# Credit Card Fraud Detection Project

Sho Nakamura

2023-07-21

## Introduction

For this project, I decided to use a data set called "Credit Card Fraud Detection" from Kaggle and analyze the data. As the name of the dataset suggests, it does data analysis to detect credit card fraud. The reason why I chose this dataset is that there are many people who create more credit cards than necessary when they become working adults. As the number of credit cards increases, the probability of fraudulent use increases. Therefore, I wanted to know how much fraudulent use is detected. I also wanted to create a model that would detect fraudulent use with a high probability.

## Goal of Project

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. Therefore, a model with a high detection rate of fraudulent use is created. At least two different models or algorithms must be used, with at least one being more advanced than linear or logistic regression for prediction problems.

## Method

Describe the process of data explosion, data visualization and modeling.

## Import Library

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table
##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(corrplot)) install.packages("corrplot", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: corrplot
## corrplot 0.92 loaded
```

```
if(!require(caTools)) install.packages("caTools", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caTools
```

```
if(!require(pROC)) install.packages("pROC", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: pROC
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
if(!require(rpart)) install.packages("rpart", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: rpart
```

```
if(!require(rpart.plot)) install.packages("rpart.plot", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: rpart.plot
```

```
if(!require(gbm)) install.packages("gbm", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: gbm
## Loaded gbm 2.1.8.1
```

```
if(!require(ranger)) install.packages("ranger", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: ranger
```

```
if(!require(Epi)) install.packages("Epi", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: Epi
```

```
library(tidyverse)
library(ggplot2)
library(caret)
library(data.table)
library(dplyr)
library(corrplot)
library(caTools)
library(pROC)
library(rpart)
library(rpart.plot)
library(gbm, quietly=TRUE)
library(ranger)
library(Epi)

# Set seed
set.seed(1996)
```

## Load Dataset

```
# Load dataset
credit_card <- read.csv("creditcard.csv")
```

## Data exploration and visualization

```r
# Check dataset
head(credit_card, 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
```

```r
tail(credit_card, 5)
```

```
##          Time          V1          V2         V3         V4          V5
## 284803 172786 -11.8811179 10.07178497 -9.8347835 -2.0666557 -5.36447278
## 284804 172787  -0.7327887 -0.05508049  2.0350297 -0.7385886  0.86822940
## 284805 172788   1.9195650 -0.30125385 -3.2496398 -0.5578281  2.63051512
## 284806 172788  -0.2404400  0.53048251  0.7025102  0.6897992 -0.37796113
## 284807 172792  -0.5334125 -0.18973334  0.7033374 -0.5062712 -0.01254568
##                V6         V7         V8        V9        V10        V11
## 284803 -2.6068373 -4.9182154  7.3053340 1.9144283  4.3561704 -1.5931053
## 284804  1.0584153  0.0243297  0.2948687 0.5848000 -0.9759261 -0.1501888
## 284805  3.0312601 -0.2968265  0.7084172 0.4324540 -0.4847818  0.4116137
## 284806  0.6237077 -0.6861800  0.6791455 0.3920867 -0.3991257 -1.9338488
## 284807 -0.6496167  1.5770063 -0.4146504 0.4861795 -0.9154266 -1.0404583
##                V12        V13         V14         V15        V16         V17
## 284803  2.71194079 -0.6892556  4.62694203 -0.92445871  1.1076406  1.99169111
## 284804  0.91580191  1.2147558 -0.67514296  1.16493091 -0.7117573 -0.02569286
## 284805  0.06311886 -0.1836987 -0.51060184  1.32928351  0.1407160  0.31350179
```

