CREDIT CARD FRAUD DETECTION

Build a model to detect fraudulent credit card transactions. Use a dataset containing information about credit card transactions, and experiment with algorithms like Logistic Regression, Decision Trees, or Random Forests to classify transactions as fraudulent or legitimate.

Dataset Link

About the Dataset:

This is a simulated credit card transaction dataset containing legitimate and fraud transactions from the duration 1st Jan 2019 - 31st Dec 2020. It covers credit cards of 1000 customers doing transactions with a pool of 800 merchants.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# use a Kaggle dataset in your Google Colab notebook
# need to install the Kaggle API
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.14)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.7.4)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.4)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.7)
```

Get Your Kaggle API Credentials

1. Go to your Kaggle account settings: https://www.kaggle.com/account
2. Scroll down to the "API" section.
3. Click on "Create New API Token." This will download a kaggle.json file containing your API credentials.

```
# to upload the kaggle.json file:
from google.colab import files
files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"anushkaumbre9503","key":"3bb0bda643187ad387f45012eea2613d"}'}
```

```
# Move the kaggle.json file to the correct directory and set the permissions.
# This command creates a directory named .kaggle in the user's home directory if it does not already exist.
# The -p flag ensures that no error is thrown if the directory already exists.
!mkdir -p ~/.kaggle
# This command copies a file named kaggle.json (which typically contains API credentials) into the .kaggle directory that was created ir
!cp kaggle.json ~/.kaggle/
# This command changes the permissions of the kaggle.json file so that only the file owner has read and write access to it.
# This is important for security, as it prevents unauthorized access to the API credentials stored in the file.
!chmod 600 ~/.kaggle/kaggle.json
```

```
# Download the dataset
# Use the Kaggle API to download the dataset.
!kaggle datasets download -d kartik2112/fraud-detection
```

```
Dataset URL: https://www.kaggle.com/datasets/kartik2112/fraud-detection
License(s): CC0-1.0
Downloading fraud-detection.zip to /content
 97% 195M/202M [00:02<00:00, 84.0MB/s]
100% 202M/202M [00:02<00:00, 93.2MB/s]
```

```
# Unzip the downloaded dataset.
!unzip fraud-detection.zip
```

```
Archive:  fraud-detection.zip
  inflating: fraudTest.csv
  inflating: fraudTrain.csv
```

```
# Load the datasets
train_data = pd.read_csv('fraudTrain.csv')
test_data = pd.read_csv('fraudTest.csv')
```

DATA PREPROCESSING

```
# Display basic information about the datasets
print("Train Data Info:")
print(train_data.info())

print("\nTest Data Info:")
print(test_data.info())
```

```
Train Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   Unnamed: 0             1296675 non-null  int64
 1   trans_date_trans_time  1296675 non-null  object
 2   cc_num                 1296675 non-null  int64
 3   merchant               1296675 non-null  object
 4   category               1296675 non-null  object
 5   amt                    1296675 non-null  float64
 6   first                  1296675 non-null  object
 7   last                   1296675 non-null  object
 8   gender                 1296675 non-null  object
 9   street                 1296675 non-null  object
 10  city                   1296675 non-null  object
 11  state                  1296675 non-null  object
 12  zip                    1296675 non-null  int64
 13  lat                    1296675 non-null  float64
 14  long                   1296675 non-null  float64
 15  city_pop               1296675 non-null  int64
 16  job                    1296675 non-null  object
 17  dob                    1296675 non-null  object
 18  trans_num              1296675 non-null  object
 19  unix_time              1296675 non-null  int64
 20  merch_lat              1296675 non-null  float64
 21  merch_long             1296675 non-null  float64
 22  is_fraud               1296675 non-null  int64
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB
None

Test Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 555719 entries, 0 to 555718
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Unnamed: 0             555719 non-null  int64
 1   trans_date_trans_time  555719 non-null  object
 2   cc_num                 555719 non-null  int64
 3   merchant               555719 non-null  object
 4   category               555719 non-null  object
 5   amt                    555719 non-null  float64
 6   first                  555719 non-null  object
 7   last                   555719 non-null  object
 8   gender                 555719 non-null  object
 9   street                 555719 non-null  object
 10  city                   555719 non-null  object
 11  state                  555719 non-null  object
 12  zip                    555719 non-null  int64
 13  lat                    555719 non-null  float64
 14  long                   555719 non-null  float64
 15  city_pop               555719 non-null  int64
 16  job                    555719 non-null  object
 17  dob                    555719 non-null  object
 18  trans_num              555719 non-null  object
```

```
# Display the first few rows of the train dataset
print("\nTrain Data Sample:")
print(train_data.head())
```

```
Train Data Sample:
   Unnamed: 0 trans_date_trans_time            cc_num  \
0           0   2019-01-01 00:00:18  2703186189652095
1           1   2019-01-01 00:00:44      630423337322
```

```
2           2   2019-01-01 00:00:51    38859492057661
3           3   2019-01-01 00:01:16  3534093764340240
4           4   2019-01-01 00:03:06   375534208663984

                            merchant       category     amt      first  \
0          fraud_Rippin, Kub and Mann       misc_net    4.97   Jennifer
1     fraud_Heller, Gutmann and Zieme    grocery_pos  107.23  Stephanie
2                fraud_Lind-Buckridge  entertainment  220.11     Edward
3  fraud_Kutch, Hermiston and Farrell  gas_transport   45.00     Jeremy
4                 fraud_Keeling-Crist       misc_pos   41.96      Tyler

      last gender                        street  ...      lat      long  \
0    Banks      F                561 Perry Cove  ...  36.0788  -81.1781
1     Gill      F  43039 Riley Greens Suite 393  ...  48.8878 -118.2105
2  Sanchez      M      594 White Dale Suite 530  ...  42.1808 -112.2620
3    White      M   9443 Cynthia Court Apt. 038  ...  46.2306 -112.1138
4   Garcia      M              408 Bradley Rest  ...  38.4207  -79.4629

   city_pop                                job         dob  \
0      3495          Psychologist, counselling  1988-03-09
1       149  Special educational needs teacher  1978-06-21
2      4154        Nature conservation officer  1962-01-19
3      1939                    Patent attorney  1967-01-12
4        99     Dance movement psychotherapist  1986-03-28

                          trans_num   unix_time  merch_lat  merch_long  \
0  0b242abb623afc578575680df30655b9  1325376018  36.011293  -82.048315
1  1f76529f8574734946361c461b024d99  1325376044  49.159047 -118.186462
2  a1a22d70485983eac12b5b88dad1cf95  1325376051  43.150704 -112.154481
3  6b849c168bdad6f867558c3793159a81  1325376076  47.034331 -112.561071
4  a41d7549acf90789359a9aa5346dcb46  1325376186  38.674999  -78.632459

   is_fraud
0         0
1         0
2         0
3         0
4         0

[5 rows x 23 columns]
```

```
# Display the first few rows of the test dataset
print("\nTest Data Sample:")
print(test_data.head())
```

```
Test Data Sample:
   Unnamed: 0 trans_date_trans_time            cc_num  \
0           0   2020-06-21 12:14:25  2291163933867244
1           1   2020-06-21 12:14:33  3573030041201292
2           2   2020-06-21 12:14:53  3598215285024754
3           3   2020-06-21 12:15:15  3591919803438423
4           4   2020-06-21 12:15:17  3526826139003047

                              merchant        category    amt   first  \
0                fraud_Kirlin and Sons   personal_care   2.86    Jeff
1                 fraud_Sporer-Keebler   personal_care  29.84  Joanne
2  fraud_Swaniawski, Nitzsche and Welch  health_fitness  41.28  Ashley
3                    fraud_Haley Group        misc_pos  60.05   Brian
4                fraud_Johnston-Casper          travel   3.19  Nathan

       last gender                       street  ...      lat      long  \
0   Elliott      M            351 Darlene Green  ...  33.9659  -80.9355
1  Williams      F             3638 Marsh Union  ...  40.3207 -110.4360
2     Lopez      F         9333 Valentine Point  ...  40.6729  -73.5365
3  Williams      M  32941 Krystal Mill Apt. 552  ...  28.5697  -80.8191
4    Massey      M     5783 Evan Roads Apt. 465  ...  44.2529  -85.0170

   city_pop                      job         dob  \
0    333497      Mechanical engineer  1968-03-19
1       302  Sales professional, IT  1990-01-17
2     34496       Librarian, public  1970-10-21
3     54767            Set designer  1987-07-25
4      1126      Furniture designer  1955-07-06

