

Credit Card Fraud Detection Technical Report

Group #1: Aneesah Emran, Kionna Hughes

1. Introduction

Fraud detection in credit card transactions is a crucial application of machine learning. This report details our analysis and findings from applying different machine learning algorithms to detect fraudulent transactions. We have followed a structured methodology, starting from dataset exploration, preprocessing, and model training, to performance evaluation and comparison.

2. Dataset Overview

The dataset consists of credit card transactions, including various features such as transaction time, amount, and a class label indicating whether the transaction is fraudulent or normal. Our first step was to load and explore the dataset.

2.1 Data Exploration

- **Dataset Dimensions:** The dataset contains multiple rows and columns, representing different transactions and their respective features.
- **Feature Names:** The dataset includes attributes such as Time, Amount, and Class.
- **Handling Missing Values:** We checked for null values to ensure data completeness.
- **Statistical Summary:** Key statistics such as quartiles, variance, and standard deviation were calculated to understand the distribution of transaction amounts.
- **Data Visualization:** Violin plots and scatter plots were used to visualize data distribution and relationships between fraud and normal transactions.

Milestone 2

I. Importing the Dataset

- **Import the dataset that contains transactions made by credit cards.**
- **Read the dataset and load it to a data frame.**

*Be sure to include a screenshot of your uploaded dataset in your submission to this assignment.

The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'sample_data' containing a file 'CreditCardFraud_Dataset (1).csv'. The code editor shows the following R code:

```
install.packages("tidyverse")
install.packages("ggplot2")
install.packages("dplyr")
install.packages("readr")

library(tidyverse)
library(ggplot2)
library(dplyr)
library(readr)
```

The output of the code execution is shown below the code cells. It indicates the installation of packages into '/usr/local/lib/R/site-library' and the loading of the tidyverse packages. The output also shows the conflicts between the installed packages and the tidyverse packages, and the successful loading of the readr package.

The code then reads the CSV file and displays the first few rows of the data frame:

```
file_path <- "/content/CreditCardFraud_Dataset (1).csv"
dataset <- read.csv(file_path)

head(dataset)
```

The output of the head function is a data frame with 6 columns: TransactionID, Time, Amount, MerchantID, TransactionType, and Class. The data frame contains 6 rows of transaction data:

	TransactionID	Time	Amount	MerchantID	TransactionType	Class
	<chr>	<int>	<int>	<chr>	<chr>	<chr>
1	T1	113813	321	M15	Mobile	Normal
2	T2	1510	309	M28	Mobile	Normal
3	T3	123891	687	M25	In-Store	Normal
4	T4	155005	206	M37	Online	Normal
5	T5	100917	137	M72	Online	Normal
6	T6	66756	938	M85	Mobile	Normal

II. Exploring Your Dataset

In this phase, we want to explore a little bit on the dataset and get more information about our data. Please provide some code to find the answer to the following questions.

1. What is the dimension of the dataset?

The dataset's dimension tells us how many transactions (rows) and features (columns) it contains. This helps determine its size and complexity. The number of rows shows how much data we have, while columns represent attributes like time, amount, and fraud status. Large datasets may need powerful computing resources, while small ones might lack enough data for accurate predictions. Knowing dimensions helps in planning preprocessing steps such as handling missing values or reducing unnecessary features. Running `dim(dataset)` provides a quick overview, ensuring the dataset was loaded

correctly. If the dataset has unexpected dimensions (e.g., too few columns), it could indicate errors in data formatting. Checking this early prevents issues later in analysis.

2. What are the first and last five transactions on the dataset?

Viewing the first and last five transactions helps understand how the data is structured. The `head(dataset, 5)` function shows the first five entries, while `tail(dataset, 5)` displays the last five. This is useful for spotting patterns, ensuring the dataset is correctly loaded, and checking for inconsistencies. If transactions are ordered by time, the first rows may contain older transactions and the last ones newer. This step can reveal anomalies, such as missing values or unusual transactions. Additionally, it helps verify if fraud cases are evenly distributed or concentrated at specific points. Understanding the dataset structure from the start makes it easier to apply further analysis.

3. Check if there are any null values in the dataset.

Missing values can affect analysis and machine learning models, so checking for them is crucial. Running `colSums(is.na(dataset))` counts missing values in each column. If a column has many missing values, it might need cleaning or imputation. Missing transaction amounts or fraud labels can lead to inaccurate results, so identifying gaps early helps determine the best way to handle them. If the dataset has no missing values, analysis can proceed smoothly. If null values are found, they may need to be replaced with averages, removed, or estimated based on other features. Checking for null values ensures that the dataset is complete and reliable before deeper analysis.

4. What are the names of the features in this dataset?

Feature names describe the dataset's columns, providing insight into the available data. Using `names(dataset)`, we can list all feature names and understand what information is included. Typical features in fraud detection datasets include `Time`, `Amount`, and `Class` (indicating fraud or normal transactions). Some datasets may also have engineered features that capture transaction behaviors. Identifying feature names early helps determine which attributes are relevant for analysis. If the dataset contains unnamed or mislabeled columns, they may need to be renamed for clarity. Checking feature names ensures proper data interpretation and smooth analysis.

5. Please provide the quartile information on the amount of transferred money in transactions.

Quartiles summarize data distribution by dividing it into four parts: minimum, Q1 (25th percentile), median (50th percentile), Q3 (75th percentile), and maximum. Running `summary(dataset$Amount)` provides these values. The median represents the typical transaction amount, while Q1 and Q3 show the spread. If the difference between Q1 and Q3 (interquartile range) is large, transaction amounts vary significantly. Quartile

analysis helps detect outliers—values far from Q1 and Q3. Fraudulent transactions often have different distributions, making this step useful for fraud detection. Understanding quartiles aids in filtering extreme values and setting appropriate thresholds for analysis.

6. What is the variance and standard deviation values for the amount of transferred money in transactions?

Variance and standard deviation measure data spread. Variance (`var(dataset$Amount)`) shows how much transaction amounts deviate from the mean, while standard deviation (`sd(dataset$Amount)`) provides a clearer measure in the same units as the data. A high standard deviation means transaction amounts vary greatly, while a low one suggests consistency. If fraud transactions have higher variance, it indicates they differ significantly from normal ones. Understanding these statistics helps in detecting unusual transaction patterns and making data-driven fraud detection decisions.

7. How many transactions' amounts are more than the mean value?

This step identifies transactions above the average amount using `sum(dataset$Amount > mean(dataset$Amount, na.rm = TRUE))`. If many transactions exceed the mean, it suggests a right-skewed distribution with some large transactions. Fraudulent transactions often involve unusually high amounts, so this count helps detect irregularities. Knowing how many transactions are above average provides insights into data distribution and can be useful in setting fraud detection thresholds. If the majority of transactions are below the mean, the dataset likely has a few extreme outliers.

