# Sentiment Analysis on Twitter feeds

**Pragya Mishra, A20406346**

**Aim:**

Analysing twitter data using RHadoop

## 1.Introduction:

Twitter is an online social networking and microblogging service with about 500 million tweets being sent per day. These tweets can be used by companies to market products, understand Brand equity, understand customer's product reviews, etc. Techniques like Text mining and Sentiment analysis can be used to on this vast amount of data using R. Hadoop, on the other hand, can provide the much needed distributed storage (HDFS) and distributed computing (MapReduce) ecosystem of handling this vast data. Few applications of this project are:

- Competitive Intelligence: understand competitors strengths and weaknesses, facilitate strategic decision making
- Community leveraging: understand community interactions which in turn helps in prioritizing customer's preferences

## 2.Technology used:

### 2.1 RHadoop

Storing and processing large amounts of data is a challenging job for existing statistical computer applications such as R and SAS. These tools are primarily known for their analytics and statistics functionality to interpret, analyse and visualise data. On the other hand, for data storage and processing needs, enterprises depend on sequel databases. And now with the growth of big data, hadoop is the go-to environment. These databases act as central data repositories that contain all information which is gathered from various business units in an enterprise. Though data management tool like hadoop can effectively store and process big data, they still lack the analytical and statistical functionality which is needed to perform various data analytics operations. Obviously, after getting to know the individual strengths of both R and Hadoop, we can say that a better solution for big data analytics would be a system integrating this. A key point to note here is that both R and Hadoop have their own working environments and with RHadoop system, we will be creating an interface between these two . Additionally, we can write MapReduce programs in R syntax and also would be able to execute HDFS commands from R console. In summary, RHadoop can act as a reliable solution for implementing Big Data Analytics.

Hadoop has multiple language support such as pig, java and hive and using R Hadoop is not a complete big data analytics solution. So depending on the problem being solved, we would pick the right tool in order to get the solution in the most optimal manner.

RHadoop is a collection of five packages: rhdfs, rmr2, rhbase, plyrmr and ravro

1. ravro – package used to connect to the Avro files from the HDFS.
2. rhdfs – It mainly provides connectivity to a distributed Hadoop file system (HDFS).It enables us to view, read and edit data stored in HDFS.
3. rhbase – package using to connect to the Hbase and NOSQL distributed database.
4. rmr2 – package providing the set of functions to write a R code that can be transformed into the MapReduce task.
5. plyrmr– package that enables to execute data manipulation functions contained in package dplyr and reshape2, but on the large sets of data stored in Hadoop clusters. Similar to rmr2, it relies on translation of the R code into the MapReduce paradigm

Among so many existing analytical tools, why R?

1. R is one of the most comprehensive statistical analysis package available. It incorporates all the standard tests and model. And most often, new technologies and ideas appear first in this tool.
2. R is free and open source under the GNU General Public License.
3. R packages are powerful and are widely used for statistical and data analysis. Number of packages = 4000 and is constantly increasing due to active community involvement.
4. Can be used for parallel computing across a number of cores and clusters.

### 2.2 Python:

We use Python is this project primarily for data extraction and pre-processing. Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy that emphasizes code readability (using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

For this project we use Python 3.6.3 with Spyder 3.2.4. Spyder (formerly Pydee[3]) is an open source cross-platform integrated development environment (IDE) for scientific programming in the Python language.

## 3.Work Flow:

1. **Data Extraction:** This includes steps getting Twitter API keys and then using Python libraries and Twitter API package to connect to Twitter Rest API and downloading the data. Data extraction can also be done by web scraping or using social media listening tools
2. **Data Storage and Exploration:** This step includes storage of data and performing some exploratory techniques on it to understand what the data is all about and if it is primarily pivoting around a single topic. The general idea received in this step about data helps in the further steps like data pre-processing and analysis.
3. **Data Pre-processing:** Text data usually does not have a predefined schema and isn't available in a specified format. It is usually in semi-structured or fully unstructured format. Before applying any analytics, data storage and cleaning methods need to be applied that would provide a structure to the data. These can be processed on Hadoop cluster by using custom scripts written through HiveQL or Pig languages.
4. **Data Analysis:** This includes primarily Textual Mining/Analysis. Here,we would generally be identifying patterns in cleaned data, implement sentiment analysis and generate insights.
5. **Data post-processing:** This part would include steps like finding the most common words, performing word cloud visualisations, creating term document matrices, looking at word trends overtime, identifying word frequency correlation, computing word co-occurrence metrics

### 3.1 Data pre-processing:

This step includes the following:

1. Converting the entire text data to lower case to avoid word duplication
2. Word extraction or tokenization: Splitting of text data into stream of words. This process makes use of single white spaces commonly and the tokenised data is used for further processing and analysis.
3. Removing infrequent or sparse words: These infrequent words occur in less than 1% of the entire text document. This helps in reducing data for fast processing. And often, the cut-off decision of the minimum acceptable number of text document is subjective.
4. Stop words removal: Deals with removing most frequently occurring words as per English vocabulary which technically will not contribute much to the final analysis. Like removing "of", "to", "the". Typically 400-500 such words exist in English language. This process would improve efficiency and effectiveness of the final analysis result.
5. Stemming process: Used to identify the root/stem of a word. It is a slightly advanced technique for improving effectiveness by reducing the data size to manageable levels.
6. Performing descriptive statistics on text data like frequency counts and computing other aggregations.

### 3.2 Data Analysis:

This step includes Textual Mining/Analysis which, in general, include:

1. Text Categorisation: Cataloguing texts into categories
2. Text Clustering: Clustering groups of automatically retrieved text into a list of meaningful categories.
3. Concept/entity extraction: Locating and classifying elements in text into predefined categories such as persons, organisations, locations, monetary values.
4. Granular taxonomies: Enabling organisation or classification of information as a set of objects and displayed as a taxonomy.
5. Sentiment Analysis: Identifying and extracting subjective information in text data

6. Document summarisation: Creating a shortened version of a text containing the most important elements.
7. Entity relation modelling: Automated learning of relationships between data types.
8. Machine learning: Building automated document classifications systems using methods such as neural networks, probabilistic models, trees.

## 3.2.1 Sentiment Analysis

Also known as Opinion Mining, it retrieves and ranks the relevant content on the basis of algorithms or a set of defined rules. It is one of the most important components of Text mining. Sentiment can be classified as positive, negative or neutral. The kind of sentences that are analysed with this fall under facts and opinions categories.
Eg:
Fact: " Product X is better than product Y"
Opinion: "I don't like X. I feel Y is better in terms of durability."

