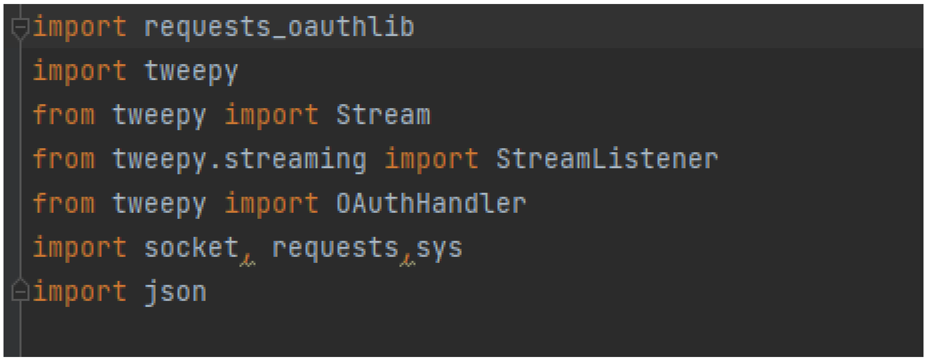
PROJECT REPORT CS644

PEPELASI GKERI

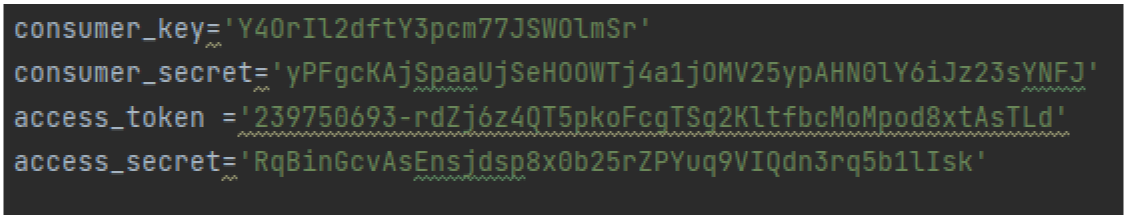
-STREAMING

The *tweepy* library is necessary for connecting to the Twitter API and building the data streaming pipeline. We import its classes; *Stream* and *StreamListener* for building the stream, and *OAuthHandler* for authenticating to Twitter. We import the socket module to create a communication channel between our local machine and the Twitter API. We need the *JSON* module to handle the data of *JSON* objects.



```
import requests_oauthlib
import tweepy
from tweepy import Stream
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
import socket, requests,sys
import json
```

Then I had to use credentials from my Twitter developer account.



```
consumer_key='Y4OrIl2dftY3pcm77JSWOlmSr'
consumer_secret='yPFgcKAjSpaaUjSeHOOWTj4a1jOMV25ypAHN0lY6iJz23sYNFJ'
access_token ='239750693-rdZj6z4QT5pkoFcgTSq2KltfbcMoMpod8xtAsTLd'
access_secret='RqBinGcvAsEnsjdsp8x0b25rZPYuq9VIQdn3rq5b1lIsk'
```

The class *TweetListener* represents a *StreamListener* instance, which connects to the Twitter API and returns one tweet at a time. As soon as we activate the Stream, it continuously creates instances.

The class consists of 3 methods; the *on_data*, the *on_error*, and the *__init__*.

The *on_data* method reads the incoming tweet JSON file, which contains one tweet, and defines which part to keep. Some example parts could be the actual tweet message, comments, or hashtags. In our case, we want to extract the actual text of the tweet. If we only request the ['text'] field from each tweet, we will only receive messages shorter than 140 characters. To be sure we receive the full message, we need to first check if the tweet is longer than 140 characters. If it is, we extract the [*'extended_tweet']['full_text'*] field and if it is not; we extract the ['text'] field. At the end of each tweet, we add the 't_end' string, so that at a later stage, we can identify the end of each tweet easier.

