COS80023 – Big Data

Project Report

Kartik Patel (101982976)

Introduction

A study of the emotion is also known as “opinion mining” or “emotion Artificial Intelligence” and text mining, alludes to the utilization of natural language processing (NLP), computational linguistics, and extricate, bio measurements to methodically recognize, examine emotional states, evaluate, and subjective information. Sentiment analysis is generally concerned with consumer service experience, such as internet and mobile based polls and feedback.

The president Donald Trump has become famous for controversial events. He has been strongly criticised for many events such as his controversial call to immediately ban something from entering in United States.

A Sentiment analysis is one of the many uses of social media analytics, where they determine whether commenting in an issue is positive or negative. We can integrate R and Shiny for predictive modelling, machine learning, text data mining in social media analytics etc., by taking advantage of the numerous R packages and compelling shiny visualizations.

In this report, I has been mine tweets and analyse their sentiment using R. Also, for visualize the results using Shiny App. We will see tweets, cities and states with the highest tweets distributed on a space temporary basis, as well as mapping the tweet for sentiments. This will allow us to assess the areas of acceptance for his comments as positive and negative.

Analysis Steps

Tweets are imported using R and the data is cleaned by removing emoticons and URLs. Lexical Analysis as well as Naive Bayes Classifier is used to predict the sentiment of tweets and subsequently express the opinion graphically through ggplots, histogram, pie chart, wordcloud and tables. The front end has been created using the Shiny App.

System Requirements

- Installation of R and R studio updated version
- Twitter Authentication to access API

Step 1: Getting Twitter API Access

Getting a Twitter API is easy. First, we need to create Twitter account if no account exists. After, need to apply for a developer account and fill the application form with necessary details so twitter give permission for creation of twitter app.

Once application has been granted permission by Twitter, we will receive the following credentials:

- Consumer Key
- Consumer Secret
- Access Token
- Access Secret

Install the necessary packages

```
# Load the required Packages
library(twitteR), library(RCurl)
library(httr)
library(tm)
library(wordcloud)
```

Step 2: Twitter Data Extracting

twitteR is the standard download package for tweet through R, but when I extracted more than ~2000 tweets for given hashtag its large amount of difficulty. I wanted to extract up to 18,000 tweets using rtweet package every 15 minutes that exist in the last 7–10. If we want more than 18k limit and extract more tweets by iterating the extraction process using certain features like retryOnRateLimit and/or max_id arguments.

```
#======= Download data from Twitter Function========

tweets_downloader <- function(tag, n, lang='en', retryonratelimit = TRUE){

#==== Twitter Credential ==========

  twitter_token <- create_token(
    app = 'kremrikpatelsentiment',
    consumer_key <- "40m33pDH4K343RWzwICRJGEpj",
    consumer_secret <- "jUAZiNaZQjdQKDBnLWd21XwgUEhXKVA5v8PrpgHoiknjXHt1ep",
    access_token <- "524280352-I0jrQDY4HeFTvGoFvVIDlzhI9btIWtwyzj5DHonq",
    access_secret <- "ftCfpsbNBZlfkh8wiaw009P5CuJ6M2zxmklcnAzg8HsNq",
    set_renv = F
  )

  tweet.df <- search_tweets(tag, n = n, include_rts = FALSE, lang = lang, token = twitter_token, retryonratelimit = retryonratelimi

  print(paste0("Total Tweets downloaded for - ",tag,": ",length(tweet.df$text)))
  print(paste0("Total Unique Texts downloaded for - ",tag,": ",length(unique(tweet.df$text))))

  tweet.df$hashtags <- as.character(tweet.df$hashtags)
  tweet.df$symbols <- as.character(tweet.df$symbols)
  tweet.df$urls_url <- as.character(tweet.df$urls_url)
  tweet.df$urls_t.co <- as.character(tweet.df$urls_t.co)
  tweet.df$urls_expanded_url <- as.character(tweet.df$urls_expanded_url)
  tweet.df$media_url <- as.character(tweet.df$media_url)
  tweet.df$media_t.co <- as.character(tweet.df$media_t.co)
  tweet.df$media_expanded_url <- as.character(tweet.df$media_expanded_url)
  tweet.df$media_type <- as.character(tweet.df$media_type)
  tweet.df$ext_media_url <- as.character(tweet.df$ext_media_url)
  tweet.df$ext_media_t.co <- as.character(tweet.df$ext_media_t.co)
  tweet.df$ext_media_expanded_url <- as.character(tweet.df$ext_media_expanded_url)
  tweet.df$mentions_user_id <- as.character(tweet.df$mentions_user_id)
  tweet.df$mentions_screen_name <- as.character(tweet.df$mentions_screen_name)
  tweet.df$geo_coords <- as.character(tweet.df$geo_coords)
  tweet.df$coords_coords <- as.character(tweet.df$coords_coords)
  tweet.df$bbox_coords <- as.character(tweet.df$bbox_coords)


  tweet.df
}
```

After data extracting, I have been saved data locally in working directory with file name tweet_df_ds.rds.

```
#============ Get The data for all hashtags ==================

tweet_df_ds <- tweets_downloader(tag="#DonaldTrump OR #donaldtrump OR #trump OR #news OR #AntiTrump", n=2500, lang='en',
                                 retryonratelimit = TRUE)
saveRDS(tweet_df_ds, file = "tweet_df_ds.rds")
```

Step 3: Twitter Data Cleaning

Tweets are one crazy collection of messy data and the code below used for treating/removing them. Few cleaning steps include —hyperlinks, removing @ (screen name mentions), uni-code characters, punctuation, digits, and RT (retweets).

```
#========== Data Cleaning for Sentiment========

tweets_cleaner <- function(tweet.df){

  tweets_txt <- unique(tweet.df$text)
  clean_tweet = gsub("&amp", "", tweets_txt) # Remove Amp
  clean_tweet = gsub("(RT|via)((?:\\b\\W*@\\w+)+)", "", clean_tweet) # Remove Retweet
  clean_tweet = gsub("@\\w+", "", clean_tweet) # Remove @
  clean_tweet = gsub("#", " ", clean_tweet) # Before removing punctuations, add a space before every hash
  clean_tweet = gsub("[[:punct:]]", "", clean_tweet) # Remove Punct
  clean_tweet = gsub("[[:digit:]]", "", clean_tweet) # Remove Digit/Numbers
  clean_tweet = gsub("http\\w+", "", clean_tweet) # Remove Links
  clean_tweet = gsub("[ \t]{2,}", " ", clean_tweet) # Remove tabs
  clean_tweet = gsub("^\\s+|\\s+$", " ", clean_tweet) # Remove extra white spaces
  clean_tweet = gsub("^ ", "", clean_tweet)  # remove blank spaces at the beginning
  clean_tweet = gsub(" $", "", clean_tweet) # remove blank spaces at the end
  clean_tweet = gsub("[^[:alnum:][:blank:]?&/\\-]", "", clean_tweet) # Remove Unicode Char


  clean_tweet <- str_replace_all(clean_tweet," "," ") #get rid of unnecessary spaces
  clean_tweet <- str_replace_all(clean_tweet, "https://t.co/[a-z,A-Z,0-9]*","") # Get rid of URLs
  clean_tweet <- str_replace_all(clean_tweet, "http://t.co/[a-z,A-Z,0-9]*","")
  clean_tweet <- str_replace(clean_tweet,"RT @[a-z,A-Z]*: ","") # Take out retweet header, there is only
  clean_tweet <- str_replace_all(clean_tweet,"#[a-z,A-Z]*","") # Get rid of hashtags
  clean_tweet <- str_replace_all(clean_tweet,"@[a-z,A-Z]*","") # Get rid of references to other screennam

  clean_tweet
}
```

