Project : Credit card Fraud Detection

Loading Libraries

```
checkAndInstallPackages <- function (packages)  {

   # Iteration to check if package is installed
   for(package in packages){

       # If package is installed then load it
       if(package %in% rownames(installed.packages()))
          do.call('library', list(package))

       # If package is not installed then download it and load
       else {
          install.packages(package)
          do.call("library", list(package))
       }
   }
}

# List of packages needed for the project
packages <- c("ggplot2",
              "dplyr",
              "corrplot",
              "rpart",
              "caret",
              "pROC",
              "rpart.plot",
              "randomForest",
              "DMwR",
              "AUC",
              "ggfortify",
              "class",
              "rfUtilities")
checkAndInstallPackages(packages)
```

```
##
## 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

## corrplot 0.84 loaded
```

```
## Loading required package: lattice

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin

## Loading required package: grid

## AUC 0.3.0

## Type AUCNews() to see the change log and ?AUC to get an overview.

##
## Attaching package: 'AUC'

## The following objects are masked from 'package:pROC':
##
##     auc, roc

## The following objects are masked from 'package:caret':
##
##     sensitivity, specificity

##
## Attaching package: 'ggfortify'

## The following object is masked from 'package:DMwR':
##
##     unscale

##
## Attaching package: 'rfUtilities'

## The following object is masked from 'package:AUC':
##
##     accuracy
```

Data Acquisition

Download the dataset from https://www.kaggle.com/mlg-ulb/creditcardfraud and import into R studio.

```
creditcard.raw <- read.csv("creditcard.csv", header = T)
creditcard <- as.data.frame(creditcard.raw)
head(creditcard)
```

```
##   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
## 1  0.40399296  0.25141210 -0.018306778  0.277837576 -0.11047391
## 2 -0.14578304 -0.06908314 -0.225775248 -0.638671953  0.10128802
## 3 -2.26185710  0.52497973  0.247998153  0.771679402  0.90941226
## 4 -1.23262197 -0.20803778 -0.108300452  0.005273597 -0.19032052
## 5  0.80348692  0.40854236 -0.009430697  0.798278495 -0.13745808
## 6 -0.03319379  0.08496767 -0.208253515 -0.559824796 -0.02639767
##           V24        V25        V26          V27         V28 Amount Class
## 1  0.06692807  0.1285394 -0.1891148  0.133558377 -0.02105305 149.62     0
## 2 -0.33984648  0.1671704  0.1258945 -0.008983099  0.01472417   2.69     0
## 3 -0.68928096 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66     0
## 4 -1.17557533  0.6473760 -0.2219288  0.062722849  0.06145763 123.50     0
## 5  0.14126698 -0.2060096  0.5022922  0.219422230  0.21515315  69.99     0
## 6 -0.37142658 -0.2327938  0.1059148  0.253844225  0.08108026   3.67     0
```

```
#Structure of dataset. Transforming Class variable to factor
str(creditcard)
```

```
## 'data.frame':    284807 obs. of  31 variables:
##  $ Time  : num  0 0 1 1 2 2 4 7 7 9 ...
##  $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
##  $ V2    : num  -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...
##  $ V3    : num  2.536 0.166 1.773 1.793 1.549 ...
##  $ V4    : num  1.378 0.448 0.38 -0.863 0.403 ...
```

```
##  $ V5    : num  -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...
##  $ V6    : num  0.4624 -0.0824 1.8005 1.2472 0.0959 ...
##  $ V7    : num  0.2396 -0.0788 0.7915 0.2376 0.5929 ...
##  $ V8    : num  0.0987 0.0851 0.2477 0.3774 -0.2705 ...
##  $ V9    : num  0.364 -0.255 -1.515 -1.387 0.818 ...
##  $ V10   : num  0.0908 -0.167 0.2076 -0.055 0.7531 ...
##  $ V11   : num  -0.552 1.613 0.625 -0.226 -0.823 ...
##  $ V12   : num  -0.6178 1.0652 0.0661 0.1782 0.5382 ...
##  $ V13   : num  -0.991 0.489 0.717 0.508 1.346 ...
##  $ V14   : num  -0.311 -0.144 -0.166 -0.288 -1.12 ...
##  $ V15   : num  1.468 0.636 2.346 -0.631 0.175 ...
##  $ V16   : num  -0.47 0.464 -2.89 -1.06 -0.451 ...
##  $ V17   : num  0.208 -0.115 1.11 -0.684 -0.237 ...
##  $ V18   : num  0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...
##  $ V19   : num  0.404 -0.146 -2.262 -1.233 0.803 ...
##  $ V20   : num  0.2514 -0.0691 0.525 -0.208 0.4085 ...
##  $ V21   : num  -0.01831 -0.22578 0.248 -0.1083 -0.00943 ...
##  $ V22   : num  0.27784 -0.63867 0.77168 0.00527 0.79828 ...
##  $ V23   : num  -0.11 0.101 0.909 -0.19 -0.137 ...
##  $ V24   : num  0.0669 -0.3398 -0.6893 -1.1756 0.1413 ...
##  $ V25   : num  0.129 0.167 -0.328 0.647 -0.206 ...
##  $ V26   : num  -0.189 0.126 -0.139 -0.222 0.502 ...
##  $ V27   : num  0.13356 -0.00898 -0.05535 0.06272 0.21942 ...
##  $ V28   : num  -0.0211 0.0147 -0.0598 0.0615 0.2152 ...
##  $ Amount: num  149.62 2.69 378.66 123.5 69.99 ...
##  $ Class : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
creditcard$Class <- as.factor(creditcard$Class)
```

Data Exploration

Exploratory Data Plots

All variables are numeric except Class variable. Time and Amount are actual variables, whereas V1 - V28 are pricipal components of actual data as raw data can't be publicly available due to privacy and confidentiality. ####Overview of dataset

```
summary(creditcard)
```

```
##       Time              V1                  V2                
##  Min.   :     0   Min.   :-56.40751   Min.   :-72.71573  
##  1st Qu.: 54202   1st Qu.: -0.92037   1st Qu.: -0.59855  
##  Median : 84692   Median :  0.01811   Median :  0.06549  
##  Mean   : 94814   Mean   :  0.00000   Mean   :  0.00000  
##  3rd Qu.:139321   3rd Qu.:  1.31564   3rd Qu.:  0.80372  
##  Max.   :172792   Max.   :  2.45493   Max.   : 22.05773  
##        V3                  V4                  V5                 
##  Min.   :-48.3256   Min.   :-5.68317   Min.   :-113.74331  
##  1st Qu.: -0.8904   1st Qu.:-0.84864   1st Qu.:  -0.69160  
##  Median :  0.1799   Median :-0.01985   Median :  -0.05434  
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :   0.00000
```

```
##  3rd Qu.:  1.0272   3rd Qu.: 0.74334   3rd Qu.:   0.61193
##  Max.   :  9.3826   Max.   :16.87534   Max.   :  34.80167
##        V6                 V7                 V8
##  Min.   :-26.1605   Min.   :-43.5572   Min.   :-73.21672
##  1st Qu.: -0.7683   1st Qu.: -0.5541   1st Qu.: -0.20863
##  Median : -0.2742   Median :  0.0401   Median :  0.02236
##  Mean   :  0.0000   Mean   :  0.0000   Mean   :  0.00000
##  3rd Qu.:  0.3986   3rd Qu.:  0.5704   3rd Qu.:  0.32735
##  Max.   : 73.3016   Max.   :120.5895   Max.   : 20.00721
##        V9                 V10                V11
##  Min.   :-13.43407   Min.   :-24.58826   Min.   :-4.79747
##  1st Qu.: -0.64310   1st Qu.: -0.53543   1st Qu.:-0.76249
##  Median : -0.05143   Median : -0.09292   Median :-0.03276
##  Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.59714   3rd Qu.:  0.45392   3rd Qu.: 0.73959
##  Max.   : 15.59500   Max.   : 23.74514   Max.   :12.01891
##       V12                V13                V14
##  Min.   :-18.6837   Min.   :-5.79188   Min.   :-19.2143
##  1st Qu.: -0.4056   1st Qu.:-0.64854   1st Qu.: -0.4256
##  Median :  0.1400   Median :-0.01357   Median :  0.0506
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :  0.0000
##  3rd Qu.:  0.6182   3rd Qu.: 0.66251   3rd Qu.:  0.4931
##  Max.   :  7.8484   Max.   : 7.12688   Max.   : 10.5268
##       V15                V16                V17
##  Min.   :-4.49894   Min.   :-14.12985   Min.   :-25.16280
##  1st Qu.:-0.58288   1st Qu.: -0.46804   1st Qu.: -0.48375
##  Median : 0.04807   Median :  0.06641   Median : -0.06568
##  Mean   : 0.00000   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.: 0.64882   3rd Qu.:  0.52330   3rd Qu.:  0.39968
##  Max.   : 8.87774   Max.   : 17.31511   Max.   :  9.25353
##       V18                V19                V20
##  Min.   :-9.498746   Min.   :-7.213527   Min.   :-54.49772
##  1st Qu.:-0.498850   1st Qu.:-0.456299   1st Qu.: -0.21172
##  Median :-0.003636   Median : 0.003735   Median : -0.06248
##  Mean   : 0.000000   Mean   : 0.000000   Mean   :  0.00000
##  3rd Qu.: 0.500807   3rd Qu.: 0.458949   3rd Qu.:  0.13304
##  Max.   : 5.041069   Max.   : 5.591971   Max.   : 39.42090
##       V21                V22                V23
##  Min.   :-34.83038   Min.   :-10.933144   Min.   :-44.80774
##  1st Qu.: -0.22839   1st Qu.: -0.542350   1st Qu.: -0.16185
##  Median : -0.02945   Median :  0.006782   Median : -0.01119
##  Mean   :  0.00000   Mean   :  0.000000   Mean   :  0.00000
##  3rd Qu.:  0.18638   3rd Qu.:  0.528554   3rd Qu.:  0.14764
##  Max.   : 27.20284   Max.   : 10.503090   Max.   : 22.52841
##       V24                V25                V26
##  Min.   :-2.83663   Min.   :-10.29540   Min.   :-2.60455
##  1st Qu.:-0.35459   1st Qu.: -0.31715   1st Qu.:-0.32698
##  Median : 0.04098   Median :  0.01659   Median :-0.05214
##  Mean   : 0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.: 0.43953   3rd Qu.:  0.35072   3rd Qu.: 0.24095
##  Max.   : 4.58455   Max.   :  7.51959   Max.   : 3.51735
##       V27                V28                Amount         Class
##  Min.   :-22.565679   Min.   :-15.43008   Min.   :    0.00   0:284315
##  1st Qu.: -0.070840   1st Qu.: -0.05296   1st Qu.:    5.60   1:   492
```

```
##  Median :  0.001342   Median :  0.01124   Median :   22.00
##  Mean   :  0.000000   Mean   :  0.00000   Mean   :   88.35
##  3rd Qu.:  0.091045   3rd Qu.:  0.07828   3rd Qu.:   77.17
##  Max.   : 31.612198   Max.   : 33.84781   Max.   :25691.16
```

Check for missing values

```
sum(is.na(creditcard))
```

```
## [1] 0
```

There are no missing values in the dataset.