Implementing Sentiment Analysis: The process of sentiment analysis involves classification of a given text on the basis of the following parameters:
1. Polarity: positive, negative or neutral
2. Emotional states in the text: sad, angry or happy
3. Scaling system or numeric values present in the text
4. Subjectivity: relating to the context
5. Classifying features based on key entities such as durability of the furniture, screen size of the cell phone, lens quality of a camera etc.

***Example 1:***
"I bought an ***iPhone*** few days back. It is really ***nice***. The touch screen and voice quality are really ***cool***. It is ***better*** than my old ***Blackberry*** phone which was so ***hard to type*** with tiny keys. However ***iPhone*** is a bit ***expensive***"

The analysis on the above line can be done on varying levels like:
- Document-level sentiment analysis: Is this review positive or negative?
- Sentence-level sentiment analysis: Is each sentence positive or negative?
- Entity-level sentiment analysis: Is the review of iPhone positive or negative?

Focus of this project would be Document-level sentiment analysis.

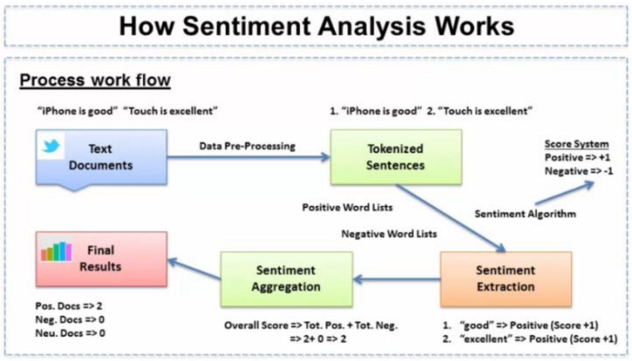


***Figure 1***

We will discuss a simple sentiment analysis algorithm. This algorithm assigns sentiment score by simply counting the number of occurrences of "positive" or "negative" words in any sentence. Referring to example 1:

Positive words: nice, cool, better
Negative words: hard, expensive

These above words are actually mapped to the positive and negative word list which are taken from the standard English dictionary. After which we assign +1 for every positive word and -1 for every negative word. The sentiment score for the above text is +1 (3-2). Therefore the total score is value 1 and sentiment polarity is positive.
At an overall level, the sentiment score of these individual text documents needs to summed up and this will be viewed as the overall sentiment expressed by the entire text data.

There are multiple challenges with the above described algorithm while dealing with:

1. Negations which are of two kinds, Direct –"I don't like my new phone", and Ambiguous –"This phone is cool, but lacks many features".
2. Co-references – "I bought a new iPhone. It is cool and awesome". What does "it" refers to?
3. Slang and writing error – Acronyms like lol, idk etc, punctuations use like "im" for "I am", "w'll" for "we will".

Many solutions can be implemented to overcome these challenges which are often subjective and dependent on the context of the text.

# Sentiment Analysis on Twitter feeds

**Pragya Mishra, A20406346**

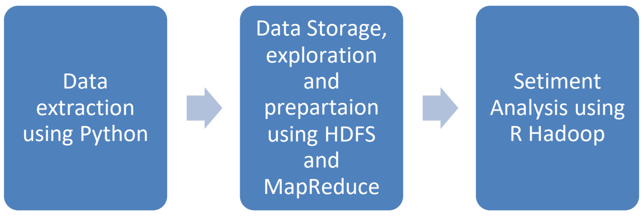

Data extraction using Python
Data Storage, exploration and prepartaion using HDFS and MapReduce
Setiment Analysis using R Hadoop

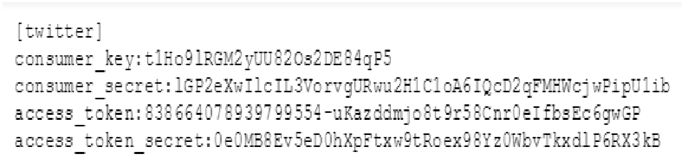
*Figure 2: High Level Work Flow*

## 4. Project Implementation:

### 4.1 Data Extraction:

- Create a Twitter Account
- Create "New App" on https://apps/twitter.com/
- Create a Twitter application
- Generate API keys, API secret keys, Access token, Access token secret.
- Install Twitter API on python.

Establish connection with twitter using the above keys and retrieve and store tweets from twitter into a csv file, named data5.csv. Code attached in Appendix A. **While extracting twitter feeds, the aim was to understand the overall sentiment of the twitter feeds on a particular date. We chose 29th April 2016.**



```
[twitter]
consumer_key:t1Ho91RGM2yUU82Os2DE84qP5
consumer_secret:lGP2eXwIlcIL3VorvgURwu2H1C1oA6IQcD2qFMHWcjwPipU1ib
access_token:838664078939799554-uKazddmjo8t9r58Cnr0eIfbsEc6gwGP
access_token_secret:0e0MB8Ev5eD0hXpFtxw9tRoex98Yz0WbvTkxdlP6RX3kB
```

***Figure 3: twitter.config file***

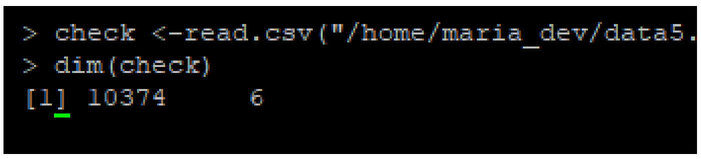
### 4.2. Integrating R and Hadoop:

- Setup Hortonworks sandbox with the name "pragyaa"
- Use Gitbash/Putty to connect to it using Secure Shell (SSH)
- Logged into maria_dev and then root for setting up R
- As root user, installed R using "yum install R" command
- Downloaded and installed required dependency packages in R using commands like "sudo Rscript -e 'install.packages(c("rJava", "Rcpp", "RJSONIO", "bitops", "digest", "functional", "stringr", "plyr", "reshape2", "caTools"), repos = "http://cran.r-project.org/")' and sudo Rscript -e 'install.packages(c("dplyr", "R.methodsS3", "Hmisc", "memoise", "lazyeval", "rjson"), repos = "http://cran.r-project.org/")'
- Downloaded and installed the previously mentioned hadoop libraries using command :
  sudo wget https://github.com/RevolutionAnalytics/rmr2/releases/download/3.3.1/rmr2_3.3.1.tar.gz

  sudo R CMD INSTALL rmr2_3.3.1.tar.gz
- Used same commands for downloading and installing rest of the RHadoop packages mentioned in section 2.1.
- Start R by typing "R" .
- Set up Hadoop environment variables using commands like:
  cmd <- system("which hadoop", intern=TRUE)
  Sys.setenv(HADOOP_CMD=cmd)
  stream <- system("find /usr -name hadoop-streaming*jar", intern=TRUE)
  Sys.setenv(HADOOP_STREAMING=stream[1])

