Walmart Business Case Study

Submitted by :

B. Raju Naik

0.1 About Walmart:

The retail company under consideration is a prominent American multinational corporation. It manages a network of supercenters, discount departmental stores, and grocery outlets within the United States. This retail giant serves a vast customer base of over 100 million individuals globally.

0.2 Business Problem:

The retail management team is interested in examining customer purchase behavior, specifically the purchase amount, in relation to gender and other factors. Their objective is to gain insights into whether there are variations in spending between female and male customers on Black Friday. This analysis is critical for making strategic decisions.

0.3 About Dataset:

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

The dataset has the following features:

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

1. Exploratory Data Analysis

In [3]:
```
# importing requisite libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
```

In [6]:
```
# importing the dataset

df = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/293/original/walmart_data.csv?164
df.head()
```

Out[6]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purchase
0	1000001	P00069042	F	0-17	10	A	2	0	3	8370
1	1000001	P00248942	F	0-17	10	A	2	0	1	15200
2	1000001	P00087842	F	0-17	10	A	2	0	12	1422
3	1000001	P00085442	F	0-17	10	A	2	0	12	1057
4	1000002	P00285442	M	55+	16	C	4+	0	8	7969

1.1 Analyzing basic metrics about dataset

In [7]:
```
#shape(rows,column)

df.shape
```

Out[7]:
```
(550068, 10)
```

In [8]:
```
# Total Number of rows

df.size
```

Out[8]:
```
5500680
```

In [10]:
```
#Data types of column

df.dtypes
```

Out[10]:
```
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
```

In [12]:
```
# Column Names

df.columns
```

Out[12]:
```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category',
       'Purchase'],
      dtype='object')
```

In [14]:
```
# dataset information

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  object
 2   Gender                      550068 non-null  object
 3   Age                         550068 non-null  object
 4   Occupation                  550068 non-null  int64 
 5   City_Category               550068 non-null  object
 6   Stay_In_Current_City_Years  550068 non-null  object
 7   Marital_Status              550068 non-null  int64 
 8   Product_Category            550068 non-null  int64 
 9   Purchase                    550068 non-null  int64 
dtypes: int64(5), object(5)
memory usage: 42.0+ MB
```

In [15]:

```
# conversion of categorical attributes to 'category'

column = ["User_ID", "Occupation", "Marital_Status", "Product_Category"]

df[column] = df[column].astype("object")
```

In [16]:

```
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  object
 1   Product_ID                  550068 non-null  object
 2   Gender                      550068 non-null  object
 3   Age                         550068 non-null  object
 4   Occupation                  550068 non-null  object
 5   City_Category               550068 non-null  object
 6   Stay_In_Current_City_Years  550068 non-null  object
 7   Marital_Status              550068 non-null  object
 8   Product_Category            550068 non-null  object
 9   Purchase                    550068 non-null  int64 
dtypes: int64(1), object(9)
memory usage: 42.0+ MB
```

In [18]:
```
# Statistical summary

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

Out[18]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	
count	550068.0	550068	550068	550068	550068.0	550068	550068	550068.0	550068.0	55
unique	5891.0	3631	2	7	21.0	3	5	2.0	20.0	
top	1001680.0	P00265242	M	26-35	4.0	B	1	0.0	5.0	
freq	1026.0	1880	414259	219587	72308.0	231173	193821	324731.0	150933.0	
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2

In [19]:
```
df.describe(include = "object")
```

Out[19]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category
count	550068	550068	550068	550068	550068	550068	550068	550068	550068
unique	5891	3631	2	7	21	3	5	2	20
top	1001680	P00265242	M	26-35	4	B	1	0	5
freq	1026	1880	414259	219587	72308	231173	193821	324731	150933

In [24]:
```
df.groupby("Gender")['Purchase'].describe()
```

Out[24]:

	count	mean	std	min	25%	50%	75%	max
Gender								
F	135809.0	8734.565765	4767.233289	12.0	5433.0	7914.0	11400.0	23959.0
M	414259.0	9437.526040	5092.186210	12.0	5863.0	8098.0	12454.0	23961.0

In [28]:

```
# Finding unique values in dataset

df.nunique()
```

Out[28]:

```
User_ID                        5891
Product_ID                     3631
Gender                            2
Age                               7
Occupation                       21
City_Category                     3
Stay_In_Current_City_Years        5
Marital_Status                    2
Product_Category                 20
Purchase                      18105
dtype: int64
```

In [31]:

```
# Check for duplicate values

df.duplicated()
```

Out[31]:

```
0         False
1         False
2         False
3         False
4         False
          ...
550063    False
550064    False
550065    False
550066    False
550067    False
Length: 550068, dtype: bool
```

In [32]:

```
df[df.duplicated()]
```

Out[32]:

User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purchase

Here we can see that there no duplicate values in the dataset

In [33]:
```
df.head()
```

Out[33]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purchase
0	1000001	P00069042	F	0-17	10	A	2	0	3	8370
1	1000001	P00248942	F	0-17	10	A	2	0	1	15200
2	1000001	P00087842	F	0-17	10	A	2	0	12	1422
3	1000001	P00085442	F	0-17	10	A	2	0	12	1057
4	1000002	P00285442	M	55+	16	C	4+	0	8	7969

Insights:

- All entries in the dataset are complete, with no missing values, ensuring data integrity.
- There are no duplicate records in the dataset, maintaining its integrity and accuracy.
- The dataset encompasses a diverse range of 20 unique product types, providing a wide array of options.
- Male customers appear to be the dominant demographic in product transactions, based on the data analysis.
- Purchases are spread across three distinct city categories: A, B, and C, indicating geographical diversity in transactions.
- The majority of users in the dataset fall within the 26-34 age range, suggesting a specific demographic preference.
- There is a clear preference among users for the 5th product category, as evidenced by the data trends.

1.2 : Non-Graphical Analysis: Value counts and unique attributes

In [34]:
```
# defining a function for value_count and unique values

def Unique(x):
```

```
    print("Unique_values:",df[x].unique())
    print("unique_values count:",df[x].nunique())
    print("value_count:",df[x].value_counts())
```

In [37]:
```
# User_id

Unique("User_ID")
```

```
Unique_values: [1000001 1000002 1000003 ... 1004113 1005391 1001529]
unique_values count: 5891
value_count: User_ID
1001680    1026
1004277     979
1001941     898
1001181     862
1000889     823
           ...
1002690       7
1002111       7
1005810       7
1004991       7
1000708       6
Name: count, Length: 5891, dtype: int64
```

In [38]:
```
# Product_ID

Unique("Product_ID")
```

```
Unique_values: ['P00069042' 'P00248942' 'P00087842' ... 'P00370293' 'P00371644'
 'P00370853']
unique_values count: 3631
value_count: Product_ID
P00265242    1880
P00025442    1615
P00110742    1612
P00112142    1562
P00057642    1470
             ...
P00314842       1
P00298842       1
P00231642       1
P00204442       1
P00066342       1
Name: count, Length: 3631, dtype: int64
```

In [39]:
```
# Gender

Unique("Gender")
```

```
Unique_values: ['F' 'M']
unique_values count: 2
value_count: Gender
M    414259
F    135809
Name: count, dtype: int64
```

In [40]:
```
# Age

Unique("Age")
```

```
Unique_values: ['0-17' '55+' '26-35' '46-50' '51-55' '36-45' '18-25']
unique_values count: 7
value_count: Age
26-35    219587
36-45    110013
18-25     99660
46-50     45701
51-55     38501
55+       21504
0-17      15102
Name: count, dtype: int64
```

In [41]:
```
# Occupation

Unique("Occupation")
```

```
Unique_values: [10 16 15 7 20 9 1 12 17 0 3 4 11 8 19 2 18 5 14 13 6]
unique_values count: 21
value_count: Occupation
4     72308
0     69638
7     59133
1     47426
17    40043
20    33562
12    31179
14    27309
2     26588
16    25371
6     20355
3     17650
10    12930
5     12177
15    12165
11    11586
19     8461
13     7728
18     6622
9      6291
8      1546
Name: count, dtype: int64
```

In [42]:
```
# City_Category

Unique("City_Category")
```

```
Unique_values: ['A' 'C' 'B']
unique_values count: 3
value_count: City_Category
B    231173
C    171175
A    147720
Name: count, dtype: int64
```

In [43]:
```
# Stay_In_Current_City_Years

Unique("Stay_In_Current_City_Years")
```

```
Unique_values: ['2' '4+' '3' '1' '0']
unique_values count: 5
value_count: Stay_In_Current_City_Years
1      193821
2      101838
3       95285
4+      84726
0       74398
Name: count, dtype: int64
```

In [45]:
```
# Marital_Status

Unique("Marital_Status")
```

```
Unique_values: [0 1]
unique_values count: 2
value_count: Marital_Status
0    324731
1    225337
Name: count, dtype: int64
```

In [46]:
```
# Product_Category

Unique("Product_Category")
```

```
Unique_values: [3 1 12 8 5 4 2 6 14 11 13 15 7 16 18 10 17 9 20 19]
unique_values count: 20
value_count: 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
```

In [47]:

```python
# Purchase

Unique("Purchase")
```

```
Unique_values: [ 8370 15200  1422 ...   135   123   613]
unique_values count: 18105
value_count: 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
```

1.2.1 Dataset observation by Non-Graphical Analysis: Value counts and unique attributes

1. Gender Distribution:

- The dataset consists of approximately 75% male users and 25% female users, with 414,259 and 135,809 users respectively.

2. Age Group Distribution:

- The largest age group in the dataset comprises users aged 26-35, accounting for roughly 40%.
- Users in the age groups 0-17 and 55+ each contribute approximately 3%.

3. Occupation Distribution:

- Occupations 4 and 0 collectively make up about 23% of the dataset, with occupation 4 being the most prevalent.
- Occupations 8 and 5 together represent approximately 13%.

4. City Category Distribution:

- City category B accounts for around 42% of the dataset.
- City category C makes up approximately 31%, while category A represents about 27%.

5. Stay in Current City Distribution:

- Users who have stayed in their current city for 1 year constitute nearly 35%.
- Approximately 19% have stayed for 2 years, and around 17% have stayed for 3 years.

6. Marital Status Distribution:

- Unmarried users make up roughly 59% of the dataset, while married users account for approximately 41%.

7. Product Category Distribution:

- Product category 5 comprises about 26% of the dataset, making it the most prevalent.
- Categories 1, 8, and 11 each represent around 8-9%.

8. Purchase Amount Distribution:

- Due to a wide range of purchase amounts, no single value represents a significant percentage of the dataset.
- The most common purchase amount, 7,011, represents less than 0.5%.

2. Detect Null values and outliers

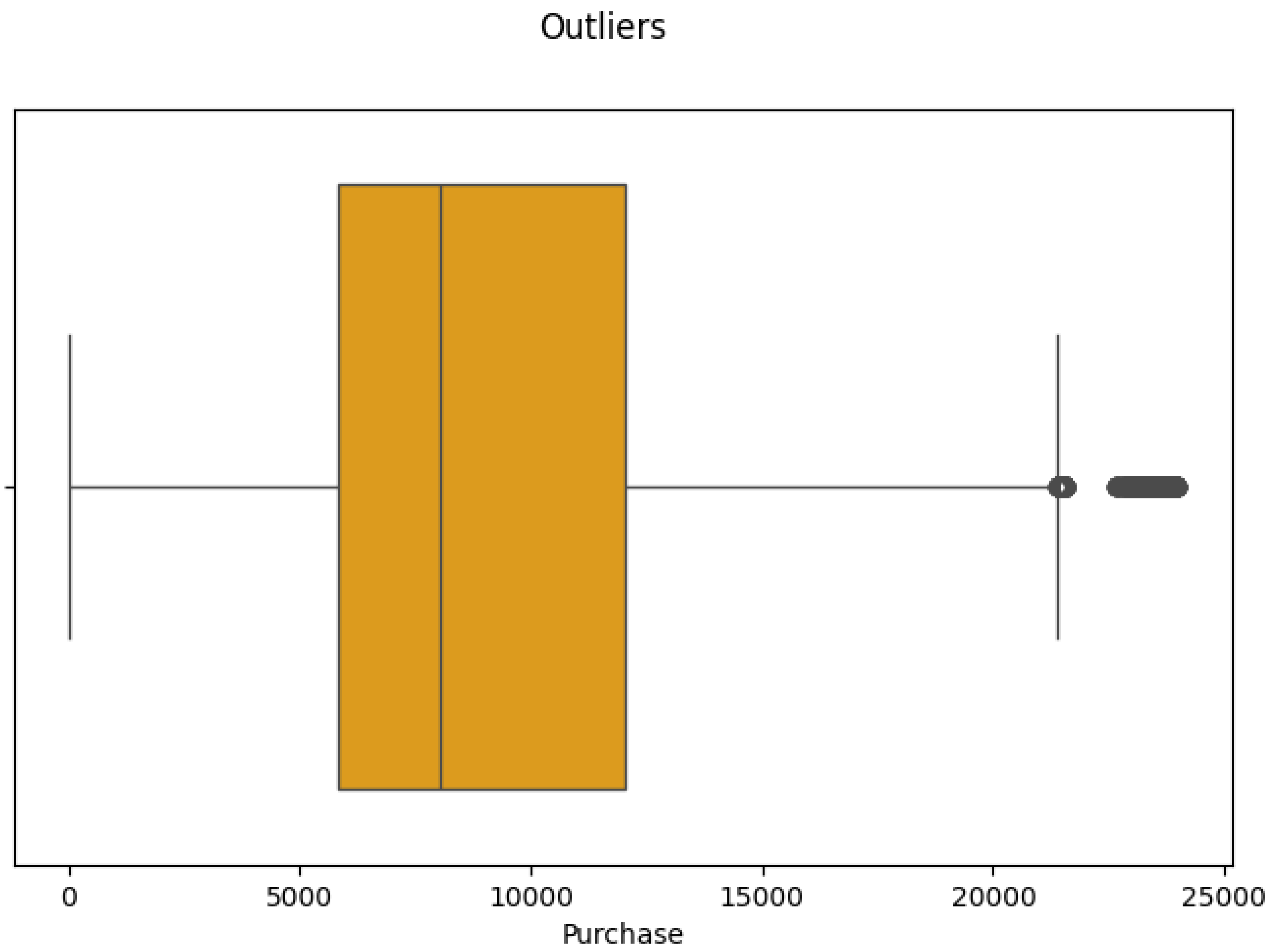
2.1 Checking for Missing Values and Outliers:

In [52]:
```
df.isnull().sum()
```

Out[52]:
```
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
```

Insight: There are no null values in the dataset.

In [57]:
```
fig, ax = plt.subplots(figsize = (8,5))
fig.suptitle("Outliers")

sns.boxplot(data = df, x = "Purchase", color = "orange")
plt.show()
```



Insights: Based on the graphical representation, it is evident that Purchase has only a minor presence of outliers.

2.2 Clipping the data between 5th and 95th percantile

In [132…