Class variable

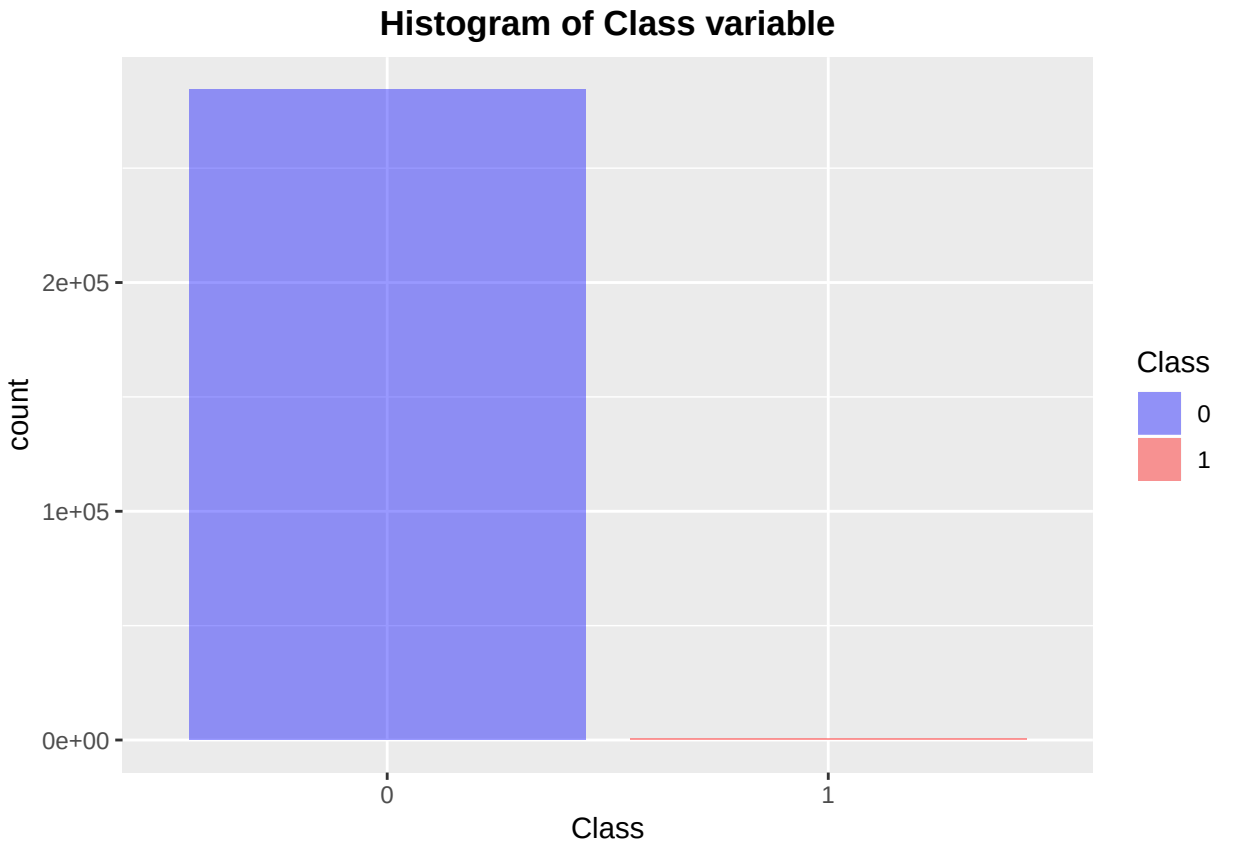
```
# Distribution of Class variables.
table(creditcard$Class)
```

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

```
#Percentage of Class variable distribution
prop.table(table(creditcard$Class))*100
```

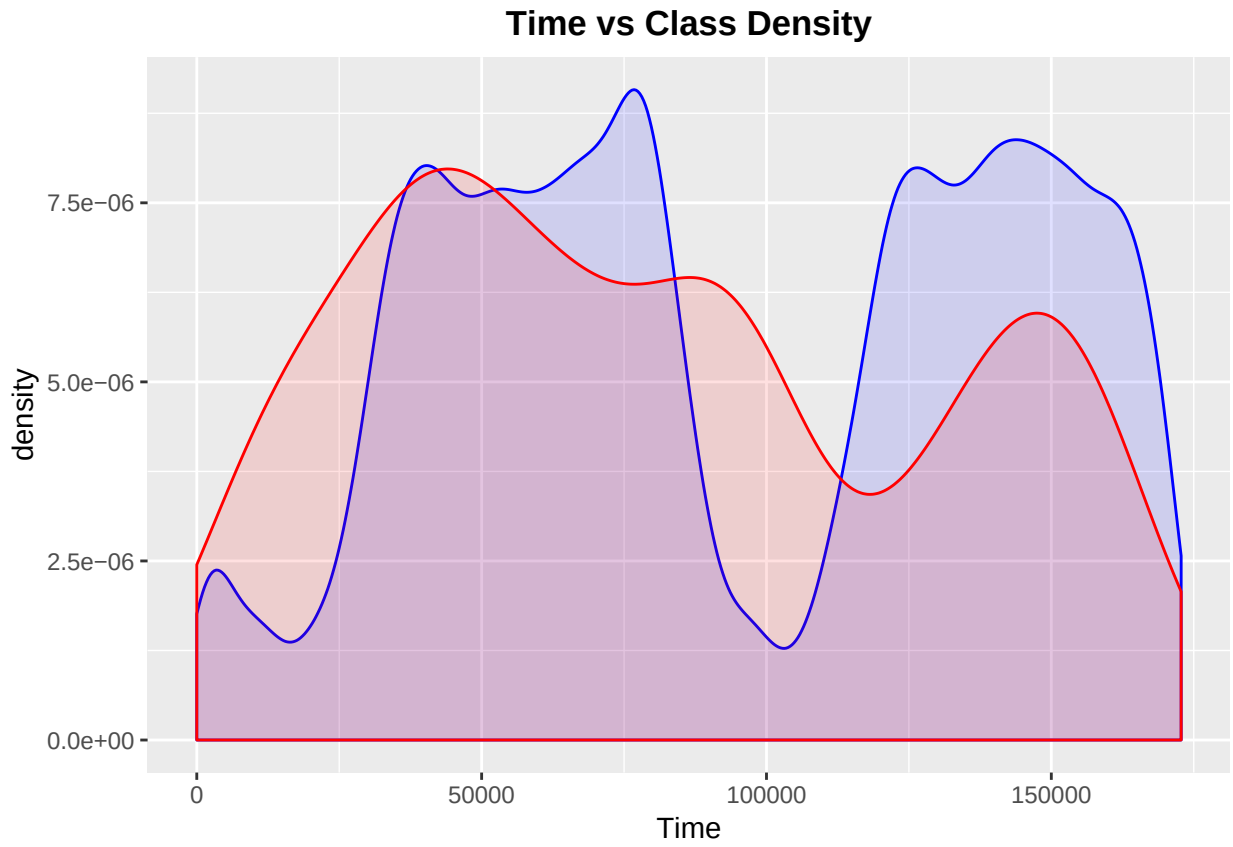
```
##
##          0          1
## 99.8272514  0.1727486
```

```
#Histogram of Class variable
ggplot(creditcard, aes(x = Class, fill = Class)) +
  geom_bar(alpha = 0.4) +
  ggtitle("Histogram of Class variable") +
  scale_fill_manual(values = c("blue","red")) +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```



Positive class (fraudulent trasaction) accounts for 0.172% of the total transactions. Hence the data is higly skewed i.e. Imbalanced. Even a "null" classifier which always predicts class=0 would obtain over 99% accuracy on this task. Hence, measure of mean accuracy can't be used due to chance of missclssification.

