1.0 Introduction

Crowdfunding, a widely adopted method of raising funds for projects or businesses, involves multiple participants engaging through online platforms. Key players include project initiators, financial backers, and facilitating entities, often digital platforms, which coordinate activities to expedite project launches.

Traditional crowdfunding models involve backers investing funds for monetary rewards. However, Initial Coin Offerings (ICOs) have introduced an innovative fundraising approach. Unlike traditional methods, ICOs utilize blockchain technology to issue and sell digital currencies, or "cryptocurrencies," to online investors. (Fisch,2019) These tokens not only hold trading value but also serve functional roles within the fundraising entity, such as in digital applications.

ICOs operate on an 'all-or-nothing' basis, setting fundraising targets within specified timeframes. Success is achieved if the project garners sufficient backing to meet its goal; otherwise, no returns are realized. Project descriptions and team profiles are typically showcased on online campaign pages to attract potential investors.

2.0 Business Understanding

The purpose of this activity is to examine the business context and objectives related to predicting the success of a team/company’s fundraising campaigns conducted through Initial Coin Offerings (ICOs) by employing different supervised machine learning models. The data is split into train and test, and we shall employ possible predictors to predict the success or failure of the campaign.

3.0 DATA UNDERSTANDING

Below table, describes the list of variables and short definitions, which could be used as prospective predictors.

Variable	Definition
ID	Unique ID number.
success	If 'Y' project achieved the goal, else 'N'.
startDate	Campaign start date.
endDate	Campaign end date.
coinNum	Number of blockchain coins to be issued.
priceUSD	Blockchain coin price (USD).
teamSize	Number of team members.
countryRegion	Country of fundraising team/company.
brandSlogan	Slogan of the company.
rating	Rating score given by investment experts, ranged 1(very poor to 5 (very good).
malinvestment	1 – Project was set up with minimum investment amount, else 0.
distributedPercentage	% of blockchain coin distributed to investors relative to all coins created.
platform	Blockchain platform for the fundraising project.
hasVideo	1 – project provided video on campaign page, else 0.
hasGithub	1 – project provided Github page on campaign page, else 0.
hasReddit	1 – project provided Reddit page on campaign page, else 0.

Hmisc is used to describe the variables in the dataset, as it provides useful insights for data analysis providing descriptive statistics, distinct /missing values, lowest and the highest values depending upon the data, whether they are quantitative or qualitative in nature.

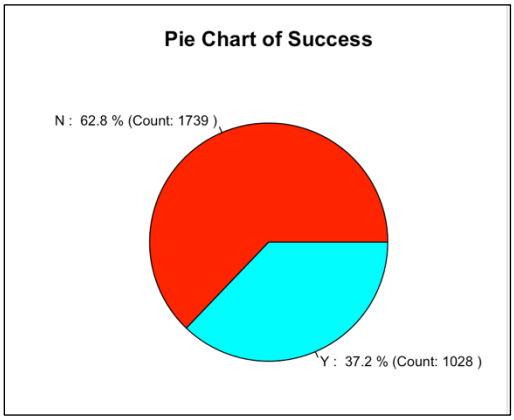
```
install.packages("Hmisc")
library(Hmisc)
describe(wbcd)
```

```
> describe(wbcd)
wbcd

 16  Variables      2767  Observations
---------------------------------------------------------------------------------------------------------------
ID
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50      .75
    2767        0     2767        1     1384    922.7    139.3    277.6    692.5   1384.0   2075.5
     .90      .95
  2490.4   2628.7

lowest :    1    2    3    4    5, highest: 2763 2764 2765 2766 2767
---------------------------------------------------------------------------------------------------------------
success
       n  missing distinct
    2767        0        2

Value          N     Y
Frequency   1739  1028
Proportion 0.628 0.372
---------------------------------------------------------------------------------------------------------------
brandSlogan
       n  missing distinct
    2767        0     2763

lowest : «Uberization» Platform for Offline Services      0-Cost Cloud For IoT, Web, Enterprise, dApps     0% Commissions on Purchases
1 Blockchain of Precious Metals  Mines           1 Coin Used in 100 Services
highest: Zamanat.                                  Zero commission decentralized freelance platform ZoneX eSports Platform
ZooomEx - Your Great Choice                       Cryptocurrency and a Digital Payment System
---------------------------------------------------------------------------------------------------------------
hasVideo
       n  missing distinct     Info      Sum     Mean      Gmd
    2767        0        2    0.597     2009   0.7261   0.3979

---------------------------------------------------------------------------------------------------------------
```

```
rating
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50      .75
    2767        0       39    0.998    3.121   0.8154      2.0      2.2      2.6      3.1      3.7
     .90      .95
     4.1      4.2

lowest : 1   1.1 1.2 1.3 1.4, highest: 4.4 4.5 4.6 4.7 4.8
---------------------------------------------------------------------------------------------------------------
priceUSD
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50      .75
    2587      180      274    0.998    19.01     37.7    0.000    0.010    0.040    0.120    0.500
     .90      .95
   1.114    2.835

lowest : 0      0.01   0.02   0.03   0.04  , highest: 642.4  647.74 888.88 1000   39384
---------------------------------------------------------------------------------------------------------------
countryRegion
       n  missing distinct
    2696       71      120

lowest : Afghanistan Andorra     Anguilla    Argentina   Armenia
highest: USA         Vanuatu     Venezuela   Vietnam     Zimbabwe
---------------------------------------------------------------------------------------------------------------
startDate
       n  missing distinct
    2767        0      760

lowest : 01/01/2018 01/01/2019 01/01/2020 01/02/2018 01/02/2019
highest: 31/08/2018 31/10/2017 31/10/2018 31/10/2019 31/12/2018
---------------------------------------------------------------------------------------------------------------
endDate
       n  missing distinct
    2767        0      776

lowest : 01/01/2018 01/01/2019 01/01/2020 01/02/2018 01/02/2019
highest: 31/10/2018 31/10/2019 31/12/2017 31/12/2018 31/12/2019
---------------------------------------------------------------------------------------------------------------
teamSize
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50      .75
    2613      154       53    0.998    13.11    8.454        4        5        7       12       17
     .90      .95
      23       28
```

```
lowest :  1  2  3  4  5, highest: 57 67 72 73 75
---------------------------------------------------------------------------------------------------------------
hasGithub
       n  missing distinct     Info      Sum     Mean      Gmd
    2767        0        2    0.732     1599   0.5779    0.488

---------------------------------------------------------------------------------------------------------------
hasReddit
       n  missing distinct     Info      Sum     Mean      Gmd
    2767        0        2    0.697     1751   0.6328   0.4649

---------------------------------------------------------------------------------------------------------------
platform
       n  missing distinct
    2761        6      129

lowest :  Ethereum  NEO      Acclaim   Achain    AION     , highest: XAYA      xDAC      YouToken  Zilliqa   Zuum
---------------------------------------------------------------------------------------------------------------
coinNum
       n   missing  distinct      Info      Mean       Gmd       .05       .10       .25       .50
    2767         0       855         1 8.178e+12 1.636e+13 5.378e+06 1.050e+07 5.000e+07 1.800e+08
     .75       .90       .95
6.000e+08 3.000e+09 7.000e+09

lowest : 1.200000e+01 2.000000e+01 4.500000e+01 6.000000e+01 8.000000e+01
highest: 2.500000e+11 5.500000e+11 1.000000e+12 1.200000e+12 2.261908e+16
---------------------------------------------------------------------------------------------------------------
minInvestment
       n  missing distinct     Info      Sum     Mean      Gmd
    2767        0        2    0.743     1254   0.4532   0.4958

---------------------------------------------------------------------------------------------------------------
distributedPercentage
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50      .75
    2767        0      111    0.995    1.061    1.277     0.14     0.25     0.40     0.55     0.70
     .90      .95
    0.80     0.85

lowest : 0      0.01   0.02   0.03   0.04  , highest: 60     62.5   100    266.25 869.75
---------------------------------------------------------------------------------------------------------------
```

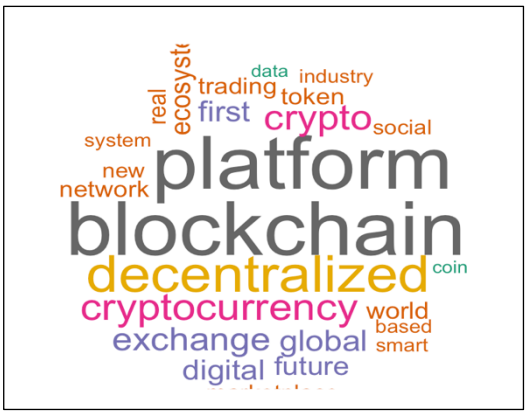
a) ***Success***



The goal of the task is to predict whether the team/company will reach the goal successfully. So success is very important as ICO is based on an “all or nothing” approach. Success is the predictor variable. As observed in the plot, there is an imbalance in the data, where it contains only 37.2% instances of success. This can lead to biased models as prediction would be on the majority class and would perform poorly in predicting the minority class, which is success. This needs appropriate handling.

b) ***BrandSlogan***

Brand slogan represents the slogan of the fundraising team. Below, shows the word cloud based on frequencies of the word, where frequency set to 25.



c) ***TeamSize***

Number of team members. There are 154 missing values. The distribution is right skewed, with the majority falling between 0 and 20. Median and mode are 12 and 13.10 respectively.


```
> median(wbcd$teamSize, na.rm = TRUE)
[1] 12
> mean(wbcd$teamSize, na.rm = TRUE)
[1] 13.10754
```

```
> sum(!complete.cases(wbcd$teamSize))
[1] 154
```

d) ***Ratings***

Ratings are given by investment experts ranging from 1(very poor) to 5 (very good). Data looks clean without missing values. The plot is normally distributed, with mean rating being 3.12.

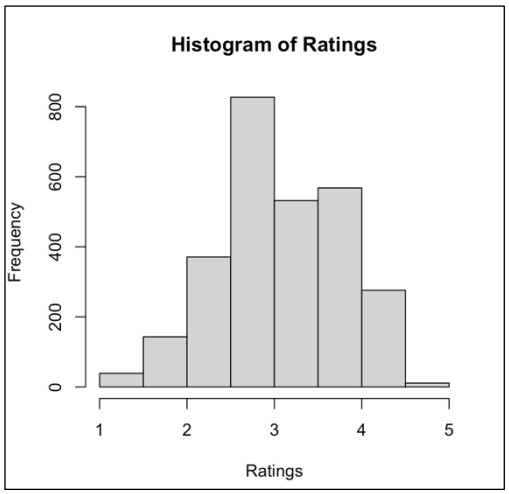


Figure 3.1.4 – Histogram of Ratings

```
> mean(wbcd$rating, na.rm = TRUE)
[1] 3.121323
```

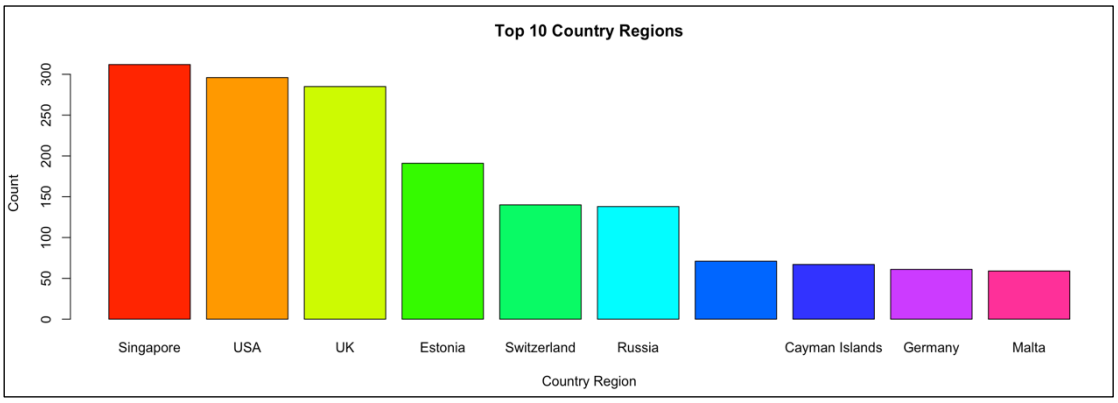
```
> sum(!complete.cases(wbcd$rating))
[1] 0
```

e) ***CountryRegion***

This is the country where the fund-raising team/company is located. There are 71 missing values. Plot shows the top 10 country regions in the dataset. There are some countries with data discrepancies, for example SINGAPORE and Singapore.

```
> sort(table(wbcd$countryRegion), decreasing = TRUE)[1:10]

     Singapore            USA             UK       Estonia    Switzerland         Russia
           312            296            285            191            140            138             71
Cayman Islands        Germany         Malta
            67             61             59
```

```
          Philippines                       Poland                     Portugal
                   14                           21                            7
           Puerto Rico                      Romania                       Russia
                    1                           24                          138
Saint Kitts and Nevis Saint Vincent and the Grenadines                    Samoa
                   11                            1                            1
         Saudi Arabia                       Serbia                    Seychelles
                    1                           10                           31
         Sierra Leone                    Singapore                     SINGAPORE
                    1                          312                            1
             Slovakia                     Slovenia                  South Africa
                    1                           24                           24
          South Korea                        Spain                        Sweden
                   30                           18                            7
          Switzerland                        Syria                      Tanzania
                  140                            1                            2
             Thailand                  Timor-Leste                       Tunisia
                   13                            1                            1
               Turkey                           UK                       Ukraine
                   16                          285                           19
United Arab Emirates                          usa                           USA
                   41                            1                          296
              Vanuatu                    Venezuela                      Vietnam
```

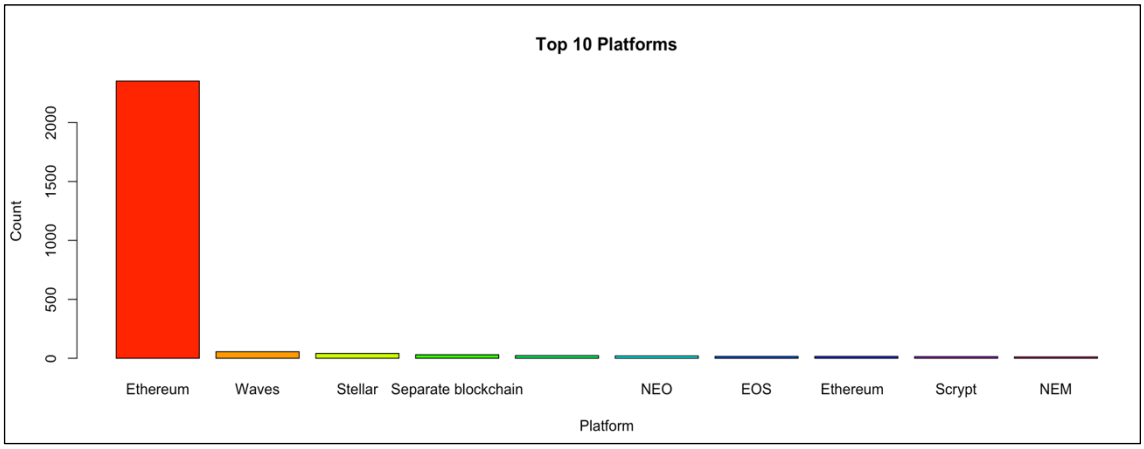


e) ***Platform***

This variable explains the blockchain platform on which the fund-raising project is based on. There are several platforms where Ethereum clearly gets the majority. As observed, there are incorrect spelling and with spaces of word Ethereum as Etherum and so on. Data cleaning to be performed.

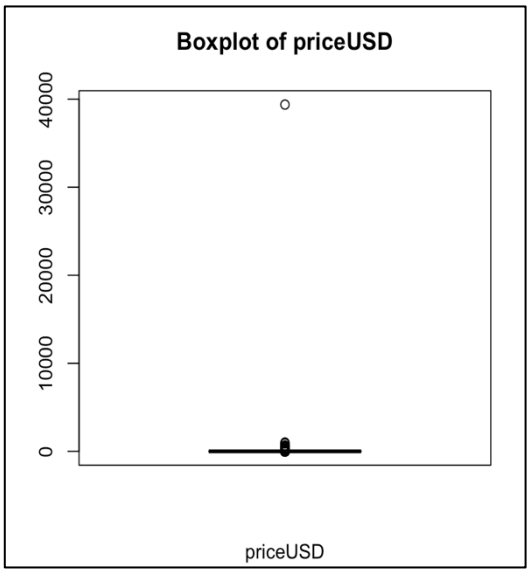
6	Ethereum 1	NEO 1	Acclaim 1
Achain 1	AION 1	Akroma 1	Apollo Blockchain 1
Ardor 1	BEP2 1	Bitcoin 10	BitForex 1

ERC20 2	ETH 1	Ethererum 2	Ethereum 2352
Ethereum 23	Ethereum 16	Ethereum 9	Ethereum 2
Ethereum 2	Ethereum, Waves 1	Etherum 1	Fiber 1



e) ***PriceUSD***

PriceUSD is the price for each blockchain coin issued, in US dollars. They have 180 missing values. Maximum value is 39384, and it's an outlier, and the project for the outlier was unsuccessful. PriceUSD is 0 for 151 cases, but it is kept as it is, as there are projects which are successes (53).




```
> sum(!complete.cases(wbcd$priceUSD))
[1] 180
```

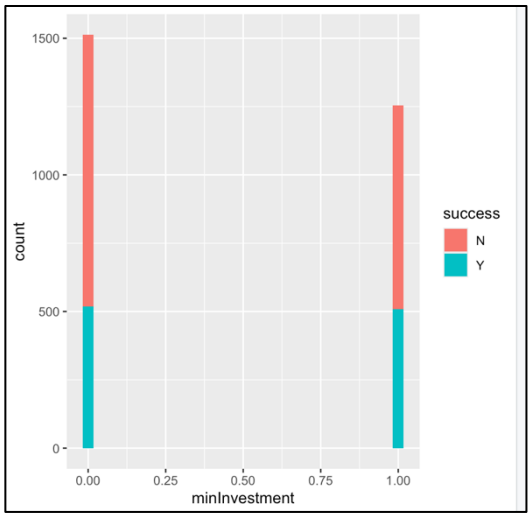
```
> nrow(subset(wbcd, priceUSD == 0))
[1] 151
>
```

```
> max(wbcd$priceUSD,na.rm = TRUE)
[1] 39384
```

```
> wbcd_usd <- subset(wbcd, priceUSD == 0)
> table(wbcd_usd$success,wbcd_usd$priceUSD)

      0
No   98
Yes  53
```

f) ***MinInvestment***

This is an indicator variable 1, if the fundraising project has been set up with minimal investment, say 10 USD, in their campaign page. If not, it's zero. The plot describes minInvestment and its involvement with success rate.

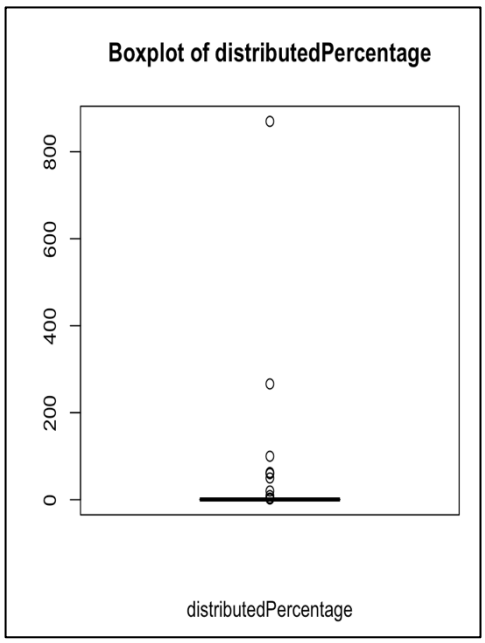



```
> sum(!complete.cases(wbcd$minInvestment))
[1] 0
```

```
     N   Y
0  994 519
1  745 509
```

f) ***Distributed Percentage***

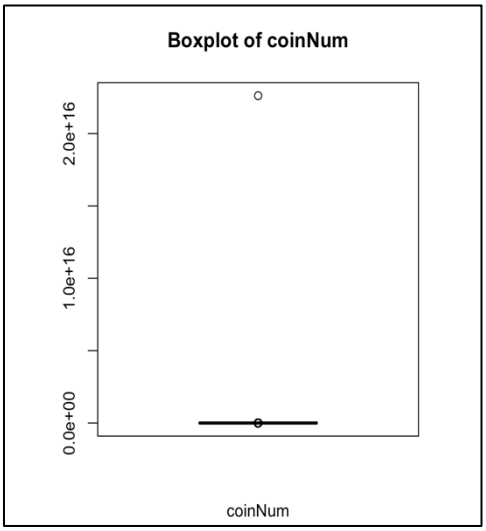
The percentage of blockchain coins distributed to investors relative to all the coins created. Retaining a large percentage of coins, signals their commitment to and engagement in the development of the project. As it is evident in the data, there are outliers and wrong values;10 values above 1.00.



distributedPercentage
1.66
4.00
9.52
62.50
266.25
869.75
20.00
100.00
50.00
60.00

g) ***coinNum***

The number of blockchain a fundraising team can freely decide to be issued. As seen in the plot, there is an outlier 22619078416760300. No missing values in the dataset for this variable.



