# The Role of Sentiment Analysis in Forecasting Successful ICOs

**Document:**

R Markdown Report

**Author:**

Ettore Falde

**Director/Co-director:**

Prof. Jose Maria Sallan Leyes

**Degree:**

Master in IT Management Engineering

**Examination session:**

Autumn Extension year 2023

MASTER FINAL THESIS

# Contents

# 1 Introduction

Over the past few years, rose the necessity to have untraceable payment methodologies that will allow people to buy and sell products and goods via not traceable coins. Therefore, since Bitcoin was released, the rise of blockchain technology was crucial in changing the landscape of the financial industry. Indeed, a new approach to raise funds for startups directly from VCs was being developed. This method, known as ICOs, involves the creation and sale of homemade cryptocurrencies or tokens to online investors. By using this method, startups can obtain financing without the need for intermediaries in exchange for popular cryptocurrencies or fiat currency.

Nonetheless, it is extremely difficult for investors to evaluate the chances of businesses obtaining money through initial coin offerings ICOs because of huge information asymmetries. This is especially important when taking into account the meager incentives available to small investors in the world of digital finance to do due diligence.

Previous studies show that the ICOs market has become a popular fundraising tool for startups, but it needs statistical methodologies to help investors make better investments and help them through the process of decision-making, without failing in some scam. In fact, we can define ICO as "an open call for funding promoted by organizations, companies, and entrepreneurs to raise money through cryptocurrencies, in exchange for a token that can be sold on the Internet or used in the future to obtain products or services and, at times, profits". Hence, a data-driven approach to investment decision-making can provide effective results, thanks to the usage of ML methodologies and traditional statistical models such linear regression and logistic regression.

In addition, also social media evaluation through SA is crucial as a recommendation system that will help to forecast the purchases of goods and services by users or investors. Then, a social media that is commonly used for this type of analysis is Twitter. Twitter data offers a wealth of market-impacting information that may be used to derive emotional intelligence through SA. The studies by *Bollen* were the best illustrations of the use and effectiveness of Twitter sentiment analysis to forecast financial markets, according to related research. Despite the study by *Bollen* having a fantastic stock market predicting record, it has received harsh criticism for using false statistical assumptions. Therefore, according to new researches, it was possible to predict the price returns of the major cryptocurrencies using sentiment analysis from Twitter.

Consequently, this master's project seeks to add to this body of literature by putting forth a novel approach to estimating the chance of a successful ICO based on SA in conjunction with conventional statistical models and machine learning models.

Fortunately, the ICO whitepaper, a document outlining the project, is a trustworthy source of information on which investors base their investment selections. In any case, the whitepaper if it's not well explained and the project doesn't show a high degree of transparency seems not to work. Hence, investors should find it necessary to highlight also the opinion of the stakeholders, and SA can fulfill this request. Luckily, part of the data needed for this project was retrieved from the database TORD. While the other part was gathered by a script using the package *snscrape*.

Thus, for the Twitter dataset to run the SA I used Python in conjunction with NLP, text analysis, and ML models that can automate the process of classification of each tweet. One of those models is VADER SA, which largely makes use of a lexicon that turns emojis and linguistic traits into sentiment scores, which represent the intensity of emotion. Moreover, VADER is also a specialized library to perform better in social media environments. While, the other is TextBlob, a Python library that provides a simple API for NLP tasks such as part-of-speech tagging, noun phrase extraction, and SA.

Consequently, the data for both the TORD database and my Twitter dataset had to be cleaned, and new variables were generated. Such as the binary variable that represents if and ICO will be "successful" (1) or "non-successful" (0). In doing this, differently than what has been done by Meoli Michele, I set the new variable that determines if an ICO will be successful, only if it can achieve a minimum investment of $500.000 (USD dollars).

Then, I divided the analysis between exploratory (or visual) and forecasting. Visual analysis used graphs like *bar*, *col*, or map plots. Thanks to the *ggplot*, *ggmap*, and *terra* packages (for R programming language). Indeed, this section is crucial because the use of exploratory analysis allows for an efficient and effective way to communicate complex data and findings to a wide audience. Finally, the predictive analysis was done for both the Twitter dataset and the TORD database with the results of the SA achieved from the Twitter dataset. Accordingly, I used various techniques such as logistic regression, random forests, GBM, and DRF, and other ML models to demonstrate how and in which terms the results of SA can predict the success of the ICOs under analysis. Additionally, I compared the results of the different models to understand the better-performing one, and contribute to the extant literature.

In conclusion, thanks to the explanatory, and especially to the forecasting analysis I was able to predict the success of ICOs. Understanding that SA alone cannot significantly contribute to the prediction of successful ICOs. Therefore, the best model has obtained from the automated ML H2O technique, which achieved a maximum f1-score of 85.53%. Then, this study also concluded that having a high-quality dataset can lead to better forecasting results, and the importance of having more relevant variables, such as the quality of the whitepaper. Which was highlighted as a crucial factor in determining the success of ICOs. To conclude, the analysis also revealed that the use of Twitter data can be a lengthy and complicated process, and using packages like *sncrape* may retrieve inaccurate data.

# 2 Context and theoretical framework

Nowadays, the concept of privacy and finance are trending arguments, especially when applied to new technologies. Hence, the starting point of modern finance derives from the common possibility of doing payments with our cards, which was considered in the past as a common social decision that must involve faith and risks. Indeed, cash compared to credit cards has a significant advantage in terms of privacy and tractability. This led to different research, intending to create and adopt a reliable methodology based on untraceable electronic cash that will also solve the double spending fraud. Therefore, rose the necessity of having untraceable coins. So, the first cryptocurrency that was published, was Bitcoin. The author, Satoshi Nakamoto, was the creator of Bitcoin and the father of the modern blockchain.

Consequently, the rise of blockchain was crucial in changing the landscape of the financial industry. Generating new methodologies for startups to gather funds directly from VCs. So, this new form of entrepreneurial finance called ICOs (Initial Coin Offering) leverages blockchain technology to sell homemade cryptocurrencies or tokens.

To sum up, in these years of rapid changes and the rise of new technologies, blockchain, and cryptocurrency transformed the landscape of finance. Indeed, these new trends have seen the rise of new methodologies of entrepreneurial finance such as ICOs.

## 2.1 Blockchain

Blockchain is a decentralized digital ledger that securely and openly logs transactions. Although it was initially developed to support the virtual currency Bitcoin, it has since been used in a wide variety of different situations.

A blockchain is fundamentally a distributed database with a list of documents called blocks that is constantly expanding. Each block has a timestamp, a cryptographic hash of the preceding block, and several fresh transactions. This results in a chain of blocks that cannot be changed and is impervious to manipulation.

A network of nodes, or computers on the network, first verifies a new transaction before adding it to the blockchain. The transaction is broadcast to the network and included in a candidate block after it has been confirmed. The first node to solve the problem is rewarded with brand-new cryptocurrency coins or tokens. The nodes then compete to solve a challenging mathematical problem that demands a lot of computer power. The blockchain is updated with new blocks at a set rate thanks to a process known as mining.

A block is considered final and cannot be changed once it has been added to the blockchain. This makes the blockchain the perfect instrument for logging and validating transactions across a variety of sectors, from logistics and supply chain management to finance and healthcare.

The blockchain has security safeguards, but it's also open and decentralized. This indicates that no central authority or middleman is required for anyone to examine the complete blockchain and verify transactions. Because of this, a blockchain is a desirable tool for businesses and individuals who value decentralization, security, and transparency.

## 2.2 Cryptocurrencies and Exchanges

### 2.2.1 Cryptocurrencies

Digital or virtual currencies that use cryptography for security and to deter counterfeiting are known as cryptocurrencies. They are decentralized and run without the help of a centralized institution like a government or bank. Using blockchain technology, which is a decentralized ledger that keeps track of all transactions, cryptocurrencies are stored and exchanged. The first cryptocurrency, Bitcoin, was developed in 2009 by an unidentified individual or group operating under the pseudonym Satoshi Nakamoto. With no need for middlemen, the peer-to-peer electronic cash system known as Bitcoin was created to enable secure, private transactions. Several alternative cryptocurrencies, often known as altcoins, have been developed since then.

Complex mathematical algorithms are used to originate and verify transactions in cryptocurrencies. Every transaction is recorded on the blockchain, a global network of computers that serves as a public ledger. Together, these computers validate and verify each transaction to make sure it is safe and accurate. One starts a transaction using their digital wallet when they want to transmit cryptocurrency to someone else. The network of computers receives the broadcast of this transaction and collaborates to verify it before adding it to the blockchain. The transaction is regarded as finished and irreversible once it has been confirmed and posted to the blockchain.

Cryptocurrencies are often seen as a way to provide more financial freedom and security to individuals, as they are not subject to government or bank regulations. However, they can also be volatile and subject to extreme price fluctuations.

### 2.2.2 Exchanges

A crypto exchange is a platform where people can trade cryptocurrencies, with some exchanges also offering price discovery and crypto storage, among other services. Previously, people obtained cryptocurrencies through mining or by organizing transactions in various online and offline forums.

Modern exchanges typically have three components: price and discovery mechanism, algorithmic trade matching engine, and trade clearing system. These three factors determine the degree of centralization of an exchange. There are various DeFi exchanges, including those based on blockchain architecture (DEX) or those implementing TEE and MPC. In contrast, CeFi works as a middleman between buyers and sellers, allowing users to trade both fiat and cryptocurrencies. Table 2 shows the main differences between these types of exchanges.

To summarize, there are advantages and disadvantages to both types of exchanges.

- Advantages of CEX: User interfaces are generally more user-friendly for beginners. Access to a variety of cryptocurrency trading platforms with a high degree of functionalities and trading options. Greater cash flow due to its ease of use and larger user base. Enables fast, real-time transactions and supports fiat transactions.
- Disadvantages of CEX: Single point of failure which makes it vulnerable to cyber attacks and billion-dollar losses. External parties and legal teams have control over centralized trades. Contradictory to the basic idea and purposes of cryptocurrencies as users have to provide their data to prevent financial fraud.
- Advantages of DEX: No restrictions of control by authorities as participants trade directly with each other. Each client has full control over their private keys and crypto assets. No identification process, so no user data is collected. Reduced possibility of system breaches and low transaction costs.
- Disadvantages of DEX: Low speed and efficiency due to operating with smart contracts and possible congestion. Relatively lower liquidity compared to CEX. Complex user interface for beginners, no access to limit orders, margin transactions, or stop losses. Only cryptocurrencies are allowed, and no fiat currencies are present.

## 2.3 ICOs

Firstly, it is important to define what an ICO is. An ICO is a decentralized method of raising capital, in which a business seeks funding by distributing their coins/tokens (crypto) to online investors. This can be considered a form of entrepreneurial finance, where startup companies sell their cryptocurrencies to raise funds in exchange for other popular cryptocurrencies or fiat currencies to develop projects, usually in the field of blockchain. An ICO typically includes features such as a white paper, a proposer team, a target sum to be collected, and a specific number of tokens or cryptocurrencies to share with clients through an exchange. However, many ICOs have been unmasked as Ponzi schemes or scams, and the United States SEC has warned investors about the risks of investing in these schemes.

When a cryptocurrency project organizer wants to raise money through an ICO, they need to determine the structure of the coin, which can be static supply and static price, static supply and dynamic price, or dynamic supply and static price. The company sets a specific funding goal in the first case, and each coin sold in the ICO has a preset price, with the total token supply fixed. In the second case, the total amount of funds received during the ICO determines the overall price per token, with the ICO proposed with a static supply of tokens and a dynamic funding goal. In the third case, the company wants to have a dynamic token supply with a static price for each token, with the amount of funding received determining the supply.

Alongside structuring the coin, the crypto-project creates a white paper, which is a simple text document providing information about the problem the project is going to solve, as well as the solution, a detailed description of the company's final product, its architecture, and how users will use their solution. Any white paper should include an introduction, disclaimer, table of contents, description of the market and the problem, description of the product and how it's going to solve said problem, tokens information, how the raised funds are going to be used, the team, and the roadmap.

The business seeking funding will release the white paper as part of the ICO campaign. The performance of the ICO on the first day may be highly affected by the quality and transparency of all the information contained in the white paper. If an ICO reaches all the funds that were established initially as the final goal, it will be deemed successful. Otherwise, if it cannot reach the threshold established, all the funding will be returned to each investor, and the ICO will be considered unsuccessful.

### 2.3.1 Types of ICOs

As we previously mentioned, fraud and financial flaws were common in the early years of Initial Coin Offerings (ICOs) as an emerging crowdfunding method. Nowadays, ICOs have evolved into various forms that are challenging to define.

- ICO: is a direct offering managed by the company issuer, as shown in Figure 1, where we can observe the rise and fall of ICOs. Due to non-regulations and scams, the popularity of ICOs declined. To address this, different types of coin offerings were introduced.
- IEO: a similar method to ICO, aims to raise capital through online trading exchange platforms, which conduct the funding instead of the project team. This approach leads to faster purchases available directly through the exchange wallet. The use of exchanges also brings transparency and a high degree of trust.
- STO: was created in response to the bubble burst of ICOs in 2018, and it is an upgraded and regulated digital version of securities. It is a cheap way for startups and new businesses to raise funds for their projects. STO represents a digital representation of real-world assets, similar to certificates issued for stocks, and requires compliance with regulations by the campaign promoters.
- IDO: is an even better version of liquidity for crypto assets, offering faster, open, and fair trading. In IDO, the hosting exchange is a decentralized exchange (DEX), which allows businesses to attract their communities to enrich both products and services. The main difference between ICO and IDO is that tokens and cryptocurrencies are listed immediately on a DEX in IDO, whereas, in ICO, the tokens are sold before listing.
- IFO: is a funding model for new DeFi projects to raise capital through pre-sales events hosted after stern project vetting by DEX. It is considered the successor of ICO, and investors can raise the desired capital through resales events hosted by the DEX.
- ILO: enables liquidity contributors to receive tokens in exchange for their agreement to contribute to the liquidity pool. The DEX with automated market makers (AMM) facilitates the process of raising funds, which can take a long time in a traditional centralized exchange (CEX) with buyers and sellers. In ILO, liquidity pools, which are normally in the form of stablecoins, are managed by highly skilled liquidity providers.
- INO: is a marketing tool to attract customers by providing a piece of digital art to the buyer, which also comes with utility for the holder. It is a funding model and marketing tactic that enables the

creator of the art piece to collect data on the number of people attracted to it. Additionally, the owner of the non-fungible token (NFT) can participate in exclusive events or other advantages, depending on the NFT.

# 3 Gathering Data

## 3.1 Creating the initial dataset

In this initial stage I decided to use *snscrape*, a scraper for social network services. This Python library has no constraints compared to the offical Twitter APIs. So I was able to retrive all the data needed for this project.

Hence, after importing the *TORD* dataframe I did some basic cleaning removing duplicates values. To conclude, I coded a script that was able to look the ICO names from the *TORD* dataset, then for each ICO, check the campaign ending period and retrive the tweets from a logical date setted as the: 2010-01-01. This date, was 4 years after the launch of Twitter, and the first ICO project was launched in July 2013. Therefore, was impossible to have wrong tweets. Moreover, if the tweets where out of context, also the words searched (which are most of the times very recognizable) can have a general impact over the users and investors.

In the end, after some days of scraping, this code was actually able to retrive a very large dataset over 2GB of .csv file.

```python
import math
from dateutil.parser import parse
import snscrape.modules.twitter as sntwitter
import pandas as pd


# Importing my starting data set
tord = pd.DataFrame(pd.read_csv('../Datasets/tord_v3.csv', sep=';'))


# Doing some basic cleaning
tord = tord[tord['ico_end'].notna()]
tord = tord[tord['ico_start'].notna()]
tord = tord.loc[tord['token'].notna()]
tord = tord.loc[tord['name'].notna()]
tord.drop_duplicates(subset='name', keep=False, inplace=True)
# dropping ALL duplicate values
#tord = tord.loc[tord["token"] != "Solid" ]
print(tord)


#Trying to get the tweets

tweets = []
limits = 1000
index = 0


for i in range(len(tord)):
    print(index)
    #query = str(tord.name.iloc[i]) + " " + str(tord.token.iloc[i])  +
    #" -filter:replies lang:en" + " until:" + str(tord.ico_end.iloc[i]) +
    #" since:2010-01-01"
    query1 = str(tord.name.iloc[i]) + " -filter:replies lang:en" +
    " until:" + str(tord.ico_end.iloc[i]) + " since:2010-01-01"
    query2 = str(tord.token.iloc[i])  + " -filter:replies lang:en" +
    " until:" + str(tord.ico_end.iloc[i]) + " since:2010-01-01"
    print(query1, query2)

    temp = 0
    try:
        for tweet in sntwitter.TwitterSearchScraper(query1 or
                                                    query2).get_items():
            #print(vars(tweet))
            #break
            if temp == limits:
                break
            else:
                try:
                    tweets.append([tord.name.iloc[i],
                                   tord.token.iloc[i],
                                   tweet.user.username,
                                   tweet.user.verified,
                                   tweet.user.created,
                                   tweet.user.followersCount,
                                   tweet.user.friendsCount,
                                   tweet.retweetCount,
                                   tweet.lang,
                                   tweet.date,
                                   tweet.likeCount,
                                   tweet.sourceLabel,
                                   tweet.id,
                                   tweet.rawContent,
                                   tweet.hashtags,
                                   tweet.conversationId,
                                   tweet.inReplyToUser,
                                   tweet.coordinates,
                                   tweet.place,
                                   tweet.inReplyToTweetId])
                except:
                    print("can't append for some reasons ")

            temp += 1
    except:
        print("Some random error to get the tweets")
    index += 1

my_twitter_dataset = pd.DataFrame(tweets, columns=['ICO_name',
                                                   'Token_name',"User",
                                                   "verified",
                                                   "Date_Created",
                                                   "Follows_Count",
                                                   "Friends_Count",
                                                   "Retweet_Count",
                                                   "Language",
                                                   "Date_Tweet",
                                                   "Number_of_Likes",
                                                   "Source_of_Tweet",
                                                   "Tweet_Id",
                                                   "Tweet",
                                                   "Hashtags",
                                                   "Conversation_Id",
                                                   "In_reply_To",
                                                   "Coordinates",
                                                   "Place",
                                                   "is_reply_to_tid"])

print(my_twitter_dataset)

my_twitter_dataset.to_csv('my_twitter_data_set.csv')
#%%
```

# 4 Sentiment Analysis

At this point was obvious that the tweets needed to be pre-processed and I had to use some models to retrive the sentiment results.

## 4.1 Roberta Model

First of all, I decided to use this model, which seems to be the actual best model overall to accomplish a very precise sentiment analysis. Anyway, this AI model was really time consuming, and after some hours/days of running I accomplished a sentiment analysis results for only 100000 records. Therefore, I decided to give up with this model and continue my SA with other solutions.

In the following chunk of code is possible to see the python script used.

```
#%%
!pip install transformers
#%%
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from scipy.special import softmax
import pandas as pd
import numpy as np
import math
from google.colab import files
#%%
from google.colab import drive
drive.mount('/content/drive')
#%%
my_twitter_db = pd.DataFrame(pd.read_csv('my_twitter_data_set.csv', sep=','))
#%%
# See the columns names
list(my_twitter_db.columns.values)

# See the number of rows of my dataset
my_twitter_db.shape[0]
#%%
# Dataset cleaning to fasten up the results and avoid errors

# 1. Removing unwanted columns for the analysis
my_twitter_db = my_twitter_db.drop('is_reply_to_tid', axis=1)
my_twitter_db = my_twitter_db.drop('In_reply_To', axis=1)
my_twitter_db = my_twitter_db.drop('Conversation_Id', axis=1)
my_twitter_db = my_twitter_db.drop('Tweet_Id', axis=1)

# 2. Removing empy rows on tweets
my_twitter_db = my_twitter_db[my_twitter_db['Tweet'].notna()]

# 3. Creating a sub-dataset with only 100K tweets
my_twitter_db_100K = my_twitter_db.head(100000)

# 4. Add 3 new columns for the values
my_twitter_db_100K["Negative"] = np.nan
my_twitter_db_100K["Neutral"] = np.nan
my_twitter_db_100K["Positive"] = np.nan
#%%
# chcking again to see if there are changes and changes are correct
```

```
list(my_twitter_db.columns.values)
my_twitter_db.shape[0]
#%%
# load model and tokenizer
roberta = "cardiffnlp/twitter-roberta-base-sentiment"

model = AutoModelForSequenceClassification.from_pretrained(roberta)
tokenizer = AutoTokenizer.from_pretrained(roberta)

labels = ['Negative', 'Neutral', 'Positive']

for i in range(len(my_twitter_db_100K)):
    tweet = my_twitter_db_100K.iloc[i]['Tweet']
    tweet_words = []
    for word in tweet.split(' '):
        if word.startswith('@') and len(word) > 1:
            word = '@user'

        elif word.startswith('http'):
            word = "http"
        tweet_words.append(word)

    tweet_proc = " ".join(tweet_words)

    # sentiment analysis
    encoded_tweet = tokenizer(tweet_proc, return_tensors='pt')
    output = model(**encoded_tweet)

    scores = output[0][0].detach().numpy()
    scores = softmax(scores)

    roberta_results=[]

    for j in range(len(scores)):
        label = labels[j]
        s = scores[j]
        #print(label, s)
        roberta_results.append(s)
    print(roberta_results)

    my_twitter_db_100K.iloc[i, my_twitter_db_100K.columns.get_loc('Negative')]
    = roberta_results[0]

    my_twitter_db_100K.iloc[i, my_twitter_db_100K.columns.get_loc('Neutral')]
    = roberta_results[1]

    my_twitter_db_100K.iloc[i, my_twitter_db_100K.columns.get_loc('Positive')]
    = roberta_results[2]
#%%
my_twitter_db_100K.to_csv('Roberta_100K.csv')
files.download('Roberta_100K.csv')
```

## 4.2 Vader

*Vader*, a Valence Aware Dictionary and sentiment Reasoner, is defined as: “is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media, and works well on texts from other domains.”

Therefore, I decided to run my first sentiment analysis and get what later I will call *valder* results. To conclude I saved a .csv file with the resutls.

```
#%%
from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm
import pandas as pd
import numpy as np
import math
import nltk
#%%
nltk.download('vader_lexicon')

my_twitter_db =
  pd.DataFrame(pd.read_csv('../Datasets/my_twitter_data_set.csv', sep=','))

#%%
list(my_twitter_db.columns.values)
#%%
my_twitter_db.shape[0]
#%%
my_twitter_db.rename(columns = {'Unnamed: 0':'ID'}, inplace = True)
#%%
# Dataset cleaning to fasten up the results and avoid errors

# Removing unwanted columns for the analysis
my_twitter_db = my_twitter_db.drop('is_reply_to_tid', axis=1)
my_twitter_db = my_twitter_db.drop('In_reply_To', axis=1)
my_twitter_db = my_twitter_db.drop('Conversation_Id', axis=1)
my_twitter_db = my_twitter_db.drop('Tweet_Id', axis=1)

# Removing empy rows on tweets
my_twitter_db = my_twitter_db[my_twitter_db['Tweet'].notna()]
#%%
my_twitter_db
#%%
sia = SentimentIntensityAnalyzer()
#%%
# Run the polarity score on the entire dataset
res = {}
for i, row in tqdm(my_twitter_db.iterrows(), total=len(my_twitter_db)):
    text = row['Tweet']
    myid = row['ID']
    res[myid] = sia.polarity_scores(text)
#%%
vaders = pd.DataFrame(res).T
vaders = vaders.reset_index().rename(columns={'index': 'ID'})
vaders = vaders.merge(my_twitter_db, how='left')
#%%
```

```
vaders
#%%
vaders.to_csv('vaders.csv')
#%%
#files.download('vaders.csv')
```

## 4.3 TextBlob

*TextBlob*, a Lexicon-based sentiment analyzer, contains some predetermined guidelines, and weight dictionary, which has some scores that assist in determining the polarity of a statement. For this reason, rule-based sentiment analyzers are another name for lexicon-based sentiment analyzers.

Hence, also in this case I runned the code and obtained the results of sentiment analysis with this tool.

```
#%%
import pandas as pd
import numpy as np
#cleaning
import re
# Sentiment Scoring
from textblob import TextBlob

#Loading
from tqdm.notebook import tqdm
#%%
# Importing my dataset of tweets and other relevant infos
my_twitter_db =
  pd.DataFrame(pd.read_csv('../Datasets/my_twitter_data_set.csv', sep=','))
#%%
my_twitter_db.rename(columns = {'Unnamed: 0':'ID'}, inplace = True)
#%%
my_twitter_db
#%%
# Dataset cleaning to fasten up the results and avoid errors

# Removing unwanted columns for the analysis
my_twitter_db = my_twitter_db.drop('is_reply_to_tid', axis=1)
my_twitter_db = my_twitter_db.drop('In_reply_To', axis=1)
my_twitter_db = my_twitter_db.drop('Conversation_Id', axis=1)
my_twitter_db = my_twitter_db.drop('Tweet_Id', axis=1)

# Removing empy rows on tweets
my_twitter_db = my_twitter_db[my_twitter_db['Tweet'].notna()]

#%%
# Creating a cleaning Function
def clean_text(text):
    pat1 = r'@[^ ]+'
    pat2 = r'https?://[A-Za-z0-9./]+'
    pat3 = r'\'s'
    pat4 = r'\#\w+'
    pat5 = r'&amp '
    pat6 = r'[^A-Za-z\s]'
    combined_pat = r'|'.join((pat1, pat2,pat3,pat4,pat5, pat6))
    text = re.sub(combined_pat,"",text).lower()
    return text.strip()
#%%
my_twitter_db["cleaned_tweet"] = my_twitter_db["Tweet"].apply(clean_text)
#%% md

#%%
my_twitter_db["cleaned_tweet"].head(100)
#%%
my_twitter_db
#%%
print("Running sentiment process")
for row in tqdm(my_twitter_db.itertuples(), total=len(my_twitter_db)):
    tweet = my_twitter_db.at[row[0], 'cleaned_tweet']
    #run sentiment using TextBlob
    analysis = TextBlob(tweet)
    #set value to dataframe
    my_twitter_db.at[row[0], 'polarity'] = analysis.sentiment[0]
    my_twitter_db.at[row[0], 'subjectivity'] = analysis.sentiment[1]
    #Create Positive / negative column depending on polarity
    if analysis.sentiment[0]>0:
        my_twitter_db.at[row[0], 'Sentiment'] = "Positive"
    elif analysis.sentiment[0]<0:
        my_twitter_db.at[row[0], 'Sentiment'] = "Negative"
    else:
        my_twitter_db.at[row[0], 'Sentiment'] = "Neutral"
#%%
my_twitter_db
#%%
my_twitter_db.to_csv('textblob.csv')
```

To conclude, I finished this process joining the two dataset and obtaingin both *VADER* and *TextBlob* sentiment analysis results.

# 5 Setup the software

The programming languages used for the development of the study and the writing of the report are R and Python. The first step is to define the work directory and to load the libraries needed:

## 5.1 Loading the libraries

```
library(dplyr)
library(readr)
library(data.table)
library(janitor)
library(tidyverse)
library(lubridate)
library(ggplot2)
library(hrbrthemes)
library(chron)
library(gridExtra)

# Map Graph
library("sf")
library("rnaturalearth")
library("rnaturalearthdata")
library(terra)
library(spData)
library(spDataLarge)
library(RColorBrewer)
library(bookdown)
library(ggmap)

# Data Analysis
library(caTools)
library(ROCR)

library(rpart)
library(rpart.plot)
library(randomForest)
library(caret)
library(e1071)
library(class)


# H2O
library(h2o)
```

# 6 Importing Data

In the first part of my work is crucial to properly load all the dataframes I retrieved in the previous stages of my work.

## 6.1 Loading my dataframes

Hence, the first step is to load the datasets in the system.

```
tord_db <- read.csv2("/Users/ettorefalde/DataspellProjects/ICO_Forecasting/Datasets/tord_v3.csv")
twitter_db <- read.csv("/Users/ettorefalde/DataspellProjects/ICO_Forecasting/Datasets/my_db.csv")
```

## 6.2 Loading Scripts

Moreover, after having loaded all my starting dataframes I can also upload a script file, that will contain some functions of my work.

```
source("functions_script.R")
```

## 6.3 Variables analysis

I finally loaded my dataset with the sentiment analysis and the TORD dataset that can be useful in the further analysis. So, the TORD dataset, was made by Paul P. Momtaz, and I downloaded it directly from his website. While my dataset was made in the previous stages of my work, and the process to obtain it and run the sentiment analysis models are described in the thesis.

### 6.3.1 TORD Variables analysis

Anyway, the TORD dataset is composed by 6415 observations of 34 variables. The variables in this datset are:

```
names(tord_db)
```

1. **id**: Variable that represent the id of the id of the ICO
2. **name**: Variable that represent the full name of the ICO
3. **token**: Variable that represent the ICO token, or the abbreviation of it
4. **country**: Variable that shows where the ICO comes from
5. **is__ico**: Boolean variable that shows if and ICO is an ICO
6. **is__ieo**: Boolean variable that shows if and ICO is an IEO
7. **is__sto**: Boolean variable that shows if and ICO is an STO
8. **ico__start**: The starting date when the ICO was launched
9. **ico__end**: The ending date when the ICO was closed
10. **price__usd**: The value of an ICO in USD
11. **raised__usd**: The amount of money raised in USD during the campaign
12. **distributed__in__ico**: The amount of ICOs distributed during the campaign
13. **sold__tokens**: The amount tokens sold during the campaign
14. **token__for__sale**: The total number of tokens for sale during the campaign
15. **whitelist**: Boolean variable that shows the presence of the list of registered and approved participants that are given exclusive access to contribute to an initial coin offering (ICO) or a presale
16. **kyc**: Boolean variable that shows if was used the know your customer technology to remain compliant in a cost and time-efficient way
17. **bonus**: Boolean variable that represent bonuses allocated for investors that prove their engagement towards the project's cause
18. **restricted__areas**: Variable that represent the restricted areas
19. **min__investment**: Variable that represent the minum investment accepted to buy the ICO

20. **bounty**: Boolean variable that represent incentives offered to an array of participants for various activities
21. **mvp**: Variable that shows if the minimum valuable product is available or not
22. **pre_ico_start**: Variable that shows the pre-ICO start date
23. **pre_ico_end**: Variable that shows the pre-ICO end date
24. **pre_ico_price_usd**: Variable that represent the pre-ICO price in USD
25. **platform**: Boolean variable that represent if a platform were used
26. **accepting**: Variable that shows the currency that are accepted in exchange of the ICOs
27. **link_white_paper**: Variable that report the link to the whitepaper
28. **linkedin_link**: Variable that report the link to the LinkedIn page
29. **github_link**: Variable that report the link to the GitHub repository
30. **website**: Variable that report the link to the website
31. **rating**: Variable that shows the rating value of the ICO in a scale 0 to 5
32. **teamsize**: Variable that represent the team sise of the company / startup
33. **Coinmarketcap_identifier**: Variable that represent the Coinmarketcap identifier
34. **ERC20**: Boolean variable that shows if the ERC20 standard is available

### 6.3.2 my_db Variables analysis

While, my dataset as 5618723 observations of 26 variables. The variables in this datset are:

```
names(twitter_db)
```

1. **X**: Auto generated variavles corressponding to the ID var
2. **ID**: Variable that represent the ID of the tweet
3. **neg**: Variable that correspond to the VADER sentiment analysis negative percentage of the tweet
4. **neu**: Variable that correspond to the VADER sentiment analysis neutral percentage of the tweet
5. **pos**: Variable that correspond to the VADER sentiment analysis positive percentage of the tweet
6. **compound**: Variable that correspond to the VADER sentiment analysis. The sum of positive, negative & neutral scores which is then normalized between -1(most extreme negative) and +1 (most extreme positive)
7. **ICO_name**: Variable that represent the full name of the ICO
8. **Token_name**: Variable that represent the ICO token, or the abbreviation of it
9. **User**: The username that published that posted the tweet on Twitter
10. **verified**: Boolean variable that shows if a profile is verified or not
11. **Date_Created**: Variable that shows the creation date of the user profile
12. **Follows_Count**: Variable that represent the total number of profiles that the user follows
13. **Friends_Count**: Variable that represent the total number of profiles that follows the user
14. **Retweet_Count**: Variable that represent the total retweet number
15. **Language**: Variable that represent the language of the post
16. **Date_Tweet**: Variable that shows the creation date of the tweet by the user
17. **Number_of_Likes**: Variable that represent the total number of likes that the tweet received
18. **Source_of_Tweet**: Variable that shows the instrument used to post the tweet (e.g. WebApp, Andoid, IOS)
19. **Tweet**: Variable that represent the content of the tweet
20. **Hashtags**: Variable that shows the hashtags used in the tweet
21. **Coordinates**: Variable that shows the cordinates of the location where the tweet is posted
22. **Place**: Variable that represent the place where the tweet is posted
23. **cleaned_tweet**: Variable that shows the content of the tweet after has been cleaned for the TextBlob model
24. **polarity**: Variable that represent the polarity of the TextBlob model
25. **subjectivity**: Variable that represent the subjectivity of the TextBlob model
26. **Sentiment**: Variable that represent the sentiment of the TextBlob model

# 7 Data Cleaning

## 7.1 Place DB

The *place_db* dataframe, will be use in the graphical representation of the maps, to show what are the main location of interests in ICOs (exculing China, where data is not available). So, this is a dataset made from the twitter dataset.

First of all, I copied the *twitter_db* in another variable, in order to subset this lsat one and obtain only the columns needed. The creation of this dataset will also lighten the *twitter_db* dropping the place related columns.

```
place_db <- twitter_db
```

### 7.1.1 Cleaning

In this section I can clean my *place_db* in order to show some map graphs in the future.

From the analysis above is clear that I can remove some unnecessary columns, and keep only: *ico_name*, *date_tweet*, *coordinates* and *place*.

So, I will unify the format of the names of the columns.

```
place_db <- place_db %>% clean_names(., "snake")
```

Then, I will display the columns names, which are the same to the beginning *my_db*.

```
names(place_db)
```

```
##  [1] "x"               "id"              "neg"             "neu"
##  [5] "pos"             "compound"        "ico_name"        "token_name"
##  [9] "user"            "verified"        "date_created"    "follows_count"
## [13] "friends_count"   "retweet_count"   "language"        "date_tweet"
## [17] "number_of_likes" "source_of_tweet" "tweet"           "hashtags"
## [21] "coordinates"     "place"           "cleaned_tweet"   "polarity"
## [25] "subjectivity"    "sentiment"
```

```
place_db <- subset(place_db, select = -c(x,id, neg, neu, pos, compound,
                                          token_name, user, verified,
                                          date_created, follows_count,
                                          friends_count, retweet_count,
                                          language, number_of_likes,
                                          source_of_tweet, tweet, hashtags,
                                          cleaned_tweet, place,
                                          polarity, subjectivity,
                                          sentiment))
```

```
#place_db %>% tabyl(coordinates)
place_db <- place_db %>% filter(coordinates != "")
place_db$hashtags <- NULL
```

Now I will do the same procedure to fix the *date_tweet* variable. The procedure is the same as did above with *my_db*.

```
typeof(place_db$date_tweet)
```

```
## [1] "character"
```

```
place_db$date_tweet = substr(place_db$date_tweet,1,nchar(place_db$date_tweet)-6)
```

```
place_db$date_tweet <- ymd_hms(place_db$date_tweet)
place_db$date_tweet_ymd <- as.Date(place_db$date_tweet)
place_db$date_tweet_time <- format(place_db$date_tweet,"%H:%M:%S")

place_db <- place_db %>% relocate(date_tweet_ymd, .after = date_tweet)
place_db <- place_db %>%relocate(date_tweet_time, .after = date_tweet_ymd)
```

In this case, I needed to create an *id* column, in order to fix *coordinates* for *place_db*.

```
place_db <- tibble::rowid_to_column(place_db, "id")
```

Therefore, I needed to remove the unwanted words in the string.

```
typeof(place_db$coordinates)
```

```
## [1] "character"
```

```
place_db$coordinates <- gsub('[Coordinates(longitude=latitute,)]', '',
                             place_db$coordinates)

temp <- read.table(text = place_db$coordinates, header = FALSE,
                   col.names = c('longitude', 'latitude'))
temp <- tibble::rowid_to_column(temp, "id")
place_db <- inner_join(temp, place_db, by="id")
place_db$coordinates <- NULL
place_db <- place_db %>% relocate(longitude, .after = date_tweet_time)
place_db <- place_db %>% relocate(latitude, .after = longitude)
rm(temp)
```

I finally retrieve my *place_db* dataset with proper columns that will help me in the future for possible geographical graphs.

## 7.2 TORD

This dataframe in the overall project has a substantial relevance. Hence, It requires a proper data cleaning.

### 7.2.1 Dimensions

First of all, I will check the dimensions of this dataframe. So, it is possibile to notice that *tord_db* has 6415 rows and 34 columns.

```
tord_db %>% dim()
```

```
## [1] 6415   34
```

```
tord_db %>% nrow()
```

```
## [1] 6415
```

```
tord_db %>% ncol()
```

```
## [1] 34
```

### 7.2.2 Head & Tail

Consecutively, is interesting to check the top and the bottom of this dataset. In the results we can notice that some ICOs' project were able to raise a lot of capital.

```
tord_db %>% head(10)
tord_db %>% tail(10)
tord_db %>% arrange(desc(raised_usd)) %>% top_n(10, raised_usd)
tord_db %>% arrange(raised_usd) %>% top_n(-10, raised_usd)
```

### 7.2.3 Names

Also in this case I will unify the columsn names. Moreover, I also need to change the *name* columns that represent the name of the ico with the name: *ico_name* because in the furhter stages will be usefull in the collapsing phase and joining with the the *twitter_db*.

```
tord_db <- tord_db %>% clean_names(., "snake")
colnames(tord_db)[colnames(tord_db) == "name"] <- "ico_name"
names(tord_db)
```

```
##  [1] "id"                        "ico_name"
##  [3] "token"                     "country"
##  [5] "is_ico"                    "is_ieo"
##  [7] "is_sto"                    "ico_start"
##  [9] "ico_end"                   "price_usd"
## [11] "raised_usd"                "distributed_in_ico"
## [13] "sold_tokens"               "token_for_sale"
## [15] "whitelist"                 "kyc"
## [17] "bonus"                     "restricted_areas"
## [19] "min_investment"            "bounty"
## [21] "mvp"                       "pre_ico_start"
## [23] "pre_ico_end"               "pre_ico_price_usd"
## [25] "platform"                  "accepting"
## [27] "link_white_paper"          "linkedin_link"
## [29] "github_link"               "website"
## [31] "rating"                    "teamsize"
## [33] "coinmarketcap_identifier"  "erc20"
```

### 7.2.4 Variable types

Clearly, a cleaned dataframe needs to have the correct types of variables. Therefore, in the following chunk of code the dataframe was properly corrected.

```r
#str(tord_db)

tord_db$ico_start <- ymd(tord_db$ico_start)
tord_db$ico_end <- ymd(tord_db$ico_end)
tord_db$price_usd <- as.numeric(tord_db$price_usd)
```

```
## Warning in base::as.numeric(x): NAs introduced by coercion
```

```r
tord_db$raised_usd <- as.numeric(tord_db$raised_usd)
```

```
## Warning in base::as.numeric(x): NAs introduced by coercion
```

```r
tord_db$distributed_in_ico <- as.numeric(tord_db$distributed_in_ico)
```

```
## Warning in base::as.numeric(x): NAs introduced by coercion
```

```r
tord_db$sold_tokens <- as.numeric(tord_db$sold_tokens)
```

```
## Warning in base::as.numeric(x): NAs introduced by coercion
```

```r
tord_db$token_for_sale <- as.numeric(tord_db$token_for_sale)
```

```
## Warning in base::as.numeric(x): NAs introduced by coercion
```

```r
tord_db$pre_ico_start <- ymd(tord_db$pre_ico_start)
tord_db$pre_ico_end <- ymd(tord_db$pre_ico_end)
tord_db$pre_ico_price_usd <- as.numeric(tord_db$pre_ico_price_usd)
```

```
## Warning in base::as.numeric(x): NAs introduced by coercion
```

```r
tord_db$rating <- as.numeric(tord_db$rating)
tord_db$teamsize <- as.integer(tord_db$teamsize)
```

### 7.2.5 NA values

Carrying on with the data cleaning I also needed to handle the NA values. So, I tried to fill those values when there was the possibility with logical data. For instance, in the case of raised_usd, when the cell was equal to NA, I converted it to 0. While, for other columsn I adopted a different strategies. In the case of *ico_start* and *ico_end* I determined the $\Delta T$, and if both *ico_start* and *ico_end* where NA values the $\Delta T$ was setted equal to 0. Instead, when a only one of *ico_start* or *ico_end* where equal to NA, the value of $\Delta T$ was setted to 25, which is the average time of ICO duration.

```r
# colnames(tord_db)[apply(tord_db, 2, anyNA)]

# token
tord_db <- tord_db %>% mutate(token = coalesce(token,ico_name))

# ico_start
# ico_end
# From these two variable I can determine the duartion
tord_db$ico_duration <- tord_db$ico_end - tord_db$ico_start
# set the NA equal to the mean
tord_db$ico_duration[is.na(tord_db$ico_duration)] <- 25
#case for negatives setting equal to the abs value
tord_db$ico_duration[tord_db$ico_duration<0] <- abs(tord_db$ico_duration)
```

```
## Warning in NextMethod("[<-"): number of items to replace is not a multiple of
## replacement length
```

```
# handling higher values 25days
tord_db$ico_duration[tord_db$ico_duration>=365] <- 25
# case for 0 values
tord_db$ico_duration[tord_db$ico_duration==0] <- 25
tord_db <- tord_db %>% relocate(ico_duration, .after = ico_end)

tord_db$ico_duration <- as.integer(tord_db$ico_duration)





# price_usd
tord_db$price_usd[is.na(tord_db$price_usd)] <- 0

# raised_usd
tord_db$raised_usd[is.na(tord_db$raised_usd)] <- 0

# distributed_in_ico
tord_db$distributed_in_ico[is.na(tord_db$distributed_in_ico)] <- 0

# sold_tokens
tord_db$sold_tokens[is.na(tord_db$sold_tokens)] <- 0

# token_for_sale
tord_db$token_for_sale[is.na(tord_db$token_for_sale)] <- 0

# whitelist
tord_db$whitelist[is.na(tord_db$whitelist)] <- "No"
tord_db$whitelist<-ifelse(tord_db$whitelist=="Yes",1,0)

# restricted_areas -> counting tht number of areas
tord_db$restricted_areas[is.na(tord_db$restricted_areas)] <- "No"
tord_db$restricted_areas_total <- gsub("[^[:alnum:] ]", "",
                                       tord_db$restricted_areas)
tord_db$restricted_areas_total <- strsplit(tord_db$restricted_areas_total, " ")
tord_db$restricted_areas_total <- sapply(tord_db$restricted_areas_total, length)
tord_db <- tord_db %>% relocate(restricted_areas_total,
                                .after = restricted_areas)

# min_investment
tord_db$min_investment[is.na(tord_db$min_investment)] <- "0 USD"

# mvp
tord_db$mvp[is.na(tord_db$mvp)] <- "No"
tord_db$mvp<-ifelse(tord_db$mvp=="Available",1,0)


# pre_ico_price_usd
tord_db$pre_ico_price_usd[is.na(tord_db$pre_ico_price_usd)] <- 0
```

```r
# accepting
tord_db$accepting[is.na(tord_db$accepting)] <- "Other"

# link_white_paper
tord_db$link_white_paper<-ifelse(tord_db$link_white_paper==
                                   is.na(tord_db$link_white_paper), 0, 1)
tord_db$link_white_paper[is.na(tord_db$link_white_paper)] <- 0

# linkedin_link
tord_db$linkedin_link<-ifelse(tord_db$linkedin_link==
                                is.na(tord_db$linkedin_link), 0, 1)
tord_db$linkedin_link[is.na(tord_db$linkedin_link)] <- 0

# github_link
tord_db$github_link<-ifelse(tord_db$github_link==
                              is.na(tord_db$github_link), 0, 1)
tord_db$github_link[is.na(tord_db$github_link)] <- 0

# website
tord_db$website<-ifelse(tord_db$website==is.na(tord_db$website), 0, 1)
tord_db$website[is.na(tord_db$website)] <- 0

# rating
tord_db$rating[is.na(tord_db$rating)] <- 0

# teamsize
tord_db$teamsize[is.na(tord_db$teamsize)] <- 0

# coinmarketcap_identifier
tord_db$coinmarketcap_identifier[is.na(tord_db$coinmarketcap_identifier)] <- 0
tord_db$coinmarketcap_identifier<-ifelse(tord_db$coinmarketcap_identifier==0,
                                         0, 1)

# erc20
tord_db$erc20[is.na(tord_db$erc20)] <- 0



# Set the new boolean variable did pre ico
tord_db$did_pre_ico <- ifelse(tord_db$pre_ico_start ==
                                is.na(tord_db$pre_ico_start) &
                                tord_db$pre_ico_end ==
                                is.na(tord_db$pre_ico_end), 0, 1)
tord_db$did_pre_ico[is.na(tord_db$did_pre_ico)] <- 0
tord_db <- tord_db %>% relocate(did_pre_ico, .after = pre_ico_end)




# pre_ico_start
# pre_ico_end
# From these two variable I can determine the duartion
tord_db$pre_ico_duration <- tord_db$pre_ico_end - tord_db$pre_ico_start
```

```r
# Handling just one pre_ico date
tord_db$pre_ico_duration[tord_db$did_pre_ico == 1 &
                           is.na(tord_db$pre_ico_end)] <- 25
tord_db$pre_ico_duration[tord_db$did_pre_ico == 1 &
                           is.na(tord_db$pre_ico_start)] <- 25

# set the NA equal to 0
tord_db$pre_ico_duration[is.na(tord_db$pre_ico_duration)] <- 0

#case for negatives setting equal to the abs value
tord_db$pre_ico_duration[tord_db$pre_ico_duration<0] <- abs(tord_db$pre_ico_duration)
```

```
## Warning in NextMethod("[<-"): number of items to replace is not a multiple of
## replacement length
```

```r
# handling higher values 25days
tord_db$pre_ico_duration[tord_db$pre_ico_duration>365] <- 25

tord_db <- tord_db %>% relocate(pre_ico_duration, .after = pre_ico_end)

tord_db$pre_ico_duration <- as.integer(tord_db$pre_ico_duration)
```

### 7.2.6 Check n's

In this stage I noticed some discrepancy in the variable *country*, because some countries were the same but wrote with different strings. Therefore, with this code I unified the entries.

```r
tabyl(tord_db$country)

# Check the Countries
tord_db %>% tabyl(country)

## Singapore
tord_db$country <- ifelse(tord_db$country ==
                            "SINGAPORE",
                          "Singapore", tord_db$country)
## USA
tord_db$country <- ifelse(tord_db$country ==
                            "USA",
                          "usa", tord_db$country)
## India
tord_db$country <- ifelse(tord_db$country ==
                            "India",
                          "india", tord_db$country)
## Curacao
tord_db$country <- ifelse(tord_db$country ==
                            "Curacao",
                          "Curaçao", tord_db$country)
## Cameroun
tord_db$country <- ifelse(tord_db$country ==
                            "Cameroun",
                          "Cameroon", tord_db$country)
```

### 7.2.7 Setting success

Generally speaking, the ICO's success is defined as the ratio between the soft cap and the total funds received. The hard cap over the total funds can be an index of full success. Anyway, in the given dataset, those values are not available, and there is the problem of all the NAs values that retrive infinite o null values of success. Therefore, I decided to set a success amount high enough to represent all ICOs in this dataset.

```
tord_db$success <- ifelse(tord_db$raised_usd >= 500000, 1, 0)
tord_db$success <- as.integer(tord_db$success)
tord_db <- tord_db %>% relocate(success, .after = token_for_sale)
```

### 7.2.8 Setting country popularity

Another variable that can be considered in the analysis is the country popularity. Thus, the *temp* variable was created to count the total number per country, and then assign to each ICO, that value corresponding to the country belonging.

```
# Set country popularity scores
t <- tabyl(tord_db$country)

temp <- tord_db %>% dplyr::select(ico_name, country) %>%
  summarise(country = t$`tord_db$country`,
            country_popularity = t$n,
            country_popularity_percentage = t$percent)

tord_db <- inner_join(tord_db, temp, by="country")

tord_db <- tord_db %>% relocate(country_popularity, .after = country)
tord_db <- tord_db %>% relocate(country_popularity_percentage,
                                .after = country_popularity)
```

## 7.3 Twitter DB Data Cleaning

Also for the *twitter_db* is important to do a data cleaning. Mainly to be sure that all the automated work done by the Python script doesn't present any mistake.

