HarvardX Data Science Professional Certificate - CYO Capstone Project

Abraham Mateos

April 3, 2023

Contents

1 Overview **1**

2 Introduction **1**

3 Aim of the project **2**

4 PROJECT **2**

1 Overview

This project is the solution for the "Choose Your Own" Project requirement, the second part of the Module 9 - Capstone course, inside the HarvardX Data Science Professional Certificate Program. The main target is to devise a credit card fraud detection system, based on a machine learning algorithm, which has been approached by several methods (one of them, the one shared below, is the main useful method). Nowadays, we are absolutely exposed to possible fraud transactions in the Internet through e-commerce and online purchases. For this reason, several international authorities, such as the International Monetary Fund, have settled down for many years the principles of policy procedures to ensure and ease up the process of bank card fraud detection via online accounts' machine learning code. In sum, this project may be a sample / resume of these proceeds which have been carried out on by the main world organizations. For this task, I have used all the techniques and resources learnt throughout all this program materials and courses.

2 Introduction

The credit card fraud detection systems might be one of the main systems any banking entity should have established inside its own software structure. Under certain recent American and British universities research analysis, fraud is one of the major ethical issues in the credit card industry. The main aims are, firstly, to identify the different types of credit card fraud, and, in second place, to review alternative techniques and procedures that have been subject for fraud detection.

The secondary target is to present, compare and analyze recently published discoveries in credit card fraud detection. This project defines common terms in credit card fraud and highlights key statistics and figures in this field. Depending on the type of fraud banks or credit card firms might face, several measures should be adopted and implemented. The proposals made in multiple documents are likely to have beneficial results

and perks in terms of cost savings and time efficiency. The relevance of the application of the techniques reviewed here strikes on shrinking credit card fraud crimes volume. However, there are still ethical issues when genuine credit card clients are unclassified as fraudulent.

3 Aim of the project

The target in this project is to devise a machine learning algorithm approached by two ways or methods. The approach to this problem is based on a specific pathway, since I came up with the linear regression model, but then opted for the decision tree method. Later, after deploying a logistic regression model, we implement a quite unique feature I hope will be welcome by the staff, the artificial neural network. This is a tool used generally to create the links among different features and variables straight forward into a machine learning training set, and quite easy to visualize. And finally, the last step is an extreme gradient boosting machine learning regression model to train, under the Bernouilli distribution of fraud (1)/ not fraud (0), as in the previous approach. After getting this final model to work through iterations, we come up with plotting the AUC by using the own XGB model and both the test and the train dataset: indeed, we obtain slightly different results as you might appreciate in the end.

4 PROJECT

Data exploration, where we start out our way with a cleaning process through the functions learnt through this program courses:

```
library(dplyr)
library(tidyr)
library(tidyverse)
library(data.table)
library(stringr)
library(caTools)
library(rpart)
library(rpart.plot)
library(pROC)
library(gbm)
library(neuralnet)
library(ranger)
library(car)
library(caret)
library(data.table)
dataset <- read.csv("C:/Users/Usuario/Desktop/creditcard.csv")

dim(dataset)
```

```
## [1] 284807     31
```

```
head(dataset, 5)
```

```
##   Time         V1          V2        V3         V4          V5          V6
## 1    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
## 2    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
## 3    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
## 4    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
```

```
## 5    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
##            V7          V8         V9         V10        V11         V12
## 1  0.23959855  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086
## 2 -0.07880298  0.08510165 -0.2554251 -0.16697441  1.6127267  1.06523531
## 3  0.79146096  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369
## 4  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
##          V13        V14        V15        V16        V17         V18        V19
## 1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.2079712  0.02579058  0.4039930
## 2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.1148047 -0.18336127 -0.1457830
## 3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.1099694 -0.12135931 -2.2618571
## 4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.6840928  1.96577500 -1.2326220
## 5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.2370332 -0.03819479  0.8034869
##           V20          V21          V22        V23         V24        V25
## 1  0.25141210 -0.018306778  0.277837576 -0.1104739  0.06692807  0.1285394
## 2 -0.06908314 -0.225775248 -0.638671953  0.1012880 -0.33984648  0.1671704
## 3  0.52497973  0.247998153  0.771679402  0.9094123 -0.68928096 -0.3276418
## 4 -0.20803778 -0.108300452  0.005273597 -0.1903205 -1.17557533  0.6473760
## 5  0.40854236 -0.009430697  0.798278495 -0.1374581  0.14126698 -0.2060096
##          V26          V27         V28 Amount Class
## 1 -0.1891148  0.133558377 -0.02105305 149.62     0
## 2  0.1258945 -0.008983099  0.01472417   2.69     0
## 3 -0.1390966 -0.055352794 -0.05975184 378.66     0
## 4 -0.2219288  0.062722849  0.06145763 123.50     0
## 5  0.5022922  0.219422230  0.21515315  69.99     0
```

```
names(dataset)
```

```
##  [1] "Time"   "V1"     "V2"     "V3"     "V4"     "V5"     "V6"     "V7"
##  [9] "V8"     "V9"     "V10"    "V11"    "V12"    "V13"    "V14"    "V15"
## [17] "V16"    "V17"    "V18"    "V19"    "V20"    "V21"    "V22"    "V23"
## [25] "V24"    "V25"    "V26"    "V27"    "V28"    "Amount" "Class"
```

```
var(dataset$Amount)
```

```
## [1] 62560.07
```

```
summary(dataset$Amount)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##     0.00     5.60    22.00    88.35    77.17 25691.16
```

```
table(dataset$Class)
```

```
##
##      0      1
## 284315    492
```