Step 4: Twitter Transformation of Data

Since transforming the tweets into lower case and deleting stooping words. Also included custom stop words, the text information should be translated to a corpus of records, the basis for additional analysis.

```
#==== Transformation of data ======
tweets_cleaner_tm <- function(clean_tweet, custom_stopwords = c("bla bla")){

  docs <- Corpus(VectorSource(clean_tweet))

  docs <- tm_map(docs, content_transformer(tolower)) # Convert the text to lower case
  docs <- tm_map(docs, removeNumbers) # Remove numbers
  docs <- tm_map(docs, removeWords, stopwords("english")) # Remove english common stopwords
  docs <- tm_map(docs, removeWords, custom_stopwords)  # Remove your own stop word
  docs <- tm_map(docs, removePunctuation) # Remove punctuations
  docs <- tm_map(docs, stripWhitespace) # Eliminate extra white spaces
  docs <- tm_map(docs, stemDocument) # Text stemming
  docs
}
```

Step 5: Twitter Data Sentiment Function Execution

The project is about sentiment analysis, for this I have used the package NLP with method get_nrc_sentiment. This function converts the data into emotion and sentiment score. To do this we need clean tweets data. Once Sentiment function has been executed successfully then load this data into local working directory with file name emotion_df_ds.rds.

```
#================== Sentiment Analysis ==========================

tweet_df_ds <- readRDS(file = "tweet_df_ds.rds")


word.df <- as.vector(tweets_cleaner(tweet_df_ds))

#=============== Emotion Analysis Data=======================
emotion_df_ds <- get_nrc_sentiment(word.df)
saveRDS(emotion_df_ds, file = "emotion_df_ds.rds")
```

Step 6: Twitter Data Transform Map Representation

This is the last step for data preparation. In this step extracted and clean tweet data transform using cities records. Data mapping with the cities and each tweet in each city joined to their respective sentiment. Once this Map data transformation and mapping is finished then after saving this data into sentimentmap_df_ds.rds into working directory.

```
#==============Seniment Map Data==============
cities <- st_read("cities.shp") %>%
  select(OBJECTID, NAME, geometry, POPULATION) %>%
  rename(city_name = NAME, CID = OBJECTID, pop = POPULATION)


  search_term <- "#DonaldTrump OR #donaldtrump OR #trump OR #news OR #AntiTrump"
  #input$n_cities
  temp <- cities %>%
    top_n(20, pop) %>%
    mutate(query = paste(geometry[]))


  # Create a character vector will house each lat long pair

  # Initialize vector
  search_xy <- character(5)
  for (i in 1:nrow(temp)) {
    search_xy[i] <- paste0(temp$geometry[[i]][2],",", temp$geometry[[i]][1],",10mi")

  }
  # Bind the query vector to the cities internal dataframe
  temp$query <- search_xy

  city <- temp
  tweets <- map_df(city$query, function (x) {search_tweets(search_term, n = 2500,
  geocode = x, lang = "en", include_rts = FALSE) %>% mutate(query = x)})
```

```
city_tweets <- left_join(tweets, city, by = "query") %>%
  select(screen_name, text, retweet_count,
         favorite_count, CID, geometry, city_name) %>%

  mutate(id = row_number()) # Create unique identifier

# Perform sentence level sentiment analyis
tweet_sentiment <- city_tweets %>%
  select(id, text) %>%
  mutate(sentence = get_sentences(text)) %$% # Note the idiosyncratic pipe
  sentiment_by(sentence, id)

# Extract Id & sentiment
id_sent <- tweet_sentiment %>%
  select(id, ave_sentiment) %>%
  as_tibble() # convert to a tibble for ease of use (though data.table is good)

# Each tweet in each city joined to their respective sentiment
city_sentiment <- inner_join(city_tweets, id_sent, by = "id")

city_agg <- city_sentiment %>%
  group_by(CID) %>%
  summarise(avg_sent = mean(ave_sentiment),
            min_sent = min(ave_sentiment),
            max_sent = max(ave_sentiment),
            tweet_min_id = which.min(ave_sentiment),
            tweet_max_id = which.max(ave_sentiment))

pos <- city_sentiment[city_agg$tweet_min_id,"text"] %>%
  rename(pos = "text")

neg <- city_sentiment[city_agg$tweet_max_id,"text"] %>%
  rename(neg = "text")

city_agg <- bind_cols(city_agg, pos, neg) %>%
  inner_join(temp) %>%
  st_as_sf()

#====== Save sentiment map data =======
saveRDS(city_agg, file = "sentimentmap_df_ds.rds")
```

Step 7: Data visualization Using Shiny

In the first six step we are extracting, cleaning, transformation of data, and sentiment analysis data need to represent or visualization using R shiny.

First, we need to install the below package for visualization data into shiny dashboard.

library(shiny)

library(shinydashboard)

library(highcharter)

library(dplyr)

library(NLP)

