Financial Analysis Project - Daily Transaction Dataset

Step 1: Import Libraries and Load Data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset

!ls /content
from google.colab import drive
drive.mount('/content/drive')
!ls "/content/drive/MyDrive/Unified Projects"

df = pd.read_csv('/content/drive/MyDrive/Unified Projects/Daily Household Transactions.csv')
# Display the first few rows of the dataset
df.head()
```

```
sample_data
Mounted at /content/drive
 analysis_outputs           'Unified Projects Datasets csv files'
'Daily Household Transactions.csv'
```

	Date	Mode	Category	Subcategory	Note	Amount	Income/Expense	Currency
0	20/09/2018 12:04:08	Cash	Transportation	Train	2 Place 5 to Place 0	30.0	Expense	INR
1	20/09/2018 12:03:15	Cash	Food	snacks	Idli medu Vada mix 2 plates	60.0	Expense	INR
2	19/09/2018	Saving Bank account 1	subscription	Netflix	1 month subscription	199.0	Expense	INR
3	17/09/2018 23:41:17	Saving Bank account 1	subscription	Mobile Service Provider	Data booster pack	19.0	Expense	INR
4	16/09/2018 17:15:08	Cash	Festivals	Ganesh Pujan	Ganesh idol	251.0	Expense	INR

Step 2: Data Cleaning

- Handle missing values.
- Correct data types.
- Remove duplicates.

Exploring the Cleaned DataFrame

To understand the structure and content of the cleaned `df` DataFrame, I will perform the following steps:

1. **Display the first 5 rows** using `df.head()`.
2. **Check the DataFrame's information** (data types, non-null counts) using `df.info()`.
3. **Generate descriptive statistics** for all columns using `df.describe(include='all')`.

```
print("First 5 rows of the cleaned DataFrame:")
display(df.head())
```

```
First 5 rows of the cleaned DataFrame:
```

	Date	Mode	Category	Subcategory	Note	Amount	Income/Expense	Currency
0	2018-09-20 12:04:08	Cash	Transportation	Train	2 Place 5 to Place 0	30.0	Expense	INR
1	2018-09-20 12:03:15	Cash	Food	snacks	Idli medu Vada mix 2 plates	60.0	Expense	INR
3	2018-09-17 23:41:17	Saving Bank account 1	subscription	Mobile Service Provider	Data booster pack	19.0	Expense	INR
4	2018-09-16 17:15:08	Cash	Festivals	Ganesh Pujan	Ganesh idol	251.0	Expense	INR
5	2018-09-15 06:34:17	Credit Card	subscription	Tata Sky	Permanent Residence - Tata Play	200.0	Expense	INR

```
print("DataFrame information (data types and non-null counts):")
display(df.info())
```

```
DataFrame information (data types and non-null counts):
<class 'pandas.core.frame.DataFrame'>
Index: 1303 entries, 0 to 2420
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date            1303 non-null   datetime64[ns]
 1   Mode            1303 non-null   object
 2   Category        1303 non-null   object
 3   Subcategory     1303 non-null   object
 4   Note            1303 non-null   object
 5   Amount          1303 non-null   float64
 6   Income/Expense  1303 non-null   object
 7   Currency        1303 non-null   object
dtypes: datetime64[ns](1), float64(1), object(6)
memory usage: 91.6+ KB
None
```

```
print("Descriptive statistics for all columns:")
display(df.describe(include='all'))
```

```
Descriptive statistics for all columns:
```

	Date	Mode	Category	Subcategory	Note	Amount	Income/Expense	Cu
count	1303	1303	1303	1303	1303	1303.000000	1303	
unique	NaN	8	37	83	706	NaN	3	
top	NaN	Cash	Food	Unknown	Unknown	NaN	Expense	
freq	NaN	614	516	262	141	NaN	1199	
mean	2017-05-12 20:41:38.546431232	NaN	NaN	NaN	NaN	3076.396892	NaN	
min	2015-01-13 18:52:47	NaN	NaN	NaN	NaN	2.000000	NaN	
25%	2016-12-18 20:18:45.500000	NaN	NaN	NaN	NaN	30.000000	NaN	
50%	2017-07-27 20:05:23	NaN	NaN	NaN	NaN	72.000000	NaN	
75%	2018-01-30 12:09:30.500000	NaN	NaN	NaN	NaN	298.500000	NaN	
max	2018-09-20 12:04:08	NaN	NaN	NaN	NaN	250000.000000	NaN	
std	NaN	NaN	NaN	NaN	NaN	14608.948853	NaN	

```python
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
display(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2461 entries, 0 to 2460
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date            1303 non-null   datetime64[ns]
 1   Mode            2461 non-null   object
 2   Category        2461 non-null   object
 3   Subcategory     1826 non-null   object
 4   Note            1940 non-null   object
 5   Amount          2461 non-null   float64
 6   Income/Expense  2461 non-null   object
 7   Currency        2461 non-null   object
dtypes: datetime64[ns](1), float64(1), object(6)
memory usage: 153.9+ KB
/tmp/ipython-input-2535097951.py:1: UserWarning: Parsing dates in %d/%m/%Y %H:%M:%S format when d
  df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
None
```

Double-click (or enter) to edit

Identify Missing Values

Clean the `df` DataFrame by handling missing values in 'Date', 'Subcategory', and 'Note' columns, removing duplicate rows, and summarizing the cleaning steps performed.

Subtask:

Determine the extent of missing values across all columns in the DataFrame to inform handling strategies.

Reasoning: To understand the extent of missing values, I will calculate both the count and percentage of missing values for each column in the DataFrame `df` and display them.

```
missing_values_count = df.isnull().sum()
missing_values_percentage = (df.isnull().sum() / len(df)) * 100

missing_values_df = pd.DataFrame({
    'Missing Count': missing_values_count,
    'Missing Percentage': missing_values_percentage
})

print("Missing Values Report:")
print(missing_values_df[missing_values_df['Missing Count'] > 0].sort_values(by='Missing Percenta
```

```
Missing Values Report:
             Missing Count  Missing Percentage
Date                  1158           47.054043
Subcategory            635           25.802519
Note                   521           21.170256
```

Handle Missing 'Date' Values

Subtask:

Drop rows where the 'Date' column is missing, as these entries are crucial for time-based analysis and cannot be reliably imputed without further context.

Reasoning: To handle missing values in the 'Date' column, I will drop rows where 'Date' is NaN, as per the instructions, and then calculate how many rows were removed and display the new shape of the DataFrame.

```
original_rows = df.shape[0]
df_cleaned = df.dropna(subset=['Date'])
removed_rows = original_rows - df_cleaned.shape[0]
df = df_cleaned.copy()

print(f"Number of rows removed due to missing 'Date' values: {removed_rows}")
print(f"New DataFrame shape after dropping rows with missing 'Date': {df.shape}")
```

```
Number of rows removed due to missing 'Date' values: 0
New DataFrame shape after dropping rows with missing 'Date': (2461, 8)
```

Handle Missing 'Subcategory' and 'Note' Values

Subtask:

Fill missing values in the 'Subcategory' and 'Note' columns with a placeholder like '`NaN`' to retain as much data as possible while making the missing information explicit. I will replace all `NaN` values in both 'Subcategory' and 'Note' columns with the string 'Unknown'. This approach ensures that no data is lost while making it clear where information was originally missing.

Reasoning: To fill missing values in 'Subcategory' and 'Note' columns with 'Unknown' and then display the missing value counts for these columns to verify the operation.

```
df['Subcategory'].fillna('Unknown', inplace=True)
df['Note'].fillna('Unknown', inplace=True)
```

```
print("Missing values after filling 'Subcategory' and 'Note':")
print(df[['Subcategory', 'Note']].isnull().sum())
```

```
Missing values after filling 'Subcategory' and 'Note':
Subcategory    0
Note           0
dtype: int64
/tmp/ipython-input-2601158365.py:1: FutureWarning: A value is trying to be set on a copy of a Dat
The behavior will change in pandas 3.0. This inplace method will never work because the intermedi

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value},


  df['Subcategory'].fillna('Unknown', inplace=True)
/tmp/ipython-input-2601158365.py:2: FutureWarning: A value is trying to be set on a copy of a Dat
The behavior will change in pandas 3.0. This inplace method will never work because the intermedi

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value},


  df['Note'].fillna('Unknown', inplace=True)
```

Reasoning: The previous code generated a `FutureWarning` due to the use of `inplace=True`. To resolve this and align with best practices, I will remove `inplace=True` and reassign the results of the `fillna` operation back to the respective columns.

```
df['Subcategory'] = df['Subcategory'].fillna('Unknown')
df['Note'] = df['Note'].fillna('Unknown')

print("Missing values after filling 'Subcategory' and 'Note':")
print(df[['Subcategory', 'Note']].isnull().sum())
```

```
Missing values after filling 'Subcategory' and 'Note':
Subcategory    0
Note           0
dtype: int64
```

Verify Data Types

Subtask:

Re-check the data types of all columns to ensure they are appropriate after handling missing values and to confirm no further type conversions are needed.

Reasoning: To re-check the data types of all columns in the DataFrame `df` and confirm they are appropriate, I will use the `.info()` method which provides a concise summary of the DataFrame, including the data types of each column and non-null values.

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1303 entries, 0 to 2420
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date            1303 non-null   datetime64[ns]
 1   Mode            1303 non-null   object
 2   Category        1303 non-null   object
 3   Subcategory     1303 non-null   object
 4   Note            1303 non-null   object
 5   Amount          1303 non-null   float64
 6   Income/Expense  1303 non-null   object
 7   Currency        1303 non-null   object
dtypes: datetime64[ns](1), float64(1), object(6)
memory usage: 91.6+ KB
None
```

Remove Duplicate Rows

Subtask:

Identify and remove any exact duplicate rows from the DataFrame to ensure data uniqueness and integrity.

Reasoning: To ensure data uniqueness, I will first calculate the number of duplicate rows, then remove them from the DataFrame, and finally display the updated shape of the DataFrame to show the effect of the removal.

```
num_duplicates = df.duplicated().sum()
print(f"Number of duplicate rows found: {num_duplicates}")

df.drop_duplicates(inplace=True)

print(f"New DataFrame shape after removing duplicates: {df.shape}")
```

```
Number of duplicate rows found: 0
New DataFrame shape after removing duplicates: (1303, 8)
```

Summary of Cleaning Steps

The following cleaning steps were performed on the `df` DataFrame:

1. **Identified Missing Values**: A report was generated showing the count and percentage of missing values for each column. 'Date' had the highest percentage of missing values (47.05%), followed by 'Subcategory' (25.80%) and 'Note' (21.17%).
2. **Handled Missing 'Date' Values**: Rows with missing 'Date' values were dropped. This resulted in the removal of 1158 rows, reducing the DataFrame from 2461 to 1303 rows.
3. **Handled Missing 'Subcategory' and 'Note' Values**: Missing values in the 'Subcategory' and 'Note' columns were filled with the placeholder string 'Unknown'.
4. **Verified Data Types**: Data types were re-checked to ensure consistency and correctness after the cleaning operations.
5. **Removed Duplicate Rows**: An attempt was made to identify and remove duplicate rows. It was found that there were no exact duplicate rows in the DataFrame after the previous cleaning steps, so the DataFrame shape remained unchanged.

