Task 1: Data Cleaning and Formatting

1.Remove/treat any special characters or non-numeric entries from financial fields.

In [61]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import re
```

In [62]:
```
df = pd.read_csv("Barclays Financial Transactional Data - barclays.csv")
df.head()
```

Out[62]:

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

In [63]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   TransactionID      800 non-null    int64
 1   CustomerID         800 non-null    object
 2   AccountID          800 non-null    object
 3   AccountType        800 non-null    object
 4   TransactionType    800 non-null    object
 5   Product            800 non-null    object
 6   Firm               800 non-null    object
 7   Region             800 non-null    object
 8   Manager            800 non-null    object
 9   TransactionDate    800 non-null    object
 10  TransactionAmount  800 non-null    float64
 11  AccountBalance     800 non-null    float64
 12  RiskScore          800 non-null    float64
 13  CreditRating       800 non-null    int64
 14  TenureMonths       800 non-null    int64
dtypes: float64(3), int64(3), object(9)
memory usage: 93.9+ KB
```

In [64]:
```
df.describe()
```

Out[64]:

	TransactionID	TransactionAmount	AccountBalance	RiskScore	CreditRatir
count	800.000000	800.000000	800.000000	800.000000	800.00000(
mean	101.261250	53695.086727	72920.295572	0.472623	583.4850(
std	57.219779	30115.644050	34005.837334	0.232112	156.6614!
min	1.000000	-59669.075480	-30766.906970	-0.380361	305.0000(
25%	54.750000	33334.555715	49949.249088	0.315380	452.7500(
50%	100.000000	52765.641760	72252.815065	0.469889	591.5000(
75%	154.000000	74158.869990	97290.666368	0.623438	718.5000(
max	199.000000	166083.829600	184008.258800	1.257012	849.0000(

In [65]:

```
print("Columns:", df.columns.tolist())
print("\nDtypes:\n", df.dtypes)
print("\nMissing count per column:\n", df.isnull().sum())
```

```
Columns: ['TransactionID', 'CustomerID', 'AccountID', 'AccountType', 'Tran
sactionType', 'Product', 'Firm', 'Region', 'Manager', 'TransactionDate',
'TransactionAmount', 'AccountBalance', 'RiskScore', 'CreditRating', 'Tenur
eMonths']

Dtypes:
 TransactionID          int64
CustomerID            object
AccountID             object
AccountType           object
TransactionType       object
Product               object
Firm                  object
Region                object
Manager               object
TransactionDate       object
TransactionAmount    float64
AccountBalance       float64
RiskScore            float64
CreditRating           int64
TenureMonths           int64
dtype: object

Missing count per column:
 TransactionID        0
CustomerID           0
AccountID            0
AccountType          0
TransactionType      0
Product              0
Firm                 0
Region               0
Manager              0
TransactionDate      0
TransactionAmount    0
AccountBalance       0
RiskScore            0
CreditRating         0
TenureMonths         0
dtype: int64
```

In [66]:

```
print("Rows,Cols:", df.shape)
display(df.dtypes)
display(df.isnull().sum())
```

```
Rows,Cols: (800, 15)
```

```
TransactionID          int64
CustomerID            object
AccountID             object
AccountType           object
TransactionType       object
Product               object
Firm                  object
Region                object
Manager               object
TransactionDate       object
TransactionAmount    float64
AccountBalance       float64
RiskScore            float64
CreditRating           int64
TenureMonths           int64
dtype: object
TransactionID        0
CustomerID           0
AccountID            0
AccountType          0
TransactionType      0
Product              0
Firm                 0
Region               0
Manager              0
TransactionDate      0
TransactionAmount    0
AccountBalance       0
RiskScore            0
CreditRating         0
TenureMonths         0
dtype: int64
```

In [67]:
```python
# Check data types of financial fields
print(df[['TransactionAmount','AccountBalance']].dtypes)

# Check for non-numeric characters (special characters)
print("Special chars in TransactionAmount:",
      df['TransactionAmount'].astype(str).str.contains(r'[^0-9\.\-]').sum

print("Special chars in AccountBalance:",
      df['AccountBalance'].astype(str).str.contains(r'[^0-9\.\-]').sum())
```

```
TransactionAmount    float64
AccountBalance       float64
dtype: object
Special chars in TransactionAmount: 0
Special chars in AccountBalance: 0
```

In [68]:
```python
df['TransactionAmount'] = pd.to_numeric(df['TransactionAmount'], errors='
df['AccountBalance'] = pd.to_numeric(df['AccountBalance'], errors='coerce
print(f"Null values: {df['TransactionAmount'].isna().sum()}")  # ✓ 0
```

```
Null values: 0
```

2.Convert currency amounts into numerical format.

In [69]:
```python
def clean_currency(col):
    return (df[col].astype(str)
              .str.replace(r'[\$,]', '', regex=True)
```

```
                .str.replace(r'\(', '-', regex=True)
                .str.replace(r'\)', '', regex=True)
                .replace({'nan': np.nan, '': np.nan})
                .astype(float)
            )


for col in ['TransactionAmount', 'AccountBalance']:
    if col in df.columns:
        df[col] = clean_currency(col)


df[['TransactionAmount','AccountBalance']].describe()
```

Out[69]:

	TransactionAmount	AccountBalance
count	800.000000	800.000000
mean	53695.086727	72920.295572
std	30115.644050	34005.837334
min	-59669.075480	-30766.906970
25%	33334.555715	49949.249088
50%	52765.641760	72252.815065
75%	74158.869990	97290.666368
max	166083.829600	184008.258800

In [70]:
```
# Convert financial fields to numeric (safeguard cleaning)
df['TransactionAmount'] = pd.to_numeric(df['TransactionAmount'], errors='
df['AccountBalance'] = pd.to_numeric(df['AccountBalance'], errors='coerce

# Confirm numeric conversion
print(df[['TransactionAmount','AccountBalance']].dtypes)
```

```
TransactionAmount    float64
AccountBalance       float64
dtype: object
```

In [71]:
```
df.columns = [c.strip().replace(" ", "_").replace(".", "").lower() for c
df.rename(columns={
    'transactiondate':'transaction_date',
    'transactionamount':'transaction_amount',
    'accountbalance':'account_balance',
}, inplace=True)
df.head(5)
```

Out[71]:

	transactionid	customerid	accountid	accounttype	transactiontype	product	fi
0	118	CUST3810	ACC49774	Savings	Deposit	Credit Card	F
1	102	CUST3109	ACC96277	Savings	Deposit	Mutual Fund	F
2	151	CUST2626	ACC21429	Credit	Payment	Personal Loan	F
3	57	CUST3725	ACC48501	Loan	Withdrawal	Credit Card	F
4	113	CUST4258	ACC11285	Loan	Transfer	Home Loan	F

In [72]:

```
df.columns = [str(c).strip().lower().replace(' ', '_').replace('-', '_')
df.columns.tolist()
```

Out[72]:

```
['transactionid',
 'customerid',
 'accountid',
 'accounttype',
 'transactiontype',
 'product',
 'firm',
 'region',
 'manager',
 'transaction_date',
 'transaction_amount',
 'account_balance',
 'riskscore',
 'creditrating',
 'tenuremonths']
```

3.Validate and format date columns.

In [73]:

```
df['transaction_date'] = pd.to_datetime(df['transaction_date'], errors='c
print("Null dates:", df['transaction_date'].isnull().sum())
df['year'] = df['transaction_date'].dt.year
df['month'] = df['transaction_date'].dt.month
df['month_year'] = df['transaction_date'].dt.to_period('M')
df[['transaction_date','year','month']].head(3)
```

```
Null dates: 478
```

Out[73]:

	transaction_date	year	month
0	2024-08-01	2024.0	8.0
1	NaT	NaN	NaN
2	NaT	NaN	NaN

4.Ensure account types and transaction categories are standardized.

In [74]:

```
df['account_type'] = df['accounttype'] if 'accounttype' in df.columns els

df['account_type'] = df['account_type'].astype(str).str.strip().str.lower
```

```
    'savings account':'savings', 'saving':'savings', 'current acct':'curr
})

df['transaction_type'] = df['transactiontype'] if 'transactiontype' in df
df['transaction_type'] = df['transaction_type'].astype(str).str.strip().s
    'deposit':'credit', 'withdrawal':'debit', 'payment':'debit'
})


print("Account types:", df['account_type'].unique())
print("Transaction types:", df['transaction_type'].unique())
```

```
Account types: ['savings' 'credit' 'loan' 'current']
Transaction types: ['credit' 'debit' 'transfer']
```

Task-2 : Descriptive Transactional Analysis

1.Calculate monthly and yearly summaries of total credits, debits, and net transaction volume.

In [75]:
```
df['Month'] = df['transaction_date'].dt.month
df['Year'] = df['transaction_date'].dt.year


monthly_summary = df.groupby(['Year','Month']).agg(
    Total_Transactions=('transaction_amount','count'),
    Total_Amount=('transaction_amount','sum')
).reset_index()

print(monthly_summary)

yearly_summary = df.groupby('Year').agg(
    Total_Transactions=('transaction_amount', 'count'),
    Total_Amount=('transaction_amount', 'sum')
).reset_index()

print(yearly_summary)
```

```
      Year  Month  Total_Transactions  Total_Amount
0   2023.0    1.0                  18  1.229380e+06
1   2023.0    2.0                   7  3.234586e+05
2   2023.0    3.0                  14  5.860676e+05
3   2023.0    4.0                  17  8.426375e+05
4   2023.0    5.0                  17  1.098851e+06
5   2023.0    6.0                  41  1.894902e+06
6   2023.0    7.0                  11  6.506957e+05
7   2023.0    8.0                  22  1.313271e+06
8   2023.0    9.0                   6  2.586965e+05
9   2023.0   10.0                  24  1.274589e+06
10  2023.0   11.0                  17  1.076542e+06
11  2023.0   12.0                  15  8.896748e+05
12  2024.0    1.0                  16  9.089603e+05
13  2024.0    2.0                  14  5.974399e+05
14  2024.0    4.0                  15  6.770419e+05
15  2024.0    5.0                  13  7.016619e+05
16  2024.0    6.0                   9  5.821224e+05
17  2024.0    7.0                   2  1.759714e+05
18  2024.0    8.0                  18  1.030467e+06
19  2024.0    9.0                   6  2.296174e+05
20  2024.0   11.0                   7  1.864861e+05
21  2024.0   12.0                  13  4.085882e+05
     Year  Total_Transactions  Total_Amount
0  2023.0                 209  1.143877e+07
1  2024.0                 113  5.498357e+06
```

3.Identify top and bottom performing accounts based on net inflow.