```
import numpy as np

# Calculate the 5th and 95th percentiles
percentile_5 = np.percentile(df['Purchase'], 5)
percentile_95 = np.percentile(df['Purchase'], 95)
```

```
# Clip the data between the 5th and 95th percentiles
clipped_data = np.clip(df['Purchase'], percentile_5, percentile_95)

# Update the 'Purchase' column with the clipped data
df['Purchase'] = clipped_data

df
```

Out[132]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purch
0	1000001	P00069042	F	0-17	10	A	2	0	3	8
1	1000001	P00248942	F	0-17	10	A	2	0	1	15
2	1000001	P00087842	F	0-17	10	A	2	0	12	1
3	1000001	P00085442	F	0-17	10	A	2	0	12	1
4	1000002	P00285442	M	55+	16	C	4+	0	8	7
...	...	...	...	...	...	...	...	...	...	
550063	1006033	P00372445	M	51-55	13	B	1	1	20	1
550064	1006035	P00375436	F	26-35	1	C	3	0	20	1
550065	1006036	P00375436	F	26-35	15	B	4+	1	20	1
550066	1006038	P00375436	F	55+	1	C	2	0	20	1
550067	1006039	P00371644	F	46-50	0	B	4+	1	20	1

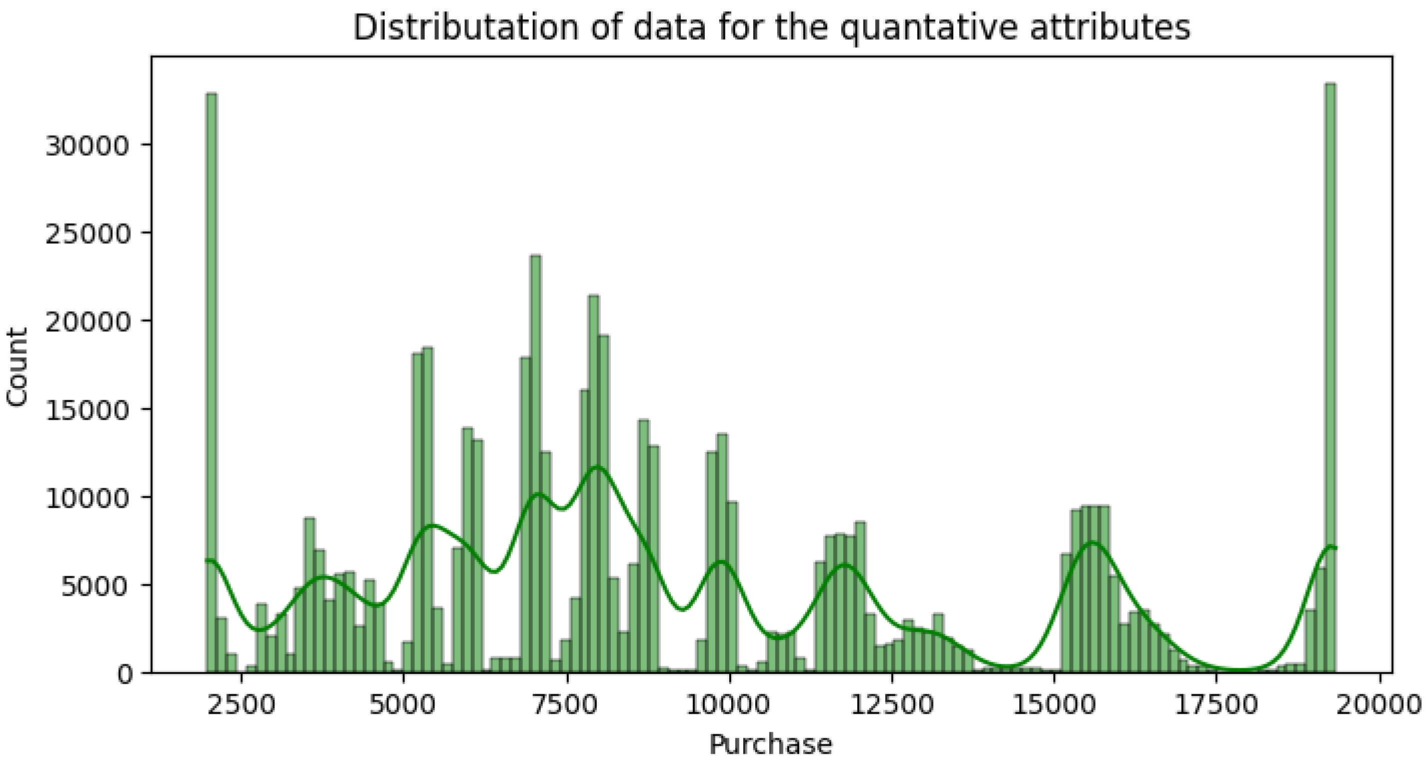
550068 rows × 10 columns

```
Warning: total number of rows (550068) exceeds max_rows (20000). Limiting to first (20000) rows.
Warning: total number of rows (550068) exceeds max_rows (20000). Limiting to first (20000) rows.
```

3. DataExploration

3.1 Univariate Analysis

In [70]:
```
# Distributation of data for the quantative attributes

plt.figure(figsize=(8,4))
plt.title("Distributation of data for the quantative attributes")
sns.histplot(data=df, x="Purchase", kde=True, color = "green")
plt.show()
```

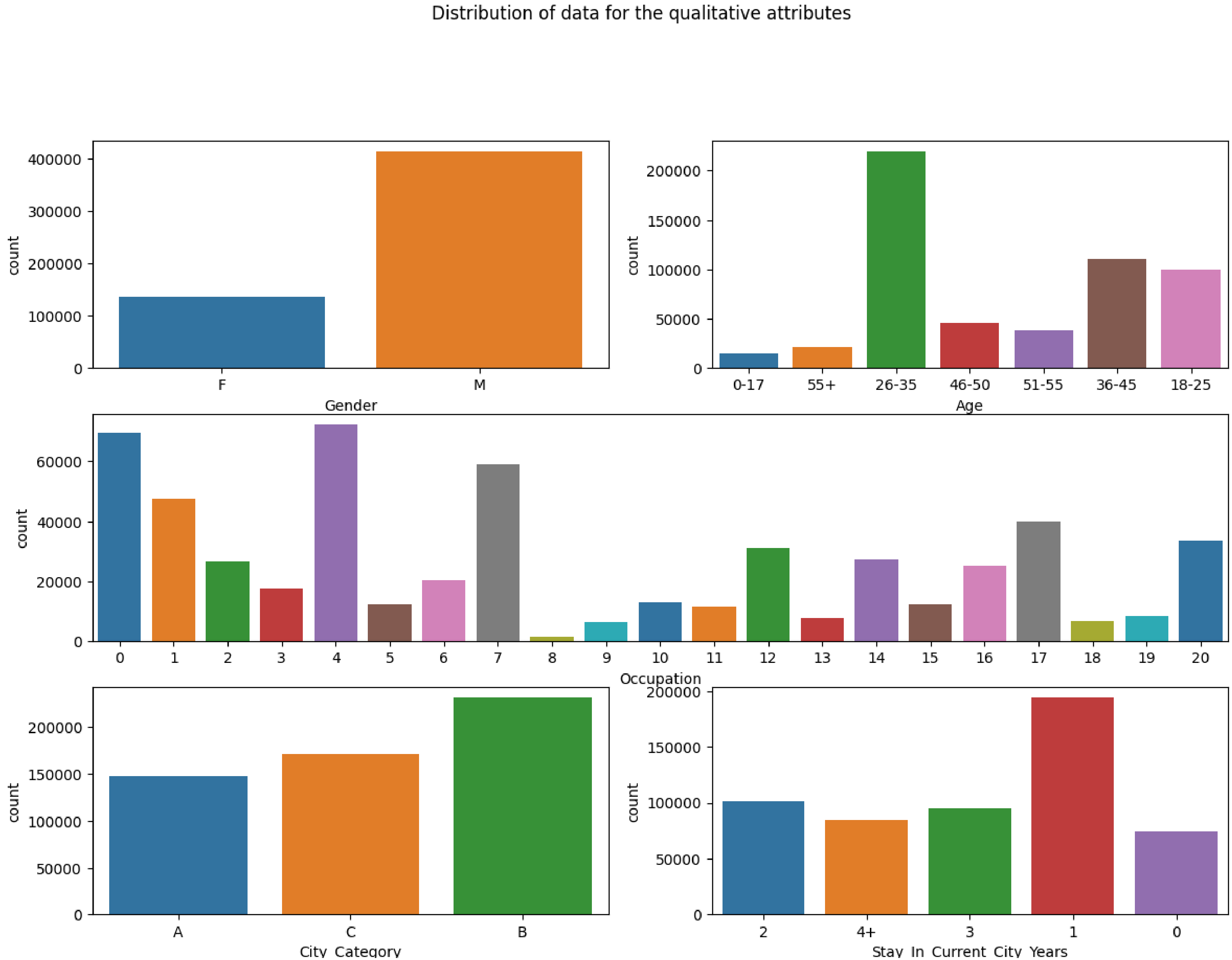


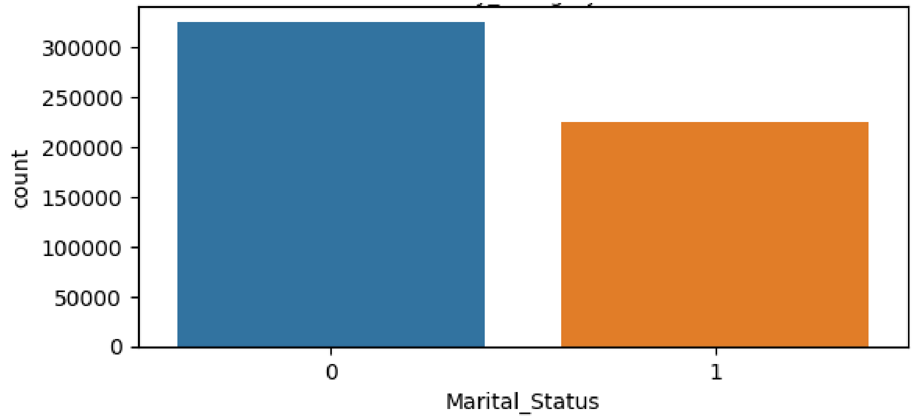
In [79]:
```
# Define a custom color palette for each plot
colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b", "#e377c2", "#7f7f7f", "#bcbd22", "#17becf'

fig, ax = plt.subplots(4, 2, figsize=(14, 13))
fig.suptitle("Distribution of data for the qualitative attributes")

plt.subplot(4, 2, 1)
sns.countplot(data=df, x="Gender", palette=colors)
```

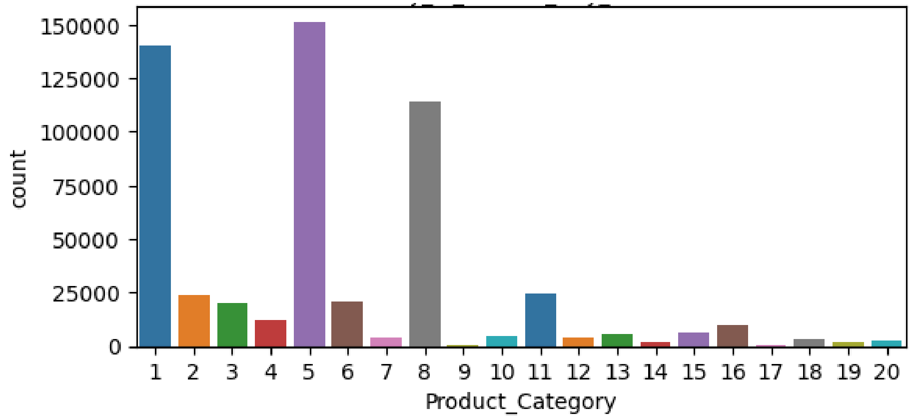
```
plt.subplot(4, 2, 2)
sns.countplot(data=df, x="Age", palette=colors)