library(tm) # text mining

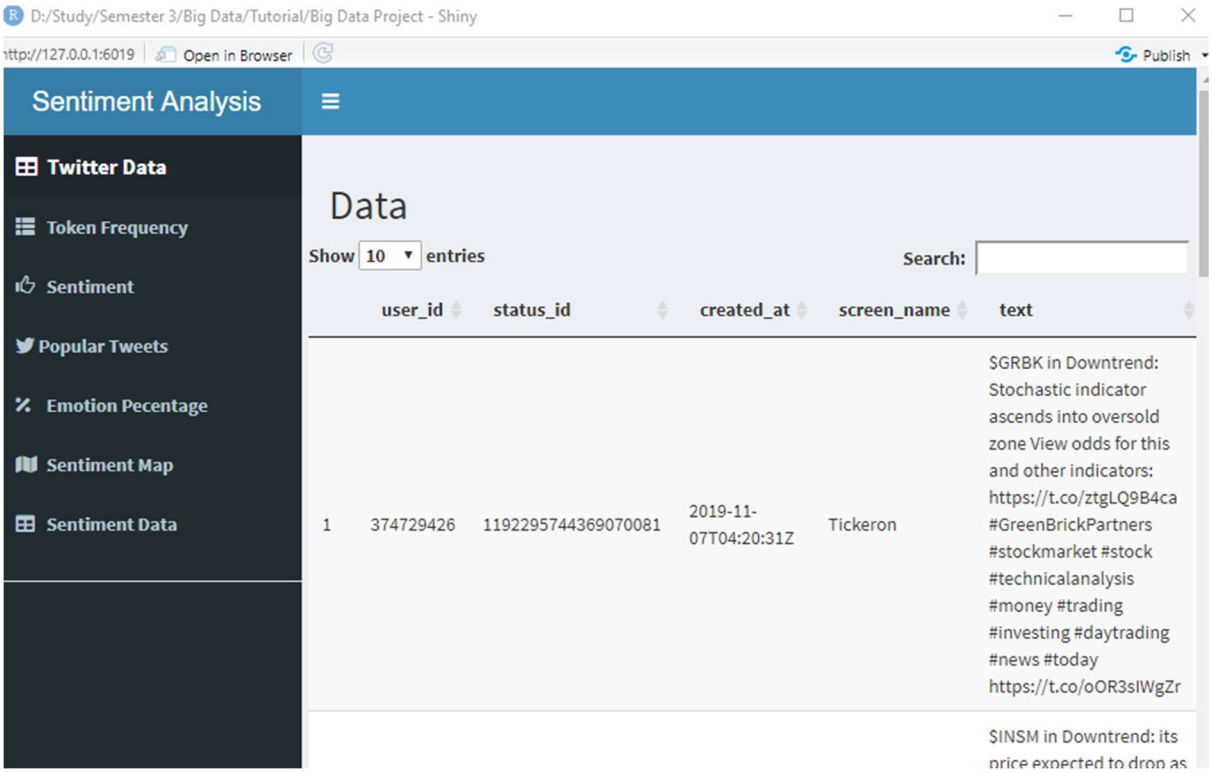
library(stringr)

library(SnowballC) # text stemming

library(RColorBrewer) # Color Palettes

```
library(wordcloud)
library(wordcloud2)
library(topicmodels)
library(tidytext)
library(slam)
library(tidyr)
library(igraph)
library(ggraph)
library(widyr)
library(quantmod)
library(rlist)
library(tweenr)
library(graphlayouts)
library(plotrix)
library(ggplot2)
library(ggmap)
library(maps)
library(mapdata)
library(leaflet)
library(curl)
library(tidyverse)
library(sf)
library(sentimentr)
library(magrittr)
library(xts)  # Require to load for leaflet
library(zoo) # Require to load for leaflet
library(TTR)  # Require to load for leaflet
```

The home page of dashboard looks like this:



The dashboard side bar menu is created on UI side using HTML and CSS.

```
#=== Dashboard Starts ===

header <- dashboardHeader(title= "Sentiment Analysis")

sidebar <- dashboardSidebar(
  sidebarMenu(id = 'menu',
              menuItem(strong("Twitter Data"),tabName = 'tdata', icon = icon("table")),
              menuItem(strong("Token Frequency"),tabName = 'token', icon = icon("th-list", lib = "glyphicon")),
              menuItem(strong("Sentiment"),tabName = 'sentiment', icon = icon("thumbs-up", lib = "glyphicon")),
              menuItem(strong("Popular Tweets"),tabName = 'popular', icon = icon("twitter")),
              menuItem(strong("Emotion Pecentage"),tabName = 'emotionalpercentages', icon = icon("percentage")),
              menuItem(strong("Sentiment Map"),tabName = 'map', icon = icon("map")),
              menuItem(strong("Sentiment Data"),tabName = 'sdata', icon = icon("table"))
  ),
  hr(),
  htmlOutput("topic_selector")
)
```

The below code is CSS for this shiny app

```
#====== HTML CSS Start============
  # Title
  tags$head(tags$style(HTML(
    '.skin-blue .main-sidebar .sidebar .sidebar-menu .active a{
    font-weight: bold;font-size: 16px;
    }
    '))),
  tags$head(tags$style(HTML(
    '.myClass {
    font-size: 20px;
    line-height: 50px;
    text-align: left;
    font-family: "Helvetica Neue",Helvetica,Arial,sans-serif;
    padding: 0 15px;
    overflow: hidden;
    color: white;
    }
    '))),

  tags$script(HTML('
                    $(document).ready(function() {
                    $("header").find("nav").append(\'<span class="myClass" style="white-space:pre">      </span>\');
                    })
                    ')),
  tags$head(tags$style(HTML('.modal-sm {width: 40px;}'))),

  # Set the boxes of all charts
  tags$head(tags$style(HTML(".col-sm-2,.col-sm-12,.col-sm-4,.col-sm-12,.col-sm-6,.col-sm-7,.col-sm-5 {
                            position: relative;
                            min-height: 1px;
                            padding-right: 5px;
                            padding-left: 5px;"))),

  tags$head(tags$style(HTML(".container-fluid {padding-left: 5px; padding-right: 5px;}"))),
  tags$head(tags$style(HTML(".form-group {margin-bottom: -15px;}"))),

  tags$head(tags$style(HTML(".box {margin-bottom: 10px;}"))),


  tags$head(tags$style(HTML("#col_word_cloud,#col_freq {padding-left:0px;padding-right:0px;} "))),
  tags$head(tags$style(HTML(".box-header {text-align: center;} "))),
  tags$head(tags$style(HTML("#network_panel {width:100%;} "))),
```