```
#Density plot for Class variable
ggplot() +
  geom_density(data = creditcard[creditcard$Class == 0,],
               aes(x=Time),
               color = "blue",
               fill = "blue",
               alpha = 0.12) +
  geom_density(data = creditcard[creditcard$Class == 1,],
               aes(x=Time),
               color = "red",
               fill = "red",
               alpha = 0.12) +
  ggtitle("Time vs Class Density") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```



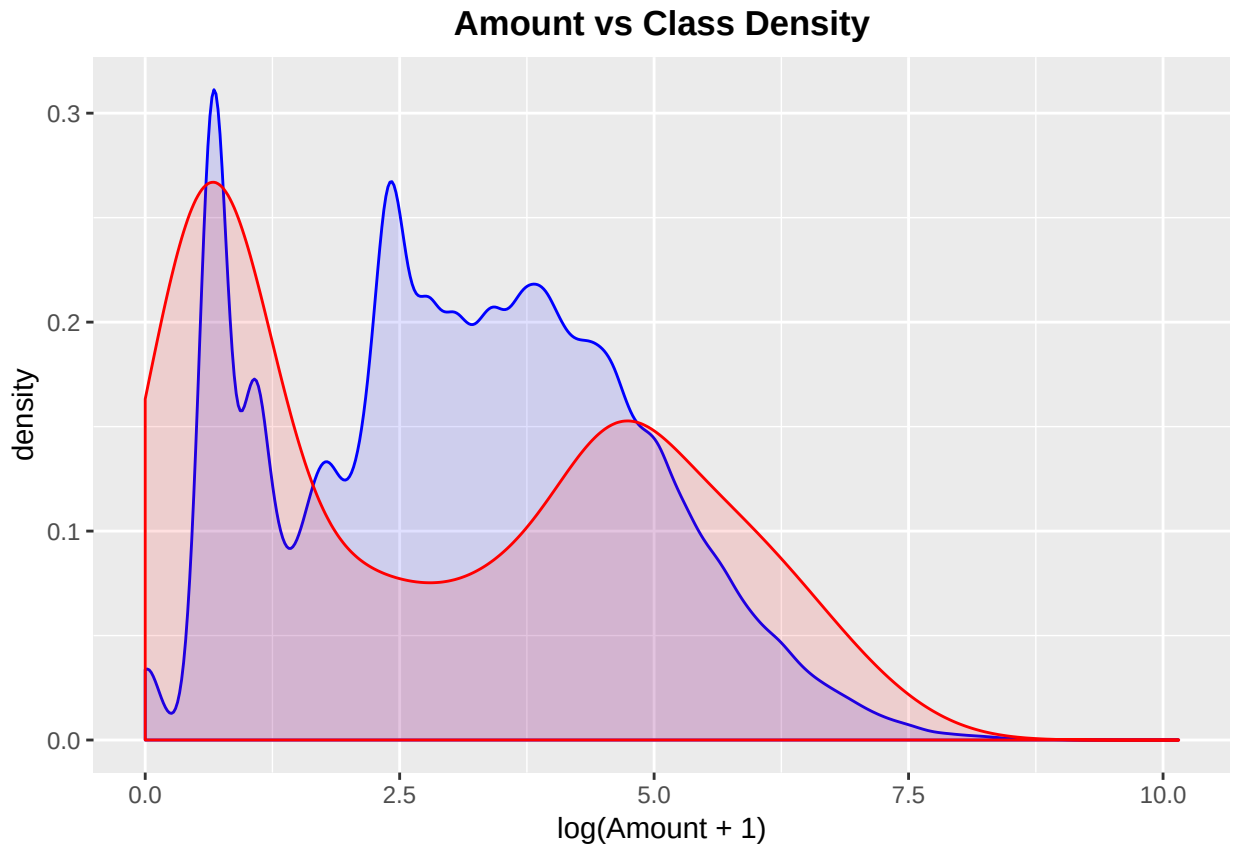
AMOUNT VARIABLE

```
summary(creditcard$Amount)
```

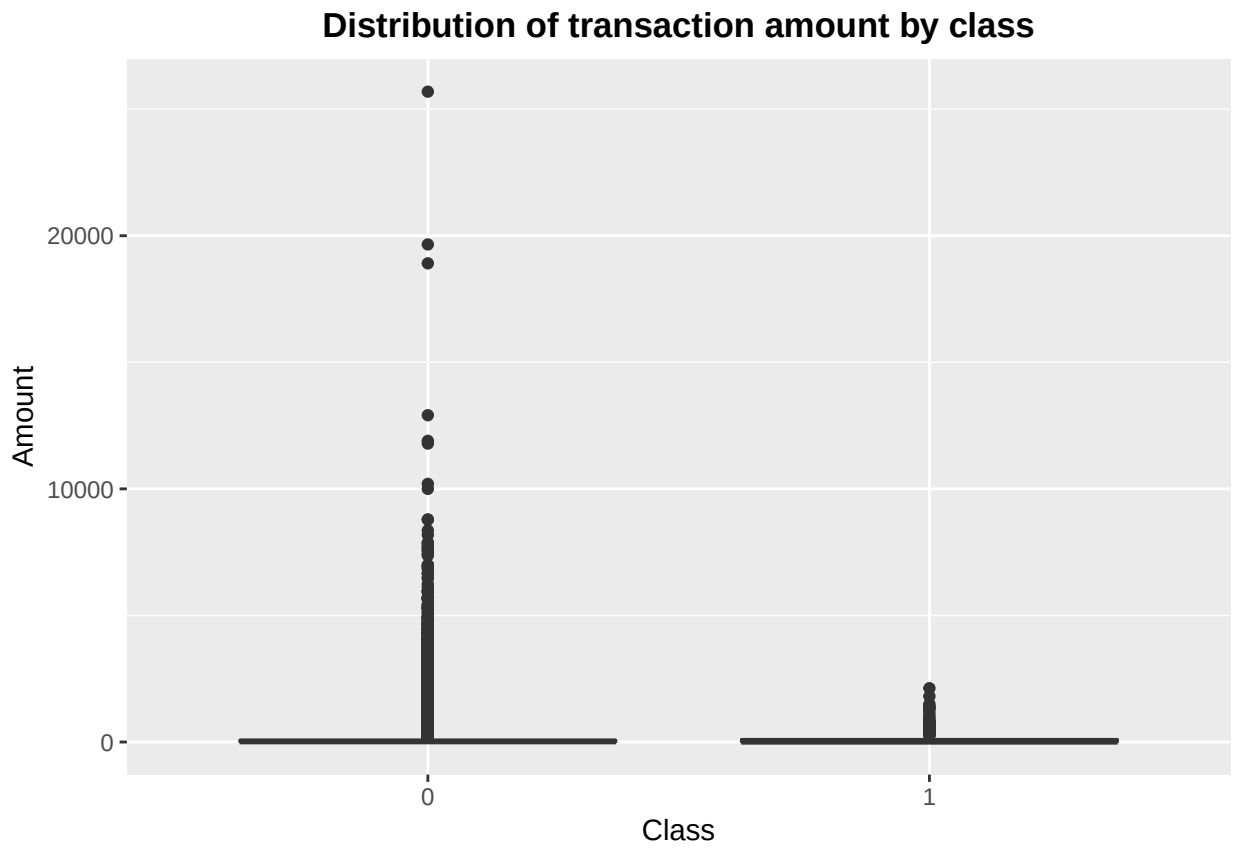
```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##      0.00      5.60     22.00     88.35     77.17 25691.16
```

From mean and maximum value of Amount, there is heavy skew observed in the dataset. Taking logarithmic values to observe the trend. Using (Amount+1) to avoid removal of out to bound values.

```
ggplot() +
  geom_density(data = creditcard[creditcard$Class == 0,],
               aes(x=log(Amount+1)), color = "blue",
               fill = "blue", alpha = 0.12) +
  geom_density(data = creditcard[creditcard$Class == 1,],
               aes(x=log(Amount+1)), color = "red",
               fill = "red", alpha = 0.12) +
  ggtitle("Amount vs Class Density") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```




```
ggplot(creditcard, aes(x = Class, y = Amount)) +
  geom_boxplot() +
  ggtitle("Distribution of transaction amount by class") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```



TIME VARIABLE

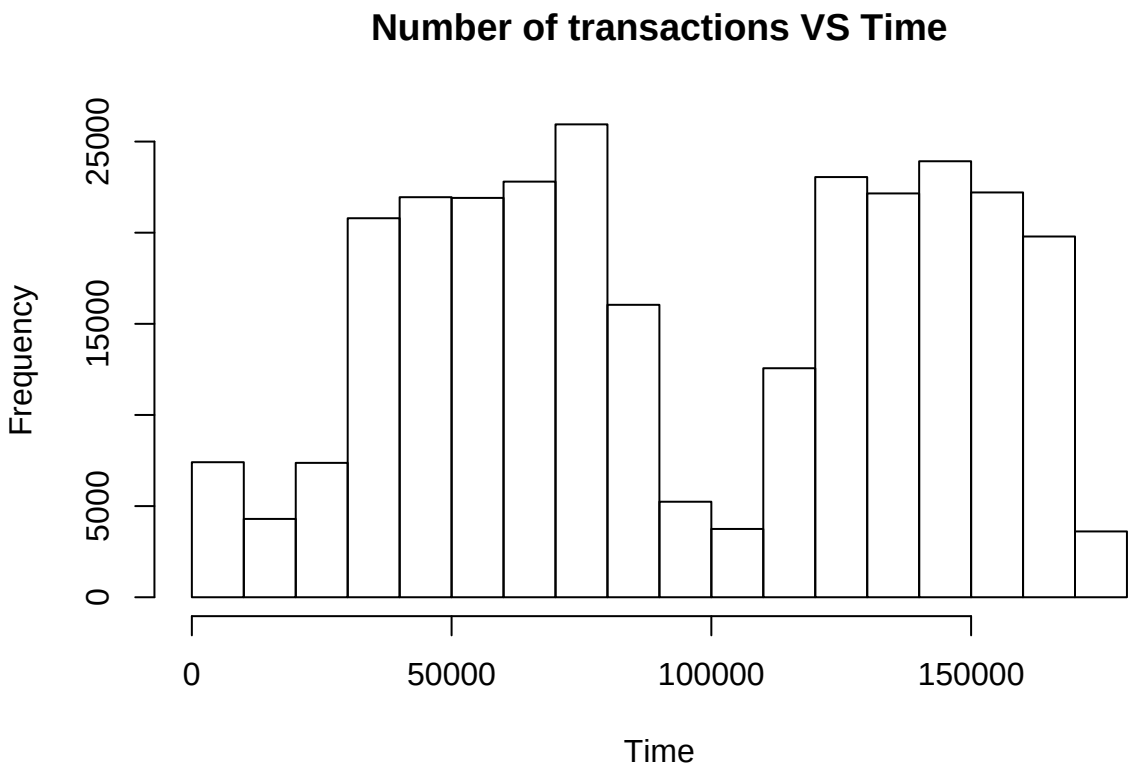
```
summary(creditcard$Time)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0   54202   84692   94814  139321  172792
```

```
#Obtaining number of days
max(creditcard$Time)/(60*60*24)
```

```
## [1] 1.999907
```

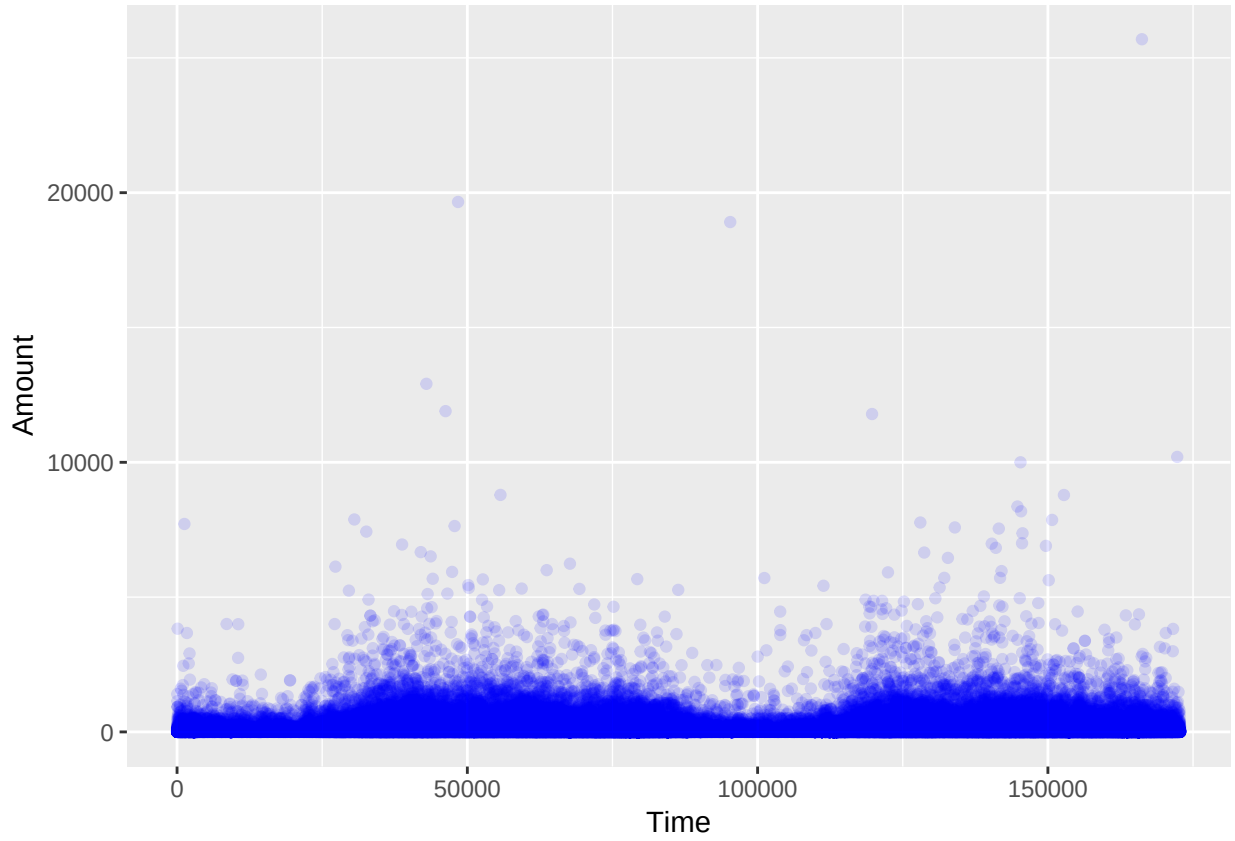
```
hist(creditcard$Time, xlab = "Time", main = "Number of transactions VS Time")
```



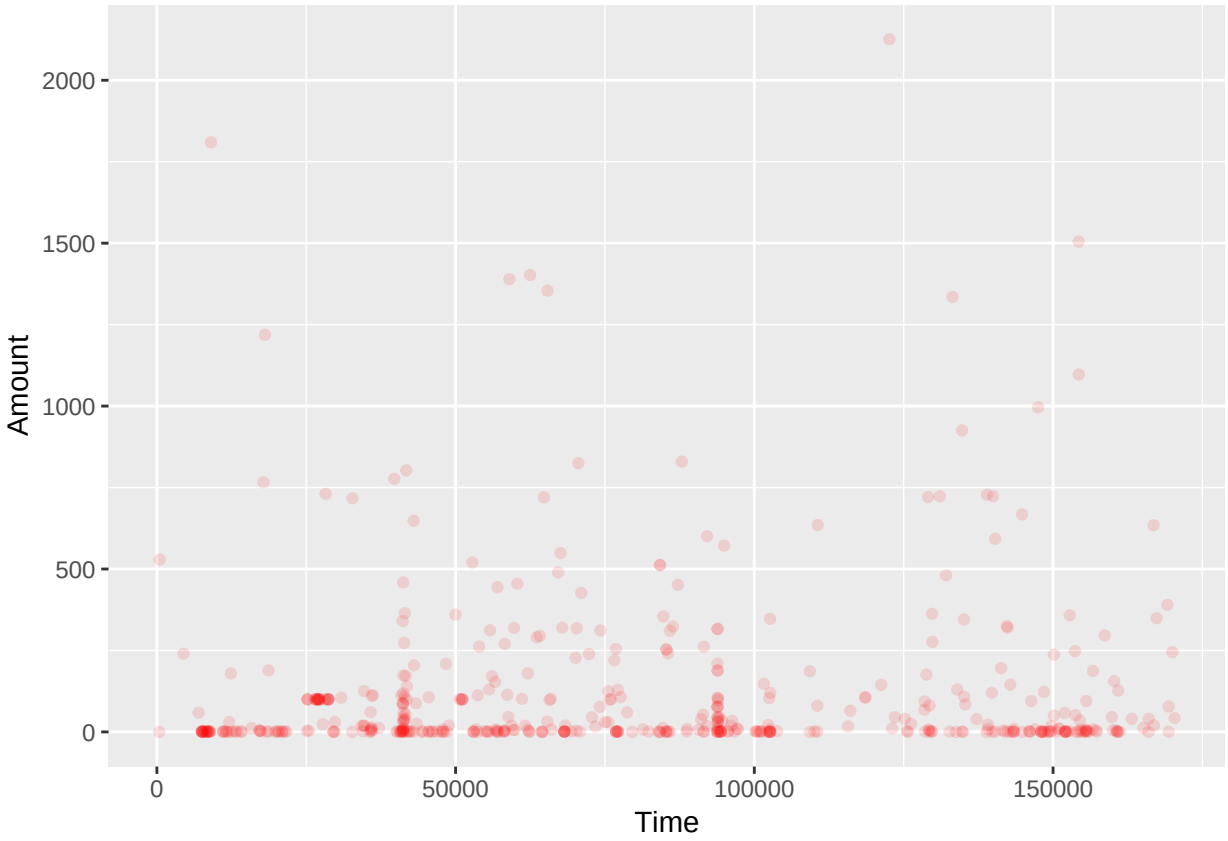
Time values are in the form of seconds past after the first transaction. The dataset available is for 2 days. From the frequency plot, there is a periodic variation for legitimate transaction.

```
#Separating frau and legit class
Fraud <- creditcard[creditcard$Class==1,]
Legit <- creditcard[creditcard$Class==0,]

#Assuming, first transaction (Time=0) took place at 12:00 am
ggplot() +
  geom_point(data=Legit, aes(Time,Amount),
             color="blue",alpha=0.12)
```

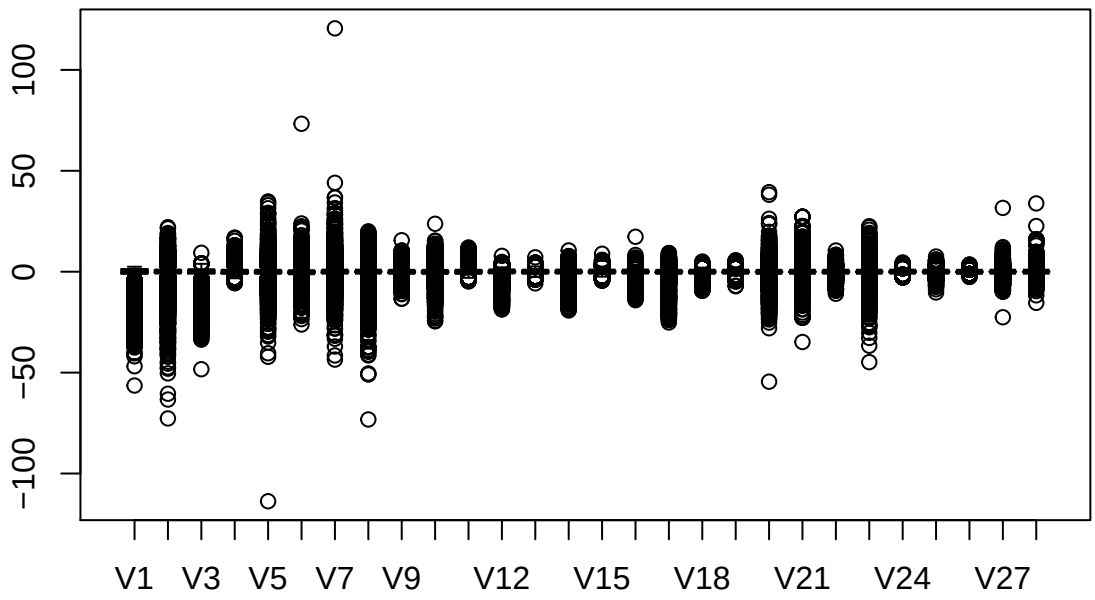



```
ggplot() +
  geom_point(data=Fraud, aes(Time,Amount),
             color="red", alpha=0.12)
```



