Fraud Detection Modeling for Credit Card Transactions:

A Machine Learning Approach

Francis Boateng

Abstract

This project investigates the effectiveness of machine learning models in identifying fraudulent transactions in a large dataset of over 550,000 records featuring European cardholder transactions from 2023. With scaled features and anonymized names for privacy, the dataset provides a robust foundation for model development. The goal is to train and evaluate various machine learning algorithms to distinguish legitimate from fraudulent transactions, assessing their accuracy and performance. The findings of this project will contribute to the development of efficient fraud detection systems, enhancing the security of financial transactions and minimizing potential losses for cardholders and financial institutions.

Exploratory Data Analysis

In this step, I will get to know the dataset better. I will look at summary statistics to understand the main trends and variations in the data. I'll also check the distribution of each variable to see if there are any patterns or unusual values. Additionally, I'll see how the variables are related, which will help me decide how to build my model. I'll examine how the variables behave differently depending on whether the transaction is normal or fraudulent. Finally, I'll check if there is an imbalance between normal and fraudulent transactions, which is essential for building accurate fraud detection models. By doing this, I will gain a deeper understanding of the data and identify significant trends and relationships to help me build a better model.

```r
# Loading the dataset and removing the id column
fraud <- read.csv("creditcard_2023.csv")
FRAUD <- fraud[,-1]
head(FRAUD)
```

```
##              V1          V2         V3           V4          V5          V6          V7
```

```
## 1 -0.26064780 -0.4696485 2.4962661 -0.08372391 0.12968124 0.7328982 0.5190136
## 2  0.98509973 -0.3560451 0.5580564 -0.42965390 0.27714026 0.4286045 0.4064660
## 3 -0.26027161 -0.9493846 1.7285378 -0.45798629 0.07406165 1.4194811 0.7435111
## 4 -0.15215210 -0.5089587 1.7468401 -1.09017794 0.24948577 1.1433123 0.5182686
## 5 -0.20681952 -0.1652802 1.5270527 -0.44829266 0.10612511 0.5305489 0.6588491
## 6  0.02530229 -0.1405138 1.1911378 -0.70797881 0.43049032 0.4589732 0.6110496
##            V8         V9       V10         V11       V12        V13       V14
## 1 -0.13000605  0.7271593 0.6377345 -0.98702001 0.2934381 -0.9413861 0.5490199
## 2 -0.13311827  0.3474519 0.5298080  0.14010733 1.5642458  0.5740740 0.6277187
## 3 -0.09557601 -0.2612966 0.6907078 -0.27298493 0.6592007  0.8051732 0.6168744
## 4 -0.06512992 -0.2056976 0.5752307 -0.75258096 0.7374830  0.5929937 0.5595350
## 5 -0.21266001  1.0499208 0.9680461 -1.20317111 1.0295774  1.4393102 0.2414540
## 6 -0.09262861  0.1808114 0.4517884  0.03607131 0.8772389 -0.2897211 0.6309925
##          V15         V16       V17       V18        V19        V20          V21
## 1  1.8048786  0.21559799 0.5123067 0.3336437  0.1242702  0.0912019 -0.110551680
## 2  0.7061213  0.78918836 0.4038099 0.2017994 -0.3406871 -0.2339842 -0.194935964
## 3  3.0690248 -0.57751352 0.8865260 0.2394417 -2.3660789  0.3616523 -0.005020278
## 4 -0.6976637 -0.03066898 0.2426292 2.1786160 -1.3450602 -0.3782233 -0.146927137
## 5  0.1530079  0.22453813 0.3664662 0.2917816  0.4453167  0.2472370 -0.106984018
## 6  0.5602009  0.74113155 0.4217663 0.3625039 -0.2427488 -0.0764003 -0.187739355
##           V22         V23        V24        V25        V26         V27
## 1  0.21760614 -0.13479449  0.1659591  0.1262800 -0.4348240 -0.08123011
## 2 -0.60576091  0.07946908 -0.5773949  0.1900897  0.2965027 -0.24805206
## 3  0.70290638  0.94504549 -1.1546656 -0.6055637 -0.3128945 -0.30025804
## 4 -0.03821246 -0.21404819 -1.8931311  1.0039631 -0.5159503 -0.16531649
## 5  0.72972739 -0.16166570  0.3125610 -0.4141162  1.0711256  0.02371160
## 6 -0.53851811 -0.05046499 -0.6315531 -0.4564800  0.2526699  0.06668093
##           V28   Amount Class
## 1 -0.15104549 17982.10     0
## 2 -0.06451192  6531.37     0
## 3 -0.24471823  2513.54     0
## 4  0.04842363  5384.44     0
## 5  0.41911727 14278.97     0
## 6  0.09581151  6901.49     0
```

```r
# Checking for missing values in the data set
colSums(is.na(FRAUD))
```

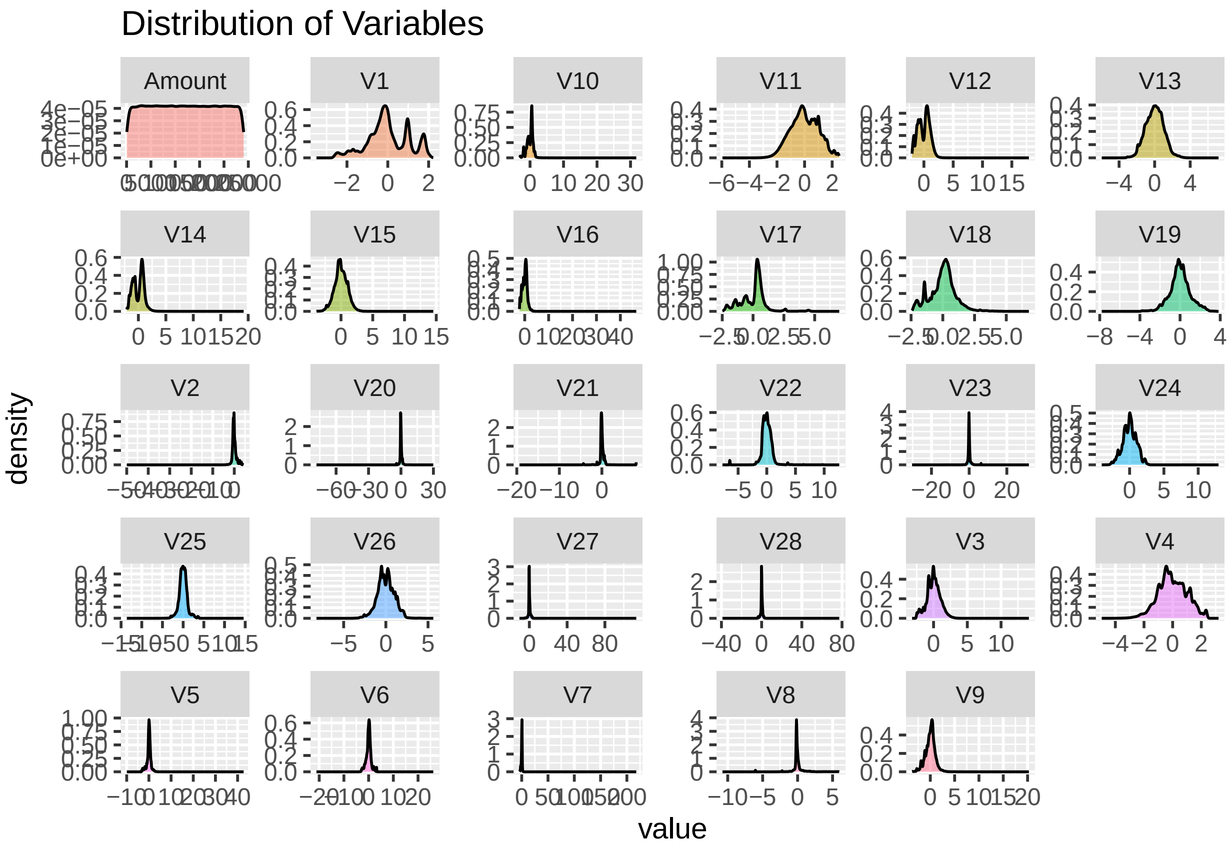
```
##     V1     V2     V3     V4     V5     V6     V7     V8     V9    V10    V11
##      0      0      0      0      0      0      0      0      0      0      0
##    V12    V13    V14    V15    V16    V17    V18    V19    V20    V21    V22
##      0      0      0      0      0      0      0      0      0      0      0
##    V23    V24    V25    V26    V27    V28 Amount  Class
##      0      0      0      0      0      0      0      0
```

```r
# Summary Statistics
print(summary(FRAUD))
```