### 4.3. Data storage and Exploration:

- Transferred data.csv file to azure sandbox.
- Started R and initialised hdfs using hdfs.init()
- Created a new directory Hadoop with a name "Project" using code:
  >setwd("/home/maria_dev")
  >hdfs.mkdir("Project")
- Transferred data5.csv into HDFS into "Project" folder using command:
  > file <- dir(getwd(), pattern = "data5.csv", full.names = TRUE)
  >hdfs.put(file, "Project")
- Loaded the content of data.csv into a dataframe called "check" by using command:
  >check <- read.csv("/home/maria_dev/data.csv",header =FALSE, sep = "," col.names=names(col.classes), colClasses=col.classes)

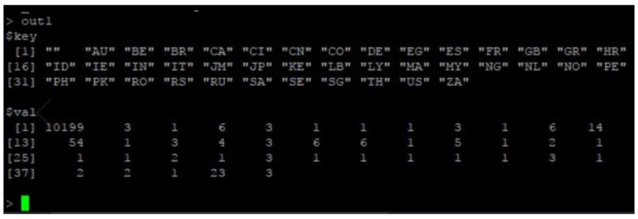


```
> check <-read.csv("/home/maria_dev/data5.
> dim(check)
[1] 10374     6
```

- Found number of tweets by location by defining mapper and reducer functions and then executing a mapreduce job. Command:

  >map1 <-function(k,v)
  {
      Keyval(v$location,1)

```
}
>reduce1 <-function(k,vv)
{
     Keyval(k,sum(vv))
}
>out1 <-from.dfs(mapreduce(input = to.dfs(check),
map=map1,reduce=reduce1))
```



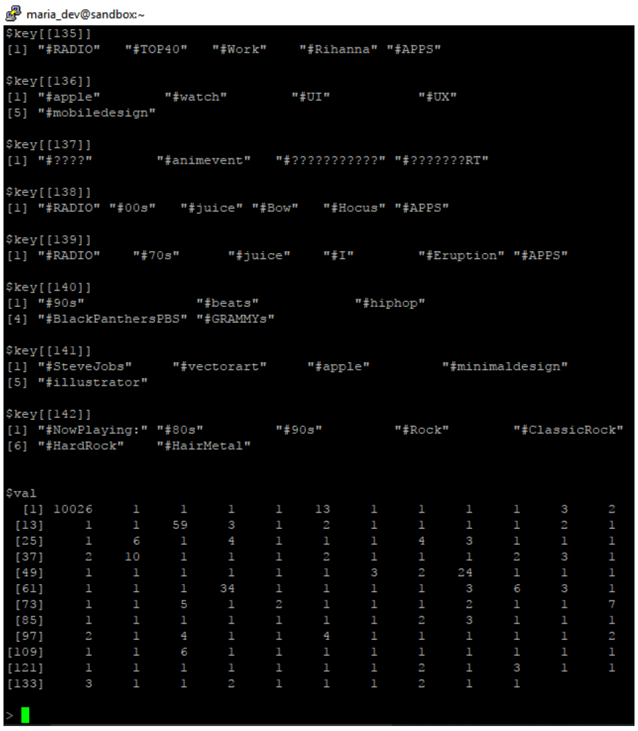
```
> out1
$key
 [1] ""   "AU" "BE" "BR" "CA" "CI" "CN" "CO" "DE" "EG" "ES" "FR" "GB" "GR" "HR"
[16] "ID" "IE" "IN" "IT" "JM" "JP" "KE" "LB" "LY" "MA" "MY" "NG" "NL" "NO" "PE"
[31] "PH" "PK" "RO" "RS" "RU" "SA" "SE" "SG" "TH" "US" "ZA"

$val
 [1] 10199     3     1     6     3     1     1     1     3     1     6    14
[13]    54     1     3     4     3     6     6     1     5     1     2     1
[25]     1     1     2     1     3     1     1     1     1     1     3     1
[37]     2     2     1    23     3

> 
```

- Found various hashtags used in the tweets and created a wordcloud vizualisation of the same to understand the most trending topic. Command:

```
>Library(stringr)
>map2 <-function(k,v)
{
x <- str_extract_all(v$text, perl("(?<=^|\\s)#\\S+"))
keyval(x ,1)
}
reduce2 <-function(k,vv)
{
     Keyval(k,sum(vv))
}
out2  <-from.dfs(mapreduce(input  =  to.dfs(check),
map=map2,reduce=reduce2))
```



maria_dev@sandbox:~

```
$key[[135]]
[1] "#RADIO"   "#TOP40"   "#Work"    "#Rihanna" "#APPS"

$key[[136]]
[1] "#apple"        "#watch"        "#UI"           "#UX"
[5] "#mobiledesign"

$key[[137]]
[1] "#????"        "#animevent"   "#???????????" "#???????RT"

$key[[138]]
[1] "#RADIO" "#00s"   "#juice" "#Bow"   "#Hocus" "#APPS"

$key[[139]]
[1] "#RADIO"    "#70s"      "#juice"    "#I"        "#Eruption" "#APPS"

$key[[140]]
[1] "#90s"             "#beats"           "#hiphop"
[4] "#BlackPanthersPBS" "#GRAMMYs"

$key[[141]]
[1] "#SteveJobs"     "#vectorart"     "#apple"         "#minimaldesign"
[5] "#illustrator"

$key[[142]]
[1] "#NowPlaying:" "#80s"         "#90s"         "#Rock"        "#ClassicRock"
[6] "#HardRock"    "#HairMetal"


$val
  [1] 10026     1     1     1     1    13     1     1     1     1     3     2
 [13]     1     1    59     3     1     2     1     1     1     1     2     1
 [25]     1     6     1     4     1     1     1     4     3     1     1     1
 [37]     2    10     1     1     1     2     1     1     1     2     3     1
 [49]     1     1     1     1     1     1     3     2    24     1     1     1
 [61]     1     1     1    34     1     1     1     1     3     6     3     1
 [73]     1     1     5     1     2     1     1     1     2     1     1     7
 [85]     1     1     1     1     1     1     1     2     3     1     1     1
 [97]     2     1     4     1     1     4     1     1     1     1     1     2
[109]     1     1     6     1     1     1     1     1     1     1     1     1
[121]     1     1     1     1     1     1     1     2     1     3     1     1
[133]     3     1     1     2     1     1     1     2     1     1

> 
```

