Data Exploration in Spark

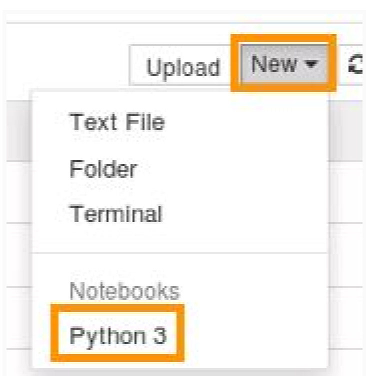
By the end of this activity, you will be able to perform the following in Spark:

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

In this activity, you will be programming in a Jupyter Python Notebook. If you have not already started the Jupyter Notebook server, see the instructions in the Reading *Instructions for Starting Jupyter*.

Step 1. **Create new Jupyter Python Notebook.** Open a web browser by clicking on the web browser icon at the top of the toolbar:

Create a new Python Notebook by clicking on *New,* and then click on *Python 3:*



Step 2. **Load data into Spark DataFrame.** First, we need to import the *SQLContext* class:

```
In [1]: from pyspark.sql import SQLContext
```

Next, we create an SQLContext:

```
In [2]: sqlContext = SQLContext(sc)
```

Then we read the weather data into a DataFrame:

```
In [3]: df = sqlContext.read.load('file:///home/cloudera/Downloads/big-data-4/daily_weather.csv',
                                  format='com.databricks.spark.csv',
                                  header='true',inferSchema='true')
```

The first argument specifies the URL to the *daily_weather.csv* file, the second argument specifies the *spark-csv*format, the third argument says the first line in *daily_weather.csv* is the header, and the fourth argument says to infer the data types.

We use *spark-csv* to read CSV data directly into a Spark DataFrame. In Spark 1.x (the version on the Cloudera VM), *spark-csv* is an external package, but *spark-csv* is integrated with Spark 2.x.

Step 4. **Look at data columns and types.** We can see the columns in the DataFrame by looking at the *columns*attribute:

```
In [4]: df.columns

Out[4]: ['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()*:

```
In [5]: 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 5. **Print summary statistics**. We can print the summary statistics for all the columns using the *describe()*method:

```
In [6]: df.describe().toPandas().transpose()
```

Out[6]:

	0	1	2	3	4
summary	count	mean	stddev	min	max
number	1095	547.0	316.24357700987383	0	1094
air_pressure_9am	1092	918.8825513138097	3.1841611803868353	907.9900000000024	929.32000000
air_temp_9am	1090	64.93300141287075	11.175514003175877	36.752000000000685	98.905999999
avg_wind_direction_9am	1091	142.23551070057584	69.13785928889183	15.500000000000046	343.4
avg_wind_speed_9am	1092	5.50828424225493	4.552813465531715	0.69345139999974	23.554978199
max_wind_direction_9am	1092	148.9535179651692	67.23801294602951	28.89999999999991	312.19999999
max_wind_speed_9am	1091	7.019513529175272	5.59820917078096	1.1855782000000479	29.840779599
rain_accumulation_9am	1089	0.20307895225211126	1.5939521253574904	0.0	24.019999999
rain_duration_9am	1092	294.1080522756142	1598.078778660148	0.0	17704.0
relative_humidity_9am	1095	34.24140205923539	25.472066802250044	6.090000000001012	92.620000000
relative_humidity_3pm	1095	35.34472714825902	22.52407945358728	5.3000000000006855	92.250000000

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

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

+-------+-----------------+
|summary| air_pressure_9am|
+-------+-----------------+
|  count|             1092|
|   mean| 918.8825513138097|
| stddev|3.1841611803868353|
|    min| 907.9900000000024|
|    max| 929.3000000000012|
+-------+-----------------+
```

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

```
In [8]: len(df.columns)
Out[8]: 11
```

```
In [9]: df.count()
Out[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 6. **Drop rows with missing values.** Let's drop the rows with missing values in the *air_pressure_9am* column:

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

Now let's see the total number of rows:

```
In [11]: df2.count()
Out[11]: 1092
```

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

Step 7. **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*:

In [12]:
```
df2.stat.corr("rain_accumulation_9am", "rain_duration_9am")
```
Out[12]: 0.7298253479609015