Leveraging Data Science for Fraud Detection and Prevention

Group 3

Alexander Zhuk, Rebecca Chavez and Shivam Patel

Shiley-Marcos School of Engineering, University of San Diego

Master of Science, Applied Data Science

ADS-502: Data Mining

Dr. Ebrahim Tarshizi

April 14, 2025

Abstract

Fraudulent transactions pose significant financial and operational risks to businesses and consumers, making fraud detection a high priority for financial institutions. This project leverages machine learning techniques to develop an effective fraud detection system using a dataset of 100,000 credit card transactions. The dataset includes key attributes such as transaction time, amount, customer age, and entry mode (PIN, and CVC). We built and evaluated the performance of five classification models: Logistic Regression (baseline), C5.0 Decision Tree, Random Forest, Naive Bayes, and CART Decision Tree. The models were assessed using accuracy, precision, recall, F1-score, and ROC AUC. Among the five models tested, Random Forest demonstrated the highest overall performance, making it a strong candidate for real-world fraud detection by effectively applying the five predictive variables.

Chart of the project flow

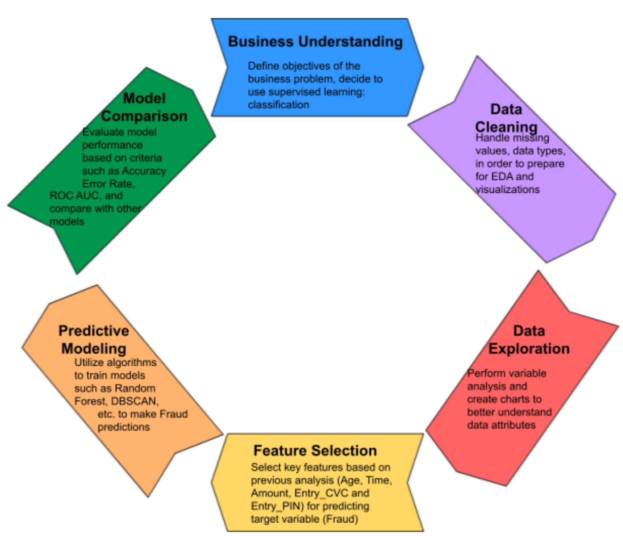


Table of Contents

Introduction

With the growing use of digital payments, credit card fraud has become a serious concern for both consumers and financial institutions. With the increase in credit card transactions comes more attempts to exploit these systems and users. The ability to accurately and quickly identify fraudulent transactions is essential to protecting users' sensitive information and maintaining trust between customers and companies. Data mining can offer powerful tools and algorithms for analyzing large amounts of transaction data to accurately identify fraud (Dal Pozzolo et al., 2018).

This paper evaluates a dataset containing data for 100,000 credit card transactions collected from October 13th to October 14th, 2020 (Verna, 2023). Each record includes 14 features related to the transaction and geographic details, which are described in Table 1.

Table 1

Features in Credit Card Transactions Dataset

Feature	Description
Time	The time that the transaction occurred (hour)
Type of Card	Visa, Mastercard, etc.
Entry Mode	Method that the card was submitted for the transaction (Tap, PIN, CVC, etc.)
Amount	Monetary amount transacted
Type of Transaction	Platform where the transaction occurred (ATM, Online, etc.)
Merchant Group	Type of service purchased by the transaction
Country of Transaction	Country where transaction occurred
Shipping Address	Country specified in the shipping information
Country of Residence	Country Specified in billing information
Gender	Gender of the cardholder
Age	Age of the cardholder
Bank	Issuing bank

Fraud	Whether fraud occurred in this transaction

To analyze the dataset, we built and evaluated the performance of five models: Logistic Regression (used as a baseline), C5.0 Decision Tree, Random Forest, Naive Bayes, and CART Decision Tree. These models were assessed using several metrics, including accuracy, precision, recall, F-1 score, and the Area Under the Curve of the Receiver Operating Characteristic (ROC AUC).

Methodology

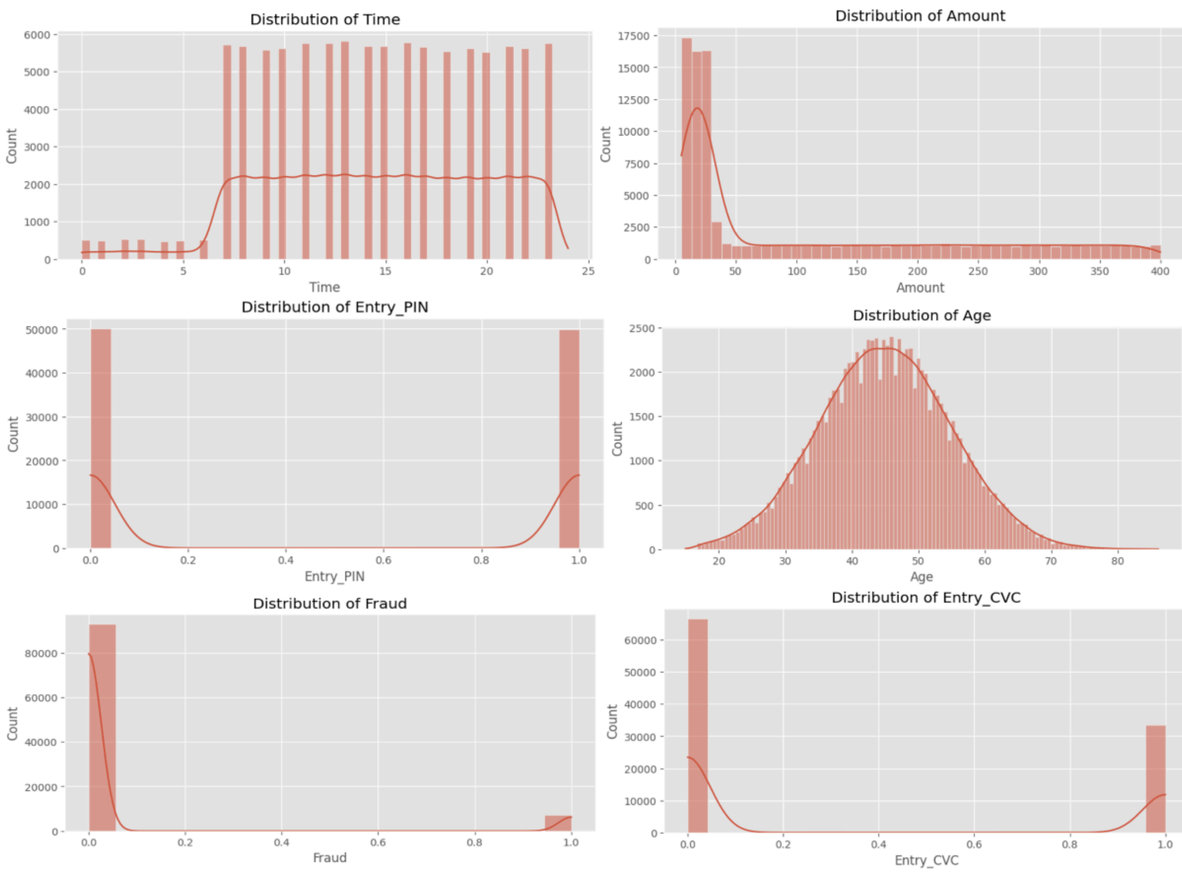
Data Preprocessing

To prepare the dataset for analysis, several preprocessing steps were performed. We began by checking for missing values across all 13 features. Transactions with missing values were removed to maintain the integrity of the overall dataset, resulting in 99,977 complete records. We then converted categorical variables into a numerical format using dummy binary variables. This allowed us to preserve the information kept in the categorical variables while using a format compatible with machine learning models. We made sure to drop one dummy variable for each set to reduce multicollinearity concerns.

Feature selection was influenced by domain knowledge and initial exploratory analysis. Features such as transaction time, amount, entry mode, and customer age were selected due to their potential for prediction. Other features were excluded to reduce dimensionality and excess noise. The distributions of the final features were investigated, which is seen in Figure 1. Once the features were selected, we then split the data into training and testing sets with an 80:20 ratio using Python's scikit-learn library.

Figure 1

Distribution of Selected Features



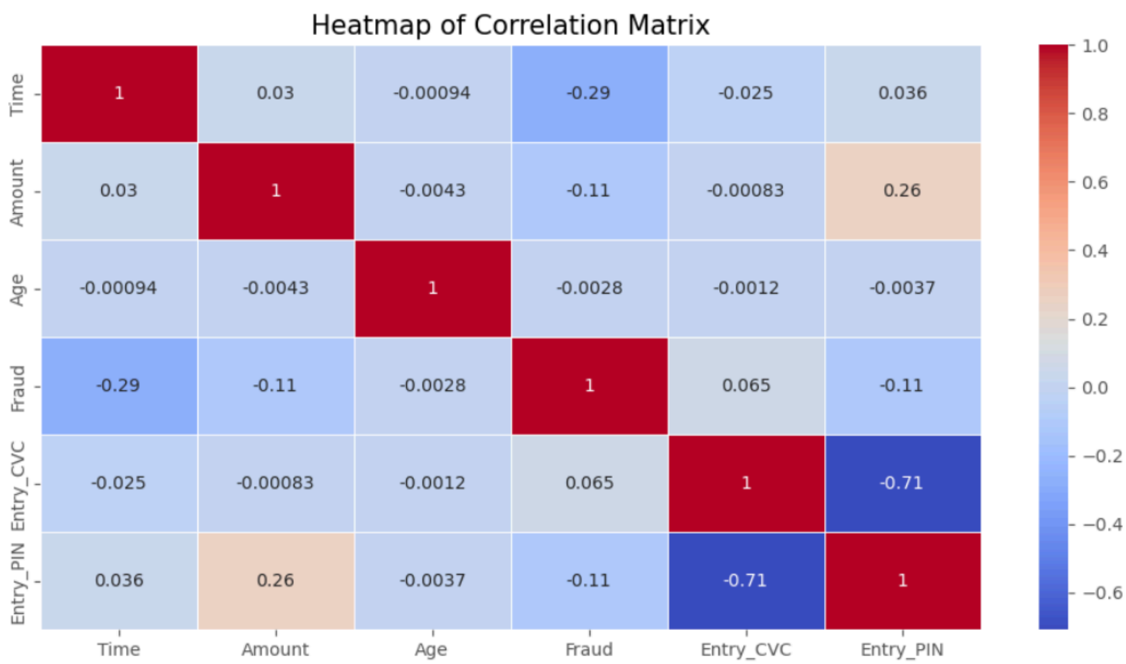
Feature Selection

Among the 13 features in the dataset, five features were selected for the fraud detection model: Transaction Time, Amount, Customer Age and Entry Mode (PIN, CVC). In order to identify relationships between numerical features (Transaction Time, Amount, Customer Age and Entry Mode (PIN, CVC) and the target variable (Fraud) we built a correlation matrix and visualized it through a heatmap (Figure 2).

The correlation heatmap in Figure 2 demonstrated a mild positive correlation (0.26) between Amount and Entry_PIN. Notably, a strong negative correlation (-0.71) was observed between Entry_CVC and Entry_PIN, suggesting a strong inverse relationship.

Figure 2

Correlation Heatmap of Credit Card Transactions Dataset



To further look into potential multicollinearity, Variance Inflation Factor (VIF) (Figure 3) will be introduced and a pairwise correlation will be examined (Figure 4). Such features as Time, Amount, Age, Entry_CVC and Entry_PIN show low VIFS, indicating weak multicollinearity.

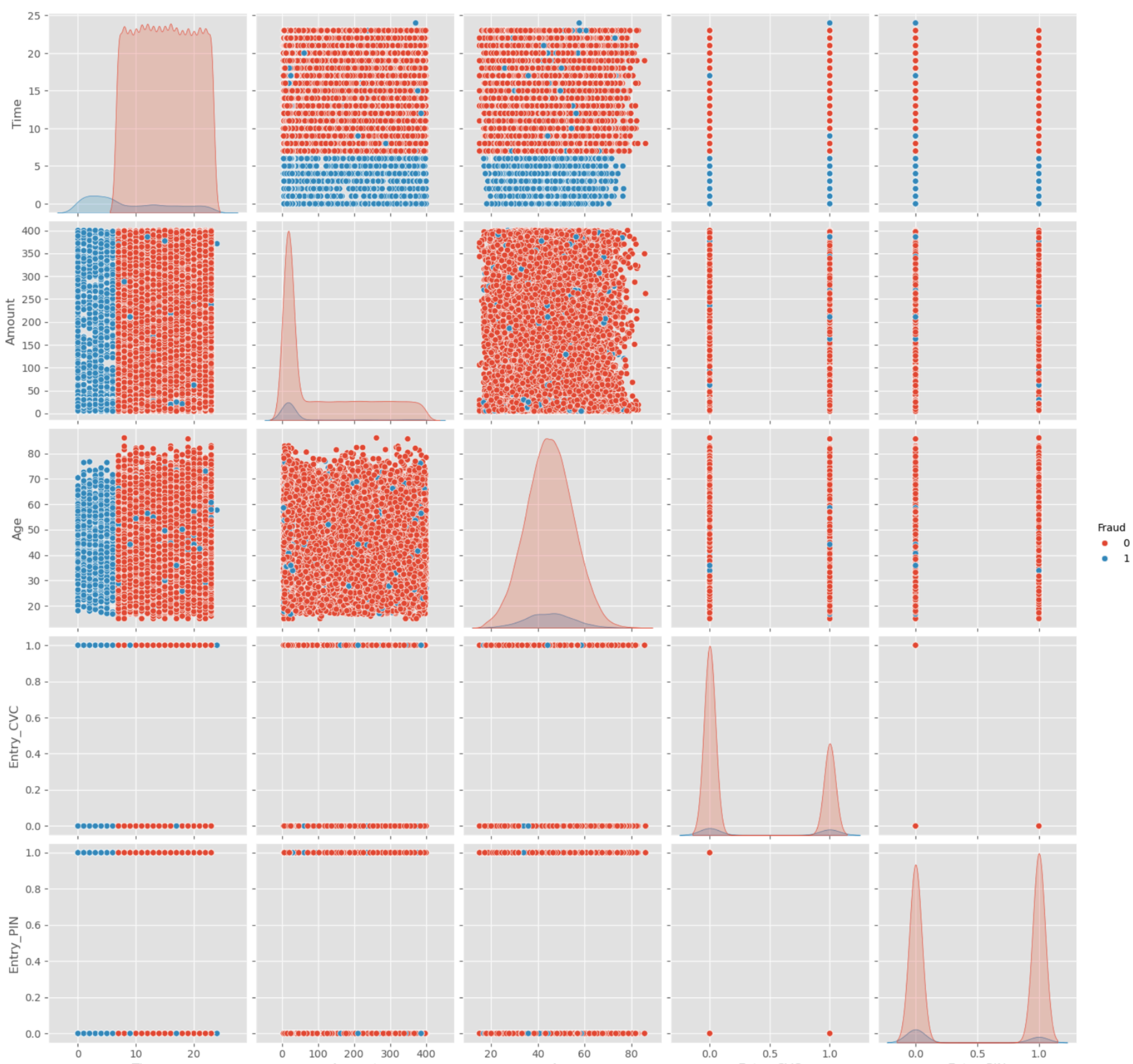
Figure 3

Variance Inflation Factors of Credit Card Transactions Dataset

	Feature	VIF
0	const	35.643625
1	Time	1.089331
2	Amount	1.158514
3	Age	1.000063
4	Fraud	1.109146
5	Entry_CVC	2.163116
6	Entry_PIN	2.322444

Figure 4

Pairplot of Credit Card Transactions Dataset



The pairplot of the Credit Card Transactions Dataset highlights minimal correlation for Time, Amount and Age features and confirms that Entry_CVC and Entry_PIN are dummy variables.

Cross Validation

In order to evaluate the models in this project, the dataset was partitioned into a training set and a test set. The dataset was split, where the training set contained 80% of the records, while the test set contained the remaining 20%. The test set is utilized alongside each model to determine the validation error rates. Two-sample t-tests were used on key numerical variables such as ‘Age’ and ‘Time’ to ensure the absence of a statistically significant difference between the training and test sets on these attributes.

Models

Logistic Regression

Logistic Regression is a probabilistic discriminative model widely used for classification problems. The following model is able to directly estimate the odds of data belonging to a particular class (Tan et al., 2018). An initial logistic regression model used in this project was created using such variables as Age, Time, Amount, Entry_CVC, and Entry_PIN. Logistic Regression was chosen as the baseline due its robustness and easy interpretability as shown in (1).

