CS5540- Principle of Big Data Management

Tweeflix

Project Phase 2 report

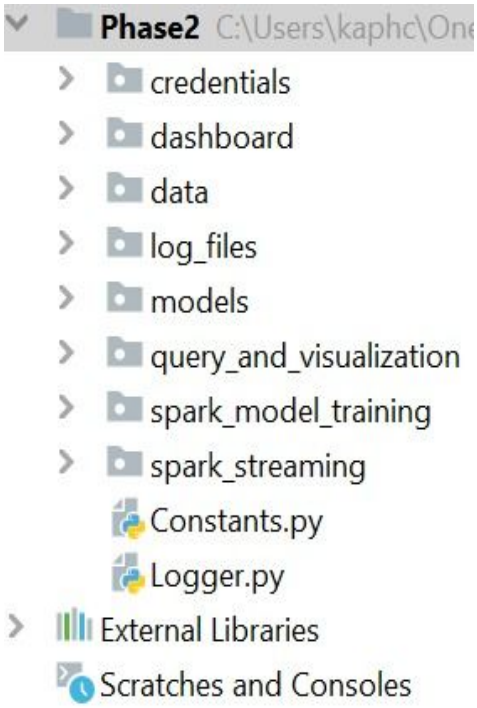
Team members

Kavin Kumar Arumugam -16262979

Shivani Sivasankar - 16231837

<u>Submitted contents</u>

1. **Phase 2 report**
2. **Log files**
3. **Training Data (Sentiment 140 +DSJ Vox articles)**
4. **PySpark models training python code (Sentiment analysis + Content analysis)**
5. **PySpark models (Sentiment analysis + Content analysis)**
6. **PySpark twitter streaming python code**
7. **Query and visualization Screenshots**



Motivation:

Promoting and protecting the health of communities is the goal of public health. Epidemiology is concerned with the dynamics of health conditions in populations. Rapid response through improved surveillance is important to combat emerging infectious diseases. The goal of this work is to use data-mining techniques such as tweet classification, sentiment analysis and content classification for public health surveillance.

Objectives:

In this project, we examine the use of information embedded in the Twitter stream to:

1. Track rapidly-evolving public sentiment with respect to commonly reported diseases
2. Track and measure actual disease activity

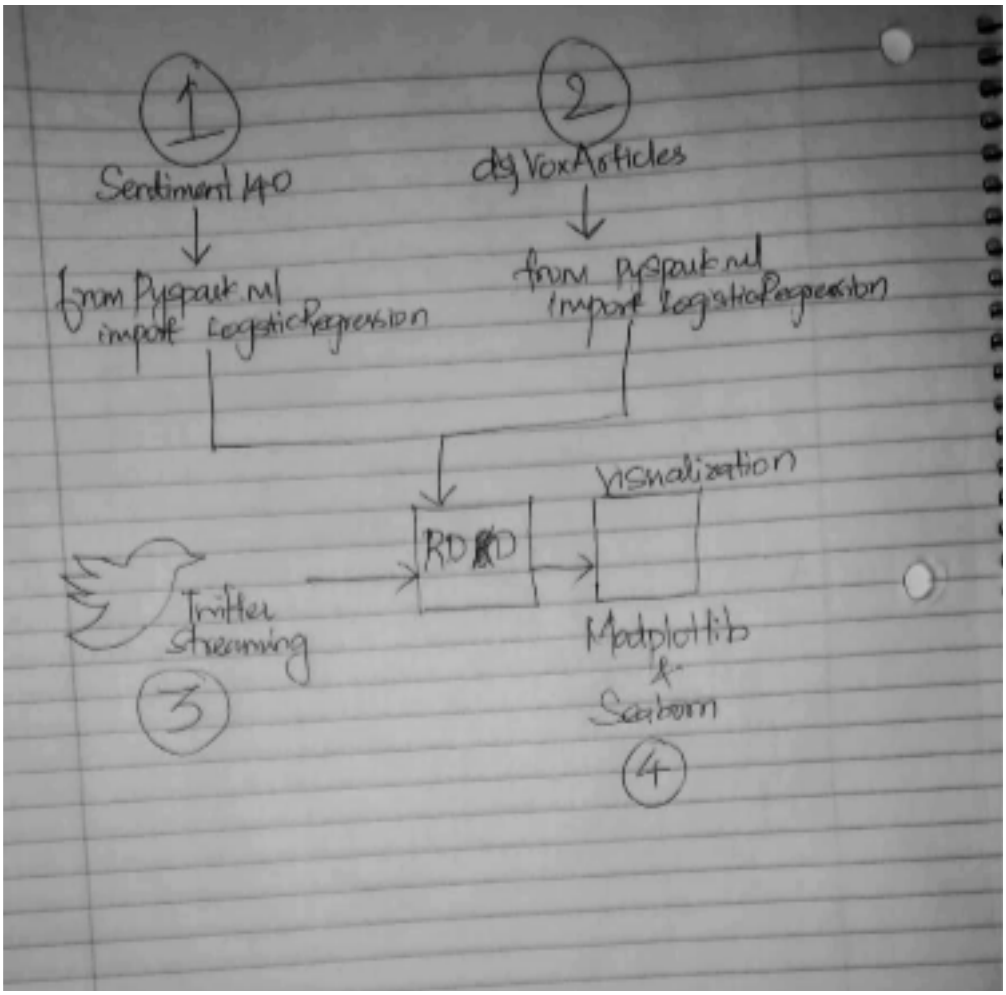
Features:

- Sentiment Analysis
- Content Analysis
- Streaming twitter data
- Query 1 - Tweets classified based on the type of content
- Query 2 -Top Trending Hashtags
- Query 3 - Sentiment analysis of tweets based on content category
- Query 4 - Proportion of streaming tweets classified by popular diseases
- Query 5: Sentiment analysis of tweets based on disease category
- Query 6- Positive sentiment tweets classified by country for HIV disease
- Query 7- Negative sentiment tweets classified by country for HIV disease
- Query 8 - Top trending Hashtags specific to health care
- Query 9 - Prevalence of disease classified by country

Technologies:

1. **PySpark.ml** -> For model training
2. **PySpark** -> SparkContext
3. **PySpark.Streaming** -> StreamingContext
4. **PySpark.sql** -> SQLContext
5. **Matplotlib**
6. **Seaborn**
7. **Jupyter-Notebook**

Design:



Dataset Description:

Tweets from “Sentiment140” originated from a Stanford research project data set included 1.6 million tweets which was loaded to train our model.

