fraudModel

December 4, 2024

1 Transaction Fraud Detection Model

Steven Weeden, Danny Phan, Michael Lleverino

1.1 Project Summary

For this project, we are leveraging a synthetic dataset of financial transactions designed for fraud detection to help us produce a fraud detection model. We will do this by finding patterns between fraudulent and legitmate transactions and using that data to determine if future transactions are legit.

1.2 Problem Statement

This project focuses on developing machine learning models to detect fraud using a realistic, synthetic financial transaction dataset.

<Expand the section with few sentences for the *Project Progress* assignment submission>

The goal of this project is to build a machine learning model that helps predict if a transaction is considered fradulent or legitimate. We will achieve this by analyzing the transaction data to find anomolies and patterns.

We will use logistical regression as our benchmark. This is a simple model to produce and will provide insight on wheater this has a linear or non-linear approach.

This data comes from Kaggle and contains millions of lines of transaction data. We plan on using our benchmark as a comparaison in preformance between models.

Our goal is to make a successful machine learning model that can successfully and accuratly predict whether or not a transaction is considered fradulent.

<Finalize for the *Project Submission* assignment submission>

[1]:
```
import pandas as pd
from sklearn.metrics import classification_report, roc_auc_score,␣
 ↪confusion_matrix
from xgboost import XGBClassifier, plot_importance
from imblearn.over_sampling import SMOTE
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

```python
df = pd.read_csv("../data/synthetic_fraud_data.csv")

df.head()
```

[1]:
```
  transaction_id customer_id       card_number  \
0    TX_a0ad2a2a  CUST_72886  6646734767813109
1    TX_3599c101  CUST_70474   376800864692727
2    TX_a9461c6d  CUST_10715  5251909460951913
3    TX_7be21fc4  CUST_16193   376079286931183
4    TX_150f490b  CUST_87572  6172948052178810

                          timestamp merchant_category merchant_type  \
0  2024-09-30 00:00:01.034820+00:00        Restaurant     fast_food
1  2024-09-30 00:00:01.764464+00:00     Entertainment        gaming
2  2024-09-30 00:00:02.273762+00:00           Grocery      physical
3  2024-09-30 00:00:02.297466+00:00               Gas         major
4  2024-09-30 00:00:02.544063+00:00        Healthcare       medical

         merchant     amount currency    country  …  device channel  \
0       Taco Bell     294.87      GBP         UK  … iOS App  mobile
1           Steam    3368.97      BRL     Brazil  …    Edge     web
2     Whole Foods  102582.38      JPY      Japan  … Firefox     web
3           Exxon     630.60      AUD  Australia  … iOS App  mobile
4  Medical Center  724949.27      NGN    Nigeria  …  Chrome     web

                 device_fingerprint       ip_address distance_from_home  \
0  e8e6160445c935fd0001501e4cbac8bc   197.153.60.199                  0
1  a73043a57091e775af37f252b3a32af9  208.123.221.203                  1
2  218864e94ceaa41577d216b149722261   10.194.159.204                  0
3  70423fa3a1e74d01203cf93b51b9631d   17.230.177.225                  0
4  9880776c7b6038f2af86bd4e18a1b1a4  136.241.219.151                  1

  high_risk_merchant transaction_hour weekend_transaction  \
0              False                0               False
1               True                0               False
2              False                0               False
3              False                0               False
4              False                0               False

                                   velocity_last_hour  is_fraud
0  {'num_transactions': 1197, 'total_amount': 334…     False
1  {'num_transactions': 509, 'total_amount': 2011…      True
2  {'num_transactions': 332, 'total_amount': 3916…     False
3  {'num_transactions': 764, 'total_amount': 2201…     False
4  {'num_transactions': 218, 'total_amount': 4827…      True
```

```
[5 rows x 24 columns]
```

1.3 Dataset

The shape of our dataset is (7483766, 24) which totales to 179,610,384 entries within this dataset.

Our attributes include transaction_id, customer_id, card_number, timestamp, merchant_category, merchant_type, merchant, amount, currency, country, city, city_size, card_type, card_present, device,channel, device_fingerprint, ip_address, distance_from_home, high_risk_merchant, transaction_hour, weekend_transaction, velocity_last_hour, is_fraud

<Complete the following for the **Project Progress**> * Description: * 1. Transaction ID: Unique identifiers for each transaction. Useful for referencing individual transactions. * 2. Customer ID: Unique identifiers for each customers. Useful for keeping track of customer activity and behavior across transactions. * 3. Card Number: A masked card number representing the credit or debit card used. * 4. Timestamp: Time in UTC format indicating when the transaction occured. Helpful for indicating when and how often the card was used. * 5. Merchant Category: High-level category for merchants, such as 'Retail' or 'Travel'. * 6. Merchant Type: Specifies the subtype of merchants within each category, such as 'Online' or 'Retail'. * 7. Merchant: The name of the merchant where the transaction took place. * 8. Amount: The amout of money spent at that merchant, local to the transaction's country. * 9. Currency: Currency code (e.g., USD, EUR) used for the transaction. * 10. Country: Country where the transaction took place, could be used for detecting trends. * 11. City: Name of the city where the transaction was made. * 12. City Size: Classification of the city size (e.g., large, medium). * 13. Card Type: Type of card used in the transaction, such as 'Gold Credit' or 'Basic Debit'. * 14. Card Present: Boolean values indicating if the card was physically present during the transaction, could be important for differentiating between in-person and online transactions. * 15. Device: Types device or browser used for the transaction (e.g., Chrome, iOS App). * 16. Channel: Types of transaction channel (web, mobile, pos). * 17. Device Fingerprint: Unique identifier for the device used, generated using hashing. * 18. IP Address: The IP address used in the transaction, simulated for privacy concerns. * 19. Distance From Home: Binary indicating whether the transaction occurred outside the customer's home country. * 20. High Risk Merchant: Boolean values that indicates higher-risk merchant categories (e.g., Travel, Entertainment). * 21. Transaction Hour: The hours between (0-23) of when the transaction occurred. * 22. Weekend Transaction: Boolean values indicating whether or not the transaction occurred on a weekend. * 23. Velocity Last Hour: Dictionary of velocity metrics within the past hour, including: num_transactions: Number of customer transactions in the last hour. total_amount: Total amount spent in the last hour. unique_merchants: Count of unique merchants in the last hour. unique_countries: Count of unique countries in the last hour. max_single_amount: Maximum single transaction amount in the last hour. * 24. Is Fraud : Binary label for fraud status (True/False).

<Expand and complete for *Project Submission*>

- What Processing Tools have you used. Why? Add final images from jupyter notebook. Use questions from 3.4 of the Datasheets For Datasets paper for a guide.>

[2]:
```
print(df.shape)
print(df.size)
```

```
(7483766, 24)
179610384
```

1.4 Data Preprocessing

<Complete for *Project Progress*> * Yes we will scale the data we are using. * First we will normalize our data since some of our features have a wide range of values. * We will then use a correlation matrix to help indentify irrelevant data and remove highly correlated features. * Next we are planning on using PCA to reduce high-dimensional features while retaining most of the varience. * Lastly we plan on using SMOTE to balance our dataset since there is a lot more non-fradulent data than there is actual fraud.

<Expand and complete for **Project Submission**>

Sample the data because of the large dataset

[3]:
```
sample = df.sample(n = 250000, random_state = 21).reset_index(drop = True)
sample.head()
```

[3]:
```
  transaction_id customer_id       card_number  \
0    TX_b5b17800  CUST_46478  5699657752967952
1    TX_9c391f0e  CUST_49102   371384150413036
2    TX_c006b556  CUST_60071   379239643871947
3    TX_1ec4e8e0  CUST_85188  6915403402912841
4    TX_6b0f6cba  CUST_12828   376941479073646

                         timestamp merchant_category merchant_type  \
0  2024-10-07 17:14:22.181495+00:00     Entertainment     streaming
1  2024-10-16 14:20:16.817017+00:00        Restaurant       premium
2  2024-10-30 16:30:10.768556+00:00     Entertainment     streaming
3  2024-10-03 08:21:32.928515+00:00        Healthcare       medical
4  2024-10-08 22:41:26.566924+00:00        Restaurant        casual

      merchant    amount currency    country  …   device channel  \
0      Spotify    169.37      EUR    Germany  …  Firefox     web
1  Ruth's Chris   823.67      CAD     Canada  …   Chrome     web
2      Spotify  21948.66      RUB     Russia  …   Safari     web
3     Lab Corp   1602.46      NGN    Nigeria  …     Edge     web
4   Applebee's     89.80      SGD  Singapore  …  Firefox     web

                 device_fingerprint      ip_address distance_from_home  \
0  ce7db67743bf5fffdfa5b2ecfb4e6de0  140.170.128.51                  0
1  d654a7f29023652d55a4574b8fc6ea4a     45.93.44.23                  0
2  6d945d3387a7d9eb150ec7d30bd3d621   52.255.14.155                  0
3  4af6ed569634c7b9311dc6968b50e29e    65.194.56.96                  1
4  fcaf3544fe07e389572c90dc02921857  193.111.77.221                  0

  high_risk_merchant transaction_hour weekend_transaction  \
0               True               17               False
```

```
1                False                 14                False
2                 True                 16                False
3                False                  8                False
4                False                 22                False

                                   velocity_last_hour  is_fraud
0  {'num_transactions': 27, 'total_amount': 25271…     False
1  {'num_transactions': 437, 'total_amount': 1567…     False
2  {'num_transactions': 6, 'total_amount': 340463…     False
3  {'num_transactions': 944, 'total_amount': 2583…      True
4  {'num_transactions': 148, 'total_amount': 2367…     False

[5 rows x 24 columns]
```

Finding if any columns in data have high percentage of nulls

[4]:
```
missing_percent = sample.isnull().mean() * 100
print(missing_percent)
```

```
transaction_id         0.0
customer_id            0.0
card_number            0.0
timestamp              0.0
merchant_category      0.0
merchant_type          0.0
merchant               0.0
amount                 0.0
currency               0.0
country                0.0
city                   0.0
city_size              0.0
card_type              0.0
card_present           0.0
device                 0.0
channel                0.0
device_fingerprint     0.0
ip_address             0.0
distance_from_home     0.0
high_risk_merchant     0.0
transaction_hour       0.0
weekend_transaction    0.0
velocity_last_hour     0.0
is_fraud               0.0
dtype: float64
```

Dropping columns that will serve no assistance in determining if transaction is fraud or not

[5]:
```
sample.drop(['transaction_id', 'customer_id', 'card_number'], axis = 1, inplace
↪= True)
```

Get key values from velocity_last_hour column and makes them all columns in the dataframe

[6]:
```python
import ast

parsed_data = []
for value in sample['velocity_last_hour']:
    try:
        parsed_data.append(ast.literal_eval(value) if isinstance(value, str)␣
 ↪else value)
    except (ValueError, SyntaxError):
        parsed_data.append({
            'num_transactions': None,
            'total_amount': None,
            'unique_merchants': None,
            'unique_countries': None,
            'max_single_amount': None
        })

sample['num_transactions'] = [int(item.get('num_transactions', 0)) for item in␣
 ↪parsed_data]
sample['total_amount'] = [float(item.get('total_amount', 0)) for item in␣
 ↪parsed_data]
sample['unique_merchants'] = [int(item.get('unique_merchants', 0)) for item in␣
 ↪parsed_data]
sample['unique_countries'] = [int(item.get('unique_countries', 0)) for item in␣
 ↪parsed_data]
sample['max_single_amount'] = [float(item.get('max_single_amount', 0)) for item␣
 ↪in parsed_data]

sample.drop('velocity_last_hour', axis = 1, inplace = True)
```

[7]:
```python
pd_categorical = sample.copy()
pd_categorical
```

[7]:
```
                               timestamp merchant_category merchant_type  \
0       2024-10-07 17:14:22.181495+00:00     Entertainment     streaming
1       2024-10-16 14:20:16.817017+00:00        Restaurant       premium
2       2024-10-30 16:30:10.768556+00:00     Entertainment     streaming
3       2024-10-03 08:21:32.928515+00:00        Healthcare       medical
4       2024-10-08 22:41:26.566924+00:00        Restaurant        casual
…                                    …                 …             …
249995  2024-10-25 11:39:56.100159+00:00        Restaurant        casual
249996  2024-09-30 08:05:52.579982+00:00           Grocery        online
249997  2024-10-16 08:08:32.574226+00:00         Education        online
249998  2024-10-29 10:36:25.543460+00:00            Retail      physical
249999  2024-10-08 21:09:15.799275+00:00         Education        online
```

