Credit Card Fraud Detection

Project Report

Submitted to:

Prof. Dr. Gopikrishnan S

Presented by:

L BHARADHWAJ REDDY	19BCD7047
REDDYBATHINA NAGA SAI RAM	19BCD7052
B N V R S ROHITH	19BCD7033
D V L SAI SRUTHI	19BCD7040

INDEX

1.INTRODUCTION

A credit card is a thin handy plastic card that contains identification information such as a signature or picture, and authorizes the person named on it to charge purchases or services to his account – charges for which he will be billed periodically. Today, the information on the card is read by automated teller machines (ATMs), store readers, bank and is also used in online internet banking system. They have a unique card number which is of utmost importance. Its security relies on the physical security of the plastic card as well as the privacy of the credit card number. There is a rapid growth in the number of credit card transactions which has led to a substantial rise in fraudulent activities. Credit card fraud is a wide-ranging term for theft and fraud committed using a credit card as a fraudulent source of funds in a given transaction.

2.OBJECTIVE

The key objective of any credit card fraud detection system is to identify suspicious events and report them to an analyst while letting normal transactions be automatically processed.

For years, financial institutions have been entrusting this task to rule-based systems that employ rule sets written by experts. But now they increasingly turn to a machine learning approach, as it can bring significant improvements to the process.

3.PROBLEM STATEMENT

The Credit Card Fraud Detection Problem includes modeling past credit card transactions with the knowledge of the ones that turned out to be a fraud. This model is then used to identify whether a new transaction is fraudulent or not. Our aim here is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications.

4.METHODOLOGY

Credit card fraud is increasing considerably with the development of modern technology and the global superhighways of communication. Credit card fraud costs consumers and the financial company billions of dollars annually, and fraudsters continuously try to find new rules and tactics to commit illegal actions. Thus, fraud detection systems have become essential for banks and financial institution, to minimize their losses. However, there is a lack of published literature on credit card fraud detection techniques, due to the unavailable credit card transactions dataset for researchers. We have used these ML algorithms Logistic Regression, Decision Tree, Artificial Neural Networks, Gradient Boosting Classifier. In this paper we trained various data mining techniques used in credit card fraud detection and evaluated each methodology based on certain design criteria. We first Cleaned the dataset and then analysed using the above mentioned ML Algorithms.

5.RESULTS OBTAINED

5.1 Cleaning of data

```
#Class of data object
class(credit_card)

#Display Internal structure of data
str(credit_card)

#Summary of data
summary(credit_card)

#Column names
names(credit_card)

#Dimensions of data
dim(credit_card)

# Data of the top
head(credit_card)

#Data from the top
tail(credit_card)
```

```
#Loading the creditcardfraud csv file into a data frame
credit_card=read.csv("creditcardfraud.csv")
credit_card
#Checking if there are any NA values in the dataset
any(is.na(credit_card))
# So from the output it is understood that there are NA values in the dataset
#Let us extract the count of NA values in the dataset
sum(is.na(credit_card))

#Replacing the NA values in each column with the mean value of the values in
#the column
credit_card$Time[is.na(credit_card$Time)]=mean(credit_card$Time,na.rm=TRUE)
credit_card$V1[is.na(credit_card$V1)]=mean(credit_card$V1,na.rm=TRUE)
credit_card$V2[is.na(credit_card$V2)]=mean(credit_card$V2,na.rm=TRUE)
credit_card$V3[is.na(credit_card$V3)]=mean(credit_card$V3,na.rm=TRUE)
credit_card$V4[is.na(credit_card$V4)]=mean(credit_card$V4,na.rm=TRUE)
credit_card$V5[is.na(credit_card$V5)]=mean(credit_card$V5,na.rm=TRUE)
credit_card$V6[is.na(credit_card$V6)]=mean(credit_card$V6,na.rm=TRUE)
credit_card$V7[is.na(credit_card$V7)]=mean(credit_card$V7,na.rm=TRUE)
credit_card$V8[is.na(credit_card$V8)]=mean(credit_card$V8,na.rm=TRUE)
credit_card$V9[is.na(credit_card$V9)]=mean(credit_card$V9,na.rm=TRUE)
credit_card$V10[is.na(credit_card$V10)]=mean(credit_card$V10,na.rm=TRUE)
credit_card$V11[is.na(credit_card$V11)]=mean(credit_card$V11,na.rm=TRUE)
credit_card$V12[is.na(credit_card$V12)]=mean(credit_card$V12,na.rm=TRUE)
```

```
credit_card$V13[is.na(credit_card$V13)]=mean(credit_card$V13,na.rm=TRUE)
credit_card$V14[is.na(credit_card$V14)]=mean(credit_card$V14,na.rm=TRUE)
credit_card$V15[is.na(credit_card$V15)]=mean(credit_card$V15,na.rm=TRUE)
credit_card$V16[is.na(credit_card$V16)]=mean(credit_card$V16,na.rm=TRUE)
credit_card$V17[is.na(credit_card$V17)]=mean(credit_card$V17,na.rm=TRUE)
credit_card$V18[is.na(credit_card$V18)]=mean(credit_card$V18,na.rm=TRUE)
credit_card$V19[is.na(credit_card$V19)]=mean(credit_card$V19,na.rm=TRUE)
credit_card$V20[is.na(credit_card$V20)]=mean(credit_card$V20,na.rm=TRUE)
credit_card$V21[is.na(credit_card$V21)]=mean(credit_card$V21,na.rm=TRUE)
credit_card$V22[is.na(credit_card$V22)]=mean(credit_card$V22,na.rm=TRUE)
credit_card$V23[is.na(credit_card$V23)]=mean(credit_card$V23,na.rm=TRUE)
credit_card$V24[is.na(credit_card$V24)]=mean(credit_card$V24,na.rm=TRUE)
credit_card$V25[is.na(credit_card$V25)]=mean(credit_card$V25,na.rm=TRUE)
credit_card$V26[is.na(credit_card$V26)]=mean(credit_card$V26,na.rm=TRUE)
credit_card$V27[is.na(credit_card$V27)]=mean(credit_card$V27,na.rm=TRUE)
credit_card$V28[is.na(credit_card$V28)]=mean(credit_card$V28,na.rm=TRUE)
credit_card$Amount[is.na(credit_card$Amount)]=mean(credit_card$Amount,na.rm=TRUE)
credit_card$Class[is.na(credit_card$Class)]=mean(credit_card$Class,na.rm=TRUE)

options(scipen = 5)
credit_card
any(is.na(credit_card))
#From the result it is understood that there are NA values in the dataset
#So the data is now cleaned
```

