TwitterSentimentAnalysis

Loading required packages and setting working directory

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)

library(twitteR)
```

```
##
## Attaching package: 'twitteR'
```

```
## The following objects are masked from 'package:dplyr':
##
##     id, location
```

```
library(httr)
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:httr':
##
##     content
```

```
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(syuzhet)
library(plotly)
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
## The following object is masked from 'package:NLP':
##
##     annotate
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:httr':
##
##     config
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

Authentication and Extracting Tweets

To extract tweets from Twitter API, we need to set up a Twitter developer account and set up Twitter Authentication

We want to analyse the sentiments for the tweets that mention the new Apple phone, iPhoneXS. We select keywords that will be used to get tweets from Twitter API. Then we use the searchTwitter function from the twitteR library. Using this, we extract the most recent tweets containing the keywords. Then we will export this as a csv and use that csv file for our analysis.

We then use only the columns we need and remove duplicate tweets.

```
tweet_df <- read.csv("/Users/shruhi/Desktop/Project/iphonex2.csv")
rt_only_tweets <- subset(tweet_df, select = c(2))
tweets_df <- unique(rt_only_tweets)
```

Cleaning Data

To conduct a sentiment analysis, we use Natural Language Processing. It consists of many many algorithms, some of which we will be implementing here using the text mining package in R called (tm).

First, we do a basic cleaning of the data. We get rid of unwanted characters that are frequently found in tweets, such as “&amp”, “RT|via”, “http\w+”, etc.

For data cleaning, tweet text is converted into a Corpora. We now have a ‘corpus’, or in simple words, a collection of written texts. To analyse the text, we need to clean this corpus. First of all, we convert all words to lowercase and remove punctuation. We also need to remove “stopwords”, i.e. words like for, that, on, etc that don’t really have an impact on the sentiment of the written text.

To visualise the frequent words, we use a wordcloud.

In any language, words are often associated that have a similar meaning - e.g. “associate” and “associated” can be interchanged as their impact is only on the grammar and syntax, not on the meaning of the sentence. To avoid repeating words, we use “stemming” in Natural Language Processing. Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form.

After this, we can create a document term matrix. A document-term matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a document-term matrix, rows correspond to documents in the collection and columns correspond to terms.

Lastly, a document-term matrix has at least a sparse percentage of empty (i.e., terms occurring zero times in the document) elements. We remove this, and specify the sparsity such that the resulting matrix contains only terms with a sparse factor of less than specified ‘sparse’, here, 0.99.

```
tweets_df$text = gsub("&amp", "", tweets_df$text)
tweets_df$text = gsub("&amp", "", tweets_df$text)
tweets_df$text = gsub("(RT|via)((?:\\b\\W*@\\w+)+)", "", tweets_df$text)
tweets_df$text = gsub("@\\w+", "", tweets_df$text)
tweets_df$text = gsub("[[:punct:]]", "", tweets_df$text)
tweets_df$text = gsub("[[:digit:]]", "", tweets_df$text)
tweets_df$text = gsub("http\\w+", "", tweets_df$text)
tweets_df$text = gsub("[ \t]{2,}", "", tweets_df$text)
tweets_df$text = gsub("^\\s+|\\s+$", "", tweets_df$text)
tweets_df$text <- iconv(tweets_df$text, "ASCII", "UTF-8", sub="")

#Cleaning the corpus
corpusiphone <- Corpus(VectorSource(tweets_df$text))

#Convert all to lower case
corpusiphone <- tm_map(corpusiphone, tolower)
```

```
## Warning in tm_map.SimpleCorpus(corpusiphone, tolower): transformation drops
## documents
```

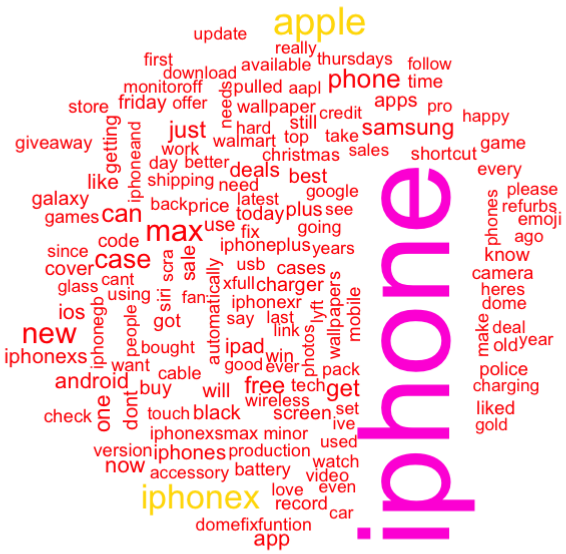
```
#Removing punctuation
corpusiphone <- tm_map(corpusiphone, removePunctuation)
```

```
## Warning in tm_map.SimpleCorpus(corpusiphone, removePunctuation):
## transformation drops documents
```

```
#Removing stopwords
corpusiphone <- tm_map(corpusiphone, removeWords, c(stopwords("en")))
```

```
## Warning in tm_map.SimpleCorpus(corpusiphone, removeWords,
## c(stopwords("en"))): transformation drops documents
```

```
#Creating a WordCloud
wordcloud(corpusiphone,colors=rainbow(7),max.words=150)
```




```
corpusiphone <- tm_map(corpusiphone, stripWhitespace)
```

```
## Warning in tm_map.SimpleCorpus(corpusiphone, stripWhitespace):
## transformation drops documents
```

```
#Creating Stem Document
corpusiphone <- tm_map(corpusiphone, stemDocument)
```

```
## Warning in tm_map.SimpleCorpus(corpusiphone, stemDocument): transformation
## drops documents
```

```
#Creating Document Term Matrix
frequenciesiphone <- DocumentTermMatrix(corpusiphone)
frequenciesiphone
```

```
## <<DocumentTermMatrix (documents: 5886, terms: 8638)>>
## Non-/sparse entries: 50044/50793224
## Sparsity           : 100%
## Maximal term length: 91
## Weighting          : term frequency (tf)
```

```
#Remove Sparse Terms
iphonesparse <- removeSparseTerms(frequenciesiphone, 0.99)

iphonesparse <- as.matrix(iphonesparse)
iphonesparse <- as.data.frame(iphonesparse)
colnames(iphonesparse) <- make.names(colnames(iphonesparse)) #make variable names R-friendly
```

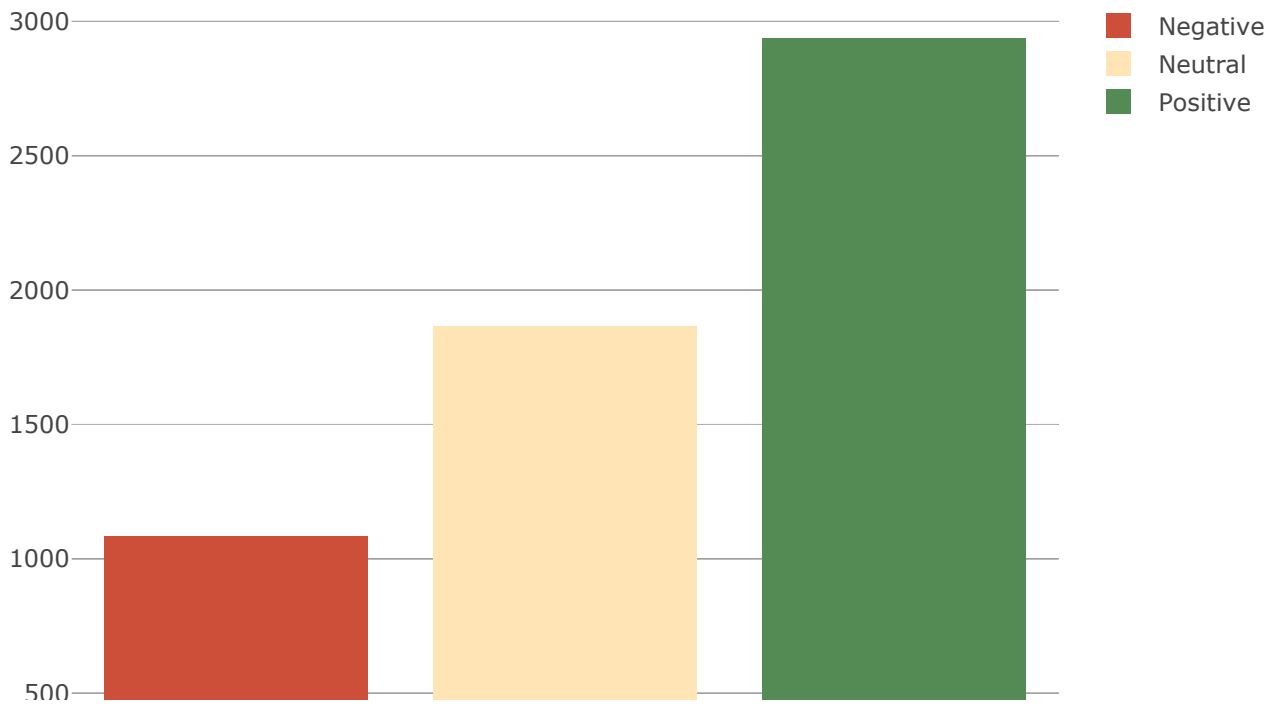
Sentiment Analysis

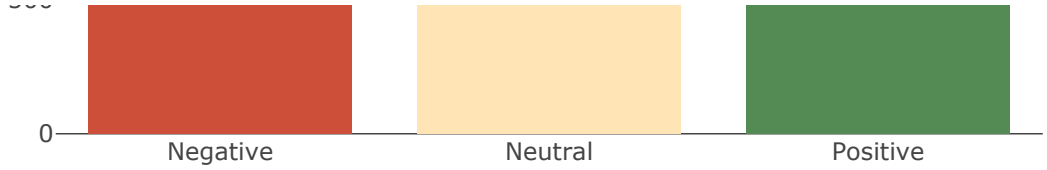
For sentiment analysis, we use the Syuzhet package. The package comes with four sentiment dictionaries and provides a method for accessing sentiment extraction tools. It iterates over a vector of strings and returns sentiment values based on user specified methods. We will implement two methods -

1. Polarity Analysis using “syuzhet” + We will get the sentiments in the form of polarities. It will be converted into categorical variables “Positive”, “Neutral”, “Negative”

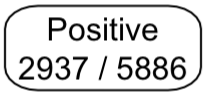
Polarity analysis

```
sentiment_value <- get_sentiment(tweets_df$text)

#Making the categorical variable of the sentiment values
category_sentiments <- ifelse(sentiment_value < 0, "Negative",
                              ifelse(sentiment_value > 0, "Positive", "Neutral"))
iphonesparse$polarity <- category_sentiments
three_polarity <- data.frame(table(iphonesparse$polarity))

col <- c("tomato3","moccasin","palegreen4")
plot_ly(three_polarity, x = three_polarity$Var1, y = three_polarity$Freq, type = "bar",
 color = three_polarity$Var1, colors = col)
```

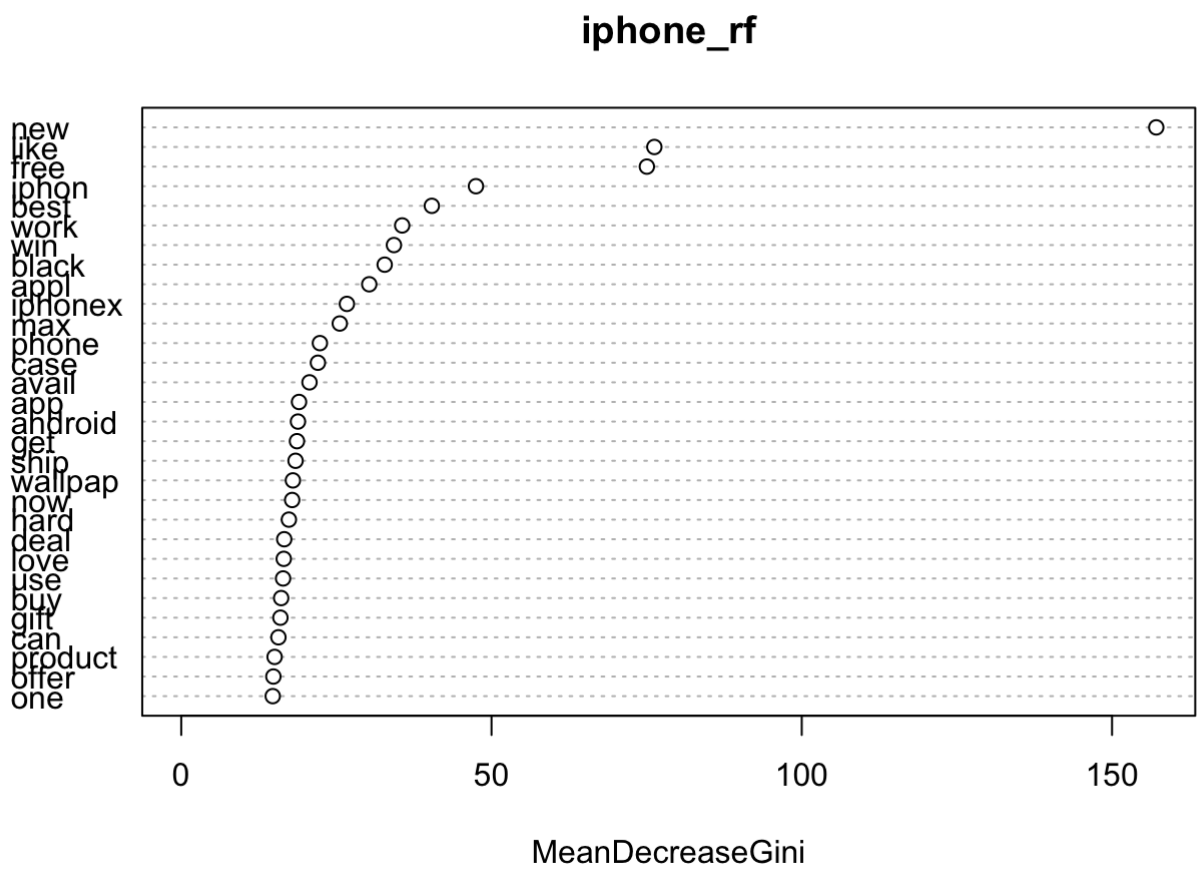





```
#To test this, we can run a CART and Random Forest
cart <- rpart(polarity ~ ., data = iphonesparse, method = "class")
prp(cart, extra = 2)
```



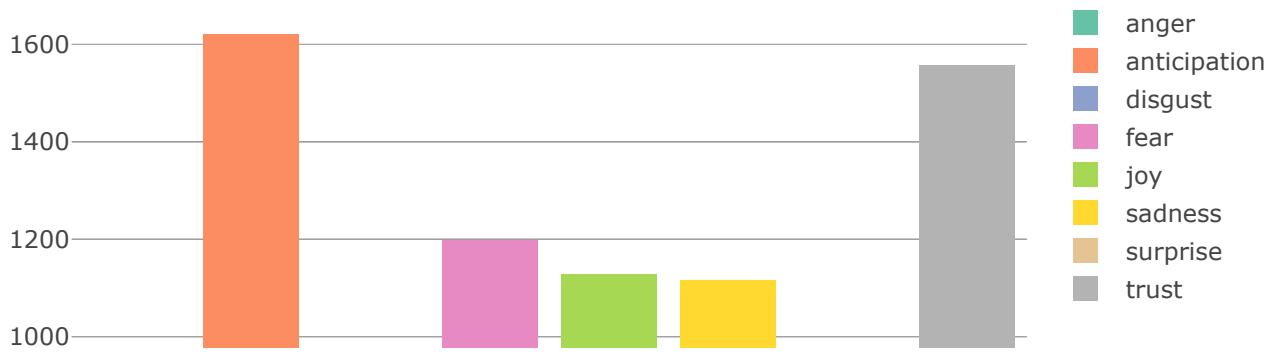

```
set.seed(345)
iphonesparse$polarity <- as.factor(iphonesparse$polarity)
iphone_rf <- randomForest(polarity ~ ., data = iphonesparse)
varImpPlot(iphone_rf)
```

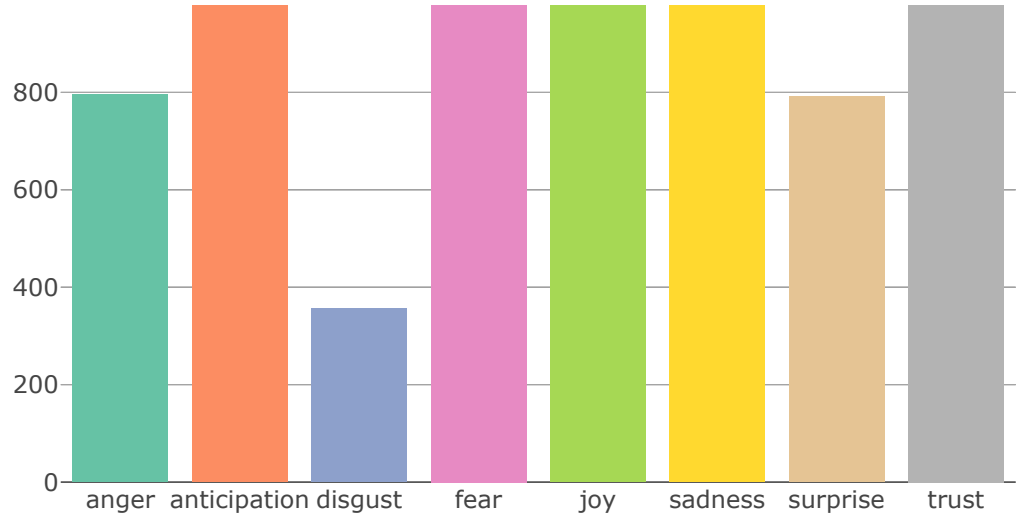


Emotion Analysis

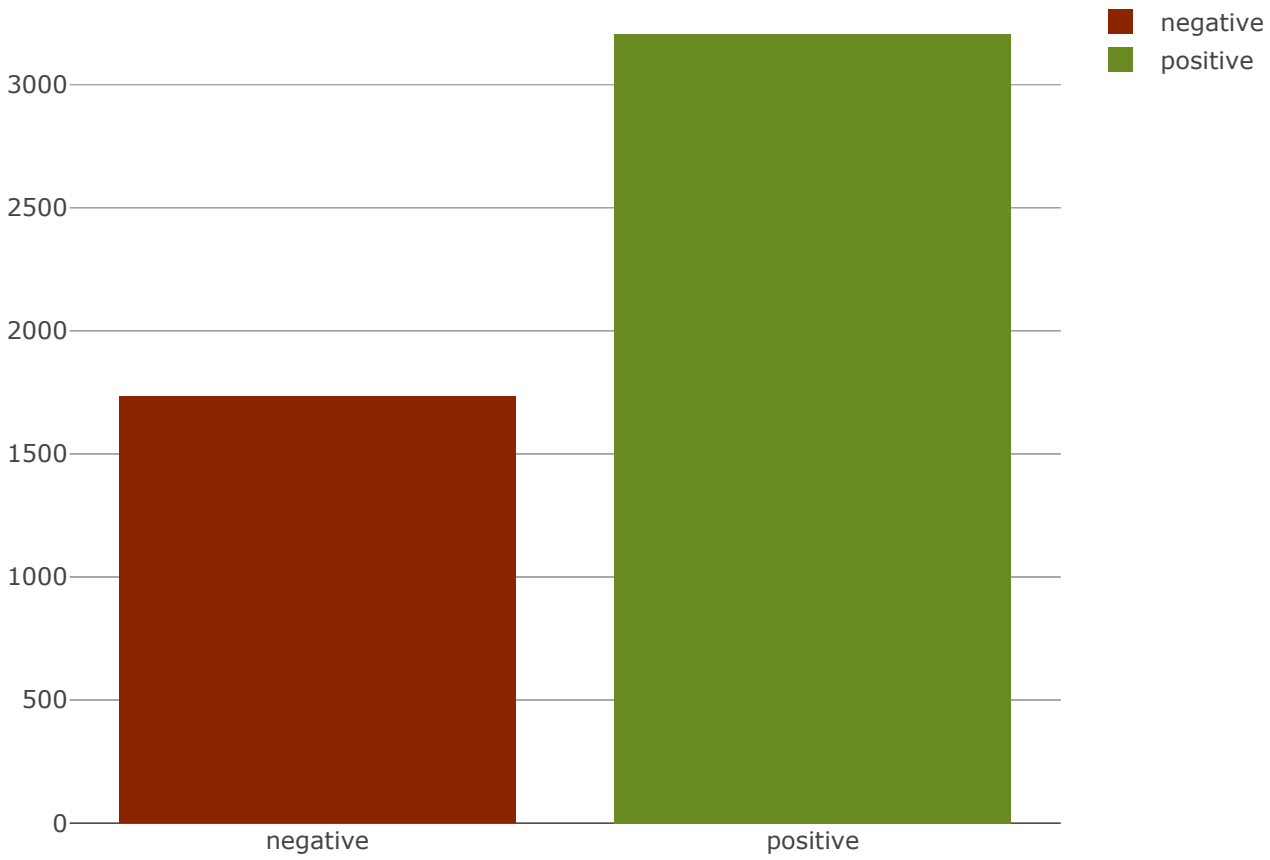
2. Emotion Analysis using “nrc” + We will get the sentiment in the form of emotions - ‘anger’, ‘anticipation’, ‘disgust’, ‘fear’, ‘joy’, ‘sadness’, ‘surprise’, ‘trust’

```
emotions <- get_nrc_sentiment(tweets_df$text)
emotion <- colSums(emotions[-c(9,10)])
emotion <- data.frame(Sum = emotion, Emotions = names(emotion))

polarity <- colSums(emotions[c(9,10)])
polarity <- data.frame(Sum = polarity, Polarity = names(polarity))

plot_ly(polarity, x = emotion$Emotions, y = emotion$Sum, type = "bar", color = emotion$E
motions)
```






```
colour <- c("orangered4", "olivedrab4")
plot_ly(polarity, x = polarity$Polarity, y = polarity$Sum, type = "bar", color = polarit
y$Polarity, colors = colour)
```



Additional analysis to be completed:

Lexicons available in syuzhet package: The nrc lexicon categorizes words in a binary fashion (“yes”/“no”) into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust. The bing lexicon categorizes words in a binary fashion into positive and negative categories. The AFINN lexicon assigns

words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment.

We will try to implement each one to get a deeper idea of analysis.