**CREDIT CARD FRAUD DETECTION**

**DATA SOURCE :**

The dataset here contains transactions made by credit cards in September 2013 by European cardholders. This dataset from Kaggle is available here. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we do not have access to the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are ‘Time’ and ‘Amount’. Feature ‘Time’ contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature ‘Amount’ is the transaction Amount, this feature can be used for example-dependant cost-senstive learning. Feature ‘Class’ is the response variable and it takes value 1 in case of fraud and 0 otherwise.

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

Dataset Link: <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>

.

Importing the libraries :

*# importing the required libraries*

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

##

## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':

##

## filter, lag

## The following objects are masked from 'package:base':

##

## intersect, setdiff, setequal, union

**library**(caret) *# for classification and regression training*

## Loading required package: lattice

## Loading required package: ggplot2

**library**(caTools) *# for splitting data into training and test set*

**library**(data.table) *# for converting data frame to table for faster execution*

##

## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':

##

## between, first, last

**library**(ggplot2) *# for basic plot*

**library**(corrplot) *# for plotting corelation plot between elements*

## corrplot 0.84 loaded

**library**(pROC) *# for plotting ROC curve*

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

##

## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':

##

## cov, smooth, var

**library**(rpart.plot) *# for plotting decision tree*

IMPORTING THE DATASET :

*# importing the dataset*

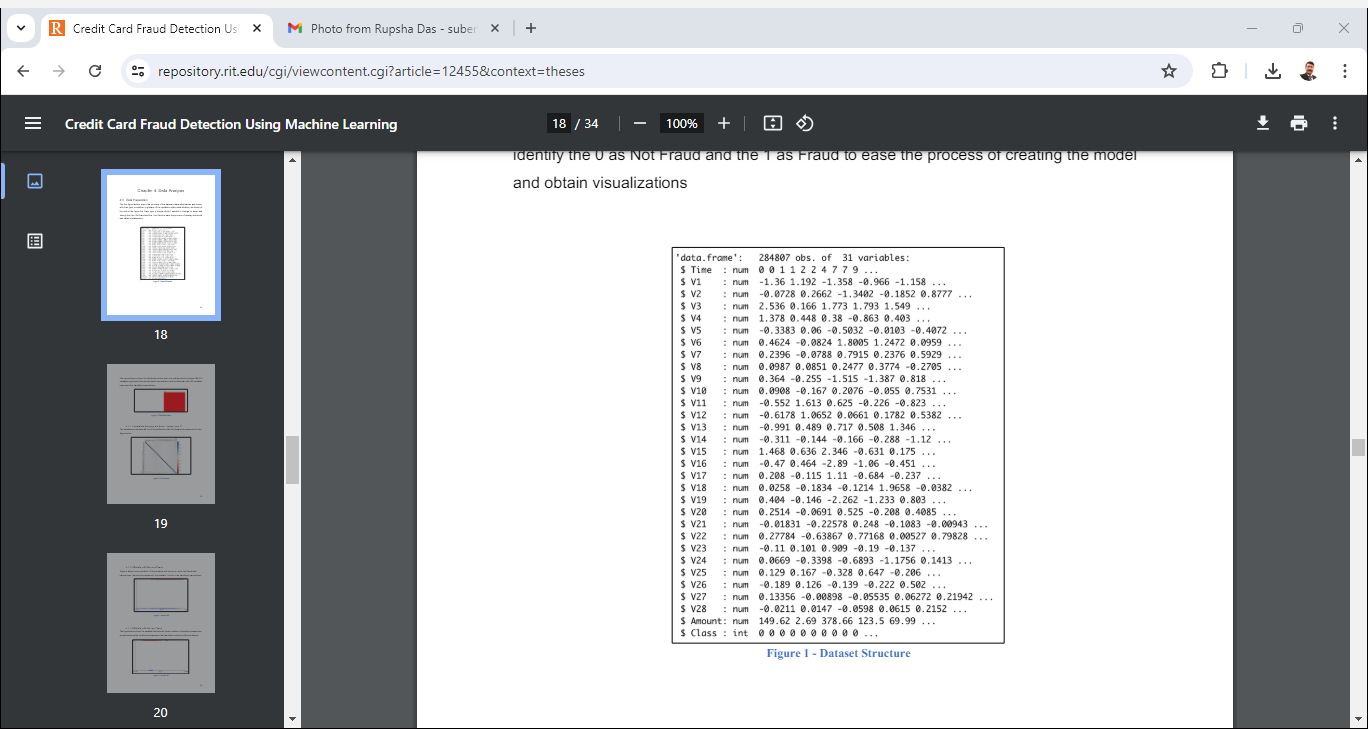
dataset <- (read.csv("data/creditcard.csv")

Data Preparation:

Data preparation is the process of gathering, cleansing, transforming and modelling data with the goal of making it ready for analysis as part of data visualization or business intelligence.

Data preparation is an important step in data analytics as well as in business intelligence. It's also a core function of business analysts.

In its most basic terms, preparation means making sure that data from multiple sources can be combined seamlessly into a single useful source. That usually requires unifying and normalizing data so that it can be used for further analysis and reporting.



Let’s explore the dataset and see if we can find anything that stands out and preprocess them for building our machine learning model. In this section of the fraud detection ML project, we will explore the data that is contained in the creditcard data dataframe. 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 dataframe –

*# exploring the credit card data*

head(dataset)

## Time V1 V2 V3 V4 V5 V6

## 1: 0 -1.3598071 -0.07278117 2.5363467 1.3781552 -0.33832077 0.46238778

## 2: 0 1.1918571 0.26615071 0.1664801 0.4481541 0.06001765 -0.08236081

## 3: 1 -1.3583541 -1.34016307 1.7732093 0.3797796 -0.50319813 1.80049938

## 4: 1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888 1.24720317

## 5: 2 -1.1582331 0.87773675 1.5487178 0.4030339 -0.40719338 0.09592146

## 6: 2 -0.4259659 0.96052304 1.1411093 -0.1682521 0.42098688 -0.02972755

## V7 V8 V9 V10 V11 V12

## 1: 0.23959855 0.09869790 0.3637870 0.09079417 -0.5515995 -0.61780086

## 2: -0.07880298 0.08510165 -0.2554251 -0.16697441 1.6127267 1.06523531

## 3: 0.79146096 0.24767579 -1.5146543 0.20764287 0.6245015 0.06608369

## 4: 0.23760894 0.37743587 -1.3870241 -0.05495192 -0.2264873 0.17822823

## 5: 0.59294075 -0.27053268 0.8177393 0.75307443 -0.8228429 0.53819555

## 6: 0.47620095 0.26031433 -0.5686714 -0.37140720 1.3412620 0.35989384

## V13 V14 V15 V16 V17 V18

## 1: -0.9913898 -0.3111694 1.4681770 -0.4704005 0.20797124 0.02579058

## 2: 0.4890950 -0.1437723 0.6355581 0.4639170 -0.11480466 -0.18336127

## 3: 0.7172927 -0.1659459 2.3458649 -2.8900832 1.10996938 -0.12135931

## 4: 0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279 1.96577500

## 5: 1.3458516 -1.1196698 0.1751211 -0.4514492 -0.23703324 -0.03819479

## 6: -0.3580907 -0.1371337 0.5176168 0.4017259 -0.05813282 0.06865315

## V19 V20 V21 V22 V23 V24

## 1: 0.40399296 0.25141210 -0.018306778 0.277837576 -0.11047391 0.06692807

## 2: -0.14578304 -0.06908314 -0.225775248 -0.638671953 0.10128802 -0.33984648

## 3: -2.26185710 0.52497973 0.247998153 0.771679402 0.90941226 -0.68928096

## 4: -1.23262197 -0.20803778 -0.108300452 0.005273597 -0.19032052 -1.17557533

## 5: 0.80348692 0.40854236 -0.009430697 0.798278495 -0.13745808 0.14126698

## 6: -0.03319379 0.08496767 -0.208253515 -0.559824796 -0.02639767 -0.37142658

## V25 V26 V27 V28 Amount Class

## 1: 0.1285394 -0.1891148 0.133558377 -0.02105305 149.62 0

## 2: 0.1671704 0.1258945 -0.008983099 0.01472417 2.69 0

## 3: -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66 0

## 4: 0.6473760 -0.2219288 0.062722849 0.06145763 123.50 0

## 5: -0.2060096 0.5022922 0.219422230 0.21515315 69.99 0

## 6: -0.2327938 0.1059148 0.253844225 0.08108026 3.67 0

tail(dataset)