```
             merchant     amount currency    country          city city_size  \
0             Spotify     169.37      EUR    Germany  Unknown City    medium
1         Ruth's Chris    823.67      CAD     Canada  Unknown City    medium
2             Spotify   21948.66      RUB     Russia  Unknown City    medium
3            Lab Corp    1602.46      NGN    Nigeria  Unknown City    medium
4          Applebee's      89.80      SGD  Singapore  Unknown City    medium
…                 …          …    …        …          …       …
249995   Olive Garden  513136.06      NGN    Nigeria  Unknown City    medium
249996      Instacart     925.19      SGD  Singapore  Unknown City    medium
249997       Coursera     425.03      USD        USA        Dallas    medium
249998           IKEA      27.31      MXN     Mexico  Unknown City    medium
249999    MasterClass  177666.12      NGN    Nigeria  Unknown City    medium

               card_type  …  distance_from_home high_risk_merchant  \
0          Premium Debit  …                   0               True
1        Platinum Credit  …                   0              False
2            Gold Credit  …                   0               True
3          Premium Debit  …                   1              False
4            Basic Debit  …                   0              False
…                    …  …                 …                 …
249995   Platinum Credit  …                   0              False
249996     Premium Debit  …                   1              False
249997      Basic Credit  …                   1              False
249998       Basic Debit  …                   1              False
249999      Basic Credit  …                   0              False

       transaction_hour weekend_transaction is_fraud  num_transactions  \
0                    17               False    False                27
1                    14               False    False               437
2                    16               False    False                 6
3                     8               False     True               944
4                    22               False    False               148
…                    …                  …       …                 …
249995               11               False    False               130
249996                8               False     True               883
249997                8               False    False               279
249998               10               False     True                98
249999               21               False    False               545

        total_amount  unique_merchants  unique_countries  max_single_amount
0       2.527161e+06                21                 6       1.897486e+06
1       1.567628e+07               102                12       1.491427e+06
2       3.404636e+05                 6                 5       2.592671e+05
3       2.583898e+07               105                12       1.871011e+06
4       2.367273e+06                71                11       6.834904e+05
…                 …                …                 …                 …
249995  6.631012e+07                72                 9       1.217103e+06
```

```
249996  3.013310e+07              105                 12        5.691281e+06
249997  1.766999e+07               95                 12        2.881764e+06
249998  3.765429e+06               64                 12        1.798893e+06
249999  7.691680e+07              103                 12        3.650894e+06

[250000 rows x 25 columns]
```

Converted all Categorical Data to Numerical

[8]:
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
columns_to_encode = ['merchant_type', 'merchant', 'currency', 'city',␣
 ↪'city_size', 'card_present', 'channel', 'high_risk_merchant',␣
 ↪'weekend_transaction', 'is_fraud']
sample[columns_to_encode] = sample[columns_to_encode].apply(lambda col: le.
 ↪fit_transform(col).astype('int64'))
```

[9]:
```python
binary_cols = ['card_present', 'high_risk_merchant', 'weekend_transaction',␣
 ↪'is_fraud']

for col in binary_cols:

    sample[col] = sample[col].map({

        1: 1, 'yes': 1, 'Yes': 1, 'True': 1, True: 1,

        0: 0, 'no': 0, 'No': 0, 'False': 0, False: 0

    }).astype(int)


for col in binary_cols:

    print(f"\n{col} value counts:")

    print(sample[col].value_counts())

    print(f"Unique values: {sorted(sample[col].unique())}")

sample.head()
```

```
card_present value counts:
card_present
0    228336
1     21664
Name: count, dtype: int64
Unique values: [0, 1]
```

```
high_risk_merchant value counts:
high_risk_merchant
0    187504
1     62496
Name: count, dtype: int64
Unique values: [0, 1]

weekend_transaction value counts:
weekend_transaction
0    185669
1     64331
Name: count, dtype: int64
Unique values: [0, 1]

is_fraud value counts:
is_fraud
0    200331
1     49669
Name: count, dtype: int64
Unique values: [0, 1]
```

[9]:

```
                         timestamp merchant_category  merchant_type  \
0  2024-10-07 17:14:22.181495+00:00     Entertainment             14
1  2024-10-16 14:20:16.817017+00:00        Restaurant             13
2  2024-10-30 16:30:10.768556+00:00     Entertainment             14
3  2024-10-03 08:21:32.928515+00:00        Healthcare              9
4  2024-10-08 22:41:26.566924+00:00        Restaurant              2

   merchant    amount  currency    country  city  city_size        card_type  \
0        79    169.37         3    Germany    10          1    Premium Debit
1        72    823.67         2     Canada    10          1  Platinum Credit
2        79  21948.66         8     Russia    10          1      Gold Credit
3        49   1602.46         7    Nigeria    10          1    Premium Debit
4         9     89.80         9  Singapore    10          1      Basic Debit

   …  distance_from_home high_risk_merchant  transaction_hour  \
0  …                   0                  1                17
1  …                   0                  0                14
2  …                   0                  1                16
3  …                   1                  0                 8
4  …                   0                  0                22

  weekend_transaction is_fraud  num_transactions  total_amount  \
0                   0        0                27  2.527161e+06
1                   0        0               437  1.567628e+07
2                   0        0                 6  3.404636e+05
```

```
3                  0       1              944   2.583898e+07
4                  0       0              148   2.367273e+06

   unique_merchants  unique_countries  max_single_amount
0                21                 6       1.897486e+06
1               102                12       1.491427e+06
2                 6                 5       2.592671e+05
3               105                12       1.871011e+06
4                71                11       6.834904e+05

[5 rows x 25 columns]
```

[10]:
```
sample.dtypes
```

[10]:
```
timestamp               object
merchant_category       object
merchant_type            int64
merchant                 int64
amount                 float64
currency                 int64
country                 object
city                     int64
city_size                int64
card_type               object
card_present             int32
device                  object
channel                  int64
device_fingerprint      object
ip_address              object
distance_from_home       int64
high_risk_merchant       int32
transaction_hour         int64
weekend_transaction      int32
is_fraud                 int32
num_transactions         int64
total_amount           float64
unique_merchants         int64
unique_countries         int64
max_single_amount      float64
dtype: object
```

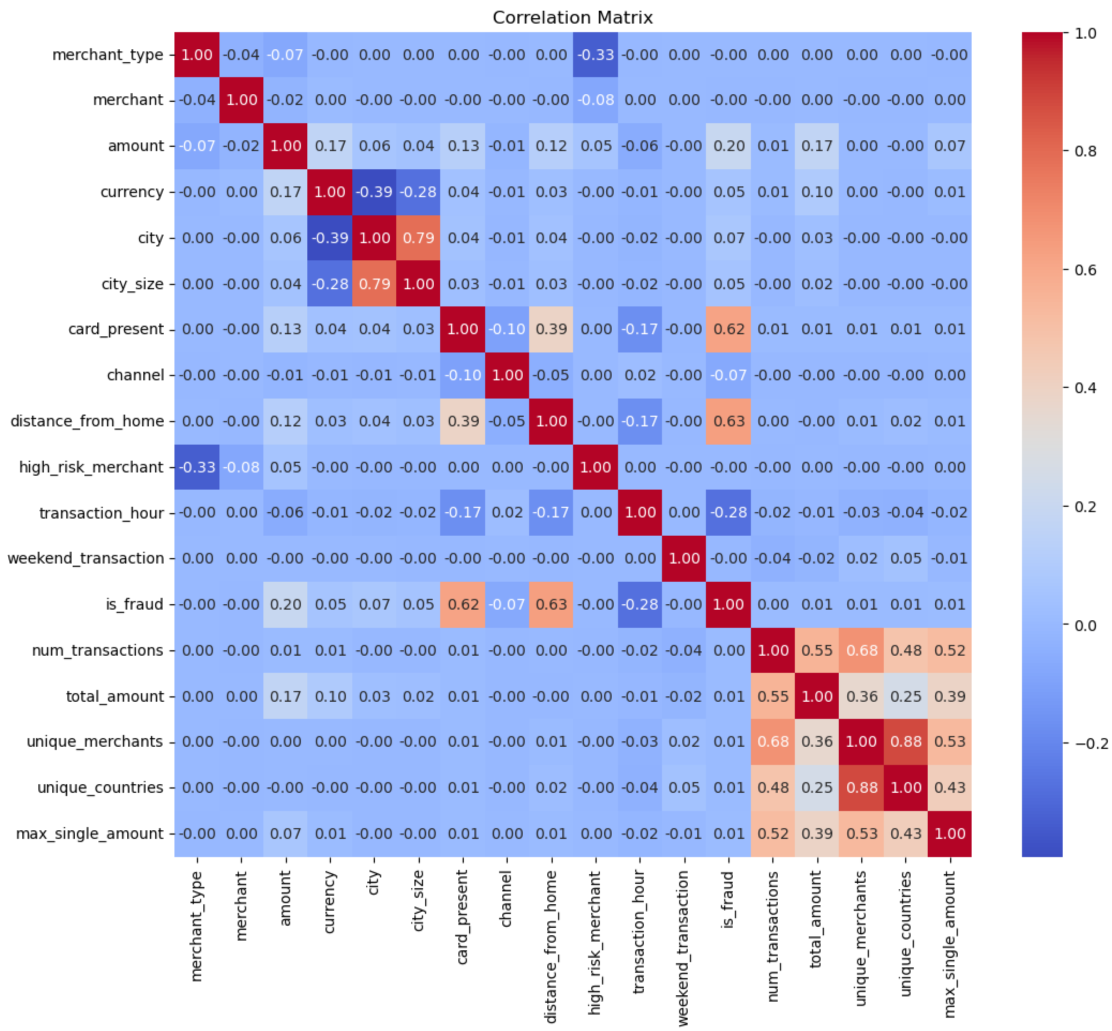
1.5 Exploratory Data Analysis

<Complete for **Project Progress**>

The EDA graphs we are planning to use histograms, box plots, line graphs, and a correlation heatmap as well. We plan to use histograms to visualize the distribution of transaction accounts to see if fraud occurs more in a certain amount. Box plots will be used to find outliers and variations in features and help us find patterns between fraudulent and non fraudulent transactions. Line graphs will help us use line graphs to identify patterns or spikes in fraud over time. Lastly we will be using a correlation heatmap to evaluate the relationship between numerical features and identify strong correlations that may indicate fraud.

<Expand and complete for the **Project Submission**> * Describe the methods you explored (usually algorithms, or data wrangling approaches). * Include images. * Justify methods for feature normalization selection and the modeling approach you are planning to use.