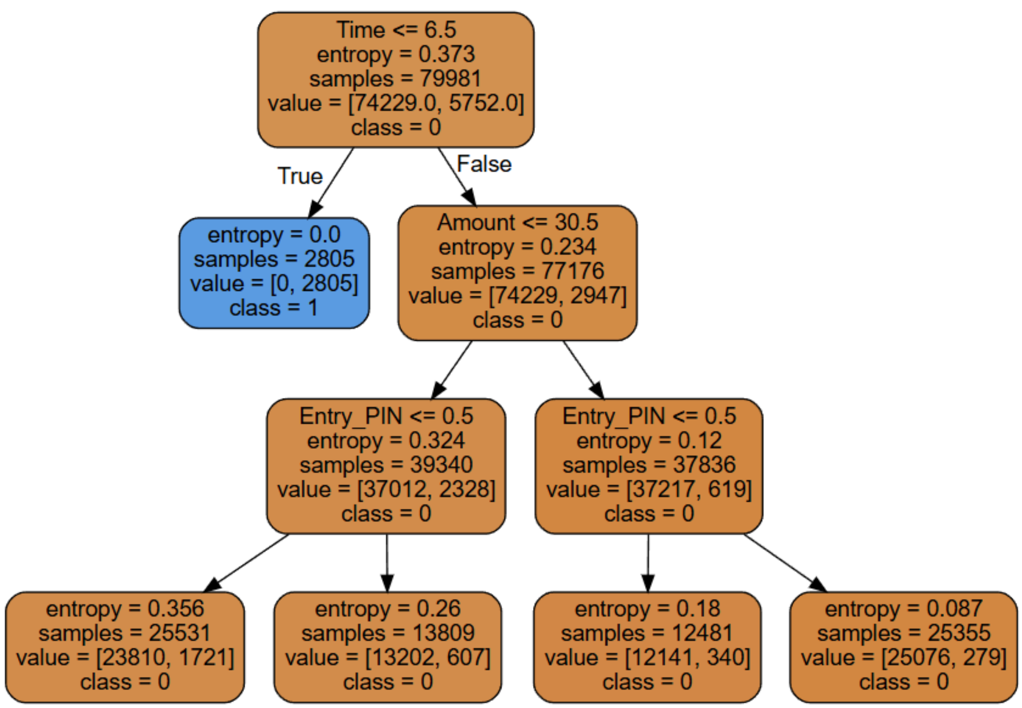
$$Fraud = 0.8904 - 0.0011Age - 0.2409Time - 0.0031Amount - 0.0096Entry\_CVC - 0.6380Entry\_PIN \tag{1}$$

C5.0

The C5.0 decision algorithm was performed with predictor variables Age, Time, Amount, Entry_CVC, and Entry_PIN by analyzing the information gain from each potential split. The root node split is based on whether the value for Time equals or is lower than 6.5. The child node for Time values lower than 6.5 terminates with 2,805 samples, while the majority fall into the second child node with 77,176 samples. Further node splits occur based on Amount value being less than or equal to $30.50, which results in two more child nodes, with relatively similar proportions of the remaining samples, about 50/50. The two child nodes split based on Entry_PIN and Entry_CVC respectively, resulting in a final four terminating nodes.

Figure 5

C5.0 Decision Tree with key predictor variables



Random Forest

The Random Forest technique was used to create decorrelated decision trees. This ensemble method has the characteristic of choosing its node-splitting criterion from a set of randomly selected attributes (Tan et al., 2019). This is beneficial for the project objective due to the increases in diversity between the ensemble of decision trees that contribute to the final prediction, considering both weak and strong attributes for every internal node. In addition to this innate decorrelation, the multiple, uncorrelated decision trees reduce the risk of overfitting the model to the training dataset. The predictions of the Random Forest model are shown in Table 2 below.

Table 2

Random Forest algorithm Predicted values alongside the actual values

	Predicted: 0	Predicted:1
Actual: 0	18427	129
Actual: 1	700	740

Naive Bayes

A Naive Bayes classification model was run with the predictor variables Age, Time, Amount, and target variable Fraud from the training dataset. The resulting model was then run using the test dataset in order to generate class predictions for the test instances. The Naive Bayes classifier can naturally handle missing values while still computing their conditional probability estimates in the training set; it can also deal with missing values in the test set by only using the existing feature values when calculating the posterior probabilities (Tan, et al., 2019). However, the variables selected to be used in the model were chosen with lower correlation in order to maintain the performance of the classifier. Upon evaluating the model by comparing the predicted values to the actual values, the model suffered from low accuracy, precision, and specificity, but displayed a relatively high sensitivity.

Table 3

Naive Bayes algorithm Predicted values alongside the actual values

	Predicted: 0	Predicted:1
Actual: 0	8279	10277
Actual: 1	303	1137

DBSCAN

In addition to the previous models, a DBSCAN clustering algorithm was utilized to highlight high density regions of points separated by areas of low density. During the overall process, the points are considered core, border or noise points. Noise points are eliminated, making DBSCAN useful even with noise present in the dataset (Tan et al., 2019). Due to this, dealing with points that do not fit with any of the defined clusters is achieved through use of the DBSCAN algorithm. For this algorithm, the dataset was scaled using the Standard Scaler from the sklearn package, and sampled with a size of n = 10, 000 to accommodate computation limits. An elbow graph was created to determine a more optimal value for the Eps parameter for the DBSCAN algorithm, with a Minpts value of 5. The code for the DBSCAN algorithm was referenced from the ‘Cluster Analysis Python IPYNB’ File from the Canvas page within the

ADS 502: Applied Data Mining course. For the elbow graph, scaling, and sampling code, the article by Kumar (2024) was referenced.

Figure 6

Elbow graph for determining Eps value

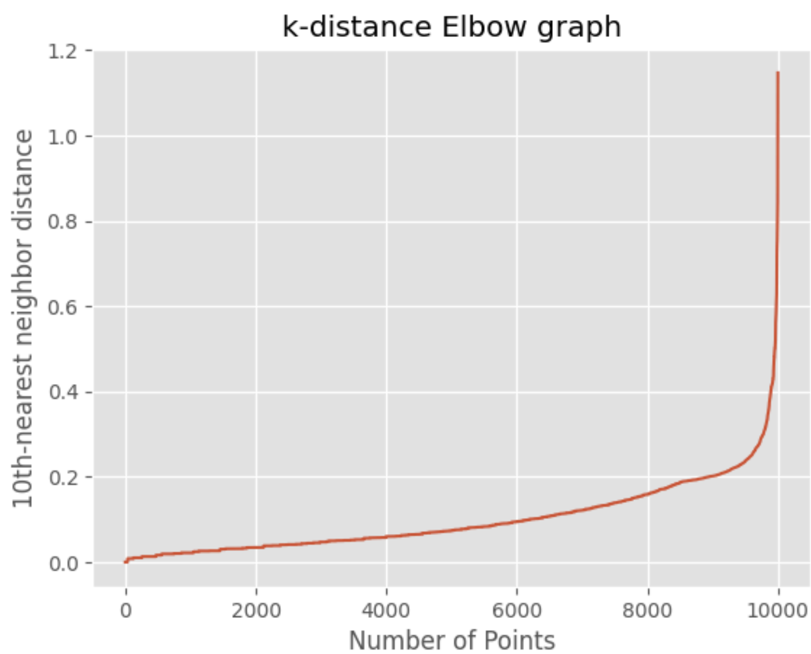
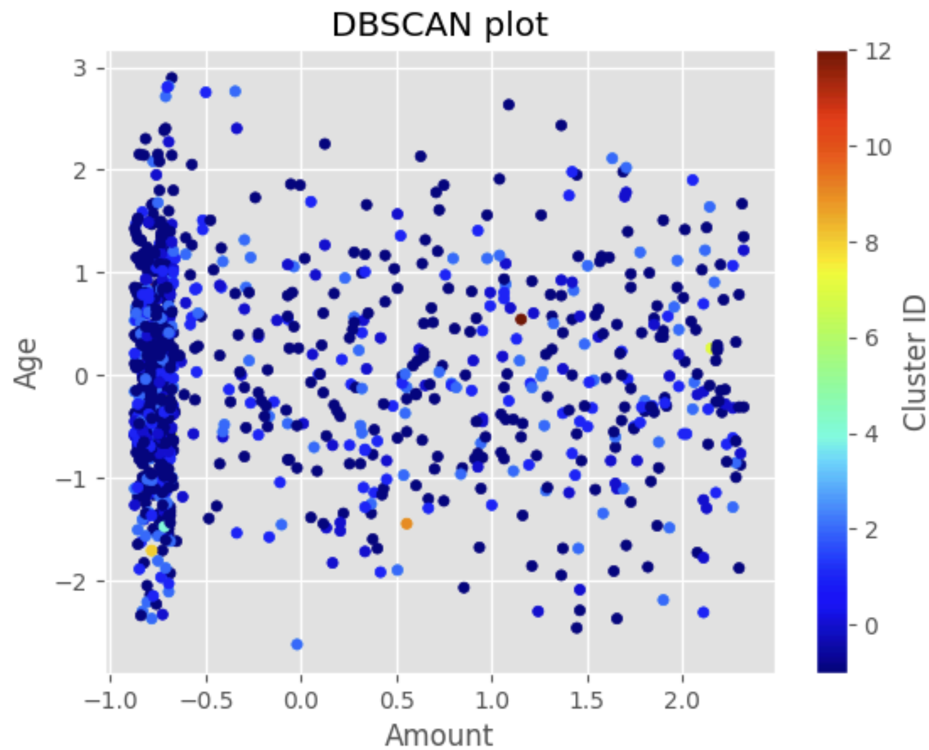


Figure 7

DBSCAN plot sampled and with standard scaling



Results

In order to evaluate the performance of our fraud detection models we implemented metrics widely used in offering insights. Accuracy, calculated as (True Positives + True Negatives) / (True Positives + False Positives + True Negatives + False Negatives), gives us the overall proportion of correct predictions. Recall, calculated as (True Positives)/(True Positives+False Negatives), allows us to capture actual fraud cases as correctly as possible. Precision, calculated as (True Positives)/(True Positives+False Positives) demonstrates how many of the fraud cases were actually fraudulent as well as Specificity, calculated as (True Negative)/(False Positive+True Negative) helps correctly identify fraudulent transactions. The F1 score is used to balance recall and precision, whereas ROC AUC displays an overall view on how the model is able to classify.

Model Comparison

Models evaluated were Logistic Regression, C5.0, Random Forest, Naive Bayes and CART. Random Forest displayed a high Accuracy of 95.94%, Recall of 51.39%, as well as Precision of 86.85% and Specificity of 99.40%. Random Forest's F1 score of 0.6457 and ROC AUC score of 0.6457 highlight its robustness despite imbalance in the dataset. C5.0 and CART models' Accuracy of 96.45% and 96.45% was better compared to Random Forest's, however showed lower Recall of 50.76% and 50.76%. Our Logistic Regression had a very low Recall of 15.21% meaning it failed to capture actual fraud cases as correctly as possible. Naive Bayes's improved Recall metric of 78.86% came at the expense of Specificity as it was only able to identify 44.62% of True Negative non-fraudulent transactions.

We selected Random Forest as our model of choice despite C5.0 and CART having a higher Accuracy of 96.45%, because Random Forest navigates between different classes and detects fraudulent transactions better. This highlights that Random Forest is better equipped to detect fraud in real world scenarios.

Table 4

Evaluation Metrics

	Evaluation Measure	Logistic Regression (Baseline)	C5.0	Random Forest	Naive Bayes	CART
0	Accuracy	0.9389	0.9645	0.9594	0.4709	0.9645
1	Error Rate	0.0611	0.0355	0.0406	0.5291	0.0355
2	Recall	0.1521	0.5076	0.5139	0.7896	0.5076
3	Precision	1.0000	1.0000	0.8685	0.0996	1.0000
4	Specificity	1.0000	1.0000	0.9940	0.4462	1.0000
5	F1	0.2640	0.6734	0.6457	0.1769	0.6734
6	ROC AUC	0.7817	0.7538	0.7539	0.6179	0.7538

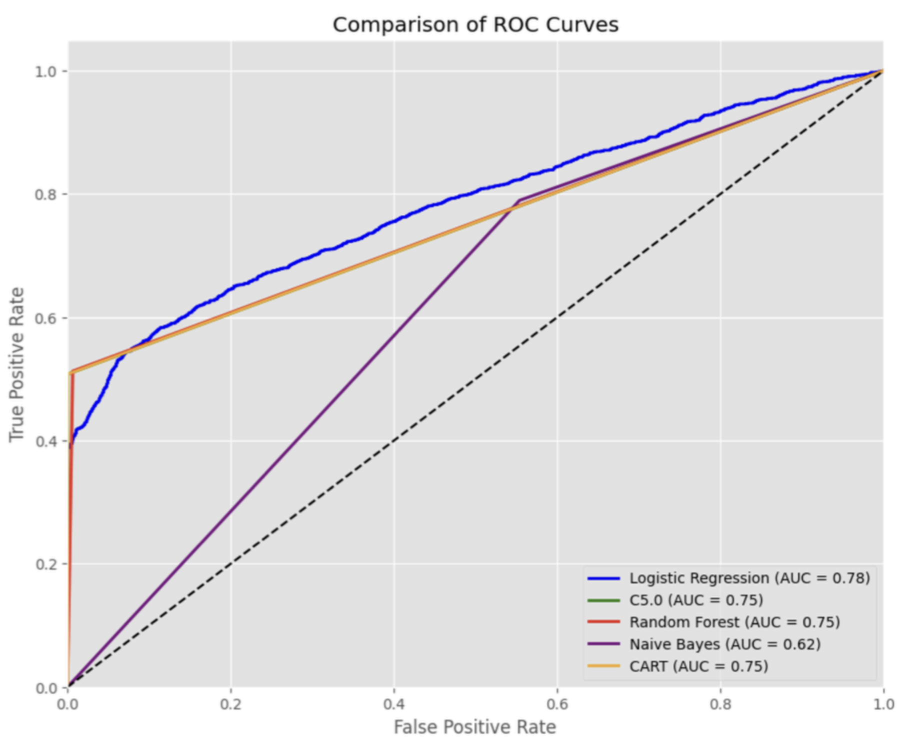
ROC Curves for the Models

The Receiver Operating Characteristic (ROC) curve is a graphical tool used for the evaluation of a classifier's performance. One of the measures examined for this project will be the area under the ROC curve also known as AUC. "If the classifier is perfect, then its area under the ROC curve will be equal to 1. If the algorithm simply performs random guessing, then its area under the ROC curve will be equal to 0.5" (Tan et al., 2018, p 529).

Our selected model of choice Random Forest performed with AUC of 0.7539.

Figure 8

Receiver Operating Characteristic (ROC)



Conclusion

Our selected model of choice is Random Forest. Random Forest displayed a high Accuracy of 95.94%, Recall of 51.39%, Precision of 86.85% and Specificity of 99.40%. Random Forest's F1 score of 0.6457 and ROC AUC score of 0.7539 highlight its robustness despite imbalance in the dataset in which 7.1% of transactions were marked as fraudulent. While minimizing false alarms Random Forest navigates between different classes and detects fraudulent transactions better than C5.0 and CART. Random Forest is better equipped to detect fraud in real world scenarios.

References

Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., & Bontempi, G. (2018). Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy. *IEEE transactions on neural networks and learning systems*, *29*(8), 3784–3797. https://doi.org/10.1109/TNNLS.2017.2736643

Kumar, R. (2024, September 29). *A Guide to the DBSCAN Clustering Algorithm*. Datacamp.com; DataCamp. https://www.datacamp.com/tutorial/dbscan-clustering-algorithm

Tan, P.-N., Steinbach, M., Karpatne, A., & Kumar, V. (2019). *Introduction to data mining* (2nd ed.). Pearson.

Verma, A. (2023). Credit Card Fraud Transaction Data. https://www.kaggle.com/datasets/anurag629/credit-card-fraud-transaction-data/data

ADS502

April 14, 2025

[ ]:
```
#Imporing Libraries - Initiated by AZ 03/16/25 11:56AM
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
import plotly.graph_objects as go
from scipy import stats
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
import statsmodels.tools.tools as stattools
from sklearn.metrics import roc_auc_score, roc_curve
```

[ ]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
```

Explaratory Data Analysis (EDA)

[ ]:
```
#Import the data from the file into a Pandas Series object in Python. ␣
↪Initiated by AZ
url = "https://raw.githubusercontent.com/AZhuk30/Applied-Data-Mining/refs/heads/
↪main/CreditCardData.csv"
data = pd.read_csv(url)
print(data)
```

```
      Transaction ID       Date Day of Week  Time Type of Card Entry Mode  \
0          #3577 209  14-Oct-20   Wednesday    19         Visa        Tap
1          #3039 221  14-Oct-20   Wednesday    17   MasterCard        PIN
2          #2694 780  14-Oct-20   Wednesday    14         Visa        Tap
3          #2640 960  13-Oct-20     Tuesday    14         Visa        Tap
4          #2771 031  13-Oct-20     Tuesday    23         Visa        CVC
…                …         …           …   …          …          …
99995      #3203 892  13-Oct-20     Tuesday    22   MasterCard        Tap
99996      #3304 849  14-Oct-20   Wednesday    23   MasterCard        PIN
99997      #3532 129  13-Oct-20     Tuesday    11   MasterCard        PIN
99998      #3107 092  14-Oct-20   Wednesday    22         Visa        Tap
```

```
99999      #3400 711  14-Oct-20   Wednesday    16         Visa        PIN

      Amount Type of Transaction Merchant Group Country of Transaction  \
0         £5                 POS  Entertainment         United Kingdom
1       £288                 POS       Services                    USA
2         £5                 POS     Restaurant                  India
3        £28                 POS  Entertainment         United Kingdom
4        £91              Online    Electronics                    USA
…        …                   …            …                      …
99995    £15                 POS    Electronics         United Kingdom
99996     £7                 ATM       Children                 Russia
99997    £21                 ATM   Subscription         United Kingdom
99998    £25                 POS       Products         United Kingdom
99999   £226                 POS     Restaurant         United Kingdom

      Shipping Address Country of Residence Gender   Age      Bank  Fraud
0       United Kingdom       United Kingdom      M  25.2       RBS      0
1                  USA                  USA      F  49.6    Lloyds      0
2                India                India      F  42.2  Barclays      0
3                India       United Kingdom      F  51.0  Barclays      0
4                  USA       United Kingdom      M  38.0   Halifax      1
…                  …                  …    …    …     …      …
99995   United Kingdom       United Kingdom      F  53.8   Halifax      0
99996           Russia               Russia      M  45.0  Barclays      0
99997   United Kingdom       United Kingdom      F  46.5      HSBC      0
99998   United Kingdom       United Kingdom      M  48.2  Barclays      0
99999   United Kingdom       United Kingdom      M  31.7     Monzo      0

[100000 rows x 16 columns]
```

[ ]:
```
data = data.dropna()
```

[ ]:
```
#Conversion of Boolean to Binary
print(data['Entry Mode'].value_counts())
```

```
Entry Mode
PIN    49966
CVC    33470
Tap    16541
Name: count, dtype: int64
```

[ ]:
```
```

[ ]:
```
entry_dummies = pd.get_dummies(data['Entry Mode'], prefix='Entry')
data = pd.concat([data, entry_dummies], axis=1)
data = data.drop('Entry Mode', axis=1)
data = data.drop('Entry_Tap', axis=1)
```

