Import all necessary Libraries

In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

import pickle
import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, accuracy_score, confusion_matrix, classi
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor, Ba
from sklearn.ensemble import StackingClassifier
from tqdm.notebook import tqdm
```

Load the Dataset

In [2]:
```
# Function to load data from a single .pkl file
def load_pkl(file_path):
    with open(file_path, 'rb') as file:
        data = pickle.load(file)
    return pd.DataFrame(data)

# Directory containing the .pkl files
directory = 'dataset/data'  # Adjust this path based on your setup

# List to hold all DataFrames
all_dataframes = []

# Iterate through the .pkl files in the directory
for filename in os.listdir(directory):
    if filename.endswith('.pkl'):
        file_path = os.path.join(directory, filename)
        df = load_pkl(file_path)
        all_dataframes.append(df)


# Combine all DataFrames into one
combined_df = pd.concat(all_dataframes, ignore_index=True)
```

EDA and Preprocessing

In [3]:
```
# Display the combined DataFrame
combined_df.head(5)
```

Out[3]:

	TRANSACTION_ID	TX_DATETIME	CUSTOMER_ID	TERMINAL_ID	TX_AMOUNT	TX_TIME_SEC
0	0	2018-04-01 00:00:31	596	3156	57.16	
1	1	2018-04-01 00:02:10	4961	3412	81.51	
2	2	2018-04-01 00:07:56	2	1365	146.00	
3	3	2018-04-01 00:09:29	4128	8737	64.49	
4	4	2018-04-01 00:10:34	927	9906	50.99	

In [4]:
```
combined_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1754155 entries, 0 to 1754154
Data columns (total 9 columns):
 #   Column             Dtype
---  ------             -----
 0   TRANSACTION_ID     int64
 1   TX_DATETIME        datetime64[ns]
 2   CUSTOMER_ID        object
 3   TERMINAL_ID        object
 4   TX_AMOUNT          float64
 5   TX_TIME_SECONDS    object
 6   TX_TIME_DAYS       object
 7   TX_FRAUD           int64
 8   TX_FRAUD_SCENARIO  int64
dtypes: datetime64[ns](1), float64(1), int64(3), object(4)
memory usage: 120.4+ MB
```

- This output shows that we have around 1754155 entries with 9 Columns
- We have 4 Object data type, 3 Interger data type, 1 Float data type and 1 Date time data type

In [5]:
```
combined_df.describe()
```

Out[5]:

	TRANSACTION_ID	TX_AMOUNT	TX_FRAUD	TX_FRAUD_SCENARIO
count	1.754155e+06	1.754155e+06	1.754155e+06	1.754155e+06
mean	8.770770e+05	5.363230e+01	8.369272e-03	1.882388e-02
std	5.063811e+05	4.232649e+01	9.110012e-02	2.113263e-01
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	4.385385e+05	2.101000e+01	0.000000e+00	0.000000e+00
50%	8.770770e+05	4.464000e+01	0.000000e+00	0.000000e+00
75%	1.315616e+06	7.695000e+01	0.000000e+00	0.000000e+00
max	1.754154e+06	2.628000e+03	1.000000e+00	3.000000e+00

In [6]:
```
combined_df.describe(include='O')
```

Out[6]:

	CUSTOMER_ID	TERMINAL_ID	TX_TIME_SECONDS	TX_TIME_DAYS
count	1754155	1754155	1754155	1754155
unique	4990	10000	1635076	183
top	382	4018	7992619	90
freq	767	376	6	9789

Inference

- The dataset has 1754155 rows and 9 columns
- There is missing values in some columns, so we have to treat with appropriate method

Checking the shape

In [7]:
```
combined_df.shape
```

Out[7]:
```
(1754155, 9)
```

In [8]:
```
combined_df.ndim
```

Out[8]:
```
2
```

Data cleansing and Exploratory data analysis

Check if there are any duplicate records in the dataset? If any, drop them.

In [9]:
```
combined_df.duplicated().sum()
```

Out[9]:
```
0
```

There is no duplicated record in this dataset

Check for Null values and impute them

In [10]:
```
combined_df.isna().sum()
```

Out[10]:
```
TRANSACTION_ID       0
TX_DATETIME          0
CUSTOMER_ID          0
TERMINAL_ID          0
TX_AMOUNT            0
TX_TIME_SECONDS      0
TX_TIME_DAYS         0
TX_FRAUD             0
TX_FRAUD_SCENARIO    0
dtype: int64
```

Inference:

- there is no null values in the dataset

Feature Engineering

Let's create features based on the scenarios described:

Amount-based feature: Flag transactions with amounts greater than 220.

Terminal-based feature: Identify terminals with fraudulent transactions.

Customer spending patterns: Track changes in spending behavior.

In [11]:
```
# Example: Calculate the average spending per customer
customer_avg_spending = combined_df.groupby('CUSTOMER_ID')['TX_AMOUNT'].mean()
df = combined_df.join(customer_avg_spending, on='CUSTOMER_ID')