8. Please provide the violin plots (with boxplot within) for the amount of money on transactions for two categories of Fraud and Normal classes.

A violin plot visualizes the distribution of transaction amounts for normal and fraudulent transactions. The `ggplot2` function plots transaction categories with density curves and boxplots inside. This helps compare data distributions, showing if fraudulent transactions have different patterns from normal ones. A wider section indicates more frequent transaction amounts, while outliers appear as points outside the range. If fraud transactions have a different shape, they may have unique transaction patterns, aiding in fraud detection. Violin plots make it easier to understand data distribution compared to traditional boxplots.

9. What's the shape of the violin plot? Is it a normal distribution?

The violin plot shows the distribution of transaction amounts for fraudulent and normal transactions. The shape of both distributions is not perfectly symmetrical, indicating that

the data is not normally distributed. Instead, the plots display a right-skewed distribution with long tails extending upwards. This suggests that most transactions are of lower amounts, while a few transactions involve very high amounts, creating the long upper tails.

The presence of multiple peaks (modes), especially in the fraudulent transactions, suggests the data may have clusters rather than following a standard bell curve. The boxplots within the violins indicate that the interquartile range (IQR) for both classes is relatively small compared to the entire range, reinforcing the skewed nature of the data.

Since normal distributions should have a symmetrical shape with a single peak in the middle, this dataset does not follow that pattern. Instead, it has heavy tails and outliers, making it non-normal.

Screenshots of code for 1-9 :

Files

Analyze your files with code written by Gemini Upload



..

sample_data
CreditCardFraud_Dataset (1).csv

```
dim(dataset)
```

```
10000 6
```

```
[6] head(dataset, 5)  
tail(dataset, 5)
```

```
A data.frame: 5 × 6  
  TransactionID Time Amount MerchantID TransactionType Class  
    <chr>    <int>  <int>    <chr>         <chr>    <chr>  
1          T1  113813    321         M15           Mobile Normal  
2          T2   1510    309         M28           Mobile Normal  
3          T3  123891    687         M25           In-Store Normal  
4          T4  155005    206         M37           Online Normal  
5          T5  100917    137         M72           Online Normal  
  
A data.frame: 5 × 6  
  TransactionID Time Amount MerchantID TransactionType Class  
    <chr>    <int>  <int>    <chr>         <chr>    <chr>  
9996        T9996  99875    310         M57           Online Normal  
9997        T9997 124316    420         M93           In-Store Normal  
9998        T9998 151332    464         M66           Online Normal  
9999        T9999  46416    974          M6           Online Normal  
10000       T10000 11633    543         M31           Online Normal
```

```
[7] colSums(is.na(dataset))
```

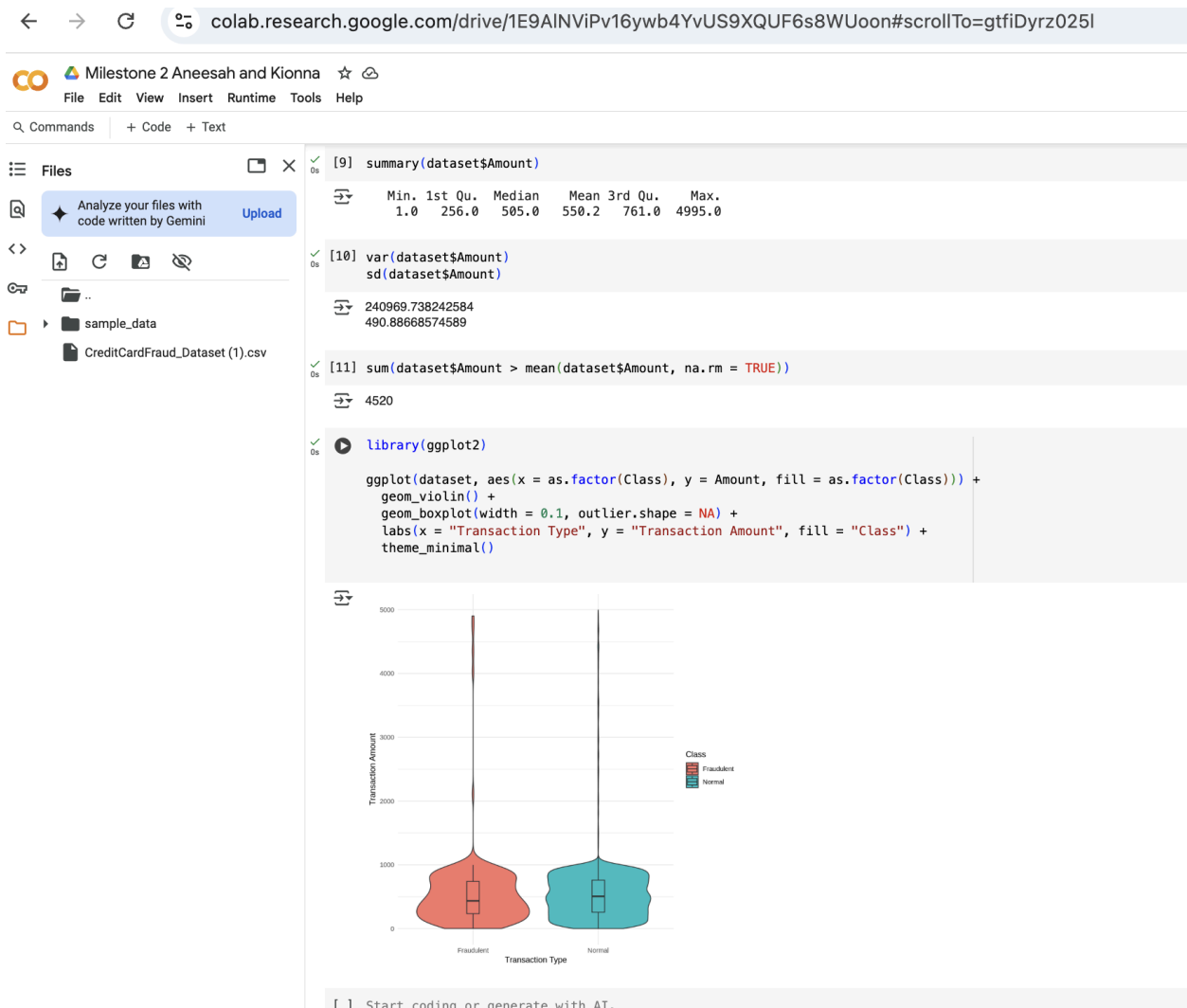
```
TransactionID: 0 Time: 0 Amount: 0 MerchantID: 0 TransactionType: 0 Class: 0
```

```
[8] names(dataset)
```

```
"TransactionID" "Time" "Amount" "MerchantID" "TransactionType" "Class"
```

```
[9] summary(dataset$Amount)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.  
 1.0   256.0   505.0  550.2  761.0 4995.0
```



3. Data Preprocessing and Splitting

Milestone 3: Data Manipulation and Modeling

I. Manipulating the Data

In this phase, we want to scale our data into a specific range. This transformation is done to avoid having extreme values in our data which negatively affects our model. We apply this scale (a.k.a feature standardization), to the number of features, as all other features have been normalized before. Please apply the feature standardization on the amount feature and provide evidence to the following requests.

