In [2]:
```
# import some important libraries

import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
import pandas as pd

# Correcting the file path with raw string
df = pd.read_csv(r'F:\DS assignment\DS-Assignment Dataset and instructions\ds\b
```

In [5]:
```
df.head(10)
```

Out[5]:

	type	mode	amount	currentBalance	transactionTimestamp	valueDate	txnId	
0	DEBIT	CARD	100.0	2180.8	2023-06-27T09:40:19+05:30	2023-06-27	S39488701	
1	DEBIT	CARD	170.0	2010.8	2023-06-28T09:51:57+05:30	2023-06-28	S76862822	
2	DEBIT	CARD	500.0	1510.8	2023-07-26T10:04:00+05:30	2023-07-26	S31451661	
3	CREDIT	OTHERS	15.0	1525.8	2023-08-06T11:10:38+05:30	2023-07-31	S66463256	
4	DEBIT	ATM	1000.0	525.8	2023-08-07T17:13:13+05:30	2023-08-07	S18475743	
5	DEBIT	UPI	1.0	524.8	2023-08-22T08:05:06+05:30	2023-08-22	S82724622	UPI
6	CREDIT	UPI	3000.0	3524.8	2023-08-22T11:49:13+05:30	2023-08-22	S90667553	UPI
7	CREDIT	UPI	300.0	3824.8	2023-08-22T12:20:04+05:30	2023-08-22	S92051775	UPI/
8	DEBIT	UPI	1200.0	2624.8	2023-08-23T08:17:48+05:30	2023-08-23	S20566812	UPI
9	CREDIT	UPI	400.0	3024.8	2023-08-23T10:51:21+05:30	2023-08-23	S25232032	U

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

Out[6]:
```
(985, 9)
```

In [7]:
```
df.describe()
```

Out[7]:

	amount	currentBalance	reference
count	985.000000	985.000000	1.590000e+02
mean	855.492802	5901.308721	9.220074e+14
std	3007.515100	8670.950436	1.586054e+11
min	1.000000	0.800000	9.200201e+14
25%	40.000000	1174.800000	9.220200e+14
50%	160.000000	2723.110000	9.220200e+14
75%	500.000000	5834.110000	9.220200e+14
max	45000.000000	58450.800000	9.220200e+14

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 985 entries, 0 to 984
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   type                  985 non-null    object 
 1   mode                  985 non-null    object 
 2   amount                985 non-null    float64
 3   currentBalance        985 non-null    float64
 4   transactionTimestamp  985 non-null    object 
 5   valueDate             985 non-null    object 
 6   txnId                 985 non-null    object 
 7   narration             985 non-null    object 
 8   reference             159 non-null    float64
dtypes: float64(3), object(6)
memory usage: 69.4+ KB
```

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

Out[9]:
```
type                      0
mode                      0
amount                    0
currentBalance            0
transactionTimestamp      0
valueDate                 0
txnId                     0
narration                 0
reference               826
dtype: int64
```

In [10]:
```
df.columns
```

Out[10]:
```
Index(['type', 'mode', 'amount', 'currentBalance', 'transactionTimestamp',
       'valueDate', 'txnId', 'narration', 'reference'],
      dtype='object')
```

1. Transaction Analysis:

Q. What is the total number of transactions made over the year?

In [12]:
```
total_transactions = df.shape[0]
print(f"Total number of transactions made over the year: {total_transactions}")
```

```
Total number of transactions made over the year: 985
```

Q. What is the distribution of transaction amounts ?

In [13]:
```
# Define small and large transactions
small_threshold = 500

# Create a new column to categorize transactions
df['transaction_category'] = df['amount'].apply(lambda x: 'Small' if x <= small