5.2 Exploratory Analysis

```
> #Class of data object
> class(credit_card)
[1] "data.frame"
> #Display Internal structure of data
> str(credit_card)
'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 NA -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  -0.338 0.06 -0.503 NA -0.407 ...
 $ 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 NA ...
 $ V12   : num  -0.6178 NA 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 NA ...
 $ 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 NA -0.0382 ...
 $ V19   : num  0.404 -0.146 -2.262 -1.233 NA ...
 $ 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  0.278 -0.639 0.772 NA 0.798 ...
 $ 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 NA 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 ...
> #Summary of data
> 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.: -0.59856   1st Qu.: -0.8904  
 Median : 84692   Median :  0.01811   Median :  0.06548   Median :  0.1799  
 Mean   : 94814   Mean   :  0.00000   Mean   : -0.00001   Mean   :  0.0000  
 3rd Qu.:139321   3rd Qu.:  1.31565   3rd Qu.:  0.80372   3rd Qu.:  1.0272  
 Max.   :172792   Max.   :  2.45493   Max.   : 22.05773   Max.   :  9.3826  
                  NA's   :2           NA's   :3           NA's   :2        
       V4                  V5                  V6                 V7          
 Min.   :-5.683171   Min.   :-113.74331   Min.   :-26.1605   Min.   :-43.5572  
 1st Qu.:-0.848642   1st Qu.:  -0.69159   1st Qu.: -0.7683   1st Qu.: -0.5541  
 Median :-0.019845   Median :  -0.05433   Median : -0.2742   Median :  0.0401  
 Mean   : 0.000001   Mean   :   0.00001   Mean   :  0.0000   Mean   :  0.0000  
 3rd Qu.: 0.743348   3rd Qu.:   0.61193   3rd Qu.:  0.3986   3rd Qu.:  0.5704  
 Max.   :16.875344   Max.   :  34.80167   Max.   : 73.3016   Max.   :120.5895  
 NA's   :1           NA's   :3                                NA's   :1        
```

```
       V8                  V9                  V10                 V11           
 Min.   :-73.21672   Min.   :-13.434066   Min.   :-24.588262   Min.   :-4.797473  
 1st Qu.: -0.20863   1st Qu.: -0.643095   1st Qu.: -0.535426   1st Qu.:-0.762467  
 Median :  0.02236   Median : -0.051428   Median : -0.092921   Median :-0.032757  
 Mean   :  0.00000   Mean   :  0.000003   Mean   : -0.000005   Mean   : 0.000008  
 3rd Qu.:  0.32735   3rd Qu.:  0.597140   3rd Qu.:  0.453898   3rd Qu.: 0.739595  
 Max.   : 20.00721   Max.   : 15.594995   Max.   : 23.745136   Max.   :12.018913  
 NA's   :1           NA's   :1            NA's   :1            NA's   :5          
      V12                 V13                 V14                V15           
 Min.   :-18.683715   Min.   :-5.791881   Min.   :-19.2143   Min.   :-4.498945  
 1st Qu.: -0.405569   1st Qu.:-0.648535   1st Qu.: -0.4256   1st Qu.:-0.582888  
 Median :  0.140029   Median :-0.013568   Median :  0.0506   Median : 0.048064  
 Mean   : -0.000003   Mean   : 0.000002   Mean   :  0.0000   Mean   :-0.000005  
 3rd Qu.:  0.618237   3rd Qu.: 0.662507   3rd Qu.:  0.4931   3rd Qu.: 0.648821  
 Max.   :  7.848392   Max.   : 7.126883   Max.   : 10.5268   Max.   : 8.877742  
 NA's   :3            NA's   :2           NA's   :3          NA's   :4          
      V16                 V17                  V18                 V19           
 Min.   :-14.129855   Min.   :-25.162799   Min.   :-9.498746   Min.   :-7.213527  
 1st Qu.: -0.468037   1st Qu.: -0.483744   1st Qu.:-0.498850   1st Qu.:-0.456295  
 Median :  0.066432   Median : -0.065670   Median :-0.003644   Median : 0.003737  
 Mean   :  0.000008   Mean   :  0.000005   Mean   :-0.000011   Mean   : 0.000001  
 3rd Qu.:  0.523305   3rd Qu.:  0.399677   3rd Qu.: 0.500798   3rd Qu.: 0.458949  
 Max.   : 17.315112   Max.   :  9.253526   Max.   : 5.041069   Max.   : 5.591971  
 NA's   :4            NA's   :2            NA's   :3           NA's   :3          
      V20                 V21                 V22                  V23           
 Min.   :-54.49772   Min.   :-34.83038   Min.   :-10.933144   Min.   :-44.80774  
 1st Qu.: -0.21172   1st Qu.: -0.22840   1st Qu.: -0.542352   1st Qu.: -0.16185  
 Median : -0.06248   Median : -0.02945   Median :  0.006791   Median : -0.01119  
 Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.000001   Mean   :  0.00000  
 3rd Qu.:  0.13304   3rd Qu.:  0.18638   3rd Qu.:  0.528555   3rd Qu.:  0.14764  
```

```
      V24                 V25                 V26                 V27           
 Min.   :-2.836627   Min.   :-10.29540   Min.   :-2.604551   Min.   :-22.565679  
 1st Qu.:-0.354590   1st Qu.: -0.31714   1st Qu.:-0.326981   1st Qu.: -0.070839  
 Median : 0.040974   Median :  0.01659   Median :-0.052142   Median :  0.001342  
 Mean   :-0.000003   Mean   :  0.00000   Mean   : 0.000001   Mean   : -0.000001  
 3rd Qu.: 0.439525   3rd Qu.:  0.35072   3rd Qu.: 0.240958   3rd Qu.:  0.091044  
 Max.   : 4.584549   Max.   :  7.51959   Max.   : 3.517346   Max.   : 31.612198  
 NA's   :1           NA's   :2           NA's   :3           NA's   :3           
      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  
 NA's   :4                                                 
> #Column names
> names(credit_card)
 [1] "Time"   "V1"     "V2"     "V3"     "V4"     "V5"     "V6"     "V7"     "V8"    
[10] "V9"     "V10"    "V11"    "V12"    "V13"    "V14"    "V15"    "V16"    "V17"   
[19] "V18"    "V19"    "V20"    "V21"    "V22"    "V23"    "V24"    "V25"    "V26"   
[28] "V27"    "V28"    "Amount" "Class" 
> #Dimensions of data
> dim(credit_card)
[1] 284807     31
```

```
> # Data of the top
> head(credit_card)
  Time         V1          V2        V3         V4          V5          V6          V7
1    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778  0.23959855
2    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081 -0.07880298
3    1         NA -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938  0.79146096
4    1 -0.9662717 -0.18522601 1.7929933 -0.8632913          NA  1.24720317  0.23760894
5    2 -1.1582331  0.87773676 1.5487178  0.4030339 -0.40719338  0.09592146  0.59294075
6    2 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755  0.47620095
          V8         V9         V10        V11         V12        V13        V14
1  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086 -0.9913898 -0.3111694
2  0.08510165 -0.2554251 -0.16697441  1.6127267          NA  0.4890950 -0.1437723
3  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369  0.7172927 -0.1659459
4  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823  0.5077569 -0.2879237
5 -0.27053268  0.8177393  0.75307443         NA  0.53819555  1.3458516 -1.1196698
6          NA -0.5686714 -0.37140720  1.3412620  0.35989384 -0.3580907 -0.1371337
         V15        V16         V17         V18         V19         V20          V21
1  1.4681770 -0.4704005  0.20797124  0.02579058  0.40399296  0.25141210 -0.018306778
2  0.6355581  0.4639170 -0.11480466 -0.18336127 -0.14578304 -0.06908314 -0.225775248
3  2.3458649 -2.8900832  1.10996938 -0.12135931 -2.26185709  0.52497973  0.247998153
4 -0.6314181 -1.0596472 -0.68409279          NA -1.23262197 -0.20803778 -0.108300452
5         NA -0.4514492 -0.23703324 -0.03819479          NA  0.40854236 -0.009430697
6  0.5176168  0.4017259 -0.05813282  0.06865315 -0.03319379  0.08496767 -0.208253515
         V22         V23         V24        V25        V26          V27         V28 Amount
1  0.2778376 -0.11047391  0.06692808  0.1285394 -0.1891148  0.133558377 -0.02105305 149.62
2 -0.6386720  0.10128802 -0.33984648  0.1671704  0.1258945 -0.008983099  0.01472417   2.69
3  0.7716794  0.90941226 -0.68928096 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66
4         NA -0.19032052 -1.17557533  0.6473760 -0.2219288  0.062722849          NA 123.50
5  0.7982785 -0.13745808  0.14126698 -0.2060096  0.5022922  0.219422230  0.21515315  69.99
6 -0.5598248 -0.02639767 -0.37142658 -0.2327938         NA  0.253844225  0.08108026   3.67

  Class
1     0
2     0
3     0
4     0
5     0
6     0
> #Data from the top
> tail(credit_card)
         Time          V1          V2         V3         V4          V5         V6
284802 172785   0.1203164  0.93100513 -0.5460121 -0.7450968  1.13031398 -0.2359732
284803 172786 -11.8811179 10.07178497 -9.8347835 -2.0666557 -5.36447278 -2.6068373
284804 172787  -0.7327887 -0.05508049  2.0350297 -0.7385886  0.86822940  1.0584153
284805 172788   1.9195650 -0.30125385 -3.2496398 -0.5578281  2.63051512  3.0312601
284806 172788  -0.2404400  0.53048251  0.7025102  0.6897992 -0.37796113  0.6237077
284807 172792  -0.5334125 -0.18973334  0.7033374 -0.5062712 -0.01254568 -0.6496167
               V7         V8         V9        V10        V11         V12        V13
284802  0.8127221  0.1150929 -0.2040635 -0.6574221  0.6448373  0.19091623 -0.5463289
284803 -4.9182154  7.3053340  1.9144283  4.3561704 -1.5931053  2.71194079 -0.6892556
284804  0.0243297  0.2948687  0.5848000 -0.9759261 -0.1501888  0.91580191  1.2147558
284805 -0.2968265  0.7084172  0.4324540 -0.4847818  0.4116137  0.06311886 -0.1836987
284806 -0.6861800  0.6791455  0.3920867 -0.3991257 -1.9338488 -0.96288614 -1.0420817
284807  1.5770063 -0.4146504  0.4861795 -0.9154266 -1.0404583 -0.03151305 -0.1880929
               V14         V15        V16         V17        V18        V19         V20
284802 -0.73170658 -0.80803553  0.5996281  0.07044075  0.3731103  0.1289038 0.000675833
284803  4.62694202 -0.92445871  1.1076406  1.99169111  0.5106323 -0.6829197 1.475829135
284804 -0.67514296  1.16493091 -0.7117573 -0.02569286 -1.2211789 -1.5455561 0.059615900
284805 -0.51060184  1.32928351  0.1407160  0.31350179  0.3956525 -0.5772518 0.001395970
284806  0.44962444  1.96256312 -0.6085771  0.50992846  1.1139806  2.8978488 0.127433516
284807 -0.08431647  0.04133345 -0.3026201 -0.66037665  0.1674299 -0.2561169 0.382948105
```

