ANA 515 Assignment 4

Christopher Spann

12/11/2021

Business Problem / Goal

According to the Federal Bureau of Investigation, credit card fraud is “the unauthorized use of a credit or debit card, or similar payment tool (ACH, EFT, recurring charge, etc.), to fraudulently obtain money or property”; moreover, credit card holders often seek security in knowing that their financial institution implements practices for detecting fraudulent purchases and dealing with them accordingly. The goal of this project is to create a model for identifying fraudulent credit card transactions given a list of numerical variables. It is important that financial institutions are able to classify a purchase as fraudulent or legitimate so that consumers are not charged for items that they did not purchase.

Dataset

-The data used for this project contains transactions made by credit cards in September of 2013 by European cardholders.

-This dataset presents transactions that occurred in two days, and overall there are 492 frauds out of 284,807 total transactions.

-The dataset was retrieved from Kaggle, but originally the dataset was collected for a research project by the machine learning group of ULB (Université Libre de Bruxelles).

https://www.kaggle.com/mlg-ulb/creditcardfraud (https://www.kaggle.com/mlg-ulb/creditcardfraud)

Importing Dataset

I first downloaded the csv file from the Kaggle site and then moved it to my working directory folder of R Studio. Then, I imported the file into R Studio to begin analysis. Below is the code used to import and save the dataset in R.

```
#First, we need to call the appropriate library for reading the data into R. Then we will use th
e read.csv function to load the dataset from our working directory into R.
library(readr)
cc_fraud <- read.csv(file = "creditcard.csv")
```

Description of Data

```
#We can get a good description of the data using a few functions in R. First, we will describe a
summary of the data including the number of rows using nrow, the number of columns using ncol, a
nd then a brief summary of each variable using the summary function. The output of the summary f
unction includes the min, 1Q, median, mean 3Q, and max for each variable in the dataset.
```

-Number of rows: 284807

-Number of columns: 31

```
summary(cc_fraud)
```

```
##       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.:139321   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  
##  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  
##  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
```