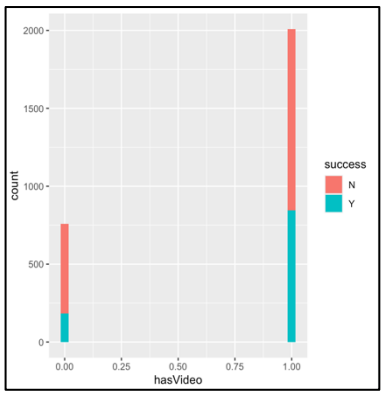

```
> sum(!complete.cases(wbcd$coinNum))
[1] 0
```

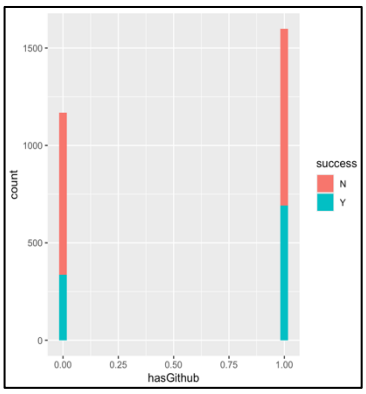
```
> max(wbcd$coinNum)
[1] 2.261908e+16
```

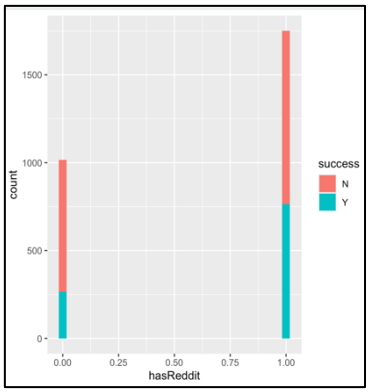
g) ***hasVideo, hasGithub, hasReddit***

These indicator variables are 1, if the fundraising project provided a video, official Github and official Reddit for community discussion on their campaign page. If not it is zero. There are no missing values for all three variables. Figure shows their relation with the predictor variable, success and they are considerably uniform to an extent.

```
> sum(!complete.cases(wbcd$hasVideo))
[1] 0
> sum(!complete.cases(wbcd$hasGithub))
[1] 0
> sum(!complete.cases(wbcd$hasReddit))
[1] 0
```



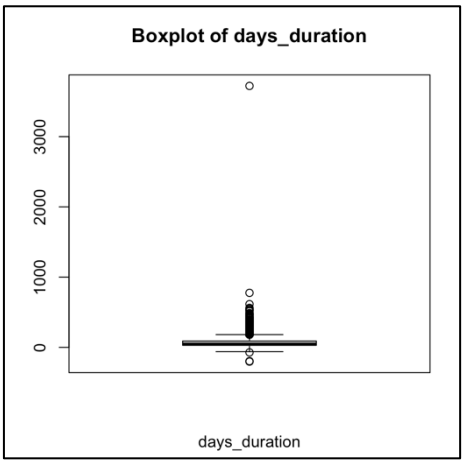




h) ***startDate, endDate***

This variable explains the start and end date of the fund raising campaign. For better analysis, **days duration** is calculated, that is the difference between these dates. However, it is observed that there are negative values and outliers as per figure below, stating that they are wrong values, outlier (max values 3722, which is over 10 years) or start and end dates are swapped giving duration to be negative values. On the contrary, there are no missing values.

```
> wbcd <- wbcd %>%
+   mutate(days_duration =
+            as.numeric(difftime(as.Date(wbcd$endDate,format = "%d/%m/%Y"),
+                                as.Date(wbcd$startDate, format = "%d/%m/%Y"), units = "days")))
>
```




```
> max(wbcd$days_duration)
[1] 3722
```

```
> sum(wbcd$days_duration < 0)
[1] 12
```

4.0 Data Preparation

This step involves all the necessary steps required to format the data and preparing them for further data modelling.

a) Data Cleaning

CountryRegion : Some countries have certain discrepancies, with spaces and variations in lower and upper cases. So to maintain consistency, they are cleaned to match the correct values.

Before	After
Curacao	Curaçao
SINGAPORE	Singapore
usa	USA
india	India

Platform : There are platform values corresponding to "Ethereum". So they are cleaned to make it uniform. 57 values were manually cleaned to achieve.

DaysDuration : There are 12 records with negative values. They are wrong data as duration is not expected to be negative, hence they are deleted. The dates were type casted to dates before duration calculation.

DistributedPercentage : All the values greater than 1.00% are cleaned. 10 values were removed

b) Removing outliers

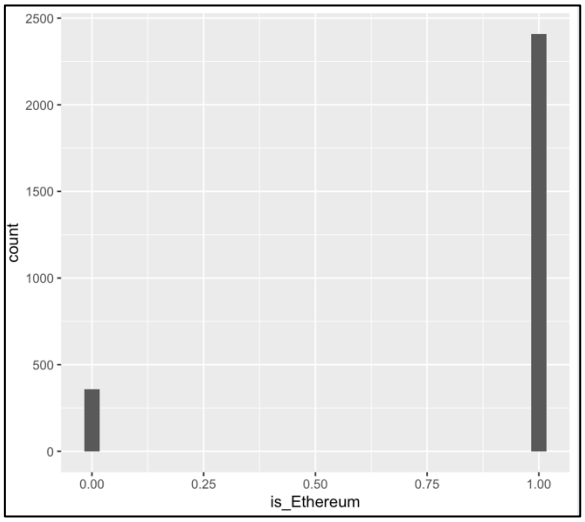
priceUSD: The value with 39384 is outlier and not part of the total proportion. This will be removed.

coinNum: There is a value 22619078416760300, which is an outlier. This will be rejected.

DaysDuration: 3722 days is an outlier, as it is over 10 years and the project was unsuccessful. It proves to be incorrect data, hence removed.

c) Creating New Variables

Platform: Ethereum dominates the dataset, influencing the predictor variable. In the encoding process a new variable Is_Ethereum is generated, when the platform is Ethereum then it is 1 and if not 0, as most of them fall in that platform.



CountryRegion: There are 71 countries with missing values. As study was conducted in Stanford (Kaal,2018), on top 25 countries based on the World ICOs, funds raised and the countries respective jurisdiction in regulating ICOs. So a variable is created, ***is_top_25***, if the country matches the top 25 it is set as 1 and if not 0, this eliminates the missing values as 0.

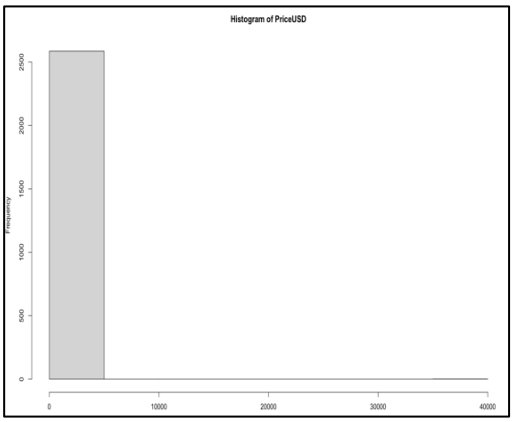
Capital_Raised: Research has been conducted on the impact of capital raised on project success (Wei, et al., 2022). Utilizing both coinNUM and priceUSD, one can readily assess the capital raised by multiplying these two variables.

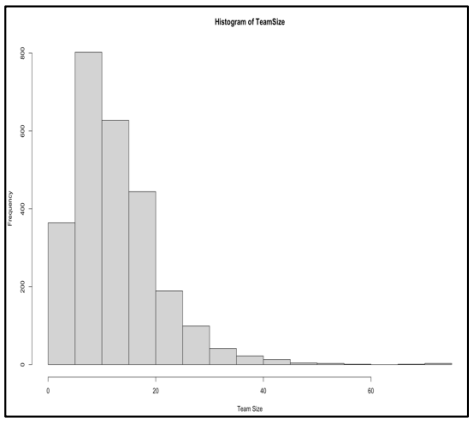
d) Data Imputation

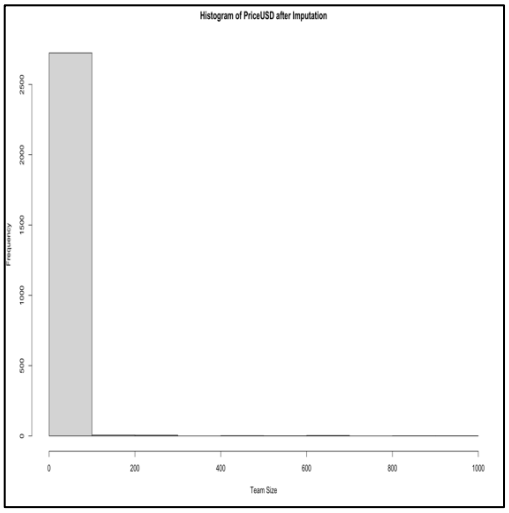
PriceUSD: 180 entries have been filled with the mean value of the entire priceUSD column following the removal of outliers, aiming to maintain the distribution intact.

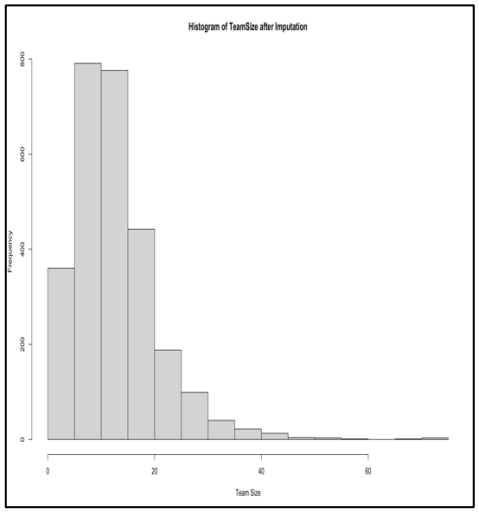
TeamSize: 154 values are missing, and they are imputed with the median of the TeamSize column so that it preserves the overall central tendency of the data.

Refer below, before and after imputation.





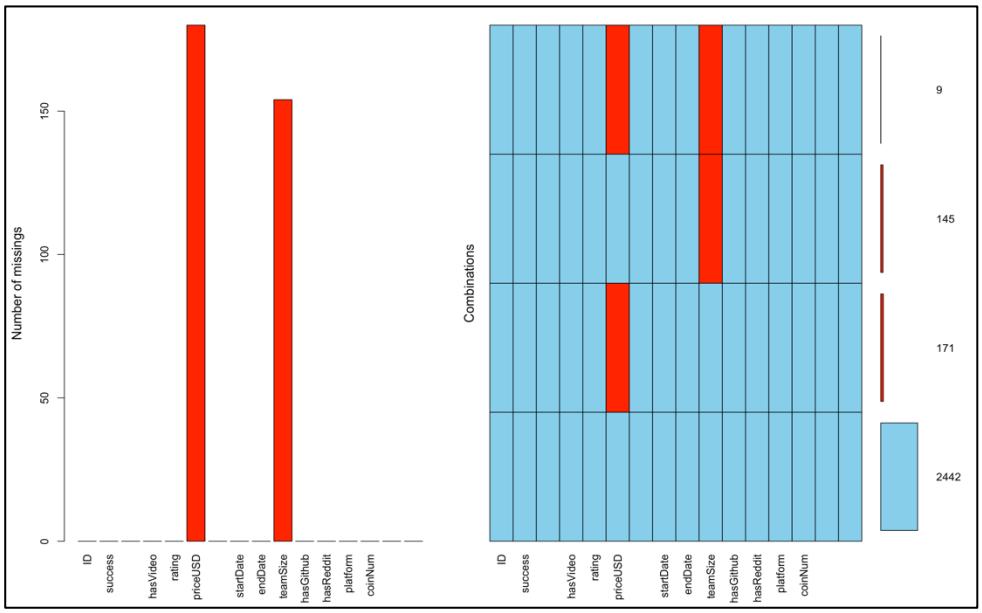




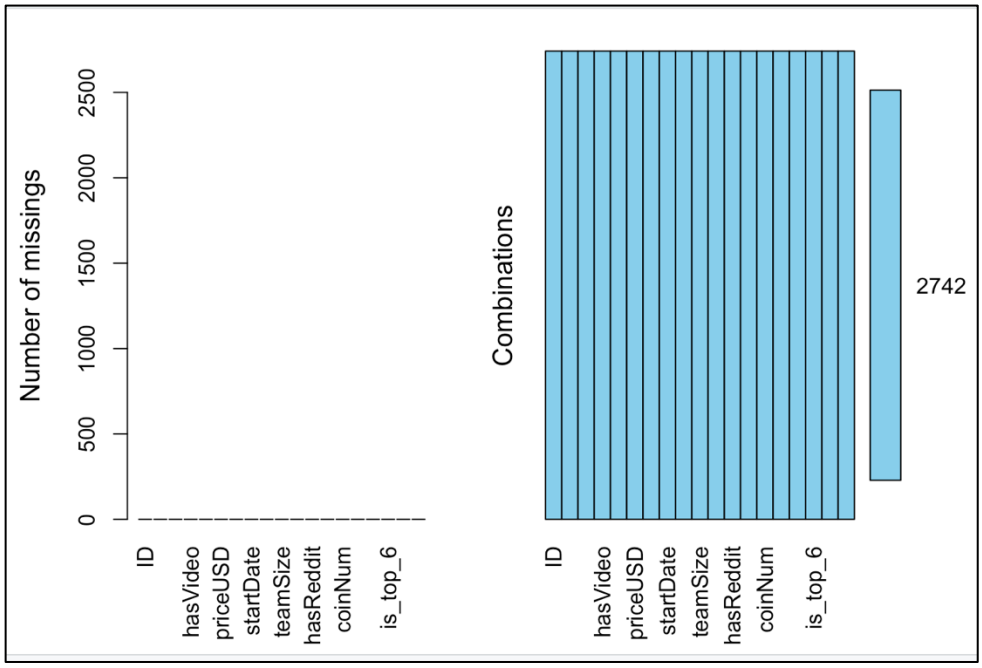
.

To summarise, we have conducted data cleaning, eliminating missing data, and enhancing the dataset through transformations to ensure its readiness for modelling purposes. Throughout these processes, only a minimal portion, comprising **0.9%** of the total data, required deletion.

BEFORE IMPUTATION - RAW DATA (2767)



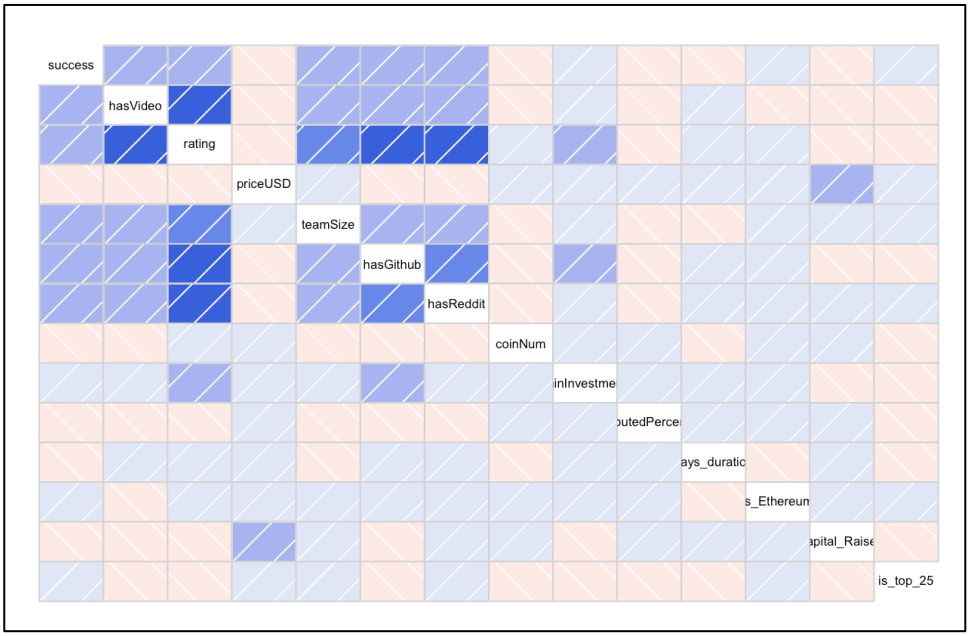
AFTER IMPUTATION – CLEANED DATA (2742)



4.1 FEATURE ENGINEERING

Feature engineering in machine learning models involves identifying and choosing the most relevant features from the data set to improve the model performance and accuracy. Based on the features, it can even overcome overfitting and improve computational efficiency.

4.1.1 CORRELATION BETWEEN FACTORS IN CLEANED DATA



Correlation matrix illustrates the relationship between the indicators and the predictor variable, success. It is evident that online campaigns, hasVideo, hasGithub, and hasReddit exhibit a positive correlation with success. Additionally, rating, teamSize, minInvestment, Is_Ethereum, and is_top25 also display positive correlations (highlighted in blue). The remaining indicators do not demonstrate any correlation.

4.1.2 CUMULATIVE FEATURE SELECTION

The following features are selected to build the model and predict the outcome.

- Ethereum blockchain has demonstrated a significant factor (Fenu,et.al,2018). ***is_Ethereum,*** is thus included in the model and it also shows positive correlation in the correlation matrix as well.
- ICO expert ratings (Shrestha,2022), can influence project success. Thus, the ***rating*** variable is incorporated into the model.
- Online campaigns, including ***hasVideo***, ***hasGitHub***, and ***hasReddit***, are known to have a meaningful impact on project success. These variables exhibit positive correlations in the correlation matrix.
- Capital raised, represented by the variable ***capital_raised*** as proposed (Šapkauskienė,et.al, 2020), is another influential feature attracting investors and contributing to project success.
- ***teamSize*** can be relevant in predicting the success. It is studied that team size and duration, ***days_duration*** are related to funding successes (Roosenboom,et.al, 2020).

- ***brandSlogan,*** is removed from the feature for modelling to avoid potential biases.
- ***minInvestment***, is another feature deemed significant for the model and has correlation with the predictor, success.
- ***distributedPercentage*** is used as a feature in the model. Though it shows no correlation, we shall still explore its significance in models.
- Top 25, ICO friendly countries (Kaal,2018) that have been influential in World ICOs, funds raised and the countries' respective jurisdiction in regulating ICOs is considered. Assumed to have an impact on the success of the goal, also has shown a possible correlation with success.

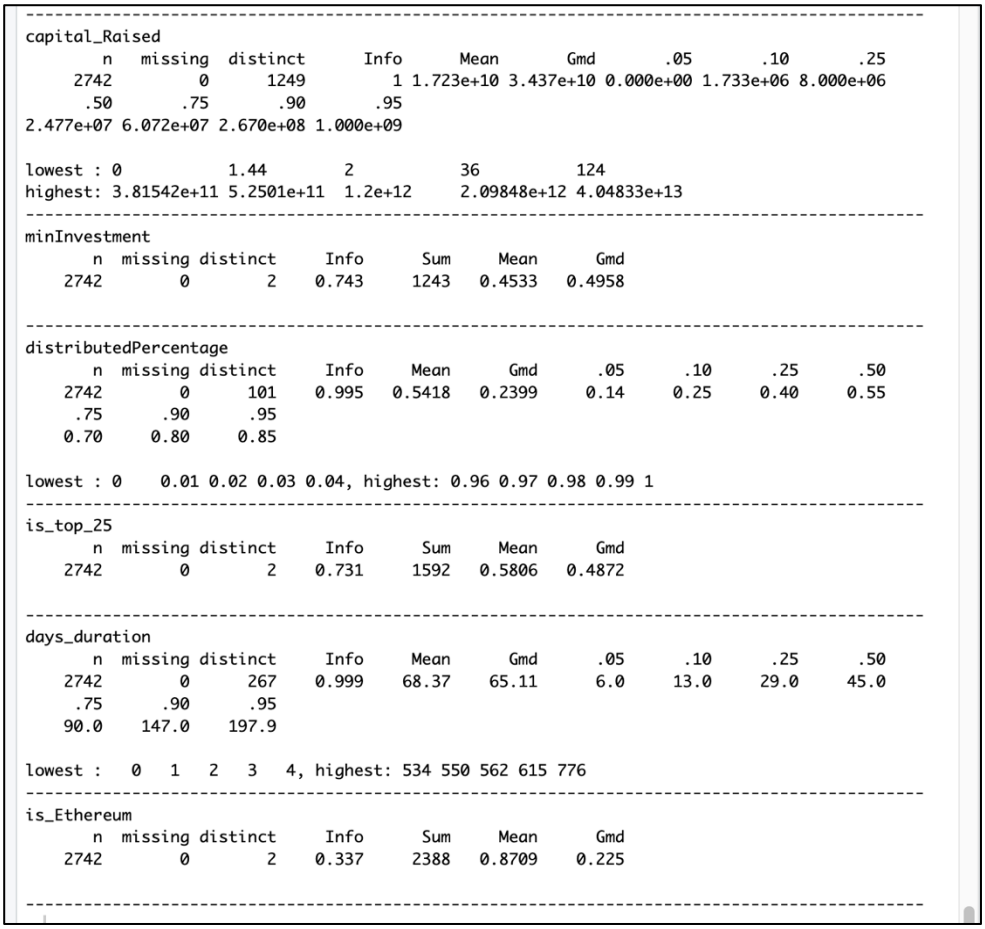
4.1.3 FINAL DATA

Based on the features selected from the above step, we conclude here by selecting 12 variables in the final dataset of 2742 observations. Summary of the selected features from the final data and their insights are shown below.

```
> describe(wbcd_model)
wbcd_model

 12  Variables      2742  Observations
--------------------------------------------------------------------------------
success
       n  missing distinct
    2742        0        2

Value         No   Yes
Frequency   1722  1020
Proportion 0.628 0.372
--------------------------------------------------------------------------------
hasVideo
       n  missing distinct     Info      Sum     Mean      Gmd
    2742        0        2    0.597     1990   0.7257   0.3982

--------------------------------------------------------------------------------
rating
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50
    2742        0       39    0.998    3.119   0.8161      2.0      2.2      2.6      3.1
     .75      .90      .95
     3.7      4.1      4.2

lowest : 1   1.1 1.2 1.3 1.4, highest: 4.4 4.5 4.6 4.7 4.8
--------------------------------------------------------------------------------
teamSize
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50
    2742        0       53    0.997    13.07    8.172        4        5        8       12
     .75      .90      .95
      17       23       27

lowest :  1  2  3  4  5, highest: 57 67 72 73 75
--------------------------------------------------------------------------------
hasGithub
       n  missing distinct     Info      Sum     Mean      Gmd
    2742        0        2    0.732     1582    0.577   0.4883

--------------------------------------------------------------------------------
hasReddit
       n  missing distinct     Info      Sum     Mean      Gmd
    2742        0        2    0.698     1733    0.632   0.4653

--------------------------------------------------------------------------------
```

