Credit Card Fraud

Lahmamssi Abdelhak

Contents

1 Intrduction

Credit card fraud poses a significant risk to both financial institutions and their customers, leading to not only financial losses but also damaging the trust and reputation of banks. With the rise in online transactions, the urgency for effective real-time fraud detection systems has increased markedly.Given these challenges, machine learning (ML) offers a promising avenue for enhancing fraud detection by leveraging its ability to discern patterns and anomalies in large datasets.

Using the dataset available in "https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud", we will analyze the challenges associated with credit card fraud detection and evaluate the effectiveness of machine learning techniques in addressing these issues.

This report aims to analyze the given dataset, and to demonstrate how well machine learning can identify fraudulent transactions and contribute to the development of a robust fraud detection system.

2 Importing libraries and dataset

```
library(readr)
library(dplyr)
library(ggplot2)
library(scales)
library(tidyverse)
library(corrplot)
library(ROSE)
library(naivebayes)
library(ggplot2)
library(caret)
library(ROCR)
library(caret)
```

```
credit_card_data <- read_csv("creditcard.csv")
```

```
## Rows: 284807 Columns: 31
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (31): Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14,...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

3 Data Exploration and analysis

3.1 Data exploration

```
 dim(credit_card_data)
```

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

The dataframe contains 284,807 observations of 30 variables that we will use to model the Class variable.

```
head(credit_card_data)
```

```
## # A tibble: 6 x 31
##    Time     V1      V2    V3     V4      V5      V6      V7      V8     V9
##   <dbl>  <dbl>   <dbl> <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
## 1     0 -1.36  -0.0728 2.54   1.38  -0.338   0.462   0.240   0.0987  0.364
## 2     0  1.19   0.266  0.166  0.448  0.0600 -0.0824 -0.0788  0.0851 -0.255
## 3     1 -1.36  -1.34   1.77   0.380 -0.503   1.80    0.791   0.248  -1.51
## 4     1 -0.966 -0.185  1.79  -0.863 -0.0103  1.25    0.238   0.377  -1.39
## 5     2 -1.16   0.878  1.55   0.403 -0.407   0.0959  0.593  -0.271   0.818
## 6     2 -0.426  0.961  1.14  -0.168  0.421  -0.0297  0.476   0.260  -0.569
## # i 21 more variables: V10 <dbl>, V11 <dbl>, V12 <dbl>, V13 <dbl>, V14 <dbl>,
## #   V15 <dbl>, V16 <dbl>, V17 <dbl>, V18 <dbl>, V19 <dbl>, V20 <dbl>,
## #   V21 <dbl>, V22 <dbl>, V23 <dbl>, V24 <dbl>, V25 <dbl>, V26 <dbl>,
## #   V27 <dbl>, V28 <dbl>, Amount <dbl>, Class <dbl>
```

The majority of the variables are the result of PCA transformation, which limits the information available from most features. However, the "Time," "Amount," and "Class" features remain in their original form and can provide valuable insights into the dataset.

```
summary(credit_card_data[, c("Class", "Time", "Amount")])
```

```
##      Class               Time            Amount
##  Min.   :0.000000   Min.   :     0   Min.   :    0.00
##  1st Qu.:0.000000   1st Qu.: 54202   1st Qu.:    5.60
##  Median :0.000000   Median : 84692   Median :   22.00
##  Mean   :0.001728   Mean   : 94814   Mean   :   88.35
##  3rd Qu.:0.000000   3rd Qu.:139321   3rd Qu.:   77.17
##  Max.   :1.000000   Max.   :172792   Max.   :25691.16
```

We can see that the `Class` variable is considered numeric, so we need to change it to a categorical variable.

```
credit_card_data$Class <- as.factor(credit_card_data$Class)
summary(credit_card_data[, c("Class")])
```

```
##  Class
##  0:284315
##  1:   492
```

The Time variable is expressed in seconds; we will change it to hours.

```
credit_card_data$Time <- credit_card_data$Time  / 3600
summary(credit_card_data$Time)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   15.06   23.53   26.34   38.70   48.00
```

We check for missing values in the dataset and found none.