This table below lists all variables in the dataset as well as the variable class, variable type, variable mean, variable standard deviation, variable min, variable max, and number of missing values by variable.

```
#This next code chunk is used to create a summary table of the variable of the dataset.
#We will create vectors of values in order to create this table.
```

```
#The first column will be the variable names of all variables in the dataset. The ls function ca
n be used to list the objects of our dataframe, cc_fraud. Then, the c function can be used to co
mbine all values into a vector.
column_names <- c(ls(cc_fraud))

#Similar to column name, we will be creating a vector of the variable class. The class function
 prints the vector of names of classes an object inherits from.
variable_class <- c(class(cc_fraud$Amount), class(cc_fraud$Class), class(cc_fraud$Time), class(c
c_fraud$V1), class(cc_fraud$V10), class(cc_fraud$V11), class(cc_fraud$V12), class(cc_fraud$V13),
class(cc_fraud$V14), class(cc_fraud$V15), class(cc_fraud$V16), class(cc_fraud$V17), class(cc_fra
ud$V18), class(cc_fraud$V19), class(cc_fraud$V2), class(cc_fraud$V20), class(cc_fraud$V21), clas
s(cc_fraud$V22), class(cc_fraud$V23), class(cc_fraud$V24), class(cc_fraud$V25), class(cc_fraud$V
26), class(cc_fraud$V27), class(cc_fraud$V28), class(cc_fraud$V3), class(cc_fraud$V4), class(cc_
fraud$V5), class(cc_fraud$V6), class(cc_fraud$V7), class(cc_fraud$V8), class(cc_fraud$V9))

#Next, we want to see the variable type of each variable in the dataset. We will be using the ty
peof function which determines the (R internal) type or storage mode of any object.
variable_type <- c(typeof(cc_fraud$Amount), typeof(cc_fraud$Class), typeof(cc_fraud$Time), typeo
f(cc_fraud$V1), typeof(cc_fraud$V10), typeof(cc_fraud$V11), typeof(cc_fraud$V12), typeof(cc_frau
d$V13), typeof(cc_fraud$V14), typeof(cc_fraud$V15), typeof(cc_fraud$V16), typeof(cc_fraud$V17),
 typeof(cc_fraud$V18), typeof(cc_fraud$V19), typeof(cc_fraud$V2), typeof(cc_fraud$V20), typeof(c
c_fraud$V21), typeof(cc_fraud$V22), typeof(cc_fraud$V23), typeof(cc_fraud$V24), typeof(cc_fraud
$V25), typeof(cc_fraud$V26), typeof(cc_fraud$V27), typeof(cc_fraud$V28), typeof(cc_fraud$V3), ty
peof(cc_fraud$V4), typeof(cc_fraud$V5), typeof(cc_fraud$V6), typeof(cc_fraud$V7), typeof(cc_frau
d$V8), typeof(cc_fraud$V9))

#To get variable mean, we will use the mean function. We will nest the mean function within roun
d to shorten our output to only 4 digits after the decimal.
variable_mean <- c(round(mean(cc_fraud$Amount), digits=4), round(mean(cc_fraud$Class), digits=4
), round(mean(cc_fraud$Time), digits=4), round(mean(cc_fraud$V1), digits=4), round(mean(cc_fraud
$V10), digits=4), round(mean(cc_fraud$V11), digits=4), round(mean(cc_fraud$V12), digits=4), roun
d(mean(cc_fraud$V13), digits=4), round(mean(cc_fraud$V14), digits=4), round(mean(cc_fraud$V15),
 digits=4), round(mean(cc_fraud$V16), digits=4), round(mean(cc_fraud$V17), digits=4), round(mean
(cc_fraud$V18), digits=4), round(mean(cc_fraud$V19), digits=4), round(mean(cc_fraud$V2), digits=
4), round(mean(cc_fraud$V20), digits=4), round(mean(cc_fraud$V21), digits=4), round(mean(cc_frau
d$V22), digits=4), round(mean(cc_fraud$V23), digits=4), round(mean(cc_fraud$V24), digits=4), rou
nd(mean(cc_fraud$V25), digits=4), round(mean(cc_fraud$V26), digits=4), round(mean(cc_fraud$V27),
digits=4), round(mean(cc_fraud$V28), digits=4), round(mean(cc_fraud$V3), digits=4), round(mean(c
c_fraud$V4), digits=4), round(mean(cc_fraud$V5), digits=4), round(mean(cc_fraud$V6), digits=4),
 round(mean(cc_fraud$V7), digits=4), round(mean(cc_fraud$V8), digits=4), round(mean(cc_fraud$V
9), digits=4))

#To get variable standard deviation, we will use the sd function. We will nest the sd function w
ithin round to shorten our output to only 4 digits after the decimal.
variable_stddev <- c(round(sd(cc_fraud$Amount), digits=4), round(sd(cc_fraud$Class), digits=4),
 round(sd(cc_fraud$Time), digits=4), round(sd(cc_fraud$V1), digits=4), round(sd(cc_fraud$V10), d
igits=4), round(sd(cc_fraud$V11), digits=4), round(sd(cc_fraud$V12), digits=4), round(sd(cc_frau
d$V13), digits=4), round(sd(cc_fraud$V14), digits=4), round(sd(cc_fraud$V15), digits=4), round(s
d(cc_fraud$V16), digits=4), round(sd(cc_fraud$V17), digits=4), round(sd(cc_fraud$V18), digits=4
), round(sd(cc_fraud$V19), digits=4), round(sd(cc_fraud$V2), digits=4), round(sd(cc_fraud$V20),
 digits=4), round(sd(cc_fraud$V21), digits=4), round(sd(cc_fraud$V22), digits=4), round(sd(cc_fr
aud$V23), digits=4), round(sd(cc_fraud$V24), digits=4), round(sd(cc_fraud$V25), digits=4), round
```

```
(sd(cc_fraud$V26), digits=4), round(sd(cc_fraud$V27), digits=4), round(sd(cc_fraud$V28), digits=
4), round(sd(cc_fraud$V3), digits=4), round(sd(cc_fraud$V4), digits=4), round(sd(cc_fraud$V5), d
igits=4), round(sd(cc_fraud$V6), digits=4), round(sd(cc_fraud$V7), digits=4), round(sd(cc_fraud
$V8), digits=4), round(sd(cc_fraud$V9), digits=4))

#To get variable minimum, we will use the min function. We will nest the min function within rou
nd to shorten our output to only 4 digits after the decimal.
variable_min <- c(round(min(cc_fraud$Amount), digits=4), round(min(cc_fraud$Class), digits=4), r
ound(min(cc_fraud$Time), digits=4), round(min(cc_fraud$V1), digits=4), round(min(cc_fraud$V10),
 digits=4), round(min(cc_fraud$V11), digits=4), round(min(cc_fraud$V12), digits=4), round(min(cc
_fraud$V13), digits=4), round(min(cc_fraud$V14), digits=4), round(min(cc_fraud$V15), digits=4),
 round(min(cc_fraud$V16), digits=4), round(min(cc_fraud$V17), digits=4), round(min(cc_fraud$V1
8), digits=4), round(min(cc_fraud$V19), digits=4), round(min(cc_fraud$V2), digits=4), round(min
(cc_fraud$V20), digits=4), round(min(cc_fraud$V21), digits=4), round(min(cc_fraud$V22), digits=4
), round(min(cc_fraud$V23), digits=4), round(min(cc_fraud$V24), digits=4), round(min(cc_fraud$V2
5), digits=4), round(min(cc_fraud$V26), digits=4), round(min(cc_fraud$V27), digits=4), round(min
(cc_fraud$V28), digits=4), round(min(cc_fraud$V3), digits=4), round(min(cc_fraud$V4), digits=4),
round(min(cc_fraud$V5), digits=4), round(min(cc_fraud$V6), digits=4), round(min(cc_fraud$V7), di
gits=4), round(min(cc_fraud$V8), digits=4), round(min(cc_fraud$V9), digits=4))

#To get variable maximum, we will use the max function. We will nest the max function within rou
nd to shorten our output to only 4 digits after the decimal.
variable_max <- c(round(max(cc_fraud$Amount), digits=4), round(max(cc_fraud$Class), digits=4), r
ound(max(cc_fraud$Time), digits=4), round(max(cc_fraud$V1), digits=4), round(max(cc_fraud$V10),
 digits=4), round(max(cc_fraud$V11), digits=4), round(max(cc_fraud$V12), digits=4), round(max(cc
_fraud$V13), digits=4), round(max(cc_fraud$V14), digits=4), round(max(cc_fraud$V15), digits=4),
 round(max(cc_fraud$V16), digits=4), round(max(cc_fraud$V17), digits=4), round(max(cc_fraud$V1
8), digits=4), round(max(cc_fraud$V19), digits=4), round(max(cc_fraud$V2), digits=4), round(max
(cc_fraud$V20), digits=4), round(max(cc_fraud$V21), digits=4), round(max(cc_fraud$V22), digits=4
), round(max(cc_fraud$V23), digits=4), round(max(cc_fraud$V24), digits=4), round(max(cc_fraud$V2
5), digits=4), round(max(cc_fraud$V26), digits=4), round(max(cc_fraud$V27), digits=4), round(max
(cc_fraud$V28), digits=4), round(max(cc_fraud$V3), digits=4), round(max(cc_fraud$V4), digits=4),
round(max(cc_fraud$V5), digits=4), round(max(cc_fraud$V6), digits=4), round(max(cc_fraud$V7), di
gits=4), round(max(cc_fraud$V8), digits=4), round(max(cc_fraud$V9), digits=4))

#To determine the number of missing values in each column, we can sum up the number of NAs using
the is.na function. The code below will generate a vector of the number of missing values in eac
h column since the is.na function indicates which elements are missing from each column.
variable_missing_values <- c(sum(is.na(cc_fraud$Amount)), sum(is.na(cc_fraud$Class)), sum(is.na
(cc_fraud$Time)), sum(is.na(cc_fraud$V1)), sum(is.na(cc_fraud$V10)), sum(is.na(cc_fraud$V11)), s
um(is.na(cc_fraud$V12)), sum(is.na(cc_fraud$V13)), sum(is.na(cc_fraud$V14)), sum(is.na(cc_fraud
$V15)), sum(is.na(cc_fraud$V16)), sum(is.na(cc_fraud$V17)), sum(is.na(cc_fraud$V18)), sum(is.na
(cc_fraud$V19)), sum(is.na(cc_fraud$V2)), sum(is.na(cc_fraud$V20)), sum(is.na(cc_fraud$V21)), su
m(is.na(cc_fraud$V22)), sum(is.na(cc_fraud$V23)), sum(is.na(cc_fraud$V24)), sum(is.na(cc_fraud$V
25)), sum(is.na(cc_fraud$V26)), sum(is.na(cc_fraud$V27)), sum(is.na(cc_fraud$V28)), sum(is.na(cc
_fraud$V3)), sum(is.na(cc_fraud$V4)), sum(is.na(cc_fraud$V5)), sum(is.na(cc_fraud$V6)), sum(is.n
a(cc_fraud$V7)), sum(is.na(cc_fraud$V8)), sum(is.na(cc_fraud$V9)))

#Next, we will create a dataframe where the variables are the vectors we just created. Using thi
s dataframe, we can print a table in the final output that summarizes each variable.
table.df <- data.frame(column_names, variable_class, variable_type, variable_mean, variable_stdd
ev, variable_min, variable_max, variable_missing_values)
```

```
knitr::kable(table.df, "simple", col.names = c("Column Name", "Variable Class", "Variable Type",
"Variable Mean", "Variable Std Dev", "Variable Min", "Variable Max", "Variable # of NAs"), align
= c("c", "c", "c", "c", "c", "c","c", "c"))
```

