Exercise on Cleaning Data with PySpark

1. Import `*` from the `pyspark.sql.types` library.

- Define a new schema using the `StructType` method.
- Define a `StructField` for `name`, `age`, and `city`. Each field should correspond to the correct datatype and not be `nullable`.

2. Load the Data Frame.

- Add the transformation for `F.lower()` to the `Destination Airport` column.
- Drop the `Destination Airport` column from the Data Frame `aa_dfw_df`. Note the time for these operations to complete.

- Show the Data Frame, noting the time difference for this action to complete.

3. View the row count of `df1` and `df2`.

- Combine `df1` and `df2` in a new DataFrame named `df3` with the `union` method.
- Save `df3` to a `parquet` file named `AA_DFW_ALL.parquet`.
- Read the `AA_DFW_ALL.parquet` file and show the count.

4. Import the `AA_DFW_ALL.parquet` file into `flights_df`.

- Use the `createOrReplaceTempView` method to alias the `flights` table.
- Run a Spark SQL query against the `flights` table.

5. Show the distinct `VOTER_NAME` entries.

- Filter `voter_df` where the `VOTER_NAME` is 1-20 characters in length.
- Filter out `voter_df` where the `VOTER_NAME` contains an `_`.
- Show the distinct `VOTER_NAME` entries again.

6. Add a new column called `splits` holding the list of possible names.

- Use the `getItem()` method and create a new column called `first_name`.
- Get the last entry of the `splits` list and create a column called `last_name`.
- Drop the `splits` column and show the new `voter_df`.

7. Add a column to `voter_df` named `random_val` with the results of the `F.rand()` method for any voter with the title **Councilmember**.

- Show some of the DataFrame rows, noting whether the `.when()` clause worked.

8. Add a column to `voter_df` named `random_val` with the results of the `F.rand()` method for any voter with the title **Councilmember**. Set `random_val` to 2 for the **Mayor**. Set any other title to the value 0.

- Show some of the Data Frame rows, noting whether the clauses worked.
- Use the `.filter` clause to find 0 in `random_val`.

9. Edit the `getFirstAndMiddle()` function to return a space separated string of names, except the last entry in the names list.

- Define the function as a user-defined function. It should return a string type.
- Create a new column on `voter_df` called `first_and_middle_name` using your UDF.

- Show the Data Frame.

10. Select the unique entries from the column `VOTER NAME` and create a new DataFrame called `voter_df`.

- Count the rows in the `voter_df` DataFrame.

- Add a ROW_ID column using the appropriate Spark function.

- Show the rows with the 10 highest ROW_IDs.

11. Print the number of partitions on each DataFrame.

- Add a `ROW_ID` field to each DataFrame.

- Show the top 10 IDs in each DataFrame.

12. Determine the highest `ROW_ID` in `voter_df_march` and save it in the variable `previous_max_ID`. The statement `.rdd.max()[0]` will get the maximum ID.

- Add a `ROW_ID` column to `voter_df_april` starting at the value of `previous_max_ID`.
- Show the `ROW_ID`'s from both Data Frames and compare.

13. Cache the unique rows in the `departures_df` DataFrame.

- Perform a count query on `departures_df`, noting how long the operation takes.

- Count the rows again, noting the variance in time of a cached DataFrame.

14. Check the caching status on the `departures_df` DataFrame.

- Remove the `departures_df` DataFrame from the cache.
- Validate the caching status again.

15. Import the `departures_full.txt.gz` file and the `departures_xxx.txt.gz` files into separate DataFrames.

- Run a count on each DataFrame and compare the run times.

16. Check the name of the Spark application instance ('spark.app.name').

- Determine the TCP port the driver runs on ('spark.driver.port').
- Determine how many partitions are configured for joins.
- Show the results.

17. Store the number of partitions in `departures_df` in the variable `before`.

- Change the `spark.sql.shuffle.partitions` configuration to **500** partitions.
- Recreate the `departures_df` DataFrame reading the distinct rows from the departures file.
- Print the number of partitions from before and after the configuration change.

