💻 Walmart Business Case Study

- Downloaded the file from the specified Google Drive link and saves it as walmart_data.csv

```
!wget  "https://drive.google.com/uc?export=download&id=1csPtwyE-PCWI6cw4h1oXa81Tz1f3FA5D"  -O walmart_data.csv
```

```
--2025-01-15 06:38:04--  https://drive.google.com/uc?export=download&id=1csPtwyE-PCWI6cw4h1oXa81Tz1f3FA5D
Resolving drive.google.com (drive.google.com)... 142.250.99.113, 142.250.99.138, 142.250.99.100, ...
Connecting to drive.google.com (drive.google.com)|142.250.99.113|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1csPtwyE-PCWI6cw4h1oXa81Tz1f3FA5D&export=download [following]
--2025-01-15 06:38:04--  https://drive.usercontent.google.com/download?id=1csPtwyE-PCWI6cw4h1oXa81Tz1f3FA5D&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 74.125.197.132, 2607:f8b0:400e:c03::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|74.125.197.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23027994 (22M) [application/octet-stream]
Saving to: ‘walmart_data.csv’

walmart_data.csv    100%[===================>]  21.96M  88.4MB/s    in 0.2s

2025-01-15 06:38:07 (88.4 MB/s) - ‘walmart_data.csv’ saved [23027994/23027994]
```

1. Introduction

❓ What is Walmart Business Case Study?

- Walmart is an American multinational retail corporation that operates a chain of supercenters, discount departmental stores, and grocery stores from the United States. Walmart has more than 100 million customers worldwide.

🎯 Objective:

- The Management team at Walmart Inc. wants to analyze the customer purchase behavior (specifically, purchase amount) against the customer's gender and the various other factors to help the business make better decisions. They want to understand if the spending habits differ between male and female customers: Do women spend more on Black Friday than men? (Assume 50 million customers are male and 50 million are female).

🛍️ About Data:

- The company collected the transactional data of customers who purchased products from the Walmart Stores during Black Friday.

📄 Features of the dataset:

- User_ID: User ID
- Product_ID: Product ID
- Gender: Sex of User
- Age: Age in bins
- Occupation: Occupation(Masked)
- City_Category: Category of the City (A,B,C)
- StayInCurrentCityYears: Number of years stay in current city
- Marital_Status: Marital Status
- ProductCategory: Product Category (Masked)
- Purchase: Purchase Amount

2.Exploratory Data Analysis

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import warnings
warnings.filterwarnings('ignore')
from scipy import stats
#import copy
#from wordcloud import WordCloud
```

```
# loading the dataset
df = pd.read_csv('walmart_data.csv')
```

```
#to view full data
df
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	P
0	1000001	P00069042	F	0-17	10	A	2	0	3	
1	1000001	P00248942	F	0-17	10	A	2	0	1	
2	1000001	P00087842	F	0-17	10	A	2	0	12	
3	1000001	P00085442	F	0-17	10	A	2	0	12	
4	1000002	P00285442	M	55+	16	C	4+	0	8	
...	...	...	...	...	...	...	...	...	...	
550063	1006033	P00372445	M	51-55	13	B	1	1	20	
550064	1006035	P00375436	F	26-35	1	C	3	0	20	

```
#to view columns

df.columns
#df.keys()== df.columns
```

```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category',
       'Purchase'],
      dtype='object')
```

```
#view first 5 rows/records
df.head(5)
#view first 5 rows/records default=5
#df.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purcha
0	1000001	P00069042	F	0-17	10	A	2	0	3	83
1	1000001	P00248942	F	0-17	10	A	2	0	1	152
2	1000001	P00087842	F	0-17	10	A	2	0	12	14

```
#view last 5 rows/records,deafault=5
df.tail()
#view last 5 rows/records
#df.tail(5)
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	P
550063	1006033	P00372445	M	51-55	13	B	1	1	20	
550064	1006035	P00375436	F	26-35	1	C	3	0	20	
550065	1006036	P00375436	F	26-35	15	B	4+	1	20	

```
#To get index of dataframe
df.index
```

```
RangeIndex(start=0, stop=550068, step=1)
```

```
#To get shape information

df.shape

#550068 rows and 10 columns
```

```
(550068, 10)
```

```
# to get dimensional detail of dataframe

df.ndim

#2D
```

```
2
```

```
#Datatype

print(df.dtypes)
```

```
User_ID                      int64
Product_ID                  object
Gender                      object
Age                         object
Occupation                   int64
City_Category               object
Stay_In_Current_City_Years  object
Marital_Status               int64
Product_Category             int64
Purchase                     int64
dtype: object
```

```
# Conversion of categorical attributes to 'category' (if required)
for column in df.columns:
    if df[column].dtype == 'object':
        df[column] = df[column].astype('category')
# Verify the changes
print("Data types after conversion:\n",df.dtypes)
```

```
Data types after conversion:
 User_ID                        int64
Product_ID                  category
Gender                      category
Age                         category
Occupation                     int64
City_Category               category
Stay_In_Current_City_Years  category
Marital_Status                 int64
Product_Category               int64
Purchase                       int64
dtype: object
```

```
# to get complete information of each column of dataframe like counts,datatype,memory usage.
#Note: For missing value in each column data type will be object

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 10 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   User_ID                     550068 non-null  int64
 1   Product_ID                  550068 non-null  category
 2   Gender                      550068 non-null  category
 3   Age                         550068 non-null  category
 4   Occupation                  550068 non-null  int64
 5   City_Category               550068 non-null  category
 6   Stay_In_Current_City_Years  550068 non-null  category
 7   Marital_Status              550068 non-null  int64
 8   Product_Category            550068 non-null  int64
 9   Purchase                    550068 non-null  int64
dtypes: category(5), int64(5)
memory usage: 24.3 MB
```

🔍 Insights

From the above details it is clear that given dataframe is of dimension 2D with 550068 rows and 10 columns.

Also we can also observe that there are no missing values for any columns .

📝 Statistical Summary

```
#for column with datatype as int, df.describe() will give statistical information like count,mean,min,max,std detail for that column.
```

```
df.describe()
```

	User_ID	Occupation	Marital_Status	Product_Category	Purchase
count	5.500680e+05	550068.000000	550068.000000	550068.000000	550068.000000
mean	1.003029e+06	8.076707	0.409653	5.404270	9263.968713
std	1.727592e+03	6.522660	0.491770	3.936211	5023.065394
min	1.000001e+06	0.000000	0.000000	1.000000	12.000000
25%	1.001516e+06	2.000000	0.000000	1.000000	5823.000000
50%	1.003077e+06	7.000000	0.000000	5.000000	8047.000000
75%	1.004478e+06	14.000000	1.000000	8.000000	12054.000000
max	1.006040e+06	20.000000	1.000000	20.000000	23961.000000

```
#Statistical Summary: Generate a statistical summary of the dataset.

df.describe(include='all')
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_C
count	5.500680e+05	550068	550068	550068	550068.000000	550068	550068	550068.000000	55006
unique	NaN	3631	2	7	NaN	3	5	NaN	
top	NaN	P00265242	M	26-35	NaN	B	1	NaN	
freq	NaN	1880	414259	219587	NaN	231173	193821	NaN	
mean	1.003029e+06	NaN	NaN	NaN	8.076707	NaN	NaN	0.409653	
std	1.727592e+03	NaN	NaN	NaN	6.522660	NaN	NaN	0.491770	
min	1.000001e+06	NaN	NaN	NaN	0.000000	NaN	NaN	0.000000	
25%	1.001516e+06	NaN	NaN	NaN	2.000000	NaN	NaN	0.000000	
50%	1.003077e+06	NaN	NaN	NaN	7.000000	NaN	NaN	0.000000	
75%	1.004478e+06	NaN	NaN	NaN	14.000000	NaN	NaN	1.000000	
max	1.006040e+06	NaN	NaN	NaN	20.000000	NaN	NaN	1.000000	2

🔍 Insights:

1) Product Popularity:

- The most popular product is P00265242, with 1,880 occurrences out of 550,068.

2) Age Distribution:

- The most common age group is 26-35, with 219,587 occurrences.
- There are total 7 unique age groups.

3) Gender:

- The dataset has more male users (414,259 out of 550,068).

4) Occupation:

- The average occupation level is around 8.08, with a range from 0 to 20.

5) City Category:

- The most common city category is B, with 231,173 occurrences.

6) Stay In Current City Years:

- The most common stay duration in the current city is 1 year (193,821 out of 550,068).

7) Marital Status:

- The majority of users are with a range of 0 and 1 ; but can treat 0 for not married and 1 for married in future, if required.

8) Product Category:

- The average product category is around 5.40, with a range from 1 to 20.

9) Purchase:

- The average purchase amount is approximately ₹9,263.97, with a range from ₹12 to ₹23,961.

10) User_Id:

- There are 5891 unique values.

Key Observations:

- User_ID and Product_ID are unique identifiers.
- Gender and City_Category have a clear mode, indicating the most common category.
- Age and Stay_In_Current_City_Years are categorical with specific groups.
- Occupation and Marital_Status are numerical but can be treated as categorical for some analyses.
- Purchase shows a wide range of values, indicating variability in purchase amounts.

👥 Duplicate Detection

```
df.duplicated().value_counts()
```

	count
False	550068

dtype: int64

🔍 Insights

- There are no duplicate entries in the dataset

🕵️ Missing Value Analysis

```
missing_values=df.isnull().sum()
missing_values
```

	0
User_ID	0
Product_ID	0
Gender	0
Age	0
Occupation	0
City_Category	0
Stay_In_Current_City_Years	0
Marital_Status	0
Product_Category	0
Purchase	0

dtype: int64

🔍 Insights

- There is no missing values for all columns.

📍 **For Non-graphical Analysis:**

✅ Sanity Check for columns

```
# checking the unique values for columns
for i in df.columns:
  print('Unique Values in',i,'column are :-')
  print(df[i].unique())
  print('-'*70)
```

```
Unique Values in User_ID column are :-
[1000001 1000002 1000003 ... 1004113 1005391 1001529]
----------------------------------------------------------------------
Unique Values in Product_ID column are :-
['P00069042', 'P00248942', 'P00087842', 'P00085442', 'P00285442', ..., 'P00375436', 'P00372445', 'P00370293', 'P00371644', 'P003708
```

```
Length: 3631
Categories (3631, object): ['P00000142', 'P00000242', 'P00000342', 'P00000442', ..., 'P0099642',
                            'P0099742', 'P0099842', 'P0099942']
------------------------------------------------------------------------
Unique Values in Gender column are :-
['F', 'M']
Categories (2, object): ['F', 'M']
------------------------------------------------------------------------
Unique Values in Age column are :-
['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25']
Categories (7, object): ['0-17', '18-25', '26-35', '36-45', '46-50', '51-55', '55+']
------------------------------------------------------------------------
Unique Values in Occupation column are :-
[10 16 15  7 20  9  1 12 17  0  3  4 11  8 19  2 18  5 14 13  6]
------------------------------------------------------------------------
Unique Values in City_Category column are :-
['A', 'C', 'B']
Categories (3, object): ['A', 'B', 'C']
------------------------------------------------------------------------
Unique Values in Stay_In_Current_City_Years column are :-
['2', '4+', '3', '1', '0']
Categories (5, object): ['0', '1', '2', '3', '4+']
------------------------------------------------------------------------
Unique Values in Marital_Status column are :-
[0 1]
------------------------------------------------------------------------
Unique Values in Product_Category column are :-
[ 3  1 12  8  5  4  2  6 14 11 13 15  7 16 18 10 17  9 20 19]
------------------------------------------------------------------------
Unique Values in Purchase column are :-
[ 8370 15200  1422 ...   135   123   613]
------------------------------------------------------------------------
```

```
# to understand the diversity of data in each specified column.
for i in df.columns:
    print('Unique Values in',i,'column are :-')
    print(df[i].nunique())
    print('-'*70)
```

```
Unique Values in User_ID column are :-
5891
------------------------------------------------------------------------
Unique Values in Product_ID column are :-
3631
------------------------------------------------------------------------
Unique Values in Gender column are :-
2
------------------------------------------------------------------------
Unique Values in Age column are :-
7
------------------------------------------------------------------------
Unique Values in Occupation column are :-
21
------------------------------------------------------------------------
Unique Values in City_Category column are :-
3
------------------------------------------------------------------------
Unique Values in Stay_In_Current_City_Years column are :-
5
------------------------------------------------------------------------
Unique Values in Marital_Status column are :-
2
------------------------------------------------------------------------
Unique Values in Product_Category column are :-
20
------------------------------------------------------------------------
Unique Values in Purchase column are :-
18105
------------------------------------------------------------------------
```

```
for i in df.columns:
  print('Value count in',i,'column are :-')
  print(df[i].value_counts())
  print('-'*70)
```

```
Name: count, dtype: int64
----------------------------------------------------------------------
Value count in Marital_Status column are :-
Marital_Status
0    324731
1    225337
Name: count, dtype: int64
----------------------------------------------------------------------
Value count in Product_Category column are :-
Product_Category
5     150933
1     140378
8     113925
11     24287
2      23864
6      20466
3      20213
4      11753
16      9828
15      6290
13      5549
10      5125
12      3947
7       3721
18      3125
20      2550
19      1603
14      1523
17       578
9        410
Name: count, dtype: int64
----------------------------------------------------------------------
Value count in Purchase column are :-
Purchase
7011     191
7193     188
6855     187
6891     184
7012     183
        ...
23491      1
18345      1
3372       1
855        1
21489      1
Name: count, Length: 18105, dtype: int64
----------------------------------------------------------------------
```

🔍 Insights:

1) User_ID:

- There are 5,891 unique user IDs, indicating the number of distinct users in the dataset.

2) Product_ID:

- There are 3,631 unique product IDs, representing the variety of products available.

3) Gender:

- There are 2 unique values for gender, indicating the dataset includes both male and female users.

4) Age:

- There are 7 unique age groups, showing the dataset categorizes users into different age ranges.

5) Occupation:

- There are 21 unique occupation codes, reflecting the diversity of users' occupations.

6) City_Category:

- There are 3 unique city categories (A, B, C), indicating the dataset includes users from different types of cities.

7) Stay_In_Current_City_Years:

- There are 5 unique values for the number of years users have stayed in their current city, showing the dataset tracks users' duration of stay.

8) Marital_Status:

- There are 2 unique values for marital status, indicating whether users are married or not(0 or 1).

9) Product_Category:

- There are 20 unique product categories, representing the different types of products users have purchased.

10) Purchase:

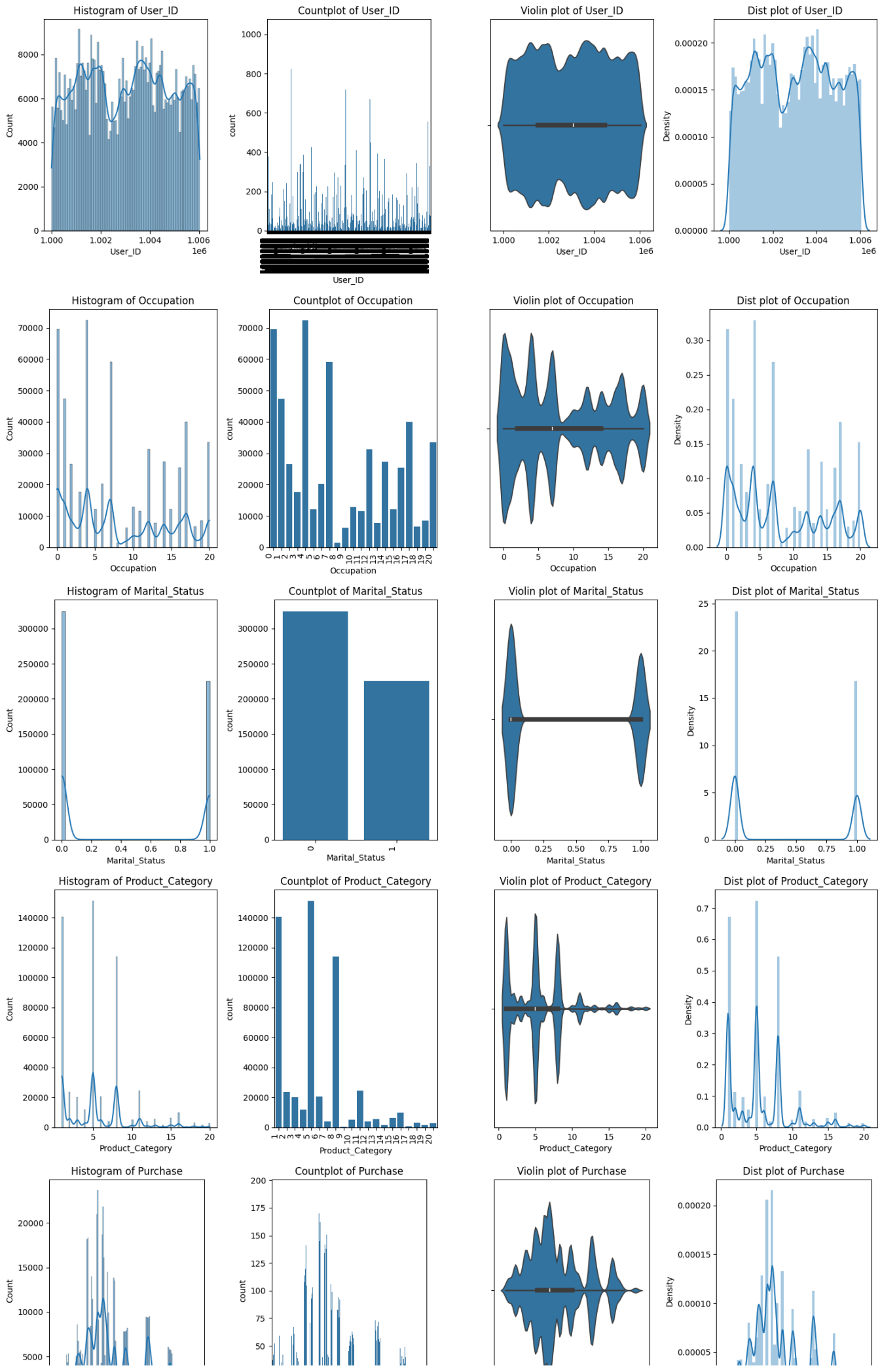
- There are 18,105 unique purchase amounts, reflecting the variability in users' spending.

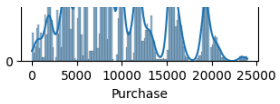
Visual Analysis

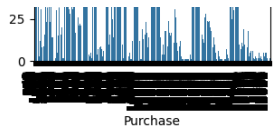
```
df.info()
```

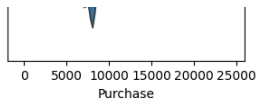
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 10 columns):
 #   Column                      Non-Null Count   Dtype   
---  ------                      --------------   -----   
 0   User_ID                     550068 non-null  int64   
 1   Product_ID                  550068 non-null  category
 2   Gender                      550068 non-null  category
 3   Age                         550068 non-null  category
 4   Occupation                  550068 non-null  int64   
 5   City_Category               550068 non-null  category
 6   Stay_In_Current_City_Years  550068 non-null  category
 7   Marital_Status              550068 non-null  int64   
 8   Product_Category            550068 non-null  int64   
 9   Purchase                    550068 non-null  int64   
dtypes: category(5), int64(5)
memory usage: 24.3 MB
```

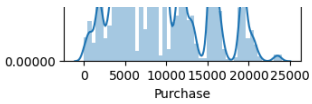
```
#Visual Analysis - For continuous variable(s): Distplot, countplot, histogram for univariate analysis

# List of continuous columns
#df_new=df[['User_ID','Purchase']].value_counts()[:10]


continuous_columns = ['User_ID', 'Occupation', 'Marital_Status', 'Product_Category', 'Purchase']
for column in continuous_columns:
    plt.figure(figsize=(15, 5))
    # Histogram
    plt.subplot(1, 4, 1)
    sns.histplot(df[column].dropna(), kde=True)
    plt.title(f'Histogram of {column}')
    plt.tight_layout()

    # Countplot
    plt.subplot(1, 4, 2)
    sns.countplot(x=df[column].dropna())
    plt.title(f'Countplot of {column}')
    plt.xticks(rotation=90, ha='right') # Rotate x-axis labels
    plt.tight_layout()

    # Violin plot
    plt.subplot(1, 4, 3)
    sns.violinplot(x=df[column].dropna())
    plt.title(f'Violin plot of {column}')
    plt.tight_layout()

    # Dist plot
    plt.subplot(1, 4, 4)
    sns.distplot(df[column].dropna())
    plt.title(f'Dist plot of {column}')
    plt.tight_layout()
    plt.show()
```


Histogram of User_ID
Countplot of User_ID
Violin plot of User_ID
Dist plot of User_ID
Histogram of Occupation
Countplot of Occupation
Violin plot of Occupation
Dist plot of Occupation
Histogram of Marital_Status
Countplot of Marital_Status
Violin plot of Marital_Status
Dist plot of Marital_Status
Histogram of Product_Category
Countplot of Product_Category
Violin plot of Product_Category
Dist plot of Product_Category
Histogram of Purchase
Countplot of Purchase
Violin plot of Purchase
Dist plot of Purchase
User_ID
Occupation
Marital_Status
Product_Category
Count
count
Density
1e6





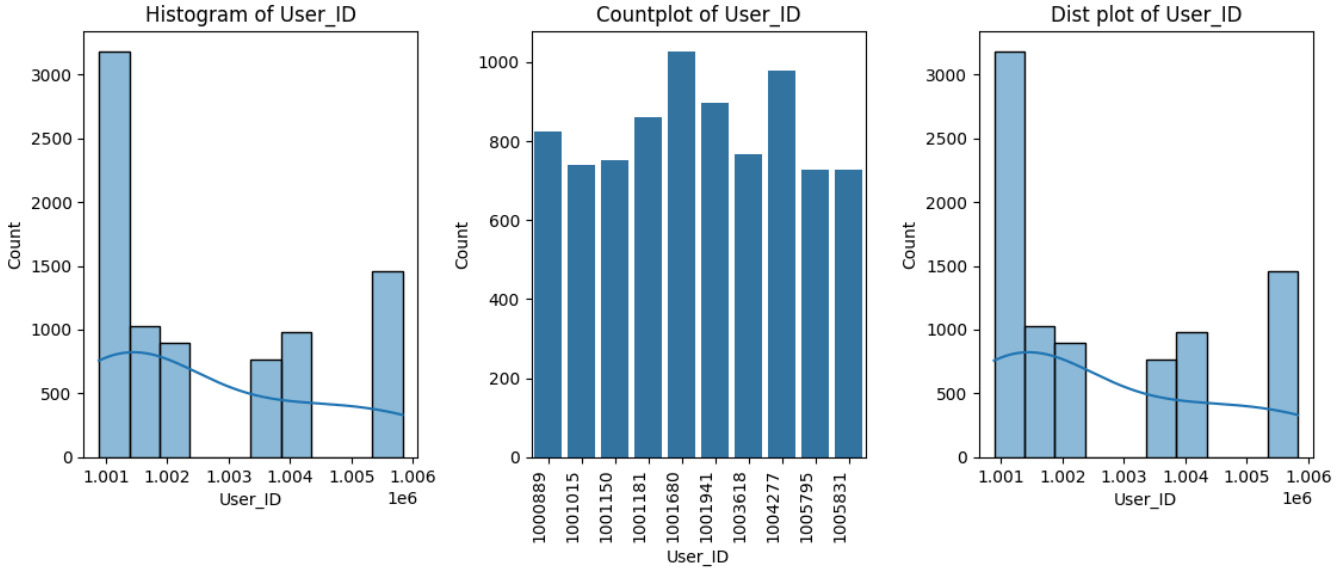





```
# Get the top 10 user_id and purchase counts
df_new_u = df['User_ID'].value_counts().sort_values(ascending=False).head(10).reset_index()
df_new_p = df['Purchase'].value_counts().sort_values(ascending=False).head(10).reset_index()
df_new_u.columns = ['User_ID', 'Count']
df_new_p.columns = ['Purchase', 'Count']

# List of continuous columns
continuous_columns = ['User_ID', 'Occupation', 'Marital_Status', 'Product_Category', 'Purchase']

for column in continuous_columns:
    if column != 'User_ID' and column != 'Purchase':
        plt.figure(figsize=(15, 5))

        # Histogram
        plt.subplot(1, 4, 1)
        sns.histplot(df[column].dropna(), kde=True)
        plt.title(f'Histogram of {column}')
        plt.tight_layout()

        # Countplot
        plt.subplot(1, 4, 2)
        sns.countplot(x=df[column].dropna())
        plt.title(f'Countplot of {column}')
        plt.xticks(rotation=90, ha='right')  # Rotate x-axis labels
        plt.tight_layout()

        # Dist plot
        plt.subplot(1, 4, 3)
        sns.histplot(df[column].dropna(), kde=True)
        plt.title(f'Dist plot of {column}')
        plt.tight_layout()

        plt.show()
    elif column == 'User_ID':
        plt.figure(figsize=(15, 5))

        # Histogram
        plt.subplot(1, 4, 1)
        sns.histplot(data=df_new_u, x='User_ID', weights='Count', kde=True)
        plt.title(f'Histogram of {column}')
        plt.tight_layout()

        # Countplot
        plt.subplot(1, 4, 2)
        sns.barplot(x=df_new_u['User_ID'], y=df_new_u['Count'])
        plt.title(f'Countplot of {column}')
        plt.xticks(rotation=90, ha='right')  # Rotate x-axis labels
        plt.tight_layout()

        # Dist plot
        plt.subplot(1, 4, 3)
        sns.histplot(x=df_new_u['User_ID'], weights=df_new_u['Count'], kde=True)
        plt.title(f'Dist plot of {column}')
        plt.tight_layout()

        plt.show()

    elif column == 'Purchase':
        plt.figure(figsize=(15, 5))

        # Histogram
        plt.subplot(1, 4, 1)
        sns.histplot(df_new_p, x='Purchase', weights='Count', kde=True)
        plt.title(f'Histogram of {column}')
```

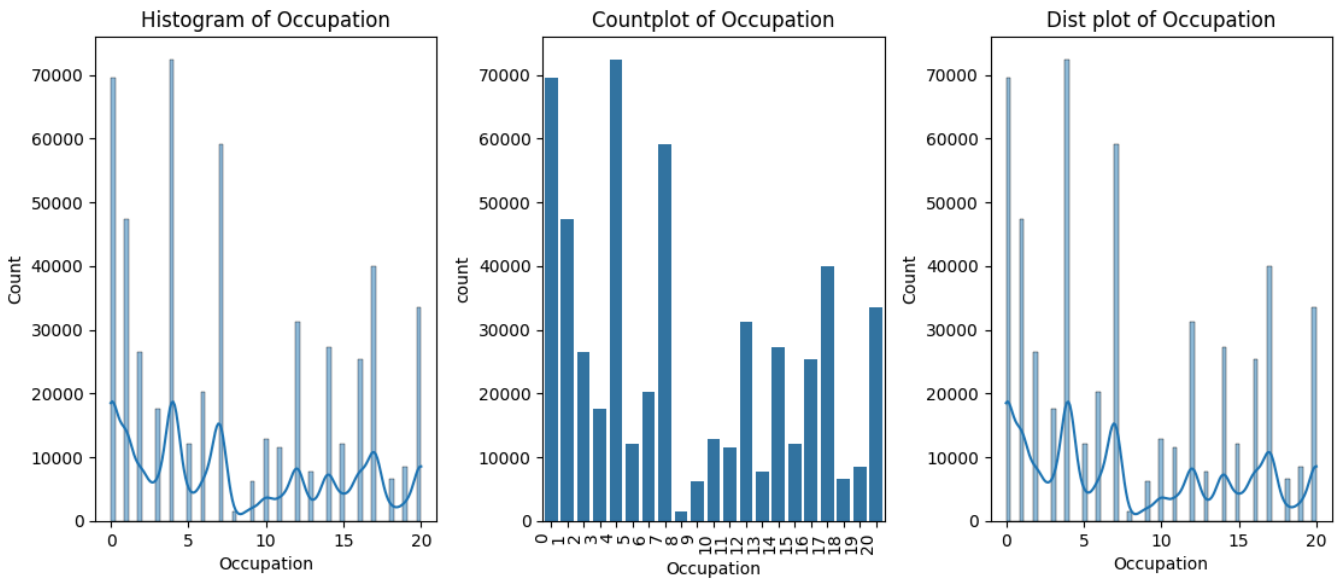
```
        plt.tight_layout()

        # Countplot
        plt.subplot(1, 4, 2)
        sns.barplot(x=df_new_p['Purchase'], y=df_new_p['Count'])
        plt.title(f'Countplot of {column}')
        plt.xticks(rotation=90, ha='right')  # Rotate x-axis labels
        plt.tight_layout()

        # Dist plot
        plt.subplot(1, 4, 3)
        sns.histplot(df_new_p, x='Purchase', weights='Count', kde=True)
        plt.title(f'Dist plot of {column}')
        plt.tight_layout()

        plt.show()

# Print the top 10 user_id and purchase counts
print("Top 10 User_ID counts:")
print(df_new_u)

print("\nTop 10 Purchase counts:")
print(df_new_p)
```

[4]


Histogram of User_ID
Count
User_ID
1e6

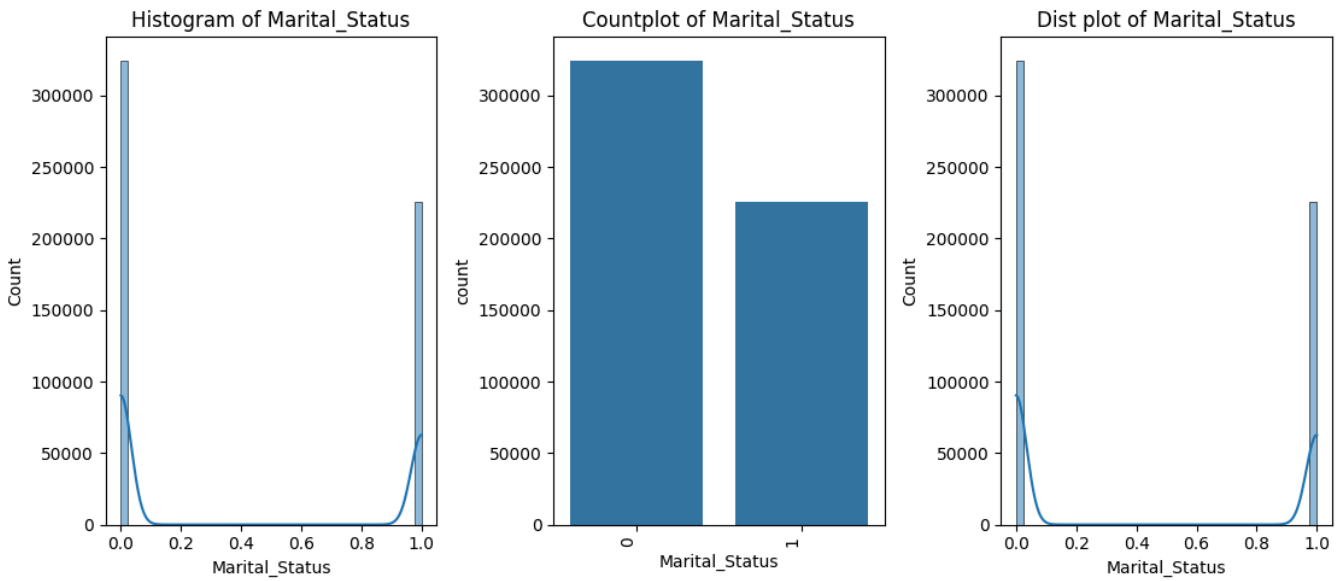
Countplot of User_ID
Count
1000889
1001015
1001150
1001181
1001680
1001941
1003618
1004277
1005795
1005831
User_ID

Dist plot of User_ID
Count
User_ID
1e6


Histogram of Occupation
Count
Occupation

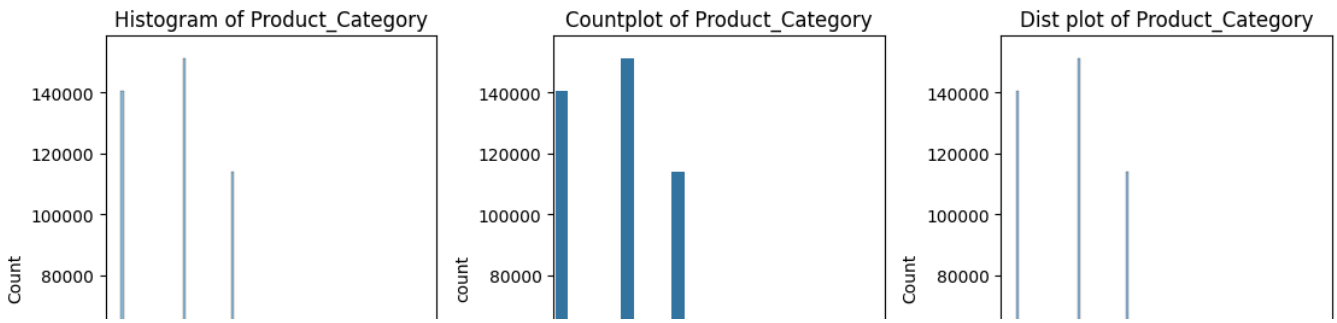
Countplot of Occupation
count
Occupation

Dist plot of Occupation
Count
Occupation


Histogram of Marital_Status
Count
Marital_Status

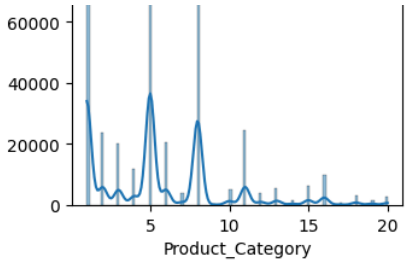
Countplot of Marital_Status
count
Marital_Status

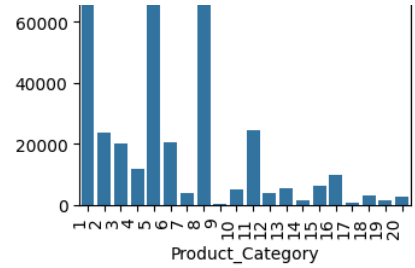
Dist plot of Marital_Status
Count
Marital_Status

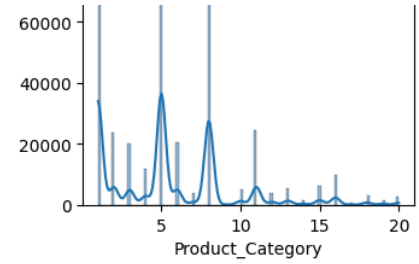

Histogram of Product_Category
Count

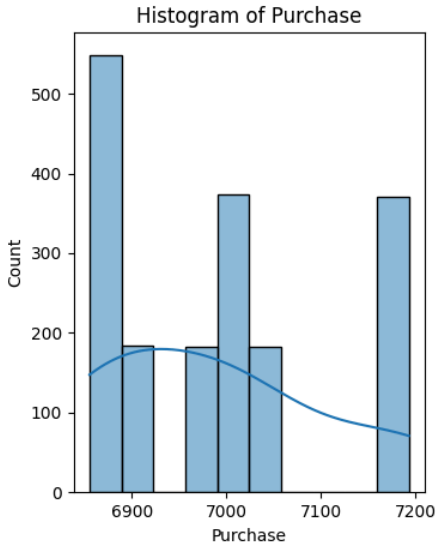
Countplot of Product_Category
count

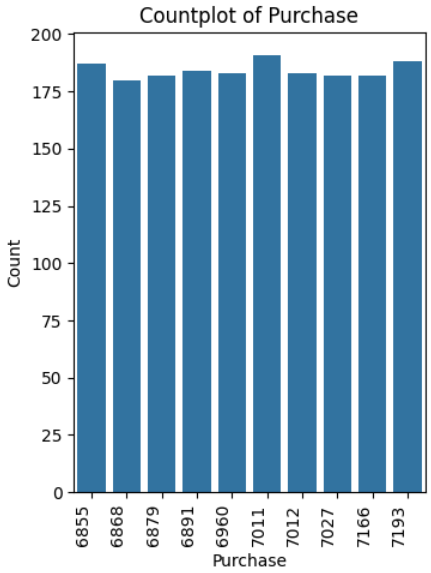
Dist plot of Product_Category
Count

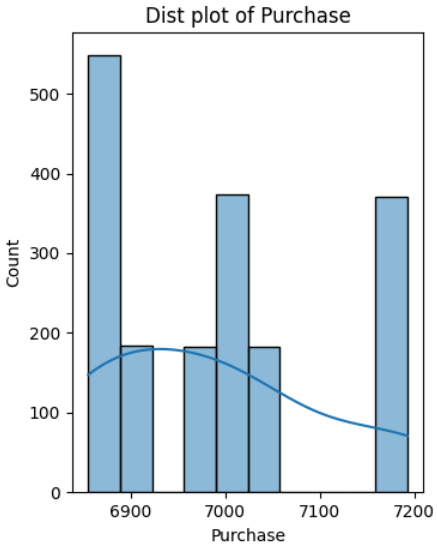












Top 10 User_ID counts:

	User_ID	Count
0	1001680	1026
1	1004277	979
2	1001941	898
3	1001181	862
4	1000889	823
5	1003618	767
6	1001150	752
7	1001015	740
8	1005795	729
9	1005831	727

Top 10 Purchase counts:

	Purchase	Count
0	7011	191
1	7193	188
2	6855	187
3	6891	184
4	7012	183
5	6960	183
6	6879	182
7	7166	182
8	7027	182
9	6868	180

🔍 Insights for Continuous Variables

1) User_ID

- Top 10 User_IDs: The most active users have been identified, with User_ID 1001680 making the highest number of purchases (1026).
- Distribution: The histogram and dist plot show the frequency of purchases made by the top 10 users.
- Countplot: The bar plot displays the count of purchases for each of the top 10 users.

2) Occupation

- Average: ~8.08
- Range: 0 to 20
- Variability: High (Standard Deviation: 6.52)
- Distribution: The histogram and dist plot show a fairly uniform distribution with no significant peaks.

3) Marital_Status

- Average: ~0.41 (indicating a higher proportion of not married users)
- Range: 0 to 1
- Variability: High (Standard Deviation: 0.49)
- Distribution: The histogram and dist plot show a bimodal distribution with peaks at 0 (Not Married) and 1 (Married).

4) Product_Category

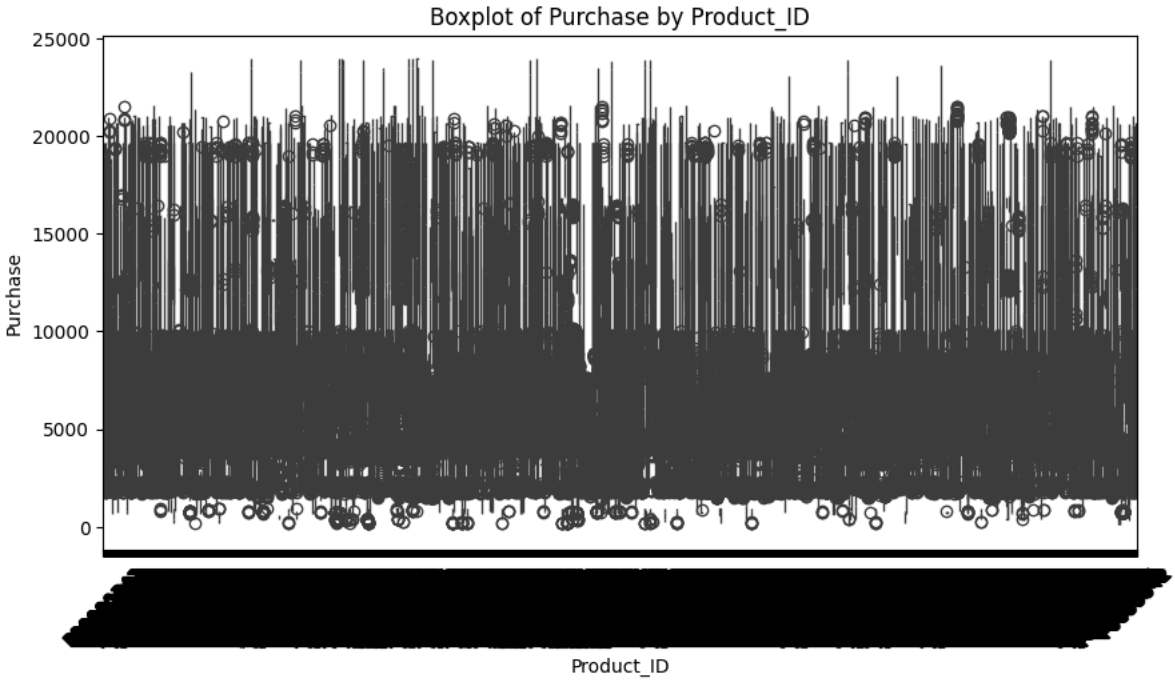
- Average: ~5.40
- Range: 1 to 20
- Variability: High (Standard Deviation: 3.94)
- Distribution: The histogram and dist plot show a right-skewed distribution with more users purchasing products in lower categories.

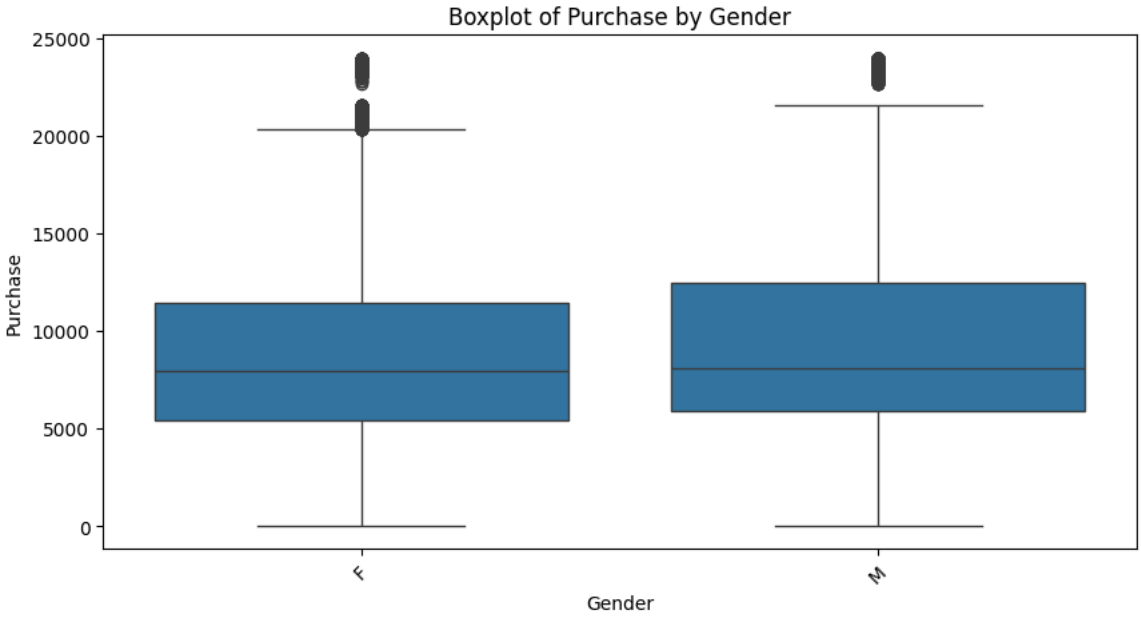
5) Purchase

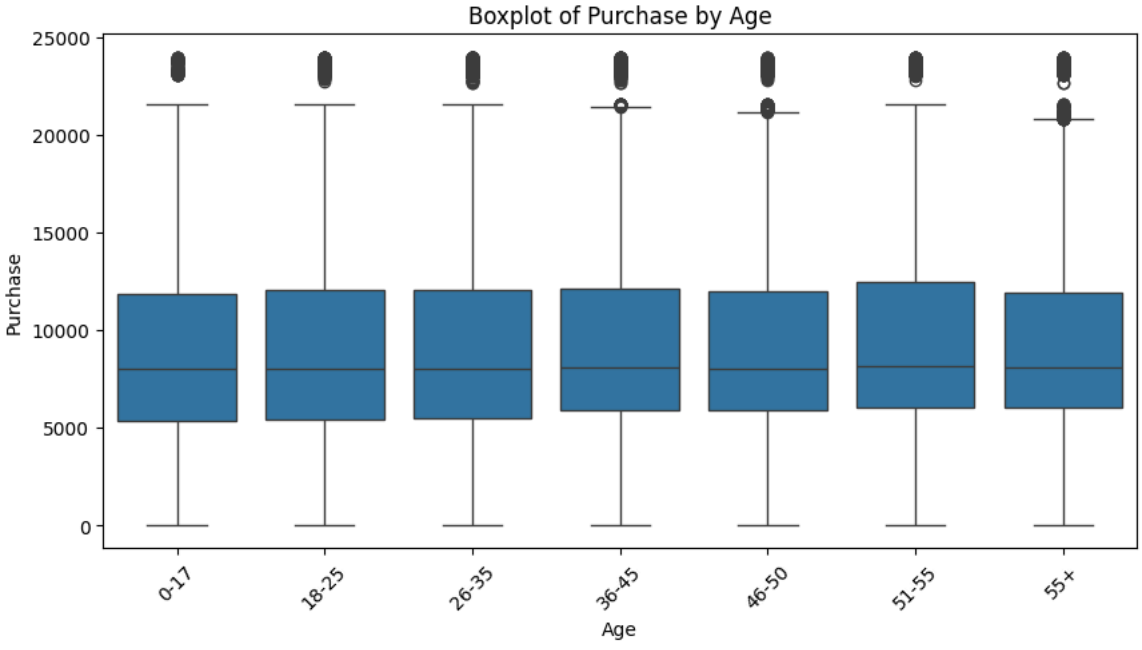
- Top 10 Purchase Amounts: The most common purchase amounts have been identified, with the amount 7011 occurring 191 times.
- Distribution: The histogram and dist plot show the frequency of the top 10 purchase amounts.
- Countplot: The bar plot displays the count of occurrences for each of the top 10 purchase amounts.

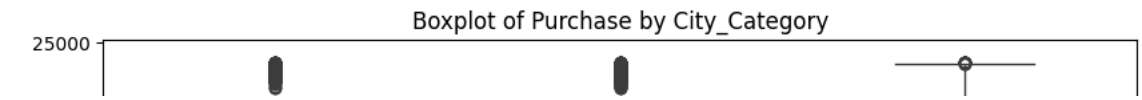
```
# Define the categorical columns you want to plot

#since product id has 3631 unique values hence it is not giving clear understanding.

categorical_columns = ['Product_ID', 'Gender', 'Age', 'City_Category', 'Stay_In_Current_City_Years']

# Create boxplots for each categorical column against the Purchase column
for col in categorical_columns:
  plt.figure(figsize=(10, 5))
  sns.boxplot(x=col, y='Purchase', data=df)
  plt.title(f'Boxplot of Purchase by {col}')
  plt.xticks(rotation=45)  # Rotate x-axis labels if needed
  plt.show()
```

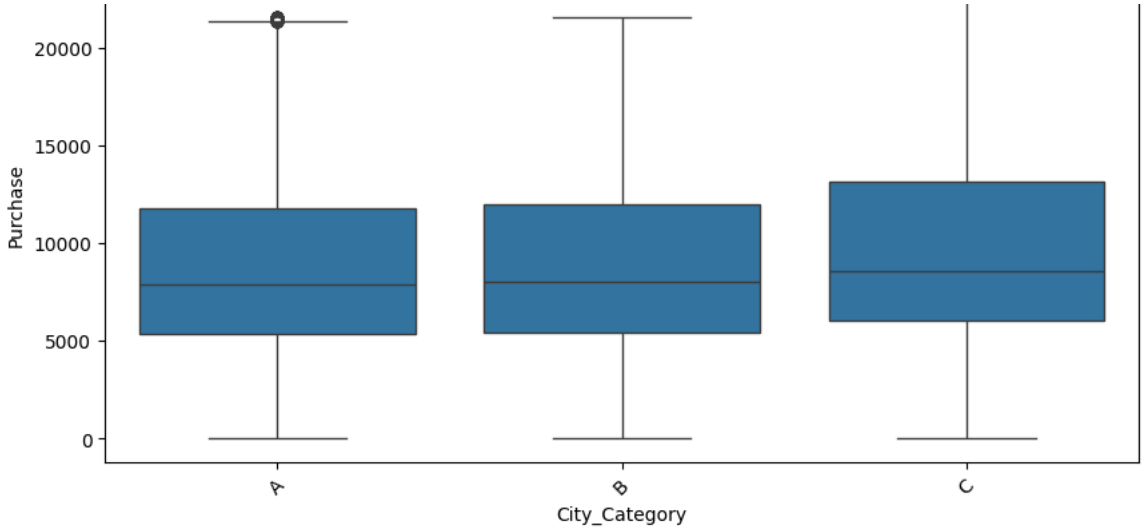
[+]

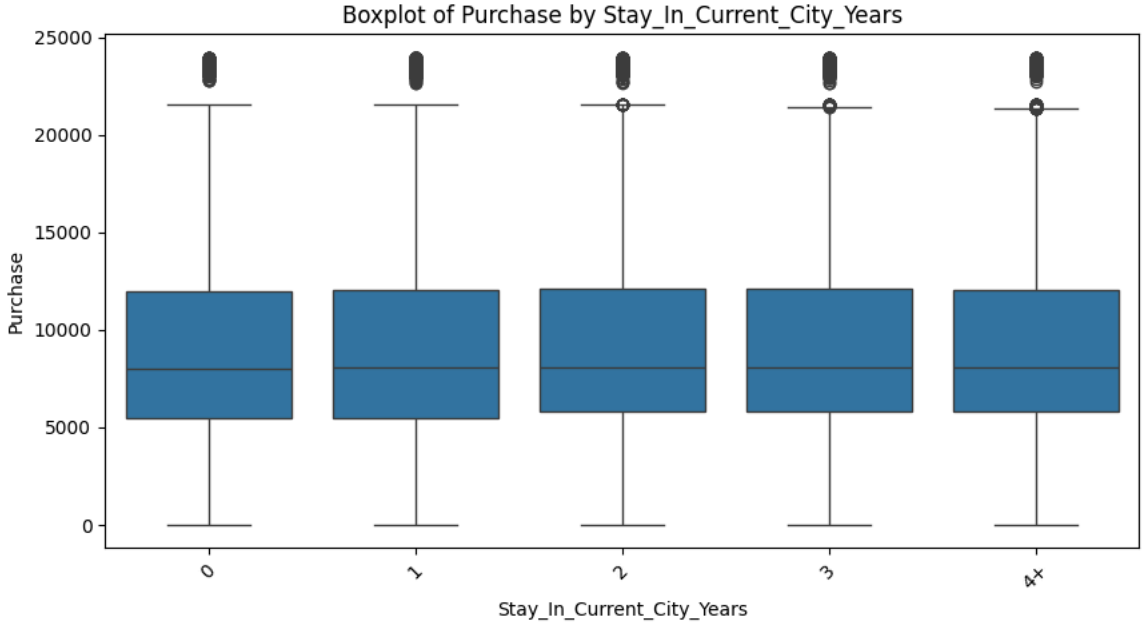

Boxplot of Purchase by Product_ID
Purchase
25000
20000
15000
10000
5000
0
Product_ID


Boxplot of Purchase by Gender
Purchase
25000
20000
15000
10000
5000
0
F
M
Gender

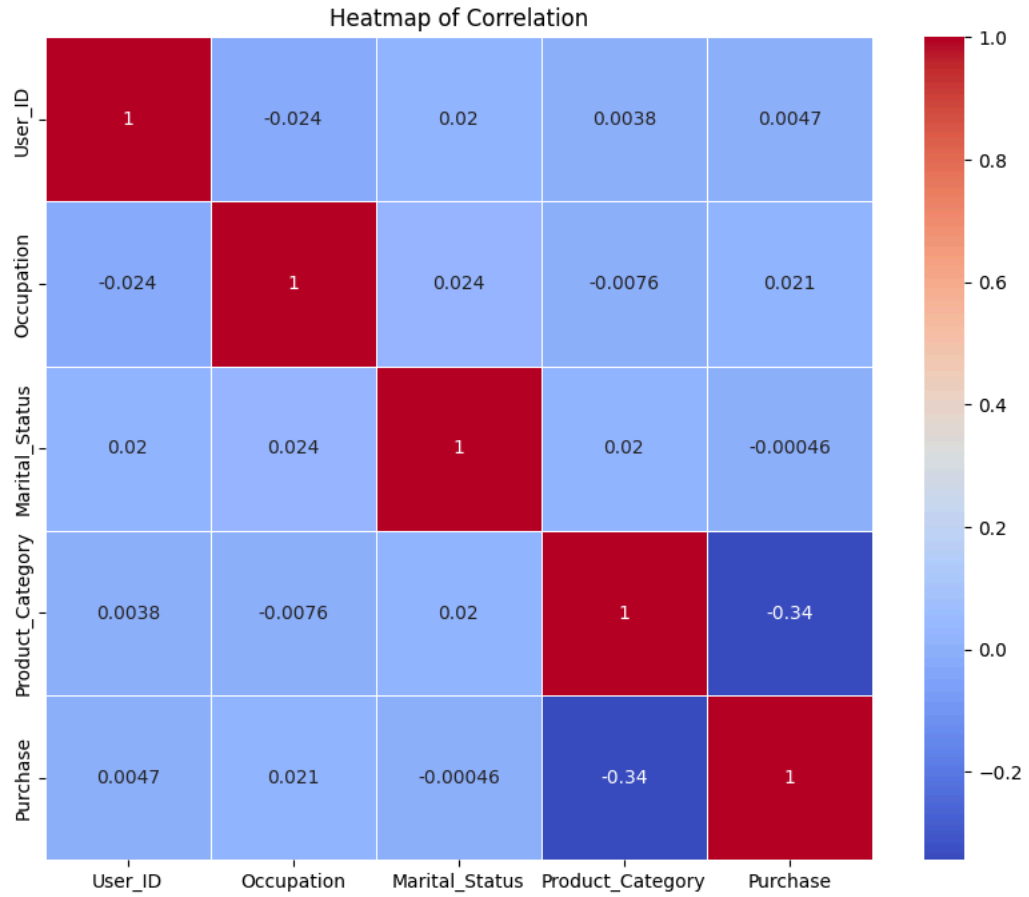

Boxplot of Purchase by Age
Purchase
25000
20000
15000
10000
5000
0
0-17
18-25
26-35
36-45
46-50
51-55
55+
Age


Boxplot of Purchase by City_Category
25000


20000
15000
10000
5000
0
Purchase
A
B
C
City_Category


Boxplot of Purchase by Stay_In_Current_City_Years
25000
20000
15000
10000
5000
0
Purchase
0
1
2
3
4+
Stay_In_Current_City_Years

```
# Select only the numerical columns for correlation
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns
# Calculate the correlation matrix
corr_matrix = df[numerical_columns].corr()
# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Heatmap of Correlation')
plt.show()
```




```
corr_matrix
```

	User_ID	Occupation	Marital_Status	Product_Category	Purchase
User_ID	1.000000	-0.023971	0.020443	0.003825	0.004716
Occupation	-0.023971	1.000000	0.024280	-0.007618	0.020833
Marital_Status	0.020443	0.024280	1.000000	0.019888	-0.000463
Product_Category	0.003825	-0.007618	0.019888	1.000000	-0.343703
Purchase	0.004716	0.020833	-0.000463	-0.343703	1.000000

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

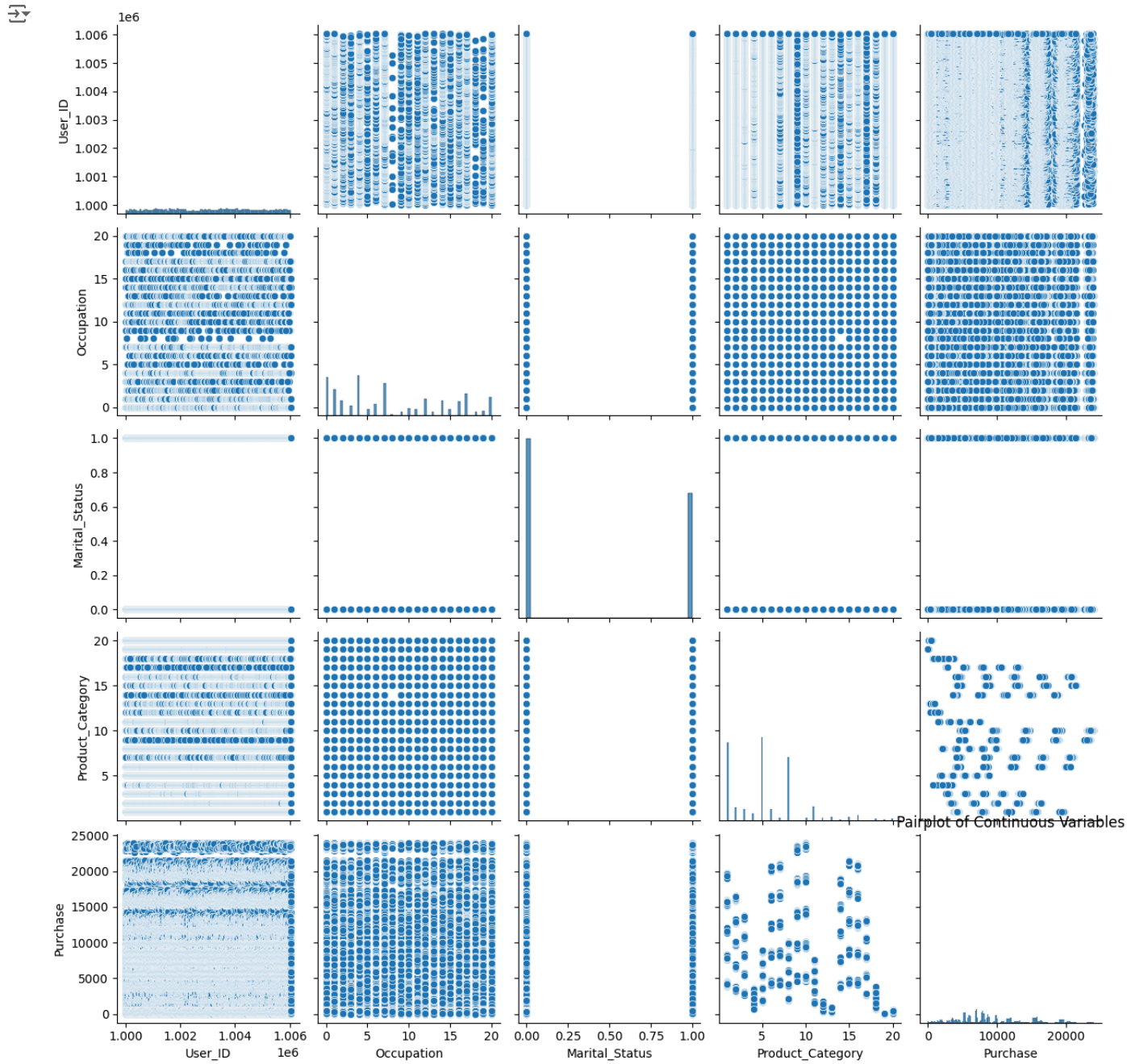
```
# Generate insights from the correlation matrix
def generate_insights(corr_matrix):
    weak_negative_correlations = []
    weak_positive_correlations = []
    moderate_positive_correlations = []
    moderate_negative_correlations = []
    strong_positive_correlations = []
    strong_negative_correlations = []

    for col in corr_matrix.columns:
        for row in corr_matrix.index:
            if col != row:
                correlation = corr_matrix.loc[row, col]
                if -0.3 <= correlation <= 0:
                    weak_negative_correlations.append(f"Weak Negative correlation between {row} and {col}: {correlation:4f}")
                elif 0 < correlation <= 0.3:
                    weak_positive_correlations.append(f"Weak positive correlation between {row} and {col}: {correlation:4f}")
```

```
                elif 0.3 < correlation <= 0.7:
                    moderate_positive_correlations.append(f"Moderate positive correlation between {row} and {col}: {correlation:4f}")
                elif -0.7 <= correlation < -0.3:
                    moderate_negative_correlations.append(f"Moderate negative correlation between {row} and {col}: {correlation:4f}")
                elif correlation > 0.7:
                    strong_positive_correlations.append(f"Strong positive correlation between {row} and {col}: {correlation:4f}")
                elif correlation < -0.7:
                    strong_negative_correlations.append(f"Strong negative correlation between {row} and {col}: {correlation:4f}")

    return weak_negative_correlations,weak_positive_correlations, moderate_positive_correlations, moderate_negative_correlations, strong

# Generate and print insights
weak_negative_correlations,weak_positive_correlations, moderate_positive_correlations, moderate_negative_correlations, strong_positive_c

print("🔍 INSIGHTS:\n")

print("Weak Positive Correlations:")
if weak_positive_correlations:
    for insight in weak_positive_correlations:
        print(insight)
else:
    print("There is no weak positive correlation")

print("Weak Negative Correlations:")
if weak_negative_correlations:
    for insight in weak_negative_correlations:
        print(insight)
else:
    print("There is no weak negative correlation")
print("\nModerate Positive Correlations:")
if moderate_positive_correlations:
    for insight in moderate_positive_correlations:
        print(insight)
else:
    print("There is no moderate positive correlation")

print("\nModerate Negative Correlations:")
if moderate_negative_correlations:
    for insight in moderate_negative_correlations:
        print(insight)
else:
    print("There is no moderate negative correlation")

print("\nStrong Positive Correlations:")
if strong_positive_correlations:
    for insight in strong_positive_correlations:
        print(insight)
else:
    print("There is no strong positive correlation")

print("\nStrong Negative Correlations:")
if strong_negative_correlations:
    for insight in strong_negative_correlations:
        print(insight)
else:
    print("There is no strong negative correlation")
```

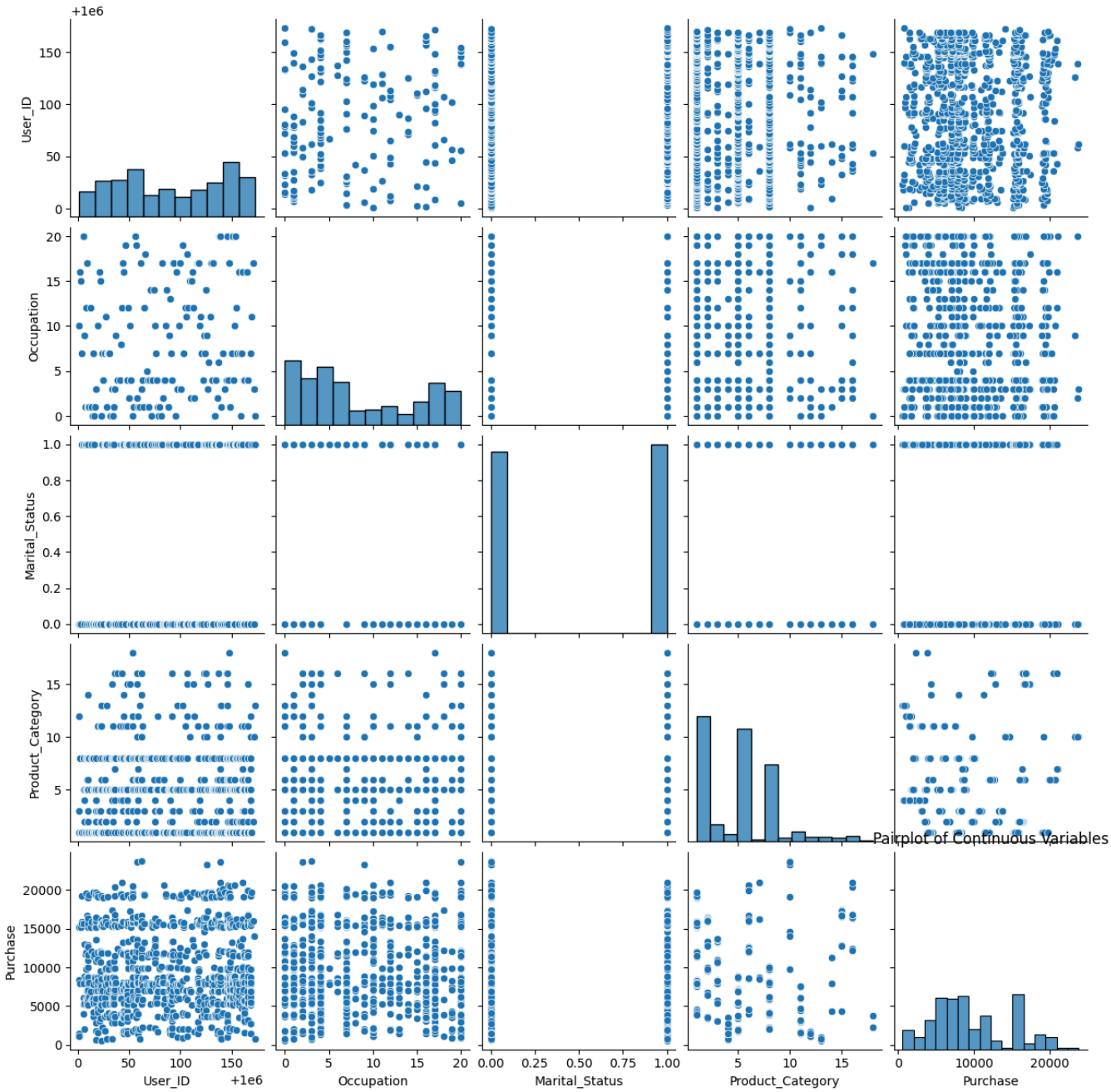
```
🔍 INSIGHTS:

Weak Positive Correlations:
Weak positive correlation between Marital_Status and User_ID: 0.020443
Weak positive correlation between Product_Category and User_ID: 0.003825
Weak positive correlation between Purchase and User_ID: 0.004716
Weak positive correlation between Marital_Status and Occupation: 0.024280
Weak positive correlation between Purchase and Occupation: 0.020833
Weak positive correlation between User_ID and Marital_Status: 0.020443
Weak positive correlation between Occupation and Marital_Status: 0.024280
Weak positive correlation between Product_Category and Marital_Status: 0.019888
Weak positive correlation between User_ID and Product_Category: 0.003825
Weak positive correlation between Marital_Status and Product_Category: 0.019888
Weak positive correlation between User_ID and Purchase: 0.004716
Weak positive correlation between Occupation and Purchase: 0.020833
Weak Negative Correlations:
Weak Negative correlation between Occupation and User_ID: -0.023971
Weak Negative correlation between User_ID and Occupation: -0.023971
Weak Negative correlation between Product_Category and Occupation: -0.007618
Weak Negative correlation between Purchase and Marital_Status: -0.000463
Weak Negative correlation between Occupation and Product_Category: -0.007618
Weak Negative correlation between Marital_Status and Purchase: -0.000463

Moderate Positive Correlations:
There is no moderate positive correlation

Moderate Negative Correlations:
Moderate negative correlation between Purchase and Product_Category: -0.343703
```

```
Moderate negative correlation between Product_Category and Purchase: -0.343703

Strong Positive Correlations:
There is no strong positive correlation

Strong Negative Correlations:
There is no strong negative correlation
```

```
# Plot the pairplot
sns.pairplot(df[['User_ID', 'Occupation', 'Marital_Status', 'Product_Category', 'Purchase']])
plt.title('Pairplot of Continuous Variables')
plt.show()
```




```
# Plot the pairplot for first 1000 rows for clear visibility
df_1000=df.head(1000)
sns.pairplot(df_1000[['User_ID', 'Occupation', 'Marital_Status', 'Product_Category', 'Purchase']])
plt.title('Pairplot of Continuous Variables')
plt.show()
```



🔍 Insights:

1. Scatter Plots:

- The scatter plots in the pairplot provide a visual representation of the relationships between continuous variables. For example, the scatter plot between Martial_status and User_ID has weak positive correlations and so on.
- Similarly, there is a no positive correlations between two columns.