Column Name	Variable Class	Variable Type	Variable Mean	Variable Std Dev	Variable Min	Variable Max	Variable # of NAs
Amount	numeric	double	88.3496	250.1201	0.0000	25691.1600	0
Class	integer	integer	0.0017	0.0415	0.0000	1.0000	0
Time	numeric	double	94813.8596	47488.1460	0.0000	172792.0000	0
V1	numeric	double	0.0000	1.9587	-56.4075	2.4549	0
V10	numeric	double	0.0000	1.0888	-24.5883	23.7451	0
V11	numeric	double	0.0000	1.0207	-4.7975	12.0189	0
V12	numeric	double	0.0000	0.9992	-18.6837	7.8484	0
V13	numeric	double	0.0000	0.9953	-5.7919	7.1269	0
V14	numeric	double	0.0000	0.9586	-19.2143	10.5268	0
V15	numeric	double	0.0000	0.9153	-4.4989	8.8777	0
V16	numeric	double	0.0000	0.8763	-14.1299	17.3151	0
V17	numeric	double	0.0000	0.8493	-25.1628	9.2535	0
V18	numeric	double	0.0000	0.8382	-9.4987	5.0411	0
V19	numeric	double	0.0000	0.8140	-7.2135	5.5920	0
V2	numeric	double	0.0000	1.6513	-72.7157	22.0577	0
V20	numeric	double	0.0000	0.7709	-54.4977	39.4209	0
V21	numeric	double	0.0000	0.7345	-34.8304	27.2028	0
V22	numeric	double	0.0000	0.7257	-10.9331	10.5031	0
V23	numeric	double	0.0000	0.6245	-44.8077	22.5284	0
V24	numeric	double	0.0000	0.6056	-2.8366	4.5845	0
V25	numeric	double	0.0000	0.5213	-10.2954	7.5196	0
V26	numeric	double	0.0000	0.4822	-2.6046	3.5173	0
V27	numeric	double	0.0000	0.4036	-22.5657	31.6122	0
V28	numeric	double	0.0000	0.3301	-15.4301	33.8478	0
V3	numeric	double	0.0000	1.5163	-48.3256	9.3826	0
V4	numeric	double	0.0000	1.4159	-5.6832	16.8753	0
V5	numeric	double	0.0000	1.3802	-113.7433	34.8017	0

Column Name	Variable Class	Variable Type	Variable Mean	Variable Std Dev	Variable Min	Variable Max	Variable # of NAs
V6	numeric	double	0.0000	1.3323	-26.1605	73.3016	0
V7	numeric	double	0.0000	1.2371	-43.5572	120.5895	0
V8	numeric	double	0.0000	1.1944	-73.2167	20.0072	0
V9	numeric	double	0.0000	1.0986	-13.4341	15.5950	0

Due to confidentiality issues, the organization releasing the data could not provide the original features on the data (such as variable names). The only features not transformed for the dataset are time and amount. ‘Time’ represents the seconds elapsed between each transaction and the first transaction in the dataset. ‘Amount’ represents the transaction amount. ‘Class’ is the response variable and it takes value 1 in case of fraud and 0 otherwise.

This dataset only contains numerical variables which are the result of PCA transformation. As you can see from the table output, most of the numerical variables (outside of time and amount) have a mean of 0.

Data Preparation

Most of the data preparation needed to perform analysis using this dataset occurred by the organization that released the data. The data had no missing values for any of the columns, and PCA (principal component analysis) transformation was performed using an orthogonal transformation in order to convert a group of correlated variables into a set of uncorrelated variables. As noted in the data description section, the organization releasing the data could not provide the variable names of original features for confidentiality purposes.

Most likely, the organization responsible for the dataset went through a variety of preparation steps in order to produce the clean dataset that we are using now. For instance, there are no missing values in the dataset - the original collector and cleaner of the data may have either assigned values to any records where they were missing, or removed records with missing values. In addition, another step that may have been taken would be to remove outliers from the dataset. Although the point of this project is to identify fraudulent transactions which by nature are anomalies or outliers compared to standard transactions, there may have been records with values that didn’t make sense at all in the context of credit card purchases. If there were any records that could impact the analysis of fraudulent transactions, they may have been removed. The numerical variables V1 trhorugh V28 appeared to be standardized since they all have mean of 0. Lastly, the organization most likely removed extraneous data. If there were any fields that aren’t relevant to the analysis, those were most likely removed from the original set of collected data. In addition, the names of variables are masked for confidentiality purposes.