5.3 Cleaning of dataset

```
> #Loading the creditcardfraud csv file into a data frame
> credit_card=read.csv("creditcardfraud.csv")
> credit_card
   Time         V1          V2          V3          V4           V5          V6
1     0 -1.3598071 -0.07278117  2.53634674  1.37815522 -0.338320770  0.46238778
2     0  1.1918571  0.26615071  0.16648011  0.44815408  0.060017649 -0.08236081
3     1         NA -1.34016307  1.77320934  0.37977959 -0.503198133  1.80049938
4     1 -0.9662717 -0.18522601  1.79299334 -0.86329128           NA  1.24720317
5     2 -1.1582331  0.87773676  1.54871785  0.40303393 -0.407193377  0.09592146
6     2 -0.4259659  0.96052304  1.14110934 -0.16825208  0.420986881 -0.02972755
7     4  1.2296576  0.14100351  0.04537077  1.20261274  0.191880989  0.27270812
8     7 -0.6442694          NA  1.07438038 -0.49219902  0.948934095  0.42811846
9     7 -0.8942861  0.28615720 -0.11319221          NA  2.669598660  3.72181806
10    9 -0.3382618  1.11959338  1.04436655 -0.22218728  0.499360806 -0.24676110
11   10  1.4490438 -1.17633882  0.91385983 -1.37566666           NA -0.62915214
12   10         NA  0.61610946 -0.87429970 -0.09401863  2.924584378  3.31702717
13   10  1.2499987 -1.22163681  0.38393015 -1.23489869 -1.485419474 -0.75323016
14   11  1.0693736  0.28772213  0.82861273  2.71252043 -0.178398016  0.33754373
15   12 -2.7918548          NA  1.64175016  1.76747274 -0.136588446  0.80759647
16   12 -0.7524170  0.34548542  2.05732291 -1.46864330 -1.158393680 -0.07784983
17   12  1.1032154 -0.04029622  1.26733209  1.28909147 -0.735997164  0.28806916
18   13 -0.4369051  0.91896621          NA -0.72721905  0.915678718 -0.12786735
19   14 -5.4012577 -5.45014783  1.18630463  1.73623880  3.049105878 -1.76340557
20   15  1.4929360 -1.02934573  0.45479473 -1.43802588 -1.555434101 -0.72096115
21   16  0.6948848 -1.36181910  1.02922104  0.83415930 -1.191208794  1.30910882
22   17  0.9624961  0.32846103 -0.17147905  2.10920407  1.129565571  1.69603769
23   18  1.1666164          NA -0.06730031  2.26156924  0.428804195  0.08947352
24   18  0.2474911  0.27766563  1.18547084 -0.09260255           NA -0.15011600
```

```
> #Checking if there are any NA values in the dataset
> any(is.na(credit_card))
[1] TRUE
> # So from the output it is understood that there are NA values in the dataset
> #Let us extract the count of NA values in the dataset
> sum(is.na(credit_card))
[1] 64
> 
```

```
> #Replacing the NA values in each column with the mean value of the values in
> #the column
> credit_card$Time[is.na(credit_card$Time)]=mean(credit_card$Time,na.rm=TRUE)
> credit_card$V1[is.na(credit_card$V1)]=mean(credit_card$V1,na.rm=TRUE)
> credit_card$V2[is.na(credit_card$V2)]=mean(credit_card$V2,na.rm=TRUE)
> credit_card$V3[is.na(credit_card$V3)]=mean(credit_card$V3,na.rm=TRUE)
> credit_card$V4[is.na(credit_card$V4)]=mean(credit_card$V4,na.rm=TRUE)
> credit_card$V5[is.na(credit_card$V5)]=mean(credit_card$V5,na.rm=TRUE)
> credit_card$V6[is.na(credit_card$V6)]=mean(credit_card$V6,na.rm=TRUE)
> credit_card$V7[is.na(credit_card$V7)]=mean(credit_card$V7,na.rm=TRUE)
> credit_card$V8[is.na(credit_card$V8)]=mean(credit_card$V8,na.rm=TRUE)
> credit_card$V9[is.na(credit_card$V9)]=mean(credit_card$V9,na.rm=TRUE)
> credit_card$V10[is.na(credit_card$V10)]=mean(credit_card$V10,na.rm=TRUE)
> credit_card$V11[is.na(credit_card$V11)]=mean(credit_card$V11,na.rm=TRUE)
> credit_card$V12[is.na(credit_card$V12)]=mean(credit_card$V12,na.rm=TRUE)
> credit_card$V13[is.na(credit_card$V13)]=mean(credit_card$V13,na.rm=TRUE)
> credit_card$V14[is.na(credit_card$V14)]=mean(credit_card$V14,na.rm=TRUE)
> credit_card$V15[is.na(credit_card$V15)]=mean(credit_card$V15,na.rm=TRUE)
> credit_card$V16[is.na(credit_card$V16)]=mean(credit_card$V16,na.rm=TRUE)
> credit_card$V17[is.na(credit_card$V17)]=mean(credit_card$V17,na.rm=TRUE)
> credit_card$V18[is.na(credit_card$V18)]=mean(credit_card$V18,na.rm=TRUE)
> credit_card$V19[is.na(credit_card$V19)]=mean(credit_card$V19,na.rm=TRUE)
> credit_card$V20[is.na(credit_card$V20)]=mean(credit_card$V20,na.rm=TRUE)
> credit_card$V21[is.na(credit_card$V21)]=mean(credit_card$V21,na.rm=TRUE)
> credit_card$V22[is.na(credit_card$V22)]=mean(credit_card$V22,na.rm=TRUE)
> credit_card$V23[is.na(credit_card$V23)]=mean(credit_card$V23,na.rm=TRUE)
> credit_card$V24[is.na(credit_card$V24)]=mean(credit_card$V24,na.rm=TRUE)
> credit_card$V25[is.na(credit_card$V25)]=mean(credit_card$V25,na.rm=TRUE)
> credit_card$V26[is.na(credit_card$V26)]=mean(credit_card$V26,na.rm=TRUE)
> credit_card$V27[is.na(credit_card$V27)]=mean(credit_card$V27,na.rm=TRUE)
```

```
> credit_card$V28[is.na(credit_card$V28)]=mean(credit_card$V28,na.rm=TRUE)
> credit_card$Amount[is.na(credit_card$Amount)]=mean(credit_card$Amount,na.rm=TRUE)
> credit_card$Class[is.na(credit_card$Class)]=mean(credit_card$Class,na.rm=TRUE)
> options(scipen = 5)
> credit_card
   Time             V1              V2              V3              V4
1     0 -1.359807134000 -0.072781173000  2.536346738000  1.3781552240000
2     0  1.191857111000  0.266150712000  0.166480113000  0.4481540780000
3     1  0.000003417694 -1.340163075000  1.773209343000  0.3797795930000
4     1 -0.966271712000 -0.185226008000  1.792993340000 -0.8632912750000
5     2 -1.158233093000  0.877736755000  1.548717847000  0.4030339340000
6     2 -0.425965884000  0.960523045000  1.141109342000 -0.1682520800000
7     4  1.229657635000  0.141003507000  0.045370774000  1.2026127370000
8     7 -0.644269442000 -0.000005590909  1.074380376000 -0.4921990180000
9     7 -0.894286082000  0.286157196000 -0.113192213000  0.0000009533731
10    9 -0.338261752000  1.119593376000  1.044366552000 -0.2221872770000
11   10  1.449043781000 -1.176338825000  0.913859833000 -1.3756666550000
12   10  0.000003417694  0.616109459000 -0.874299703000 -0.0940186260000
13   10  1.249998742000 -1.221636809000  0.383930151000 -1.2348986880000
14   11  1.069373588000  0.287722129000  0.828612727000  2.7125204300000
15   12 -2.791854766000 -0.000005590909  1.641750161000  1.7674727440000
16   12 -0.752417043000  0.345485415000  2.057322913000 -1.4686432980000
17   12  1.103215435000 -0.040296215000  1.267332089000  1.2890914700000
18   13 -0.436905071000  0.918966213000 -0.000001822373 -0.7272190540000
19   14 -5.401257663000 -5.450147834000  1.186304631000  1.7362388000000
20   15  1.492935977000 -1.029345732000  0.454794734000 -1.4380258800000
21   16  0.694884776000 -1.361819103000  1.029221040000  0.8341592990000
22   17  0.962496070000  0.328461026000 -0.171479054000  2.1092040680000
23   18  1.166616382000 -0.000005590909 -0.067300314000  2.2615692390000
24   18  0.247491128000  0.277665627000  1.185470842000 -0.0926025500000
```

```
7    0.0051677690000   4.99      0
8   -1.0853391880000  40.80      0
9    0.1424043300000  93.20      0
10   0.0830756490000   3.68      0
11   0.0162532620000   7.80      0
12  -0.0543373880000   9.99      0
13   0.0424220890000 121.50      0
14   0.0212933110000  27.50      0
15  -0.0301536370000  58.80      0
16   0.1293940590000  15.99      0
17  -0.0000004069958  12.99      0
18   0.1310237890000   0.89      0
19   0.9495942460000  46.80      0
20   0.0076022560000   5.00      0
21   0.0634986490000 231.71      0
22  -0.0146053280000  34.09      0
23  -0.0000004069958   2.28      0
24   0.2504753520000  22.75      0
25   0.0145997520000   0.89      0
26   0.2432316720000  26.43      0
27   0.0300411910000  41.88      0
28  -0.0000004069958  16.00      0
29   0.1526646450000  33.00      0
30   0.0242203490000  12.99      0
31   0.0118362310000  17.28      0
32   0.0044526310000   4.45      0
 [ reached 'max' / getOption("max.print") -- omitted 284775 rows ]
```

```
> any(is.na(credit_card))
[1] FALSE
```

