atory-data-analysis-classification

April 25, 2024

0.0.1 This notebook serves as a comprehensive guide to understanding and utilising the SAML-D dataset, specifically designed for enhancing anti-money laundering (AML) strategies. Initially, it delves into exploratory data analysis, providing a thorough examination of the dataset's characteristics. Following the EDA, there is a preprocessing phase, ensuring the data is optimised for modeling. Then an experiment employing the XGBoost classifier is conducted, a powerful machine learning tool well-known for its efficiency and effectiveness. This experiment is not merely a demonstration of predictive modeling but a practical application aimed at detecting potential money laundering activities, showcasing the dataset's utility in developing robust transaction monitoinrg solutions in AML.

[1]:
```
pip install seaborn
```

```
Requirement already satisfied: seaborn in /opt/conda/lib/python3.10/site-
packages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in
/opt/conda/lib/python3.10/site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=0.25 in /opt/conda/lib/python3.10/site-
packages (from seaborn) (2.2.0)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in
/opt/conda/lib/python3.10/site-packages (from seaborn) (3.7.5)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site-
packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(1.4.5)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(21.3)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.10/site-
packages (from matplotlib!=3.6.1,>=3.1->seaborn) (9.5.0)
```

```
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-
packages (from pandas>=0.25->seaborn) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.10/site-
packages (from pandas>=0.25->seaborn) (2023.4)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-
packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```python
import pandas as pd
import numpy as np
from sklearn import preprocessing
import math
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
from PIL import Image
from scipy.stats import skew
from matplotlib.transforms import Bbox
```

[3]:
```python
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt
```

[4]:
```python
df = pd.read_csv("/kaggle/input/synthetic-transaction-monitoring-dataset-aml/
↪SAML-D.csv")
```

[5]:
```python
df.shape
```

[5]:
```
(9504852, 12)
```

[6]:
```python
df.head()
```

[6]:
```
       Time        Date  Sender_account  Receiver_account    Amount  \
0  10:35:19  2022-10-07      8724731955        2769355426   1459.15
1  10:35:20  2022-10-07      1491989064        8401255335   6019.64
2  10:35:20  2022-10-07       287305149        4404767002  14328.44
3  10:35:21  2022-10-07      5376652437        9600420220  11895.00
4  10:35:21  2022-10-07      9614186178        3803336972    115.25
```

```
  Payment_currency Received_currency Sender_bank_location  \
0        UK pounds         UK pounds                   UK
1        UK pounds            Dirham                   UK
2        UK pounds         UK pounds                   UK
3        UK pounds         UK pounds                   UK
4        UK pounds         UK pounds                   UK

  Receiver_bank_location  Payment_type  Is_laundering        Laundering_type
0                     UK  Cash Deposit              0  Normal_Cash_Deposits
1                    UAE  Cross-border              0         Normal_Fan_Out
2                     UK        Cheque              0   Normal_Small_Fan_Out
3                     UK           ACH              0          Normal_Fan_In
4                     UK  Cash Deposit              0  Normal_Cash_Deposits
```

[7]:
```
df.tail()
```

[7]:
```
             Time        Date  Sender_account  Receiver_account    Amount  \
9504847  10:57:01  2023-08-23      2453933570         519744068   2247.25
9504848  10:57:06  2023-08-23      9805510177        5416607878    927.18
9504849  10:57:06  2023-08-23      7282330957        2995527149   1455.14
9504850  10:57:11  2023-08-23       940337377        4812815165  25995.70
9504851  10:57:12  2023-08-23       105185176        6824994831   9586.08

        Payment_currency Received_currency Sender_bank_location  \
9504847        UK pounds         UK pounds                   UK
9504848        UK pounds         UK pounds                   UK
9504849        UK pounds         UK pounds                   UK
9504850        UK pounds         UK pounds                   UK
9504851        UK pounds         UK pounds                   UK

        Receiver_bank_location Payment_type  Is_laundering  \
9504847                     UK          ACH              0
9504848                     UK   Debit card              0
9504849                     UK          ACH              0
9504850                     UK          ACH              0
9504851                     UK          ACH              0

              Laundering_type
9504847  Normal_Small_Fan_Out
9504848  Normal_Small_Fan_Out
9504849  Normal_Small_Fan_Out
9504850         Normal_Fan_In
9504851        Normal_Fan_Out
```

0.0.2 Exploratory Data Analysis

[8]:
```python
# number of transactions per payment type
transactions_per_payment_type = df['Payment_type'].value_counts()

# number of laundering transactions per payment type
laundering_transactions_per_payment_type = df[df['Is_laundering'] == 1].
↪groupby('Payment_type').size()

transactions_per_payment_type, laundering_transactions_per_payment_type
```

[8]:
```
(Payment_type
 Credit card        2012909
 Debit card         2012103
 Cheque             2011419
 ACH                2008807
 Cross-border        933931
 Cash Withdrawal     300477
 Cash Deposit        225206
 Name: count, dtype: int64,
 Payment_type
 ACH                1159
 Cash Deposit       1405
 Cash Withdrawal    1334
 Cheque             1087
 Credit card        1136
 Cross-border       2628
 Debit card         1124
 dtype: int64)
```

[9]:
```python
df.describe()
```

[9]:

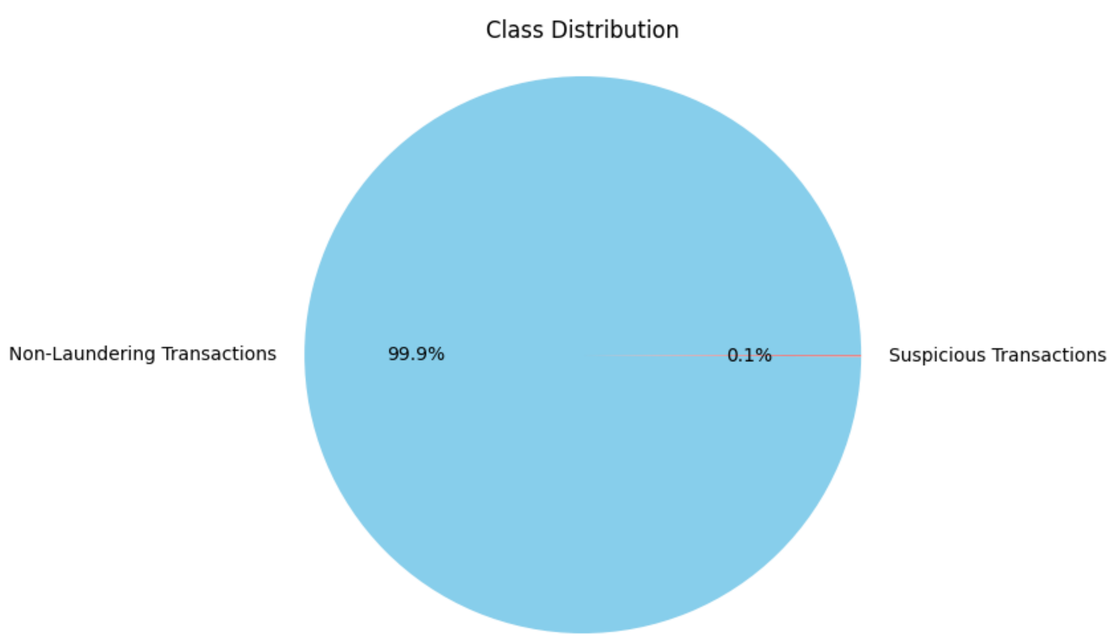
	Sender_account	Receiver_account	Amount	Is_laundering
count	9.504852e+06	9.504852e+06	9.504852e+06	9.504852e+06
mean	5.006619e+09	5.006006e+09	8.762968e+03	1.038733e-03
std	2.885814e+09	2.884763e+09	2.561495e+04	3.221263e-02
min	9.018000e+03	9.018000e+03	3.730000e+00	0.000000e+00
25%	2.513133e+09	2.513219e+09	2.143688e+03	0.000000e+00
50%	5.001017e+09	5.002572e+09	6.113720e+03	0.000000e+00
75%	7.505051e+09	7.502397e+09	1.045846e+04	0.000000e+00
max	9.999987e+09	9.999971e+09	1.261850e+07	1.000000e+00