### 7.3.1 Dimensions

First of all, I will check the dimensions of this dataframe. So, it is possibile to notice that *tord_db* has 5618723 rows and 26 columns.

```
twitter_db %>% dim()
```

```
## [1] 5618723      26
```

```
twitter_db %>% nrow()
```

```
## [1] 5618723
```

```
twitter_db %>% ncol()
```

```
## [1] 26
```

### 7.3.2 Head & Tail

Consecutively, is interesting to check the top and the bottom of this dataset. In the results we can notice that some tweet have a really positive impact, but some of them have also an higher value of *subjectivity*.

```
twitter_db %>% head(10)
twitter_db %>% tail(10)
twitter_db %>% arrange(desc(polarity)) %>% top_n(10, polarity)
twitter_db %>% arrange(polarity) %>% top_n(-10, polarity)
```

### 7.3.3 Names

I will unify the columns names, in the same way I did it for the *TORD* dataframe.

```
twitter_db <- twitter_db %>% clean_names(., "snake")
names(twitter_db)
```

### 7.3.4 Converting and adding variables

Thus, in this part some variables where converted in boolean variables, with 0 and 1. Moreover, I was also able to count the number of *hashtags* and the number words used in the tweets and the in the cleaned tweets.

```
# verified
twitter_db$verified<-ifelse(twitter_db$verified=="False",0,1)


# tweet
twitter_db$tweet_words <- str_count(twitter_db$tweet, "\\S+")
twitter_db <- twitter_db %>% relocate(tweet_words, .after = tweet)

# hashtags
twitter_db$hashtags_total <- gsub("[^[:alnum:] ]", "", twitter_db$hashtags)
twitter_db$hashtags_total <- strsplit(twitter_db$hashtags_total, " ")
twitter_db$hashtags_total <- sapply(twitter_db$hashtags_total, length)
twitter_db <- twitter_db %>% relocate(hashtags_total, .after = hashtags)


# cleaned_tweet
twitter_db$cleaned_tweet_words <- str_count(twitter_db$cleaned_tweet, "\\S+")
twitter_db <- twitter_db %>% relocate(cleaned_tweet_words,
                                      .after = cleaned_tweet)

# date_tweet
twitter_db$date_tweet <- ymd_hms(twitter_db$date_tweet)
twitter_db$date_tweet_ymd <- as.Date(twitter_db$date_tweet)
twitter_db$date_tweet_time <- format(twitter_db$date_tweet,"%H:%M:%S")
twitter_db <- twitter_db %>% relocate(date_tweet_ymd, .after = date_tweet)
twitter_db <- twitter_db %>%relocate(date_tweet_time, .after = date_tweet_ymd)
```

## 7.4 Variable types

Like in the previous case, also in this dataframe is crucial to have proper types for each variable.

```
#str(twitter_db)

twitter_db$neu <- as.numeric(twitter_db$neu)
```

```
## Warning in base::as.numeric(x): NAs introduced by coercion
```

```
twitter_db$neu <- as.numeric(twitter_db$neu)
twitter_db$date_tweet_time <- hms(twitter_db$date_tweet_time)
```

```
## Warning in .parse_hms(..., order = "HMS", quiet = quiet): Some strings failed to
## parse, or all strings are NAs
```

```
twitter_db$subjectivity <- as.numeric(twitter_db$subjectivity)
```

### 7.4.1 NA values

As we can see, in this chunk of code I tried to covert all the possible variables into 0 from NA. I also created a new *id* column and filled NA values in the *sentiment* column as neutral.

```
# colnames(twitter_db)[apply(twitter_db, 2, anyNA)]

# id
twitter_db <- subset(twitter_db, select = -c(id, x))
twitter_db <- tibble::rowid_to_column(twitter_db, "id")

# neg
twitter_db$neg[is.na(twitter_db$neg)] <- 0

# neu
twitter_db$neu[is.na(twitter_db$neu)] <- 0

# pos
twitter_db$pos[is.na(twitter_db$pos)] <- 0

# compound
twitter_db$compound[is.na(twitter_db$compound)] <- 0

# follows_count
twitter_db$follows_count[is.na(twitter_db$follows_count)] <- 0

# friends_count
twitter_db$friends_count[is.na(twitter_db$friends_count)] <- 0

# retweet_count
twitter_db$retweet_count[is.na(twitter_db$retweet_count)] <- 0

# number_of_likes
twitter_db$number_of_likes[is.na(twitter_db$number_of_likes)] <- 0

# polarity
twitter_db$polarity[is.na(twitter_db$polarity)] <- 0

# subjectivity
twitter_db$subjectivity[is.na(twitter_db$subjectivity)] <- 0.5

# sentiment
twitter_db$sentiment[is.na(twitter_db$sentiment)] <- "Neutral"
```

### 7.4.2 Removing

To lighten a bit this dataframe I also decided to remove the unwanted columns not useful for furter analysis.

```
# Dropping unwanted cols
twitter_db <- subset(twitter_db, select = -c(user, date_created,
                                             date_tweet, source_of_tweet,
                                             tweet, hashtags, coordinates,
                                             place, cleaned_tweet))
```

### 7.4.3 Check n's

In some varibles such as *language* and *sentiment* I notied some empty values different from NAs. So, I cleaned them up.

```
tabyl(twitter_db$language)
twitter_db <- twitter_db %>% filter(language != "")

tabyl(twitter_db$sentiment)
twitter_db <- twitter_db %>% filter(sentiment != "")
```

### 7.4.4 Change Columns names

Another important step to better clarify which sentiment analysis model correspond my results, I need to change the name of the those columns.

Firstly, I will highlight the results of the VADER Sentiment Analysis.

```
twitter_db <- rename(twitter_db, vader_negative = neg)
twitter_db <- rename(twitter_db, vader_neutral = neu)
twitter_db <- rename(twitter_db, vader_positive = pos)
twitter_db <- rename(twitter_db, vader_compound = compound)
```

Secondly, I will highlight the results of the TextBlob Sentiment Analysis.

```
twitter_db <- rename(twitter_db, textblob_polarity = polarity)
twitter_db <- rename(twitter_db, textblob_subjectivity = subjectivity)
twitter_db <- rename(twitter_db, textblob_sentiment = sentiment)
```

# 8 Explenatory Analysis

Obviously, a part containing a graphical analysis is needed in order to better visualize their impact. Hence, in the following code is possible to notice different graphs that try to represent the main variable at their best.

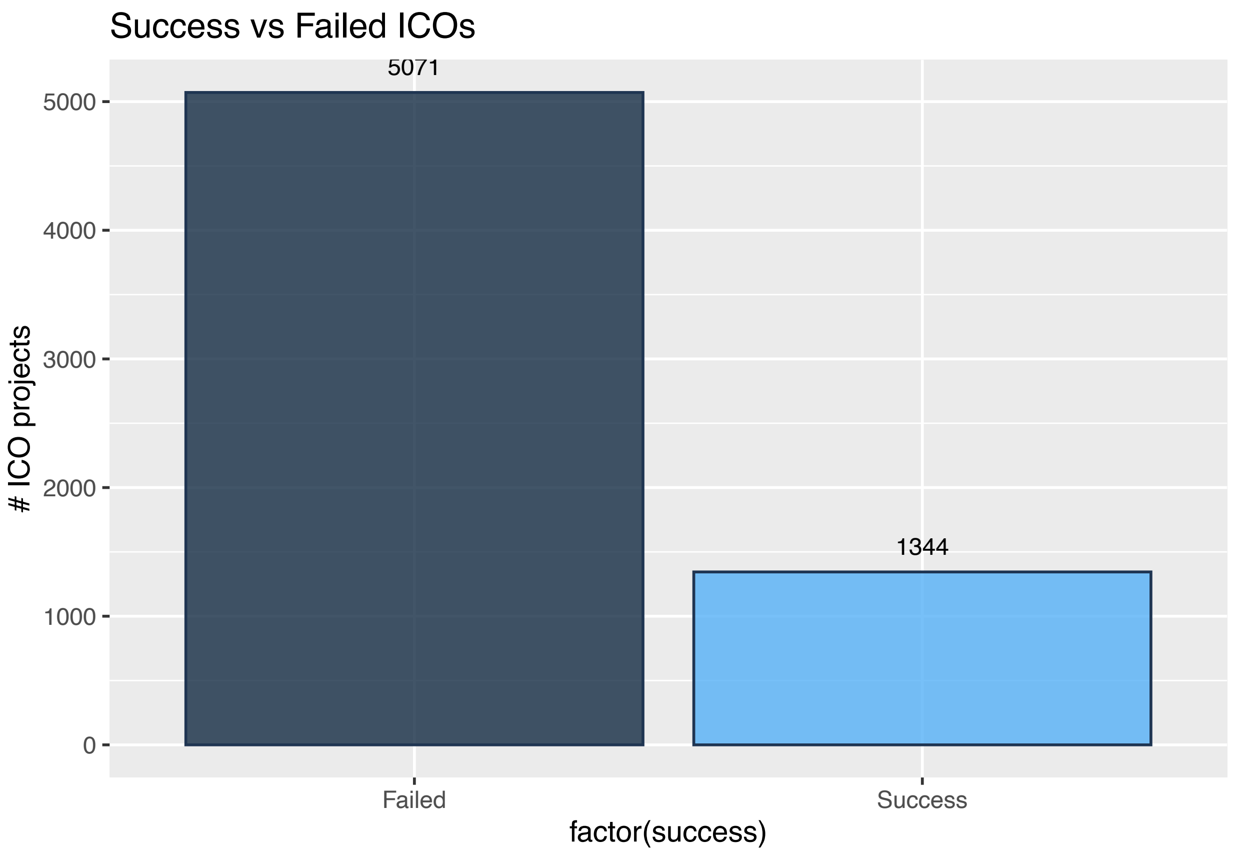
## 8.1 TORD Analysis

I will start this analysis with the TORD dataframe.

### 8.1.1 Success

Thus, the first graph that I wanted to see how many ICOs where able to reach $500000. In the graph, is possible to notice that only the 29% of the ICO projects where able to reach the half million dollar capital.

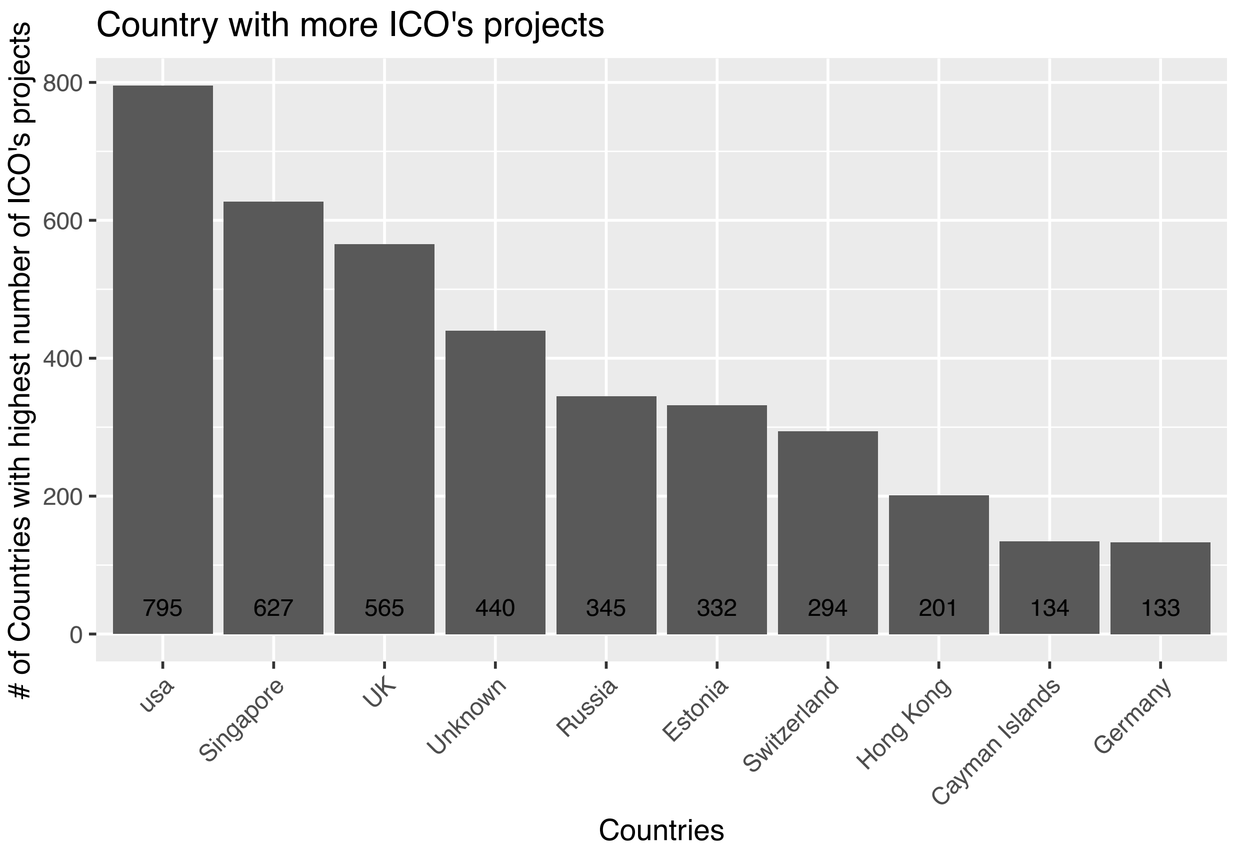
```
ggplot(tord_db, aes(x=factor(success), fill= success)) +
  geom_bar(color="#1F3552", alpha=0.8) +
  geom_text(aes(label = after_stat(count)),
            stat = "count", vjust = -1, colour = "black", size=3) +
  ggtitle("Success vs Failed ICOs") +
  ylab("# ICO projects") +
  scale_x_discrete(labels = c("Failed", "Success")) +
  theme(legend.position = "none")
```



### 8.1.2 Main 5 Countries

In this plot, I can show that the main ICO project where from USA and Singapore. In this graph we don't know if this is a success factor. Anyway, for sure means that the interest from people to develop this projects is form those countries. Moreover, I can also imagine that the majority of investors are from those countries.

```r
t <- tord_db %>%
  group_by(country) %>%
  summarise(total = n()) %>%
  top_n(n = 10, wt = total)


ggplot(t, aes(x = reorder(country, -total), y = total)) +
  geom_col() +
  geom_text(aes(label = total, y = ..count..), vjust = -1, stat = "count",
            color = "black", size = 3) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("Country with more ICO's projects") +
  xlab("Countries") +
  ylab("# of Countries with highest number of ICO's projects")
```



```r
rm(t)
```

### 8.1.3 Number of ICO start

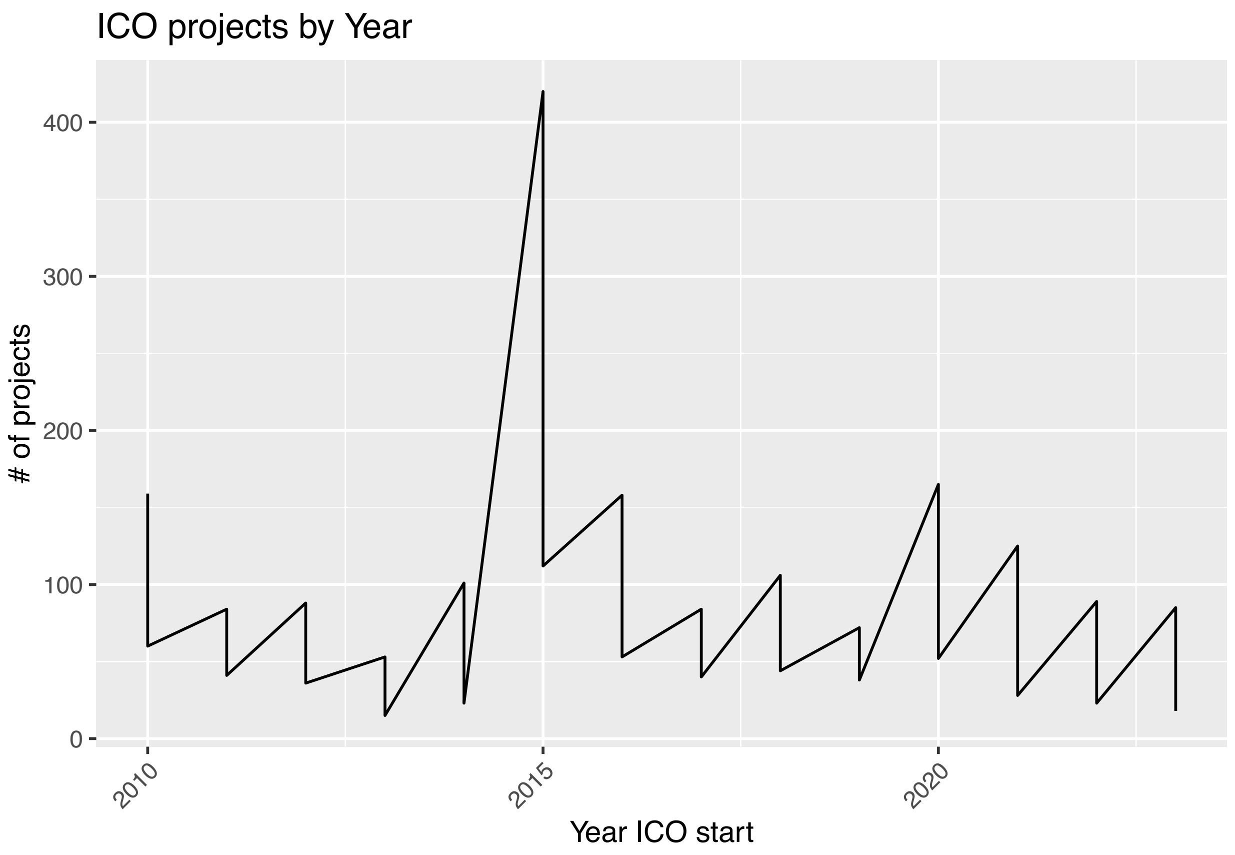
The following graphs represent the number of ICOs in which year started. Is possible to notice that in the 2015 there was a peak of project. Consecutively, also the successful ICOs were higher. Anyway, in from 2017 to 2019 is clear how the proportion of total number of ICOs and the success of those is higher.

```r
# t <- tord_db %>%
#     group_by(success, year = lubridate::floor_date(ico_start, "year")) %>%
#     summarise(count = n())
```

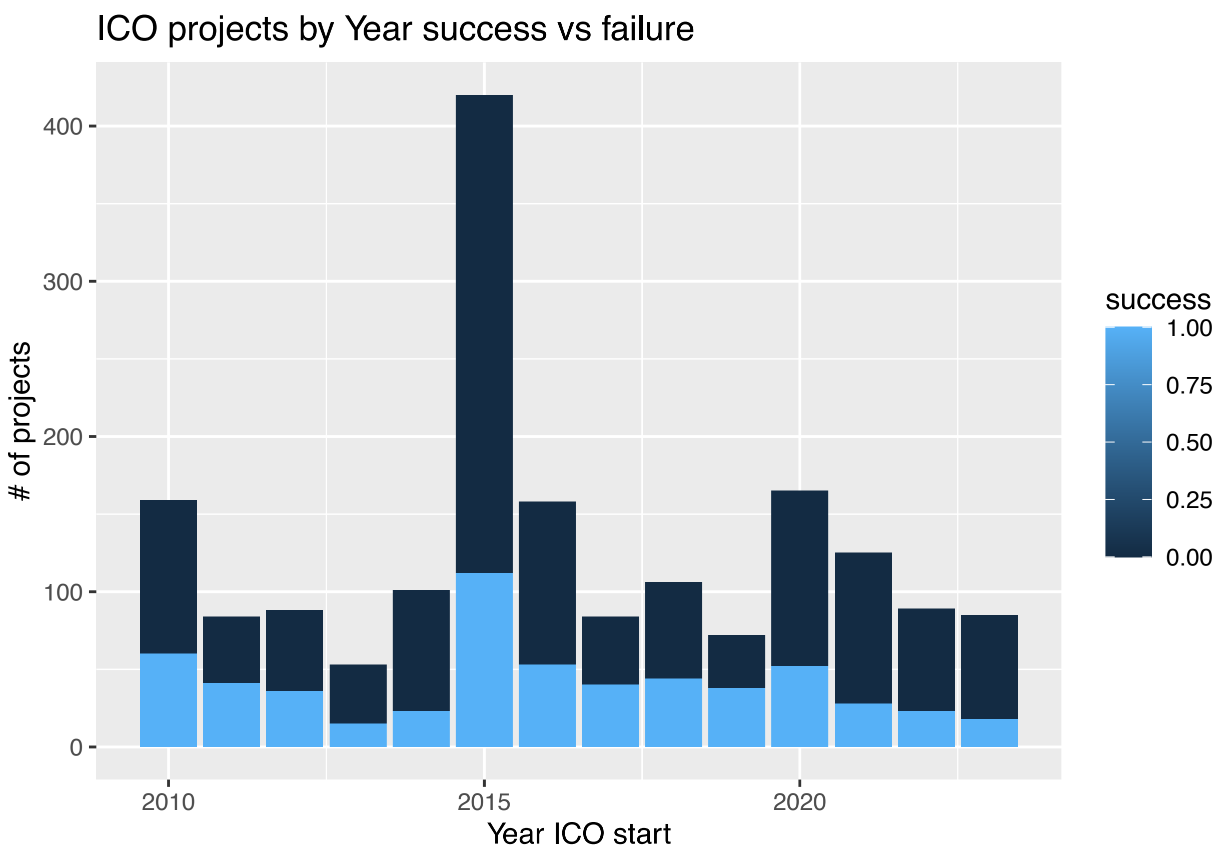
```
t <- tord_db %>%
    group_by(success, year = lubridate::floor_date(ico_start, "year")) %>%
  summarise(count = n())
```

```
## `summarise()` has grouped output by 'success'. You can override using the
## `.groups` argument.
```

```
t <- t %>% filter(between(year, "2010-01-01", "2023-01-01"))


ggplot(t, aes(x = year, y = count)) +
  geom_line() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ylab("# of projects") + xlab("Year ICO start") +
  ggtitle("ICO projects by Year")
```



```
ggplot(t, aes(fill=success, y=count, x=year)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# of projects") + xlab("Year ICO start") +
  ggtitle("ICO projects by Year success vs failure")
```



```
rm(t)
```

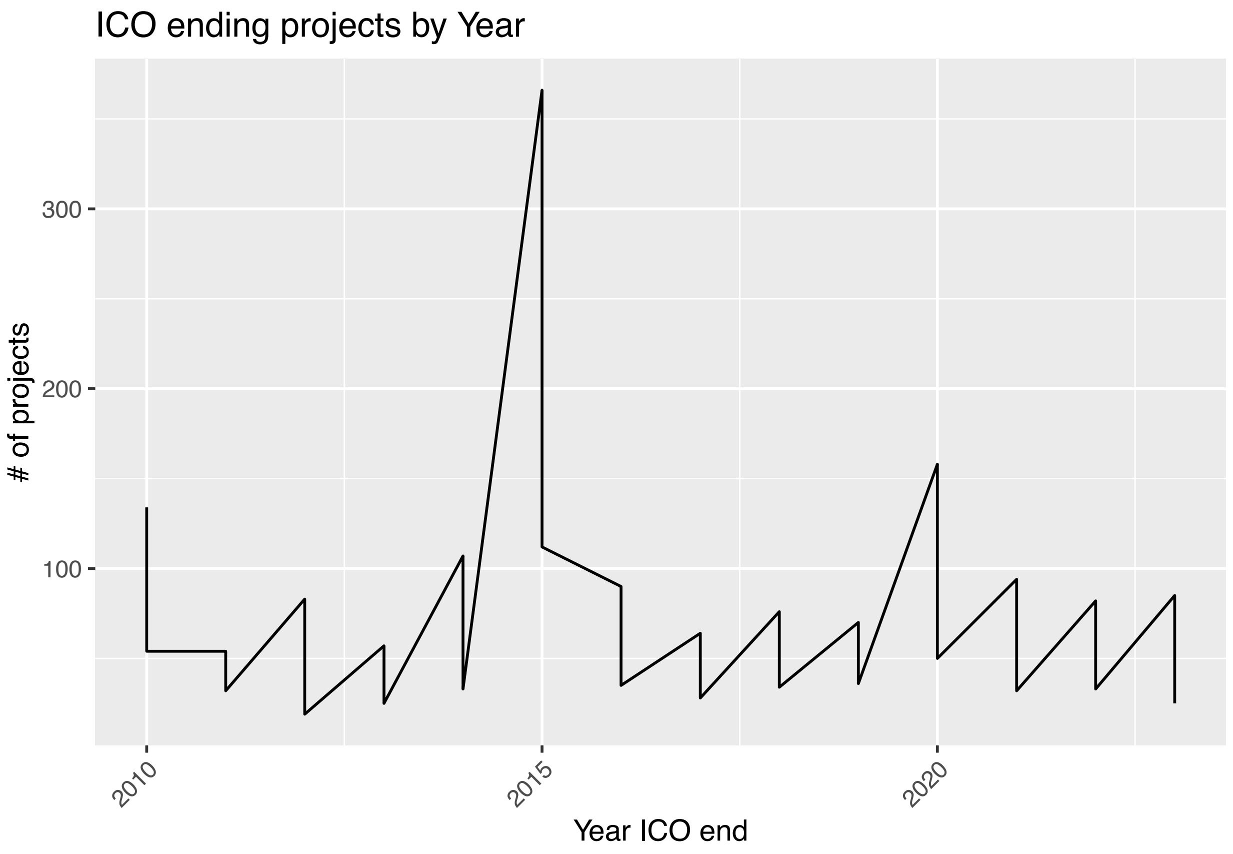
### 8.1.4 Number of ICO end

Also in this case we can notice that the favorite years of ICO success were from 2017 to 2019.

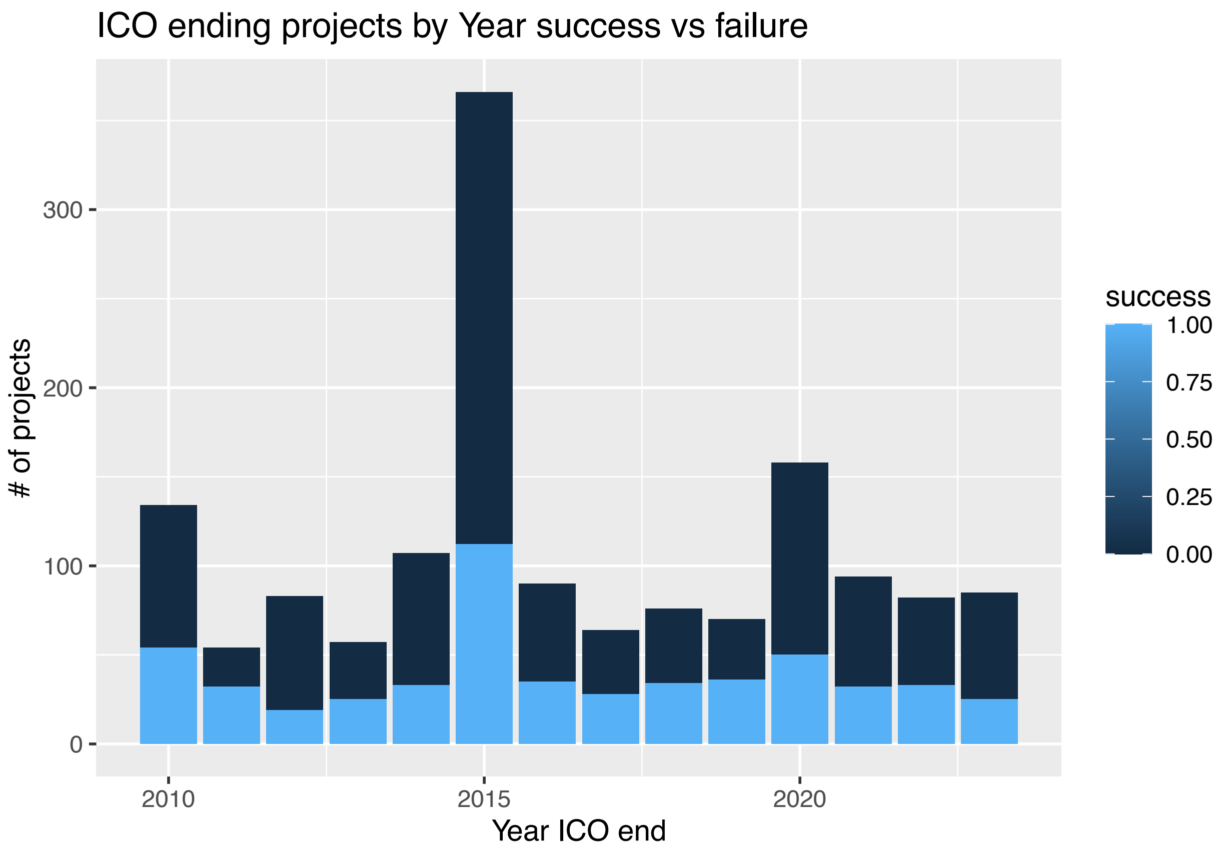
```
# t <- tord_db %>%
#     group_by(success, year = lubridate::floor_date(ico_end, "year")) %>%
#     summarise(count = n())

t <- tord_db %>%
    group_by(success, year = lubridate::floor_date(ico_end, "year")) %>%
  summarise(count = n())
```

```
## `summarise()` has grouped output by 'success'. You can override using the
## `.groups` argument.
```

```
t <- t %>% filter(between(year, "2010-01-01", "2023-01-01"))


ggplot(t, aes(x = year, y = count)) +
  geom_line() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ylab("# of projects") + xlab("Year ICO end") +
  ggtitle("ICO ending projects by Year")
```



```
ggplot(t, aes(fill=success, y=count, x=year)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# of projects") + xlab("Year ICO end") +
  ggtitle("ICO ending projects by Year success vs failure")
```

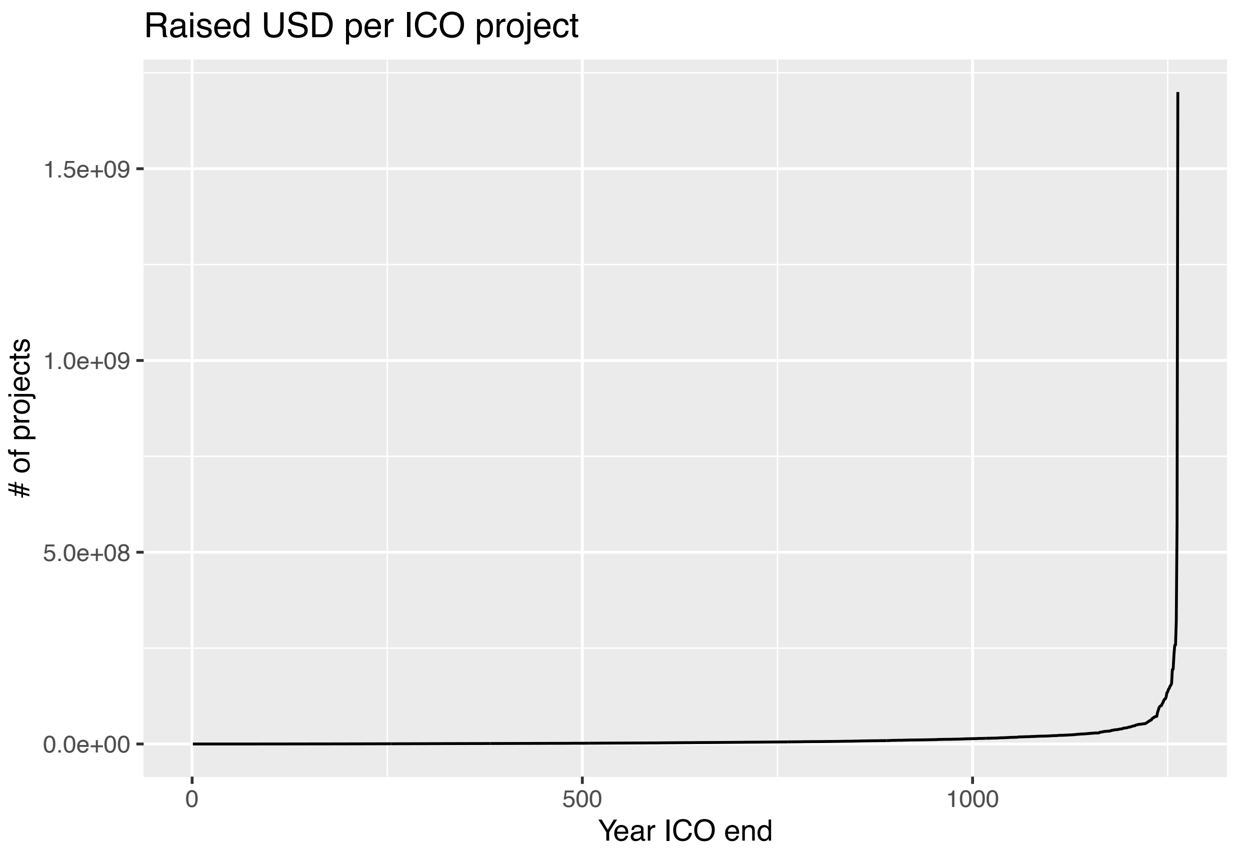


```
rm(t)
```

### 8.1.5 Raised USD

Obviously, due to the *raised_usd* variable is the main factor in determining ICO success in this analysis is also clear that only those ICOs projects who reaced success raised the needed capital. Anyway, is also interesting to notice that, there are some ICOs, that where able to raise a insanely large capital.

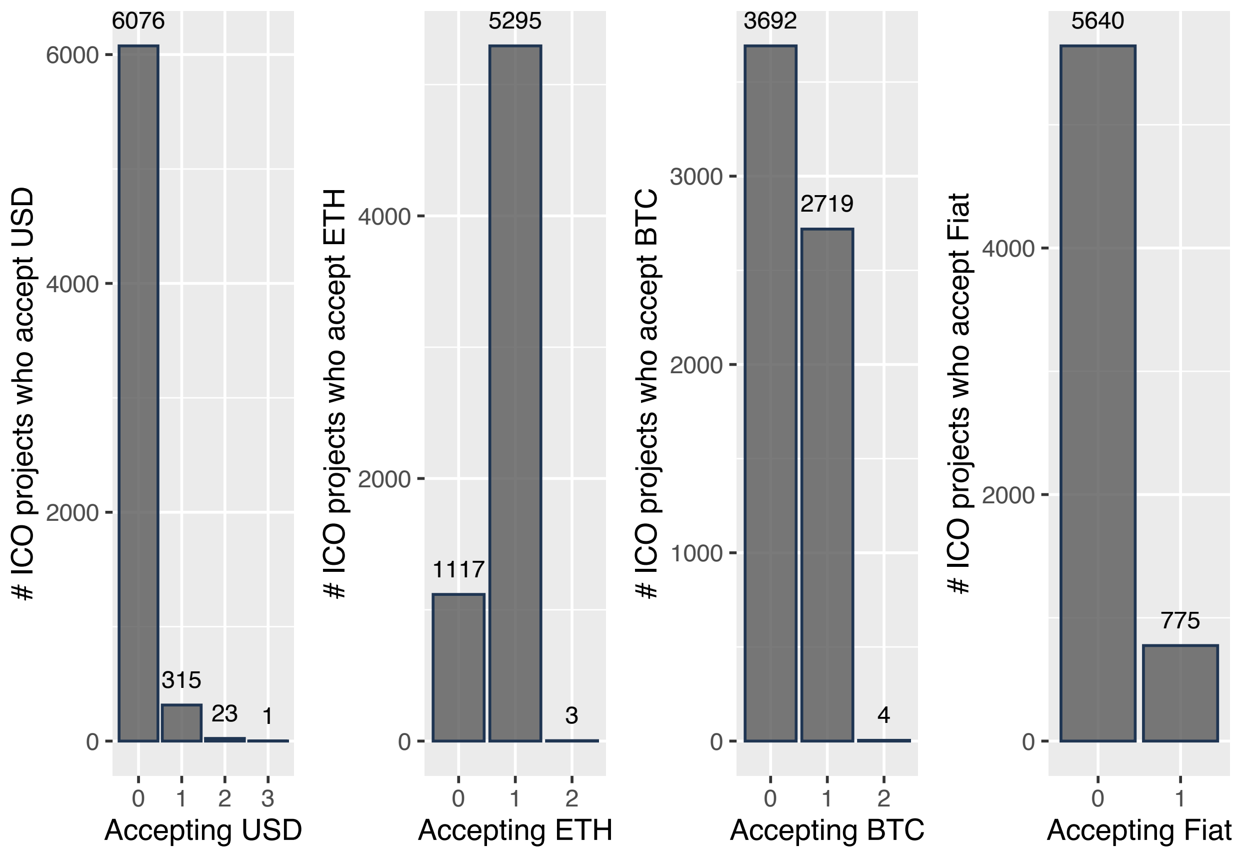
```
t <- tord_db %>%
    group_by(raised_usd) %>%
    summarise(count = n())

t  <- tibble::rowid_to_column(t, "id")


ggplot(t, aes(x = id, y = raised_usd)) +
  geom_line()+
  ylab("# of projects") + xlab("Year ICO end") +
  ggtitle("Raised USD per ICO project")
```



### 8.1.6 Accepting

This plots are very interesting because represent the main currencies accepted in exchange of the sold token. Surprisingly, the main currency is ETH, while the second most accepted is BTC. Only few ICOs accepts Fiat currency.

```
str1 <- c("USD", "ETH", "BTC", "Fiat")
t <- tord_db %>% dplyr::select(accepting)

t[c("USD", "ETH", "BTC", "Fiat")] <- lapply(str1, function(x)
  lengths(regmatches(tord_db$accepting, gregexpr(x, tord_db$accepting))))

USD <- t %>% group_by(USD) %>%
    summarise(count = n())
ETH <- t %>% group_by(ETH) %>%
    summarise(count = n())
BTC <- t %>% group_by(BTC) %>%
    summarise(count = n())
Fiat <- t %>% group_by(Fiat) %>%
    summarise(count = n())


# Graphs
g_usd <- ggplot(t, aes(x=factor(USD))) +
  geom_bar(color="#1F3552", alpha=0.8) +
  geom_text(aes(label = after_stat(count)),
            stat = "count", vjust = -1, colour = "black", size=3) +
  xlab("Accepting USD") +
  ylab("# ICO projects who accept USD")

g_eth <- ggplot(t, aes(x=factor(ETH))) +
  geom_bar(color="#1F3552", alpha=0.8) +
  geom_text(aes(label = after_stat(count)),
            stat = "count", vjust = -1, colour = "black", size=3) +
  xlab("Accepting ETH") +
  ylab("# ICO projects who accept ETH")

g_btc <- ggplot(t, aes(x=factor(BTC))) +
  geom_bar(color="#1F3552", alpha=0.8) +
  geom_text(aes(label = after_stat(count)),
            stat = "count", vjust = -1, colour = "black", size=3) +
  xlab("Accepting BTC") +
  ylab("# ICO projects who accept BTC")

g_fiat <- ggplot(t, aes(x=factor(Fiat))) +
  geom_bar(color="#1F3552", alpha=0.8) +
  geom_text(aes(label = after_stat(count)),
            stat = "count", vjust = -1, colour = "black", size=3) +
  xlab("Accepting Fiat") +
  ylab("# ICO projects who accept Fiat")

grid.arrange(g_usd, g_eth, g_btc, g_fiat, ncol=4)
```

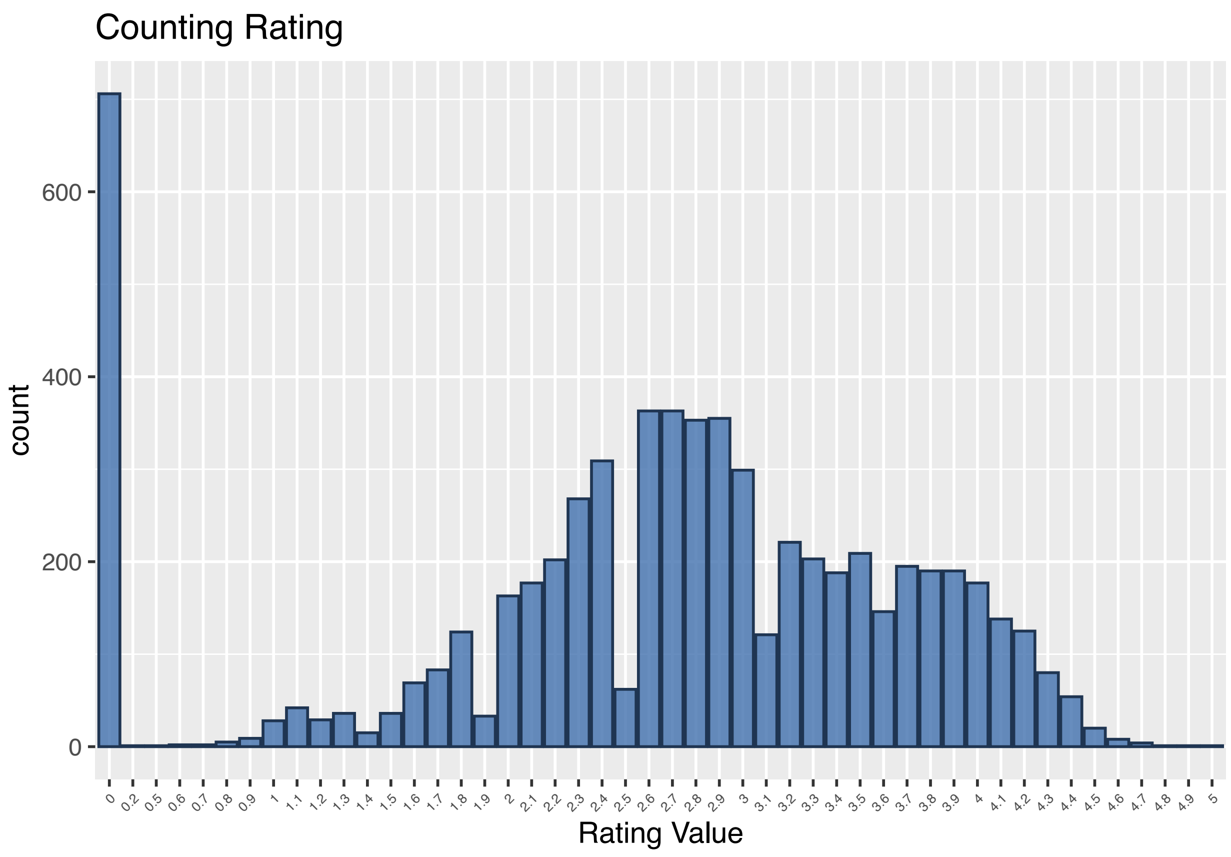


```
rm(BTC, ETH, USD, Fiat, t, g_fiat, g_eth, g_btc, g_usd, str1)
```

### 8.1.7 Rating

The *rating* variable clearly has an impact in determining the success of an ICO. In the following graphs we can notice that higher is the rating, higher is the probability to success.

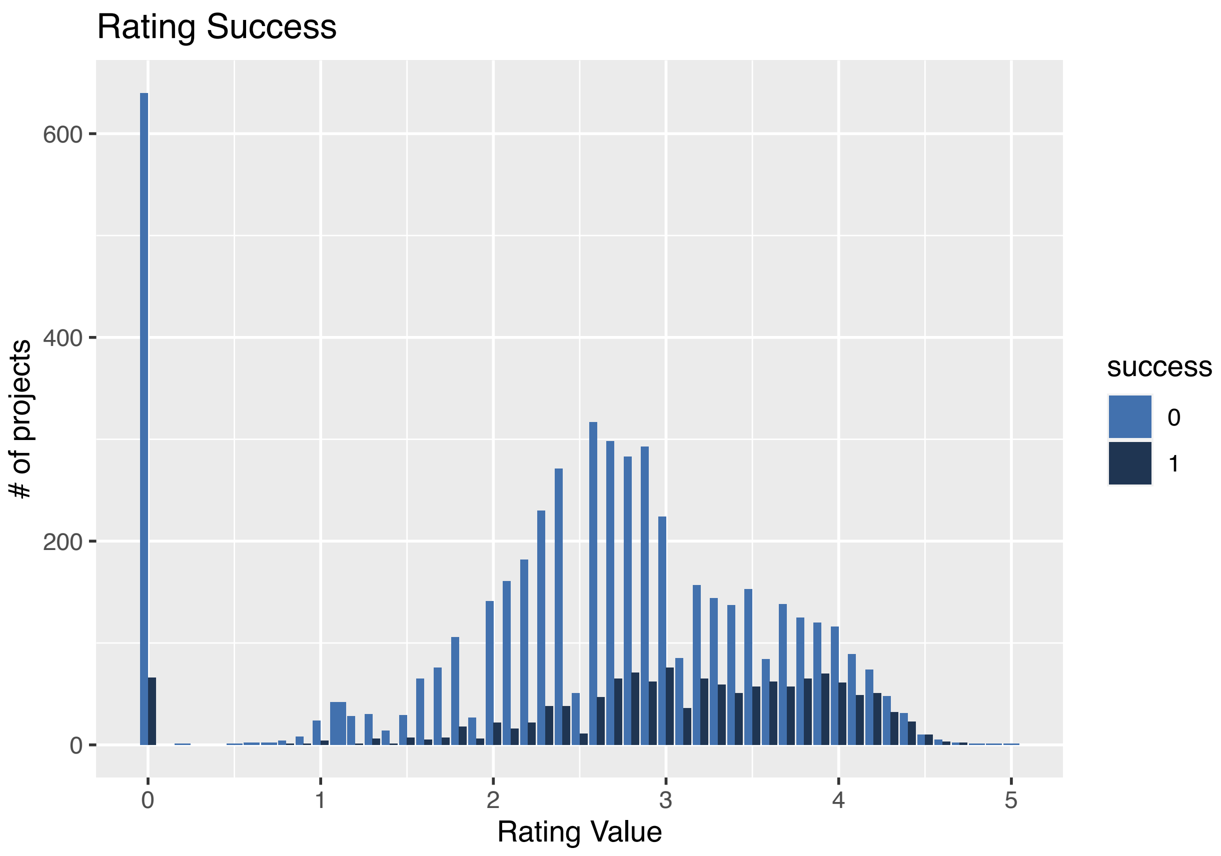
```
t <- tord_db %>% dplyr::select(ico_name, rating) %>%
  distinct(ico_name, .keep_all = TRUE)


ggplot(t, aes(factor(rating))) +
  geom_bar(fill="#4271AE", color="#1F3552", alpha=0.8) +
  xlab("Rating Value") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size=5)) +
  ggtitle("Counting Rating")
```



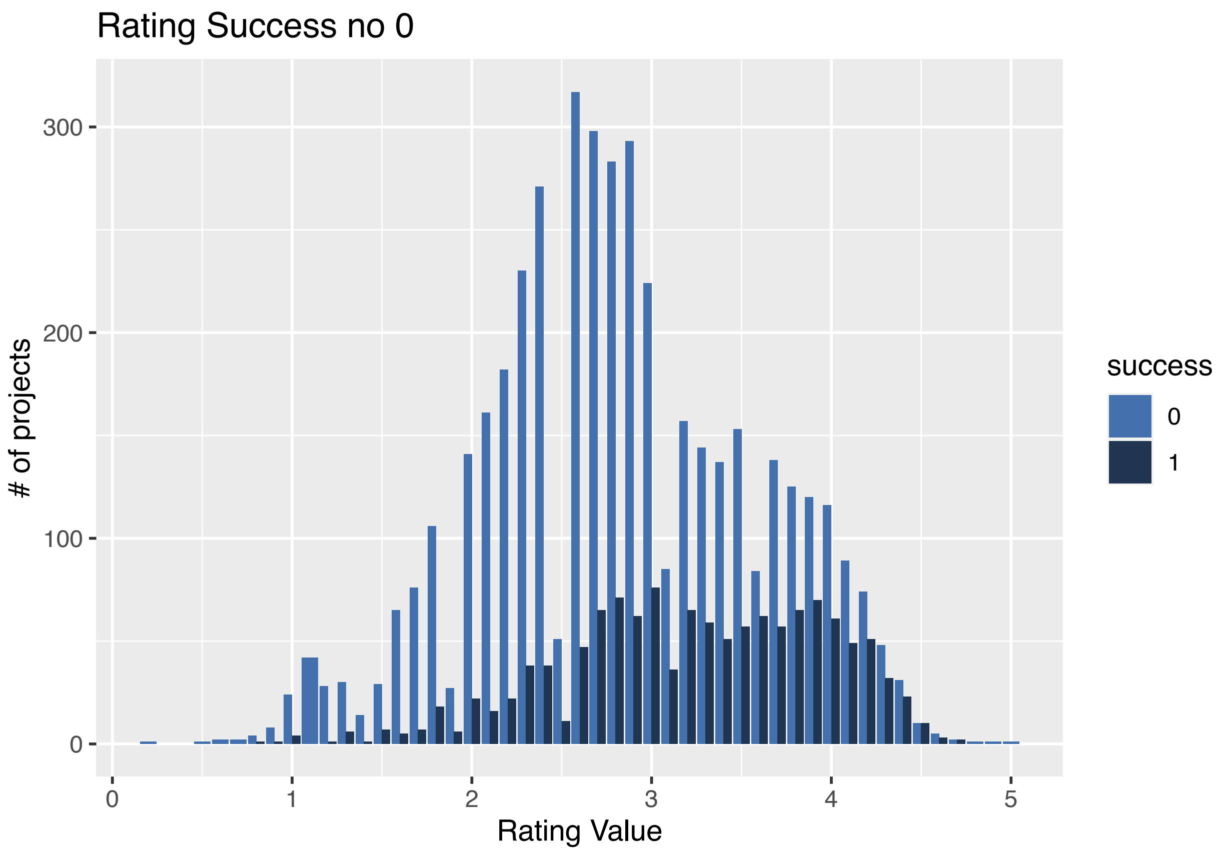
```
# Define custom colors
my_colors <- c("#4271AE", "#1F3552")

t <- tord_db %>%
  group_by(success, rating) %>%
  summarise(count = n())
```

```
## `summarise()` has grouped output by 'success'. You can override using the
## `.groups` argument.
```

```
# Convert 'success' variable to factor
t$success <- factor(t$success)

ggplot(t, aes(fill=success, y=count, x=rating)) +
  geom_bar(position="dodge", stat="identity") +
  ylab("# of projects") + xlab("Rating Value") +
  ggtitle("Rating Success")  +
  scale_fill_manual(values = my_colors)
```



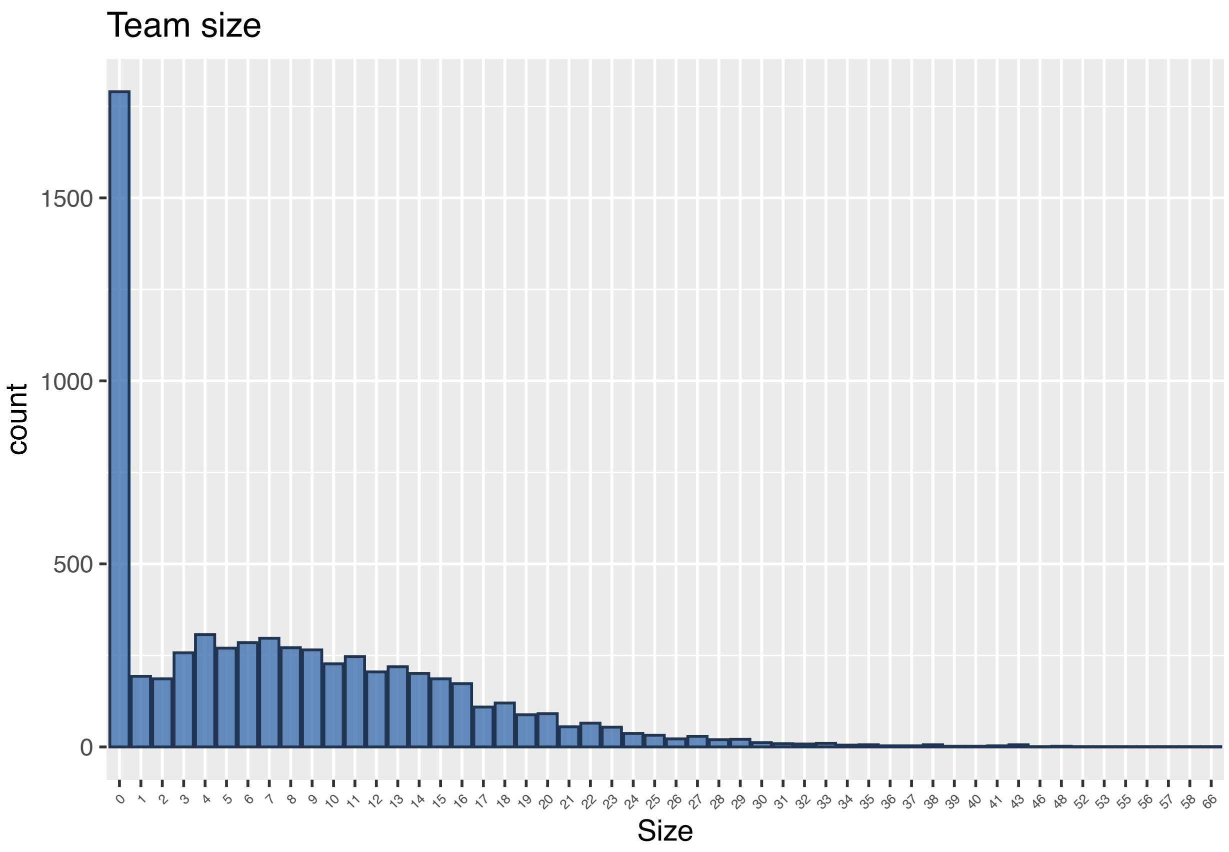
```
t <- t %>% filter(rating > 0)

# Convert 'success' variable to factor
t$success <- factor(t$success)

ggplot(t, aes(fill=success, y=count, x=rating)) +
  geom_bar(position="dodge", stat="identity") +
  ylab("# of projects") + xlab("Rating Value") +
  ggtitle("Rating Success no 0") +
  scale_fill_manual(values = my_colors)
```



### 8.1.8 Team size

Other very interesting plots are given by the *teamsize* variable. In the following graphs we can notice that the average team size is from 1 to 20 people. Moreover, the success of the ICO is higher when the team members are around 10.