                          trans_num   unix_time  merch_lat  merch_long  \
0  2da90c7d74bd46a0caf3777415b3ebd3  1371816865  33.986391  -81.200714
1  324cc204407e99f51b0d6ca0055005e7  1371816873  39.450498 -109.960431
2  c81755dbbbea9d5c77f094348a7579be  1371816893  40.495810  -74.196111
3  2159175b9efe66dc301f149d3d5abf8c  1371816915  28.812398  -80.883061
4  57ff021bd3f328f8738bb535c302a31b  1371816917  44.959148  -85.884734

   is_fraud
0         0
1         0
2         0
3         0
4         0
```

```
[5 rows x 23 columns]
```

```
# Check for missing values in both datasets
print("\nMissing Values in Train Data:")
print(train_data.isnull().sum())
print("\nMissing Values in Test Data:")
print(test_data.isnull().sum())
```

```
Missing Values in Train Data:
Unnamed: 0               0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64

Missing Values in Test Data:
Unnamed: 0               0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

```
# Check for missing values in both datasets
print("\nMissing Values in Train Data:")
print(train_data.isnull().sum())
print("\nMissing Values in Test Data:")
print(test_data.isnull().sum())
```

```
Missing Values in Train Data:
Unnamed: 0               0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
```

```
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64

Missing Values in Test Data:
Unnamed: 0               0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

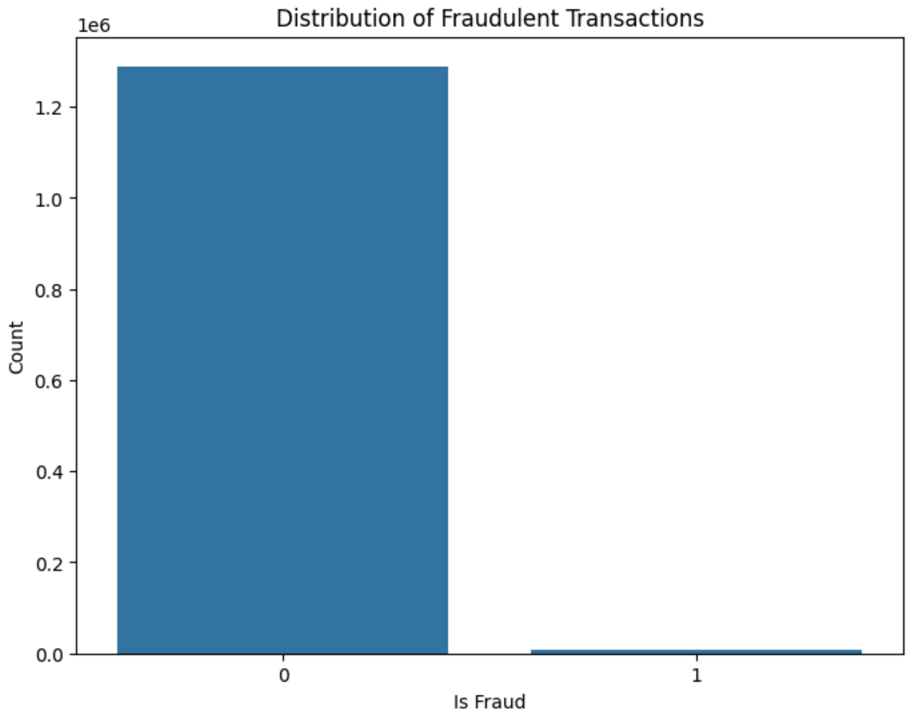
```
# Removing rows with missing values
# --------------------------------
# because it's just single row in each set,
# that's why there will no huge data loss.
train_data.dropna(inplace=True)
test_data.dropna(inplace=True)
```

EXLPORATORY DATA ANALYSIS

```
# Display summary statistics of the train dataset
print("\nTrain Data Summary Statistics:")
print(train_data.describe())
```

```
Train Data Summary Statistics:
         Unnamed: 0        cc_num           amt           zip           lat  \
count  1.296675e+06  1.296675e+06  1.296675e+06  1.296675e+06  1.296675e+06
mean   6.483370e+05  4.171920e+17  7.035104e+01  4.880067e+04  3.853762e+01
std    3.743180e+05  1.308806e+18  1.603160e+02  2.689322e+04  5.075808e+00
min    0.000000e+00  6.041621e+10  1.000000e+00  1.257000e+03  2.002710e+01
25%    3.241685e+05  1.800429e+14  9.650000e+00  2.623700e+04  3.462050e+01
50%    6.483370e+05  3.521417e+15  4.752000e+01  4.817400e+04  3.935430e+01
75%    9.725055e+05  4.642255e+15  8.314000e+01  7.204200e+04  4.194040e+01
max    1.296674e+06  4.992346e+18  2.894890e+04  9.978300e+04  6.669330e+01

               long      city_pop     unix_time     merch_lat    merch_long  \
count  1.296675e+06  1.296675e+06  1.296675e+06  1.296675e+06  1.296675e+06
mean  -9.022634e+01  8.882444e+04  1.349244e+09  3.853734e+01 -9.022646e+01
std    1.375908e+01  3.019564e+05  1.284128e+07  5.109788e+00  1.377109e+01
min   -1.656723e+02  2.300000e+01  1.325376e+09  1.902779e+01 -1.666712e+02
25%   -9.679800e+01  7.430000e+02  1.338751e+09  3.473357e+01 -9.689728e+01
50%   -8.747690e+01  2.456000e+03  1.349250e+09  3.936568e+01 -8.743839e+01
75%   -8.015800e+01  2.032800e+04  1.359385e+09  4.195716e+01 -8.023680e+01
max   -6.795030e+01  2.906700e+06  1.371817e+09  6.751027e+01 -6.695090e+01

           is_fraud
count  1.296675e+06
mean   5.788652e-03
std    7.586269e-02
min    0.000000e+00
25%    0.000000e+00
50%    0.000000e+00
75%    0.000000e+00
max    1.000000e+00
```

```
# Visualize the distribution of the target variable (fraudulent or not)
plt.figure(figsize=(8, 6))
sns.countplot(x='is_fraud', data=train_data)
plt.title('Distribution of Fraudulent Transactions')
plt.xlabel('Is Fraud')
plt.ylabel('Count')
plt.show()
```

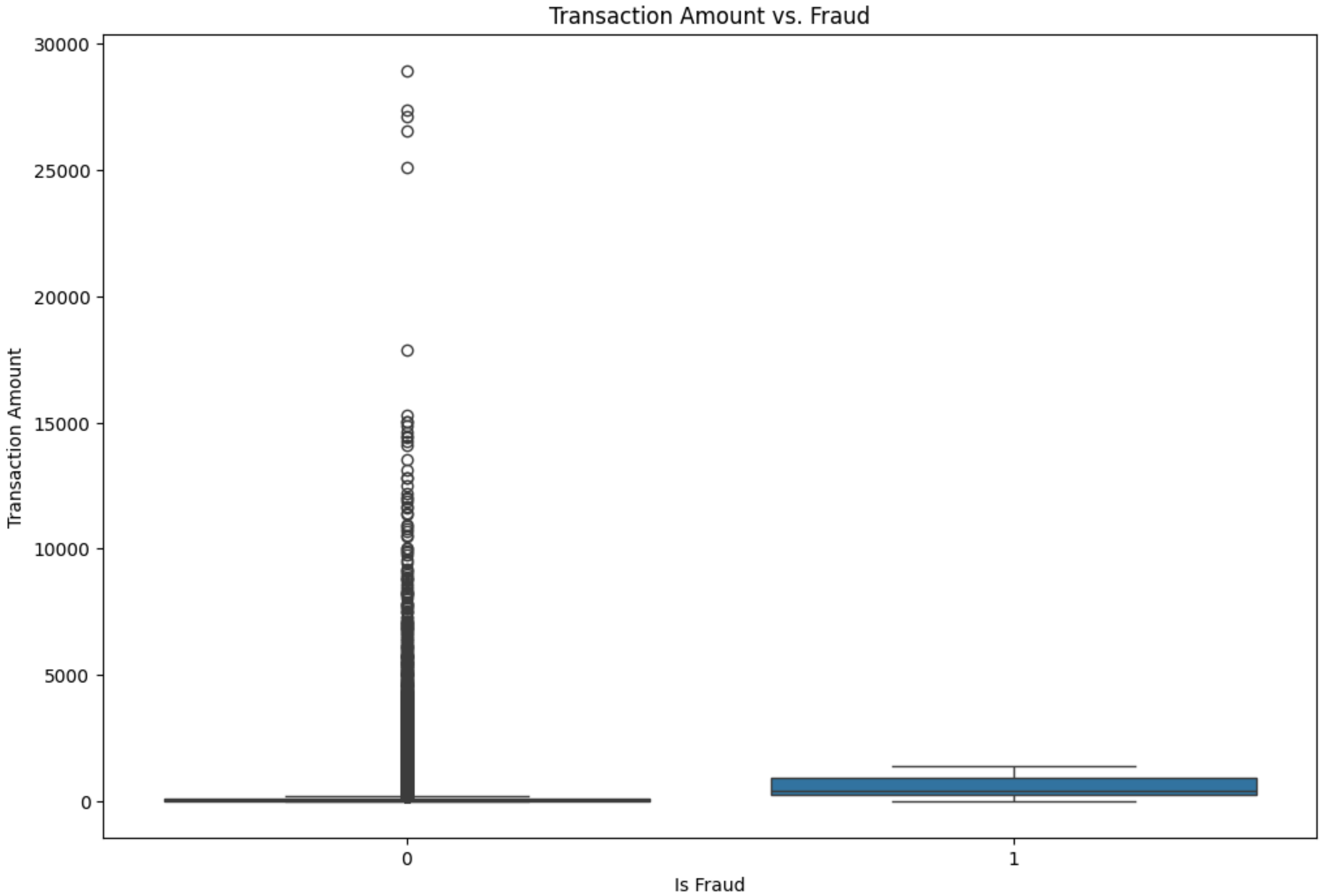