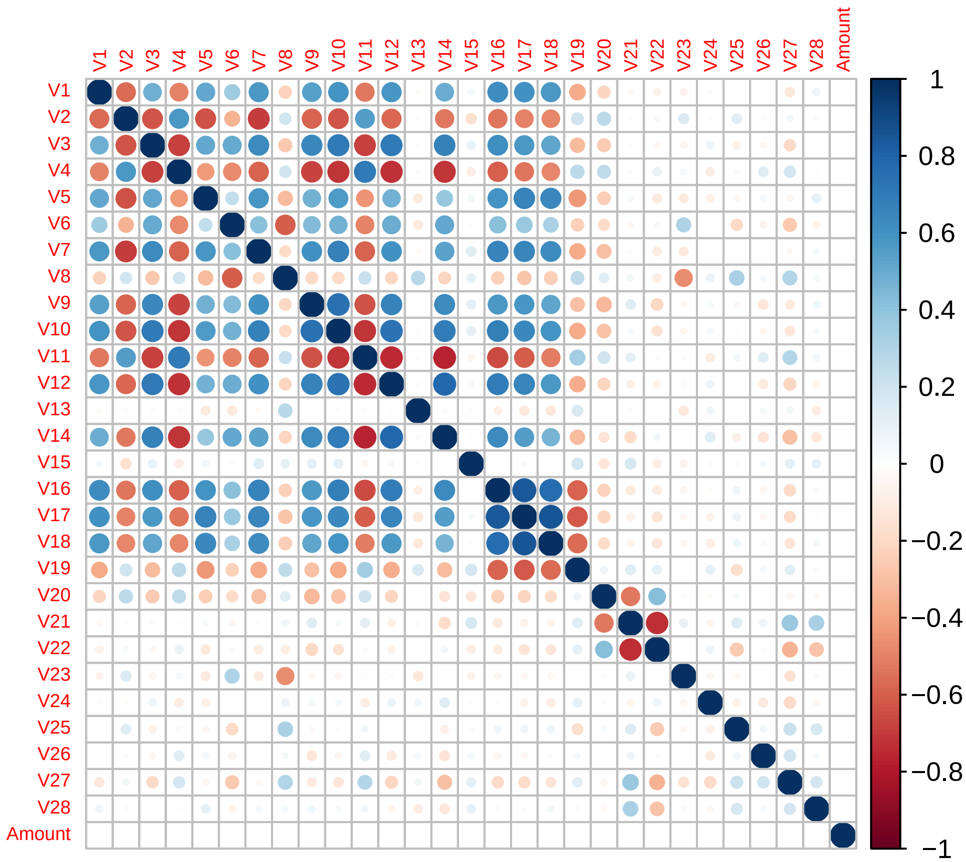
```
##        V1                  V2                  V3                   V4
##  Min.   :-3.49558   Min.   :-49.9666   Min.   :-3.183760   Min.   :-4.95122
##  1st Qu.:-0.56529   1st Qu.: -0.4867   1st Qu.:-0.649299   1st Qu.:-0.65602
##  Median :-0.09364   Median : -0.1359   Median : 0.000353   Median :-0.07376
##  Mean   : 0.00000   Mean   :  0.0000   Mean   : 0.000000   Mean   : 0.00000
##  3rd Qu.: 0.83266   3rd Qu.:  0.3436   3rd Qu.: 0.628538   3rd Qu.: 0.70700
##  Max.   : 2.22905   Max.   :  4.3619   Max.   :14.125834   Max.   : 3.20154
##        V5                  V6                   V7                  V8
##  Min.   :-9.95279   Min.   :-21.11111   Min.   : -4.3518   Min.   :-10.7563
##  1st Qu.:-0.29350   1st Qu.: -0.44587   1st Qu.: -0.2835   1st Qu.: -0.1923
##  Median : 0.08109   Median :  0.07872   Median :  0.2334   Median : -0.1145
##  Mean   : 0.00000   Mean   :  0.00000   Mean   :  0.0000   Mean   :  0.0000
##  3rd Qu.: 0.43974   3rd Qu.:  0.49779   3rd Qu.:  0.5260   3rd Qu.:  0.0473
##  Max.   :42.71689   Max.   : 26.16840   Max.   :217.8730   Max.   :  5.9580
##        V9                 V10                 V11                 V12
##  Min.   :-3.75192   Min.   :-3.1633   Min.   :-5.95472   Min.   :-2.0204
##  1st Qu.:-0.56874   1st Qu.:-0.5901   1st Qu.:-0.70145   1st Qu.:-0.8311
##  Median : 0.09253   Median : 0.2626   Median :-0.04105   Median : 0.1621
##  Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.55926   3rd Qu.: 0.5925   3rd Qu.: 0.74777   3rd Qu.: 0.7447
##  Max.   :20.27006   Max.   :31.7227   Max.   : 2.51357   Max.   :17.9136
##       V13                 V14                 V15                 V16
##  Min.   :-5.95523   Min.   :-2.1074   Min.   :-3.86181   Min.   :-2.2145
##  1st Qu.:-0.69667   1st Qu.:-0.8732   1st Qu.:-0.62125   1st Qu.:-0.7163
##  Median : 0.01761   Median : 0.2305   Median :-0.03926   Median : 0.1340
##  Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.68561   3rd Qu.: 0.7518   3rd Qu.: 0.66541   3rd Qu.: 0.6556
##  Max.   : 7.18749   Max.   :19.1695   Max.   :14.53220   Max.   :46.6529
##       V17                V18                 V19                 V20
##  Min.   :-2.4849   Min.   :-2.42195   Min.   :-7.80499   Min.   :-78.1478
##  1st Qu.:-0.6195   1st Qu.:-0.55605   1st Qu.:-0.56531   1st Qu.: -0.3502
##  Median : 0.2716   Median : 0.08729   Median :-0.02598   Median : -0.1234
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000   Mean   :  0.0000
##  3rd Qu.: 0.5182   3rd Qu.: 0.54439   3rd Qu.: 0.56012   3rd Qu.:  0.2482
##  Max.   : 6.9941   Max.   : 6.78372   Max.   : 3.83167   Max.   : 29.8728
##       V21                  V22                 V23                  V24
##  Min.   :-19.38252   Min.   :-7.73480   Min.   :-30.29545   Min.   :-4.0680
##  1st Qu.: -0.16644   1st Qu.:-0.49049   1st Qu.: -0.23763   1st Qu.:-0.6516
##  Median : -0.03743   Median :-0.02733   Median : -0.05969   Median : 0.0159
##  Mean   :  0.00000   Mean   : 0.00000   Mean   :  0.00000   Mean   : 0.0000
##  3rd Qu.:  0.14798   3rd Qu.: 0.46388   3rd Qu.:  0.15572   3rd Qu.: 0.7007
```

```
##  Max.   :  8.08708   Max.   :12.63251   Max.   : 31.70763   Max.   :12.9656
##       V25                 V26                 V27                 V28
##  Min.   :-13.612633   Min.   :-8.22697   Min.   :-10.4986   Min.   :-39.03524
##  1st Qu.: -0.554148   1st Qu.:-0.63189   1st Qu.: -0.3050   1st Qu.: -0.23188
##  Median : -0.008193   Median :-0.01189   Median : -0.1729   Median : -0.01393
##  Mean   :  0.000000   Mean   : 0.00000   Mean   :  0.0000   Mean   :  0.00000
##  3rd Qu.:  0.550015   3rd Qu.: 0.67289   3rd Qu.:  0.3340   3rd Qu.:  0.40959
##  Max.   : 14.621509   Max.   : 5.62329   Max.   :113.2311   Max.   : 77.25594
##      Amount              Class
##  Min.   :   50.01   Min.   :0.0
##  1st Qu.: 6054.89   1st Qu.:0.0
##  Median :12030.15   Median :0.5
##  Mean   :12041.96   Mean   :0.5
##  3rd Qu.:18036.33   3rd Qu.:1.0
##  Max.   :24039.93   Max.   :1.0
```

