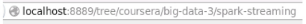
By the end of this activity, you will be able to:

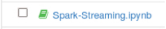
1. Read streaming data into Spark
2. Create and apply computations over a sliding window of data

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

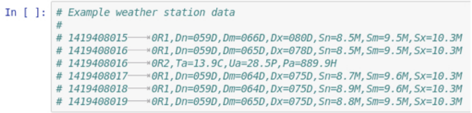
Navigate to *localhost:8889/tree/Downloads/big-data-3/spark-streaming*:


localhost:8889/tree/coursera/big-data-3/spark-streaming

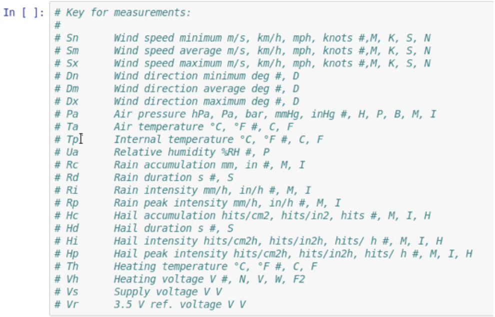
Open the Spark Streaming Notebook by clicking on *Spark-Streaming.ipynb*:


□ Spark-Streaming.ipynb

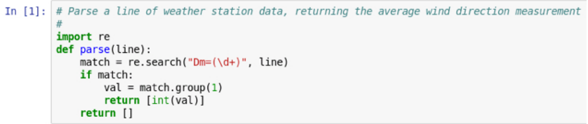
Step 2. **Look at sensor format and measurement types**. The first cell in the notebook gives an example of the streaming measurements coming from the weather station:


```
In [ ]: # 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
```

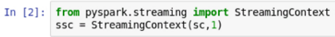
Each line contains a timestamp and a set of measurements. Each measurement has an abbreviation, and for this exercise, we are interested in the average wind direction, which is *Dm*. The next cell lists the abbreviations used for each type of measurement:


```
In [ ]: # 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:

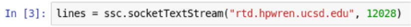

```
In [1]: # Parse a line of weather station data, returning the average wind direction measurement
        #
        import re
        def parse(line):
            match = re.search("Dm=(\d+)", line)
            if match:
                val = match.group(1)
                return [int(val)]
            return []
```

Step 3. **Import and create streaming context.** Next, we will import and create a new instance of Spark's StreamingContext:


```
In [2]: from pyspark.streaming import StreamingContext
        ssc = StreamingContext(sc,1)
```

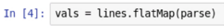
Similar to the SparkContext, the StreamingContext provides an interface to Spark's streaming capabilities. The argument *sc* is the SparkContext, and 1 specifies a batch interval of one second.

Step 4. **Create DStream of weather data**. Let's open a connection to the streaming weather data:


```
In [3]: lines = ssc.socketTextStream("rtd.hpwren.ucsd.edu", 12028)
```

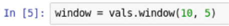
Instead of 12028, you may find that port 12020 works instead. This create a new variable *lines* to be a Spark DStream that streams the lines of output from the weather station.

Step 5. **Read measurement**. Next, let's read the average wind speed from each line and store it in a new DStream *vals:*


```
In [4]: vals = lines.flatMap(parse)
```

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 6. **Create sliding window of data**. We can create a sliding window over the measurements by calling the *window()* method:


```
In [5]: window = vals.window(10, 5)
```

This create a new DStream called *window* that combines the ten seconds worth of data and moves by five seconds.

Step 7. **Define and call analysis function**. We would like to find the minimum and maximum values in our window. Let's define a function that prints these values for an RDD:

```
In [6]: def stats(rdd):
            print(rdd.collect())
            if rdd.count() > 0:
                print("max = {}, min = {}".format(rdd.max(), rdd.min()))
```

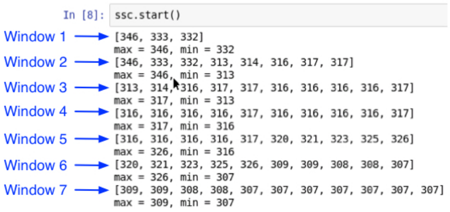
This function first prints the entire contents of the RDD by calling the *collect()* method. This is done to demonstrate the sliding window and would not be practical if the RDD was containing a large amount of data. Next, we check if the size of the RDD is greater than zero before printing the maximum and minimum values.

Next, we call the *stats()* function for each RDD in our sliding window:

```
In [7]: window.foreachRDD(lambda rdd: stats(rdd))
```

This line calls the *stats()* function defined above for each RDD in the DStream window.

Step 8. **Start the stream processing**. We call *start()* on the StreamingContext to begin the processing:



```
In [8]: ssc.start()

[346, 333, 332]
max = 346, min = 332
[346, 333, 332, 313, 314, 316, 317, 317]
max = 346, min = 313
[313, 314, 316, 317, 317, 316, 316, 316, 316, 317]
max = 317, min = 313
[316, 316, 316, 316, 317, 316, 316, 316, 316, 317]
max = 317, min = 316
[316, 316, 316, 316, 317, 320, 321, 323, 325, 326]
max = 326, min = 316
[320, 321, 323, 325, 326, 309, 309, 308, 308, 307]
max = 326, min = 307
[309, 309, 308, 308, 307, 307, 307, 307, 307, 307]
max = 309, min = 307
```

The sliding window contains ten seconds worth of data and slides every five seconds. In the beginning, the number of values in the windows are increasing as the data accumulates, and after Window 3, the size stays (approximately) the same. Since the window slides half as often as the size of the window, the second half of a window becomes the first half of the next window. For example, the second half of Window 5 is 310, 321, 323, 325, 326, which becomes the first half of Window 6.

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

```
In [9]: ssc.stop()
```

Mark as completed