5.4 Data Modelling

After we have cleaning our entire dataset, we will split our dataset into training set as well as test set with a split ratio of 0.80. This means that 80% of our data will be attributed to the train data whereas 20% will be attributed to the test data.

```
> #data modelling
> library(caTools)
Warning message:
package 'caTools' was built under R version 4.1.2
> set.seed(100)
> das=sample.split(data$Class,SplitRatio=0.80)
> train_data = subset(data,das==TRUE)
> test_data = subset(data,das==FALSE)
> dim(train_data)
[1] 227846     32
> dim(test_data)
[1] 56961    32
>
```

5.5 Logistic Regression

We will first fit the model by using the glm() function. A logistic regression is used for modeling the outcome probability of a class such as pass/fail, positive/negative and in our case – fraud/not fraud.

```
Console   Terminal   Jobs
R 4.1.0 · C:/Users/sruth/OneDrive/Desktop/CSE4027 LAB/
> #Get the workspace
> getwd()
[1] "C:/Users/sruth/OneDrive/Desktop"
> setwd("C:/Users/sruth/OneDrive/Desktop/CSE4027 LAB")
> dir()
 [1] "BirthsKingCounty2001.txt"          "bmi_data (1).csv"
 [3] "bmi_data.csv"                      "colon.txt"
 [5] "COVID_country_wise_latest.csv"     "credit_cards.csv"
 [7] "CSE4027 LAB.Rproj"                 "CSV"
 [9] "customer_segmentation_cleaned.csv" "diabetes.csv"
[11] "Diabetes_Updated.csv"              "Excel datasets"
[13] "House.xlsx"                        "Iris (2).csv"
[15] "Json"                              "lab sheeet 7_19BCD7040.docx"
[17] "Mall_Customers.csv"                "MRI.txt"
[19] "output.xlsx"                       "SalaryData.txt"
[21] "Shr.xlsx"                          "STATA"
[23] "Student_Data_cleaned.csv"          "Student_Data_Uncleaned.csv"
[25] "StudentsPerformance.csv"           "txt"
[27] "weatherHistory.csv"
> #reading the cleaned datset
> data=read.csv("credit_cards.csv")
> data
    X Time            V1            V2            V3            V4            V5          V6
1   1    0 -1.359807e+00 -7.278117e-02  2.536347e+00  1.378155e+00 -3.383208e-01  0.46238778
2   2    0  1.191857e+00  2.661507e-01  1.664801e-01  4.481541e-01  6.001765e-02 -0.08236081
3   3    1  3.417694e-06 -1.340163e+00  1.773209e+00  3.797796e-01 -5.031981e-01  1.80049938
4   4    1 -9.662717e-01 -1.852260e-01  1.792993e+00 -8.632913e-01  1.157317e-05  1.24720317
5   5    2 -1.158233e+00  8.777368e-01  1.548718e+00  4.030339e-01 -4.071934e-01  0.09592146
6   6    2 -4.259659e-01  9.605230e-01  1.141109e+00 -1.682521e-01  4.209869e-01 -0.02972755
7   7    4  1.229658e+00  1.410035e-01  4.537077e-02  1.202613e+00  1.918810e-01  0.27270812
8   8    7 -6.442694e-01 -5.590909e-06  1.074380e+00 -4.921990e-01  9.489341e-01  0.42811846
9   9    7 -8.942861e-01  2.861572e-01 -1.131922e-01  9.533730e-07  2.669599e+00  3.72181806
10 10    9 -3.382618e-01  1.119593e+00  1.044367e+00 -2.221873e-01  4.993608e-01 -0.24676110
11 11   10  1.449044e+00 -1.176339e+00  9.138598e-01 -1.375667e+00  1.157317e-05 -0.62915214
12 12   10  3.417694e-06  6.161095e-01 -8.742997e-01 -9.401863e-02  2.924584e+00  3.31702717
13 13   10  1.249999e+00 -1.221637e+00  3.839302e-01 -1.234899e+00 -1.485419e+00 -0.75323016
14 14   11  1.069374e+00  2.877221e-01  8.286127e-01  2.712520e+00 -1.783980e-01  0.33754373
```

```
> #data modelling
> library(caTools)
Warning message:
package 'caTools' was built under R version 4.1.2
> set.seed(100)
> das=sample.split(data$Class,SplitRatio=0.80)
> train_data = subset(data,das==TRUE)
> test_data = subset(data,das==FALSE)
> dim(train_data)
[1] 227846     32
> dim(test_data)
[1] 56961    32
>
```

```
> #Fitting The Logistic Regression Model
> Logistic_Model=glm(Class~.,test_data,family=binomial())
> summary(Logistic_Model)

Call:
glm(formula = Class ~ ., family = binomial(), data = test_data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.4800  -0.0297  -0.0184  -0.0101   4.4517

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -7.585e+00  1.735e+00  -4.372 1.23e-05 ***
X            3.906e-05  1.750e-05   2.231 0.025649 *
Time        -7.710e-05  3.048e-05  -2.530 0.011411 *
V1           2.021e-02  1.790e-01   0.113 0.910070
V2           3.048e-01  7.096e-01   0.430 0.667506
V3          -2.253e-01  1.253e-01  -1.798 0.072143 .
V4           1.062e+00  9.693e-01   1.096 0.273087
V5           4.313e-01  7.123e-01   0.605 0.544861
V6          -2.001e-01  1.875e-01  -1.067 0.285884
V7           2.530e-01  5.966e-01   0.424 0.671499
V8          -3.138e-01  7.127e-02  -4.404 1.06e-05 ***
V9           4.887e-01  1.504e+00   0.325 0.745213
V10         -1.116e+00  1.159e+00  -0.963 0.335359
V11         -3.130e-01  3.656e-01  -0.856 0.391998
V12          4.119e-02  4.511e-01   0.091 0.927251
V13         -2.616e-01  4.498e-01  -0.582 0.560780
V14         -4.673e-01  2.525e-01  -1.851 0.064157 .
V15          4.978e-02  2.867e-01   0.174 0.862177
V16          6.848e-01  2.141e+00   0.320 0.749069
V17         -1.874e-01  3.922e-01  -0.478 0.632758
V18         -7.366e-01  1.917e+00  -0.384 0.700797
V19          6.013e-01  1.096e+00   0.549 0.583275
V20         -5.114e-02  5.900e-01  -0.087 0.930929
```

```
V9          4.887e-01  1.504e+00   0.325 0.745213
V10        -1.116e+00  1.159e+00  -0.963 0.335359
V11        -3.130e-01  3.656e-01  -0.856 0.391998
V12         4.119e-02  4.511e-01   0.091 0.927251
V13        -2.616e-01  4.498e-01  -0.582 0.560780
V14        -4.673e-01  2.525e-01  -1.851 0.064157 .
V15         4.978e-02  2.867e-01   0.174 0.862177
V16         6.848e-01  2.141e+00   0.320 0.749069
V17        -1.874e-01  3.922e-01  -0.478 0.632758
V18        -7.366e-01  1.917e+00  -0.384 0.700797
V19         6.013e-01  1.096e+00   0.549 0.583275
V20        -5.114e-02  5.900e-01  -0.087 0.930929
V21         4.971e-01  5.220e-01   0.952 0.340901
V22         8.267e-01  8.596e-01   0.962 0.336198
V23        -2.140e-01  1.691e-01  -1.265 0.205762
V24        -1.666e-01  3.095e-01  -0.538 0.590487
V25         8.020e-02  3.015e-01   0.266 0.790228
V26         3.079e-01  3.678e-01   0.837 0.402510
V27        -8.526e-01  2.493e-01  -3.420 0.000626 ***
V28        -3.873e-01  1.555e-01  -2.491 0.012751 *
Amount      3.992e-04  8.861e-04   0.451 0.652308
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1443.40  on 56960  degrees of freedom
Residual deviance:  490.75  on 56929  degrees of freedom
AIC: 554.75

Number of Fisher Scoring iterations: 16
```