5.0 MODELLING

The purpose of the task is to predict whether a company/team will successfully reach its fundraising goal through an Initial Coin Offering, and data modelling involves the process of selecting appropriate machine learning algorithms to make accurate predictions based on the data, namely supervised machine learning algorithms. These algorithms are trained on a dataset, where the input data, the features selected for the model are associated with corresponding target (predictor variable), thereby making predictions on test data. The available data is divided into training and testing sets with 80% of them, allocated for training and remaining 20% for testing. This segregation ensures that the model is trained on larger portions of data while having unseen data for evaluating its performance. In this scenario, the data set is imbalanced, where there are more instances of failures than the other class, success, which is considered accordingly in the evaluation of the models.



```
---------------------------------------------------------------------------------------------
capital_Raised
       n   missing  distinct      Info      Mean       Gmd       .05       .10       .25
    2742         0      1249         1 1.723e+10 3.437e+10 0.000e+00 1.733e+06 8.000e+06
     .50       .75       .90       .95
2.477e+07 6.072e+07 2.670e+08 1.000e+09

lowest : 0           1.44        2           36          124
highest: 3.81542e+11 5.2501e+11  1.2e+12     2.09848e+12 4.04833e+13
---------------------------------------------------------------------------------------------
minInvestment
       n  missing distinct     Info      Sum     Mean      Gmd
    2742        0        2    0.743     1243   0.4533   0.4958

---------------------------------------------------------------------------------------------
distributedPercentage
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50
    2742        0      101    0.995   0.5418   0.2399     0.14     0.25     0.40     0.55
     .75      .90      .95
    0.70     0.80     0.85

lowest : 0    0.01 0.02 0.03 0.04, highest: 0.96 0.97 0.98 0.99 1
---------------------------------------------------------------------------------------------
is_top_25
       n  missing distinct     Info      Sum     Mean      Gmd
    2742        0        2    0.731     1592   0.5806   0.4872

---------------------------------------------------------------------------------------------
days_duration
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50
    2742        0      267    0.999    68.37    65.11      6.0     13.0     29.0     45.0
     .75      .90      .95
    90.0    147.0    197.9

lowest :   0   1   2   3   4, highest: 534 550 562 615 776
---------------------------------------------------------------------------------------------
is_Ethereum
       n  missing distinct     Info      Sum     Mean      Gmd
    2742        0        2    0.337     2388   0.8709    0.225

---------------------------------------------------------------------------------------------
```

5.1 Classification Models

Below are the proposed models that can be employed for this classification problem. and they are described as below.

a) **Random forest**

This classification model is an ensemble learning method that builds multiple decision trees and merges their predictions. This method is useful when the data is imbalanced and averages the predictions from multiple trees, thereby reducing overfitting to the majority class. A function, **randomForest(),** is used to create the model and the parameters where **ntree = 500,** is set, which is the number of trees and **mtry = 4**, that considers 4 variables to randomly consider from the data set in each split.

b) Support Vector Machines

SVMs work to find optimal hyperplanes that separate different classes. The model aims to maximise the margin, which is the distance between the hyperplane and nearest data points (support vectors) from each class. By choosing the right kernel, *kernel trick*, we can capture complex relationships between features. The function, **ksvm(),** that is kernel support vector machines, is used to build the model. Models were evaluated using different kernel tricks, taking the input data from linear to a higher dimensional space to find its non-linear boundaries. We employ it using ***vanilladot, rbfdot, polydot and tanhdot*** and their metrics are evaluated.

c) Decision Trees

Decision Trees, an algorithm where they work repeatedly partitioning the features into subsets. It is based on 'divide and conquer' and used where the results need to be interpretable and transparent. Division is based on the entropy feature with highest information gain and will be split up first. The function, ***C5.0*** is used to build the decision tree model.

d) Adaptive Boosting

This model, an ensemble learning technique, combines the prediction from weak classifiers and builds strong classifiers. It performs well for imbalanced data sets. Here we perform AdaBoost technique on decision trees to improve the performance on the prediction of minority class. We use the same function, ***C5.0,*** with **trials = 10**, that specifies the number of iterations for boosting.

e) **K-NN**

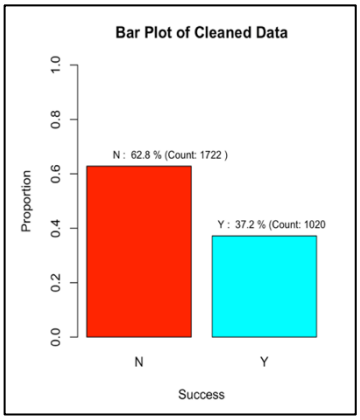
These algorithms are called “lazy learners”. They make predictions based on k data points from training data, use their values, to find value of the input data. K-NN is based on Euclidean distance; the measure the similarity or distance between two data points. It uses **knn()** function to build the model. K is selected as ***K=45*** , an odd number to avoid ties and ensure that there is always a majority class. K fold (10-fold) cross validation was done using **trainControl()** function in **caret** package to find out the optimal value of K that maximises the performance of the model on the training data and that was ***K=65.***

f) **Logistic Regression**

A base-line supervised machine learning model, for classification problems. Unlike linear regression, logistic regressions models a relationship between independent variables and probability of an outcome. A function, **glm()** is used for developing the model.

6.0 EVALUATION

The classification models were tested on cleaned-up data , as two partitions, as discussed previously. Dataset is imbalanced, with the "success" class being the minority, which is considered the positive class. All models were trained with "success" representing the positive outcome, and their performance was evaluated accordingly.
Normally accuracy is considered to be a key metric for evaluation, In this scenario we have only 37.2% of positive classes. Therefore the models are expected to exhibit better **precision** – metric that measure the proportion of true positive predictions among all positive predictions and **sensitivity/recall**, the ability to correctly identify positive instances, or the metric that measures proportion of actual positive cases that are correctly identified by the classifier, as the purpose of the assignment is to predict the success of the fund raising campaign.



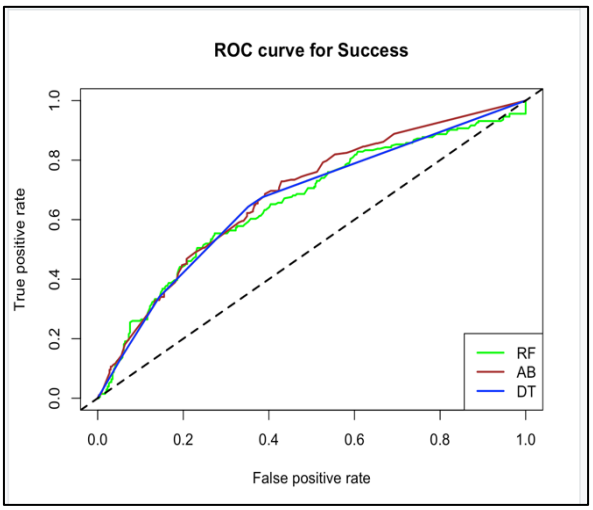
The best model is the one that balances between correctly identifying positive instances (sensitivity/recall) and minimizing false positives. Considering this criterion, the **Random Forest model** emerges as the most suitable choice. Despite slightly lower sensitivity/recall compared to some other models, its notably high specificity of 80.81% indicates its ability to accurately identify negative instances, which is crucial in an imbalanced scenario where the majority class dominates. Moreover, the Random Forest model achieves a reasonable accuracy of 67.15% and a balanced F1 score of 50.00%, reflecting its effectiveness in handling the class imbalance while maintaining overall classification performance. Therefore, in this scenario, the Random Forest model is the preferred choice for its ability to strike a balance between sensitivity and specificity, making it well-suited for imbalanced datasets with a majority class in the negative category. On the contrary, after performing the K-Fold cross validation, AdaBoost has the highest sensitivity/recall among the remaining models It effectively identifies positive instances, crucial in an imbalanced dataset. AdaBoost also maintains a relatively high specificity of 80.04%, indicating its ability to correctly identify negative instances. There is logistic regression above Adaboost, but the model was built based on the 0.5% threshold, so that predictions may not always be true and would possibly require threshold adjustment.

Model	Accuracy	Sensitivity	Specificity	Precision	F1	AUC
Random Forest	0.6715	0.4412	0.8081	0.5769	0.5000	0.6579
SVM - vanilladot	0.6557	0.6625	0.6545	0.2465	0.3593	0.6905
SVM - rbfdot	0.6503	0.6055	0.6614	0.3070	0.4074	0.6905
SVM - polydot	0.6557	0.6625	0.6545	0.2465	0.3593	0.6905
SVM - tanhdot	0.5519	0.4258	0.6294	0.4140	0.4198	0.5619
Decision Trees	0.6794	0.3404	0.8560	0.5517	0.4211	0.6642
AdaBoost	0.6776	0.3564	0.8449	0.5447	0.4309	0.6836
K-NN, K=45	0.6576	0.2872	0.8504	0.5000	0.3648	0.6411
K-NN, K= 65	0.6648	0.2659	0.8725	0.5208	0.3521	0.6508
Logistic Regression, cut off :0.5	0.6515	0.5870	0.6732	0.5870	0.4589	0.7021
Logistic Regression, cut off :0.2	0.4799	0.4277	0.8750	0.9628	0.4277	0.7021

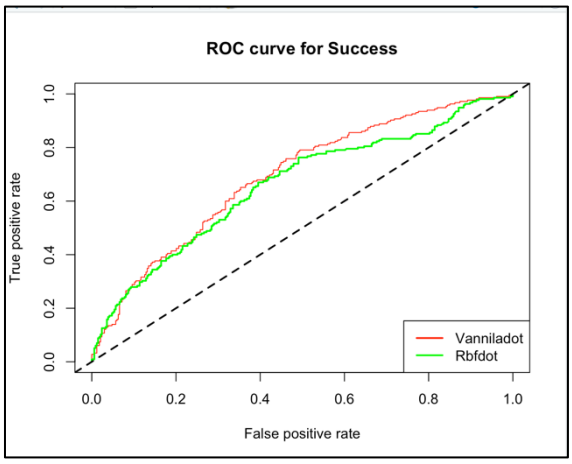
Additionally, **K-Fold Cross Validation** was conducted on the entire dataset, using **K=10** folds to assess performance and predictability. The table below presents the metrics for each model following validation.

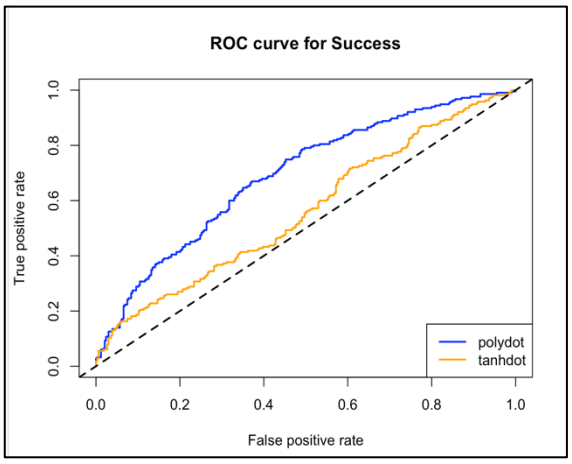
Model	Accuracy	Sensitivity	Specificity	Precision	F1
Random Forest	0.6481	0.3846	0.8042	0.5370	0.4470
Decision Trees	0.6265	0.4578	0.7476	0.5630	0.4985
AdaBoost	0.6525	0.4394	0.8004	0.6064	0.5087
K-NN, K=65	0.6509	0.2784	0.8716	0.5636	0.3708
SVM - vanilladot	0.5844	0.325	0.7338	0.4126	0.3636
SVM - rbfdot	0.5709	0.2745	0.7428	0.3817	0.3183
SVM - polydot	0.5558	0.3079	0.6995	0.3725	0.3364
SVM - tanhdot	0.5198	0.4112	0.5828	0.3623	0.3845
Logistic Regression	0.6675	0.6739	0.6697	0.2783	0.3059

The AUC values provided for the classification models serve as a performance metric, with values exceeding 0.5 indicates good model performance. The ROC curves displayed for the Random Forest, Adaptive Boosting, and Decision Tree models illustrate AUC values ranging from 0.65 to 0.68, thereby showcasing their ability to effectively differentiate between positive and negative classes.

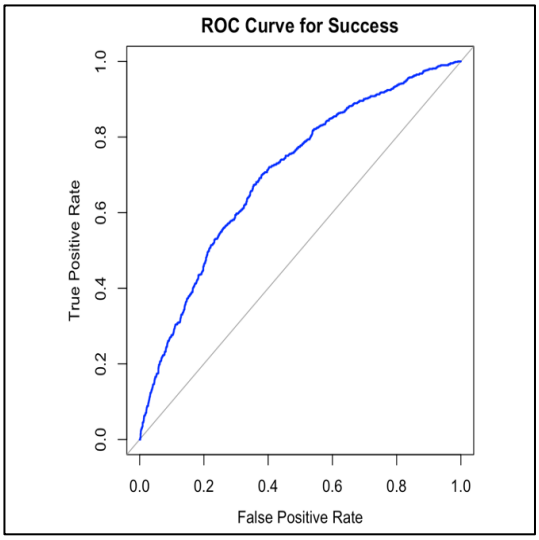


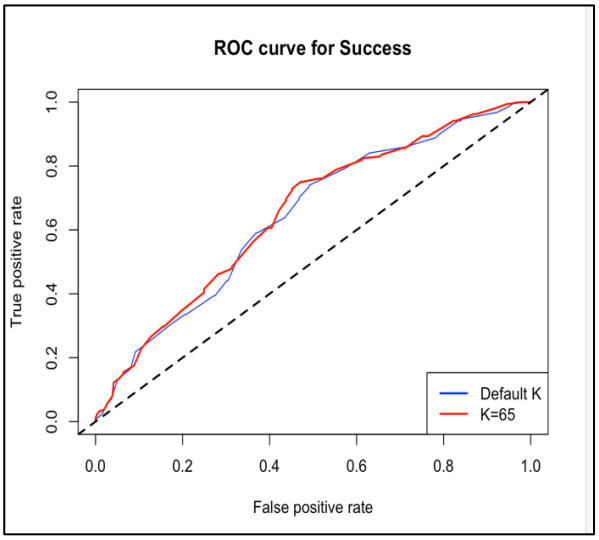
These are ROC curves of SVM models, with their kernel tricks applied. Except for ***tanhot*** other SVM kernels indicates good performance of 0.69.





The Roc curve for Logistic Regression is as below. It indicated a better performance of 0.70 and can be considered as a better classifier, but may not be highly accurate in predicting the positive class as precision was 0.58. But when the threshold was changed to 0.2 from 0.5, the values reached up to 0.96. On the other hand, KNN had 0.64 and 0.65 respectively.





Random forest model shows the importance of maintaining high-quality ratings, businesses or projects with larger teams, amount of funding raised, a project duration of a project or venture as strong predictors to success of the ICO campaigns.

```
> importance(rf_model)
                         MeanDecreaseGini
hasVideo                         21.88669
rating                          167.54333
teamSize                        166.88577
hasGithub                        24.70591
hasReddit                        24.91515
capital_Raised                  192.81074
minInvestment                    27.87714
distributedPercentage           152.15936
is_top_25                        28.12051
days_duration                   192.59670
is_Ethereum                      17.88721
```

In logistic regression too, days duration, size of the team and expert ratings have significant impact on determining the success as well.

```
Call:
glm(formula = success ~ hasVideo + rating + teamSize + hasGithub +
    hasReddit + capital_Raised + minInvestment + distributedPercentage +
    is_top_25 + days_duration + is_Ethereum, family = binomial,
    data = LR_train_data)

Coefficients:
                      Estimate Std. Error z value Pr(>|z|)
(Intercept)          -3.322e+00  3.051e-01 -10.889  < 2e-16 ***
hasVideo              1.910e-01  1.232e-01   1.550 0.121061
rating                6.213e-01  8.950e-02   6.942 3.87e-12 ***
teamSize              3.605e-02  6.610e-03   5.453 4.94e-08 ***
hasGithub             3.842e-02  1.088e-01   0.353 0.723885
hasReddit             2.057e-01  1.142e-01   1.801 0.071627 .
capital_Raised       -8.208e-12  7.632e-12  -1.075 0.282181
minInvestment         8.867e-02  9.712e-02   0.913 0.361227
distributedPercentage -4.071e-02  2.238e-01  -0.182 0.855682
is_top_25             1.677e-01  9.560e-02   1.754 0.079395 .
days_duration        -2.699e-03  7.543e-04  -3.578 0.000347 ***
is_Ethereum           1.146e-01  1.420e-01   0.807 0.419539
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2884.2  on 2193  degrees of freedom
Residual deviance: 2626.2  on 2182  degrees of freedom
AIC: 2650.2

Number of Fisher Scoring iterations: 10
```

6.1 LIMITATIONS

- More data with success would have facilitated the development of an efficient model.
- Inclusion of brand slogan using sentiment analysis or imputation of median to teamSize and mean to priceUSD could have impacted the accuracy of the models.
- Determining optimal K value for the K fold and performing cross validation on the models would have improved the performance of the models. Currently, cross validation was done after setting k=10.

7.0 RECOMMENDATIONS (DEPLOYMENT)

From the analysis, **Random Forest** can effectively identify the most influential features for ICO success, such as project characteristics, market sentiment, team experience, and bitcoins. It has feature importance, to understand significant features that can influence the prediction. It can handle non-linear relationships between features and fundraising outcomes. The model combines multiple decision trees to make predictions, resulting in a more robust and accurate model making it scalable and computationally efficient, for analysing large datasets commonly encountered in ICOs. The model could be improved after adding more significant variables that could potentially have an impact on the success. Its ability to handle high-dimensional data and parallel processing capabilities accelerates model training and prediction, enabling timely decision-making for fundraising strategies. Teams can understand how the model makes predictions, validate its reasoning and can communicate the results effectively to stakeholders and investors. ICO teams can make informed decisions, optimize fundraising strategies, and increase the likelihood of achieving their fundraising goals, by leveraging the model. Investors from grass root level or corporates can make use of the model to predict the success of their ICO campaigns.

REFERENCES

Fenu, G., Marchesi, L., Marchesi, M. and Tonelli, R., 2018, March. The ICO phenomenon and its relationships with ethereum smart contract environment. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)* (pp. 26-32). IEEE.

Roosenboom, P., van der Kolk, T. and de Jong, A., 2020. What determines success in initial coin offerings?. *Venture Capital*, *22*(2), pp.161-183.

Fisch, C., 2019. Initial coin offerings (ICOs) to finance new ventures. *Journal of Business Venturing*, *34*(1), pp.1-22.

Huang, W., Vismara, S. and Wei, X., 2022. Confidence and capital raising. *Journal of Corporate Finance*, *77*, p.101900.

Kaal, W.A., 2018. Initial coin offerings: The top 25 jurisdictions and their comparative regulatory responses (as of May 2018). *Stan. J. Blockchain L. & Pol'y*, *1*, p.41.

Shrestha, G., 2022. *Factors Affecting Success of Initial Coin Offerings (ICOs)* (Master's thesis).

Šapkauskienė, A. and Višinskaitė, I., 2020. Initial Coin Offerings (ICOs): benefits, risks and success measures. *Entrepreneurship and sustainability issues.*, *7*(3), pp.1472-1483.

