GALGOTIAS UNIVERSITY
(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

CREDIT CARD FRAUD DETECTION

A Report for the Evaluation 3 of Project 2

Submitted by

ROHIT SINGHAL

(1713104003)

In partial fulfilment for the award of the degree

Of

BACHELOR OF COMPUTER APPLICATIONS

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Under the Supervision of

MR. PRABHAT CHANDRA GUPTA

Associate Professor

MAY- 2020

SCHOOL OF COMPUTING AND SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project report "**CREDIT CARD FRAUD DETECTION**" is the Bonafide work of **"ROHIT SINGHAL (1713104003)"** who carried out the project work under my supervision**.**

SIGNATURE OF HEAD
Dr. MUNISH SHABARWAL,
PhD (Management), PhD (CS)
Professor & Dean,
School of Computing Science & Engineering

SIGNATURE OF SUPERVISOR
MR. PRABHAT CHANDRA GUPTA
Associate Professor
School of Computing Science & Engineering

TABLE OF CONTENTS

S.NO. TITLE

1.ABSTRACT

Credit card plays a very important rule in today's economy. It becomes an unavoidable part of household, business and global activities. Although using credit cards provides enormous benefits when used carefully and responsibly, significant credit and financial damages may be caused by fraudulent activities. Many techniques have been proposed to confront therewith credit card fraud. However, all of these techniques have the same goal of avoiding the credit card fraud; each one has its own drawbacks, advantages and characteristics. In this paper, after investigating difficulties of credit card fraud detection, we seek to review the state of the art in credit card fraud detection techniques, datasets and evaluation criteria. The advantages and disadvantages of fraud detection methods are enumerated and compared. Furthermore, a classification of mentioned techniques into two main fraud detection approaches, namely, misuses (supervised) and anomaly detection (unsupervised) is presented. Again, a classification of techniques is proposed based on capability to process the numerical and categorical datasets. Different datasets used in literature are then described and grouped into real and synthesized data and the effective and common attributes are extracted for further usage. Moreover, evaluation employed criterions in literature are collected and discussed. Consequently, open issues for credit card fraud detection are explained as guidelines for new researchers.

2.LITERATURE SURVEY

Credit card fraud detection has drawn a lot of research interest and a number of techniques, with special emphasis on neural networks, data mining and distributed data mining have been suggested.

Ghosh and Reilly have proposed credit card fraud detection with a neural network. They have built a detection system, which is trained on a large sample of labelled credit card account transactions. These transactions contain example fraud cases due to lost cards, stolen cards, application fraud, counterfeit fraud, mail-order fraud, and non-received issue (NRI) fraud. Recently, Syeda et al. have used parallel granular neural networks (PGNNs) for improving the speed of data mining and knowledge discovery process in credit card fraud detection.

A complete system has been implemented for this purpose. Stolfo et al. suggest a credit card fraud detection system (FDS) using Meta learning techniques to learn models of fraudulent credit card transactions.

Meta learning is a general strategy that provides a means for combining and integrating a number of separately built classifiers or models. A Meta classifier is thus trained on the correlation of the predictions of the base classifiers. The same group has also worked on a cost-based model for fraud and intrusion detection. They use Java agents for Meta learning (JAM), which is a distributed data mining system for credit card fraud detection A number of important performance metrics like True Positive—False Positive (TP-FP) spread and accuracy have been defined by them. Alekerov et al. present CARDWATCH, a database mining system used for credit card fraud detection. The system, based on a neural learning module, provides an interface to a variety of commercial databases.

Kim and Kim have identified skewed distribution of data and mix of legitimate and fraudulent transactions as the two main reasons for the complexity of credit card fraud detection. Based on this observation, they use fraud density of real transaction data as a confidence value and generate the weighted fraud score to reduce the number of misdetections.

3. INTRODUCTION

a. PURPOSE

To detect the fraud transactions taking place in the users accounts by their card duplicity or any such irreveleant procedures.

Reduce the fraud activities and overcoming the losses.

b. MOTIVATION

Increased used of the plastic money over the hard cash.

Increasing online transaction over the network.

Increased number of user requirement of the safer turnovers and transactions.

c.**SCOPE**

Can be highly developed and reduce more fraud activities.

Highley complexity can increase the detection of the irregular activities.

4.**SYSTEM ANALYSIS**

a.**PRPOSED SYSTEM**

In proposed system, we present a Hidden Markov Model (HMM). Which does not require fraud signatures and yet is able to detect frauds by considering a cardholder's spending habit. Card transaction processing sequence by the stochastic process of an HMM. The details of items purchased in Individual transactions are usually not known to any Fraud Detection System(FDS) running at the bank that issues credit cards to the cardholders. Hence, we feel that HMM is an ideal choice for addressing this problem.

Another important advantage of the HMM-based approach is a drastic reduction in the number of False Positives transactions identified as malicious by an FDS although they are actually genuine. An FDS runs at a credit card issuing bank. Each incoming transaction is submitted to the FDS for verification. FDS receives the card details and the value of purchase to verify, whether the transaction is genuine or not. The types of goods that are bought in that transaction are not known to the FDS. It tries to find any anomaly in the transaction based on the spending profile of the cardholder, shipping address, and billing address, etc. If the FDS confirms the transaction to be of fraud, it raises an alarm, and the issuing bank declines the transaction.

b.**EXISTING SYSTEM**

The detection of the fraud use of the card is found much faster that the existing system.

In case of the existing system even the original card holder is also checked for fraud detection. But in this system no need to check the original user as we maintain a log.

The log which is maintained will also be a proof for the bank for the transaction made.

We can find the most accurate detection using this technique.

This reduce the tedious work of an employee in the bank

c.CORE FETURES

The system stores previous transaction patterns for each user.

Based upon the user spending ability and even country, it calculates user's characteristics.

More than 20 -30 %deviation of user's transaction (spending history and operating country) is considered as an invalid attempt and system takes action.

5.METHODOLOGY ADOPTED

a.IMPORTING THE DATASHEETS

We are importing the datasets that contain transactions made by creditcards.

Code:

1. library(ranger)
2. library(caret)
3. library(data.table)
4. creditcard_data <- read.csv("/home/dataflair/data/Credit Card/creditcard.csv")

Input Screenshot:

```
library(ranger)
library(caret)
```

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

```
library(data.table)
creditcard_data <- read.csv("/home/dataflair/data/Credit Card/creditcard.csv")
```

b.DATA EXPLORATION

In this section of the fraud detection ML project, we will explore the data that is contained in the creditcard_datadataframe. We will proceed by displaying the creditcard_data using the head () function as well as the tail () function. We will then proceed to explore the other components of this data frame.

Code:

1. **dim**(creditcard_data)
2. **head**(creditcard_data,6)

Output Screenshot:

```
dim(creditcard_data)
```

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

```
head(creditcard_data,6)
```

```
##   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
```

Code:

1. **tail**(creditcard_data,6)

Output Screenshot

```
tail(creditcard_data,6)
```

```
##          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
## 284802  0.19091623 -0.5463289 -0.73170658 -0.80803553  0.5996281
## 284803  2.71194079 -0.6892556  4.62694203 -0.92445871  1.1076406
## 284804  0.91580191  1.2147558 -0.67514296  1.16493091 -0.7117573
## 284805  0.06311886 -0.1836987 -0.51060184  1.32928351  0.1407160
## 284806 -0.96288614 -1.0420817  0.44962444  1.96256312 -0.6085771
## 284807 -0.03151305 -0.1880929 -0.08431647  0.04133346 -0.3026201
```

Code:

1. **table**(creditcard_data$Class)
2. **summary**(creditcard_data$Amount)
3. **names**(creditcard_data)
4. **var**(creditcard_data$Amount)

Output screenshots:

```
table(creditcard_data$Class)
```

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

```
summary(creditcard_data$Amount)
```

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

```
names(creditcard_data)
```

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

```
var(creditcard_data$Amount)
```

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

Code:

1. **sd**(creditcard_data$Amount)

output screenshot:

```
sd(creditcard_data$Amount)
```

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

c.DATA MANUPULATION

In this section of the R data science project, we will scale our data using the scale () function. We will apply this to the amount component of our creditcard_data amount. Scaling is also known as feature standardization. With the help of scaling, the data is structured according to a specified range. Therefore, there are no extreme values in our dataset that might interfere with the functioning of our model.