We start then a whole process of deleting NA values, for those ones which may be remaining inside our dataset or which, by default, may have been converted to:

```
##   Time     V1     V2     V3     V4     V5     V6     V7     V8     V9    V10
##      0      0      0      0      0      0      0      0      0      0      0
##    V11    V12    V13    V14    V15    V16    V17    V18    V19    V20    V21
##      0      0      0      0      0      0      0      0      0      0      0
##    V22    V23    V24    V25    V26    V27    V28 Amount  Class
##      0      0      0      0      0      0      0      0      0
```

```
##    Time         V1          V2          V3         V4          V5          V6
## 1     0 -1.3598071 -0.07278117  2.53634674  1.3781552 -0.33832077  0.46238778
## 2     0  1.1918571  0.26615071  0.16648011  0.4481541  0.06001765 -0.08236081
## 3     1 -1.3583541 -1.34016307  1.77320934  0.3797796 -0.50319813  1.80049938
## 4     1 -0.9662717 -0.18522601  1.79299334 -0.8632913 -0.01030888  1.24720317
## 5     2 -1.1582331  0.87773675  1.54871785  0.4030339 -0.40719338  0.09592146
## 6     2 -0.4259659  0.96052304  1.14110934 -0.1682521  0.42098688 -0.02972755
## 7     4  1.2296576  0.14100351  0.04537077  1.2026127  0.19188099  0.27270812
## 8     7 -0.6442694  1.41796355  1.07438038 -0.4921990  0.94893409  0.42811846
## 9     7 -0.8942861  0.28615720 -0.11319221 -0.2715261  2.66959866  3.72181806
## 10    9 -0.3382618  1.11959338  1.04436655 -0.2221873  0.49936081 -0.24676110
##              V7          V8         V9         V10        V11         V12
## 1   0.239598554  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086
## 2  -0.078802983  0.08510165 -0.2554251 -0.16697441  1.6127267  1.06523531
## 3   0.791460956  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369
## 4   0.237608940  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5   0.592940745 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
## 6   0.476200949  0.26031433 -0.5686714 -0.37140720  1.3412620  0.35989384
## 7  -0.005159003  0.08121294  0.4649600 -0.09925432 -1.4169072 -0.15382583
## 8   1.120631358 -3.80786424  0.6153747  1.24937618 -0.6194678  0.29147435
## 9   0.370145128  0.85108444 -0.3920476 -0.41043043 -0.7051166 -0.11045226
## 10  0.651583206  0.06953859 -0.7367273 -0.36684564  1.0176145  0.83638957
##            V13         V14         V15        V16          V17         V18
## 1  -0.9913898 -0.31116935  1.46817697 -0.4704005  0.207971242  0.02579058
## 2   0.4890950 -0.14377230  0.63555809  0.4639170 -0.114804663 -0.18336127
## 3   0.7172927 -0.16594592  2.34586495 -2.8900832  1.109969379 -0.12135931
## 4   0.5077569 -0.28792375 -0.63141812 -1.0596472 -0.684092786  1.96577500
## 5   1.3458516 -1.11966983  0.17512113 -0.4514492 -0.237033239 -0.03819479
## 6  -0.3580907 -0.13713370  0.51761681  0.4017259 -0.058132823  0.06865315
## 7  -0.7510627  0.16737196  0.05014359 -0.4435868  0.002820512 -0.61198734
## 8   1.7579642 -1.32386522  0.68613250 -0.0761270 -1.222127345 -0.35822157
## 9  -0.2862536  0.07435536 -0.32878305 -0.2100773 -0.499767969  0.11876486
## 10  1.0068435 -0.44352282  0.15021910  0.7394528 -0.540979922  0.47667726
##             V19         V20          V21          V22         V23         V24
## 1   0.40399296  0.25141210 -0.018306778  0.277837576 -0.11047391  0.06692807
## 2  -0.14578304 -0.06908314 -0.225775248 -0.638671953  0.10128802 -0.33984648
## 3  -2.26185710  0.52497973  0.247998153  0.771679402  0.90941226 -0.68928096
## 4  -1.23262197 -0.20803778 -0.108300452  0.005273597 -0.19032052 -1.17557533
## 5   0.80348692  0.40854236 -0.009430697  0.798278495 -0.13745808  0.14126698
## 6  -0.03319379  0.08496767 -0.208253515 -0.559824796 -0.02639767 -0.37142658
## 7  -0.04557504 -0.21963255 -0.167716266 -0.270709726 -0.15410379 -0.78005542
## 8   0.32450473 -0.15674185  1.943465340 -1.015454710  0.05750353 -0.64970901
## 9   0.57032817  0.05273567 -0.073425100 -0.268091632 -0.20423267  1.01159180
## 10  0.45177296  0.20371145 -0.246913937 -0.633752642 -0.12079408 -0.38504993
##             V25         V26          V27          V28 Amount Class
## 1   0.12853936 -0.18911484  0.133558377 -0.021053053 149.62     0
## 2   0.16717040  0.12589453 -0.008983099  0.014724169   2.69     0
```

```
## 3  -0.32764183 -0.13909657 -0.055352794 -0.059751841 378.66     0
## 4   0.64737603 -0.22192884  0.062722849  0.061457629 123.50     0
## 5  -0.20600959  0.50229222  0.219422230  0.215153147  69.99     0
## 6  -0.23279382  0.10591478  0.253844225  0.081080257   3.67     0
## 7   0.75013694 -0.25723685  0.034507430  0.005167769   4.99     0
## 8  -0.41526657 -0.05163430 -1.206921081 -1.085339188  40.80     0
## 9   0.37320468 -0.38415731  0.011747356  0.142404330  93.20     0
## 10 -0.06973305  0.09419883  0.246219305  0.083075649   3.68     0
```

Then we proceed with running the means of the main variables...

```
## [1] 88.34962
```

```
## [1] 0.001727486
```

And then we summarize the resulting dataset after all this data cleaning process:

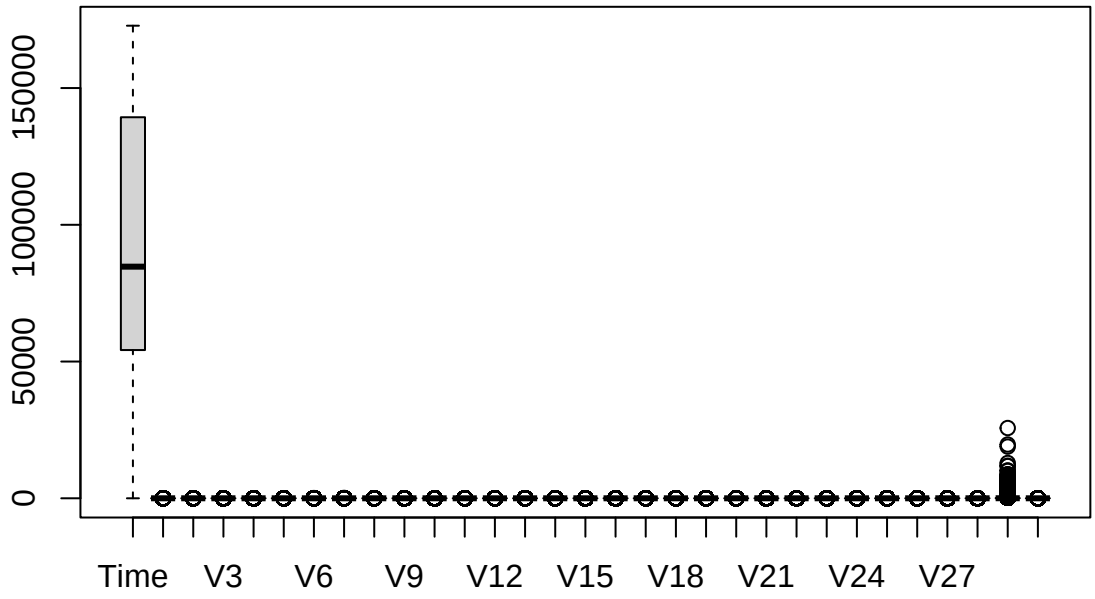
```
##       Time              V1                  V2                  V3
##  Min.   :     0   Min.   :-56.40751   Min.   :-72.71573   Min.   :-48.3256
##  1st Qu.: 54202   1st Qu.: -0.92037   1st Qu.: -0.59855   1st Qu.: -0.8904
##  Median : 84692   Median :  0.01811   Median :  0.06549   Median :  0.1799
##  Mean   : 94814   Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.0000
##  3rd Qu.:139321   3rd Qu.:  1.31564   3rd Qu.:  0.80372   3rd Qu.:  1.0272
##  Max.   :172792   Max.   :  2.45493   Max.   : 22.05773   Max.   :  9.3826
##        V4                  V5                  V6                  V7
##  Min.   :-5.68317   Min.   :-113.74331   Min.   :-26.1605   Min.   :-43.5572
##  1st Qu.:-0.84864   1st Qu.:  -0.69160   1st Qu.: -0.7683   1st Qu.: -0.5541
##  Median :-0.01985   Median :  -0.05434   Median : -0.2742   Median :  0.0401
##  Mean   : 0.00000   Mean   :   0.00000   Mean   :  0.0000   Mean   :  0.0000
##  3rd Qu.: 0.74334   3rd Qu.:   0.61193   3rd Qu.:  0.3986   3rd Qu.:  0.5704
##  Max.   :16.87534   Max.   :  34.80167   Max.   : 73.3016   Max.   :120.5895
##        V8                  V9                 V10                 V11
##  Min.   :-73.21672   Min.   :-13.43407   Min.   :-24.58826   Min.   :-4.79747
##  1st Qu.: -0.20863   1st Qu.: -0.64310   1st Qu.: -0.53543   1st Qu.:-0.76249
##  Median :  0.02236   Median : -0.05143   Median : -0.09292   Median :-0.03276
##  Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.32735   3rd Qu.:  0.59714   3rd Qu.:  0.45392   3rd Qu.: 0.73959
##  Max.   : 20.00721   Max.   : 15.59500   Max.   : 23.74514   Max.   :12.01891
##       V12                 V13                 V14                 V15
##  Min.   :-18.6837   Min.   :-5.79188   Min.   :-19.2143   Min.   :-4.49894
##  1st Qu.: -0.4056   1st Qu.:-0.64854   1st Qu.: -0.4256   1st Qu.:-0.58288
##  Median :  0.1400   Median :-0.01357   Median :  0.0506   Median : 0.04807
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :  0.0000   Mean   : 0.00000
##  3rd Qu.:  0.6182   3rd Qu.: 0.66251   3rd Qu.:  0.4931   3rd Qu.: 0.64882
##  Max.   :  7.8484   Max.   : 7.12688   Max.   : 10.5268   Max.   : 8.87774
##       V16                 V17                 V18
##  Min.   :-14.12985   Min.   :-25.16280   Min.   :-9.498746
##  1st Qu.: -0.46804   1st Qu.: -0.48375   1st Qu.:-0.498850
##  Median :  0.06641   Median : -0.06568   Median :-0.003636
##  Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.000000
##  3rd Qu.:  0.52330   3rd Qu.:  0.39968   3rd Qu.: 0.500807
##  Max.   : 17.31511   Max.   :  9.25353   Max.   : 5.041069
##       V19                 V20                 V21
```

```
##  Min.   :-7.213527   Min.   :-54.49772   Min.   :-34.83038
##  1st Qu.:-0.456299   1st Qu.: -0.21172   1st Qu.: -0.22839
##  Median : 0.003735   Median : -0.06248   Median : -0.02945
##  Mean   : 0.000000   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.: 0.458949   3rd Qu.:  0.13304   3rd Qu.:  0.18638
##  Max.   : 5.591971   Max.   : 39.42090   Max.   : 27.20284
##       V22                 V23                 V24
##  Min.   :-10.933144   Min.   :-44.80774   Min.   :-2.83663
##  1st Qu.: -0.542350   1st Qu.: -0.16185   1st Qu.:-0.35459
##  Median :  0.006782   Median : -0.01119   Median : 0.04098
##  Mean   :  0.000000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.528554   3rd Qu.:  0.14764   3rd Qu.: 0.43953
##  Max.   : 10.503090   Max.   : 22.52841   Max.   : 4.58455
##       V25                 V26                 V27
##  Min.   :-10.29540   Min.   :-2.60455   Min.   :-22.565679
##  1st Qu.: -0.31715   1st Qu.:-0.32698   1st Qu.: -0.070840
##  Median :  0.01659   Median :-0.05214   Median :  0.001342
##  Mean   :  0.00000   Mean   : 0.00000   Mean   :  0.000000
##  3rd Qu.:  0.35072   3rd Qu.: 0.24095   3rd Qu.:  0.091045
##  Max.   :  7.51959   Max.   : 3.51735   Max.   : 31.612198
##       V28                Amount              Class
##  Min.   :-15.43008   Min.   :    0.00   Min.   :0.000000
##  1st Qu.: -0.05296   1st Qu.:    5.60   1st Qu.:0.000000
##  Median :  0.01124   Median :   22.00   Median :0.000000
##  Mean   :  0.00000   Mean   :   88.35   Mean   :0.001728
##  3rd Qu.:  0.07828   3rd Qu.:   77.17   3rd Qu.:0.000000
##  Max.   : 33.84781   Max.   :25691.16   Max.   :1.000000
```

```
##   Time         V1          V2        V3         V4          V5          V6
## 1    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
## 2    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
## 3    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
## 4    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
## 5    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
##            V7          V8         V9         V10        V11         V12
## 1  0.23959855  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086
## 2 -0.07880298  0.08510165 -0.2554251 -0.16697441  1.6127267  1.06523531
## 3  0.79146096  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369
## 4  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
##          V13        V14        V15        V16        V17         V18        V19
## 1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.2079712  0.02579058  0.4039930
## 2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.1148047 -0.18336127 -0.1457830
## 3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.1099694 -0.12135931 -2.2618571
## 4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.6840928  1.96577500 -1.2326220
## 5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.2370332 -0.03819479  0.8034869
##           V20          V21          V22        V23         V24        V25
## 1  0.25141210 -0.018306778  0.277837576 -0.1104739  0.06692807  0.1285394
## 2 -0.06908314 -0.225775248 -0.638671953  0.1012880 -0.33984648  0.1671704
## 3  0.52497973  0.247998153  0.771679402  0.9094123 -0.68928096 -0.3276418
## 4 -0.20803778 -0.108300452  0.005273597 -0.1903205 -1.17557533  0.6473760
## 5  0.40854236 -0.009430697  0.798278495 -0.1374581  0.14126698 -0.2060096
##          V26          V27         V28 Amount Class
## 1 -0.1891148  0.133558377 -0.02105305 149.62     0
```

```
## 2  0.1258945 -0.008983099  0.01472417   2.69     0
## 3 -0.1390966 -0.055352794 -0.05975184 378.66     0
## 4 -0.2219288  0.062722849  0.06145763 123.50     0
## 5  0.5022922  0.219422230  0.21515315  69.99     0
```

