Manipulating Recommender Dataset with Apache Spark

In [1]:
```
#Import required libraries
import pyspark
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql import functions as f
from pyspark.sql.functions import datediff,col
from pyspark.sql.types import IntegerType
from pyspark.sql.functions import when
from pyspark.sql.window import Window
from pyspark.sql.functions import lag

#Initialize spark context and session
sc = SparkContext()

spark = SparkSession.builder.appName("movie").getOrCreate()
```

```
22/07/03 18:24:44 WARN Utils: Your hostname, Sruthys-MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 192.168.0.103 instead (on interface en0)
22/07/03 18:24:44 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/07/03 18:24:45 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

In [2]:
```
#Reading the tags file
df = spark.read.format("csv").option("delimiter", "::").load("Desktop/Sem2/tags.dat")
```

In [3]:
```
df.show()
```

```
+---+-----+-------------------+----------+
|_c0|  _c1|                _c2|       _c3|
+---+-----+-------------------+----------+
| 15| 4973|         excellent!|1215184630|
| 20| 1747|           politics|1188263867|
| 20| 1747|             satire|1188263867|
| 20| 2424|     chick flick 212|1188263835|
| 20| 2424|              hanks|1188263835|
| 20| 2424|               ryan|1188263835|
| 20| 2947|             action|1188263755|
| 20| 2947|               bond|1188263756|
| 20| 3033|              spoof|1188263880|
| 20| 3033|          star wars|1188263880|
| 20| 7438|             bloody|1188263801|
| 20| 7438|            kung fu|1188263801|
| 20| 7438|          Tarantino|1188263801|
| 21|55247|                  R|1205081506|
| 21|55253|              NC-17|1205081488|
| 25|   50|       Kevin Spacey|1166101426|
| 25| 6709|        Johnny Depp|1162147221|
| 31|   65|       buddy comedy|1188263759|
| 31|  546|strangely compelling|1188263674|
| 31| 1091|        catastrophe|1188263741|
+---+-----+-------------------+----------+
only showing top 20 rows
```

In [4]:
```
#Changing the column names

df2 = df.withColumnRenamed("_c0","UserID").withColumnRenamed("_c1","MovieID").withColumnRenamed("_c2","Tag").wi
```

In [5]:
```
df2.printSchema()
```

```
root
 |-- UserID: string (nullable = true)
 |-- MovieID: string (nullable = true)
 |-- Tag: string (nullable = true)
 |-- Timestamp: string (nullable = true)
```

In [6]:
```
df2.show()
```

```
+------+-------+--------------------+----------+
|UserID|MovieID|                 Tag| Timestamp|
+------+-------+--------------------+----------+
|    15|   4973|          excellent!|1215184630|
|    20|   1747|            politics|1188263867|
|    20|   1747|              satire|1188263867|
|    20|   2424|     chick flick 212|1188263835|
|    20|   2424|               hanks|1188263835|
|    20|   2424|                ryan|1188263835|
|    20|   2947|              action|1188263755|
|    20|   2947|                bond|1188263756|
|    20|   3033|               spoof|1188263880|
|    20|   3033|           star wars|1188263880|
|    20|   7438|              bloody|1188263801|
|    20|   7438|             kung fu|1188263801|
|    20|   7438|           Tarantino|1188263801|
|    21|  55247|                   R|1205081506|
|    21|  55253|               NC-17|1205081488|
|    25|     50|        Kevin Spacey|1166101426|
|    25|   6709|         Johnny Depp|1162147221|
|    31|     65|        buddy comedy|1188263759|
|    31|    546|strangely compelling|1188263674|
|    31|   1091|         catastrophe|1188263741|
+------+-------+--------------------+----------+
only showing top 20 rows
```

In [7]:
```
#Change the column to timestamp. The time is originally in unix format
#I converted it from string to timestamp type

df3 = df2.withColumn("Timestamp",f.from_unixtime(df2["Timestamp"]).cast("timestamp"))
```

In [8]:
```
df3.show()
```