[11]:
```
numerical_columns = sample.select_dtypes(include=['float64', 'int64', 'int32']).columns.tolist()
numerical_data = sample[numerical_columns]
correlation_matrix = numerical_data.corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title("Correlation Matrix")
plt.show()
```



[12]:
```python
nominal_cols = ['merchant_category', 'country', 'card_type', 'device']

sample = pd.get_dummies(

    sample,

    columns=nominal_cols,

    drop_first=True

)



print("\nNew columns created after one-hot encoding:")
```

```
print([col for col in sample.columns if any(x in col for x in nominal_cols)])
```

```
New columns created after one-hot encoding:
['device_fingerprint', 'merchant_category_Entertainment',
'merchant_category_Gas', 'merchant_category_Grocery',
'merchant_category_Healthcare', 'merchant_category_Restaurant',
'merchant_category_Retail', 'merchant_category_Travel', 'country_Brazil',
'country_Canada', 'country_France', 'country_Germany', 'country_Japan',
'country_Mexico', 'country_Nigeria', 'country_Russia', 'country_Singapore',
'country_UK', 'country_USA', 'card_type_Basic Debit', 'card_type_Gold Credit',
'card_type_Platinum Credit', 'card_type_Premium Debit', 'device_Chip Reader',
'device_Chrome', 'device_Edge', 'device_Firefox', 'device_Magnetic Stripe',
'device_NFC Payment', 'device_Safari', 'device_iOS App']
```

[13]:
```
convertBool = sample.select_dtypes('boolean').columns
for boolean_col in convertBool:
    sample[boolean_col] = sample[boolean_col].astype('int')

sample.head()
```

[13]:
```
                         timestamp  merchant_type  merchant    amount  \
0  2024-10-07 17:14:22.181495+00:00             14        79    169.37
1  2024-10-16 14:20:16.817017+00:00             13        72    823.67
2  2024-10-30 16:30:10.768556+00:00             14        79  21948.66
3  2024-10-03 08:21:32.928515+00:00              9        49   1602.46
4  2024-10-08 22:41:26.566924+00:00              2         9     89.80

   currency  city  city_size  card_present  channel  \
0         3    10          1             0        2
1         2    10          1             0        2
2         8    10          1             0        2
3         7    10          1             0        2
4         9    10          1             0        2

                 device_fingerprint  … card_type_Platinum Credit  \
0  ce7db67743bf5fffdfa5b2ecfb4e6de0  …                         0
1  d654a7f29023652d55a4574b8fc6ea4a  …                         1
2  6d945d3387a7d9eb150ec7d30bd3d621  …                         0
3  4af6ed569634c7b9311dc6968b50e29e  …                         0
4  fcaf3544fe07e389572c90dc02921857  …                         0

   card_type_Premium Debit  device_Chip Reader  device_Chrome  device_Edge  \
0                        1                   0              0            0
1                        0                   0              1            0
2                        0                   0              0            0
3                        1                   0              0            1
```

```
4                        0                   0              0             0

   device_Firefox  device_Magnetic Stripe  device_NFC Payment  device_Safari  \
0               1                       0                   0              0
1               0                       0                   0              0
2               0                       0                   0              1
3               0                       0                   0              0
4               1                       0                   0              0

   device_iOS App
0               0
1               0
2               0
3               0
4               0

[5 rows x 51 columns]
```

[14]:
```
null_counts = sample.isnull().sum()

if null_counts.any():

    print("\nColumns with null values:")

    print(null_counts[null_counts > 0])

else:

    print("\nNo null values found in the dataset")
```

```
No null values found in the dataset
```

[15]:
```
sample.select_dtypes(include=['int32'])
```

[15]:
```
        card_present  high_risk_merchant  weekend_transaction  is_fraud  \
0                  0                   1                    0         0
1                  0                   0                    0         0
2                  0                   1                    0         0
3                  0                   0                    0         1
4                  0                   0                    0         0
…                …                 …                   …       …
249995             0                   0                    0         0
249996             0                   0                    0         1
249997             0                   0                    0         0
249998             0                   0                    0         1
249999             0                   0                    0         0
```

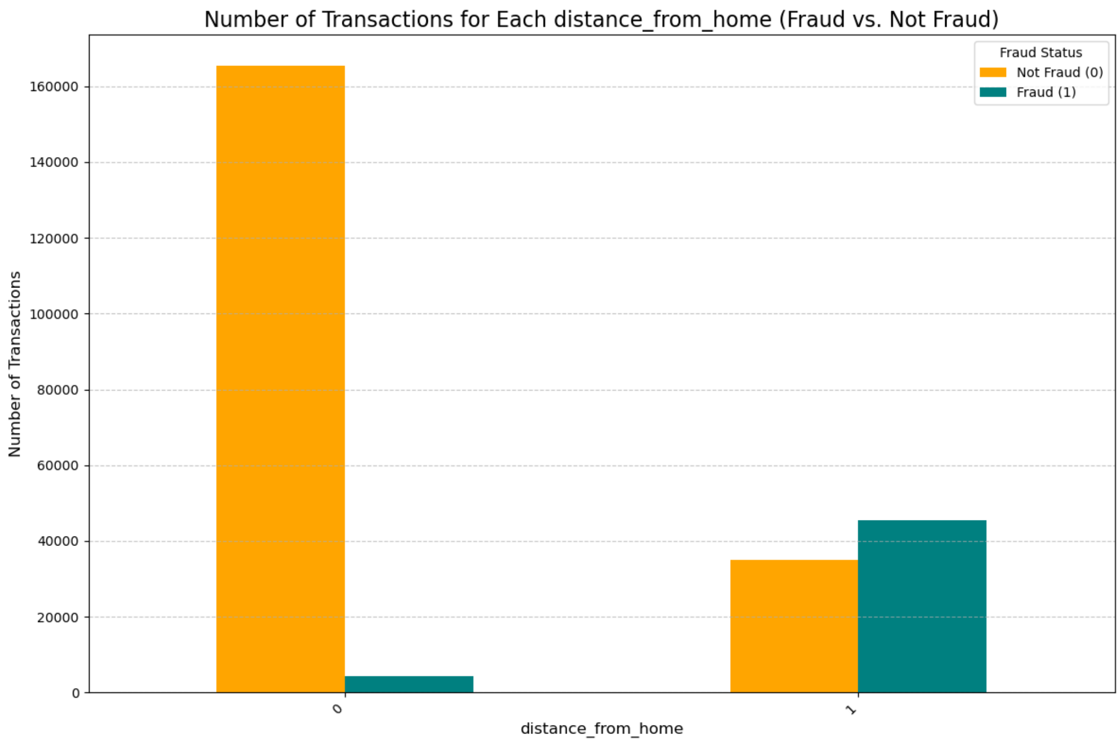
```
        merchant_category_Entertainment  merchant_category_Gas  \
0                                     1                      0
1                                     0                      0
2                                     1                      0
3                                     0                      0
4                                     0                      0
…                                   …                    …
249995                                0                      0
249996                                0                      0
249997                                0                      0
249998                                0                      0
249999                                0                      0

        merchant_category_Grocery  merchant_category_Healthcare  \
0                               0                             0
1                               0                             0
2                               0                             0
3                               0                             1
4                               0                             0
…                             …                           …
249995                          0                             0
249996                          1                             0
249997                          0                             0
249998                          0                             0
249999                          0                             0

        merchant_category_Restaurant  merchant_category_Retail  …  \
0                                  0                         0  …
1                                  1                         0  …
2                                  0                         0  …
3                                  0                         0  …
4                                  1                         0  …
…                                …                       …  …
249995                             1                         0  …
249996                             0                         0  …
249997                             0                         0  …
249998                             0                         1  …
249999                             0                         0  …

        card_type_Platinum Credit  card_type_Premium Debit  \
0                               0                        1
1                               1                        0
2                               0                        0
3                               0                        1
4                               0                        0
…                             …                      …
249995                          1                        0
```

```
249996                        0                         1
249997                        0                         0
249998                        0                         0
249999                        0                         0

        device_Chip Reader  device_Chrome  device_Edge  device_Firefox  \
0                        0              0            0               1
1                        0              1            0               0
2                        0              0            0               0
3                        0              0            1               0
4                        0              0            0               1
…                      …             …          …             …
249995                   0              0            0               1
249996                   0              1            0               0
249997                   0              0            0               0
249998                   0              0            0               0
249999                   0              0            0               0

        device_Magnetic Stripe  device_NFC Payment  device_Safari  \
0                            0                   0              0
1                            0                   0              0
2                            0                   0              1
3                            0                   0              0
4                            0                   0              0
…                          …                 …            …
249995                       0                   0              0
249996                       0                   0              0
249997                       0                   0              0
249998                       0                   0              0
249999                       0                   0              0

        device_iOS App
0                    0
1                    0
2                    0
3                    0
4                    0
…                  …
249995               0
249996               0
249997               1
249998               1
249999               0

[250000 rows x 34 columns]
```

```
[16]: correlation = sample['distance_from_home'].corr(sample['is_fraud'])
print(f"Correlation between 'your_column' and 'is_fraud': {correlation}")
```

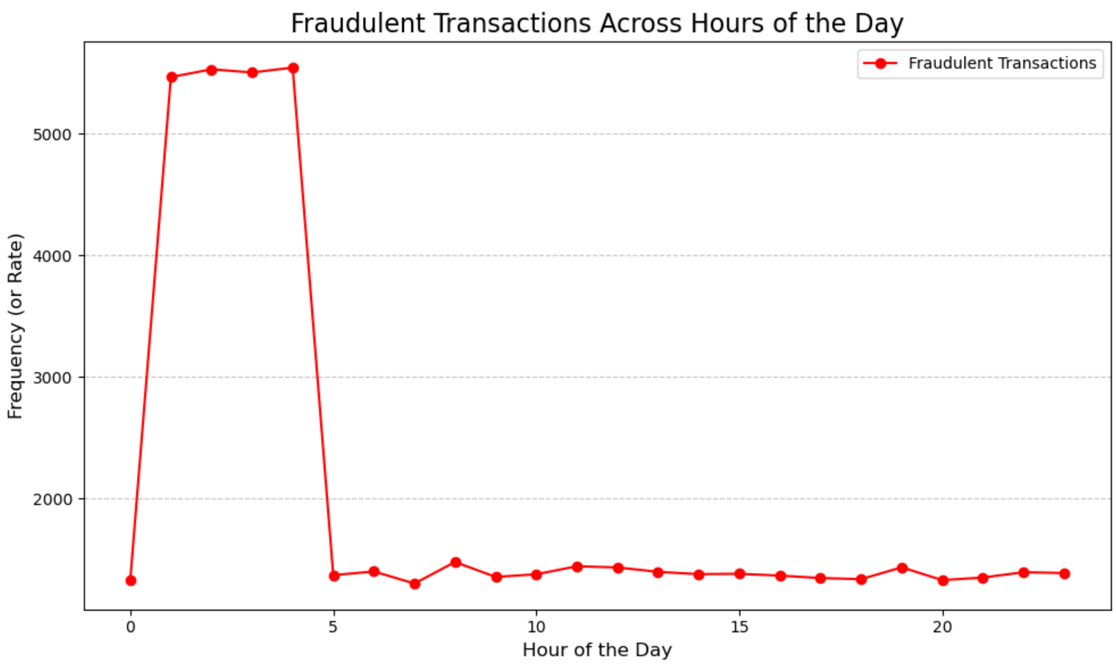
```
Correlation between 'your_column' and 'is_fraud': 0.6315171194299821
```

```
[17]: category_column = 'distance_from_home'

grouped_data = sample.groupby([category_column, 'is_fraud']).size().
 ↪reset_index(name='count')

pivot_data = grouped_data.pivot(index=category_column, columns='is_fraud',␣
 ↪values='count').fillna(0)

pivot_data.columns = ['Not Fraud (0)', 'Fraud (1)']

pivot_data.plot(kind='bar', stacked=False, figsize=(12, 8), color=['orange',␣
 ↪'teal'])

plt.title(f'Number of Transactions for Each {category_column} (Fraud vs. Not␣
 ↪Fraud)', fontsize=16)
plt.xlabel(category_column, fontsize=12)
plt.ylabel('Number of Transactions', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.legend(title='Fraud Status')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

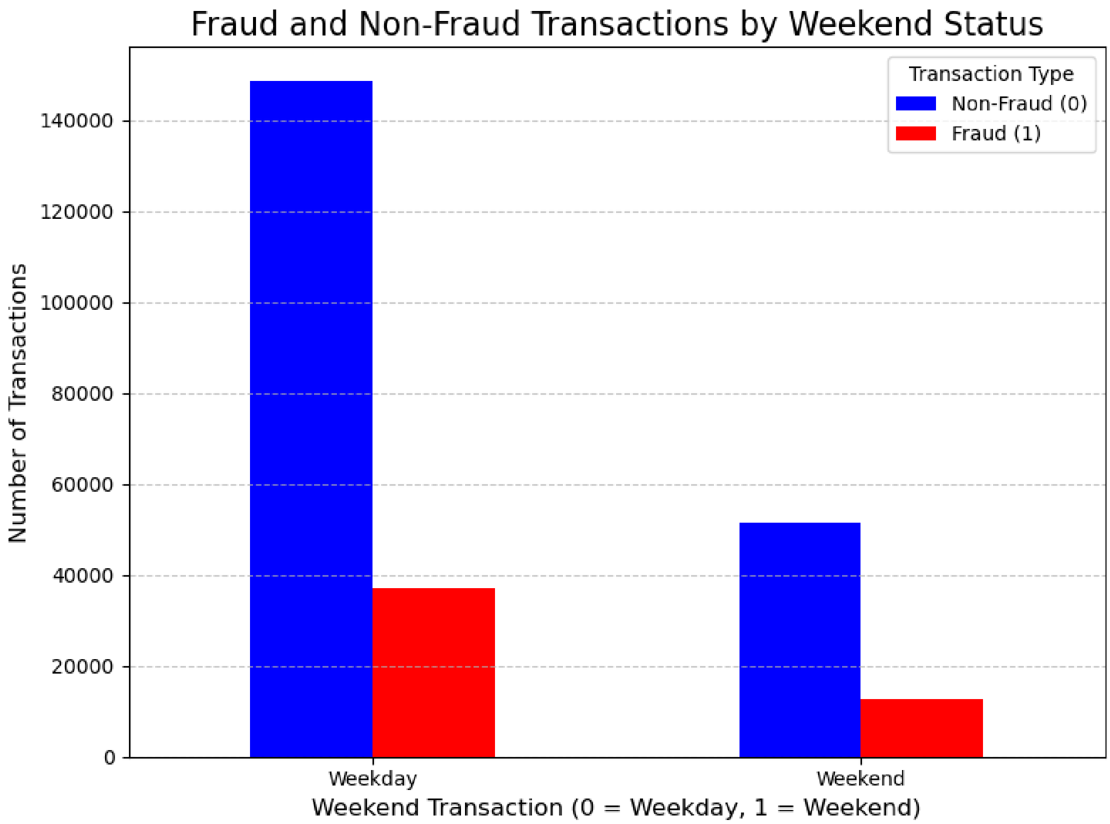


As you can see from the graph the farther you are from home the more likely the transaction is fraud

[18]:
```
fraud_by_hour = sample[sample['is_fraud'] == 1].groupby('transaction_hour').
    ↪size()