And we plot it as a content review:



Winsorization is quite useful at this point. When an outlier is negatively impacting a model results, it is possible to replace this with a less extreme maximum value. In Winsorizing, values located out of a predetermined percentile range of the data are identified and set to this percentile. Rather, winsorizing a vector means a predefined quantum of the smallest and/or the largest values is replaced instead by less extreme values. Thus, the substitution values are the most extreme retained values in reference to those ones above 95th percentile. Data wrangling, using mainly head and scale functions which will let me scale or grow up the chosen amount to a more realistic sample to analyze:

```
install.packages("robustHD", repos="https://cran.rstudio.com")
require(robustHD)
sum(df$Amount > quantile(df$Amount, .95))
```

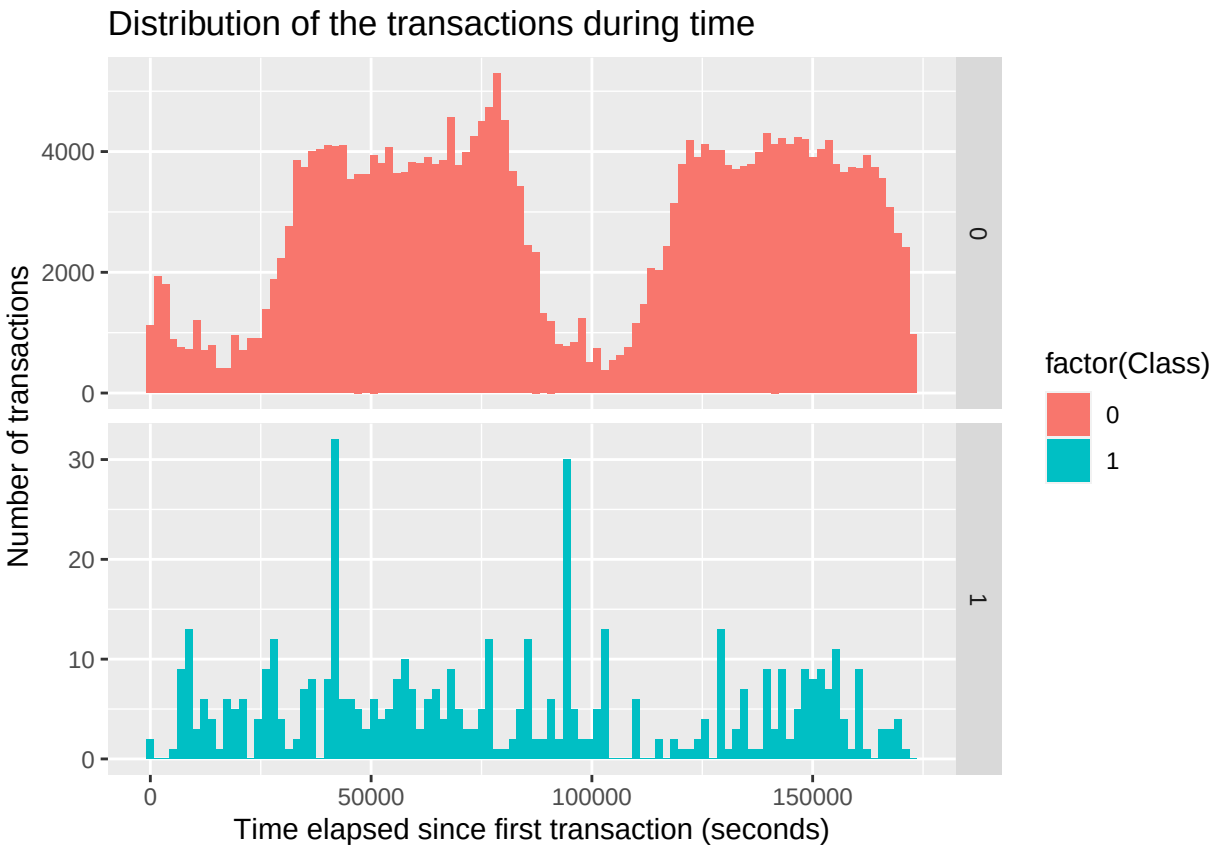
```
## [1] 14232
```

```
df <- df %>% mutate(wins_total_amount = winsorize(Amount))
head(df, 5)
```