Predicting The Accuracy of the Model by building the confusion matrix.

```
Number of Fisher Scoring iterations: 16

> #predicting the accuracy
> res<-predict(Logistic_Model,test_data,type = "response")
> res
           7           12           28           32           34           36           39
3.066158e-03 7.528463e-04 7.529045e-06 9.438246e-04 7.484798e-04 9.116453e-04 9.934911e-04
          42           44           48           63           74           77           79
4.247785e-04 2.778978e-04 2.704975e-03 3.849611e-03 6.319473e-04 8.987106e-04 6.382294e-04
          83           84           91           95          107          111          112
3.481370e-06 1.188128e-06 2.837971e-04 6.753974e-04 4.178518e-03 2.506393e-03 1.398394e-03
         115          117          120          127          128          132          144
1.581715e-03 1.573216e-02 1.299391e-03 5.109543e-04 2.485867e-03 8.566951e-04 3.389153e-04
         147          152          156          157          161          170          174
5.814577e-08 1.440541e-03 4.413185e-04 1.784535e-04 2.622557e-06 1.041129e-03 2.697894e-03
         177          183          190          191          194          202          203
3.572954e-03 1.391615e-03 1.289244e-03 6.805024e-04 2.508976e-04 7.101873e-04 4.058908e-03
         204          206          207          217          224          233          239
1.241804e-03 4.297158e-04 1.705015e-03 4.774572e-04 2.675200e-03 1.408637e-04 1.185961e-05
         252          255          256          262          266          269          270
```

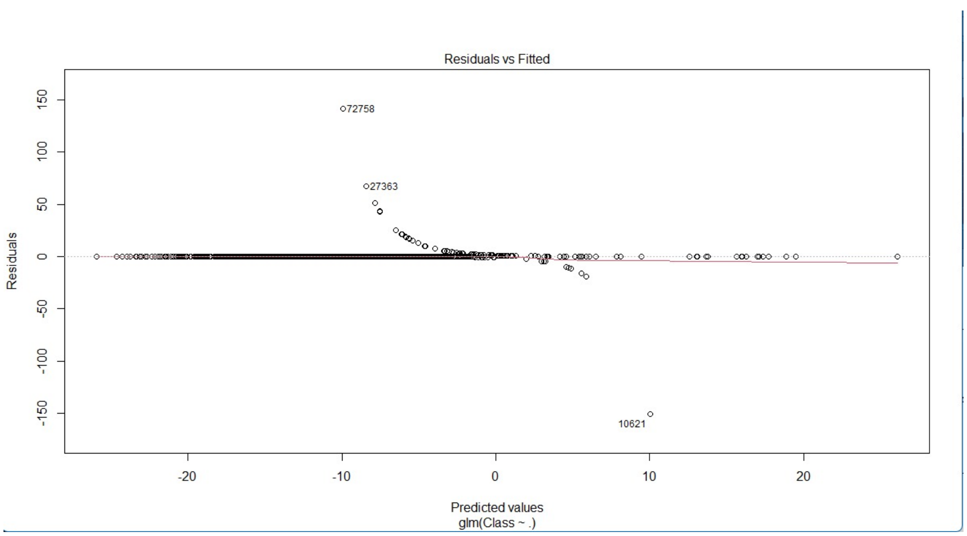
```
        4922         4931         4940         4954         4955         4959
4.739170e-05 1.364489e-04 6.252649e-05 8.251308e-05 3.019718e-04 2.121739e-04
 [ reached getOption("max.print") -- omitted 55961 entries ]
> restr<-predict(Logistic_Model,train_data,type = "response")
> restr
           1            2            3            4            5            6            8
1.906884e-03 4.627910e-04 4.768969e-06 1.630890e-05 7.687218e-04 3.139000e-04 7.103767e-03
           9           10           11           13           14           15           16
9.101237e-04 9.814237e-04 1.554715e-06 6.544175e-06 5.146173e-03 4.626025e-03 5.806834e-04
          17           18           19           20           21           22           23
2.277785e-03 6.511351e-04 1.742250e-03 7.578350e-07 2.612297e-06 1.504905e-03 1.430025e-03
          24           25           26           27           29           30           31
6.383821e-03 4.734834e-05 6.992614e-04 2.071782e-03 6.660891e-03 6.732000e-04 2.607378e-03
          33           35           37           38           40           41           43
7.484506e-04 7.496244e-04 2.630619e-03 5.982678e-04 8.961274e-04 5.877931e-04 2.667806e-03
          45           46           47           49           50           51           52
3.211085e-04 1.591172e-03 6.453823e-04 9.176758e-06 1.911019e-03 2.552337e-03 6.085619e-03
          53           54           55           56           57           58           59
1.314855e-03 1.810249e-03 1.804388e-03 4.130193e-03 1.795995e-03 3.531467e-04 1.028365e-03
          60           61           62           64           65           66           67
3.760466e-04 1.241695e-03 7.221381e-06 5.324375e-04 2.050350e-03 5.755374e-04 7.594554e-03
          68           69           70           71           72           73           75
```

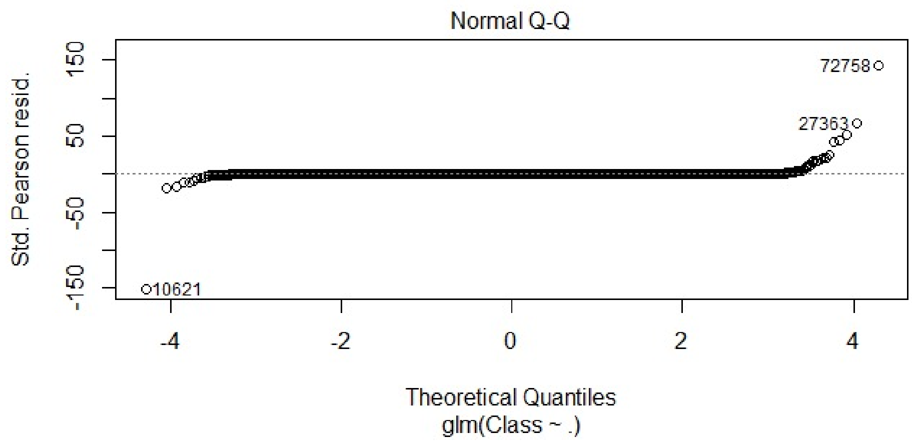
```
> #Building the confusion matrix for training data
> cm<-table(Actual_Value=train_data$Class,Predicted_Value=restr>0.5)
> cm
            Predicted_Value
Actual_Value  FALSE   TRUE
           0 227406     46
           1    151    243
> #Accuracy
> accuracy=(cm[[1,1]]+cm[[2,2]])/sum(cm)
> accuracy
[1] 0.9991354
```