```
data[['Entry_PIN', 'Entry_CVC']] = data[['Entry_PIN', 'Entry_CVC']].astype(int)
data
```

```
[ ]:       Transaction ID       Date Day of Week  Time Type of Card Amount  \
0           #3577 209  14-Oct-20   Wednesday    19         Visa     £5
1           #3039 221  14-Oct-20   Wednesday    17   MasterCard   £288
2           #2694 780  14-Oct-20   Wednesday    14         Visa     £5
3           #2640 960  13-Oct-20     Tuesday    14         Visa    £28
4           #2771 031  13-Oct-20     Tuesday    23         Visa    £91
…                 …          …           …   …     …      …
99995       #3203 892  13-Oct-20     Tuesday    22   MasterCard    £15
99996       #3304 849  14-Oct-20   Wednesday    23   MasterCard     £7
99997       #3532 129  13-Oct-20     Tuesday    11   MasterCard    £21
99998       #3107 092  14-Oct-20   Wednesday    22         Visa    £25
99999       #3400 711  14-Oct-20   Wednesday    16         Visa   £226

      Type of Transaction Merchant Group Country of Transaction  \
0                     POS  Entertainment         United Kingdom
1                     POS       Services                    USA
2                     POS     Restaurant                  India
3                     POS  Entertainment         United Kingdom
4                  Online    Electronics                    USA
…                    …             …                      …
99995                 POS    Electronics         United Kingdom
99996                 ATM       Children                 Russia
99997                 ATM   Subscription         United Kingdom
99998                 POS       Products         United Kingdom
99999                 POS     Restaurant         United Kingdom

      Shipping Address Country of Residence Gender   Age      Bank  Fraud  \
0       United Kingdom       United Kingdom      M  25.2       RBS      0
1                  USA                  USA      F  49.6    Lloyds      0
2                India                India      F  42.2  Barclays      0
3                India       United Kingdom      F  51.0  Barclays      0
4                  USA       United Kingdom      M  38.0   Halifax      1
…                  …                    …    …   …     …      …
99995   United Kingdom       United Kingdom      F  53.8   Halifax      0
99996           Russia               Russia      M  45.0  Barclays      0
99997   United Kingdom       United Kingdom      F  46.5      HSBC      0
99998   United Kingdom       United Kingdom      M  48.2  Barclays      0
99999   United Kingdom       United Kingdom      M  31.7     Monzo      0

       Entry_CVC  Entry_PIN
0              0          0
1              0          1
2              0          0
3              0          0
```

```
4               1          0
…          …          …
99995           0          0
99996           0          1
99997           0          1
99998           0          0
99999           0          1

[99977 rows x 17 columns]
```

[ ]:
```
data['Amount'] = data['Amount'].astype(str)
data['Amount'] = data['Amount'].str.replace(r'[^\d.]', '', regex=True)
data['Amount'] = data['Amount'].replace('', np.nan)
data['Amount'] = pd.to_numeric(data['Amount'], errors='coerce')

conf = data.select_dtypes(include=['float64', 'int64', 'int32']).columns.
 ↪tolist()
conf
```

[ ]:
```
['Time', 'Amount', 'Age', 'Fraud', 'Entry_CVC', 'Entry_PIN']
```

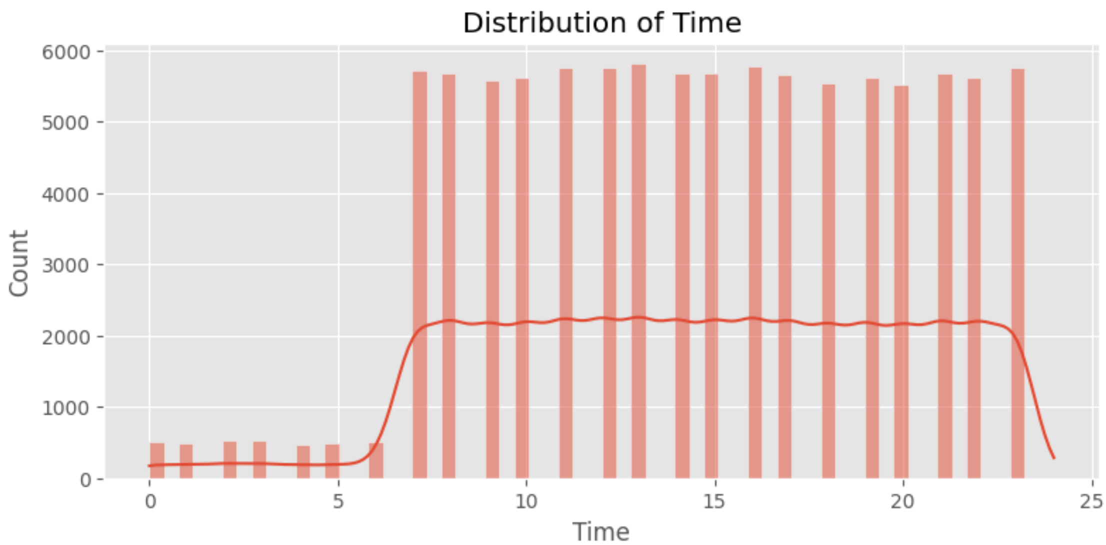
[ ]:
```
plt.style.use('ggplot')
```

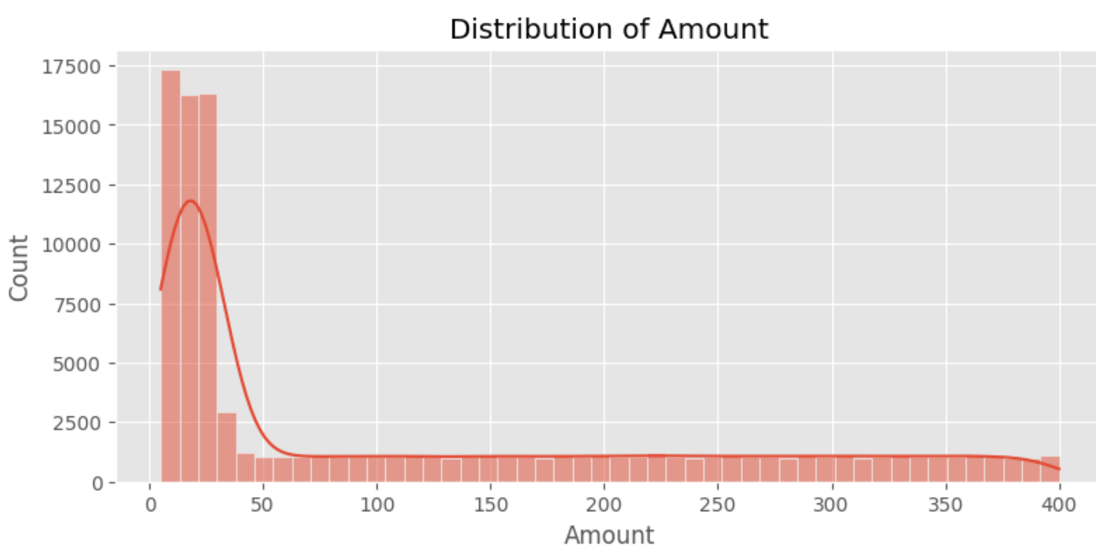
[ ]:
```
data.describe()
```

[ ]:
```
               Time        Amount           Age         Fraud     Entry_CVC  \
count  99977.000000  99977.000000  99977.000000  99977.000000  99977.000000
mean      14.563100    112.579933     44.993595      0.071937      0.334777
std        5.308202    123.435613      9.948121      0.258384      0.471915
min        0.000000      5.000000     15.000000      0.000000      0.000000
25%       10.000000     17.000000     38.200000      0.000000      0.000000
50%       15.000000     30.000000     44.900000      0.000000      0.000000
75%       19.000000    208.000000     51.700000      0.000000      1.000000
max       24.000000    400.000000     86.100000      1.000000      1.000000

          Entry_PIN
count  99977.000000
mean       0.499775
std        0.500002
min        0.000000
25%        0.000000
50%        0.000000
75%        1.000000
max        1.000000
```

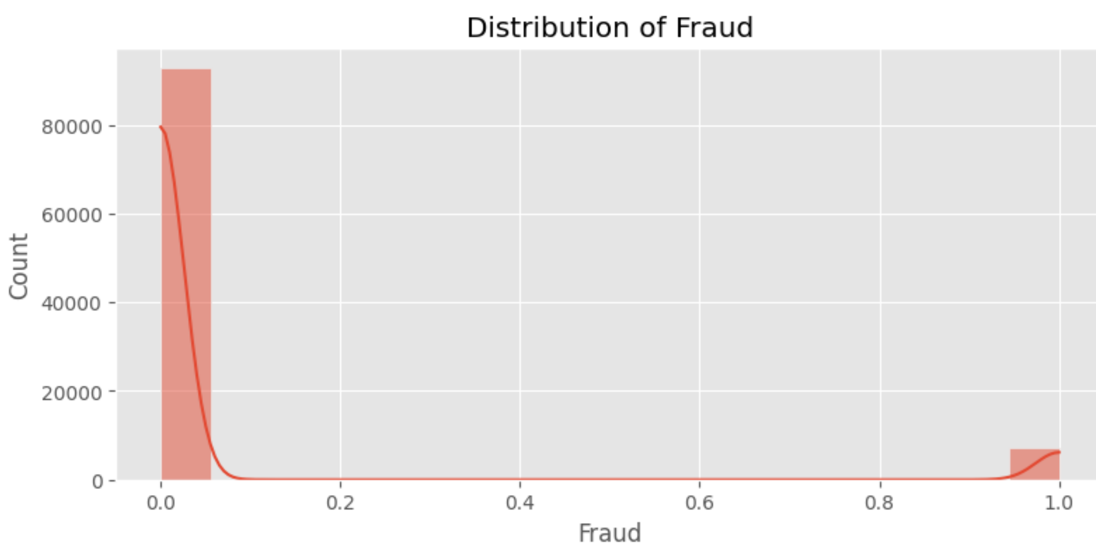
Univariate Analysis

[ ]:
```python
for column in conf:
  plt.figure(figsize=(20, 4))
  plt.subplot(1, 2, 1)
  sns.histplot(data[column], kde = True)
  plt.title(f'Distribution of {column}')
  plt.show()
```

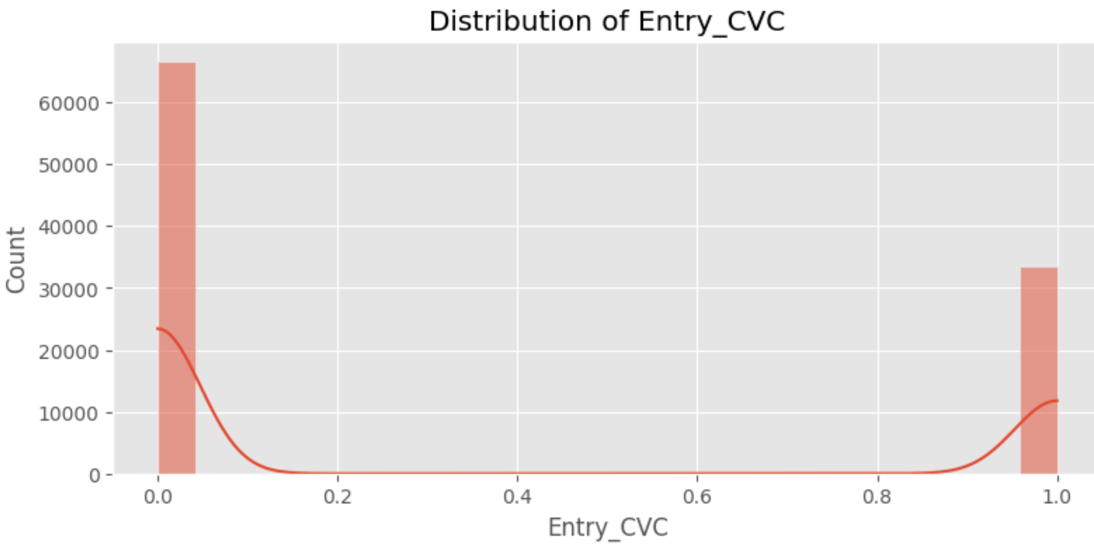


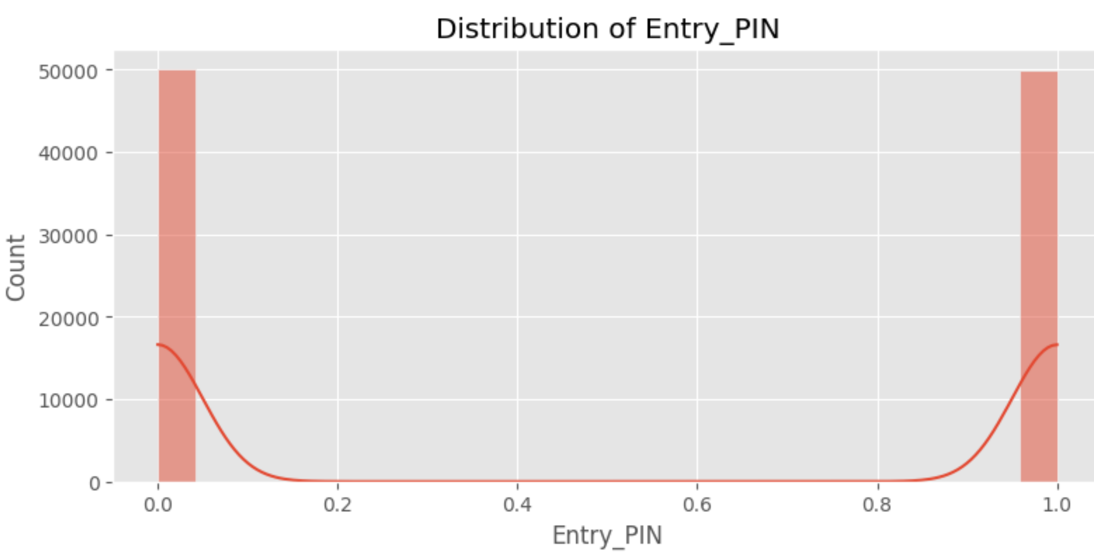


Distribution of Age
Count
Age
0
500
1000
1500
2000
2500
20
30
40
50
60
70
80



Distribution of Fraud
Count
Fraud
0
20000
40000
60000
80000
0.0
0.2
0.4
0.6
0.8
1.0

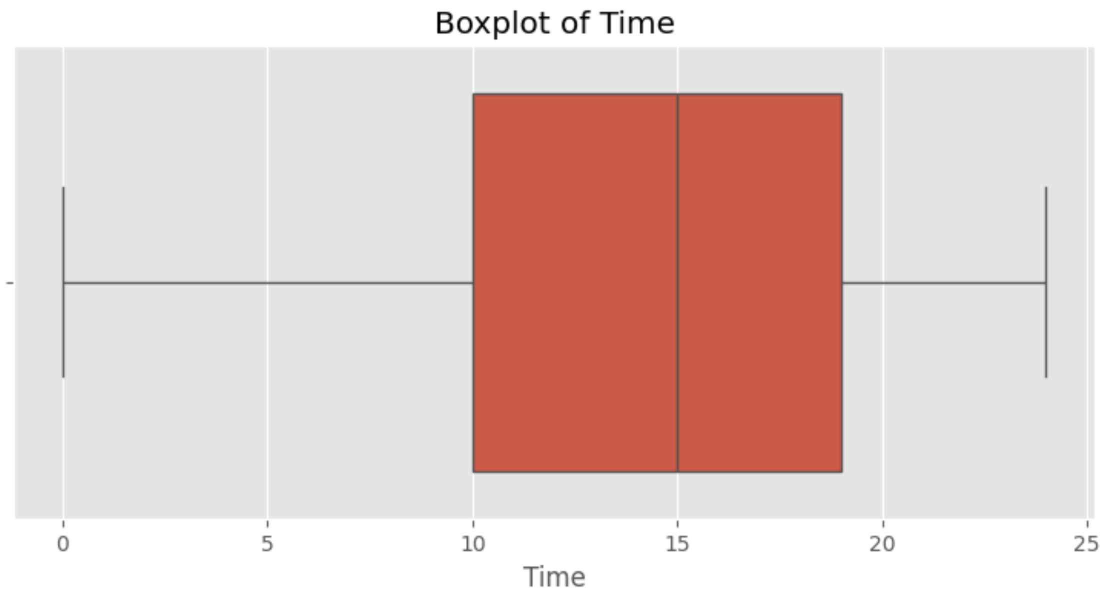


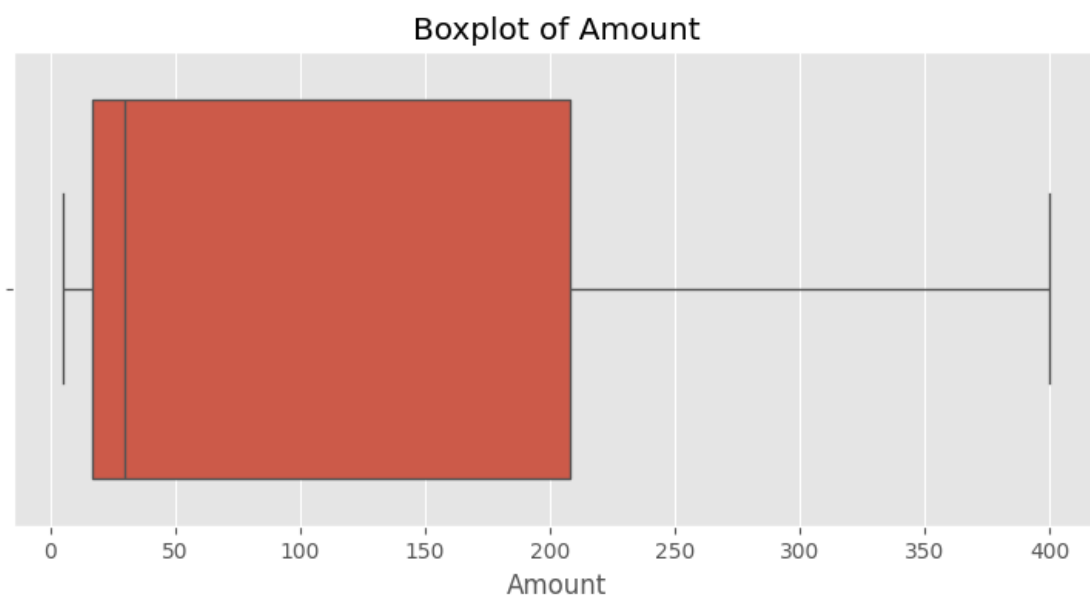


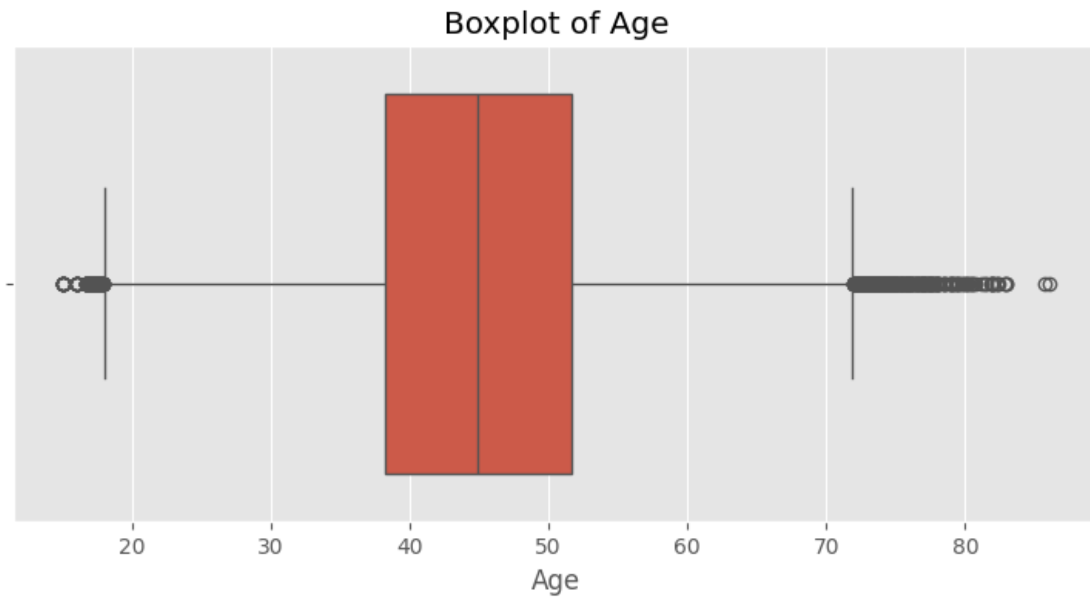
[ ]:
```
data['Amount'] = data['Amount'].astype(str)
data['Amount'] = data['Amount'].str.replace(r'[^\d.]', '', regex=True)
data['Amount'] = data['Amount'].replace('', np.nan)
data['Amount'] = pd.to_numeric(data['Amount'], errors='coerce')
numerical_columns = ["Time", "Amount", "Age", "Entry_CVC", "Entry_PIN", "Fraud"]
corr_matrix = data[numerical_columns].corr()
numeric_data = data.select_dtypes(include=['number'])
numeric_data

