Spark

July 6, 2018

1 Distributed Data Analytics - Exercise 9(Krithika Murugesan -277537)

Exercise 1: Apache spark basics : Part A

The given lists are stored as RDD and the join operations are used to perform the right outer join, where all elements from first table are taken with matching entries from second table, while full outer join is intersection of all elements. The frequency of character s in both the tables is done using map reduce and aggregate function after joining the two tables together

In [1]:
```
#import packages
from pyspark import SparkContext
from pyspark.sql import SQLContext
sc = SparkContext()

sqlContext = SQLContext(sc)
from pyspark.sql import Row

#Given lists
a = ["spark", "rdd", "python", "context", "create", "class"]
b = ["operation", "apache", "scala", "lambda","parallel","partition"]

#To RDD
rddA = sc.parallelize(a)
rddB = sc.parallelize(b)

A = rddA.map(lambda x: Row(name = x))
B = rddB.map(lambda x: Row(name = x))

schemaA = sqlContext.createDataFrame(A)
schemaB = sqlContext.createDataFrame(B)

print(type(schemaA),type(schemaB))
```

```
<class 'pyspark.sql.dataframe.DataFrame'> <class 'pyspark.sql.dataframe.DataFrame'>
```

In [2]:
```
#Right outer join
schemaA.join(schemaB, "name" ,"right_outer").show()
```

```
+---------+
|     name|
+---------+
|operation|
|   lambda|
|partition|
| parallel|
|    scala|
|   apache|
+---------+
```

```
In [3]: #Full outer join
        joined = schemaA.join(schemaB, "name" ,"full_outer")
        joined.show()
```

```
+---------+
|     name|
+---------+
|operation|
|   lambda|
|  context|
|partition|
|   create|
|      rdd|
| parallel|
|    scala|
|   apache|
|    spark|
|    class|
|   python|
+---------+
```

```
In [4]: #Map reduce : frequncy of s,where map counts s in seperate parts and reduce sums it
        joined.show()
        count = joined.rdd.map(lambda x: sum([each.count('s') for each in x]))
                                                    .reduce(lambda x, y: x + y)
        print("No: of 's' in Schema A and B ",count)
```

```
+---------+
|     name|
+---------+
|operation|
|   lambda|
|  context|
|partition|
```

```
|   create|
|      rdd|
| parallel|
|    scala|
|   apache|
|    spark|
|    class|
|   python|
+---------+

No: of 's' in Schema A and B  4
```

```
In [33]: #Using aggregate function, count "s"
         import pyspark.sql.functions as F
         count = joined.rdd.aggregate(0, lambda i, x: i + x[0].count('s'), lambda i, j: i+j)
         print("No: of 's' in Schema A and B ",count)
```

```
No: of 's' in Schema A and B  4
```

Part b) Basic Operations on DataFrames

```
In [9]: #Reading the json file into RDD
        import json

        stud = []
        for line in open('Student.json', 'r'):
            stud.append(json.loads(line))
        print(stud)

        temp=[json.dumps(stud)]
        jsonRDD = sc.parallelize(temp)
        df = sqlContext.read.json(jsonRDD)
        df.show()
```

```
[{'s_id': 1, 'first_name': 'Alan', 'last_name': 'Joe', 'dob': 'October 14, 1983', 'course': 'Hu
+-----------------+-----------------+----------+---------+------+----+
|           course|              dob|first_name|last_name|points|s_id|
+-----------------+-----------------+----------+---------+------+----+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    17|   2|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    16|   3|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|
|          Business|  17 February 1981| Christian|   Kiriam|    10|   6|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|
|     Deep Learning|  January 13, 1978|      John|     null|    10|   8|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    15|   9|
```

```
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|
|  Computer Science|      July 2, 1985|     April|    Black|  null|  12|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|
|    Data Analytics|              null|   Bridget|    Twain|     6|  18|
|          Business|      7 March 1980|   Darlene|    Mills|    19|  19|
|    Data Analytics|      June 2, 1985|   Zachary|     null|    10|  20|
+------------------+------------------+----------+---------+------+----+
```