We will do a few data preparation steps in order to prepare the dataset for analysis, including standardizing the amount variable, removing variables we will not need, and creating training and test datasets for the modeling stage.

```
#First, we will standardize the amount variable using the scale function. The other numeric vari
ables appear to be standardized, so standardizing the amount variable will ensure that all numer
ic variables used for analysis are standardized. The goal in this is to ensure that there are no
extreme values in our dataset that interfere with the functionality of models we develop later.
cc_fraud$Amount=scale(cc_fraud$Amount)

#Next, we will be removing the Time variable as it is not relevant for our analysis. To do this,
we will create a new data frame and remove the first column, time.
cc_fraud_updated=cc_fraud[, -c(1)]

#We will double check that the time variable was removed using the head function to return the f
irst part of the dataframe.
head(cc_fraud_updated)
```

```
##           V1          V2        V3         V4          V5          V6
## 1 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
## 2  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
## 3 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
## 4 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
## 5 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
## 6 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755
##            V7          V8         V9         V10        V11         V12
## 1  0.23959855  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086
## 2 -0.07880298  0.08510165 -0.2554251 -0.16697441  1.6127267  1.06523531
## 3  0.79146096  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369
## 4  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
## 6  0.47620095  0.26031433 -0.5686714 -0.37140720  1.3412620  0.35989384
##          V13        V14        V15        V16         V17         V18
## 1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.20797124  0.02579058
## 2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.11480466 -0.18336127
## 3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.10996938 -0.12135931
## 4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279  1.96577500
## 5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.23703324 -0.03819479
## 6 -0.3580907 -0.1371337  0.5176168  0.4017259 -0.05813282  0.06865315
##           V19         V20          V21          V22         V23         V24
## 1  0.40399296  0.25141210 -0.018306778  0.277837576 -0.11047391  0.06692807
## 2 -0.14578304 -0.06908314 -0.225775248 -0.638671953  0.10128802 -0.33984648
## 3 -2.26185710  0.52497973  0.247998153  0.771679402  0.90941226 -0.68928096
## 4 -1.23262197 -0.20803778 -0.108300452  0.005273597 -0.19032052 -1.17557533
## 5  0.80348692  0.40854236 -0.009430697  0.798278495 -0.13745808  0.14126698
## 6 -0.03319379  0.08496767 -0.208253515 -0.559824796 -0.02639767 -0.37142658
##          V25        V26          V27         V28      Amount Class
## 1  0.1285394 -0.1891148  0.133558377 -0.02105305  0.24496383     0
## 2  0.1671704  0.1258945 -0.008983099  0.01472417 -0.34247394     0
## 3 -0.3276418 -0.1390966 -0.055352794 -0.05975184  1.16068389     0
## 4  0.6473760 -0.2219288  0.062722849  0.06145763  0.14053401     0
## 5 -0.2060096  0.5022922  0.219422230  0.21515315 -0.07340321     0
## 6 -0.2327938  0.1059148  0.253844225  0.08108026 -0.33855582     0
```

```
#It looks like we have the data prepared as we need it - so we will move on to creating a traini
ng and testing dataset for the modeling.

#First, we will load the caTools package. This package contains the sample split function which
 can be used to split the original dataset into training and testing sets.
library(caTools)
```

```
## Warning: package 'caTools' was built under R version 4.1.2
```

```
#Next, we will set a seed so that this analysis can be repeated.
set.seed(123)

#Finally, we can create our training and testing data using the sample.split function and using
 the subset function. We will use 80% of the data for training and the remaining for testing.
data_sample = sample.split(cc_fraud_updated$Class, SplitRatio = 0.80)
training_data <- subset(cc_fraud_updated, data_sample == TRUE)
testing_data <- subset(cc_fraud_updated, data_sample == FALSE)

#We can view the dimensions of the new datasets using the dim function

dim(training_data)
```

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

```
dim(testing_data)
```

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

As we can see, our training dataset has 227846 rows and the testing dataset has 56961 rows. Now, we can move to the modeling phase of the analysis.

Data Modeling and Evaluation

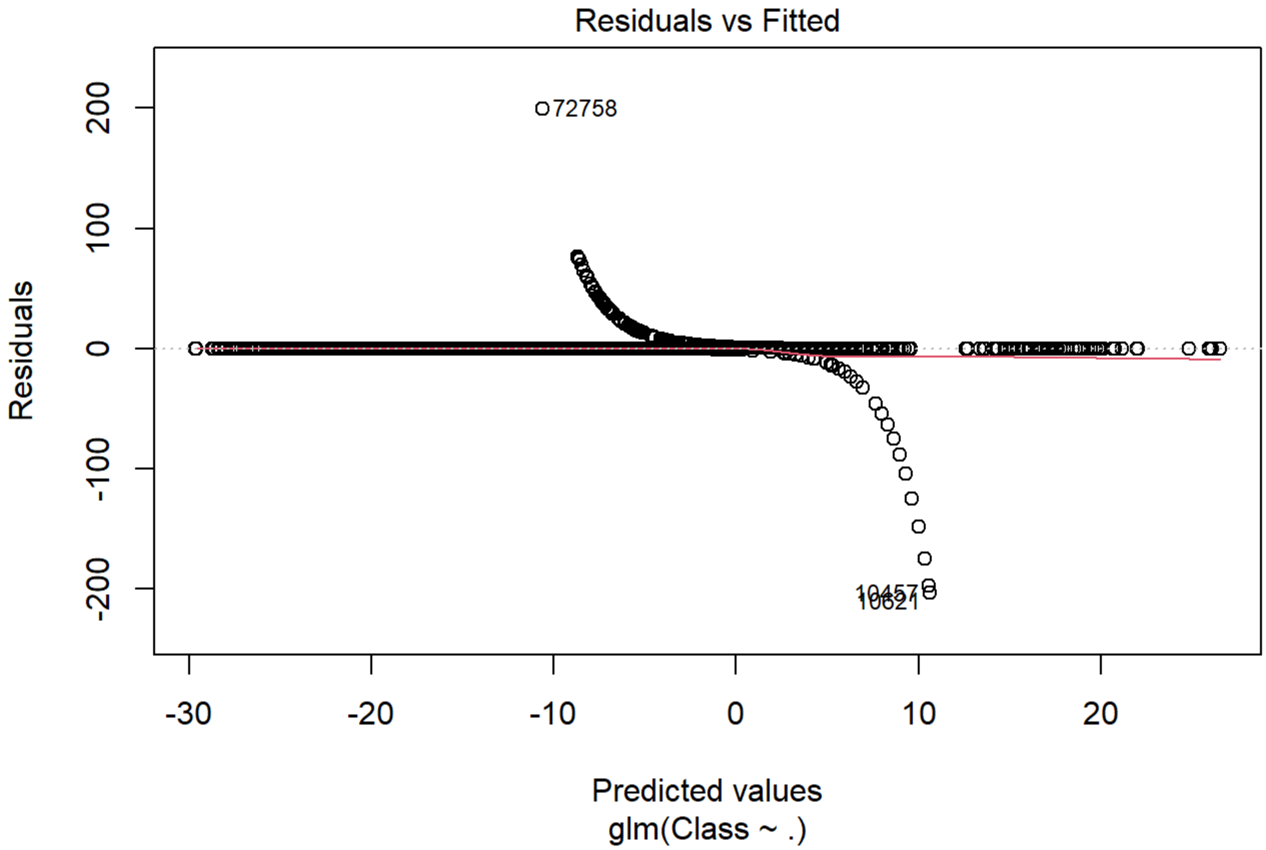
We will use 2 different methods of modeling for detecting fraudulent transactions in our credit card dataset. We will begin with a logistic regression model, then move to a decision tree model.