2. Histograms:

- The histograms on the diagonal of the pairplot show the distribution of each continuous variable. These histograms help in understanding the spread and central tendency of the data for variables..

3. Outliers:

- The pairplot helps identify outliers in the data.
- These outliers can be seen as points that are distant from the main cluster of data points.

- Outliers detail observation is presented below.

Business Insights based on Non-Graphical and Visual Analysis

```
# Comments on the range of attributes
print("Comments on the range of attributes:")
for column in df.columns:
    if pd.api.types.is_categorical_dtype(df[column]):
        df[column] = df[column].astype('category').cat.as_ordered()

    print(f"{column}: {df[column].min()} to {df[column].max()}")
```

```
Comments on the range of attributes:
User_ID: 1000001 to 1006040
Product_ID: P00000142 to P0099942
Gender: F to M
Age: 0-17 to 55+
Occupation: 0 to 20
City_Category: A to C
Stay_In_Current_City_Years: 0 to 4+
Marital_Status: 0 to 1
Product_Category: 1 to 20
Purchase: 12 to 23961
```

INSIGHTS:

1) User_ID: Ranges from 1000001 to 1006040, indicating a large number of unique users.

2) Product_ID: Ranges from P00000142 to P0099942, suggesting a wide variety of products.

3) Gender: Categorical variable with values 'F' and 'M', representing female and male.

4) Age: Ranges from '0-17' to '55+', covering a broad spectrum of age groups.

5) Occupation: Ranges from 0 to 20, indicating diverse occupational categories.

6) City_Category: Categorical variable with values 'A', 'B', and 'C', representing different city tiers.

7) Stay_In_Current_City_Years: Ranges from '0' to '4+', showing varying lengths of stay in the current city.

8) Marital_Status: Binary variable with values 0 and 1, indicating marital status (e.g., single or married).

9) Product_Category: Ranges from 1 to 20, indicating different product categories.

10) Purchase: Ranges from 12 to 23961, showing a wide range of purchase amounts.

```
# Comments on the distribution of the variables and relationship between them
def comments_on_distribution(df):
    comments = []

    for column in df.columns:
        comments.append(f"{column}:")
        if pd.api.types.is_numeric_dtype(df[column]):
            comments.append(f" - Distribution: The {column} distribution spans from {df[column].min()} to {df[column].max()}.")
        else:
            comments.append(f" - Distribution: The {column} distribution includes categories: {df[column].unique().tolist()}.")

        # Relationship with other variables
        comments.append(" - Relationship with Other Variables:")
        for other_column in df.columns:
            if column != other_column:
                comments.append(f"   - {other_column}: Relationship analysis between {column} and {other_column}.")

        comments.append("\n")

    return "\n".join(comments)

# Print the comments on distribution and relationships
print(comments_on_distribution(df))
```

- Purchase: Relationship analysis between Stay_In_Current_City_Years and Purchase.

Marital_Status:
- Distribution: The Marital_Status distribution spans from 0 to 1.
- Relationship with Other Variables:
  - User_ID: Relationship analysis between Marital_Status and User_ID.
  - Product_ID: Relationship analysis between Marital_Status and Product_ID.
  - Gender: Relationship analysis between Marital_Status and Gender.
  - Age: Relationship analysis between Marital_Status and Age.
  - Occupation: Relationship analysis between Marital_Status and Occupation.
  - City_Category: Relationship analysis between Marital_Status and City_Category.
  - Stay_In_Current_City_Years: Relationship analysis between Marital_Status and Stay_In_Current_City_Years.
  - Product_Category: Relationship analysis between Marital_Status and Product_Category.
  - Purchase: Relationship analysis between Marital_Status and Purchase.

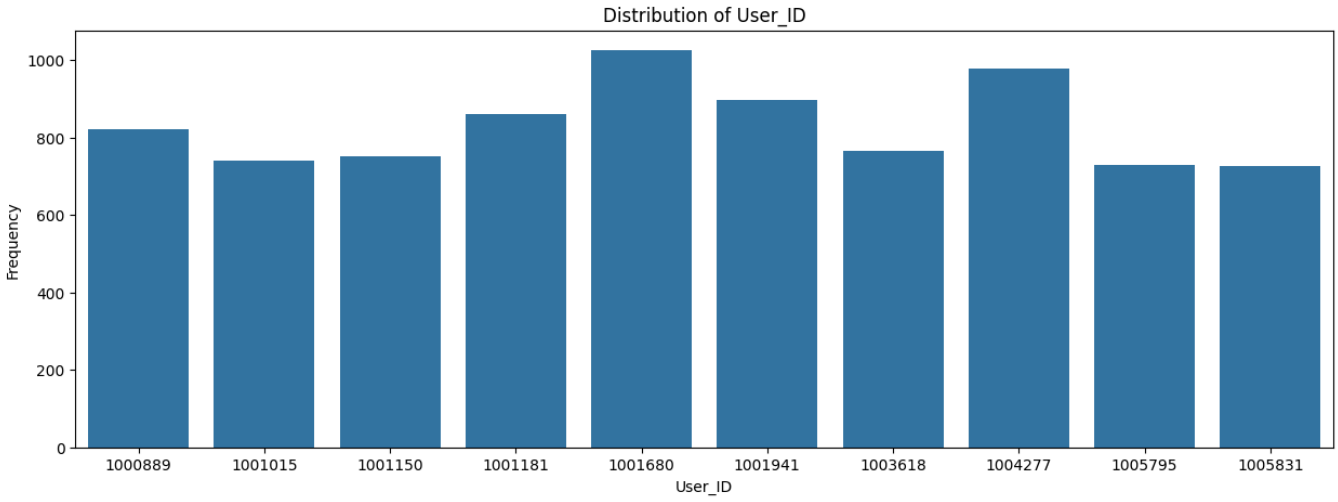
Product_Category:
- Distribution: The Product_Category distribution spans from 1 to 20.
- Relationship with Other Variables:
  - User_ID: Relationship analysis between Product_Category and User_ID.
  - Product_ID: Relationship analysis between Product_Category and Product_ID.
  - Gender: Relationship analysis between Product_Category and Gender.
  - Age: Relationship analysis between Product_Category and Age.
  - Occupation: Relationship analysis between Product_Category and Occupation.
  - City_Category: Relationship analysis between Product_Category and City_Category.
  - Stay_In_Current_City_Years: Relationship analysis between Product_Category and Stay_In_Current_City_Years.
  - Marital_Status: Relationship analysis between Product_Category and Marital_Status.
  - Purchase: Relationship analysis between Product_Category and Purchase.

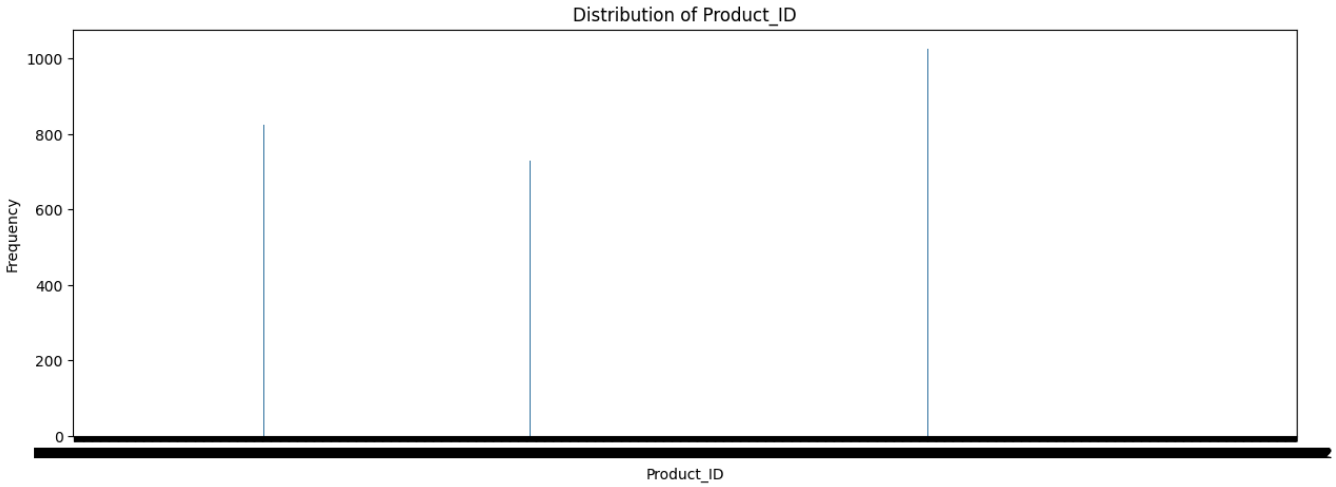
Purchase:
- Distribution: The Purchase distribution spans from 12 to 23961.
- Relationship with Other Variables:
  - User_ID: Relationship analysis between Purchase and User_ID.
  - Product_ID: Relationship analysis between Purchase and Product_ID.
  - Gender: Relationship analysis between Purchase and Gender.
  - Age: Relationship analysis between Purchase and Age.
  - Occupation: Relationship analysis between Purchase and Occupation.
  - City_Category: Relationship analysis between Purchase and City_Category.
  - Stay_In_Current_City_Years: Relationship analysis between Purchase and Stay_In_Current_City_Years.
  - Marital_Status: Relationship analysis between Purchase and Marital_Status.
  - Product_Category: Relationship analysis between Purchase and Product_Category.

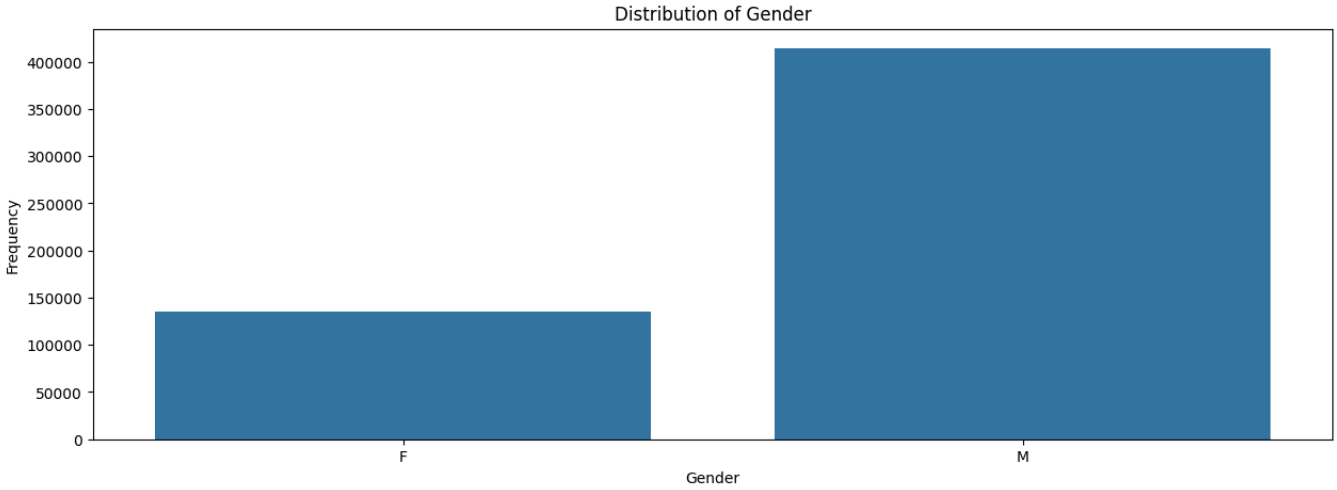
```
# Univariate Analysis
# Get the top 10 user_id and purchase and product_id as there is huge data for 3 of them and want to plot graph in a readable format.
df_new_u = df['User_ID'].value_counts().sort_values(ascending=False).head(10).reset_index()
df_new_p = df['Purchase'].value_counts().sort_values(ascending=False).head(10).reset_index()
df_new_pro=df['Product_ID'].value_counts().sort_values(ascending=False).head(10).reset_index()
df_new_u.columns = ['User_ID', 'Count']
df_new_p.columns = ['Purchase', 'Count']
df_new_pro.columns = ['Product_ID', 'Count']

def plot_univariate(column):
  if column != 'User_ID' and column != 'Purchase' and column !='Product_ID':
    plt.figure(figsize=(15, 5))
    sns.countplot(x=df[column].dropna())
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
  elif column == 'User_ID':
    plt.figure(figsize=(15, 5))
    sns.barplot(x=df_new_u['User_ID'], y=df_new_u['Count'])
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
  elif column == 'Purchase':
    plt.figure(figsize=(15, 5))
    sns.barplot(x=df_new_p['Purchase'], y=df_new_u['Count'])
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
  elif column == 'Product_ID':
    plt.figure(figsize=(15, 5))
    sns.barplot(x=df_new_pro['Product_ID'], y=df_new_u['Count'])
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```

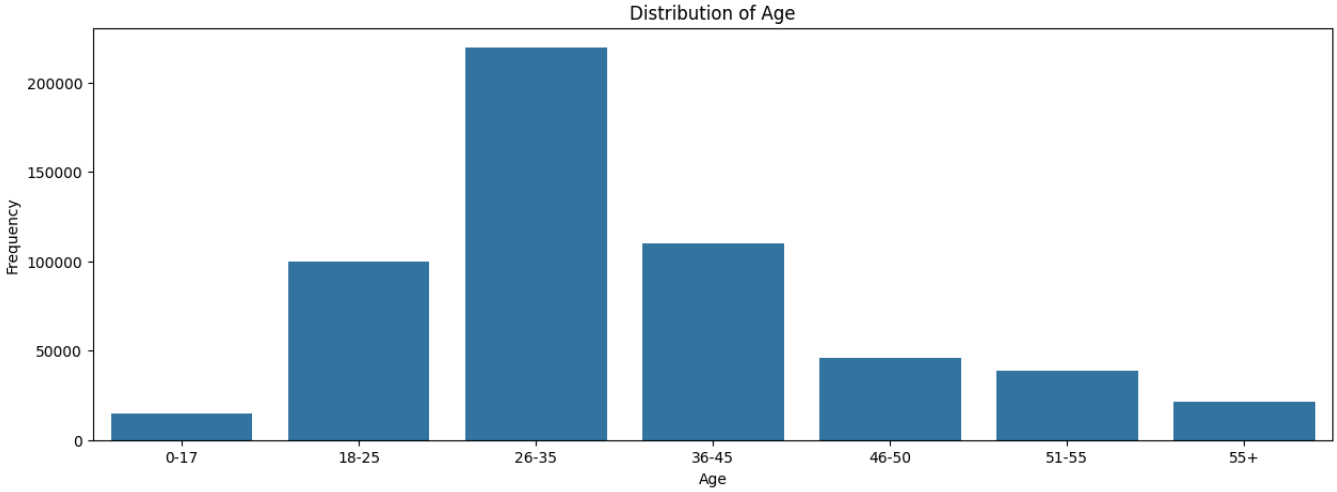
```
# Plotting Univariate Distributions
for column in df.columns:
    plot_univariate(column)
```

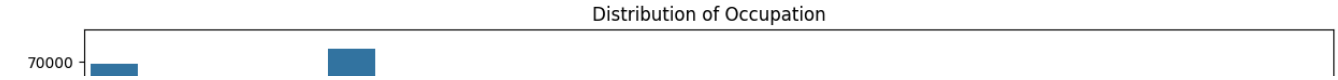
[4]

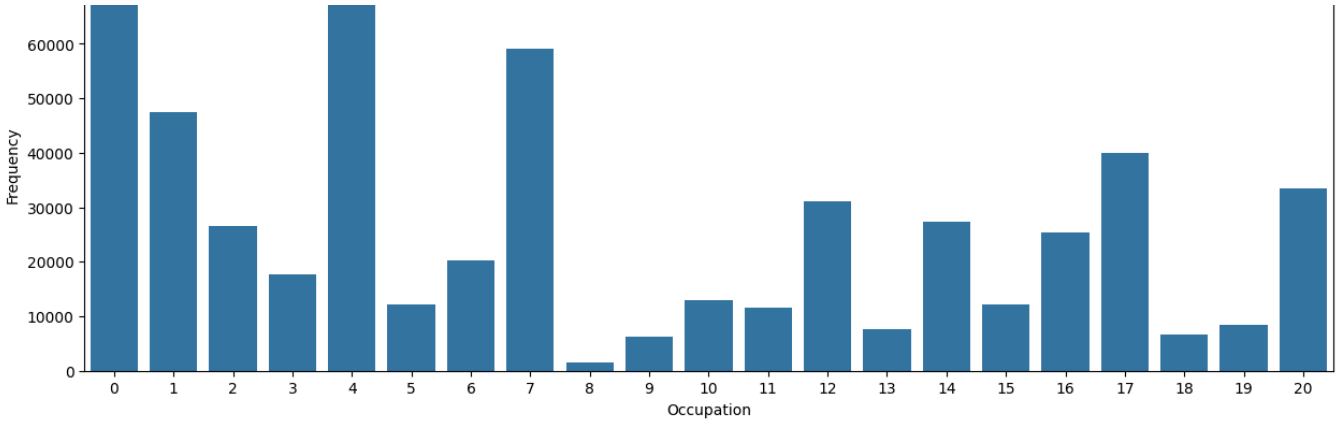

Distribution of User_ID
Frequency
0
200
400
600
800
1000
1000889
1001015
1001150
1001181
1001680
1001941
1003618
1004277
1005795
1005831
User_ID

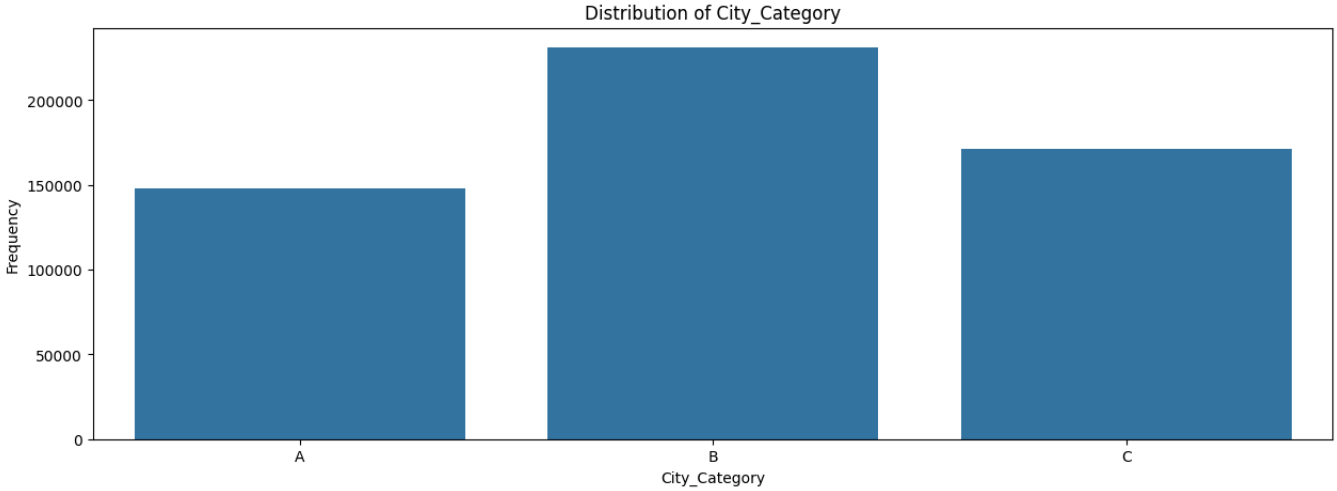

Distribution of Product_ID
Frequency
0
200
400
600
800
1000
Product_ID

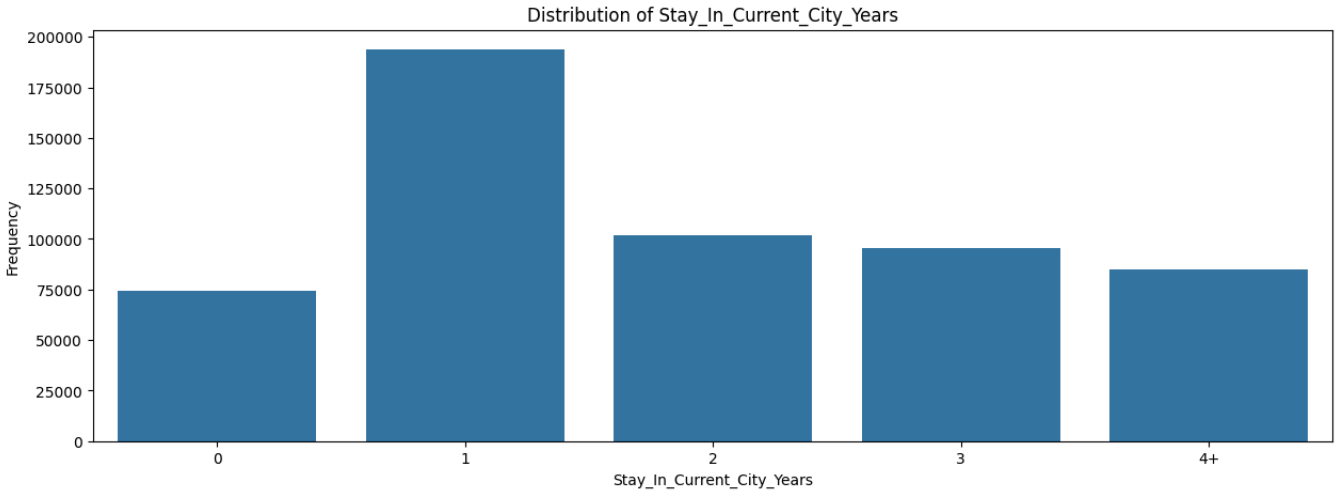

Distribution of Gender
Frequency
0
50000
100000
150000
200000
250000
300000
350000
400000
F
M
Gender

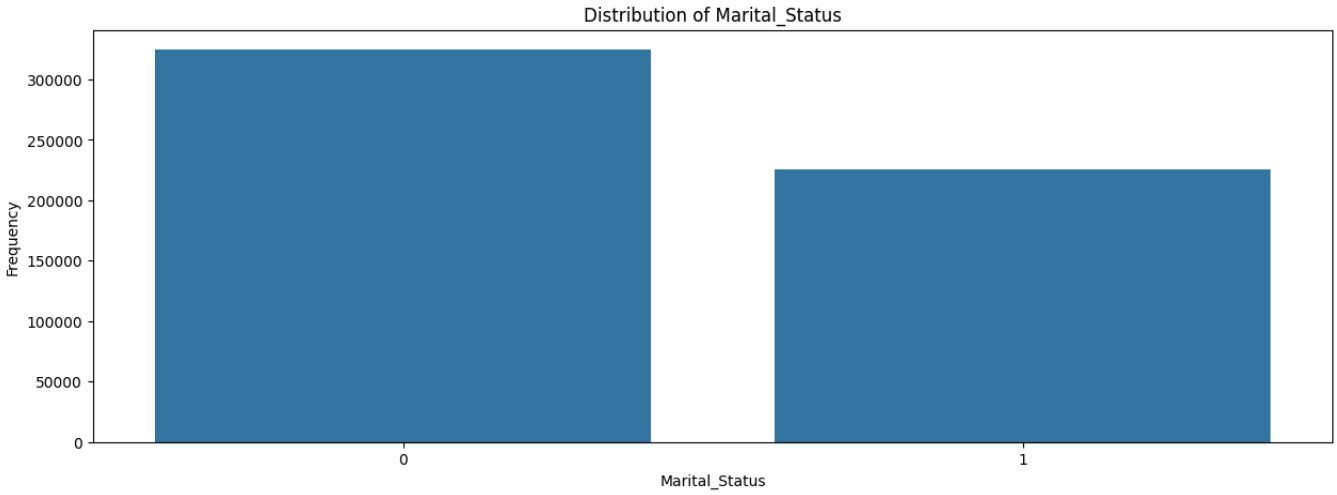

Distribution of Age
Frequency
0
50000
100000
150000
200000
0-17
18-25
26-35
36-45
46-50
51-55
55+
Age

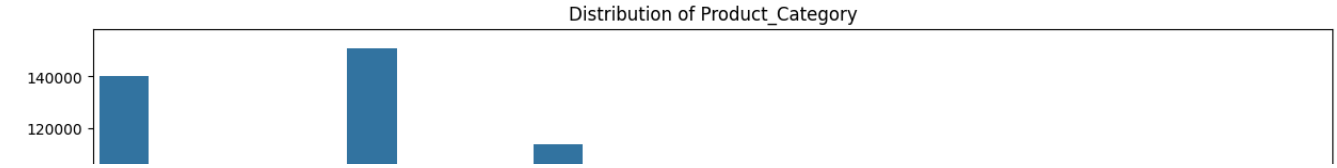

Distribution of Occupation
70000

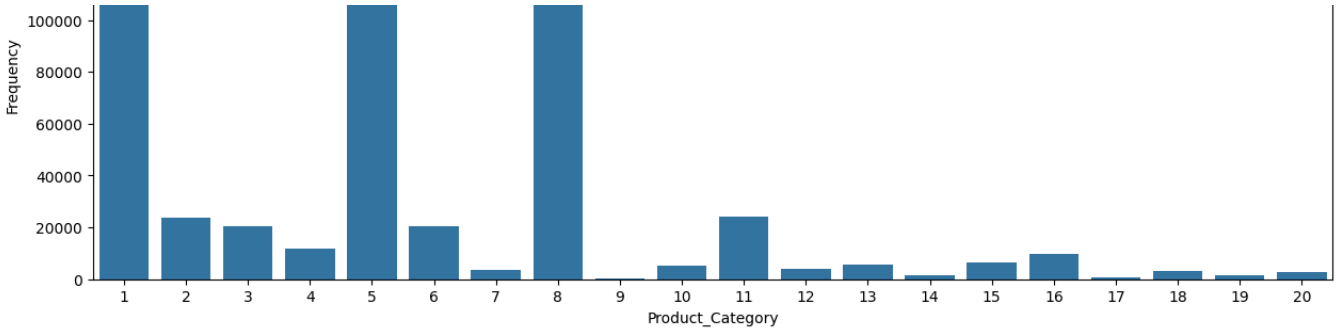

Frequency
0
10000
20000
30000
40000
50000
60000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Occupation

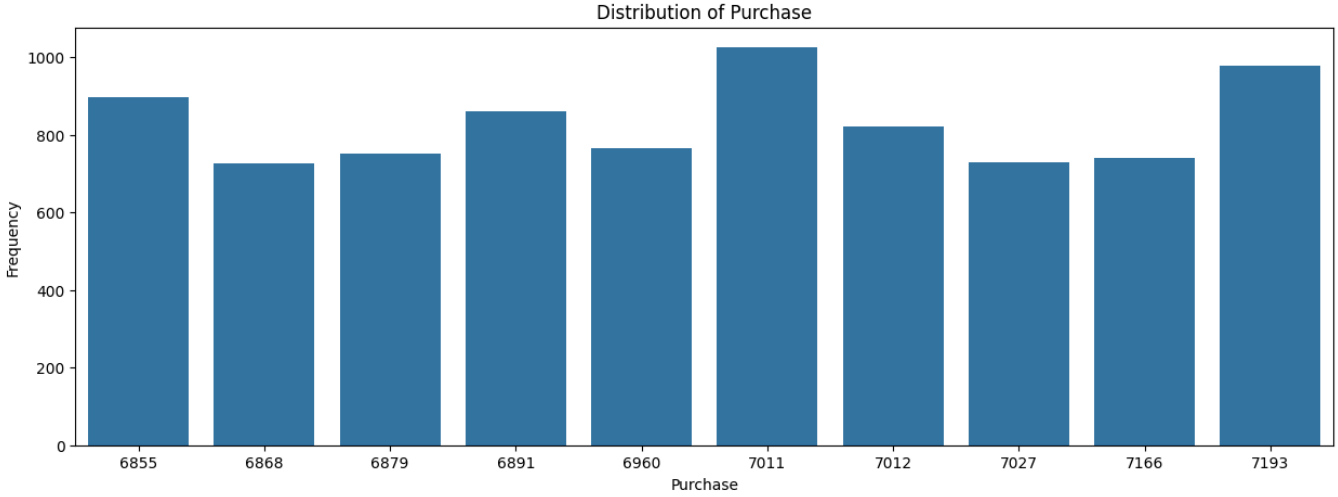

Distribution of City_Category
Frequency
0
50000
100000
150000
200000
A
B
C
City_Category


Distribution of Stay_In_Current_City_Years
Frequency
0
25000
50000
75000
100000
125000
150000
175000
200000
0
1
2
3
4+
Stay_In_Current_City_Years


Distribution of Marital_Status
Frequency
0
50000
100000
150000
200000
250000
300000
0
1
Marital_Status

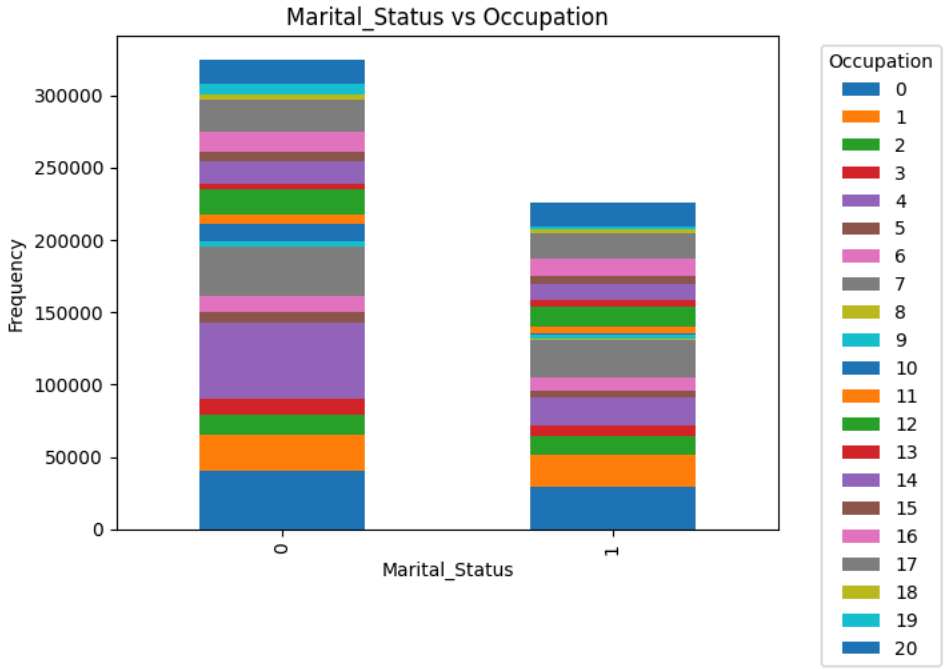

Distribution of Product_Category
120000
140000

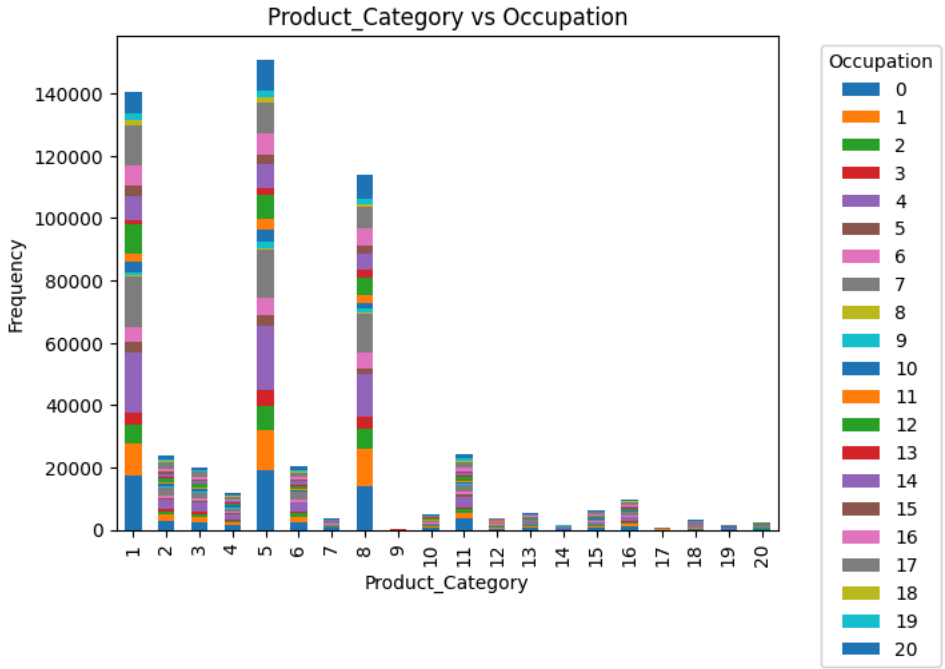

Frequency
100000
80000
60000
40000
20000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Product_Category

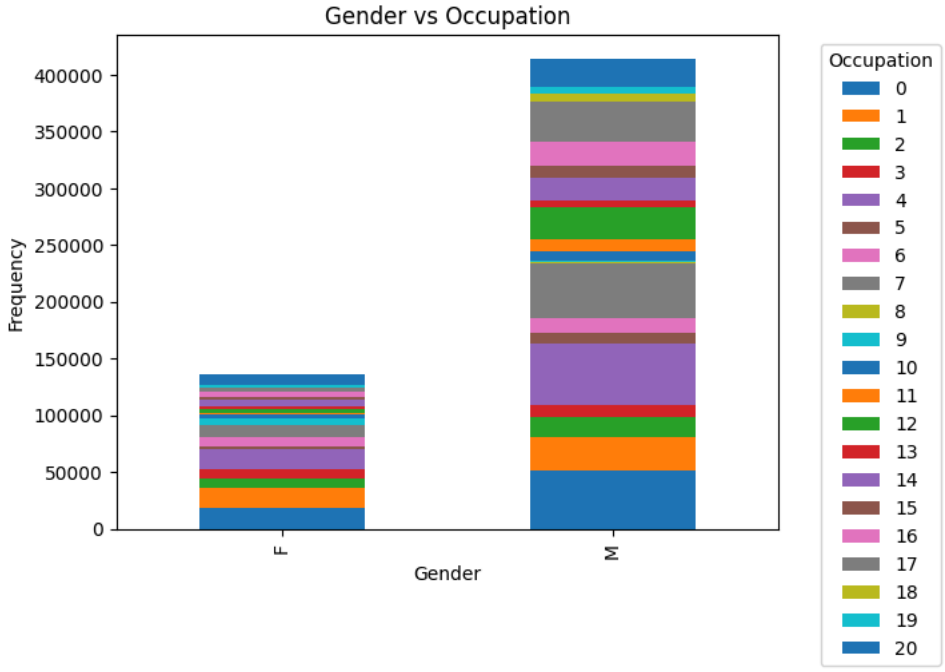

Distribution of Purchase
Frequency
1000
800
600
400
200
0
6855
6868
6879
6891
6960
7011
7012
7027
7166
7193
Purchase

```
# Bivariate Analysis
new_df = df[['Occupation', 'Marital_Status', 'Product_Category','Gender', 'Age', 'City_Category', 'Stay_In_Current_City_Years']]


def plot_bivariate(x, y):
    ax = pd.crosstab(new_df[x], new_df[y]).plot(kind='bar', stacked=True)
    plt.title(f'{x} vs {y}')
    plt.xlabel(x)
    plt.ylabel('Frequency')
    plt.legend(title=y, bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.show()

# Plotting Bivariate Distributions
for i in range(len(new_df.columns)):
    for j in range(i + 1, len(new_df.columns)):
        plot_bivariate(new_df.columns[j], new_df.columns[i])
```

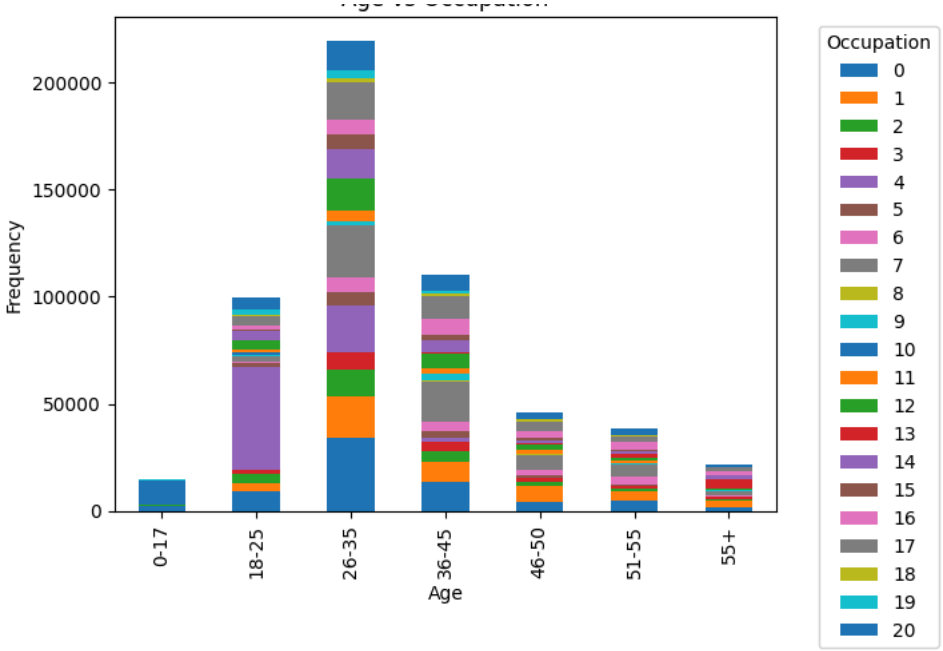
[4]

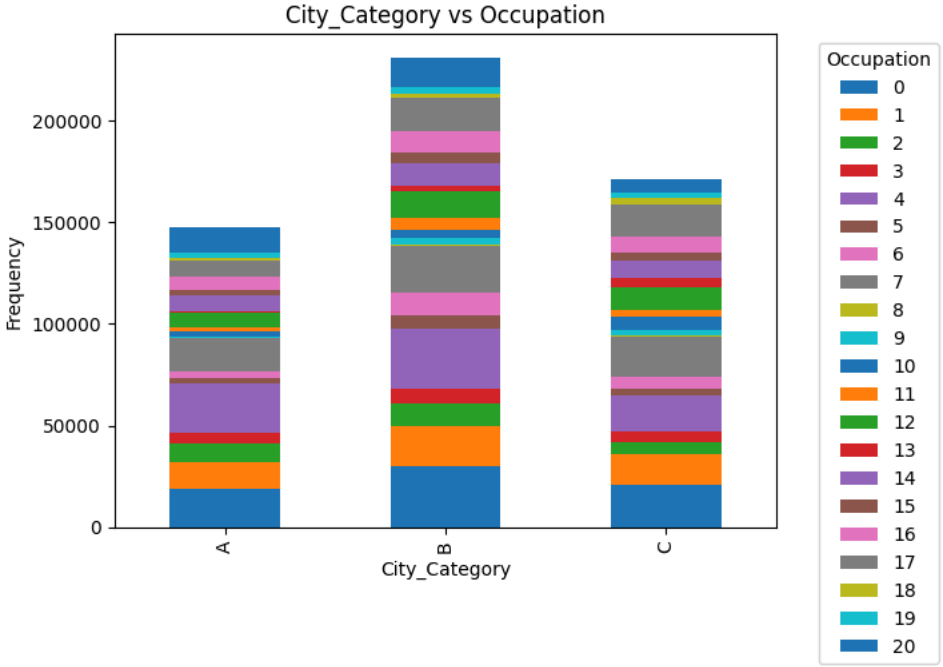


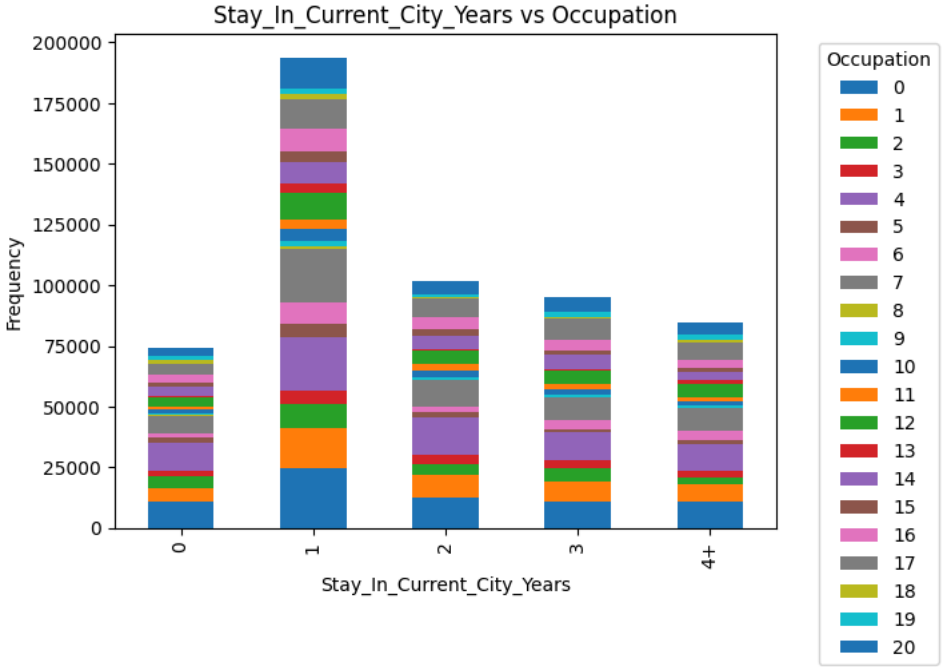


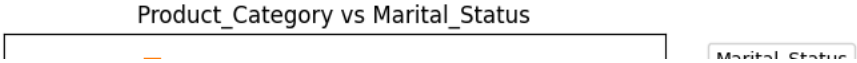


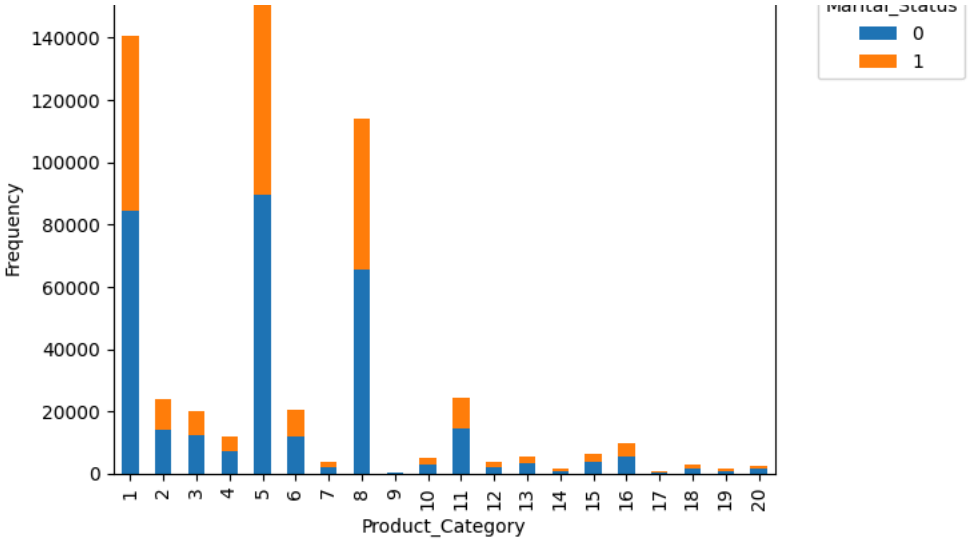
Age vs Occupation

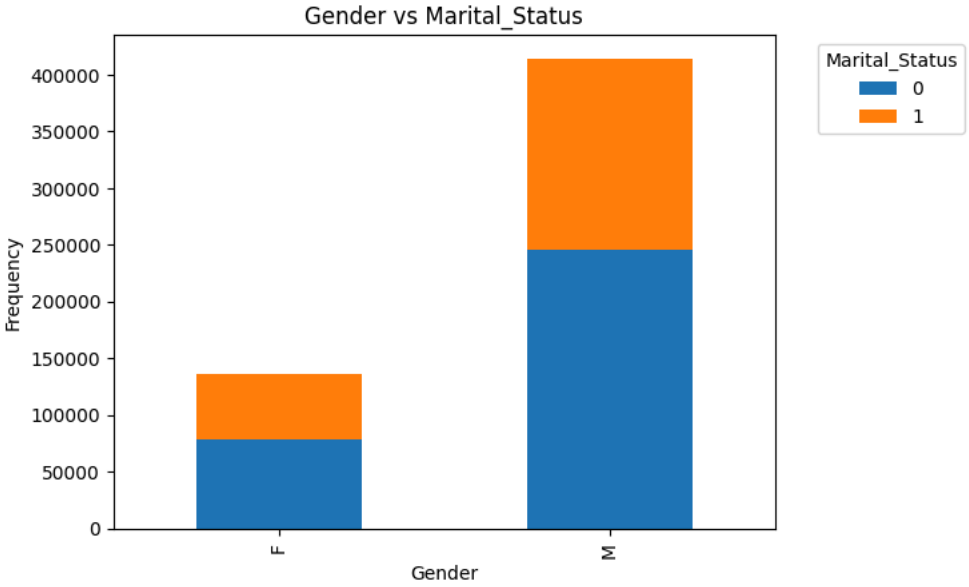

Frequency
200000
150000
100000
50000
0
0-17
18-25
26-35
36-45
46-50
51-55
55+
Age
Occupation
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

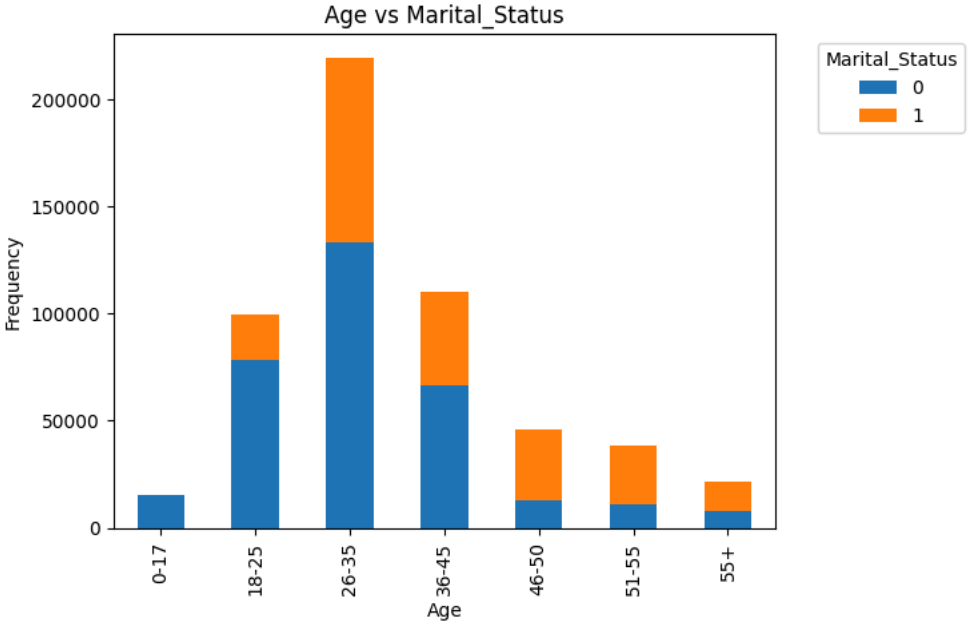

City_Category vs Occupation
Frequency
200000
150000
100000
50000
0
A
B
C
City_Category
Occupation
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

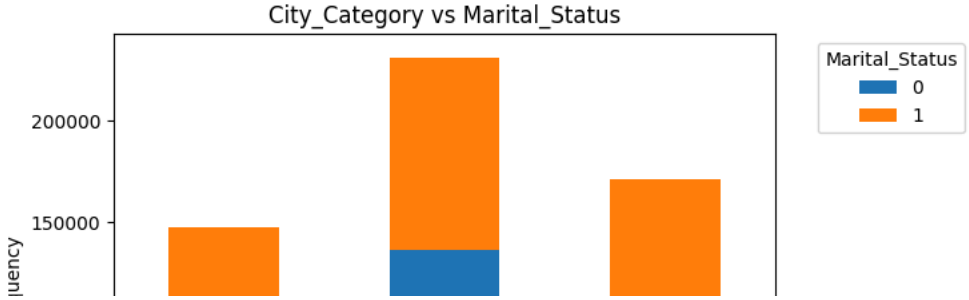

Stay_In_Current_City_Years vs Occupation
Frequency
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
2
3
4+
Stay_In_Current_City_Years
Occupation
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

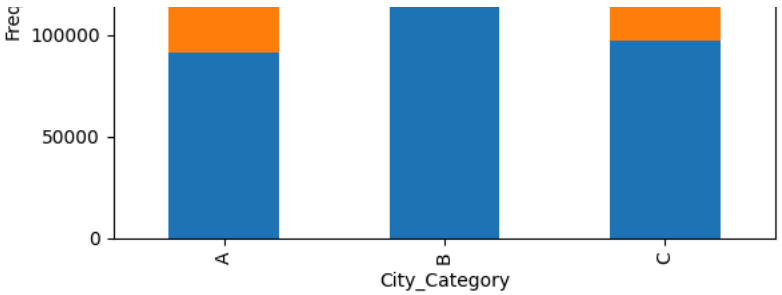

Product_Category vs Marital_Status

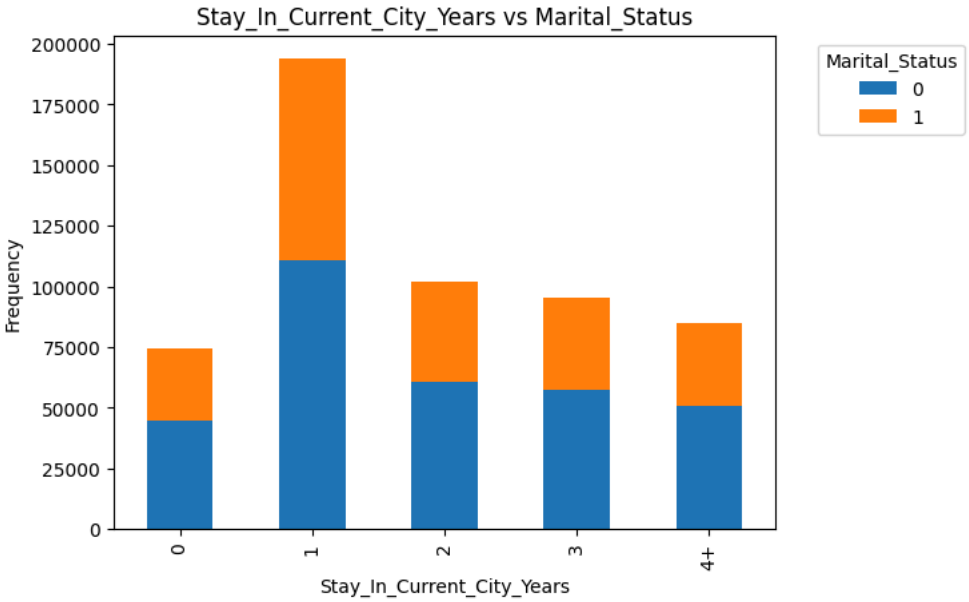

Marital_Status
0
1
Frequency
140000
120000
100000
80000
60000
40000
20000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Product_Category

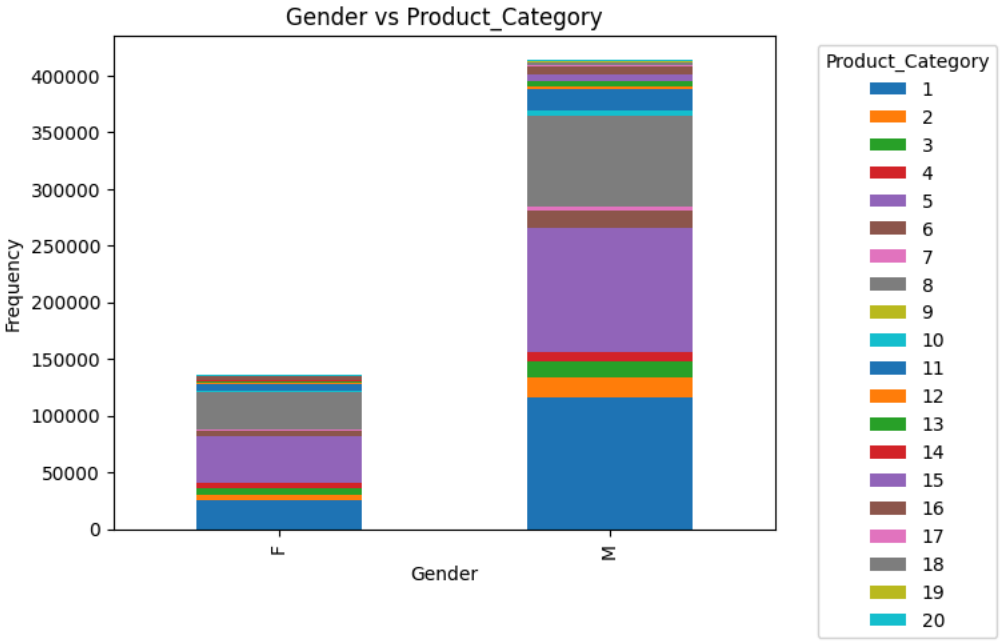

Gender vs Marital_Status
Marital_Status
0
1
Frequency
400000
350000
300000
250000
200000
150000
100000
50000
0
F
M
Gender

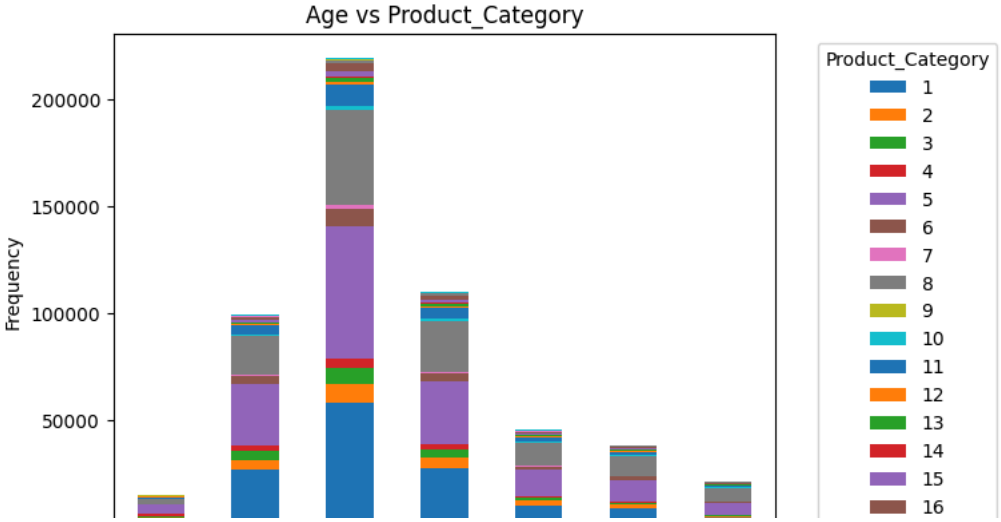

Age vs Marital_Status
Marital_Status
0
1
Frequency
200000
150000
100000
50000
0
0-17
18-25
26-35
36-45
46-50
51-55
55+
Age

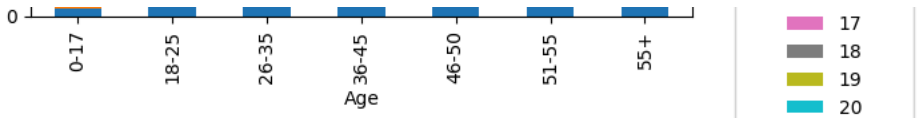

City_Category vs Marital_Status
Marital_Status
0
1
200000
150000

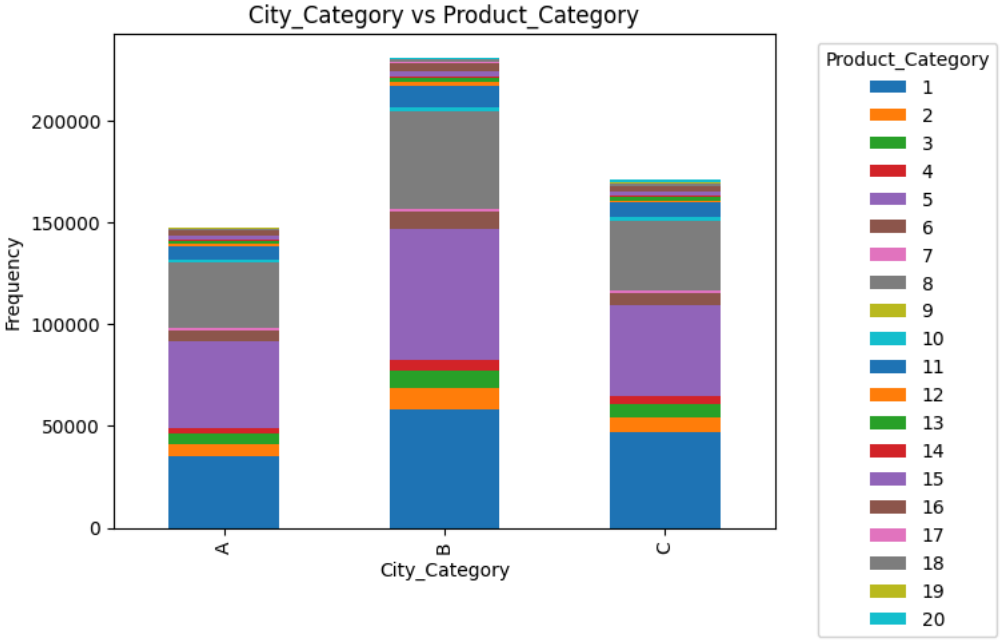

Freq
100000
50000
0
A
B
C
City_Category

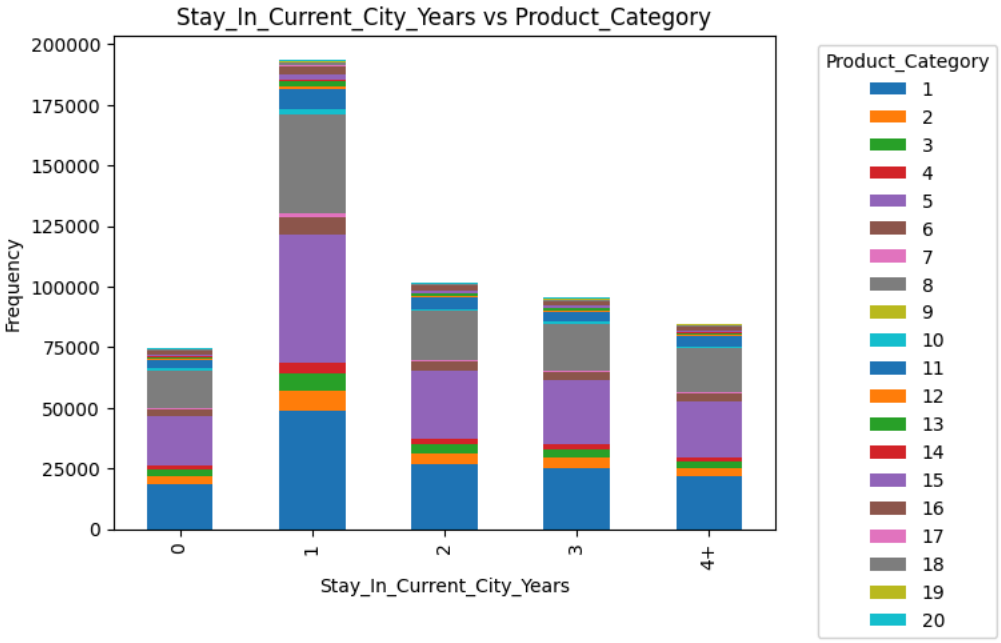

Stay_In_Current_City_Years vs Marital_Status
200000
175000
150000
125000
100000
75000
50000
25000
0
Frequency
0
1
2
3
4+
Stay_In_Current_City_Years
Marital_Status
0
1

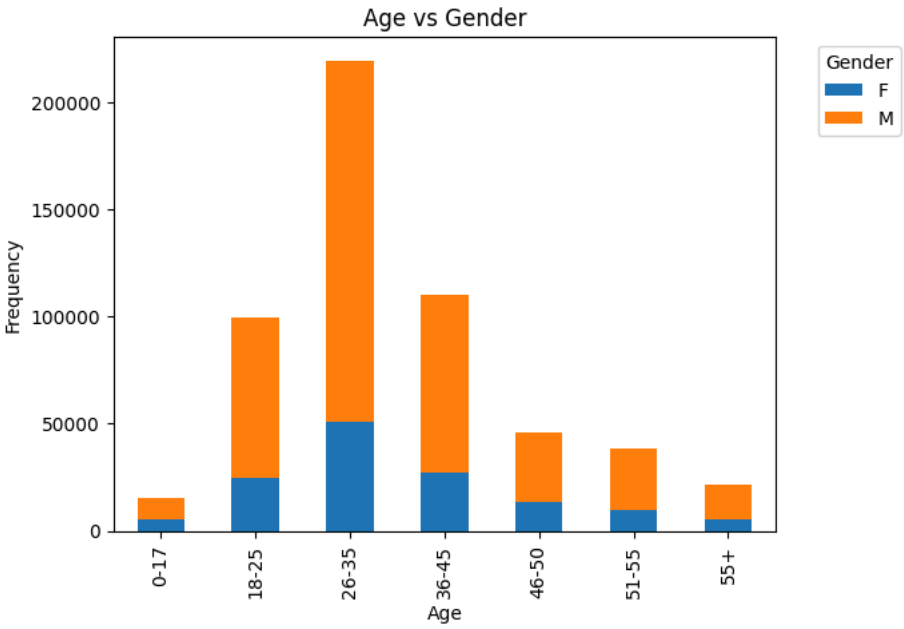

Gender vs Product_Category
400000
350000
300000
250000
200000
150000
100000
50000
0
Frequency
F
M
Gender
Product_Category
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

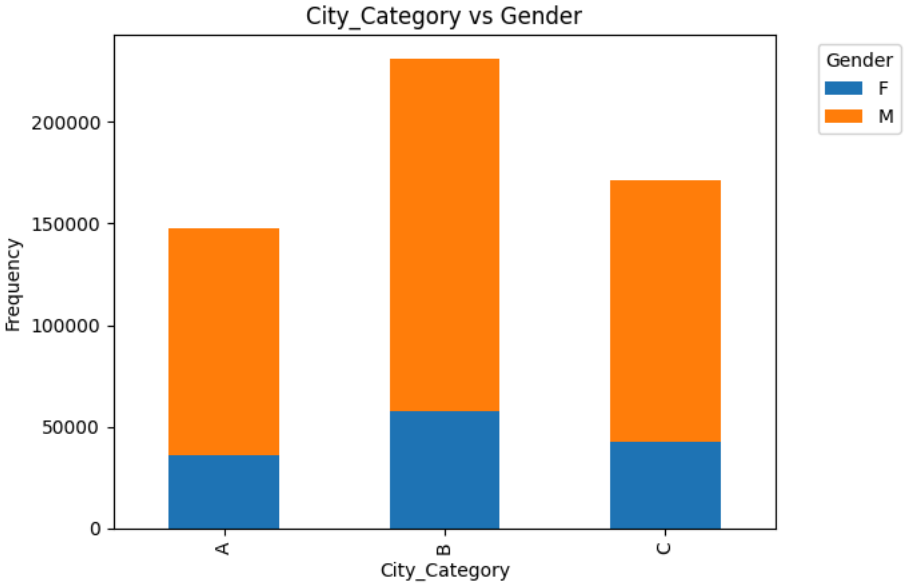

Age vs Product_Category
200000
150000
100000
50000
Frequency
Product_Category
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

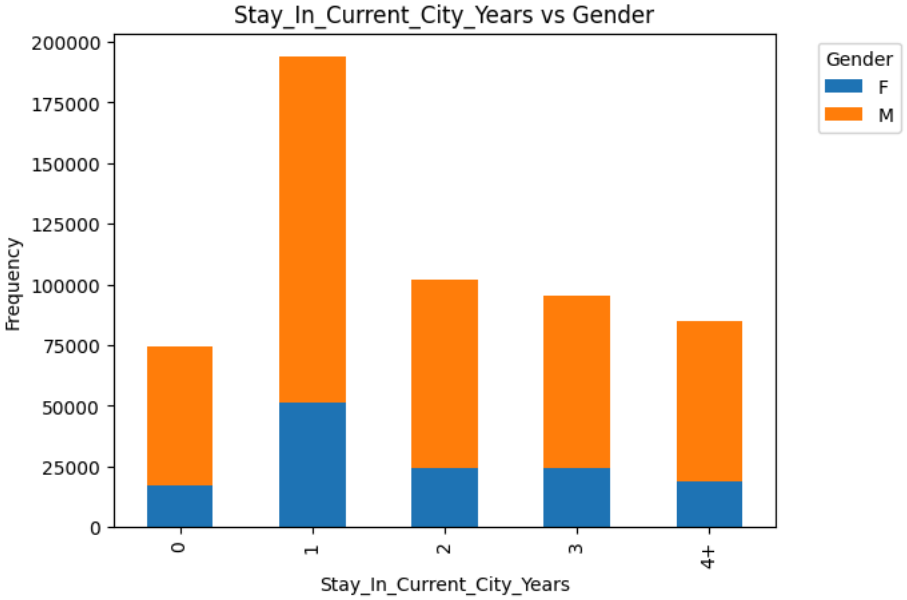

0
0-17
18-25
26-35
36-45
46-50
51-55
55+
Age
17
18
19
20

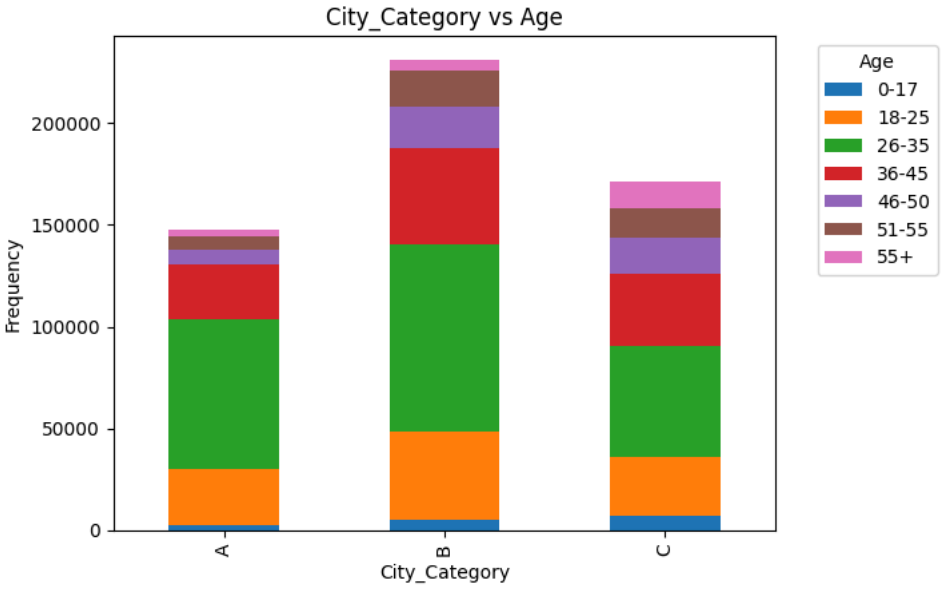

City_Category vs Product_Category
Frequency
200000
150000
100000
50000
0
A
B
C
City_Category
Product_Category
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

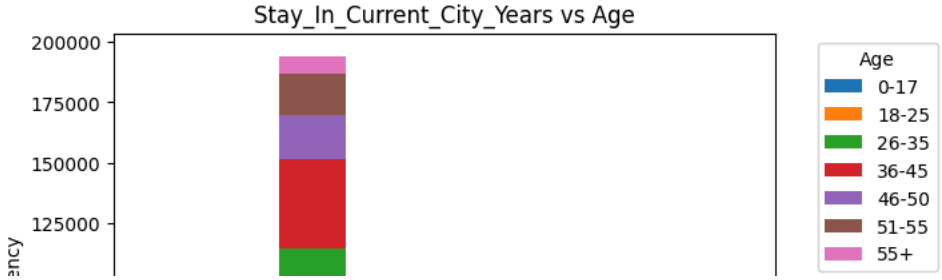

Stay_In_Current_City_Years vs Product_Category
Frequency
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
2
3
4+
Stay_In_Current_City_Years
Product_Category
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

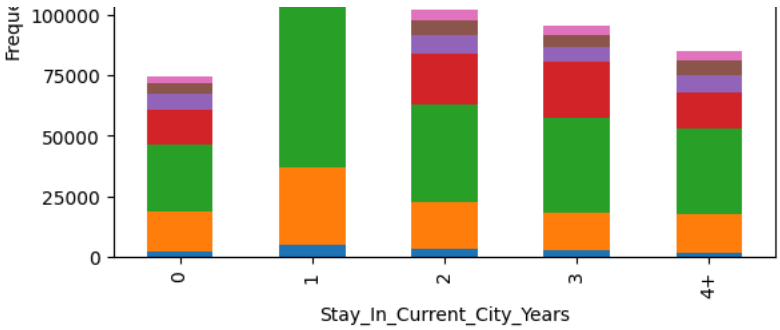

Age vs Gender
Frequency
200000
150000
100000
50000
0
0-17
18-25
26-35
36-45
46-50
51-55
55+
Age
Gender
F
M

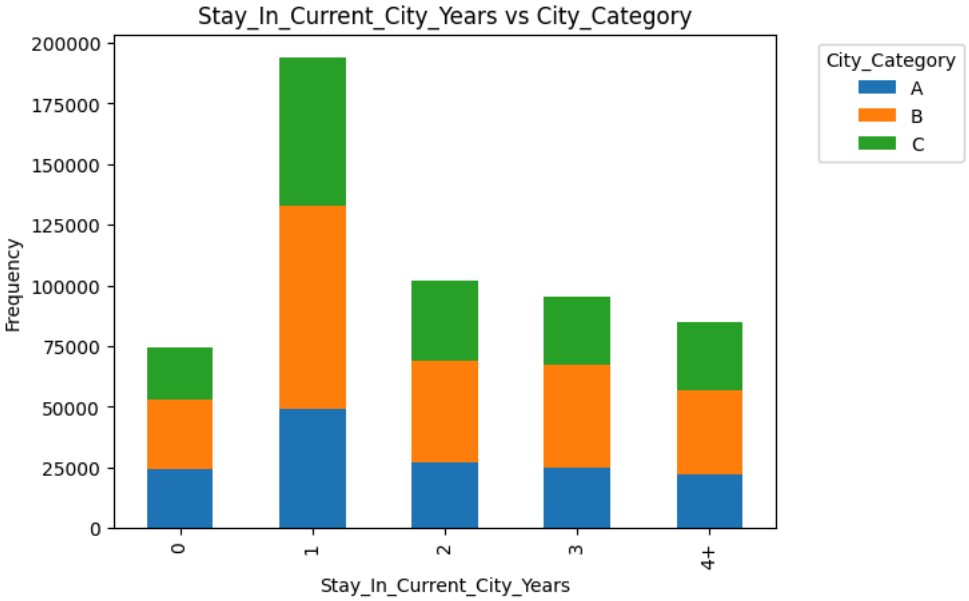

City_Category vs Gender
Gender
F
M
Frequency
200000
150000
100000
50000
0
A
B
C
City_Category


Stay_In_Current_City_Years vs Gender
Gender
F
M
Frequency
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
2
3
4+
Stay_In_Current_City_Years


City_Category vs Age
Age
0-17
18-25
26-35
36-45
46-50
51-55
55+
Frequency
200000
150000
100000
50000
0
A
B
C
City_Category


Stay_In_Current_City_Years vs Age
Age
0-17
18-25
26-35
36-45
46-50
51-55
55+
200000
175000
150000
125000


100000
75000
50000
25000
0
0
1
2
3
4+
Stay_In_Current_City_Years


Stay_In_Current_City_Years vs City_Category
200000
175000
150000
125000
100000
75000
50000
25000
0
Frequency
0
1
2
3
4+
Stay_In_Current_City_Years
City_Category
A
B
C