In [76]:
```python
df['Inflow'] = df.apply(lambda x: x['transaction_amount'] if x['transacti
df['Outflow'] = df.apply(lambda x: x['transaction_amount'] if x['transact

account_flow = df.groupby('accountid').agg(
    Total_Inflow=('Inflow', 'sum'),
    Total_Outflow=('Outflow', 'sum')
).reset_index()


account_flow['Net_Inflow'] = account_flow['Total_Inflow'] - account_flow[


top_accounts = account_flow.sort_values(by='Net_Inflow', ascending=False)
bottom_accounts = account_flow.sort_values(by='Net_Inflow', ascending=Tru

print("Top Performing Accounts (Net Inflow):")
print(top_accounts)

print("\nBottom Performing Accounts (Net Inflow):")
print(bottom_accounts)
```

```
Top Performing Accounts (Net Inflow):
    accountid  Total_Inflow  Total_Outflow  Net_Inflow
0    ACC10117             0              0           0
97   ACC49422             0              0           0
123  ACC64022             0              0           0
124  ACC64393             0              0           0
125  ACC64785             0              0           0
126  ACC65144             0              0           0
127  ACC65545             0              0           0
128  ACC66086             0              0           0
129  ACC66190             0              0           0
130  ACC67701             0              0           0

Bottom Performing Accounts (Net Inflow):
    accountid  Total_Inflow  Total_Outflow  Net_Inflow
0    ACC10117             0              0           0
122  ACC62809             0              0           0
123  ACC64022             0              0           0
124  ACC64393             0              0           0
125  ACC64785             0              0           0
126  ACC65144             0              0           0
127  ACC65545             0              0           0
128  ACC66086             0              0           0
129  ACC66190             0              0           0
130  ACC67701             0              0           0
```

4.Identify and flag accounts as dormant or inactive if there is a gap of two months or more between consecutive transactions.

In [77]:
```python
df_sorted = df.sort_values(by=['accountid', 'transaction_date'])


df_sorted['Gap_Days'] = df_sorted.groupby('accountid')['transaction_date'


dormant_accounts = df_sorted.groupby('accountid')['Gap_Days'].max().reset
dormant_accounts['Dormant_Flag'] = dormant_accounts['Gap_Days'].apply(lam

print(dormant_accounts.head(20))
```

```
   accountid  Gap_Days Dormant_Flag
0   ACC10117     298.0          Yes
1   ACC10996       NaN           No
2   ACC11062     235.0          Yes
3   ACC11188       NaN           No
4   ACC11285       NaN           No
5   ACC11837       NaN           No
6   ACC12182       NaN           No
7   ACC12334     235.0          Yes
8   ACC13357       NaN           No
9   ACC15228     329.0          Yes
10  ACC15359      27.0           No
11  ACC15671       NaN           No
12  ACC15925       NaN           No
13  ACC16241       NaN           No
14  ACC16664       NaN           No
15  ACC17688       NaN           No
16  ACC18057       NaN           No
17  ACC18140       NaN           No
18  ACC18177       NaN           No
19  ACC19156     246.0          Yes
```

2.Plot trends in total credits vs. debits over time.

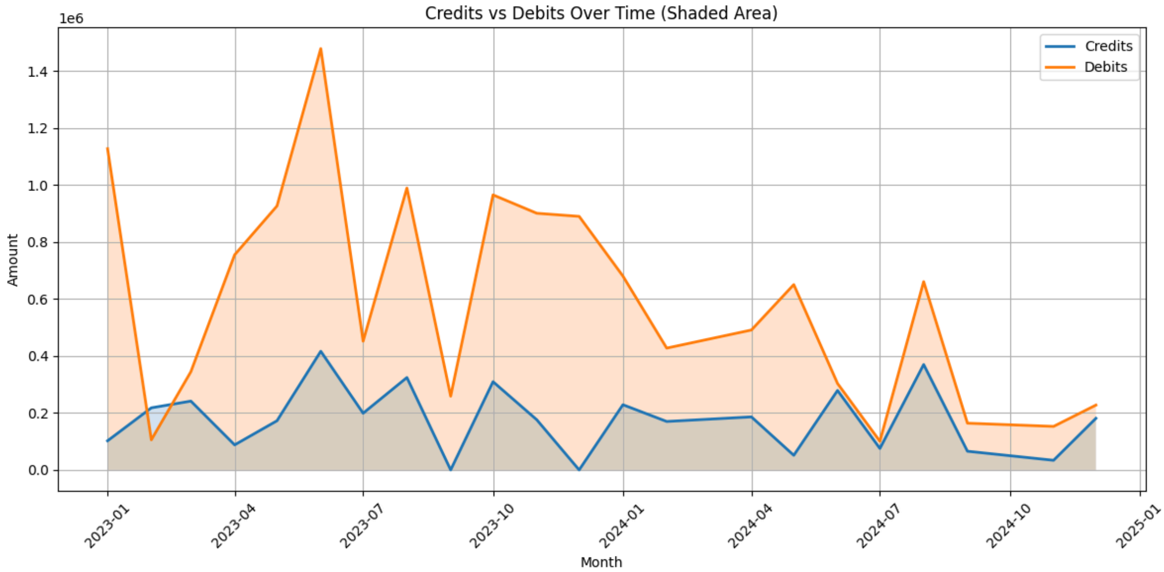
In [78]:
```
# Ensure consistent credit/debit direction
df['direction'] = df['transaction_type'].apply(lambda x: 'credit' if str(
df['month_year'] = df['transaction_date'].dt.to_period('M')
monthly_cd = df.groupby(['month_year', 'direction'])['transaction_amount'
monthly_cd.index = monthly_cd.index.to_timestamp()

# Rename for clarity
monthly_cd.rename(columns={'credit': 'Total Credits', 'debit': 'Total Deb

monthly_cd.head()
```

Out[78]:

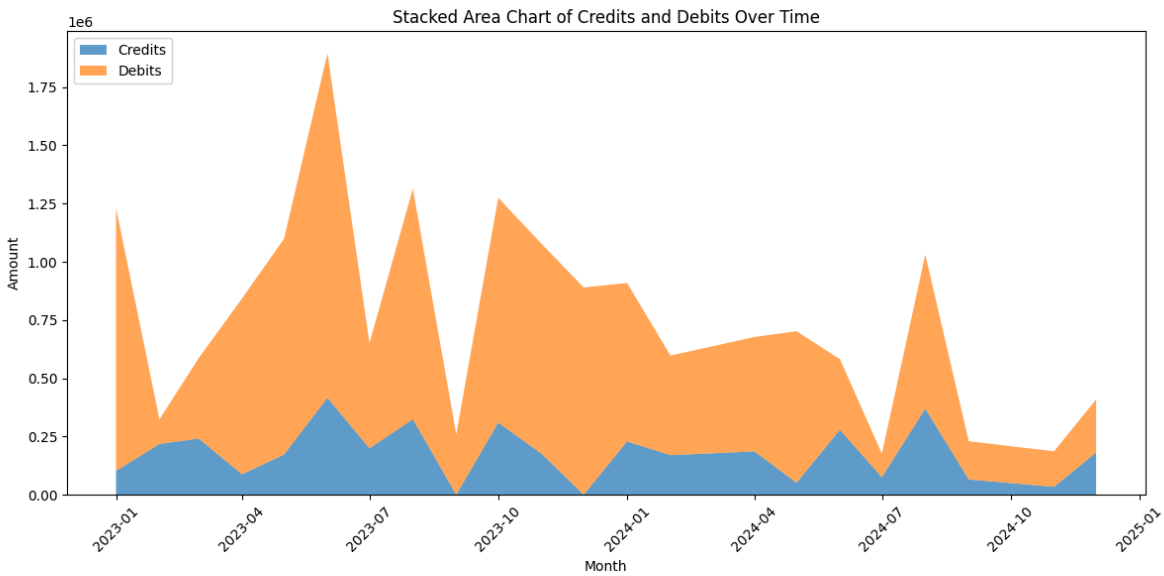
direction	Total Credits	Total Debits
month_year		
2023-01-01	102230.777000	1.127149e+06
2023-02-01	217868.116620	1.055905e+05
2023-03-01	241474.616656	3.445930e+05
2023-04-01	87843.639073	7.547938e+05
2023-05-01	172541.331860	9.263097e+05

In [79]:
```
plt.figure(figsize=(12,6))

plt.plot(monthly_cd.index, monthly_cd['Total Credits'], label='Credits',
plt.plot(monthly_cd.index, monthly_cd['Total Debits'], label='Debits', li

# Shading
plt.fill_between(monthly_cd.index, monthly_cd['Total Credits'], alpha=0.2
plt.fill_between(monthly_cd.index, monthly_cd['Total Debits'], alpha=0.2)

plt.title("Credits vs Debits Over Time (Shaded Area)")
plt.xlabel("Month")
```

