In [1]:
```python
# TASK 1: DATA CLEANING & FORMATTING
import pandas as pd
import numpy as np
```

In [8]:
```python
# Load dataset
df = pd.read_excel(r"C:\Users\hp\OneDrive\Desktop\barclays.xlsx")
```

In [10]:
```python
print(df.head())
```

```
  TransactionID CustomerID AccountID AccountType TransactionType  \
0           118   CUST3810  ACC49774     Savings         Deposit
1           102   CUST3109  ACC96277     Savings         Deposit
2           151   CUST2626  ACC21429      Credit         Payment
3            57   CUST3725  ACC48501        Loan      Withdrawal
4           113   CUST4258  ACC11285        Loan        Transfer

        Product    Firm Region    Manager      TransactionDate  \
0    Credit Card  Firm D   West  Manager 4  2024-08-01 00:00:00
1    Mutual Fund  Firm B  North  Manager 4           17-12-2023
2  Personal Loan  Firm C   West  Manager 1           22-05-2024
3    Credit Card  Firm A   East  Manager 4           24-12-2023
4      Home Loan  Firm A   West  Manager 4           15-01-2023

  TransactionAmount  AccountBalance  RiskScore  CreditRating  TenureMonths
0      20664.409820     88483.42208   0.483333           522            29
1      94924.359120     56670.15864   0.788989           686           130
2      -7871.160407     84968.05587   0.547782           618           157
3      24979.808160    115196.96420   0.125587           803           155
4      72890.748550    111602.76610   1.048787           657            68
```

In [11]:
```python
# 1. Standardize Column Names
df.columns = [c.strip() for c in df.columns]
```

In [12]:
```python
# 2. Clean & Format Date Column
df["TransactionDate"] = pd.to_datetime(
    df["TransactionDate"],
    dayfirst=True,
    errors="coerce")
```

In [13]:
```python
# 3. Clean Currency / Numerdef clean_numeric(col):
def clean_numeric(col):
    return pd.to_numeric(col, errors="coerce")

df["TransactionAmount_clean"] = clean_numeric(df["TransactionAmount"])
df["AccountBalance_clean"] = clean_numeric(df["AccountBalance"])
```

In [14]:
```python
# 4. Standardize Transaction Type
df["TransactionType_norm"] = (
    df["TransactionType"]
    .astype(str)
    .str.upper()
    .str.strip()
)
```

```python
df["txn_cd"] = df["TransactionType_norm"].replace({
    "CR": "CREDIT",
    "DR": "DEBIT",
    "CREDIT": "CREDIT",
    "DEBIT": "DEBIT",
    "WITHDRAWAL": "DEBIT",
    "DEPOSIT": "CREDIT",
    "TRANSFER": "DEBIT",
    "PAYMENT": "DEBIT"
})

# If any type is missing, infer from amount sign
df.loc[df["txn_cd"].isna(), "txn_cd"] = df.loc[
    df["txn_cd"].isna(), "TransactionAmount_clean"
].apply(lambda x: "DEBIT" if x < 0 else ("CREDIT" if x > 0 else np.nan))
```

In [15]:
```python
# 5. Drop rows with missing essential fields
before = df.shape[0]

df = df.dropna(subset=["TransactionDate", "TransactionAmount_clean"])

after = df.shape[0]
print(f"Rows removed in cleaning: {before - after}")
```

```
Rows removed in cleaning: 0
```

In [16]:
```python
# 6. Final cleaned dataset
df_clean = df.copy()

print("TASK 1 COMPLETED — Data cleaned successfully!")
df_clean.head()
```

```
TASK 1 COMPLETED — Data cleaned successfully!
```

Out[16]:

	TransactionID	CustomerID	AccountID	AccountType	TransactionType	Product	Firm
0	118	CUST3810	ACC49774	Savings	Deposit	Credit Card	Firm D
1	102	CUST3109	ACC96277	Savings	Deposit	Mutual Fund	Firm B
2	151	CUST2626	ACC21429	Credit	Payment	Personal Loan	Firm C
3	57	CUST3725	ACC48501	Loan	Withdrawal	Credit Card	Firm A
4	113	CUST4258	ACC11285	Loan	Transfer	Home Loan	Firm A

In [17]:
```python
# TASK 2: DESCRIPTIVE TRANSACTIONAL ANALYSIS
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [18]:
```python
df = pd.read_excel(r"C:\Users\hp\OneDrive\Desktop\barclays.xlsx")
df["TransactionDate"] = pd.to_datetime(df["TransactionDate"], dayfirst=True, errors
df["amount"] = pd.to_numeric(df["TransactionAmount"], errors="coerce")
df["balance"] = pd.to_numeric(df["AccountBalance"], errors="coerce")
df_clean = df.copy()
df_clean = df_clean.dropna(subset=["TransactionDate", "amount"])
df_clean = df_clean.rename(columns={"TransactionDate": "txn_date", "AccountID": "ac
```

In [20]:
```python
# 1. MONTHLY / YEARLY SUMMARIES
# Extract month and year
df_clean["year"] = df_clean["txn_date"].dt.year
df_clean["month"] = df_clean["txn_date"].dt.to_period("M").astype(str)

# Create credit and debit columns
df_clean["credit_amt"] = df_clean["amount"].apply(lambda x: x if x > 0 else 0)
df_clean["debit_amt"] = df_clean["amount"].apply(lambda x: -x if x < 0 else 0)