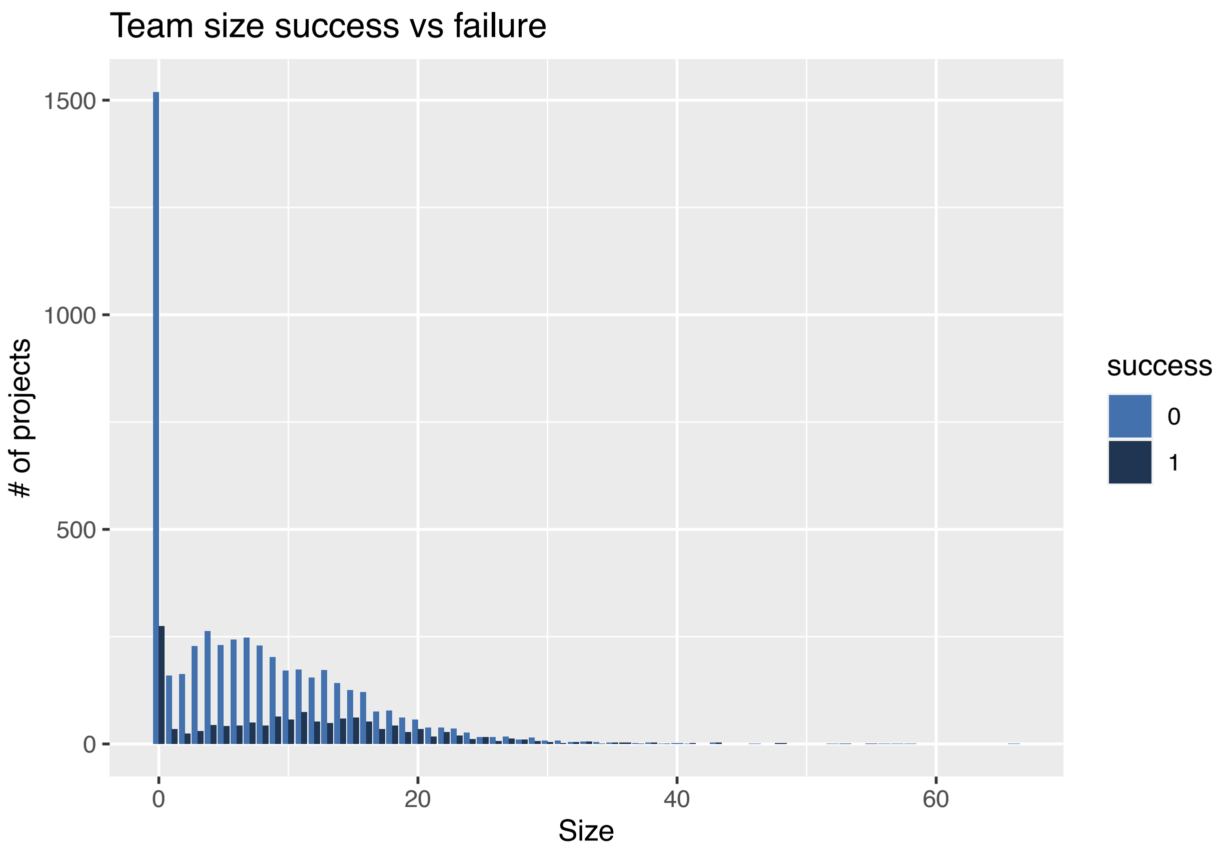
```
t <- tord_db %>% dplyr::select(ico_name, teamsize) %>%
  distinct(ico_name, .keep_all = TRUE)


ggplot(t, aes(factor(teamsize))) +
  geom_bar(fill="#4271AE", color="#1F3552", alpha=0.8) +
  xlab("Size") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size=5)) +
  ggtitle("Team size")
```



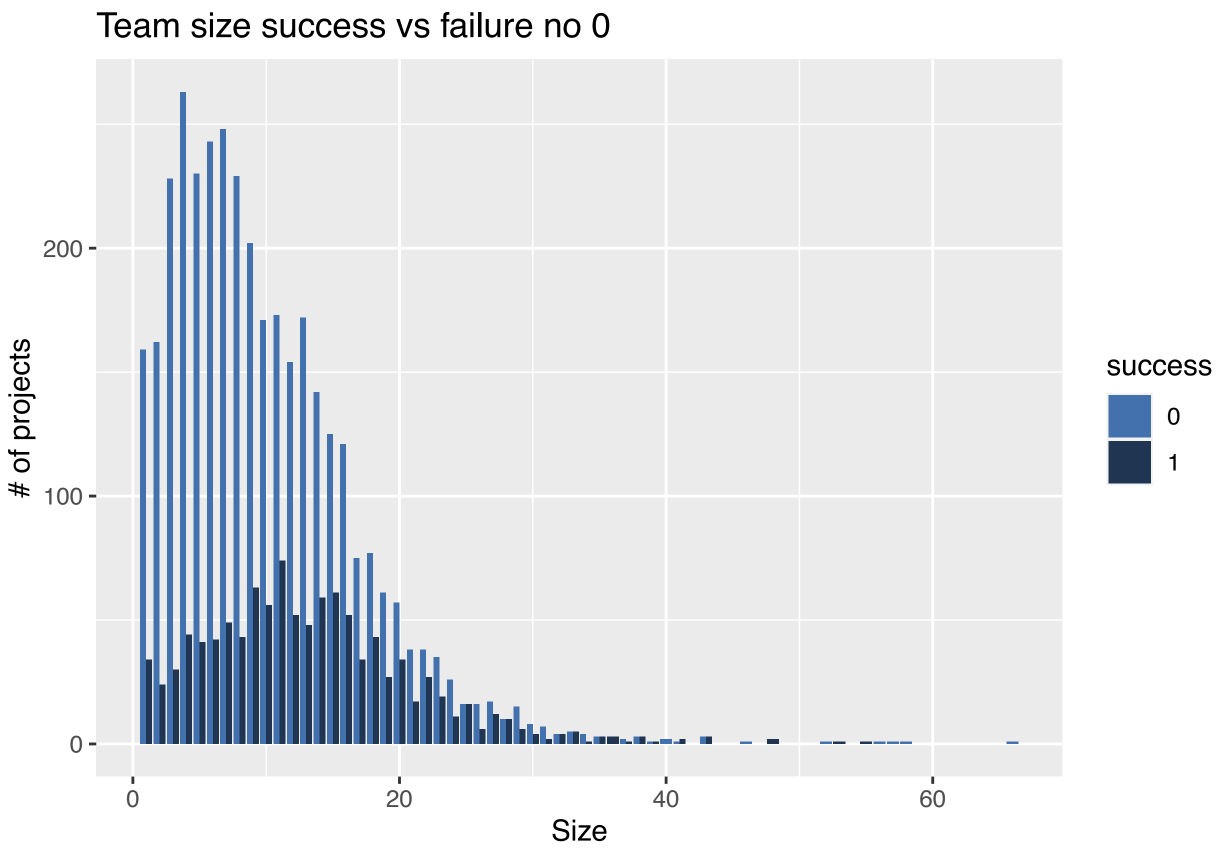
```
# Define custom colors
my_colors <- c("#4271AE", "#1F3552")


t <- tord_db %>%
    group_by(success, teamsize) %>%
    summarise(count = n())
```

```
## `summarise()` has grouped output by 'success'. You can override using the
## `.groups` argument.
```

```
t$success <- factor(t$success)

ggplot(t, aes(fill=success, y=count, x=teamsize)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# of projects") + xlab("Size") +
  ggtitle("Team size success vs failure")  +
  scale_fill_manual(values = my_colors)
```



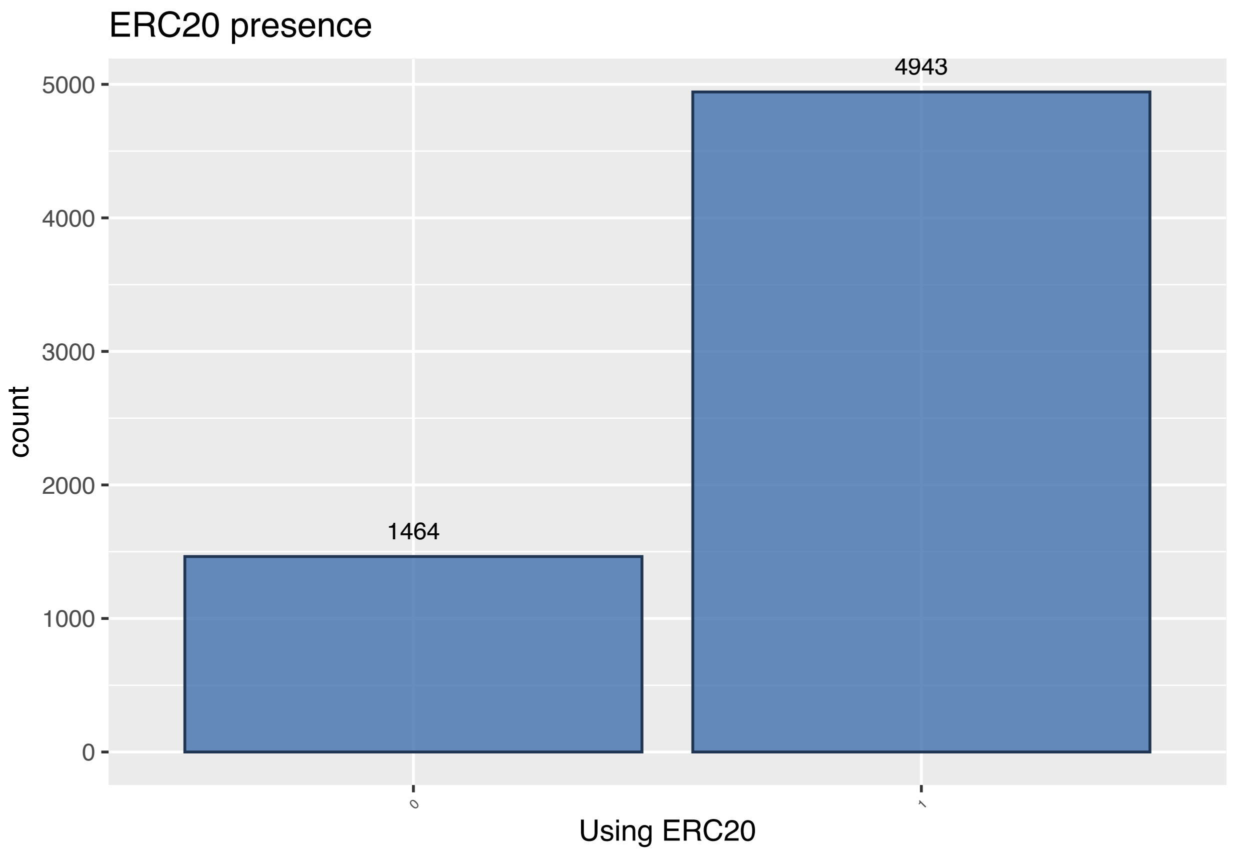
```
t <- t %>% filter(teamsize>0)
t$success <- factor(t$success)
ggplot(t, aes(fill=success, y=count, x=teamsize)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# of projects") + xlab("Size") +
  ggtitle("Team size success vs failure no 0")  +
  scale_fill_manual(values = my_colors)
```



### 8.1.9 ERC20

The ERC20 is the standard for fungible tokens created using the Ethereum blockchain. So, in with this plots, we can notice that the majority of the ICOs actually use this technology. Moreover, the usage of ERC20 implies also an higher visible possibilty to reach success.

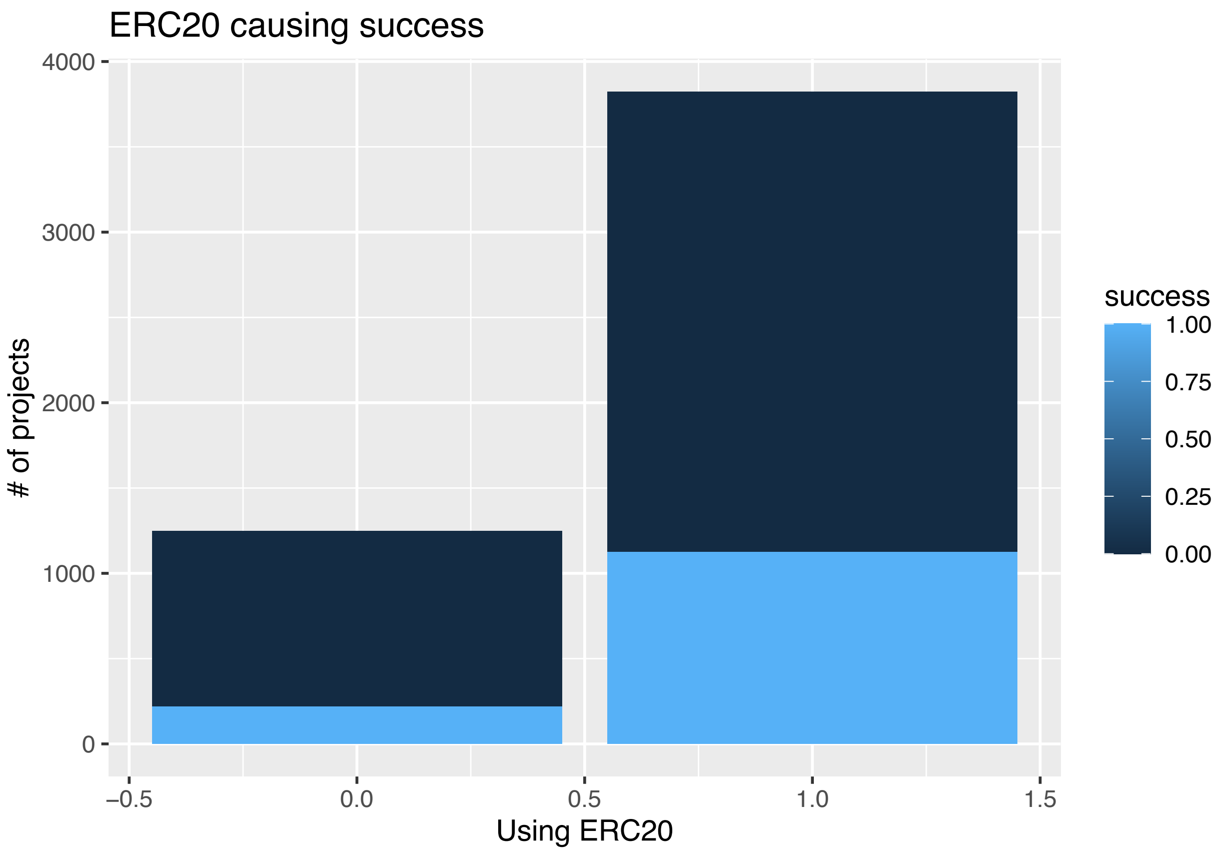
```
t <- tord_db %>% dplyr::select(ico_name, erc20) %>%
  distinct(ico_name, .keep_all = TRUE)


ggplot(t, aes(factor(erc20))) +
  geom_bar(fill="#4271AE", color="#1F3552", alpha=0.8) +
  xlab("Using ERC20") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size=5)) +
  ggtitle("ERC20 presence") +
  geom_text(aes(label = after_stat(count)),
            stat = "count", vjust = -1, colour = "black", size=3)
```



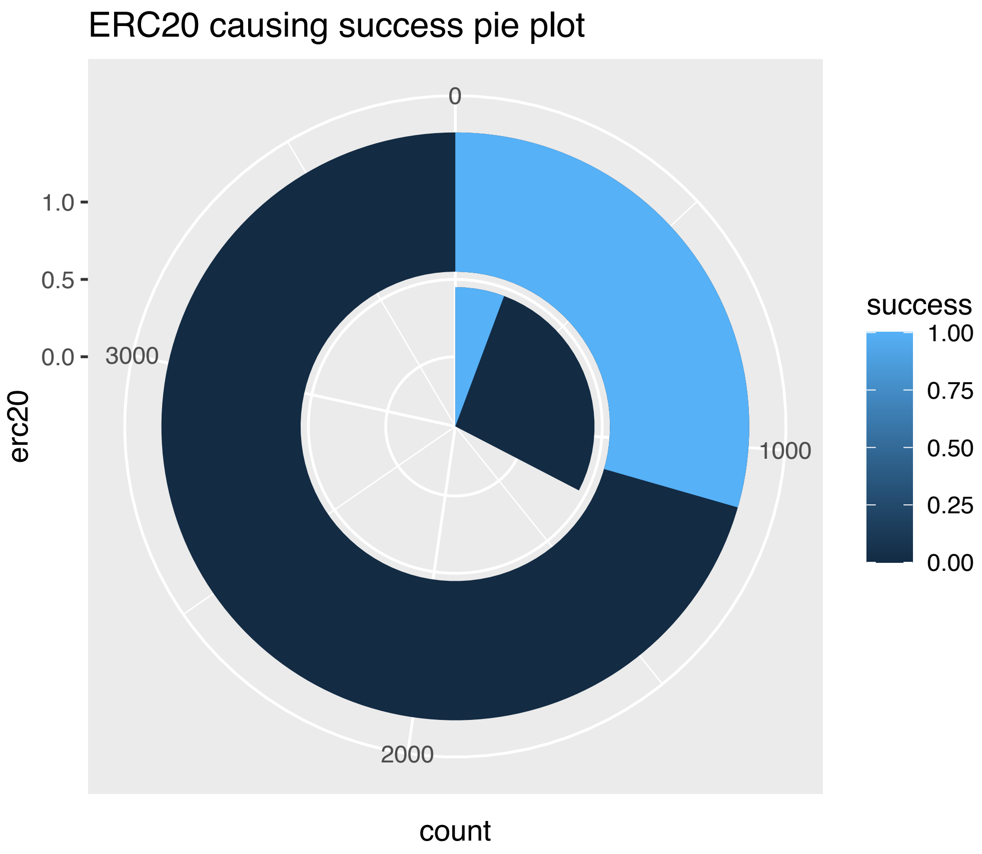
```
t <- tord_db %>%
    group_by(success, erc20) %>%
    summarise(count = n())
```

```
## `summarise()` has grouped output by 'success'. You can override using the
## `.groups` argument.
```

```
ggplot(t, aes(fill=success, y=count, x=erc20)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# of projects") + xlab("Using ERC20") +
  ggtitle("ERC20 causing success")
```



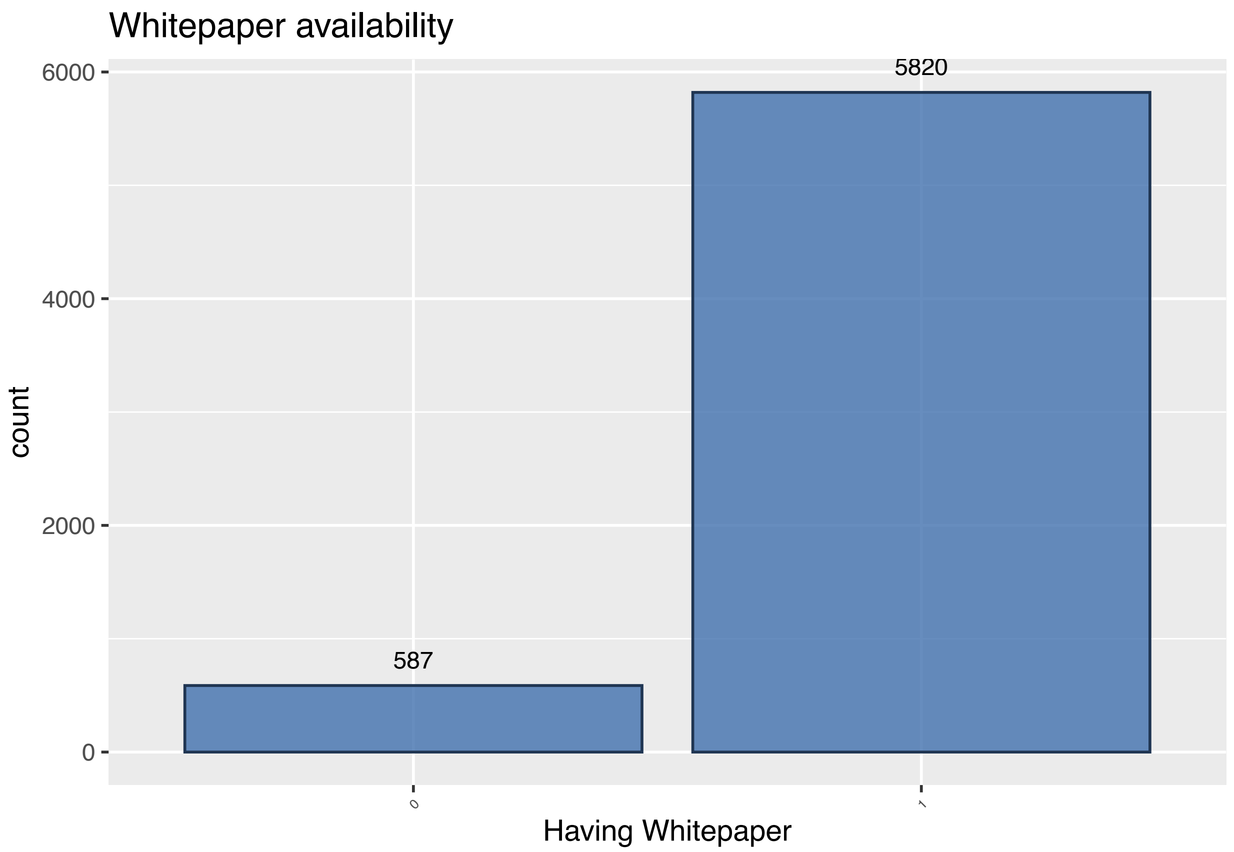
```
ggplot(t, aes(fill=success, y=count, x=erc20)) +
    geom_bar(position="dodge", stat="identity") +
  coord_polar("y", start=0) +
  ggtitle("ERC20 causing success pie plot")
```



### 8.1.10 Whitepaper

In other studies, was possible to notice how this paper has a crucial relevance in determining ICO's success. So, in the first graph we can notice that the majority of ICOs projects have a whitepaper. Clearly, also the successful ICOs are higher when the stratup write a proper whitepaper.

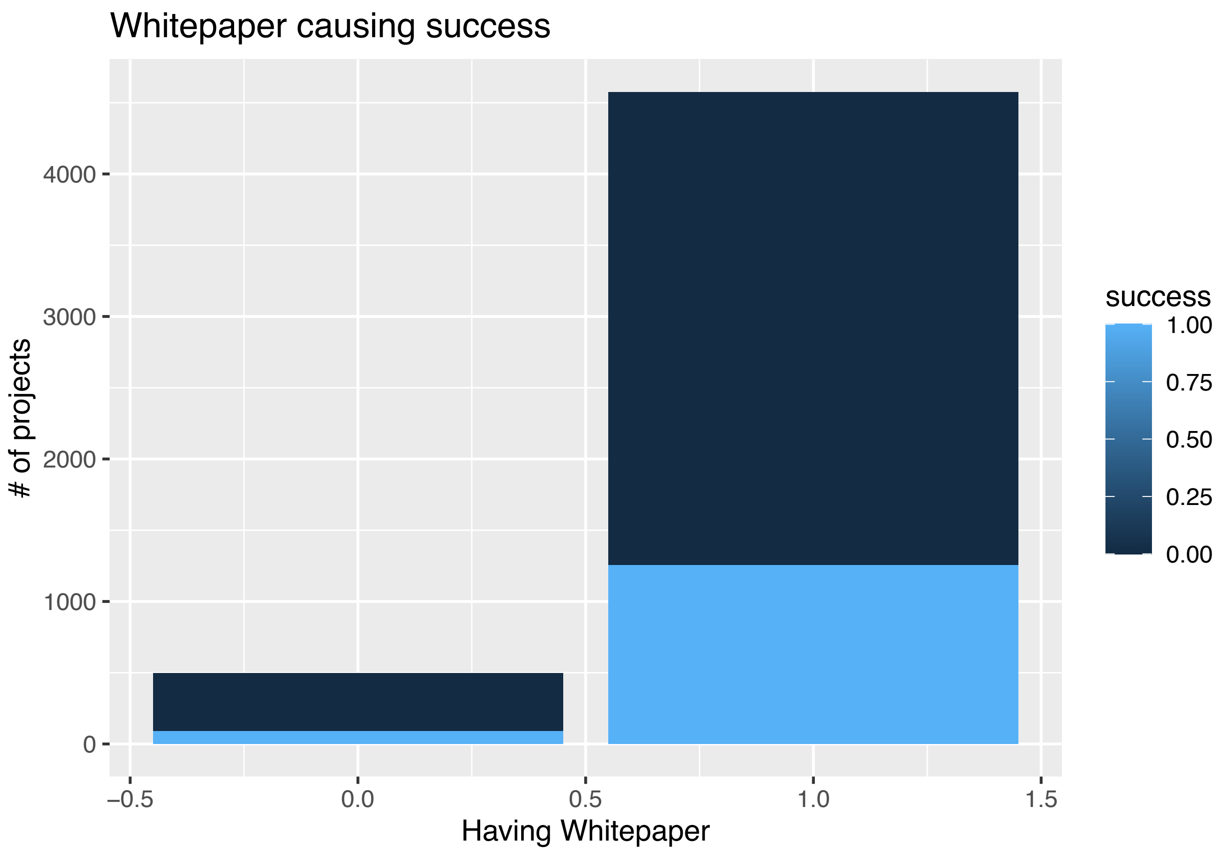
```r
t <- tord_db %>% dplyr::select(ico_name, link_white_paper) %>%
  distinct(ico_name, .keep_all = TRUE)


ggplot(t, aes(factor(link_white_paper))) +
  geom_bar(fill="#4271AE", color="#1F3552", alpha=0.8) +
  xlab("Having Whitepaper") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size=5)) +
  ggtitle("Whitepaper availability") +
  geom_text(aes(label = after_stat(count)),
            stat = "count", vjust = -1, colour = "black", size=3)
```



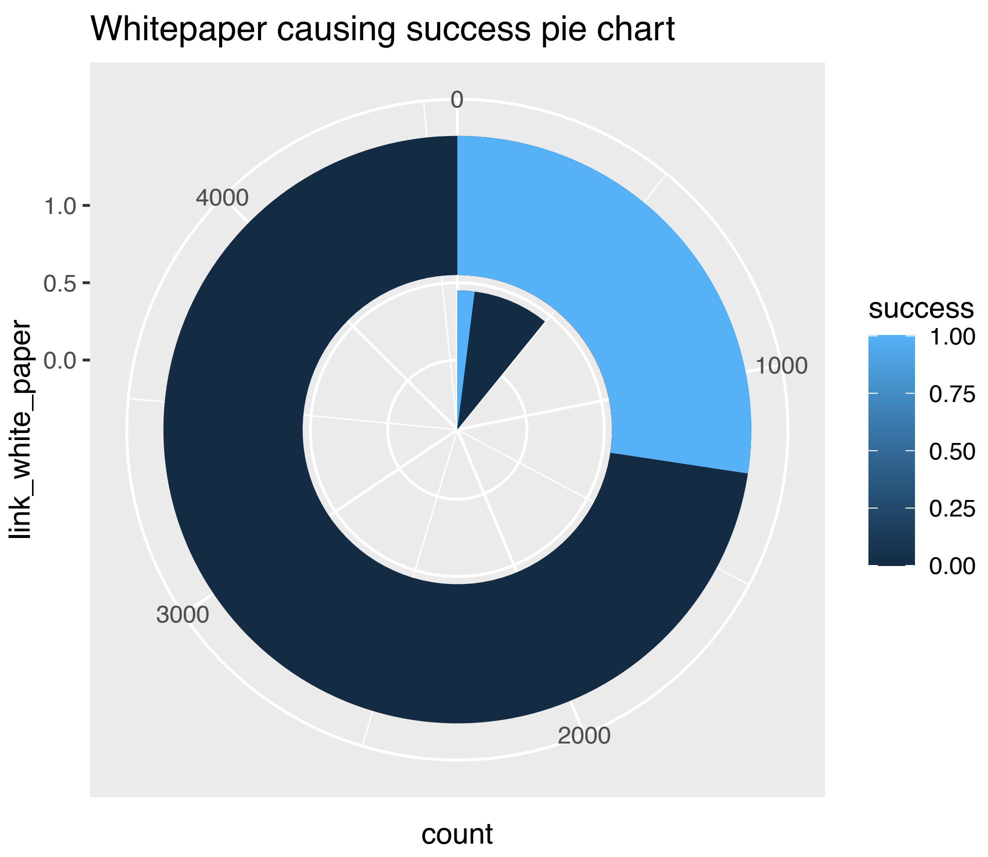
```
t <- tord_db %>%
    group_by(success, link_white_paper) %>%
    summarise(count = n())
```

```
## `summarise()` has grouped output by 'success'. You can override using the
## `.groups` argument.
```

```
ggplot(t, aes(fill=success, y=count, x=link_white_paper)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# of projects") + xlab("Having Whitepaper") +
  ggtitle("Whitepaper causing success")
```



```
ggplot(t, aes(fill=success, y=count, x=link_white_paper)) +
    geom_bar(position="dodge", stat="identity") +
  coord_polar("y", start=0) +
  ggtitle("Whitepaper causing success pie chart")
```



### 8.1.11 Social

The main social platflorms that an ICO project can be listed are: LinkedIn, GitHub and having a website. The following graphs shows those 3 social platforms, and how those social can retrive success. Clearly, LinkedIn seems to be the most crucial social network to have, because fewer ICO project have a profile, but the ratio of failure and succesful ICO is higher compared to GitHub or having a website.

```
# LinkedIn
t <- tord_db %>% dplyr::select(ico_name, linkedin_link) %>%
  distinct(ico_name, .keep_all = TRUE)


g1 <- ggplot(t, aes(factor(linkedin_link))) +
  geom_bar(fill="#4271AE", color="#1F3552", alpha=0.8) +
  xlab("Having LinkedIn") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size=5)) +
  ggtitle("LinkedIn availability") +
  geom_text(aes(label = after_stat(count)),
            stat = "count", vjust = -1, colour = "black", size=3)

t <- tord_db %>% group_by(success, linkedin_link) %>% summarise(count = n())
```

```
## `summarise()` has grouped output by 'success'. You can override using the
## `.groups` argument.
```

```
g2 <- ggplot(t, aes(fill=success, y=count, x=linkedin_link)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# of projects") + xlab("Having LinkedIn") +
  ggtitle("LinkedIn causing success")
```

```
# GitHub
t <- tord_db %>% dplyr::select(ico_name, github_link) %>%
  distinct(ico_name, .keep_all = TRUE)


g3 <- ggplot(t, aes(factor(github_link))) +
  geom_bar(fill="#4271AE", color="#1F3552", alpha=0.8) +
  xlab("Having GitHub") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size=5)) +
  ggtitle("GitHub availability") +
  geom_text(aes(label = after_stat(count)),
            stat = "count", vjust = -1, colour = "black", size=3)
t <- tord_db %>%
    group_by(success, github_link) %>%
    summarise(count = n())
```

```
## `summarise()` has grouped output by 'success'. You can override using the
## `.groups` argument.
```

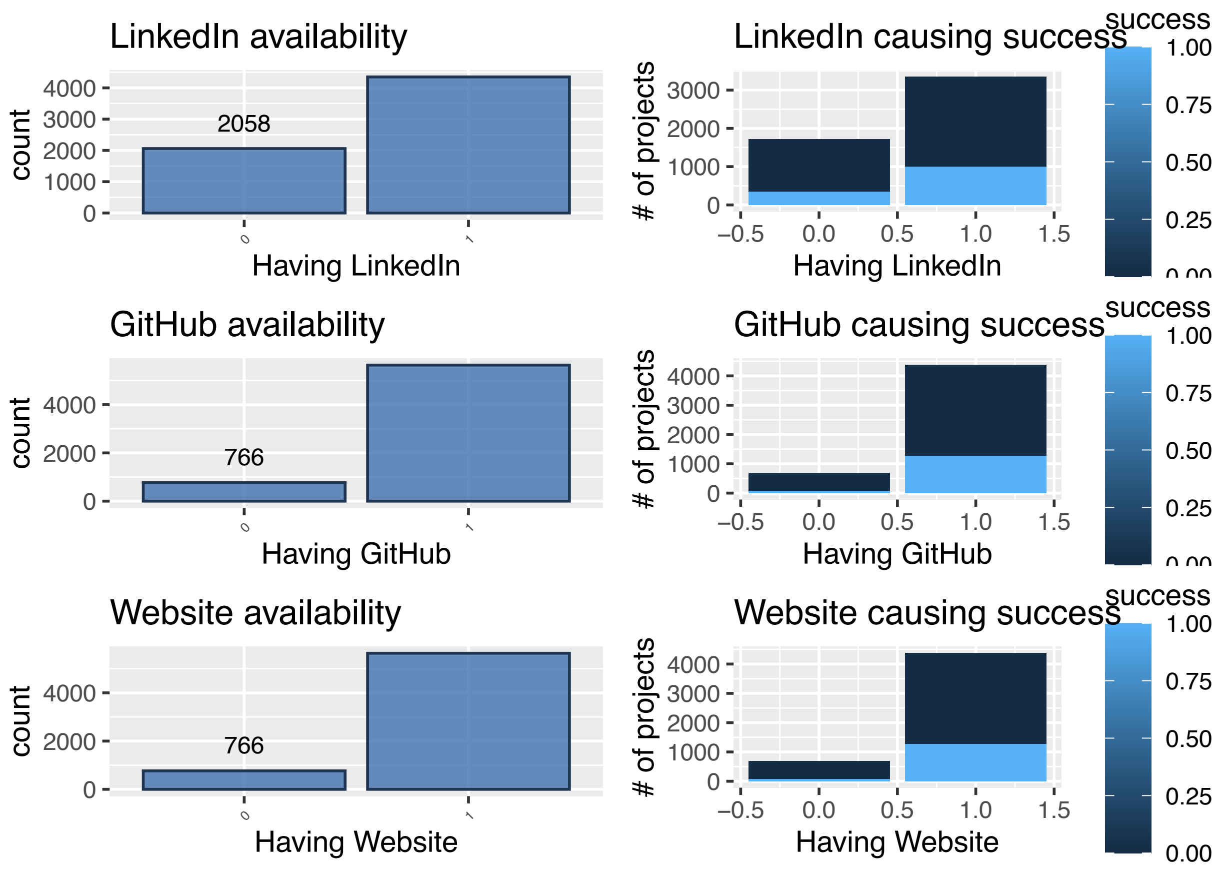
```
g4 <- ggplot(t, aes(fill=success, y=count, x=github_link)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# of projects") + xlab("Having GitHub") +
  ggtitle("GitHub causing success")
```

```
# website
t <- tord_db %>% dplyr::select(ico_name, website) %>%
  distinct(ico_name, .keep_all = TRUE)


g5 <- ggplot(t, aes(factor(website))) +
  geom_bar(fill="#4271AE", color="#1F3552", alpha=0.8) +
  xlab("Having Website") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size=5)) +
  ggtitle("Website availability") +
  geom_text(aes(label = after_stat(count)),
            stat = "count", vjust = -1, colour = "black", size=3)
t <- tord_db %>%
    group_by(success, website) %>%
    summarise(count = n())
```

```
## `summarise()` has grouped output by 'success'. You can override using the
## `.groups` argument.
```

```
g6 <- ggplot(t, aes(fill=success, y=count, x=website)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# of projects") + xlab("Having Website") +
  ggtitle("Website causing success")
```

```
grid.arrange(g1, g2, g3, g4, g5, g6, ncol=2)
```

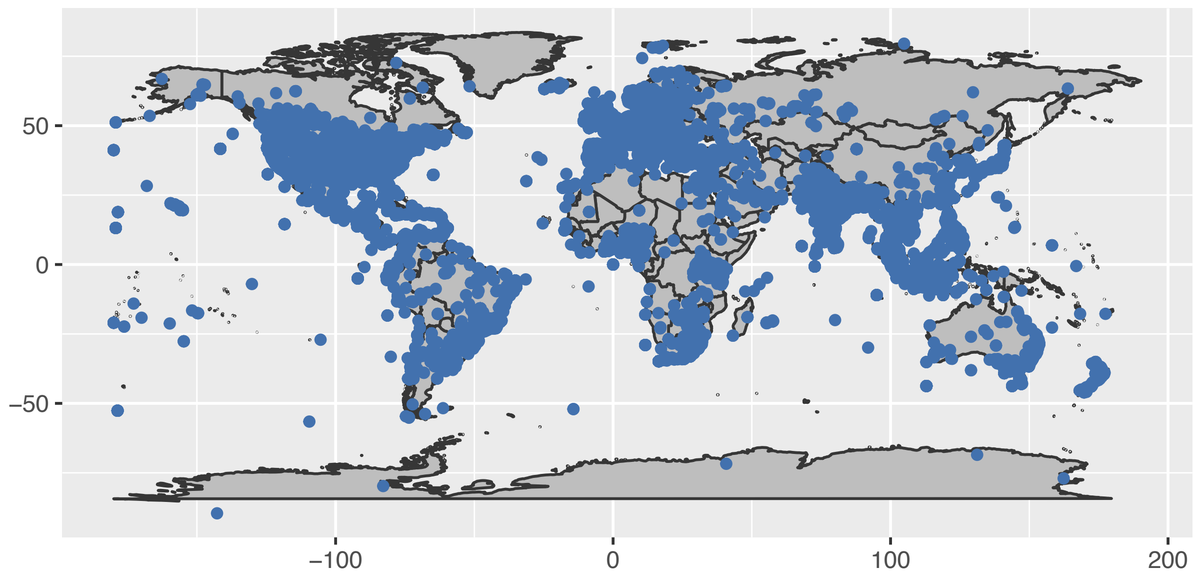


```
rm(g1, g2, g3, g4, g5, g6, t)
```

## 8.2 Geography Map Analysis

Is now time to use the *place_db* dataframe. Hence, with this data I can plot some world maps that will represent where the tweets are posted. ### World Map In this first graph, even if is a bit messy, we can notice that the majority of tweets are from USA, Europe and India or other countries located in the east part of the world.

```
world_map <- map_data("world")
place_db %>%
  # need to swap longitude & latitude
  # changing the names of longitude & latitude to x & y just for clarity
  rename(x = longitude, y = latitude) %>%
  # somehow blue points have y values flipped
  ggplot() +
  geom_polygon(aes(x = long, y = lat, group = group),
               data = world_map, fill = "grey", color = "grey21") +
  geom_point(aes(x = x, y = y), color="#4271AE") +
  scale_color_identity() +
  coord_fixed() +
  xlab("") +
  ylab("")
```



While in the following map showing like an heatmap the main location of interest. At first glance, is obsious that the area aroung New York and London are the most interesting ones.

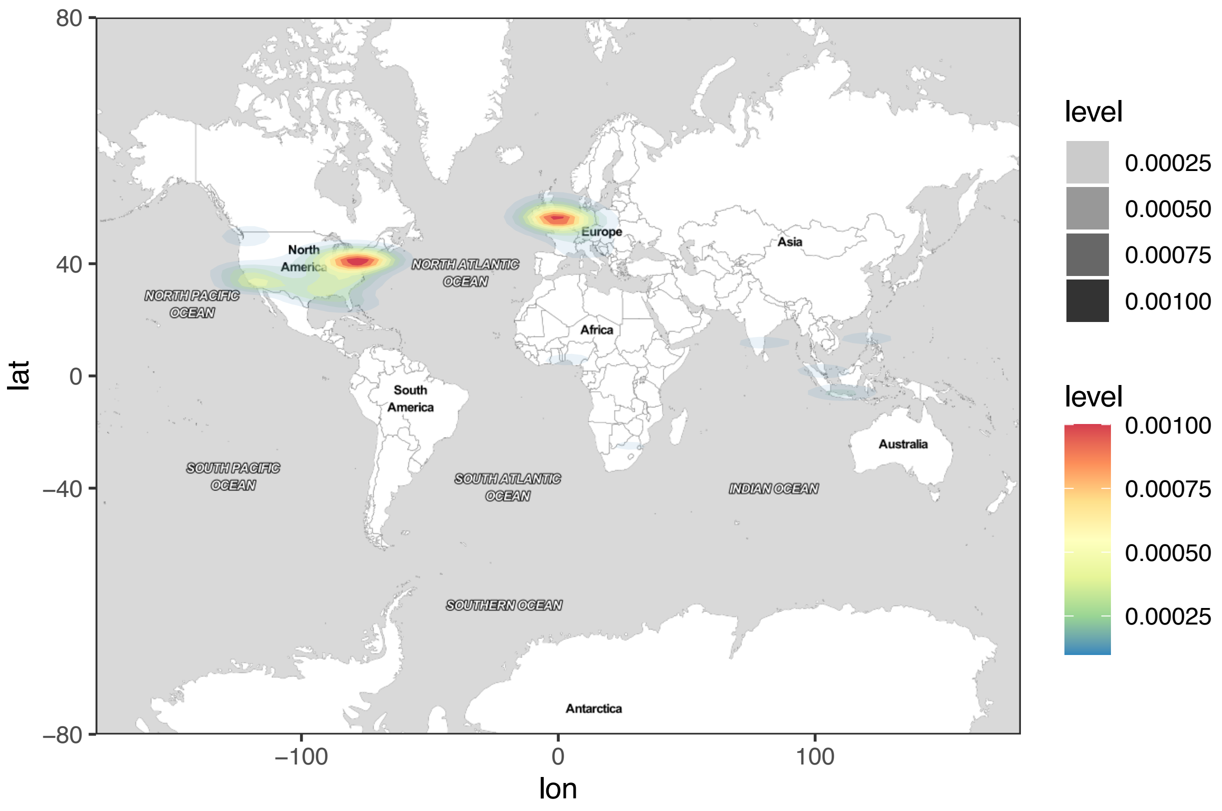
```
map_bounds <- c(left = -180, bottom = -80, right = 180, top = 80)

coords.map <- get_stamenmap(map_bounds, zoom = 2, maptype = "toner-lite")
```

```
## i Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.
```

```
## Service Unavailable (HTTP 503). Failed to acquire tile /toner-lite/
## 2/4/0.png.Service Unavailable (HTTP 503). Failed to acquire tile /toner-lite/
## 2/4/1.png.Service Unavailable (HTTP 503). Failed to acquire tile /toner-lite/
## 2/4/2.png.Service Unavailable (HTTP 503). Failed to acquire tile /toner-lite/
## 2/4/3.png.
```

```
coords.map <- ggmap(coords.map, extent="device", legend="none")
coords.map <- coords.map +
  stat_density2d(data=place_db,
                 aes(x=longitude,
                     y=latitude,
                     fill=..level..,
                     alpha=..level..), geom="polygon")

coords.map <- coords.map +
  scale_fill_gradientn(colours=rev(brewer.pal(7, "Spectral")))

coords.map <- coords.map + theme_bw()
coords.map
```

```
## Warning: Removed 1 rows containing non-finite values (`stat_density2d()`).
```



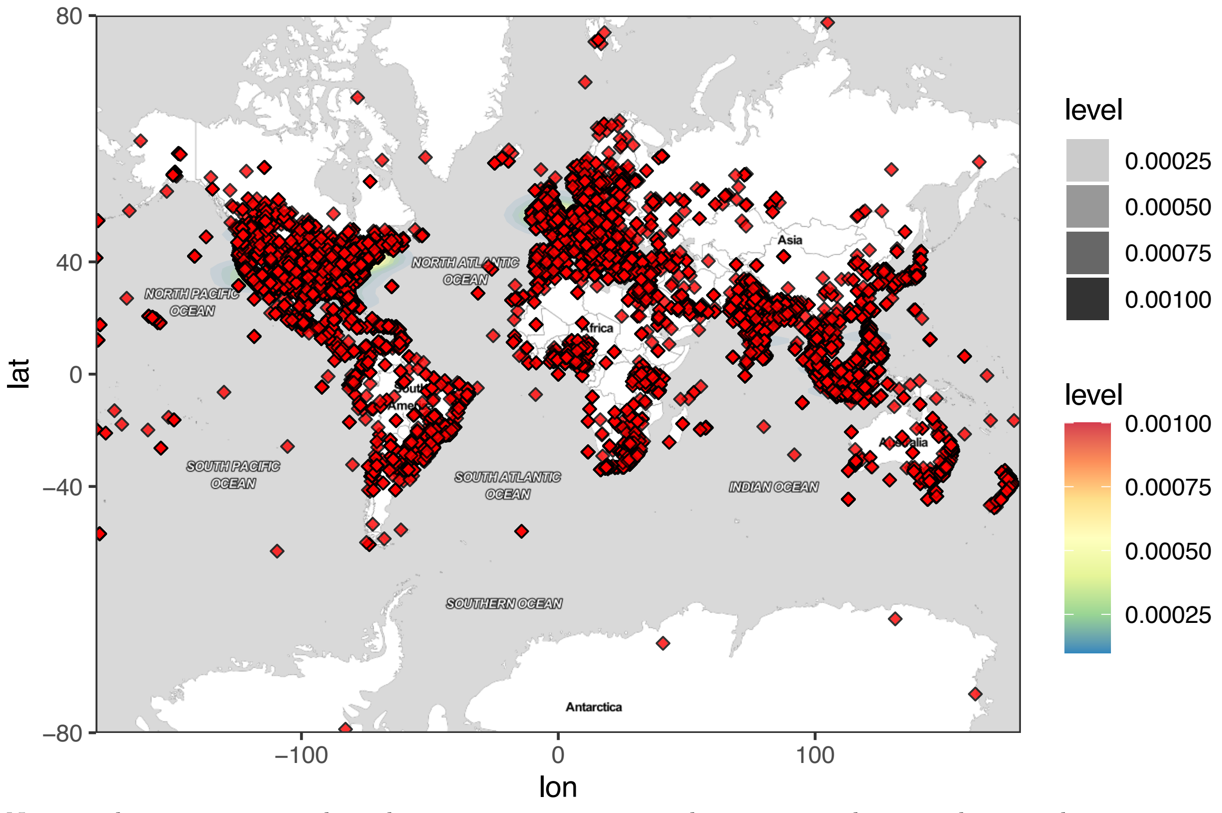
```
#ggsave(filename="./coords.png")
```

Consecutively, I can also add other points to my map in order to locate the tweets production. This graph is almost like the first world map graph that I plotted.

```
coords.map + geom_point(data=place_db,
                        aes(x=longitude, y=latitude),
                        fill="red", shape=23, alpha=0.8)
```

```
## Warning: Removed 1 rows containing non-finite values (`stat_density2d()`).
```

```
## Warning: Removed 1 rows containing missing values (`geom_point()`).
```



Now can be interesting analyse the seven continents, in order to zoom a bit into the main locations.