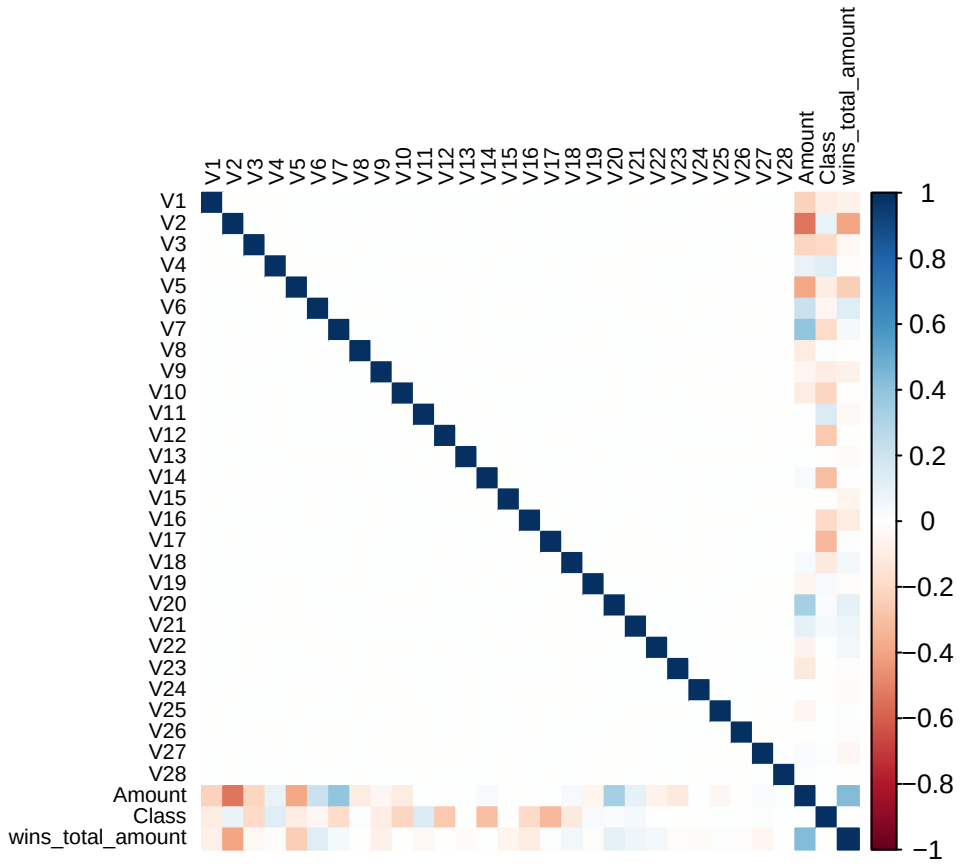
```
##   Time         V1          V2        V3         V4          V5          V6
```

```
## 1    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
## 2    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
## 3    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
## 4    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
## 5    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
##           V7          V8         V9         V10        V11         V12
## 1  0.23959855  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086
## 2 -0.07880298  0.08510165 -0.2554251 -0.16697441  1.6127267  1.06523531
## 3  0.79146096  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369
## 4  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
##          V13        V14        V15        V16        V17         V18        V19
## 1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.2079712  0.02579058  0.4039930
## 2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.1148047 -0.18336127 -0.1457830
## 3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.1099694 -0.12135931 -2.2618571
## 4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.6840928  1.96577500 -1.2326220
## 5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.2370332 -0.03819479  0.8034869
##           V20          V21          V22        V23         V24        V25
## 1  0.25141210 -0.018306778  0.277837576 -0.1104739  0.06692807  0.1285394
## 2 -0.06908314 -0.225775248 -0.638671953  0.1012880 -0.33984648  0.1671704
## 3  0.52497973  0.247998153  0.771679402  0.9094123 -0.68928096 -0.3276418
## 4 -0.20803778 -0.108300452  0.005273597 -0.1903205 -1.17557533  0.6473760
## 5  0.40854236 -0.009430697  0.798278495 -0.1374581  0.14126698 -0.2060096
##          V26          V27         V28 Amount Class wins_total_amount
## 1 -0.1891148  0.133558377 -0.02105305 149.62     0          81.95634
## 2  0.1258945 -0.008983099  0.01472417   2.69     0           2.69000
## 3 -0.1390966 -0.055352794 -0.05975184 378.66     0          81.95634
## 4 -0.2219288  0.062722849  0.06145763 123.50     0          81.95634
## 5  0.5022922  0.219422230  0.21515315  69.99     0          69.99000
```

Data visualization: We can plot by using histograms the distribution of transactions during time running:



And analyze the correlation rate among the variables:



Data wrangling: By using head and scale functions, I will be able to scale or grow up the selected amount to a more realistic sample to be analyzed:

```
##           V1          V2        V3         V4          V5          V6
## 1 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
## 2  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
## 3 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
## 4 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
## 5 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
## 6 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755
##            V7          V8         V9         V10        V11         V12
## 1  0.23959855  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086
## 2 -0.07880298  0.08510165 -0.2554251 -0.16697441  1.6127267  1.06523531
## 3  0.79146096  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369
## 4  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
## 6  0.47620095  0.26031433 -0.5686714 -0.37140720  1.3412620  0.35989384
##           V13        V14        V15        V16         V17         V18
## 1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.20797124  0.02579058
## 2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.11480466 -0.18336127
## 3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.10996938 -0.12135931
## 4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279  1.96577500
## 5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.23703324 -0.03819479
## 6 -0.3580907 -0.1371337  0.5176168  0.4017259 -0.05813282  0.06865315
##           V19         V20          V21          V22         V23          V24
## 1  0.40399296  0.25141210 -0.018306778  0.277837576 -0.11047391  0.06692807
```