# ----- Monthly Summary -----
monthly_summary = df_clean.groupby("month").agg(
    total_credits=("credit_amt", "sum"),
    total_debits=("debit_amt", "sum"),
    net_flow=("amount", "sum"),
    txn_count=("amount", "count")
).reset_index()

print("\nMONTHLY SUMMARY:")
print(monthly_summary)
```

```
MONTHLY SUMMARY:
      month  total_credits  total_debits      net_flow  txn_count
0   2023-01   3.129819e+06  21678.504760  3.108140e+06         56
1   2023-02   1.791555e+06      0.000000  1.791555e+06         35
2   2023-03   9.006768e+05   3955.291271  8.967215e+05         20
3   2023-04   2.077770e+06      0.000000  2.077770e+06         36
4   2023-05   2.277883e+06   2920.562029  2.274963e+06         42
5   2023-06   3.366631e+06  20209.220980  3.346422e+06         64
6   2023-07   1.932753e+06      0.000000  1.932753e+06         32
7   2023-08   3.321373e+06  16065.179120  3.305308e+06         58
8   2023-09   1.945303e+06  76572.211768  1.868730e+06         37
9   2023-10   4.300185e+06   4859.967874  4.295325e+06         81
10  2023-11   3.455663e+06      0.000000  3.455663e+06         57
11  2023-12   2.039669e+06      0.000000  2.039669e+06         35
12  2024-01   1.615146e+06  68071.807681  1.547074e+06         29
13  2024-02   1.596122e+06  28414.948580  1.567707e+06         35
14  2024-03   1.862764e+06  24201.021900  1.838563e+06         38
15  2024-04   1.753062e+06   7398.591120  1.745664e+06         34
16  2024-05   2.106700e+06   7871.160407  2.098829e+06         38
17  2024-06   1.737806e+06   3723.918966  1.734083e+06         27
18  2024-07   1.759714e+05      0.000000  1.759714e+05          2
19  2024-08   1.030467e+06      0.000000  1.030467e+06         18
20  2024-09   2.296174e+05      0.000000  2.296174e+05          6
21  2024-11   1.967346e+05  10248.516872  1.864861e+05          7
22  2024-12   4.377129e+05  29124.624850  4.085882e+05         13
```

In [21]:
```
# ----- Yearly Summary -----
yearly_summary = df_clean.groupby("year").agg(
    total_credits=("amount", lambda x: x[x > 0].sum()),
    total_debits=("amount", lambda x: -x[x < 0].sum()),
    net_flow=("amount", "sum"),
    txn_count=("amount", "count")
).reset_index()

print("\nYEARLY SUMMARY:")
print(yearly_summary)
```

```
YEARLY SUMMARY:
   year  total_credits   total_debits      net_flow  txn_count
0  2023   3.053928e+07  146260.937802  3.039302e+07        553
1  2024   1.274210e+07  179054.590376  1.256305e+07        247
```

In [22]:
```
# 2. TOP & BOTTOM ACCOUNTS BY NET INFLOW
acct_performance = df_clean.groupby("account_id").agg(
    net_inflow=("amount", "sum"),
    txn_count=("amount", "count"),
    avg_balance=("balance", "mean")
).reset_index().sort_values("net_inflow", ascending=False)

print("\nTOP 5 ACCOUNTS (Net Inflow):")
print(acct_performance.head())

print("\nBOTTOM 5 ACCOUNTS (Net Inflow):")
print(acct_performance.tail())
```

```
TOP 5 ACCOUNTS (Net Inflow):
    account_id    net_inflow  txn_count   avg_balance
109   ACC54589  720103.59559         10  54155.282135
180   ACC92558  645089.10225          9  50319.279200
97    ACC49422  564943.53345         11  62362.266610
156   ACC80131  556260.45347          8  84181.033307
79    ACC42710  552578.28377         10  73091.720323

BOTTOM 5 ACCOUNTS (Net Inflow):
    account_id    net_inflow  txn_count    avg_balance
104   ACC51971  27219.074646          2   64424.243320
11    ACC15671  25166.389040          1  120586.085000
106   ACC52650  19904.777256          2   63233.764705
57    ACC32212   1592.907285          1   76367.387200
41    ACC28154  -2920.562029          1   95667.606520
```

In [23]:
```
# 3. FLAG DORMANT / INACTIVE ACCOUNTS
def find_dormancy(group):
    group = group.sort_values("txn_date")
    group["prev_date"] = group["txn_date"].shift(1)
    group["gap_days"] = (group["txn_date"] - group["prev_date"]).dt.days
    group["dormant_flag"] = group["gap_days"] >= 60
    return group

dormant_df = df_clean.groupby("account_id", group_keys=False).apply(find_dormancy)

dormant_accounts = dormant_df.groupby("account_id")["dormant_flag"].any().reset_ind
```

```python
print("\nDORMANT ACCOUNTS (60+ days inactivity):")
print(dormant_accounts[dormant_accounts["dormant_flag"] == True])
```

```
DORMANT ACCOUNTS (60+ days inactivity):
    account_id  dormant_flag
0     ACC10117          True
1     ACC10996          True
2     ACC11062          True
3     ACC11188          True
4     ACC11285          True
..         ...           ...
188   ACC97225          True
189   ACC97411          True
190   ACC99117          True
191   ACC99409          True
192   ACC99549          True

[161 rows x 2 columns]
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_14224\4151369234.py:9: DeprecationWarning:
DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecate
d, and in a future version of pandas the grouping columns will be excluded from the
operation. Either pass `include_groups=False` to exclude the groupings or explicitly
select the grouping columns after groupby to silence this warning.
  dormant_df = df_clean.groupby("account_id", group_keys=False).apply(find_dormancy)
```

In [24]:

```python
# TASK 3: CUSTOMER PROFILE BUILDING
# 1. Create transaction activity statistics per account
acct_stats = df_clean.groupby("account_id").agg(
    txn_count_total=("amount", "count"),
    first_txn=("txn_date", "min"),
    last_txn=("txn_date", "max"),
    avg_balance=("balance", "mean")
).reset_index()
```

In [25]:

```python
# Calculate time period for each account
acct_stats["period_days"] = (acct_stats["last_txn"] - acct_stats["first_txn"]).dt.d
```

In [26]:

```python
# Transactions per year & month
acct_stats["txn_per_year"] = acct_stats["txn_count_total"] / (acct_stats["period_da
acct_stats["avg_txn_per_month"] = acct_stats["txn_count_total"] / (acct_stats["peri
```

In [27]:

```python
# 2. Activity Level Rubric
def activity_level(row):
    if row["txn_per_year"] > 120 or row["avg_txn_per_month"] > 10:
        return "High"
    elif row["txn_per_year"] >= 36 or row["avg_txn_per_month"] >= 3:
        return "Medium"
    else:
        return "Low"

acct_stats["activity_level"] = acct_stats.apply(activity_level, axis=1)
```

In [28]:

```python
# 3. Balance segmentation (Low / Medium / High)
def balance_bucket(balance):
```

```python
    if pd.isna(balance):
        return "Unknown"
    if balance < 1000:
        return "Low"
    elif balance <= 10000:
        return "Medium"
    else:
        return "High"

acct_stats["balance_bucket"] = acct_stats["avg_balance"].apply(balance_bucket)

print("\nACCOUNT ACTIVITY PROFILE:")
print(acct_stats.head())
```

```
ACCOUNT ACTIVITY PROFILE:
  account_id  txn_count_total  first_txn   last_txn   avg_balance  \
0   ACC10117                4 2023-01-06 2024-06-22  97828.704775
1   ACC10996                5 2023-01-17 2024-06-21  56982.152538
2   ACC11062                2 2023-08-11 2024-04-02  65947.316965
3   ACC11188                4 2023-04-26 2023-12-24  81169.114065
4   ACC11285                3 2023-01-15 2024-03-30  62574.613950

   period_days  txn_per_year  avg_txn_per_month activity_level balance_bucket
0          534      2.735955           0.227996            Low           High
1          522      3.498563           0.291547            Low           High
2          236      3.095339           0.257945            Low           High
3          243      6.012346           0.501029            Low           High
4          441      2.484694           0.207058            Low           High
```

In [32]:

```python
# 4. Profile Categories
# High Net Inflow accounts
acct_performance = df_clean.groupby("account_id").agg(
    net_inflow=("amount", "sum")
).reset_index()

high_net_inflow = acct_performance[acct_performance["net_inflow"] > 0] \
                    .sort_values("net_inflow", ascending=False)
print("\nHIGH NET INFLOW ACCOUNTS:")
print(high_net_inflow.head())
```

```
HIGH NET INFLOW ACCOUNTS:
    account_id    net_inflow
109   ACC54589  720103.59559
180   ACC92558  645089.10225
97    ACC49422  564943.53345
156   ACC80131  556260.45347
79    ACC42710  552578.28377
```

In [33]:

```python
# High frequency but low balance accounts
high_freq_low_balance = acct_stats[
    (acct_stats["activity_level"] == "High") &
    (acct_stats["balance_bucket"] == "Low")
]
print("\nHIGH FREQUENCY + LOW BALANCE ACCOUNTS:")
print(high_freq_low_balance.head())
```

```
HIGH FREQUENCY + LOW BALANCE ACCOUNTS:
Empty DataFrame
Columns: [account_id, txn_count_total, first_txn, last_txn, avg_balance, period_day
s, txn_per_year, avg_txn_per_month, activity_level, balance_bucket]
Index: []
```

In [34]:

```
# Accounts with negative or near-zero balance
near_zero_negative = acct_stats[acct_stats["avg_balance"] <= 0]
print("\nACCOUNTS WITH NEAR-ZERO OR NEGATIVE BALANCE:")
print(near_zero_negative.head())
```

```
ACCOUNTS WITH NEAR-ZERO OR NEGATIVE BALANCE:
Empty DataFrame
Columns: [account_id, txn_count_total, first_txn, last_txn, avg_balance, period_day
s, txn_per_year, avg_txn_per_month, activity_level, balance_bucket]
Index: []
```

In [35]:

```
# TASK 4: FINANCIAL RISK IDENTIFICATION
import pandas as pd
import numpy as np
# 1. Detect Large Withdrawals (Top 1% Debit Transactions)
# Filter only debit transactions
debits = df_clean[df_clean["amount"] < 0].copy()

# 99th percentile threshold
withdrawal_threshold = debits["amount"].abs().quantile(0.99)

# Flag large withdrawals
debits["large_withdrawal"] = debits["amount"].abs() >= withdrawal_threshold

large_withdrawals = debits[debits["large_withdrawal"] == True]

print("\nLARGE WITHDRAWALS DETECTED:")
print(large_withdrawals.head())


# 2. Detect Overdrafts (Negative Balance)


overdrafts = df_clean[df_clean["balance"] < 0].sort_values("balance")

print("\nOVERDRAFT ACCOUNTS (Negative Balances):")
print(overdrafts.head())


# 3. Balance Volatility (Std Dev & Coefficient of Variation)


bal_vol = df_clean.groupby("account_id")["balance"].agg(["std", "mean"]).reset_inde
bal_vol = bal_vol.rename(columns={"std": "balance_std", "mean": "balance_mean"})

