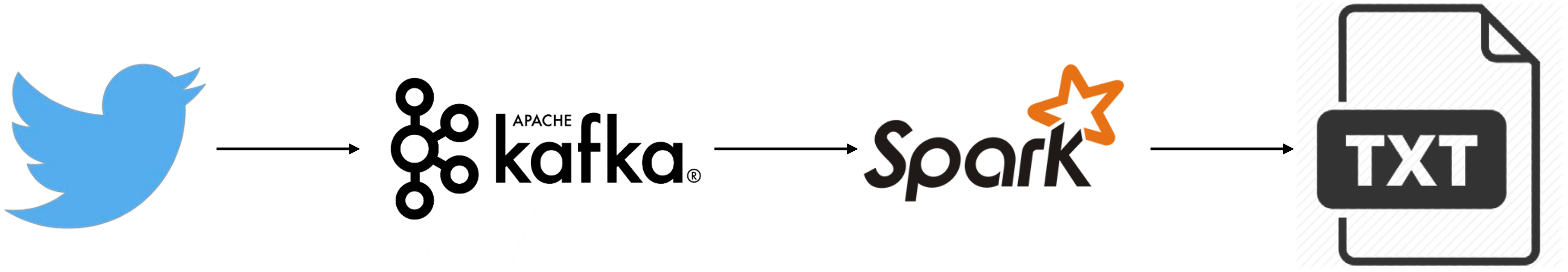
# Apache Kafka + Apache Spark

## Apache Kafka + Apache Spark

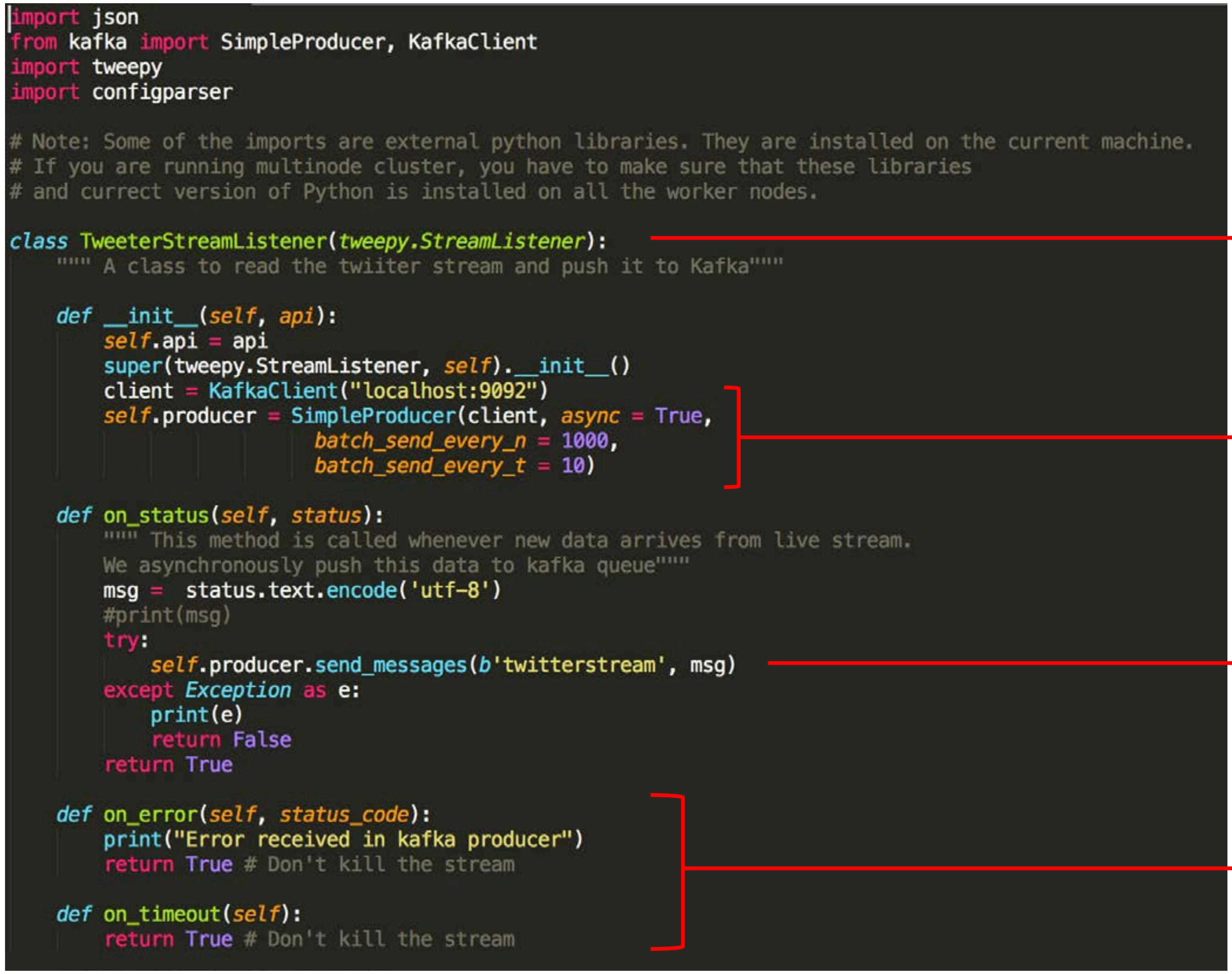
- Apache Kafka and Apache Spark are part of the Data Science software stack.