Step 7.1: Twitter Data Table representation

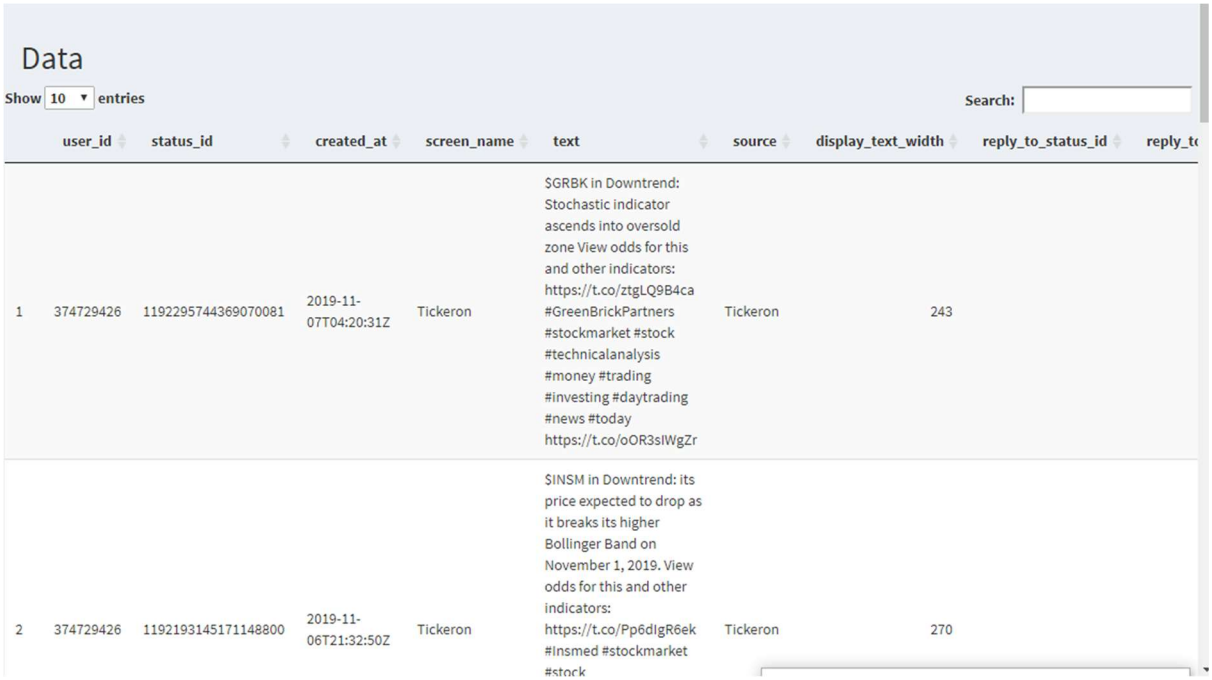
The first tab is about twitter data which we have extracted from twitter. This is simple data table rendering in UI side. First, I have created new tab into dashboard body and inside this I have shown the data into table format using the function dataTableOutput.

```
# Create a new  tab for twitter  data
  tabItems(
    tabItem("tdata",
            fluidPage(
              titlePanel("Data"),

              # Create a new Row in the UI for Twitter Data
              fluidRow(
                DT::dataTableOutput("table")
              )

            )
    ),
```

At the server side, I have read the data from file tweet_df_ds.rds and render data table to UI.

```
  #=== Read Data ===
  tweet_df_ds <- readRDS(file = "tweet_df_ds.rds")

  tweet_df_final <- rbind(
    cbind(tweet_df_ds, topic = "Donald Trump")
    )

  #  data display
  output$table <- DT::renderDataTable(DT::datatable({
    data <-
      tweet_df_final
  }))
```

Data is display into shiny app like below:



Data

Show 10 entries Search:

	user_id	status_id	created_at	screen_name	text	source	display_text_width	reply_to_status_id	reply_to
1	374729426	1192295744369070081	2019-11-07T04:20:31Z	Tickeron	$GRBK in Downtrend: Stochastic indicator ascends into oversold zone View odds for this and other indicators: https://t.co/ztgLQ9B4ca #GreenBrickPartners #stockmarket #stock #technicalanalysis #money #trading #investing #daytrading #news #today https://t.co/oOR3sIWgZr	Tickeron	243		
2	374729426	1192193145171148800	2019-11-06T21:32:50Z	Tickeron	$INSM in Downtrend: its price expected to drop as it breaks its higher Bollinger Band on November 1, 2019. View odds for this and other indicators: https://t.co/Pp6dIgR6ek #Insmed #stockmarket #stock	Tickeron	270		

Step 7.2: Word Cloud and Token Frequencies

Second Tab is for word cloud and Token Frequencies visualization.

Word Cloud

First, I am going to make some clouds of words, a great way to see the most common words use in this dataset. wordcloud2 provides conventional wordcloud applications with a HTML5 interface, along with some great feature such as personalised clouds and letter clouds.

Token Frequencies (Word Frequencies)

Given the amount of tweets, term-document matrix can become huge and you may encounter size allocation errors in R while performing operations on it. slam package comes to the rescue which performs these operations smoothly on sparse matrices. I regret not knowing the power of tidy text, a beautiful package developed by Julia Silge and David Robinson to do text processing much more elegantly. Since the size of the words will reflect their frequency, I'll remove the topic (twitter search term) from each tweet first; otherwise these would dominate the word clouds.

UI Code Step 7.2:

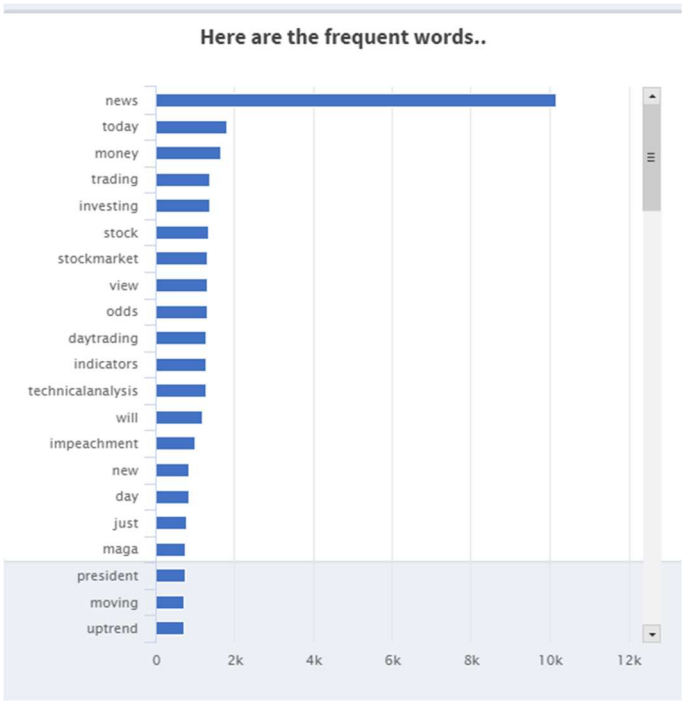
```
# Create a new  tab for Word of cloud and Frequencies of word
tabItem("token",
        fluidPage(
          fluidRow(
            column(7, id = "col_word_cloud",
                   box(width=10, height=450, solidHeader = F, title = strong("The Word Cloud"),
                       radioButtons("word_cloud_gram",NULL, c("Uni-gram","Bi-gram"), selected = "Uni-gram", inline = T
                       #plotOutput("word_cloud_plot",height = "300px")
                       wordcloud2Output("word_cloud_plot",height = 500))
            ),
            column(5, id = "col_freq",
                   box(width=12, height=450, solidHeader = F, title = strong("Here are the frequent words.."),
                       highchartOutput("word_freq_plot", height=500)
                   )

            )
          )
        )
),
```

