In [1]:
```python
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
```

In [106…
```python
df = pd.read_csv("creditcard.csv")
df
```

Out[106]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Categ
0	768805383	Existing Customer	45	M	3	High School	Married	60K − 80K	E
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	E
2	713982108	Existing Customer	51	M	3	Graduate	Married	80K − 120K	E
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	E
4	709106358	Existing Customer	40	M	3	Uneducated	Married	60K − 80K	E
...	...	...	...	...	...	...	...	...	
10122	772366833	Existing Customer	50	M	2	Graduate	Single	40K − 60K	E
10123	710638233	Attrited Customer	41	M	2	Unknown	Divorced	40K − 60K	E
10124	716506083	Attrited Customer	44	F	1	High School	Married	Less than $40K	E
10125	717406983	Attrited Customer	30	M	2	Graduate	Unknown	40K − 60K	E
10126	714337233	Attrited Customer	43	F	2	Graduate	Married	Less than $40K	Si

10127 rows × 23 columns

In [3]:
```python
df.shape
```

Out[3]:
```
(10127, 23)
```

In [4]:
```python
df.head(5)
```

Out[4]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	I
0	768805383	Existing Customer	45	M	3	High School	Married	60K − 80K	Blue	
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	
2	713982108	Existing Customer	51	M	3	Graduate	Married	80K − 120K	Blue	
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	
4	709106358	Existing Customer	40	M	3	Uneducated	Married	60K − 80K	Blue	

5 rows × 23 columns

In [5]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 23 columns):
 #   Column
Non-Null Count  Dtype
---  ------
--------------  -----
 0   CLIENTNUM
10127 non-null  int64
 1   Attrition_Flag
10127 non-null  object
 2   Customer_Age
10127 non-null  int64
 3   Gender
10127 non-null  object
 4   Dependent_count
10127 non-null  int64
 5   Education_Level
10127 non-null  object
 6   Marital_Status
10127 non-null  object
 7   Income_Category
10127 non-null  object
 8   Card_Category
10127 non-null  object
 9   Months_on_book
10127 non-null  int64
 10  Total_Relationship_Count
10127 non-null  int64
 11  Months_Inactive_12_mon
10127 non-null  int64
 12  Contacts_Count_12_mon
10127 non-null  int64
 13  Credit_Limit
10127 non-null  float64
 14  Total_Revolving_Bal
10127 non-null  int64
 15  Avg_Open_To_Buy
10127 non-null  float64
 16  Total_Amt_Chng_Q4_Q1
10127 non-null  float64
 17  Total_Trans_Amt
10127 non-null  int64
 18  Total_Trans_Ct
10127 non-null  int64
 19  Total_Ct_Chng_Q4_Q1
10127 non-null  float64
 20  Avg_Utilization_Ratio
10127 non-null  float64
 21  Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_
Months_Inactive_12_mon_1  10127 non-null  float64
 22  Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_
Months_Inactive_12_mon_2  10127 non-null  float64
dtypes: float64(7), int64(10), object(6)
memory usage: 1.8+ MB
```

In [6]:
```
df.describe()
```

Out[6]:

	CLIENTNUM	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_
count	1.012700e+04	10127.000000	10127.000000	10127.000000	10127.000000	10127.000000	1012
mean	7.391776e+08	46.325960	2.346203	35.928409	3.812580	2.341167	
std	3.690378e+07	8.016814	1.298908	7.986416	1.554408	1.010622	
min	7.080821e+08	26.000000	0.000000	13.000000	1.000000	0.000000	
25%	7.130368e+08	41.000000	1.000000	31.000000	3.000000	2.000000	
50%	7.179264e+08	46.000000	2.000000	36.000000	4.000000	2.000000	
75%	7.731435e+08	52.000000	3.000000	40.000000	5.000000	3.000000	
max	8.283431e+08	73.000000	5.000000	56.000000	6.000000	6.000000	

TERMS:

TOTAL AMOUNT CHANGE Q4 TO Q1 : The change in total amount from Q4 to Q1 represents the difference in the total amount of something (such as revenue, sales, expenses, etc.) between the fourth quarter (Q4) and the first quarter (Q1) of a specific time period, typically in a fiscal or calendar year.

TOTAL CT CHANGE Q4 TO Q1 : represent the rate of change in transaction activity among customers.

Total transaction count : no of transactions counted

Total transaction amount : Toal amount in no of transactions

TOTAL RELATIONSHIP COUNT: refers to the total number of financial products or accounts held by a customer within the bank. This indicates customer loyalty and support to the bank.

CREDIT LIMIT : Credit given to each customer based on their income and qualifications.

TOTAL REVOLVING BALANCE : The balance that carries over from one month to the next.

AVERAGE OPEN TO BUY : for any open account on any business day , the excess of the credit limit and the amount of receivables.

AVERAGE UTILISATION RATIO : (total credit card balances / total credit card credit limits) * 100

MONTHS INACTIVE : No of months a customer account is inactive

CONTACTS COUNT : How many times the credit card user contacted by the credit card issuer?

CREDIT CARD ANOMALY DETECTION

USING Z SCORE

In [107…

```
#create a new dataframe with the selected variables
import pandas as pd

var = ['Total_Relationship_Count','Credit_Limit','Months_Inactive_12_mon','Contacts_Count_12_mon','Total_Revolv
df1 = df[var]
df1
```

Out[107]:

	Total_Relationship_Count	Credit_Limit	Months_Inactive_12_mon	Contacts_Count_12_mon	Total_Revolving_Bal	Avg_Open_To_Buy	T
0	5	12691.00	1	3	777	11914.00	
1	6	8256.00	1	2	864	7392.00	
2	4	3418.00	1	0	0	3418.00	
3	3	3313.00	4	1	2517	796.00	
4	5	4716.00	1	0	0	4716.00	
...	...	...	...	...	...	...	
10122	3	4003.00	2	3	1851	2152.00	
10123	4	4277.00	2	3	2186	2091.00	
10124	5	5409.00	3	4	0	5409.00	
10125	4	5281.00	3	3	0	5281.00	
10126	6	10388.00	2	4	1961	8427.00	

10127 rows × 9 columns

In [8]:

```
import pandas as pd
import numpy as np
from scipy import stats

# Initialize an empty dataframe to store z-scores for each variable
z_df = pd.DataFrame()

# Applying z-score for the selected variables
selected_var = ['Total_Relationship_Count', 'Credit_Limit', 'Months_Inactive_12_mon', 'Contacts_Count_12_mon',

for var in selected_var:
    z_scores = np.abs(stats.zscore(df1[var]))  # Z-score calculated for each selected variable using np.abs
    z_df[var + '_Z_score'] = z_scores          # Calculated Z-scores are placed in the DataFrame