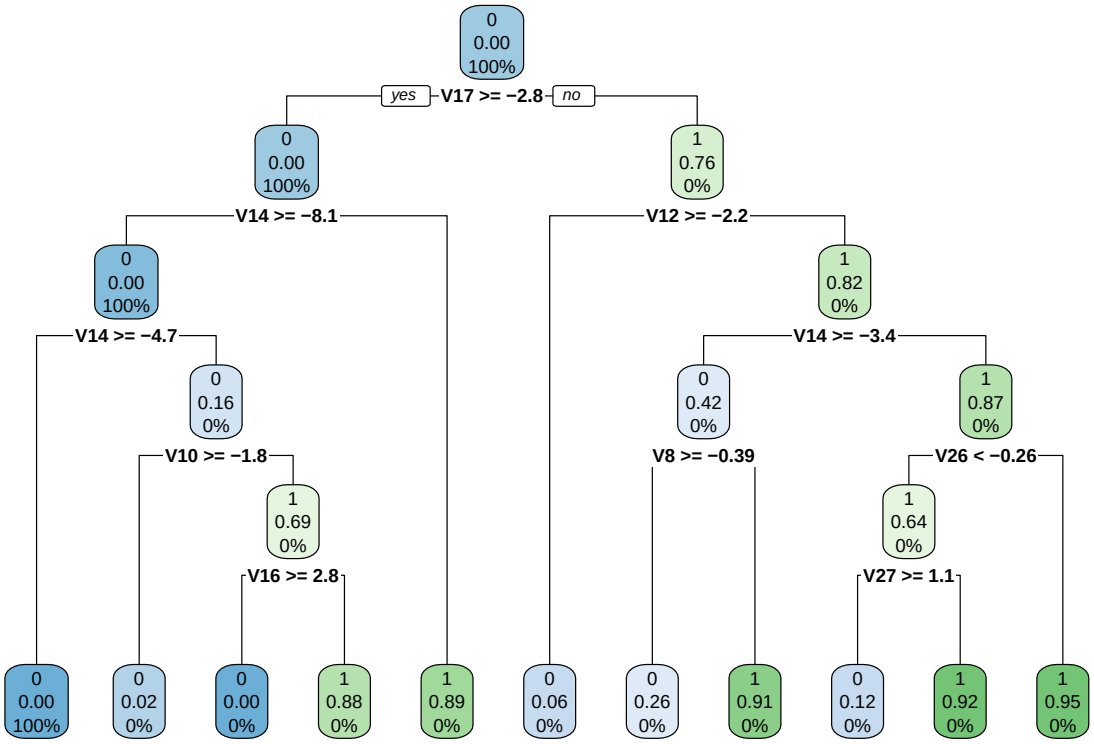
```
## 2 -0.14578304 -0.06908314 -0.225775248 -0.638671953  0.10128802 -0.33984648
## 3 -2.26185710  0.52497973  0.247998153  0.771679402  0.90941226 -0.68928096
## 4 -1.23262197 -0.20803778 -0.108300452  0.005273597 -0.19032052 -1.17557533
## 5  0.80348692  0.40854236 -0.009430697  0.798278495 -0.13745808  0.14126698
## 6 -0.03319379  0.08496767 -0.208253515 -0.559824796 -0.02639767 -0.37142658
##          V25        V26          V27         V28      Amount Class
## 1  0.1285394 -0.1891148  0.133558377 -0.02105305  0.24496383     0
## 2  0.1671704  0.1258945 -0.008983099  0.01472417 -0.34247394     0
## 3 -0.3276418 -0.1390966 -0.055352794 -0.05975184  1.16068389     0
## 4  0.6473760 -0.2219288  0.062722849  0.06145763  0.14053401     0
## 5 -0.2060096  0.5022922  0.219422230  0.21515315 -0.07340321     0
## 6 -0.2327938  0.1059148  0.253844225  0.08108026 -0.33855582     0
##   wins_total_amount
## 1          81.95634
## 2           2.69000
## 3          81.95634
## 4          81.95634
## 5          69.99000
## 6           3.67000
```

We now proceed to model our dataset, by splitting it up in the partition of the train set and the test set:

```
## [1] 199364     31
```

```
## [1] 85443    31
```

We start our algorithms roadmap with a decision tree model, where as we have been taught through the program, we can achieve a partition of the credit card dataset, classify the type of the retrieved data and solve the linear regression, as we are managing continuous input and output data:



I think it may be appropriate to implement now a logistic regression model, by making use of the class and test data, and the binomial distribution specification. With the concerning library for the ROC (Receiver Operating Characteristic), we then make the predictions and include them into our validation set (test set) and devise the respective visualization with the 'roc' function:

```
lr_model <- glm(Class~., train, family=binomial())

summary(lr_model)
```

```
##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.4655  -0.0296  -0.0194  -0.0124   4.1894
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -8.6313026  0.2013956 -42.857  < 2e-16 ***
## V1                0.0752351  0.0454484   1.655 0.097845 .
## V2                0.0038569  0.0610235   0.063 0.949605
## V3               -0.0044668  0.0512083  -0.087 0.930490
## V4                0.6639949  0.0786365   8.444  < 2e-16 ***
## V5                0.0931429  0.0729881   1.276 0.201907
## V6               -0.1405155  0.0869175  -1.617 0.105953
```