```
+------+-------+--------------------+-------------------+
|UserID|MovieID|                 Tag|          Timestamp|
+------+-------+--------------------+-------------------+
|    15|   4973|          excellent!|2008-07-04 17:17:10|
|    20|   1747|            politics|2007-08-28 03:17:47|
|    20|   1747|              satire|2007-08-28 03:17:47|
|    20|   2424|     chick flick 212|2007-08-28 03:17:15|
|    20|   2424|               hanks|2007-08-28 03:17:15|
|    20|   2424|                ryan|2007-08-28 03:17:15|
|    20|   2947|              action|2007-08-28 03:15:55|
|    20|   2947|                bond|2007-08-28 03:15:56|
|    20|   3033|               spoof|2007-08-28 03:18:00|
|    20|   3033|           star wars|2007-08-28 03:18:00|
|    20|   7438|              bloody|2007-08-28 03:16:41|
|    20|   7438|             kung fu|2007-08-28 03:16:41|
|    20|   7438|           Tarantino|2007-08-28 03:16:41|
|    21|  55247|                   R|2008-03-09 17:51:46|
|    21|  55253|               NC-17|2008-03-09 17:51:28|
|    25|     50|        Kevin Spacey|2006-12-14 14:03:46|
|    25|   6709|         Johnny Depp|2006-10-29 19:40:21|
|    31|     65|        buddy comedy|2007-08-28 03:15:59|
|    31|    546|strangely compelling|2007-08-28 03:14:34|
|    31|   1091|         catastrophe|2007-08-28 03:15:41|
+------+-------+--------------------+-------------------+
only showing top 20 rows
```

The dataframe represents the tag given by different users for different movies in different timestamps.

A tagging session for a user can be defined as the duration in which he/she generated tagging activities. Typically, an inactive duration of 30 mins is considered as a termination of the tagging session. Our task is to separate out tagging sessions for each user.

First we group the timestamps for each user using window function. Then the prev timestamps are taken and put in another column for each window for each user using lag function. So for the first entry in a window, lag would be null.

In [9]:
```
w = Window().partitionBy("UserID").orderBy("Timestamp")
```

In [10]:
```
#Lag column containes the prev timestamp

df4 = df3.withColumn("lag",lag("Timestamp",1).over(w)) \
```

In [11]:
```
#Calculate the time difference between the lag and the current timestamp.
#The time difference is also displayed in another column

df5 = df4.withColumn("timediff", col("Timestamp").cast("long")-col("lag").cast("long"))
df5.show(20)
```

```
[Stage 6:>                                                          (0 + 1) / 1]
+------+-------+---------------+-------------------+-------------------+--------+
|UserID|MovieID|            Tag|          Timestamp|                lag|timediff|
+------+-------+---------------+-------------------+-------------------+--------+
|  1000|    277|children's story|2007-08-31 06:05:11|               null|    null|
|  1000|   1994|    sci-fi. dark|2007-08-31 06:05:36|2007-08-31 06:05:11|      25|
|  1000|   5377|        romance|2007-08-31 06:05:50|2007-08-31 06:05:36|      14|
|  1000|   7147|   family bonds|2007-08-31 06:06:01|2007-08-31 06:05:50|      11|
|  1000|    362|animated classic|2007-08-31 06:06:11|2007-08-31 06:06:01|      10|
|  1000|    276|         family|2007-08-31 06:07:15|2007-08-31 06:06:11|      64|
| 10003|  42013|       Passable|2006-06-16 06:33:55|               null|    null|
| 10003|  51662| FIOS on demand|2008-04-12 00:35:26|2006-06-16 06:33:55|57520891|
| 10003|  54997| FIOS on demand|2008-04-12 00:35:35|2008-04-12 00:35:26|       9|
| 10003|  55765| FIOS on demand|2008-04-12 00:35:42|2008-04-12 00:35:35|       7|
| 10003|  55363| FIOS on demand|2008-04-12 00:37:00|2008-04-12 00:35:42|      78|
| 10003|  56152| FIOS on demand|2008-04-12 00:38:46|2008-04-12 00:37:00|     106|
| 10003|  55116| FIOS on demand|2008-04-12 00:40:36|2008-04-12 00:38:46|     110|
| 10003|  56174| FIOS on demand|2008-04-12 00:41:10|2008-04-12 00:40:36|      34|
| 10003|  55176| FIOS on demand|2008-04-12 00:42:35|2008-04-12 00:41:10|      85|
| 10003|  55247| FIOS on demand|2008-04-12 00:42:36|2008-04-12 00:42:35|       1|
| 10003|  54881| FIOS on demand|2008-04-12 00:42:38|2008-04-12 00:42:36|       2|
| 10003|  55820| FIOS on demand|2008-04-12 00:44:33|2008-04-12 00:42:38|     115|
| 10003|  53123| FIOS on demand|2008-04-12 00:44:35|2008-04-12 00:44:33|       2|
| 10003|  53550| FIOS on demand|2008-04-12 00:45:37|2008-04-12 00:44:35|      62|
+------+-------+---------------+-------------------+-------------------+--------+
only showing top 20 rows
```

In [12]:
```
#Replacing the null time difference values with 0 as they represent the first tag for each user
df6 = df5.na.fill({'timediff': 0})
```

In [13]:
```
df6.show()
```