# Coefficient of Variation = std / mean
bal_vol["coeff_var"] = bal_vol["balance_std"] / bal_vol["balance_mean"].abs().repla

# Sort by highest volatility
bal_vol_sorted = bal_vol.sort_values("coeff_var", ascending=False)
```

```python
print("\nACCOUNTS WITH HIGH BALANCE VOLATILITY:")
print(bal_vol_sorted.head())


# 4. Anomaly Detection using Z-score


# Compute per-account Z-score for transaction amounts
df_clean["amount_z"] = df_clean.groupby("account_id")["amount"].transform(
    lambda x: (x - x.mean()) / x.std(ddof=0)
)

# Flag anomalies where |z| > 3
df_clean["anomaly_flag"] = df_clean["amount_z"].abs() > 3

anomalies = df_clean[df_clean["anomaly_flag"] == True]

print("\nANOMALOUS TRANSACTIONS (Z-score Method):")
print(anomalies.head())


# 5. Suspicious Account Identification
# (Any account with anomalies, overdrafts, or large withdrawals)


suspicious_accounts = set(large_withdrawals["account_id"]) \
                      | set(overdrafts["account_id"]) \
                      | set(anomalies["account_id"])

suspicious_accounts_df = pd.DataFrame({"account_id": list(suspicious_accounts)})

print("\nSUSPICIOUS ACCOUNTS:")
print(suspicious_accounts_df.head())
```

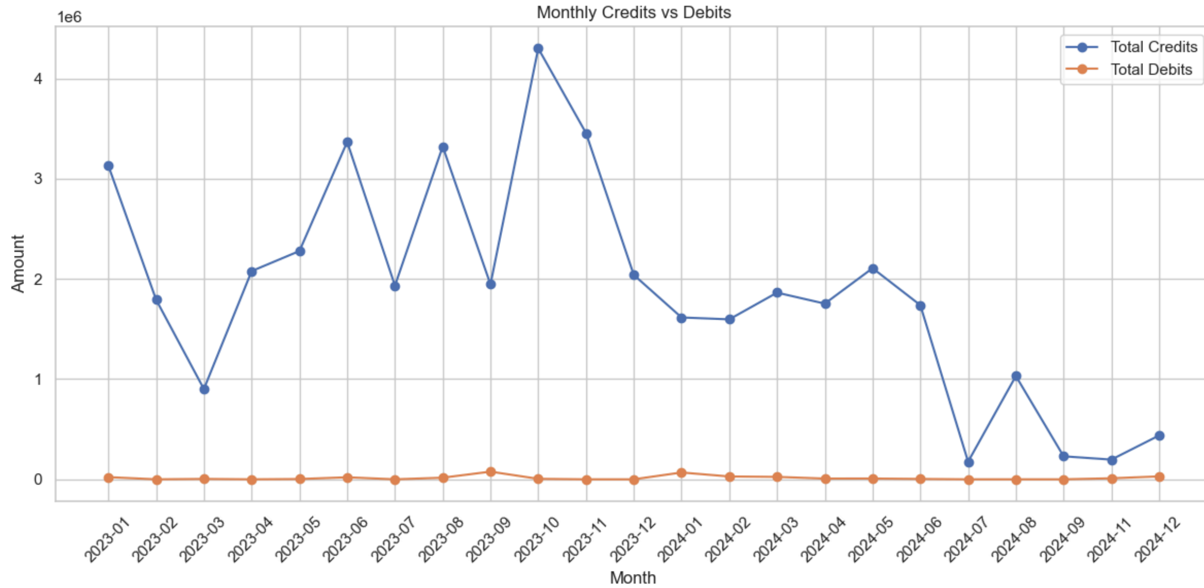
```
LARGE WITHDRAWALS DETECTED:
     TransactionID CustomerID account_id AccountType TransactionType  \
589            190   CUST1738   ACC90887     Savings      Withdrawal   

             Product    Firm Region    Manager   txn_date  ...  RiskScore  \
589  Savings Account  Firm C   East  Manager 2 2024-01-20  ...   0.226499   

     CreditRating  TenureMonths       amount      balance  year    month  \
589           502           172 -59669.07548  56004.32009  2024  2024-01   

     credit_amt    debit_amt  large_withdrawal  
589         0.0  59669.07548              True  

[1 rows x 22 columns]

OVERDRAFT ACCOUNTS (Negative Balances):
     TransactionID CustomerID account_id AccountType TransactionType  \
150             96   CUST8091   ACC42467        Loan        Transfer   
236             99   CUST4584   ACC11285      Credit         Payment   
281              9   CUST2109   ACC77592     Savings         Payment   
789            141   CUST3810   ACC72197      Credit      Withdrawal   
412             48   CUST6082   ACC71938        Loan         Payment   

             Product    Firm   Region    Manager   txn_date  ...  \
150    Personal Loan  Firm C    South  Manager 4 2023-10-21  ...   
236      Mutual Fund  Firm D     West  Manager 1 2024-03-30  ...   
281      Mutual Fund  Firm C  Central  Manager 1 2024-03-22  ...   
789        Home Loan  Firm E     West  Manager 4 2024-01-29  ...   
412  Savings Account  Firm B    North  Manager 2 2024-05-03  ...   