[10]:
```python
laundering_stats = df[df['Is_laundering'] == 1]['Amount'].agg(['max', 'mean',␣
↪'min'])

normal_stats = df[df['Is_laundering'] == 0]['Amount'].agg(['max', 'mean',␣
↪'min'])
```

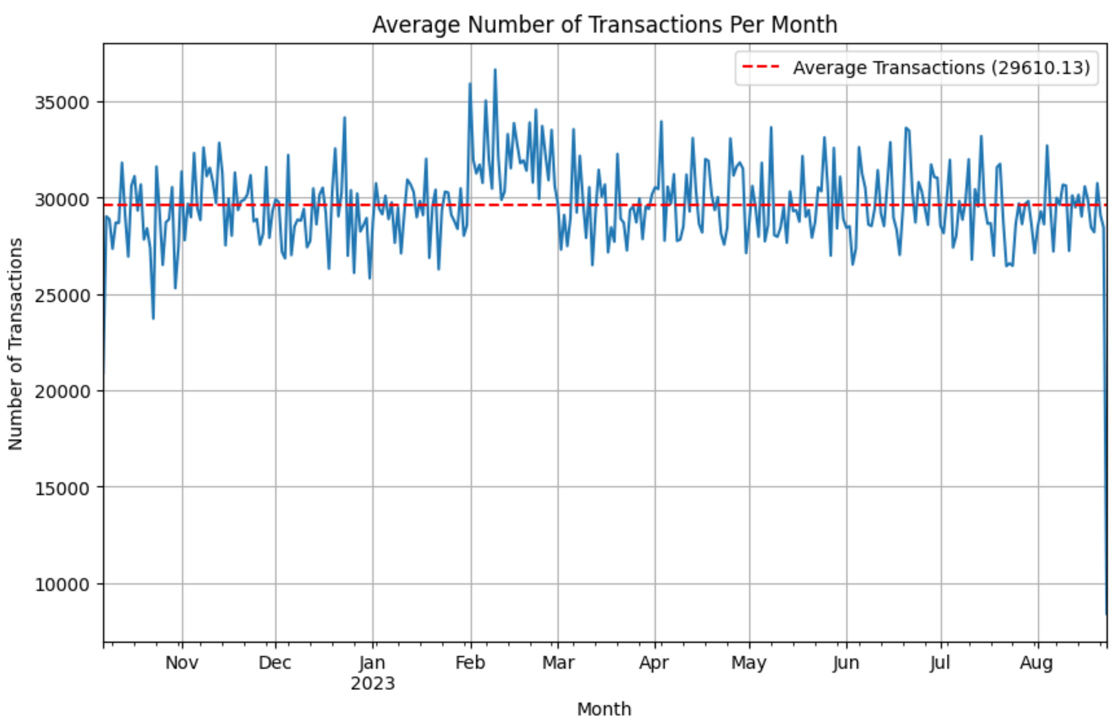
```
print("Laundering Transactions Stats:\n", laundering_stats)
print("\nNormal Transactions Stats:\n", normal_stats)
```

```
Laundering Transactions Stats:
 max     1.261850e+07
mean    4.058767e+04
min     1.582000e+01
Name: Amount, dtype: float64

Normal Transactions Stats:
 max     999962.190000
mean      8729.875874
min          3.730000
Name: Amount, dtype: float64
```

[11]:
```
class_distribution = df['Is_laundering'].value_counts()

plt.figure(figsize=(10, 6))
plt.pie(class_distribution, labels=['Non-Laundering Transactions', 'Suspicious␣
↪Transactions'], autopct='%1.1f%%', colors=['skyblue', 'lightcoral'])

plt.title('Class Distribution')
plt.axis('equal')

plt.show()
```