total_by_hour = sample.groupby('transaction_hour').size()

plt.figure(figsize=(10, 6))
fraud_by_hour.plot(kind='line', marker='o', color='red', label='Fraudulent␣
    ↪Transactions')

plt.title('Fraudulent Transactions Across Hours of the Day', fontsize=16)
plt.xlabel('Hour of the Day', fontsize=12)
plt.ylabel('Frequency (or Rate)', fontsize=12)
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



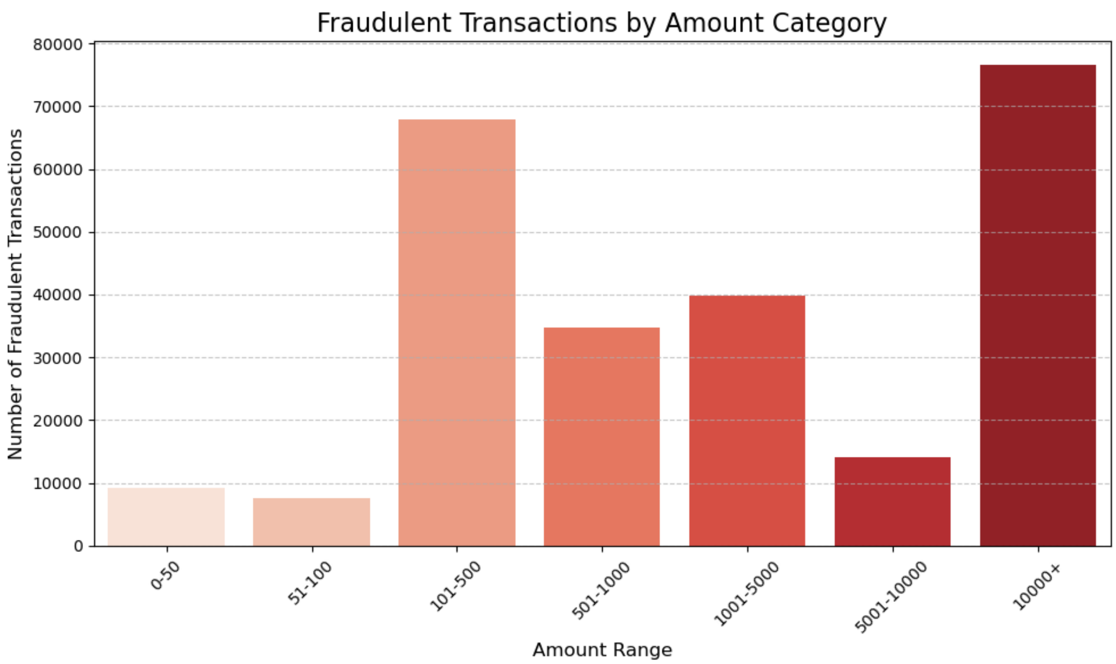
[19]:
```
weekend_fraud = sample.groupby(['weekend_transaction', 'is_fraud']).size().
↪reset_index(name='count')

weekend_pivot = weekend_fraud.pivot(index='weekend_transaction',␣
↪columns='is_fraud', values='count').fillna(0)
weekend_pivot.columns = ['Non-Fraud (0)', 'Fraud (1)']

weekend_pivot.plot(kind='bar', figsize=(8, 6), color=['blue', 'red'])

plt.title('Fraud and Non-Fraud Transactions by Weekend Status', fontsize=16)
plt.xlabel('Weekend Transaction (0 = Weekday, 1 = Weekend)', fontsize=12)
plt.ylabel('Number of Transactions', fontsize=12)
plt.xticks(ticks=[0, 1], labels=['Weekday', 'Weekend'], rotation=0)
plt.legend(title='Transaction Type')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



[20]:
```python
bins = [0, 50, 100, 500, 1000, 5000, 10000, float('inf')]
labels = ['0-50', '51-100', '101-500', '501-1000', '1001-5000', '5001-10000',␣
↪'10000+']

# Create a new column 'amount_category' based on the bins
sample['amount_category'] = pd.cut(sample['amount'], bins=bins, labels=labels)

# Count the number of fraudulent transactions in each category
fraud_amount_counts = sample['amount_category'].value_counts()

# Plot a bar graph for fraud transactions by amount category
plt.figure(figsize=(10, 6))
sns.barplot(x=fraud_amount_counts.index, y=fraud_amount_counts.values,␣
↪palette='Reds')

# Add titles and labels
plt.title('Fraudulent Transactions by Amount Category', fontsize=16)
plt.xlabel('Amount Range', fontsize=12)
plt.ylabel('Number of Fraudulent Transactions', fontsize=12)
plt.xticks(rotation=45)
```

```
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\732006053.py:12:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=fraud_amount_counts.index, y=fraud_amount_counts.values,
palette='Reds')
```



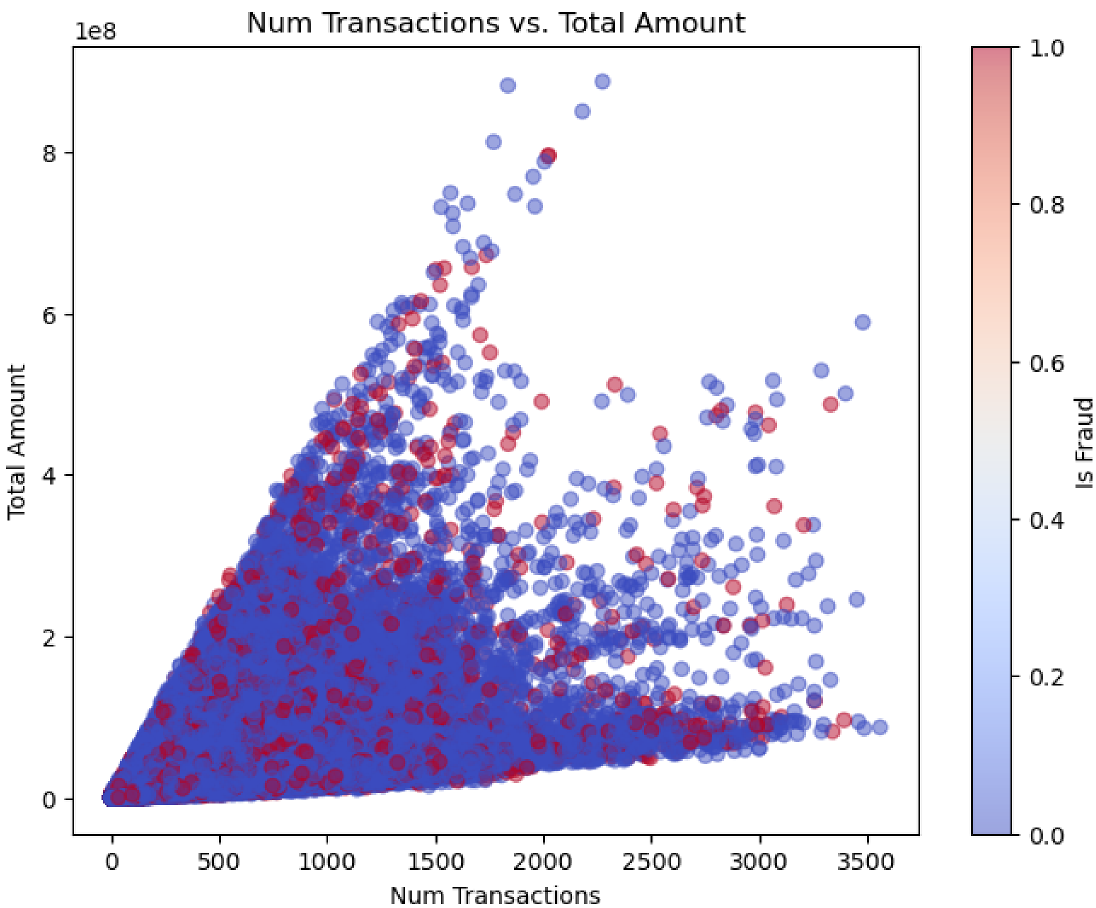
[21]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

numerical_features = [
    'num_transactions', 'total_amount', 'unique_merchants',
    'unique_countries', 'max_single_amount'
]

# Calculate VIF
X = sample[numerical_features]
X = X.fillna(0)  # Handle missing values (if any)

vif_data = pd.DataFrame()
```

```
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
 ↪shape[1])]

print(vif_data)

sample['transaction_diversity'] = sample['unique_merchants'] /␣
 ↪(sample['num_transactions'] + 1)

numerical_features = [
    'num_transactions', 'total_amount',
    'transaction_diversity', 'max_single_amount'
]

# Calculate VIF
X = sample[numerical_features]
X = X.fillna(0)  # Handle missing values (if any)

vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
 ↪shape[1])]

print(vif_data)
```

```
             Feature        VIF
0   num_transactions   5.443724
1       total_amount   1.933122
2   unique_merchants  49.864462
3   unique_countries  36.337352
4  max_single_amount   3.854857
                 Feature       VIF
0       num_transactions  3.289582
1           total_amount  1.918665
2  transaction_diversity  1.304706
3      max_single_amount  3.163730
```