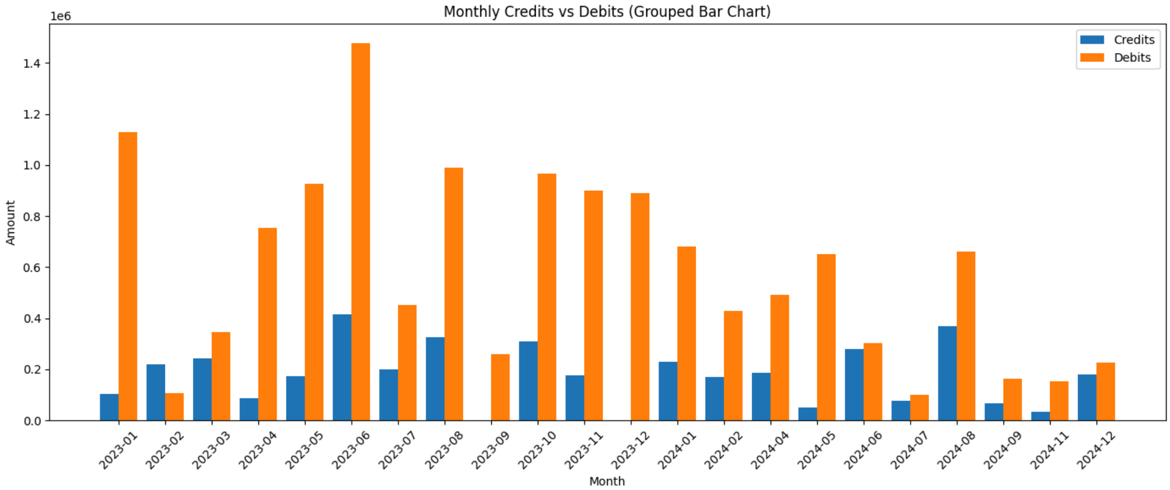
```
plt.ylabel("Amount")
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [80]:
```
plt.figure(figsize=(12,6))

plt.stackplot(
    monthly_cd.index,
    monthly_cd['Total Credits'],
    monthly_cd['Total Debits'],
    labels=['Credits', 'Debits'],
    alpha=0.7
)

plt.title("Stacked Area Chart of Credits and Debits Over Time")
plt.xlabel("Month")
plt.ylabel("Amount")
plt.legend(loc="upper left")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [81]:
```
plt.figure(figsize=(14,6))

bar_width = 0.4
x = range(len(monthly_cd))

plt.bar([i - bar_width/2 for i in x], monthly_cd['Total Credits'],
        width=bar_width, label='Credits')
plt.bar([i + bar_width/2 for i in x], monthly_cd['Total Debits'],
        width=bar_width, label='Debits')

plt.title("Monthly Credits vs Debits (Grouped Bar Chart)")
plt.xlabel("Month")
plt.ylabel("Amount")
plt.xticks(x, monthly_cd.index.strftime("%Y-%m"), rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```



Task3 : Customer Profile Building

1.Group accounts by activity levels: High, Medium, Low based on transaction frequency on your analysis and rubrics. Do not forget to mention the rubric in the headings.

In [82]:
```
## (Rubric: High ≥ 20, Medium = 10–19, Low < 10 Transactions)

activity = df.groupby('accountid').size().reset_index(name='Transaction_C

# Apply rubric
def categorize_activity(count):
    if count >= 20:
        return 'High Activity'
    elif count >= 10:
        return 'Medium Activity'
    else:
        return 'Low Activity'

activity['Rubric_Activity'] = activity['Transaction_Count'].apply(categor

print(activity.head(20))
```

```
   accountid  Transaction_Count Rubric_Activity
0   ACC10117                  4    Low Activity
1   ACC10996                  5    Low Activity
2   ACC11062                  2    Low Activity
3   ACC11188                  4    Low Activity
4   ACC11285                  3    Low Activity
5   ACC11837                  2    Low Activity
6   ACC12182                  4    Low Activity
7   ACC12334                  5    Low Activity
8   ACC13357                  5    Low Activity
9   ACC15228                  6    Low Activity
10  ACC15359                  2    Low Activity
11  ACC15671                  1    Low Activity
12  ACC15925                  4    Low Activity
13  ACC16241                  5    Low Activity
14  ACC16664                  3    Low Activity
15  ACC17688                  2    Low Activity
16  ACC18057                  3    Low Activity
17  ACC18140                  2    Low Activity
18  ACC18177                  1    Low Activity
19  ACC19156                  4    Low Activity
```

2.Segment customers by average balance and transaction volume.

In [83]:
```python
# (Rubric: Balance → High ≥ 100000, Medium = 50000–99999, Low < 50000
#           Volume → High ≥ 20, Medium = 10–19, Low < 10)


avg_balance = df.groupby('customerid')['account_balance'].mean().reset_in


txn_volume = df.groupby('customerid').size().reset_index(name='Txn_Count'


customer_segments = avg_balance.merge(txn_volume, on='customerid')


def balance_segment(x):
    if x >= 100000:
        return 'High Balance'
    elif x >= 50000:
        return 'Medium Balance'
    else:
        return 'Low Balance'


def volume_segment(x):
    if x >= 20:
        return 'High Volume'
    elif x >= 10:
        return 'Medium Volume'
    else:
        return 'Low Volume'


customer_segments['Balance_Segment'] = customer_segments['Avg_Balance'].a
customer_segments['Volume_Segment'] = customer_segments['Txn_Count'].appl
```

```
print(customer_segments.head(20))
```

```
   customerid    Avg_Balance  Txn_Count Balance_Segment Volume_Segment
0    CUST1042   96595.402820          5  Medium Balance     Low Volume
1    CUST1114   72673.007480          3  Medium Balance     Low Volume
2    CUST1121   85215.172188          6  Medium Balance     Low Volume
3    CUST1189   53990.275130          5  Medium Balance     Low Volume
4    CUST1223   61344.838365          2  Medium Balance     Low Volume
5    CUST1376   87696.928343          4  Medium Balance     Low Volume
6    CUST1467   44806.615600          1     Low Balance     Low Volume
7    CUST1497   86983.045765          6  Medium Balance     Low Volume
8    CUST1498   83825.890365          6  Medium Balance     Low Volume
9    CUST1547   42838.552500          1     Low Balance     Low Volume
10   CUST1555  103144.435045          2    High Balance     Low Volume
11   CUST1569   56422.186490          7  Medium Balance     Low Volume
12   CUST1609   77891.239170          1  Medium Balance     Low Volume
13   CUST1644   84221.898085          2  Medium Balance     Low Volume
14   CUST1738   60324.901703          3  Medium Balance     Low Volume
15   CUST1747   58698.842550          2  Medium Balance     Low Volume
16   CUST1749  125940.500663          3    High Balance     Low Volume
17   CUST1768   58128.507212          5  Medium Balance     Low Volume
18   CUST1776   81997.356134          7  Medium Balance     Low Volume
19   CUST1840   83865.492208          4  Medium Balance     Low Volume
```

3.Create profiles for:

○ High-net inflow accounts ○ High-frequency low-balance accounts ○ Accounts with negative or near-zero balances

In [84]:
```
# ---- High-Net Inflow Accounts ----

df['Credit'] = df.apply(lambda x: x['transaction_amount'] if x['transacti
df['Debit'] = df.apply(lambda x: x['transaction_amount'] if x['transactio


inflow = df.groupby('accountid').agg(
    Total_Credit=('Credit', 'sum'),
    Total_Debit=('Debit', 'sum')
).reset_index()


inflow['Net_Inflow'] = inflow['Total_Credit'] - inflow['Total_Debit']


high_net_inflow = inflow.sort_values(by='Net_Inflow', ascending=False).he

print(high_net_inflow)
```

```
    accountid   Total_Credit   Total_Debit     Net_Inflow
145  ACC76549  192020.505780   13505.718630  178514.787150
74   ACC39544  142078.535970       0.000000  142078.535970
123  ACC64022  225525.211690  105021.425860  120503.785830
184  ACC95164  207626.550010   90943.711100  116682.838910
25   ACC21878  209175.590091  122758.498210   86417.091881
150  ACC77638  124503.520620   41636.969790   82866.550830
179  ACC92360  233048.539910  169976.444400   63072.095510
121  ACC62446   49845.569140       0.000000   49845.569140
167  ACC86784   40509.319390       0.000000   40509.319390
17   ACC18140   59369.980450   19907.388160   39462.592290
185  ACC95774  137117.254250   99874.345850   37242.908400
138  ACC71938  162824.737530  127772.363250   35052.374280
106  ACC52650   27316.676020   -7411.898764   34728.574784
26   ACC22036   34620.171763       0.000000   34620.171763
55   ACC31539  101445.901850   67252.327070   34193.574780
86   ACC45951   77196.240100   49624.304090   27571.936010
186  ACC96277  122649.241200   95850.274230   26798.966970
173  ACC88516   56158.330740   30731.935060   25426.395680
11   ACC15671   25166.389040       0.000000   25166.389040
64   ACC34821  154644.985900  129502.571189   25142.414711
```

In [85]:
```
avg_balance_acc[avg_balance_acc['Avg_Balance'] < 50000]
```

Out[85]:

	accountid	Avg_Balance
33	ACC24880	32516.729161
52	ACC30146	49391.925607
54	ACC30852	27044.090537
78	ACC42467	47684.159638
82	ACC43771	45725.221140
91	ACC48303	46465.831480
121	ACC62446	16878.131640
128	ACC66086	46633.161371
134	ACC70460	33835.403142
138	ACC71938	39744.702438
141	ACC74631	44700.284518
151	ACC77773	48124.053400
170	ACC88252	47568.923920
178	ACC92104	47099.688920

In [86]:
```
# ---- High-Frequency Low-Balance Accounts ----

avg_balance_acc = df.groupby('accountid')['account_balance'].mean().reset

txn_count_acc = df.groupby('accountid').size().reset_index(name='Txn_Coun
```

```
hf_lb = avg_balance_acc.merge(txn_count_acc, on='accountid')


high_freq_low_bal = hf_lb[(hf_lb['Txn_Count'] >= 20) & (hf_lb['Avg_Balanc

print(high_freq_low_bal)
```

```
Empty DataFrame
Columns: [accountid, Avg_Balance, Txn_Count]
Index: []
```

In [87]:
```
# ---- Accounts with Negative or Near-Zero Balances ----

negative_bal = df[df['account_balance'] < 0][['accountid', 'account_balan


near_zero_bal = df[(df['account_balance'] >= 0) & (df['account_balance']

print("Negative Balance Accounts:")
print(negative_bal.head(20))

print("\nNear-Zero Balance Accounts:")
print(near_zero_bal.head(20))
```

```
Negative Balance Accounts:
    accountid  account_balance
89   ACC21264     -8247.315181
150  ACC42467    -30766.906970
163  ACC41829     -2531.437176
217  ACC19156     -9649.975980
236  ACC11285    -17751.216810
281  ACC77592    -14999.180830
343  ACC49422     -8103.107327
359  ACC24880     -5199.930807
392  ACC32890      -800.699930
412  ACC71938    -12493.956390
549  ACC49140     -1065.037291
660  ACC45521     -9207.916702
709  ACC49364     -2614.910348
789  ACC72197    -14934.034490