APPENDIX

Random Forest

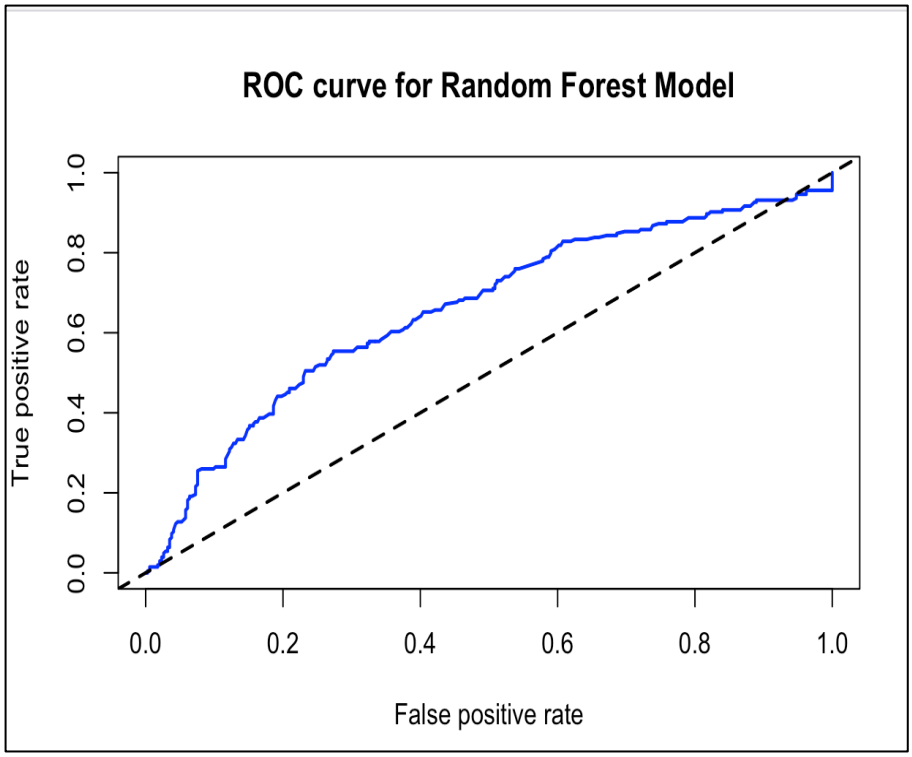
Cross Table and Confusion Matrix

Below are the cross table and confusion matrix for the random forest model. Also providing the evaluation metrices achieved by using this model.

```
> CrossTable(x = RF_test_data_labels, y = rf_predict, prop.chisq=FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  548


                    | rf_predict
RF_test_data_labels |         N |         Y | Row Total |
--------------------|-----------|-----------|-----------|
                  N |       278 |        66 |       344 |
                    |     0.808 |     0.192 |     0.628 |
                    |     0.709 |     0.423 |           |
                    |     0.507 |     0.120 |           |
--------------------|-----------|-----------|-----------|
                  Y |       114 |        90 |       204 |
                    |     0.559 |     0.441 |     0.372 |
                    |     0.291 |     0.577 |           |
                    |     0.208 |     0.164 |           |
--------------------|-----------|-----------|-----------|
       Column Total |       392 |       156 |       548 |
                    |     0.715 |     0.285 |           |
--------------------|-----------|-----------|-----------|
```

```
> confusionMatrix_RF
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 278 114
         Y  66  90

               Accuracy : 0.6715
                 95% CI : (0.6305, 0.7108)
    No Information Rate : 0.6277
    P-Value [Acc > NIR] : 0.0182857

                  Kappa : 0.2619

 Mcnemar's Test P-Value : 0.0004598

            Sensitivity : 0.4412
            Specificity : 0.8081
         Pos Pred Value : 0.5769
         Neg Pred Value : 0.7092
              Precision : 0.5769
                 Recall : 0.4412
                     F1 : 0.5000
             Prevalence : 0.3723
         Detection Rate : 0.1642
   Detection Prevalence : 0.2847
      Balanced Accuracy : 0.6247

       'Positive' Class : Y
```

ROC Curve



AUC Value

```
A performance instance
   'Area under the ROC curve'
> auc_object_RF@y.values[[1]]
[1] 0.6579956
```

K-fold Cross Verification on Random Forest

```
> avg_accuracy
[1] 0.6481892
> avg_sensitivity
[1] 0.3846405
> avg_specificity
[1] 0.8042969
> avg_precision
[1] 0.537094
> avg_recall
[1] 0.3846405
> avg_f1
[1] 0.4470931
```

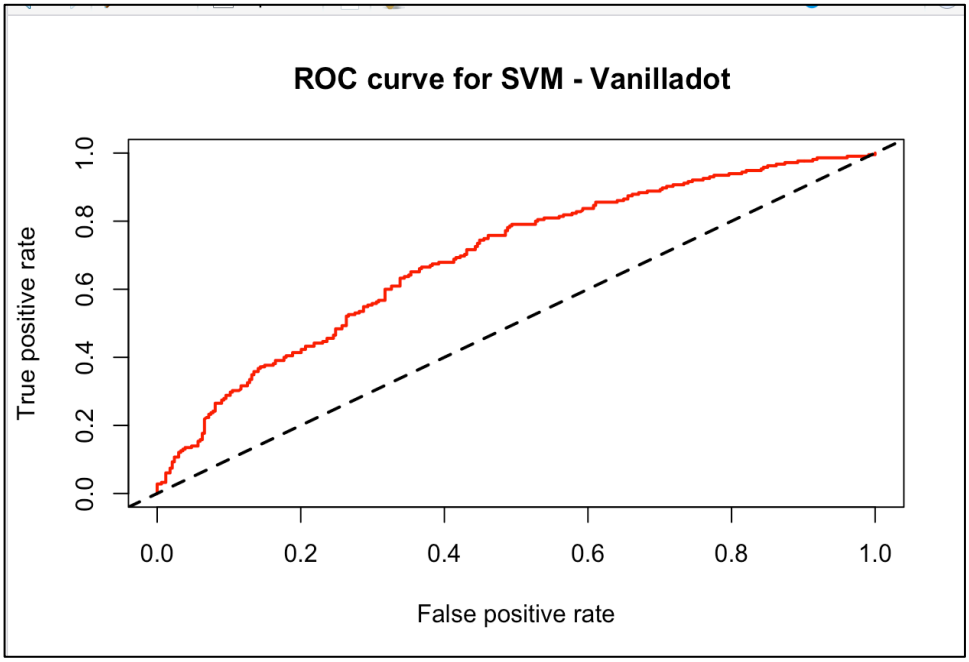
Support Vector Machines

Below are the cross table and confusion matrix for the SVM, linear kernal or ***vanillabot***

Also providing the evaluation metrices achieved by using this model.

```
> CrossTable(svm_results$actual_type, svm_results$predict_type,
+            prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  549


                         | svm_results$predict_type
svm_results$actual_type |         N |         Y | Row Total |
------------------------|-----------|-----------|-----------|
                      N |       307 |        27 |       334 |
                        |     0.655 |     0.338 |           |
------------------------|-----------|-----------|-----------|
                      Y |       162 |        53 |       215 |
                        |     0.345 |     0.662 |           |
------------------------|-----------|-----------|-----------|
           Column Total |       469 |        80 |       549 |
                        |     0.854 |     0.146 |           |
------------------------|-----------|-----------|-----------|
```

```
> confusionMatrix(svm_results$actual_type, svm_results$predict_type, positive = "Y", mode ="everything")
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 307  27
         Y 162  53

               Accuracy : 0.6557
                 95% CI : (0.6143, 0.6955)
    No Information Rate : 0.8543
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1865

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.66250
            Specificity : 0.65458
         Pos Pred Value : 0.24651
         Neg Pred Value : 0.91916
              Precision : 0.24651
                 Recall : 0.66250
                     F1 : 0.35932
             Prevalence : 0.14572
         Detection Rate : 0.09654
   Detection Prevalence : 0.39162
      Balanced Accuracy : 0.65854

       'Positive' Class : Y
```



AUC Curve

```
> auc_SVM
[1] 0.6905236
```

```
agreement
    FALSE      TRUE
0.3442623 0.6557377
```

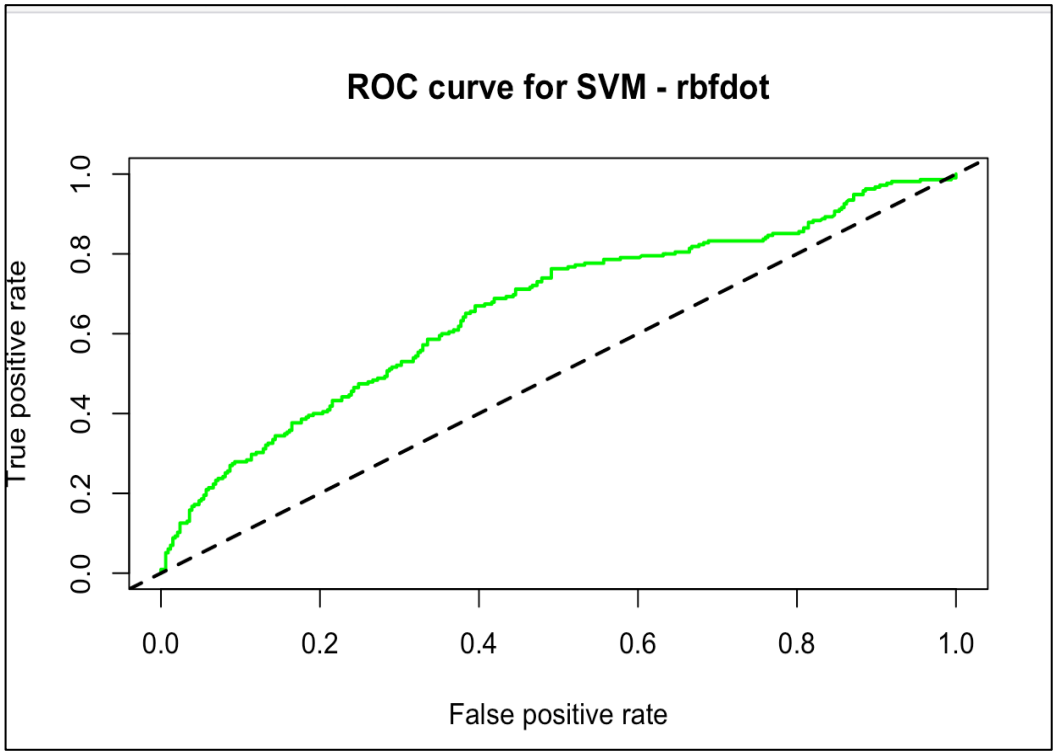
Cross Table and Confusion Matrix

Below are the cross table and confusion matrix for the SVM, ***rbfdot***

Also providing the evaluation metrices achieved by using this model.

```
> CrossTable(svm_results_rbf$actual_type, svm_results_rbf$predict_type, dnn=c('Y', 'N'),
+            prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  549


             | N
           Y |         N |         Y | Row Total |
-------------|-----------|-----------|-----------|
           N |       291 |        43 |       334 |
             |     0.661 |     0.394 |           |
-------------|-----------|-----------|-----------|
           Y |       149 |        66 |       215 |
             |     0.339 |     0.606 |           |
-------------|-----------|-----------|-----------|
Column Total |       440 |       109 |       549 |
             |     0.801 |     0.199 |           |
-------------|-----------|-----------|-----------|
```

```
> confusionMatrix(svm_results_rbf$actual_type, svm_results_rbf$predict_type, positive = "Y")
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 291  43
         Y 149  66

               Accuracy : 0.6503
                 95% CI : (0.6087, 0.6902)
    No Information Rate : 0.8015
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1954

 Mcnemar's Test P-Value : 3.517e-14

            Sensitivity : 0.6055
            Specificity : 0.6614
         Pos Pred Value : 0.3070
         Neg Pred Value : 0.8713
             Prevalence : 0.1985
         Detection Rate : 0.1202
   Detection Prevalence : 0.3916
      Balanced Accuracy : 0.6334

       'Positive' Class : Y
```




```
> auc_SVM_rbf
[1] 0.6905236
>
```

```
> table(agreement_rbf)
agreement_rbf
FALSE  TRUE
  192   357
> prop.table(table(agreement_rbf))
agreement_rbf
    FALSE      TRUE
0.3497268 0.6502732
```

K-Fold rbfdot

```
> avg_accuracy
[1] 0.5709073
> avg_sensitivity
[1] 0.274537
> avg_specificity
[1] 0.742811
> avg_precision
[1] 0.3817582
> avg_recall
[1] 0.274537
> avg_f1
[1] 0.3183715
>
```

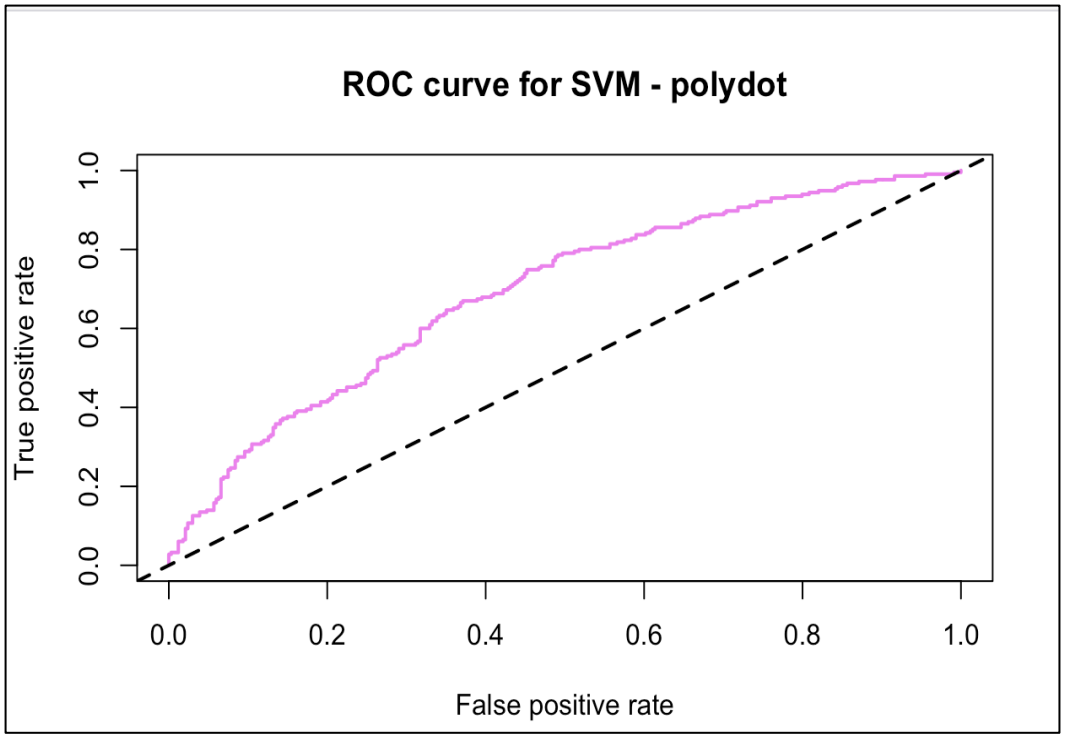
Cross Table and Confusion Matrix

Below are the cross table and confusion matrix for the SVM, ***polydot***
Also providing the evaluation metrices achieved by using this model.

```
   Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  549


             | N
           Y |         N |         Y | Row Total |
-------------|-----------|-----------|-----------|
           N |       307 |        27 |       334 |
             |     0.655 |     0.338 |           |
-------------|-----------|-----------|-----------|
           Y |       162 |        53 |       215 |
             |     0.345 |     0.662 |           |
-------------|-----------|-----------|-----------|
Column Total |       469 |        80 |       549 |
             |     0.854 |     0.146 |           |
-------------|-----------|-----------|-----------|
```

```
> confusionMatrix(svm_results_pd$actual_type, svm_results_pd$predict_type, positive = "Y",mode = "everythin
g")
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 307  27
         Y 162  53

               Accuracy : 0.6557
                 95% CI : (0.6143, 0.6955)
    No Information Rate : 0.8543
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1865

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.66250
            Specificity : 0.65458
         Pos Pred Value : 0.24651
         Neg Pred Value : 0.91916
              Precision : 0.24651
                 Recall : 0.66250
                     F1 : 0.35932
             Prevalence : 0.14572
         Detection Rate : 0.09654
   Detection Prevalence : 0.39162
      Balanced Accuracy : 0.65854

       'Positive' Class : Y
```

ROC Curve



AUC Value

```
> auc_SVM_pd
[1] 0.6905027
>
```

```
agreement_pd
    FALSE      TRUE
0.3442623 0.6557377
>
```

K-FOLD polyhot

```
> avg_accuracy <- mean(accuracy_SVM_pd)
> avg_sensitivity <- mean(sensitivity_SVM_pd)
> avg_specificity <- mean(specificity_SVM_pd)
> avg_precision <- mean(precision_SVM_pd)
> avg_recall <- mean(recall_SVM_pd)
> avg_f1 <- mean(f1_SVM_pd)
> avg_accuracy
[1] 0.5558222
> avg_sensitivity
[1] 0.3079321
> avg_specificity
[1] 0.6995256
> avg_precision
[1] 0.3725526
> avg_recall
[1] 0.3079321
> avg_f1
[1] 0.3364241
>
```

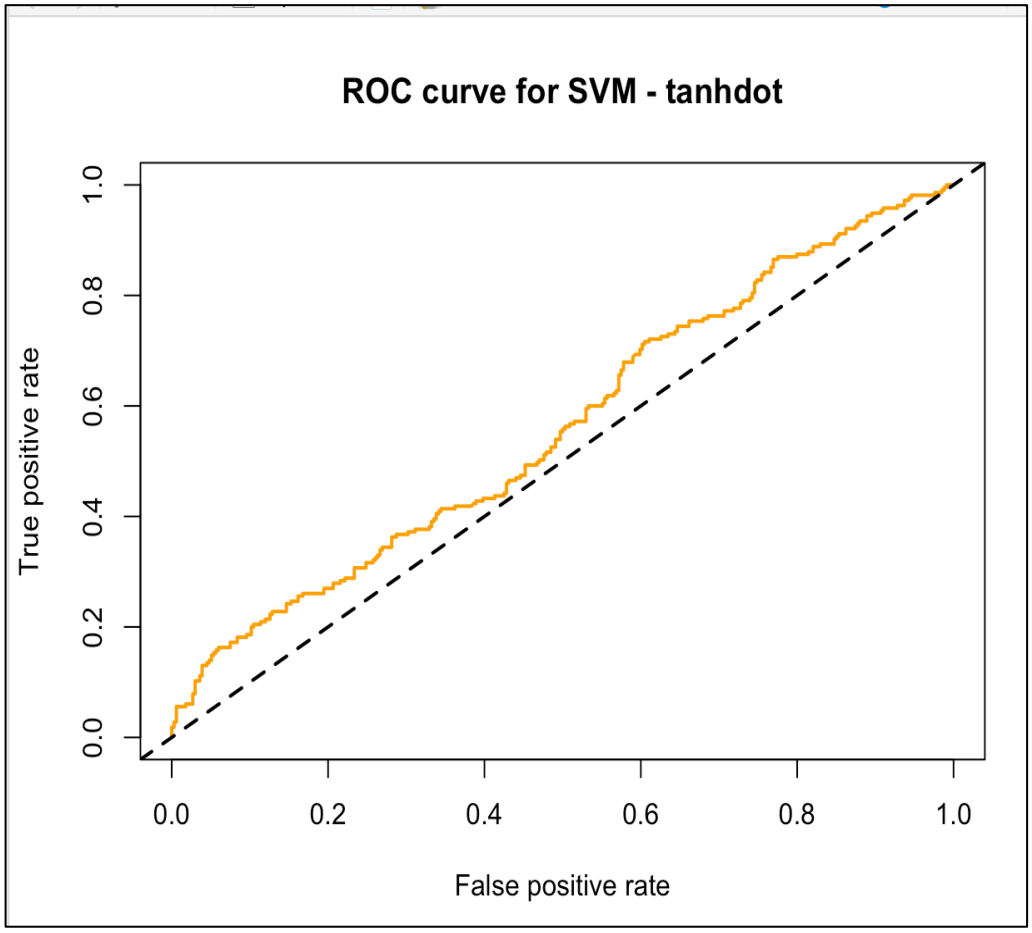
Cross Table and Confusion Matrix

Below are the cross table and confusion matrix for the SVM, ***tanhdot***

Also providing the evaluation metrices achieved by using this model.

```
> CrossTable(svm_results_th$actual_type, svm_results_th$predict_type, dnn=c('Y', 'N'),
+            prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  549


             | N
           Y |         N |         Y | Row Total |
-------------|-----------|-----------|-----------|
           N |       214 |       120 |       334 |
             |     0.629 |     0.574 |           |
-------------|-----------|-----------|-----------|
           Y |       126 |        89 |       215 |
             |     0.371 |     0.426 |           |
-------------|-----------|-----------|-----------|
Column Total |       340 |       209 |       549 |
             |     0.619 |     0.381 |           |
-------------|-----------|-----------|-----------|
```

```
> th_confusion_matrix
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 214 120
         Y 126  89

               Accuracy : 0.5519
                 95% CI : (0.5092, 0.594)
    No Information Rate : 0.6193
    P-Value [Acc > NIR] : 0.9995

                  Kappa : 0.0549

 Mcnemar's Test P-Value : 0.7499

            Sensitivity : 0.4258
            Specificity : 0.6294
         Pos Pred Value : 0.4140
         Neg Pred Value : 0.6407
              Precision : 0.4140
                 Recall : 0.4258
                     F1 : 0.4198
             Prevalence : 0.3807
         Detection Rate : 0.1621
   Detection Prevalence : 0.3916
      Balanced Accuracy : 0.5276

       'Positive' Class : Y
```




```
agreement_th
    FALSE      TRUE
0.4480874 0.5519126
```

K-fold tanhot

```
> avg_accuracy <- mean(accuracy_SVM_th)
> avg_sensitivity <- mean(sensitivity_SVM_th)
> avg_specificity <- mean(specificity_SVM_th)
> avg_precision <- mean(precision_SVM_th)
> avg_recall <- mean(recall_SVM_th)
> avg_f1 <- mean(f1_SVM_th)
> avg_accuracy
[1] 0.5198528
> avg_sensitivity
[1] 0.4112191
> avg_specificity
[1] 0.5828589
> avg_precision
[1] 0.3623514
> avg_recall
[1] 0.4112191
> avg_f1
[1] 0.3845745
>
```

Decision Trees

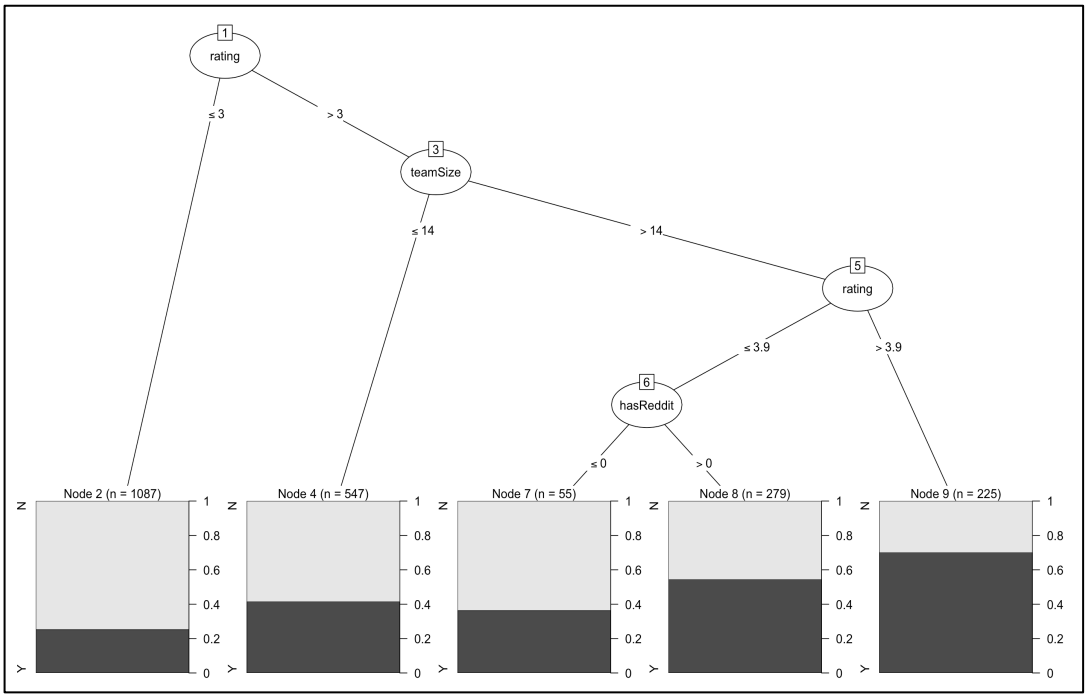
Cross Table and Confusion Matrix

Below are the cross table and confusion matrix for the Decision Trees.