18. Create a new DataFrame `normal_df` by joining `flights_df` with `airports_df`.

- Determine which type of join is used in the query plan.

19. Import the `broadcast()` method from `pyspark.sql.functions`.

- Create a new DataFrame `broadcast_df` by joining `flights_df` with `airports_df`, using the broadcasting.
- Show the query plan and consider differences from the original.

20. Execute `.count()` on the normal DataFrame.

- Execute `.count()` on the broadcasted DataFrame.
- Print the count and duration of the DataFrames noting and differences.

21. Import the file `2015-departures.csv.gz` to a DataFrame. Note the header is already defined.

- Filter the DataFrame to contain only flights with a duration over 0 minutes. Use the index of the column, not the column name (remember to use `.printSchema()` to see the column names / order).

- Add an ID column.
- Write the file out as a JSON document named `output.json`.

22. Import the `annotations.csv.gz` file to a DataFrame and perform a row count. Specify a separator character of **|**.

- Query the data for the number of rows beginning with #.

- Import the file again to a new DataFrame, but specify the comment character in the options to remove any commented rows.

- Count the new DataFrame and verify the difference is as expected.

23. Create a new variable `tmp_fields` using the `annotations_df` DataFrame column `'_c0'` splitting it on the tab character.

- Create a new column in `annotations_df` named `'colcount'` representing the number of fields defined in the previous step.
- Filter out any rows from `annotations_df` containing fewer than 5 fields.
- Count the number of rows in the DataFrame and compare to the `initial_count`.

24. Split the content of the `'_c0'` column on the tab character and store in a variable called `split_cols`.

- Add the following columns based on the first four entries in the variable above: folder, filename, width, height on a DataFrame named `split_df`.
- Add the `split_cols` variable as a column.

25. Create a new function called `retriever` that takes two arguments, the split columns (cols) and the total number of columns (colcount). This function should return a list of the entries that have not been defined as columns yet (i.e., everything after item 4 in the list).

- Define the function as a Spark UDF, returning an Array of strings.
- Create the new column `dog_list` using the UDF and the available columns in the DataFrame.
- Remove the columns `_c0`, `colcount`, and `split_cols`.

26. Rename the `_c0` column to `folder` on the `valid_folders_df` DataFrame.

- Count the number of rows in `split_df`.
- Join the two DataFrames on the folder name, and call the resulting DataFrame `joined_df`. Make sure to broadcast the smaller DataFrame.
- Check the number of rows remaining in the DataFrame and compare.

27. Determine the row counts for each DataFrame.

- Create a DataFrame containing only the invalid rows.
- Validate the count of the new DataFrame is as expected.
- Determine the number of distinct folder rows removed.

28. Select the column representing the dog details from the DataFrame and show the first 10 un-truncated rows.

- Create a new schema as you've done before, using *breed*, *start_x*, *start_y*, *end_x*, and *end_y* as the names. Make sure to specify the proper data types for each field in the schema (any number value is an integer).

29. Create a Python function to split each entry in `dog_list` to its appropriate parts. Make sure to convert any strings into the appropriate types or the DogType will not parse correctly.

- Create a UDF using the above function.
- Use the UDF to create a new column called `dogs`. Drop the previous column in the same command.
- Show the number of dogs in the new column for the first 10 rows.

30. Define a Python function to take a list of tuples (the dog objects) and calculate the total number of "dog" pixels per image.

- Create a UDF of the function and use it to create a new column called `'dog_pixels'` on the DataFrame.
- Create another column, `'dog_percent'`, representing the percentage of `'dog_pixels'` in the image. Make sure this is between 0-100%. Use the string name of the column alone (ie, "columnname" rather than df.columnname).
- Show the first 10 rows with more than 60% `'dog_pixels'` in the image. Use a SQL style string for this (ie, 'columnname > ____').