## Time V1 V2 V3 V4 V5 V6

## 1: 172785 0.1203164 0.93100513 -0.5460121 -0.7450968 1.13031398 -0.2359732

## 2: 172786 -11.8811179 10.07178497 -9.8347835 -2.0666557 -5.36447278 -2.6068373

## 3: 172787 -0.7327887 -0.05508049 2.0350297 -0.7385886 0.86822940 1.0584153

## 4: 172788 1.9195650 -0.30125385 -3.2496398 -0.5578281 2.63051512 3.0312601

## 5: 172788 -0.2404400 0.53048251 0.7025102 0.6897992 -0.37796113 0.6237077

## 6: 172792 -0.5334125 -0.18973334 0.7033374 -0.5062712 -0.01254568 -0.6496167

## V7 V8 V9 V10 V11 V12

## 1: 0.8127221 0.1150929 -0.2040635 -0.6574221 0.6448373 0.19091623

## 2: -4.9182154 7.3053340 1.9144283 4.3561704 -1.5931053 2.71194079

## 3: 0.0243297 0.2948687 0.5848000 -0.9759261 -0.1501888 0.91580191

## 4: -0.2968265 0.7084172 0.4324540 -0.4847818 0.4116137 0.06311886

## 5: -0.6861800 0.6791455 0.3920867 -0.3991257 -1.9338488 -0.96288614

## 6: 1.5770063 -0.4146504 0.4861795 -0.9154266 -1.0404583 -0.03151305

## V13 V14 V15 V16 V17 V18

## 1: -0.5463289 -0.73170658 -0.80803553 0.5996281 0.07044075 0.3731103

## 2: -0.6892556 4.62694203 -0.92445871 1.1076406 1.99169111 0.5106323

## 3: 1.2147558 -0.67514296 1.16493091 -0.7117573 -0.02569286 -1.2211789

## 4: -0.1836987 -0.51060184 1.32928351 0.1407160 0.31350179 0.3956525

## 5: -1.0420817 0.44962444 1.96256312 -0.6085771 0.50992846 1.1139806

## 6: -0.1880929 -0.08431647 0.04133346 -0.3026201 -0.66037665 0.1674299

## V19 V20 V21 V22 V23 V24

## 1: 0.1289038 0.0006758329 -0.3142046 -0.8085204 0.05034266 0.102799590

## 2: -0.6829197 1.4758291347 0.2134541 0.1118637 1.01447990 -0.509348453

## 3: -1.5455561 0.0596158999 0.2142053 0.9243836 0.01246304 -1.016225669

## 4: -0.5772518 0.0013959703 0.2320450 0.5782290 -0.03750086 0.640133881

## 5: 2.8978488 0.1274335158 0.2652449 0.8000487 -0.16329794 0.123205244

## 6: -0.2561169 0.3829481049 0.2610573 0.6430784 0.37677701 0.008797379

## V25 V26 V27 V28 Amount Class

## 1: -0.4358701 0.1240789 0.217939865 0.06880333 2.69 0

## 2: 1.4368069 0.2500343 0.943651172 0.82373096 0.77 0

## 3: -0.6066240 -0.3952551 0.068472470 -0.05352739 24.79 0

## 4: 0.2657455 -0.0873706 0.004454772 -0.02656083 67.88 0

## 5: -0.5691589 0.5466685 0.108820735 0.10453282 10.00 0

## 6: -0.4736487 -0.8182671 -0.002415309 0.01364891 217.00 0

*# view the table from class column (0 for legit transactions and 1 for fraud)*

table(dataset$Class)

##

## 0 1

## 284315 492

*# view names of colums of dataset*

names(dataset)

## [1] "Time" "V1" "V2" "V3" "V4" "V5" "V6" "V7"

## [9] "V8" "V9" "V10" "V11" "V12" "V13" "V14" "V15"

## [17] "V16" "V17" "V18" "V19" "V20" "V21" "V22" "V23"

## [25] "V24" "V25" "V26" "V27" "V28" "Amount" "Class"

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

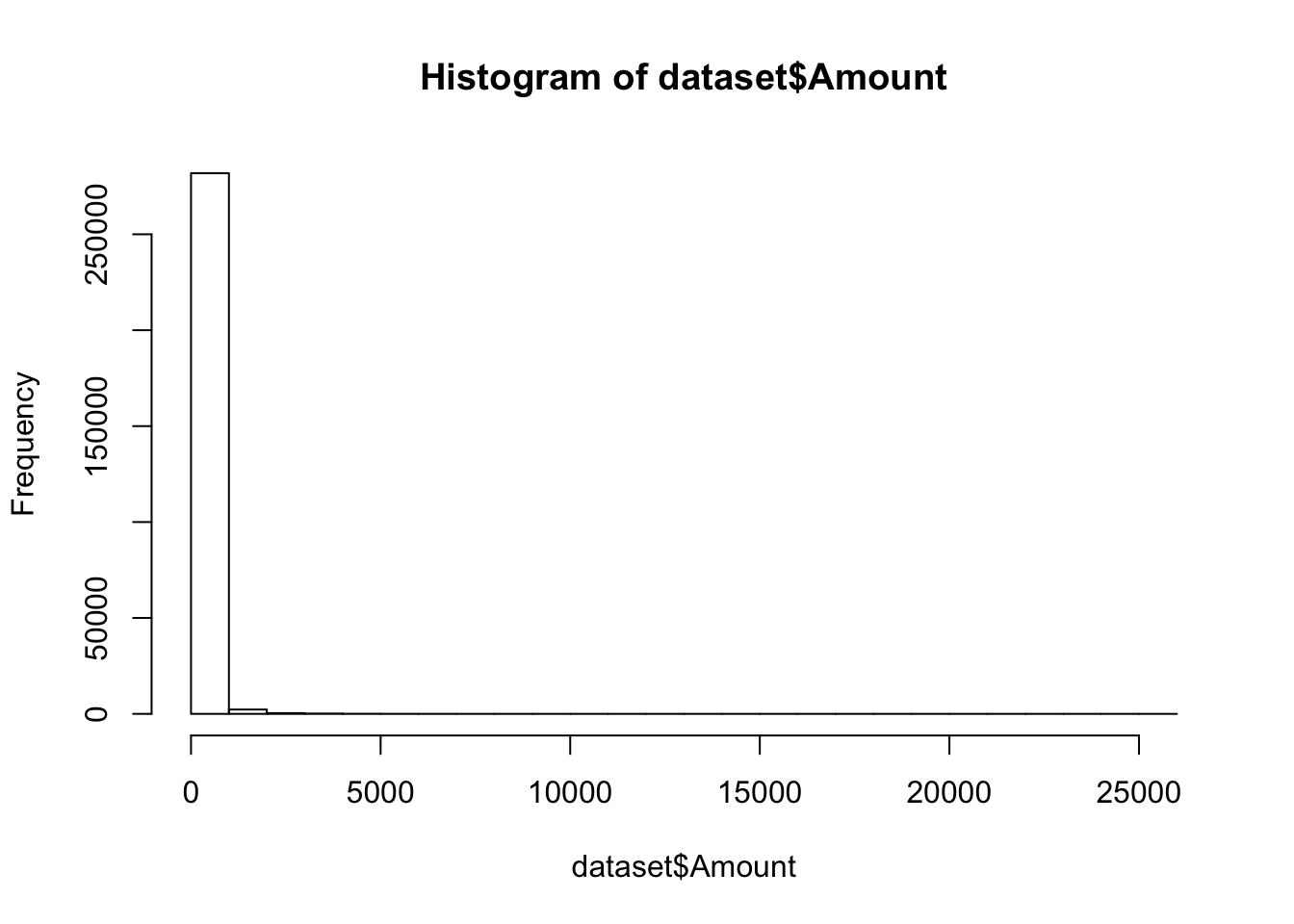
*# view summary of amount and histogram*

summary(dataset$Amount)