- For creating a wordcloud visualization of the above results, downloaded and installed RColorBrewer and wordcloud libraries using command:

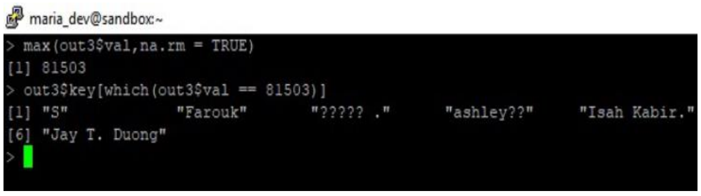
```
>system("wget --no-check-certificate https://cran.r-project.org/src/contrib/RColorBrewer_1.1-2.tar.gz")
>install.packages("RColorBrewer_1.1-2.tar.gz ", repos = NULL, type="source")
>system("wget --no-check-certificate https://cran.r-project.org/src/contrib/wordcloud_2.5.tar.gz ")
>install.packages("wordcloud_2.5.tar.gz ", repos = NULL, type="source")
```

- Created the wordcount visualization using:

```
>library(“RColorBrewer”)
>Library(“wordcloud”)
>pal2 <-brewer.pal(8,”Dark2”)
>wordcloud(out2$key,out2$val, random.order =T,
colors =pal2)
```

- Found the user with maximum number of retweets using command:

```
>map3 <-function(k,v)
{
     Keyval(v$username, v$retweets)
}
>reduce3 <-function(k,vv)
{
     Keyval(k,sum(vv))
}
>out3 <-from.dfs(mapreduce(input = to.dfs(check),
map=map3,reduce=reduce3))
```



maria_dev@sandbox:~

```
> max(out3$val,na.rm = TRUE)
[1] 81503
> out3$key[which(out3$val == 81503)]
[1] "S"              "Farouk"         "????? ."        "ashley??"       "Isah Kabir."
[6] "Jay T. Duong"
> 
```

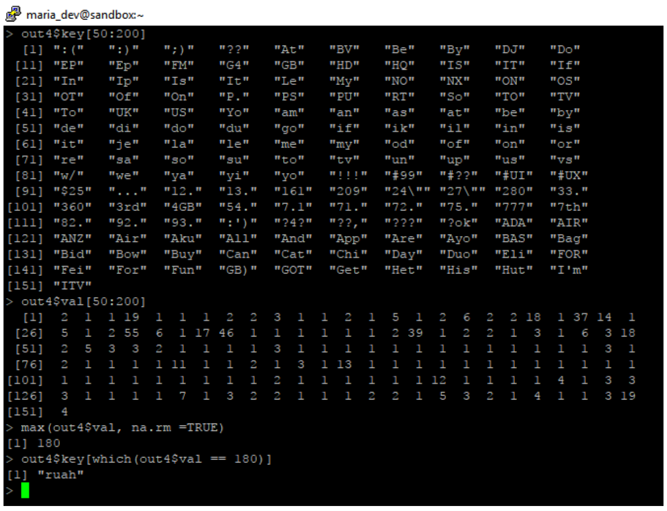
## 4.4 Data Pre-processing:

- Extracted individual words from the text and found its frequency distribution too using the following command:

```
>map4 <- function(k,v)
{
x <- unlist(strsplit(v$text, “  ”))
keyval(x,1)
}
reduce4 <-function(k,vv)
{
    Keyval(k,sum(vv))
}
>out4 <- from.dfs(mapreduce(input = to.dfs(check),
map=map4,reduce=reduce4))
```

- Retained “out2” generated in the data exploration step as a dataframe to hold hashtags.
- Modified “out4” to remove any special characters in individual words like “!” or “@”.

```
Map5  <-function(k,v) {
x<- str_replace_all(k, "!", "")
keyval(x,1)
}
>out5 <-from.dfs(mapreduce(input = to.dfs(check),
map=map5,reduce=reduce4))
```

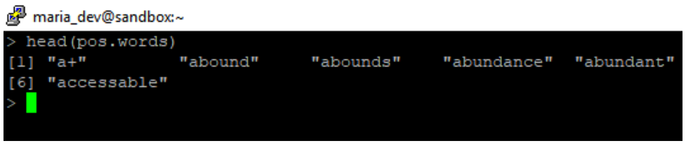


```
maria_dev@sandbox:~
> out4$key[50:200]
  [1] ":("   ":)"   ";)"   "??"   "At"   "BV"   "Be"   "By"   "DJ"   "Do"
 [11] "EP"   "Ep"   "FM"   "G4"   "GB"   "HD"   "HQ"   "IS"   "IT"   "If"
 [21] "In"   "Ip"   "Is"   "It"   "Le"   "My"   "NO"   "NX"   "ON"   "OS"
 [31] "OT"   "Of"   "On"   "P."   "PS"   "PU"   "RT"   "So"   "TO"   "TV"
 [41] "To"   "UK"   "US"   "Yo"   "am"   "an"   "as"   "at"   "be"   "by"
 [51] "de"   "di"   "do"   "du"   "go"   "if"   "ik"   "il"   "in"   "is"
 [61] "it"   "je"   "la"   "le"   "me"   "my"   "od"   "of"   "on"   "or"
 [71] "re"   "sa"   "so"   "su"   "to"   "tv"   "un"   "up"   "us"   "vs"
 [81] "w/"   "we"   "ya"   "yi"   "yo"   "!!!"  "#99"  "#??"  "#UI"  "#UX"
 [91] "$25"  "..."  "12."  "13."  "161"  "209"  "24\"" "27\"" "280"  "33."
[101] "360"  "3rd"  "4GB"  "54."  "7.1"  "71."  "72."  "75."  "777"  "7th"
[111] "82."  "92."  "93."  ":')"  "?4?"  "??,"  "???"  "?ok"  "ADA"  "AIR"
[121] "ANZ"  "Air"  "Aku"  "All"  "And"  "App"  "Are"  "Ayo"  "BAS"  "Bag"
[131] "Bid"  "Bow"  "Buy"  "Can"  "Cat"  "Chi"  "Day"  "Duo"  "Eli"  "FOR"
[141] "Fei"  "For"  "Fun"  "GB)"  "GOT"  "Get"  "Het"  "His"  "Hut"  "I'm"
[151] "ITV"
> out4$val[50:200]
  [1]  2  1  1 19  1  1  1  2  2  3  1  1  2  1  5  1  2  6  2  2 18  1 37 14  1
 [26]  5  1  2 55  6  1 17 46  1  1  1  1  1  1  2 39  1  2  2  1  3  1  6  3 18
 [51]  2  5  3  3  2  1  1  1  1  3  1  1  1  1  1  1  1  1  1  1  1  1  1  3  1
 [76]  2  1  1  1  1 11  1  1  2  1  3  1 13  1  1  1  1  1  1  1  1  1  1  1  1
[101]  1  1  1  1  1  1  1  1  1  2  1  1  1  1  1  1 12  1  1  1  1  4  1  3  3
[126]  3  1  1  1  1  7  1  3  2  2  1  1  1  2  2  1  5  3  2  1  4  1  1  3 19
[151]  4
> max(out4$val, na.rm =TRUE)
[1] 180
> out4$key[which(out4$val == 180)]
[1] "ruah"
>
```