```
[Stage 7:>                                                          (0 + 1) / 1]
+------+-------+---------------+-------------------+-------------------+--------+
|UserID|MovieID|            Tag|          Timestamp|                lag|timediff|
+------+-------+---------------+-------------------+-------------------+--------+
|  1000|    277|children's story|2007-08-31 06:05:11|               null|       0|
|  1000|   1994|    sci-fi. dark|2007-08-31 06:05:36|2007-08-31 06:05:11|      25|
|  1000|   5377|        romance|2007-08-31 06:05:50|2007-08-31 06:05:36|      14|
|  1000|   7147|   family bonds|2007-08-31 06:06:01|2007-08-31 06:05:50|      11|
|  1000|    362|animated classic|2007-08-31 06:06:11|2007-08-31 06:06:01|      10|
|  1000|    276|         family|2007-08-31 06:07:15|2007-08-31 06:06:11|      64|
| 10003|  42013|       Passable|2006-06-16 06:33:55|               null|       0|
| 10003|  51662| FIOS on demand|2008-04-12 00:35:26|2006-06-16 06:33:55|57520891|
| 10003|  54997| FIOS on demand|2008-04-12 00:35:35|2008-04-12 00:35:26|       9|
| 10003|  55765| FIOS on demand|2008-04-12 00:35:42|2008-04-12 00:35:35|       7|
| 10003|  55363| FIOS on demand|2008-04-12 00:37:00|2008-04-12 00:35:42|      78|
| 10003|  56152| FIOS on demand|2008-04-12 00:38:46|2008-04-12 00:37:00|     106|
| 10003|  55116| FIOS on demand|2008-04-12 00:40:36|2008-04-12 00:38:46|     110|
| 10003|  56174| FIOS on demand|2008-04-12 00:41:10|2008-04-12 00:40:36|      34|
| 10003|  55176| FIOS on demand|2008-04-12 00:42:35|2008-04-12 00:41:10|      85|
| 10003|  55247| FIOS on demand|2008-04-12 00:42:36|2008-04-12 00:42:35|       1|
| 10003|  54881| FIOS on demand|2008-04-12 00:42:38|2008-04-12 00:42:36|       2|
| 10003|  55820| FIOS on demand|2008-04-12 00:44:33|2008-04-12 00:42:38|     115|
| 10003|  53123| FIOS on demand|2008-04-12 00:44:35|2008-04-12 00:44:33|       2|
| 10003|  53550| FIOS on demand|2008-04-12 00:45:37|2008-04-12 00:44:35|      62|
+------+-------+---------------+-------------------+-------------------+--------+
only showing top 20 rows
```

If the time diff is greater than 30 mins, then we set a col value as 1, otherwise 0. This is done so as to identify when a session starts

In [14]:
```
df7 = df6.withColumn("session", when(df6.timediff>= 1800,1) \
      .otherwise(0))
```

We now add a column called session_id. This column containes the session id for each tag. The session ids of a user start from 0. If there is inactivity for more than 30 mins, new session starts.

This is done by using window function on each user and time stamp and by taking two consecutive rows in a window. Then we find the sum of the previously calculated column session, where whenever new session starts 1 occurs, otherwise it is 0. Hence by finding sum of consecutive rows, we can give session ids.

In [15]:
```
W = Window.partitionBy(df7.UserID).orderBy(df7.Timestamp).rowsBetween(Window.unboundedPreceding, Window.currentl
df8 = df7.withColumn('session_id', f.sum(df7['session']).over(W))
df8.show(20)
```