## Min. 1st Qu. Median Mean 3rd Qu. Max.

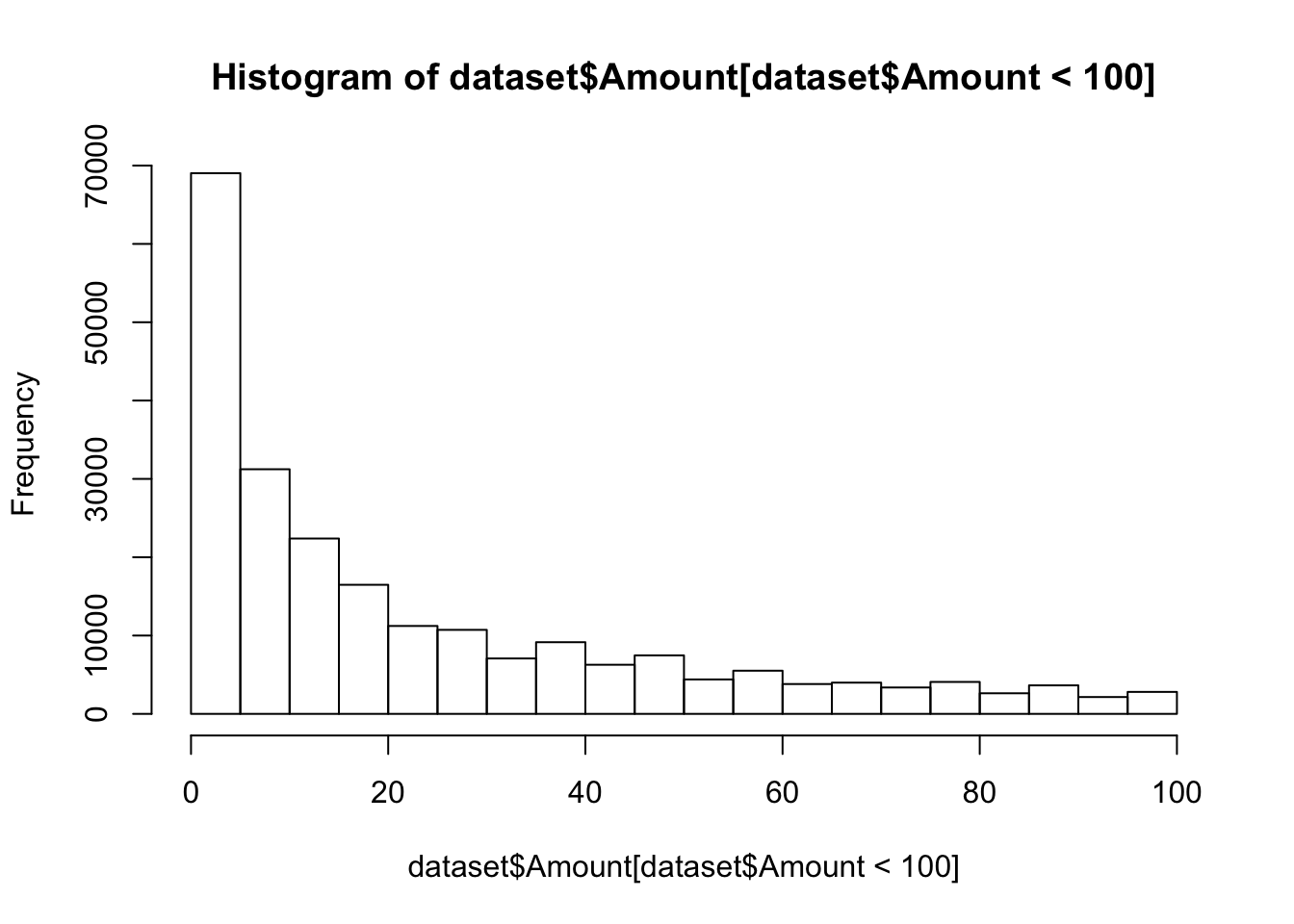
## 0.00 5.60 22.00 88.35 77.17 25691.16

hist(dataset$Amount)



Here we see that the dataset is having maximum frequency when the respective amount is low and as amount increses , frequency gradually decreases and becomes zero eventually.

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



Here a closer view of the above graph is shown where we only throw light upon the dataset having amount < 100

*>summary(df)*

Time V1 V2 V3

Min. : 0 Min. :-56.40751 Min. :-72.71573 Min. :-48.3256

1st Qu.: 54202 1st Qu.: -0.92037 1st Qu.: -0.59855 1st Qu.: -0.8904

Median : 84692 Median : 0.01811 Median : 0.06549 Median : 0.1799

Mean : 94814 Mean : 0.00000 Mean : 0.00000 Mean : 0.0000

3rd Qu.:139320 3rd Qu.: 1.31564 3rd Qu.: 0.80372 3rd Qu.: 1.0272

Max. :172792 Max. : 2.45493 Max. : 22.05773 Max. : 9.3826

V4 V5 V6 V7

Min. :-5.68317 Min. :-113.74331 Min. :-26.1605 Min. :-43.5572

1st Qu.:-0.84864 1st Qu.: -0.69160 1st Qu.: -0.7683 1st Qu.: -0.5541

Median :-0.01985 Median : -0.05434 Median : -0.2742 Median : 0.0401

Mean : 0.00000 Mean : 0.00000 Mean : 0.0000 Mean : 0.0000

3rd Qu.: 0.74334 3rd Qu.: 0.61193 3rd Qu.: 0.3986 3rd Qu.: 0.5704

Max. :16.87534 Max. : 34.80167 Max. : 73.3016 Max. :120.5895

V8 V9 V10 V11

Min. :-73.21672 Min. :-13.43407 Min. :-24.58826 Min. :-4.79747

1st Qu.: -0.20863 1st Qu.: -0.64310 1st Qu.: -0.53543 1st Qu.:-0.76249

V10 V11

Min. :-73.21672 Min. :-13.43407 Min. :-24.58826 Min. :-4.79747

1st Qu.: -0.20863 1st Qu.: -0.64310 1st Qu.: -0.53543 1st Qu.:-0.76249

Median : 0.02236 Median : -0.05143 Median : -0.09292 Median :-0.03276

Mean : 0.00000 Mean : 0.00000 Mean : 0.00000 Mean : 0.00000

3rd Qu.: 0.32735 3rd Qu.: 0.59714 3rd Qu.: 0.45392 3rd Qu.: 0.73959

Max. : 20.00721 Max. : 15.59500 Max. : 23.74514 Max. :12.01891

V12 V13 V14 V15

Min. :-18.6837 Min. :-5.79188 Min. :-19.2143 Min. :-4.49894

1st Qu.: -0.4056 1st Qu.:-0.64854 1st Qu.: -0.4256 1st Qu.:-0.58288

Median : 0.1400 Median :-0.01357 Median : 0.0506 Median : 0.04807

Mean : 0.0000 Mean : 0.00000 Mean : 0.0000 Mean : 0.00000

3rd Qu.: 0.6182 3rd Qu.: 0.66251 3rd Qu.: 0.4931 3rd Qu.: 0.64882

Max. : 7.8484 Max. : 7.12688 Max. : 10.5268 Max. : 8.87774

V16 V17 V18

Min. :-14.12985 Min. :-25.16280 Min. :-9.498746

1st Qu.: -0.46804 1st Qu.: -0.48375 1st Qu.:-0.498850

Median : 0.06641 Median : -0.06568 Median :-0.003636

Mean : 0.00000 Mean : 0.00000 Mean : 0.000000

Median : 0.06641 Median : -0.06568 Median :-0.003636

Mean : 0.00000 Mean : 0.00000 Mean : 0.000000

3rd Qu.: 0.52330 3rd Qu.: 0.39968 3rd Qu.: 0.500807

Max. : 17.31511 Max. : 9.25353 Max. : 5.041069

V19 V20 V21

Min. :-7.213527 Min. :-54.49772 Min. :-34.83038

1st Qu.:-0.456299 1st Qu.: -0.21172 1st Qu.: -0.22839

Median : 0.003735 Median : -0.06248 Median : -0.02945

Mean : 0.000000 Mean : 0.00000 Mean : 0.00000

3rd Qu.: 0.458949 3rd Qu.: 0.13304 3rd Qu.: 0.18638

Max. : 5.591971 Max. : 39.42090 Max. : 27.20284

V22 V23 V24

Min. :-10.933144 Min. :-44.80774 Min. :-2.83663

1st Qu.: -0.542350 1st Qu.: -0.16185 1st Qu.:-0.35459

Median : 0.006782 Median : -0.01119 Median : 0.04098

Mean : 0.000000 Mean : 0.00000 Mean : 0.00000

3rd Qu.: 0.528554 3rd Qu.: 0.14764 3rd Qu.: 0.43953

Max. : 10.503090 Max. : 22.52841 Max. : 4.58455

V25 V26 V27

Min. :-10.29540 Min. :-2.60455 Min. :-22.565679

1st Qu.: -0.31715 1st Qu.:-0.32698 1st Qu.: -0.070840

Median : 0.01659 Median :-0.05214 Median : 0.001342

Mean : 0.00000 Mean : 0.00000 Mean : 0.000000

3rd Qu.: 0.35072 3rd Qu.: 0.24095 3rd Qu.: 0.091045

Max. : 7.51959 Max. : 3.51735 Max. : 31.612198

V28 Amount Class

Min. :-15.43008 Min. : 0.00 Min. :0.000000

1st Qu.: -0.05296 1st Qu.: 5.60 1st Qu.:0.000000

Median : 0.01124 Median : 22.00 Median :0.000000

Mean : 0.00000 Mean : 88.35 Mean :0.001728

3rd Qu.: 0.07828 3rd Qu.: 77.17 3rd Qu.:0.000000

Max. : 33.84781 Max. :25691.16 Max. :1.000000

Now we try to check the missing values

# checking missing values

*>colSums(is.na(df))*

0

**V19**

0

**V20**

0

**V21**

0

**V22**

0

**V23**

0

**V24**

0

**V25**

0

**V26**

0

**V27**

0

**V28**

0

**Amount**

0

**Class**

0

**Time**

0

**V1**

0

**V2**

0

**V3**

0

**V4**

0

**V5**

0

**V6**

0

**V7**

0

**V8**

0

**V9**

0

**V10**

0

**V11**

0

**V12**

0

**V13**

0

**V14**

0

**V15**

0

**V16**

0

**V17**

0

**V18**

0

**V19**

0

**V20**

0

**V21**

0

**V22**

0

**V23**

0

**V24**

0

**V25**

0

**V26**

0

**V27**

0

**V28**

0

**Amount**

0

**Class**

0

So we see that none of the variables have missing values

*# checking class imbalance*

*>table(df$Class)*

0 1

284315 492

*# class imbalance in percentage*

*>prop.table(table(df$Class))*

0 1

0.998272514 0.001727486

*# view variance and standard deviation of amount column*

var(dataset$Amount)

## [1] 62560.07

sd(dataset$Amount)

## [1] 250.1201

*# check whether there are any missing values in colums*

colSums(is.na(dataset))

## Time V1 V2 V3 V4 V5 V6 V7 V8 V9 V10

## 0 0 0 0 0 0 0 0 0 0 0

## V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21

## 0 0 0 0 0 0 0 0 0 0 0

## V22 V23 V24 V25 V26 V27 V28 Amount Class

## 0 0 0 0 0 0 0 0 0

Let’s first visualize the transactions over time and see if time is an important factor to be considered for this classification.