```
## 284806 -0.96288614 -1.0420817  0.44962444  1.96256312 -0.6085771  0.50992846
## 284807 -0.03151305 -0.1880929 -0.08431647  0.04133346 -0.3026201 -0.66037665
##               V18        V19        V20       V21        V22         V23
## 284803  0.5106323 -0.6829197 1.47582913 0.2134541 0.1118637  1.01447990
## 284804 -1.2211789 -1.5455561 0.05961590 0.2142053 0.9243836  0.01246304
## 284805  0.3956525 -0.5772518 0.00139597 0.2320450 0.5782290 -0.03750086
## 284806  1.1139806  2.8978488 0.12743352 0.2652449 0.8000487 -0.16329794
## 284807  0.1674299 -0.2561169 0.38294810 0.2610573 0.6430784  0.37677701
##                 V24        V25        V26          V27         V28 Amount Class
## 284803 -0.509348453  1.4368069  0.2500343  0.943651172  0.82373096   0.77     0
## 284804 -1.016225669 -0.6066240 -0.3952551  0.068472470 -0.05352739  24.79     0
## 284805  0.640133881  0.2657455 -0.0873706  0.004454772 -0.02656083  67.88     0
## 284806  0.123205244 -0.5691589  0.5466685  0.108820735  0.10453282  10.00     0
## 284807  0.008797379 -0.4736487 -0.8182671 -0.002415309  0.01364891 217.00     0
```

```
names(credit_card)
```

```
##  [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"
```

```
summary(credit_card)
```

```
##       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.:139320   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
```

```
# Check fraud counts
table(credit_card$Class)
```

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

```
# Check proportion of classes
prop.table(table(credit_card$Class))
```

```
##
##           0           1
## 0.998272514 0.001727486
```

The probability of credit fraud is fairly low.

```r
# Check the missing values
colSums(is.na(credit_card))
```

```
##   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
```

There are no missing values.

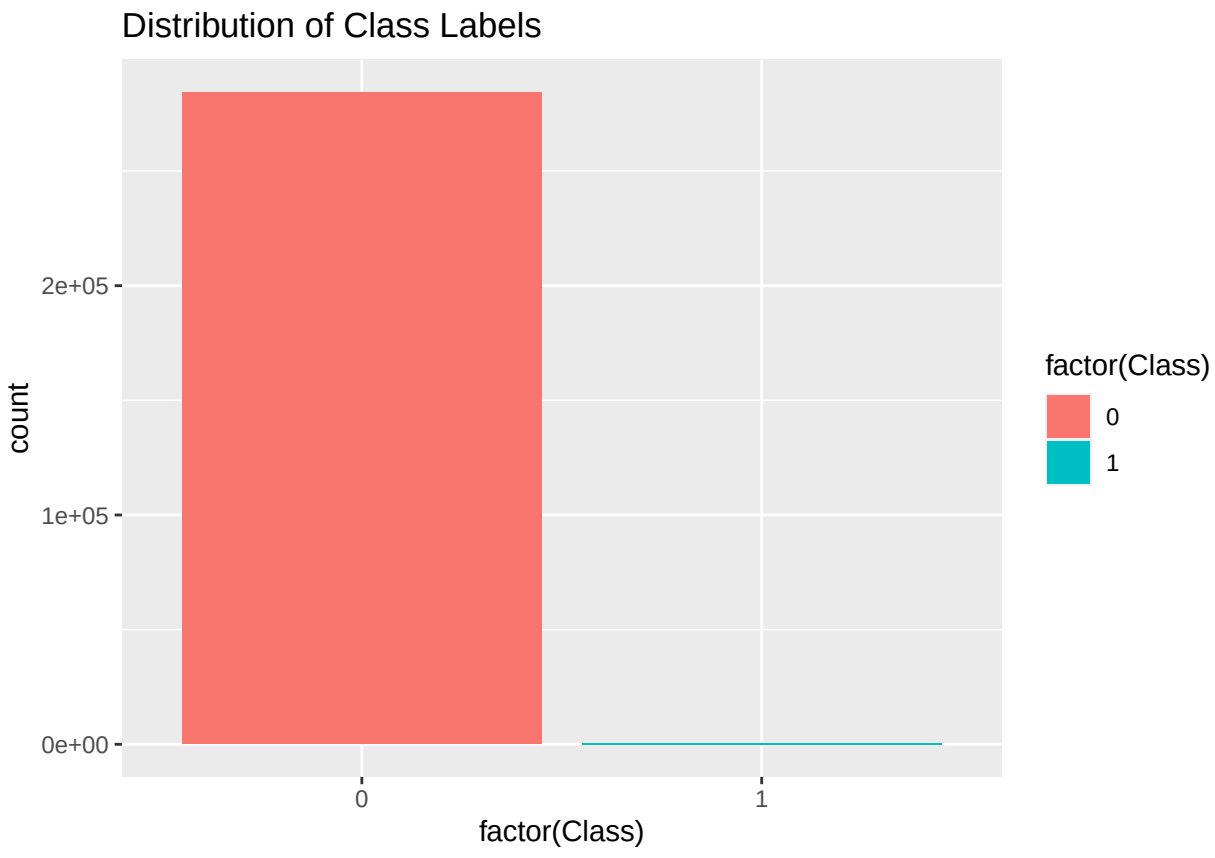
```r
var(credit_card$Amount)
```

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