- Sentiment Analysis uses Apache Kafka to store streaming data from Twitter and Apache Spark to process data. How does this work?
  - Collect tweets from Twitter and feed it into Apache Kafka
  - Kafka supplies this data in batches for processing in Apache Spark
  - Output the number of positive and negative words being mentioned



# Sentiment Analysis using Kafka + Spark

- Step 1: Collect tweets from Twitter and feed it into Apache Kafka



```python
import json
from kafka import SimpleProducer, KafkaClient
import tweepy
import configparser

# Note: Some of the imports are external python libraries. They are installed on the current machine.
# If you are running multinode cluster, you have to make sure that these libraries
# and currect version of Python is installed on all the worker nodes.

class TweeterStreamListener(tweepy.StreamListener):
    """ A class to read the twiiter stream and push it to Kafka"""

    def __init__(self, api):
        self.api = api
        super(tweepy.StreamListener, self).__init__()
        client = KafkaClient("localhost:9092")
        self.producer = SimpleProducer(client, async = True,
                                  batch_send_every_n = 1000,
                                  batch_send_every_t = 10)

    def on_status(self, status):
        """ This method is called whenever new data arrives from live stream.
        We asynchronously push this data to kafka queue"""
        msg =  status.text.encode('utf-8')
        #print(msg)
        try:
            self.producer.send_messages(b'twitterstream', msg)
        except Exception as e:
            print(e)
            return False
        return True

    def on_error(self, status_code):
        print("Error received in kafka producer")
        return True # Don't kill the stream

    def on_timeout(self):
        return True # Don't kill the stream
```

Listens to the twitter stream continuously

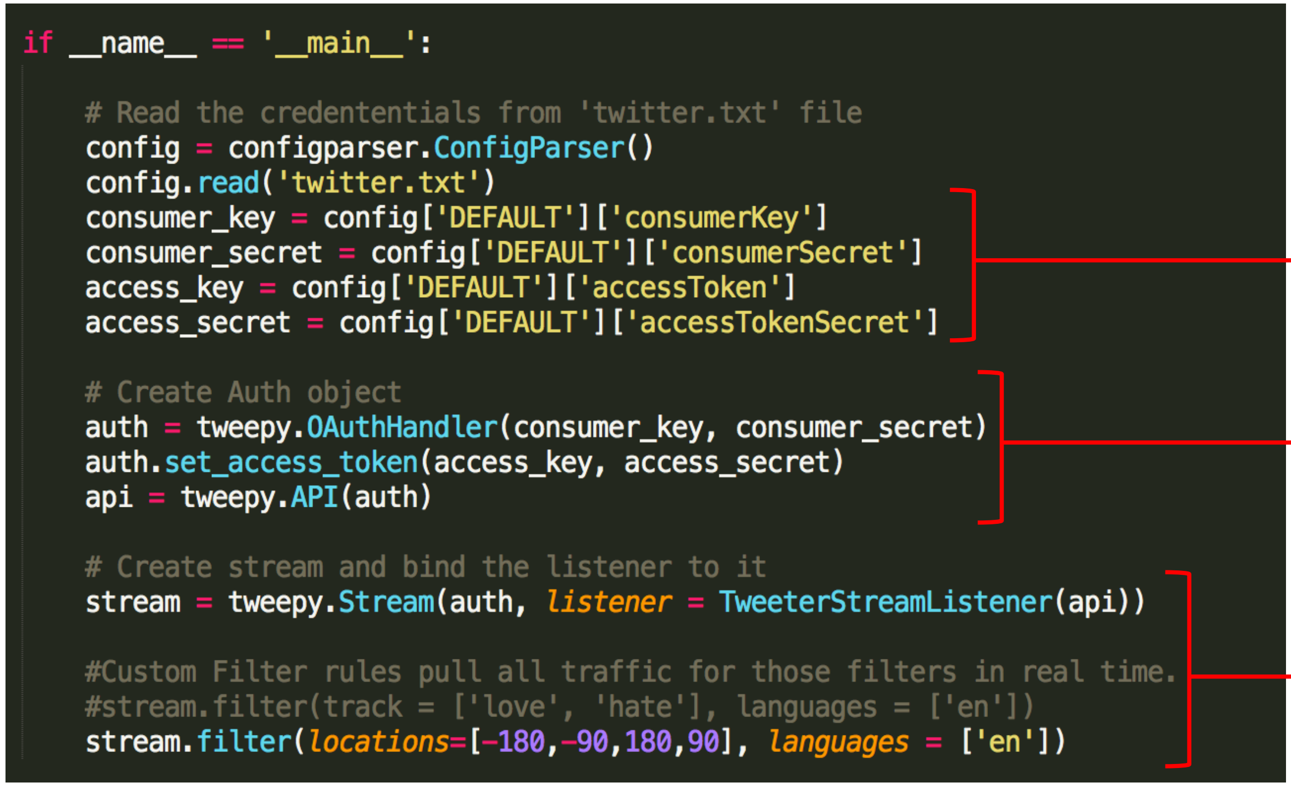
Create a connection to Kafka that sends information in batches of 1000 tweets or every 10 seconds

