Enhancing Anti-Money Laundering using Machine Learning !!!

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
df = pd.read_csv('money_laundering.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	Time	Date	Sender_account	Receiver_account	Amount	Payment_currency	Received_
0	10:35:19	07-10-2022	8724731955	2769355426	1459.15	UK pounds	L
1	10:35:20	07-10-2022	1491989064	8401255335	6019.64	UK pounds	
2	10:35:20	07-10-2022	287305149	4404767002	14328.44	UK pounds	L
3	10:35:21	07-10-2022	5376652437	9600420220	11895.00	UK pounds	L
4	10:35:21	07-10-2022	9614186178	3803336972	115.25	UK pounds	L

In [5]:
```
df.tail()
```

Out[5]:

	Time	Date	Sender_account	Receiver_account	Amount	Payment_currency	Re
1048570	09:21:10	12-11-2022	3848621169	3388139373	21559.31	UK pounds	
1048571	09:21:12	12-11-2022	8276335513	7220829571	14590.90	UK pounds	
1048572	09:21:13	12-11-2022	3014566949	6653549199	7141.85	UK pounds	
1048573	09:21:15	12-11-2022	1426173499	3569198271	14675.91	UK pounds	
1048574	09:21:16	12-11-2022	7798805872	9278506258	32473.03	UK pounds	

In [6]:
```
df.shape
```

Out[6]:
```
(1048575, 12)
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['Time', 'Date', 'Sender_account', 'Receiver_account', 'Amount',
       'Payment_currency', 'Received_currency', 'Sender_bank_location',
       'Receiver_bank_location', 'Payment_type', 'Is_laundering',
       'Laundering_type'],
      dtype='object')
```

In [8]:
```python
df.duplicated().sum()
```

Out[8]:
```
0
```

In [9]:
```python
df.isnull().sum()
```

Out[9]:
```
Time                      0
Date                      0
Sender_account            0
Receiver_account          0
Amount                    0
Payment_currency          0
Received_currency         0
Sender_bank_location      0
Receiver_bank_location    0
Payment_type              0
Is_laundering             0
Laundering_type           0
dtype: int64
```

In [10]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 12 columns):
 #   Column                  Non-Null Count    Dtype  
---  ------                  --------------    -----  
 0   Time                    1048575 non-null  object 
 1   Date                    1048575 non-null  object 
 2   Sender_account          1048575 non-null  int64  
 3   Receiver_account        1048575 non-null  int64  
 4   Amount                  1048575 non-null  float64
 5   Payment_currency        1048575 non-null  object 
 6   Received_currency       1048575 non-null  object 
 7   Sender_bank_location    1048575 non-null  object 
 8   Receiver_bank_location  1048575 non-null  object 
 9   Payment_type            1048575 non-null  object 
 10  Is_laundering           1048575 non-null  int64  
 11  Laundering_type         1048575 non-null  object 
dtypes: float64(1), int64(3), object(8)
memory usage: 96.0+ MB
```

In [11]:
```
df.describe()
```

Out[11]:

	Sender_account	Receiver_account	Amount	Is_laundering
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06
mean	5.007747e+09	5.032774e+09	8.707645e+03	9.117135e-04
std	2.889940e+09	2.882542e+09	2.444582e+04	3.018084e-02
min	9.217200e+04	4.823800e+04	5.190000e+00	0.000000e+00
25%	2.501276e+09	2.528695e+09	2.114535e+03	0.000000e+00
50%	4.999680e+09	5.042954e+09	6.104870e+03	0.000000e+00
75%	7.509491e+09	7.543159e+09	1.034893e+04	0.000000e+00
max	9.999913e+09	9.999971e+09	6.213932e+06	1.000000e+00

In [12]:
```
df.nunique()
```

Out[12]:
```
Time                         85770
Date                            37
Sender_account               71046
Receiver_account            278803
Amount                      759925
Payment_currency                13
Received_currency               13
Sender_bank_location            18
Receiver_bank_location          18
Payment_type                     8
Is_laundering                    2
Laundering_type                 28
dtype: int64
```

In [13]:
```
object_columns = df.select_dtypes(include=['object', 'bool']).columns
print("Object type columns:")
print(object_columns)

numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
print("\nNumerical type columns:")
print(numerical_columns)
```

```
Object type columns:
Index(['Time', 'Date', 'Payment_currency', 'Received_currency',
       'Sender_bank_location', 'Receiver_bank_location', 'Payment_type',
       'Laundering_type'],
      dtype='object')

Numerical type columns:
Index(['Sender_account', 'Receiver_account', 'Amount', 'Is_laundering'], d
type='object')
```

In [14]:
```
def classify_features(df):
    categorical_features = []
    non_categorical_features = []
    discrete_features = []
    continuous_features = []

    for column in df.columns:
        if df[column].dtype in ['object', 'bool']:
            if df[column].nunique() < 15:
                categorical_features.append(column)
            else:
                non_categorical_features.append(column)
        elif df[column].dtype in ['int64', 'float64']:
            if df[column].nunique() < 10:
                discrete_features.append(column)
            else:
                continuous_features.append(column)

    return categorical_features, non_categorical_features, discrete_feature
```

In [15]:
```
categorical, non_categorical, discrete, continuous = classify_features(df)
```

In [16]:
```
print("Categorical Features:", categorical)
print("Non-Categorical Features:", non_categorical)
print("Discrete Features:", discrete)
print("Continuous Features:", continuous)
```

```
Categorical Features: ['Payment_currency', 'Received_currency', 'Payment_t
ype']
Non-Categorical Features: ['Time', 'Date', 'Sender_bank_location', 'Receiv
er_bank_location', 'Laundering_type']
Discrete Features: ['Is_laundering']
Continuous Features: ['Sender_account', 'Receiver_account', 'Amount']
```

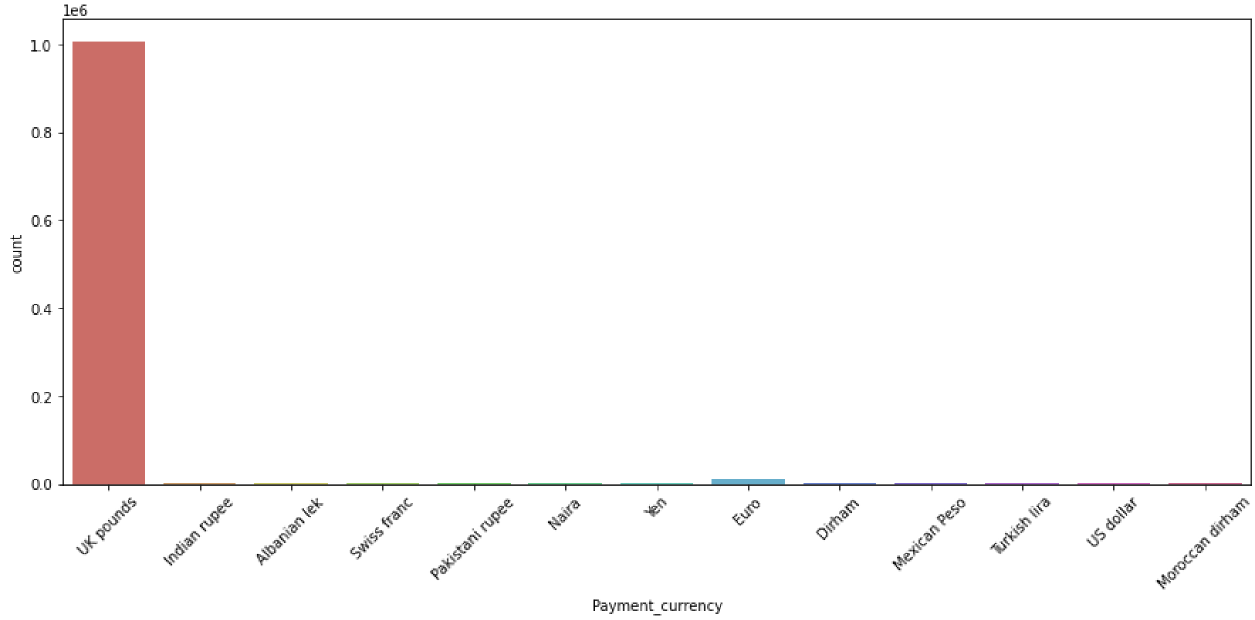
In [17]:
```
for i in categorical:
    print(i, ':')
    print(df[i].unique())
    print()
```

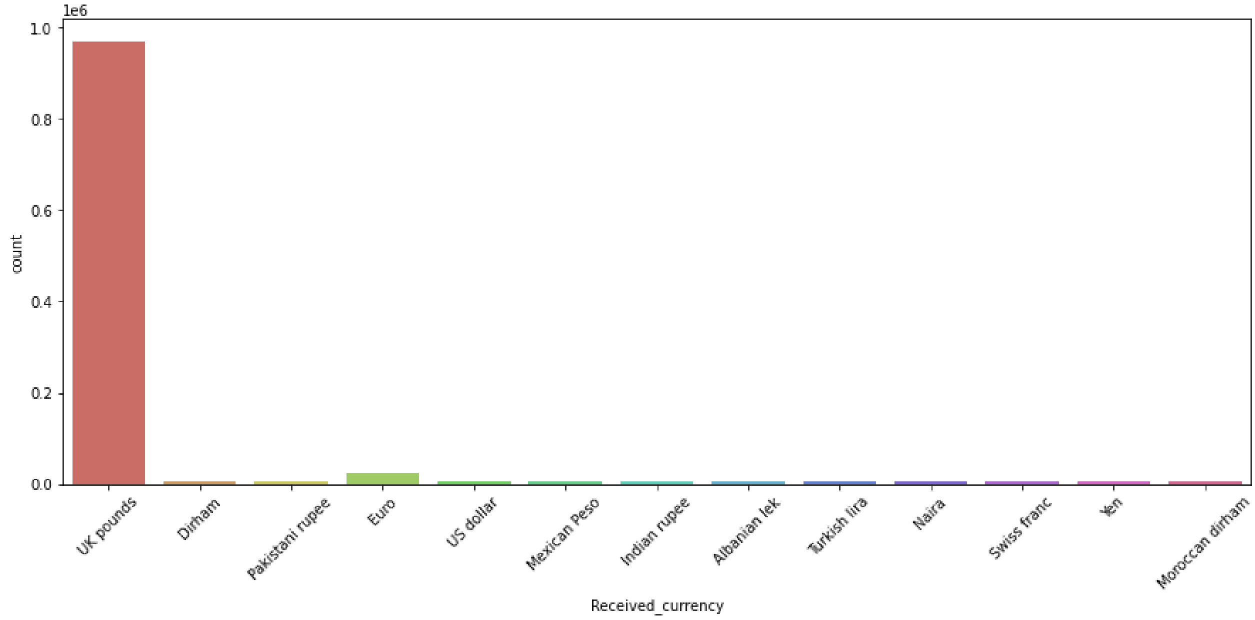
```
Payment_currency :
['UK pounds' 'Indian rupee' 'Albanian lek' 'Swiss franc' 'Pakistani rupee'
 'Naira' 'Yen' 'Euro' 'Dirham' 'Mexican Peso' 'Turkish lira' 'US dollar'
 'Moroccan dirham']

Received_currency :
['UK pounds' 'Dirham' 'Pakistani rupee' 'Euro' 'US dollar' 'Mexican Peso'
 'Indian rupee' 'Albanian lek' 'Turkish lira' 'Naira' 'Swiss franc' 'Yen'
 'Moroccan dirham']

Payment_type :
['Cash Deposit' 'Cross-border' 'Cheque' 'ACH' 'Credit card' 'Debit card'
 'Cash Withdrawal' 'Cross-border Withdrawal']
```

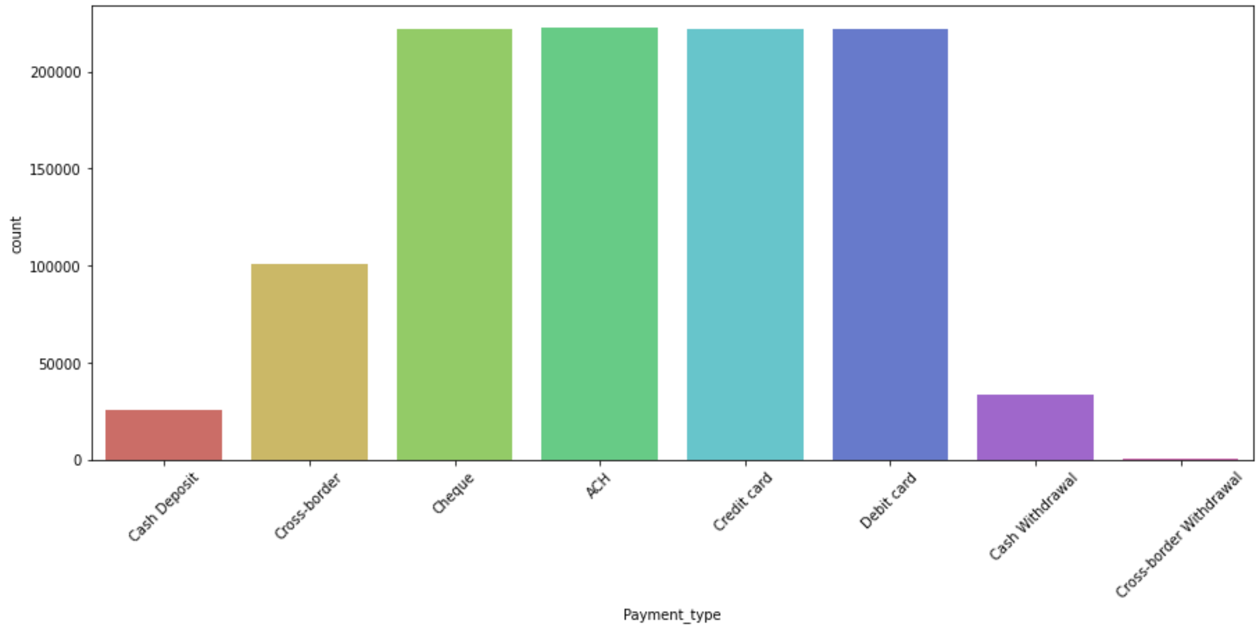
In [18]:
```
for i in categorical:
    print(i, ':')
    print(df[i].value_counts())
    print()
```

