Streaming and Analysis of Twitter Data

Data Engineering Project By :-
Kashish (11011811)
Shekhar Singh (11011694)
Siddarth Venkateswaran (11011753)

Table of Authorship

I. Introduction:

This project consists of creating a pipeline for live Twitter streams for current trending topics. The main areas of focus are:

1) Generating the current trending topics from a location by making a Twitter API call
2) Extracting live Twitter streams for select two trending topics for a location
3) Making use of a distributed streaming platform to produce, store and consume collected tweets in the form of messages
4) Ingest the consumed messages into a database
5) Answer 5 business questions on the data that has been ingested into the database.

In the sections below, the procedure required to build and run this pipeline are explained in detail.

II. Process Overview:

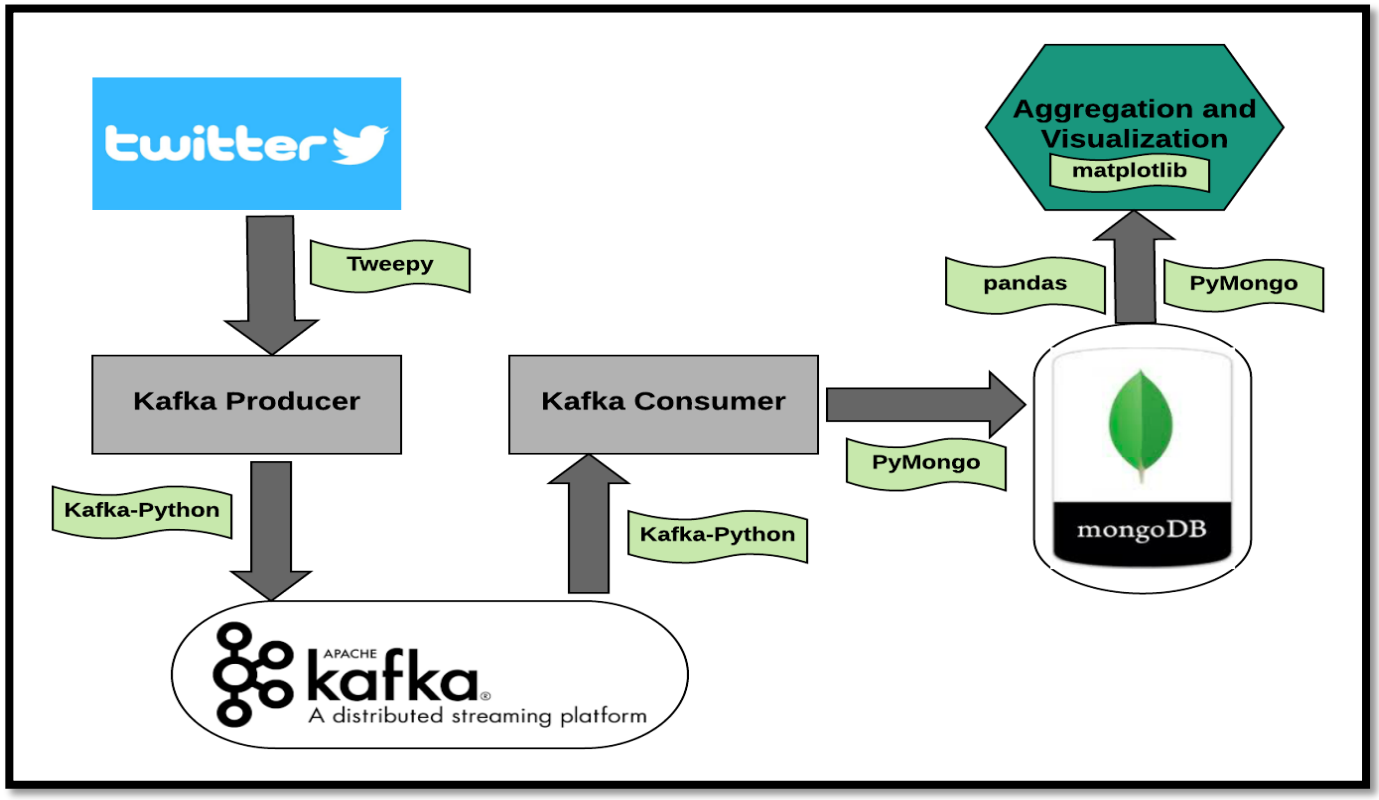

Figure 1: Process overview of the pipeline

The above flowchart depicts the process that was followed in this pipeline in order to extract and analyze tweets:

1) Live-Streaming of Tweets:

Using the Twitter API call, we first generated what the current trending topics were for a particular location. We took the top two trending topics as keywords and collected tweets based on them for a set period of time. The producer then sent these tweets to the Kafka Broker in the form of messages to a topic name that we assigned.

2) Consuming Messages:

All the tweets stored in a topic in the broker were sent in the form of messages to the Kafka consumer.

3) Data Analysis:

Once all the tweets were consumed by the Kafka Consumer, they were ingested into the database in the JSON format, after which queries were run in order to answer 5 business questions.

III. Data Source Used: Twitter Stream

Twitter is an online news and social networking service on which people or entities in the form of 'Twitter Handles' interact with each other with messages called as 'Tweets'. Each tweet can have a maximum of 280 characters. These can be extracted by making a Twitter API call using Python's Tweepy library. Each tweet structure is embedded in a JSON format from which we get various parameters like the user who posted it, the tweet content, number of likes and shares, location and source from which it was posted etc. Using these details, we can perform various analysis related to the tweet or the twitter handle. Below is a snippet showing the JSON structure of a tweet:


```
{
    "_id" : ObjectId("5c374cd2bbc0995680a90607"),
    "created_at" : "Thu Jan 10 13:35:21 +0000 2019",
    "id" : NumberLong(1083356625237352449),
    "id_str" : "1083356625237352449",
    "text" : "Brain Teaser!\n\nTell us...\n#destinychristal #brainteaser #tbt❤️ #thursday #thursdaythoughts https://t.co/ocAOGhuV5j",
    "display_text_range" : [
        0,
        90
    ],
    "source" : "<a href=\"http://twitter.com/download/android\" rel=\"nofollow\">Twitter for Android</a>",
    "truncated" : false,
    "in_reply_to_status_id" : null,
    "in_reply_to_status_id_str" : null,
    "in_reply_to_user_id" : null,
    "in_reply_to_user_id_str" : null,
    "in_reply_to_screen_name" : null,
    "user" : {
        "id" : NumberLong(1004361236556013569),
        "id_str" : "1004361236556013569",
        "name" : "Destiny Christal Engineering services ltd",
        "screen_name" : "DestinychrisLtd",
        "location" : "Ikeja, Nigeria",
        "url" : "http://www.destinychristalengr.com",
        "description" : "We are a group of Real Estate Experts who are into Real Estate construction, development and other Engineering services.\n\nContact us on 07032986725",
        "translator_type" : "none",
        "protected" : false,
        "verified" : false,
        "followers_count" : 621,
        "friends_count" : 711,
        "listed_count" : 1,
        "favourites_count" : 26,
        "statuses_count" : 385,
        "created_at" : "Wed Jun 06 13:55:53 +0000 2018",
        "utc_offset" : null,
        "time_zone" : null,
        "geo_enabled" : false,
        "lang" : "en",
        "contributors_enabled" : false,
        "is_translator" : false,
        "profile_background_color" : "F5F8FA",
        "profile_background_image_url" : "",
        "profile_background_image_url_https" : "",
        "profile_background_tile" : false,
        "profile_link_color" : "1DA1F2",
        "profile_sidebar_border_color" : "C0DEED",
        "profile_sidebar_fill_color" : "DDEEF6",
        "profile_text_color" : "333333",
        "profile_use_background_image" : true,
        "profile_image_url" : "http://pbs.twimg.com/profile_images/1019964257365045251/4m9g42gc_normal.jpg",
        "profile_image_url_https" : "https://pbs.twimg.com/profile_images/1019964257365045251/4m9g42gc_normal.jpg",
        "profile_banner_url" : "https://pbs.twimg.com/profile_banners/1004361236556013569/1532329790",
        "default_profile" : true,
        "default_profile_image" : false,
        "following" : null,
        "follow_request_sent" : null,
        "notifications" : null
    },
    "geo" : null,
    "coordinates" : null,
    "place" : null,
    "contributors" : null,
    "is_quote_status" : false,
    "quote_count" : 0,
    "reply_count" : 0,
    "retweet_count" : 0,
    "favorite_count" : 0,
    "entities" : {
        "hashtags" : [
            {
                "text" : "destinychristal",
                "indices" : [
                    26,
                    42
                ]
```