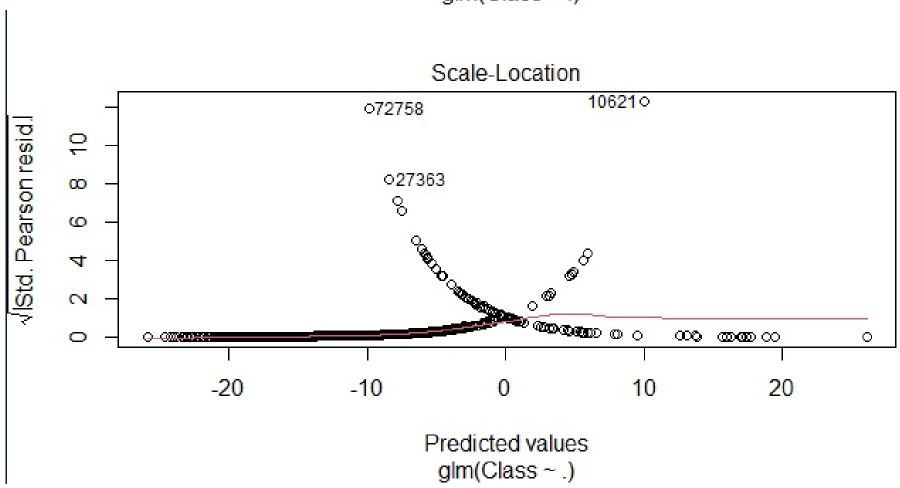
```
> #Building  the confusion matrix for testing Data
> cmte<-table(Actual_Value=test_data$Class,Predicted_Value=res>0.5)
> cmte
            Predicted_Value
Actual_Value FALSE  TRUE
           0 56853    10
           1    43    55
> #accuracy
> accuracy=(cmte[[1,1]]+cmte[[2,2]])/sum(cmte)
> accuracy
[1] 0.9990695
>
```

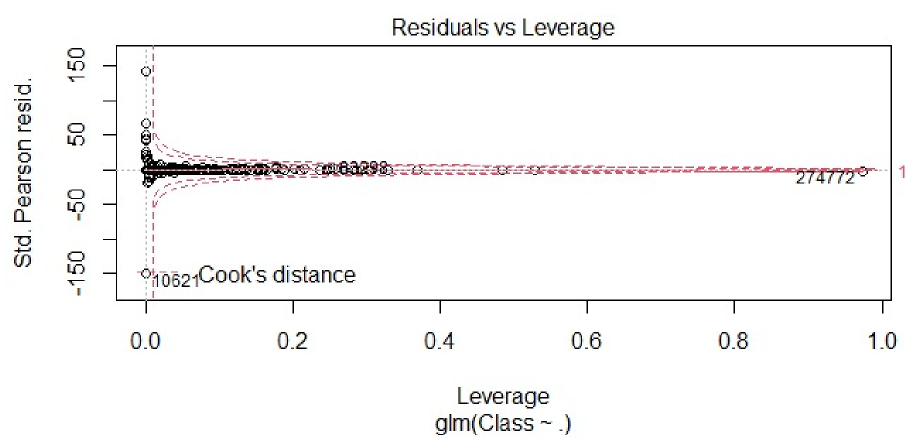
```
           1    43    55
> #accuracy
> accuracy=(cmte[[1,1]]+cmte[[2,2]])/sum(cmte)
> accuracy
[1] 0.9990695
> plot(Logistic_Model)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
```

After we have summarized our model, we will visual it through the following plots





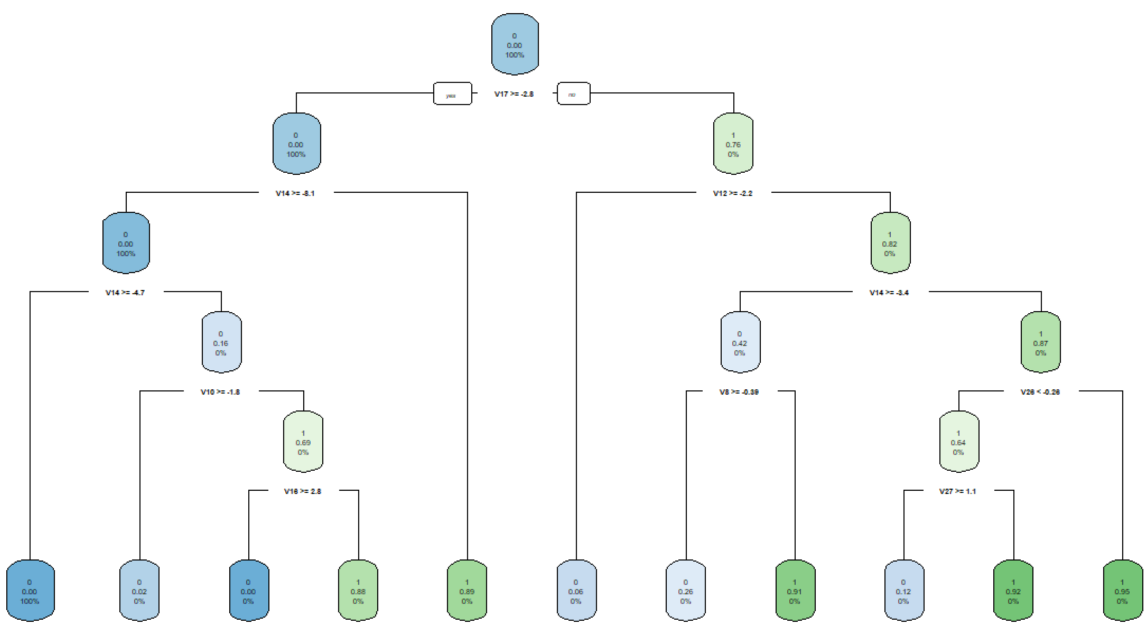




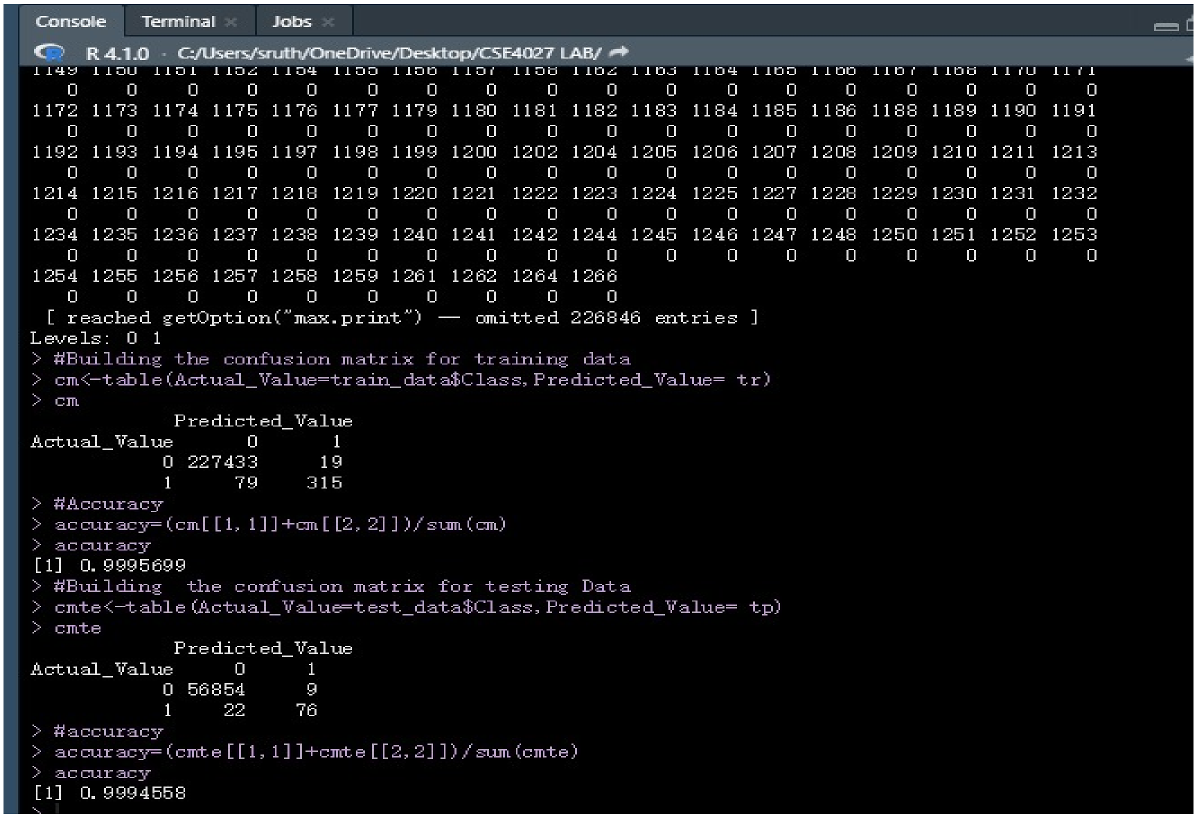
5.6 Decision Tree

We will now implement our decision tree model and will plot it using the rpart.plot() function. We will specifically use the recursive parting to plot the decision tree.

```
R 4.1.0 · C:/Users/sruth/OneDrive/Desktop/CSE4027 LAB/
> #data modelling
> library(caTools)
> set.seed(100)
> das=sample.split(data$Class,SplitRatio=0.80)
> train_data = subset(data,das==TRUE)
> test_data = subset(data,das==FALSE)
> dim(train_data)
[1] 227846     32
> dim(test_data)
[1] 56961     32
> #decision tree
> library(rpart)
Warning message:
package 'rpart' was built under R version 4.1.2
> library(rpart.plot)
Warning message:
package 'rpart.plot' was built under R version 4.1.2
> decisiontree_model <- rpart(Class ~ . , data, method = 'class')
> predicted_val <- predict(decisiontree_model, data, type = 'class')
> probability <- predict(decisiontree_model, data, type = 'prob')
> rpart.plot(decisiontree_model)
>
```