```r
sd(credit_card$Amount)
```

```
## [1] 250.1201
```

```r
# Plot distribution of class labels
credit_card %>%
  ggplot(aes(x = factor(Class), fill = factor(Class))) +
  geom_bar() +
  ggtitle("Distribution of Class Labels")
```

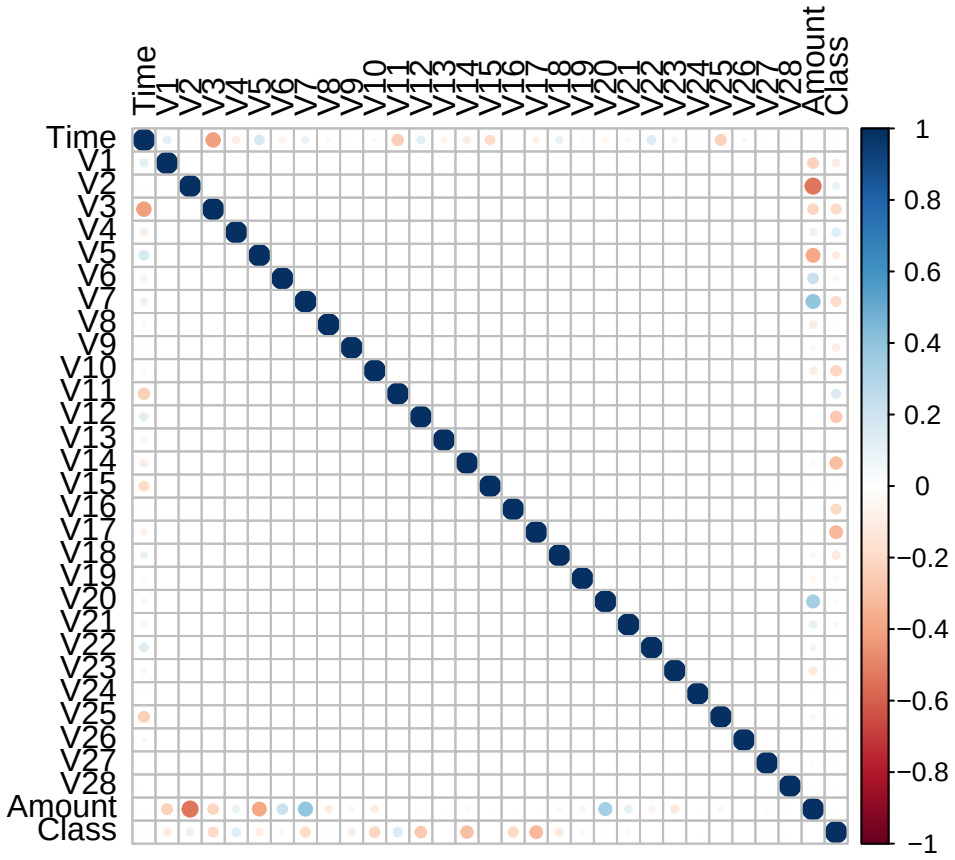


It turns out that fraudulent use of credit cards is rare.

```
# Plot distribution of time of transaction by class
credit_card %>%
  ggplot(aes(x = Time, fill = factor(Class))) +
  geom_histogram(bins = 100) +
  ggtitle("Distribution of time of transaction by class") +
  labs(x = "Time (seconds)", y = "Number of Transactions") +
  facet_grid(Class ~ ., scales = 'free_y')
```


About two times, the time when the credit card was fraudulently used was concentrated. However, since it is unknown what happened from time, the element of time is unlikely to be used.

