Credit card fraud detection

Jyothi

2025-03-15

```r
library(dplyr)
library(tidyverse)
library(janitor)
library(xgboost)
library(dplyr)
library(ggplot2)
library(lubridate)
library(precrec)
library(caret)
```

1. According to the Nilson report, credit cards transaction frauds will grow upto $400 billion in the next decade. Credit card fraud detection includes various anomaly detection, outlier modeling and predictive modeling to identify fraudulent transactions. Imbalanced data, with rare fraud transactions, evolving fraud techniques are few of the many challenges the credit card industry faces in fraud identification.

2. The data set used in this analysis has been obtained from the Kaggle credit card fraud detection data. The observations include anonymized transactions from 2013 European cardholders.

3.

```r
#Uploading the dataset onto R environment
hw_data <- read.csv("C:\\Users\\jyoth\\Downloads\\archive\\creditcard.csv")
```

4. The data set contains transactions made by credit cards, and they are labelled as fraudulent or genuine. The columns include PCA values obtained from the original data inorder to reduce the dimensionality of the data. There are a total of 28 PCA columns with a mean of 0. The data also includes a time column, amount column and a class column that has information on whether the observation is fraud or genuine. The dataset has 284,807 rows and 31 columns in total, with no missing values. The PCA columns are labelled from V1 to V28.

```r
summary(hw_data)
```

```
##       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
```

5. The data variables are converted to PCA columns due to privacy restrictions. This helps in reducing the dimensionality of the data, but it also poses the challenge of only being able to interpret the analysis results towards the respective PCA columns, and not the actual variables. The time column in the dataset includes the number of seconds passed since the first transaction in the dataset. For meaningful interpretation of this information, the date variable has been converted to date and time, assuming the first transaction happened on 01-01-2013, at 00:00:00. The trends of the genuine vs fraud transactions by hour of the day has been plotted. The results indicate that, while the genuine transactions exhibit higher frequencies during the day, the frequency of fraud transactions remain more or less constant during all hours of the day.

```r
str(hw_data$Time)
```

```
##  int [1:284807] 0 0 1 1 2 2 4 7 7 9 ...
```

```r
glimpse(hw_data)
```

```
## Rows: 284,807
## Columns: 31
## $ Time   <int> 0, 0, 1, 1, 2, 2, 4, 7, 7, 9, 10, 10, 10, 11, 12, 12, 12, 13, 1~
## $ V1     <dbl> -1.3598071, 1.1918571, -1.3583541, -0.9662717, -1.1582331, -0.4~
## $ V2     <dbl> -0.07278117, 0.26615071, -1.34016307, -0.18522601, 0.87773676, ~
## $ V3     <dbl> 2.53634674, 0.16648011, 1.77320934, 1.79299334, 1.54871785, 1.1~
## $ V4     <dbl> 1.37815522, 0.44815408, 0.37977959, -0.86329128, 0.40303393, -0~
## $ V5     <dbl> -0.33832077, 0.06001765, -0.50319813, -0.01030888, -0.40719338,~
## $ V6     <dbl> 0.46238778, -0.08236081, 1.80049938, 1.24720317, 0.09592146, -0~
## $ V7     <dbl> 0.239598554, -0.078802983, 0.791460956, 0.237608940, 0.59294074~
## $ V8     <dbl> 0.098697901, 0.085101655, 0.247675787, 0.377435875, -0.27053267~
## $ V9     <dbl> 0.3637870, -0.2554251, -1.5146543, -1.3870241, 0.8177393, -0.56~
## $ V10    <dbl> 0.09079417, -0.16697441, 0.20764287, -0.05495192, 0.75307443, -~
## $ V11    <dbl> -0.55159953, 1.61272666, 0.62450146, -0.22648726, -0.82284288, ~
## $ V12    <dbl> -0.61780086, 1.06523531, 0.06608369, 0.17822823, 0.53819555, 0.~
## $ V13    <dbl> -0.99138985, 0.48909502, 0.71729273, 0.50775687, 1.34585159, -0~
## $ V14    <dbl> -0.31116935, -0.14377230, -0.16594592, -0.28792375, -1.11966984~
## $ V15    <dbl> 1.468176972, 0.635558093, 2.345864949, -0.631418118, 0.17512113~
## $ V16    <dbl> -0.47040053, 0.46391704, -2.89008319, -1.05964725, -0.45144918,~
## $ V17    <dbl> 0.207971242, -0.114804663, 1.109969379, -0.684092786, -0.237033~
## $ V18    <dbl> 0.02579058, -0.18336127, -0.12135931, 1.96577500, -0.03819479, ~
## $ V19    <dbl> 0.40399296, -0.14578304, -2.26185709, -1.23262197, 0.80348692, ~
## $ V20    <dbl> 0.25141210, -0.06908314, 0.52497973, -0.20803778, 0.40854236, 0~
## $ V21    <dbl> -0.018306778, -0.225775248, 0.247998153, -0.108300452, -0.00943~
## $ V22    <dbl> 0.277837576, -0.638671953, 0.771679402, 0.005273597, 0.79827849~
## $ V23    <dbl> -0.110473910, 0.101288021, 0.909412262, -0.190320519, -0.137458~
## $ V24    <dbl> 0.06692808, -0.33984648, -0.68928096, -1.17557533, 0.14126698, ~
## $ V25    <dbl> 0.12853936, 0.16717040, -0.32764183, 0.64737603, -0.20600959, -~
## $ V26    <dbl> -0.18911484, 0.12589453, -0.13909657, -0.22192884, 0.50229222, ~
## $ V27    <dbl> 0.133558377, -0.008983099, -0.055352794, 0.062722849, 0.2194222~
## $ V28    <dbl> -0.021053053, 0.014724169, -0.059751841, 0.061457629, 0.2151531~
## $ Amount <dbl> 149.62, 2.69, 378.66, 123.50, 69.99, 3.67, 4.99, 40.80, 93.20, ~
## $ Class  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

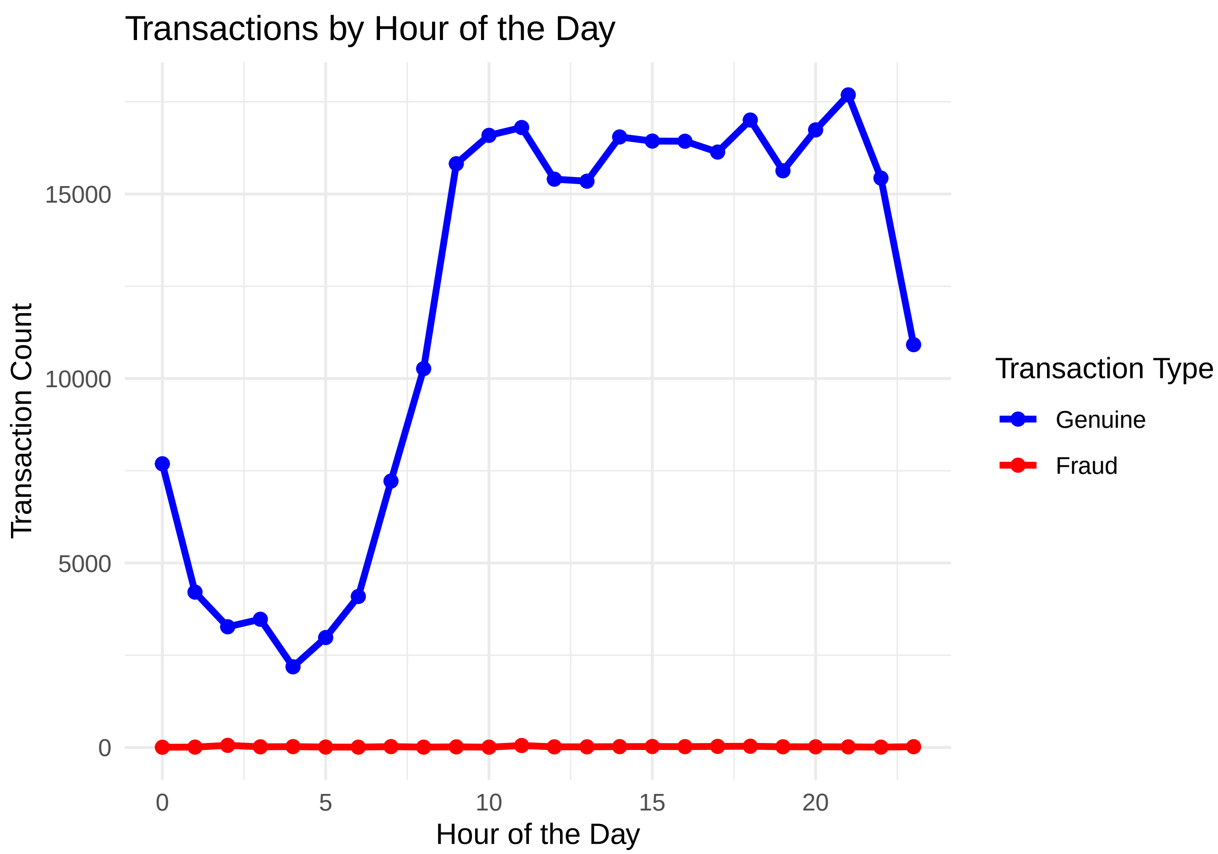
```r
start_date <- as.POSIXct("2013-01-01 00:00:00", tz = "UTC")