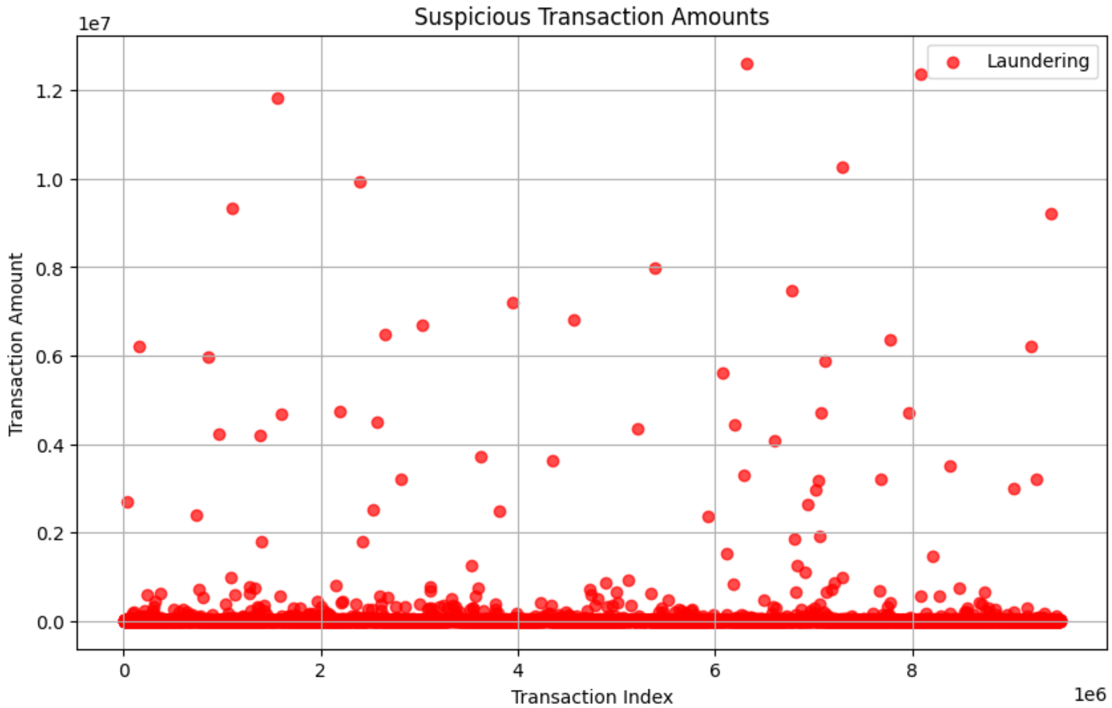


[12]:
```
df['Date'] = pd.to_datetime(df['Date'])
monthly_transactions = df.groupby(df['Date'].dt.to_period('D')).size()

# number of transactions per month
average_monthly_transactions = monthly_transactions.mean()

plt.figure(figsize=(10, 6))
monthly_transactions.plot(kind='line')
plt.axhline(y=average_monthly_transactions, color='r', linestyle='--',␣
 ↪label=f'Average Transactions ({average_monthly_transactions:.2f})')
plt.xlabel('Month')
plt.ylabel('Number of Transactions')
plt.title('Average Number of Transactions Per Month')
plt.legend()
plt.grid(True)
plt.show()
```



[13]:
```
# Separate the data
laundering_data = df[df['Is_laundering'] == 1]
non_laundering_data = df[df['Is_laundering'] == 0]

plt.figure(figsize=(10, 6))
```

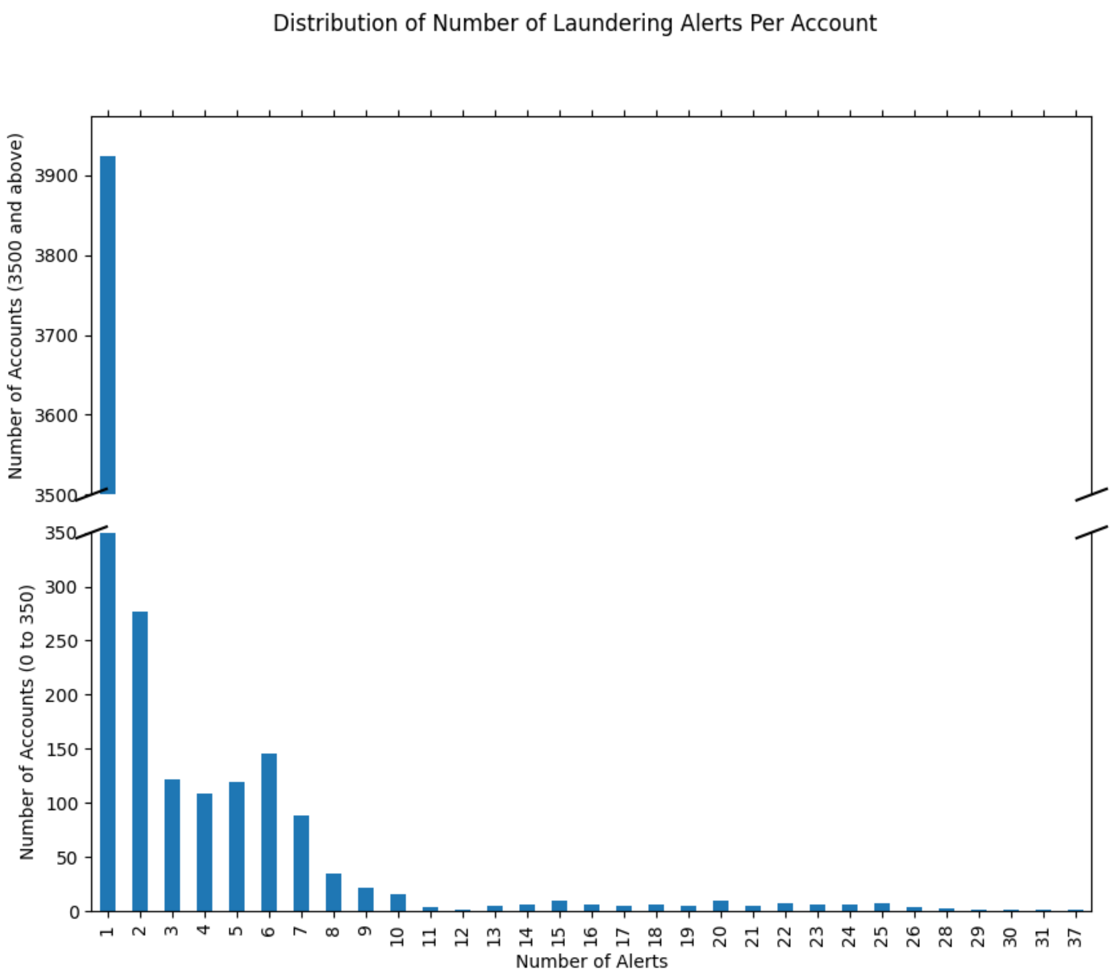
```
plt.scatter(laundering_data.index, laundering_data['Amount'], color='red',␣
↪label='Laundering', alpha=0.7)

plt.title('Suspicious Transaction Amounts')
plt.xlabel('Transaction Index')
plt.ylabel('Transaction Amount')
plt.legend()
plt.grid(True)
plt.show()
```



[14]:
```
laundering_df = df[df['Is_laundering'] == 1]

account_alert_counts = laundering_df.groupby('Sender_account').size()
alert_distribution = account_alert_counts.value_counts()
alert_distribution = alert_distribution.sort_index()

fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(10, 8))
fig.subplots_adjust(hspace=0.1)

alert_distribution.plot(kind='bar', ax=ax1)
alert_distribution.plot(kind='bar', ax=ax2)

fig.suptitle('Distribution of Number of Laundering Alerts Per Account')
```

```
ax1.set_ylim(3500, alert_distribution.max()+50)
ax2.set_ylim(0, 350)

ax1.spines['bottom'].set_visible(False)
ax2.spines['top'].set_visible(False)
ax1.xaxis.tick_top()
ax1.tick_params(labeltop=False)
ax2.xaxis.tick_bottom()

d = .015  # diagonal lines size
kwargs = dict(transform=ax1.transAxes, color='k', clip_on=False)
ax1.plot((-d, +d), (-d, +d), **kwargs)
ax1.plot((1 - d, 1 + d), (-d, +d), **kwargs)

kwargs.update(transform=ax2.transAxes)
ax2.plot((-d, +d), (1 - d, 1 + d), **kwargs)
ax2.plot((1 - d, 1 + d), (1 - d, 1 + d), **kwargs)

ax2.set_ylabel('Number of Accounts (0 to 350)')
ax1.set_ylabel('Number of Accounts (3500 and above)')
ax2.set_xlabel('Number of Alerts')
```

[14]: Text(0.5, 0, 'Number of Alerts')