Predicting the model by building the confusion matrix

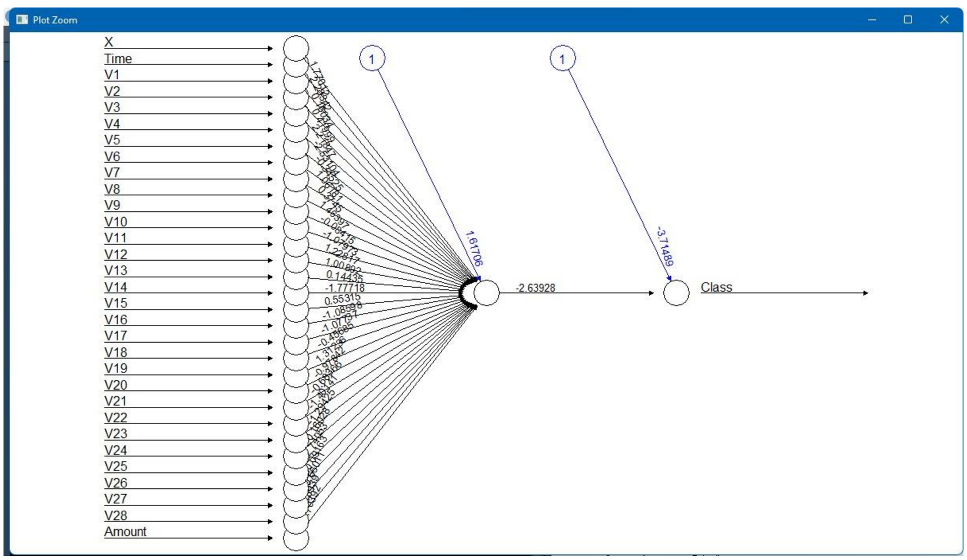


```
1149 1150 1151 1152 1154 1155 1156 1157 1158 1162 1163 1164 1165 1166 1167 1168 1170 1171
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
1172 1173 1174 1175 1176 1177 1179 1180 1181 1182 1183 1184 1185 1186 1188 1189 1190 1191
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
1192 1193 1194 1195 1197 1198 1199 1200 1202 1204 1205 1206 1207 1208 1209 1210 1211 1213
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1227 1228 1229 1230 1231 1232
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
1234 1235 1236 1237 1238 1239 1240 1241 1242 1244 1245 1246 1247 1248 1250 1251 1252 1253
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
1254 1255 1256 1257 1258 1259 1261 1262 1264 1266
   0    0    0    0    0    0    0    0    0    0
 [ reached getOption("max.print") -- omitted 226846 entries ]
Levels: 0 1
> #Building the confusion matrix for training data
> cm<-table(Actual_Value=train_data$Class,Predicted_Value= tr)
> cm
            Predicted_Value
Actual_Value      0      1
           0 227433     19
           1     79    315
> #Accuracy
> accuracy=(cm[[1,1]]+cm[[2,2]])/sum(cm)
> accuracy
[1] 0.9995699
> #Building  the confusion matrix for testing Data
> cmte<-table(Actual_Value=test_data$Class,Predicted_Value= tp)
> cmte
            Predicted_Value
Actual_Value     0     1
           0 56854     9
           1    22    76
> #accuracy
> accuracy=(cmte[[1,1]]+cmte[[2,2]])/sum(cmte)
> accuracy
[1] 0.9994558
```

5.6 Artificial Neural Networks

We import the neural net package that would allow us to implement our ANNs. Then we proceeded to plot it using the plot() function. Now, in the case of Artificial Neural Networks, there is a range of values that is between 1 and 0. We set a threshold as 0.5, that is, values above 0.5 will correspond to 1 and the rest will be 0.

```
R 4.1.0 · C:/Users/sruth/OneDrive/Desktop/CSE4027 LAB/
29 -1.838913e-01 -2.774640e-01  1.826875e-01  1.526646e-01  33.00     0
30  4.235470e-07 -3.950695e-01  8.146112e-02  2.422035e-02  12.99     0
31  5.091357e-01  2.888578e-01 -2.270498e-02  1.183623e-02  17.28     0
 [ reached 'max' / getOption("max.print") -- omitted 284776 rows ]
> #data modelling
> library(caTools)
Warning message:
package 'caTools' was built under R version 4.1.2
> set.seed(100)
> das=sample.split(data$Class,SplitRatio=0.80)
> train_data = subset(data,das==TRUE)
> test_data = subset(data,das==FALSE)
> dim(train_data)
[1] 227846     32
> dim(test_data)
[1] 56961    32
> library(neuralnet)
Warning message:
package 'neuralnet' was built under R version 4.1.2
> ANN =neuralnet(Class~.,train_data,linear.output=FALSE)
> plot(ANN)
```



Predicting the model by building the confusion matrix

```
4959    0.001736456
 [ reached getOption("max.print") — omitted 55961 rows ]
> predANN=compute(ANN,test_data)
> resultANN=predANN$net.result
> resultANN=ifelse(resultANN>0.5,1,0)
> Antp<-predict(ANN ,test_data,type = "class")
> Antp
```

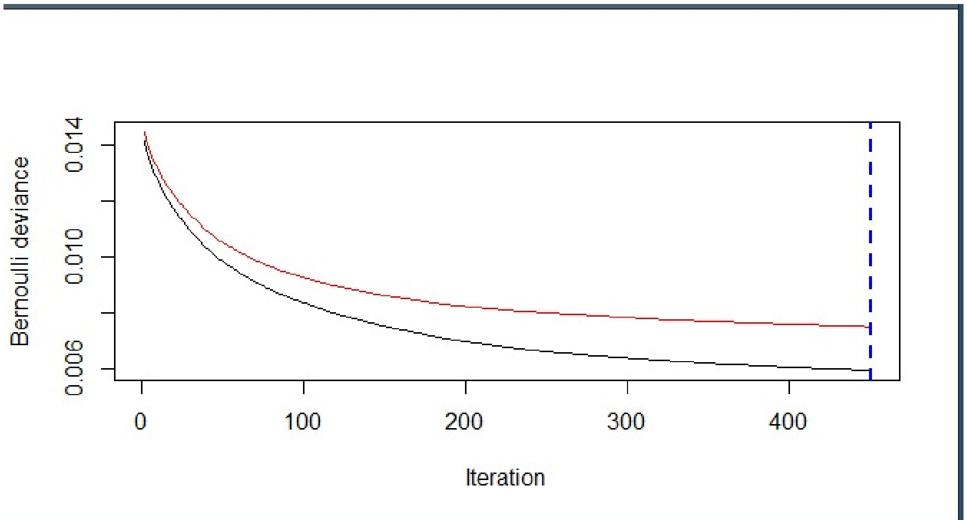
```
 [ reached getOption("max.print") -- omitted 55961 rows ]
> cm<-table(Actual_Value=test_data$Class,Predicted_Value= Antp)
> cm
            Predicted_Value
Actual_Value 0.00173645637912099 0.00173645637913137
           0               56862                   1
           1                  98                   0
> #Accuracy
> accuracy=(cm[[1,1]]+cm[[2,2]])/sum(cm)
> accuracy
[1] 0.998262
```

```
> #Building  the confusion matrix for training Data
> Antr<-predict(ANN ,train_data,type = "class")
> cmte<-table(Actual_Value=train_data$Class,Predicted_Value= Antr)
> cmte
            Predicted_Value
Actual_Value 0.00173645637912099 0.00173645637917017 0.00173660344378561
           0              227450                   1                   1
           1                 394                   0                   0
> #accuracy
> accuracy=(cmte[[1,1]]+cmte[[2,2]])/sum(cmte)
> #accuracy
> accuracy=(cmte[[1,1]]+cmte[[2,2]])/sum(cmte)
> accuracy
[1] 0.998262
>
```

