EXPENSE TRACKER PROJECT

1. Data collection:

Create a Sample Dataset: Simulate expense data that includes:

- Dates: The date of each transaction.
- Amounts: Monetary value of each transaction.
- Categories: Classification of transactions (e.g., food, transportation, utilities).
- Payment Methods: Methods used for transactions (e.g., Cash, Credit Card).
- Any other relevant fields: Additional details such as merchant names and notes.

```
!pip install faker
```

```
Collecting faker
  Downloading Faker-30.8.2-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: python-dateutil>=2.4 in /usr/local/lib/python3.10/dist-packages (from faker) (2.8.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from faker) (4.12.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.4->faker) (1.16.
Downloading Faker-30.8.2-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 15.9 MB/s eta 0:00:00
Installing collected packages: faker
Successfully installed faker-30.8.2
```

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from faker import Faker
import random
```

```
#initialize Fake
fake = Faker()
```

```
# Number of records to generate
num_income_records = 100
num_expense_records = 200
total_records = num_income_records + num_expense_records

# Define possible categories and payment methods
expense_categories = ['Food', 'Shopping', 'Travel', 'Transportation', 'Groceries', 'Bills & Utilities',
                      'Entertainment', 'Health', 'Education', 'Miscellaneous']
income_categories = ['Salary', 'Freelance', 'Investment Returns', 'Gifts', 'Tax Refund', 'Cash Back Rewards', 'From Parents']
payment_methods = ['Cash', 'Credit Card', 'Debit Card', 'Mobile Payment', 'Bank Transfer']


# Define merchants/sources for each category
merchant_dict = {
    # Expense categories
    'Food': ['McDonald\'s', 'KFC', 'Subway', 'Starbucks', 'Pizza Hut', 'IHOP', 'Happy Lemon', 'Panda Express', 'Local restaurant
    'Shopping': ['Outlet mall', 'Target', 'Amazon', 'Best Buy', 'Adidas', 'Nike'],
    'Travel': ['American Airlines', 'United Airlines', 'Delta Airlines', 'Southwest Airlines', 'Continental Hotel', 'Marriott Hc
    'Transportation': ['Uber', 'Lyft', 'Taxi Service', 'Public Transit'],
    'Groceries': ['Walmart', 'Costco', 'Kroger', 'Whole Foods', 'Safeway', 'HEB', 'Indian store', 'Trader Joe\'s', 'Sam\'s Club'
    'Bills & Utilities': ['AT&T', 'Verizon', 'Comcast', 'Electric Company', 'Water Company'],
    'Entertainment': ['Netflix', 'Spotify', 'AMC Theatres', 'Hulu', 'Disney+', 'Cinemark'],
    'Health': ['CVS Pharmacy', 'Walgreens', 'Rite Aid', 'Doctor\'s Office', 'Dental Clinic'],
    'Education': ['Udemy', 'Coursera', 'University Tuition Fees', 'Bookstore', 'Online Course', 'LinkedIn'],
    'Miscellaneous': ['Gift Shop', 'Charity Donation', 'Miscellaneous Purchase', 'Orange Theory Fitness', 'Spa', 'Salon'],

    # Income categories
    'Salary': ['Employer Inc.', 'Company LLC', 'Business Corp.'],
    'Freelance': ['Client A', 'Client B', 'Client C'],
    'Investment Returns': ['Investment Bank', 'Stock Brokerage'],
    'Gifts': ['Family Member', 'Friend', 'Relative'],
    'Tax Refund': ['IRS', 'State Tax Agency'],
    'Cash Back Rewards': ['Credit Card Rewards', 'Bank Rewards Program'],
    'From Parents': ['Mom', 'Dad', 'Parents']
}

# Define notes templates for each category
```

```
notes_dict = {
    # Expense categories
    'Food': ['Lunch at {}', 'Dinner at {}', 'Coffee from {}', 'Breakfast at {}'],
    'Shopping': ['Purchased items from {}', 'Shopping spree at {}', 'Bought clothes at {}'],
    'Travel': ['Flight booked with {}', 'Travel expenses with {}', 'Hotel stay at {}'],
    'Transportation': ['Ride with {}', 'Commute via {}', 'Transport fare for {}'],
    'Groceries': ['Groceries from {}', 'Weekly shopping at {}'],
    'Bills & Utilities': ['Paid bill to {}', 'Utility payment to {}'],
    'Entertainment': ['Subscription to {}', 'Movie night at {}', 'Concert tickets from {}'],
    'Health': ['Medical services at {}', 'Prescription from {}', 'Appointment at {}'],
    'Education': ['Course enrollment at {}', 'Books purchased from {}', 'Tuition fee for {}'],
    'Miscellaneous': ['Donation to {}', 'Miscellaneous expense at {}'],

    # Income categories
    'Salary': ['Monthly salary from {}', 'Paycheck from {}'],
    'Freelance': ['Freelance payment from {}', 'Consulting fee from {}'],
    'Investment Returns': ['Dividend from {}', 'Interest earned from {}'],
    'Gifts': ['Gift received from {}', 'Cash gift from {}'],
    'Tax Refund': ['Tax refund from {}'],
    'Cash Back Rewards': ['Cash back from {}', 'Rewards from {}'],
    'From Parents': ['Money received from {}', 'Allowance from {}']
}

# List to hold generated data
data = []

# Function to introduce missing values
def introduce_missing_value(value, missing_probability):
    return None if random.random() < missing_probability else value

# Set the probability of missing values (5%)
missing_prob = 0.05

# Generate income data
for _ in range(num_income_records):
    category = random.choice(income_categories)
    merchant = random.choice(merchant_dict.get(category, ['Unknown Source']))
    amount = round(random.uniform(100.0, 3000.0), 2)
    payment_method = random.choice(payment_methods)
    date = fake.date_between(start_date='-1y', end_date='today')
    transaction_id = fake.uuid4()
    user_id = fake.uuid4()

    # Generate a meaningful note
    notes_template = random.choice(notes_dict.get(category, ['Income from {}']))
    note = notes_template.format(merchant)

    # Introduce missing values
    merchant = introduce_missing_value(merchant, missing_prob)
    payment_method = introduce_missing_value(payment_method, missing_prob)
    note = introduce_missing_value(note, missing_prob)
    category = introduce_missing_value(category, missing_prob)

    transaction = {
        'Transaction_ID': transaction_id,
        'Date': date,
        'Amount': amount,
        'Category': category,
        'Payment_Method': payment_method,
        'Merchant': merchant,
        'User_ID': user_id,
        'Notes': note,
        'Type': 'Income'
    }

    data.append(transaction)

# Generate expense data
for _ in range(num_expense_records):
    category = random.choice(expense_categories)
    merchant = random.choice(merchant_dict.get(category, ['Unknown Merchant']))
    amount = round(random.uniform(1.0, 800.0), 2)
    payment_method = random.choice(payment_methods)
    date = fake.date_between(start_date='-1y', end_date='today')
    transaction_id = fake.uuid4()
    user_id = fake.uuid4()
    # Generate a meaningful note
    notes_template = random.choice(notes_dict.get(category, ['Expense at {}']))
    note = notes_template.format(merchant)

    # Introduce missing values
    merchant = introduce_missing_value(merchant, missing_prob)
    payment_method = introduce_missing_value(payment_method, missing_prob)
    note = introduce_missing_value(note, missing_prob)
    category = introduce_missing_value(category, missing_prob)

    transaction = {
        'Transaction_ID': transaction_id,
        'Date': date,
        'Amount': amount,
        'Category': category,
        'Payment_Method': payment_method,
        'Merchant': merchant,
        'User_ID': user_id,
        'Notes': note,
        'Type': 'Expense'
    }

    data.append(transaction)


# Create DataFrame
df = pd.DataFrame(data)

#adding duplicates
# Introduce Duplicates
# Decide on the number of duplicates (e.g., 3% of total records)
duplicate_percentage = 0.03  # 3%
num_duplicates = int(total_records * duplicate_percentage)

# Randomly select transactions to duplicate
duplicate_indices = np.random.choice(df.index, size=num_duplicates, replace=False)
duplicate_transactions = df.loc[duplicate_indices]

# Concat duplicates to the DataFrame
df = pd.concat([df, duplicate_transactions], ignore_index=True)

# Shuffle the DataFrame
df = df.sample(frac=1).reset_index(drop=True)

# Ensure that dates are in datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Save to CSV
df.to_csv('expense_data.csv', index=False)
```

2.Clean and Validate Data:

- Check for duplicates, missing values, and outliers.
- Standardize data formats (e.g., date formats, currency).
- Data Quality Assessments

```
df.head(5)
```

	Transaction_ID	Date	Amount	Category	Payment_Method	Merchant	User_ID	Notes	Type
0	a361dcc9-7617-4a43-95d3-14afccd486f9	2024-03-24	366.23	Health	Credit Card	Dental Clinic	28522173-0dea-40fa-a825-cf07602cb9d3	Prescription from Dental Clinic	Expense
1	33022b45-7a78-4051-a2ed-5169e16cda9f	2023-12-14	283.19	Shopping	Credit Card	Nike	3e9c0b4d-224d-41ab-aa05-32ed1b8e4ce9	Bought clothes at Nike	Expense
2	80f712d7-3c61-4e12-82fd-4ce06bba4ade	2024-08-07	1737.24	From Parents	Bank Transfer	Mom	8fd33f36-33fe-4de9-8c69-5fb2e9e0d4d4	Allowance from Mom	Income
3	4a45286c-c9f5-49e8-9ea3-879ed2b5a9ab	2024-07-14	278.61	Bills & Utilities	Debit Card	Comcast	70c88524-da18-44df-83f4-1c64a7550dad	Utility payment to Comcast	Expense
4	daa3d5e8-4df3-4c29-a2eb-4603064f94f2	2024-10-20	656.89	Food	Debit Card	Pizza Hut	33484cc4-7bb5-47ce-aaba-6d45434c7ed0	Dinner at Pizza Hut	Expense

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
#To check total number of records
print(f"Total records: {len(df)}")
```

```
Total records: 309
```

```
#Check non-null value count
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 309 entries, 0 to 308
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Transaction_ID  309 non-null    object
 1   Date            309 non-null    datetime64[ns]
 2   Amount          309 non-null    float64
 3   Category        295 non-null    object
 4   Payment_Method  291 non-null    object
 5   Merchant        282 non-null    object
 6   User_ID         309 non-null    object
 7   Notes           293 non-null    object
 8   Type            309 non-null    object
dtypes: datetime64[ns](1), float64(1), object(7)
memory usage: 21.9+ KB
```

```
#Check data types
df.dtypes
```

	0
Transaction_ID	object
Date	datetime64[ns]
Amount	float64
Category	object
Payment_Method	object
Merchant	object
User_ID	object
Notes	object
Type	object

dtype: object

```
#check duplicates
df.duplicated().sum()
```

```
9
```

```
#drop duplicates
df.drop_duplicates(subset='Transaction_ID', keep='first', inplace=True)
df.duplicated().sum()
```

```
0
```

```
#check unique values in categorical columns
df['Category'].unique()
```

```
array(['Health', 'Shopping', 'From Parents', 'Bills & Utilities', 'Food',
       'Tax Refund', 'Transportation', 'Entertainment',
       'Investment Returns', 'Salary', 'Miscellaneous', 'Groceries',
       'Travel', 'Cash Back Rewards', 'Education', 'Gifts', None,
       'Freelance'], dtype=object)
```

```
df['Payment_Method'].unique()
```

```
array(['Credit Card', 'Bank Transfer', 'Debit Card', 'Mobile Payment',
       'Cash', None], dtype=object)
```

```
#check missing values
missing_values = df.isnull().sum()
missing_values
```

```
                  0
Transaction_ID    0
          Date    0
        Amount    0
      Category   14
Payment_Method   16
      Merchant   25
       User_ID    0
         Notes   15
          Type    0

dtype: int64
```

```
#replace missing values in category
mode_category = df['Category'].mode()[0]
df['Category'].fillna(mode_category, inplace=True)
```

```
<ipython-input-15-9f21a0f89c5b>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through cha
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are

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


  df['Category'].fillna(mode_category, inplace=True)
```

```
#replace payment method
mode_payment = df['Payment_Method'].mode()[0]
df['Payment_Method'].fillna(mode_payment, inplace=True)
```

```
<ipython-input-16-74162b817101>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through cha
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are

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


  df['Payment_Method'].fillna(mode_payment, inplace=True)
```

```
#replace merchant
df['Merchant'].fillna('Unknown Merchant', inplace=True)
```

```
<ipython-input-17-cf987c620bcf>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through cha
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are

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


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

```
#replace notes
df['Notes'].fillna('No notes available', inplace=True)
```

```
<ipython-input-18-adeff01b1f3f>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through cha
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are

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


  df['Notes'].fillna('No notes available', inplace=True)
```

```
#validate
df.isnull().sum()
```

	0
Transaction_ID	0
Date	0
Amount	0
Category	0
Payment_Method	0
Merchant	0
User_ID	0
Notes	0
Type	0

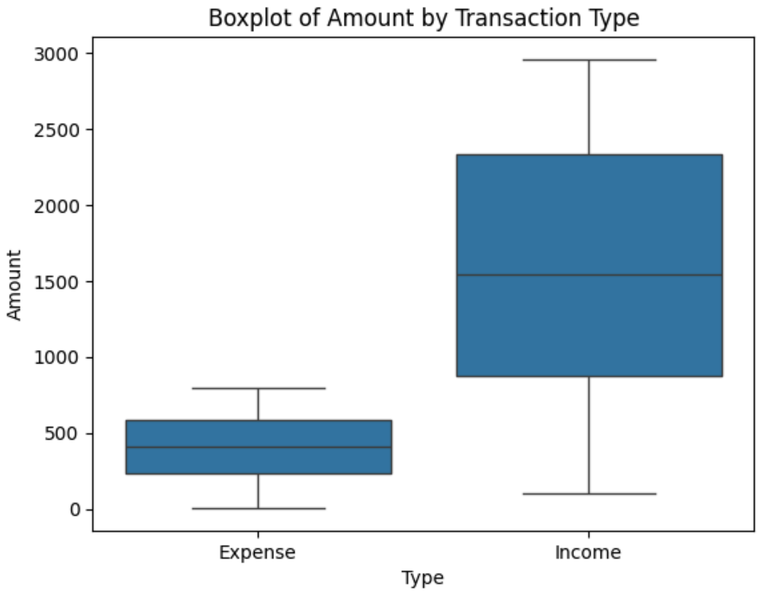
dtype: int64

```
#Data Transformation

#Extract 'Year' and 'Month' from 'Date'

df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Month_Name'] = df['Date'].dt.month_name()
```

```
#ordical encoding

# Define the chronological order of months
months_order = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December']

# Create a mapping dictionary for ordinal encoding
month_map = {month: index for index, month in enumerate(months_order, start=1)}

# Perform ordinal encoding on the 'Month' column
df['Month_Num'] = df['Month'].map(month_map)

# Verify the encoding
print(f"After Ordinal Encoding:\n{df[['Month']].head()}\n")
```

```
After Ordinal Encoding:
   Month
0      3
1     12
2      8
3      7
4     10
```

```
#Outlier Detection and Handling
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(x='Type', y='Amount', data=df)
plt.title('Boxplot of Amount by Transaction Type')
plt.show()
```



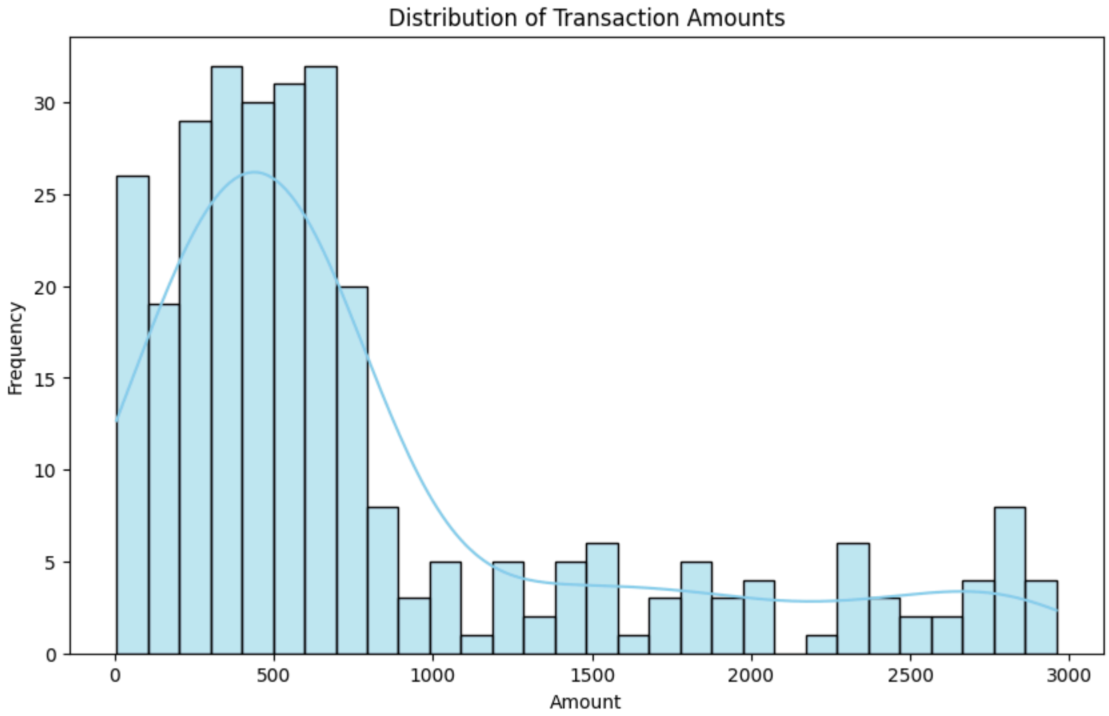
```
# For Expenses
expense_amounts = df[df['Type'] == 'Expense']['Amount']
Q1_expense = expense_amounts.quantile(0.25)
Q3_expense = expense_amounts.quantile(0.75)
IQR_expense = Q3_expense - Q1_expense
IQR_expense
```

```
354.6225
```

```
# For Income
income_amounts = df[df['Type'] == 'Income']['Amount']
Q1_income = income_amounts.quantile(0.25)
Q3_income = income_amounts.quantile(0.75)
IQR_income = Q3_income - Q1_income
IQR_income
```

```
1460.1350000000002
```

```
#to check skewness
# Plot histogram with Kernel Density Estimate
plt.figure(figsize=(10,6))
sns.histplot(df['Amount'], bins=30, kde=True, color='skyblue')
plt.title('Distribution of Transaction Amounts')
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.show()
```



Interpretation for skewness:

Skewness > 0: Right-skewed distribution.

Skewness < 0: Left-skewed distribution.

Skewness ≈ 0: Symmetrical distribution.

```
skewness = df['Amount'].skew()
print(f'Skewness of transaction amounts: {skewness}')
```

```
Skewness of transaction amounts: 1.4875661477454254
```

3. Data Analysis:

1.Descriptive Statistics-Generate summary statistics for numerical and categorical variables (mean, median, mode, variance).

2.Univariate Analysis-Analyze individual variables to understand their distributions (Identify spending trends over time).

3.Bivariate Analysis-Explore relationships between pairs of variables.

4.Multivariate Analysis-Examine interactions among multiple variables.

```
# 1.Descriptive Statistics

#total number of transactions
total_transactions = len(df)
total_transactions
```

```
300
```

```
#date range of transactions
start_date = df['Date'].min()
end_date = df['Date'].max()
print(f"Date Range: {start_date} to {end_date}")
```

```
Date Range: 2023-11-11 00:00:00 to 2024-11-10 00:00:00
```

```
#total amount of transactions

total_income = df[df['Type'] == 'Income']['Amount'].sum()
total_expenses = df[df['Type'] == 'Expense']['Amount'].sum()
net_income = total_income + total_expenses  # Expenses are negative
print(f"Total Income: ${total_income:.2f}")
print(f"Total Expenses: ${total_expenses:.2f}")
print(f"Net Income: ${net_income:.2f}")
```

```
Total Income: $157963.12
Total Expenses: $81055.25
Net Income: $239018.37
```

```
#average amount of transactions
average_amount = df['Amount'].mean()
average_amount
```

```
796.7279
```

```
avg_income = df[df['Type'] == 'Income']['Amount'].mean()
avg_expenses = df[df['Type'] == 'Expense']['Amount'].mean()
print(f"avg_income: ${avg_income:.2f}")
print(f"avg_expenses: ${avg_expenses:.2f}")
```

```
avg_income: $1579.63
avg_expenses: $405.28
```

```
#highest transactions

max_amount = df['Amount'].max()
max_amount
```

```
2958.9
```

```
#lowest transactions
min_amount = df['Amount'].min()
min_amount
```

```
5.55
```

```
#Highest income and expense
highest_income = df[df['Type'] == 'Income']['Amount'].max()
highest_expense = df[df['Type'] == 'Expense']['Amount'].max()
print(f"Highest Income: ${highest_income:.2f}")
print(f"Highest Expense: ${highest_expense:.2f}")
```

```
Highest Income: $2958.90
Highest Expense: $797.36
```

```
#Lowest income and expense
lowest_income = df[df['Type'] == 'Income']['Amount'].min()
lowest_expense = df[df['Type'] == 'Expense']['Amount'].min()
print(f"lowest_income: ${lowest_income:.2f}")
print(f"lowest_expense: ${lowest_expense:.2f}")
```

```
lowest_income: $100.96
lowest_expense: $5.55
```

```
category_stats = df.groupby('Category')['Amount'].agg(['sum', 'mean', 'count'])
category_stats
```

Category	sum	mean	count
Bills & Utilities	4114.23	293.873571	14
Cash Back Rewards	33993.85	1699.692500	20
Education	8946.54	426.025714	21
Entertainment	6935.48	385.304444	18
Food	7212.00	450.750000	16
Freelance	14968.84	1663.204444	9
From Parents	21418.17	1529.869286	14
Gifts	17458.38	1342.952308	13
Groceries	9536.15	414.615217	23
Health	5830.04	364.377500	16
Investment Returns	25262.46	1403.470000	18
Miscellaneous	9064.68	431.651429	21
Salary	24052.82	1603.521333	15
Shopping	3998.15	285.582143	14
Tax Refund	18091.67	2010.185556	9
Transportation	8378.02	440.948421	19
Travel	19756.89	493.922250	40

Next steps: Generate code with category_stats | View recommended plots | New interactive sheet

```
expense_stats = df.groupby('Type')['Amount'].agg(['sum', 'mean', 'median','count'])
expense_stats
```

Type	sum	mean	median	count
Expense	81055.25	405.27625	407.955	200
Income	157963.12	1579.63120	1543.120	100

Next steps: Generate code with expense_stats | View recommended plots | New interactive sheet

```
#Numerical Variables

#Summary Statistics

df['Amount'].describe()
```

	Amount
count	300.000000
mean	796.727900
std	756.245511
min	5.550000
25%	311.000000
50%	561.355000
75%	857.445000
max	2958.900000

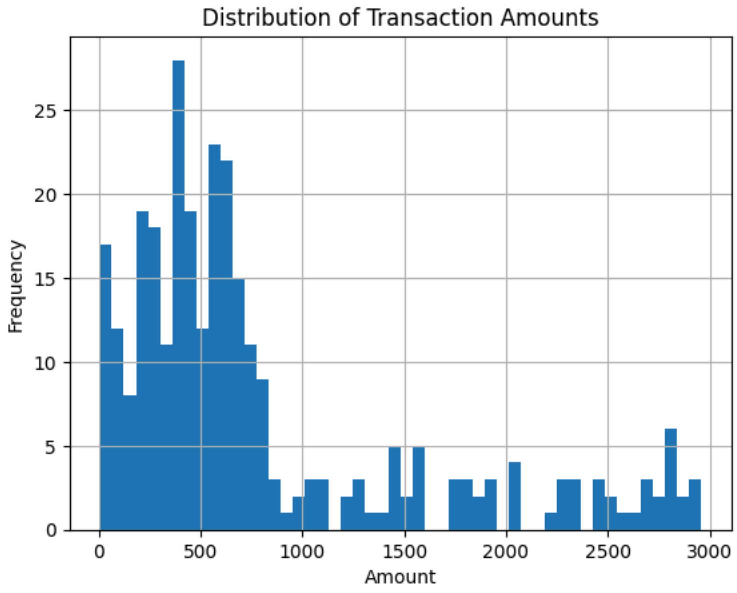
dtype: float64

```
#Categorical Variables

#Frequency Counts
```

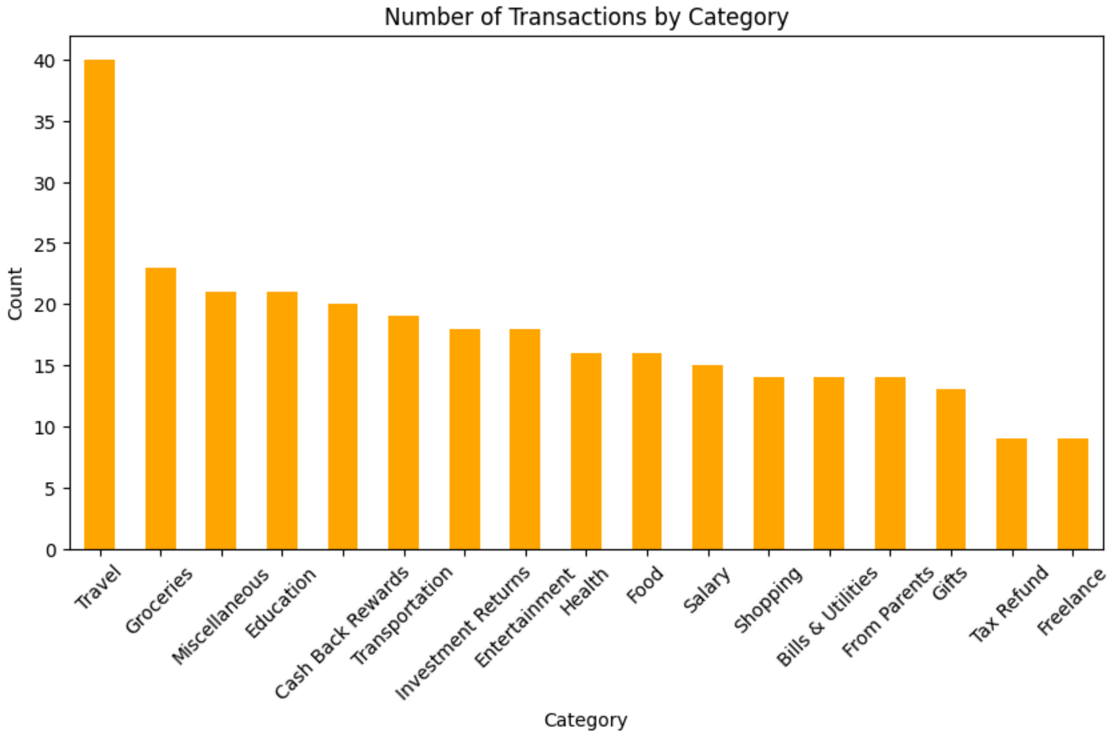
```
df['Category'].value_counts()
```

```
                         count
          Category
            Travel          40
         Groceries          23
     Miscellaneous          21
         Education          21
 Cash Back Rewards          20
    Transportation          19
Investment Returns          18
     Entertainment          18
            Health          16
              Food          16
            Salary          15
          Shopping          14
  Bills & Utilities          14
      From Parents          14
             Gifts          13
        Tax Refund           9
         Freelance           9

dtype: int64
```

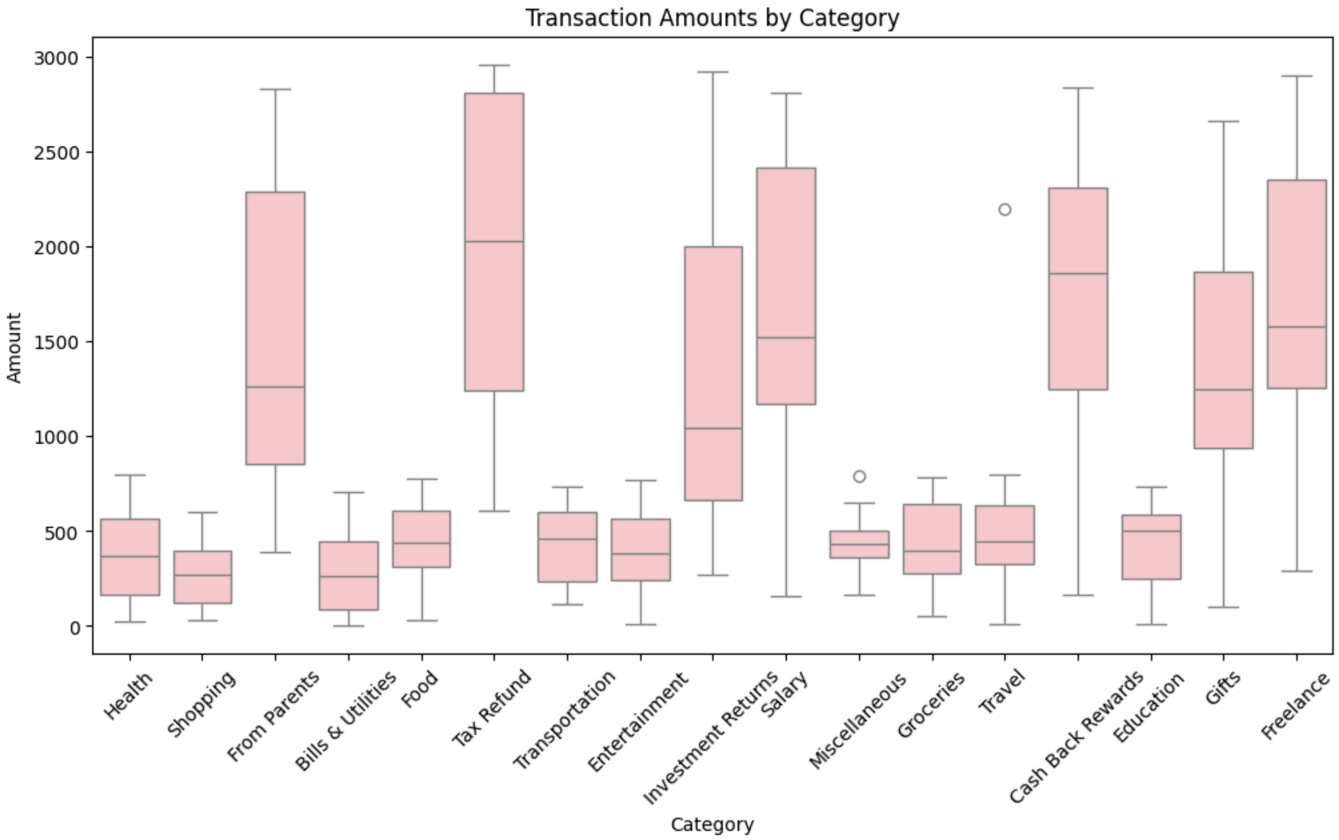
```
#2. Univariate Analysis
#Histograms

#Amount Distribution

df['Amount'].hist(bins=50)
plt.title('Distribution of Transaction Amounts')
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.show()
```



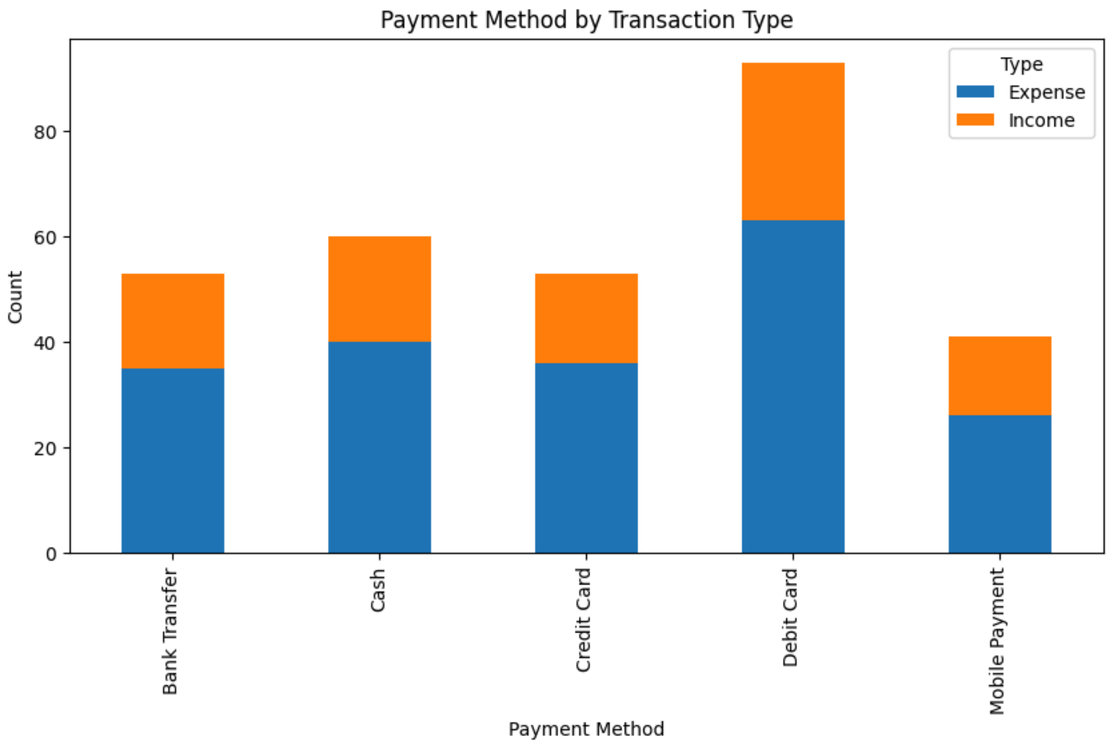
```
#transactions by category
df['Category'].value_counts().plot(kind='bar', figsize=(10,5), color='Orange')
plt.title('Number of Transactions by Category')
plt.xlabel('Category')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



```
# Bivariate Analysis
#a. Boxplots

plt.figure(figsize=(12,6))
sns.boxplot(x='Category', y='Amount', data=df, color='pink')
plt.title('Transaction Amounts by Category')
plt.xlabel('Category')
plt.ylabel('Amount')
plt.xticks(rotation=45)
plt.show()
```



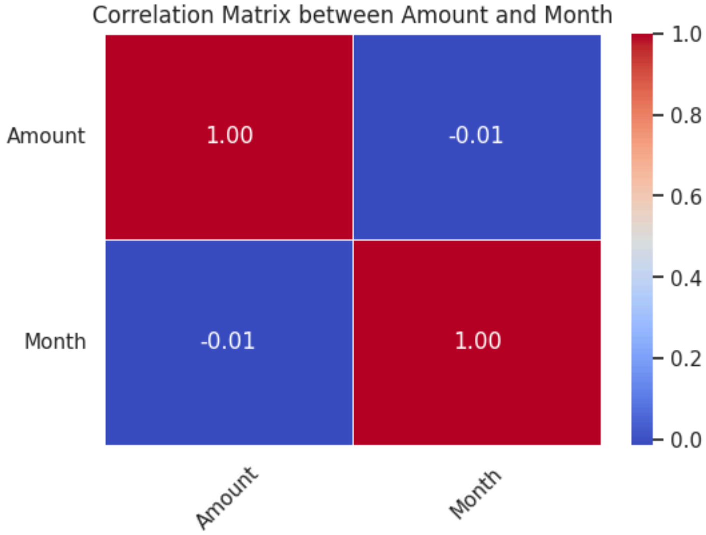
```
#b. Grouped Bar Charts

#Payment Method by Type

payment_type = df.groupby(['Payment_Method', 'Type']).size().unstack()
payment_type.plot(kind='bar', stacked=True, figsize=(10,5))
plt.title('Payment Method by Transaction Type')
plt.xlabel('Payment Method')
plt.ylabel('Count')
plt.show()
```



```
# Compute the correlation matrix for 'Amount' and 'Month_Num'
corr_matrix = df[['Amount', 'Month']].corr()

# Display the correlation matrix
print(f"Correlation Matrix:\n{corr_matrix}\n")
```

```
Correlation Matrix:
          Amount     Month
Amount  1.000000 -0.014166
Month  -0.014166  1.000000
```

```
#3. Multivariate Analysis
#a. Heatmaps

#Correlation Matrix

sns.set(style="whitegrid")

plt.figure(figsize=(6, 4))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)

plt.title('Correlation Matrix between Amount and Month')
plt.xticks(rotation=45)
plt.yticks(rotation=0)

plt.show()
```



```
 #Pivot Tables

#Monthly Income and Expenses


monthly_data = df.pivot_table(values='Amount', index='Month_Name', columns='Type', aggfunc='sum')
monthly_data = monthly_data.reindex(index=['January', 'February', 'March', 'April', 'May', 'June',
                                           'July', 'August', 'September', 'October', 'November', 'December'])
monthly_data
```

Type	Expense	Income
Month_Name		
January	5187.19	16133.12
February	7443.62	5972.49
March	6100.78	13811.13
April	5532.32	10327.79
May	5534.53	22918.01
June	7356.17	11444.96
July	10821.66	9790.20
August	6638.57	16502.56
September	5726.86	17700.11
October	7686.77	6558.35
November	5687.29	13137.64
December	7339.49	13666.76

Next steps: Generate code with monthly_data | View recommended plots | New interactive sheet

```
#Hypothesis Testing
#whether the mean transaction amount differs between two categories

from scipy.stats import ttest_ind

# Extract amounts for two categories
category_a = df[df['Category'] == 'Food']['Amount']
category_b = df[df['Category'] == 'Shopping']['Amount']

# Perform T-test
t_stat, p_value = ttest_ind(category_a, category_b, equal_var=False)

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")
```

```
T-statistic: 2.3057
P-value: 0.0288
```

4. Data Visualization Create Interactive Dashboards:

Use tools like Tableau, Power BI, or Python libraries (Matplotlib, Seaborn, Plotly). Dashboards should include: Monthly spending summaries Category-wise expenditure breakdowns Trends over time Visual Reports:

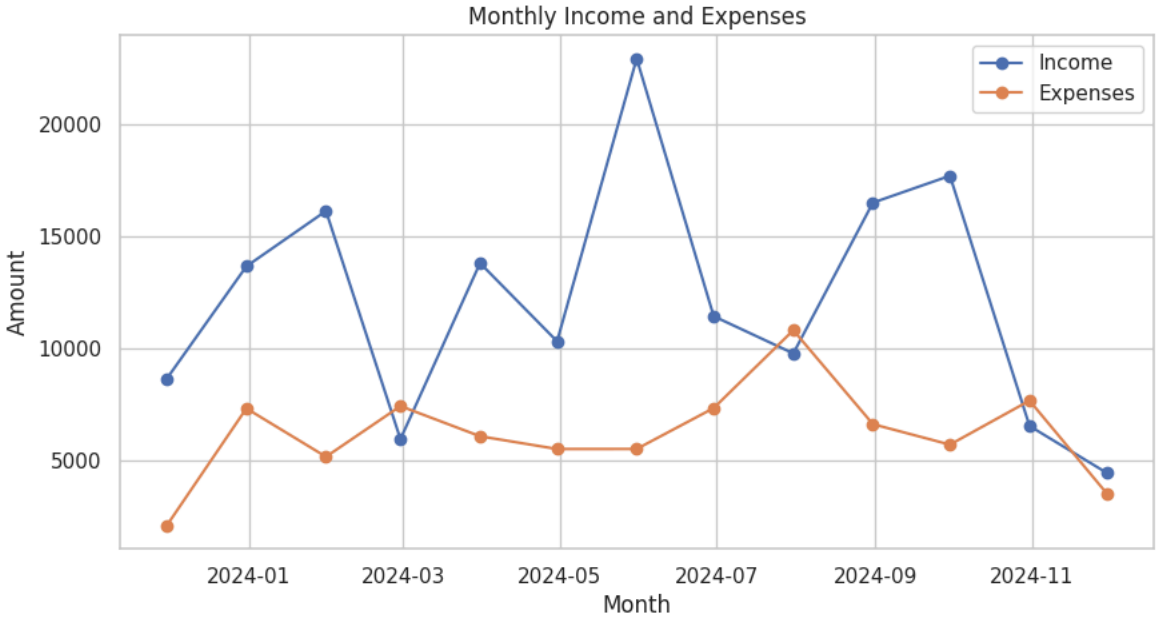
Deliverable: Data Visualization Reports with screenshots and explanations. Performance Metrics:

Define Key Performance Indicators (KPIs). Deliverable: Performance Metrics Reports Business Intelligence Presentation:

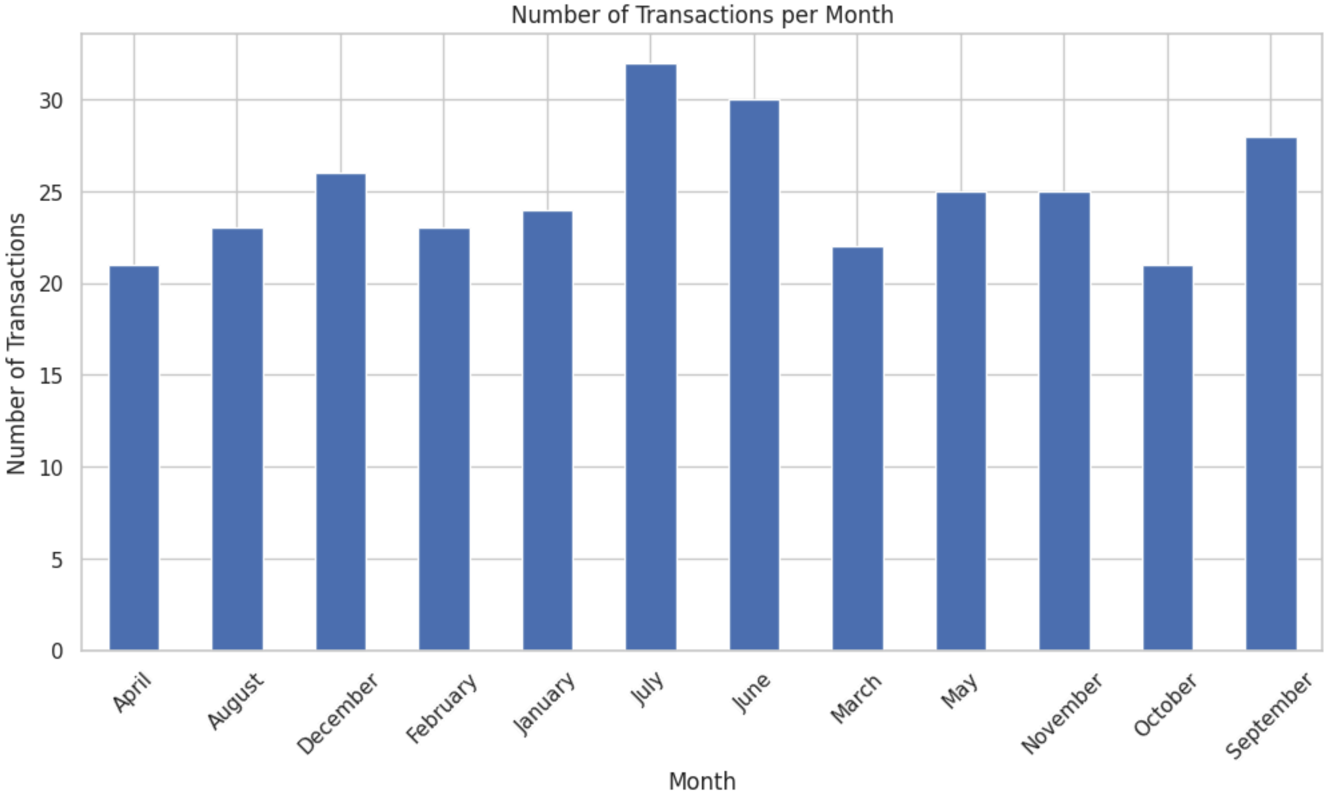
Prepare a slide deck summarizing insights. Deliverable: Business Intelligence Presentations

```
#a. Time Series Plots

#Monthly Income and Expenses

df.set_index('Date', inplace=True)
monthly_income = df[df['Type'] == 'Income']['Amount'].resample('M').sum()
monthly_expense = df[df['Type'] == 'Expense']['Amount'].resample('M').sum()

plt.figure(figsize=(10,5))
plt.plot(monthly_income.index, monthly_income.values, label='Income', marker='o')
plt.plot(monthly_expense.index, monthly_expense.values, label='Expenses', marker='o')
plt.title('Monthly Income and Expenses')
plt.xlabel('Month')
plt.ylabel('Amount')
plt.legend()
plt.show()
```

```
<ipython-input-48-e5aa2ced7c63>:6: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME'
  monthly_income = df[df['Type'] == 'Income']['Amount'].resample('M').sum()
<ipython-input-48-e5aa2ced7c63>:7: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME'
  monthly_expense = df[df['Type'] == 'Expense']['Amount'].resample('M').sum()
```



```
transactions_per_month = df.groupby('Month_Name').size()
transactions_per_month.plot(kind='bar', figsize=(12, 6))
plt.title('Number of Transactions per Month')
plt.xlabel('Month')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45)
plt.show()
```



```
#b.Pie Charts

#Expense Distribution by Category


expense_distribution = df[df['Type'] == 'Expense']['Category'].value_counts()
expense_distribution.plot(kind='pie', autopct='%1.1f%%', figsize=(8,8))
plt.title('Expense Distribution by Category')
plt.ylabel('')
plt.show()
```

Expense Distribution by Category

```
# Plot monthly expenses
# Filter expenses
expenses = df[df['Type'] == 'Expense']
```