Source: https://www.kaggle.com/kazanova/sentiment140

1. **Target:** The polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)
2. **ids:** The id of tweet (2087)
3. **date:** The date of tweet (Sat May 16 23:58:44 UTC 2009)
4. **flag:** The query (lyx). If there is no query, then this value is NO_QUERY.
5. **user:** The user that tweeted (robotickilldozr)
6. **text:** The text of tweet (Lyx is cool)

Data Preprocessing:

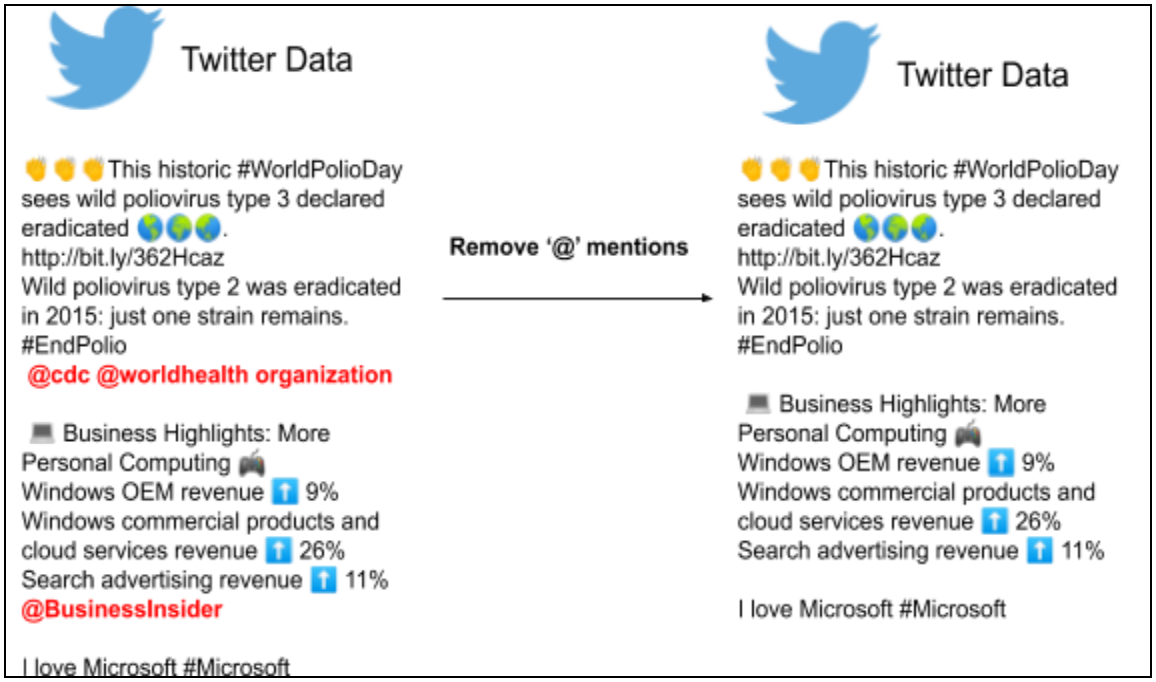
Step 1: Removing unnecessary columns

Target	ids	date	flag	user	text
0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	TheSpecialOne	1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D
0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	texting it... and might cry as a result School today also. Blah!

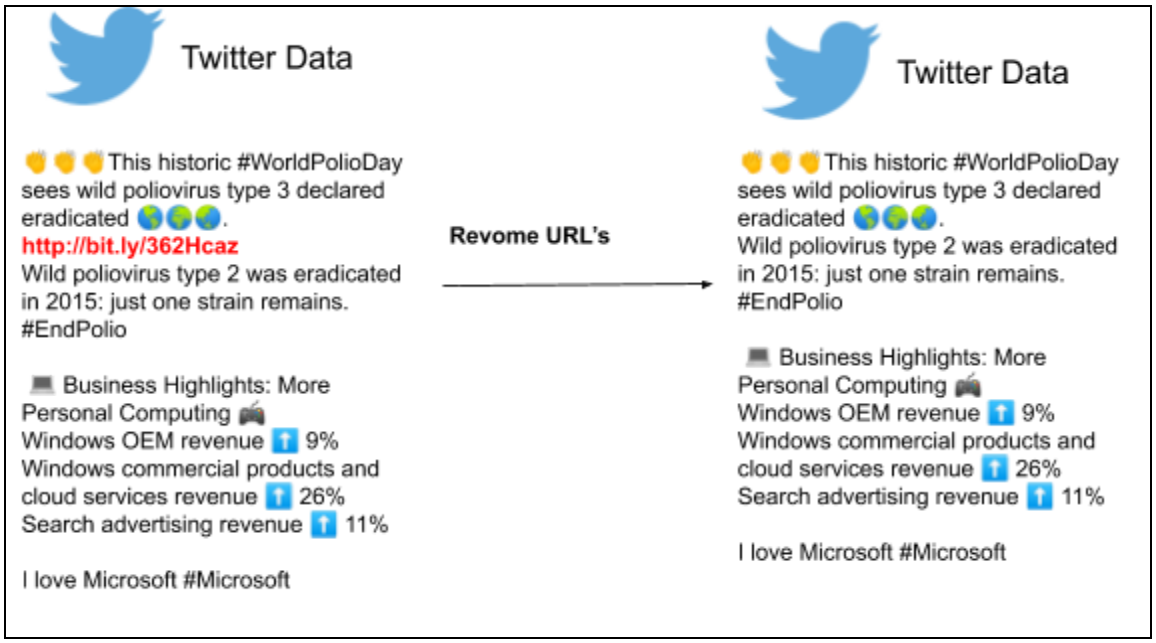
Modified to

Target	ids	text
0	1467810369	@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D
0	1467810672	is upset that he can't update his Facebook by texting it... and might cry as a result School today also. Blah!