### 8.2.1 Maps per continents

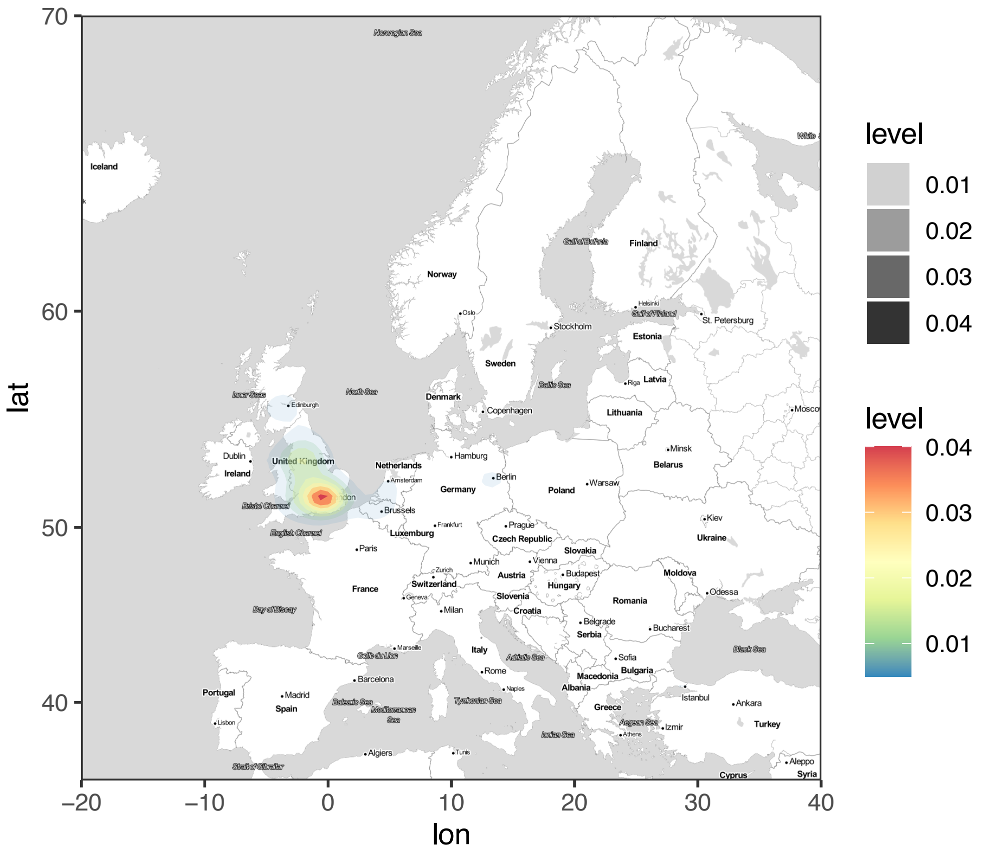
**8.2.1.1 Europe** First of all, the european situation is pretty clear, and the the most of tweets are posted from UK, more precisely around London. Anyway, if we watch carefully we can notice that even also in the other European Countries there is an evident interest.

```
# Europe
map_bounds <- c(left = -20, bottom = 35, right = 40, top = 70)

coords.map <- get_stamenmap(map_bounds, zoom = 5, maptype = "toner-lite")
```

```
## i Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.
```

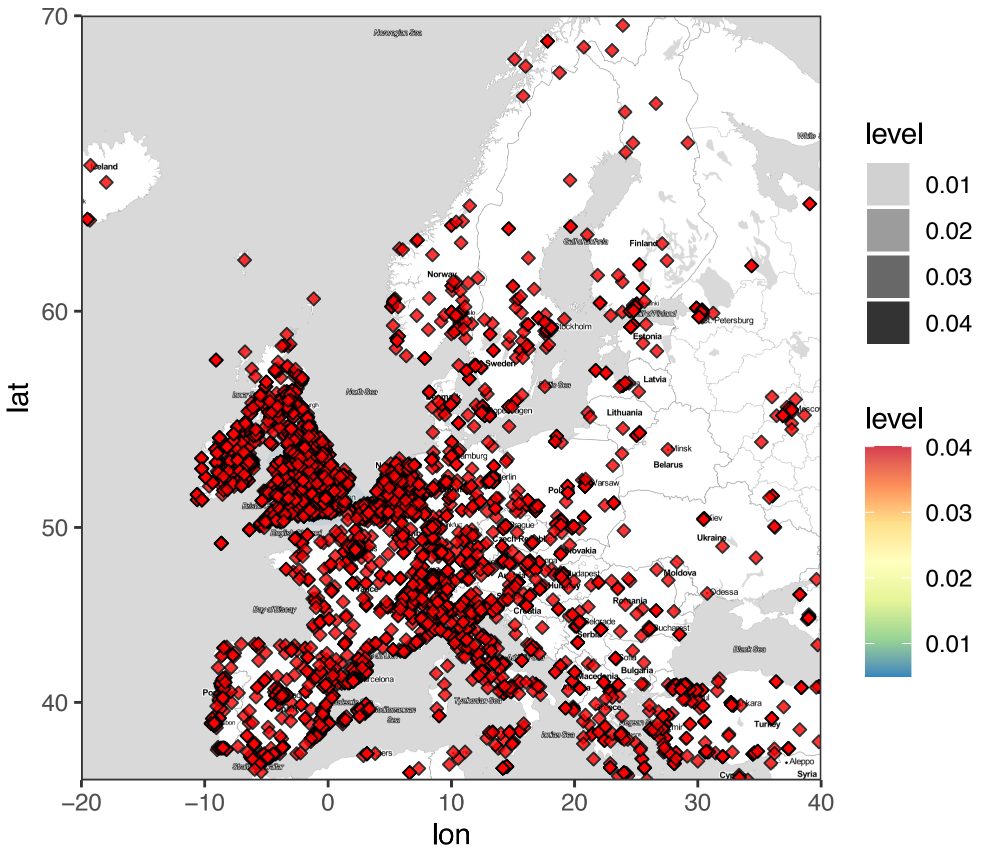
```
coords.map <- ggmap(coords.map, extent="device", legend="none")
coords.map <- coords.map +
  stat_density2d(data=place_db,
                 aes(x=longitude,
                     y=latitude,
                     fill=..level..,
                     alpha=..level..), geom="polygon")
coords.map <- coords.map +
  scale_fill_gradientn(colours=rev(brewer.pal(7, "Spectral")))

coords.map <- coords.map + theme_bw()
coords.map
```

```
## Warning: Removed 63610 rows containing non-finite values (`stat_density2d()`).
```



```
# Points graph
coords.map + geom_point(data=place_db,  aes(x=longitude, y=latitude), fill="red", shape=23, alpha=0.8)
```

```
## Warning: Removed 63610 rows containing non-finite values (`stat_density2d()`).
## Warning: Removed 63610 rows containing missing values (`geom_point()`).
```



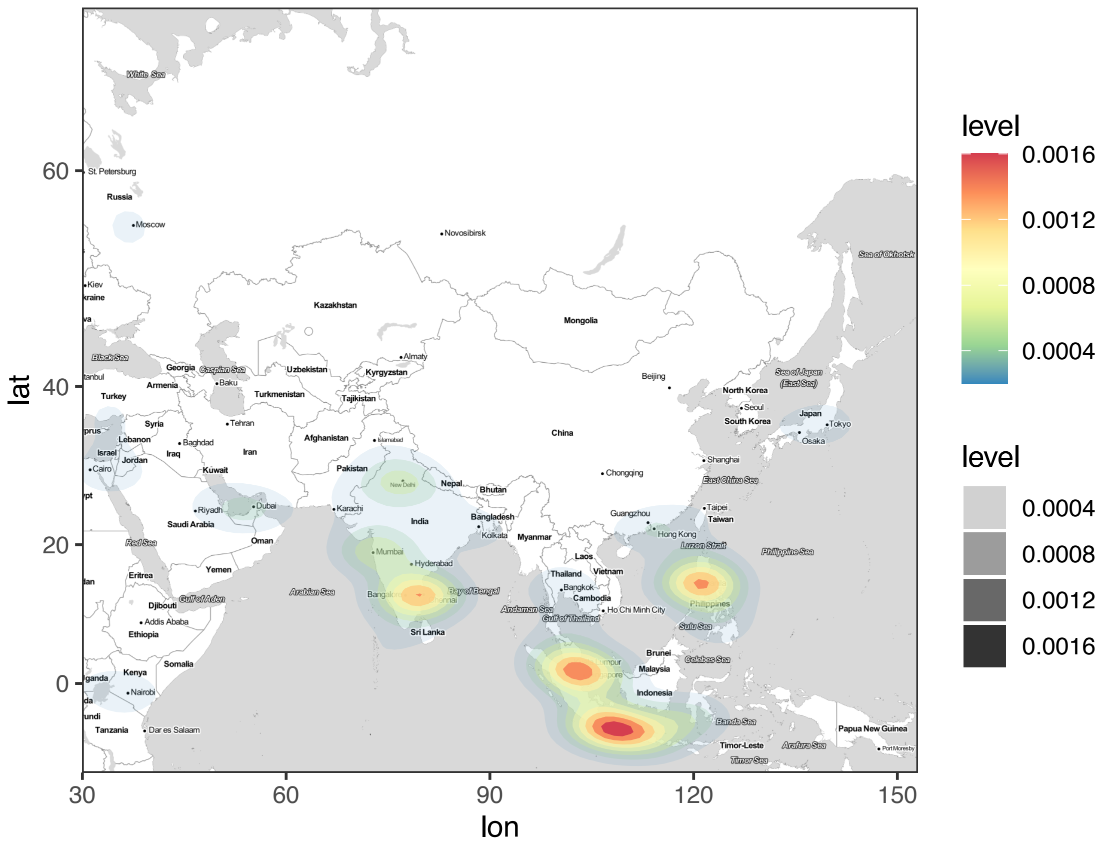
**8.2.1.2 Asia** Then in the case of Asia, in the following graphs is obvious that in Philippines were really interest in tweeting arguments about ICOs.

```r
#Asia
map_bounds <- c(left = 30, bottom = -13, right = 153, top = 70)

coords.map <- get_stamenmap(map_bounds, zoom = 4, maptype = "toner-lite")
```

```
## i Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.
```

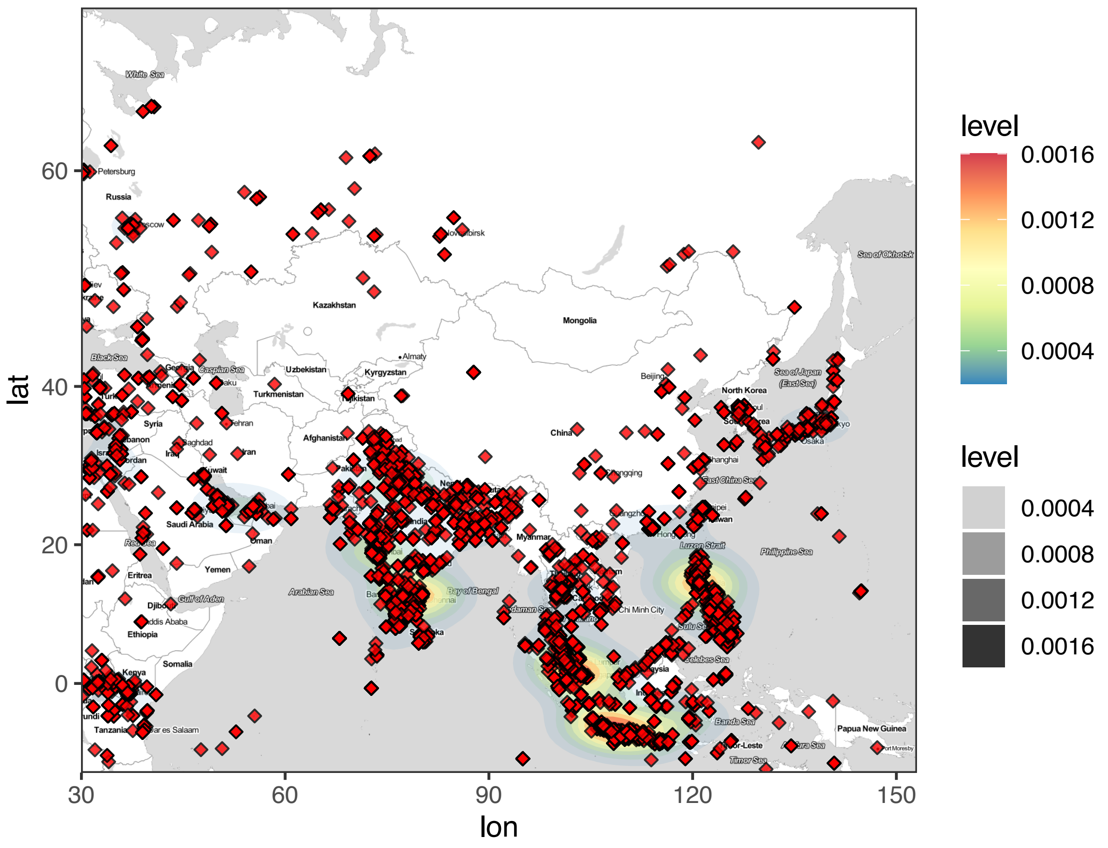
```r
coords.map <- ggmap(coords.map, extent="device", legend="none")
coords.map <- coords.map +
  stat_density2d(data=place_db,
                 aes(x=longitude,
                     y=latitude,
                     fill=..level..,
                     alpha=..level..), geom="polygon")

coords.map <- coords.map +
  scale_fill_gradientn(colours=rev(brewer.pal(7, "Spectral")))

coords.map <- coords.map + theme_bw()
coords.map
```

```
## Warning: Removed 66082 rows containing non-finite values (`stat_density2d()`).
```



```
# Points graph
coords.map + geom_point(data=place_db,  aes(x=longitude, y=latitude), fill="red", shape=23, alpha=0.8)
```

```
## Warning: Removed 66082 rows containing non-finite values (`stat_density2d()`).

## Warning: Removed 66082 rows containing missing values (`geom_point()`).
```



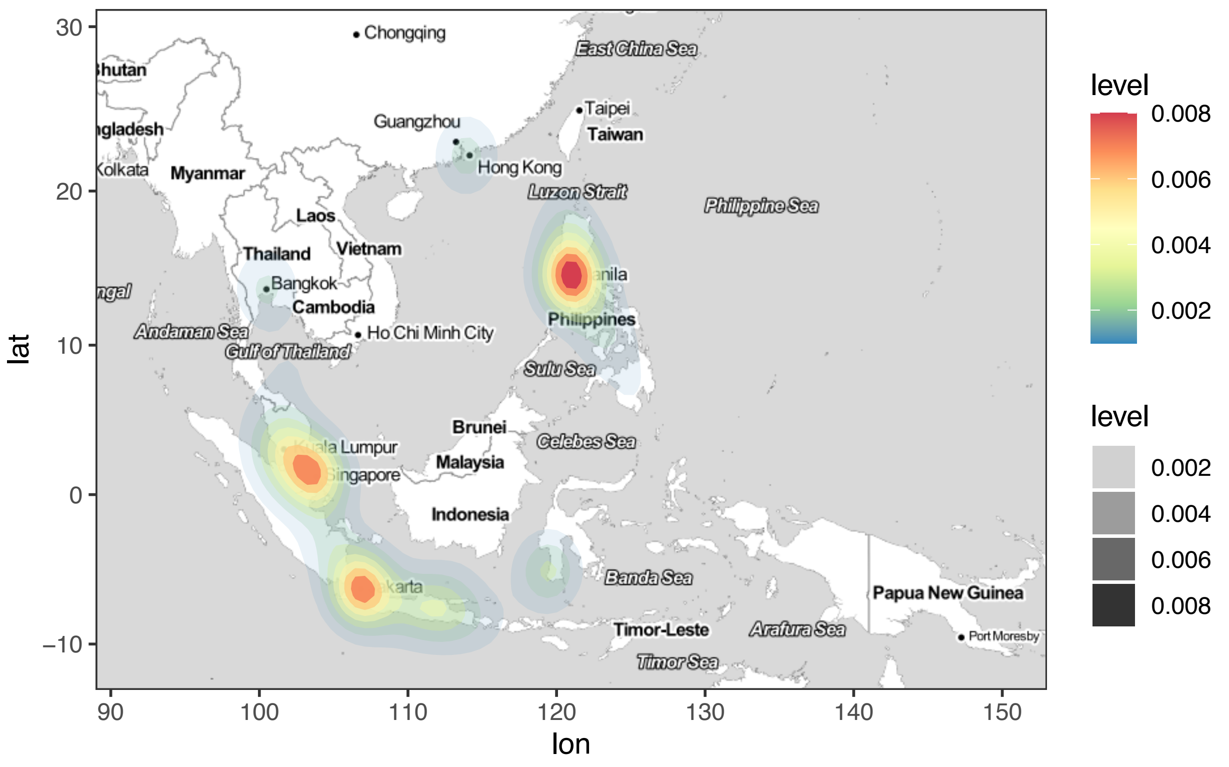
**8.2.1.2.1 South Est Asia** Going in detail where there is more concentration.

```
# South Est Asia
map_bounds <- c(left = 89, bottom = -13, right = 153, top = 31)

coords.map <- get_stamenmap(map_bounds, zoom = 4, maptype = "toner-lite")
```

```
## i Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.
```

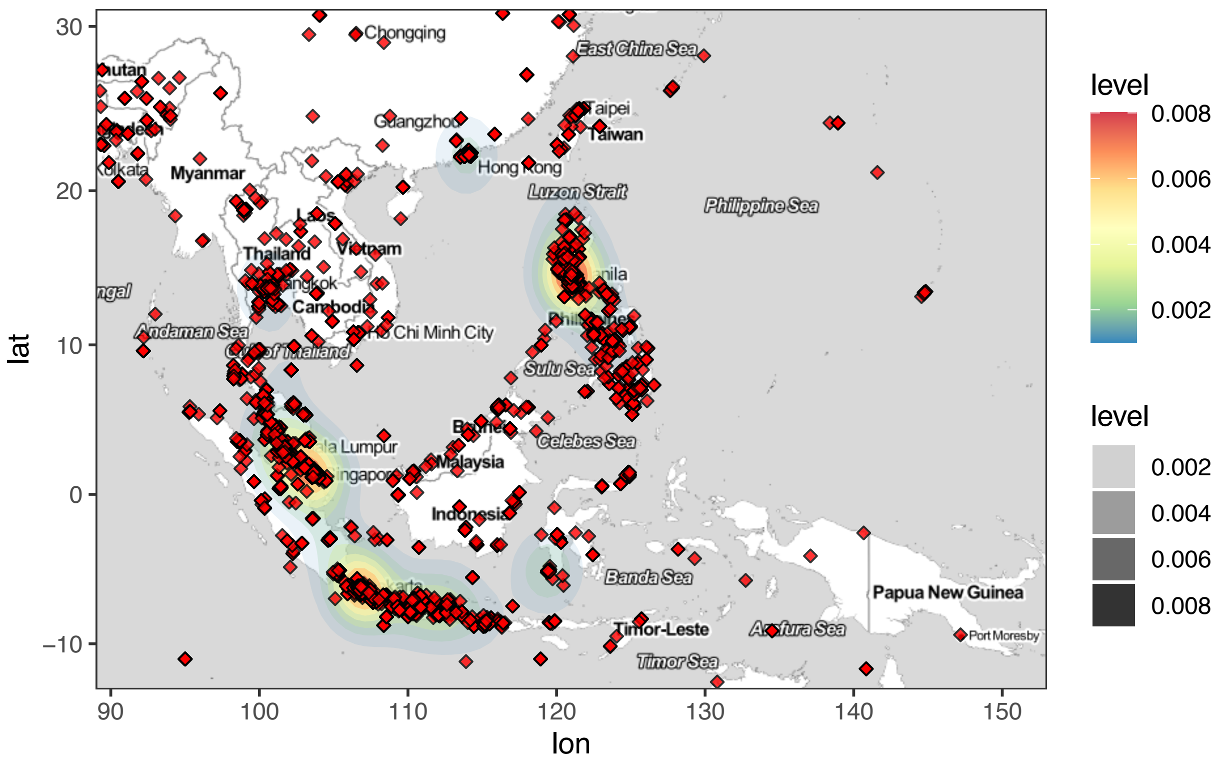
```
coords.map <- ggmap(coords.map, extent="device", legend="none")
coords.map <- coords.map +
  stat_density2d(data=place_db,
                 aes(x=longitude,
                     y=latitude,
                     fill=..level..,
                     alpha=..level..), geom="polygon")
coords.map <- coords.map +
  scale_fill_gradientn(colours=rev(brewer.pal(7, "Spectral")))

coords.map <- coords.map + theme_bw()
coords.map
```

```
## Warning: Removed 73529 rows containing non-finite values (`stat_density2d()`).
```



```r
# Points graph
coords.map + geom_point(data=place_db,  aes(x=longitude, y=latitude), fill="red", shape=23, alpha=0.8)
```

```
## Warning: Removed 73529 rows containing non-finite values (`stat_density2d()`).
## Warning: Removed 73529 rows containing missing values (`geom_point()`).
```



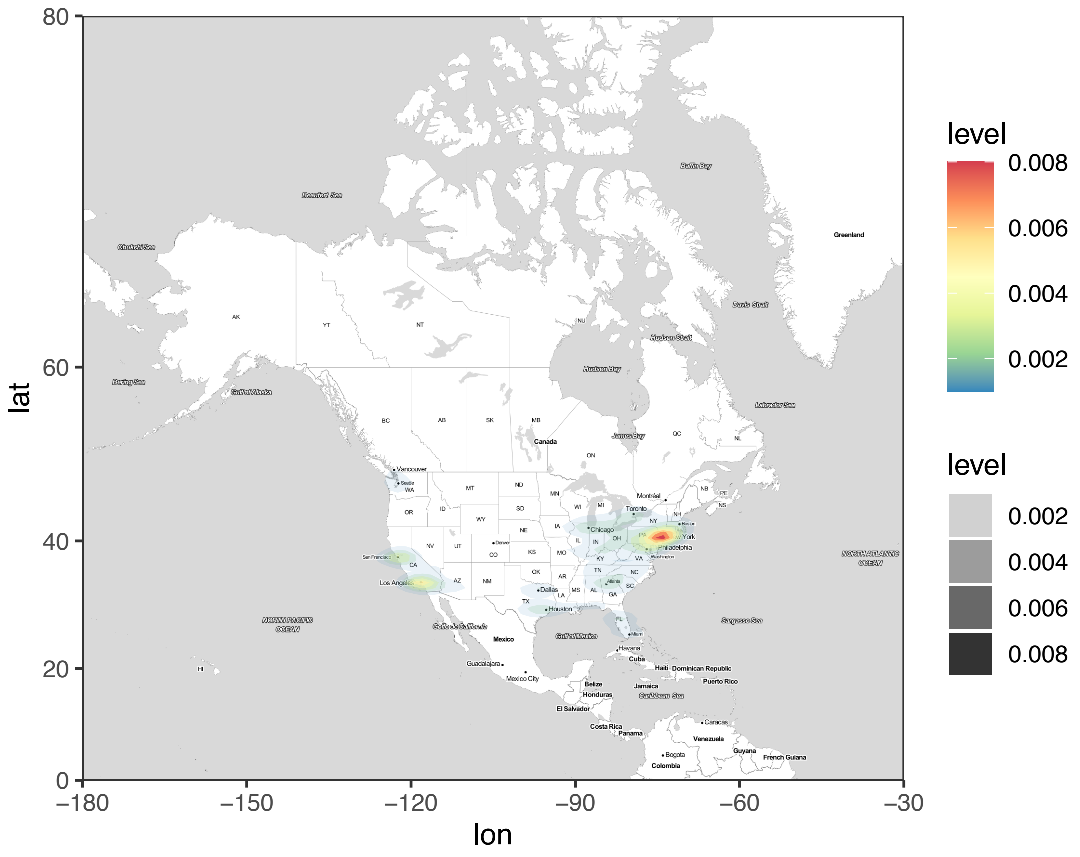
### 8.2.2 Nord America

While in the case of North America the highest concentration of tweets is, as said before, locate around New York.

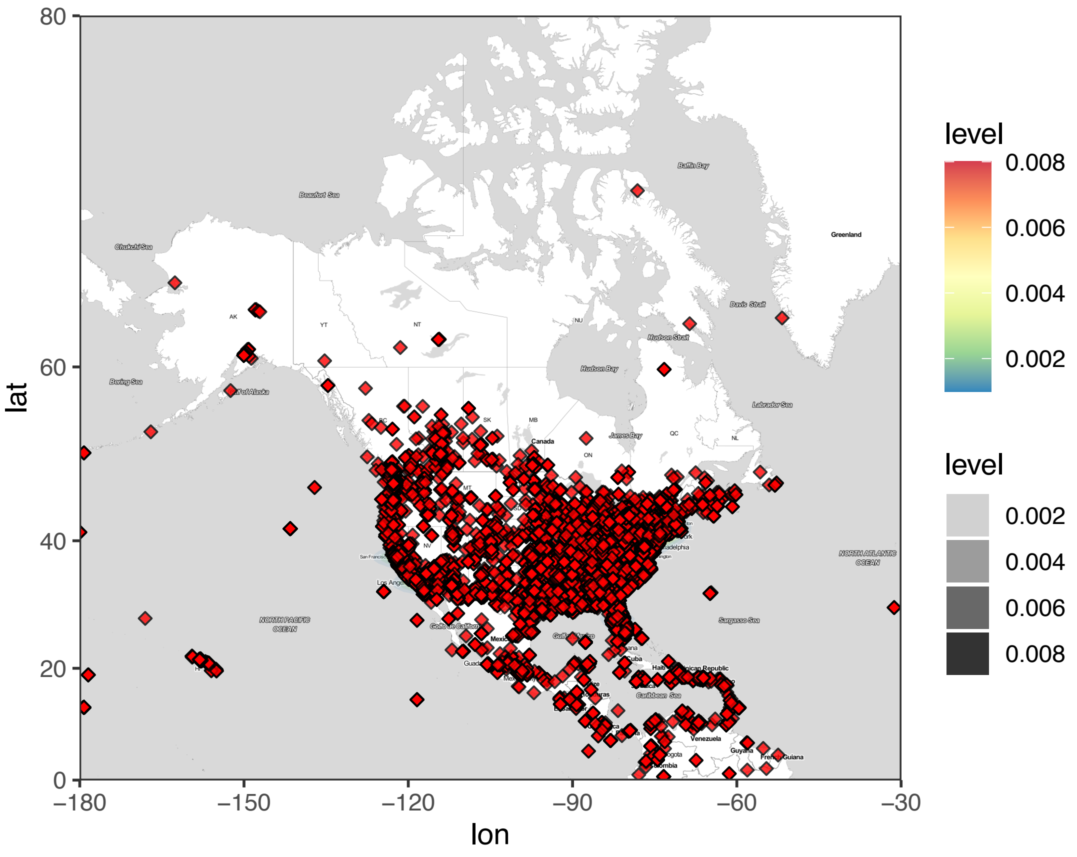
```
# North America
map_bounds <- c(left = -180, bottom = 0, right = -30, top = 80)

coords.map <- get_stamenmap(map_bounds, zoom = 4, maptype = "toner-lite")
```

```
## i Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.

## i 56 tiles needed, this may take a while (try a smaller zoom?)
```

```
coords.map <- ggmap(coords.map, extent="device", legend="none")
coords.map <- coords.map +
  stat_density2d(data=place_db,
                 aes(x=longitude,
                     y=latitude,
                     fill=..level.., alpha=..level..), geom="polygon")
coords.map <- coords.map +
  scale_fill_gradientn(colours=rev(brewer.pal(7, "Spectral")))

coords.map <- coords.map + theme_bw()
coords.map
```



```
# Points graph
coords.map + geom_point(data=place_db,  aes(x=longitude, y=latitude), fill="red", shape=23, alpha=0.8)
```



### 8.2.3 South America

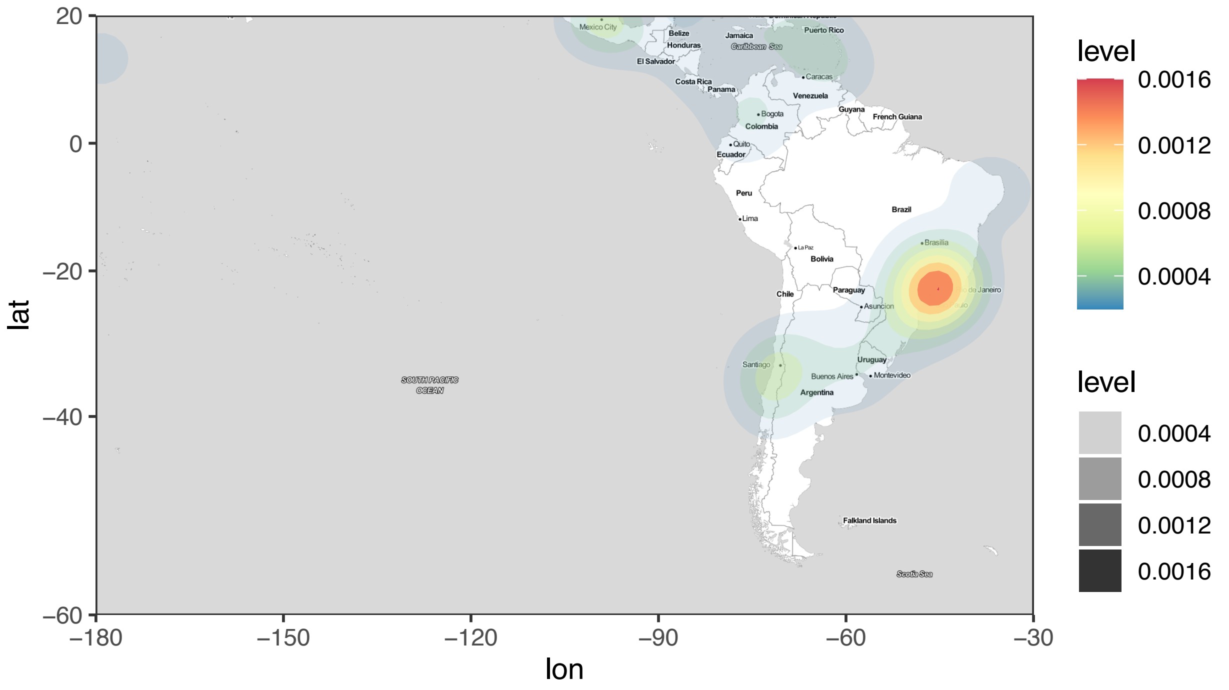
In the case of South America, the majority of tweets is produced in Rio de Janeiro.

```
# South America
map_bounds <- c(left = -180, bottom = -60, right = -30, top = 20)

coords.map <- get_stamenmap(map_bounds, zoom = 4, maptype = "toner-lite")
```

```
## i Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.
```

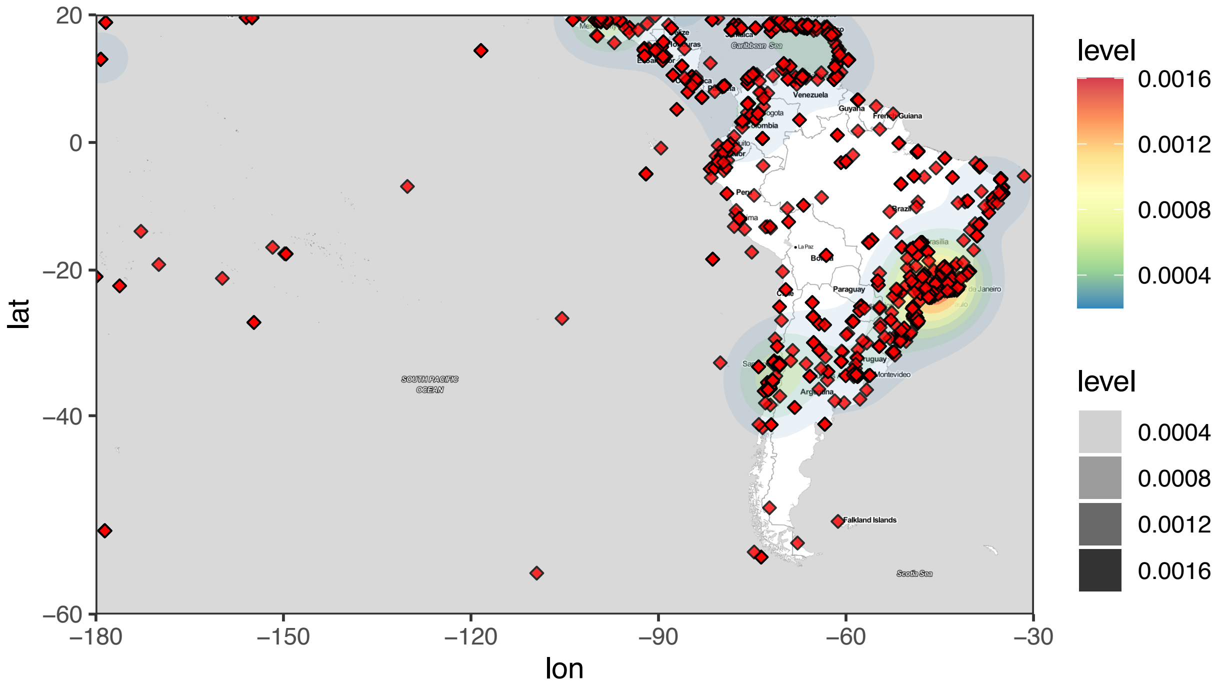
```
coords.map <- ggmap(coords.map, extent="device", legend="none")
coords.map <- coords.map +
  stat_density2d(data=place_db,
                 aes(x=longitude,
                     y=latitude,
                     fill=..level..,
                     alpha=..level..), geom="polygon")
coords.map <- coords.map +
  scale_fill_gradientn(colours=rev(brewer.pal(7, "Spectral")))

coords.map <- coords.map + theme_bw()
coords.map
```

```
## Warning: Removed 79224 rows containing non-finite values (`stat_density2d()`).
```



```
# Points graph
coords.map + geom_point(data=place_db,  aes(x=longitude, y=latitude), fill="red", shape=23, alpha=0.8)
```

```
## Warning: Removed 79224 rows containing non-finite values (`stat_density2d()`).

## Warning: Removed 79224 rows containing missing values (`geom_point()`).
```



### 8.2.4 Africa

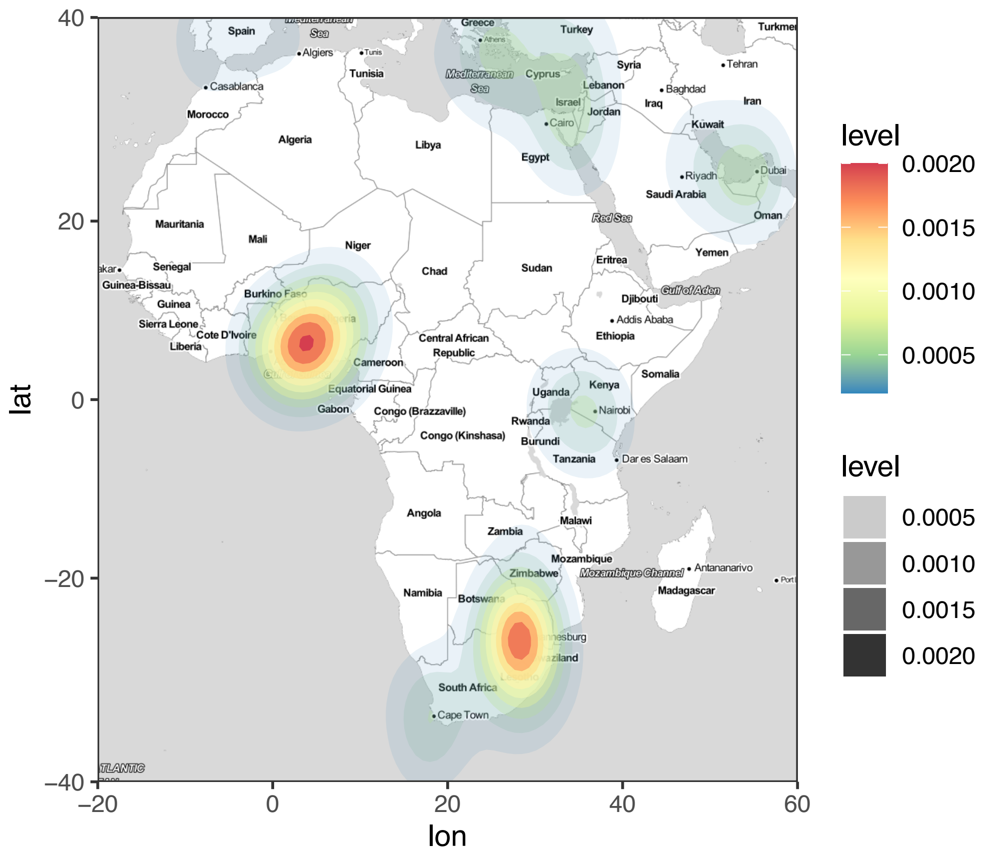
While in the case of Africa the main locations are Johannesburg and Lagos.

```
# Africa
map_bounds <- c(left = -20, bottom = -40, right = 60, top = 40)

coords.map <- get_stamenmap(map_bounds, zoom = 4, maptype = "toner-lite")
```

```
## i Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.
```

```
coords.map <- ggmap(coords.map, extent="device", legend="none")
coords.map <- coords.map +
  stat_density2d(data=place_db,
                aes(x=longitude,
                    y=latitude,
                    fill=..level.., alpha=..level..), geom="polygon")
coords.map <- coords.map +
  scale_fill_gradientn(colours=rev(brewer.pal(7, "Spectral")))

coords.map <- coords.map + theme_bw()
coords.map
```

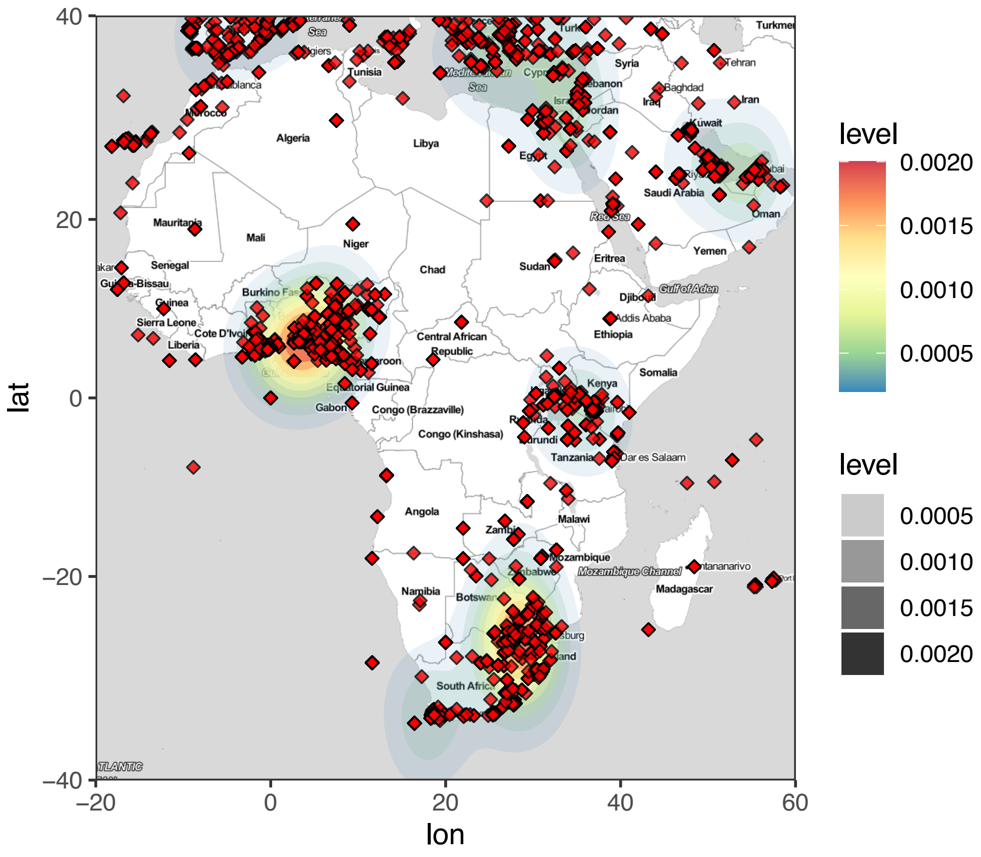
```
## Warning: Removed 75360 rows containing non-finite values (`stat_density2d()`).
```



```
# Points graph
coords.map + geom_point(data=place_db,  aes(x=longitude, y=latitude), fill="red", shape=23, alpha=0.8)
```

```
## Warning: Removed 75360 rows containing non-finite values (`stat_density2d()`).
```

```
## Warning: Removed 75360 rows containing missing values (`geom_point()`).
```



### 8.2.5 Oceania

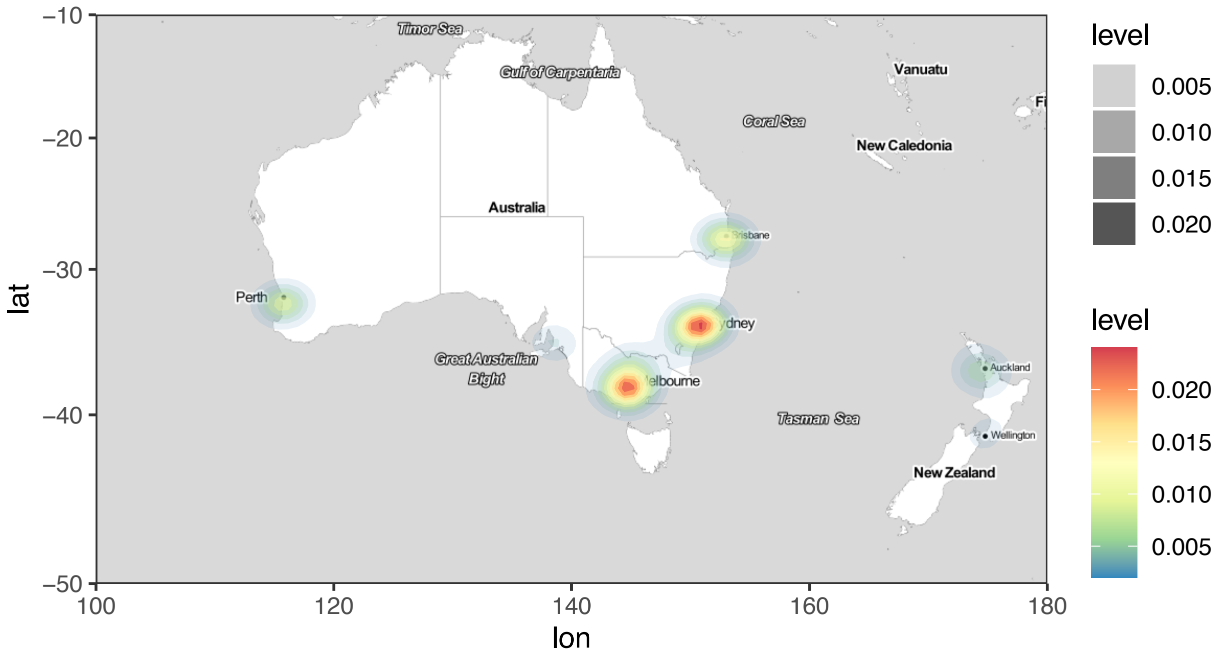
Hence, in the case of Oceania the main tweets where from Melbourne and Sydney.

```
# Oceania
map_bounds <- c(left = 100, bottom = -50, right = 180, top = -10)

coords.map <- get_stamenmap(map_bounds, zoom = 4, maptype = "toner-lite")
```

```
## i Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.

## Service Unavailable (HTTP 503). Failed to acquire tile /toner-lite/
## 4/16/8.png.Service Unavailable (HTTP 503). Failed to acquire tile /toner-lite/
## 4/16/9.png.Service Unavailable (HTTP 503). Failed to acquire tile /toner-lite/
## 4/16/10.png.
```

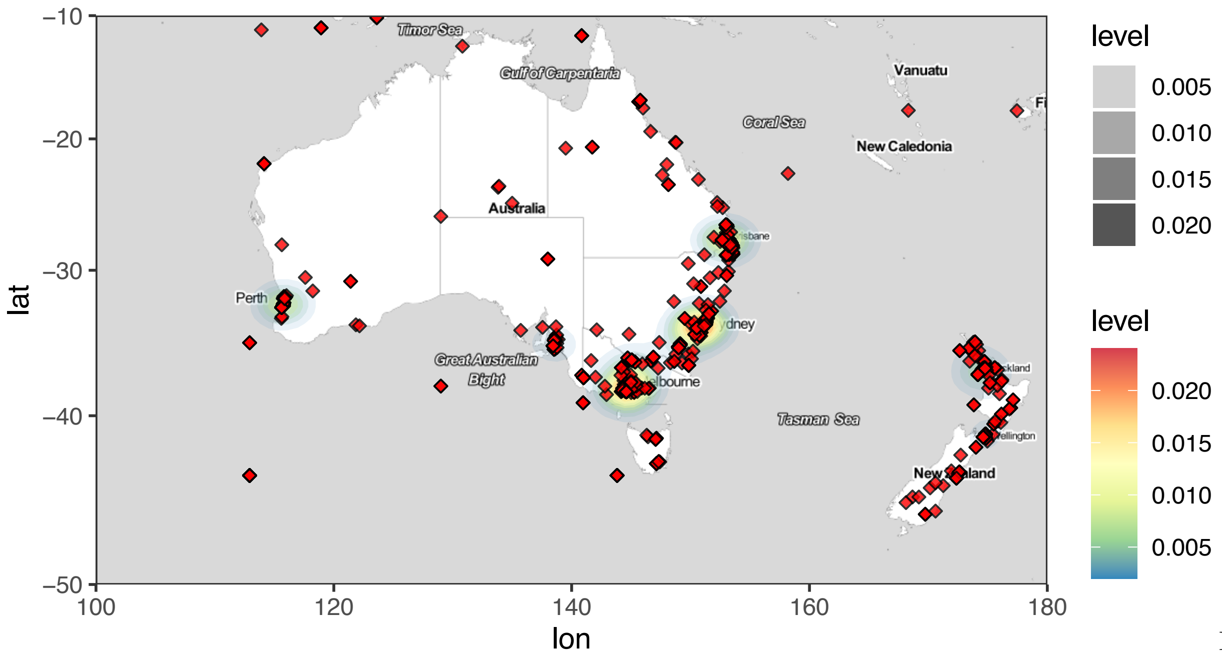
```
coords.map <- ggmap(coords.map, extent="device", legend="none")
coords.map <- coords.map +
  stat_density2d(data=place_db,
               aes(x=longitude,
                   y=latitude,
                   fill=..level.., alpha=..level..), geom="polygon")

coords.map <- coords.map +
  scale_fill_gradientn(colours=rev(brewer.pal(7, "Spectral")))

coords.map <- coords.map + theme_bw()
coords.map
```

```
## Warning: Removed 80527 rows containing non-finite values (`stat_density2d()`).
```



```
# Points graph
coords.map + geom_point(data=place_db,  aes(x=longitude, y=latitude), fill="red", shape=23, alpha=0.8)
```

```
## Warning: Removed 80527 rows containing non-finite values (`stat_density2d()`).

## Warning: Removed 80527 rows containing missing values (`geom_point()`).
```



Re-
moving unwanted variables

```
rm(coords.map, world_map, map_bounds, temp)
```

## 8.3 Twitter DB Analysis

### 8.3.1 Adding the success variable

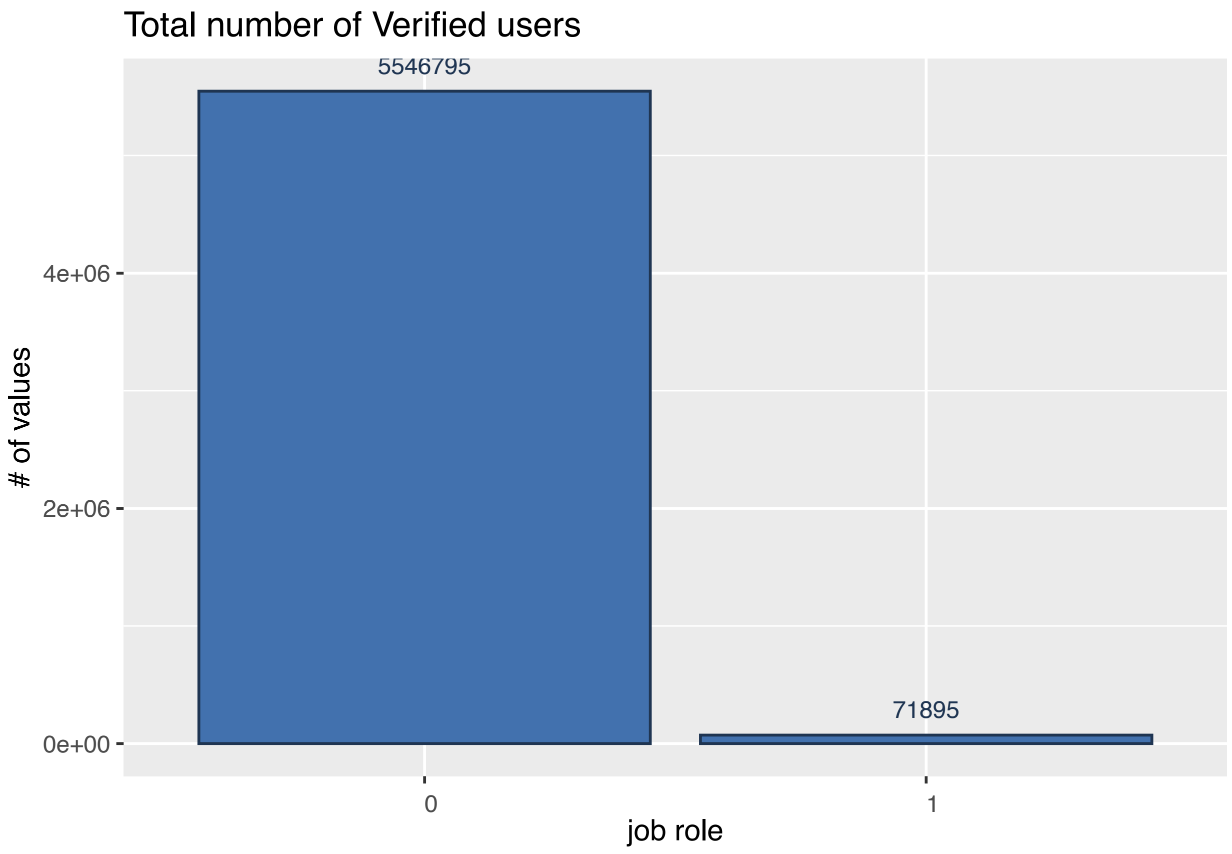
From *tord_db* I will add for each ICO the if it is successful or not.

```
# t <- tord_db %>% select(ico_name, success)
# twitter_db <- inner_join(test, t, by="ico_name")
t <- tord_db %>% dplyr::select(ico_name, success)
twitter_db<- inner_join(twitter_db, t, by="ico_name")
```

### 8.3.2 Verified accounts

First of all, we can notice that the vast mojirity of accounts are not verified. So, the verified account can clearly have an higher impact of influence.

```
t <- twitter_db %>% tabyl(verified)

ggplot(t, aes(x = reorder(verified, -n), y = n)) +
  geom_bar(stat = "identity", fill = "#4271AE", colour = "#1F3552") +
  theme(axis.text.x=element_text(angle=0,hjust=0,vjust=0)) +
  geom_text(aes(label = n), vjust = -1, colour = "#1F3552", size=3)+
  labs(y= "# of values", x = "job role") +
  ggtitle("Total number of Verified users")
```



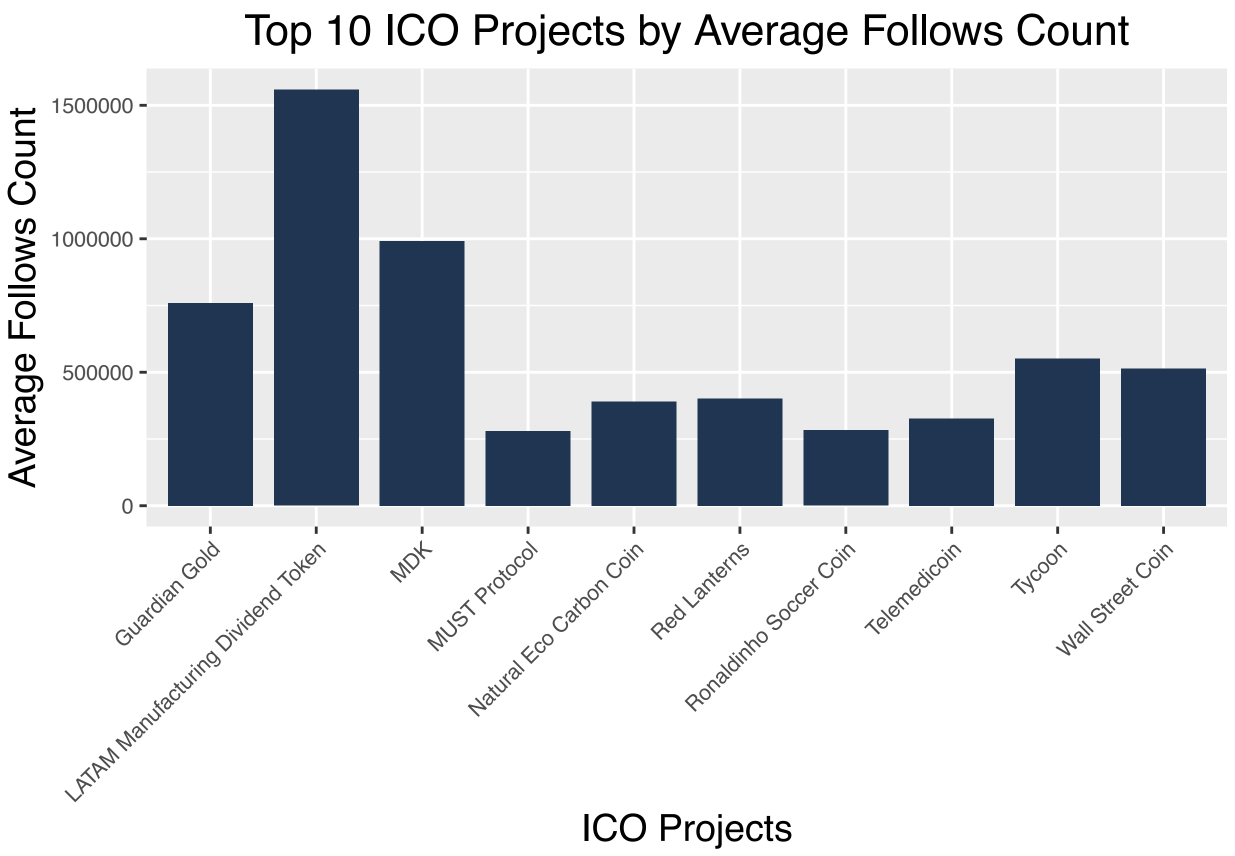
```
rm(t)
```

### 8.3.3 Follows count

Due the complication of the dataset, and the large amount of ICOs involved I decided to group the first and last 10 ICOs where the follows count visible in the graphs is the average of each follows count related to a user who posted a tweet.