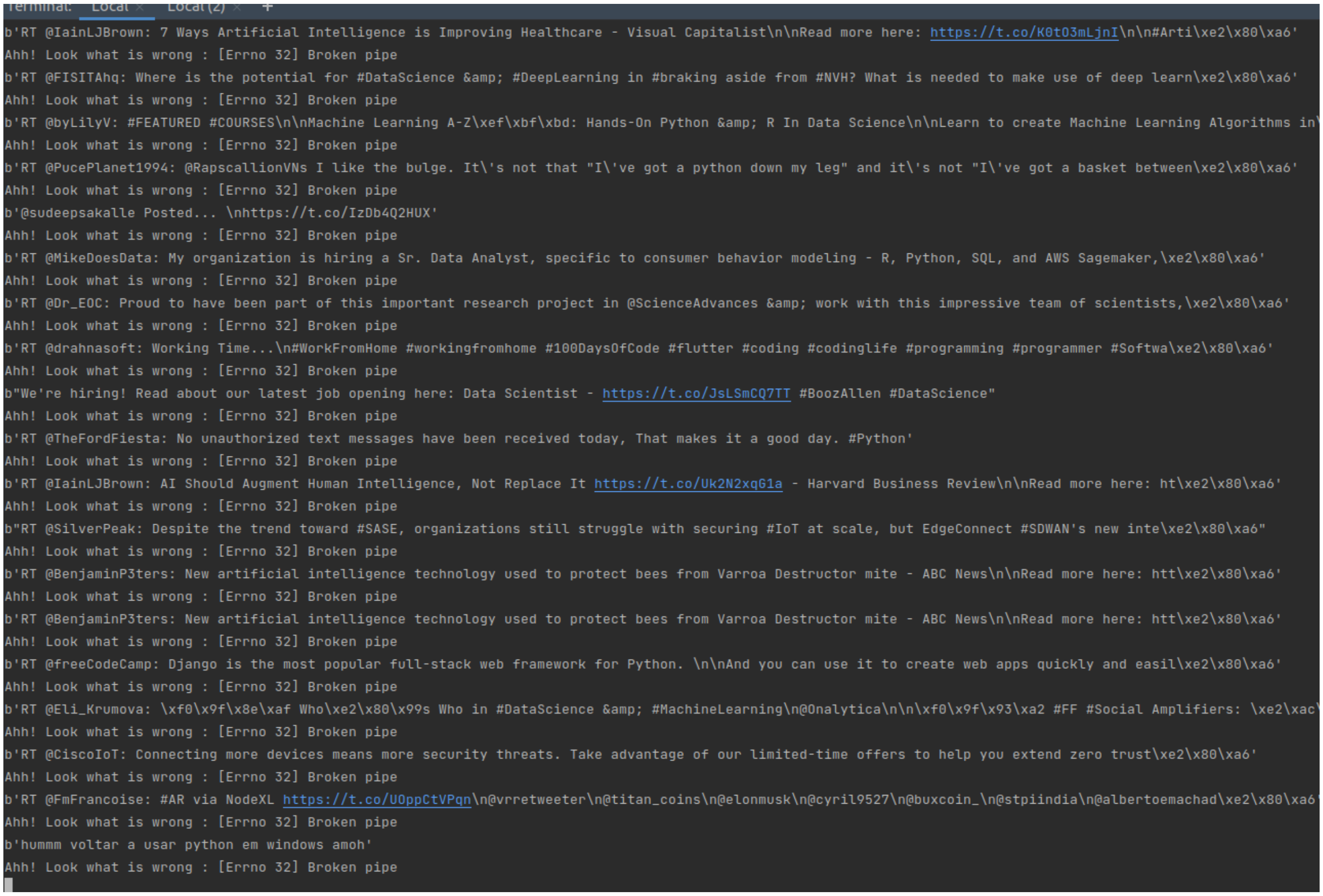
The *on_error* method makes sure that the stream works and the __init__ method initializes the socket of the Twitter API.

To get data from the Twitter API, we first use the pre-defined credentials to authenticate the connection to the API. After the authentication, we stream the tweet data objects that contain a selected keyword and language. The returned objects are tweets of the TweetListener class.

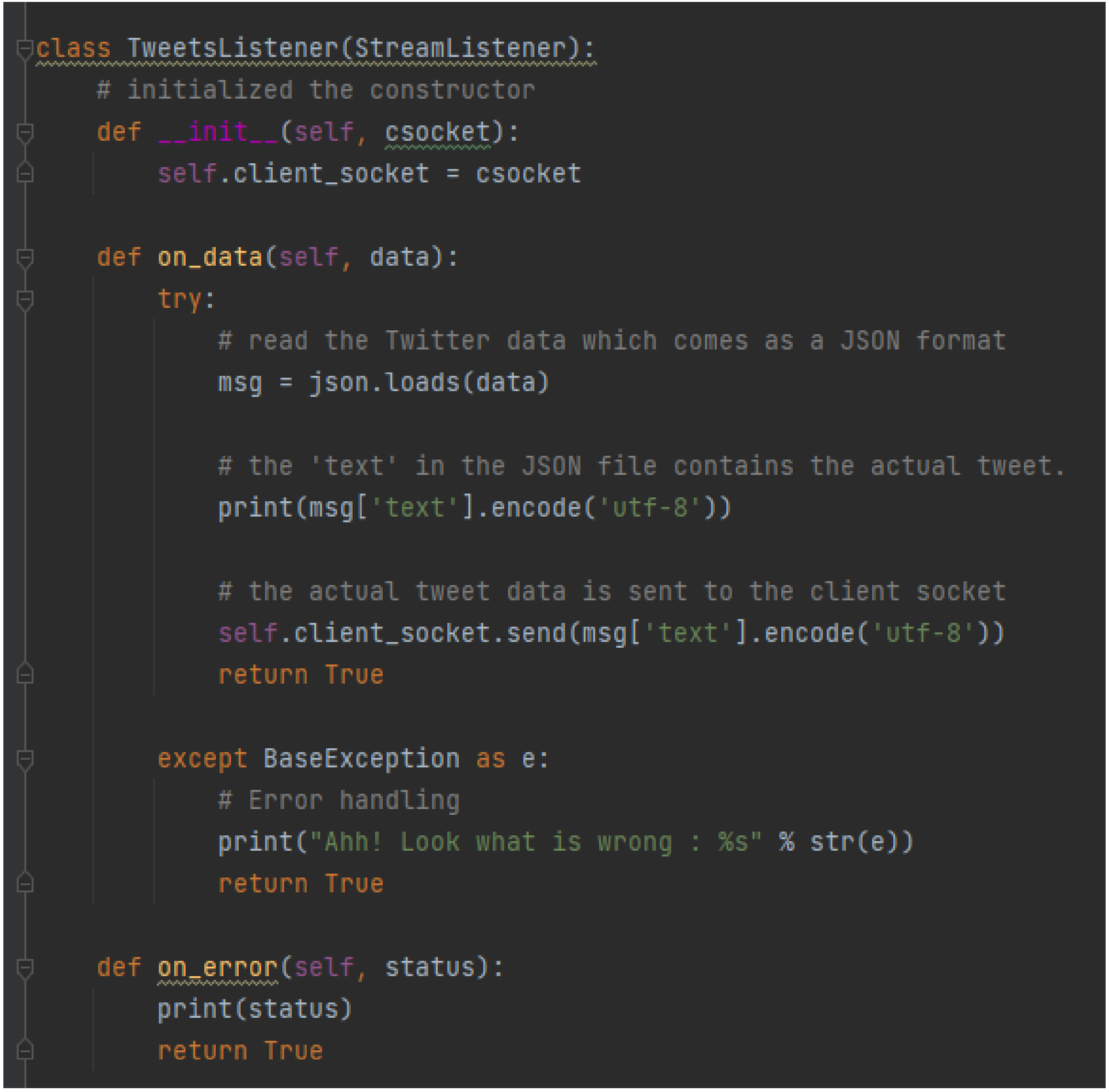
Before streaming the data from the Twitter API, we need to create a listening TCP socket in the local machine (server) in a pre-defined local IP address and a port. The socket consists of a server-side, which is our local machine, and a client-side, which will be the Twitter API. The open socket from the server-side listens for the client. When the client-side is up and running, the socket will receive the data from the Twitter API. To query the tweets that are related to a specific topic, we also select one or multiple keywords for streaming.