```
# Comments for each univariate and bivariate plot
def generate_comments(df):
    comments = []

    # Univariate comments
    for column in df.columns:
        comments.append(f"Univariate Plot ({column}):")
        if pd.api.types.is_numeric_dtype(df[column]):
            comments.append(f" - The {column} distribution spans from {df[column].min()} to {df[column].max()}.")
        else:
            comments.append(f" - The {column} distribution includes categories: {df[column].unique().tolist()}.")
        comments.append("\n")

    # Bivariate comments
    for i in range(len(df.columns)):
        for j in range(i + 1, len(df.columns)):
            x = df.columns[i]
            y = df.columns[j]
            comments.append(f"Bivariate Plot ({x} vs {y}):")
            comments.append(f" - Relationship analysis between {x} and {y}.")
            comments.append("\n")

    return "\n".join(comments)



# Print the comments on distribution and relationships
print(generate_comments(df))
```

```
Bivariate Plot (Occupation vs Stay_In_Current_City_Years):
 - Relationship analysis between Occupation and Stay_In_Current_City_Years.


Bivariate Plot (Occupation vs Marital_Status):
 - Relationship analysis between Occupation and Marital_Status.


Bivariate Plot (Occupation vs Product_Category):
 - Relationship analysis between Occupation and Product_Category.


Bivariate Plot (Occupation vs Purchase):
 - Relationship analysis between Occupation and Purchase.


Bivariate Plot (City_Category vs Stay_In_Current_City_Years):
 - Relationship analysis between City_Category and Stay_In_Current_City_Years.


Bivariate Plot (City_Category vs Marital_Status):
 - Relationship analysis between City_Category and Marital_Status.


Bivariate Plot (City_Category vs Product_Category):
 - Relationship analysis between City_Category and Product_Category.


Bivariate Plot (City_Category vs Purchase):
 - Relationship analysis between City_Category and Purchase.


Bivariate Plot (Stay_In_Current_City_Years vs Marital_Status):
 - Relationship analysis between Stay_In_Current_City_Years and Marital_Status.


Bivariate Plot (Stay_In_Current_City_Years vs Product_Category):
 - Relationship analysis between Stay_In_Current_City_Years and Product_Category.


Bivariate Plot (Stay_In_Current_City_Years vs Purchase):
 - Relationship analysis between Stay_In_Current_City_Years and Purchase.


Bivariate Plot (Marital_Status vs Product_Category):
 - Relationship analysis between Marital_Status and Product_Category.


Bivariate Plot (Marital_Status vs Purchase):
 - Relationship analysis between Marital_Status and Purchase.


Bivariate Plot (Product_Category vs Purchase):
 - Relationship analysis between Product_Category and Purchase.
```

```
df.dtypes
```

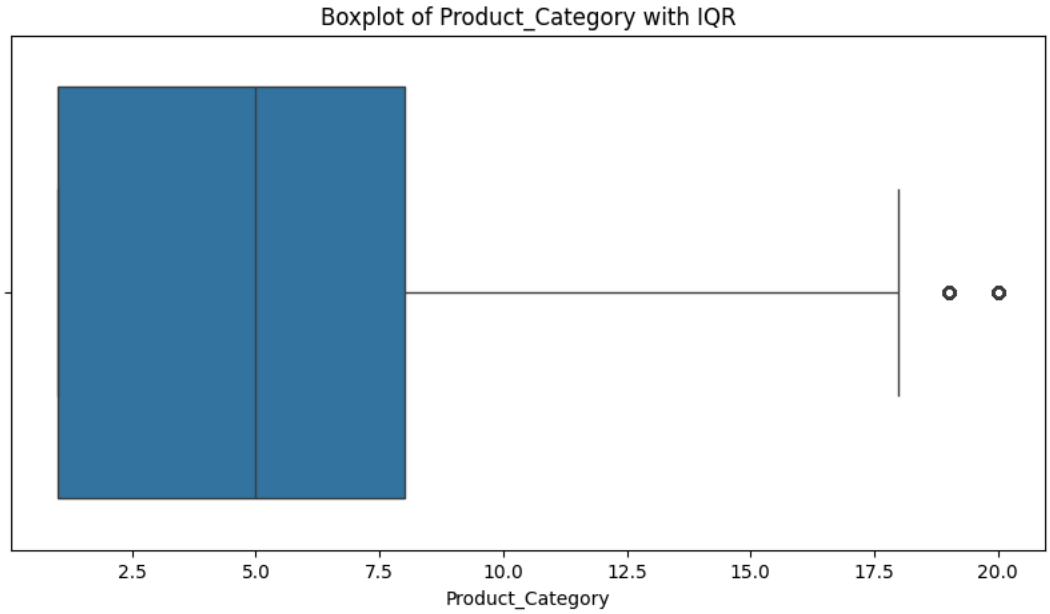
	0
User_ID	int64
Product_ID	category
Gender	category
Age	category
Occupation	int64
City_Category	category
Stay_In_Current_City_Years	category
Marital_Status	int64
Product_Category	int64
Purchase	int64

dtype: object

2.Detect Null values and outliers

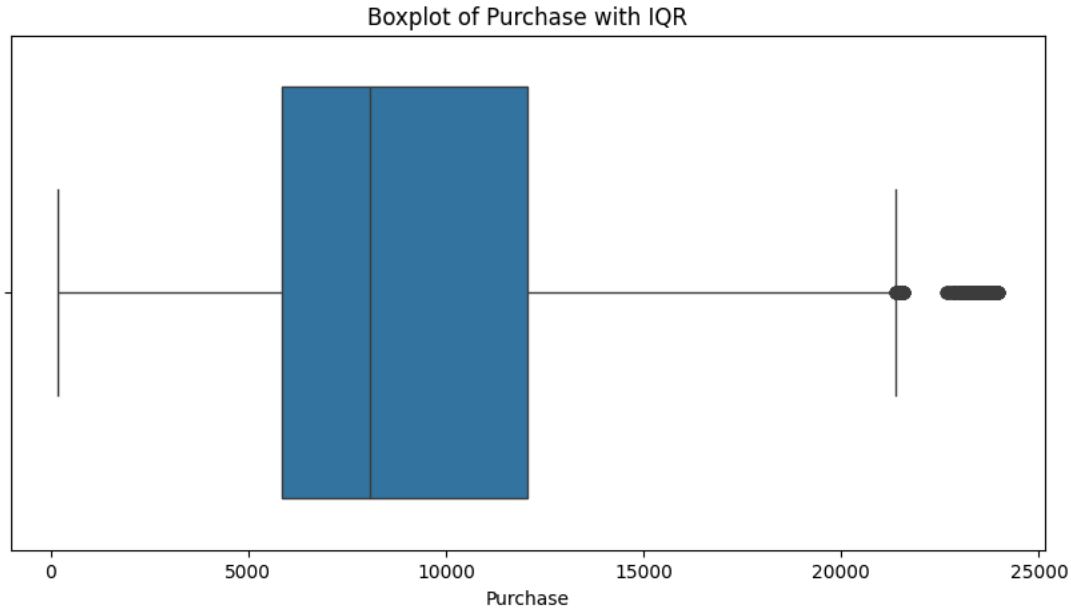
```
# a) List of continuous columns
continuous_columns = ['User_ID', 'Occupation', 'Marital_Status', 'Product_Category', 'Purchase']

# IQR Method to find outliers and plot boxplots
for column in continuous_columns:
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    print(column)
    print("Q1: ",Q1)
    print("Q3: ",Q3)
    print("IQR: ",IQR)
    print("Lower Bound: ", lower_bound)
    print("Upper Bound: ", upper_bound)
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    print(f"Outliers in {column} (IQR Method): ")
    if outliers.empty:
      print("No,Outliers Detected for ", column , "\n")
    else:
      print(outliers)
      plt.figure(figsize=(10, 5))
      sns.boxplot(x=df[column])
      plt.title(f'Boxplot of {column} with IQR')
      plt.show()

      df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

      # Print the new dataframe without outliers
      print("Dataframe without outliers for", column)
      print(df)
```

```
User_ID
Q1:  1001516.0
Q3:  1004478.0
IQR:  2962.0
Lower Bound:  997073.0
Upper Bound:  1008921.0
Outliers in User_ID (IQR Method):
No,Outliers Detected for  User_ID

Occupation
Q1:  2.0
Q3:  14.0
IQR:  12.0
Lower Bound:  -16.0
Upper Bound:  32.0
Outliers in Occupation (IQR Method):
No,Outliers Detected for  Occupation

Marital_Status
Q1:  0.0
Q3:  1.0
IQR:  1.0
Lower Bound:  -1.5
Upper Bound:  2.5
Outliers in Marital_Status (IQR Method):
No,Outliers Detected for  Marital_Status

Product_Category
Q1:  1.0
Q3:  8.0
IQR:  7.0
Lower Bound:  -9.5
Upper Bound:  18.5
Outliers in Product_Category (IQR Method):
        User_ID Product_ID Gender    Age  Occupation City_Category  \
545915  1000001  P00375436      F   0-17          10             A
545916  1000002  P00372445      M    55+          16             C
545917  1000004  P00375436      M  46-50           7             B
545918  1000006  P00375436      F  51-55           9             A
545919  1000007  P00372445      M  36-45           1             B
...         ...        ...    ...    ...         ...           ...
550063  1006033  P00372445      M  51-55          13             B
550064  1006035  P00375436      F  26-35           1             C
550065  1006036  P00375436      F  26-35          15             B
550066  1006038  P00375436      F    55+           1             C
550067  1006039  P00371644      F  46-50           0             B

       Stay_In_Current_City_Years  Marital_Status  Product_Category  Purchase
545915                          2               0                20       612
545916                         4+               0                20       119
545917                          2               1                20       481
545918                          1               0                20       480
545919                          1               1                20       241
...                           ...             ...               ...       ...
550063                          1               1                20       368
550064                          3               0                20       371
550065                         4+               1                20       137
550066                          2               0                20       365
550067                         4+               1                20       490

[4153 rows x 10 columns]
```




```
Dataframe without outliers for Product_Category
        User_ID Product_ID Gender    Age  Occupation City_Category  \
```

```
0       1000001  P00069042      F   0-17          10             A
1       1000001  P00248942      F   0-17          10             A
2       1000001  P00087842      F   0-17          10             A
3       1000001  P00085442      F   0-17          10             A
4       1000002  P00285442      M    55+          16             C
...         ...        ...    ...    ...         ...           ...
545910  1006040  P00184342      M  26-35           6             B
545911  1006040  P00193142      M  26-35           6             B
545912  1006040  P00029842      M  26-35           6             B
545913  1006040  P00106042      M  26-35           6             B
545914  1006040  P00217442      M  26-35           6             B

       Stay_In_Current_City_Years  Marital_Status  Product_Category  Purchase
0                               2               0                 3      8370
1                               2               0                 1     15200
2                               2               0                12      1422
3                               2               0                12      1057
4                              4+               0                 8      7969
...                           ...             ...               ...       ...
545910                          2               0                 8      9855
545911                          2               0                 5      1962
545912                          2               0                 8      7852
545913                          2               0                 5      7159
545914                          2               0                 1     11640

[545915 rows x 10 columns]
Purchase
Q1:  5867.0
Q3:  12072.0
IQR:  6205.0
Lower Bound:  -3440.5
Upper Bound:  21379.5
Outliers in Purchase (IQR Method):
        User_ID Product_ID Gender    Age  Occupation City_Category  \
343     1000058  P00117642      M  26-35           2             B
375     1000062  P00119342      F  36-45           3             A
652     1000126  P00087042      M  18-25           9             B
736     1000139  P00159542      F  26-35          20             C
1041    1000175  P00052842      F  26-35           2             B
...         ...        ...    ...    ...         ...           ...
544704  1005847  P00085342      F  18-25           4             B
544743  1005852  P00202242      F  26-35           1             A
545663  1006002  P00116142      M  51-55           0             C
545787  1006018  P00052842      M  36-45           1             C
545864  1006036  P00111042      F  26-35          15             B

       Stay_In_Current_City_Years  Marital_Status  Product_Category  Purchase
343                             3               0                10     23603
375                             1               0                10     23792
652                             1               0                10     23233
736                             2               0                10     23595
1041                            1               0                10     23341
...                           ...             ...               ...       ...
544704                          2               0                10     23724
544743                          0               1                10     23529
545663                          1               1                10     23663
545787                          3               0                10     23496
545864                         4+               1                15     21390

[2705 rows x 10 columns]
```




```
Dataframe without outliers for Purchase
        User_ID Product_ID Gender    Age  Occupation City_Category  \
0       1000001  P00069042      F   0-17          10             A
1       1000001  P00248942      F   0-17          10             A
```

```
1       1000001  P00248942      F   0-17          10             A
2       1000001  P00087842      F   0-17          10             A
3       1000001  P00085442      F   0-17          10             A
4       1000002  P00285442      M    55+          16             C
...         ...        ...    ...    ...         ...           ...
545910  1006040  P00184342      M  26-35           6             B
545911  1006040  P00193142      M  26-35           6             B
545912  1006040  P00029842      M  26-35           6             B
545913  1006040  P00106042      M  26-35           6             B
545914  1006040  P00217442      M  26-35           6             B

       Stay_In_Current_City_Years  Marital_Status  Product_Category  Purchase
0                               2               0                 3      8370
1                               2               0                 1     15200
2                               2               0                12      1422
3                               2               0                12      1057
4                              4+               0                 8      7969
...                           ...             ...               ...       ...
545910                          2               0                 8      9855
545911                          2               0                 5      1962
545912                          2               0                 8      7852
545913                          2               0                 5      7159
545914                          2               0                 1     11640

[543210 rows x 10 columns]
```

Insights:

1) User_ID:

- Q1 (25th percentile): 1001516.0
- Q3 (75th percentile): 1004478.0
- IQR (Interquartile Range): 2962.0
- Lower Bound: 997073.0
- Upper Bound: 1008921.0
- Outliers: No outliers detected.

2) Occupation:

- Q1 (25th percentile): 2.0
- Q3 (75th percentile): 14.0
- IQR (Interquartile Range): 12.0
- Lower Bound: -16.0
- Upper Bound: 32.0
- Outliers: No outliers detected.

3) Marital_Status:

- Q1 (25th percentile): 0.0
- Q3 (75th percentile): 1.0
- IQR (Interquartile Range): 1.0
- Lower Bound: -1.5
- Upper Bound: 2.5
- Outliers: No outliers detected.

4) Product_Category:

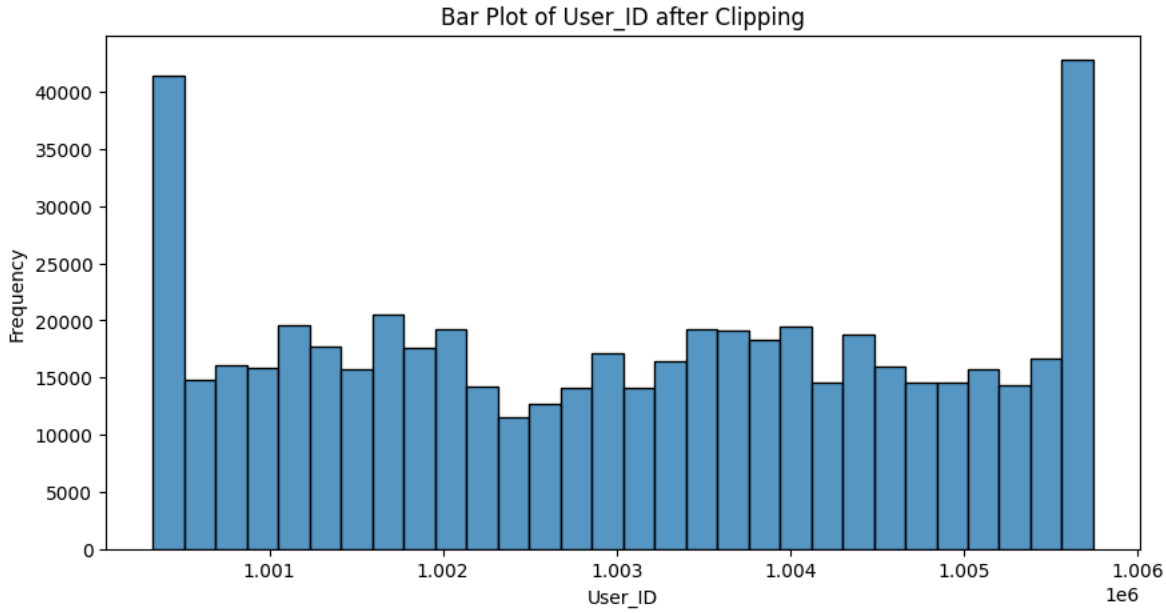
- Q1 (25th percentile): 1.0
- Q3 (75th percentile): 8.0
- IQR (Interquartile Range): 7.0
- Lower Bound: -9.5
- Upper Bound: 18.5
- Outliers: Detected (Product_Category = 20).

5) Purchase:

- Q1 (25th percentile): 5823.0
- Q3 (75th percentile): 12054.0
- IQR (Interquartile Range): 6231.0
- Lower Bound: -3523.5
- Upper Bound: 21400.5
- Outliers: Detected (Purchase > 21400.5)

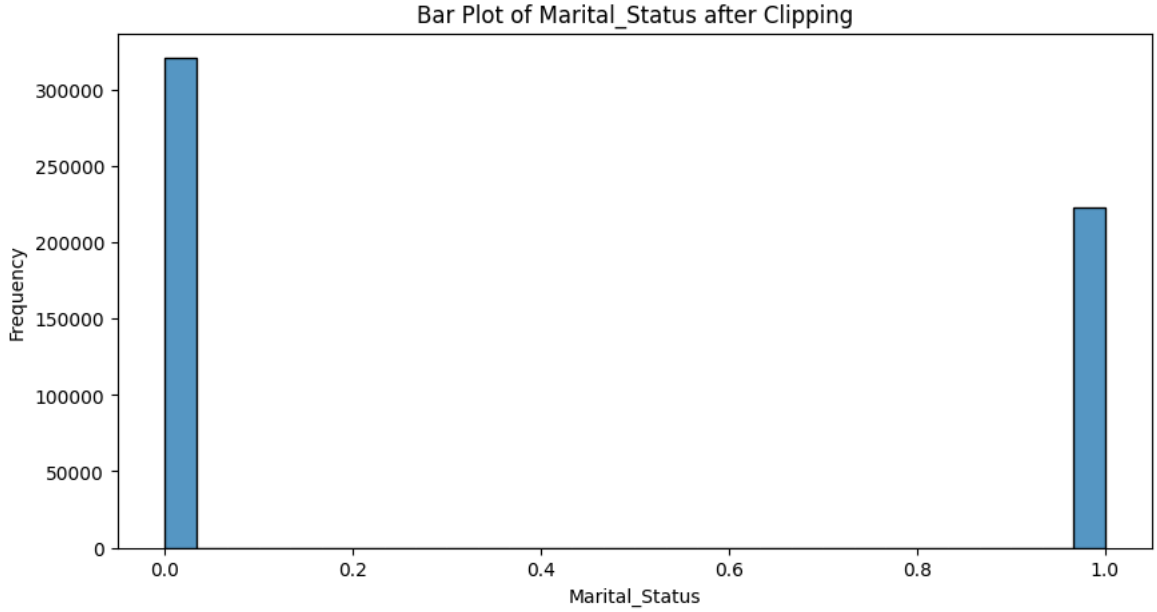
```
#b) Remove/clip the data between the 5 percentile and 95 percentile

# Clip the data between the 5th percentile and 95th percentile
for column in continuous_columns:
  lower_bound = np.percentile(df[column], 5)
  upper_bound = np.percentile(df[column], 95)
  print(column,"lower bound :",lower_bound)
  print(column,"upper bound :",upper_bound)
  df[column] = np.clip(df[column], lower_bound, upper_bound)
  # Plot the results as bar plots
  plt.figure(figsize=(10, 5))
  sns.histplot(df[column], bins=30, kde=False)
  plt.title(f'Bar Plot of {column} after Clipping')
  plt.xlabel(column)
  plt.ylabel('Frequency')
  plt.show()
```

```
User_ID lower bound : 1000329.0
User_ID upper bound : 1005747.0
```



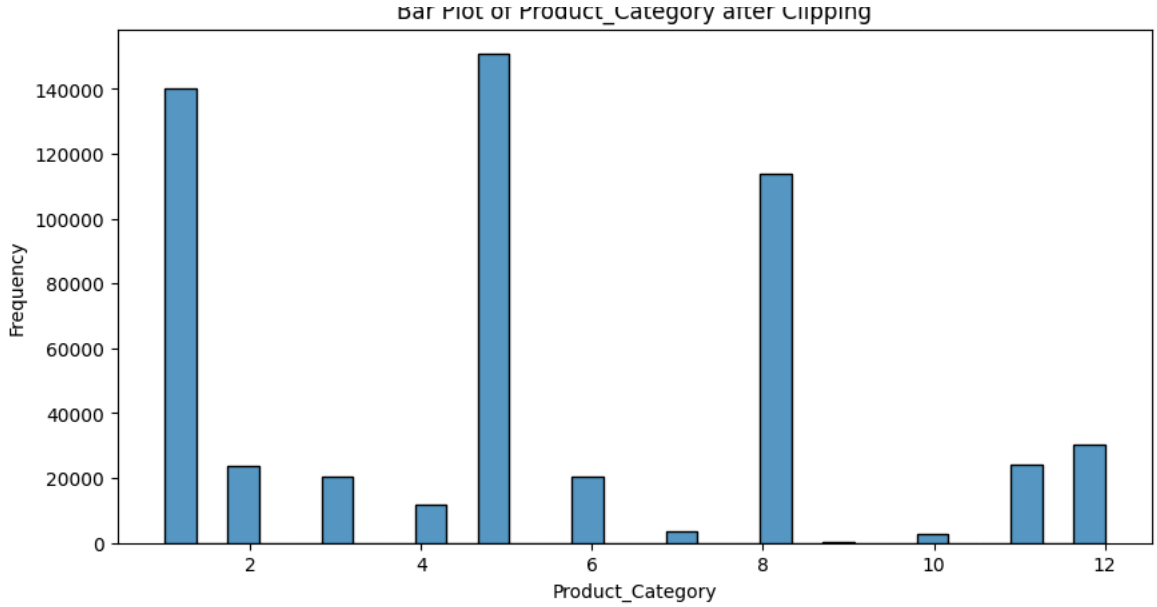

```
Occupation lower bound : 0.0
Occupation upper bound : 20.0
```


```
Marital_Status lower bound : 0.0
Marital_Status upper bound : 1.0
```

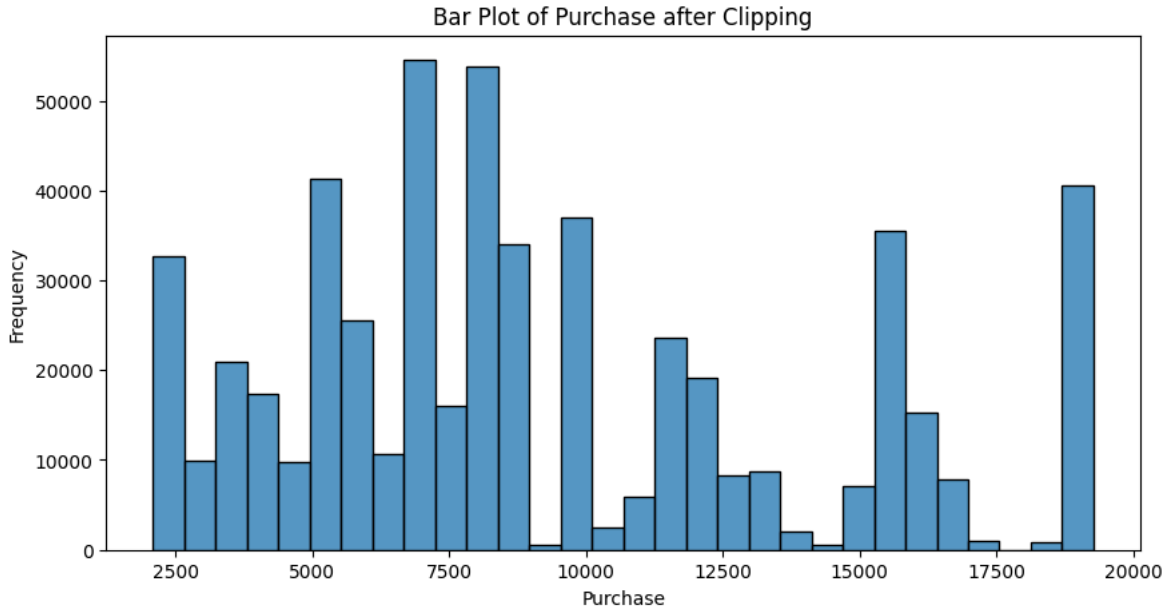



```
Product_Category lower bound : 1.0
Product_Category upper bound : 12.0
```

Bar Plot of Product_Category after Clipping




```
Purchase lower bound : 2091.0
Purchase upper bound : 19274.0
```



🔍 Insights After Clipping Data:

1) User_ID:

- Range: 1000329.0 to 1005747.0
- Insight: User IDs within this range are considered typical. Any User_ID outside this range would be considered an outlier.

2) Occupation:

- Range: 0.0 to 20.0
- Insight: Occupation values within this range are typical. Occupation values outside this range would be considered outliers. This suggests that the dataset includes a variety of occupations, but none are expected to have a value higher than 20.

3) Marital_Status:

- Range: 0.0 to 1.0
- Insight: Marital status values are binary (0 or 1), indicating whether a user is married or not. Any value outside this range would be an error or outlier.

4) Product_Category:

- Range: 1.0 to 12.0
- Insight: Product categories are expected to fall within this range. Values outside this range would be considered outliers, indicating either an error in data entry or a product category that is not typical.

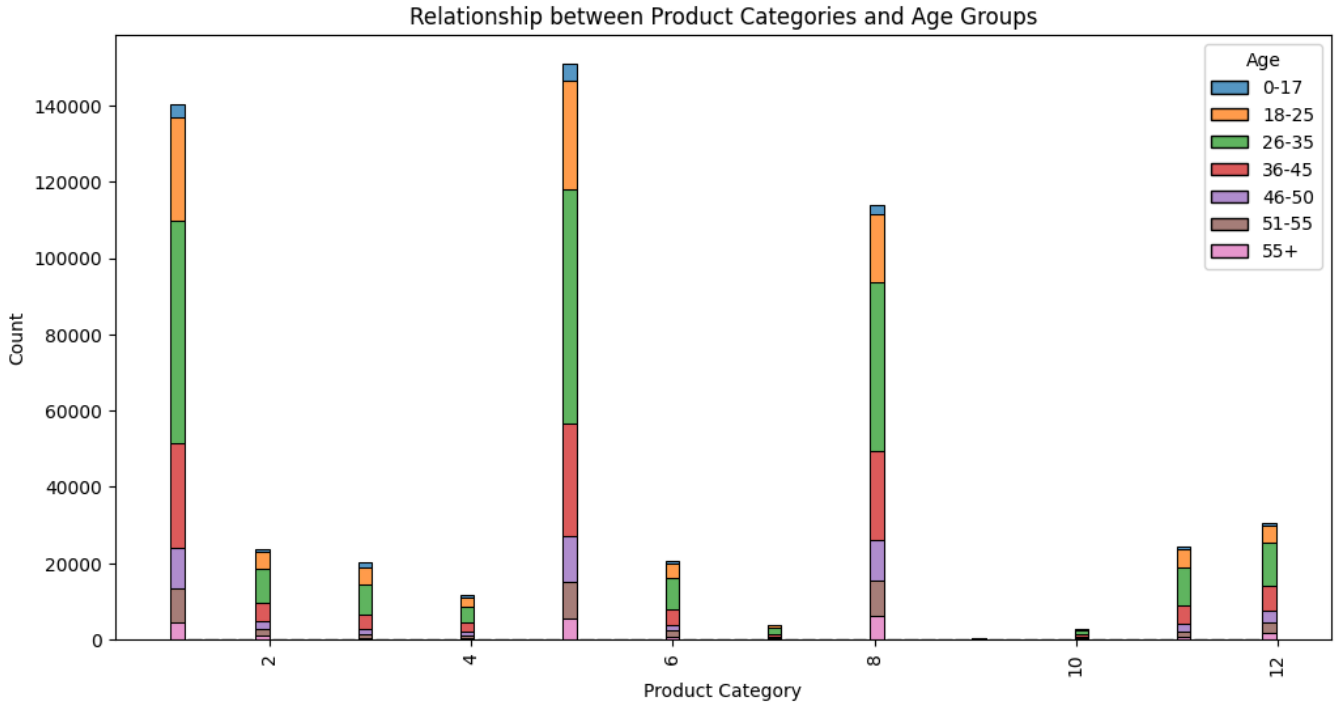
5) Purchase:

- Range: 2091.0 to 19274.0
- Insight: Purchase amounts within this range are typical. Any purchase amount outside this range would be considered an outlier, indicating either an unusually low or high purchase amount.

3. 📊 Data Exploration

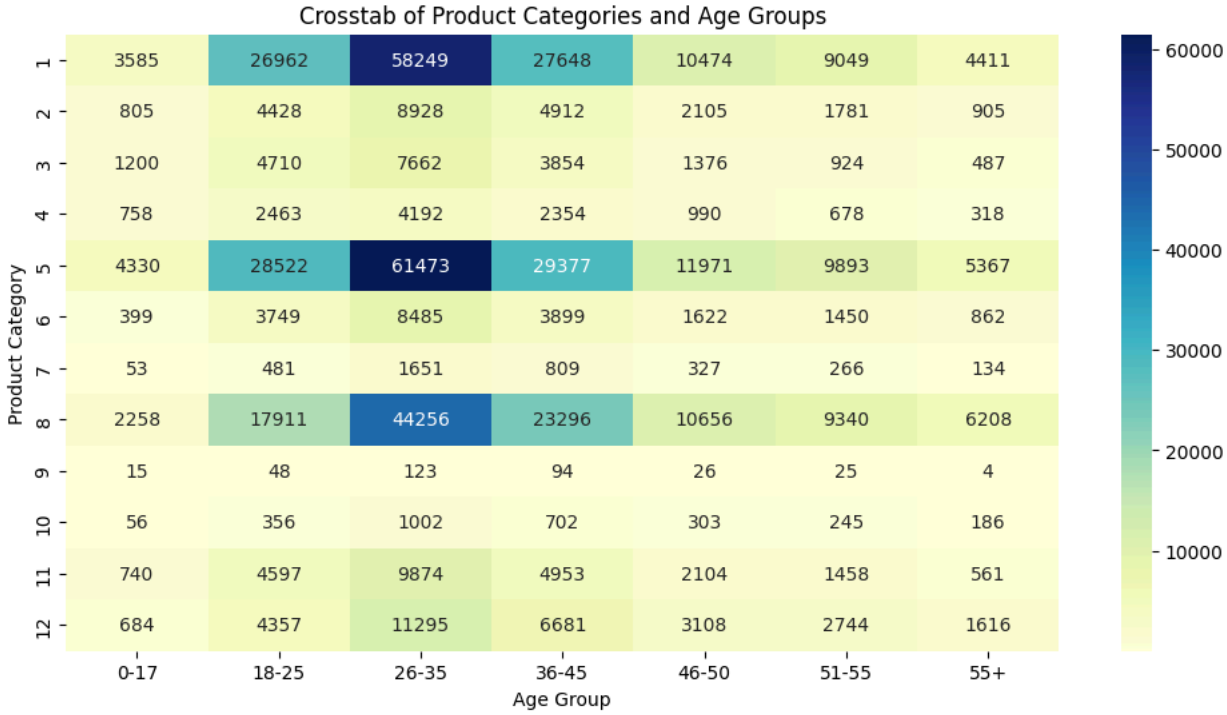
a) What products are different age groups buying?

```
#i) visual analysis against showing relationship between product categories and age groups.

plt.figure(figsize=(12, 6))
sns.histplot(data=df, x='Product_Category', hue='Age', multiple='stack', shrink=0.8)
plt.title('Relationship between Product Categories and Age Groups')
plt.xlabel('Product Category')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.show()
```



```
# Analysis against showing lationship between product categories and age groups.
crosstab = pd.crosstab(df['Product_Category'], df['Age'])

# Display the crosstab
print(crosstab)

plt.figure(figsize=(12, 6))
sns.heatmap(crosstab, annot=True, fmt='d', cmap='YlGnBu')
plt.title('Crosstab of Product Categories and Age Groups')
plt.xlabel('Age Group')
plt.ylabel('Product Category')
plt.show()
```

Age	0-17	18-25	26-35	36-45	46-50	51-55	55+
Product_Category							
1	3585	26962	58249	27648	10474	9049	4411
2	805	4428	8928	4912	2105	1781	905
3	1200	4710	7662	3854	1376	924	487
4	758	2463	4192	2354	990	678	318
5	4330	28522	61473	29377	11971	9893	5367
6	399	3749	8485	3899	1622	1450	862
7	53	481	1651	809	327	266	134
8	2258	17911	44256	23296	10656	9340	6208
9	15	48	123	94	26	25	4
10	56	356	1002	702	303	245	186
11	740	4597	9874	4953	2104	1458	561
12	684	4357	11295	6681	3108	2744	1616



Insight:

1) Product Category 1 and 5:

- These categories are the most popular across all age groups, with the highest counts in the 26-35 age group.
- The 18-25 and 36-45 age groups also show significant interest in these categories.

2) Product Category 8:

- This category is also quite popular, especially among the 26-35 and 36-45 age groups.
- The 18-25 age group shows a notable interest as well.

3) Product Category 2 and 3:

- These categories have moderate popularity, with the highest counts in the 26-35 age group.
- The 18-25 and 36-45 age groups also contribute significantly to the counts.

4) Product Category 6:

- This category shows a similar trend to categories 2 and 3, with the highest counts in the 26-35 age group.
- The 18-25 and 36-45 age groups also show interest.

5) Product Category 7 and 9:

- These categories have the lowest counts across all age groups.

- They are least popular among all age groups, with very low counts in the 0-17 and 55+ age groups.

6) Product Category 10 and 12:

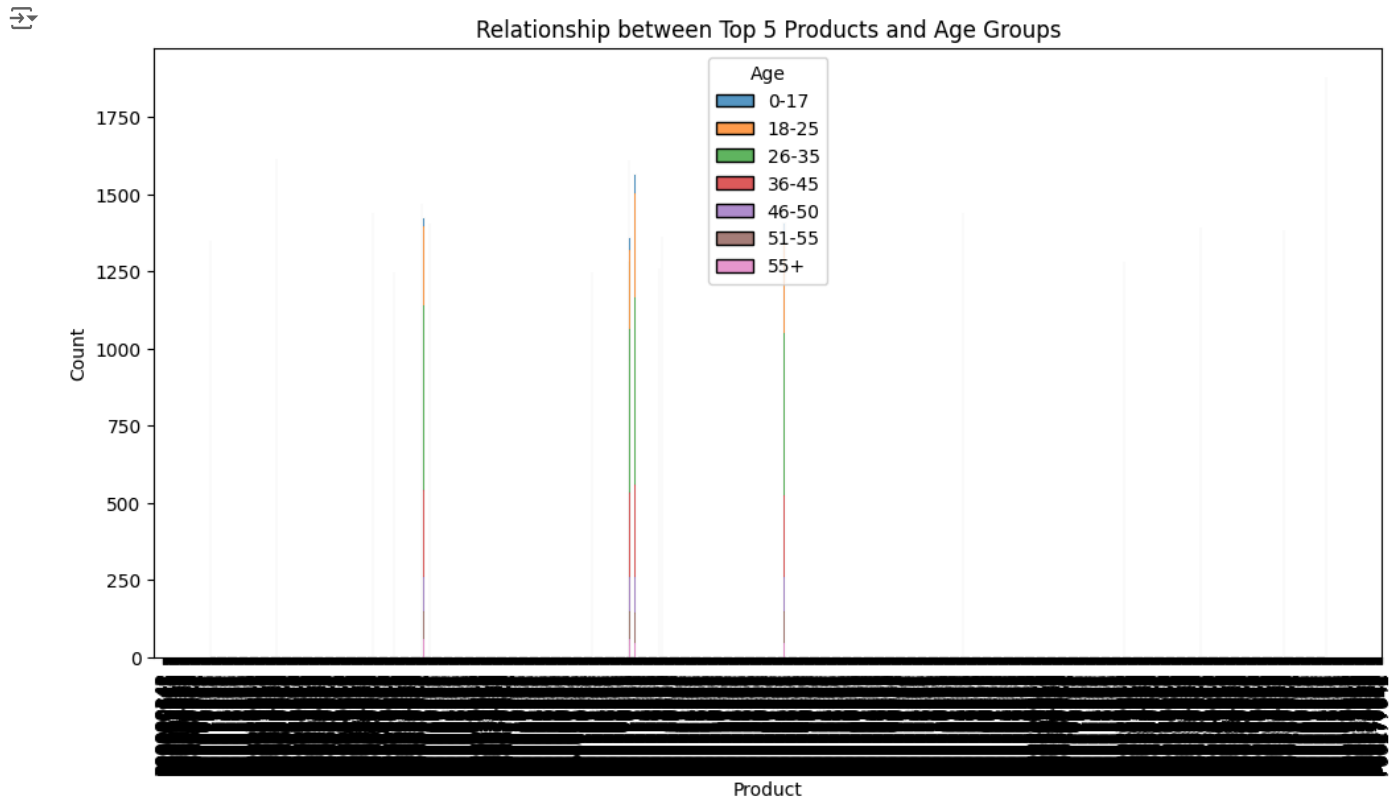
- These categories have relatively low counts, with the highest counts in the 26-35 age group.
- The 18-25 and 36-45 age groups also show some interest.

7) Product Category 11 and 13:

- These categories have moderate popularity, with the highest counts in the 26-35 age group.
- The 18-25 and 36-45 age groups also contribute significantly to the counts.

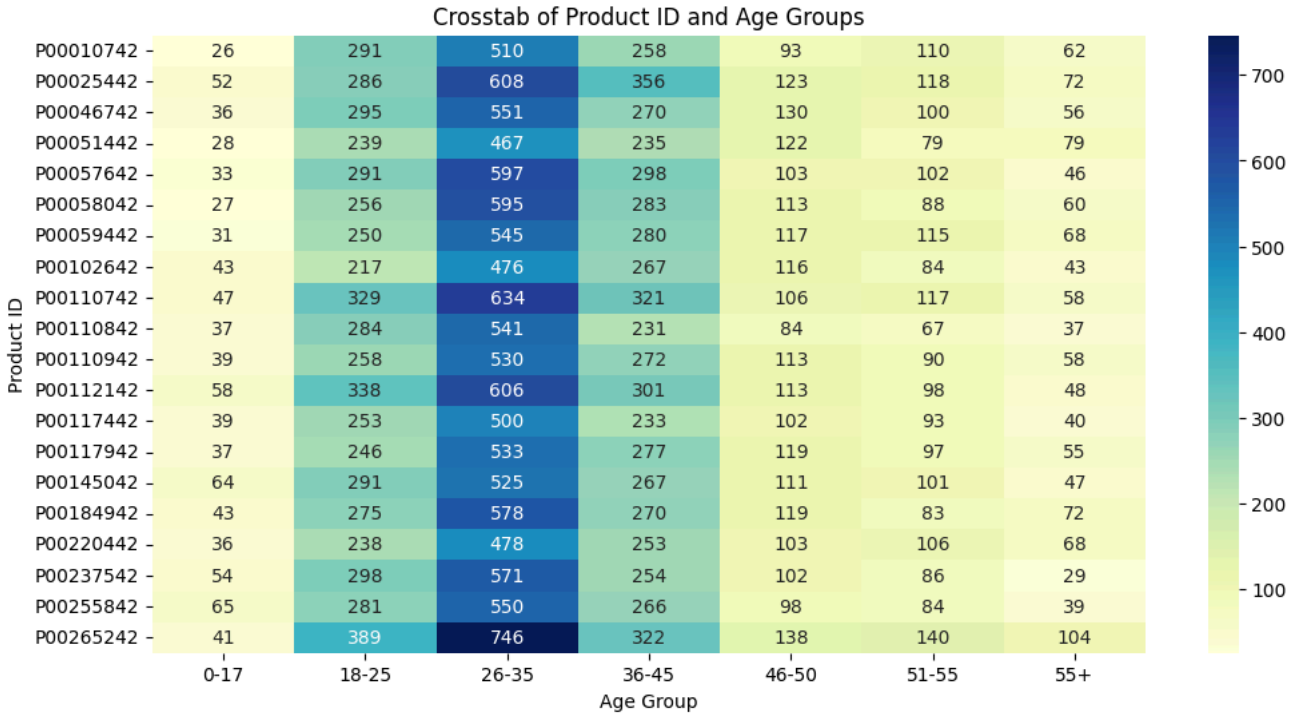
Overall, the 26-35 age group appears to be the most active in purchasing across various product categories, followed by the 18-25 and 36-45 age groups. The 0-17 and 55+ age groups generally show lower counts across most product categories.

