credML

shruthi

2021-02-06

The objective of the project is to train a machine learning algorithm on the dataset to successfully predict fraudulent transactions.

Given the class imbalance ratio, we will be using measuring the accuracy using the Area Under the Precision-Recall Curve (AUC). Confusion matrix accuracy is not meaningful for unbalanced classification.

```
library(dplyr) # for data manipulation
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(stringr) # for data manipulation
library(caret) # for sampling
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(caTools) # for train/test split
library(ggplot2) # for data visualization
library(corrplot) # for correlations
```

```
## corrplot 0.92 loaded
```

```
library(Rtsne) # for tsne plotting
library(ROSE)# for ROSE sampling
```

```
## Loaded ROSE 0.0-4
```

```
library(rpart)# for decision tree model
library(Rborist)# for random forest model
```

```
## Rborist 0.2-3
```

```
## Type RboristNews() to see new features/changes/bug fixes.
```

```
library(xgboost) # for xgboost model
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(rpart.plot)
```

```
# function to set plot height and width
fig <- function(width, heigth){
     options(repr.plot.width = width, repr.plot.height = heigth)
}
# loading the data
dataset = read.csv(file.choose())
head(dataset)
```

```
##   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
## 6    2 -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 149.62     0
## 2  0.1671704  0.1258945 -0.008983099  0.01472417   2.69     0
## 3 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66     0
## 4  0.6473760 -0.2219288  0.062722849  0.06145763 123.50     0
## 5 -0.2060096  0.5022922  0.219422230  0.21515315  69.99     0
## 6 -0.2327938  0.1059148  0.253844225  0.08108026   3.67     0
```

```
tail(dataset)
```

```
##          Time          V1          V2         V3         V4          V5
## 284802 172785   0.1203164  0.93100513 -0.5460121 -0.7450968  1.13031398
## 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
## 284802 -0.2359732  0.8127221  0.1150929 -0.2040635 -0.6574221  0.6448373
## 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
## 284802  0.19091623 -0.5463289 -0.73170658 -0.80803553  0.5996281  0.07044075
## 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
## 284802  0.3731103  0.1289038 0.0006758329 -0.3142046 -0.8085204  0.05034266
## 284803  0.5106323 -0.6829197 1.4758291347  0.2134541  0.1118637  1.01447990
## 284804 -1.2211789 -1.5455561 0.0596158999  0.2142053  0.9243836  0.01246304
## 284805  0.3956525 -0.5772518 0.0013959703  0.2320450  0.5782290 -0.03750086
## 284806  1.1139806  2.8978488 0.1274335158  0.2652449  0.8000487 -0.16329794
## 284807  0.1674299 -0.2561169 0.3829481049  0.2610573  0.6430784  0.37677701
##                 V24        V25        V26          V27         V28 Amount Class
## 284802  0.102799590 -0.4358701  0.1240789  0.217939865  0.06880333   2.69     0
## 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
```

```
# view the table from class column (0 for legit transactions and 1 for fraud)
table(dataset$Class)
```

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

```
# view names of colums  of dataset
names(dataset)
```

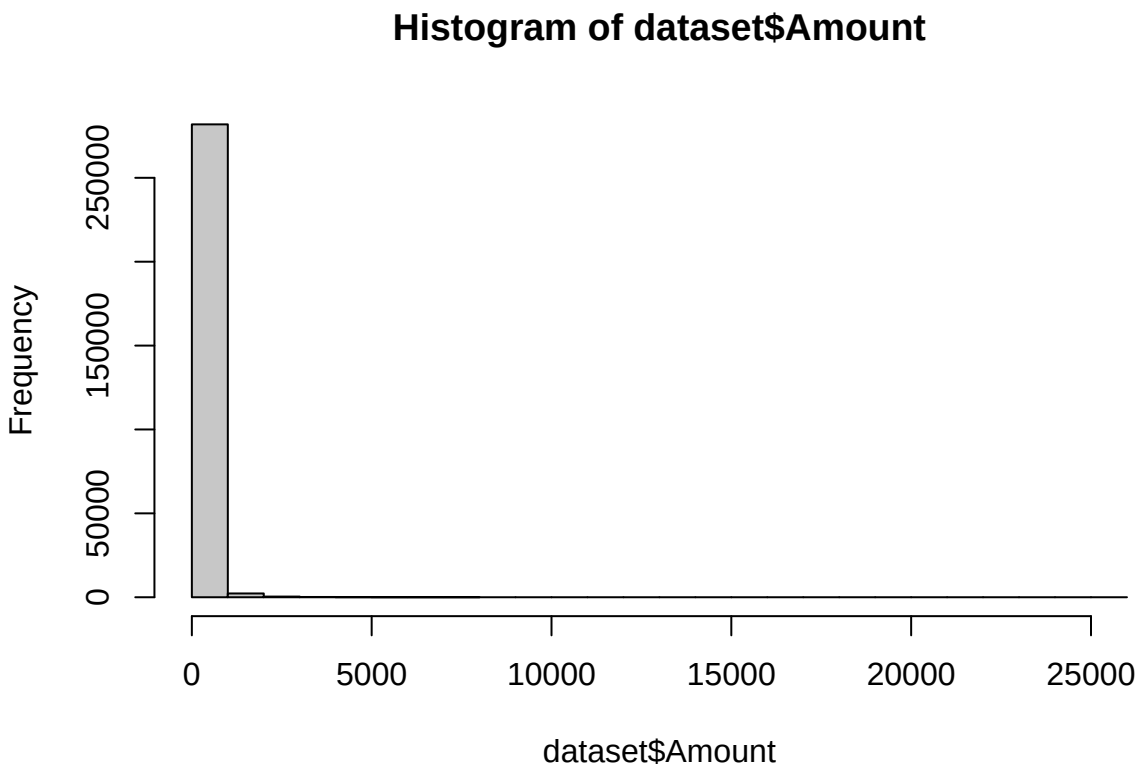
```
##  [1] "Time"   "V1"     "V2"     "V3"     "V4"     "V5"     "V6"     "V7"
##  [9] "V8"     "V9"     "V10"    "V11"    "V12"    "V13"    "V14"    "V15"
## [17] "V16"    "V17"    "V18"    "V19"    "V20"    "V21"    "V22"    "V23"
## [25] "V24"    "V25"    "V26"    "V27"    "V28"    "Amount" "Class"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

By looking at the data, we can see that there are 28 anonymous variables v1 - v28, one time column, one amount column and one label column(0 for not fraud and 1 for fraud). We will visualize this data into histogram and bar plot to find any connection or relation between variables.

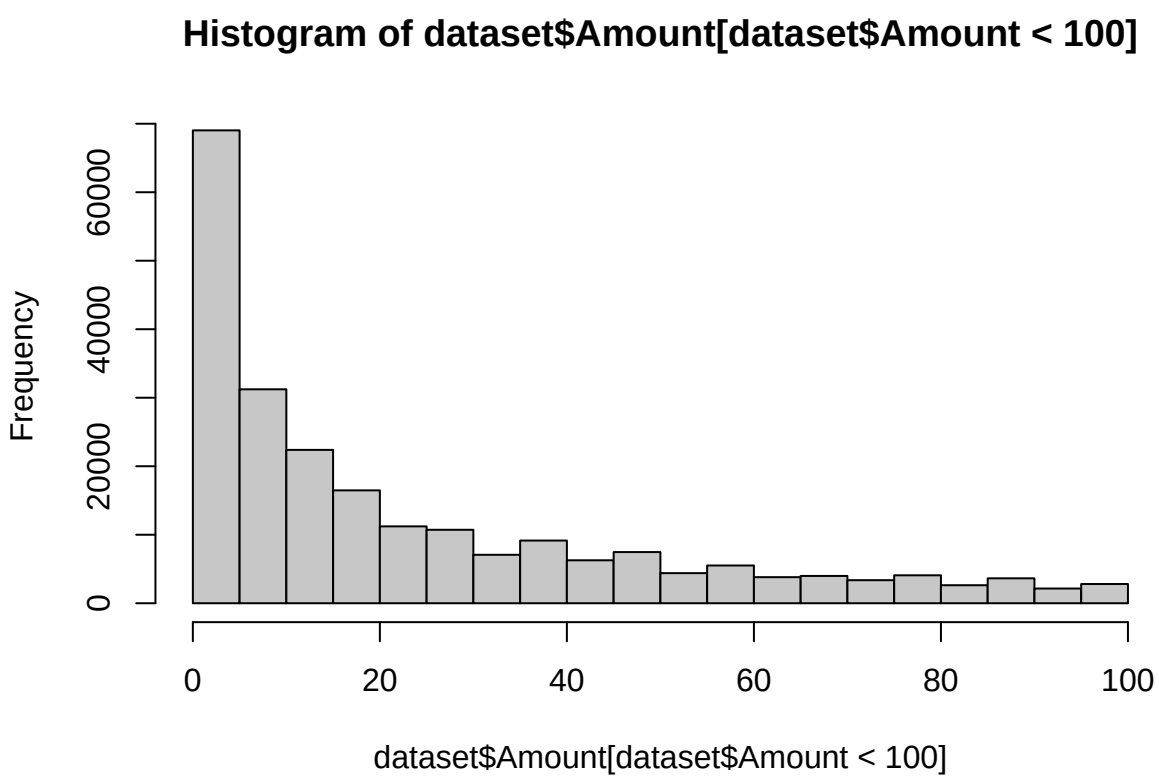
```
summary(dataset$Amount)
```

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