*# visualizing the distribution of transcations across time*

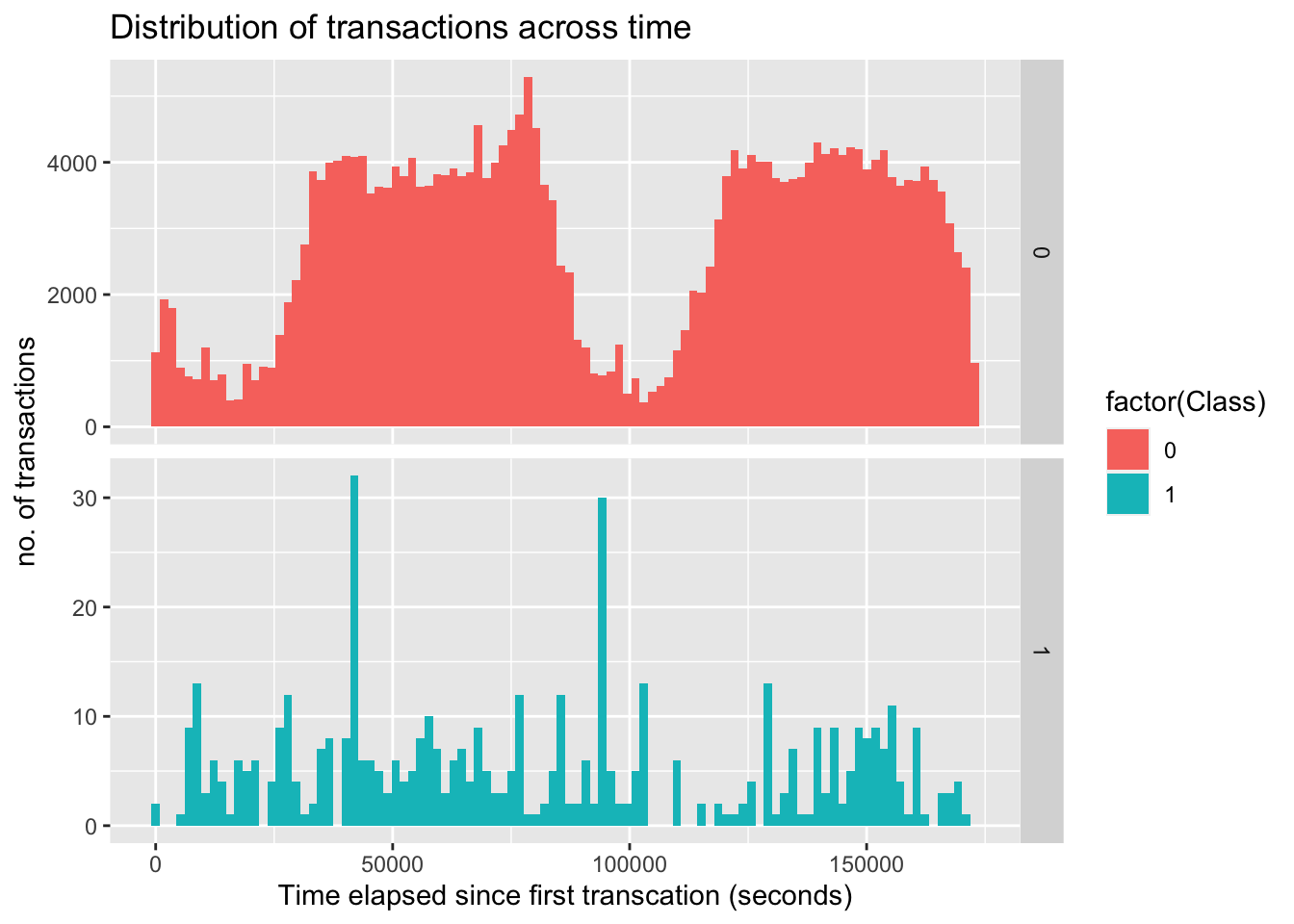
dataset %>%

ggplot(aes(x = Time, fill = factor(Class))) +

geom\_histogram(bins = 100) +

labs(x = "Time elapsed since first transcation (seconds)", y = "no. of transactions", title = "Distribution of transactions across time") +

facet\_grid(Class ~ ., scales = 'free\_y') + theme()

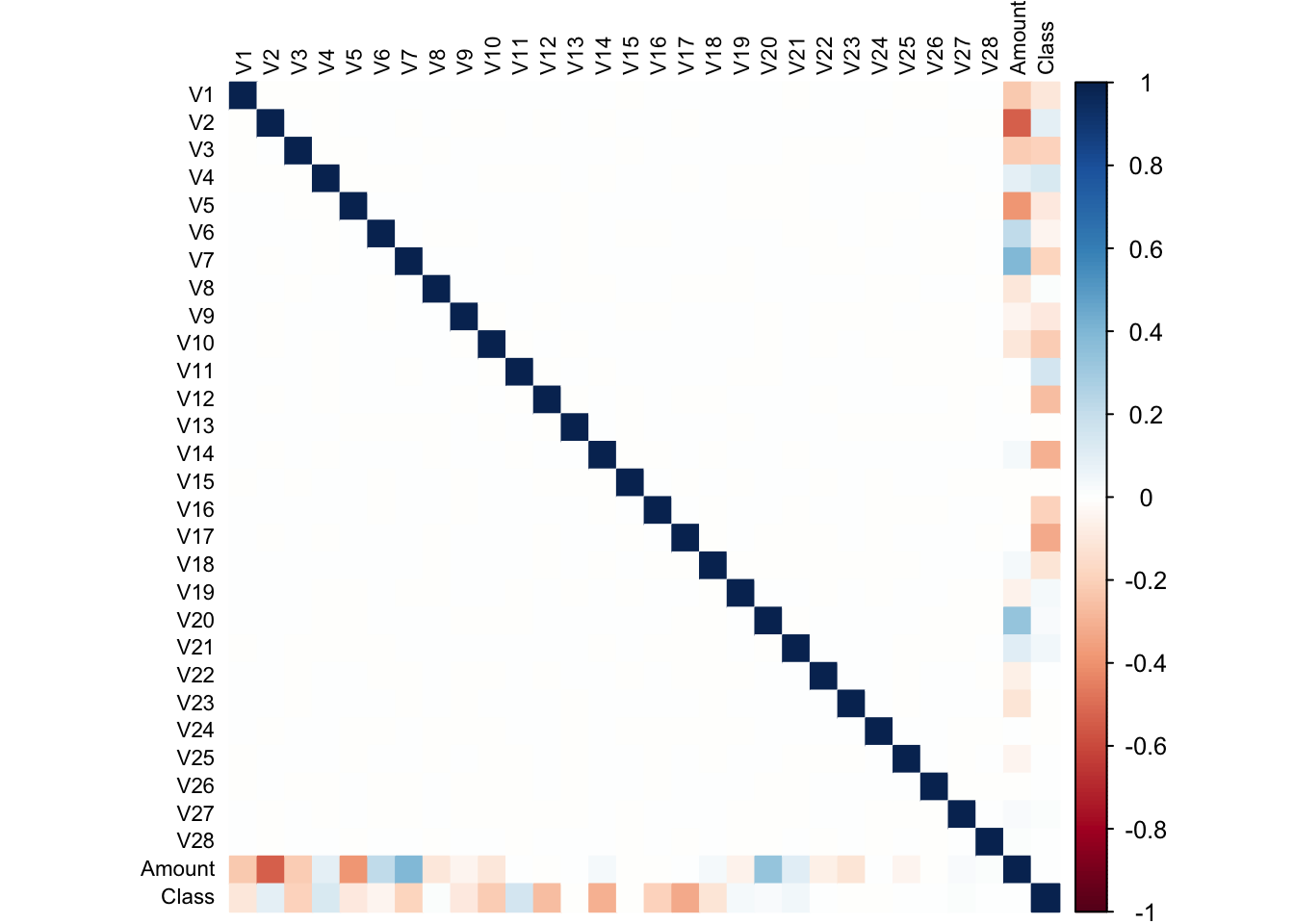
The time vs amount histogram looks pretty similar in both transactions. Since time doesn’t contribute much in fraud detection we can remove the time column from the data.

Next we check the corelation between all the variables and amount and class and see if there are any variables that correlate with each other.

*# correlation of anonymous variables with amount and class*

correlation <- cor(dataset[, -1], method = "pearson")

corrplot(correlation, number.cex = 1, method = "color", type = "full", tl.cex=0.7, tl.col="black")



From the above graph, we can see that most of the features are not correlated. In fact, all the anonymous variables are independent to each other.