```
# X-Axis (Is Fraud): This axis represents the class labels for the transactions.
# Typically, 0 indicates non-fraudulent transactions, and 1 indicates fraudulent transactions.
# Y-Axis (Count): This axis represents the count of transactions in each class.
```

```
"""The bar for 0 (non-fraudulent transactions) is significantly higher than the bar for 1 (fraudulent transactions),
indicating that there are many more non-fraudulent transactions compared to fraudulent ones in your dataset.
The imbalance between the two classes is quite severe, which is common in fraud detection datasets.
This class imbalance can affect the performance of machine learning models,
as they may become biased towards the majority class (non-fraudulent transactions)."""
```

```
'The bar for 0 (non-fraudulent transactions) is significantly higher than the bar fo
r 1 (fraudulent transactions),\nindicating that there are many more non-fraudulent t
ransactions compared to fraudulent ones in your dataset.\nThe imbalance between the
two classes is quite severe, which is common in fraud detection datasets.\nThis clas
```

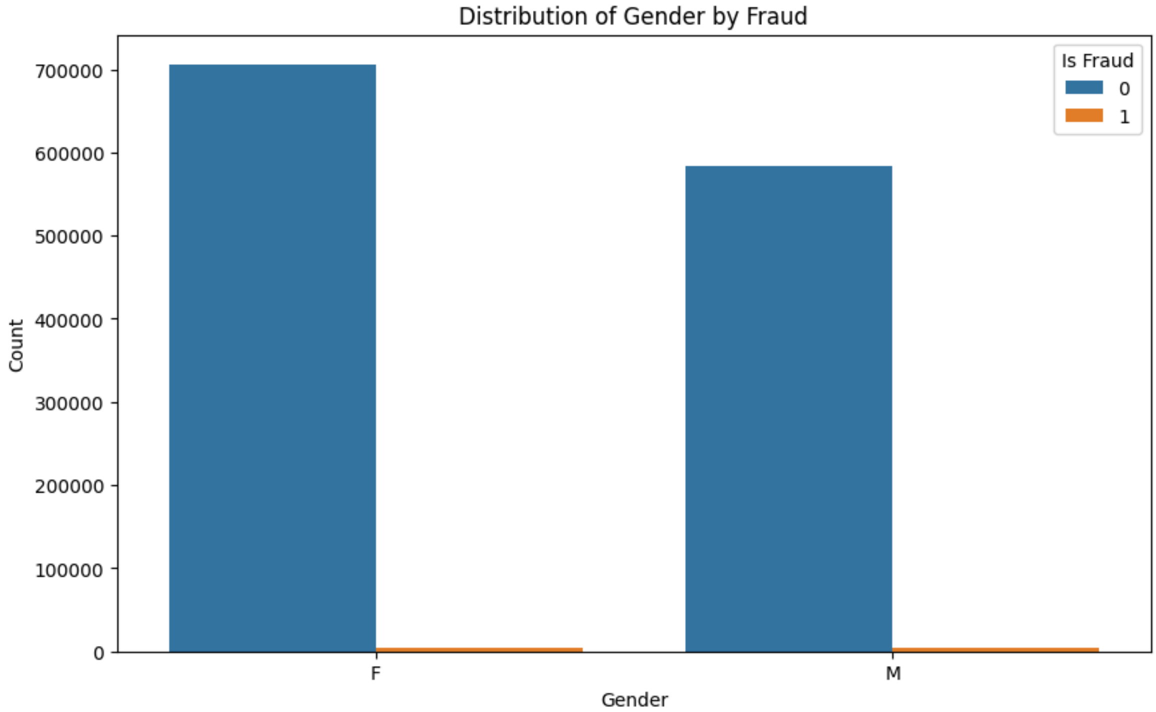
```
# Explore the distribution of transaction amounts by fraud status
plt.figure(figsize=(12, 8))
sns.boxplot(x='is_fraud', y='amt', data=train_data)
plt.title('Transaction Amount vs. Fraud')
plt.xlabel('Is Fraud')
plt.ylabel('Transaction Amount')
plt.show()
```




```
"""Non-Fraudulent Transactions (0):
The transaction amounts for non-fraudulent transactions have a wide range with many outliers, as indicated by the numerous circles above
Most of the transaction amounts are clustered towards the lower end, with a median value close to zero.
The presence of many outliers indicates that there are some very high-value transactions, but they are not the majority.

Fraudulent Transactions (1):
The transaction amounts for fraudulent transactions have a much narrower range compared to non-fraudulent transactions.
The median value for fraudulent transactions is higher than that for non-fraudulent transactions.
There are fewer outliers in fraudulent transactions, suggesting that fraudulent transactions are more consistently within a certain rang
```