# Calculate the distribution of transaction amounts
transaction_distribution = df['transaction_category'].value_counts()

print(transaction_distribution)
```

```
transaction_category
Small    745
Large    240
Name: count, dtype: int64
```

Q. Analyze the frequency of different transaction types (debit vs. credit)?

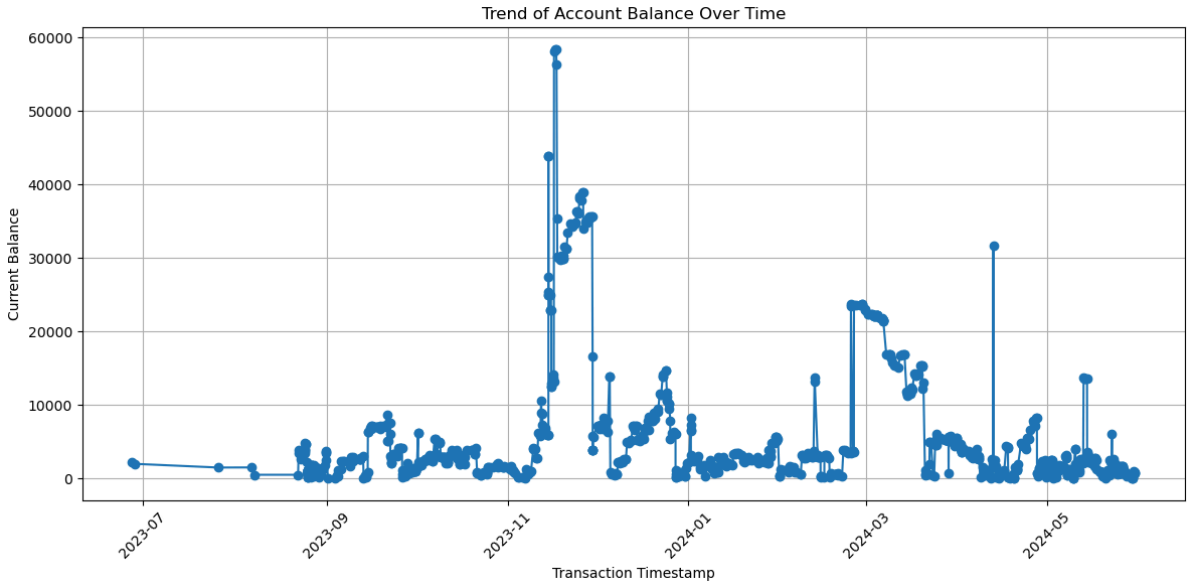
In [14]:
```
transaction_type_frequency = df['type'].value_counts()

print(transaction_type_frequency)
```

```
type
DEBIT     695
CREDIT    290
Name: count, dtype: int64
```

2. Balance Analysis

Q. What is the trend of the account balance over time?

In [15]:
```
# Convert the 'transactionTimestamp' to datetime format
df['transactionTimestamp'] = pd.to_datetime(df['transactionTimestamp'])

# Sort the DataFrame by 'transactionTimestamp'
df = df.sort_values(by='transactionTimestamp')

# Plot the trend of account balance over time
plt.figure(figsize=(12, 6))
plt.plot(df['transactionTimestamp'], df['currentBalance'], marker='o')
plt.xlabel('Transaction Timestamp')
plt.ylabel('Current Balance')
plt.title('Trend of Account Balance Over Time')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