Figure 2: JSON structure of a tweet

IV. Tools Utilized:

To build this pipeline, the following tools were used:

#	Name	Description	Used For
1	Mac OS Terminal	Since our pipeline was built using a MacBook, we used the Mac OS Terminal which is a powerful application, using which we could run various commands to either run a program or to install new packages.	Installing Zookeeper, Kafka and various Python packages. Also used to test the performance of the above-mentioned applications as well as to perform a few queries on the database.
2	Jupyter Notebook	Jupyter Notebook is an open-source web application through which we could code, document as well as analyze, all at one place.	Building the entire pipeline, right from extracting tweets to analyzing them.
3	Kafka	Kafka is a distributed streaming platform through which we could publish, subscribe, store and process streams of records.	Storing, publishing and consuming streams of tweets collected using Twitter API.
4	MongoDB	MongoDB is a NoSQL database using which we could store huge amounts of unstructured data and analyze them.	Storing tweets collected by the Kafka consumer in a JSON format and performing analysis on them.
5	Robo 3T	An open-source MongoDB management tool.	Easier understanding of the tweet structure and cross verifying the data.

V. Libraries Used:

The following Python[1] libraries were used to build our pipeline:

Sl No.	Library Name	Usage	Installation Command
1	Tweepy	Used for accessing Twitter API	pip install tweepy
2	Kafka-Python	A Python client for the Apache Kafka stream processing system	pip install kafka-python
3	PyMongo	It contains various tools used for working with MongoDB, a NoSQL database	pip install pymongo
4	Pandas	An open sourced library which provides easy-to-use data-structures and data analysis tools for Python programming	pip install pandas
5	Matplotlib	Used for data visualization in a graphical format	pip install matplotlib
6	Datetime	A package to manipulate date and times	pip install DateTime
7	JSON	A built-in package used for encoding or decoding data in JSON format	pip install jsonlib-python3
8	Beautiful Soup	A package for scraping web content	Pip install beautifulsoup4

[1] Python library installations were done by referring to the Python Package Index(PyPI) page.

VI. Setting Up Kafka:

The following section describes the steps needed to setup Kafka in our systems:

A. Installing and running Kafka and Zookeeper:

While Kafka[2] is a distributed streaming platform through which we can publish, subscribe, store and process streams of records, Zookeeper[3] is an equally important application meant for broker management within the Kafka ecosystem. Since we've built our pipeline using MacBook, these applications were installed using Homebrew, which is an open-source software package management system that is used to install various applications in macOS.

Also note, in order to install Kafka, we should also install Java Development Kit as a prerequisite.

The commands are as follows:

$ brew cask install java
$ brew install kafka

Once these applications were installed, two separate instances of terminals – one for Zookeeper and one for Kafka – were opened and the following commands were executed:

$ zookeeper-server-start /usr/local/etc/kafka/zookeeper.properties
$ kafka-server-start /usr/local/etc/kafka/server.properties

These were done in order to run the Zookeeper and Kafka servers in the background that is vital to run our pipeline.

B. Testing Kafka Application:

In order to test the Kafka application, two consoles – one for Kafka Producer and one for Kafka Consumer - were initiated in order to send and receive messages:

1. Kafka Producer Console:

In order to initialize the Kafka Producer Console, the following command was executed in a newly opened terminal:

$ kafka-console-producer --broker-list localhost:9092 --topic test
>---Type a test message here---

Here, the producer console will listen to the localhost at port 9092. Also a new topic called 'test' is created in which all the messages will be stored and sent to the broker.

[2] Kafka installation and testing was performed using the steps mentioned in this medium article.
[3] Importance of Zookeeper for Kafka was understood from the following Stack Overflow discussion.

2. Kafka Consumer Console:

In order to initialize the Kafka Consumer Console, the following command was executed in a newly opened terminal:

$ kafka-console-consumer --bootstrap-server localhost:9092 --topic test --from-beginning
---messages entered in the producer are received here---

Here the consumer console will listen to the localhost at port 9092, and all messages stored in the topic 'test' in the Kafka broker will be consumed here.

If all the messages entered in the producer console are read in the consumer console, it indicates that Kafka has been installed and is running successfully.

VII. Making the Twitter API Call:

One must have a Twitter account and generate the access keys from the Twitter Developer platform as prerequisites to make a Twitter API[4] call. The procedure to generate them are as follows:

1) Visit the Twitter Developers page using this link
2) Click on Apps from your user-name dropdown
3) Click on 'Create an app'
4) Fill up all the mandatory details and click on the 'Create' button

The following keys will be generated which will authenticate the Python program to access the live stream of tweets:

```
## Twiter API Access Details ##
access_token="##access_token##"
access_token_secret="##access_token_secret##"
consumer_key="##consumer_key##"
consumer_secret="##consumer_secret##"
```

Figure 3: Access keys that are generated after creating a Twitter application

From the above keys, we assign the following variables:

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
```

Figure 4: Variables created to complete the authentication process.

[4] Making a Twitter API call and building the Kafka Producer were done by referring to this blog.

From the above code snippet:

1) auth is a variable created for Twitter authentication using the credentials that we have provided. This will be done with the help of OAuthHandler class by providing consumer_key and consumer_secret credentials as the arguments
2) auth.set_access_token is a method provided from the OAuthHandler class, created to complete the authentication process. It takes in the access_token and access_token_secret as the arguments

VIII. Generating the current trends:

Every time one logs on to twitter, there will be a section on the left pane displaying the current trends[5] based on the user's location which is either set as default, or has been edited by the user. These trends are the hottest topics of discussion and can generate the maximum tweets when filtered by them. Our Pipeline was built around generating and analyzing tweets for these trending topics based on entering a particular location.

A. Libraries to be imported:

We began by importing the following libraries:

```
import tweepy
from tweepy import OAuthHandler
import pandas as pd
from pymongo import MongoClient
```

Figure 5: Libraries needed to generate the current trending topics

Where:

1) Tweepy is used for accessing the Twitter API
2) OAuthHandler is a class responsible for authenticating the python program to access Twitter based on the credentials provided by us
3) Pandas is a library using which we can generate data frames
4) Pymongo is a Python package for working with MongoDB.

B. Accessing the WOEID:

In order to generate the trending topics from a location, we need to access the Where On Earth Identifier (WOEID)[6], which is a 32-bit reference identifier assigned by Yahoo!, that identifies any feature on earth. These values were accessed and stored in a new collection in our database. This ID was accessed from the database using the following lines of code:

[5] Getting Twitter trends using Tweepy was executed by referring to this Stack Overflow discussion.
[6] The JSON format of Yahoo's WOEID was extracted from this page.

