# Parallel and Distributed Computing
# Lab8
# Spark Streaming

In this lab you will work with Spark's Structured Streaming API by streaming Twitter tweets and compiling a windowed list of trending hashtags.

**Streaming data setup**

We will be using Twitter tweets that occurred during the 2018 FIFA World Cup (https://www.kaggle.com/rgupta09/world-cup-2018-tweets) as our data for this lab. I have downloaded the large file of tweets and placed it on the course site. We will be simulating a stream by creating a large number of files from this large file and placing them in a directory on the dbfs. We will then use Spark Streaming to read its input from this file one file at a time as our simulated stream. So, the first step will be to load in the single large data and create the small files.

You can check out the Kaggle site to get the details on the fields in each line of tweet data. But to help you get to the streaming portion, I'll provide the Schema you should use for the CSV file:

```
fifaSchema = StructType( \
                    [StructField('ID', LongType(), True), \
                     StructField('lang', StringType(), True), \
                     StructField('Date', TimestampType(), True), \
                     StructField('Source', StringType(), True), \
                     StructField('len', LongType(), True), \
                     StructField('Orig_Tweet', StringType(), True), \
                     StructField('Tweet', StringType(), True), \
                     StructField('Likes', LongType(), True), \
                     StructField('RTs', LongType(), True), \
                     StructField('Hashtags', StringType(), True), \
                     StructField('UserMentionNames', StringType(), True), \
                     StructField('UserMentionID', StringType(), True), \
                     StructField('Name', StringType(), True), \
                     StructField('Place', StringType(), True), \
                     StructField('Followers', LongType(), True), \
                     StructField('Friends', LongType(), True), \
                    ])
```

There will be lines in the data that do not fully conform to this Schema, so just drop those lines from your dataset. Then, keeping the full set of data as read, partition it to create many smaller

partitions. I created 50, but your code will run faster if you have less for testing. Note that for testing purposes, you might want to order by the date when you partition so the tweets are in order. But for your general streaming run, having the tweets come in out of order is a good test for your streaming code.

**Static Windows**

The next thing you should do is to put together your query in the static (non-streaming) environment. It will be much easier to test statically on the full CSV file you read in initially.

The first thing you should do is to reduce the columns you are working with. You only need the ID, Date and Hashtag columns. Also, some of the tweets do not have hashtags and those will be null in the DataFrame. So you can just filter those rows out at this point as well using .isNotNull(). Note that it might be easier if you referenced column names with f.col('name') rather than fifaDF['name'] so your code is easier to reuse later in the steaming portion.

When you look at the resulting DataFrame, you will see that certain rows have multiple hashTags like "WorldCup,FRA,ENG". This was how the data for the tweet was created in the first place from the Twitter API. Since we are going to be adding up all occurrences of each hash tag separately, we will need to have that comma separated string broken apart so each hashtag is on its own row. To do this you will use several DataFrame features. The first is .withColumn. That allows you to work with one column of the DataFrame. You need to specify the column you want and what is going to happen to it. What you need to do is to explode the column so you get a row for each element. But explode takes a list of elements, not a comma separated string of elements. So you will also have to use split which will create the list from the string. Use the following example as a guide:

```
from pyspark.sql import Row
df = spark.createDataFrame([Row(a=1, b=3,c="7,8,9", d='foo')])
df.show()
df = df.withColumn('c', f.explode(f.split('c', ',')))
df.show()
```

After you have the data transformed into a more useable form, you can then use DataFrame operations window a counting aggregation over the hashtags. Your window should be a sliding window. Each window should be 60 minutes long, with a sliding rate of every 30 minutes. This will produce a large number of entries for each window since there are many single hashtags. Since we only care about tending tags, you should only keep those that have aggregate counts greater than 100 within their window. Note that in SQL you would do this with a HAVING clause to execute after your GROUP BY. However with DataFrames, you can simply do another .filter after the groupby agg. That is, in SQL we needed two names: WHERE and HAVING since we didn't really specify the order. But with DataFrame function, we get explicit control of the order and can get by with a single function: filter.

The ordering doesn't matter when you produce the resulting windowed DataFrame. But it might be nice to see it ordered when showing it for human consumption. Thus when you show it, order it by window and count. The following is a sample of what your data should look like:

```
+------------------------------------------+-------------+-----+
|window |Hashtags |count|
+------------------------------------------+-------------+-----+
|[2018-06-29 23:00:00, 2018-06-30 00:00:00]|WorldCup     |152  |
|[2018-06-29 23:30:00, 2018-06-30 00:30:00]|WorldCup     |1270 |
|[2018-06-29 23:30:00, 2018-06-30 00:30:00]|FIFAStadiumDJ|615  |
|[2018-06-30 00:00:00, 2018-06-30 01:00:00]|WorldCup     |2278 |
|[2018-06-30 00:00:00, 2018-06-30 01:00:00]|FIFAStadiumDJ|1073 |
|[2018-06-30 00:00:00, 2018-06-30 01:00:00]|EXO          |157  |
|[2018-06-30 00:00:00, 2018-06-30 01:00:00]|worldcup     |144  |
|[2018-06-30 00:30:00, 2018-06-30 01:30:00]|WorldCup     |2158 |
|[2018-06-30 00:30:00, 2018-06-30 01:30:00]|FIFAStadiumDJ|990  |
|[2018-06-30 00:30:00, 2018-06-30 01:30:00]|EXO          |156  |
|[2018-06-30 00:30:00, 2018-06-30 01:30:00]|worldcup     |150  |
```

**Structured Streaming**

Up to this point, you haven't done any streaming at all. You have divided up the file so you can simulate streaming on the data. And you have tested the transformation query you will use in a static environment where it is easier to test. Now you are to code the streaming version. Obviously, you will use your directory of small csv files as your sink. The transformation query is similar to your static query. And then the action is replaced with a sink. Please use a memory sink.

The only additional issue is that we would like to have a watermark to prevent data that comes in way too late from being added to the stream. Use 24 hours as your watermark value.

Once you start your streaming running, you can (in another command) issue a query to show the results as they are being produced by the streaming.

**Submission**

You are upload 2 documents as your submission. The first is your Python source code (PY). Make sure to have your name in a comment at the top of your source code. Second is a document that has a cut-paste of your results from running your code. You should include the

static windowed results (say the first 20 or so) so I can tell if your overall query is right. And then some capture of a result from the streaming – which will obviously be different for everyone depending on exactly when you issued the query. But I would still like to see streaming results as one of your two outputs.