```
Payment_currency :
UK pounds          1005855
Euro                 11633
Swiss franc           3279
Yen                   3197
Turkish lira          3139
Dirham                3129
US dollar             2860
Naira                 2695
Indian rupee          2692
Pakistani rupee       2657
Moroccan dirham       2522
Albanian lek          2470
Mexican Peso          2447
Name: Payment_currency, dtype: int64

Received_currency :
UK pounds          969035
Euro                24388
Mexican Peso         6690
Naira                5729
Albanian lek         5611
Moroccan dirham      5225
Yen                  4795
Dirham               4777
US dollar            4731
Swiss franc          4660
Pakistani rupee      4415
Indian rupee         4366
Turkish lira         4153
Name: Received_currency, dtype: int64

Payment_type :
ACH                           222636
Credit card                   222166
Cheque                        221939
Debit card                    221781
Cross-border                  100595
Cash Withdrawal                33577
Cash Deposit                   25154
Cross-border Withdrawal          727
Name: Payment_type, dtype: int64
```

In [19]:
```
for i in categorical:
    plt.figure(figsize=(15, 6))
    sns.countplot(x=i, data=df, palette='hls')
    plt.xticks(rotation = 45)
    plt.show()
```





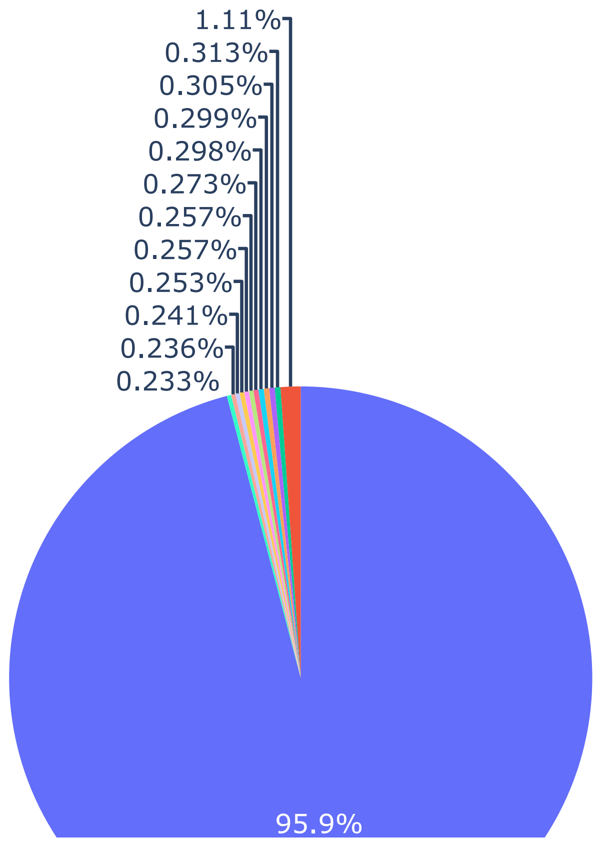


In [20]:

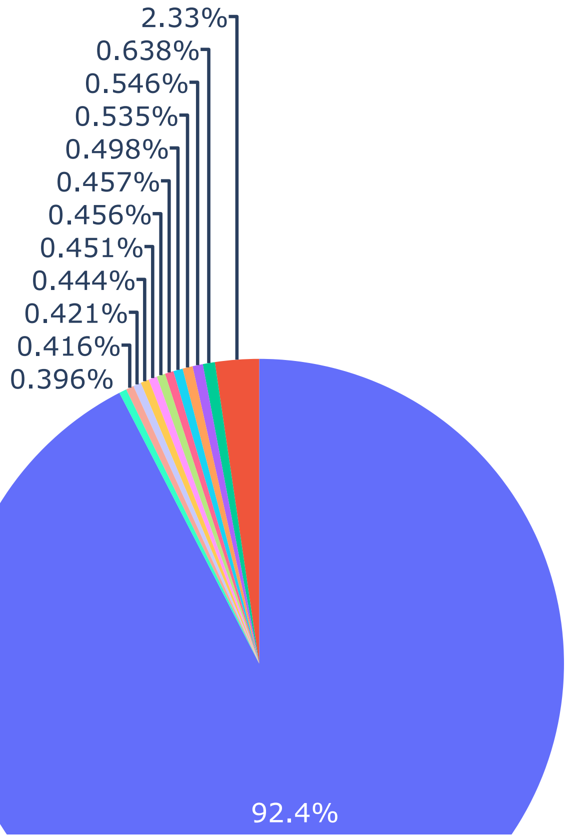
```
import plotly.express as px
```

In [21]:
```
for i in categorical:
    fig = px.pie(df, names=i)
    print('Pieplot for:', i)
    fig.show()
```

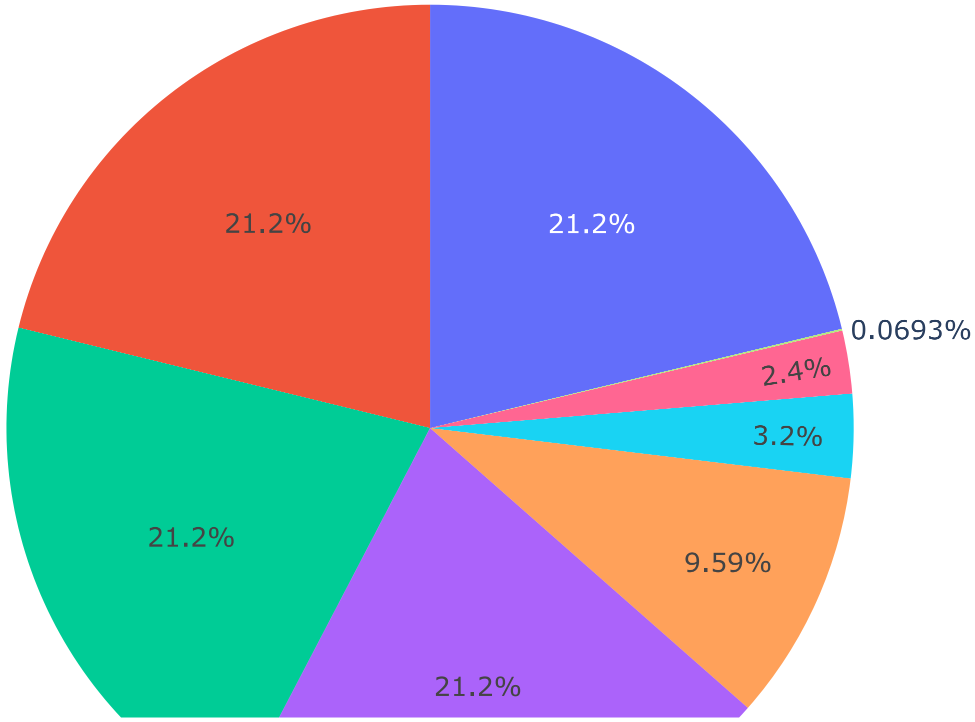
Pieplot for: Payment_currency



Pieplot for: Received_currency



Pieplot for: Payment_type



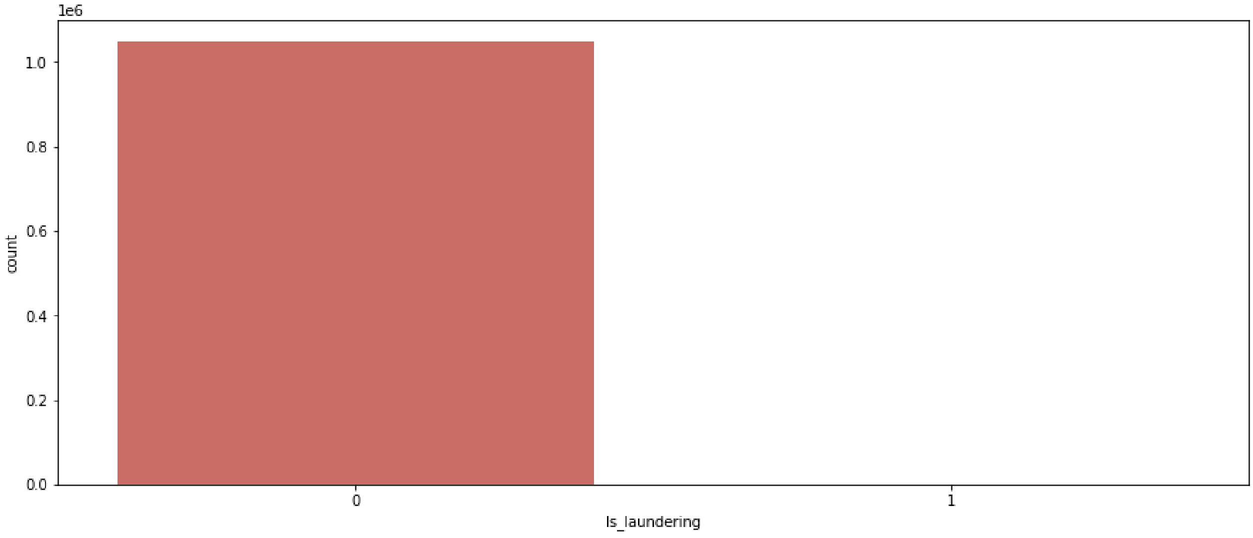
In [22]:
```
for i in discrete:
    print(i, ':')
    print(df[i].unique())
    print()
```

