```
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
path=("/content/drive/MyDrive/Walmart /PS_20174392719_1491204439457_log.csv")
```

Double-click (or enter) to edit

```
import pandas as pd
import numpy as np
import os
import csv
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
import random
import missingno as msno
import xgboost as xgb
from sklearn import *
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import train_test_split
from scipy.stats.mstats import winsorize
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVC
from sklearn.inspection import permutation_importance
from xgboost import plot_importance, plot_tree
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import OneClassSVM
from sklearn.preprocessing import StandardScaler
from keras.models import Model
from keras.layers import Input, Dense
from tensorflow.keras.models import Sequential
from sklearn.metrics import roc_curve, auc
from tensorflow.keras.layers import Dense
from sklearn.decomposition import PCA
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.metrics import roc_curve, auc
from sklearn.datasets import make_classification
from keras.models import Sequential
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_recall_curve, average_precision_score
```

```
#Reading the CSV
data=pd.read_csv(path)
data
```

Displaying the first 10 values in the dataset to study the data

```
data.head(10)
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isF
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.0	0.00	0	
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.0	0.00	0	
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.0	0.00	1	
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.0	0.00	1	
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.0	0.00	0	
5	1	PAYMENT	7817.71	C90045638	53860.00	46042.29	M573487274	0.0	0.00	0	
6	1	PAYMENT	7107.77	C154988899	183195.00	176087.23	M408069119	0.0	0.00	0	
7	1	PAYMENT	7861.64	C1912850431	176087.23	168225.59	M633326333	0.0	0.00	0	
8	1	PAYMENT	4024.36	C1265012928	2671.00	0.00	M1176932104	0.0	0.00	0	
9	1	DEBIT	5337.77	C712410124	41720.00	36382.23	C195600860	41898.0	40348.79	0	

Displaying the last 10 values in the dataset to study the data

```
data.tail(10)
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFr
6362610	742	TRANSFER	63416.99	C778071008	63416.99	0.0	C1812552860	0.00	0.00	
6362611	742	CASH_OUT	63416.99	C994950684	63416.99	0.0	C1662241365	276433.18	339850.17	
6362612	743	TRANSFER	1258818.82	C1531301470	1258818.82	0.0	C1470998563	0.00	0.00	
6362613	743	CASH_OUT	1258818.82	C1436118706	1258818.82	0.0	C1240760502	503464.50	1762283.33	
6362614	743	TRANSFER	339682.13	C2013999242	339682.13	0.0	C1850423904	0.00	0.00	
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	C776919290	0.00	339682.13	
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C1881841831	0.00	0.00	
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C1365125890	68488.84	6379898.11	
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C2080388513	0.00	0.00	
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	C873221189	6510099.11	7360101.63	

Displaying the basic information about the datas in the dataset

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype  
---  ------          -----  
 0   step            int64  
 1   type            object 
 2   amount          float64
 3   nameOrig        object 
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        object 
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64  
 10  isFlaggedFraud  int64  
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

Displaying the number of rows and columns in the dataset

```
data.shape
```

```
(6362620, 11)
```

Displaying statistical information about the dataset

```
data.describe()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.224996e+06	1.290820e-03	2.514687e-06
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674129e+06	3.590480e-02	1.585775e-03
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.146614e+05	0.000000e+00	0.000000e+00
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.111909e+06	0.000000e+00	0.000000e+00
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.561793e+08	1.000000e+00	1.000000e+00

Finding the missing values and summing if any

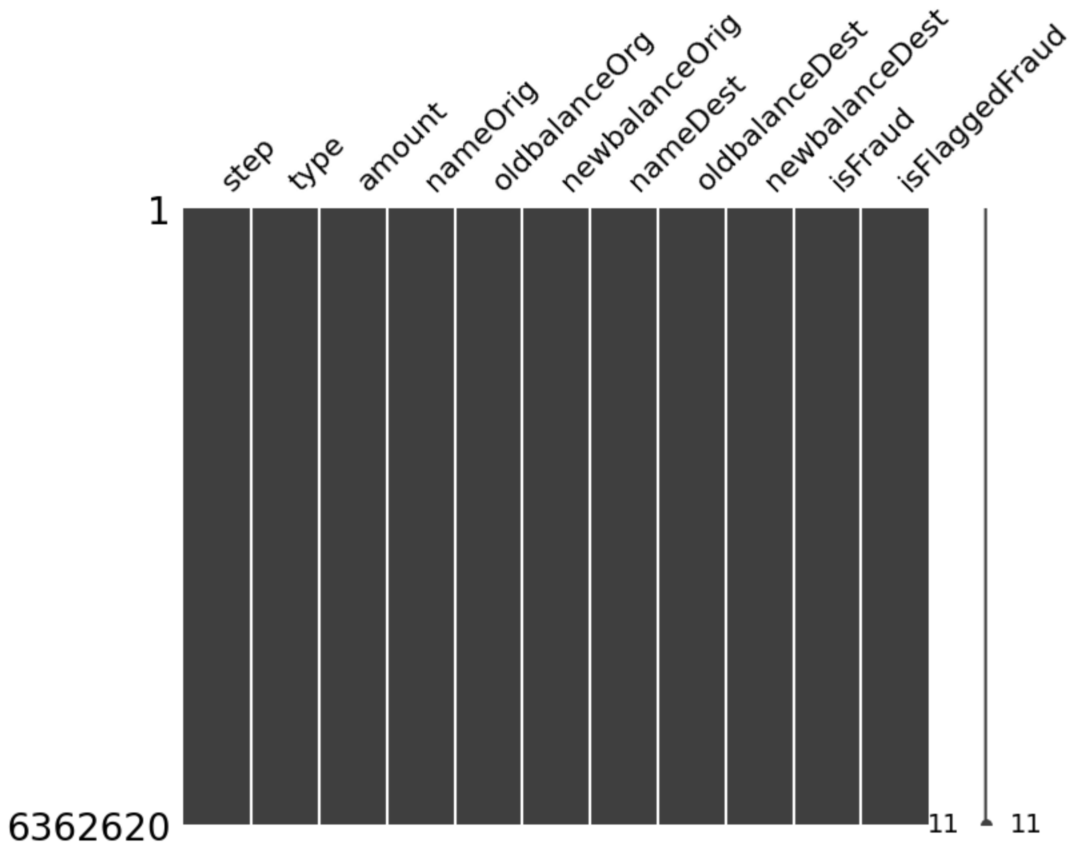
```
#checking for missing values
data.isnull().sum()
```

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

Plotting graph to display the missing values

```
#checking for missing values
msno.matrix(data,figsize=(8, 6))
```

```
<Axes: >
```



```
correlation = data.corr()
correlation
```

```
ıing: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only v
```

ıceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
)10058	-0.010299	0.027665	0.025888	0.031578	0.003277
)02762	-0.007861	0.294137	0.459304	0.076688	0.012295
)00000	0.998803	0.066243	0.042029	0.010154	0.003835
)98803	1.000000	0.067812	0.041837	-0.008148	0.003776
)66243	0.067812	1.000000	0.976569	-0.005885	-0.000513
)42029	0.041837	0.976569	1.000000	0.000535	-0.000529
)10154	-0.008148	-0.005885	0.000535	1.000000	0.044109
)03835	0.003776	-0.000513	-0.000529	0.044109	1.000000

```
# checking type column categories
data["type"].unique()
```

```
array(['PAYMENT', 'TRANSFER', 'CASH_OUT', 'DEBIT', 'CASH_IN'],
      dtype=object)
```

```
# getting the categories in type column
unique_categories = data['type'].unique()
print(unique_categories)
```

```
['PAYMENT' 'TRANSFER' 'CASH_OUT' 'DEBIT' 'CASH_IN']
```

```
data = pd.DataFrame(data)

# Find duplicate rows based on all columns
duplicate_rows = data[data.duplicated()]

# Display the duplicate rows
print("Duplicate Rows except first occurrence:")
print(duplicate_rows)
```

```
Duplicate Rows except first occurrence:
Empty DataFrame
Columns: [0]
Index: []
```

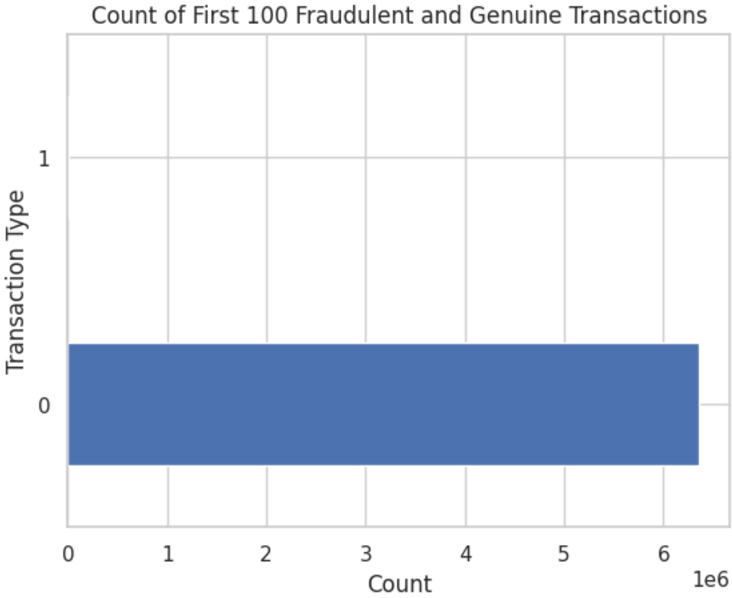
```
quantity = data['type'].values
print(quantity)
```

```
['PAYMENT' 'PAYMENT' 'TRANSFER' ... 'CASH_OUT' 'TRANSFER' 'CASH_OUT']
```

As we see above there is no NULL Values in data, we dont have to think about filling NAs we can proceed to EDA

```
data['isFraud'].value_counts().plot(kind="barh")
plt.xlabel('Count')
plt.ylabel('Transaction Type')
plt.title('Fraudulent and Genuine Transactions')
plt.show()
```


```
data['isFraud'].value_counts().head(100).plot(kind="barh")
plt.xlabel('Count')
plt.ylabel('Transaction Type')
plt.title('Count of First 100 Fraudulent and Genuine Transactions')
plt.show()
```



The Feature isFlaggedFound is a categorical feature and that is the target feature too, this is a classification based problem

```
data["isFlaggedFraud"].value_counts()
```

```
0    6362604
1         16
Name: isFlaggedFraud, dtype: int64
```

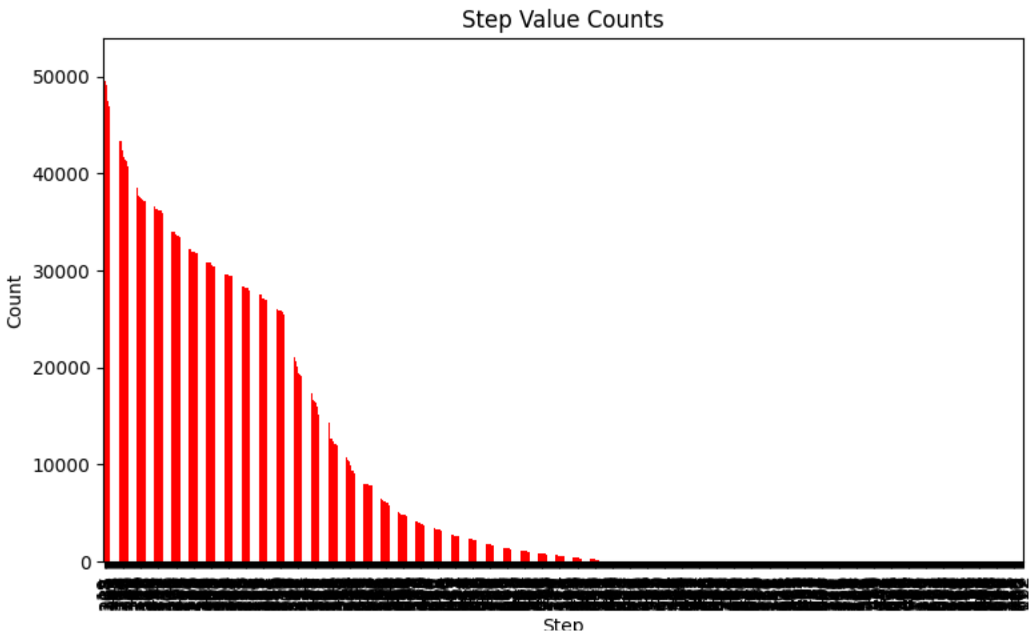
While the Target Column isFlaggedFraud is represented in bar graph, we can find that the data for fraud transaction is less (less than 10%) . The data for non fraud transaction will tend to dominate the fraud transactions.

Exploratory Data Analysis

```
data["step"].value_counts()
```

```
19     51352
18     49579
187    49083
235    47491
307    46968
       ...
432        4
706        4
693        4
112        2
662        2
Name: step, Length: 743, dtype: int64
```

```
plt.rcParams["figure.figsize"] = [8,5]
plt.rcParams["figure.autolayout"] = True
fig, ax = plt.subplots()
data["step"].value_counts().plot(ax=ax, kind="bar", color='red')
plt.title('Step Value Counts')
plt.xlabel('Step')
plt.ylabel('Count')
plt.show()
```




```
data["nameDest"].value_counts()
```

```
C1286084959    113
C985934102     109
C665576141     105
C2083562754    102
C248609774     101
              ...
M1470027725      1
M1330329251      1
M1784358659      1
M2081431099      1
C2080388513      1
Name: nameDest, Length: 2722362, dtype: int64
```

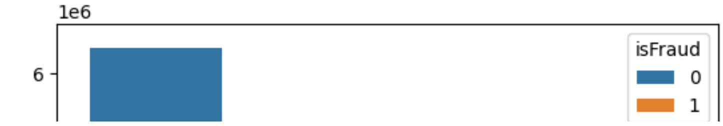
```
data["nameOrig"].value_counts()
```

```
C1902386530    3
C363736674     3
C545315117     3
C724452879     3
C1784010646    3
              ..
C98968405      1
C720209255     1
C1567523029    1
C644777639     1
C1280323807    1
Name: nameOrig, Length: 6353307, dtype: int64
```

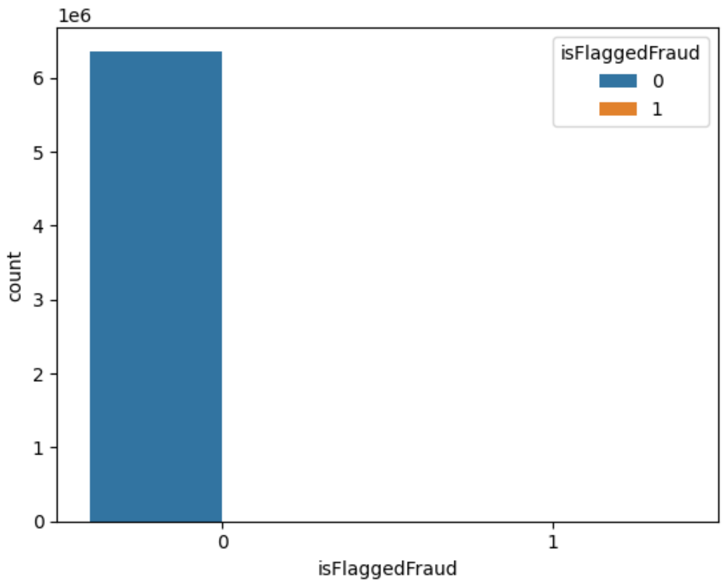
```
data['isFraud'].value_counts()
```

```
0    6354407
1       8213
Name: isFraud, dtype: int64
```

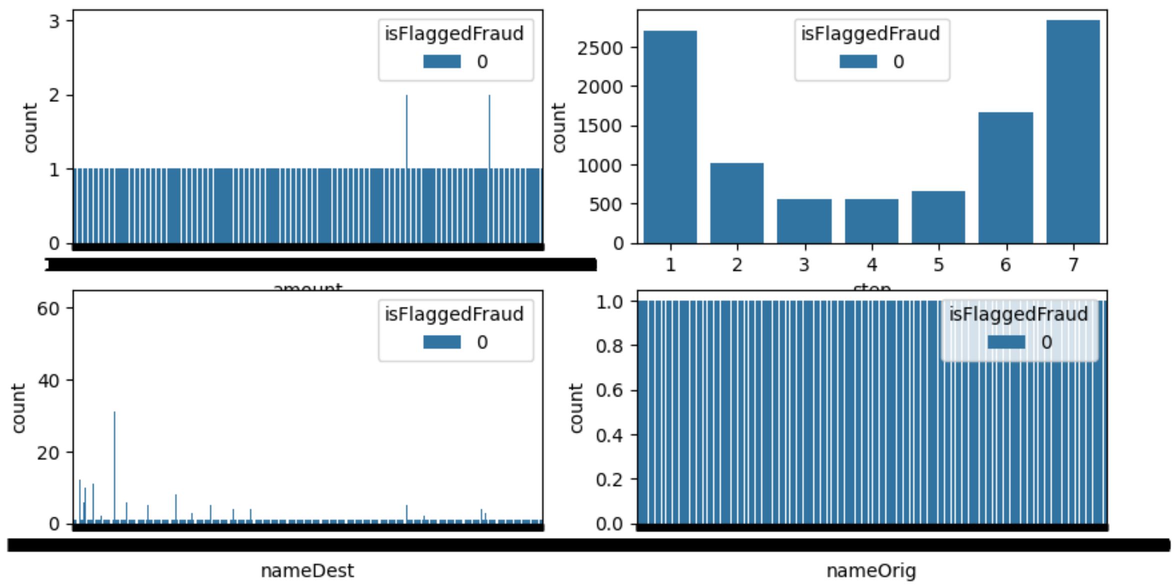
```
sns.countplot(x = 'isFraud', hue = 'isFraud', data = data);
```



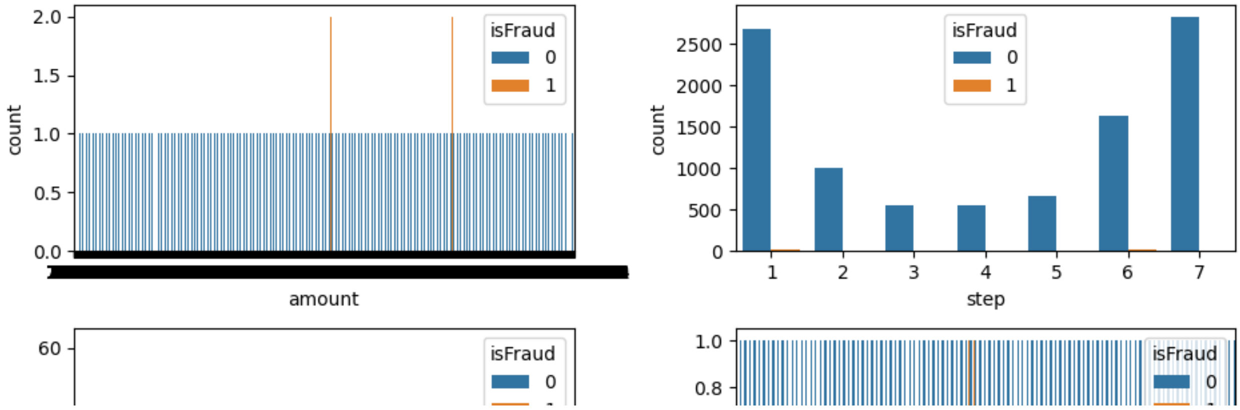

```
sns.countplot(x = 'isFlaggedFraud', hue = 'isFlaggedFraud', data = data);
```




```
fig, axes = plt.subplots(2, 2, figsize = (10,5))
sns.countplot(ax = axes[0, 0], x = 'amount', hue = 'isFlaggedFraud', data = data.head(10000))
sns.countplot(ax = axes[0, 1], x = 'step', hue = 'isFlaggedFraud', data = data.head(10000))
sns.countplot(ax = axes[1, 0], x = 'nameDest', hue = 'isFlaggedFraud', data = data.head(10000))
sns.countplot(ax = axes[1, 1], x = 'nameOrig', hue = 'isFlaggedFraud', data = data.head(10000))
plt.show()
```

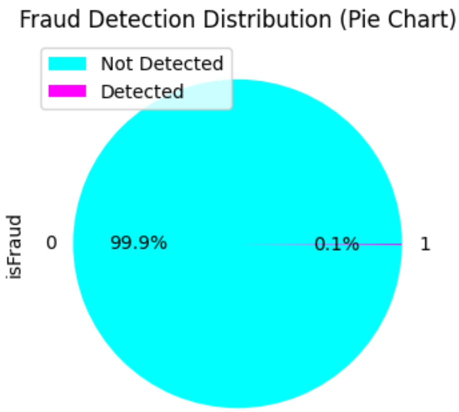



```
fig, axes = plt.subplots(2, 2, figsize = (10, 5))
sns.countplot(ax = axes[0, 0], x = 'amount', hue = 'isFraud', data = data.head(10000))
sns.countplot(ax = axes[0, 1], x = 'step', hue = 'isFraud', data = data.head(10000))
sns.countplot(ax = axes[1, 0], x = 'nameDest', hue = 'isFraud', data = data.head(10000))
sns.countplot(ax = axes[1, 1], x = 'nameOrig', hue = 'isFraud', data = data.head(10000))
plt.show()
```

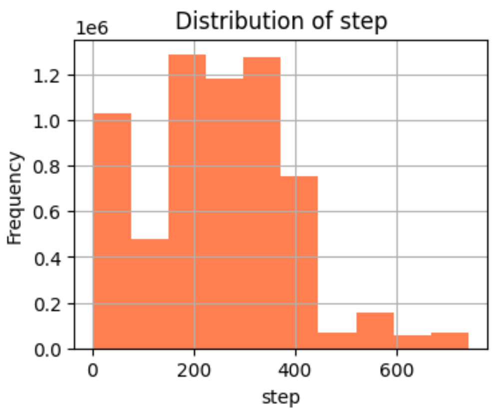


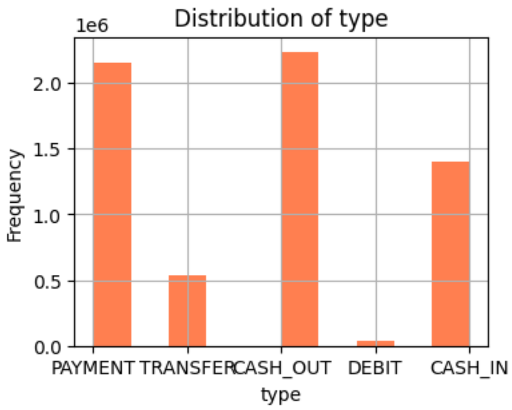
lternatively we can look at the pie chart. 99.8% of the transactions are normal, while only 0.2% are fraudulent.

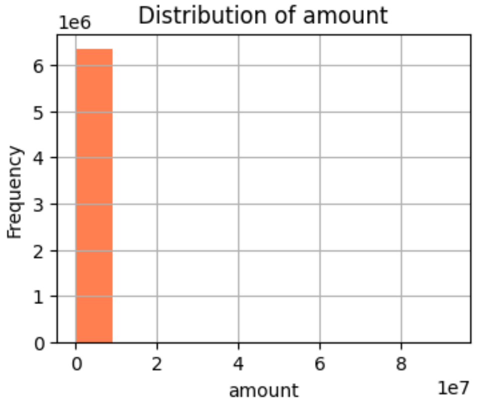
```
#Checking for class imbalence in target feature
fraud_count = data["isFraud"].value_counts()
colors = ['cyan','magenta']

#piechart
plt.figure(figsize=(6,4))
fraud_count.plot(kind='pie', autopct='%1.1f%%', colors=colors)
plt.legend(loc='upper left', labels=['Not Detected', 'Detected'])
plt.title('Fraud Detection Distribution (Pie Chart)')
plt.show()

# Bar chart (linear scale)
plt.figure(figsize=(6,4))
ax = fraud_count.plot(kind='bar', color=colors, ylabel='Fraud', log=False)
plt.title('Fraud Detection Distribution (Linear Scale)')
plt.show()

# Bar chart (log scale)
plt.figure(figsize=(6,4))
ax = fraud_count.plot(kind='bar', color=colors, ylabel='Fraud (LOG)', log=True)
plt.title('Fraud Detection Distribution (Log Scale)')
plt.show()
```



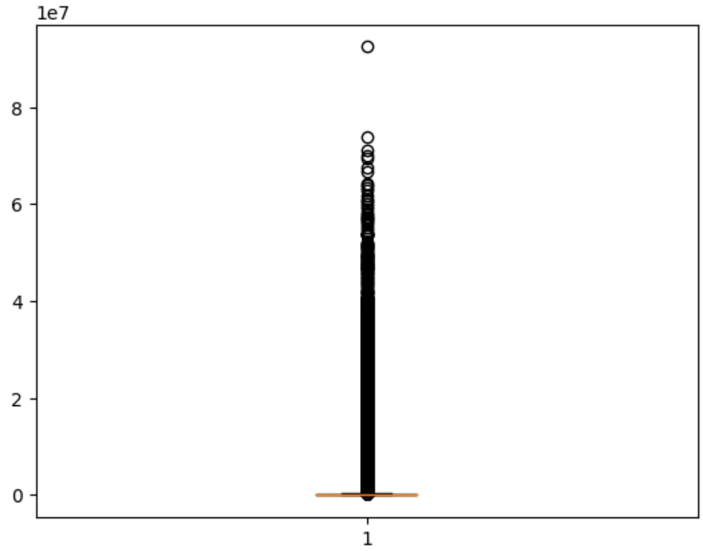

```
#Histogram
def plotPerColumnDistribution(dataframe, color='yellow', figsize=(6, 4)):
    for column in dataframe.columns:
        plt.figure(figsize=figsize)
        dataframe[column].hist(color=color)
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.show()
plotPerColumnDistribution(data, color='coral', figsize=(4, 3))
```



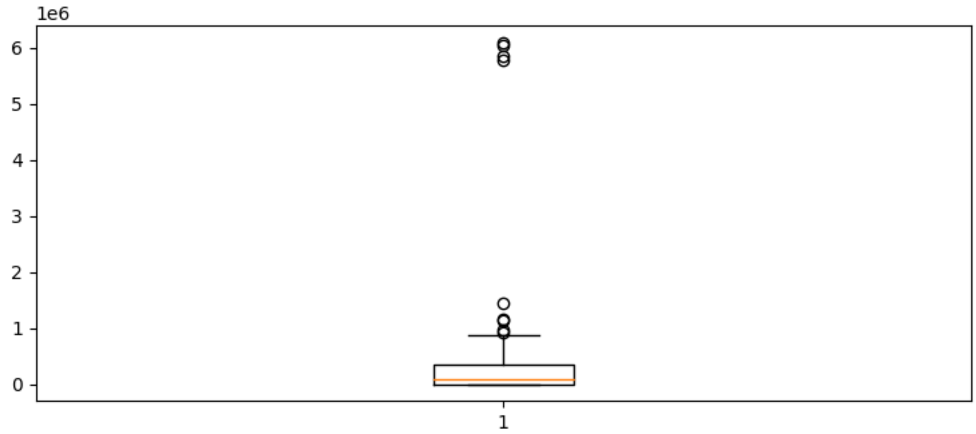




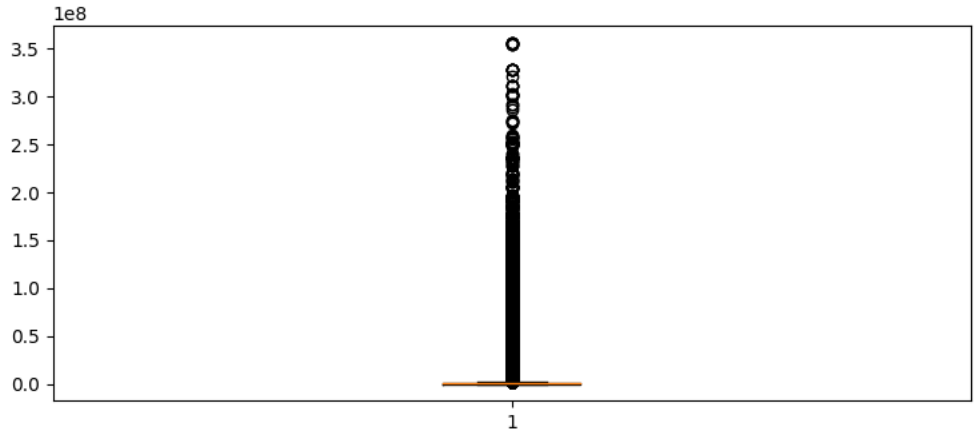
```
#Checking for Outliers
plt.boxplot(data["amount"])
plt.show()
```




```
plt.boxplot(data["oldbalanceDest"][100:200])
plt.show()
```



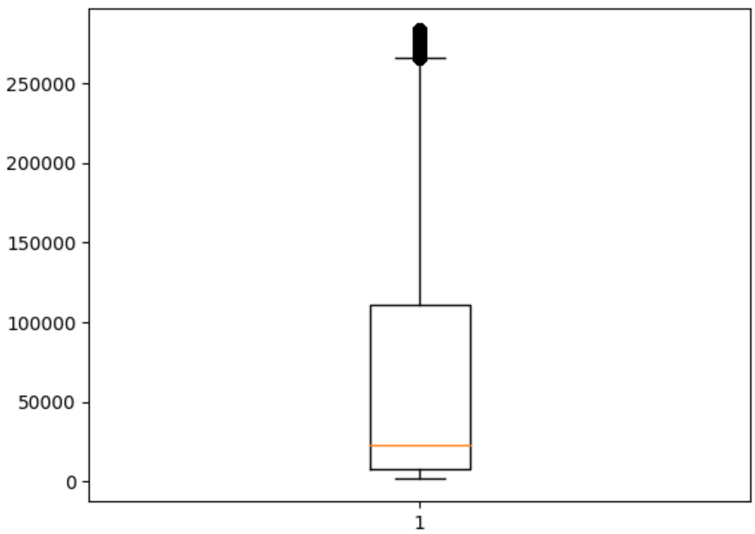

```
plt.boxplot(data["newbalanceDest"])
plt.show()
```



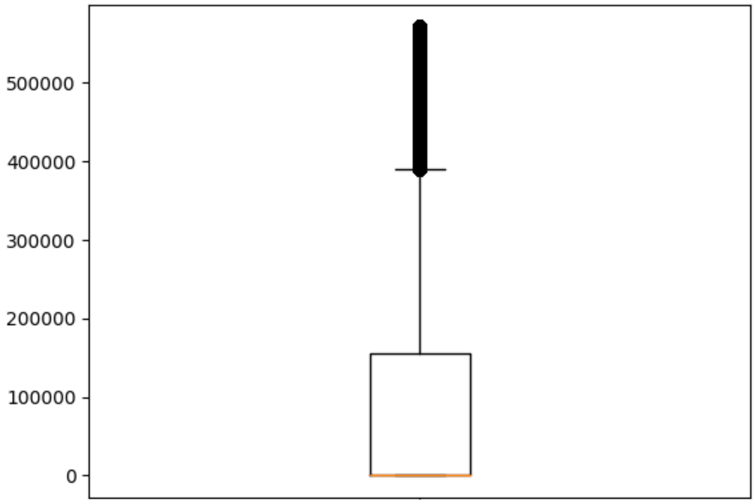

```
#handling outliers using WINSORIZE
numeric_columns = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']
for column in numeric_columns:
    data[column + '_winsorized'] = winsorize(data[column], limits=[0.05, 0.05])
print(data)
```

```
6362611   742  CASH_OUT  63416.99   C994950684       63416.99            0.00

            nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud  \
1        M2044282225            0.00            0.00        0               0
2         C553264065            0.00            0.00        1               0
3          C38997010        21182.00            0.00        1               0
4        M1230701703            0.00            0.00        0               0
8        M1176932104            0.00            0.00        0               0
...              ...             ...             ...      ...             ...
6362597  C2109905271       513746.19       562189.07        1               0
6362604  C1930074465            0.00            0.00        1               0
6362605   C830041824            0.00        54652.46        1               0
6362610  C1812552860            0.00            0.00        1               0
6362611  C1662241365       276433.18       339850.17        1               0

         amount_winsorized  oldbalanceOrg_winsorized  \
1                  1864.28                  21249.00
2                  1546.72                    181.00
3                  1546.72                    181.00
4                 11668.14                  41554.00
8                  4024.36                   2671.00
...                    ...                       ...
6362597           48442.88                  48442.88
6362604           54652.46                  51695.00
6362605           54652.46                  51695.00
6362610           63416.99                  51695.00
6362611           63416.99                  51695.00

         newbalanceOrig_winsorized  oldbalanceDest_winsorized  \
1                         19020.12                       0.00
2                             0.00                       0.00
3                             0.00                   21182.00
4                         19020.12                       0.00
8                             0.00                       0.00
...                            ...                        ...
6362597                       0.00                  513746.19
6362604                       0.00                       0.00
6362605                       0.00                       0.00
6362610                       0.00                       0.00
6362611                       0.00                  276433.18

         newbalanceDest_winsorized
1                             0.00
2                             0.00
3                             0.00
4                             0.00
8                             0.00
...                            ...
6362597                  562189.07
6362604                       0.00
6362605                   54652.46
6362610                       0.00
6362611                  339850.17

[2759812 rows x 16 columns]
```

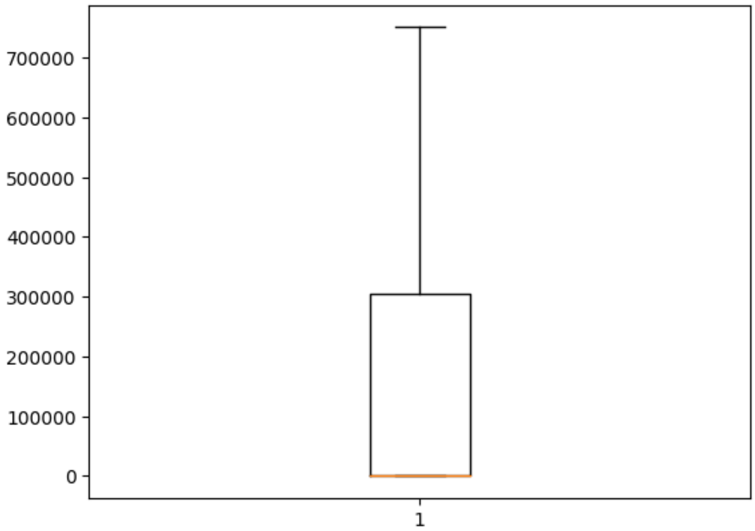
```
plt.boxplot(data["amount_winsorized"])
plt.show()
```




```
plt.boxplot(data["oldbalanceDest_winsorized"])
plt.show()
```




```
plt.boxplot(data["newbalanceDest_winsorized"])
plt.show()
```



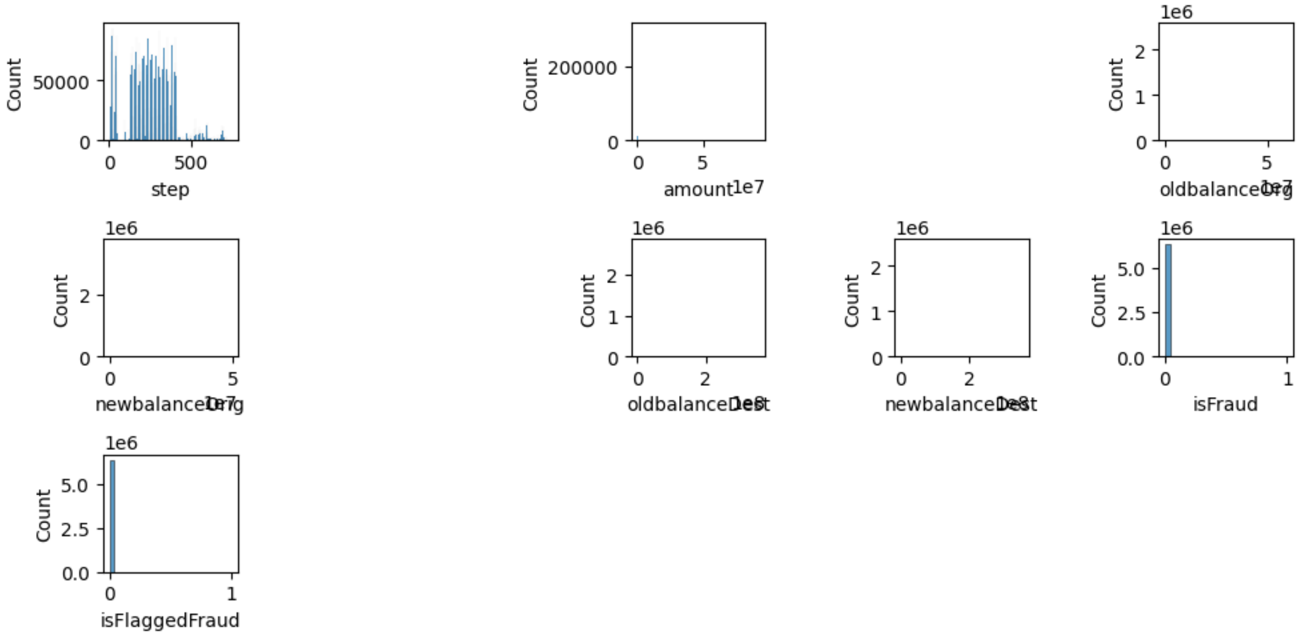
Creating a scatter plot using the 'amount' as the x-axis and 'isFraud' as the y-axis. The color of each point is determined by the 'isFraud' column, using a colormap

```
plt.scatter(data['amount'], data['isFraud'], c=data['isFraud'], cmap='bwr')
plt.title('Scatter Plot')
plt.xlabel('amount')
plt.ylabel('isFraud')
plt.colorbar(label='Fraud Status')
plt.show()
```

This code generates a grid of histograms for numeric columns in the dataset, providing a quick overview of the distribution of each numerical variable. Non-numeric columns are skipped, and the layout is organized in a 3x5 grid for better visualization.

```
# checking numerical features distribution
plt.figure(figsize=(10, 5))
plotnumber = 1
for column in data.columns:
    if plotnumber <= 14:
        if data[column].dtype in [np.float64, np.int64]:
            ax = plt.subplot(3, 5, plotnumber)
            sns.histplot(data[column])
            plt.xlabel(column)
        else:
            print(f"Skipping non-numeric column: {column}")

    plotnumber += 1
plt.tight_layout()
plt.show()
```

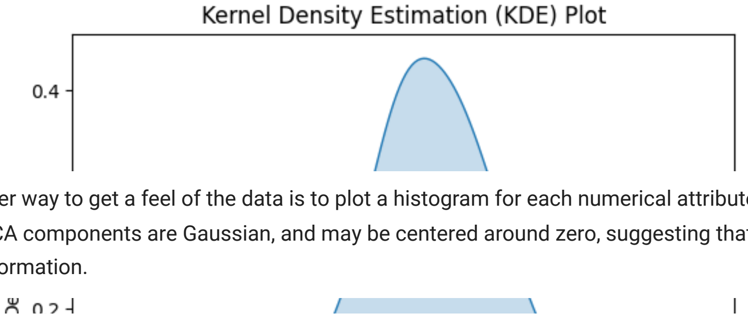
```
Skipping non-numeric column: type
Skipping non-numeric column: nameOrig
Skipping non-numeric column: nameDest
```



This code generates a Kernel Density Estimation (KDE) plot for a dataset of 100 randomly generated numbers. The shaded area represents the estimated probability density function, providing insights into the underlying distribution.

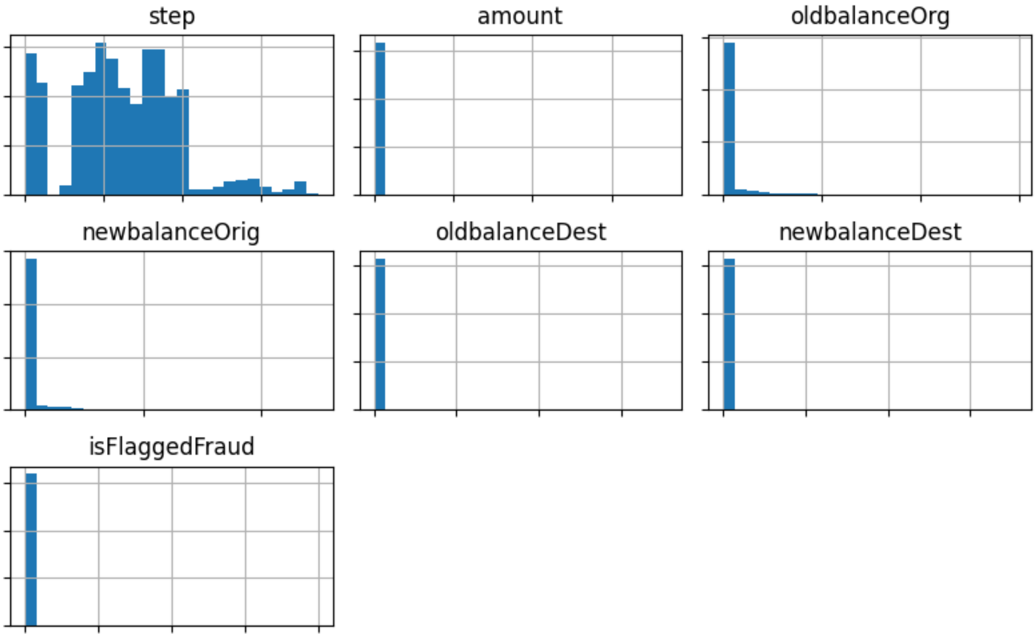
```
np.random.seed(42)
data = np.random.randn(100)
sns.kdeplot(data, shade=True, label='KDE Plot')
plt.xlabel('X-axis Label')
plt.ylabel('Density')
plt.title('Kernel Density Estimation (KDE) Plot')
plt.show()
```

```
<ipython-input-5-3d8b84e0d485>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data, shade=True, label='KDE Plot')
```



Another way to get a feel of the data is to plot a histogram for each numerical attribute. For step to isFlaggedFraud, the distributions of most of the PCA components are Gaussian, and may be centered around zero, suggesting that the attributes have been normalized by the PCA transformation.

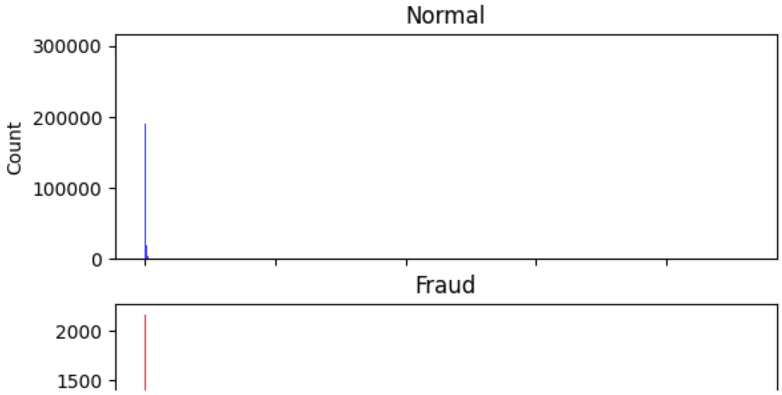
```
ax = data.drop('isFraud', axis=1).hist(bins=25, figsize=(8, 5))
for axis in ax.flatten():
    axis.set_xticklabels([])
    axis.set_yticklabels([])
plt.show()
```



Double-click (or enter) to edit

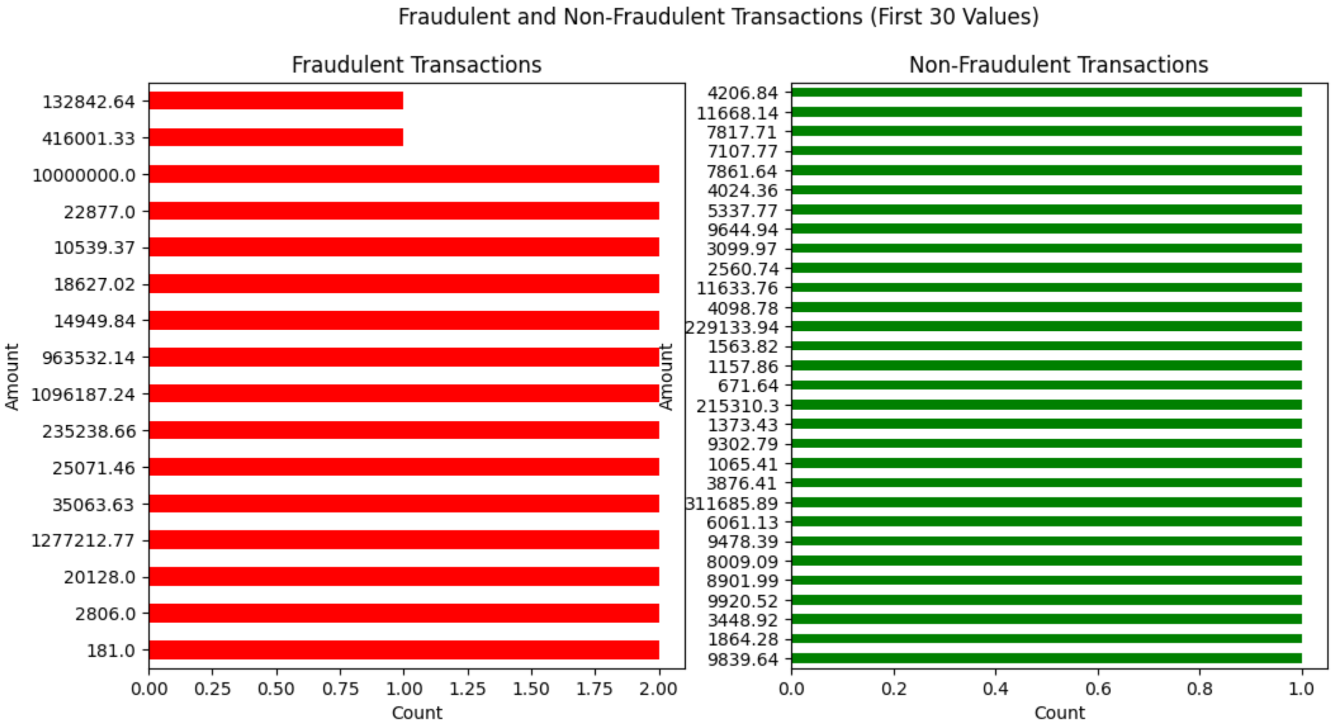
Both normal and fraud transactions happen across all the time span. The normal transations show some pattern. Compared with normal transactions above, the fraud's pattern is not very clear

```
normal = data[data['isFraud'] == 0]
fraud = data[data['isFraud'] == 1]

fig, axes = plt.subplots(2, 1, sharex=True)

# Plot histogram for 'Time' in the 'Normal' case
axes[0].set_title('Normal')
sns.histplot(ax=axes[0], data=normal, x="amount", color='b')

# Plot histogram for 'Time' in the 'Fraud' case
axes[1].set_title('Fraud')
sns.histplot(ax=axes[1], data=fraud, x="amount", color='r')
plt.show()
```



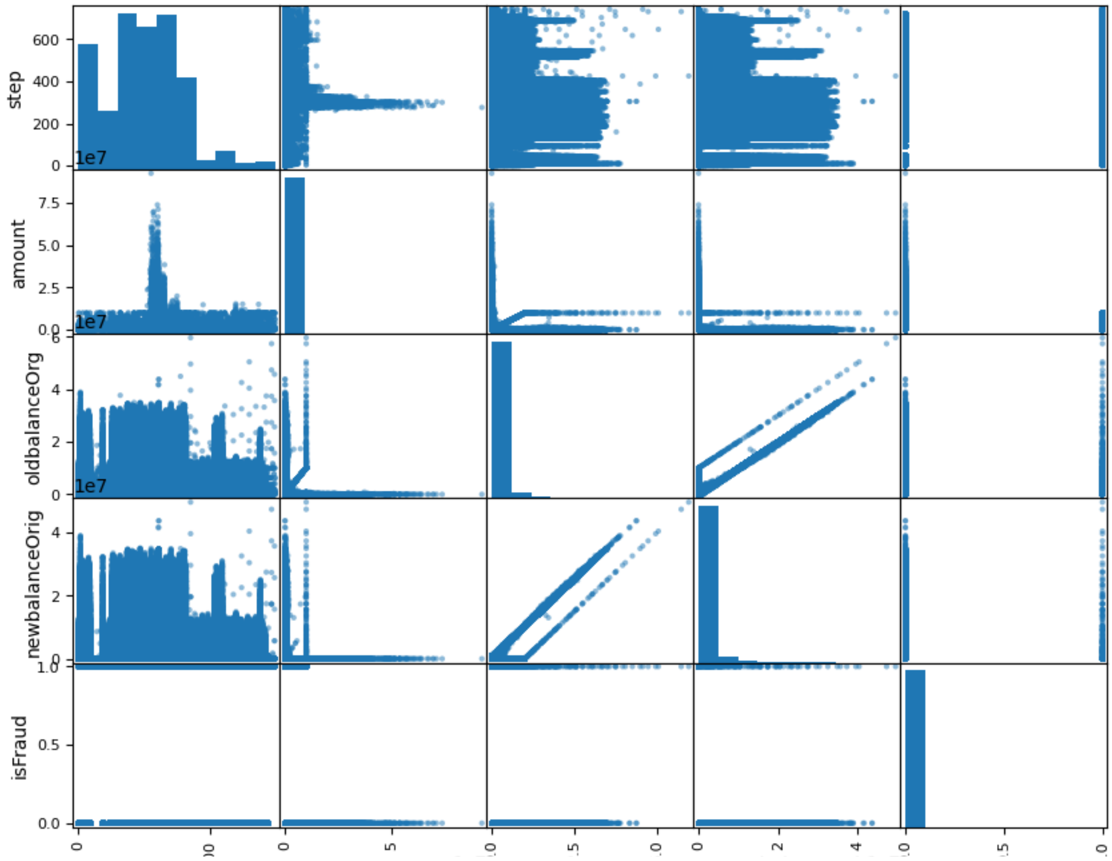
The side-by-side bar chart compares the distribution of transaction amounts for the first 30 values in both fraudulent and non-fraudulent transactions. The left plot ('Fraudulent Transactions') indicates the count of unique amounts in red, while the right plot ('Non-Fraudulent Transactions') shows the count in green. This visualization provides insights into the frequency and variation of transaction amounts for both fraudulent and non-fraudulent cases in the sampled data

```
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Plotting for fraudulent transactions
data[data['isFraud'] == 1].head(30)['amount'].value_counts().plot(kind="barh", ax=axes[0], color='r')
axes[0].set_title('Fraudulent Transactions')
axes[0].set_xlabel('Count')
axes[0].set_ylabel('Amount')

# Plotting for non-fraudulent transactions
data[data['isFraud'] == 0].head(30)['amount'].value_counts().plot(kind="barh", ax=axes[1], color='g')
axes[1].set_title('Non-Fraudulent Transactions')
axes[1].set_xlabel('Count')
axes[1].set_ylabel('Amount')

plt.suptitle('Fraudulent and Non-Fraudulent Transactions (First 30 Values)')
plt.show()
```



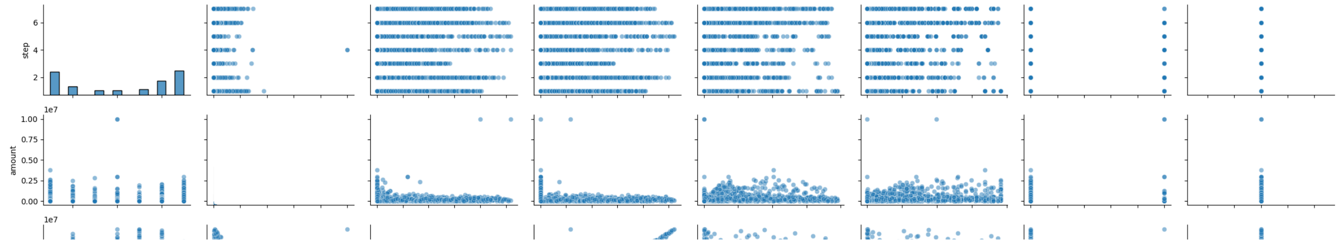
This scatter matrix displays pairwise relationships among the selected columns step, amount, oldbalanceOrg, newbalanceOrig, isFraud

```
columns_of_interest = ['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'isFraud']
scatter_matrix(data[columns_of_interest], alpha=0.5, figsize=(10, 8), diagonal='hist')
plt.show()
```



Here, we used Pair Plot because each variable in the dataset is compared with every other variable.

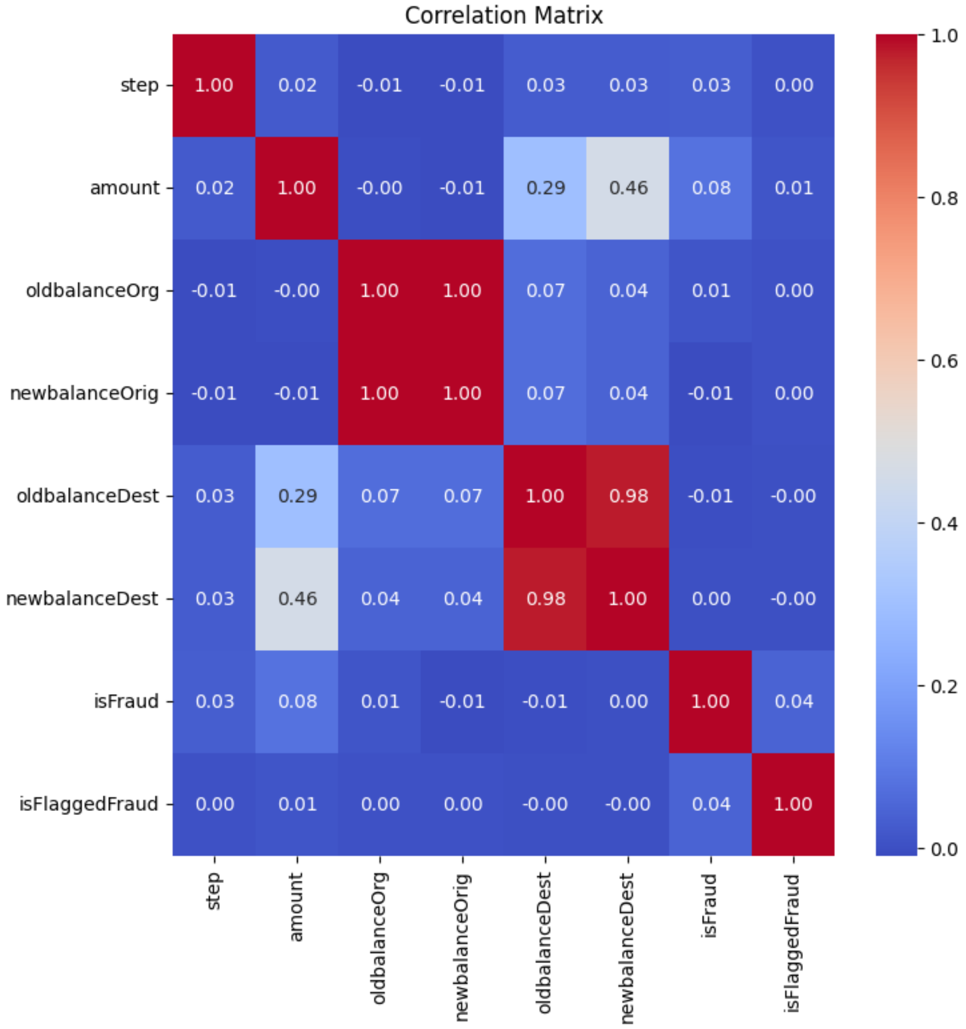
```
new_data = data.head(10000)
sns.pairplot(new_data, plot_kws={'alpha': 0.5},height=2, aspect=1.5)
plt.show()
```



To visualize the correlation matrix of a dataset we used Heatmap.Heatmap is a graphical representation of data where values in a matrix are represented as colors.

```
# Heatmap: Correlation matrix
correlation_matrix = data.corr()
plt.figure(figsize=(8, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

```
<ipython-input-49-50d324c42451>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  correlation_matrix = data.corr()
```



By using confusion matrix and classification report, we are performing on the test set, providing insights into the model's ability to accurately identify fraud (1) and non-fraud (0) instances.

```
X = data[['amount']]
y = data['isFraud']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = IsolationForest(contamination=0.1, random_state=42)
model.fit(X_train_scaled)
y_pred = model.predict(X_test_scaled)
y_pred_binary = [1 if pred == -1 else 0 for pred in y_pred]
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_binary))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_binary))
```

```
Confusion Matrix:
[[1144441  126463]
 [    746     874]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.90      0.95   1270904
           1       0.01      0.54      0.01      1620

    accuracy                           0.90   1272524
   macro avg       0.50      0.72      0.48   1272524
weighted avg       1.00      0.90      0.95   1272524
```

```
def classify_fraud(row):
    if row['isFraud'] == 1:
        return 'Fraud'
    else:
        return 'Non-Fraud'
data['isFraud'] = data.apply(classify_fraud, axis=1)
print(data['isFraud'])
```

```
0          Non-Fraud
1          Non-Fraud
2          Non-Fraud
3          Non-Fraud
4          Non-Fraud
             ...
6362615    Non-Fraud
6362616    Non-Fraud
6362617    Non-Fraud
6362618    Non-Fraud
6362619    Non-Fraud
Name: isFraud, Length: 6362620, dtype: object
```

```
non_fraud_rows = data[data['isFraud'] == 0]
fraud_rows = data[data['isFraud'] == 1]
print("Non-Fraud values:")
print(non_fraud_rows)
print("\nFraud values:")
print(fraud_rows)
```

```
Non-Fraud values:
Empty DataFrame
Columns: [step, type, amount, nameOrig, oldbalanceOrg, newbalanceOrig, nameDest, oldbalanceDest, newbalanceDest, isFraud, isFlagged
Index: []

Fraud values:
Empty DataFrame
Columns: [step, type, amount, nameOrig, oldbalanceOrg, newbalanceOrig, nameDest, oldbalanceDest, newbalanceDest, isFraud, isFlagged
Index: []
```

```
x_df=[]
y_df=[]

x_df=data.drop(['isFraud'],axis=1)
y_df=data["isFraud"]
len(data)
ind_col=x_df.columns
length=data.shape[0]

for i in range(4):
  var=y_df[i]
  if(var==1):
    x_df.append(x_df)
    print(var,x_df.iloc[i,])
  else:
    print(var,x_df.iloc[i,])
```

```
0 step                      1
type                  PAYMENT
amount                9839.64
nameOrig          C1231006815
oldbalanceOrg        170136.0
newbalanceOrig      160296.36
nameDest          M1979787155
oldbalanceDest            0.0
newbalanceDest            0.0
isFlaggedFraud              0
Name: 0, dtype: object
0 step                      1
type                  PAYMENT
amount                1864.28
nameOrig          C1666544295
oldbalanceOrg         21249.0
newbalanceOrig       19384.72
nameDest          M2044282225
oldbalanceDest            0.0
newbalanceDest            0.0
isFlaggedFraud              0
Name: 1, dtype: object
<ipython-input-40-7709cc6acd97>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future
  x_df.append(x_df)
1 step                      1
type                 TRANSFER
amount                  181.0
nameOrig          C1305486145
oldbalanceOrg           181.0
newbalanceOrig            0.0
nameDest           C553264065
oldbalanceDest            0.0
newbalanceDest            0.0
isFlaggedFraud              0
Name: 2, dtype: object
1 step                     1
type                CASH_OUT
amount                 181.0
nameOrig          C840083671
oldbalanceOrg          181.0
newbalanceOrig           0.0
nameDest           C38997010
oldbalanceDest       21182.0
newbalanceDest           0.0
isFlaggedFraud             0
Name: 3, dtype: object
```

```
len(data)
```

```
6362620
```

```
data.head(4)
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFl
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1	

```
random_data = []

# Selecting 15,000 random samples
random_samples = np.random.choice(data.index, size=15000, replace=False)

for i in random_samples:
    label = data.loc[i, 'isFraud']
    random_data.append(data.loc[i, :])
random_data = pd.DataFrame(random_data)
```

```
print(random_data)
```

```
          step      type      amount     nameOrig  oldbalanceOrg  \
1640081    157   PAYMENT    17131.25  C2057763801      706288.82
1520712    153  TRANSFER  1084092.43   C663148259           0.00
1300767    136  TRANSFER     7295.26  C1415385963           0.00
1943910    177  CASH_OUT   219137.03  C1585181145           0.00
909492      43   CASH_IN    81628.50  C1881519428      160819.93
...        ...       ...         ...          ...            ...
333718      16   PAYMENT    11502.00   C455856147           0.00
1158166    131   PAYMENT     7559.39  C1193613165           0.00
2329339    188   CASH_IN   152018.89   C553607766      254418.76
425760      18   PAYMENT    25781.16  C1278549836           0.00
3555051    260   PAYMENT     6100.25   C412379240       15356.26

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
1640081       689157.57  M1972679676            0.00            0.00        0
1520712            0.00  C1313381818      1179602.94      2070891.54        0
1300767            0.00  C1060928555      2104850.76      2557383.04        0
1943910            0.00  C1828825845      2864275.22      3293748.45        0
909492        242448.42  C1802243408      1500630.36      1419001.86        0
...                 ...          ...             ...             ...      ...
333718             0.00  M1061942061            0.00            0.00        0
1158166            0.00   M284190298            0.00            0.00        0
2329339       406437.65   C980696620        41067.53            0.00        0
425760             0.00   M344673576            0.00            0.00        0
3555051         9256.01  M1950952818            0.00            0.00        0

         isFlaggedFraud
1640081               0
1520712               0
1300767               0
1943910               0
909492                0
...                 ...
333718                0
1158166               0
2329339               0
425760                0
3555051               0

[15000 rows x 11 columns]
```

```
len(random_data)
```

```
15000
```

```
random_data['normalisednewbalanceDest']=winsorize(random_data["newbalanceDest"], limits=[0.05, 0.05])
```

```
plt.boxplot(random_data["normalisednewbalanceDest"])
plt.show()
```

1e6

```
columns_to_label_encode = ['type','nameOrig', 'nameDest']

label_encoder = LabelEncoder()

for column in columns_to_label_encode:
    random_data[column + '_encoded'] = label_encoder.fit_transform(random_data[column])

encoded = random_data.drop(columns=columns_to_label_encode)

print(encoded)
```

```
         step     amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  \
1366043   138   29683.45           0.00            0.00            0.00
5524443   381    2601.89      935543.75       932941.86       603570.04
3547935   260  456830.95      105867.00            0.00            0.00
645036     35  175821.76       23867.00            0.00       591846.73
3576469   261   15953.15       19318.00         3364.85            0.00
...       ...        ...            ...             ...             ...
3401413   255  176377.29       11621.00       187998.29      1391652.23
4565085   327   53312.95         287.00            0.00            0.00
2344837   189   12180.70     1266013.65      1278194.36       151782.59
1881224   164  125525.36       23635.00            0.00       379393.93
4772286   335   63077.59       90389.00        27311.41            0.00

         newbalanceDest  isFraud  isFlaggedFraud  type_encoded  \
1366043            0.00        0               0             3
5524443       606171.93        0               0             1
3547935       456830.95        0               0             4
645036        767668.49        0               0             1
3576469            0.00        0               0             3
...                 ...      ...             ...           ...
3401413      1215274.93        0               0             0
4565085        53312.95        0               0             1
2344837       139601.89        0               0             0
1881224       504919.29        0               0             1
4772286        63077.59        0               0             4

         nameOrig_encoded  nameDest_encoded
1366043             12022             11271
5524443              9671              7181
3547935               787              6302
645036               5682              8590
3576469              9712             12022
...                   ...               ...
3401413              2156              4209
4565085              8292              2100
2344837             12680              4757
1881224             14213               886
4772286             13144              5709

[15000 rows x 11 columns]
```

```
random_data.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFra
2071126	181	CASH_OUT	219626.67	C322687638	100057.0	0.0	C1668430365	0.0	219626.67	
2817355	225	PAYMENT	47414.04	C1289239103	0.0	0.0	M1766511204	0.0	0.00	
3590604	262	CASH_OUT	96921.22	C554595301	21506.0	0.0	C1362505632	25490.6	122411.82	
3915939	284	PAYMENT	12945.10	C286946403	13005.0	59.9	M936907676	0.0	0.00	
280416	15	PAYMENT	18596.89	C1693319373	4179.0	0.0	M848645043	0.0	0.00	

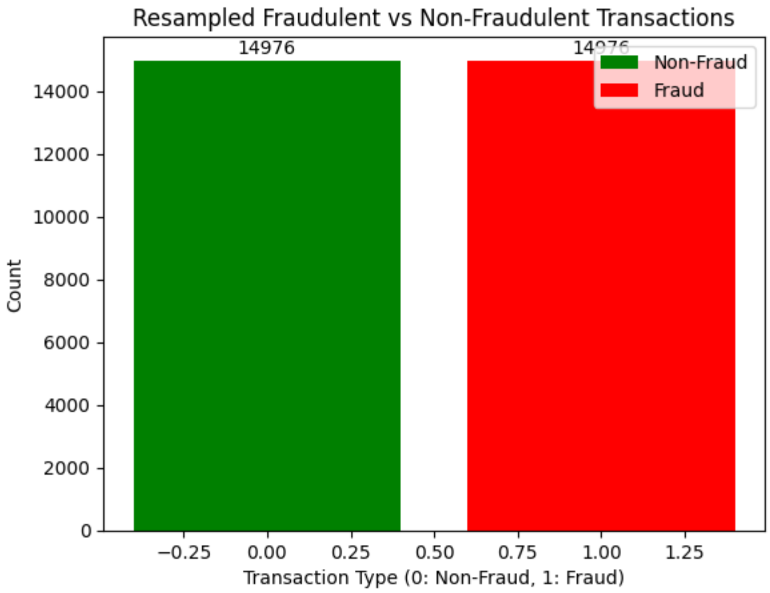
```
print(encoded)
```


```
         step     amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  \
1366043   138   29683.45           0.00            0.00            0.00
5524443   381    2601.89      935543.75       932941.86       603570.04
3547935   260  456830.95      105867.00            0.00            0.00
645036     35  175821.76       23867.00            0.00       591846.73
3576469   261   15953.15       19318.00         3364.85            0.00
...       ...        ...            ...             ...             ...
3401413   255  176377.29       11621.00       187998.29      1391652.23
4565085   327   53312.95         287.00            0.00            0.00
2344837   189   12180.70     1266013.65      1278194.36       151782.59
1881224   164  125525.36       23635.00            0.00       379393.93
4772286   335   63077.59       90389.00        27311.41            0.00

         newbalanceDest  isFraud  isFlaggedFraud  type_encoded  \
1366043            0.00        0               0             3
5524443       606171.93        0               0             1
```

```
3547935    456830.95    0    0    4
645036     767668.49    0    0    1
3576469         0.00    0    0    3
...              ...  ...  ...  ...
3401413   1215274.93    0    0    0
4565085     53312.95    0    0    1
2344837    139601.89    0    0    0
1881224    504919.29    0    0    1
4772286     63077.59    0    0    4

         nameOrig_encoded  nameDest_encoded
1366043             12022             11271
5524443              9671              7181
3547935               787              6302
645036               5682              8590
3576469              9712             12022
...                   ...               ...
3401413              2156              4209
4565085              8292              2100
2344837             12680              4757
1881224             14213               886
4772286             13144              5709

[15000 rows x 11 columns]
```

```
smote=SMOTE(random_state=42)
model_data=random_data.drop(['isFraud','type','nameOrig','nameDest'],axis=1)
X_train_resampled , Y_train_resampled=smote.fit_resample(model_data,random_data["isFraud"])
```

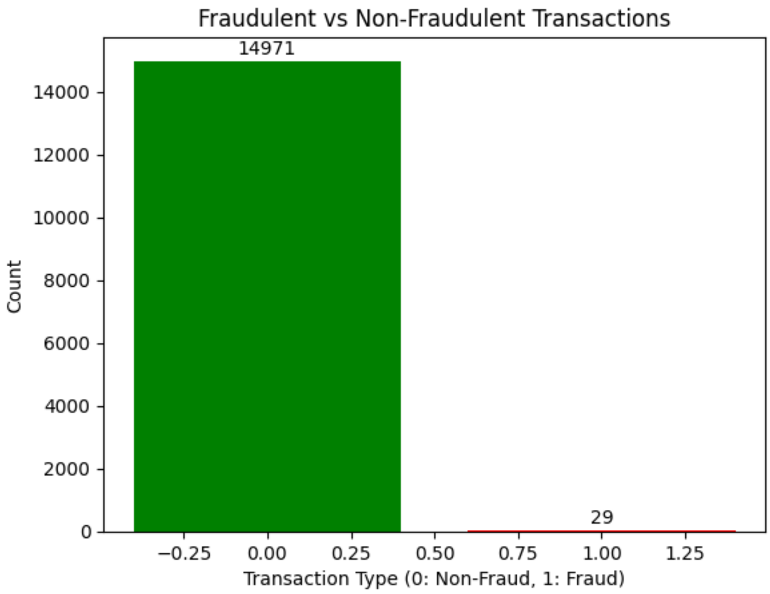
```
smote = SMOTE(random_state=42)
model_data = random_data.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)
X_train_resampled, Y_train_resampled = smote.fit_resample(model_data, random_data["isFraud"])
resampled_data = pd.DataFrame(X_train_resampled, columns=model_data.columns)
resampled_data['isFraud'] = Y_train_resampled
fraud_counts_resampled = resampled_data['isFraud'].value_counts()
plt.bar(fraud_counts_resampled.index, fraud_counts_resampled, color=['green', 'red'], label=['Non-Fraud', 'Fraud'])
plt.title('Resampled Fraudulent vs Non-Fraudulent Transactions')
plt.xlabel('Transaction Type (0: Non-Fraud, 1: Fraud)')
plt.ylabel('Count')

for i, count in enumerate(fraud_counts_resampled):
    plt.text(i, count + 100, str(count), ha='center', va='bottom')
plt.legend()
plt.show()
```




```
count_0=0
count_1=0
for i in Y_train_resampled:
  if i==1:
    count_0=count_0+1
  else:
    count_1=count_1+1
print(count_0)
print(count_1)
```

```
14976
14976
```

```
fraud_counts = random_data['isFraud'].value_counts()
plt.bar(fraud_counts.index, fraud_counts, color=['green', 'red'])
plt.title('Fraudulent vs Non-Fraudulent Transactions')
plt.xlabel('Transaction Type (0: Non-Fraud, 1: Fraud)')
plt.ylabel('Count')
for i, count in enumerate(fraud_counts):
    plt.text(i, count + 100, str(count), ha='center', va='bottom')
plt.show()
```



MODEL

```
X_train = random_data.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)
Y_train = random_data['isFraud']
X_train, X_test, Y_train, Y_test = train_test_split(X_train, Y_train, test_size=0.2, random_state=42)
print("X_train shape:", X_train.shape)
print("Y_train shape:", Y_train.shape)
print("X_test shape:", X_test.shape)
print("Y_test shape:", Y_test.shape)
```

```
X_train shape: (12000, 10)
Y_train shape: (12000,)
X_test shape: (3000, 10)
Y_test shape: (3000,)
```

```
X_train
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFlaggedFraud	type_encoded	nameOrig_enco
4549333	327	873536.39	0.00	0.00	4403933.62	5277470.00	0	4	13
3691525	277	169080.64	0.00	0.00	248825.50	417906.14	0	1	10
3842681	282	37908.28	0.00	0.00	0.00	0.00	0	3	10
4193188	305	36930.53	0.00	0.00	228051.87	264982.40	0	1	8
4760342	334	18945.78	41651.00	22705.22	0.00	0.00	0	3	3
...	...	...	...	...	...	...	...	...	
3423227	256	2810.74	239782.82	236972.08	0.00	0.00	0	3	[illegible]
5944232	405	6815.93	646.00	0.00	0.00	0.00	0	3	8
3534990	259	73783.46	22150.00	0.00	3241546.23	3315329.68	0	1	11
1655752	158	281690.26	0.00	0.00	2813053.26	3094743.52	0	1	3
2448609	203	604735.72	13468.00	0.00	1557133.76	2161869.48	0	4	7

12000 rows × 10 columns

```
Y_train
```

```
4549333    0
3691525    0
3842681    0
4193188    0
4760342    0
          ..
3423227    0
5944232    0
3534990    0
1655752    0
2448609    0
Name: isFraud, Length: 12000, dtype: int64
```

LogisticRegression

```
model = LogisticRegression(random_state=42)
model.fit(X_train, Y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
conf_matrix = confusion_matrix(Y_test, y_pred)
classification_rep = classification_report(Y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 1.0
Confusion Matrix:
 [[2995    0]
 [   0    5]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2995
           1       1.00      1.00      1.00         5

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

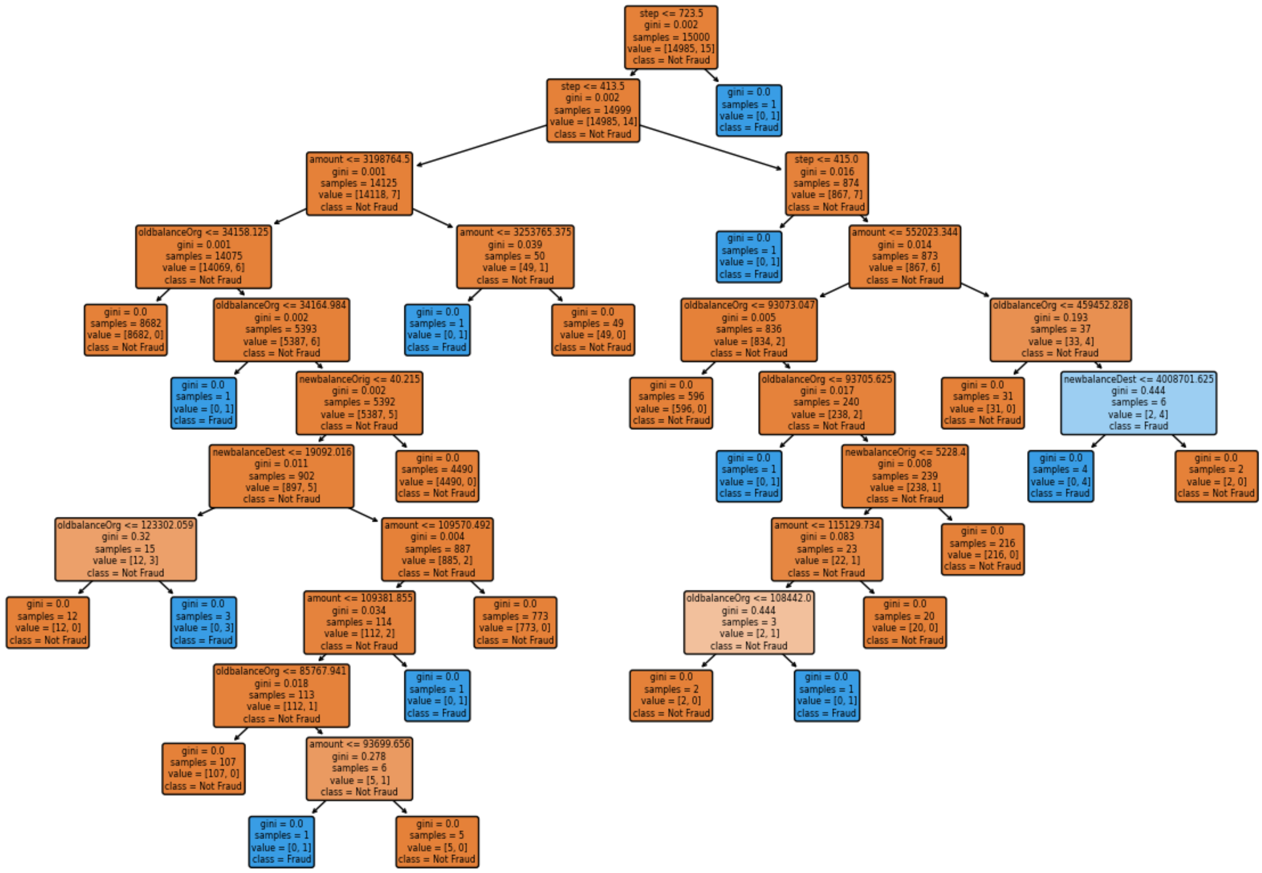
```
X, y = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0, random_state=42)
model = LogisticRegression()
model.fit(X, y)
plt.figure(figsize=(6, 4))
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', cmap=plt.cm.Paired)
plt.title('Logistic Regression Decision Boundary')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

DecisionTreeClassifier

```
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, Y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
conf_matrix = confusion_matrix(Y_test, y_pred)
classification_rep = classification_report(Y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.9993333333333333
Confusion Matrix:
 [[2994    1]
 [   1    4]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2995
           1       0.80      0.80      0.80         5

    accuracy                           1.00      3000
   macro avg       0.90      0.90      0.90      3000
weighted avg       1.00      1.00      1.00      3000
```

```
X = random_data.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)
y = random_data['isFraud']
model = DecisionTreeClassifier(random_state=42)
model.fit(X, y)
plt.figure(figsize=(15, 10))
plot_tree(model, feature_names=X.columns, class_names=['Not Fraud', 'Fraud'], filled=True, rounded=True)
plt.show()
```

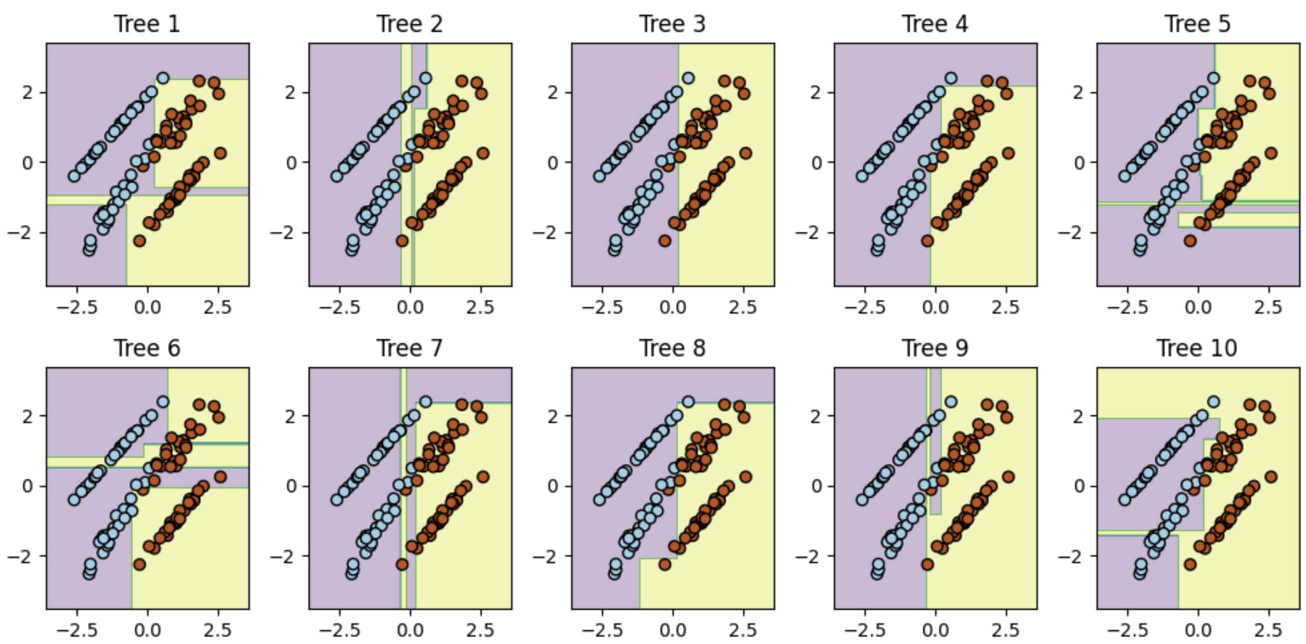


RandomForestClassifier

```
model = RandomForestClassifier(random_state=42)
model.fit(X_train, Y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
conf_matrix = confusion_matrix(Y_test, y_pred)
classification_rep = classification_report(Y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.9983333333333333
Confusion Matrix:
 [[2995    0]
 [   5    0]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2995
           1       0.00      0.00      0.00         5

    accuracy                           1.00      3000
   macro avg       0.50      0.50      0.50      3000
weighted avg       1.00      1.00      1.00      3000

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
```

```
X, y = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0, random_state=42)
model = RandomForestClassifier(n_estimators=10, random_state=42)
model.fit(X, y)
plt.figure(figsize=(10, 5))
for tree_idx, tree in enumerate(model.estimators_):
    plt.subplot(2, 5, tree_idx + 1)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
    Z = tree.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', cmap=plt.cm.Paired)
    plt.title(f'Tree {tree_idx + 1}')

plt.tight_layout()
plt.show()
```

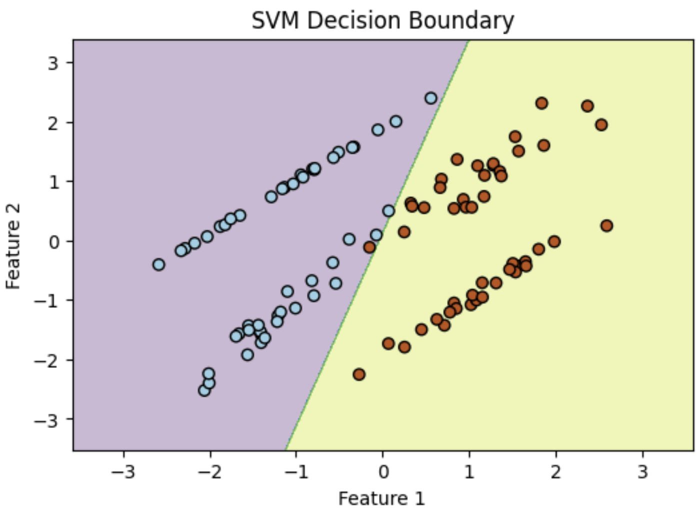


SVC

```
model = SVC(random_state=42)
model.fit(X_train, Y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
conf_matrix = confusion_matrix(Y_test, y_pred)
classification_rep = classification_report(Y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.9983333333333333
Confusion Matrix:
 [[2995    0]
 [   5    0]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2995
           1       0.00      0.00      0.00         5

    accuracy                           1.00      3000
   macro avg       0.50      0.50      0.50      3000
weighted avg       1.00      1.00      1.00      3000

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
```

```
X, y = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0, random_state=42)
model = SVC(kernel='linear', C=1)
model.fit(X, y)
plt.figure(figsize=(6, 4))
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', cmap=plt.cm.Paired)
plt.title('SVM Decision Boundary')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

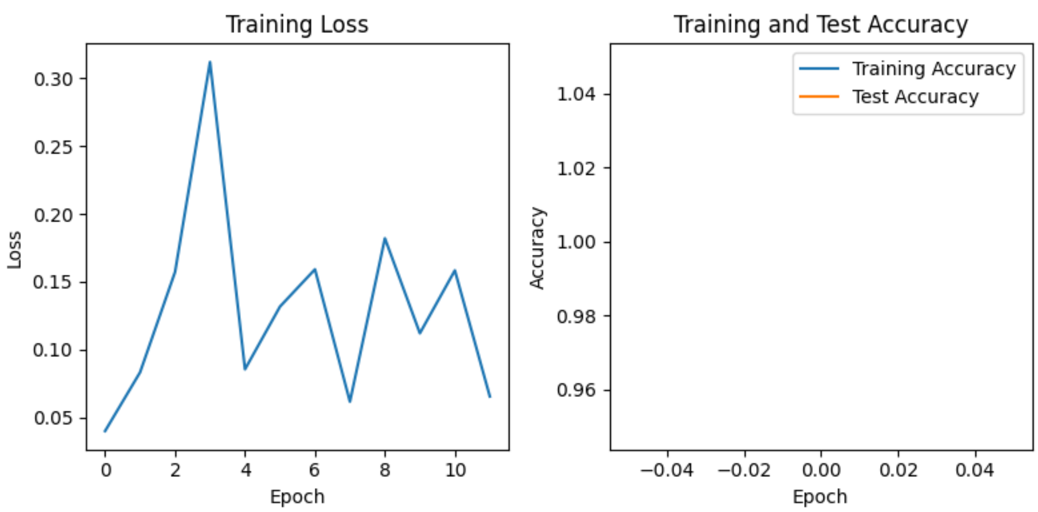


MPL Classifer

```
model = MLPClassifier(random_state=42)
model.fit(X_train, Y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
conf_matrix = confusion_matrix(Y_test, y_pred)
classification_rep = classification_report(Y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.9986666666666667
Confusion Matrix:
 [[2993    2]
 [   2    3]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2995
           1       0.60      0.60      0.60         5

    accuracy                           1.00      3000
   macro avg       0.80      0.80      0.80      3000
weighted avg       1.00      1.00      1.00      3000
```

```
model = MLPClassifier(random_state=42)
model.fit(X_train, Y_train)
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(model.loss_curve_)
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')

plt.subplot(1, 2, 2)
plt.plot(model.score(X_train, Y_train), label='Training Accuracy')
plt.plot(accuracy, label='Test Accuracy')
plt.title('Training and Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



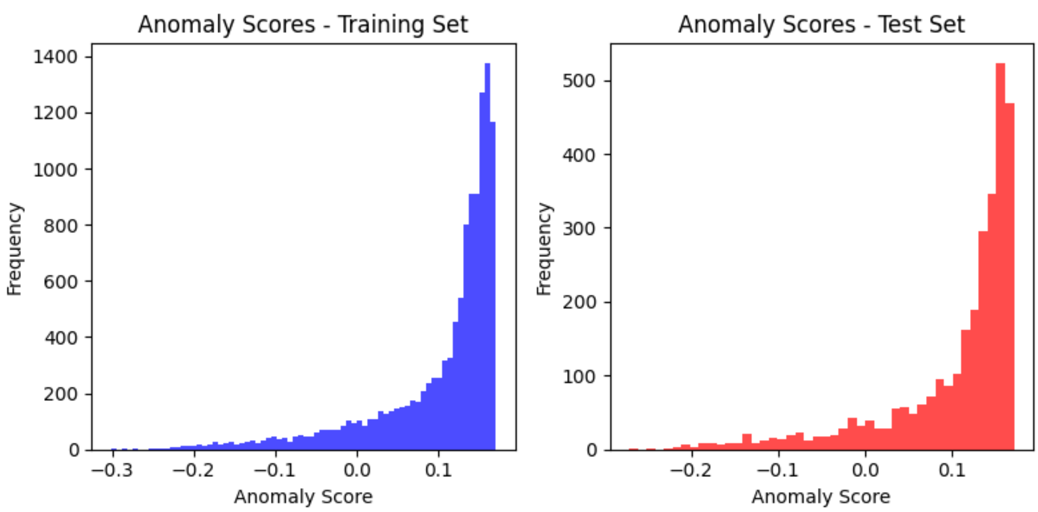
IsolationForest

```
model = IsolationForest(random_state=42)
model.fit(X_train)
y_pred = model.predict(X_test)
y_pred_binary = [1 if pred == -1 else 0 for pred in y_pred]
accuracy = accuracy_score(Y_test, y_pred_binary)
conf_matrix = confusion_matrix(Y_test, y_pred_binary)
classification_rep = classification_report(Y_test, y_pred_binary)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.8863333333333333
Confusion Matrix:
 [[2657  338]
 [   3    2]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.89      0.94      2995
           1       0.01      0.40      0.01         5

    accuracy                           0.89      3000
   macro avg       0.50      0.64      0.48      3000
weighted avg       1.00      0.89      0.94      3000
```

```
model = IsolationForest(random_state=42)
model.fit(X_train)
anomaly_scores_train = model.decision_function(X_train)
anomaly_scores_test = model.decision_function(X_test)
threshold = 0.0
y_pred_binary_train = [1 if score < threshold else 0 for score in anomaly_scores_train]
y_pred_binary_test = [1 if score < threshold else 0 for score in anomaly_scores_test]
accuracy_train = accuracy_score(Y_train, y_pred_binary_train)
accuracy_test = accuracy_score(Y_test, y_pred_binary_test)
conf_matrix_test = confusion_matrix(Y_test, y_pred_binary_test)
classification_rep_test = classification_report(Y_test, y_pred_binary_test)
print("Training Accuracy:", accuracy_train)
print("Test Accuracy:", accuracy_test)
print("Confusion Matrix (Test Set):\n", conf_matrix_test)
print("Classification Report (Test Set):\n", classification_rep_test)
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.hist(anomaly_scores_train, bins='auto', color='blue', alpha=0.7)
plt.title('Anomaly Scores - Training Set')
plt.xlabel('Anomaly Score')
plt.ylabel('Frequency')
plt.subplot(1, 2, 2)
plt.hist(anomaly_scores_test, bins='auto', color='red', alpha=0.7)
plt.title('Anomaly Scores - Test Set')
plt.xlabel('Anomaly Score')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

```
Training Accuracy: 0.8860833333333333
Test Accuracy: 0.8863333333333333
Confusion Matrix (Test Set):
 [[2657  338]
 [   3    2]]
Classification Report (Test Set):
               precision    recall  f1-score   support

           0       1.00      0.89      0.94      2995
           1       0.01      0.40      0.01         5

    accuracy                           0.89      3000
   macro avg       0.50      0.64      0.48      3000
weighted avg       1.00      0.89      0.94      3000
```



K Nearest Neighbours

```
X_train, X_test, Y_train, Y_test = train_test_split(X_train, Y_train, test_size=0.2, random_state=42)
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, Y_train)
Y_pred = knn_classifier.predict(X_test)
accuracy = accuracy_score(Y_test, Y_pred)
conf_matrix = confusion_matrix(Y_test, Y_pred)
classification_rep = classification_report(Y_test, Y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.9991666666666666
Confusion Matrix:
 [[2398    0]
 [   2    0]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2398
           1       0.00      0.00      0.00         2

    accuracy                           1.00      2400
   macro avg       0.50      0.50      0.50      2400
weighted avg       1.00      1.00      1.00      2400

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
```

```
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, Y_train)
Y_pred = knn_classifier.predict(X_test)
accuracy = accuracy_score(Y_test, Y_pred)
conf_matrix = confusion_matrix(Y_test, Y_pred)
classification_rep = classification_report(Y_test, Y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
plt.figure(figsize=(8,4))
sns.scatterplot(x=X_test.iloc[:, 0], y=X_test.iloc[:, 1], hue=Y_test, palette='viridis', label='True Labels', alpha=0.7)
sns.scatterplot(x=X_test.iloc[:, 0], y=X_test.iloc[:, 1], hue=Y_pred, palette='inferno', marker='X', label='Predicted Labels', alpha=0.5)
plt.title('k-NN Classifier - Test Set Predictions')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
Accuracy: 0.9991666666666666
Confusion Matrix:
 [[2398    0]
 [   2    0]]
Classification Report:
```

AdaBoost Classifer(Ensemble methods)

```
           0       1.00      1.00      1.00      2398
```

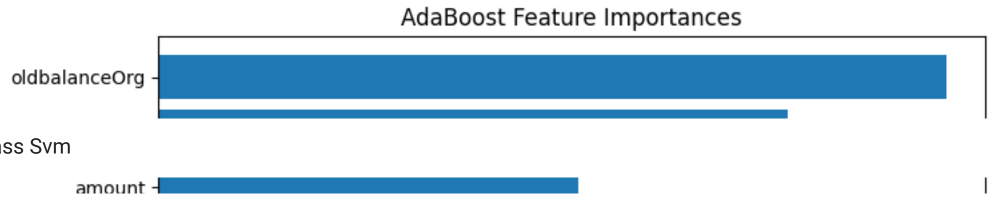
```
model = AdaBoostClassifier(random_state=42)
model.fit(X_train, Y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
conf_matrix = confusion_matrix(Y_test, y_pred)
classification_rep = classification_report(Y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.99875
Confusion Matrix:
 [[2397    1]
 [   2    0]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2398
           1       0.00      0.00      0.00         2

    accuracy                           1.00      2400
   macro avg       0.50      0.50      0.50      2400
weighted avg       1.00      1.00      1.00      2400
```

0 100 200 300 400 500 600 700

```
model = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1), random_state=42)
model.fit(X_train, Y_train)
feature_importances = model.feature_importances_
sorted_indices = np.argsort(feature_importances)
plt.figure(figsize=(8, 4))
plt.barh(range(len(sorted_indices)), feature_importances[sorted_indices], align="center")
plt.yticks(range(len(sorted_indices)), np.array(X_train.columns)[sorted_indices])
plt.title('AdaBoost Feature Importances')
plt.show()
if X_train.shape[1] >= 2:
    x_min, x_max = X_train.iloc[:, 0].min() - 1, X_train.iloc[:, 0].max() + 1
    y_min, y_max = X_train.iloc[:, 1].min() - 1, X_train.iloc[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))

    plt.figure(figsize=(10, 6))
    for tree in model.estimators_:
        Z = tree.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z, alpha=0.1)

    plt.scatter(X_train.iloc[:, 0], X_train.iloc[:, 1], c=Y_train, cmap='viridis', marker='o')
    plt.title('AdaBoost Decision Boundaries')
    plt.xlabel(X_train.columns[0])
    plt.ylabel(X_train.columns[1])
    plt.show()
else:
    print("Not enough features to visualize decision boundaries.")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` ir
  warnings.warn(
```

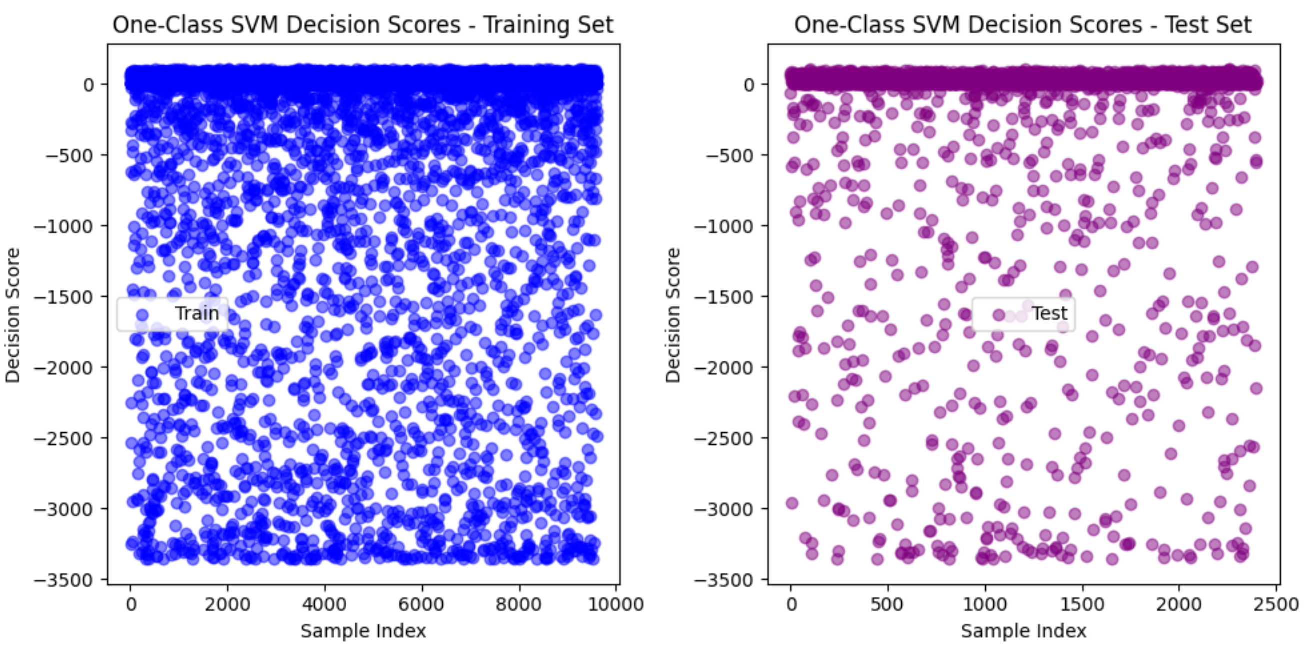


One Class Svm

```
model = OneClassSVM()
model.fit(X_train)
y_pred = model.predict(X_test)
y_pred_binary = [1 if pred == -1 else 0 for pred in y_pred]
accuracy = accuracy_score(Y_test, y_pred_binary)
conf_matrix = confusion_matrix(Y_test, y_pred_binary)
classification_rep = classification_report(Y_test, y_pred_binary)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.5204166666666666
Confusion Matrix:
 [[1248 1150]
 [   1    1]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.52      0.68      2398
           1       0.00      0.50      0.00         2

    accuracy                           0.52      2400
   macro avg       0.50      0.51      0.34      2400
weighted avg       1.00      0.52      0.68      2400
```

```
model = OneClassSVM()
model.fit(X_train)
decision_scores_train = model.decision_function(X_train)
decision_scores_test = model.decision_function(X_test)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(range(len(X_train)), decision_scores_train, c='blue', label='Train', alpha=0.5)
plt.title('One-Class SVM Decision Scores - Training Set')
plt.xlabel('Sample Index')
plt.ylabel('Decision Score')
plt.legend()
plt.subplot(1, 2, 2)
plt.scatter(range(len(X_test)), decision_scores_test, c='purple', label='Test', alpha=0.5)
plt.title('One-Class SVM Decision Scores - Test Set')
plt.xlabel('Sample Index')
plt.ylabel('Decision Score')
plt.legend()
plt.tight_layout()
plt.show()
```