## 4.5 Data Analysis:

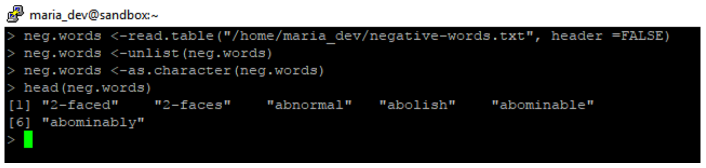
As discussed before, we use Lexical technique to perform sentiment analysis. Therefore, we download a list of positive and negative words on to our Virtual Machine and then using R Hadoop, format it and store it in HDFS, using the following commands:

```
>pos.words  <-read.table("/home/maria_dev/positive-
words.txt", header = FALSE)
> pos.words <-unlist(pos.words)
>pos.words<- as.character(pos.words)
>to.dfs(pos.words)
```



```
maria_dev@sandbox:~
> head(pos.words)
[1] "a+"           "abound"       "abounds"      "abundance"    "abundant"
[6] "accessable"
>
```

```
>neg.words  <-read.table("/home/maria_dev/negative-
words.txt", header = FALSE)
>neg.words <-unlist(neg.words)
>neg.words<- as.character(neg.words)
>to.dfs(neg.words)
```



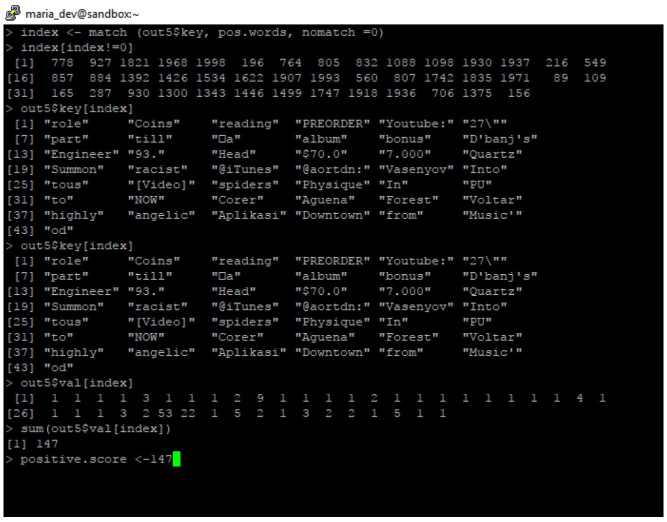
```
maria_dev@sandbox:~
> neg.words <-read.table("/home/maria_dev/negative-words.txt", header =FALSE)
> neg.words <-unlist(neg.words)
> neg.words <-as.character(neg.words)
> head(neg.words)
[1] "2-faced"     "2-faces"     "abnormal"    "abolish"     "abominable"
[6] "abominably"
>
```

Now, all the required data sets needed for analysis are stored in different data frames, i.e., out5, pos.words and neg.words.

### 4.5.1 Sentiment Analysis:

- We compare the extracted words of the tweets in out5 to the positive and negative word lists namely pos.words and neg. words. Command:
- For positive words comparison:

```
>index <-match( out5$key, pos.words, nomatch =0)
>sum(out5$key[index])
```

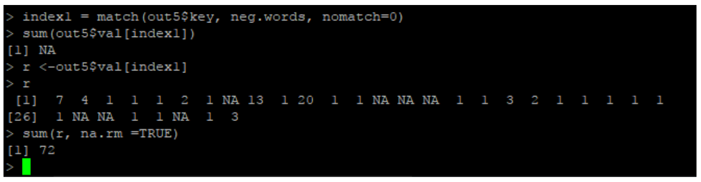


```
maria_dev@sandbox:~
> index <- match (out5$key, pos.words, nomatch =0)
> index[index!=0]
 [1]  778  927 1821 1968 1998  196  764  805  832 1088 1098 1930 1937  216  549
[16]  857  884 1392 1426 1534 1622 1907 1993  560  807 1742 1835 1971   89  109
[31]  165  287  930 1300 1343 1446 1499 1747 1918 1936  706 1375  156
> out5$key[index]
 [1] "role"     "Coins"    "reading"  "PREORDER" "Youtube:" "27\""
 [7] "part"     "till"     "□a"       "album"    "bonus"    "D'banj's"
[13] "Engineer" "93."      "Head"     "$70.0"    "7.000"    "Quartz"
[19] "Summon"   "racist"   "@iTunes"  "@aortdn:" "Vasenyov" "Into"
[25] "tous"     "[Video]"  "spiders"  "Physique" "In"       "PU"
[31] "to"       "NOW"      "Corer"    "Aguena"   "Forest"   "Voltar"
[37] "highly"   "angelic"  "Aplikasi" "Downtown" "from"     "Music'"
[43] "od"
> out5$key[index]
 [1] "role"     "Coins"    "reading"  "PREORDER" "Youtube:" "27\""
 [7] "part"     "till"     "□a"       "album"    "bonus"    "D'banj's"
[13] "Engineer" "93."      "Head"     "$70.0"    "7.000"    "Quartz"
[19] "Summon"   "racist"   "@iTunes"  "@aortdn:" "Vasenyov" "Into"
[25] "tous"     "[Video]"  "spiders"  "Physique" "In"       "PU"
[31] "to"       "NOW"      "Corer"    "Aguena"   "Forest"   "Voltar"
[37] "highly"   "angelic"  "Aplikasi" "Downtown" "from"     "Music'"
[43] "od"
> out5$val[index]
 [1]  1  1  1  1  3  1  1  1  2  9  1  1  1  1  2  1  1  1  1  1  1  1  1  4  1
[26]  1  1  1  3  2 53 22  1  5  2  1  3  2  2  1  5  1  1
> sum(out5$val[index])
[1] 147
> positive.score <-147
```