Server Code:

```
output$word_cloud_plot <- renderWordcloud2({

  if(input$word_cloud_gram == "Uni-gram"){

    set.seed(1234)

    d1 <- (d() %>% filter(freq>1) %>% arrange(desc(freq)))[1:100,]
    wordcloud2(data = d1, size=0.8, minSize = 0.0, fontWeight = 'bold',
               ellipticity = 0.65)

  } else if(input$word_cloud_gram == "Bi-gram"){

    progress <- shiny::Progress$new()
    progress$set(message = "Bi-gram", value = 0)
    on.exit(progress$close())

    progress$set(value = 0.3, detail = "Creating Bitokens...")
    bitoken <- data.frame(text = sapply(docs(), as.character), stringsAsFactors = FALSE) %>%
      unnest_tokens(bigram, text, token = "ngrams", n = 2)
    two_word <- bitoken %>% count(bigram, sort = TRUE)
    progress$set(value = 0.9, detail = paste("Parsing a dataframe..Almost done"))
    sort_two <- two_word[order(two_word$n,decreasing=TRUE),]
    names(sort_two) <- c("word", "freq")
    d1 <- (sort_two %>% filter(freq>2) %>% arrange(desc(freq)))[1:50,]
    wordcloud2(data = d1, size=0.8, minSize = 0.0, fontWeight = 'bold',
               ellipticity = 0.65)


  }

})
```

```
#=== Word Freq plot ===
output$word_freq_plot <- renderHighchart(
  hc <- highchart() %>%

    hc_chart(type = "bar") %>%

    hc_tooltip(crosshairs = TRUE, shared = FALSE,useHTML=TRUE,
               formatter = JS(paste0("function() {
                                     //console.log(this);
                                     //console.log(this.point.y);
                                     var result='';
                                     result='<br/><span style=\\'color:'+this.series.color+'\\'>'+this.x.name+'</span>:<b
                                     +Math.round(this.point.y.toFixed(0)/100)/10 + 'K' + '</b>';
                                     return result;
    }"))) %>%
    hc_xAxis(categories = d()[1:100,]$word,

             labels = list(style = list(fontSize= '11px')), max=20, scrollbar = list(enabled = T)
    )    %>%
    hc_add_series(name="Word", data = d()[1:100,]$freq, type ="column",

                  color = "#4472c4", showInLegend= F)

)
```

Word cloud and Token frequencies output:



Step 7.3: Sentiment Analysis

There are three widely used sentiment lexicons based on unigrams.

1) NRC - Used for Emotion Radar Display. Get the emotions scores across all the tweets.
2) AFINN – Used for Sentiment Polarity Display. It's identified +ve, -ve and neutral tweets in data set.
3) Bing – Used for Differentiate positive and negative tweets.

UI code:

```
# Create a new  tab for sentiment radar, polarity chart, and postive and negative tweet
tabItem("sentiment",
        fluidPage(
          fluidRow(
            column(width = 6, id = "col_emotion",
                   box(width=NULL, height=550, solidHeader = F, title = strong("Emotions Radar"),
                       highchartOutput("emotion_polar_plot",height=500)
                   )
            ),
            column(width = 6,
                   box(width=NULL, height=270, solidHeader = F, title = strong("Sentiment Polarity"),
                       highchartOutput("sentiment_plot",height = 210)
                   ),
                   box(width=NULL, height=270, solidHeader = F, title = strong("Most Positive/Negative Tweet
                       htmlOutput("pos_tweet"),
                       htmlOutput("neg_tweet")
                   )
            )
          )
        )

),
```

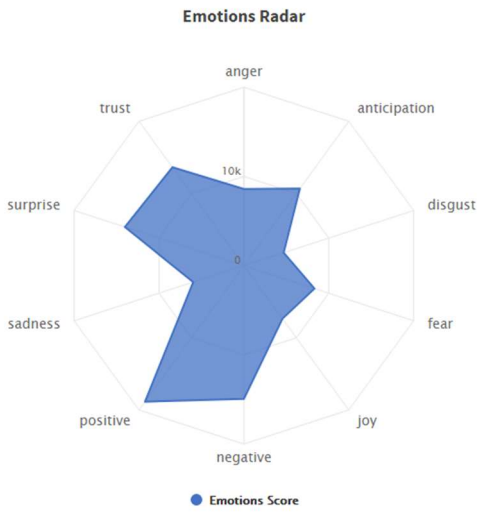
Emotion Radar

NRC — The NRC Emotion Lexicon is a list of English words and their associations with eight basic emotions (anger, fear, anticipation, trust, surprise, sadness, joy, and disgust) and two sentiments (negative and positive). The annotations were manually done by crowdsourcing.

Server Code:

```
emotion_score <- reactive({

  progress <- shiny::Progress$new()
  progress$set(message = "Getting Emotions", value = 0.2)
  on.exit(progress$close())

  x <- data.frame(tweet_nbr = 1:length(cleaned_tweets()), clean_tweet = cleaned_tweets())
  x$clean_tweet <- as.character(x$clean_tweet)

  df <- x  %>% unnest_tokens(output = word, input = clean_tweet, token = "words")
  df <- df %>% inner_join(get_sentiments("nrc"))
  progress$set(value = 0.8, detail = paste("COllating.."))
  df <- df %>% group_by(sentiment) %>% summarise(score = n())
  names(df) <- c("emotion", "score")
  df[c(1:8,9:10),]
  #df
})
```

```
## Emotion  Radar Chart Display

output$emotion_polar_plot <- renderHighchart(
  hc <- highchart() %>%

    hc_chart(polar = T) %>%
    hc_xAxis(categories = emotion_score()$emotion,
             labels = list(style = list(fontSize= '14px')), title =NULL, tickmarkPlacement = "on", lineWidth = 0) %>%
    hc_plotOptions(series = list(marker = list(enabled = F))) %>%
    hc_yAxis(gridLineInterpolation = "polygon", lineWidth = 0, min = 0) %>%
    hc_add_series(name = "Emotions Score", emotion_score()$score, type ="area", color = "#4472c4", pointPlacement = "on")
)
```



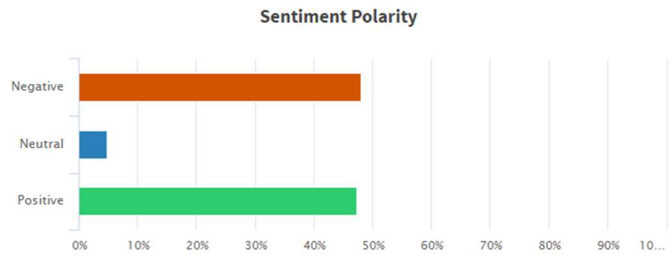
Sentiment Polarity

AFINN — AFINN is a list of English words rated for valence with an integer between -5 (negative) and +5 (positive). The words have been manually labelled by Finn Arup Nielsen in 2009–2011.