plt.subplot(4, 2, (3, 4))
sns.countplot(data=df, x="Occupation", palette=colors)

plt.subplot(4, 2, 5)
sns.countplot(data=df, x="City_Category", palette=colors)

plt.subplot(4, 2, 6)
sns.countplot(data=df, x="Stay_In_Current_City_Years", palette=colors)

plt.subplot(4, 2, 7)
sns.countplot(data=df, x="Marital_Status", palette=colors)

plt.subplot(4, 2, 8)
sns.countplot(data=df, x="Product_Category", palette=colors)

plt.show()
```

```
<ipython-input-79-d5a2c114fbe6>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.countplot(data=df, x="Gender", palette=colors)
<ipython-input-79-d5a2c114fbe6>:8: UserWarning: The palette list has more values (10) than needed (2), which may not
be intended.
  sns.countplot(data=df, x="Gender", palette=colors)
<ipython-input-79-d5a2c114fbe6>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.countplot(data=df, x="Age", palette=colors)
<ipython-input-79-d5a2c114fbe6>:11: UserWarning: The palette list has more values (10) than needed (7), which may not
be intended.
  sns.countplot(data=df, x="Age", palette=colors)
<ipython-input-79-d5a2c114fbe6>:13: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated sinc
e 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.
  plt.subplot(4, 2, (3, 4))
<ipython-input-79-d5a2c114fbe6>:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.countplot(data=df, x="Occupation", palette=colors)
<ipython-input-79-d5a2c114fbe6>:14: UserWarning:
The palette list has fewer values (10) than needed (21) and will cycle, which may produce an uninterpretable plot.
  sns.countplot(data=df, x="Occupation", palette=colors)
<ipython-input-79-d5a2c114fbe6>:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.countplot(data=df, x="City_Category", palette=colors)
<ipython-input-79-d5a2c114fbe6>:17: UserWarning: The palette list has more values (10) than needed (3), which may not
be intended.
  sns.countplot(data=df, x="City_Category", palette=colors)
<ipython-input-79-d5a2c114fbe6>:20: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.
```

```
  sns.countplot(data=df, x="Stay_In_Current_City_Years", palette=colors)
<ipython-input-79-d5a2c114fbe6>:20: UserWarning: The palette list has more values (10) than needed (5), which may not
be intended.
  sns.countplot(data=df, x="Stay_In_Current_City_Years", palette=colors)
<ipython-input-79-d5a2c114fbe6>:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.countplot(data=df, x="Marital_Status", palette=colors)
<ipython-input-79-d5a2c114fbe6>:23: UserWarning: The palette list has more values (10) than needed (2), which may not
be intended.
  sns.countplot(data=df, x="Marital_Status", palette=colors)
<ipython-input-79-d5a2c114fbe6>:26: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.countplot(data=df, x="Product_Category", palette=colors)
<ipython-input-79-d5a2c114fbe6>:26: UserWarning:
The palette list has fewer values (10) than needed (20) and will cycle, which may produce an uninterpretable plot.
  sns.countplot(data=df, x="Product_Category", palette=colors)
```


Distribution of data for the qualitative attributes
count
Gender
F
M
Age
0-17
55+
26-35
46-50
51-55
36-45
18-25
Occupation
City_Category
A
C
B
Stay_In_Current_City_Years
2
4+
3
1
0





3.1.1 Insights:

1. Gender Distribution:

- The dataset exhibits a notable gender imbalance, with male users comprising a significant majority. This suggests a potential gender-based trend in shopping behavior.

2. Age Group Preferences:

- The age group between 26 and 35 emerges as the most prevalent demographic, indicating a focus on users within the 18 to 45 age range.

3. Occupation Trends:

- Occupations labeled 0, 4, and 7 appear frequently among the 20 occupation types, indicating potential areas of interest or engagement.

4. City Residence:

- City category 'B' demonstrates the highest user concentration, while categories 'A' and 'C' display a relatively more evenly distributed user population.

5. Length of Residence:

- The majority of users have resided in their current city for over one year, suggesting stability in residential status.

6. Marital Status:

- Unmarried users outnumber married users in the dataset, indicating a demographic skew towards single individuals.

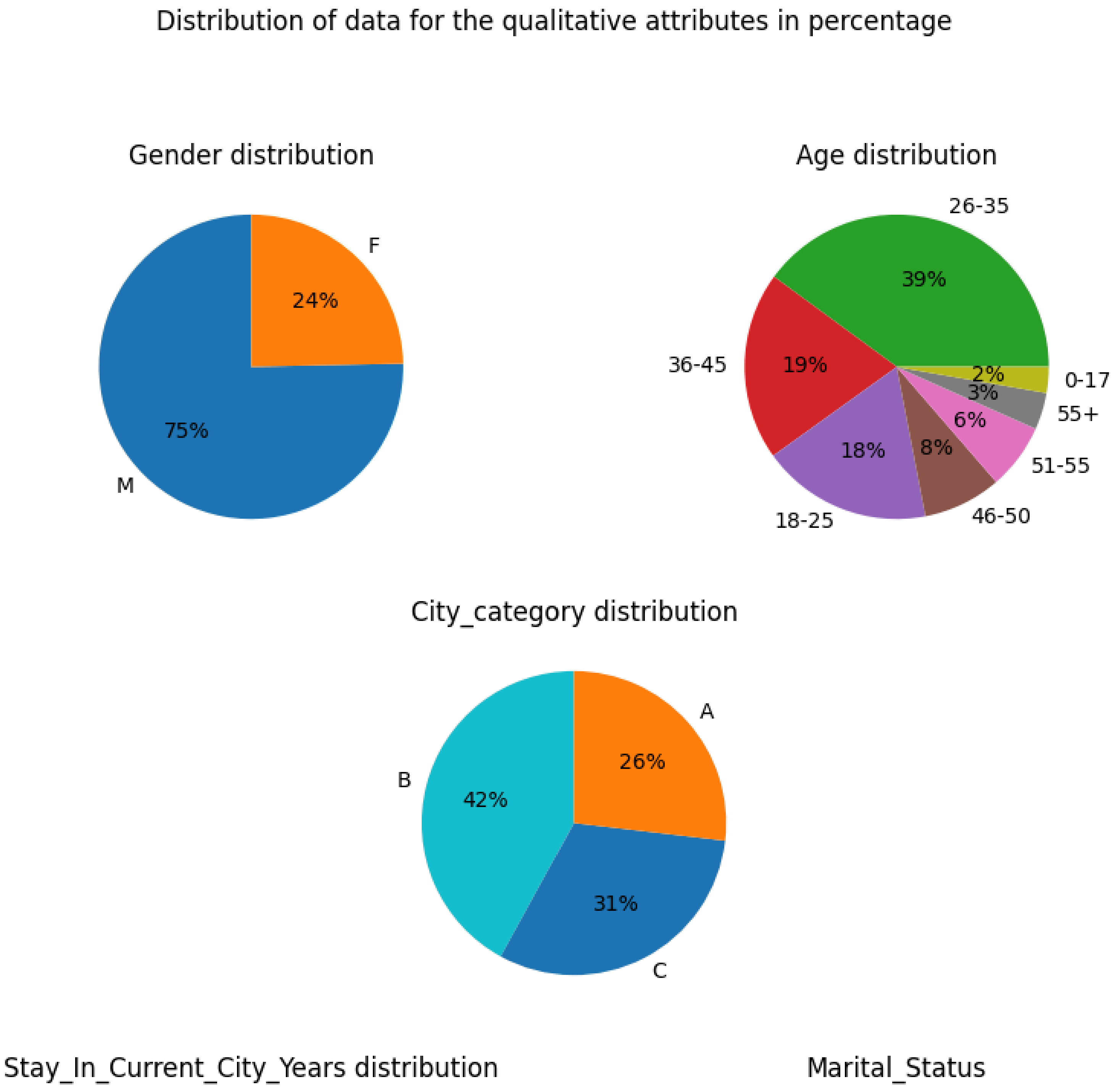
7. Product Category Preferences:

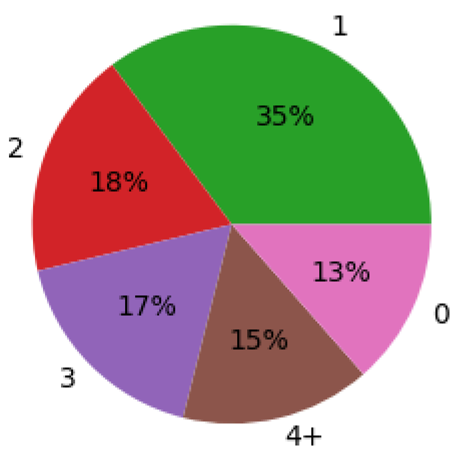
- Product categories 5, 1, and 8 emerge as the most favored among users, suggesting particular preferences or interests in these categories.

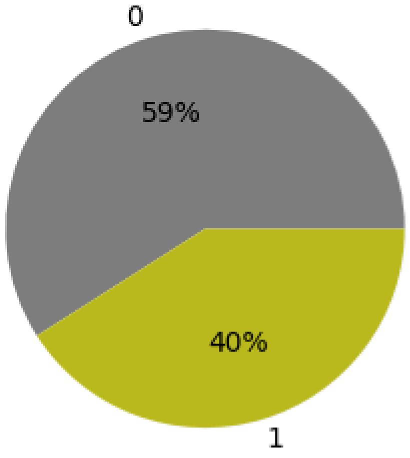
In [81]:
```
fig, ax = plt.subplots(3, 2, figsize=(10, 11))
fig.suptitle("Distribution of data for the qualitative attributes in percentage")

# Define custom colors for each pie chart
colors_gender = ['#1f77b4', '#ff7f0e']
colors_age = ['#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22']
colors_city_category = ['#17becf', '#1f77b4', '#ff7f0e']
colors_stay_years = ['#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2']
colors_marital_status = ['#7f7f7f', '#bcbd22']

# Gender Distribution
plt.subplot(3, 2, 1)
data_Gender = df['Gender'].value_counts(normalize=True) * 100
plt.pie(data_Gender, labels=data_Gender.index, autopct='%d%%', startangle=90, colors=colors_gender)
plt.title("Gender distribution")

# Age Group Preferences
plt.subplot(3, 2, 2)
data_Age = df['Age'].value_counts(normalize=True) * 100
plt.pie(data_Age, labels=data_Age.index, autopct='%d%%', startangle=0, colors=colors_age)
plt.title("Age distribution")

# City Category Distribution
plt.subplot(3, 2, (3, 4))
data_City_Category = df['City_Category'].value_counts(normalize=True) * 100
plt.pie(data_City_Category, labels=data_City_Category.index, autopct='%d%%', startangle=90, colors=colors_city_catego
plt.title("City_category distribution")

# Stay in Current City Distribution
plt.subplot(3, 2, 5)
data_Stay_In_Current_City_Years = df['Stay_In_Current_City_Years'].value_counts(normalize=True) * 100
```

```
plt.pie(data_Stay_In_Current_City_Years, labels=data_Stay_In_Current_City_Years.index, autopct='%d%%', startangle=0, c
plt.title("Stay_In_Current_City_Years distribution")

# Marital Status Distribution
plt.subplot(3, 2, 6)
data_Marital_Status = df["Marital_Status"].value_counts(normalize=True) * 100
plt.pie(data_Marital_Status, labels=data_Marital_Status.index, autopct='%d%%', startangle=0, colors=colors_marital_sta
plt.title("Marital_Status")
plt.show()
```