# Flag transactions significantly higher than the average spending
df['customer_spending_fraud'] = combined_df['TX_AMOUNT'] > df['customer_avg_sp
```

In [12]:
```
# Create a feature indicating high transaction amounts
df['amount_fraud'] = combined_df['TX_AMOUNT'] > 220
```

In [13]:
```
# Assume we have a list of fraudulent terminals (this should be dynamically ge
fraudulent_terminals = np.random.choice(df['TERMINAL_ID'].unique(), 2, replace

# Flag transactions from fraudulent terminals
df['terminal_fraud'] = combined_df['TERMINAL_ID'].isin(fraudulent_terminals)
```

In [14]:
```
df['predicted_fraud'] = df[['amount_fraud', 'terminal_fraud', 'customer_spendi
```

Seperate the target and independent features and split the data into train and test

In [15]:
```
# Features and target variable
X = df[['amount_fraud', 'terminal_fraud', 'customer_spending_fraud']]
y = df['TX_FRAUD']

# Split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, randc
print('Shape of X_train:',X_train.shape)
print('Shape of X_test:',X_test.shape)
print('Shape of y_train:',y_train.shape)
print('Shape of y_test:',y_test.shape)
```

```
Shape of X_train: (1227908, 3)
Shape of X_test: (526247, 3)
Shape of y_train: (1227908,)
Shape of y_test: (526247,)
```

Build the Classification model to predict the Fraud Transaction and save the model using pickle

In [16]:
```
def fit_n_predict(model, X_train, X_test, y_train, y_test):
    # Fit the model with train data
    model.fit(X_train, y_train)

    # Making prediction on test set
    y_pred= model.predict(X_test)

    # Calculating the accuracy score
    accuracy = accuracy_score(y_test, y_pred)

    return accuracy
```

In [17]:
```
# Intializing the models
lr = LogisticRegression()
nb = GaussianNB()
knn = KNeighborsClassifier()
dt = DecisionTreeClassifier()
rf = RandomForestClassifier(n_estimators=30, random_state=10)
adb = AdaBoostClassifier()
gb = GradientBoostingClassifier()
```

In [18]:
```
result = pd.DataFrame(columns = ['Accuracy'])

for model,model_name in zip([rf],
                            ['Random Forest']):

    result.loc[model_name] = fit_n_predict(model,X_train,X_test,y_train,y_test


result
```

Out[18]:

	Accuracy
Random Forest	0.99387

Predict "Outcome" using Random Forest Classifier

In [19]:
```
y_train_pred_rf = rf.predict(X_train)
y_test_pred_rf = rf.predict(X_test)


accuracy_train = accuracy_score(y_train,y_train_pred_rf)
accuracy_test = accuracy_score(y_test,y_test_pred_rf)