PROJECT REPORT CS644

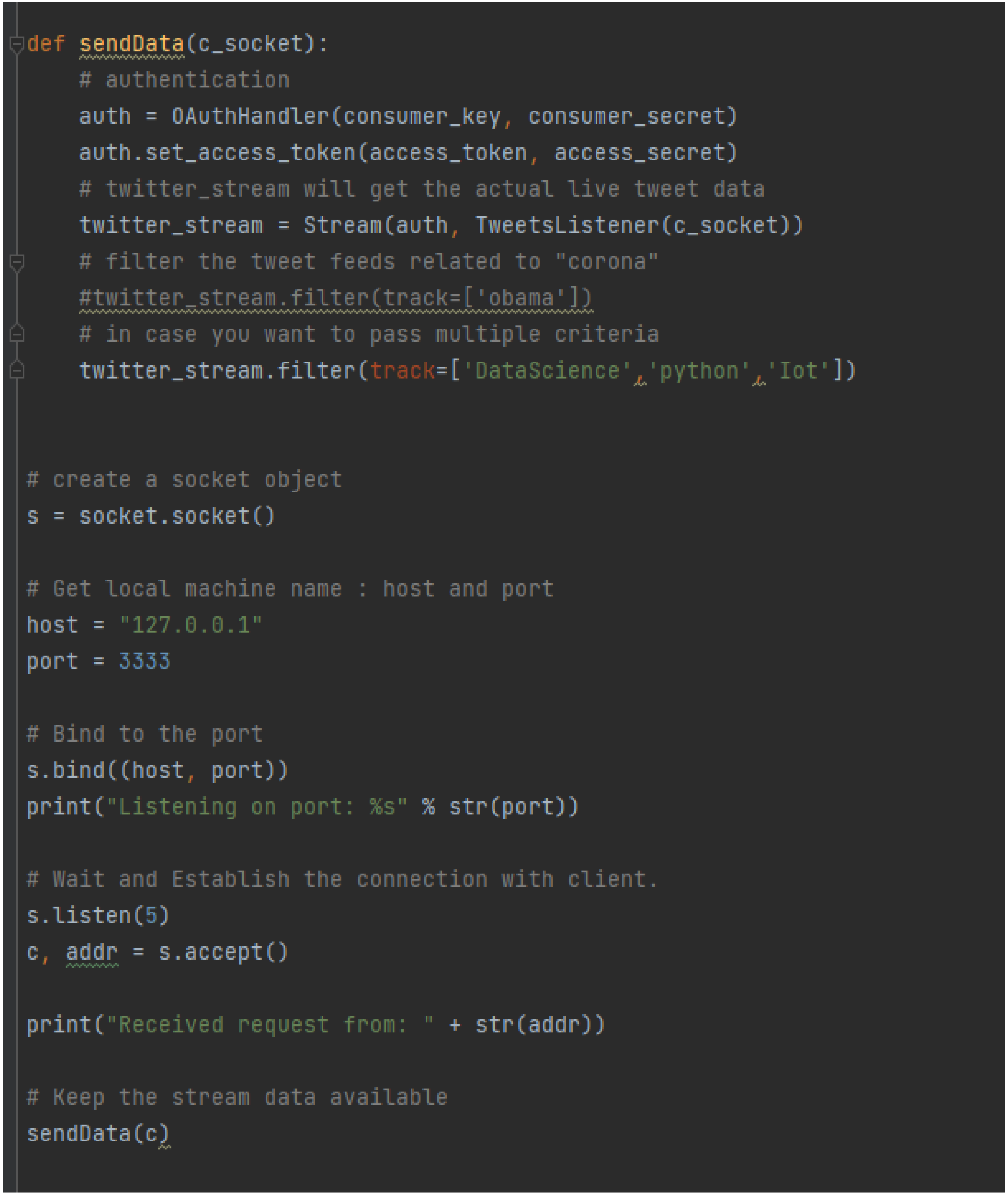
I streamed data from twitter with key words “data science”, “python”, “iot”.