Also providing the evaluation metrices achieved by using this model.

```
> DT_model

Call:
C5.0.formula(formula = success ~ ., data = DT_train)

Classification Tree
Number of samples: 2193
Number of predictors: 11

Tree size: 5

Non-standard options: attempt to group attributes
```

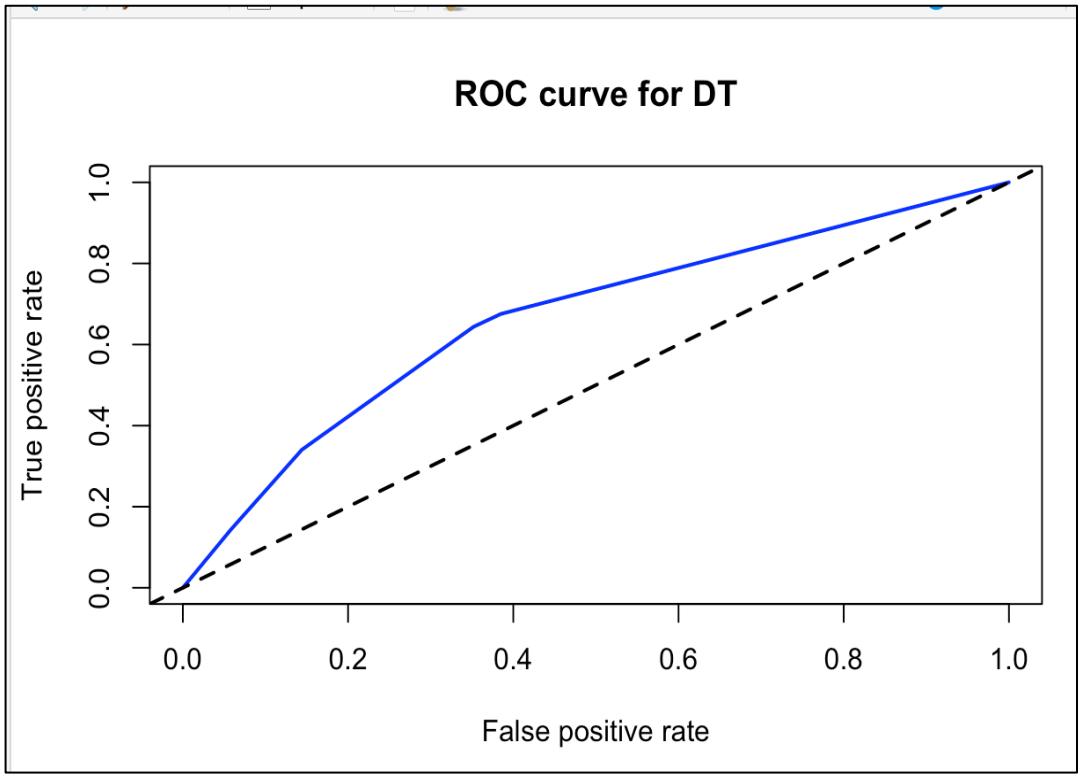
```
C5.0 [Release 2.07 GPL Edition]         Wed May  8 12:00:32 2024
-------------------------------

Class specified by attribute `outcome'

Read 2193 cases (12 attributes) from undefined.data

Decision tree:

rating <= 3: N (1087/275)
rating > 3:
:...teamSize <= 14: N (547/227)
    teamSize > 14:
    :...rating > 3.9: Y (225/67)
        rating <= 3.9:
        :...hasReddit <= 0: N (55/20)
            hasReddit > 0: Y (279/127)


Evaluation on training data (2193 cases):

            Decision Tree
          ----------------
          Size      Errors

             5  716(32.6%)   <<


           (a)   (b)    <-classified as
          ----  ----
          1167   194    (a): class N
           522   310    (b): class Y


        Attribute usage:

        100.00% rating
         50.43% teamSize
         15.23% hasReddit


Time: 0.0 secs
```




```
> library(gmodels)
> CrossTable(DT_pred, DT_test$success, prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE)


   Cell Contents
|-------------------------|
|                       N |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  549


             | DT_test$success
     DT_pred |         N |         Y | Row Total |
-------------|-----------|-----------|-----------|
           N |       309 |       124 |       433 |
             |     0.563 |     0.226 |           |
-------------|-----------|-----------|-----------|
           Y |        52 |        64 |       116 |
             |     0.095 |     0.117 |           |
-------------|-----------|-----------|-----------|
Column Total |       361 |       188 |       549 |
-------------|-----------|-----------|-----------|
```

```
> confusionMatrix_DT
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 309 124
         Y  52  64

               Accuracy : 0.6794
                 95% CI : (0.6386, 0.7183)
    No Information Rate : 0.6576
    P-Value [Acc > NIR] : 0.1504

                  Kappa : 0.2162

 Mcnemar's Test P-Value : 8.707e-08

            Sensitivity : 0.3404
            Specificity : 0.8560
         Pos Pred Value : 0.5517
         Neg Pred Value : 0.7136
              Precision : 0.5517
                 Recall : 0.3404
                     F1 : 0.4211
             Prevalence : 0.3424
         Detection Rate : 0.1166
   Detection Prevalence : 0.2113
      Balanced Accuracy : 0.5982

       'Positive' Class : Y
```

```
> auc_object_DT
A performance instance
  'Area under the ROC curve'
> auc_object_DT@y.values[[1]]
[1] 0.6642969
>
```



DT – K Fold Validation

```
> avg_accuracy
[1] 0.626567
> avg_sensitivity
[1] 0.4578321
> avg_specificity
[1] 0.7476074
> avg_precision
[1] 0.5630237
> avg_recall
[1] 0.4578321
> avg_f1
[1] 0.4985604
```

Adaptive Boosting

```
> DT_boost10

Call:
C5.0.default(x = select(DT_train, -success), y = DT_train$success, trials = 10)

Classification Tree
Number of samples: 2193
Number of predictors: 11

Number of boosting iterations: 10
Average tree size: 3.6

Non-standard options: attempt to group attributes

>
```

AdaBoost – K Fold Validation

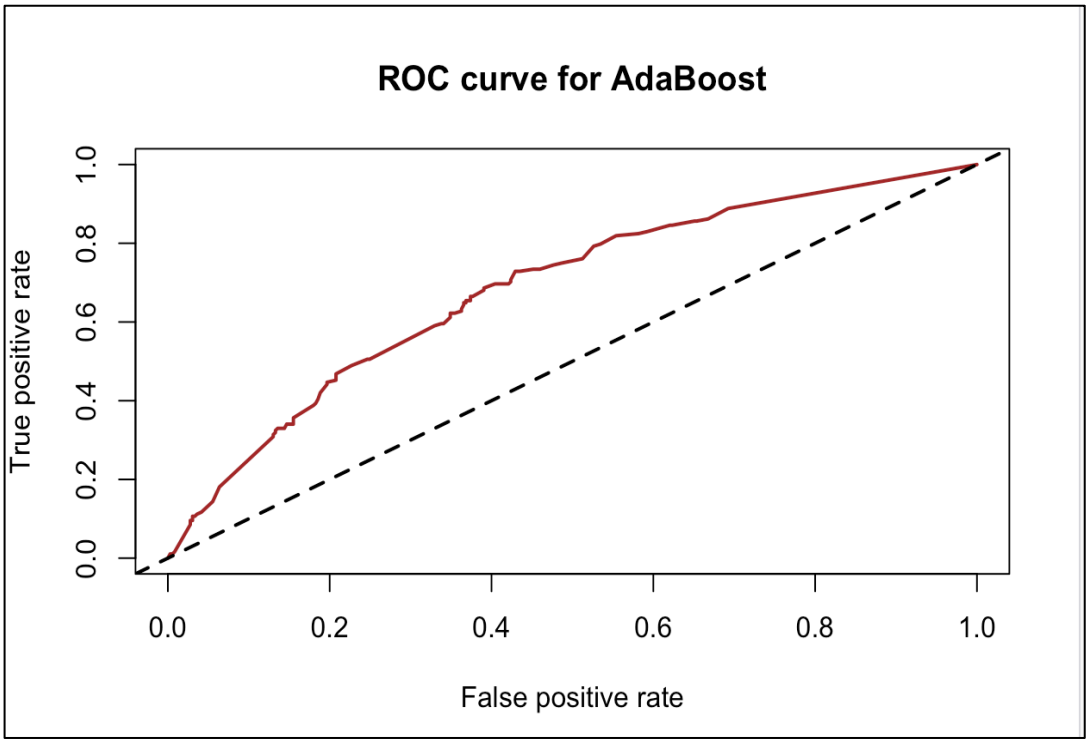
```
> avg_f1 <- mean(f1_DTBoost)
> avg_accuracy
[1] 0.6525363
> avg_sensitivity
[1] 0.4394163
> avg_specificity
[1] 0.8004318
> avg_precision
[1] 0.6064369
> avg_recall
[1] 0.4394163
> avg_f1
[1] 0.5087929
>
```

```
Evaluation on training data (2193 cases):

Trial       Decision Tree
-----     ----------------
          Size      Errors

   0         5  716(32.6%)
   1         3  770(35.1%)
   2         3  857(39.1%)
   3         5  816(37.2%)
   4         2  813(37.1%)
   5         2  796(36.3%)
   6         4  868(39.6%)
   7         4  851(38.8%)
   8         5  792(36.1%)
   9         3  726(33.1%)
boost           699(31.9%)   <<


           (a)   (b)    <-classified as
          ----  ----
          1153   208    (a): class N
           491   341    (b): class Y


        Attribute usage:

        100.00% rating
        100.00% teamSize
        100.00% days_duration
         97.49% hasVideo
         78.66% hasGithub
         63.25% hasReddit


Time: 0.0 secs
```

Cross Table and Confusion Matrix

```
> CrossTable(DT_pred_boost10, DT_test$success, prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c
('predicted default', 'actual default'))


   Cell Contents
|-------------------------|
|                       N |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  549


                 | actual default
predicted default |         N |         Y | Row Total |
------------------|-----------|-----------|-----------|
                N |       305 |       121 |       426 |
                  |     0.556 |     0.220 |           |
------------------|-----------|-----------|-----------|
                Y |        56 |        67 |       123 |
                  |     0.102 |     0.122 |           |
------------------|-----------|-----------|-----------|
     Column Total |       361 |       188 |       549 |
------------------|-----------|-----------|-----------|
```

```
> adaboost_matrix
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 305 121
         Y  56  67

               Accuracy : 0.6776
                 95% CI : (0.6367, 0.7166)
    No Information Rate : 0.6576
    P-Value [Acc > NIR] : 0.1726

                  Kappa : 0.2194

 Mcnemar's Test P-Value : 1.505e-06

            Sensitivity : 0.3564
            Specificity : 0.8449
         Pos Pred Value : 0.5447
         Neg Pred Value : 0.7160
              Precision : 0.5447
                 Recall : 0.3564
                     F1 : 0.4309
             Prevalence : 0.3424
         Detection Rate : 0.1220
   Detection Prevalence : 0.2240
      Balanced Accuracy : 0.6006

       'Positive' Class : Y
```



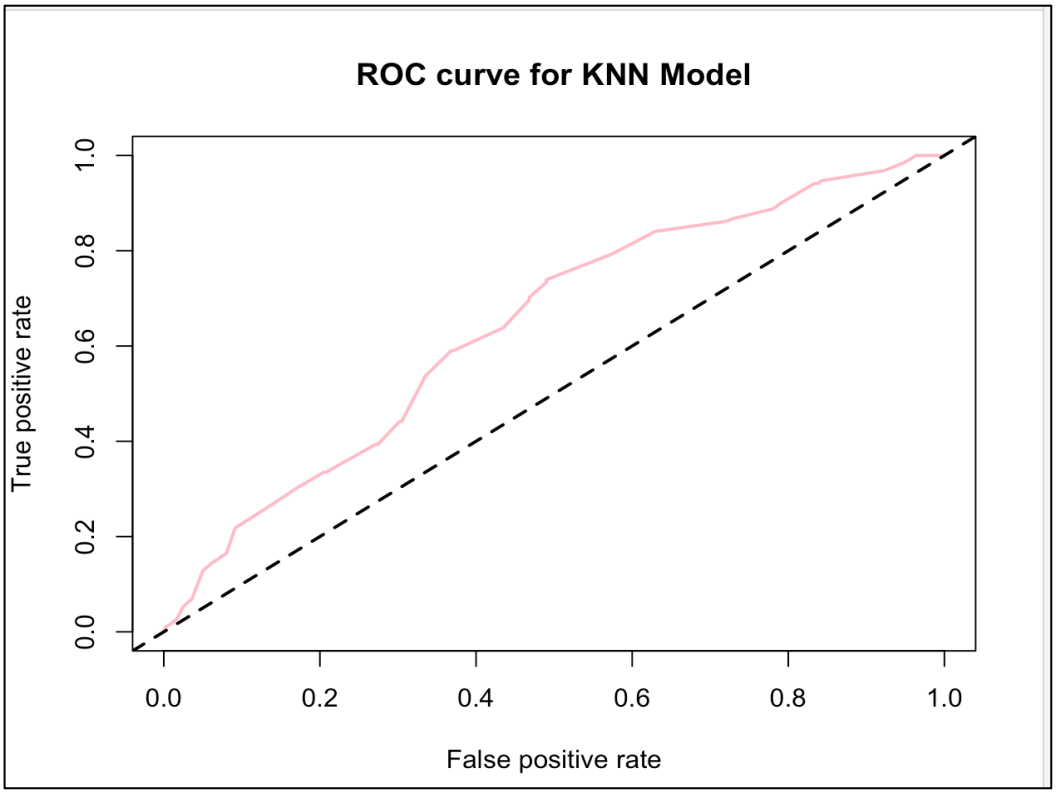

```
A performance instance
  'Area under the ROC curve'
> auc_object_DTBoost@y.values[[1]]
[1] 0.6836138
```

K-NN

Cross Table and Confusion Matrix

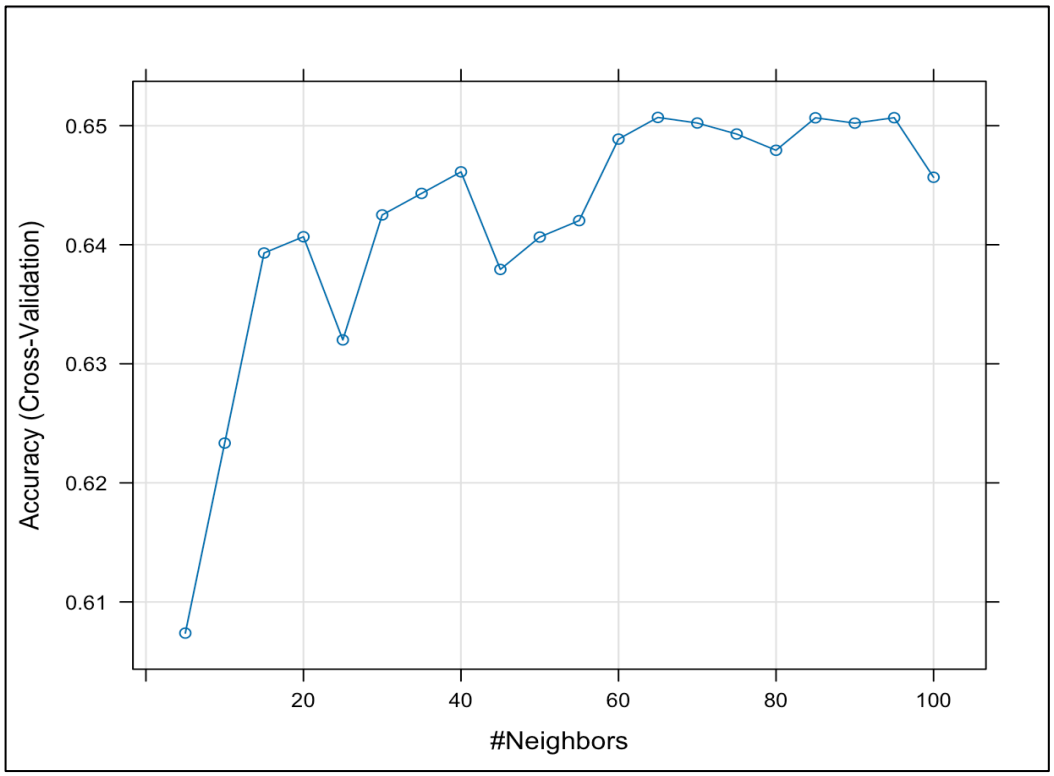
```
> CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  549


                 | wbcd_test_pred
wbcd_test_labels |         N |         Y | Row Total |
-----------------|-----------|-----------|-----------|
               N |       307 |        54 |       361 |
                 |     0.850 |     0.150 |     0.658 |
                 |     0.696 |     0.500 |           |
                 |     0.559 |     0.098 |           |
-----------------|-----------|-----------|-----------|
               Y |       134 |        54 |       188 |
                 |     0.713 |     0.287 |     0.342 |
                 |     0.304 |     0.500 |           |
                 |     0.244 |     0.098 |           |
-----------------|-----------|-----------|-----------|
    Column Total |       441 |       108 |       549 |
                 |     0.803 |     0.197 |           |
-----------------|-----------|-----------|-----------|
```

```
+                                                mode = "everything")
> confusionMatrix_KNN
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 307 134
         Y  54  54

               Accuracy : 0.6576
                 95% CI : (0.6162, 0.6972)
    No Information Rate : 0.6576
    P-Value [Acc > NIR] : 0.5198

                  Kappa : 0.1533

 Mcnemar's Test P-Value : 8.329e-09

            Sensitivity : 0.28723
            Specificity : 0.85042
         Pos Pred Value : 0.50000
         Neg Pred Value : 0.69615
              Precision : 0.50000
                 Recall : 0.28723
                     F1 : 0.36486
             Prevalence : 0.34244
         Detection Rate : 0.09836
   Detection Prevalence : 0.19672
      Balanced Accuracy : 0.56882

       'Positive' Class : Y
```




```
> auc_object_KNN@y.values[[1]]
[1] 0.6411564
>
```

