DS/CMPSC 410 Sparing 2021

Instructor: Professor John Yen

TA: Rupesh Prajapati and Dongkuan Xu

Lab 4: Data Frames, Join, and Spark Submit

The goals of this lab are for you to be able to

- Use Data Frames in Spark for Processing Structured Data

- Perform Join on DataFrames

- Transfer files from ICDS to XSEDE using Globus Connect

- Run Spark-submit (Cluster Mode) in XSEDE

- Apply the obove to find Movies that have the largest number of Netflix movie reviews

Total Number of Exercises:

- Exercise 1: 5 points
- Exercise 2: 5 points
- Exercise 3: 5 points
- Exercise 4: 10 points
- Exercise 5: 5 points
- Exercise 6: 5 points
- Exercise 7: 5 points
- Exercise 8: 20 points ## Total Points: 60 points

Due: midnight, February 14, 2021

The first thing we need to do in each Jupyter Notebook running pyspark is to import pyspark first.

In [3]:
```python
import pyspark
```

Once we import pyspark, we need to import "SparkContext". Every spark program needs a SparkContext object

In order to use Spark SQL on DataFrames, we also need to import SparkSession from PySpark.SQL

In [4]:
```python
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.types import StructField, StructType, StringType, LongType, IntegerType, FloatType
from pyspark.sql.functions import col, column
from pyspark.sql.functions import expr
from pyspark.sql.functions import split
from pyspark.sql import Row
# from pyspark.ml import Pipeline
# from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler, IndexToString
# from pyspark.ml.clustering import KMeans
```

We then create a Spark Session variable (rather than Spark Context) in order to use DataFrame.

- Note: We temporarily use "local" as the parameter for master in this notebook so that we can test it in ICDS Roar. However, we need to change "local" to "Yarn" before we submit it to XSEDE to run in cluster mode.

In [5]:
```python
ss=SparkSession.builder.master("local").appName("lab4").getOrCreate()
```

Exercise 1 (5 points) (a) Add your name below AND (b) replace the path below with the path of your home directory.

Answer for Exercise 1

- a: Your Name: Kangdong Yuan

In [6]:
```python
movies_DF = ss.read.csv("/storage/home/kky5082/ds410/lab4/movies_2.csv", header=True, inferSchema=True)
```

In [7]:
```python
movies_DF.printSchema()
```

```
root
 |-- MovieID: integer (nullable = true)
 |-- MovieTitle: string (nullable = true)
 |-- Genres: string (nullable = true)
```

In [8]:
```python
movies_DF.first()
```

Out[8]:
```
Row(MovieID=1, MovieTitle='Toy Story (1995)', Genres='Adventure|Animation|Children|Comedy|Fantasy')
```

In [9]:
```python
ratings_DF = ss.read.csv("/storage/home/kky5082/ds410/lab4/ratings_2.csv", header=True, inferSchema=True)
```

In [10]:
```python
ratings_DF.printSchema()
```

```
root
 |-- UserID: integer (nullable = true)
 |-- MovieID: integer (nullable = true)
 |-- Rating: double (nullable = true)
 |-- RatingID: integer (nullable = true)
```

In [20]:
```python
ratings_DF.first()
```

Out[20]:
```
Row(UserID=1, MovieID=31, Rating=2.5, RatingID=1260759144)
```

2. DataFrames Transformations

DataFrame in Spark provides higher-level transformations that are convenient for selecting rows, columns, and for creating new columns. These transformations are part of Spark SQL.

2.1 Select Transformation

Select columns from a DataFrame.

In [21]:
```python
movies_DF.select("Genres").show(5)
```

```
+--------------------+
|              Genres|
+--------------------+
|Adventure|Animati...|
|Adventure|Childre...|
|      Comedy|Romance|
|Comedy|Drama|Romance|
|              Comedy|
+--------------------+
only showing top 5 rows
```

In [22]:
```python
# Step 1: Convert the DataFrame movies_DF into an RDD
from pyspark.sql.functions import split
movies_RDD = movies_DF.rdd
movies_RDD.take(2)
```

Out[22]:
```
[Row(MovieID=1, MovieTitle='Toy Story (1995)', Genres='Adventure|Animation|Children|Comedy|Fantasy'),
 Row(MovieID=2, MovieTitle='Jumanji (1995)', Genres='Adventure|Children|Fantasy')]
```

In [23]:
```python
# Step 2: Map to each row of the DF-converted RDD to extract the column "Generes". Save the mapping result
# in a new RDD (whih contains only values of the column)
Genres_column = movies_RDD.map(lambda row: row.Genres)
Genres_column.take(2)
```

Out[23]:
```
['Adventure|Animation|Children|Comedy|Fantasy', 'Adventure|Children|Fantasy']
```

In [24]:
```python
# Step 3: Split the multiple Generes of a movie (separated by |) into a tuple.
Genres_list_rdd= Genres_column.flatMap(lambda string: string.split("|"))
Genres_list_rdd.take(10)
```

Out[24]:
```
['Adventure',
 'Animation',
 'Children',
 'Comedy',
 'Fantasy',
 'Adventure',
 'Children',
 'Fantasy',
 'Comedy',
 'Romance']
```

Exercise 2 (5 points)

Complete the following code to compute the total number of movies in each genere.