```
'Non-Fraudulent Transactions (0):\nThe transaction amounts for non-fraudulent transactions have a wide range with many outliers, as
indicated by the numerous circles above the box.\nMost of the transaction amounts are clustered towards the lower end, with a media
n value close to zero.\nThe presence of many outliers indicates that there are some very high-value transactions, but they are not
the majority.\n\nFraudulent Transactions (1):\nThe transaction amounts for fraudulent transactions have a much narrower range compa
red to non-fraudulent transactions.\nThe median value for fraudulent transactions is higher than that for non-fraudulent transactio
ns.\nThere are fewer outliers in fraudulent transactions, suggesting that fraudulent transactions are more consistently within a ce
```

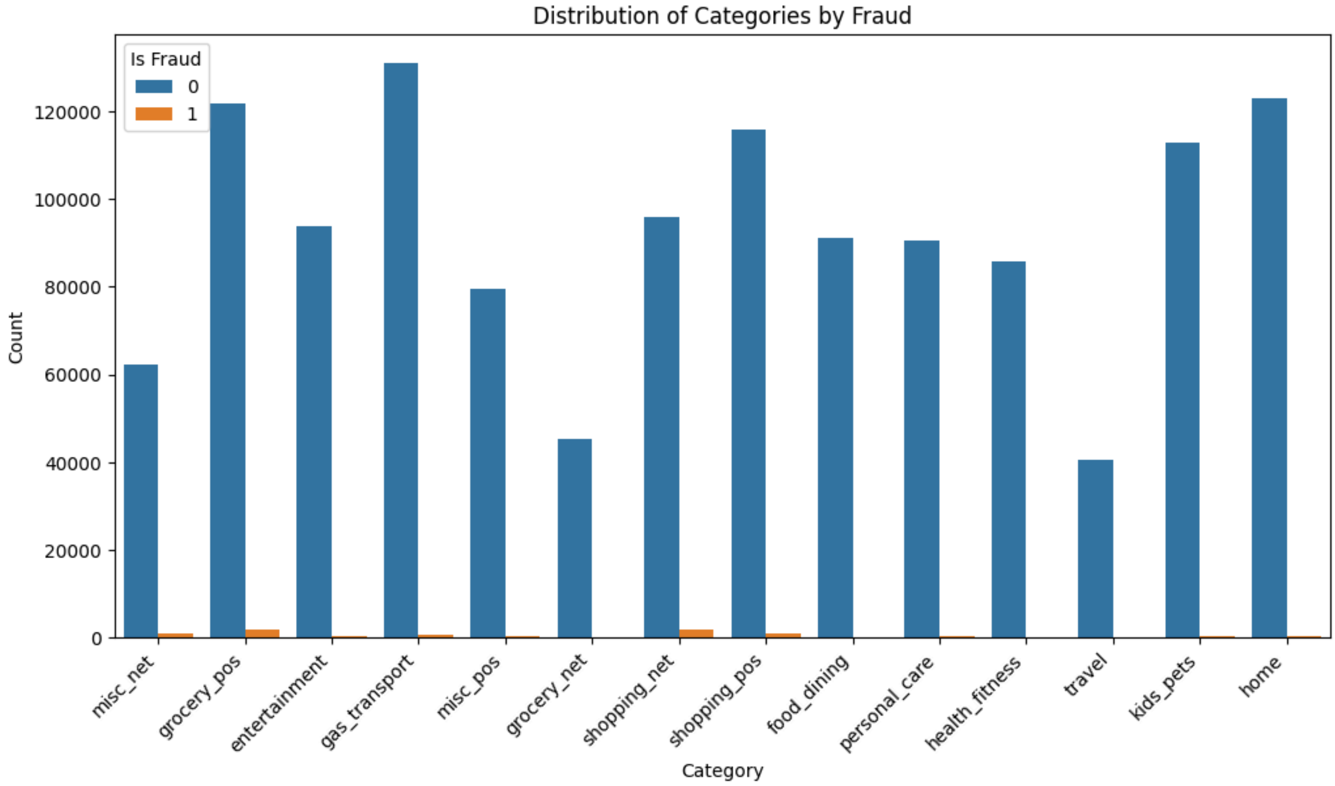
```
# Explore categorical features (e.g., gender)
plt.figure(figsize=(10, 6))
sns.countplot(x='gender', hue='is_fraud', data=train_data)
plt.title('Distribution of Gender by Fraud')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(title='Is Fraud')
plt.show()
```



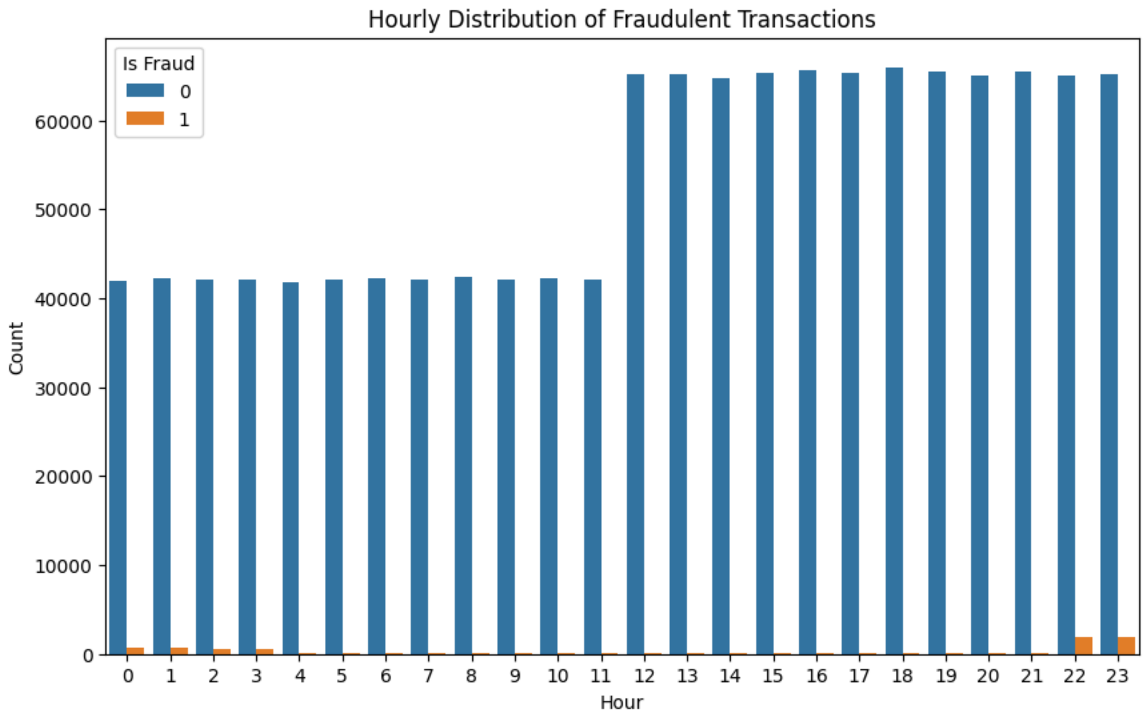

```
"""
There are slightly more non-fraudulent transactions for females compared to males.
The count of fraudulent transactions is very small for both genders, as indicated by the very small orange bars.
There doesn't appear to be a significant difference between the number of fraudulent transactions for females and males.

Gender Distribution: The distribution of fraudulent transactions across genders seems quite balanced,
suggesting that gender may not be a strong discriminator for fraudulent transactions in this dataset.
"""
```

```
'\nThere are slightly more non-fraudulent transactions for females compared to males.\nThe count of fraudulent transactions is very small for both genders, as indicated by the very small orange bars.\nThere doesn't appear to be a significant difference between the number of fraudulent transactions for females and males.\n\nGender Distribution: The distribution of fraudulent transactions across genders seems quite balanced.\nsuggesting that gender may not be a strong discriminator for fraudulent transactions in this data
```

```
# Explore categorical features (e.g., category)
plt.figure(figsize=(12, 6))
sns.countplot(x='category', hue='is_fraud', data=train_data)
plt.title('Distribution of Categories by Fraud')
plt.xlabel('Category')
plt.ylabel('Count')
plt.xticks(rotation=45, ha="right")
plt.legend(title='Is Fraud')
plt.show()
```

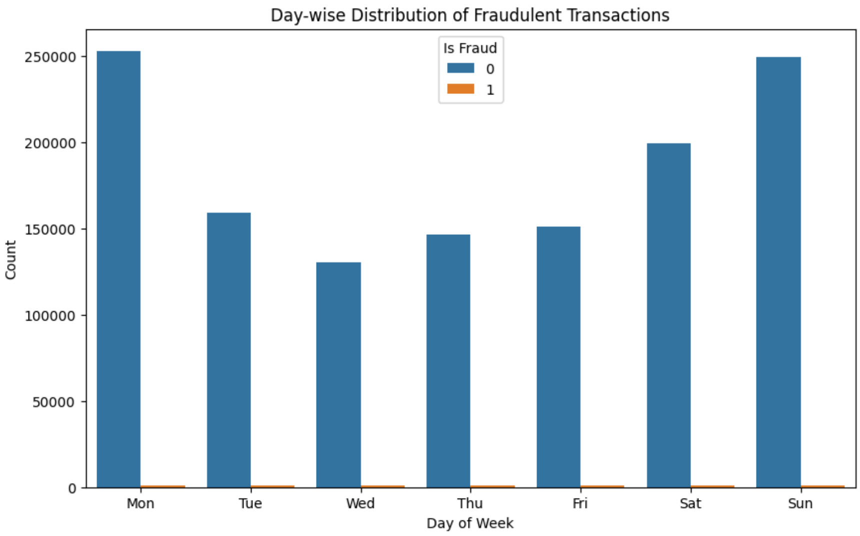



```
# Time analysis: Extract hours and days from 'trans_date_trans_time'
train_data['trans_hour'] = pd.to_datetime(train_data['trans_date_trans_time']).dt.hour
train_data['trans_day'] = pd.to_datetime(train_data['trans_date_trans_time']).dt.dayofweek

# Plot hourly distribution of fraud
plt.figure(figsize=(10, 6))
sns.countplot(x='trans_hour', hue='is_fraud', data=train_data)
plt.title('Hourly Distribution of Fraudulent Transactions')
plt.xlabel('Hour')
plt.ylabel('Count')
plt.legend(title='Is Fraud')
plt.show()
```

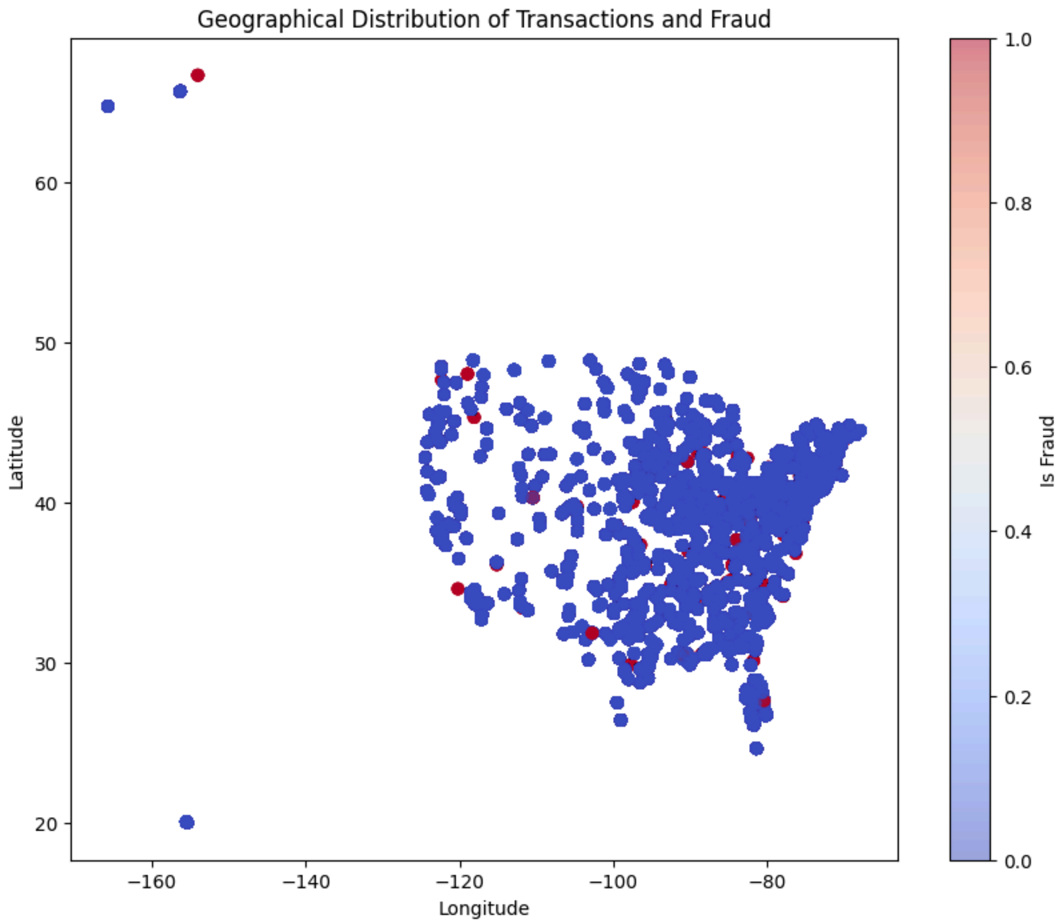