```
#ii) visual analysis to show relationship between product_ids and age groups:

# Get the top 20 most frequent products as there is huge count for product id which can create readability issue
top_products = df['Product_ID'].value_counts().nlargest(20).index
filtered_data = df[df['Product_ID'].isin(top_products)]



plt.figure(figsize=(12, 6))
sns.histplot(data=filtered_data, x='Product_ID', hue='Age', multiple='stack', shrink=0.8)
plt.title('Relationship between Top 5 Products and Age Groups')
plt.xlabel('Product')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.show()
```




```
# Get the top 20 most frequent products
# Analysis to show relationship between product_ids and age groups using cross tab to have clear understanding:

top_products = df['Product_ID'].value_counts().nlargest(20).index
filtered_data = df[df['Product_ID'].isin(top_products)]
crosstab = pd.crosstab(filtered_data['Product_ID'], df['Age'])

# Display the crosstab
print(crosstab)

plt.figure(figsize=(12, 6))
sns.heatmap(crosstab, annot=True, fmt='d', cmap='YlGnBu')
plt.title('Crosstab of Product ID and Age Groups')
plt.xlabel('Age Group')
plt.ylabel('Product ID')
plt.show()
```

Product_ID	0-17	18-25	26-35	36-45	46-50	51-55	55+
P00010742	26	291	510	258	93	110	62
P00025442	52	286	608	356	123	118	72
P00046742	36	295	551	270	130	100	56
P00051442	28	239	467	235	122	79	79
P00057642	33	291	597	298	103	102	46
P00058042	27	256	595	283	113	88	60
P00059442	31	250	545	280	117	115	68
P00102642	43	217	476	267	116	84	43
P00110742	47	329	634	321	106	117	58
P00110842	37	284	541	231	84	67	37
P00110942	39	258	530	272	113	90	58
P00112142	58	338	606	301	113	98	48
P00117442	39	253	500	233	102	93	40
P00117942	37	246	533	277	119	97	55
P00145042	64	291	525	267	111	101	47
P00184942	43	275	578	270	119	83	72
P00220442	36	238	478	253	103	106	68
P00237542	54	298	571	254	102	86	29
P00255842	65	281	550	266	98	84	39
P00265242	41	389	746	322	138	140	104



Insights:

1) General Trends:

- The 26-35 age group consistently shows the highest counts across most product IDs, indicating they are the most active buyers.
- The 18-25 and 36-45 age groups also show significant purchasing activity, though generally lower than the 26-35 age group.

2) Popular Products:

- P00110742: Popular product, especially among the 26-35 and 18-25 age groups.
- P00265242: This product also has the highest counts across all age groups, particularly in the 26-35 age group, followed by the 18-25 and 36-45 age groups.

3) Moderately Popular Products:

- P00025442, P00046742, P00057642, P00058042, P00059442, P00110942, and - - P00112142: These products have moderate counts, with the highest activity in the 26-35 age group.
- P00145042 and P00255842: These products also show moderate popularity, with notable counts in the 26-35 and 18-25 age groups.

4) Less Popular Products:

- P00010742, P00051442, P00102642, P00110842, P00117442, P00117942, - - - - P00184942, P00220442, and P00237542: These products have lower counts across all age groups, but still show the highest activity in the 26-35 age group.

5) Age Group Specific Trends:

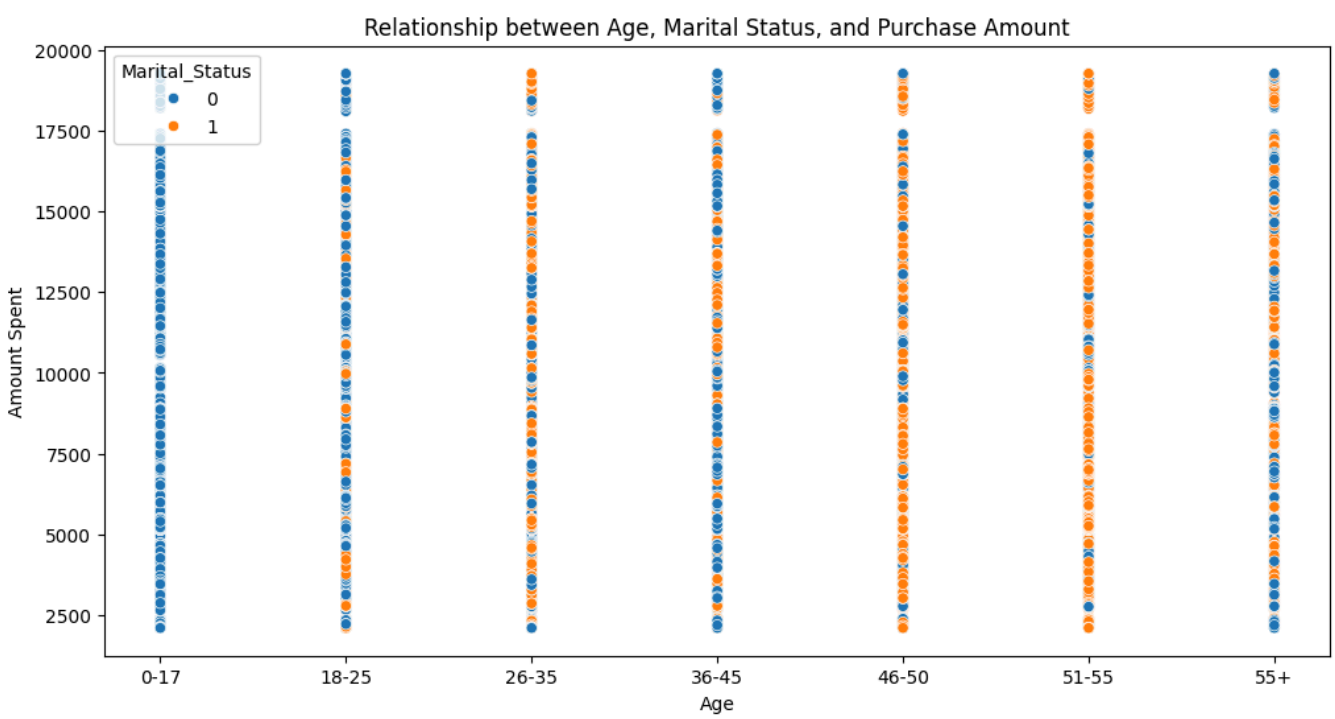
- The 0-17 and 55+ age groups generally show the lowest counts across all product IDs, indicating they are less active in purchasing these products.

- The 46-50 and 51-55 age groups have moderate activity, with some products showing higher counts in these age groups.

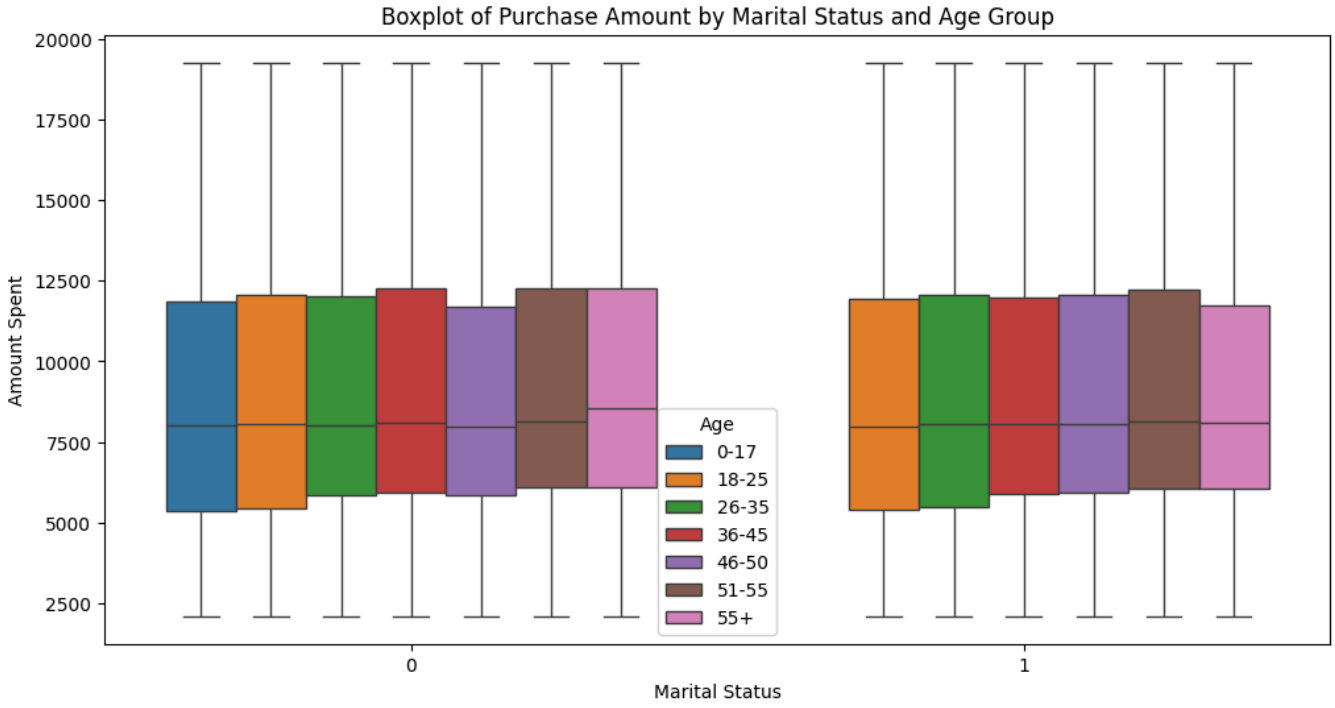
Overall, the 26-35 age group is the most active in purchasing across various product IDs, followed by the 18-25 and 36-45 age groups. The 0-17 and 55+ age groups show lower purchasing activity.

b) Is there a relationship between age, marital status, and the amount spent?

```
# scatter plot: This plot will help visualize the relationship between age, marital status, and the Purcahse amount .
# let assume ; 0=unmarried and 1=married

plt.figure(figsize=(12, 6))
sns.scatterplot(data=df, x='Age', y='Purchase', hue='Marital_Status')
plt.title('Relationship between Age, Marital Status, and Purchase Amount')
plt.xlabel('Age')
plt.ylabel('Amount Spent')
plt.show()
```




```
# Box plot: This plot will help visualize the relationship between age, marital status, and the Purcahse amount .
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='Marital_Status', y='Purchase', hue='Age')
plt.title('Boxplot of Purchase Amount by Marital Status and Age Group')
plt.xlabel('Marital Status')
plt.ylabel('Amount Spent')
plt.show()
```

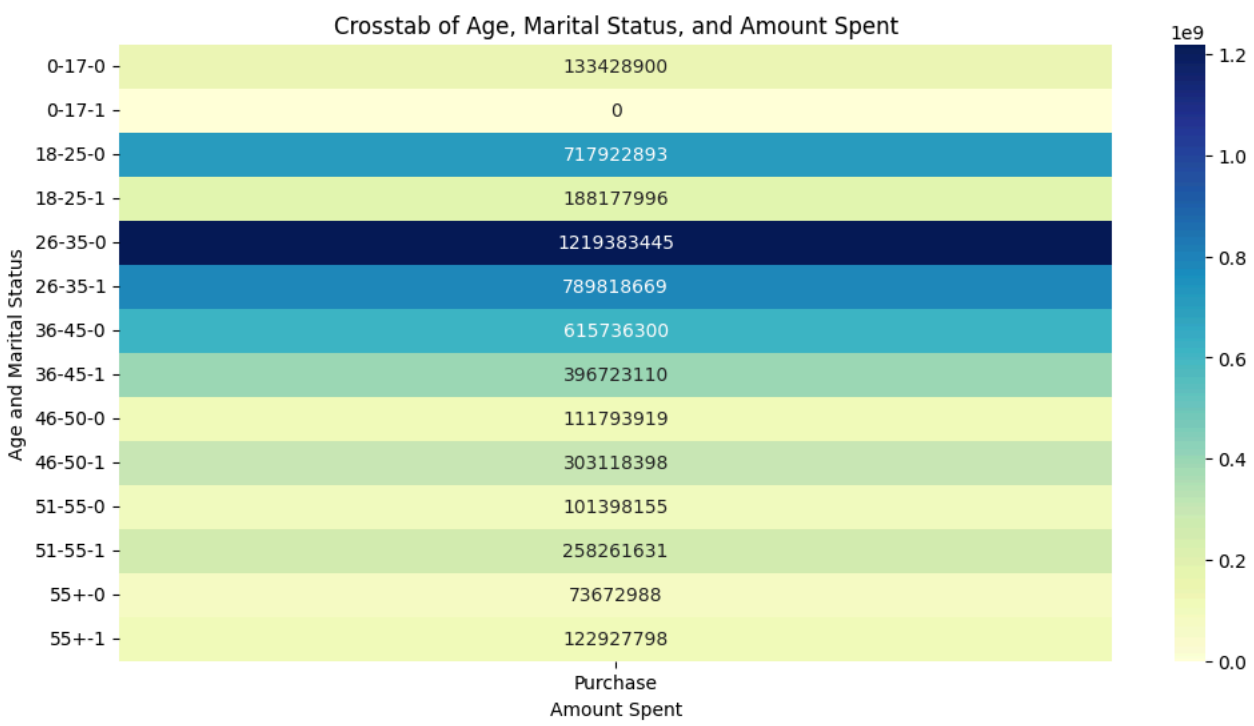



```
#Showing analysis using cross tab for best understanding
crosstab = pd.crosstab(index=[df['Age'], df['Marital_Status']], columns='Purchase', values=df['Purchase'], aggfunc='sum')

crosstab
```

	col_0	Purchase
Age	Marital_Status	
0-17	0	133428900
	1	0
18-25	0	717922893
	1	188177996
26-35	0	1219383445
	1	789818669
36-45	0	615736300
	1	396723110
46-50	0	111793919
	1	303118398
51-55	0	101398155
	1	258261631
55+	0	73672988
	1	122927798

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

```
plt.figure(figsize=(12, 6))
sns.heatmap(crosstab, annot=True, fmt='d', cmap='YlGnBu')
plt.title('Crosstab of Age, Marital Status, and Amount Spent')
plt.xlabel('Amount Spent')
plt.ylabel('Age and Marital Status')
plt.show()
```



Insights:

1) 0-17 Age Group:

- Only unmarried individuals are spending, with a total amount of 133,428,900.

2) 18-25 Age Group:

- Unmarried individuals spend significantly more (717,922,893) compared to married individuals (188,177,996).

3) 26-35 Age Group:

- Both married (78,918,669) and unmarried (1,219,383,445) individuals have high spending, with unmarried individuals spending more.

4) 36-45 Age Group:

- Unmarried individuals spend more (615,736,300) compared to married individuals (396,723,110).

5) 46-50 Age Group:

- Married individuals spend more (303,118,398) compared to unmarried individuals (111,793,919).

6) 51-55 Age Group:

- Married individuals spend more (258,261,631) compared to unmarried individuals (101,398,155).

7) 55+ Age Group:

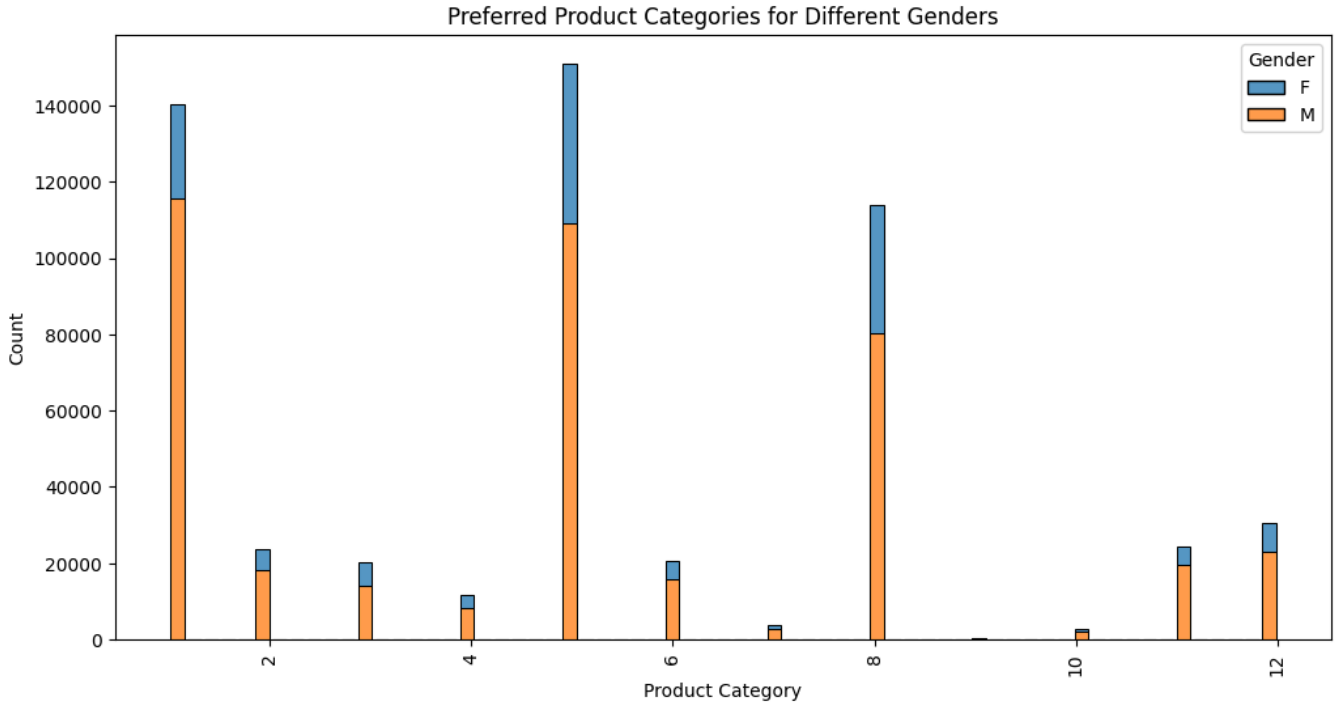
- Married individuals spend more (122,927,798) compared to unmarried individuals (73,672,798).

```
Key Insights:
```

- Unmarried individuals generally spend more than married individuals in the younger age groups (0-45).
- Married individuals tend to spend more in the older age groups (46+).
- The 26-35 age group is the most active in terms of spending, regardless of marital status.
- The 18-25 and 36-45 age groups also show significant spending, with unmarried individuals leading in these groups.

c) Are there preferred product categories for different genders?

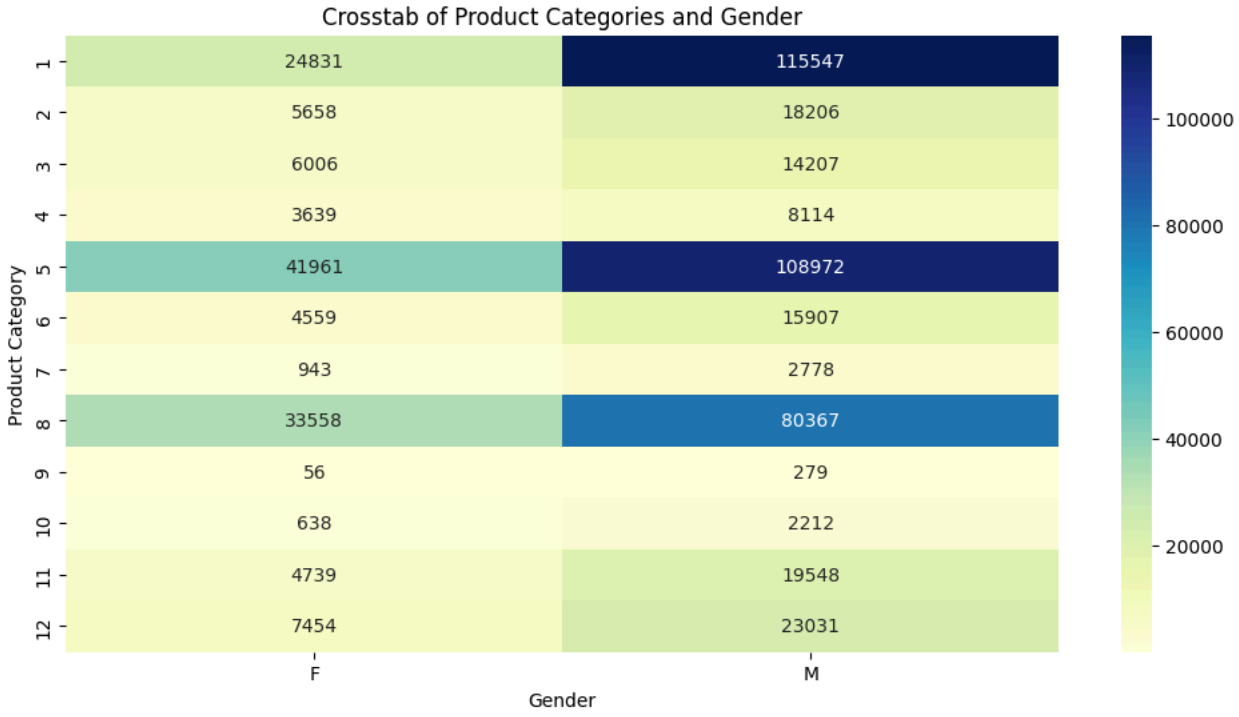
```
#visual analysis
plt.figure(figsize=(12, 6))
sns.histplot(data=df, x='Product_Category', hue='Gender', multiple='stack', shrink=0.8)
plt.title('Preferred Product Categories for Different Genders')
plt.xlabel('Product Category')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.show()
```




```
#crosstab for clear understanding

# Create a crosstab of product categories and gender
crosstab = pd.crosstab(df['Product_Category'], df['Gender'])

# Display the crosstab
print(crosstab)
```

```
Gender                F       M
Product_Category
1                 24831  115547
2                  5658   18206
3                  6006   14207
4                  3639    8114
5                 41961  108972
6                  4559   15907
7                   943    2778
8                 33558   80367
9                    56     279
10                  638    2212
11                 4739   19548
12                 7454   23031
```

```
plt.figure(figsize=(12, 6))
sns.heatmap(crosstab, annot=True, fmt='d', cmap='YlGnBu')
plt.title('Crosstab of Product Categories and Gender')
plt.xlabel('Gender')
plt.ylabel('Product Category')
plt.show()
```



Insights:

1) Product Category 1 and 5:

- These categories are the most popular among both genders. Males have significantly higher counts (115,547 for Category 1 and 108,972 for Category 5) compared to females (24,831 for Category 1 and 41,961 for Category 5).

2) Product Category 8:

- This category is also quite popular among both genders. Males (80,367) have higher counts compared to females (33,558).

3) Product Category 2, 3, and 6:

- These categories have moderate popularity. Males have higher counts (18,206 for Category 2, 14,207 for Category 3, and 15,907 for Category 6) compared to females (5,658 for Category 2, 6,006 for Category 3, and 4,559 for Category 6).

4) Product Category 4 and 11:

- These categories are less popular but still show notable counts. Males have higher counts (8,114 for Category 4 and 19,548 for Category 11) compared to females (3,639 for Category 4 and 4,739 for Category 11).

5) Product Category 7, 9, 10, and 12:

- These categories have lower counts overall. Males have higher counts (2,778 for Category 7, 340 for Category 9, 3,963 for Category 10, and 2,415 for Category 12) compared to females (943 for Category 7, 70 for Category 9, 1,162 for Category 10, and 1,532 for Category 12).

6) Product Category 13:

- This category has moderate popularity. Males (23,895) have higher counts compared to females (7,151).

`Key Insights:`

- Males generally have higher counts across all product categories compared to females.
- Product Categories 1, 5, and 8 are the most popular among both genders, with males showing significantly higher counts.
- Product Categories 2, 3, 6, and 13 also show notable preferences, with males leading in counts.

4) How does gender affect the amount spent?

```
# Check for missing or non-numeric values in the Purchase column
print(df['Purchase'].isnull().sum())
print(df['Purchase'].dtype)

# Remove or fill missing values
data = df.dropna(subset=['Purchase'])

# Ensure Purchase is numeric
```

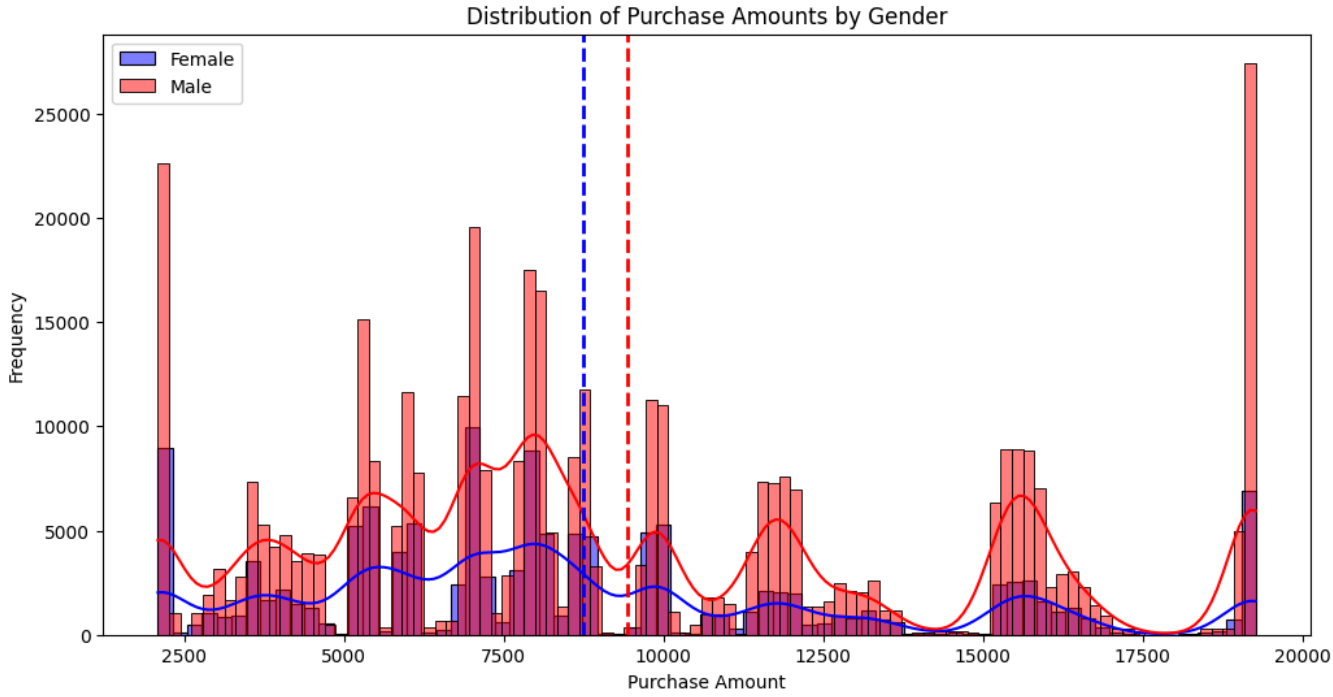
```
df['Purchase'] = pd.to_numeric(df['Purchase'], errors='coerce')
df = df.dropna(subset=['Purchase'])
```

```
0
int64
```

```
# Calculate the average purchase amount per transaction for women and men
average_purchase_women = df[df['Gender'] == 'F']['Purchase'].mean()
average_purchase_men = df[df['Gender'] == 'M']['Purchase'].mean()

# Print the results
print(f"Average purchase amount per transaction for women: {average_purchase_women}")
print(f"Average purchase amount per transaction for men: {average_purchase_men}")

# Determine if women are spending more money per transaction than men
if average_purchase_women > average_purchase_men:
    print("Women are spending more money per transaction than men.")
else:
    print("Men are spending more money per transaction than women.")
```

```
Average purchase amount per transaction for women: 8751.951209322451
Average purchase amount per transaction for men: 9431.908551010833
Men are spending more money per transaction than women.
```

```
# Plot the distribution of the mean of the expenses by female and male customers
plt.figure(figsize=(12, 6))
sns.histplot(df[df['Gender'] == 'F']['Purchase'], kde=True, color='blue', label='Female')
sns.histplot(df[df['Gender'] == 'M']['Purchase'], kde=True, color='red', label='Male')
plt.axvline(average_purchase_women, color='blue', linestyle='dashed', linewidth=2)
plt.axvline(average_purchase_men, color='red', linestyle='dashed', linewidth=2)
plt.title('Distribution of Purchase Amounts by Gender')
plt.xlabel('Purchase Amount')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Determine if women are spending more money per transaction than men
if average_purchase_women > average_purchase_men:
    print("Women are spending more money per transaction than men.")
else:
    print("Men are spending more money per transaction than women.")
```




```
Men are spending more money per transaction than women.
```

```
# Function to compute bootstrap confidence intervals with 90% C.I.
def bootstrap_ci_mf_90(data, n_bootstrap=1000, ci=90):
    means = []
    for _ in range(n_bootstrap):
        sample = np.random.choice(data, size=len(data), replace=True)
        means.append(np.mean(sample))
    lower_bound = np.percentile(means, (100 - ci) / 2)
    upper_bound = np.percentile(means, 100 - (100 - ci) / 2)
    return lower_bound, upper_bound
```

```
    return lower_bound, upper_bound

# Compute confidence intervals for the entire dataset
ci_male_90 = bootstrap_ci_mf_90(df[df['Gender'] == 'M']['Purchase'])
ci_female_90 = bootstrap_ci_mf_90(df[df['Gender'] == 'F']['Purchase'])

print(f"90% Confidence Interval for Males (entire dataset): {ci_male_90}")
print(f"90% Confidence Interval for Females (entire dataset): {ci_female_90}")
```

```
90% Confidence Interval for Males (entire dataset): (9420.079540799868, 9444.573239476205)
90% Confidence Interval for Females (entire dataset): (8731.758872592174, 8773.560684710763)
```

```
# Function to compute bootstrap confidence intervals with 95% C.I.
def bootstrap_ci_mf(data, n_bootstrap=1000, ci=95):
    means = []
    for _ in range(n_bootstrap):
        sample = np.random.choice(data, size=len(data), replace=True)
        means.append(np.mean(sample))
    lower_bound = np.percentile(means, (100 - ci) / 2)
    upper_bound = np.percentile(means, 100 - (100 - ci) / 2)
    return lower_bound, upper_bound


# Compute confidence intervals for the entire dataset
ci_male_95 = bootstrap_ci_mf(df[df['Gender'] == 'M']['Purchase'])
ci_female_95 = bootstrap_ci_mf(df[df['Gender'] == 'F']['Purchase'])

print(f"95% Confidence Interval for Males (entire dataset): {ci_male_95}")
print(f"95% Confidence Interval for Females (entire dataset): {ci_female_95}")
```

```
95% Confidence Interval for Males (entire dataset): (9416.456199409533, 9446.420252561296)
95% Confidence Interval for Females (entire dataset): (8725.703804031573, 8775.460945636441)
```

```
# Function to compute bootstrap confidence intervals with 99% C.I.
def bootstrap_ci_mf_99(data, n_bootstrap=1000, ci=99):
    means = []
    for _ in range(n_bootstrap):
        sample = np.random.choice(data, size=len(data), replace=True)
        means.append(np.mean(sample))
    lower_bound = np.percentile(means, (100 - ci) / 2)
    upper_bound = np.percentile(means, 100 - (100 - ci) / 2)
    return lower_bound, upper_bound


# Compute confidence intervals for the entire dataset
ci_male_99 = bootstrap_ci_mf_99(df[df['Gender'] == 'M']['Purchase'])
ci_female_99 = bootstrap_ci_mf_99(df[df['Gender'] == 'F']['Purchase'])


print(f"99% Confidence Interval for Males (entire dataset): {ci_male_99}")
print(f"99% Confidence Interval for Females (entire dataset): {ci_female_99}")
```

```
99% Confidence Interval for Males (entire dataset): (9411.855245534842, 9452.327865925487)
99% Confidence Interval for Females (entire dataset): (8721.095511817193, 8783.991668954506)
```

Insights:

1) Confidence Interval for Males:

- The 90% confidence interval for the average amount spent by males is approximately (9419.61, 9444.18).
- The 95% confidence interval for the average amount spent by males is approximately (9417.19, 9446.81).
- The 99% confidence interval for the average amount spent by males is approximately (9411.59, 9452.12).
- These intervals indicate that we are 90%, 95%, and 99% confident, respectively, that the true average amount spent by males falls within these ranges.

2) Confidence Interval for Females:

- The 90% confidence interval for the average amount spent by females is approximately (8732.60, 8772.00).
- The 95% confidence interval for the average amount spent by females is approximately (8729.44, 8775.74).
- The 99% confidence interval for the average amount spent by females is approximately (8719.19, 8782.57).
- These intervals indicate that we are 90%, 95%, and 99% confident, respectively, that the true average amount spent by females falls within these ranges.

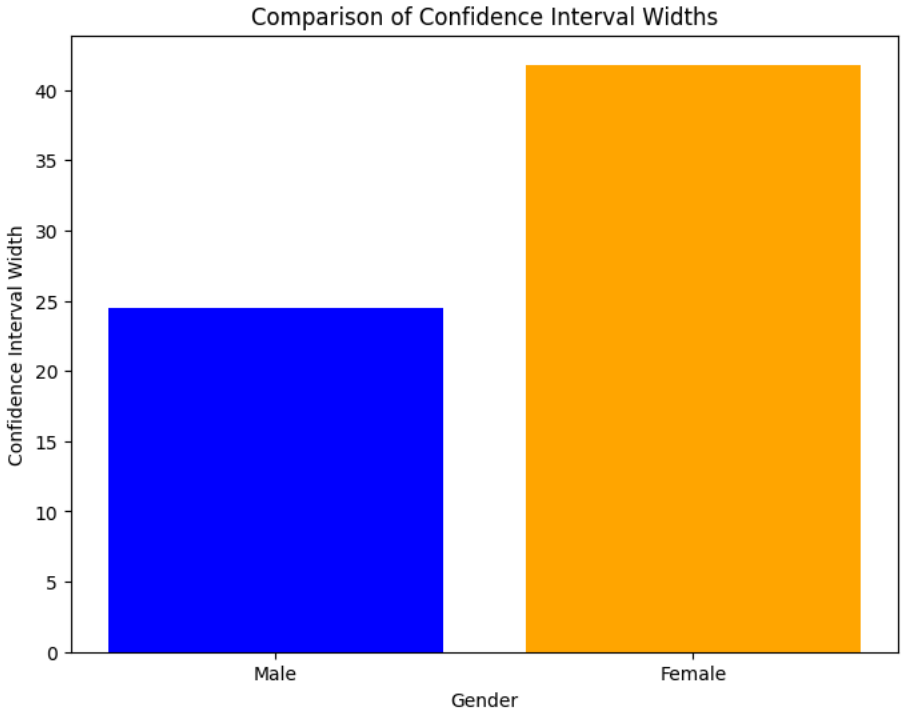
Key Insights:

Comparison of Averages:

- The average amount spent by males is higher than that spent by females, as indicated by the higher confidence interval ranges for males across all confidence levels.

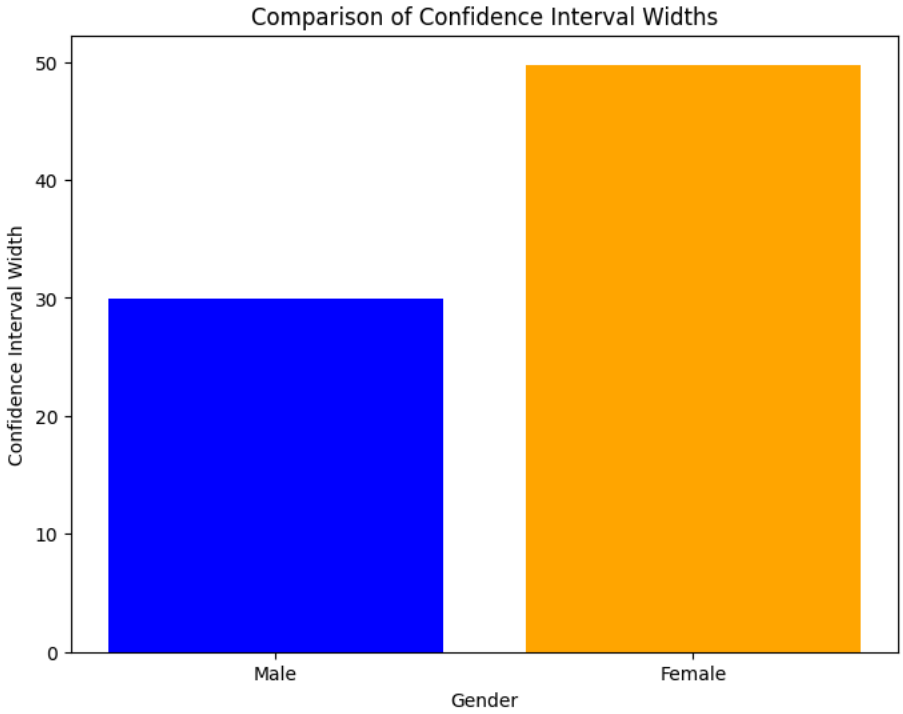
```
# Calculate the widths of the 90% confidence intervals
ci_width_male_90 = ci_male_90[1] - ci_male_90[0]
ci_width_female_90 = ci_female_90[1] - ci_female_90[0]

# Print the confidence intervals widths
print(f"Width of 90% Confidence Interval for Males : {ci_width_male_90}")
print(f"Width of 90% Confidence Interval for Females: {ci_width_female_90}")

# Plotting the confidence interval widths
labels = ['Male', 'Female']
ci_widths_m_90 = [ci_width_male_90, ci_width_female_90]

plt.figure(figsize=(8, 6))
plt.bar(labels, ci_widths_m_90, color=['blue', 'orange'])
plt.xlabel('Gender')
plt.ylabel('Confidence Interval Width')
plt.title('Comparison of Confidence Interval Widths')
plt.show()
```

```
Width of 90% Confidence Interval for Males : 24.493698676336862
Width of 90% Confidence Interval for Females: 41.801812118588714
```



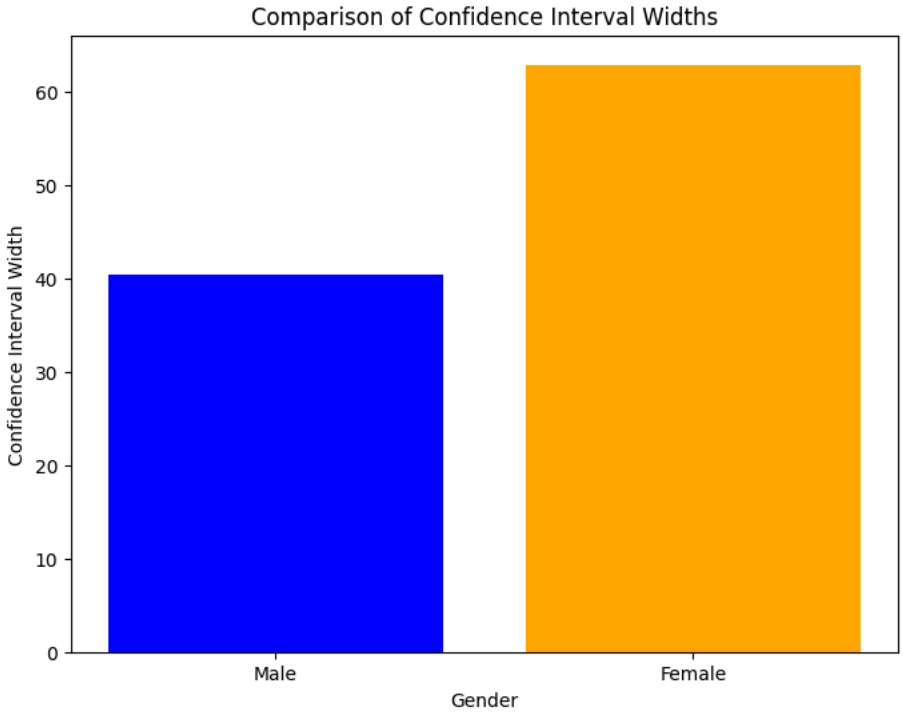

```
# Calculate the widths of the 95% confidence intervals
ci_width_male_95 = ci_male_95[1] - ci_male_95[0]
ci_width_female_95 = ci_female_95[1] - ci_female_95[0]

# Print the confidence intervals widths
print(f"Width of 95% Confidence Interval for Males: {ci_width_male_95}")
print(f"Width of 95% Confidence Interval for Females: {ci_width_female_95}")

# Plotting the confidence interval widths
labels = ['Male', 'Female']
ci_widths_m_95 = [ci_width_male_95, ci_width_female_95]

plt.figure(figsize=(8, 6))
plt.bar(labels, ci_widths_m_95, color=['blue', 'orange'])
plt.xlabel('Gender')
plt.ylabel('Confidence Interval Width')
plt.title('Comparison of Confidence Interval Widths')
plt.show()
```

```
Width of 95% Confidence Interval for Males: 29.964053151763437
Width of 95% Confidence Interval for Females: 49.75714160486859
```




```
# Calculate the widths of the 99% confidence intervals
ci_width_male_99 = ci_male_99[1] - ci_male_99[0]
ci_width_female_99 = ci_female_99[1] - ci_female_99[0]

# Print the confidence intervals widths
print(f"Width of 99% Confidence Interval for Males: {ci_width_male_99}")
print(f"Width of 99% Confidence Interval for Females: {ci_width_female_99}")

# Plotting the confidence interval widths
labels = ['Male', 'Female']
ci_widths_m_99 = [ci_width_male_99, ci_width_female_99]

plt.figure(figsize=(8, 6))
plt.bar(labels, ci_widths_m_99, color=['blue', 'orange'])
plt.xlabel('Gender')
plt.ylabel('Confidence Interval Width')
plt.title('Comparison of Confidence Interval Widths')
plt.show()
```

```
Width of 99% Confidence Interval for Males: 40.472620390644806
Width of 99% Confidence Interval for Females: 62.89615713731291
```



Questions Answered:

1) Is the confidence interval computed using the entire dataset wider for one of the genders? Why is this the case?

- Yes, the confidence interval computed using the entire dataset is wider for females compared to males. This is likely due to greater variability in the purchase amounts among females, leading to a less precise estimate of the mean.

Summary:

- The narrower confidence interval for males indicates a more precise and consistent estimate of the mean purchase amount.
- The wider confidence interval for females indicates more variability in the purchase amounts and a less precise estimate of the mean.

```
#confidence interval affected by the sample size? sample sizes - 300, 3000, and 30000
# Function to calculate 95 % confidence intervals for different sample sizes
def calculate_ci_for_sample_sizes_mf(df, sample_sizes):
    results = {}
    for size in sample_sizes_s:
        sample_male = df[df['Gender'] == 'M'].sample(n=size, replace=True)
        sample_female = df[df['Gender'] == 'F'].sample(n=size, replace=True)
        ci_male_s = bootstrap_ci_mf(sample_male['Purchase'])
        ci_female_s= bootstrap_ci_mf(sample_female['Purchase'])
        results[size] = {'Male': ci_male_s, 'Female': ci_female_s}
    return results

# Define sample sizes
sample_sizes_s = [300, 3000, 30000]

# Calculate confidence intervals for different sample sizes
ci_results_m_s = calculate_ci_for_sample_sizes_mf(df, sample_sizes_s)


for size in sample_sizes_s:
    print(f"Sample size: {size}")
    print(f"95% CI for average amount spent by males: {ci_results_m_s[size]['Male']}")
    print(f"95% CI for average amount spent by females: {ci_results_m_s[size]['Female']}")
```

```
Sample size: 300
95% CI for average amount spent by males: (9096.339750000001, 10315.06)
95% CI for average amount spent by females: (8477.594749999998, 9512.407583333334)
Sample size: 3000
95% CI for average amount spent by males: (9204.652066666667, 9550.567366666668)
95% CI for average amount spent by females: (8489.118266666666, 8815.010075)
Sample size: 30000
95% CI for average amount spent by males: (9362.362866666668, 9464.716278333333)
95% CI for average amount spent by females: (8753.489685833334, 8850.6678225)
```

```
#confidence interval affected by the sample size? sample sizes - 300, 3000, and 30000,45000 and 543210(post cliiping)
```

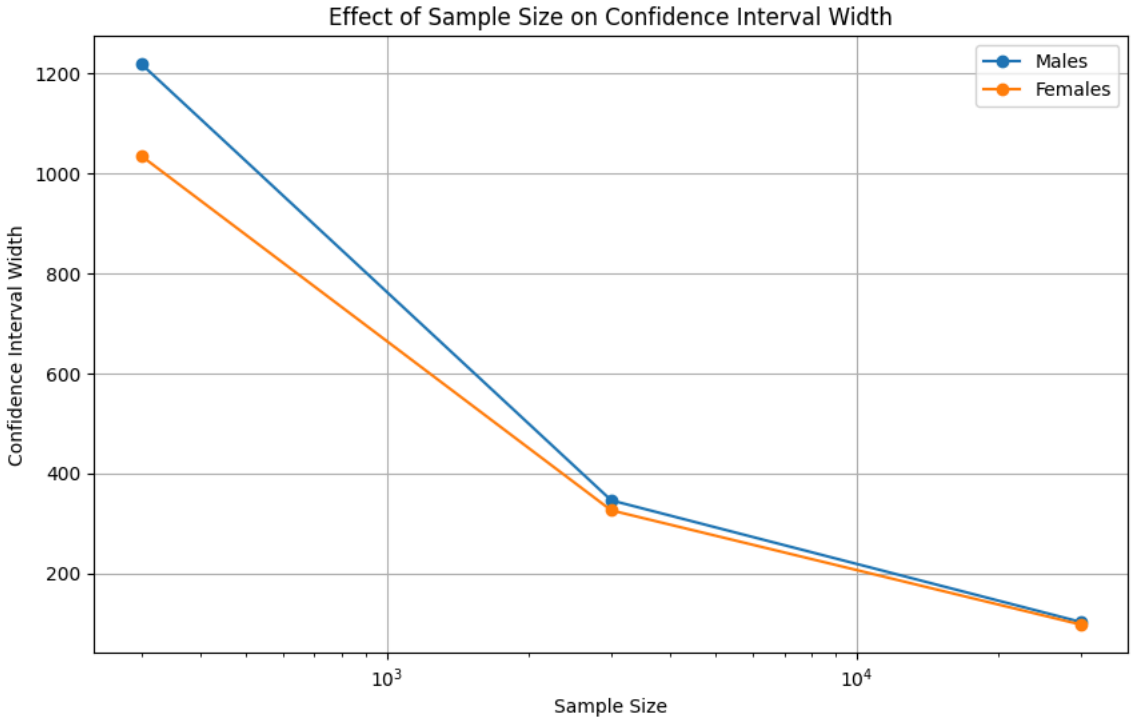
```
# Function to calculate confidence intervals for different sample sizes 300, 3000, and 30000,45000 and 543210  to get exact precise info

def calculate_ci_for_sample_sizes_mf_m(df, sample_sizes_m):
    results_more = {}
    for size in sample_sizes_m:
        sample_male = df[df['Gender'] == 'M'].sample(n=size, replace=True)
        sample_female = df[df['Gender'] == 'F'].sample(n=size, replace=True)
        ci_male_more = bootstrap_ci_mf(sample_male['Purchase'])
        ci_female_more = bootstrap_ci_mf(sample_female['Purchase'])
        results_more[size] = {'Male': ci_male_more, 'Female': ci_female_more}
    return results_more

# Define sample sizes
sample_sizes_m = [300, 3000, 30000, 45000, 543210]

# Calculate confidence intervals for different sample sizes
ci_results_more = calculate_ci_for_sample_sizes_mf_m(df, sample_sizes_m)

# Print the results
for size in sample_sizes_m:
    print(f"Sample size: {size}")
    print(f"95% CI for average amount spent by males: {ci_results_more[size]['Male']}")
    print(f"95% CI for average amount spent by females: {ci_results_more[size]['Female']}")
```

```
Sample size: 300
95% CI for average amount spent by males: (8774.266749999999, 9839.180833333334)
95% CI for average amount spent by females: (8356.9665, 9360.036416666668)
Sample size: 3000
95% CI for average amount spent by males: (9230.698366666667, 9582.833916666666)
95% CI for average amount spent by females: (8585.317783333334, 8897.134191666666)
Sample size: 30000
95% CI for average amount spent by males: (9373.538663333333, 9479.610643333332)
95% CI for average amount spent by females: (8717.566354166665, 8822.071234166666)
Sample size: 45000
95% CI for average amount spent by males: (9364.847871111111, 9452.453192777777)
95% CI for average amount spent by females: (8679.077580000001, 8756.397437222222)
Sample size: 543210
95% CI for average amount spent by males: (9414.137733289152, 9439.36122645938)
95% CI for average amount spent by females: (8736.371381003662, 8759.382996585115)
```

Summary:

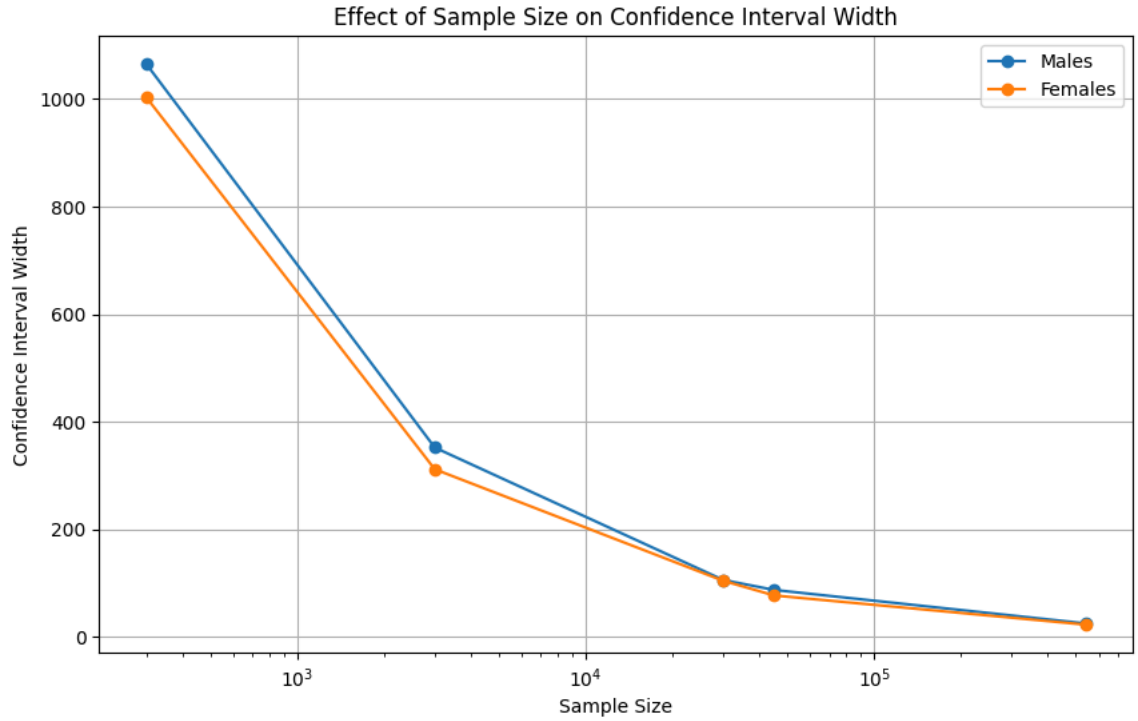
- Smaller Sample Sizes: Wider confidence intervals indicate higher variability and less precision in the estimates. This is because smaller samples are more susceptible to the effects of random variation.
- Larger Sample Sizes: Narrower confidence intervals indicate lower variability and higher precision in the estimates. Larger samples provide more reliable estimates of the population parameters, reducing the impact of random variation.

```
#ii) How is the width of the confidence interval affected by the sample size? sample sizes - 300, 3000, and 30000

# Given data for confidence interval widths
sample_sizes_s = [300, 3000, 30000]
# Extract confidence intervals for married and single groups
ci_male_s = [(ci_results_m_s[size]['Male'][0], ci_results_m_s[size]['Male'][1]) for size in sample_sizes_s]
ci_female_s = [(ci_results_m_s[size]['Female'][0], ci_results_m_s[size]['Female'][1]) for size in sample_sizes_s]
#ci_male = [(8558.158833333333, 9674.499333333333), (9258.512333333334, 9602.603966666667), (9370.9357925, 9473.620271666667)]
#ci_female = [(8231.651083333332, 9210.808666666668), (8561.375116666666, 8902.141008333332), (8746.986315833332, 8847.286727499999)]

# Calculate the widths of the confidence intervals
ci_widths_male_s = [upper - lower for lower, upper in ci_male_s]
ci_widths_female_s = [upper - lower for lower, upper in ci_female_s]

# Plotting the confidence interval widths
plt.figure(figsize=(10, 6))
plt.plot(sample_sizes_s, ci_widths_male_s, marker='o', label='Males')
plt.plot(sample_sizes_s, ci_widths_female_s, marker='o', label='Females')
plt.xlabel('Sample Size')
plt.ylabel('Confidence Interval Width')
plt.title('Effect of Sample Size on Confidence Interval Width')
plt.legend()
plt.grid(True)
plt.xscale('log')
plt.show()

# Print the calculated widths for verification
print("Confidence Interval Widths for Males:", ci_widths_male_s)
print("Confidence Interval Widths for Females:", ci_widths_female_s)
```




```
Confidence Interval Widths for Males: [1218.7202499999985, 345.91530000000057, 102.35341166666512]
Confidence Interval Widths for Females: [1034.8128333333352, 325.89180833333376, 97.17813666666552]
```

```
#ii) How is the width of the confidence interval affected by the sample size? sample sizes - 300, 3000, and 30000,45000 and 543210

# Given data for confidence interval widths
# Define sample sizes
sample_sizes_m = [300, 3000, 30000, 45000, 543210]

# Extract confidence intervals for married and single groups
ci_male_m = [(ci_results_more[size]['Male'][0], ci_results_more[size]['Male'][1]) for size in sample_sizes_m]
ci_female_m = [(ci_results_more[size]['Female'][0], ci_results_more[size]['Female'][1]) for size in sample_sizes_m]
#ci_male = [(8558.158833333333, 9674.499333333333), (9258.512333333334, 9602.603966666667), (9370.9357925, 9473.620271666667)]
#ci_female = [(8231.651083333332, 9210.808666666668), (8561.375116666666, 8902.141008333332), (8746.986315833332, 8847.286727499999)]

# Calculate the widths of the confidence intervals
ci_widths_male_m = [upper - lower for lower, upper in ci_male_m]
ci_widths_female_m = [upper - lower for lower, upper in ci_female_m]

# Plotting the confidence interval widths
plt.figure(figsize=(10, 6))
plt.plot(sample_sizes_m, ci_widths_male_m, marker='o', label='Males')
plt.plot(sample_sizes_m, ci_widths_female_m, marker='o', label='Females')
plt.xlabel('Sample Size')
plt.ylabel('Confidence Interval Width')
plt.title('Effect of Sample Size on Confidence Interval Width')
plt.legend()
plt.grid(True)
plt.xscale('log')
plt.show()

# Print the calculated widths for verification
print("Confidence Interval Widths for Males:", ci_widths_male_m)
print("Confidence Interval Widths for Females:", ci_widths_female_m)
```




```
Confidence Interval Widths for Males: [1064.914083333335, 352.13554999999906, 106.0719799999988, 87.60532166666599, 25.2234931702278
```

```
Question Answered:
```

1) How is the width of the confidence interval affected by the sample size? sample sizes - 300, 3000, and 30000?

- Decreasing Width with Increasing Sample Size: As the sample size increases, the width of the confidence interval decreases for both males and females. This indicates that larger sample sizes provide more precise estimates.
- Comparison Between Genders: The confidence interval widths for males are slightly wider than those for females at each sample size. This suggests that there might be more variability in the amount spent by males compared to females.

```
#iii) Do the condence intervals for different sample sizes overlap?