Therefore, positive score = 147

- For negative words comparison:

```
>index1  <-match(  out5$key,  neg.words,  nomatch
=0)
>sum(out5$key[index1])
```



```
> index1 = match(out5$key, neg.words, nomatch=0)
> sum(out5$val[index1])
[1] NA
> r <-out5$val[index1]
> r
 [1]  7  4  1  1  1  2  1 NA 13  1 20  1  1 NA NA NA  1  1  3  2  1  1  1  1  1
[26]  1 NA NA  1  1 NA  1  3
> sum(r, na.rm =TRUE)
[1] 72
>
```

Therefore, negative score = 72.

## 5. Results:

1. Overall sentiment of the twitter feeds on 29th April 2016 was positive (147-72 = +75)
2. The most tweeted word on that particular day was “ruah”
3. Found names of 5 users whose tweets were most retweeted.

## 6. Drawbacks of the given implementation:

1. The extracted words could be formatted more for comparison with the positive and negative words. For e.g., special characters like “!#$^&*” and numbers could be removed for the words extracted. So “amazing123!” would become “amazing”. This would help compare “amazing” with positive word list. Also, the words extracted could be turned to lowercase, again for better overall comparison.
2. The reference word list (pos.words and neg.words) need to be more extensive so as to include different languages and slangs.
3. Graphical representation of the results could have been better.

## 7. Future Improvements:

1. Instead of lexical, machine learning algorithm can be implemented for a more adaptive sentiment analysis.
2. The implementation can be built around a specific product or a country or a personality, depending upon the application of the implementation.
3. Alternative approaches discussed in section 8 can be used for real-time stream processing of data.

## 8. Alternative Approach:

In this project, we have used tools like Python for data extraction and R Hadoop for data preparation and Analysis. There are several approaches to accomplish the same goal. For example, we could have used Apache Flume for setting up STREAMING API and efficiently collecting, aggregating, and moving large amounts of streaming data into the Hadoop Distributed File System (HDFS). Similarly, stream processing platforms like Apache Storm/ Apache Flink/ Apache Spark Streamingetc can be used to process the high velocity incoming data. We will look at all these platforms in detail individually.

### 8.1 Apache Flume:

Flume is a distributed, reliable, and available system for efficiently collecting, aggregating, and moving high volume, streaming data flows from many sources to a centralised store. It has a simple and flexible architecture based on streaming data flows. It was Created by Cloudera. Cloudera gave control of Flume to Apache Software Foundation in 2011.

#### 8.1.1 Architecture:

Event: Singular unit of data that can be transported, like a single log entry or a tweet.

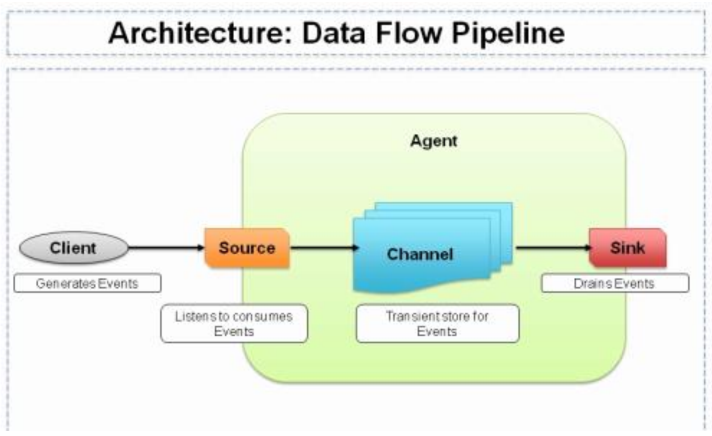
Clients: They produce data in the form of events (not files)

Source: Listens for and consumes event

Channels: Mechanisms by which Flume agents transfer events from their sources to their sinks. They are a transient store for these events before they are drained by the sinks. This temporary storage allows source and sinks to run asynchronously.

Sink: An interface implementation that can remove events from channels and transmit them to the next agent in the flow or the event’s final destination.

Agent: It acts as a container for the entire Flume data flow.It is an independent process that hosts flume components such as sources, channels and sinks, and thus have the ability to receive, store and forward events to their next-hop destination.


Architecture: Data Flow Pipeline
Agent
Client
Generates Events
Source
Listens to consumes Events
Channel
Transient store for Events
Sink
Drains Events

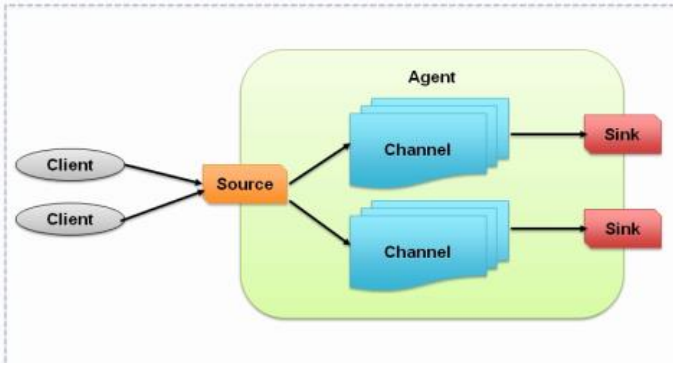


*Figure 4: Apache Flume Architecture*

### 8.1.2 Downloading and configuration:

- Download Apache Flume on the Hortonworks Sandbox using:
  wget http://www.apache.org/dyn/closer.lua/flume/1.8.0/apache-flume-1.8.0-bin.tar.gz
- After download, unzip it using: tar -zxvf apache-flume-1.8.0-bin.tar.gz
- Download JAR file containing the twitter source class using command:
  wget http://files.cloudera.com/samples/flume-sources-1.0-SNAPSHOT.jar
- Now we need to create/edit a configuration file on the flume/conf folder on the Virtual Machine.
- Create twitter-agent.conf by using:
  vi twitter-agent.conf
- Define components of agents:
  Twitter-agent.sources =twitter-source
  Twitter-agent.channels=mem-channel
  Twitter-agent.sink =hdfs-sink
- Flow definition:
  Twitter-agent.sources.twitter-source.channels = mem-channel
  Twitter-agent.sinks.hdfs-sink.channel=mem-channel
- Configure source:

  Twitter-agent.sources.twitter-source-type = com.cloudera.flume.source.TwitterSource
  Twitter -agent.sources.twitter-source.consumerKey= <Consumer Key>
  Twitter-agent.source.twitter-source.consumerSecret = < Consumer Secret>
  Twitter-agent.source.twitter-source.accessToken = < Access Token>
  Twitter-agent.source.twitter-source.accessTokenSecret = < Access Token Secret>
  Twitter-agent.sources.twitter-source.keywords = Hadoop, big data, analytics

- Configure Channel:
  Twitter-agent.channels.mem-channel.type = memory

- Configure Sink:
  Twitter-agent.sinks.hdfs-sink.type = hdfs
  Twitter-agent.sinks.hdfs-sink.hdfs.path = hdfs://localhost:54310/flume-new/tweets
  Twitter-agent.sinks.hdfs-sink.fileType = DataStream

- Now save and close .conf file.
- Enter command: Flume-ng agent -n twitter-agent -f twitter-agent.conf
- Open a duplicate window to check if the tweets are getting extracted. Command:

  Hadoop fs -ls /flume-new/tweets

## <u>8.2 Open-source Stream-Processing platforms:</u>
(Martin Andreoni Lopez)

### 6.1.1 Apache Storm

Apache Storm is a real-time stream processor, written in Java and Clojure. Stream data abstraction is called tuples, composed by the data and an identifier. In Storm, applications consists of topologies forming a directed acyclic graph composed of inputs nodes, called spouts, and processing nodes, called bolts. A topology works as a data graph. The nodes process the data as the data stream advance in the graph. A topology is analog to a MapReduce Job in Hadoop. The grouping type used defines the link between two nodes in the processing graph. The grouping type allow the designer to set how the data should flow in topology. Storm has eight data grouping types that represent how data is sent to the next graph-processing node, and their parallel instances, which perform the same processing logic. The main grouping types are: shuffle, field, and all grouping. In shuffle grouping, the stream is randomly sent across the bolt instances. In field grouping, each bolt instance is responsible for all samples with the same key specified in the tuple. Finally, in all grouping, samples are sent to all parallel instances.

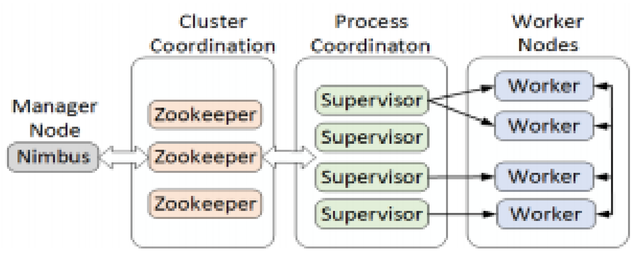


***Figure 5: Nimbus receives topologies and communicates to Supervisors which coordinate process in workers, all the coordination between Nimbus and Supervisor is made by Zookeeper who store the cluster state.***

### 6.2.2 Apache Flink:

Apache Flink is a hybrid processing platform, supporting both stream and batch processing. Flink core is the stream processing, making batch processing a special class of application. Analytics jobs in Flink compile into a directed graph of tasks. Apache Flink is written in Java and Scala. Similar to Storm, Flink uses a master-worker model. The job manager interfaces with clients applications, with responsibilities similar to Storm master node. The job manager receives client applications, organizes the tasks and sends them to workers. In addition, the job manager maintains the state of all executions and the status of each worker. The workers states are informed through the mechanism Heartbeat, like in Storm. Task manager has a similar function as a worker in Storm. Task Managers perform tasks assigned by the job manager and exchange information with other workers when needed. Each task manager provides a number of processing slots to the cluster that are used to perform tasks in parallel.

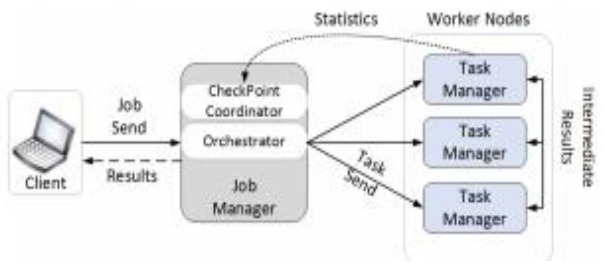


***Figure 6: Architecture of Flink system. The Job manager receives jobs from clients, divides the jobs into tasks, and sends the tasks to the workers. Workers communicate statistics and results***

### 6.2.3 Apache Spark Streaming:

Spark is a project initiated by UC Berkeley and is a platform for distributed data processing, written in Java and Scala. Spark has different libraries running on the top of the Spark Engine, including Spark Streaming [8] for stream processing. The stream abstraction is called Discrete Stream (D-Stream) defined as a set of short, stateless, deterministic tasks. In Spark, streaming computation is treated as a series of deterministic batch computations on small time intervals. Similar to MapReduce, a job in Spark is defined as a parallel computation that consists of multiple tasks, and a task is a unit of work that is sent to the Task Manager. When a stream enters Spark, it divides data into micro-batches, which are the input data of the Distributed Resilient Dataset (RDD), the main class in Spark Engine, stored in memory. Then the Spark Engine executes by generating jobs to process the micro-batches.

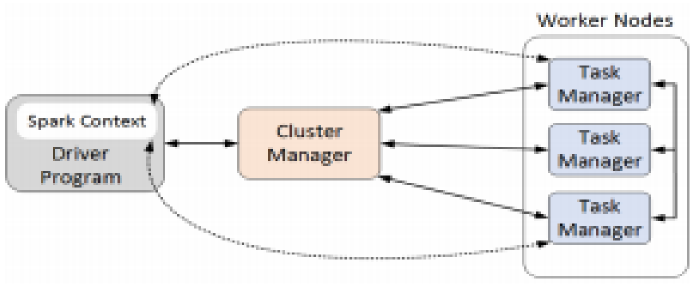
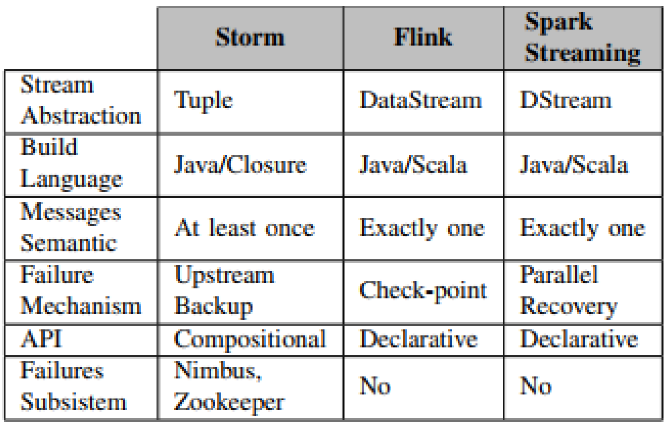


***Figure 7: Cluster architecture of Spark Streaming system.***

Applications or jobs within the Spark run as independent processes in the cluster which is coordinated by the master or Driver Program, responsible for scheduling tasks and creating the Spark Context. The Spark Context connects to various types of cluster managers, such as the Spark StandAlone, Figure 3: Cluster architecture of Spark Streaming system. Mesos or Hadoop YARN (Yet Another Resource Negotiator). These cluster managers are responsible for resource allocation between applications. Once connected, Spark executes task within the task managers, which perform processing and data storage, equivalent to Storm workers, and results are then communicated to the Spark Context. The mechanism described in Storm, in which each worker process runs within a topology, can be applied to Spark, where applications or jobs are equivalent to topologies. A disadvantage of this concept in Spark is the messages exchange between different programs, which is only done indirectly such as writing data to a file, worsen the latency that could be around seconds in applications of several operations.



| | Storm | Flink | Spark Streaming |
|---|---|---|---|
| Stream Abstraction | Tuple | DataStream | DStream |
| Build Language | Java/Closure | Java/Scala | Java/Scala |
| Messages Semantic | At least once | Exactly one | Exactly one |
| Failure Mechanism | Upstream Backup | Check-point | Parallel Recovery |
| API | Compositional | Declarative | Declarative |
| Failures Subsistem | Nimbus, Zookeeper | No | No |

***Table 1: Overview of the comparison between Stream Processing Systems.***

.In comparison, Storm has the highest processing rate. As far as behavior of the systems are concerned during node failure, Spark streaming, using micro-batch processing model, recovers the failure without losing any messages. Apache Flink, using a checkpoint algorithm, has a lower message loss rate, about a 12.8% during the redistribution process after a failure. Storm loses 10% more, about 22.2% of messages since it uses a subsystem, Zookeeper, for nodes synchronization. Therefore, in order to select a platform, the user should take into account the application requirements and balance the compromise between high processing rates and fault tolerance.

## 9. Conclusion:

The aim of the project was to extract , store, manage and analyse Twitter feeds ( Unstructured Big Data) . We di so using R Hadoop and Lexical sentiment analysis techniques. There are multiple ways of doing so. We could have used Apache Storm instead or Flume instead too. We can use machine learning algorithms on Big Data too. The decision of how platform and what technique to use depends upon the final application and aim of the project. For the current set of twitter feeds ( tweets on 29th April 2016), the overall sentiment is positive.

## References:

1. M. A. Lopez, A. G. P. Lobato and O. C. M. B. Duarte, "A Performance Comparison of Open-Source Stream Processing Platforms," *2016 IEEE Global Communications Conference (GLOBECOM)*, Washington, DC, 2016, pp. 1-6. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7841533&isnumber=7841475

2. Z. Zhao, Zheng Feng, Yong Zhang, Li Ning, J. Fan and S. Feng, "Collecting, managing and analyzing social networking data effectively," *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, Zhangjiajie, 2015, pp. 1642-1646. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7382191&isnumber=7381900

3. Shubham S Deshmukh, Harshal Joshi, Pranali Pandhare, Aniket More, Prof. Aniket M Junghare,"Twitter Data Analysis using R", International Journal of Science, Engineering and Technology Research(IJSETR), Volume 6,Issue 4, April 2017, ISSN:2278-7798 URL: http://ijsetr.org/wp-content/uploads/2017/04/IJSETR- VOL-6- ISSUE-4- 504-507.pdf

# Sentiment Analysis on Twitter feeds

**Pragya Mishra, A20406346**

## Appendix A : Python Code to extract Twitter feeds

```
# -*- coding: utf-8 -*-
"""
Created on Tue  14 14:23:53 2017

@author: Pragya
"""

import configparser
from TwitterAPI import TwitterAPI

def get_twitter(config_file):
    """ Read the config_file and construct an instance of TwitterAPI.
    Args:
      config_file ... A config file in ConfigParser format with Twitter credentials
    Returns:
      An instance of TwitterAPI.
    """
    config = configparser.ConfigParser()
    config.read(config_file)
    twitter = TwitterAPI(
                   config.get('twitter', 'consumer_key'),
                   config.get('twitter', 'consumer_secret'),
                   config.get('twitter', 'access_token'),
                   config.get('twitter', 'access_token_secret'))
    return twitter

twitter = get_twitter('C:/Users/jaide/Desktop/Pragya/Big data project/twitter.cfg')
print('Established Twitter connection.')

type(twitter)

import csv
import time
csvfile = "C:/Users/jaide/Desktop/Pragya/Big data project/output.txt"
for i in range(100):
    #time.sleep(61 * 2)
    request = twitter.request('search/tweets', {'q': '24-09-2016'})
    if request.status_code == 200:
        tweets = [r['text'] for r in request]
        tweetsuser =[r['user'] for r in request]
        print('found %d tweets' % len(tweets))
        for j in range(0,len(tweets)):
            with open(csvfile, "a+", encoding="utf-8") as output:
                writer = csv.writer(output, delimiter =' ', quotechar =' ', lineterminator = '\n')
                writer.writerow(tweets[j]+"   end of this tweet"+ "\t")
            with open(csvfile1, "a+", encoding="utf-8") as output:
                writer = csv.writer(output, delimiter =' ', quotechar =' ', lineterminator = '\n')
                writer.writerow(tweetsuser[j])
    else:
        print('Got error: %s \nsleeping for 15 minutes.' % request.text)
        time.sleep(61 * 15)
```