Code:

1. **head**(creditcard_data)

Output Screenshot:

```
head(creditcard_data)
```

```
##   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
```

Code:

1. creditcard_data$Amount=**scale**(creditcard_data$Amount)
2. NewData=creditcard_data[,**-c**(1)]
3. **head**(NewData)

Output Screenshot:

```
creditcard_data$Amount=scale(creditcard_data$Amount)
NewData=creditcard_data[,-c(1)]
head(NewData)
```

```
##            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
## 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
```

d.DATA MODELING

After we have standardized 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.

Code:

1. **library**(caTools)
2. set.**seed**(123)
3. data_sample = sample.**split**(NewData$Class,SplitRatio=0.80)
4. train_data = **subset**(NewData,data_sample==TRUE)
5. test_data = **subset**(NewData,data_sample==FALSE)

6. **dim**(train_data)
7. **dim**(test_data)

Output Screenshot:

```
library(caTools)
set.seed(123)
data_sample = sample.split(NewData$Class,SplitRatio=0.80)
train_data = subset(NewData,data_sample==TRUE)
test_data = subset(NewData,data_sample==FALSE)
dim(train_data)
```

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

```
dim(test_data)
```

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

```
 Logistic_Model=glm(Class~.,test_data,family=binomial())
```

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

```
 summary(Logistic_Model)
```

```
##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = test_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.9019  -0.0254  -0.0156  -0.0078   4.0877
```

e.FITTING LOGISTIC REGRESSION MODEL

In this section of credit card fraud detection project, we will fit our first model. We will begin with logistic regression. 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.

Code:

1. Logistic_Model=**glm**(Class~.,test_data,family=**binomial**())
2. **summary**(Logistic_Model)

Output Screenshot:

```
Logistic_Model=glm(Class~.,test_data,family=binomial())
```

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

```
summary(Logistic_Model)
```

```
##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = test_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.9019  -0.0254  -0.0156  -0.0078   4.0877
```

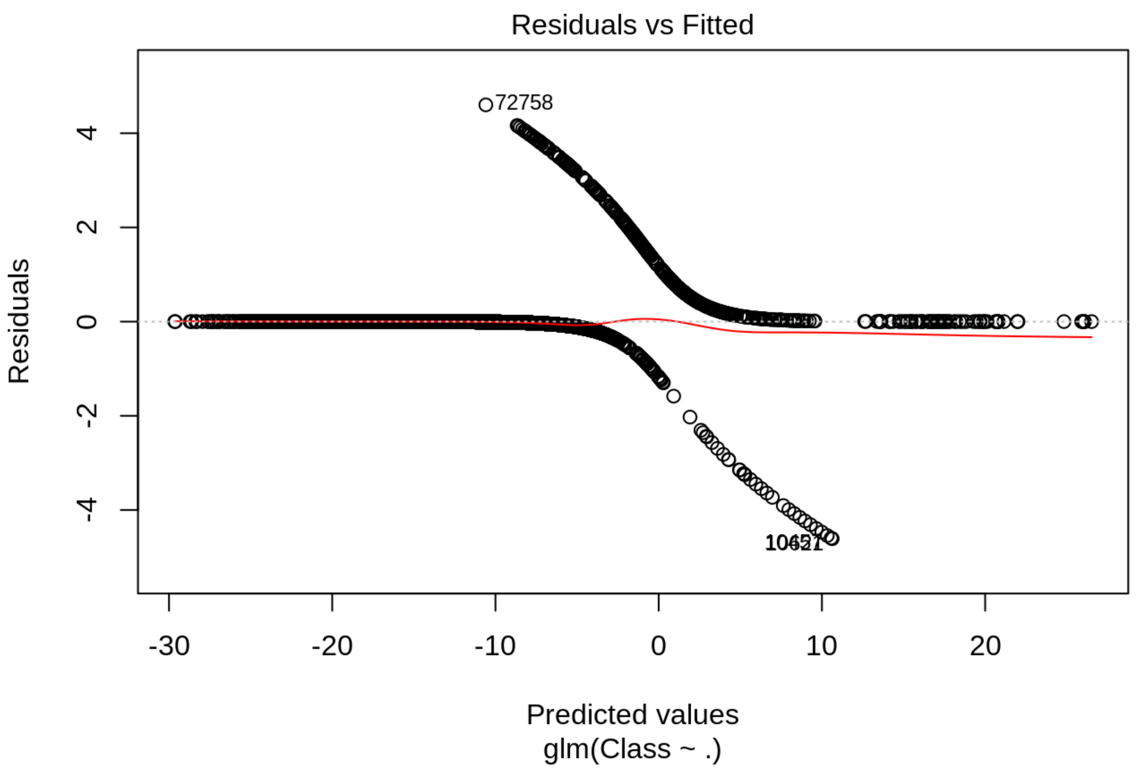
Code:

1. **plot**(Logistic_Model)

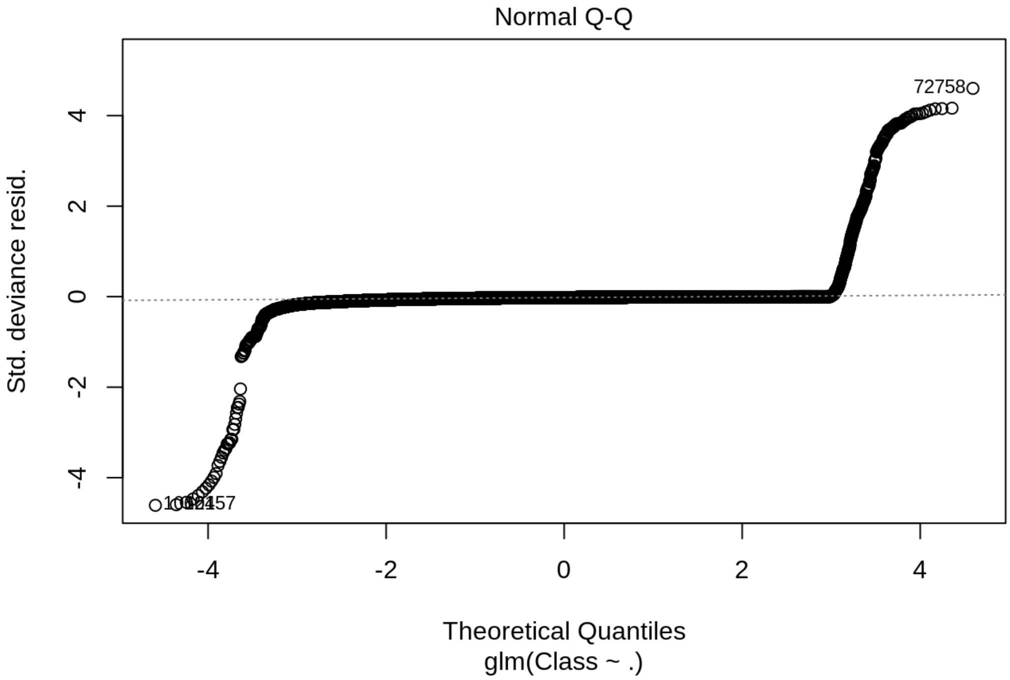
Input Screenshot:

```
plot(Logistic_Model)
```