```
# Plot correlation
corr <- cor(credit_card, use = "pairwise.complete.obs")
corrplot(corr, tl.col = "black")
```




Most of the elements seem unimportant.

## Build Model

```
# Create dataset
credit_card$Amount <- scale(credit_card$Amount)
data <- credit_card[, -1]
head(data)
```

```
##           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
```

```
# Split train and test data
split <- sample.split(data$Class, SplitRatio = 0.80)
train <- subset(data, split == TRUE)
test <- subset(data, split == FALSE)
dim(train)
```

```
## [1] 227846     30
```

```
dim(test)
```

```
## [1] 56961    30
```

### Logistic Regression Model

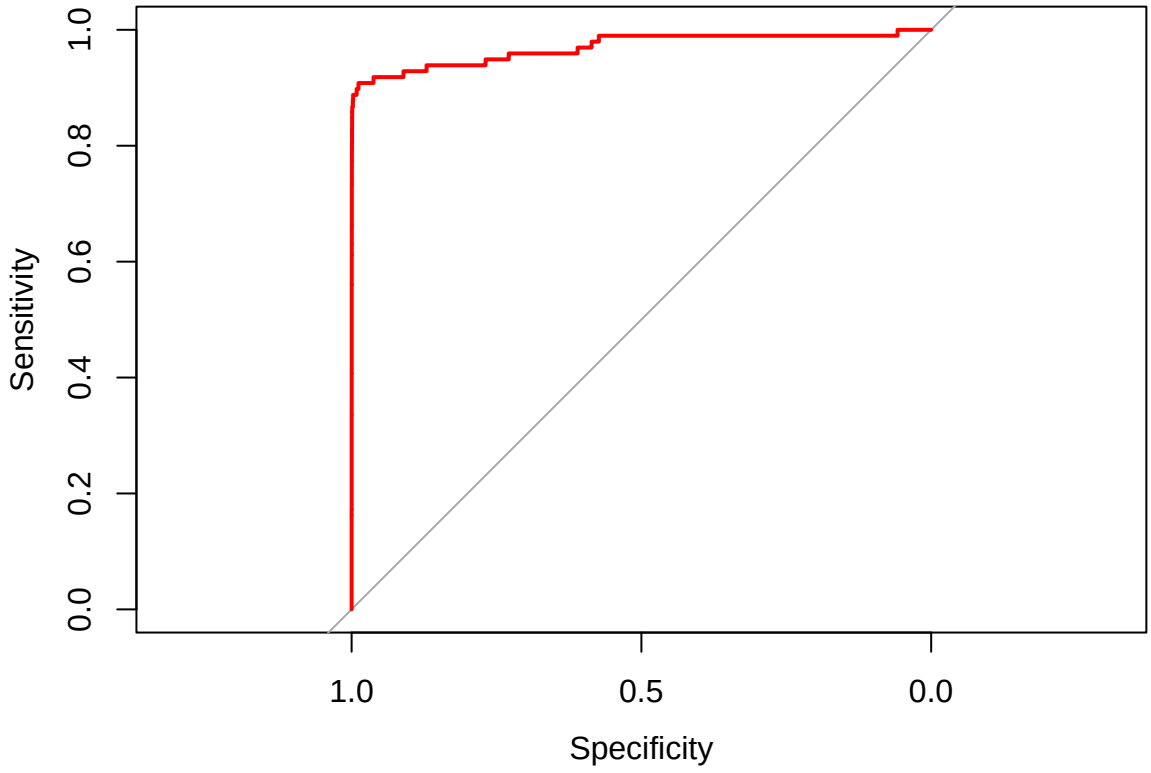
```
## Logistic_Regression_Model ##
log_model <- glm(Class ~ ., train, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
log_pred <- predict(log_model, test, probability = TRUE)
roc(test$Class, log_pred, plot = TRUE, col = "red")
```

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

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



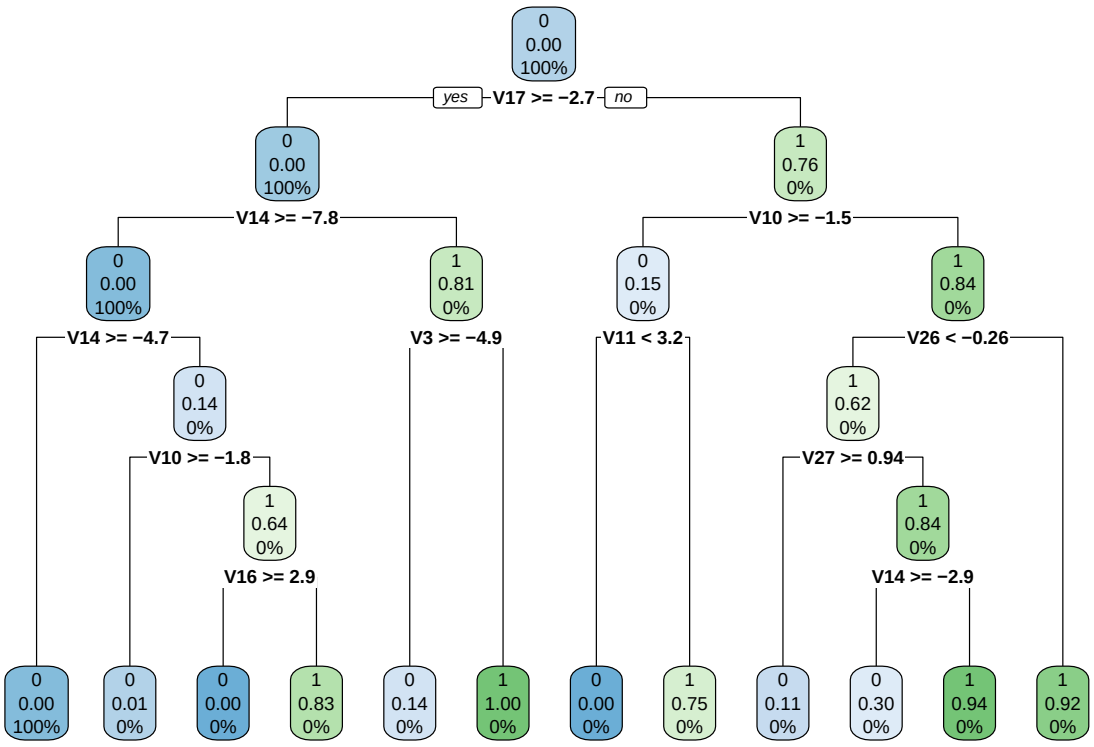