from matplotlib import pyplot as plt
```

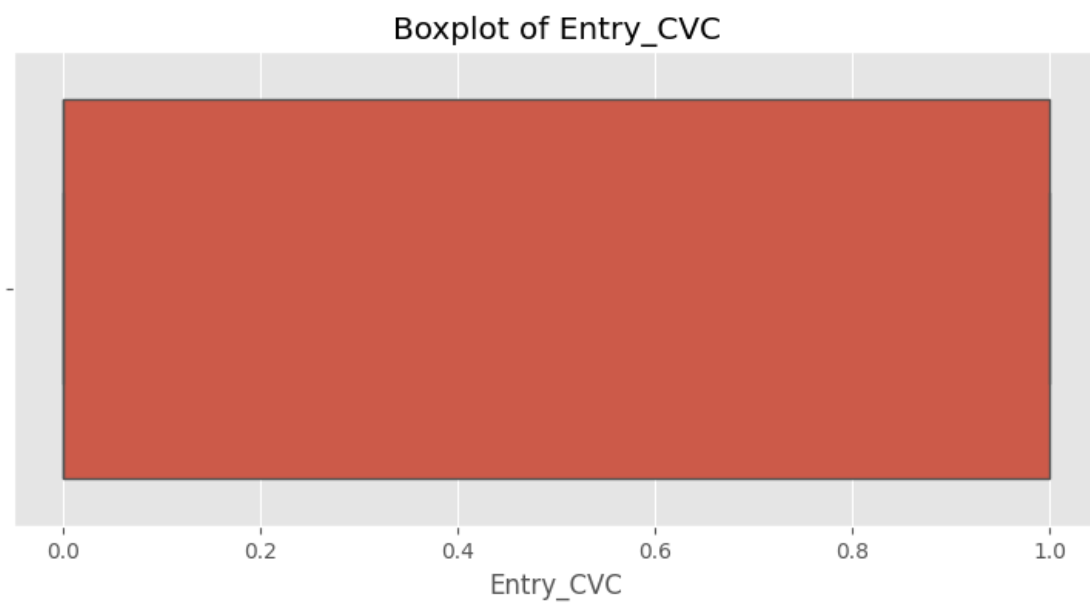
```
numeric_data['Fraud'].plot(kind='hist', bins=20, title='Fraud')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

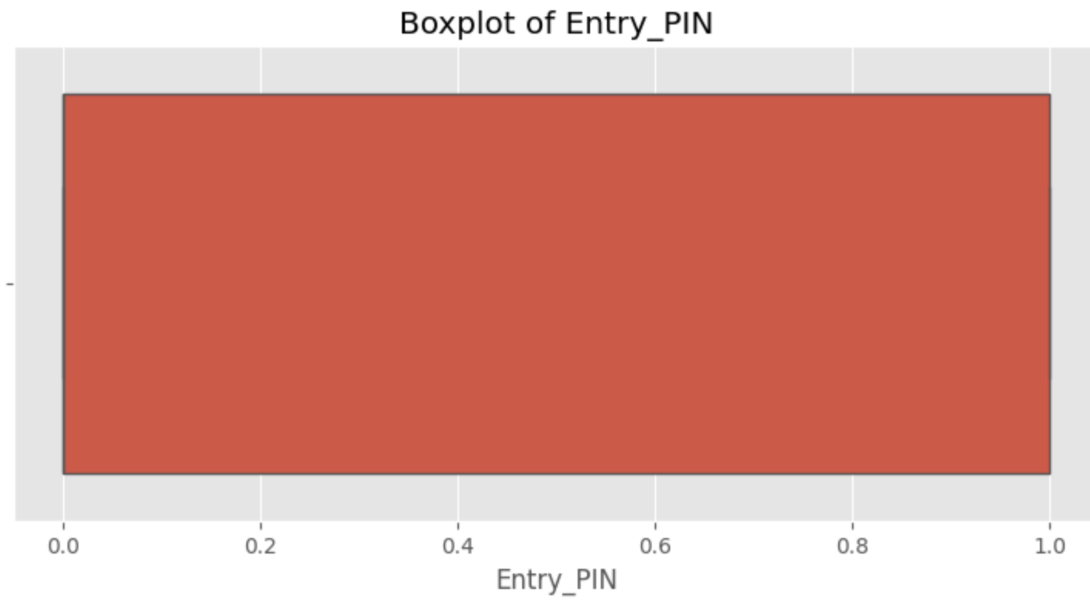
[ ]:
```
#Plot boxplots of all continuous features
plt.style.use('ggplot')
for column in conf:
  if column != 'Fraud':
    plt.figure(figsize=(20, 4))
    plt.subplot(1, 2, 1)
    sns.boxplot(x=data[column])
    plt.title(f'Boxplot of {column}')
    plt.show()
```


Boxplot of Time
0
5
10
15
20
25
Time


Boxplot of Amount
0
50
100
150
200
250
300
350
400
Amount


Boxplot of Age
20
30
40
50
60
70
80
Age


Boxplot of Entry_CVC
0.0
0.2
0.4
0.6
0.8
1.0
Entry_CVC



Multivariate Analysis

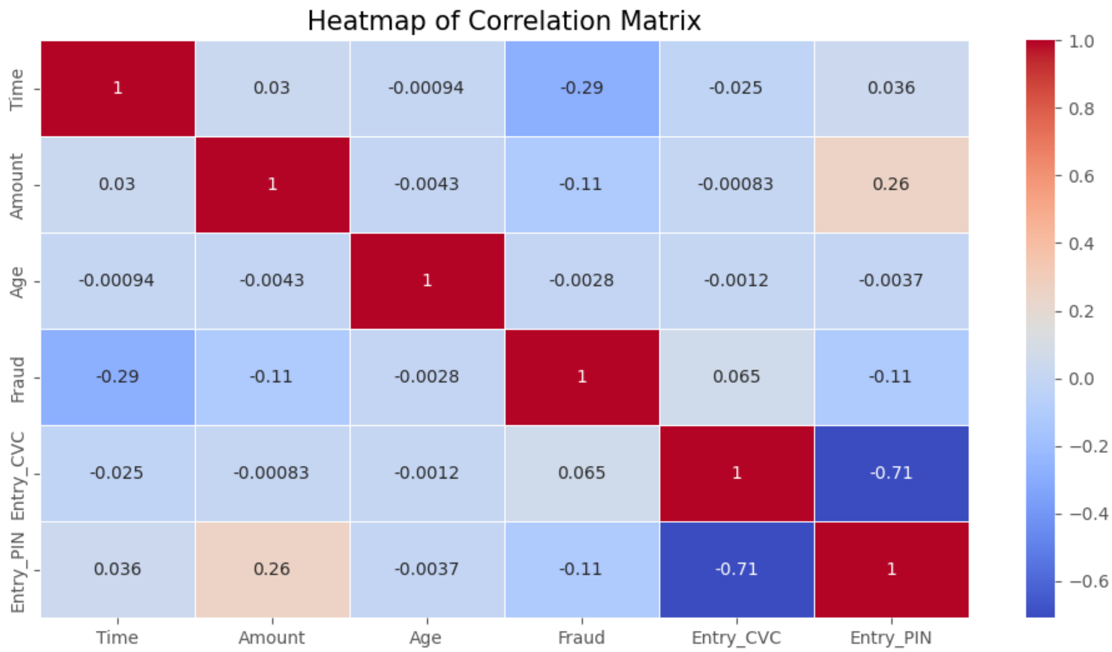
[ ]:
```
#Correlation Matrix
```

[ ]:
```
data['Amount'] = data['Amount'].astype(str)
data['Amount'] = data['Amount'].str.replace(r'[^\d.]', '', regex=True)
data['Amount'] = data['Amount'].replace('', np.nan)
data['Amount'] = pd.to_numeric(data['Amount'], errors='coerce')
numerical_columns = ["Time", "Amount", "Age", "Entry_CVC", "Entry_PIN", "Fraud"]
corr_matrix = data[numerical_columns].corr()
numeric_data = data.select_dtypes(include=['number'])
numeric_data
```

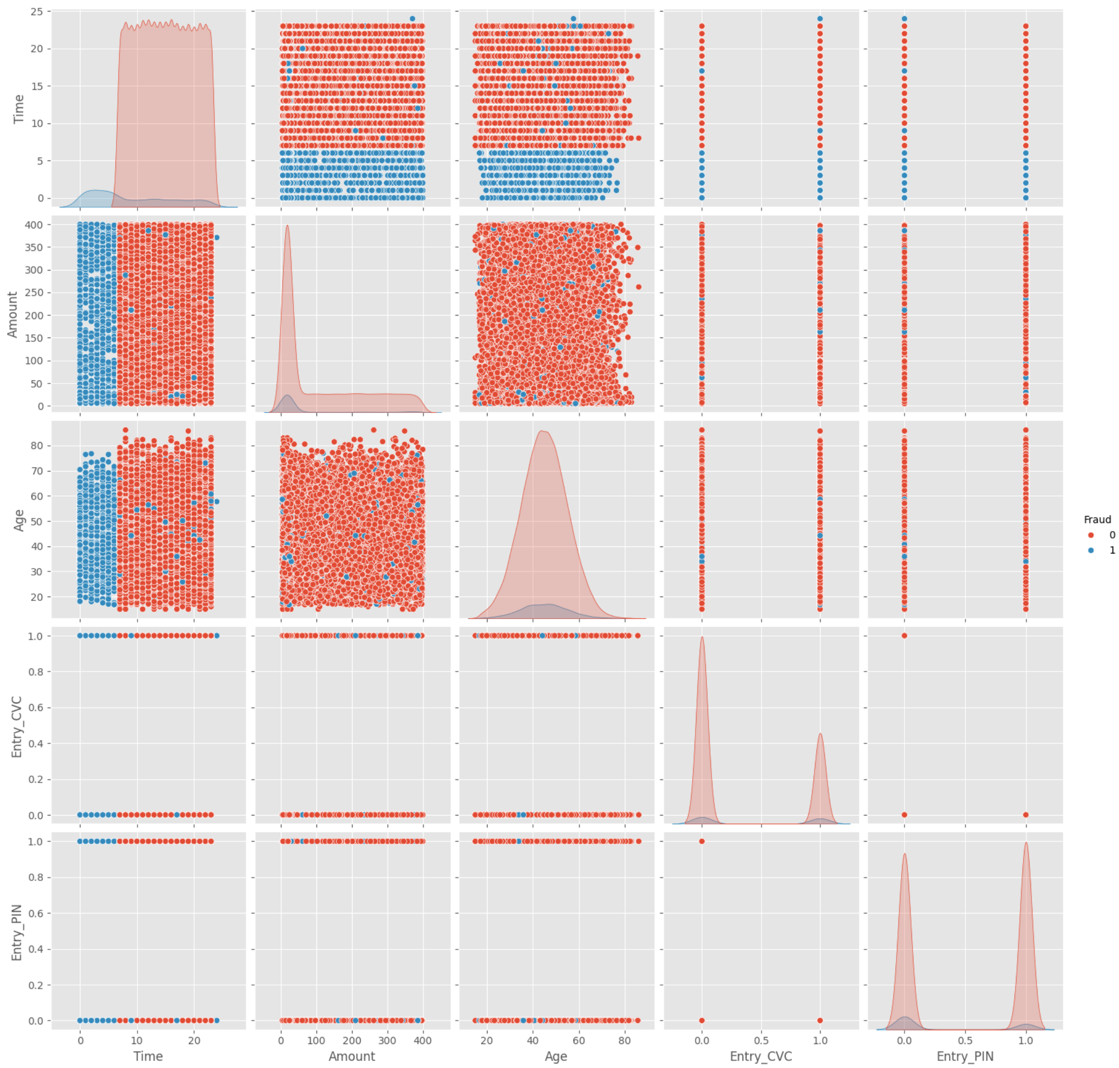
[ ]:

```
       Time  Amount   Age  Fraud  Entry_CVC  Entry_PIN
0        19       5  25.2      0          0          0
1        17     288  49.6      0          0          1
2        14       5  42.2      0          0          0
3        14      28  51.0      0          0          0
4        23      91  38.0      1          1          0
…        …       …     …      …          …          …
99995    22      15  53.8      0          0          0
99996    23       7  45.0      0          0          1
99997    11      21  46.5      0          0          1
99998    22      25  48.2      0          0          0
99999    16     226  31.7      0          0          1

[99977 rows x 6 columns]
```

[ ]:
```python
plt.figure(figsize=(12,6))
sns.heatmap(numeric_data.corr(), annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Heatmap of Correlation Matrix", fontsize=15)
plt.show()
```



[ ]:
```python
sns.pairplot(numeric_data, hue ="Fraud", height=3)
plt.show()
```



Train_test Split of the Data ("CreditCardData.csv")

```
[ ]: #Partitioned the data set, so that 80% of the records are included in the
      ↪training data set and and 20% are included in the test data set
     #Will use a bar graph to confirm the proportion - Initiated by AZ 03/30/2025 4:
      ↪56pm
     project_data_train, project_data_test = train_test_split(data, test_size=0.2,
      ↪random_state=7)
```

```
[ ]: x = ['Original Dataset', 'Training Data', 'Test Data']
     y = [data.shape[0], project_data_train.shape[0], project_data_test.shape[0]]
     fig = go.Figure(data=[go.Bar(x=x, y=y)])
     fig.update_layout(title_text='Confirming Split')
     fig.show()
```

```
[ ]: #Identified the total number of records in the training data set and how many␣
      ↪records in the training data set have a Fraud value of 1
     project_data_train.shape[0] #There are 80000 records in the training set -␣
      ↪Initiated by AZ 03/30/2025 4:56pm
```

```
[ ]: 79981
```

```
[ ]: bool(project_data_train.shape[0] == round(data.shape[0]*.8))
     project_data_train['Fraud'].value_counts() #- Initiated by AZ 03/30/2025 4:56pm
```

```
[ ]: Fraud
     0    74229
     1     5752
     Name: count, dtype: int64
```

```
[ ]: ratio = project_data_train['Fraud'].value_counts()/[1]/project_data_train.
      ↪shape[0] * 100 #- Initiated by AZ 03/30/2025 4:56pm
     ratio
```

```
[ ]: Fraud
     0    92.808292
     1     7.191708
     Name: count, dtype: float64
```

```
[ ]: #Identified the total number of records in the training data set and how many␣
      ↪records in the training data set have a Fraud value of 1
     project_data_train.shape[0] #There are 20000 records in the training set -␣
      ↪Initiated by AZ 03/30/2025 4:56pm
     bool(project_data_test.shape[0] == round(data.shape[0]*.2))
     project_data_test['Fraud'].value_counts() #- Initiated by AZ 03/30/2025 4:56pm
```

```
[ ]: Fraud
     0    18556
     1     1440
     Name: count, dtype: int64
```

```
[ ]: ratio = project_data_test['Fraud'].value_counts()/[1]/project_data_test.
      ↪shape[0] * 100 #- Initiated by AZ 03/30/2025 4:56pm
     ratio
```

```
[ ]: Fraud
     0    92.79856
     1     7.20144
     Name: count, dtype: float64
```

```
[ ]: #Validate partition of testing for the difference in mean of Age for the␣
      ↪training set versus the test set. - Initiated by AZ
```

[ ]:
```
print("Training Age Mean:", round(project_data_train['Age'].mean(),2)) #-
 ↪Initiated by AZ
print("Test Age Mean:", round(project_data_test['Age'].mean(),2))
```

```
Training Age Mean: 44.99
Test Age Mean: 45.0
```

[ ]:
```
stats.ttest_ind(project_data_train['Age'], project_data_test['Age']) #-
 ↪Initiated by AZ
```

[ ]:
```
TtestResult(statistic=np.float64(-0.1667300712495496),
pvalue=np.float64(0.8675827804689238), df=np.float64(99975.0))
```

The p value is not less than 0.05; so there is not a statistically significant difference in the two datasets on this variable. This validates that the training and test sets should be similar on this variable.