```
+------+-------+---------------+-------------------+-------------------+--------+-------+----------+
|UserID|MovieID|            Tag|          Timestamp|                lag|timediff|session|session_id|
+------+-------+---------------+-------------------+-------------------+--------+-------+----------+
|  1000|    277|children's story|2007-08-31 06:05:11|               null|       0|      0|         0|
|  1000|   1994|    sci-fi. dark|2007-08-31 06:05:36|2007-08-31 06:05:11|      25|      0|         0|
|  1000|   5377|        romance|2007-08-31 06:05:50|2007-08-31 06:05:36|      14|      0|         0|
|  1000|   7147|   family bonds|2007-08-31 06:06:01|2007-08-31 06:05:50|      11|      0|         0|
|  1000|    362|animated classic|2007-08-31 06:06:11|2007-08-31 06:06:01|      10|      0|         0|
|  1000|    276|         family|2007-08-31 06:07:15|2007-08-31 06:06:11|      64|      0|         0|
| 10003|  42013|       Passable|2006-06-16 06:33:55|               null|       0|      0|         0|
| 10003|  51662| FIOS on demand|2008-04-12 00:35:26|2006-06-16 06:33:55|57520891|      1|         1|
| 10003|  54997| FIOS on demand|2008-04-12 00:35:35|2008-04-12 00:35:26|       9|      0|         1|
| 10003|  55765| FIOS on demand|2008-04-12 00:35:42|2008-04-12 00:35:35|       7|      0|         1|
| 10003|  55363| FIOS on demand|2008-04-12 00:37:00|2008-04-12 00:35:42|      78|      0|         1|
| 10003|  56152| FIOS on demand|2008-04-12 00:38:46|2008-04-12 00:37:00|     106|      0|         1|
| 10003|  55116| FIOS on demand|2008-04-12 00:40:36|2008-04-12 00:38:46|     110|      0|         1|
| 10003|  56174| FIOS on demand|2008-04-12 00:41:10|2008-04-12 00:40:36|      34|      0|         1|
| 10003|  55176| FIOS on demand|2008-04-12 00:42:35|2008-04-12 00:41:10|      85|      0|         1|
| 10003|  55247| FIOS on demand|2008-04-12 00:42:36|2008-04-12 00:42:35|       1|      0|         1|
| 10003|  54881| FIOS on demand|2008-04-12 00:42:38|2008-04-12 00:42:36|       2|      0|         1|
| 10003|  55820| FIOS on demand|2008-04-12 00:44:33|2008-04-12 00:42:38|     115|      0|         1|
| 10003|  53123| FIOS on demand|2008-04-12 00:44:35|2008-04-12 00:44:33|       2|      0|         1|
| 10003|  53550| FIOS on demand|2008-04-12 00:45:37|2008-04-12 00:44:35|      62|      0|         1|
+------+-------+---------------+-------------------+-------------------+--------+-------+----------+
only showing top 20 rows
```

To Calculate the frequency of tagging for each user session.

We first group the dataframe by user and session_id. Then we find the number of tags in each session of each user.

In [16]:
```
gr = df8.groupby(['UserID','session_id'])
df_grouped = gr.agg(f.count(col('session_id')).alias('freq'))
```

In [17]:
```
df_grouped.show()
```

```
[Stage 15:>                                                         (0 + 1) / 1]
+------+----------+----+
|UserID|session_id|freq|
+------+----------+----+
|  1000|         0|   6|
| 10003|         0|   1|
| 10003|         1|  18|
| 10003|         2|  38|
| 10020|         0|   2|
| 10025|         0|   1|
| 10032|         0|  39|
| 10032|         1|   1|
| 10032|         2|   1|
| 10032|         3|   1|
| 10032|         4|   4|
| 10032|         5|   1|
| 10032|         6|   1|
| 10032|         7|   4|
| 10032|         8|   1|
| 10032|         9|   1|
| 10032|        10|   1|
| 10032|        11|   1|
| 10051|         0|   1|
| 10058|         0|  35|
+------+----------+----+
only showing top 20 rows
```

To find a mean and standard deviation of the tagging frequency of each user.

First we group the above found dataframe by userid and then find the mean and standard deviation of the frequency values. Thus we get mean and std of tagging frequency for each user.

In [18]:
```
std_df = df_grouped.groupby('UserID').agg(f.stddev("freq").alias("std"))
mean_df = df_grouped.groupby('UserID').agg(f.mean("freq").alias("mean"))

std_df.show()
mean_df.show()
```

```
+------+------------------+
|UserID|               std|
+------+------------------+
|  1000|              null|
| 10003|18.520259177452136|
| 10020|              null|
| 10025|              null|
| 10032|10.873933246182093|
| 10051|              null|
| 10058|15.044378795195676|
| 10059|0.7071067811865476|
| 10064|              null|
| 10084|2.0615528128088303|
| 10094|              null|
|  1010|              null|
| 10117|0.7071067811865476|
| 10125|              null|
| 10132|1.4127396551853897|
| 10154|              null|
| 10167|              null|
|  1017|              null|
| 10181|              null|
| 10191|0.5773502691896258|
+------+------------------+
only showing top 20 rows