     AccountBalance  RiskScore  CreditRating  TenureMonths        amount  \
150    -30766.90697   0.404256           375            39   15062.59519   
236    -17751.21681   0.812135           724            30   88130.72313   
281    -14999.18083   0.536948           416            85  134330.67900   
789    -14934.03449   0.305936           836           236   12927.63720   
412    -12493.95639  -0.000778           529           118  109852.80310   

         balance  year    month    credit_amt  debit_amt  
150 -30766.90697  2023  2023-10   15062.59519        0.0  
236 -17751.21681  2024  2024-03   88130.72313        0.0  
281 -14999.18083  2024  2024-03  134330.67900        0.0  
789 -14934.03449  2024  2024-01   12927.63720        0.0  
412 -12493.95639  2024  2024-05  109852.80310        0.0  

[5 rows x 21 columns]

ACCOUNTS WITH HIGH BALANCE VOLATILITY:
    account_id   balance_std  balance_mean  coeff_var
110   ACC55331  73097.164038  54330.176256   1.345425
141   ACC74631  50452.054088  44700.284518   1.128674
4     ACC11285  70126.826097  62574.613950   1.120691
78    ACC42467  52980.932871  47684.159638   1.111080
134   ACC70460  36970.340677  33835.403142   1.092653

ANOMALOUS TRANSACTIONS (Z-score Method):
Empty DataFrame
```

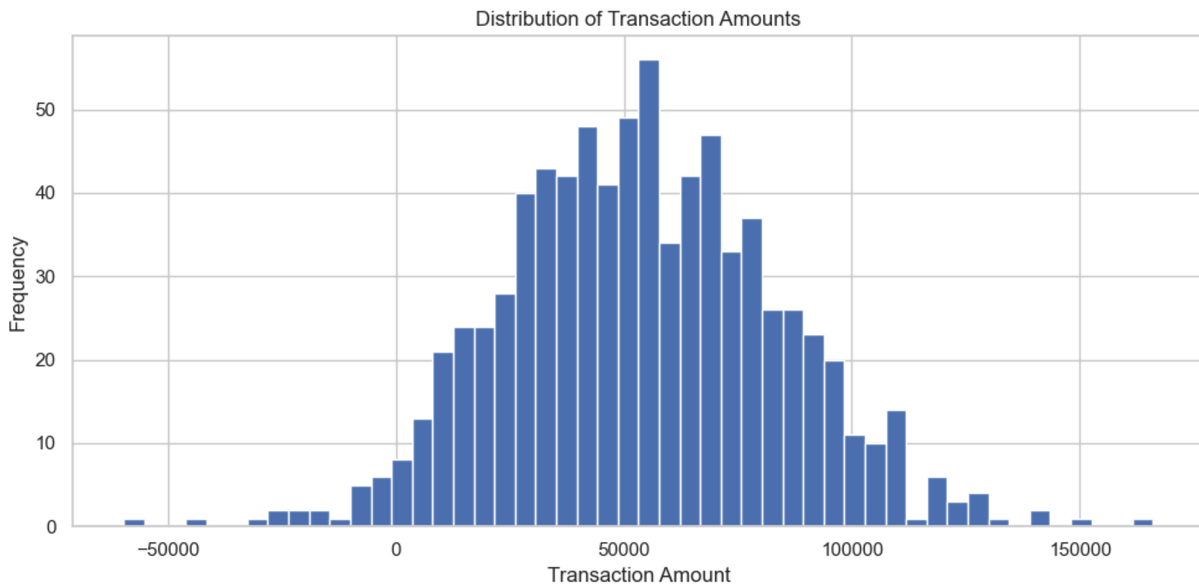
```
Columns: [TransactionID, CustomerID, account_id, AccountType, TransactionType, Produ
ct, Firm, Region, Manager, txn_date, TransactionAmount, AccountBalance, RiskScore, C
reditRating, TenureMonths, amount, balance, year, month, credit_amt, debit_amt, amou
nt_z, anomaly_flag]
Index: []

[0 rows x 23 columns]

SUSPICIOUS ACCOUNTS:
  account_id
0   ACC72197
1   ACC11285
2   ACC19156
3   ACC45521
4   ACC49422
```

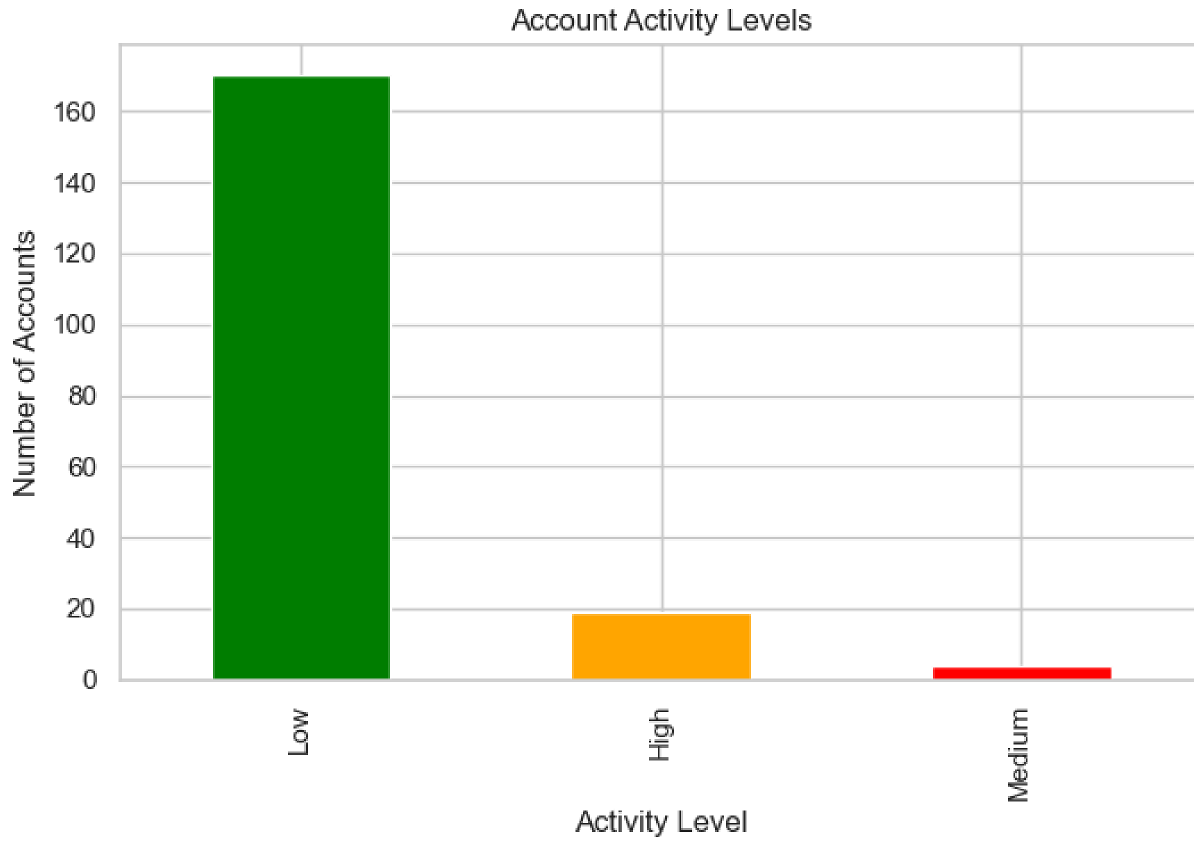
In [37]:

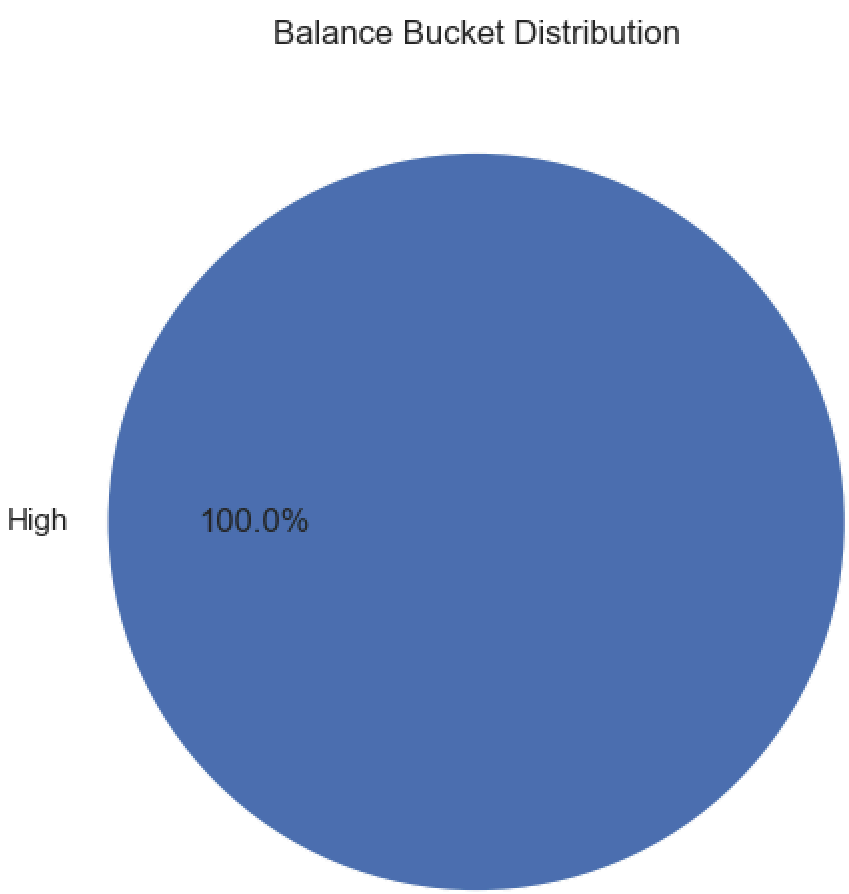
```python
# TASK 5: VISUALIZATIONS (EDA PLOTS)
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")


# 1. Monthly Credits vs Debits (Line Chart)


plt.figure(figsize=(12, 6))
plt.plot(monthly_summary["month"], monthly_summary["total_credits"], marker="o", la
plt.plot(monthly_summary["month"], monthly_summary["total_debits"], marker="o", lab
plt.xticks(rotation=45)
plt.xlabel("Month")
plt.ylabel("Amount")
plt.title("Monthly Credits vs Debits")
plt.legend()
plt.tight_layout()
plt.show()


# 2. Distribution of Transaction Amounts


plt.figure(figsize=(10, 5))
plt.hist(df_clean["amount"], bins=50)
plt.xlabel("Transaction Amount")
plt.ylabel("Frequency")
plt.title("Distribution of Transaction Amounts")
plt.tight_layout()
plt.show()


# 3. Activity Levels (Bar Chart)


plt.figure(figsize=(7, 5))
acct_stats["activity_level"].value_counts().plot(kind="bar", color=["green", "orang
plt.xlabel("Activity Level")
plt.ylabel("Number of Accounts")
```

```python
plt.title("Account Activity Levels")
plt.tight_layout()
plt.show()