```
colSums(is.na(credit_card_data))
```

```
##   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
```

```
sum(duplicated(credit_card_data))
```

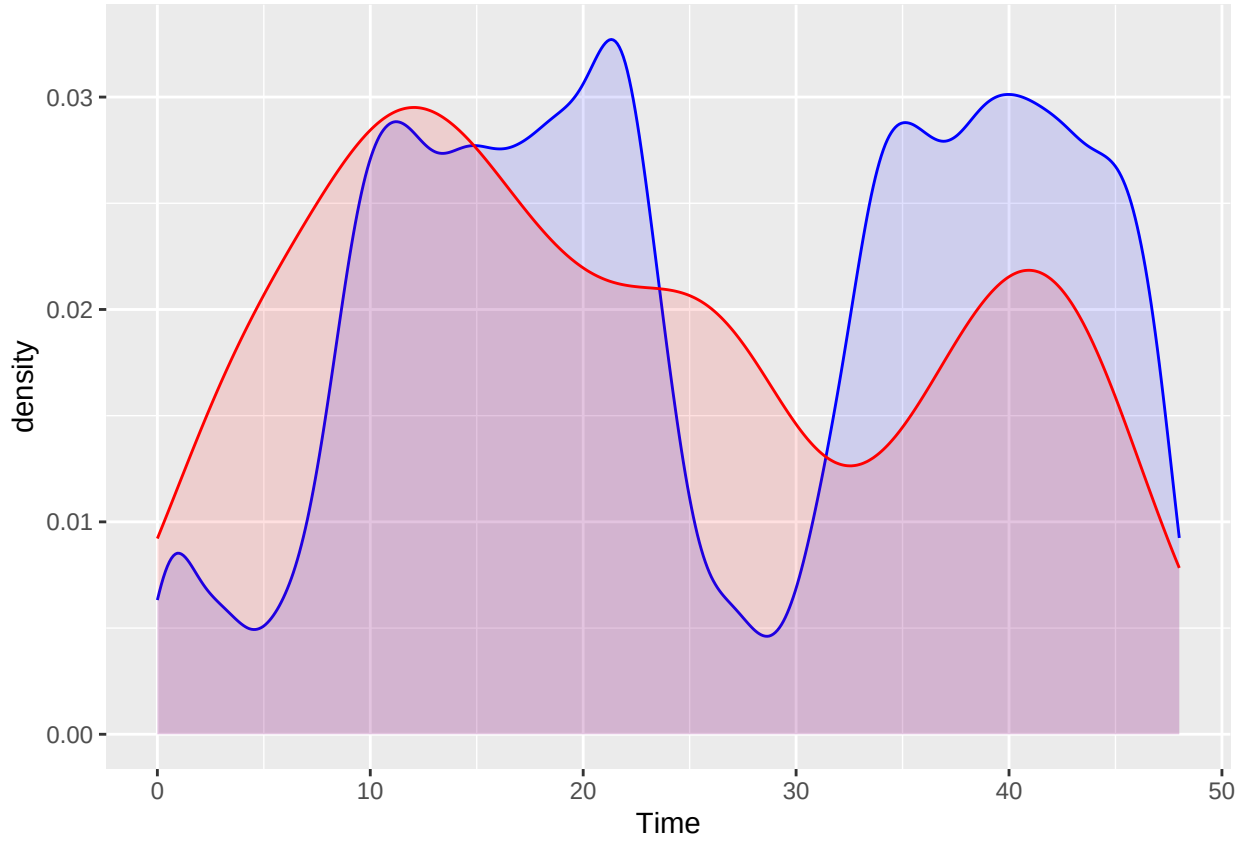
```
## [1] 1081
```

```
credit_card_data <- distinct(credit_card_data)
```

There are 1,081 duplicate rows in the dataset, we remove all duplicate rows.

3.2 Time Variable Analysis

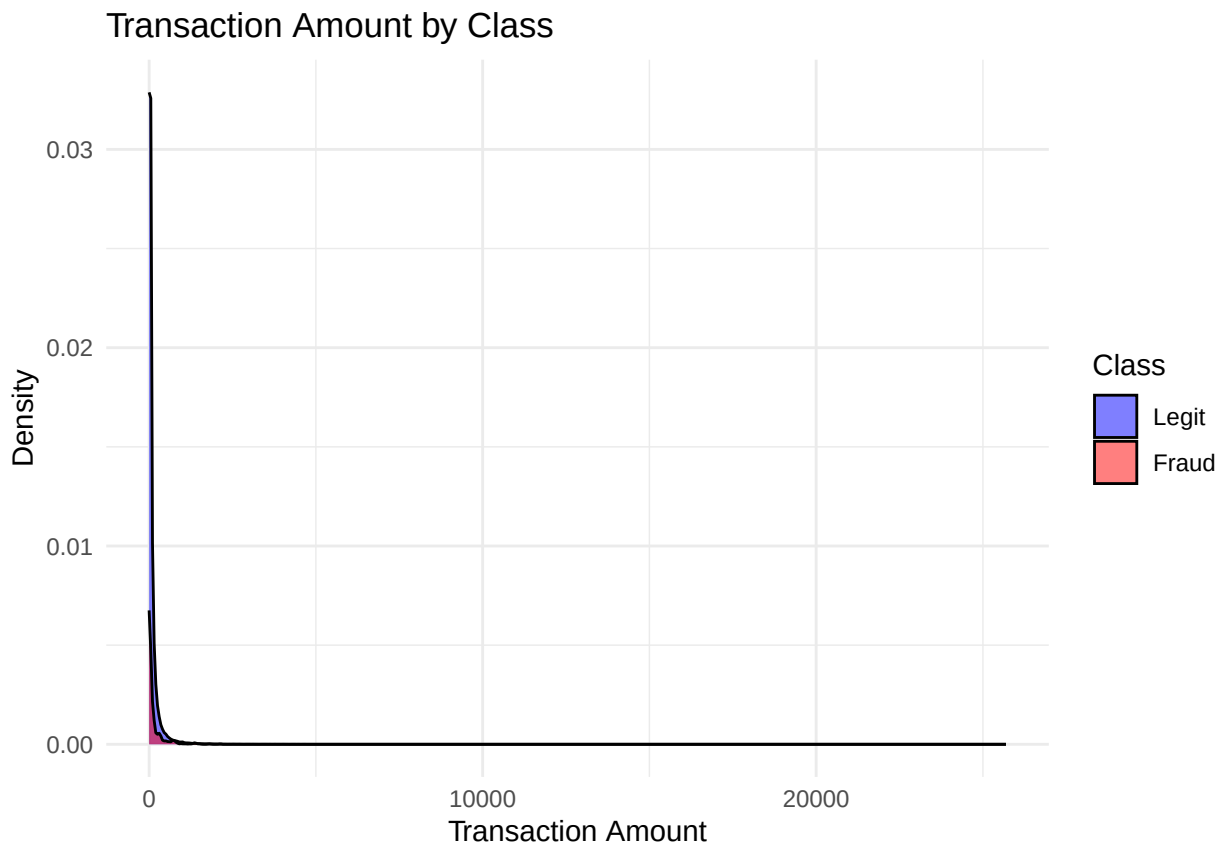
```r
credit_card.true = credit_card_data[credit_card_data$Class == "0",]
credit_card.false = credit_card_data[credit_card_data$Class == "1",]