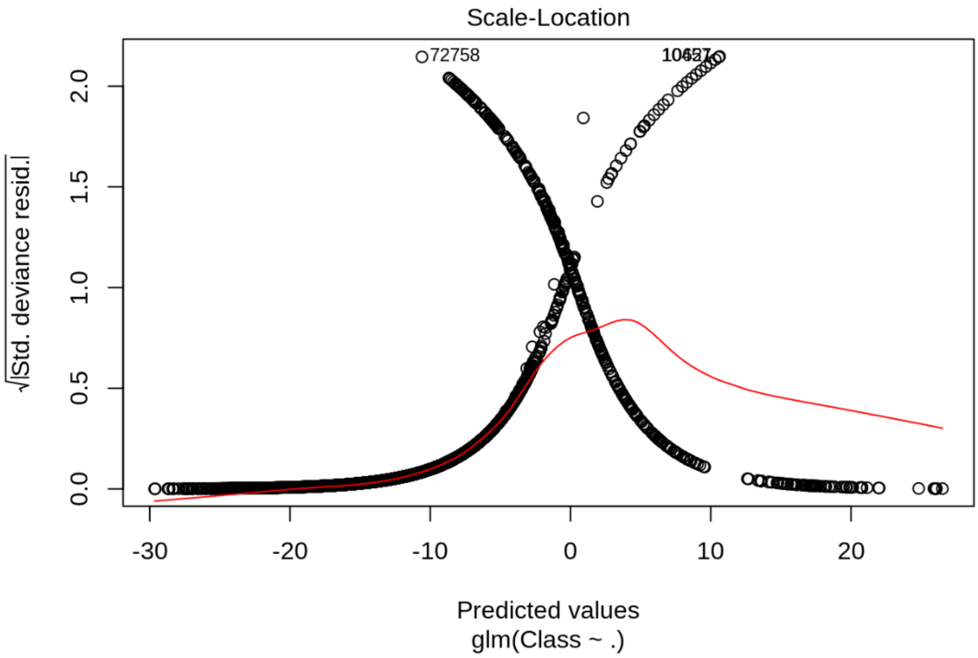
Output:



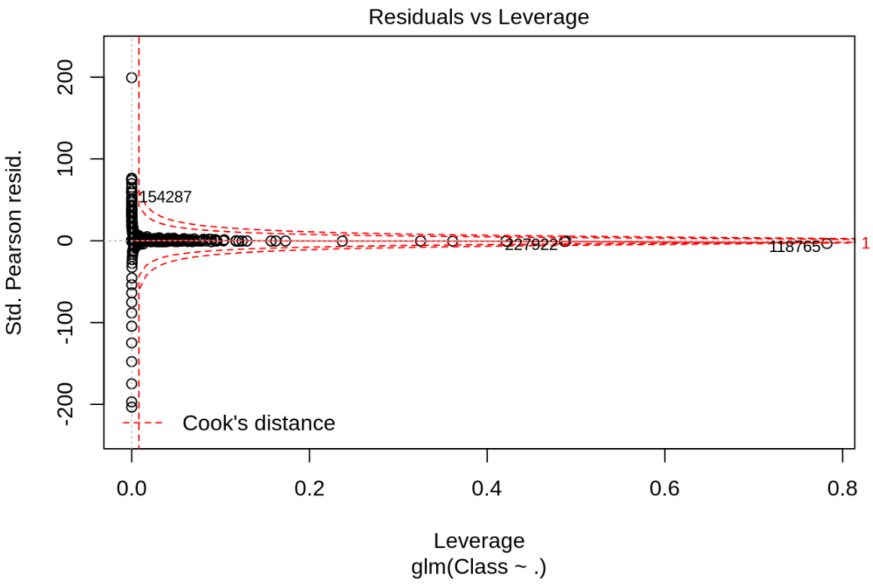
Output:



Output:



Output:



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.

Code:

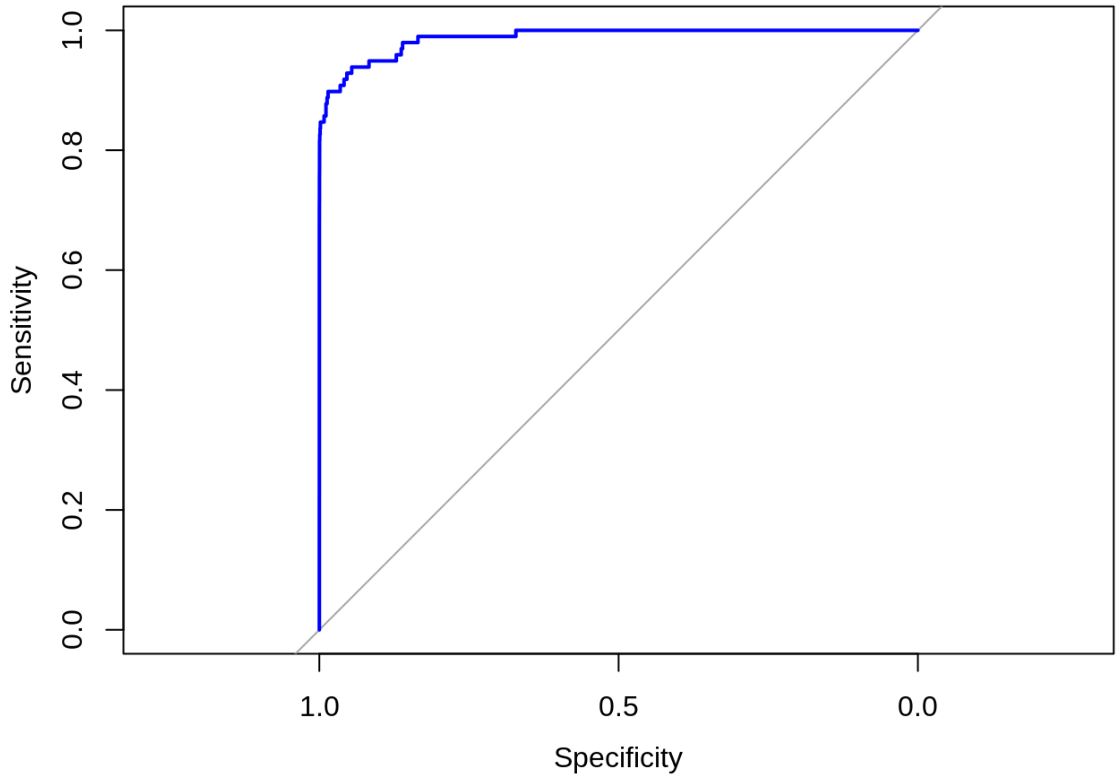
1. **library**(pROC)
2. lr.predict <- **predict**(Logistic_Model,train_data, probability = TRUE)
3. auc.gbm = **roc**(test_data$Class, lr.predict, plot = TRUE, col = "blue")

Output Screenshot:

```
Logistic_Model=glm(Class~.,train_data,family=binomial())
summary(Logistic_Model)
```

```
##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.6108  -0.0292  -0.0194  -0.0125   4.6021
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.651305   0.160212 -53.999  < 2e-16 ***
## V1           0.072540   0.044144   1.643 0.100332
## V2           0.014818   0.059777   0.248 0.804220
```

Output:



f. FITTING A DECISION TREE MODEL

In this section, we will implement a decision tree algorithm. **Decision Trees** to plot the outcomes of a decision. These outcomes are basically a consequence through which we can conclude as to what class the object belongs to. We will now implement our decision tree model and will plot it using the rpart.plot() function.