```
Is_laundering :
[0 1]
```

In [23]:
```
for i in discrete:
    print(i, ':')
    print(df[i].value_counts())
    print()
```

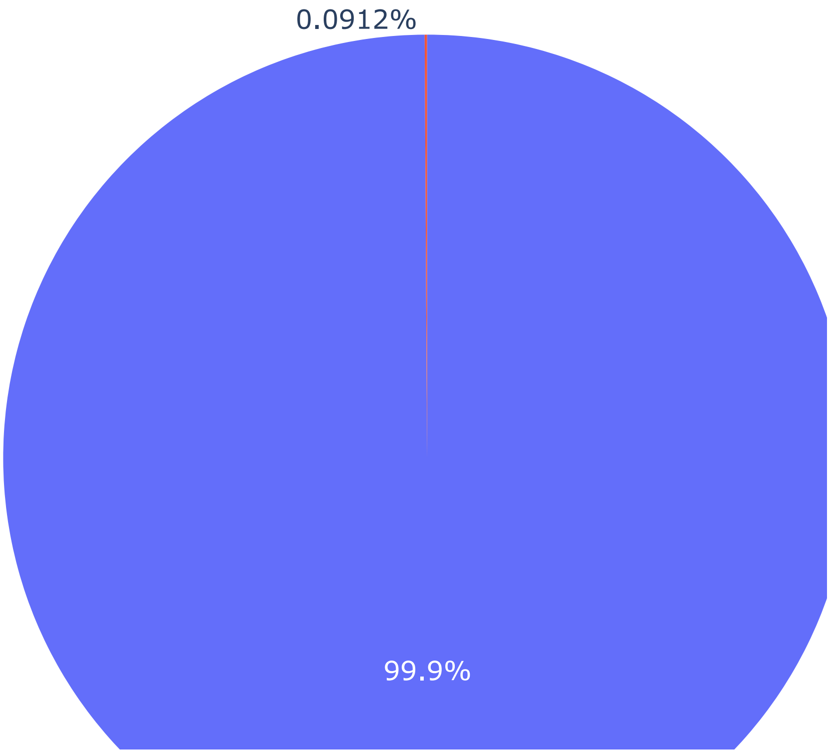
```
Is_laundering :
0    1047619
1        956
Name: Is_laundering, dtype: int64
```

In [24]:
```
for i in discrete:
    plt.figure(figsize=(15,6))
    sns.countplot(df[i], data = df, palette='hls')
    plt.show()
```

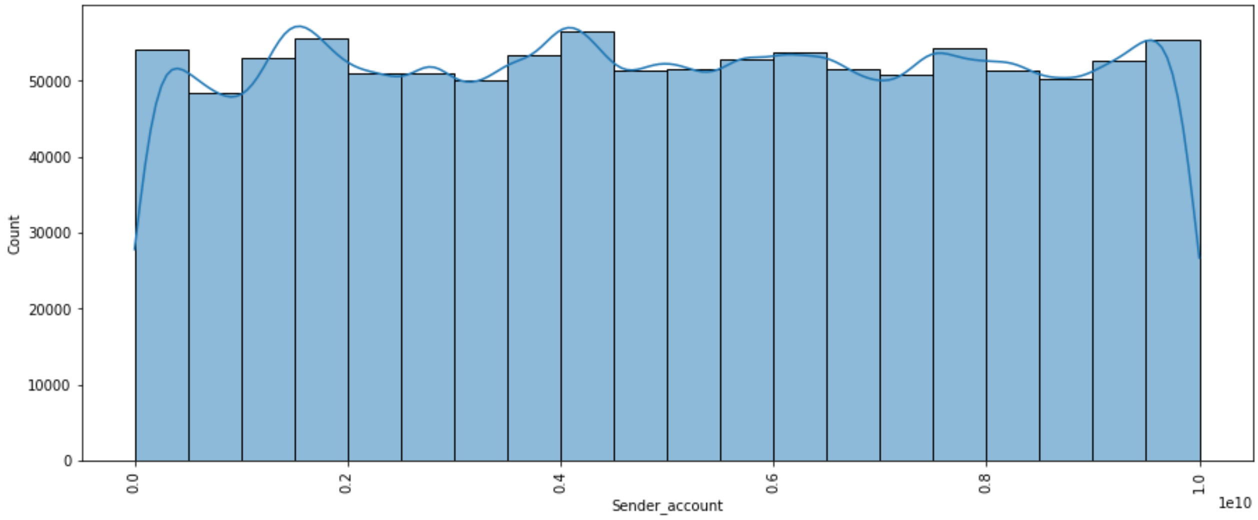


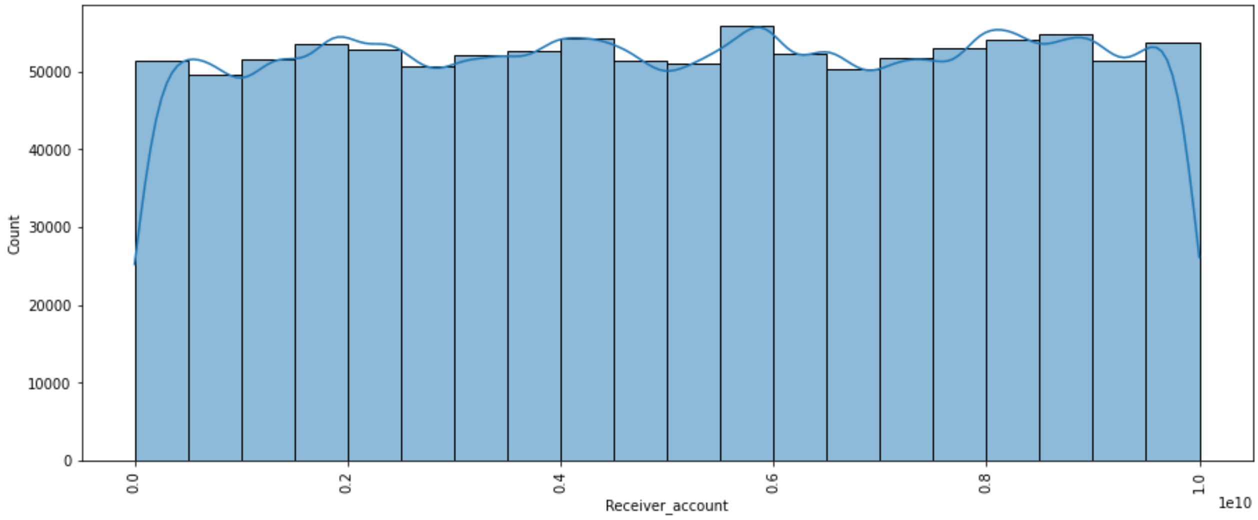
In [25]:
```
for i in discrete:
    fig = px.pie(df, names=i)
    print('Pieplot for:', i)
    fig.show()
```

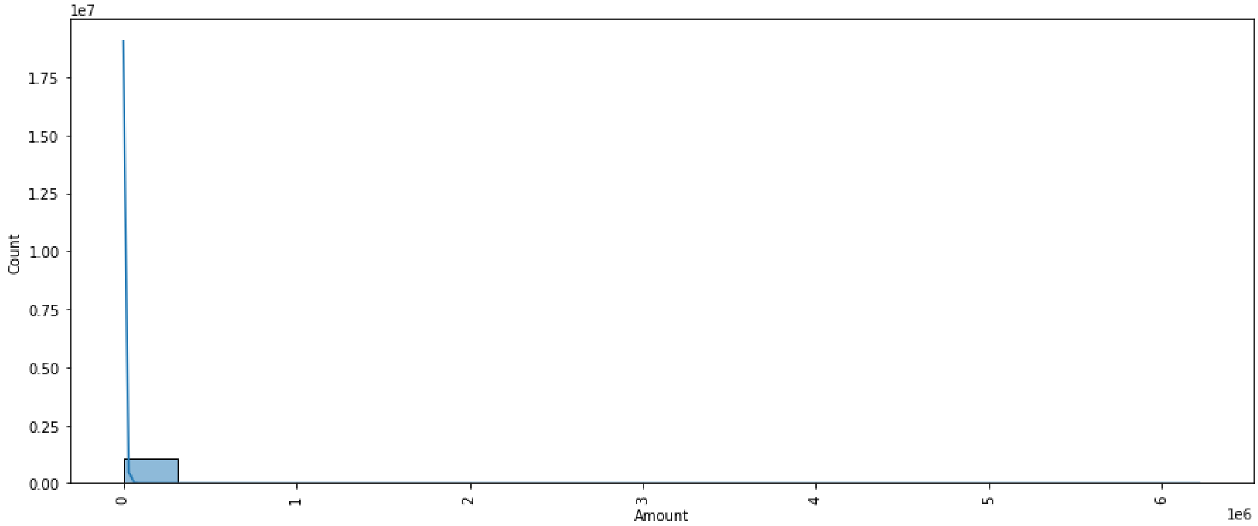
Pieplot for: Is_laundering



In [26]:
```
for i in continuous:
    plt.figure(figsize=(15,6))
    sns.histplot(df[i], bins = 20, kde = True, palette='hls')
    plt.xticks(rotation = 90)
    plt.show()
```

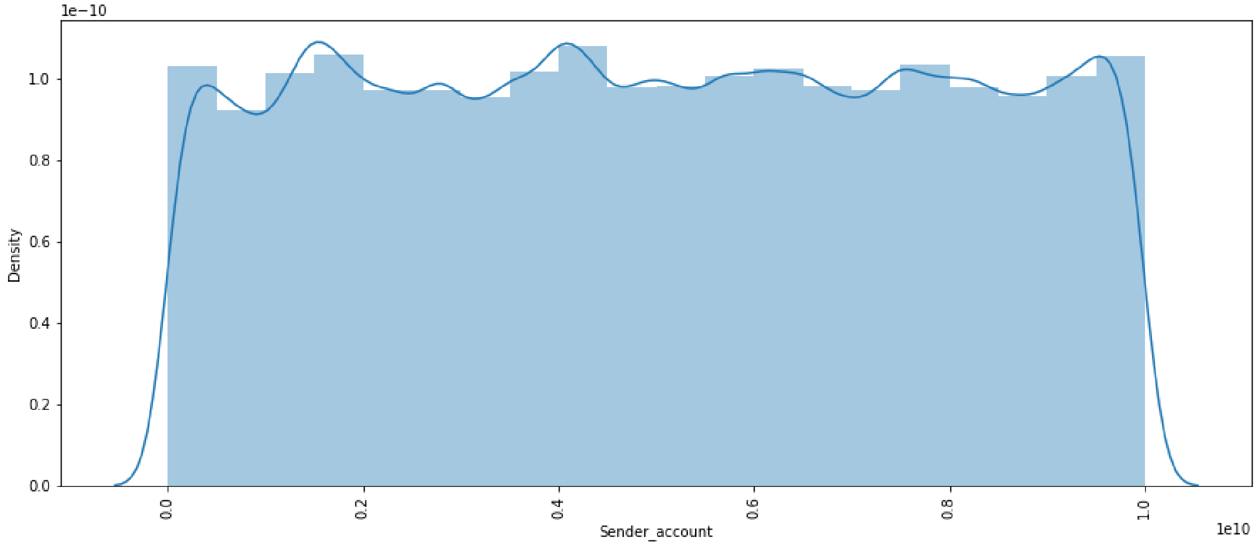


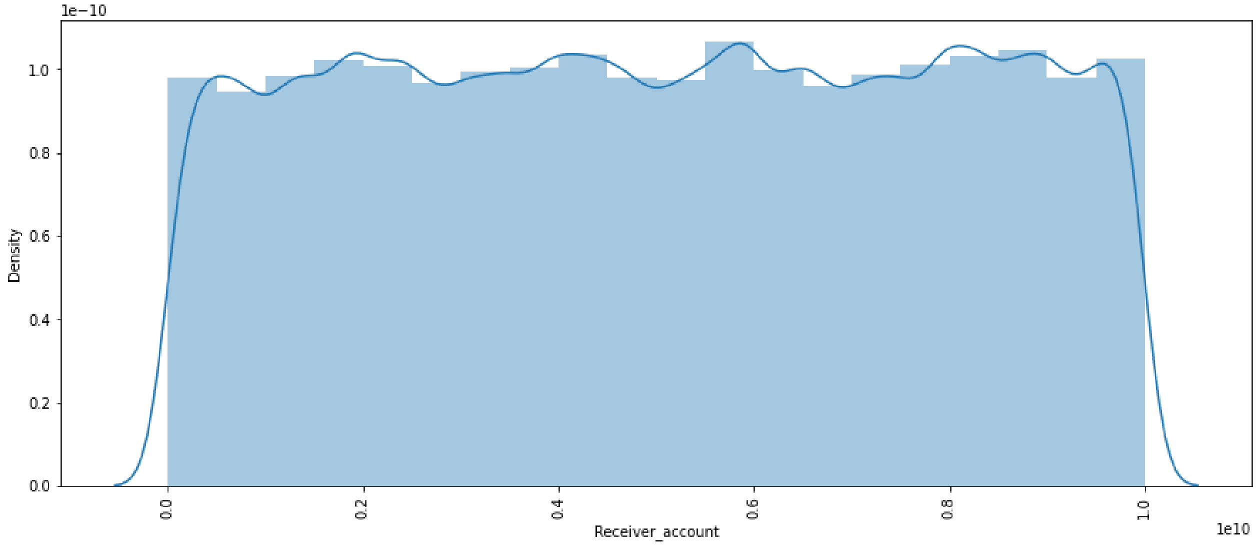


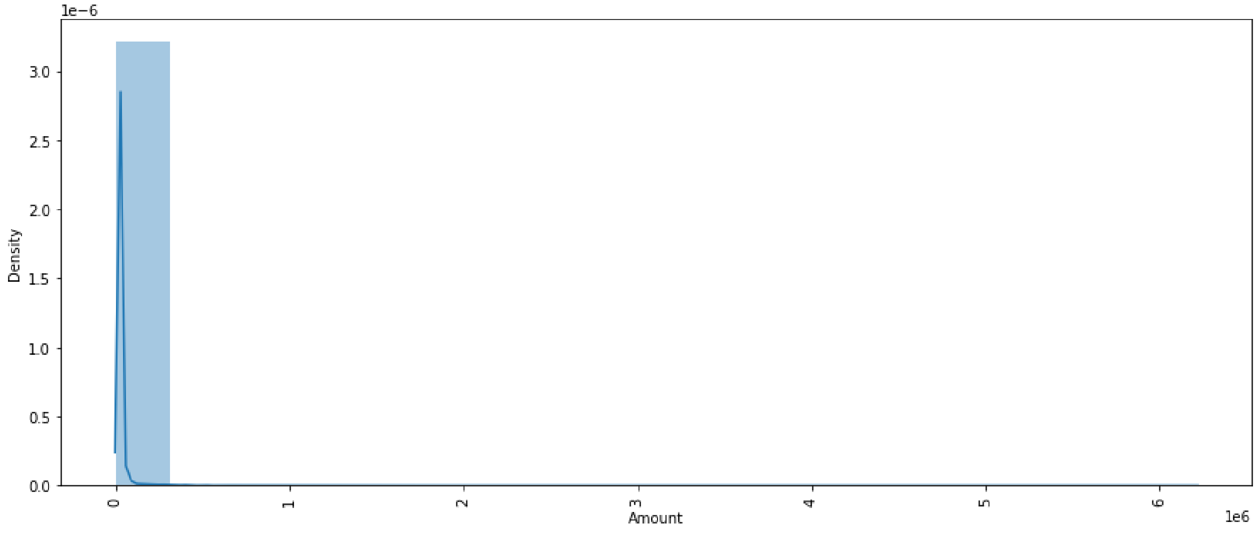


In [27]:

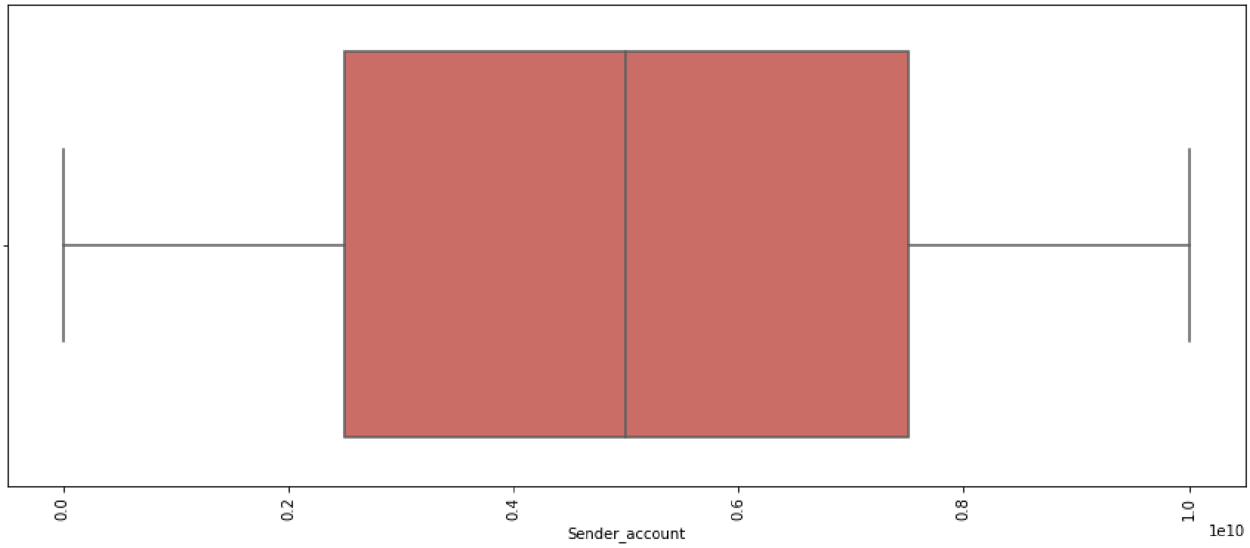
```
for i in continuous:
    plt.figure(figsize=(15,6))
    sns.distplot(df[i], bins = 20, kde = True)
    plt.xticks(rotation = 90)
    plt.show()
```

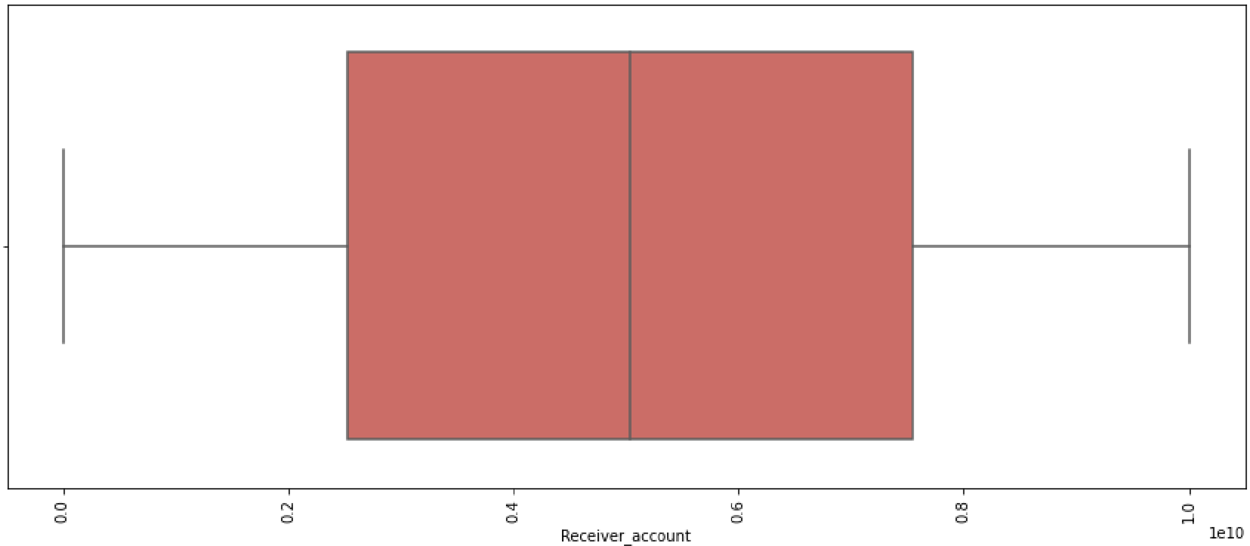


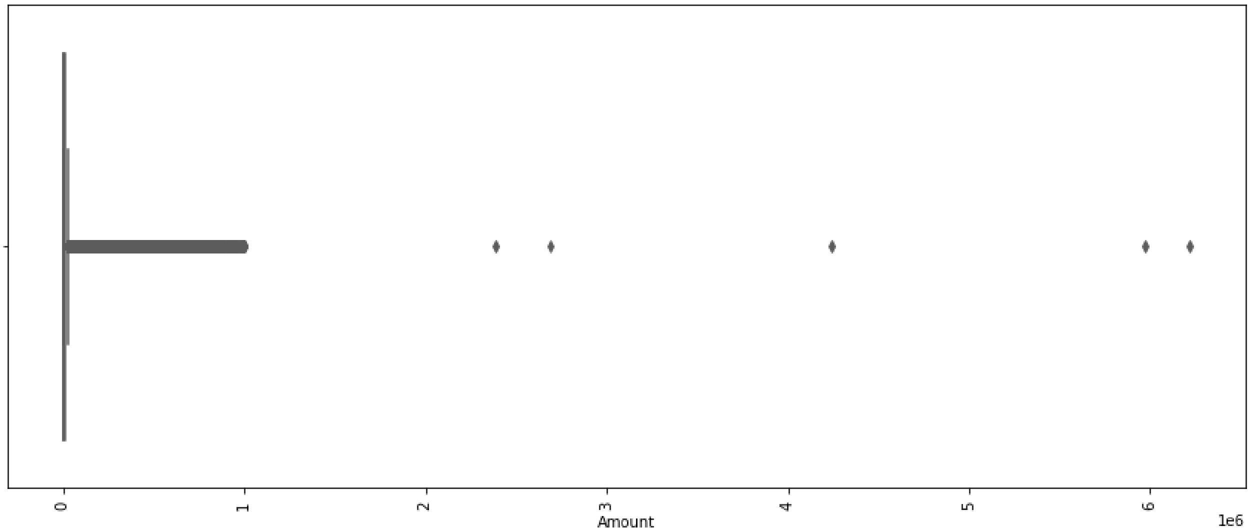




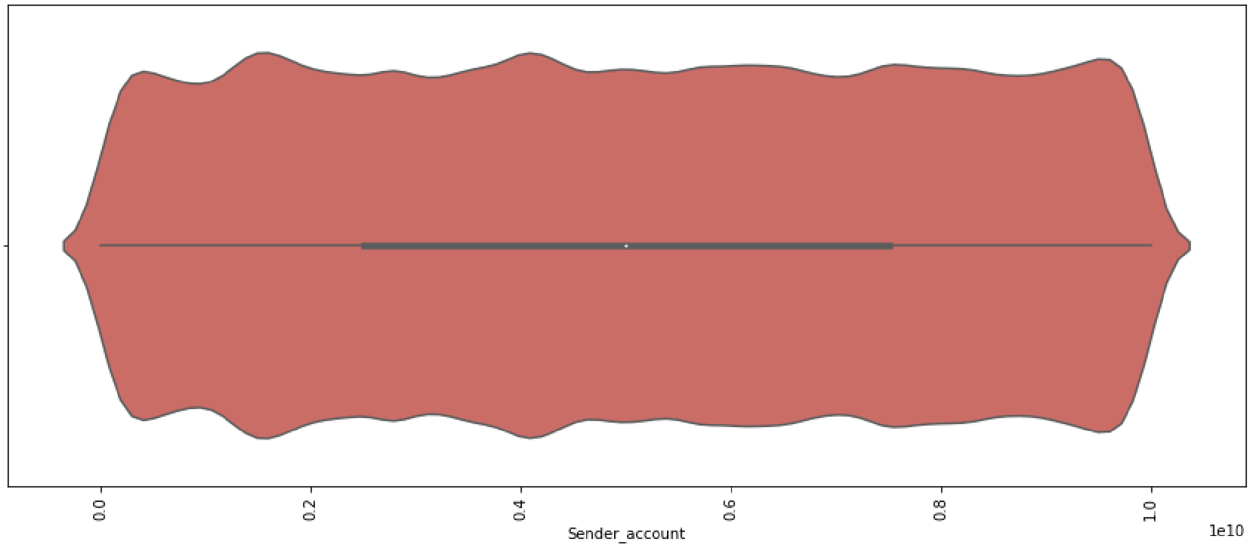
In [28]:
```python
for i in continuous:
    plt.figure(figsize=(15,6))
    sns.boxplot(i, data = df, palette='hls')
    plt.xticks(rotation = 90)
    plt.show()
```

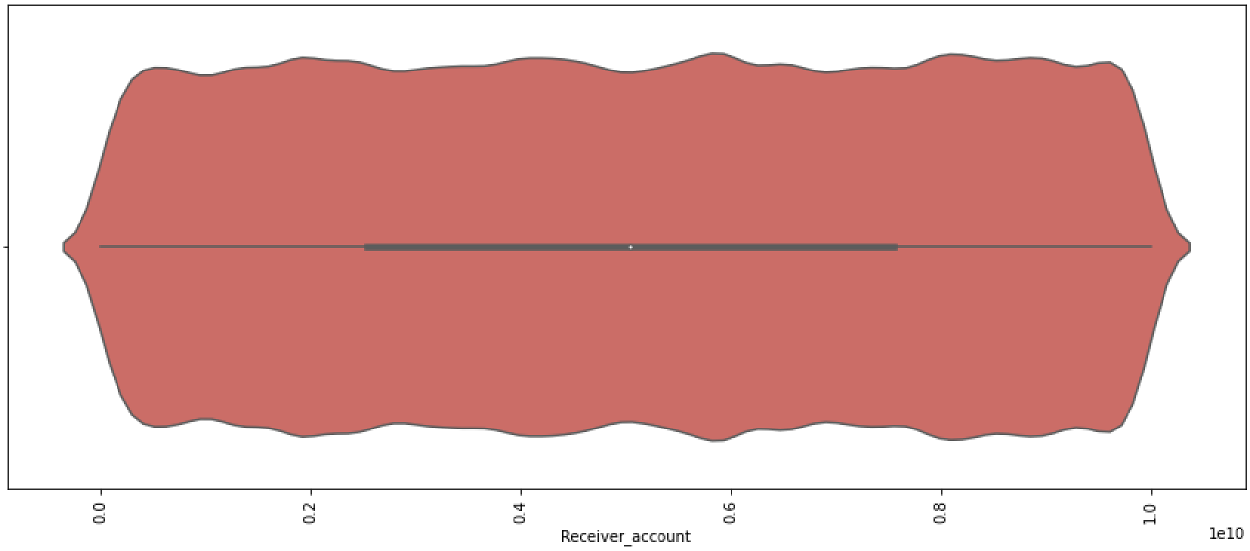


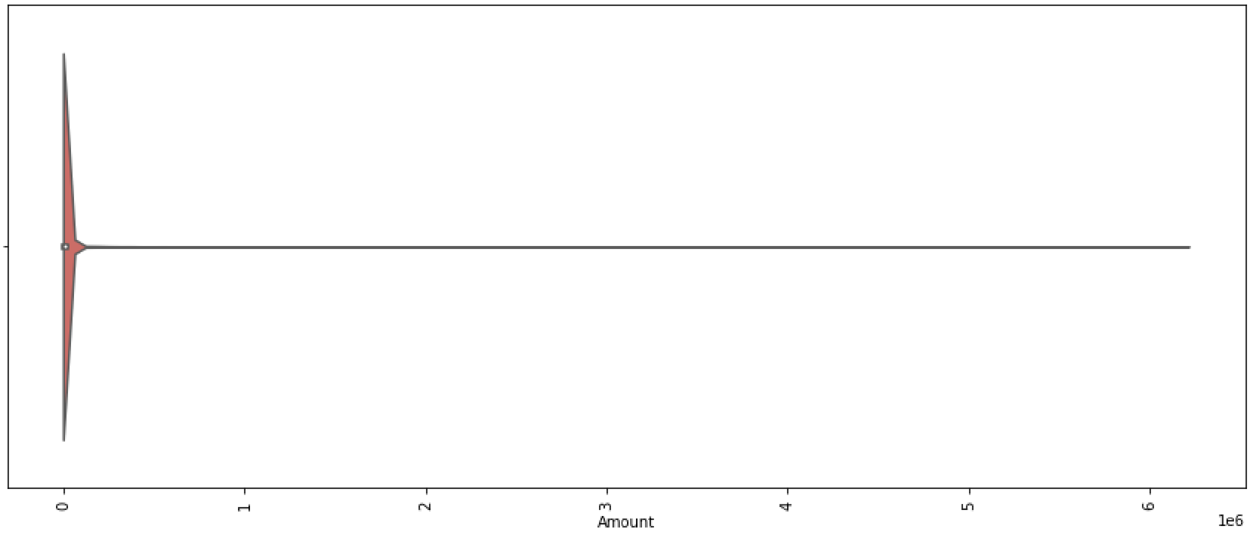




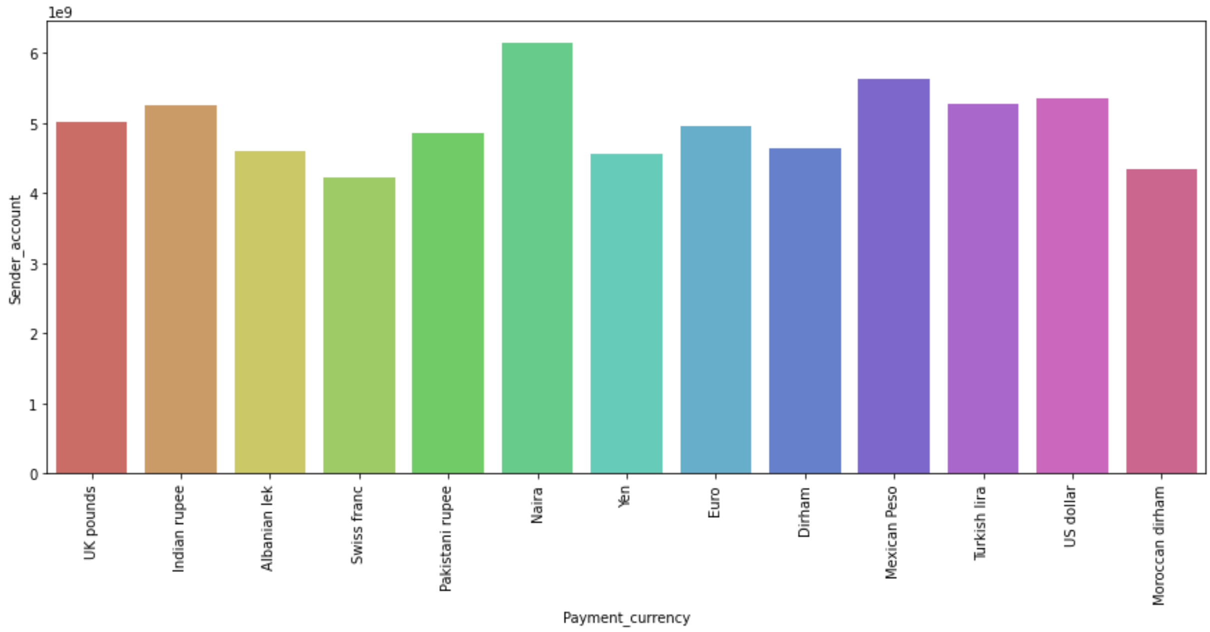
In [29]:
```
for i in continuous:
    plt.figure(figsize=(15,6))
    sns.violinplot(i, data = df, palette='hls')