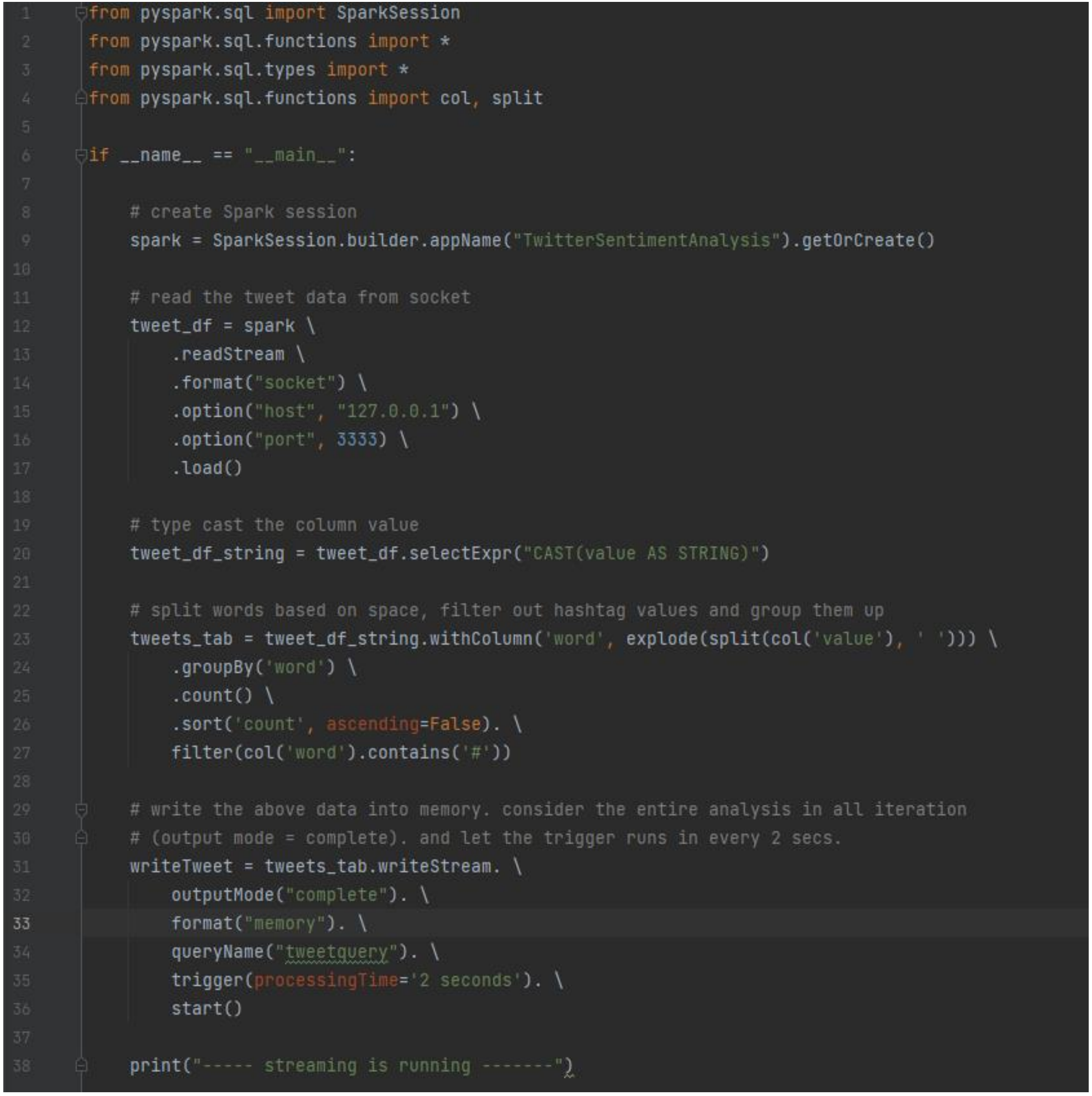
```
b'RT @IainLJBrown: 7 Ways Artificial Intelligence is Improving Healthcare - Visual Capitalist\n\nRead more here: https://t.co/K0tO3mLjnI\n\n#Arti\xe2\x80\xa6'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @FISITAhq: Where is the potential for #DataScience &amp; #DeepLearning in #braking aside from #NVH? What is needed to make use of deep learn\xe2\x80\xa6'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @byLilyV: #FEATURED #COURSES\n\nMachine Learning A-Z\xef\xbf\xbd: Hands-On Python &amp; R In Data Science\n\nLearn to create Machine Learning Algorithms in
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @PucePlanet1994: @RapscallionVNs I like the bulge. It\'s not that "I\'ve got a python down my leg" and it\'s not "I\'ve got a basket between\xe2\x80\xa6'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'@sudeepsakalle Posted... \nhttps://t.co/IzDb4Q2HUX'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @MikeDoesData: My organization is hiring a Sr. Data Analyst, specific to consumer behavior modeling - R, Python, SQL, and AWS Sagemaker,\xe2\x80\xa6'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @Dr_EOC: Proud to have been part of this important research project in @ScienceAdvances &amp; work with this impressive team of scientists,\xe2\x80\xa6'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @drahnasoft: Working Time...\n#WorkFromHome #workingfromhome #100DaysOfCode #flutter #coding #codinglife #programming #programmer #Softwa\xe2\x80\xa6'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b"We're hiring! Read about our latest job opening here: Data Scientist - https://t.co/JsLSmCQ7TT #BoozAllen #DataScience"
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @TheFordFiesta: No unauthorized text messages have been received today, That makes it a good day. #Python'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @IainLJBrown: AI Should Augment Human Intelligence, Not Replace It https://t.co/Uk2N2xqG1a - Harvard Business Review\n\nRead more here: ht\xe2\x80\xa6'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b"RT @SilverPeak: Despite the trend toward #SASE, organizations still struggle with securing #IoT at scale, but EdgeConnect #SDWAN's new inte\xe2\x80\xa6"
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @BenjaminP3ters: New artificial intelligence technology used to protect bees from Varroa Destructor mite - ABC News\n\nRead more here: htt\xe2\x80\xa6'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @BenjaminP3ters: New artificial intelligence technology used to protect bees from Varroa Destructor mite - ABC News\n\nRead more here: htt\xe2\x80\xa6'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @freeCodeCamp: Django is the most popular full-stack web framework for Python. \n\nAnd you can use it to create web apps quickly and easil\xe2\x80\xa6'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @Eli_Krumova: \xf0\x9f\x8e\xaf Who\xe2\x80\x99s Who in #DataScience &amp; #MachineLearning\n@Onalytica\n\n\xf0\x9f\x93\xa2 #FF #Social Amplifiers: \xe2\xac
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @CiscoIoT: Connecting more devices means more security threats. Take advantage of our limited-time offers to help you extend zero trust\xe2\x80\xa6'
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'RT @FmFrancoise: #AR via NodeXL https://t.co/UOppCtVPqn\n@vrretweeter\n@titan_coins\n@elonmusk\n@cyril9527\n@buxcoin_\n@stpiindia\n@albertoemachad\xe2\x80\xa6
Ahh! Look what is wrong : [Errno 32] Broken pipe
b'hummm voltar a usar python em windows amoh'
Ahh! Look what is wrong : [Errno 32] Broken pipe
```



```python
class TweetsListener(StreamListener):
    # initialized the constructor
    def __init__(self, csocket):
        self.client_socket = csocket

    def on_data(self, data):
        try:
            # read the Twitter data which comes as a JSON format
            msg = json.loads(data)

            # the 'text' in the JSON file contains the actual tweet.
            print(msg['text'].encode('utf-8'))

            # the actual tweet data is sent to the client socket
            self.client_socket.send(msg['text'].encode('utf-8'))
            return True

        except BaseException as e:
            # Error handling
            print("Ahh! Look what is wrong : %s" % str(e))
            return True

    def on_error(self, status):
        print(status)
        return True
```