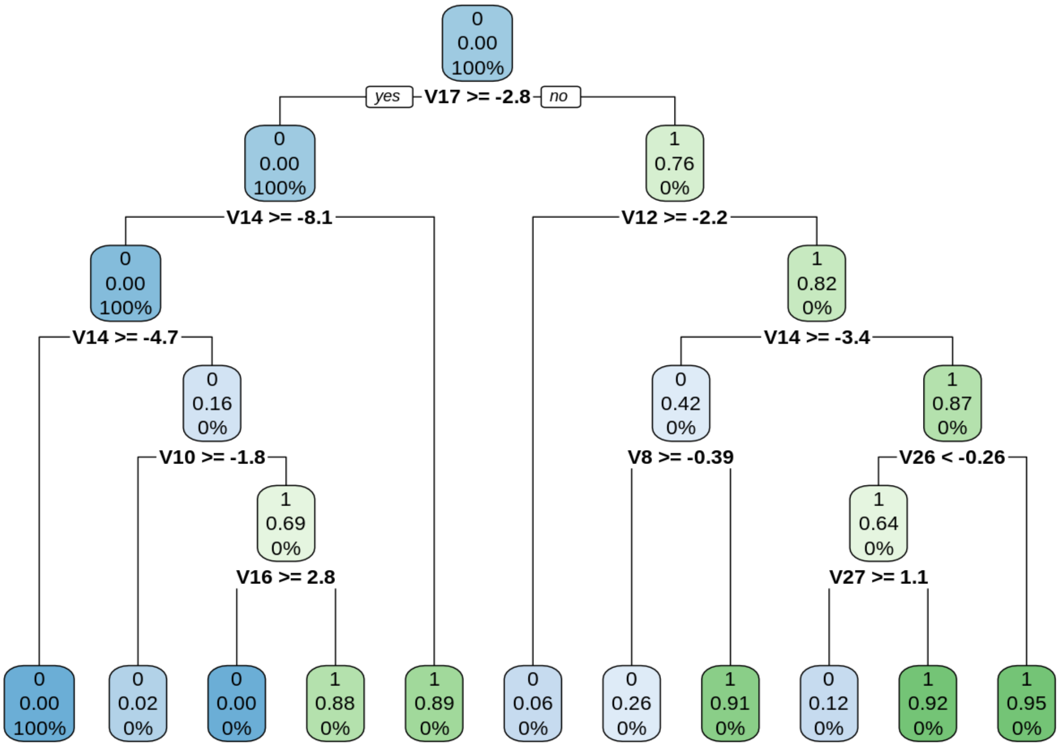
Code:

1. **library**(rpart)
2. **library**(rpart.plot)
3. decisionTree_model <**-** **rpart**(Class ~ . , creditcard_data, method = 'class')
4. predicted_val <**-** **predict**(decisionTree_model, creditcard_data, type = 'class')
5. probability <- **predict**(decisionTree_model, creditcard_data, type = 'prob')
6. rpart.**plot**(decisionTree_model)

Input Screenshot:

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

rpart.plot(decisionTree_model)
```

Output:



g.ATRIFICIAL NEURAL NETWORK

Artificial Neural Networks are a type of machine learning algorithm that is modeled after the human nervous system. The ANN models are able to learn the patterns using the historical data and are able to perform classification on the input data. We import the neuralnet 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

Code:

```
1. library(neuralnet)
```

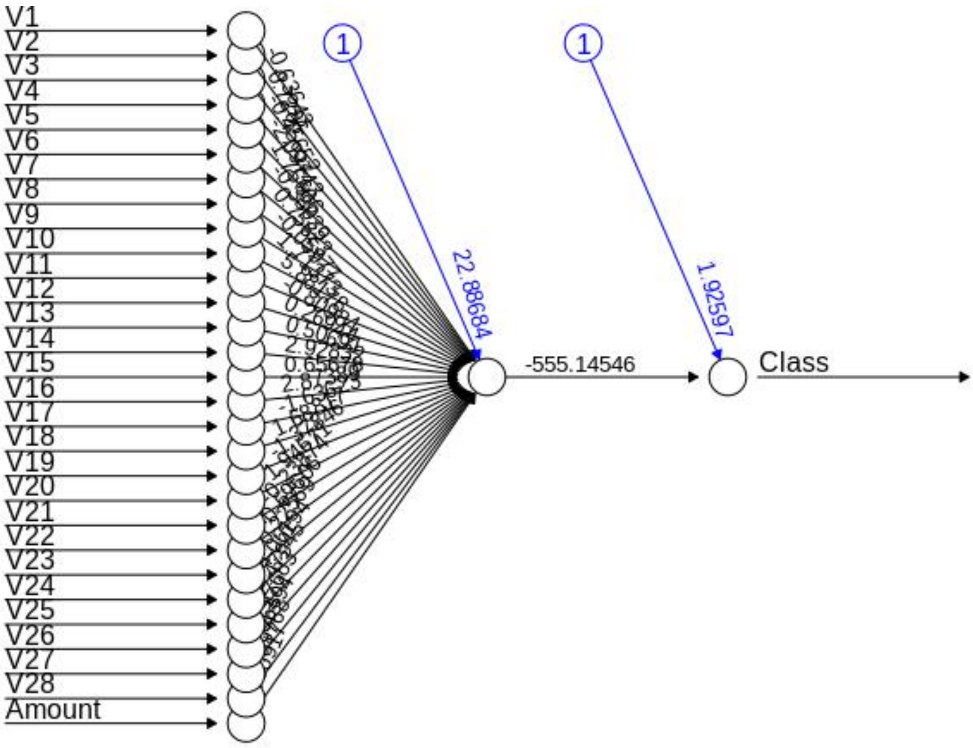
2. ANN_model =**neuralnet** (Class~.,train_data,linear.output=FALSE)
3. **plot**(ANN_model)
4.
5. predANN=**compute**(ANN_model,test_data)
6. resultANN=predANN$net.result
7. resultANN=**ifelse**(resultANN>0.5,1,0)

Input Screenshot:

```
library(neuralnet)
ANN_model =neuralnet (Class~.,train_data,linear.output=FALSE)
plot(ANN_model)
```

```
predANN=compute(ANN_model,test_data)
resultANN=predANN$net.result
resultANN=ifelse(resultANN>0.5,1,0)
```

Output:



h.GRADIENT BOOSTING(GSM)

Gradient Boosting is a popular machine learning algorithm that is used to perform classification and regression tasks. 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.

Code:

```
library(gbm, quietly=TRUE)

# Get the time to train the GBM model
system.time(
model_gbm <- gbm(Class ~ .
, distribution = "bernoulli"
, data = rbind(train_data, test_data)
, n.trees = 500
, interaction.depth = 3
, n.minobsinnode = 100
, shrinkage = 0.01
, bag.fraction = 0.5
, train.fraction = nrow(train_data) / (nrow(train_data) + nrow(test_data))
)
)
# Determine best iteration based on test data
gbm.iter = gbm.perf(model_gbm, method = "test")
```

Input Screenshot:

```
library(gbm, quietly=TRUE)
```

```
## Loaded gbm 2.1.5
```

```
# Get the time to train the GBM model
system.time(
    model_gbm <- gbm(Class ~ .
        , distribution = "bernoulli"
        , data = rbind(train_data, test_data)
        , n.trees = 500
        , 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
## 345.781   0.144 345.971
```