In [25]:
```python
# Step 4:
Genres_count_rdd = Genres_list_rdd.map(lambda x: (x, 1))
Genres_count_rdd.take(3)
```

Out[25]:
```
[('Adventure', 1), ('Animation', 1), ('Children', 1)]
```

In [26]:
```python
Genres_total_rdd = Genres_count_rdd.reduceByKey(lambda x, y: x + y, 1)
Genres_total_rdd.collect()
```

Out[26]:
```
[('Adventure', 1117),
 ('Animation', 447),
 ('Children', 583),
 ('Comedy', 3315),
 ('Fantasy', 654),
 ('Romance', 1545),
 ('Drama', 4365),
 ('Action', 1545),
 ('Crime', 1100),
 ('Thriller', 1729),
 ('Horror', 877),
 ('Mystery', 543),
 ('Sci-Fi', 792),
 ('Documentary', 495),
 ('IMAX', 153),
 ('War', 367),
 ('Musical', 394),
 ('Western', 168),
 ('Film-Noir', 133),
 ('(no genres listed)', 18)]
```

2.2 Transforming a Column Using Split

We can transform a column value that represents a list using a special character such as "|" or "-") using split Spark SQL function.

In [27]:
```python
Splitted_Generes_DF= movies_DF.select(split(col("Genres"), '\|'))
Splitted_Generes_DF.show(2)
```

```
+---------------------+
|split(Genres, \|, -1)|
+---------------------+
| [Adventure, Anima...|
| [Adventure, Child...|
+---------------------+
only showing top 2 rows
```

2.3 Adding a Column to a DataFrame using withColumn

We often need to transform content of a column into another column. For example, if we transform the column Genres in the movies DataFrame into an array of genre categories, we can more easily check whether a movie is of certain genre.

In [28]:
```python
movies2_DF= movies_DF.withColumn("Genres_Array",split(col("Genres"), '\|') )
```

In [29]:
```python
movies2_DF.printSchema()
```

```
root
 |-- MovieID: integer (nullable = true)
 |-- MovieTitle: string (nullable = true)
 |-- Genres: string (nullable = true)
 |-- Genres_Array: array (nullable = true)
 |    |-- element: string (containsNull = true)
```

In [30]:
```python
movies2_DF.show(2)
```

```
+-------+----------------+--------------------+--------------------+
|MovieID|      MovieTitle|              Genres|        Genres_Array|
+-------+----------------+--------------------+--------------------+
|      1|Toy Story (1995)|Adventure|Animati...|[Adventure, Anima...|
|      2|  Jumanji (1995)|Adventure|Childre...|[Adventure, Child...|
+-------+----------------+--------------------+--------------------+
only showing top 2 rows
```

3. Computing Total Reviews and Average Rating for Each Movie

Because it is convenient and efficient to compute both total reviews and average rating for each movie using key value pairs, we will convert the reviews Data Frame into RDD.

In [31]:
```python
# Step 3.1 Convert the Reviews DF into RDD
ratings_RDD = ratings_DF.rdd
ratings_RDD.take(2)
```

Out[31]:
```
[Row(UserID=1, MovieID=31, Rating=2.5, RatingID=1260759144),
 Row(UserID=1, MovieID=1029, Rating=3.0, RatingID=1260759179)]
```

In [32]:
```python
# Step 3.2 Transform the ratings_RDD into key value pairs where key is Movie ID
movie_ratings_RDD = ratings_RDD.map(lambda row: (row.MovieID, row.Rating))
```

In [33]:
```python
movie_ratings_RDD.take(2)
```

Out[33]:
```
[(31, 2.5), (1029, 3.0)]
```

Exercise 3 (5 points)

Complete the code below to compute the total number of reviews for each movie.

In [34]:
```python
# Step 3.3 Compute total number of reviews for each movie
movie_review_count_RDD = movie_ratings_RDD.map(lambda x: (x[0], 1))
movie_review_total_RDD = movie_review_count_RDD.reduceByKey(lambda x, y: x+y, 1)
```

In [35]:
```python
movie_review_total_RDD.take(4)
```

Out[35]:
```
[(31, 42), (1029, 42), (1061, 33), (1129, 48)]
```

In [36]:
```python
# Step 3.4 Compute average rating for each movie
rating_total_RDD = movie_ratings_RDD.reduceByKey(lambda x, y: x+y, 1)
```

In [37]:
```python
rating_total_RDD.take(4)
```

Out[37]:
```
[(31, 133.5), (1029, 155.5), (1061, 117.0), (1129, 159.0)]
```

Join Transformation on Two RDDs

Two Key Value Pairs RDDs can be joined on the RDD (similar to the join operation in SQL) to return a new RDD, whose rows is an inner join of the two input RDDs. Only key value pairs occur in both input RDDs occur in the output RDD.