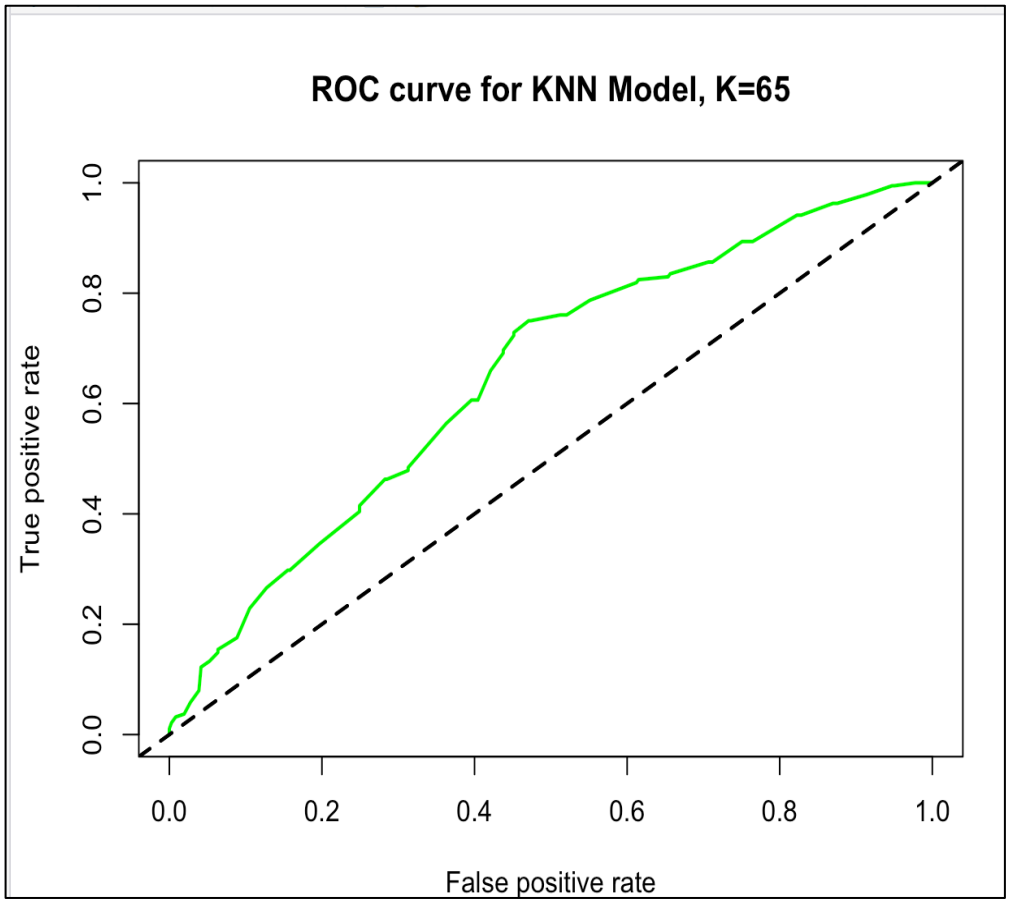
Determining the best k, k = 65




```
> print(knn_model_cv)
k-Nearest Neighbors

2193 samples
   9 predictor
   2 classes: 'N', 'Y'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1974, 1974, 1974, 1974, 1972, 1973, ...
Resampling results across tuning parameters:

  k    Accuracy   Kappa
    5  0.6073810  0.1297818
   10  0.6233401  0.1569568
   15  0.6393073  0.1782994
   20  0.6406730  0.1747745
   25  0.6320075  0.1535103
   30  0.6425057  0.1738631
   35  0.6443136  0.1741728
   40  0.6461214  0.1742318
   45  0.6379271  0.1562415
   50  0.6406461  0.1549273
   55  0.6420263  0.1581157
   60  0.6488735  0.1697023
   65  0.6506938  0.1728652
   70  0.6502248  0.1714708
   75  0.6492992  0.1698143
   80  0.6479355  0.1661785
   85  0.6506628  0.1708577
   90  0.6502145  0.1686604
   95  0.6506711  0.1685529
  100  0.6456669  0.1580143

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 65.
```

```
> CrossTable(x = wbcd_test_labels, y = wbcd_test_pred_65, prop.chisq=FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  549


                 | wbcd_test_pred_65
wbcd_test_labels |         N |         Y | Row Total |
-----------------|-----------|-----------|-----------|
               N |       315 |        46 |       361 |
                 |     0.873 |     0.127 |     0.658 |
                 |     0.695 |     0.479 |           |
                 |     0.574 |     0.084 |           |
-----------------|-----------|-----------|-----------|
               Y |       138 |        50 |       188 |
                 |     0.734 |     0.266 |     0.342 |
                 |     0.305 |     0.521 |           |
                 |     0.251 |     0.091 |           |
-----------------|-----------|-----------|-----------|
    Column Total |       453 |        96 |       549 |
                 |     0.825 |     0.175 |           |
-----------------|-----------|-----------|-----------|
```

```
> confusionMatrix_KNN_65 <- confusionMatrix(data = factor(wbcd_test_pred_65, levels = levels(factor(wbcd_tes
t_labels))),
+                                         reference = factor(wbcd_test_labels),positive = "Y",
+                                         mode= "everything")
> confusionMatrix_KNN_65
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 315 138
         Y  46  50

               Accuracy : 0.6648
                 95% CI : (0.6236, 0.7043)
    No Information Rate : 0.6576
    P-Value [Acc > NIR] : 0.3781

                  Kappa : 0.1569

 Mcnemar's Test P-Value : 1.965e-11

            Sensitivity : 0.26596
            Specificity : 0.87258
         Pos Pred Value : 0.52083
         Neg Pred Value : 0.69536
              Precision : 0.52083
                 Recall : 0.26596
                     F1 : 0.35211
             Prevalence : 0.34244
         Detection Rate : 0.09107
   Detection Prevalence : 0.17486
      Balanced Accuracy : 0.56927

       'Positive' Class : Y
```




```
A performance instance
  'Area under the ROC curve'
> auc_object_KNN_65@y.values[[1]]
[1] 0.6508222
>
```

K fold cross validation

```
> avg_accuracy <- mean(accuracy)
> avg_sensitivity <- mean(sensitivity)
> avg_specificity <- mean(specificity)
> avg_precision <- mean(precision)
> avg_recall <- mean(recall)
> avg_f1 <- mean(f1)
> avg_accuracy
[1] 0.6509861
> avg_sensitivity
[1] 0.2784314
> avg_specificity
[1] 0.8716326
> avg_precision
[1] 0.5636504
> avg_recall
[1] 0.2784314
> avg_f1
[1] 0.370844
>
```

Logistic Regression

Cross table and Confusion Matrix

```
> confusion_matrix_LR
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 276 134
         1  57  81

               Accuracy : 0.6515
                 95% CI : (0.6099, 0.6914)
    No Information Rate : 0.6077
    P-Value [Acc > NIR] : 0.01935

                  Kappa : 0.2195

 Mcnemar's Test P-Value : 3.816e-08

            Sensitivity : 0.3767
            Specificity : 0.8288
         Pos Pred Value : 0.5870
         Neg Pred Value : 0.6732
              Precision : 0.5870
                 Recall : 0.3767
                     F1 : 0.4589
             Prevalence : 0.3923
         Detection Rate : 0.1478
   Detection Prevalence : 0.2518
      Balanced Accuracy : 0.6028

       'Positive' Class : 1
```

```
> print(cross_table)
       Predicted
Actual   0   1
     0 276  57
     1 134  81
```

AUC value

```
Call:
roc.default(response = LR_train_data$success, predictor = logistic_model$fitted.values,     plot = TRUE, legacy.axes = TRUE, xlab = "False Po
sitive Rate",     ylab = "True Positive Rate", col = "blue", main = "ROC Curve for Success")

Data: logistic_model$fitted.values in 1389 controls (LR_train_data$success 0) < 805 cases (LR_train_data$success 1).
Area under the curve: 0.7021
>
```

When threshold > 0.2

```
          Reference
Prediction   0   1
         0  56 277
         1   8 207

               Accuracy : 0.4799
                 95% CI : (0.4374, 0.5227)
    No Information Rate : 0.8832
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1072

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.4277
            Specificity : 0.8750
         Pos Pred Value : 0.9628
         Neg Pred Value : 0.1682
              Precision : 0.9628
                 Recall : 0.4277
                     F1 : 0.5923
             Prevalence : 0.8832
         Detection Rate : 0.3777
   Detection Prevalence : 0.3923
      Balanced Accuracy : 0.6513

       'Positive' Class : 1
```

```
> print(cross_table)
        Predicted
Actual    0   1
     0   56 277
     1    8 207
>
```

Data-Visualization.R

```
install.packages("skimr")
library(skimr)
library(dplyr)
install.packages("caret")
library(caret)

library(dplyr)
library(tidyr)
library(ggplot2)

rm(list=ls())
wbcd <- read.csv('LUBS5990M_courseworkData_2324.csv', stringsAsFactors
= FALSE, encoding = 'UTF-8')
str(wbcd)
wbcd_raw <- wbcd #To store the raw data
head(wbcd)

install.packages("Hmisc")
library(Hmisc)
describe(wbcd)

#wbcd_raw$success <- ifelse(wbcd_raw$success == "Yes", 1, 0)
#wbcd$success <- ifelse(wbcd_raw$success == "Yes", 1, 0)

#------------------------------ success ----------------------------
----------------------------------------#
pie(prop.table(table(wbcd$success)),
    labels = paste(names(table(wbcd$success)), ": ", round(prop.table(
table(wbcd$success)) * 100, 1), "% (Count:", table(wbcd$success), ")")
,
    main = "Pie Chart of Success",
    col = rainbow(length(table(wbcd$success))),
    cex = 0.8)



# Create a bar plot
barplot(prop.table(table(wbcd$success)),
        main = "Bar Plot of Cleaned Data",
        col = rainbow(length(table(wbcd$success))),
        xlab = "Success",
        ylab = "Proportion",
        ylim = c(0, 1))

# Add labels to the bars
text(x = 1:length(table(wbcd$success)),
     y = prop.table(table(wbcd$success)),
     labels = paste(names(table(wbcd$success)), ": ",
                    round(prop.table(table(wbcd$success)) * 100, 1),
                    "% (Count:", table(wbcd$success), ")"),
     cex = 0.8,
```

```
     pos = 3)
#------------------------------ brandslogan ------------------------
-----------------------------------------#
install.packages('topicmodels')
install.packages('wordcloud')
library(topicmodels)
library(wordcloud)
install.packages("tm")
install.packages("tokenizers")
library(tm)
library(tokenizers)
head(wbcd)
myCorpus <- Corpus(VectorSource(wbcd$brandSlogan))
dtmdocs <- DocumentTermMatrix(myCorpus,
                              control = list(lemma=TRUE,
                                             removePunctuation = TRUE,
                                             removeNumbers = TRUE,
                                             stopwords = TRUE,
                                             tolower = TRUE))
dtm.new <- as.matrix(dtmdocs)
frequency <- colSums(dtm.new)
frequency <- sort(frequency, decreasing=TRUE)
doc_length <- rowSums(dtm.new)
words <- names(frequency)
wordcloud(words[1:100], frequency[1:25], rot.per=0.15,
          random.order = FALSE, scale=c(4,0.5),
          random.color = FALSE, colors=brewer.pal(8,"Dark2"))


#------------------------------ TeamSize ---------------------------
---------------------------------------#

hist(wbcd$teamSize, main = "Histogram of Team Size",xlab = "Team Size"
, ylab = "Frequency")
median(wbcd$teamSize, na.rm = TRUE)
mean(wbcd$teamSize, na.rm = TRUE)
sum(!complete.cases(wbcd$teamSize))

#------------------------------ Rating -----------------------------
-------------------------------------#
hist(wbcd$rating, main = "Histogram of Ratings",xlab = "Ratings", ylab
= "Frequency")
sum(!complete.cases(wbcd$rating))
mean(wbcd$rating, na.rm = TRUE)


#------------------------------ CountryRegion ----------------------
-----------------------------------------#
# Create a bar plot for the top 10 countryRegion
barplot(sort(table(wbcd$countryRegion), decreasing = TRUE)[1:10],
        main = "Top 10 Country Regions",
        xlab = "Country Region",
        ylab = "Count",
        col = rainbow(length(sort(table(wbcd$countryRegion), decreasin
```

```
g = TRUE)[1:10])))

sort(table(wbcd$countryRegion), decreasing = TRUE)[1:10]
table(wbcd$countryRegion)

#------------------------------ Platform ----------------------------
----------------------------------------#
barplot(sort(table(wbcd$platform), decreasing = TRUE)[1:10],
        main = "Top 10 Platforms",
        xlab = "Platform",
        ylab = "Count",
        col = rainbow(length(sort(table(wbcd$platform), decreasing = T
RUE)[1:10])))
table(wbcd$platform)

#------------------------------ priceUSD ----------------------------
----------------------------------------#
boxplot(wbcd$priceUSD,
        main = "Boxplot of priceUSD",
        xlab = "priceUSD")
sum(!complete.cases(wbcd$priceUSD))
max(wbcd$priceUSD,na.rm = TRUE)
max(wbcd$priceUSD,na.rm = TRUE)


#------------------------------ minInvestment ----------------------
----------------------------------------------#
ggplot(wbcd)+geom_histogram(aes(minInvestment,fill=success))
sum(!complete.cases(wbcd$minInvestment))
table(wbcd$minInvestment, wbcd$success)

#------------------------------ distributedPercentage ---------------
----------------------------------------------------#
boxplot(wbcd$distributedPercentage,
        main = "Boxplot of distributedPercentage",
        xlab = "distributedPercentage")
subset(wbcd, distributedPercentage > 1)

#------------------------------ coinNum ----------------------------
----------------------------------------#
boxplot(wbcd$coinNum,
        main = "Boxplot of coinNum",
        xlab = "coinNum")
max(wbcd$coinNum)
sum(!complete.cases(wbcd$coinNum))

#------------------------------ hasVideo, hasGithub, hasReddit ------
-----------------------------------#
ggplot(wbcd)+geom_histogram(aes(hasVideo,fill=success))
ggplot(wbcd)+geom_histogram(aes(hasGithub,fill=success))
ggplot(wbcd)+geom_histogram(aes(hasReddit,fill=success))

table(wbcd$hasVideo, wbcd$success)
table(wbcd$hasGithub, wbcd$success)
```

```
table(wbcd$hasReddit, wbcd$success)

sum(!complete.cases(wbcd$hasVideo))
sum(!complete.cases(wbcd$hasGithub))
sum(!complete.cases(wbcd$hasReddit))

#-------------------------------#---------------days_duration---------
#-------------------------------#--------------------------#


wbcd <- wbcd %>%
  mutate(days_duration =
           as.numeric(difftime(as.Date(wbcd$endDate,format = "%d/%m/%Y
"),
                               as.Date(wbcd$startDate, format = "%d/%m
/%Y"), units = "days")))

sum(!complete.cases(wbcd$days_duration))
sum(wbcd$days_duration < 0)
max(wbcd$days_duration)   #3722
min(wbcd$days_duration) #-203
sum(!complete.cases(wbcd$days_duration))


boxplot(wbcd$days_duration,
        main = "Boxplot of days_duration",
        xlab = "days_duration")


#-------------------------------#-----------------------------#------
-------------------------#--------------------------#


install.packages("corrgram")
library(corrgram)
corrgram(wbcd_raw) #need to make the dependent variable numeric TBD
summary(wbcd_raw)
wbcd_old_raw <- wbcd_raw

corrgram(wbcd_raw)
wbcd_raw <- wbcd_raw[-1]
corrgram(wbcd_raw)
corrgram(wbcd_cleaned)
wbcd_cleaned_raw <- wbcd_cleaned
wbcd_cleaned <- wbcd
#wbcd_cleaned_raw$success <- ifelse(wbcd_cleaned_raw$success == "Yes",
1, 0)
wbcd_cleaned$success <- ifelse(wbcd_cleaned$success == "Yes", 1, 0)
wbcd_old_raw$success <- ifelse(wbcd_old_raw$success == "Yes", 1, 0)
wbcd_cleaned <- wbcd_cleaned[-1]
corrgram(wbcd_cleaned)
#wbcd_cleaned <- subset(wbcd_cleaned, select = -is_top_6)

head(wbcd_cleaned)
```

Data Pre Processing.R

```
wbcd <- wbcd[-1]
head(wbcd)
skim(wbcd) #to get the complete summary of the dataset

table(wbcd$success)
table(wbcd$platform)

library(dplyr)

wbcd <- read.csv('LUBS5990M_courseworkData_2324.csv', stringsAsFactors
= FALSE, encoding = 'UTF-8')
wbcd_raw <- read.csv('LUBS5990M_courseworkData_2324.csv', stringsAsFac
tors = FALSE, encoding = 'UTF-8')


#that the target feature is coded as a factor
#wbcd$success <- factor(wbcd$success, levels = c("N", "Y"),
#                       labels = c("No", "Yes"))

head(wbcd)

round(prop.table(table(wbcd$success)) * 100, digits = 1)
table(wbcd$success)
table(wbcd$countryRegion)
names(sort(table(wbcd$countryRegion), decreasing = TRUE))[1:5]

#------------------------------ Country ----------------------------
----------------------------------------#
wbcd$countryRegion <- gsub("Curacao", "Curaçao", wbcd$countryRegion, i
gnore.case = TRUE)
wbcd$countryRegion <- gsub("SINGAPORE", "Singapore", wbcd$countryRegio
n, ignore.case = TRUE)
wbcd$countryRegion <- gsub("usa", "USA", wbcd$countryRegion, ignore.ca
se = TRUE)
wbcd$countryRegion <- gsub("india", "India", wbcd$countryRegion, ignor
e.case = TRUE)

wbcd$is_top_25 <- ifelse(wbcd$countryRegion %in% c("USA", "UK", "Switz
erland","Russia",
                                                   "Singapore","Lithua
nia","Australia",
                                                   "Gibraltar","German
y","Canada",
                                                  "Israel","Ukraine",
"France","Spain",
                                                  "Poland","Liechtenst
ein","China",
                                                  "Luxembourg",
                                                  "Costa Rica",
                                                  "Argentina",
                                                  "Serbia",
```

```
                                                    "Slovakia",
                                                    "Slovenia",
                                                    "Sweden"
                                                    ), 1, 0)



#------------------------------ Days Duration -----------------------
-------------------------------------------#
wbcd <- wbcd %>%
  mutate(days_duration =
           as.numeric(difftime(as.Date(wbcd$endDate,format = "%d/%m/%Y
"),
                               as.Date(wbcd$startDate, format = "%d/%m
/%Y"), units = "days")))

summary(wbcd$days_duration)
table(wbcd$days_duration)

#------------------------------ Platform Cleansing ------------------
-------------------------------------------------#

wbcd$platform <- gsub("Ethereum", "Ethereum", wbcd$platform, ignore.ca
se = TRUE)
wbcd$platform <- gsub("Ethererum", "Ethereum", wbcd$platform, ignore.c
ase = TRUE)
wbcd$platform <- gsub("Ethereum, Waves", "Ethereum", wbcd$platform, ig
nore.case = TRUE)
wbcd$platform <- gsub("Etherum", "Ethereum", wbcd$platform, ignore.cas
e = TRUE)
wbcd$platform <- gsub("Ethereum   ", "Ethereum", wbcd$platform, ignore
.case = TRUE)
wbcd$platform <- gsub("Ethereum ", "Ethereum", wbcd$platform, ignore.c
ase = TRUE)
wbcd$platform <- gsub("Ethereum ", "Ethereum", wbcd$platform, ignore.c
ase = TRUE)
wbcd$platform <- gsub("Ethereum ", "Ethereum", wbcd$platform, ignore.c
ase = TRUE)
wbcd$platform <- gsub(" Ethereum", "Ethereum", wbcd$platform, ignore.c
ase = TRUE)

# Made 1 as ethereum
head(wbcd)
sum(complete.cases(wbcd)) #2442
sum(!complete.cases(wbcd)) #325
#2767

wbcd$is_Ethereum <- 0
wbcd$is_Ethereum <- ifelse(grepl("Ethereum", gsub(" ", "", wbcd$platfo
rm)), 1, 0)
table(wbcd$platform, wbcd$is_Ethereum)
table(wbcd$is_Ethereum,wbcd$success)
```

```
ggplot(wbcd)+geom_histogram(aes(is_Ethereum))

#------------------------------ Days duration remove less than 0 ----
-------------------------------------------------------------------#
wbcd <- subset(wbcd, days_duration >=0) #removed 12 data less than 0
max(wbcd$days_duration, na.rm = TRUE)

#removed outlier
wbcd <- subset(wbcd, days_duration < 3722)

#------------------------------ Removed distributedPercentage > 1 ---
-------------------------------------------------------------------#
wbcd <- subset(wbcd, distributedPercentage <= 1) ##removed all data le
ss than 1.00 %

#------------------------------ Removed priceUSD oulier -------------
-----------------------------------------------------------#

nrow(subset(wbcd, priceUSD != 39384 | is.na(priceUSD)))

wbcd_usd <- subset(wbcd, priceUSD == 0)
table(wbcd_usd$success,wbcd_usd$priceUSD)

wbcd <- subset(wbcd, priceUSD != 39384 | is.na(priceUSD)) #removed 1 m
ore record with this extreme outlier)
nrow(wbcd) #2743

#------------------------------ Imputation- PriceUSD ----------------
-----------------------------------------------------#

mean_priceUSD <- mean(wbcd$priceUSD, na.rm = TRUE)
mean_priceUSD
wbcd$priceUSD <- ifelse(is.na(wbcd$priceUSD), mean_priceUSD, wbcd$pric
eUSD)


#Before Imputation
hist(wbcd_raw$priceUSD, main = "Histogram of PriceUSD",xlab = "Team Si
ze", ylab = "Frequency")
hist(wbcd_raw$teamSize, main = "Histogram of TeamSize",xlab = "Team Si
ze", ylab = "Frequency")

nrow(wbcd)
nrow(wbcd_raw)


#------------------------------ Imputation- TeamSize ----------------
-----------------------------------------------------#

mean(wbcd$teamSize, na.rm = TRUE)
median(wbcd$teamSize, na.rm = TRUE)
max(wbcd$teamSize, na.rm = TRUE)
min(wbcd$teamSize, na.rm = TRUE)
```

```
median_teamSize <- median(wbcd$teamSize, na.rm = TRUE)
wbcd$teamSize <- ifelse(is.na(wbcd$teamSize), median_teamSize, wbcd$te
amSize)
  #Median imputed It preserves the overall central tendency of the dat
a.

#After Imputation
hist(wbcd$priceUSD, main = "Histogram of PriceUSD after Imputation",xl
ab = "Team Size", ylab = "Frequency")
hist(wbcd$teamSize, main = "Histogram of TeamSize after Imputation",xl
ab = "Team Size", ylab = "Frequency")

#------------------------------ Clean coinNUM ----------------------
-----------------------------------------------#

wbcd <- wbcd[wbcd$coinNum != 22619078416760300, ] #2743 cleaned
#removed 1 extreme value from coinnum

#------------ Capital Raising ----------------- #

wbcd$capital_Raised <- wbcd$priceUSD * wbcd$coinNum
head(wbcd)

ggplot(wbcd)+geom_histogram(aes(is_top_25,fill=success))

pie(prop.table(table(wbcd$is_top_25)),
    labels = paste(names(table(wbcd$is_top_25)), ": ", round(prop.tabl
e(table(wbcd$is_top_25)) * 100, 1), "% (Count:", table(wbcd$is_top_25)
, ")"),
    main = "Pie Chart of Top 25 Countries",
    col = rainbow(length(table(wbcd$is_top_25))),
    cex = 0.8)

mean(wbcd$capital_Raised, na.rm = TRUE)
median(wbcd$capital_Raised, na.rm = TRUE)
max(wbcd$capital_Raised, na.rm = TRUE)
min(wbcd$capital_Raised, na.rm = TRUE)
sum(!complete.cases(wbcd$capital_Raised))



#remove a column
#wbcd <- wbcd[, !colnames(wbcd) %in% c("is_top_5")]




raw_data <- nrow(wbcd_raw)
cleaned_data <- nrow(wbcd)

#------------------------------ Percentage of Deleted Data ----------
-----------------------------------------------------------#

percent_deleted <- ((raw_data - cleaned_data)/raw_data)*100
```

```
percent_deleted #0.907% was cleaned to prepare data for ML processing



#----------------- Part 2  ----------------------
install.packages("VIM",dependencies = T)
library("VIM")
aggr(wbcd_raw, numbers=TRUE, prop=FALSE)
aggr(wbcd, numbers=TRUE, prop=FALSE)


nrow(wbcd)
write.csv(wbcd, file = '/Users/deepu/Documents/ML/wbcd.csv', row.names
= FALSE)
write.csv(wbcd, row.names = FALSE)

wbcd <-wbcd_cleaned
head(wbcd_cleaned)
head(wbcd)

wbcd_cleaned <- wbcd #Master Copy


wbcd_cleaned <- wbcd_cleaned[-1]
wbcd_cleaned$success <- ifelse(wbcd_cleaned$success == "Yes", 1, 0)

corrgram(wbcd_cleaned)


wbcd_model <- subset(wbcd, select = c("success","hasVideo","rating","t
eamSize",
                                      "hasGithub","hasReddit","capital
_Raised",
                                      "minInvestment","distributedPerc
entage",
                                      "is_top_25","days_duration","is_
Ethereum"))




library(Hmisc)
describe(wbcd_cleaned)
```