z_df.head()
```

Out[8]:

	Total_Relationship_Count_Z_score	Credit_Limit_Z_score	Months_Inactive_12_mon_Z_score	Contacts_Count_12_mon_Z_score	Total_Revolvir
0	0.763943	0.446622	1.327136	0.492404	
1	1.407306	0.041367	1.327136	0.411616	
2	0.120579	0.573698	1.327136	2.219655	
3	0.522785	0.585251	1.641478	1.315636	
4	0.763943	0.430877	1.327136	2.219655	

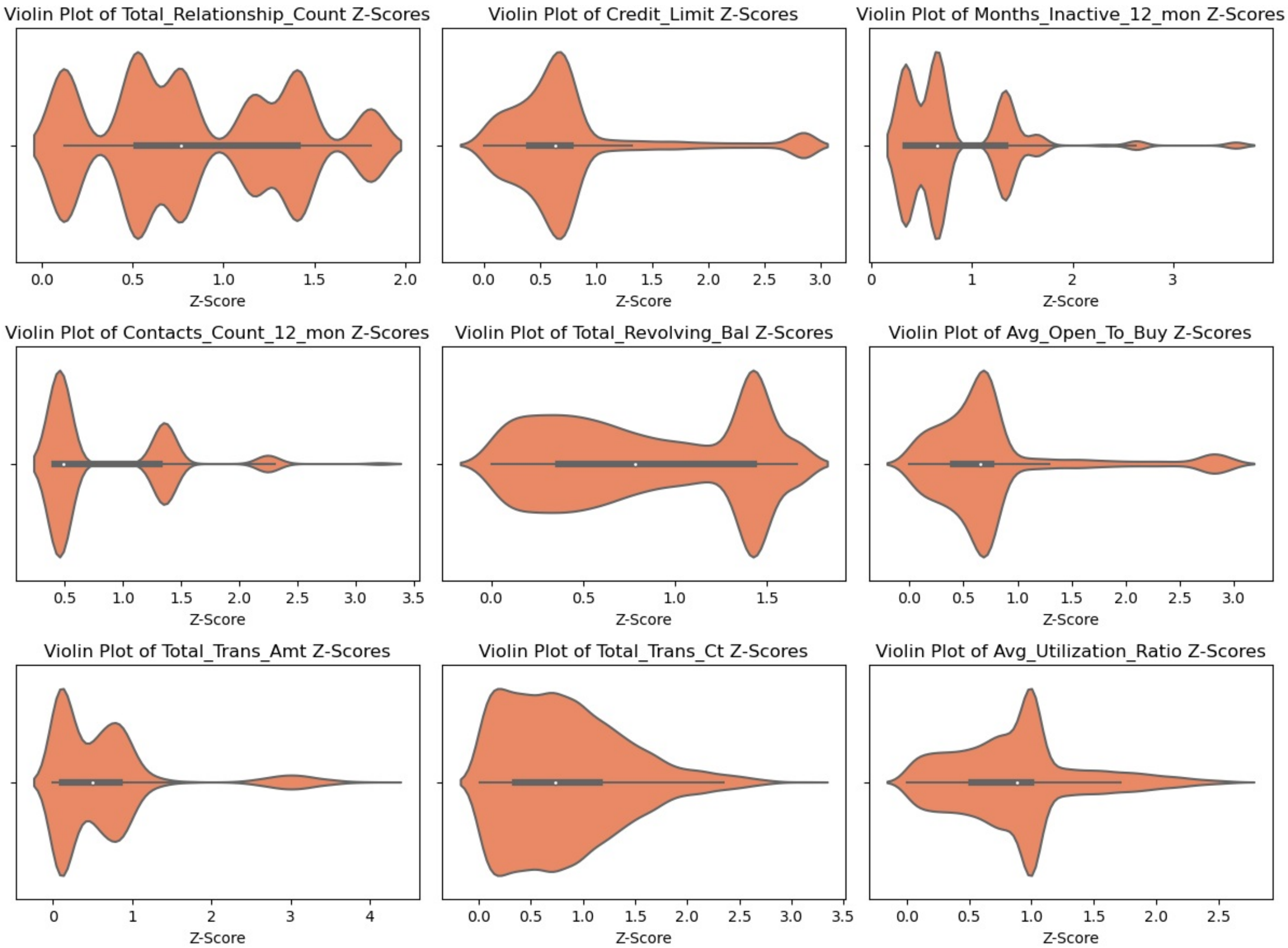
VISUALIZATION USING VIOLIN PLOT

VIOLIN PLOT:

Violin plots are a combination of box plot and histograms. It portrays the distribution, median, interquartile range of data. So we see that iqr and median are the statistical information provided by box plot whereas distribution is being provided by the histogram.

In [9]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a 3x3 grid for violin plots
fig, axes = plt.subplots(3, 3, figsize=(12, 9))

# Assuming you have 9 variables in 'selected_var'
selected_var = ['Total_Relationship_Count', 'Credit_Limit', 'Months_Inactive_12_mon',
                'Contacts_Count_12_mon', 'Total_Revolving_Bal', 'Avg_Open_To_Buy',
                'Total_Trans_Amt', 'Total_Trans_Ct', 'Avg_Utilization_Ratio']

# Loop through the selected variables and create violin plots
for i, var in enumerate(selected_var):
    row, col = i // 3, i % 3
    sns.violinplot(x=z_df[var + '_Z_score'], color='coral', scale='width', ax=axes[row, col])
    axes[row, col].set_title(f'Violin Plot of {var} Z-Scores')
    axes[row, col].set_xlabel('Z-Score')

# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```



In [10]:

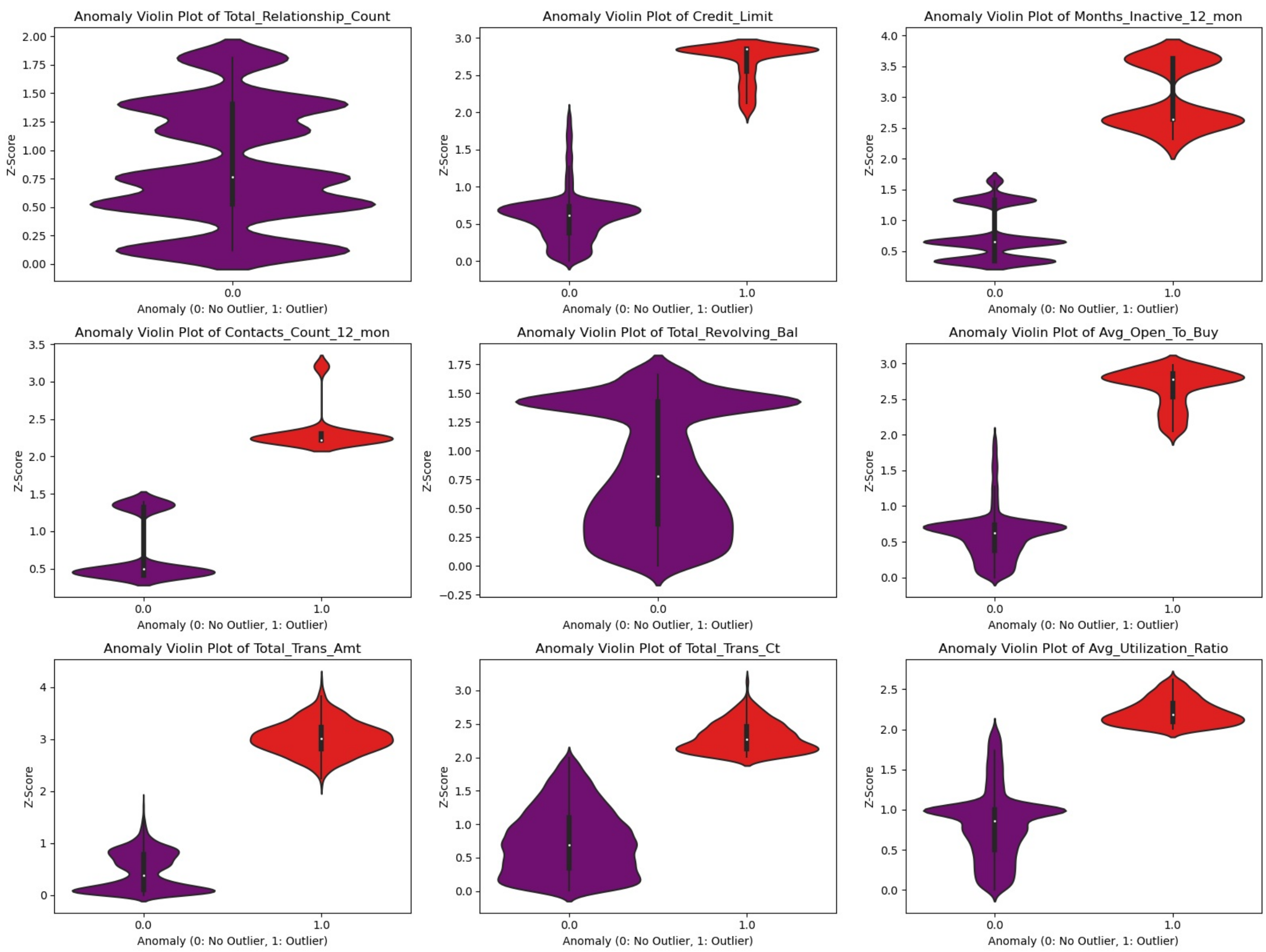
```
# Create an empty DataFrame to store the anomalies
anomaly_df = pd.DataFrame()

# Applying z-score for the selected variable
selected_var = ['Total_Relationship_Count','Credit_Limit','Months_Inactive_12_mon','Contacts_Count_12_mon','Tot

for var in selected_var:
    z_scores = np.abs(stats.zscore(df1[var]))
    z_df[var + '_Z_score'] = z_scores