```
##
## Call:
## roc.default(response = test$Class, predictor = log_pred, plot = TRUE,     col = "red")
##
## Data: log_pred in 56863 controls (test$Class 0) < 98 cases (test$Class 1).
## Area under the curve: 0.9696
```

```
print(roc(test$Class, log_pred, col = "red"))
```

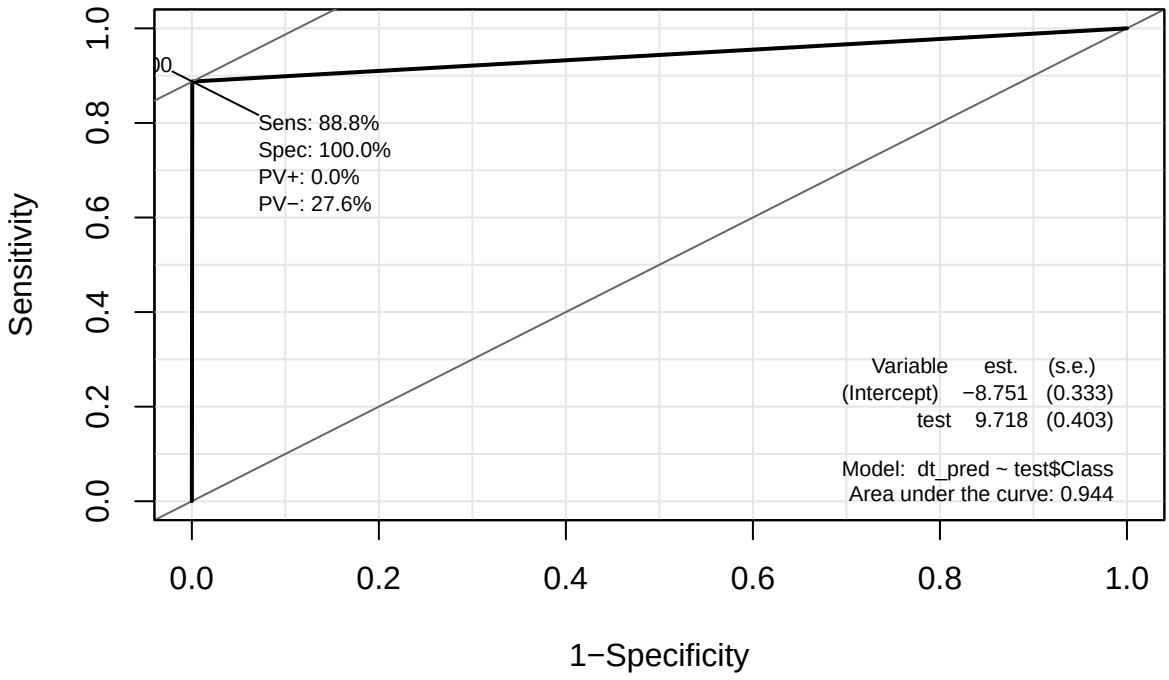
```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