Random-Forest.R

```
#_-----------------#_-----------------#_-----------------#_-----------
------#_-----------------
#_-----------------  Random Forest - Regression Tree Ensemble Model --
----------##_-----------------
#_-----------------#_-----------------#_-----------------#_-----------
------#_-----------------
install.packages("randomForest")
library(randomForest)
library(caret)
library(pROC)
library(gmodels)
library(caret)
library(tidyverse)
library(ROCR)
wbcd_RF <- wbcd
wbcd_RF <- subset(wbcd_RF, select = c("success","hasVideo","rating","t
eamSize",
                                      "hasGithub","hasReddit","capital_R
aised",
                                      "minInvestment","distributedPercen
tage",
                                      "is_top_25","days_duration","is_Et
hereum"))
head(wbcd_RF)
#wbcd_RF <- wbcd_cleaned
wbcd_RF$success <- factor(wbcd_RF$success)

RF_train_index <- createDataPartition(wbcd_RF$success, p = 0.8, list =
FALSE)
RF_train_data <- wbcd_RF[RF_train_index, ]
RF_test_data <- wbcd_RF[-RF_train_index, ]
RF_train_data_labels <- wbcd_RF[RF_train_index, 1]
RF_test_data_labels <- wbcd_RF[-RF_train_index, "success"]

set.seed(123) # for reproducibility
rf_model <- randomForest(success ~ ., data = RF_train_data, ntree = 50
0, mtry = 4)
rf_predict <- predict(rf_model, RF_test_data)
rf_predict1 <- predict(rf_model, RF_test_data, type = "prob" )
summary(rf_predict)
tree_details <- getTree(rf_model, labelVar = TRUE)
tree_details
plot(rf_model)
importance(rf_model)
print(summary(rf_model))
#plot(rf_model, main="Error Rate by Class", col=c("blue", "red"))
library(gmodels)
CrossTable(x = RF_test_data_labels, y = rf_predict, prop.chisq=FALSE)
summary(rf_predict)
confusionMatrix_RF <- confusionMatrix(rf_predict , RF_test_data_labels
, positive = "Y", mode = "everything")
```

```
confusionMatrix_RF
#Accuracy : 0.6861

accuracy_RF <- confusionMatrix_RF$overall['Accuracy']
sensitivity_RF <- confusionMatrix_RF$byClass['Sensitivity'] # is also
called recall
specificity_RF <- confusionMatrix_RF$byClass['Specificity']
precision_RF <- confusionMatrix_RF$byClass['Precision'] # Precision is
also called Positive Predictive Value (PPV)
recall_RF <- confusionMatrix_RF$byClass['Recall']
f1_RF <- confusionMatrix_RF$byClass['F1']

RF_test_prob <- predict(rf_model, RF_test_data, type = "prob")
RF_results <- data.frame(actual_type = RF_test_data_labels,
                         predict_type = rf_predict,
                         prob_Y = round(RF_test_prob[ , 2], 5),
                         prob_N = round(RF_test_prob[ , 1], 5))
sensitivity(RF_results$predict_type, RF_results$actual_type, positive
= "Y")
specificity(RF_results$predict_type, RF_results$actual_type, negative
= "N")
posPredValue(RF_results$predict_type, RF_results$actual_type, positive
= "Y") # this is precision
sensitivity(RF_results$predict_type, RF_results$actual_type, positive
= "Y") # this is recall


#------------ ROC ---------
predict_object_RF <- prediction(rf_predict1[,2],RF_test_data_labels)
roc_RF <- performance(predict_object_RF, measure = "tpr", x.measure =
"fpr")
plot(roc_RF, main = "ROC curve for Random Forest Model", col = "blue",
lwd = 2)
abline(a = 0, b = 1, lwd = 2, lty = 2)
auc_object_RF <- performance(predict_object_RF, measure = "auc")
auc_object_RF
auc_object_RF@y.values[[1]]







wbcd_train[, -1]

#----------- K fold Validation on RF --------------------#

library(caret)
num_folds <- 10
folds <- createFolds(wbcd_RF$success, k = num_folds)
accuracy_RF <- numeric(num_folds)
sensitivity_RF <- numeric(num_folds)
specificity_RF <- numeric(num_folds)
```

```
precision_RF <- numeric(num_folds)
recall_RF <- numeric(num_folds)
f1_RF <- numeric(num_folds)

# Iterate over each fold
for (i in 1:num_folds) {
  train_indices <- unlist(folds[-i])
  test_indices <- unlist(folds[i])
  RF_train_data <- wbcd_RF[test_indices, ]
  RF_test_data <- wbcd_RF[-test_indices, ]
  RF_train_data_labels <- wbcd_RF[test_indices, 1]
  RF_test_data_labels <- wbcd_RF[-test_indices, "success"]
  RF_model <- randomForest(success ~ ., data = RF_train_data, ntree =
500)
  RF_predictions <- predict(RF_model, newdata = RF_test_data)
  confusion_matrix_rf <- confusionMatrix(data = factor(RF_predictions,
levels = levels(factor(RF_test_data_labels))),
                                         reference = factor(RF_test_dat
a_labels), positive = "Y")

  accuracy_RF[i] <- confusion_matrix_rf$overall['Accuracy']
  sensitivity_RF[i] <- confusion_matrix_rf$byClass['Sensitivity']
  specificity_RF[i] <- confusion_matrix_rf$byClass['Specificity']
  precision_RF[i] <- confusion_matrix_rf$byClass['Precision']
  recall_RF[i] <- confusion_matrix_rf$byClass['Recall']
  f1_RF[i] <- confusion_matrix_rf$byClass['F1']
}

avg_accuracy <- mean(accuracy_RF)
avg_sensitivity <- mean(sensitivity_RF)
avg_specificity <- mean(specificity_RF)
avg_precision <- mean(precision_RF)
avg_recall <- mean(recall_RF)
avg_f1 <- mean(f1_RF)
```

LogisticRegression.R

```
#_-----------------Logistic Regression-------------------------------
---------------##_----------------
#_-----------------#_-----------------#_-----------------#_----------
------#_-----------------
wbcd_LR <- wbcd

wbcd_LR <- subset(wbcd_LR, select = c("success","hasVideo","rating","t
eamSize",
                                      "hasGithub","hasReddit","capital_R
aised",
                                      "minInvestment","distributedPercen
tage",
                                      "is_top_25","days_duration","is_Et
hereum"))
table(wbcd_LR$success)
wbcd_LR$success <- ifelse(wbcd_LR$success == "Y", 1, 0)
set.seed(123)
train_indices <- createDataPartition(wbcd_LR$success, p = 0.8, list =
FALSE)  # 80% train, 20% test
LR_train_data <- wbcd_LR[train_indices, ]
LR_test_data <- wbcd_LR[-train_indices, ]
LR_test_data_labels <- wbcd_LR[-RF_train_index, 1]
logistic_model <- glm(success ~ hasVideo + rating + teamSize + hasGith
ub +
                        hasReddit + capital_Raised + minInvestment + d
istributedPercentage +
                        is_top_25 + days_duration + is_Ethereum,
                      data = LR_train_data, family = binomial)
table(LR_train_data$success)
summary(logistic_model)
par(pty = "s")
roc(LR_train_data$success, logistic_model$fitted.values, plot = TRUE,l
egacy.axes = TRUE,
    xlab="False Positive Rate", ylab = "True Positive Rate", col="blue
", main = "ROC Curve for Success")
summary(logistic_model)
lr_pred <- predict(logistic_model, newdata = LR_test_data, type = "res
ponse")
lr_pred_binary <- ifelse(lr_pred > 0.2, 1, 0)
cross_table <- table(Actual = LR_test_data$success, Predicted = lr_pre
d_binary)
print(cross_table)
confusion_matrix_LR <- confusionMatrix(as.factor(LR_test_data$success)
,as.factor(lr_pred_binary), mode ="everything", positive = "1")
confusion_matrix_LR


#######----------------- K-Fold Cross Validation for Logistic Regressi
on --------

K <- 10
```

```
accuracy <- numeric(K)
sensitivity <- numeric(K)
specificity <- numeric(K)
precision <- numeric(K)
f1 <- numeric(K)
auc <- numeric(K)

for (i in 1:K) {
  validation_indices <- ((i - 1) * nrow(LR_train_data) / K + 1):(i * n
row(LR_train_data) / K)
  validation_set <- LR_train_data[validation_indices, ]
  training_set <- LR_train_data[-validation_indices, ]
  logistic_model <- glm(success ~ hasVideo + rating + teamSize + hasGi
thub +
                          hasReddit + capital_Raised + minInvestment +
distributedPercentage +
                          is_top_25 + days_duration + is_Ethereum,
                        data = training_set, family = binomial)
  predicted_probs <- predict(logistic_model, newdata = validation_set,
type = "response")
  predicted_classes <- ifelse(predicted_probs > 0.5, 1, 0)
  confusion_matrix <- table(predicted_classes, validation_set$success)
  TP <- confusion_matrix[2, 2]
  TN <- confusion_matrix[1, 1]
  FP <- confusion_matrix[1, 2]
  FN <- confusion_matrix[2, 1]

  accuracy[i] <- (TP + TN) / sum(confusion_matrix)
  sensitivity[i] <- TP / (TP + FN)
  specificity[i] <- TN / (TN + FP)
  precision[i] <- TP / (TP + FP)
  f1[i] <- 2 * (precision[i] * sensitivity[i]) / (precision[i] + sensi
tivity[i])
  auc[i] <- roc(response = validation_set$success, predictor = predict
ed_probs)$auc
}
mean_accuracy <- mean(accuracy)
mean_sensitivity <- mean(sensitivity)
mean_specificity <- mean(specificity)
mean_precision <- mean(precision)
mean_f1 <- mean(f1)
mean_auc <- mean(auc)
```

Decision_Tree.R

```
#_-----------------#_-----------------#_-----------------#_-----------
------#_-----------------
#_-----------------  Decision Trees ---------------------------------
------------##_-----------------
#_-----------------#_-----------------#_-----------------#_-----------
------#_-----------------
wbcd_DT <- wbcd
nrow(wbcd_DT)
wbcd_DT <- subset(wbcd_DT, select = c("success","hasVideo","rating","t
eamSize",
                                      "hasGithub","hasReddit","capital
_Raised",
                                      "minInvestment","distributedPerc
entage",
                                      "is_top_25","days_duration","is_
Ethereum"))
head(wbcd_DT)
table(wbcd$success)
wbcd_DT$success <- factor(wbcd_DT$success)
smp_size <- floor(0.8 * nrow(wbcd_DT))
set.seed(12345)
sample(10, 5)

set.seed(12345)
train_ind <- sample(nrow(wbcd_DT), smp_size)
DT_train <- wbcd_DT[train_ind, ]
DT_test <- wbcd_DT[-train_ind, ]
DT_wbcd_train_labels <- wbcd_DT[train_ind, 1]
DT_wbcd_test_labels <- wbcd_DT[-train_ind, "success"]
prop.table(table(DT_train$success))
install.packages("C50")
library(C50)
library(tidyverse)
DT_model <- C5.0(success ~., DT_train)
DT_model
summary(DT_model)
print(summary(DT_model))
plot(DT_model)
DT_pred <- predict(DT_model, DT_test)
DT_predict1 <- predict(DT_model, DT_test, type = "prob" )
library(gmodels)
CrossTable(DT_pred, DT_test$success, prop.chisq = FALSE, prop.c = FALS
E, prop.r = FALSE)
confusionMatrix_DT <- confusionMatrix(DT_pred , DT_wbcd_test_labels, p
ositive = "Y", mode = "everything")
confusionMatrix_DT
library(ROCR)
predict_object_DT <- prediction(DT_predict1[,2],DT_wbcd_test_labels)
roc_DT <- performance(predict_object_DT, measure = "tpr", x.measure =
"fpr")
```

```
plot(roc_DT, main = "ROC curve for DT", col = "blue", lwd = 2)
abline(a = 0, b = 1, lwd = 2, lty = 2)
auc_object_DT <- performance(predict_object_DT, measure = "auc")
auc_object_DT
auc_object_DT@y.values[[1]]
#Accuracy : 0.6642
```

```{rK-fold Validation ——————————————————————-#}

library(caret) library(C50) # Set seed for reproducibility set.seed(12345) num_folds <- 10 folds <- createFolds(DT_wbcd_train_labels, k = num_folds) accuracy_DT <- numeric(num_folds) sensitivity_DT <- numeric(num_folds) specificity_DT <- numeric(num_folds) precision_DT <- numeric(num_folds) recall_DT <- numeric(num_folds) f1_DT <- numeric(num_folds)

for (i in 1:num_folds) { train_indices <- unlist(folds[-i]) test_indices <- unlist(folds[i]) fold_train_data <- wbcd_DT[train_indices, ] fold_test_data <- wbcd_DT[test_indices, ] DT_model <- C5.0(success ~ ., data = fold_train_data) DT_predictions <- predict(DT_model, newdata = fold_test_data) confusion_matrix_dt <- confusionMatrix(data = factor(DT_predictions, levels = levels(factor(fold_test_data$success))), $reference = factor(fold_test_data$success$)$, positive = "Y")

# Store the evaluation metrics for the current fold accuracy_DT[i] <- confusion_matrix_dt$overall['Accuracy']$ $sensitivity_DT[i] <- confusion_matrix_dt$byClass['Sensitivity']$ specificity_DT[i] <- confusion_matrix_dt$byClass['Specificity']$ $precision_DT[i] <- confusion_matrix_dt$byClass['Precision']$ recall_DT[i] <- confusion_matrix_dt$byClass['Recall']$ $f1_DT[i] <- confusion_matrix_dt$byClass['F1'] } avg_accuracy <- mean(accuracy_DT) avg_sensitivity <- mean(sensitivity_DT) avg_specificity <- mean(specificity_DT) avg_precision <- mean(precision_DT) avg_recall <- mean(recall_DT) avg_f1 <- mean(f1_DT)

```{r#}

#Adaptive Boosting
DT_boost10 <- C5.0(select(DT_train, -success), DT_train$success, trial
s = 10)
DT_boost10
summary(DT_boost10)
view(DT_train)
DT_pred_boost10 <- predict(DT_boost10, DT_test)
DT_pred_boost10_prob <- predict(DT_boost10, DT_test, type = "prob")
CrossTable(DT_pred_boost10, DT_test$success, prop.chisq = FALSE, prop.
c = FALSE, prop.r = FALSE, dnn = c('predicted default', 'actual defaul
t'))
adaboost_matrix <- confusionMatrix(as.factor(DT_pred_boost10), as.fact
or(DT_wbcd_test_labels), positive = "Y", mode = "everything")
adaboost_matrix

library(ROCR)
predict_object_DTBoost <- prediction(DT_pred_boost10_prob[,2],DT_wbcd_
```
```

```
test_labels)
roc_DTBoost <- performance(predict_object_DTBoost, measure = "tpr", x.
measure = "fpr")
plot(roc_DTBoost, main = "ROC curve for AdaBoost", col = "brown", lwd
= 2)
abline(a = 0, b = 1, lwd = 2, lty = 2)
auc_object_DTBoost <- performance(predict_object_DTBoost, measure = "a
uc")
auc_object_DTBoost
auc_object_DTBoost@y.values[[1]]
#Accuracy : 0.6836



library(caret)
library(C50)
# Set seed for reproducibility
set.seed(12345)
num_folds <- 10
folds <- createFolds(DT_wbcd_train_labels, k = num_folds)
accuracy_DTBoost <- numeric(num_folds)
sensitivity_DTBoost <- numeric(num_folds)
specificity_DTBoost <- numeric(num_folds)
precision_DTBoost <- numeric(num_folds)
recall_DTBoost <- numeric(num_folds)
f1_DTBoost <- numeric(num_folds)

for (i in 1:num_folds) {
  train_indices <- unlist(folds[-i])
  test_indices <- unlist(folds[i])
  fold_train_data <- wbcd_DT[train_indices, ]
  fold_test_data <- wbcd_DT[test_indices, ]
  DT_model <- C5.0(success ~ ., data = fold_train_data)
  DT_predictions <- predict(DT_boost10, newdata = fold_test_data)
  confusion_matrix_dt <- confusionMatrix(data = factor(DT_predictions,
levels = levels(factor(fold_test_data$success))),
                                         reference = factor(fold_test_
data$success), positive = "Y")

  # Store the evaluation metrics for the current fold
  accuracy_DTBoost[i] <- confusion_matrix_dt$overall['Accuracy']
  sensitivity_DTBoost[i] <- confusion_matrix_dt$byClass['Sensitivity']
  specificity_DTBoost[i] <- confusion_matrix_dt$byClass['Specificity']
  precision_DTBoost[i] <- confusion_matrix_dt$byClass['Precision']
  recall_DTBoost[i] <- confusion_matrix_dt$byClass['Recall']
  f1_DTBoost[i] <- confusion_matrix_dt$byClass['F1']
}
avg_accuracy <- mean(accuracy_DTBoost)
avg_sensitivity <- mean(sensitivity_DTBoost)
avg_specificity <- mean(specificity_DTBoost)
avg_precision <- mean(precision_DTBoost)
avg_recall <- mean(recall_DTBoost)
avg_f1 <- mean(f1_DTBoost)
```