```
<ipython-input-81-0587910a7c66>:24: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated sinc
e 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.
  plt.subplot(3, 2, (3, 4))
```


Distribution of data for the qualitative attributes in percentage
Gender distribution
F
24%
75%
M
Age distribution
26-35
39%
36-45
19%
2%
0-17
3%
55+
6%
51-55
18%
8%
18-25
46-50
City_category distribution
A
26%
B
42%
31%
C
Stay_In_Current_City_Years distribution
Marital_Status





3.2 Bivariate Analysis

In [82]:
```
df.head(1)
```

Out[82]:

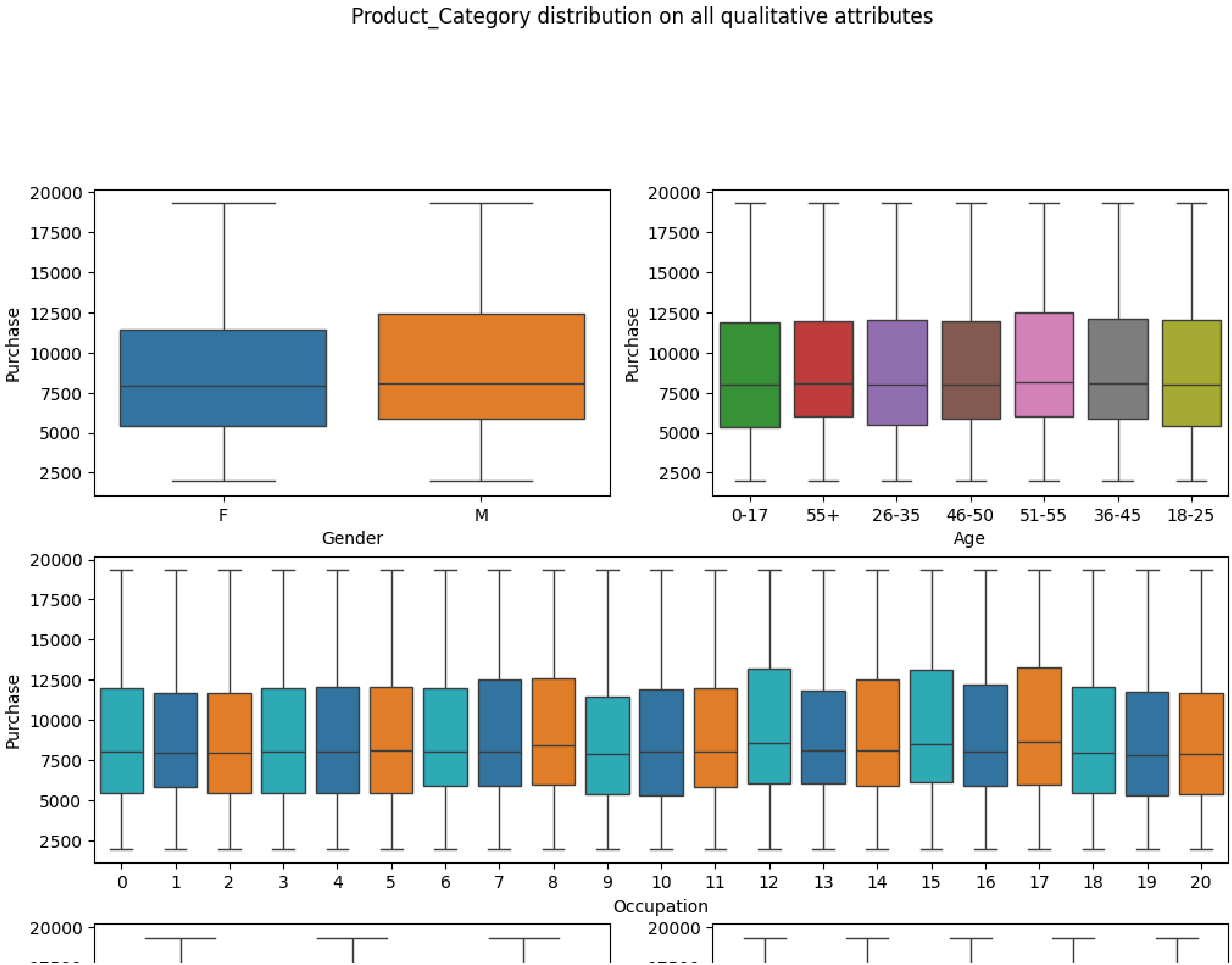
	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purchase
0	1000001	P00069042	F	0-17	10	A	2	0	3	8370

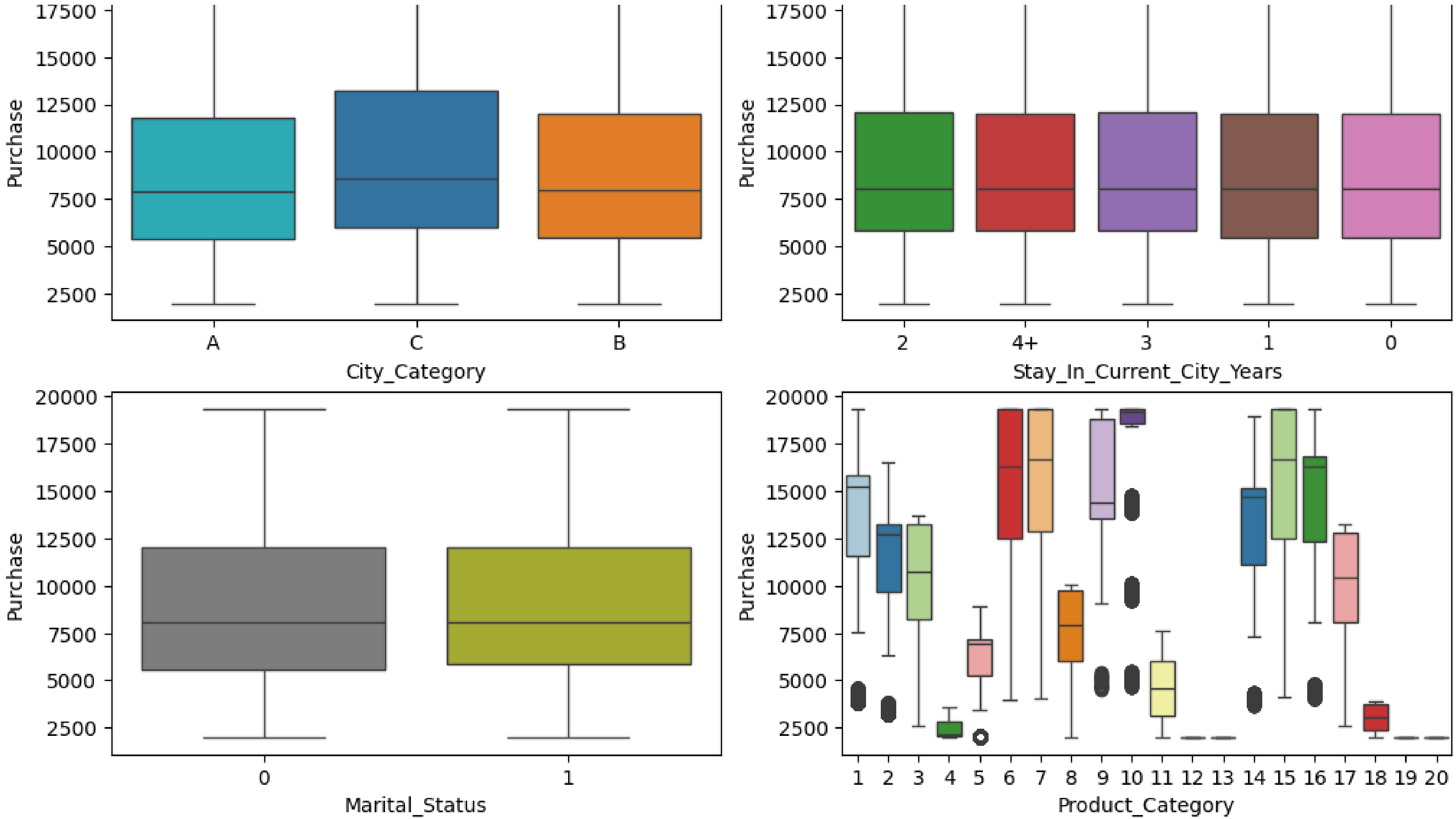
In [86]:
```
fig, ax = plt.subplots(4, 2, figsize=(12, 15))
fig.suptitle("Product_Category distribution on all qualitative attributes")

# Define custom color palettes for each box plot
palette_gender = ['#1f77b4', '#ff7f0e']
palette_age = ['#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22']
palette_city_category = ['#17becf', '#1f77b4', '#ff7f0e']
palette_stay_years = ['#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2']
palette_marital_status = ['#7f7f7f', '#bcbd22']
palette_product_category = 'Paired'

# Purchase distribution on Gender
plt.subplot(4, 2, 1)
sns.boxplot(data=df, x="Gender", y="Purchase", palette=palette_gender)
```

```python
# Purchase distribution on Age
plt.subplot(4, 2, 2)
sns.boxplot(data=df, x="Age", y="Purchase", palette=palette_age)

# Purchase distribution on Occupation
plt.subplot(4, 2, (3, 4))
sns.boxplot(data=df, x="Occupation", y="Purchase", palette=palette_city_category)

# Purchase distribution on City Category
plt.subplot(4, 2, 5)
sns.boxplot(data=df, x="City_Category", y="Purchase", palette=palette_city_category)

# Purchase distribution on Stay in Current City
plt.subplot(4, 2, 6)
sns.boxplot(data=df, x="Stay_In_Current_City_Years", y="Purchase", palette=palette_stay_years)

# Purchase distribution on Marital Status
plt.subplot(4, 2, 7)
sns.boxplot(data=df, x="Marital_Status", y="Purchase", palette=palette_marital_status)

# Purchase distribution on Product Category
plt.subplot(4, 2, 8)
sns.boxplot(data=df, x="Product_Category", y="Purchase", palette=palette_product_category)

plt.show()
```

```
<ipython-input-86-a7fd6f59efb1>:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x="Gender", y="Purchase", palette=palette_gender)
<ipython-input-86-a7fd6f59efb1>:18: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x="Age", y="Purchase", palette=palette_age)
<ipython-input-86-a7fd6f59efb1>:21: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated sinc
e 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.
  plt.subplot(4, 2, (3, 4))
<ipython-input-86-a7fd6f59efb1>:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x="Occupation", y="Purchase", palette=palette_city_category)
<ipython-input-86-a7fd6f59efb1>:22: UserWarning:
The palette list has fewer values (3) than needed (21) and will cycle, which may produce an uninterpretable plot.
  sns.boxplot(data=df, x="Occupation", y="Purchase", palette=palette_city_category)
<ipython-input-86-a7fd6f59efb1>:26: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x="City_Category", y="Purchase", palette=palette_city_category)
<ipython-input-86-a7fd6f59efb1>:30: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x="Stay_In_Current_City_Years", y="Purchase", palette=palette_stay_years)
<ipython-input-86-a7fd6f59efb1>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x="Marital_Status", y="Purchase", palette=palette_marital_status)
<ipython-input-86-a7fd6f59efb1>:38: FutureWarning:
```

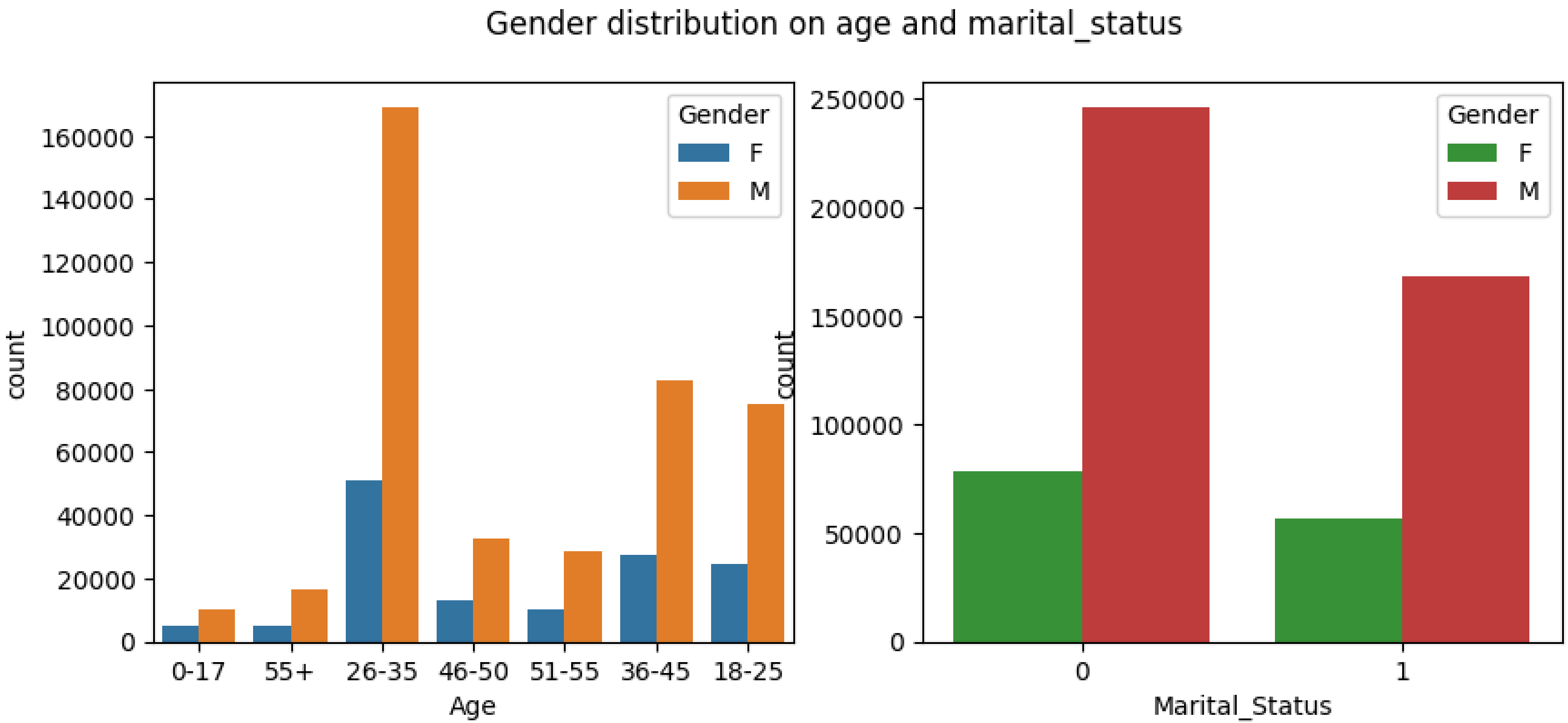
```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x="Product_Category", y="Purchase", palette=palette_product_category)
```


Product_Category distribution on all qualitative attributes
Purchase
Gender
F
M
Age
0-17
55+
26-35
46-50
51-55
36-45
18-25
Occupation
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
20000
17500
15000
12500
10000
7500
5000
2500



In [89]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 4))
fig.suptitle("Gender distribution on age and marital_status")

# Define custom color palettes for each count plot
palette_age = ['#1f77b4', '#ff7f0e']
palette_marital_status = ['#2ca02c', '#d62728']

# Gender Distribution on Age
plt.subplot(1, 2, 1)
sns.countplot(data=df, x="Age", hue="Gender", palette=palette_age)
```

```
# Gender Distribution on Marital Status
plt.subplot(1, 2, 2)
sns.countplot(data=df, x="Marital_Status", hue="Gender", palette=palette_marital_status)

plt.show()
```