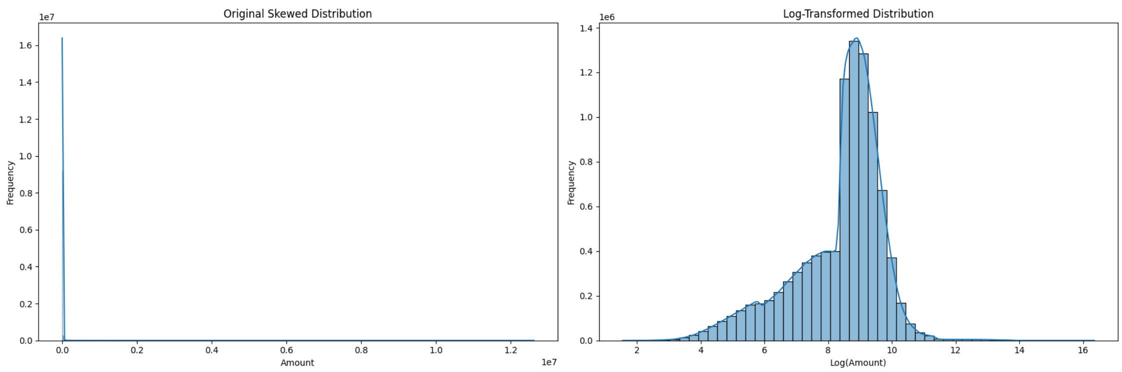


[15]:
```
skewed_data = df['Amount']

original_skewness = skew(skewed_data)
print(f"Original Skewness: {original_skewness}")

# Apply a log transformation
log_transformed_data = np.log1p(skewed_data)
# skewness after log transformation
transformed_skewness = skew(log_transformed_data)
print(f"Log-Transformed Skewness: {transformed_skewness}")

fig, ax = plt.subplots(1, 2, figsize=(18, 6))
sns.histplot(skewed_data, bins=500, kde=True, ax=ax[0])
ax[0].set_title('Original Skewed Distribution')
ax[0].set_xlabel('Amount')
ax[0].set_ylabel('Frequency')
```

```
sns.histplot(log_transformed_data, bins=50, kde=True, ax=ax[1])
ax[1].set_title('Log-Transformed Distribution')
ax[1].set_xlabel('Log(Amount)')
ax[1].set_ylabel('Frequency')

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

```
Original Skewness: 102.16408577285024
Log-Transformed Skewness: -1.0103052224946008
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[16]:
```
total_amount_pivot = pd.pivot_table(df, index=["Payment_type"],␣
↪values='Amount', aggfunc=np.sum)
laundering_count_pivot = df[df['Is_laundering'] == 1].groupby('Payment_type').
↪size().to_frame('Laundering_Count')
normal_count_pivot = df[df['Is_laundering'] == 0].groupby('Payment_type').
↪size().to_frame('Normal_Count')

combined_pivot = total_amount_pivot.join([laundering_count_pivot,␣
↪normal_count_pivot], how='outer')
combined_pivot = combined_pivot.fillna(0)

cm = sns.light_palette("blue", as_cmap=True)
styled_combined_pivot = combined_pivot.style.background_gradient(cmap=cm)
```

```
styled_combined_pivot
```

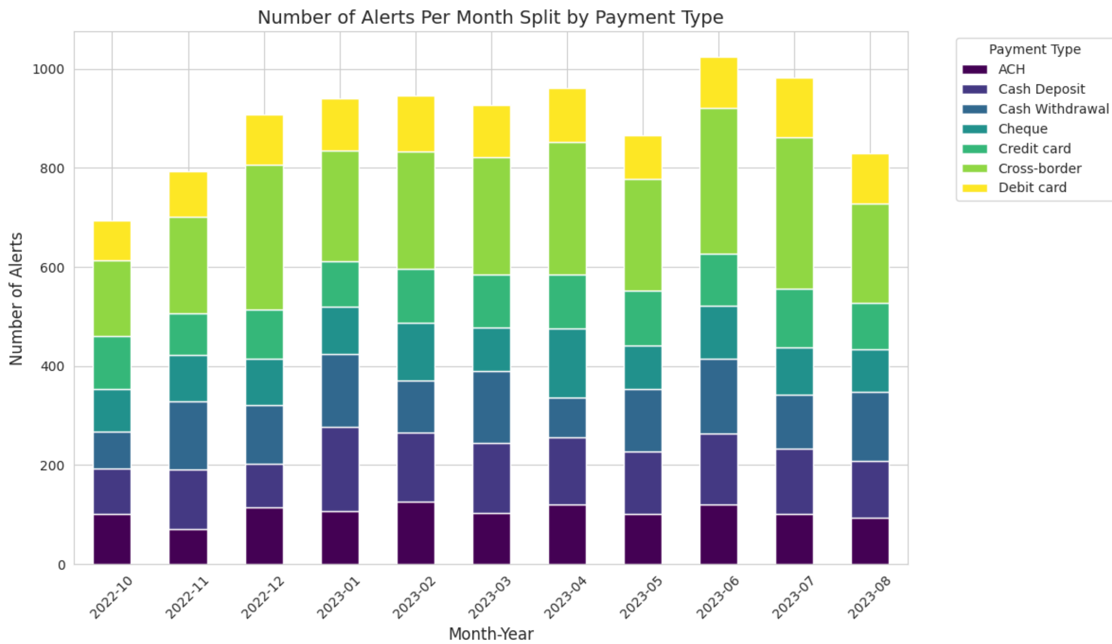
```
/tmp/ipykernel_17/1828936050.py:1: FutureWarning: The provided callable
<function sum at 0x7c8d487d7400> is currently using DataFrameGroupBy.sum. In a
future version of pandas, the provided callable will be used directly. To keep
current behavior pass the string "sum" instead.
  total_amount_pivot = pd.pivot_table(df, index=["Payment_type"],
values='Amount', aggfunc=np.sum)
```