# Converting seconds to actual Date-Time format
hw_data$Time <- start_date + hw_data$Time

# Viewing the dataset
glimpse(hw_data)
```

```
## Rows: 284,807
## Columns: 31
## $ Time   <dttm> 2013-01-01 00:00:00, 2013-01-01 00:00:00, 2013-01-01 00:00:01,~
## $ V1     <dbl> -1.3598071, 1.1918571, -1.3583541, -0.9662717, -1.1582331, -0.4~
## $ V2     <dbl> -0.07278117, 0.26615071, -1.34016307, -0.18522601, 0.87773676, ~
## $ V3     <dbl> 2.53634674, 0.16648011, 1.77320934, 1.79299334, 1.54871785, 1.1~
## $ V4     <dbl> 1.37815522, 0.44815408, 0.37977959, -0.86329128, 0.40303393, -0~
## $ V5     <dbl> -0.33832077, 0.06001765, -0.50319813, -0.01030888, -0.40719338,~
## $ V6     <dbl> 0.46238778, -0.08236081, 1.80049938, 1.24720317, 0.09592146, -0~
## $ V7     <dbl> 0.239598554, -0.078802983, 0.791460956, 0.237608940, 0.59294074~
## $ V8     <dbl> 0.098697901, 0.085101655, 0.247675787, 0.377435875, -0.27053267~
## $ V9     <dbl> 0.3637870, -0.2554251, -1.5146543, -1.3870241, 0.8177393, -0.56~
## $ V10    <dbl> 0.09079417, -0.16697441, 0.20764287, -0.05495192, 0.75307443, -~
## $ V11    <dbl> -0.55159953, 1.61272666, 0.62450146, -0.22648726, -0.82284288, ~
## $ V12    <dbl> -0.61780086, 1.06523531, 0.06608369, 0.17822823, 0.53819555, 0.~
## $ V13    <dbl> -0.99138985, 0.48909502, 0.71729273, 0.50775687, 1.34585159, -0~
## $ V14    <dbl> -0.31116935, -0.14377230, -0.16594592, -0.28792375, -1.11966984~
## $ V15    <dbl> 1.468176972, 0.635558093, 2.345864949, -0.631418118, 0.17512113~
## $ V16    <dbl> -0.47040053, 0.46391704, -2.89008319, -1.05964725, -0.45144918,~
## $ V17    <dbl> 0.207971242, -0.114804663, 1.109969379, -0.684092786, -0.237033~
## $ V18    <dbl> 0.02579058, -0.18336127, -0.12135931, 1.96577500, -0.03819479, ~
## $ V19    <dbl> 0.40399296, -0.14578304, -2.26185709, -1.23262197, 0.80348692, ~
## $ V20    <dbl> 0.25141210, -0.06908314, 0.52497973, -0.20803778, 0.40854236, 0~
## $ V21    <dbl> -0.018306778, -0.225775248, 0.247998153, -0.108300452, -0.00943~
## $ V22    <dbl> 0.277837576, -0.638671953, 0.771679402, 0.005273597, 0.79827849~
## $ V23    <dbl> -0.110473910, 0.101288021, 0.909412262, -0.190320519, -0.137458~