# 4. Balance Bucket Segmentation (Pie Chart)


plt.figure(figsize=(6, 6))
acct_stats["balance_bucket"].value_counts().plot(kind="pie", autopct="%1.1f%%")
plt.title("Balance Bucket Distribution")
plt.ylabel("")
plt.show()


# 5. Net Inflow per Account (Top 10)


top10 = acct_performance.sort_values("net_inflow", ascending=False).head(10)

plt.figure(figsize=(10,5))
sns.barplot(x="account_id", y="net_inflow", data=top10, palette="viridis")
plt.xticks(rotation=45)
plt.title("Top 10 Accounts by Net Inflow")
plt.xlabel("Account ID")
plt.ylabel("Net Inflow Amount")
plt.tight_layout()
plt.show()


# 6. Heatmap: Correlation Between Numeric Variables


plt.figure(figsize=(8, 5))
sns.heatmap(df_clean[["amount", "balance"]].corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.tight_layout()
plt.show()

# 7. Transaction Volume Over Time


df_clean["date_only"] = df_clean["txn_date"].dt.date
txn_volume = df_clean.groupby("date_only")["amount"].count()

plt.figure(figsize=(12, 5))
plt.plot(txn_volume.index, txn_volume.values, color="purple")
plt.xlabel("Date")
plt.ylabel("Number of Transactions")
plt.title("Daily Transaction Volume Trend")
plt.tight_layout()
plt.show()
```

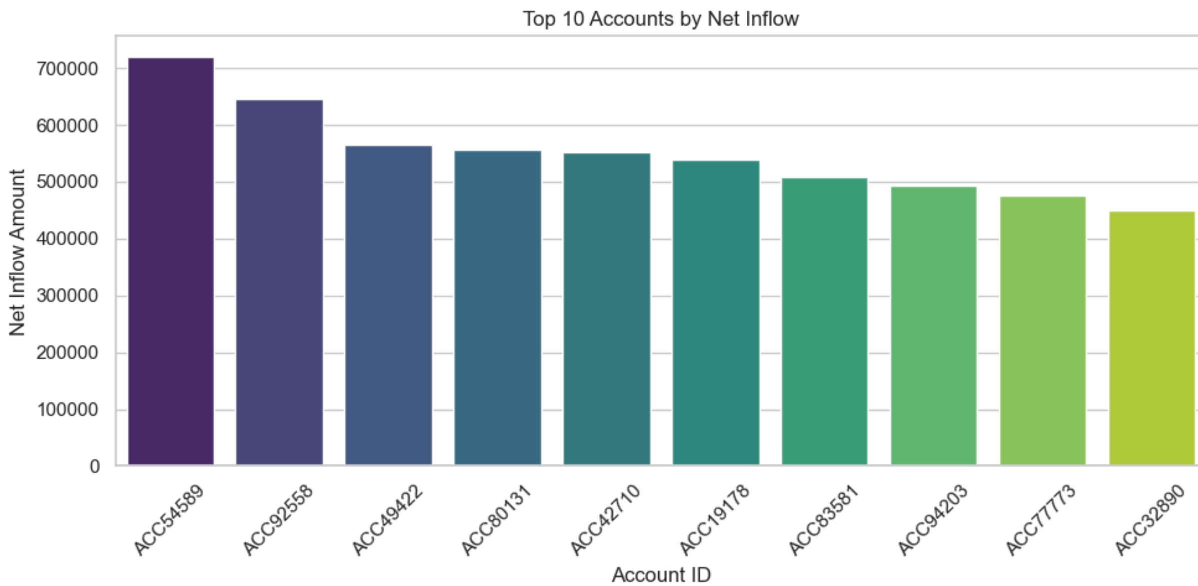

Monthly Credits vs Debits
1e6
Amount
Month
Total Credits
Total Debits
0
1
2
3
4
2023-01
2023-02
2023-03
2023-04
2023-05
2023-06
2023-07
2023-08
2023-09
2023-10
2023-11
2023-12
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
2024-09
2024-11
2024-12

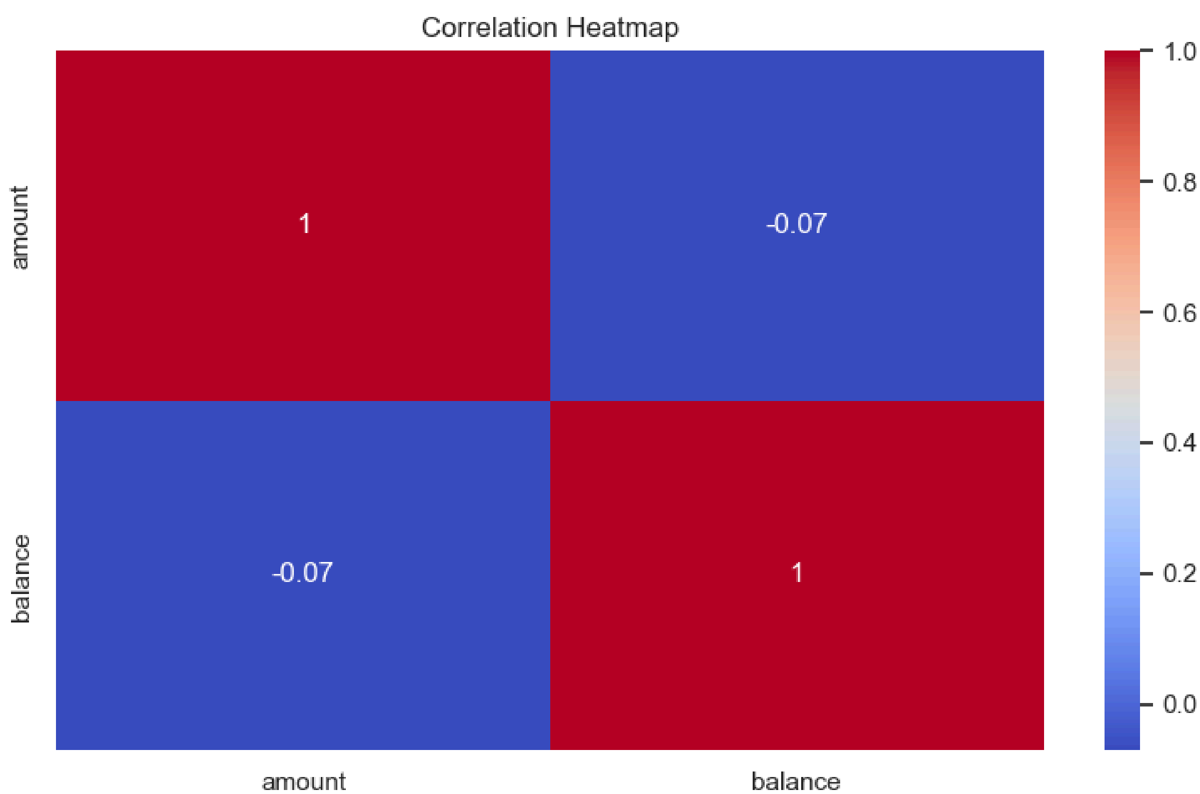

Distribution of Transaction Amounts
Frequency
Transaction Amount
0
10
20
30
40
50
−50000
0
50000
100000
150000

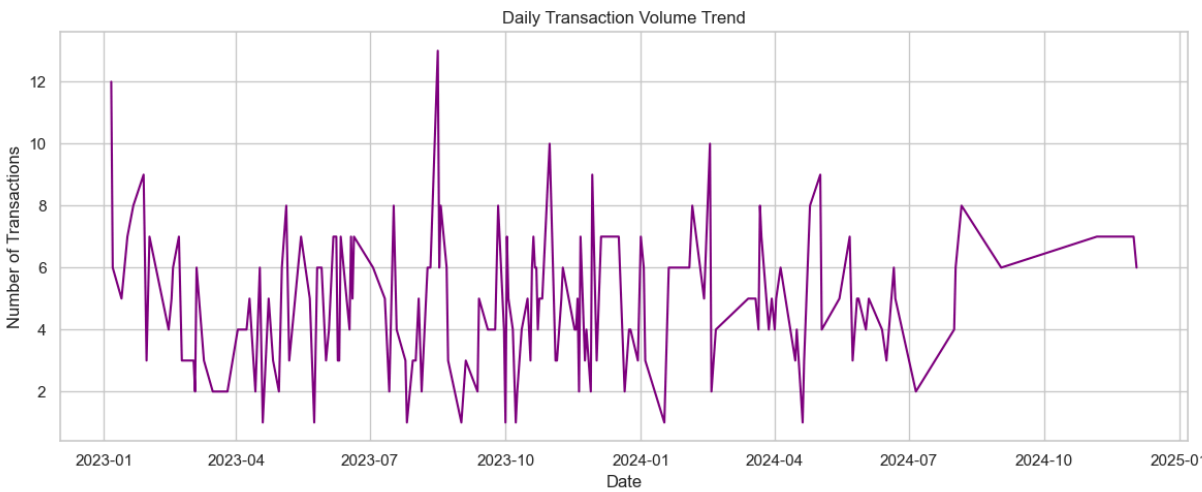

Account Activity Levels
Number of Accounts
0
20
40
60
80
100
120
140
160
Low
High
Medium
Activity Level


Balance Bucket Distribution
High
100.0%