1. Show the first 5 transactions.
2. Apply the scale function on the amount feature.
3. Save the new dataset in another data frame.
4. Could you show the first 5 transactions after the changes?

← → ↺ colab.research.google.com/drive/1SORLEmVtep0RM3Z33oigdpwH5yfphP6#scrollTo=R9VI3vo-9k_S

Milestone 3 Aneesah and Kionna ☆ ☁
File Edit View Insert Runtime Tools Help

Q Commands + Code + Text

Files

Analyze your files with code written by Gemini Upload

sample_data
CreditCardFraud_Dataset-3 (1)...

```
# Install required package if not installed
install.packages("readr")

# Load necessary library
library(readr)

# Load the dataset (modify the path if necessary)
dataset <- read_csv("/content/CreditCardFraud_Dataset-3 (1).csv")

# View the first few rows
head(dataset, 5)
```

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

Rows: 10000 Columns: 6
— Column specification —
Delimiter: ","
chr (4): TransactionID, MerchantID, TransactionType, Class
dbl (2): Time, Amount

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.
A tibble: 5 × 6

TransactionID	Time	Amount	MerchantID	TransactionType	Class
<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>
T1	113813	321	M15	Mobile	Normal
T2	1510	309	M28	Mobile	Normal
T3	123891	687	M25	In-Store	Normal
T4	155005	206	M37	Online	Normal
T5	100917	137	M72	Online	Normal

```
[2] # Standardize the 'Amount' feature
dataset$Amount <- scale(dataset$Amount)

# Save the new dataset in another data frame
scaled_dataset <- dataset

# Show the first 5 transactions after scaling
head(scaled_dataset, 5)
```

```
[2] # Standardize the 'Amount' feature
dataset$Amount <- scale(dataset$Amount)

# Save the new dataset in another data frame
scaled_dataset <- dataset

# Show the first 5 transactions after scaling
head(scaled_dataset, 5)
```

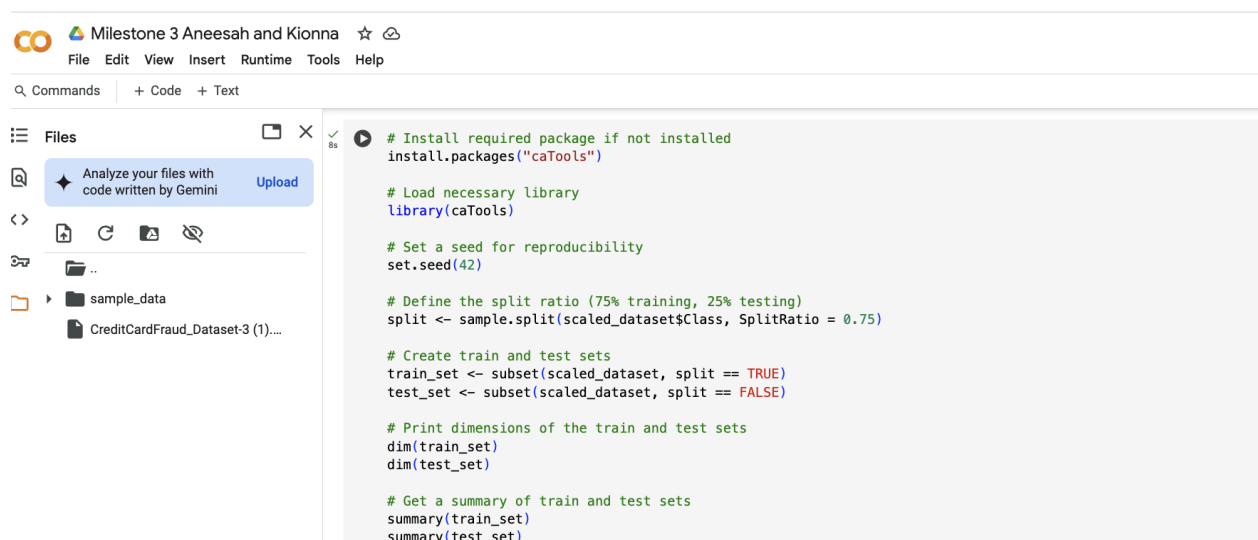
A tibble: 5 × 6

TransactionID	Time	Amount	MerchantID	TransactionType	Class
<chr>	<dbl>	<dbl[,1]>	<chr>	<chr>	<chr>
T1	113813	-0.4669750	M15	Mobile	Normal
T2	1510	-0.4914205	M28	Mobile	Normal
T3	123891	0.2786146	M25	In-Store	Normal

II. Modeling the Data

In this phase, we split our dataset into train and test sets. For training our model, we need to have two groups, where one is used to learn the model, and the other is used to check how effective our model is in detecting unknown data. For this purpose, we randomly divide them into a 75%/25% ratio. The first creates our training set and the last is our test set. Please prepare your train and test sets based on the original dataset, and provide evidence for the following steps.

1. Select a seed value for each split operation.
2. Set the split ratio (75%/25%).
3. Divide the dataset and save the new splits into train and test sets.
4. Print out the train set and test set dimensions.
5. Give a summary of train and test sets.
6. Create the histogram plot based on the Amount feature for the two sets.



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'sample_data' containing a file 'CreditCardFraud_Dataset-3 (1)...'. The code editor contains the following R code:

```
# Install required package if not installed
install.packages("caTools")

# Load necessary library
library(caTools)

# Set a seed for reproducibility
set.seed(42)

# Define the split ratio (75% training, 25% testing)
split <- sample.split(scaled_dataset$Class, SplitRatio = 0.75)

# Create train and test sets
train_set <- subset(scaled_dataset, split == TRUE)
test_set <- subset(scaled_dataset, split == FALSE)

# Print dimensions of the train and test sets
dim(train_set)
dim(test_set)

# Get a summary of train and test sets
summary(train_set)
summary(test_set)
```

sample_data
CreditCardFraud_Dataset-3 (1)...