```
hist(dataset$Amount)
```




```
hist(dataset$Amount[dataset$Amount < 100])
```




```
# view variance and standard deviation of amount column
var(dataset$Amount)
```

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

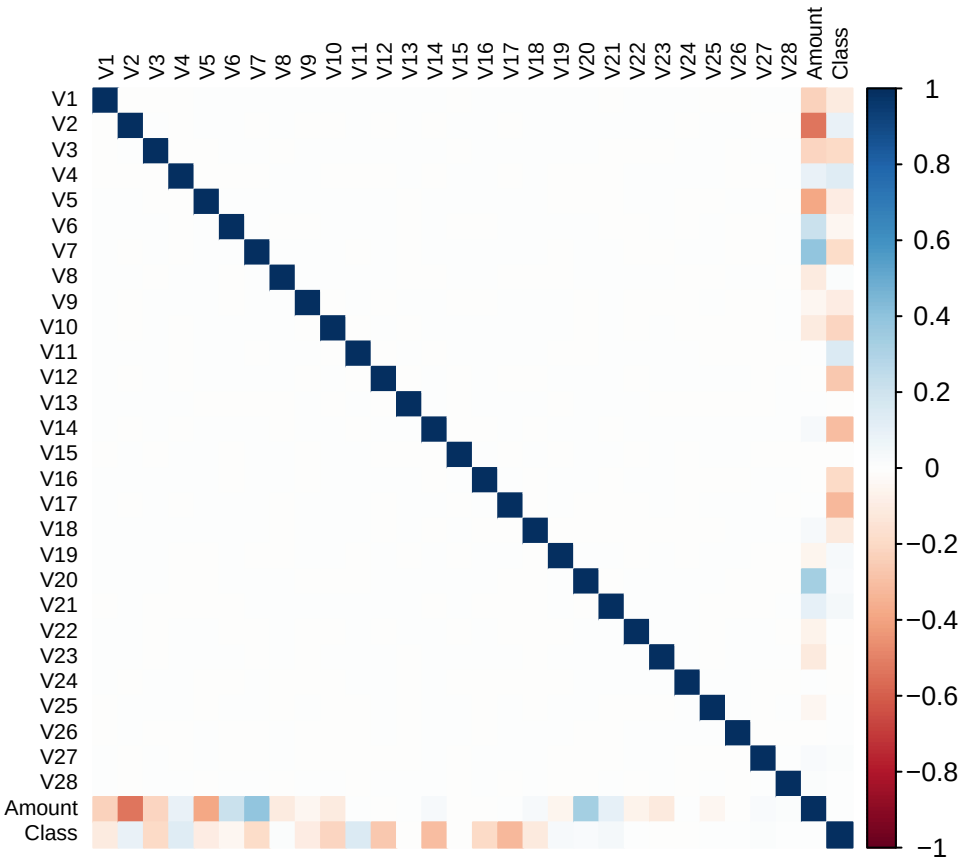
```
sd(dataset$Amount)
```

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

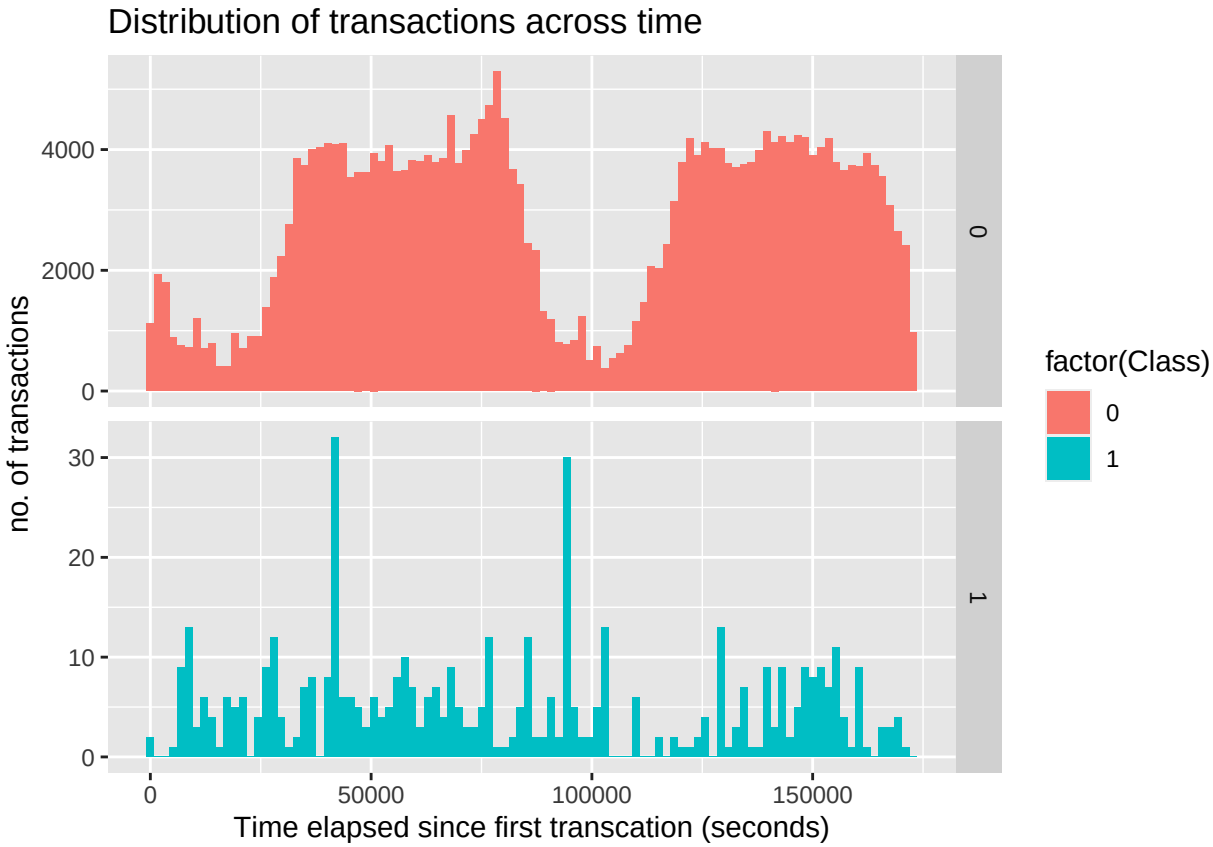
```
# check whether there are any missing values in colums
colSums(is.na(dataset))
```

```
##   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
```

```
# correlation of anonymous variables with amount and class
correlation <- cor(dataset[, -1], method = "pearson")
corrplot(correlation, number.cex = 1, method = "color", type = "full", tl.cex=0.7, tl.col="black")
```



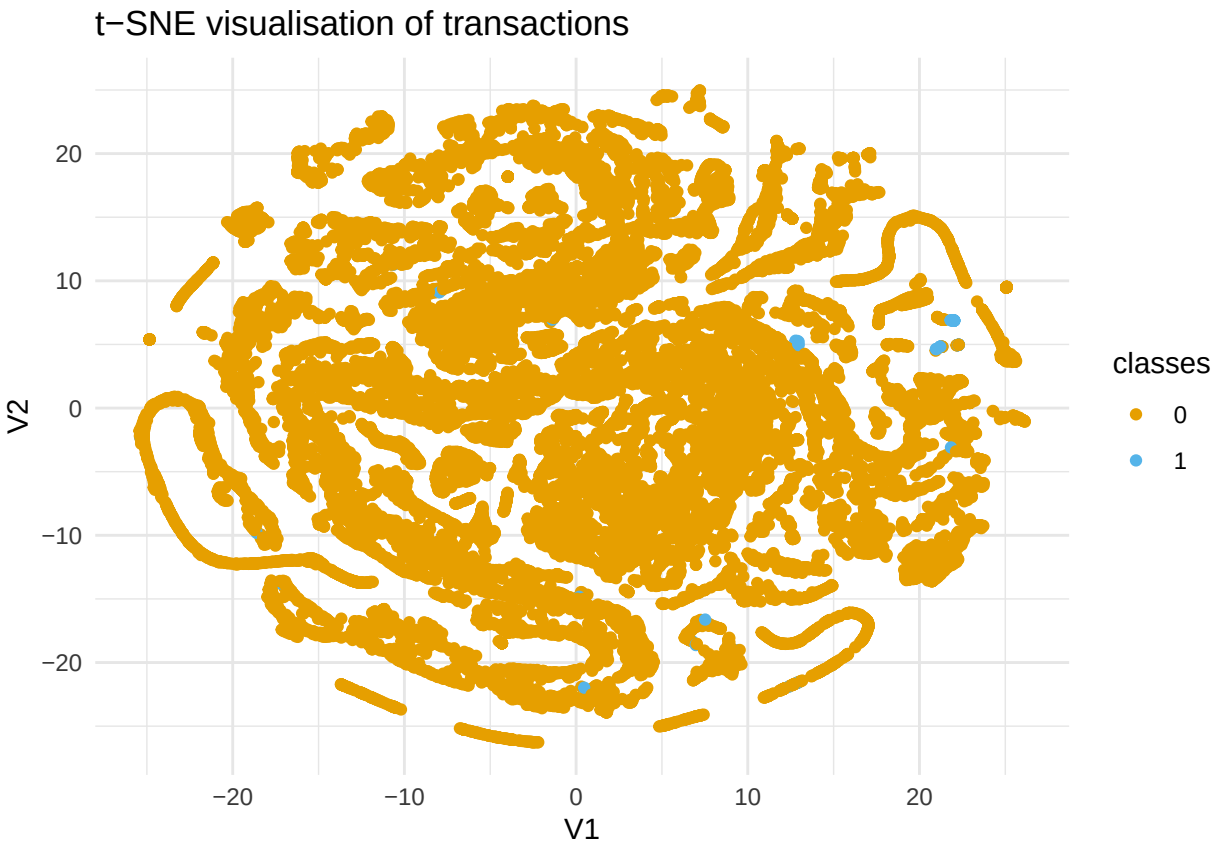

```
# visualizing the distribution of transcations across time
dataset %>%
  ggplot(aes(x = Time, fill = factor(Class))) +
  geom_histogram(bins = 100) +
  labs(x = "Time elapsed since first transcation (seconds)", y = "no. of transactions", title = "Distril
  facet_grid(Class ~ ., scales = 'free_y') + theme()
```



pretty similar in both transactions. Since time doesn't contribute much in fraud detection we can remove the time column from the data. that most of the features are not corelated.In fact, all the anonymous variables are independent to each other.

The last visualization we can observe is the visualization of transactions using t-SNE (t-Distributed Stochastic Neighbor Embedding). This helps us reduce the dimensionality of the data and find any discoverable patterns if present. If there are no patttern present, it would be difficult to train the model.

```
# only use 10% of data to compute SNE and perplexity to 20
tsne_data <- 1:as.integer(0.1*nrow(dataset))
tsne <- Rtsne(dataset[tsne_data,-c(1, 31)], perplexity = 20, theta = 0.5, pca = F, verbose = F, max_ite
classes <- as.factor(dataset$Class[tsne_data])
tsne_matrix <- as.data.frame(tsne$Y)
ggplot(tsne_matrix, aes(x = V1, y = V2)) + geom_point(aes(color = classes)) + theme_minimal() + ggtitle
```



Since, most of the fraud transactions lie near the edge of the blob of data, we can use different models to differentiate fraud transactions.

```
# scaling the data using standardization and remove the first column (time) from the data set
dataset$Amount <- scale(dataset$Amount)
new_data <- dataset[, -c(1)]
head(new_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
```

```
# change 'Class' variable to factor
new_data$Class <- as.factor(new_data$Class)
levels(new_data$Class) <- c("Not Fraud", "Fraud")
```

```
# split the data into training set and test set
set.seed(101)
split <- sample.split(new_data$Class, SplitRatio = 0.8)
train_data <- subset(new_data, split == TRUE)
test_data <- subset(new_data, split == FALSE)
dim(train_data)
```

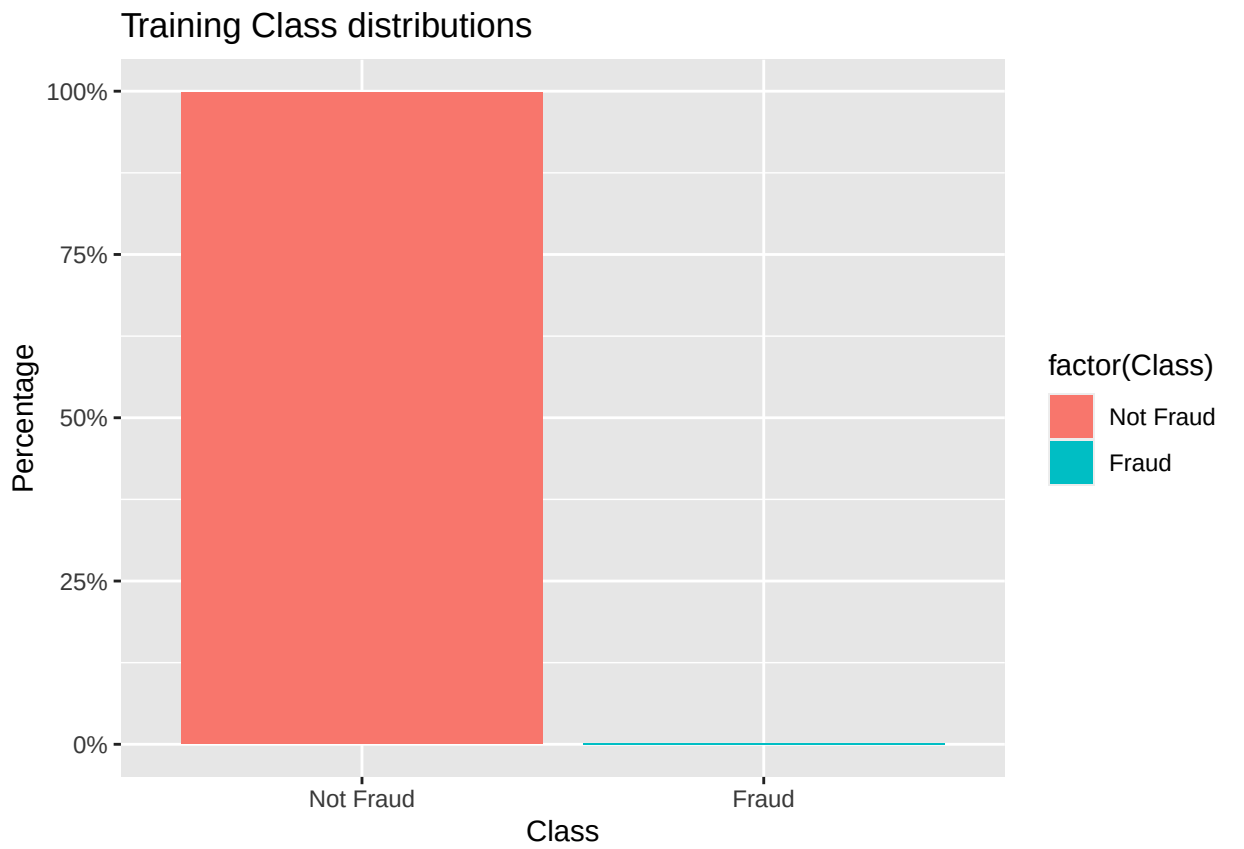
```
## [1] 227846     30
```

```
dim(test_data)
```

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

```
# visualize the training data
train_data %>% ggplot(aes(x = factor(Class), y = prop.table(stat(count)), fill = factor(Class))) +
  geom_bar(position = "dodge") +
  scale_y_continuous(labels = scales::percent) +
  labs(x = 'Class', y = 'Percentage', title = 'Training Class distributions') +
  theme_grey()
```

```
## Warning: `stat(count)` was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(count)` instead.
```



Since the data is heavily unbalanced with 99% of non-fraudulent data, this may result in our model perfoming less accurately and being heavily baised towards non-fraudulent transactions. So, We sample the data using ROSE (Random over sampling examples), Over sampling or Down sampling method, and examine the area under ROC curve at each sampling methods

```
set.seed(9560)
up_train_data <- upSample(x = train_data[, -30],
                          y = train_data$Class)
table(up_train_data$Class)
```

```
##
## Not Fraud     Fraud
##    227452    227452
```

```
set.seed(9560)
down_train_data <- downSample(x = train_data[, -30],
                          y = train_data$Class)
table(down_train_data$Class)
```

```
##
## Not Fraud     Fraud
##       394       394
```

we will use Down Sampling to reduce the time for model training and execution. Now, we will test each models and see which one classifies the data better using ROC-AUC curve.

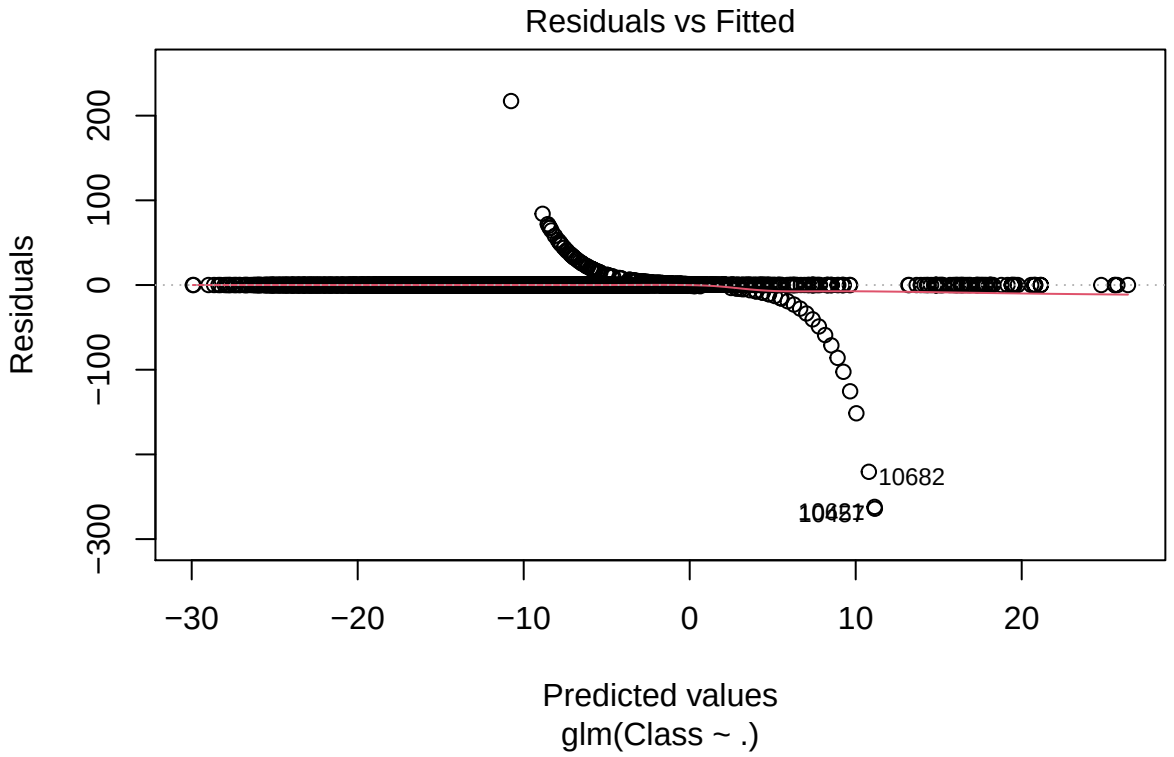
```r
# fitting the logistic model
logistic_model <- glm(Class ~ .,train_data, family=binomial())
```

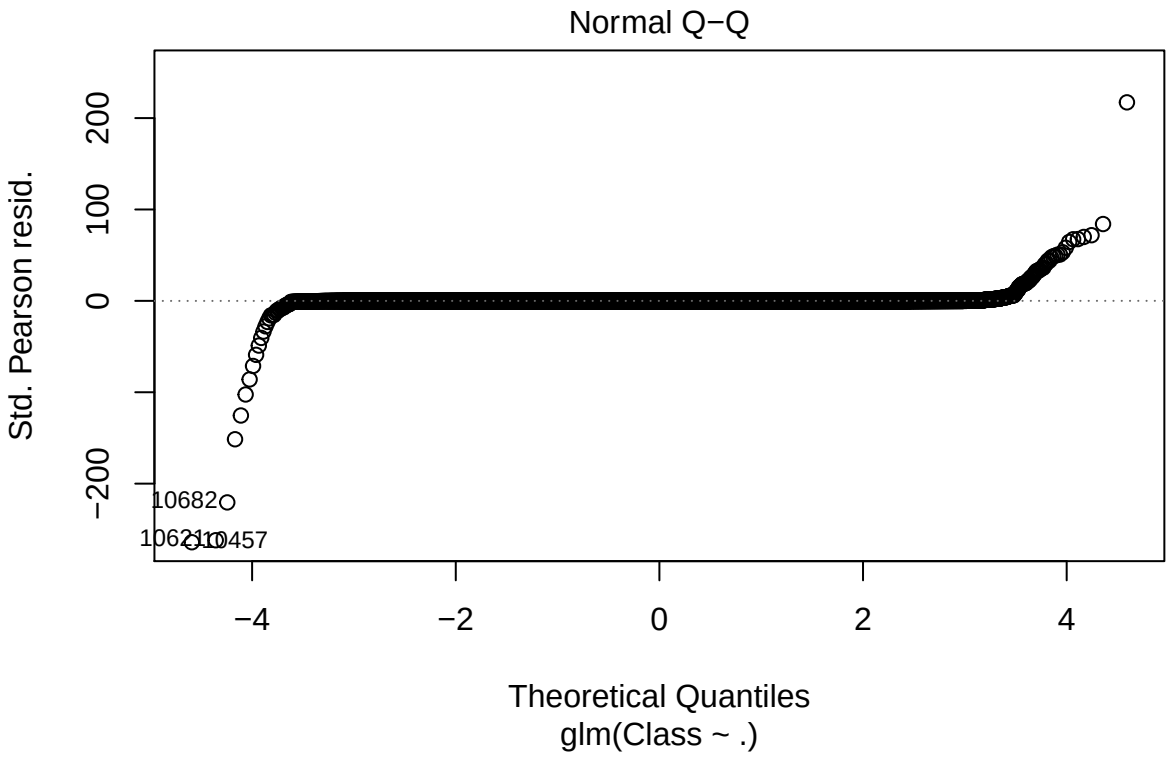
```r
summary(logistic_model)
```

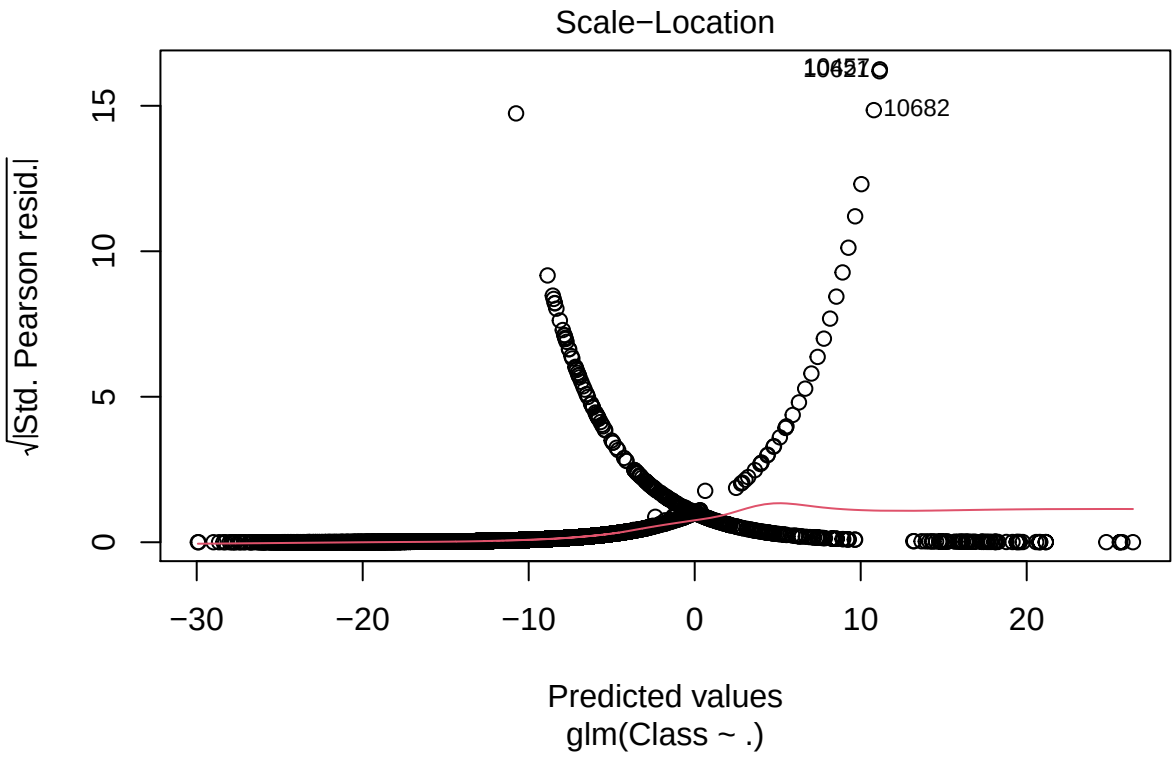
```
##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.7229  -0.0293  -0.0191  -0.0121   4.6394
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.687149   0.164544 -52.795  < 2e-16 ***
## V1           0.084228   0.045950   1.833 0.066796 .
## V2          -0.001398   0.065362  -0.021 0.982930
## V3           0.062092   0.051311   1.210 0.226237
## V4           0.716013   0.082519   8.677  < 2e-16 ***
## V5           0.082620   0.074982   1.102 0.270521
## V6          -0.128168   0.087108  -1.471 0.141193
## V7          -0.128104   0.075603  -1.694 0.090186 .
## V8          -0.165925   0.033045  -5.021 5.14e-07 ***
## V9          -0.228039   0.122095  -1.868 0.061801 .
## V10         -0.863852   0.106469  -8.114 4.91e-16 ***
## V11         -0.019299   0.082523  -0.234 0.815093
## V12          0.116797   0.094981   1.230 0.218815
## V13         -0.298690   0.089906  -3.322 0.000893 ***
## V14         -0.559990   0.067084  -8.348  < 2e-16 ***
## V15         -0.135719   0.091947  -1.476 0.139929
## V16         -0.205345   0.140406  -1.463 0.143601
## V17          0.038270   0.075853   0.505 0.613887
## V18         -0.040921   0.142073  -0.288 0.773324
## V19          0.068355   0.106007   0.645 0.519049
## V20         -0.453581   0.092009  -4.930 8.23e-07 ***
## V21          0.382916   0.062362   6.140 8.24e-10 ***
## V22          0.589199   0.142789   4.126 3.69e-05 ***
## V23         -0.096850   0.062597  -1.547 0.121820
## V24          0.115586   0.165135   0.700 0.483962
## V25         -0.065060   0.142226  -0.457 0.647354
## V26          0.064003   0.206586   0.310 0.756703
## V27         -0.825747   0.132892  -6.214 5.18e-10 ***
## V28         -0.290026   0.092154  -3.147 0.001648 **
## Amount       0.244986   0.110707   2.213 0.026903 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5799.1  on 227845  degrees of freedom
## Residual deviance: 1824.2  on 227816  degrees of freedom
## AIC: 1884.2
```

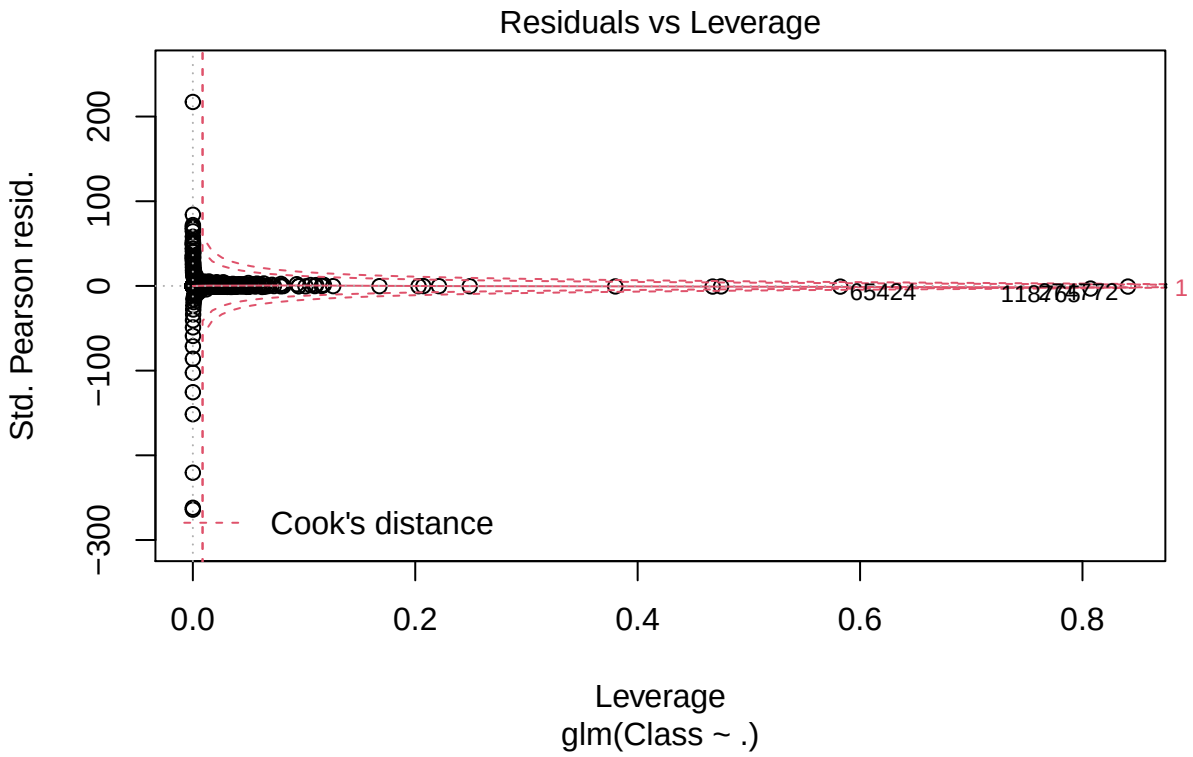
```
##
## Number of Fisher Scoring iterations: 12
```

```
plot(logistic_model)
```

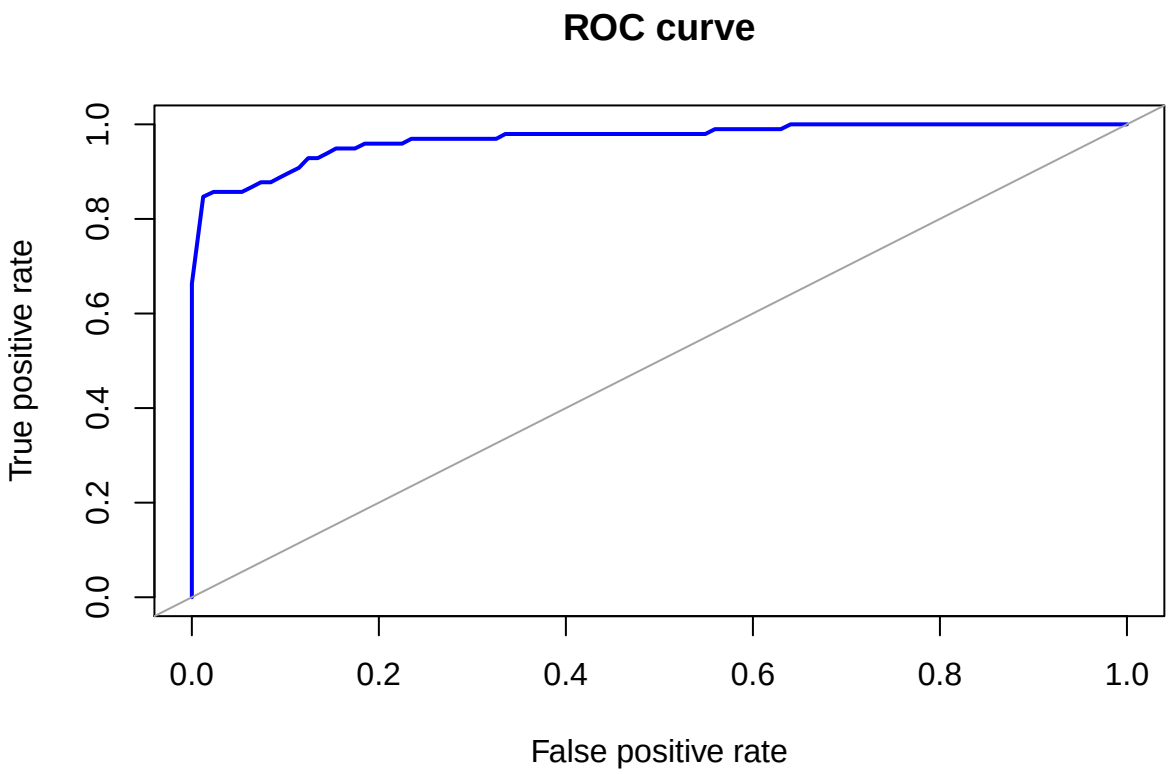



Normal Q−Q
Std. Pearson resid.
200
100
0
−200
10682
10621
10457
−4
−2
0
2
4
Theoretical Quantiles
glm(Class ~ .)


Scale−Location
10682
√|Std. Pearson resid.|
0
5
10
15
−30
−20
−10
0
10
20
Predicted values
glm(Class ~ .)

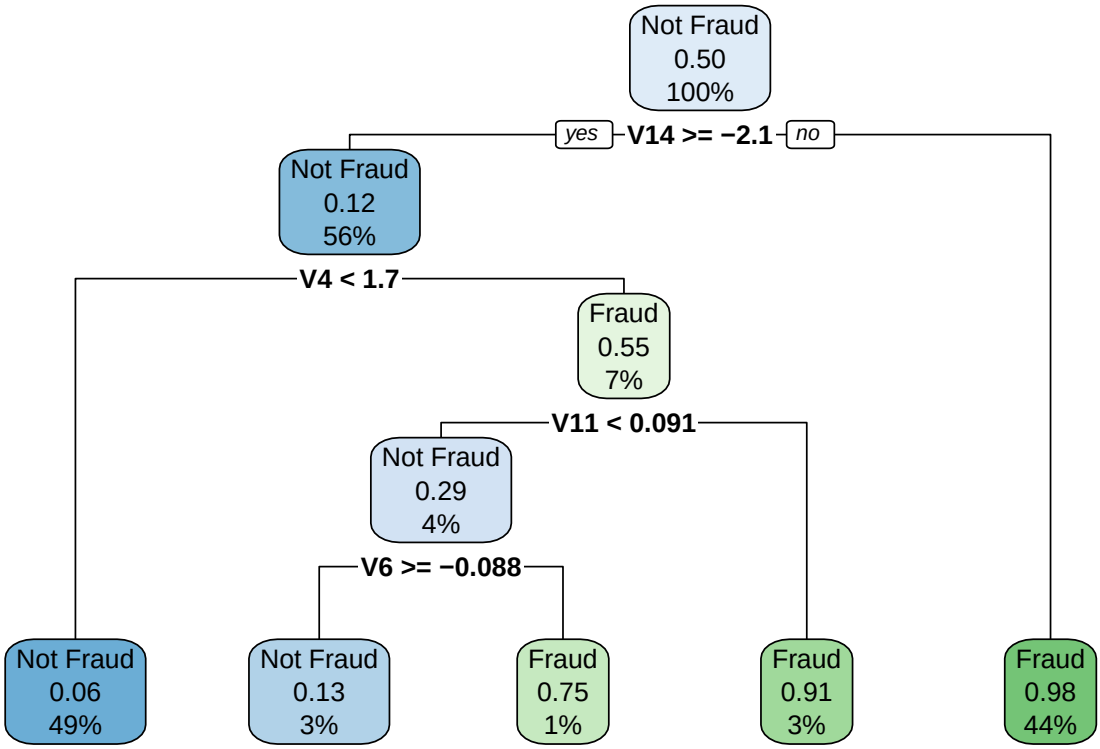



```
logistic_predictions <- predict(logistic_model, test_data, type='response')
roc.curve(test_data$Class, logistic_predictions, plotit = TRUE, col = "blue")
```




```
## Area under the curve (AUC): 0.969
```

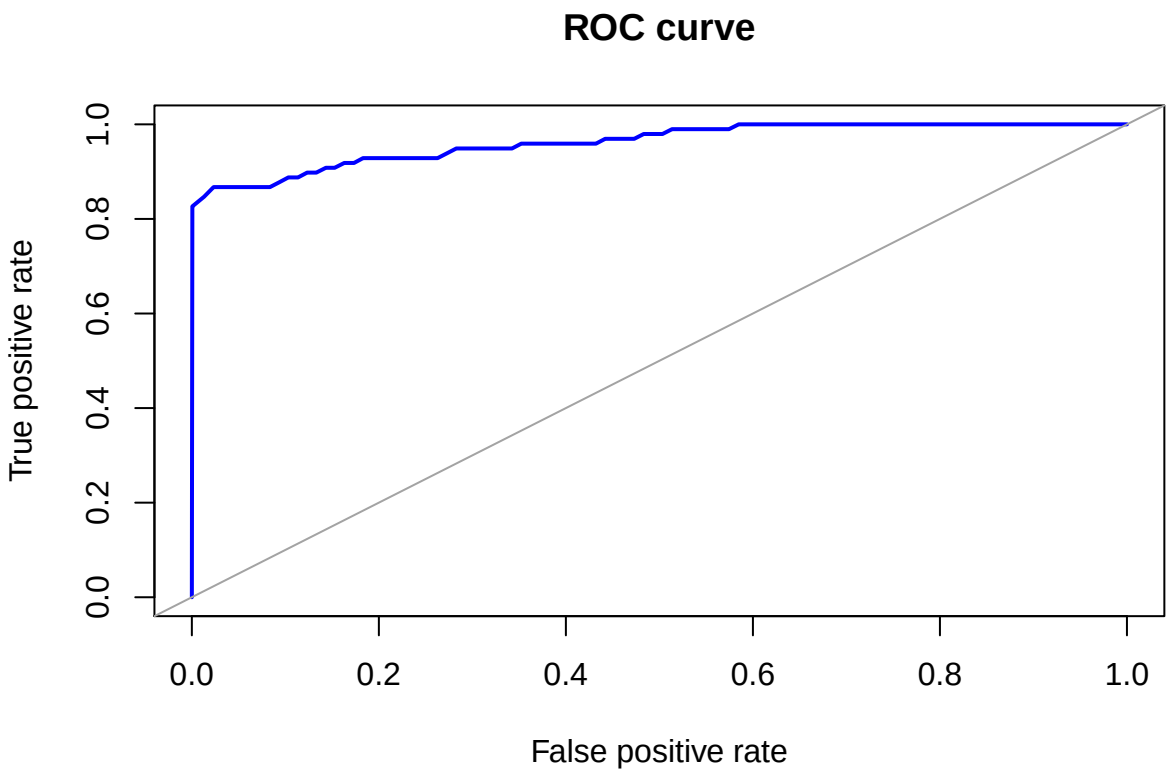
```
decisionTree_model <- rpart(Class ~ . ,down_train_data, method = 'class')
predicted_val <- predict(decisionTree_model,down_train_data, type = 'class')
probability <- predict(decisionTree_model,down_train_data, type = 'prob')
rpart.plot(decisionTree_model)
```




```
x = down_train_data[, -30]
y = down_train_data[,30]

rf_fit <- Rborist(x, y, ntree = 1000, minNode = 20, maxLeaf = 13)


rf_pred <- predict(rf_fit, test_data[,-30], ctgCensus = "prob")
```

```
## Warning in data.table::setDT(x): Some columns are a multi-column type (such as
## a matrix column): [29]. setDT will retain these columns as-is but subsequent
## operations like grouping and joining may fail. Please consider as.data.table()
## instead which will create a new column for each embedded column.
```

```
prob <- rf_pred$prob

roc.curve(test_data$Class, prob[,2], plotit = TRUE, col = 'blue')
```

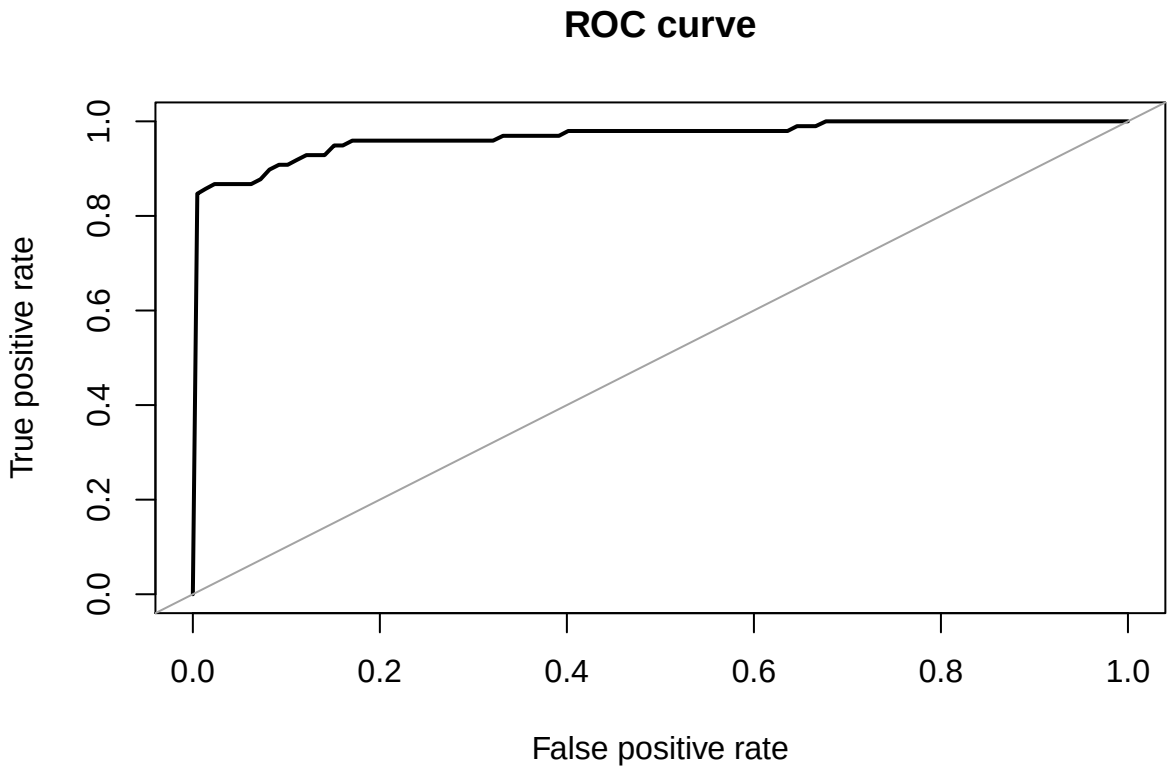



```
## Area under the curve (AUC): 0.962
```

From the random forest model, we got area under the ROC Curve: 0.962

```
set.seed(40)

#Convert class labels from factor to numeric
labels <- down_train_data$Class
y <- recode(labels, 'Not Fraud' = 0, "Fraud" = 1)

# xgb fit
xgb_fit <- xgboost(data = data.matrix(down_train_data[,-30]),
 label = y,
 eta = 0.1,
 gamma = 0.1,
 max_depth = 10,
 nrounds = 300,
 objective = "binary:logistic",
 colsample_bytree = 0.6,
 verbose = 0,
 nthread = 7
)
```

```
# XGBoost predictions
xgb_pred <- predict(xgb_fit, data.matrix(test_data[,-30]))
roc.curve(test_data$Class, xgb_pred, plotit = TRUE)
```

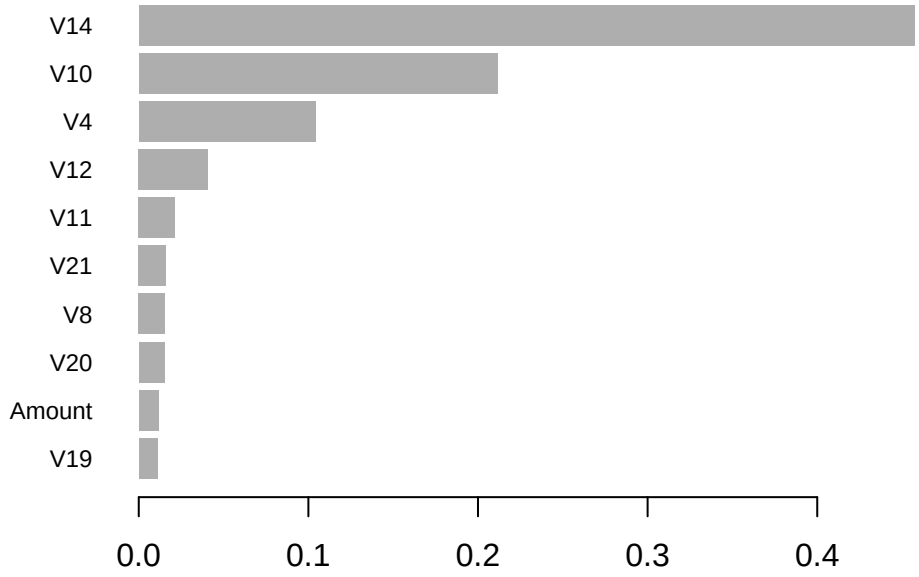



```
## Area under the curve (AUC): 0.967
```

From the XGBoost model, we got area under the ROC Curve: 0.968

We can also check which variables has signigicant role in fraud detection. V14 stood out in decision tree model. Let's compare it with XGboost model.

```
names <- dimnames(data.matrix(train_data[,-30]))[[2]]

# Compute feature importance matrix
importance_matrix <- xgb.importance(names, model = xgb_fit)
# Nice graph
xgb.plot.importance(importance_matrix[1:10,])
```



As we can see v14 has significant role in distinguishing the fraud and non-fraud transactions. Conclusion: From the above plots and models, we can clarify that XGBoost performed better than logistic and Random Forest Model, although the margin was not very high. We can also fine tune the XGBoost model to make it perform even better. It is really great how models are able to find the distinguishing features between fraud and non-fraud transactions from such a big data.