ggplot() +
  geom_density(data = credit_card.true, aes(x = Time), color = "blue", fill = "blue", alpha = 0.12) +
  geom_density(data = credit_card.false, aes(x = Time), color = "red", fill = "red", alpha = 0.12)
```



The density of fraud over a 2-day period reveals significant peaks at hours 11-12 (surpassing the density of legitimate transactions) and 41. This pattern suggests specific time windows where fraudulent activity is more concentrated, possibly indicating targeted attack periods or vulnerabilities.

3.3 Amount Variable Analysis

```r
ggplot(credit_card_data, aes(x = Amount, fill = factor(Class))) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("0" = "blue", "1" = "red"), labels = c("Legit", "Fraud")) +
  labs(x = "Transaction Amount", y = "Density", fill = "Class", title = "Transaction Amount by Class") +
  theme_minimal()
```

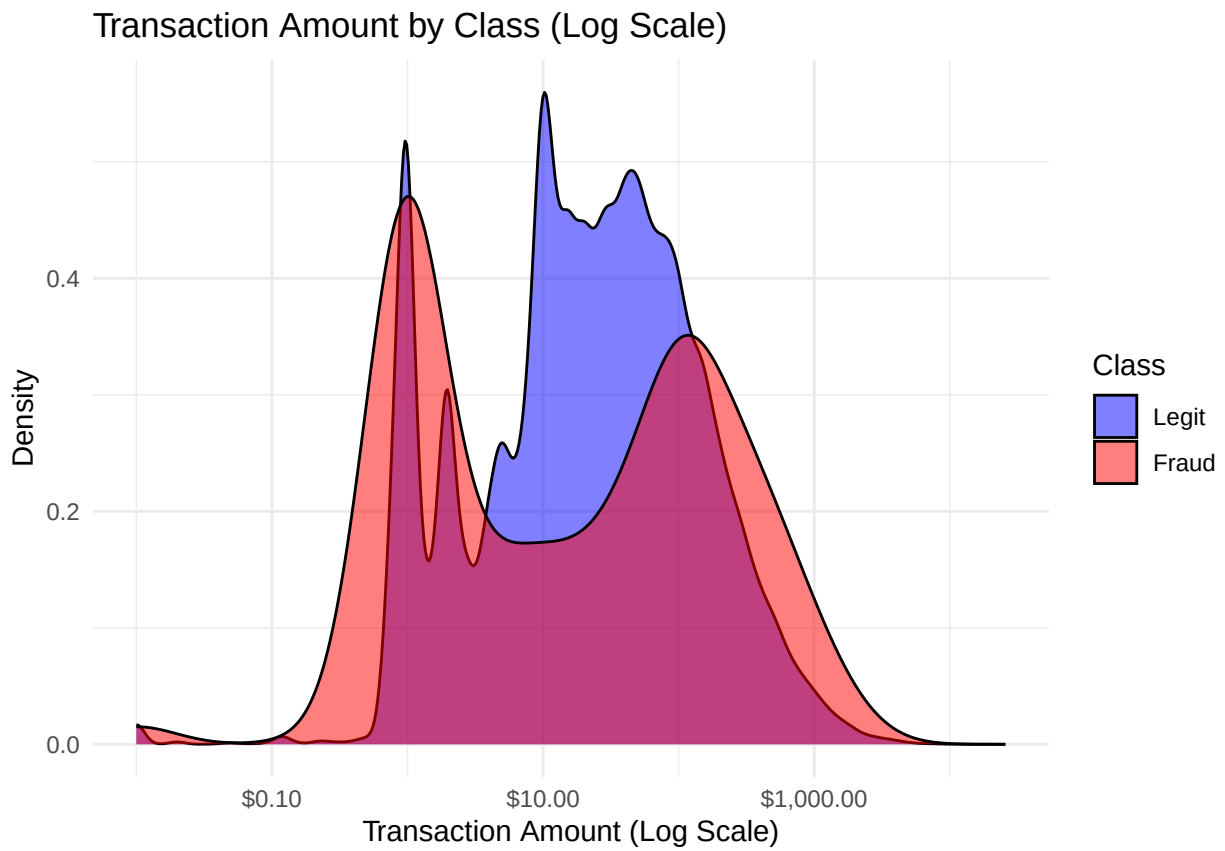


To better visualize the density of the "Amount" variable, we perform a log transformation.

```
ggplot(credit_card_data, aes(x = Amount, fill = factor(Class))) +
  geom_density(alpha = 0.5) +
  scale_x_log10(labels = scales::dollar) +
  scale_fill_manual(values = c("0" = "blue", "1" = "red"), labels = c("Legit", "Fraud")) +
  labs(x = "Transaction Amount (Log Scale)", y = "Density", fill = "Class", title = "Transaction Amount
  theme_minimal()
```

```
## Warning in scale_x_log10(labels = scales::dollar): log-10 transformation
## introduced infinite values.
```

```
## Warning: Removed 1808 rows containing non-finite outside the scale range
## (`stat_density()`).
```

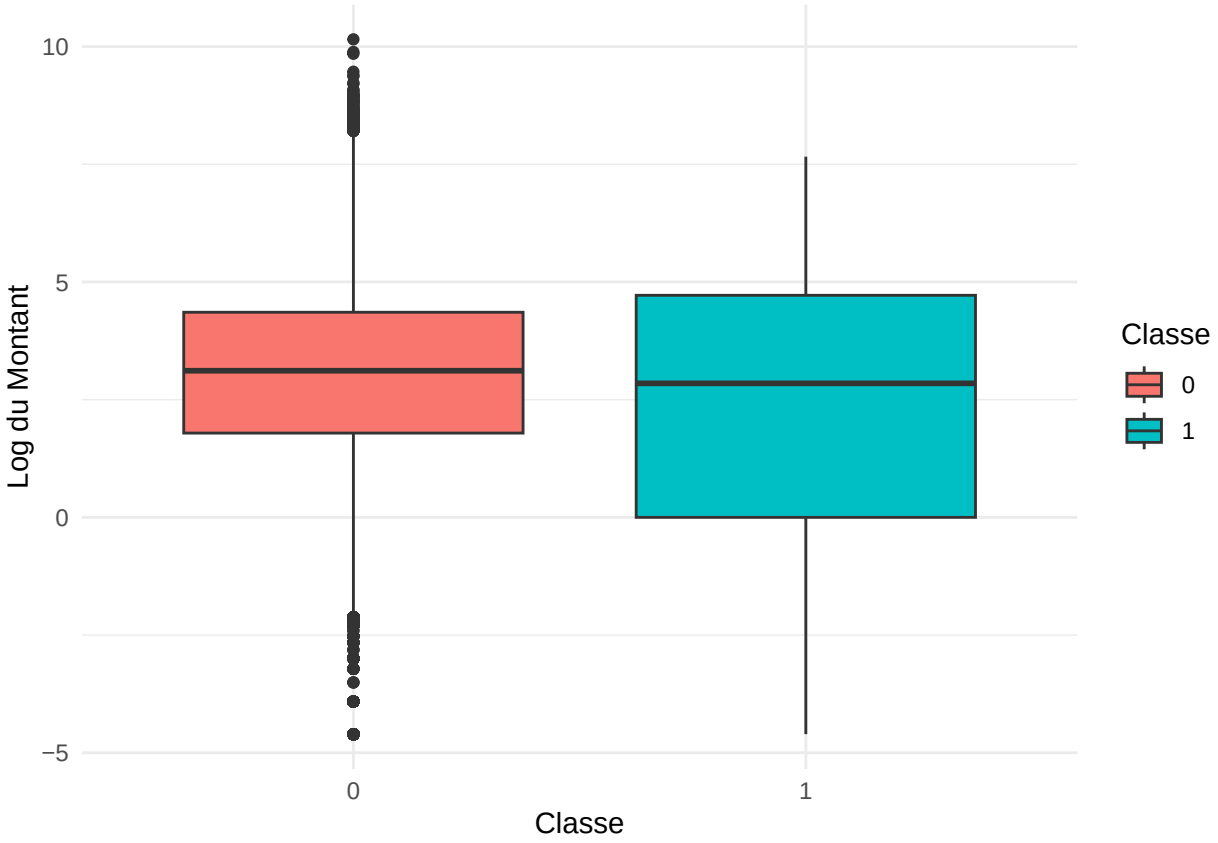


We can see that the amount density of frauds peaks at lower levels and higher levels, which means that the fraudulent transactions tend to cluster around smaller and larger amounts, indicating a potential strategy by fraudsters to exploit both ends of the spectrum.

```
ggplot(credit_card_data, aes(x = factor(Class), y = log(credit_card_data$Amount), fill = factor(Class)))
  geom_boxplot() +
  labs(x = "Classe", y = "Log du Montant") +
  scale_fill_discrete(name = "Classe") +
  theme_minimal()
```

```
## Warning: Use of ‘credit_card_data$Amount‘ is discouraged.
## i Use ‘Amount‘ instead.
```

```
## Warning: Removed 1808 rows containing non-finite outside the scale range
## (‘stat_boxplot()‘).
```



From the boxplot, we observe that both legitimate and fraudulent transactions have approximately the same mean, but legitimate transactions exhibit more outliers. This indicates that while the central tendency of transaction amounts is similar for both groups, legitimate transactions show greater variability, with more extreme values compared to fraudulent ones.

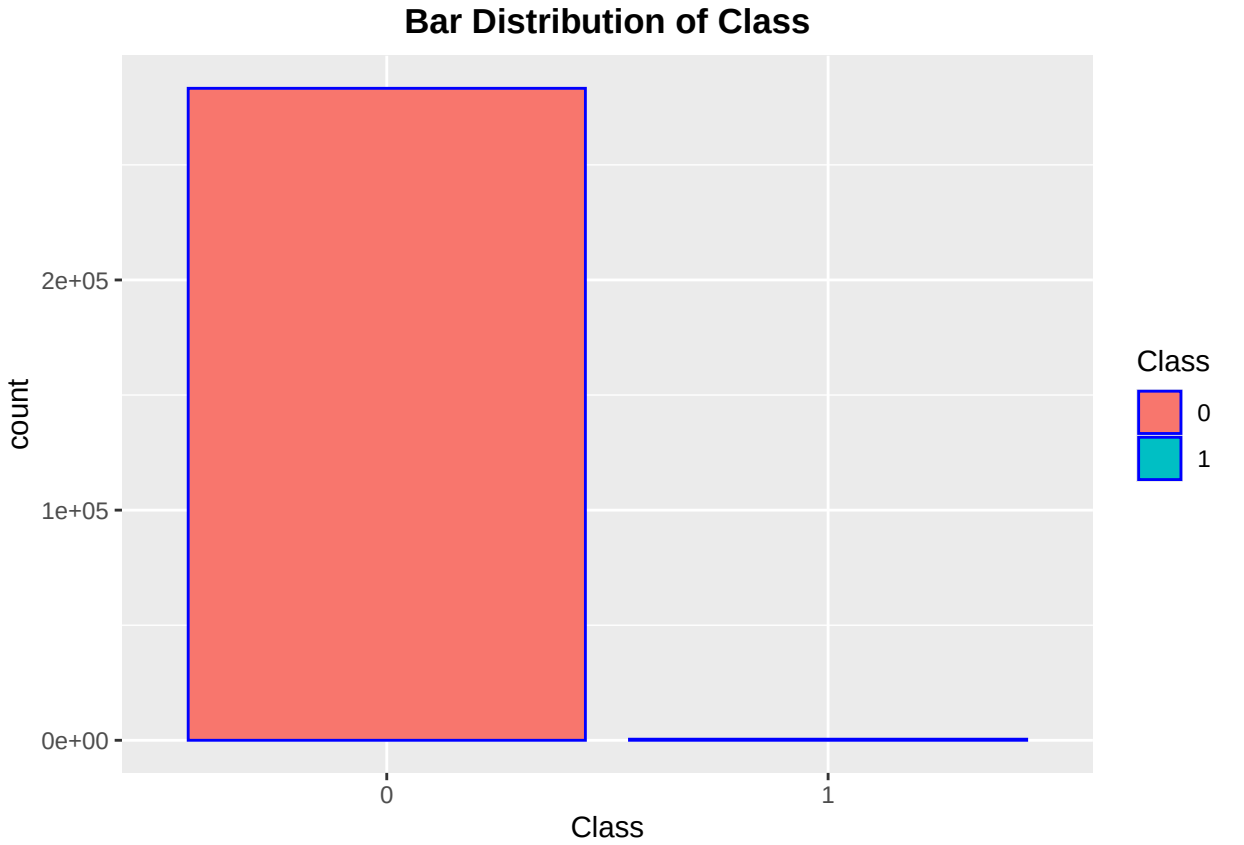
In order to ensure fair comparison and prevent bias in our analysis, we perform scaling on the variable "Amount".

```
credit_card_data$Amount <- scale(credit_card_data$Amount)

summary(credit_card_data$Amount)
```

```
##        V1
##  Min.   : -0.35333
##  1st Qu.: -0.33096
##  Median : -0.26547
##  Mean   :  0.00000
##  3rd Qu.: -0.04378
##  Max.   :102.24738
```

3.4 Class Variable Analysis

```
ggplot(credit_card_data, aes(x=Class, fill = Class)) + geom_bar(color = "blue") +
  ggtitle("Bar Distribution of Class") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```



In our dataset, we observe a significant class imbalance, with fraud being notably lower compared to non-fraudulent transactions.

```
counts <- table(credit_card_data$Class)
result <- data.frame(table(credit_card_data$Class), round(prop.table(counts), 5))
noms <- c("Class", "Value", "none", "Proportion")

names(result) <- noms

print(result[c(1,2,4)])
```

```
##   Class  Value Proportion
## 1     0 283253    0.99833
## 2     1    473    0.00167
```

In order to balance our data, we need to resample our dataset which we ll do in the next chapter.

4 Data Manipulation

4.1 Split data to test and train sets

By splitting data into training and test sets helps evaluate model performance objectively by preventing overfitting and ensuring the model generalizes well to new, unseen data. This practice provides an unbiased estimate of model accuracy and aids in hyperparameter tuning and model validation.

```r
set.seed(123)

indices <- createDataPartition(credit_card_data$Class, p=0.8, list = F)
trainData <- credit_card_data[indices,]
testData <- credit_card_data[-indices,]
```

4.2 Resampling

To address class imbalance in a dataset, we can use techniques such as upsampling and downsampling. Upsampling involves increasing the number of instances in the minority class by replicating existing instances or generating new ones using methods like SMOTE (Synthetic Minority Over-sampling Technique). This helps ensure that the minority class is better represented in the training process. Conversely, downsampling reduces the number of instances in the majority class by randomly removing some of them, which balances the dataset by making the majority class less dominant.