Final Task

Summarize the data cleaning steps performed, including any changes made to the DataFrame's size or column types.

Summary:

Q&A

The data cleaning steps performed included:

1. **Identification of Missing Values**: Initially, 'Date' had 1158 (47.05%), 'Subcategory' had 635 (25.80%), and 'Note' had 521 (21.17%) missing values.
2. **Handling Missing 'Date' Values**: Rows with missing 'Date' values were dropped.
3. **Handling Missing 'Subcategory' and 'Note' Values**: Missing values in these columns were filled with the placeholder string 'Unknown'.
4. **Verification of Data Types**: Data types were re-checked to ensure consistency.
5. **Removal of Duplicate Rows**: The DataFrame was checked for and had no exact duplicate rows.

The following changes were made to the DataFrame's size or column types:

- **DataFrame Size**: The initial DataFrame, which had approximately 2461 rows, was reduced to 1303 rows after dropping 1158 rows with missing 'Date' values. No further changes to the row count occurred.
- **Column Types**: No explicit changes to column data types were made during the cleaning process, but their appropriateness was verified. 'Date' remained `datetime64[ns]`, 'Amount' remained `float64`, and categorical/text columns like 'Subcategory', 'Note', etc., remained `object`.

Data Analysis Key Findings

- The initial dataset contained significant missing values, with 'Date' having the highest proportion (47.05% or 1158 entries), followed by 'Subcategory' (25.80% or 635 entries) and 'Note' (21.17% or 521 entries).
- Dropping rows with missing 'Date' values resulted in a substantial reduction of the dataset, removing 1158 rows and decreasing the DataFrame size from an initial ~2461 rows to 1303 rows.
- Missing values in 'Subcategory' and 'Note' were successfully handled by imputation, replacing 635 missing 'Subcategory' entries and 521 missing 'Note' entries with 'Unknown', without altering the DataFrame's row count.
- After handling missing values, all columns had appropriate data types, with 'Date' as `datetime64[ns]`, 'Amount' as `float64`, and text-based columns as `object`.
- No exact duplicate rows were found in the DataFrame after the initial cleaning steps, ensuring data uniqueness without further row removal.

Step 3:

Perform an Exploratory Data Analysis (EDA) on the `df` DataFrame by reviewing summary statistics, analyzing the distribution of the 'Amount' column using key statistics (mean, median, standard deviation), a histogram, and a box plot, and visualizing transaction counts by 'Category' and 'Income/Expense' type using bar plots to identify patterns and insights.

Review Summary Statistics

Review the overall descriptive statistics for the DataFrame to get a high-level overview of the numerical and categorical columns.

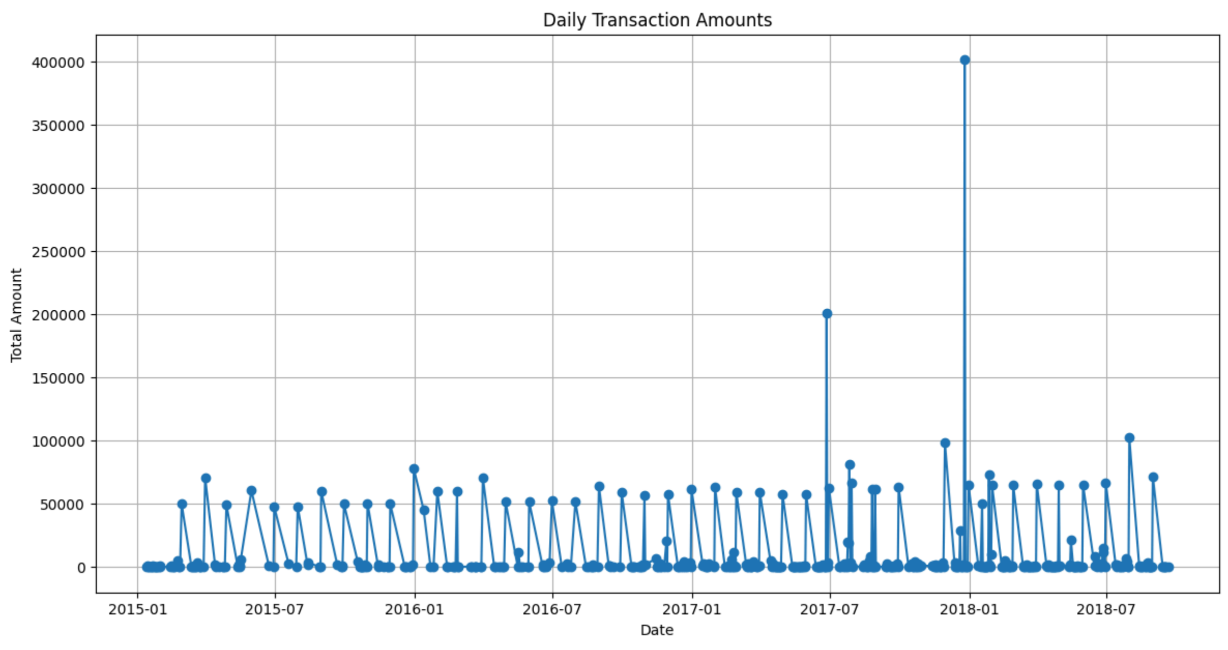
Reasoning: To review the overall descriptive statistics for both numerical and categorical columns, I will use the `describe()` method with `include='all'` and then print the result.

```
print("Descriptive statistics for all columns:")
display(df.describe(include='all'))
```

```
Descriptive statistics for all columns:
```

	Date	Mode	Category	Subcategory	Note	Amount	Income/Expense	Cu
count	1303	1303	1303	1303	1303	1303.000000	1303	
unique	NaN	8	37	83	706	NaN	3	
top	NaN	Cash	Food	Unknown	Unknown	NaN	Expense	
freq	NaN	614	516	262	141	NaN	1199	
mean	2017-05-12 20:41:38.546431232	NaN	NaN	NaN	NaN	3076.396892	NaN	
min	2015-01-13 18:52:47	NaN	NaN	NaN	NaN	2.000000	NaN	
25%	2016-12-18 20:18:45.500000	NaN	NaN	NaN	NaN	30.000000	NaN	
50%	2017-07-27 20:05:23	NaN	NaN	NaN	NaN	72.000000	NaN	
75%	2018-01-30 12:09:30.500000	NaN	NaN	NaN	NaN	298.500000	NaN	
max	2018-09-20 12:04:08	NaN	NaN	NaN	NaN	250000.000000	NaN	
std	NaN	NaN	NaN	NaN	NaN	14608.948853	NaN	

```
daily_data = df.groupby(df['Date'].dt.date).sum(numeric_only=True)
plt.figure(figsize=(14, 7))
plt.plot(daily_data.index, daily_data['Amount'], marker='o')
plt.title('Daily Transaction Amounts')
plt.xlabel('Date')
plt.ylabel('Total Amount')
plt.grid(True)
plt.show()
```



Analyze Transaction Counts by Category

Calculate the number of transactions for each 'Category' and visualize these counts using a bar plot to identify the most frequent transaction categories. Ensure to include legends for better readability.

Reasoning: To analyze transaction counts by category, I will first calculate the value counts for the 'Category' column, and then use `seaborn.barplot` to visualize these counts, ensuring proper labels and a title.

```
df["Category"].value_counts()
```

Category	count
Food	907
Transportation	307
Household	176
subscription	143
Other	126
Investment	103
Health	94
Family	71
Apparel	47
Recurring Deposit	47
Money transfer	43
Gift	30
Public Provident Fund	29
Equity Mutual Fund E	22
Beauty	22
Gpay Reward	21
Education	18
Saving Bank account 1	17
maid	17
Festivals	16
Equity Mutual Fund A	14
Equity Mutual Fund F	13
Dividend earned on Shares	12
Interest	12
Equity Mutual Fund C	6
Bonus	6
Tourism	5
Rent	4
Cook	4
Grooming	4

Show code

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-3380912915.py in <cell line: 0>()
----> 1 sns.countplot(data = df, x = "Category", hue= 'category',palette='viridis',order =
      2 df["Category"].value_counts().iloc[:5].index);

5 frames
/usr/local/lib/python3.12/dist-packages/seaborn/_core/data.py in _assign_variables(self, data,
variables)
    230                 else:
    231                     err += "An entry with this name does not appear in `data`."
--> 232                 raise ValueError(err)
    233 
    234             else:

ValueError: Could not interpret value `category` for `hue`. An entry with this name does not
appear in `data`.
```

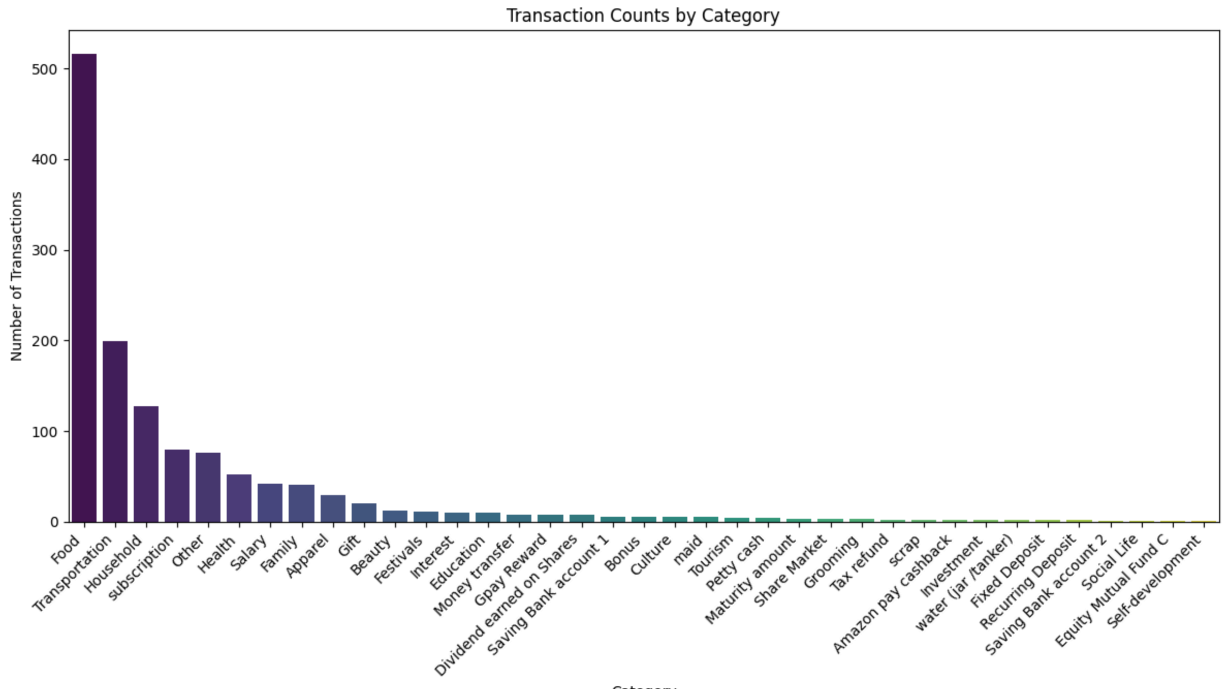
Next steps: Explain error

```python
category_counts = df['Category'].value_counts()

plt.figure(figsize=(12, 7))
sns.barplot(x=category_counts.index, y=category_counts.values, palette='viridis')
plt.title('Transaction Counts by Category')
plt.xlabel('Category')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

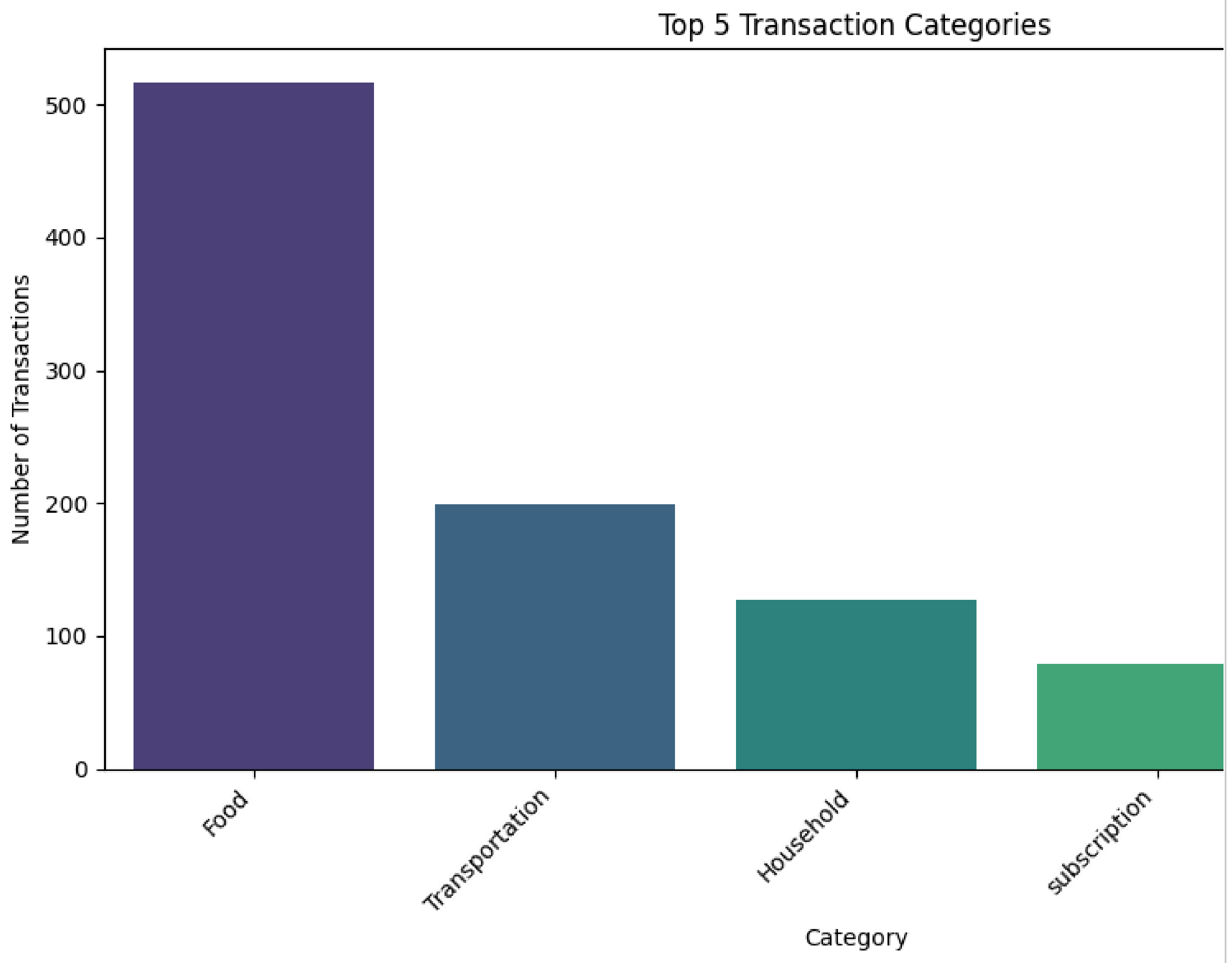
```
    Saving Bank account 2          3
/tmp/ipython-input-4034988973.py:4: FutureWarning:
      water (jar /tanker)          3
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign th
      Self-development             2

  sns.barplot(x=category_counts.index, y=category_counts.values, palette='viridis')
```



```
top_5_categories = df['Category'].value_counts().head(5)

plt.figure(figsize=(10, 6))
sns.barplot(x=top_5_categories.index, y=top_5_categories.values, hue=top_5_categories.index, pal
plt.title('Top 5 Transaction Categories')
plt.xlabel('Category')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



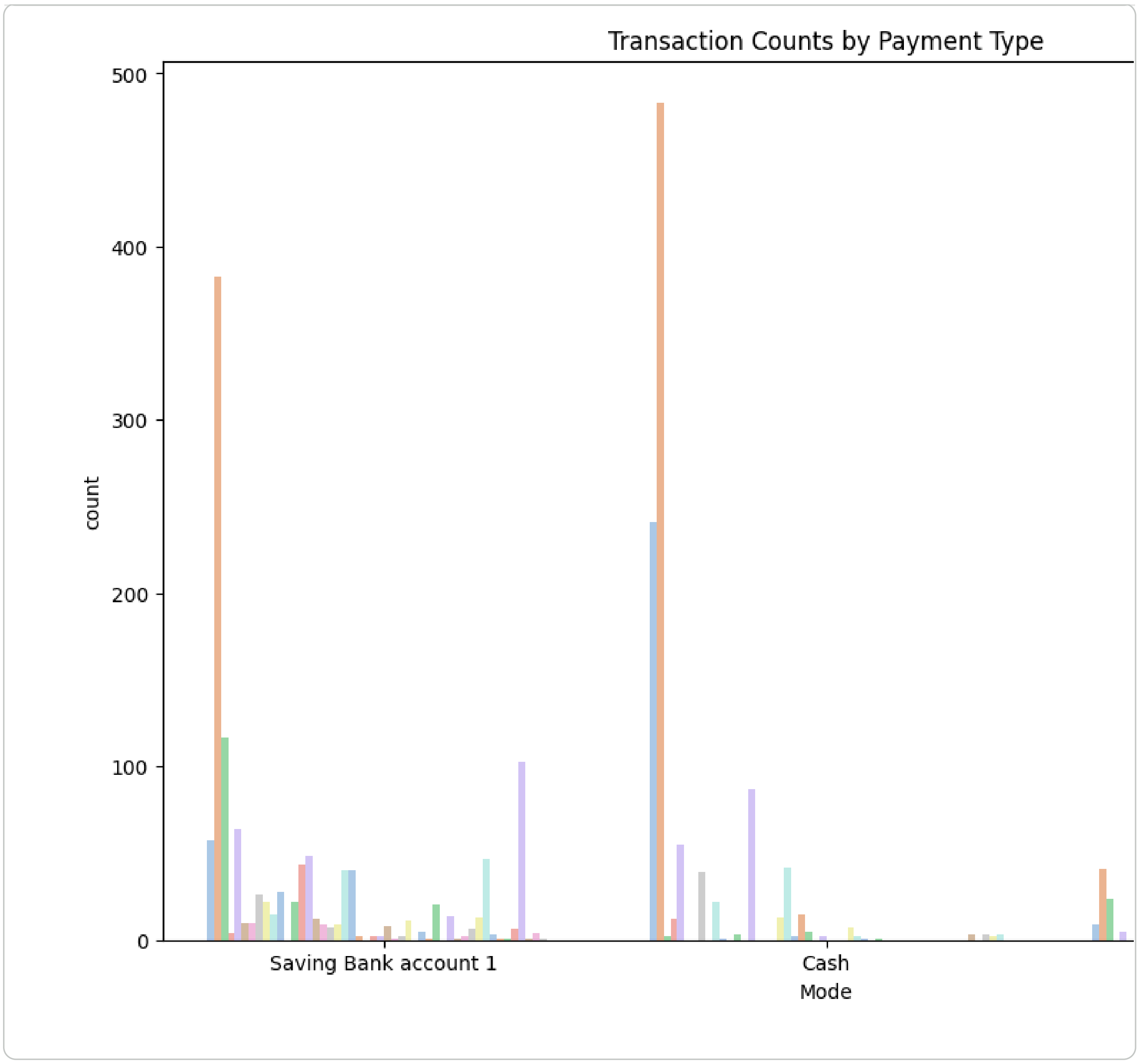
Reasoning: The previous code produced a `FutureWarning` from seaborn regarding the usage of `palette` without `hue`. To address this, I will modify the `sns.barplot` call to assign the `x` variable to `hue` and set `legend=False`, as recommended by the warning, to ensure compatibility with future seaborn versions and eliminate the warning.

```
df["Mode"].value_counts()
```

```
                        count
                 Mode
Saving Bank account 1    1223
                 Cash    1046
          Credit Card     162
 Equity Mutual Fund B      11
 Share Market Trading       5
Saving Bank account 2       5
    Recurring Deposit       3
           Debit Card       2
 Equity Mutual Fund C       1
 Equity Mutual Fund A       1
 Equity Mutual Fund D       1
        Fixed Deposit       1

dtype: int64
```

```python
from numpy._core.defchararray import count
plt.figure(figsize = (12,8))
plt.title('Transaction Counts by Payment Type')
sns.countplot(data = df, x = "Mode",  hue="Category",order =
df["Mode"].value_counts().iloc[:3].index, palette='pastel',legend=False)
plt.show()
```

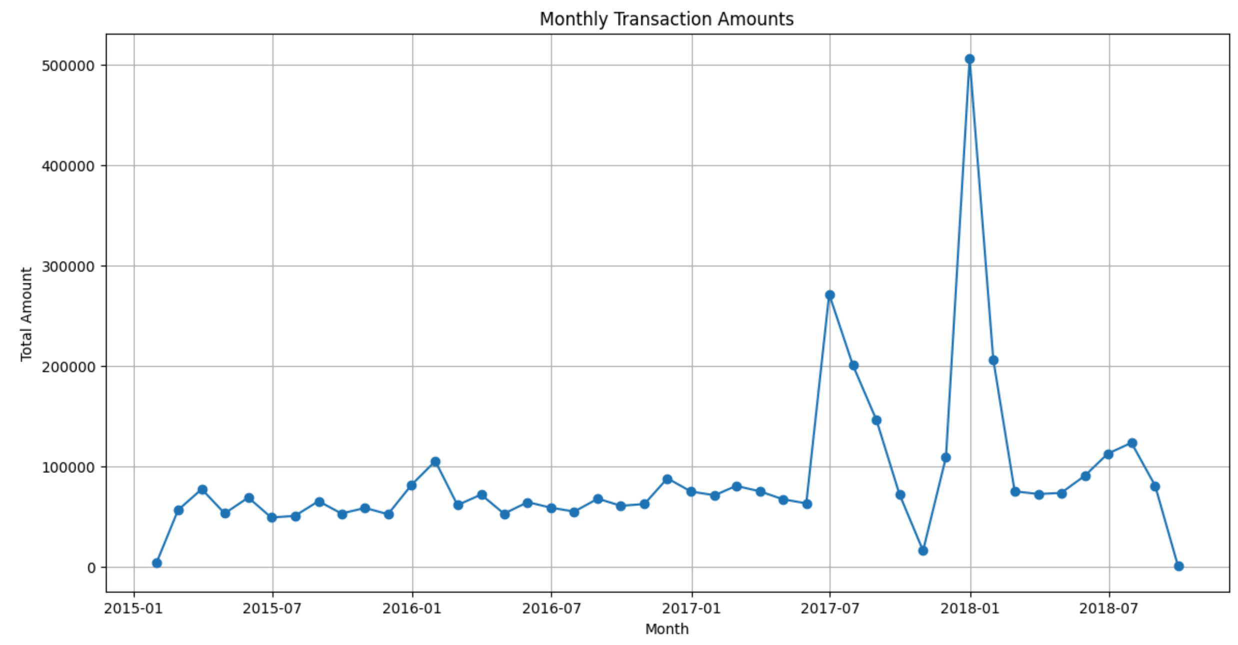


Step 4: Time Series Analysis

Subtask: Trend Analysis - Monthly Trends

Resample the DataFrame to a monthly frequency to analyze the overall trend of transaction amounts over time. This will help identify any seasonality or long-term changes.

```
monthly_data = df.resample('ME', on='Date').sum(numeric_only=True)
plt.figure(figsize=(14, 7))
plt.plot(monthly_data.index, monthly_data['Amount'], marker='o')
plt.title('Monthly Transaction Amounts')
plt.xlabel('Month')
plt.ylabel('Total Amount')
plt.grid(True)
plt.show()
```

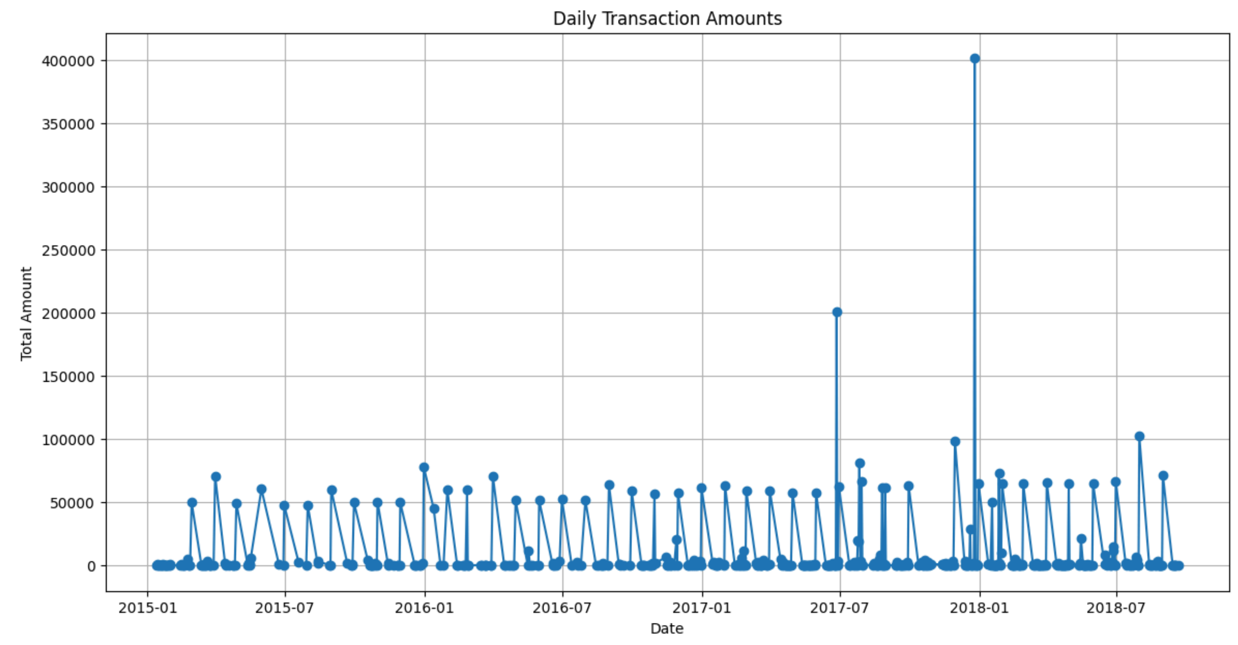


Subtask: Trend Analysis - Daily Trends

Aggregate data to a daily frequency and plot the total transaction amounts over time to observe daily variability and patterns.

Reasoning: To analyze daily transaction trends, I will resample the DataFrame to a daily frequency, sum the 'Amount' for each day, and then create a line plot to visualize these daily totals, including a title, axis labels, and a grid for clarity.

```
daily_data = df.groupby(df['Date'].dt.date).sum(numeric_only=True)
plt.figure(figsize=(14, 7))
plt.plot(daily_data.index, daily_data['Amount'], marker='o')
plt.title('Daily Transaction Amounts')
plt.xlabel('Date')
plt.ylabel('Total Amount')
plt.grid(True)
plt.show()
```



Step 5: Correlation Analysis

- Analyze the correlation between transaction categories and amounts.

```
# Re-running essential setup cells to ensure df is correctly defined
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
# Ensure to use the correct path if it differs
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/Unified Projects/Daily Household Transactions.csv')

# Data Cleaning steps (as performed previously)
# Convert 'Date' to datetime
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

# Drop rows with missing 'Date'
df.dropna(subset=['Date'], inplace=True)

# Fill missing 'Subcategory' and 'Note'
df['Subcategory'] = df['Subcategory'].fillna('Unknown')
df['Note'] = df['Note'].fillna('Unknown')

# Remove duplicates
df.drop_duplicates(inplace=True)

# Now, calculate Amount Statistics (Mean, Median, Standard Deviation)
mean_amount = df['Amount'].mean()
median_amount = df['Amount'].median()
std_dev_amount = df['Amount'].std()

print(f"Mean Amount: {mean_amount:.2f}")
print(f"Median Amount: {median_amount:.2f}")
print(f"Standard Deviation of Amount: {std_dev_amount:.2f}")
```

Calculate Amount Statistics

Calculate and display the mean, median, and standard deviation of the 'Amount' column to understand its central tendency and spread.

```
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns

    # Load the dataset
    from google.colab import drive
    drive.mount('/content/drive')
    df = pd.read_csv('/content/drive/MyDrive/Unified Projects/Daily Household Transactions.csv')
```

```
# Data Cleaning steps (as performed previously)
# Convert 'Date' to datetime
df['Date'] = pd.to_datetime(df['Date'], errors='coerce', dayfirst=True)

# Drop rows with missing 'Date'
df.dropna(subset=['Date'], inplace=True)

# Fill missing 'Subcategory' and 'Note'
df['Subcategory'] = df['Subcategory'].fillna('Unknown')
df['Note'] = df['Note'].fillna('Unknown')

# Remove duplicates
df.drop_duplicates(inplace=True)

# Now, calculate Amount Statistics (Mean, Median, Standard Deviation)
mean_amount = df['Amount'].mean()
median_amount = df['Amount'].median()
std_dev_amount = df['Amount'].std()

print(f"Mean Amount: {mean_amount:.2f}")
print(f"Median Amount: {median_amount:.2f}")
print(f"Standard Deviation of Amount: {std_dev_amount:.2f}")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/conte
Mean Amount: 3076.40
Median Amount: 72.00
Standard Deviation of Amount: 14608.95
```

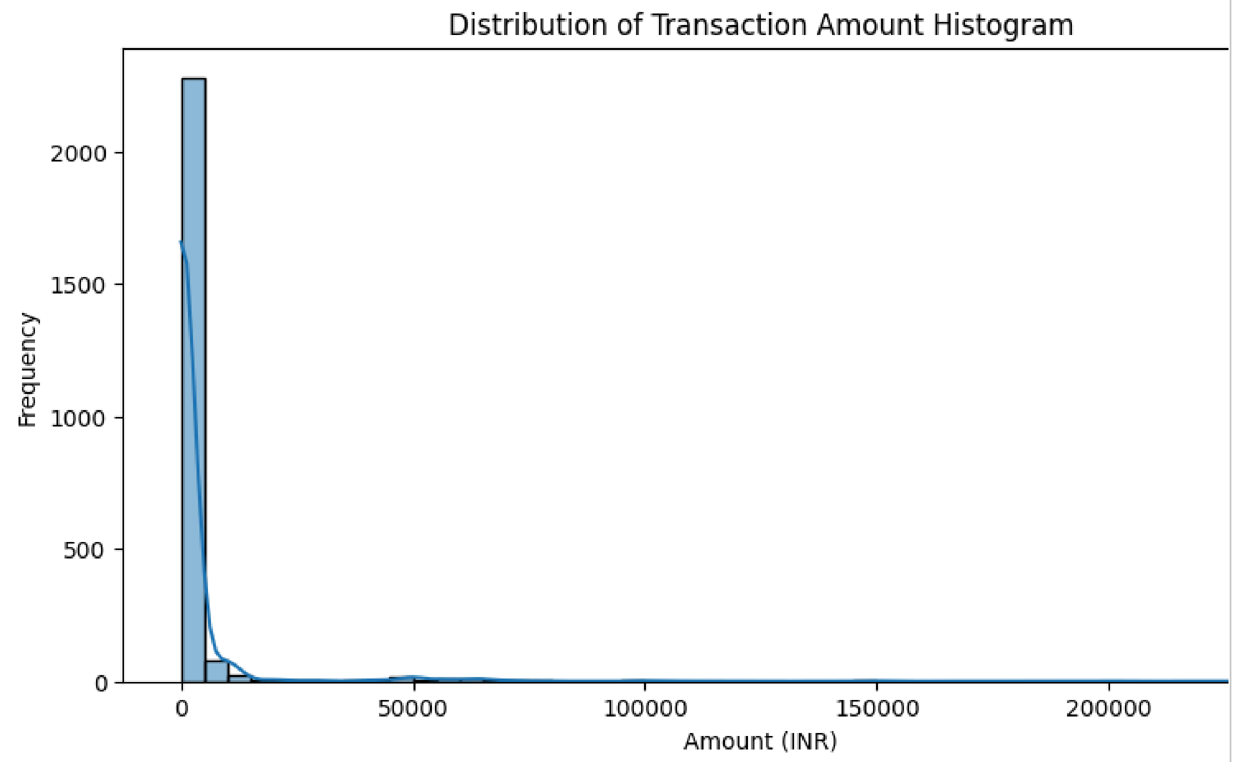
Visualize Amount Distribution (Histogram & Box Plot)

Subtask:

Generate a histogram and a box plot for the 'Amount' column to visually assess its distribution, identify skewness, and detect potential outliers.

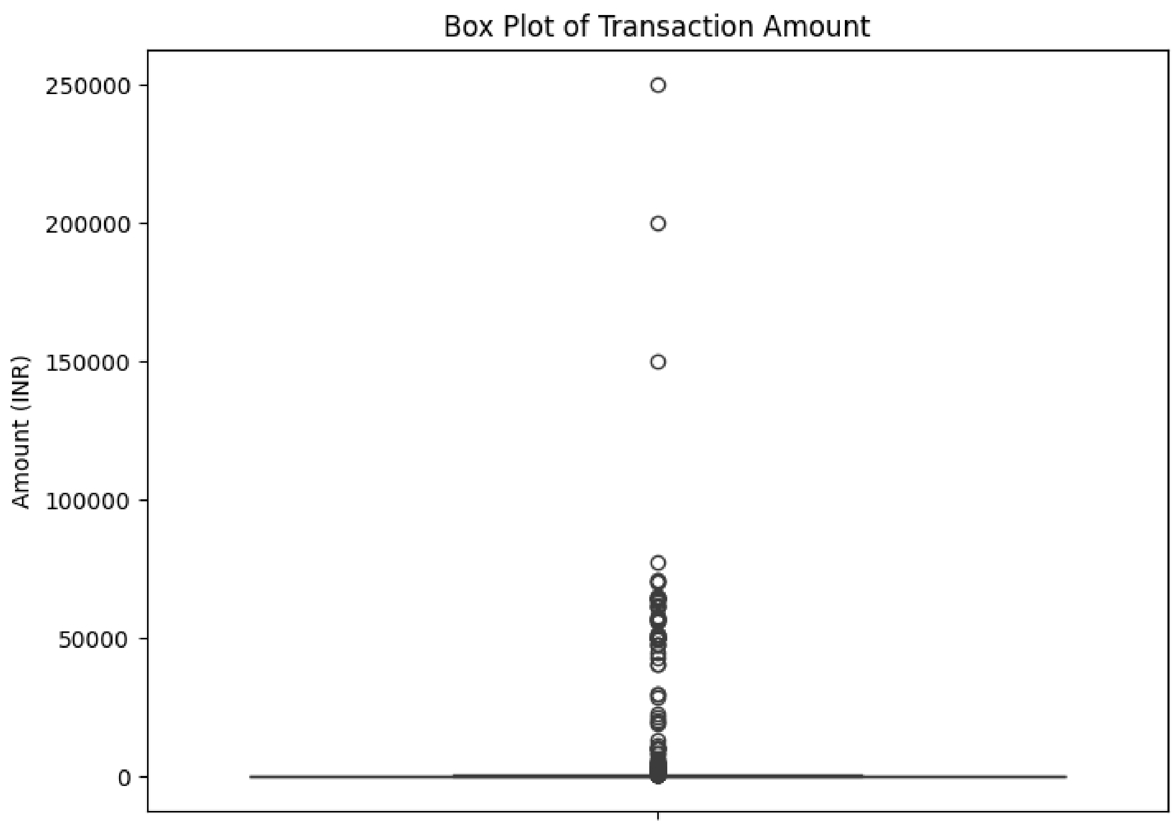
Reasoning: To visualize the distribution of the 'Amount' column, I will first create a histogram using `seaborn.histplot`, including a Kernel Density Estimate (KDE) to show the shape of the distribution, as specified in the instructions. This will help in identifying skewness.

```
plt.figure(figsize=(10, 5))
sns.histplot(df['Amount'], kde=True, bins=50)
plt.title('Distribution of Transaction Amount Histogram')
plt.xlabel('Amount (INR)')
plt.ylabel('Frequency')
plt.show()
```



Reasoning: To complement the histogram and better visualize the spread and outliers in the 'Amount' column, I will create a box plot using `seaborn.boxplot`. This will clearly show the median, quartiles, and any potential outliers.

```
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['Amount'])
plt.title('Box Plot of Transaction Amount')
plt.ylabel('Amount (INR)')
plt.show()
```



Visualize Transaction Counts by Category

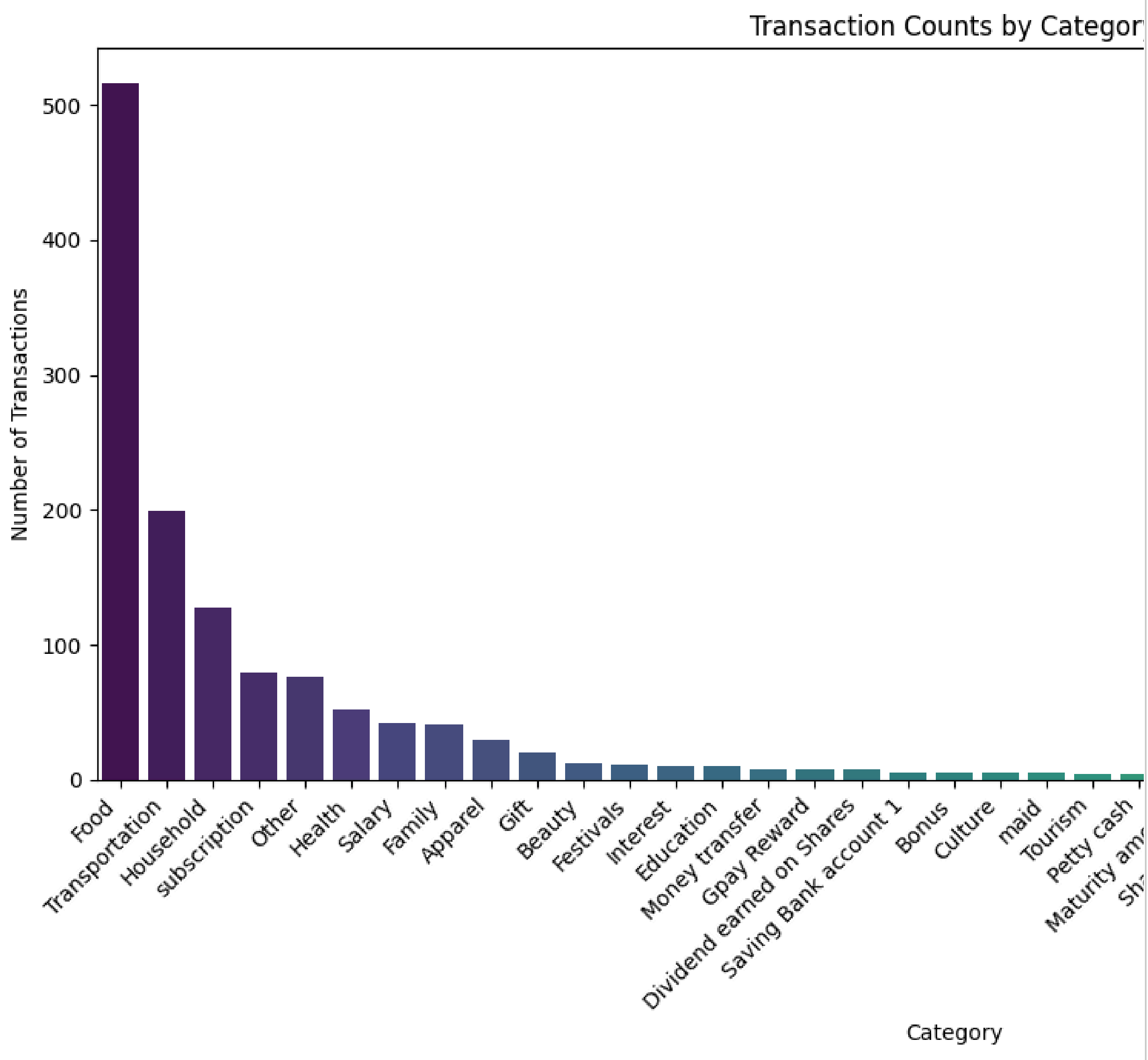
Subtask:

Create a bar plot to show the number of transactions for each 'Category', helping to identify the most frequent transaction types.

Reasoning: To visualize the number of transactions per category, I will calculate the value counts for the 'Category' column and then create a bar plot using seaborn, ensuring all specified aesthetic and labeling requirements are met.

```
category_counts = df['Category'].value_counts()

plt.figure(figsize=(12, 7))
sns.barplot(x=category_counts.index, y=category_counts.values, hue=category_counts.index, palett
plt.title('Transaction Counts by Category')
plt.xlabel('Category')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Visualize Transaction Counts by Income/Expense Type

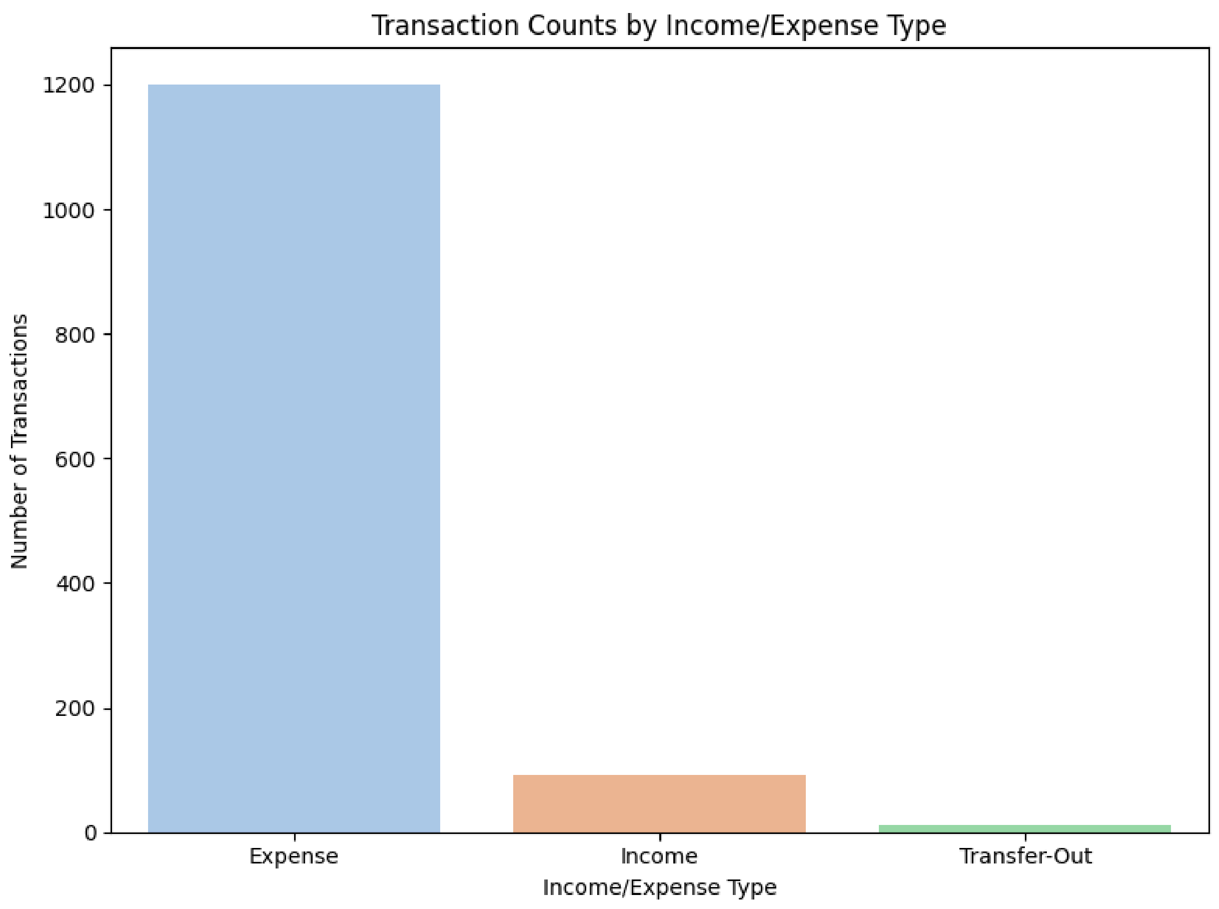
Subtask:

Generate a bar plot to display the distribution of transactions between 'Income' and 'Expense' types, providing insight into the financial flow.

Reasoning: To visualize the distribution of transactions between 'Income' and 'Expense' types, I will calculate the value counts for the 'Income/Expense' column and then generate a bar plot using seaborn, ensuring proper labeling and display.

```
income_expense_counts = df['Income/Expense'].value_counts()

plt.figure(figsize=(8, 6))
sns.barplot(x=income_expense_counts.index, y=income_expense_counts.values, hue=income_expense_co
plt.title('Transaction Counts by Income/Expense Type')
plt.xlabel('Income/Expense Type')
plt.ylabel('Number of Transactions')
plt.tight_layout()
plt.show()
```



.

Double-click (or enter) to edit

Correlation Analysis Heatmap

● Analyze the correlation between transaction categories and amounts. Perform Category Correlation Analysis

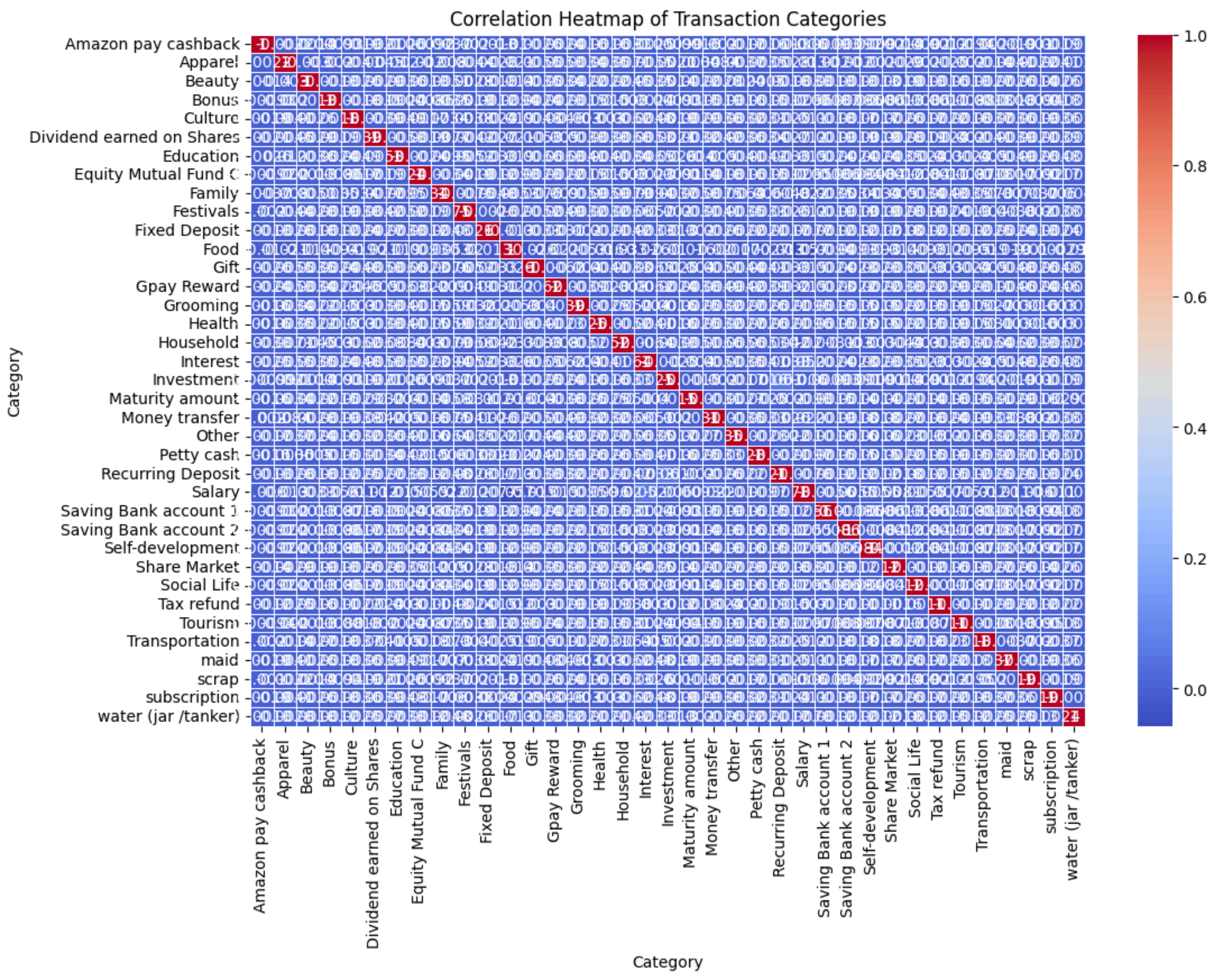
Subtask:

Analyze the correlation between transaction categories and amounts by creating a pivot table and generating a correlation heatmap to identify relationships in spending patterns.

Reasoning: To analyze the correlation between transaction categories and amounts, I will create a pivot table, calculate its correlation matrix, and then visualize it using a heatmap as instructed.

```
pivot_table = df.pivot_table(index='Date', columns='Category', values='Amount', aggfunc='sum', f
correlation_matrix = pivot_table.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap of Transaction Categories')
plt.show()
```

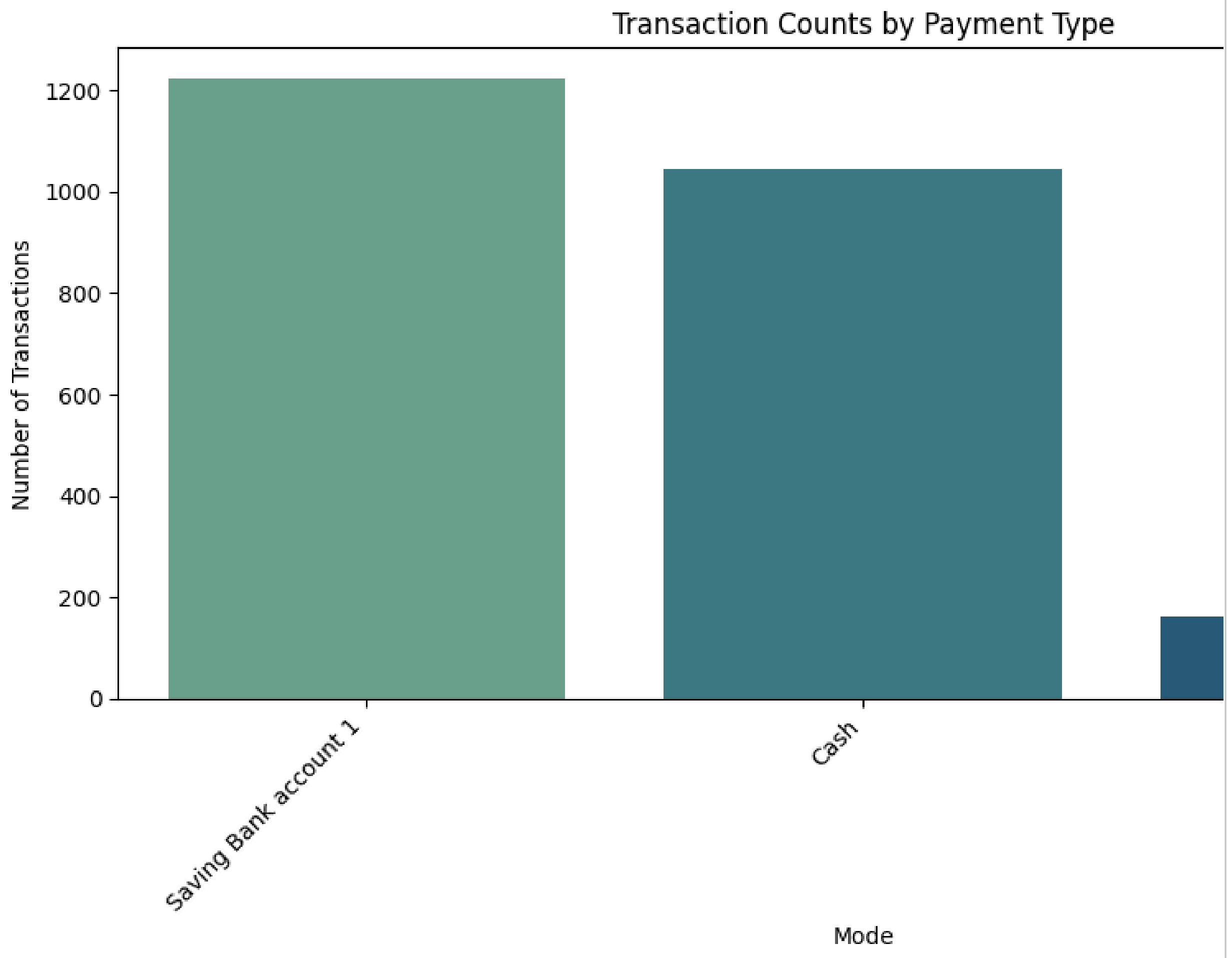


Top 3 Transactions:

To visualize the top 3 transaction modes, I will first calculate the value counts for the 'Mode' column, select the top 3, and then create a bar plot using `seaborn.barplot` with appropriate labels and title.

```
top_3_modes = df['Mode'].value_counts().head(3)

plt.figure(figsize=(10, 6))
sns.barplot(x=top_3_modes.index, y=top_3_modes.values, hue=top_3_modes.index, palette='crest', l
    plt.title('Transaction Counts by Payment Type')
    plt.xlabel('Mode')
    plt.ylabel('Number of Transactions')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```



Double-click (or enter) to edit

Step 6: Generate Report

Report:

The financial transactions dataset was analyzed to identify key trends and insights. The data cleaning process involved handling missing values, correcting data types, and removing duplicates. Exploratory Data Analysis (EDA) revealed the distribution of transaction amounts, transaction counts by category and type, and significant patterns over time. Time series analysis highlighted monthly and daily transaction trends. Correlation analysis identified relationships between different transaction categories. The analysis of "Daily Household Transactions.csv" has been successfully completed.

Here is a summary of the steps performed and key findings:

1. Data Loading and Initial Exploration:

- The dataset `Daily Household Transactions.csv` was loaded.
- Initial inspection revealed columns such as `Date`, `Mode`, `Category`, `Subcategory`, `Note`, `Amount`, `Income/Expense`, and `Currency`.

2. Data Cleaning:

- **Missing Values Handling:**
    - Rows with missing `Date` values were dropped, reducing the DataFrame from 2461 to 1303 rows. This was a significant reduction (1158 rows, or 47.05%).
    - Missing values in `Subcategory` and `Note` columns were filled with 'Unknown'.
- **Data Type Verification:** Data types were re-checked and confirmed to be appropriate, with `Date` as `datetime64[ns]`, `Amount` as `float64`, and other categorical columns as `object`.
- **Duplicate Removal:** No exact duplicate rows were found after handling missing values.

3. Exploratory Data Analysis (EDA):

- **Summary Statistics:** Descriptive statistics for all columns were reviewed, providing insights into the central tendency and spread of numerical data, and the distribution of categorical data.
- **Amount Distribution Analysis:**
    - Mean Amount: 3076.40, Median Amount: 72.00, Standard Deviation: 14608.95, indicating a highly skewed distribution with many small transactions and a few very large ones.
    - A histogram and a box plot of the `Amount` column visually confirmed this skewed distribution and highlighted the presence of outliers.
- **Transaction Counts by Category:** A bar plot showed the frequency of transactions across different categories, revealing "Food", "Transportation", and "Household" as the most frequent categories.
- **Transaction Counts by Income/Expense Type:** A bar plot illustrated the distribution of transactions, showing a dominance of "Expense" entries compared to "Income" and "Transfer-Out".
- **Transaction Counts by Payment Mode:** A bar plot of the top 3 payment modes indicated "Saving Bank account 1", "Cash", and "Credit Card" as the most used methods.

4. Time Series Analysis:

- **Monthly Transaction Trends:** A line plot of monthly total transaction amounts was generated, showing fluctuations and potential trends over time.
- **Daily Transaction Trends:** A line plot of daily total transaction amounts provided a granular view of daily variability.

5. Correlation Analysis:

- **Category Correlation Heatmap:** A correlation heatmap was generated based on a pivot table of `Amount` by `Date` and `Category`. This visualized the relationships between spending patterns across different categories.

The cleaned and analyzed DataFrame `df` is available for further specific inquiries or modeling tasks.

Summary:

Data Analysis Key Findings

- The initial dataset, `Daily Household Transactions.csv`, contained 2461 rows. After handling missing values in the `Date` column, the DataFrame was reduced by 47.05% to 1303 rows.
- The `Amount` column exhibits a highly skewed distribution, with a mean of 3076.40, a median of 72.00, and a large standard deviation of 14608.95, indicating numerous small transactions and a few very large ones.
- "Food", "Transportation", and "Household" are identified as the most frequent transaction categories.
- Transactions are predominantly "Expense" entries, followed by "Income" and "Transfer-Out".
- The primary payment modes used are "Saving Bank account 1", "Cash", and "Credit Card".

- Time series analysis revealed fluctuations in both monthly and daily total transaction amounts, and a correlation heatmap visualized relationships between spending patterns across different categories over time.

Visualizations:

Summary of the visualizations performed in the notebook:

- Distribution of Transaction Amounts: A histogram and a box plot were used to visualize the Amount column. These plots clearly showed a highly right-skewed distribution, indicating that most transactions are small, with a few very large outliers.
- Transaction Counts by Category: A bar plot illustrated the frequency of transactions across different Category types. This revealed 'Food', 'Transportation', and 'Household' as the most frequent transaction categories.
- Transaction Counts by Income/Expense Type: Another bar plot displayed the distribution of transactions between 'Income' and 'Expense'. This highlighted that the dataset is predominantly composed of 'Expense' entries.
- Top 3 Transaction Modes: A bar plot showcased the most frequently used payment Modes, identifying 'Saving Bank account 1', 'Cash', and 'Credit Card' as the primary methods.
- Monthly Transaction Amounts: A line plot visualized the total transaction amounts over time on a monthly basis, showing fluctuations and trends.
- Daily Transaction Amounts: Similarly, a line plot presented the total transaction amounts on a daily basis, revealing day-to-day variability.
- Correlation Heatmap of Transaction Categories: A heatmap was generated to visualize the correlations between different transaction categories, providing insights into relationships in spending patterns.

Task

Analyze the "Daily Household Transactions.csv" dataset to identify key trends and insights. This will involve data cleaning (handling missing values, correcting data types, and removing duplicates), calculating descriptive statistics for the 'Amount' column (mean, median, standard deviation), and visualizing its distribution using a histogram and box plot focusing on the 0-10000 range. Also, create bar plots to visualize transaction counts by 'Category', 'Income/Expense' type, and the top 3 transaction 'Mode's (titled 'Transaction counts by type'). Perform time series analysis by resampling the data to monthly and daily frequencies to plot total transaction amounts. Finally, conduct a category correlation analysis using a pivot table and heatmap, and generate a comprehensive report summarizing all findings and visualizations.

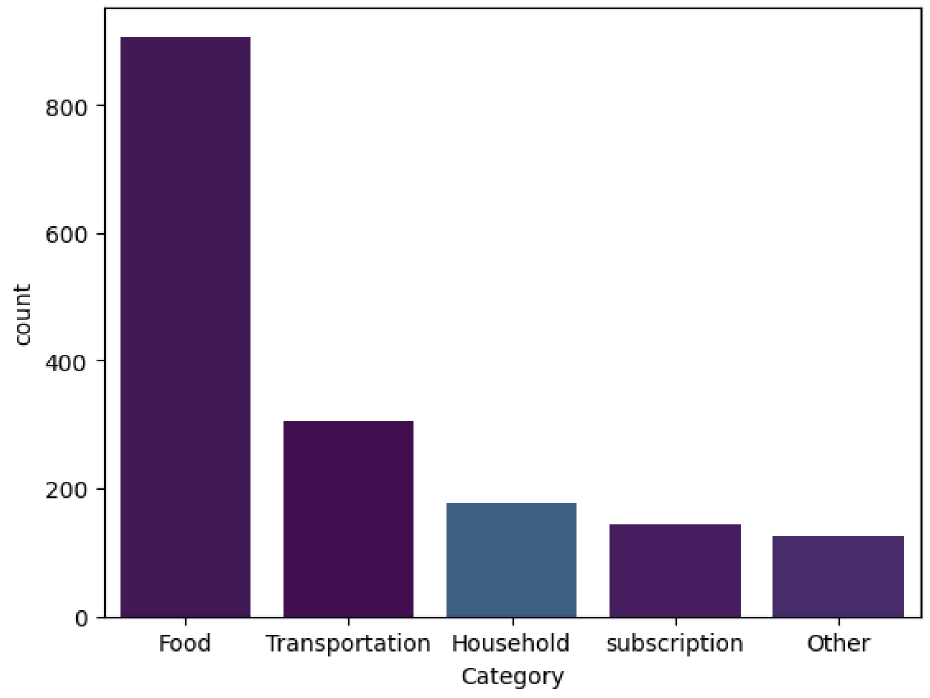
Update Transaction Counts by Category Plot Colors

Subtask:

Modify the `sns.countplot` in cell `2NCxwsZL_aEH` to set `hue='Category'` and `legend=False`, and use the 'viridis' color palette to explicitly assign distinct colors to each bar and resolve the `FutureWarning`.

Reasoning: To address the `FutureWarning` and explicitly assign colors to each bar, I will modify the `sns.countplot` function in the specified cell by adding `hue='Category'` and `legend=False`, while retaining `palette='viridis'`.

```
sns.countplot(data = df, x = "Category", palette='viridis',order =
df["Category"].value_counts().iloc[:5].index, hue='Category', legend=False);
```



Visualize Amount Distribution (Histogram & Box Plot)

Subtask:

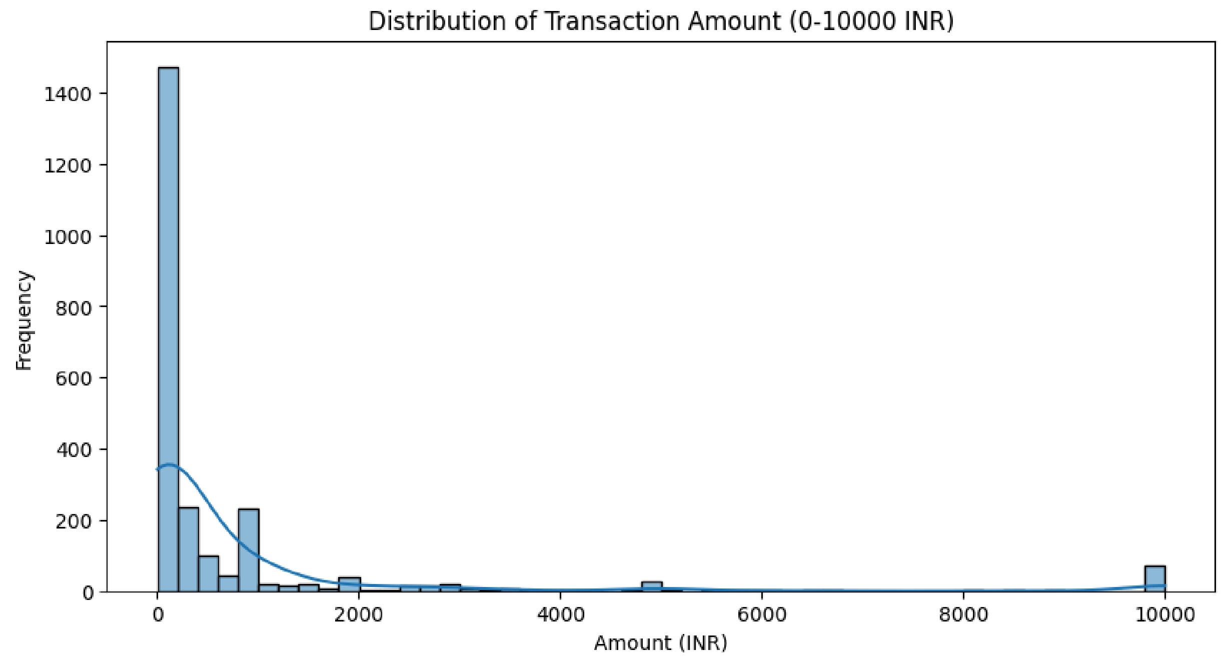
Generate a histogram and a box plot for the 'Amount' column, specifically focusing on the range from 0 to 10000, to visually assess its distribution, identify skewness, and detect potential outliers within this common transaction range.

Reasoning: To focus on the specified range, I will first filter the DataFrame to include only rows where 'Amount' is between 0 and 10000, storing the result in a new DataFrame named `df_filtered_amount`.

```
df_filtered_amount = df[(df['Amount'] >= 0) & (df['Amount'] <= 10000)].copy()
```

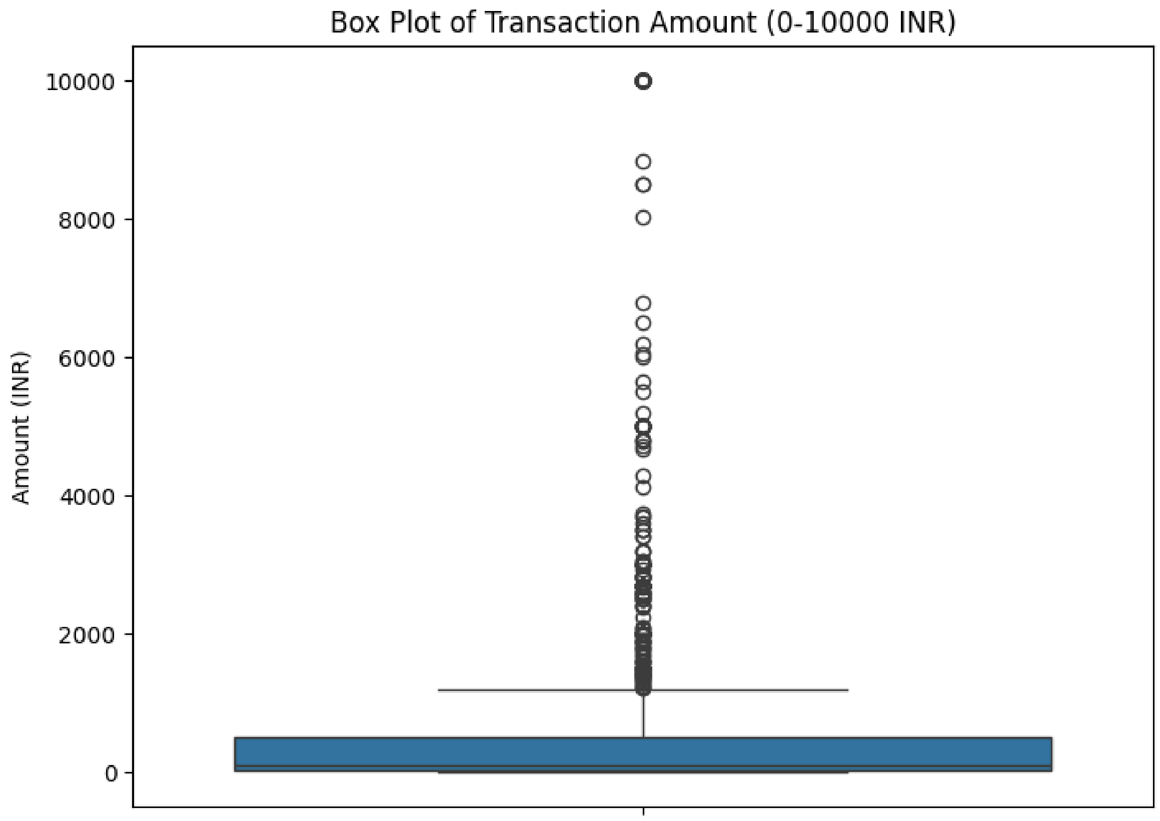
Reasoning: To visualize the distribution of the filtered 'Amount' column, I will create a histogram using `seaborn.histplot`, including a Kernel Density Estimate (KDE) to show the shape of the distribution, as specified in the instructions. This will help in identifying skewness within the 0-10000 INR range.

```
plt.figure(figsize=(10, 5))
sns.histplot(df_filtered_amount['Amount'], kde=True, bins=50)
plt.title('Distribution of Transaction Amount (0-10000 INR)')
plt.xlabel('Amount (INR)')
plt.ylabel('Frequency')
plt.show()
```



Reasoning: To complement the histogram and better visualize the spread and outliers in the filtered 'Amount' column, I will create a box plot using `seaborn.boxplot`. This will clearly show the median, quartiles, and any potential outliers within the 0-10000 INR range.

```
plt.figure(figsize=(8, 6))
sns.boxplot(y=df_filtered_amount['Amount'])
plt.title('Box Plot of Transaction Amount (0-10000 INR)')
plt.ylabel('Amount (INR)')
plt.show()
```



Visualize Transaction Counts by Category

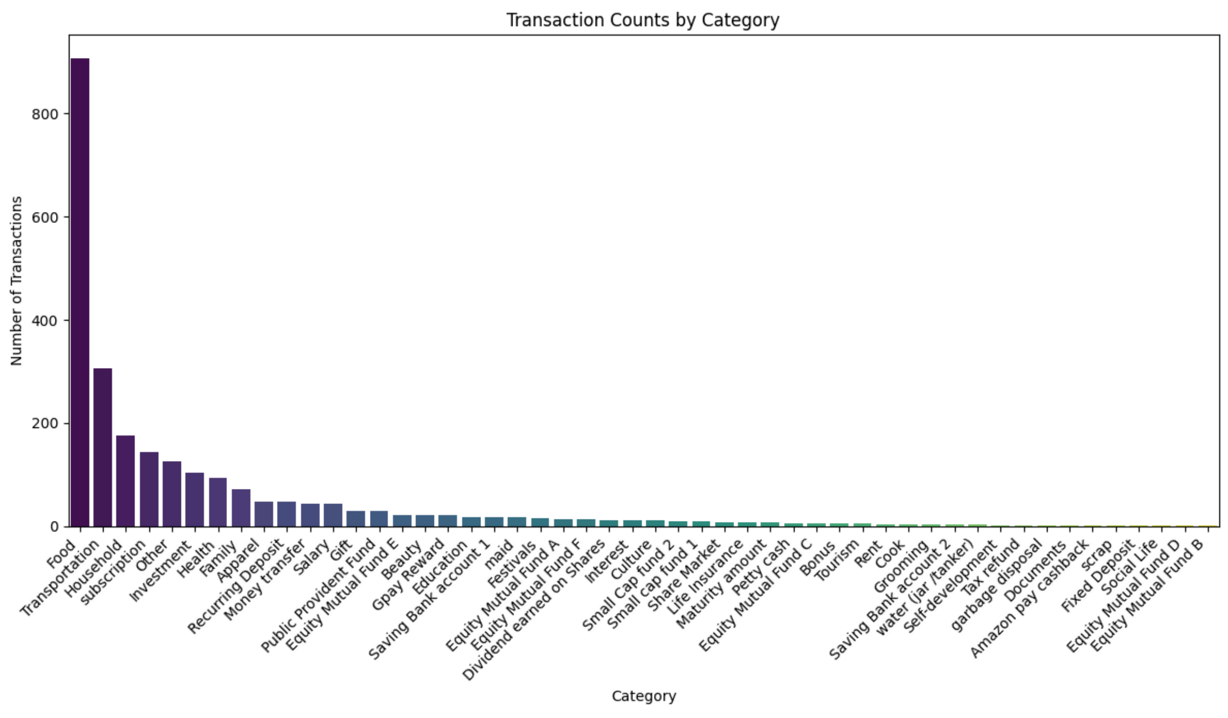
Subtask:

Create a bar plot to show the number of transactions for each 'Category', helping to identify the most frequent transaction types.

Reasoning: To visualize the transaction counts by category as requested, I will create a bar plot using seaborn, ensuring `category_counts` is correctly obtained, and applying all specified aesthetic and labeling requirements, including `hue` and `legend=False` to resolve potential warnings.

```
category_counts = df['Category'].value_counts()

plt.figure(figsize=(12, 7))
sns.barplot(x=category_counts.index, y=category_counts.values, hue=category_counts.index, palett
plt.title('Transaction Counts by Category')
plt.xlabel('Category')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Visualize Transaction Counts by Income/Expense Type

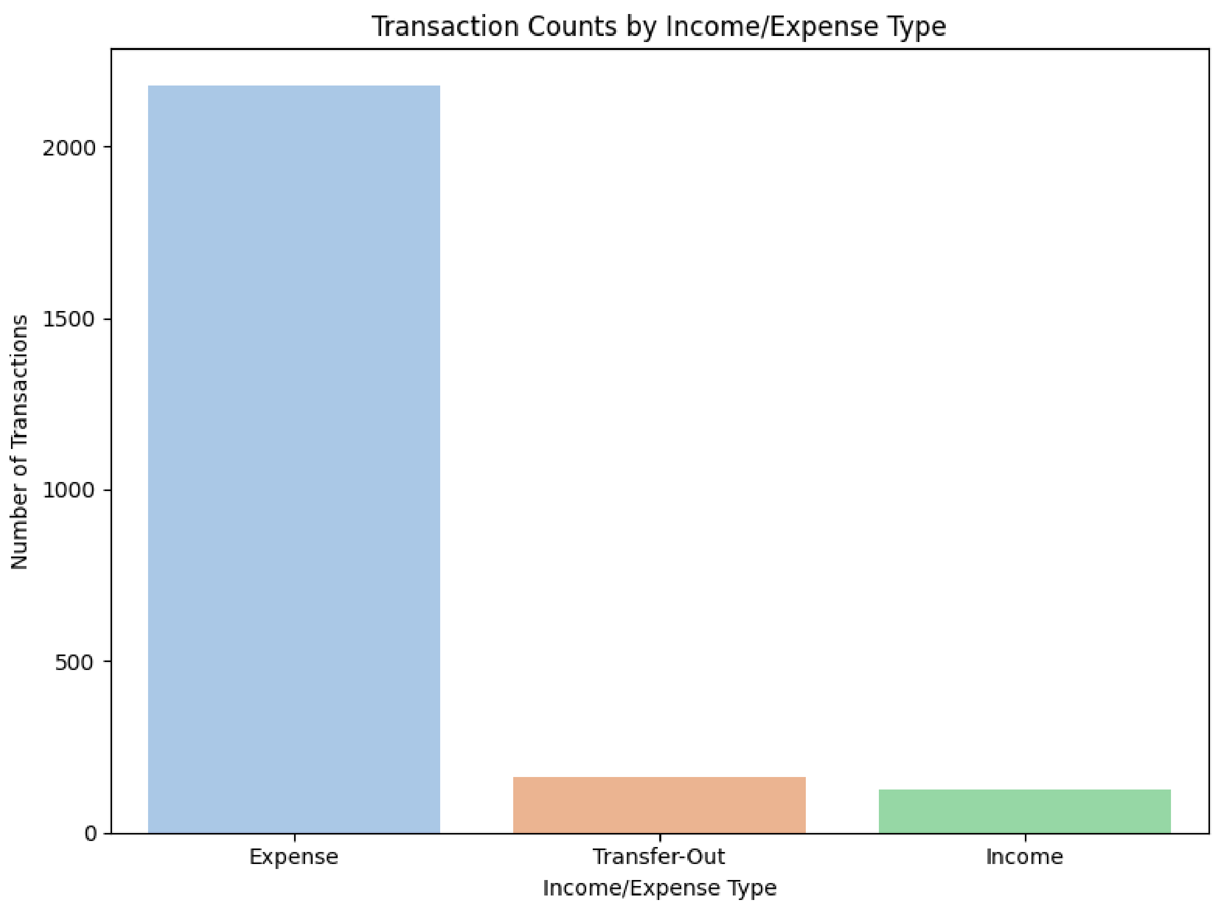
Subtask:

Generate a bar plot to display the distribution of transactions between 'Income' and 'Expense' types, providing insight into the financial flow.

Reasoning: To visualize the distribution of transactions between 'Income' and 'Expense' types, I will calculate the value counts for the 'Income/Expense' column and then generate a bar plot using seaborn, ensuring proper labeling and display as per the instructions.

```
income_expense_counts = df['Income/Expense'].value_counts()

plt.figure(figsize=(8, 6))
sns.barplot(x=income_expense_counts.index, y=income_expense_counts.values, hue=income_expense_co
plt.title('Transaction Counts by Income/Expense Type')
plt.xlabel('Income/Expense Type')
plt.ylabel('Number of Transactions')
plt.tight_layout()
plt.show()
```



Visualize Top 3 Transaction Modes

Subtask:

Create a bar plot to visualize the count of the top 3 transaction 'Mode's to understand the most frequently used payment methods, with the title 'Transaction counts by type'.

Reasoning: To visualize the top 3 transaction modes as requested, I will first get the value counts for the 'Mode' column and select the top 3, then create a bar plot using seaborn, ensuring all specified aesthetic and labeling requirements are met, including the title 'Transaction counts by type'.

```
top_3_modes = df['Mode'].value_counts().head(3)

plt.figure(figsize=(10, 6))
sns.barplot(x=top_3_modes.index, y=top_3_modes.values, hue=top_3_modes.index, palette='crest',
plt.title('Transaction counts by type')
plt.xlabel('Mode')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45, ha='right')
```

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

Transaction counts by type