Near-Zero Balance Accounts:
Empty DataFrame
Columns: [accountid, account_balance]
Index: []
```

Task 4 :Financial Risk Identification

1.Track accounts with frequent large withdrawals or overdrafts.

In [88]:
```
# ---- Frequent Large Withdrawals (>= 50,000) ----

# Filter only debit/withdrawal transactions
withdrawals = df[df['transactiontype'].str.lower() == 'debit']

# Filter large withdrawals
large_withdrawals = withdrawals[withdrawals['transaction_amount'] >= 5000

# Count large withdrawals per account
large_wd_count = large_withdrawals.groupby('accountid').size().reset_inde

print("Accounts with Large Withdrawals:")
```

```
print(large_wd_count.head(20))

overdrafts = df[df['account_balance'] < 0][['accountid', 'account_balance
print("Overdraft Accounts:")
print(overdrafts.head(20))
```

```
Accounts with Large Withdrawals:
Empty DataFrame
Columns: [accountid, Large_Withdrawal_Count]
Index: []
Overdraft Accounts:
    accountid  account_balance
89   ACC21264     -8247.315181
150  ACC42467    -30766.906970
163  ACC41829     -2531.437176
217  ACC19156     -9649.975980
236  ACC11285    -17751.216810
281  ACC77592    -14999.180830
343  ACC49422     -8103.107327
359  ACC24880     -5199.930807
392  ACC32890      -800.699930
412  ACC71938    -12493.956390
549  ACC49140     -1065.037291
660  ACC45521     -9207.916702
709  ACC49364     -2614.910348
789  ACC72197    -14934.034490
```

In [89]:

```
overdrafts = df[df['account_balance'] < 0][['accountid', 'account_balance
print("Overdraft Accounts:")
print(overdrafts.head(20))
```

```
Overdraft Accounts:
    accountid  account_balance
89   ACC21264     -8247.315181
150  ACC42467    -30766.906970
163  ACC41829     -2531.437176
217  ACC19156     -9649.975980
236  ACC11285    -17751.216810
281  ACC77592    -14999.180830
343  ACC49422     -8103.107327
359  ACC24880     -5199.930807
392  ACC32890      -800.699930
412  ACC71938    -12493.956390
549  ACC49140     -1065.037291
660  ACC45521     -9207.916702
709  ACC49364     -2614.910348
789  ACC72197    -14934.034490
```

2.Calculate balance volatility using standard deviation or coefficient of variation.

In [90]:

```
# ---- Balance Volatility (Std Dev & Coefficient of Variation) ----

balance_vol = df.groupby('accountid')['account_balance'].agg(['mean', 'st
balance_vol['CV'] = balance_vol['std'] / balance_vol['mean']

print(balance_vol.head(20))
```

```
   accountid           mean           std        CV
0   ACC10117   97828.704775   9308.031969  0.095146
1   ACC10996   56982.152538  18946.737199  0.332503
2   ACC11062   65947.316965  22572.552392  0.342282
3   ACC11188   81169.114065  20160.417506  0.248375
4   ACC11285   62574.613950  70126.826097  1.120691
5   ACC11837   63096.777775  41574.241723  0.658896
6   ACC12182   93952.097922  39413.851888  0.419510
7   ACC12334   58469.937674  43584.668914  0.745420
8   ACC13357   78367.596968  19418.026324  0.247781
9   ACC15228   78477.880060  35711.118042  0.455047
10  ACC15359   66401.687335  15440.020242  0.232525
11  ACC15671  120586.085000           NaN       NaN
12  ACC15925   81257.319228  27201.194577  0.334754
13  ACC16241   55945.625702  26112.139934  0.466741
14  ACC16664  102867.122137  52068.392699  0.506171
15  ACC17688   72567.906705  25212.562288  0.347434
16  ACC18057   64862.261870  27875.043837  0.429758
17  ACC18140  111048.708525  36772.764690  0.331141
18  ACC18177   60440.182510           NaN       NaN
19  ACC19156   64046.130650  61272.537291  0.956694
```

3.Use IQR or z-score methods to detect anomalies.

In [91]:
```
Q1 = df['transaction_amount'].quantile(0.25)
Q3 = df['transaction_amount'].quantile(0.75)
IQR = Q3 - Q1

iqr_anomalies = df[(df['transaction_amount'] < (Q1 - 1.5 * IQR)) |
                   (df['transaction_amount'] > (Q3 + 1.5 * IQR))]

print("IQR-Based Anomalies:")
print(iqr_anomalies.head(20))
```

```
IQR-Based Anomalies:
     transactionid customerid accountid accounttype transactiontype  \
6               14   CUST5558  ACC82947      Credit         Payment
48               6   CUST2427  ACC80131     Current         Deposit
200            169   CUST2188  ACC40939        Loan         Payment
312            167   CUST9843  ACC21264     Current        Transfer
589            190   CUST1738  ACC90887     Savings      Withdrawal
622             12   CUST1121  ACC43309      Credit        Transfer
710            117   CUST3041  ACC95164      Credit         Deposit

             product    firm   region    manager transaction_date  ...  \
6        Credit Card  Firm A     East  Manager 2              NaT  ...
48       Mutual Fund  Firm C    South  Manager 1              NaT  ...
200      Mutual Fund  Firm C  Central  Manager 2       2024-12-03  ...
312        Home Loan  Firm C     East  Manager 2              NaT  ...
589  Savings Account  Firm C     East  Manager 2              NaT  ...
622  Savings Account  Firm B    South  Manager 1              NaT  ...
710      Credit Card  Firm A     West  Manager 3       2024-06-04  ...

     month_year account_type transaction_type Month    Year  Inflow  \
6           NaT       credit            debit   NaN     NaN       0
48          NaT      current           credit   NaN     NaN       0
200     2024-12         loan            debit  12.0  2024.0       0
312         NaT      current         transfer   NaN     NaN       0
589         NaT      savings            debit   NaN     NaN       0
622         NaT       credit         transfer   NaN     NaN       0
710     2024-06       credit           credit   6.0  2024.0       0

     Outflow direction      Credit         Debit
6          0     debit       0.000  -43698.75917
48         0    credit  141600.740       0.00000
200        0     debit       0.000  -29124.62485
312        0     debit       0.000  149404.33020
589        0     debit       0.000  -59669.07548
622        0     debit       0.000  166083.82960
710        0    credit  142081.629       0.00000

[7 rows x 27 columns]
```

In [92]:
```python
df['zscore'] = (df['transaction_amount'] - df['transaction_amount'].mean(

zscore_anomalies = df[(df['zscore'] > 3) | (df['zscore'] < -3)]

print("Z-Score Anomalies:")
print(zscore_anomalies.head(20))
```

```
Z-Score Anomalies:
     transactionid customerid accountid accounttype transactiontype  \
6               14   CUST5558  ACC82947      Credit         Payment
312            167   CUST9843  ACC21264     Current        Transfer
589            190   CUST1738  ACC90887     Savings      Withdrawal
622             12   CUST1121  ACC43309      Credit        Transfer

             product    firm region    manager transaction_date  ...  \
6        Credit Card  Firm A   East  Manager 2              NaT  ...
312        Home Loan  Firm C   East  Manager 2              NaT  ...
589  Savings Account  Firm C   East  Manager 2              NaT  ...
622  Savings Account  Firm B  South  Manager 1              NaT  ...

     account_type  transaction_type  Month  Year  Inflow  Outflow  directi
on  \
6          credit             debit    NaN   NaN       0        0      deb
it
312       current          transfer    NaN   NaN       0        0      deb
it
589       savings             debit    NaN   NaN       0        0      deb
it
622        credit          transfer    NaN   NaN       0        0      deb
it

    Credit         Debit    zscore
6      0.0  -43698.75917 -3.233995
312    0.0  149404.33020  3.178057
589    0.0  -59669.07548 -3.764295
622    0.0  166083.82960  3.731906

[4 rows x 28 columns]
```

4.Highlight customers with irregular or suspicious transaction behavior.

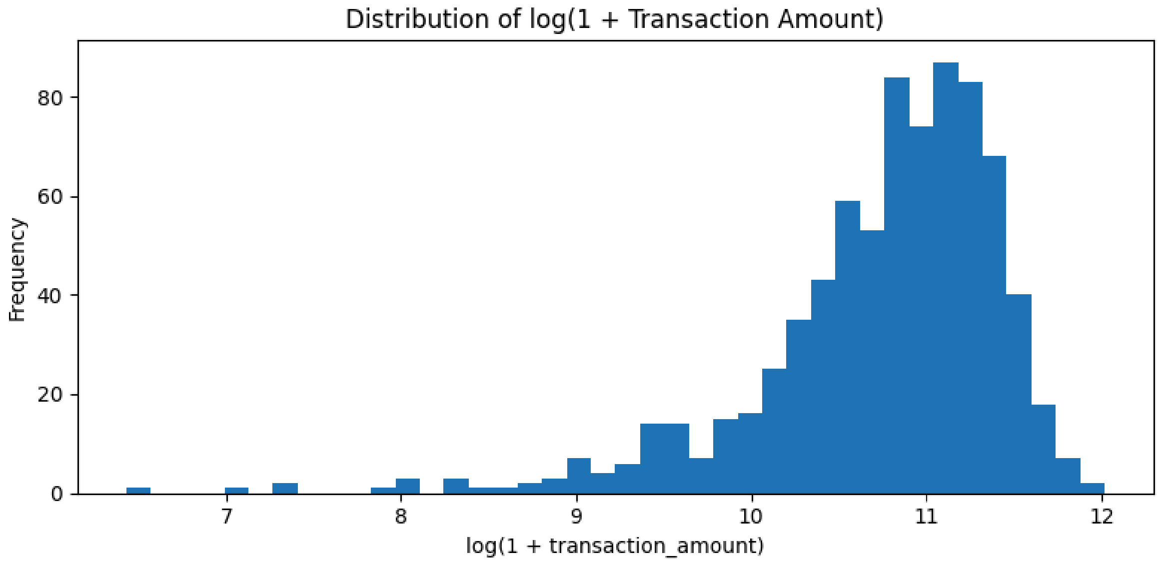
In [93]:
```
risk_large_wd = set(large_wd_count['accountid'])