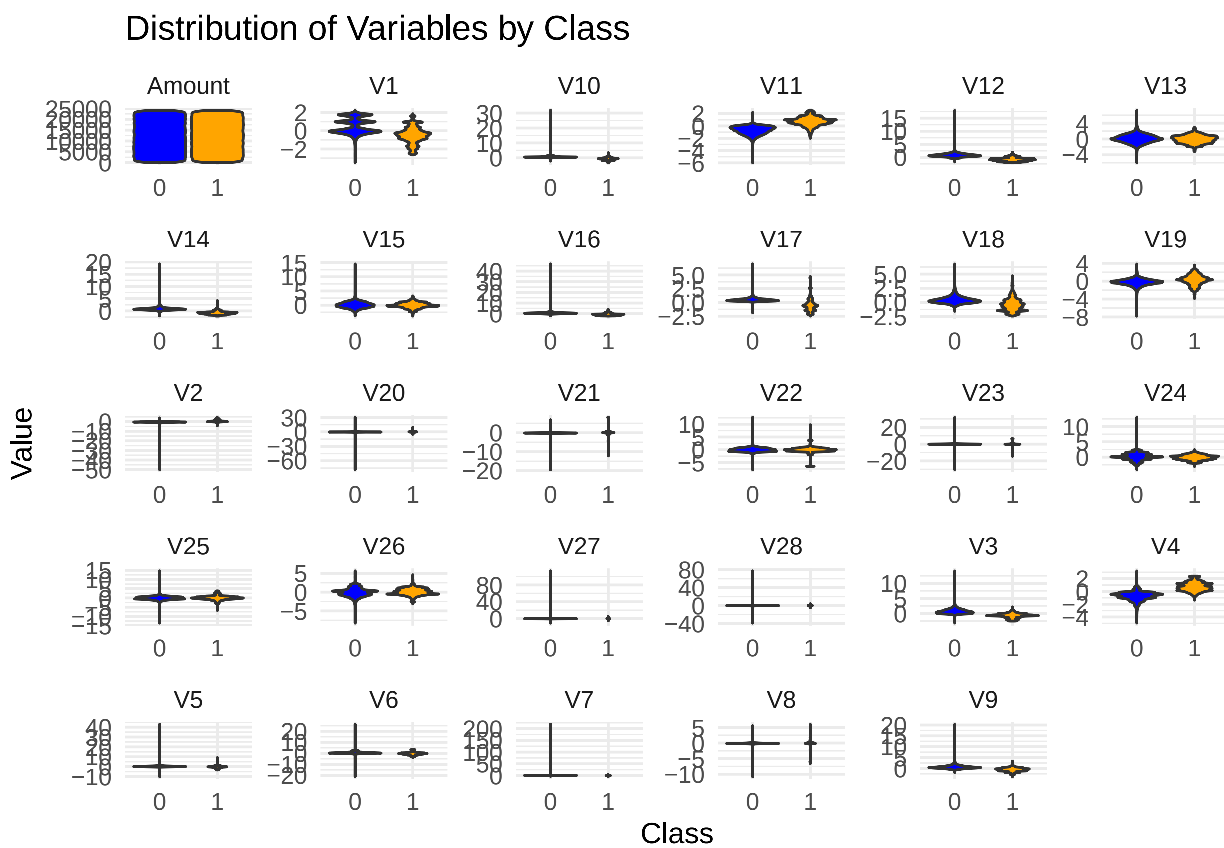
```r
# Creating distribution plots for the variables
FRAUD %>%
  select(-Class) %>%
  gather() %>%
  ggplot(aes(value, fill = key)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~key, scales = "free") +
  labs(title = "Distribution of Variables") +
  guides(fill = "none")
```



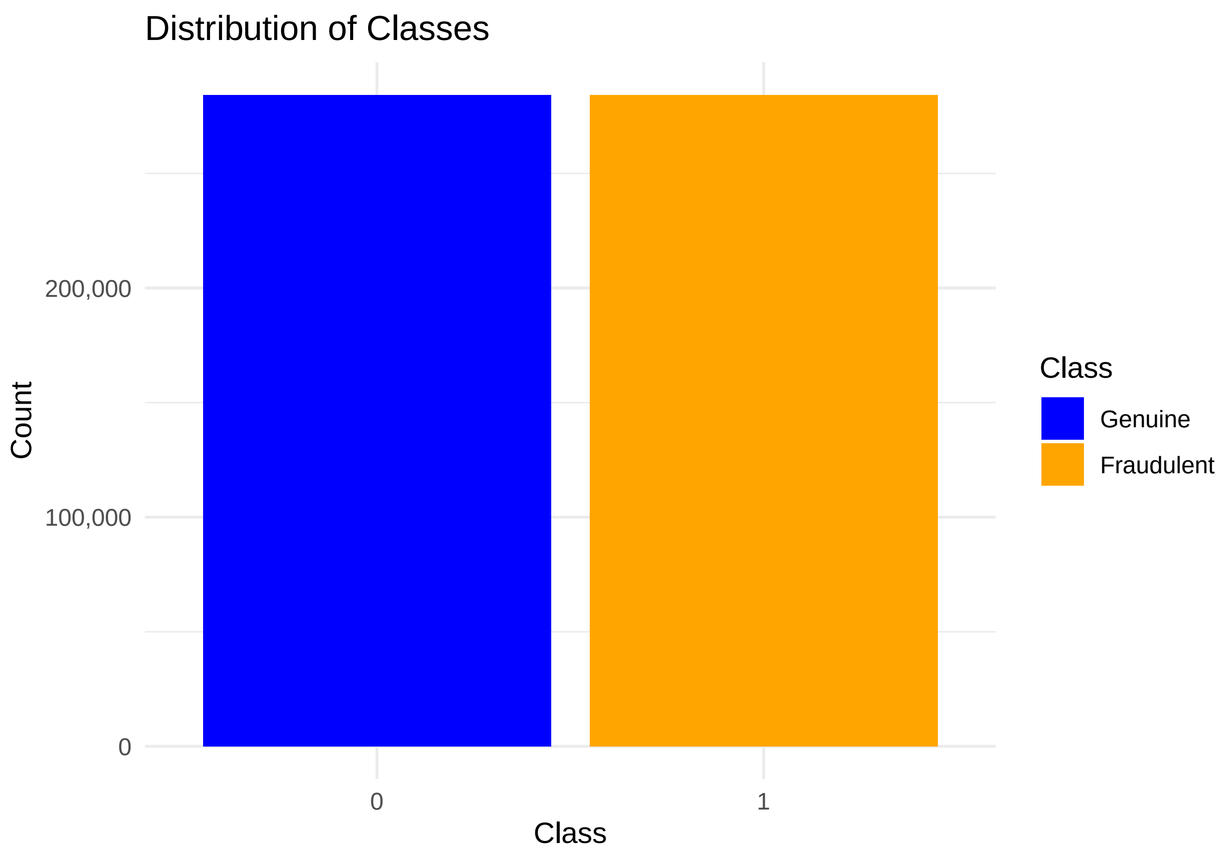

```
# Creating the correlation  plot to check the correlation  between the variables
corr_matrix <- FRAUD %>%
  select(-Class) %>%
  cor()
corrplot(corr_matrix, method = "circle", tl.cex = 0.6)
```




```r
# Creating a violin plot distribution by class
FRAUD %>% gather(key = "Variable", value = "Value", -Class) %>%
    mutate(Class = as.factor(Class)) %>%
    ggplot(aes(x = Class, y = Value, fill = Class)) +
    geom_violin() +
    facet_wrap(~ Variable, scales = "free") +
    labs(title = "Distribution of Variables by Class",
         x = "Class",
         y = "Value") +
    scale_fill_manual(values = c("0" = "blue", "1" = "orange")) +
    theme_minimal() +
    guides(fill = "none")
```




```r
# Plotting the class distribution
ggplot(FRAUD, aes(x = factor(Class), fill = factor(Class))) +
  geom_bar() +
  labs(x = "Class", y = "Count", fill = "Class", title = "Distribution of Classes") +
  scale_fill_manual(values = c("0" = "blue", "1" = "orange"), labels = c("Genuine",    '
  scale_y_continuous(labels = scales::comma) +
  theme_minimal()
```



By exploring the data, I found some interesting things. Some variables are very closely related, meaning they tend to change together in a predictable way, while others move in opposite directions. When I looked at the patterns of fraudulent and genuine transactions, I saw some clear differences that could help me tell them apart.

I also noticed that the number of fraudulent and genuine transactions is almost equal, which is unusual because in real life, there are usually many more genuine transactions than fraudulent ones. This suggests that the data was created artificially, rather than coming from real-world situations. As a result, I won't need to use special techniques to balance the data before training my model.

Data Prepation for Model Building

I'll now prepare the dataset for model training and testing. I'll normalize the 'Amount' feature to be on the same scale as the other variables. This is because it's the only feature that still needs to be transformed using PCA.

Then, I'll divide the dataset into two parts: training and testing sets. I'll use 70% of the data to train the models and the remaining 30% to test their performance. This way, I can ensure the models are well-trained and test them thoroughly in a controlled environment.

```r
# Normalizing the 'Amount' variable
preProcessRange <- preProcess(FRAUD["Amount"], method = c("center", "scale"))
FRAUD_norm <- predict(preProcessRange, FRAUD["Amount"])
FRAUD <- bind_cols(FRAUD[, -which(names(FRAUD) %in% "Amount")], FRAUD_norm)
head(FRAUD)
```

```
##            V1         V2        V3          V4         V5        V6        V7
## 1 -0.26064780 -0.4696485 2.4962661 -0.08372391 0.12968124 0.7328982 0.5190136
## 2  0.98509973 -0.3560451 0.5580564 -0.42965390 0.27714026 0.4286045 0.4064660
## 3 -0.26027161 -0.9493846 1.7285378 -0.45798629 0.07406165 1.4194811 0.7435111
## 4 -0.15215210 -0.5089587 1.7468401 -1.09017794 0.24948577 1.1433123 0.5182686
## 5 -0.20681952 -0.1652802 1.5270527 -0.44829266 0.10612511 0.5305489 0.6588491
## 6  0.02530229 -0.1405138 1.1911378 -0.70797881 0.43049032 0.4589732 0.6110496
##            V8         V9       V10         V11       V12        V13       V14
## 1 -0.13000605  0.7271593 0.6377345 -0.98702001 0.2934381 -0.9413861 0.5490199
## 2 -0.13311827  0.3474519 0.5298080  0.14010733 1.5642458  0.5740740 0.6277187
## 3 -0.09557601 -0.2612966 0.6907078 -0.27298493 0.6592007  0.8051732 0.6168744
## 4 -0.06512992 -0.2056976 0.5752307 -0.75258096 0.7374830  0.5929937 0.5595350
## 5 -0.21266001  1.0499208 0.9680461 -1.20317111 1.0295774  1.4393102 0.2414540
## 6 -0.09262861  0.1808114 0.4517884  0.03607131 0.8772389 -0.2897211 0.6309925
##          V15         V16       V17       V18        V19        V20          V21
## 1  1.8048786  0.21559799 0.5123067 0.3336437  0.1242702  0.0912019 -0.110551680
## 2  0.7061213  0.78918836 0.4038099 0.2017994 -0.3406871 -0.2339842 -0.194935964
## 3  3.0690248 -0.57751352 0.8865260 0.2394417 -2.3660789  0.3616523 -0.005020278
## 4 -0.6976637 -0.03066898 0.2426292 2.1786160 -1.3450602 -0.3782233 -0.146927137
## 5  0.1530079  0.22453813 0.3664662 0.2917816  0.4453167  0.2472370 -0.106984018
## 6  0.5602009  0.74113155 0.4217663 0.3625039 -0.2427488 -0.0764003 -0.187739355
##           V22         V23        V24        V25        V26         V27
## 1  0.21760614 -0.13479449  0.1659591  0.1262800 -0.4348240 -0.08123011
## 2 -0.60576091  0.07946908 -0.5773949  0.1900897  0.2965027 -0.24805206
## 3  0.70290638  0.94504549 -1.1546656 -0.6055637 -0.3128945 -0.30025804
## 4 -0.03821246 -0.21404819 -1.8931311  1.0039631 -0.5159503 -0.16531649
## 5  0.72972739 -0.16166570  0.3125610 -0.4141162  1.0711256  0.02371160
## 6 -0.53851811 -0.05046499 -0.6315531 -0.4564800  0.2526699  0.06668093
##           V28 Class     Amount
## 1 -0.15104549     0  0.8584462
## 2 -0.06451192     0 -0.7963686
## 3 -0.24471823     0 -1.3770097
## 4  0.04842363     0 -0.9621185
## 5  0.41911727     0  0.3232843
## 6  0.09581151     0 -0.7428803
```

```r
# Splitting the data into training and test sets (70% training, 30% test)
set.seed(123)
split <- sample.split(FRAUD$Class, SplitRatio = 0.7)
training_set <- subset(FRAUD, split == TRUE)
test_set <- subset(FRAUD, split == FALSE)

nrow(training_set)
```

```
## [1] 398040
```

```r
nrow(test_set)
```

```
## [1] 170590
```

Developing a Statistical Model for Predictive Analytics

I'll now create four different machine-learning models to help identify fraudulent transactions. These models are logistic regression, decision trees, random forests, and a simple neural network.

Next, I'll test how well these models work by using two important measures: the Area Under the Precision-Recall Curve (AUPRC) and the Area Under the ROC Curve (AUROC). This will help me see how well each model detects fraudulent transactions.

1. Logistic Regression

```r
set.seed(123)
glm_model = glm(Class~. ,data = training_set, family = binomial(link = 'logit'))
summary(glm_model)
```

```
##
## Call:
## glm(formula = Class ~ ., family = binomial(link = "logit"), data = training_set)
##
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -8.490  -0.141   0.000   0.000   3.889
##
## Coefficients:
##              Estimate Std. Error  z value Pr(>|z|)
## (Intercept)  9.043530   0.098505   91.808  < 2e-16 ***
```

```
## V1          -0.686264   0.022348  -30.708  < 2e-16 ***
## V2           0.176351   0.019680    8.961  < 2e-16 ***
## V3          -1.182274   0.023568  -50.163  < 2e-16 ***
## V4           3.634341   0.032042  113.424  < 2e-16 ***
## V5          -0.002097   0.017339   -0.121    0.904
## V6          -0.476370   0.019753  -24.117  < 2e-16 ***
## V7          -1.089877   0.031219  -34.910  < 2e-16 ***
## V8          -2.805206   0.053037  -52.892  < 2e-16 ***
## V9          -0.465499   0.026723  -17.419  < 2e-16 ***
## V10         -1.899152   0.037673  -50.412  < 2e-16 ***
## V11          1.872530   0.021693   86.319  < 2e-16 ***
## V12         -2.798026   0.030775  -90.919  < 2e-16 ***
## V13          0.009754   0.010479    0.931    0.352
## V14         -3.355914   0.031563 -106.324  < 2e-16 ***
## V15         -0.241435   0.010313  -23.411  < 2e-16 ***
## V16         -0.807547   0.029040  -27.808  < 2e-16 ***
## V17         -1.935311   0.032241  -60.026  < 2e-16 ***
## V18         -0.949279   0.024826  -38.238  < 2e-16 ***
## V19         -0.071113   0.014976   -4.748 2.05e-06 ***
## V20          0.150098   0.014699   10.212  < 2e-16 ***
## V21          0.255312   0.033909    7.529 5.10e-14 ***
## V22          0.447060   0.018055   24.761  < 2e-16 ***
## V23         -0.331602   0.012987  -25.533  < 2e-16 ***
## V24         -0.159295   0.010990  -14.494  < 2e-16 ***
## V25          0.173847   0.012963   13.411  < 2e-16 ***
## V26         -0.110695   0.011690   -9.469  < 2e-16 ***
## V27          0.194615   0.026363    7.382 1.56e-13 ***
## V28          0.147788   0.012256   12.059  < 2e-16 ***
## Amount       0.007752   0.009629    0.805    0.421
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 551801  on 398039  degrees of freedom
## Residual deviance:  74941  on 398010  degrees of freedom
## AIC: 75001
##
## Number of Fisher Scoring iterations: 25
```

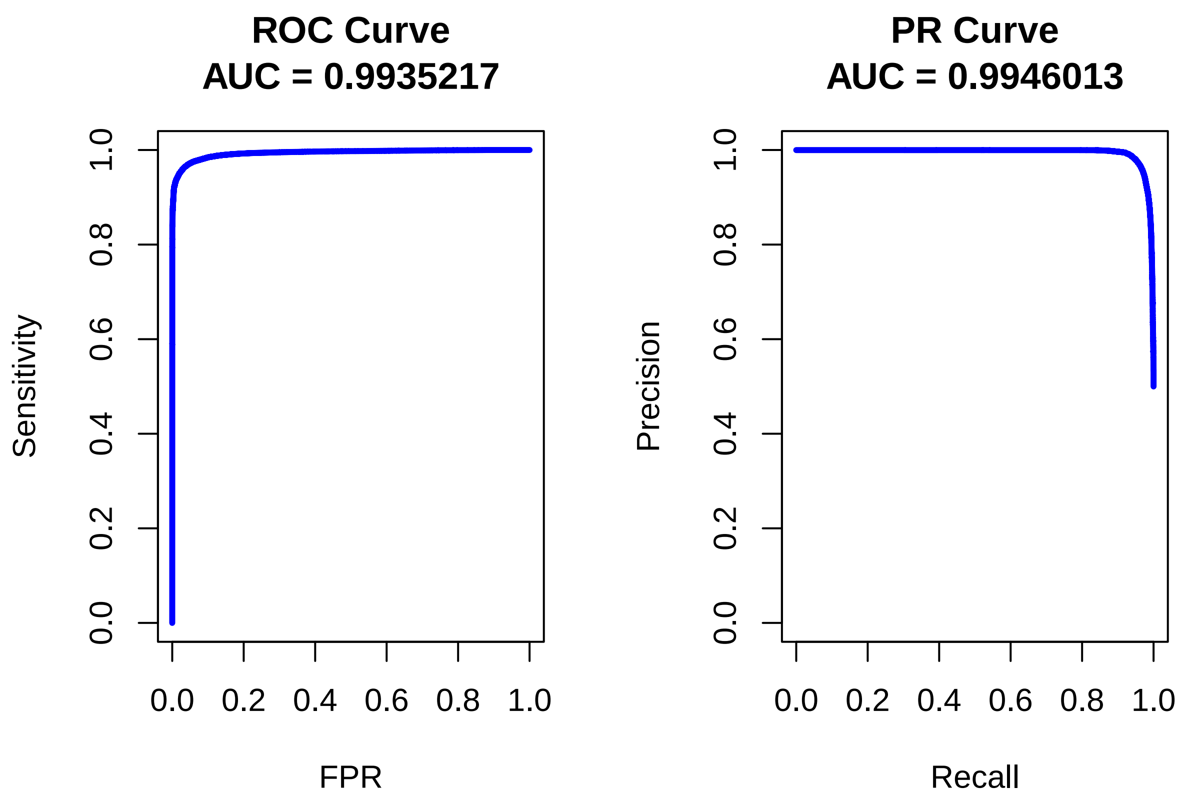
```r
pred_glm <- predict(glm_model, newdata = test_set, type = 'response')
```

```r
# ROC and PR Curves for GLM  model
glm_fg <- pred_glm[test_set$Class == 1]
```

```r
glm_bg <- pred_glm[test_set$Class == 0]

glm_roc <- roc.curve(scores.class0 = glm_fg , scores.class1 = glm_bg, curve = T)
glm_pr <- pr.curve(scores.class0 = glm_fg , scores.class1 = glm_bg, curve = T)

options(repr.plot.width=12, repr.plot.height=7)
par(mfrow = c(1, 2))
plot(glm_roc, col = "blue", main = "ROC Curve")
plot(glm_pr, col = "blue", main = "PR Curve")
```



The Logistic Regression model showed AUROC of 0.9935217 and AUPRC of 0.9946013.

2. Decision Tree

```r
set.seed(123)
dt_model <- rpart(Class ~ ., data = training_set, method = "class")
summary(dt_model)
```

```
## Call:
```

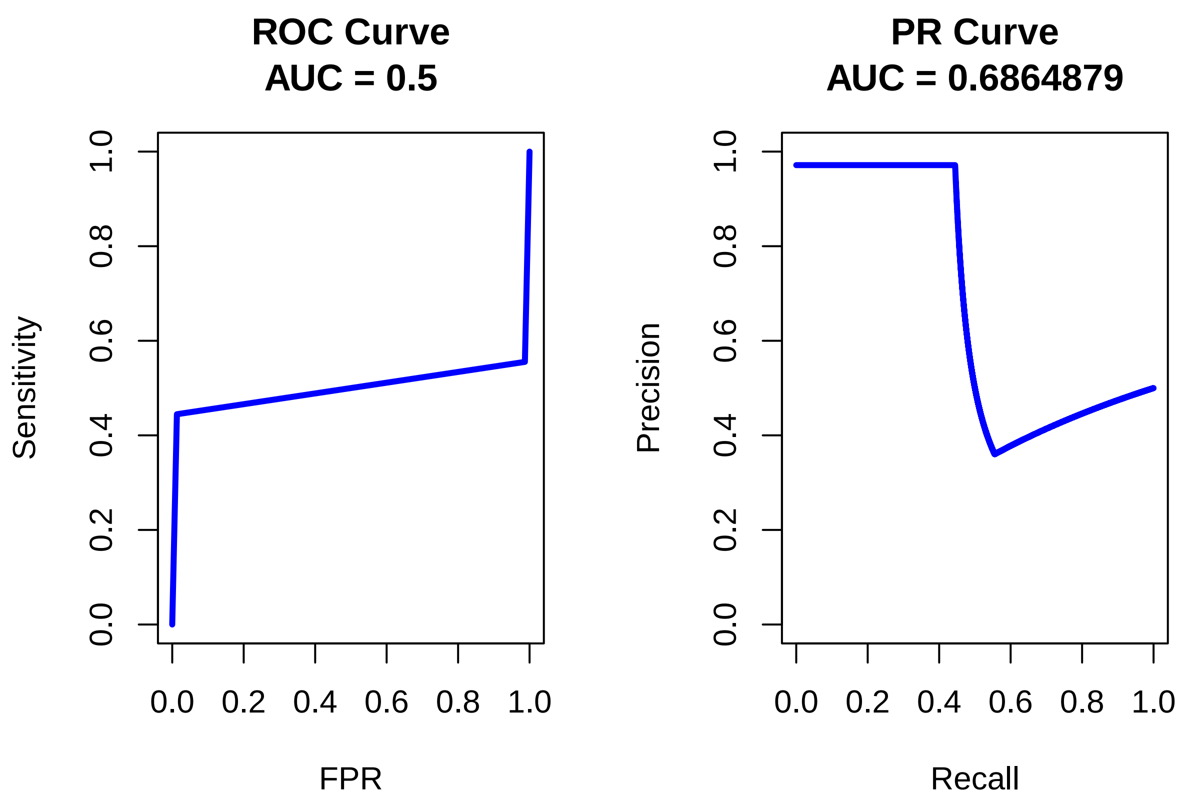
```
## rpart(formula = Class ~ ., data = training_set, method = "class")
##   n= 398040
##
##          CP nsplit rel error    xerror         xstd
## 1 0.8637825      0 1.0000000 1.0039092 0.0015850148
## 2 0.0100000      1 0.1362175 0.1365642 0.0007995806
##
## Variable importance
## V14 V10 V12 V11 V17  V3
##  20  17  16  16  16  15
##
## Node number 1: 398040 observations,    complexity param=0.8637825
##   predicted class=0  expected loss=0.5  P(node) =1
##     class counts: 199020 199020
##    probabilities: 0.500 0.500
##   left son=2 (216200 obs) right son=3 (181840 obs)
##   Primary splits:
##       V14 < 0.008223371 to the right, improve=149607.7, (0 missing)
##       V10 < 0.0121387   to the right, improve=138591.8, (0 missing)
##       V12 < -0.2313897  to the right, improve=126566.6, (0 missing)
##       V4  < -0.03006013 to the left,  improve=122499.8, (0 missing)
##       V17 < 0.09257648  to the right, improve=122089.1, (0 missing)
##   Surrogate splits:
##       V10 < 0.01730558  to the right, agree=0.948, adj=0.886, (0 split)
##       V12 < -0.2313897  to the right, agree=0.918, adj=0.821, (0 split)
##       V11 < 0.2068063   to the left,  agree=0.918, adj=0.820, (0 split)
##       V17 < 0.09264274  to the right, agree=0.913, adj=0.809, (0 split)
##       V3  < -0.1272986  to the right, agree=0.883, adj=0.744, (0 split)
##
## Node number 2: 216200 observations
##   predicted class=0  expected loss=0.1024283  P(node) =0.5431615
##     class counts: 194055 22145
##    probabilities: 0.898 0.102
##
## Node number 3: 181840 observations
##   predicted class=1  expected loss=0.02730422  P(node) =0.4568385
##     class counts:  4965 176875
##    probabilities: 0.027 0.973
```

```r
pred_dt <- predict(dt_model, newdata = test_set, type = "prob")
```

```r
# ROC and PR Curves for decision tree  model
dt_fg <- pred_dt[test_set$Class == 1]
dt_bg <- pred_dt[test_set$Class == 0]
```

```r
dt_roc <- roc.curve(scores.class0 = dt_fg , scores.class1 = dt_bg, curve = T)
dt_pr <- pr.curve(scores.class0 = dt_fg , scores.class1 = dt_bg, curve = T)

par(mfrow = c(1, 2))
plot(dt_roc, col = "blue", main = "ROC Curve")
plot(dt_pr, col = "blue", main = "PR Curve")
```



The Decision Tree model showed AUROC of 0.5 and AUPRC of 0.6864879.

3. Random Forest

```r
set.seed(123)
rf_model <- randomForest(Class ~ ., data = training_set, ntree = 10)
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values.  Are you sure you want to do regression?
```

```r
summary(rf_model)
```

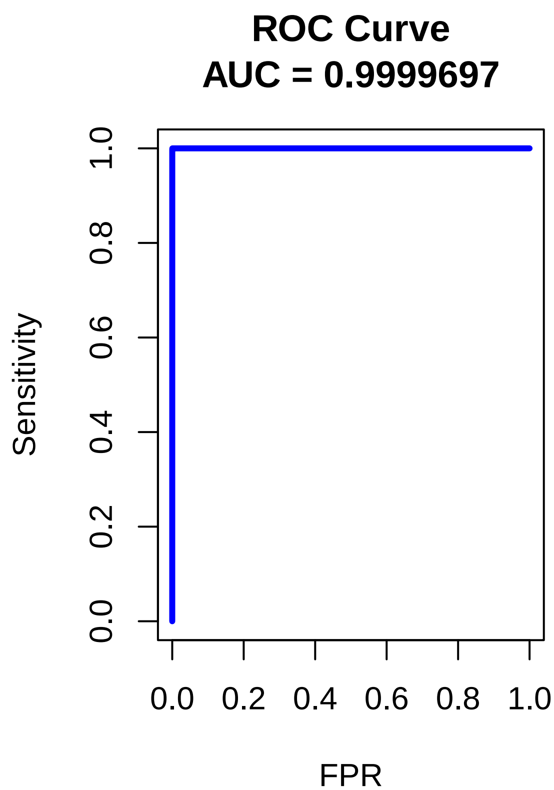
```
##                 Length Class  Mode
## call                 4 -none- call
## type                 1 -none- character
## predicted       398040 -none- numeric
## mse                 10 -none- numeric
## rsq                 10 -none- numeric
## oob.times       398040 -none- numeric
## importance          29 -none- numeric
## importanceSD         0 -none- NULL
## localImportance      0 -none- NULL
## proximity            0 -none- NULL
## ntree                1 -none- numeric
## mtry                 1 -none- numeric
## forest              11 -none- list
## coefs                0 -none- NULL
## y               398040 -none- numeric
## test                 0 -none- NULL
## inbag                0 -none- NULL
## terms                3 terms  call
```

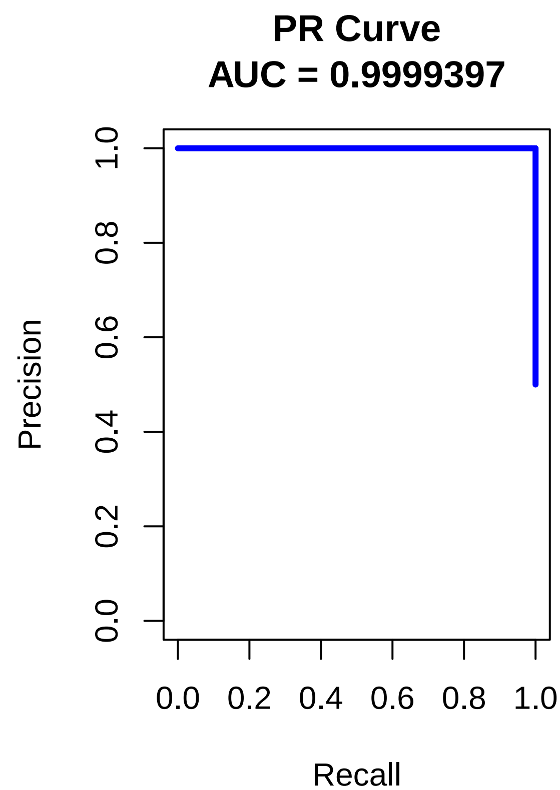
```r
pred_rf <- predict(rf_model, newdata = test_set, type = "class")
```

```r
# ROC and PR Curves for random forest model
rf_fg <- pred_rf[test_set$Class == 1]
rf_bg <- pred_rf[test_set$Class == 0]

rf_roc <- roc.curve(scores.class0 = rf_fg , scores.class1 = rf_bg, curve = T)
rf_pr <- pr.curve(scores.class0 = rf_fg , scores.class1 = rf_bg, curve = T)

par(mfrow = c(1, 2))
plot(rf_roc, col = "blue", main = "ROC Curve")
plot(rf_pr, col = "blue", main = "PR Curve")
```





The Random Forest model showed AUROC of 0.9999996 and AUPRC of 0.9999995.

4. Neural Network

```
set.seed(123)
nn_model <- nnet(Class ~ ., data = training_set, size = 10, linout = FALSE, maxit = 200)
```

```
## # weights:  311
## initial  value 102009.589037
## iter  10 value 34177.761401
## iter  20 value 24791.985455
## iter  30 value 20678.775000
## iter  40 value 20465.142281
## iter  50 value 19403.995463
## iter  60 value 19247.385257
## iter  70 value 18598.562075
## iter  80 value 18299.518540
## iter  90 value 18257.028098
## iter 100 value 18118.303157
## iter 110 value 17824.777563
```

```
## iter 120 value 17715.848363
## iter 130 value 17526.599233
## iter 140 value 17381.762525
## iter 150 value 17371.795110
## iter 160 value 17354.815553
## iter 170 value 17299.419974
## iter 180 value 17227.509980
## iter 190 value 17171.664654
## iter 200 value 17144.778410
## final  value 17144.778410
## stopped after 200 iterations
```

```r
summary(nn_model)
```

```
## a 29-10-1 network with 311 weights
## options were -
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1  i9->h1
##   36.07   69.87  -56.45   84.33  -90.61   45.56   55.64   57.26  -24.29   66.09
## i10->h1 i11->h1 i12->h1 i13->h1 i14->h1 i15->h1 i16->h1 i17->h1 i18->h1 i19->h1
##   80.43  -87.45   93.90   -4.06   98.10  -16.21   73.68   63.76   56.49  -42.18
## i20->h1 i21->h1 i22->h1 i23->h1 i24->h1 i25->h1 i26->h1 i27->h1 i28->h1 i29->h1
##  -22.11  -17.28   -5.83   -2.99    4.25   -8.79   -3.59  -22.05   -6.19    9.30
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2  i9->h2
##   54.15   49.25  131.86 -129.91  217.28 -192.38  102.10   13.00  -41.57 -121.43
## i10->h2 i11->h2 i12->h2 i13->h2 i14->h2 i15->h2 i16->h2 i17->h2 i18->h2 i19->h2
## -122.96  120.72 -275.77  -60.87 -268.02 -100.08  102.82  117.60  -69.52  -98.05
## i20->h2 i21->h2 i22->h2 i23->h2 i24->h2 i25->h2 i26->h2 i27->h2 i28->h2 i29->h2
##  -22.14   55.41 -162.66 -121.61 -112.03  228.38   89.89   29.21   29.46   17.02
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3  i9->h3
##  343.41   93.61   47.99 -259.34  112.10  -37.53   47.87   78.08    6.54   44.55
## i10->h3 i11->h3 i12->h3 i13->h3 i14->h3 i15->h3 i16->h3 i17->h3 i18->h3 i19->h3
## -191.26  191.94 -110.57   79.27 -349.74 -237.74  -76.20  130.42  137.13 -117.00
## i20->h3 i21->h3 i22->h3 i23->h3 i24->h3 i25->h3 i26->h3 i27->h3 i28->h3 i29->h3
## -129.78   28.89  158.57   71.94   62.95 -194.33 -405.25   -2.75  101.09  271.12
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4  i9->h4
## -210.89   72.20  -59.82  129.27 -128.92   33.14   98.24   58.78  -29.31   99.31
## i10->h4 i11->h4 i12->h4 i13->h4 i14->h4 i15->h4 i16->h4 i17->h4 i18->h4 i19->h4
##  129.62 -155.84  165.76    0.52  186.84  -42.16  135.08  112.20   59.59  -52.70
## i20->h4 i21->h4 i22->h4 i23->h4 i24->h4 i25->h4 i26->h4 i27->h4 i28->h4 i29->h4
##  -53.15  -34.08   10.55   11.97    9.95   24.50   -8.41  -73.49  -53.46   11.23
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5  i9->h5
## -170.52   64.14  -22.78  119.64 -116.99   15.27   56.00   42.74   39.80   55.83
## i10->h5 i11->h5 i12->h5 i13->h5 i14->h5 i15->h5 i16->h5 i17->h5 i18->h5 i19->h5
##  113.52 -164.15  172.56   22.97  183.53   34.80  131.32  138.12   85.96  -14.61
```

```
## i20->h5 i21->h5 i22->h5 i23->h5 i24->h5 i25->h5 i26->h5 i27->h5 i28->h5 i29->h5
##  -14.58  -58.02   42.76  -36.38   -2.81   46.34   40.22  -44.00  -33.35   20.11
##    b->h6   i1->h6   i2->h6   i3->h6   i4->h6   i5->h6   i6->h6   i7->h6   i8->h6   i9->h6
## -135.26   60.28   43.30   18.30 -158.50   53.18  -38.42   61.25   85.88   76.33
## i10->h6 i11->h6 i12->h6 i13->h6 i14->h6 i15->h6 i16->h6 i17->h6 i18->h6 i19->h6
##   61.29 -139.39  109.10  125.98  208.59 -121.09   37.66   66.14   25.22   -4.70
## i20->h6 i21->h6 i22->h6 i23->h6 i24->h6 i25->h6 i26->h6 i27->h6 i28->h6 i29->h6
##  -22.93   10.71 -112.80  -21.21  -15.28  -93.50  -19.75   31.13   -5.26   10.83
##    b->h7   i1->h7   i2->h7   i3->h7   i4->h7   i5->h7   i6->h7   i7->h7   i8->h7   i9->h7
##  -95.33  162.23   22.28  116.16 -111.47  136.74   72.73  -65.38   13.25   29.74
## i10->h7 i11->h7 i12->h7 i13->h7 i14->h7 i15->h7 i16->h7 i17->h7 i18->h7 i19->h7
##  165.07   -2.28  169.07  -30.17  144.57   24.97  248.62  208.23   66.01  -80.06
## i20->h7 i21->h7 i22->h7 i23->h7 i24->h7 i25->h7 i26->h7 i27->h7 i28->h7 i29->h7
##   76.42  -50.07 -119.57    8.50   -8.15    2.04   37.70  -39.54  -41.87  174.51
##    b->h8   i1->h8   i2->h8   i3->h8   i4->h8   i5->h8   i6->h8   i7->h8   i8->h8   i9->h8
##  124.65 -121.36  -66.42  -91.95  -38.00   32.87  -28.32   67.47  -32.37    5.10
## i10->h8 i11->h8 i12->h8 i13->h8 i14->h8 i15->h8 i16->h8 i17->h8 i18->h8 i19->h8
##  -97.68   39.67  -86.34    9.90 -115.50   37.20   48.99   50.09   55.77  -84.45
## i20->h8 i21->h8 i22->h8 i23->h8 i24->h8 i25->h8 i26->h8 i27->h8 i28->h8 i29->h8
##  -32.95   36.92   -4.23  -36.76 -104.81   85.92  109.39    0.34  -12.55   -0.72
##    b->h9   i1->h9   i2->h9   i3->h9   i4->h9   i5->h9   i6->h9   i7->h9   i8->h9   i9->h9
## -298.18   71.28   52.50  169.27 -163.22   40.04   32.21  -37.27   40.17   42.86
## i10->h9 i11->h9 i12->h9 i13->h9 i14->h9 i15->h9 i16->h9 i17->h9 i18->h9 i19->h9
##  144.94  -99.53  189.55   25.80  178.69  -16.65  135.96   98.99   32.11  -74.66
## i20->h9 i21->h9 i22->h9 i23->h9 i24->h9 i25->h9 i26->h9 i27->h9 i28->h9 i29->h9
##   29.55  -73.92   64.25  -52.59  -40.95   12.49   80.74 -119.47  -13.32   14.37
##   b->h10  i1->h10  i2->h10  i3->h10  i4->h10  i5->h10  i6->h10  i7->h10
##   167.83   -58.48    11.27  -121.24   156.36   -18.19    -7.46    -2.06
##  i8->h10  i9->h10 i10->h10 i11->h10 i12->h10 i13->h10 i14->h10 i15->h10
##   -37.84   -76.36    36.29    79.75  -156.68    93.96   -92.71    15.08
## i16->h10 i17->h10 i18->h10 i19->h10 i20->h10 i21->h10 i22->h10 i23->h10
##  -103.94  -139.67  -114.53    85.75     5.06   -11.53   105.26    38.60
## i24->h10 i25->h10 i26->h10 i27->h10 i28->h10 i29->h10
##    47.39  -102.69    68.28    -6.92   -32.39    26.50
##     b->o    h1->o    h2->o    h3->o    h4->o    h5->o    h6->o    h7->o    h8->o    h9->o
##   335.70 -165.74  571.94  616.76   61.80 -160.02 -363.83 -455.47  292.62 -194.20
##   h10->o
##   323.28
```