[16]:

```
<pandas.io.formats.style.Styler at 0x7c8d207bc760>
```

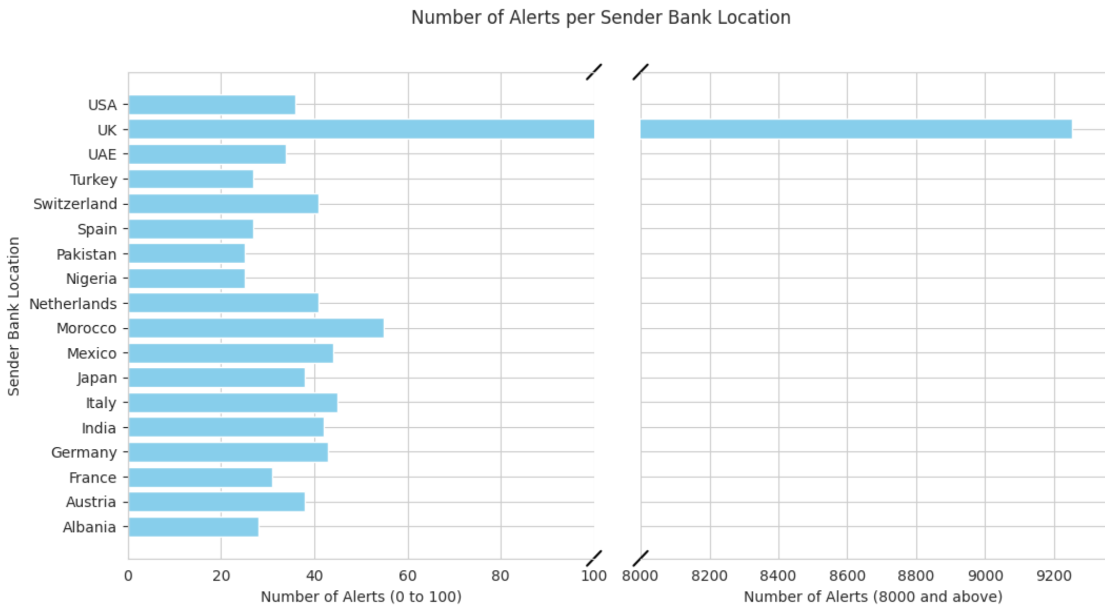
[17]:

```
df['Date'] = pd.to_datetime(df['Date'])
grouped_data = df.groupby(['Date', 'Payment_type']).agg({'Is_laundering':␣
↪'sum'}).reset_index()
grouped_data['Month_Year'] = grouped_data['Date'].dt.to_period('M')
monthly_alerts = grouped_data.groupby(['Month_Year', 'Payment_type']).
↪agg({'Is_laundering': 'sum'}).reset_index()

pivot_data = monthly_alerts.pivot(index='Month_Year', columns='Payment_type',␣
↪values='Is_laundering')

sns.set_style("whitegrid")
fig, ax = plt.subplots(figsize=(12, 7))
pivot_data.plot(kind='bar', ax=ax, stacked=True, colormap='viridis')
pivot_data = monthly_alerts.pivot(index='Month_Year', columns='Payment_type',␣
↪values='Is_laundering')

plt.title('Number of Alerts Per Month Split by Payment Type', fontsize=14)
plt.xlabel('Month-Year', fontsize=12)
plt.ylabel('Number of Alerts', fontsize=12)
plt.xticks(rotation=45)

plt.legend(title='Payment Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



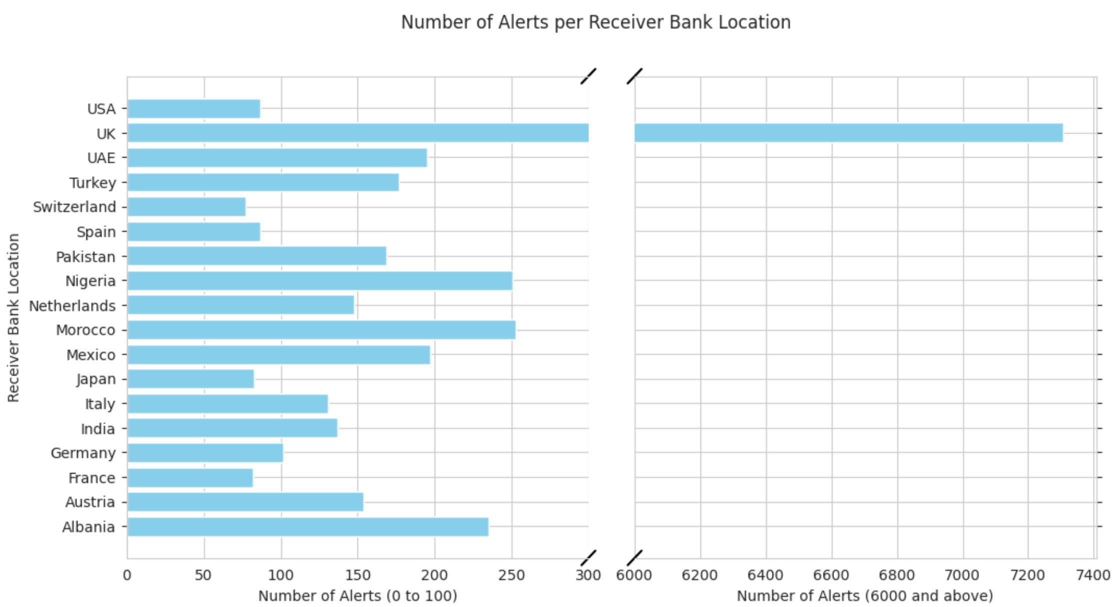
[18]:
```
alerts_per_location = df.groupby('Sender_bank_location')['Is_laundering'].sum().
↪reset_index()

fig, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(12, 6))
fig.subplots_adjust(wspace=0.1)

ax1.barh(alerts_per_location['Sender_bank_location'],␣
↪alerts_per_location['Is_laundering'], color='skyblue')
ax2.barh(alerts_per_location['Sender_bank_location'],␣
↪alerts_per_location['Is_laundering'], color='skyblue')

ax1.set_xlim(0, 100)  # Set the left subplot values
ax2.set_xlim(8000, max(alerts_per_location['Is_laundering']) + 100)  # Set the␣
↪right subplot values

fig.suptitle('Number of Alerts per Sender Bank Location')

ax1.spines['right'].set_visible(False)
ax2.spines['left'].set_visible(False)
ax1.yaxis.tick_left()
ax2.yaxis.tick_right()
ax2.set_yticks([])

d = .015  # Size of diagonal lines
kwargs = dict(transform=ax1.transAxes, color='k', clip_on=False)
ax1.plot((1 - d, 1 + d), (-d, +d), **kwargs)
```

```
ax1.plot((1 - d, 1 + d), (1 - d, 1 + d), **kwargs)

kwargs.update(transform=ax2.transAxes)
ax2.plot((-d, +d), (-d, +d), **kwargs)
ax2.plot((-d, +d), (1 - d, 1 + d), **kwargs)

ax1.set_xlabel('Number of Alerts (0 to 100)')
ax2.set_xlabel('Number of Alerts (8000 and above)')
ax1.set_ylabel('Sender Bank Location')

ax1.set_yticks(range(len(alerts_per_location['Sender_bank_location'])))
ax1.set_yticklabels(alerts_per_location['Sender_bank_location'])