### Data Manipulation :

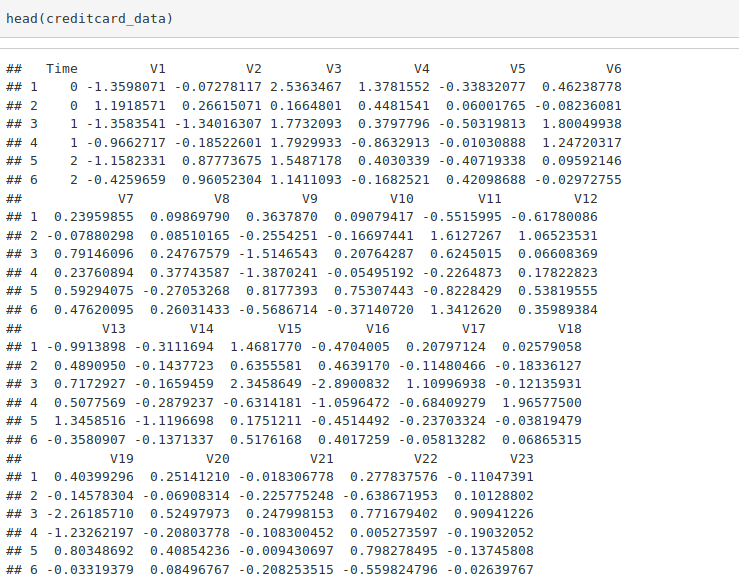
### Data manipulation is the process of organizing or arranging data in order to make it easier to interpret. Data manipulation typically requires the use of a type of [database language](https://www.indeed.com/career-advice/career-development/database-languages) called data manipulation language (DML). DML is a type of coding language that allows you to reorganize data by modifying it within its database program

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. We will carry this out as follows:

**Code:**

*>head(creditcard\_data)*

**Output Screenshot:**

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/Data-Manipulation-1.png)

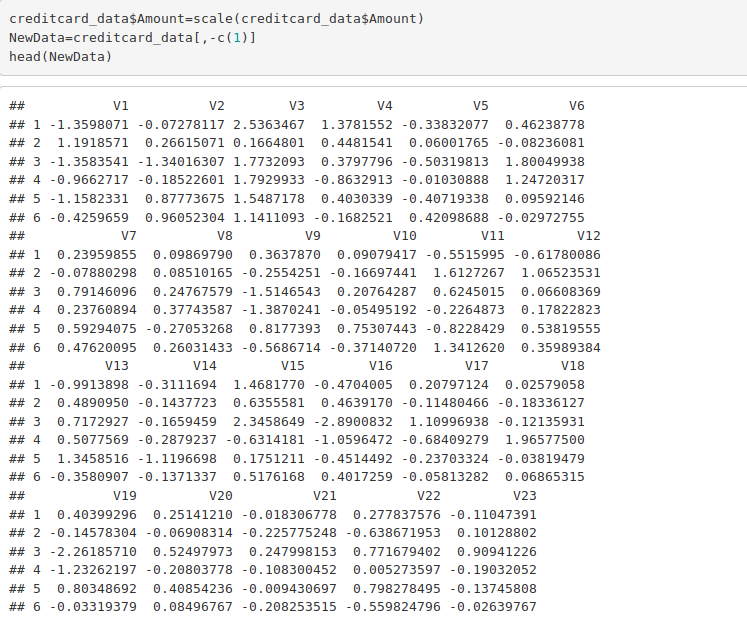
**Code:**

**>***creditcard\_data$Amount=scale(creditcard\_data$Amount)*

*NewData=creditcard\_data[,-c(1)]*

*>head(NewData)*

**Output Screenshot:**

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/Data-Manipulation-2.png)

# Hereat first we divide the dataset into two parts in a 8:2 ratio , then we fit logistic regression on training dataset and then predict the probabilities for the test dataset. After that by the method of comparison we check the model accuracy.

library(caTools)

set.seed(101)

sample<- sample(c(TRUE,FALSE),nrow(data), replace=TRUE, prob=c(0.8,0.2))

train\_data <- data[sample, ]

test\_data <- data[sample, ]

dim(train\_data)

# # [1] 454576 31

dim(test\_data)

# # [1] 454576 31

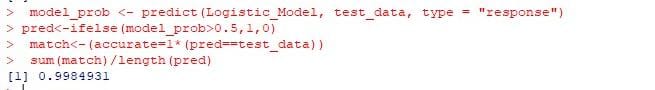
>  model\_prob <- predict(Logistic\_Model, test\_data, type = "response")

> pred<-ifelse(model\_prob>0.5,1,0)

>  match<-(accurate=1\*(pred==test\_data))

>  sum(match)/length(pred)

[1] 0.9984931



Now we visualize the training dataset

*# visualize the training data*

train\_data %>% ggplot(aes(x = factor(Class), y = prop.table(stat(count)), fill = factor(Class))) +

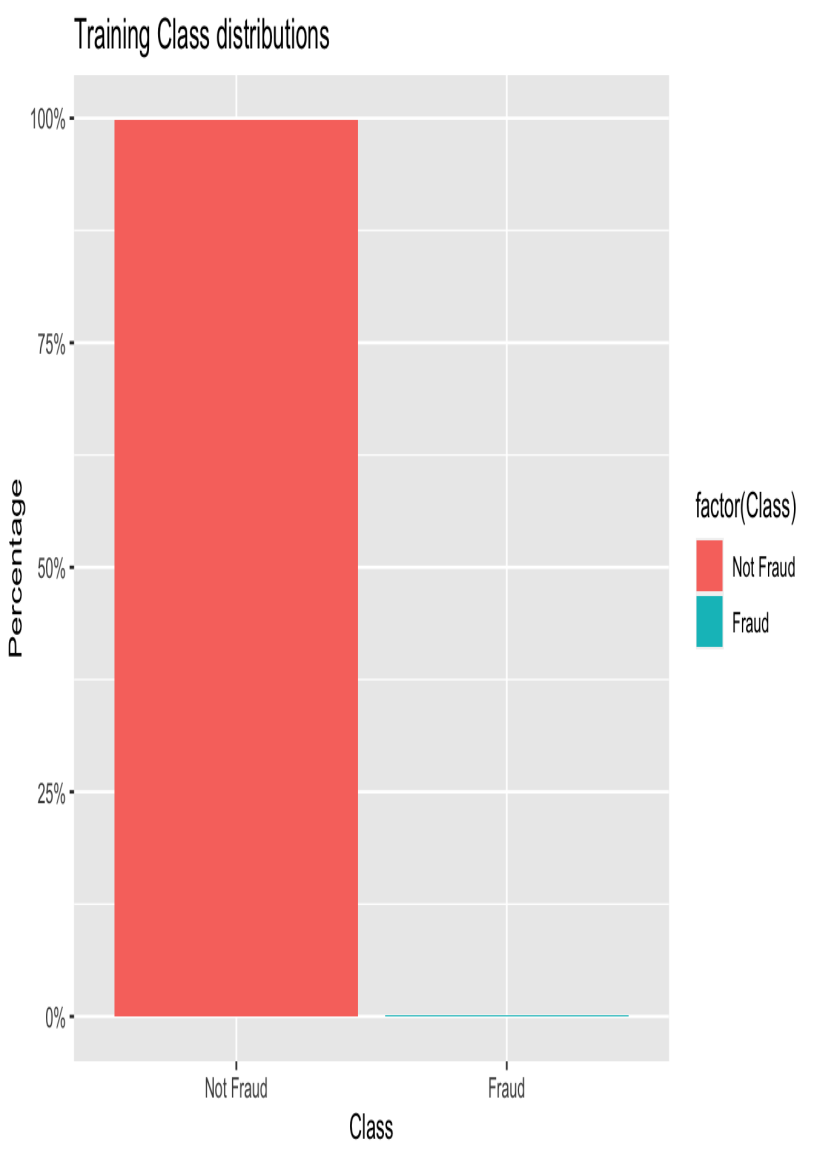
geom\_bar(position = "dodge") +

scale\_y\_continuous(labels = scales::percent) +

labs(x = 'Class', y = 'Percentage', title = 'Training Class distributions') +

theme\_grey()

Since the data is heavily unbalanced with 99% of non-fraudulent data, this may result in our model perfoming less accurately and being heavily biased towards non-fraudulent transactions.



In this section of credit card fraud detection project, we will fit our first model.

**Code:**

>Logistic\_Model=glm(**Class**~.,test\_data,family=binomial())

>summary(Logistic\_Model)

**Output Screenshot:**

Logistic\_Model=glm(Class~.,train\_data,family=binomial())

summary(Logistic\_Model)