risk_overdraft = set(overdrafts['accountid'])


high_vol_accounts = set(balance_vol[balance_vol['CV'] > 1]['accountid'])


risk_anomaly_iqr = set(iqr_anomalies['accountid'])
risk_anomaly_z = set(zscore_anomalies['accountid'])


risky_accounts = risk_large_wd.union(risk_overdraft, high_vol_accounts,
                                     risk_anomaly_iqr, risk_anomaly_z)

risky_accounts_df = pd.DataFrame({'accountid': list(risky_accounts)})

print("Final Risky Accounts:")
print(risky_accounts_df.head(20))
```

```
Final Risky Accounts:
   accountid
0   ACC43309
1   ACC55331
2   ACC49422
3   ACC49364
4   ACC24880
5   ACC19156
6   ACC95164
7   ACC72197
8   ACC29646
9   ACC82947
10  ACC77592
11  ACC11285
12  ACC21264
13  ACC32890
14  ACC42467
15  ACC71938
16  ACC49140
17  ACC70460
18  ACC74631
19  ACC90887
```

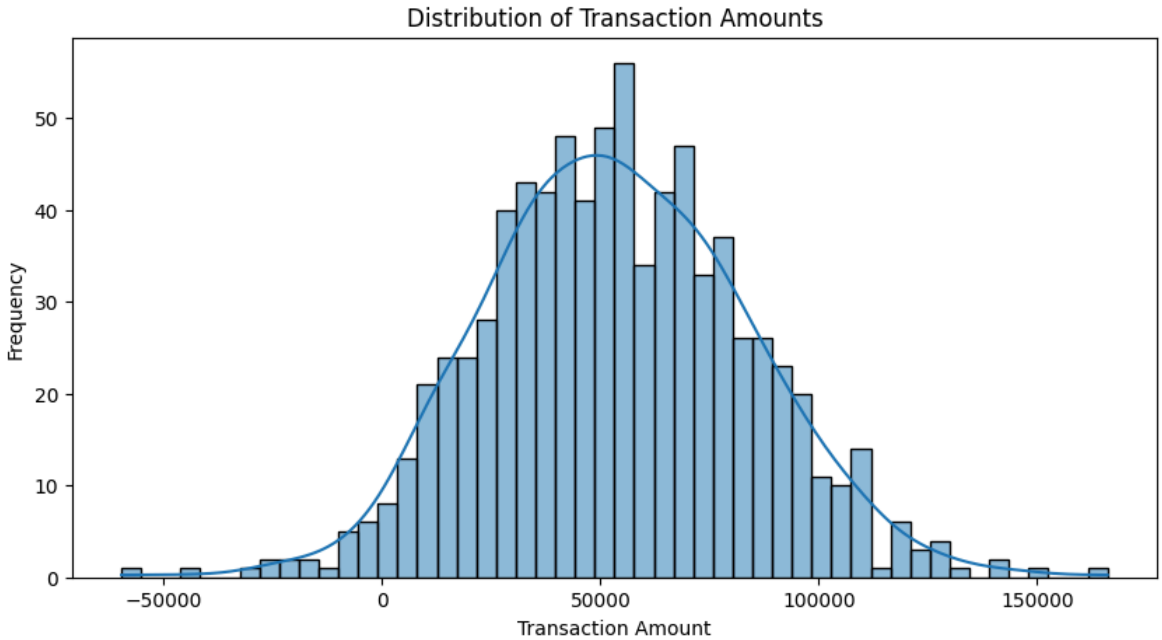
Task 5:Visualisation

Conduct extensive exploratory data analysis with attractive visualizations for your findings

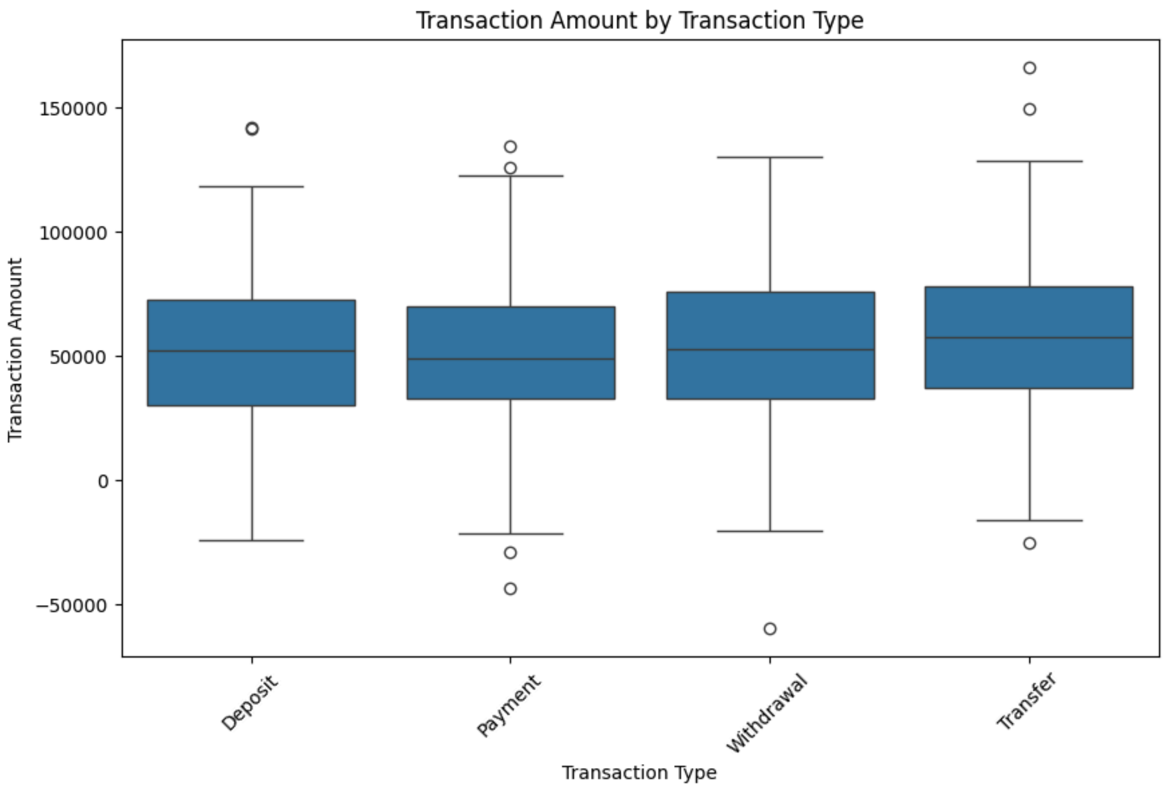
In [94]:
```
# Distribution of Transaction Amounts (log scale)
import numpy as np
import matplotlib.pyplot as plt

pos_amounts = df['transaction_amount'][df['transaction_amount']>0]

plt.figure(figsize=(8,4))
plt.hist(np.log1p(pos_amounts), bins=40)
plt.title("Distribution of log(1 + Transaction Amount)")
plt.xlabel("log(1 + transaction_amount)")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```



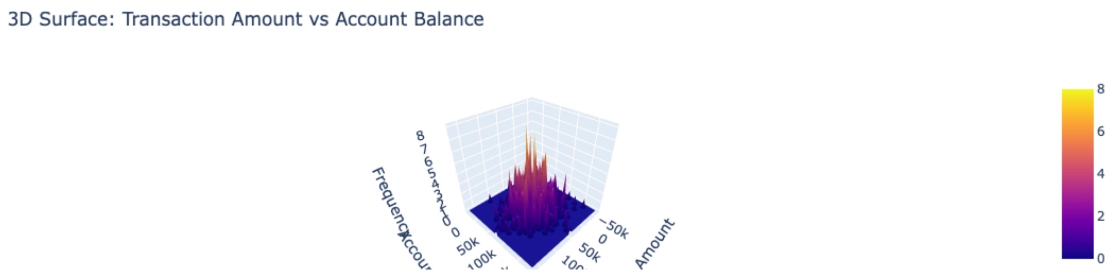
In [95]:
```
plt.figure(figsize=(10,5))
sns.histplot(df['transaction_amount'], bins=50, kde=True)
plt.title("Distribution of Transaction Amounts")
plt.xlabel("Transaction Amount")
plt.ylabel("Frequency")
plt.show()
```



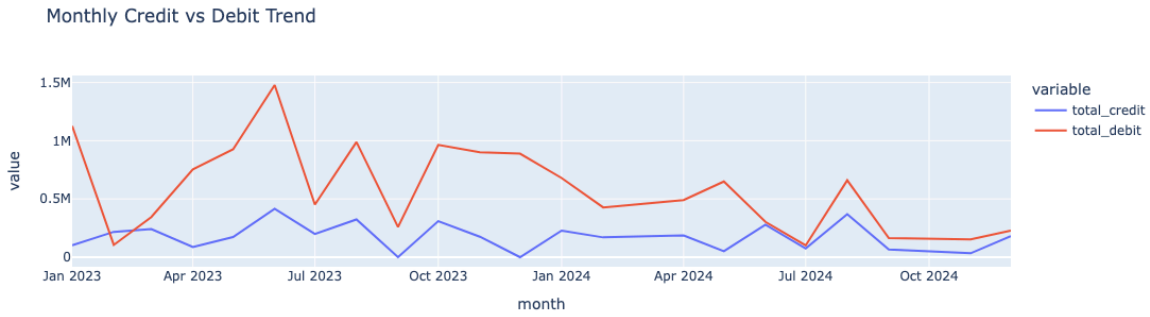
In [96]:
```
plt.figure(figsize=(10,6))
sns.boxplot(data=df, x='transactiontype', y='transaction_amount')
plt.title("Transaction Amount by Transaction Type")
plt.xlabel("Transaction Type")
plt.ylabel("Transaction Amount")
plt.xticks(rotation=45)
plt.show()
```



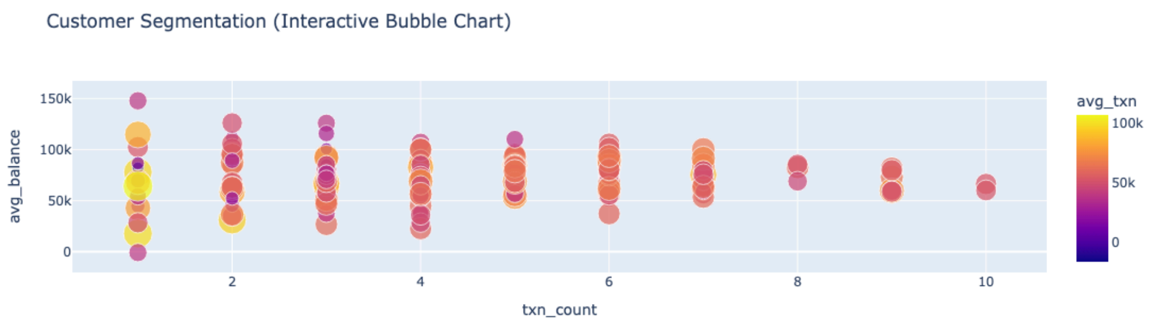
In [97]:
```
import plotly.graph_objects as go
import numpy as np

z = np.histogram2d(df['transaction_amount'],
                   df['account_balance'],
                   bins=40)[0]

x = np.linspace(df['transaction_amount'].min(), df['transaction_amount'].
y = np.linspace(df['account_balance'].min(), df['account_balance'].max(),

fig = go.Figure(data=[go.Surface(z=z, x=x, y=y)])
fig.update_layout(title="3D Surface: Transaction Amount vs Account Balanc
                  scene=dict(
                      xaxis_title='Transaction Amount',
                      yaxis_title='Account Balance',
                      zaxis_title='Frequency'))
fig.show()
```



In [101…
```
import plotly.express as px

monthly = (
    df.groupby(df['transaction_date'].dt.to_period('M'))
      .agg(
          total_credit=('Credit', 'sum'),
          total_debit=('Debit', 'sum')
      )
      .reset_index()
)

monthly['month'] = monthly['transaction_date'].dt.to_timestamp()

fig = px.line(
    monthly,
    x='month',
    y=['total_credit', 'total_debit'],
    title="Monthly Credit vs Debit Trend"
)

fig.update_layout(hovermode='x unified')
fig.show()
```

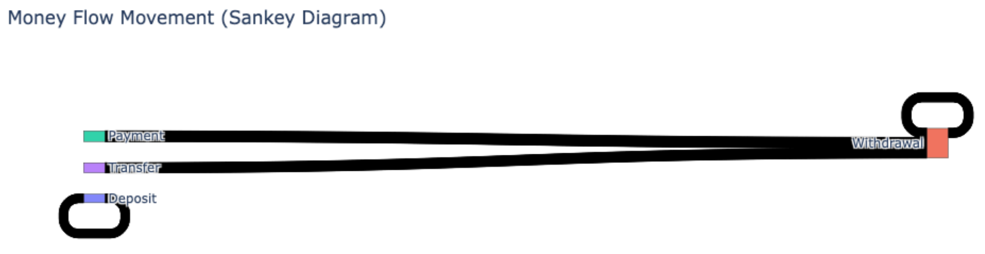


In [99]:
```python
# Customer segmentation preprocessing
cust = df.groupby('customerid').agg(
    avg_balance=('account_balance','mean'),
    txn_count=('transactionid','count'),
    avg_txn=('transaction_amount','mean')
).reset_index()

# Fix: Ensure bubble sizes are always positive
cust['bubble_size'] = cust['avg_txn'].abs()

import plotly.express as px

fig = px.scatter(
    cust,
    x='txn_count',
    y='avg_balance',
    size='bubble_size',
    color='avg_txn',
    hover_name='customerid',
    title="Customer Segmentation (Interactive Bubble Chart)",
)

fig.show()
```



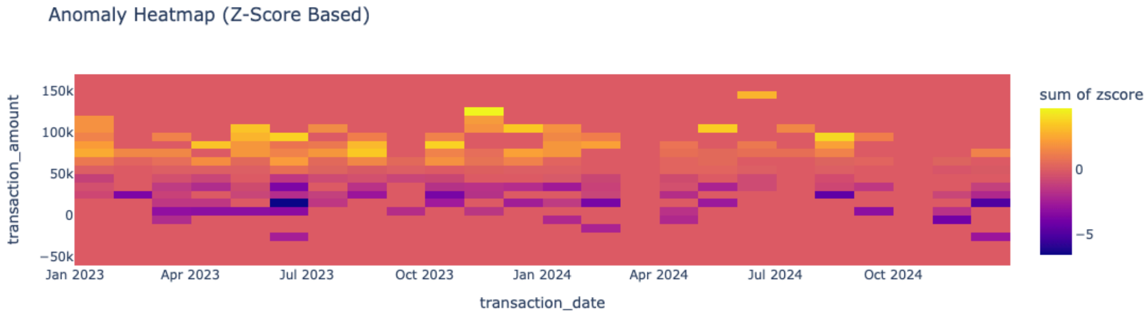
In [104…
```python
import plotly.graph_objects as go

sources = ['Deposit', 'Withdrawal', 'Payment', 'Transfer']
targets = ['Credit', 'Debit']

source_ids = df['transactiontype'].astype('category').cat.codes
target_ids = df.apply(lambda x: 0 if x['Credit']>0 else 1, axis=1)

fig = go.Figure(data=[go.Sankey(
    node=dict(label=sources+targets),
    link=dict(
        source=source_ids,
```

```
        target=target_ids,
        value=df['transaction_amount']
    )
)])
fig.update_layout(title_text="Money Flow Movement (Sankey Diagram)")
fig.show()
```



In [105…

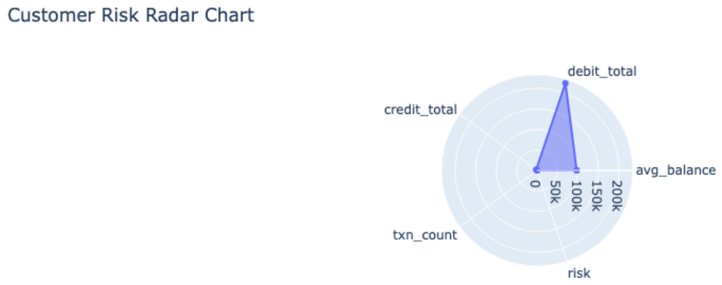
```
df['zscore'] = (df['transaction_amount'] - df['transaction_amount'].mean(

fig = px.density_heatmap(df,
                         x='transaction_date',
                         y='transaction_amount',
                         z='zscore',
                         title="Anomaly Heatmap (Z-Score Based)",
                         nbinsx=40,
                         nbinsy=40)

fig.show()
```



In [107…

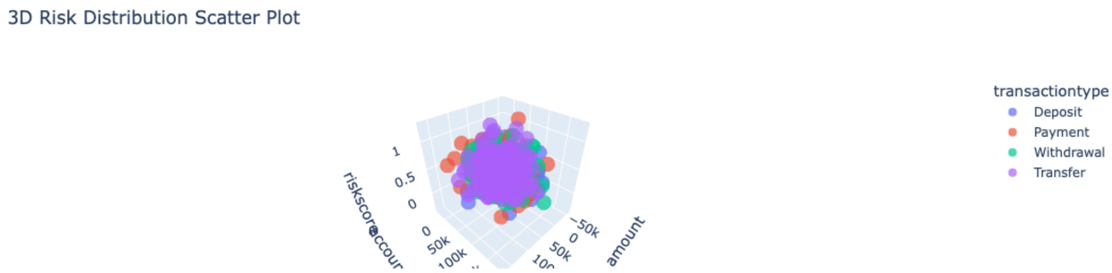
```
risk = df.groupby('customerid').agg(
    avg_balance=('account_balance','mean'),
    debit_total=('Debit','sum'),
    credit_total=('Credit','sum'),
    txn_count=('transactionid','count'),
    risk=('riskscore','mean')
).reset_index().head(1)  # Example: first customer

categories = ['avg_balance','debit_total','credit_total','txn_count','ris
values = risk[categories].values.flatten().tolist()
values += values[:1]

fig = go.Figure()
fig.add_trace(go.Scatterpolar(
    r=values,
    theta=categories + [categories[0]],
    fill='toself'
```

```
))
fig.update_layout(title="Customer Risk Radar Chart")
fig.show()
```



In [108…

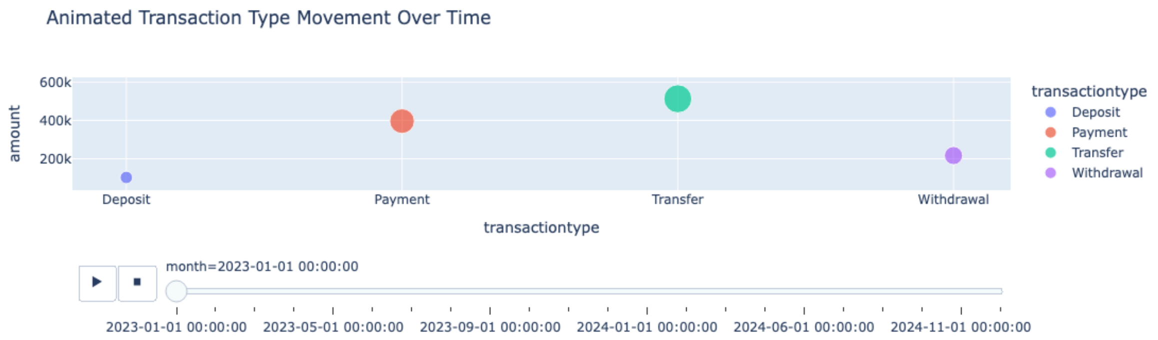
```
fig = px.scatter_3d(df,
                    x='transaction_amount',
                    y='account_balance',
                    z='riskscore',
                    color='transactiontype',
                    title="3D Risk Distribution Scatter Plot",
                    opacity=0.7)
fig.show()
```



In [109…

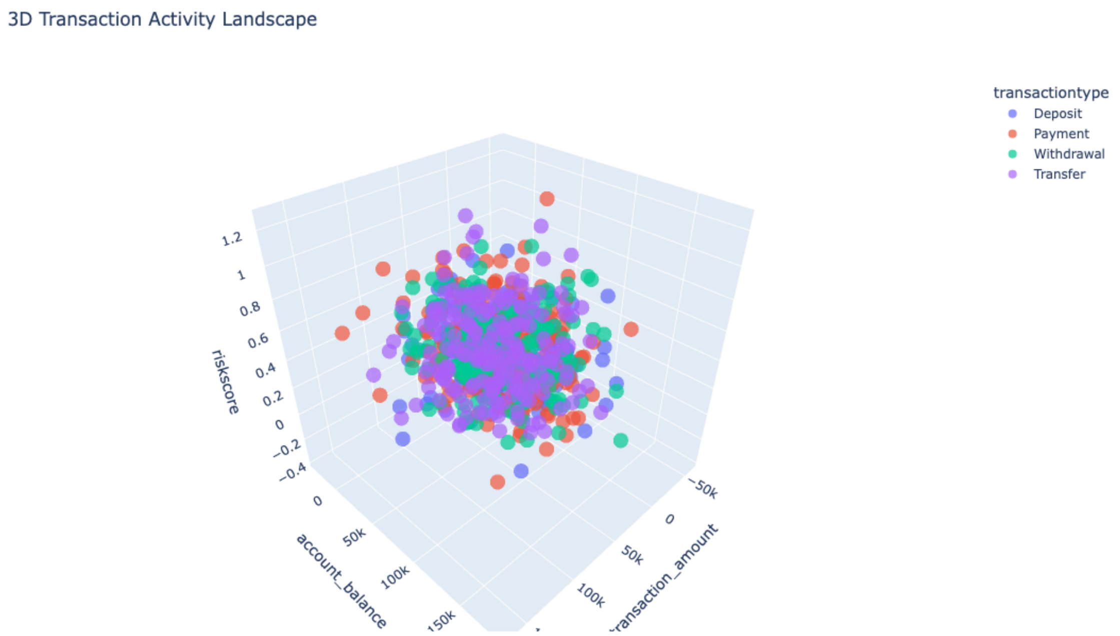
```
import plotly.express as px

# Group monthly sums
grouped = df.groupby([
    df['transaction_date'].dt.to_period('M'),
    'transactiontype'
]).agg(
    amount=('transaction_amount', 'sum')
).reset_index()

# Convert period to timestamp
grouped['month'] = grouped['transaction_date'].dt.to_timestamp()

# FIX: Bubble size must be positive
grouped['bubble_size'] = grouped['amount'].abs()

# Create animation
fig = px.scatter(
    grouped,
    x='transactiontype',
    y='amount',
    size='bubble_size',       # FIXED
    animation_frame='month',
```

```
    color='transactiontype',
    title="Animated Transaction Type Movement Over Time",
    hover_data=['amount', 'bubble_size']
)

fig.show()
```



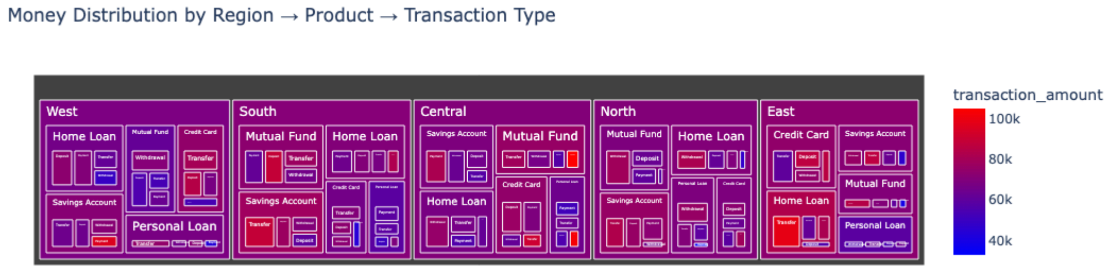
In [110…

```
import plotly.express as px

df['month'] = df['transaction_date'].dt.to_period('M').dt.to_timestamp()

fig = px.scatter_3d(
    df,
    x="transaction_amount",
    y="account_balance",
    z="riskscore",
    color="transactiontype",
    title="3D Transaction Activity Landscape",
    opacity=0.7,
    height=700
)

fig.show()
```



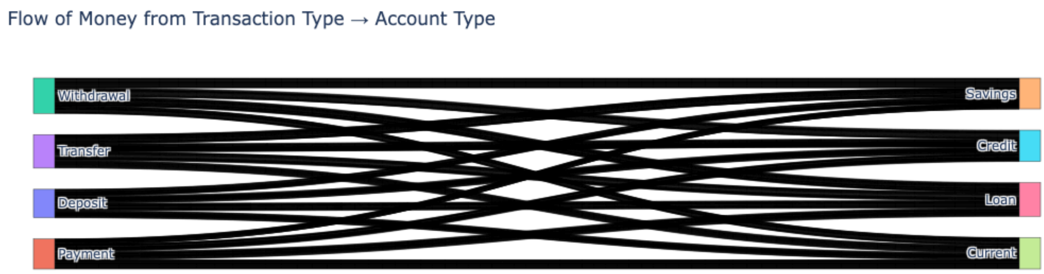
In [111…

```
fig = px.treemap(
    df,
    path=['region','product','transactiontype'],
    values='transaction_amount',
    color='transaction_amount',
    color_continuous_scale='bluered',
    title="Money Distribution by Region → Product → Transaction Type"
)

fig.show()
```



In [43]:

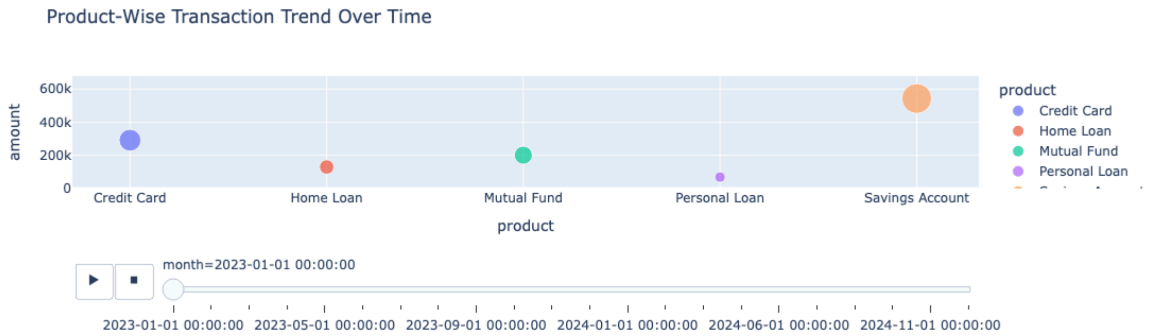
```
import plotly.graph_objects as go

sources = df['transactiontype']
targets = df['accounttype']
amounts = df['transaction_amount'].abs()

fig = go.Figure(data=[go.Sankey(
    node=dict(
        label=list(df['transactiontype'].unique()) + list(df['accounttype
        pad=20,
        thickness=20
    ),
    link=dict(
        source=df['transactiontype'].astype('category').cat.codes,
        target=df['accounttype'].astype('category').cat.codes + df['trans
        value=amounts
    ))])

fig.update_layout(title="Flow of Money from Transaction Type → Account Ty
fig.show()
```



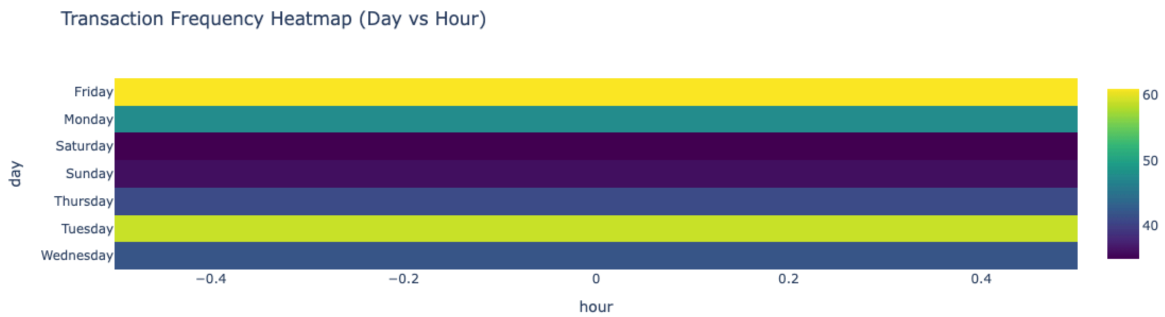
In [44]:

```
grouped = df.groupby([df['transaction_date'].dt.to_period('M'),'product']
    amount=('transaction_amount','sum')
).reset_index()
```

```
grouped['month'] = grouped['transaction_date'].dt.to_timestamp()
grouped['bubble'] = grouped['amount'].abs()

fig = px.scatter(
    grouped,
    x='product',
    y='amount',
    size='bubble',
    color='product',
    animation_frame='month',
    title="Product-Wise Transaction Trend Over Time"
)
fig.show()
```



In [45]:
```
df['hour'] = df['transaction_date'].dt.hour
df['day'] = df['transaction_date'].dt.day_name()

pivot = df.pivot_table(index='day', columns='hour', values='transactionid

fig = px.imshow(
    pivot,
    color_continuous_scale='Viridis',
    title="Transaction Frequency Heatmap (Day vs Hour)"
)
fig.show()
```



In [46]:
```
cust = df.groupby('customerid').agg(
    avg_txn=('transaction_amount','mean'),
    avg_balance=('account_balance','mean'),
    risk=('riskscore','mean')
).reset_index()

radar_df = cust.melt(id_vars='customerid', var_name='Metric', value_name=

fig = px.line_polar(
```

```
    radar_df,
    r='Value',
    theta='Metric',
    color='customerid',
    line_close=True,
    title="Customer-wise Radar Chart (Interactive)",
)

fig.show()
```

Task 6: Hypothesis Testing

1.Test whether high-volume transaction accounts have statistically higher average balances than low-volume accounts.

