# Analyzing Sensor Data with Spark Streaming

By the end of this activity, you will be able to:

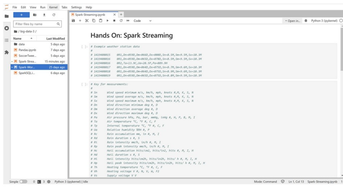
1. Read streaming data into Spark
2. Create and apply computations over a sliding window of data

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: Exploring Pandas DataFrames*, 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. Open your notebook.** Once you're in JupyterLab, go to the *big-data-3* folder and open the *Spark-Streaming.ipynb* notebook:




**Step 3. Look at sensor format and measurement types**. The first cell in the notebook gives an example of the streaming measurements coming from the weather station:

```
[ ]: # Example weather station data
     #
     # 1419408015    0R1,Dn=059D,Dm=066D,Dx=080D,Sn=8.5M,Sm=9.5M,Sx=10.3M
     # 1419408016    0R1,Dn=059D,Dm=065D,Dx=078D,Sn=8.5M,Sm=9.5M,Sx=10.3M
     # 1419408016    0R2,Ta=13.9C,Ua=28.5P,Pa=889.9H
     # 1419408017    0R1,Dn=059D,Dm=064D,Dx=075D,Sn=8.7M,Sm=9.6M,Sx=10.3M
     # 1419408018    0R1,Dn=059D,Dm=064D,Dx=075D,Sn=8.9M,Sm=9.6M,Sx=10.3M
     # 1419408019    0R1,Dn=059D,Dm=065D,Dx=075D,Sn=8.8M,Sm=9.5M,Sx=10.3M
```

Each line contains a timestamp and a set of measurements. Each measurement has an abbreviation, and for this exercise, we are interested in the average wind direction, which is *Dm*. The next cell lists the abbreviations used for each type of measurement:

```
[ ]: # Key for measurements:
     #
     # Sn      Wind speed minimum m/s, km/h, mph, knots #,M, K, S, N
     # Sm      Wind speed average m/s, km/h, mph, knots #,M, K, S, N
     # Sx      Wind speed maximum m/s, km/h, mph, knots #,M, K, S, N
     # Dn      Wind direction minimum deg #, D
     # Dm      Wind direction average deg #, D
     # Dx      Wind direction maximum deg #, D
     # Pa      Air pressure hPa, Pa, bar, mmHg, inHg #, H, P, B, M, I
     # Ta      Air temperature °C, °F #, C, F
     # Tp      Internal temperature °C, °F #, C, F
     # Ua      Relative humidity %RH #, P
     # Rc      Rain accumulation mm, in #, M, I
     # Rd      Rain duration s #, S
     # Ri      Rain intensity mm/h, in/h #, M, I
     # Rp      Rain peak intensity mm/h, in/h #, M, I
     # Hc      Hail accumulation hits/cm2, hits/in2, hits #, M, I, H
     # Hd      Hail duration s #, S
     # Hi      Hail intensity hits/cm2h, hits/in2h, hits/ h #, M, I, H
     # Hp      Hail peak intensity hits/cm2h, hits/in2h, hits/ h #, M, I, H
     # Th      Heating temperature °C, °F #, C, F
     # Vh      Heating voltage V #, N, V, W, F2
     # Vs      Supply voltage V V
     # Vr      3.5 V ref. voltage V V
```

The third cell defines a function that parses each line and returns the average wind direction (*Dm*). Run this cell:

```
[1]: import re
     def parse(line):
         match = re.search(r"Dm=(\d+)", line)
         if match:
             val = match.group(1)
             return int(val)
         return None
```

**Step 4. Import and create SparkSession.** Next, we will import and create a new instance of SparkSession:

```
[2]: from pyspark.sql import SparkSession
     from pyspark.sql.functions import *
     from pyspark.sql.types import IntegerType, TimestampType
     from pyspark.sql.window import Window

     spark = SparkSession.builder.appName("StructuredStreaming").getOrCreate()
```

We also import a set of functions that are part of Spark's Structured Streaming. These functions are built on top of SparkSQL and are designed to interact with streaming data.

**Step 5. Create DStream of weather data**. Let's open a connection to the streaming weather data:

```
[3]: lines = (
         spark
         .readStream
         .format("socket")
         .option("host", "rtd.hpwren.ucsd.edu")
         .option("port", 12024)
         .load()
     )
     lines
```

```
[3]: DataFrame[value: string]
```

Instead of 12024, you may find that port 12028 or 12020 works instead. This create a new variable *lines* to be a Spark DataFrame that streams the lines of output from the weather station.

**Step 6. Read measurement**. Next, let's read the average wind speed from each line and store it in a new DataFrame *parsed_lines:*

```
[4]: parsed_lines =lines.withColumn("parsed", udf(parse, IntegerType())("value"))
     parsed_lines
```

```
[4]: DataFrame[value: string, parsed: int]
```

This line uses *flatMap()* to iterate over the *lines* DStream, and calls the *parse()* function we defined above to get the average wind speed.

**Step 7. Create sliding window of data**. Now we will create a windowed DataFrame that will return values for the last 10 seconds and create a couple of new columns, the max and min wind direction over that time span:

```
[5]: windowed_data = (
         parsed_lines
         .withColumn("time", current_timestamp())
         .groupBy(window("time", "10 seconds"))
         .agg(collect_list("parsed").alias("wind_direction"))
         .withColumn("max_val", array_max("wind_direction"))
         .withColumn("min_val", array_min("wind_direction"))
         .select("wind_direction", "max_val", "min_val")
     )
     windowed_data
```

```
[5]: DataFrame[wind_direction: array<int>, max_val: int, min_val: int]
```

**Step 8. Start the stream processing**. We define the query to trigger every 10 seconds and call *start()* to begin the processing:

```
[6]: query = (
         windowed_data
         .writeStream
         .outputMode("update")
         .trigger(processingTime="10 seconds")
         .foreachBatch(lambda batch_df, epoch_id: batch_df.show(truncate=False))
         .start()
     )
```

```
+--------------+-------+-------+
|wind_direction|max_val|min_val|
+--------------+-------+-------+
+--------------+-------+-------+

+------------------------------------+-------+-------+
|wind_direction                      |max_val|min_val|
+------------------------------------+-------+-------+
|[317, 314, 311, 314, 313, 312, 311]|317    |311    |
+------------------------------------+-------+-------+

+------------------------------------------+-------+-------+
|wind_direction                            |max_val|min_val|
+------------------------------------------+-------+-------+
|[309, 308, 308, 309, 313, 314, 312, 312]|314    |308    |
+------------------------------------------+-------+-------+

+----------------------------------------------+-------+-------+
|wind_direction                                |max_val|min_val|
+----------------------------------------------+-------+-------+
|[314, 313, 315, 317, 317, 316, 315, 314, 311, 311]|317    |311    |
+----------------------------------------------+-------+-------+

+----------------------------------------------------+-------+-------+
|wind_direction                                      |max_val|min_val|
+----------------------------------------------------+-------+-------+
|[314, 320, 316, 321, 322, 325, 326, 326, 326, 324]|326    |314    |
+----------------------------------------------------+-------+-------+
```

The sliding window contains ten seconds worth of data and slides every ten seconds. In the beginning, the number of values in the windows are increasing as the data accumulates until the size stays (approximately) the same.

When we are done, call *stop()* on the StreamingContext:

```
[7]: query.stop()
```

**Step 9. 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

Like Dislike Report an issue