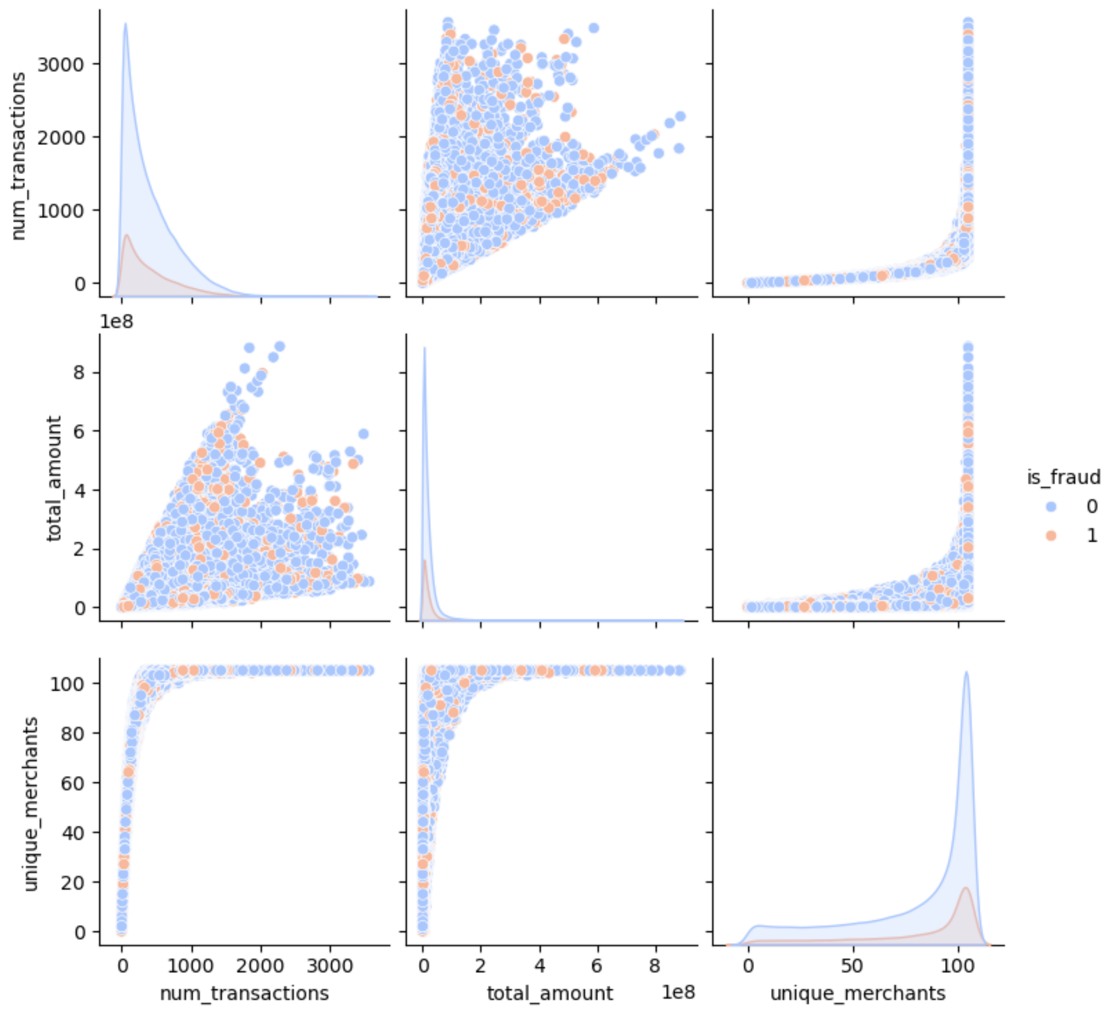
[22]:

```
plt.figure(figsize=(8, 6))
plt.scatter(sample['num_transactions'], sample['total_amount'], alpha=0.5,␣
 ↪c=sample['is_fraud'], cmap='coolwarm')
plt.title('Num Transactions vs. Total Amount')
plt.xlabel('Num Transactions')
plt.ylabel('Total Amount')
plt.colorbar(label='Is Fraud')
plt.show()
```



[23]:

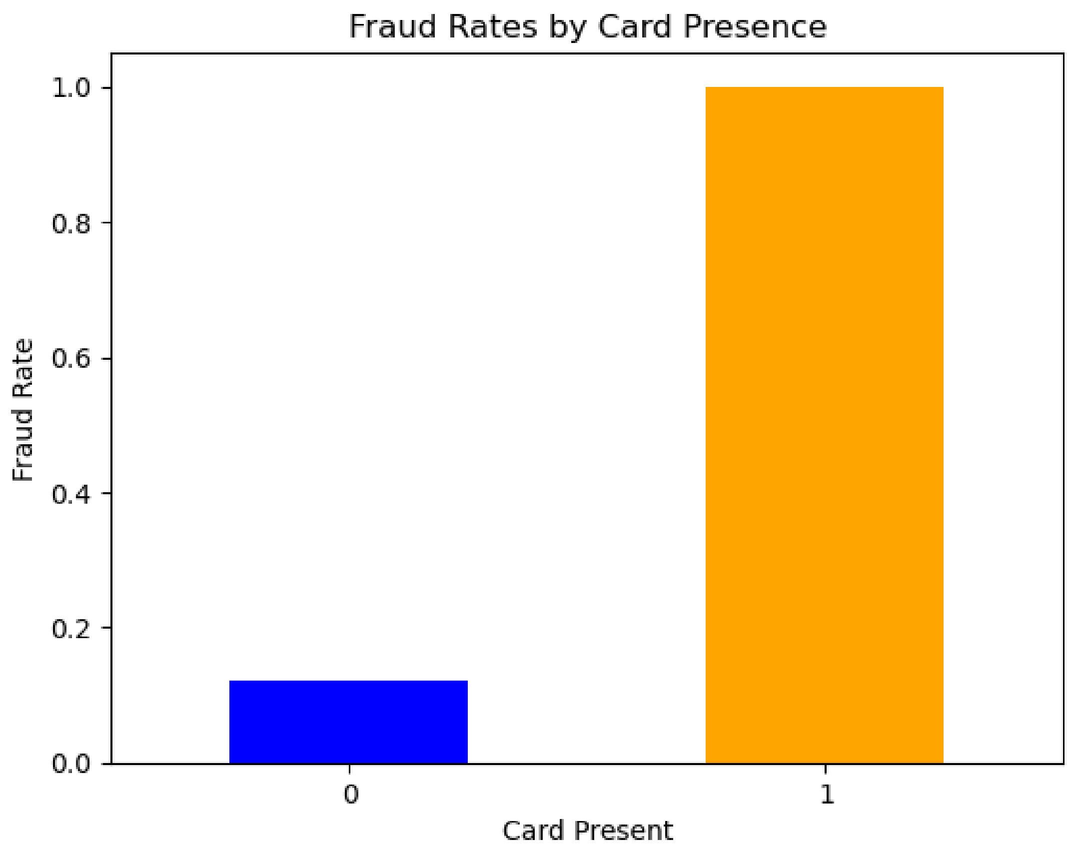
```
sns.pairplot(
    sample[['num_transactions', 'total_amount', 'unique_merchants',␣
 ↪'is_fraud']],
    hue='is_fraud',
    palette='coolwarm',
    diag_kind='kde'
)
plt.show()
```



[24]:
```
fraud_rates_card_present = sample.groupby('card_present')['is_fraud'].mean()
print(fraud_rates_card_present)
```

```
card_present
0    0.122648
1    1.000000
Name: is_fraud, dtype: float64
```

[25]:
```
fraud_rates_card_present.plot(kind='bar', color=['blue', 'orange'])
plt.title('Fraud Rates by Card Presence')
plt.ylabel('Fraud Rate')
plt.xlabel('Card Present')
plt.xticks(rotation=0)
plt.show()
```



[26]:

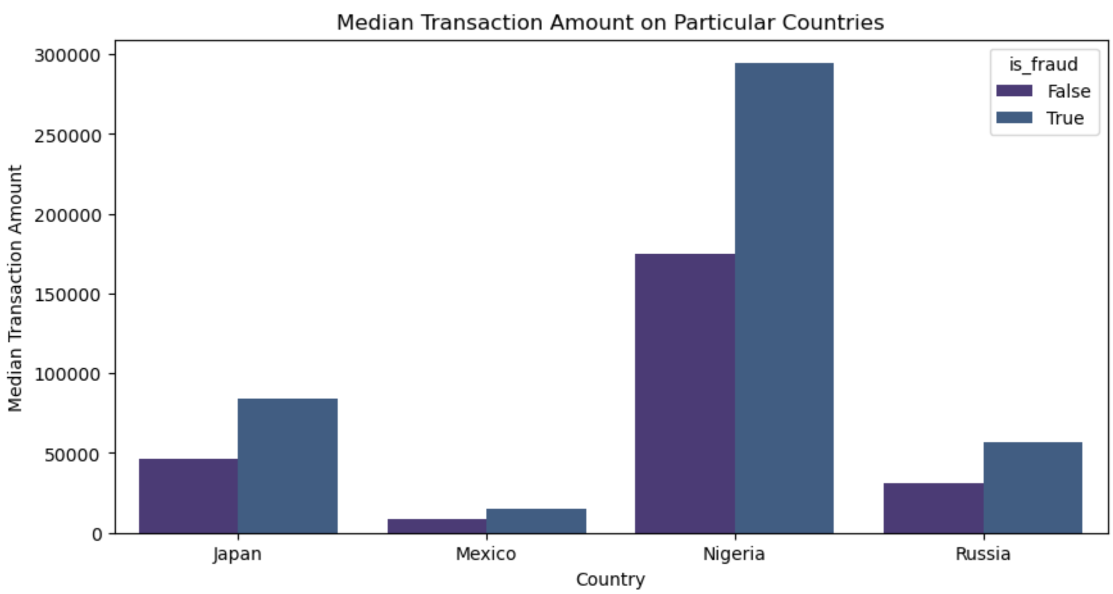
```python
fraud_country = pd_categorical[pd_categorical['is_fraud'] == 1]
fraud_country_grouped = fraud_country.groupby(['country',
 ↪'is_fraud'])['amount'].median().reset_index()
ranked_country = fraud_country_grouped.sort_values(by='amount',
 ↪ascending=False).reset_index(drop=True)
ranked_country
```

[26]:

	country	is_fraud	amount
0	Nigeria	True	294516.610
1	Japan	True	83717.120
2	Russia	True	56978.935
3	Mexico	True	14976.800
4	Brazil	True	3811.930
5	Australia	True	964.510
6	Canada	True	954.245
7	Singapore	True	937.950
8	USA	True	707.840
9	Germany	True	690.450
10	France	True	665.640
11	UK	True	531.210

[27]:
```
country_specific = pd_categorical[pd_categorical['country'].isin(['Russia',␣
↪'Nigeria', 'Japan', 'Mexico'])]
country_vis = country_specific.groupby(['country','is_fraud'])['amount'].
↪agg('median').reset_index()
country_vis

plt.figure(figsize=(10,5))
sns.barplot(data = country_vis, x = 'country', y = 'amount', hue = 'is_fraud',␣
↪palette = sns.color_palette("viridis"))
plt.title("Median Transaction Amount on Particular Countries")
plt.xlabel('Country')
plt.ylabel('Median Transaction Amount')
plt.show()
```

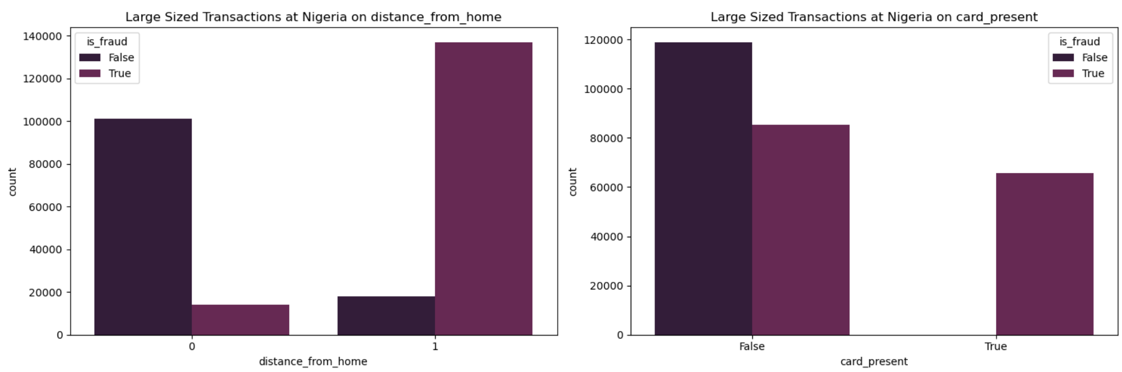
```
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\675984055.py:6: UserWarning:
The palette list has more values (6) than needed (2), which may not be intended.
  sns.barplot(data = country_vis, x = 'country', y = 'amount', hue = 'is_fraud',
palette = sns.color_palette("viridis"))
```