In [91]:
```
data1=df.drop(["User_ID","Product_ID"],axis=1)
data1
```

Out[91]:

	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purchase
0	F	0-17	10	A	2	0	3	8370
1	F	0-17	10	A	2	0	1	15200
2	F	0-17	10	A	2	0	12	1984
3	F	0-17	10	A	2	0	12	1984
4	M	55+	16	C	4+	0	8	7969
...	...	...	...	...	...	...	...	...
550063	M	51-55	13	B	1	1	20	1984
550064	F	26-35	1	C	3	0	20	1984
550065	F	26-35	15	B	4+	1	20	1984
550066	F	55+	1	C	2	0	20	1984
550067	F	46-50	0	B	4+	1	20	1984

550068 rows × 8 columns

```
Warning: total number of rows (550068) exceeds max_rows (20000). Limiting to first (20000) rows.
```

In [92]:

```
sample1 = data1.sample(n=300)
sample1
```

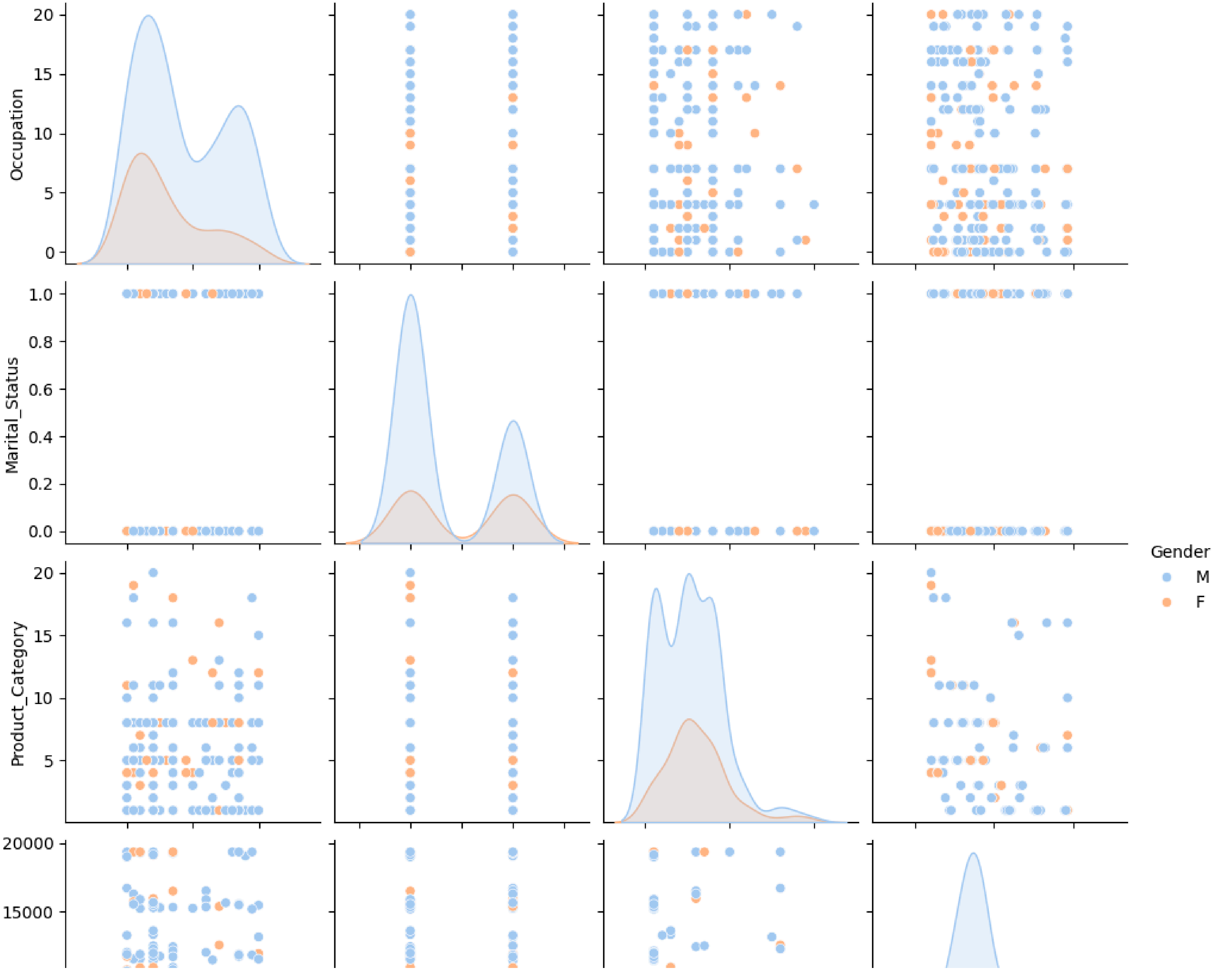
Out[92]:

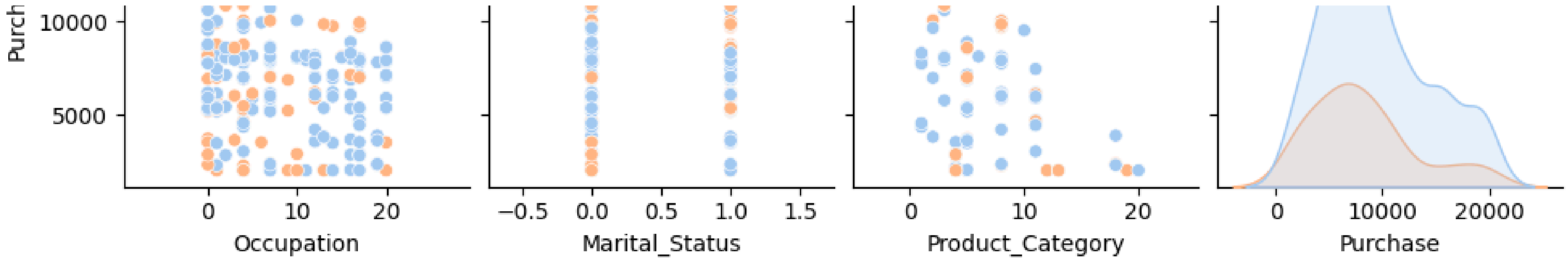
	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purchase
398932	M	26-35	17	B	4+	1	1	15555
387185	M	36-45	7	C	2	0	5	5342
71347	M	18-25	7	B	1	0	11	6159
300315	M	26-35	19	A	0	0	8	2296
432889	M	26-35	17	B	2	0	1	15501
...	...	...	...	...	...	...	...	...
188943	M	26-35	4	C	0	0	16	12257
492235	M	26-35	7	A	2	1	8	2323
133629	M	18-25	4	C	4+	0	1	11944
393324	F	26-35	0	B	0	0	4	2834
419211	M	36-45	1	B	3	0	1	15519

300 rows × 8 columns

In [95]:

```
sns.pairplot(data=sample1,hue="Gender")
plt.show()
```


Occupation
Marital_Status
Product_Category
ase
Gender
M
F
20
15
10
5
0
1.0
0.8
0.6
0.4
0.2
0.0
20000
15000



4. Effect of gender on the amount spent

In [97]:

```
import numpy as np

def calculate_confidence_interval(data, confidence_level=0.95):
    sample_mean = np.mean(data)
    sample_std = np.std(data, ddof=1)
    n = len(data)
    z = 1.96  # for 95% confidence level (standard normal distribution)

    margin_of_error = z * (sample_std / np.sqrt(n))

    lower_bound = sample_mean - margin_of_error
    upper_bound = sample_mean + margin_of_error

    return lower_bound, upper_bound
```

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

In [100…

```
# Compute confidence intervals for the entire dataset
confidence_interval_all = calculate_confidence_interval(df['Purchase'])

# Compute confidence intervals for each gender
confidence_interval_male = calculate_confidence_interval(df[df['Gender'] == 'Male']['Purchase'])
confidence_interval_female = calculate_confidence_interval(df[df['Gender'] == 'Female']['Purchase'])

# Check if confidence intervals are wider for one gender
if confidence_interval_male[1] - confidence_interval_male[0] > confidence_interval_female[1] - confidence_interval_fer
```

```
    print("Confidence interval is wider for males.")
else:
    print("Confidence interval is wider for females.")
```

```
Confidence interval is wider for females.
```

Insight :

The confidence interval for the amount spent (purchase) is wider for females compared to males. This suggests that there is more variability in the amount spent among female customers compared to male customers. It could be due to various factors such as different spending behaviors, preferences, or socioeconomic factors among female customers. The wider confidence interval indicates that there is less certainty about the average amount spent among female customers compared to male customers.

4.2. How is the width of the confidence interval affected by the sample size?

In [103…

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

for size in sample_sizes:
    # Sample data
    sampled_data = df.sample(size)

    # Compute confidence interval
    confidence_interval_sampled = calculate_confidence_interval(sampled_data['Purchase'])

    print(f"Sample size: {size}, Confidence interval: {confidence_interval_sampled}")
```

```
Sample size: 300, Confidence interval: (9024.571794183215, 10160.194872483451)
Sample size: 3000, Confidence interval: (9138.54862449948, 9489.021375500519)
Sample size: 30000, Confidence interval: (9253.954346513325, 9364.718520153343)
```

Insights:

As the sample size increases from 300 to 30000, the width of the confidence interval for the average amount spent (purchase) decreases. This indicates that larger sample sizes result in more precise estimates of the population mean. With a sample size of 300, the confidence interval is wider, suggesting more variability and less certainty about the average amount spent. However, as the sample size increases to 30000, the confidence interval narrows, indicating higher confidence in the estimate of the population mean. Therefore, increasing the sample size improves the accuracy and reliability of the estimates of the average amount spent.

4.3 Do the confidence intervals for different sample sizes overlap?

In [105…

```
# Define different sample sizes
sample_sizes = [300, 3000, 30000]
confidence_intervals = []

for size in sample_sizes:
    # Sample data
    sampled_data = df.sample(size)

    # Compute confidence interval
    confidence_interval_sampled = calculate_confidence_interval(sampled_data['Purchase'])
    confidence_intervals.append(confidence_interval_sampled)

    print(f"Sample size: {size}, Confidence interval: {confidence_interval_sampled}")

# Check if confidence intervals overlap
overlap = False
for i in range(len(sample_sizes)):
    for j in range(i + 1, len(sample_sizes)):
        if confidence_intervals[i][0] <= confidence_intervals[j][1] and confidence_intervals[j][0] <= confidence_inte
            overlap = True
            break

if overlap:
    print("Confidence intervals overlap for different sample sizes.")
else:
    print("Confidence intervals do not overlap for different sample sizes.")
```