We might choose upsampling when we want to retain all available data from the majority class, as this preserves potentially valuable information that could be lost through downsampling. Upsampling can also be particularly useful when the minority class contains critical or rare events that we need the model to recognize effectively. By enhancing the representation of the minority class, upsampling improves the model's ability to detect and classify minority class instances accurately, leading to better overall performance in imbalanced datasets.

```r
set.seed(9560)
up_train <- upSample(x = trainData[, -ncol(trainData)],
                     y = trainData$Class)
table(up_train$Class)
```

```
##
##      0      1
## 226603 226603
```

5 Data Modelling

5.1 Logistic Regression

5.1.1 Model implementation

Logistic regression is a widely used statistical method for binary classification problems. It models the probability that a given input belongs to a particular class by fitting a logistic function to the data. Logistic regression is interpretable and efficient, making it suitable for problems where understanding the relationship between features and the outcome is crucial. It provides probabilistic outputs and a straightforward decision boundary, which is particularly useful in our context for gaining insights into the factors influencing the outcome.

```r
model_lr <- glm(Class ~ ., data = up_train, family = "binomial")
```

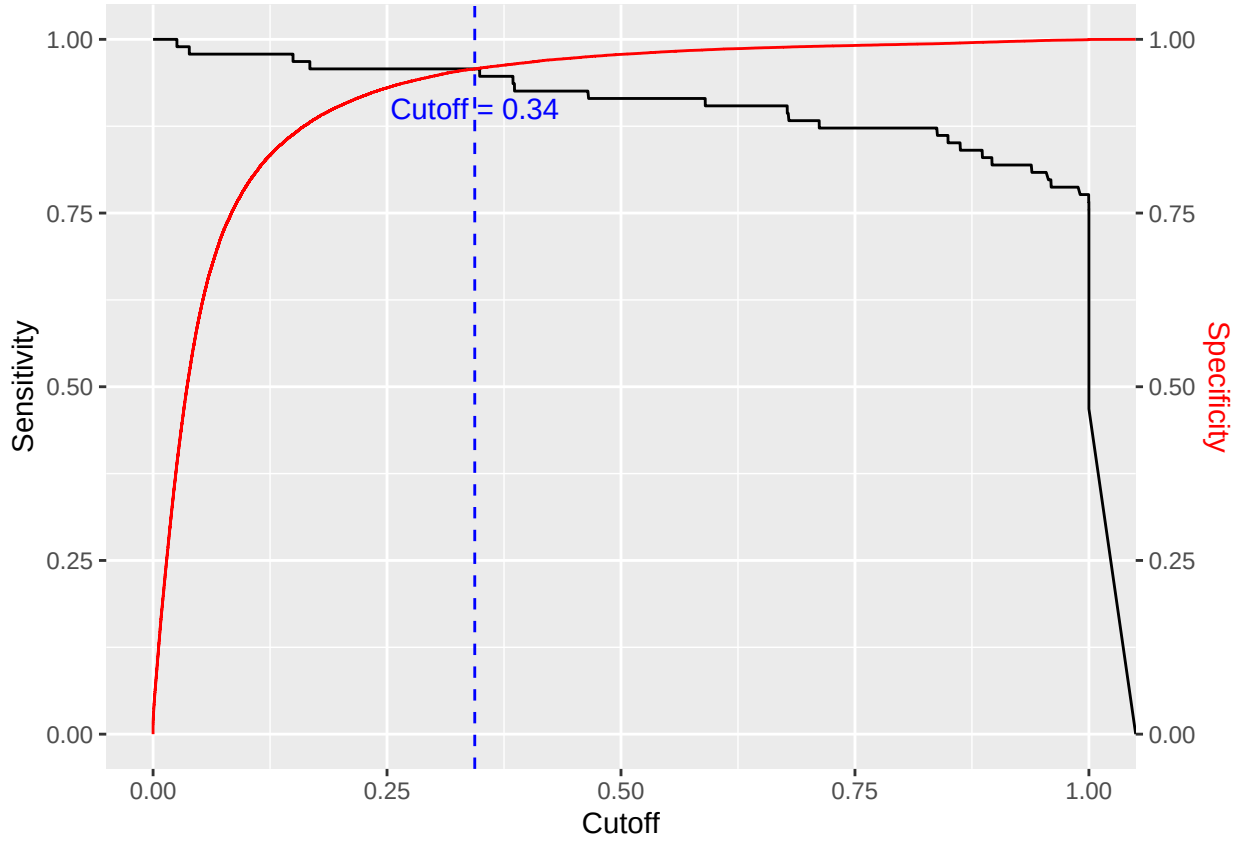
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
testData$Amount=as.numeric(testData$Amount)
probabilities <- model_lr %>% predict(testData, type = "response")
```

5.1.2 Optimal Threshold

To obtain the optimal threshold in logistic regression, we calculate sensitivity and specificity across different cutoffs, and identify the point where these metrics intersect (i.e., where sensitivity and specificity are balanced).

```r
# Calculate sensitivity and specificity across different cutoffs
perf_obj <- performance(prediction(probabilities, testData$Class), measure = "tpr", x.measure = "fpr")


# Extract cutoff values
cutoffs <- perf_obj@alpha.values[[1]]
sens_values <- perf_obj@y.values[[1]]
spec_values <- 1 - perf_obj@x.values[[1]]

# Create data frames for sensitivity and specificity
sens <- data.frame(x = cutoffs, y = sens_values)
spec <- data.frame(x = cutoffs, y = spec_values)


# Find the point where sensitivity and specificity curves meet
intersect_idx <- which.min(abs(sens$y - spec$y))
intersect_cutoff <- sens$x[intersect_idx]
intersect_sens <- sens$y[intersect_idx]
intersect_spec <- spec$y[intersect_idx]

# Plot sensitivity and specificity
ggplot() +
  geom_line(data = sens, aes(x = x, y = y), color = "black") +
  geom_line(data = spec, aes(x = x, y = y), color = "red") +
  geom_vline(xintercept = intersect_cutoff, linetype = "dashed", color = "blue") +
  annotate("text", x = intersect_cutoff, y = 0.9,
           label = paste("Cutoff =", round(intersect_cutoff, 2)), color = "blue") +
  scale_x_continuous(name = "Cutoff") +
  scale_y_continuous(name = "Sensitivity", sec.axis = sec_axis(~., name = "Specificity")) +
  theme(axis.title.y.right = element_text(color = "red"), legend.position = "none")
```



The optimal cutoff is 0,34. We run the model using the optimal threshold.