```
# Plot day-wise distribution of fraud
plt.figure(figsize=(10, 6))
sns.countplot(x='trans_day', hue='is_fraud', data=train_data)
plt.title('Day-wise Distribution of Fraudulent Transactions')
plt.xlabel('Day of Week')
```

```
plt.ylabel('Count')
plt.xticks([0, 1, 2, 3, 4, 5, 6], ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'])
plt.legend(title='Is Fraud')
plt.show()
```

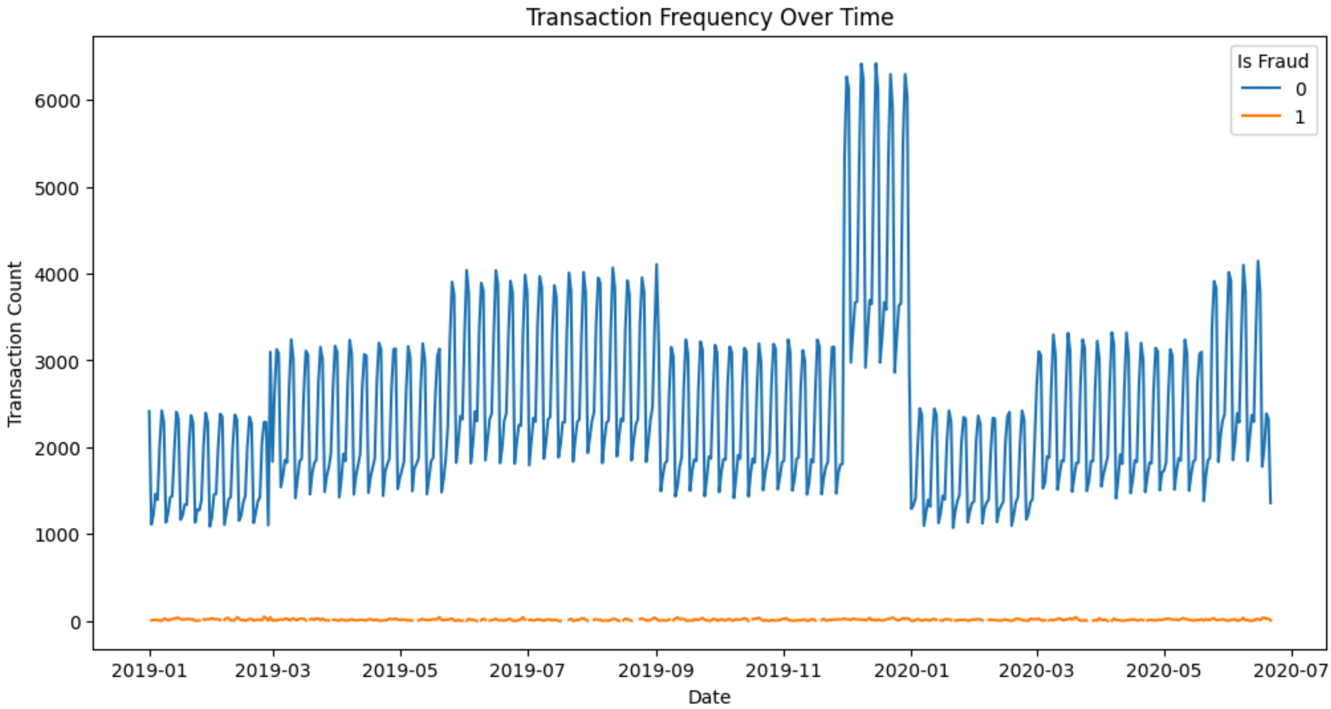



```
# Scatter plot of geographical data
plt.figure(figsize=(10, 8))
plt.scatter(train_data['long'], train_data['lat'], c=train_data['is_fraud'], cmap='coolwarm', alpha=0.5)
plt.title('Geographical Distribution of Transactions and Fraud')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.colorbar(label='Is Fraud')
plt.show()
```

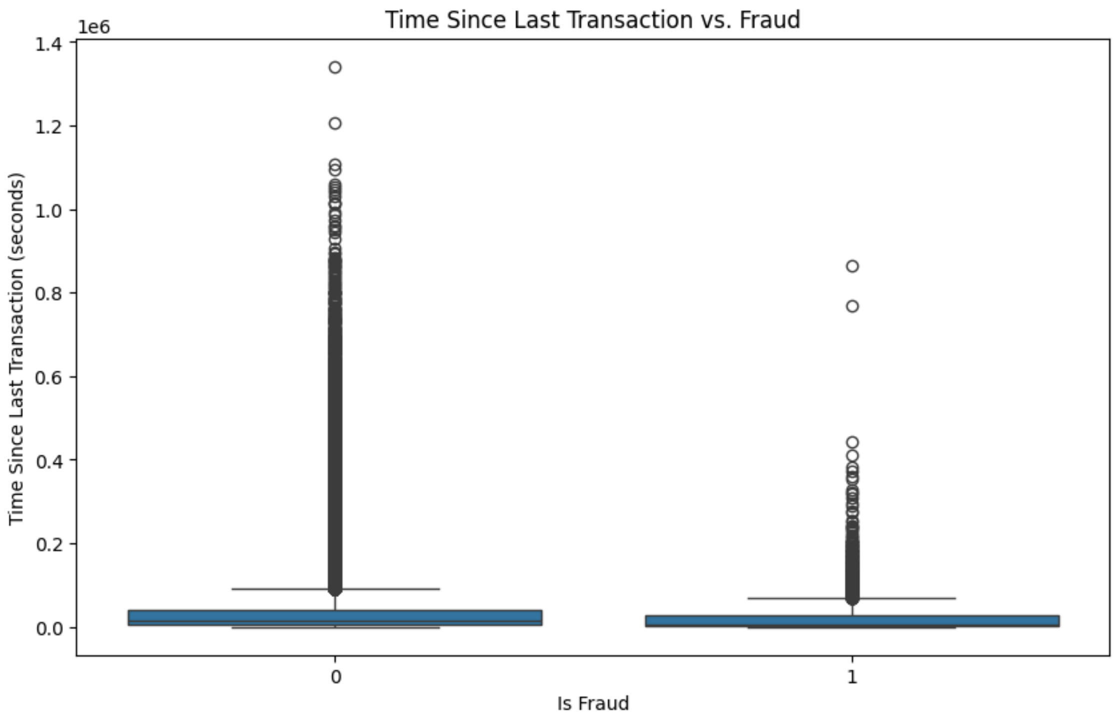