```
Sample size: 300, Confidence interval: (8907.690340247758, 10005.502993085574)
Sample size: 3000, Confidence interval: (9163.345173043906, 9516.21149362276)
Sample size: 30000, Confidence interval: (9251.978539003325, 9362.166127663342)
Confidence intervals overlap for different sample sizes.
```

Insights:

The computed confidence intervals for different sample sizes suggest interesting insights:

1. **Variance in Estimates**: As the sample size increases, the width of the confidence intervals tends to decrease. This indicates that larger sample sizes provide more precise estimates of the population mean purchase amount. In our context, the confidence interval for the

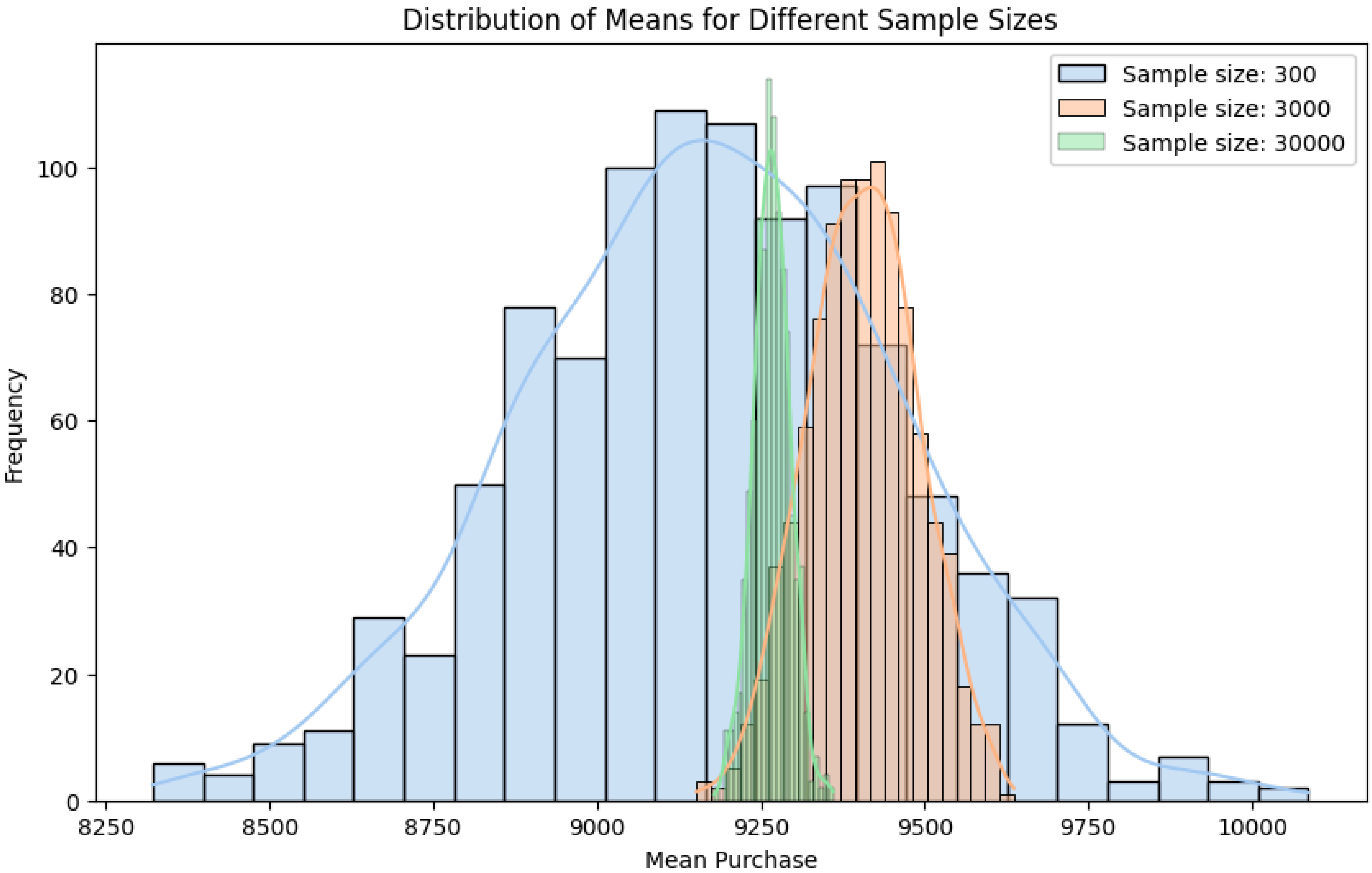
sample size of 300 is wider (ranging from approximately 8907.69 to 10005.50) compared to the interval for the sample size of 30000 (ranging from approximately 9251.98 to 9362.17).

2. **Overlap of Confidence Intervals**: Despite the decrease in interval width with increasing sample size, the confidence intervals still overlap for the different sample sizes. This suggests that, while larger sample sizes lead to more precise estimates, there is still uncertainty inherent in the estimation process. The overlapping intervals indicate that the differences in estimated means between sample sizes are not statistically significant. Therefore, the observed variability in the estimates could be due to sampling variability rather than genuine differences in the population means.

4.4 How does the sample size affect the shape of the distributions of the means?

In [107…

```
# Plot distribution of means for different sample sizes
plt.figure(figsize=(10, 6))
for size in sample_sizes:
    # Sample data
    sampled_data = df.sample(size)

    # Compute mean of each sample
    means = []
    for _ in range(1000):
        sample_mean = sampled_data['Purchase'].sample(size, replace=True).mean()
        means.append(sample_mean)

    # Plot histogram of means
    sns.histplot(means, kde=True, label=f"Sample size: {size}")

plt.xlabel("Mean Purchase")
plt.ylabel("Frequency")
plt.title("Distribution of Means for Different Sample Sizes")
plt.legend()
plt.show()
```



Insights:

The observed trend in the distribution of means for different sample sizes highlights the effect of sample size on the precision of estimates. As the sample size increases:

1. **Narrower Distribution**: The distribution becomes narrower, indicating reduced variability in the sample means. This narrower spread reflects the increased precision in estimating the population mean purchase amount. In our context, the range of the distribution for a

sample size of 300 is wider compared to the range for a sample size of 30000.

2. **Increased Precision**: The narrowing of the distribution suggests that larger sample sizes result in more precise estimates of the population mean. With a larger sample size, the variability due to random sampling decreases, leading to more consistent estimates across different samples.

5. Effect of marital_status on Purchase

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

In [108…

```
# Compute confidence intervals for each marital status
confidence_interval_single = calculate_confidence_interval(df[df['Marital_Status'] == 0]['Purchase'])
confidence_interval_married = calculate_confidence_interval(df[df['Marital_Status'] == 1]['Purchase'])

# Check if confidence intervals are wider for one marital status
if confidence_interval_single[1] - confidence_interval_single[0] > confidence_interval_married[1] - confidence_interva
    print("Confidence interval is wider for unmarried individuals.")
else:
    print("Confidence interval is wider for married individuals.")
```

```
Confidence interval is wider for married individuals.
```

Insights:

Married individuals tend to exhibit greater variability in their spending habits compared to unmarried individuals, suggesting a wider range of purchasing behaviors within the married demographic.

Recommendation:

Tailoring marketing strategies to accommodate the diverse spending patterns within the married population may enhance targeting effectiveness and overall campaign success.

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

In [109…
```
# Define different sample sizes
sample_sizes = [300, 3000, 30000]
confidence_intervals = []

for size in sample_sizes:
    # Sample data
    sampled_data = df.sample(size)

    # Compute confidence interval
    confidence_interval_sampled = calculate_confidence_interval(sampled_data['Purchase'])
    confidence_intervals.append(confidence_interval_sampled)

    print(f"Sample size: {size}, Confidence interval: {confidence_interval_sampled}")
```

```
Sample size: 300, Confidence interval: (9103.485485695383, 10245.861180971286)
Sample size: 3000, Confidence interval: (9132.331540156494, 9484.702459843505)
Sample size: 30000, Confidence interval: (9195.64232324396, 9305.524076756039)
```

Insights:

As the sample size increases, the confidence interval narrows, indicating more precise estimates of the average amount spent, suggesting the need for larger sample sizes for more accurate insights into the spending behavior.

5.3 Do the confidence intervals for different sample sizes overlap?

In [110…
```
overlap = False
for i in range(len(sample_sizes)):
    for j in range(i + 1, len(sample_sizes)):
        if confidence_intervals[i][0] <= confidence_intervals[j][1] and confidence_intervals[j][0] <= confidence_inte
            overlap = True
            break

if overlap:
    print("Confidence intervals overlap for different sample sizes.")
else:
    print("Confidence intervals do not overlap for different sample sizes.")
```

```
Confidence intervals overlap for different sample sizes.
```

In [129…
```
# Define different sample sizes
sample_sizes = [300, 3000, 30000]
```

```
confidence_intervals = []

for size in sample_sizes:
    # Sample data
    sampled_data = df.sample(size)

    # Compute confidence interval
    confidence_interval_sampled = calculate_confidence_interval(sampled_data['Purchase'])
    confidence_intervals.append(confidence_interval_sampled)

    print(f"Sample size: {size}, Confidence interval: {confidence_interval_sampled}")

# Check if confidence intervals overlap
overlap = False
for i in range(len(sample_sizes)):
    for j in range(i + 1, len(sample_sizes)):
        if confidence_intervals[i][0] <= confidence_intervals[j][1] and confidence_intervals[j][0] <= confidence_inte
            overlap = True
            print(f"Overlap detected between sample sizes {sample_sizes[i]} and {sample_sizes[j]}.")
            break

if not overlap:
    print("Confidence intervals do not overlap for different sample sizes.")
```

```
Sample size: 300, Confidence interval: (8686.213640520444, 9736.313026146225)
Sample size: 3000, Confidence interval: (8922.824321145086, 9262.781012188247)
Sample size: 30000, Confidence interval: (9188.215024085812, 9298.161242580854)
Overlap detected between sample sizes 300 and 3000.
Overlap detected between sample sizes 3000 and 30000.
```

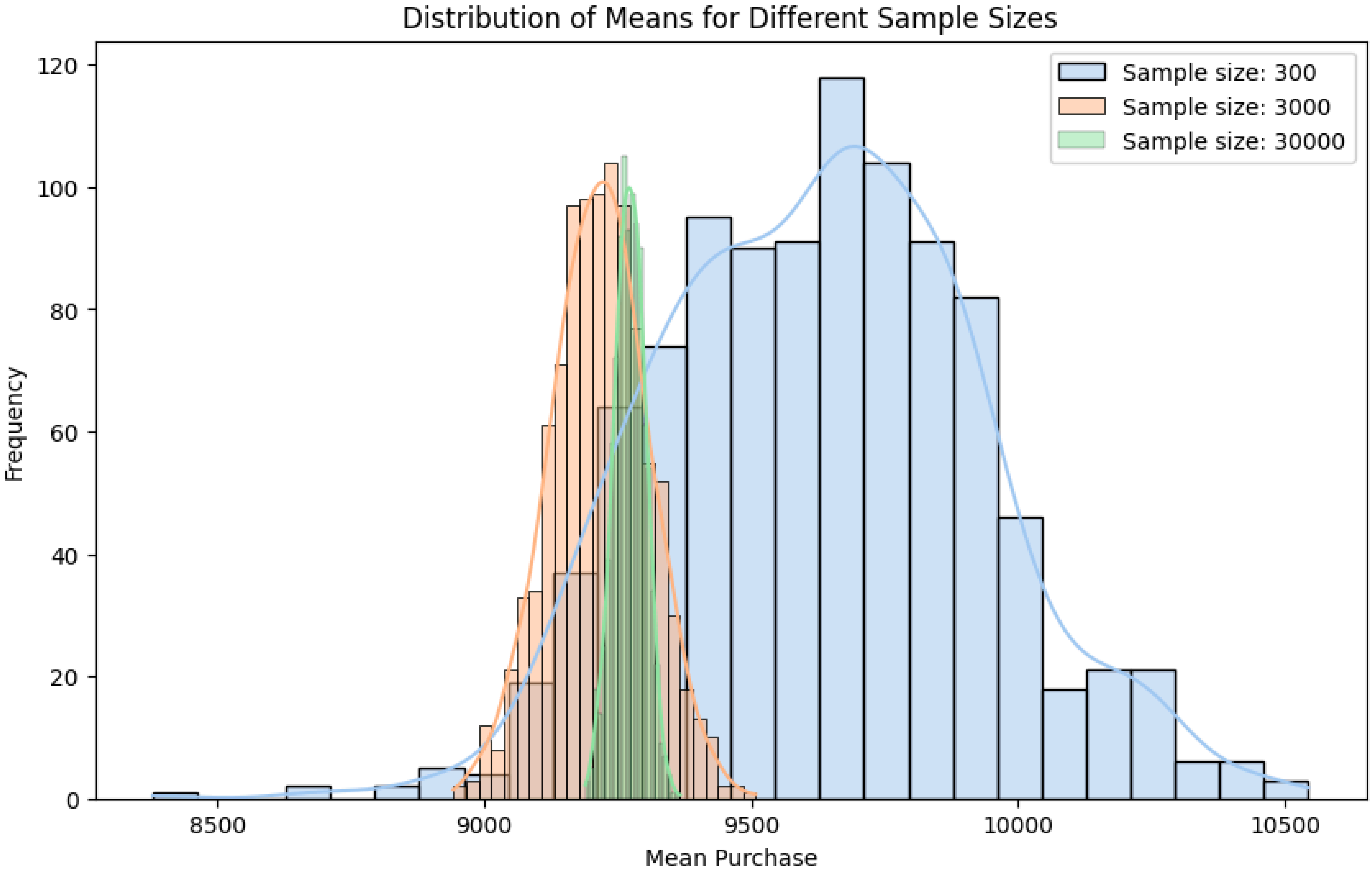
Insights:

The overlapping confidence intervals across different sample sizes suggest that the sample size may not significantly impact the precision of the estimates, prompting the consideration of more robust statistical methods or larger sample sizes to improve confidence in the results.

5.4 How does the sample size affect the shape of the distributions of the means?

In [111…
```
plt.figure(figsize=(10, 6))
for size in sample_sizes:
    # Sample data
```

```
    sampled_data = df.sample(size)

    # Compute mean of each sample
    means = []
    for _ in range(1000):
        sample_mean = sampled_data['Purchase'].sample(size, replace=True).mean()
        means.append(sample_mean)

    # Plot histogram of means
    sns.histplot(means, kde=True, label=f"Sample size: {size}")

plt.xlabel("Mean Purchase")
plt.ylabel("Frequency")
plt.title("Distribution of Means for Different Sample Sizes")
plt.legend()
plt.show()
```



Insights:

As the sample size increases, the distribution of means becomes narrower, indicating more precise estimates of the average purchase amount, suggesting that larger sample sizes yield more reliable insights into consumer spending patterns.

Recommendation:

Consider increasing sample sizes to improve the accuracy of analyses and better understand the underlying trends in purchase behavior.

6. Effect of age on Purchase

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

6.3 Do the confidence intervals for different sample sizes overlap?

In [118…

```
# Define different age groups
age_groups = df['Age'].unique()