Server code:

```
#==== Sentiment Score Calcution======

sentiment_score <- reactive({
  progress <- shiny::Progress$new()
  progress$set(message = "Working on Sentiment", value = 0)
  on.exit(progress$close())

  x <- data.frame(tweet_nbr = 1:length(cleaned_tweets()), clean_tweet = cleaned_tweets())
  x$clean_tweet <- as.character(x$clean_tweet)

  progress$set(detail = "Getting score...", value = 0.6)
  df <- x  %>% unnest_tokens(output = word, input = clean_tweet, token = "words")
  df <- df %>% inner_join(get_sentiments("afinn"))
  df <- df %>% group_by(tweet_nbr) %>% summarize(score = sum(value))

  progress$set(detail = "Getting score...", value = 0.8)
  df$category_senti <- ifelse(df$score < 0, "Negative", ifelse(df$score > 0, "Positive", "Neutral"))
  df1 <- df %>% left_join(x)

  x <- list()
  x[[1]] <- as.data.frame(df1)
  x[[2]] <- as.character(df1[df1$score == max(df1$score),"clean_tweet"][1,1])
  x[[3]] <- as.character(df1[df1$score == min(df1$score),"clean_tweet"][1,1])

  x
})

senti_df <- reactive({
  sentiment_score()[[1]] %>% group_by(category_senti) %>% summarise(score = n()) %>%
    mutate(score_pct = score/sum(score)*100, coloract = c("#d35400", "#2980b9", "#2ecc71"))
})
```

```
#====== Sentiment Polarity dispaly====
output$sentiment_plot <- renderHighchart(

  hc <- highchart() %>%

    hc_chart(type = "bar") %>%

    hc_tooltip(crosshairs = TRUE, shared = FALSE,useHTML=TRUE,
               formatter = JS(paste0("function() {
                                     console.log(this.point.y);
                                     var result='';
                                     result='<br/><span style=\\'color:'+this.series.color+'\\'>'+this.series.name+'</span>:<b> '+Math.round(this.point.y
                                     return result;
                 }")))%>%
    hc_xAxis(categories = senti_df()$category_senti,

             labels = list(style = list(fontSize= '12px')) #max=20, scrollbar = list(enabled = T)
    )    %>%
    hc_colors(colors = senti_df()$coloract) %>%
    hc_add_series(name="Sentiment", data = senti_df()$score_pct, colorByPoint = TRUE,
                  type ="column",

                  color = "#4472c4", showInLegend= F) %>%
    hc_yAxis(labels=list(format = '{value}%'),min=0,
             max=100,showFirstLabel = TRUE,showLastLabel=TRUE)

)
```

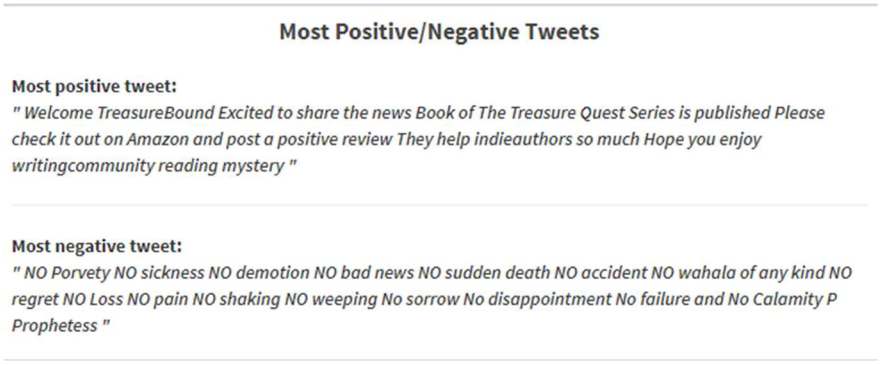


Most Positive/Negative Tweets

Bing — It categorises tweets into positive and negative elements wise representation.

Server Code:

```
output$pos_tweet <- renderUI({
  HTML(paste("<b>","Most positive tweet: ","</b>"),"<br>","<i style = 'font-weight: bold'>","\"",sentiment_score()[[2]][1],"\"","</i>","<hr>
})
output$neg_tweet <- renderText({
  HTML(paste("<b>","Most negative tweet: ","</b>"),"<br>","<i style = 'font-weight: bold'>","\"",sentiment_score()[[3]][1],"\"","</i>")
})
```



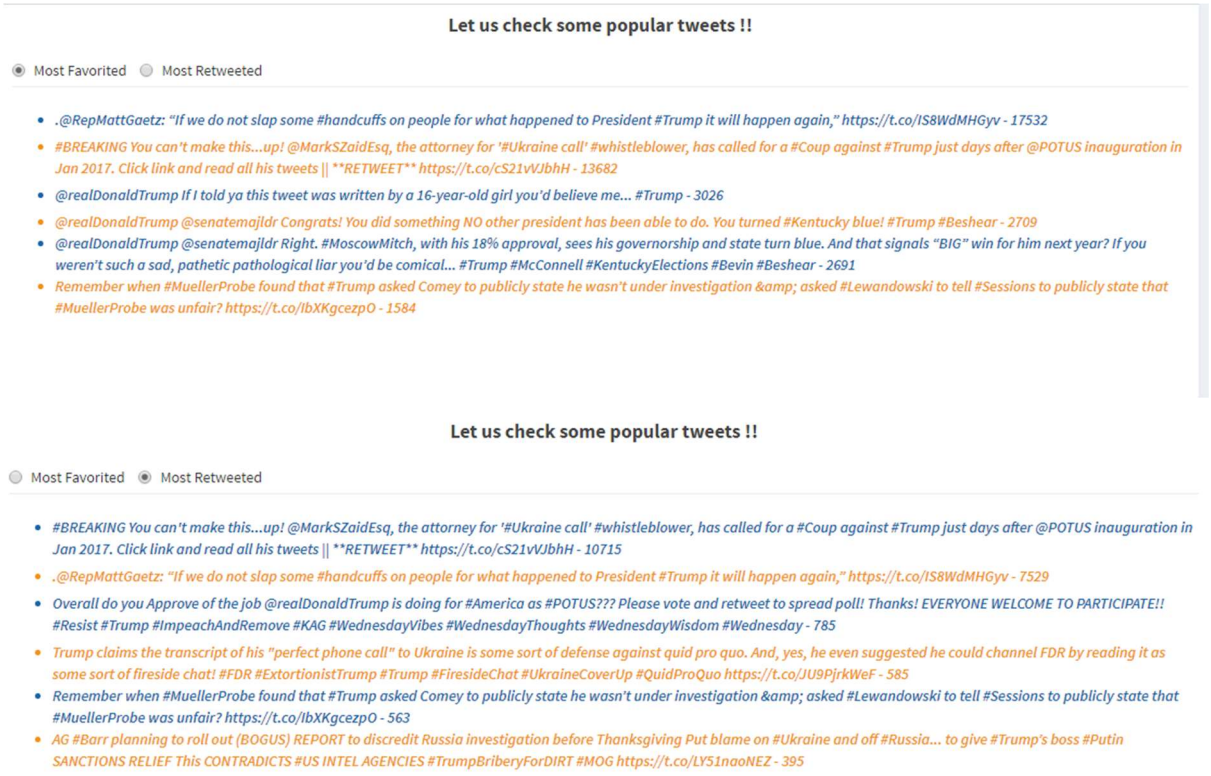
Step 7.4: Popular Tweets

In sentiment analysis of data set, we can also fine some top tweets in those topics based on retweets and favourites.
UI Code:

```
tabItem("popular",
        fluidPage(
          fluidRow(
            box(width=12, height=370, title = strong("Let us check some popular tweets !!"),
                radioButtons("fav_rt_button",NULL, c("Most Favorited","Most Retweeted"), selected = "Most Favorited", inline = T),hr()
                htmlOutput("fav_rt_tweets")),

          )
        )
)
```