[28]:
```
nigeria_high_amount = df[(df['country'] == 'Nigeria') & (df['amount'] > 294516)]
print(f'number of abnormal transactions in Nigeria :{nigeria_high_amount.
↪shape[0]}')

cols = ['distance_from_home', 'card_present']

fig, axes = plt.subplots(1,2, figsize = (15,5))
for i, col in enumerate(cols):
```

```
    tmp_data = nigeria_high_amount[[col, 'is_fraud']].value_counts().
↪reset_index()
    sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud',
    palette = sns.color_palette('rocket'), ax = axes[i])
    axes[i].set_title(f'Large Sized Transactions at Nigeria on {col}')

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

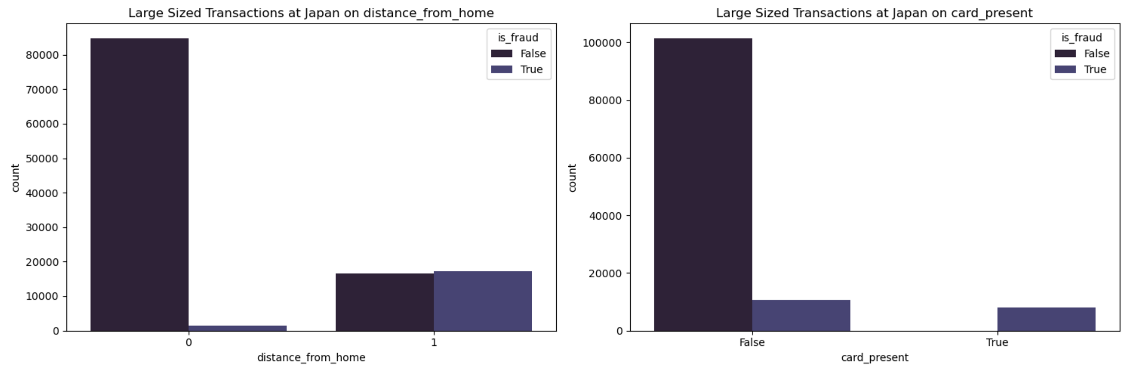
```
number of abnormal transactions in Nigeria :270139

C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\2165583713.py:9: UserWarning:
The palette list has more values (6) than needed (2), which may not be intended.
  sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud',
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\2165583713.py:9: UserWarning:
The palette list has more values (6) than needed (2), which may not be intended.
  sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud',
```



[29]:
```
japan_high_amount = df[(df['country'] == 'Japan') & (df['amount'] > 83717)]
print(f'number of abnormal transactions in Japan : {japan_high_amount.
↪shape[0]}')

cols = ['distance_from_home', 'card_present']

fig, axes = plt.subplots(1,2, figsize = (15,5))
for i, col in enumerate(cols):
    tmp_data = japan_high_amount[[col, 'is_fraud']].value_counts().reset_index()
    sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud', palette = sns.
↪color_palette('mako'), ax = axes[i])
    axes[i].set_title(f'Large Sized Transactions at Japan on {col}')

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

```
number of abnormal transactions in Japan : 120193
```

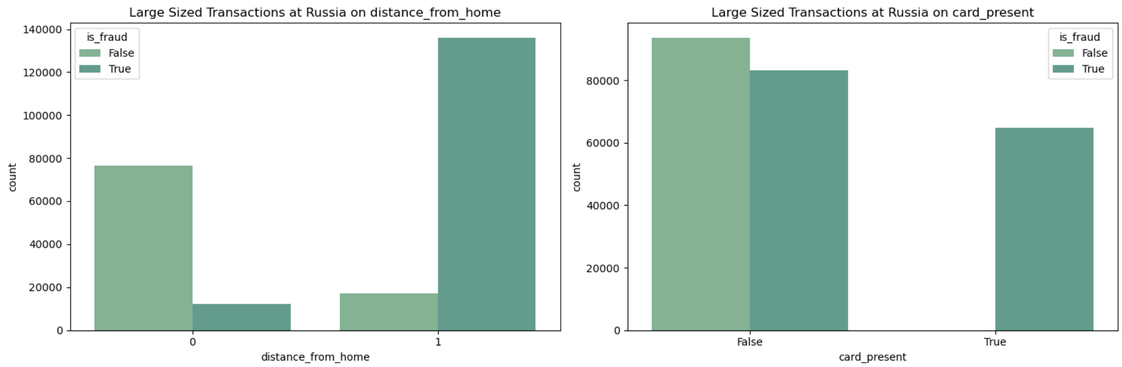
```
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\3783204104.py:9: UserWarning:
The palette list has more values (6) than needed (2), which may not be intended.
  sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud', palette =
sns.color_palette('mako'), ax = axes[i])
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\3783204104.py:9: UserWarning:
The palette list has more values (6) than needed (2), which may not be intended.
  sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud', palette =
sns.color_palette('mako'), ax = axes[i])
```



[30]:
```
russia_high_amount = df[(df['country'] == 'Russia') & (df['amount'] > 56978)]
print(f'number of abnormal transactions in Russia : {russia_high_amount.
 ↪shape[0]}')

cols = ['distance_from_home', 'card_present']

fig, axes = plt.subplots(1,2, figsize = (15,5))
for i, col in enumerate(cols):
    tmp_data = russia_high_amount[[col,'is_fraud']].value_counts().reset_index()
    sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud', palette = sns.
 ↪color_palette('crest'), ax = axes[i])
    axes[i].set_title(f'Large Sized Transactions at Russia on {col}')

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

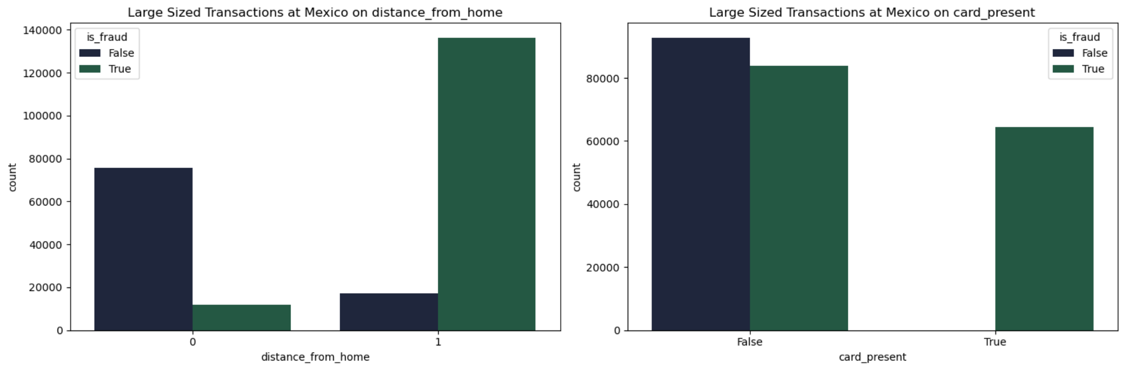
```
number of abnormal transactions in Russia : 241867
```

```
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\2560148670.py:9: UserWarning:
The palette list has more values (6) than needed (2), which may not be intended.
  sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud', palette =
sns.color_palette('crest'), ax = axes[i])
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\2560148670.py:9: UserWarning:
The palette list has more values (6) than needed (2), which may not be intended.
  sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud', palette =
sns.color_palette('crest'), ax = axes[i])
```



[31]:
```
mexico_high_amount = df[(df['country'] == 'Mexico') & (df['amount'] > 14976)]
print(f'number of abnormal transactions in Mexico : {mexico_high_amount.
  ↪shape[0]}')

cols = ['distance_from_home', 'card_present']

fig, axes = plt.subplots(1,2, figsize = (15,5))
for i, col in enumerate(cols):
    tmp_data = mexico_high_amount[[col, 'is_fraud']].value_counts().
  ↪reset_index()
    sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud', palette = sns.
  ↪color_palette('cubehelix'), ax = axes[i])
    axes[i].set_title(f'Large Sized Transactions at Mexico on {col}')

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

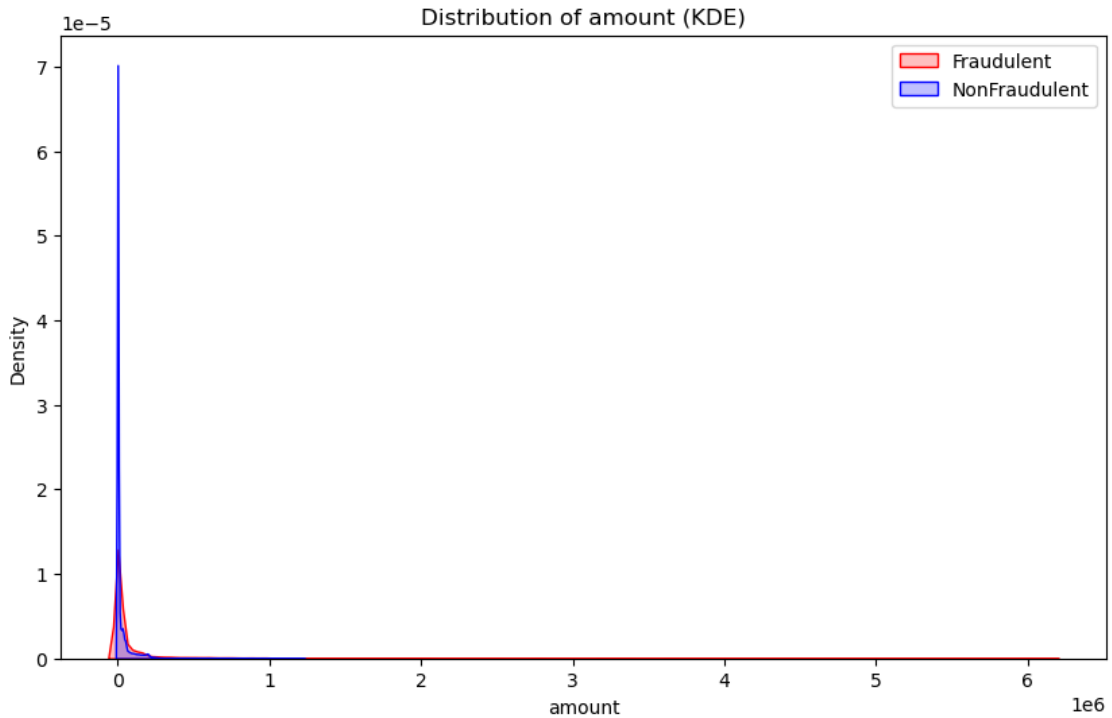
```
number of abnormal transactions in Mexico : 241002

C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\3976898031.py:9: UserWarning:
The palette list has more values (6) than needed (2), which may not be intended.
  sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud', palette =
sns.color_palette('cubehelix'), ax = axes[i])
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\3976898031.py:9: UserWarning:
The palette list has more values (6) than needed (2), which may not be intended.
  sns.barplot(tmp_data, x = col, y = 'count', hue = 'is_fraud', palette =
sns.color_palette('cubehelix'), ax = axes[i])
```



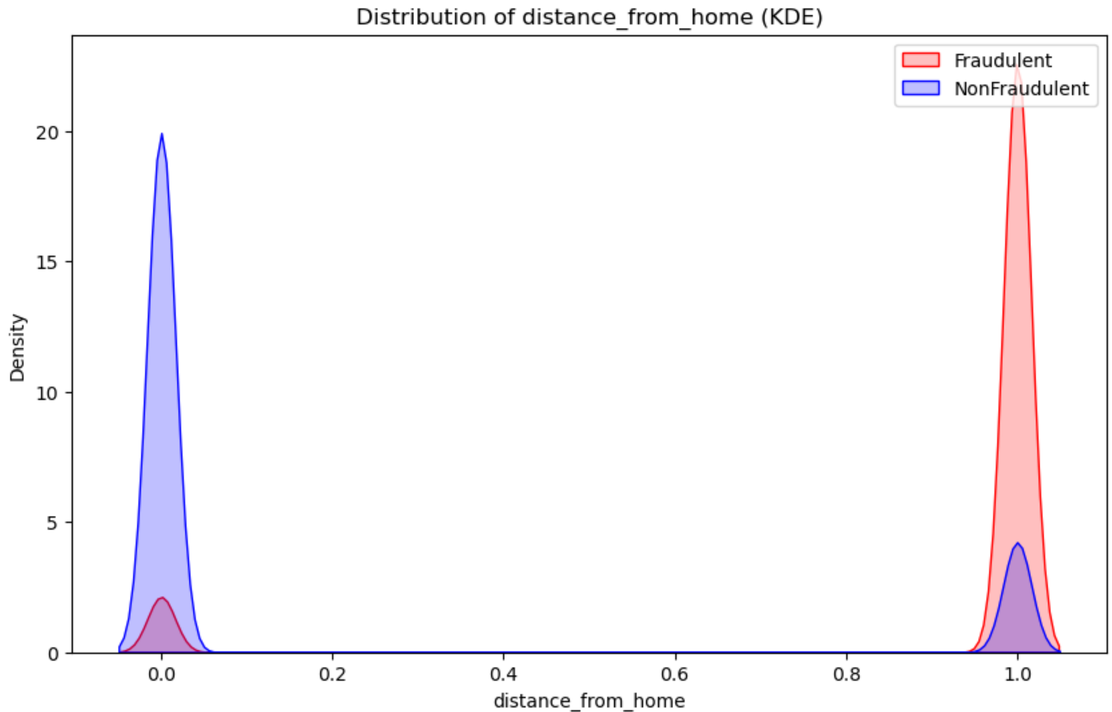

```
[32]: fraudulent = sample[sample['is_fraud'] == 1]
      non_fraudulent = sample[sample['is_fraud'] == 0]
```

```
[33]: features = ['amount', 'distance_from_home', 'max_single_amount']

      for feature in features:
          plt.figure(figsize=(10, 6))
          sns.kdeplot(fraudulent[feature], shade=True, label='Fraudulent', color='red', bw_adjust=0.5)
          sns.kdeplot(non_fraudulent[feature], shade=True, label='NonFraudulent', color='blue', bw_adjust=0.5)
          plt.title(f'Distribution of {feature} (KDE)')
          plt.xlabel(feature)
          plt.ylabel('Density')
          plt.legend()
          plt.show()
```