KNN.R

```
#_-----------------#_-----------------#_-----------------#_-----------
------#_-----------------
#_-----------------KNN-----------------------------------------------
##_-----------------
#_-----------------#_-----------------#_-----------------#_-----------
------#_-----------------
nrow(wbcd)

# Model Preparation with Clean Data

wbcd_knn <- wbcd
head(wbcd_knn)
nrow(wbcd_knn)
wbcd_knn <- wbcd_knn[-1]

#We apply normalization to rescale the features to a standard range of
values.
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

head(wbcd_knn)
wbcd_knn <- subset(wbcd_knn, select = c("success","hasVideo","rating",
"teamSize",
                                        "hasGithub","hasReddit","capit
al_Raised",
                                        "minInvestment","distributedPe
rcentage",
                                        "is_top_25","days_duration","i
s_Ethereum"))

wbcd_DT$success <- factor(wbcd_DT$success)

head(wbcd_knn)

str(wbcd_knn)

head(wbcd_knn)
wbcd_knn_n <- as.data.frame(lapply(wbcd_knn[2:11], normalize))
nrow(wbcd_knn_n) #2743

set.seed(12345)
smp_size <- floor(0.8 * nrow(wbcd_knn))
train_ind <- sample(nrow(wbcd_knn), smp_size)
wbcd_train <- wbcd_knn_n[train_ind, ]
wbcd_test<- wbcd_knn_n[-train_ind, ]
nrow(wbcd_train)
nrow(wbcd_test)

wbcd_train_labels <- wbcd_knn[train_ind, 1]
wbcd_test_labels <- wbcd_knn[-train_ind, "success"]
```

```
table(wbcd_test_labels)
library(class)

K = 45 #training size = 2194 so we try its square root 46 as the value
of k first
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test, cl = wbcd_
train_labels, k=K)

library(gmodels)

CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)
confusionMatrix_KNN <- confusionMatrix(data = factor(wbcd_test_pred, l
evels = levels(factor(wbcd_test_labels))),
                                       reference = factor(wbcd_test_la
bels),positive = "Y",
                                       mode = "everything")


confusionMatrix_KNN


install.packages("ROCR")
library(ROCR)

accuracy_knn <- confusionMatrix_KNN$overall['Accuracy']
sensitivity_knn <- confusionMatrix_KNN$byClass['Sensitivity'] # is als
o called recall
specificity_knn <- confusionMatrix_KNN$byClass['Specificity']
precision_knn <- confusionMatrix_KNN$byClass['Precision'] # Precision
is also called Positive Predictive Value (PPV)
recall_knn <- confusionMatrix_KNN$byClass['Recall']
f1_knn <- confusionMatrix_KNN$byClass['F1']

#_-----------------#_----------------- ROC Curve -----------------#_--
---------------#_-----------------
wbcd_test_pred
wbcd_train_labels_knn <- factor(wbcd_train_labels)
wbcd_KNN_pred_prob <- predict(caret::knn3(wbcd_train, wbcd_train_label
s_knn, k = K), wbcd_test)

wbcd_KNN_results <- data.frame(actual_type = wbcd_test_labels,
                               predict_type = wbcd_test_pred,
                               prob_Yes = round(wbcd_KNN_pred_prob[ ,
2], 5),
                               prob_No = round(wbcd_KNN_pred_prob[ , 1
], 5))
pred_object <- prediction(wbcd_KNN_results$prob_Yes, wbcd_KNN_results$
actual_type)
roc_KNN <- performance(pred_object, measure = "tpr", x.measure = "fpr"
)
plot(roc_KNN, main = "ROC curve for KNN Model", col = "pink", lwd = 2)
abline(a = 0, b = 1, lwd = 2, lty = 2)
```

```
auc_object_KNN <- performance(pred_object, measure = "auc")
auc_object_KNN
auc_object_KNN@y.values[[1]]



#Choose the best k
#10-fold cross-validation to find the optimal value of k that maximize
s the model's performance on the training data
train_control <- trainControl(method = "cv", number = 10)
knn_model_cv <- train(x = wbcd_train[, -1],
                      y = wbcd_train_labels,
                      method = "knn",
                      trControl = train_control,
                      tuneGrid = data.frame(k=seq(5,100,by=5)))
pred_knn_k <- predict(knn_model_cv,wbcd_test )
plot(knn_model_cv)
print(knn_model_cv)

K=65
wbcd_test_pred_65 <- knn(train = wbcd_train, test = wbcd_test, cl = wb
cd_train_labels, k=K)
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred_65, prop.chisq=FAL
SE)
confusionMatrix_KNN_65 <- confusionMatrix(data = factor(wbcd_test_pred
_65, levels = levels(factor(wbcd_test_labels))),
                                          reference = factor(wbcd_test
_labels),positive = "Y")

confusionMatrix_KNN_65 <- confusionMatrix(data = factor(wbcd_test_pred
_65, levels = levels(factor(wbcd_test_labels))),
                                          reference = factor(wbcd_test
_labels),positive = "Y",
                                          mode= "everything")
confusionMatrix_KNN_65

#0.6648

#_-----------------#_----------------- ROC Curve for k= 65------------
-----#_-----------------#_-----------------
wbcd_test_pred
wbcd_train_labels_knn <- factor(wbcd_train_labels)
wbcd_KNN_pred_prob_65 <- predict(caret::knn3(wbcd_train, wbcd_train_la
bels_knn, k = K), wbcd_test)

wbcd_KNN_results_65 <- data.frame(actual_type = wbcd_test_labels,
                                  predict_type = wbcd_test_pred,
                                  prob_Yes = round(wbcd_KNN_pred_prob_65[
, 2], 5),
                                  prob_No = round(wbcd_KNN_pred_prob_65[
, 1], 5))
pred_object_65 <- prediction(wbcd_KNN_results_65$prob_Yes, wbcd_KNN_re
sults_65$actual_type)
roc_KNN_65 <- performance(pred_object_65, measure = "tpr", x.measure =
```

```
"fpr")
plot(roc_KNN_65, main = "ROC curve for KNN Model, K=65", col = "green"
, lwd = 2)
abline(a = 0, b = 1, lwd = 2, lty = 2)
auc_object_KNN_65 <- performance(pred_object_65, measure = "auc")
auc_object_KNN_65
auc_object_KNN_65@y.values[[1]]



#----------- K fold Validation on k  = 65 --------------------#

library(caret)
num_folds <- 10
folds <- createFolds(wbcd_knn$success, k = num_folds)
accuracy <- numeric(num_folds)
sensitivity <- numeric(num_folds)
specificity <- numeric(num_folds)
precision <- numeric(num_folds)
recall <- numeric(num_folds)
f1 <- numeric(num_folds)

# Iterate over each fold
for (i in 1:num_folds) {
  train_indices <- unlist(folds[-i])
  test_indices <- unlist(folds[i])
  wbcd_train <- wbcd_knn_n[train_indices, ]
  wbcd_test <- wbcd_knn_n[test_indices, ]
  wbcd_train_labels <- wbcd_knn[train_indices, "success"]
  wbcd_test_labels <- wbcd_knn[test_indices, "success"]

  knn_model <- knn(train = wbcd_train[, -1], test = wbcd_test[, -1], c
l = wbcd_train_labels, k = 65)
   confusion_matrix <- confusionMatrix(data = factor(knn_model, levels
= levels(factor(wbcd_test_labels))),
                                       reference = factor(wbcd_test_l
abels), positive = "Y")

   accuracy[i] <- confusion_matrix$overall['Accuracy']
   sensitivity[i] <- confusion_matrix$byClass['Sensitivity']
   specificity[i] <- confusion_matrix$byClass['Specificity']
   precision[i] <- confusion_matrix$byClass['Precision']
   recall[i] <- confusion_matrix$byClass['Recall']
   f1[i] <- confusion_matrix$byClass['F1']
}

avg_accuracy <- mean(accuracy)
avg_sensitivity <- mean(sensitivity)
avg_specificity <- mean(specificity)
avg_precision <- mean(precision)
avg_recall <- mean(recall)
avg_f1 <- mean(f1)
```

SVM.R

```
#_-----------------#_-----------------#_-----------------#_-----------
------#_-----------------
#_-----------------  SVM ------------##_-----------------
#_-----------------#_-----------------#_-----------------#_-----------
------#_-----------------
library(gmodels)
library(caret)
library(tidyverse)
library(ROCR)
wbcd_svm <- wbcd
wbcd_svm <- subset(wbcd_svm, select = c("success","hasVideo","rating",
"teamSize",
                                        "hasGithub","hasReddit","capit
al_Raised",
                                        "minInvestment","distributedPe
rcentage",
                                        "is_top_25","days_duration","i
s_Ethereum"))

wbcd_svm$success <- factor(wbcd_svm$success)

smp_size <- floor(0.8 * nrow(wbcd_svm))
# Set Seed so that same sample can be reproduced in future
set.seed(123)
SVM_train_ind <- sample(nrow(wbcd_svm), smp_size)
SVM_train <- wbcd_svm[SVM_train_ind, ]
SVM_test <- wbcd_svm[-SVM_train_ind, ]
SVM_train_labels <- wbcd_svm[SVM_train_ind, 1]
SVM_test_labels <- wbcd_svm[-SVM_train_ind, "success"]
#simple linear SVM
install.packages("kernlab")
library(kernlab)
svm_classifier <- ksvm(success ~ ., data = SVM_train,
                       kernel = "vanilladot", prob.model = TRUE)


#---------------------------- Vanillabot SVM -------------------------
--#
svm_pred <- predict(svm_classifier, select(SVM_test, -success))
SVM_predict_prob <- predict(svm_classifier, select(SVM_test, -success)
, type = "probabilities")
svm_results <- data.frame(actual_type = SVM_test_labels,
                          predict_type = svm_pred,
                          prob_Y = round(SVM_predict_prob[ , 2], 5),
                          prob_N = round(SVM_predict_prob[ , 1], 5))
CrossTable(svm_results$actual_type, svm_results$predict_type,
           prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE)
confusionMatrix(svm_results$actual_type, svm_results$predict_type, pos
itive = "Y", mode ="everything")
```

```
pred_object_svm <- prediction(svm_results$prob_Y, svm_results$actual_t
ype)
roc_svm <- performance(pred_object_svm, measure = "tpr", x.measure = "
fpr")
plot(roc_svm, main = "ROC curve for SVM - Vanilladot ", col = "red", l
wd = 2)
abline(a = 0, b = 1, lwd = 2, lty = 2)
auc_object_SVM <- performance(pred_object_svm, measure = "auc")
auc_SVM <- auc_object_SVM@y.values[[1]]
auc_SVM
agreement <- svm_pred == SVM_test$success
table(agreement)
prop.table(table(agreement))
#Accuracy : 0.6557
```

```{rK-fold Validation for Vanillabot ————————————————————-#}
library(caret) library(C50) # Set seed for reproducibility set.seed(12345)
num_folds <- 10 folds <- createFolds(SVM_train_labels, k = num_folds)
accuracy_SVM <- numeric(num_folds) sensitivity_SVM <- numeric(num_folds)
specificity_SVM <- numeric(num_folds) precision_SVM <- numeric(num_folds)
recall_SVM <- numeric(num_folds) f1_SVM <- numeric(num_folds)

for (i in 1:num_folds) { train_indices <- unlist(folds[-i]) test_indices <-
unlist(folds[i]) fold_train_data <- wbcd_svm[train_indices, ] fold_test_data <-
wbcd_svm[test_indices, ] fold_train_labels <- SVM_train_labels[train_indices]
fold_test_labels <- SVM_train_labels[test_indices] SVM_model <- ksvm(success ~ .,
data = fold_train_data, kernel = “vanilladot”) SVM_predictions <-
predict(SVM_model, newdata = fold_test_data) confusion_matrix_SVM <-
confusionMatrix(data = factor(SVM_predictions, levels =
levels(factor(fold_test_labels))), reference = factor(fold_test_labels), positive = “Y”)

# Store the evaluation metrics for the current fold accuracy_SVM[i] <-
confusion_matrix_dt$overall['Accuracy']$sensitivity_SVM[i] <
-$confusion_matrix_dt$byClass['Sensitivity'] specificity_SVM[i] <-
confusion_matrix_dt$byClass['Specificity']$precision_SVM[i] <
-$confusion_matrix_dt$byClass['Precision'] recall_SVM[i] <-
confusion_matrix_dt$byClass['Recall']$f1_SVM[i] <
-$confusion_matrix_dt$byClass['F1'] }

avg_accuracy <- mean(accuracy_SVM) avg_sensitivity <- mean(sensitivity_SVM)
avg_specificity <- mean(specificity_SVM) avg_precision <- mean(precision_SVM)
avg_recall <- mean(recall_SVM) avg_f1 <- mean(f1_SVM)

#————————————- rbfdot SVM ————————————————————–#

set.seed(123) svm_classifier_rbf <- ksvm(success ~ ., data = SVM_train, kernel =
“rbfdot”, prob.model=TRUE) svm_predictions_rbf <- predict(svm_classifier_rbf,
select(SVM_test, -success)) svm_predictions_rbf_prob <- predict(svm_classifier_rbf,
select(SVM_test, -success), type = “probabilities”) svm_results_rbf <-
data.frame(actual_type = SVM_test_labels, predict_type = svm_predictions_rbf,
prob_Y = round(svm_predictions_rbf_prob[ , 2], 5), prob_N =
round(svm_predictions_rbf_prob[ , 1], 5))
```

CrossTable(svm_results_rbf$actual_type$, $svm_results_rbf$ predict_type, dnn=c('Y', 'N'), prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE) confusionMatrix(svm_results_rbf$actual_type$, $svm_results_rbf$ predict_type, positive = "Y",mode = "everything") pred_object_svm_rbf <- prediction(svm_results_rbf$prob_Y$, $svm_results_rbf$ actual_type) roc_svm_rbf <- performance(pred_object_svm_rbf, measure = "tpr", x.measure = "fpr") plot(roc_svm_rbf, main = "ROC curve for SVM - rbfdot", col = "green", lwd = 2) abline(a = 0, b = 1, lwd = 2, lty = 2) auc_object_SVM_rbf <- performance(pred_object_svm_rbf, measure = "auc") auc_SVM_rbf <- auc_object_SVM@y.values[[1]] auc_SVM_rbf agreement_rbf <- svm_predictions_rbf == SVM_test$success table(agreement_rbf) prop.table(table(agreement_rbf)) #Accuracy : 0.6503

```
```{rK-fold Validation for Rbfdot ----------------------------------
-------#}
library(caret)
library(C50)
# Set seed for reproducibility
set.seed(333345)
num_folds <- 10
folds <- createFolds(SVM_train_labels, k = num_folds)
accuracy_SVM_rbf <- numeric(num_folds)
sensitivity_SVM_rbf <- numeric(num_folds)
specificity_SVM_rbf <- numeric(num_folds)
precision_SVM_rbf <- numeric(num_folds)
recall_SVM_rbf <- numeric(num_folds)
f1_SVM_rbf <- numeric(num_folds)

for (i in 1:num_folds) {
  train_indices <- unlist(folds[-i])
  test_indices <- unlist(folds[i])
  fold_train_data <- wbcd_svm[train_indices, ]
  fold_test_data <- wbcd_svm[test_indices, ]
  fold_train_labels <- SVM_train_labels[train_indices]
  fold_test_labels <- SVM_train_labels[test_indices]
  SVM_model_rbf <- ksvm(success ~ ., data = fold_train_data, kernel =
"rbfdot")
  SVM_predictions_rbf <- predict(SVM_model_rbf, newdata = fold_test_da
ta)
  confusion_matrix_SVM_rbf <- confusionMatrix(data = factor(SVM_predic
tions_rbf, levels = levels(factor(fold_test_labels))),
                                              reference = factor(fold_test
_labels), positive = "Y")

  # Store the evaluation metrics for the current fold
  accuracy_SVM_rbf[i] <- confusion_matrix_SVM_rbf$overall['Accuracy']
  sensitivity_SVM_rbf[i] <- confusion_matrix_SVM_rbf$byClass['Sensitiv
ity']
  specificity_SVM_rbf[i] <- confusion_matrix_SVM_rbf$byClass['Specific
ity']
  precision_SVM_rbf[i] <- confusion_matrix_SVM_rbf$byClass['Precision'
]
  recall_SVM_rbf[i] <- confusion_matrix_SVM_rbf$byClass['Recall']
```
```

```
  f1_SVM_rbf[i] <- confusion_matrix_SVM_rbf$byClass['F1']
}

avg_accuracy <- mean(accuracy_SVM_rbf)
avg_sensitivity <- mean(sensitivity_SVM_rbf)
avg_specificity <- mean(specificity_SVM_rbf)
avg_precision <- mean(precision_SVM_rbf)
avg_recall <- mean(recall_SVM_rbf)
avg_f1 <- mean(f1_SVM_rbf)

#---------------------------- polydot SVM ----------------------------
------------------#

set.seed(123)
svm_classifier_pd <- ksvm(success ~ ., data = SVM_train, kernel = "pol
ydot", prob.model =TRUE)
svm_predictions_pd <- predict(svm_classifier_pd, select(SVM_test, -suc
cess))
svm_predictions_pd_prob <- predict(svm_classifier_pd, select(SVM_test,
-success), type = "probabilities")
svm_results_pd <- data.frame(actual_type = SVM_test_labels,
                             predict_type = svm_predictions_pd,
                             prob_Y = round(svm_predictions_pd_prob[
, 2], 5),
                             prob_N = round(svm_predictions_pd_prob[
, 1], 5))
CrossTable(svm_results_pd$actual_type, svm_results_pd$predict_type, dn
n=c('Y', 'N'),
           prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE)
confusionMatrix(svm_results_pd$actual_type, svm_results_pd$predict_typ
e, positive = "Y",mode = "everything")
pred_object_svm_pd <- prediction(svm_results_pd$prob_Y, svm_results_pd
$actual_type)
roc_svm_pd <- performance(pred_object_svm_pd, measure = "tpr", x.measu
re = "fpr")
plot(roc_svm_pd, main = "ROC curve for SVM - polydot", col = "violet",
lwd = 2)
abline(a = 0, b = 1, lwd = 2, lty = 2)
auc_object_SVM_pd <- performance(pred_object_svm_pd, measure = "auc")
auc_SVM_pd <- auc_object_SVM_pd@y.values[[1]]
auc_SVM_pd
agreement_pd <- svm_predictions_pd == SVM_test$success
table(agreement_pd)
prop.table(table(agreement_pd))
```

```{rK-fold Validation for polydot ——————————————————————-#}
library(caret) library(C50) # Set seed for reproducibility set.seed(333345)
num_folds <- 10 folds <- createFolds(SVM_train_labels, k = num_folds)
accuracy_SVM_pd <- numeric(num_folds) sensitivity_SVM_pd <-
numeric(num_folds) specificity_SVM_pd <- numeric(num_folds) precision_SVM_pd
<- numeric(num_folds) recall_SVM_pd <- numeric(num_folds) f1_SVM_pd <-
numeric(num_folds)
```

for (i in 1:num_folds) { train_indices <- unlist(folds[-i]) test_indices <- unlist(folds[i]) fold_train_data <- wbcd_svm[train_indices,] fold_test_data <- wbcd_svm[test_indices,] fold_train_labels <- SVM_train_labels[train_indices] fold_test_labels <- SVM_train_labels[test_indices] SVM_model_pd <- ksvm(success ~ ., data = fold_train_data, kernel = "polydot") SVM_predictions_pd <- predict(SVM_model_pd, newdata = fold_test_data) confusion_matrix_SVM_pd <- confusionMatrix(data = factor(SVM_predictions_pd, levels = levels(factor(fold_test_labels))), reference = factor(fold_test_labels), positive = "Y")

Store the evaluation metrics for the current fold accuracy_SVM_pd[i] <- confusion_matrix_SVM_pd$overall['Accuracy']sensitivity_SVM_pd[i] <- −confusion_matrix_SVM_pd$byClass['Sensitivity'] specificity_SVM_pd[i] <- confusion_matrix_SVM_pd$byClass['Specificity']precision_SVM_pd[i] <- −confusion_matrix_SVM_pd$byClass['Precision'] recall_SVM_pd[i] <- confusion_matrix_SVM_pd$byClass['Recall']f1_SVM_pd[i] <- −confusion_matrix_SVM_pd$byClass['F1'] }

avg_accuracy <- mean(accuracy_SVM_pd) avg_sensitivity <- mean(sensitivity_SVM_pd) avg_specificity <- mean(specificity_SVM_pd) avg_precision <- mean(precision_SVM_pd) avg_recall <- mean(recall_SVM_pd) avg_f1 <- mean(f1_SVM_pd)

#————————————- tanhdot SVM ————————————————————–#

set.seed(123) svm_classifier_th <- ksvm(success ~ ., data = SVM_train, kernel = "tanhdot", prob.model =TRUE) svm_predictions_th <- predict(svm_classifier_th, select(SVM_test, -success)) svm_predictions_th_prob <- predict(svm_classifier_th, select(SVM_test, -success), type = "probabilities") svm_results_th <- data.frame(actual_type = SVM_test_labels, predict_type = svm_predictions_th, prob_Y = round(svm_predictions_th_prob[, 2], 5), prob_N = round(svm_predictions_th_prob[, 1], 5)) CrossTable(svm_results_th$actual_type, svm_results_th$predict_type, dnn=c('Y', 'N'), prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE) th_confusion_matrix <- confusionMatrix(svm_results_th$actual_type, svm_results_th$predict_type, positive = "Y", mode = "everything") th_confusion_matrix th_confusion_matrix$overall['Accuracy']

pred_object_svm_th <- prediction(svm_results_th$prob_Y, svm_results_th$actual_type) roc_svm_th <- performance(pred_object_svm_th, measure = "tpr", x.measure = "fpr") plot(roc_svm_th, main = "ROC curve for SVM - tanhdot", col = "orange", lwd = 2) abline(a = 0, b = 1, lwd = 2, lty = 2) auc_object_SVM_th <- performance(pred_object_svm_th, measure = "auc") auc_SVM_th <- auc_object_SVM_th@y.values[[1]] auc_SVM_th agreement_th <- svm_predictions_th == SVM_test$success table(agreement_th) prop.table(table(agreement_th))

```
```{rK-fold Validation for tanhdot ----------------------------------
--------#}
library(caret)
library(C50)
# Set seed for reproducibility
set.seed(333345)
```
```

```
num_folds <- 10
folds <- createFolds(SVM_train_labels, k = num_folds)
accuracy_SVM_th <- numeric(num_folds)
sensitivity_SVM_th <- numeric(num_folds)
specificity_SVM_th<- numeric(num_folds)
precision_SVM_th <- numeric(num_folds)
recall_SVM_th <- numeric(num_folds)
f1_SVM_th <- numeric(num_folds)

for (i in 1:num_folds) {
  train_indices <- unlist(folds[-i])
  test_indices <- unlist(folds[i])
  fold_train_data <- wbcd_svm[train_indices, ]
  fold_test_data <- wbcd_svm[test_indices, ]
  fold_train_labels <- SVM_train_labels[train_indices]
  fold_test_labels <- SVM_train_labels[test_indices]
  SVM_model_th <- ksvm(success ~ ., data = fold_train_data, kernel = "
tanhdot")
  SVM_predictions_th<- predict(SVM_model_th, newdata = fold_test_data)
  confusion_matrix_SVM_th <- confusionMatrix(data = factor(SVM_predict
ions_th, levels = levels(factor(fold_test_labels))),
                                             reference = factor(fold_t
est_labels), positive = "Y")

  # Store the evaluation metrics for the current fold
  accuracy_SVM_th[i] <- confusion_matrix_SVM_th$overall['Accuracy']
  sensitivity_SVM_th[i] <- confusion_matrix_SVM_th$byClass['Sensitivit
y']
  specificity_SVM_th[i] <- confusion_matrix_SVM_th$byClass['Specificit
y']
  precision_SVM_th[i] <- confusion_matrix_SVM_th$byClass['Precision']
  recall_SVM_th[i] <- confusion_matrix_SVM_th$byClass['Recall']
  f1_SVM_th[i] <- confusion_matrix_SVM_th$byClass['F1']
}

avg_accuracy <- mean(accuracy_SVM_th)
avg_sensitivity <- mean(sensitivity_SVM_th)
avg_specificity <- mean(specificity_SVM_th)
avg_precision <- mean(precision_SVM_th)
avg_recall <- mean(recall_SVM_th)
avg_f1 <- mean(f1_SVM_th)
```