1. Replacing the missing values in the points table by the avergae value, first the average is computed and the Nana are replaced
2. Replacing the missing values in other columns as well using fillna

```
In [10]: #Compute average
         from pyspark.sql.functions import avg
         m = (df.select(avg("Points"))).toPandas()
         m = float(m.iloc[0])

         #Replacing missing values with average
         df1 = df.fillna(m)

         #Replacing Nan with unknown and --
         df1 = df1.fillna({'dob':"August 15, 1991",'last_name':"__"})
         df1.show()
```

```
+------------------+------------------+----------+---------+------+----+
|            course|               dob|first_name|last_name|points|s_id|
+------------------+------------------+----------+---------+------+----+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    17|   2|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    16|   3|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|
|          Business|  17 February 1981| Christian|   Kiriam|    10|   6|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|
|     Deep Learning|  January 13, 1978|      John|       __|    10|   8|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    15|   9|
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|
|  Computer Science|      July 2, 1985|     April|    Black|    11|  12|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|
```

```
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|
|    Data Analytics|   August 15, 1991|   Bridget|    Twain|     6|  18|
|          Business|      7 March 1980|   Darlene|    Mills|    19|  19|
|    Data Analytics|      June 2, 1985|   Zachary|       __|    10|  20|
+------------------+------------------+----------+---------+------+----+
```

3. Changing the date format in dob column since each row is of different format, using date parser to get the date parts and convert it to required format as new_col

```
In [31]: #DAtetime packages
         from dateutil import parser
         import datetime
         from pyspark.sql.types import TimestampType,DateType
         from pyspark.sql.functions import UserDefinedFunction,col,date_format

         #Extract the date information using parser
         udf = UserDefinedFunction(lambda x:parser.parse(x), TimestampType())
         df2 = df1.withColumn("New",udf(df1.dob))

         #Convert the time stamp to required format,filling unknown with random value
         #to avoid errors...It will be replaced at the end
         func =  UserDefinedFunction(lambda x: datetime.datetime
                                    .strptime(str(x), '%Y-%m-%d %H:%M:%S'), TimestampType())
         df = df2.withColumn('new_col', date_format(func(col('New')), 'dd-MM-yyy'))
         df.show()
```

```
+------------------+------------------+----------+---------+------+----+----------+
|            course|               dob|first_name|last_name|points|s_id|   new_col|
+------------------+------------------+----------+---------+------+----+----------+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|14-10-1983|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    17|   2|26-09-1980|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    16|   3|12-06-1982|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|05-04-1987|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|01-11-1978|
|          Business|  17 February 1981| Christian|   Kiriam|    10|   6|17-02-1981|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|01-01-1984|
|     Deep Learning|  January 13, 1978|      John|       __|    10|   8|13-01-1978|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    15|   9|26-12-1989|
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|30-12-1987|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|12-06-1975|
|  Computer Science|      July 2, 1985|     April|    Black|    11|  12|02-07-1985|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|22-07-1980|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|07-02-1986|
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|18-05-1987|
```

```
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|10-08-1984|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|16-12-1990|
|    Data Analytics|   August 15, 1991|   Bridget|    Twain|     6|  18|15-08-1991|
|          Business|      7 March 1980|   Darlene|    Mills|    19|  19|07-03-1980|
|    Data Analytics|      June 2, 1985|   Zachary|       __|    10|  20|02-06-1985|
+------------------+------------------+----------+---------+------+----+----------+
```