```
# Transaction Frequency Analysis
plt.figure(figsize=(10, 6))
train_data['trans_date_trans_time'] = pd.to_datetime(train_data['trans_date_trans_time'])
train_data['trans_date'] = train_data['trans_date_trans_time'].dt.date
transaction_counts = train_data.groupby(['trans_date', 'is_fraud']).size().unstack()
transaction_counts.plot(kind='line', figsize=(12, 6))
plt.title('Transaction Frequency Over Time')
plt.xlabel('Date')
plt.ylabel('Transaction Count')
plt.legend(title='Is Fraud')
plt.show()
```

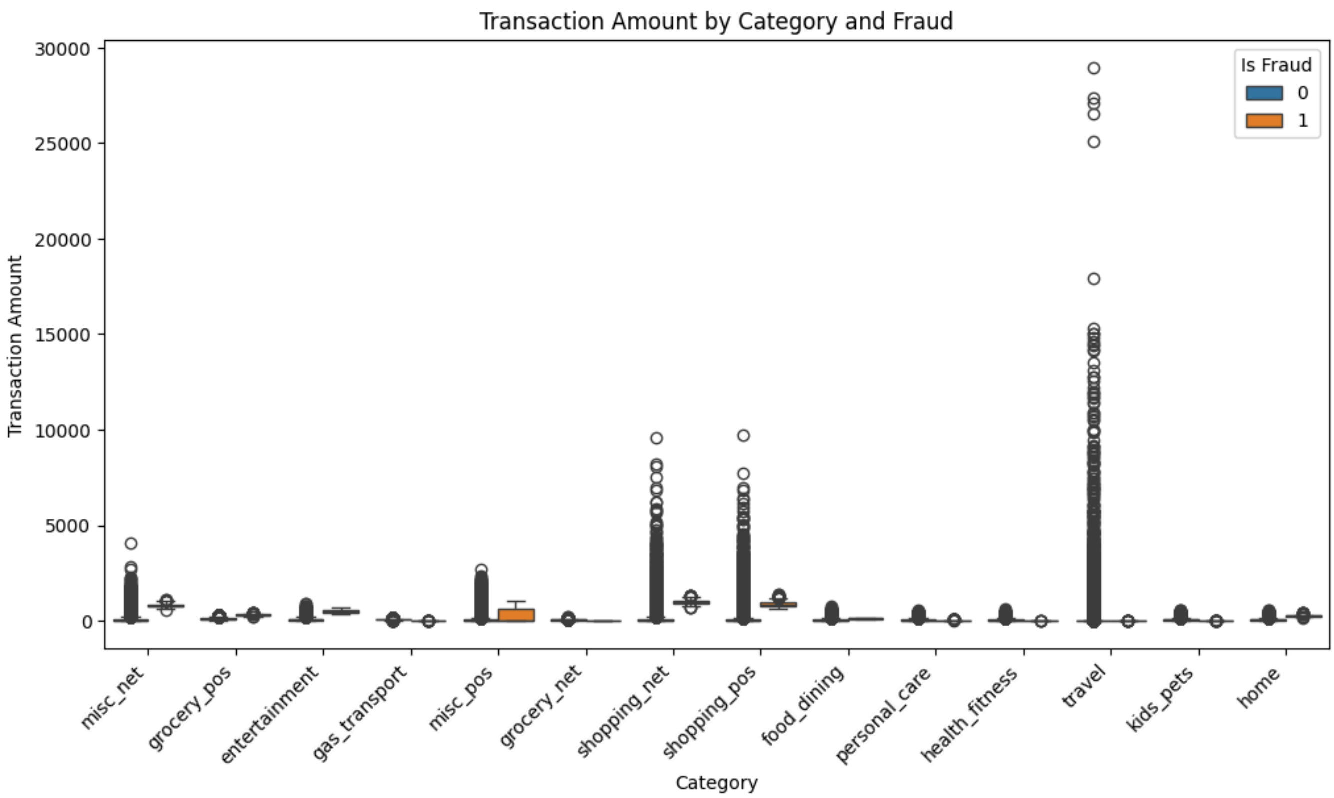
```
<Figure size 1000x600 with 0 Axes>
```



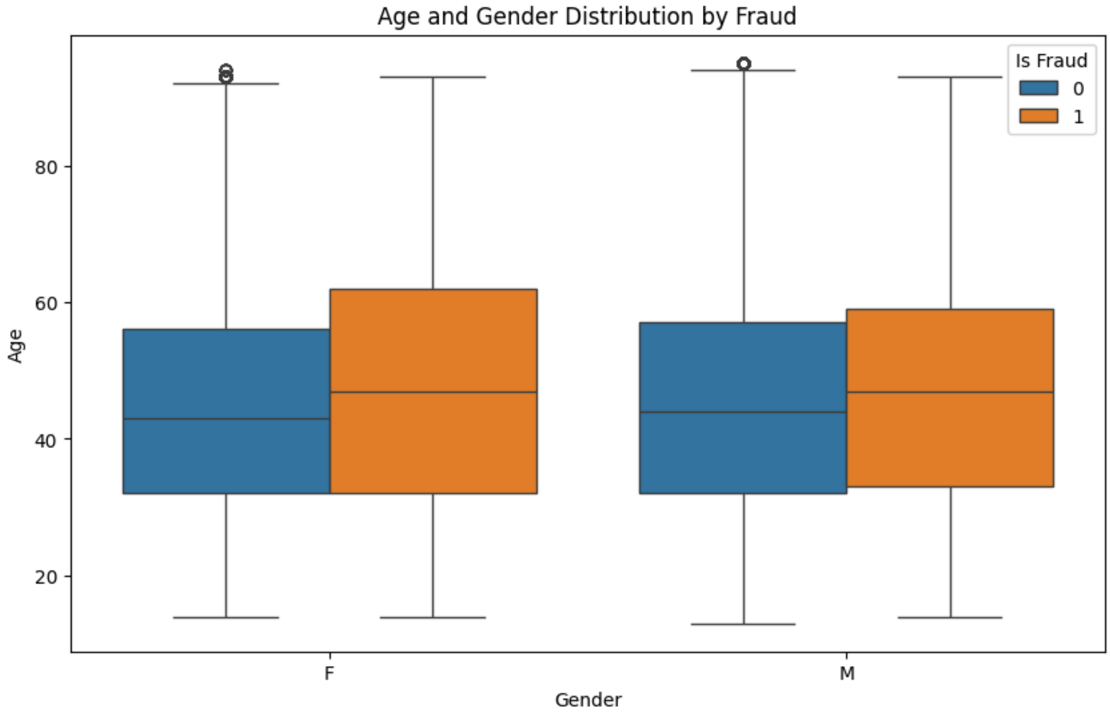