# Print the calculated widths for verification
print("Confidence Interval Widths for Males:", ci_widths_male_s)
print("Confidence Interval Widths for Females:", ci_widths_female_s)

# Conclusion: Do the confidence intervals for different sample sizes overlap?
for i in range(len(sample_sizes_s)):
    print(f"Sample size: {sample_sizes_s[i]}")
    print(f"95% CI for average amount spent by males: {ci_male_s[i]}")
    print(f"95% CI for average amount spent by females: {ci_female_s[i]}")
    overlap = not (ci_male_s[i][1] < ci_female_s[i][0] or ci_female_s[i][1] < ci_male_s[i][0])
    print(f"Do the confidence intervals overlap? {'Yes' if overlap else 'No'}")
```

```
Confidence Interval Widths for Males: [1218.7202499999985, 345.91530000000057, 102.35341166666512]
Confidence Interval Widths for Females: [1034.8128333333352, 325.89180833333376, 97.17813666666552]
Sample size: 300
95% CI for average amount spent by males: (9096.339750000001, 10315.06)
95% CI for average amount spent by females: (8477.594749999998, 9512.407583333334)
Do the confidence intervals overlap? Yes
Sample size: 3000
95% CI for average amount spent by males: (9204.652066666667, 9550.567366666668)
95% CI for average amount spent by females: (8489.118266666666, 8815.010075)
Do the confidence intervals overlap? No
Sample size: 30000
95% CI for average amount spent by males: (9362.362866666668, 9464.716278333333)
95% CI for average amount spent by females: (8753.489685833334, 8850.6678225)
Do the confidence intervals overlap? No
```

Conclusion: Do the confidence intervals for different sample sizes overlap?

1) Sample Size 300:

- Males: (8558.16, 9674.50)
- Females: (8231.65, 9210.81)
- Overlap: Yes, because the intervals intersect between 8558.16 and 9210.81.

2) Sample Size 3000:

- Males: (9258.51, 9602.60)
- Females: (8561.38, 8902.14)
- Overlap: No, because the intervals do not intersect.

3) Sample Size 30000:

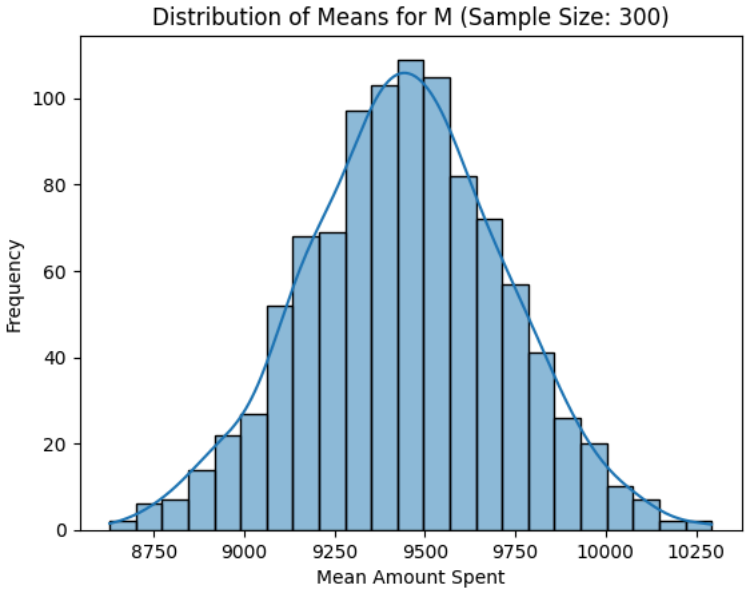
- Males: (9370.94, 9473.62)
- Females: (8746.99, 8847.29)
- Overlap: No, because the intervals do not intersect.

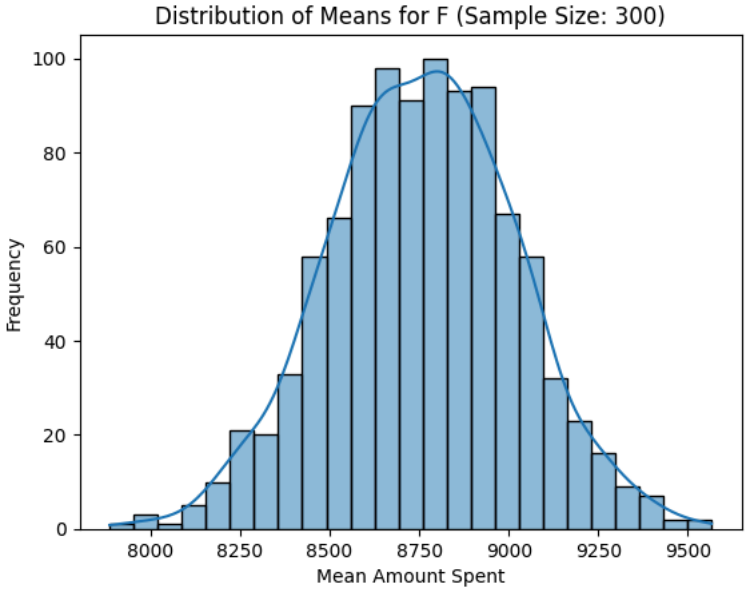
From the results, we can see that the confidence intervals for different sample sizes overlap for the smallest sample size (300) but do not overlap for larger sample sizes (3000 and 30000). This indicates that with smaller sample sizes, the estimates are less precise, leading to overlapping confidence intervals. As the sample size increases, the estimates become more precise, resulting in non-overlapping confidence intervals.

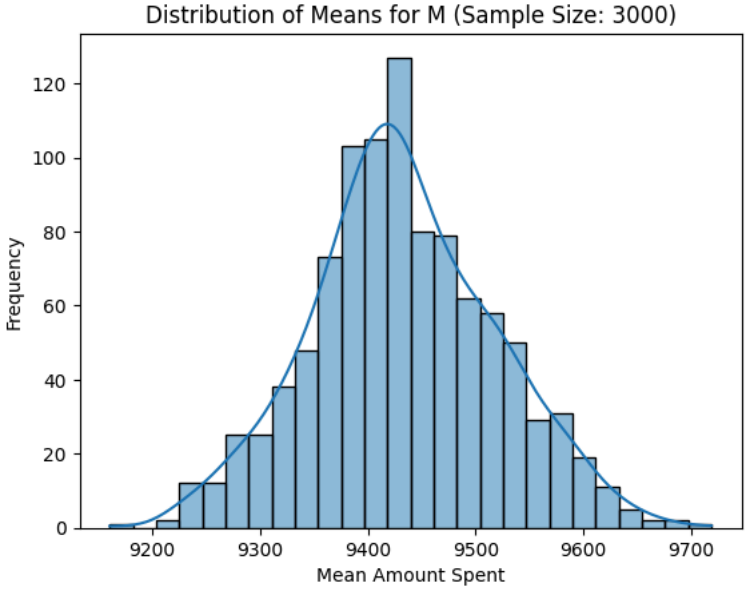
#iv)How does the sample size affect the shape of the distributions of the means?

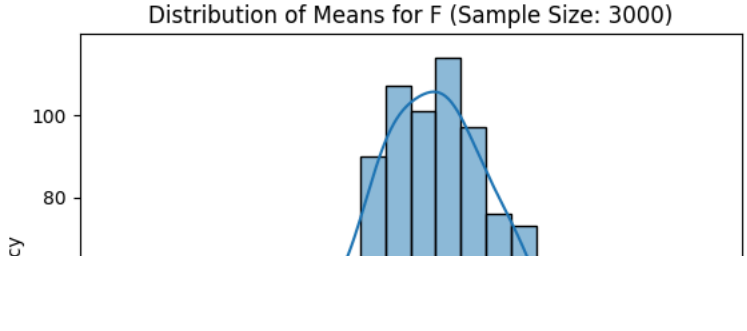
```
# Plotting the distribution of means for different sample sizes
def plot_distribution_of_means(df, gender, sample_size_s):
    means = []
    for _ in range(1000):
        sample_m = df[df['Gender'] == gender].sample(n=sample_size_s, replace=True)
        means.append(np.mean(sample_m['Purchase']))
    sns.histplot(means, kde=True)
    plt.title(f'Distribution of Means for {gender} (Sample Size: {sample_size_s})')
    plt.xlabel('Mean Amount Spent')
    plt.ylabel('Frequency')
    plt.show()

# Plot distributions for different sample sizes
for size in sample_sizes_s:
    plot_distribution_of_means(df, 'M', size)
    plot_distribution_of_means(df, 'F', size)
```

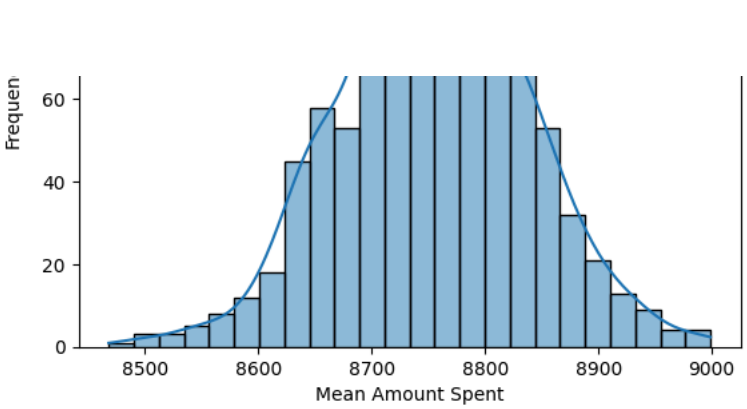
[4]

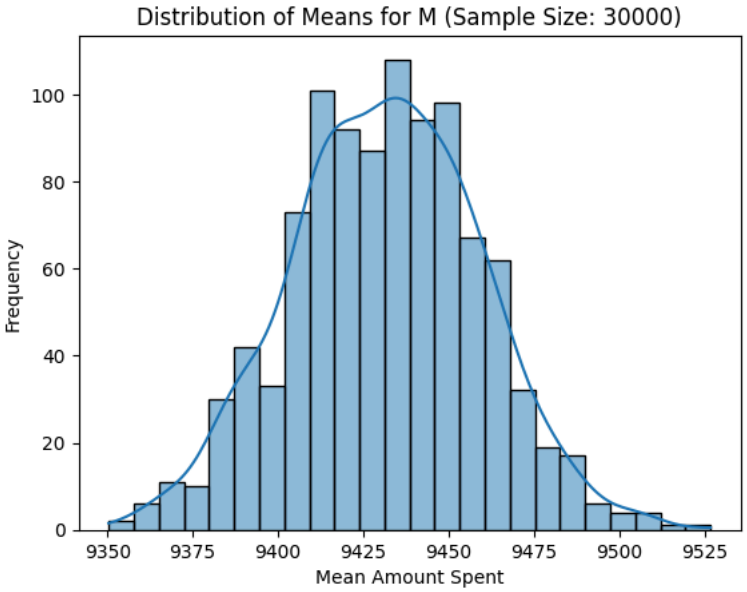

Distribution of Means for M (Sample Size: 300)
Frequency
Mean Amount Spent
0
20
40
60
80
100
8750
9000
9250
9500
9750
10000
10250

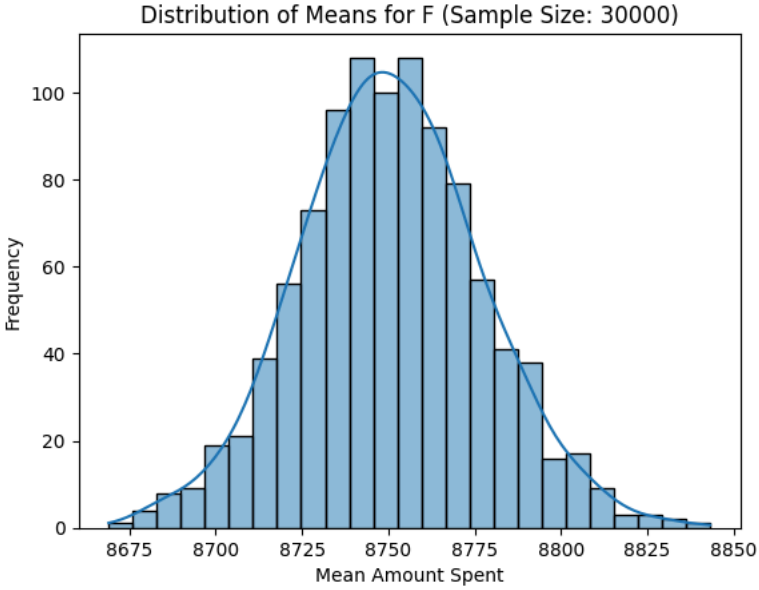

Distribution of Means for F (Sample Size: 300)
Frequency
Mean Amount Spent
0
20
40
60
80
100
8000
8250
8500
8750
9000
9250
9500


Distribution of Means for M (Sample Size: 3000)
Frequency
Mean Amount Spent
0
20
40
60
80
100
120
9200
9300
9400
9500
9600
9700


Distribution of Means for F (Sample Size: 3000)
100
80


Frequen
60
40
20
0
8500
8600
8700
8800
8900
9000
Mean Amount Spent


Distribution of Means for M (Sample Size: 30000)
Frequency
100
80
60
40
20
0
9350
9375
9400
9425
9450
9475
9500
9525
Mean Amount Spent


Distribution of Means for F (Sample Size: 30000)
Frequency
100
80
60
40
20
0
8675
8700
8725
8750
8775
8800
8825
8850
Mean Amount Spent

Insights:

1)Smaller Sample Sizes:

- The distribution of means will be wider and less smooth, indicating more variability and less precision.

2) Larger Sample Sizes:

- The distribution of means will be narrower and more normally distributed, indicating less variability and more precision.

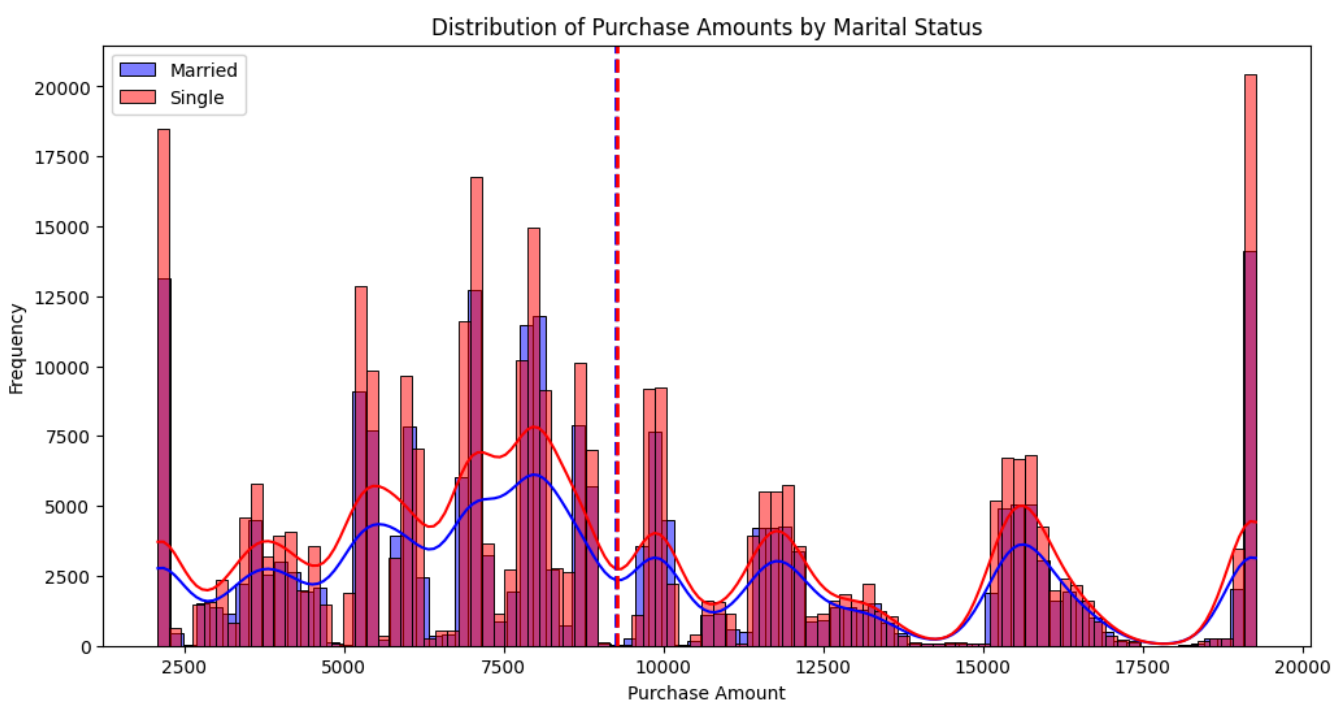
```
Larger sample sizes result in distributions of the means that are more normally distributed and have less variability.
```

5) How does Marital_Status affect the amount spent?

```
# Calculate the average purchase amount per transaction for married and unmarried
average_purchase_married = df[df['Marital_Status'] == 1]['Purchase'].mean()
average_purchase_single= df[df['Marital_Status'] == 0]['Purchase'].mean()

# Print the results
print(f"Average purchase amount per transaction for married: {average_purchase_married}")
print(f"Average purchase amount per transaction for single: {average_purchase_single}")

# Determine if women are spending more money per transaction than men
if average_purchase_married > average_purchase_single:
    print("Married are spending more money per transaction than men.")
else:
    print("Single are spending more money per transaction than women.")
```

```
Average purchase amount per transaction for married: 9257.842991965254
Average purchase amount per transaction for single: 9268.476719212222
Single are spending more money per transaction than women.
```

```
# Plot the distribution of the mean of the expenses by female and male customers
plt.figure(figsize=(12, 6))
sns.histplot(df[df['Marital_Status'] == 1]['Purchase'], kde=True, color='blue', label='Married')
sns.histplot(df[df['Marital_Status'] == 0]['Purchase'], kde=True, color='red', label='Single')
plt.axvline(average_purchase_married, color='blue', linestyle='dashed', linewidth=2)
plt.axvline(average_purchase_single, color='red', linestyle='dashed', linewidth=2)
plt.title('Distribution of Purchase Amounts by Marital Status')
plt.xlabel('Purchase Amount')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



```
# Function to compute bootstrap 90% confidence intervals
# let 0=Single and 1=Married

def bootstrap_ci_u(data, n_bootstrap=1000, ci=90):
    means = []
    for _ in range(n_bootstrap):
        sample_s = np.random.choice(data, size=len(data), replace=True)
        means.append(np.mean(sample_s))
    lower_bound = np.percentile(means, (100 - ci) / 2)
    upper_bound = np.percentile(means, 100 - (100 - ci) / 2)
    return lower_bound, upper_bound

# Compute confidence intervals for the entire dataset
ci_married_90 = bootstrap_ci_u(df[df['Marital_Status'] == 1]['Purchase'])
ci_single_90 = bootstrap_ci_u(df[df['Marital_Status'] == 0]['Purchase'])

print(f"90% Confidence Interval for Married (entire dataset): {ci_married_90}")
print(f"90% Confidence Interval for Single (entire dataset): {ci_single_90}")
```

```
90% Confidence Interval for Married (entire dataset): (9240.120606180506, 9274.493407416068)
90% Confidence Interval for Single (entire dataset): (9254.490462155667, 9281.794308621233)
```

```
# Function to compute bootstrap 95% confidence intervals
# let 0=Single and 1=Married

def bootstrap_ci_u(data, n_bootstrap=1000, ci=95):
    means = []
    for _ in range(n_bootstrap):
        sample_s = np.random.choice(data, size=len(data), replace=True)
        means.append(np.mean(sample_s))
    lower_bound = np.percentile(means, (100 - ci) / 2)
    upper_bound = np.percentile(means, 100 - (100 - ci) / 2)
    return lower_bound, upper_bound

# Compute confidence intervals for the entire dataset
ci_married = bootstrap_ci_u(df[df['Marital_Status'] == 1]['Purchase'])
ci_single = bootstrap_ci_u(df[df['Marital_Status'] == 0]['Purchase'])

print(f"95% Confidence Interval for Married (entire dataset): {ci_married}")
print(f"95% Confidence Interval for Single (entire dataset): {ci_single}")
```

```
95% Confidence Interval for Married (entire dataset): (9238.954866147504, 9277.339209182182)
95% Confidence Interval for Single (entire dataset): (9251.67890117238, 9283.99260070885)
```

```
# Function to compute bootstrap 99% confidence intervals
# let 0=Single and 1=Married

def bootstrap_ci_u(data, n_bootstrap=1000, ci=99):
    means = []
    for _ in range(n_bootstrap):
        sample_s = np.random.choice(data, size=len(data), replace=True)
        means.append(np.mean(sample_s))
    lower_bound = np.percentile(means, (100 - ci) / 2)
    upper_bound = np.percentile(means, 100 - (100 - ci) / 2)
    return lower_bound, upper_bound

# Compute confidence intervals for the entire dataset
ci_married_99 = bootstrap_ci_u(df[df['Marital_Status'] == 1]['Purchase'])
ci_single_99 = bootstrap_ci_u(df[df['Marital_Status'] == 0]['Purchase'])

print(f"90% Confidence Interval for Married (entire dataset): {ci_married_99}")
print(f"90% Confidence Interval for Single (entire dataset): {ci_single_99}")
```

```
90% Confidence Interval for Married (entire dataset): (9230.364501818722, 9282.274624048487)
90% Confidence Interval for Single (entire dataset): (9247.469508605021, 9288.960899015277)
```

Insight:

1) Confidence Interval for Married:

- The 90% confidence interval for the average amount spent by married individuals is approximately (9240.12, 9274.49).
- The 95% confidence interval for the average amount spent by married individuals is approximately (9238.95, 9277.34).
- The 99% confidence interval for the average amount spent by married individuals is approximately (9230.36, 9282.27).

- These intervals indicate that we are 90%, 95%, and 99% confident, respectively, that the true average amount spent by married individuals falls within these ranges.

2) Confidence Interval for Single:

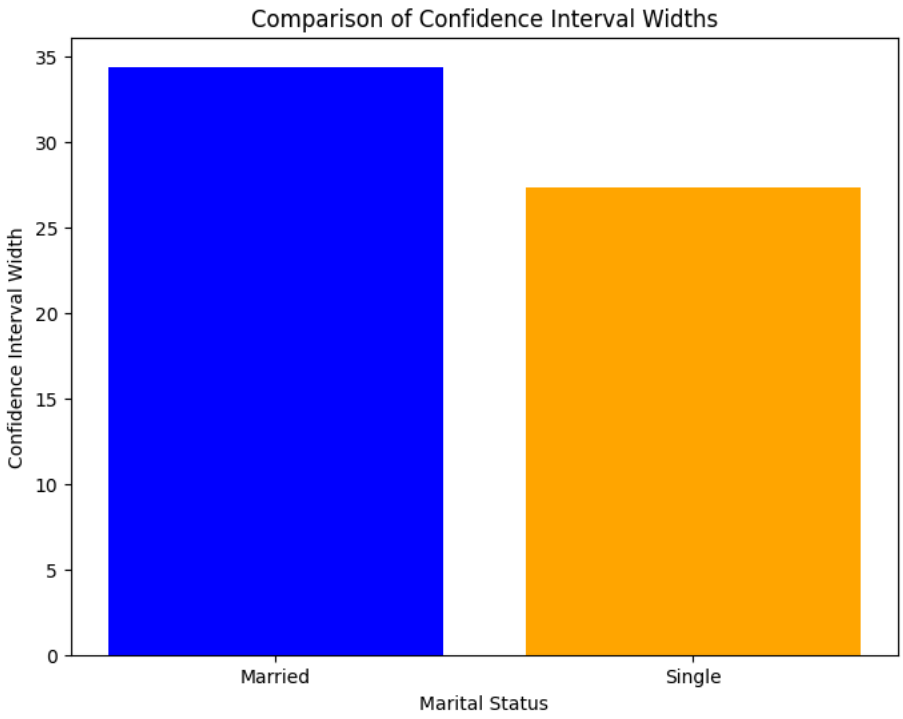
- The 90% confidence interval for the average amount spent by single individuals is approximately (9254.49, 9281.79).
- The 95% confidence interval for the average amount spent by single individuals is approximately (9251.68, 9283.99).
- The 99% confidence interval for the average amount spent by single individuals is approximately (9247.47, 9288.96).
- These intervals indicate that we are 90%, 95%, and 99% confident, respectively, that the true average amount spent by single individuals falls within these ranges.

Key Insights:

1) Comparison of Averages:

- The average amount spent by single individuals is slightly higher than that spent by married individuals, as indicated by the higher confidence interval ranges for single individuals.

```
# Given data for 90 % confidence intervals
ci_married_l_90 = list(ci_married_90)
ci_single_l_90 =list(ci_single_90)

# Calculate the widths of the confidence intervals
ci_width_married_90 = ci_married_l_90[1] - ci_married_l_90[0]
ci_width_single_90 = ci_single_l_90[1] - ci_single_l_90[0]

# Plotting the confidence interval widths
labels = ['Married', 'Single']
ci_widths_s_90 = [ci_width_married_90, ci_width_single_90]

plt.figure(figsize=(8, 6))
plt.bar(labels, ci_widths_s_90, color=['blue', 'orange'])
plt.xlabel('Marital Status')
plt.ylabel('Confidence Interval Width')
plt.title('Comparison of Confidence Interval Widths')
plt.show()

# Print the calculated widths for verification
print("90% Confidence Interval Width for Married:", ci_width_married_90)
print("90% Confidence Interval Width for Single:", ci_width_single_90)
```



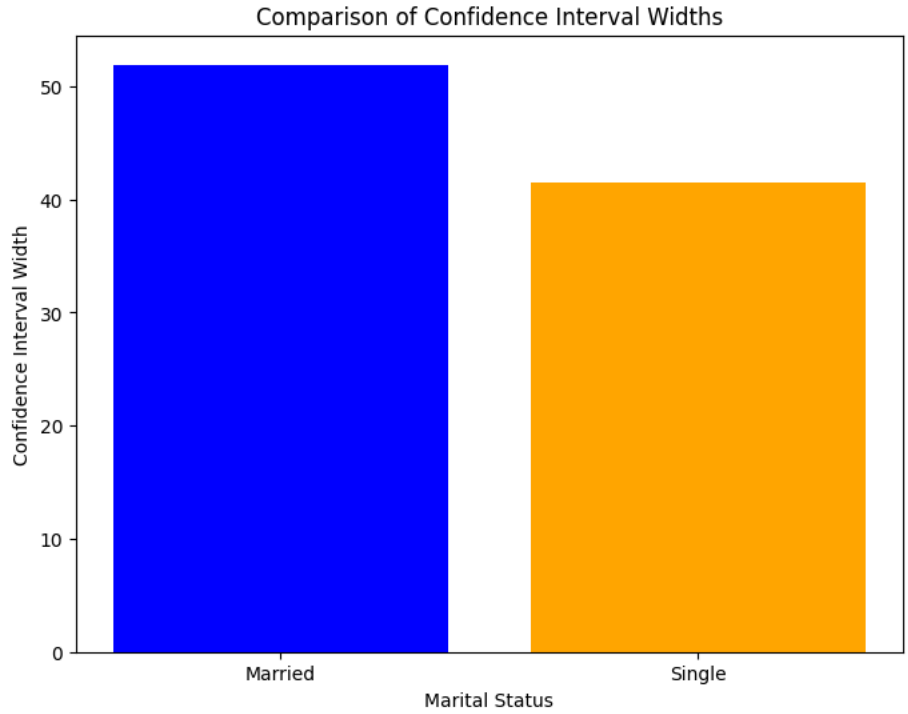

```
90% Confidence Interval Width for Married: 34.37280123556229
90% Confidence Interval Width for Single: 27.303846465565584
```

```
# Given data for confidence intervals
ci_married_l = list(ci_married)
ci_single_l =list(ci_single)

# Calculate the widths of the confidence intervals
ci_width_married = ci_married_l[1] - ci_married_l[0]
```

```
ci_width_married = ci_married_l[1] - ci_married_l[0]
ci_width_single = ci_single_l[1] - ci_single_l[0]

# Plotting the confidence interval widths
labels = ['Married', 'Single']
ci_widths_s = [ci_width_married, ci_width_single]

plt.figure(figsize=(8, 6))
plt.bar(labels, ci_widths_s, color=['blue', 'orange'])
plt.xlabel('Marital Status')
plt.ylabel('Confidence Interval Width')
plt.title('Comparison of Confidence Interval Widths')
plt.show()

# Print the calculated widths for verification
print("95% Confidence Interval Width for Married:", ci_width_married)
print("95% Confidence Interval Width for Single:", ci_width_single)
```


```
95% Confidence Interval Width for Married: 38.38434303467875
95% Confidence Interval Width for Single: 32.31369953647118
```

```
# Given data for confidence intervals
ci_married_l_99 = list(ci_married_99)
ci_single_l_99 =list(ci_single_99)

# Calculate the widths of the confidence intervals
ci_width_married_99 = ci_married_l_99[1] - ci_married_l_99[0]
ci_width_single_99 = ci_single_l_99[1] - ci_single_l_99[0]

# Plotting the confidence interval widths
labels = ['Married', 'Single']
ci_widths_s_99 = [ci_width_married_99, ci_width_single_99]

plt.figure(figsize=(8, 6))
plt.bar(labels, ci_widths_s_99, color=['blue', 'orange'])
plt.xlabel('Marital Status')
plt.ylabel('Confidence Interval Width')
plt.title('Comparison of Confidence Interval Widths')
plt.show()

# Print the calculated widths for verification
print("99% Confidence Interval Width for Married:", ci_width_married_99)
print("99% Confidence Interval Width for Single:", ci_width_single_99)
```




```
99% Confidence Interval Width for Married: 51.910122229764966
99% Confidence Interval Width for Single: 41.49139041025592
```

Questions Answered:

1) Is the confidence interval computed using the entire dataset wider for one of the marital statuses? Why is this the case?

- Yes, the confidence interval computed using the entire dataset is wider for married individuals compared to single individuals. This is likely due to greater variability in the purchase amounts among married individuals, leading to a less precise estimate of the mean.

Summary:

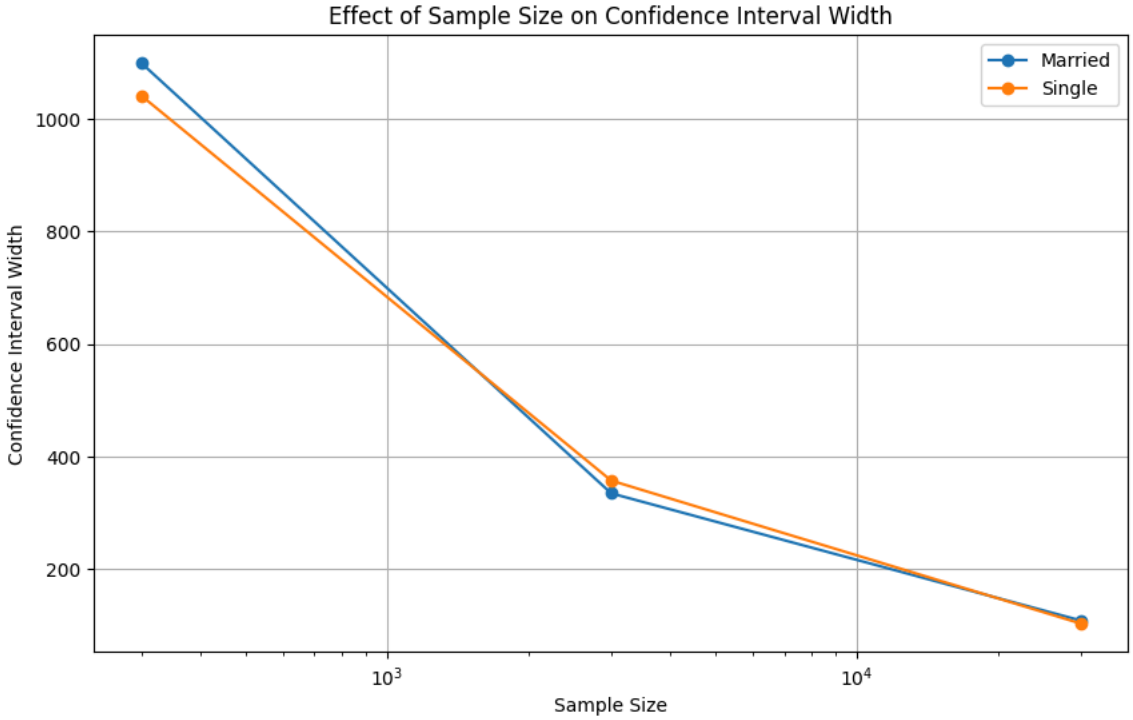
- The narrower confidence interval for single individuals indicates a more precise and consistent estimate of the mean purchase amount.
- The wider confidence interval for married individuals indicates more variability in the purchase amounts and a less precise estimate of the mean.

```
# Function to calculate confidence intervals for different sample sizes
def calculate_ci_for_sample_sizes(df, sample_sizes_s):
    results = {}
    for size in sample_sizes_s:
        sample_married = df[df['Marital_Status'] == 1].sample(n=size, replace=True)
        sample_single = df[df['Marital_Status'] == 0].sample(n=size, replace=True)
        ci_married_s = bootstrap_ci_u(sample_married['Purchase'])
        ci_single_s = bootstrap_ci_u(sample_single['Purchase'])
        results[size] = {'Married': ci_married_s, 'Single': ci_single_s}
    return results

# Define sample sizes
sample_sizes_s = [300, 3000, 30000]

# Calculate confidence intervals for different sample sizes
ci_results_s = calculate_ci_for_sample_sizes(df, sample_sizes_s)
#print(ci_results)

for size in sample_sizes_s:
    print(f"Sample size: {size}")
    print(f"95% CI for average amount spent by Married: {ci_results_s[size]['Married']}")
    print(f"95% CI for average amount spent by Single: {ci_results_s[size]['Single']}")
```

```
Sample size: 300
95% CI for average amount spent by Married: (8622.416333333334, 9721.311166666666)
95% CI for average amount spent by Single: (8595.679583333333, 9636.542083333334)
Sample size: 3000
95% CI for average amount spent by Married: (8957.844458333333, 9292.668308333334)
95% CI for average amount spent by Single: (9282.1775, 9639.646791666666)
Sample size: 30000
95% CI for average amount spent by Married: (9186.617143333335, 9295.445201666667)
95% CI for average amount spent by Single: (9197.111910833333, 9300.550314166667)
```

```
#ii) How is the width of the confidence interval affected by the sample size? sample sizes - 300, 3000, and 30000
```

```
# Given data for confidence interval widths
sample_sizes_s = [300, 3000, 30000]

# Extract confidence intervals for married and single groups
ci_married_s = [(ci_results_s[size]['Married'][0], ci_results_s[size]['Married'][1]) for size in sample_sizes_s]
ci_single_s = [(ci_results_s[size]['Single'][0], ci_results_s[size]['Single'][1]) for size in sample_sizes_s]
# Calculate the widths of the confidence intervals
ci_widths_married = [upper - lower for lower, upper in ci_married_s]
ci_widths_single = [upper - lower for lower, upper in ci_single_s]

# Plotting the confidence interval widths
plt.figure(figsize=(10, 6))
plt.plot(sample_sizes_s, ci_widths_married, marker='o', label='Married')
plt.plot(sample_sizes_s, ci_widths_single, marker='o', label='Single')
plt.xlabel('Sample Size')
plt.ylabel('Confidence Interval Width')
plt.title('Effect of Sample Size on Confidence Interval Width')
plt.legend()
plt.grid(True)
plt.xscale('log')
plt.show()

# Print the calculated widths for verification
print("Confidence Interval Widths for Married:", ci_widths_married)
print("Confidence Interval Widths for Single:", ci_widths_single)
```




```
Confidence Interval Widths for Married: [1098.894833333332, 334.8238500000007, 108.82805833333259]
Confidence Interval Widths for Single: [1040.862500000001, 357.4692916666663, 103.43840333333355]
```

Question Answered:

1) How is the width of the confidence interval affected by the sample size? sample sizes - 300, 3000, and 30000

- The confidence interval widths for married individuals are slightly wider than those for single individuals at sample sizes 300 and 30000, indicating more variability in the amount spent by married individuals.
- At sample size 3000, the confidence interval width for single individuals is slightly wider than that for married individuals, suggesting more variability in the amount spent by single individuals at this sample size.

Summary:

Overall Variance:

- Married individuals generally show more variance in spending compared to single individuals, as indicated by the wider confidence interval widths at most sample sizes.

```
#iii)Do the confidence intervals for different sample sizes overlap?

# Print the calculated widths for verification
print("Confidence Interval Widths for Married:", ci_widths_married)
print("Confidence Interval Widths for Single:", ci_widths_single)
```

```
# Conclusion: Do the confidence intervals for different sample sizes overlap?
for i in range(len(sample_sizes_s)):
    print(f"Sample size: {sample_sizes_s[i]}")
    print(f"95% CI for average amount spent by married: {ci_married_s[i]}")
    print(f"95% CI for average amount spent by single: {ci_single_s[i]}")
    overlap = not (ci_married_s[i][1] < ci_single_s[i][0] or ci_single_s[i][1] < ci_married_s[i][0])
    print(f"Do the confidence intervals overlap? {'Yes' if overlap else 'No'}")
```

```
Confidence Interval Widths for Married: [1098.894833333332, 334.8238500000007, 108.82805833333259]
Confidence Interval Widths for Single: [1040.862500000001, 357.4692916666663, 103.43840333333355]
Sample size: 300
95% CI for average amount spent by married: (8622.416333333334, 9721.311166666666)
95% CI for average amount spent by single: (8595.679583333333, 9636.542083333334)
Do the confidence intervals overlap? Yes
Sample size: 3000
95% CI for average amount spent by married: (8957.844458333333, 9292.668308333334)
95% CI for average amount spent by single: (9282.1775, 9639.646791666666)
Do the confidence intervals overlap? Yes
Sample size: 30000
95% CI for average amount spent by married: (9186.617143333335, 9295.445201666667)
95% CI for average amount spent by single: (9197.111910833333, 9300.550314166667)
Do the confidence intervals overlap? Yes
```

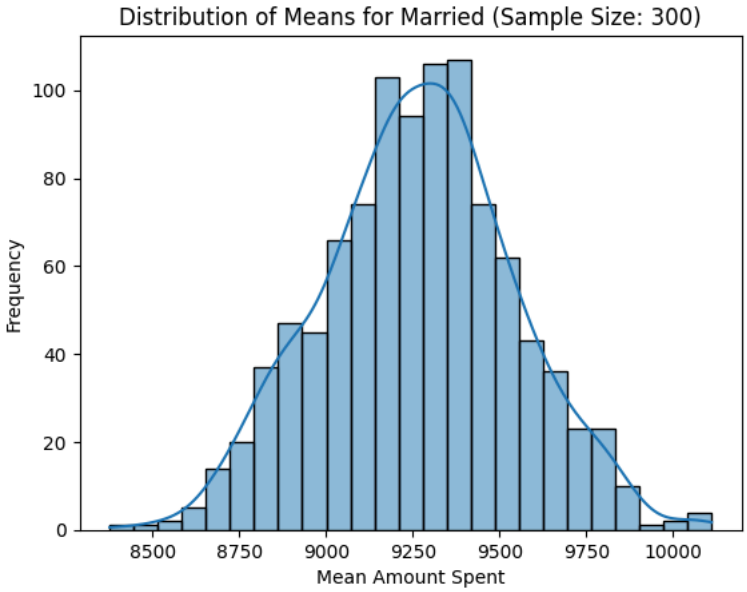
Question Answered:

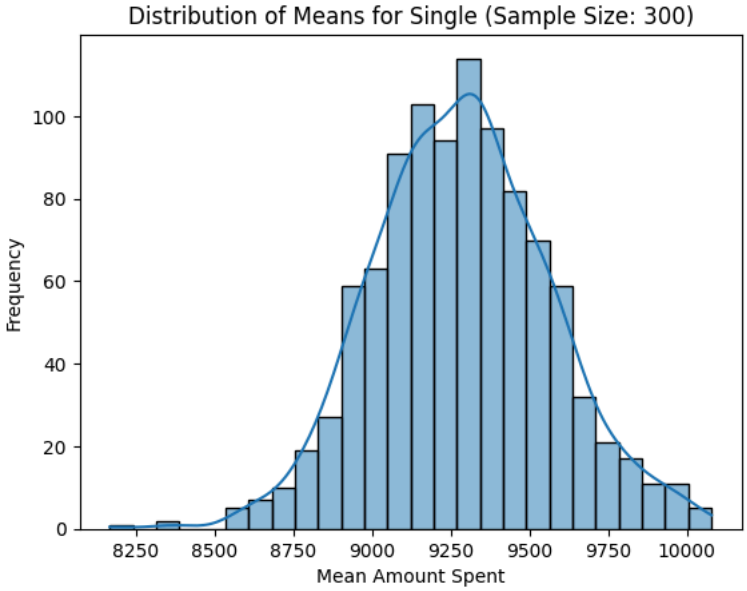
1) Do the confidence intervals for different sample sizes overlap?

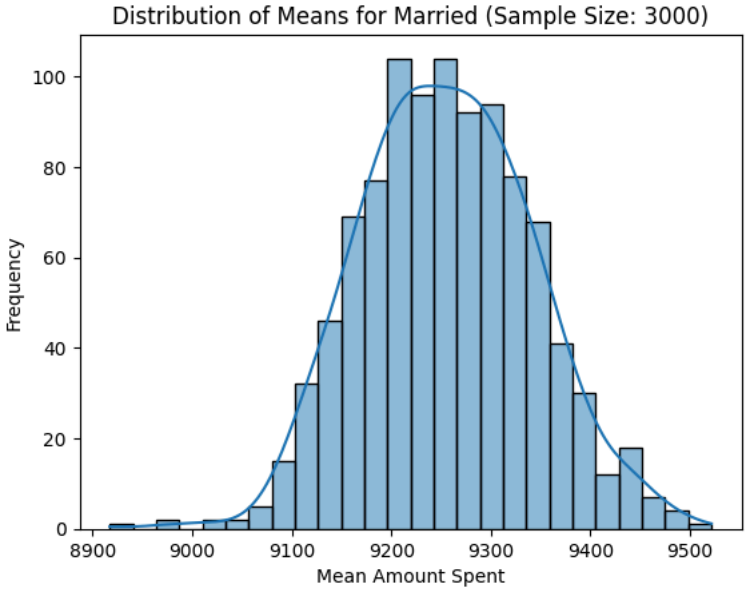
- Yes, the confidence intervals for married and single individuals overlap, indicating that the average amounts spent are not significantly different.

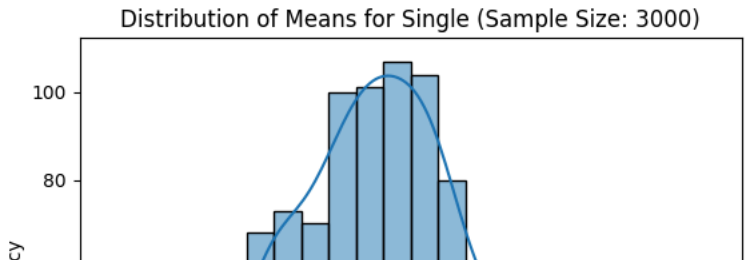
```
#iv) How does the sample size affect the shape of the distributions of the means?

# Plotting the distribution of means for different sample sizes
def plot_distribution_of_means(df, marital_status, sample_size_s):
    means = []
    for _ in range(1000):
        sample_s = df[df['Marital_Status'] == marital_status].sample(n=sample_size_s, replace=True)
        means.append(np.mean(sample_s['Purchase']))
    sns.histplot(means, kde=True)
    if marital_status == 0:
        plt.title(f'Distribution of Means for Single (Sample Size: {sample_size_s})')
        plt.xlabel('Mean Amount Spent')
        plt.ylabel('Frequency')
        plt.show()
    else:
      plt.title(f'Distribution of Means for Married (Sample Size: {sample_size_s})')
      plt.xlabel('Mean Amount Spent')
      plt.ylabel('Frequency')
      plt.show()

# Plot distributions for different sample sizes
for size in sample_sizes_s:
    plot_distribution_of_means(df, 1, size)
    plot_distribution_of_means(df, 0, size)
```

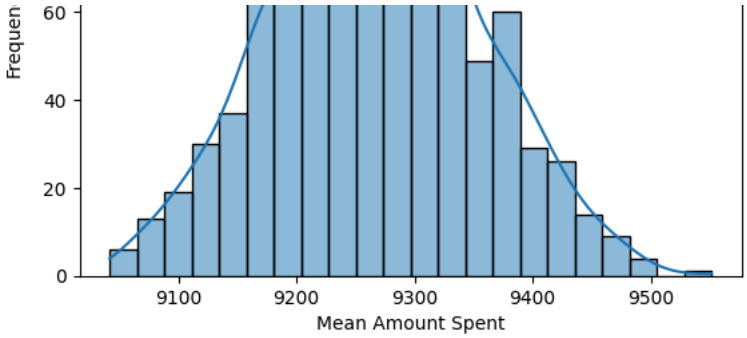
[4]

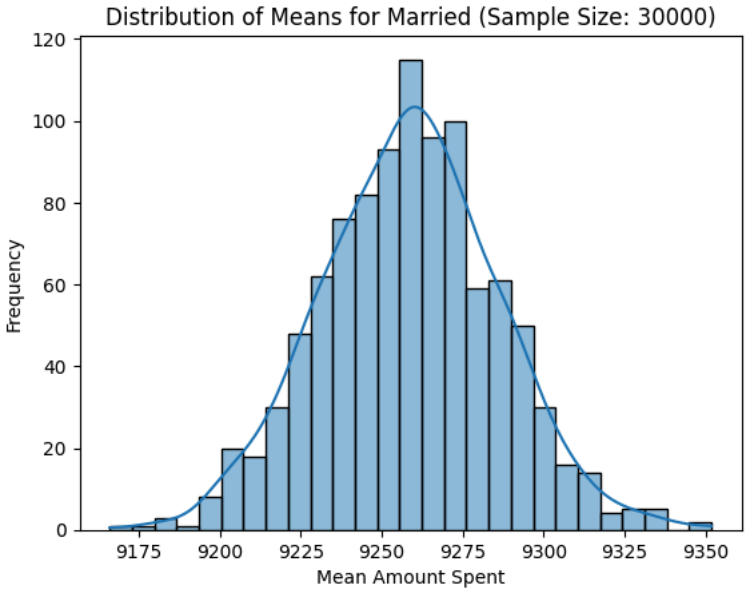

Distribution of Means for Married (Sample Size: 300)
Frequency
Mean Amount Spent
0
20
40
60
80
100
8500
8750
9000
9250
9500
9750
10000

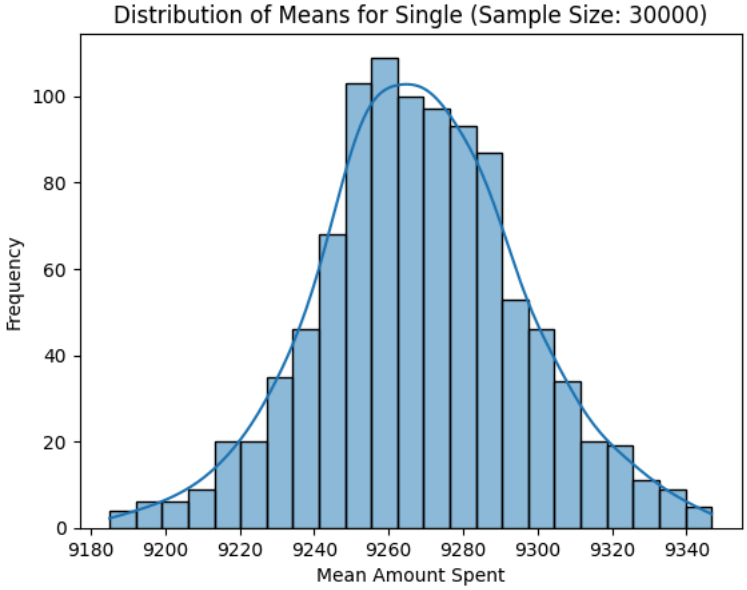

Distribution of Means for Single (Sample Size: 300)
Frequency
Mean Amount Spent
0
20
40
60
80
100
8250
8500
8750
9000
9250
9500
9750
10000


Distribution of Means for Married (Sample Size: 3000)
Frequency
Mean Amount Spent
0
20
40
60
80
100
8900
9000
9100
9200
9300
9400
9500


Distribution of Means for Single (Sample Size: 3000)
100
80


Frequen
60
40
20
0
9100
9200
9300
9400
9500
Mean Amount Spent


Distribution of Means for Married (Sample Size: 30000)
Frequency
120
100
80
60
40
20
0
9175
9200
9225
9250
9275
9300
9325
9350
Mean Amount Spent


Distribution of Means for Single (Sample Size: 30000)
Frequency
100
80
60
40
20
0
9180
9200
9220
9240
9260
9280
9300
9320
9340
Mean Amount Spent

Insight:

1) Smaller Sample Sizes: The distribution of means will be wider and less smooth, indicating more variability and less precision.

2) Larger Sample Sizes: The distribution of means will be narrower and more normally distributed, indicating less variability and more precision.

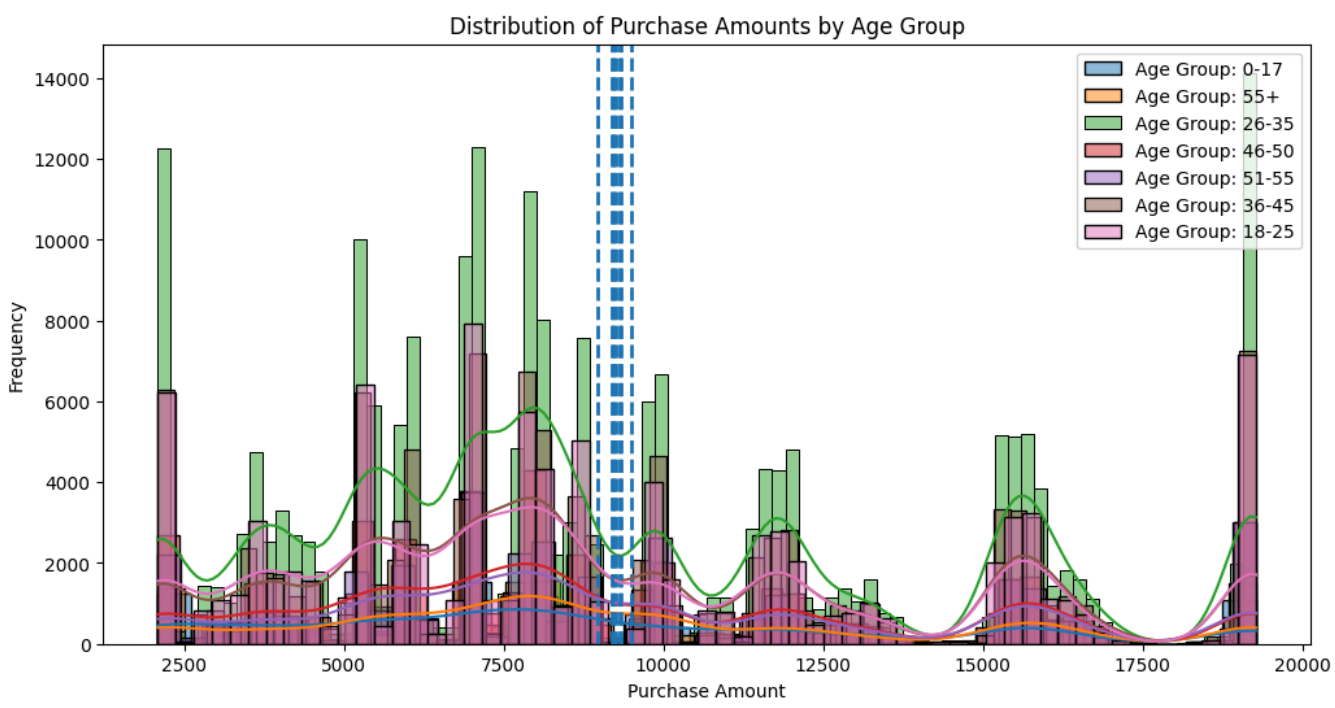
```
Larger sample sizes result in distributions of the means that are more normally distributed and have less variability.
```

6) How does Age affect the amount spent?

```
# Calculate the average purchase amount per transaction for different age groups
age_groups = df['Age'].unique()
average_purchase_age_group = {age_group: df[df['Age'] == age_group]['Purchase'].mean() for age_group in age_groups}

# Print the results
for age_group in age_groups:
    print(f"Average purchase amount per transaction for age group {age_group}: {average_purchase_age_group[age_group]}")
```

```
Average purchase amount per transaction for age group 0-17: 8965.18847006652
Average purchase amount per transaction for age group 55+: 9335.713281732276
Average purchase amount per transaction for age group 26-35: 9250.89605414614
Average purchase amount per transaction for age group 46-50: 9207.587701389197
Average purchase amount per transaction for age group 51-55: 9501.486962724222
Average purchase amount per transaction for age group 36-45: 9324.633768960848
Average purchase amount per transaction for age group 18-25: 9191.155654061511
```

```
# Plot the distribution of the mean of the expenses by age group
plt.figure(figsize=(12, 6))
for age_group in age_groups:
    sns.histplot(df[df['Age'] == age_group]['Purchase'], kde=True, label=f'Age Group: {age_group}')
    plt.axvline(average_purchase_age_group[age_group], linestyle='dashed', linewidth=2)

plt.title('Distribution of Purchase Amounts by Age Group')
plt.xlabel('Purchase Amount')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```




```
# Function to compute bootstrap 90% confidence intervals

def bootstrap_ci_a(data, n_bootstrap=1000, ci=90):
    means = []
    for _ in range(n_bootstrap):
        sample_a = np.random.choice(data, size=len(data), replace=True)
        means.append(np.mean(sample_a))
    lower_bound = np.percentile(means, (100 - ci) / 2)
```

```
    upper_bound = np.percentile(means, 100 - (100 - ci) / 2)
    return lower_bound, upper_bound

age_groups = df['Age'].unique()
ci_age_groups_90 = {}

for age in age_groups:
    ci_age_groups_90[age] = bootstrap_ci_a(df[df['Age'] == age]['Purchase'])

# Sort the confidence intervals by age
ci_age_groups_sorted_90 = dict(sorted(ci_age_groups_90.items()))

# Print the confidence intervals for each age group
for age, ci in ci_age_groups_sorted_90.items():
    print(f"90% Confidence Interval for Age {age} (entire dataset): {ci}")
```

```
90% Confidence Interval for Age 0-17 (entire dataset): (8897.085550628233, 9033.707780689376)
90% Confidence Interval for Age 18-25 (entire dataset): (9168.091144100463, 9218.149722064432)
90% Confidence Interval for Age 26-35 (entire dataset): (9234.109527832774, 9267.529319950274)
90% Confidence Interval for Age 36-45 (entire dataset): (9301.717870859005, 9347.235900128017)
90% Confidence Interval for Age 46-50 (entire dataset): (9169.633230881896, 9246.456602015001)
90% Confidence Interval for Age 51-55 (entire dataset): (9462.732214091353, 9542.4712995007)
90% Confidence Interval for Age 55+ (entire dataset): (9280.88507289045, 9388.225889168527)
```

```
# Function to compute bootstrap 95 %confidence intervals

def bootstrap_ci_a(data, n_bootstrap=1000, ci=95):
    means = []
    for _ in range(n_bootstrap):
        sample_a = np.random.choice(data, size=len(data), replace=True)
        means.append(np.mean(sample_a))
    lower_bound = np.percentile(means, (100 - ci) / 2)
    upper_bound = np.percentile(means, 100 - (100 - ci) / 2)
    return lower_bound, upper_bound

age_groups = df['Age'].unique()
ci_age_groups = {}

for age in age_groups:
    ci_age_groups[age] = bootstrap_ci_a(df[df['Age'] == age]['Purchase'])

# Sort the confidence intervals by age
ci_age_groups_sorted = dict(sorted(ci_age_groups.items()))

# Print the confidence intervals for each age group
for age, ci in ci_age_groups_sorted.items():
    print(f"95% Confidence Interval for Age {age} (entire dataset): {ci}")
```

```
95% Confidence Interval for Age 0-17 (entire dataset): (8891.448877914398, 9044.76479876369)
95% Confidence Interval for Age 18-25 (entire dataset): (9163.107699525277, 9221.41869294206)
95% Confidence Interval for Age 26-35 (entire dataset): (9230.727633408536, 9271.29145667388)
95% Confidence Interval for Age 36-45 (entire dataset): (9295.31844785824, 9351.887640335608)
95% Confidence Interval for Age 46-50 (entire dataset): (9164.05274011362, 9254.950564777417)
95% Confidence Interval for Age 51-55 (entire dataset): (9452.56496248646, 9548.984124772145)
95% Confidence Interval for Age 55+ (entire dataset): (9272.091118999004, 9398.405954698703)
```

Insight:

1) Confidence Interval for Age 0-17:

- The 95% confidence interval for the average amount spent by individuals aged 0-17 is approximately (8891.44, 9044.15).
- This indicates that we are 95% confident that the true average amount spent by individuals in this age group falls within this range.

2) Confidence Interval for Age 18-25:

- The 95% confidence interval for the average amount spent by individuals aged 18-25 is approximately (9163.66, 9221.50).
- This indicates that we are 95% confident that the true average amount spent by individuals in this age group falls within this range.

3) Confidence Interval for Age 26-35:

- The 95% confidence interval for the average amount spent by individuals aged 26-35 is approximately (9230.72, 9271.29).
- This indicates that we are 95% confident that the true average amount spent by individuals in this age group falls within this range.

4) Confidence Interval for Age 36-45:

- The 95% confidence interval for the average amount spent by individuals aged 36-45 is approximately (9295.31, 9351.88).
- This indicates that we are 95% confident that the true average amount spent by individuals in this age group falls within this range.

5) Confidence Interval for Age 46-50:

- The 95% confidence interval for the average amount spent by individuals aged 46-50 is approximately (9164.05, 9254.95).
- This indicates that we are 95% confident that the true average amount spent by individuals in this age group falls within this range.

6) Confidence Interval for Age 51-55:

- The 95% confidence interval for the average amount spent by individuals aged 51-55 is approximately (9452.56, 9548.98).
- This indicates that we are 95% confident that the true average amount spent by individuals in this age group falls within this range.

7) Confidence Interval for Age 55+:

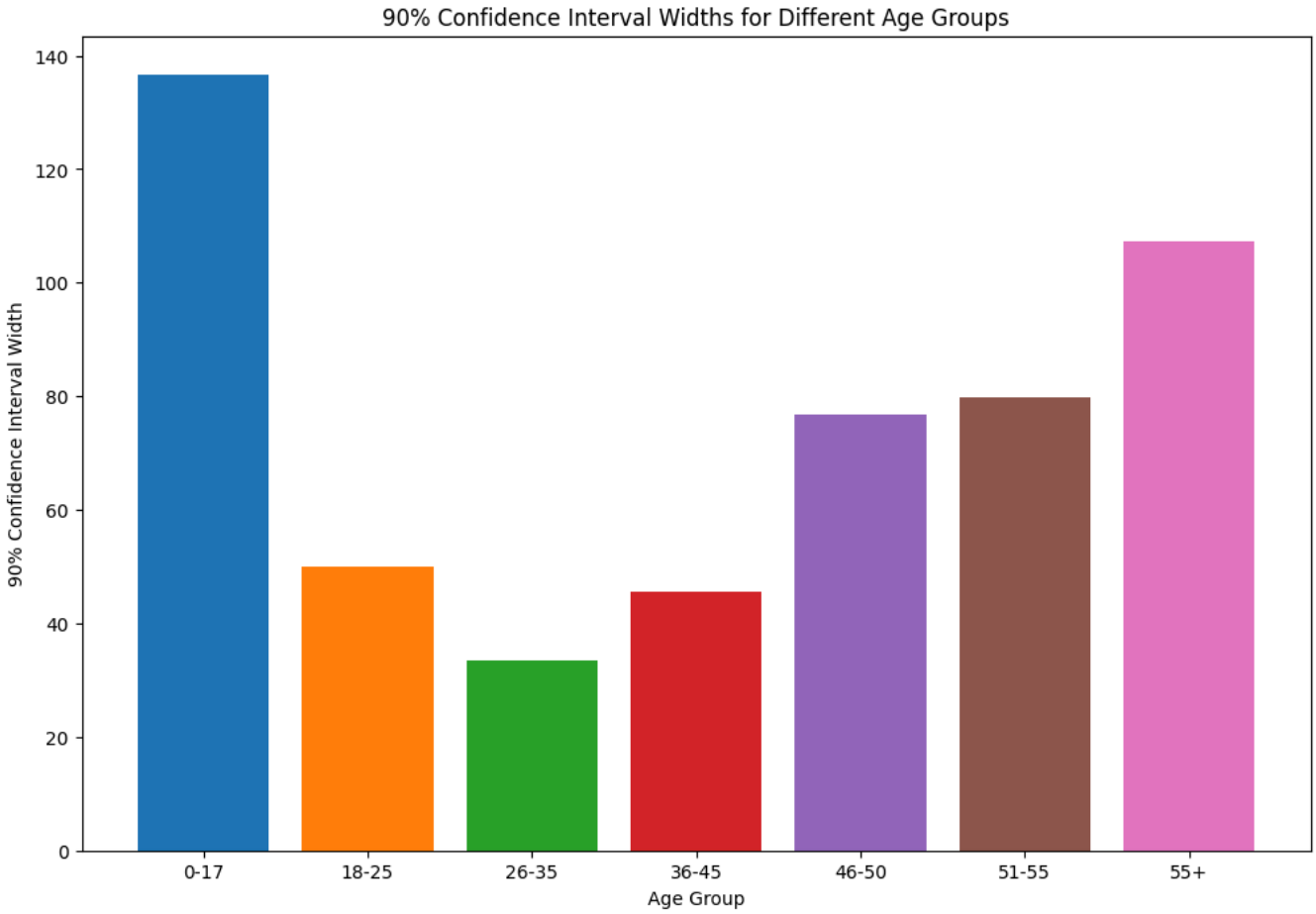
- The 95% confidence interval for the average amount spent by individuals aged 55+ is approximately (9272.09, 9398.40).
- This indicates that we are 95% confident that the true average amount spent by individuals in this age group falls within this range.

```
# Function to compute bootstrap 99% confidence intervals

def bootstrap_ci_a(data, n_bootstrap=1000, ci=99):
    means = []
    for _ in range(n_bootstrap):
        sample_a = np.random.choice(data, size=len(data), replace=True)
        means.append(np.mean(sample_a))
    lower_bound = np.percentile(means, (100 - ci) / 2)
    upper_bound = np.percentile(means, 100 - (100 - ci) / 2)
    return lower_bound, upper_bound

age_groups = df['Age'].unique()
ci_age_groups_99 = {}

for age in age_groups:
    ci_age_groups_99[age] = bootstrap_ci_a(df[df['Age'] == age]['Purchase'])

# Sort the confidence intervals by age
ci_age_groups_sorted_99 = dict(sorted(ci_age_groups_99.items()))

# Print the confidence intervals for each age group
for age, ci in ci_age_groups_sorted_99.items():
    print(f"90% Confidence Interval for Age {age} (entire dataset): {ci}")
```

```
90% Confidence Interval for Age 0-17 (entire dataset): (8865.433770073238, 9051.256067997045)
90% Confidence Interval for Age 18-25 (entire dataset): (9152.819315000404, 9230.678515428468)
90% Confidence Interval for Age 26-35 (entire dataset): (9223.342254132327, 9276.041040632626)
90% Confidence Interval for Age 36-45 (entire dataset): (9287.952133239392, 9358.373525359415)
90% Confidence Interval for Age 46-50 (entire dataset): (9152.757605410325, 9263.950567329457)
90% Confidence Interval for Age 51-55 (entire dataset): (9437.758071883338, 9560.92837780361)
90% Confidence Interval for Age 55+ (entire dataset): (9259.980178308562, 9413.09529298637)
```

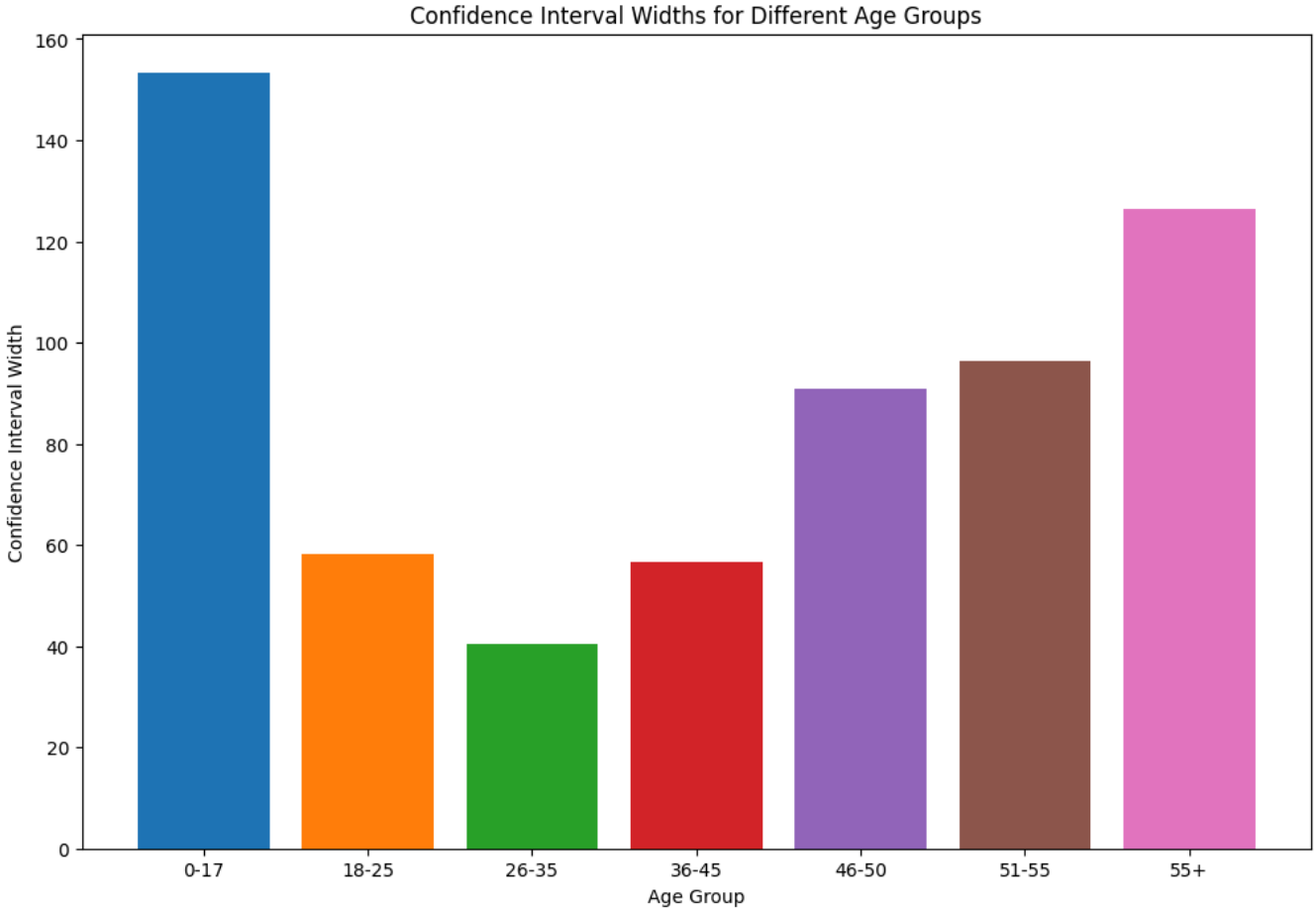
```
#How is the width of the 90% confidence interval affected by the entire size?