plt.show()
```



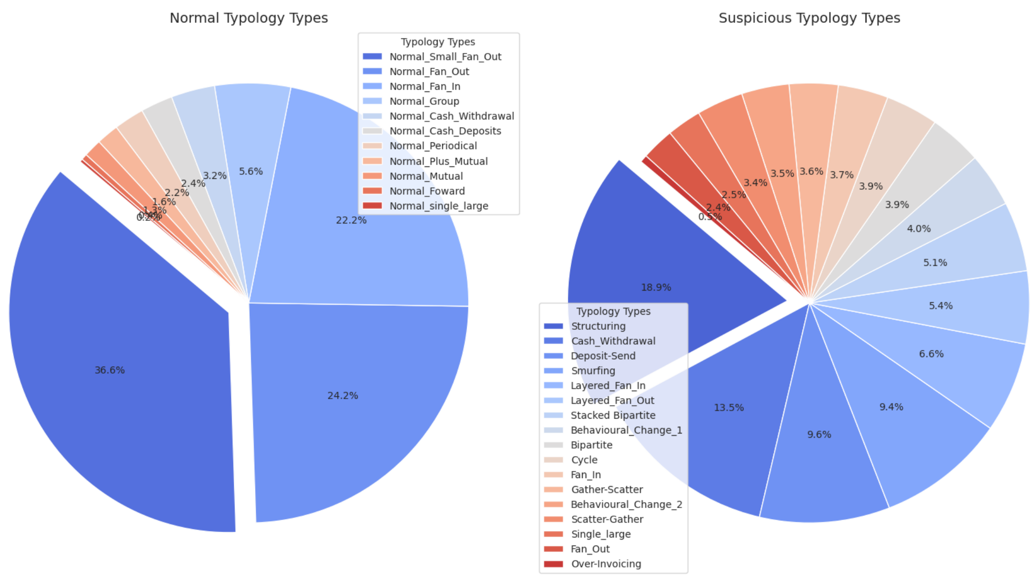
[19]:
```
alerts_per_location = df.groupby('Receiver_bank_location')['Is_laundering'].
    ↪sum().reset_index()

fig, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(12, 6))
fig.subplots_adjust(wspace=0.1)  # Adjust the spacing between subplots

ax1.barh(alerts_per_location['Receiver_bank_location'],␣
    ↪alerts_per_location['Is_laundering'], color='skyblue')
ax2.barh(alerts_per_location['Receiver_bank_location'],␣
    ↪alerts_per_location['Is_laundering'], color='skyblue')

ax1.set_xlim(0, 300)  # Set the left subplot values
```

```python
ax2.set_xlim(6000, max(alerts_per_location['Is_laundering']) + 100)  # Set the right subplot values

fig.suptitle('Number of Alerts per Receiver Bank Location')

ax1.spines['right'].set_visible(False)
ax2.spines['left'].set_visible(False)
ax1.yaxis.tick_left()
ax2.yaxis.tick_right()
ax2.set_yticks([])

d = .015  # Size of diagonal lines
kwargs = dict(transform=ax1.transAxes, color='k', clip_on=False)
ax1.plot((1 - d, 1 + d), (-d, +d), **kwargs)
ax1.plot((1 - d, 1 + d), (1 - d, 1 + d), **kwargs)

kwargs.update(transform=ax2.transAxes)
ax2.plot((-d, +d), (-d, +d), **kwargs)
ax2.plot((-d, +d), (1 - d, 1 + d), **kwargs)

ax1.set_xlabel('Number of Alerts (0 to 100)')
ax2.set_xlabel('Number of Alerts (6000 and above)')
ax1.set_ylabel('Receiver Bank Location')

ax1.set_yticks(range(len(alerts_per_location['Receiver_bank_location'])))
ax1.set_yticklabels(alerts_per_location['Receiver_bank_location'])

plt.show()
```



[20]:

```python
normal_data = df[df['Is_laundering'] == 0]['Laundering_type'].value_counts()
laundering_data = df[df['Is_laundering'] == 1]['Laundering_type'].value_counts()

# palette_normal = sns.color_palette("husl", len(normal_data))
palette_normal = sns.color_palette("coolwarm", len(normal_data))
palette_laundering = sns.color_palette("coolwarm", len(laundering_data))

fig, axs = plt.subplots(1, 2, figsize=(16, 10))

explode_normal = [0.1] + [0] * (len(normal_data) - 1)
explode_laundering = [0.1] + [0] * (len(laundering_data) - 1)

patches, texts, autotexts = axs[0].pie(normal_data, explode=explode_normal,
 ↪autopct='%1.1f%%', colors=palette_normal, startangle=140)
axs[0].set_title('Normal Typology Types', fontsize=14)
axs[0].legend(patches, normal_data.index, loc='best', title="Typology Types",
 ↪fontsize=10)

patches, texts, autotexts = axs[1].pie(laundering_data,
 ↪explode=explode_laundering, autopct='%1.1f%%', colors=palette_laundering,
 ↪startangle=140)
axs[1].set_title('Suspicious Typology Types', fontsize=14)
axs[1].legend(patches, laundering_data.index, loc='best', title="Typology
 ↪Types", fontsize=10)

for text in texts + autotexts:
  text.set_fontsize(10)

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



[21]:
```
df = pd.read_csv("/kaggle/input/synthetic-transaction-monitoring-dataset-aml/
↪SAML-D.csv")
```

0.1 Pre-Processing the Data

[22]:
```
df['Hour'] = pd.to_datetime(df['Time']).dt.hour

df['Date_Year'] = pd.to_datetime(df['Date']).dt.year
df['Date_Month'] = pd.to_datetime(df['Date']).dt.month
df['Date_Day'] = pd.to_datetime(df['Date']).dt.day

df.drop(columns=['Laundering_type'], inplace=True)
df.drop(columns=['Time', 'Date'], inplace=True)
```

```
/tmp/ipykernel_17/3885589180.py:1: UserWarning: Could not infer format, so each
element will be parsed individually, falling back to `dateutil`. To ensure
parsing is consistent and as-expected, please specify a format.
  df['Hour'] = pd.to_datetime(df['Time']).dt.hour
```

[23]:
```
skewed_data = df['Amount']
original_skewness = skew(skewed_data)
print(f"Original Skewness: {original_skewness}")

log_transformed_data = np.log1p(skewed_data)
transformed_skewness = skew(log_transformed_data)
print(f"Log-Transformed Skewness: {transformed_skewness}")

df['Amount'] = log_transformed_data
```