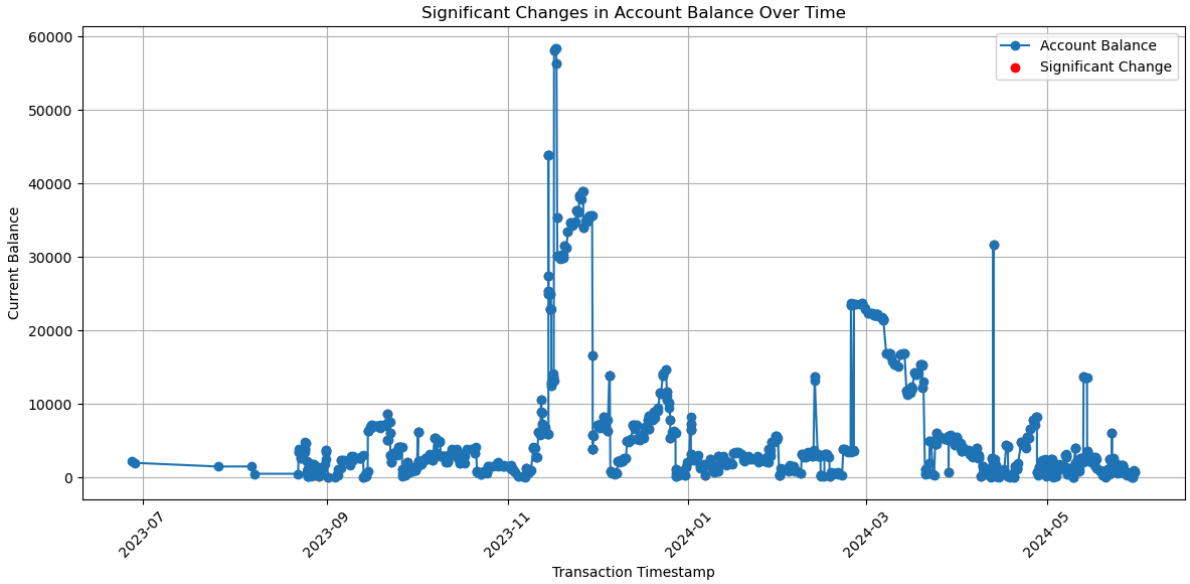


Q. Identify any periods with significant changes in the account balance.

In [16]:

```
# Calculate the balance change between consecutive transactions
df['balance_change'] = df['currentBalance'].diff()

# Define a threshold for significant change (you can adjust this as needed)
threshold = 1000  # Example threshold, change according to your dataset and cri

# Identify periods with significant changes in account balance
significant_changes = df[abs(df['balance_change']) > threshold]

# Print or visualize the periods with significant balance changes
print("Periods with significant changes in account balance:")
print(significant_changes[['transactionTimestamp', 'currentBalance', 'balance_c

# Example of plotting significant changes
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(df['transactionTimestamp'], df['currentBalance'], marker='o', label='A
plt.scatter(significant_changes['transactionTimestamp'], significant_changes['c
plt.xlabel('Transaction Timestamp')
plt.ylabel('Current Balance')
plt.title('Significant Changes in Account Balance Over Time')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Periods with significant changes in account balance:
         transactionTimestamp  currentBalance  balance_change
6   2023-08-22 11:49:13+05:30         3524.80          3000.0
8   2023-08-23 08:17:48+05:30         2624.80         -1200.0
17  2023-08-25 10:24:38+05:30         2244.80         -2480.0
18  2023-08-25 10:39:35+05:30          794.80         -1450.0
36  2023-08-27 12:19:54+05:30          315.80         -1499.0
..                        ...             ...             ...
911 2024-05-17 18:51:36+05:30         1259.31         -1300.0
925 2024-05-21 05:47:33+05:30         1119.31          1070.0
930 2024-05-22 04:42:07+05:30         2454.31          2050.0
935 2024-05-22 20:21:48+05:30         6085.31          3920.0
936 2024-05-22 20:25:35+05:30         2165.31         -3920.0

[150 rows x 3 columns]
```



3. Spending Patterns:

Q. What are the main categories of expenses (e.g., fuel, Ecommerce, food, shopping, ATM withdrawals, UPI transactions)?

In [17]:
```
# Function to categorize expenses based on narration
def categorize_expenses(row):
    if 'FILLING STATION' in row['narration']:
        return 'Fuel'
    elif 'ATM' in row['narration']:
        return 'ATM Withdrawal'
    elif 'UPI' in row['narration']:
        return 'UPI Transaction'
    elif 'PRCR' in row['narration']:
        return 'Ecommerce'
    elif 'IntPd' in row['narration']:
        return 'Interest Paid'
    else:
        return 'Other'

# Apply categorization function to the DataFrame
df['expense_category'] = df.apply(categorize_expenses, axis=1)

# Count occurrences of each expense category
expense_counts = df['expense_category'].value_counts()

# Print or analyze the results
print("Main categories of expenses:")
print(expense_counts)
```

```
Main categories of expenses:
expense_category
UPI Transaction    789
Other              185
Ecommerce            4
Interest Paid        4
ATM Withdrawal       3
Name: count, dtype: int64
```

Q. Analyze the frequency and amount of spending in each category.

In [18]:
```
# Group by expense categories and aggregate frequency and total amount spent
category_summary = df.groupby('expense_category').agg({
    'amount': ['count', 'sum']
}).reset_index()

# Rename columns for clarity
category_summary.columns = ['expense_category', 'transaction_count', 'total_amo

# Sort categories by total amount spent (descending) for better visualization
category_summary = category_summary.sort_values(by='total_amount_spent', ascen

# Print or analyze the results
print("Frequency and amount of spending in each category:")
print(category_summary)
```

```
Frequency and amount of spending in each category:
  expense_category  transaction_count  total_amount_spent
4  UPI Transaction                789           587092.90
3            Other                185           241102.51
0   ATM Withdrawal                  3            13500.00
1        Ecommerce                  4              830.00
2    Interest Paid                  4              135.00
```