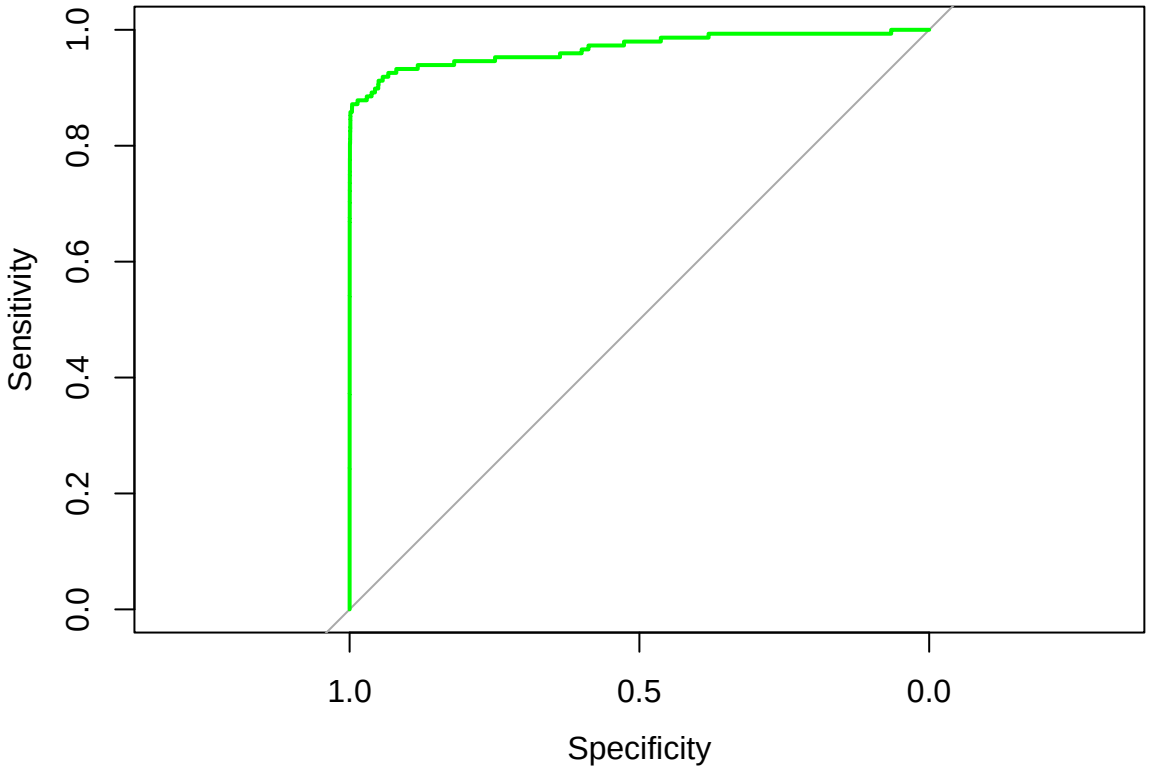
```
## V7                -0.1344989  0.0692662  -1.942 0.052165 .
## V8                -0.1435432  0.0353424  -4.062 4.88e-05 ***
## V9                -0.3789974  0.1196115  -3.169 0.001532 **
## V10               -0.7439183  0.0990413  -7.511 5.86e-14 ***
## V11               -0.0485694  0.0897454  -0.541 0.588376
## V12                0.1214408  0.0997819   1.217 0.223581
## V13               -0.3525109  0.0977923  -3.605 0.000313 ***
## V14               -0.5583445  0.0708427  -7.881 3.24e-15 ***
## V15               -0.1161922  0.0984089  -1.181 0.237718
## V16               -0.1409224  0.1411941  -0.998 0.318243
## V17                0.0300958  0.0770088   0.391 0.695937
## V18               -0.0782706  0.1440762  -0.543 0.586952
## V19                0.0765777  0.1078875   0.710 0.477833
## V20               -0.4721932  0.0820280  -5.756 8.59e-09 ***
## V21                0.4152254  0.0671502   6.184 6.27e-10 ***
## V22                0.7277348  0.1501965   4.845 1.26e-06 ***
## V23               -0.0811573  0.0610884  -1.329 0.184005
## V24                0.2486966  0.1762775   1.411 0.158296
## V25               -0.1048963  0.1470020  -0.714 0.475492
## V26                0.0945119  0.2137898   0.442 0.658432
## V27               -0.8488366  0.1158790  -7.325 2.39e-13 ***
## V28               -0.3147532  0.0861529  -3.653 0.000259 ***
## Amount             0.3033112  0.0923522   3.284 0.001022 **
## wins_total_amount -0.0004037  0.0030880  -0.131 0.895990
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5064.6  on 199363  degrees of freedom
## Residual deviance: 1625.8  on 199333  degrees of freedom
## AIC: 1687.8
##
## Number of Fisher Scoring iterations: 11
```

```
predicted_2 <- predict(lr_model, test, probability = TRUE)

auc_curve <- roc(test$Class, predicted_2, plot= TRUE, col="green")
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```



After the LR Model, for its own nature and reliance on the human nerves network and its similarity to this problem's casuistry, it would be a good step to build up an Artificial Neural Network, where we should analyze our train set into a neural model, so that we can create a result which would fit the human mind. I have fulfilled this target by not reading the data as a linear description, but in a networking way, by linking and making sense among info nods with other data related to the same variable. Later, that training result is integrated into our test set, in order to come out with a result in a default case of 0.5 - 1:

As the last step, we take up an Extreme Gradient Boosting regression model (XGB model), which I think is the best step in order to come out with the best iteration among variables in the regression at the fitting process:

```
set.seed(9560)
install.packages("drat", repos="https://cran.rstudio.com")
drat:::addRepo("dmlc")
install.packages("xgboost", repos="http://dmlc.ml/drat/", type = "source")
require(xgboost)