5.7 Gradient Boosting Classifier

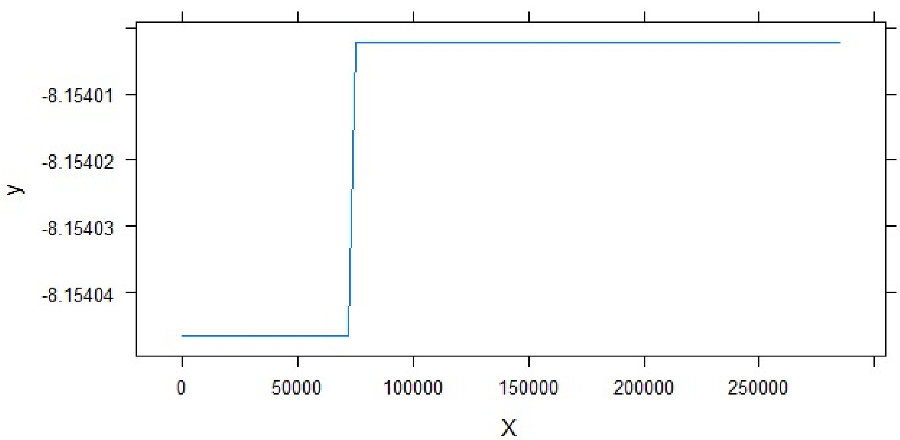
This model comprises of several underlying ensemble models like weak decision trees. These decision trees combine together to form a strong model of gradient boosting.

```
> # Get the time to train the GBM model
> system.time(
+   model_gbm <- gbm(Class ~ .
+                    , distribution = "bernoulli"
+                    , data = rbind(train_data, test_data)
+                    , n.trees = 450
+                    , interaction.depth = 3
+                    , n.minobsinnode = 100
+                    , shrinkage = 0.01
+                    , bag.fraction = 0.5
+                    , train.fraction = nrow(train_data) / (nrow(train_data) + nrow(test_data))
+   )
+ )
   user  system elapsed
 183.89    0.06  188.14
> # Determine best iteration based on test data
> gbm.iter = gbm.perf(model_gbm, method = "test")
>
```




```
> # Determine best iteration based on test data
> gbm.iter = gbm.perf(model_gbm, method = "test")
> model.influence = relative.influence(model_gbm, n.trees = gbm.iter, sort. = TRUE)
> plot(model_gbm)
>
```

Visualizing our model

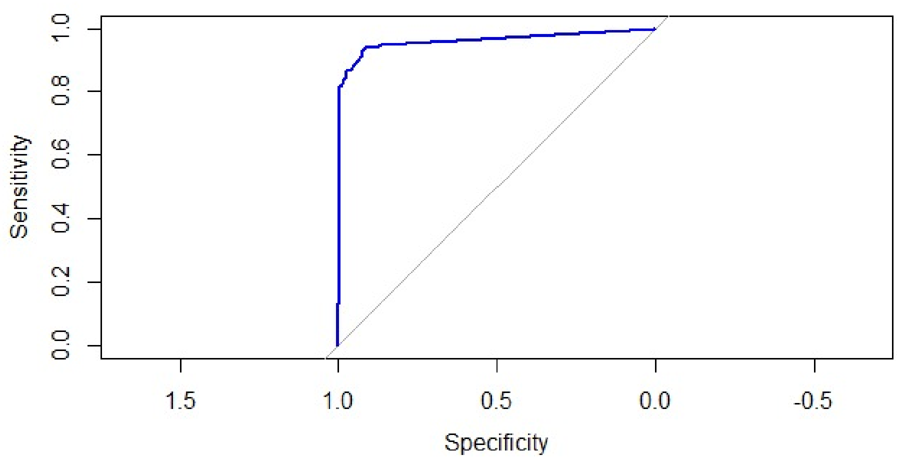


In order to assess the performance of our model, we will delineate the ROC curve. ROC is also known as Receiver Optimistic Characteristics. For this, we will first import the ROC package and then plot our ROC curve to analyze its performance.

```
> library(pROC)
> # Plot and calculate AUC on test data
> gbm_test = predict(model_gbm, newdata = test_data, n.trees = gbm.iter)
> gbm_auc = roc(test_data$Class, gbm_test, plot = TRUE, col = "Blue")
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> print(gbm_auc)

Call:
roc.default(response = test_data$Class, predictor = gbm_test,     plot = TRUE, col = "Blue")

Data: gbm_test in 56863 controls (test_data$Class 0) < 98 cases (test_data$Class 1).
Area under the curve: 0.9645
> |
```



Predicting the model accuracy by building the confusion matrix

```
R 4.1.0 · C:/Users/sruth/OneDrive/Desktop/CSE4027 LAB/
> #predicting the accuracy
> res<-predict(model_gbm,test_data,type = "response")
Using 450 trees...

> res
   [1] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937
   [7] 0.0002813525 0.0002754937 0.0002754937 0.0002754937 0.0002977037 0.0002754937
  [13] 0.0002754937 0.0002754937 0.0005824560 0.0002754937 0.0002754937 0.0002754937
  [19] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0003165984 0.0002807372
  [25] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002972570 0.0002754937
  [31] 0.0002754937 0.0002754937 0.0002754937 0.0002872079 0.0002754937 0.0002977037
  [37] 0.0002754937 0.0002754937 0.0003759469 0.0002754937 0.0002754937 0.0002754937
  [43] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937
  [49] 0.0002869425 0.0003165984 0.0002754937 0.0002754937 0.0002754937 0.0002754937
  [55] 0.0002754937 0.0002754937 0.0002754937 0.0002825317 0.0002754937 0.0002754937
  [61] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937
  [67] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002825317 0.0002754937
```

```
 [985] 0.0003165984 0.0002754937 0.0002977037 0.0002754937 0.0002879092 0.0002754937
 [991] 0.0002754937 0.0003165984 0.0003165984 0.3896726453 0.0002754937 0.0002754937
 [997] 0.0002754937 0.0002754937 0.0002754937 0.0002754937
 [ reached getOption("max.print") -- omitted 55961 entries ]
> restr<-predict(model_gbm,train_data,type = "response")
Using 450 trees...

> restr
   [1] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937
   [7] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0003165984
  [13] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937
  [19] 0.0002754937 0.0002825317 0.0002872079 0.0002754937 0.0002754937 0.0002754937
  [25] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937
  [31] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937
  [37] 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0002754937 0.0003165984
```

```
  all arguments must have the same length
> #Building  the confusion matrix for testing Data
> cmte<-table(Actual_Value=test_data$Class,Predicted_Value=res>0.5)
> cmte
            Predicted_Value
Actual_Value FALSE  TRUE
           0 56849    14
           1    27    71
> #accuracy
> accuracy=(cmte[[1,1]]+cmte[[2,2]])/sum(cmte)
> accuracy
[1] 0.9992802
```

```
  all arguments must have the same length
> #Building the confusion matrix for training data
> cm<-table(Actual_Value=train_data$Class,Predicted_Value=restr>0.5)
> cm
            Predicted_Value
Actual_Value  FALSE   TRUE
           0 227423     29
           1     89    305
> #Accuracy
> accuracy=(cm[[1,1]]+cm[[2,2]])/sum(cm)
> accuracy
[1] 0.9994821
>
```

6.CONCLUSION

Credit card fraud is the most common problem resulting in loss of a lot of money for people and loss for some banks and credit card companies. This project want to help the people from their wealth loss and also for the banked company and trying to develop the model which more efficiently separate the fraud and fraud less transaction by using the time and amount feature in data set given in the Kaggle. Out of all the 4 models , Decision tree is the best due to high accuracy rate and hence is best model for this particular dataset. We learned how to develop our credit card fraud detection model using machine learning. We used a variety of ML algorithms to implement this model and also plotted the respective performance curves for the models. We learnt how data can be analyzed and visualized to discern fraudulent transactions from other types of data.

7.REFERENCE

https://www.google.com/amp/s/spd.group/machine-learning/credit-card-fraud-detection/

https://en.m.wikipedia.org/wiki/Credit_card_fraud

https://www.google.com/amp/s/www.chargebackgurus.com/blog/credit-card-fraud-detection%3fhs_amp=true

8.BIBLIOGRAPHY

(1) WorldPay. (2015, Nov). Global payments report preview: your definitive guide to the world of online payments. Retrieved September 28, 2016.

(2) Federal Trade Commision. (2008). consumer sentinel network – data book for January - December 2008. Retrieved Oct 20, 2016.

(3) Bhatla, T.P., Prabhu, V., and Dua, A. (2003). understanding credit card frauds. Crads Business Review# 2003-1, Tata Consultancy Services.

(4)The Nilson Report. (2015). Global fraud losses reach $16.31 Billion. Edition: July 2015, Issue 1068.

(5) Y. Sahin and E. Duman, “Detecting credit card fraud by decision trees and support vector machines”, Proceedings of the International MultiConference of Engineers and Computer Scientists 2011 Vol I, IMECS 2011, March 2011.