```
# Determine best iteration based on test data
gbm.iter = gbm.perf(model_gbm, method = "test")
```

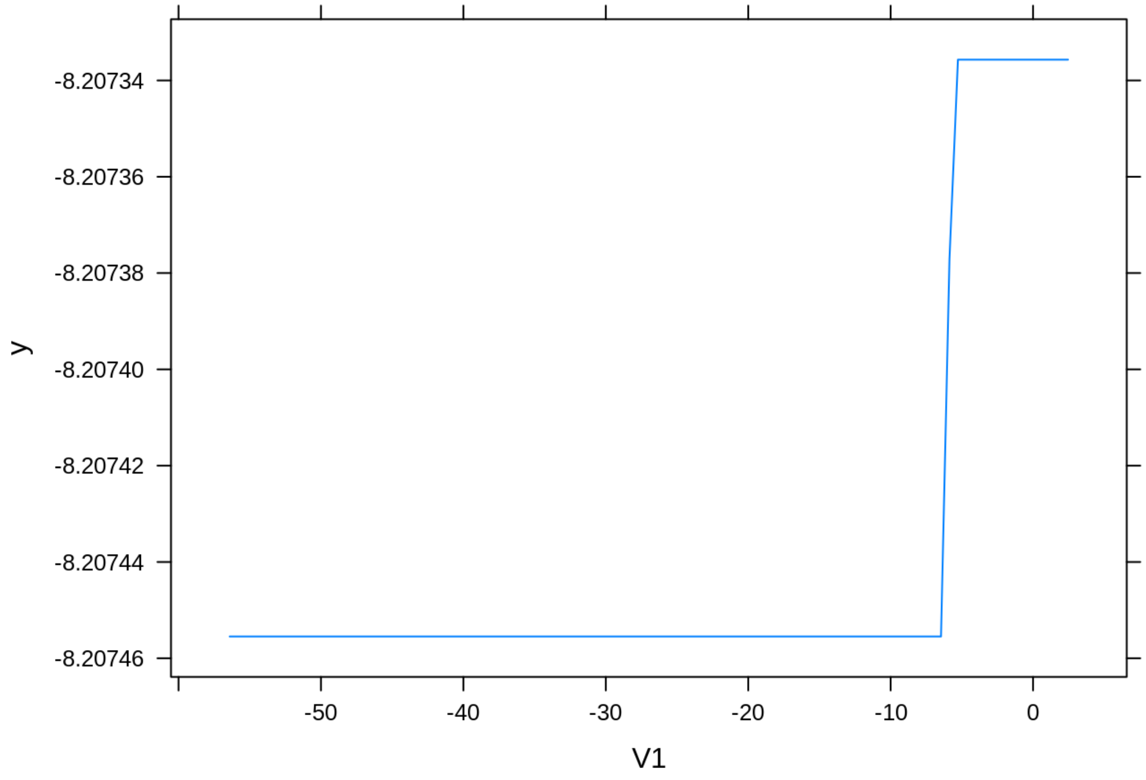
Code:

```
model. influence = relative. influence (model_gbm, n. trees = gbm.iter, sort. = TRUE)
#Plot the gbm model

plot(model_gbm)
```

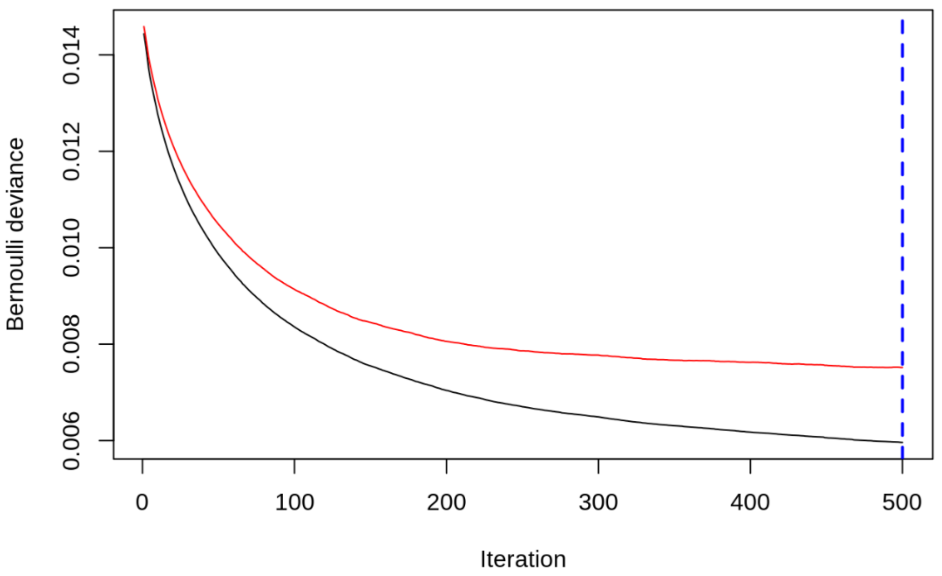
Input Screenshot:

```
model.influence = relative.influence(model_gbm, n.trees = gbm.iter, sort. = TRUE)
#Plot the gbm model

plot(model_gbm)
```

Output:



Output:



Code:

1. # Plot and calculate AUC on test data
2. gbm_test = **predict** (model_gbm, newdata = test_data, n. trees = gbm. iter)
3. gbm_auc = **roc** (test_data$Class, gbm_test, plot = TRUE, col = "red")

Output Screenshot:

```
# 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 = "red")
```

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

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

Code:

1. **print**(gbm_auc)

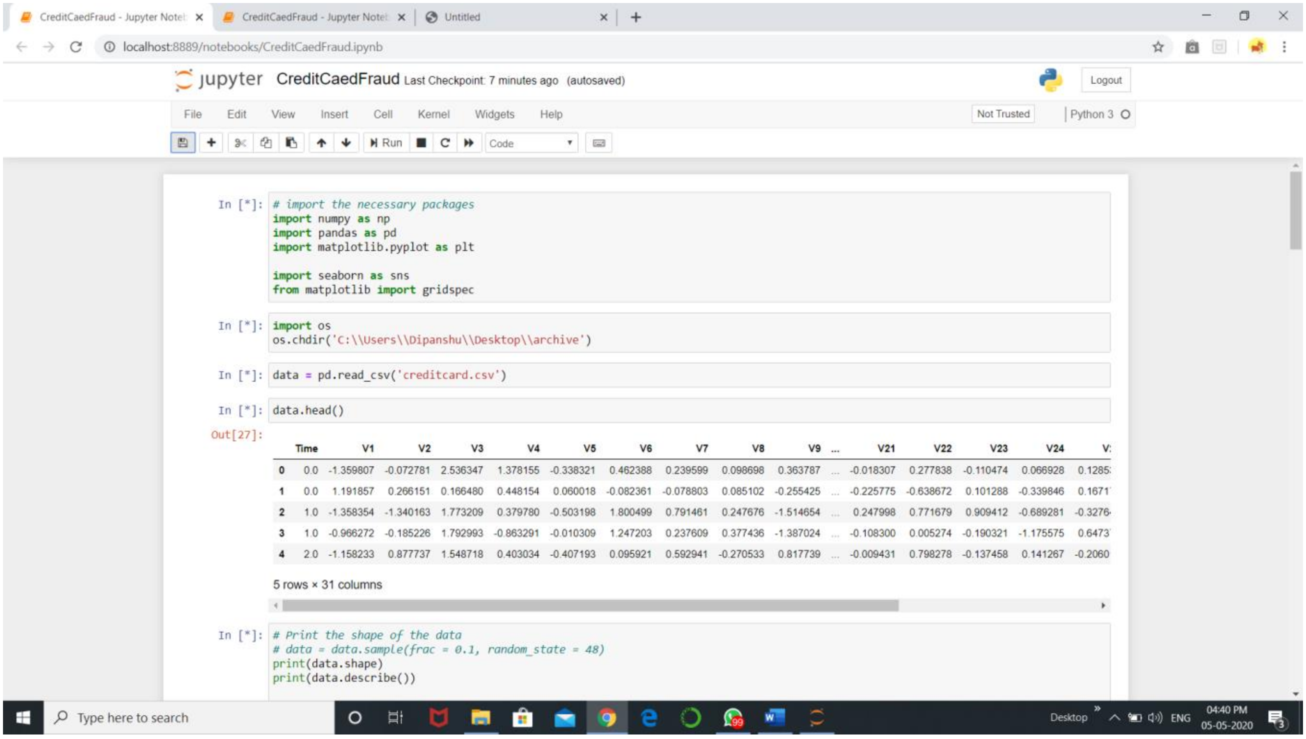
Output Screenshot:

```
print(gbm_auc)
```

```
##
## Call:
## roc.default(response = test_data$Class, predictor = gbm_test,     plot = TRUE, col = "red")
##
## Data: gbm_test in 56863 controls (test_data$Class 0) < 98 cases (test_data$Class 1).
## Area under the curve: 0.9555
```

6.IMPLIMENTATION AND SCREENSHOTS AND IMAGES OF THR RUNNING PROJECTS

i)



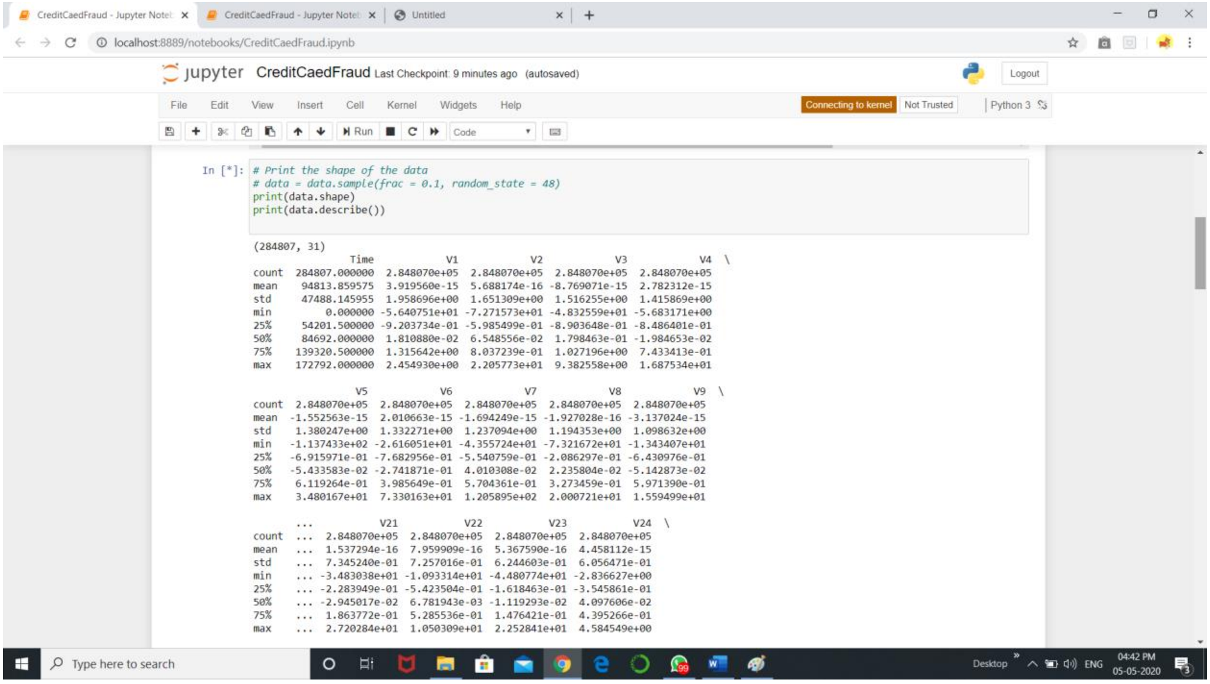
```
In [*]: # import the necessary packages
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt

        import seaborn as sns
        from matplotlib import gridspec

In [*]: import os
        os.chdir('C:\\Users\\Dipanshu\\Desktop\\archive')

In [*]: data = pd.read_csv('creditcard.csv')

In [*]: data.head()
Out[27]:
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V:
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.1285
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.1671
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.3276
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.6473
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.2060

5 rows × 31 columns

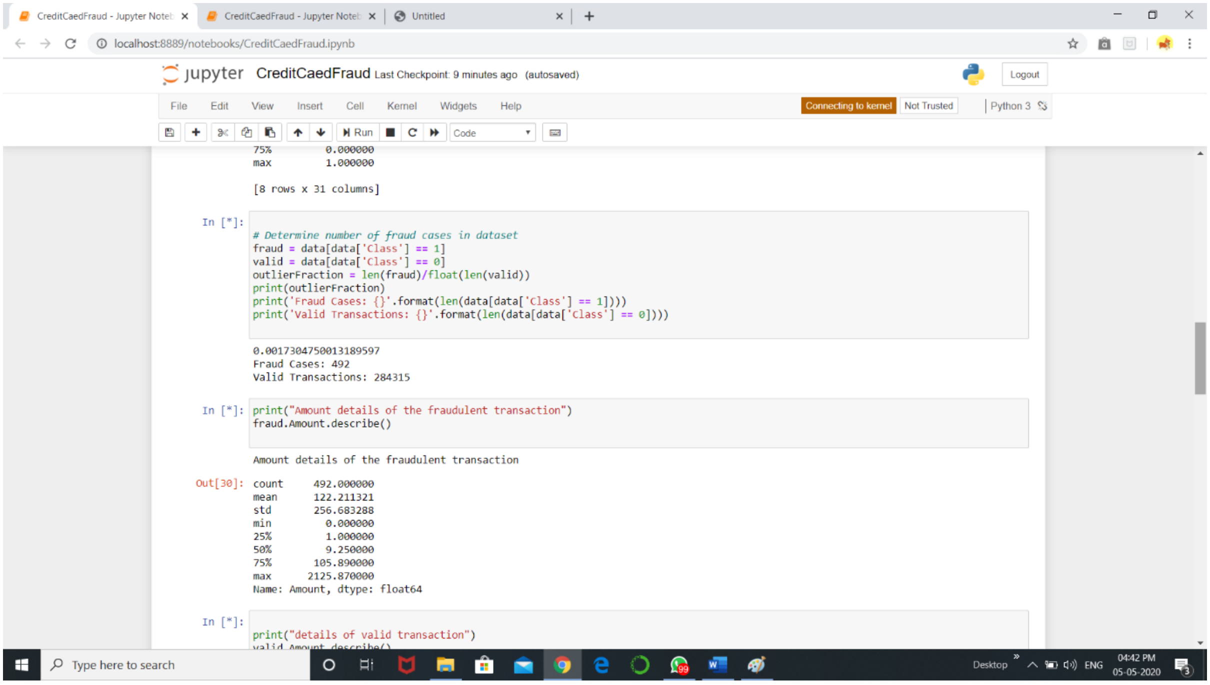
```
In [*]: # Print the shape of the data
        # data = data.sample(frac = 0.1, random_state = 48)
        print(data.shape)
        print(data.describe())
```

ii)

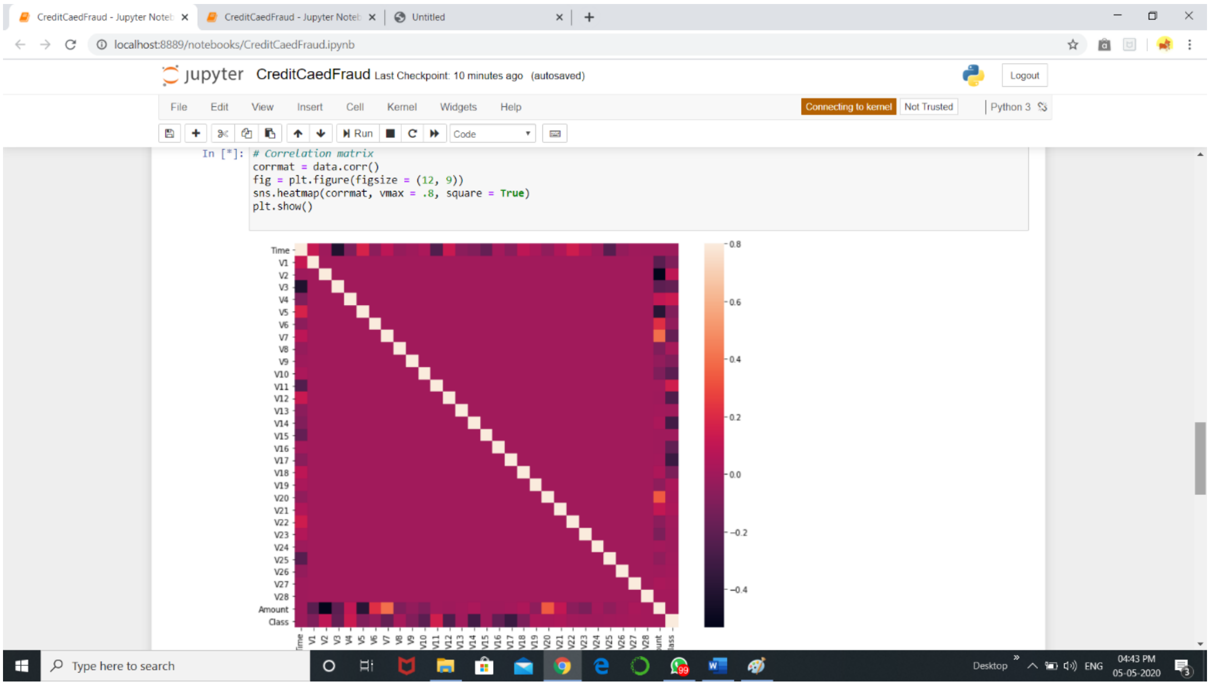


CreditCaedFraud - Jupyter Notel
CreditCaedFraud - Jupyter Notel
Untitled
localhost:8889/notebooks/CreditCaedFraud.ipynb
jupyter CreditCaedFraud Last Checkpoint: 9 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Connecting to kernel
Not Trusted
Python 3
Run
Code
In [*]:
# Print the shape of the data
# data = data.sample(frac = 0.1, random_state = 48)
print(data.shape)
print(data.describe())
(284807, 31)
Time V1 V2 V3 V4 \
count 284807.000000 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05
mean 94813.859575 3.919560e-15 5.688174e-16 -8.769071e-15 2.782312e-15
std 47488.145955 1.958696e+00 1.651309e+00 1.516255e+00 1.415869e+00
min 0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25% 54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50% 84692.000000 1.810880e-02 6.548556e-02 1.798463e-01 -1.984653e-02
75% 139320.500000 1.315642e+00 8.037239e-01 1.027196e+00 7.433413e-01
max 172792.000000 2.454930e+00 2.205773e+01 9.382558e+00 1.687534e+01
V5 V6 V7 V8 V9 \
count 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05
mean -1.552563e-15 2.010663e-15 -1.694249e-15 -1.927028e-16 -3.137024e-15
std 1.380247e+00 1.332271e+00 1.237094e+00 1.194353e+00 1.098632e+00
min -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25% -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50% -5.433583e-02 -2.741871e-01 4.010308e-02 2.235804e-02 -5.142873e-02
75% 6.119264e-01 3.985649e-01 5.704361e-01 3.273459e-01 5.971390e-01
max 3.480167e+01 7.330163e+01 1.205895e+02 2.000721e+01 1.559499e+01
... V21 V22 V23 V24 \
count ... 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05
mean ... 1.537294e-16 7.959909e-16 5.367590e-16 4.458112e-15
std ... 7.345240e-01 7.257016e-01 6.244603e-01 6.056471e-01
min ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25% ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50% ... -2.945017e-02 6.781943e-03 -1.119293e-02 4.097606e-02
75% ... 1.863772e-01 5.285536e-01 1.476421e-01 4.395266e-01
max ... 2.720284e+01 1.050309e+01 2.252841e+01 4.584549e+00
Type here to search
Desktop
ENG
04:42 PM
05-05-2020

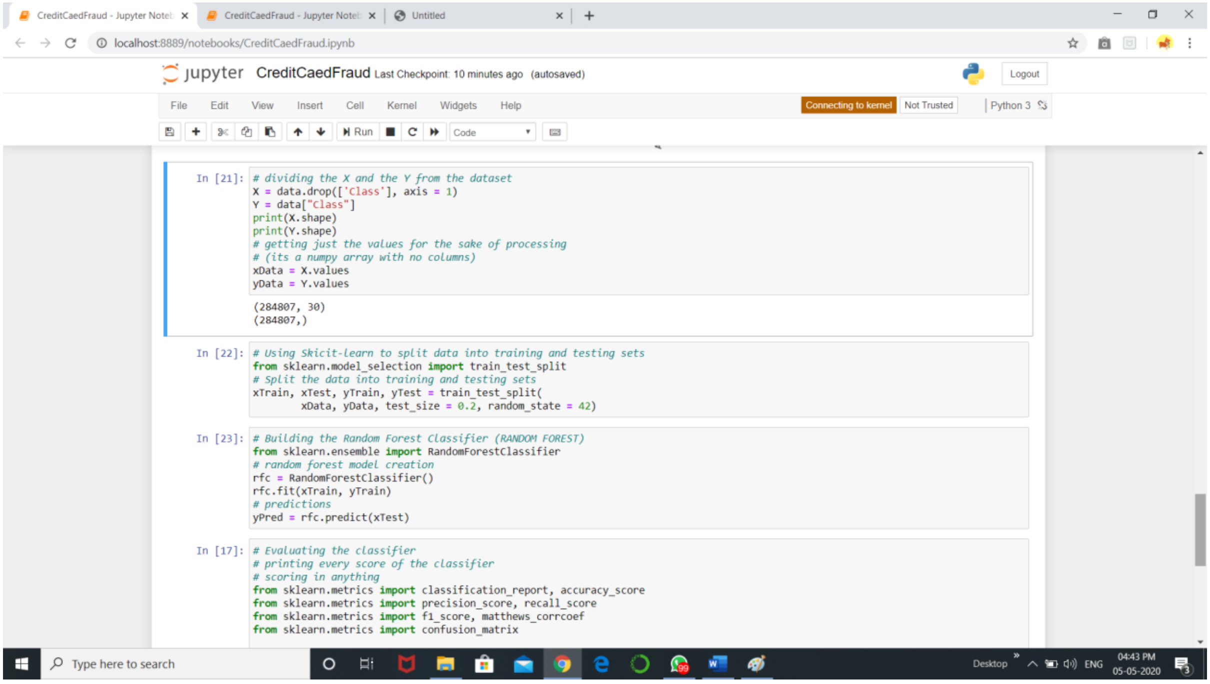
iii)


CreditCaedFraud - Jupyter Noteb
CreditCaedFraud - Jupyter Noteb
Untitled
localhost:8889/notebooks/CreditCaedFraud.ipynb
jupyter CreditCaedFraud Last Checkpoint: 9 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Connecting to kernel
Not Trusted
Python 3
Run
Code
75%   0.000000
max   1.000000
[8 rows x 31 columns]
In [*]:
# Determine number of fraud cases in dataset
fraud = data[data['Class'] == 1]
valid = data[data['Class'] == 0]
outlierFraction = len(fraud)/float(len(valid))
print(outlierFraction)
print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
0.0017304750013189597
Fraud Cases: 492
Valid Transactions: 284315
In [*]:
print("Amount details of the fraudulent transaction")
fraud.Amount.describe()
Amount details of the fraudulent transaction
Out[30]:
count   492.000000
mean   122.211321
std   256.683288
min   0.000000
25%   1.000000
50%   9.250000
75%   105.890000
max   2125.870000
Name: Amount, dtype: float64
In [*]:
print("details of valid transaction")
Type here to search
Desktop
ENG
04:42 PM
05-05-2020

iv)