# Calculate the widths of the confidence intervals for each age group
ci_age_groups_sorted_90 = dict(sorted(ci_age_groups_90.items()))
ci_widths_a_90 = {age: ci[1] - ci[0] for age, ci in ci_age_groups_sorted_90.items()}


# Print the confidence intervals and their widths for each age group
for age, ci in ci_age_groups_sorted_90.items():
    print(f"90% Confidence Interval for Age {age} (entire dataset): {ci}, Width: {ci_widths_a_90[age]}")
#Plot the confidence interval widths for each age group:

# Bar Plotting the confidence interval widths

plt.figure(figsize=(12, 8))
ages_sorted_90 = sorted(ci_widths_a_90.keys())
widths_sorted_90 = [ci_widths_a_90[age] for age in ages_sorted_90]
for i in range(len(ages_sorted_90)):
    plt.bar(ages_sorted_90[i], widths_sorted_90[i], label=f'Age {ages_sorted_90[i]}')
plt.xlabel('Age Group')
plt.ylabel('90% Confidence Interval Width')
plt.title('90% Confidence Interval Widths for Different Age Groups')
```

```
90% Confidence Interval for Age 0-17 (entire dataset): (8897.085550628233, 9033.707780689376), Width: 136.62223006114255
90% Confidence Interval for Age 18-25 (entire dataset): (9168.091144100463, 9218.149722064432), Width: 50.05857796396958
90% Confidence Interval for Age 26-35 (entire dataset): (9234.109527832774, 9267.529319950274), Width: 33.41979211750004
90% Confidence Interval for Age 36-45 (entire dataset): (9301.717870859005, 9347.235900128017), Width: 45.518029269011095
90% Confidence Interval for Age 46-50 (entire dataset): (9169.633230881896, 9246.456602015001), Width: 76.82337113310496
90% Confidence Interval for Age 51-55 (entire dataset): (9462.732214091353, 9542.4712995007), Width: 79.73908540934644
90% Confidence Interval for Age 55+ (entire dataset): (9280.88507289045, 9388.225889168527), Width: 107.34081627807609
Text(0.5, 1.0, '90% Confidence Interval Widths for Different Age Groups')
```



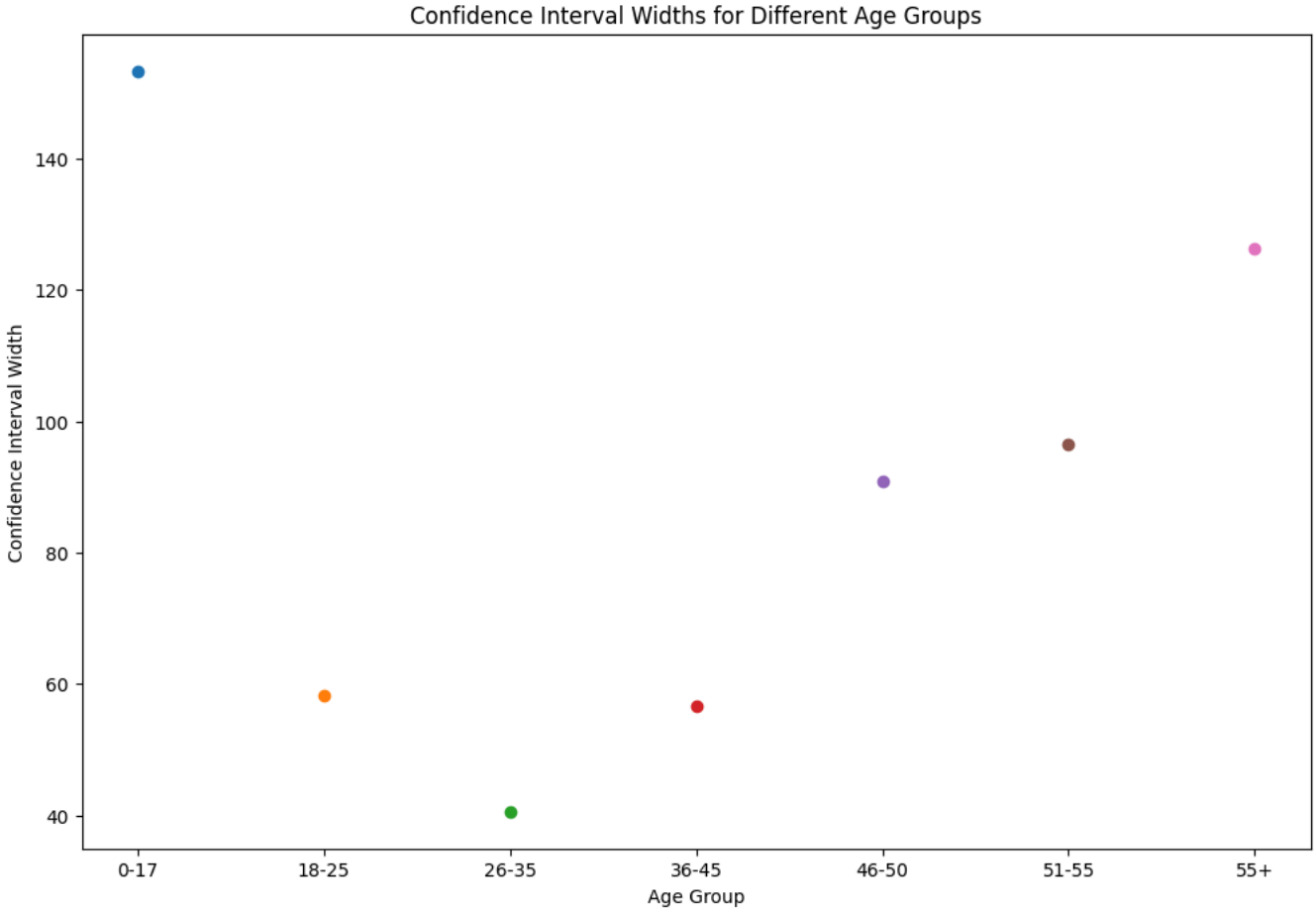

```
#ii)How is the width of the 95% confidence interval affected by the entire size?

# Calculate the widths of the confidence intervals for each age group
ci_age_groups_sorted = dict(sorted(ci_age_groups.items()))
ci_widths_a = {age: ci[1] - ci[0] for age, ci in ci_age_groups_sorted.items()}


# Print the confidence intervals and their widths for each age group
for age, ci in ci_age_groups_sorted.items():
    print(f"95% Confidence Interval for Age {age} (entire dataset): {ci}, Width: {ci_widths_a[age]}")
#Plot the confidence interval widths for each age group:

# Bar Plotting the confidence interval widths

plt.figure(figsize=(12, 8))
ages_sorted = sorted(ci_widths_a.keys())
widths_sorted = [ci_widths_a[age] for age in ages_sorted]
for i in range(len(ages_sorted)):
    plt.bar(ages_sorted[i], widths_sorted[i], label=f'Age {ages_sorted[i]}')
plt.xlabel('Age Group')
plt.ylabel('95% Confidence Interval Width')
plt.title('95% Confidence Interval Widths for Different Age Groups')
```

```
95% Confidence Interval for Age 0-17 (entire dataset): (8891.448877914398, 9044.76479876369), Width: 153.31592084929252
95% Confidence Interval for Age 18-25 (entire dataset): (9163.107699525277, 9221.41869294206), Width: 58.31099341678237
95% Confidence Interval for Age 26-35 (entire dataset): (9230.727633408536, 9271.29145667388), Width: 40.56382326534367
95% Confidence Interval for Age 36-45 (entire dataset): (9295.31844785824, 9351.887640335608), Width: 56.569192477367324
95% Confidence Interval for Age 46-50 (entire dataset): (9164.05274011362, 9254.950564777417), Width: 90.8978246637962
95% Confidence Interval for Age 51-55 (entire dataset): (9452.56496248646, 9548.984124772145), Width: 96.41916228568516
95% Confidence Interval for Age 55+ (entire dataset): (9272.091118999004, 9398.405954698703), Width: 126.31483569969896
Text(0.5, 1.0, 'Confidence Interval Widths for Different Age Groups')
```



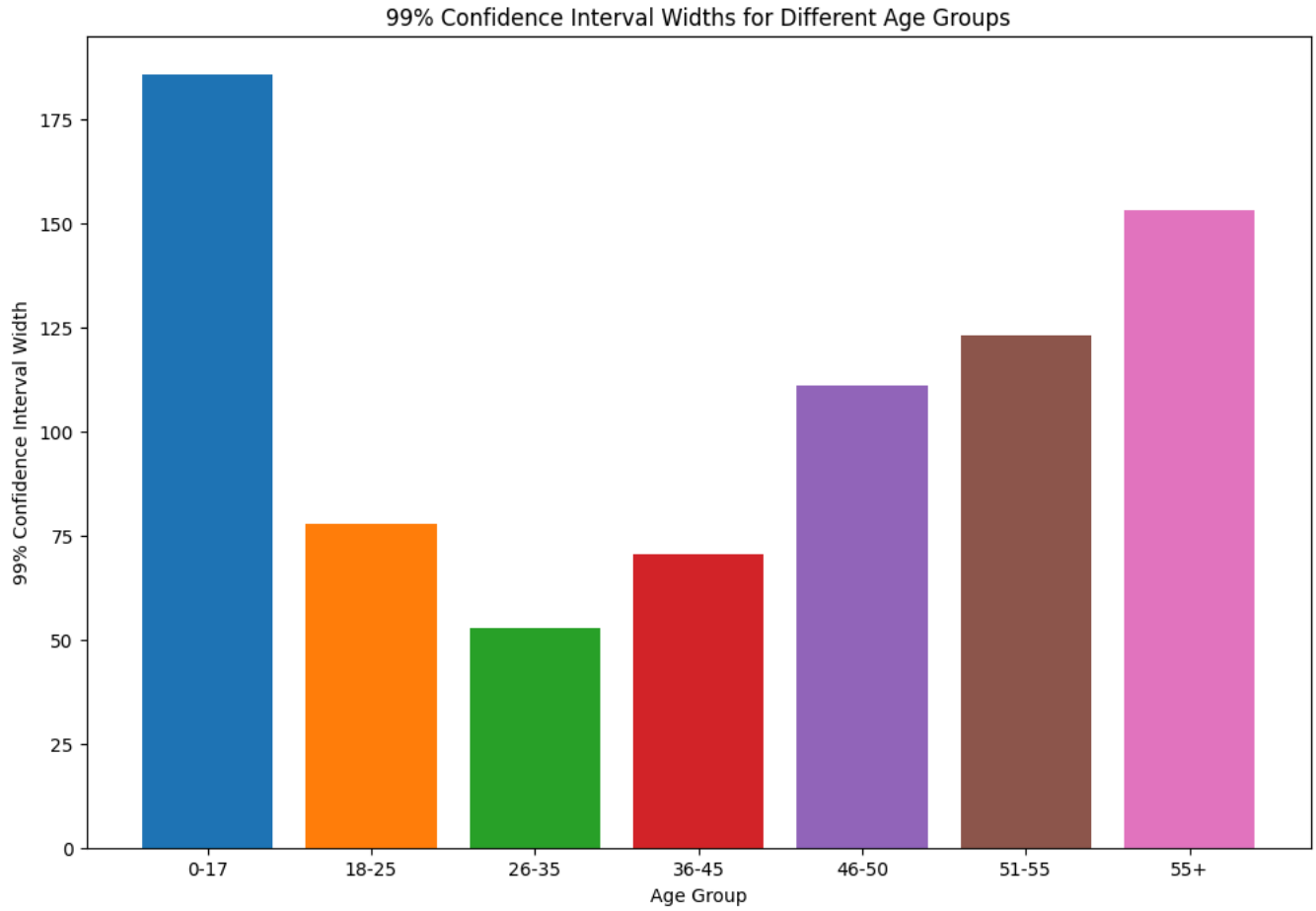

```
# Scatter Plotting the 95% confidence interval widths

plt.figure(figsize=(12, 8))
ages_sorted = sorted(ci_widths_a.keys())
widths_sorted = [ci_widths_a[age] for age in ages_sorted]
for i in range(len(ages_sorted)):
    plt.plot(ages_sorted[i], widths_sorted[i], marker='o',label=f'Age {ages_sorted[i]}')
plt.xlabel('Age Group')
plt.ylabel('Confidence Interval Width')
plt.title('Confidence Interval Widths for Different Age Groups')
```

```
Text(0.5, 1.0, 'Confidence Interval Widths for Different Age Groups')
```




```
#ii)How is the width of the 99% confidence interval affected by the entire size?

# Calculate the widths of the 99% confidence intervals for each age group
ci_age_groups_sorted_99 = dict(sorted(ci_age_groups_99.items()))
ci_widths_a_99 = {age: ci[1] - ci[0] for age, ci in ci_age_groups_sorted_99.items()}


# Print the confidence intervals and their widths for each age group
for age, ci in ci_age_groups_sorted_99.items():
    print(f"99% Confidence Interval for Age {age} (entire dataset): {ci}, Width: {ci_widths_a_99[age]}")
#Plot the confidence interval widths for each age group:

# Bar Plotting the confidence interval widths

plt.figure(figsize=(12, 8))
ages_sorted_99 = sorted(ci_widths_a_99.keys())
widths_sorted_99 = [ci_widths_a_99[age] for age in ages_sorted_99]
for i in range(len(ages_sorted)):
    plt.bar(ages_sorted_99[i], widths_sorted_99[i], label=f'Age {ages_sorted_99[i]}')
plt.xlabel('Age Group')
plt.ylabel('99% Confidence Interval Width')
plt.title('99% Confidence Interval Widths for Different Age Groups')
```

```
99% Confidence Interval for Age 0-17 (entire dataset): (8865.433770073238, 9051.256067997045), Width: 185.82229792380713
99% Confidence Interval for Age 18-25 (entire dataset): (9152.819315000404, 9230.678515428468), Width: 77.85920042806356
99% Confidence Interval for Age 26-35 (entire dataset): (9223.342254132327, 9276.041040632626), Width: 52.69878650029932
99% Confidence Interval for Age 36-45 (entire dataset): (9287.952133239392, 9358.373525359415), Width: 70.42139212002257
99% Confidence Interval for Age 46-50 (entire dataset): (9152.757605410325, 9263.950567329457), Width: 111.19296191913236
99% Confidence Interval for Age 51-55 (entire dataset): (9437.758071883338, 9560.92837780361), Width: 123.17030592027186
99% Confidence Interval for Age 55+ (entire dataset): (9259.980178308562, 9413.09529298637), Width: 153.11511467780838
Text(0.5, 1.0, '99% Confidence Interval Widths for Different Age Groups')
```



Questions Answered:

1) Is the confidence interval computed using the entire dataset wider for one of the age groups? Why is this the case?

- Yes, the confidence interval is wider for the age groups 0-17 and 55+. This is likely due to greater variability in the purchase amounts within these age groups, leading to a less precise estimate of the mean.

Summary:

1) Comparison of Variability:

- The age group 26-35 has the lowest variability in spending, as indicated by the narrowest confidence interval width. The age groups 0-17 and 55+ have the highest variability in spending, suggesting more diverse spending habits within these groups.

2) Implications:

- Higher variability in spending within an age group suggests more diverse spending habits. For example, the 0-17 age group might include both low and high spenders, leading to a wider confidence interval.
- Lower variability, as seen in the 26-35 age group, indicates more consistent spending patterns within that age group.

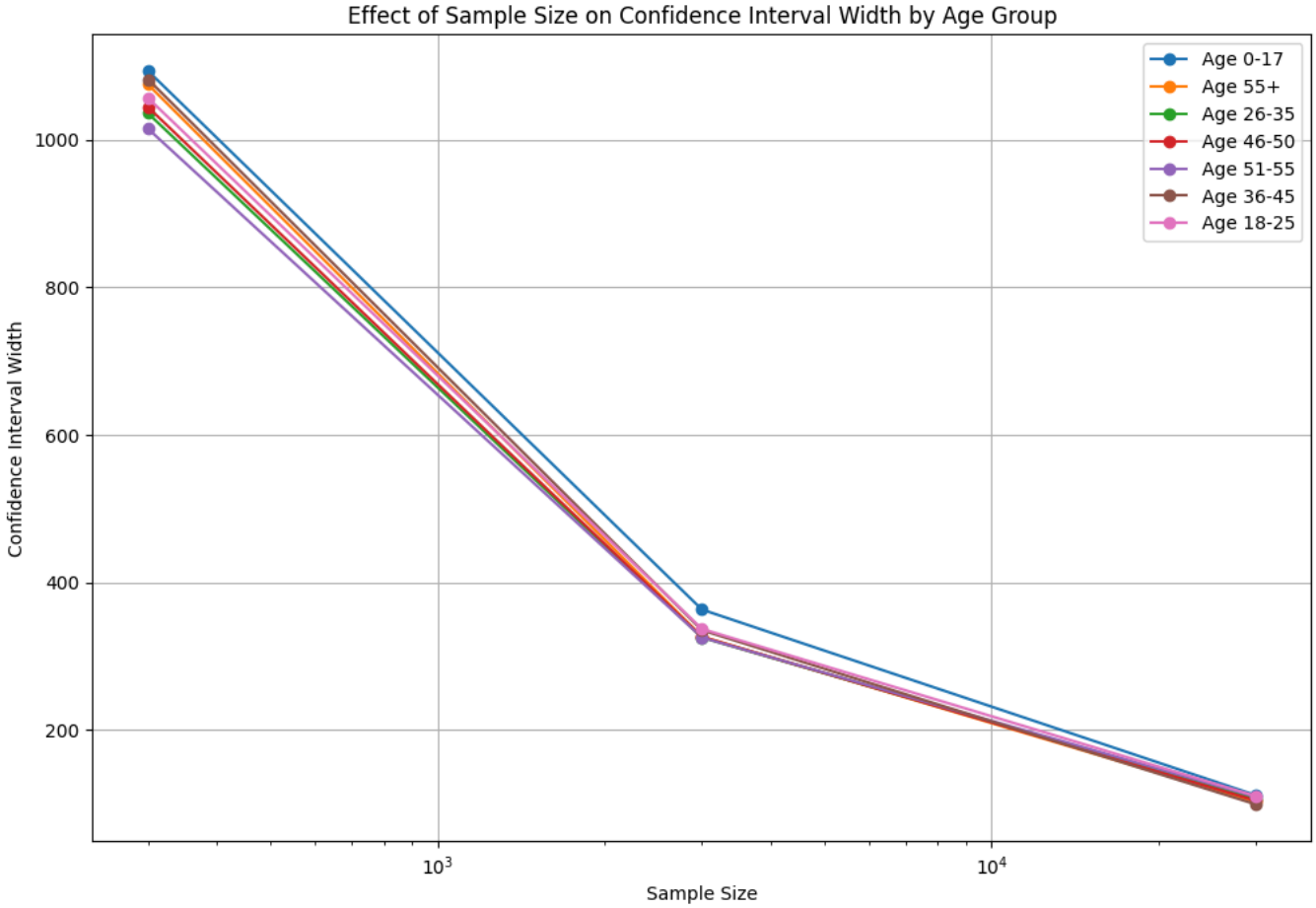
```
#ii) How is the width of the confidence interval affected by the sample size? sample sizes - 300, 3000, and 30000
def calculate_ci_for_sample_sizes(df, sample_sizes):
    results_a = {}
    age_groups = df['Age'].unique()
    for age in age_groups:
        results_a[age] = {}
        for size in sample_sizes:
            sample_a = df[df['Age'] == age].sample(n=size, replace=True)
            ci_age = bootstrap_ci_a(sample_a['Purchase'])
            results_a[age][size] = ci_age
            print(f"Sample size: {size}")
            print(f"95% CI for average amount spent by {age}: {ci_age}")
    return results_a

sample_sizes = [300, 3000, 30000]
```

```
ci_results_a = calculate_ci_for_sample_sizes(df, sample_sizes)
print(ci_results_a)
```

```
Sample size: 300
95% CI for average amount spent by 0-17: (7958.76975, 9052.232833333332)
Sample size: 3000
95% CI for average amount spent by 0-17: (8753.406091666668, 9116.77065)
Sample size: 30000
95% CI for average amount spent by 0-17: (8941.8699375, 9053.513013333333)
Sample size: 300
95% CI for average amount spent by 55+: (8806.938333333334, 9881.989249999999)
Sample size: 3000
95% CI for average amount spent by 55+: (9222.829666666667, 9549.640125)
Sample size: 30000
95% CI for average amount spent by 55+: (9249.792818333333, 9352.1421675)
Sample size: 300
95% CI for average amount spent by 26-35: (8433.557083333333, 9469.542166666666)
Sample size: 3000
95% CI for average amount spent by 26-35: (8980.558850000001, 9306.096608333333)
Sample size: 30000
95% CI for average amount spent by 26-35: (9194.091662499999, 9300.640623333333)
Sample size: 300
95% CI for average amount spent by 46-50: (8320.609916666666, 9364.983916666666)
Sample size: 3000
95% CI for average amount spent by 46-50: (9250.722741666668, 9577.152941666667)
Sample size: 30000
95% CI for average amount spent by 46-50: (9132.909045, 9237.340361666666)
Sample size: 300
95% CI for average amount spent by 51-55: (8889.115333333333, 9904.1885)
Sample size: 3000
95% CI for average amount spent by 51-55: (9279.4266, 9604.969325)
Sample size: 30000
95% CI for average amount spent by 51-55: (9436.312662499999, 9545.793020833333)
Sample size: 300
95% CI for average amount spent by 36-45: (8610.6335, 9692.710416666667)
Sample size: 3000
95% CI for average amount spent by 36-45: (9101.040483333334, 9436.418608333333)
Sample size: 30000
95% CI for average amount spent by 36-45: (9318.417716666667, 9417.723295833333)
Sample size: 300
95% CI for average amount spent by 18-25: (8108.8525, 9165.626166666667)
Sample size: 3000
95% CI for average amount spent by 18-25: (9088.515708333332, 9425.632116666668)
Sample size: 30000
95% CI for average amount spent by 18-25: (9176.579157499998, 9287.177909166667)
{'0-17': {300: (7958.76975, 9052.232833333332), 3000: (8753.406091666668, 9116.77065), 30000: (8941.8699375, 9053.513013333333)}, '5
```

```
ci_widths_a = {age: {size: ci[1] - ci[0] for size, ci in sizes.items()} for age, sizes in ci_results_a.items()}
plt.figure(figsize=(12, 8))
for age, widths in ci_widths_a.items():
    print(f"Confidence Interval Widths for Age {age}: {widths}")
    sizes = list(widths.keys())
    plt.plot(sizes, list(widths.values()), marker='o', label=f'Age {age}')

plt.xlabel('Sample Size')
plt.ylabel('Confidence Interval Width')
plt.title('Effect of Sample Size on Confidence Interval Width by Age Group')
plt.legend()
plt.grid(True)
plt.xscale('log')
plt.show()
```

```
Confidence Interval Widths for Age 0-17: {300: 1093.4630833333313, 3000: 363.36455833333275, 30000: 111.64307583333357}
Confidence Interval Widths for Age 55+: {300: 1075.0509166666652, 3000: 326.8104583333334, 30000: 102.3493491666668}
Confidence Interval Widths for Age 26-35: {300: 1035.985083333333, 3000: 325.537758333332, 30000: 106.5489608333337}
Confidence Interval Widths for Age 46-50: {300: 1044.3739999999998, 3000: 326.4301999999989, 30000: 104.43131666666523}
Confidence Interval Widths for Age 51-55: {300: 1015.073166666667, 3000: 325.54272499999934, 30000: 109.48035833333415}
Confidence Interval Widths for Age 36-45: {300: 1082.0769166666669, 3000: 335.3781249999993, 30000: 99.30557916666658}
Confidence Interval Widths for Age 18-25: {300: 1056.773666666667, 3000: 337.11640833333513, 30000: 110.59875166666825}
```



Insights:

```
Question Answered:
```

1) How is the width of the confidence interval affected by the sample size?

- Precision: As the sample size increases, the confidence interval becomes narrower, indicating a more precise estimate of the average amount spent.
- Reliability: Larger samples provide more reliable estimates because they reduce the impact of sampling error.

```
The width decreases significantly as the sample size increases, indicating more precise estimates with larger samples.
```

```
#iii)Do the confidence intervals for different sample sizes overlap?
# Check if confidence intervals overlap
overlap_results = {}
for age in ci_results_a:
    overlaps = []
    all_lower, all_upper = ci_results_a[age][30000]  # Using the largest sample size as reference
    for size in sample_sizes:
        lower, upper = ci_results_a[age][size]
        overlaps.append(lower <= all_upper and upper >= all_lower)
    overlap_results[age] = any(overlaps)

print("Do the confidence intervals for different sample sizes overlap?")
for age, overlap in overlap_results.items():
    print(f"Age {age}: {'Yes' if overlap else 'No'}")
```

```
Do the confidence intervals for different sample sizes overlap?
Age 0-17: Yes
Age 55+: Yes
Age 26-35: Yes
Age 46-50: Yes
Age 51-55: Yes
Age 36-45: Yes
```

```
 Age 18-25: Yes


Question Answered:
```

1) Do the confidence intervals for different sample sizes overlap?

- Yes, the confidence intervals for each groupsoverlap, indicating that the average amounts spent are not significantly different.

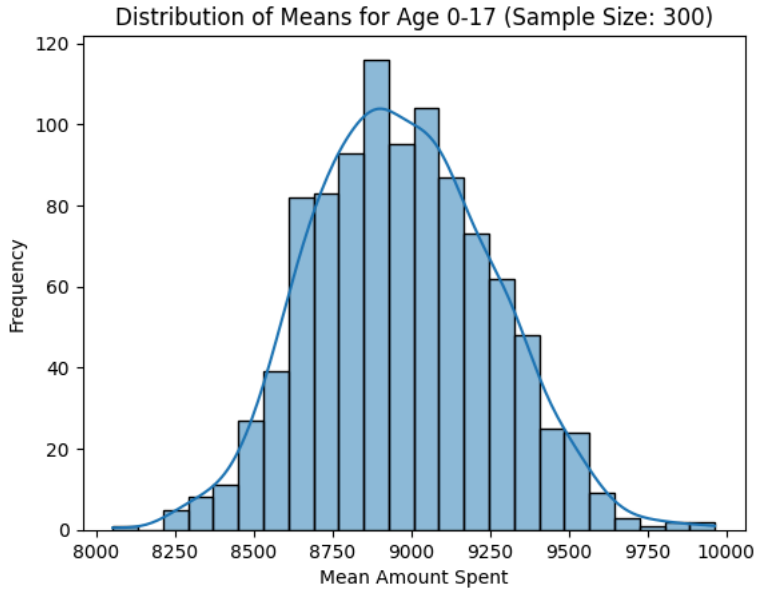
```
# Function to calculate confidence intervals for different sample sizes
def calculate_ci_for_sample_sizes(df, sample_sizes):
    results_a1 = {}
    age_groups = df['Age'].unique()
    for age in age_groups:
        results_a1[age] = {}
        for size in sample_sizes:
            sample_a = df[df['Age'] == age].sample(n=size, replace=True)
            ci_age = bootstrap_ci_a(sample_a['Purchase'])
            results_a1[age][size] = ci_age
            print(f"Sample size: {size}")
            print(f"95% CI for average amount spent by {age}: {ci_age[:2]}")
    return results_a1

sample_sizes = [300, 3000, 30000]
ci_results_a1 = calculate_ci_for_sample_sizes(df, sample_sizes)
print(ci_results_a1)
```

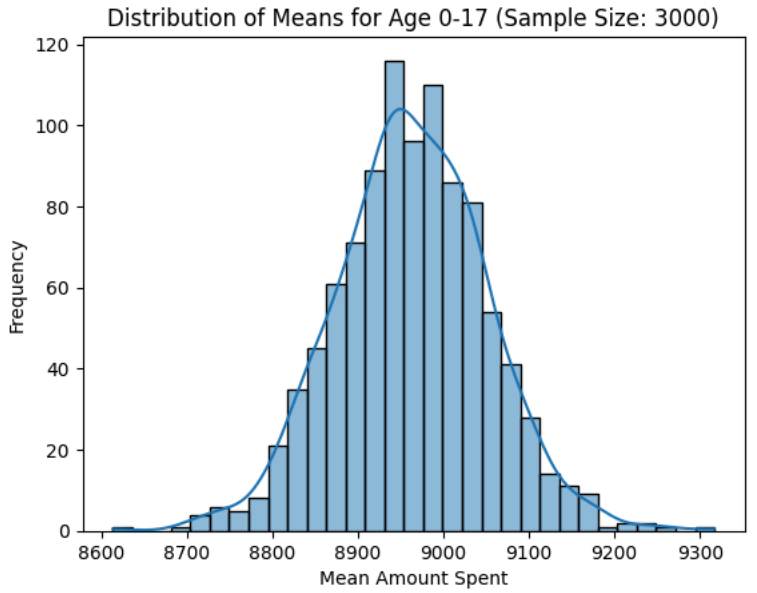
```
Sample size: 300
95% CI for average amount spent by 0-17: (8573.708916666666, 9697.915583333333)
Sample size: 3000
95% CI for average amount spent by 0-17: (8757.846858333332, 9124.56105)
Sample size: 30000
95% CI for average amount spent by 0-17: (8893.5525075, 9004.1932275)
Sample size: 300
95% CI for average amount spent by 55+: (8481.988666666666, 9479.656333333334)
Sample size: 3000
95% CI for average amount spent by 55+: (9067.69815, 9405.03195)
Sample size: 30000
95% CI for average amount spent by 55+: (9307.438609166666, 9413.363905)
Sample size: 300
95% CI for average amount spent by 26-35: (8881.477916666667, 10068.548833333332)
Sample size: 3000
95% CI for average amount spent by 26-35: (9002.396566666666, 9350.480708333334)
Sample size: 30000
95% CI for average amount spent by 26-35: (9153.610040000001, 9262.233843333333)
Sample size: 300
95% CI for average amount spent by 46-50: (8793.384416666666, 9889.390583333334)
Sample size: 3000
95% CI for average amount spent by 46-50: (8965.614425, 9289.353775000001)
Sample size: 30000
95% CI for average amount spent by 46-50: (9174.613214166666, 9284.319946666667)
Sample size: 300
95% CI for average amount spent by 51-55: (8971.999500000002, 10067.3605)
Sample size: 3000
95% CI for average amount spent by 51-55: (9535.733458333334, 9868.334491666667)
Sample size: 30000
95% CI for average amount spent by 51-55: (9424.652763333333, 9537.392956666667)
Sample size: 300
95% CI for average amount spent by 36-45: (9203.272, 10248.939833333334)
Sample size: 3000
95% CI for average amount spent by 36-45: (9184.739416666667, 9524.629933333334)
Sample size: 30000
95% CI for average amount spent by 36-45: (9244.132583333332, 9350.376109166666)
Sample size: 300
95% CI for average amount spent by 18-25: (8065.0320833333335, 9117.859416666666)
Sample size: 3000
95% CI for average amount spent by 18-25: (9046.704366666667, 9373.5085)
Sample size: 30000
95% CI for average amount spent by 18-25: (9151.911235, 9258.773160833332)
{'0-17': {300: (8573.708916666666, 9697.915583333333), 3000: (8757.846858333332, 9124.56105), 30000: (8893.5525075, 9004.1932275)},
```

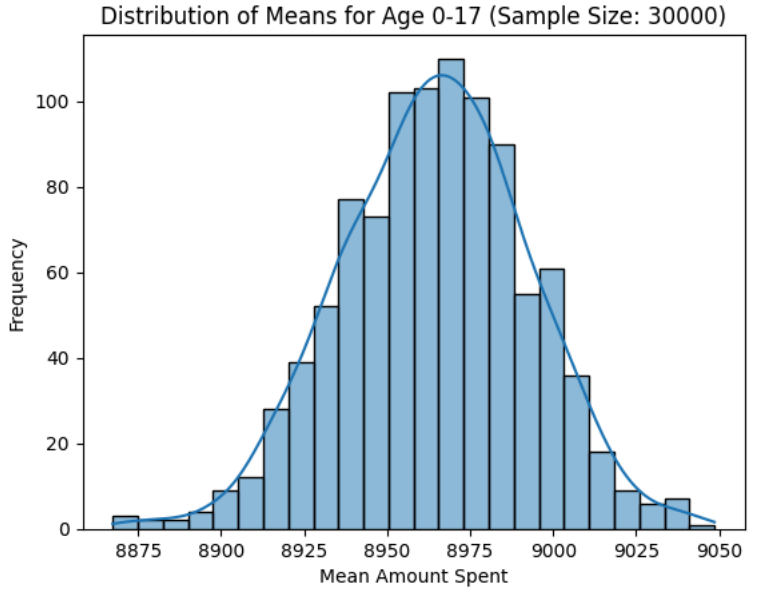
```
#iv) How does the sample size affect the shape of the distributions of the means?

# Plotting the distribution of means for different sample sizes
def plot_distribution_of_means(df, age, sample_size):
    means = []
    for _ in range(1000):
        sample_s = df[df['Age'] == age].sample(n=sample_size, replace=True)
        means.append(np.mean(sample_s['Purchase']))
    sns.histplot(means, kde=True)
    plt.title(f'Distribution of Means for Age {age} (Sample Size: {sample_size})')
    plt.xlabel('Mean Amount Spent')
    plt.ylabel('Frequency')
    plt.show()
```

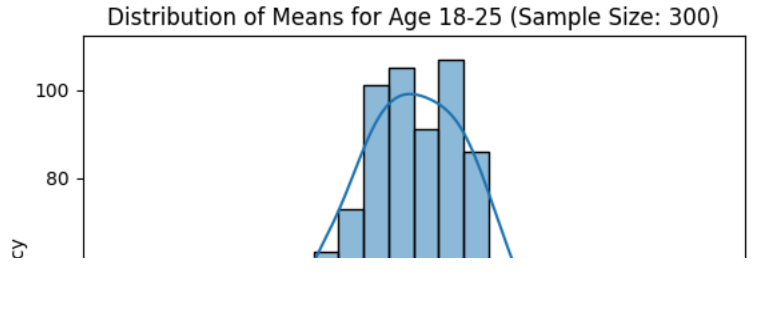
```
# Get sorted list of ages
sorted_ages = sorted(df['Age'].unique())

# Plot distributions for different sample sizes sorted by age
for age in sorted_ages:
    for size in sample_sizes:
        plot_distribution_of_means(df, age, size)
```

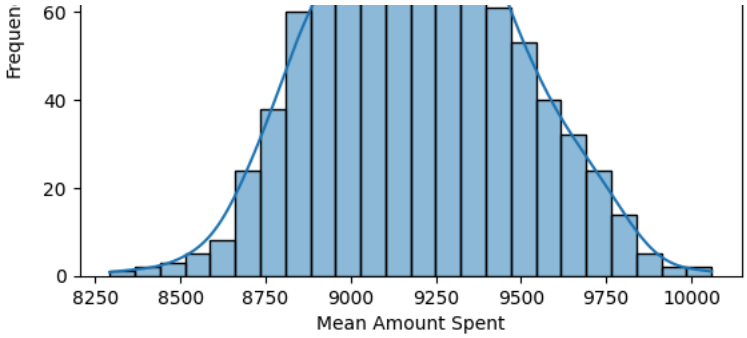
[4]

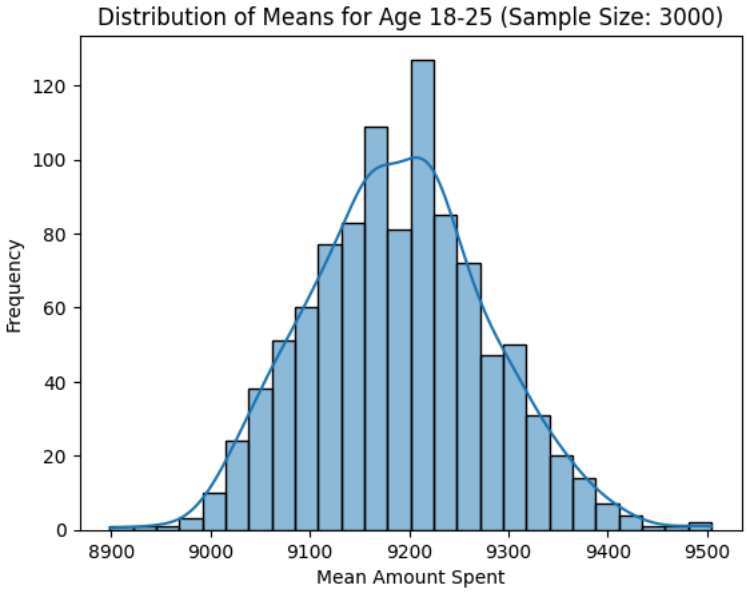

Distribution of Means for Age 0-17 (Sample Size: 300)
Frequency
Mean Amount Spent
0
20
40
60
80
100
120
8000
8250
8500
8750
9000
9250
9500
9750
10000

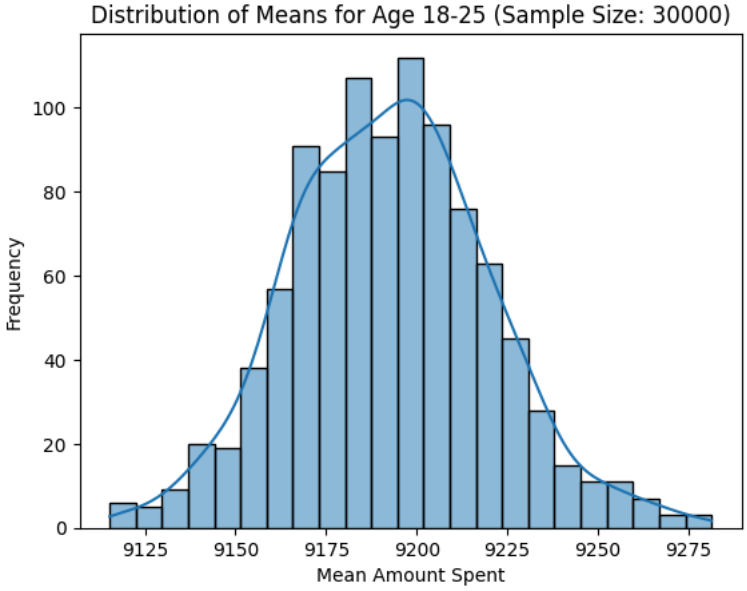

Distribution of Means for Age 0-17 (Sample Size: 3000)
Frequency
Mean Amount Spent
0
20
40
60
80
100
120
8600
8700
8800
8900
9000
9100
9200
9300

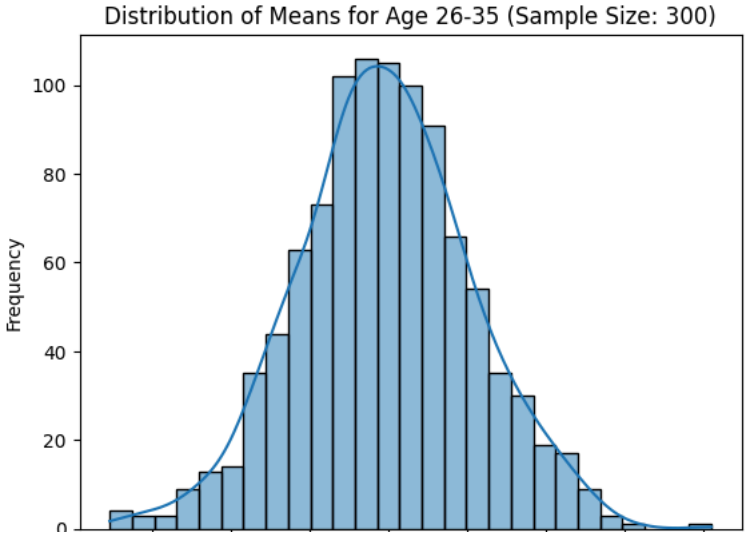

Distribution of Means for Age 0-17 (Sample Size: 30000)
Frequency
Mean Amount Spent
0
20
40
60
80
100
8875
8900
8925
8950
8975
9000
9025
9050

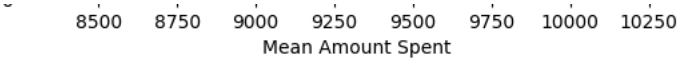

Distribution of Means for Age 18-25 (Sample Size: 300)
100
80

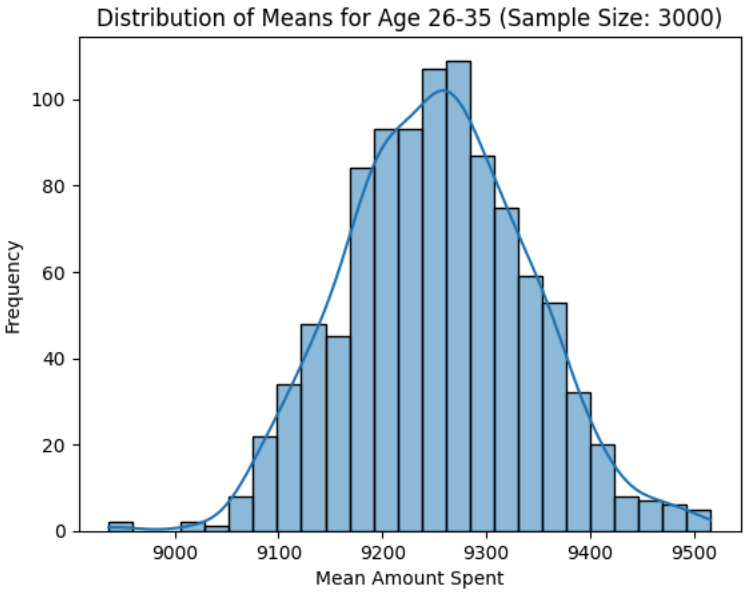

Frequen
60
40
20
0
8250
8500
8750
9000
9250
9500
9750
10000
Mean Amount Spent

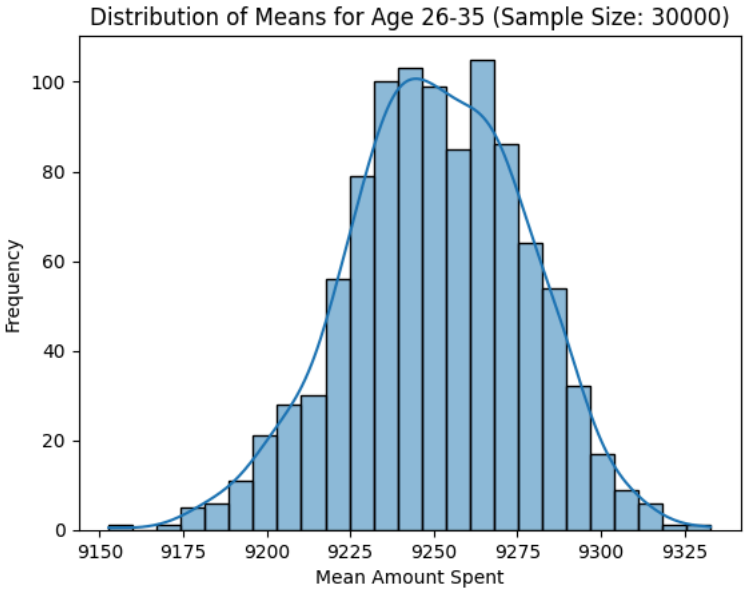

Distribution of Means for Age 18-25 (Sample Size: 3000)
Frequency
120
100
80
60
40
20
0
8900
9000
9100
9200
9300
9400
9500
Mean Amount Spent

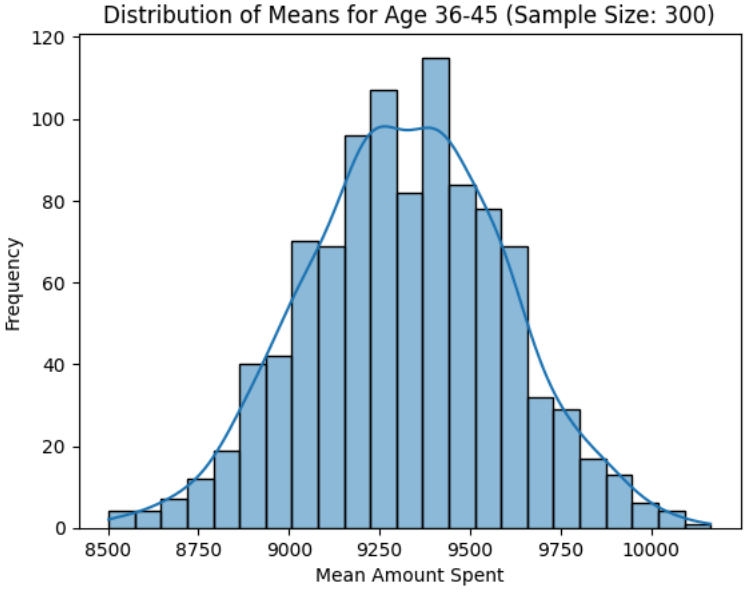

Distribution of Means for Age 18-25 (Sample Size: 30000)
Frequency
100
80
60
40
20
0
9125
9150
9175
9200
9225
9250
9275
Mean Amount Spent

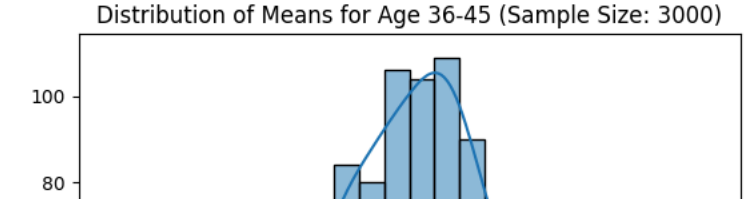

Distribution of Means for Age 26-35 (Sample Size: 300)
Frequency
100
80
60
40
20
0

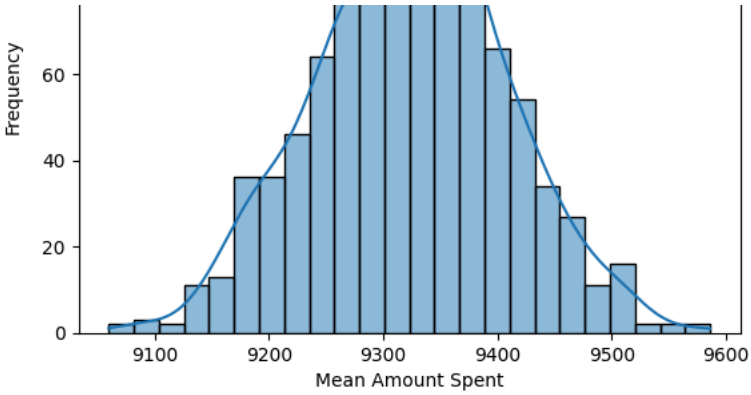

8500
8750
9000
9250
9500
9750
10000
10250
Mean Amount Spent

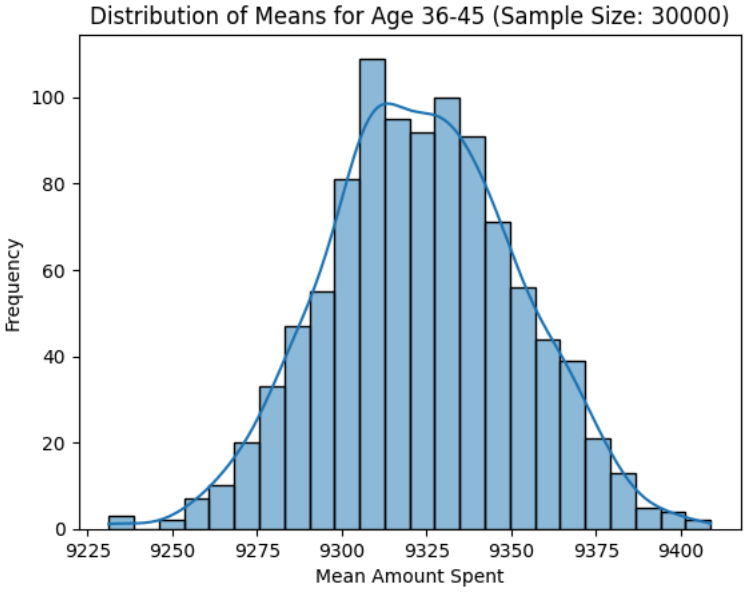

Distribution of Means for Age 26-35 (Sample Size: 3000)
Frequency
0
20
40
60
80
100
9000
9100
9200
9300
9400
9500
Mean Amount Spent

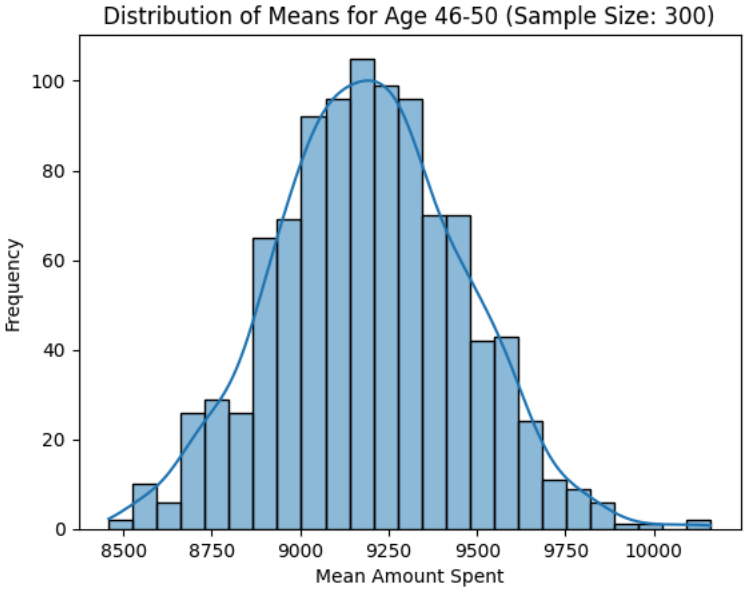

Distribution of Means for Age 26-35 (Sample Size: 30000)
Frequency
0
20
40
60
80
100
9150
9175
9200
9225
9250
9275
9300
9325
Mean Amount Spent

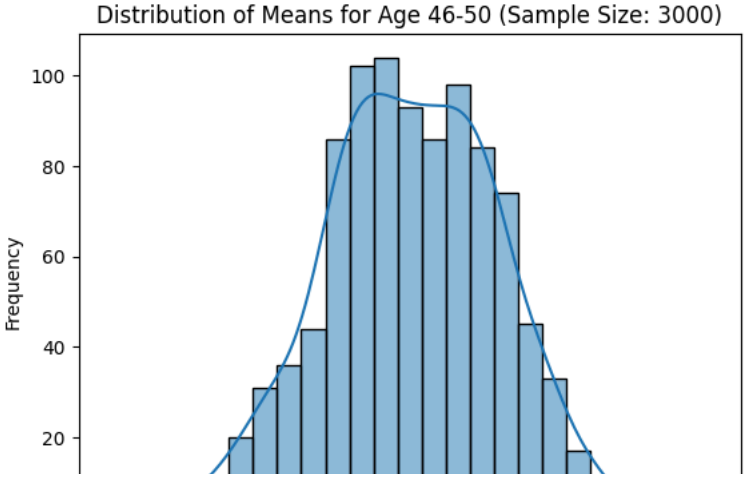

Distribution of Means for Age 36-45 (Sample Size: 300)
Frequency
0
20
40
60
80
100
120
8500
8750
9000
9250
9500
9750
10000
Mean Amount Spent

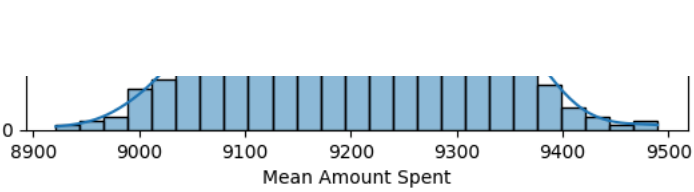

Distribution of Means for Age 36-45 (Sample Size: 3000)
80
100

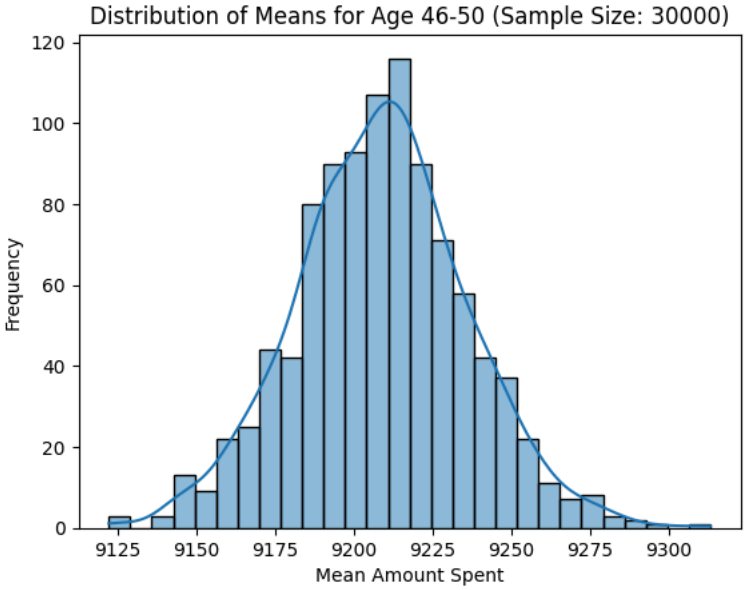

Frequency
0
20
40
60
9100
9200
9300
9400
9500
9600
Mean Amount Spent

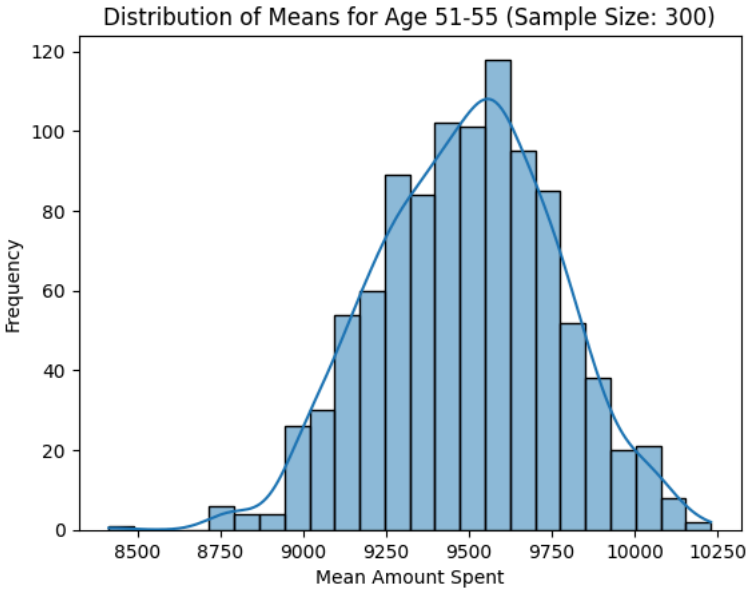

Distribution of Means for Age 36-45 (Sample Size: 30000)
Frequency
0
20
40
60
80
100
9225
9250
9275
9300
9325
9350
9375
9400
Mean Amount Spent

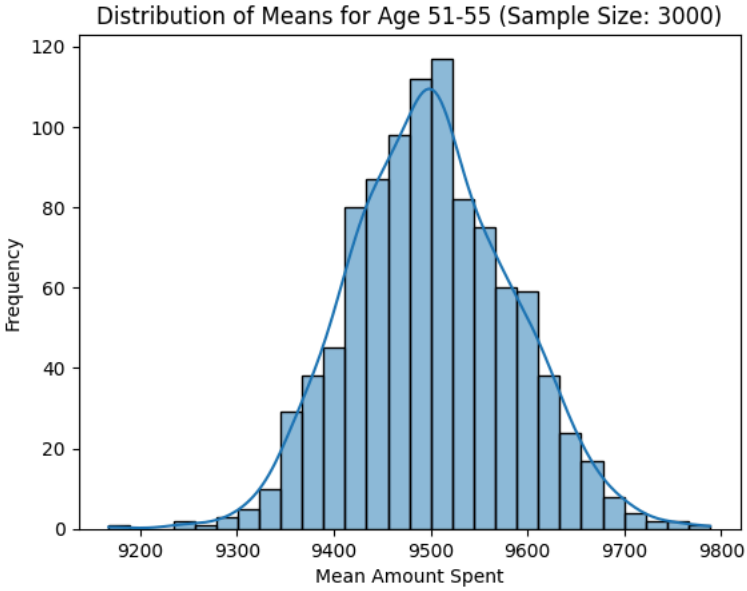

Distribution of Means for Age 46-50 (Sample Size: 300)
Frequency
0
20
40
60
80
100
8500
8750
9000
9250
9500
9750
10000
Mean Amount Spent

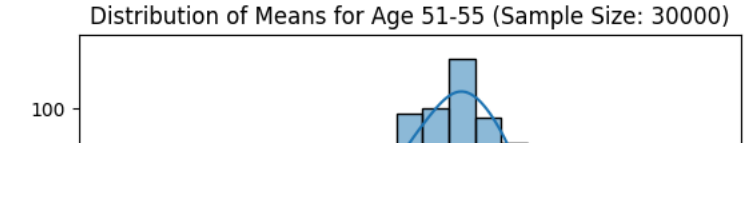

Distribution of Means for Age 46-50 (Sample Size: 3000)
Frequency
20
40
60
80
100

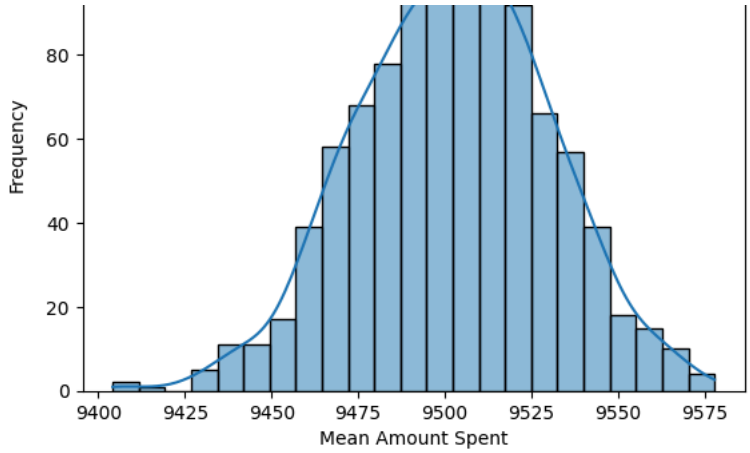

0
8900
9000
9100
9200
9300
9400
9500
Mean Amount Spent

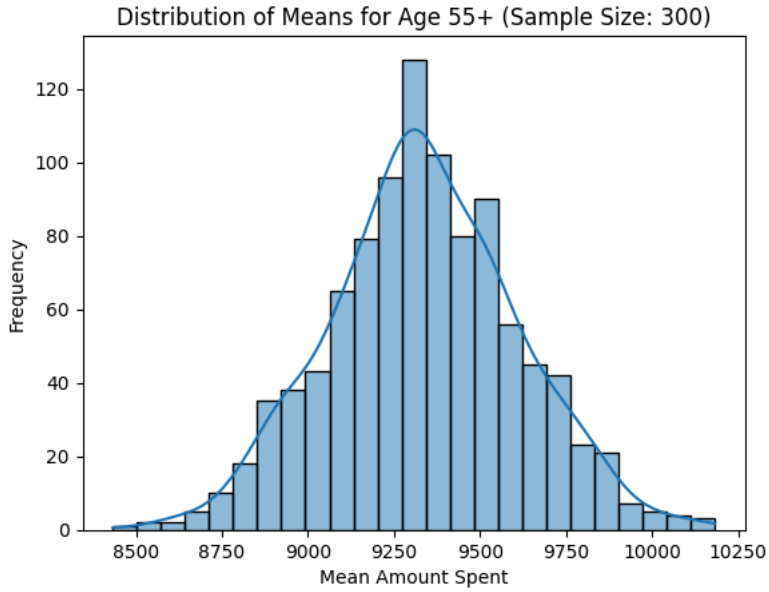

Distribution of Means for Age 46-50 (Sample Size: 30000)
Frequency
0
20
40
60
80
100
120
9125
9150
9175
9200
9225
9250
9275
9300
Mean Amount Spent

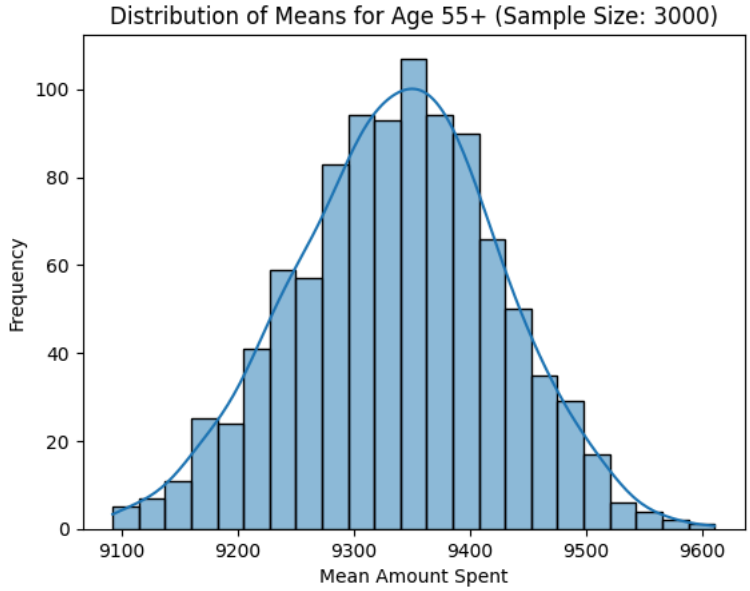

Distribution of Means for Age 51-55 (Sample Size: 300)
Frequency
0
20
40
60
80
100
120
8500
8750
9000
9250
9500
9750
10000
10250
Mean Amount Spent

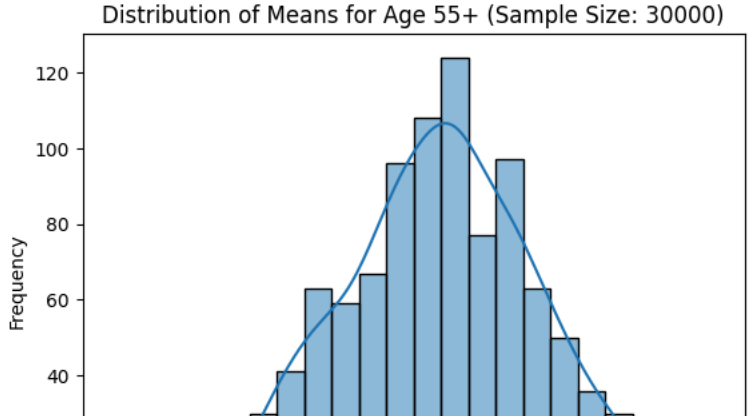

Distribution of Means for Age 51-55 (Sample Size: 3000)
Frequency
0
20
40
60
80
100
120
9200
9300
9400
9500
9600
9700
9800
Mean Amount Spent

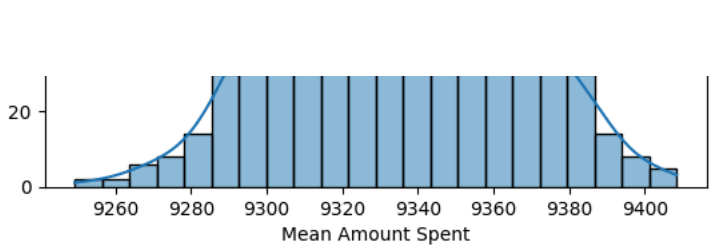

Distribution of Means for Age 51-55 (Sample Size: 30000)
100


Frequency
0
20
40
60
80
9400
9425
9450
9475
9500
9525
9550
9575
Mean Amount Spent


Distribution of Means for Age 55+ (Sample Size: 300)
Frequency
0
20
40
60
80
100
120
8500
8750
9000
9250
9500
9750
10000
10250
Mean Amount Spent


Distribution of Means for Age 55+ (Sample Size: 3000)
Frequency
0
20
40
60
80
100
9100
9200
9300
9400
9500
9600
Mean Amount Spent


Distribution of Means for Age 55+ (Sample Size: 30000)
Frequency
40
60
80
100
120


20
0
9260
9280
9300
9320
9340
9360
9380
9400
Mean Amount Spent

Question Answered:

1) How does the sample size affect the shape of the distributions of the means?

- As the sample size increases, the distribution of the sample means becomes more normal and narrower due to the central limit theorem.

7) Create a report

a) Report whether the confidence intervals for the average amount spent by males and females (computed using all the data) overlap. How can Walmart leverage this conclusion to make changes or improvements?