    plt.xticks(rotation = 90)
    plt.show()
```



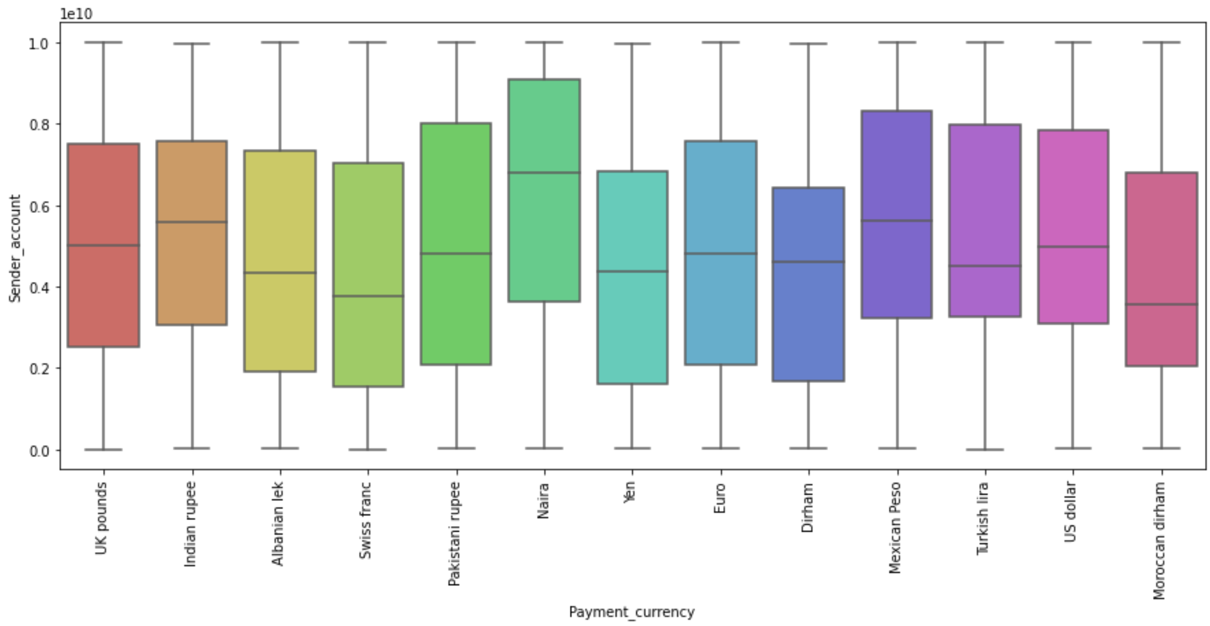




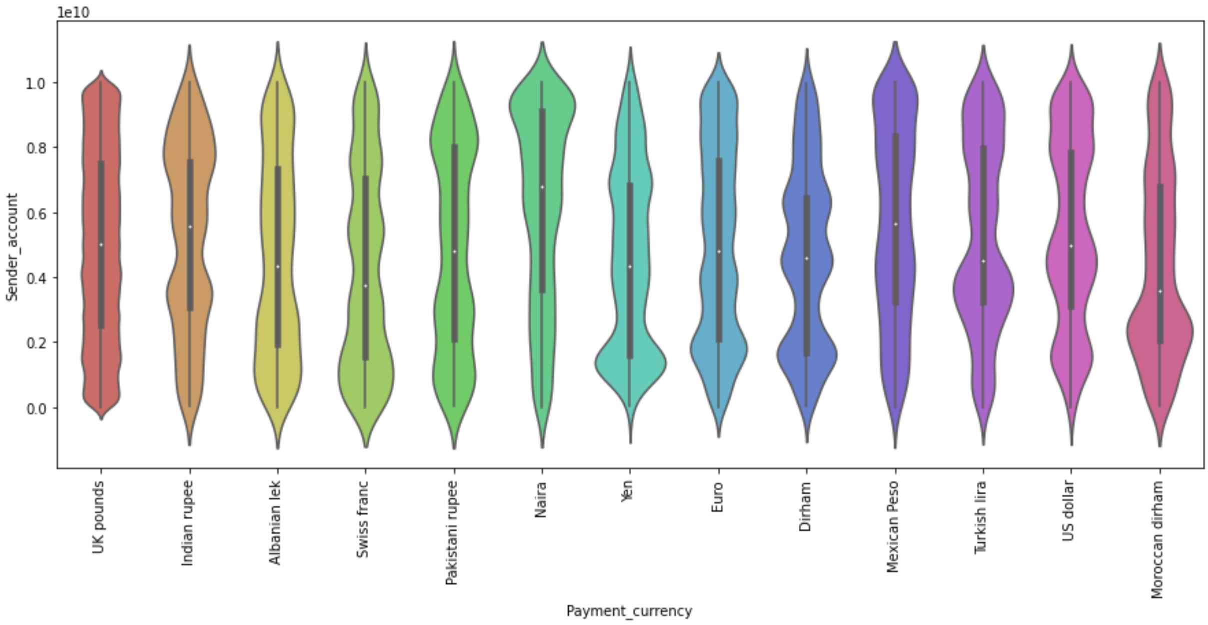
In [30]:
```python
for i in categorical:
    for j in continuous:
        plt.figure(figsize=(15,6))
        sns.barplot(x = i, y = j, data = df, ci = None, palette='hls')
        plt.xticks(rotation = 90)
        plt.show()
```



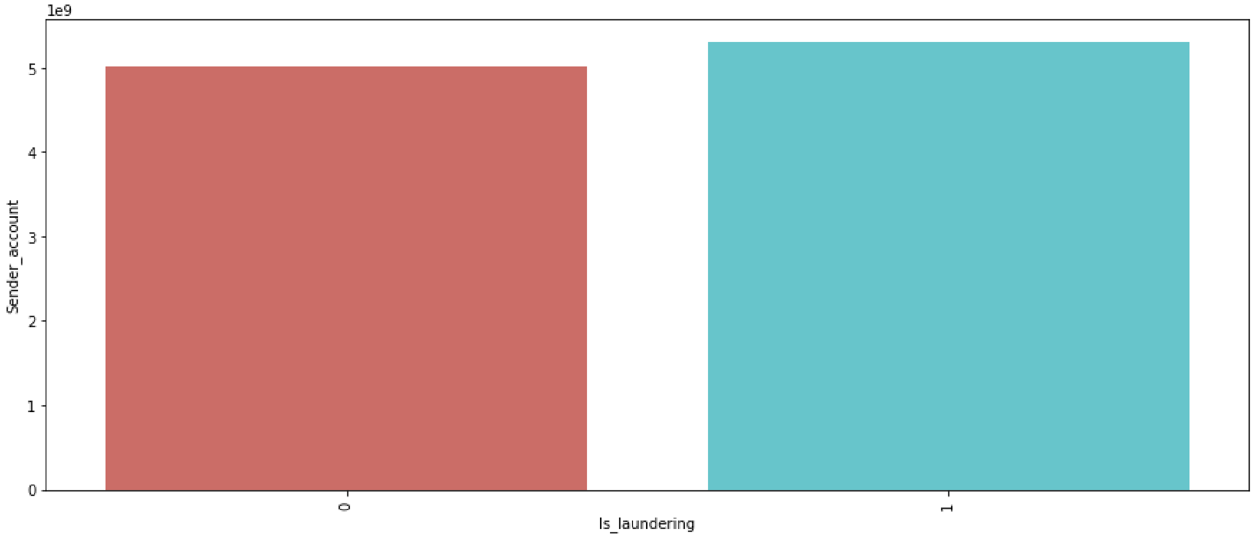
In [31]:
```python
for i in categorical:
    for j in continuous:
        plt.figure(figsize=(15,6))
        sns.boxplot(x = i, y = j, data = df, palette='hls')
        plt.xticks(rotation = 90)
        plt.show()
```

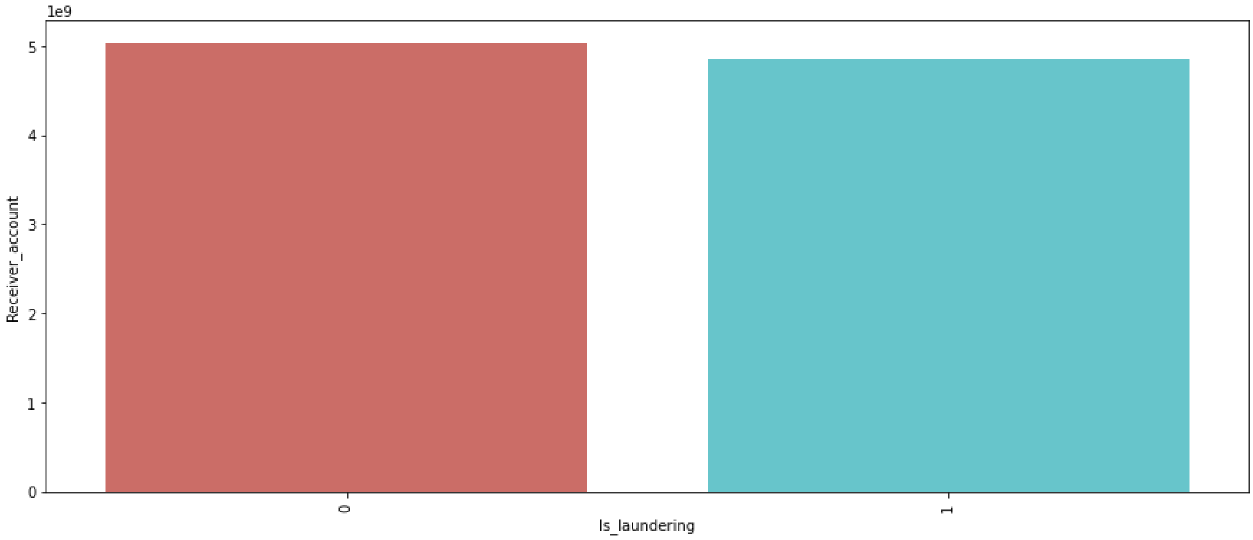


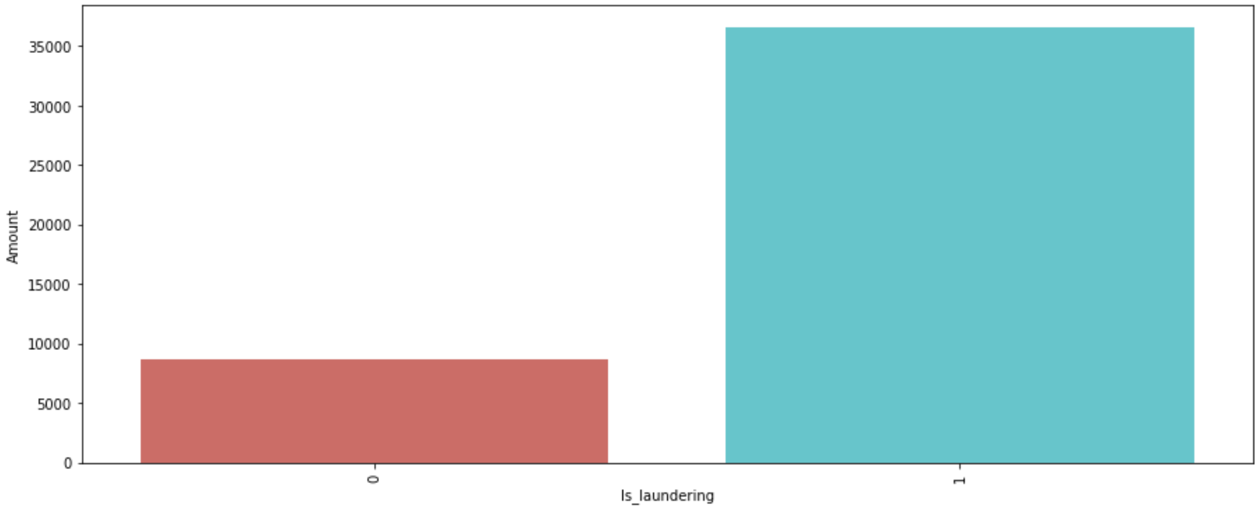
In [32]:
```python
for i in categorical:
    for j in continuous:
        plt.figure(figsize=(15,6))
        sns.violinplot(x = i, y = j, data = df, palette='hls')
        plt.xticks(rotation = 90)
        plt.show()
```



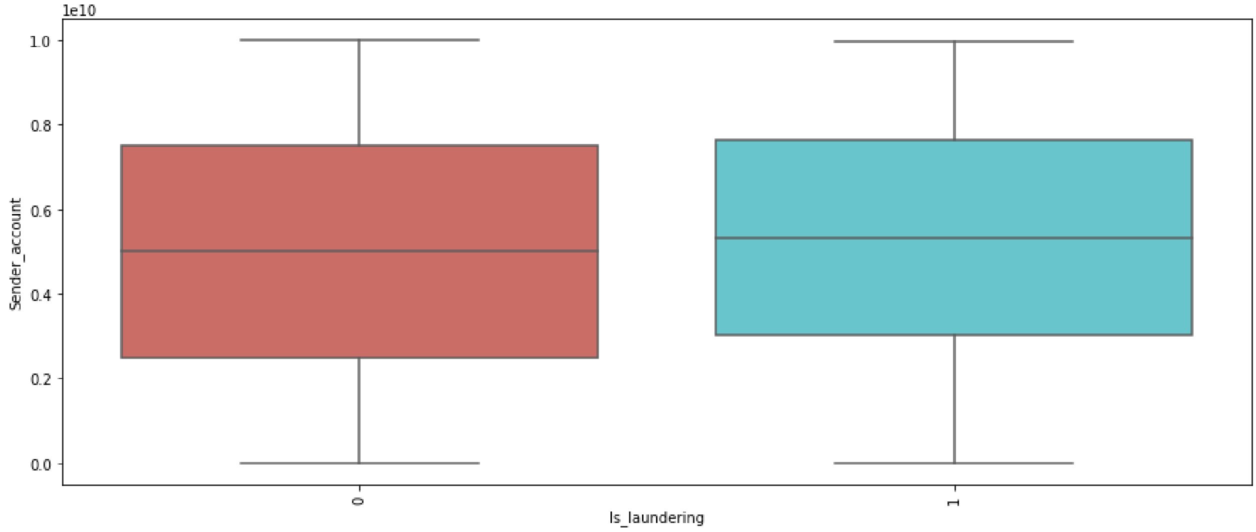
In [33]:
```
for i in discrete:
    for j in continuous:
        plt.figure(figsize=(15,6))
        sns.barplot(x = i, y = j, data = df, ci = None, palette='hls')
        plt.xticks(rotation = 90)
        plt.show()
```

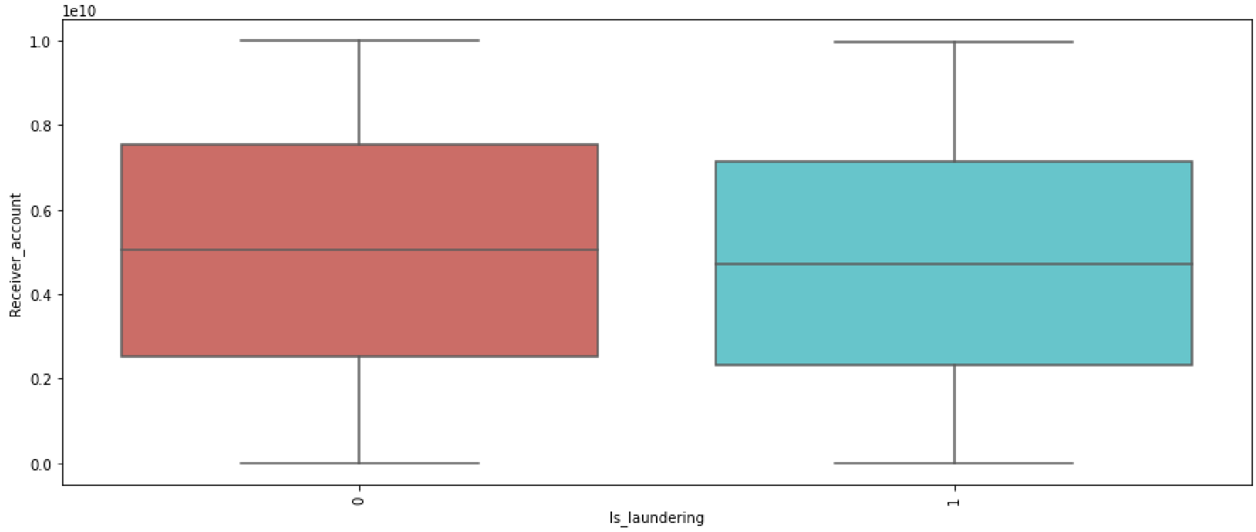


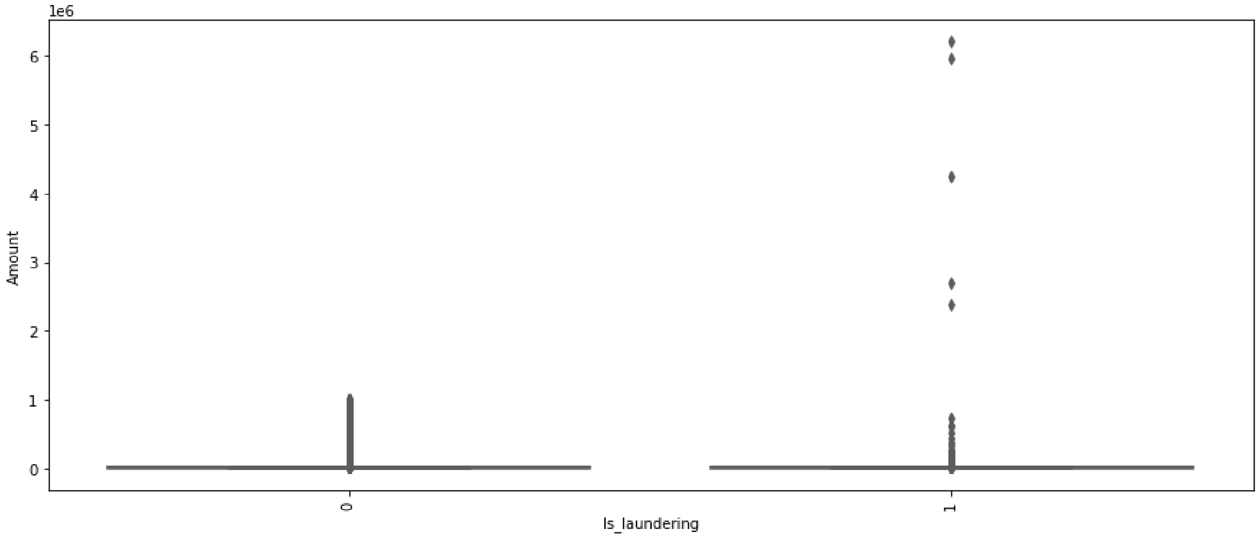




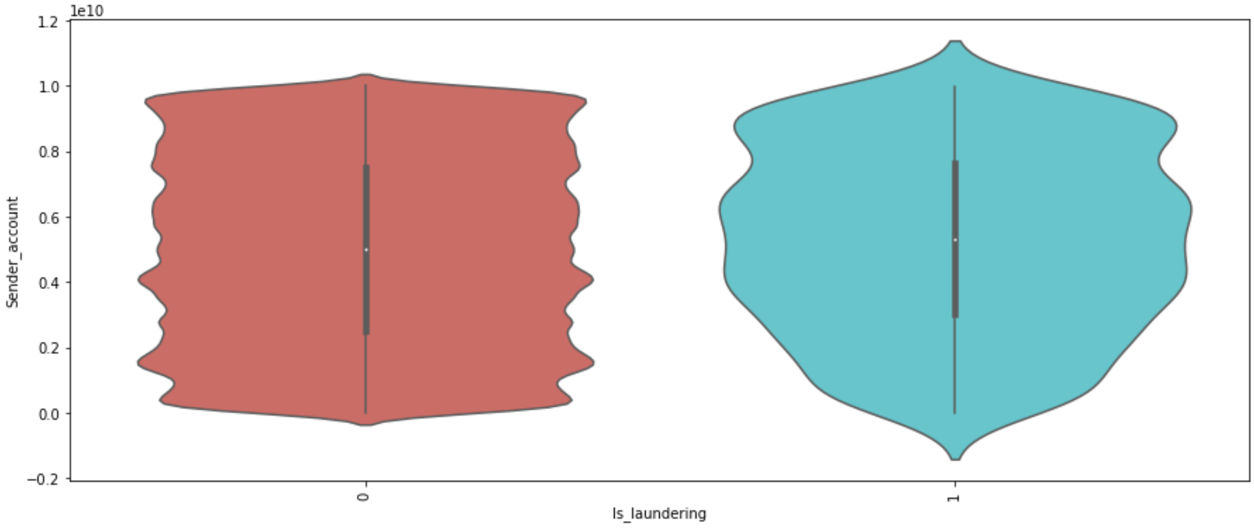
In [34]:
```
for i in discrete:
    for j in continuous:
        plt.figure(figsize=(15,6))
        sns.boxplot(x = i, y = j, data = df, palette='hls')
        plt.xticks(rotation = 90)
        plt.show()
```

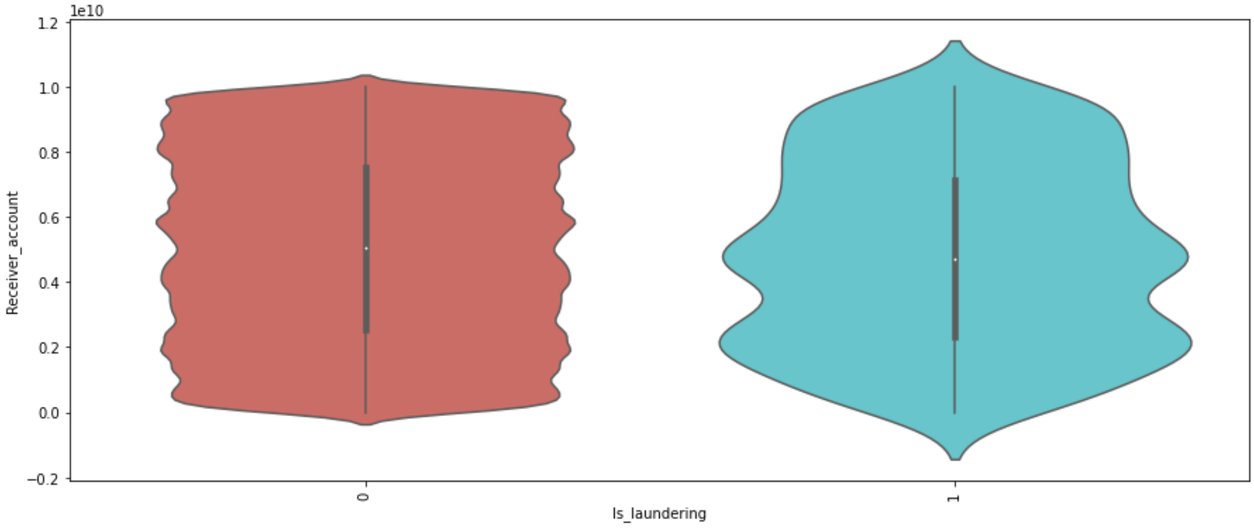




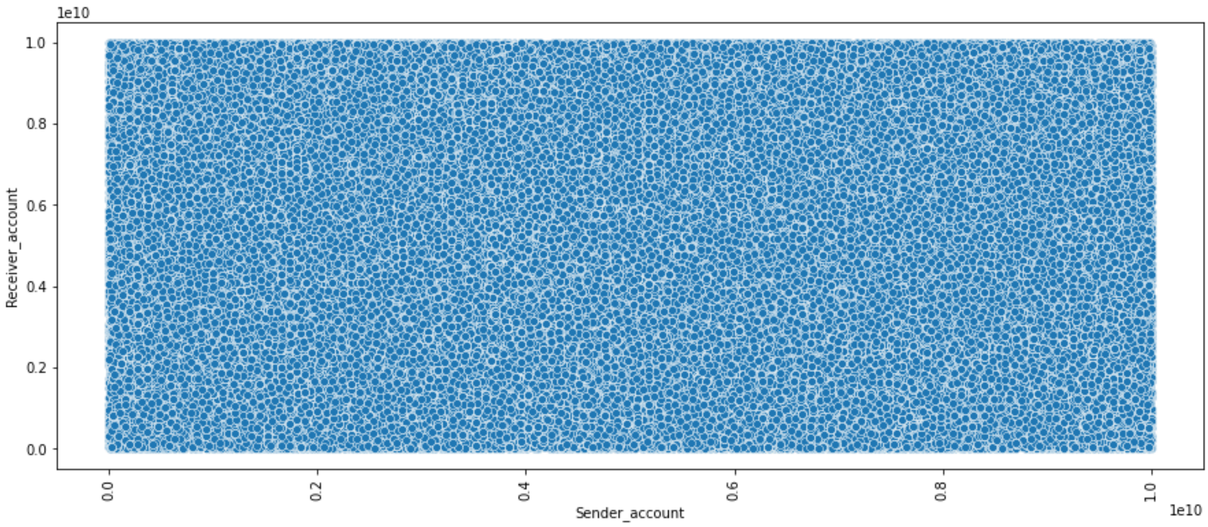


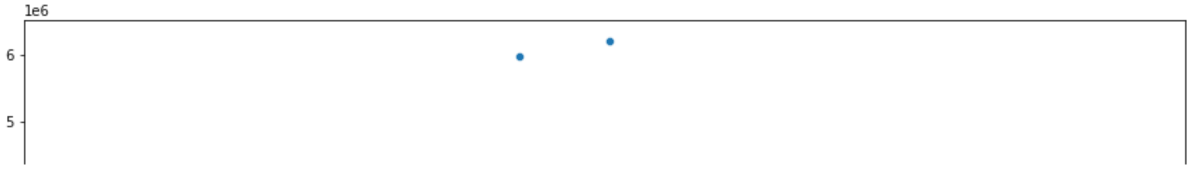
In [35]:
```python
for i in discrete:
    for j in continuous:
        plt.figure(figsize=(15,6))
        sns.violinplot(x = i, y = j, data = df, palette='hls')
        plt.xticks(rotation = 90)
        plt.show()
```





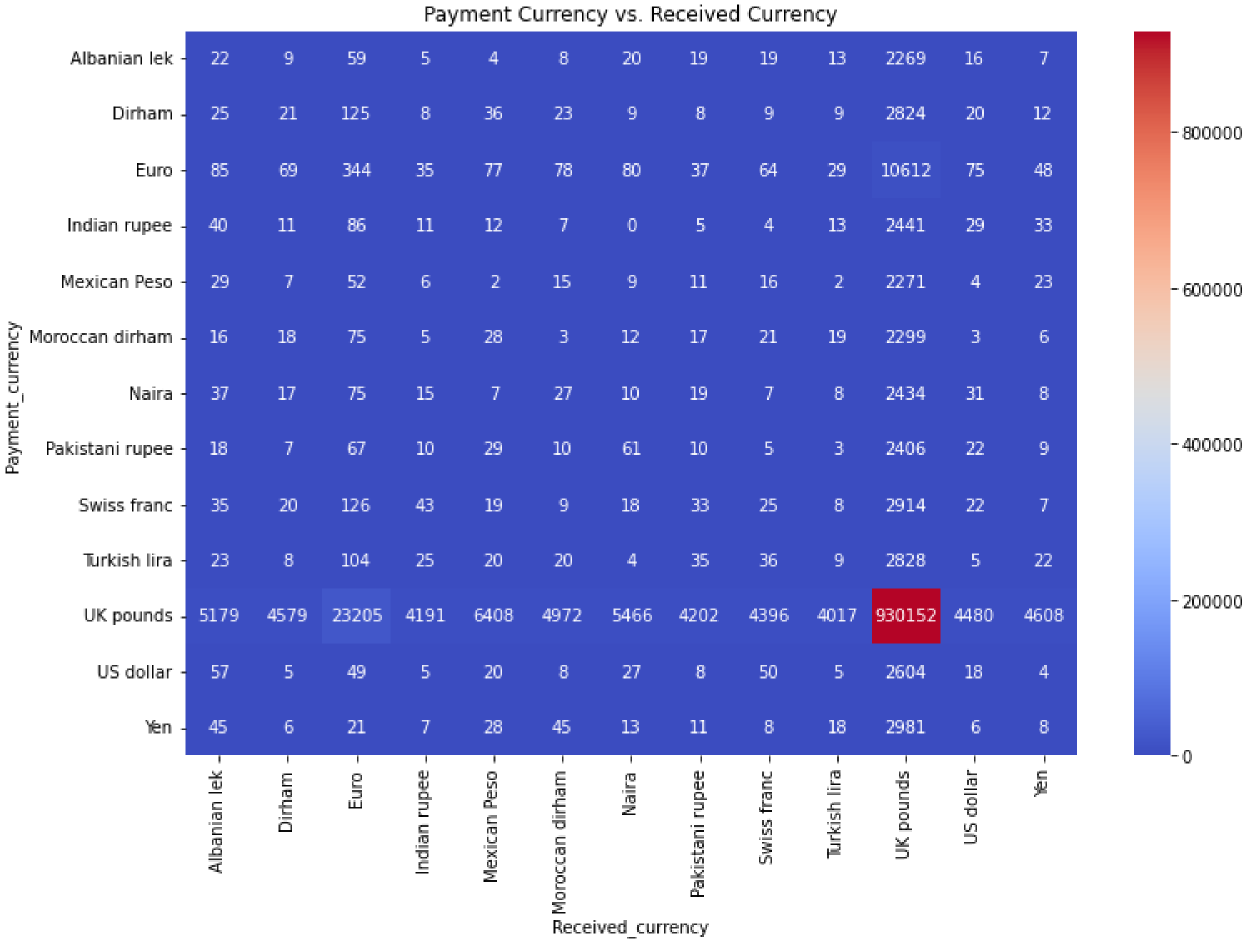
In [36]:
```
for i in continuous:
    for j in continuous:
        if i != j:
            plt.figure(figsize=(15,6))
            sns.scatterplot(x = df[i], y = df[j], data = df, palette = 'hls
            plt.xticks(rotation = 90)
            plt.show()
```



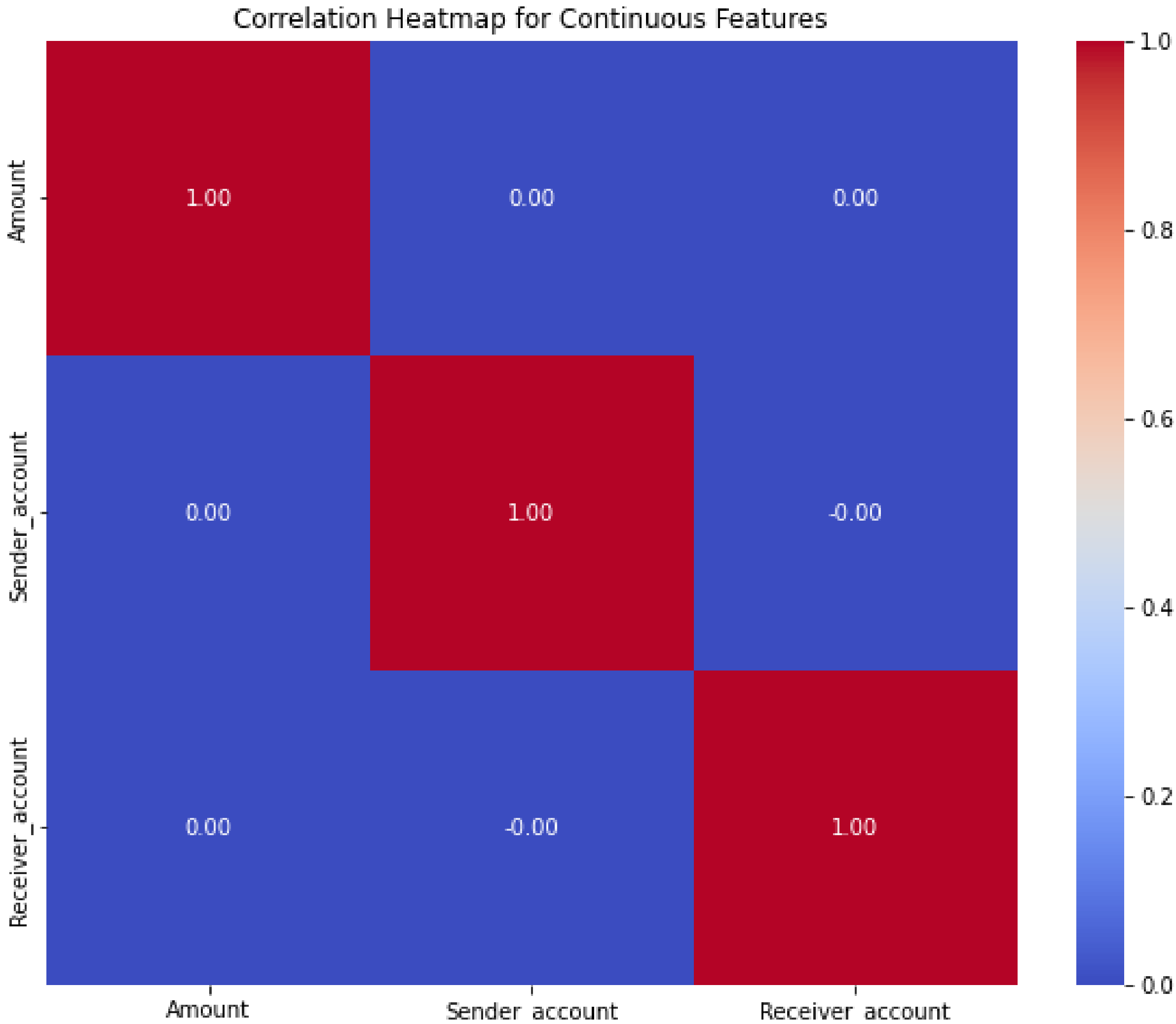


In [37]:

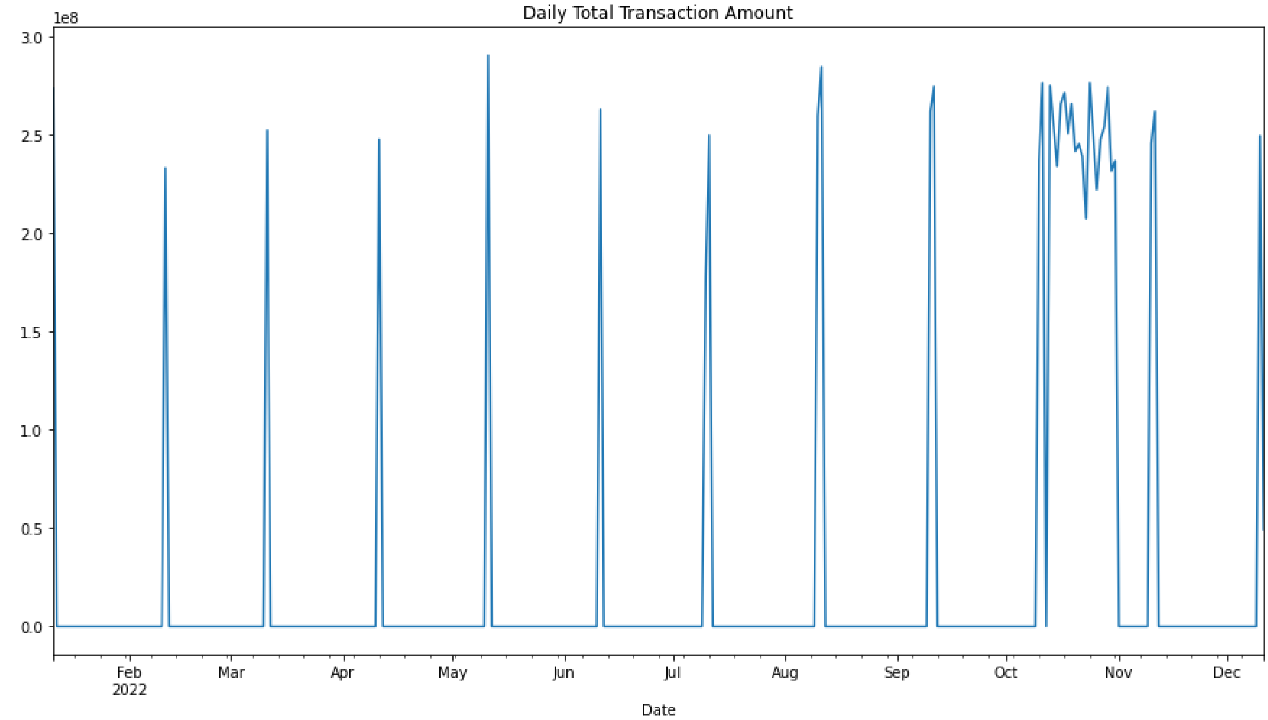
```
payment_received = pd.crosstab(df['Payment_currency'], df['Received_currenc
plt.figure(figsize=(12, 8))
sns.heatmap(payment_received, annot=True, cmap='coolwarm', fmt='d')
plt.title('Payment Currency vs. Received Currency')
plt.show()
```



In [38]:
```
continuous_features = ['Amount', 'Sender_account', 'Receiver_account', 'Time
plt.figure(figsize=(10, 8))
sns.heatmap(df[continuous_features].corr(), annot=True, cmap='coolwarm', fmt
plt.title('Correlation Heatmap for Continuous Features')
plt.show()
```



In [39]:
```
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
plt.figure(figsize=(15, 8))
df['Amount'].resample('D').sum().plot(title='Daily Total Transaction Amount
plt.show()
```



In [40]:
```
df
```

Out[40]:

	Time	Sender_account	Receiver_account	Amount	Payment_currency	Received_cu
Date						
2022-07-10	10:35:19	8724731955	2769355426	1459.15	UK pounds	UK
2022-07-10	10:35:20	1491989064	8401255335	6019.64	UK pounds	
2022-07-10	10:35:20	287305149	4404767002	14328.44	UK pounds	UK
2022-07-10	10:35:21	5376652437	9600420220	11895.00	UK pounds	UK
2022-07-10	10:35:21	9614186178	3803336972	115.25	UK pounds	UK
...	...	...	...	...	...	
2022-12-11	09:21:10	3848621169	3388139373	21559.31	UK pounds	Moroccan
2022-12-11	09:21:12	8276335513	7220829571	14590.90	UK pounds	UK
2022-12-11	09:21:13	3014566949	6653549199	7141.85	UK pounds	UK
2022-12-11	09:21:15	1426173499	3569198271	14675.91	UK pounds	UK
2022-12-11	09:21:16	7798805872	9278506258	32473.03	UK pounds	UK

1048575 rows × 11 columns

In [41]:
```
df.reset_index(inplace=True)
```

In [42]:
```
df
```

Out[42]:

	Date	Time	Sender_account	Receiver_account	Amount	Payment_currency	Re
0	2022-07-10	10:35:19	8724731955	2769355426	1459.15	UK pounds	
1	2022-07-10	10:35:20	1491989064	8401255335	6019.64	UK pounds	
2	2022-07-10	10:35:20	287305149	4404767002	14328.44	UK pounds	
3	2022-07-10	10:35:21	5376652437	9600420220	11895.00	UK pounds	
4	2022-07-10	10:35:21	9614186178	3803336972	115.25	UK pounds	
...	...	...	...	...	...	...	
1048570	2022-12-11	09:21:10	3848621169	3388139373	21559.31	UK pounds	
1048571	2022-12-11	09:21:12	8276335513	7220829571	14590.90	UK pounds	
1048572	2022-12-11	09:21:13	3014566949	6653549199	7141.85	UK pounds	
1048573	2022-12-11	09:21:15	1426173499	3569198271	14675.91	UK pounds	
1048574	2022-12-11	09:21:16	7798805872	9278506258	32473.03	UK pounds	

1048575 rows × 12 columns

In [43]:
```
df = pd.get_dummies(df, columns=['Payment_currency', 'Received_currency', 'I
```

In [44]:
```
df
```

Out[44]:

	Date	Time	Sender_account	Receiver_account	Amount	Sender_bank_location
0	2022-07-10	10:35:19	8724731955	2769355426	1459.15	UK
1	2022-07-10	10:35:20	1491989064	8401255335	6019.64	UK
2	2022-07-10	10:35:20	287305149	4404767002	14328.44	UK
3	2022-07-10	10:35:21	5376652437	9600420220	11895.00	UK
4	2022-07-10	10:35:21	9614186178	3803336972	115.25	UK
...	...	...	...	...	...	...
1048570	2022-12-11	09:21:10	3848621169	3388139373	21559.31	UK
1048571	2022-12-11	09:21:12	8276335513	7220829571	14590.90	UK
1048572	2022-12-11	09:21:13	3014566949	6653549199	7141.85	UK
1048573	2022-12-11	09:21:15	1426173499	3569198271	14675.91	UK
1048574	2022-12-11	09:21:16	7798805872	9278506258	32473.03	UK

1048575 rows × 40 columns

In [45]:
```
df = pd.get_dummies(df, columns=['Sender_bank_location' , 'Receiver_bank_loc
```

In [46]:
```
df
```

Out[46]:

	Date	Time	Sender_account	Receiver_account	Amount	Is_laundering	Paymer
0	2022-07-10	10:35:19	8724731955	2769355426	1459.15	0	
1	2022-07-10	10:35:20	1491989064	8401255335	6019.64	0	
2	2022-07-10	10:35:20	287305149	4404767002	14328.44	0	
3	2022-07-10	10:35:21	5376652437	9600420220	11895.00	0	
4	2022-07-10	10:35:21	9614186178	3803336972	115.25	0	
...	...	...	...	...	...	...	
1048570	2022-12-11	09:21:10	3848621169	3388139373	21559.31	0	
1048571	2022-12-11	09:21:12	8276335513	7220829571	14590.90	0	
1048572	2022-12-11	09:21:13	3014566949	6653549199	7141.85	0	
1048573	2022-12-11	09:21:15	1426173499	3569198271	14675.91	0	
1048574	2022-12-11	09:21:16	7798805872	9278506258	32473.03	0	

1048575 rows × 98 columns

In [47]:
```
df.drop(['Date', 'Time', 'Sender_account', 'Receiver_account'], axis=1, inp
```

In [48]:
```
df
```

Out[48]:

	Amount	Is_laundering	Payment_currency_Dirham	Payment_currency_Euro	Paymer
0	1459.15	0	0	0	
1	6019.64	0	0	0	
2	14328.44	0	0	0	
3	11895.00	0	0	0	
4	115.25	0	0	0	
...	...	...	...	...	
1048570	21559.31	0	0	0	
1048571	14590.90	0	0	0	
1048572	7141.85	0	0	0	
1048573	14675.91	0	0	0	
1048574	32473.03	0	0	0	

1048575 rows × 94 columns

In [49]:
```
from sklearn.model_selection import train_test_split

