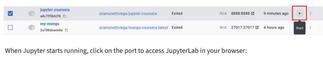
Handling Missing Values in Spark

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

1. Remove rows containing missing values from a DataFrame.
2. Impute missing values with the average value.

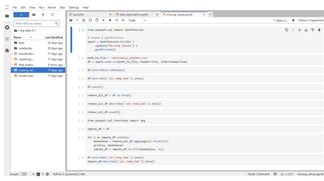
For this activity, you should have completed the creation of the JupyterLab container. If not follow, Steps 1-3 on the previous activity *Hand On: Data Exploration in Spark*, and then come back to Step 2 of this activity.

Step 1. Start the container. Open Docker Desktop and start your *jupyter-coursera* container.



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

Step 2. Step 2. Open your notebook. Once you're in JupyterLab, go to the *big-data-4* folder and open the *missing_values.ipynb* notebook.



Step 3. Load classes and weather data. Run the first cell in the notebook to load the *SparkSession* class, create an instance of *SparkSession*, and read the weather data into a DataFrame.

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

     # Create a SparkSession
     spark = SparkSession.builder \
         .appName("Missing Values") \
         .getOrCreate()
     df = spark.read.csv('data/daily_weather.csv', header=True, inferSchema=True)
```

Step 4. Print summary statistics. We can print the summary statistics for all the columns using *describe()*:

```
[2]: df.describe().toPandas().transpose()
```

[3]:

	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	3.1845611803386633	907.9900000000024	929.3200000000012
air_temp_9am	1090	64.93300141287072	11.175514003175877	36.752000000000685	98.90599999999992
avg_wind_direction_9am	1091	142.23555107005759	69.13789528889189	15.500000000000046	343.4
avg_wind_speed_9am	1092	5.508284244225493	4.552813465531785	0.693453139999974	23.554978199999763
max_wind_direction_9am	1092	148.95351796516923	67.23801294602953	28.899999999999991	312.1999999999993
max_wind_speed_9am	1091	7.019513529175272	5.598209170780958	1.1855782000000479	29.840779599999996
rain_accumulation_9am	1089	0.2030789522521126	1.593952152553748893	0.0	24.019999999999907
rain_duration_9am	1092	294.1080522756142	1598.0787786601481	0.0	17704.0
relative_humidity_9am	1095	34.24144205923536	25.472066802250055	6.090000000010312	92.62000000000002
relative_humidity_3pm	1095	35.34472714825898	22.524079453587273	5.300000000006855	92.2500000000003

Let's just look at the statistics for the air temperature at 9am:

```
[4]: df.describe(['air_temp_9am']).show()
```

```
+-------+------------------+
|summary|      air_temp_9am|
+-------+------------------+
|  count|              1090|
|   mean| 64.93300141287072|
| stddev|11.175514003175877|
|    min|36.752000000000685|
|    max| 98.90599999999992|
+-------+------------------+
```

This says that there are 1090 rows. The total number of rows in the DataFrame is 1095:

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

```
[5]: 1095
```

This means that 5 of the rows in the *air_temp_9am* column are missing values.

Step 5. Remove missing values. We can drop all the rows missing a value in any calling using *na.drop()*:

```
[6]: remove_all_df = df.na.drop()
```

Let's look at the summary statistics for *air_temp_9am* with the missing values dropped:

```
[7]: remove_all_df.describe(['air_temp_9am']).show()
```

```
+-------+------------------+
|summary|      air_temp_9am|
+-------+------------------+
|  count|              1064|
|   mean| 65.02260949558733|
| stddev|11.168033449415704|
|    min|36.752000000000685|
|    max| 98.90599999999992|
+-------+------------------+
```

We can see that the mean and standard deviation is close to the original values: mean is 64.933 vs. 65.022, and standard deviation is 11.175 vs. 11.168.

The count is 1064, which means that 1095 - 1064 = 31 rows were dropped. We can see this agrees with the total number of rows in the new DataFrame:

```
[8]: remove_all_df.count()
```

```
[8]: 1064
```

Step 6. Impute missing values. Instead of removing rows containing missing values, let's replace the values with the mean value for that column. First, we'll load the *avg* function and make a copy of the original DataFrame:

```
[8]: from pyspark.sql.functions import avg
     impute_df = df
```

Next, we'll iterate through each column in the DataFrame: compute the mean value for that column and then replace any missing values in that column with the mean.

```
[9]: for x in impute_df.columns:
         meanValue = remove_all_df.agg(avg(x)).first()[0]
         print(x, meanValue)
         impute_df = impute_df.na.fill(meanValue, [x])
```

```
number 545.0018769992481
air_pressure_9am 918.9031798641051
air_temp_9am 65.02260949558733
avg_wind_direction_9am 142.30675564934037
avg_wind_speed_9am 5.48579305071369
max_wind_direction_9am 148.48042413321315
max_wind_speed_9am 6.999713658875691
rain_accumulation_9am 0.18202347650615522
rain_duration_9am 266.3936973996037
relative_humidity_9am 34.07743985327709
relative_humidity_3pm 35.14838093290533
```

The *agg()* function performs an aggregate calculation on the DataFrame and *avg(x)* specifies to compute the mean on column *x*. The *agg()* function returns a DataFrame, *first()* returns the first *Row*, and *[0]* gets the first value.

The last line of code uses *na.fill()* to replace the missing values with the mean value (first argument) in column *x* (second argument).

The output of executing this cell prints the mean values for each column and we can see the mean value for *air_temp_9am* is the same as the mean when we removed all the missing values in step 4, *i.e.*, 65.022.

Step 7. Print imputed data summary statistics. Let's call *describe()* to show the summary statistics for the original and imputed *air_temp_9am*:

The count for the imputed data is larger since the 5 rows with missing data have replaced with real values. Additionally, we can see that the means are close, but not equal, and this is probably due to round-off error.

```
[10]: df.describe(['air_temp_9am']).show()
      impute_df.describe(['air_temp_9am']).show()
```

```
+-------+------------------+
|summary|      air_temp_9am|
+-------+------------------+
|  count|              1090|
|   mean| 64.93300141287072|
| stddev|11.175514003175877|
|    min|36.752000000000685|
|    max| 98.90599999999992|
+-------+------------------+

+-------+------------------+
|summary|      air_temp_9am|
+-------+------------------+
|  count|              1095|
|   mean| 64.93341058219818|
| stddev| 11.14994819992023|
|    min|36.752000000000685|
|    max| 98.90599999999992|
+-------+------------------+
```

Step 8. Exiting the container. To exit JupyterLab, simply close the tab in your browser. To stop the container, go to Docker Desktop and click on the *stop* button. We recommend not to delete the container, as this container will be used for multiple activities across this specialization.

Go to next item ✓ Completed