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

also installing the dependency 'bitops'

```
7500 · 6
2500 · 6
TransactionID      Time      Amount.V1      MerchantID
Length:7500      Min.   :   15      Min.   :~-1.118857      Length:7500
Class :character  1st Qu.: 44406      1st Qu.:~-0.593786      Class :character
Mode  :character  Median : 88629      Median :~-0.086032      Mode  :character
                  Mean  : 88350      Mean  :~-0.000746
                  3rd Qu.:131938      3rd Qu.: 0.435983
                  Max.  :172793      Max.  : 9.054571

TransactionType    Class
Length:7500      Length:7500
Class :character  Class :character
Mode  :character  Mode  :character
```

```
TransactionID      Time      Amount.V1      MerchantID
Length:2500      Min.   :   42      Min.   :~-1.116819      Length:2500
Class :character  1st Qu.: 43083      1st Qu.:~-0.605500      Class :character
Mode  :character  Median : 85013      Median :~-0.107422      Mode  :character
                  Mean  : 86795      Mean  : 0.002239
                  3rd Qu.:132069      3rd Qu.: 0.413065
                  Max.  :172749      Max.  : 8.867155

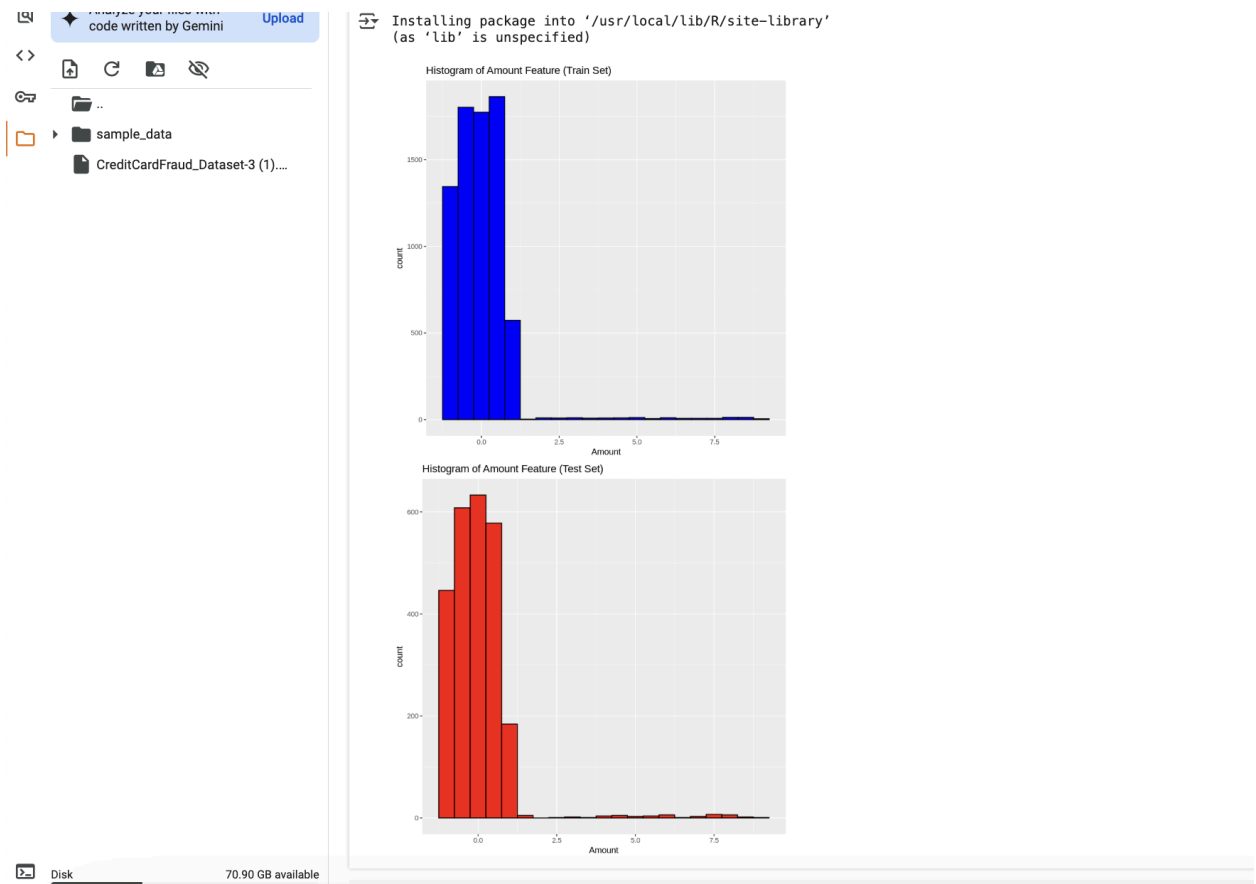
TransactionType    Class
Length:2500      Length:2500
Class :character  Class :character
Mode  :character  Mode  :character
```

```
[4] # Install required package if not installed
install.packages("ggplot2")

# Load necessary library
library(ggplot2)

# Histogram for train set
ggplot(train_set, aes(x = Amount)) +
  geom_histogram(binwidth = 0.5, fill = "blue", color = "black") +
  ggtitle("Histogram of Amount Feature (Train Set)")

# Histogram for test set
ggplot(test_set, aes(x = Amount)) +
  geom_histogram(binwidth = 0.5, fill = "red", color = "black") +
  ggtitle("Histogram of Amount Feature (Test Set)")
```



3.1 Feature Scaling

To normalize transaction amounts, feature standardization was applied, ensuring that values were within a comparable range.

3.2 Train-Test Split

The dataset was divided into training (75%) and testing (25%) sets using random splitting. This ensures that our models generalize well to unseen data.