4. Insert age, a coulmn with value current date-dob is added to the RDD to get the currrent age

```
In [30]: #Get current year
         from pyspark.sql.functions import year
         i = 2018

         df = df.withColumn("Age",i-year(col("new_col")))

         targetDf.show()
```

```
+------------------+------------------+----------+---------+------+----+----------+---+
|            course|               dob|first_name|last_name|points|s_id|   new_col|Age|
+------------------+------------------+----------+---------+------+----+----------+---+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|14-10-1983| 35|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    17|   2|26-09-1980| 38|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    16|   3|12-06-1982| 36|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|05-04-1987| 31|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|01-11-1978| 40|
|          Business|  17 February 1981| Christian|   Kiriam|    10|   6|17-02-1981| 37|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|01-01-1984| 34|
|     Deep Learning|  January 13, 1978|      John|       __|    10|   8|13-01-1978| 40|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    15|   9|26-12-1989| 29|
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|30-12-1987| 31|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|12-06-1975| 43|
|  Computer Science|      July 2, 1985|     April|    Black|    11|  12|02-07-1985| 33|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|22-07-1980| 38|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|07-02-1986| 32|
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|18-05-1987| 31|
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|10-08-1984| 34|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|16-12-1990| 28|
|    Data Analytics|   August 15, 1991|   Bridget|    Twain|     6|  18|15-08-1991| 27|
|          Business|      7 March 1980|   Darlene|    Mills|    19|  19|07-03-1980| 38|
|    Data Analytics|      June 2, 1985|   Zachary|       __|    10|  20|02-06-1985| 33|
+------------------+------------------+----------+---------+------+----+----------+---+
```

```
In [29]: #Replacing the random values, the final output
         from pyspark.sql.functions import when
```

```
        targetDf = df.withColumn("new_col",when(df["dob"] == 'Feb 28, 2018', "Unknown")
                                 .otherwise(df['new_col']))
        targetDf = targetDf.withColumn("dob",when(df["dob"] == 'Feb 28, 2018', "Unknown")
                                       .otherwise(df['dob']))

        targetDf.show()
```

```
+------------------+------------------+----------+---------+------+----+----------+---+
|            course|               dob|first_name|last_name|points|s_id|   new_col|Age|
+------------------+------------------+----------+---------+------+----+----------+---+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|14-10-1983| 35|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    17|   2|26-09-1980| 38|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    16|   3|12-06-1982| 36|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|05-04-1987| 31|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|01-11-1978| 40|
|          Business| 17 February 1981| Christian|   Kiriam|    10|   6|17-02-1981| 37|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|01-01-1984| 34|
|     Deep Learning|  January 13, 1978|      John|       __|    10|   8|13-01-1978| 40|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    15|   9|26-12-1989| 29|
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|30-12-1987| 31|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|12-06-1975| 43|
|  Computer Science|      July 2, 1985|     April|    Black|    11|  12|02-07-1985| 33|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|22-07-1980| 38|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|07-02-1986| 32|
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|18-05-1987| 31|
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|10-08-1984| 34|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|16-12-1990| 28|
|    Data Analytics|   August 15, 1991|   Bridget|    Twain|     6|  18|15-08-1991| 27|
|          Business|      7 March 1980|   Darlene|    Mills|    19|  19|07-03-1980| 38|
|    Data Analytics|      June 2, 1985|   Zachary|       __|    10|  20|02-06-1985| 33|
+------------------+------------------+----------+---------+------+----+----------+---+
```

5. Updating points to 20 if the score is one stddev greater

```
In [24]: from pyspark.sql.functions import mean as _mean, stddev as _stddev, col

         #Computing stddev
         df_stats = targetDf.select(_stddev(col('points')).alias('std')).collect()
         std = df_stats[0]['std']
         #print(std)

         #Conditionally checking if points > std, then changing it to 20 else leaving it as it
         targetDf = targetDf.withColumn("points",when(targetDf["points"] > std, 20)
                                        .otherwise(targetDf['dob']))
```

```
0.0
+------------------+------------------+----------+---------+------+----+----------+---+
```

```
|            course|               dob|first_name|last_name|points|s_id|   new_col|Age|
+------------------+------------------+----------+---------+------+----+----------+---+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    20|   1|14-10-1983| 35|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    20|   2|26-09-1980| 38|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    20|   3|12-06-1982| 36|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    20|   4|05-04-1987| 31|
|        Psychology|  November 1, 1978|      Kira| Schommer|    20|   5|01-11-1978| 40|
|          Business|  17 February 1981| Christian|   Kiriam|    20|   6|17-02-1981| 37|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    20|   7|01-01-1984| 34|
|     Deep Learning|  January 13, 1978|      John|       __|    20|   8|13-01-1978| 40|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    20|   9|26-12-1989| 29|
|           Physics|  30 December 1987|     Marta|   Brooks|    20|  10|30-12-1987| 31|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    20|  11|12-06-1975| 43|
|  Computer Science|      July 2, 1985|     April|    Black|    20|  12|02-07-1985| 33|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    20|  13|22-07-1980| 38|
|        Psychology|   7 February 1986|      Mark|    Weber|    20|  14|07-02-1986| 32|
|       Informatics|      May 18, 1987|     Rosie|   Norman|    20|  15|18-05-1987| 31|
|          Business|   August 10, 1984|    Martin|   Steele|    20|  16|10-08-1984| 34|
|  Machine Learning|  16 December 1990|     Colin| Martinez|    20|  17|16-12-1990| 28|
|    Data Analytics|   August 15, 1991|   Bridget|    Twain|    20|  18|15-08-1991| 27|
|          Business|      7 March 1980|   Darlene|    Mills|    20|  19|07-03-1980| 38|
|    Data Analytics|      June 2, 1985|   Zachary|       __|    20|  20|02-06-1985| 33|
+------------------+------------------+----------+---------+------+----+----------+---+
```