labels <- train$Class
```

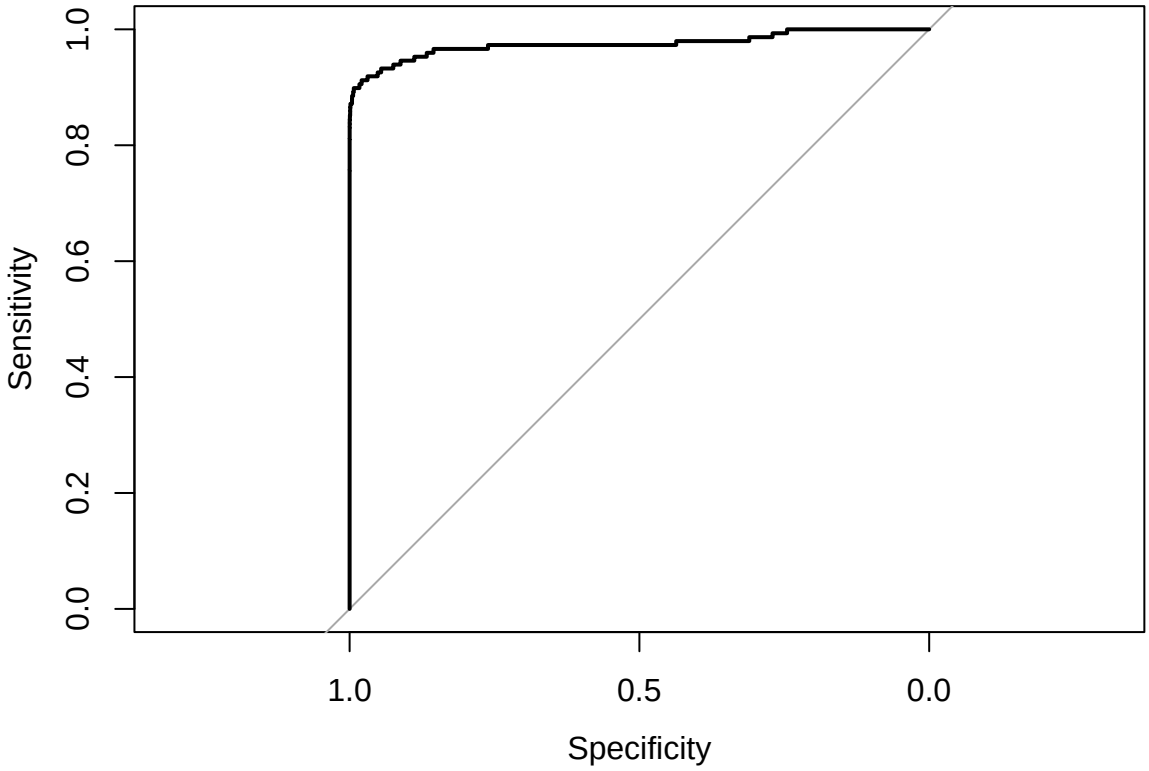
We fit the model and configure its parameters:

We then calculate and plot the Area Under Curve for the ROC by using both the mostly used test dataset for this target and the train dataset as well, so that we can retrieve more realistic information on the decisions approach results:

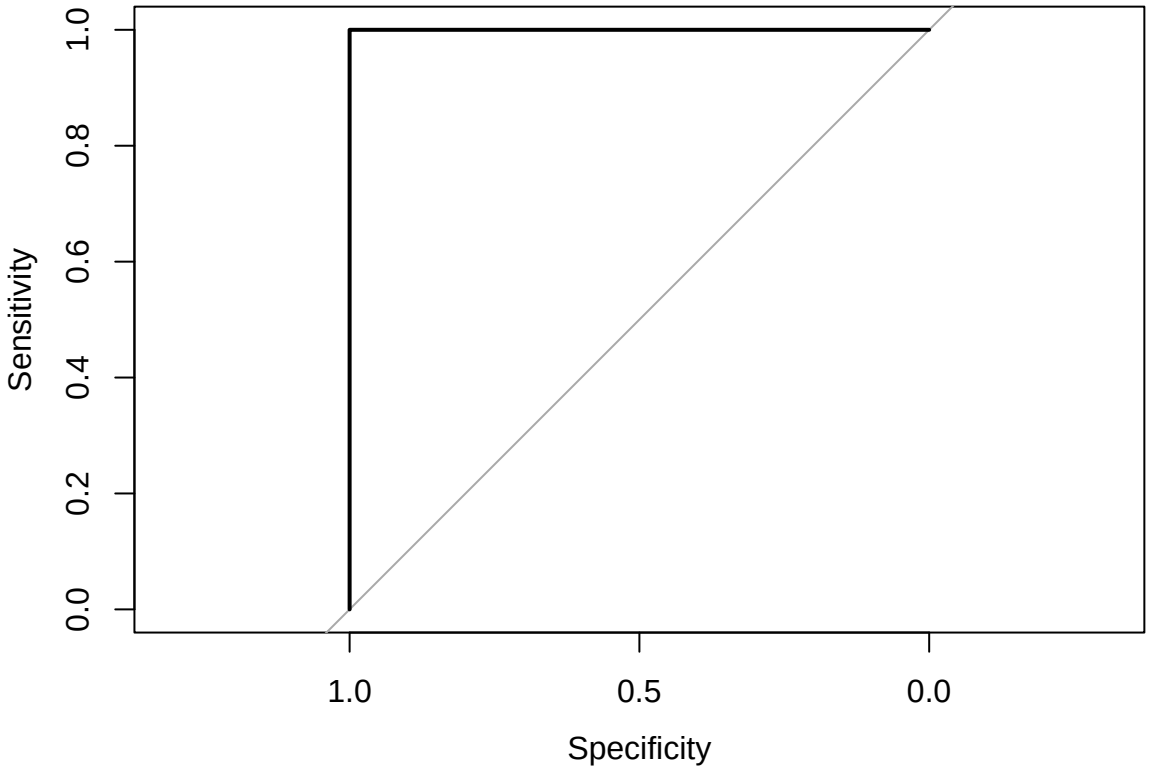
```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases

##
## Call:
## roc.default(response = test$Class, predictor = xgb_pred, plot = TRUE)
##
## Data: xgb_pred in 85295 controls (test$Class 0) < 148 cases (test$Class 1).
## Area under the curve: 0.9748
```




```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = train$Class, predictor = xgb_pred, plot = TRUE)
##
## Data: xgb_pred in 199020 controls (train$Class 0) < 344 cases (train$Class 1).
## Area under the curve: 1
```

Conclusion: as we can appreciate through this approach, we have settled down some core and very well-known ML principles, but have arranged them in an order and manner (and as well along with some other intrinsic and previous tips) which have let me get that final AUC score after running the XGB model.

Nevertheless, needless to say it is quite essential to first-off implement the tree model method, so that we can then create an artificial neural network, and therefore, connect human mind features to credit cards data, gathering and collecting different data into each "variable classification". To sum up, we carry out an extreme gradient boosting regression model, in order to apply different values to the different parameters included in our final training model, and then to figure out the best iteration.