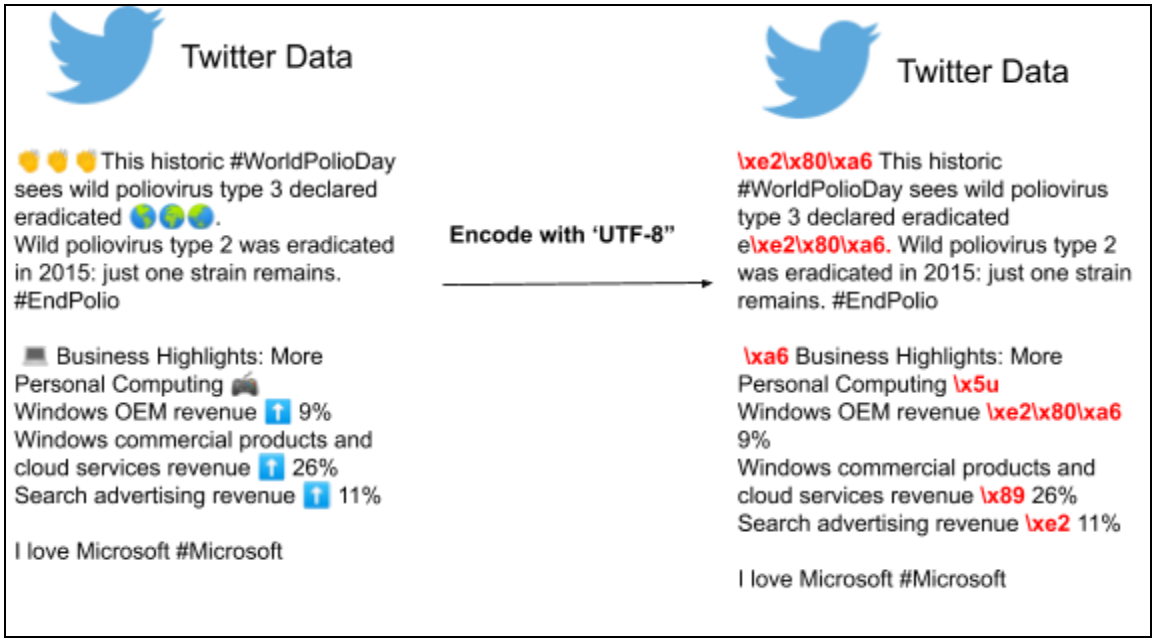
Step 2: Remove '@' mentions



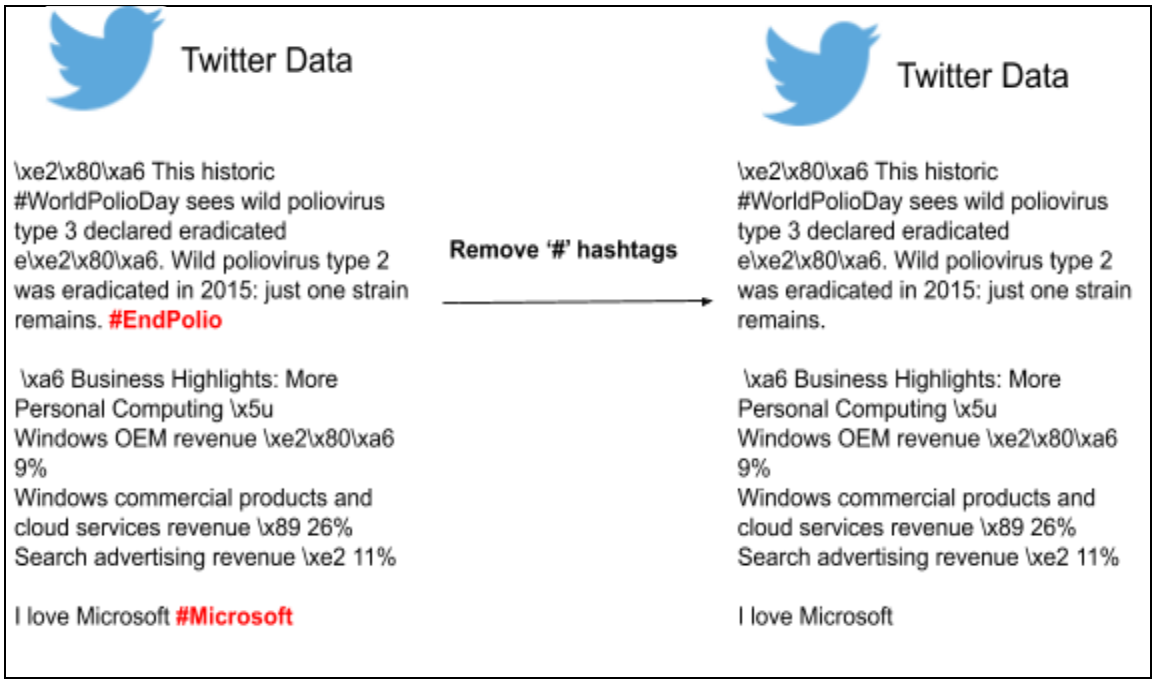
Step 3: Remove URL's



Step 4: Encode with 'UTF-8'



Step 5: Remove '#' hashtags



Sentiment Analysis:

Step 1: Tokenizing the column “text” which contains tweets to “words”
Tokenization is the process of breaking a sentence into individual terms or words. A simple Tokenizer class provides this functionality.