```
## Fetching details for city and its woeid into memory ##
client = MongoClient('localhost:27017')
db = client.DataEngineering
collection = client.DataEngineering.CityNames

city_details = collection.find({}, {"_id" :0, "city" :1, "woeid" :1})
formated_city_details = pd.DataFrame.from_records(city_details)
```

Figure 6: Fetching the WOEID's of each city from the database

From the above code snippet:

1) Client is a MongoDB instance which will listen to the localhost that we've assigned to it
2) db is the name of the database which will have all the required collections related to this pipeline. Here we've used DataEngineering as the name of the database.
3) Collection is the dump within the database in which all the messages from the broker will be stored in a JSON format. Here we've used CityNames as the name of the collection that contains the raw dump of all cities and their corresponding WOEIDs.

C. Printing the Current Trends:

Entering the City Name is one of the important parameters in our pipeline. Once we enter the it, the Python code will access the database for its corresponding WOEID, based on which the current trending topics for that location are generated by making a Twitter API call. Below is the code snippet using which we implemented it:

```
## Taking city name as an input from user ##
city_name = input("Please enter the city name of your choice: ")
print ("\nBelow is the trending topics for city " + city_name + " !" )
for index in range(0,467):
    if city_name == formated_city_details['city'][index]:
        woe_id = formated_city_details['woeid'][index] ## Taking woeid of the city name provided by user ##

print("\n")
trending_topics = api.trends_place(woe_id)
data = trending_topics[0]
trends = data['trends'] ## fething the trends ##
names = [trend['name'] for trend in trends] ## fetching the name from each trend ##
trendNames = ' '.join(names) ## joining all the trend names together with a ' ' separating them ##
print(trendNames)
```

Figure 7: Generating the current trending topics based on the location that was entered as an input

For instance, when we entered the City Name as Berlin, the above lines of code captured the WOEID of Berlin that was stored in the database. Using this WOEID, the following trending topics were be generated:

```
Please enter the city name of your choice: Berlin

Below is the trending topics for city Berlin !


#Schneechaos #goldmünze #HandballWM #Bachelor #StarkeFamilienGesetz Instrument der Spaltung Ultraschall Noten Zucker
Promille Limonade Deutsche Industrie sturz raubüberfall Kinderarmut flughäfen #checkpoint #Kantholz #Magnitz #icymi #
Bekennerschreiben #Rabiot #KiKvorGericht #dogsarebetterthanpeople #Poggenburg #aufgehtsDHB #Schneefall #CORGER #Akten
zeichenXY #brennpunkt #streik #Winter #Pavard #lanz #AfDLieder #iHeartAwards #ThursdayMotivation #ThursdayThoughts #e
rnährungsreport #CES2019 #lemonaid #schindlersliste #HelloMyLove #PortfolioDay #BodeMuseum #fakinghitler #shutdown #A
usOpen #FinTech #Digitalisierung
```

Figure 8: Trending topics generated for Berlin on the 10th of January, 2019

IX. Building the Kafka Producer:

The following steps were followed in order to build the Kafka Producer[7] using Kafka-Python in order to extract a live stream of tweets for two trending topics that were used as keywords, assigning it a topic name, and sending it in the form of messages to the Kafka Broker:

A. Assigning a Topic:

In order to extract tweets for a set of keywords and store them within a topic, the following lines of code were entered:

```
## Taking keywords and topic name as input from user ##
keyword_one = input("Please enter a keyword: ")
keyword_two = input("\nPlease enter another keyword: ")
topic_name = input("\nPlease assign a topic name of your choice: ")
print ("\nSearching for tweets which contain keywords " + keyword_one + " and " + keyword_two + " !" )
```

Figure 9: Inserting the trending topics as keywords and assigning it a topic name

Here for example, when we entered keyword_one as “Thursday”, keyword_two as “Pelosi” and topic as “TrendingTopics”, the Twitter API captured all live tweets related to these keywords, inserted them under the given topic name in the Kafka producer, and sent them to the Kafka Broker.

B. Defining the Start Time:

Here we intended to capture the tweets only for two trending topics for a limited time period, due to space constraints within our systems. In order to test that, we defined the time at which the tweets were starting to get captured. These were implemented using the below lines of code:

```
StartTime = datetime.now()
print("\nKafka Producer has been started at: ",StartTime.strftime("%c"))  ## Prints the start time of Kafka Producer ##
```

Figure 10: Capturing the time at which the Kafka Producer started sending the tweets to the Kafka Broker

C. Setting a Time Limit:

Due to uncertainty of the number of tweets that could be generated (a topic like Trump could generate nearly one hundred thousand records within minutes) and due to space constraints within our systems, we set a time limit for collecting tweets for only 10 minutes using the below lines of code:

[7] Understanding the nitty-gritties of making a Twitter API call and building a Kafka Producer using Python were done by watching this video.

```
class StdOutListener(StreamListener):
    def on_data(self, data):
        if (time.time() - self.start_time) < self.limit:
            producer.send_messages(topic_name, data.encode('utf-8'))
            #print (data)
            return True
        else:
            print
            return False

    def on_error(self, status):
        print (status)

    def __init__(self, time_limit=600):     ## Setting the run time limit in seconds ##
        self.start_time = time.time()
        self.limit = time_limit
        super(StdOutListener, self).__init__()
```

Figure 11: Setting a time limit to capture the tweets

From the above code snippet:

1) StreamListener is a class from the Tweepy module that will listen to the live stream of tweets.

2) StdOutListener is a class that will inherit tweets from the StreamListener.

3) on_data is a method that takes in data that is streamed in from the streamlistener (in this case tweets) and we can perform any action with it.

4) on_error is a method that we're overriding from the streamlistener class that happens if there are any errors. If any error occurs, it will be displayed in the form of a status message on the screen.

5) Using the __init__ constructor method, we're setting the time limit parameter.

Once the time limit exceeded 10 minutes, the Twitter API stopped sending live tweet streams to the Kafka producer.

D. Defining the Kafka Producer:

Once the topic name, keywords and time limit were set, we needed to specify the Kafka broker, create the authentication objects, set our access tokens and finally create a function in which the Twitter API would read a stream of tweets by taking all these parameters into consideration. This was executed using the following lines of code:

```
kafka = KafkaClient("localhost:9092")
producer = SimpleProducer(kafka)
l = StdOutListener()
auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
stream = Stream(auth, l)
stream.filter(track=[keyword_one, keyword_two])
```

Figure 12: Defining the Kafka Producer

From the above code snippet:

1) The Kafka Client will listen to the local host at port 9092.

2) 'l' is a listener object created that we've inherited from the StdOutListener class.

3) 'auth' is a variable created for authentication using the credentials that we have provided. This will be done with the help of OAuthHandler class by providing consumer_key and consumer_secret credentials as the arguments.

4) OAuthHandler is a class responsible for authenticating based on the twitter application credentials provided by us.

5) 'auth.set_access_token' is a method provided from the OAuthHandler class, created to complete the authentication process. It takes in the access_token and access_token_secret as the arguments.

6) 'stream' is a variable created using the Stream class that we imported above, in which we provide the authentication token(auth) and the listener object(l) that is responsible for data handling.

7) 'stream.filter' is a method provided by the Stream class that helps us to filter through tweets related to the keywords that we've entered here.

E. Defining the End Time:

As mentioned earlier, since we were capturing tweets only for a set period of time, along with the start time, we also captured the end time to test if the tweets that were captured as messages in the Kafka producer were done so only for the time limit that was mentioned earlier. This was executed using the following lines of code:

```
EndTime = datetime.now()
print("\nKafka Producer has been stopped at: ",EndTime.strftime("%c"))  ## Prints the end time of Kafka Producer ##
```

Figure 13: Capturing the time at which the Kafka Producer stopped sending tweets to the Kafka Broker

Using the above lines of code explained in sections VIII A through F, we entered the following parameters:

1) Keyword_one = Thursday
2) Keyword_two = Pelosi
3) Topic_name = TrendingTopics
4) Time_limit = 600 seconds

Based on the above parameters, we got the following output:

```
Please enter a keyword: Thursday

Please enter another keyword: Pelosi

Please assign a topic name of your choice: TrendingTopics

Searching for tweets which contain keywords Thursday and Pelosi !

Kafka Producer has been started at:  Thu Jan 10 14:35:25 2019

Kafka Producer has been stopped at:  Thu Jan 10 14:45:25 2019
```

Figure 14: Printing the start time and the end time of the Kafka Producer

F. Testing the Kafka Producer:

In order to test the Kafka Producer that we built using the lines of code above, we opened a Kafka Consumer Console and ran the following command in order to read all the tweets in the form of messages:

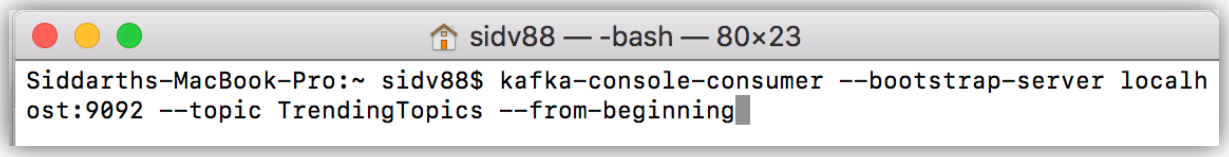

Figure 15: Command line to check the messages sent by the Kafka Producer

Based on the above command, we got the JSON structure of all the tweets that were collected during the mentioned time period of 10 minutes. Below is a snippet of the same:

Figure 16: Sample JSON structure of a tweet sent as a message from the Kafka Producer

X. Building the Kafka Consumer:

The following steps were followed to build the Kafka Consumer[8] in order to consume all messages stored in a topic in the Kafka Broker:

A. Libraries to be imported:

In order to build a Kafka Consumer, we imported the following libraries:

```
from kafka import KafkaConsumer
from pymongo import MongoClient
from json import loads
from datetime import datetime
```

Figure 17: Libraries needed to build the Kafka Consumer

Where:

1) KafkaConsumer from the python-kafka package will consume all messages from the Kafka Broker stored under the topic 'TrendingTopics'
2) MongoClient from Pymongo will create a new collection in the database and ingest these messages in them
3) Json loads will decode the messages that were encoded in the Kafka Producer and
4) datetime is used to set a time limit for tweet consumption.

B. Topic to be Consumed:

In order to consume the tweets that were stored as messages in the Kafka Broker, we had to enter the Topic Name from which we had to access these values. This was executed using the below lines of code:

```
## Taking topic name to be consumed as input from user ##
topic_name = input("Please enter topic name which you want Kafka to consume: ")
print ("\nConsuming data for topic " + topic_name + " !" )
```

Figure 18: Topic Name from which the tweets stored as messages need to be consumed

C. Defining the Start Time:

Here we intend to run the consumer for a limited time period. In order to test it, the time at which the Kafka Consumer started consuming the messages were captured using the below lines of code:

```
TweetsWritten = 0
StartTime = datetime.now()
print("\nKafka Consumer has been started at: ",StartTime.strftime("%c"))  ## Prints the start time of Kafka Consumer #
```

Figure 19: Capturing the time at which the Kafka Consumer started consuming the tweets from the Kafka Broker

[8] Building the Kafka Consumer and ingesting the tweets into MongoDB were made by referring to this Medium article.

D. Defining the Kafka Consumer:

In order to define the Kafka Consumer, we had to enter certain parameters using which we could consume all tweets stored in the form of messages in the Kafka Broker. These were executed using the below lines of code:

```
consumer = KafkaConsumer(
    topic_name,        ## topic name in Kafka ##
     bootstrap_servers=['localhost:9092'],
     auto_offset_reset='earliest',
     enable_auto_commit=True,
     consumer_timeout_ms= 90 * 1000,    ## Setting the run time limit in milliseconds ##
     #group_id='my-group',
     value_deserializer=lambda x: loads(x.decode('utf-8')))
```

Figure 20: Defining the Kafka Consumer

From the above code snippet:

1) Topic_name is the name of the topic in which all the tweets to be consumed were stored,
2) Bootstrap_server is the host port that the consumer should contact to get the metadata of the stored tweets
3) Auto_offset_set was set to 'earliest' as a safety measure in order to consume tweets from the beginning in case of an offset
4) Enable_auto_commit was set to True in order to avoid message duplication
5) Consumer_timeout_ms is the time period in milliseconds for which we consumed the tweets from the broker
6) Value_deserializer decoded the messages that were encoded while inserting them in the broker.

E. Defining the End Time:

In order to test the if the Kafka Consumer had run only for the set duration that we had mentioned in the consumer_timeout_ms variable while defining it, we captured the end time along with the start time that was captured earlier. This was executed using the below lines of code:

```
EndTime = datetime.now()
print("\nKafka Consumer has been stopped at: ",EndTime.strftime("%c"))  ## Prints the end time of Kafka Consumer ##
```

Figure 21: Capturing the time at which the Kafka Consumer stopped consuming the tweets from the Kafka Broker

XI. Data Ingestion:

Once the messages were consumed by the Kafka Consumer, they were ingested into a database for further analysis. Here we made use of MongoDB. They were done as follows:

A. Defining the Mongo Client:

In order to define the Mongo Client, we had to enter a set of parameters using which we could ingest the tweets in the database. Here we made use of MongoDB which is a NoSQL database that's used to store huge amounts of unstructured data. These were implemented using the below lines of code:

```
## Setting up connection with MongoDB ##
client = MongoClient('localhost:27017')
db = client.DataEngineering
collection = client.DataEngineering.topic_name
db.topic_name.drop()
```

Figure 22: Data ingestion in the 'topic_name' collection in the database

From the above code snippet:

1) Client is a MongoDB instance which would listen to the localhost that we had assigned to it
2) Db is the name of the database which would have all the required collections related to this pipeline. Here we used DataEngineering as the database name.
3) Collection is the dump within the database in which all the messages from the broker would be stored in a JSON format. Here we used topic_name as the name of the collection.

B. Counting the number of messages written:

In order to keep a live track of the number of tweets ingested into MongoDB, the following lines of code were executed through which we could track each record that was entered from the broker on to the database:

```
for tweet in consumer:
    tweet = tweet.value
    collection.insert_one(tweet)
    TweetsWritten = TweetsWritten + 1
    print("\nTweet number %d written" %(TweetsWritten))

print("\nWritten %d tweets into MongoDB" %(TweetsWritten)) ## Prints the number of records written into MongoDB ##
```

Figure 23: Capturing the number of tweets ingested into the database

Below is a snippet of the output that we got upon executing the above lines of code:

```
Tweet number 5271 written

Tweet number 5272 written

Tweet number 5273 written

Tweet number 5274 written

Written 5274 tweets into MongoDB

Kafka Consumer has been stopped at:  Thu Jan 10 14:48:36 2019
```

Figure 24: Final output displaying the number of tweets collected and ingested into the database

From the above figure, we can see that there was a total of 5274 tweets collected in the database for the given set of keywords on which further analysis was performed.

XII. Analysis:

Once the tweets were ingested into the database, we could come up with the following business questions[9] that could be answered using a few simple queries:

A. Top 10 Languages:

Each tweet structure has a 'lang' parameter which indicates the language in which the tweet has been generated.

```
import pandas as pd
import matplotlib.pyplot as plt
from pymongo import MongoClient

client = MongoClient('localhost:27017')
db = client.DataEngineering
collection = client.DataEngineering.topic_name

print("List of top 10 languages used in these tweets apart from English and Undefined:\n")
language_count = collection.aggregate([{'$group' : {'_id' : '$lang', 'count' : {'$sum' : 1}}},
                                       {'$match' : {'count' : {'$gt' : 1}}},
                                       {'$sort' : {'count' : -1}},
                                       {'$limit' : 12},
                                       {'$project' : {'language' : '$_id', 'count' : 1, '_id' : 0}}])
language = pd.DataFrame.from_records(language_count)
language = language.iloc[2:]
print(language)

#Plotting the graph
language_plot=language.plot.bar(x='language', y='count', title="Top 10 Languages", width=0.75, figsize=(15, 7.5),
                                rot=25, color='goldenrod')
language_plot.set_xlabel("Language", fontsize=20)
language_plot.set_ylabel("Count", fontsize=20)

for data in language_plot.patches:
    language_plot.annotate(int(data.get_height()), (data.get_x() * 1.0, data.get_height() * 1.01))
plt.show()
```

Figure 25: Query to capture the top 10 languages in which the captured tweets were posted

From the above query, we could generate the following data frame and chart:

List of top 10 languages used in these tweets apart from English and Undefined:

	count	language
2	60	ar
3	38	tl
4	13	ja
5	10	es
6	5	in
7	4	pl
8	4	fr
9	4	de
10	3	it
11	3	ko

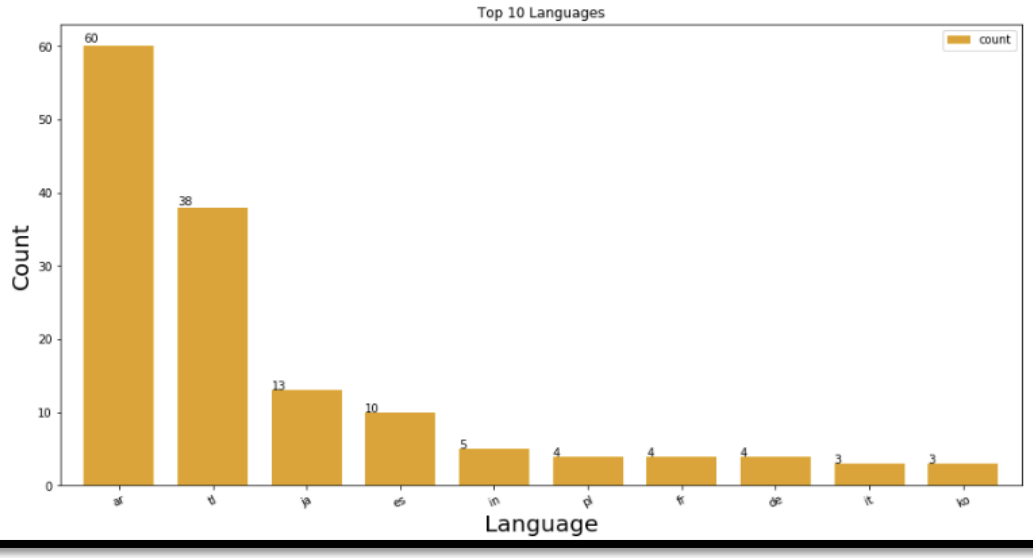

Figure 26: Data Frame and Chart depicting the top 10 languages in which the collected tweets were posted

[9] We got an idea of performing tweet analysis by referring to this article.

Of the 5274 tweets that were ingested into the database, we could see that apart from English and Undefined which constituted 90% of collected data, the next highest was Arabic at 60 followed by Thai at 38.

B. Top 10 Hashtags:

One of the easiest ways to deep-dive into the select set of trending topics is by filtering through hashtags related to it. One can extract the hashtags either from the text of the tweet or from the 'hashtags' parameter that is embedded within the 'entities' parameter of a tweet JSON structure.

```
import pandas as pd
import matplotlib.pyplot as plt
from pymongo import MongoClient

client = MongoClient('localhost:27017')
db = client.DataEngineering
collection = client.DataEngineering.topic_name

print("List of top 10 hashtags used in these tweets:\n")
hashtag_count = collection.aggregate([{'$unwind' : '$entities.hashtags'},
                                      {'$group' : {'_id' : '$entities.hashtags.text', 'TagCount' : {'$sum' : 1}}},
                                      {'$sort' : {'TagCount' : -1}},
                                      {'$limit' : 10},
                                      {'$project' : {'hashtags' : '$_id', 'TagCount' : 1, '_id' : 0}}])
hashtag = pd.DataFrame.from_records(hashtag_count)
print(hashtag)

#Plotting the graph
hashtag_plot=hashtag.plot.bar(x='hashtags', y='TagCount', title="Top 10 Hashtags", width=0.75, figsize=(15, 7.5),
                              rot=25, color='khaki')
hashtag_plot.set_xlabel("Hashtags", fontsize=20)
hashtag_plot.set_ylabel("TagCount", fontsize=20)

