Working with Spark DataFrames - LAB 1

1. **How many elements can we find (in our DataFrame)?**

To find the elements in our DataFrame we'll use the command **`count()`**.

The resulting operation when applying the command to the DataFrame is **`c.count()`**, which when printing it gives as output:

```
Number of elements in the DataFrame: 1002
```

So, in total we have 1002 elements in the DataFrame.

2. **How many unique customers**?

To find how many unique customers we have in our DataFrame we'll first select the column 'customers' from the DataFrame and from this column we'll count those unique elements.

In order to do that we'll use the following commands:

- **`select(<column_name>)`** : to select only the target column
- **`distinct()`**: to obtain those unique elements in the target column
- **`count()`**: to count those unique elements that we've obtained from the column

The operation applied to the DataFrame that we obtain from combining these commands is:

```
c.select("customer").distinct().count()
```

The output is:

```
Unique customers: 31
```

Given this, we can say that there are 31 unique customers in total.

3. **How many products were purchased by each customer?**

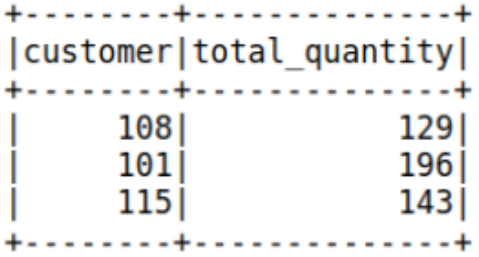
In order to do this we need to group the different customers in the DataFrame and for each of them compute the total quantity of products that they have purchased.

For this we want to use the commands **`groupBy()`**, **`agg()`** and **`exp()`**.

The resulting operation combining them is:

```
c.groupBy("customer").agg(expr("sum(quantity) as total_quantity"))
```

And the output (only showing 3 rows):



```
+--------+--------------+
|customer|total_quantity|
+--------+--------------+
|     108|           129|
|     101|           196|
|     115|           143|
+--------+--------------+
```

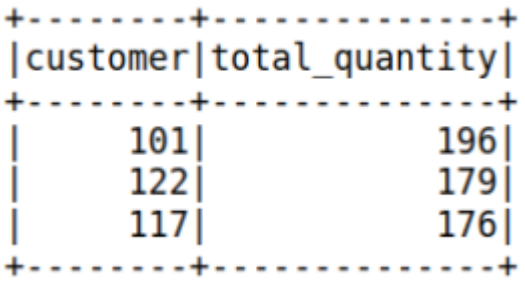
4. **Sort customers by quantity**

For this we'll use the stored variable that we've obtained from the previous question containing the quantity of products purchased for each customer (products_per_customer). Then we'll use the commands **`orderBy()`** and **`desc()`** to order the customers by the total quantity of bought products in a descending way.

The resulting operation is:

```
products_per_customer.orderBy(desc("total_quantity"))
```

The output for this is (showing only 3 rows):



```
+--------+--------------+
|customer|total_quantity|
+--------+--------------+
|     101|           196|
|     122|           179|
|     117|           176|
+--------+--------------+
```

5. **How many times customer id number 100 has purchased more than 5 items?**

To determine how many times customer ID 100 has purchased more than 5 items, we use the commands: **`where()`**, **`expr()`** and **`count()`**.

`where()` is used to obtain those elements where the expression (inside **`expr()`**) is satisfied. Using this, we filter by customer (**`where(expr("customer = 100"))`**) and then once we obtain the customer, by quantity of items (**`where(expr("quantity > 5"))`**). Afterwards we use **`count()`** to obtain just the number of products that satisfy these premises.

The operation applied to the DataFrame is as follows:

```
c.where(expr("customer = 100")).where(expr("quantity > 5")).count()
```

The output of this operation is:

```
Customer with id 100 has purchased 16 times more than 5 items.
```

6. **Which were the products bought by the customer with the largest number of transactions? We are interested in the customer that has done more purchases. You do not need to consider quantities of products, just how many times a customer has done a transaction.**

To identify the products bought by the customer with the largest number of transactions (based solely on the count of transactions), we use the following Spark commands: **`groupBy()`**, **`count()`**, **`orderBy()`**, **`head()`**, **`where()`**, **`expr()`**, **`select()`**, and **`distinct()`**.