```
C:\Users\hp\AppData\Local\Temp\ipykernel_14224\1854793806.py:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x="account_id", y="net_inflow", data=top10, palette="viridis")
```







In [38]:

```python
# TASK 6: HYPOTHESIS TESTING
import pandas as pd
import numpy as np
from scipy import stats
# Extract groups
high_group = acct_stats[acct_stats["activity_level"] == "High"]["avg_balance"].drop
low_group  = acct_stats[acct_stats["activity_level"] == "Low"]["avg_balance"].dropn

print("\nNumber of high-volume accounts:", len(high_group))
print("Number of low-volume accounts:", len(low_group))


# Perform Welch’s t-test (safe even with unequal variances)


if len(high_group) >= 5 and len(low_group) >= 5:

    tstat, pvalue = stats.ttest_ind(
        high_group,
        low_group,
        equal_var=False,     # Welch's t-test
        nan_policy='omit'
    )

    print("\n------ HYPOTHESIS TEST RESULT ------")
    print("T-statistic:", tstat)
    print("P-value:", pvalue)

    # Interpretation
    alpha = 0.05
    if pvalue < alpha:
        print("\nConclusion: Reject H0")
        print("High-volume accounts have statistically higher average balances.")
    else:
        print("\nConclusion: Fail to Reject H0")
        print("No significant difference between balances of high- and low-volume a

else:
    print("\nNot enough data to run the test (need ≥ 5 accounts in each group).")
```

```
Number of high-volume accounts: 19
Number of low-volume accounts: 170

------ HYPOTHESIS TEST RESULT ------
T-statistic: 0.2506171047017368
P-value: 0.8047125420463144

Conclusion: Fail to Reject H0
No significant difference between balances of high- and low-volume accounts.
```

In [ ]:

```
##---- link vedio presentation----
[watch vedio] (https://www.loom.com/share/bb740adb865c4305816b3f4be11add02)
```