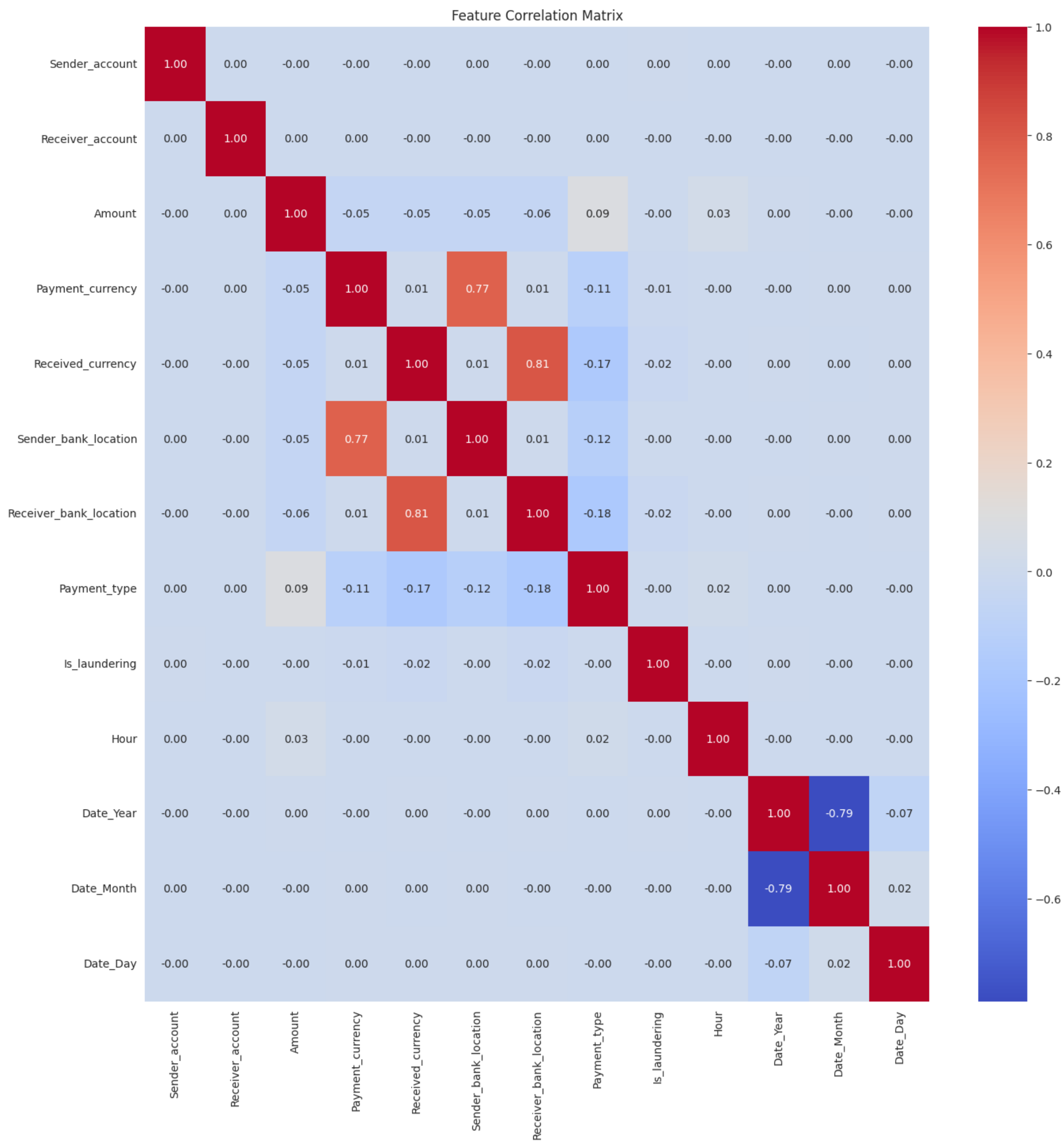
```
Original Skewness: 102.16408577285024
Log-Transformed Skewness: -1.0103052224946008
```

[24]:
```python
categorical_cols = ['Sender_account', 'Receiver_account', 'Payment_currency',␣
 ↪'Received_currency',
                    'Sender_bank_location', 'Receiver_bank_location',␣
 ↪'Payment_type',
                    'Date_Year', 'Date_Month', 'Date_Day']

for col in categorical_cols:
    encoder = preprocessing.LabelEncoder()
    df[col] = encoder.fit_transform(df[col])

numerical_cols = ['Hour', 'Amount']

scaler = preprocessing.StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

df.head()
```

[24]:
```
   Sender_account  Receiver_account    Amount  Payment_currency  \
0          255277            180462 -0.756957                10
1           43554            548141  0.253092                10
2            8553            287765  0.871335                10
3          157249            626370  0.738639                10
4          281324            248338 -2.561195                10

   Received_currency  Sender_bank_location  Receiver_bank_location  \
0                 10                    16                      16
1                  1                    16                      15
2                 10                    16                      16
3                 10                    16                      16
4                 10                    16                      16

   Payment_type  Is_laundering      Hour  Date_Year  Date_Month  Date_Day
0             1              0 -0.732897          0           8         6
1             5              0 -0.732897          0           8         6
2             3              0 -0.732897          0           8         6
3             0              0 -0.732897          0           8         6
4             1              0 -0.732897          0           8         6
```

[25]:
```python
correlation_matrix = df.corr()

plt.figure(figsize=(16, 16))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Feature Correlation Matrix')
```

```
plt.xticks(rotation=90)
plt.yticks(rotation=0)

plt.show()
```



0.2 Split the Data

[26]:
```
X = df.drop(columns=['Is_laundering'])
y = df['Is_laundering']
```

[27]:
```
total_size = len(df)
train_size = int(0.8 * total_size)
test_and_validation_size = total_size - train_size
test_size = int(0.5 * test_and_validation_size)
validation_size = test_and_validation_size - test_size

X_train = X[:train_size]
y_train = y[:train_size]

X_validation = X[train_size:train_size+test_size]
y_validation = y[train_size:train_size+test_size]

X_test = X[train_size+test_size:]
y_test = y[train_size+test_size:]

print(X_train.shape, y_train.shape, X_test.shape, y_test.shape, X_validation.
 ↪shape, y_validation.shape)
```

```
(7603881, 12) (7603881,) (950486, 12) (950486,) (950485, 12) (950485,)
```