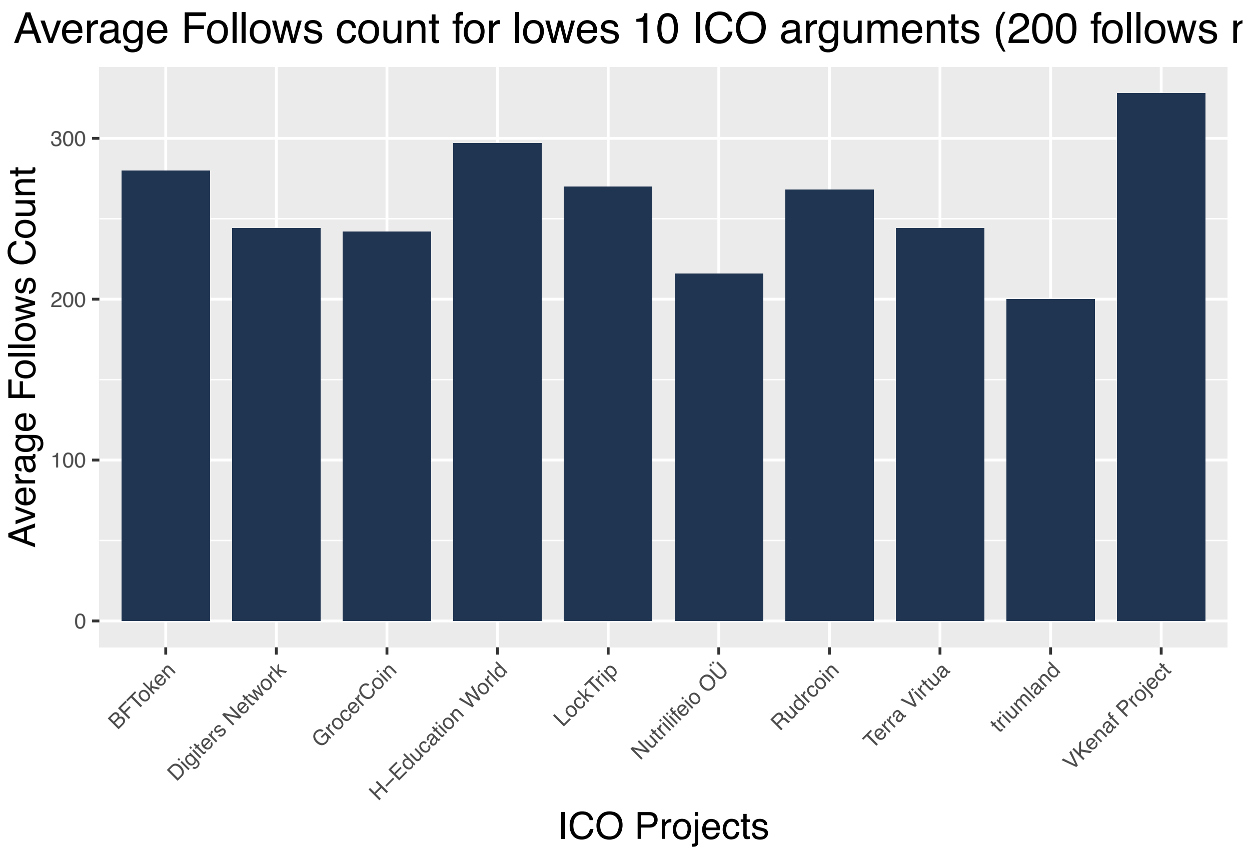
There fore, in this first graph is obvious that *LATAM* is more popular, and the follows count compared to the other ICOs is visible higher. Anyway, also the *DEAP* and the *MDK* seems to be more incline to success.

```
t <- twitter_db %>% dplyr::select(ico_name, follows_count) %>%
  filter(follows_count>=200) %>% group_by(ico_name) %>%
  summarize(average=mean(follows_count))

t$average <- round(t$average)


t %>%
  arrange(desc(average)) %>%
  top_n(10, average) %>%
  ggplot(aes(x = average, y = ico_name)) +
  geom_col(width = 0.8, fill = "#1F3552", show.legend = FALSE) +
  coord_flip() +
  theme(plot.title = element_text(size = 16, hjust = 0.5),
        axis.text.y = element_text(size = 8, hjust = 1),
        axis.text.x = element_text(size = 8, angle = 45, hjust = 1),
        axis.title = element_text(size = 14)) +
  labs(title = "Top 10 ICO Projects by Average Follows Count",
       x = "Average Follows Count",
       y = "ICO Projects")
```



Clearly, all the results above are more inclined to be successful compared to the following results, showing the lowest 10 values where the *follows_count* variable is set to a minimum of 200 followers. This may be pretty obvious to understand, because those ICOs where the average follows count is 0 are not interesting, and means that the popularity of the user who posted the tweet is approximately to 0. So, the tweet posted may be insignificant.

```r
t %>%
  arrange(desc(average)) %>%
  top_n(-10, average) %>%
  ggplot(aes(x = average, y = ico_name)) +
  geom_col(width = 0.8, fill = "#1F3552", show.legend = FALSE) +
  coord_flip() +
  theme(plot.title = element_text(size = 16, hjust = 0.5),
        axis.text.y = element_text(size = 8, hjust = 1),
        axis.text.x = element_text(size = 8, angle = 45, hjust = 1),
        axis.title = element_text(size = 14)) +
  labs(title = "Average Follows count for lowes 10 ICO arguments (200 follows min)",
       x = "Average Follows Count",
       y = "ICO Projects")
```

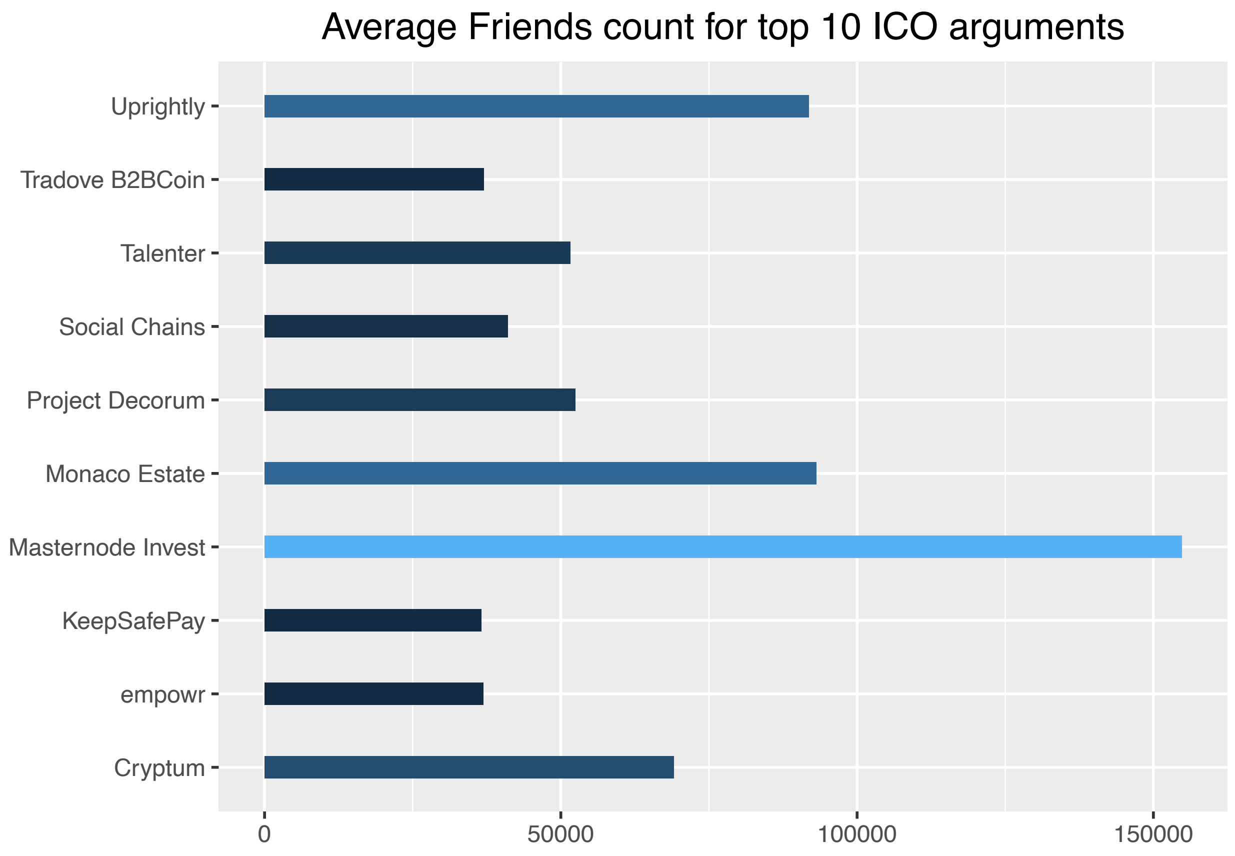


```
# free up some space...
rm(t)
```

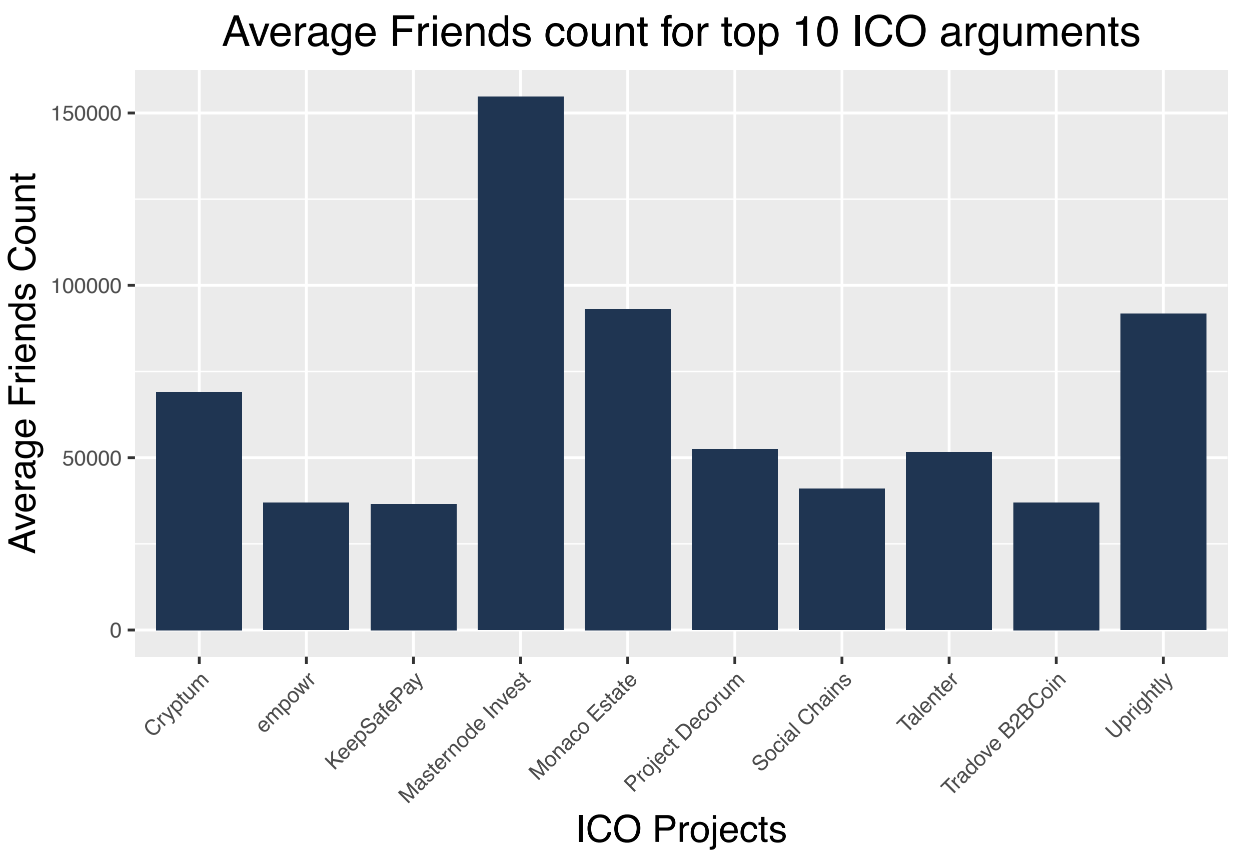
### 8.3.4 Friends Count

The same analysis done before, can be represented in the follwing lines of code. Clearly, the friends indicator, can be perceived and intended as less relevant compare the the followers one.

```
t = twitter_db %>%
  dplyr::select(ico_name, friends_count)  %>%
  group_by(ico_name) %>% summarize(average=mean(friends_count))

t$average <- round(t$average)

t %>% arrange(desc(average)) %>% top_n(10, average) %>% ggplot(aes(x = ico_name, y = average)) +
  geom_col(aes(fill = average), width = 0.3, show.legend = FALSE) +
  coord_flip() +
  theme(plot.title = element_text(size = 14, hjust = 0.5)) +
  labs(title = "Average Friends count for top 10 ICO arguments",
       x = NULL,
       y = NULL)
```



```
t %>%
  arrange(desc(average)) %>%
  top_n(10, average) %>%
  ggplot(aes(x = average, y = ico_name)) +
  geom_col(width = 0.8, fill = "#1F3552", show.legend = FALSE) +
  coord_flip() +
  theme(plot.title = element_text(size = 16, hjust = 0.5),
        axis.text.y = element_text(size = 8, hjust = 1),
        axis.text.x = element_text(size = 8, angle = 45, hjust = 1),
        axis.title = element_text(size = 14)) +
  labs(title = "Average Friends count for top 10 ICO arguments",
       x = "Average Friends Count",
       y = "ICO Projects")
```



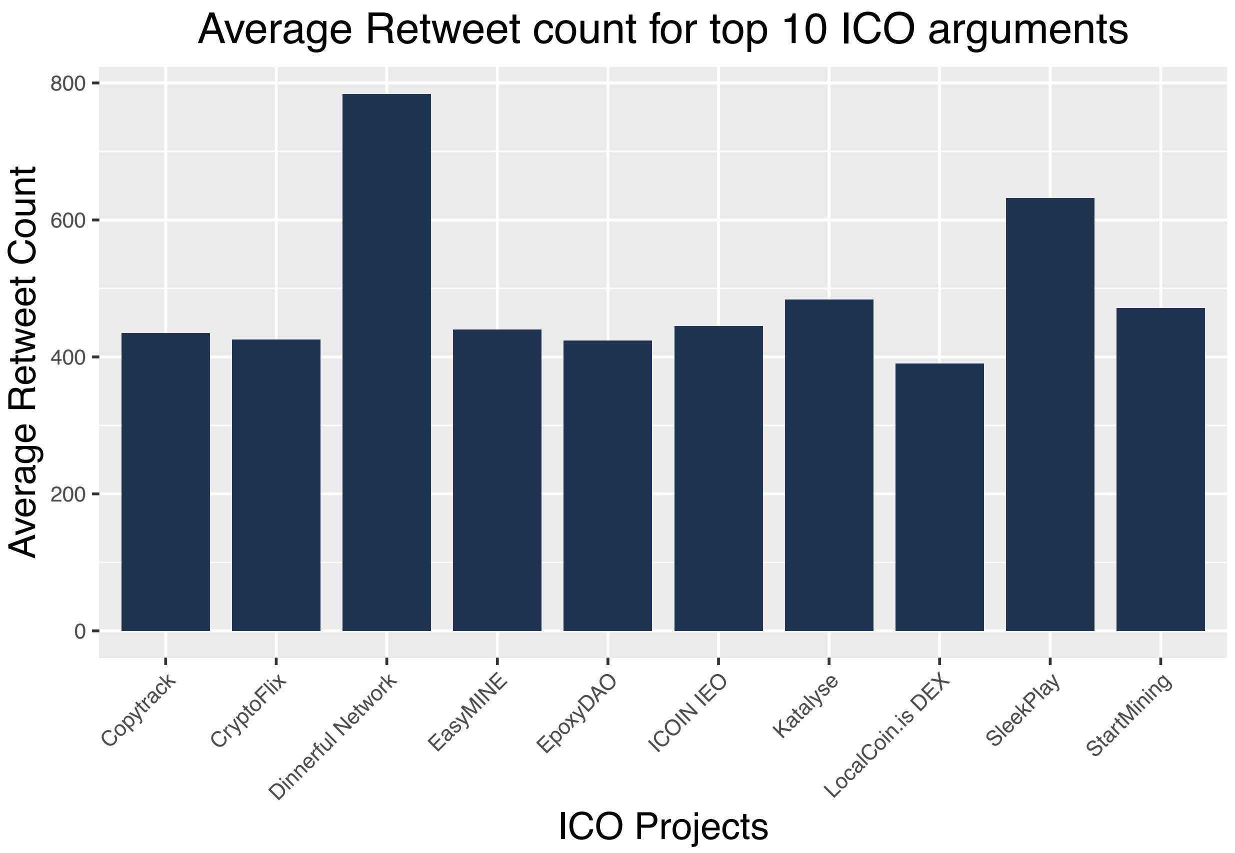
```r
# free up some space...
rm(t)
```

In this case showing the lowest 10 elements has no sense because they are all up to 0.

### 8.3.5 Retweet count

This graphical representation can show what are the most retweeted tweets, containing the ICO name inside. Clearly, this can be a nice indicator to understand the campaign and the popularity that this argument had during the ICO campaign.

```r
t = twitter_db %>%
  dplyr::select(ico_name, retweet_count) %>% filter(retweet_count > 0) %>%
  group_by(ico_name) %>% summarize(average=mean(retweet_count))
t$average <- round(t$average)


t %>%
  arrange(desc(average)) %>%
  top_n(10, average) %>%
  ggplot(aes(x = average, y = ico_name)) +
  geom_col(width = 0.8, fill = "#1F3552", show.legend = FALSE) +
  coord_flip() +
  theme(plot.title = element_text(size = 16, hjust = 0.5),
        axis.text.y = element_text(size = 8, hjust = 1),
        axis.text.x = element_text(size = 8, angle = 45, hjust = 1),
        axis.title = element_text(size = 14)) +
  labs(title = "Average Retweet count for top 10 ICO arguments",
       x = "Average Retweet Count",
       y = "ICO Projects")
```



```
# free up some space...
rm(t)
```

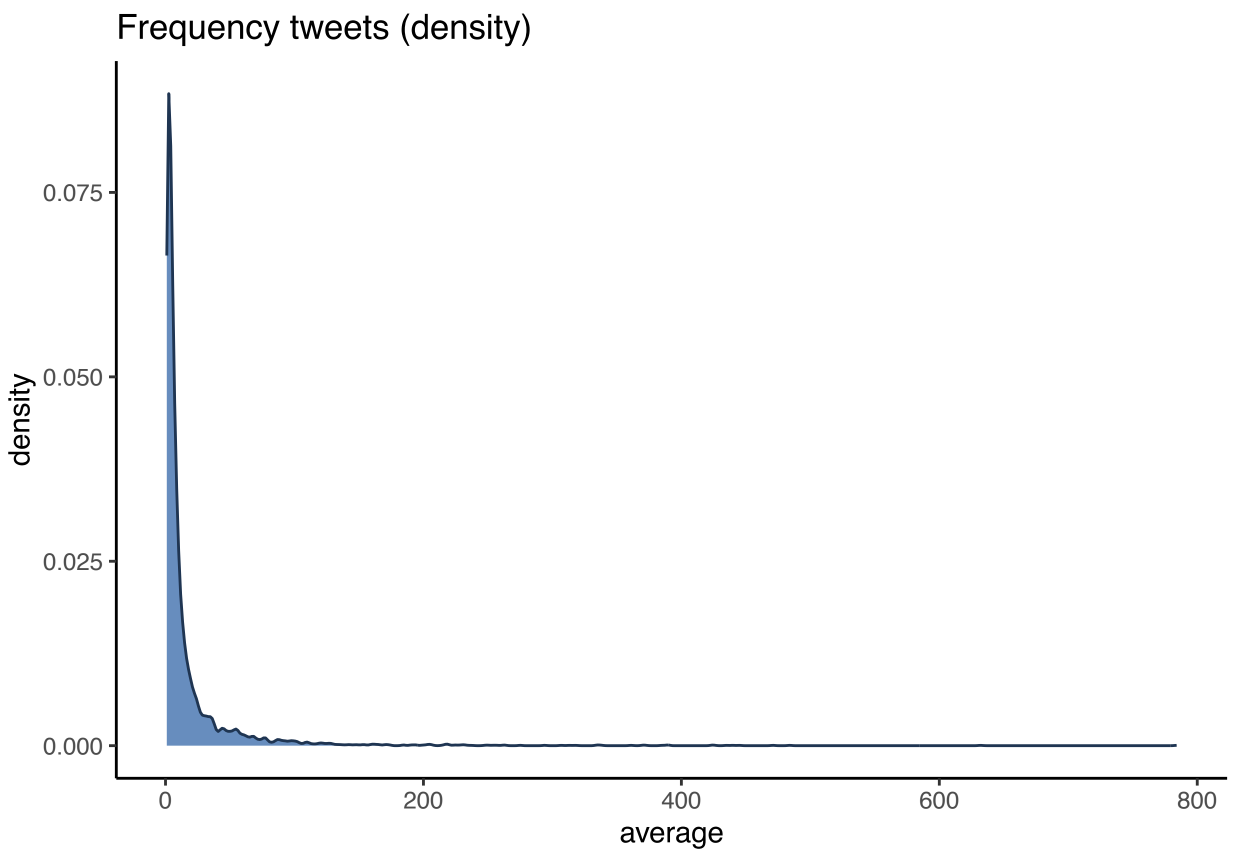
In this case we can see that the argument Telescopia is over retweeted. The reason can be the common word of the ICO, to the word: telescope. Hence, excluding that ICO, we can focus on the others.

To conclude, in this case showing the last 10 ICOs who are mentioned in a tweet and then is calculated the average retweet count is unnecessary because all those arguments are equel to 0.

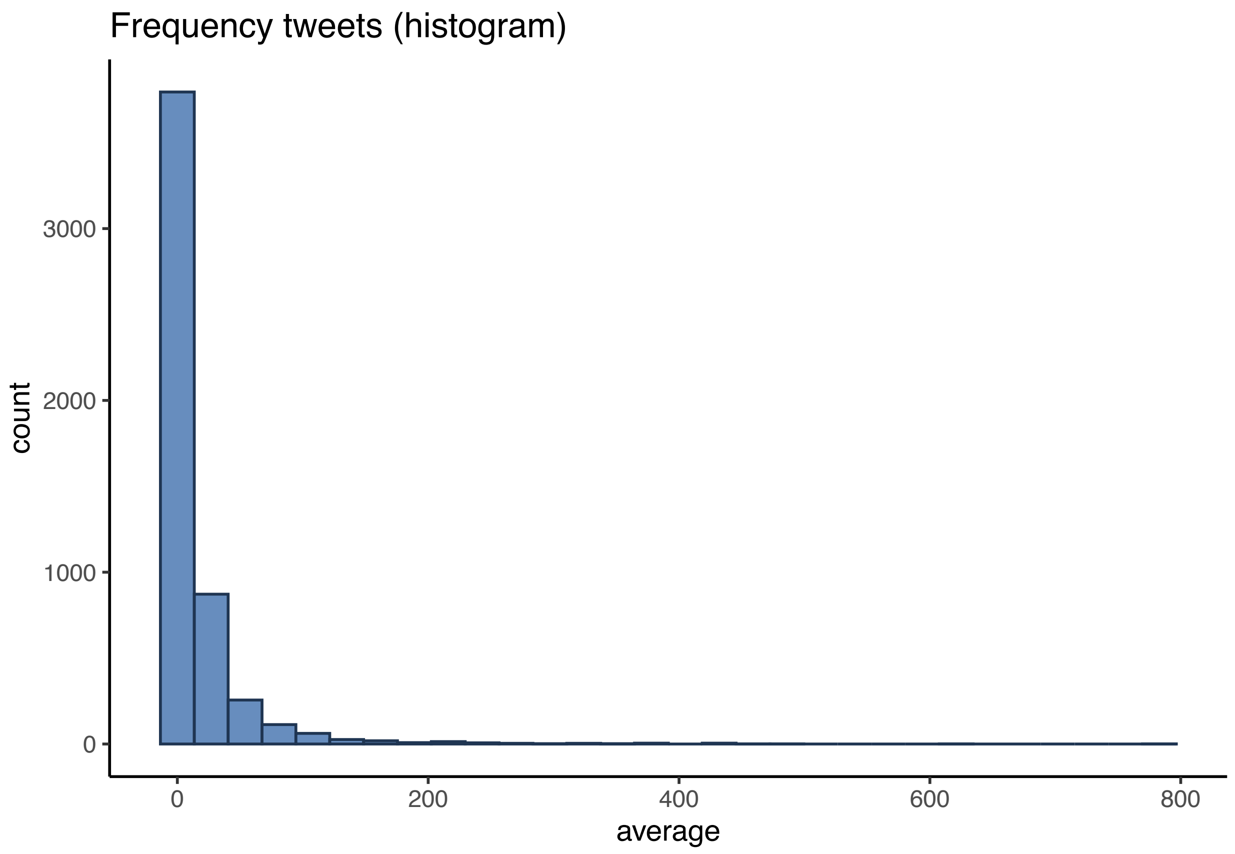
### 8.3.6 Frequency tweet per ICO

In this case might be interesting understand how the previous processes of data gathering influenced the actual dataset. To better explaining, I want to understand for each ICO, how many tweets I could found on Twitter (starting with a maximum limit per ICO equal to 1000).

```
t = twitter_db %>%
  dplyr::select(ico_name, retweet_count) %>% filter(retweet_count > 0) %>%
  group_by(ico_name) %>% summarize(average=mean(retweet_count))

t$average <- round(t$average)


ggplot(t, aes(x=average)) +
  geom_density(fill="#4271AE", color="#1F3552", alpha=0.8) +
  ggtitle("Frequency tweets (density)") +
  theme_classic()
```



```
ggplot(t, aes(x=average)) +
  geom_histogram(bins = 30, fill="#4271AE", color="#1F3552", alpha=0.8) +
  ggtitle("Frequency tweets (histogram)") +
  theme_classic()
```

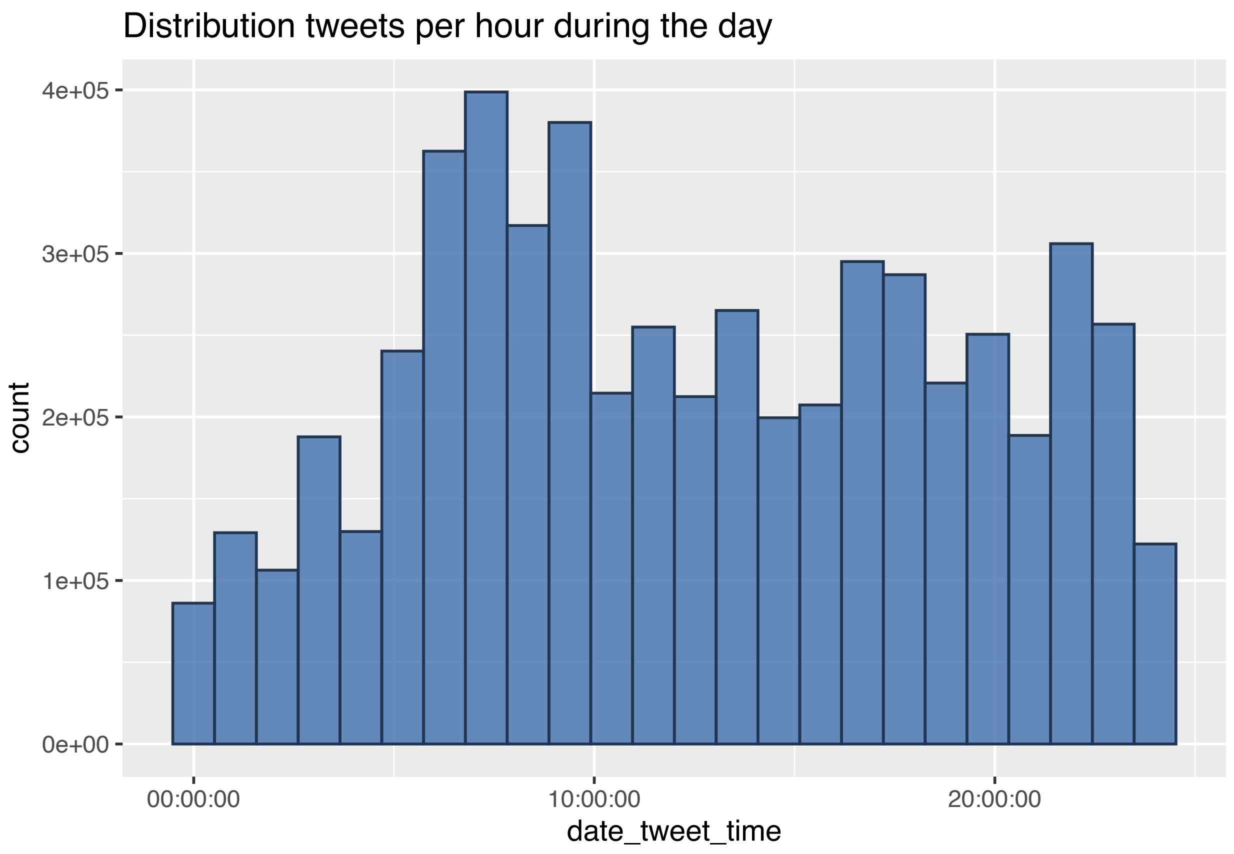


```
rm(t)
```

### 8.3.7 Distribution tweets per hour

In this case, the time where a post is posted on twitter can be relevant. Strangely, even if the main hours are generlally known in the lunch break and after working hourd, in this graph we can notice that the main tweets are tweeted in the morning. I can suggest, that referring ICOs and companies they can have a social pages and they post in those hours most.

```
test <- twitter_db
test %>%
  ggplot(aes(x = date_tweet_time)) +
  geom_histogram(bins = 24, fill="#4271AE", color="#1F3552", alpha=0.8) +
  scale_x_time() + ggtitle("Distribution tweets per hour during the day")
```

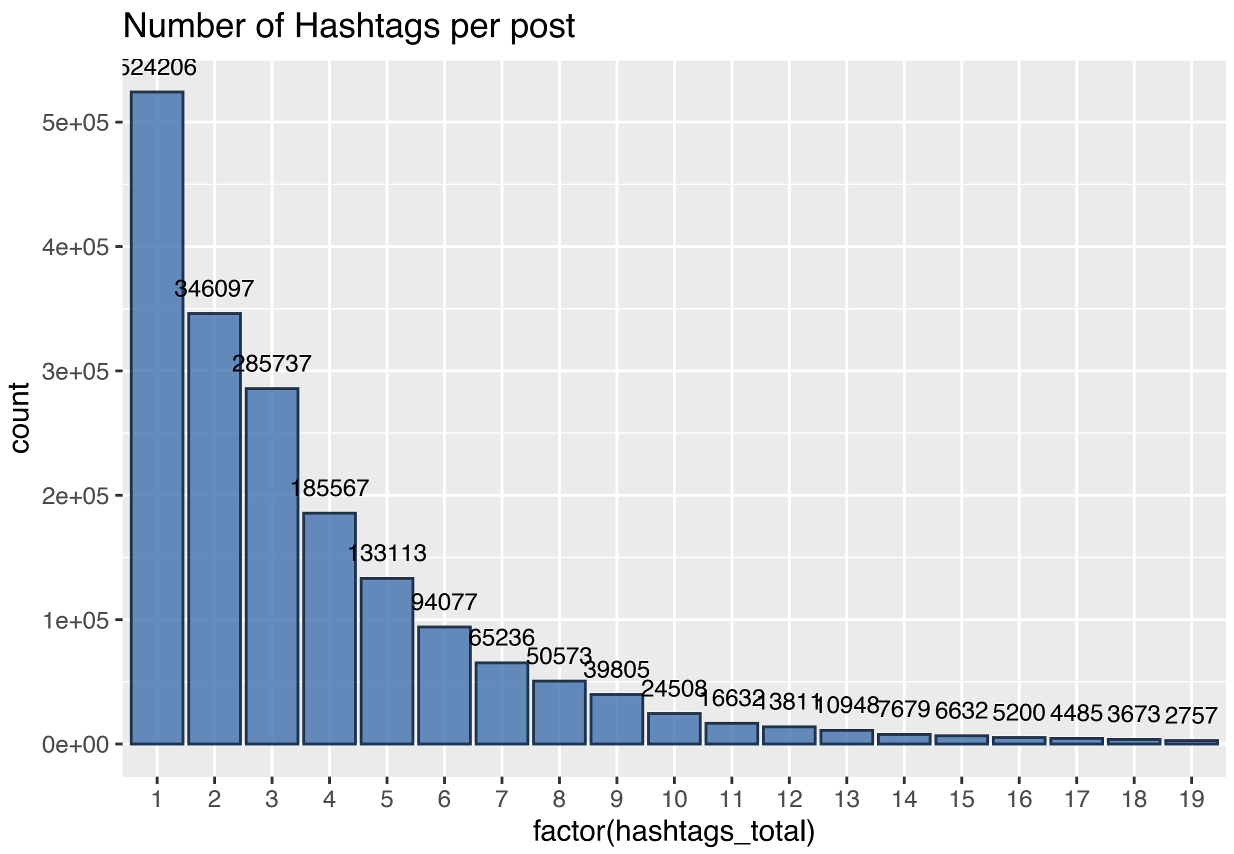


```
rm(test)
```

### 8.3.8 Hashtags

The usage of hashtags is also relevant, and in this graph we can notice that the majority of tweets (excluding those with 0 hashtagas) have from 1 to 5 hashtags.

```
t <- twitter_db %>% dplyr::select(ico_name, hashtags_total) %>%
  filter(hashtags_total>0) %>%
  filter(hashtags_total<20)

ggplot(t, aes(x=factor(hashtags_total))) +
  geom_bar(fill="#4271AE", color="#1F3552", alpha=0.8) +
  geom_text(aes(label = after_stat(count)),
            stat = "count", vjust = -1, colour = "black", size=3) +
  ggtitle("Number of Hashtags per post")
```

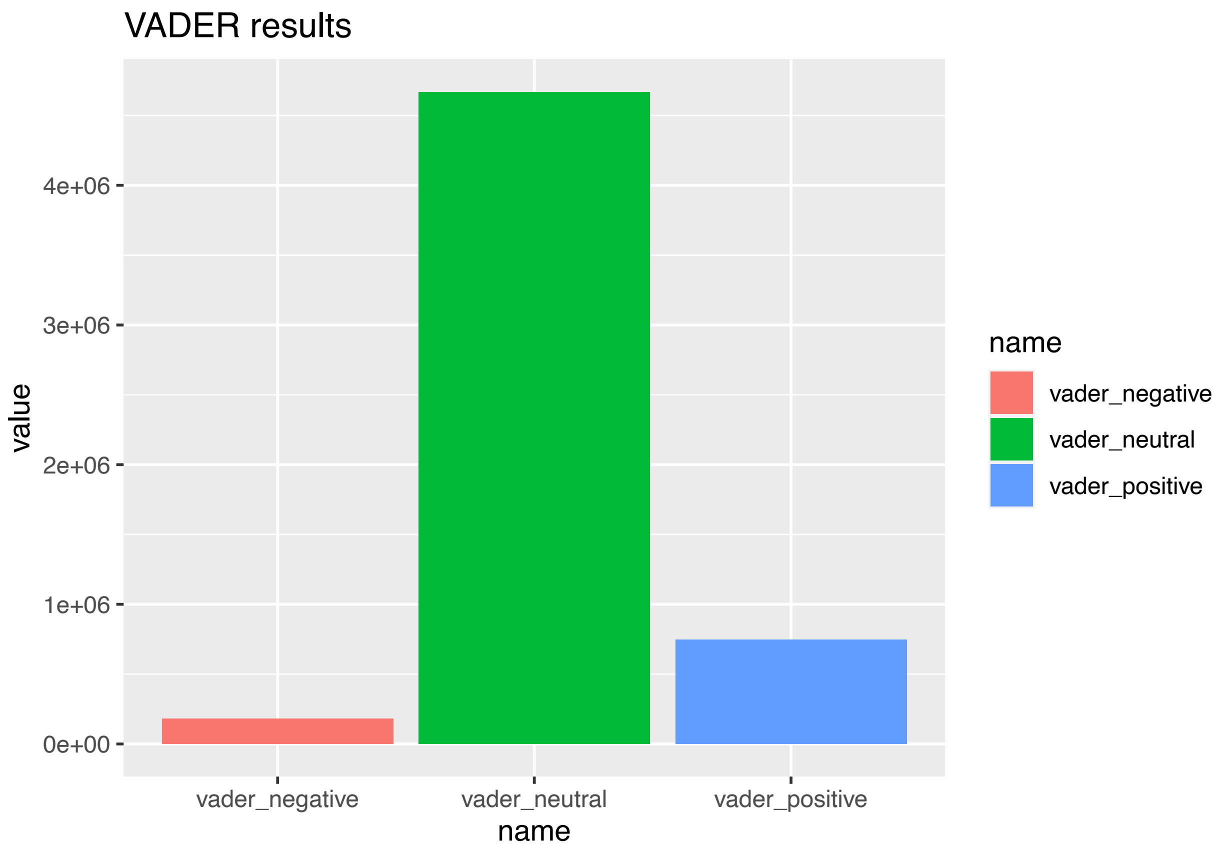


```
rm(t)
```

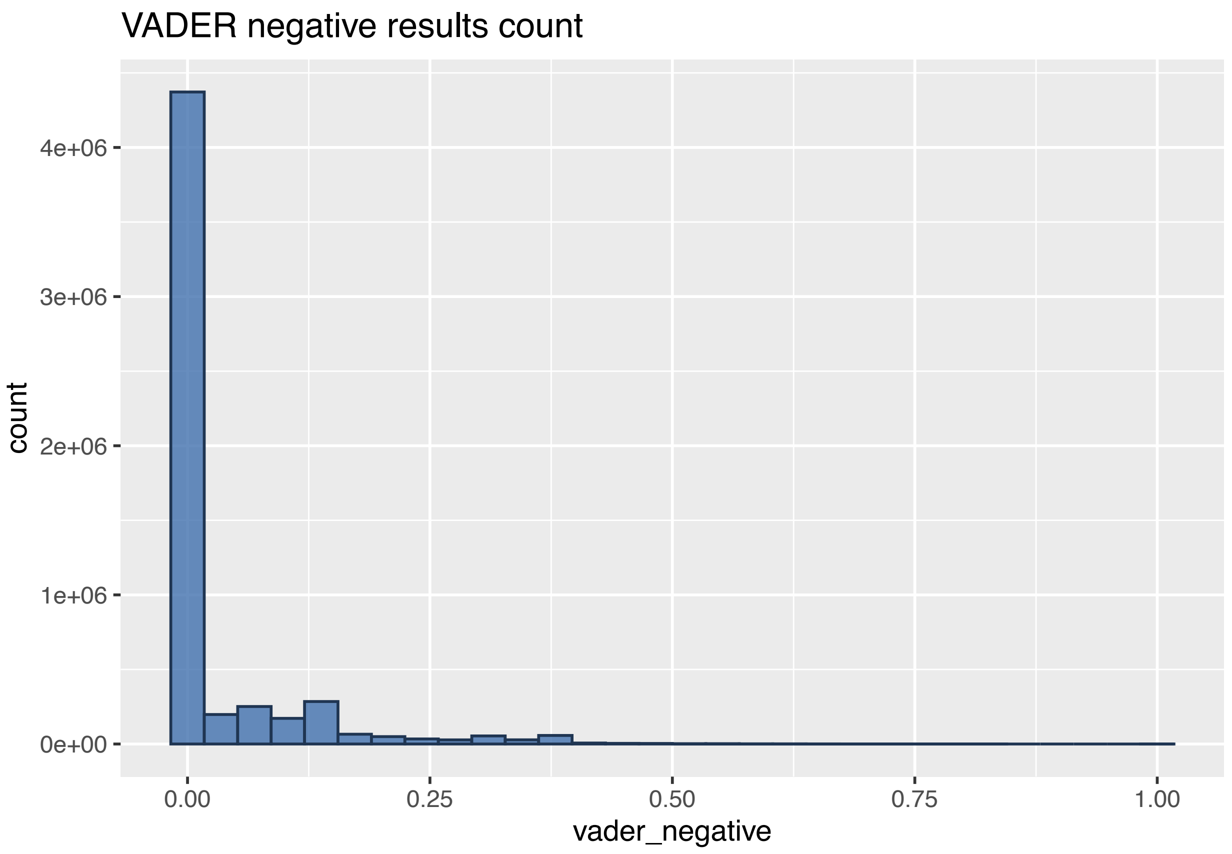
### 8.3.9 VADER

The vader results are very easy to explain. The Neutral values are the majority, but there is also an high percetage of positive tweets.

```
t <- twitter_db %>% dplyr::select(vader_negative, vader_neutral, vader_positive)
t %>%
  summarise(across(everything(), sum)) %>%
  pivot_longer(cols = everything()) %>%
  ggplot(aes(x = name, y = value, fill = name)) +
  geom_col() + ggtitle("VADER results")
```



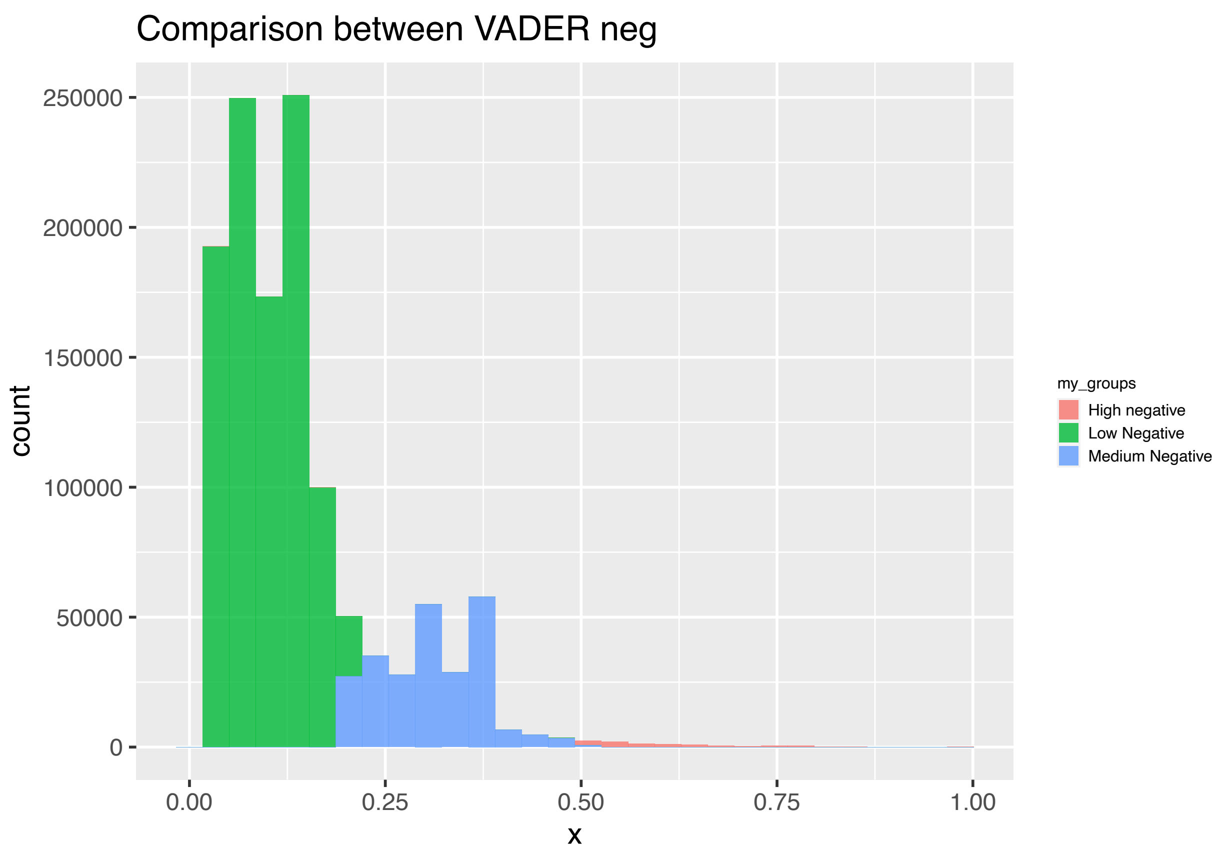
```
ggplot(t, aes(vader_negative)) +
  geom_histogram(fill="#4271AE", color="#1F3552", alpha=0.8, bins = 30) +
  ggtitle("VADER negative results count")
```



Now, I will create graphs that shows the Negative and Positive tweets results from the VADER sentiment analysis The following one is referring to the negative results, and shows that the majority have a 0 value, due to the majority results overall is dominated by neutral tweets.