```
model_lr_prediction <- ifelse(probabilities > 0.34, "1", "0")
model_lr_prediction<-as.factor(model_lr_prediction)
```

5.2 Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' theorem, with the assumption of independence between every pair of features given the class label. Despite its simplicity and the strong assumption of feature independence, Naive Bayes often performs surprisingly well, especially in text classification and spam filtering. It is computationally efficient and requires a small amount of training data. In our context, Naive Bayes is relevant because it provides a robust baseline model and handles large datasets effectively, offering a different perspective compared to logistic regression.

```
NB <- naive_bayes(Class ~ ., data = up_train, laplace = 1)

model_nb_prediction <- predict(NB, newdata = testData, type = "class")
```

```
## Warning: predict.naive_bayes(): more features in the newdata are provided as
## there are probability tables in the object. Calculation is performed based on
## features to be found in the tables.
```

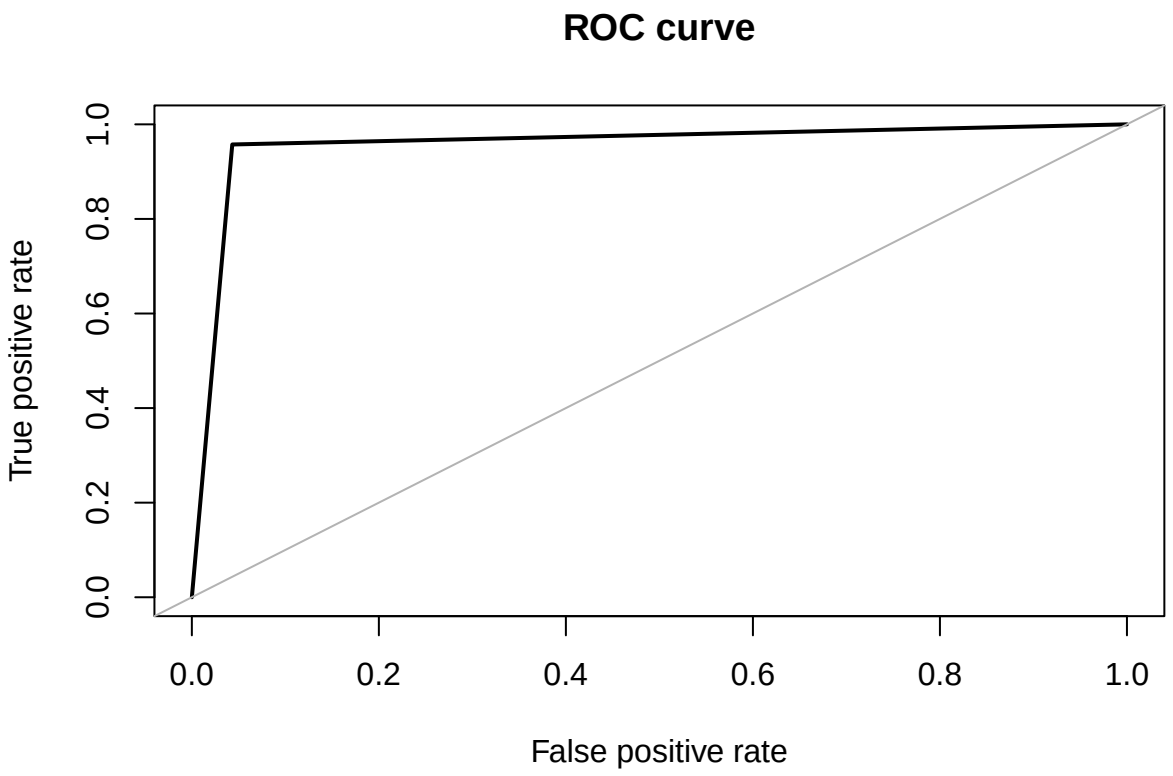
5.3 Model validation and comparaison

To validate and compare models, we use the metrics accuracy, precision, recall, and AUC, ensuring an unbiased assessment of each model's performance on a separate test set.

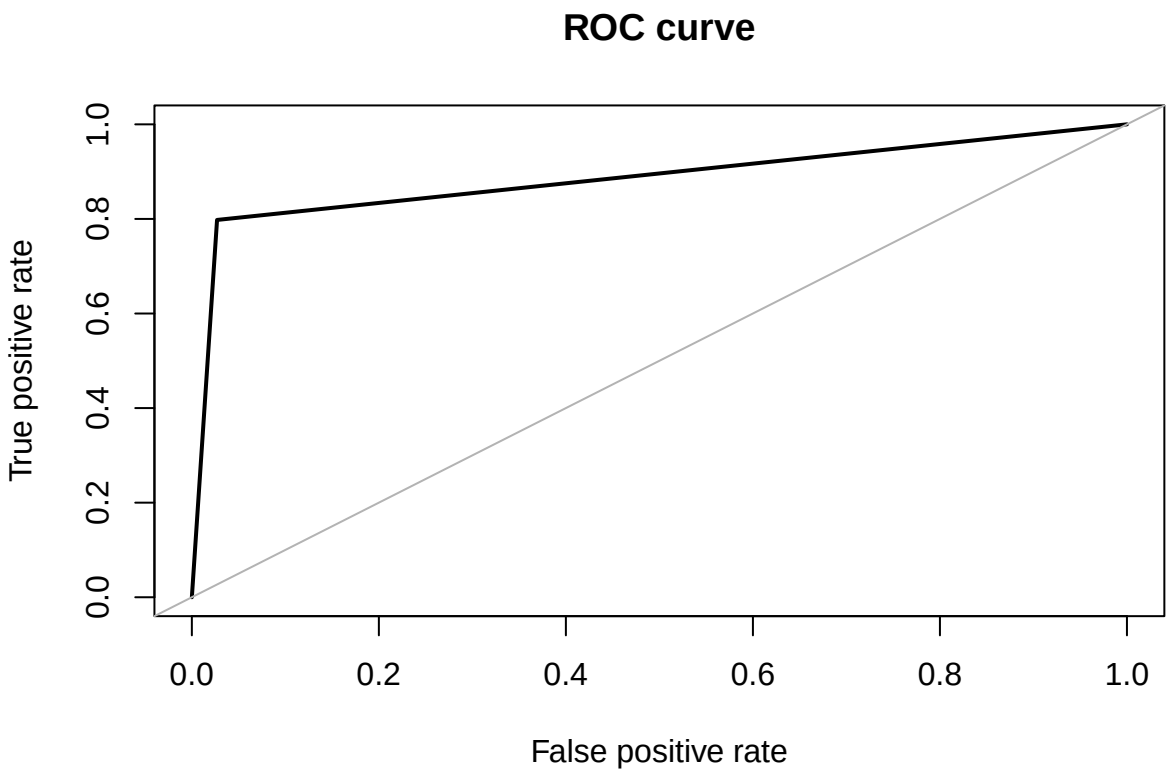
```
# Evaluate the models
confusion_lr <- confusionMatrix(data = factor(model_lr_prediction, levels = levels(testData$Class)), re
confusion_nb <- confusionMatrix(data = model_nb_prediction, reference = testData$Class)

# Compare performance
comparison_table <- data.frame(
  Model = c("Logistic Regression", "Naive Bayes"),
  Accuracy = c(confusion_lr$overall['Accuracy'], confusion_nb$overall['Accuracy']),
  Precision = c(confusion_lr$byClass['Pos Pred Value'], confusion_nb$byClass['Pos Pred Value']),
  Recall = c(confusion_lr$byClass['Sensitivity'], confusion_nb$byClass['Sensitivity']),
  F1_Score = c(2 * (confusion_lr$byClass['Pos Pred Value'] * confusion_lr$byClass['Sensitivity']) / (co
               2 * (confusion_nb$byClass['Pos Pred Value'] * confusion_nb$byClass['Sensitivity']) / (co
)

print(comparison_table)
```

```
##                 Model  Accuracy Precision    Recall  F1_Score
## 1 Logistic Regression 0.9567708 0.9999262 0.9567696 0.9778720
## 2         Naive Bayes 0.9727548 0.9996554 0.9730450 0.9861707
```

```
# ROC for Logistic Regression

roc_lr<-roc.curve(testData$Class, model_lr_prediction, plotit = TRUE)
```




```
# ROC  for Naive Bayes
roc_nb <- roc.curve(testData$Class, model_nb_prediction, plotit = TRUE)
```



We can see that the Area in the logistic regression model is wider than the Naive Bayes.

```
# Add AUC to the comparison table
comparison_table$AUC <- c(roc_lr$auc, roc_nb$auc)
print(comparison_table)
```

```
##                 Model  Accuracy Precision    Recall  F1_Score       AUC
## 1 Logistic Regression 0.9567708 0.9999262 0.9567696 0.9778720 0.9571082
## 2         Naive Bayes 0.9727548 0.9996554 0.9730450 0.9861707 0.8854587
```

The Naive Bayes model demonstrates higher accuracy, recall, and F1 Score, suggesting it is better at correctly identifying fraudulent transactions. On the other hand, Logistic Regression exhibits higher precision and AUC, indicating it may be more effective at distinguishing between fraudulent and non-fraudulent transactions, crucial for minimizing false positives in credit card fraud detection.

The choice between them should consider the priority of correctly identifying fraudulent transactions versus minimizing false alarms. Therefore, in this specific context, the Naive Bayes model might be considered the better choice.

6 Conclusion

This project aimed to develop and assess machine learning models for credit card fraud detection. While we extensively compared the performance of Logistic Regression and Naive Bayes models, our reliance on PCA-derived variables limited our data understanding and variable selection process.

Addressing imbalanced data required the use of resampling methods, yet we acknowledge the availability of more advanced techniques.

Looking ahead, our project highlights the need to explore advanced model selection approaches and leverage more powerful machine learning and deep learning methods to enhance fraud detection accuracy further.

‘‘‘