Detection of Outliers

```
#Outliers
boxplot(creditcard[,-c(1,31,30)],
        col = rainbow(28, alpha = 0.3))
```




```
#Detecting outlier percentage
OutlierPercentage = function(dataframe){
  num=sapply(dataframe, is.numeric)
  NumOut = function(x){
    q1 = quantile(x, 0.25, na.rm = T)
    q2 = quantile(x, 0.50, na.rm = T)
    q3 = quantile(x, 0.75, na.rm = T)
    iqr = q3-q1
    outlier = x[x > q3 + 1.5*iqr | x < q1 - 1.5*iqr]
    percentage = (length(outlier) / length(x))*100
    return(percentage)
  }
  data=sapply(dataframe[,num], NumOut)
   return(as.data.frame(data))
}
OutlierPercentage(creditcard)
```

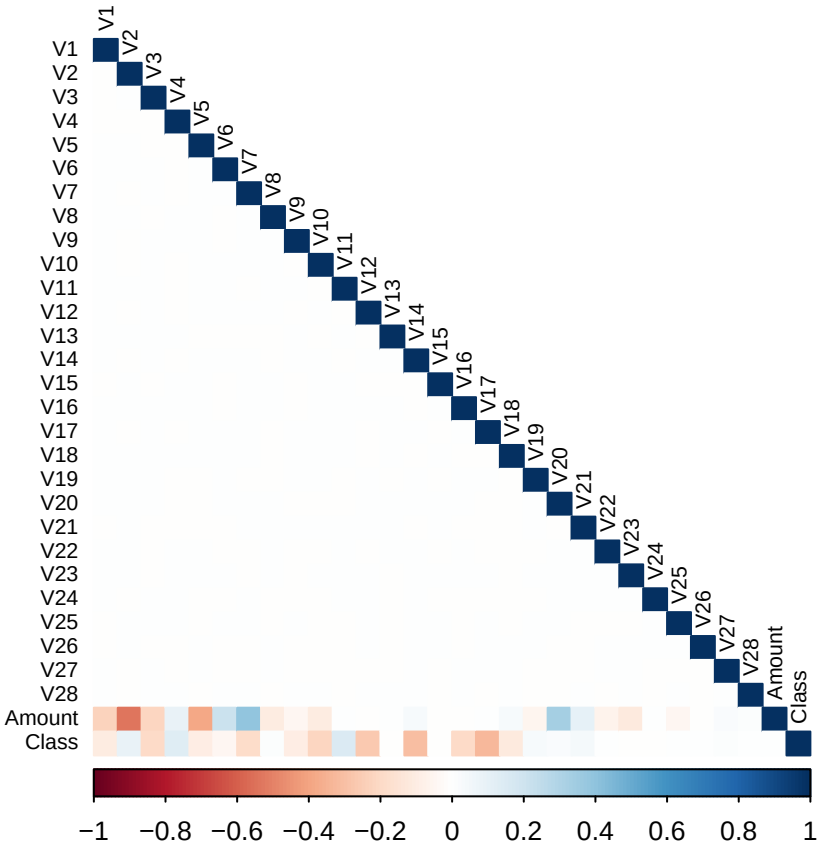
```
##              data
## Time    0.0000000
## V1      2.4795739
## V2      4.7491810
## V3      1.1807996
## V4      3.9142296
## V5      4.3169585
## V6      8.0633552
## V7      3.1417767
```

```
## V8      8.4738086
## V9      2.9082853
## V10     3.3341877
## V11     0.2738697
## V12     5.3889125
## V13     1.1825552
## V14     4.9679256
## V15     1.0161267
## V16     2.8735249
## V17     2.6052730
## V18     2.6449490
## V19     3.5831282
## V20     9.7504626
## V21     5.0901137
## V22     0.4624184
## V23     6.5100226
## V24     1.6762228
## V25     1.8844340
## V26     1.9648393
## V27    13.7507154
## V28    10.6535303
## Amount 11.2019719
```

Removing of outliers depends on the model, in the feature selection stage. Not all models are sensitive to outliers. Hence, we shall deal with them later.

Correlation/Collinearity Analysis

```r
#Correlation plot
corrplot(cor(creditcard.raw[,-c(1)]),
         method = "color", type = "lower",
         tl.col = "black", tl.cex = 0.7)
```



As the predictors V1 to V28 are principal components, there is no correlation betwwen them. But few predictors are correlated to the Class and Amount variables.

Data Cleaning and Shaping

Data Imputaion

We have a clean data with no missing values. Hence, we will remove a few values and impute them to construct a model.

```
set.seed(123)
credit.Impute <- as.data.frame(creditcard)
summary(credit.Impute)
```

```
##       Time              V1                   V2
##  Min.   :     0   Min.   :-56.40751   Min.   :-72.71573
##  1st Qu.: 54202   1st Qu.: -0.92037   1st Qu.: -0.59855
##  Median : 84692   Median :  0.01811   Median :  0.06549
##  Mean   : 94814   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.:139321   3rd Qu.:  1.31564   3rd Qu.:  0.80372
##  Max.   :172792   Max.   :  2.45493   Max.   : 22.05773
##        V3                  V4                   V5
##  Min.   :-48.3256   Min.   :-5.68317   Min.   :-113.74331
##  1st Qu.: -0.8904   1st Qu.:-0.84864   1st Qu.:  -0.69160
```

```
##  Median :  0.1799   Median :-0.01985   Median :  -0.05434
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :   0.00000
##  3rd Qu.:  1.0272   3rd Qu.: 0.74334   3rd Qu.:   0.61193
##  Max.   :  9.3826   Max.   :16.87534   Max.   :  34.80167
##        V6                 V7                 V8
##  Min.   :-26.1605   Min.   :-43.5572   Min.   :-73.21672
##  1st Qu.: -0.7683   1st Qu.: -0.5541   1st Qu.: -0.20863
##  Median : -0.2742   Median :  0.0401   Median :  0.02236
##  Mean   :  0.0000   Mean   :  0.0000   Mean   :  0.00000
##  3rd Qu.:  0.3986   3rd Qu.:  0.5704   3rd Qu.:  0.32735
##  Max.   : 73.3016   Max.   :120.5895   Max.   : 20.00721
##        V9                 V10                V11
##  Min.   :-13.43407   Min.   :-24.58826   Min.   :-4.79747
##  1st Qu.: -0.64310   1st Qu.: -0.53543   1st Qu.:-0.76249
##  Median : -0.05143   Median : -0.09292   Median :-0.03276
##  Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.59714   3rd Qu.:  0.45392   3rd Qu.: 0.73959
##  Max.   : 15.59500   Max.   : 23.74514   Max.   :12.01891
##       V12                V13                V14
##  Min.   :-18.6837   Min.   :-5.79188   Min.   :-19.2143
##  1st Qu.: -0.4056   1st Qu.:-0.64854   1st Qu.: -0.4256
##  Median :  0.1400   Median :-0.01357   Median :  0.0506
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :  0.0000
##  3rd Qu.:  0.6182   3rd Qu.: 0.66251   3rd Qu.:  0.4931
##  Max.   :  7.8484   Max.   : 7.12688   Max.   : 10.5268
##       V15                V16                V17
##  Min.   :-4.49894   Min.   :-14.12985   Min.   :-25.16280
##  1st Qu.:-0.58288   1st Qu.: -0.46804   1st Qu.: -0.48375
##  Median : 0.04807   Median :  0.06641   Median : -0.06568
##  Mean   : 0.00000   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.: 0.64882   3rd Qu.:  0.52330   3rd Qu.:  0.39968
##  Max.   : 8.87774   Max.   : 17.31511   Max.   :  9.25353
##       V18                V19                V20
##  Min.   :-9.498746   Min.   :-7.213527   Min.   :-54.49772
##  1st Qu.:-0.498850   1st Qu.:-0.456299   1st Qu.: -0.21172
##  Median :-0.003636   Median : 0.003735   Median : -0.06248
##  Mean   : 0.000000   Mean   : 0.000000   Mean   :  0.00000
##  3rd Qu.: 0.500807   3rd Qu.: 0.458949   3rd Qu.:  0.13304
##  Max.   : 5.041069   Max.   : 5.591971   Max.   : 39.42090
##       V21                V22                V23
##  Min.   :-34.83038   Min.   :-10.933144   Min.   :-44.80774
##  1st Qu.: -0.22839   1st Qu.: -0.542350   1st Qu.: -0.16185
##  Median : -0.02945   Median :  0.006782   Median : -0.01119
##  Mean   :  0.00000   Mean   :  0.000000   Mean   :  0.00000
##  3rd Qu.:  0.18638   3rd Qu.:  0.528554   3rd Qu.:  0.14764
##  Max.   : 27.20284   Max.   : 10.503090   Max.   : 22.52841
##       V24                V25                V26
##  Min.   :-2.83663   Min.   :-10.29540   Min.   :-2.60455
##  1st Qu.:-0.35459   1st Qu.: -0.31715   1st Qu.:-0.32698
##  Median : 0.04098   Median :  0.01659   Median :-0.05214
##  Mean   : 0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.: 0.43953   3rd Qu.:  0.35072   3rd Qu.: 0.24095
##  Max.   : 4.58455   Max.   :  7.51959   Max.   : 3.51735
##       V27                V28                Amount         Class
```

```
##  Min.   :-22.565679   Min.   :-15.43008   Min.   :    0.00   0:284315
##  1st Qu.: -0.070840   1st Qu.: -0.05296   1st Qu.:    5.60   1:   492
##  Median :  0.001342   Median :  0.01124   Median :   22.00
##  Mean   :  0.000000   Mean   :  0.00000   Mean   :   88.35
##  3rd Qu.:  0.091045   3rd Qu.:  0.07828   3rd Qu.:   77.17
##  Max.   : 31.612198   Max.   : 33.84781   Max.   :25691.16
```

```
#Introducing missing values randomly
for(i in 1:29){
credit.Impute[c(sample(1:nrow(creditcard),20)),i] <- NA
}
#Number of missing values
sum(is.na(credit.Impute))
```

```
## [1] 580
```

As the features are normalized to a mean of 0, we can impute the values with median

```
for(i in 1:29){
credit.Impute[c(which(is.na(credit.Impute[,i]))),i] <- median(credit.Impute[,i], na.rm = T)
}

#Checking missing values
sum(is.na(credit.Impute))
```

```
## [1] 0
```

Normalization of feature values

Features V1 to V28 are normalised to mean = 0. We also normalize the variable Amount using min-max.