##
## Call:
## roc.default(response = test$Class, predictor = log_pred, col = "red")
##
## Data: log_pred in 56863 controls (test$Class 0) < 98 cases (test$Class 1).
## Area under the curve: 0.9696
```

**Decision Tree Model**

```
## Decision Tree ##
dt_model <- rpart(Class ~ ., train, method = "class")
dt_pred <- predict(dt_model, test, type = "class")
rpart.plot(dt_model)
```




```
ROC(test$Class, dt_pred, plot = "ROC")
```

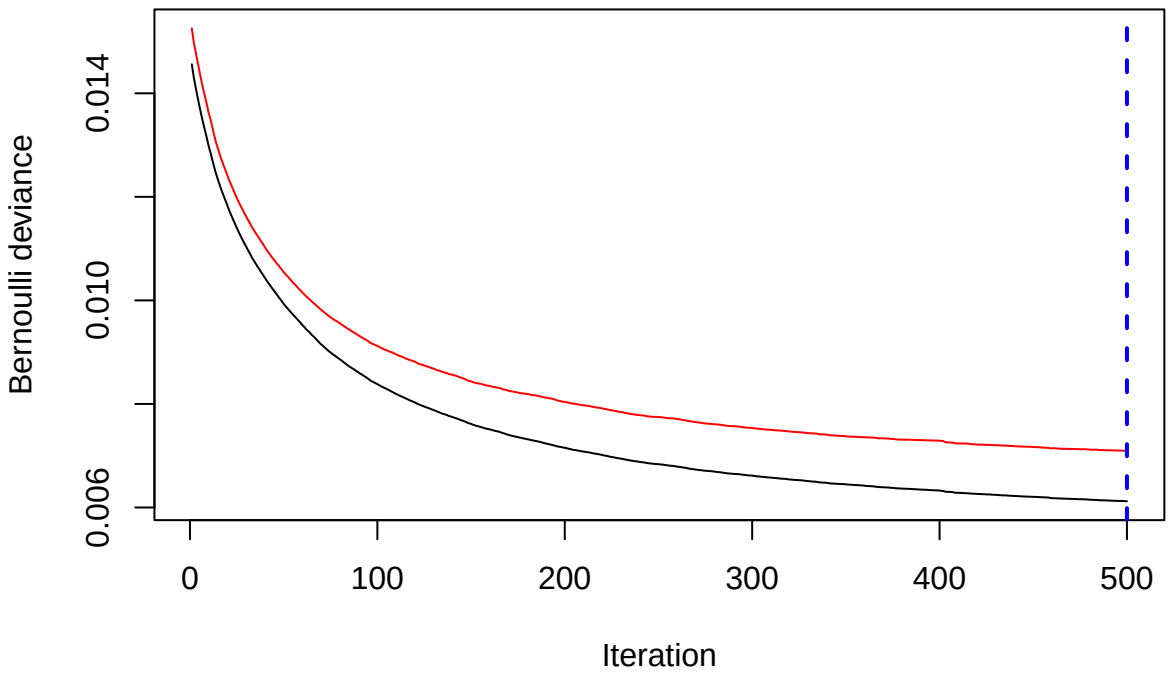



### Gradient Boosting Model (GBM)

```
## Gradient Boosting (GBM) ##
system.time(
  gbm_model <- gbm(Class ~ .,
                   distribution = "bernoulli",
                   data = rbind(train, test),
                   n.trees = 500,
                   interaction.depth = 3,
                   n.minobsinnode = 100,
                   shrinkage = 0.01,
                   bag.fraction = 0.5,
                   train.fraction = nrow(train) / (nrow(train) + nrow(test))
 )
)
```