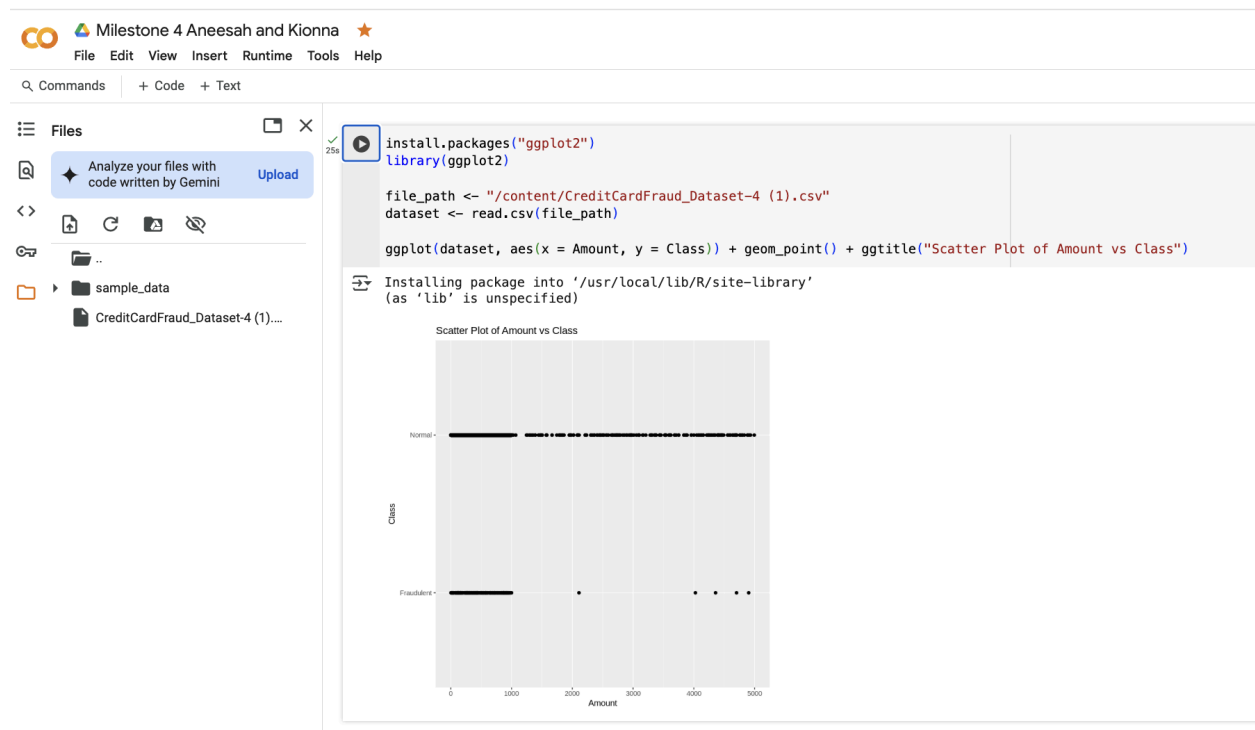
4. Machine Learning Models

Milestone 4: Correlation Plot and Setting & Fitting the Model

This milestone includes four phases: (1) Correlation Plot; (2) Set Your Model; (3) Fitting the Model; and (4) More Train and Test Sets. You must complete all sections in order to successfully meet the requirements of this milestone.

I. Correlation Plot

Provide the relationship between the Amount and Class features using the scatter plot.



II. Set Your Model

1. Set a logistic regression model.
2. Train the logistic regression model and define the label feature based on the Class feature

2. Plot your model.
3. Print out the confusion matrix.
4. Provide the ROC curve for your model.
5. Calculate the AUC for your model's curve.

The screenshot shows a Jupyter Notebook titled "Milestone 4 Aneesah and Kionna". The interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar with icons for file operations. The left sidebar shows a file explorer with a folder named "sample_data" containing a file "CreditCardFraud_Dataset-4 (1)....". The main area displays R code for installing the 'pROC' package and its dependency 'caret', making predictions on test data, displaying the model summary, plotting the model, generating and printing the confusion matrix, computing and plotting the ROC curve, and calculating the AUC value.

```
install.packages("pROC")
install.packages("caret")
library(pROC)
library(caret)

# Make predictions on test data
predictions <- predict(model, test_set, type = "response")
predicted_classes <- ifelse(predictions > 0.5, 1, 0)

# Display model summary
summary(model)

# Plot the model
plot(model)

# Generate and print confusion matrix
conf_matrix <- table(test_set$class, predicted_classes)
print(conf_matrix)

# Compute and plot the ROC curve
roc_curve <- roc(test_set$class, predictions)
plot(roc_curve, main = "ROC Curve")

# Calculate AUC value
auc_value <- auc(roc_curve)
print(auc_value)
```

Installing package into '/usr/local/lib/R/site-library'

(as 'lib' is unspecified)

also installing the dependency 'plyr'

Installing package into '/usr/local/lib/R/site-library'

(as 'lib' is unspecified)

also installing the dependencies 'listenv', 'parallelly', 'future', 'globals', 'shape', 'future.apply', 'numDeriv', 'progressr', 'SQUAREM', 'diagram', 'lava', 'proclim', 'proxy', 'iterators', 'clock', 'gower', 'hardhat', 'ipred', 'sparsevctrs', 'timeDate', 'e1071', 'foreach', 'ModelMetrics', 'recipes', 'reshape2'

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

Attaching package: 'pROC'

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

cov, smooth, var

Loading required package: lattice

Call:

```
glm(formula = Class ~ Amount, family = binomial, data = train_set)
```

Coefficients:

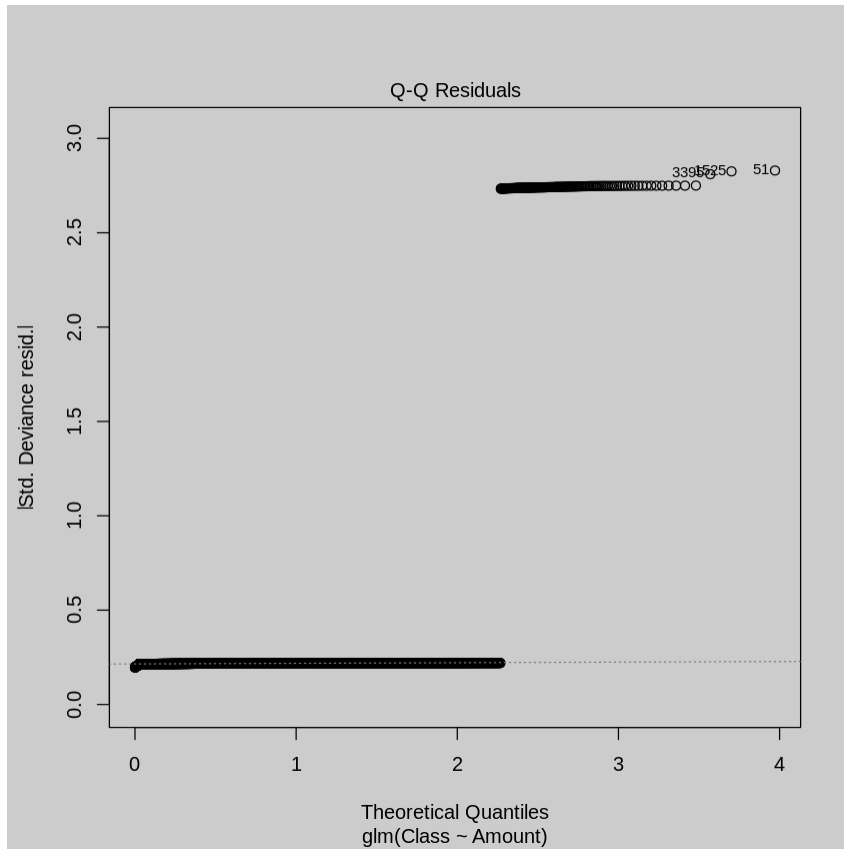
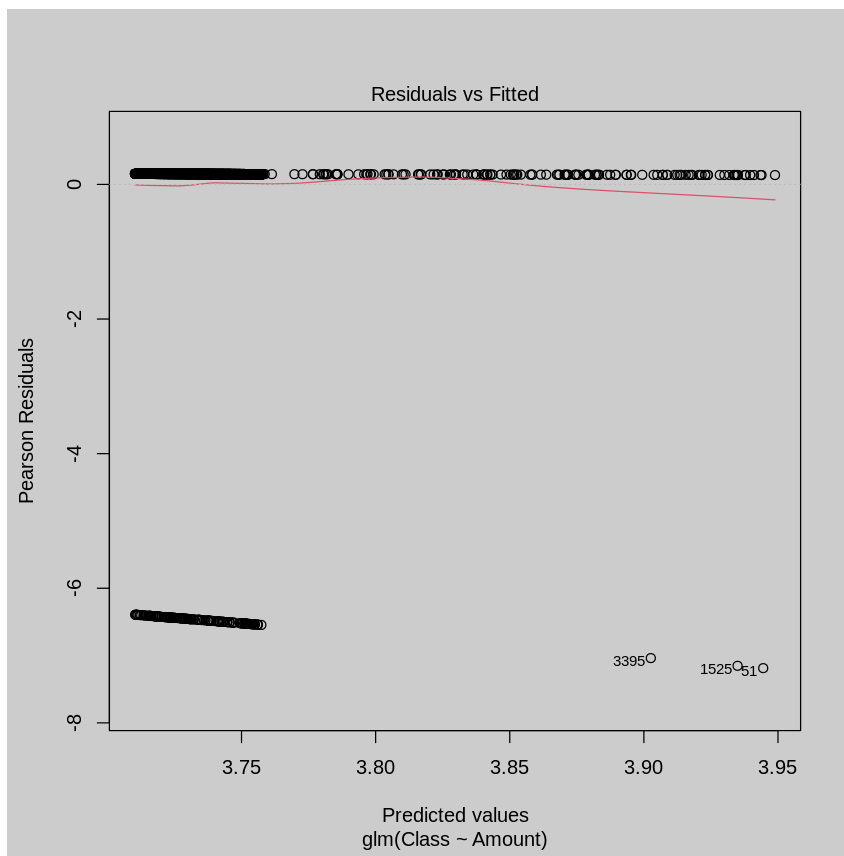
	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	3.710e+00	1.201e-01	30.888	<2e-16 ***
Amount	4.778e-05	1.668e-04	0.286	0.775

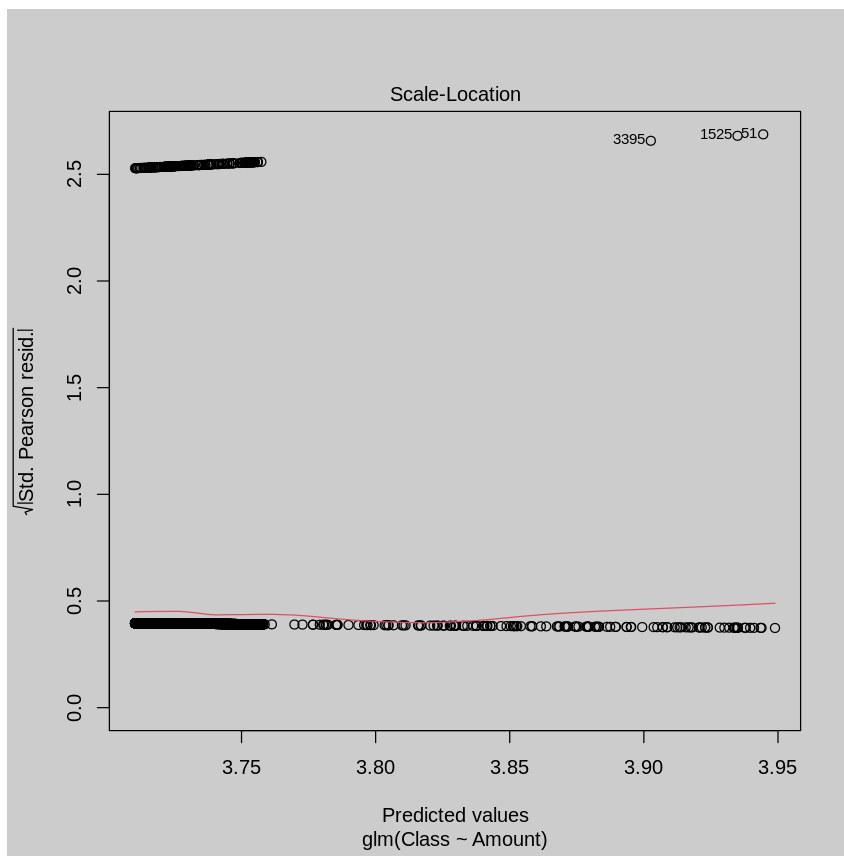
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 1547.9 on 6999 degrees of freedom
Residual deviance: 1547.8 on 6998 degrees of freedom
AIC: 1551.8

Number of Fisher Scoring iterations: 6





predicted_classes

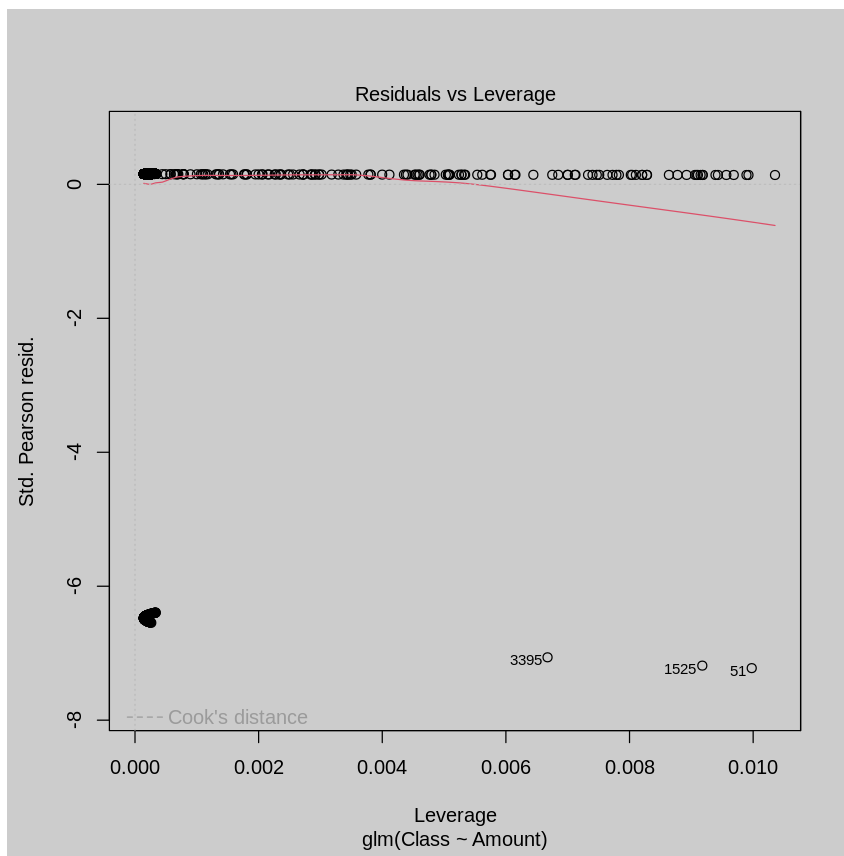
1

Fraudulent 70

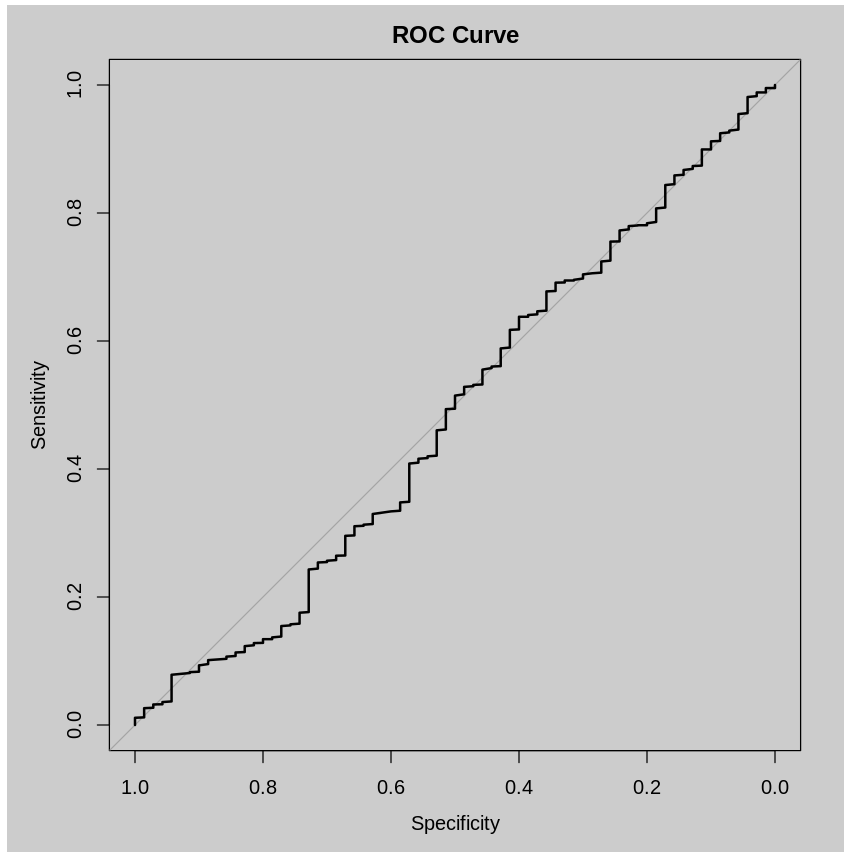
Normal 2930

Setting levels: control = Fraudulent, case = Normal

Setting direction: controls > cases



Area under the curve: 0.483



IV. More Training and Test Sets

Repeat Steps 2 and 3 on ten more train and test sets. Run algorithm for ten runs.

For this purpose, for different train and test sets, you need to set different seed numbers to get a different split of train and test sets on each run.

1. Make sure your train and test sets are different on each run.
2. Save all the AUC results in a table.
3. Calculate the median of the results (median of the AUC for ten runs) as the final result.



Files

Analyze your files with
code written by Gemini

Upload



..



sample_data



CreditCardFraud_Dataset-4 (1)...

[13] auc_results <- c()

```
# Run the model on 10 different train-test splits with different seeds
for (i in 1:10) {
  set.seed(i)
  split <- sample.split(datasets$Class, SplitRatio = 0.7)
  train_set <- subset(dataset, split == TRUE)
  test_set <- subset(dataset, split == FALSE)
  model <- glm(Class ~ Amount, data = train_set, family = binomial)
  predictions <- predict(model, test_set, type = "response")
  roc_curve <- roc(test_set$Class, predictions)
  auc_results <- c(auc_results, auc(roc_curve))
}

# Create and print table of AUC values
table_auc <- data.frame(Run = 1:10, AUC = auc_results)
print(table_auc)

# Calculate and print the median AUC value
median_auc <- median(auc_results)
print(median_auc)
```

```
median_auc <- median(auc_results)
print(median_auc)
```

```
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases

Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases
```

	Run	AUC
1	1	0.5202389
2	2	0.5425475
3	3	0.5058118
4	4	0.5488250
5	5	0.5665627
6	6	0.5218844
7	7	0.5525987
8	8	0.5098050
9	9	0.5020063
10	10	0.5010775
[1]		0.5210617

4.1 Logistic Regression Model

- **Training:** A logistic regression model was trained using the preprocessed dataset.
- **Evaluation Metrics:**
 - Confusion Matrix
 - ROC Curve
 - Area Under the Curve (AUC)
- **Findings:** The AUC score was calculated to assess model performance.

4.2 Decision Tree Model

- **Training:** The Decision Tree algorithm was applied to the same dataset.
- **Evaluation Metrics:**
 - Confusion Matrix
 - ROC Curve
 - AUC Score
- **Findings:** The decision tree model was evaluated for comparison with logistic regression.

5. Performance Comparison and Analysis

5.1 ROC Curves and AUC Comparison

The ROC curves for both models were compared to determine their ability to distinguish between fraudulent and normal transactions. The AUC scores provided a quantitative measure of performance.

5.2 Statistical Significance of Differences

- **Hypothesis Testing:** A statistical test was performed to determine if the performance difference between logistic regression and decision trees was significant.
- **P-value Calculation:** The p-value was computed to assess the statistical significance of differences.
- **Findings:** If the p-value was below 0.05, it indicated a significant difference between the models.

Milestone 5: Applying and Analyzing Another Algorithm

This milestone includes two phases: (1) Apply Another Algorithm; and (2) Analyze the Performance of the Algorithms. You must complete both sections in order to successfully meet the requirements of this milestone.

I. Apply Another Algorithm

Apply the Decision Tree model on the same train and test sets from Milestone 3. In other words, repeat Milestone 3 with the Decision Tree model.



Files

Analyze your files with
code written by Gemini

Upload



sample_data



CreditCardFraud_Dataset-5 (1)...