## $ V24    <dbl> 0.06692808, -0.33984648, -0.68928096, -1.17557533, 0.14126698, ~
## $ V25    <dbl> 0.12853936, 0.16717040, -0.32764183, 0.64737603, -0.20600959, -~
## $ V26    <dbl> -0.18911484, 0.12589453, -0.13909657, -0.22192884, 0.50229222, ~
## $ V27    <dbl> 0.133558377, -0.008983099, -0.055352794, 0.062722849, 0.2194222~
## $ V28    <dbl> -0.021053053, 0.014724169, -0.059751841, 0.061457629, 0.2151531~
## $ Amount <dbl> 149.62, 2.69, 378.66, 123.50, 69.99, 3.67, 4.99, 40.80, 93.20, ~
## $ Class  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

```r
# Extracting hour information from Time variable
hw_data$Hour <- hour(hw_data$Time)

# Counting transactions per hour for each class
hourly_trends <- hw_data %>%
  group_by(Hour, Class) %>%
  summarise(TransactionCount = n(), .groups = "drop")

# Plotting the transaction volume by hour
ggplot(hourly_trends, aes(x = Hour, y = TransactionCount, color = factor(Class))) +
  geom_line(size = 1.2) +
```

```r
  geom_point(size = 2) +
  scale_color_manual(values = c("blue", "red"), labels = c("Genuine", "Fraud")) +
  labs(title = "Transactions by Hour of the Day",
       x = "Hour of the Day",
       y = "Transaction Count",
       color = "Transaction Type") +
  theme_minimal()
```

```
## Warning: Using 'size' aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use 'linewidth' instead.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```

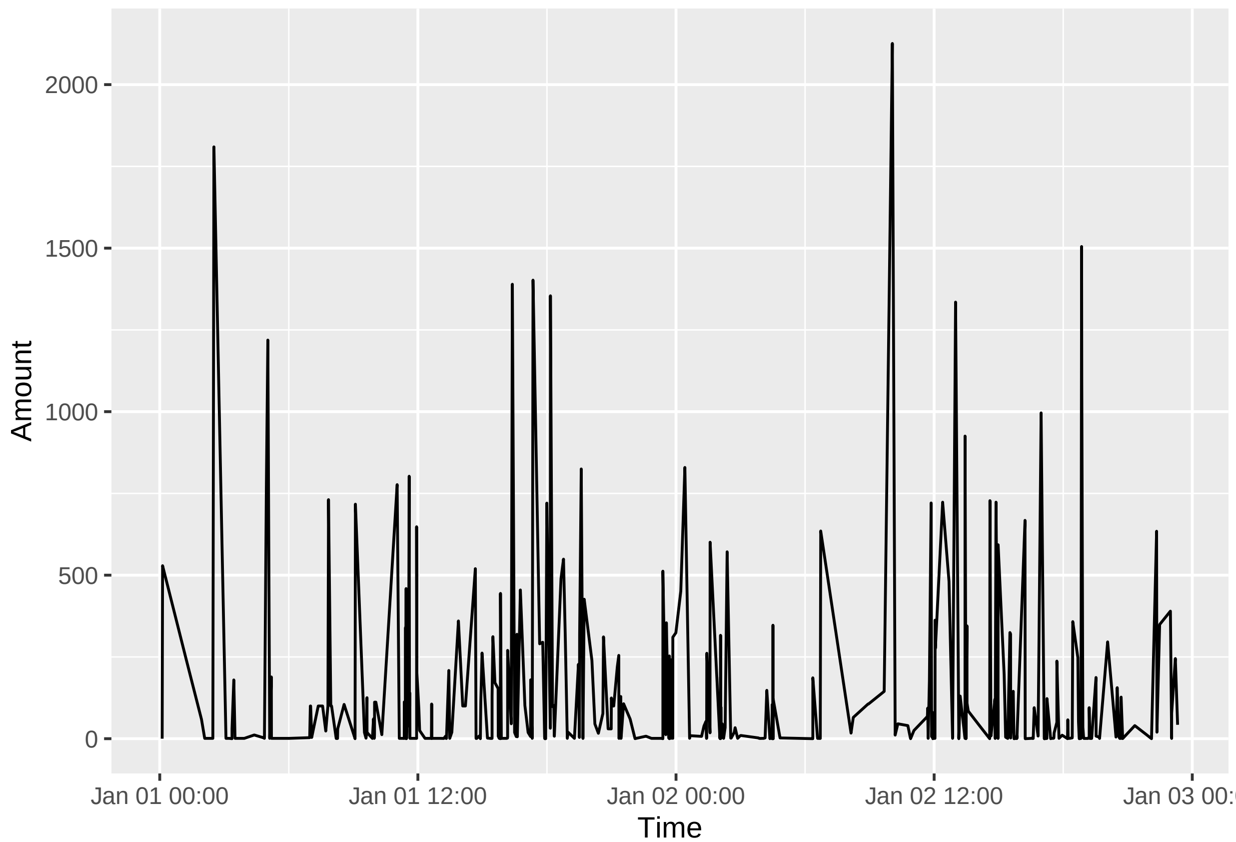


The maximum amount among fraudulent transactions is calculated. A line graph is plotted to understand the trends in the amount among fraudulent transactions. The data has been grouped by the amount in hundreds(rounding off to the lowest whole number), and a graph has been plotted to visualize the trends in genuine versus fraudulent transactions by amount. Majority of the fraud transactions are found to be below 500. Another graph is plotted to understand the trends observed in genuine transactions versus fraudulent transactions.

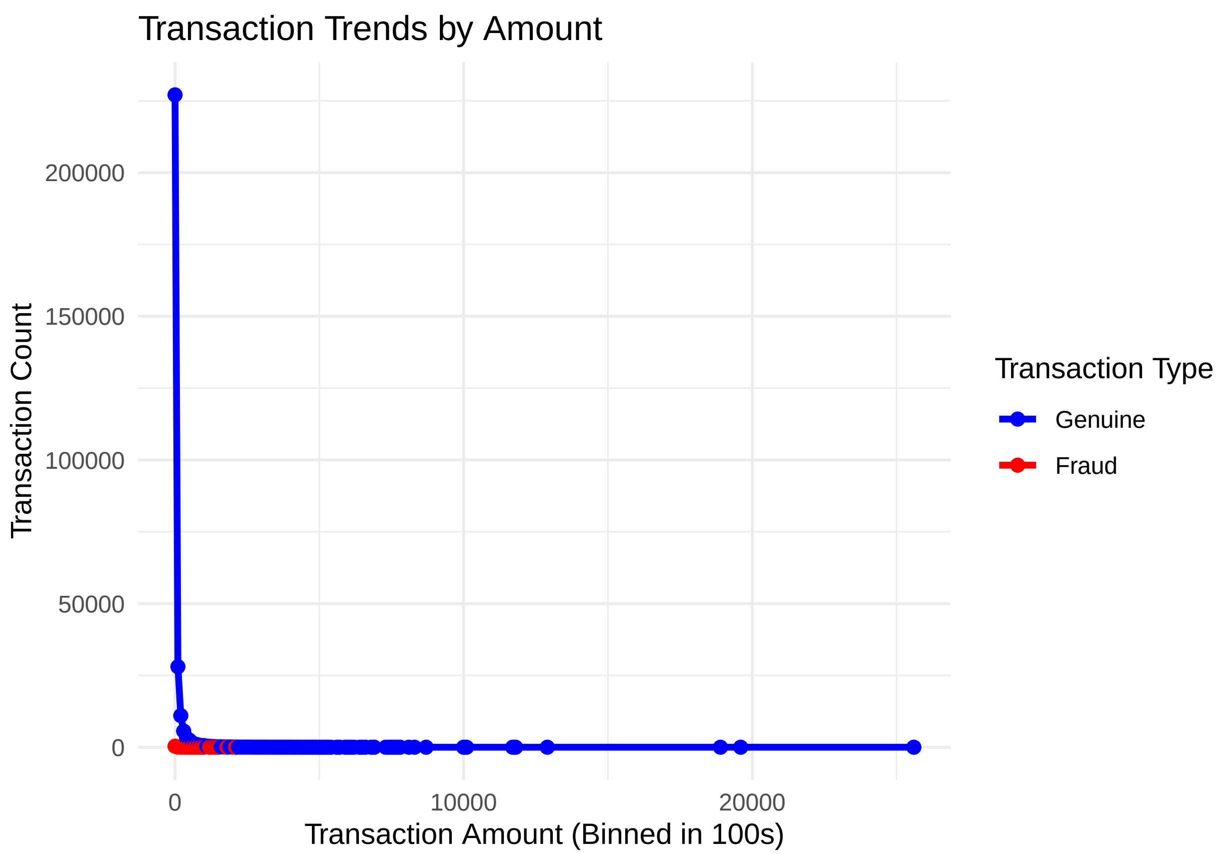
```r
hw_data %>%
  filter(Class == 1) %>%
  summarise(MaxAmount = max(Amount, na.rm = TRUE))
```