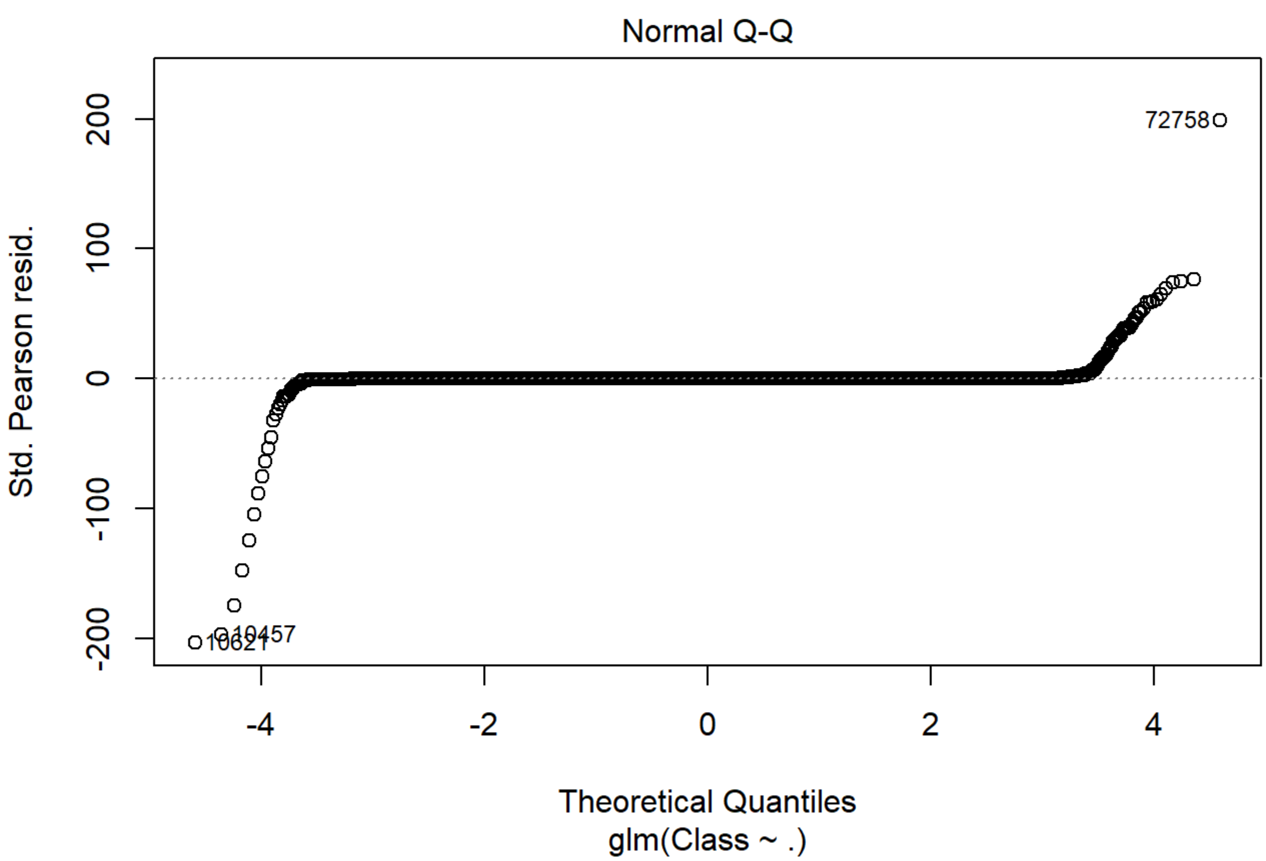
Logistic Regression Model

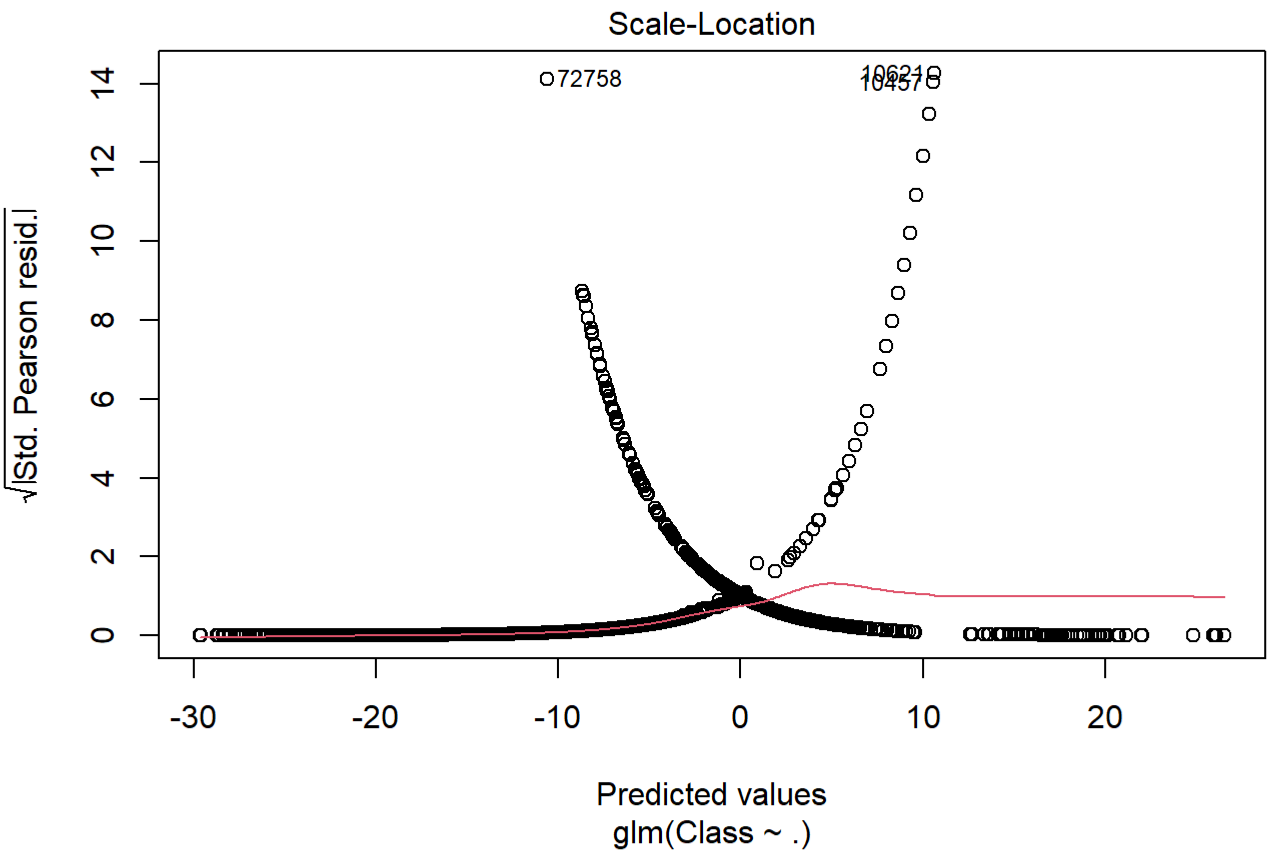
```
#A logistic regression is used for modeling the probability of success for a binary response var
iable. We can use the glm function to create the linear model and specify that it will be a bino
mial response variable.
#We are specifying class as the response variable, and including all other variables available i
n the dataframe as predictors.
Logistic_Model <- glm(Class~.,training_data,family=binomial())
summary(Logistic_Model)
```