In [47]:
```
# Group by account
acc = df.groupby('accountid').agg(
    avg_balance = ('account_balance','mean'),
    txn_count   = ('transactionid','count')
).reset_index()

# Create high vs low groups using quantiles
high_volume = acc[acc['txn_count'] >= acc['txn_count'].quantile(0.70)]
low_volume  = acc[acc['txn_count'] <= acc['txn_count'].quantile(0.30)]

len(high_volume), len(low_volume)
```

Out[47]: (70, 72)

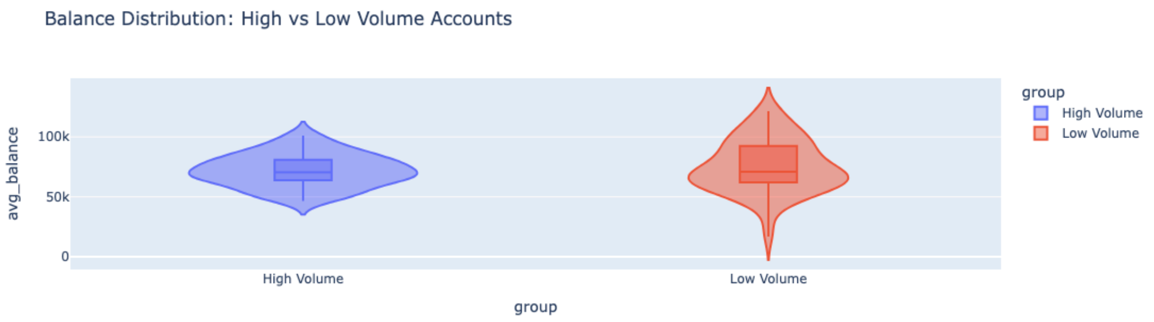
In [48]:
```
from scipy.stats import ttest_ind

stat, p = ttest_ind(
    high_volume['avg_balance'],
    low_volume['avg_balance'],
    equal_var=False,   # Welch's t-test
    alternative='greater'  # high_volume > low_volume
)

stat, p
```

Out[48]: (np.float64(-1.1286641394111137), np.float64(0.869351579213825))

In [49]:
```
alpha = 0.05

if p < alpha:
    print("Reject H0 → High-volume accounts have significantly higher ave
else:
    print("Fail to Reject H0 → No evidence that high-volume accounts have
```

```
Fail to Reject H0 → No evidence that high-volume accounts have higher bala
nces.
```

In [50]:
```
import plotly.express as px

temp = pd.concat([
    high_volume.assign(group='High Volume'),
    low_volume.assign(group='Low Volume')
])

fig = px.violin(
    temp,
    x='group',
    y='avg_balance',
    box=True,
    color='group',
    title="Balance Distribution: High vs Low Volume Accounts"
)
fig.show()
```



2.Conduct hypothesis testing based on segmentation.

In [51]:
```
# Create segmentation using quantiles
acc['segment'] = pd.qcut(
    acc['txn_count'],
    q=3,
    labels=['Low Activity', 'Medium Activity', 'High Activity']
)
```

In [52]:
```
from scipy.stats import f_oneway

low  = acc[acc['segment']=='Low Activity']['avg_balance']
med  = acc[acc['segment']=='Medium Activity']['avg_balance']
high = acc[acc['segment']=='High Activity']['avg_balance']

stat, p = f_oneway(low, med, high)
stat, p
```

Out[52]:
```
(np.float64(0.5625886339724638), np.float64(0.5706784512284537))
```

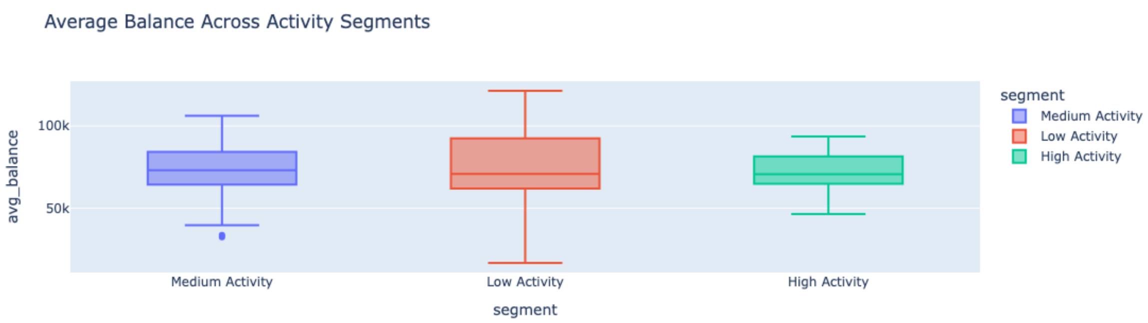
In [53]:
```
alpha = 0.05

if p < alpha:
    print("Reject H0 → At least one segment has a significantly different
else:
    print("Fail to Reject H0 → No significant difference between segments
```

```
Fail to Reject H0 → No significant difference between segments.
```

In [54]:
```
low  = acc[acc['segment']=='Low Activity']['avg_balance']
med  = acc[acc['segment']=='Medium Activity']['avg_balance']
high = acc[acc['segment']=='High Activity']['avg_balance']

from scipy.stats import ttest_ind

# Pairwise comparisons
pair1 = ttest_ind(low, med, equal_var=False)
pair2 = ttest_ind(low, high, equal_var=False)
pair3 = ttest_ind(med, high, equal_var=False)

pair1, pair2, pair3

alpha = 0.05 / 3
alpha

results = {
    "Low vs Medium": pair1.pvalue < alpha,
    "Low vs High":   pair2.pvalue < alpha,
    "Medium vs High":pair3.pvalue < alpha
}

results
```

Out[54]:
```
{'Low vs Medium': np.False_,
 'Low vs High': np.False_,
 'Medium vs High': np.False_}
```

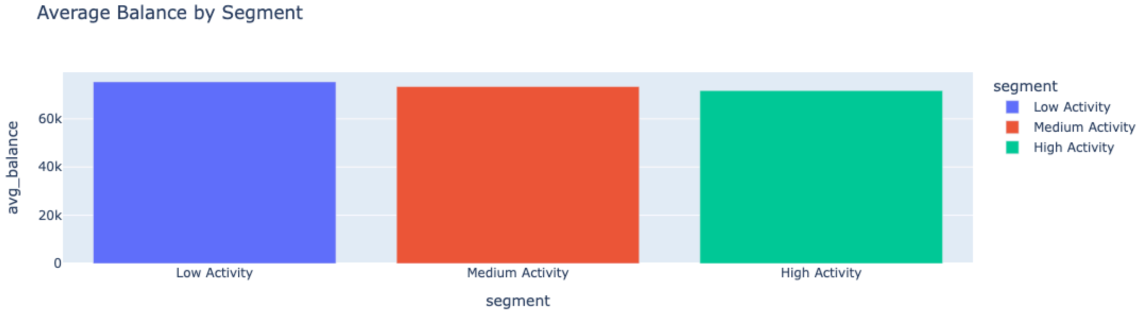
In [55]:
```
fig = px.box(
    acc,
    x='segment',
    y='avg_balance',
    color='segment',
    title="Average Balance Across Activity Segments"
)
fig.show()
```



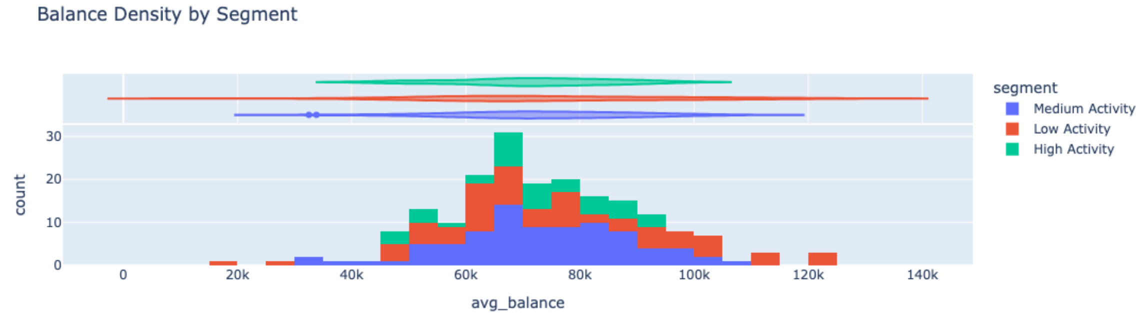
In [56]:
```
seg_mean = acc.groupby('segment')['avg_balance'].mean().reset_index()
```

```
fig = px.bar(
    seg_mean,
    x='segment',
    y='avg_balance',
    color='segment',
    title="Average Balance by Segment"
)
fig.show()
```

```
/var/folders/z1/ts9x85b10ml74v7hz4fzw7v80000gn/T/ipykernel_1343/185204049
0.py:1: FutureWarning:

The default of observed=False is deprecated and will be changed to True in
a future version of pandas. Pass observed=False to retain current behavior
or observed=True to adopt the future default and silence this warning.
```



In [57]:
```
fig = px.histogram(
    acc,
    x='avg_balance',
    color='segment',
    marginal='violin',
    nbins=40,
    title="Balance Density by Segment"
)
fig.show()
```



In [ ]:

In [ ]:

In [ ]: