**Spark Streaming**

**Extract Data**

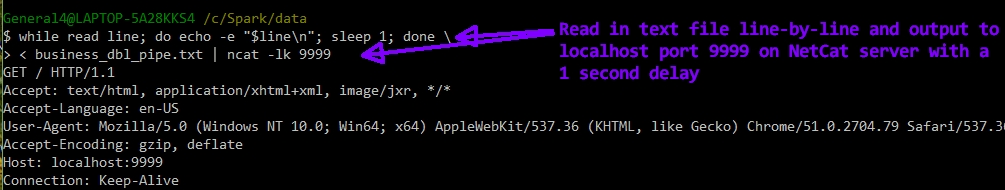
I did not need to extract any additional data to work with Spark Streaming. My existing data was used as an input stream. How I accomplished this is explained in the next section.

**Load Data**

Spark Streaming requires a DStream Source that provides data that updates or continually changes., I chose to use the **socketTextStream** method from the Streaming API to create a DStream from an input TCP source. To provide a streaming input source, I used a simple shell script that reads the lines of a Yelp data file, business\_dbl\_pipe.txt, and every second sends a line to port 9999 of localhost on a NetCat server. The one second interval is the input data’s velocity. Port 9999 will provide input for my Spark Streaming script. This stream of data can be thought of as new user input data being captured from a website.

$ while read line; do echo -e "$line\n"; sleep 1; done \

> < business\_dbl\_pipe.txt | ncat -lk 9999





When doing batch processing with Spark, you do not need to instantiate a SparkContext since it is done automatically for you. This is not true with SparkStreaming; you must explicitly create a StreamingContext. The following lines of code show how my script instantiates a StreamingContext and creates a connection to the data stream. Notice that the StreamingContext constructor has a numeric batch duration argument, which is the number of seconds that streaming data will be split into batches.

**ssc = StreamingContext(sc, 10)** # Instantiate a StreamingContext – uses an existing SparkContex, sc

**lines = ssc.socketTextStream('localhost', 9999)** # Connect to the data stream

**Query**

For Spark Streaming, I used the same query as Spark Batch.

Problem: **Find average review rating for all businesses in a specific zip code**

**import** re

**from** pyspark**.**streaming **import** StreamingContext

ssc **=** StreamingContext**(**sc**,** 10**)**

lines **=** ssc**.**socketTextStream**(**'localhost'**,** 9999**)**

bus2 **=** lines**.**filter**(lambda** line**:** len**(**line**)** **>** 0**)** \

**.**map**(lambda** x**:** x**.**split**(**'||'**))** \

**.**map**(lambda** x**:** **(**x**[**0**],**x**[**1**],**x**[**2**],**x**[**3**],**x**[**4**],**x**[**5**],**x**[**6**].**replace**(**'"'**,**''**),**x**[**7**],**x**[**8**],** \

x**[**9**].**replace**(**'"'**,**''**),**x**[**10**],**x**[**11**],**x**[**12**]))** \

**.**map**(lambda** x**:** **(**x**[**0**],**x**[**1**],**x**[**2**],**x**[**3**],**x**[**4**],**x**[**5**],**x**[**6**],**x**[**7**],**x**[**8**],**float**(**x**[**9**]),**x**[**10**],**x**[**11**],**x**[**12**]))** \

**.**map**(lambda** x**:** **(**x**[**6**],** x**[**9**]))**

postalCodeCount **=** bus2**.**map**(lambda** x**:** **(**x**[**0**],** 1**))**

numReviewsForPostalCode **=** postalCodeCount**.**reduceByKey**(lambda** v1**,** v2**:** v1 **+** v2**)**

sum\_stars **=** bus2**.**reduceByKey**(lambda** v1**,** v2**:** v1 **+** v2**)**