# # Call:

# # glm(formula = Class ~ ., family = binomial(), data = train\_data)

# # Coefficients:

             # # Estimate Std. Error z value Pr(>|z|)

# # # # (Intercept) -1.829e+01 3.180e-01 -57.529 < 2e-16 \*\*\*

# # id 9.859e-05 1.545e-06 63.822 < 2e-16 \*\*\*

# # V1 -7.409e-01 5.894e-02 -12.571 < 2e-16 \*\*\*

# # V2 -5.970e-02 5.789e-02 -1.031 0.302350

# # V3 1.050e-01 7.405e-02 1.418 0.156221

# # V4 3.702e+00 1.029e-01 35.958 < 2e-16 \*\*\*

# # V5 -4.619e-01 6.539e-02 -7.064 1.62e-12 \*\*\*

# # V6 -6.676e-01 7.802e-02 -8.557 < 2e-16 \*\*\*

# # V7 -1.216e+00 9.165e-02 -13.272 < 2e-16 \*\*\*

# # V8 -2.482e+00 1.303e-01 -19.045 < 2e-16 \*\*\*

# # V9 1.012e+00 1.026e-01 9.863 < 2e-16 \*\*\*

# # V10 -2.398e+00 1.251e-01 -19.163 < 2e-16 \*\*\*

# # V11 3.015e+00 8.366e-02 36.032 < 2e-16 \*\*\*

# # V12 -4.266e+00 1.391e-01 -30.665 < 2e-16 \*\*\*

# # V13 8.796e-01 5.553e-02 15.841 < 2e-16 \*\*\*

# # V14 -1.376e+00 1.098e-01 -12.539 < 2e-16 \*\*\*

# # V15 -2.837e-01 4.123e-02 -6.880 5.99e-12 \*\*\*

# # V16 -8.626e-01 9.140e-02 -9.438 < 2e-16 \*\*\*

# # V17 -1.075e+00 1.010e-01 -10.645 < 2e-16 \*\*\*

# # V14 -1.376e+00 1.098e-01 -12.539 < 2e-16 \*\*\*

# # V15 -2.837e-01 4.123e-02 -6.880 5.99e-12 \*\*\*

# # V16 -8.626e-01 9.140e-02 -9.438 < 2e-16 \*\*\*

# # V17 -1.075e+00 1.010e-01 -10.645 < 2e-16 \*\*\*

# # V18 -1.176e+00 8.850e-02 -13.293 < 2e-16 \*\*\*

# # V19 -2.129e-01 5.160e-02 -4.127 3.67e-05 \*\*\*

# # V20 1.562e-01 4.501e-02 3.471 0.000518 \*\*\*

# # V21 2.487e-01 1.263e-01 1.969 0.048972 \*

# # V22 1.548e-01 5.915e-02 2.618 0.008852 \*\*

# # V23 -2.730e-01 3.806e-02 -7.174 7.30e-13 \*\*\*

# # V24 -7.268e-02 3.757e-02 -1.934 0.053055 .

# # V25 3.530e-01 3.930e-02 8.983 < 2e-16 \*\*\*

# # V26 -8.133e-02 3.597e-02 -2.261 0.023741 \*

# # V27 8.648e-02 8.595e-02 1.006 0.314328

# # V28 1.147e-01 3.308e-02 3.468 0.000525 \*\*\*

# # Amount -1.019e-05 4.948e-06 -2.060 0.039424 \*

---

# # Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

# # (Dispersion parameter for binomial family taken to be 1)

    # # Null deviance: 630176.1 on 454575 degrees of freedom

# # Residual deviance: 6592.3 on 454545 degrees of freedom

# # AIC: 6654.3

# # Number of Fisher Scoring iterations: 13

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

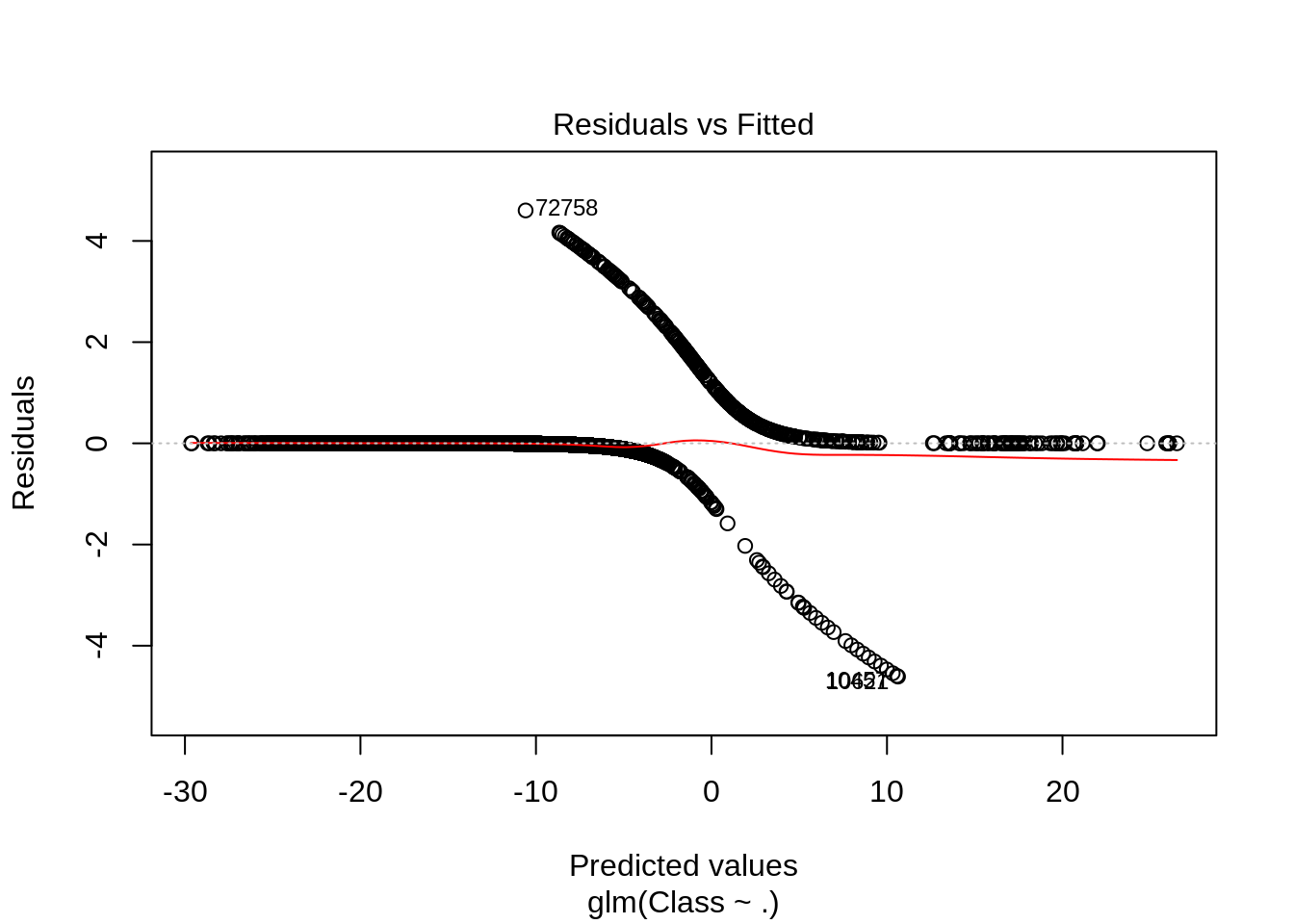
**Code:**

>plot(Logistic\_Model)

**Input Screenshot:**

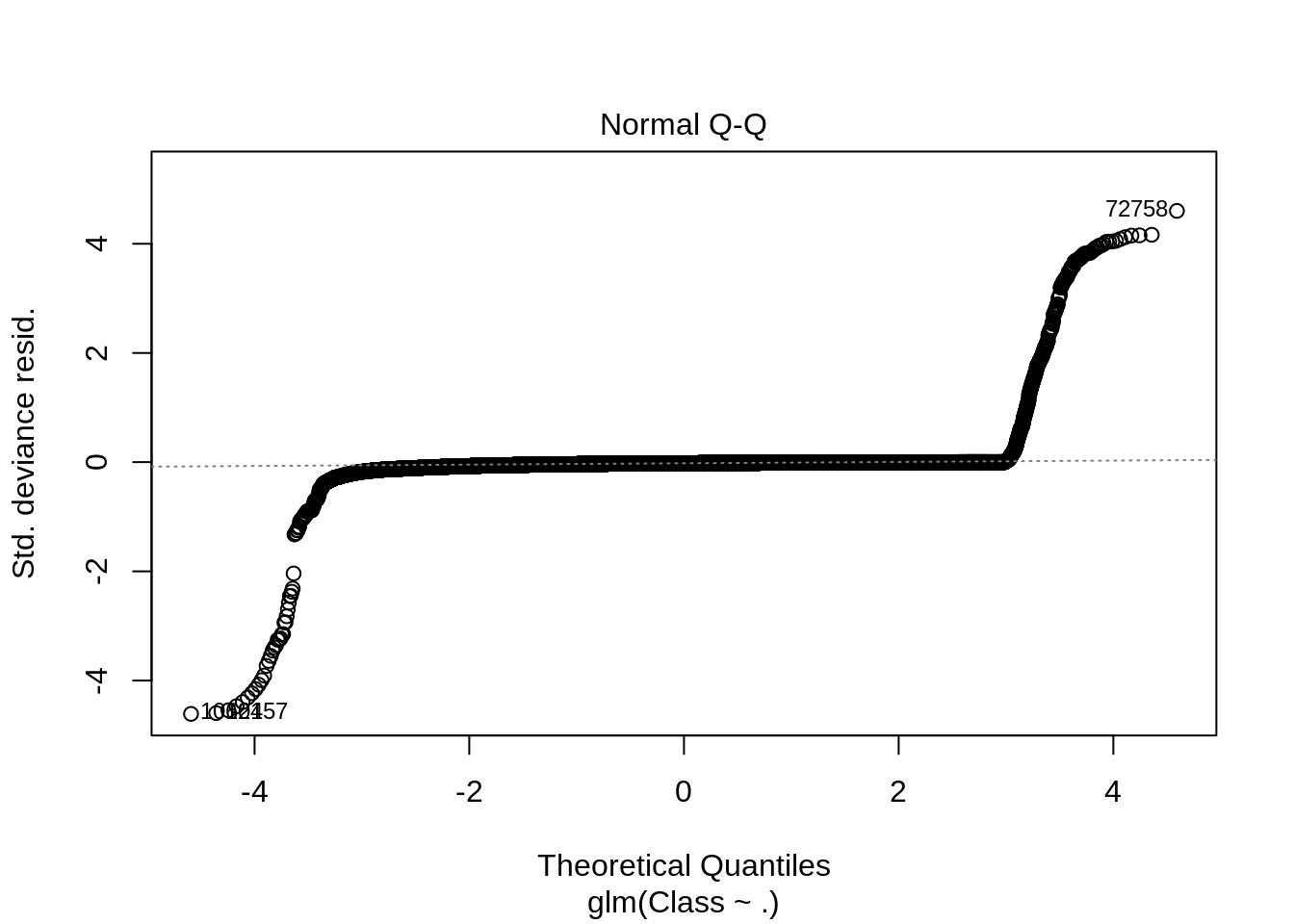
[Data Modeling ](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/Data-Modeling-2.png)