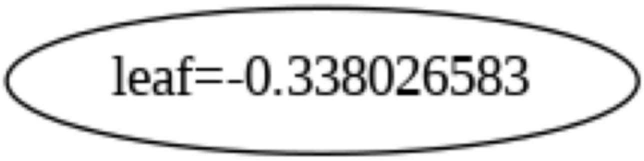
XG Booster

```
model = xgb.XGBClassifier(random_state=42)
model.fit(X_train, Y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
conf_matrix = confusion_matrix(Y_test, y_pred)
classification_rep = classification_report(Y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.9991666666666666
Confusion Matrix:
 [[2398    0]
 [   2    0]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2398
           1       0.00      0.00      0.00         2

    accuracy                           1.00      2400
   macro avg       0.50      0.50      0.50      2400
weighted avg       1.00      1.00      1.00      2400

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
```

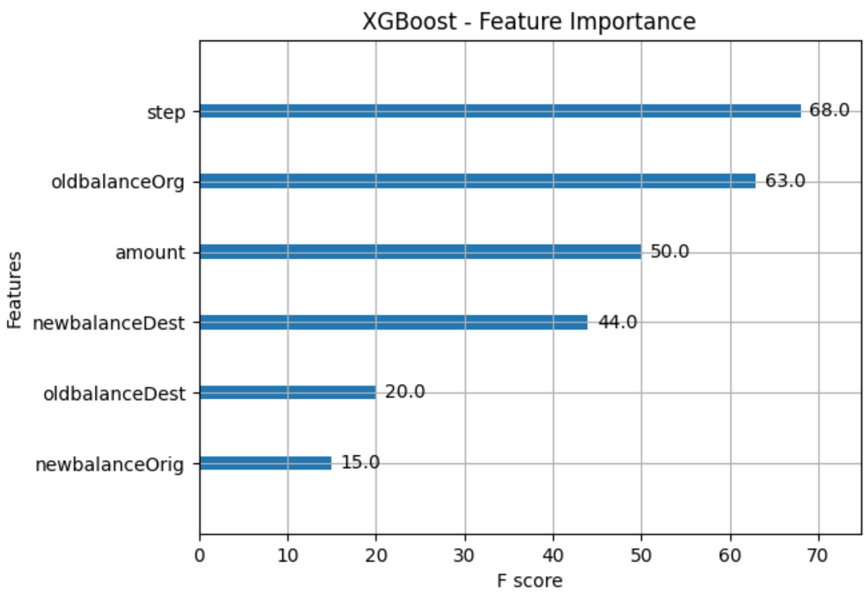
```
model = xgb.XGBClassifier(random_state=42)
model.fit(X_train, Y_train)
booster = model.get_booster()
plt.figure(figsize=(20, 10))
xgb.plot_tree(booster, num_trees=0, rankdir='LR')
plt.show()
```

<Figure size 2000x1000 with 0 Axes>




```
plt.figure(figsize=(10, 6))
plot_importance(model, max_num_features=10)
plt.title('XGBoost - Feature Importance')
plt.show()
```

<Figure size 1000x600 with 0 Axes>



Autoencoder

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
input_dim = X_train.shape[1]
encoding_dim = 10
input_layer = Input(shape=(input_dim,))
encoder_layer = Dense(encoding_dim, activation="relu")(input_layer)
decoder_layer = Dense(input_dim, activation="sigmoid")(encoder_layer)

autoencoder = Model(inputs=input_layer, outputs=decoder_layer)
autoencoder.compile(optimizer="adam", loss="mean_squared_error")
autoencoder.fit(X_train, X_train, epochs=50, batch_size=256, shuffle=True, validation_data=(X_test, X_test))

decoded_data = autoencoder.predict(X_test)

mse = np.mean(np.power(X_test - decoded_data, 2), axis=1)

threshold = np.percentile(mse, 95)

y_pred = np.where(mse > threshold, 1, 0)

accuracy = np.mean(y_test == y_pred)
conf_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
classification_rep = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
   macro avg       0.26      0.45      0.33        20
weighted avg       0.29      0.50      0.37        20
```

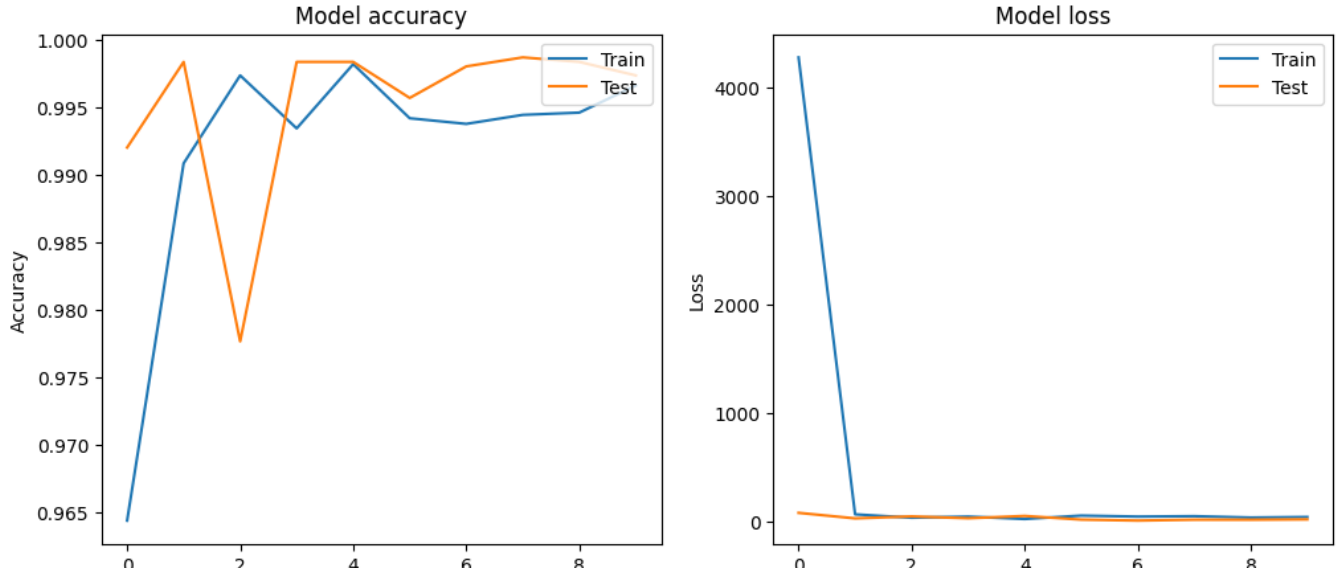
Artificial Neural Network (ANN)

```
model = Sequential()
model.add(Dense(16, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
y_pred_prob = model.predict(X_test)
y_pred = np.round(y_pred_prob)
accuracy = np.mean(y_test == y_pred.flatten())
conf_matrix = pd.crosstab(y_test, y_pred.flatten(), rownames=['Actual'], colnames=['Predicted'])
classification_rep = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Epoch 1/10
3/3 [==============================] - 2s 185ms/step - loss: 0.7070 - accuracy: 0.4750 - val_loss: 0.7582 - val_accuracy: 0.2500
Epoch 2/10
3/3 [==============================] - 0s 81ms/step - loss: 0.6937 - accuracy: 0.5000 - val_loss: 0.7402 - val_accuracy: 0.2500
Epoch 3/10
3/3 [==============================] - 0s 96ms/step - loss: 0.6813 - accuracy: 0.5500 - val_loss: 0.7237 - val_accuracy: 0.4500
Epoch 4/10
3/3 [==============================] - 0s 44ms/step - loss: 0.6708 - accuracy: 0.6125 - val_loss: 0.7086 - val_accuracy: 0.4500
Epoch 5/10
3/3 [==============================] - 0s 77ms/step - loss: 0.6596 - accuracy: 0.6625 - val_loss: 0.6929 - val_accuracy: 0.4500
Epoch 6/10
3/3 [==============================] - 0s 64ms/step - loss: 0.6489 - accuracy: 0.7125 - val_loss: 0.6772 - val_accuracy: 0.5000
Epoch 7/10
3/3 [==============================] - 0s 94ms/step - loss: 0.6390 - accuracy: 0.7625 - val_loss: 0.6619 - val_accuracy: 0.6500
Epoch 8/10
3/3 [==============================] - 0s 83ms/step - loss: 0.6289 - accuracy: 0.8250 - val_loss: 0.6477 - val_accuracy: 0.8000
Epoch 9/10
3/3 [==============================] - 0s 61ms/step - loss: 0.6187 - accuracy: 0.8625 - val_loss: 0.6341 - val_accuracy: 0.8500
Epoch 10/10
3/3 [==============================] - 0s 62ms/step - loss: 0.6094 - accuracy: 0.9625 - val_loss: 0.6207 - val_accuracy: 0.9500
1/1 [==============================] - 0s 326ms/step
Accuracy: 0.95
Confusion Matrix:
 Predicted  0.0  1.0
Actual
0           10    1
1            0    9
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.91      0.95        11
           1       0.90      1.00      0.95         9

    accuracy                           0.95        20
   macro avg       0.95      0.95      0.95        20
weighted avg       0.96      0.95      0.95        20
```

```
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=10, batch_size=32, validation_data=(X_test, Y_test))
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper right')
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```

```
Epoch 1/10
375/375 [==============================] - 3s 4ms/step - loss: 4281.4780 - accuracy: 0.9644 - val_loss: 80.1702 - val_accuracy: 0.99
Epoch 2/10
375/375 [==============================] - 2s 4ms/step - loss: 66.3509 - accuracy: 0.9908 - val_loss: 29.0905 - val_accuracy: 0.9983
Epoch 3/10
375/375 [==============================] - 2s 4ms/step - loss: 35.3238 - accuracy: 0.9973 - val_loss: 47.4948 - val_accuracy: 0.9977
Epoch 4/10
375/375 [==============================] - 4s 10ms/step - loss: 45.7150 - accuracy: 0.9934 - val_loss: 30.2276 - val_accuracy: 0.998
Epoch 5/10
375/375 [==============================] - 4s 10ms/step - loss: 23.2478 - accuracy: 0.9982 - val_loss: 51.5190 - val_accuracy: 0.998
Epoch 6/10
375/375 [==============================] - 3s 8ms/step - loss: 54.9783 - accuracy: 0.9942 - val_loss: 17.8174 - val_accuracy: 0.9957
Epoch 7/10
375/375 [==============================] - 2s 6ms/step - loss: 45.7325 - accuracy: 0.9937 - val_loss: 8.8456 - val_accuracy: 0.9980
Epoch 8/10
375/375 [==============================] - 2s 5ms/step - loss: 49.4173 - accuracy: 0.9944 - val_loss: 16.9957 - val_accuracy: 0.9987
Epoch 9/10
375/375 [==============================] - 2s 4ms/step - loss: 36.9019 - accuracy: 0.9946 - val_loss: 16.8499 - val_accuracy: 0.9983
Epoch 10/10
375/375 [==============================] - 2s 4ms/step - loss: 41.7464 - accuracy: 0.9966 - val_loss: 19.8583 - val_accuracy: 0.9973
```



Model Comparison

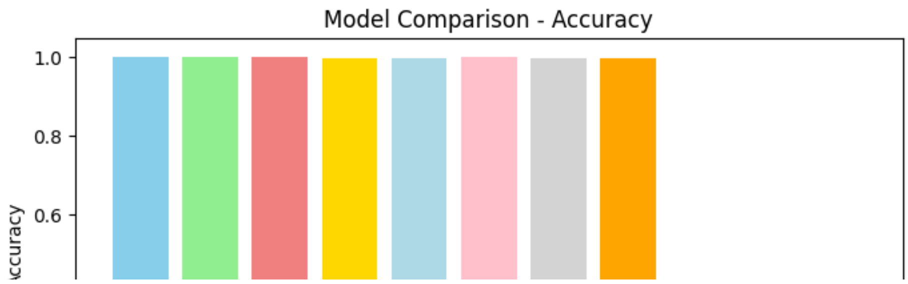
Comapraring all the models and finding which model gives the best accuracy

```
X = random_data.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)
y = random_data['isFraud']
def create_neural_network():
    model = Sequential()
    model.add(Dense(units=64, activation='relu', input_dim=X.shape[1]))
    model.add(Dense(units=1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
models = {
    'LogisticRegression': LogisticRegression(random_state=42),
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'AdaBoost': AdaBoostClassifier(random_state=42),
    'MLP': MLPClassifier(random_state=42),
    'XGBClassifier': xgb.XGBClassifier(random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=3),
    'SVM': SVC(random_state=42),
    'OneClassSVM': OneClassSVM(),
    'Autoencoder': StandardScaler(),
    'Isolation Forest': IsolationForest(random_state=42),
    'ANN': create_neural_network()
}
scorer = make_scorer(accuracy_score)
results = {}
for model_name, model in models.items():
    try:
        scores = cross_val_score(model, X, y, cv=5, scoring=scorer, n_jobs=-1)
        mean_score = scores.mean()
        results[model_name] = mean_score
    except Exception as e:
        continue
best_model = max(results, key=results.get)
best_accuracy = results[best_model]
print("Model Comparison:")
for model_name, result in results.items():
    print(f"{model_name}: {result}")
print(f"\nBest Model: {best_model} with Accuracy: {best_accuracy}")
```

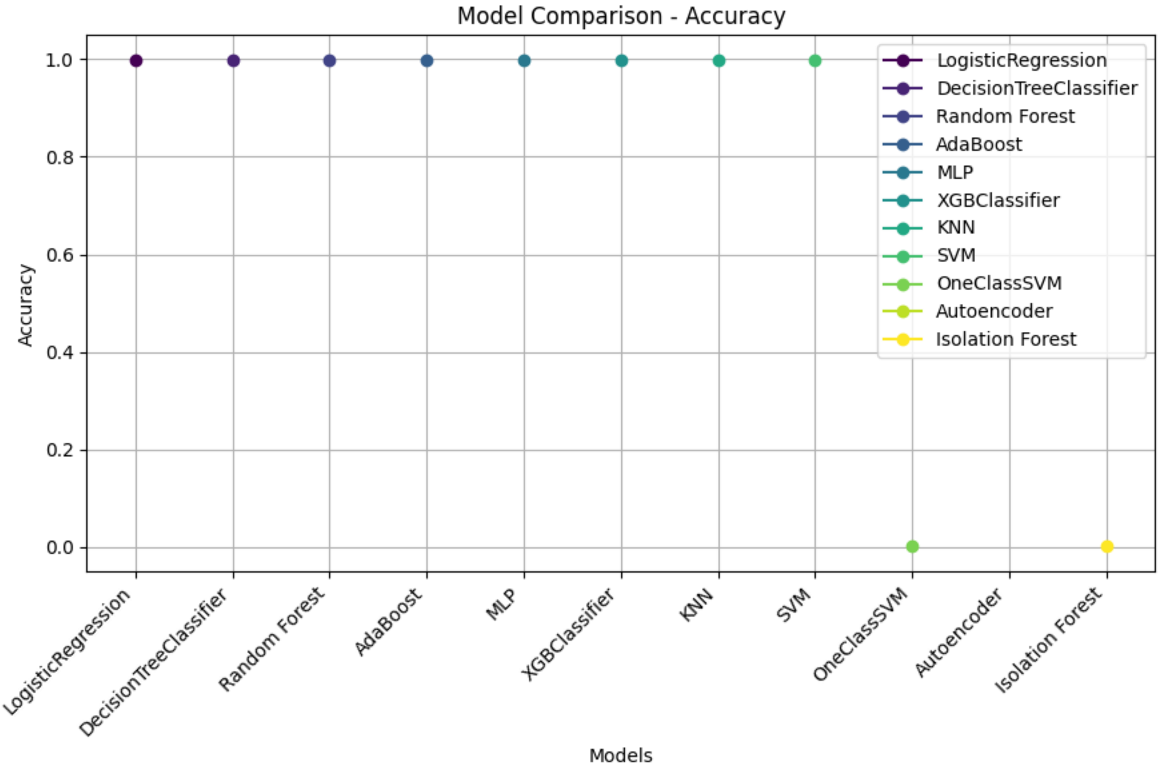
```
Model Comparison:
LogisticRegression: 0.9996
DecisionTreeClassifier: 0.9992666666666666
Random Forest: 0.9991999999999999
AdaBoost: 0.999
MLP: 0.9983333333333334
XGBClassifier: 0.9994666666666667
KNN: 0.9989333333333332
SVM: 0.9986666666666666
OneClassSVM: 0.0007333333333333333
Autoencoder: nan
Isolation Forest: 0.0008666666666666666

Best Model: LogisticRegression with Accuracy: 0.9996
```

Comparing the accuracy of different models using a bar chart. Each model is represented by a colored bar, and the chart provides a quick overview of their relative performance. The varying colors help distinguish between models, offering a concise summary of their accuracy in a visually appealing manner.

```
plt.figure(figsize=(8, 4))
colors = ['skyblue', 'lightgreen', 'lightcoral', 'gold', 'lightblue', 'pink', 'lightgray', 'orange', 'purple', 'brown', 'black', 'cyan']
plt.bar(results.keys(), results.values(), color=colors)
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Comparison - Accuracy')
plt.xticks(rotation=45, ha='right')
plt.show()
```

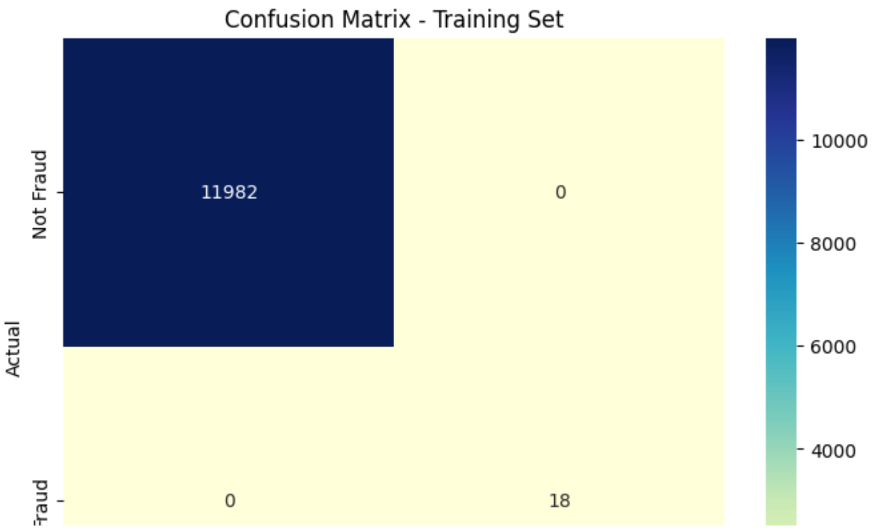



```
plt.figure(figsize=(9, 6))
colors = plt.cm.viridis(np.linspace(0, 1, len(results)))
for i, (model_name, result) in enumerate(results.items()):
    plt.plot(i, result, marker='o', color=colors[i], label=model_name)
plt.title('Model Comparison - Accuracy')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.xticks(np.arange(len(results)), list(results.keys()), rotation=45, ha='right')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



confusion matrix visualizes the model's performance on the training set, showing counts of true positive, true negative, false positive, and false negative predictions

```
classifier = RandomForestClassifier(random_state=42)
classifier.fit(X_train, Y_train)
Y_pred_train = classifier.predict(X_train)
cm_train = confusion_matrix(Y_train, Y_pred_train)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_train, annot=True, fmt='d', cmap='YlGnBu',
            xticklabels=['Not Fraud', 'Fraud'], yticklabels=['Not Fraud', 'Fraud'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Training Set')
plt.show()
```

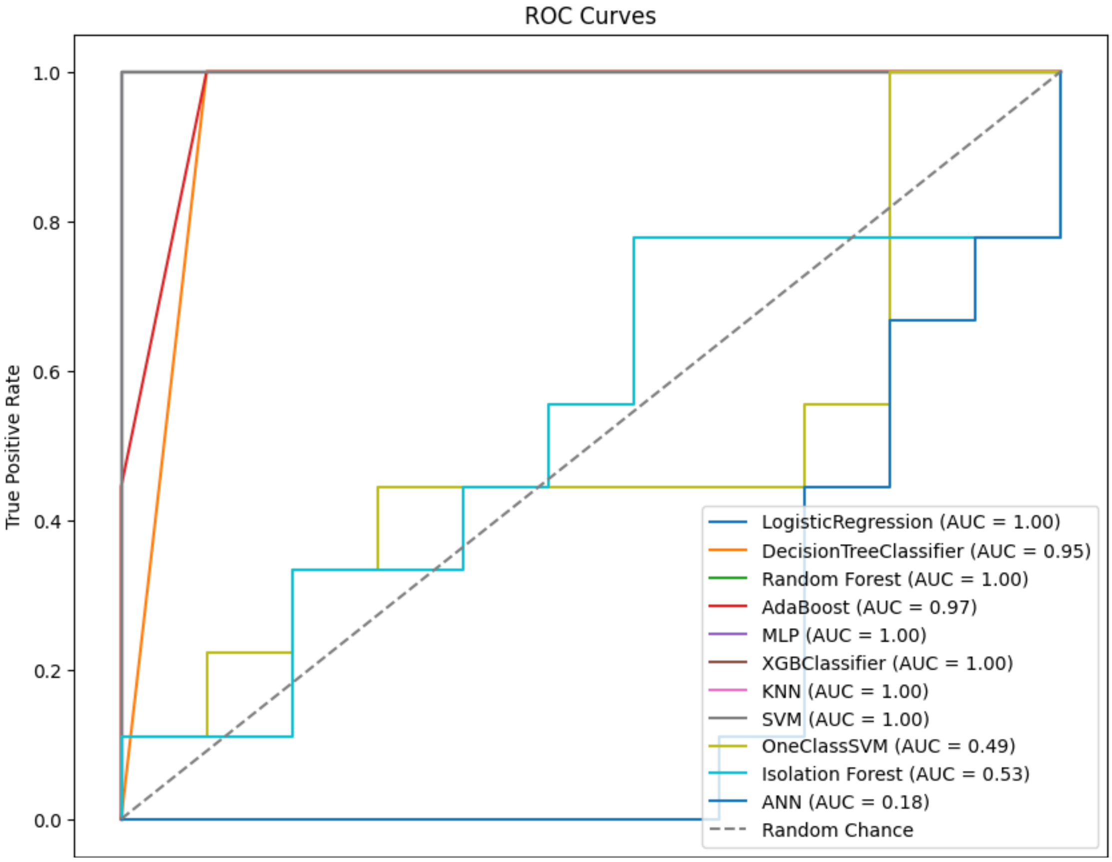


ROC Curve

This code compares Receiver Operating Characteristic (ROC) curves for various classifiers, assessing their binary classification performance.The plot helps identify classifiers with higher AUC values, indicating better trade-offs between true positive and false positive rates.


```
classifiers = {
    'LogisticRegression': LogisticRegression(random_state=42),
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'AdaBoost': AdaBoostClassifier(random_state=42),
    'MLP': MLPClassifier(random_state=42),
    'XGBClassifier': xgb.XGBClassifier(random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=3),
    'SVM': SVC(random_state=42, probability=True),
    'OneClassSVM': OneClassSVM(),
    'Isolation Forest': IsolationForest(random_state=42),
    'ANN': Sequential()
}

plt.figure(figsize=(10, 8))
for name, classifier in classifiers.items():
    if name == 'Autoencoder':

        continue

    if name == 'ANN':

        classifier.add(Dense(1, activation='sigmoid'))
        classifier.compile(optimizer='adam', loss='binary_crossentropy')
        y_pred_proba = classifier.predict(X_test)
    elif isinstance(classifier, (OneClassSVM, IsolationForest)):

        classifier.fit(X_train)
        y_pred_proba = -classifier.decision_function(X_test)
    else:

        classifier.fit(X_train, y_train)
        if hasattr(classifier, 'predict_proba'):
            y_pred_proba = classifier.predict_proba(X_test)[:, 1]
        else:
            y_pred_proba = classifier.decision_function(X_test)


    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Chance')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves    ')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimi:
  warnings.warn(
1/1 [==============================] - 0s 119ms/step
```

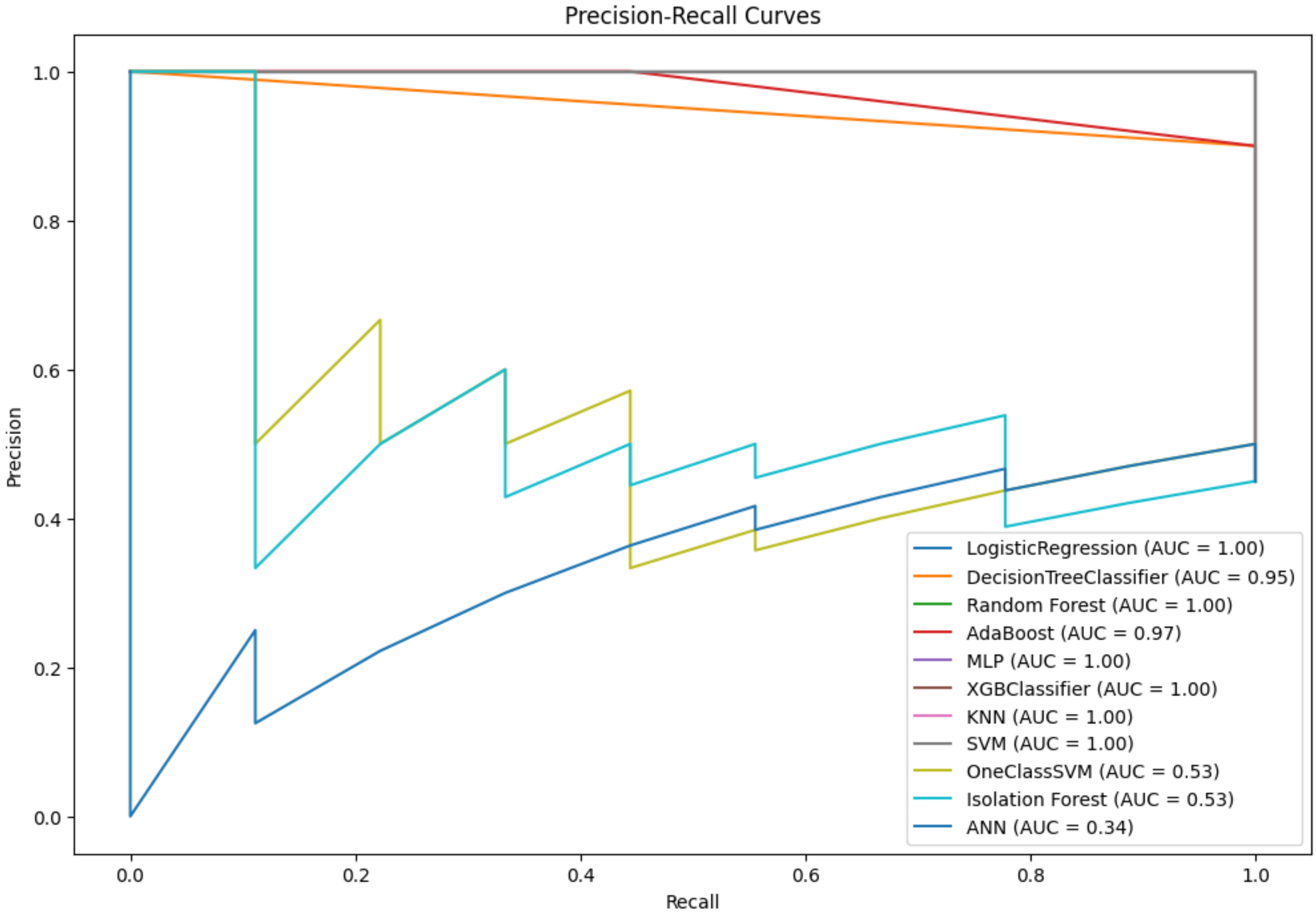


Precision Recall Curve

This code compares the Precision-Recall curves for various classifiers, evaluating their performance on a binary classification task.The plot helps identify classifiers with better trade-offs between precision and recall, crucial for tasks with imbalanced classes.

```
classifiers = {
    'LogisticRegression': LogisticRegression(random_state=42),
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'AdaBoost': AdaBoostClassifier(random_state=42),
    'MLP': MLPClassifier(random_state=42),
    'XGBClassifier': xgb.XGBClassifier(random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=3),
    'SVM': SVC(random_state=42, probability=True),
    'OneClassSVM': OneClassSVM(),
    'Isolation Forest': IsolationForest(random_state=42),
    'ANN': Sequential()
}
plt.figure(figsize=(12, 8))
for name, classifier in classifiers.items():
    if name == 'Autoencoder':
         continue
    if name == 'ANN':
        classifier.add(Dense(1, activation='sigmoid'))
        classifier.compile(optimizer='adam', loss='binary_crossentropy')
        y_pred_proba = classifier.predict(X_test)
    elif isinstance(classifier, (OneClassSVM, IsolationForest)):
        classifier.fit(X_train)
        y_pred_proba = -classifier.decision_function(X_test)
    else:
        classifier.fit(X_train, y_train)
        if hasattr(classifier, 'predict_proba'):
            y_pred_proba = classifier.predict_proba(X_test)[:, 1]
        else:
            y_pred_proba = classifier.decision_function(X_test)
    precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
    pr_auc = auc(recall, precision)
    plt.plot(recall, precision, label=f'{name} (AUC = {pr_auc:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curves')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimi:
  warnings.warn(
1/1 [==============================] - 0s 151ms/step
```



This code calculates and visualizes the permutation importance of features in a machine learning model. Permutation importance measures the impact of shuffling each feature's values on the model's performance

```
perm_importance = permutation_importance(model, X_test, y_test, n_repeats=30, random_state=42)
indices = np.argsort(perm_importance.importances_mean)[::-1]
plt.figure(figsize=(6, 3))
plt.bar(range(X_test.shape[1]), perm_importance.importances_mean[indices])
plt.xticks(range(X_test.shape[1]), X_test.columns[indices], rotation=45, ha='right')
plt.xlabel('Feature')
plt.ylabel('Permutation Importance')
plt.title('Permutation Importance')
plt.show()
```

Permutation Importance