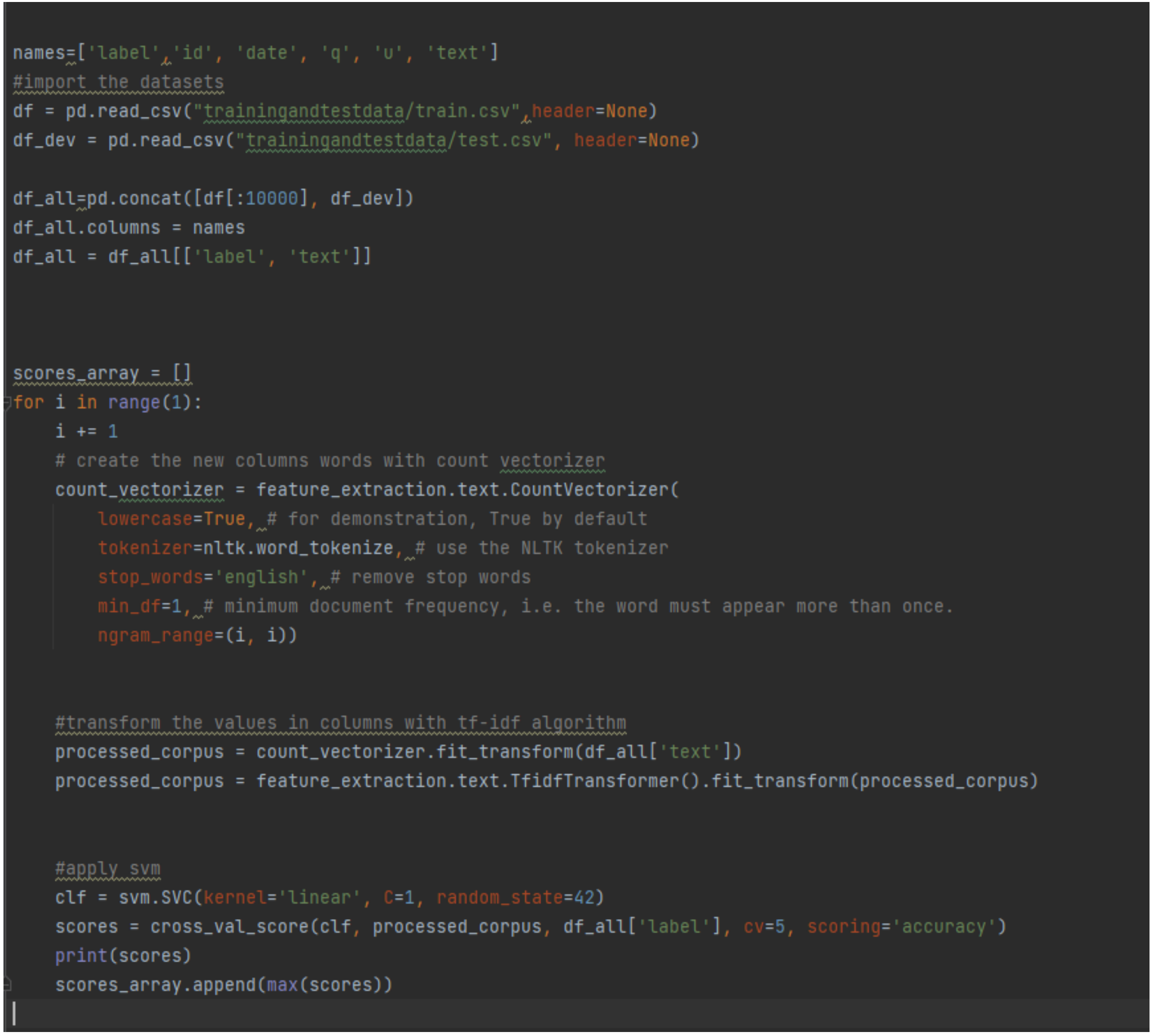
```
def sendData(c_socket):
    # authentication
    auth = OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_secret)
    # twitter_stream will get the actual live tweet data
    twitter_stream = Stream(auth, TweetsListener(c_socket))
    # filter the tweet feeds related to "corona"
    #twitter_stream.filter(track=['obama'])
    # in case you want to pass multiple criteria
    twitter_stream.filter(track=['DataScience','python','Iot'])


# create a socket object
s = socket.socket()

# Get local machine name : host and port
host = "127.0.0.1"
port = 3333

# Bind to the port
s.bind((host, port))
print("Listening on port: %s" % str(port))

# Wait and Establish the connection with client.
s.listen(5)
c, addr = s.accept()

print("Received request from: " + str(addr))

# Keep the stream data available
sendData(c)
```



```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql.functions import col, split

if __name__ == "__main__":

    # create Spark session
    spark = SparkSession.builder.appName("TwitterSentimentAnalysis").getOrCreate()

    # read the tweet data from socket
    tweet_df = spark \
        .readStream \
        .format("socket") \
        .option("host", "127.0.0.1") \
        .option("port", 3333) \
        .load()

    # type cast the column value
    tweet_df_string = tweet_df.selectExpr("CAST(value AS STRING)")

    # split words based on space, filter out hashtag values and group them up
    tweets_tab = tweet_df_string.withColumn('word', explode(split(col('value'), ' '))) \
        .groupBy('word') \
        .count() \
        .sort('count', ascending=False). \
        filter(col('word').contains('#'))

    # write the above data into memory. consider the entire analysis in all iteration
    # (output mode = complete). and let the trigger runs in every 2 secs.
    writeTweet = tweets_tab.writeStream. \
        outputMode("complete"). \
        format("memory"). \
        queryName("tweetquery"). \
        trigger(processingTime='2 seconds'). \
        start()

    print("----- streaming is running -------")
```