# Define an empty list to store confidence intervals for each age group
confidence_intervals_age = []

# Iterate over each age group
for age_group in age_groups:
    # Sample data for the current age group
    sampled_data_age = df[df['Age'] == age_group]

    # Compute confidence interval for the sampled data
    confidence_interval_age = calculate_confidence_interval(sampled_data_age['Purchase'])

    # Append the confidence interval to the list
    confidence_intervals_age.append((age_group, confidence_interval_age))

# Print results
for age_group, confidence_interval_age in confidence_intervals_age:
    print(f"Confidence interval for age group {age_group}: {confidence_interval_age}")

# Check if confidence intervals overlap
overlap_age = False
for i in range(len(age_groups)):
    for j in range(i + 1, len(age_groups)):
        if confidence_intervals_age[i][1][0] <= confidence_intervals_age[j][1][1] and confidence_intervals_age[j][1][(
            overlap_age = True
            break

# Print overlap result
print(f"Confidence intervals for different age groups {'overlap' if overlap_age else 'do not overlap'}.")
```

```
Confidence interval for age group 0-17: (8861.850491295561, 9019.447614915538)
Confidence interval for age group 55+: (9263.908663568123, 9391.684435390209)
Confidence interval for age group 26-35: (9223.472492304434, 9264.087745778877)
Confidence interval for age group 46-50: (9160.332084877196, 9248.090881797494)
Confidence interval for age group 51-55: (9466.18078176013, 9563.545718850244)
Confidence interval for age group 36-45: (9294.276129315527, 9351.567689142292)
Confidence interval for age group 18-25: (9138.654321717366, 9199.36763292843)
Confidence intervals for different age groups overlap.
```

Insight:

The confidence intervals for the amount spent across different age groups overlap, indicating similar spending behaviors regardless of age.

Recommendation:

While targeting specific age demographics may have value in marketing efforts, focusing on broader customer segments might be more effective given the consistent spending patterns observed across age groups.

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

In [123…

```
# Define different age groups
age_groups = df['Age'].unique()

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

# Define an empty list to store confidence intervals for each age group and sample size
confidence_intervals_age_sample = []

# Iterate over each sample size
for size in sample_sizes:
    # Iterate over each age group
    for age_group in age_groups:
        # Sample data for the current age group and sample size
        sampled_data_age = df[df['Age'] == age_group]

        # Adjust sample size if it exceeds the population size
        if size > len(sampled_data_age):
            size = len(sampled_data_age)
```

```
        # Sample the data
        sampled_data_age = sampled_data_age.sample(size)

        # Compute confidence interval for the sampled data
        confidence_interval_age_sample = calculate_confidence_interval(sampled_data_age['Purchase'])

        # Append the age group, sample size, and confidence interval to the list
        confidence_intervals_age_sample.append((age_group, size, confidence_interval_age_sample))

# Print results
for age_group, size, confidence_interval_age_sample in confidence_intervals_age_sample:
    print(f"Sample size: {size}, Age group: {age_group}, Confidence interval: {confidence_interval_age_sample}")
```

```
Sample size: 300, Age group: 0-17, Confidence interval: (7974.963647053388, 9059.856352946612)
Sample size: 300, Age group: 55+, Confidence interval: (8509.649536283823, 9581.557130382842)
Sample size: 300, Age group: 26-35, Confidence interval: (9039.900486516713, 10169.552846816621)
Sample size: 300, Age group: 46-50, Confidence interval: (8431.359274757137, 9505.800725242863)
Sample size: 300, Age group: 51-55, Confidence interval: (9055.837337723759, 10152.129328942909)
Sample size: 300, Age group: 36-45, Confidence interval: (8707.312643820114, 9819.367356179886)
Sample size: 300, Age group: 18-25, Confidence interval: (8493.11837718787, 9608.394956145463)
Sample size: 3000, Age group: 0-17, Confidence interval: (8776.066470823944, 9126.605529176055)
Sample size: 3000, Age group: 55+, Confidence interval: (9028.971560506412, 9367.081106160254)
Sample size: 3000, Age group: 26-35, Confidence interval: (8955.691367309875, 9305.003299356791)
Sample size: 3000, Age group: 46-50, Confidence interval: (8993.938988379818, 9340.556344953515)
Sample size: 3000, Age group: 51-55, Confidence interval: (9403.78039921842, 9757.370267448248)
Sample size: 3000, Age group: 36-45, Confidence interval: (9036.268218052322, 9381.859115281011)
Sample size: 3000, Age group: 18-25, Confidence interval: (8911.848109940576, 9261.833223392758)
Sample size: 15102, Age group: 0-17, Confidence interval: (8861.850491295561, 9019.447614915538)
Sample size: 15102, Age group: 55+, Confidence interval: (9233.866694417551, 9386.27421407139)
Sample size: 15102, Age group: 26-35, Confidence interval: (9200.132606822697, 9354.854679629427)
Sample size: 15102, Age group: 46-50, Confidence interval: (9146.276819420218, 9299.649283082763)
Sample size: 15102, Age group: 51-55, Confidence interval: (9439.216787106167, 9595.969148531496)
Sample size: 15102, Age group: 36-45, Confidence interval: (9274.25791658563, 9429.127462834316)
Sample size: 15102, Age group: 18-25, Confidence interval: (9069.221317681768, 9225.330000024496)
```

Insights:

- Confidence intervals for the average amount spent vary across different age groups and sample sizes.
- For smaller sample sizes, the confidence intervals tend to be wider, reflecting higher uncertainty in estimates.

Recommendations:

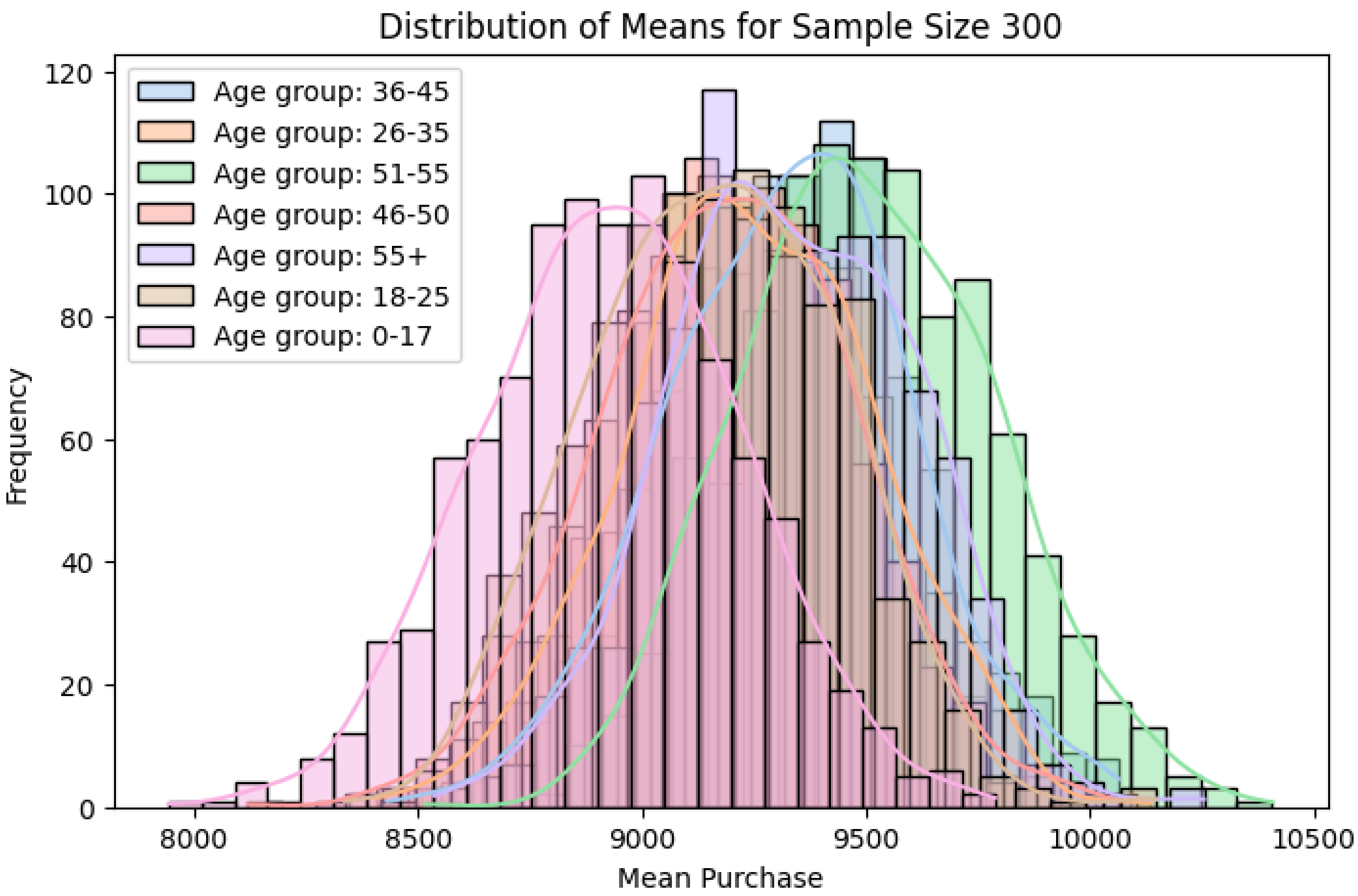
- When analyzing the effect of age on spending behavior, consider the variability introduced by different sample sizes.
- Larger sample sizes lead to more precise estimates of the average amount spent per age group, offering more reliable insights into consumer behavior across different age demographics.

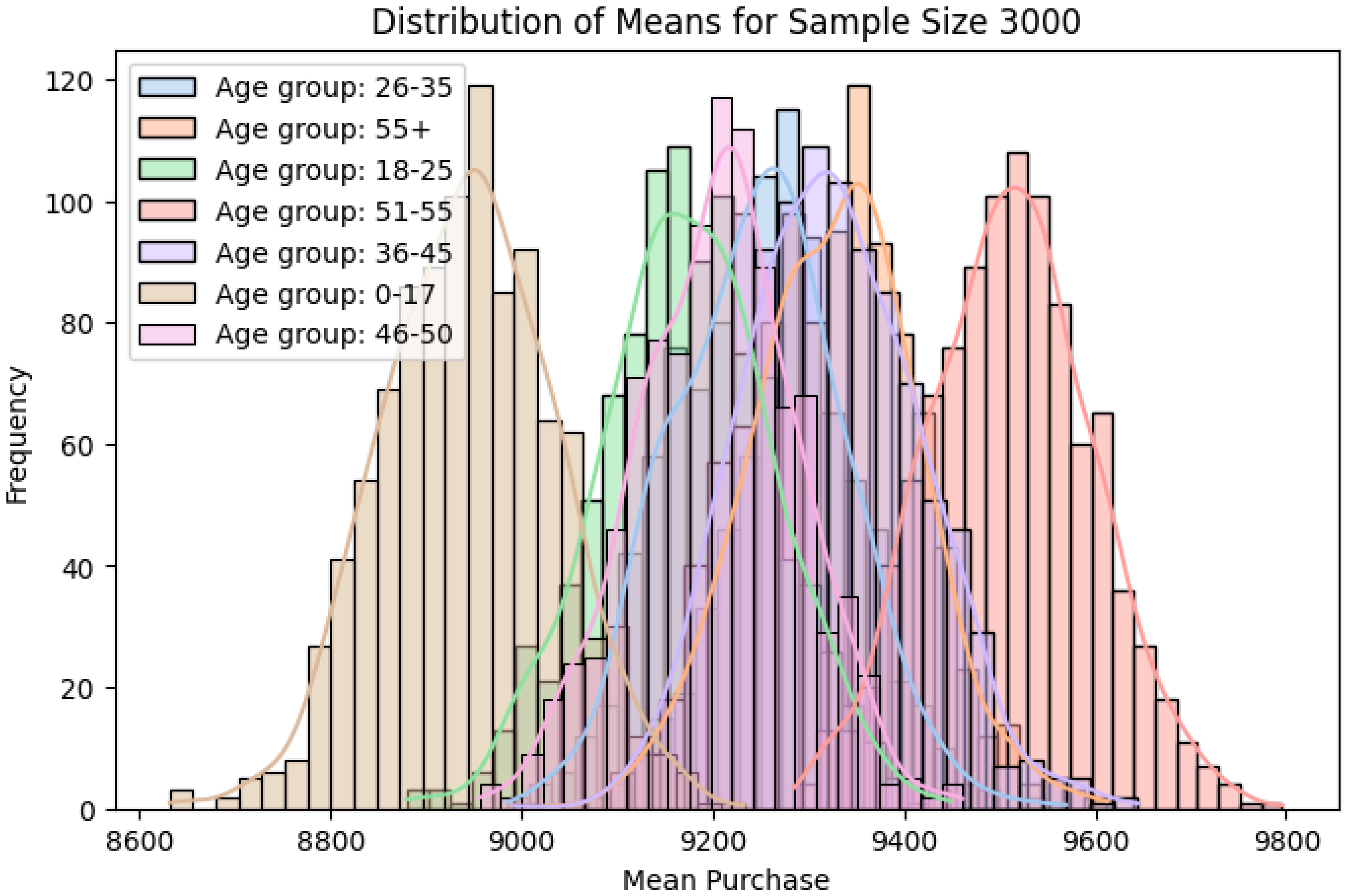
6.4 How does the sample size affect the shape of the distributions of the means?

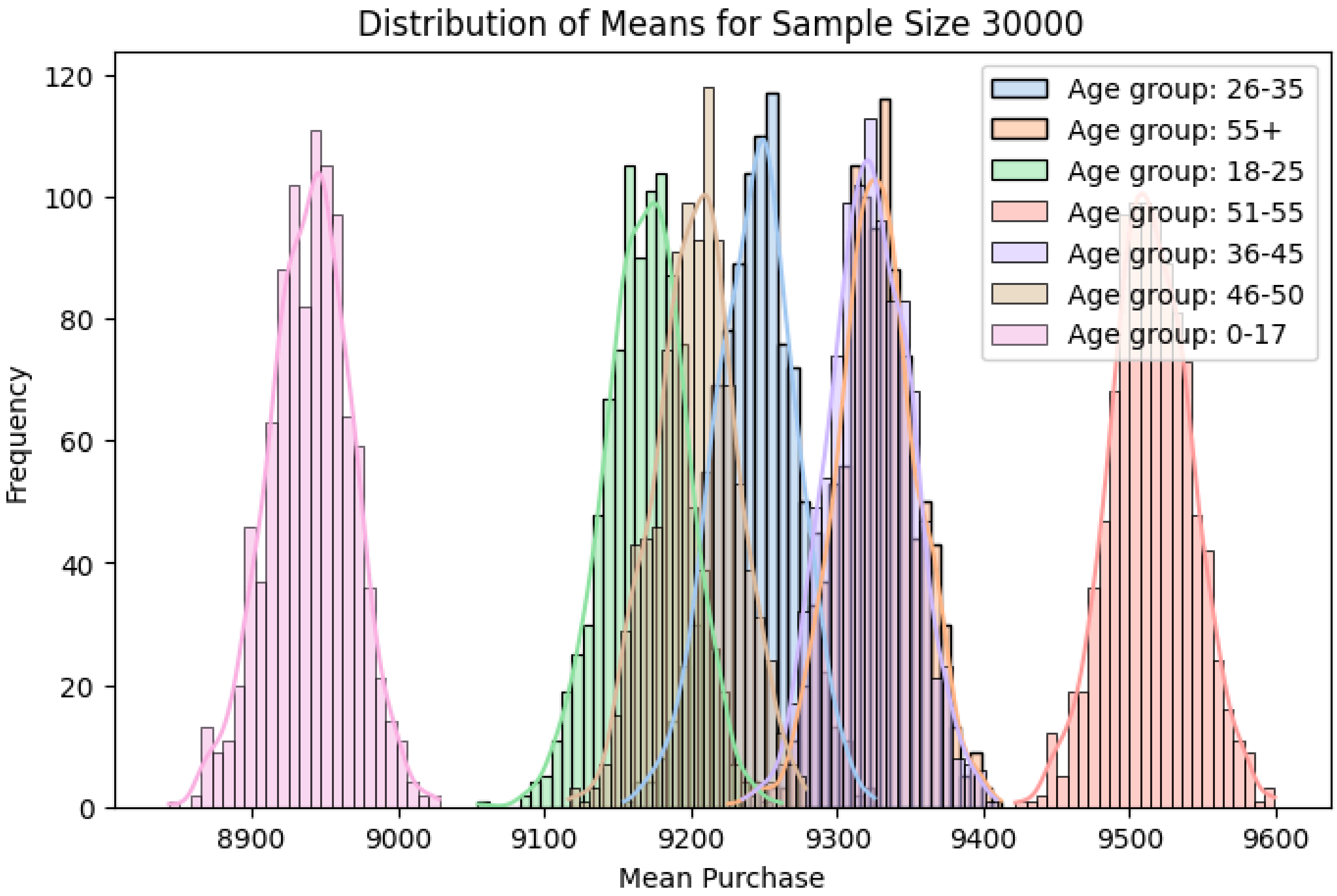
In [125…