4.Income Analysis:

Q. What are the main sources of income (e.g., salary, UPI credits)?

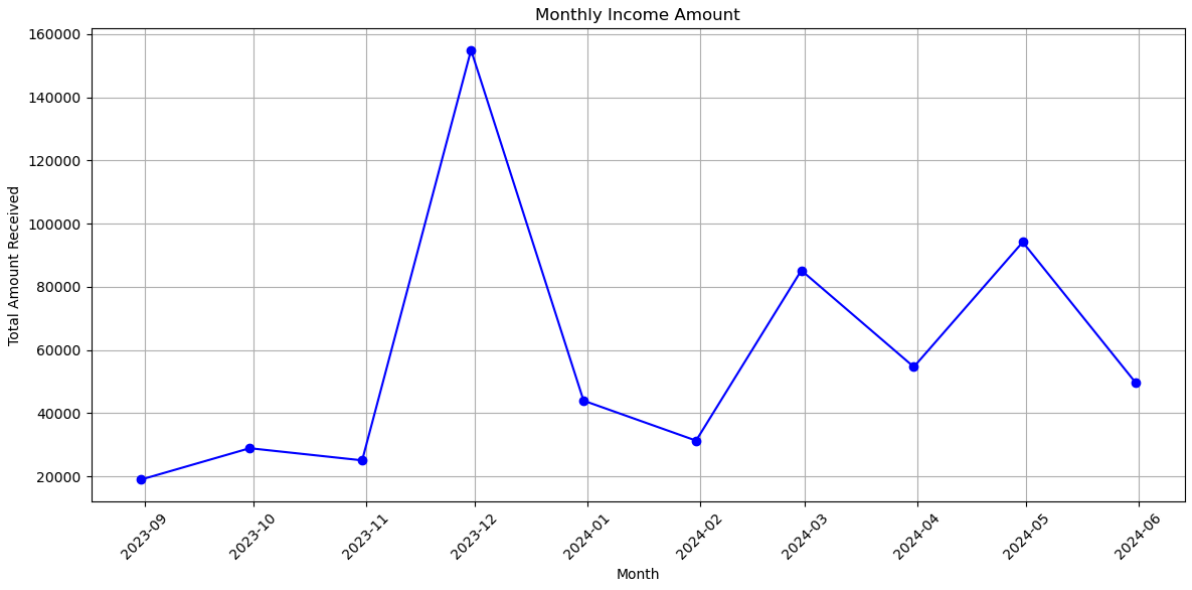
In [19]:
```
# Categorize income sources based on transaction descriptions or other identifi
def categorize_income(transaction_description):
    if 'salary' in transaction_description.lower():
        return 'Salary'
    elif 'upi' in transaction_description.lower():
        return 'UPI Credits'
    else:
        return 'Other Income'

# Apply categorization function to create a new column 'income_source'
df['income_source'] = df['narration'].apply(categorize_income)

# Group by income sources and aggregate frequency and total amount received
income_summary = df[df['income_source'] != 'Other Income'].groupby('income_sour
    'amount': ['count', 'sum']
}).reset_index()

# Rename columns for clarity
income_summary.columns = ['income_source', 'transaction_count', 'total_amount_r

# Sort by total amount received (descending) for better visualization
income_summary = income_summary.sort_values(by='total_amount_received', ascendi

# Print or analyze the results
print("Main sources of income:")
print(income_summary)
```

```
Main sources of income:
  income_source  transaction_count  total_amount_received
0   UPI Credits                789               587092.9
```

Q.Identify any patterns in the timing and amount of income received.