**Output:**

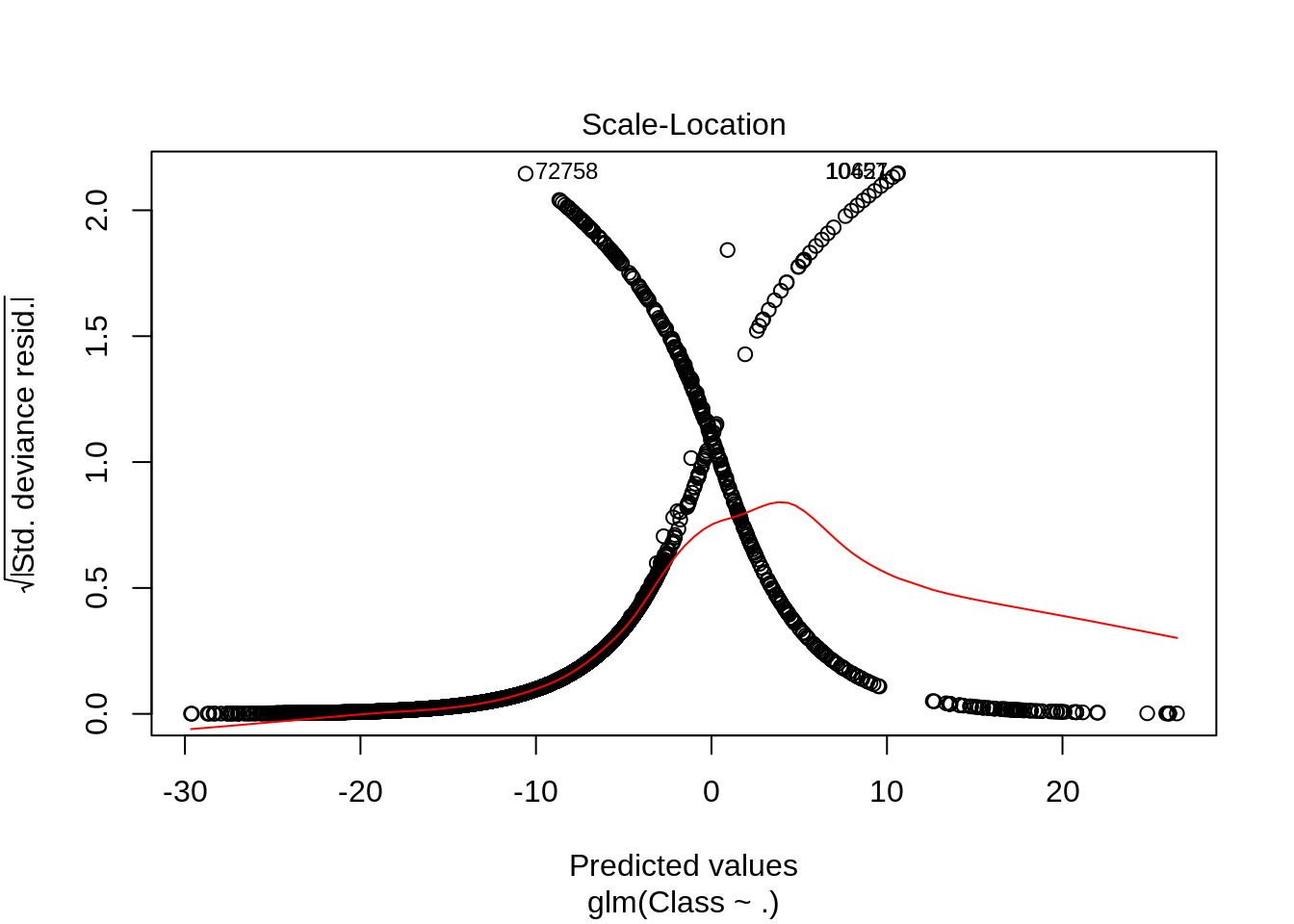
[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/logistic-regression-model-output-1-1.png)

From the above graph, we can observe that the Homoscedasticity is high.

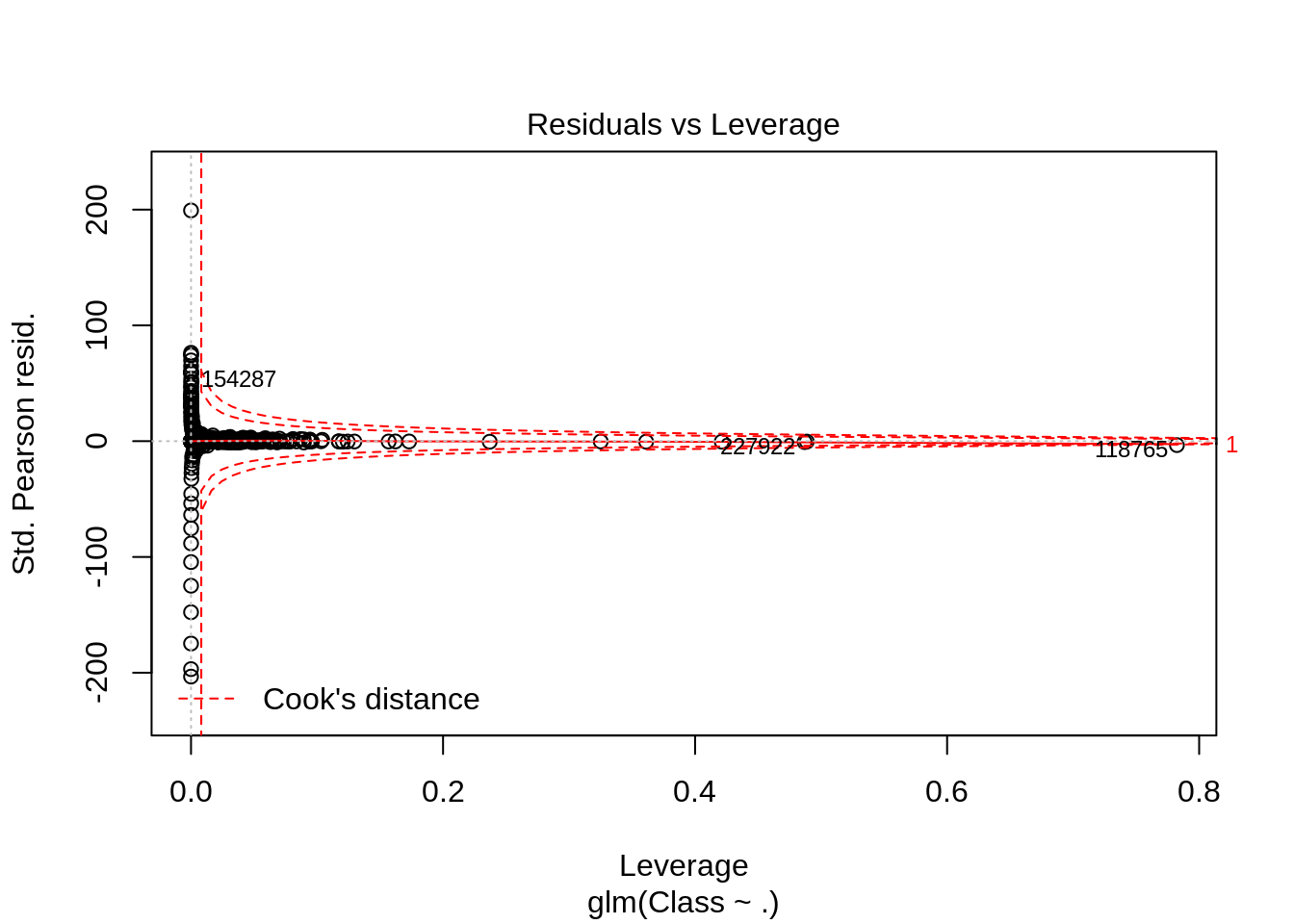
**Output:**

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/logistic-output-3.png)

**Output:**

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/logistic-regression-output-3.png)

**Output:**

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/logistic-output-4.png)

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](https://en.wikipedia.org/wiki/Receiver_operating_characteristic) curve to analyze its performance.