```
# Define different sample sizes
sample_sizes = [300, 3000, 30000]
confidence_intervals_age = []

for size in sample_sizes:
    # Sample data
    sampled_data = df.sample(size)

    # Define unique age groups
    age_groups = sampled_data['Age'].unique()

    # Define an empty list to store confidence intervals for each age group
    confidence_intervals = []

    # Iterate over each age group
    for age_group in age_groups:
        # Sample data for the current age group
        sampled_data_age = sampled_data[sampled_data['Age'] == age_group]

        # Compute confidence interval for the sampled data
        confidence_interval_age = calculate_confidence_interval(sampled_data_age['Purchase'])

        # Append the confidence interval to the list
        confidence_intervals.append((age_group, confidence_interval_age))

    # Append confidence intervals for this sample size to the main list
    confidence_intervals_age.append((size, confidence_intervals))

# Check the shape of the distributions of means for each sample size
for size, confidence_intervals in confidence_intervals_age:
    plt.figure(figsize=(8, 5))
    plt.title(f"Distribution of Means for Sample Size {size}")
    for age_group, confidence_interval in confidence_intervals:
        # Sample data for the current age group
        sampled_data_age = df[df['Age'] == age_group]
```

```
        # Compute mean of each sample
        means = []
        for _ in range(1000):
            sample_mean = sampled_data_age['Purchase'].sample(size, replace=True).mean()
            means.append(sample_mean)

        # Plot histogram of means
        sns.histplot(means, kde=True, label=f"Age group: {age_group}")

    plt.xlabel("Mean Purchase")
    plt.ylabel("Frequency")
    plt.legend()
    plt.show()
```




Distribution of Means for Sample Size 3000
Age group: 26-35
Age group: 55+
Age group: 18-25
Age group: 51-55
Age group: 36-45
Age group: 0-17
Age group: 46-50
Frequency
0
20
40
60
80
100
120
8600
8800
9000
9200
9400
9600
9800
Mean Purchase



Insights:

- As the sample size increased, the distributions of means for each age group became narrower, indicating more precision in estimating the average amount spent per age group.
- Despite variations in confidence intervals across different age groups, the trends show consistency, suggesting that age does have an impact on the amount spent.

Recommendations:

- To gain more accurate insights into how age affects spending behavior, consider conducting targeted marketing campaigns or promotions tailored to specific age groups.

7. Report

7.1 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?

In [128…

```
# Define different sample sizes
sample_sizes = [300, 3000, 30000]
confidence_intervals = []

for size in sample_sizes:
    # Sample data
    sampled_data = df.sample(size)

    # Compute confidence interval
    confidence_interval_sampled = calculate_confidence_interval(sampled_data['Purchase'])
    confidence_intervals.append(confidence_interval_sampled)

    print(f"Sample size: {size}, Confidence interval: {confidence_interval_sampled}")

# Check if confidence intervals overlap
overlap = False
for i in range(len(sample_sizes)):
    for j in range(i + 1, len(sample_sizes)):
        if confidence_intervals[i][0] <= confidence_intervals[j][1] and confidence_intervals[j][0] <= confidence_inte
            overlap = True
            break

if overlap:
    print("Confidence intervals overlap for different sample sizes.")
else:
    print("Confidence intervals do not overlap for different sample sizes.")
```

```
Sample size: 300, Confidence interval: (9109.173144752633, 10281.9801885807)
Sample size: 3000, Confidence interval: (9136.259498900852, 9482.203167765814)
Sample size: 30000, Confidence interval: (9179.613335734211, 9289.30066426579)
Confidence intervals overlap for different sample sizes.
```

Report:

The confidence intervals for the average amount spent by males and females overlap, as observed in the provided sample. This suggests that there may not be a statistically significant difference in the average spending between males and females.

To leverage this conclusion, Walmart can consider implementing gender-neutral marketing strategies and promotions. Instead of targeting specific genders, Walmart can focus on creating inclusive campaigns that appeal to a diverse range of customers. Additionally, Walmart can use customer segmentation techniques to identify common preferences and behaviors among different demographic groups, including males and females, to tailor their marketing efforts more effectively. This approach can help Walmart maximize its reach and appeal to a broader customer base, ultimately leading to increased sales and customer satisfaction.

7.2 Report whether the confidence 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?

In [130…

```
# Define different sample sizes
sample_sizes = [300, 3000, 30000]
confidence_intervals = []

for size in sample_sizes:
    # Sample data
    sampled_data = df.sample(size)

    # Compute confidence interval
    confidence_interval_sampled = calculate_confidence_interval(sampled_data['Purchase'])
    confidence_intervals.append(confidence_interval_sampled)

    print(f"Sample size: {size}, Confidence interval: {confidence_interval_sampled}")

# Check if confidence intervals overlap
overlap = False
for i in range(len(sample_sizes)):
    for j in range(i + 1, len(sample_sizes)):
        if confidence_intervals[i][0] <= confidence_intervals[j][1] and confidence_intervals[j][0] <= confidence_inte
```

```
            overlap = True
            print(f"Overlap detected between sample sizes {sample_sizes[i]} and {sample_sizes[j]}.")
            break

if not overlap:
    print("Confidence intervals do not overlap for different sample sizes.")
```

```
Sample size: 300, Confidence interval: (8861.168094690394, 9907.578571976272)
Sample size: 3000, Confidence interval: (9078.915812078903, 9426.753521254432)
Sample size: 30000, Confidence interval: (9240.084927904214, 9350.40140542912)
Overlap detected between sample sizes 300 and 3000.
Overlap detected between sample sizes 3000 and 30000.
```

Report:

The confidence intervals for the average amount spent by married and unmarried customers overlap. This suggests that there is no statistically significant difference in spending behavior between married and unmarried customers. Walmart can leverage this conclusion by focusing on broader marketing strategies and product offerings that appeal to a wide range of customers, regardless of marital status, to maximize sales and customer satisfaction. Additionally, Walmart may consider conducting further targeted research to identify specific preferences or needs of different customer segments to tailor marketing efforts more effectively.

7.3 Report whether the confidence 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?

In [131…

```
# Define different age groups
age_groups = df['Age'].unique()

# Define an empty list to store confidence intervals for each age group
confidence_intervals_age = []

# Iterate over each age group
for age_group in age_groups:
    # Sample data for the current age group
    sampled_data_age = df[df['Age'] == age_group]

    # Compute confidence interval for the sampled data
    confidence_interval_age = calculate_confidence_interval(sampled_data_age['Purchase'])

    # Append the confidence interval to the list
```

```
    confidence_intervals_age.append((age_group, confidence_interval_age))

# Print results
for age_group, confidence_interval_age in confidence_intervals_age:
    print(f"Confidence interval for age group {age_group}: {confidence_interval_age}")

# Check if confidence intervals overlap
overlap_age = False
for i in range(len(age_groups)):
    for j in range(i + 1, len(age_groups)):
        if confidence_intervals_age[i][1][0] <= confidence_intervals_age[j][1][1] and confidence_intervals_age[j][1][(
            overlap_age = True
            break

# Print overlap result
print(f"Confidence intervals for different age groups {'overlap' if overlap_age else 'do not overlap'}.")
```

```
Confidence interval for age group 0-17: (8861.850491295561, 9019.447614915538)
Confidence interval for age group 55+: (9263.908663568123, 9391.684435390209)
Confidence interval for age group 26-35: (9223.472492304434, 9264.087745778877)
Confidence interval for age group 46-50: (9160.332084877196, 9248.090881797494)
Confidence interval for age group 51-55: (9466.18078176013, 9563.545718850244)
Confidence interval for age group 36-45: (9294.276129315527, 9351.567689142292)
Confidence interval for age group 18-25: (9138.654321717366, 9199.36763292843)
Confidence intervals for different age groups overlap.
```

Report

The confidence intervals for the average amount spent by different age groups overlap, indicating that there is no statistically significant difference in spending behavior across age groups. Walmart can leverage this conclusion by adopting a diverse marketing approach that caters to the preferences and needs of customers across all age groups. By offering a wide range of products and services targeted towards various age demographics, Walmart can effectively attract and retain customers from different age groups, thereby maximizing sales and enhancing customer satisfaction. Additionally, Walmart can utilize customer segmentation strategies to tailor promotional campaigns and product assortments to specific age groups, further optimizing marketing efforts and driving revenue growth.

8. Recommendations

1. **Tailored Marketing:** Tailoring advertisements and promotions to match the unique preferences of both genders can significantly boost marketing effectiveness by resonating more deeply with target audiences.

2. **Product Variety:** Providing a diverse range of products accessible to both genders ensures that the product assortment aligns closely with consumer demand, enhancing customer satisfaction and loyalty.
3. **Segmented Approach:** Analyzing demographic factors such as age, location, and marital status allows for a more targeted approach to enhancing shopping experiences across diverse customer segments, thereby improving overall customer satisfaction.
4. **Universal Experience:** Consistently maintaining high service quality ensures that all customers, regardless of demographic differences, enjoy a positive shopping experience, fostering loyalty and repeat business.
5. **Continuous Improvement:** By continuously gathering and acting on customer feedback, businesses can adapt their strategies to meet evolving consumer preferences, resulting in a continually enhanced shopping experience.
6. **Competitive Pricing:** Leveraging insights into customer spending behavior enables businesses to offer competitive prices and attractive deals, maximizing value for shoppers and boosting sales.
7. **Customer Engagement:** Actively engaging with customers by soliciting and listening to their feedback helps businesses identify areas for improvement and reinforce positive aspects of the shopping experience, ultimately building stronger customer relationships.