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
credit.Impute$Amount <- normalize(credit.Impute$Amount)
```

Dummy code

As there is no categorical variable in the dataset, it is not possible perform dummy coding.

PCA

The variables V1 - V28 are said to be in the form of principal components, thus we can perform PCA on them to verify the same.

```
credit.pca <- prcomp(credit.Impute[,-c(1,30,31)], center = T)

#Comparing the variable and its pca values
abs(head(credit.Impute[,c(2,7,10,25)]))
```

```
##          V1         V6        V9        V24
## 1 1.3598071 0.46238778 0.3637870 0.06692807
## 2 1.1918571 0.08236081 0.2554251 0.33984648
## 3 1.3583541 1.80049938 1.5146543 0.68928096
## 4 0.9662717 1.24720317 1.3870241 1.17557533
## 5 1.1582331 0.09592146 0.8177393 0.14126698
## 6 0.4259659 0.02972755 0.5686714 0.37142658
```

```
abs(head(credit.pca$x[,c(1,6,9,24)]))
```

```
##            PC1        PC6       PC9       PC24
## [1,] 1.3595827 0.46269352 0.3635559 0.06674334
## [2,] 1.1917483 0.08227328 0.2557858 0.33995770
## [3,] 1.3578794 1.80023038 1.5147385 0.68976132
## [4,] 0.9660318 1.24703515 1.3875554 1.17566922
## [5,] 1.1582241 0.09613915 0.8188297 0.14114123
## [6,] 0.4261385 0.03028132 0.5695030 0.37144309
```

```
which(round(abs(head(credit.Impute[,c(2,7,10,25)])),
            digits = 2) != round(abs(head(credit.pca$x[,c(1,6,9,24)])),
                                 digits = 2))
```

```
## integer(0)
```

The variable and their PCA values are equal. Hence, the dataset is already dimensionally reduced.

Data Sampling

Due to imbalanced nature of the dataset, the classification algorithms tend to be biased towards the majority class. Hence, we will sample the dataset using SMOTE (up-sample)

```
#Over-sampling the data
credit.data <- SMOTE(Class~.,credit.Impute,
                     perc.over = 900, perc.under = 150)
table(credit.data$Class)
```

```
##
##    0    1
## 6642 4920
```

```
ggplot(credit.data, aes(x = Class, fill = Class)) +
  geom_bar(alpha = 0.4) +
  ggtitle("Histogram of Class variable") +
  scale_fill_manual(values = c("blue","red")) +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

Thus, we obtain a balanced subset of the dataset.

Model Construction and Evaluation

Training and Validation Subsets

```
#Creating 80-20 split on sampled dataset
set.seed(123)
split.index <- sample(seq_len(nrow(credit.data)),
                      size = floor(0.8 * nrow(credit.data)))
credit.train <- credit.data[split.index,]
credit.test <- credit.data[-split.index,-31]
#Storing class output differently
credit.test.target <- as.data.frame(credit.data[-split.index,31])
colnames(credit.test.target)[1] <- c("Class")
```

Models

For this Classification problem, the models used are: Decision Tree Random Forest K Nearest-Neighbour

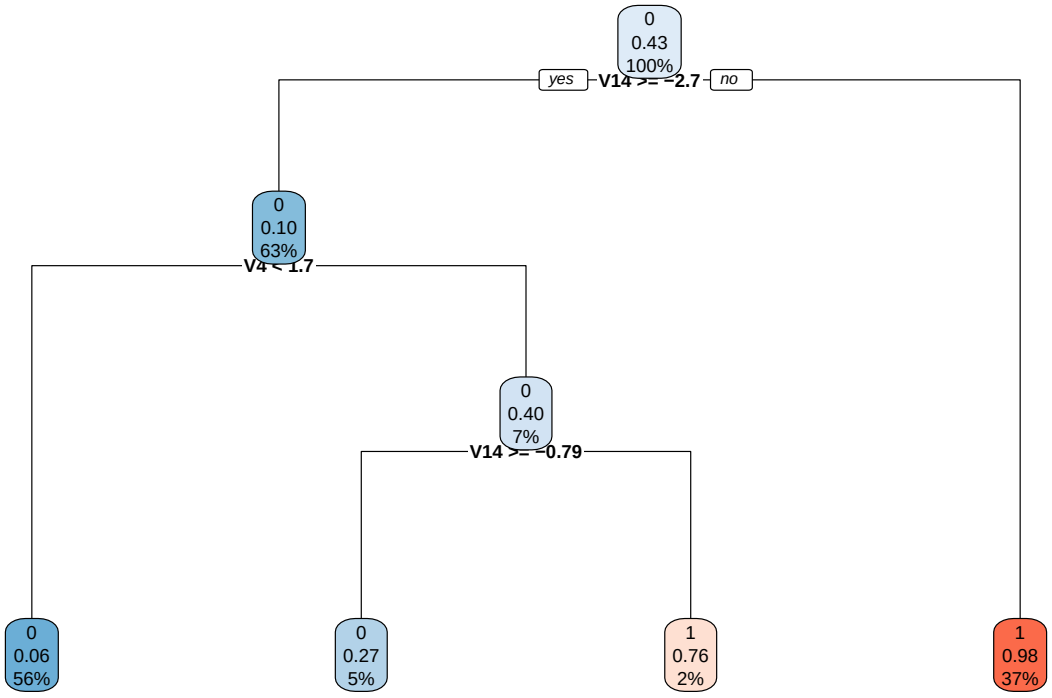
Decision Tree

```
#Building the model
mod.DT <- rpart(Class ~ .,data = credit.train, method = "class")

##  Evaluation: Holdout method
mod.DT.Pred <- predict(mod.DT, credit.test)
credit.test.target$Pred.DT <- as.factor(ifelse(mod.DT.Pred[,2]>0.5,1,0))
conf.DT <- confusionMatrix(credit.test.target$Pred.DT,
                           credit.test.target$Class, positive = '1')
conf.DT
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1301  113
##          1   30  869
##
##                Accuracy : 0.9382
##                  95% CI : (0.9276, 0.9476)
##     No Information Rate : 0.5754
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8721
##
##  Mcnemar's Test P-Value : 7.023e-12
##
##             Sensitivity : 0.8849
##             Specificity : 0.9775
##          Pos Pred Value : 0.9666
##          Neg Pred Value : 0.9201
##              Prevalence : 0.4246
##          Detection Rate : 0.3757
##    Detection Prevalence : 0.3887
##       Balanced Accuracy : 0.9312
##
##        'Positive' Class : 1
##
```

```
#Plot the tree
rpart.plot(mod.DT, cex = 0.6, box.palette = "BuRd")
```

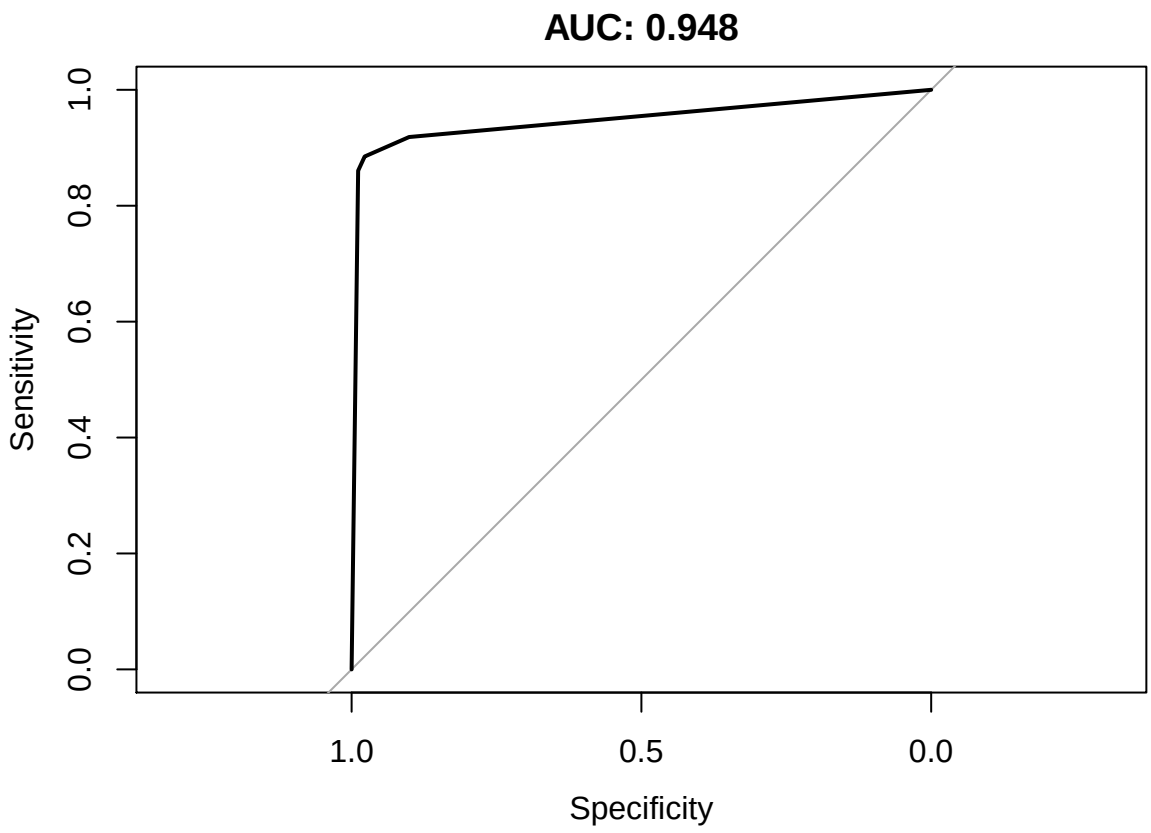



```
#AUC and ROC
roc.DT <- pROC::roc(credit.test.target$Class, mod.DT.Pred[,1])
```

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

```
## Setting direction: controls > cases
```

```
plot(roc.DT, main = paste0("AUC: ", round(pROC::auc(roc.DT), 3)))
```




```
auc.DT <- round(pROC::auc(roc.DT), 3)

#Model Accuracy
mod.DT.Acc <- conf.DT$overall[1]*100
mod.DT.Acc
```

```
## Accuracy
## 93.81755
```

Random Forest

```
#Building the model
mod.RF <- randomForest(Class~., data = credit.train)

##  Evaluation: Holdout method
mod.RF.Pred <- predict(mod.RF, credit.test)
credit.test.target$Pred.RF <- as.factor(mod.RF.Pred)
conf.RF <- confusionMatrix(credit.test.target$Pred.RF,
                           credit.test.target$Class, positive = '1')
conf.RF
```

```
## Confusion Matrix and Statistics
##
```

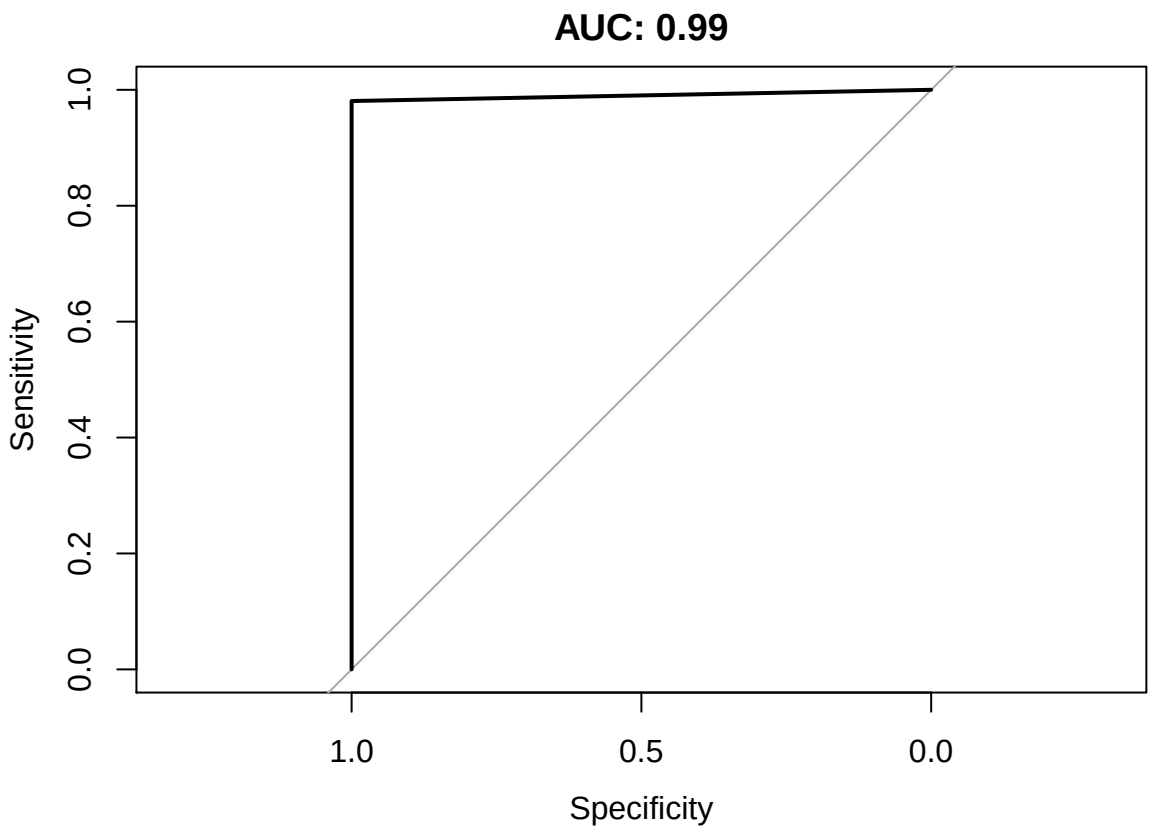
```
##           Reference
## Prediction    0    1
##          0 1331   19
##          1    0  963
##
##                Accuracy : 0.9918
##                  95% CI : (0.9872, 0.995)
##     No Information Rate : 0.5754
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9831
##
##  Mcnemar's Test P-Value : 3.636e-05
##
##             Sensitivity : 0.9807
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9859
##              Prevalence : 0.4246
##          Detection Rate : 0.4163
##    Detection Prevalence : 0.4163
##       Balanced Accuracy : 0.9903
##
##        'Positive' Class : 1
##
```

```
credit.test.target$Pred.RF <- ordered(credit.test.target$Pred.RF, levels = c("0","1"))
#AUC and ROC
roc.RF <- pROC::roc(credit.test.target$Class, credit.test.target$Pred.RF)
```

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

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

```
plot(roc.RF, main = paste0("AUC: ", round(pROC::auc(roc.RF), 3)))
```




```
auc.RF <- round(pROC::auc(roc.RF), 3)

#Model Accuracy
mod.RF.Acc <- conf.RF$overall[1]*100
mod.RF.Acc
```

```
## Accuracy
## 99.17856
```

K Nearest neighbours

```
#Building the model
kVal <- round(sqrt(nrow(credit.train)))
mod.KNN <- knn3(Class~.,data = credit.train)

##  Evaluation: Holdout method
mod.KNN.Pred <- predict(mod.KNN, credit.test)
credit.test.target$Pred.KNN <- as.factor(ifelse(mod.KNN.Pred[,2]>0.5,1,0))
conf.KNN <- confusionMatrix(credit.test.target$Pred.KNN,
                            credit.test.target$Class, positive = '1')
conf.KNN
```

```
## Confusion Matrix and Statistics
```

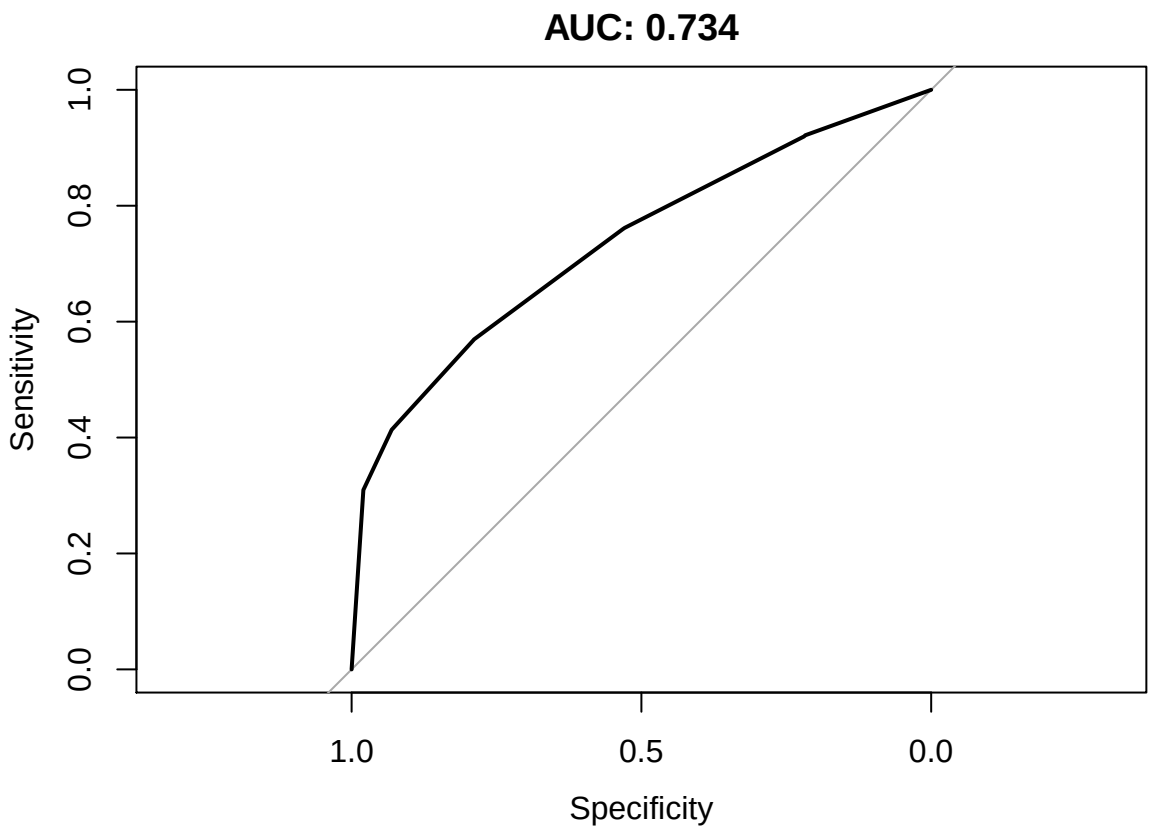
```
##
##           Reference
## Prediction    0    1
##          0 1051  424
##          1  280  558
##
##                Accuracy : 0.6956
##                  95% CI : (0.6764, 0.7143)
##     No Information Rate : 0.5754
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3649
##
##  Mcnemar's Test P-Value : 7.065e-08
##
##             Sensitivity : 0.5682
##             Specificity : 0.7896
##          Pos Pred Value : 0.6659
##          Neg Pred Value : 0.7125
##              Prevalence : 0.4246
##          Detection Rate : 0.2412
##    Detection Prevalence : 0.3623
##       Balanced Accuracy : 0.6789
##
##        'Positive' Class : 1
##
```

```
#AUC and ROC
roc.KNN <- pROC::roc(credit.test.target$Class, mod.KNN.Pred[,1])
```

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

```
## Setting direction: controls > cases
```

```
plot(roc.KNN, main = paste0("AUC: ", round(pROC::auc(roc.KNN), 3)))
```




```
auc.KNN <- round(pROC::auc(roc.KNN), 3)

#Model Accuracy
mod.KNN.Acc <- conf.KNN$overall[1]*100
mod.KNN.Acc
```

```
## Accuracy
## 69.56334
```

Tuning of Models and Cross Validation

```
# Formatting the data
credit.tr <- credit.train
levels(credit.tr$Class) <- make.names(levels(credit.tr$Class))
levels(credit.test.target$Class) <- make.names(levels(credit.test.target$Class))

# Setting up train control
train.controll <- trainControl(method = "cv",
                               number = 5,
                               classProbs = T,
                               summaryFunction = twoClassSummary)

## Decision tree
```

```
DT.grid <- expand.grid(maxdepth = 3:10)
#Get the best model
DT.best.model <- train(Class ~ ., method = "rpart2",
                       data = credit.tr,
                       tuneGrid = DT.grid,
                       trControl = train.controll,
                       metric="ROC")


## Random Forest
RF.grid <- expand.grid(mtry=sqrt(ncol(credit.tr)))
ntrees <- c(10, 50, 100, 150, 200)
RF.best.model <- data.frame(mtry=c(), ROC=c(), Sens=c(), ROCSD=c(), SensSD=c(), SpecSD=c())
for(ntree in ntrees) {
  rf_models <- train(Class ~ .,
                     method = "rf",
                     data = credit.tr,
                     tuneGrid = RF.grid,
                     trControl = train.controll,
                     metric="ROC",
                     ntree = ntree)
  RF.best.model <- rbind(RF.best.model, as.matrix(rf_models$result))
}
row.names(RF.best.model) <- ntrees
#Get the best ntree
best.ntree <- as.integer(row.names
                         (RF.best.model[which.max(RF.best.model$ROC), ]))
#Get the best model
RF.best.model <- train(Class ~ .,
                       method = "rf",
                       data = credit.tr,
                       tuneGrid = RF.grid,
                       ntree = best.ntree)



## K Nearest neighbour
KNN.grid <- expand.grid(k = c(5, 11, 21, 25))
#Get the best model
KNN.best.model <- train(Class ~ .,
                        method = "knn",
                        data = credit.tr,
                        tuneGrid = KNN.grid,
                        trControl = train.controll,
                        metric="ROC")
```

```
# Testing individual tuned models
#KNN
pred_knn_tr <- predict(KNN.best.model, credit.tr[,-length(colnames(credit.train))])
tune.KNN <- predict(KNN.best.model, credit.test[,-length(colnames(credit.train))])
conf.tune.KNN <- confusionMatrix(tune.KNN, credit.test.target$Class, positive = "X1")
conf.tune.KNN
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction   X0   X1
##         X0 1102  504
##         X1  229  478
##
##                Accuracy : 0.6831
##                  95% CI : (0.6637, 0.702)
##     No Information Rate : 0.5754
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3267
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.4868
##             Specificity : 0.8279
##          Pos Pred Value : 0.6761
##          Neg Pred Value : 0.6862
##              Prevalence : 0.4246
##          Detection Rate : 0.2067
##    Detection Prevalence : 0.3057
##       Balanced Accuracy : 0.6574
##
##        'Positive' Class : X1
##
```

```
#Random Forest
pred_rf_tr <- predict(RF.best.model, credit.tr[,-length(colnames(credit.train))])
tune.RF <- predict(RF.best.model, credit.test[,-length(colnames(credit.train))])
conf.tune.RF <- confusionMatrix(tune.RF, credit.test.target$Class,positive = "X1")
conf.tune.RF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   X0   X1
##         X0 1331   18
##         X1    0  964
##
##                Accuracy : 0.9922
##                  95% CI : (0.9877, 0.9954)
##     No Information Rate : 0.5754
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.984
##
##  Mcnemar's Test P-Value : 6.151e-05
##
##             Sensitivity : 0.9817
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9867
##              Prevalence : 0.4246
```

```
##          Detection Rate : 0.4168
##    Detection Prevalence : 0.4168
##       Balanced Accuracy : 0.9908
##
##        'Positive' Class : X1
##
```

```
#DEcision Tree
pred_dt_tr <- predict(DT.best.model, credit.tr[,-length(colnames(credit.train))])
tune.DT <- predict(DT.best.model, credit.test[,-length(colnames(credit.train))])
conf.tune.DT<- confusionMatrix(tune.DT, credit.test.target$Class,positive = "X1")
conf.tune.DT
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   X0   X1
##         X0 1301  113
##         X1   30  869
##
##                Accuracy : 0.9382
##                  95% CI : (0.9276, 0.9476)
##     No Information Rate : 0.5754
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8721
##
##  Mcnemar's Test P-Value : 7.023e-12
##
##             Sensitivity : 0.8849
##             Specificity : 0.9775
##          Pos Pred Value : 0.9666
##          Neg Pred Value : 0.9201
##              Prevalence : 0.4246
##          Detection Rate : 0.3757
##    Detection Prevalence : 0.3887
##       Balanced Accuracy : 0.9312
##
##        'Positive' Class : X1
##
```

Comparison of Models

```
#Comparing the accuracies and AUC for all 3 models

Compare.models <- data.frame(
  Accuracy = c(mod.DT.Acc,mod.RF.Acc,mod.KNN.Acc),
  Tuned_Accuracy = c(conf.tune.DT$overall[1] * 100,
                     conf.tune.RF$overall[1] * 100,
                     conf.tune.KNN$overall[1] * 100),
  Area_Under_Curve  = c(auc.DT * 100,auc.RF * 100,
                        auc.KNN * 100) ,
```

```
  Sensitivity = c(conf.DT$byClass[1] * 100,
                  conf.RF$byClass[1] * 100,
                  conf.KNN$byClass[1] * 100)
  )

rownames(Compare.models) <-
  c("Decision Tree","Random Forest","K Nearest Neighbour")
Compare.models
```

```
##                       Accuracy Tuned_Accuracy Area_Under_Curve Sensitivity
## Decision Tree         93.81755       93.81755             94.8    88.49287
## Random Forest         99.17856       99.22179             99.0    98.06517
## K Nearest Neighbour 69.56334       68.30955             73.4    56.82281
```

Interpretation of Results

The metrics used to evaluate these models are Accuracy and Sensitivity. It can be seen that Random Forest outperforms all the other models with an accuracy of 99% and sensitivity of 98% in the test set. KNN performs poorly because of the significant overlap in feature distributions for the positive and negative classes. Since the features in the dataset are already a result of dimensionality reduction, the models do not have good interpretability.

Model stacking is a widely used approach to further improve model performance. A mode stacking approach can be tried to develope a better model.

Stacked Ensemble Model

```
# Model Stacking
credit_ensemble <- credit.tr
credit_ensemble$pred_knn <- pred_knn_tr
credit_ensemble$pred_rf <- pred_rf_tr
credit_ensemble$pred_dt <- pred_dt_tr

#Training stacked model using Logistic Regression
stacked_model <- train(Class ~ .,
                       data = credit_ensemble,
                       method="glm",
                       maxit=50)

# Testing Stacked Model
pred_knn_test <- predict(KNN.best.model, credit.test)
pred_rf_test <- predict(RF.best.model, credit.test)
pred_dt_test <- predict(DT.best.model, credit.test)
credit_ensemble_test <- credit.test
credit_ensemble_test$pred_knn <- pred_knn_test
credit_ensemble_test$pred_rf <- pred_rf_test
credit_ensemble_test$pred_dt <- pred_dt_test
pr_stacked <- predict(stacked_model, credit_ensemble_test)
prob_stacked <- predict(stacked_model,
                        credit_ensemble_test, type="prob")
```

```
roc_stacked <- pROC::roc(credit.test.target$Class, prob_stacked[,1])
```

```
## Setting levels: control = X0, case = X1
```

```
## Setting direction: controls > cases
```

```
conf.stack <- confusionMatrix(pr_stacked,
                              credit.test.target$Class,positive = "X1")
stacked_perf <- data.frame(Accuracy = conf.stack$overall[1]*100,
                           Sensitivity = conf.stack$byClass[1]*100,
                           AUC = pROC::auc(roc_stacked) * 100)
row.names(stacked_perf) <- "Stacked Logistic Regression"
stacked_perf
```

```
##                              Accuracy Sensitivity     AUC
## Stacked Logistic Regression 99.22179    98.16701 99.0835
```

It can be seen that the stacked model does not affect the performance of the best performing model (Random Forest). Hence, in our case, we will not be using a stacked model.