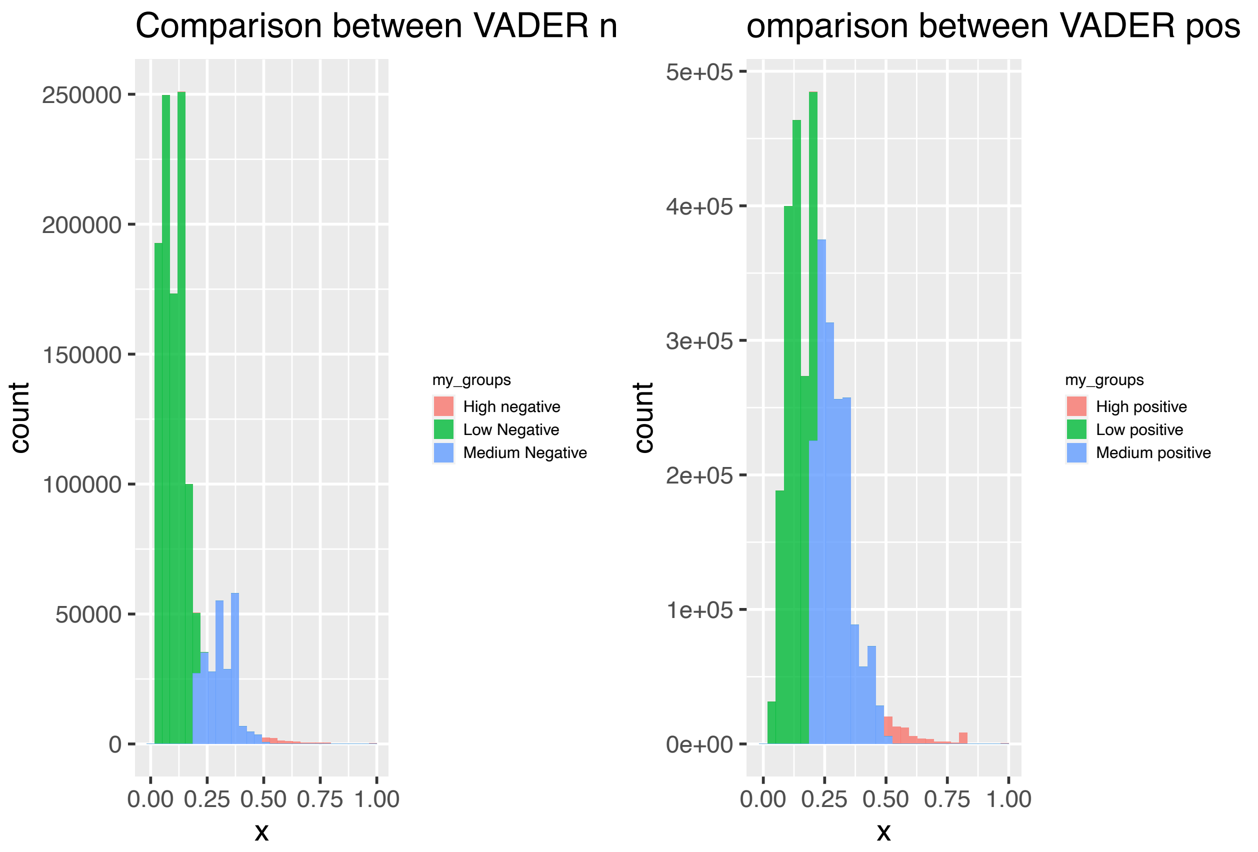
```
#Filtering for all negative that are at least higher than 0
t <- twitter_db %>%
  dplyr::select(vader_negative, vader_neutral, vader_positive) %>%
  filter(vader_negative > 0)
my_groups <- rep("Low Negative", nrow(t))

# Specify groups for different colors
my_groups[t$vader_negative > 0.2 & t$vader_negative <= 0.5] <- "Medium Negative"
my_groups[t$vader_negative > 0.5] <- "High negative"


# Create data frame with values & groups
data <- data.frame(x = t$vader_negative,
                   my_groups = my_groups)

neg_graph <- ggplot(data, aes(x, fill = my_groups)) +
  geom_histogram(alpha=0.8, bins = 30) +
  guides(shape = guide_legend(override.aes = list(size = 0.5)),
         color = guide_legend(override.aes = list(size = 0.5))) +
  theme(legend.title = element_text(size = 6),
        legend.text  = element_text(size = 6),
        legend.key.size = unit(0.6, "lines")) +
  ggtitle("Comparison between VADER neg")
```

```
neg_graph
```



While, this graph is referring to the positive results. To conclude, I used gridExtra to plot both negative and positive graph together. In the positive graph, we can notice also that the medium positive tweets are the majority compared to the high positive tweets.

```
#Filtering for all negative that are at least higher than 0
t <- twitter_db %>%
  dplyr::select(vader_negative, vader_neutral, vader_positive) %>%
  filter(vader_positive > 0)
my_groups <- rep("Low positive", nrow(t))

# Specify groups for different colors
my_groups[t$vader_positive > 0.2 & t$vader_positive <= 0.5] <- "Medium positive"
my_groups[t$vader_positive > 0.5] <- "High positive"


 # Create data frame with values & groups
data <- data.frame(x = t$vader_positive,
                   my_groups = my_groups)

pos_graph <- ggplot(data, aes(x, fill = my_groups)) +
  geom_histogram(alpha=0.8, bins = 30) +
  guides(shape = guide_legend(override.aes = list(size = 0.5)),
         color = guide_legend(override.aes = list(size = 0.5))) +
  theme(legend.title = element_text(size = 6),
        legend.text  = element_text(size = 6),
        legend.key.size = unit(0.6, "lines")) +
  ggtitle("omparison between VADER pos")



# Create a grid extra to show both graph pos and neg
grid.arrange(neg_graph, pos_graph, ncol=2)
```



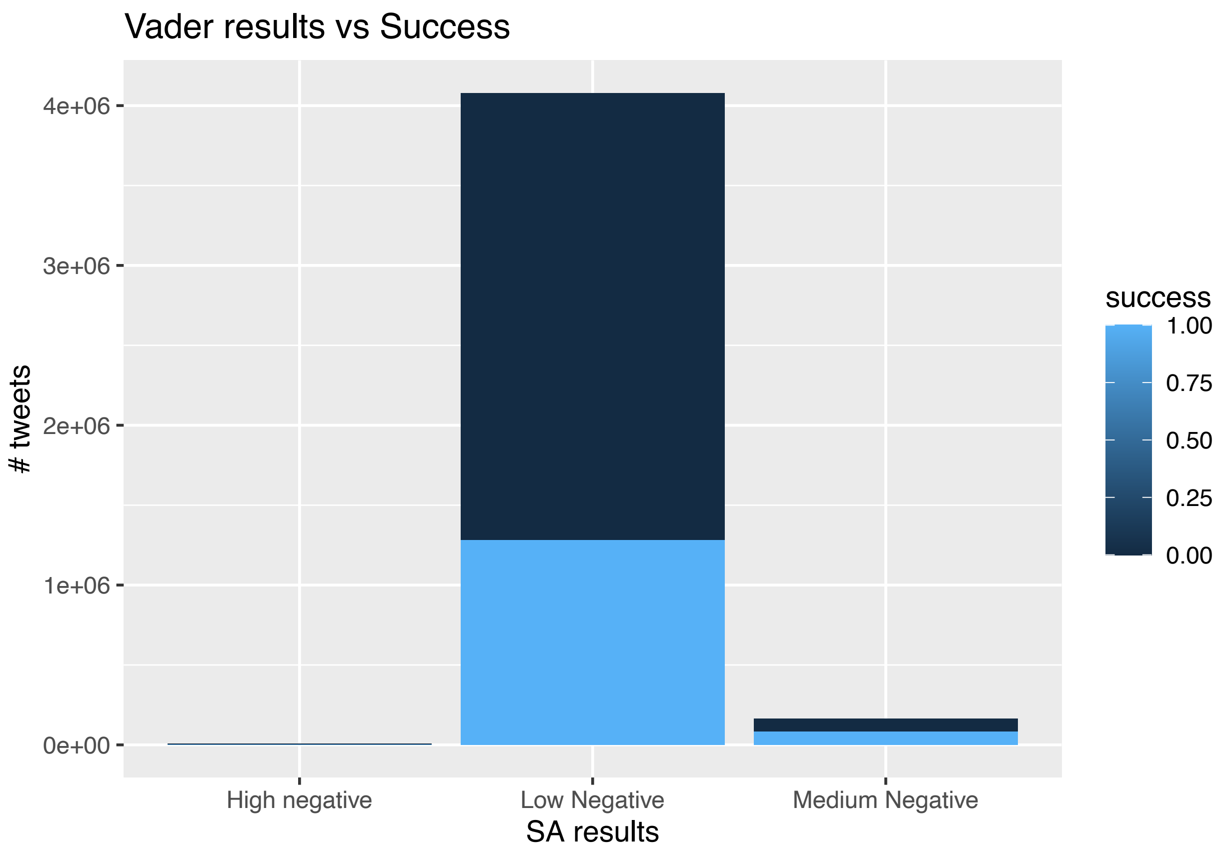
```
rm(my_groups, t, pos_graph, neg_graph, data)
```

**8.3.9.1 Vader negative results success** Analysing vader negative results alone, is possible to see that the success of low negative tweets is higher, but this can be due to the high number of low-negative results.

```
t <- twitter_db %>% dplyr::select(vader_negative, success)

my_groups <- rep("Low Negative", nrow(t))

# Specify groups for different colors
my_groups[t$vader_negative > 0.2 & t$vader_negative <= 0.5] <- "Medium Negative"
my_groups[t$vader_negative > 0.5] <- "High negative"


 # Create data frame with values & groups
data <- data.frame(x = t$vader_negative,
                   my_groups = my_groups, success = t$success)

t <- data %>%
    group_by(my_groups, success) %>%
    summarise(count = n())
```

```
## `summarise()` has grouped output by 'my_groups'. You can override using the
## `.groups` argument.
```

```
ggplot(t, aes(fill=success, y=count, x=my_groups)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# tweets") + xlab("SA results") +
  ggtitle("Vader results vs Success")
```



```
rm(t, data, my_groups)
```

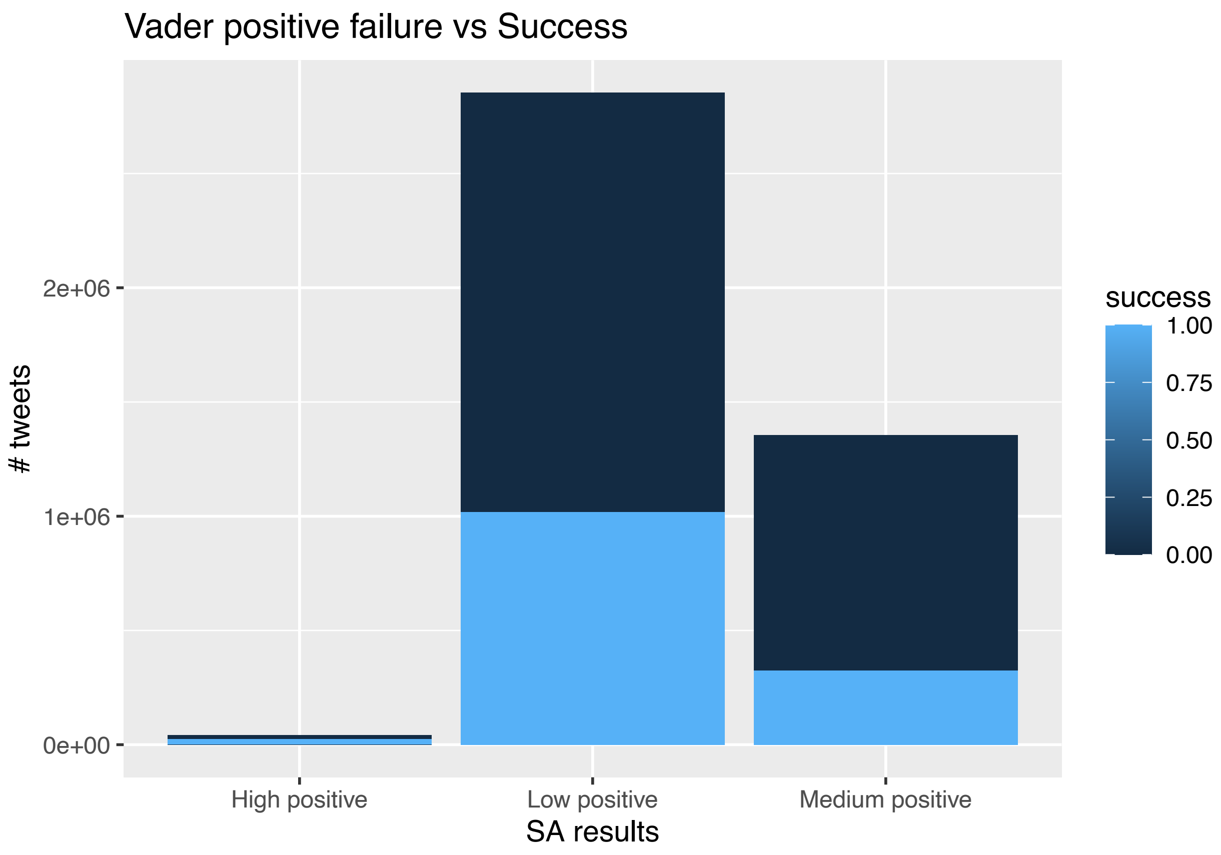
**8.3.9.2 Vader positive results success** Also in this case we can notice that the low positive results are more incline to succeed. Anyway, the high positive tweets are almost all succeeded.

```
t <- twitter_db %>% dplyr::select(vader_positive, success)

my_groups <- rep("Low positive", nrow(t))

# Specify groups for different colors
my_groups[t$vader_positive > 0.2 & t$vader_positive <= 0.5] <- "Medium positive"
my_groups[t$vader_positive > 0.5] <- "High positive"


# Create data frame with values & groups
data <- data.frame(x = t$vader_positive,
                   my_groups = my_groups, success = t$success)
```

```
t <- data %>%
    group_by(my_groups, success) %>%
    summarise(count = n())
```

```
## `summarise()` has grouped output by 'my_groups'. You can override using the
## `.groups` argument.
```

```
ggplot(t, aes(fill=success, y=count, x=my_groups)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# tweets") + xlab("SA results") +
  ggtitle("Vader positive failure vs Success")
```



```
# printing the rations
#high positive
t$count[2] / (t$count[1] + t$count[2])
```

```
## [1] 0.3583917
```

```
# low positive
t$count[4] / (t$count[3] + t$count[4])
```

```
## [1] 0.2631576
```

```
# medium positive
t$count[6] / (t$count[5] + t$count[6])
```

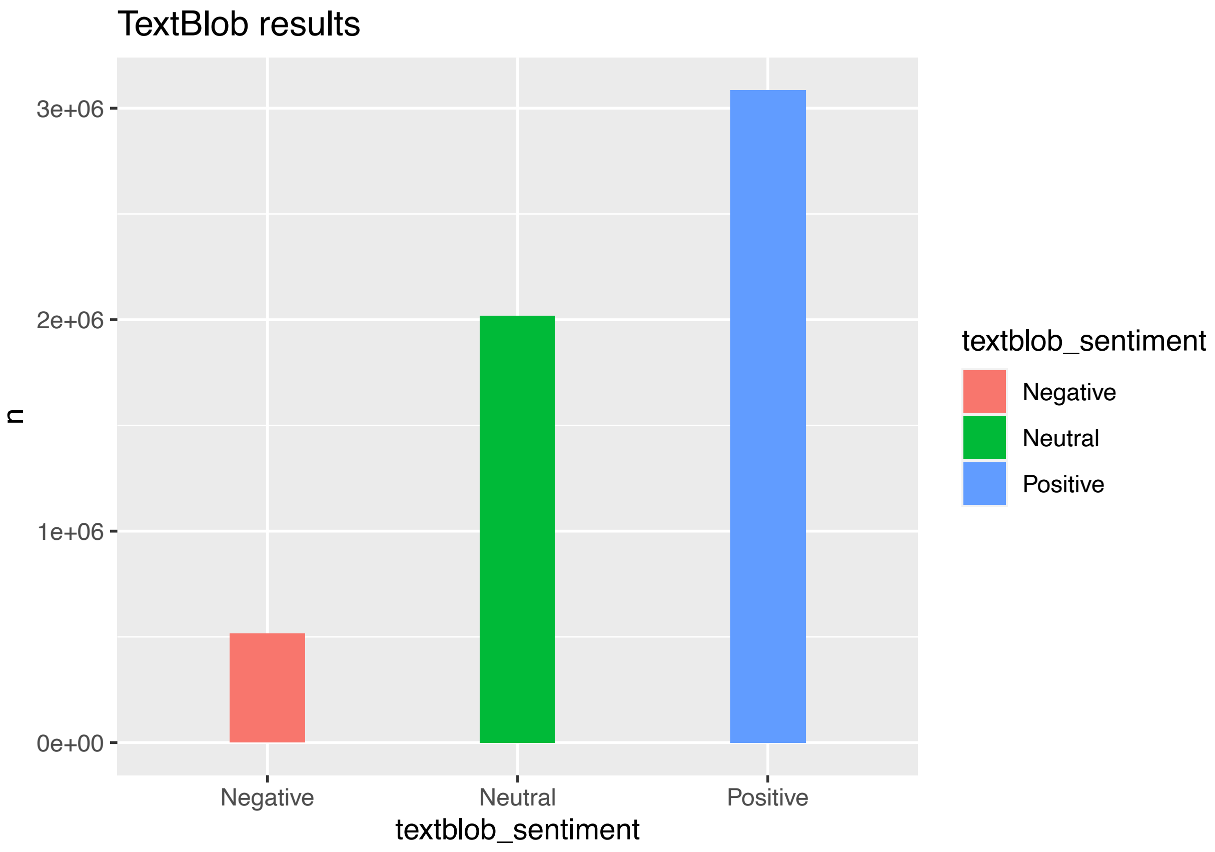
```
## [1] 0.1928678
```

```
#rm(t, data, my_groups)
```

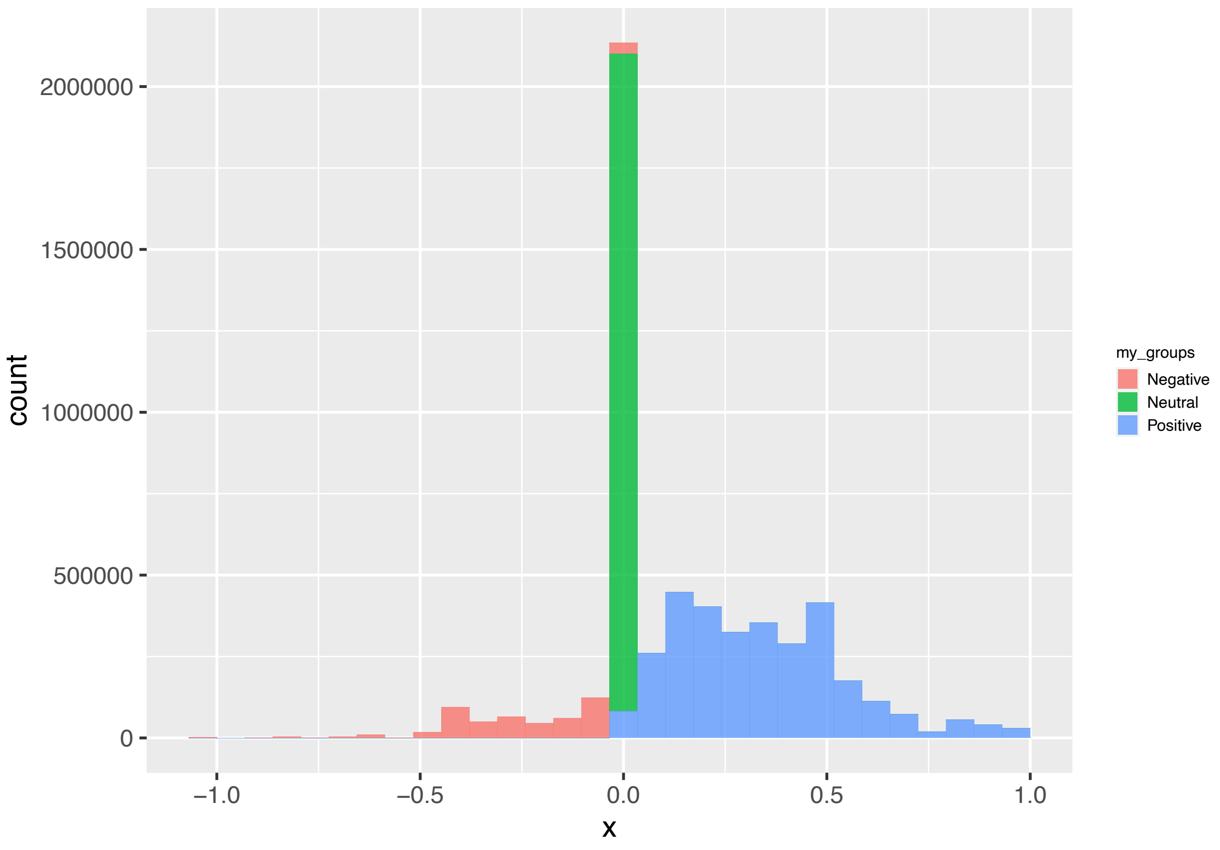
### 8.3.10 TextBlob

Using TextBlob the positive tweets are the vast majority. So, compared to VADER we have a more positive evaluation of tweets.

```r
t <- twitter_db %>% tabyl(textblob_sentiment)
ggplot(t, aes(x = textblob_sentiment, y= n)) +
  geom_col(aes(fill = textblob_sentiment), width = 0.3, show.legend = TRUE) +
  ggtitle("TextBlob results")
```



```r
t <- twitter_db %>% dplyr::select(textblob_polarity)
my_groups <- rep("Negative", nrow(t))

# Specify groups for different colors
my_groups[t$textblob_polarity == 0] <- "Neutral"
my_groups[t$textblob_polarity > 0] <- "Positive"


# Create data frame with values & groups
data <- data.frame(x = t$textblob_polarity,
                   my_groups = my_groups)

ggplot(data, aes(x, fill = my_groups)) +
  geom_histogram(alpha=0.8, bins = 30) +
  guides(shape = guide_legend(override.aes = list(size = 0.5)),
         color = guide_legend(override.aes = list(size = 0.5))) +
  theme(legend.title = element_text(size = 6),
        legend.text  = element_text(size = 6),
        legend.key.size = unit(0.6, "lines"))
```

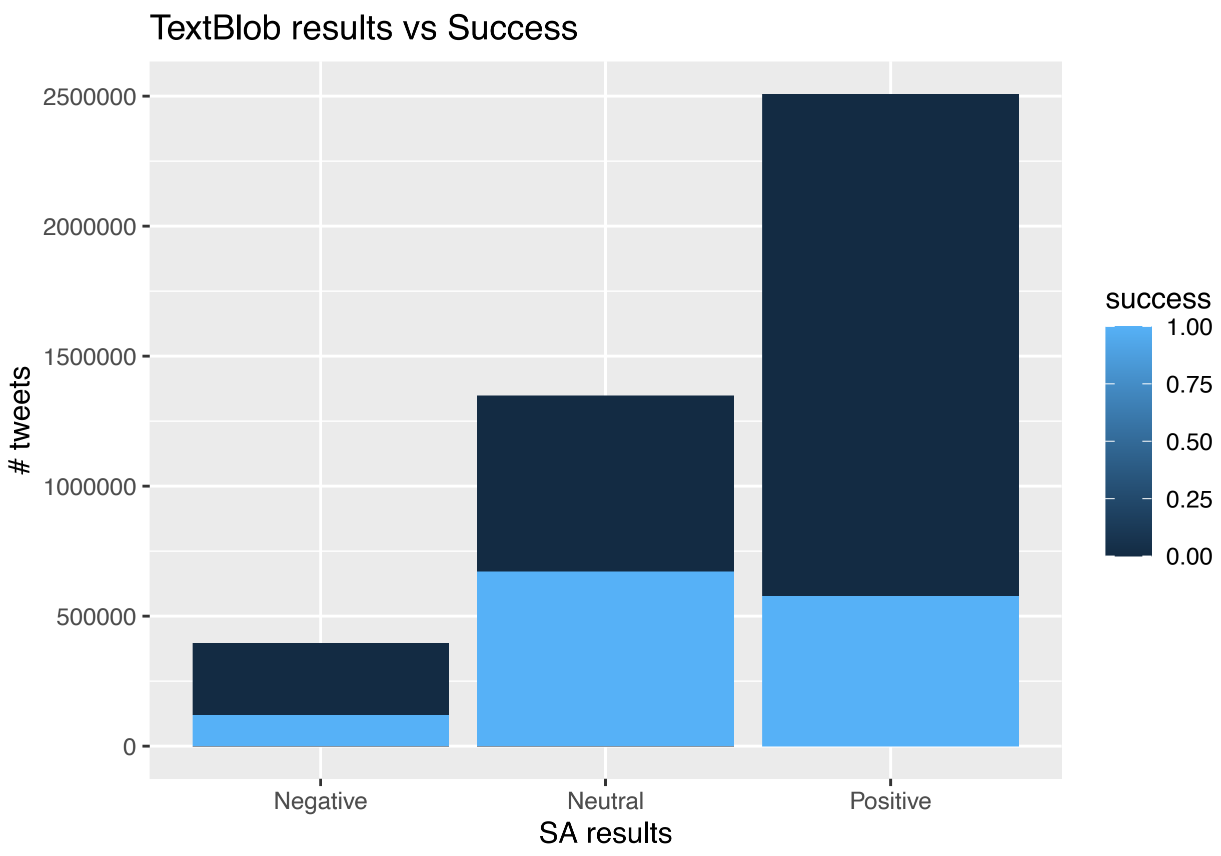


**8.3.10.1 TextBlob results success** This graph is really explicative, and from this plot we can understand that maybe the tweets is not a success factor for the ICOs, at least not all. In fact, we can notice that neutral resulst have better success raio comapre to the positive ones. The reson could be clearly caused by the automatoin process, while in a perfect dataset, we sould have only tweets strictly related to the ICOs projects and posted by popular accounts.

```
t <- twitter_db %>% dplyr::select(textblob_polarity, success)
my_groups <- rep("Negative", nrow(t))  # Specify groups for different colors
my_groups[t$textblob_polarity == 0] <- "Neutral"
my_groups[t$textblob_polarity > 0] <- "Positive"


# Create data frame with values & groups
data <- data.frame(x = t$textblob_polarity,
                       my_groups = my_groups, success = t$success)

t <- data %>%
    group_by(my_groups, success) %>%
    summarise(count = n())
```

```
## `summarise()` has grouped output by 'my_groups'. You can override using the
## `.groups` argument.
```

```
ggplot(t, aes(fill=success, y=count, x=my_groups)) +
    geom_bar(position="dodge", stat="identity") +
  ylab("# tweets") + xlab("SA results") +
  ggtitle("TextBlob results vs Success")
```



```
rm(t, data, my_groups)
```

# 9 Predictive Analysis

Now, is time to do do predictive analysis. So, I decided to proceed with the *twitter_db* alone using the package *h2o*. As we can notice, *titter_db* is a very large dataset and thanks to this package the results where faster and more accurate compared to the base models.

Secondly, I collapsed the *twitter_db*, computing the mean of the main variables. Thus, I joined the collapsed dataset into the *tord_db* finally obtaining a final dataframe.

To sum up, in this section I will display different models and analyse how they will be able to forecast the correct success of the ICOs under consideration.

## 9.1 Twitter DB

After the long process of data cleaning and variable generation, I was finally able to obtain a cleaned dataframe, that was converted in an *h2o* dataframe to compute different models.

In this section, after an initial *h2o* setting setting phase I tried different models: - Generalized Linear Models (GLM) - Distributed Random Forest (DRF) - Gradient Boosting Machine (GBM) - Naïve Bayes Classifier - Deep Learning (Neural Networks) - Machine Learning (ML)

### 9.1.1 h2o settings

First of all, I needed to set the *h2o* package, and copy my *twitter_db* as an *h2o* variable.

To do so, the *id* column must be removed, and also those columns still containing NAs values. Once, that part was done, h2o had to be initialized. Then, the *twitter_db* had to be converted as an *h2o* element.

In this sections is possibile to notice also how I decided to split the h2o dataframe in the test and training subsets.

To conclude, the $x$ and the $y$ variables corresponds to an array of variables that represent respectively the response variable and the predictors.

```r
# remove id column
twitter_db_h2o <- twitter_db[, -1]

# remove other columns that generates conversion issues
twitter_db_h2o <- subset(twitter_db_h2o, select = -c(date_tweet_ymd, date_tweet_time))

# begin h2o instance
h2o.init()
h2o.init(nthreads = -1)


# convert dataset in h2oFrame
twitter_db_h2o <- as.h2o(twitter_db_h2o)
twitter_db_h2o

# take a look to the content of the h2oFrame
h2o.describe(twitter_db_h2o)

# Conver as a Factor success variable
twitter_db_h2o$success <- h2o.asfactor(twitter_db_h2o$success)

# Construct test and train sets using sampling
```

```r
twitter_db_h2o_split = h2o.splitFrame(data = twitter_db_h2o,
   ratios = 0.85)
twitter_db_h2o_train = twitter_db_h2o_split[[1]]
twitter_db_h2o_train

twitter_db_h2o_test = twitter_db_h2o_split[[2]]
twitter_db_h2o_test

# # Display a summary using table-like functions
h2o.table(twitter_db_h2o_train$success)
h2o.table(twitter_db_h2o_test$success)


# Set predictor and response variables
y = "success"
x = c("vader_negative", "vader_neutral", "vader_positive", "vader_compound",
      "ico_name", "token_name", "verified", "follows_count", "friends_count",
      "retweet_count", "language", "number_of_likes", "tweet_words",
      "hashtags_total", "cleaned_tweet_words", "textblob_polarity",
      "textblob_subjectivity", "textblob_sentiment")
```

**9.1.1.1 Converting in h2o dataframe** I finally was ready to apply this arrays in my models and try to find out the efficiency of the model, but also of my dataframe.

### 9.1.2 Generalized Linear Models (GLM)

Regression models are estimated using generalized linear models (GLM) for outcomes with exponential distributions. These include the Poisson, binomial, and gamma distributions in addition to the Gaussian (or normal) distribution. Each has a particular function and can be used for classification or prediction based on the distribution and connection function used.

```r

h2o.init(nthreads = -1)
# Define the data for the model and display the esults
# twitter_model_h20_glm <- h2o.glm(training_frame=twitter_db_h2o_train,
#     x=x, y=y, family = "binomial")

twitter_model_h20_glm <- h2o.upload_model("/Users/ettorefalde/DataspellProjects/ICO_Forecasting/R/GLM_mo

# View model information: training statistics, performance, important variables
summary(twitter_model_h20_glm)


# Predict using GLM model
pred = h2o.predict(object = twitter_model_h20_glm, newdata =
   twitter_db_h2o_test)


# Convert results in a dataframe to check

# predictions <- as.data.frame(pred)
# predictions
# final_db_h2o_test_df <- as.data.frame(final_db_h2o_test)
# results_prediction <- cbind(final_db_h2o_test_df, forecast = predictions$predict)
# results_prediction <- results_prediction %>% relocate(forecast, .after = success)



# Look at summary of predictions: probability of TRUE class (p1)
summary(pred)
```

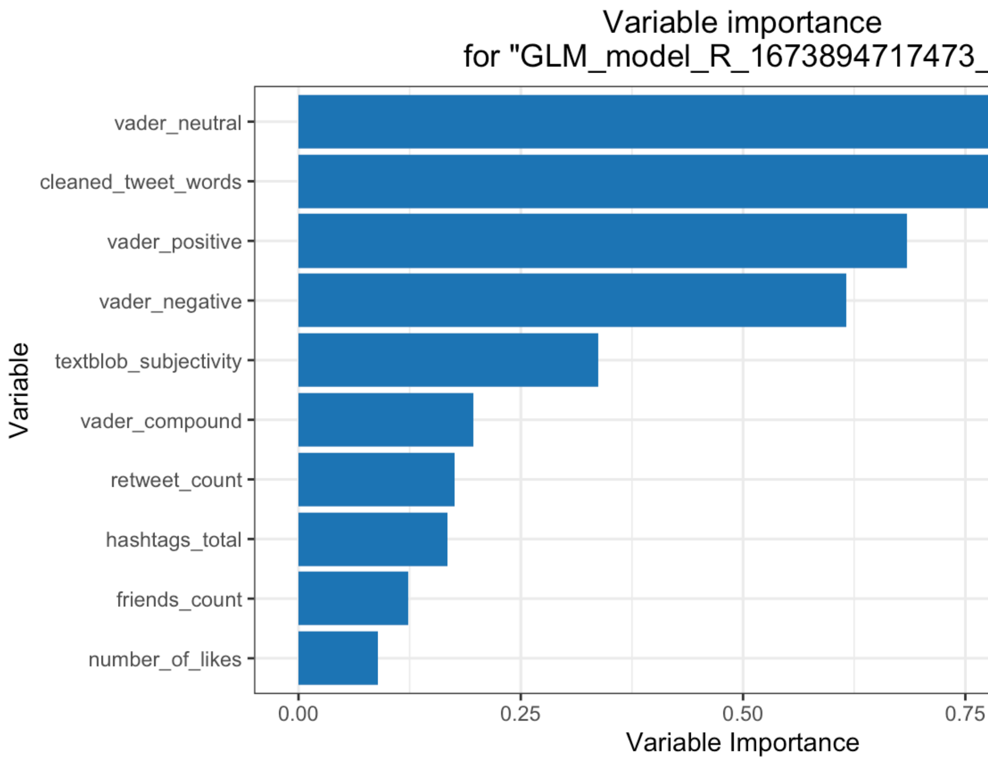
**9.1.2.1 Model 1** Logistic Regression (Binomial Family) was used in this model, because this model is used for binary classification problems where the response is a categorical variable with two levels. It models the probability of an observation belonging to an output category given the data.

Hence, the results are pretty clear, and show: - max f1: threshold = 0.237056; value = 0.459113
- max accuracy: threshold = 0.462706; value = 0.761491
- max precision: threshold = 0.462706; value = 0.637676 - max senitivity (recall): threshold = 0.005629; value = 1.000000
- max specificity: threshold = 0.996049; value = 0.999988

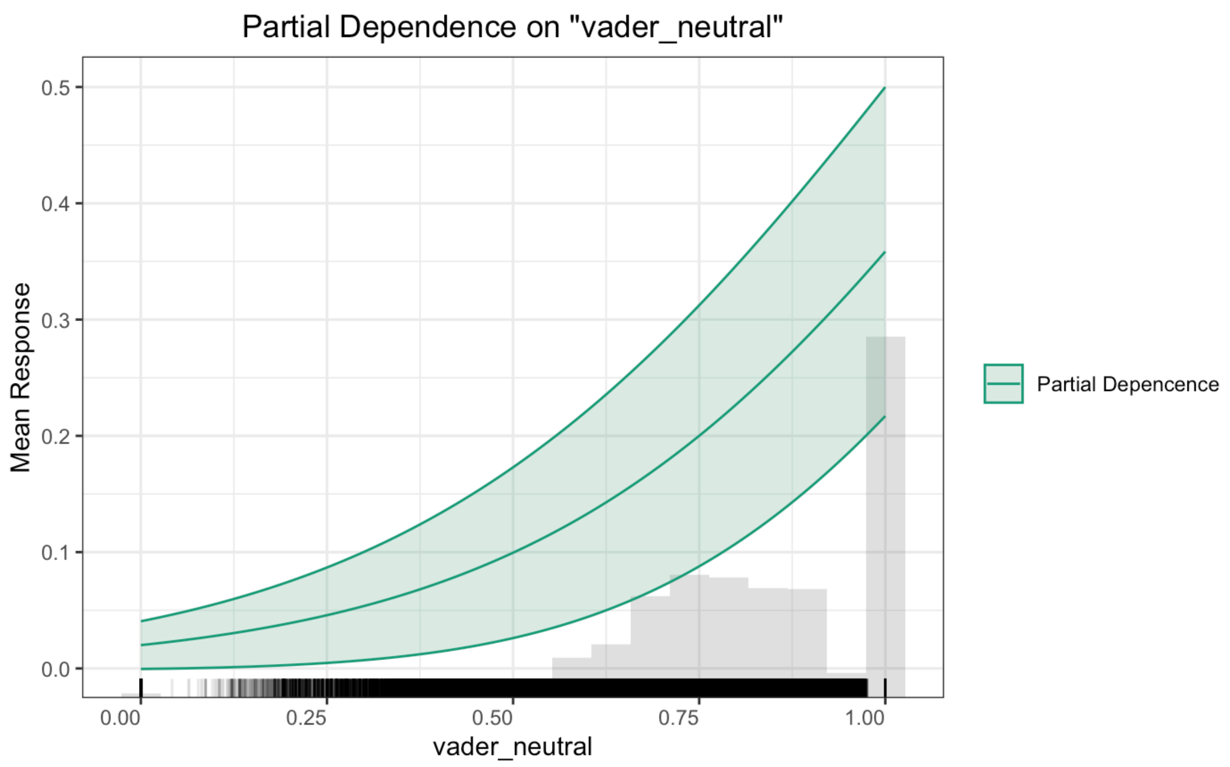
This, can be interpreted that the model is unbalanced or there si teh need to modify the model. Therefore, I will proceed with the second solution and modify the model, also taking in consideration the following plots.

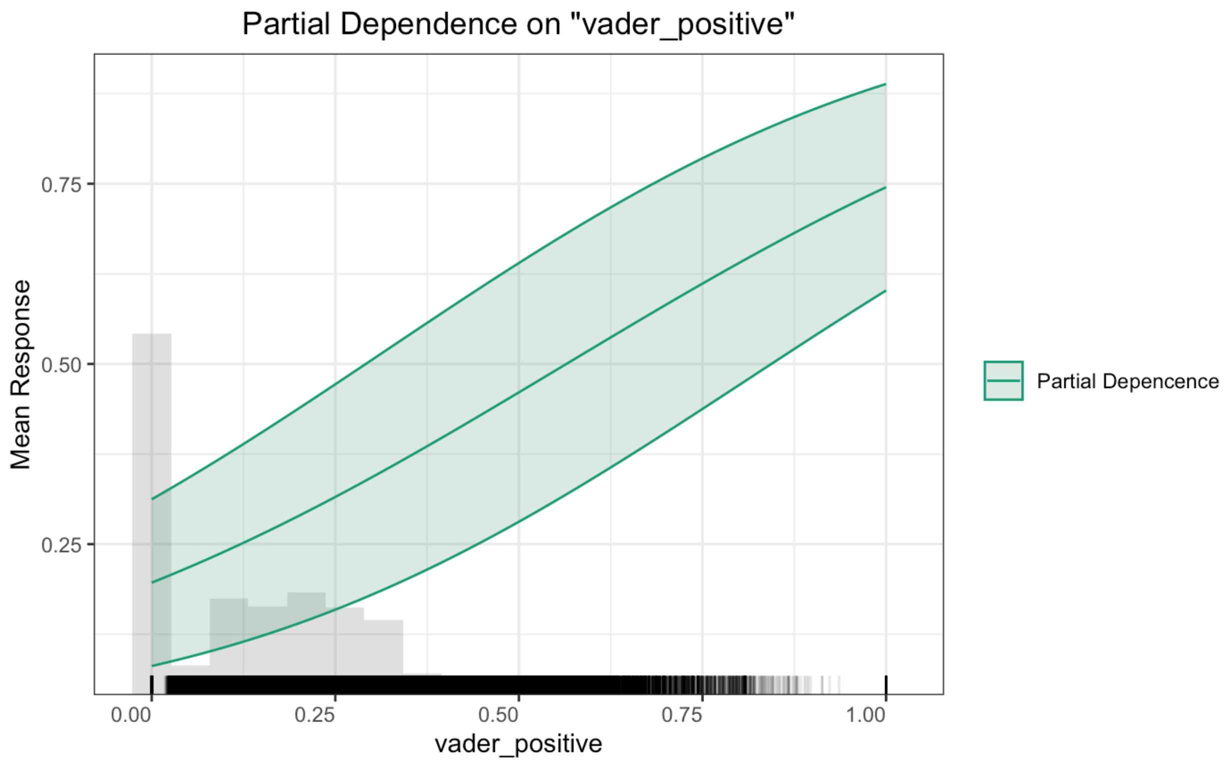
```
#h2o.explain(twitter_model_h20_glm, twitter_db_h2o_test)
```

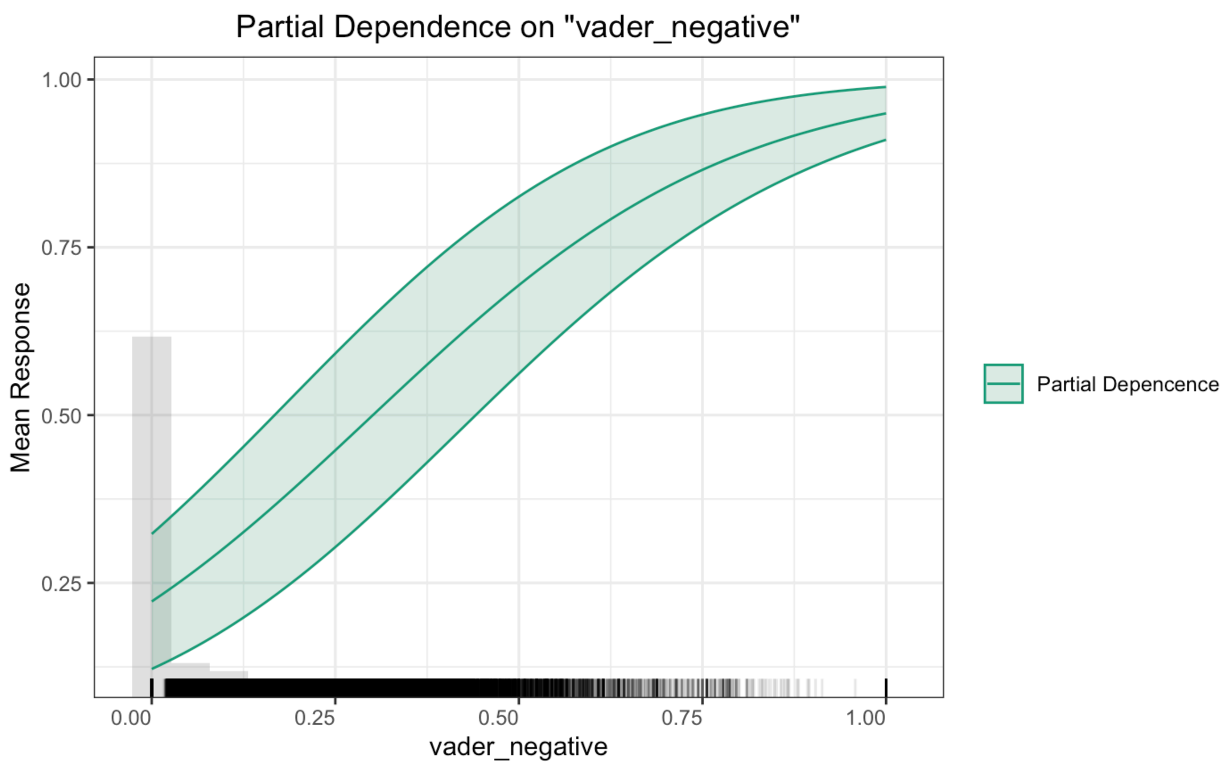
**9.1.2.2 Explaining GLM** In the following plots is possible to notice the main relevant varialves and their

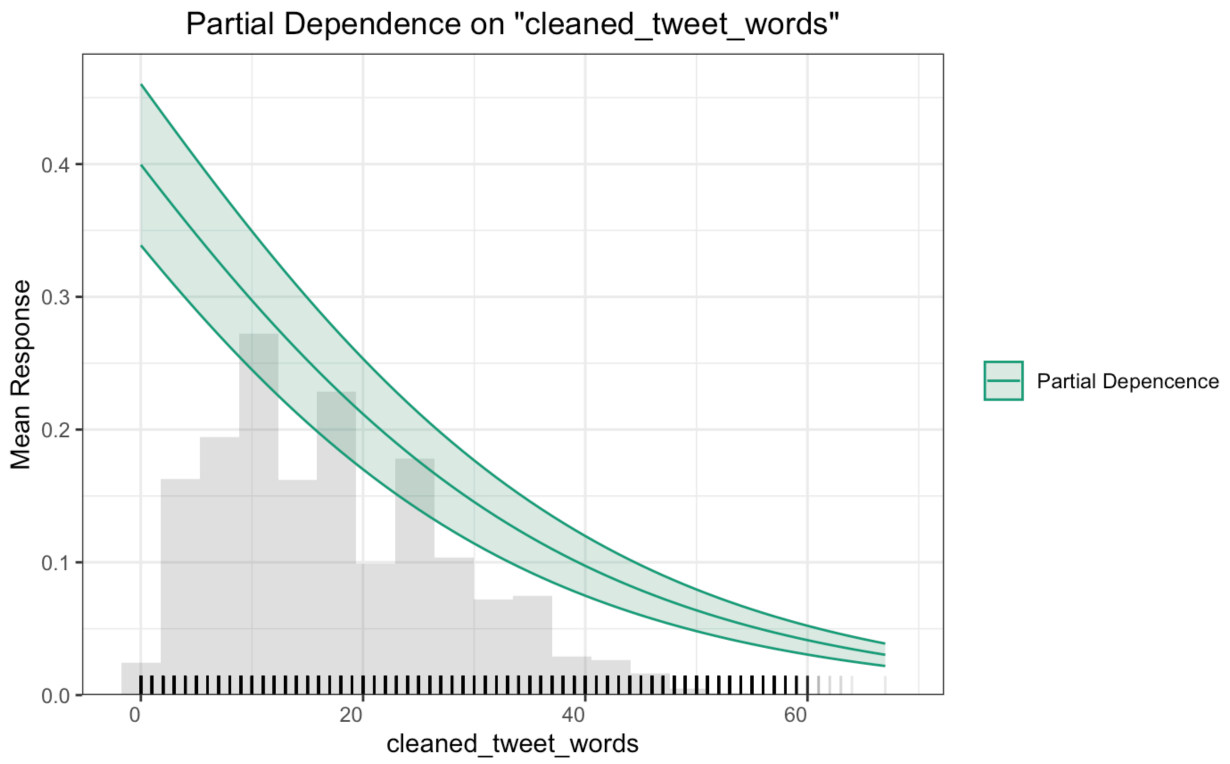


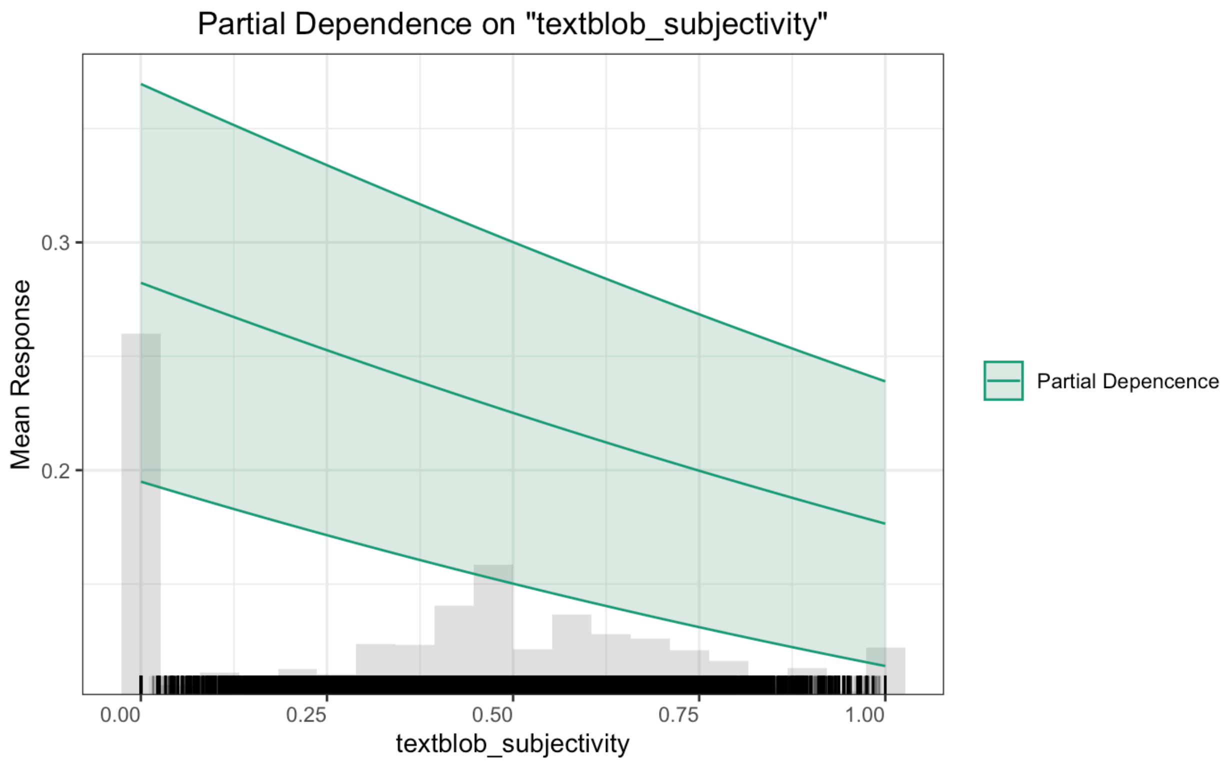
dependance to the success of the ICOs.