In [20]:
```
import pandas as pd
import matplotlib.pyplot as plt



# Convert 'transactionTimestamp' to datetime format if not already
df['transactionTimestamp'] = pd.to_datetime(df['transactionTimestamp'])

# Extract relevant columns
income_data = df[df['income_source'] != 'Other Income'][['transactionTimestamp'

# Group by month and aggregate total income amount
income_monthly = income_data.resample('M', on='transactionTimestamp').sum()

# Plotting to visualize patterns in income timing and amount
plt.figure(figsize=(12, 6))
plt.plot(income_monthly.index, income_monthly['amount'], marker='o', linestyle=
plt.title('Monthly Income Amount')
plt.xlabel('Month')
plt.ylabel('Total Amount Received')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\NEHA\AppData\Local\Temp\ipykernel_17024\2060818383.py:13: FutureWarn
ing: 'M' is deprecated and will be removed in a future version, please use 'M
E' instead.
  income_monthly = income_data.resample('M', on='transactionTimestamp').sum()
```

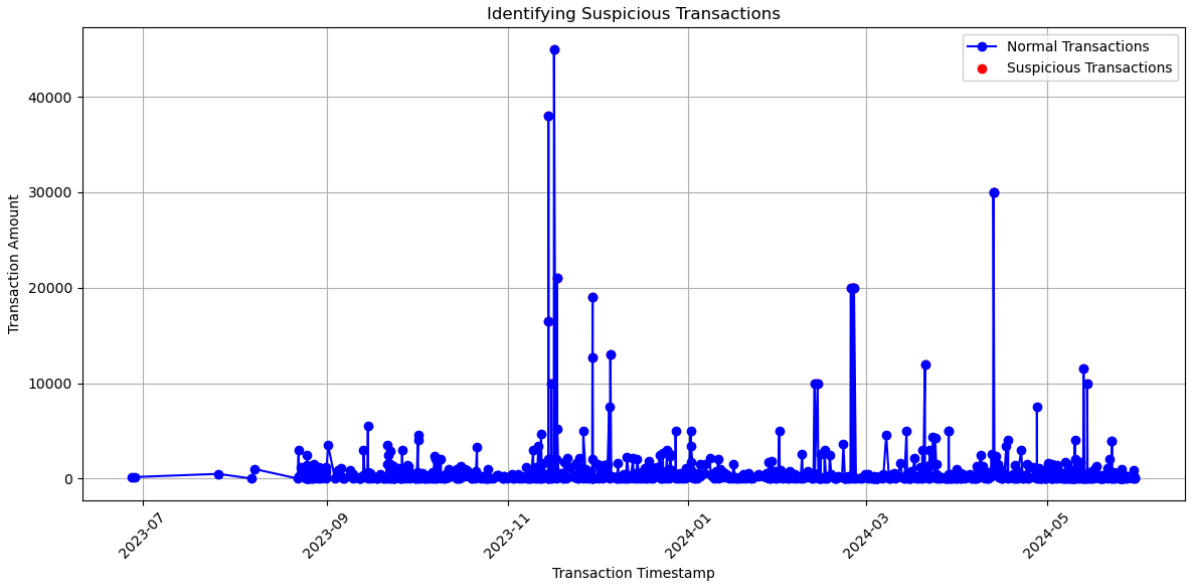


5. Alert Generation:

Q. Identify any unusual or suspicious transactions.

In [21]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Convert 'transactionTimestamp' to datetime format if not already
df['transactionTimestamp'] = pd.to_datetime(df['transactionTimestamp'])

# Extract relevant columns
transaction_data = df[['transactionTimestamp', 'amount']]

# Calculate z-score for 'amount' to identify outliers
z_scores = np.abs(stats.zscore(transaction_data['amount']))

# Define threshold for outlier detection (e.g., z-score > 3)
threshold = 3

# Filter transactions with z-score above threshold as suspicious
suspicious_transactions = transaction_data[z_scores > threshold]

# Visualize suspicious transactions (optional)
plt.figure(figsize=(12, 6))
plt.plot(transaction_data['transactionTimestamp'], transaction_data['amount'],
plt.scatter(suspicious_transactions['transactionTimestamp'], suspicious_transac
plt.title('Identifying Suspicious Transactions')
plt.xlabel('Transaction Timestamp')
plt.ylabel('Transaction Amount')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()

# Output suspicious transactions for further investigation
print("Suspicious Transactions:")
print(suspicious_transactions)
```