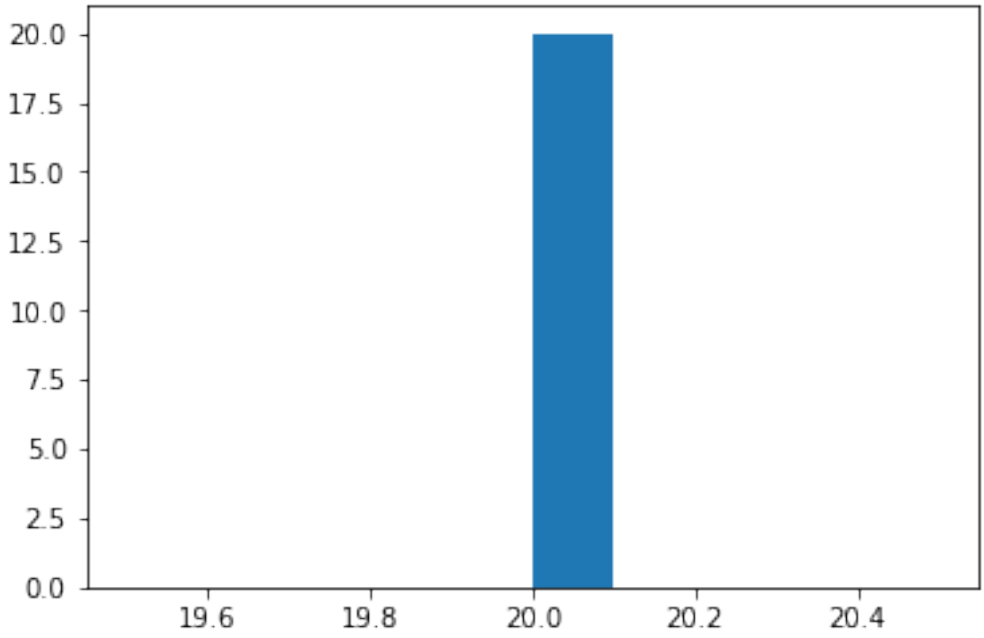
6. Histogram of previous result

```
In [17]: #generating values using rdd functions
         import pandas as pd
         import matplotlib.pyplot as plt
         histogram = targetDf.select('points').rdd.flatMap(lambda x: x).histogram(5)

         print(histogram)
```

```
(['20', '20'], [20])
```

```
In [23]: import matplotlib.pyplot as plt
         targetDf.groupBy("points").count().rdd.values().histogram(5)
         #Matplotlib

         df_pd = targetDf.toPandas()
         plt.hist(df_pd['points'])
```

```
Out[23]: (array([ 0.,  0.,  0.,  0.,  0., 20.,  0.,  0.,  0.,  0.]),
          array([19.5, 19.6, 19.7, 19.8, 19.9, 20. , 20.1, 20.2, 20.3, 20.4, 20.5]),
          <a list of 10 Patch objects>)
```


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
19.6
19.8
20.0
20.2
20.4

movielens

July 6, 2018

1 Exercise 2: Manipulating Recommender Dataset with Apache Spark

In [1]:
```
#importing the packages
from pyspark import SparkContext
from pyspark.sql import SQLContext
import pandas as pd

sc = SparkContext()
sqlContext = SQLContext(sc)

#Reading the data as pandas dataframe
train = pd.read_csv(r'/home/kritz/Documents/DDL/Ex09/tags.dat',sep="::",header=None)
```

```
/home/kritz/anaconda3/envs/scripts/lib/python3.6/site-packages/ipykernel_launcher.py:8: ParserW
```

The tag data is read as a dataframe and converted into a RDD, to acheive this all the object data type has to be converted into the respective data types which is done below

In [2]:
```
#Convert to str
train.columns=["userId","movieaId","tag","timestamp"]
train.head()
train.info()
train['tag']= train['tag'].astype(str)
data = sqlContext.createDataFrame(train)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 95580 entries, 0 to 95579
Data columns (total 4 columns):
userId       95580 non-null int64
movieaId     95580 non-null int64
tag          95564 non-null object
timestamp    95580 non-null int64
dtypes: int64(3), object(1)
memory usage: 2.9+ MB
```

```
In [4]: #Converting the timestamp to timestamp datatype
        from pyspark.sql.functions import *
        data.dtypes
        data = data.withColumn('new_time',to_timestamp(data.time, 'dd-MM-yyyy HH:mm:ss'))
```

1. To create user sessions of 30 mins, rank over similar to SQl is implemented. The data is partitioned by user id and ordered by the time. So for each user, the timestamp is ordered a lagcolumn is created so we can subtract the consecutive time entries to get the differencce in time between the sessions. Then this value is checked against 30, if it is within 30 minutes it is assigned the same session or it is assigned next session. Same seeion is denoted by 0 and next session by 1, so at the end a cumulative sum will give the session id over which mean and averages can be calculated

```
In [5]: #Rank over
        from pyspark.sql import Window
        w = Window.partitionBy("userId").orderBy(asc("new_time"))

        #Creating lag
        dataNew = data.withColumn('lag',lag(data.new_time).over(w))
        timeFmt = "yyyy-MM-dd'T'HH:mm:ss"

        #Finding time difference and assigning initial session or not
        dataNew = dataNew.withColumn('diff',when((unix_timestamp(dataNew.new_time
                                                                 , format=timeFmt)
                    - unix_timestamp(dataNew.lag, format=timeFmt))/60 < 30,0)
                                                                   .otherwise(1))

        #Cumulative sum to get session id
        dataNew = dataNew.withColumn('session', sum('diff').over(w))

        #For readability
        columns_to_drop = ['time', 'lag','timestamp','diff']
        dataNew = dataNew.drop(*columns_to_drop)
        dataNew.show(10)
```

```
+------+--------+--------------------+-------------------+-------+
|userId|movieaId|                 tag|           new_time|session|
+------+--------+--------------------+-------------------+-------+
|  1806|   43560|              comedy|2006-05-18 22:23:28|      1|
|  1806|   43560|                kids|2006-05-18 22:23:28|      1|
|  1806|    7018|            language|2007-02-22 16:24:59|      2|
|  1806|    7152|              nudity|2007-04-13 19:05:53|      3|
|  1806|    7152|                dark|2007-04-13 19:06:30|      3|
|  1806|   44709|        heartwarming|2007-04-13 19:26:25|      3|
|  1806|   44199|intelligent thriller|2007-04-13 19:28:17|      3|
|  1806|   43936|               tense|2007-04-13 19:29:36|      3|
|  1806|   43928|              stupid|2007-04-13 19:30:29|      3|
|  1806|   42734|              clever|2007-04-13 19:32:16|      3|
```

```
+------+--------+-------------------+-------------------+-------+
only showing top 10 rows
```

2. Frequency of sessions is the how many sessions the user has logged in, which is the max of seesion id

```
In [6]: frequency = dataNew.groupBy("userId").max("session").orderBy('max(session)'
                                                                    ,ascending=False)
        frequency.show(10)
```

```
+------+------------+
|userId|max(session)|
+------+------------+
| 10555|         891|
| 23172|         482|
|   146|         333|
| 33384|         269|
| 47448|         199|
| 34745|         144|
| 11898|         127|
| 30167|         115|
| 64633|         108|
|  8041|         104|
+------+------------+
only showing top 10 rows
```

3. Mean and stddev for each user

```
In [7]: #Mean
        dataNew.groupBy("userId").mean("session").orderBy('avg(session)',
                                                          ascending=False).show()
```

```
+------+------------------+
|userId|      avg(session)|
+------+------------------+
| 10555| 530.8542914171657|
| 23172|267.95190877540904|
|   146| 128.9478155339806|
| 33384|109.33039647577093|
| 47448| 91.14512195121951|
| 11898| 62.71298174442191|
| 64633| 56.24504249291785|
| 34745| 53.15585443037975|
| 41838|43.367198838896954|
|  6362|          43.28125|
```

```
| 23388| 38.94854586129754|
| 50970| 34.81666666666667|
|  8041| 34.44179104477612|
| 32828|29.953929539295395|
|  3962|29.456043956043956|
| 19460|29.116071428571427|
| 48621| 29.08076923076923|
| 49882| 28.65049928673324|
| 24221|  27.5472972972973|
| 39689| 27.19685039370079|
+------+------------------+
only showing top 20 rows
```

```
In [8]: #Stddev
        dataNew.groupBy("userId").agg(stddev("session")).show()
```

```
+------+--------------------+
|userId|stddev_samp(session)|
+------+--------------------+
|  1806|  1.2004900959975617|
|  2040|                 0.0|
| 15437|                 NaN|
| 15663|                 NaN|
| 15846|                 0.0|
| 18295|                 0.5|
| 18730|                 NaN|
| 19141|                 NaN|
| 25649|  1.2909944487358056|
| 27919|  0.5773502691896258|
| 29018|                 NaN|
| 31156|                 NaN|
| 37098|                 NaN|
| 39104|                 NaN|
| 39713|  0.5773502691896258|
| 48280|  0.5773502691896257|
| 50049|                 0.0|
| 55700|                 NaN|
| 60016|                 NaN|
| 60738|                 0.0|
+------+--------------------+
only showing top 20 rows
```

4. Mean and stddev for across user

```
In [11]: avg = dataNew.agg(mean("session")).collect()[0]['avg(session)']
         std = dataNew.agg(stddev("session")).collect()[0]['stddev_samp(session)']
```

```
In [2]: print("Across all users: \n mean: ",avg ,"\n std :",std)

Across all users:
 mean:  61.71259677756853
 std : 150.1792632124109
```

4. List of users greater than 2 std dev from mean, the mean of each user is compared to 2 std dev if it is greater then the user id is filtered and stored in users

```
In [26]: #Cond check
         userMean = dataNew.groupBy("userId").mean("session").orderBy('avg(session)'
                                                                      ,ascending=False)
         temp = userMean.withColumn("checkMean",when(userMean["avg(session)"] < (2*std)
                                                                     , 1).otherwise(0))

In [32]: #Actual users flitered after condition
         users = temp.filter(temp.checkMean == 1)
         users.select('userId').distinct().show()

+------+
|userId|
+------+
| 62989|
|  1806|
| 25649|
| 18295|
| 27919|
| 48280|
| 39713|
|  2040|
| 15437|
| 15663|
| 15846|
| 18730|
| 19141|
| 29018|
| 31156|
| 37098|
| 39104|
| 50049|
| 55700|
| 60016|
+------+
only showing top 20 rows
```

Bonus

July 6, 2018

1 Bonus : Analysis of Movie dataset using Apache Spark MapReduce

Here also the data is read as pandas dataframe which is converted to RDD, also to acheive this all object datatype is converted to string

```
In [1]: #importing packages
        from pyspark import SparkContext
        from pyspark.sql import SQLContext
        import pandas as pd

        sc = SparkContext()
        sqlContext = SQLContext(sc)

        #Reading as pandas
        movie = pd.read_csv(r'/home/kritz/Documents/DDL/Ex09/movies.csv')
        rating = pd.read_csv(r'/home/kritz/Documents/DDL/Ex09/ratings.csv')
```

```
In [2]: #Converting object type to string
        movie['title']= movie['title'].astype(str)
        movie['genres']= movie['genres'].astype(str)

        #RDD
        movie = sqlContext.createDataFrame(movie)
        rating = sqlContext.createDataFrame(rating)
```

Since, the operations require a comparison between the two datasets, they are joined using the common field user id

```
In [3]: #join
        data = movie.join(rating, "movieId" ,"inner")
        data.show(5)
```

```
+-------+--------------+------+------+------+----------+
|movieId|         title|genres|userId|rating| timestamp|
+-------+--------------+------+------+------+----------+
|     26|Othello (1995)| Drama|     9|   3.0| 938628655|
|     26|Othello (1995)| Drama|    70|   5.0| 853954729|
|     26|Othello (1995)| Drama|    86|   4.0| 848160245|
```

```
|     26|Othello (1995)| Drama|   119|   5.0| 913117867|
|     26|Othello (1995)| Drama|   294|   3.5|1082754003|
+-------+--------------+------+------+------+----------+
only showing top 5 rows
```

