Data Exploration in Spark

By the end of this activity, you will be able to perform the following in Spark:

1. Read CSV files into Spark Dataframes.
2. Generate summary statistics.
3. Compute correlation coefficients between two columns.

NOTE: If you have completed the previous course of this specialization *Big Data Integration and Processing*, you should already have downloaded the *jupyter-coursera* container. In this case, you can start your container as before and skip to step 4. Otherwise, follow steps 1-3.

Step 1. Open a terminal shell. Open your local terminal shell.

Step 2. Start Docker. Make sure to start Docker by opening Docker Desktop.

Once you have started Docker, go back to your terminal and run *docker pull pramonettivega/jupyter-coursera:latest* to pull a Docker image for this activity.

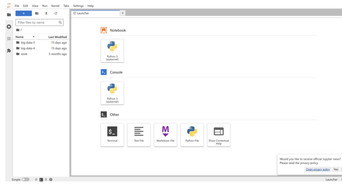
```
docker pull pramonettivega/jupyter-coursera:latest
```

Step 3. Run the container and access Jupyter. Run *docker run -p 8888:8888 pramonettivega/jupyter-coursera* to start the container.

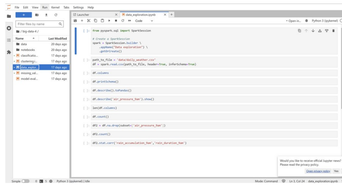
```
docker run --name Jupyter-coursera -p 8888:8888 pramonettivega/jupyter-coursera
```

When Jupyter starts running, click on the port to access JupyterLab in your browser:

Once you access, you should the following page:



Step 4. Open your notebook. Double click on the *big-data-4* folder and then open the *data_exploration.ipynb* notebook.



Step 5. Load data into Spark DataFrame. First, we need to import the *SparkSession* class:

```
[1]: from pyspark.sql import SparkSession

     # Create a SparkSession
     spark = SparkSession.builder \
         .appName("Data exploration") \
         .getOrCreate()
```

Then we read the weather data into a DataFrame:

```
[2]: df = spark.read.csv('data/daily_weather.csv', header=True, inferSchema=True)
```

The first argument specifies the location of the *daily_weather.csv* file, the second argument says the first line in *daily_weather.csv* is the header, and the third argument says to infer the data types.

Step 6. Look at data columns and types. We can see the *columns* in the DataFrame by looking at the *columns* attribute:

```
[3]: df.columns
```

```
['number',
 'air_pressure_9am',
 'air_temp_9am',
 'avg_wind_direction_9am',
 'avg_wind_speed_9am',
 'max_wind_direction_9am',
 'max_wind_speed_9am',
 'rain_accumulation_9am',
 'rain_duration_9am',
 'relative_humidity_9am',
 'relative_humidity_3pm']
```

The data type for each column by calling *printSchema()*:

```
[4]: df.printSchema()
```

```
root
 |-- number: integer (nullable = true)
 |-- air_pressure_9am: double (nullable = true)
 |-- air_temp_9am: double (nullable = true)
 |-- avg_wind_direction_9am: double (nullable = true)
 |-- avg_wind_speed_9am: double (nullable = true)
 |-- max_wind_direction_9am: double (nullable = true)
 |-- max_wind_speed_9am: double (nullable = true)
 |-- rain_accumulation_9am: double (nullable = true)
 |-- rain_duration_9am: double (nullable = true)
 |-- relative_humidity_9am: double (nullable = true)
 |-- relative_humidity_3pm: double (nullable = true)
```

Step 7. Print summary statistics. We can print the summary statistics for all the columns using the *describe()* method:

```
df.describe().toPandas().transpose()
```

	0	1	2	3	4
summary	count	mean	stddev	min	max
number	1095	547.0	316.24357700987383	0	1094
air_pressure_9am	1092	918.8825513138094	3.184161180386833	907.9900000000024	929.3200000000012
air_temp_9am	1090	64.93300141287072	11.175514003175877	36.752000000000685	98.90599999999992
avg_wind_direction_9am	1091	142.23555107005759	69.13785928889189	15.500000000000046	343.4
avg_wind_speed_9am	1092	5.508284243225493	4.552813465531185	0.693451399999974	23.554978199999763
max_wind_direction_9am	1092	148.95351796516923	67.23801294602953	28.899999999999991	332.1999999999993
max_wind_speed_9am	1091	7.019513529175272	5.598209170780958	1.1855782000000479	29.840779599999996
rain_accumulation_9am	1089	0.20307895252511126	1.5939521253574893	0.0	24.019999999999907
rain_duration_9am	1092	294.10805227756142	1598.0787786601481	0.0	17704.0
relative_humidity_9am	1095	34.24140205923536	25.472066802250055	6.090000000010012	92.62000000000002
relative_humidity_3pm	1095	35.344727148252898	22.524079453587273	5.300000000006855	92.25000000000003

We can also see the summary statistics for just one column:

```
[7]: df.describe('air_pressure_9am').show()
```

```
+-------+-----------------+
|summary| air_pressure_9am|
+-------+-----------------+
|  count|             1092|
|   mean|918.8825513138094|
| stddev|3.184161180386833|
|    min|907.9900000000024|
|    max|929.3200000000012|
+-------+-----------------+
```

Let's count the number of columns and rows in the DataFrame:

```
[8]: len(df.columns)
```

```
[8]: 11
```

```
[9]: df.count()
```

```
[9]: 1095
```

The number of rows in the DataFrame is 1095, but the summary statistics for *air_pressure_9am* says there are only 1092 rows. These are different since 1095 - 1092 = 3 rows have missing values.

Step 8. Drop rows with missing values. Let's drop the rows with missing values in the *air_pressure_9am* column:

```
[10]: df2 = df.na.drop(subset=['air_pressure_9am'])
```

Now let's see the total number of rows:

```
[11]: df2.count()
```

```
[11]: 1092
```

The total number of rows and number of rows in the summary statistics are now the same.

Step 9. Compute correlation between two columns. We can compute the correlation between two columns in a DataFrame by using the *corr()* method. Let's compute the correlation between *rain_accumulation_9am* and *rain_duration_9am*:

```
[12]: df2.stat.corr('rain_accumulation_9am','rain_duration_9am')
```

```
[12]: 0.7298253479609021
```

Go to next item ✓ Completed