```
##   MaxAmount
## 1   2125.87
```

```r
fraud_trend <- hw_data %>% filter(Class==1) %>% arrange(Time)

ggplot(data = fraud_trend, aes(x=Time, y=Amount))+
  geom_line()
```




```r
hw_data <- hw_data %>% mutate(spend_bin = floor(Amount/100)*100)

spending_trends <- hw_data %>%
  group_by(spend_bin, Class) %>%
  summarise(TransactionCount = n(), .groups = "drop")

# Plotting the trend using a line graph
ggplot(spending_trends, aes(x = spend_bin, y = TransactionCount, color = factor(Class))) +
  geom_line(size = 1.2) +
  geom_point(size = 2) +
  scale_color_manual(values = c("blue", "red"), labels = c("Genuine", "Fraud")) +
  labs(title = "Transaction Trends by Amount",
       x = "Transaction Amount (Binned in 100s)",
       y = "Transaction Count",
       color = "Transaction Type") +
  theme_minimal()
```



6. The code identifies the principal components (PCs) that are most correlated with the "Class" variable, which indicates whether a credit card transaction is genuine or fraudulent. It calculates the correlation of each PC with the "Class" column, takes the absolute values, and ranks them in descending order. The top 10 most correlated PCs are then selected. Finally, a subset of the original dataset is created containing these top 10 PCs along with the "Class" column.

```
correlations <- sapply(hw_data[,2:29], function(x) cor(x, hw_data$Class, use="complete.obs"))

cor_df <- data.frame(PC = names(correlations), Correlation = abs(correlations))

cor_df <- cor_df[order(-cor_df$Correlation), ]

print(cor_df[1:10, ])
```

```
##      PC Correlation
## V17 V17   0.3264811
## V14 V14   0.3025437
## V12 V12   0.2605929
## V10 V10   0.2168829
## V16 V16   0.1965389
## V3   V3   0.1929608
## V7   V7   0.1872566
## V11 V11   0.1548756
## V4   V4   0.1334475
## V18 V18   0.1114853
```

```r
selected_pca <- hw_data %>% select(V17, V14, V12, V10, V16, V3, V7, V11, V4, V18,Class)
```

```r
# Sorting by Class (Fraud cases first)
sorted_pca <- selected_pca %>% arrange(desc(Class))

# Checking first few rows to confirm sorting
head(sorted_pca)
```

```
##           V17        V14         V12        V10        V16         V3
## 1  -2.8300557  -4.289254  -2.8999074 -2.7722721 -1.1407472 -1.6098507
## 2   0.5997174  -1.692029  -0.5031409 -0.8385866  0.6667797  1.0884628
## 3  -4.7818309  -1.470102  -6.5601243 -1.5254116 -2.2821938 -0.3597447
## 4 -12.5984185  -6.771097 -10.9128193 -4.8016374 -7.3580832 -2.5928442
## 5   6.7393844  -6.079337  -4.6096284 -2.4474693  2.5818510 -4.3045969
## 6  -1.1290559 -10.691196  -9.8544848 -6.1878906 -2.0419738 -6.2406966
##           V7        V11       V4         V18 Class
## 1 -2.5373873  3.2020332 3.997906 -0.01682247     1
## 2  0.3255743 -0.4145754 2.288644  1.72532101     1
## 3  0.5623198  2.0329122 2.330243 -2.61566494     1
## 4 -3.4961973  4.8958442 2.679787 -5.13154863     1
## 5  1.7134450  2.1013439 4.732795  3.04249318     1
## 6 -1.6317347  5.6643947 6.675732  0.11645252     1
```

6. To build a fraud detection model, the XGBoost algorithm is used. XGBoost is a powerful machine learning model known for its speed and accuracy in classification tasks. The goal here is to train the model to distinguish between fraudulent and genuine transactions based on numerical features extracted from PCA-transformed data. XGBoost improves upon decision trees by reducing overfitting and iteratively correcting misclassified points. Unlike Random Forest, XGBoost builds trees sequentially, focusing on errors from previous trees.

First, the dataset is prepared by selecting the relevant features. The Time column is removed, as it does not provide predictive power in this case. The remaining columns are converted into a numerical matrix format,suitable for XGBoost. The target variable, Class, which indicates whether a transaction is fraud (1) or genuine (0), is extracted as a separate numeric vector.

Seed is set, and the dataset is split into training and test data at 80:20 ratio.

The XGBoost parameters are selected to optimize performance. The objective function is set to binary:logistic. The evaluation metric chosen is AUC - Area Under the Curve, which is useful for measuring model performance on imbalanced datasets. A learning rate (eta) of 0.1 is used to control the rate at which the model learns from errors. The maximum depth of decision trees is set to 6 to prevent overfitting. Subsample and colsample_bytree are set to 0.8, ensuring that only a portion of the dataset and features are used in each tree to improve generalization.

Once the dataset is converted into the appropriate XGBoost DMatrix format, the model is trained for 100 boosting rounds. Each round updates the model to improve fraud detection accuracy while minimizing misclassifications.

Feature importance is analyzed to understand which variables contribute most to fraud detection. The xgb.importance() function ranks the features, and the top 10 most important features are visualized. This helps interpret the model's decision-making process and identify the most relevant factors in detecting fraudulent transactions.

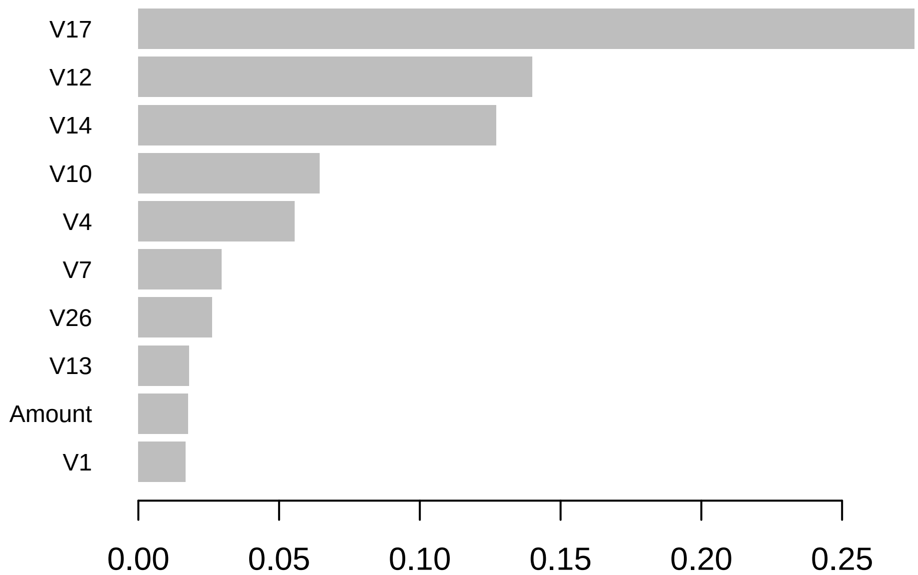
To assess how well the model performs, predictions are generated on the dataset. Since XGBoost outputs probability scores, they are converted into binary predictions, with a threshold of 0.5 used to classify transactions as either fraud or genuine.

A confusion matrix is created to compare predicted values against actual values. The confusion matrix provides insights into how many fraudulent transactions were correctly identified (True Positives) and how many genuine transactions were incorrectly classified as fraud (False Positives).

Several key performance metrics are then calculated:

Accuracy measures the overall correctness of the model in predicting fraud and genuine transactions. Precision evaluates how many of the predicted fraudulent transactions were actually fraud. Recall determines how many of the actual fraudulent transactions were successfully detected. F1-Score is the harmonic mean of Precision and Recall, providing a balanced measure of model performance. The results of these metrics indicate how effectively the model identifies fraudulent transactions. If precision is high but recall is low, it means the model is conservative in predicting fraud, possibly missing some actual fraudulent transactions. Conversely, if recall is high but precision is low, the model flags too many genuine transactions as fraud.

```r
#Setting a seed for reproducibility
set.seed(42)

#Removing Time column
hw_data1 <- hw_data %>% select(-Time)

# Splitting data into training (80%) and test (20%)
train_indices <- createDataPartition(hw_data1$Class, p = 0.8, list = FALSE)

train_data <- hw_data1[train_indices, ]
test_data  <- hw_data1[-train_indices, ]

#Converting training and test Data into matrices for XGBoost
train_features_matrix <- data.matrix(train_data %>% select(-Class))
train_labels_vector <- as.numeric(train_data$Class)

test_features_matrix <- data.matrix(test_data %>% select(-Class))
test_labels_vector <- as.numeric(test_data$Class)

#Converting to XGBoost DMatrix format
dtrain <- xgb.DMatrix(data = train_features_matrix, label = train_labels_vector)
dtest  <- xgb.DMatrix(data = test_features_matrix, label = test_labels_vector)

#Defining XGBoost Parameters
params <- list(
  objective = "binary:logistic",
  eval_metric = "auc",
  eta = 0.1,
  max_depth = 6,
  subsample = 0.8,
  colsample_bytree = 0.8
)

#Training XGBoost Model
xgb_model <- xgb.train(params = params, data = dtrain, nrounds = 100)

#Featuring Importance Plot
importance_matrix <- xgb.importance(feature_names = colnames(train_features_matrix), model = xgb_model)
xgb.plot.importance(importance_matrix, top_n = 10)
```




```r
#Making predictions on the Test Data
test_predictions <- predict(xgb_model, dtest)

#Converting probabilities into binary labels
predicted_test_labels <- ifelse(test_predictions > 0.5, 1, 0)

#Creating confusion matrix
confusion_matrix <- table(predicted_test_labels, test_labels_vector)
print(confusion_matrix)
```

```
##                      test_labels_vector
## predicted_test_labels     0     1
##                     0 56857    21
##                     1     2    81
```

```r
#computing model performance metrics
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", round(accuracy, 4)))
```

```
## [1] "Accuracy: 0.9996"
```

```r
precision <- confusion_matrix[2,2] / sum(confusion_matrix[,2])
print(paste("Precision:", round(precision, 4)))
```

```
## [1] "Precision: 0.7941"
```

```
recall <- confusion_matrix[2,2] / sum(confusion_matrix[2,])
print(paste("Recall:", round(recall, 4)))
```

```
## [1] "Recall: 0.9759"
```

```
f1_score <- 2 * (precision * recall) / (precision + recall)
print(paste("F1-Score:", round(f1_score, 4)))
```