[ ]:
```
#Validate partition of testing for the difference in mean of Time for the
 ↪training set versus the test set. - Initiated by AZ
```

[ ]:
```
print("Training Time Mean:", round(project_data_train['Time'].mean(),2))
print("Test Time Mean:", round(project_data_test['Time'].mean(),2))
```

```
Training Time Mean: 14.58
Test Time Mean: 14.51
```

[ ]:
```
stats.ttest_ind(project_data_train['Time'], project_data_test['Time']) #-
 ↪Initiated by AZ
```

[ ]:
```
TtestResult(statistic=np.float64(1.5620960576376095),
pvalue=np.float64(0.11826852669817096), df=np.float64(99975.0))
```

The p value is not less than 0.05; so there is not a statistically significant difference in the two datasets on this variable. This validates that the training and test sets should be similar on this variable.

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.tools.tools as stattools
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import graphviz
from sklearn.tree import plot_tree
from sklearn import tree
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
```

```python
from sklearn.svm import SVC
from tabulate import tabulate
```

[ ]:
```python
ytrain = project_data_train[['Fraud']]
```

[ ]:
```python
#Converted column Amount into string, gott rid of currency symbol, made sure␣
↪empty strings are converted to Nan and finally convert to float
project_data_train['Amount'] = project_data_train['Amount'].astype(str)
project_data_train['Amount'] = project_data_train['Amount'].str.replace(r'[^\d.
↪]', '', regex=True)
project_data_train['Amount'] = project_data_train['Amount'].replace('', np.nan)
project_data_train['Amount'] = pd.to_numeric(project_data_train['Amount'],␣
↪errors='coerce')

project_data_test['Amount'] = project_data_test['Amount'].astype(str)
project_data_test['Amount'] = project_data_test['Amount'].str.replace(r'[^\d.
↪]', '', regex=True)
project_data_train['Amount'] = project_data_train['Amount'].replace('', np.nan)
project_data_test['Amount'] = pd.to_numeric(project_data_test['Amount'],␣
↪errors='coerce')
```

[ ]:
```python
#Will be using Age as one of the variables to predict Fraud
Xtrain = project_data_train[['Age', 'Time', 'Amount', 'Entry_CVC', 'Entry_PIN']]
```

[ ]:
```python
X_names = ["Age", "Time", "Amount", 'Entry_CVC', 'Entry_PIN']
```

[ ]:
```python
y_names =["0", "1"]
```

[ ]:
```python
Xtrain
```

[ ]:

	Age	Time	Amount	Entry_CVC	Entry_PIN
61689	62.6	20	240	0	1
12950	43.9	18	12	0	1
67886	55.6	21	98	0	1
24212	40.4	21	19	1	0
9198	53.9	14	126	0	1
…	…	…	…	…	…
53471	46.5	21	258	1	0
10751	46.7	11	21	0	1
49701	54.4	18	27	1	0
58577	38.1	23	159	0	1
61629	53.3	20	378	1	0

[79981 rows x 5 columns]

Baseline Model (Logistic Regression)

[ ]:
```
X = pd.DataFrame(Xtrain[['Age','Time','Amount','Entry_CVC','Entry_PIN']])
```

[ ]:
```
X = sm.add_constant(X)
```

[ ]:
```
y = pd.DataFrame(ytrain[['Fraud']])
```

[ ]:
```
logreg01 = sm.Logit(y, X).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.208073
         Iterations 8
```

[ ]:
```
logreg01.summary()
```

[ ]:

Dep. Variable:	Fraud	**No. Observations:**	79981
Model:	Logit	**Df Residuals:**	79975
Method:	MLE	**Df Model:**	5
Date:	Sun, 13 Apr 2025	**Pseudo R-squ.:**	0.1953
Time:	22:21:58	**Log-Likelihood:**	-16642.
converged:	True	**LL-Null:**	-20681.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	0.7263	0.080	9.045	0.000	0.569	0.884
Age	-0.0018	0.001	-1.215	0.224	-0.005	0.001
Time	-0.2231	0.003	-69.536	0.000	-0.229	-0.217
Amount	-0.0036	0.000	-22.240	0.000	-0.004	-0.003
Entry_CVC	0.1062	0.039	2.749	0.006	0.030	0.182
Entry_PIN	-0.6043	0.042	-14.523	0.000	-0.686	-0.523

[ ]:
```
#p(Fraud) = (exp(0.7623-0.0018(Age)-0.2231(Time)-0.0036(Amount)+0.
↪1062(Entry_CVC)-0.6043(Entry_PIN))/(1+exp(0.7623-0.0018(Age)-0.2231(Time)-0.
↪0036(Amount)+0.1062(Entry_CVC)-0.6043(Entry_PIN))
```

[ ]:
```
#Validating the model usinfg the test data set - AZ
```

[ ]:
```
y_test = project_data_test[['Fraud']]
X_test = pd.
↪DataFrame(project_data_test[['Age','Time','Amount','Entry_CVC','Entry_PIN']])
```

[ ]:
```
X_test = sm.add_constant(X_test)
```

[ ]:
```
logreg01_test = sm.Logit(y_test, X_test).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.205573
         Iterations 8
```

[ ]:
```python
logreg01_test.summary()
```

[ ]:

Dep. Variable:	Fraud	**No. Observations:**	19996
Model:	Logit	**Df Residuals:**	19990
Method:	MLE	**Df Model:**	5
Date:	Sun, 13 Apr 2025	**Pseudo R-squ.:**	0.2057
Time:	22:21:58	**Log-Likelihood:**	-4110.6
converged:	True	**LL-Null:**	-5175.3
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.8904	0.160	5.554	0.000	0.576	1.205
Age	-0.0011	0.003	-0.386	0.699	-0.007	0.005
Time	-0.2409	0.007	-36.364	0.000	-0.254	-0.228
Amount	-0.0031	0.000	-9.903	0.000	-0.004	-0.002
Entry_CVC	-0.0096	0.079	-0.122	0.903	-0.164	0.144
Entry_PIN	-0.6380	0.083	-7.645	0.000	-0.802	-0.474

[ ]:
```python
#Age and Entry_CVC have p-values higher than 0.05, and can be omitted from the␣
↪model.
```

[ ]:
```python
logreg01.predict(X_test)
test_predictions_log = logreg01.predict(X_test)
predicted_log = (test_predictions_log > 0.5).astype(int)  # For binary␣
↪classification
```

[ ]:
```python
cm_log = confusion_matrix(y_test, predicted_log)
cm_log
```

[ ]:
```
array([[18556,     0],
       [ 1221,   219]])
```

[ ]:
```python
TN_log = cm_log[0][0]
FP_log = cm_log[0][1]
FN_log = cm_log[1][0]
TP_log = cm_log[1][1]

table = [['', "Predicted: 0", "Predicted: 1"], ["Actual: 0", TN_log, FP_log],␣
↪["Actual: 1", FN_log, TP_log]]
print(tabulate(table, headers='firstrow'))
```

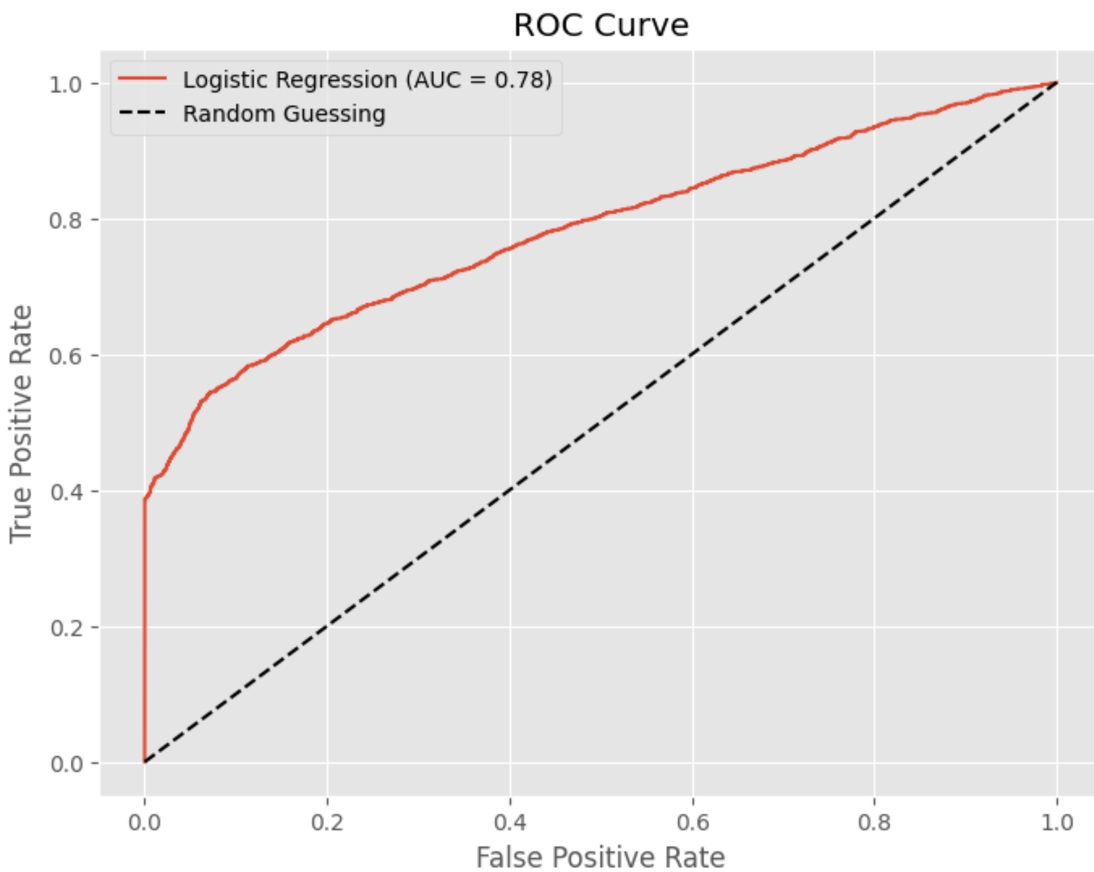
```
             Predicted: 0    Predicted: 1
---------  --------------  --------------
Actual: 0           18556               0
Actual: 1            1221             219
```

[ ]:
```
GT_log = TN_log + FP_log + FN_log + TP_log
Accuracy_log = (TN_log + TP_log)/GT_log
ErrorRate_log = 1-Accuracy_log
Sensitivity_log = TP_log/(FN_log + TP_log)
Recall_log = Sensitivity_log
Specificity_log = TN_log/(TN_log + FP_log)
Precision_log = TP_log/(FP_log + TP_log)
F1_log = (2*Precision_log*Recall_log)/(Precision_log + Recall_log)
F2_log = (5*Precision_log*Recall_log)/((4*Precision_log) + Recall_log)
F0_5_log = (1.25*Precision_log*Recall_log)/((.25*Precision_log)+Recall_log)
roc_auc = roc_auc_score(y_test, predicted_log)
```

[ ]:
```
roc_auc
```

[ ]:
```
np.float64(0.5760416666666667)
```

[ ]:
```
y_prob = logreg01.predict(X_test)
from sklearn.metrics import roc_curve, roc_auc_score

fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```



C5.0 Model

```
[ ]: c5_01 = DecisionTreeClassifier(criterion="entropy", max_leaf_nodes=5).
      ↪fit(Xtrain,ytrain)
```

```
[ ]: train_predictions = c5_01.predict(Xtrain)
     train_predictions
```

```
[ ]: array([0, 0, 0, …, 0, 0, 0])
```

```
[ ]: ytest= project_data_test[['Fraud']]
```

```
[ ]: Xtest = project_data_test[['Age', 'Time', 'Amount', 'Entry_CVC', 'Entry_PIN']]
```

```
[ ]: test_predictions = c5_01.predict(Xtest)
```

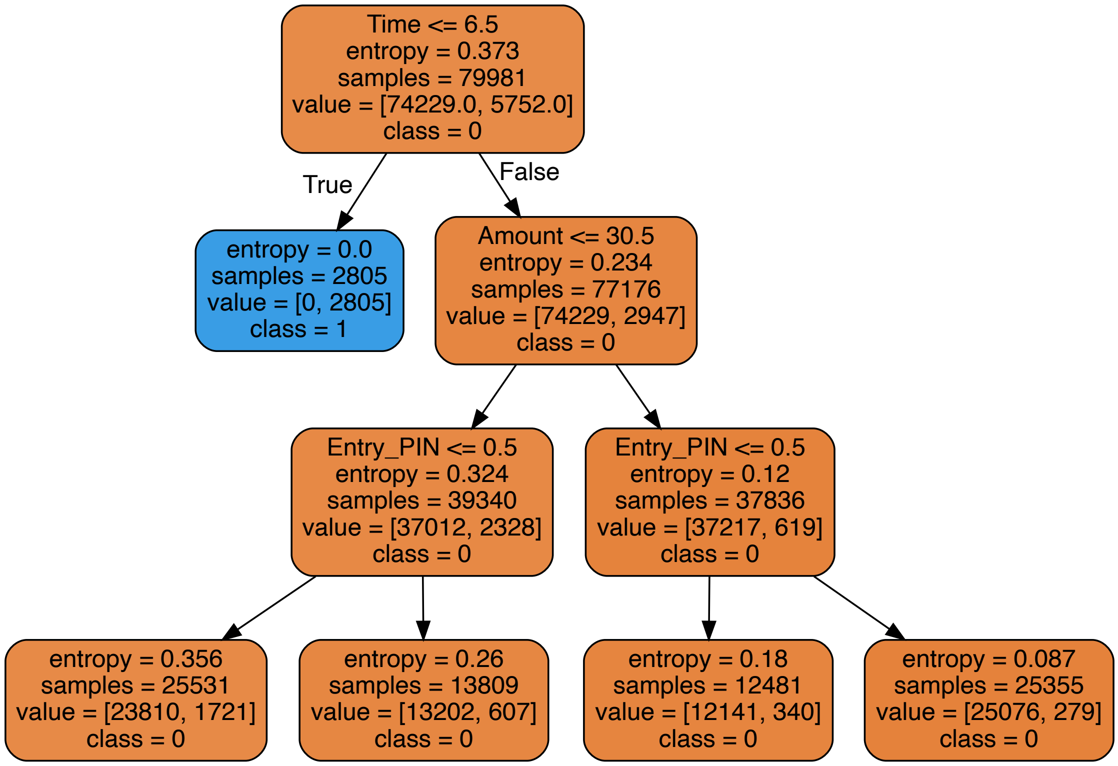
```
[ ]: test_predictions
```

```
[ ]: array([0, 0, 0, …, 0, 0, 0])
```

[ ]:
```python
export_graphviz(c5_01, out_file="project_data_test_c50_01.dot",␣
    ↪feature_names=X_names, class_names=y_names, filled=True, rounded=True)
```

[ ]:
```python
###C
with open("project_data_test_c50_01.dot") as f:dot_graph = f.read()
graphviz.Source(dot_graph)
```

[ ]:



[ ]:
```python
cm = confusion_matrix(ytest, test_predictions)
cm
```

[ ]:
```
array([[18556,     0],
       [  709,   731]])
```

[ ]:
```python
c5_01 = DecisionTreeClassifier(criterion="entropy", max_leaf_nodes=5).
    ↪fit(Xtest,ytest)
```