```
# Transaction Velocity Analysis
train_data['time_since_last_transaction'] = train_data.groupby('cc_num')['unix_time'].diff()
plt.figure(figsize=(10, 6))
sns.boxplot(x='is_fraud', y='time_since_last_transaction', data=train_data)
plt.title('Time Since Last Transaction vs. Fraud')
plt.xlabel('Is Fraud')
plt.ylabel('Time Since Last Transaction (seconds)')
plt.show()
```



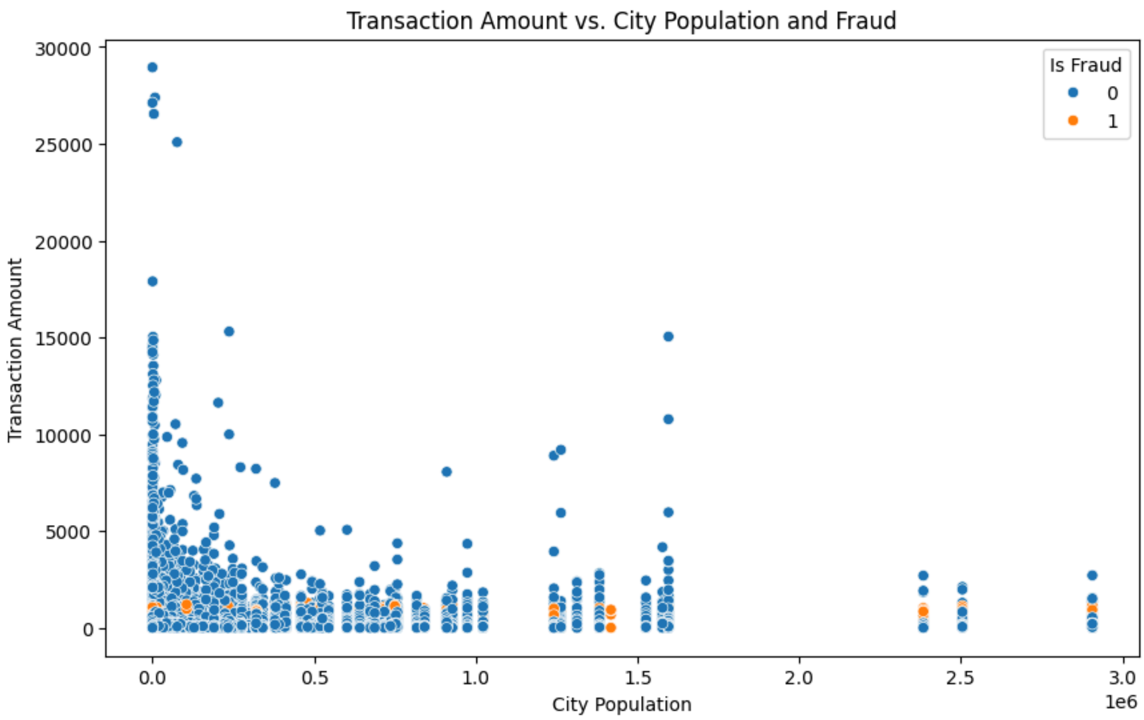
```
# Transaction Amount by Category
plt.figure(figsize=(12, 6))
sns.boxplot(x='category', y='amt', hue='is_fraud', data=train_data)
plt.title('Transaction Amount by Category and Fraud')
plt.xlabel('Category')
plt.ylabel('Transaction Amount')
plt.xticks(rotation=45, ha="right")
plt.legend(title='Is Fraud')
plt.show()
```



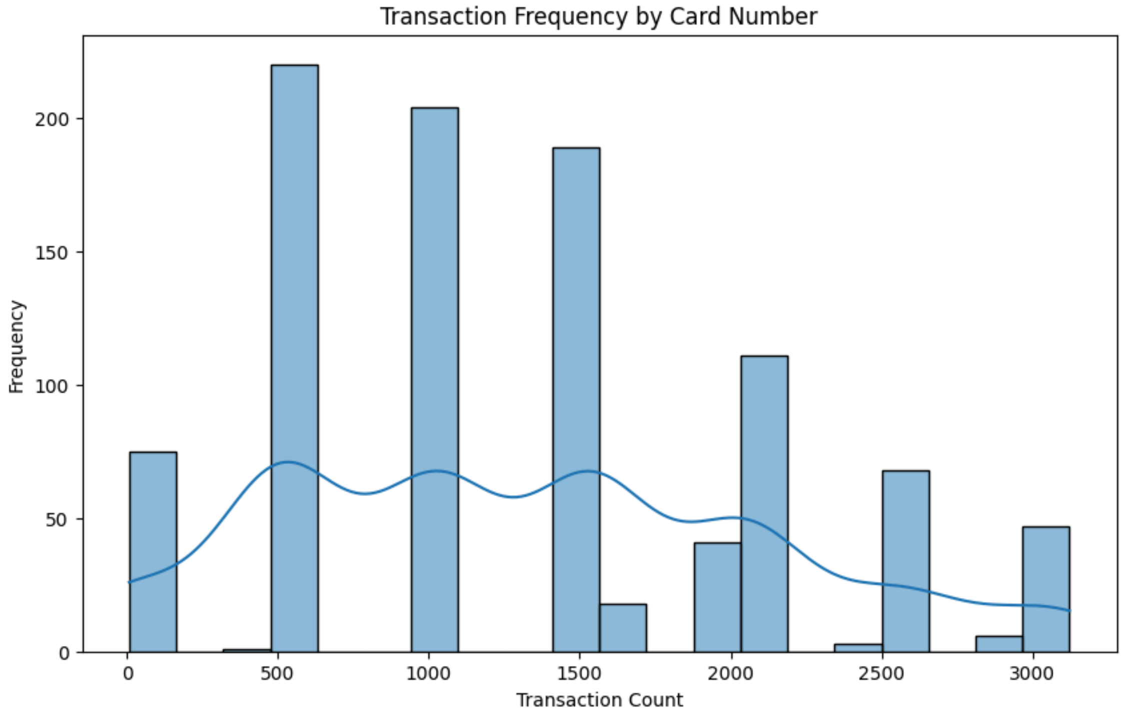

```
# Age and Gender Analysis
train_data['dob'] = pd.to_datetime(train_data['dob'])
train_data['age'] = (train_data['trans_date_trans_time'] - train_data['dob']).dt.days // 365
plt.figure(figsize=(10, 6))
sns.boxplot(x='gender', y='age', hue='is_fraud', data=train_data)
plt.title('Age and Gender Distribution by Fraud')
plt.xlabel('Gender')
plt.ylabel('Age')
plt.legend(title='Is Fraud')
plt.show()
```



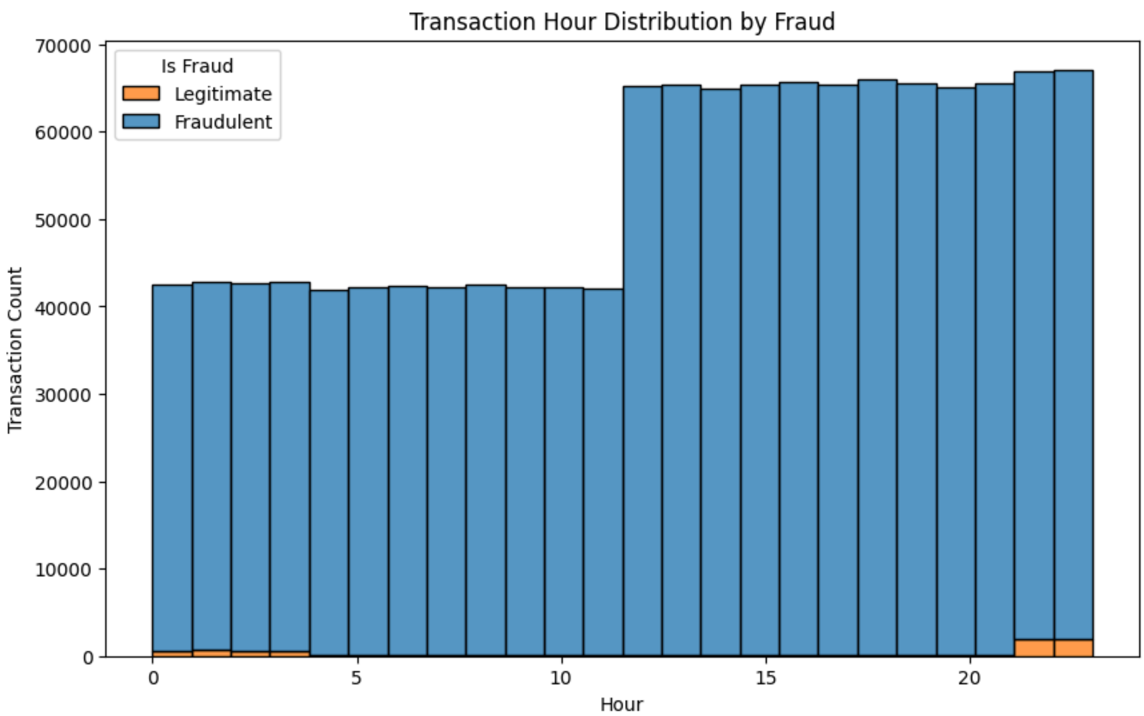

```
# Transaction Amount vs. City Population
plt.figure(figsize=(10, 6))
sns.scatterplot(x='city_pop', y='amt', hue='is_fraud', data=train_data)
plt.title('Transaction Amount vs. City Population and Fraud')
plt.xlabel('City Population')
plt.ylabel('Transaction Amount')
plt.legend(title='Is Fraud')
plt.show()
```




```
# Transaction Frequency by Card Number
card_transaction_counts = train_data['cc_num'].value_counts()
plt.figure(figsize=(10, 6))
sns.histplot(card_transaction_counts, bins=20, kde=True)
plt.title('Transaction Frequency by Card Number')
plt.xlabel('Transaction Count')
plt.ylabel('Frequency')
plt.show()
```




```
# Transaction Time and Fraud Correlation
train_data['trans_hour'] = pd.to_datetime(train_data['trans_date_trans_time']).dt.hour
plt.figure(figsize=(10, 6))
sns.histplot(data=train_data, x='trans_hour', hue='is_fraud', multiple='stack', bins=24)
plt.title('Transaction Hour Distribution by Fraud')
plt.xlabel('Hour')
plt.ylabel('Transaction Count')
plt.legend(title='Is Fraud', labels=['Legitimate', 'Fraudulent'])  # Specify legend labels
plt.show()
```



MODEL BUILDING

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_curve, auc, confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
```

```
# Encode categorical variables
encoder = OneHotEncoder(drop='first')
categorical_cols = ['gender', 'category', 'state']
encoded_train_features = encoder.fit_transform(train_data[categorical_cols]).toarray()
encoded_test_features = encoder.transform(test_data[categorical_cols]).toarray()
```

```
# Feature scaling
scaler = StandardScaler()
numerical_cols = ['amt', 'lat', 'long','city_pop', 'unix_time', 'merch_lat', 'merch_long']
scaled_train_features = scaler.fit_transform(train_data[numerical_cols])
scaled_test_features = scaler.transform(test_data[numerical_cols])
```

```
# Concatenate encoded and scaled features for both train and test data
final_train_features = pd.concat([pd.DataFrame(encoded_train_features), pd.DataFrame(scaled_train_features)], axis=1)
final_test_features = pd.concat([pd.DataFrame(encoded_test_features), pd.DataFrame(scaled_test_features)], axis=1)
```

