IntegerType())
    RatingList_df = RatingList_df.withColumnRenamed("value", "Rating_" + str(RatingIter))

    #add 'sequential' DUMMY index and join both dataframe to get the final result
```

```
    RatingList_df = RatingList_df.withColumn("row_idx",
                                             row_number().over(Window.orderBy(monotonically_increasing_id())))
    CombRatings_df = CombRatings_df.withColumn("row_idx",
                                               row_number().over(Window.orderBy(monotonically_increasing_id())))

    CombRatings_df = CombRatings_df.join(RatingList_df, on=['row_idx'], how='inner').drop('row_idx')

CombRatings_df_pd = CombRatings_df.toPandas()
CombRatings_df_pd.plot(x='Movie_Title',
                       y=['Rating_1', 'Rating_2', 'Rating_3', 'Rating_4', 'Rating_5' ], kind="bar" )
plt.show()

# In[8]:

# How the rating is spread
RatingSpreadQuery = "SELECT Rating, Count(*) AS TotalCount FROM \
                     Netflix_TempView group By Rating Order By Rating"
Rating_Spread = spark.sql(RatingSpreadQuery)
Rating_Spread.show()

"""
+------+----------+
|Rating|TotalCount|
+------+----------+
|     1|   4617990|
|     2|  10132080|
|     3|  28811247|
|     4|  33750958|
|     5|  23168232|
+------+----------+
"""
spread = Rating_Spread.collect()
#create a numeric value for every label
indexes = list(range(len(spread)))

#split words and counts to different lists
values = [r['TotalCount'] for r in spread]
labels = [r['Rating'] for r in spread]

Total_Ratings= sum(values)
values_percent = [(i/Total_Ratings)*100 for i in values]

# Plot it
fig1, ax1 = plt.subplots()
ax1.pie(values_percent, labels=labels, autopct='%1.1f%%',shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()

""" We Notice that the majority of the movies were rated 4 """



# In[10]:

## Dividing the data into training and test set
(training, test) = CombinedData_1.randomSplit([0.8, 0.2], seed=12345)
training.describe().show()
test.describe().show()
"""
+-------+------------------+------------------+------------------+------------------+--------------------+
|summary|          movie_id|       Customer_id|            Rating|      Release_Year|         Movie_Title|
```

```
+-------+------------------+------------------+------------------+------------------+------------------+
|  count|          19243476|          19243476|          19243476|          19243476|          19243476|
|   mean| 2308.328942338692|1322358.9873269259|3.5997748535659566| 1994.406017551091|           Infinity|
| stddev|1304.0138914191778|  764581.037489603| 1.086121398204337|12.598343040295761|               NaN|
|    min|                 1|                 6|                 1|              1915|'N Sync: 'N the Mix|
|    max|              4499|           2649429|                 5|              2005|           s-Cry-ed|
+-------+------------------+------------------+------------------+------------------+------------------+

+-------+------------------+------------------+------------------+------------------+------------------+
|summary|          movie_id|       Customer_id|            Rating|      Release_Year|       Movie_Title|
+-------+------------------+------------------+------------------+------------------+------------------+
|  count|           4810288|           4810288|           4810288|           4810288|           4810288|
|   mean|2308.3037520830353|1321990.7265673075| 3.599072030614383|1994.4039521126385|           Infinity|
| stddev| 1303.490951113763| 764565.5373457455|1.0861053257733961|12.602776332955534|               NaN|
|    min|                 1|                 6|                 1|              1915|'N Sync: 'N the Mix|
|    max|              4499|           2649429|                 5|              2005|           s-Cry-ed|
+-------+------------------+------------------+------------------+------------------+------------------+
"""
```

```
# In[11]:

## Building the Recommendation Model (ALS) using Collaborative Filtering
# using coldStartStrategy= drop, to avoid NaN value of RMSE
Recommendation = ALS(userCol="Customer_id", itemCol="movie_id", ratingCol="Rating", coldStartStrategy="drop")
Recommendation_model = Recommendation.fit(training)

# In[12]:

##Evaluating the model
predictions = Recommendation_model.transform(test)
predictions.select('Rating', 'prediction').show()

evaluator = RegressionEvaluator(metricName="rmse", labelCol="Rating", predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("")
print("Root-mean-square error (RMSE) = " + str(rmse))
print("")


# In[14]:

# Confusion Matrix (let us PANDA the Rating column)
y_true = predictions.select('Rating').toPandas()

# Since ALS predictions can go between 0 and 6, we will clamp them using SQL
# PySpark doesn't have UPDATE SQL Query, how terrible is this insane world, well its FREE !
predictions = predictions.withColumn('prediction_New',when(predictions.prediction > 5, 5)
                                     .otherwise(predictions.prediction)).drop(predictions.prediction).select(col('prediction_New')
                                    .alias('prediction'),col('Rating'))
predictions = predictions.withColumn('prediction_New',when(predictions.prediction < 1, 1)
                                     .otherwise(predictions.prediction)).drop(predictions.prediction).select(col('prediction_New')
                                    .alias('prediction'),col('Rating'))

# (let us PANDA the ALS prediction column)
y_pred = predictions.select('prediction').toPandas()

# convert floating point of ALS prediction to int types now
y_pred['prediction'] = y_pred['prediction'].astype(int)

# Confusion Matrix calc
confMatrix = confusion_matrix(y_true, y_pred)
```

```
print ('Confusion Matrix: ', confMatrix)
print("Prediction Accuracy is ", (confMatrix[0,0]+confMatrix[1,1]+confMatrix[2,2]+confMatrix[3,3]+confMatrix[4,4])/(confMatrix.sum()) )

# Let us Seaborn confusion Matrix for ratings between [1,5]
xticklabels = np.arange(1,6,1)
fig, ax = plt.subplots()
ax.xaxis.set_ticks_position('top')
ax.xaxis.set_label_position('top')
sns.heatmap(confMatrix, xticklabels=xticklabels, yticklabels=xticklabels, fmt='d', annot=True)
plt.xlabel("PREDICTED Ratings")
plt.ylabel("ACTUAL Ratings")


# In[15]:

##Visualising the Outcomes:
predictions_pd = predictions.toPandas()

df = pd.DataFrame({'RealScore': predictions_pd['Rating'], 'Predicted': predictions_pd['prediction']})
sns.violinplot(x="RealScore", y="Predicted", data=df)


# In[16]:
## Create a Recommender Function
# Recommend movies to a particular user

def recommendMovies(model, user, nbRecommendations):
    # Create a Spark DataFrame with the specified user and all the movies listed in the ratings DataFrame
    dataSet = CombinedData.select('movie_id').distinct().withColumn('Customer_id', lit(user))

    # Create a Spark DataFrame with the movies that have already been rated by this user
    moviesAlreadyRated = CombinedData.filter(CombinedData.Customer_id == user).select('movie_id', 'Customer_id')

    # Apply the recommender system to the data set without the already rated movies to predict ratings
    predictions = model.transform(dataSet.subtract(moviesAlreadyRated)).dropna().orderBy('prediction',
                                  ascending=False).limit(nbRecommendations).select('movie_id', 'prediction')

    # Join with the movies DataFrame to get the movies titles and genres
    recommendations = predictions.join(movietitles_sp_df,
                                    predictions.movie_id == movietitles_sp_df.movie_id).select(predictions.movie_id,
                                     movietitles_sp_df.Movie_Title, predictions.prediction)

    return recommendations

# Now let us see how the model recommends
ALSRecommends = recommendMovies(Recommendation_model, 2442, 10)
print('10 Movies recommended for Customer ID 2442 : ')
ALSRecommends.show(truncate=False)

"""
10 Movies recommended for Customer ID 2442 :
+--------+----------------------------------------+----------+
|movie_id|Movie_Title                             |prediction|
+--------+----------------------------------------+----------+
|76      |I Love Lucy: Season 2                   |4.219988  |
|1072    |As Time Goes By: Series 8               |4.4039474 |
|1947    |Gilmore Girls: Season 3                 |4.296383  |
|2162    |CSI: Season 1                           |4.225175  |
|3005    |As Time Goes By: Series 1 and 2         |4.231754  |
|3180    |Thin Man Collection: Alias Nick and Nora|4.2234507 |
|3456    |Lost: Season 1                          |4.31278   |
```

```
|3962    |Finding Nemo (Widescreen)               |4.2510896 |
|4233    |Little House on the Prairie: Season 9   |4.255509  |
|4294    |Ghost Hunters: Season 1                 |4.3872037 |
+--------+----------------------------------------+----------+
"""
```