+------+------------------+
|UserID|              mean|
+------+------------------+
|  1000|               6.0|
| 10003|              19.0|
| 10020|               2.0|
| 10025|               1.0|
| 10032| 4.666666666666667|
| 10051|               1.0|
| 10058|25.333333333333332|
| 10059|               2.5|
| 10064|               1.0|
| 10084|              3.75|
| 10094|               2.0|
|  1010|               4.0|
| 10117|               1.5|
| 10125|              21.0|
| 10132|            1.5625|
| 10154|               8.0|
| 10167|               1.0|
|  1017|               7.0|
| 10181|              11.0|
| 10191|2.6666666666666665|
+------+------------------+
only showing top 20 rows
```

To find a mean and standard deviation of the tagging frequency for across users.

First we group the dataframe by users and then find the count of distinct session ids in each user. Thus we get the count of sessions for each user. Then we find the mean and standard deviation for these counts. Thus we get the mean and std of tagging freq across users

In [19]:
```
gro = df8.groupby('UserID')
df_grouped_across = gro.agg(f.countDistinct(col('session_id')).alias('freq'))
df_grouped_across.show()
```

```
+------+----+
|UserID|freq|
+------+----+
|  1000|   1|
| 10003|   3|
| 10020|   1|
| 10025|   1|
| 10032|  12|
| 10051|   1|
| 10058|   3|
| 10059|   2|
| 10064|   1|
| 10084|   4|
| 10094|   1|
|  1010|   1|
| 10117|   2|
| 10125|   1|
| 10132|  16|
| 10154|   1|
| 10167|   1|
|  1017|   1|
| 10181|   1|
| 10191|   3|
+------+----+
only showing top 20 rows
```

In [20]:
```
std_across = df_grouped_across.agg({'freq': 'stddev' })
mean_across = df_grouped_across.agg({'freq':'mean'})

std_across.show()
mean_across.show()
```

```
+------------------+
|      stddev(freq)|
+------------------+
|19.114155365086855|
+------------------+

+-----------------+
|        avg(freq)|
+-----------------+
|3.265901721127463|
+-----------------+
```

To provide the list of users with a mean tagging frequency within the two standard deviation from the mean frequency of all users.

We see that the mean+2*std for all users = 3.3 + 19.22* = 43 Here we take the dataframe which we had found above having the mean tagging frequency for each user. Then we find users such that their mean is within 43.

In [21]:
```
final = mean_df.filter(mean_df.mean<=43)
final.show()
```

```
+------+------------------+
|UserID|              mean|
+------+------------------+
|  1000|               6.0|
| 10003|              19.0|
| 10020|               2.0|
| 10025|               1.0|
| 10032| 4.666666666666667|
| 10051|               1.0|
| 10058|25.333333333333332|
| 10059|               2.5|
| 10064|               1.0|
| 10084|              3.75|
| 10094|               2.0|
|  1010|               4.0|
| 10117|               1.5|
| 10125|              21.0|
| 10132|            1.5625|
| 10154|               8.0|
| 10167|               1.0|
|  1017|               7.0|
| 10181|              11.0|
| 10191|2.6666666666666665|
+------+------------------+
only showing top 20 rows
```

In [22]:
```
final.count()
```

Out[22]:
```
3944
```

In [23]:
```
mean_df.count()
```

Out[23]:
```
4009
```

We see that 3944 users have mean within 2 std from mean of all users. The total number of users in the dataframe is 4009.

References

https://stackoverflow.com/questions/54337991/pyspark-from-unixtime-unix-timestamp-does-not-convert-to-timestamp
https://sparkbyexamples.com/pyspark/pyspark-rename-dataframe-column/ https://stackoverflow.com/questions/63136798/spark-read-dat-file-with-a-special-case https://sparkbyexamples.com/pyspark/pyspark-window-functions/
https://stackoverflow.com/questions/32880370/pyspark-combining-session-data-without-explicit-session-key-iterating-over-a
https://stackoverflow.com/questions/44020818/how-to-calculate-date-difference-in-pyspark
https://stackoverflow.com/questions/46421677/how-to-count-unique-id-after-groupby-in-pyspark
https://stackoverflow.com/questions/39504950/python-pyspark-get-sum-of-a-pyspark-dataframe-column-values
https://stackoverflow.com/questions/37707305/pyspark-multiple-conditions-in-when-clause