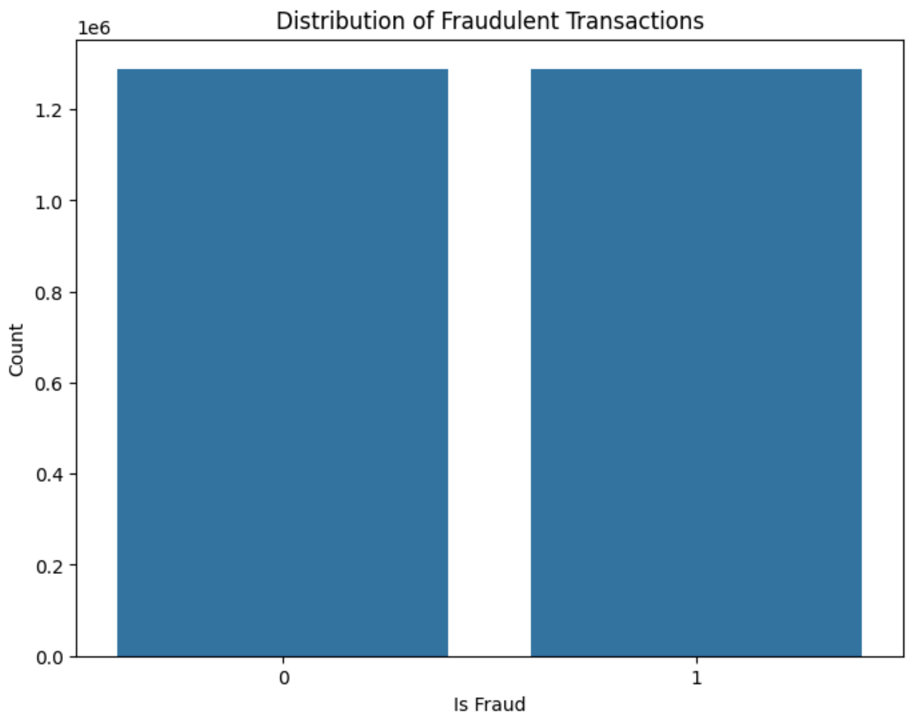
```
# Define target variables
train_target = train_data['is_fraud']
test_target = test_data['is_fraud']
```

```
# Generating synthetic data to balance the imbalanced dataset
smote = SMOTE(random_state=36)

x_train_resample, y_train_resample = smote.fit_resample(final_train_features, train_target)
```

```
# checking newly created data
print('Current length of the training set: ', len(y_train_resample))
```

```
Current length of the training set:  2578338
```

```
plt.figure(figsize=(8, 6))
sns.countplot(x=y_train_resample)
plt.title('Distribution of Fraudulent Transactions')
plt.xlabel('Is Fraud')
plt.ylabel('Count')
plt.show()
```




```
X_shuffled, y_shuffled = shuffle(x_train_resample, y_train_resample, random_state=42)
```

```
x_train, x_validation, y_train, y_validation = train_test_split(X_shuffled, y_shuffled, test_size=0.5)
```

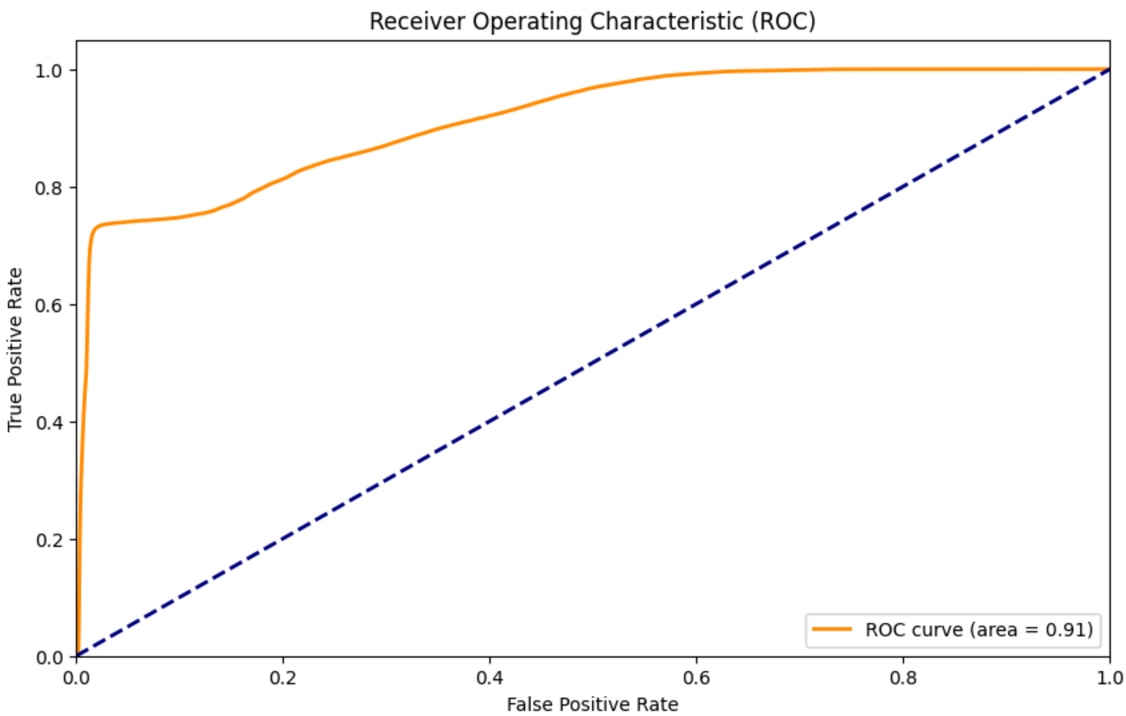
```
# for the initial selection process we will use a tiny
# portion of the actual training dataset
x_train_copy = x_train
y_train_copy = y_train
```

```
x_train = x_train[:10000]
y_train = y_train[:10000]
```

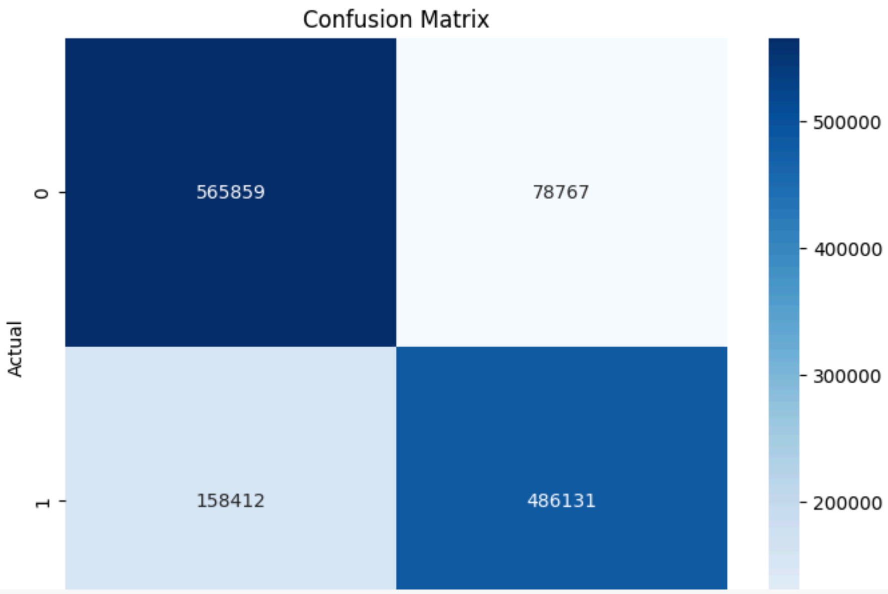
```
# Train Logistic Regression model
lg_model = LogisticRegression()
lg_model.fit(x_train, y_train)

# Make predictions on test data
lg_predictions = lg_model.predict(x_validation)

# Calculate evaluation metrics on test data
lg_accuracy = accuracy_score(y_validation, lg_predictions)


# Print evaluation metrics with 3 decimal places, multiplied by 100
print("Logistic Regression Accuracy: {:.3f}%".format(lg_accuracy * 100))
```

```
/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(
Logistic Regression Accuracy: 81.602%
```

```
# Calculate ROC curve and AUC
probs = lg_model.predict_proba(x_validation)[:, 1]
fpr, tpr, thresholds = roc_curve(y_validation, probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
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 (ROC)')
plt.legend(loc="lower right")
plt.show()
```



```
# Calculate and plot confusion matrix
conf_matrix = confusion_matrix(y_validation, lg_predictions)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
# Train Random Forest model
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier()
rf_model.fit(x_train, y_train)
# Make predictions on test data
rf_predictions = rf_model.predict(x_validation)

# Calculate evaluation metrics on test data
rf_accuracy = accuracy_score(y_validation, rf_predictions)


# Print evaluation metrics with 3 decimal places, multiplied by 100
print("Random Forest Accuracy: {:.3f}%".format(rf_accuracy * 100))
```