Partial Dependence on "vader_neutral"
Mean Response
0.5
0.4
0.3
0.2
0.1
0.0
0.00
0.25
0.50
0.75
1.00
vader_neutral
Partial Depencence


Partial Dependence on "vader_positive"
Mean Response
0.75
0.50
0.25
0.00
0.25
0.50
0.75
1.00
vader_positive
Partial Depencence


Partial Dependence on "vader_negative"
Mean Response
1.00
0.75
0.50
0.25
0.00
0.25
0.50
0.75
1.00
vader_negative
Partial Depencence


Partial Dependence on "cleaned_tweet_words"
Mean Response
0.4
0.3
0.2
0.1
0.0
0
20
40
60
cleaned_tweet_words
Partial Depencence



To conclude, is necessary to make some adjustment to the model.

```r
h2o.init(nthreads = -1)

x = c("vader_negative", "vader_neutral", "vader_positive", "vader_compound",
      "verified", "follows_count", "friends_count",
      "retweet_count", "number_of_likes", "tweet_words",
      "hashtags_total", "cleaned_tweet_words", "textblob_polarity",
      "textblob_subjectivity", "textblob_sentiment")

twitter_model_h20_glm <- h2o.glm(training_frame=twitter_db_h2o_train,
    x=x, y=y, family = "binomial", balance_classes = TRUE, nfolds = 5,
    seed = 573)

# View model information: training statistics, performance, important variables
summary(twitter_model_h20_glm)


# Predict using GLM model
pred = h2o.predict(object = twitter_model_h20_glm, newdata =
   twitter_db_h2o_test)


# Convert results in a dataframe to check

# predictions <- as.data.frame(pred)
# predictions
# final_db_h2o_test_df <- as.data.frame(final_db_h2o_test)
# results_prediction <- cbind(final_db_h2o_test_df, forecast = predictions$predict)
# results_prediction <- results_prediction %>% relocate(forecast, .after = success)



# Look at summary of predictions: probability of TRUE class (p1)
summary(pred)
```

```

# Sensitivity
GLM_Twitter_sensitivity = 2107308 / (2107308 + 369863)
GLM_Twitter_sensitivity
```

#### 9.1.2.3 Model 2

```
## [1] 0.8506914
```

```
# Specificity
GLM_Twitter_specifity = 1506564 / (1506564 + 791978)
GLM_Twitter_specifity
```

```
## [1] 0.6554433
```

Hence, the results are pretty clear, and show: - max f1: threshold = 0.233796; value = 0.458196
- max accuracy: threshold = 0.462029; value = 0.761622
- max precision: threshold = 0.462029; value = 0.638048 - max senitivity (recall): threshold = 0.006959; value = 1.000000
- max specificity: threshold = 0.993259; value = 0.999988

To conclude, this model is slighlty better compared to the previous one, and the main variables that impact the results are: 1 vader_neutral
2 cleaned_tweet_words 3 vader_positive
4 vader_negative
5 textblob_subjectivity
6 retweet_count
7 vader_compound
8 hashtags_total
9 number_of_likes 10 friends_count

```

# download the model built above to your local machine

# twitter_model_h20_glm <- h2o.download_model(twitter_model_h20_glm, path = "/Users/ettorefalde/Dataspe
```

#### 9.1.2.4 Download GLM

### 9.1.3 Distributed Random Forest (DRF)

Among the most effective tools for classification and regression is Distributed Random Forest (DRF). Instead of producing just one classification or regression tree in response to a batch of data, DRF creates a forest of them. Each of these trees is a subset of rows and columns weak learner. Increased tree cover will lower the variance. When making a final prediction, whether for a class or a numerical value, both classification and regression use the average prediction over all of their trees.

```
h2o.init(nthreads = -1)
# Build and train the model:
# twitter_model_h2o_drf <- h2o.randomForest(x = x,
#                                y = y,
#                                ntrees = 50,
#                                max_depth = 25,
#                                min_rows = 10,
#                                calibrate_model = TRUE,
#                                calibration_frame = twitter_db_h2o_test,
#                                binomial_double_trees = TRUE,
#                                training_frame = twitter_db_h2o_train,
#                                validation_frame = twitter_db_h2o_test)


twitter_model_h2o_drf <- h2o.upload_model("/Users/ettorefalde/DataspellProjects/ICO_Forecasting/R/DRF_m

# Eval performance:
perf <- h2o.performance(twitter_model_h2o_drf)
perf

# Variable importance
va_plot <- h2o.varimp_plot(twitter_model_h2o_drf)

learning_curve_plot <- h2o.learning_curve_plot(twitter_model_h2o_drf)
learning_curve_plot
```

```
# Sensitivity
DRF_Twitter_sensitivity = 3027953 / (3027953 + 402735)
DRF_Twitter_sensitivity
```

```
## [1] 0.8826081
```

```
# Specificity
DRF_Twitter_specifity = 585736 / (585736 + 759286)
DRF_Twitter_specifity
```

```
## [1] 0.4354843
```

The results of this model are pretty impressive. In fact, compared to the previous model I was able to get acceptable results.

- max f1: threshold = 0.314068; value = 0.605722

- max accuracy: threshold = 0.418700; value = 0.836884

- max precision: threshold = 0.999187; value = 1.000000
- max senitivity (recall): threshold = 0.006475; value = 1.000000

- max specificity: threshold = 0.999187; value = 0.999988

While, the binomial metrics are: - MSE: 0.1157803 - RMSE: 0.340265 - LogLoss: 0.3501232 - Mean Per-Class Error: 0.2543348 - AUC: 0.8652959 - AUCPR: 0.7275513 - Gini: 0.7305918 - R^2: 0.3711521

Moreover, we also have to consider that these results are taken without analyzing the ICOs characteristics.

To conclude, in the following images is possible to see the variables importance, and the learning curve of the model.

```

# download the model built above to your local machine
```

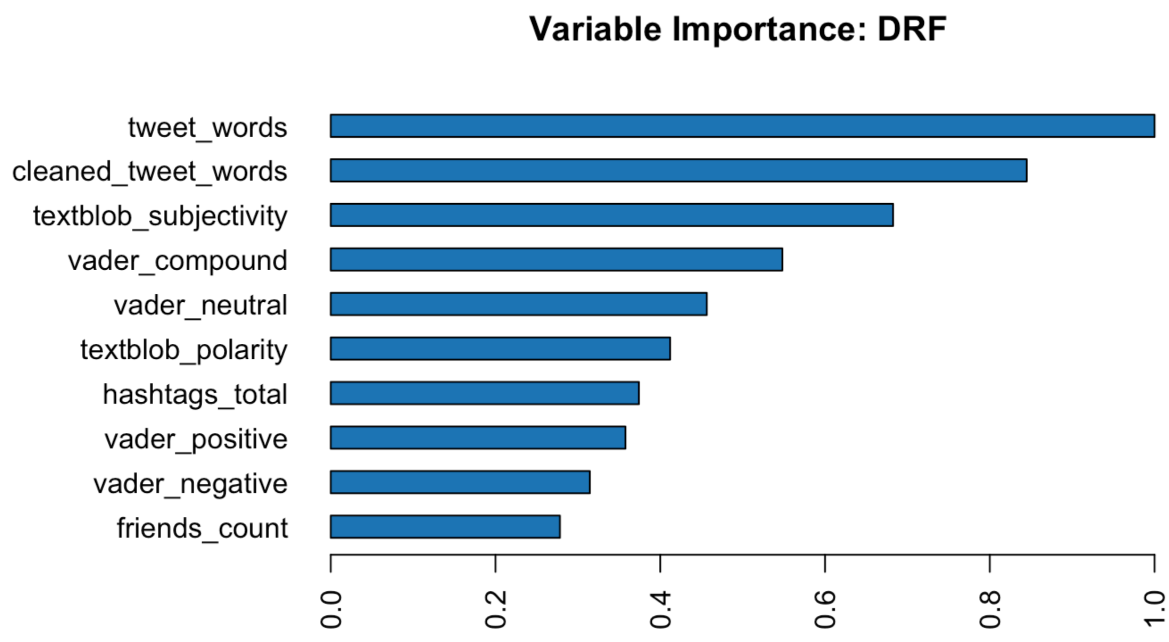


Figure 1: Variables importance in the DRF model

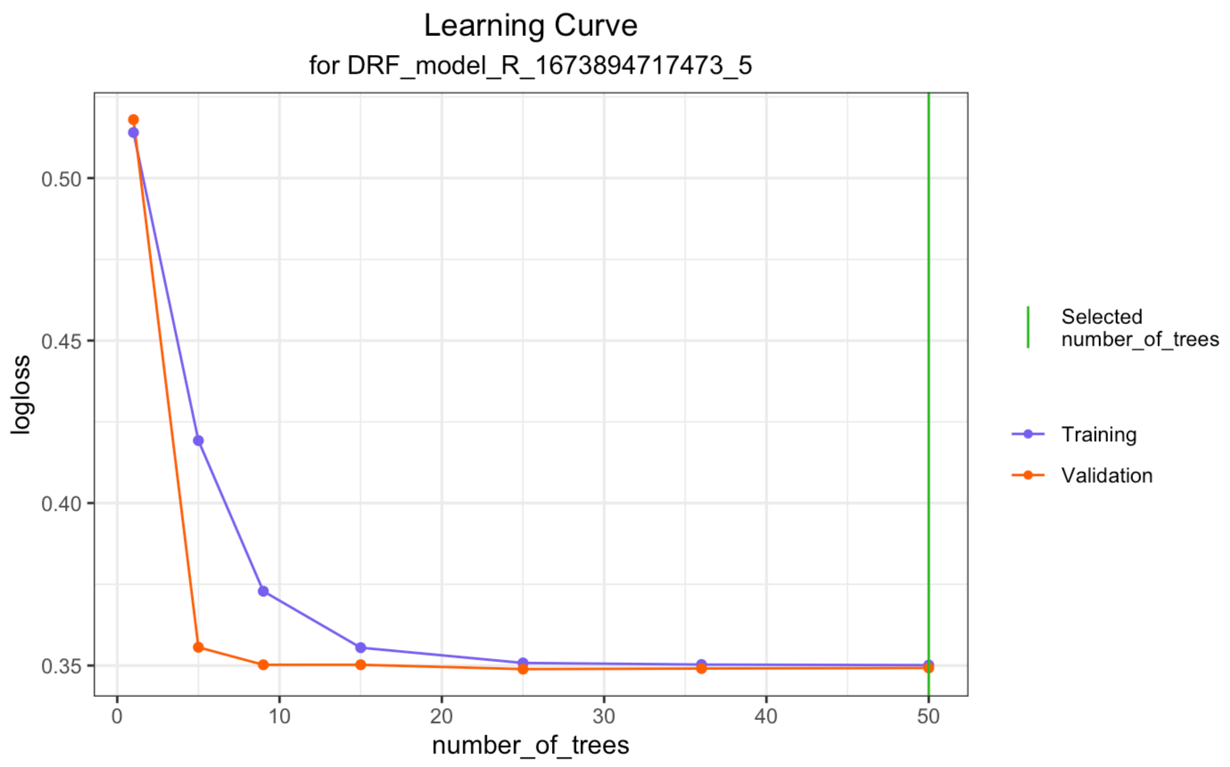


Figure 2: Lerning curve of the DRF model

```
# twitter_model_h2o_drf <- h2o.download_model(twitter_model_h2o_drf, path = "/Users/ettorefalde/Dataspe
```

#### 9.1.3.1 Download DRF

### 9.1.4 Gradient Boosting Machine (GBM)

The forward learning ensemble method Gradient Boosting Machine (for Regression and Classification) uses ensemble learning. The guiding heuristic states that accurate approximations can yield strong prediction results. By building each tree in parallel, the H2O's GBM successively constructs regression trees on each of the dataset's features.

Hence, in the following code is possible to see the code and the model parameters adopted.

```
h2o.init(nthreads = -1)


twitter_model_h2o_gbm <- h2o.gbm(y = y, x = x, training_frame
   = twitter_db_h2o_train, ntrees = 50, max_depth = 25, min_rows =
    1, learn_rate = 0.01, distribution= "multinomial")

# twitter_model_h2o_gbm <- h2o.upload_model("/Users/ettorefalde/DataspellProjects/ICO_Forecasting/R/GBM

# To obtain the Mean-squared Error by tree from the
twitter_model_h2o_gbm@model$scoring_history

twitter_model_h2o_gbm@model$training_metrics


h2o.performance(twitter_model_h2o_gbm)


# Predict using GBM model
pred = h2o.predict(object = twitter_model_h2o_gbm, newdata = twitter_db_h2o_test)

# Look at summary of predictions: probability of TRUE class (p1)
summary(pred$p1)

# Showing forecasts in a dataframe
# predictions <- as.data.frame(pred)
# predictions
# final_db_h2o_test_df <- as.data.frame(twitter_db_h2o_test)
#
# results_prediction <- cbind(final_db_h2o_test_df, forecast = predictions$predict)
# results_prediction <- results_prediction %>% relocate(forecast, .after = success)


va_plot <- h2o.varimp_plot(twitter_model_h2o_gbm)
```

```
# Sensitivity
GBM_Twitter_sensitivity = 3336004 / (3336004 + 286284)
GBM_Twitter_sensitivity
```

```
## [1] 0.920966
```

```r
# Specificity
GBM_Twitter_specifity = 278059 / (278059 + 278059)
GBM_Twitter_specifity
```

```
## [1] 0.5
```

In this case the results are:

- max f1: threshold = 0.285236; value = 0.752362

- max accuracy: threshold = 0.306364; value = 0.891786

- max precision: threshold = 0.579462; value = 1.000000
- max senitivity (recall): threshold = 0.147495; value = 1.000000

- max specificity: threshold = 0.579462; value = 1.000000

MSE: 0.1218119 RMSE: 0.3490156 LogLoss: 0.3994295 Mean Per-Class Error: 0.1688492 AUC: 0.9471321 AUCPR: 0.8778498 Gini: 0.8942643 R^2: 0.3383916

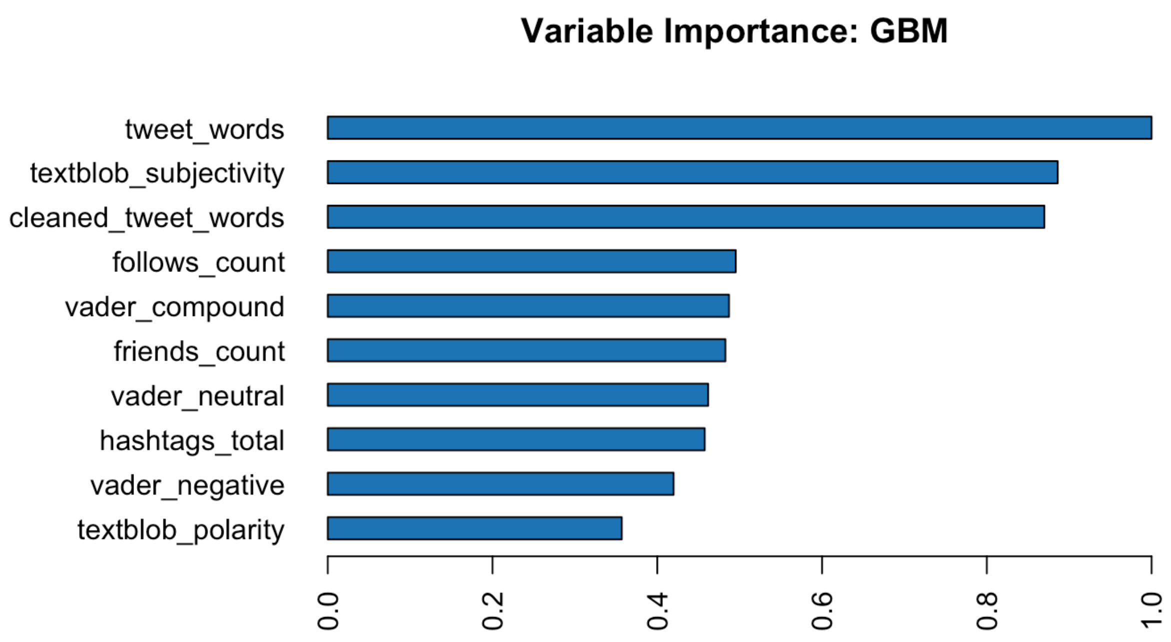


Figure 3: Variables importance in the GBM model

```r
# download the model built above to your local machine

# twitter_model_h2o_gbm <- h2o.download_model(twitter_model_h2o_gbm, path = "/Users/ettorefalde/Dataspe
```

#### 9.1.4.1 Download GBM

### 9.1.5 Naïve Bayes Classifier

Naive Bayes is a classification technique that applies the Bayes Theorem under the premise that covariates are independent. The Nave Bayes classifier makes the assumption that predictor variables are independent of one another conditional on the response, and that numerical predictors have a Gaussian distribution with mean and standard deviation calculated from the training dataset.

Alternatives to decision trees for classification issues include naive Bayes models. Every row in the training dataset with at least one NA will be completely skipped when creating a Naive Bayes classifier. When predicting, certain predictors are excluded from the probability computation if the test dataset contains missing values.

```
# twitter_model_h2o_naiveBayes <- h2o.naiveBayes(x = x,
#                               y = y,
#                               training_frame = twitter_db_h2o_train,
#                               laplace = 0,
#                               nfolds = 5,
#                               seed = 1234)


twitter_model_h2o_naiveBayes <- h2o.upload_model("/Users/ettorefalde/DataspellProjects/ICO_Forecasting/


# Eval performance:
perf <- h2o.performance(twitter_model_h2o_naiveBayes)
perf

# Generate the predictions on a test set (if necessary):
pred <- h2o.predict(twitter_model_h2o_naiveBayes, newdata = twitter_db_h2o_test)
pred
```

```
# Sensitivity
NB_Twitter_sensitivity = 2154250 / (2154250 + 431221)
NB_Twitter_sensitivity
```

```
## [1] 0.8332138
```

```
# Specificity
NB_Twitter_specifity = 1459439 / (1459439 + 730800)
NB_Twitter_specifity
```

```
## [1] 0.6663378
```

In this case the results are:

- max f1: threshold = 0.371500; value = 0.436004
- max accuracy: threshold = 0.996385; value = 0.763133
- max precision: threshold = 0.996385; value = 0.757028
- max senitivity (recall): threshold = 0.000417; value = 1.000000
- max specificity: threshold = 0.999492; value = 0.997086

MSE: 0.2396995 RMSE: 0.4895911 LogLoss: 0.7478979 Mean Per-Class Error: 0.3874799 AUC: 0.6294589 AUCPR: 0.3366893 Gini: 0.2589177

```
# download the model built above to your local machine


# twitter_model_h2o_naiveBayes <- h2o.download_model(twitter_model_h2o_naiveBayes, path = "/Users/ettor
```

#### 9.1.5.1 Download Naïve Bayes Classifier

### 9.1.6 Deep Learning (Neural Networks)

A multi-layer feedforward artificial neural network that is trained via stochastic gradient descent via back-propagation is the foundation of H2O's deep learning system. Numerous hidden layers made up of neurons with the tanh, rectifier, and maxout activation functions may be present in the network. High prediction accuracy is made possible by sophisticated features like adaptive learning rate, rate annealing, momentum training, dropout, L1 or L2 regularization, checkpointing, and grid search. Every compute node trains a duplicate of the global model parameters on its local data using multi-threading (asynchronously), and it periodically adds to the global model by network-wide averaging of the model parameters.

```
# Build and train the model:
# twittwe_model_h2o_dl <- h2o.deeplearning(x = x,
#                          y = y,
#                          hidden = c(1),
#                          epochs = 1000,
#                          train_samples_per_iteration = -1,
#                          reproducible = TRUE,
#                          activation = "Tanh",
#                          single_node_mode = FALSE,
#                          balance_classes = FALSE,
#                          force_load_balance = FALSE,
#                          seed = 23123,
#                          tweedie_power = 1.5,
#                          score_training_samples = 0,
#                          score_validation_samples = 0,
#                          training_frame = twitter_db_h2o_train,
#                          stopping_rounds = 0, max_runtime_secs = 300)

twitter_model_h2o_dl <- h2o.upload_model("/Users/ettorefalde/DataspellProjects/ICO_Forecasting/R/DeepLe

# Eval performance:
perf <- h2o.performance(twitter_model_h2o_dl)
perf

va_plot <- h2o.varimp_plot(twitter_model_h2o_dl)
```

```
# Sensitivity
NN_Twitter_sensitivity = 1786459 / (1786459 + 176959)
NN_Twitter_sensitivity
```

```
## [1] 0.909872
```

```
# Specificity
NN_Twitter_specifity = 1827385 / (1827385 + 984945)
NN_Twitter_specifity
```

```
## [1] 0.6497762
```

In this case the results are:

- max f1: threshold = 0.106600; value = 0.495665

- max accuracy: threshold = 0.375909; value = 0.595928

- max precision: threshold = 0.375909; value = 0.354755
- max senitivity (recall): threshold = 0.086787; value = 1.000000

- max specificity: threshold = 0.380764; value = 0.554323

MSE: 0.1682733 RMSE: 0.4102113 LogLoss: 0.507387 Mean Per-Class Error: 0.3289816 AUC: 0.6459952 AUCPR: 0.3163967 Gini: 0.2919904

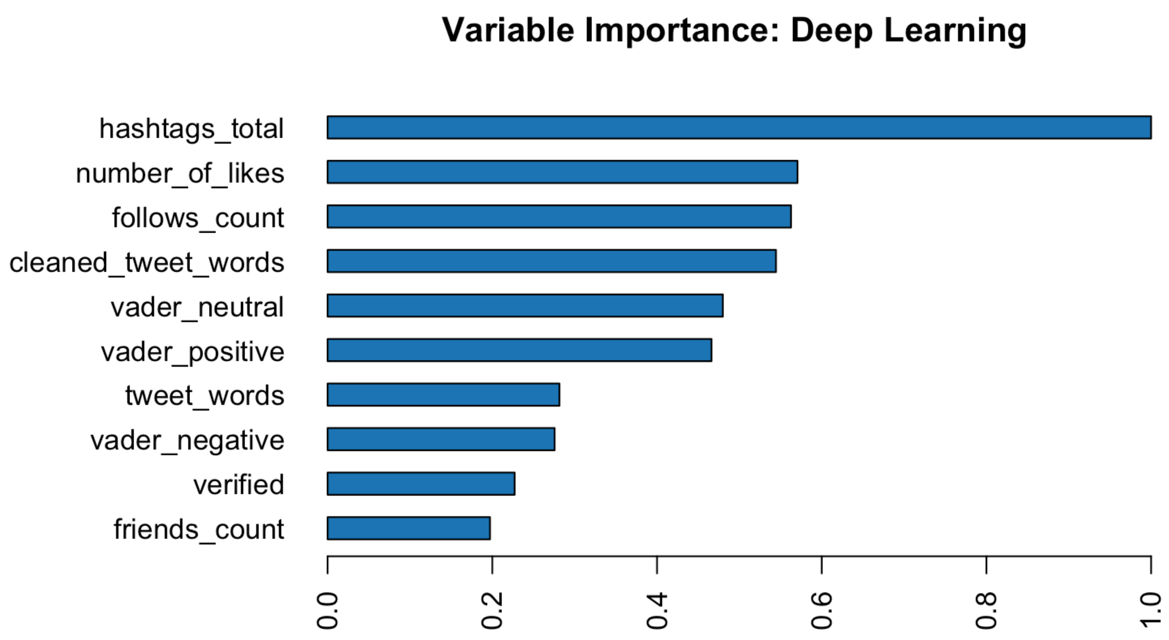


Figure 4: Variables importance in the DL model

```
# download the model built above to your local machine


# twittwe_model_h2o_dl <- h2o.download_model(twittwe_model_h2o_dl, path = "/Users/ettorefalde/Dataspell.
```

#### 9.1.6.1 Download Deep Learning Model

### 9.1.7 ML

Despite the increase in people entering the industry, the demand for machine learning professionals has recently overtaken the supply. There have been significant advancements made in the creation of non-experts' friendly machine learning software to close this gap. Creating straightforward, uniform interfaces to a range of machine learning algorithms was one of the first steps toward simplification of machine learning (e.g. H2O).

Although H2O has made it simple for novices to experiment with machine learning, highly effective machine learning models still necessitate a significant amount of data science expertise. Particularly Deep Neural Networks are notoriously challenging for a non-expert to correctly adjust. We have created a user-friendly interface that automates the process of training a large number of candidate models in order to make machine learning software really accessible to non-experts.H2O's AutoML can also be a useful tool for advanced users because it offers a straightforward wrapper function that executes a large number of modeling-related tasks that ordinarily require many lines of code, freeing up their time to concentrate on other data science pipeline tasks like feature engineering, model deployment, and data preprocessing.

The machine learning workflow, which involves the automatic training and adjustment of numerous models within a user-specified time-limit, can be automated using H2O's AutoML.

H2O provides a variety of model explainability techniques that work with both individual models and AutoML objects (groups of models) (e.g. leader model). A single function call can automatically create explanations, offering a user-friendly interface for perusing and illuminating the AutoML models.

```r
# Run AutoML for 20 base models
twitter_model_h2o_aml <- h2o.automl(x = x, y = y,
                  training_frame = twitter_db_h2o_train,
                  max_models = 20,
                  seed = 1, max_runtime_secs_per_model = 300)

# View the AutoML Leaderboard
lb <- twitter_model_h2o_aml@leaderboard
print(lb, n = nrow(lb))  # Print all rows instead of default (6 rows)
```

## 9.2 Final DB

My analysis is engind with this section. In fact, I tought interesting to compute the mean of the main variables in the *twitter_db*, collapse that datast in order to join it with the *tord_db*.

Doing so, I was able to have a dataset with both Twitter variables and ICOs characteristics variables.

### 9.2.1 Collapsing

First of all, I will collapse *twitter_db* my the mean values of its main columns and and join with *tord_db*.

```r
# keeping only the variables I am intered in
t <- twitter_db
t <- t %>% dplyr::select(ico_name, vader_negative, vader_neutral, vader_positive,
                         vader_compound, verified, follows_count, friends_count,
                           retweet_count, number_of_likes, tweet_words,
                           hashtags_total, cleaned_tweet_words,
                           textblob_polarity, textblob_subjectivity)

temp <- aggregate(cbind(vader_negative, vader_neutral, vader_positive,
                         vader_compound, verified, follows_count, friends_count,
                           retweet_count, number_of_likes, tweet_words,
                           hashtags_total, cleaned_tweet_words,
                           textblob_polarity, textblob_subjectivity) ~ ico_name,
                  data = t, FUN = mean)

# join twitter, and tord_db
final_db <- inner_join(tord_db, temp, by="ico_name")
```

```r
# removing unwanted cols
final_db <- subset(final_db, select = -c(pre_ico_start, pre_ico_end, ico_start, ico_end, token))

# check nulls
colnames(final_db)[apply(final_db, 2, anyNA)]
```

```
## character(0)
```

```r
# Remove vars...
rm(t, temp)
```

In conclusion, I obtain a dataset containing the main variables of both datasets.

Moving forward, having a lighter dataset I was also able to make models with both base R models and models with *h2o* package.

### 9.2.2 Logistic regression

First of all, I started with a logistic regression. This model predicts the probability to the outcome (y) beign true.

```r
#str(final_db)


t <- table(final_db$success)
t
```

```
##
##    0    1
## 4065 1294
```

```r
# Baseline accuracy
baseline_func(t)
```

```
##        0
## 0.758537
```

So, from the code above we can notice that the *baseline accuracy* = 0.758537

#### 9.2.2.1 Splitting Dataframe

In this step, I set a split ratio = 0.75, where 0.75 of the dataset is the training one, while the other 25% is for the test.

```r
set.seed(88)
split <- sample.split(final_db$success, SplitRatio = 0.75)

# Create training and test dataset
final_db_train <- subset(final_db, split == TRUE)
final_db_test <- subset(final_db, split == FALSE)
```

```r

GLM_model = glm(success ~ country_popularity + country_popularity_percentage +
                 is_ico +  is_ieo + is_sto + ico_duration + price_usd +
                  distributed_in_ico + sold_tokens + token_for_sale +
                  whitelist + kyc + bonus + restricted_areas_total +
                  bounty + mvp + pre_ico_duration + did_pre_ico +
                  pre_ico_price_usd + platform + link_white_paper +
                  linkedin_link + github_link + website + rating +
                  teamsize + coinmarketcap_identifier + erc20 + vader_negative +
                  vader_neutral + vader_positive + vader_compound +
                  verified + follows_count + friends_count + retweet_count +
                  number_of_likes + tweet_words + hashtags_total +
                  cleaned_tweet_words + textblob_polarity +
                  textblob_subjectivity,
data=final_db_train, family=binomial)

summary(GLM_model)
```

#### 9.2.2.2 Generating the model

```
##
## Call:
## glm(formula = success ~ country_popularity + country_popularity_percentage +
##     is_ico + is_ieo + is_sto + ico_duration + price_usd + distributed_in_ico +
##     sold_tokens + token_for_sale + whitelist + kyc + bonus +
##     restricted_areas_total + bounty + mvp + pre_ico_duration +
##     did_pre_ico + pre_ico_price_usd + platform + link_white_paper +
##     linkedin_link + github_link + website + rating + teamsize +
##     coinmarketcap_identifier + erc20 + vader_negative + vader_neutral +
##     vader_positive + vader_compound + verified + follows_count +
##     friends_count + retweet_count + number_of_likes + tweet_words +
##     hashtags_total + cleaned_tweet_words + textblob_polarity +
##     textblob_subjectivity, family = binomial, data = final_db_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0410  -0.6990  -0.5019  -0.1155   2.8514
##
## Coefficients: (3 not defined because of singularities)
##                                Estimate Std. Error z value Pr(>|z|)
## (Intercept)                   2.832e+02  3.113e+02   0.910 0.363083
## country_popularity           -1.463e-04  1.523e-04  -0.960 0.337050
## country_popularity_percentage        NA         NA      NA       NA
## is_ico                        6.107e-01  1.778e-01   3.435 0.000593 ***
## is_ieo                               NA         NA      NA       NA
## is_sto                        5.861e-01  4.007e-01   1.463 0.143492
## ico_duration                 -2.104e-03  1.518e-03  -1.386 0.165863
## price_usd                    -1.186e-05  2.570e-05  -0.461 0.644503
## distributed_in_ico            1.067e-04  3.621e-04   0.295 0.768352
## sold_tokens                   5.642e-11  8.062e-11   0.700 0.484023
## token_for_sale               -1.314e-12  2.273e-12  -0.578 0.563333
## whitelist                     4.394e-02  1.069e-01   0.411 0.680926
## kyc                          -1.081e-01  1.026e-01  -1.054 0.292003
## bonus                        -3.292e+00  5.961e-01  -5.522 3.36e-08 ***
## restricted_areas_total       -4.140e-03  6.620e-03  -0.625 0.531699
## bounty                        4.078e-02  1.035e-01   0.394 0.693589
## mvp                          -3.743e-01  1.175e-01  -3.187 0.001440 **
## pre_ico_duration             -3.853e-03  1.951e-03  -1.975 0.048295 *
## did_pre_ico                  -1.714e-01  9.976e-02  -1.718 0.085721 .
## pre_ico_price_usd            -1.688e-03  4.043e-03  -0.417 0.676356
## platform                     -7.716e-02  8.695e-02  -0.887 0.374841
## link_white_paper              4.253e-01  1.865e-01   2.280 0.022585 *
## linkedin_link                 3.245e-01  9.401e-02   3.451 0.000558 ***
## github_link                  -3.232e-01  2.604e-01  -1.241 0.214516
## website                              NA         NA      NA       NA
## rating                        4.841e-01  7.054e-02   6.862 6.78e-12 ***
## teamsize                      3.223e-02  6.066e-03   5.312 1.08e-07 ***
## coinmarketcap_identifier      1.389e+00  9.199e-02  15.101  < 2e-16 ***
## erc20                        -1.009e-01  1.262e-01  -0.800 0.423816
## vader_negative               -2.868e+02  3.116e+02  -0.920 0.357383
## vader_neutral                -2.858e+02  3.114e+02  -0.918 0.358713
## vader_positive               -2.889e+02  3.113e+02  -0.928 0.353469
## vader_compound                1.712e+00  1.029e+00   1.664 0.096054 .
## verified                     -1.438e+00  1.330e+00  -1.082 0.279462
## follows_count                 1.395e-06  2.150e-06   0.649 0.516607
## friends_count                 9.154e-07  9.395e-06   0.097 0.922377
## retweet_count                 3.353e-03  6.025e-03   0.556 0.577927
## number_of_likes               8.833e-03  4.819e-03   1.833 0.066827 .
## tweet_words                  -8.852e-02  4.865e-02  -1.820 0.068832 .
## hashtags_total                7.602e-02  5.636e-02   1.349 0.177372
## cleaned_tweet_words           2.611e-02  4.886e-02   0.534 0.593152
## textblob_polarity            -3.006e-02  7.715e-01  -0.039 0.968915
## textblob_subjectivity         5.493e-01  5.819e-01   0.944 0.345166
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4442.1  on 4018  degrees of freedom
## Residual deviance: 3715.1  on 3979  degrees of freedom
## AIC: 3795.1
##
## Number of Fisher Scoring iterations: 6
```

```
GLM_model = glm(success ~ is_ico + bonus + mvp + link_white_paper +
                  rating + teamsize + coinmarketcap_identifier,
data=final_db_train, family=binomial)




summary(GLM_model)
```

```
##
## Call:
## glm(formula = success ~ is_ico + bonus + mvp + link_white_paper +
##     rating + teamsize + coinmarketcap_identifier, family = binomial,
##     data = final_db_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1807  -0.6933  -0.5422  -0.1243   2.9020
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -3.760596   0.259459 -14.494  < 2e-16 ***
## is_ico                    0.608079   0.171092   3.554 0.000379 ***
## bonus                    -3.251580   0.590046  -5.511 3.57e-08 ***
## mvp                      -0.383423   0.107584  -3.564 0.000365 ***
## link_white_paper          0.462987   0.182592   2.536 0.011224 *
## rating                    0.367978   0.045885   8.019 1.06e-15 ***
## teamsize                  0.033775   0.005562   6.072 1.26e-09 ***
## coinmarketcap_identifier  1.489556   0.087289  17.065  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4442.1  on 4018  degrees of freedom
## Residual deviance: 3799.5  on 4011  degrees of freedom
## AIC: 3815.5
##
## Number of Fisher Scoring iterations: 6
```

```
predictTrain = predict(GLM_model, type="response")
summary(predictTrain)
```

#### 9.2.2.3 Predicting

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.002489 0.131882 0.189026 0.241354 0.299158 0.907230
```

```
t <- table(final_db_train$success, predictTrain > 0.5)

accuracy_func(t)
```

#### 9.2.2.4 Theresholding

```
## [1] 0.7842747
```

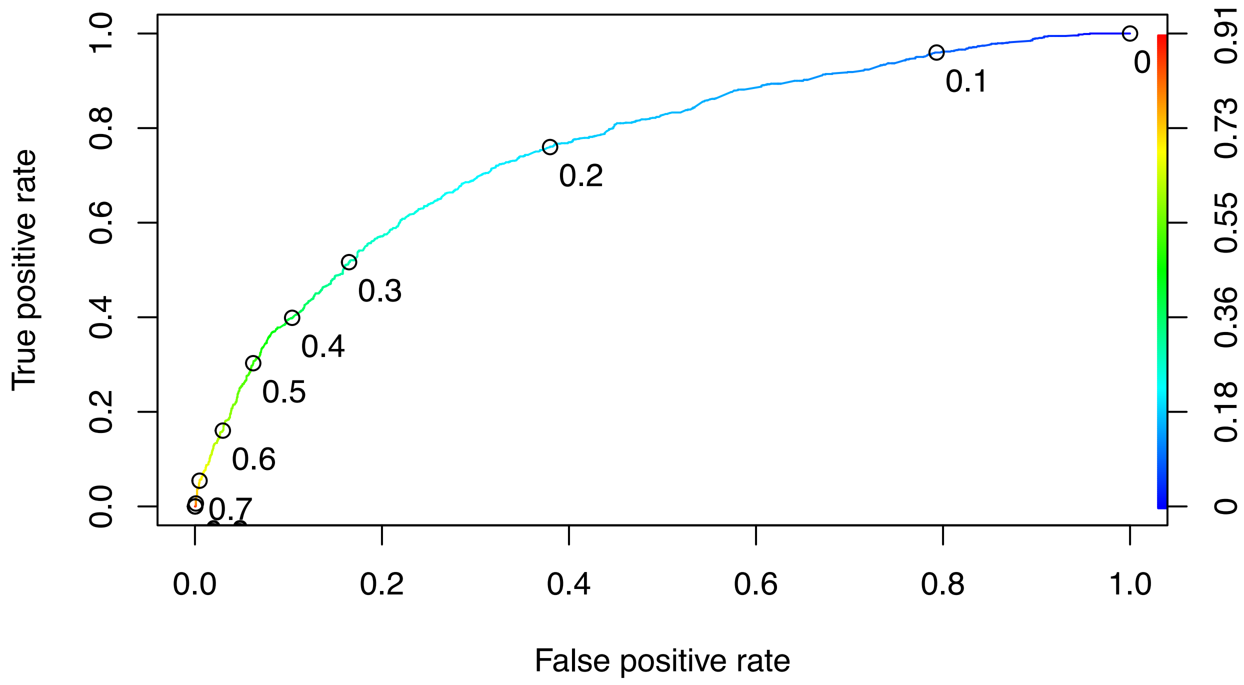
```
sensitivity_func(t)
```

```
## [1] 0.3020619
```

```
specificity_func(t)
```

```
## [1] 0.9376845
```

```
ROCRpred = prediction(predictTrain, final_db_train$success)
ROCRperf = performance(ROCRpred, "tpr", "fpr")
plot(ROCRperf, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1), text.adj=c(-0.2,1.7))
```



#### 9.2.2.5 ROC

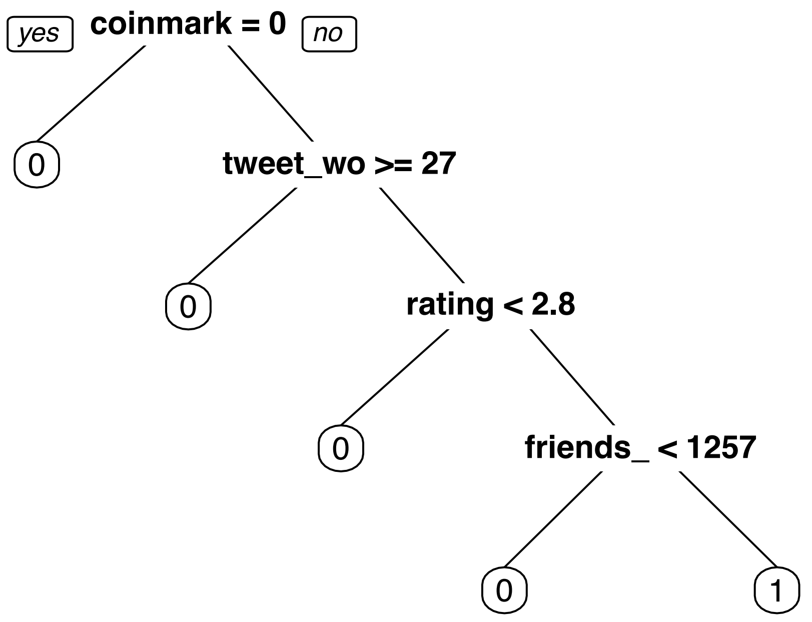
```
as.numeric(performance(ROCRpred, "auc")@y.values)
```

#### 9.2.2.6 Interpreting the data

```
## [1] 0.757041
```

### 9.2.3 CART

```
TREE_model = rpart(success ~ country_popularity + country_popularity_percentage +
                 is_ico +  is_ieo + is_sto + ico_duration + price_usd +
                  distributed_in_ico + sold_tokens + token_for_sale +
                  whitelist + kyc + bonus + restricted_areas_total +
                  bounty + mvp + pre_ico_duration + did_pre_ico +
                  pre_ico_price_usd + platform + link_white_paper +
                  linkedin_link + github_link + website + rating +
                  teamsize + coinmarketcap_identifier + erc20 + vader_negative +
                  vader_neutral + vader_positive + vader_compound +
                  verified + follows_count + friends_count + retweet_count +
                  number_of_likes + tweet_words + hashtags_total +
                  cleaned_tweet_words + textblob_polarity +
                  textblob_subjectivity, data = final_db_train, method="class", minbucket=25)


# success ~ is_ico + bonus + mvp + link_white_paper +
#                    rating + teamsize + coinmarketcap_identifier
prp(TREE_model)
```



#### Predicting

```r
PredictCART = predict(TREE_model, newdata = final_db_test, type = "class")
t <- table(final_db_test$success, PredictCART)

accuracy_func(t)
```

```
## [1] 0.7798507
```

```r
sensitivity_func(t)
```

```
## [1] 0.2592593
```

```r
specificity_func(t)
```

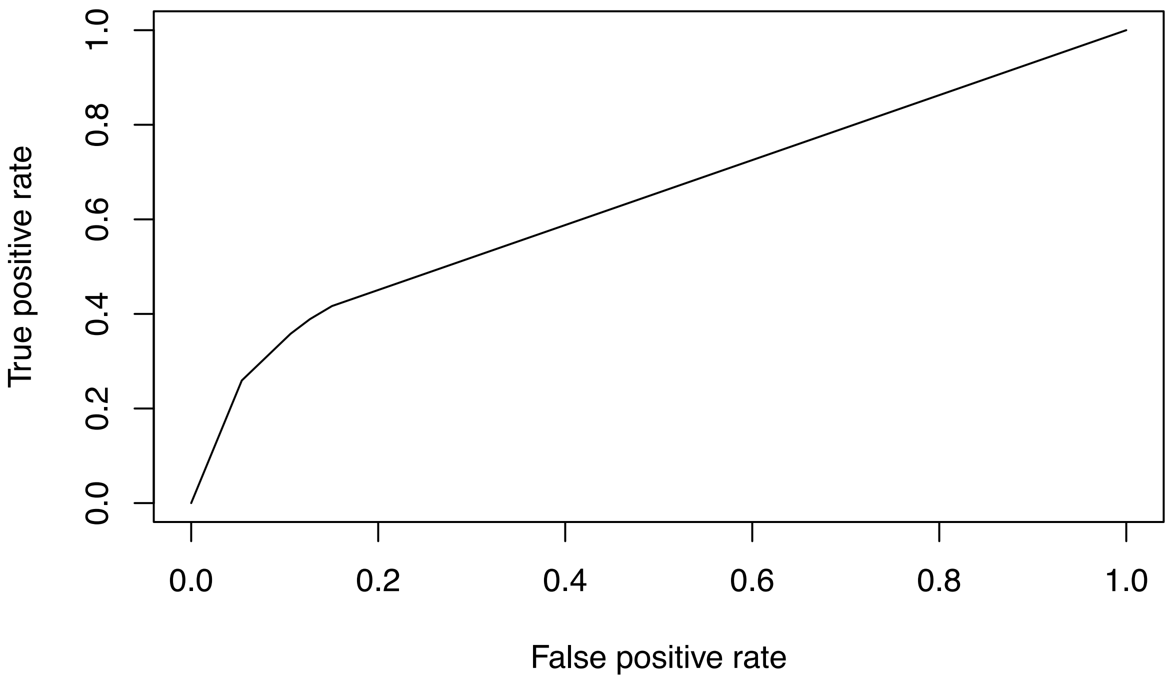
```
## [1] 0.9458661
```

```r
#PredictCART


PredictROC = predict(TREE_model, newdata = final_db_test)
table(final_db_test$success, PredictCART)
```

#### 9.2.3.1 Evaluate the model

```
##    PredictCART
##       0   1
##   0 961  55
##   1 240  84
```

```r
pred = prediction(PredictROC[,2], final_db_test$success)
perf = performance(pred, "tpr", "fpr")
plot(perf)
```



```
as.numeric(performance(pred, "auc")@y.values)
```

```
## [1] 0.6420163
```

### 9.2.4 Random Forest

```
#
final_db_train$success = as.factor(final_db_train$success)

RandomForest_model = randomForest(success ~ country_popularity + country_popularity_percentage +
                  is_ico +  is_ieo + is_sto + ico_duration + price_usd +
                   distributed_in_ico + sold_tokens + token_for_sale +
                   whitelist + kyc + bonus + restricted_areas_total +
                   bounty + mvp + pre_ico_duration + did_pre_ico +
                   pre_ico_price_usd + platform + link_white_paper +
                   linkedin_link + github_link + website + rating +
                   teamsize + coinmarketcap_identifier + erc20 + vader_negative +
                   vader_neutral + vader_positive + vader_compound +
                   verified + follows_count + friends_count + retweet_count +
                   number_of_likes + tweet_words + hashtags_total +
                   cleaned_tweet_words + textblob_polarity +
                   textblob_subjectivity, data = final_db_train, ntree=200, nodesize=25)

PredictForest = predict(RandomForest_model, newdata = final_db_test)
t <- table(final_db_test$success, PredictForest)
t
```

```
##    PredictForest
##       0   1
##   0 986  30
##   1 262  62
```

```
accuracy_func(t)
```

```
## [1] 0.7820896
```

```
sensitivity_func(t)
```

```
## [1] 0.191358
```

```
specificity_func(t)
```

```
## [1] 0.9704724
```

```
numFolds = trainControl( method = "cv", number = 10 )
cpGrid = expand.grid( .cp = seq(0.01,0.5,0.01))
train(success ~ country_popularity + country_popularity_percentage +
                 is_ico +  is_ieo + is_sto + ico_duration + price_usd +
                  distributed_in_ico + sold_tokens + token_for_sale +
                  whitelist + kyc + bonus + restricted_areas_total +
                  bounty + mvp + pre_ico_duration + did_pre_ico +
                  pre_ico_price_usd + platform + link_white_paper +
                  linkedin_link + github_link + website + rating +
                  teamsize + coinmarketcap_identifier + erc20 + vader_negative +
                  vader_neutral + vader_positive + vader_compound +
                  verified + follows_count + friends_count + retweet_count +
                  number_of_likes + tweet_words + hashtags_total +
                  cleaned_tweet_words + textblob_polarity +
                  textblob_subjectivity, data = final_db_train,
       method = "rpart", trControl = numFolds, tuneGrid = cpGrid )
```

#### 9.2.4.1 Cross Validation

```
## CART
##
## 4019 samples
##   42 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3617, 3617, 3617, 3617, 3618, 3617, ...
## Resampling results across tuning parameters:
##
##   cp    Accuracy   Kappa
##   0.01  0.7780561  0.25355518
##   0.02  0.7720853  0.25704453
##   0.03  0.7569050  0.07005142
##   0.04  0.7586463  0.00000000
##   0.05  0.7586463  0.00000000
##   0.06  0.7586463  0.00000000
##   0.07  0.7586463  0.00000000
##   0.08  0.7586463  0.00000000
##   0.09  0.7586463  0.00000000
##   0.10  0.7586463  0.00000000
##   0.11  0.7586463  0.00000000
##   0.12  0.7586463  0.00000000
##   0.13  0.7586463  0.00000000
##   0.14  0.7586463  0.00000000
##   0.15  0.7586463  0.00000000
##   0.16  0.7586463  0.00000000
##   0.17  0.7586463  0.00000000
##   0.18  0.7586463  0.00000000
##   0.19  0.7586463  0.00000000
##   0.20  0.7586463  0.00000000
##   0.21  0.7586463  0.00000000
##   0.22  0.7586463  0.00000000
##   0.23  0.7586463  0.00000000
##   0.24  0.7586463  0.00000000
##   0.25  0.7586463  0.00000000
##   0.26  0.7586463  0.00000000
##   0.27  0.7586463  0.00000000
##   0.28  0.7586463  0.00000000
##   0.29  0.7586463  0.00000000
##   0.30  0.7586463  0.00000000
##   0.31  0.7586463  0.00000000
##   0.32  0.7586463  0.00000000
##   0.33  0.7586463  0.00000000
##   0.34  0.7586463  0.00000000
##   0.35  0.7586463  0.00000000
##   0.36  0.7586463  0.00000000
##   0.37  0.7586463  0.00000000
##   0.38  0.7586463  0.00000000
##   0.39  0.7586463  0.00000000
##   0.40  0.7586463  0.00000000
##   0.41  0.7586463  0.00000000
##   0.42  0.7586463  0.00000000
##   0.43  0.7586463  0.00000000
##   0.44  0.7586463  0.00000000
##   0.45  0.7586463  0.00000000
##   0.46  0.7586463  0.00000000
##   0.47  0.7586463  0.00000000
##   0.48  0.7586463  0.00000000
##   0.49  0.7586463  0.00000000
##   0.50  0.7586463  0.00000000
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01.
```

```
# cp = 0.01

RandomForest_modelCV = rpart(success ~ country_popularity + country_popularity_percentage +
                 is_ico +  is_ieo + is_sto + ico_duration + price_usd +
                  distributed_in_ico + sold_tokens + token_for_sale +
                  whitelist + kyc + bonus + restricted_areas_total +
                  bounty + mvp + pre_ico_duration + did_pre_ico +
                  pre_ico_price_usd + platform + link_white_paper +
                  linkedin_link + github_link + website + rating +
                  teamsize + coinmarketcap_identifier + erc20 + vader_negative +
                  vader_neutral + vader_positive + vader_compound +
                  verified + follows_count + friends_count + retweet_count +
                  number_of_likes + tweet_words + hashtags_total +
                  cleaned_tweet_words + textblob_polarity +
                  textblob_subjectivity, data = final_db_train,
                 method="class", cp = 0.01)

PredictCV = predict(RandomForest_modelCV, newdata = final_db_test, type = "class")

t <- table(final_db_test$success, PredictCV)
t
```

```
##    PredictCV
##       0   1
##   0 961  55
##   1 240  84
```

```
accuracy_func(t)
```

```
## [1] 0.7798507
```

```
sensitivity(t)
```

```
## [1] 0.8001665
```

```
specificity_func(t)
```

```
## [1] 0.9458661
```

```
#PredictCV
```

### 9.2.5 h2o settings

```
# remove id columsn
final_db_h2o <- final_db[, -1]

# begin h2o instance
h2o.init()


# convert dataset in h2oFrame
final_db_h2o <- as.h2o(final_db_h2o)
final_db_h2o

# take a look to the content of the h2oFrame
h2o.describe(final_db_h2o)

# Conver as a Factor success variable
final_db_h2o$success <- h2o.asfactor(final_db_h2o$success)

# Construct test and train sets using sampling
final_db_h2o_split = h2o.splitFrame(data = final_db_h2o,
   ratios = 0.85)
final_db_h2o_train = final_db_h2o_split[[1]]
final_db_h2o_train

final_db_h2o_test = final_db_h2o_split[[2]]
final_db_h2o_test
```

```
# # Display a summary using table-like functions
h2o.table(final_db_h2o_train$success)
h2o.table(final_db_h2o_test$success)


# Set predictor and response variables
y = "success"
x = c("country_popularity", "country_popularity_percentage",
      "is_ico", "is_ieo", "is_sto", "ico_duration", "price_usd",
      "distributed_in_ico", "sold_tokens",
      "token_for_sale", "whitelist",
      "kyc", "bonus", "restricted_areas_total",
      "bounty", "mvp", "pre_ico_duration", "did_pre_ico",
      "pre_ico_price_usd", "platform",
      "link_white_paper", "linkedin_link", "github_link",
      "website", "rating", "teamsize", "coinmarketcap_identifier", "erc20",
      "vader_negative", "vader_positive", "vader_neutral", "vader_compound",
      "follows_count",
      "friends_count", "retweet_count", "number_of_likes", "tweet_words",
      "hashtags_total", "cleaned_tweet_words", "textblob_polarity",
      "textblob_subjectivity")
```

#### 9.2.5.1 Converting in *h2o* dataframe

### 9.2.6 Generalized Linear Models (GLM)

```
final_model_h2o_glm <- h2o.glm(training_frame=final_db_h2o_train,
    x=x, y=y, family = "binomial", alpha = 0.5,
    validation_frame = final_db_h2o_test, balance_classes = TRUE, nfolds = 10,
    seed = 573)

# Print the coefficients table
final_model_h2o_glm@model$coefficients_table

# Predict using GLM model
pred = h2o.predict(object = final_model_h2o_glm, newdata =
   final_db_h2o_test)

summary(pred)
```

```
h2o.explain(final_model_h2o_glm, final_db_h2o_test)
```

```
# Sensitivity
GLM_Final_sensitivity = 432 / (432 + 68)
GLM_Final_sensitivity
```

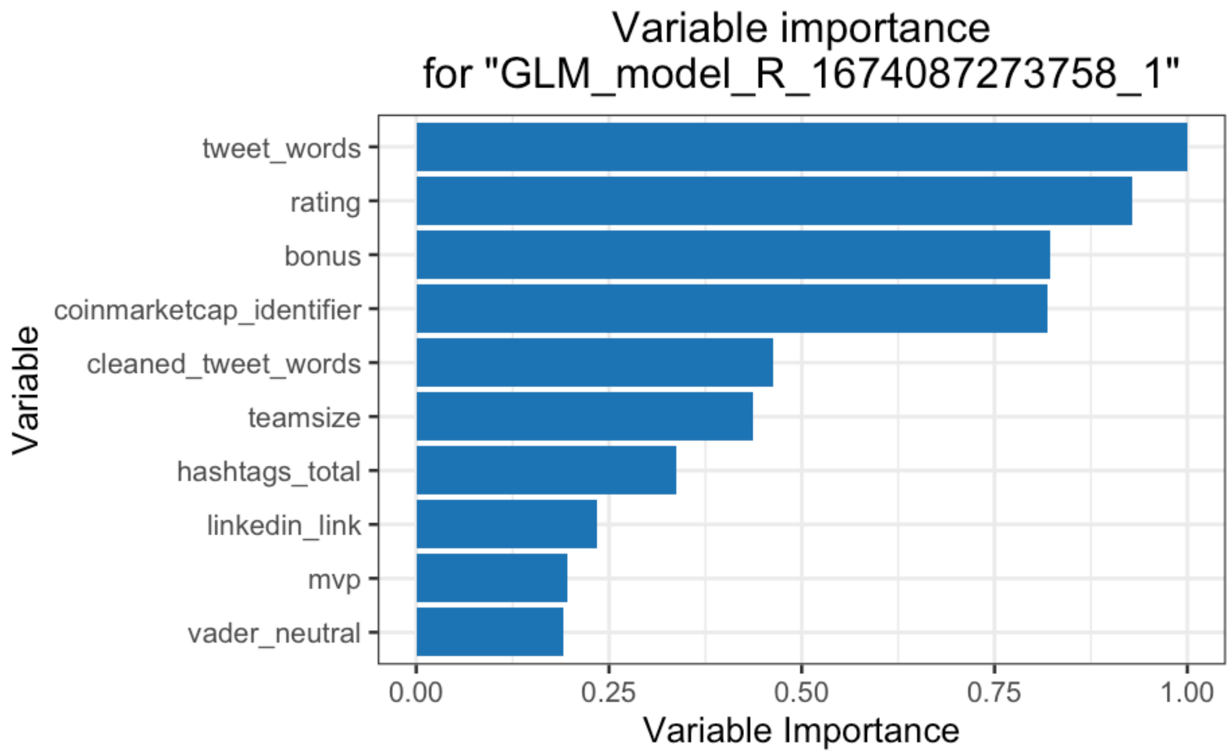
#### 9.2.6.1 Explaining GLM

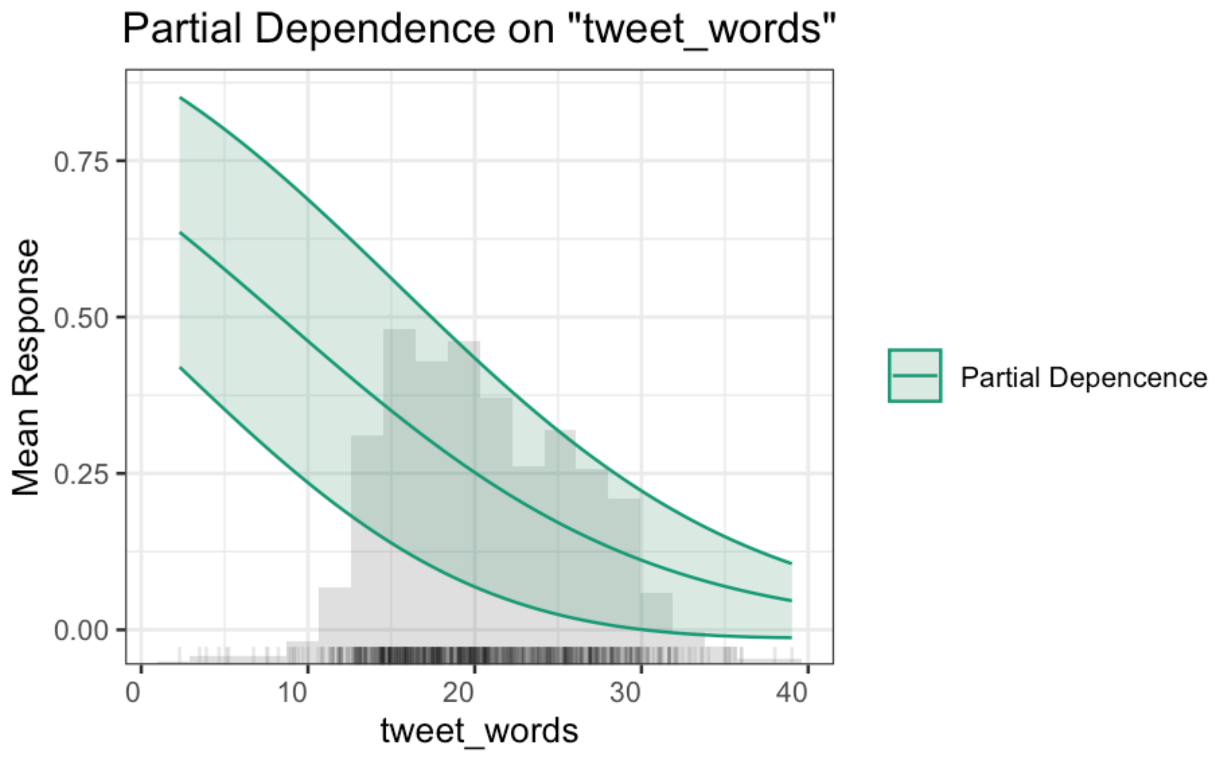
```
## [1] 0.864
```

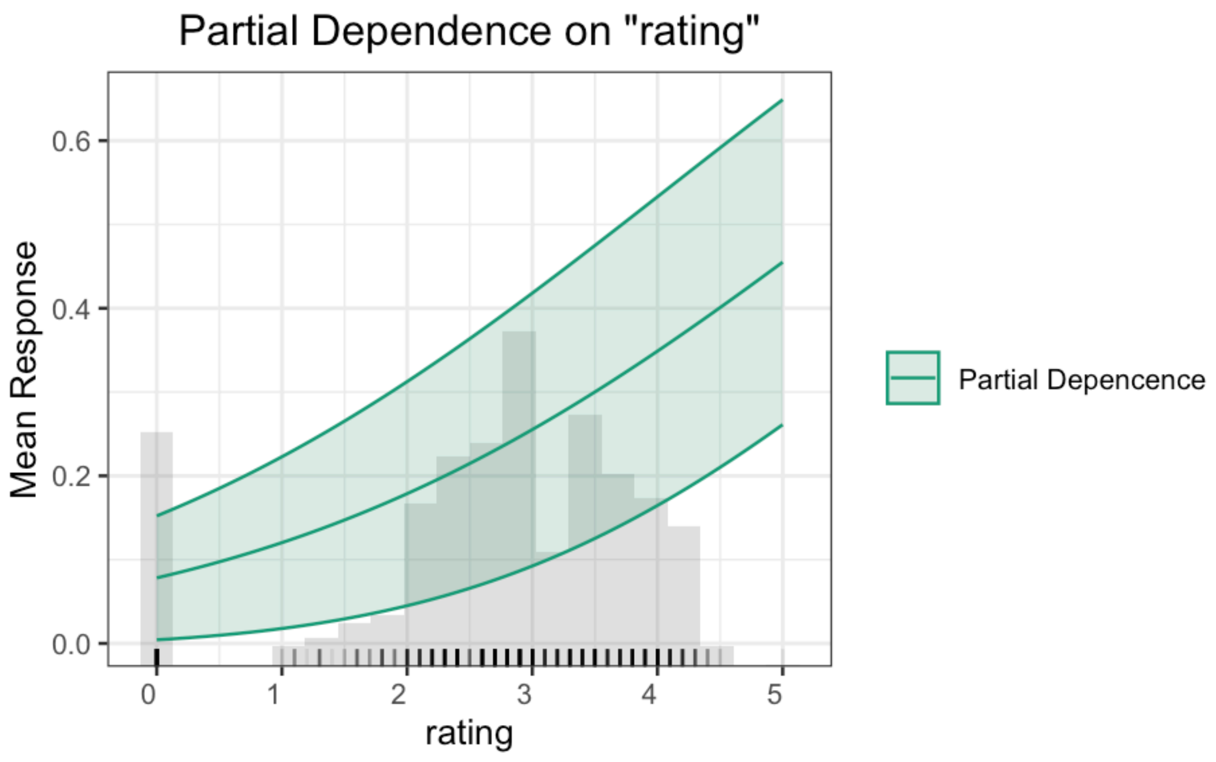
```
# Specificity
GLM_Final_specifity = 173 / (173 + 125)
```

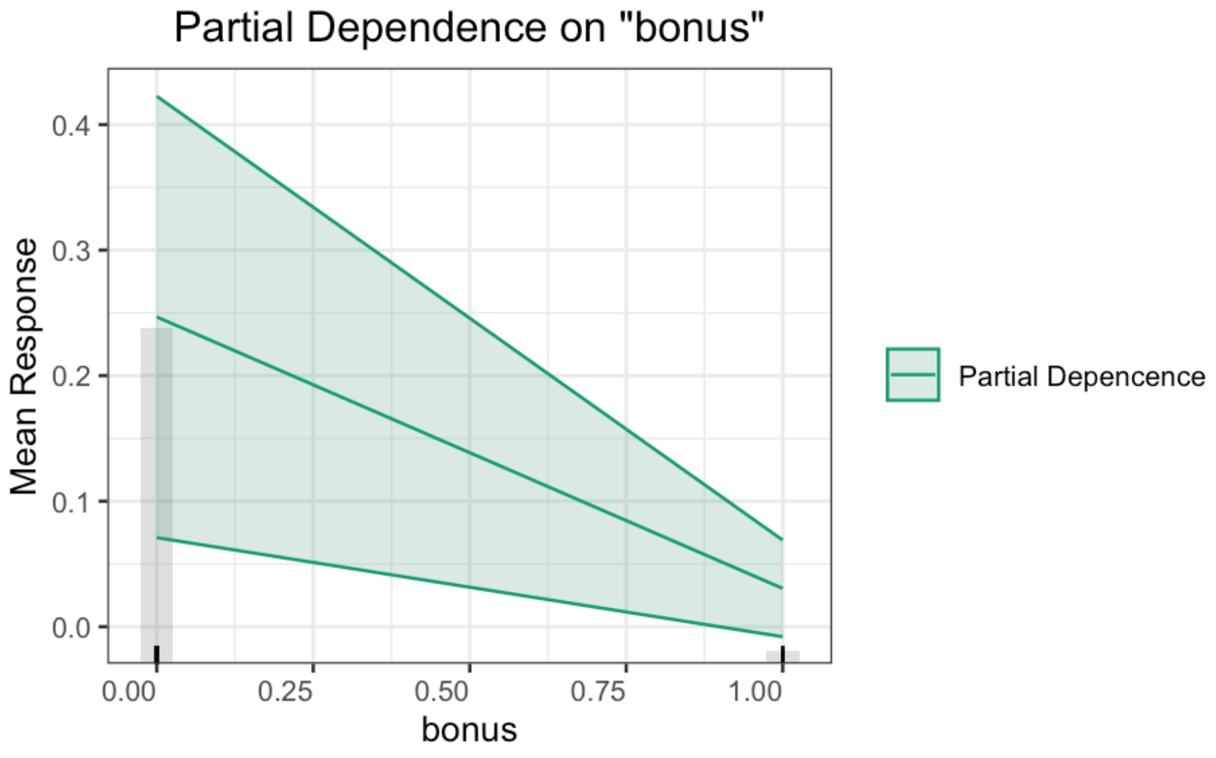
```
GLM_Final_specifity
```

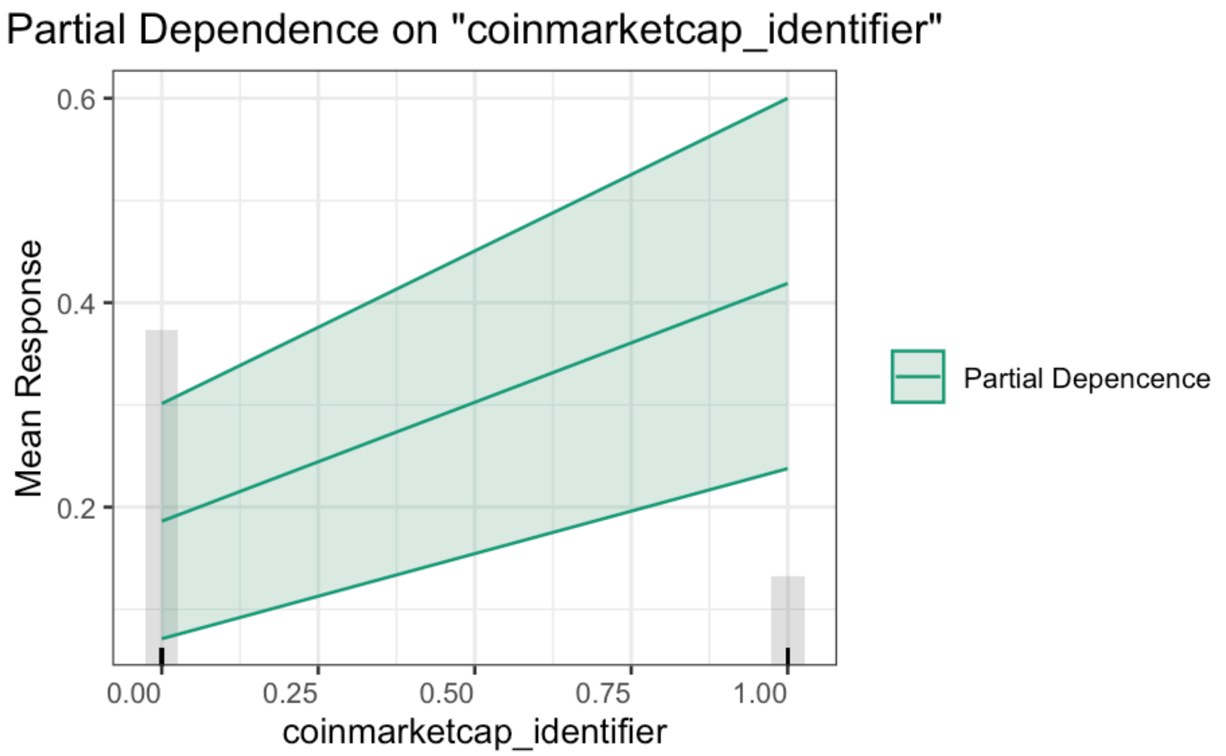
```
## [1] 0.5805369
```

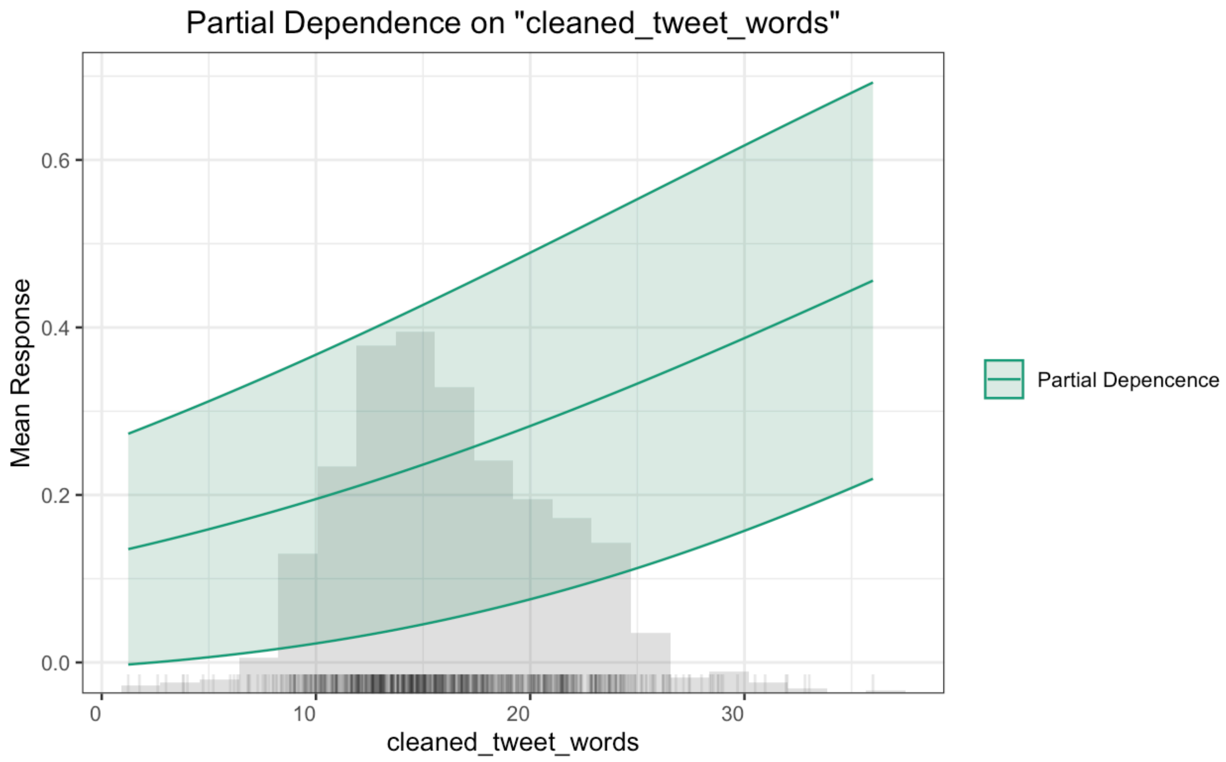





Partial Dependence on "rating"
Mean Response
0.0
0.2
0.4
0.6
0
1
2
3
4
5
rating
Partial Depencence


Partial Dependence on "bonus"
Mean Response
0.0
0.1
0.2
0.3
0.4
0.00
0.25
0.50
0.75
1.00
bonus
Partial Depencence





### 9.2.7 Distributed Random Forest (DRF)

```
h2o.init(nthreads = -1)
```

```
# Build and train the model:
final_model_h2o_drf <- h2o.randomForest(x = x,
                              y = y,
                              ntrees = 200,
                              max_depth = 25,
                              min_rows = 10,
                              calibrate_model = TRUE,
                              calibration_frame = final_db_h2o_train,
                              binomial_double_trees = TRUE,
                              training_frame = final_db_h2o_train,
                              validation_frame = final_db_h2o_test,
                              nfolds = 5)

# Eval performance:
perf <- h2o.performance(final_model_h2o_drf)
perf


va_plot <- h2o.varimp_plot(final_model_h2o_drf)
learning_curve_plot <- h2o.learning_curve_plot(final_model_h2o_drf)
learning_curve_plot
```

```
# Sensitivity
DRF_Final_sensitivity = 2454 / (2454 + 309)
DRF_Final_sensitivity
```

```
## [1] 0.888165
```

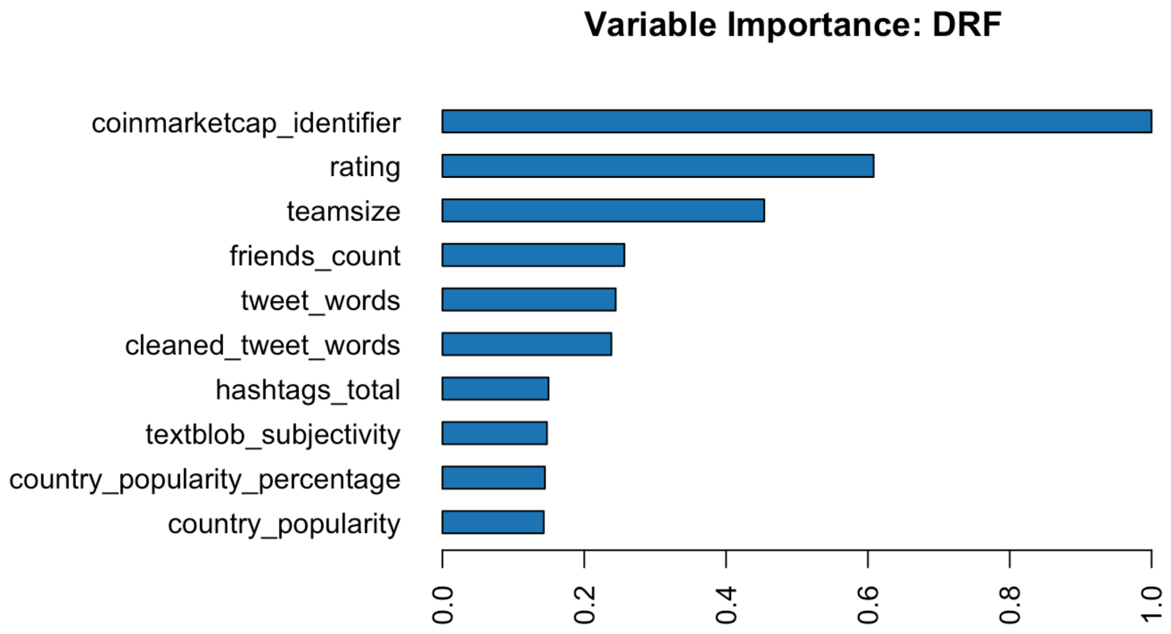
```
# Specificity
DRF_Final_specifity = 1049 / (1049 + 765)
DRF_Final_specifity
```

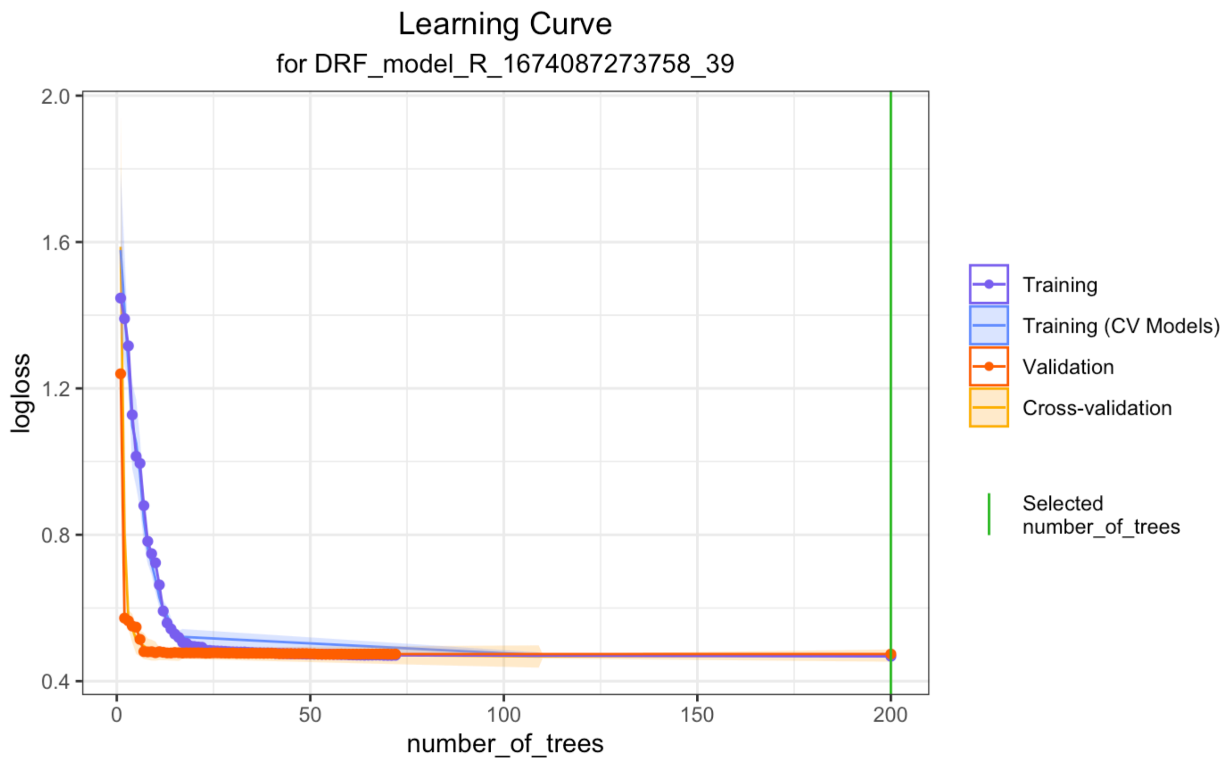
```
## [1] 0.57828
```

In this case the results are:

- max f1: threshold = 0.254381; value = 0.544874
- max accuracy: threshold = 0.435570; value = 0.787852
- max precision: threshold = 0.739857; value = 1.000000
- max senitivity (recall): threshold = 0.023401; value = 1.000000
- max specificity: threshold = 0.739857; value = 1.000000

MSE: 0.1512032 RMSE: 0.3888485 LogLoss: 0.4680392 Mean Per-Class Error: 0.2881624 AUC: 0.7704185 AUCPR: 0.5143923 Gini: 0.5408371 R^2: 0.1748612





### 9.2.8 Gradient Boosting Machine (GBM)

```
h2o.init(nthreads = -1)
```

```
# final_model_h2o_gbm <- h2o.gbm(y = y, x = x, training_frame
#    = final_db_h2o_train, ntrees = 1000, max_depth = 25, min_rows =
#     1, learn_rate = 0.01, distribution= "multinomial")

final_model_h2o_gbm <- h2o.gbm(x = x,
                    y = y,
                    nfolds = 5,
                    seed = 1111,
                    keep_cross_validation_predictions = TRUE,
                    training_frame = final_db_h2o_train)

# To obtain the Mean-squared Error by tree from the
final_model_h2o_gbm@model$scoring_history

final_model_h2o_gbm@model$training_metrics


h2o.performance(final_model_h2o_gbm)


# Predict using GBM model
pred = h2o.predict(object = final_model_h2o_gbm, newdata =
   final_db_h2o_test)

# Look at summary of predictions: probability of TRUE class (p1)
summary(pred$p1)

# Showing forecasts in a dataframe
predictions <- as.data.frame(pred)
predictions
final_db_h2o_test_df <- as.data.frame(final_db_h2o_test)

results_prediction <- cbind(final_db_h2o_test_df, forecast = predictions$predict)
results_prediction <- results_prediction %>% relocate(forecast, .after = success)



va_plot <- h2o.varimp_plot(final_model_h2o_gbm)
```

```
# Sensitivity
GBM_Final_sensitivity = 3116 / (3116 + 330)
GBM_Final_sensitivity
```

```
## [1] 0.9042368
```

```
# Specificity
GBM_Final_specifity = 387 / (387 + 744)
GBM_Final_specifity
```

```
## [1] 0.3421751
```

In this case the results are:

- max f1: threshold = 0.361900; value = 0.701117

- max accuracy: threshold = 0.400020; value = 0.858790

- max precision: threshold = 0.889980; value = 1.000000
- max senitivity (recall): threshold = 0.061727; value = 1.000000

- max specificity: threshold = 0.889980; value = 1.000000

MSE: 0.1100558 RMSE: 0.3317466 LogLoss: 0.3604949 Mean Per-Class Error: 0.1991648 AUC: 0.896855 AUCPR: 0.7767386 Gini: 0.7937099 R^2: 0.3994087

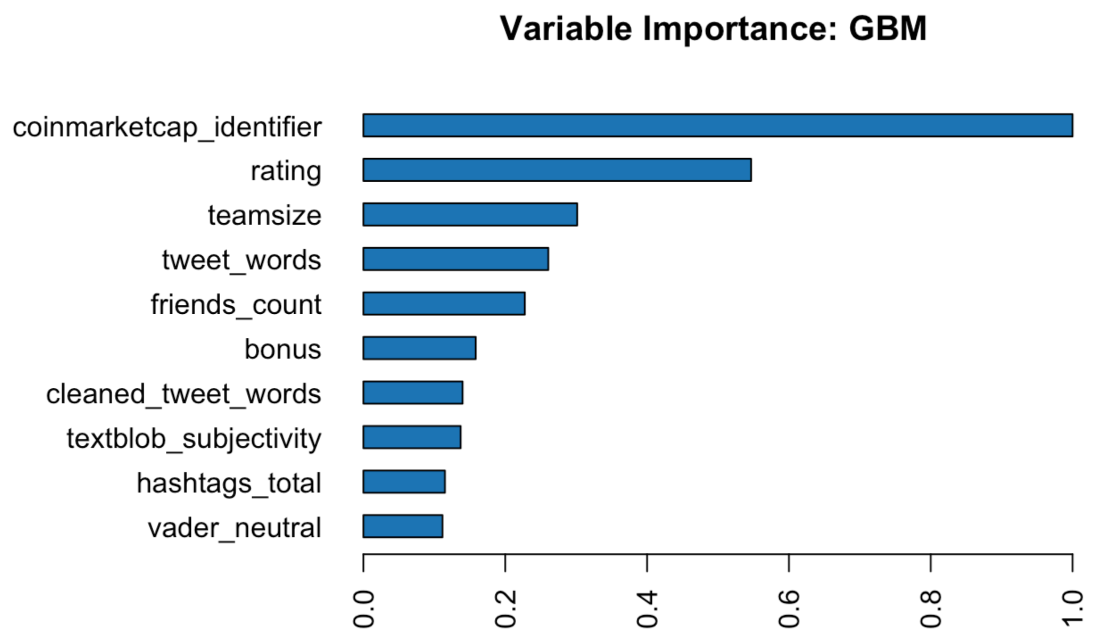


Figure 5: Variable importance

### 9.2.9 Naïve Bayes Classifier

```
final_model_h2o_naiveBayes <- h2o.naiveBayes(x = x,
                          y = y,
                          training_frame = final_db_h2o_train,
                          laplace = 0,
                          nfolds = 5,
                          seed = 1234)


# Eval performance:
perf <- h2o.performance(final_model_h2o_naiveBayes)
perf

# Generate the predictions on a test set (if necessary):
pred <- h2o.predict(final_model_h2o_naiveBayes, newdata = final_db_h2o_test)
pred
```

### 9.2.10 Deep Learning (Neural Networks)

```
# Build and train the model:
final_model_h2o_dl <- h2o.deeplearning(x = x,
                                       y = y,
                                       hidden = c(1),
                                       epochs = 1000,
                                       train_samples_per_iteration = -1,
                                       reproducible = TRUE,
                                       activation = "Tanh",
                                       single_node_mode = FALSE,
                                       balance_classes = FALSE,
                                       force_load_balance = FALSE,
                                       seed = 23123,
                                       tweedie_power = 1.5,
                                       score_training_samples = 0,
                                       score_validation_samples = 0,
                                       training_frame = final_db_h2o_train,
                                       stopping_rounds = 0)


# Eval performance:
perf <- h2o.performance(final_model_h2o_dl)
perf



va_plot <- h2o.varimp_plot(final_model_h2o_dl)
```

```
# Sensitivity
NA_Final_sensitivity = 2594 / (2594 + 370)
NA_Final_sensitivity
```

```
## [1] 0.8751687
```

```
# Specificity
NA_Final_specifity = 866 / (866 + 731)
NA_Final_specifity
```

```
## [1] 0.5422668
```

In this case the results are:

- max f1: threshold = 0.302630; value = 0.546959
- max accuracy: threshold = 0.549821; value = 0.788517
- max precision: threshold = 0.604743; value = 0.752809
- max senitivity (recall): threshold = 0.091305; value = 1.000000
- max specificity: threshold = 0.607544; value = 0.996492

MSE: 0.1504625 RMSE: 0.3878949 LogLoss: 0.4673524 Mean Per-Class Error: 0.2941289 AUC: 0.7670906 AUCPR: 0.5174154 Gini: 0.5341812

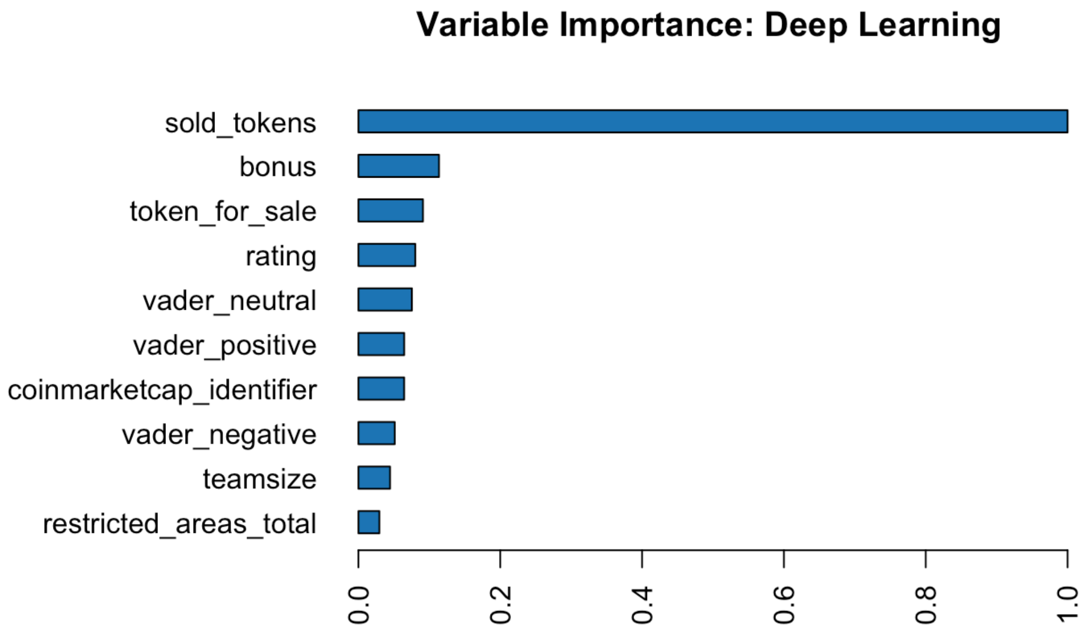


Figure 6: Variable importance

#### 9.2.11 ML

```
# Run AutoML for 20 base models
final_model_h2o_aml <- h2o.automl(x = x, y = y,
                  training_frame = final_db_h2o_train,
                  max_models = 20,
                  seed = 1)

# View the AutoML Leaderboard
lb <- final_model_h2o_aml@leaderboard
print(lb, n = nrow(lb))

# The leader model is stored here
final_model_h2o_aml@leader

# If you need to generate predictions on a test set, you can make
# predictions directly on the `"H2OAutoML"` object, or on the leader
# model object directly

pred <- h2o.predict(final_model_h2o_aml, final_db_h2o_test)  # predict(aml, test) also works

# or:
pred <- h2o.predict(final_model_h2o_aml@leader, final_db_h2o_test)
pred
```

```
# Get leaderboard with all possible columns
lb <- h2o.get_leaderboard(object = final_model_h2o_aml, extra_columns = "ALL")
lb_table <- as.data.frame(lb)


# Methods for an AutoML object
h2o.varimp_heatmap(final_model_h2o_aml)
h2o.model_correlation_heatmap(final_model_h2o_aml, final_db_h2o_test)
h2o.pd_multi_plot(final_model_h2o_aml, final_db_h2o_test, "textblob_polarity")
```

```
# Sensitivity
ML_Final_sensitivity = 3179 / (3179 + 264)
ML_Final_sensitivity
```

```
## [1] 0.9233227
```

```
# Specificity
ML_Final_specifity = 335 / (335 + 806)
ML_Final_specifity
```

```
## [1] 0.2936021
```

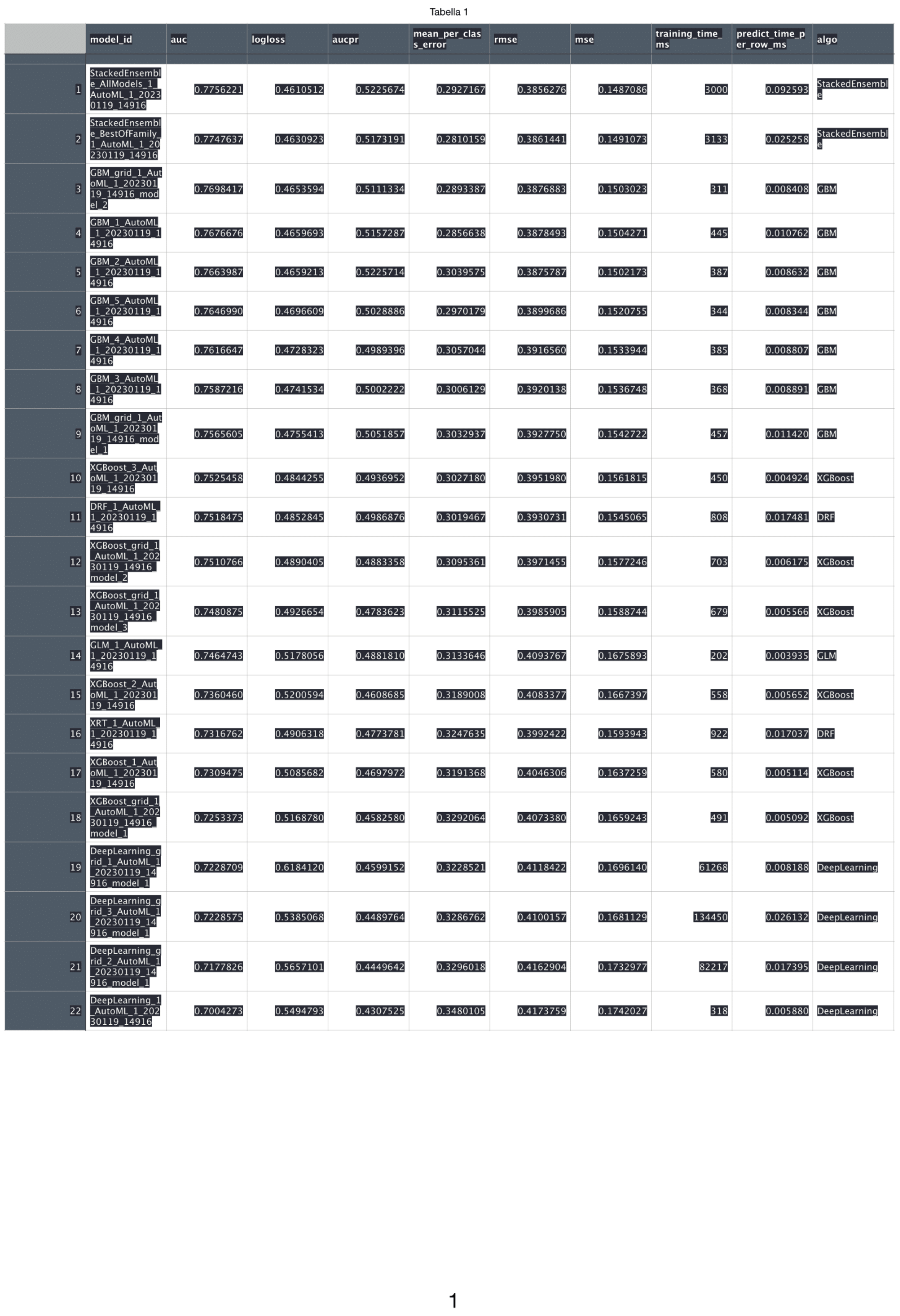


Tabella 1

| | model_id | auc | logloss | aucpr | mean_per_class_error | rmse | mse | training_time_ms | predict_time_per_row_ms | algo |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | StackedEnsemble_AllModels_1_AutoML_1_20230119_14916 | 0.7756221 | 0.4610512 | 0.5225674 | 0.2927167 | 0.3856276 | 0.1487086 | 3000 | 0.092593 | StackedEnsemble |
| 2 | StackedEnsemble_BestOfFamily_1_AutoML_1_20230119_14916 | 0.7747637 | 0.4630923 | 0.5173191 | 0.2810159 | 0.3861441 | 0.1491073 | 3133 | 0.025258 | StackedEnsemble |
| 3 | GBM_grid_1_AutoML_1_20230119_14916_model_2 | 0.7698417 | 0.4653594 | 0.5111334 | 0.2893387 | 0.3876883 | 0.1503023 | 311 | 0.008408 | GBM |
| 4 | GBM_1_AutoML_1_20230119_14916 | 0.7676676 | 0.4659693 | 0.5157287 | 0.2856638 | 0.3878493 | 0.1504271 | 445 | 0.010762 | GBM |
| 5 | GBM_2_AutoML_1_20230119_14916 | 0.7663987 | 0.4659213 | 0.5225714 | 0.3039575 | 0.3875787 | 0.1502173 | 387 | 0.008632 | GBM |
| 6 | GBM_5_AutoML_1_20230119_14916 | 0.7646990 | 0.4696609 | 0.5028886 | 0.2970179 | 0.3899686 | 0.1520755 | 344 | 0.008344 | GBM |
| 7 | GBM_4_AutoML_1_20230119_14916 | 0.7616647 | 0.4728323 | 0.4989396 | 0.3057044 | 0.3916560 | 0.1533944 | 385 | 0.008807 | GBM |
| 8 | GBM_3_AutoML_1_20230119_14916 | 0.7587216 | 0.4741534 | 0.5002222 | 0.3006129 | 0.3920138 | 0.1536748 | 368 | 0.008891 | GBM |
| 9 | GBM_grid_1_AutoML_1_20230119_14916_model_1 | 0.7565605 | 0.4755413 | 0.5051857 | 0.3032937 | 0.3927750 | 0.1542722 | 457 | 0.011420 | GBM |
| 10 | XGBoost_3_AutoML_1_20230119_14916 | 0.7525458 | 0.4844255 | 0.4936952 | 0.3027180 | 0.3951980 | 0.1561815 | 450 | 0.004924 | XGBoost |
| 11 | DRF_1_AutoML_1_20230119_14916 | 0.7518475 | 0.4852845 | 0.4986876 | 0.3019467 | 0.3930731 | 0.1545065 | 808 | 0.017481 | DRF |
| 12 | XGBoost_grid_1_AutoML_1_20230119_14916_model_2 | 0.7510766 | 0.4890405 | 0.4883358 | 0.3095361 | 0.3971455 | 0.1577246 | 703 | 0.006175 | XGBoost |
| 13 | XGBoost_grid_1_AutoML_1_20230119_14916_model_3 | 0.7480875 | 0.4926654 | 0.4783623 | 0.3115525 | 0.3985905 | 0.1588744 | 679 | 0.005566 | XGBoost |
| 14 | GLM_1_AutoML_1_20230119_14916 | 0.7464743 | 0.5178056 | 0.4881810 | 0.3133646 | 0.4093767 | 0.1675893 | 202 | 0.003935 | GLM |
| 15 | XGBoost_2_AutoML_1_20230119_14916 | 0.7360460 | 0.5200594 | 0.4608685 | 0.3189008 | 0.4083377 | 0.1667397 | 558 | 0.005652 | XGBoost |
| 16 | XRT_1_AutoML_1_20230119_14916 | 0.7316762 | 0.4906318 | 0.4773781 | 0.3247635 | 0.3992422 | 0.1593943 | 922 | 0.017037 | DRF |
| 17 | XGBoost_1_AutoML_1_20230119_14916 | 0.7309475 | 0.5085682 | 0.4697972 | 0.3191368 | 0.4046306 | 0.1637259 | 580 | 0.005114 | XGBoost |
| 18 | XGBoost_grid_1_AutoML_1_20230119_14916_model_1 | 0.7253373 | 0.5168780 | 0.4582580 | 0.3292064 | 0.4073380 | 0.1659243 | 491 | 0.005092 | XGBoost |
| 19 | DeepLearning_grid_1_AutoML_1_20230119_14916_model_1 | 0.7228709 | 0.6184120 | 0.4599152 | 0.3228521 | 0.4118422 | 0.1696140 | 61268 | 0.008188 | DeepLearning |
| 20 | DeepLearning_grid_3_AutoML_1_20230119_14916_model_1 | 0.7228575 | 0.5385068 | 0.4489764 | 0.3286762 | 0.4100157 | 0.1681129 | 134450 | 0.026132 | DeepLearning |
| 21 | DeepLearning_grid_2_AutoML_1_20230119_14916_model_1 | 0.7177826 | 0.5657101 | 0.4449642 | 0.3296018 | 0.4162904 | 0.1732977 | 82217 | 0.017395 | DeepLearning |
| 22 | DeepLearning_1_AutoML_1_20230119_14916 | 0.7004273 | 0.5494793 | 0.4307525 | 0.3480105 | 0.4173759 | 0.1742027 | 318 | 0.005880 | DeepLearning |

1

The table of models is:

In this case the results of the leader model are:

- max f1: threshold = 0.352035; value = 0.852769
- max accuracy: threshold = 0.398021; value = 0.929506
- max precision: threshold = 0.911254; value = 1.000000
- max senitivity (recall): threshold = 0.122644; value = 1.000000
- max specificity: threshold = 0.911254; value = 1.000000

H2OBinomialMetrics: stackedensemble ** Reported on cross-validation data. 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) **

MSE: 0.1487086 RMSE: 0.3856276 LogLoss: 0.4610512 Mean Per-Class Error: 0.2927167 AUC: 0.7756221 AUCPR: 0.5225674 Gini: 0.5512442

# 10 References

[1] dplyr: Hadley Wickham, Romain François, Lionel Henry, and Kirill Müller (2021). dplyr: A Grammar of Data Manipulation. R package version 1.0.7. https://CRAN.R-project.org/package=dplyr

[2] readr: Hadley Wickham and Jim Hester (2021). readr: Read Rectangular Text Data. R package version 2.1.1. https://CRAN.R-project.org/package=readr

[3] data.table: Matt Dowle and Arun Srinivasan (2021). data.table: Extension of Data.frame. R package version 1.14.2. https://CRAN.R-project.org/package=data.table

[4] janitor: Sam Firke (2021). janitor: Simple Tools for Examining and Cleaning Dirty Data. R package version 2.1.0. https://CRAN.R-project.org/package=janitor

[5] tidyverse: Hadley Wickham (2017). tidyverse: Easily Install and Load the 'Tidyverse'. R package version 1.3.1. https://CRAN.R-project.org/package=tidyverse

[6] lubridate: Vitalie Spinu, Garrett Grolemund, and Hadley Wickham (2021). lubridate: Make Dealing with Dates a Little Easier. R package version 1.8.0. https://CRAN.R-project.org/package=lubridate

[7] ggplot2: Hadley Wickham (2016). ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York. ISBN 978-3-319-24277-4, https://ggplot2.tidyverse.org

[8] hrbrthemes: Ryan Timpe and Claus O. Wilke (2021). hrbrthemes: Additional Themes, Theme Components, and Utilities for 'ggplot2'. R package version 0.8.1. https://CRAN.R-project.org/package=hrbrthem es

[9] chron: Kurt Hornik and Gabor Grothendieck (2020). chron: Chronological Objects which can Handle Dates and Times. R package version 2.3-56. https://CRAN.R-project.org/package=chron

[10] gridExtra: Baptiste Auguie (2017). gridExtra: Miscellaneous Functions for "Grid" Graphics. R package version 2.3. https://CRAN.R-project.org/package=gridExtra

[11] Simple Features for R (sf): Pebesma, E., & Bivand, R. (2021). sf: Simple Features for R. R package version 1.0-2. https://CRAN.R-project.org/package=sf

[12] Natural Earth map data (rnaturalearth): South, A. (2018). rnaturalearth: World Map Data from Natural Earth. R package version 0.1.0. https://CRAN.R-project.org/package=rnaturalearth

[13] Natural Earth map data (rnaturalearthdata): South, A. (2017). rnaturalearthdata: World Map Data from Natural Earth. R package version 0.1.0. https://CRAN.R-project.org/package=rnaturalearthdata

[14] Terrain Analysis using Digital Elevation Models (terra): Hijmans, R. J. (2021). terra: Spatial Data Analysis using Terra. R package version 1.4-22. https://CRAN.R-project.org/package=terra

[15] Datasets for Spatial Analysis (spData): Pebesma, E., & Bivand, R. (2020). spData: Datasets for Spatial Analysis. R package version 0.3.8. https://CRAN.R-project.org/package=spData

[16] Large datasets for Spatial Analysis (spDataLarge): Pebesma, E., & Bivand, R. (2021). spDataLarge: Large datasets for Spatial Analysis. R package version 0.5.2. https://CRAN.R-project.org/package=spDa taLarge

[17] Color schemes for maps and graphics (RColorBrewer): Neuwirth, E. (2014). RColorBrewer: ColorBrewer Palettes. R package version 1.1-2. https://CRAN.R-project.org/package=RColorBrewer

[18] Authoring Books and Technical Documents with R Markdown (bookdown): Xie, Y. (2020). bookdown: Authoring Books and Technical Documents with R Markdown. R package version 0.22. https://CRAN.R-project.org/package=bookdown

[19] Spatial Visualization with ggplot2 (ggmap): Kahle, D., & Wickham, H. (2013). ggmap: Spatial Visualization with ggplot2. The R Journal, 5(1), 144-161. https://journal.r-project.org/archive/2013-1/kahle-wickham.pdf

[20] caTools: “caTools: Tools: moving window statistics, GIF, Base64, ROC AUC, etc.” R package version 1.18.2. https://CRAN.R-project.org/package=caTools

[21] ROCR: “ROCR: Visualizing the Performance of Scoring Classifiers.” R package version 1.0-11. https://CRAN.R-project.org/package=ROCR

[22] rpart: “Recursive Partitioning and Regression Trees.” R package version 4.1-15. https://CRAN.R-project.org/package=rpart

[23] rpart.plot: “Plot ‘rpart’ Models: An Enhanced Version of ‘plot.rpart’.” R package version 3.1. https://CRAN.R-project.org/package=rpart.plot

[24] randomForest: “Random Forests for Classification and Regression.” R package version 4.6-14. https://CRAN.R-project.org/package=randomForest

[25] caret: “Classification and Regression Training.” R package version 6.0-93. https://CRAN.R-project.org/package=caret

[26] e1071: “Miscellaneous Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien.” R package version 1.7-10. https://CRAN.R-project.org/package=e1071

[27] class: “Functions for Classification.” R package version 7.3-19. https://CRAN.R-project.org/package=class

[28] h2o: “R Interface for ‘H2O’.” R package version 3.34.0.1. https://CRAN.R-project.org/package=h2o