```
## [1] "F1-Score: 0.8757"
```

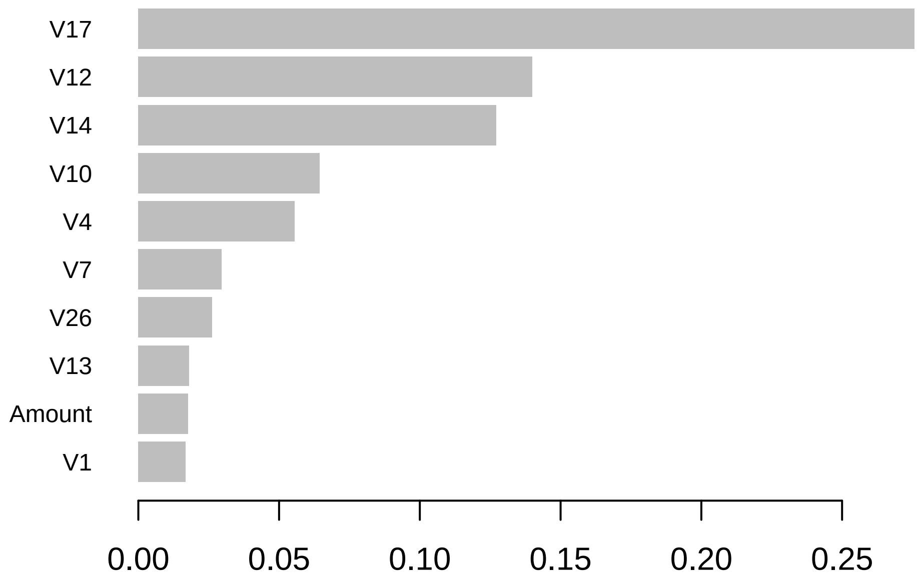
8. To further analyze the model's effectiveness, various visualizations are created.

Feature importance plot The first visualization highlights the most important features in the dataset that influence fraud detection. The xgb.plot.importance() function is used to display the top 10 features ranked by their contribution to the model. This allows us to understand which PCA-transformed variables play a crucial role in predicting fraudulent activity.

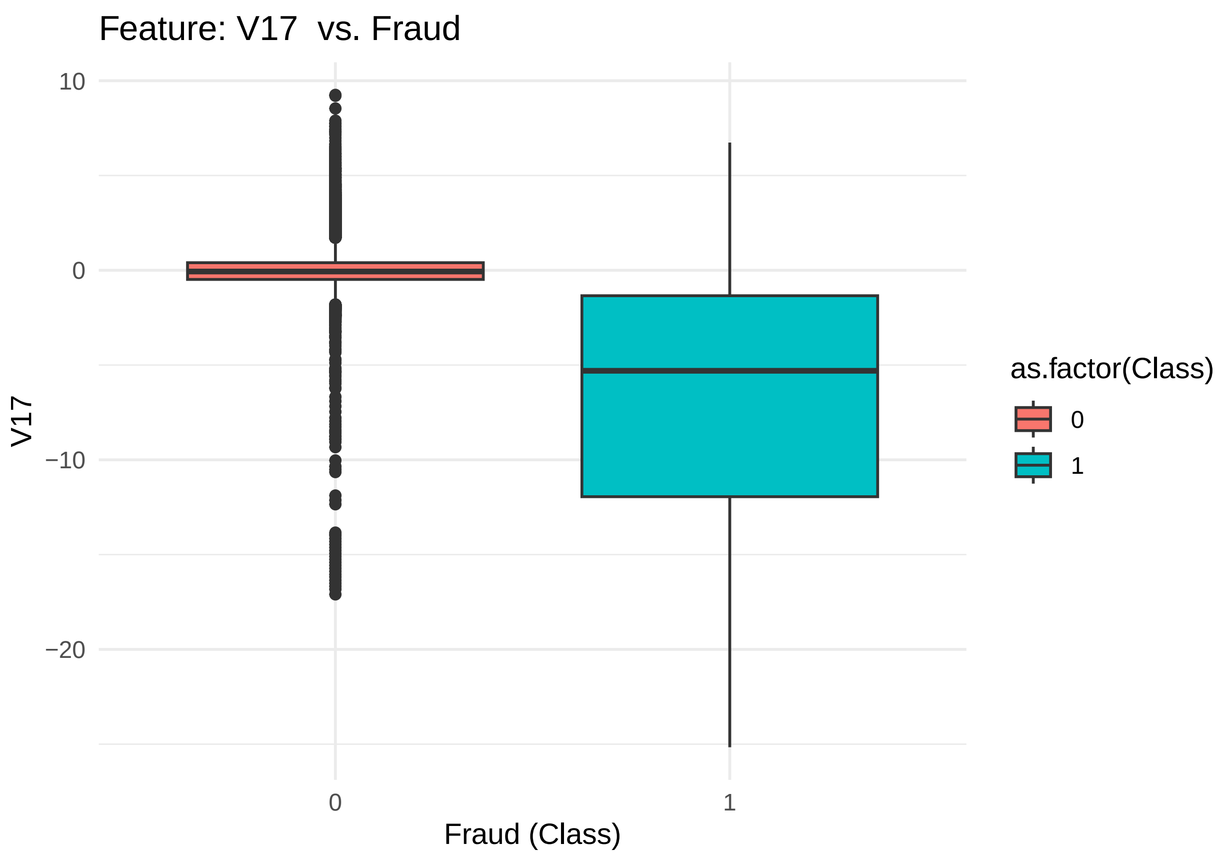
Boxplot of an important feature A boxplot is created to visualize how one of the most influential features (e.g., V17) varies between genuine and fraudulent transactions. This plot shows the distribution of this feature for fraud and non-fraud transactions, helping us see whether fraudulent transactions exhibit distinct patterns compared to genuine transactions.

Confusion Matrix heatmap The confusion matrix is visualized as a heatmap, where the intensity of colors represents the number of correctly and incorrectly classified transactions. The diagonal of the heatmap represents correct predictions, while off-diagonal elements indicate misclassified transactions. A high number of false positives (genuine transactions wrongly flagged as fraud) can impact the customer experience, while a high number of false negatives (fraud that was missed) can result in financial losses.

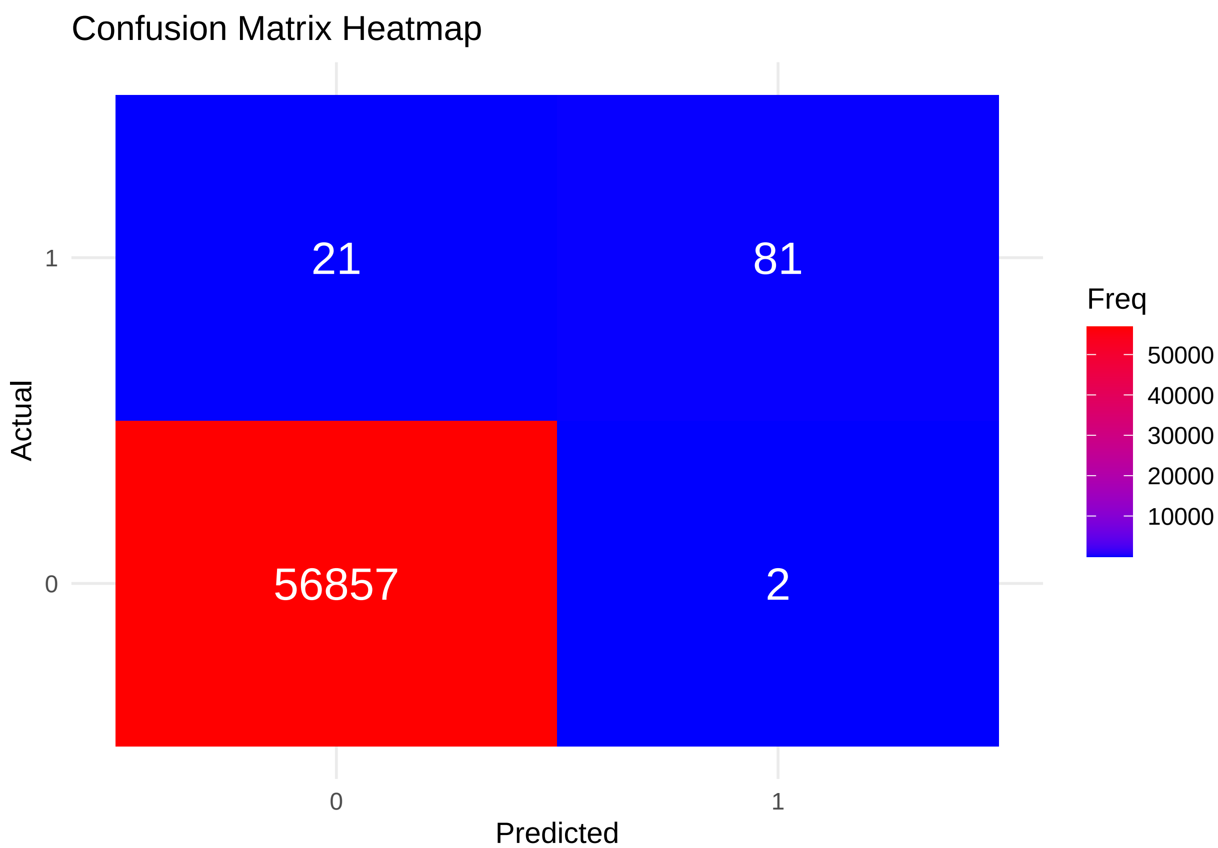
```
#visualization of feature importance
xgb.plot.importance(importance_matrix, top_n = 10)
```




```r
#Boxplot for one important feature
top_feature <- "V17"
ggplot(hw_data, aes(x = as.factor(Class), y = get(top_feature), fill = as.factor(Class))) +
  geom_boxplot() +
  labs(title = paste("Feature:", top_feature, " vs. Fraud"), x = "Fraud (Class)", y = top_feature) +
  theme_minimal()
```




```r
#confusion matrix heatmap
cm_df <- as.data.frame(as.table(confusion_matrix))
ggplot(cm_df, aes(x = predicted_test_labels, y = test_labels_vector, fill = Freq)) +
  geom_tile() +
  geom_text(aes(label = Freq), color = "white", size = 6) +
  labs(title = "Confusion Matrix Heatmap", x = "Predicted", y = "Actual") +
  scale_fill_gradient(low = "blue", high = "red") +
  theme_minimal()
```



7. Results:

The analysis identifies V17 as the top strongly correlated PCA feature with the class column, showing a correlation of 0.32. The XGBoost model exhibited high specificity (99%), correctly identifying the majority of genuine transactions, with only 21 false positives. However, it misclassified 2 fraudulent transactions, resulting in a sensitivity of 79%, indicating that some fraud cases were missed. The overall model accuracy reached 99%, demonstrating strong performance in distinguishing between fraud and genuine transactions. Precision was measured at 79%, showing the proportion of correctly identified fraud cases among all fraud predictions. The model achieved a recall of 96%, meaning it can successfully detect the majority of actual fraud cases. The F1-score, which balances precision and recall, was 87%, highlighting the model's effectiveness in handling fraud detection within an imbalanced dataset. The boxplot comparing Feature V17 with fraudulent and genuine transactions reveals distinct patterns. Fraudulent transactions exhibit a lower median V17 value, with most values below zero and a wider interquartile range, indicating greater variability. In contrast, genuine transactions have a median closer to zero, a narrower distribution, and several outliers. This suggests that negative values of V17 are more indicative of fraud, making it a valuable predictor in fraud detection. The presence of outliers in genuine transactions could indicate potential false positives, warranting further investigation.The confusion matrix shows the distribution of the actual and predicted values.