In [38]:
```python
# Step 3.5 Join the two RDDs (one counts total reviews, the other computes sum of ratings)
joined_RDD = rating_total_RDD.join(movie_review_total_RDD)
```

In [39]:
```python
joined_RDD.take(4)
```

Out[39]:
```
[(31, (133.5, 42)),
 (1029, (155.5, 42)),
 (1061, (117.0, 33)),
 (1129, (159.0, 48))]
```

Exercise 4 (10 points)

Complete the following code to compute average rating for each movie.

In [40]:
```python
# Step 3.6 Compute average rating for each movie
average_rating_RDD = joined_RDD.map(lambda x: (x[0], x[1][0]/x[1][1] ))
```

In [41]:
```python
average_rating_RDD.take(4)
```

Out[41]:
```
[(31, 3.1785714285714284),
 (1029, 3.7023809523809526),
 (1061, 3.5454545454545454),
 (1129, 3.3125)]
```

Exercise 5 (5 points)

Complete the following code to combine the two RDDs into one in the form of

```
(<movieID>, (<average rating>, <total review>)
```

In [42]:
```python
# Step 3.7 We want to keep both average review and total number of reviews for each movie.
# So we do another join her.
avg_rating_total_review_RDD = average_rating_RDD.join(movie_review_total_RDD)
```

In [43]:
```python
avg_rating_total_review_RDD.take(4)
```

Out[43]:
```
[(1172, (4.260869565217392, 46)),
 (2150, (3.513888888888889, 36)),
 (2294, (3.2735849056603774, 53)),
 (2968, (3.5697674418604652, 43))]
```

Transforming RDD to Data Frame

An RDD can be transformed to a Data Frame using toDF(). We want to transform the RDD containing average rating and total reviews for each movie into a Data Frame so that we can answer questions that involve bothmovie reviews and generes such as the following:

- What movies in a genre (e.g., comedy) has a top 10 average review among those that receive at least k reviews?

In [44]:
```python
# Before transforming to Data Frame, we first convert the key value pairs of avg_rating_total_reivew_RDD
# which has the format of (<movie ID> (<average rating> <review total>) )  to a tuple of the format
# (<movie ID> <average rating> <review total>)
avg_rating_total_review_tuple_RDD = avg_rating_total_review_RDD.map(lambda x: (x[0], x[1][0], x[1][1]) )
```

In [45]:
```python
avg_rating_total_review_tuple_RDD.take(4)
```

Out[45]:
```
[(1172, 4.260869565217392, 46),
 (2150, 3.513888888888889, 36),
 (2294, 3.2735849056603774, 53),
 (2968, 3.5697674418604652, 43)]
```

Defining a Schema for Data Frame

As we have seen before, each Data Frame has a Schema, which defines the names of the column and the type of values for the column (e.g., string, integer, or float). There are two ways to specify the schema of a Data Frame:

- Infer the schema from the heading and the value of an input file (e.g., CSV). This is how the schema of movies_DF was created in the beginning of this notebook.
- Explicitly specify the Schema We will use one approach in the second category here to specify the column names and the type of column values of the DataFrame to be converted from the RDD above.

In [46]:
```python
schema = StructType([ StructField("MovieID", IntegerType(), True ), \
                      StructField("AvgRating", FloatType(), True ), \
                      StructField("TotalReviews", IntegerType(), True) \
                     ])
```

In [47]:
```python
# Convert the RDD to a Data Frame
avg_review_DF = avg_rating_total_review_tuple_RDD.toDF(schema)
```

In [48]:
```python
avg_review_DF.printSchema()
```

```
root
 |-- MovieID: integer (nullable = true)
 |-- AvgRating: float (nullable = true)
 |-- TotalReviews: integer (nullable = true)
```

In [49]:
```python
avg_review_DF.take(4)
```

Out[49]:
```
[Row(MovieID=1172, AvgRating=4.26086950302124, TotalReviews=46),
 Row(MovieID=2150, AvgRating=3.5138888359069824, TotalReviews=36),
 Row(MovieID=2294, AvgRating=3.2735848426818848, TotalReviews=53),
 Row(MovieID=2968, AvgRating=3.569767475128174, TotalReviews=43)]
```

Join Transformation on Two DataFrames

We want to join the avg_rating_total_review_DF with movies2_DF

In [50]:
```python
joined_DF = avg_review_DF.join(movies2_DF,'MovieID', 'inner')
```

In [51]:
```python
movies2_DF.printSchema()
```

```
root
 |-- MovieID: integer (nullable = true)
 |-- MovieTitle: string (nullable = true)
 |-- Genres: string (nullable = true)
 |-- Genres_Array: array (nullable = true)
 |    |-- element: string (containsNull = true)
```

In [52]:
```python
joined_DF.show(4)
```

```
+-------+---------+------------+--------------------+--------------------+--------------------+
|MovieID|AvgRating|TotalReviews|          MovieTitle|              Genres|        Genres_Array|
+-------+---------+------------+--------------------+--------------------+--------------------+
|   1172|4.2608695|          46|Cinema Paradiso (...|               Drama|             [Drama]|
|   2150|3.5138888|          36|Gods Must Be Craz...|    Adventure|Comedy| [Adventure, Comedy]|
|   2294|3.2735848|          53|         Antz (1998)|Adventure|Animati...|[Adventure, Anima...|
|   2968|3.5697675|          43| Time Bandits (1981)|Adventure|Comedy|...|[Adventure, Comed...|
+-------+---------+------------+--------------------+--------------------+--------------------+
only showing top 4 rows
```

Filter Data Frame on Elements of a Column Using ArrayContains

In [53]:
```python
from pyspark.sql.functions import array_contains
Adventure_DF = joined_DF.filter(array_contains('Genres_Array', \
                                               "Adventure")).select("MovieID","AvgRating","TotalReviews","MovieTitl
e")
```

In [54]:
```python
Adventure_DF.show(5)
```

```
+-------+---------+------------+--------------------+
|MovieID|AvgRating|TotalReviews|          MovieTitle|
+-------+---------+------------+--------------------+
|   2150|3.5138888|          36|Gods Must Be Craz...|
|   2294|3.2735848|          53|         Antz (1998)|
|   2968|3.5697675|          43| Time Bandits (1981)|
|     10|3.4508197|         122|    GoldenEye (1995)|
|    150|   3.9025|         200|    Apollo 13 (1995)|
+-------+---------+------------+--------------------+
only showing top 5 rows
```

In [55]:
```python
Sorted_Adventure_DF = Adventure_DF.orderBy('TotalReviews', ascending=False)
```

In [56]:
```python
Sorted_Adventure_DF.show(10)
```

```
+-------+---------+------------+--------------------+
|MovieID|AvgRating|TotalReviews|          MovieTitle|
+-------+---------+------------+--------------------+
|    260|4.2216496|         291|Star Wars: Episod...|
|    480|3.7062044|         274|Jurassic Park (1993)|
|      1|3.8724697|         247|    Toy Story (1995)|
|   1196| 4.232906|         234|Star Wars: Episod...|
|   1270|4.0154867|         226|Back to the Futur...|
|   1198| 4.193182|         220|Raiders of the Lo...|
|    780| 3.483945|         218|Independence Day ...|
|   1210| 4.059908|         217|Star Wars: Episod...|
|    588|3.6744187|         215|      Aladdin (1992)|
|    590|3.7178218|         202|Dances with Wolve...|
+-------+---------+------------+--------------------+
only showing top 10 rows
```

Selecting Rows from a DataFrame

Conditions for selecting rows from a DataFrame can be described as .where(). In the condition, the row's value of a column can be referred to as col(''). For example, the condition below select all adventure movies whose average rating is above 3.5.

In [57]:
```python
Top_Adventure_DF = Sorted_Adventure_DF.where(col('AvgRating')>3.0)
```

Exercise 6 (5 ponts)

Complete the code below for selecting all adventure movies whose average rating is above 3.5 and who has received at least 100 reviews.

In [58]:
```python
Top_Adventure2_DF = Sorted_Adventure_DF.where(col('AvgRating')>3.5).where(col('TotalReviews')>100)
```

Exercise 7 (5 points)

Commplete the following code to save the output (Adventure Movies selected by Exercise 6) as text file.

In [59]:
```python
output_path = "/storage/home/kky5082/ds410/lab4/Lab4_Sorted_Adventure_Movies"
Top_Adventure2_DF.rdd.saveAsTextFile(output_path)
```

Exercise 8 (20 points)

Copy this Notebook (right click on the notebook icon on the left, select Duplicate) to another notebook. Rename the noteook as "Lab4TopMovieReviews_fast". In that new notebook, implement a more efficient data transformation-action pipeline for obtaining all Adventure movies who have more than 50 TotalReviews, and whose average rating is higher than 3.0. Save the output in a slightly different name (e.g., "Lab4_Sorted_Adventure_Movies_faster".

In [ ]:
```python

```

In [ ]:
```python

```