Server Code:

```
#==== Popular Tweets=====
 fav_rt_tweets <- reactive({

   tweets_df <- tweet_df_final %>% filter(topic == topic_selector)
   x <- list()
   x[[1]] <- head(tweets_df %>% select(text, favorite_count) %>% arrange(desc(favorite_count)))
   x[[2]] <- head(tweets_df %>% select(text, retweet_count) %>% arrange(desc(retweet_count)))
   x
 })

 output$fav_rt_tweets <- renderUI({
   if(input$fav_rt_button == "Most Favorited"){
     tags$ul(
       tags$li(tags$i(paste(fav_rt_tweets()[[1]][1,]$text,"-",paste(fav_rt_tweets()[[1]][1,]$favorite_count))), style = "margin-bottom: 5px; color: #1D65A6;
       tags$li(tags$i(paste(fav_rt_tweets()[[1]][2,]$text,"-",paste(fav_rt_tweets()[[1]][2,]$favorite_count))), style = "margin-bottom: 5px; color: #F1931B;
       tags$li(tags$i(paste(fav_rt_tweets()[[1]][3,]$text,"-",paste(fav_rt_tweets()[[1]][3,]$favorite_count))), style = "margin-bottom: 5px; color: #1D65A6;
       tags$li(tags$i(paste(fav_rt_tweets()[[1]][4,]$text,"-",paste(fav_rt_tweets()[[1]][4,]$favorite_count))), style = "margin-bottom: 0px; color: #F1931B;
       tags$li(tags$i(paste(fav_rt_tweets()[[1]][5,]$text,"-",paste(fav_rt_tweets()[[1]][5,]$favorite_count))), style = "margin-bottom: 0px; color: #1D65A6;
       tags$li(tags$i(paste(fav_rt_tweets()[[1]][6,]$text,"-",paste(fav_rt_tweets()[[1]][6,]$favorite_count))), style = "margin-bottom: 0px; color: #F1931B;
     )
   } else if(input$fav_rt_button == "Most Retweeted"){
     tags$ul(
       tags$li(tags$i(paste(fav_rt_tweets()[[2]][1,]$text,"-",paste(fav_rt_tweets()[[2]][1,]$retweet_count))), style = "margin-bottom: 5px; color: #1D65A6;
       tags$li(tags$i(paste(fav_rt_tweets()[[2]][2,]$text,"-",paste(fav_rt_tweets()[[2]][2,]$retweet_count))), style = "margin-bottom: 5px; color: #F1931B;
       tags$li(tags$i(paste(fav_rt_tweets()[[2]][3,]$text,"-",paste(fav_rt_tweets()[[2]][3,]$retweet_count))), style = "margin-bottom: 5px; color: #1D65A6;
       tags$li(tags$i(paste(fav_rt_tweets()[[2]][4,]$text,"-",paste(fav_rt_tweets()[[2]][4,]$retweet_count))), style = "margin-bottom: 0px; color: #F1931B;
       tags$li(tags$i(paste(fav_rt_tweets()[[2]][5,]$text,"-",paste(fav_rt_tweets()[[2]][5,]$retweet_count))), style = "margin-bottom: 0px; color: #1D65A6;
       tags$li(tags$i(paste(fav_rt_tweets()[[2]][6,]$text,"-",paste(fav_rt_tweets()[[2]][6,]$retweet_count))), style = "margin-bottom: 0px; color: #F1931B;

     )
   }

 })
```



Step 7.5: Emotion Percentage Representation

Another way to represents Emotions via Percentage Table and Pie Chart.

```
# Create a new  tab for emotion table and chart
tabItem("emotionalpercentages",
        fluidPage(

          # Create a new Row in the UI for Emotion table and Chart
          fluidRow(
            column(7, id = "emotionpertable",
                   box(width=12, height=450, solidHeader = F, title = strong("Emotion Pecentage Table"),
                       DT::dataTableOutput("emotionpertable"))
            ),
            column(5, id = "emotionperchart",
                   box(width=12, height=450, solidHeader = F, title = strong("Emotion PIE Chart"),
                       plotOutput("distPie") )

            )
          )

        )
),

# emotion pecentage display
output$emotionpertable<-DT::renderDataTable(DT::datatable({
  withProgress(message = 'Calculating Emotional Sentiment',
               value = 0, {
                 for (i in 1:3) {
                   incProgress(1/3)
                   Sys.sleep(0.25)
                 }
               },env = parent.frame(n=1))

  value <- emotion_df_ds

  prop.table(value[,1:8])

  sentimentscores <- round(colSums(prop.table((value[,1:8])))*100,digits = 1)

  sentimentscores <- as.data.frame(sentimentscores)
  colnames(sentimentscores) <- c("Percentages")

  Emotions <- c("anger","anticipation","disgust","fear","joy","sadness",
                "surprise","trust")

  Percentages<- sentimentscores$Percentages
  emotionality<- cbind(Emotions,Percentages)
  emotionality

}))
```

```
#Percentage Pie Chart

output$distPie <- renderPlot({
  value <- emotion_df_ds

  prop.table(value[,1:8])

  sentimentscores <- round(colSums(prop.table((value[,1:8])))*100,digits = 1)

  sentimentscores <- as.data.frame(sentimentscores)
  colnames(sentimentscores) <- c("Percentages")

  slices <- sentimentscores$Percentages

  labels <- c("anger","anticipation","disgust","fear","joy","sadness",
              "surprise","trust")

  lbls <- paste(labels, slices) # add percents to labels
  lbls <- paste(lbls,"%",sep="") # ad % to labels
  pie3D(slices,labels=lbls,explode=0.1,radius = 1.5
        ,main="")
})
```

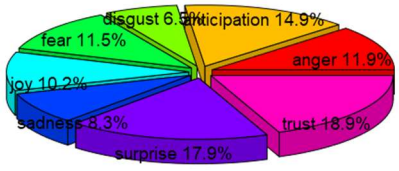
Emotion Pecentage Table

Show 10 entries Search:

Emotions	Percentages
anger	11.9
anticipation	14.9
disgust	6.5
fear	11.5
joy	10.2
sadness	8.3
surprise	17.9
trust	18.9

Showing 1 to 8 of 8 entries Previous 1 Next

Emotion PIE Chart



Step 7.6: Sentiment Data Representation

This is simple data table rendering in UI side. First, I have created new tab into dashboard body and inside this I have shown the data into table format using the function dataTableOutput.

UI Code:

```
# Create a new  tab for sentiment data
tabItem("sdata",
        fluidPage(
          titlePanel("Sentiment Data"),

          # Create a new Row in the UI for Sentiment Data
          fluidRow(
            DT::dataTableOutput("stable")
          )

        )
),
```

At the server side, I have read the data from file emotion_df_ds.rds and render data table to UI.

```
#===== Sentiment Analysis ======
emotion_df_ds <- readRDS(file = "emotion_df_ds.rds")

emotion_df_final <- rbind(
  cbind(emotion_df_ds, topic = "Donald Trump")


)

 #  Sentiment data display
output$stable <- DT::renderDataTable(DT::datatable({
  data <- emotion_df_final
}))
```

Sentiment Data

Show 10 entries Search:

	anger	anticipation	disgust	fear	joy	sadness	surprise	trust	negative	positive	topic
1	1	1	0	0	1	0	1	1	0	1	Donald Trump
2	1	2	0	0	1	0	1	1	0	1	Donald Trump
3	1	1	0	0	1	0	1	1	0	1	Donald Trump
4	1	1	0	1	1	1	1	1	1	1	Donald Trump
5	1	1	0	0	1	0	1	1	0	1	Donald Trump
6	1	1	0	1	1	1	1	1	1	1	Donald Trump
7	1	1	0	0	1	0	1	1	0	1	Donald Trump
8	1	1	0	0	1	0	1	1	0	1	Donald Trump
9	1	1	0	0	1	0	1	1	0	1	Donald Trump
10	1	1	0	0	1	0	1	1	0	1	Donald Trump

Showing 1 to 10 of 17,189 entries Previous 1 2 3 4 5 … 1719 Next

Step 7.7: Sentiment Map

There's good reason to believe that residents of certain states hold particularly positive or negative sentiments towards politicians. The Twitter API provides longitudes and latitudes for most of the scraped tweets. But we want to convert these coordinates to US states. One way to reverse geocode coordinates is by querying the Google Maps API. The package ggmap interfaces with the Google Maps API, and it makes reverse geocoding the coordinates very easy. Note however that the Google Maps API only allows 2500 queries per day, and the returns are slow. Another way is with geonames.org.

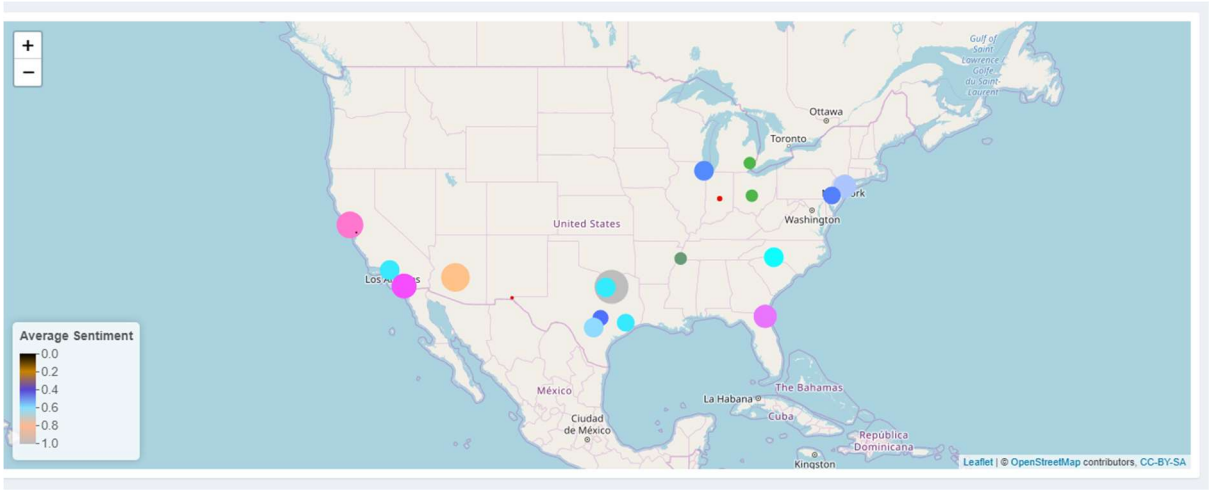
Create the map of by-state sentiment. If avg sentiment is <0.5 then it's indicating negativity in tweets else if 0.5 is neutral and else >0.5 is indicates positive response in tweets.

UI code:

```
# Create a new  tab for sentiment map
tabItem("map",
        fluidPage(

          # Create a new Row in the UI for Sentiment Map
          fluidRow(
            column(width = 12,
                   box(width = NULL, solidHeader = TRUE,
                       leafletOutput("tweet_sentiment", height = 500)
                   )#,
                   #box(width = NULL,
                   #     dataTableOutput("top_tweets")
                   # )
            )
          )
        )
),
```

Server Code:

```
# Seniment Map Display

output$tweet_sentiment <- renderLeaflet({

  progress <- shiny::Progress$new()
  progress$set(message = "Preparing Sentiment Map", value = 0)
  on.exit(progress$close())

  sentimentmap_df_ds <- readRDS(file = "sentimentmap_df_ds.rds")

  locations <- sentimentmap_df_ds


  from_min <- min(locations$avg_sent)
  to_min <- 0

  from_max <- max(locations$avg_sent)
  to_max <- 1

  z <- (locations$avg_sent - from_min) * (to_max - to_min) / (from_max - from_min) + to_min
  tbl <- bind_cols(locations %>% select(city_name, pos, neg, avg_sent))

  pal <- colorNumeric(
    palette = palette(c("#F54C54", "blue")),
    domain = z
  )
  map <- leaflet(locations) %>%
    addTiles() %>%
    addCircles(radius = z*(10^5.2), stroke = FALSE,  fillOpacity = 1,
               popup = paste("City:", locations$city_name, "<br>",
                             "Sentiment:", round(locations$avg_sent,4)),  color = ~pal(z))  %>%

    leaflet::addLegend(position = "bottomleft", pal = pal, values = ~z, opacity = 1,
              title = "Average Sentiment")

  map

})
```



Summary

In this report, I have completed the tasks how to integrate R and R shiny for text mining, sentiment analysis and visualization. Using these tools together enables us to answer detailed questions.

We used a sample from the most recent tweets that contain Donald Trump and since I was not able to reverse geocode all the tweets, I scraped because of the constraint imposed by Shiny maps API, we just used about 18000 tweets. The average sentiment is slightly above zero. Some states show strong positive sentiment. However, statistically speaking, to make robust conclusions, mining ample size sample data is important.

The accuracy of our sentiment analysis depends on how fully the words in the tweets are included in the lexicon. Moreover, since tweets may contain slang, jargon and colloquial words which may not be included in the lexicon, sentiment analysis needs careful evaluation.