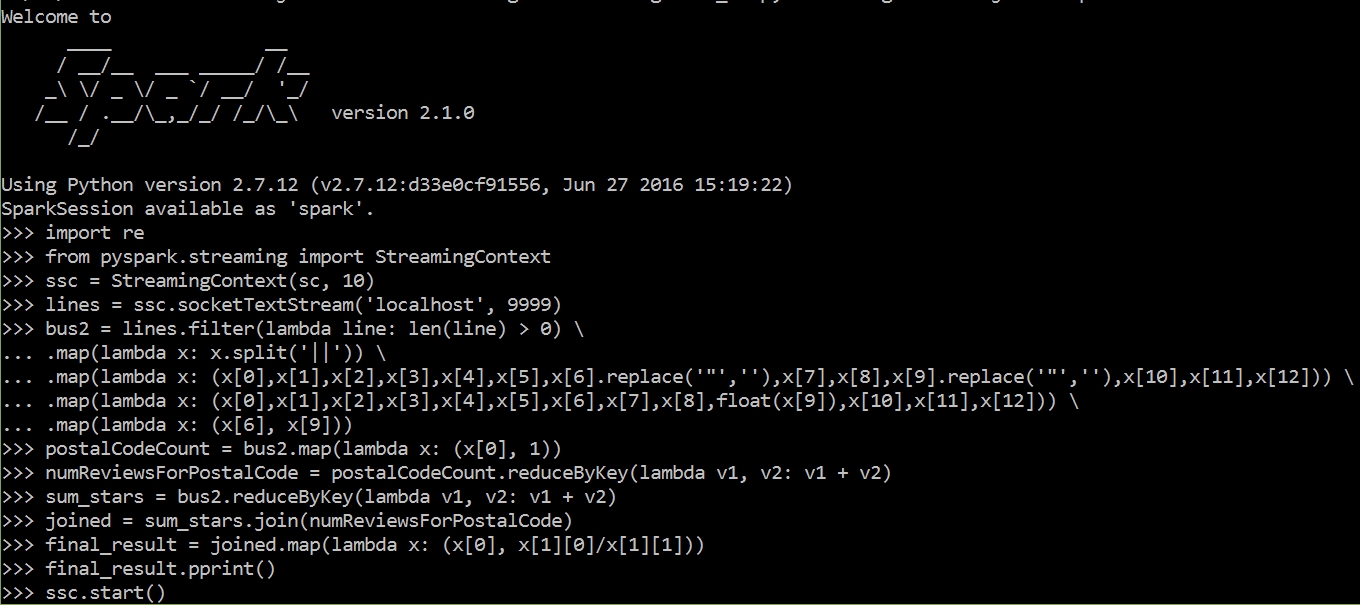
joined **=** sum\_stars**.**join**(**numReviewsForPostalCode**)**

final\_result **=** joined**.**map**(lambda** x**:** **(**x**[**0**],** x**[**1**][**0**]/**x**[**1**][**1**]))**

final\_result**.**pprint**()**

ssc**.**start**()**

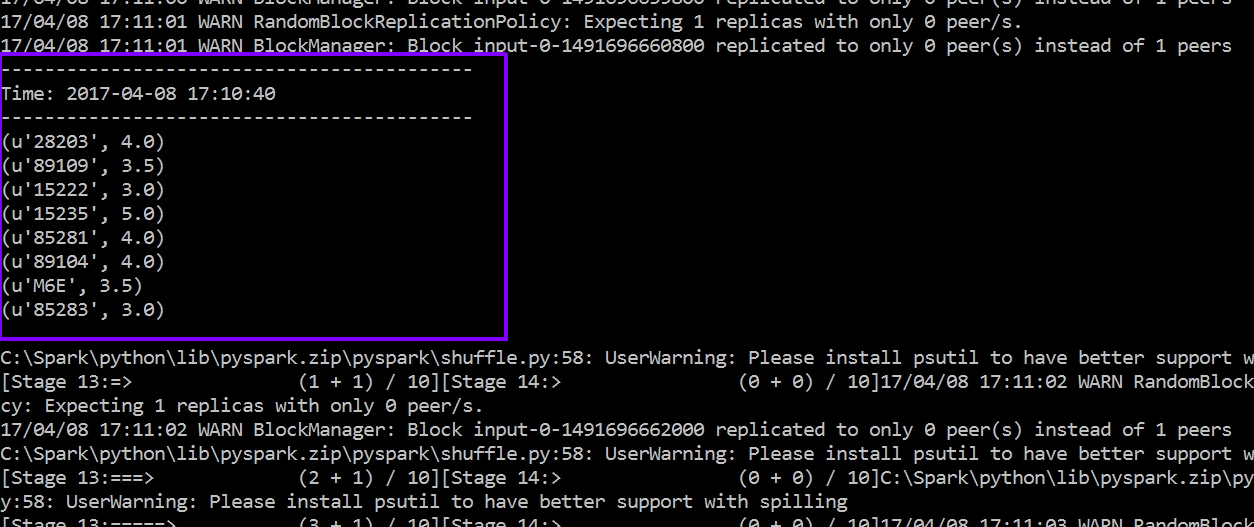
Notice that my solution is almost identical to my batch script. The only difference is the code explicitly creates a StreamingContext and makes a connection to a TCP socket as an input data stream. This is a stateless Spark Streaming solution to my problem query. This means that each batch of a batch interval is processed independently of any other batches with a stream. To make a state DStream, I would use the **updateStateByKey( aFunction )** transformation method. This method returns a new DStream where the state for each key is updated by applying the function that is specified as an argument.

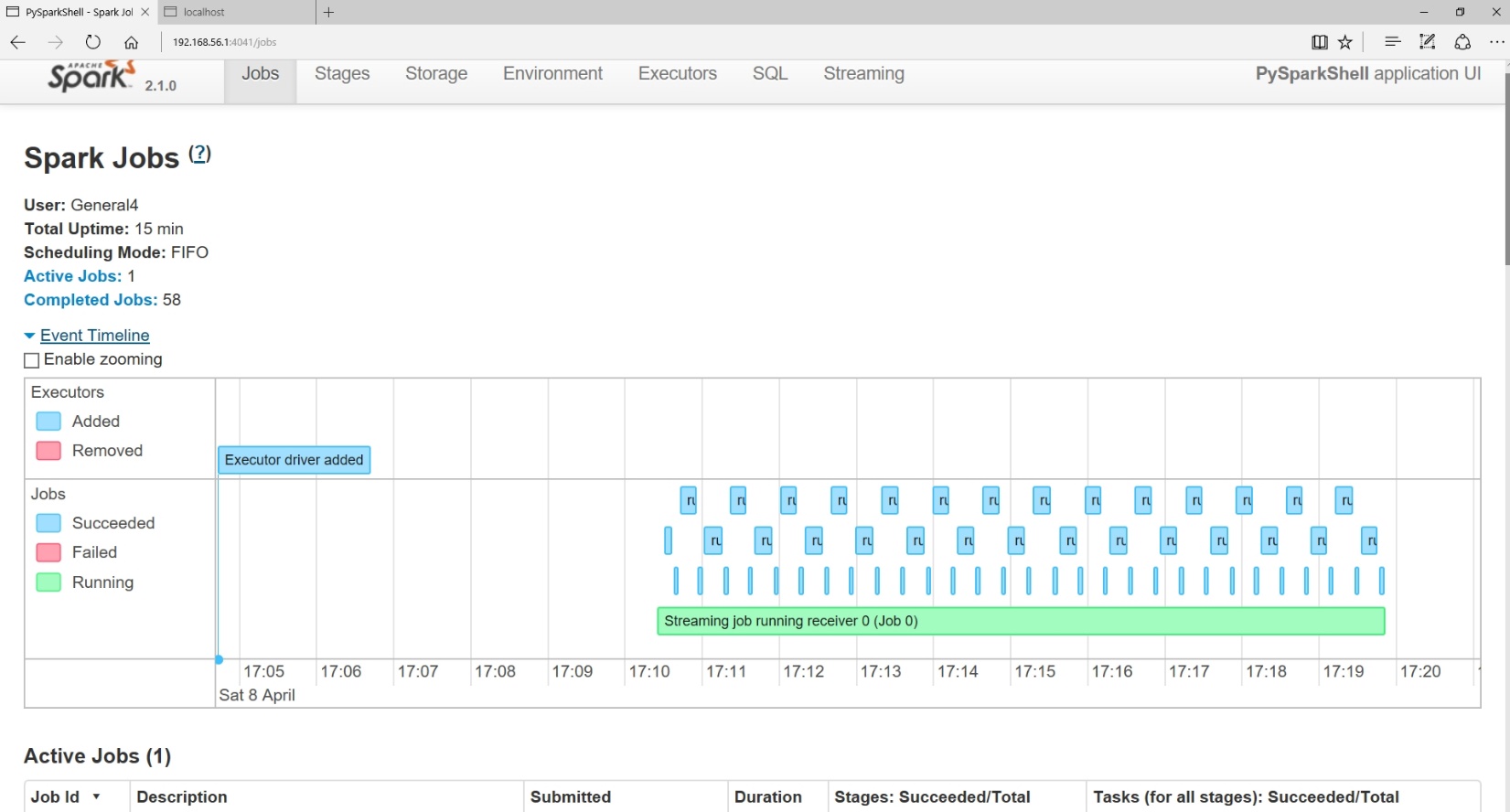


I ran my streaming script from my local Spark installation and not on Databricks. When using local mode, you must specify at least two worker threads. That is what the 2 means in the following command.

**C:\Spark>pyspark --master local[2]**

After entering **ssc.start( )** and pressing enter, I started the flow of input data by opening another command window and entering the script that I mentioned above in the Load Data section. With this data streaming, my script captures lines and performs work on them. The next image is just one of many captured results. As Spark runs the script, the output is very verbose.





**Update**

**< TO DO >**

**Optimize Quality**

**< TO DO >**

**Optimize Performance**