for data in hashtag_plot.patches:
    hashtag_plot.annotate(int(data.get_height()), (data.get_x() * 1.0, data.get_height() * 1.01))
plt.show()
```

Figure 27: Query to capture the top 10 hashtags that were used from the collected set of tweets

From the above query, we could generate the following data frame and chart depicting the top 10 hashtags from the live-stream of tweets that we had extracted:

List of top 10 hashtags used in these tweets:

	TagCount	hashtags
0	47	Trump
1	37	Thursday
2	35	FakeNews
3	33	tbt
4	28	thursday
5	26	BuildTheWall
6	25	Sarcasm
7	24	Pelosi
8	20	MAGA
9	18	ALDUB182ndWeeksary

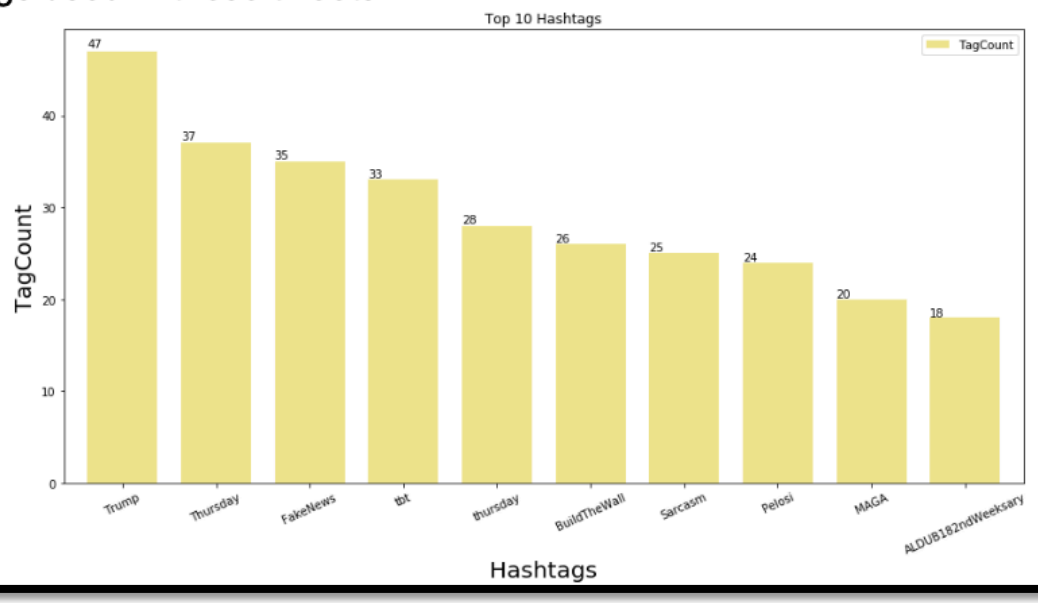

Figure 28: Data Frame and Chart depicting the top 10 hashtags that were used for the collected set of tweets

From the above chart, we can see that the highest number of hashtags that was used for the collected set of 5274 tweets was for Trump at 47 followed by Thursday at 37.

C. Top 10 Sources:

From the 'source' parameter, we can check the device from which a tweet was generated.

```
import pandas as pd
import matplotlib.pyplot as plt
from pymongo import MongoClient

client = MongoClient('localhost:27017')
db = client.DataEngineering
collection = client.DataEngineering.topic_name

print("List of top 10 sources that these tweets have been tweeted from:\n")
from bs4 import BeautifulSoup
source_count = collection.aggregate([{'$group' : {'_id' : '$source', 'count' : {'$sum' : 1}}},
                                     {'$sort' : {'count' : -1}},
                                     {'$limit' : 10},
                                     {'$project' : {'source' : '$_id', 'count' : 1, '_id' : 0}}])
source_detail = pd.DataFrame.from_records(source_count)
source_name = source_detail.source
source_detail['source'] = [BeautifulSoup(source_name).getText() for source_name in source_detail['source']]
print(source_detail)

#Plotting the graph
source_plot=source_detail.plot.bar(x='source', y='count', title ="Top 10 Sources", width=0.75, figsize=(15, 7.5),
                                   rot=25, color='chocolate')
source_plot.set_xlabel("Source", fontsize=20)
source_plot.set_ylabel("Count", fontsize=20)

for data in source_plot.patches:
    source_plot.annotate(int(data.get_height()), (data.get_x() * 1.0, data.get_height() * 1.01))
plt.show()
```

Figure 29: Query to capture the top 10 sources from which the collected set of tweets were posted

These sources were embedded within an HTML tag which had to be stripped using Beautiful Soup, a Python package meant for web-scraping. From the above code snippet, we could generate the following data frame and chart depicting the top 10 sources from which these live twitter streams were generated:

List of top 10 sources that these tweets have been tweeted from:

	count	source
0	1923	Twitter for iPhone
1	1454	Twitter for Android
2	828	Twitter Web Client
3	313	Twitter Lite
4	313	Twitter for iPad
5	75	Facebook
6	63	Instagram
7	61	TweetDeck
8	34	IFTTT
9	23	dlvr.it

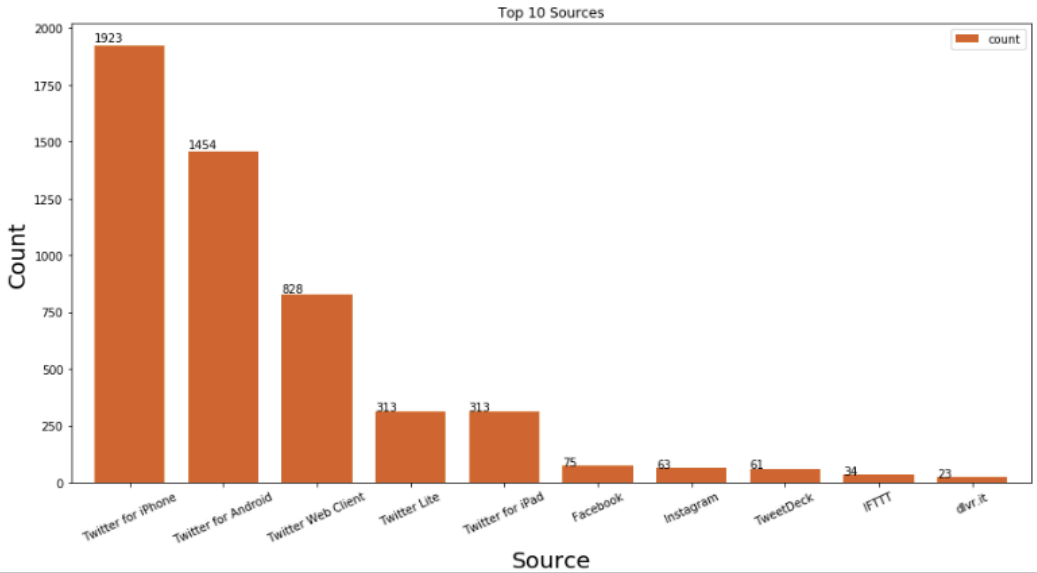

Figure 30: Data Frame and Chart depicting the top 10 sources from which the collected set of tweets were posted

From the above chart, we could see that the highest number of tweets for the collected set of 5274 tweets were from iPhone at 1923, followed by Android Phones at 1454.

D. Top 10 locations:

Using the 'location' parameter that is embedded within the 'user' parameter, we could find out the locations from which the tweets for the selected trending topics were generated from.

```
import pandas as pd
import matplotlib.pyplot as plt
from pymongo import MongoClient

client = MongoClient('localhost:27017')
db = client.DataEngineering
collection = client.DataEngineering.topic_name

print("List of top 10 locations that these tweets have been tweeted from:\n")
location_count = collection.aggregate([{'$group' : {'_id' : '$user.location', 'count' : {'$sum' : 1}}},
                                       {'$match' : {'count' : {'$gt' : 1}}},
                                       {'$sort' : {'count' : -1}},
                                       {'$limit' : 10},
                                       {'$project' : {'location' : '$_id', 'count' : 1, '_id' : 0}}])
location = pd.DataFrame.from_records(location_count)
location = location.iloc[1:]
print(location)

#Plotting the graph
location_plot=location.plot.bar(x='location', y='count', title ="Top 10 Locations", width=0.75, figsize=(15, 7.5),
                                rot=25, color='gold')
location_plot.set_xlabel("Location", fontsize=20)
location_plot.set_ylabel("Count", fontsize=20)

for data in location_plot.patches:
    location_plot.annotate(int(data.get_height()), (data.get_x() * 1.0, data.get_height() * 1.01))
plt.show()
```

Figure 31: Query to capture the top 10 locations from which the collected set of tweets were posted

From the above query, we could generate the following data frame and chart showing the locations from which these tweets were generated:

List of top 10 locations that these tweets have been tweeted from:

	count	location
1	168	United States
2	64	USA
3	62	Florida, USA
4	47	Texas, USA
5	33	Kingdom of Saudi Arabia
6	24	New Jersey, USA
7	22	California, USA
8	20	Houston, TX
9	20	Ohio, USA

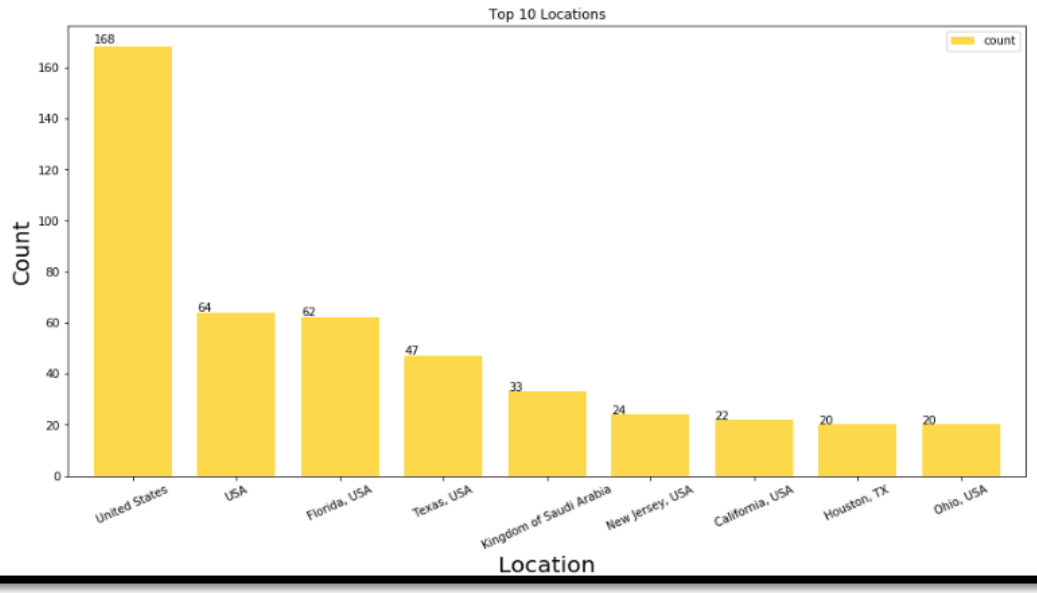

Figure 32: Data Frame and Chart depicting the top 10 locations from which the collected set of tweets were posted

As we can see from the above chart, most of them from the collected set of 5274 tweets were generated from the United States.

Kindly note, many users disable the location access feature to Twitter due to which the location with the highest number of tweets returns a value of 'None' as the location. This value has been omitted from the above chart.

E. User Popularity:

This analysis was performed to check the popularity levels of the users with the highest number of tweets. This was done by capturing the top 10 users with the highest number of tweets, and checking their corresponding following and followers count using the below query:

```
import pandas as pd
import matplotlib.pyplot as plt
from pymongo import MongoClient

client = MongoClient('localhost:27017')
db = client.DataEngineering
collection = client.DataEngineering.topic_name

print("List of top 10 Tweet generators and their popularity levels:\n")
pd.set_option('max_colwidth', 800)
resulta = collection.aggregate(
            [
                {"$group": { "_id": { "User Name": "$user.name"},
                             "Tweets":{"$last" : "$user.statuses_count"},
                             "Followers":{"$last" : "$user.followers_count"},
                             "Following":{"$last" : "$user.friends_count"}  }}
            ]
        )

resultb = pd.DataFrame.from_records(resulta)
resultb['_id'] = resultb['_id'].astype(str)
#splitting _id to A and B
resultb['A'], resultb['B'] = resultb['_id'].str.split(':', 1).str
#splitting B to C and D
resultb['C'], resultb['D'] = resultb['B'].str.split('}', 1).str
#renaming C to User Name
resultb = resultb.rename(columns = {'C' : 'User Name'})
#rearranging columns
final = resultb[["User Name", "Tweets","Followers", "Following"]]

final2 = final[["User Name", "Tweets","Followers", "Following"]]
#changing var type
final2['Tweets'] = final2['Tweets'].astype(int)
final2['Followers'] = final2['Followers'].astype(int)
final2['Following'] = final2['Following'].astype(int)
```

```
#sorting users in descending order by number of tweets
final2 = final2.sort_values('Following', ascending = [0])
print(final2.head(10))

#printing the top 10 users by No. of tweets
Top10Users = final2.head(10)

#Plotting the graph
ax=Top10Users.plot.bar(x='User Name', y=['Followers', 'Following'], title ="Top 10 Twitter Handles", width = 0.75,
                       figsize=(22, 10), rot=25)
ax.set_xlabel("Handle Name", fontsize=20)
ax.set_ylabel("Numbers", fontsize=20)

for p in ax.patches:
    ax.annotate(int(p.get_height()), (p.get_x() * 1.0, p.get_height() * 1.01))
plt.show()
```

Figure 33: Query to capture the user popularity

From the above query, we got the following results:

List of top 10 Tweet generators and their popularity levels:

User Name	Tweets	Followers	Following
'[illegible]Edward Shim[illegible]'	326884	259141	280037
'Marion Spekker'	1110431	263595	262747
'Jena C.'	634972	269173	157894
'[illegible]Jenny[illegible]'	121166	183969	156500
Bulldog (collectibulldogs)'	738418	183337	147347
'Stephanie Collins'	878129	138821	138336
'Trade Alerts 📈'	1595608	132782	131554
'Author Dennis Cardiff'	247562	113554	124732
'bud watcher'	134960	109883	120706
'Andrei'	743609	114139	107840

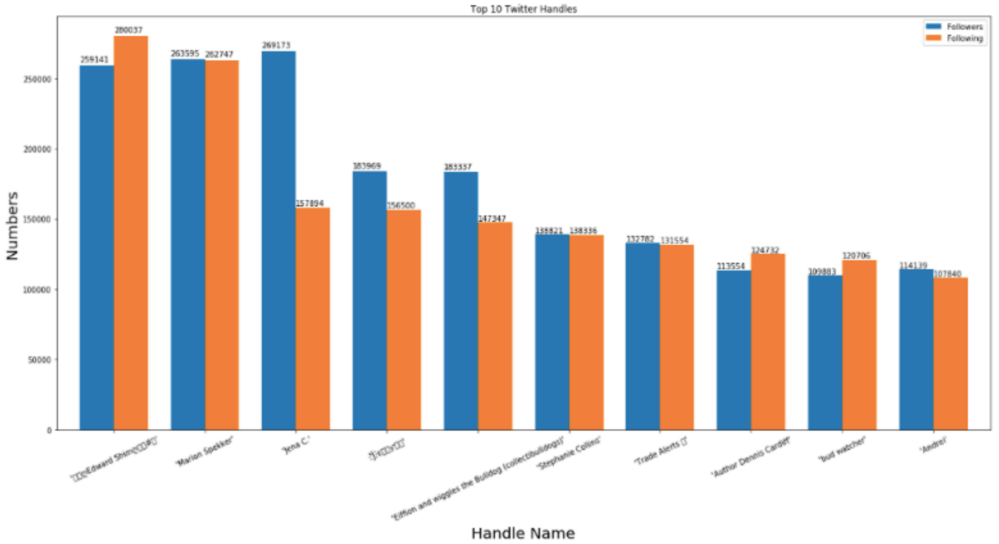

Figure 34: Data frame and chart depicting the popularity levels of users with the highest number of tweets

From the above graph we could see that:

1) Only user with user name 'Jena C' had a high Followers to Following ratio, is placed third in the number of tweets posted, and is a prominent tweeter in subjects related to the trending topics that were entered as keywords.
2) The rest may be regular social media aficionados who just tweet regularly.

XIII. Challenges Faced:

We faced 5 challenges while building this pipeline of which 3 were solved and 2 remain unsolved. More about it below:

A. Challenges that were solved:

1. Limiting Tweet Production:

Some topics have the capability to generate huge amount of tweets within a short space of time, and in order to avoid manually stopping the producer, we implemented a time function using Python's Time module, through which we extracted tweets only for a limited period. The code snippet is as follows:

```
def __init__(self, time_limit=600):      ## Setting the run time limit in seconds ##
    self.start_time = time.time()
    self.limit = time_limit
    super(StdOutListener, self).__init__()
```

Figure 35: Code snippet to insert a time limit for capturing tweets

In the code snippet above, we captured tweets for the created topic for only 600 seconds. Once the time limit had elapsed, the Kafka Producer stopped producing tweets to the Kafka Broker.

2. Stopping the Kafka Consumer:

Kafka Consumer has the ability to consume huge number of tweets in a short space of time. Considering that we were producing real-time tweets for only 10 minutes from the Kafka Producer, these were getting consumed and ingested by the Kafka Consumer within seconds. Though the Kafka Producer would have stopped producing tweets, the consumer would still be running in the background waiting for more tweets to be generated.

In order to tackle this, we implemented a consumer_timeout argument using which the consumer would consume tweets only for a limited period of time and would stop running. The code snippet is as follows:

```
consumer = KafkaConsumer(
    topic_name,        ## topic name in Kafka ##
     bootstrap_servers=['localhost:9092'],
     auto_offset_reset='earliest',
     enable_auto_commit=True,
     consumer_timeout_ms= 90 * 1000,    ## Setting the run time limit in milliseconds ##
     #group_id='my-group',
     value_deserializer=lambda x: loads(x.decode('utf-8')))
```

Figure 36: Code snippet to stop tweet consumption

In the above code snippet, we consumed tweets for only 90 seconds.

3. Finding distinct values for multiple parameters:

While performing analysis on the collected data, we wanted to perform various actions like finding tweets with highest number of retweets, users with highest number of tweets etc. The challenge with Twitter data is, every time a user tweets or every time a tweet gets retweeted, all new data related to the user or the tweet is added to the database, while old data still remains intact. Due to this, there are chances of one user or tweet appearing multiple times in the analysis.

After multiple attempts, we stumbled upon the '$last' function in the MongoDB documentation which helped us answer the 5th business question in the Analysis section above. Below is a code snippet of the same:

```
resulta = collection.aggregate(
            [
                {"$group": { "_id": { "User Name": "$user.name"},
                             "Tweets":{"$last" : "$user.statuses_count"},
                             "Followers":{"$last" : "$user.followers_count"},
                             "Following":{"$last" : "$user.friends_count"}  }}
            ]
        )
```

Figure 37: Code snippet to extract distinct values using multiple parameters

B. Challenges that remain unsolved:

1. Plotting a Map:

Each tweet structure has bounding box coordinates that comprises of minimum latitude, maximum latitude, minimum longitude and maximum longitude. These are embedded within the place parameter. Using these coordinates, we wanted to plot a map showing the location with the highest number of tweets. This was tried using Basemap and Matplotlib libraries. However, at the time of executing the code, we hit a roadblock:

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-185-526df15f58c1> in <module>
----> 1 from mpl_toolkits.basemap import Basemap
      2 import matplotlib.pyplot as plt
      3 # setup Lambert Conformal basemap.
      4 m = Basemap(width=12000000,height=9000000,projection='lcc',
      5             resolution='c',lat_1=45.,lat_2=55,lat_0=50,lon_0=-107.)

~/anaconda3/lib/python3.6/site-packages/mpl_toolkits/basemap/__init__.py in <module>
     15 from distutils.version import LooseVersion
     16 from matplotlib import __version__ as _matplotlib_version
---> 17 from matplotlib.cbook import is_scalar, dedent
     18 # check to make sure matplotlib is not too old.
     19 _matplotlib_version = LooseVersion(_matplotlib_version)

ImportError: cannot import name 'is_scalar'
```

Figure 38: Error message while trying to import Basemap

The above error says 'cannot import name 'is_scalar'' which may be due to an outdated version of matplotlib. Various methods like upgrading matplotlib, or uninstalling and reinstalling the same library were tried, however this error persisted.

2. Case Sensitive Hashtags Detection:

Twitter hashtags are case sensitive. For instance, topics with hashtags #Trump and #trump get bucketed differently, and it was a challenge to find the number of tweets using the hashtag 'Trump' while avoiding the case sensitivities.

XIV. Bibliography:

(n.d.). Retrieved from Python Package Index: https://pypi.org/

Code Beautify. (n.d.). Retrieved from Code Beautify: https://codebeautify.org/jsonviewer/f83352

Dorpe, S. V. (2018, Aug 13). *Kafka-Python explained in 10 lines of code*. Retrieved from Towards Data Science: https://towardsdatascience.com/kafka-python-explained-in-10-lines-of-code-800e3e07dad1

Freeman, J. (2014, Jun 27). *Do Twitter analysis the easy way with MongoDB*. Retrieved from InfoWorld: https://www.infoworld.com/article/2608083/application-development/do-twitter-analysis-the-easy-way-with-mongodb.html?page=3

LucidProgramming. (2018, Jan 15). *Twitter API with Python: Part 1 -- Streaming Live Tweets*. Retrieved from YouTube: https://www.youtube.com/watch?v=wlnx-7cm4Gg&t=38s

Petrone, J. (2014, May 23). *Is Zookeeper a must for Kafka?* . Retrieved from Stack Overflow: https://stackoverflow.com/questions/23751708/is-zookeeper-a-must-for-kafka

Rowe, W. (5. July 2017). *Working with Streaming Twitter Data Using Kafka*. Retrieved from bmc blogs: https://www.bmc.com/blogs/working-streaming-twitter-data-using-kafka

Senshin. (2014, Jan 18). *Python: Get Twitter Trends in tweepy, and parse JSON*. Retrieved from Stack Overflow: https://stackoverflow.com/questions/21203260/python-get-twitter-trends-in-tweepy-and-parse-json

Thakur, A. (28. Aug 2018). *Apache Kafka Installation on Mac using Homebrew*. Retrieved from Medium: https://medium.com/@Ankitthakur/apache-kafka-installation-on-mac-using-homebrew-a367cdefd273