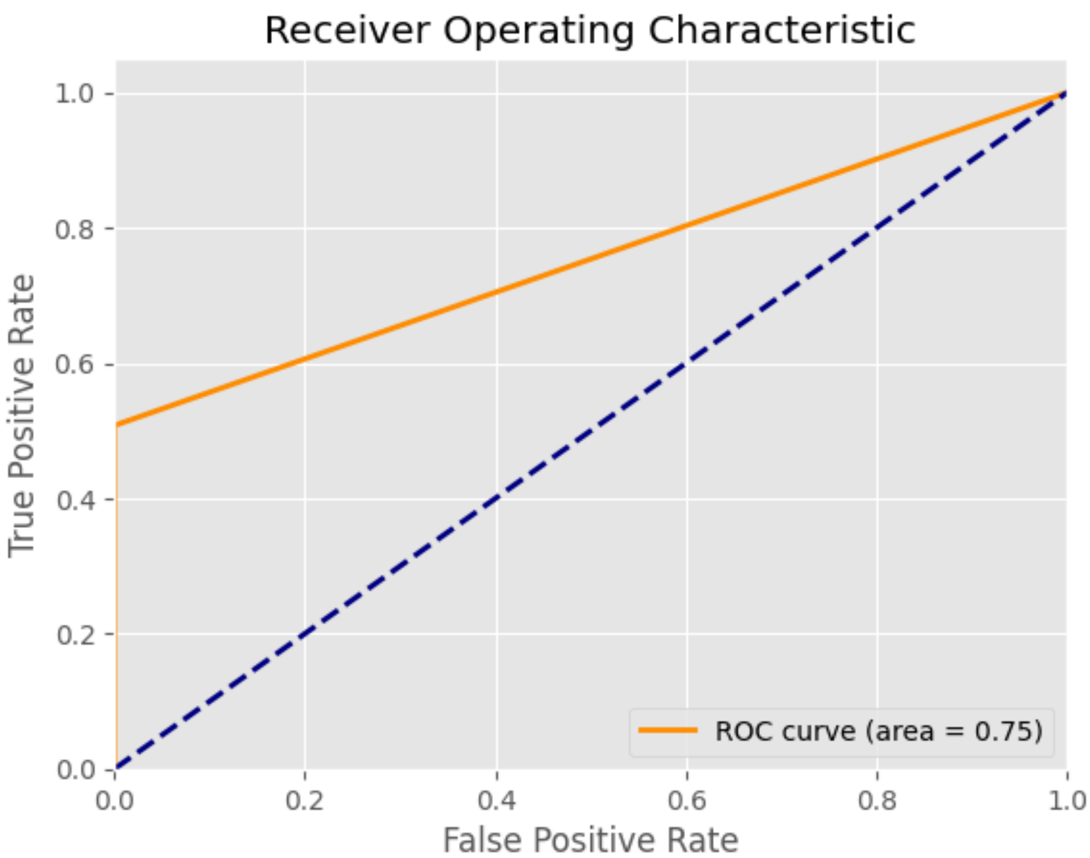
[ ]:
```python
TN = cm[0][0]
FP = cm[0][1]
FN = cm[1][0]
TP = cm[1][1]
```

```
table = [['', "Predicted: 0", "Predicted: 1"], ["Actual: 0", TN, FP], ["Actual:␣
↪1", FN, TP]]
print(tabulate(table, headers='firstrow'))
```

```
             Predicted: 0    Predicted: 1
---------  --------------  --------------
Actual: 0           18556               0
Actual: 1             709             731
```

[ ]:
```
GT = TN + FP + FN + TP
Accuracy = (TN + TP)/GT
ErrorRate = 1-Accuracy
Sensitivity = TP/(FN + TP)
Recall = Sensitivity
Specificity = TN/(TN + FP)
Precision = TP/(FP + TP)
F1 = (2*Precision*Recall)/(Precision + Recall)
F2 = (5*Precision*Recall)/((4*Precision) + Recall)
F0_5 = (1.25*Precision*Recall)/((.25*Precision)+Recall)
roc_auc_c50 = roc_auc_score(y_test, test_predictions)
```

[ ]:
```
fpr2, tpr2, thresholds2 = roc_curve(y_test, test_predictions)
roc_auc_c50 = roc_auc_score(y_test, test_predictions)

plt.figure()
plt.plot(fpr2, tpr2, color='darkorange', lw=2, label='ROC curve (area = %0.2f)'␣
↪% roc_auc_c50)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```




```
[ ]: roc_auc_c50
```

```
[ ]: np.float64(0.7538194444444444)
```

```
[ ]: Accuracy2 = 0
     ErrorRate2 = 1-Accuracy2
     Sensitivity2 = 0
     Recall2 = Sensitivity2
     Specificity2 = 0
     Precision2 = 0
     F1_2 = 0
     F2_2 = 0
     F0_5_2 = 0
```

```
[ ]: #C5.0 model - Initiated by AZ 7:38pm
     print(tabulate({' ': ['Accuracy', "Error Rate", "Sensitivity", "Specificity",␣
     ↪"Precision", "F1", "F2", "F0.5"],
     'Logistic Regression (Baseline)': [Accuracy_log, ErrorRate_log,␣
     ↪Sensitivity_log, Specificity_log,Precision_log, F1_log, F2_log, F0_5_log],
```

```
'C5.0': [Accuracy, ErrorRate, Sensitivity, Specificity, Precision, F1, F2,␣
↪F0_5]},headers="keys"))
```

```
               Logistic Regression (Baseline)       C5.0
-----------  -------------------------------  ---------
Accuracy                           0.938938   0.964543
Error Rate                         0.0610622  0.0354571
Sensitivity                        0.152083   0.507639
Specificity                        1          1
Precision                          1          1
F1                                 0.264014   0.673422
F2                                 0.183141   0.563087
F0.5                               0.472798   0.837534
```

Random Forest

```
[ ]: #Random Forest
```

```
[ ]: from sklearn.ensemble import RandomForestClassifier
     import numpy as np
```

```
[ ]: rfy = np.ravel(ytrain)
     rfy
```

```
[ ]: array([0, 0, 0, …, 0, 0, 0])
```

```
[ ]: rf01 = RandomForestClassifier(n_estimators = 100,
     criterion="gini").fit(Xtrain,rfy)
```

```
[ ]: rf01.predict(Xtrain)
```

```
[ ]: array([0, 0, 0, …, 0, 0, 0])
```

```
[ ]: train_predictions_randomforest = rf01.predict(Xtrain)
```

```
[ ]: randomforest_test = rf01.predict(Xtest)
     randomforest_test
```

```
[ ]: array([0, 0, 0, …, 0, 0, 0])
```

```
[ ]: cm3 = confusion_matrix(ytest, randomforest_test)
     cm3
```

```
[ ]: array([[18429,   127],
            [  700,   740]])
```

```
[ ]: TN3 = cm3[0][0]
     FP3 = cm3[0][1]
```

```
FN3 = cm3[1][0]
TP3 = cm3[1][1]
```

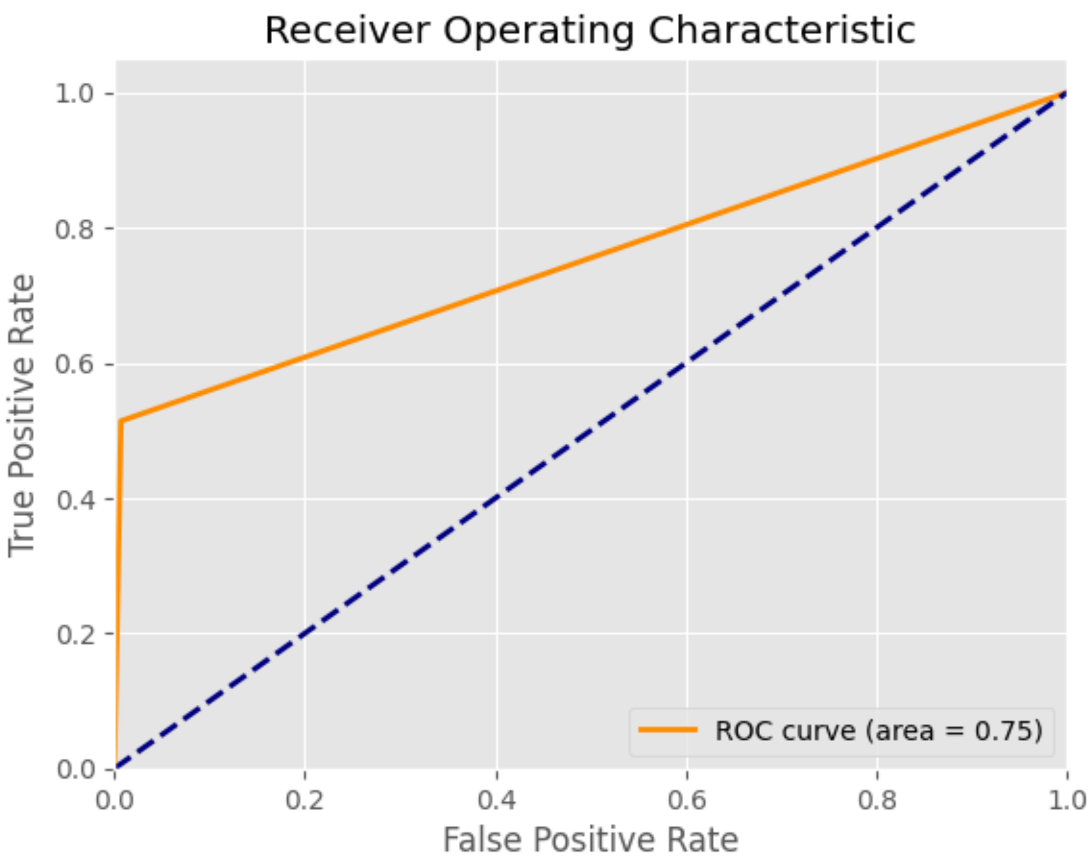
[ ]:
```
TN3 = cm3[0][0]
FP3 = cm3[0][1]
FN3 = cm3[1][0]
TP3 = cm3[1][1]

table3 = [['', "Predicted: 0", "Predicted: 1"], ["Actual: 0", TN3, FP3],␣
 ↪["Actual: 1", FN3, TP3]]
print(tabulate(table3, headers='firstrow'))
```

```
             Predicted: 0    Predicted: 1
---------  --------------  --------------
Actual: 0           18429             127
Actual: 1             700             740
```

[ ]:
```
GT3 = TN3 + FP3 + FN3 + TP3
Accuracy3 = (TN3 + TP3)/GT3
ErrorRate3 = 1-Accuracy3
Sensitivity3 = TP3/(FN3 + TP3)
Recall3 = Sensitivity3
Specificity3 = TN3/(TN3 + FP3)
Precision3 = TP3/(FP3 + TP3)
F1_3 = (2*Precision3*Recall3)/(Precision3 + Recall3)
F2_3 = (5*Precision3*Recall3)/((4*Precision3) + Recall3)
F0_5_3 = (1.25*Precision3*Recall3)/((.25*Precision3)+Recall3)
roc_auc_randomforest = roc_auc_score(y_test, randomforest_test)
roc_auc_randomforest
```

[ ]: np.float64(0.7535223707216593)

[ ]:
```
fpr3, tpr3, thresholds3 = roc_curve(y_test, randomforest_test)
roc_auc_randomforest = roc_auc_score(y_test, randomforest_test)

plt.figure()
plt.plot(fpr3, tpr3, color='darkorange', lw=2, label='ROC curve (area = %0.2f)'␣
 ↪% roc_auc_randomforest)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



[ ]:
```
print(tabulate({' ': ['Accuracy', "Error Rate", "Sensitivity", "Specificity",␣
↪"Precision", "F1", "F2", "F0.5"],
'Logistic Regression (Baseline)': [Accuracy_log, ErrorRate_log,␣
↪Sensitivity_log, Specificity_log,Precision_log, F1_log, F2_log, F0_5_log],
'Random Forest': [Accuracy3, ErrorRate3, Sensitivity3, Specificity3,␣
↪Precision3, F1_3, F2_3, F0_5_3]},headers="keys"))
```

```
             Logistic Regression (Baseline)    Random Forest
-----------  --------------------------------  ---------------
Accuracy                           0.938938          0.958642
Error Rate                         0.0610622         0.0413583
Sensitivity                        0.152083          0.513889
Specificity                        1                 0.993156
Precision                          1                 0.853518
F1                                 0.264014          0.641526
F2                                 0.183141          0.558322
F0.5                               0.472798          0.753871
```

[ ]:
```
from sklearn.tree import plot_tree
```

Naive Bayes

[ ]:
```
nb_01 = MultinomialNB().fit(Xtrain, ytrain)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().
```

[ ]:
```
nb_01
```

[ ]:
```
MultinomialNB()
```

[ ]:
```
nb_01.predict(Xtrain)
```

[ ]:
```
array([0, 1, 0, …, 1, 0, 0])
```

[ ]:
```
train_predictions_nb_01 = nb_01.predict(Xtest)
train_predictions_nb_01
```

[ ]:
```
array([1, 0, 0, …, 0, 0, 1])
```

[ ]:
```
cm4 = confusion_matrix(ytest, train_predictions_nb_01)
cm4
```

[ ]:
```
array([[ 8279, 10277],
       [  303,  1137]])
```

[ ]:
```
TN4 = cm4[0][0]
FP4 = cm4[0][1]
FN4 = cm4[1][0]
TP4 = cm4[1][1]

table4 = [['', "Predicted: 0", "Predicted: 1"], ["Actual: 0", TN4, FP4],␣
 ↪["Actual: 1", FN4, TP4]]
print(tabulate(table4, headers='firstrow'))
```

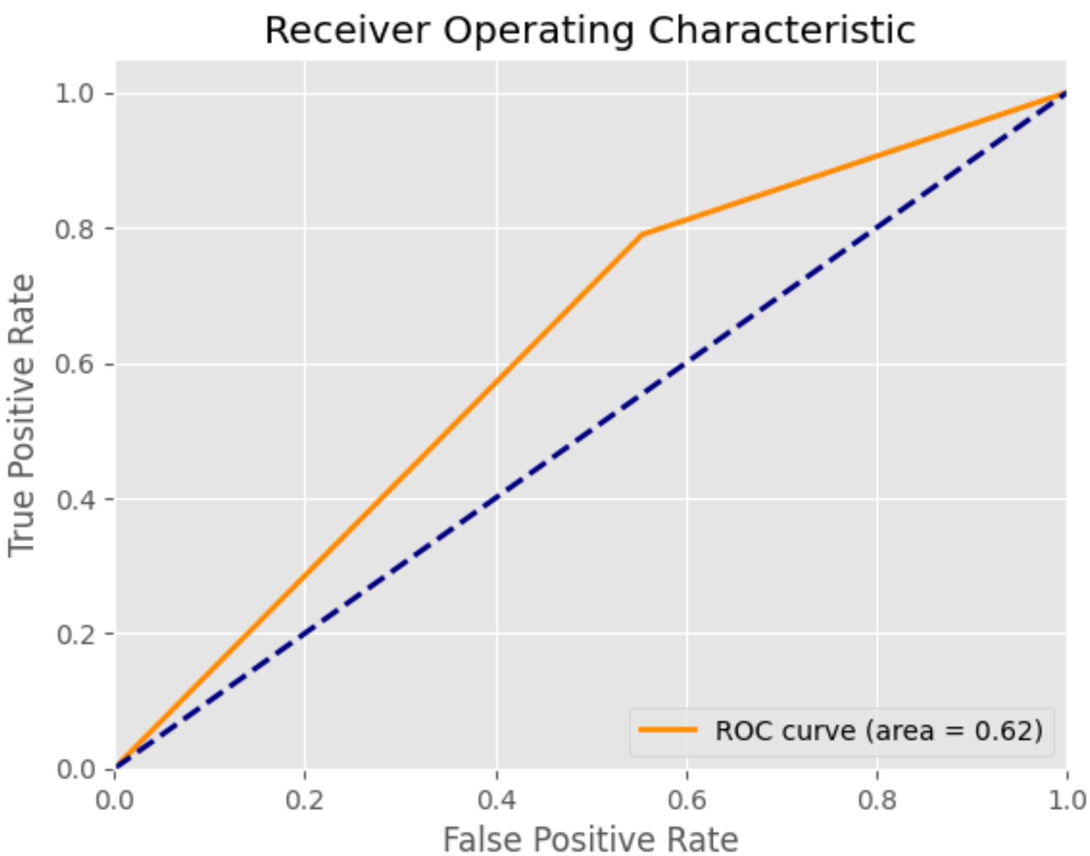
```
             Predicted: 0    Predicted: 1
---------  --------------  --------------
Actual: 0            8279           10277
Actual: 1             303            1137
```

[ ]:
```
GT4 = TN4 + FP4 + FN4 + TP4
Accuracy4 = (TN4 + TP4)/GT4
ErrorRate4 = 1-Accuracy4
Sensitivity4 = TP4/(FN4 + TP4)
```

```python
Recall4 = Sensitivity4
Specificity4 = TN4/(TN4 + FP4)
Precision4 = TP4/(FP4 + TP4)
F1_4 = (2*Precision4*Recall4)/(Precision4 + Recall4)
F2_4 = (5*Precision4*Recall4)/((4*Precision4) + Recall4)
F0_5_4 = (1.25*Precision4*Recall4)/((.25*Precision4)+Recall4)
roc_auc_naivebayes = roc_auc_score(y_test, train_predictions_nb_01)
roc_auc_naivebayes
```

[ ]: np.float64(0.6178731497449163)

[ ]:
```python
fpr4, tpr4, thresholds4 = roc_curve(y_test, train_predictions_nb_01)
roc_auc_train_predictions_nb = roc_auc_score(y_test, train_predictions_nb_01)

plt.figure()
plt.plot(fpr4, tpr4, color='darkorange', lw=2, label='ROC curve (area = %0.2f)'
 ↪% roc_auc_train_predictions_nb)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```




```
[ ]: print(tabulate({' ': ['Accuracy', "Error Rate", "Sensitivity", "Specificity",␣
     ↪"Precision", "F1", "F2", "F0.5"],
     'Logistic Regression (Baseline)': [Accuracy_log, ErrorRate_log,␣
     ↪Sensitivity_log, Specificity_log,Precision_log, F1_log, F2_log, F0_5_log],
     'Naive Bayes': [Accuracy4, ErrorRate4, Sensitivity4, Specificity4, Precision4,␣
     ↪F1_4, F2_4, F0_5_4]},headers="keys"))
```

```
               Logistic Regression (Baseline)    Naive Bayes
-----------  --------------------------------  -------------
Accuracy                            0.938938       0.470894
Error Rate                          0.0610622      0.529106
Sensitivity                         0.152083       0.789583
Specificity                         1              0.446163
Precision                           1              0.0996145
F1                                  0.264014       0.17691
F2                                  0.183141       0.331024
F0.5                                0.472798       0.120711
```

CART

[ ]:
```
cart_01 = DecisionTreeClassifier(criterion = "gini", max_leaf_nodes = 5).
    ↪fit(Xtrain, ytrain)
```

[ ]:
```
cart_preds = cart_01.predict(Xtrain)
cart_preds
```

[ ]: array([0, 0, 0, …, 0, 0, 0])

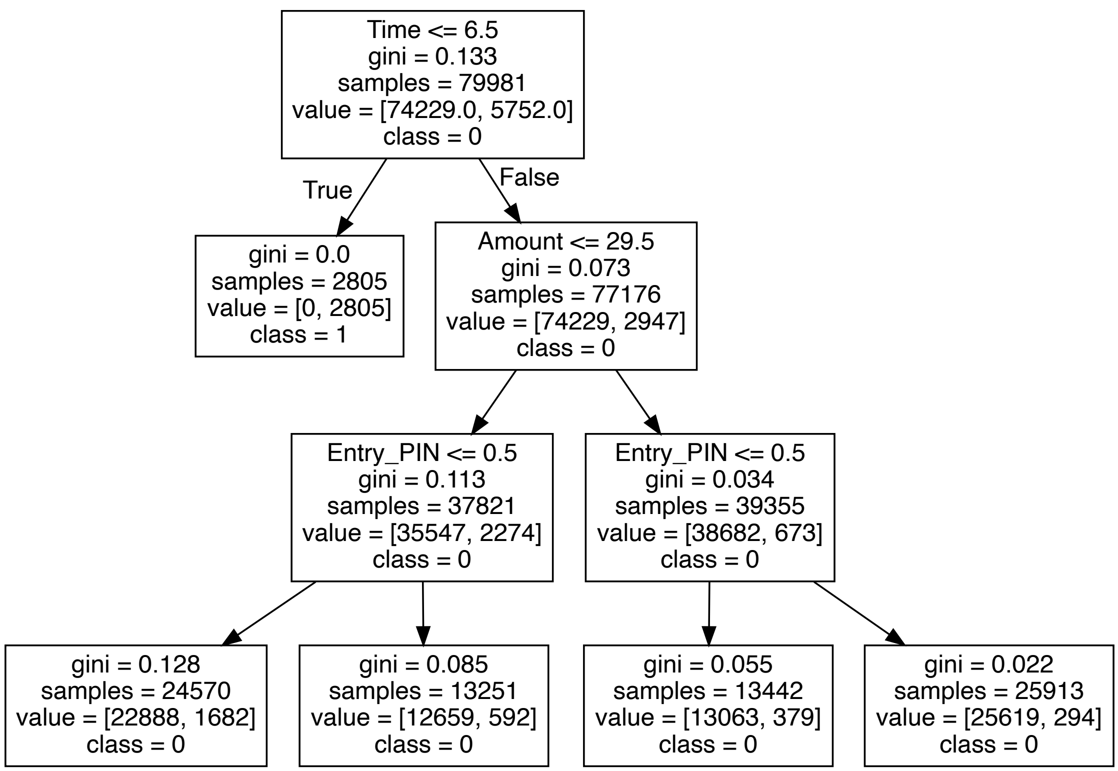
[ ]:
```
cart_test = cart_01.predict(Xtest)
cart_test
```

[ ]: array([0, 0, 0, …, 0, 0, 0])

[ ]:
```
export_graphviz(cart_01, out_file="cart_01_test.dot", feature_names=X_names,␣
    ↪class_names=y_names)
```

[ ]:
```
with open("cart_01_test.dot") as f:cart_graph = f.read()
graphviz.Source(cart_graph)
```

[ ]:



[ ]:
```
cart_matrix = confusion_matrix(ytest, cart_test)
cart_matrix
```

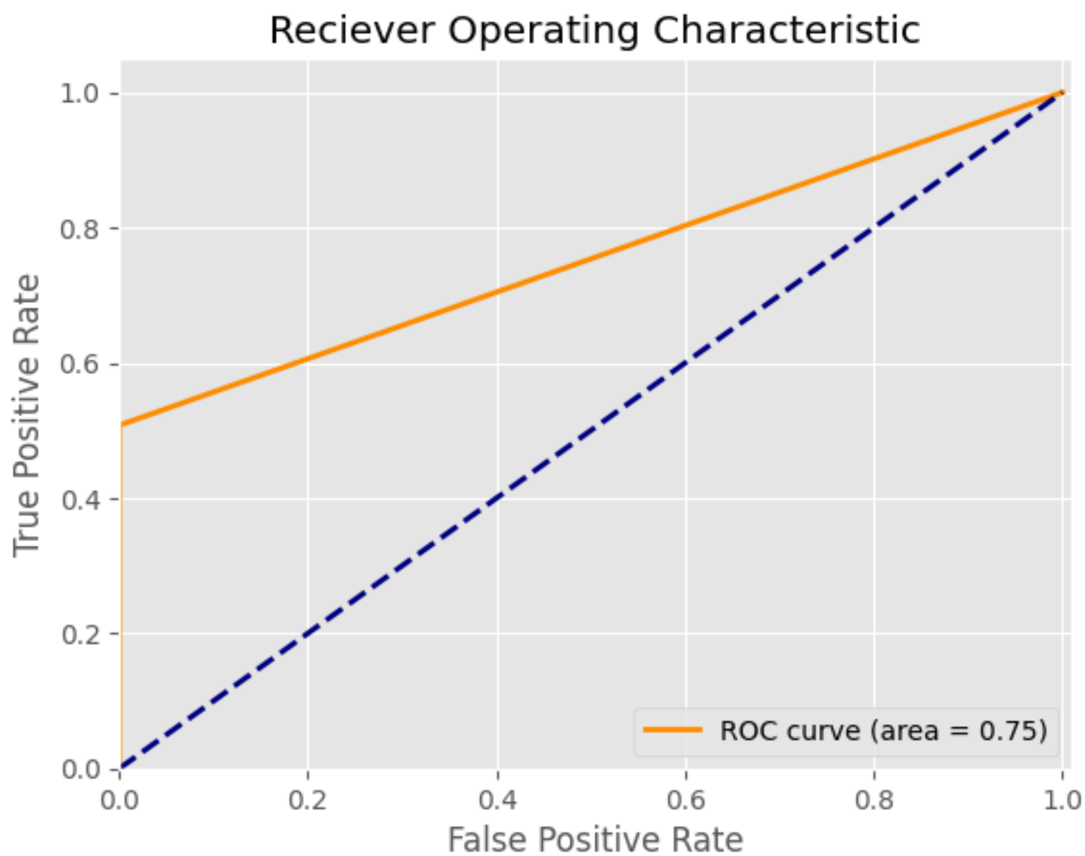
```
[ ]: array([[18556,     0],
            [  709,   731]])
```

```
[ ]: TN5 = cart_matrix[0][0]
     FP5 = cart_matrix[0][1]
     FN5 = cart_matrix[1][0]
     TP5 = cart_matrix[1][1]
     table = [['', "Predicted: 0", "Predicted: 1"], ["Actual: 0", TN5, FP5],␣
      ↪["Actual: 1", FN5, TP5]]
     print (tabulate(table, headers='firstrow'))
```

```
             Predicted: 0    Predicted: 1
---------  --------------  --------------
Actual: 0           18556               0
Actual: 1             709             731
```

```
[ ]: GT5 = TN5 + FP5 + FN5 + TP5
     Accuracy5 = (TN5 + TP5)/GT5
     ErrorRate5 = 1 - Accuracy5
     Sensitivity5 = TP5/(FN5 + TP5)
     Recall5 = Sensitivity5
     Specificity5 = TN5/(TN5 + FP5)
     Precision5 = TP5/(FP5 + TP5)
     F1_5 = (2 * Precision5 * Recall5)/(Precision5 + Recall5)
     F2_5 = (5 * Precision5 * Recall5)/((4 * Precision5) + Recall5)
     F0_5_5 = (1.25 * Precision5 * Recall5)/((.25 * Precision5) + Recall5)
     roc_auc_cart = roc_auc_score(y_test, cart_test)
```

```
[ ]: fpr5, tpr5, thresholds5 = roc_curve(y_test, cart_test)

     plt.figure()
     plt.plot(fpr5, tpr5, color="darkorange", lw=2, label='ROC curve (area = %0.2f)'␣
      ↪% roc_auc_cart)
     plt.plot([0,1], [0,1], color="navy", lw=2, linestyle='--')
     plt.xlim([0.0, 1.01])
     plt.ylim([0.0, 1.05])
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.title('Reciever Operating Characteristic')
     plt.legend(loc="lower right")
     plt.show()
```



[ ]:
```
print(tabulate({' ': ['Accuracy', "Error Rate", "Sensitivity", "Specificity",␣
↪"Precision", "F1", "F2", "F0.5"],
'Logistic Regression (Baseline)': [Accuracy_log, ErrorRate_log,␣
↪Sensitivity_log, Specificity_log, Precision_log, F1_log, F2_log, F0_5_log],
'CART': [Accuracy5, ErrorRate5, Sensitivity5, Specificity5, Precision5, F1_5,␣
↪F2_5, F0_5_5]}, headers="keys"))
```

	Logistic Regression (Baseline)	CART
Accuracy	0.938938	0.964543
Error Rate	0.0610622	0.0354571
Sensitivity	0.152083	0.507639
Specificity	1	1
Precision	1	1
F1	0.264014	0.673422
F2	0.183141	0.563087
F0.5	0.472798	0.837534

Model Evaluations

[ ]:

```

```

[ ]:

```
model_evaluation_table = {'Evaluation Measure': ['Accuracy', 'Error Rate',␣
 ↪'Recall', 'Precision','Specificity', 'F1', 'ROC AUC'],
'Logistic Regression (Baseline)': [0] * 7, # Initialize with placeholder val
'C5.0': [0] * 7,
'Random Forest': [0] * 7,
'Naive Bayes': [0] *7,
'CART': [0] * 7}
model_evaluation_df = pd.DataFrame(model_evaluation_table)
model_evaluation_df
```

[ ]:

```
  Evaluation Measure  Logistic Regression (Baseline)  C5.0  Random Forest  \
0           Accuracy                               0     0              0
1         Error Rate                               0     0              0
2             Recall                               0     0              0
3          Precision                               0     0              0
4        Specificity                               0     0              0
5                 F1                               0     0              0
6            ROC AUC                               0     0              0

   Naive Bayes  CART
0            0     0
1            0     0
2            0     0
3            0     0
4            0     0
5            0     0
6            0     0
```

[ ]:

```
def format_metric(metric):
  return f'{metric:.4f}'
```

[ ]:

```
#Adding logistic regression
GT_log = TN_log + FP_log + FN_log + TP_log
metrics_log = {
    'Accuracy': (TN_log + TP_log)/GT_log,
    'Error Rate': 1 - ((TN_log + TP_log)/GT_log),
    'Recall': TP_log/(FN_log + TP_log),
    'Precision': TP_log/(FP_log + TP_log),
    'Specificity': TN_log/(TN_log + FP_log),
    'F1': (2 * (TP_log/(FP_log + TP_log)) * (TP_log/(FN_log + TP_log))) /
          ((TP_log/(FP_log + TP_log)) + (TP_log/(FN_log + TP_log))),
    'ROC AUC': roc_auc_score(y_test, test_predictions_log)
}

model_evaluation_df['Logistic Regression (Baseline)'] = [
```

```
    metrics_log['Accuracy'],
    metrics_log['Error Rate'],
    metrics_log['Recall'],
    metrics_log['Precision'],
    metrics_log['Specificity'],
    metrics_log['F1'],
    metrics_log['ROC AUC']
]

model_evaluation_df['Logistic Regression (Baseline)'] =␣
 ↪model_evaluation_df['Logistic Regression (Baseline)'].apply(lambda x: f"{x:.
 ↪4f}" if isinstance(x, (int, float)) else x)

model_evaluation_df
```

[ ]:

```
  Evaluation Measure Logistic Regression (Baseline)  C5.0  Random Forest  \
0           Accuracy                         0.9389     0              0
1         Error Rate                         0.0611     0              0
2             Recall                         0.1521     0              0
3          Precision                         1.0000     0              0
4        Specificity                         1.0000     0              0
5                 F1                         0.2640     0              0
6            ROC AUC                         0.7817     0              0

   Naive Bayes  CART
0            0     0
1            0     0
2            0     0
3            0     0
4            0     0
5            0     0
6            0     0
```

[ ]:

```
#Adding C5.0
GT = TN + FP + FN + TP
metrics_log = {
    'Accuracy': (TN + TP)/GT,
    'Error Rate': 1 - ((TN + TP)/GT),
    'Recall': TP/(FN + TP),
    'Precision': TP/(FP + TP),
    'Specificity': TN/(TN + FP),
    'F1': (2 * (TP/(FP + TP)) * (TP/(FN + TP))) /
          ((TP/(FP + TP)) + (TP/(FN + TP))),
    'ROC AUC': roc_auc_c50
}
```

```python
model_evaluation_df['C5.0'] = [
    metrics_log['Accuracy'],
    metrics_log['Error Rate'],
    metrics_log['Recall'],
    metrics_log['Precision'],
    metrics_log['Specificity'],
    metrics_log['F1'],
    metrics_log['ROC AUC']
]

model_evaluation_df['C5.0'] = model_evaluation_df['C5.0'].apply(lambda x: f"{x:.
↪4f}" if isinstance(x, (int, float)) else x)

model_evaluation_df
```

[ ]:

```
  Evaluation Measure Logistic Regression (Baseline)    C5.0  Random Forest  \
0           Accuracy                         0.9389  0.9645              0
1         Error Rate                         0.0611  0.0355              0
2             Recall                         0.1521  0.5076              0
3          Precision                         1.0000  1.0000              0
4        Specificity                         1.0000  1.0000              0
5                 F1                         0.2640  0.6734              0
6            ROC AUC                         0.7817  0.7538              0

   Naive Bayes  CART
0            0     0
1            0     0
2            0     0
3            0     0
4            0     0
5            0     0
6            0     0
```

[ ]:

```python
#Adding Random Forest
GT3 = TN3 + FP3 + FN3 + TP3
metrics_log = {
    'Accuracy': (TN3 + TP3)/GT3,
    'Error Rate': 1 - ((TN3 + TP3)/GT3),
    'Recall': TP3/(FN3 + TP3),
    'Precision': TP3/(FP3 + TP3),
    'Specificity': TN3/(TN3 + FP3),
    'F1': (2 * (TP3/(FP3 + TP3)) * (TP3/(FN3 + TP3))) /
          ((TP3/(FP3 + TP3)) + (TP3/(FN3 + TP3))),
    'ROC AUC': roc_auc_randomforest
}
```

```
model_evaluation_df['Random Forest'] = [
    metrics_log['Accuracy'],
    metrics_log['Error Rate'],
    metrics_log['Recall'],
    metrics_log['Precision'],
    metrics_log['Specificity'],
    metrics_log['F1'],
    metrics_log['ROC AUC']
]

model_evaluation_df['Random Forest'] = model_evaluation_df['Random Forest'].
 ↪apply(lambda x: f"{x:.4f}" if isinstance(x, (int, float)) else x)

model_evaluation_df
```

[ ]:
```
  Evaluation Measure Logistic Regression (Baseline)    C5.0 Random Forest  \
0           Accuracy                          0.9389  0.9645        0.9586
1         Error Rate                          0.0611  0.0355        0.0414
2             Recall                          0.1521  0.5076        0.5139
3          Precision                          1.0000  1.0000        0.8535
4        Specificity                          1.0000  1.0000        0.9932
5                 F1                          0.2640  0.6734        0.6415
6            ROC AUC                          0.7817  0.7538        0.7535

  Naive Bayes  CART
0           0     0
1           0     0
2           0     0
3           0     0
4           0     0
5           0     0
6           0     0
```

[ ]:
```
#Adding Naive Bayes
GT4 = TN4 + FP4 + FN4 + TP4
metrics_log = {
    'Accuracy': (TN4 + TP4)/GT4,
    'Error Rate': 1 - ((TN4 + TP4)/GT4),
    'Recall': TP4/(FN4 + TP4),
    'Precision': TP4/(FP4 + TP4),
    'Specificity': TN4/(TN4 + FP4),
    'F1': (2 * (TP4/(FP4 + TP4)) * (TP4/(FN4 + TP4))) /
          ((TP4/(FP4 + TP4)) + (TP4/(FN4 + TP4))),
    'ROC AUC': roc_auc_naivebayes
}
```

```python
model_evaluation_df['Naive Bayes'] = [
    metrics_log['Accuracy'],
    metrics_log['Error Rate'],
    metrics_log['Recall'],
    metrics_log['Precision'],
    metrics_log['Specificity'],
    metrics_log['F1'],
    metrics_log['ROC AUC']
]

model_evaluation_df['Naive Bayes'] = model_evaluation_df['Naive Bayes'].
 ↪apply(lambda x: f"{x:.4f}" if isinstance(x, (int, float)) else x)

model_evaluation_df
```

[ ]:

```
  Evaluation Measure Logistic Regression (Baseline)    C5.0 Random Forest  \
0           Accuracy                         0.9389  0.9645        0.9586
1         Error Rate                         0.0611  0.0355        0.0414
2             Recall                         0.1521  0.5076        0.5139
3          Precision                         1.0000  1.0000        0.8535
4        Specificity                         1.0000  1.0000        0.9932
5                 F1                         0.2640  0.6734        0.6415
6            ROC AUC                         0.7817  0.7538        0.7535

  Naive Bayes  CART
0      0.4709     0
1      0.5291     0
2      0.7896     0
3      0.0996     0
4      0.4462     0
5      0.1769     0
6      0.6179     0
```

[ ]:

```python
#Adding CART
GT5 = TN5 + FP5 + FN5 + TP5
metrics_log = {
    'Accuracy': (TN5 + TP5)/GT5,
    'Error Rate': 1 - ((TN5 + TP5)/GT5),
    'Recall': TP5/(FN5 + TP5),
    'Precision': TP5/(FP5 + TP5),
    'Specificity': TN5/(TN5 + FP5),
    'F1': (2 * (TP5/(FP5 + TP5)) * (TP5/(FN5 + TP5))) /
          ((TP5/(FP5 + TP5)) + (TP5/(FN5 + TP5))),
    'ROC AUC': roc_auc_cart
}
```