0.3 XGB Classifier Experiment

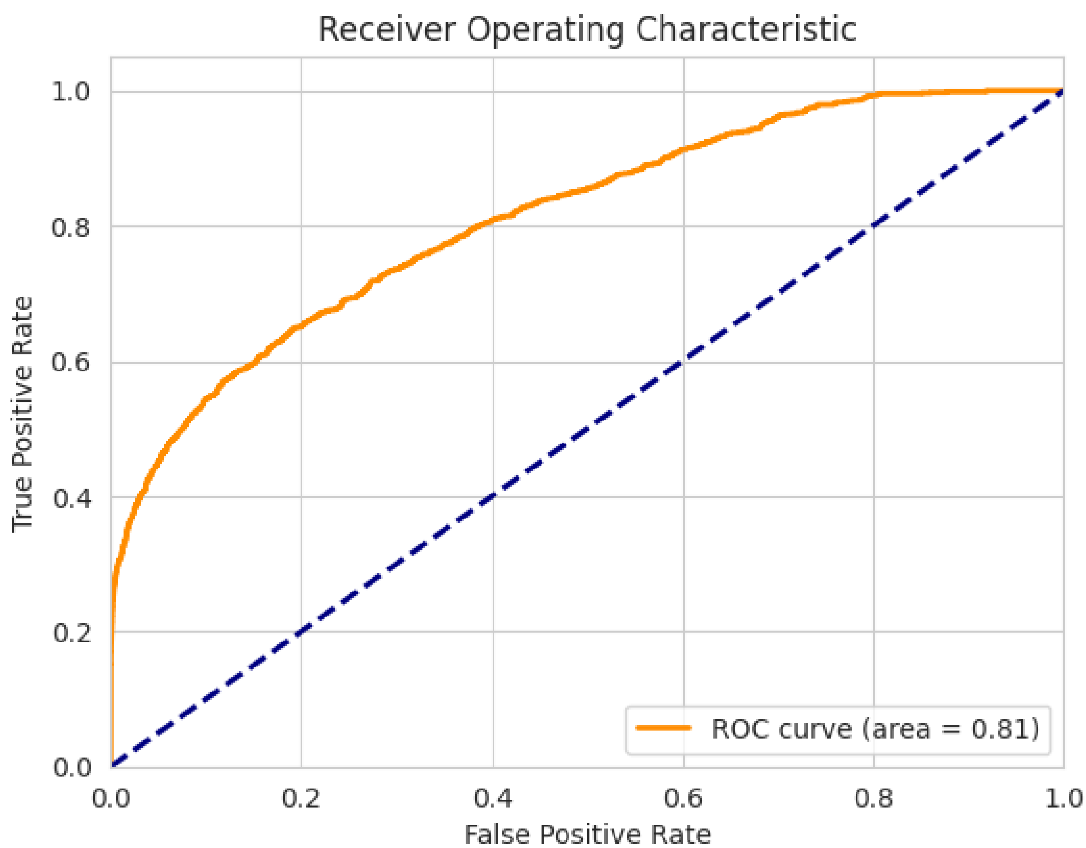
[28]:
```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    # 'max_depth': [4,8,16],
    # 'eta': [0.1,0.2,0.3],
}

xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
 ↪random_state=42)

grid_search = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring='roc_auc',
    cv=2,
    verbose=2
)

grid_search.fit(X_train, y_train)

print("Best Parameters: ", grid_search.best_params_)

best_model = grid_search.best_estimator_
```

```
val_predictions = best_model.predict_proba(X_validation)[:, 1]
val_auc = roc_auc_score(y_validation, val_predictions)
print("Validation AUC: ", val_auc)

test_predictions = best_model.predict_proba(X_test)[:, 1]
test_auc = roc_auc_score(y_test, test_predictions)
print("Test AUC: ", test_auc)
```

```
Fitting 2 folds for each of 1 candidates, totalling 2 fits
[CV] END … total time=  31.5s
[CV] END … total time=  31.1s
Best Parameters:  {}
Validation AUC:  0.8271847313025383
Test AUC:  0.8120470486529636
```

[29]:
```
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

test_probabilities = best_model.predict_proba(X_test)[:, 1]

test_auc = roc_auc_score(y_test, test_probabilities)
print("Test Set AUC: ", test_auc)

fpr, tpr, thresholds = roc_curve(y_test, test_probabilities)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪test_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')
plt.legend(loc="lower right")
plt.show()
```

```
Test Set AUC:  0.8120470486529636
```

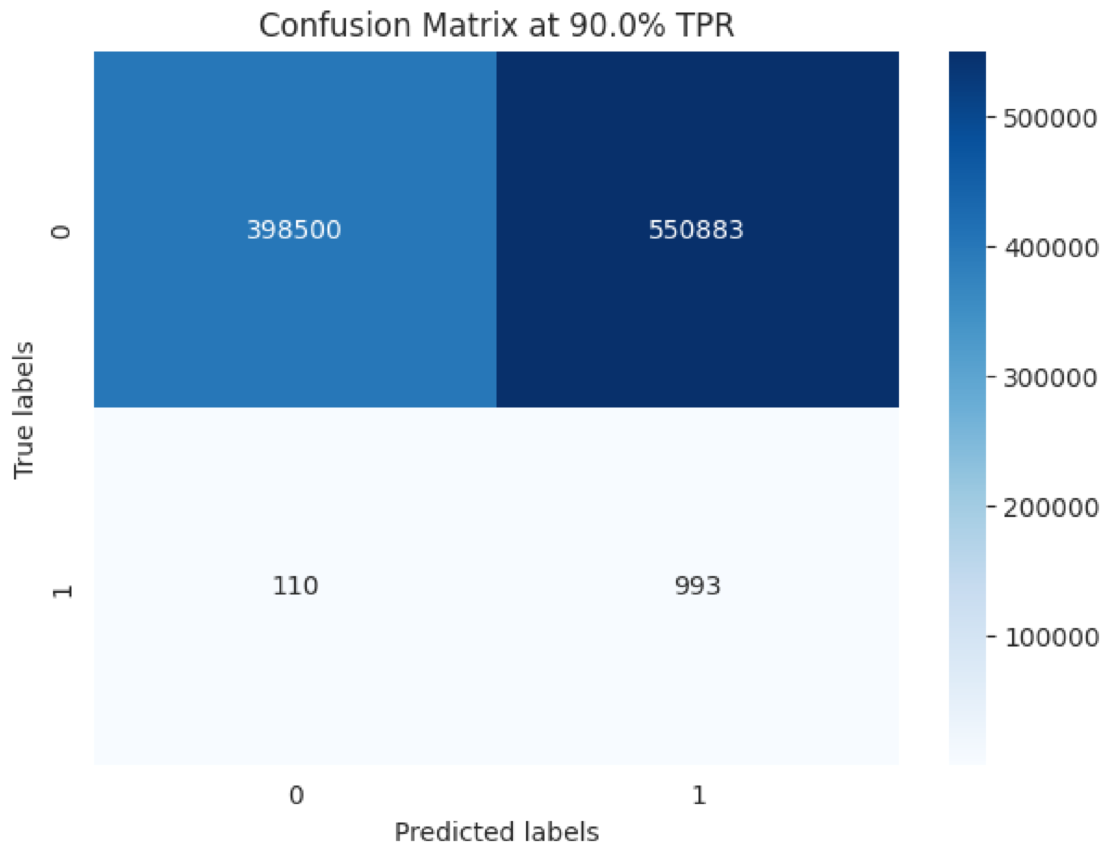


[30]:

```
# Confusion Matrix, TPR, and FPR at around a TPR of 0.9
desired_tpr = 0.9
closest_threshold = thresholds[np.argmin(np.abs(tpr - desired_tpr))]
print(f"Desired TPR of around 90%:")

y_pred = (test_probabilities >= closest_threshold).astype(int)
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(7,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title(f'Confusion Matrix at {desired_tpr*100}% TPR')
plt.show()

tn, fp, fn, tp = cm.ravel()
fpr_cm = fp / (fp + tn)
tpr_cm = tp / (tp + fn)

print(f"False Positive Rate (FPR): {fpr_cm:.3f}")
```

```python
print(f"True Positive Rate (TPR): {tpr_cm:.3f}")
```

```
Desired TPR of around 90%:
```




```
False Positive Rate (FPR): 0.580
True Positive Rate (TPR): 0.900
```