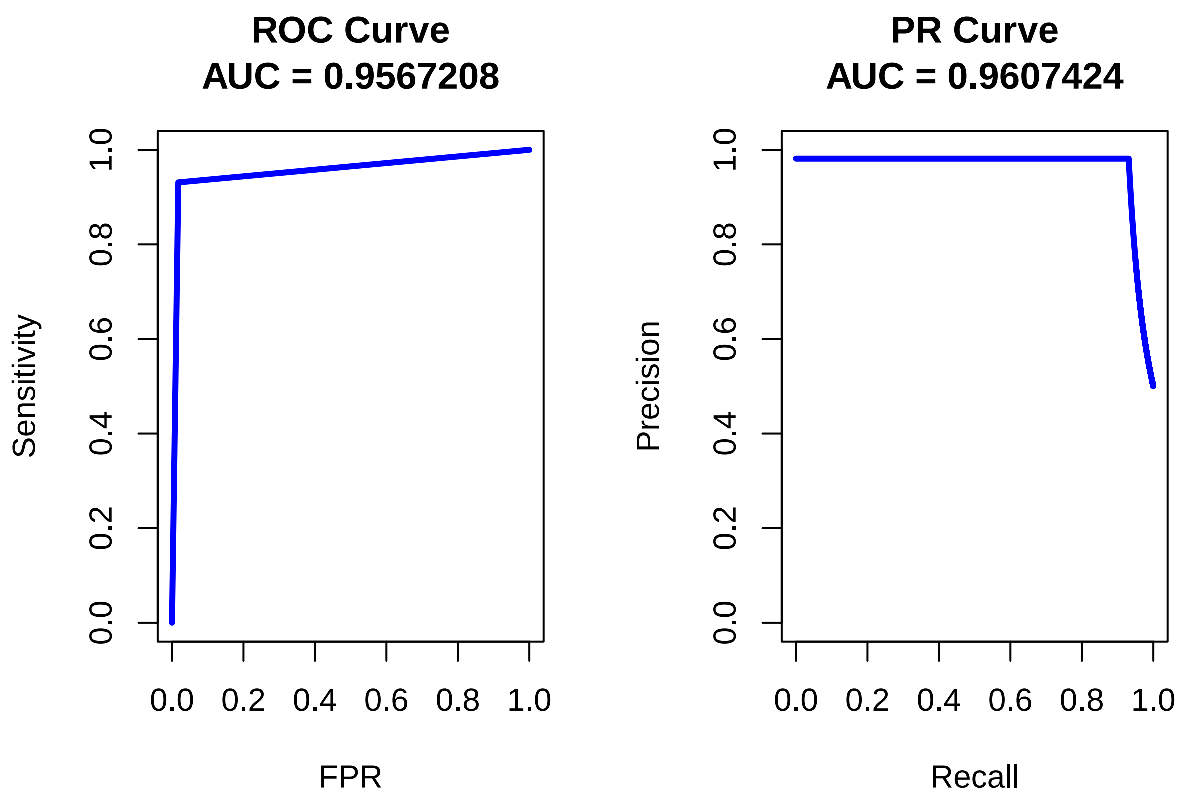
```r
nn_predictions_probs <- predict(nn_model, newdata=test_set[, !names(test_set) %in% "Clas
pred_nn <- ifelse(nn_predictions_probs > 0.5, 1, 0)
```

```r
# ROC and PR Curves for random forest model
nn_fg <- pred_nn[test_set$Class == 1]
```

```r
nn_bg <- pred_nn[test_set$Class == 0]

nn_roc <- roc.curve(scores.class0 = nn_fg , scores.class1 = nn_bg, curve = T)
nn_pr <- pr.curve(scores.class0 = nn_fg , scores.class1 = nn_bg, curve = T)

par(mfrow = c(1, 2))
plot(nn_roc, col = "blue", main = "ROC Curve")
plot(nn_pr, col = "blue", main = "PR Curve")
```



The Neural Network model showed AUROC of 0.9491647 and AUPRC of 0.9487003.

Model Evaluation and Comparison

Next, I'll check how well the four machine-learning models work. I'll use plots to show the results for each model, looking at how good they are at finding fraudulent transactions (AUPRC) and how well they can tell apart real and fake transactions (AUROC). These plots will help me see which model is best at finding fraud and how good it is at not mistaking genuine transactions for fake ones. By looking at these numbers, I'll find the model that's best at detecting fraud.