```python
model_evaluation_df['CART'] = [
    metrics_log['Accuracy'],
    metrics_log['Error Rate'],
    metrics_log['Recall'],
    metrics_log['Precision'],
    metrics_log['Specificity'],
    metrics_log['F1'],
    metrics_log['ROC AUC']
]

model_evaluation_df['CART'] = model_evaluation_df['CART'].apply(lambda x: f"{x:.
↪4f}" if isinstance(x, (int, float)) else x)

model_evaluation_df
```

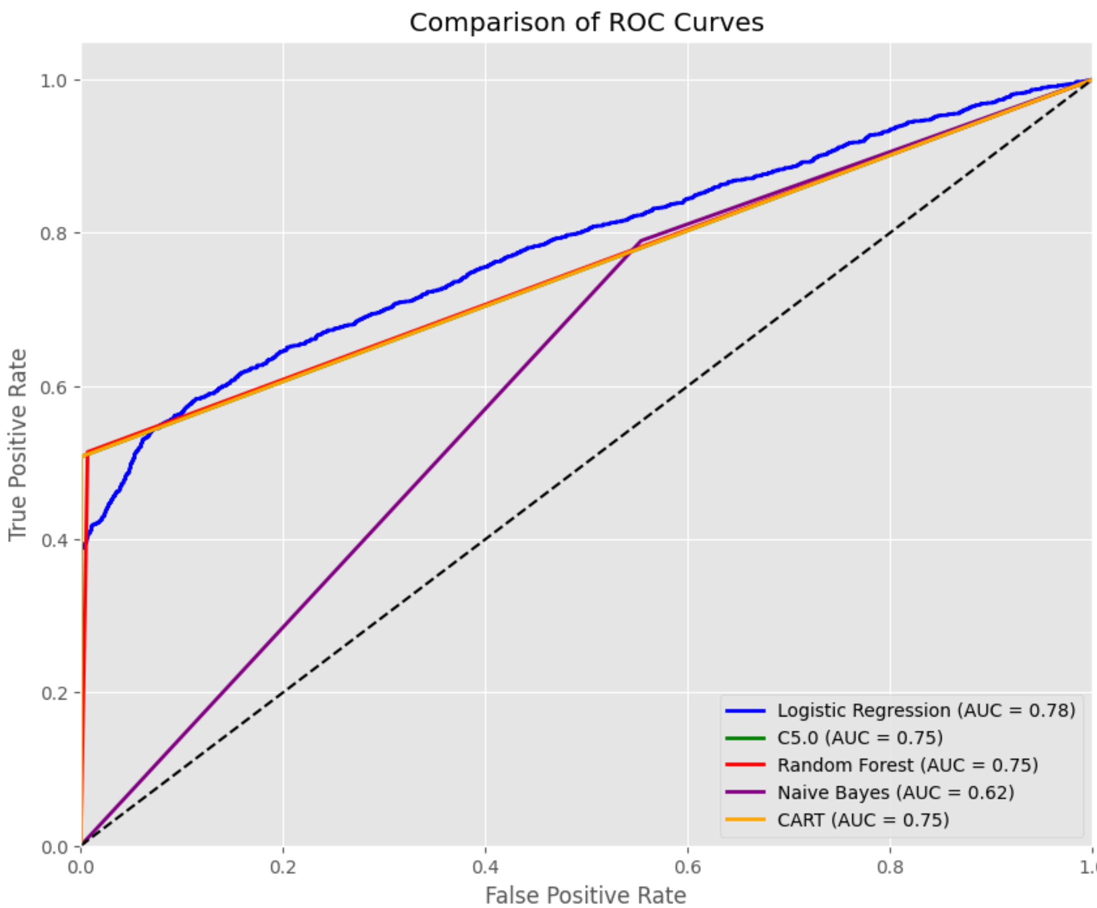
[ ]:

```
  Evaluation Measure Logistic Regression (Baseline)    C5.0 Random Forest  \
0           Accuracy                          0.9389  0.9645        0.9586
1         Error Rate                          0.0611  0.0355        0.0414
2             Recall                          0.1521  0.5076        0.5139
3          Precision                          1.0000  1.0000        0.8535
4        Specificity                          1.0000  1.0000        0.9932
5                 F1                          0.2640  0.6734        0.6415
6            ROC AUC                          0.7817  0.7538        0.7535

  Naive Bayes    CART
0      0.4709  0.9645
1      0.5291  0.0355
2      0.7896  0.5076
3      0.0996  1.0000
4      0.4462  1.0000
5      0.1769  0.6734
6      0.6179  0.7538
```

[ ]:

```python
plt.figure(figsize=(10, 8))

# Plot all ROC curves
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {roc_auc:.2f})',␣
↪color='blue', lw=2)
plt.plot(fpr2, tpr2, label=f'C5.0 (AUC = {roc_auc_c50:.2f})', color='green',␣
↪lw=2)
plt.plot(fpr3, tpr3, label=f'Random Forest (AUC = {roc_auc_randomforest:.2f})',␣
↪color='red', lw=2)
plt.plot(fpr4, tpr4, label=f'Naive Bayes (AUC = {roc_auc_naivebayes:.2f})',␣
↪color='purple', lw=2)
plt.plot(fpr5, tpr5, label=f'CART (AUC = {roc_auc_cart:.2f})', color='orange',␣
↪lw=2)
```

```python
# Plot random guessing line
plt.plot([0, 1], [0, 1], 'k--',)

# Customize the plot
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Comparison of ROC Curves')
plt.legend(loc="lower right")
plt.grid(True)

plt.show()
```



[ ]:
```python
#Naive Bayes classification
from sklearn.naive_bayes import MultinomialNB
import statsmodels.tools.tools as stattools
```

```
#Version of dataframe without missing values
project_data_train_cl = project_data_train.dropna()

#Array of predictor variables
X_vars = pd.concat((project_data_train_cl['Age'],␣
↪project_data_train_cl['Time'], project_data_train_cl['Amount']), axis = 1)

#Target variable
Y_var = project_data_train_cl['Fraud']

#NB model using tranformed training set
nb_1 = MultinomialNB().fit(X_vars, Y_var)
```

[ ]:
```
#Predicting values in the test set
X_vtest = pd.concat((project_data_test['Age'], project_data_test['Time'],␣
↪project_data_test['Amount']), axis = 1)

Y_vtest = project_data_test['Fraud']

Y_predicted1 = nb_1.predict(X_vtest)
```

[ ]:
```
#Comparing predicted values to actual values
ypred1 = pd.crosstab(project_data_test['Fraud'], Y_predicted1, rownames =␣
↪['Actual'], colnames = ['Predicted'])
ypred1['Total'] = ypred1.sum(axis = 1); ypred1.loc['Total'] = ypred1.sum()

ypred1
```

[ ]:

Predicted	0	1	Total
Actual			
0	8263	10293	18556
1	303	1137	1440
Total	8566	11430	19996

[ ]:
```
cm_nb = confusion_matrix(project_data_test['Fraud'], Y_predicted1)
cm_nb
```

[ ]:
```
array([[ 8263, 10293],
       [  303,  1137]])
```

[ ]:
```
TN_nb = cm_nb[0][0]
FP_nb = cm_nb[0][1]
FN_nb = cm_nb[1][0]
TP_nb = cm_nb[1][1]
```

```
table_nb = [['', "Predicted: 0", "Predicted: 1"], ["Actual: 0", TN_nb, FP_nb],␣
↪["Actual: 1", FN_nb, TP_nb]]
print(tabulate(table_nb, headers='firstrow'))
```

```
             Predicted: 0    Predicted: 1
---------  --------------  --------------
Actual: 0            8263           10293
Actual: 1             303            1137
```

[ ]:
```
GT_nb = TN_nb + FP_nb + FN_nb + TP_nb
Accuracy_nb = (TN_nb + TP_nb)/GT_nb
ErrorRate_nb = 1-Accuracy_nb
Sensitivity_nb = TP_nb/(FN_nb + TP_nb)
Recall_nb = Sensitivity_nb
Specificity_nb = TN_nb/(TN_nb + FP_nb)
Precision_nb = TP_nb/(FP_nb + TP_nb)
F1_nb = (2*Precision_nb*Recall_nb)/(Precision_nb + Recall_nb)
F2_nb = (5*Precision_nb*Recall_nb)/((4*Precision_nb) + Recall_nb)
F0_5_nb = (1.25*Precision_nb*Recall_nb)/((.25*Precision_nb)+Recall_nb)
roc_auc_naivebayes1 = roc_auc_score(project_data_test['Fraud'], Y_predicted1)
roc_auc_naivebayes1
```

[ ]:
```
np.float64(0.6174420223467701)
```

[ ]:
```
print(tabulate({' ': ['Accuracy', "Error Rate", "Sensitivity", "Specificity",␣
↪"Precision", "F1", "F2", "F0.5"],
'Naive Bayes': [Accuracy_nb, ErrorRate_nb, Sensitivity_nb, Specificity_nb,␣
↪Precision_nb, F1_nb, F2_nb, F0_5_nb]},headers="keys"))
```

```
               Naive Bayes
-----------  -------------
Accuracy         0.470094
Error Rate       0.529906
Sensitivity      0.789583
Specificity      0.445301
Precision        0.0994751
F1               0.17669
F2               0.330716
F0.5             0.120547
```

[ ]:
```
db_data = data[['Age', 'Time', 'Amount', 'Entry_CVC', 'Entry_PIN']]
db_data
```

[ ]:

	Age	Time	Amount	Entry_CVC	Entry_PIN
0	25.2	19	5	0	0
1	49.6	17	288	0	1
2	42.2	14	5	0	0

```
3      51.0    14      28           0          0
4      38.0    23      91           1          0
…      …   …    …       …        …
99995  53.8    22      15           0          0
99996  45.0    23       7           0          1
99997  46.5    11      21           0          1
99998  48.2    22      25           0          0
99999  31.7    16     226           0          1

[99977 rows x 5 columns]
```

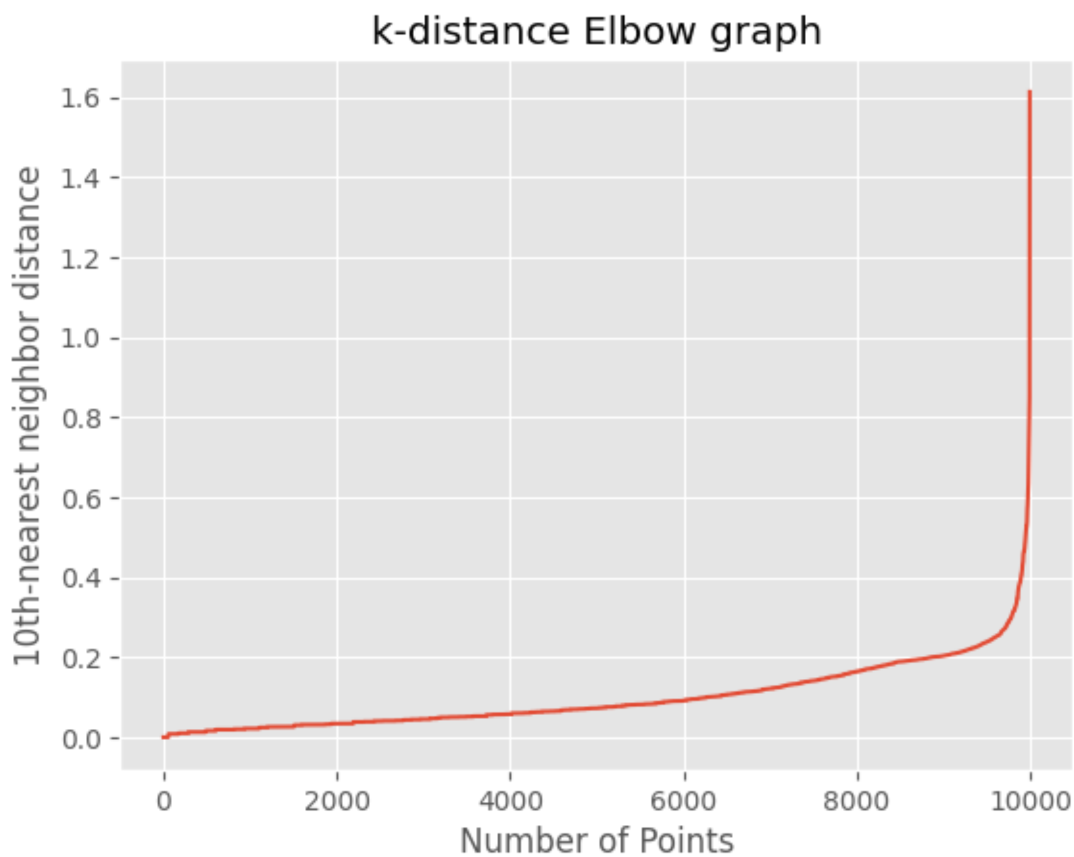
[ ]:
```
#DBSCAN
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()


db_scaled = scaler.fit_transform(db_data)

#Restoring scaled dataset into a pd DataFrame with its column names
scaled_df = pd.DataFrame(db_scaled, columns = db_data.columns)

#Reducing sample size for DBSCAN computation
scaled_df = scaled_df.sample(n=10000)
scaled_df
```

[ ]:
```
            Age      Time    Amount  Entry_CVC  Entry_PIN
9920  -0.069722  1.024250 -0.741928  -0.709405   -0.99955
13157 -0.612540  0.270696  2.304209   1.409632   -0.99955
2929  -0.321027 -0.482859 -0.733827  -0.709405   -0.99955
3468   0.774663 -1.424802 -0.069510  -0.709405    1.00045
4328  -0.119983  0.270696  0.554300  -0.709405    1.00045
…           …        …        …         …          …
61460 -0.733167 -0.106082 -0.790537   1.409632   -0.99955
65955 -0.140087 -1.236414 -0.669016   1.409632   -0.99955
25978  0.312262  0.647473 -0.758131  -0.709405    1.00045
25716  0.060957  1.589416 -0.839146   1.409632   -0.99955
56057 -0.461757 -0.859636 -0.717624   1.409632   -0.99955

[10000 rows x 5 columns]
```

[ ]:
```
from sklearn.neighbors import NearestNeighbors

nei = NearestNeighbors(n_neighbors = 10)
nei_fit = nei.fit(scaled_df)
distances, indices = nei_fit.kneighbors(scaled_df)
```

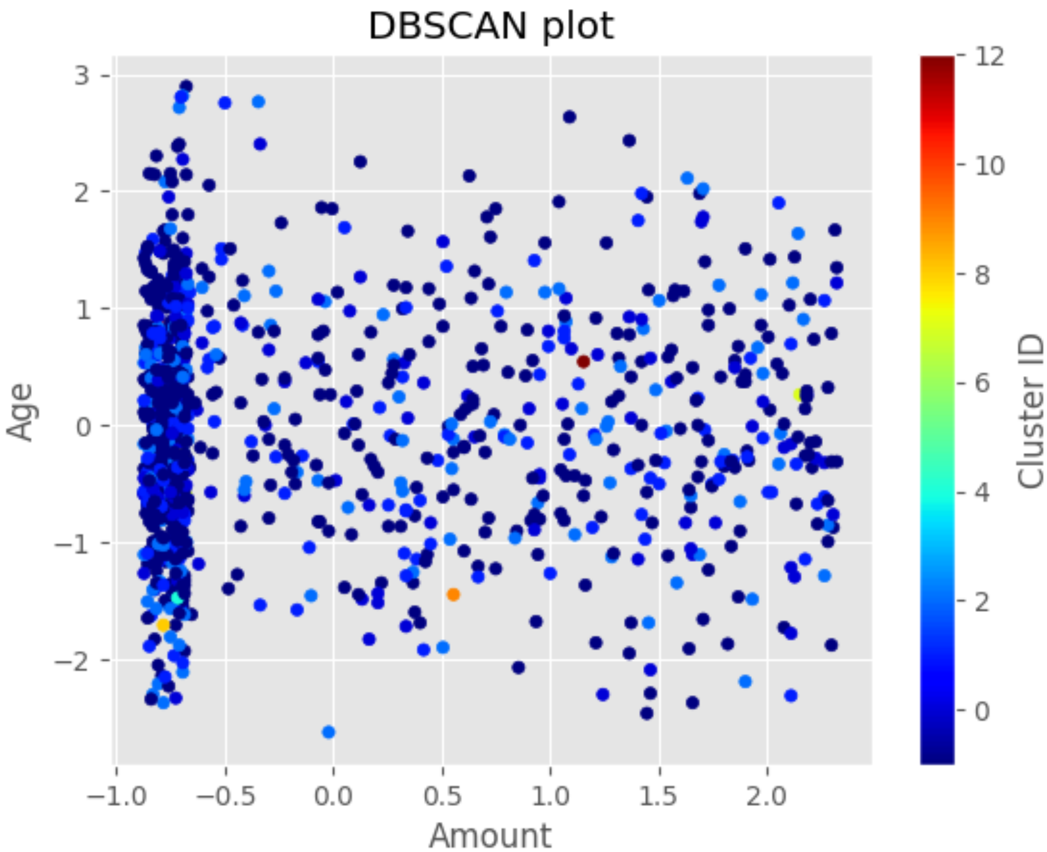
```
distances = np.sort(distances, axis = 0)
distances = distances[:,1]
plt.title('k-distance Elbow graph')
plt.xlabel('Number of Points')
plt.ylabel('10th-nearest neighbor distance')
plt.plot(distances)
```

```
[ ]: [<matplotlib.lines.Line2D at 0x79f680704e10>]
```




```
[ ]: db = DBSCAN(eps = 0.2, min_samples=10).fit(scaled_df)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
db_labels = pd.DataFrame(db.labels_, columns = ['Cluster ID'])

db_result = pd.concat((scaled_df, db_labels), axis=1)
db_result.plot.scatter(x = 'Amount', y = 'Age', c = 'Cluster ID', colormap =␣
↪'jet', title = 'DBSCAN plot')
```

```
[ ]: <Axes: title={'center': 'DBSCAN plot'}, xlabel='Amount', ylabel='Age'>
```



DBSCAN plot
Age
Amount
Cluster ID
3
2
1
0
−1
−2
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
12
10
8
6
4
2
0