```
##    user  system elapsed
## 374.977   4.061 387.992
```

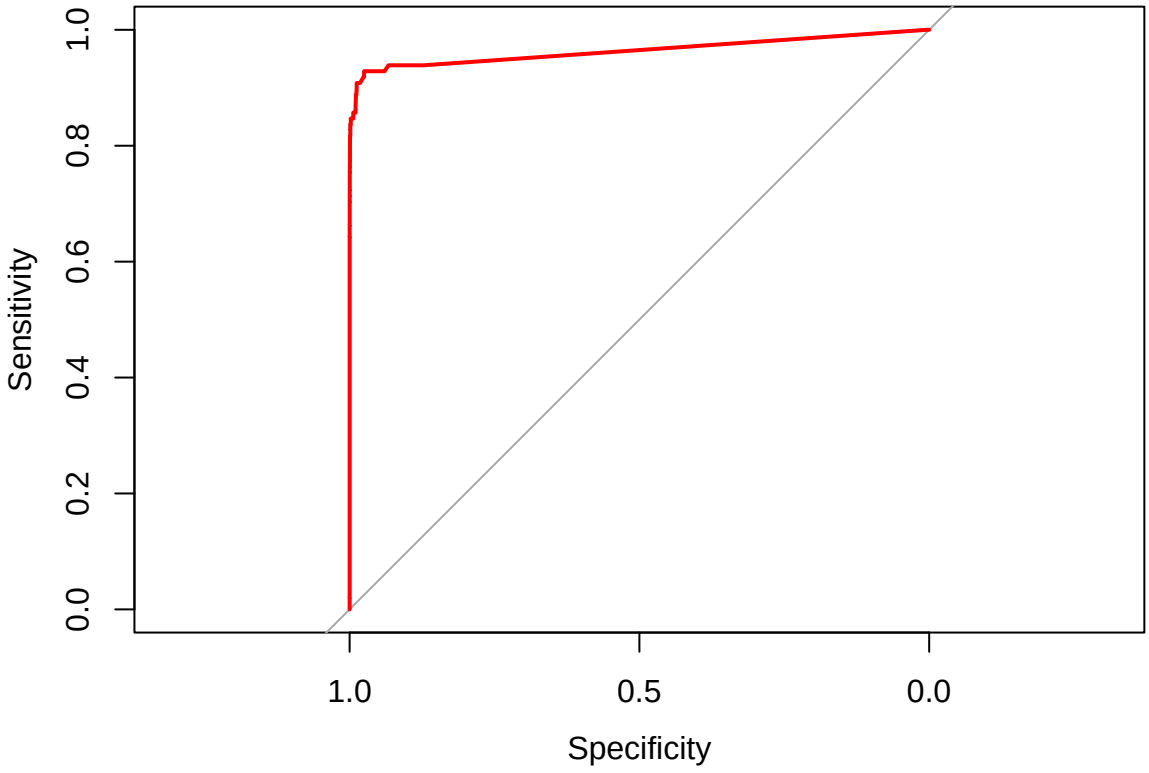
```
gbm.iter <- gbm.perf(gbm_model, method = "test")
```




```
model.influence = relative.influence(gbm_model, n.trees = gbm.iter, sort. = TRUE)
gbm_test <- predict(gbm_model, newdata = test, n.trees = gbm.iter)
gbm_auc <- roc(test$Class, gbm_test, plot = TRUE, col = "red")
```

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

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




```
print(gbm_auc)
```

```
##
## Call:
## roc.default(response = test$Class, predictor = gbm_test, plot = TRUE,     col = "red")
##
## Data: gbm_test in 56863 controls (test$Class 0) < 98 cases (test$Class 1).
## Area under the curve: 0.9637
```

## Results

Analysis was performed with three models. How to show the result:

| Model | AUC |
|---|---|
| Logistic Regression Model | 0.970 |
| Decision Tree Model | 0.944 |
| Gradient Boosting Model (GBM) | 0.967 |

## Conclusions

As you can see from the results, the Logistic Regression Model scored the highest. This was an unexpected result for me. I thought Decision Tree Model had the highest score, but it turned out to be the lowest score. That said, I think they are all good models.

In the future, I think it will be possible to exceed the score of the Logistic Regression Model by adjusting the parameters of the Gradient Boosting Model. Also, I think you can get a higher score by creating another model.