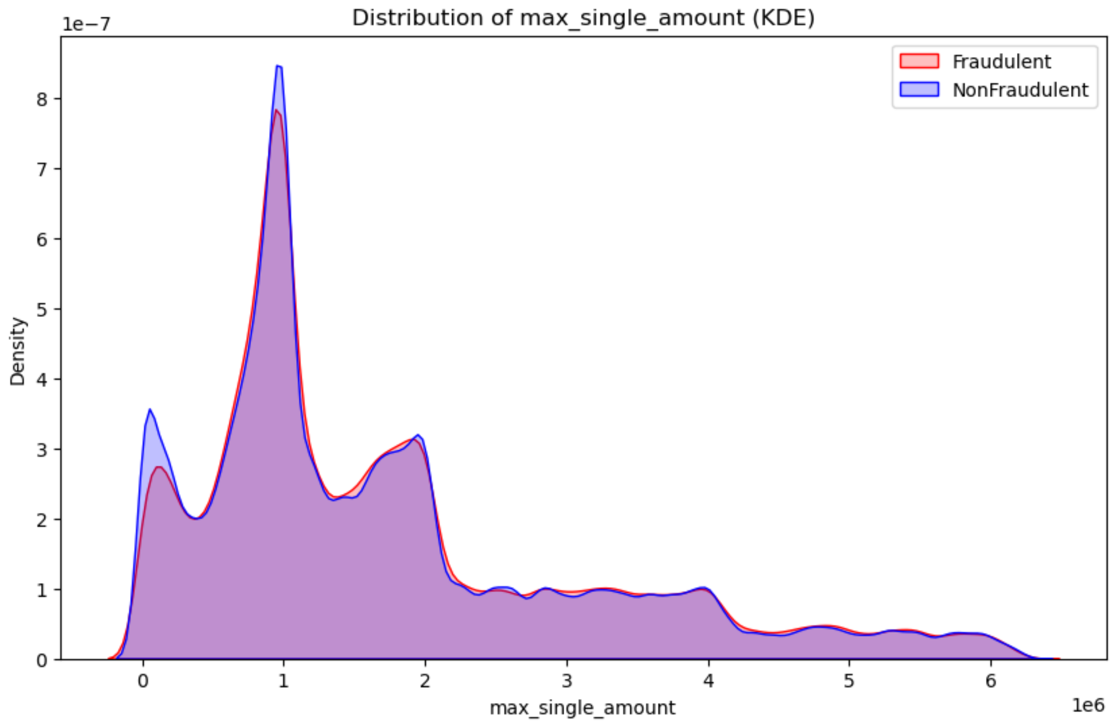
```
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\2471544914.py:5:
FutureWarning:

`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(fraudulent[feature], shade=True, label='Fraudulent', color='red',
bw_adjust=0.5)
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\2471544914.py:6:
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(non_fraudulent[feature], shade=True, label='NonFraudulent',
color='blue', bw_adjust=0.5)
```



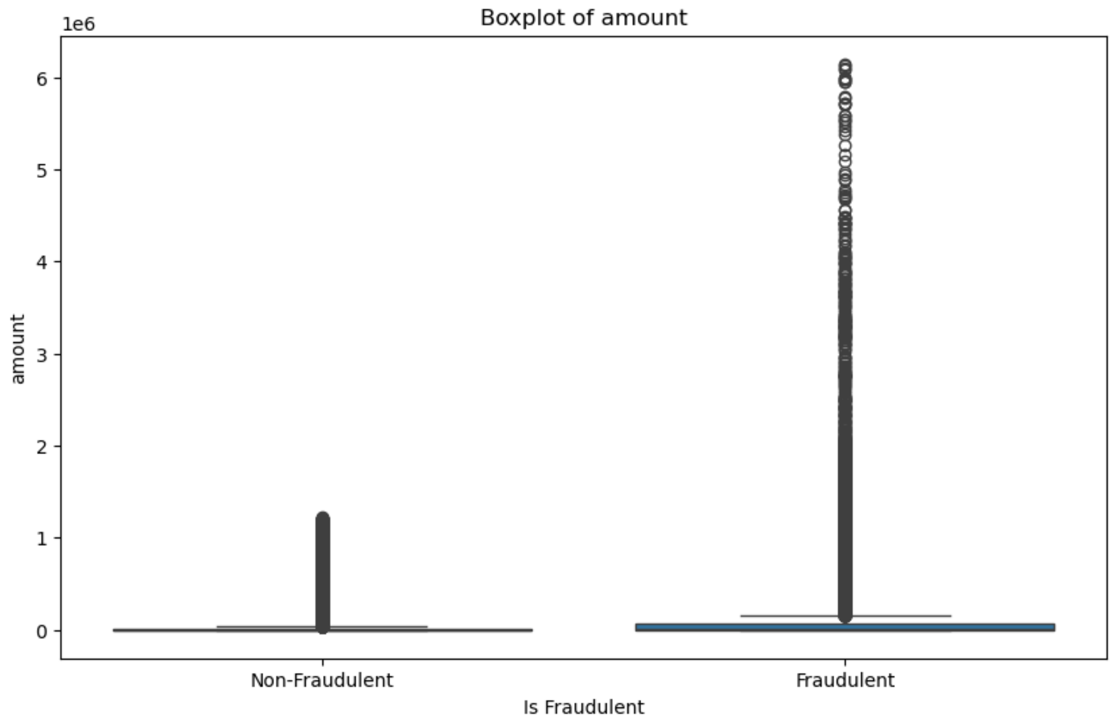

```
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\2471544914.py:5:
FutureWarning:

`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(fraudulent[feature], shade=True, label='Fraudulent', color='red',
bw_adjust=0.5)
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\2471544914.py:6:
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(non_fraudulent[feature], shade=True, label='NonFraudulent',
color='blue', bw_adjust=0.5)
```

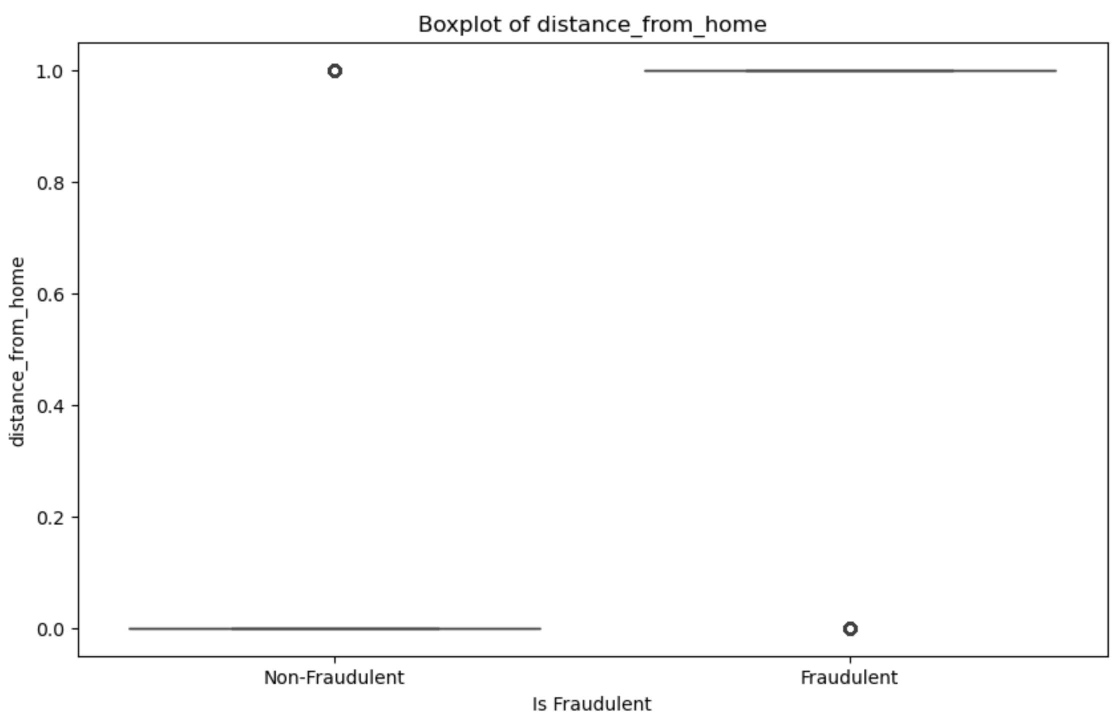


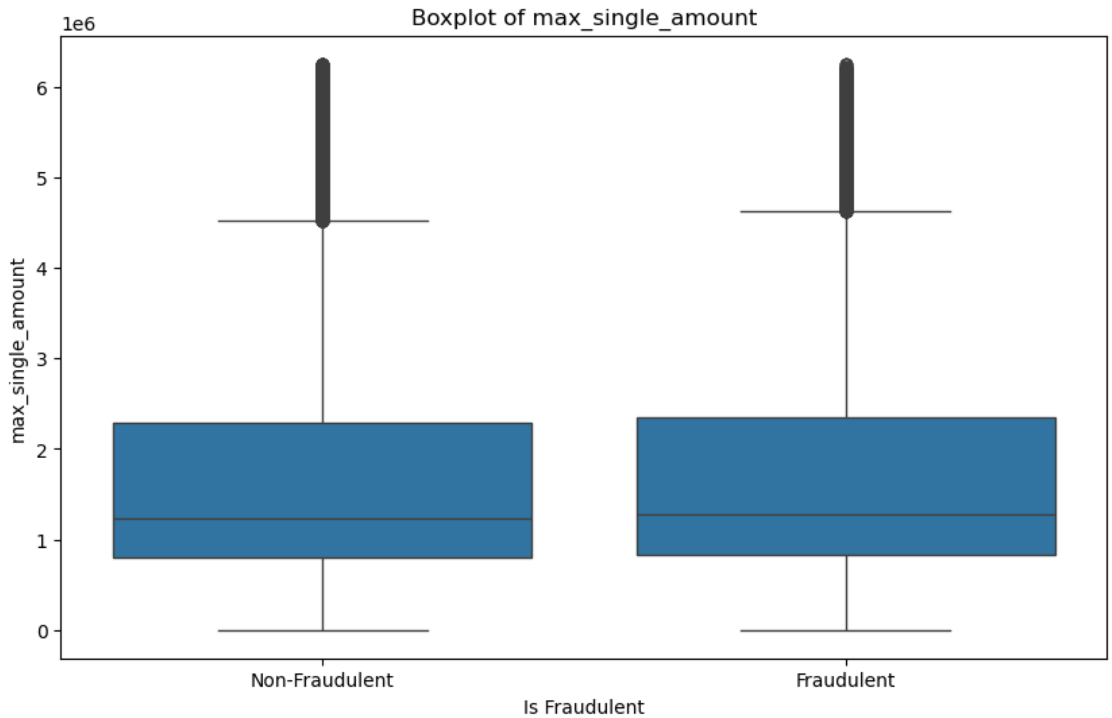

```
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\2471544914.py:5:
FutureWarning:

`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(fraudulent[feature], shade=True, label='Fraudulent', color='red',
bw_adjust=0.5)
C:\Users\skiki\AppData\Local\Temp\ipykernel_26764\2471544914.py:6:
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(non_fraudulent[feature], shade=True, label='NonFraudulent',
color='blue', bw_adjust=0.5)
```



[34]:
```python
for feature in features:
    plt.figure(figsize=(10, 6))
    sns.boxplot(data= sample, x='is_fraud', y=feature)
    plt.title(f'Boxplot of {feature}')
    plt.xlabel('Is Fraudulent')
    plt.ylabel(feature)
    plt.xticks([0, 1], ['Non-Fraudulent', 'Fraudulent'])
    plt.show()
```