Dropping unwanted columns

```
In [4]: columns_to_drop = ['movieId', 'genres','userId','timestamp']
        test = data.drop(*columns_to_drop)
        test.show(10)
```

```
+--------------------+------+
|               title|rating|
+--------------------+------+
|      Othello (1995)|   3.0|
|      Othello (1995)|   5.0|
|      Othello (1995)|   4.0|
|      Othello (1995)|   5.0|
|      Othello (1995)|   3.5|
|City of Lost Chil...|   4.5|
|City of Lost Chil...|   3.0|
|City of Lost Chil...|   4.0|
|City of Lost Chil...|   4.0|
|City of Lost Chil...|   4.0|
+--------------------+------+
only showing top 10 rows
```

1. Movie with highest average rating using map reduce, the map function is used to find sum across all divisions, the reduce function is a max function here which gets the movie title with max rating, first such movie is displayed

```
In [6]: final = test.rdd.groupByKey().mapValues(lambda x: sum(x) / len(x))
                                                        .max(key=lambda x:x[1])
        print(final)
```

```
('Even Dwarfs Started Small (Auch Zwerge haben klein angefangen) (1971)', 5.0)
```

2. Find the user who has assign the lowest average ratings among all the users the number of ratings greater than 40? First the datset is filtered to get the subset of users with number of ratings greater than 40 using filter and aggregate functions,and the user list is displayed

```
In [7]: #Filtering users
        from pyspark.sql.functions import countDistinct
        df = data.groupby('userId').agg(countDistinct("movieId"))
        filtered = df.filter(df['count(DISTINCT movieId)'] > 40)
```

```
In [8]: #joining with original data to get the filtered user's ratings
        newData = data.join(filtered, "userId" ,"inner")
        columns_to_drop = ['movieId','title','genres','count(DISTINCT movieId)','timestamp']
        newData = newData.drop(*columns_to_drop)
        newData.show()
```

```
+------+------+
|userId|rating|
+------+------+
|    26|   4.0|
|    26|   2.0|
|    26|   2.0|
|    26|   3.0|
|    26|   3.5|
|    26|   3.5|
|    26|   4.0|
|    26|   2.0|
|    26|   3.0|
|    26|   4.0|
|    26|   4.0|
|    26|   3.5|
|    26|   5.0|
|    26|   4.5|
|    26|   3.0|
|    26|   3.0|
|    26|   3.5|
|    26|   4.5|
|    26|   3.5|
|    26|   3.5|
+------+------+
only showing top 20 rows
```

```
In [9]: #Getting the user with min of average values
        user = newData.rdd.groupByKey().mapValues(lambda x: sum(x) / len(x))
                                                              .min(key=lambda x:x[1])
        print(user)
```

```
(581, 1.4591836734693877)
```

3. Find the movie genre with the highest average ratings?, now the ratings are grouped by genre and the average for each genre is calculated using map function the reduce function gets the max of this value

```
In [10]: columns_to_drop = ['movieId', 'title','userId','timestamp']
         genre = data.drop(*columns_to_drop)
         genre.show()
```

```
+--------------------+------+
|              genres|rating|
+--------------------+------+
|               Drama|   3.0|
|               Drama|   5.0|
|               Drama|   4.0|
|               Drama|   5.0|
|               Drama|   3.5|
|Adventure|Drama|F...|   4.5|
|Adventure|Drama|F...|   3.0|
|Adventure|Drama|F...|   4.0|
|Adventure|Drama|F...|   4.0|
|Adventure|Drama|F...|   4.0|
|Adventure|Drama|F...|   5.0|
|Adventure|Drama|F...|   5.0|
|Adventure|Drama|F...|   3.0|
|Adventure|Drama|F...|   5.0|
|Adventure|Drama|F...|   3.5|
|Adventure|Drama|F...|   2.5|
|Adventure|Drama|F...|   5.0|
|Adventure|Drama|F...|   5.0|
|Adventure|Drama|F...|   4.0|
|Adventure|Drama|F...|   5.0|
+--------------------+------+
only showing top 20 rows
```

```
In [11]: movieGenre = genre.rdd.groupByKey().mapValues(lambda x: sum(x) / len(x))
                                                                    .max(key=lambda x:x[1])
         print(movieGenre)
```

```
('Action|Comedy|Drama|Romance', 5.0)
```