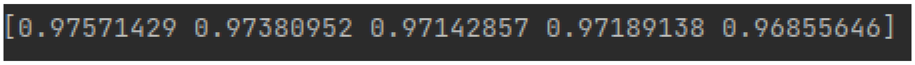
-**Machine Learning**

Then I sued scikit learn library with svm algorithm with kernel being linear , bias=1 and used vectorization *tfidfTransformer* which basically what it does is categorizing the words with how rare is each one and also used ngram=1 .



```
names=['label','id', 'date', 'q', 'u', 'text']
#import the datasets
df = pd.read_csv("trainingandtestdata/train.csv",header=None)
df_dev = pd.read_csv("trainingandtestdata/test.csv", header=None)

df_all=pd.concat([df[:10000], df_dev])
df_all.columns = names
df_all = df_all[['label', 'text']]



scores_array = []
for i in range(1):
    i += 1
    # create the new columns words with count vectorizer
    count_vectorizer = feature_extraction.text.CountVectorizer(
        lowercase=True, # for demonstration, True by default
        tokenizer=nltk.word_tokenize, # use the NLTK tokenizer
        stop_words='english', # remove stop words
        min_df=1, # minimum document frequency, i.e. the word must appear more than once.
        ngram_range=(i, i))


    #transform the values in columns with tf-idf algorithm
    processed_corpus = count_vectorizer.fit_transform(df_all['text'])
    processed_corpus = feature_extraction.text.TfidfTransformer().fit_transform(processed_corpus)


    #apply svm
    clf = svm.SVC(kernel='linear', C=1, random_state=42)
    scores = cross_val_score(clf, processed_corpus, df_all['label'], cv=5, scoring='accuracy')
    print(scores)
    scores_array.append(max(scores))
```