Boxplot of amount
1e6
6
5
4
3
2
1
0
amount
Non-Fraudulent
Fraudulent
Is Fraudulent


Boxplot of distance_from_home
1.0
0.8
0.6
0.4
0.2
0.0
distance_from_home
Non-Fraudulent
Fraudulent
Is Fraudulent



[35]:
```
sample = sample.drop(columns=['unique_merchants', 'num_transactions',␣
↪'timestamp', 'device_fingerprint', 'ip_address', 'amount_category'])
```

[37]:
```
sample.columns
```

[37]:
```
Index(['merchant_type', 'merchant', 'amount', 'currency', 'city', 'city_size',
       'card_present', 'channel', 'distance_from_home', 'high_risk_merchant',
       'transaction_hour', 'weekend_transaction', 'is_fraud', 'total_amount',
       'unique_countries', 'max_single_amount',
       'merchant_category_Entertainment', 'merchant_category_Gas',
       'merchant_category_Grocery', 'merchant_category_Healthcare',
       'merchant_category_Restaurant', 'merchant_category_Retail',
       'merchant_category_Travel', 'country_Brazil', 'country_Canada',
       'country_France', 'country_Germany', 'country_Japan', 'country_Mexico',
       'country_Nigeria', 'country_Russia', 'country_Singapore', 'country_UK',
       'country_USA', 'card_type_Basic Debit', 'card_type_Gold Credit',
       'card_type_Platinum Credit', 'card_type_Premium Debit',
       'device_Chip Reader', 'device_Chrome', 'device_Edge', 'device_Firefox',
       'device_Magnetic Stripe', 'device_NFC Payment', 'device_Safari',
       'device_iOS App', 'transaction_diversity'],
      dtype='object')
```

[38]:
```
selected_features = ['amount', 'merchant_type', 'currency', 'card_present',␣
↪'transaction_diversity', 'distance_from_home', 'merchant',␣
↪'transaction_hour', 'country_Nigeria', 'country_Japan', 'country_Russia',␣
↪'country_Mexico' ]

X = sample.drop(columns=['is_fraud'])
y = sample['is_fraud']

X = X[selected_features]
```

[39]:
```
from sklearn.model_selection import train_test_split

X_train, X_test , y_train, y_test = train_test_split(X,y, test_size = 0.
↪2,random_state = 42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[39]:
```
((200000, 12), (50000, 12), (200000,), (50000,))
```

[40]:
```
# Apply SMOTE
smote = SMOTE(random_state=42, sampling_strategy=0.9)

# Generate new samples for the training set
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Check the new class distribution
print(f'Percentage of Fraudulent Transaction : {y_train_smote.
↪value_counts(normalize = True)[0] * 100}%')
print(f'Percentage of Normal Transaction : {y_train_smote.
↪value_counts(normalize = True)[1] * 100}%')
```

```
Percentage of Fraudulent Transaction : 52.63171730315517%
Percentage of Normal Transaction : 47.36828269684483%
```

[41]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_smote = scaler.fit_transform(X_train_smote)
X_test = scaler.transform(X_test)
```

1.6 Machine Learning Approaches

<Complete for **Project Progress**>

- We will use a logistic regression model as our baseline for this model. Logistic regression is quick and easy to implement and works well with binary classification.
- We are considering to use gradient boosting or this project as it works well with tabular data and can caputure complex patterns and interaction between features.

- We have decided to use tree-based models such as gradient boosting and random forest since they work very well with tabular data.

<Expand and complete for **Project Submission**>

- Describe the methods/datasets (you can have unscaled, selected, scaled version, multiple data farmes) that you ended up using for modeling.
- Justify the selection of machine learning tools you have used
  - How they informed the next steps?
- Make sure to include at least twp models: (1) baseline model, and (2) improvement model(s).
  - The baseline model is typically the simplest model that's applicable to that data problem, something we have learned in the class.
  - Improvement model(s) are available on Kaggle challenge site, and you can research github.com and papers with code for approaches.

[42]:
```
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression()
linear_model.fit(X_train_smote,y_train_smote)
```

[42]:
```
LinearRegression()
```

[43]:
```
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

y_pred = linear_model.predict(X_test)
mse = mean_squared_error(y_test,y_pred)

print("Test mean squared error (MSE): {:.2f}".format(mse))
print(linear_model.score(X_test,y_test))
```

```
Test mean squared error (MSE): 0.09
0.39933594031716646
```

[44]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

log_reg = LogisticRegression(max_iter=1000, solver='liblinear', random_state=42)

log_reg.fit(X_train_smote, y_train_smote)

# Make predictions
y_pred = log_reg.predict(X_test)

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```python
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

```
Confusion Matrix:
[[34686  5473]
 [  935  8906]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.86      0.92     40159
           1       0.62      0.90      0.74      9841

    accuracy                           0.87     50000
   macro avg       0.80      0.88      0.83     50000
weighted avg       0.90      0.87      0.88     50000


Accuracy Score:
0.87184
```

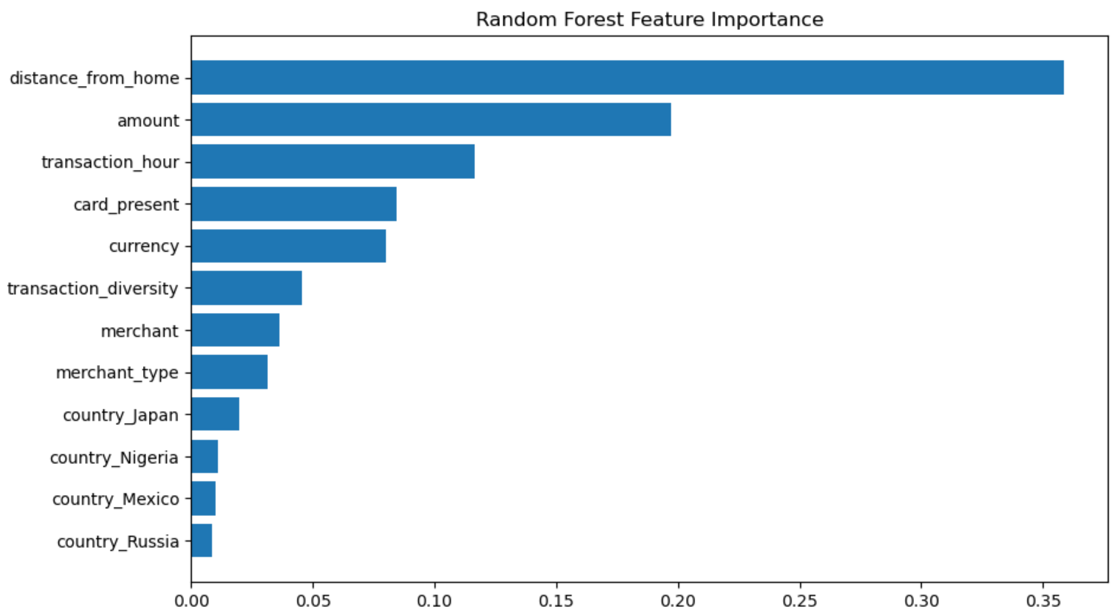
[45]:
```python
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the Random Forest model
rf_model = RandomForestClassifier(random_state=42, n_jobs=-1)
rf_model.fit(X_train_smote, y_train_smote)

y_pred = rf_model.predict(X_test)
y_pred_proba = rf_model.predict_proba(X_test)[:, 1]

# Evaluation
print("Classification Report:\n", classification_report(y_test, y_pred))
roc_auc = roc_auc_score(y_test, y_pred_proba)
print("ROC-AUC Score:", roc_auc)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.97      0.97     40159
           1       0.88      0.92      0.90      9841

    accuracy                           0.96     50000
   macro avg       0.93      0.95      0.94     50000
weighted avg       0.96      0.96      0.96     50000
```

```
ROC-AUC Score: 0.9878277465608906
```

[46]:

```
# Feature importance
importance = rf_model.feature_importances_
feature_importance = pd.DataFrame({'Feature': X.columns, 'Importance':␣
↪importance})
feature_importance = feature_importance.sort_values(by='Importance',␣
↪ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(feature_importance['Feature'], feature_importance['Importance'])
plt.title('Random Forest Feature Importance')
plt.gca().invert_yaxis()
plt.show()
```



[47]:

```
model = XGBClassifier()
model.fit(X_train_smote, y_train_smote)
```

[47]:

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
```

```
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, …)
```

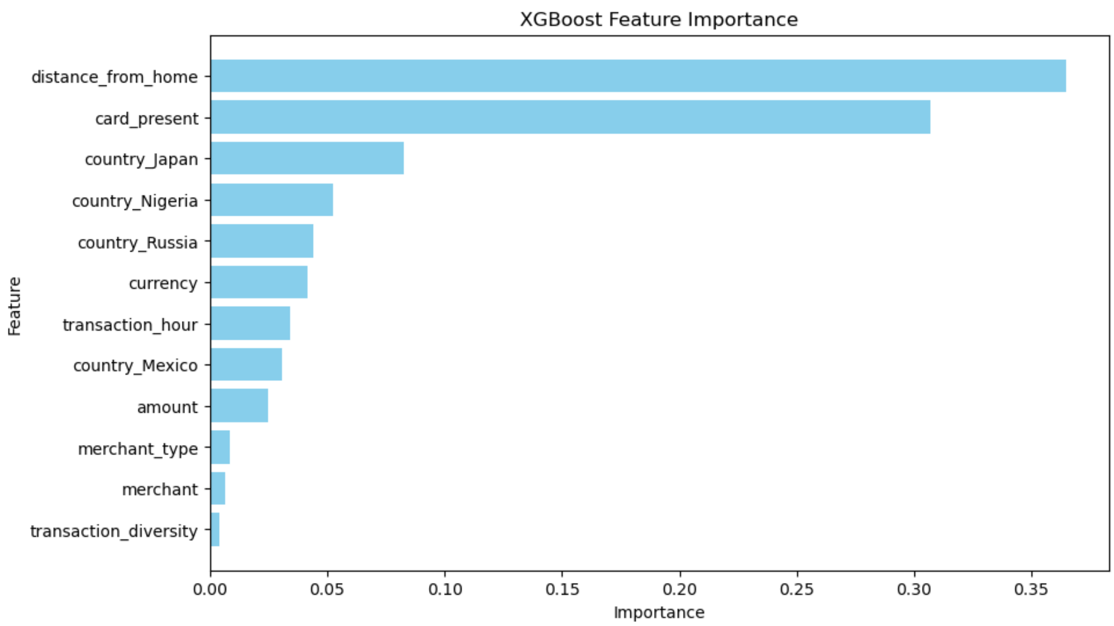
[48]:

```
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

y_prob = model.predict_proba(X_test)[:, 1]
print('ROC-AUC:', roc_auc_score(y_test, y_prob))
```

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.98     40159
           1       0.89      0.92      0.90      9841

    accuracy                           0.96     50000
   macro avg       0.93      0.95      0.94     50000
weighted avg       0.96      0.96      0.96     50000

ROC-AUC: 0.9900894693010991
```

[49]:

```
importance = model.feature_importances_

# Create DataFrame with feature names and their importances
feature_importance = pd.DataFrame({'Feature': X.columns, 'Importance':␣
 ↪importance})

# Sort the features by importance
feature_importance = feature_importance.sort_values(by='Importance',␣
 ↪ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(feature_importance['Feature'], feature_importance['Importance'],␣
 ↪color='skyblue')
plt.title('XGBoost Feature Importance')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.gca().invert_yaxis()  # Invert y-axis to have the most important feature at␣
 ↪the top
plt.show()
```



1.7 Experiments

< **Project Progress** should include experiments you have completed thus far.> * Thus far we have done correlation and visual analysis so far but plan on doing more experiment for over and undder sampling and the correct scaling for our data.

<**Project Submission** should only contain final version of the experiments. Please use visualizations whenever possible.> * Describe how did you evaluate your solution * What evaluation metrics did you use? * Describe a baseline model. * How much did your model outperform the baseline? * Were there other models evaluated on the same dataset(s)? * How did your model do in comparison to theirs? * Show graphs/tables with results * Present error analysis and suggestions for future improvement.