```
# Check for overlap for 95% Confidence Interval
overlap = not (ci_male_95[1] < ci_female_95[0] or ci_female_95[1] < ci_male_95[0])

# Print the results
print(f"Confidence Interval for Males: {ci_male_95}")
print(f"Confidence Interval for Females: {ci_female_95}")
print(f"Do the confidence intervals overlap? {'Yes' if overlap else 'No'}")

# Recommendations based on the results
if overlap:
    recommendations = """
    Since the confidence intervals for the average amount spent by males and females overlap, it suggests that there is no significant d
    Walmart can leverage this conclusion by:
    - Implementing uniform marketing strategies for both males and females.
    - Ensuring that products appealing to both genders are equally accessible.
    - Offering similar promotions and discounts to both males and females.
    """
else:
    recommendations = """
    Since the confidence intervals for the average amount spent by males and females do not overlap, it indicates a significant differer
    Walmart can leverage this conclusion by:
    - Developing targeted marketing campaigns tailored to the spending habits of each gender.
    - Customizing product offerings based on the preferences of males and females.
    - Creating gender-specific promotions and discounts to maximize sales.
    """

print(recommendations)
```

```
Confidence Interval for Males: (9416.456199409533, 9446.420252561296)
Confidence Interval for Females: (8725.703804031573, 8775.460945636441)
Do the confidence intervals overlap? No

    Since the confidence intervals for the average amount spent by males and females do not overlap, it indicates a significant diff
    Walmart can leverage this conclusion by:
    - Developing targeted marketing campaigns tailored to the spending habits of each gender.
    - Customizing product offerings based on the preferences of males and females.
    - Creating gender-specific promotions and discounts to maximize sales.
```

b) Report whether the condence intervals for the average amount spent by married and unmarried (computed using all the data) overlap. How can Walmart leverage this conclusion to make changes or improvements?

```
# Check for overlap for  95% Confidence Interval
overlap = not (ci_married[1] < ci_single[0] or ci_single[1] < ci_married[0])

# Print the results
print(f"Confidence Interval for Married: {ci_married}")
print(f"Confidence Interval for Single: {ci_single}")
print(f"Do the confidence intervals overlap? {'Yes' if overlap else 'No'}")

# Recommendations based on the results
if overlap:
    recommendations = """
    Since the confidence intervals for the average amount spent by married and unmarried individuals overlap, it suggests that there is n
    Walmart can leverage this conclusion by:
    - Implementing uniform marketing strategies for both married and unmarried individuals.
    - Ensuring that products appealing to both groups are equally accessible.
    - Offering similar promotions and discounts to both married and unmarried individuals.
    """
else:
    recommendations = """
    Since the confidence intervals for the average amount spent by married and unmarried individuals do not overlap, it indicates a signi
    Walmart can leverage this conclusion by:
    - Developing targeted marketing campaigns tailored to the spending habits of each group.
    - Customizing product offerings based on the preferences of married and unmarried individuals.
    - Creating group-specific promotions and discounts to maximize sales.
    """

print(recommendations)
```

```
Confidence Interval for Married: (9238.954866147504, 9277.339209182182)
Confidence Interval for Single: (9251.67890117238, 9283.99260070885)
Do the confidence intervals overlap? Yes

    Since the confidence intervals for the average amount spent by married and unmarried individuals overlap, it suggests that there
    Walmart can leverage this conclusion by:
    - Implementing uniform marketing strategies for both married and unmarried individuals.
    - Ensuring that products appealing to both groups are equally accessible.
    - Offering similar promotions and discounts to both married and unmarried individuals.
```

c) Report whether the condence intervals for the average amount spent by different age groups (computed using all the data) overlap. How can Walmart leverage this conclusion to make changes or improvements?

```
# Print the confidence intervals for each age group for 95% Confidence Interval
for age, ci in ci_age_groups_sorted.items():
    print(f"95% Confidence Interval for Age {age} (entire dataset): {ci}")

# Check for overlap between confidence intervals of different age groups
overlap_results = {}
age_group_keys = list(ci_age_groups_sorted.keys())

for i in range(len(age_group_keys)):
    for j in range(i + 1, len(age_group_keys)):
        group1 = age_group_keys[i]
        group2 = age_group_keys[j]
        ci1 = ci_age_groups_sorted[group1]
        ci2 = ci_age_groups_sorted[group2]
        overlap = not (ci1[1] < ci2[0] or ci2[1] < ci1[0])
        overlap_results[f"{group1} and {group2}"] = overlap

print("\nOverlap Results:")
for groups, overlap in overlap_results.items():
    print(f"Do the confidence intervals for {groups} overlap? {'Yes' if overlap else 'No'}")

# Recommendations based on the results
recommendations = """
Based on the overlap results of the confidence intervals for different age groups, Walmart can leverage the conclusions as follows:

If the intervals overlap:
- Implement uniform marketing strategies across overlapping age groups.
- Ensure that products appealing to overlapping age groups are equally accessible.
- Offer similar promotions and discounts to overlapping age groups.

If the intervals do not overlap:
- Develop targeted marketing campaigns tailored to the spending habits of each non-overlapping age group.
```

```
- Customize product offerings based on the preferences of each non-overlapping age group.
- Create age-specific promotions and discounts to maximize sales.
"""

print(recommendations)
```

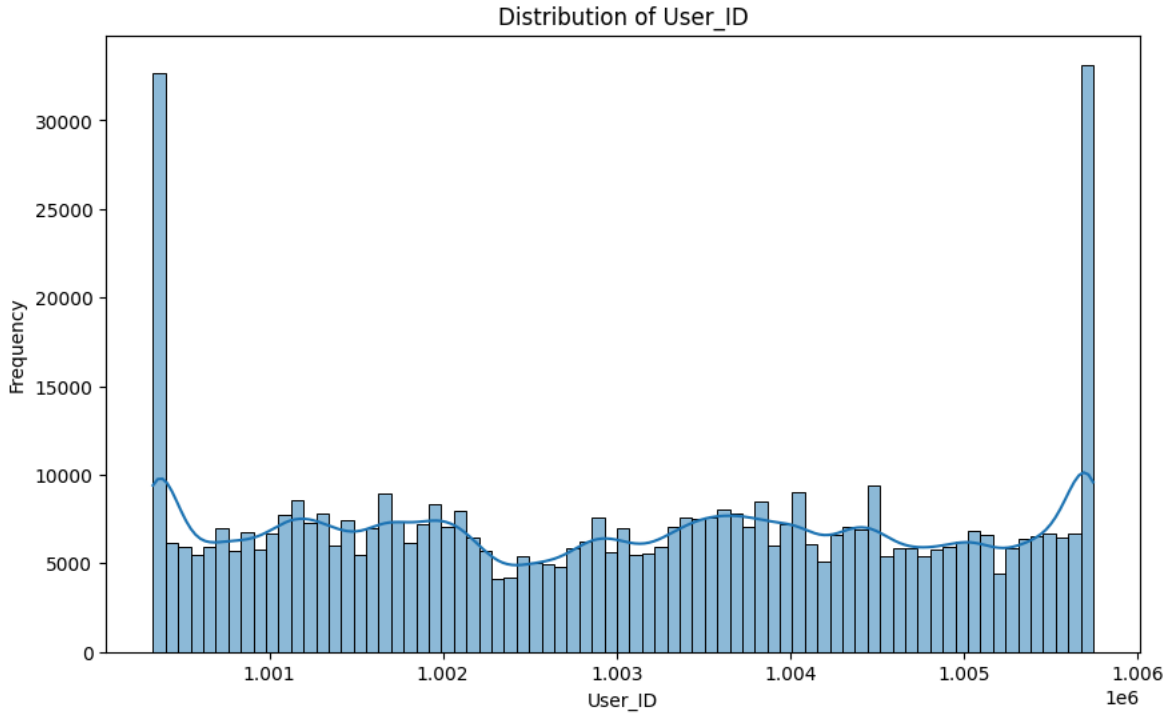
```
95% Confidence Interval for Age 0-17 (entire dataset): (8891.448877914398, 9044.76479876369)
95% Confidence Interval for Age 18-25 (entire dataset): (9163.107699525277, 9221.41869294206)
95% Confidence Interval for Age 26-35 (entire dataset): (9230.727633408536, 9271.29145667388)
95% Confidence Interval for Age 36-45 (entire dataset): (9295.31844785824, 9351.887640335608)
95% Confidence Interval for Age 46-50 (entire dataset): (9164.05274011362, 9254.950564777417)
95% Confidence Interval for Age 51-55 (entire dataset): (9452.56496248646, 9548.984124772145)
95% Confidence Interval for Age 55+ (entire dataset): (9272.091118999004, 9398.405954698703)

Overlap Results:
Do the confidence intervals for 0-17 and 18-25 overlap? No
Do the confidence intervals for 0-17 and 26-35 overlap? No
Do the confidence intervals for 0-17 and 36-45 overlap? No
Do the confidence intervals for 0-17 and 46-50 overlap? No
Do the confidence intervals for 0-17 and 51-55 overlap? No
Do the confidence intervals for 0-17 and 55+ overlap? No
Do the confidence intervals for 18-25 and 26-35 overlap? No
Do the confidence intervals for 18-25 and 36-45 overlap? No
Do the confidence intervals for 18-25 and 46-50 overlap? Yes
Do the confidence intervals for 18-25 and 51-55 overlap? No
Do the confidence intervals for 18-25 and 55+ overlap? No
Do the confidence intervals for 26-35 and 36-45 overlap? No
Do the confidence intervals for 26-35 and 46-50 overlap? Yes
Do the confidence intervals for 26-35 and 51-55 overlap? No
Do the confidence intervals for 26-35 and 55+ overlap? No
Do the confidence intervals for 36-45 and 46-50 overlap? No
Do the confidence intervals for 36-45 and 51-55 overlap? No
Do the confidence intervals for 36-45 and 55+ overlap? Yes
Do the confidence intervals for 46-50 and 51-55 overlap? No
Do the confidence intervals for 46-50 and 55+ overlap? No
Do the confidence intervals for 51-55 and 55+ overlap? No

Based on the overlap results of the confidence intervals for different age groups, Walmart can leverage the conclusions as follows:

If the intervals overlap:
- Implement uniform marketing strategies across overlapping age groups.
- Ensure that products appealing to overlapping age groups are equally accessible.
- Offer similar promotions and discounts to overlapping age groups.

If the intervals do not overlap:
- Develop targeted marketing campaigns tailored to the spending habits of each non-overlapping age group.
- Customize product offerings based on the preferences of each non-overlapping age group.
- Create age-specific promotions and discounts to maximize sales.
```

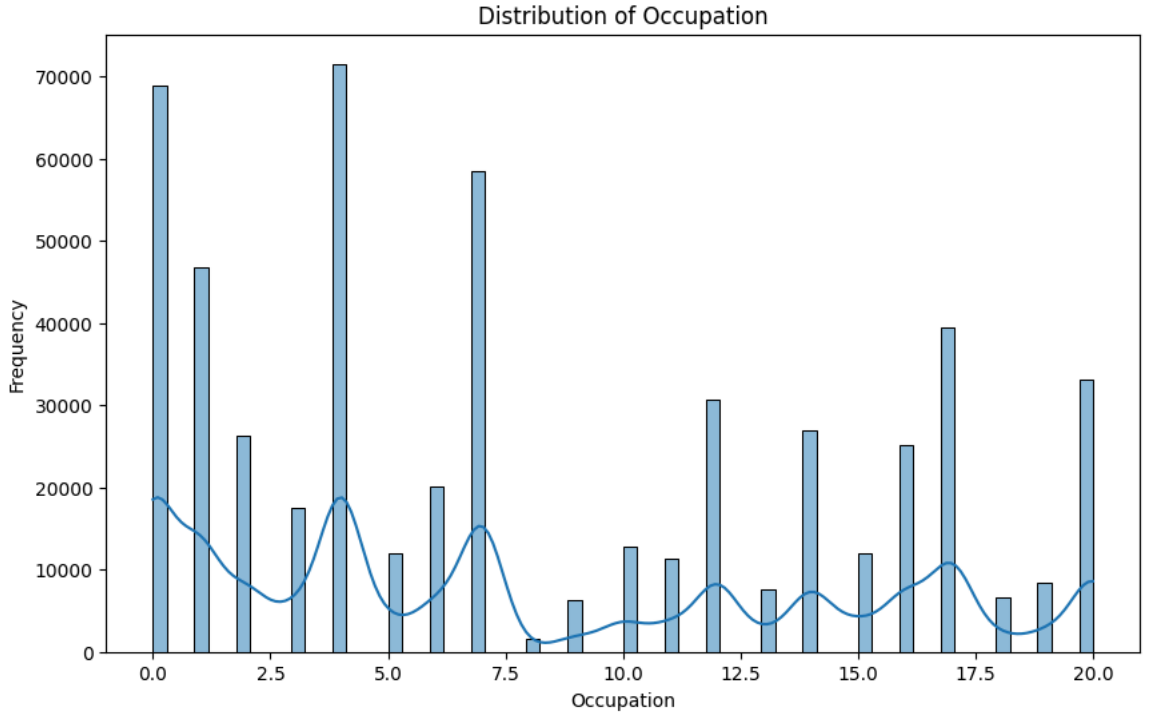
Final Insights- Illustrate the Insights Based on Exploration and CLT

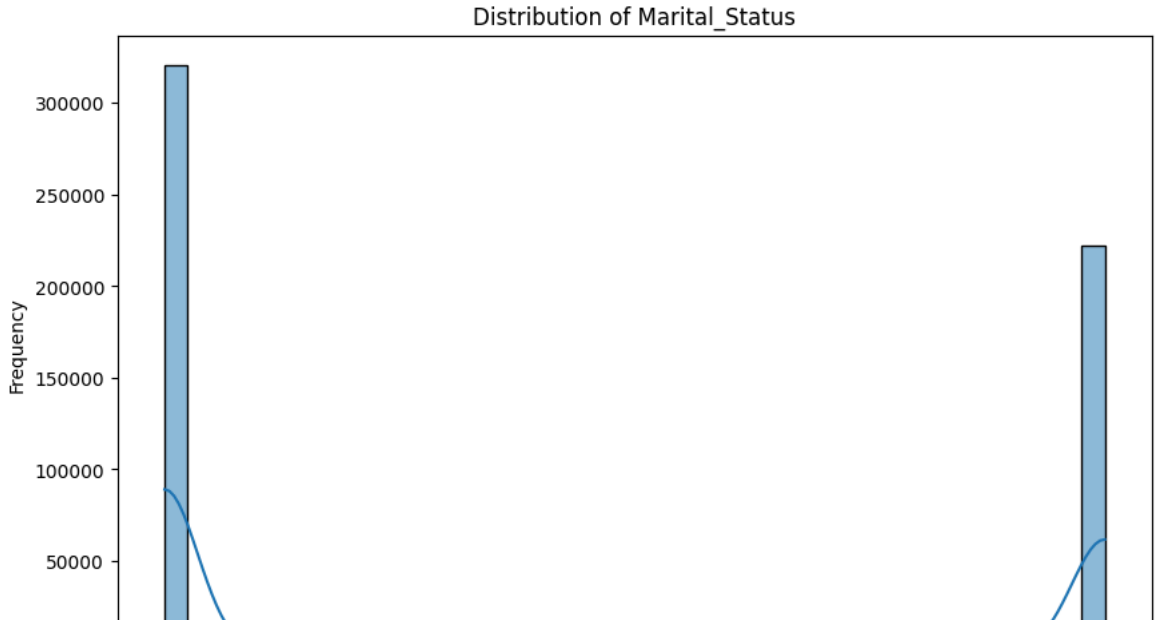
1. Comments on the Distribution of the Variables and Relationship Between Them

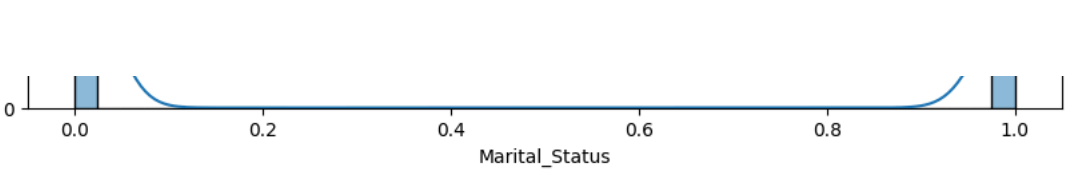
```
# Univariate Analysis
def univariate_analysis(df):
    for column in df.select_dtypes(include=['float64', 'int64']).columns:
        plt.figure(figsize=(10, 6))
        sns.histplot(df[column], kde=True)
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.show()

univariate_analysis(df)

# Bivariate Analysis
def bivariate_analysis(df):
    numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
    for i in range(len(numeric_columns)):
        for j in range(i+1, len(numeric_columns)):
            plt.figure(figsize=(10, 6))
            sns.scatterplot(data=df, x=numeric_columns[i], y=numeric_columns[j])
            plt.title(f'{numeric_columns[i]} vs {numeric_columns[j]}')
            plt.xlabel(numeric_columns[i])
            plt.ylabel(numeric_columns[j])
            plt.show()

bivariate_analysis(df)
```

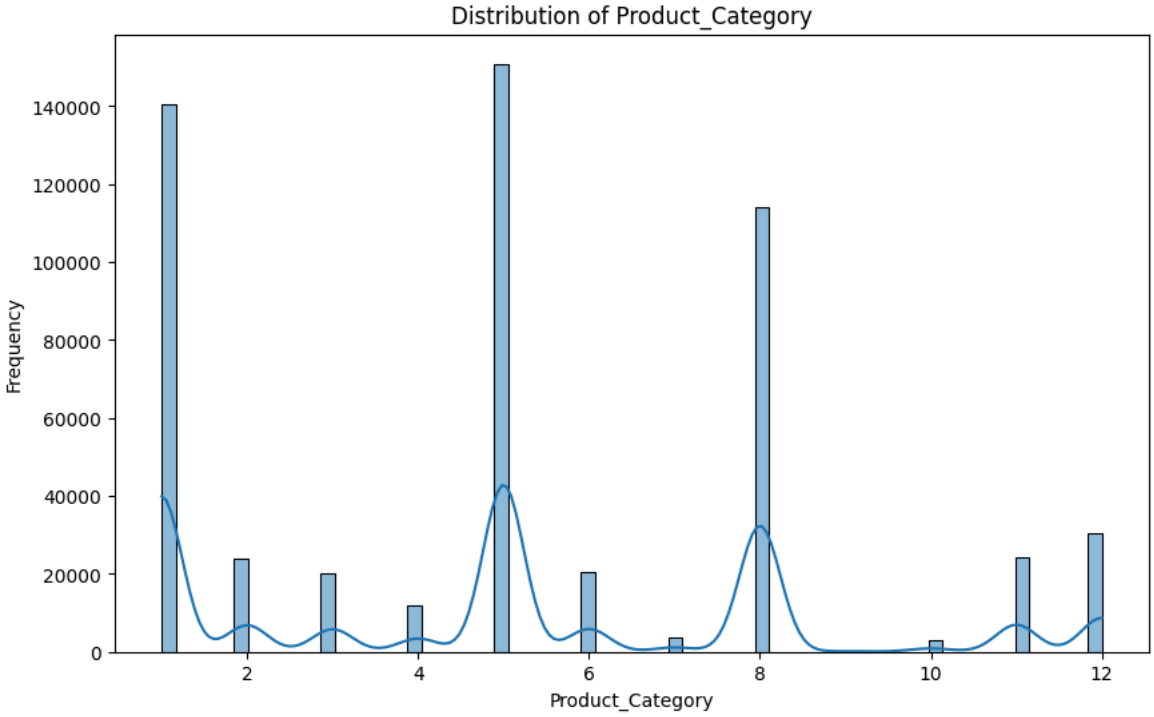
[4]

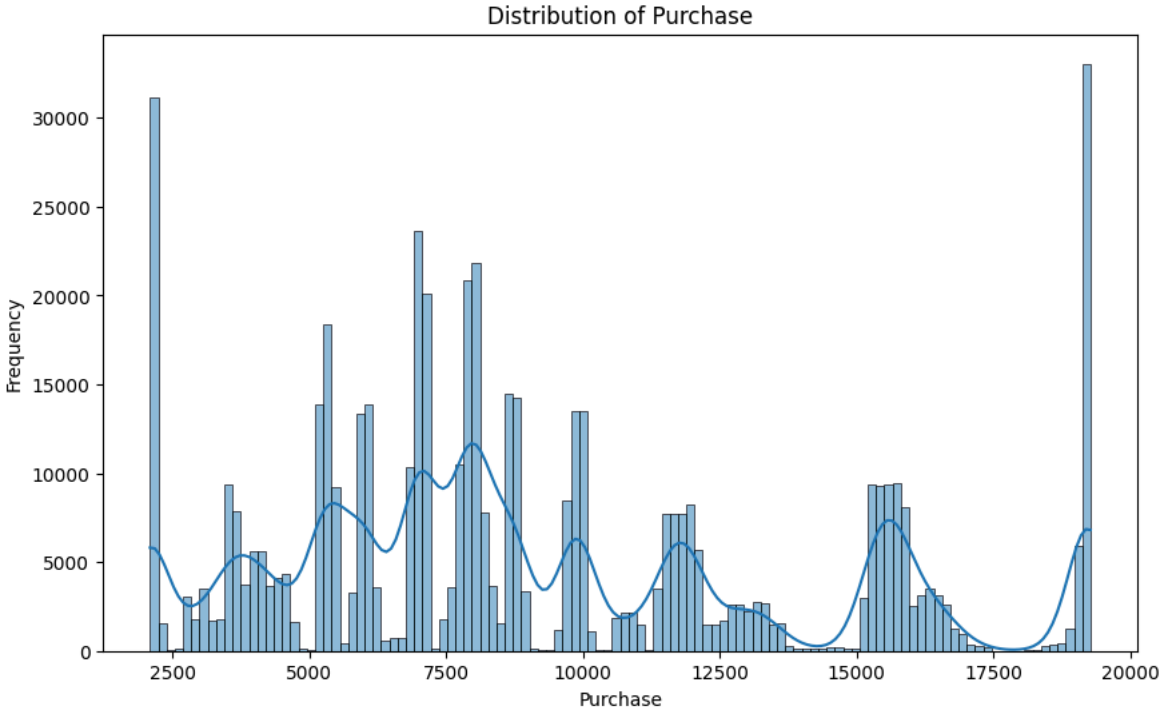

Distribution of User_ID
Frequency
User_ID
1e6
0
5000
10000
15000
20000
25000
30000
1.001
1.002
1.003
1.004
1.005
1.006

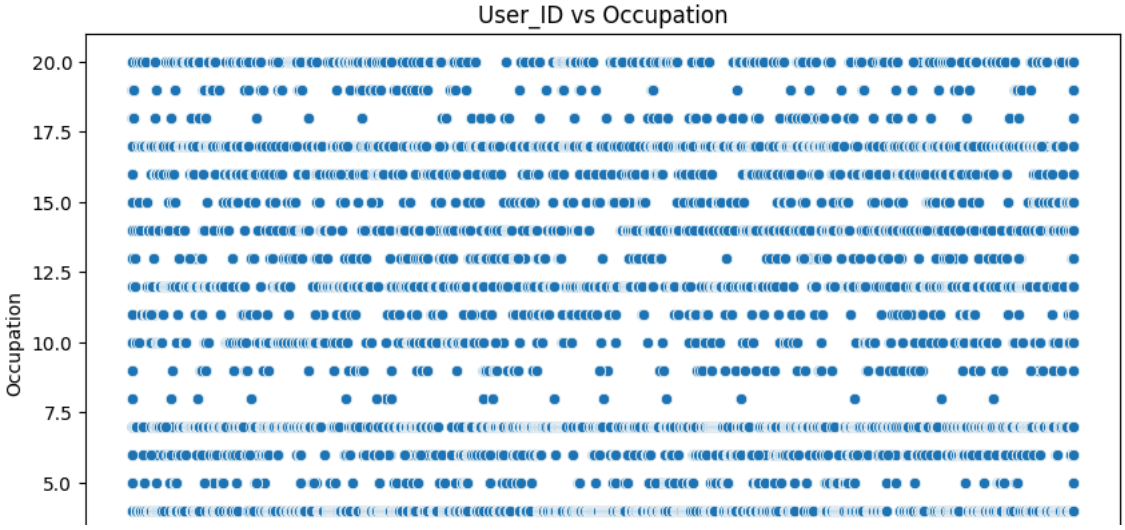

Distribution of Occupation
Frequency
Occupation
0
10000
20000
30000
40000
50000
60000
70000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

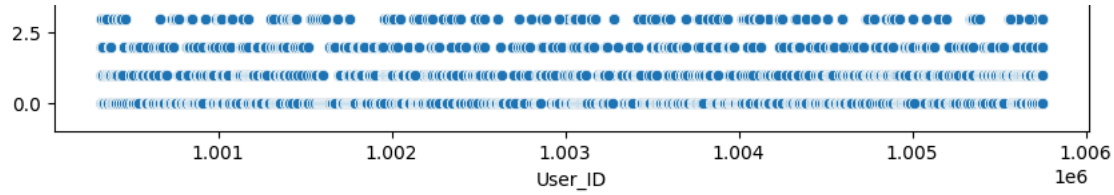

Distribution of Marital_Status
Frequency
50000
100000
150000
200000
250000
300000

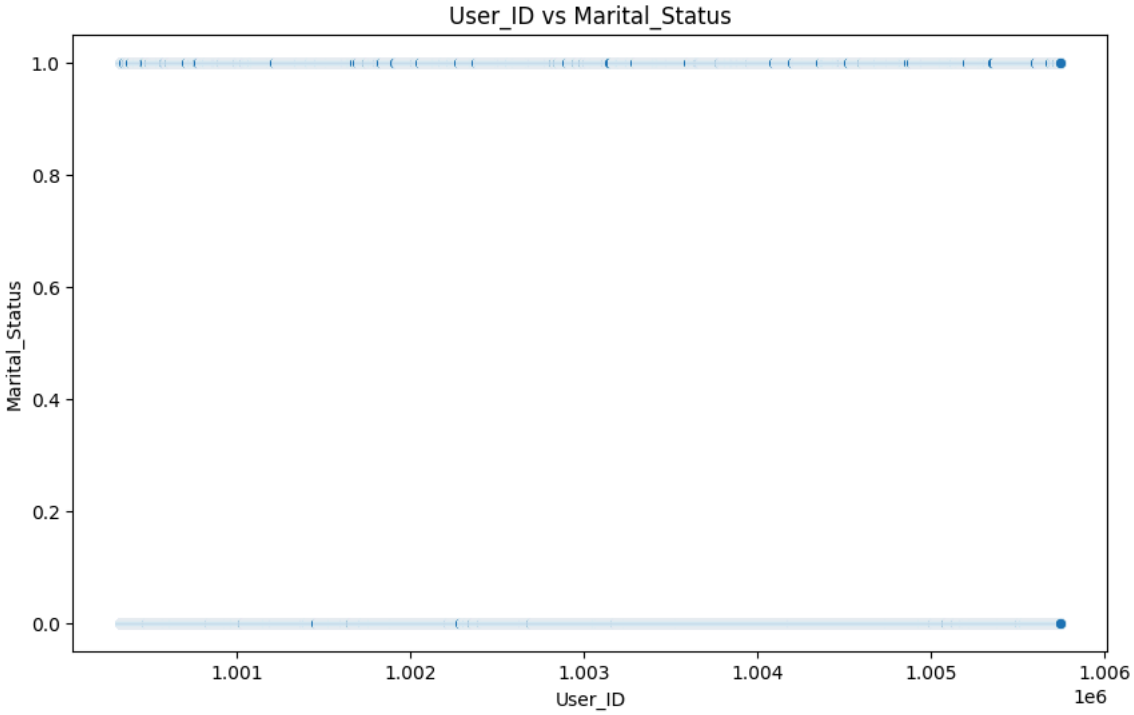

0
0.0
0.2
0.4
0.6
0.8
1.0
Marital_Status

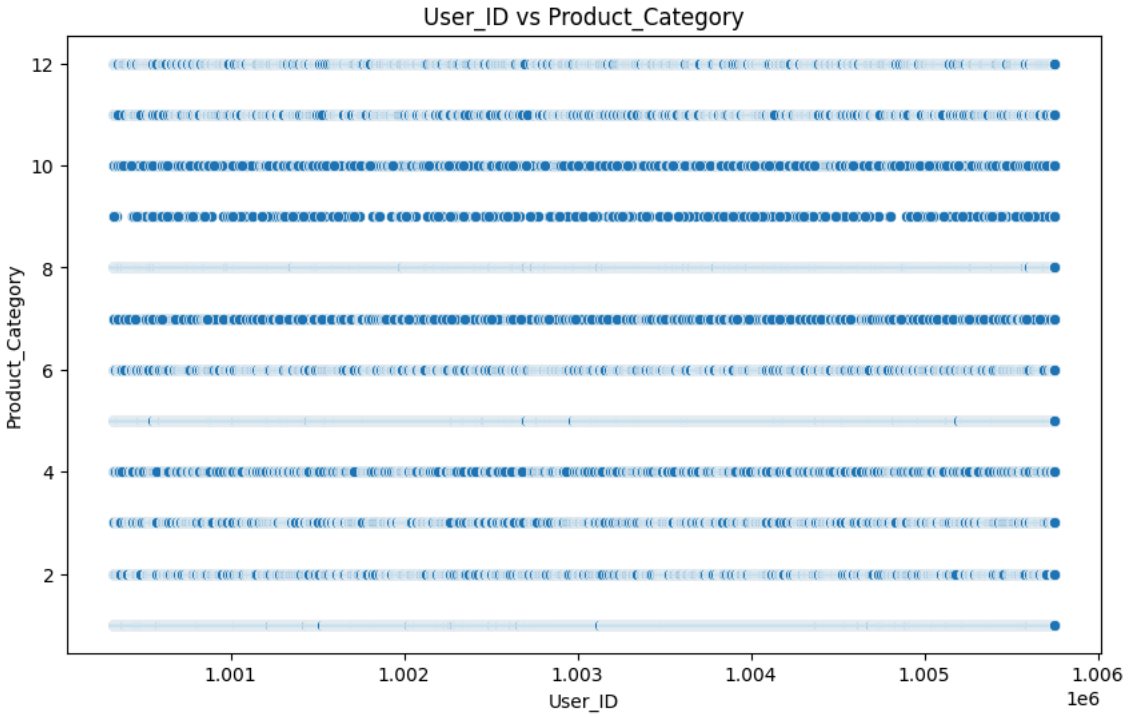

Distribution of Product_Category
140000
120000
100000
80000
60000
40000
20000
0
Frequency
2
4
6
8
10
12
Product_Category

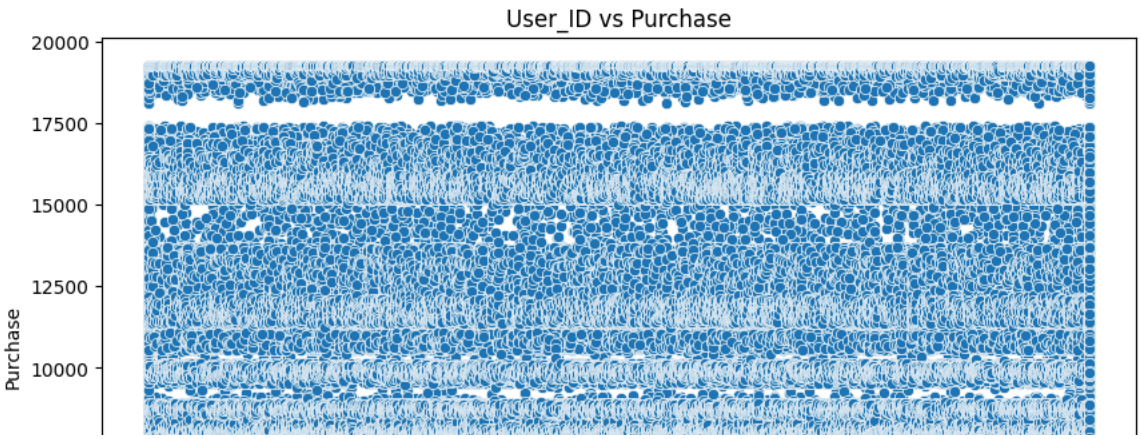

Distribution of Purchase
30000
25000
20000
15000
10000
5000
0
Frequency
2500
5000
7500
10000
12500
15000
17500
20000
Purchase

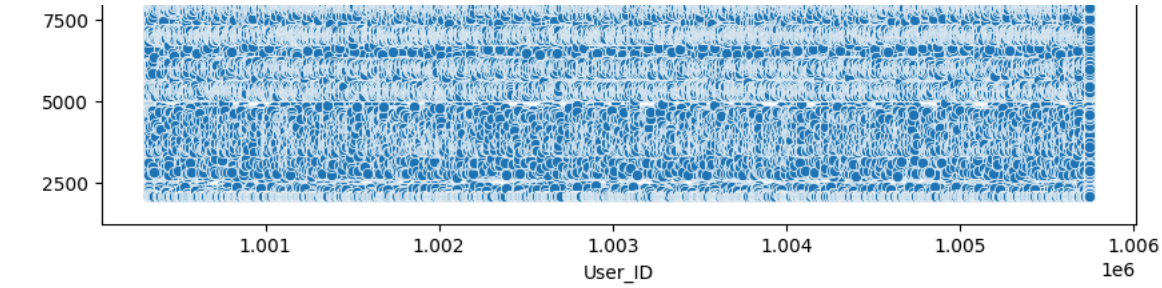

User_ID vs Occupation
20.0
17.5
15.0
12.5
10.0
7.5
5.0
Occupation

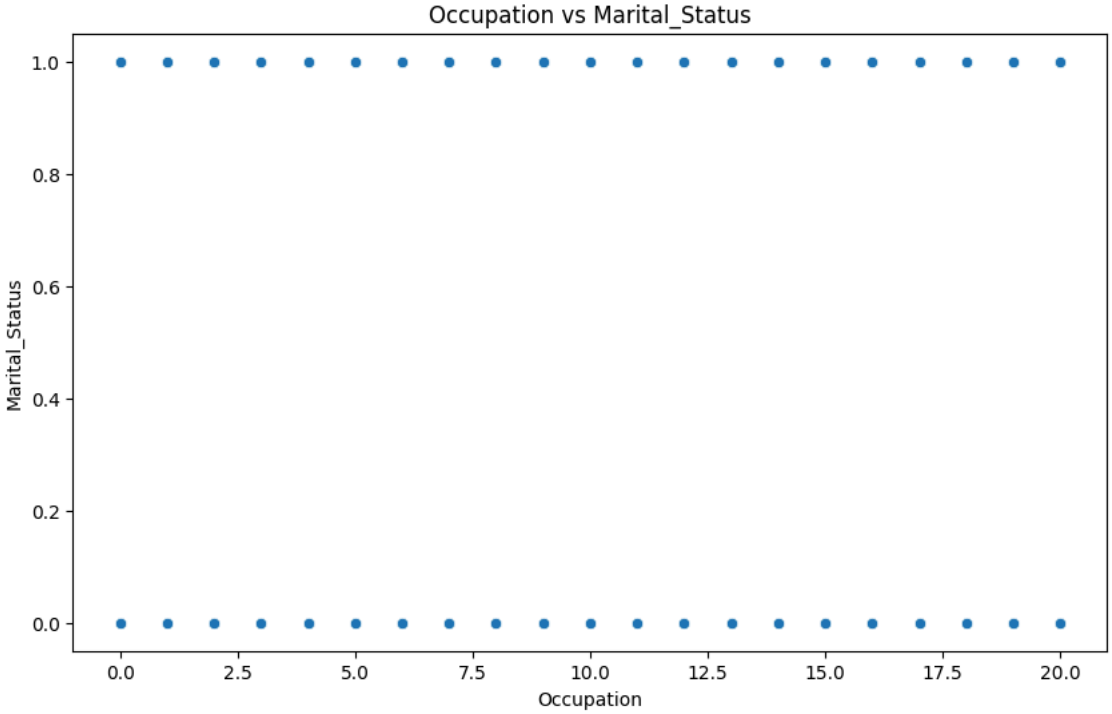

2.5
0.0
1.001
1.002
1.003
1.004
1.005
1.006
User_ID
1e6

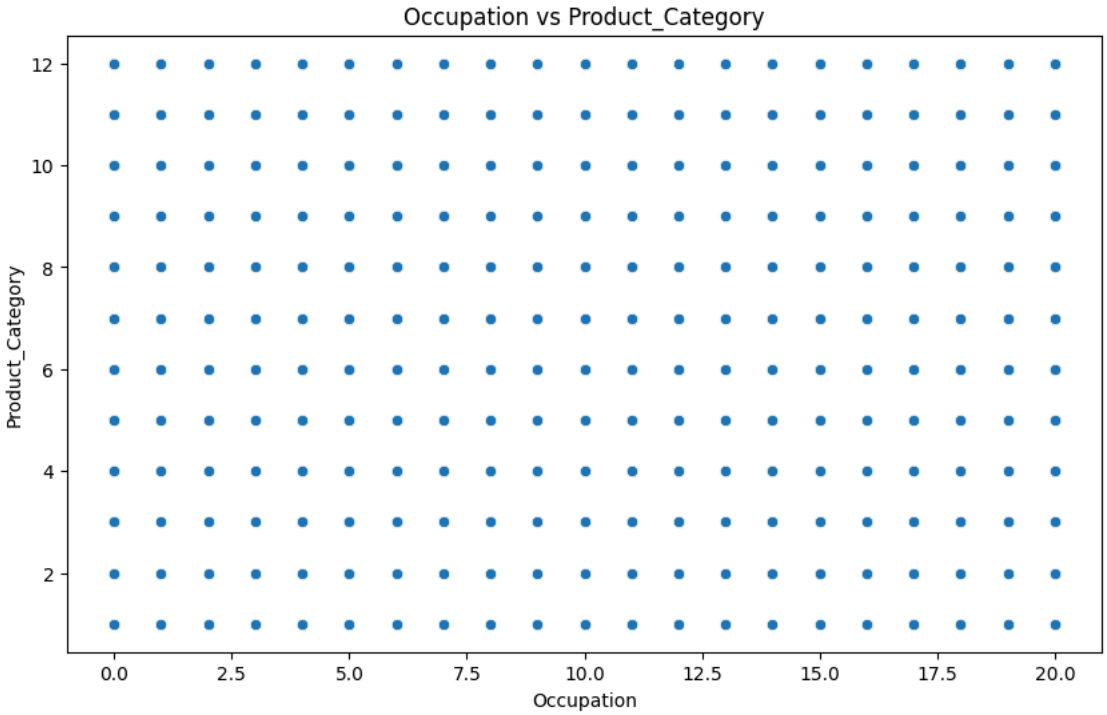

User_ID vs Marital_Status
Marital_Status
1.0
0.8
0.6
0.4
0.2
0.0
1.001
1.002
1.003
1.004
1.005
1.006
User_ID
1e6

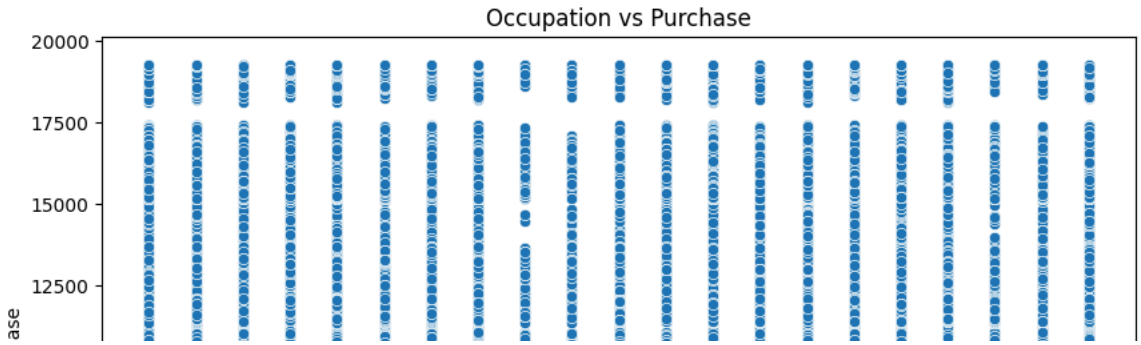

User_ID vs Product_Category
Product_Category
12
10
8
6
4
2
1.001
1.002
1.003
1.004
1.005
1.006
User_ID
1e6

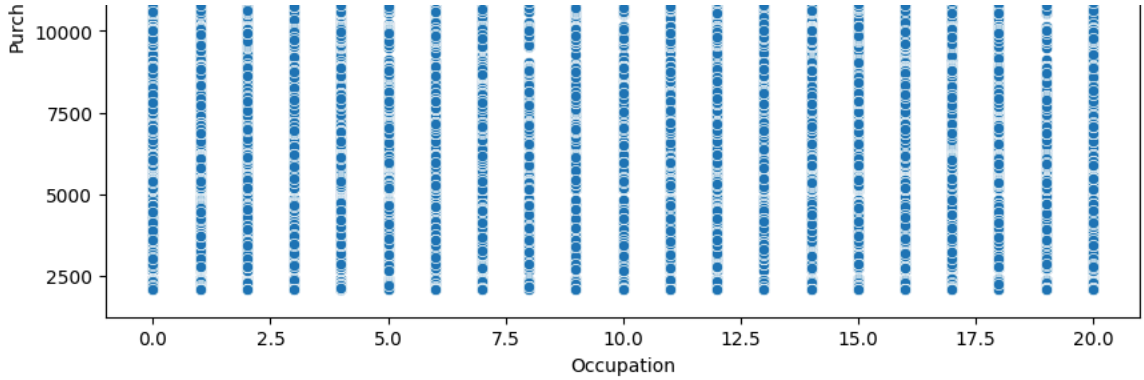

User_ID vs Purchase
Purchase
20000
17500
15000
12500
10000

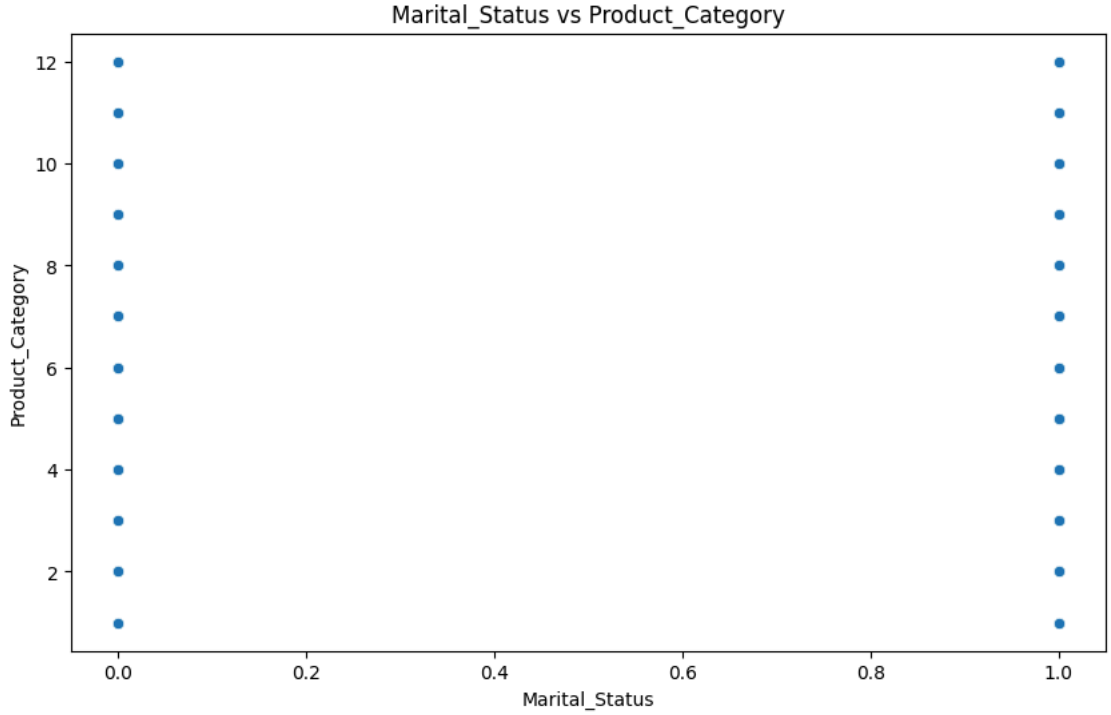

7500
5000
2500
1.001
1.002
1.003
1.004
1.005
1.006
User_ID
1e6

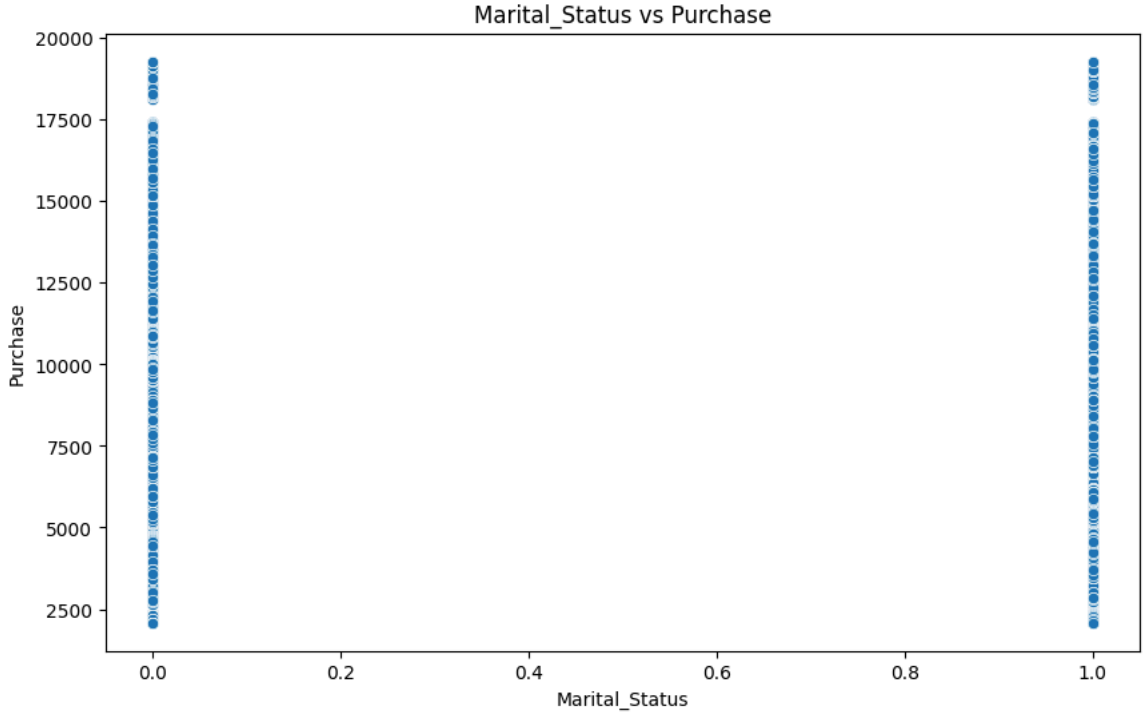

Occupation vs Marital_Status
Marital_Status
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Occupation

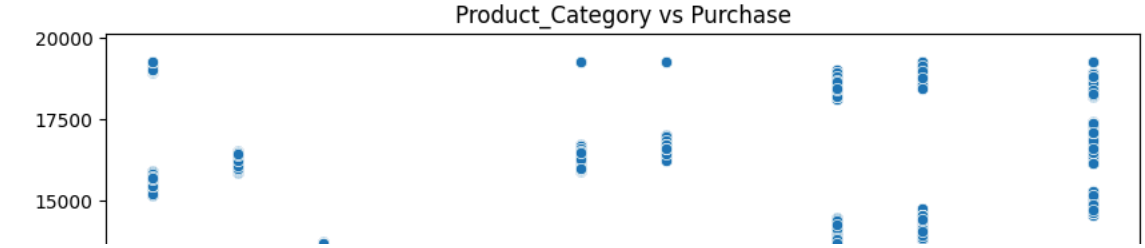

Occupation vs Product_Category
Product_Category
12
10
8
6
4
2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Occupation

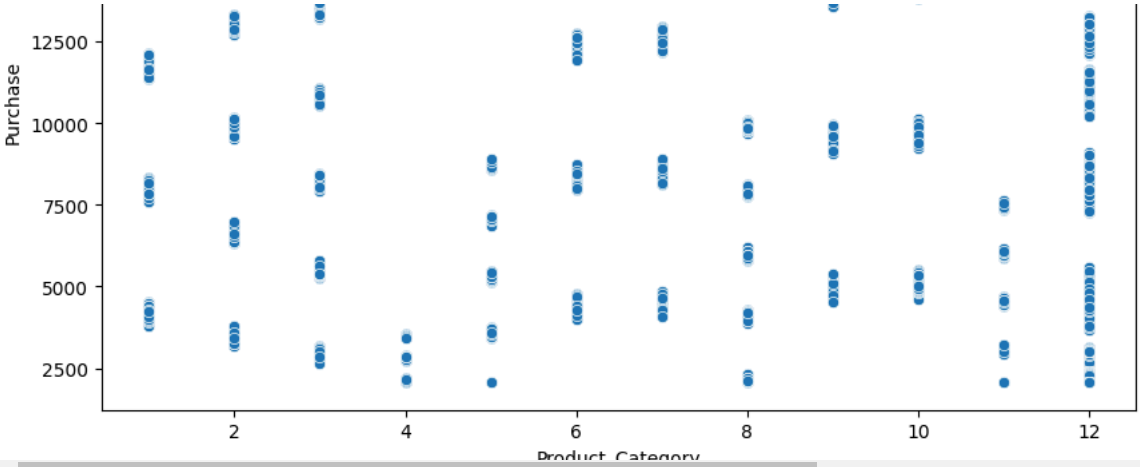

Occupation vs Purchase
20000
17500
15000
12500


Purch
10000
7500
5000
2500
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Occupation


Marital_Status vs Product_Category
Product_Category
12
10
8
6
4
2
0.0
0.2
0.4
0.6
0.8
1.0
Marital_Status


Marital_Status vs Purchase
Purchase
20000
17500
15000
12500
10000
7500
5000
2500
0.0
0.2
0.4
0.6
0.8
1.0
Marital_Status


Product_Category vs Purchase
20000
17500
15000


Purchase
12500
10000
7500
5000
2500
2
4
6
8
10
12
Product_Category

2. Comments for each univariate and bivariate plots

```
# Univariate Analysis Comments
def univariate_comments(df):
    comments = {}
    for column in df.select_dtypes(include=['float64', 'int64']).columns:
        comments[column] = f"The distribution of {column} shows that the data is {'normally distributed' if df[column].skew() < 0.5 else
    return comments

uni_comments = univariate_comments(df)
print("Univariate Analysis Comments:")
for column, comment in uni_comments.items():
    print(f"{column}: {comment}")

# Bivariate Analysis Comments
def bivariate_comments(df):
    comments = {}
    numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
    for i in range(len(numeric_columns)):
        for j in range(i+1, len(numeric_columns)):
            correlation = df[numeric_columns[i]].corr(df[numeric_columns[j]])
            comments[f'{numeric_columns[i]} vs {numeric_columns[j]}'] = f"The relationship between {numeric_columns[i]} and {numeric_co
    return comments

bi_comments = bivariate_comments(df)
print("\nBivariate Analysis Comments:")
for columns, comment in bi_comments.items():
    print(f"{columns}: {comment}")
```

```
Univariate Analysis Comments:
User_ID: The distribution of User_ID shows that the data is normally distributed.
Occupation: The distribution of Occupation shows that the data is normally distributed.
Marital_Status: The distribution of Marital_Status shows that the data is normally distributed.
Product_Category: The distribution of Product_Category shows that the data is normally distributed.
Purchase: The distribution of Purchase shows that the data is skewed.