text	**words**
tayalejandro noooo it is for flu and cold apparently we catch colds from the nasty airplane air and not washing hands	['tayalejandro', 'noooo', 'it', 'is', 'for', 'flu', 'and', 'cold', 'apparently', 'we', 'catch', 'colds', 'from', 'the', 'nasty', 'airplane', 'air', 'and', 'not', 'washing', 'hands']
trying to clean the anti virus rogue software off of client computer s using malwarebytes wish me luck	['trying', 'to', 'clean', 'the', 'anti', 'virus', 'rogue', 'software', 'off', 'of', 'client', 'computer', 's', 'using', 'malwarebytes', 'wish', 'me', 'luck']

Step 2: Applying TF to the column “words” to “tf”
Term Frequency is the frequency of words in the column to the total number of words in the document.

words	**tf**
['tayalejandro', 'noooo', 'it', 'is', 'for', 'flu', 'and', 'cold', 'apparently', 'we', 'catch', 'colds', 'from', 'the', 'nasty', 'airplane', 'air', 'and', 'not', 'washing', 'hands']	(65536,[5163,8026,10019,15017,15766,15889,16332,16417,20639,24016,26141,27587,35633,38302,40636,51159,55380,55439,57498,64228],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])
['trying', 'to', 'clean', 'the', 'anti', 'virus', 'rogue', 'software', 'off', 'of', 'client', 'computer', 's', 'using', 'malwarebytes', 'wish', 'me', 'luck']	(65536,[1143,8436,9639,14771,16636,20719,21316,22357,24439,26623,28997,38302,53920,54083,57400,57938,62817,64881],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])

Step 3: Applying IDF to the column “tf” to “features”
Inverse Document Frequency is the log of the total number of documents divided by the number of words that contain the specific word

tf	**features**
(65536,[5163,8026,10019,15017,15766,15889,16332,16417,20639,24016,26141,27587,35633,38302,40636,51159,55380,55439,57498,64228],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0,1.0,1.0,1.0,1.	(65536,[5163,8026,10019,15017,15766,15889,16332,16417,20639,24016,26141,27587,35633,38302,40636,51159,55380,55439,57498,64228],[9.196396435776752,2.728072880503789,6.1947070980474646,8.987185978766465,6.8447217

0,1.0,1.0,1.0])	32586182,1.9792760581438174,2.071275270377485,3.3218544455865877,1.8076369357844004,5.2473321586682555,3.541440022343303,7.3983673535351,3.3727366838610515,1.316307797559611,6.3631830917205345,7.606006718313345,6.405501457434872,7.421550688990758,6.239457293846079,10.060608018606017])
(65536,[1143,8436,9639,14771,16636,20719,21316,22357,24439,26623,28997,38302,53920,54083,57400,57938,62817,64881],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])	(65536,[1143,8436,9639,14771,16636,20719,21316,22357,24439,26623,28997,38302,53920,54083,57400,57938,62817,64881],[5.734599096555953,1.249945800162861,2.238282890604683,8.278848632712545,11.320861658830541,4.042959434320993,6.228403865311913,4.863678967855373,2.326930982603277,7.495658661144479,2.212927107598086,1.316307797559611,9.18389627301252,4.4663961256532945,5.8514699595758985,7.253185343690602,3.783123226878734,5.223615632050939])

Step 4: Sentiment to label indexer
String column of labels to a column of label indices is encoded by the StringIndexer where the indices are ordered by label frequencies,most frequent label gets index 0.

sentiment	**label**
Positive	1
Negative	0

Step 5: Logistic regression
Logistic regression is used as a linear model for classification where it is used to determine the output value which can be in a binary form of 0 or 1.

Dataset Description:

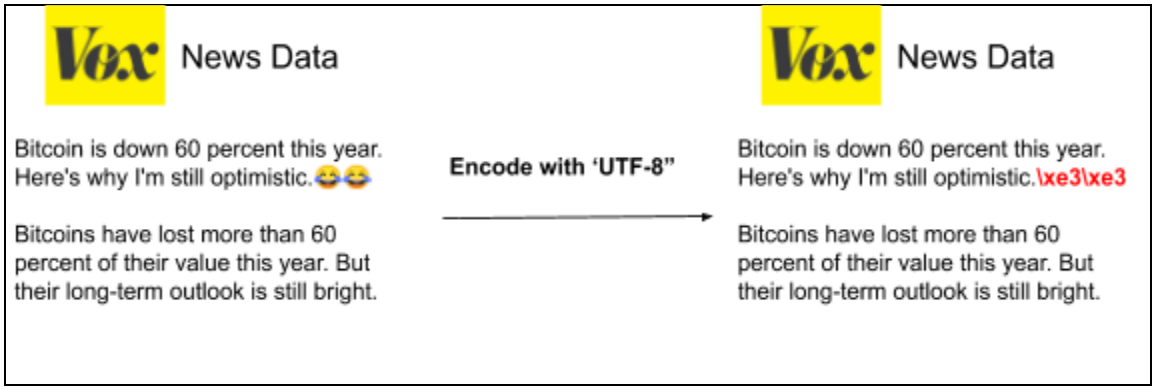
Articles from Vox.com published before March 21, 2017

Source:
https://data.world/elenadata/vox-articles/workspace/file?filename=dsjVoxArticles.tsv

1. **Title:** Article title
2. **Author:** Article author(s)
3. **Category:** Article category
4. **Published date:** Date of publication
5. **Updated on:** Date when the article was last updated
6. **SLUG:** Article URL
7. **Blurb:** Short Description
8. **Body:** Article body with html tags

Dataset Preprocessing:

Step 1: Encode with 'UTF-8'



Step 2: Selected only Category and Body columns

Content Classification:

Repeated steps 1 to 5 from Sentiment Analysis

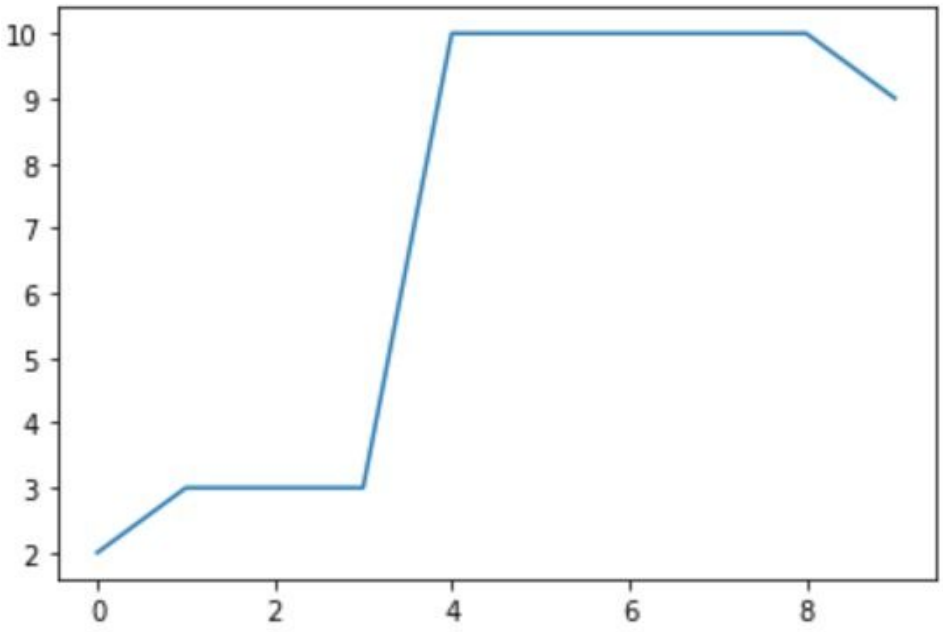
Twitter Streaming:

- Tweets from Twitter API are obtained using Python and passed on to the Spark Streaming instance
- The following libraries were imported : socket, sys, requests, requests_oauthlib, json
- The generated twitter access token, access key, consumer token and consumer key was used in OAuth for connecting to Twitter
- A new function called get_tweets was calls the Twitter API URL and returns the response for a stream of tweets.
- Then, we created a function from the above one and extracted the tweets' text from the whole tweets' JSON object. After that, it sends every tweet to Spark Streaming instance through a TCP connection.
- To make the app host socket connection with spark, we configured the IP to be localhost and the port 9009. Then we called the get_tweets method, to get the tweets from Twitter and passed it along with the socket connection to send_tweets_to_spark for sending the tweets to Spark.
- We then created an instance of Spark Context sc, and then the Streaming Context ssc from sc with a batch interval two seconds that will do the transformation on all streams received every two seconds.
- We defined a checkpoint in order to allow periodic RDD checkpointing and we defined our main DStream dataStream that will connect to the socket server to read the tweets from the port where each record in DStream will be a tweet

Query 1 - Number of tweets per iteration (every 2 seconds)

The number of tweets streamed every two second (per iteration) were visualized using line graph.

```
count = 0
series = []
time_series = []
iterator = 0

while count < 1:
    time.sleep(2)

    query_10_source = sqlContext.sql("SELECT count(*) as cnt FROM tweets")
    query_10_list = query_10_source.collect()

    for item in query_10_list:
        series.append(item.cnt)
        time_series.append(iterator)
        iterator = iterator + 1

    display.clear_output(wait=True)

    plt.plot(time_series, series)
    plt.ylabel('some numbers')
    plt.show()
```

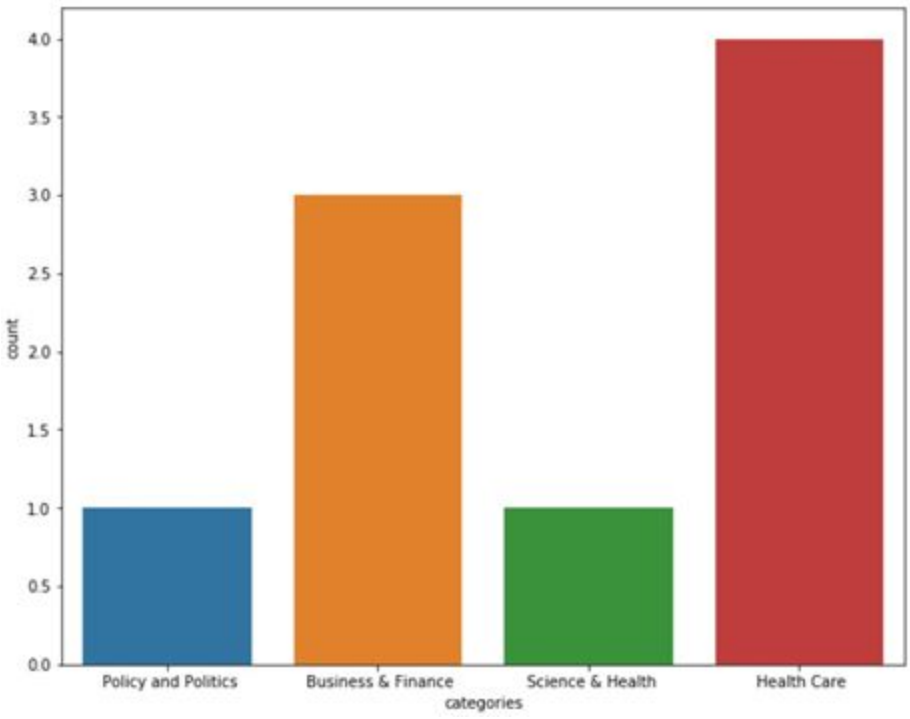


Query 2 - Tweets classified based on the type of content

Streaming tweets classified based on the type of category (Policy and politics, Criminal and Justice, Business and Finance, Science and Health, Healthcare) people are talking about right now

SELECT cast(count(*) as int) as count, _6 as categories FROM tweets WHERE _6 IS NOT NULL GROUP BY _6

```
count = 0
while count < 1:
    time.sleep(2)

    query_1_source = sqlContext.sql("SELECT cast(count(*) as int) as count, _6 as categories FROM tweets WHERE _6 IS NOT NULL
    query_1_df = query_1_source.toPandas()

    display.clear_output(wait=True)
    plt.figure( figsize = (10, 8))
    sns.barplot( x="categories", y="count", data=query_1_df)
    plt.show()
```



Query 3 -Top Trending Hashtags

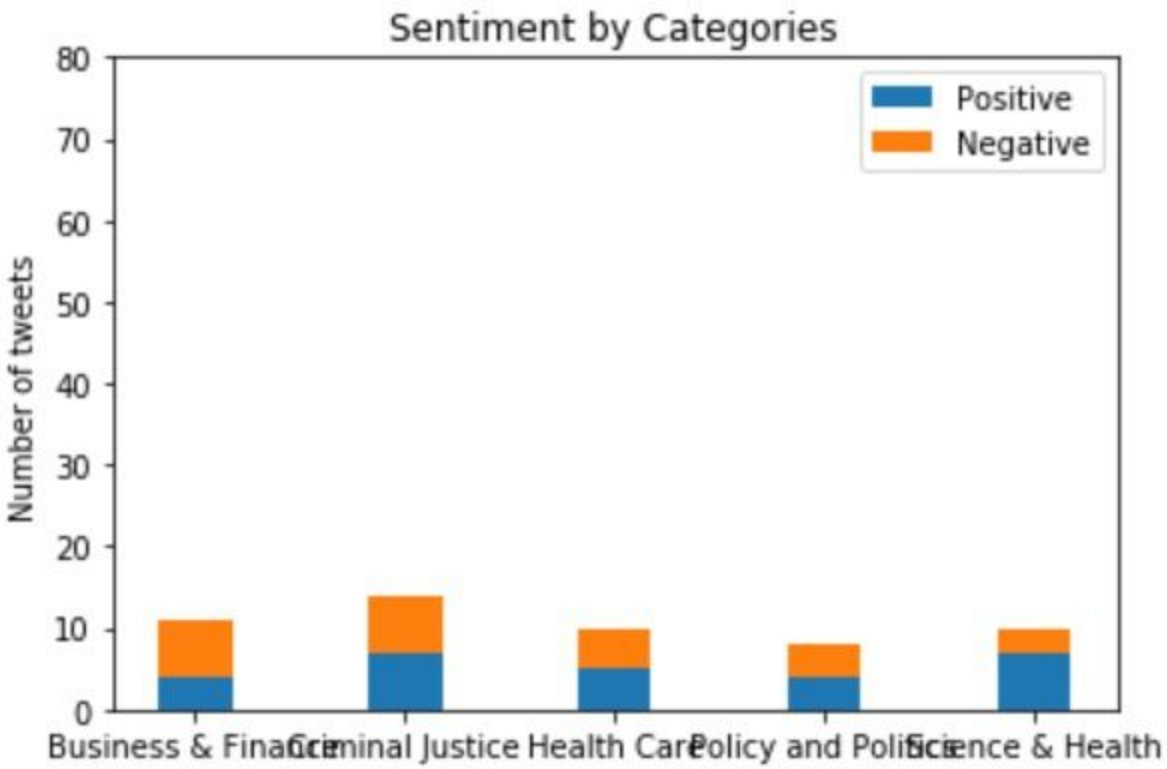
Top trending hashtags from the streaming twitter data represented as a word cloud.

```
count = 0
hashtags = " "

while count < 1:
    time.sleep(2)

    query_2_source = sqlContext.sql("SELECT entities.hashtag from tweets where entities.hashtag is not null")
    query_2_list = query_2_source.select('_7').collect()

    for item in query_2_list:
        hashtags = hashtags + item._7

    print(hashtags)
    wordcloud = WordCloud().generate(hashtags)

    display.clear_output(wait=True)
    plt.figure( figsize = (10, 8))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
```

Query 4 - Sentiment analysis of tweets based on content category

The number of tweets which are positive or negative in sentiment are classified based on the content category (Policy and politics, Criminal and Justice, Business and Finance, Science and Health, Healthcare) which is represented by a stacked bar chart.

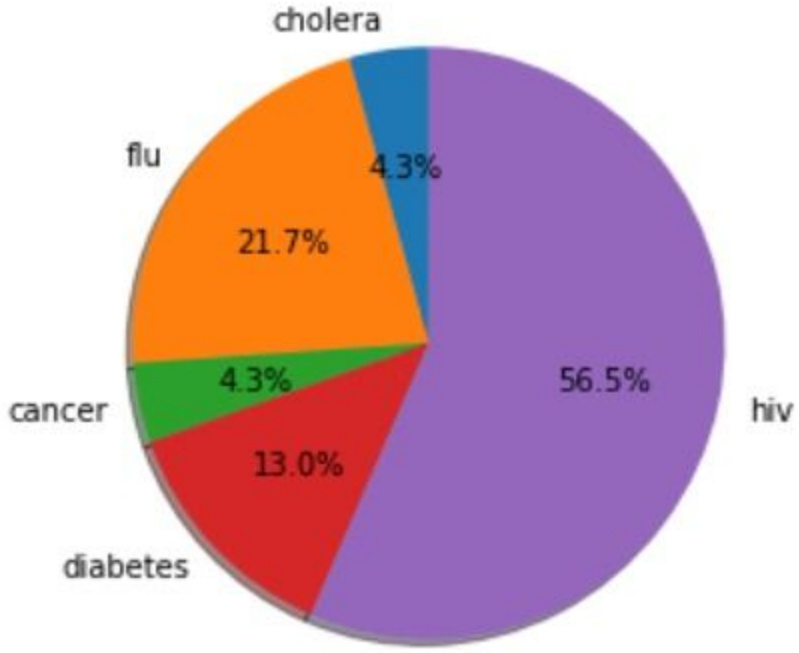
```
import numpy as np

count = 0
positive_count_values = [0, 0, 0, 0, 0]
negative_count_values = [0, 0, 0, 0, 0]

while count < 1:
    time.sleep(2)

    query_4_positive_source = sqlContext.sql("SELECT count(*) as cnt, _6 FROM tweets WHERE _5 LIKE '%1%' GROUP BY _6")
    query_4_positive_list = query_4_positive_source.collect()
    for item in query_4_positive_list:
        if item._6 == ' Business & Finance ':
            positive_count_values[0] = positive_count_values[0] + 1
        if item._6 == ' Criminal Justice ':
            positive_count_values[1] = positive_count_values[1] + 1
        if item._6 == ' Health Care ':
            positive_count_values[2] = positive_count_values[2] + 1
        if item._6 == ' Policy and Politics ':
            positive_count_values[3] = positive_count_values[3] + 1
        if item._6 == ' Science & Health ':
            positive_count_values[4] = positive_count_values[4] + 1
```



Query 5 - Proportion of streaming tweets classified by popular diseases

To identify the number of people currently talking about the popular diseases (Cholera, influenza, cancer, diabetes, HIV/AIDS) where its proportion is represented in the form of pie chart

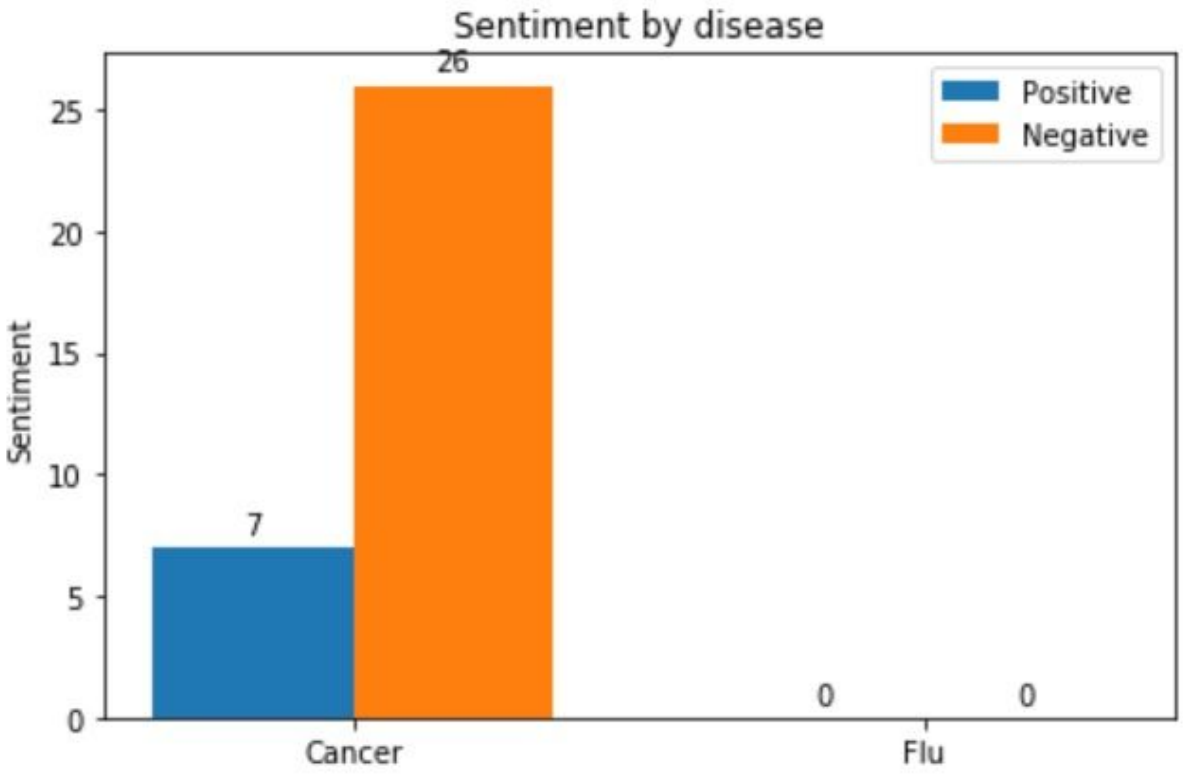
```
cholera_val = 1
flu_val = 1
cancer_val = 1
diabetes_val = 1
hiv_val = 1


while count < 1:

    cholera_source = sqlContext.sql("SELECT count(*) as cnt FROM tweets WHERE _2 LIKE '%cholera%'")
    cholera_list = cholera_source.select('cnt').collect()
    cholera_val = cholera_val + cholera_list[0].cnt

    flu_source = sqlContext.sql("SELECT count(*) as cnt FROM tweets WHERE _2 LIKE '%flu%' OR _2 LIKE 'influenza'")
    flu_list = flu_source.select('cnt').collect()
    flu_val = flu_val + flu_list[0].cnt

    cancer_source = sqlContext.sql("SELECT count(*) as cnt FROM tweets WHERE _2 LIKE '%cancer%'")
    cancer_list = cancer_source.select('cnt').collect()
    cancer_val = cancer_val + cancer_list[0].cnt

    diabetes_source = sqlContext.sql("SELECT count(*) as cnt FROM tweets WHERE _2 LIKE '%diabetes%'")
    diabetes_list = diabetes_source.select('cnt').collect()
    diabetes_val = diabetes_val + diabetes_list[0].cnt
```



Query 6: Sentiment analysis of tweets based on disease category

The number of tweets which are positive or negative in sentiment are classified based on the disease category (Cholera, influenza, cancer, diabetes, HIV/AIDS) which is represented by a grouped bar chart.

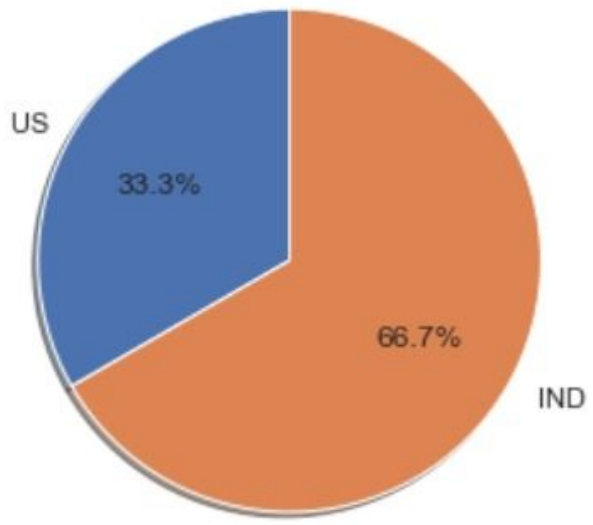
```
count = 0
positive = [0, 0]
negative = [0, 0]

while count < 1:

    cancer_positive_source = sqlContext.sql("SELECT count(*) as cnt FROM tweets WHERE _2 LIKE '%cancer%'")
    cancer_positive_list = cancer_positive_source.select('cnt').collect()
    positive[0] = positive[0] + cancer_positive_list[0].cnt

    flu_positive_source = sqlContext.sql("SELECT count(*) as cnt FROM tweets WHERE _2 LIKE '%flu%' OR _2 LIKE 'influenza'")
    flu_positive_list = flu_positive_source.select('cnt').collect()
    positive[1] = positive[1] + flu_positive_list[0].cnt

    cancer_negative_source = sqlContext.sql("SELECT count(*) as cnt FROM tweets WHERE _2 LIKE '%cancer%'")
    cancer_negative_list = cancer_negative_source.select('cnt').collect()
    negative[0] = negative[0] + cancer_negative_list[0].cnt

    flu_negative_source = sqlContext.sql("SELECT count(*) as cnt FROM tweets WHERE _2 LIKE '%flu%' OR _2 LIKE 'influenza'")
    flu_negative_list = flu_negative_source.select('cnt').collect()
    negative[1] = negative[1] + flu_negative_list[0].cnt

    labels = ['Cancer', 'Flu']
```



Query 7- Positive sentiment tweets classified by country for HIV disease

Positive sentiments about the tweets which mentions HIV was counted and its proportion was grouped by country and represented as a pie chart.

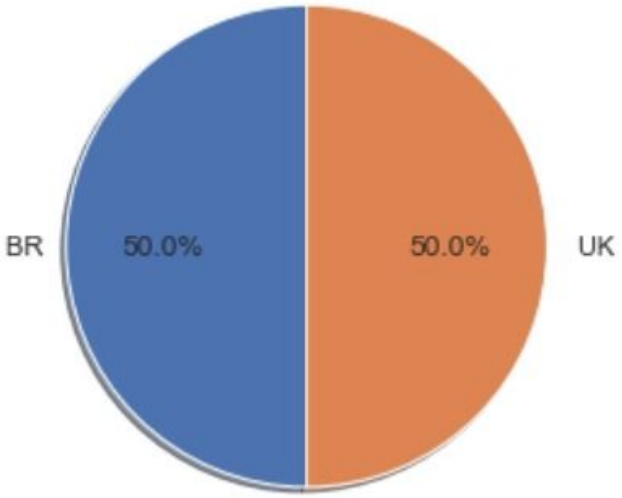
```
count = 0

while count < 1:
    time.sleep(2)

    query_6_source = sqlContext.sql("SELECT pos.pos_cnt as pos_cnt, neg.neg_cnt as neg_cnt, neg.country FROM (SELECT count(*)
    query_6_list = query_6_source.collect()

    pos_labels = []
    pos_count = []
    total_pos_count = 0
    for item in query_6_list:
        pos_labels.append(item.country)
        pos_count.append(item.pos_cnt)
        total_pos_count = total_pos_count + item.pos_cnt

    for x in pos_count:
        x = x/total_pos_count

    display.clear_output(wait=True)
    plt.figure( figsize = (10, 8))
    fig2, ax2 = plt.subplots()
    ax2.pie(pos_count, labels=pos_labels, autopct='%1.1f%%', shadow=True, startangle=90)
    ax2.axis('equal')
    plt.show()
```



Query 8- Negative sentiment tweets classified by country for HIV disease

Negative sentiments about the tweets which mentions HIV was counted and its proportion was grouped by country and represented as a pie chart.

```
count = 0

while count < 1:
    time.sleep(2)

    query_6_source = sqlContext.sql("SELECT pos.pos_cnt as pos_cnt, neg.neg_cnt as neg_cnt, neg.country FROM (SELECT count(*)
    query_6_list = query_6_source.collect()

    neg_labels = []
    neg_count = []
    total_neg_count = 0

    for item in query_6_list:
        neg_labels.append(item.country)
        neg_count.append(item.neg_cnt)
        total_neg_count = total_neg_count + item.neg_cnt

    for x in neg_count:
        x = x/total_neg_count

    display.clear_output(wait=True)
    plt.figure( figsize = (10, 8))
    fig1, ax1 = plt.subplots()
    ax1.pie(neg_count, labels=neg_labels, autopct='%1.1f%%', shadow=True, startangle=90)
    ax1.axis('equal')
    plt.show()
```



Query 9 - Top trending Hashtags specific to health care

Analysis of the most frequently mentioned Hashtags in the health care content category is represented by a word cloud

```
count = 0
hashtags = ""

while count < 1:
    time.sleep(2)

    query_2_source = sqlContext.sql("SELECT _7 from tweets where _6 LIKE '%Health%' OR _6 LIKE '%Health%'")
    query_2_list = query_2_source.select('_7').collect()

    for item in query_2_list:
        hashtags = hashtags + item._7

    print(hashtags)
    wordcloud = WordCloud().generate(hashtags)

    display.clear_output(wait=True)
    plt.figure( figsize = (10, 8))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
```

Query 10 - Prevalence of disease classified by country

The number of streaming tweets which mention the five most common diseases (Cholera, influenza, cancer, diabetes, HIV/AIDS) were classified based on the location of the tweets.

```
count = 0

while count < 1:
    time.sleep(10)

    query_9_source = sqlContext.sql("SELECT count(*) as cnt, _4, _9 FROM tweets GROUP BY _4, _9")
    query_9_df = query_9_source.toPandas()
    print(query_9_df)
```

	cnt	_4	_9
0	1	AUS	cholera
1	1	BR	flu
2	3	AUS	diabetes
3	2	US	flu
4	1	BR	cancer
5	1	IND	flu
6	1	IND	cholera