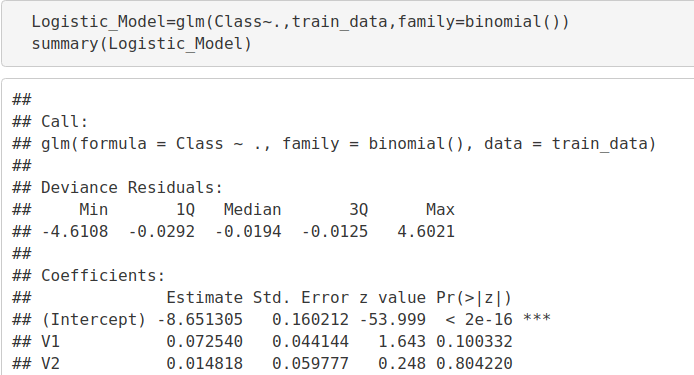
**Code:**

*>library(pROC)*

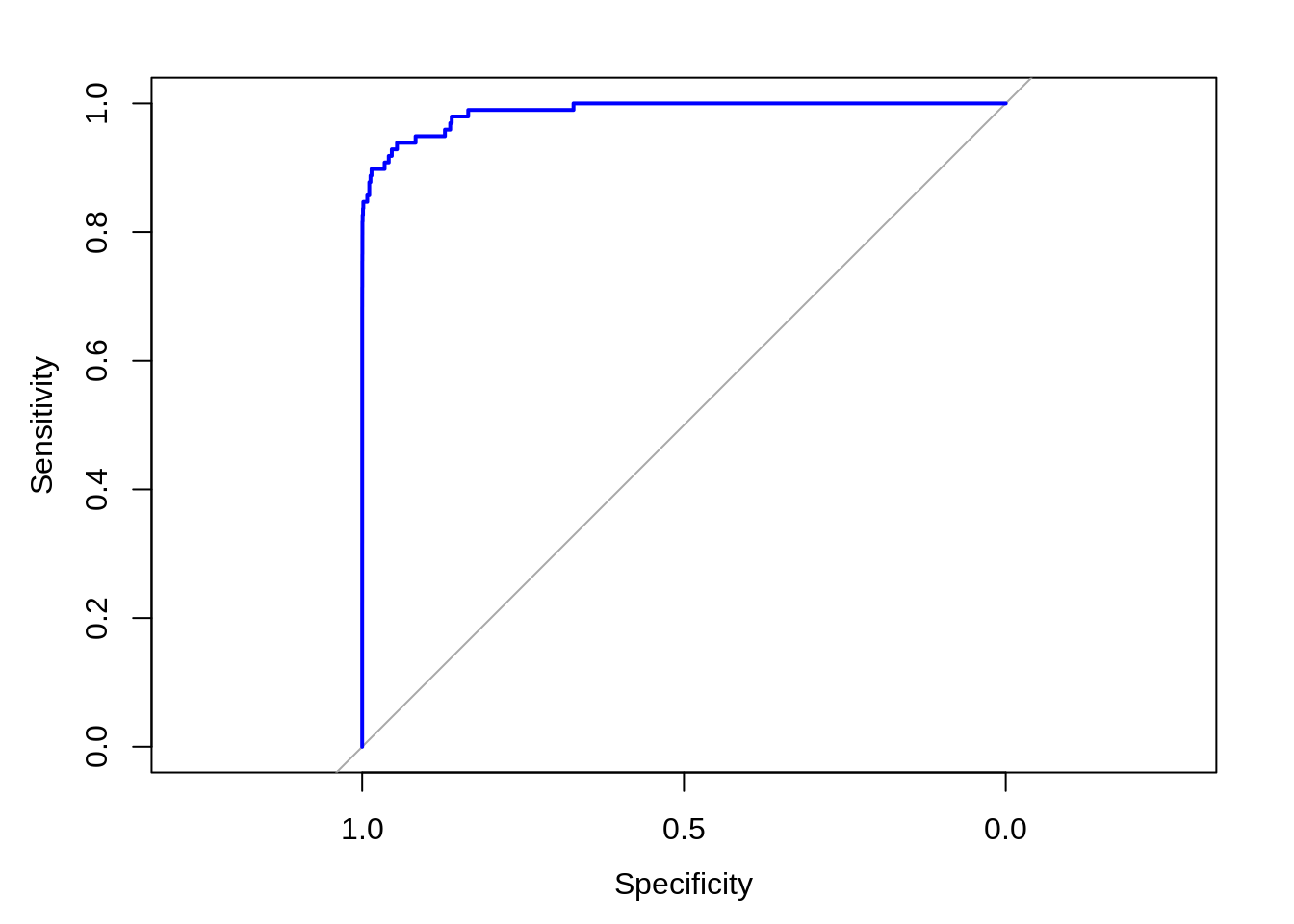
*>lr.predict <- predict(Logistic\_Model,train\_data, probability =* ***TRUE****)*

*>auc.gbm = roc(test\_data$****Class****, lr.predict, plot =* ***TRUE****, col = "blue")*

**Output Screenshot:**

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/logistic-ROC.png)

**Output:**

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/logistic_ROC.png)

From the above graph, we can see that the sensitivity is high that means the true positivity rate is high.

### Fitting a Decision Tree Model

In this section, we will implement a decision tree algorithm. [*Decision Trees*](https://data-flair.training/blogs/r-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. We will specifically use the recursive parting to plot the decision tree.

**Code:**

*>library(rpart)*

*>library(rpart.plot)*

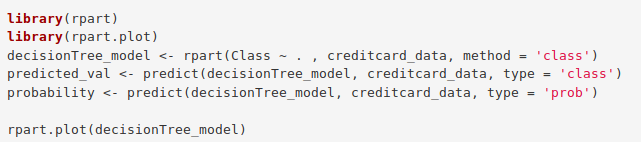
*>decisionTree\_model <- rpart(****Class*** *~ . , creditcard\_data, method = 'class')*

*>predicted\_val <- predict(decisionTree\_model, creditcard\_data, type = 'class')*

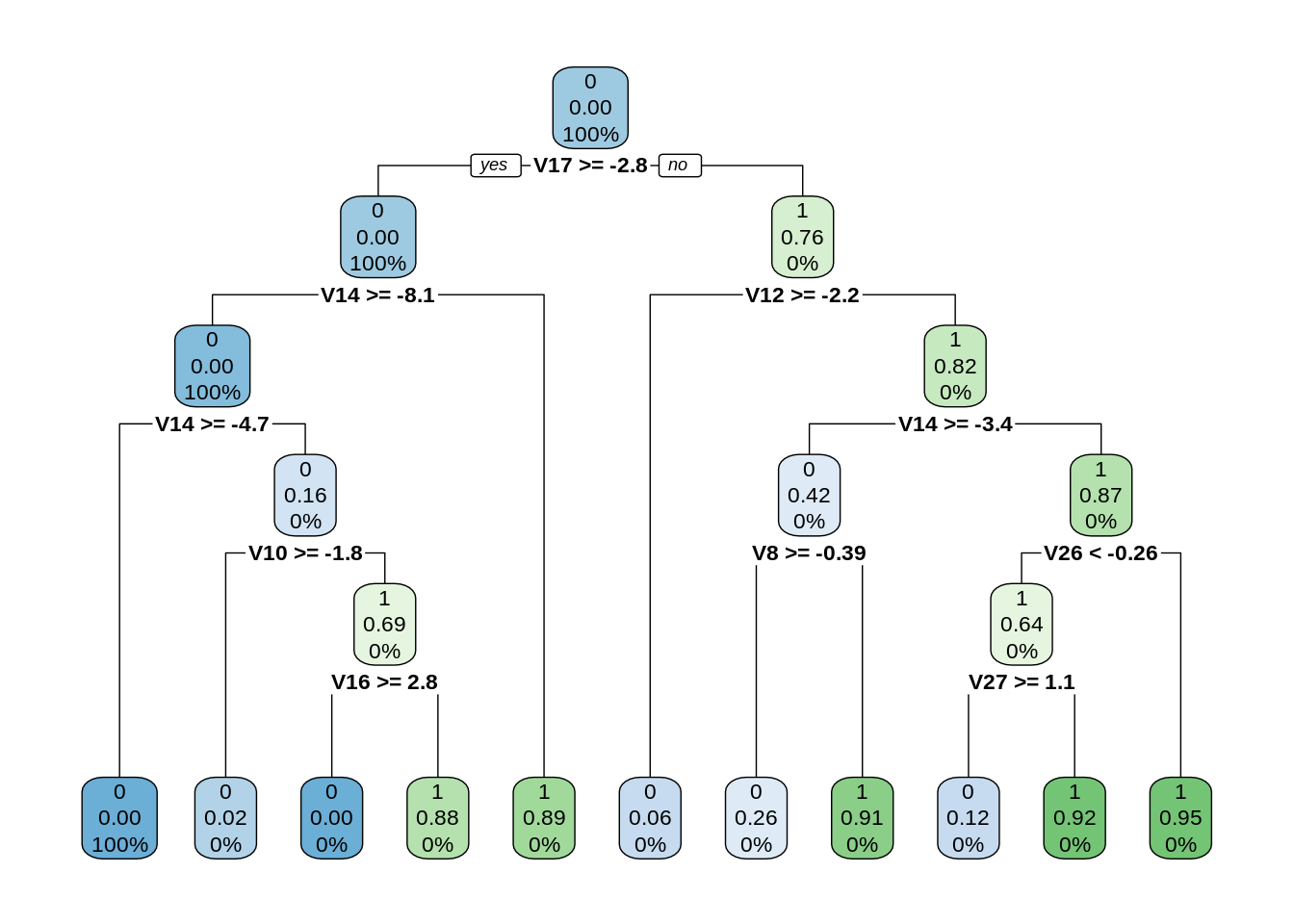
*>probability <- predict(decisionTree\_model, creditcard\_data, type = 'prob')*

*>rpart.plot(decisionTree\_model)*

**Input Screenshot:**

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/DecisionTree-ML-Code.png)

**Output:**

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/Decision-Tree-machine-learning-Plot.png)

In this particular project , we investigated the data , checked for any missing values, then we scale the data to structure it according to a specific range.

Then we split the data into two parts test data and train data . For fitting with Logistic Regression Model we only use train data . then we predict on the test dataset and find the accuracy of the model.

After fitting logistic regression , the accuracy obtained is 0.998

While we couldn’t reach out goal of 100% accuracy in fraud detection, we did end up creating a system that can, with enough time and data, get very close to that goal.