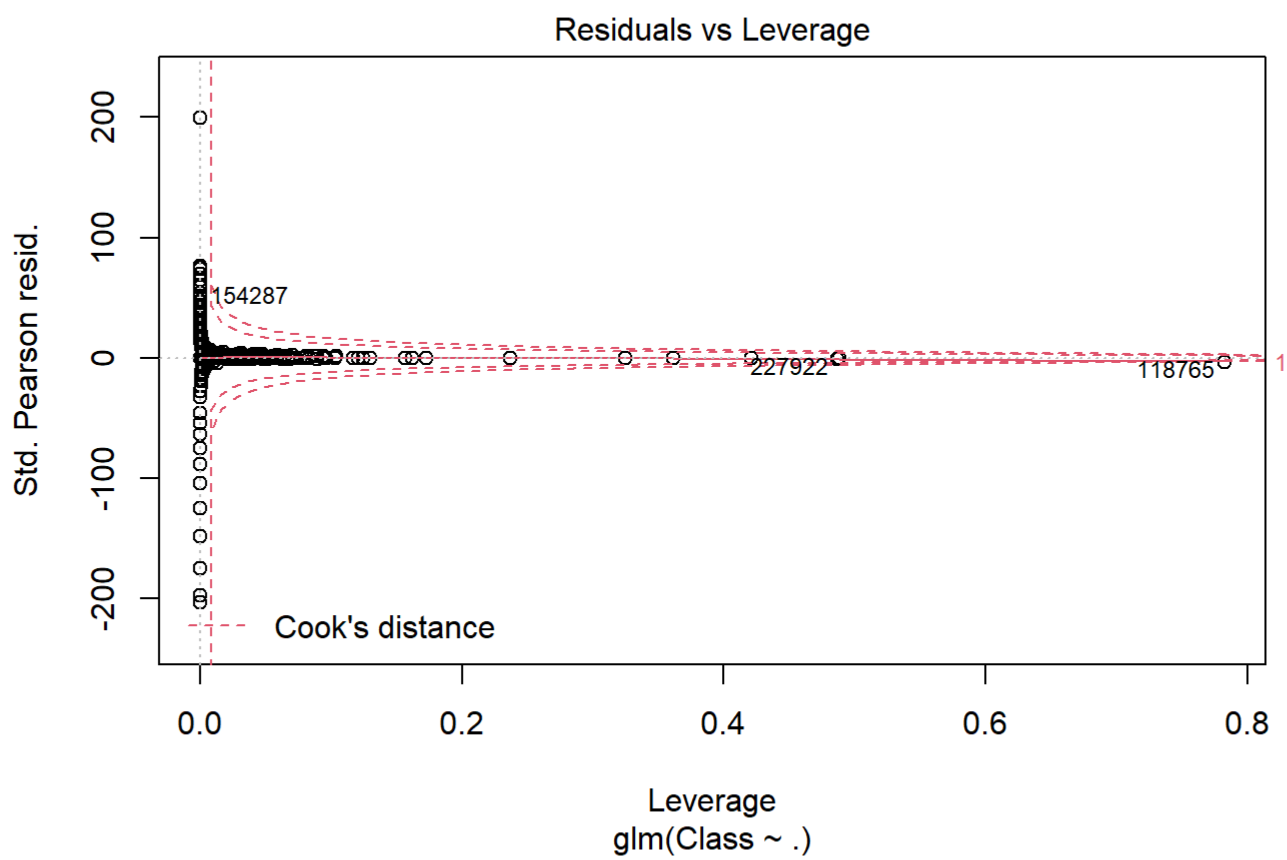
```
##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = training_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.6108  -0.0292  -0.0194  -0.0125   4.6021
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.651305   0.160212 -53.999  < 2e-16 ***
## V1           0.072540   0.044144   1.643 0.100332
## V2           0.014818   0.059777   0.248 0.804220
## V3           0.026109   0.049776   0.525 0.599906
## V4           0.681286   0.078071   8.726  < 2e-16 ***
## V5           0.087938   0.071553   1.229 0.219079
## V6          -0.148083   0.085192  -1.738 0.082170 .
## V7          -0.117344   0.068940  -1.702 0.088731 .
## V8          -0.146045   0.035667  -4.095 4.23e-05 ***
## V9          -0.339828   0.117595  -2.890 0.003855 **
## V10         -0.785462   0.098486  -7.975 1.52e-15 ***
## V11          0.001492   0.085147   0.018 0.986018
## V12          0.087106   0.094869   0.918 0.358532
## V13         -0.343792   0.092381  -3.721 0.000198 ***
## V14         -0.526828   0.067084  -7.853 4.05e-15 ***
## V15         -0.095471   0.094037  -1.015 0.309991
## V16         -0.130225   0.138629  -0.939 0.347537
## V17          0.032463   0.074471   0.436 0.662900
## V18         -0.100964   0.140985  -0.716 0.473909
## V19          0.083711   0.105134   0.796 0.425897
## V20         -0.463946   0.081871  -5.667 1.46e-08 ***
## V21          0.381206   0.065880   5.786 7.19e-09 ***
## V22          0.610874   0.142086   4.299 1.71e-05 ***
## V23         -0.071406   0.058799  -1.214 0.224589
## V24          0.255791   0.170568   1.500 0.133706
## V25         -0.073955   0.142634  -0.519 0.604109
## V26          0.120841   0.202553   0.597 0.550783
## V27         -0.852018   0.118391  -7.197 6.17e-13 ***
## V28         -0.323854   0.090075  -3.595 0.000324 ***
## Amount       0.292477   0.092075   3.177 0.001491 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5799.1  on 227845  degrees of freedom
## Residual deviance: 1790.9  on 227816  degrees of freedom
## AIC: 1850.9
##
## Number of Fisher Scoring iterations: 12
```

```
#After creating the model, we can create some visualizations.

#The plot command will give us 4 useful plots for analyzing the logistic model: Residuals vs Fit
ted values Plot, Normal Q-Q Plot, Scale-Location Plot, and a Residuals vs Leverage Plot.
plot(Logistic_Model)
```


Residuals vs Fitted
Residuals
200
100
0
-100
-200
72758
10457
10621
-30
-20
-10
0
10
20
Predicted values
glm(Class ~ .)


Normal Q-Q
Std. Pearson resid.
200
100
0
-100
-200
72758
10457
10621
-4
-2
0
2
4
Theoretical Quantiles
glm(Class ~ .)