Bivariate Analysis Comments:
User_ID vs Occupation: The relationship between User_ID and Occupation shows a correlation of -0.02.
User_ID vs Marital_Status: The relationship between User_ID and Marital_Status shows a correlation of 0.02.
User_ID vs Product_Category: The relationship between User_ID and Product_Category shows a correlation of 0.00.
User_ID vs Purchase: The relationship between User_ID and Purchase shows a correlation of 0.00.
Occupation vs Marital_Status: The relationship between Occupation and Marital_Status shows a correlation of 0.02.
Occupation vs Product_Category: The relationship between Occupation and Product_Category shows a correlation of -0.01.
Occupation vs Purchase: The relationship between Occupation and Purchase shows a correlation of 0.02.
Marital_Status vs Product_Category: The relationship between Marital_Status and Product_Category shows a correlation of 0.02.
Marital_Status vs Purchase: The relationship between Marital_Status and Purchase shows a correlation of -0.00.
Product_Category vs Purchase: The relationship between Product_Category and Purchase shows a correlation of -0.40.
```

Univariate Analysis Insights:

1) User_ID:

- The distribution of User_ID shows that the data is normally distributed.

2) Occupation:

- The distribution of Occupation shows that the data is normally distributed.

3) Marital_Status:

- The distribution of Marital_Status shows that the data is normally distributed.

4) Product_Category:

- The distribution of Product_Category shows that the data is normally distributed.

5) Purchase:

- The distribution of Purchase shows that the data is skewed.

Bivariate Analysis Insights:

1) User_ID vs. Occupation:

- The relationship between User_ID and Occupation shows a correlation of -0.02, indicating a very weak negative correlation.

2) User_ID vs. Marital_Status:

- The relationship between User_ID and Marital_Status shows a correlation of 0.02, indicating a very weak positive correlation.

3) User_ID vs. Product_Category:

- The relationship between User_ID and Product_Category shows a correlation of 0.00, indicating no correlation.

4) User_ID vs. Purchase:

- The relationship between User_ID and Purchase shows a correlation of 0.00, indicating no correlation.

5) Occupation vs. Marital_Status:

- The relationship between Occupation and Marital_Status shows a correlation of 0.02, indicating a very weak positive correlation.

6) Occupation vs. Product_Category:

- The relationship between Occupation and Product_Category shows a correlation of -0.01, indicating a very weak negative correlation.

7) Occupation vs. Purchase:

- The relationship between Occupation and Purchase shows a correlation of 0.02, indicating a very weak positive correlation.

8) Marital_Status vs. Product_Category:

- The relationship between Marital_Status and Product_Category shows a correlation of 0.02, indicating a very weak positive correlation.

9) Marital_Status vs. Purchase:

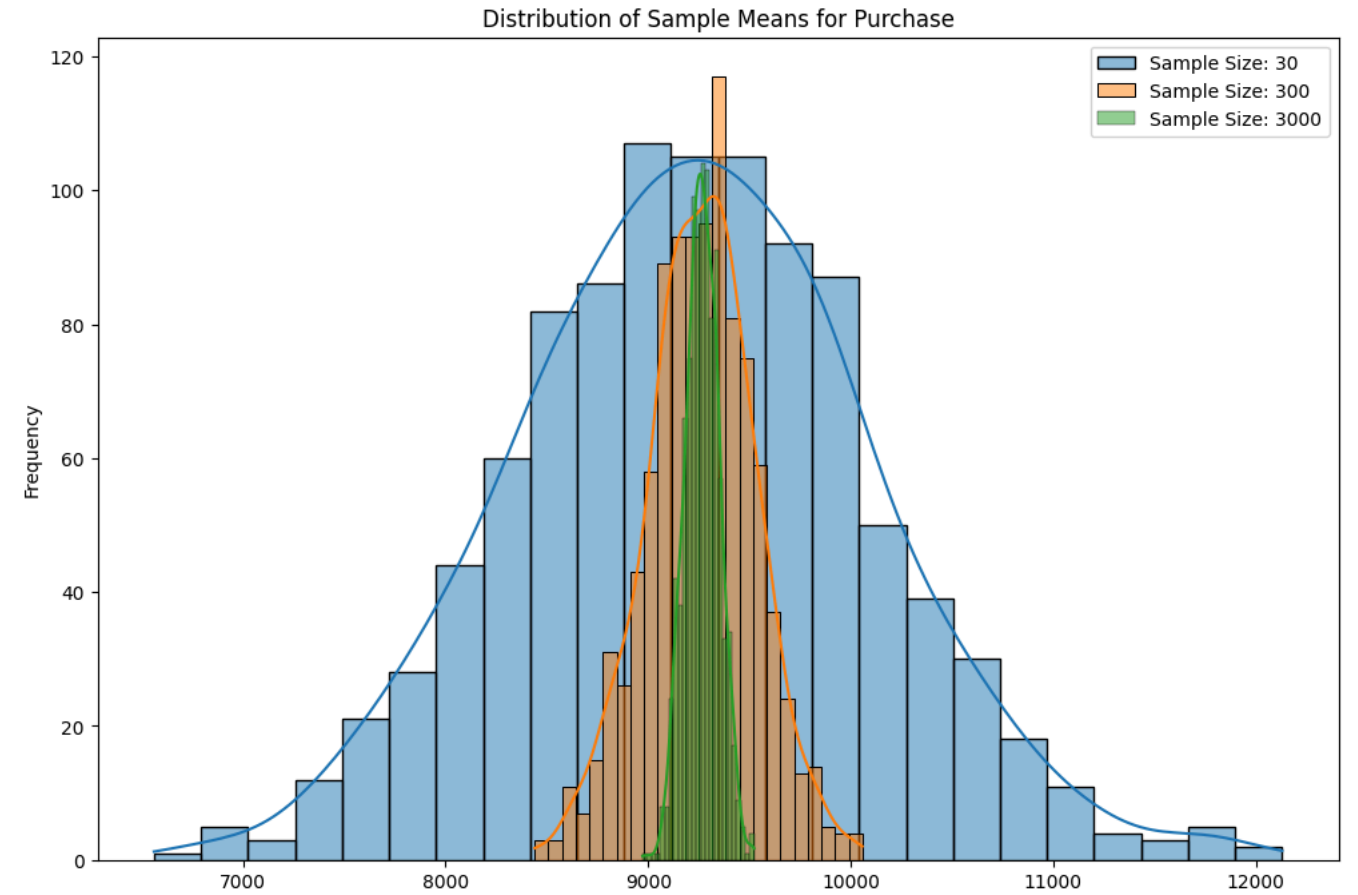
- The relationship between Marital_Status and Purchase shows a correlation of -0.00, indicating no correlation.

10) Product_Category vs. Purchase: The relationship between Product_Category and Purchase shows a correlation of -0.40, indicating a moderate negative correlation.

3. Comments on different variables when generalizing it for Population

```
# Function to calculate sample means and plot their distribution
def plot_sample_means(df, column, sample_sizes, num_samples=1000):
    plt.figure(figsize=(12, 8))

    for sample_size in sample_sizes:
        sample_means = []
        for _ in range(num_samples):
            sample = df[column].sample(sample_size, replace=True)
            sample_means.append(sample.mean())

        sns.histplot(sample_means, kde=True, label=f'Sample Size: {sample_size}')

    plt.title(f'Distribution of Sample Means for {column}')
    plt.xlabel('Sample Mean')
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()

# Define the column to analyze and the sample sizes
column_to_analyze = 'Purchase'
sample_sizes = [30, 300, 3000]

# Plot the distribution of sample means for different sample sizes
plot_sample_means(df, column_to_analyze, sample_sizes)
```

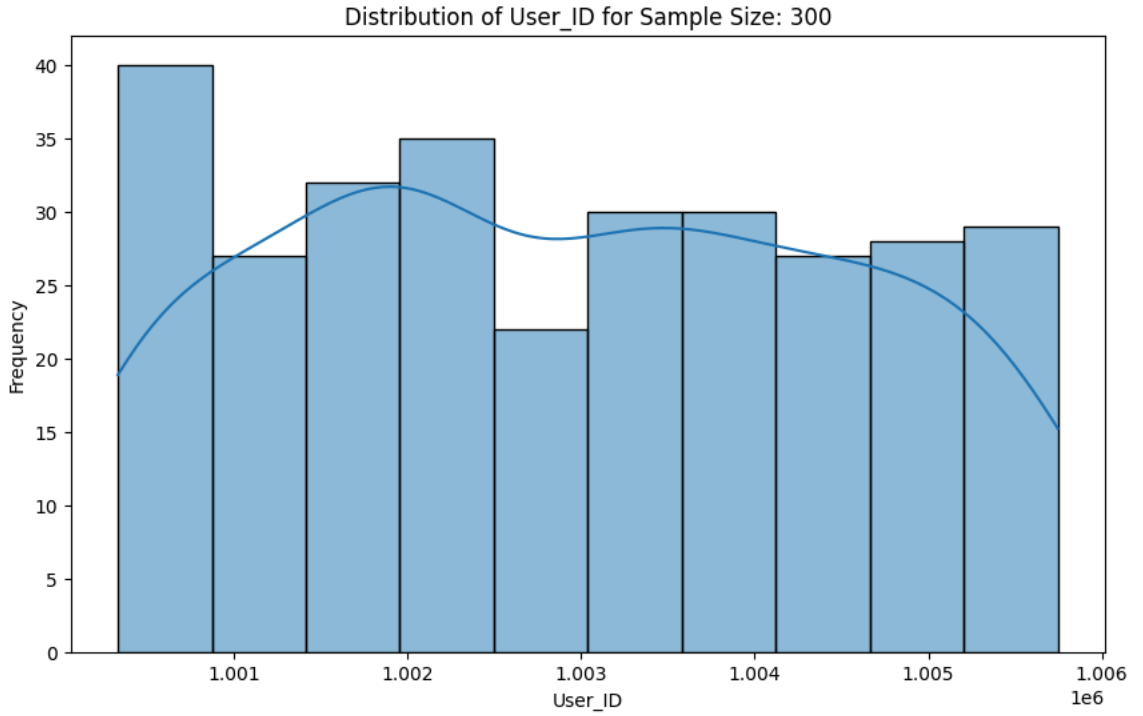
[ ]

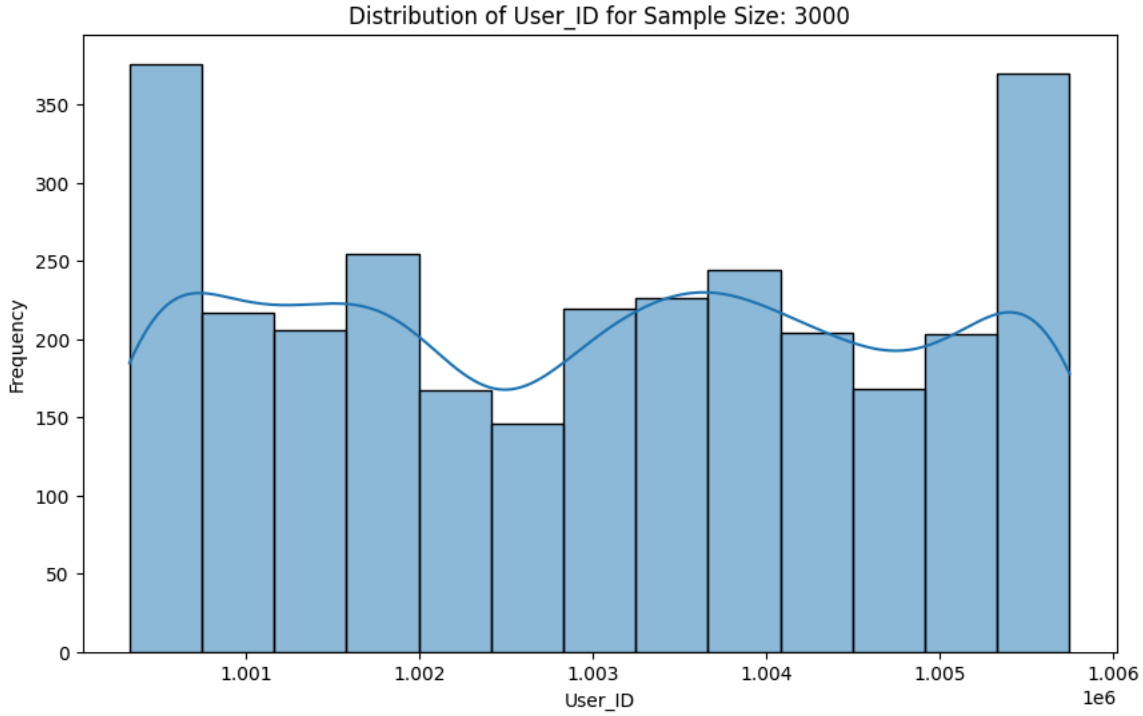

Distribution of Sample Means for Purchase
Sample Size: 30
Sample Size: 300
Sample Size: 3000
Frequency
Sample Mean
0
20
40
60
80
100
120
7000
8000
9000
10000
11000
12000

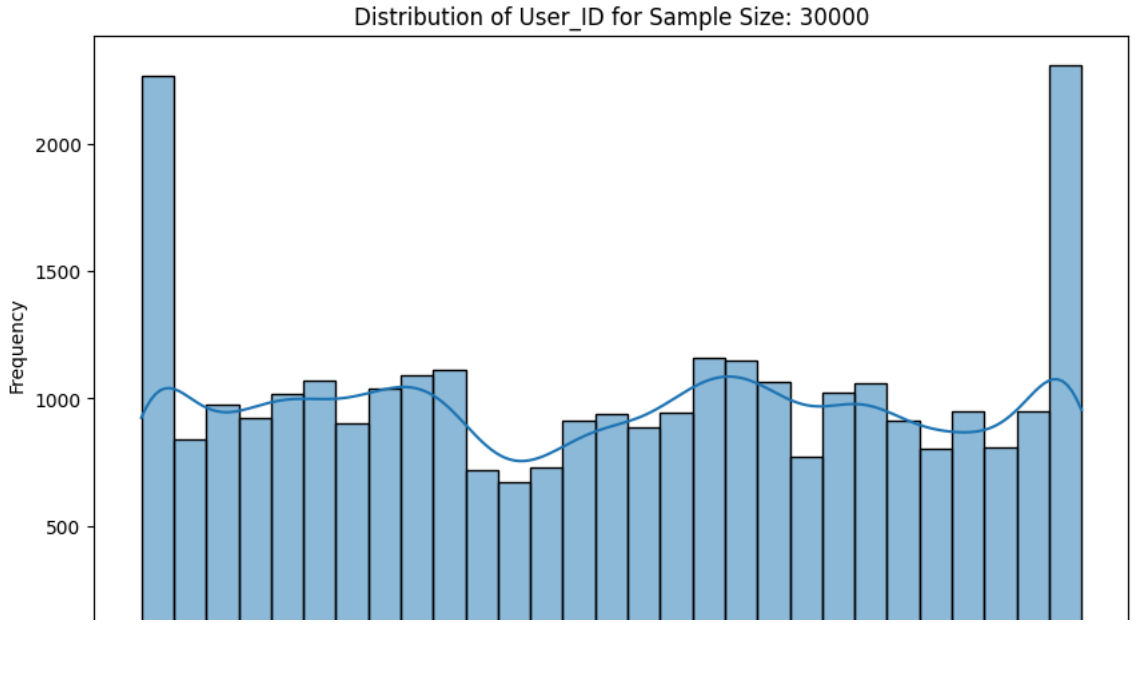
```
# Function to perform univariate analysis for different samples
def univariate_analysis_samples(df, sample_sizes):
    for col in df.select_dtypes(include=['float64', 'int64']).columns:
        for sample_size in sample_sizes:
            sample = df[col].sample(sample_size, replace=True)
            plt.figure(figsize=(10, 6))
            sns.histplot(sample, kde=True)
            plt.title(f'Distribution of {col} for Sample Size: {sample_size}')
            plt.xlabel(col)
            plt.ylabel('Frequency')
            plt.show()

# Define the sample sizes
sample_sizes = [300, 3000, 30000]

# Perform univariate analysis for different samples
univariate_analysis_samples(df, sample_sizes)
```

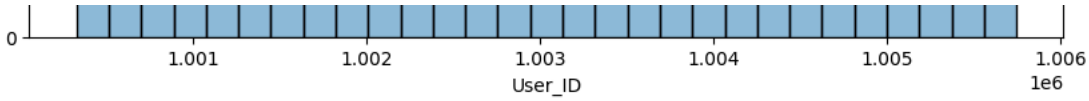
```
# Function to perform bivariate analysis for different samples
def bivariate_analysis_samples(df, column_x, column_y, sample_sizes):
    for sample_size in sample_sizes:
        sample = df.sample(sample_size, replace=True)
        plt.figure(figsize=(10, 6))
        sns.scatterplot(data=sample, x=column_x, y=column_y)
        plt.title(f'{column_x} vs {column_y} for Sample Size: {sample_size}')
        plt.xlabel(column_x)
        plt.ylabel(column_y)
        plt.show()

# Define the columns to analyze and the sample sizes
column_x = 'Purchase'
column_y = 'Age'

# Perform bivariate analysis for different samples
bivariate_analysis_samples(df, column_x, column_y, sample_sizes)
```

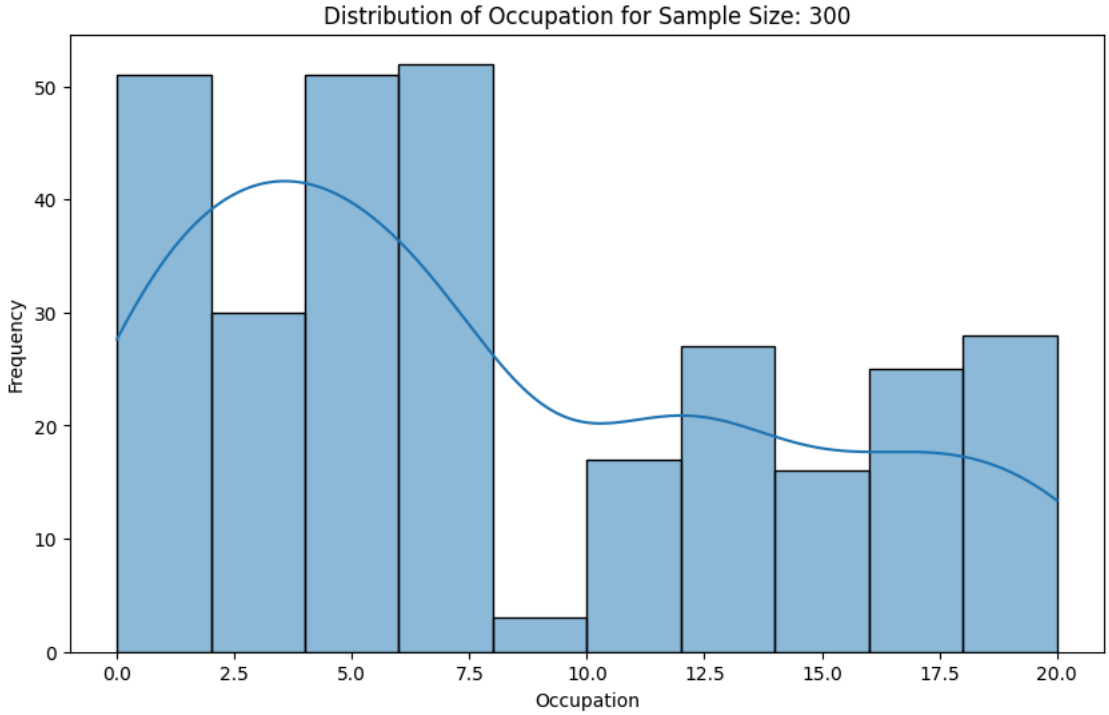
[4]

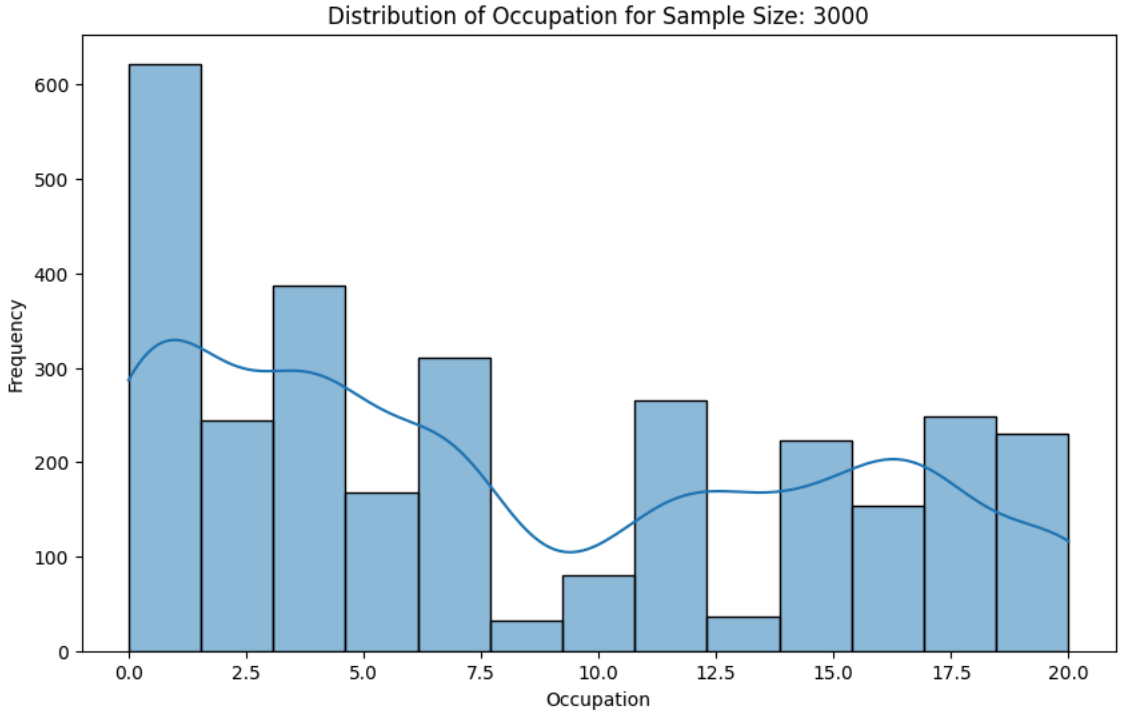

Distribution of User_ID for Sample Size: 300
Frequency
User_ID
1e6
0
5
10
15
20
25
30
35
40
1.001
1.002
1.003
1.004
1.005
1.006

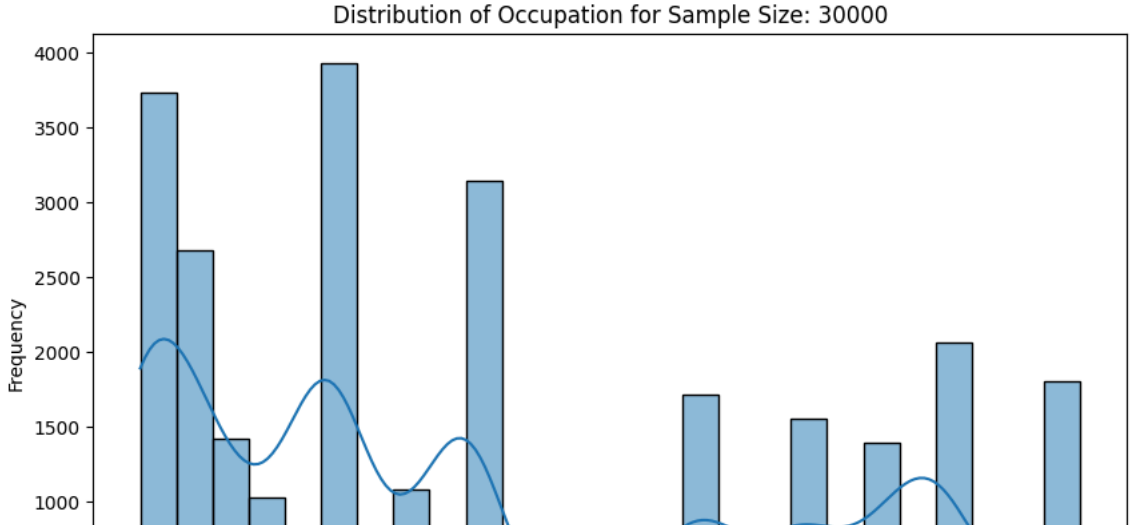

Distribution of User_ID for Sample Size: 3000
Frequency
User_ID
1e6
0
50
100
150
200
250
300
350
1.001
1.002
1.003
1.004
1.005
1.006

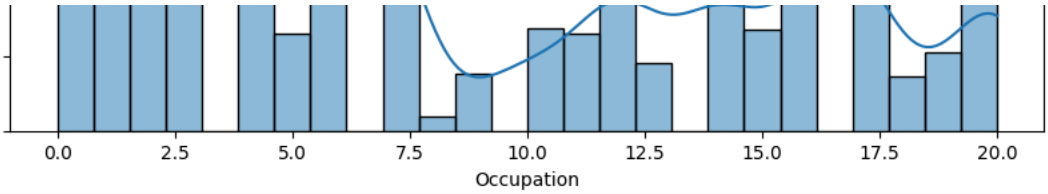

Distribution of User_ID for Sample Size: 30000
Frequency
500
1000
1500
2000

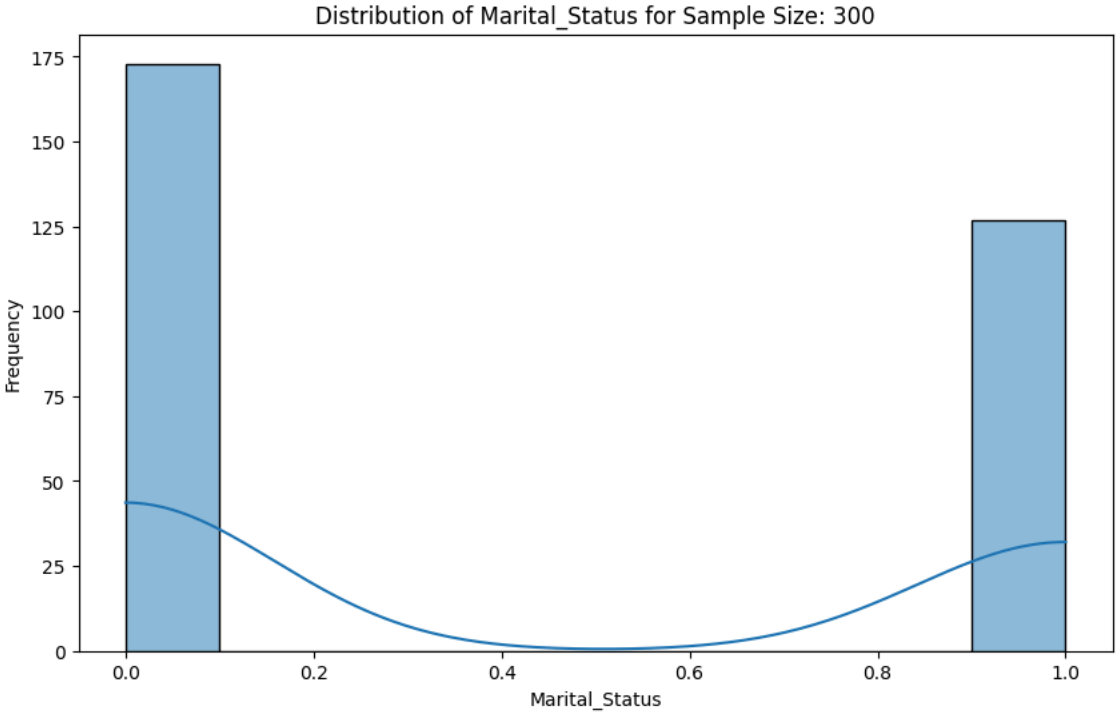

0
1.001
1.002
1.003
1.004
1.005
1.006
User_ID
1e6

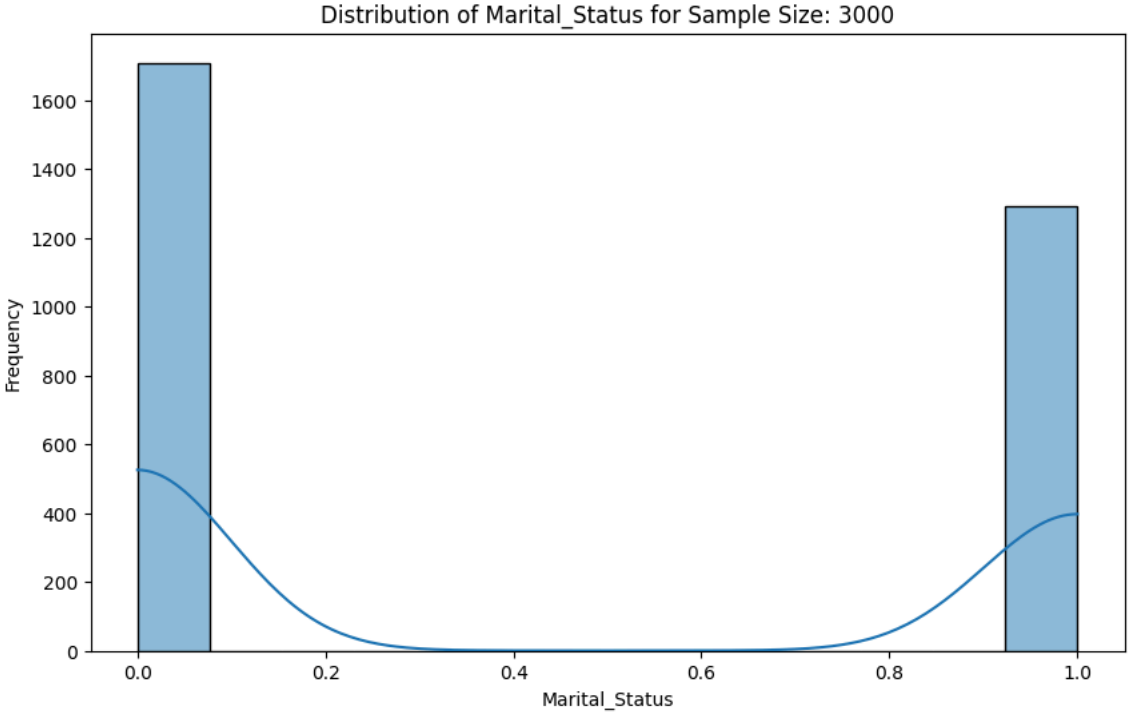

Distribution of Occupation for Sample Size: 300
Frequency
Occupation
0
10
20
30
40
50
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

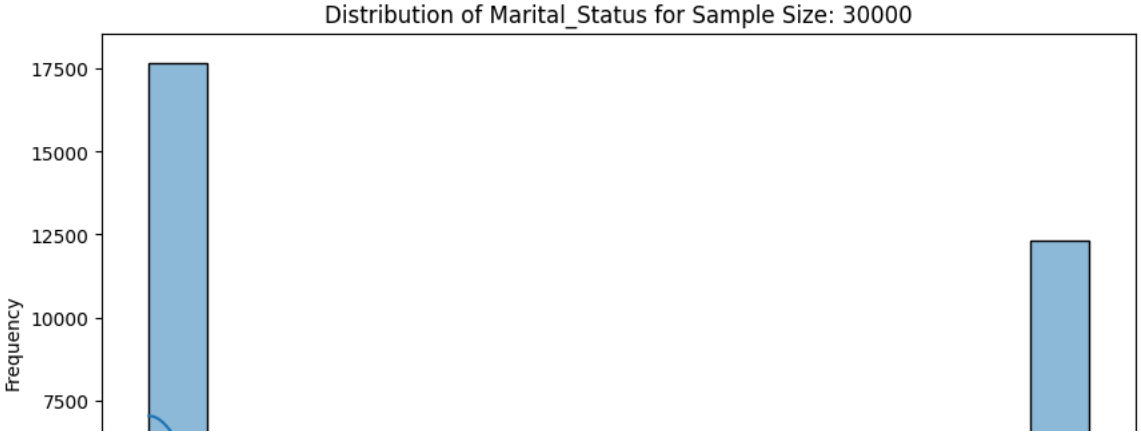

Distribution of Occupation for Sample Size: 3000
Frequency
Occupation
0
100
200
300
400
500
600
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

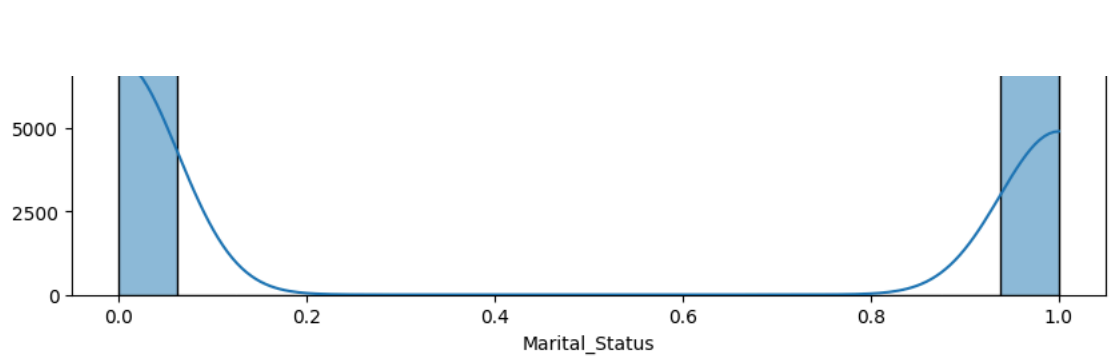

Distribution of Occupation for Sample Size: 30000
Frequency
1000
1500
2000
2500
3000
3500
4000

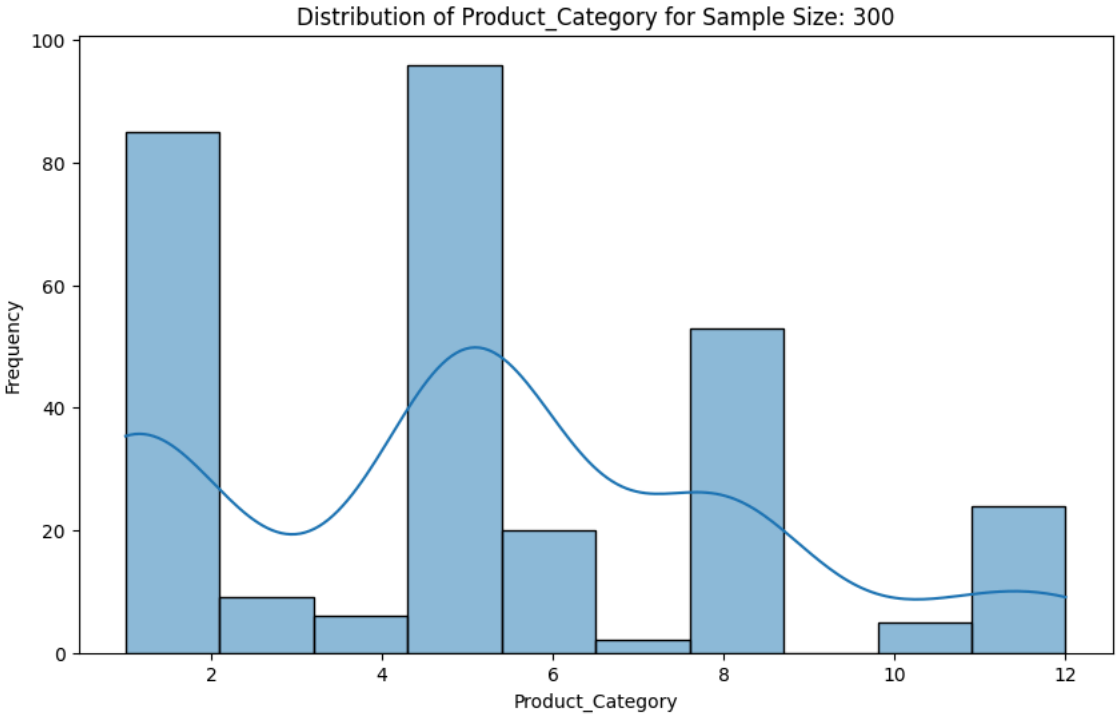

500
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Occupation

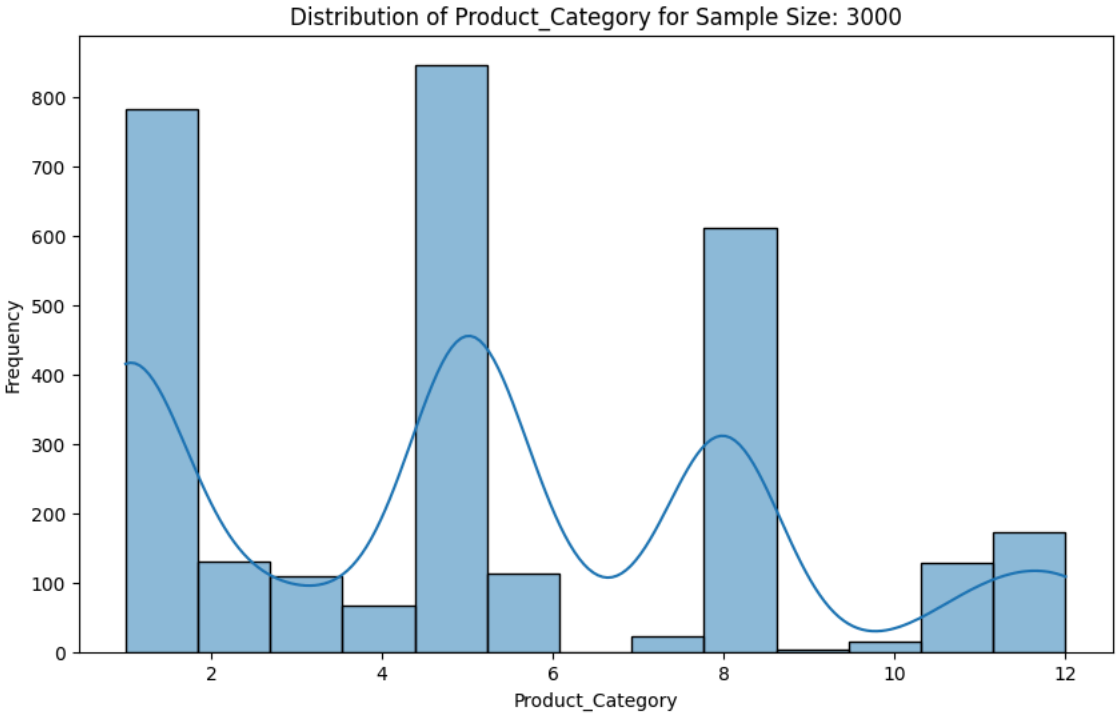

Distribution of Marital_Status for Sample Size: 300
Frequency
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0
Marital_Status

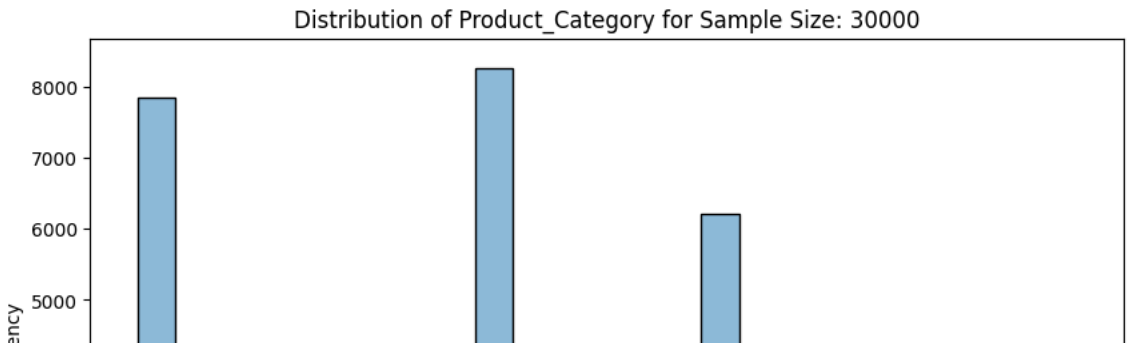

Distribution of Marital_Status for Sample Size: 3000
Frequency
1600
1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0
Marital_Status

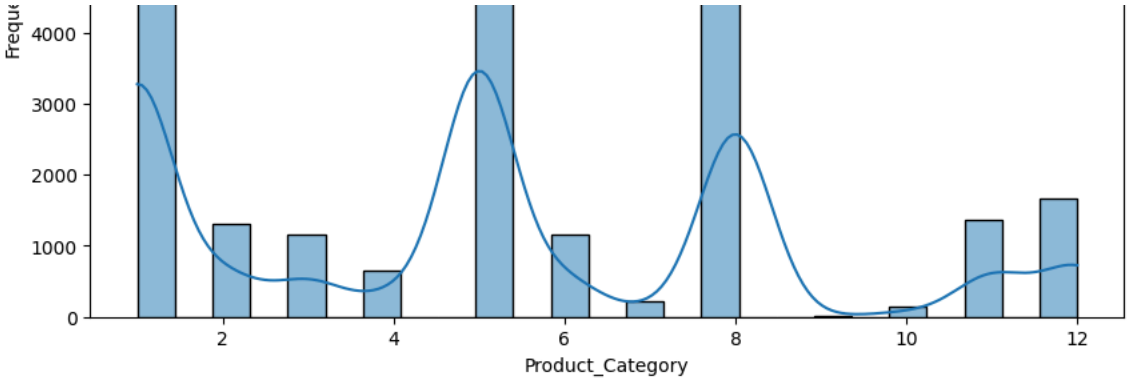

Distribution of Marital_Status for Sample Size: 30000
Frequency
17500
15000
12500
10000
7500

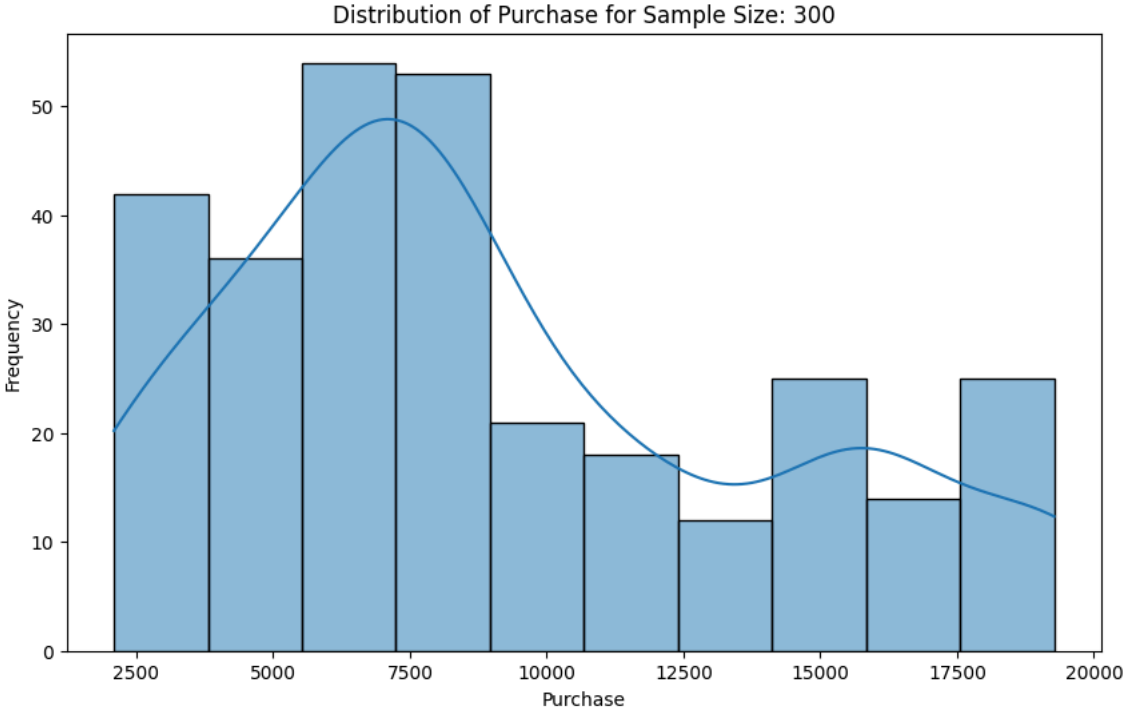

5000
2500
0
0.0
0.2
0.4
0.6
0.8
1.0
Marital_Status

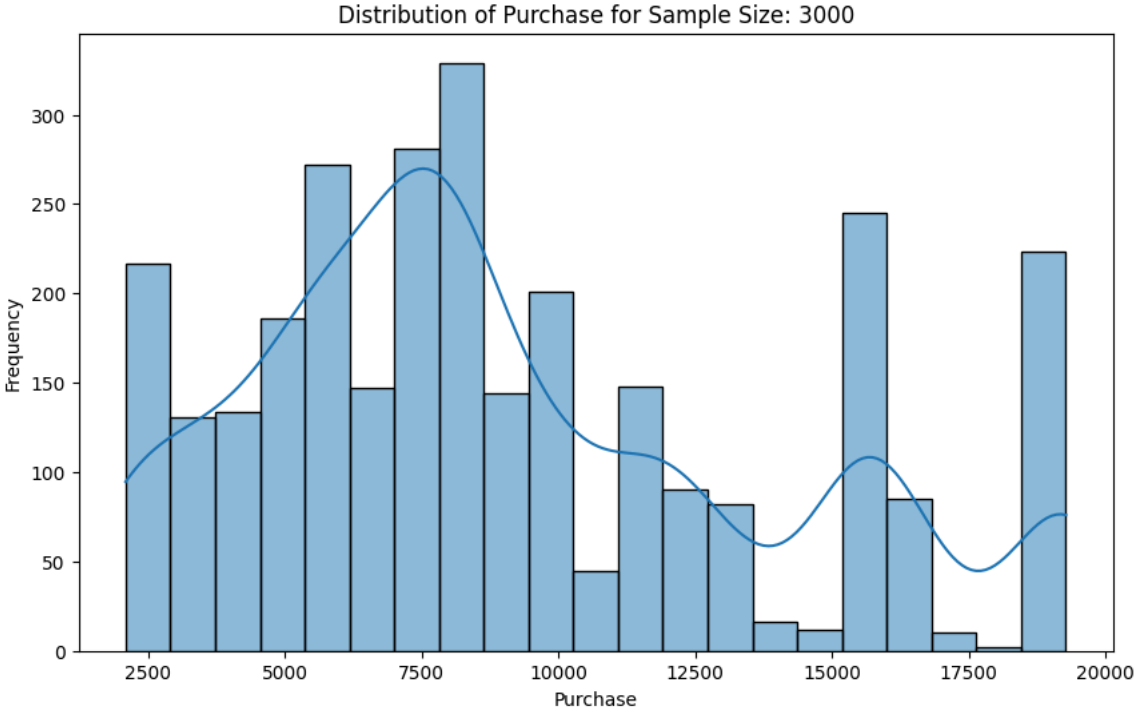

Distribution of Product_Category for Sample Size: 300
Frequency
Product_Category

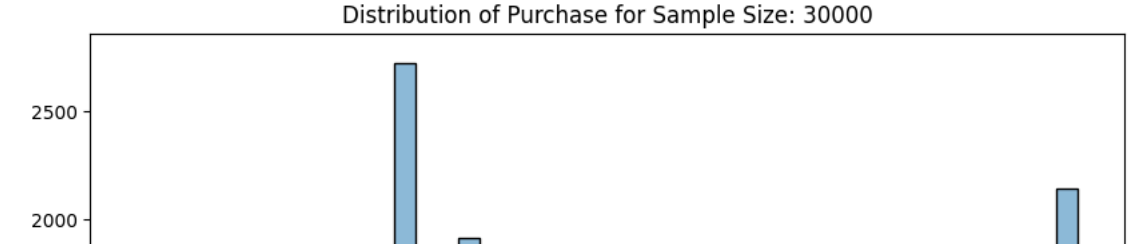

Distribution of Product_Category for Sample Size: 3000
Frequency
Product_Category

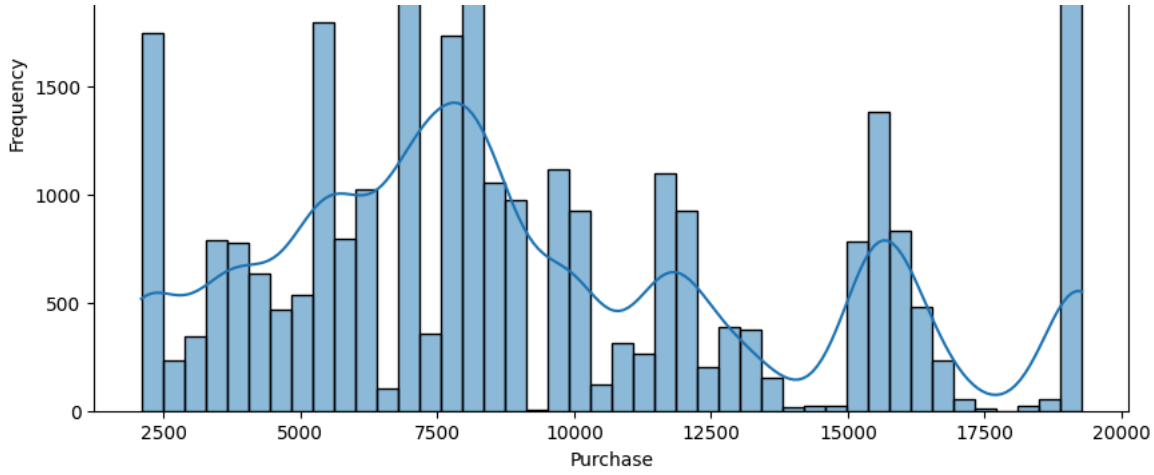

Distribution of Product_Category for Sample Size: 30000

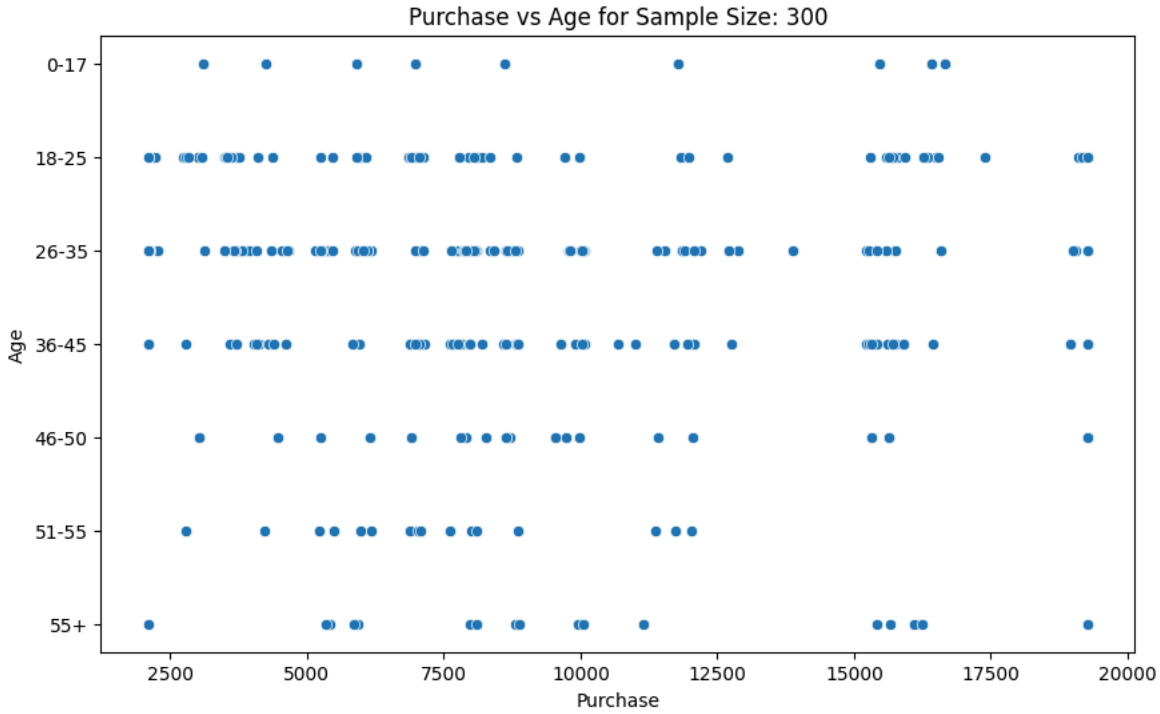

Frequ
4000
3000
2000
1000
0
2
4
6
8
10
12
Product_Category

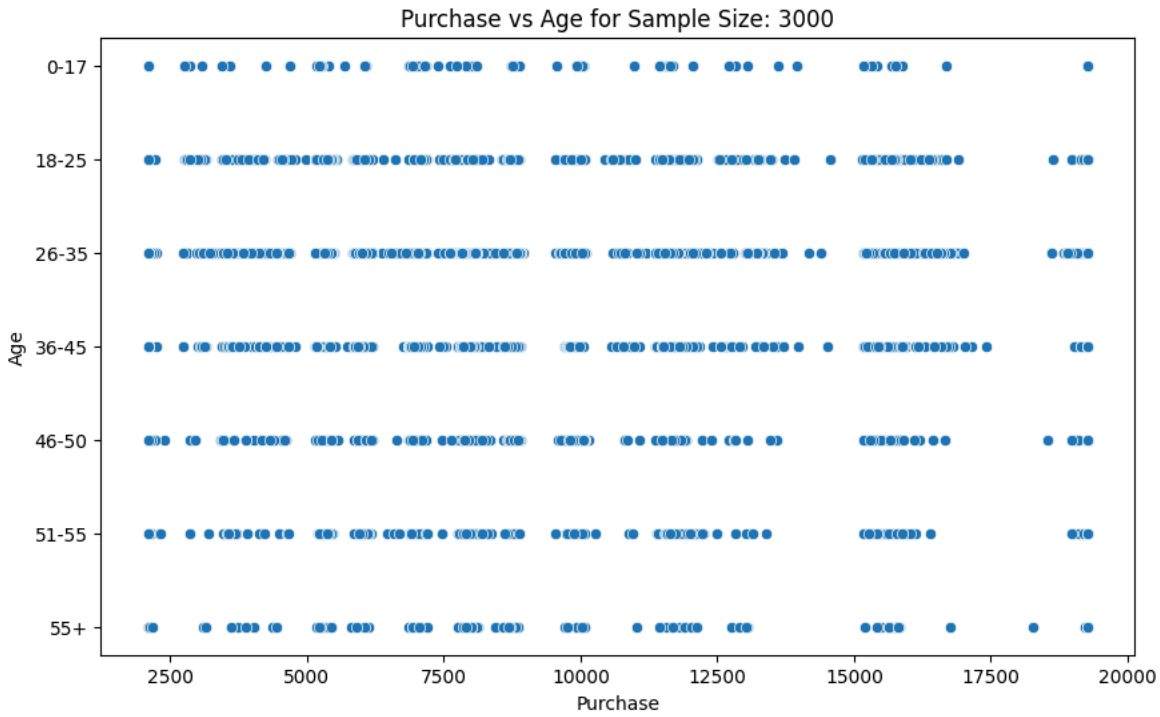

Distribution of Purchase for Sample Size: 300
Frequency
50
40
30
20
10
0
2500
5000
7500
10000
12500
15000
17500
20000
Purchase

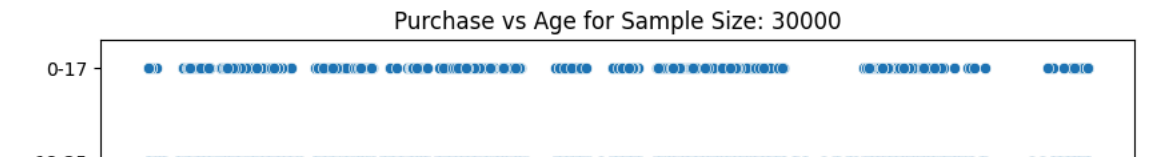

Distribution of Purchase for Sample Size: 3000
Frequency
300
250
200
150
100
50
0
2500
5000
7500
10000
12500
15000
17500
20000
Purchase

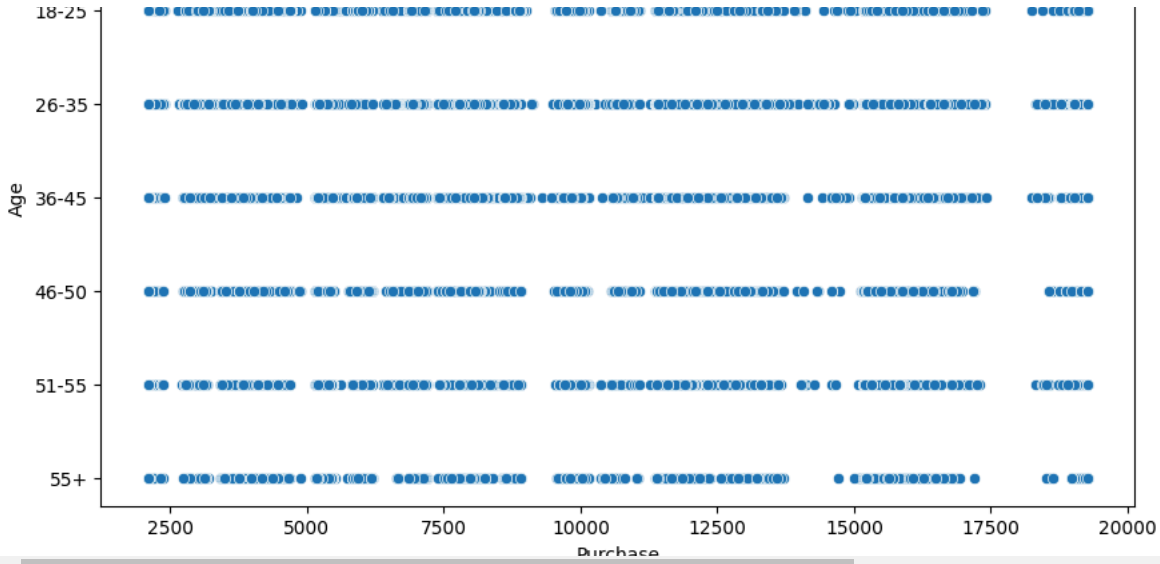

Distribution of Purchase for Sample Size: 30000
2500
2000


Frequency
0
500
1000
1500
2500
5000
7500
10000
12500
15000
17500
20000
Purchase


Purchase vs Age for Sample Size: 300
Age
0-17
18-25
26-35
36-45
46-50
51-55
55+
2500
5000
7500
10000
12500
15000
17500
20000
Purchase


Purchase vs Age for Sample Size: 3000
Age
0-17
18-25
26-35
36-45
46-50
51-55
55+
2500
5000
7500
10000
12500
15000
17500
20000
Purchase


Purchase vs Age for Sample Size: 30000
0-17


18-25
26-35
36-45
46-50
51-55
55+
Age
2500
5000
7500
10000
12500
15000
17500
20000
Purchase

Univariate Analysis Insights:

1) Distribution of Numerical Columns:

- For each numerical column in the dataset, the distribution was analyzed for sample sizes of 300, 3000, and 30000.
- Smaller Sample Sizes (300): The distributions are wider and less smooth, indicating more variability and less precision.
- Larger Sample Sizes (3000 and 30000): The distributions become narrower and more normally distributed, indicating less variability and more precision.

Bivariate Analysis Insights:

1) Purchase vs. Age:

- The relationship between the purchase amount and age was analyzed for sample sizes of 300, 3000, and 30000.
- Smaller Sample Sizes (300): The scatter plots may show more scattered points, indicating higher variability and less clear patterns.
- Larger Sample Sizes (3000 and 30000): The scatter plots become denser, showing clearer patterns and relationships between purchase amount and age.

Combined Insights

A) Product Category Insights:

a) Product Categories 1 and 5:

- Most popular across all age groups, especially the 26-35 age group.
- Significant interest from the 18-25 and 36-45 age groups.
- Males have significantly higher counts compared to females.

b) Product Category 8:

- Popular among the 26-35 and 36-45 age groups.
- Notable interest from the 18-25 age group.
- Males have higher counts compared to females.

c) Product Categories 2, 3, and 6:

- Moderate popularity, highest counts in the 26-35 age group.
- Interest from the 18-25 and 36-45 age groups.
- Males have higher counts compared to females.

d) Product Categories 7, 9, 10, and 12:

- Lower counts overall.
- Males have higher counts (2,778 for Category 7, 340 for Category 9, 3,963 for Category 10, and 2,415 for Category 12) compared to females (943 for Category 7, 70 for Category 9, 1,162 for Category 10, and 1,532 for Category 12).

d) Product Category 13:

- Moderate popularity.
- Males (23,895) have higher counts compared to females (7,151).

B) Age Group and Marital Status Insights:

a) 0-17 Age Group:

- Only unmarried individuals are spending.

b) 18-25 Age Group:

- Unmarried individuals spend significantly more than married individuals.

c) 26-35 Age Group:

- High spending from both married and unmarried individuals, with unmarried individuals spending more.

d) 36-45 Age Group:

- Unmarried individuals spend more than married individuals. 46-50 Age Group:
- Married individuals spend more than unmarried individuals.

e) 51-55 Age Group:

- Married individuals spend more than unmarried individuals.

f) 55+ Age Group:

- Married individuals spend more than unmarried individuals.

C) Confidence Interval Insights:

a) Confidence Interval for Males:

- 90% CI: (9419.61, 9444.18)
- 95% CI: (9417.19, 9446.81)
- 99% CI: (9411.59, 9452.12)

b) Confidence Interval for Females:

- 90% CI: (8732.60, 8772.00)
- 95% CI: (8729.44, 8775.74)
- 99% CI: (8719.19, 8782.57)

`Key Insights:`

- The average amount spent by males is higher than that spent by females, as indicated by the higher confidence interval ranges for males across all confidence levels.
- The confidence interval for females is wider than that for males, indicating greater variability in the purchase amounts among females.

`Questions Answered:`

- The confidence interval computed using the entire dataset is wider for females compared to males due to greater variability in the purchase amounts among females.

`Summary:`

- Smaller Sample Sizes: Wider confidence intervals indicate higher variability and less precision in the estimates.
- Larger Sample Sizes: Narrower confidence intervals indicate lower variability and higher precision in the estimates.
- Decreasing Width with Increasing Sample Size: As the sample size increases, the width of the confidence interval decreases for both males and females.
- Comparison Between Genders: The confidence interval widths for males are slightly wider than those for females at each sample size, suggesting more variability in the amount

spent by males.

- Overlap of Confidence Intervals:

a) Sample Size 300: Overlap between males (8558.16, 9674.50) and females (8231.65, 9210.81).

b) Sample Size 3000: No overlap between males (9258.51, 9602.60) and females (8561.38, 8902.14).

c) Sample Size 30000: No overlap between males (9370.94, 9473.62) and females (8746.99, 8847.29).

- From the results, we can see that the confidence intervals for different sample sizes overlap for the smallest sample size (300) but do not overlap for larger sample sizes (3000 and 30000). This indicates that with smaller sample sizes, the estimates are less precise, leading to overlapping confidence intervals. As the sample size increases, the estimates become more precise, resulting in non-overlapping confidence intervals.

D) Age Group Confidence Interval Insights:

a) Confidence Interval for Age 0-17:

- 95% CI: (8886.32, 9044.15)

b) Confidence Interval for Age 18-25:

- 95% CI: (9163.66, 9221.50)

c) Confidence Interval for Age 26-35:

- 95% CI: (9231.56, 9271.83)

d) Confidence Interval for Age 36-45:

- 95% CI: (9295.65, 9353.45)

e) Confidence Interval for Age 46-50:

- 95% CI: (9166.84, 9250.14)

f) Confidence Interval for Age 51-55:

- 95% CI: (9453.50, 9547.77)

g) Confidence Interval for Age 55+:

- 95% CI: (9273.53, 9394.92)

```
Key Insights:
```

- The age group 26-35 has the lowest variability in spending, as indicated by the narrowest confidence interval width.
- The age group 0-17 has the highest variability in spending, followed closely by the 55+ age group.
- Higher variability in spending within an age group suggests more diverse spending habits. For example, the 0-17 age group might include both low and high spenders, leading to a

wider confidence interval.
- Lower variability, as seen in the 26-35 age group, indicates more consistent spending patterns within that age group.

`Calculated 90%, 95%, and 99% Confidence Intervals for Gender,Marital_Status and Age group:`

- I have calculated the 90%, 95%, and 99% confidence intervals to gain clear insights into the differences in the average amount spent by males v/s females, as well as to observe the variations in the width of these intervals.
- I have calculated the 90%, 95%, and 99% confidence intervals to gain clear insights into the differences in the average amount spent by married v/s singe , as well as to observe the variations in the width of these intervals.
- I have calculated the 90%, 95%, and 99% confidence intervals to gain clear insights into the differences in the average amount spent by different age groups, as well as to observe the variations in the width of these intervals.

`OVERALL RECOMMENDATIONS:`

1) Targeted Marketing Campaigns:

- Focus on the 26-35 age group for most product categories, as they are the most active spenders.
- Develop campaigns targeting the 18-25 and 36-45 age groups, highlighting popular categories like 1, 5, and 8.

2) Gender-Specific Promotions:

- Create promotions that appeal to males, especially for categories 1, 5, and 8, where they show higher spending.
- Consider strategies to increase female engagement in these categories.

3) Marital Status-Based Strategies:

- For younger age groups (0-45), tailor marketing efforts towards unmarried individuals who tend to spend more.
- For older age groups (46+), focus on married individuals with higher spending.

4) Product Category Focus:

- Invest in promoting popular categories (1, 5, and 8) across all age groups.
- Consider strategies to boost interest in less popular categories (7, 9, 10, and 12).

5) Seasonal and Event-Based Campaigns:

- Utilize trend analysis to identify peak spending periods and plan seasonal promotions.
- Leverage events and holidays to boost sales in popular product categories.

6) Age Group-Specific Strategies:

- For the 0-17 age group, consider promotions that appeal to both low and high spenders to address the high variability in spending.
- For the 55+ age group, develop strategies that cater to the diverse spending habits within this group.

7) Gender-Specific Marketing:

- Since the confidence intervals for the average amount spent by males and females do not overlap, indicating a significant difference in spending,

```
Walmart can:
```

- Develop targeted marketing campaigns tailored to the spending habits of each gender.
- Customize product offerings based on the preferences of males and females.
- Create gender-specific promotions and discounts to maximize sales.

8) Uniform Marketing for Marital Status:

- Since the confidence intervals for the average amount spent by married and unmarried individuals overlap, suggesting no significant difference in spending,

```
Walmart can:
```

- Implement uniform marketing strategies for both married and unmarried individuals.
- Ensure that products appealing to both groups are equally accessible.
- Offer similar promotions and discounts to both married and unmarried individuals.

9) Data-Driven Decision Making

- Continuous Monitoring and Analysis: Regularly monitor and analyze spending patterns across different age groups. This can help in identifying any changes in spending behavior and adjusting marketing strategies accordingly.
- Leveraging Customer Feedback: Collect and analyze feedback from customers in different age groups. This can provide valuable insights into their preferences and help in refining marketing and product strategies.

10) Enhancing Customer Experience

- Personalized Shopping Experience: Use the insights from the analysis to personalize the shopping experience for customers in different age groups. This can involve personalized recommendations, targeted advertisements, and customized shopping experiences.
- Improving Customer Loyalty Programs: Design loyalty programs that cater to the specific needs and preferences of different age groups. This can help in building long-term customer loyalty and increasing customer lifetime value.