`groupBy()` is used to group all different customers, then we use **`count()`** to count all purchases for each customer and then **`orderBy(desc("count"))`** to order them in descending order by this count. With this we obtain the customers ordered by number of purchases.

Then from this, we get the first customer, which is the one with the most purchases with the command **`head(1)`** to access the first element in the descending ranking that we've computed and then **`[0]["customer"]`** to obtain that specific customer.

Once we have the customer, we obtain it in the dataframe using **`where()`** and **`expr()`** by doing **`where(expr("customer") == first_customer_transactions)`** where we find that customer that matches the one with the most purchases. Then we just use **`select("product")`** to get the products from that customer and **`distinct()`** to obtain the different products that they bought.

The resulting operations are:

```
transactions = c.groupBy("customer").count().orderBy(desc("count"))

first_customer_transactions = transactions.head(1)[0]["customer"]

products = c.where(expr("customer") == first_customer_transactions).
select("product").distinct()
```

And the output is:

```
+-------+
|product|
+-------+
|      1|
|      6|
|      3|
|      5|
|      9|
|      4|
|      8|
|      7|
|     10|
|      2|
+-------+
```

Working with Spark DataFrames - LAB 2

1. **Can you obtain a basic summary list of statistics for our new movie ratings dataframe? Interesting information is the count, mean, max, and some selected percentiles**.

To obtain a basic summary list of statistics of the dataframe we'll use **`movie_ratings.describe()`**.

The describe command already gives us all this interesting information such as the count, mean, standard deviation or min and max for each column.

The result of this operation is:

```
+-------+-----------------+------------------+------------------+--------------------+--------------------+------------------+
|summary|          movieId|            userId|            rating|           timestamp|               title|            genres|
+-------+-----------------+------------------+------------------+--------------------+--------------------+------------------+
|  count|           100836|            100836|            100836|              100836|              100836|            100836|
|   mean|19435.2957177992|326.12756356856676| 3.501556983616962|1.2059460873684695E9|                null|              null|
| stddev|35530.9871987003| 182.6184914635004|1.0425292390606342|2.1626103599513078E8|                null|              null|
|    min|                1|                 1|               0.5|           828124615|"11'09""01 - Sept...|(no genres listed)|
|    max|           193609|               610|               5.0|          1537799250|À nous la liberté...|           Western|
+-------+-----------------+------------------+------------------+--------------------+--------------------+------------------+
```

Then to obtain the percentiles we'll use **`select()`**, **`expr(percentile_approx())`** and **`alias()`**.

We want to compute the percentiles of the ratings, so we'll select them, **`select()`**, then to obtain the percentiles we'll use the expression **`expr('percentile_approx(rating, array(0.25, 0.5, 0.75))`** where we select the 'rating' column and compute the 25%, 50% and 75% percentiles. Finally we give them a name for the resulting columns in the data frame that we create with these percentiles using **`alias(25%, 50%, 75%)`**.

The resulting operation is:

```
movie.ratings.select(expr('percentile_approx(rating, array(0.25,
                0.5, 0.75)))alias(25%, 50%, 75%)
```

And its result:

```
+---------------+
|  25%, 50%, 75%|
+---------------+
|[3.0, 3.5, 4.0]|
+---------------+
```

2. **What kind of join operations are used in left semi and left anti? Can you explain these operations with our validation example?**

The left_semi operation returns those rows from the left data frame that match the columns in both the left dataframe and right dataframe, whereas left_anti returns those which do not match from the left data frame. In the example we are creating the validation_df by using the left semi to obtain those rows where userId matches in v_df and train_df, then we do the same thing but for movieId with the resulting data frame of the previous operation and train_df. We use left anti to get those rows that do not match columns movieId and UserId in v_df and validation_df to later perform an union with the train_df and add those which aren't in the train_df to it.

3. **Train_df now has more or less records than initially? Why**?

Now, train_df has more records than initially because we have added the ones that were in v_df and that were not initially in train_df.

4. **Create a new DF derived from train_df grouping all records with the same rating, count them, and sort by the rating column in descending order.**

To do this we'll use the commands `groupBy()`, `count()`, `orderBy()` and `desc()`.

We use `groupBy("rating")` to group all the records with the same rating, then we count them by using `count()` and order them in a descending way with the combination of `orderBy(desc("rating"))`.

The resulting operation is:

```
train_df.groupBy("rating").count().orderBy(desc("rating"))
```

And the result:

```
+------+-----+
|rating|count|
+------+-----+
|   5.0|10644|
|   4.5| 6826|
|   4.0|21655|
|   3.5|10569|
|   3.0|16128|
|   2.5| 4508|
|   2.0| 6089|
|   1.5| 1468|
|   1.0| 2325|
|   0.5| 1099|
+------+-----+
```

5. **Extend the previous DataFrame to have a new column with the unique number of ratings for each movie. You need to consider a countDistinct with both "movieId" and "userId" so that a user only ranks once for each movie**.

We come from this operation in the cell before this exercise, where we obtain a dataframe with the rating for each genre and their number of movies from the train_with_genres_exploded data frame.

```
train_with_genres_exploded
      .groupBy("genre")
      .agg(
         mean(col("rating")).alias("genre_rating"),
         countDistinct("movieId").alias("num_movies")

      )

)
```

In order to obtain the same data frame with a new column with the unique number of ratings for genre, we'll add to the previous code:

```
countDistinct("userId","movieId").alias("num_ratings")
```

We use `countDistinct("userId", "movieId")` to obtain the number of unique users that have rated the movie and then `alias("num_ratings")` to rename the resulting column.

All in all, the resulting operation is:

```
train_with_genres_exploded
      .groupBy("genre")
      .agg(
         mean(col("rating")).alias("genre_rating"),
         countDistinct("movieId").alias("num_movies"),
         countDistinct("userId","movieId").alias("num_ratings")
      )

)
```

And the result:

```
+-----------+-----------------+----------+-----------+
|      genre|      genre_rating|num_movies|num_ratings|
+-----------+-----------------+----------+-----------+
|      Crime|3.6534308211473565|      1196|      13335|
|    Romance| 3.499146874146874|      1591|      14652|
|   Thriller| 3.489255925733943|      1889|      21221|
|  Adventure|3.5100289495450787|      1262|      19344|
|      Drama|3.6531895378424757|      4349|      33798|
|        War|3.8031265887137775|       381|       3934|
|Documentary|3.7878937007874014|       438|       1016|
|    Fantasy|3.4873953974895398|       778|       9560|
|    Mystery|3.6281553398058253|       573|       6180|
|    Musical| 3.570217917675545|       333|       3304|
+-----------+-----------------+----------+-----------+
```

6. **Can you program a top 10 list of best average rating genres? and a top 10 list of genres with the most ratings?**

To obtain these top 10s we'll use the commands `orderBy()`, `desc()`, `select()` and `limit()` for both of them.

We use the combination of `orderBy(desc("genre_rating"))` for the average genre rating top 10 and `orderBy(desc("num_ratings"))` for the most rated genres top 10. Then we use `select('genre', "genre_rating")` and `select('genre', "num_ratings")` to select the genres and then the genre rating of the number of ratings depending on the top 10 that we want. After that we set a limit of 10 using `limit(10)`.

The resulting operations are:

```
top10_genere_rating = mean_ratings.orderBy(desc("genre_rating")).select('genre', "genre_rating").limit(10)

top10_num_ratings = mean_ratings.orderBy(desc("num_ratings")).select('genre', "num_ratings").limit(10)
```

And the results:

```
+----------+------------------+
|     genre|      genre_rating|
+----------+------------------+
|  Film-Noir| 3.908440629470672|
|       War|3.8031265887137775|
|Documentary|3.7878937007874014|
|     Crime|3.6534308211473565|
|     Drama|3.6531895378424757|
|      IMAX| 3.633771275007465|
|  Animation| 3.630434782608696|
|   Mystery|3.6281553398058253|
|   Western|3.6067807351077312|
|   Musical| 3.570217917675545|
+----------+------------------+
```

```
+---------+-----------+
|    genre|num_ratings|
+---------+-----------+
|    Drama|      33798|
|   Comedy|      31487|
|   Action|      24555|
| Thriller|      21221|
|Adventure|      19344|
|  Romance|      14652|
|   Sci-Fi|      13734|
|    Crime|      13335|
|  Fantasy|       9560|
| Children|       7424|
+---------+-----------+
```