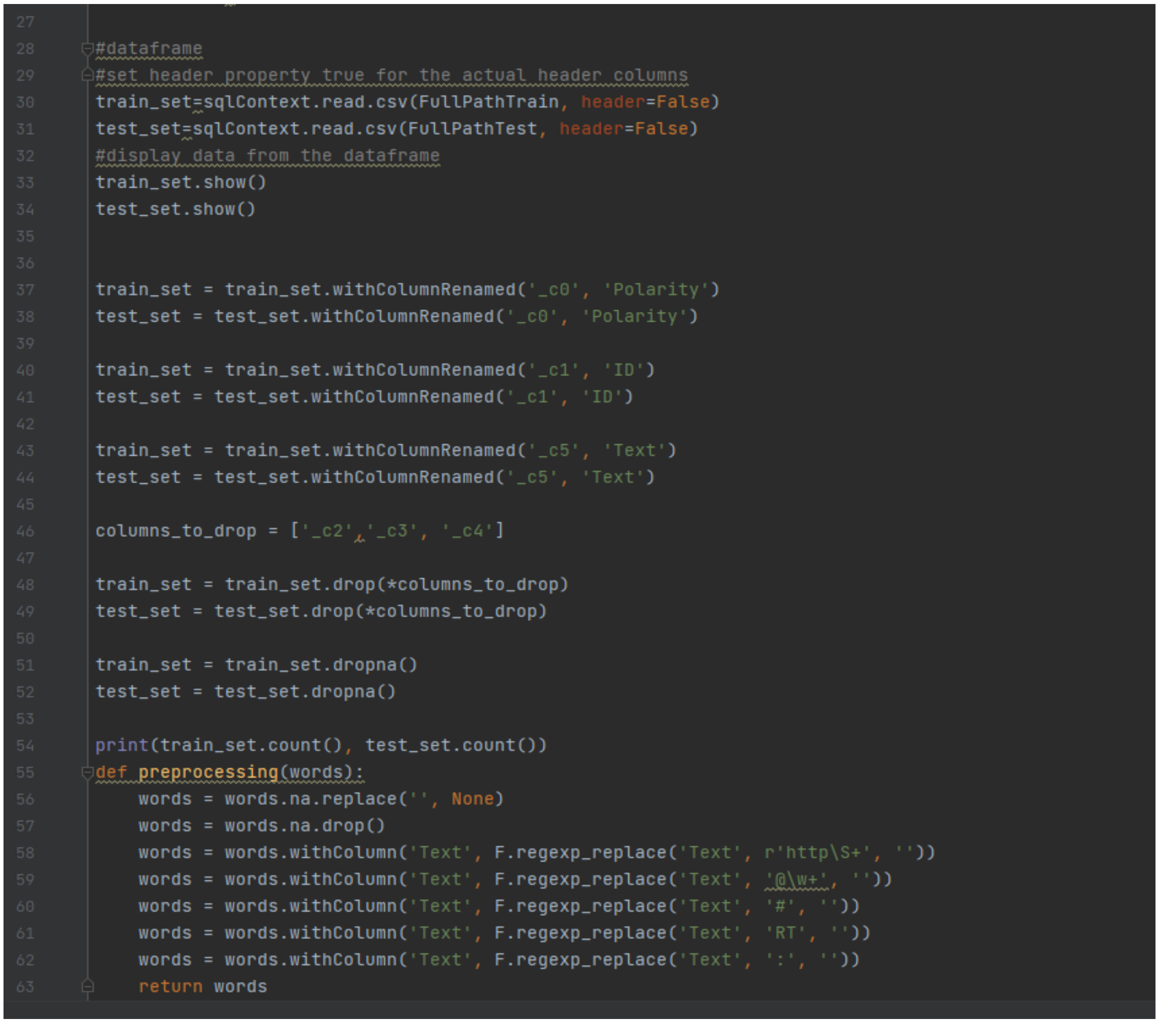
And the result of accuracy from cross validation was:



```
[0.97571429 0.97380952 0.97142857 0.97189138 0.96855646]
```