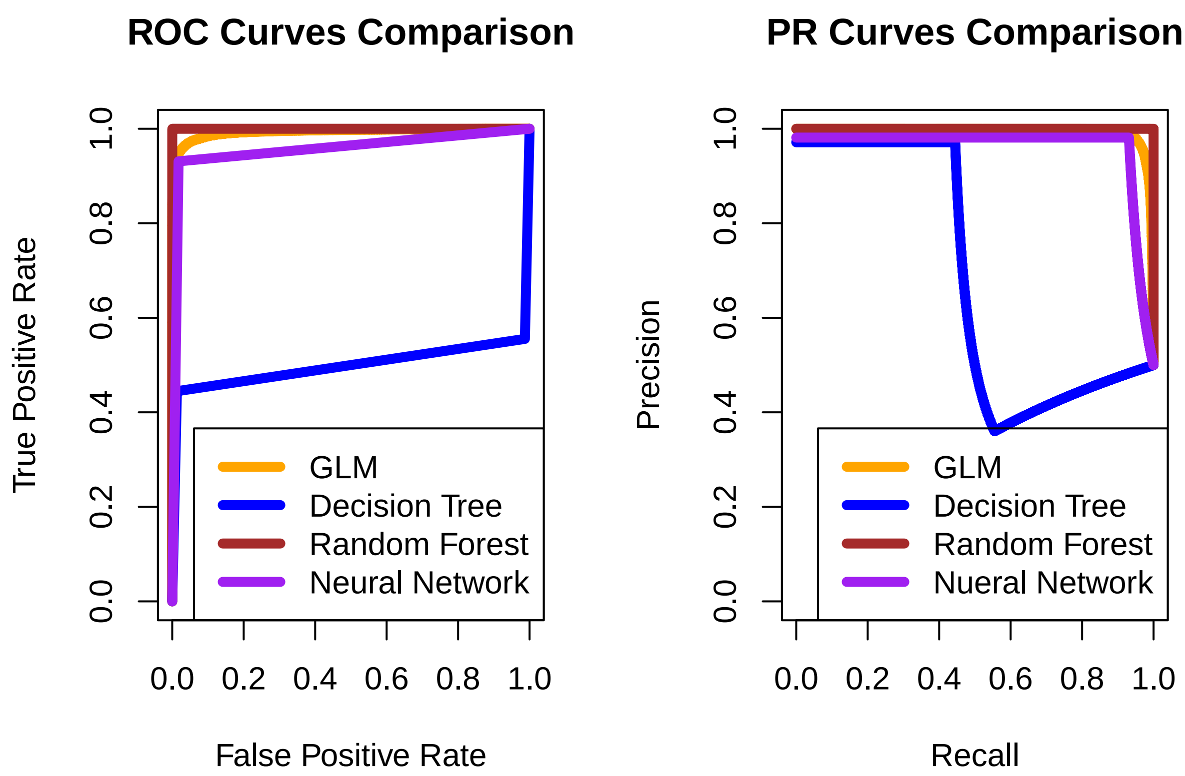
```r
par(mfrow = c(1, 2))

# Extract ROC data for all models
roc_data_glm <- data.frame(FPR = glm_roc$curve[, 1], TPR = glm_roc$curve[, 2])
roc_data_dt <- data.frame(FPR = dt_roc$curve[, 1], TPR = dt_roc$curve[, 2])
roc_data_rf <- data.frame(FPR = rf_roc$curve[, 1], TPR = rf_roc$curve[, 2])
roc_data_nn <- data.frame(FPR = nn_roc$curve[, 1], TPR = nn_roc$curve[, 2])

# Plot ROC curves
plot(roc_data_glm$FPR, roc_data_glm$TPR, type = 'l', col = "orange", lwd = 5, xlab = "Fa
lines(roc_data_dt$FPR, roc_data_dt$TPR, col = "blue", lwd = 5)
lines(roc_data_rf$FPR, roc_data_rf$TPR, col = "brown", lwd = 5)
lines(roc_data_nn$FPR, roc_data_nn$TPR, col = "purple", lwd = 5)
legend("bottomright", legend = c("GLM", "Decision Tree", "Random Forest", "Neural Networ

# Extract PR data for all models
pr_data_glm <- data.frame(Recall = glm_pr$curve[, 1], Precision = glm_pr$curve[, 2])
pr_data_dt <- data.frame(Recall = dt_pr$curve[, 1], Precision = dt_pr$curve[, 2])
pr_data_rf <- data.frame(Recall = rf_pr$curve[, 1], Precision = rf_pr$curve[, 2])
pr_data_nn <- data.frame(Recall = nn_pr$curve[, 1], Precision = nn_pr$curve[, 2])

# Plot PR curves
plot(pr_data_glm$Recall, pr_data_glm$Precision, type = 'l', col = "orange", , lwd = 5, x
lines(pr_data_dt$Recall, pr_data_dt$Precision, col = "blue", lwd = 5)
lines(pr_data_rf$Recall, pr_data_rf$Precision, col = "brown", lwd = 5)
lines(pr_data_nn$Recall, pr_data_nn$Precision, col = "purple", lwd = 5)
legend("bottomright", legend = c("GLM", "Decision Tree", "Random Forest", "Nueral Networ
```




```
# Reset the plotting parameters to default
par(mfrow = c(1, 1))
```

```
# Extre AUC ROC values
auc_roc_glm <- glm_roc$auc
auc_roc_dt <- dt_roc$auc
auc_roc_rf <- rf_roc$auc
auc_roc_nn <- nn_roc$auc

# Extre AUC PR values
auc_pr_glm <- glm_pr$auc.integral
auc_pr_dt <- dt_pr$auc.integral
auc_pr_rf <- rf_pr$auc.integral
auc_pr_nn <- nn_pr$auc.integral

# Create a data frame
auc_table <- data.frame(
  Model = c("GLM", "Decision Tree", "Random Forest", "Neural Network"),
  AUC_ROC = c(auc_roc_glm, auc_roc_dt, auc_roc_rf, auc_roc_nn),
  AUC_PR = c(auc_pr_glm, auc_pr_dt, auc_pr_rf, auc_pr_nn)
)
```

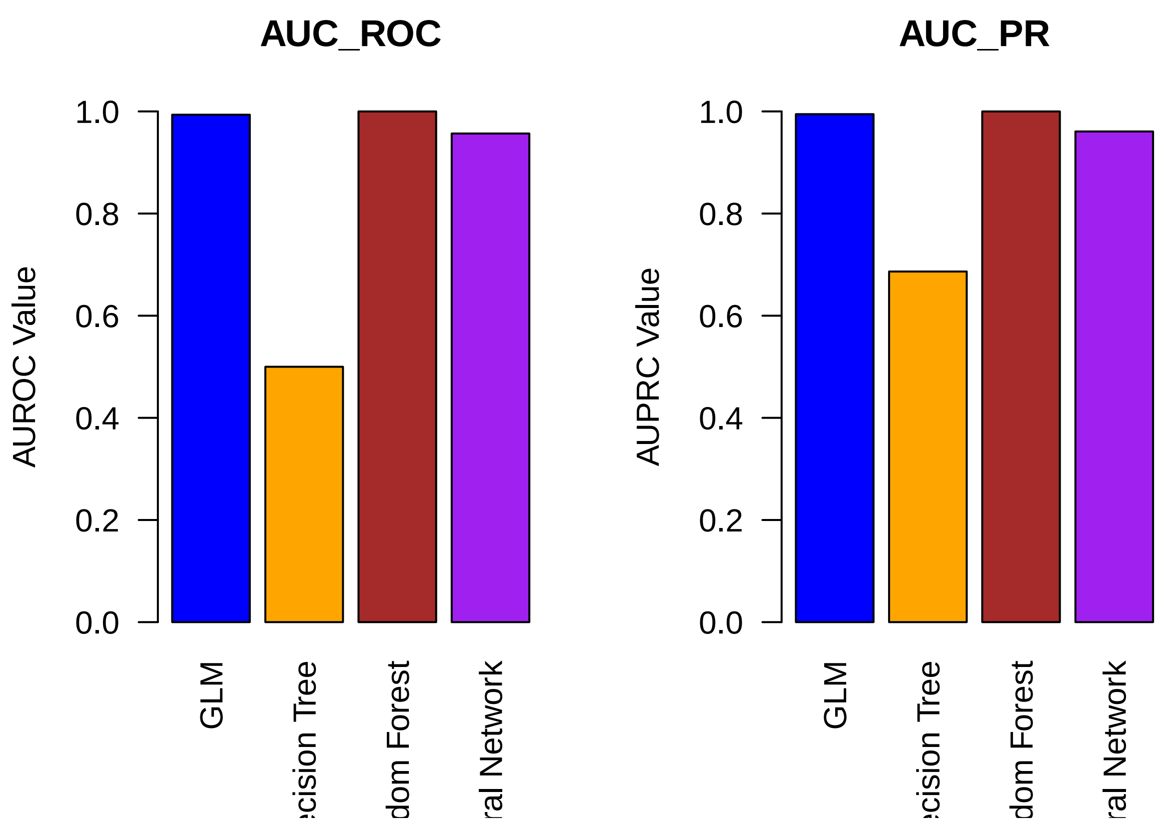
```r
# Print the table
print(auc_table)
```

```
##             Model   AUC_ROC    AUC_PR
## 1             GLM 0.9935217 0.9946013
## 2   Decision Tree 0.5000000 0.6864879
## 3   Random Forest 0.9999697 0.9999397
## 4 Neural Network 0.9567208 0.9607424
```

```r
# Create the bar plot for the results
par(mfrow = c(1, 2))

# Create bar plot for AUROC
barplot(auc_table$AUC_ROC, names.arg = auc_table$Model, col = c("blue", "orange", "brown

# Create bar plot for AUPRC
barplot(auc_table$AUC_PR, names.arg = auc_table$Model, col = c("blue", "orange", "brown'
```



```r
# Reset graphical parameters to default
par(mfrow = c(1, 1))
```

Conclusion

We wanted to find the best models for detecting credit card fraud using a dataset with fraudulent and genuine transactions. I developed four machine learning models: Logistic Regression, Decision Trees, Random Forest, and Neural Network.

The Random Forest model was almost perfect at telling apart genuine and fraudulent transactions, scoring 0.9999996. Logistic Regression was also perfect, with a score of 0.9935217, followed closely by Neural Network at 0.9491647. Decision Tree could have done better, with a score of 0.5, which means it struggled to differentiate between real and fake transactions.

When we looked at how well the models handled imbalanced data, Random Forest again did exceptionally well, scoring 0.9999995, showing it's very effective at detecting fraud. Logistic Regression was also robust, with a score of 0.9946013. Neural Network was decent, scoring 0.9487003, while Decision Tree struggled again, with a score of 0.6864879.

Overall, Random Forest was the most accurate model.