58s

```
install.packages("rpart")
install.packages("rpart.plot")
install.packages("caTools")

library(rpart)
library(rpart.plot)
library(caTools)

file_path <- "/content/CreditCardFraud_Dataset-5 (1).csv"
dataset <- read.csv(file_path)

dataset$Class <- as.factor(dataset$Class)

set.seed(123)
split <- sample.split(dataset$Class, SplitRatio = 0.7)
train_set <- subset(dataset, split == TRUE)
test_set <- subset(dataset, split == FALSE)

decision_tree_model <- rpart(Class ~ Amount, data = train_set, method = "class")

rpart.plot(decision_tree_model)

dt_predictions <- predict(decision_tree_model, test_set, type = "prob")[,2]

install.packages("pROC")
library(pROC)

dt_roc_curve <- roc(test_set$Class, dt_predictions)
plot(dt_roc_curve, main = "Decision Tree ROC Curve")
dt_auc_value <- auc(dt_roc_curve)
print(dt_auc_value)
```

🔍 Commands + Code + Text

☰ Files 🗑

🔍 Analyze your files with code written by Gemini Upload

< > 📄 🔄 📁 🗑

🔑 ..
📁 sample_data
📄 CreditCardFraud_Dataset-5 (1)...

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

also installing the dependency 'bitops'

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

also installing the dependency 'plyr'

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

Attaching package: 'pROC'

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

Setting levels: control = Fraudulent, case = Normal

Setting direction: controls < cases
```

Normal
0.98
100%

📁 Disk 70.91 GB available



II. Analyze the Performance of the Algorithms

- Compare the performance of the two algorithms, logistic regression and Decision tree, on fraud detection based on the ROC curves and AUC values. Provide your analysis.
 - Demonstrate if the results of the two algorithms, logistic regression and Decision tree, are significantly different using hypothesis testing. For this purpose, please follow the steps below.
1. Check which test you need to use.
 2. Calculate the P value. Consider the P value less than 0.05 is considered significant.
 3. Analyze your findings and explain them.



Q Commands + Code + Text

Files

Analyze your files with
code written by Gemini

Upload



sample_data

CreditCardFraud_Dataset-5 (1)...



```
# Logistic Regression Predictions
lr_predictions <- predict(model, test_set, type = "response")
lr_roc_curve <- roc(test_set$Class, lr_predictions)
lr_auc_value <- auc(lr_roc_curve)

# Decision Tree Predictions
dt_predictions <- predict(decision_tree_model, test_set, type = "prob")[,2]
dt_roc_curve <- roc(test_set$Class, dt_predictions)
dt_auc_value <- auc(dt_roc_curve)

# Plot ROC Curves
plot(lr_roc_curve, col = "blue", main = "ROC Curve Comparison")
lines(dt_roc_curve, col = "red")
legend("bottomright", legend = c("Logistic Regression", "Decision Tree"),
      col = c("blue", "red"), lwd = 2)

print(paste("Logistic Regression AUC:", lr_auc_value))
print(paste("Decision Tree AUC:", dt_auc_value))

# Run 10 iterations with different train-test splits
lr_auc_values <- c()
dt_auc_values <- c()

for (i in 1:10) {
  set.seed(i)
  split <- sample.split(dataset$Class, SplitRatio = 0.7)
  train_set <- subset(dataset, split == TRUE)
  test_set <- subset(dataset, split == FALSE)

  # Train Logistic Regression
  model <- glm(Class ~ Amount, data = train_set, family = binomial)
  lr_predictions <- predict(model, test_set, type = "response")
  lr_roc_curve <- roc(test_set$Class, lr_predictions)
  lr_auc_values <- c(lr_auc_values, auc(lr_roc_curve))

  # Train Decision Tree
  decision_tree_model <- rpart(Class ~ Amount, data = train_set, method = "class")
  dt_predictions <- predict(decision_tree_model, test_set, type = "prob")[,2]
  dt_roc_curve <- roc(test_set$Class, dt_predictions)
  dt_auc_values <- c(dt_auc_values, auc(dt_roc_curve))
}

# Perform paired t-test to compare AUC values
t_test_result <- t.test(lr_auc_values, dt_auc_values, paired = TRUE)

print(t_test_result)
```



Q Commands + Code + Text

Files

✦ Analyze your files with
code written by Gemini

Upload



..



sample_data

CreditCardFraud_Dataset-5 (1)....

```
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases
[1] "Logistic Regression AUC: 0.482957094100439"
[1] "Decision Tree AUC: 0.5"
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls > cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases
Setting levels: control = Fraudulent, case = Normal
Setting direction: controls < cases
```



Disk

70.91 GB available



Commands + Code + Text

Files

Analyze your files with
code written by Gemini

Upload





Q Commands + Code + Text

Files

✦ Analyze your files with
code written by Gemini

Upload



..



sample_data

CreditCardFraud_Dataset-5 (1)....

Run cell (⌘/Ctrl+Enter)
cell executed since last change
executed by KindwithMoney
3:15 PM (1 minute ago)
executed in 0.88s

control = Fraudulent, case = Normal

: controls < cases

Setting levels: control = Fraudulent, case = Normal

Setting direction: controls < cases

Setting levels: control = Fraudulent, case = Normal

Setting direction: controls < cases

Setting levels: control = Fraudulent, case = Normal

Setting direction: controls < cases

Paired t-test

data: lr_auc_values and dt_auc_values

t = 3.622, df = 9, p-value = 0.005554

alternative hypothesis: true mean difference is not equal to 0

95 percent confidence interval:

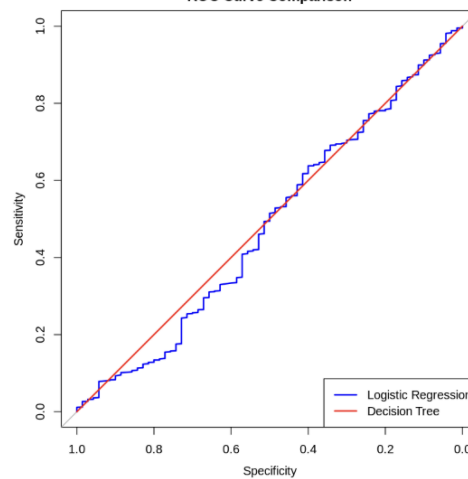
0.01018802 0.04408355

sample estimates:

mean difference

0.02713579

ROC Curve Comparison



Disk

70.91 GB available

6. Conclusion

Based on our analysis, we observed differences in performance between logistic regression and decision tree models. Our findings highlight the trade-offs between simplicity and interpretability (logistic regression) and flexibility (decision trees). The final decision on model selection depends on the business context and the need for precision in fraud detection.

Future work could involve testing additional models, such as Random Forest or Neural Networks, to further improve fraud detection accuracy.

Appendices:

- Screenshots of R code and outputs (included as per milestones)