Tweet preprocessing and sentiment analysis

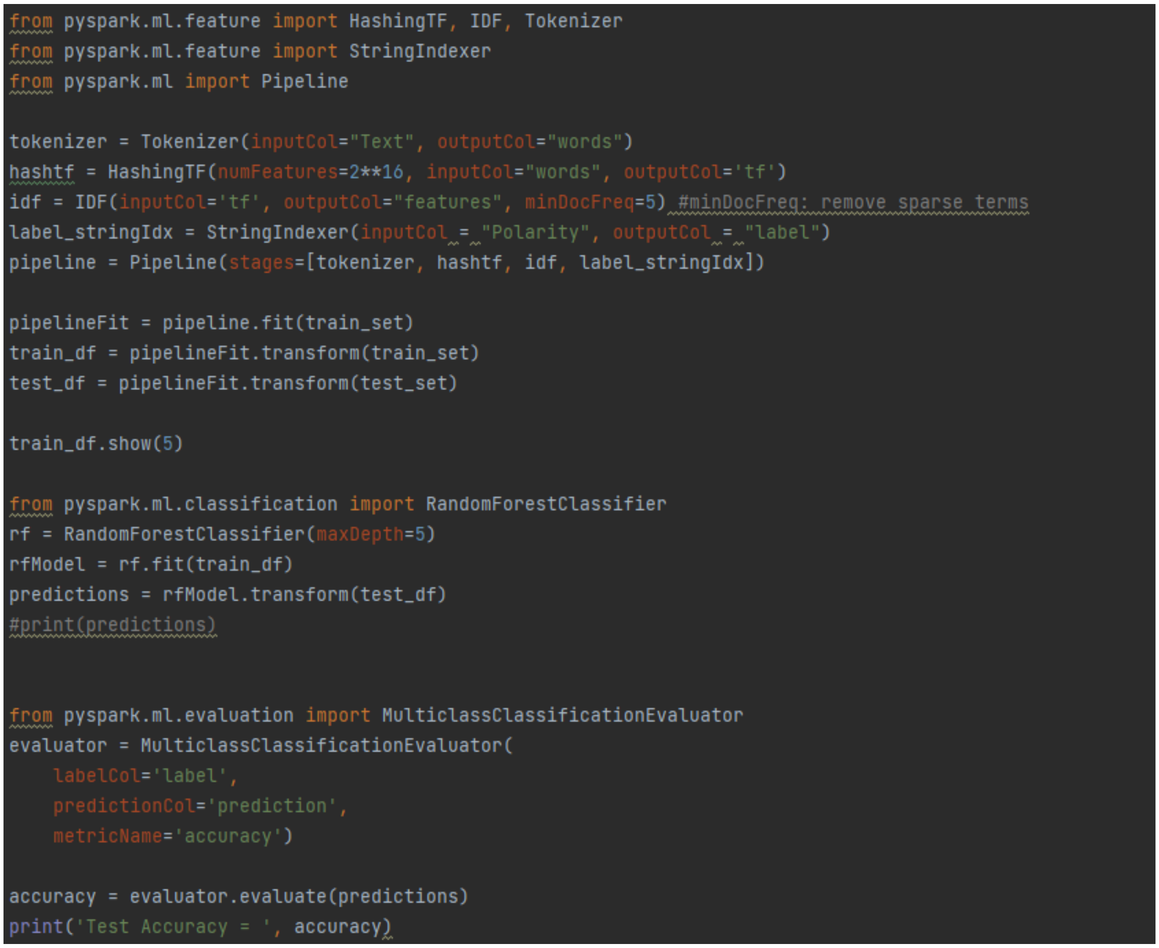
We use pyspark , which is the Python API for Spark. Here, we use **Spark Structured Streaming**, which is a stream processing engine built on the Spark SQL engine and that's why we import the pyspark.sql module. We import its classes; SparkSession to create a stream session, function, and types to make a list of built-in functions and data types available. We preprocess the tweets so we can have only the clean text of the tweet. In each batch, we receive many tweets from the Twitter API and split the tweets at the string t_end. Then, we remove the empty rows and apply regular expressions to clean up the tweet text. In more detail; we remove the links (https://..), the usernames (@..), the hashtags (#), the string that shows if the current tweet is a retweet (RT), and the character :.



```python

#dataframe
#set header property true for the actual header columns
train_set=sqlContext.read.csv(FullPathTrain, header=False)
test_set=sqlContext.read.csv(FullPathTest, header=False)
#display data from the dataframe
train_set.show()
test_set.show()


train_set = train_set.withColumnRenamed('_c0', 'Polarity')
test_set = test_set.withColumnRenamed('_c0', 'Polarity')

train_set = train_set.withColumnRenamed('_c1', 'ID')
test_set = test_set.withColumnRenamed('_c1', 'ID')

train_set = train_set.withColumnRenamed('_c5', 'Text')
test_set = test_set.withColumnRenamed('_c5', 'Text')

columns_to_drop = ['_c2','_c3', '_c4']

train_set = train_set.drop(*columns_to_drop)
test_set = test_set.drop(*columns_to_drop)

train_set = train_set.dropna()
test_set = test_set.dropna()

print(train_set.count(), test_set.count())
def preprocessing(words):
    words = words.na.replace('', None)
    words = words.na.drop()
    words = words.withColumn('Text', F.regexp_replace('Text', r'http\S+', ''))
    words = words.withColumn('Text', F.regexp_replace('Text', '@\w+', ''))
    words = words.withColumn('Text', F.regexp_replace('Text', '#', ''))
    words = words.withColumn('Text', F.regexp_replace('Text', 'RT', ''))
    words = words.withColumn('Text', F.regexp_replace('Text', ':', ''))
    return words
```

- **Errors**

I had an issue running the code for classification and couldn’t resolve it:



```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline

tokenizer = Tokenizer(inputCol="Text", outputCol="words")
hashtf = HashingTF(numFeatures=2**16, inputCol="words", outputCol='tf')
idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5) #minDocFreq: remove sparse terms
label_stringIdx = StringIndexer(inputCol = "Polarity", outputCol = "label")
pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label_stringIdx])

pipelineFit = pipeline.fit(train_set)
train_df = pipelineFit.transform(train_set)
test_df = pipelineFit.transform(test_set)

train_df.show(5)

from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(maxDepth=5)
rfModel = rf.fit(train_df)
predictions = rfModel.transform(test_df)
#print(predictions)


from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(
    labelCol='label',
    predictionCol='prediction',
    metricName='accuracy')

accuracy = evaluator.evaluate(predictions)
print('Test Accuracy = ', accuracy)
```

py4j.protocol.Py4JJavaError: An error occurred while calling o448.evaluate.

Then I couldn’t predict the streaming data and save them because the code I used with the vectorization *tfidfTransformer* because the algorithm couldn’t find the new words that weren’t on the training data and categorize them.