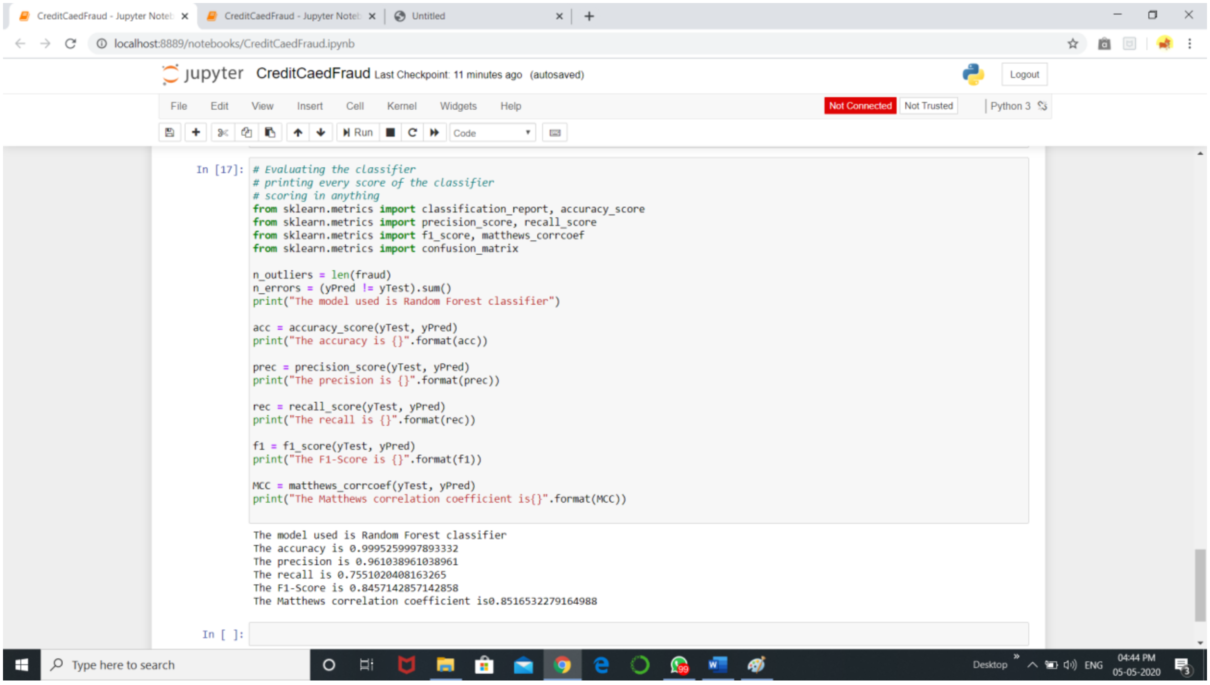

CreditCaedFraud - Jupyter Notebook
localhost:8889/notebooks/CreditCaedFraud.ipynb
jupyter CreditCaedFraud Last Checkpoint: 10 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Connecting to kernel
Not Trusted
Python 3
Run
Code
In [*]:
# Correlation matrix
corrmat = data.corr()
fig = plt.figure(figsize = (12, 9))
sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()
Type here to search
Desktop
ENG
04:43 PM
05-05-2020

v)



CreditCaedFraud - Jupyter Notel
CreditCaedFraud - Jupyter Notel
Untitled
localhost:8889/notebooks/CreditCaedFraud.ipynb
jupyter CreditCaedFraud Last Checkpoint: 10 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Connecting to kernel
Not Trusted
Python 3
Run
Code
In [21]: # dividing the X and the Y from the dataset
X = data.drop(['Class'], axis = 1)
Y = data["Class"]
print(X.shape)
print(Y.shape)
# getting just the values for the sake of processing
# (its a numpy array with no columns)
xData = X.values
yData = Y.values
(284807, 30)
(284807,)
In [22]: # Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
xTrain, xTest, yTrain, yTest = train_test_split(
xData, yData, test_size = 0.2, random_state = 42)
In [23]: # Building the Random Forest Classifier (RANDOM FOREST)
from sklearn.ensemble import RandomForestClassifier
# random forest model creation
rfc = RandomForestClassifier()
rfc.fit(xTrain, yTrain)
# predictions
yPred = rfc.predict(xTest)
In [17]: # Evaluating the classifier
# printing every score of the classifier
# scoring in anything
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, matthews_corrcoef
from sklearn.metrics import confusion_matrix
Type here to search
Desktop
ENG
04:43 PM
05-05-2020

vi)


CreditCaedFraud - Jupyter Notel
CreditCaedFraud - Jupyter Notel
Untitled
localhost:8889/notebooks/CreditCaedFraud.ipynb
jupyter CreditCaedFraud Last Checkpoint: 11 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Connected
Not Trusted
Python 3
Run
Code
In [17]:
# Evaluating the classifier
# printing every score of the classifier
# scoring in anything
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, matthews_corrcoef
from sklearn.metrics import confusion_matrix
n_outliers = len(fraud)
n_errors = (yPred != yTest).sum()
print("The model used is Random Forest classifier")
acc = accuracy_score(yTest, yPred)
print("The accuracy is {}".format(acc))
prec = precision_score(yTest, yPred)
print("The precision is {}".format(prec))
rec = recall_score(yTest, yPred)
print("The recall is {}".format(rec))
f1 = f1_score(yTest, yPred)
print("The F1-Score is {}".format(f1))
MCC = matthews_corrcoef(yTest, yPred)
print("The Matthews correlation coefficient is{}".format(MCC))
The model used is Random Forest classifier
The accuracy is 0.9995259997893332
The precision is 0.961038961038961
The recall is 0.7551020408163265
The F1-Score is 0.8457142857142858
The Matthews correlation coefficient is0.8516532279164988
In [ ]:
Type here to search
Desktop
ENG
04:44 PM
05-05-2020

7. RESULTS AND OBSERVATIONS

The data set is highly skewed, consisting of 492 frauds in a total of 284,807 observations. This resulted in only 0.172% fraud cases. This skewed set is justified by the low number of fraudulent transactions. The dataset consists of numerical values from the 28 'Principal Component Analysis (PCA)' transformed features, namely V1 to V28. Furthermore, there is no metadata about the original features provided, so pre-analysis or feature study could not be done.

- The 'Time' and 'Amount' features are not transformed data.
- There is no missing value in the dataset.

8. SOFTEWARE AND HARDWARE SPECIFICATIONS REQUIREMENTS

PYTHON BASED SOFTWARE
EXAMPLE: ANACONDA, R. STUDIO, IDLE ETC

- Processor: Preferably 1.0 GHz or Greater.
- RAM: 2 GB or Greater.

9. FUTURE SCOPE

Can be highly developed and reduce more fraud activities.

Highley complexity can increase the detection of the irregular activities.

10. PROBLEM STATEMENT

The Credit Card Fraud Detection Problem includes modelling past credit card transactions with the knowledge of the ones that turned out to be 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.

11. REFERENCES

R. J. Bolton and D. J. Hand. Unsupervised profiling methods for fraud detection. In conference of Credit Scoring and Credit Connol VII, Edinburgh. UK, Sept 5-7,2001.

Khyati Chaudhary, Jyoti Yadav, Bhawna Mallick, ―A review of Fraud Detection Techniques: Credit Card‖, International Journal of Computer Applications (0975 – 8887) Volume 45– No.1, May 2012.

K. C. Cox, S. G. Eick, G. J. Wills, and R. J. Brachman. Visual data mining: Recognizing telephone calling fraud's Data Mining and Knowledge Discover, 1(2):22>231, 1997.

Hollmn and Jaakko. Pmbabilistic Appmaches to Fraud Detecrion, Licentiate's ntesis. Helsinki University of Technology, Department of Computer Science and Engineering, 1999.

https://rpubs.com/slazien/fraud_detection

12. CONCLUSION

PROPERLY WORKING OF THE PROGRAM.
HIGH RATE FOR THE FRAUD DETECTION.
UPTO 99% DETECTION.
ASSUMED APPROX 384000 TRANSACTIONS FROM WHICH 492 FRAUDS DETECTED.