Scale-Location
√|Std. Pearson resid.|
14
12
10
8
6
4
2
0
72758
106210
104578
-30
-20
-10
0
10
20
Predicted values
glm(Class ~ .)


Residuals vs Leverage
Std. Pearson resid.
200
100
0
-100
-200
154287
227922
118765
1
Cook's distance
0.0
0.2
0.4
0.6
0.8
Leverage
glm(Class ~ .)

```r
#To assess the performance of the model, we will delineate the ROC curve. This curve shows the t
rade-off between the sensitivity of the model and the specificity of the model.

#We will need the pROC library, so we will call that first.
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.1.2
```

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

```
##
## Attaching package: 'pROC'
```

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

```r
#We will use the logistic model we created to make predictions on the testing data. We can do th
is using the predict function and calling out the model, data, and probability = TRUE
#Then, we can compare the predictions of the testing data to the actual classes of the testing d
ata and plot it on a ROC curve using the roc function.
lr.predict <- predict(Logistic_Model,testing_data, probability = TRUE)
auc.gbm = roc(testing_data$Class, lr.predict, plot = TRUE, col = "blue")
```

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

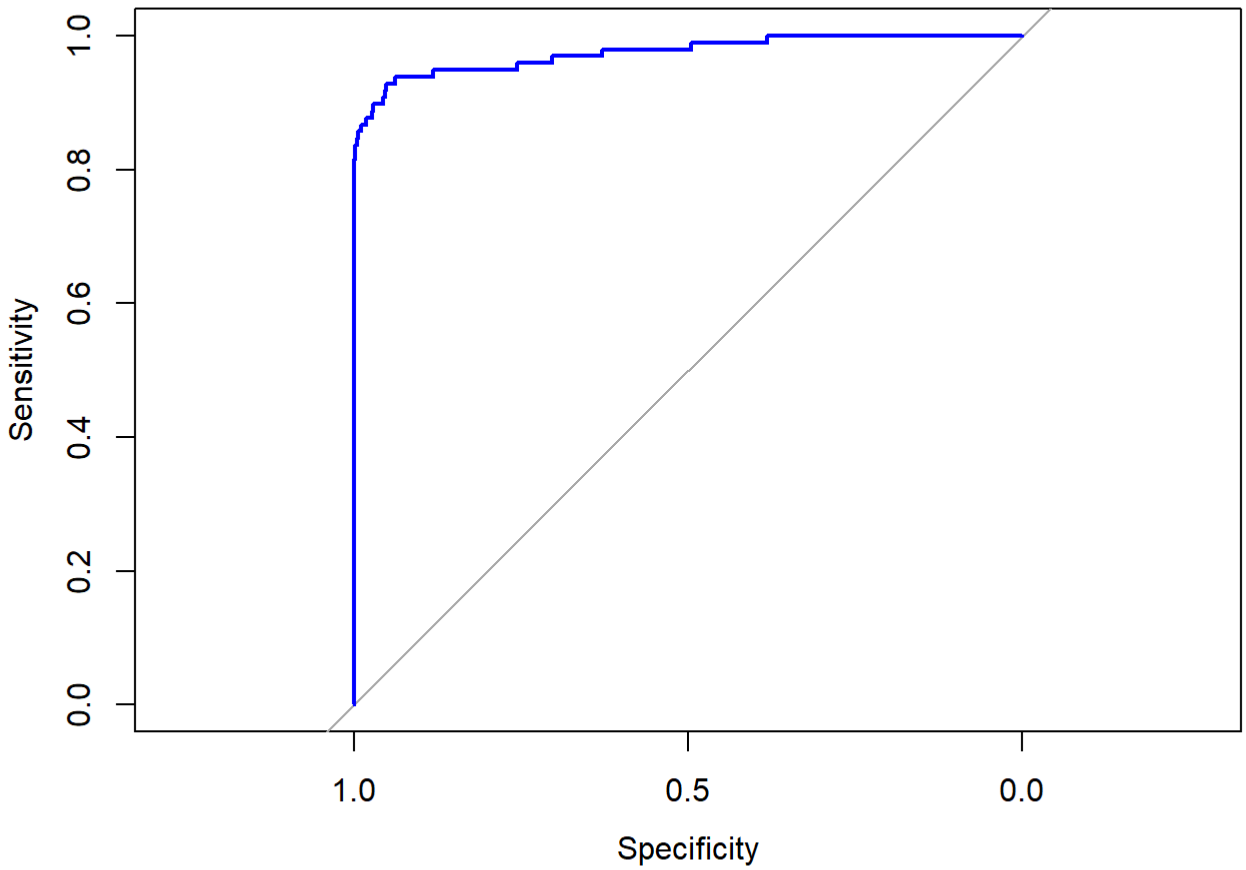
```
## Setting direction: controls < cases
```

```r
#We can also create a confusion matrix to evaluate model performance. We will need the caret pac
kage to do this.
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.2
```

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

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




```
#use logistic regression model to predict probability of fraudulent transaction
#If a transaction has a predicted probability greater than 0.50, we will classify that as 1, or
 fraudulent.
#We will be creating a pred variable in the testing_data in order to compare the predicted class
ifications to the actual classifications.
predicted_prob <- predict(Logistic_Model, testing_data, type = "response")
testing_data$pred <- ifelse(predicted_prob >0.50, "1", "0")
testing_data$pred <- as.factor(testing_data$pred)
testing_data$Class <- as.factor(testing_data$Class)
#create the confusion matrix
confusionMatrix(testing_data$Class, testing_data$pred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 56856     7
##          1    41    57
##
##                Accuracy : 0.9992
##                  95% CI : (0.9989, 0.9994)
##     No Information Rate : 0.9989
##     P-Value [Acc > NIR] : 0.02253
##
##                   Kappa : 0.7033
##
##  Mcnemar's Test P-Value : 1.906e-06
##
##             Sensitivity : 0.9993
##             Specificity : 0.8906
##          Pos Pred Value : 0.9999
##          Neg Pred Value : 0.5816
##              Prevalence : 0.9989
##          Detection Rate : 0.9982
##    Detection Prevalence : 0.9983
##       Balanced Accuracy : 0.9450
##
##        'Positive' Class : 0
##
```

A ROC curve that is close to a 45 degree angle has no predictive power. Since our curve is close to the top left, the logistic model appears to be a useful predictive model for classifying credit card transactions as fraudulent or not. In addition, the confusion matrix indicates a very high accuracy for our logisitic regression model. More will be discussed with regard to model performance after creating the decision tree model.

Decision Tree Model

A decision tree creates a tree of various decisions and their possible consequences. This type of model can be very useful for predicting the classification of a credit card transaction.

```
#First, we will load the necessary packages for fitting a decision tree model. The rpart package will be used to fit the regression tree and the rpart.plot package will be used for plotting the tree.
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 4.1.2
```

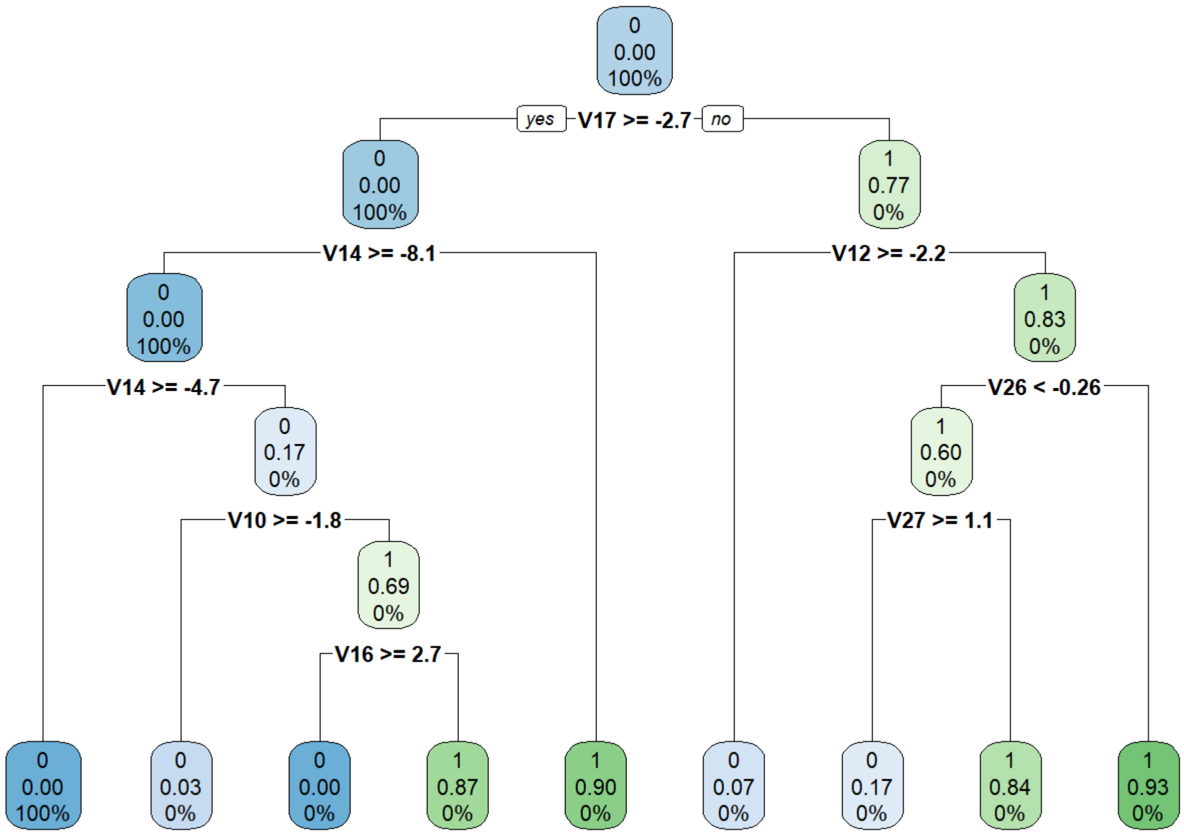
```
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.1.2
```

```
#We can use the rpart function to create our decision tree model. We will fit the decision tree
 using the training dataset. We are going to specify the method as class for classification, and
use the training data just as we did with the logistic regression model.
decisionTree_model <- rpart(Class ~ . , training_data, method = 'class')

#Next, we can generate a list of predicted classifications using the predict function and our de
cision tree model.
predicted_val <- predict(decisionTree_model, testing_data, type = 'class')

#We can also show the predicted probabilities of of class being 1, a fraudulent transaction.
pred_prob_tree <- predict(decisionTree_model, testing_data, type = 'prob')

#Using the predicted probabilities, we can evaluate the performance of the model. If a predicted
probability is greater than 0,5, we will assume the tree is classifying the transaction as fraud
ulent.
#We will create a pred_tree variable in the testing_data in order to evaluate model performance.
testing_data$pred_tree <- ifelse(pred_prob_tree >0.50, "1", "0")
testing_data$pred_tree <- as.factor(testing_data$pred)

#Similar to the logistic model, we can create a confusion matrix. This matrix will be comparing
 the predicted classifications against the actual classifications for the testing data.
confusionMatrix(testing_data$Class, testing_data$pred_tree)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 56856     7
##          1    41    57
##
##                Accuracy : 0.9992
##                  95% CI : (0.9989, 0.9994)
##     No Information Rate : 0.9989
##     P-Value [Acc > NIR] : 0.02253
##
##                   Kappa : 0.7033
##
##  Mcnemar's Test P-Value : 1.906e-06
##
##             Sensitivity : 0.9993
##             Specificity : 0.8906
##          Pos Pred Value : 0.9999
##          Neg Pred Value : 0.5816
##              Prevalence : 0.9989
##          Detection Rate : 0.9982
##    Detection Prevalence : 0.9983
##       Balanced Accuracy : 0.9450
##
##        'Positive' Class : 0
##
```

```
#Lastly, we can plot the decision tree to actually visualize the decision making process for cla
ssification. The rpart.plot function can be used to plot a decision tree model.
rpart.plot(decisionTree_model)
```



Both a logistic regression and a decision tree can be used as tools to classify transactions - based on the results of the confusion matrices, both models do equally well in predicting the correct classification. Out of the 64 total fraudulent transactions in the testing data, both models were able to successfully classify 57 of those, a rate of 89%. For transactions that were not fraudulent, the models incorrectly classified some as fraudulent 41 times out of the 56,897 total. This is a false-positive rate of roughly 0% which is very good. Overall, both models did an excellent job of classifying transactions as fraudulent or not, with an overall accuracy of 99.92% - one thing to keep in mind, though, is that there were only 64 total fraudulent transactions in the testing dataset, so it is crucial to classify fraud correctly as much as possible since they are anomalies in the dataset.

Summary

Identifying fraudulent credit card transactions is an extremely important machine learning problem that many financial companies are actively attempting to perfect. Given a dataset of credit card transactions, we were able to successfully create 2 models to predict if a credit card transaction is fraudulent or not. Both models had overall accuracy around 99%, but I think a decision tree has more value due to the fact that an individual can see the decision making process visualized.