    # Identify anomalies based on the threshold of 2
    anomalies      = np.where(z_scores > 2)
    anomaly_column = np.zeros(len(df1))  # Create a placeholder for anomalies
    anomaly_column[anomalies] = 1
    anomaly_df[var] = anomaly_column     #0.0 represent normal z-score and 1.0 represents anomaly

anomaly_df.head()
```

Out[10]:

	Total_Relationship_Count	Credit_Limit	Months_Inactive_12_mon	Contacts_Count_12_mon	Total_Revolving_Bal	Avg_Open_To_Buy	Total_T
0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	1.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	0.0	0.0	

In [11]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have created 'anomaly_df' and 'z_df' as you mentioned earlier.

# Create a 3x3 grid for violin plots
fig, axes = plt.subplots(3, 3, figsize=(16, 12))

# Assuming you have 'selected_var' as you mentioned earlier.
selected_var = ['Total_Relationship_Count', 'Credit_Limit', 'Months_Inactive_12_mon',
                'Contacts_Count_12_mon', 'Total_Revolving_Bal', 'Avg_Open_To_Buy',
                'Total_Trans_Amt', 'Total_Trans_Ct', 'Avg_Utilization_Ratio']

for i, var in enumerate(selected_var):
    row, col = i // 3, i % 3  # Calculate the row and column for each subplot

    sns.violinplot(x=anomaly_df[var], y=z_df[var + '_Z_score'], scale='width', ax=axes[row, col],
                   palette={0: 'purple', 1: 'red'})

    axes[row, col].set_title(f'Anomaly Violin Plot of {var}')
    axes[row, col].set_xlabel('Anomaly (0: No Outlier, 1: Outlier)')
    axes[row, col].set_ylabel('Z-Score')

# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```



RESULT : In this method we finds out that Total_Relationship_count and Total_Revolving_Bal has no anomaly compared to other 9 variables

In [12]:

```
# Calculate the mean and standard deviation for a feature
mean_1 = df['Total_Relationship_Count'].mean()
std_1 = df['Total_Relationship_Count'].std()
```

```
# Calculate the mean and standard deviation for a feature
mean_2 = df['Credit_Limit'].mean()
std_2 = df['Credit_Limit'].std()

# Calculate the mean and standard deviation for a feature
mean_3 = df['Months_Inactive_12_mon'].mean()
std_3 = df['Months_Inactive_12_mon'].std()

# Calculate the mean and standard deviation for a feature
mean_4 = df['Contacts_Count_12_mon'].mean()
std_4 = df['Contacts_Count_12_mon'].std()

# Calculate the mean and standard deviation for a feature
mean_5 = df['Total_Revolving_Bal'].mean()
std_5 = df['Total_Revolving_Bal'].std()

# Calculate the mean and standard deviation for a feature
mean_6 = df['Avg_Open_To_Buy'].mean()
std_6 = df['Avg_Open_To_Buy'].std()

# Calculate the mean and standard deviation for a feature
mean_7 = df['Total_Trans_Amt'].mean()
std_7 = df['Total_Trans_Amt'].std()

# Calculate the mean and standard deviation for a feature
mean_8 = df['Total_Trans_Ct'].mean()
std_8 = df['Total_Trans_Ct'].std()

# Calculate the mean and standard deviation for a feature
mean_9 = df['Avg_Utilization_Ratio'].mean()
std_9 = df['Avg_Utilization_Ratio'].std()
```

In [104…

```
# Define the z-score calculation function
def calculate_z_score(user_input, feature_mean, feature_std):
    z_score = (user_input - feature_mean) / feature_std
    return z_score

# Define the function to check for fraud
def check_for_fraud(user_inputs):
    # Define group-wise features and their means and standard deviations
    customer_profile_features = ["Total_Relationship_Count", "Credit_Limit"]
    customer_engagement_features = ["Months_Inactive_12_mon", "Contacts_Count_12_mon"]
    credit_card_usage_features = ["Total_Revolving_Bal", "Avg_Open_To_Buy"]
    transaction_history_features = ["Total_Trans_Amt", "Total_Trans_Ct", "Avg_Utilization_Ratio"]

    # Calculate z-scores for each group
    customer_profile_z_scores = [calculate_z_score(user_inputs[feature], mean, std) for feature, mean, std in z
    customer_engagement_z_scores = [calculate_z_score(user_inputs[feature], mean, std) for feature, mean, std i
    credit_card_usage_z_scores = [calculate_z_score(user_inputs[feature], mean, std) for feature, mean, std in
    transaction_history_z_scores = [calculate_z_score(user_inputs[feature], mean, std) for feature, mean, std i

    # Check if more than one group has exceeded the z-score threshold
    exceeded_groups = 0
    if all(z >= 2 for z in customer_profile_z_scores):
        exceeded_groups += 1
        fraud_group = "Customer Profile"
    if all(z >= 2 for z in customer_engagement_z_scores):
        exceeded_groups += 1
        fraud_group = "Customer Engagement"
    if all(z >= 2 for z in credit_card_usage_z_scores):
        exceeded_groups += 1
        fraud_group = "Credit Card Usage"
    if all(z >= 2 for z in transaction_history_z_scores):
        exceeded_groups += 1
        fraud_group = "Transaction History"

    # Determine the result based on the number of exceeded groups
    if exceeded_groups >= 2:
        return "Fraud in Multiple Groups"
    elif exceeded_groups == 1:
        return f"Fraud in {fraud_group}"
    else:
        return "Normal Transaction"

# Collect user inputs for each feature
user_inputs = {}
for feature in ["Total_Relationship_Count", "Credit_Limit", "Months_Inactive_12_mon", "Contacts_Count_12_mon",
                "Total_Revolving_Bal", "Avg_Open_To_Buy", "Total_Trans_Amt", "Total_Trans_Ct", "Avg_Utilization
    user_input = float(input(f"Enter value for {feature}: "))
    user_inputs[feature] = user_input

# Call the check_for_fraud function with user inputs
result = check_for_fraud(user_inputs)
print("Result:", result)
```

```
Enter value for Total_Relationship_Count: 8
Enter value for Credit_Limit: 8000
Enter value for Months_Inactive_12_mon: 12
Enter value for Contacts_Count_12_mon: 8
Enter value for Total_Revolving_Bal: 500
Enter value for Avg_Open_To_Buy: 3000
Enter value for Total_Trans_Amt: 2000
Enter value for Total_Trans_Ct: 15
Enter value for Avg_Utilization_Ratio: 0.625
Result: Fraud in Customer Engagement
```

In [14]:
```
## check with these values for sample
## Fraud in Customer Engagement:
Total_Relationship_Count: 8
Credit_Limit: 8000
Months_Inactive_12_mon: 12
Contacts_Count_12_mon: 8
Total_Revolving_Bal: 5000
Avg_Open_To_Buy: 3000
Total_Trans_Amt: 2000
Total_Trans_Ct: 15
Avg_Utilization_Ratio: 0.625
```

- The code is designed to check if a financial transaction is potentially **fraudulent** or not based on certain features of the transaction.
- It collects user inputs for various features of the transaction, such as the **total relationship count**, **credit limit**, **months inactive in the last 12 months**, etc.
- For each feature, it calculates a **Z-score**. The Z-score measures how far away a particular value is from the mean (average) value for that feature.
- The code groups these features into four categories: **customer profile**, **customer engagement**, **credit card usage**, and **transaction history**.
- It then checks if the **Z-scores** for any of these groups exceed a threshold of **2**. If all the Z-scores in a group are greater than or equal to 2, it suggests potential **fraud** in that category.
- If more than one group exceeds this threshold, it suggests **"Fraud in Multiple Groups."**
- If only one group exceeds the threshold, it specifies the category where potential **fraud** is detected, such as **"Fraud in Customer Profile"** or **"Fraud in Credit Card Usage."**
- If none of the Z-scores in any group exceed the threshold, it concludes that the transaction is **"Normal."**
- The final result is printed to indicate whether the transaction is normal or potentially fraudulent, and if fraudulent, which category it falls into.

Isolation Forest Algorithm

In [15]:
```
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_m
```

In [67]:
```
# Define a function to normalize using Min-Max scaling
def min_max_scaling(column):
    min_val = column.min()
    max_val = column.max()
    normalized_column = (column - min_val) / (max_val - min_val)
    return normalized_column
```

In [68]:
```
# Normalize the columns using the defined function
normalized_df = df1.apply(min_max_scaling)
```

In [69]:
```
normalized_df.head()
```

Out[69]:

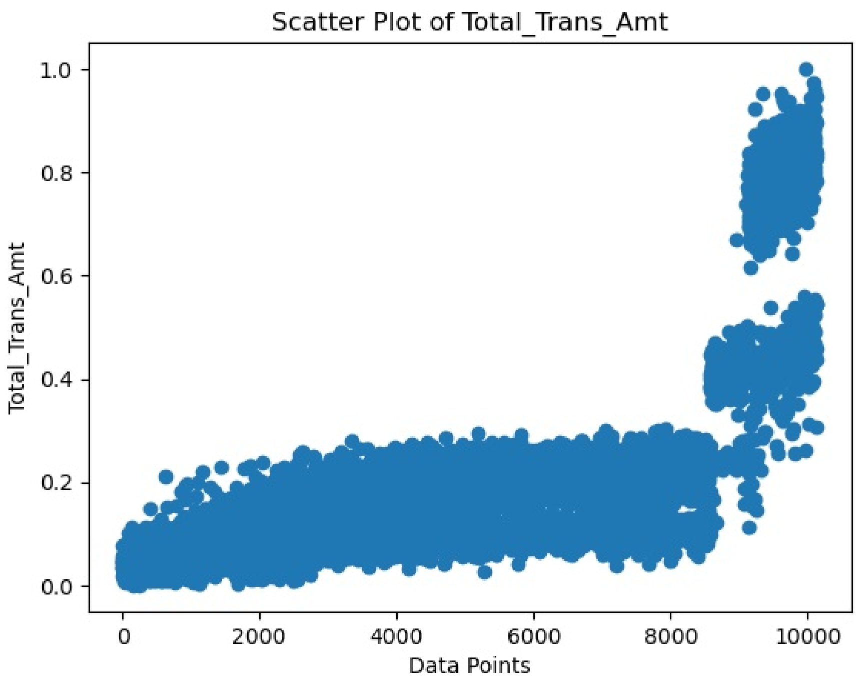
	Total_Relationship_Count	Credit_Limit	Months_Inactive_12_mon	Contacts_Count_12_mon	Total_Revolving_Bal	Avg_Open_To_Buy	Total_T
0	0.8	0.340190	0.166667	0.500000	0.308701	0.345116	
1	1.0	0.206112	0.166667	0.333333	0.343266	0.214093	
2	0.6	0.059850	0.166667	0.000000	0.000000	0.098948	
3	0.4	0.056676	0.666667	0.166667	1.000000	0.022977	
4	0.8	0.099091	0.166667	0.000000	0.000000	0.136557	

In [19]:
```
normalized_df.describe()
```

Out[19]:

	Total_Relationship_Count	Credit_Limit	Months_Inactive_12_mon	Contacts_Count_12_mon	Total_Revolving_Bal	Avg_Open_To_Buy	T
count	10127.000000	10127.000000	10127.000000	10127.000000	10127.000000	10127.000000	
mean	0.562516	0.217477	0.390195	0.409220	0.461984	0.216328	
std	0.310882	0.274771	0.168437	0.184371	0.323793	0.263399	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.400000	0.033760	0.333333	0.333333	0.142630	0.038290	
50%	0.600000	0.094042	0.333333	0.333333	0.506953	0.100571	
75%	0.800000	0.291109	0.500000	0.500000	0.708780	0.285574	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

In [20]:
```
# Assuming your normalized DataFrame is named 'normalized_df'
column_name = 'Total_Trans_Amt'  # Replace with the name of the column you want to analyze

plt.scatter(range(len(normalized_df)), normalized_df[column_name])
plt.xlabel('Data Points')
plt.ylabel(column_name)
plt.title(f'Scatter Plot of {column_name}')
plt.show()
```



In [21]:
```
# Create and train the Isolation Forest model
model = IsolationForest(contamination=0.05, random_state=42)
model.fit(normalized_df[[column_name]])
```

```
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
```

Out[21]:
```
IsolationForest
IsolationForest(contamination=0.05, random_state=42)
```

In [22]:
```
# Predict anomalies (-1) and inliers (1)
anomaly_predictions_tran = model.predict(normalized_df[[column_name]])
```

In [23]:
```
anomaly_predictions_tran
```

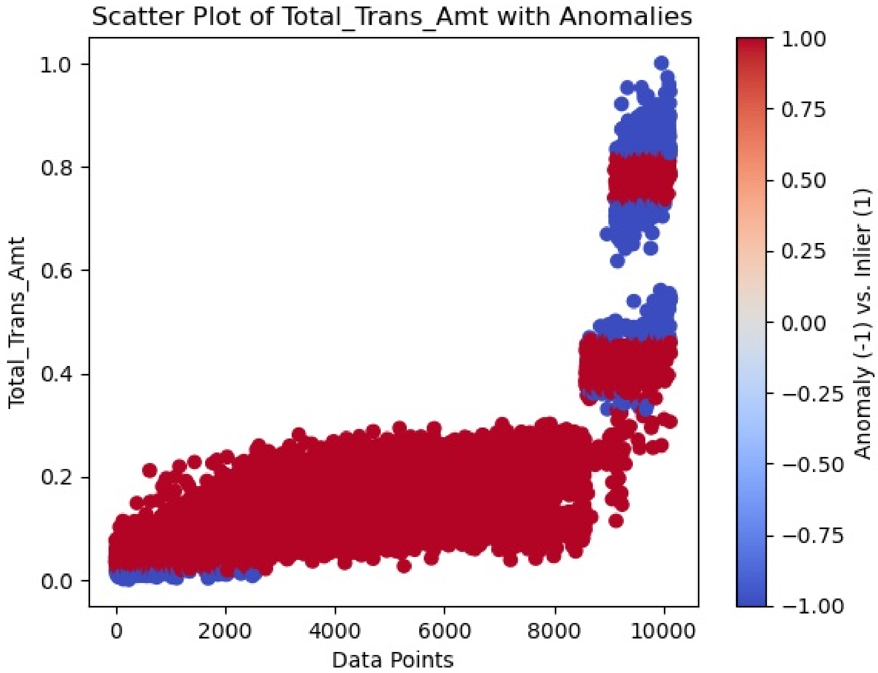
Out[23]:
```
array([ 1,  1,  1, ..., -1,  1, -1])
```

In [ ]:

In [24]:
```
# Add the predictions as a new column in your DataFrame
normalized_df['tran_anomaly'] = anomaly_predictions_tran
```

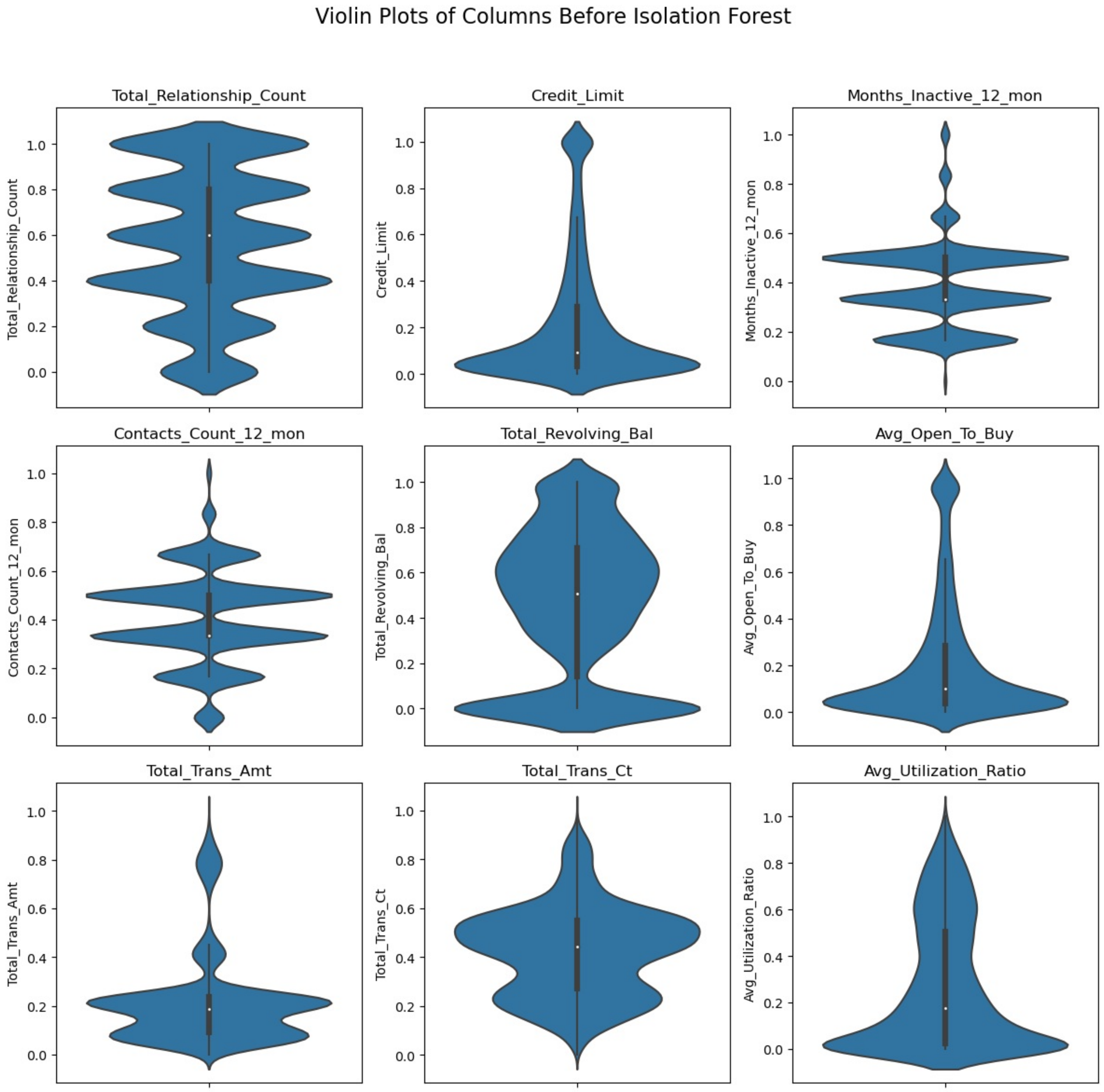
In [25]:
```
# Create a scatter plot with different colors for inliers (1) and outliers (-1)
plt.scatter(range(len(normalized_df)), normalized_df[column_name], c=normalized_df['tran_anomaly'], cmap='coolw
plt.xlabel('Data Points')
plt.ylabel(column_name)
plt.title(f'Scatter Plot of {column_name} with Anomalies')
plt.colorbar(label='Anomaly (-1) vs. Inlier (1)')
```

```
plt.show()
```



In [26]:

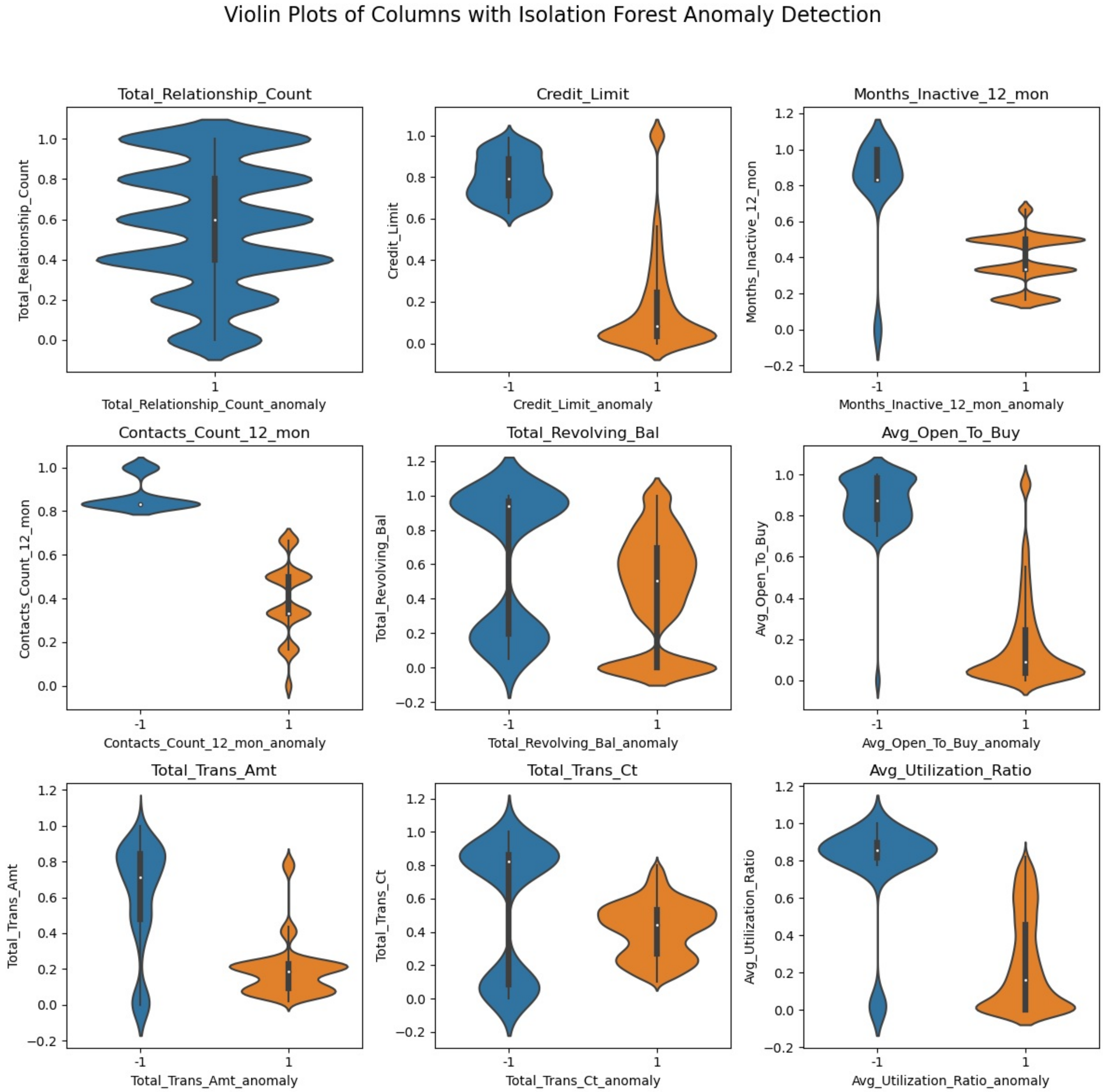
```
# Assuming your normalized DataFrame is named 'normalized_df'
columns_to_visualize = ['Total_Relationship_Count', 'Credit_Limit', 'Months_Inactive_12_mon',
                        'Contacts_Count_12_mon', 'Total_Revolving_Bal', 'Avg_Open_To_Buy',
                        'Total_Trans_Amt', 'Total_Trans_Ct', 'Avg_Utilization_Ratio']

# Create a 3x3 subplot for visualizing each column using violin plots
fig, axes = plt.subplots(3, 3, figsize=(12, 12))
fig.suptitle('Violin Plots of Columns Before Isolation Forest', fontsize=16)

# Loop through the columns and create violin plots
for i, column in enumerate(columns_to_visualize):
    row = i // 3
    col = i % 3

    sns.violinplot(data=normalized_df, y=column, ax=axes[row, col])
    axes[row, col].set_ylabel(column)
    axes[row, col].set_title(column)

# Adjust subplot layout
plt.tight_layout(rect=[0, 0, 1, 0.95])

# Show the plot
plt.show()
```



In [27]:
```
columns_to_visualize = ['Total_Relationship_Count', 'Credit_Limit', 'Months_Inactive_12_mon',
                        'Contacts_Count_12_mon', 'Total_Revolving_Bal', 'Avg_Open_To_Buy',
                        'Total_Trans_Amt', 'Total_Trans_Ct', 'Avg_Utilization_Ratio']

# Create a 3x3 subplot for visualizing each column using violin plots
fig, axes = plt.subplots(3, 3, figsize=(12, 12))
fig.suptitle('Violin Plots of Columns with Isolation Forest Anomaly Detection', fontsize=16)

# Initialize the Isolation Forest model
model = IsolationForest(contamination=0.05, random_state=42)

# Loop through the columns and create violin plots with anomaly detection
for i, column in enumerate(columns_to_visualize):
    row = i // 3
    col = i % 3

    # Fit the Isolation Forest model to the current column
    model.fit(normalized_df[[column]])

    # Predict anomalies (-1) and inliers (1)
    anomaly_predictions = model.predict(normalized_df[[column]])

    # Add the predictions as a new column in the DataFrame
    normalized_df[column + '_anomaly'] = anomaly_predictions

    # Create a violin plot with anomaly colors
    sns.violinplot(data=normalized_df, y=column, x=column + '_anomaly', ax=axes[row, col])
    axes[row, col].set_ylabel(column)
    axes[row, col].set_title(column)

# Adjust subplot layout
plt.tight_layout(rect=[0, 0, 1, 0.95])
```

```python
# Show the plot
plt.show()
```

```
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
```



In [ ]:

Workflow:

Workflow:

1. **Grouping:**
   - The code groups user inputs into categories, namely Customer Profile, Customer Engagement, Credit Card Usage, and Transaction History.
2. **DataFrame Creation:**
   - User inputs are transformed into a DataFrame.
3. **Model Application:**
   - The Isolation Forest model is applied to each category.
4. **Prediction:**
   - Anomalies (-1) and inliers (1) are predicted for each category based on the Isolation Forest model.
5. **Result Storage:**
   - The code counts the number of anomalies (-1) in each category and stores the results.
6. **Display Result:**
   - After the user clicks the 'Submit' button, the code determines the most fraudulent category by identifying which category has the most anomalies.
7. **Output:**
   - The result is displayed, indicating the most fraudulent category and the number of anomalies in that category.

In [29]:

```
normalized_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 19 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Total_Relationship_Count          10127 non-null  float64
 1   Credit_Limit                      10127 non-null  float64
 2   Months_Inactive_12_mon            10127 non-null  float64
 3   Contacts_Count_12_mon             10127 non-null  float64
 4   Total_Revolving_Bal               10127 non-null  float64
 5   Avg_Open_To_Buy                   10127 non-null  float64
 6   Total_Trans_Amt                   10127 non-null  float64
 7   Total_Trans_Ct                    10127 non-null  float64
 8   Avg_Utilization_Ratio             10127 non-null  float64
 9   tran_anomaly                      10127 non-null  int32
 10  Total_Relationship_Count_anomaly  10127 non-null  int32
 11  Credit_Limit_anomaly              10127 non-null  int32
 12  Months_Inactive_12_mon_anomaly    10127 non-null  int32
 13  Contacts_Count_12_mon_anomaly     10127 non-null  int32
 14  Total_Revolving_Bal_anomaly       10127 non-null  int32
 15  Avg_Open_To_Buy_anomaly           10127 non-null  int32
 16  Total_Trans_Amt_anomaly           10127 non-null  int32
 17  Total_Trans_Ct_anomaly            10127 non-null  int32
 18  Avg_Utilization_Ratio_anomaly     10127 non-null  int32
dtypes: float64(9), int32(10)
memory usage: 1.1 MB
```

In [103…

```
# Sample Isolation Forest model (replace with your trained model)
model = IsolationForest(contamination=0.05, random_state=42)
model.fit(normalized_df)

def detect_fraud(input_values):
    # Group the input values into categories
    customer_profile_features = ["Total_Relationship_Count", "Credit_Limit"]
    customer_engagement_features = ["Months_Inactive_12_mon", "Contacts_Count_12_mon"]
    credit_card_usage_features = ["Total_Revolving_Bal", "Avg_Open_To_Buy"]
    transaction_history_features = ["Total_Trans_Amt", "Total_Trans_Ct", "Avg_Utilization_Ratio"]

    # Create a DataFrame from user inputs
    user_data = pd.DataFrame({
        'Total_Relationship_Count': [input_values[0]],
        'Credit_Limit': [input_values[1]],
        'Months_Inactive_12_mon': [input_values[2]],
        'Contacts_Count_12_mon': [input_values[3]],
        'Total_Revolving_Bal': [input_values[4]],
        'Avg_Open_To_Buy': [input_values[5]],
        'Total_Trans_Amt': [input_values[6]],
        'Total_Trans_Ct': [input_values[7]],
        'Avg_Utilization_Ratio': [input_values[8]]
    })

    # Predict anomalies (-1) and inliers (1) for each category
    results = {}
    for category, features in zip(["Customer Profile", "Customer Engagement", "Credit Card Usage", "Transaction
                                  [customer_profile_features, customer_engagement_features, credit_card_usage_f
        # Fit the Isolation Forest model to the category
```

```
        model.fit(normalized_df[features])

        # Predict anomalies for the user data in the category
        category_predictions = model.predict(user_data[features])

        # Count the number of anomalies (-1)
        num_anomalies = sum(category_predictions == -1)

        # Store the result
        results[category] = num_anomalies

    return results

# Collect user inputs
user_inputs = {}
user_inputs[0] = float(input("Enter value for Total_Relationship_Count: "))
user_inputs[1] = float(input("Enter value for Credit_Limit: "))
user_inputs[2] = float(input("Enter value for Months_Inactive_12_mon: "))
user_inputs[3] = float(input("Enter value for Contacts_Count_12_mon: "))
user_inputs[4] = float(input("Enter value for Total_Revolving_Bal: "))
user_inputs[5] = float(input("Enter value for Avg_Open_To_Buy: "))
user_inputs[6] = float(input("Enter value for Total_Trans_Amt: "))
user_inputs[7] = float(input("Enter value for Total_Trans_Ct: "))
user_inputs[8] = float(input("Enter value for Avg_Utilization_Ratio: "))
# Detect frad for each category using the user input
fraud_results = detect_fraud(list(user_inputs.values()))


# Detect fraud for each category using the user inputs
fraud_results = detect_fraud(user_inputs)

# Determine which category has more anomalies
most_fraudulent_category = max(fraud_results, key=fraud_results.get)
print(f"The most fraudulent category is {most_fraudulent_category} with {fraud_results[most_fraudulent_category
```

```
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
Enter value for Total_Relationship_Count: 1
Enter value for Credit_Limit: 5
Enter value for Months_Inactive_12_mon: 4
Enter value for Contacts_Count_12_mon: 7
Enter value for Total_Revolving_Bal: 5
Enter value for Avg_Open_To_Buy: 8
Enter value for Total_Trans_Amt: 9
Enter value for Total_Trans_Ct: 8
Enter value for Avg_Utilization_Ratio: 5
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
D:\anaconda_2023\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but I
solationForest was fitted with feature names
  warnings.warn(
The most fraudulent category is Customer Profile with 1 anomalies.
```

This function takes user inputs for each attribute, groups them into categories, applies the Isolation Forest model to each category, and counts the number of anomalies (-1). Finally, it determines and prints which category has the most anomalies, which can be considered the most fraudulent category based on the user inputs.

DB Scan Clustering Algorthim

In [31]:
```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
```

In [32]:
```
db = normalized_df[['Total_Relationship_Count', 'Credit_Limit', 'Months_Inactive_12_mon', 'Contacts_Count_12_mo
```

In [33]:
```
db
```

In [33]: db

Out[33]:

	Total_Relationship_Count	Credit_Limit	Months_Inactive_12_mon	Contacts_Count_12_mon	Total_Revolving_Bal	Avg_Open_To_Buy	To
0	0.8	0.340190	0.166667	0.500000	0.308701	0.345116	
1	1.0	0.206112	0.166667	0.333333	0.343266	0.214093	
2	0.6	0.059850	0.166667	0.000000	0.000000	0.098948	
3	0.4	0.056676	0.666667	0.166667	1.000000	0.022977	
4	0.8	0.099091	0.166667	0.000000	0.000000	0.136557	
...	...	...	...	...	...	...	
10122	0.4	0.077536	0.333333	0.500000	0.735399	0.062266	
10123	0.6	0.085819	0.333333	0.500000	0.868494	0.060499	
10124	0.8	0.120042	0.500000	0.666667	0.000000	0.156637	
10125	0.6	0.116172	0.500000	0.500000	0.000000	0.152928	
10126	1.0	0.270566	0.333333	0.666667	0.779102	0.244082	

10127 rows × 9 columns

In [ ]:

In [34]:

```
# Columns to analyze
cols = ['Total_Relationship_Count', 'Credit_Limit', 'Months_Inactive_12_mon', 'Contacts_Count_12_mon', 'Total_R

# Extract columns
X = df1[cols]

# Initialize output DataFrame
dbscan_clusters = pd.DataFrame(index=X.index)

# DBSCAN per column
for col in cols:

    # Normalize
    normalized = (X[col] - X[col].mean()) / X[col].std()

    # Get DBSCAN labels
    db = DBSCAN(eps=0.1, min_samples=10).fit(normalized.values.reshape(-1,1))

    # Record labels in dataframe
    label_col = col + '_label'
    dbscan_clusters[label_col] = db.labels_

print(dbscan_clusters)
```

```
       Total_Relationship_Count_label  Credit_Limit_label  \
0                                   0                   0
1                                   1                   0
2                                   2                   0
3                                   3                   0
4                                   0                   0
...                               ...                 ...
10122                               3                   0
10123                               2                   0
10124                               0                   0
10125                               2                   0
10126                               1                   0

       Months_Inactive_12_mon_label  Contacts_Count_12_mon_label  \
0                                 0                            0
1                                 0                            1
2                                 0                            2
3                                 1                            3
4                                 0                            2
...                             ...                          ...
10122                             2                            0
10123                             2                            0
10124                             3                            4
10125                             3                            0
10126                             2                            4

       Total_Revolving_Bal_label  Avg_Open_To_Buy_label  \
0                              0                      0
1                              0                      0
2                              1                      0
3                              0                      0
4                              1                      0
...                          ...                    ...
10122                          0                      0
10123                          0                      0
10124                          1                      0
10125                          1                      0
10126                          0                      0

       Total_Trans_Amt_label  Total_Trans_Ct_label  \
0                          0                     0
1                          0                     0
2                          0                     0
3                          0                     0
4                          0                     0
...                      ...                   ...
10122                      1                     0
10123                      0                     0
10124                      0                     0
10125                      0                     0
10126                      0                     0

       Avg_Utilization_Ratio_label
0                                0
1                                0
2                                0
3                                0
4                                0
...                            ...
10122                            0
10123                            0
10124                            0
10125                            0
10126                            0

[10127 rows x 9 columns]
```

In [35]:

```
outlier_rows = []

for col in dbscan_clusters.columns:
    outliers = dbscan_clusters[dbscan_clusters[col]==-1]
    outlier_rows.append(outliers)

outliers_df = pd.concat(outlier_rows, ignore_index=True).drop_duplicates()

print(outliers_df)
```

```
   Total_Relationship_Count_label  Credit_Limit_label  \
0                               4                   0   
1                               2                   0   
2                               3                   0   
3                               3                   0   
4                               4                   0   
5                               5                   0   

   Months_Inactive_12_mon_label  Contacts_Count_12_mon_label  \
0                             0                            1   
1                             6                            1   
2                             3                            0   
3                             2                            0   
4                             1                            0   
5                             2                            3   

   Total_Revolving_Bal_label  Avg_Open_To_Buy_label  Total_Trans_Amt_label  \
0                          0                      0                     -1   
1                          0                      0                     -1   
2                          0                      0                     -1   
3                          0                      0                     -1   
4                          0                      0                      1   
5                          0                      0                      1   

   Total_Trans_Ct_label  Avg_Utilization_Ratio_label  
0                     0                            0  
1                     0                            0  
2                     0                            0  
3                     0                            0  
4                    -1                            0  
5                    -1                            0  
```

In [36]:
```
outliers_df.head()
```

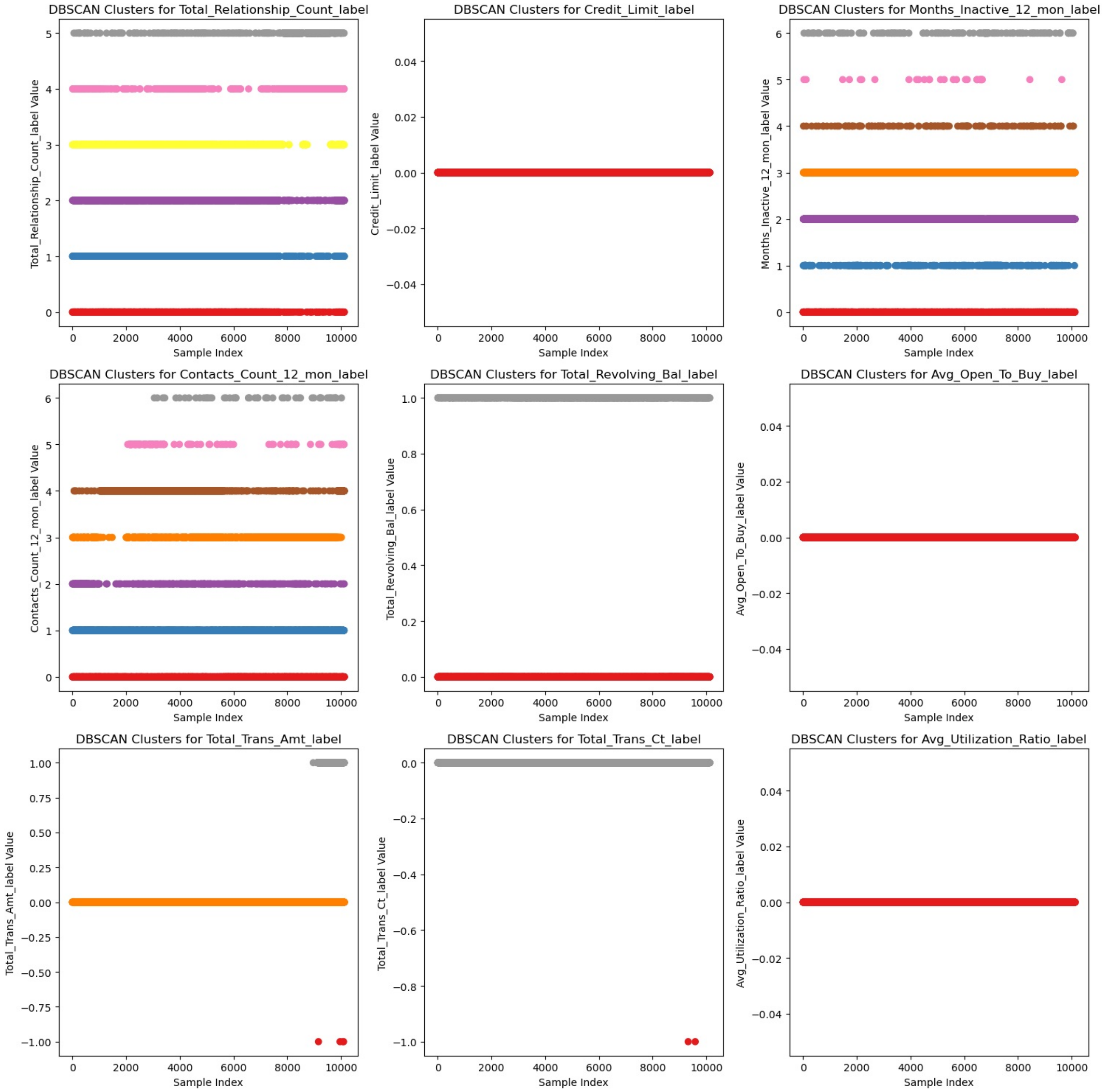
Out[36]:

	Total_Relationship_Count_label	Credit_Limit_label	Months_Inactive_12_mon_label	Contacts_Count_12_mon_label	Total_Revolving_Bal_label
0	4	0	0	1	0
1	2	0	6	1	0
2	3	0	3	0	0
3	3	0	2	0	0
4	4	0	1	0	0

In [37]:
```
for column in outliers_df.columns:
    print(column, outliers_df[column].unique())
```

```
Total_Relationship_Count_label [4 2 3 5]
Credit_Limit_label [0]
Months_Inactive_12_mon_label [0 6 3 2 1]
Contacts_Count_12_mon_label [1 0 3]
Total_Revolving_Bal_label [0]
Avg_Open_To_Buy_label [0]
Total_Trans_Amt_label [-1  1]
Total_Trans_Ct_label [ 0 -1]
Avg_Utilization_Ratio_label [0]
```

In [38]:
```
# Sample data
X = dbscan_clusters[['Total_Relationship_Count_label', 'Credit_Limit_label', 'Months_Inactive_12_mon_label', 'C

# Plot each column
fig, axs = plt.subplots(3, 3, figsize=(15, 15))

for i in range(3):
    for j in range(3):
        col = X.columns[i*3 + j]
        # Scatter plot data, colored by cluster label
        axs[i, j].scatter(X.index, X[col], c=X[col], cmap='Set1')

        # Label plot
        axs[i, j].set_title(f'DBSCAN Clusters for {col}')
        axs[i, j].set_xlabel('Sample Index')
        axs[i, j].set_ylabel(f'{col} Value')

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



In [39]:
```
cols = ['Total_Relationship_Count', 'Credit_Limit', 'Months_Inactive_12_mon', 'Contacts_Count_12_mon', 'Total_R
```

In [120…
```
df1
```

Out[120]:

	Total_Relationship_Count	Credit_Limit	Months_Inactive_12_mon	Contacts_Count_12_mon	Total_Revolving_Bal	Avg_Open_To_Buy	T
0	5.00	12691.00	1.00	3.00	777.00	11914.00	
1	6.00	8256.00	1.00	2.00	864.00	7392.00	
2	4.00	3418.00	1.00	0.00	0.00	3418.00	
3	3.00	3313.00	4.00	1.00	2517.00	796.00	
4	5.00	4716.00	1.00	0.00	0.00	4716.00	
...	...	...	...	...	...	...	
10126	6.00	10388.00	2.00	4.00	1961.00	8427.00	
10127	1.00	2.00	3.00	4.00	5.00	6.00	
10128	1.00	2.00	4.00	57.00	8.00	4.00	
10129	1.00	2.00	5.00	8.00	4.00	7.00	
10130	1.00	2.00	4.00	5.00	7.00	8.00	

10131 rows × 9 columns

In [40]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import DBSCAN

def add_and_check_outlier(df1, cols):
```

```
    # Get user input
    new_data = [float(input(f"Enter value for {col}: ")) for col in cols]
    new_df = pd.DataFrame([new_data], columns=cols)

    # Normalize user input
    scaler = MinMaxScaler()
    scaler.fit(df[cols])
    new_df[cols] = scaler.transform(new_df[cols])

   # Add new row
    df1 = df1.append(new_df, ignore_index=True)

    # DBSCAN on each column
    outlier_cols = []
    for col in cols:
        db = DBSCAN(eps=0.5, min_samples=5).fit(df1[[col]])
        if db.labels_[-1] == -1:
            outlier_cols.append(col)

    return outlier_cols

outlier_cols = add_and_check_outlier(df1, cols)
print(f"Outlier Columns: {outlier_cols}")
```

```
Enter value for Total_Relationship_Count: 1
Enter value for Credit_Limit: 4
Enter value for Months_Inactive_12_mon: 5
Enter value for Contacts_Count_12_mon: 2
Enter value for Total_Revolving_Bal: 3
Enter value for Avg_Open_To_Buy: 6
Enter value for Total_Trans_Amt: 8
Enter value for Total_Trans_Ct: 9
Enter value for Avg_Utilization_Ratio: 7
D:\mlproject\ipykernel_5636\1454078023.py:16: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  df1 = df1.append(new_df, ignore_index=True)
Outlier Columns: ['Total_Relationship_Count', 'Credit_Limit', 'Avg_Open_To_Buy', 'Total_Trans_Amt', 'Total_Trans_Ct', 'Avg_Utilization_Ratio']
```

How it Works:

1. Users input values for the specified features related to the transaction.
2. After entering values, clicking the 'Run DBSCAN' button adds the user input as a new row to the dataset.

DBSCAN Algorithm:

- The DBSCAN algorithm is then applied to each column independently.
- It identifies outliers in each column based on density and minimum samples parameters.

Result Display:

- If the last row (user input) is considered an outlier in any column, the column name is added to the list of outlier columns.
- The code then displays the result, indicating whether the last row (user input) is considered an outlier in any columns.
- If there are outlier columns, it specifies which columns they are; otherwise, it indicates that the last row (user inputs) is not considered an outlier.

Local Outlier Factor(LOF) Algorithm

In [111…

```
df1
```

Out[111]:

	Total_Relationship_Count	Credit_Limit	Months_Inactive_12_mon	Contacts_Count_12_mon	Total_Revolving_Bal	Avg_Open_To_Buy	T
0	5	12691.00	1	3	777	11914.00	
1	6	8256.00	1	2	864	7392.00	
2	4	3418.00	1	0	0	3418.00	
3	3	3313.00	4	1	2517	796.00	
4	5	4716.00	1	0	0	4716.00	
...	...	...	...	...	...	...	
10122	3	4003.00	2	3	1851	2152.00	
10123	4	4277.00	2	3	2186	2091.00	
10124	5	5409.00	3	4	0	5409.00	
10125	4	5281.00	3	3	0	5281.00	
10126	6	10388.00	2	4	1961	8427.00	

10127 rows × 9 columns

In [119…

```
from sklearn.neighbors import LocalOutlierFactor


# Columns to analyze
cols = ['Total_Relationship_Count', 'Credit_Limit', 'Months_Inactive_12_mon', 'Contacts_Count_12_mon', 'Total_R

# Load or create your dataset
# Example: normalized_df = pd.read_csv('your_dataset.csv')
# Assuming normalized_df is your dataset

# User input for a new row
new_row = {}
for col in cols:
    new_row[col] = float(input(f"Enter the value for {col}: "))

# Add the user input as a new row to the dataset
df1 = df1.append(new_row, ignore_index=True)

# Fit LOF model
lof = LocalOutlierFactor()

# Calculate LOF scores for the entire dataset (including the new row)
scores2 = lof.fit_predict(df1[cols])

# Check if the LOF score for the last row (newly appended row) is <= -3
if scores2[-1] <= -3:
    print("The last row (newly appended row) is considered as an outlier.")
else:
    print("The last row (newly appended row) is not considered as an outlier.")
```

```
Enter the value for Total_Relationship_Count: 1
Enter the value for Credit_Limit: 2
Enter the value for Months_Inactive_12_mon: 4
Enter the value for Contacts_Count_12_mon: 5
Enter the value for Total_Revolving_Bal: 7
Enter the value for Avg_Open_To_Buy: 8
Enter the value for Total_Trans_Amt: 9
Enter the value for Total_Trans_Ct: 4
Enter the value for Avg_Utilization_Ratio: 4
D:\mlproject\ipykernel_5636\142195752.py:17: FutureWarning: The frame.append method is deprecated and will be r
emoved from pandas in a future version. Use pandas.concat instead.
  df1 = df1.append(new_row, ignore_index=True)
The last row (newly appended row) is not considered as an outlier.
```

User Input Process:

1. Users input values for the specified features related to the transaction.
2. Click the 'Run LOF' button.

Algorithm Execution:

- The code adds the user input as a new row to the dataset.
- The LOF algorithm is applied to the entire dataset, including the newly appended row.
- It calculates LOF scores, measuring the local density deviation of a data point with respect to its neighbors.

Result Interpretation:

- The code checks if the LOF score for the last row (newly appended row) is less than or equal to -3.
- A score below this threshold indicates that the last row is considered an outlier.

Final Result Display:

- The code displays the result, indicating whether the last row is considered an outlier based on the LOF score.
- If it is, the code specifies that the last row is considered an outlier; otherwise, it indicates that the last row is not considered an outlier.

In [113…

```
lof_scores
```

Out[113]:

	Total_Relationship_Count_lof_score	Credit_Limit_lof_score	Months_Inactive_12_mon_lof_score	Contacts_Count_12_mon_lof_score	Tot
0	-1.00	-1.05	-1.00	-1.00	
1	-1.00	-1.00	-1.00	-1.00	
2	-1.00	-0.94	-1.00	-1.00	
3	-1.00	-1.00	-1.00	-1.00	
4	-1.00	-1.02	-1.00	-1.00	
...	...	...	...	...	
10123	-1.00	-1.09	-1.00	-1.00	
10124	-1.00	-1.13	-1.00	-1.00	
10125	-1.00	-1.08	-1.00	-1.00	
10126	-1.00	-1.12	-1.00	-1.00	
10127	-1.00	-14363000000001.00	-1.00	-1.00	

10128 rows × 9 columns

In [97]:

```
# Get ranges
ranges = lof_scores.agg(['min','max']).T

# Set display options
pd.set_option('display.float_format', '{:.2f}'.format)

# Print ranges
print(ranges)
```

```
                                       min   max
Total_Relationship_Count_lof_score       -1.00 -1.00
Credit_Limit_lof_score             -35415100.54 -0.93
Months_Inactive_12_mon_lof_score         -1.00 -1.00
Contacts_Count_12_mon_lof_score          -1.00 -1.00
Total_Revolving_Bal_lof_score      -10607867.50 -0.85
Avg_Open_To_Buy_lof_score          -36467419.02 -0.93
Total_Trans_Amt_lof_score                -3.48 -0.91
Total_Trans_Ct_lof_score           -38759690.92 -0.95
Avg_Utilization_Ratio_lof_score    -33783784.72 -0.84
```

In [98]:

```
# Filter rows with scores <= -3
filter_rows = np.any(lof_scores <= -3, axis=1)
outliers = lof_scores[filter_rows]
```

In [99]:

```
outliers
```

Out[99]:

	Total_Relationship_Count_lof_score	Credit_Limit_lof_score	Months_Inactive_12_mon_lof_score	Contacts_Count_12_mon_lof_score	Total
8	-1.00	-1.06	-1.00	-1.00	
16	-1.00	-1.02	-1.00	-1.00	
19	-1.00	-1.06	-1.00	-1.00	
26	-1.00	-1.03	-1.00	-1.00	
40	-1.00	-0.97	-1.00	-1.00	
...	...	...	...	...	
10088	-1.00	-1.00	-1.00	-1.00	
10093	-1.00	-1.17	-1.00	-1.00	
10102	-1.00	-0.98	-1.00	-1.00	
10106	-1.00	-1.09	-1.00	-1.00	
10111	-1.00	-0.96	-1.00	-1.00	

438 rows × 9 columns

In [ ]:

In [ ]:

In [ ]:

In [108…

```
# Assuming your DataFrame is called df
df1.to_csv('Anomaly.csv', index=False)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Processing math: 100%