X = df.drop('Is_laundering', axis=1)
y = df['Is_laundering']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, str
```

In [50]:
```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)

X_train, y_train = ros.fit_resample(X_train, y_train)
```

In [51]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion
```

In [52]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [53]:
```
dt_model = DecisionTreeClassifier(random_state=42)
```

In [54]:
```
dt_model.fit(X_train, y_train)
```

Out[54]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

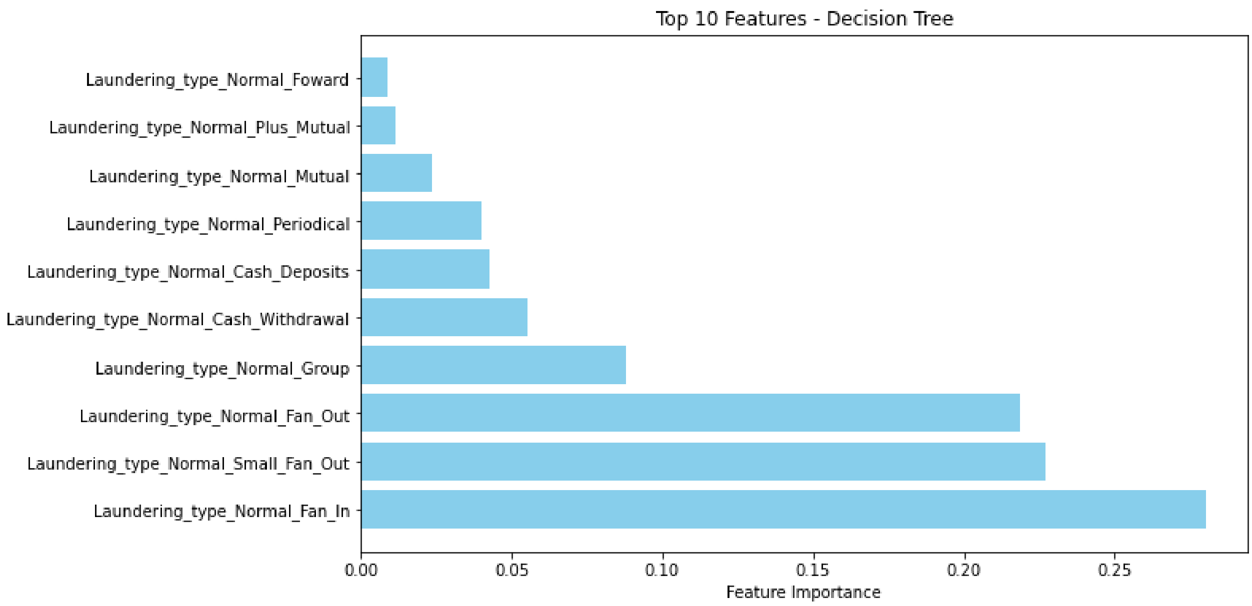
In [55]:
```python
feature_importances = dt_model.feature_importances_
```

In [56]:
```python
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': fe
```

In [57]:
```python
feature_importance_df = feature_importance_df.sort_values(by='Importance', a
```

In [58]:
```python
top_10_features = feature_importance_df.head(10)
print("Top 10 Features:")
print(top_10_features)

plt.figure(figsize=(10, 6))
plt.barh(top_10_features['Feature'], top_10_features['Importance'], color='s
plt.xlabel('Feature Importance')
plt.title('Top 10 Features - Decision Tree')
plt.show()
```

```
Top 10 Features:
                                 Feature  Importance
78           Laundering_type_Normal_Fan_In    0.280115
85     Laundering_type_Normal_Small_Fan_Out    0.226929
79          Laundering_type_Normal_Fan_Out    0.218850
81            Laundering_type_Normal_Group    0.087827
77  Laundering_type_Normal_Cash_Withdrawal    0.055104
76    Laundering_type_Normal_Cash_Deposits    0.042784
83       Laundering_type_Normal_Periodical    0.040157
82           Laundering_type_Normal_Mutual    0.023790
84      Laundering_type_Normal_Plus_Mutual    0.011259
80           Laundering_type_Normal_Foward    0.008721
```



In [59]:
```python
y_pred = dt_model.predict(X_test)
```

In [60]:
```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{classification_rep}')
```

```
Accuracy: 1.00
Confusion Matrix:
[[209524      0]
 [     0    191]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209524
           1       1.00      1.00      1.00       191

    accuracy                           1.00    209715
   macro avg       1.00      1.00      1.00    209715
weighted avg       1.00      1.00      1.00    209715
```

In [61]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [62]:
```
rf_classifier = RandomForestClassifier(n_estimators=10, random_state=42)
```

In [63]:
```
rf_classifier.fit(X_train, y_train)
```

Out[63]:
```
▾                    RandomForestClassifier
RandomForestClassifier(n_estimators=10, random_state=42)
```

In [64]:
```
y_pred = rf_classifier.predict(X_test)
```

In [65]:
```
accuracy_rf = accuracy_score(y_test, y_pred)
conf_matrix_rf = confusion_matrix(y_test, y_pred)
classification_rep_rf = classification_report(y_test, y_pred)

print(f'Random Forest Test Accuracy: {accuracy_rf:.2f}')
print(f'Random Forest Confusion Matrix:\n{conf_matrix_rf}')
print(f'Random Forest Classification Report:\n{classification_rep_rf}')
```

```
Random Forest Test Accuracy: 1.00
Random Forest Confusion Matrix:
[[209524      0]
 [     8    183]]
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209524
           1       1.00      0.96      0.98       191

    accuracy                           1.00    209715
   macro avg       1.00      0.98      0.99    209715
weighted avg       1.00      1.00      1.00    209715
```

In [66]:
```
top_feature_names = [
    'Laundering_type_Normal_Fan_In',
    'Laundering_type_Normal_Small_Fan_Out',
    'Laundering_type_Normal_Fan_Out',
    'Laundering_type_Normal_Group',
    'Laundering_type_Normal_Cash_Withdrawal',
    'Laundering_type_Normal_Cash_Deposits',
    'Laundering_type_Normal_Periodical',
    'Laundering_type_Normal_Mutual',
    'Laundering_type_Normal_Plus_Mutual',
    'Laundering_type_Normal_Foward'
]
```

In [67]:
```
X_train = X_train[top_feature_names]
X_test = X_test[top_feature_names]
```

In [68]:
```
dt_model.fit(X_train, y_train)
```

Out[68]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [69]:
```
y_pred = dt_model.predict(X_test)
```

In [70]:
```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{classification_rep}')
```

```
Accuracy: 1.00
Confusion Matrix:
[[209088    436]
 [     0    191]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209524
           1       0.30      1.00      0.47       191

    accuracy                           1.00    209715
   macro avg       0.65      1.00      0.73    209715
weighted avg       1.00      1.00      1.00    209715
```

In [71]:
```
rf_classifier.fit(X_train, y_train)
```

Out[71]:
```
▾ RandomForestClassifier
RandomForestClassifier(n_estimators=10, random_state=42)
```

In [72]:
```
y_pred = rf_classifier.predict(X_test)
```

In [73]:
```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{classification_rep}')
```

```
Accuracy: 1.00
Confusion Matrix:
[[209088    436]
 [     0    191]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209524
           1       0.30      1.00      0.47       191

    accuracy                           1.00    209715
   macro avg       0.65      1.00      0.73    209715
weighted avg       1.00      1.00      1.00    209715
```

Thanks !!!