Create a topic in Kafka under which tweets are saved

Either on encountering an error or timeout, do not stop the stream

# Sentiment Analysis using Kafka + Spark

- Step 1: Collect tweets from Twitter and feed it into Apache Kafka



```
if __name__ == '__main__':

    # Read the credententials from 'twitter.txt' file
    config = configparser.ConfigParser()
    config.read('twitter.txt')
    consumer_key = config['DEFAULT']['consumerKey']
    consumer_secret = config['DEFAULT']['consumerSecret']
    access_key = config['DEFAULT']['accessToken']
    access_secret = config['DEFAULT']['accessTokenSecret']

    # Create Auth object
    auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_key, access_secret)
    api = tweepy.API(auth)

    # Create stream and bind the listener to it
    stream = tweepy.Stream(auth, listener = TweeterStreamListener(api))

    #Custom Filter rules pull all traffic for those filters in real time.
    #stream.filter(track = ['love', 'hate'], languages = ['en'])
    stream.filter(locations=[-180,-90,180,90], languages = ['en'])
```

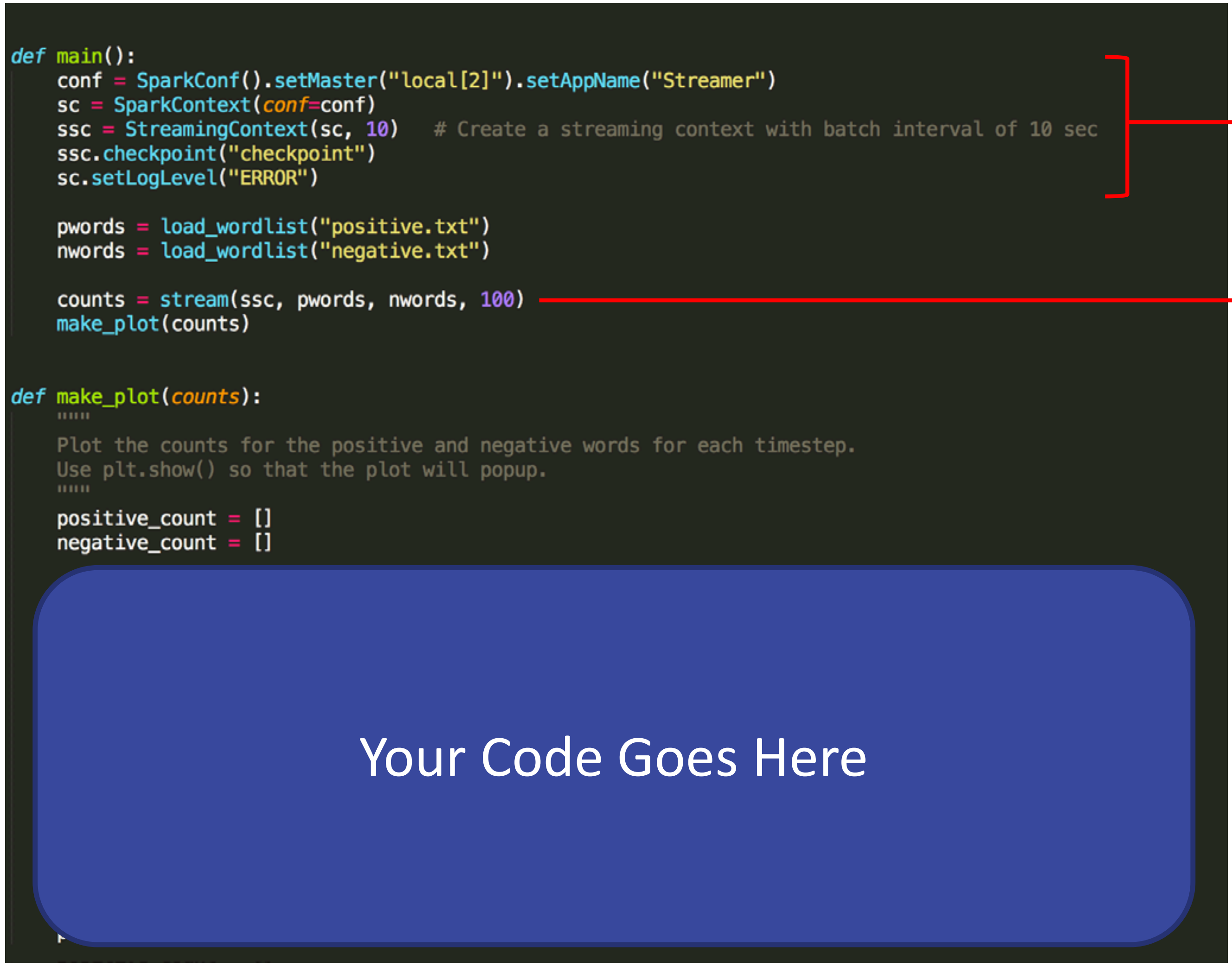
Create a Twitter developer account to get API access keys.

Authentication to connect to Twitter via Tweepy

Set the listener to the stream and filter information needed

# Sentiment Analysis using Kafka + Spark

- Step 2: Kafka supplies this data in batches for processing in Apache Spark



```
def main():
    conf = SparkConf().setMaster("local[2]").setAppName("Streamer")
    sc = SparkContext(conf=conf)
    ssc = StreamingContext(sc, 10)   # Create a streaming context with batch interval of 10 sec
    ssc.checkpoint("checkpoint")
    sc.setLogLevel("ERROR")

    pwords = load_wordlist("positive.txt")
    nwords = load_wordlist("negative.txt")

    counts = stream(ssc, pwords, nwords, 100)
    make_plot(counts)


def make_plot(counts):
    """
    Plot the counts for the positive and negative words for each timestep.
    Use plt.show() so that the plot will popup.
    """
    positive_count = []
    negative_count = []
```

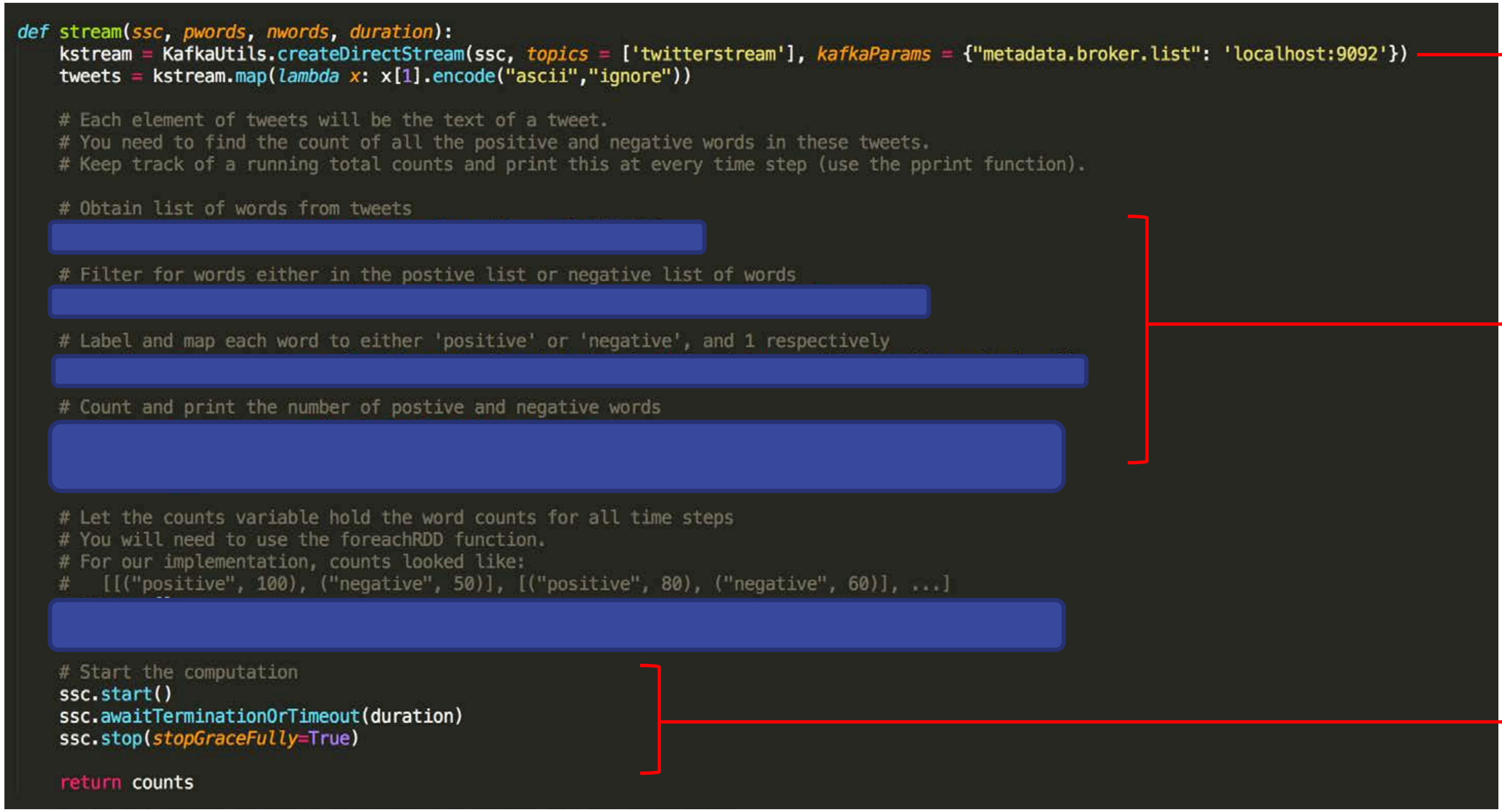
Your Code Goes Here

Create Spark and Streaming context to stream in batches of 10 seconds

Call stream function to compute number of positive and negative words for 100 seconds

# Sentiment Analysis using Kafka + Spark

- Step 2: Kafka supplies this data in batches for processing in Apache Spark



```
def stream(ssc, pwords, nwords, duration):
    kstream = KafkaUtils.createDirectStream(ssc, topics = ['twitterstream'], kafkaParams = {"metadata.broker.list": 'localhost:9092'})
    tweets = kstream.map(lambda x: x[1].encode("ascii","ignore"))

    # Each element of tweets will be the text of a tweet.
    # You need to find the count of all the positive and negative words in these tweets.
    # Keep track of a running total counts and print this at every time step (use the pprint function).

    # Obtain list of words from tweets
    [redacted]

    # Filter for words either in the postive list or negative list of words
    [redacted]

    # Label and map each word to either 'positive' or 'negative', and 1 respectively
    [redacted]

    # Count and print the number of postive and negative words
    [redacted]

    # Let the counts variable hold the word counts for all time steps
    # You will need to use the foreachRDD function.
    # For our implementation, counts looked like:
    #   [[("positive", 100), ("negative", 50)], [("positive", 80), ("negative", 60)], ...]
    [redacted]

    # Start the computation
    ssc.start()
    ssc.awaitTerminationOrTimeout(duration)
    ssc.stop(stopGraceFully=True)

    return counts
```

Connect to Kafka topic

Filter out words from tweet

Check if positive or negative

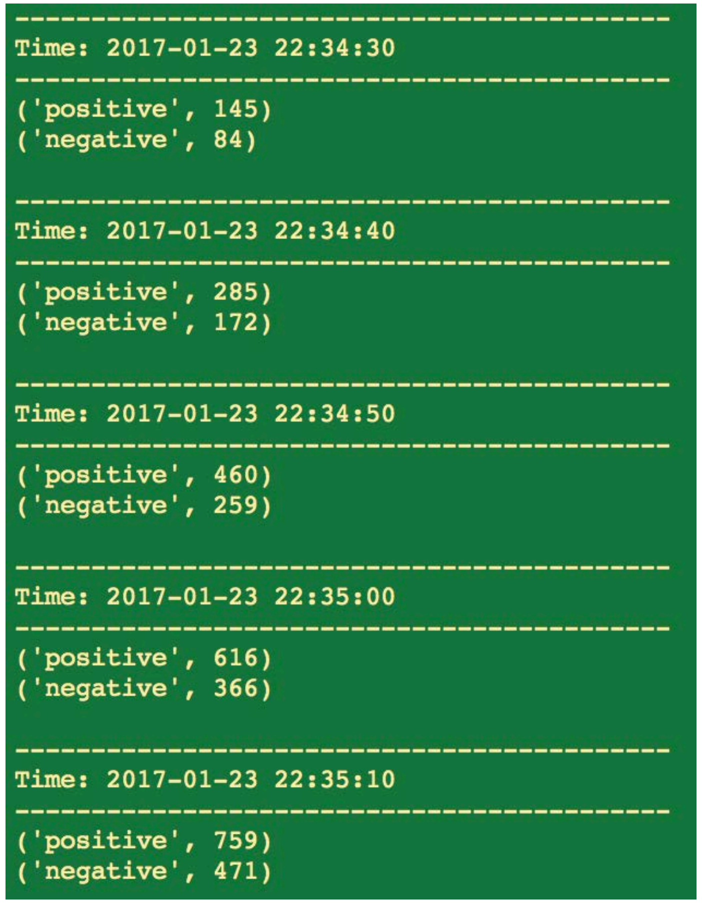
Map to 'positive' with count 1 or 'negative' with count 1

Roll up and get count

Continue for 100 seconds

# Sentiment Analysis using Kafka + Spark

- Step 3: Output the number of positive and negative words being mentioned



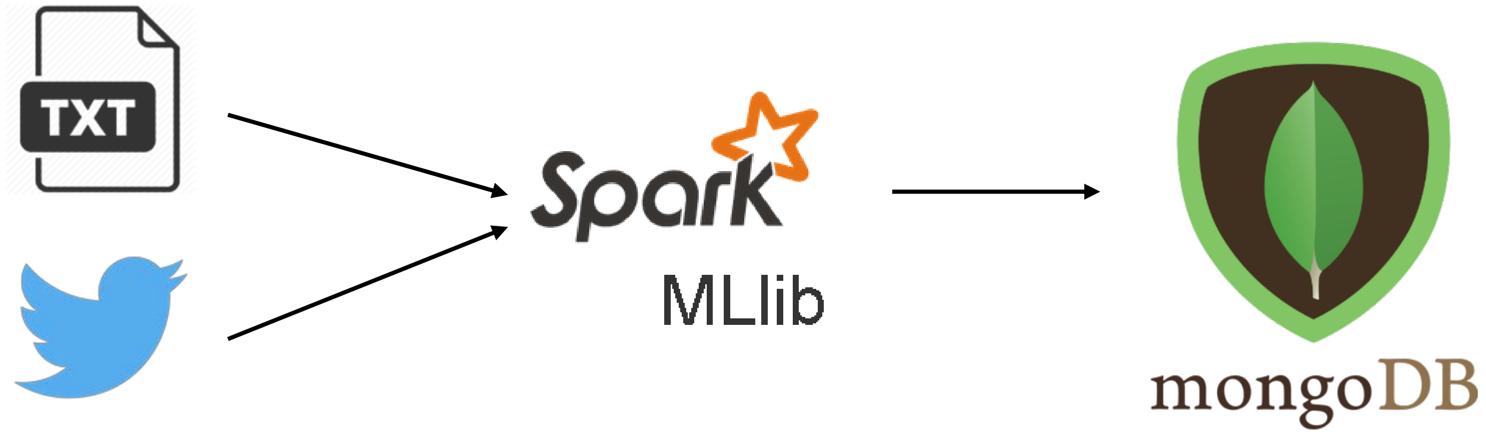
```
-------------------------------------------
Time: 2017-01-23 22:34:30
-------------------------------------------
('positive', 145)
('negative', 84)

-------------------------------------------
Time: 2017-01-23 22:34:40
-------------------------------------------
('positive', 285)
('negative', 172)

-------------------------------------------
Time: 2017-01-23 22:34:50
-------------------------------------------
('positive', 460)
('negative', 259)

-------------------------------------------
Time: 2017-01-23 22:35:00
-------------------------------------------
('positive', 616)
('negative', 366)

-------------------------------------------
Time: 2017-01-23 22:35:10
-------------------------------------------
('positive', 759)
('negative', 471)
```

## Apache Spark + MongoDB

# Apache Spark + MongoDB

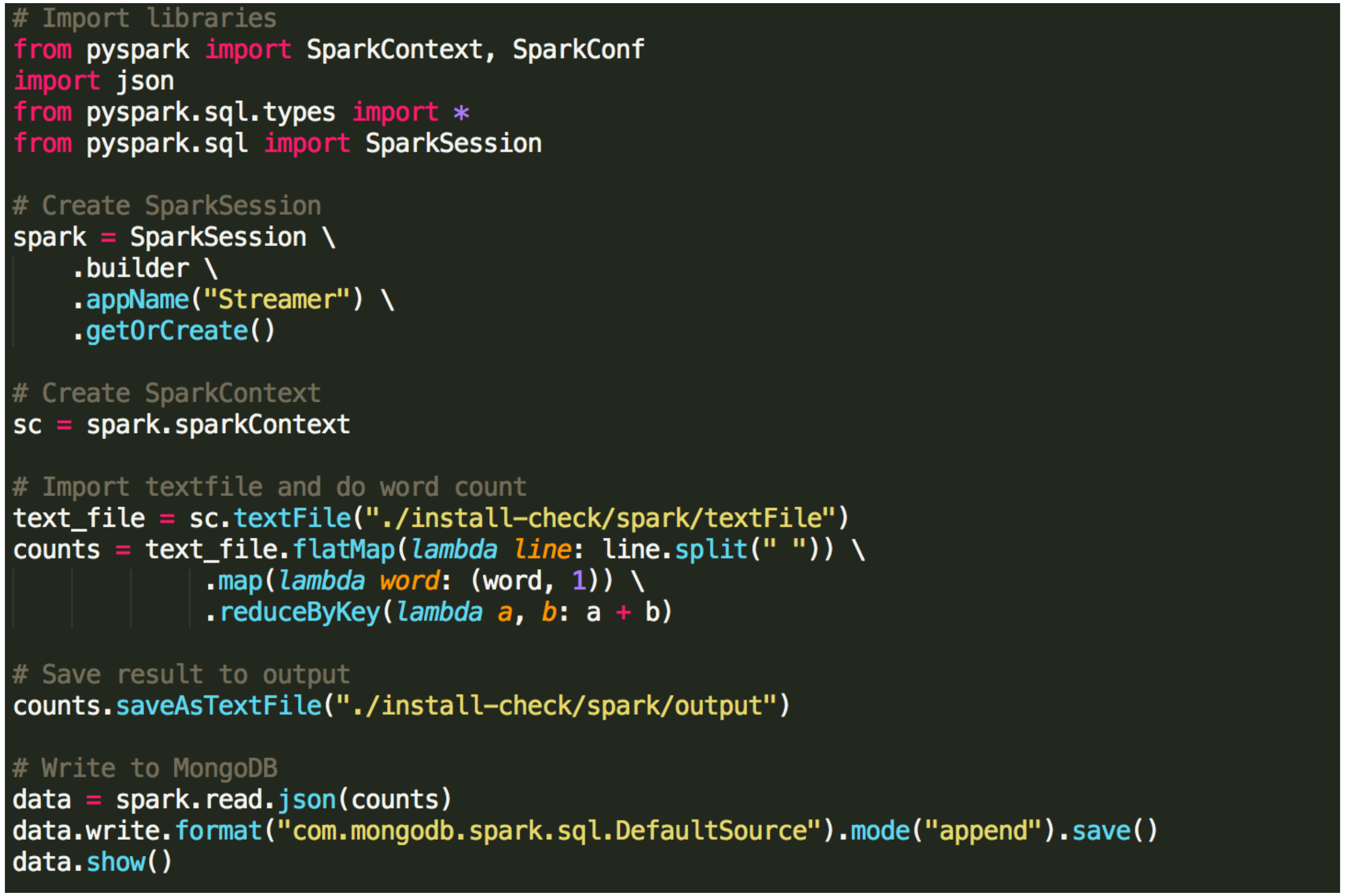
- Apache Spark and MongoDB are part of the Data Science software stack.

- Word count uses Apache Spark to process data and the result is stored in MongoDB. How does this work?
  - Import data in text file
  - Apache Spark computes the word count
  - Result is stored in MongoDB collection

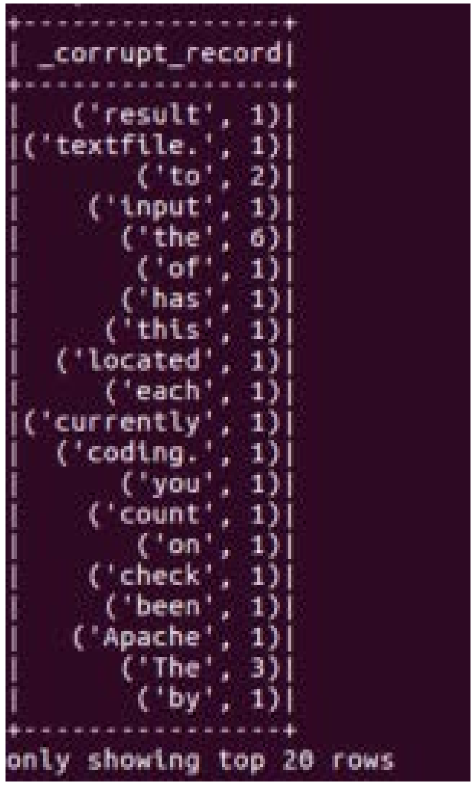


# Word Count using Spark + MongoDB

- Step 1: Import data in text file. Apache Spark computes the word count.



```python
# Import libraries
from pyspark import SparkContext, SparkConf
import json
from pyspark.sql.types import *
from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession \
    .builder \
    .appName("Streamer") \
    .getOrCreate()

# Create SparkContext
sc = spark.sparkContext

# Import textfile and do word count
text_file = sc.textFile("./install-check/spark/textFile")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)

# Save result to output
counts.saveAsTextFile("./install-check/spark/output")

# Write to MongoDB
data = spark.read.json(counts)
data.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").save()
data.show()
```

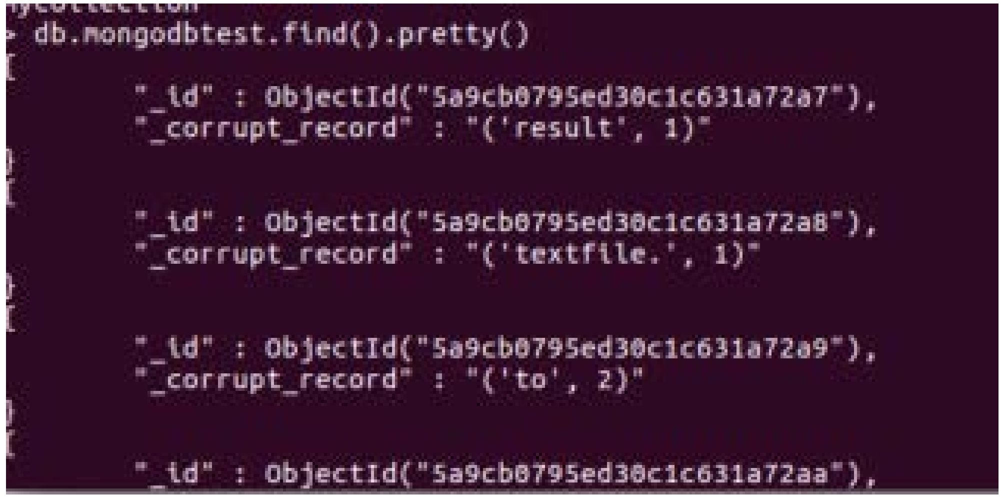


```
+----------------+
| _corrupt_record|
+----------------+
|   ('result', 1)|
|('textfile.', 1)|
|       ('to', 2)|
|    ('input', 1)|
|      ('the', 6)|
|       ('of', 1)|
|      ('has', 1)|
|     ('this', 1)|
|  ('located', 1)|
|     ('each', 1)|
|('currently', 1)|
|  ('coding.', 1)|
|      ('you', 1)|
|    ('count', 1)|
|       ('on', 1)|
|    ('check', 1)|
|     ('been', 1)|
|   ('Apache', 1)|
|      ('The', 3)|
|       ('by', 1)|
+----------------+
only showing top 20 rows
```

## Word Count using Spark + MongoDB

- Step 2: Result is stored in MongoDB collection.

```
1. sudo service mongod start

2. $SPARK_HOME/bin/spark-submit --conf "spark.mongodb.input.uri=mongodb://127.0.0.1/local.coll?
readPreference=primaryPreferred" --conf "spark.mongodb.output.uri=mongodb://127.0.0.1/local.coll" --
packages org.mongodb.spark:mongo-spark-connector_2.11:2.0.0 ./install-check/spark/testing.py

3. mongo

This will open the mongo shell. Type the following commands in the shell.

4. use local

5. db.coll.find().pretty()

You will see the results in MongoDB.
```



```
> db.mongodbtest.find().pretty()
{
        "_id" : ObjectId("5a9cb0795ed30c1c631a72a7"),
        "_corrupt_record" : "('result', 1)"
}
{
        "_id" : ObjectId("5a9cb0795ed30c1c631a72a8"),
        "_corrupt_record" : "('textfile.', 1)"
}
{
        "_id" : ObjectId("5a9cb0795ed30c1c631a72a9"),
        "_corrupt_record" : "('to', 2)"
}
{
        "_id" : ObjectId("5a9cb0795ed30c1c631a72aa"),
```