print('Accuracy train:', accuracy_train)
print('Accuracy test:', accuracy_test)
```

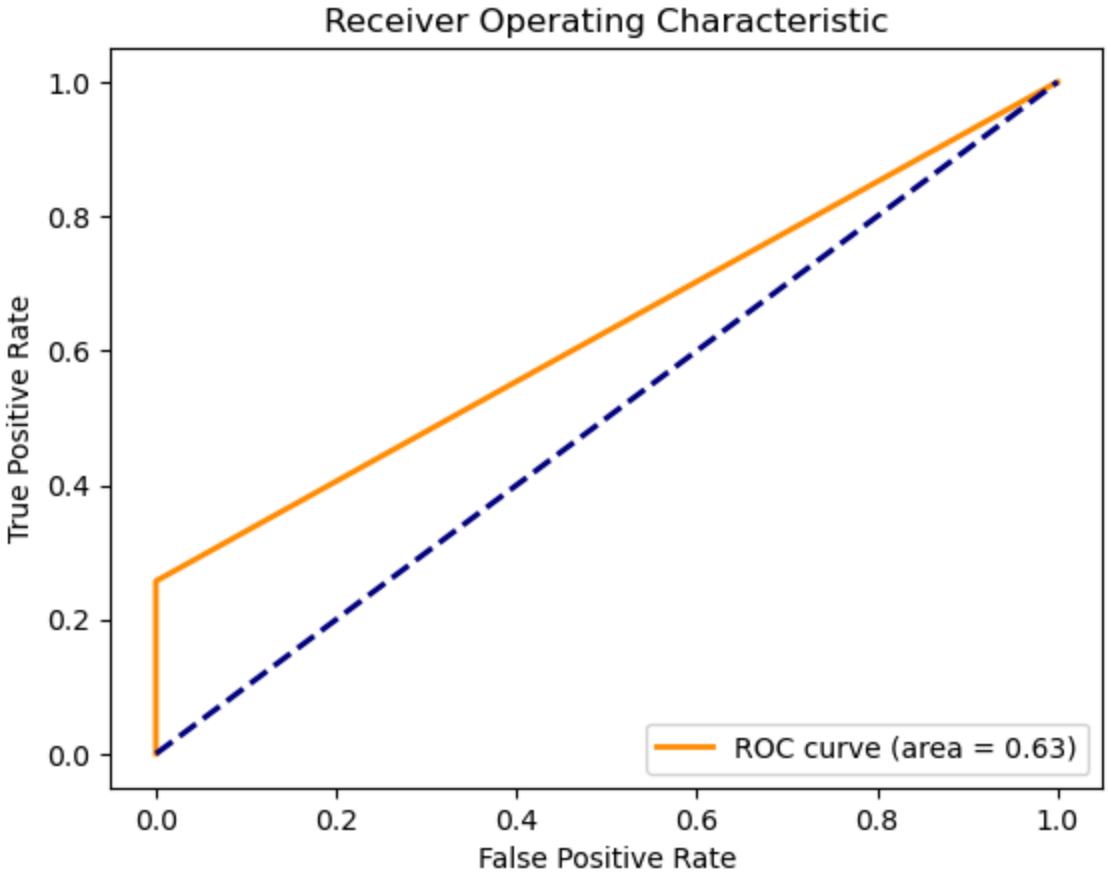
```
Accuracy train: 0.993761747622786
Accuracy test: 0.9938697987827009
```

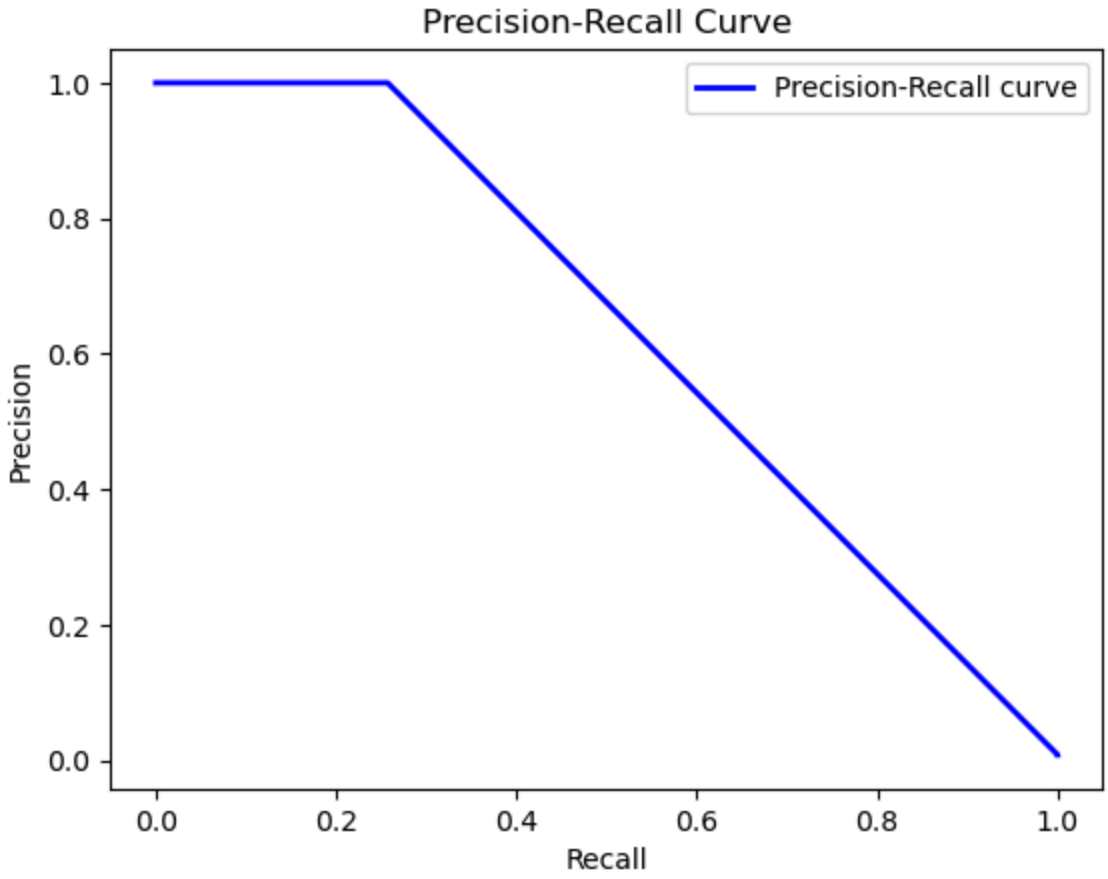
In [20]:
```
## Compute precision, recall and F1-score
report = classification_report(y_test, y_test_pred_rf, target_names=['Legit',
print(report)
```

```
              precision    recall  f1-score   support

       Legit       0.99      1.00      1.00    521905
       Fraud       1.00      0.26      0.41      4342

    accuracy                           0.99    526247
   macro avg       1.00      0.63      0.70    526247
weighted avg       0.99      0.99      0.99    526247
```

In [21]:
```
# Predictions
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Legit', 'Frau
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# ROC Curve and AUC
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_au
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
plt.figure()
plt.plot(recall, precision, lw=2, color='b', label='Precision-Recall curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='upper right')
plt.show()
```


Legit
Fraud
Actual
521905
0
3226
1116
Legit
Fraud
Predicted
500000
400000
300000
200000
100000
0


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.63)



In [22]:
```
combined_df.to_csv('output.csv', index=False)
```

In [23]:
```
import pickle
# Saving model to disk
pickle.dump(rf, open('model.pkl','wb'))
```