```
Suspicious Transactions:
         transactionTimestamp   amount
275 2023-11-14 18:31:11+05:30  37999.0
277 2023-11-14 18:49:41+05:30  16500.0
285 2023-11-15 17:48:21+05:30  10000.0
291 2023-11-16 15:51:14+05:30  45000.0
295 2023-11-17 16:34:54+05:30  21000.0
332 2023-11-29 16:15:33+05:30  19000.0
333 2023-11-29 17:09:47+05:30  12700.0
353 2023-12-05 15:50:06+05:30  13000.0
535 2024-02-12 13:22:57+05:30  10000.0
537 2024-02-13 14:01:51+05:30  10000.0
570 2024-02-24 18:55:15+05:30  20000.0
572 2024-02-25 11:08:34+05:30  20000.0
576 2024-02-25 20:08:58+05:30  20000.0
648 2024-03-20 18:56:48+05:30  12000.0
743 2024-04-12 20:47:44+05:30  30000.0
744 2024-04-12 20:50:06+05:30  30000.0
884 2024-05-13 06:54:41+05:30  11530.0
889 2024-05-14 11:51:56+05:30  10000.0
```

Q Generate alerts for low balance or high expenditure periods.

In [22]:

```
# Convert 'transactionTimestamp' to datetime format if not already
df['transactionTimestamp'] = pd.to_datetime(df['transactionTimestamp'])

# Sort dataframe by 'transactionTimestamp' if not already sorted
df.sort_values(by='transactionTimestamp', inplace=True)

# Calculate cumulative sum of 'amount' to get 'currentBalance' over time
df['currentBalance'] = df['amount'].cumsum()

# Define thresholds for low balance and high expenditure
low_balance_threshold = 1000  # Example threshold for low balance
high_expenditure_threshold = 500  # Example threshold for high expenditure

# Generate alerts for low balance or high expenditure periods
alerts = []

for index, row in df.iterrows():
    if row['currentBalance'] < low_balance_threshold:
        alerts.append(f"Low Balance Alert: Balance is {row['currentBalance']} a

    if row['amount'] > high_expenditure_threshold:
        alerts.append(f"High Expenditure Alert: {row['amount']} spent at {row['

# Display alerts
print("Alerts:")
for alert in alerts:
    print(alert)

# Optional: Visualize account balance over time
plt.figure(figsize=(12, 6))
plt.plot(df['transactionTimestamp'], df['currentBalance'], marker='o', linestyl
plt.axhline(y=low_balance_threshold, color='r', linestyle='--', label=f'Low Bal
plt.axhline(y=df['currentBalance'].mean(), color='g', linestyle='--', label='Av
plt.title('Account Balance Over Time')
plt.xlabel('Transaction Timestamp')
plt.ylabel('Account Balance')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Alerts:
Low Balance Alert: Balance is 100.0 at 2023-06-27 09:40:19+05:30
Low Balance Alert: Balance is 270.0 at 2023-06-28 09:51:57+05:30
Low Balance Alert: Balance is 770.0 at 2023-07-26 10:04:00+05:30
Low Balance Alert: Balance is 785.0 at 2023-08-06 11:10:38+05:30
High Expenditure Alert: 1000.0 spent at 2023-08-07 17:13:13+05:30
High Expenditure Alert: 3000.0 spent at 2023-08-22 11:49:13+05:30
High Expenditure Alert: 1200.0 spent at 2023-08-23 08:17:48+05:30
High Expenditure Alert: 750.0 spent at 2023-08-24 18:22:22+05:30
High Expenditure Alert: 2480.0 spent at 2023-08-25 10:24:38+05:30
High Expenditure Alert: 1450.0 spent at 2023-08-25 10:39:35+05:30
High Expenditure Alert: 1000.0 spent at 2023-08-25 16:56:59+05:30
High Expenditure Alert: 700.0 spent at 2023-08-26 20:36:16+05:30
High Expenditure Alert: 1499.0 spent at 2023-08-27 12:19:54+05:30
High Expenditure Alert: 850.0 spent at 2023-08-28 20:27:41+05:30
High Expenditure Alert: 1200.0 spent at 2023-08-29 11:49:00+05:30
High Expenditure Alert: 600.0 spent at 2023-08-29 20:02:21+05:30
High Expenditure Alert: 600.0 spent at 2023-08-30 16:19:31+05:30
High Expenditure Alert: 750.0 spent at 2023-08-31 13:42:41+05:30
```

In [ ]: