Walmart Business case study

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

```
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/293/original/walmart_data.csv?1641285094
```

```
--2024-11-26 12:01:30--  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/293/original/walmart_data.csv?1641285094
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 18.64.229.71, 18.64.229.172, 18.64.229.135, ...
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|18.64.229.71|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23027994 (22M) [text/plain]
Saving to: ‘walmart_data.csv?1641285094’

walmart_data.csv?16 100%[===================>]  21.96M  9.83MB/s    in 2.2s

2024-11-26 12:01:33 (9.83 MB/s) - ‘walmart_data.csv?1641285094’ saved [23027994/23027994]
```

```
import pandas as pd
```

```
df=pd.read_csv("walmart_data.csv?1641285094")
```

```
df.head()
```

	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

```
df.dtypes
```

	0
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

```
df.shape
```

```
(550068, 10)
```

```
#Number of rows: 550,068
#Number of columns: 10
```

```
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

Start coding or generate with AI.

```
df.nunique()
```

	0
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

```
Start coding or generate with AI.
```

```
#Checkng for the missing values
```

```
df.isna().sum()
```

	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

```
#There is a no missing values n the dataset.
```

```
df.head()
```

	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

```
cat_cols = ['Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category']

df[cat_cols].melt().groupby(['variable', 'value'])[['value']].count()*100/len(df)
```

		value
variable	value	
Age	0-17	2.745479
	18-25	18.117760
	26-35	39.919974
	36-45	19.999891
	46-50	8.308246
	51-55	6.999316
	55+	3.909335
City_Category	A	26.854862
	B	42.026259
	C	31.118880
Gender	F	24.689493
	M	75.310507
Marital_Status	0	59.034701
	1	40.965299
Occupation	0	12.659889
	1	8.621843
	2	4.833584
	3	3.208694
	4	13.145284
	5	2.213726
	6	3.700452
	7	10.750125
	8	0.281056
	9	1.143677
	10	2.350618

	11	2.106285
	12	5.668208
	13	1.404917
	14	4.964659
	15	2.211545
	16	4.612339
	17	7.279645
	18	1.203851
	19	1.538173
	20	6.101427
Product_Category	**1**	25.520118
	2	4.338373
	3	3.674637
	4	2.136645
	5	27.438971
	6	3.720631
	7	0.676462
	8	20.711076
	9	0.074536
	10	0.931703
	11	4.415272
	12	0.717548
	13	1.008784
	14	0.276875
	15	1.143495
	16	1.786688
	17	0.105078

	18	0.568112
	19	0.291419
	20	0.463579
Stay_In_Current_City_Years	**0**	13.525237
	1	35.235825
	2	18.513711
	3	17.322404
	4+	15.402823

Observations :

1. 75% are male and 25% are female.
2. 60% are single and 40% are married.
3. ~ 80% of the users are between the age 18-50.
4. 35% Staying in the city from 1 year, 18% from 2 years, 17% from 3 year.
5. Total of 20 product categories are there
6. There are 20 differnent types of occupations in the city.

```
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
```

```
Start coding or generate with AI.
```

```
#coverting some columns to category
```

```
df["Marital_Status"]=df["Marital_Status"].astype("category")
df["Product_Category"]=df["Product_Category"].astype("category")
df["Gender"]=df["Gender"].astype("category")
df["Age"]=df["Age"].astype("category")
df["City_Category"]=df["City_Category"].astype("category")
df["Stay_In_Current_City_Years"]=df["Stay_In_Current_City_Years"].astype("category")
df["Occupation"]=df["Occupation"].astype("category")
```

```
df.dtypes
```

	0
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

```
Start coding or generate with AI.
```

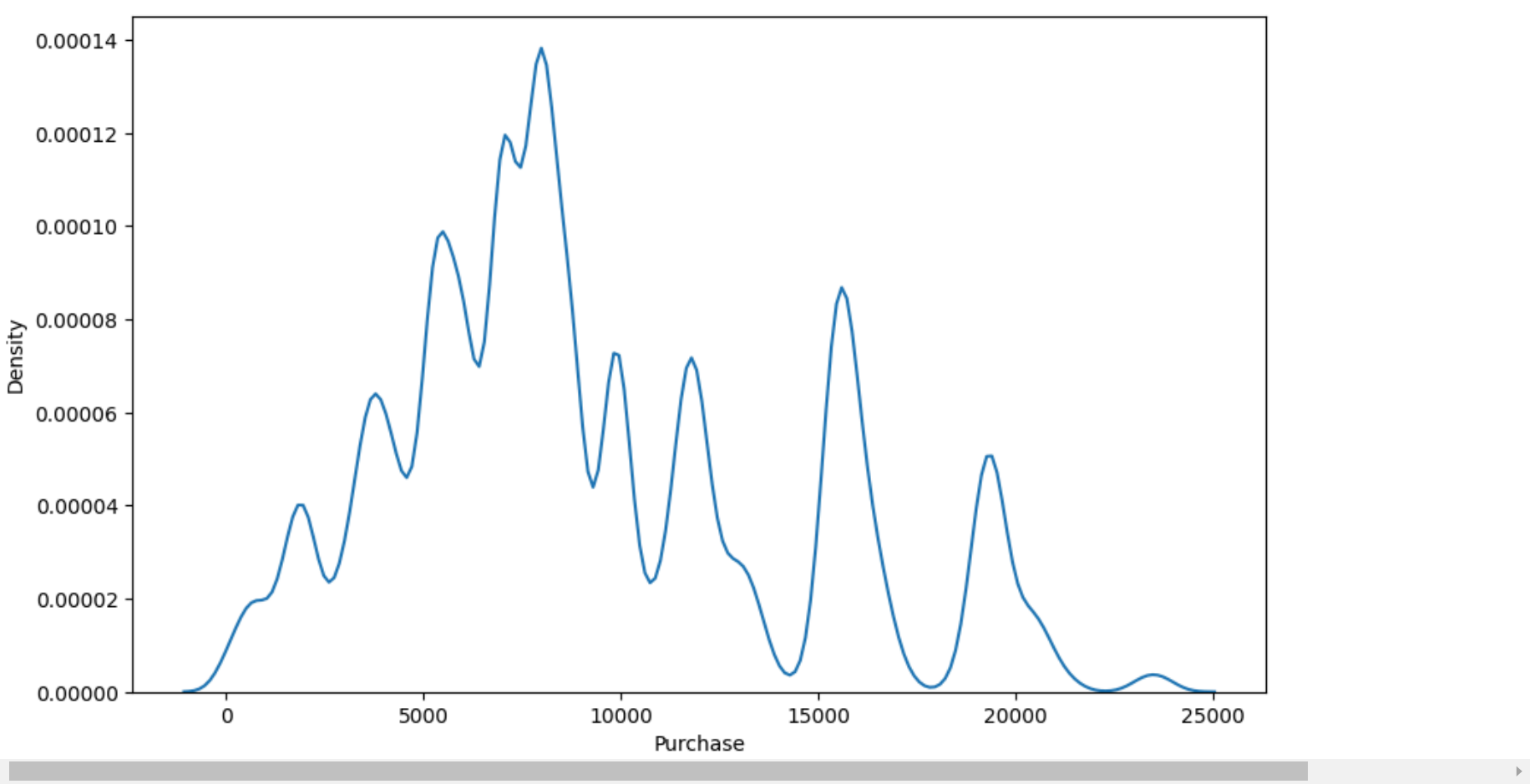
```
for item in df.select_dtypes(exclude="int64").columns:
  print(f'count of values of unique attributes in {item}: ')
  print(df[item].value_counts())
  print("..........................................................................................")
  print("..........................................................................................")
```

```
count of values of unique attributes in Product_ID:
Product_ID
P00265242    1880
P00025442    1615
P00110742    1612
P00112142    1562
P00057642    1470
             ...
P00314842       1
```

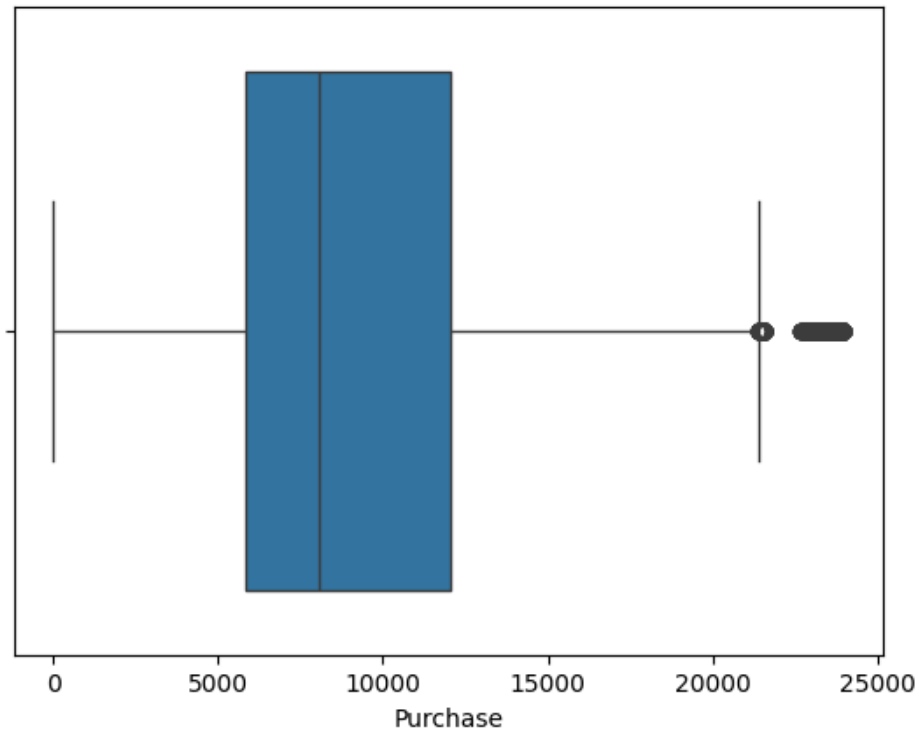
```
P00298842       1
P00231642       1
P00204442       1
P00066342       1
Name: count, Length: 3631, dtype: int64
..................................................................................................
..................................................................................................
count of values of unique attributes in Gender:
Gender
M    414259
F    135809
Name: count, dtype: int64
..................................................................................................
..................................................................................................
count of values of unique attributes in Age:
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
..................................................................................................
..................................................................................................
count of values of unique attributes in City_Category:
City_Category
B    231173
C    171175
A    147720
Name: count, dtype: int64
..................................................................................................
..................................................................................................
count of values of unique attributes in Stay_In_Current_City_Years:
Stay_In_Current_City_Years
1     193821
2     101838
3      95285
4+     84726
0      74398
Name: count, dtype: int64
..................................................................................................
..................................................................................................
```

#Visual Analysis

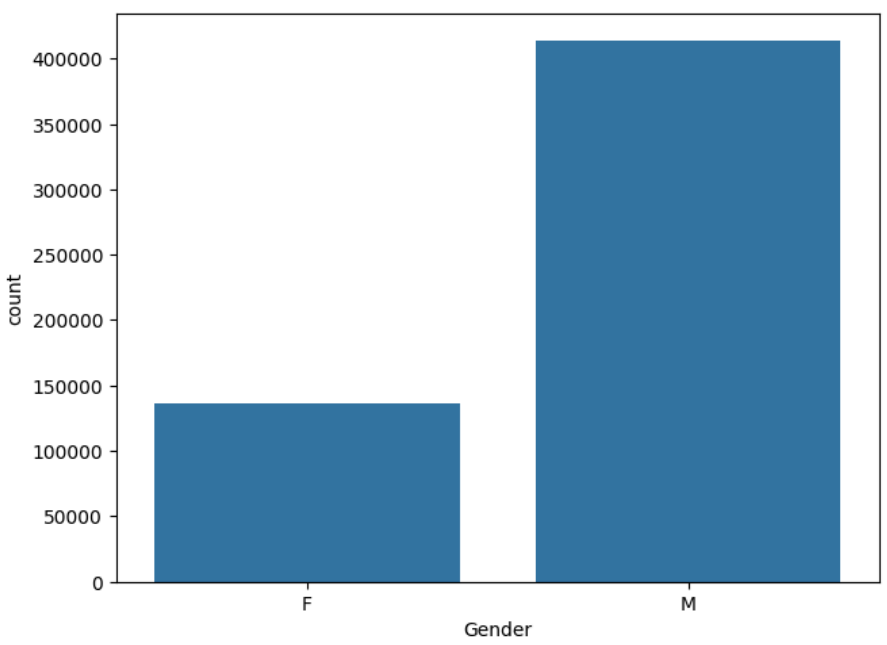
```
#Distribution of purchase


import matplotlib.pyplot as plt
import seaborn as sns


plt.figure(figsize=(10, 6))
sns.kdeplot(data=df, x="Purchase")
plt.show()
```

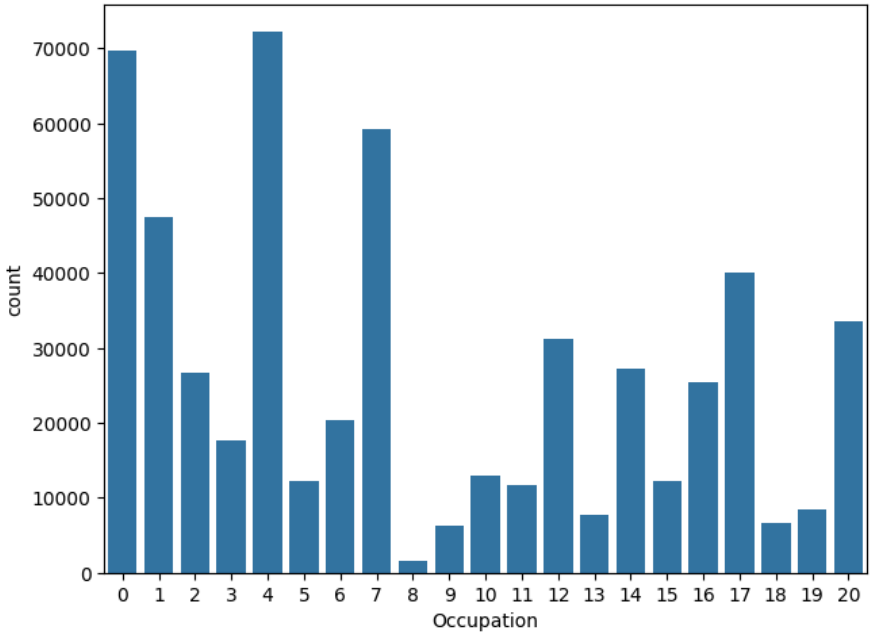


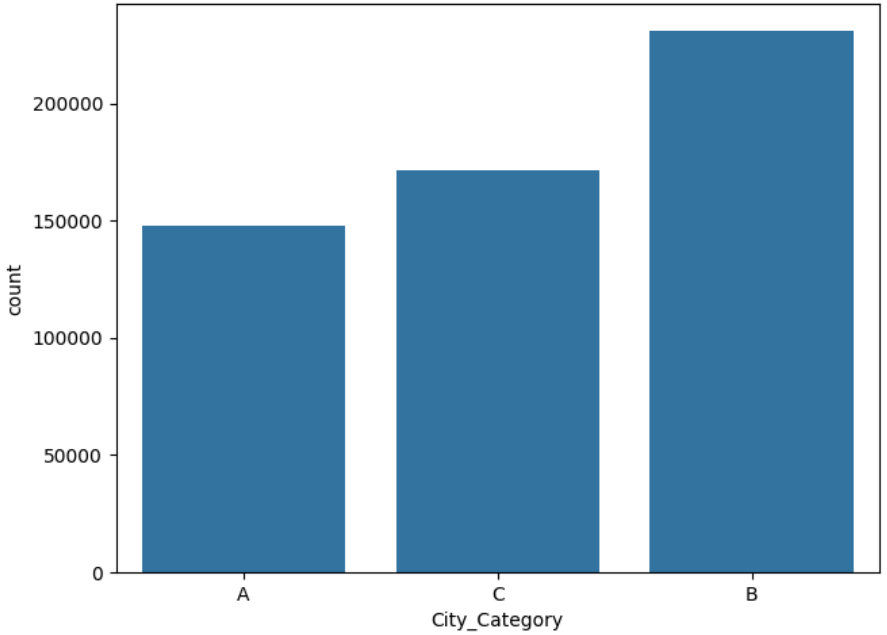

```
sns.boxplot(data=df,x="Purchase")
plt.show()
```

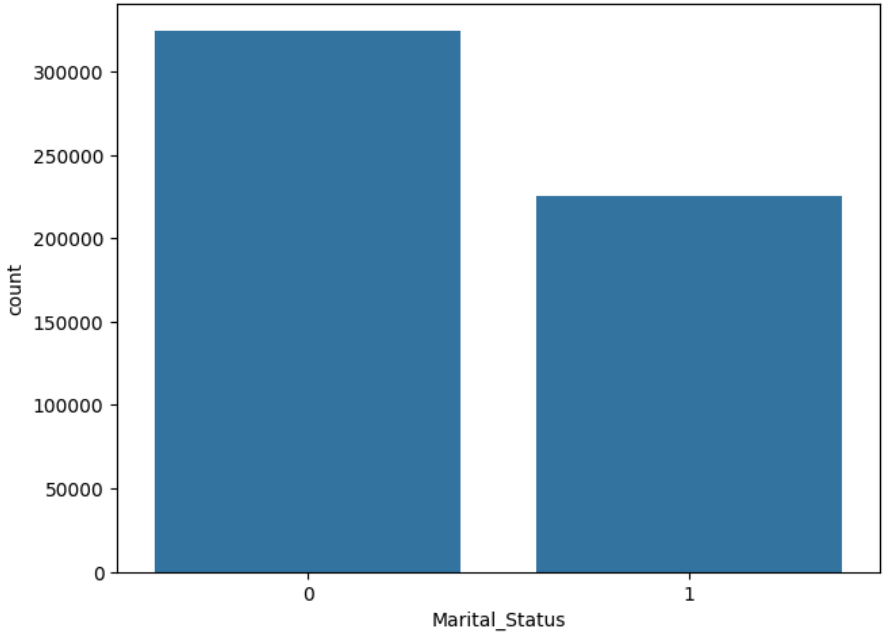


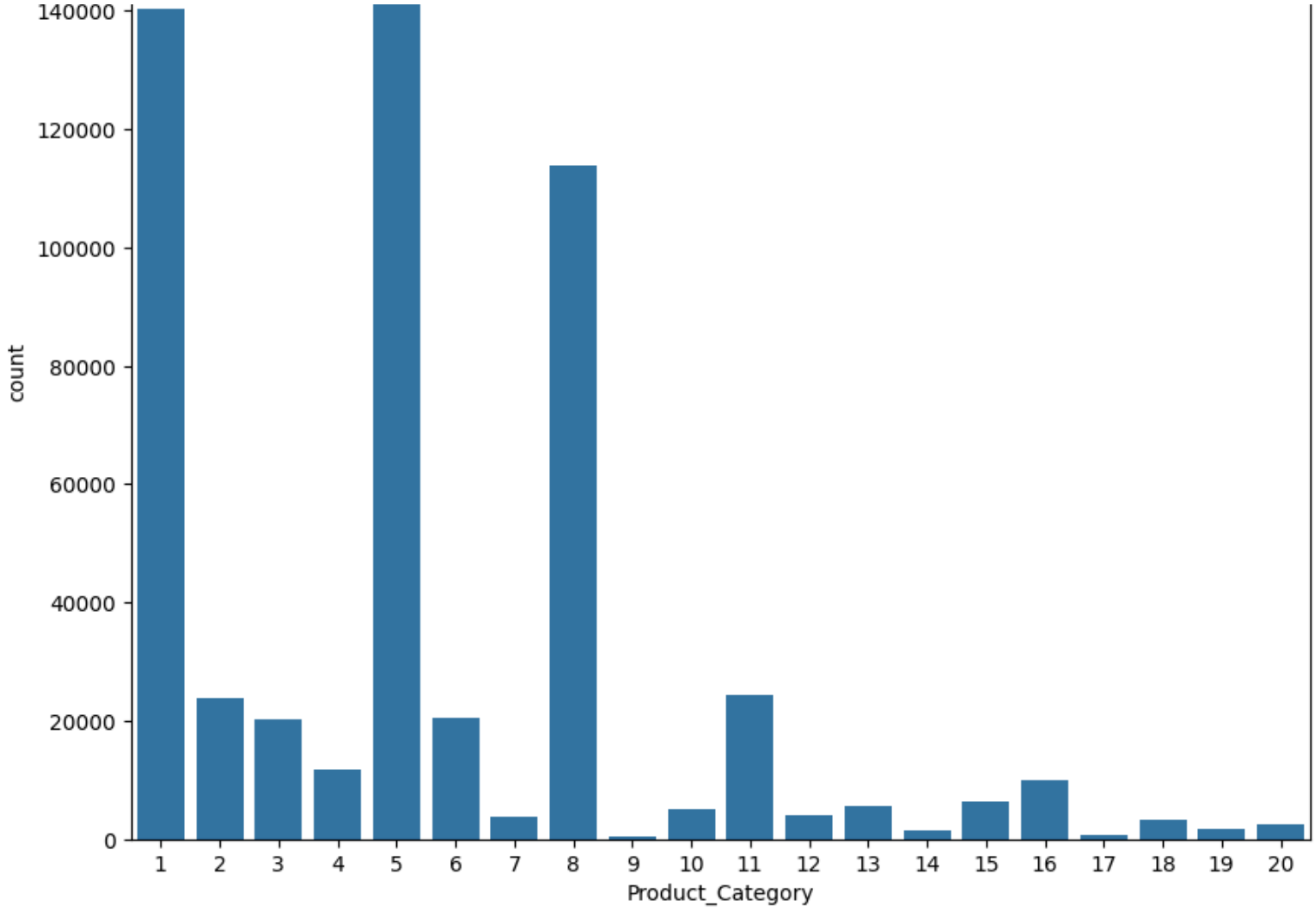

```
cat_cols = ['Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category']


fig,axs = plt.subplots(nrows=2,ncols=2,figsize=(16,12))
sns.countplot(data=df,x="Gender",ax=axs[0,0])
sns.countplot(data=df,x="Occupation",ax=axs[0,1])
sns.countplot(data=df,x="City_Category",ax=axs[1,0])
sns.countplot(data=df,x="Marital_Status",ax=axs[1,1])
plt.show()

plt.figure(figsize=(10,8))
sns.countplot(data=df,x="Product_Category")
plt.show()
```


count
400000
350000
300000
250000
200000
150000
100000
50000
0
F
M
Gender


count
70000
60000
50000
40000
30000
20000
10000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Occupation


count
200000
150000
100000
50000
0
A
C
B
City_Category


count
300000
250000
200000
150000
100000
50000
0
0
1
Marital_Status


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

Double-click (or enter) to edit

Observations:

1.Most of the users are male.

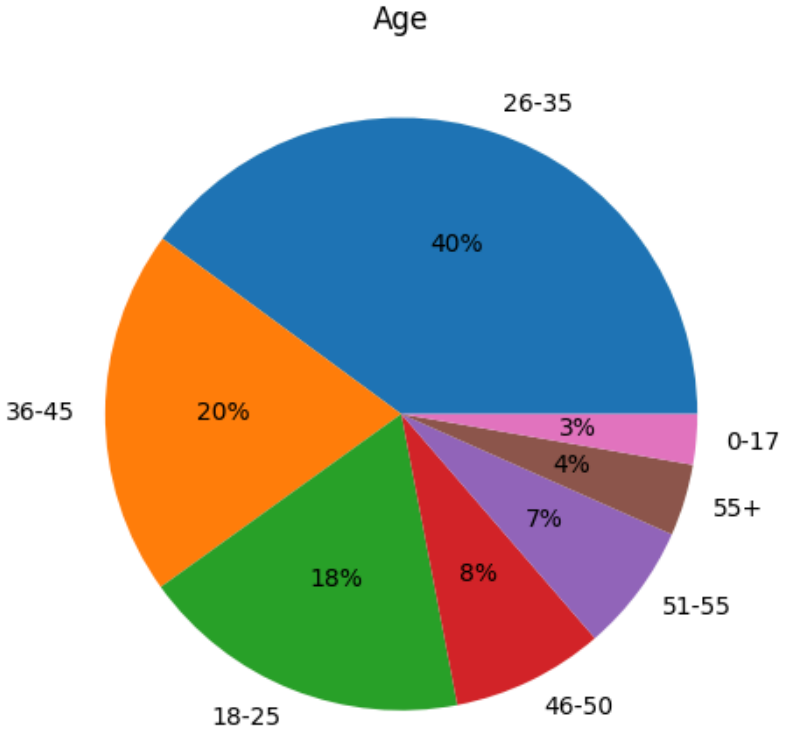
2.More users belong to B cty category.

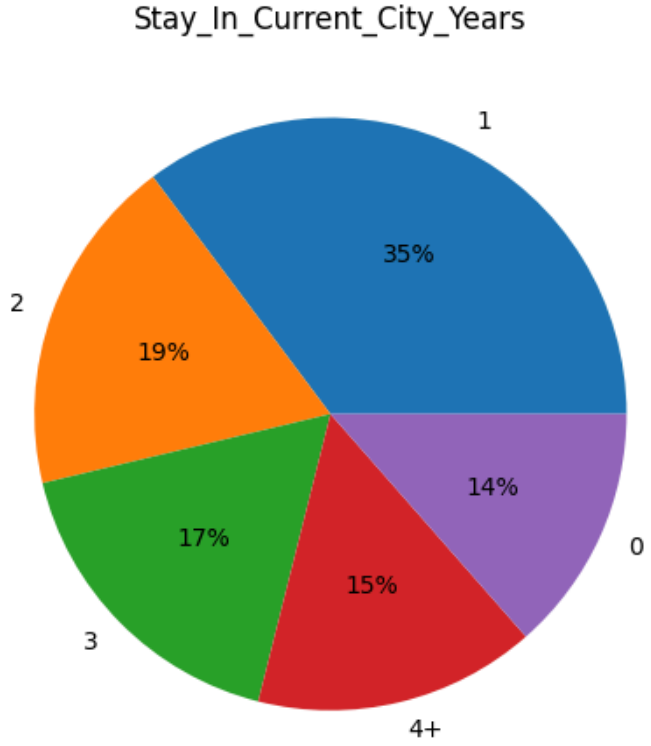
3.More users are single as compared to married.

4Product_Category 1,5,8,11 have highest purchasing frequency.

Start coding or generate with AI.

```
fig,axs = plt.subplots(nrows=1,ncols=2,figsize=(12,8))

data=df["Age"].value_counts(normalize=True)*100
axs[0].pie(x=data.values,labels=data.index,autopct="%.0f%%")
axs[0].set_title("Age")

data=df["Stay_In_Current_City_Years"].value_counts(normalize=True)*100
axs[1].pie(x=data.values,labels=data.index,autopct="%.0f%%")
axs[1].set_title("Stay_In_Current_City_Years")

plt.show()
```





Start coding or generate with AI.

```
#Bi-variate analysis

cat_cols = ['Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category']
```

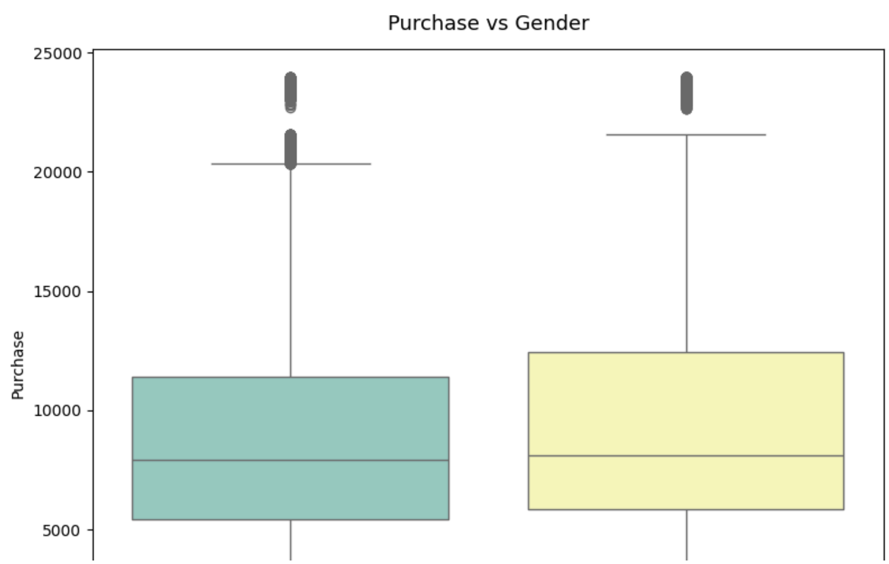
Start coding or generate with AI.

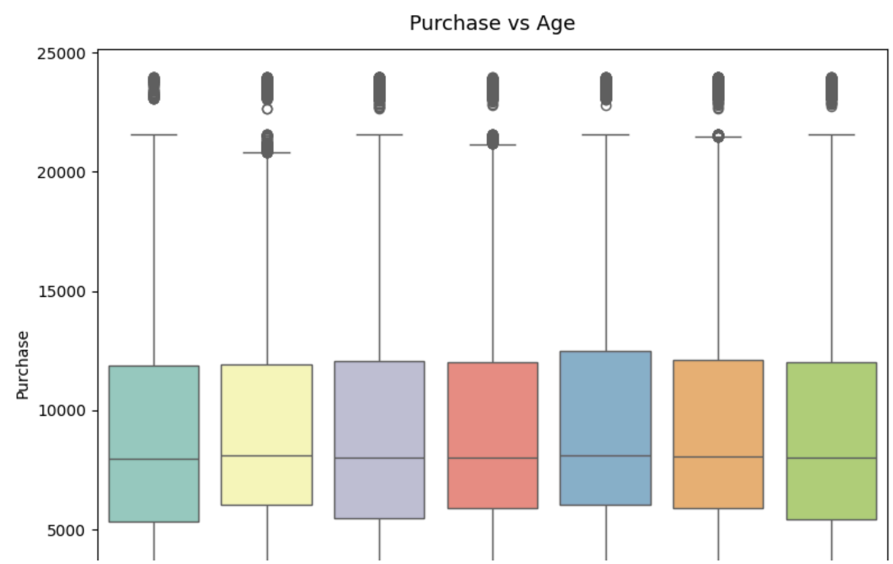
Start coding or generate with AI.

```
fig,axs = plt.subplots(nrows=3,ncols=2,figsize=(20,16))
fig.subplots_adjust(top=1.3)
count=0
for row in range(3):
```

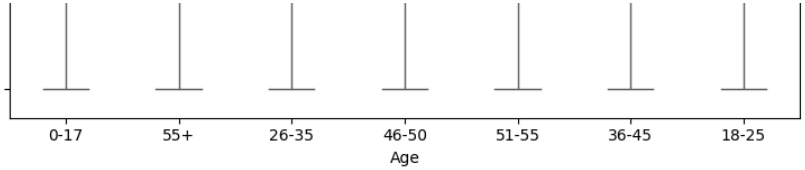
```
    for col in range(2):
      sns.boxplot(data=df,y="Purchase",x=cat_cols[count],ax=axs[row,col],palette="Set3")
      axs[row,col].set_title(f"Purchase vs {cat_cols[count]}",pad=12,fontsize=13)
      count+=1
plt.show()

plt.figure(figsize=(10,8))
sns.boxplot(data=df,y="Purchase",x=cat_cols[-1],palette="Set3")
plt.show()
```

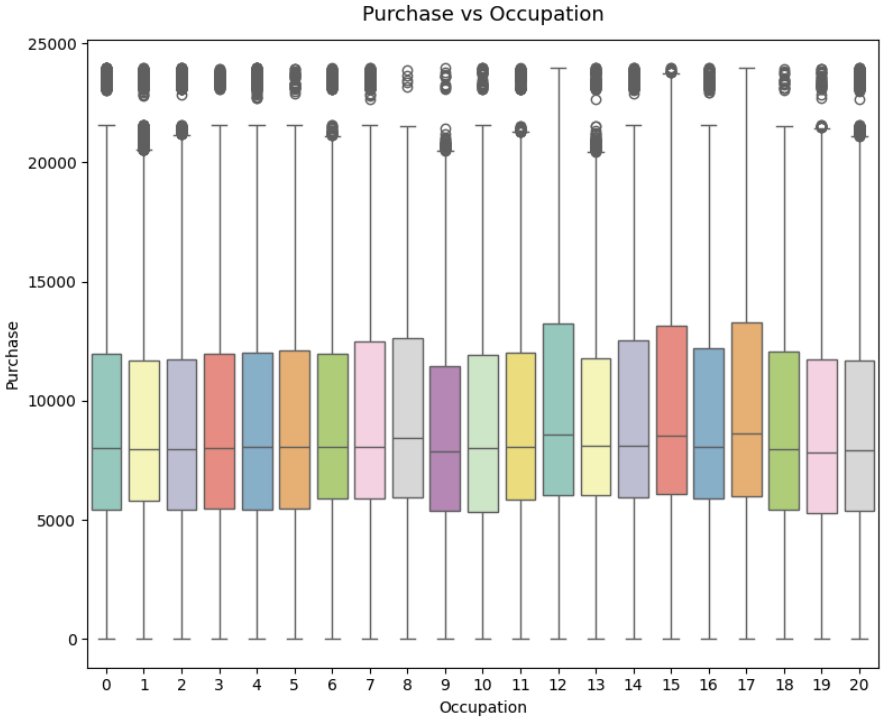
```
<ipython-input-26-2b5deada38b9>:6: FutureWarning:

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

  sns.boxplot(data=df,y="Purchase",x=cat_cols[count],ax=axs[row,col],palette="Set3")
<ipython-input-26-2b5deada38b9>:6: FutureWarning:

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

  sns.boxplot(data=df,y="Purchase",x=cat_cols[count],ax=axs[row,col],palette="Set3")
<ipython-input-26-2b5deada38b9>:6: FutureWarning:

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

  sns.boxplot(data=df,y="Purchase",x=cat_cols[count],ax=axs[row,col],palette="Set3")
<ipython-input-26-2b5deada38b9>:6: FutureWarning:

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

  sns.boxplot(data=df,y="Purchase",x=cat_cols[count],ax=axs[row,col],palette="Set3")
<ipython-input-26-2b5deada38b9>:6: FutureWarning:

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

  sns.boxplot(data=df,y="Purchase",x=cat_cols[count],ax=axs[row,col],palette="Set3")
<ipython-input-26-2b5deada38b9>:6: FutureWarning:

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

  sns.boxplot(data=df,y="Purchase",x=cat_cols[count],ax=axs[row,col],palette="Set3")
```

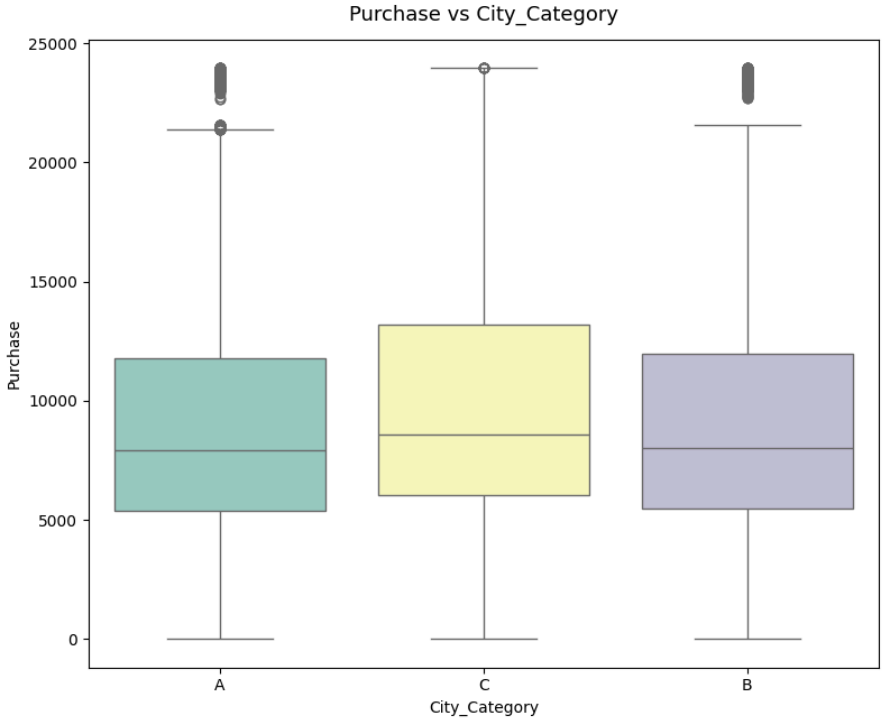




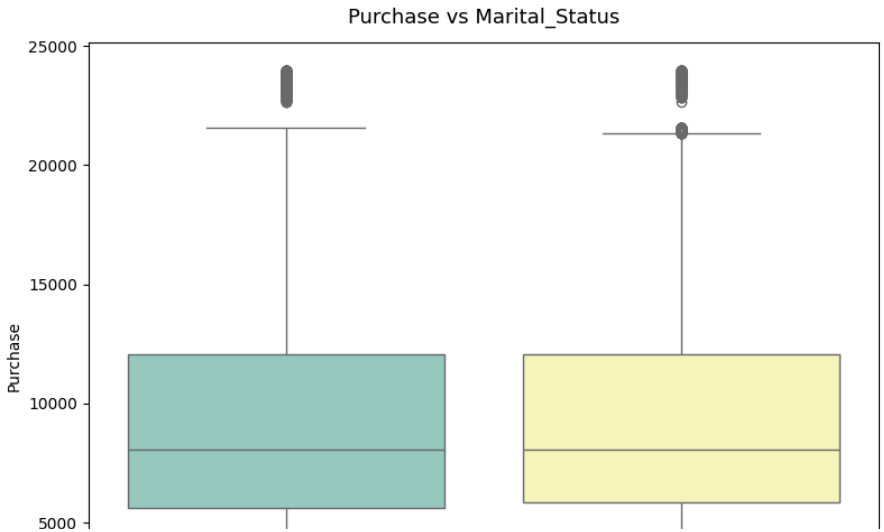

0
F
M
Gender

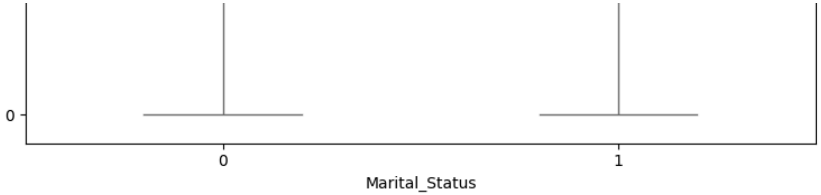

0
0-17
55+
26-35
46-50
51-55
36-45
18-25
Age


Purchase vs Occupation
Purchase
Occupation

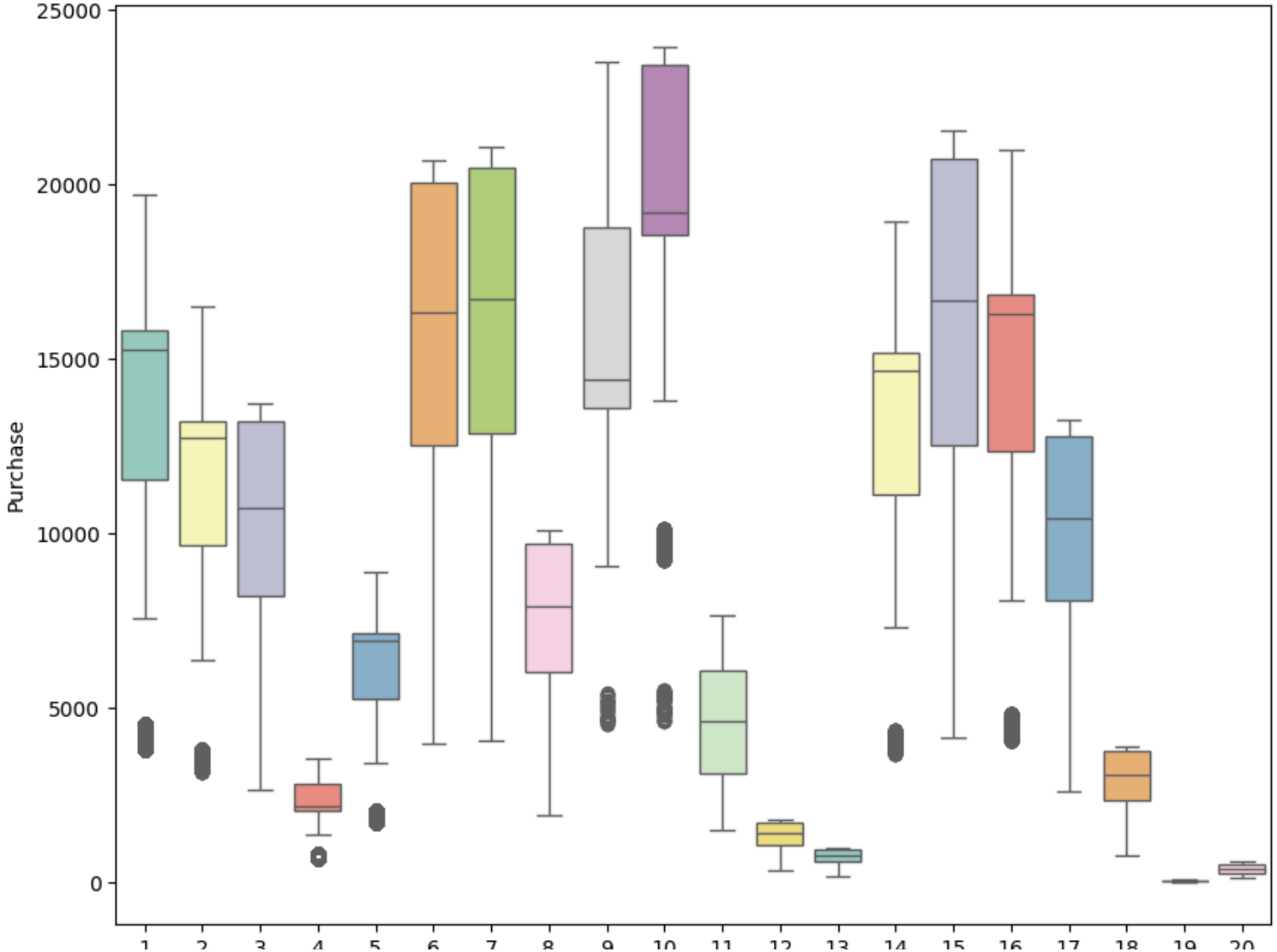

Purchase vs City_Category
Purchase
A
C
B
City_Category


Purchase vs Stay_In_Current_City_Years
Purchase


Purchase vs Marital_Status
Purchase




```
<ipython-input-26-2b5deada38b9>:12: FutureWarning:

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

  sns.boxplot(data=df,y="Purchase",x=cat_cols[-1],palette="Set3")
```

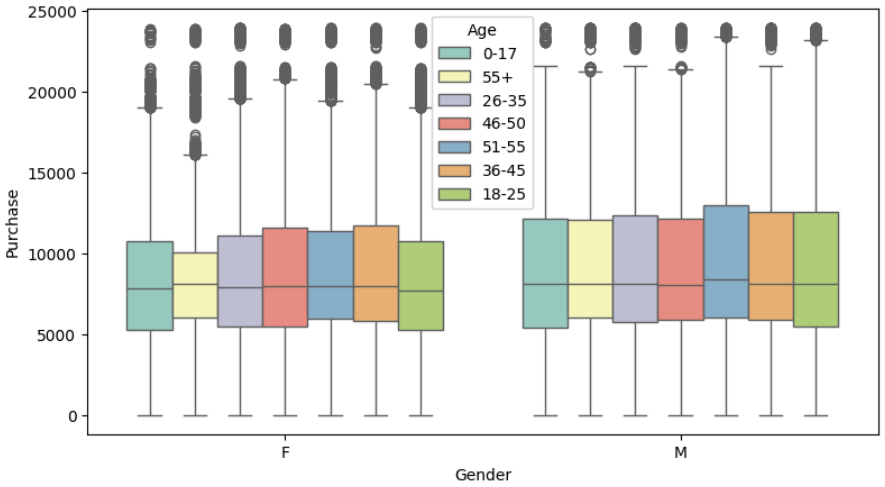


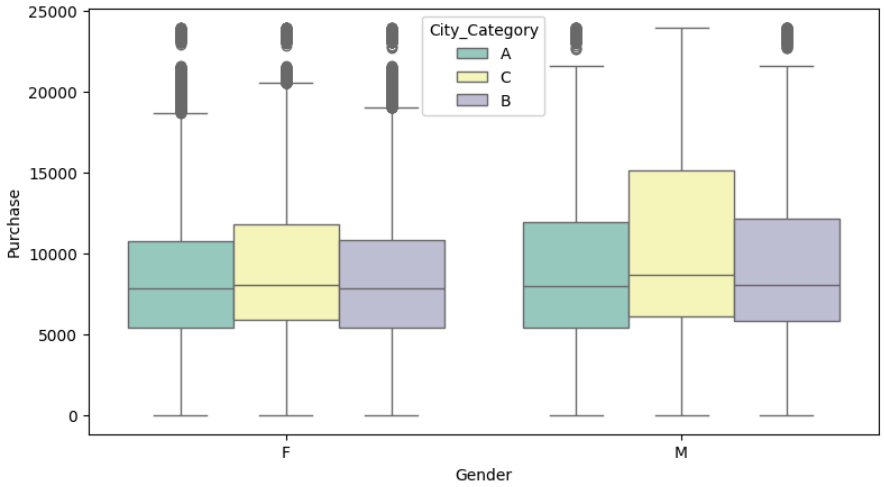
Product_Category

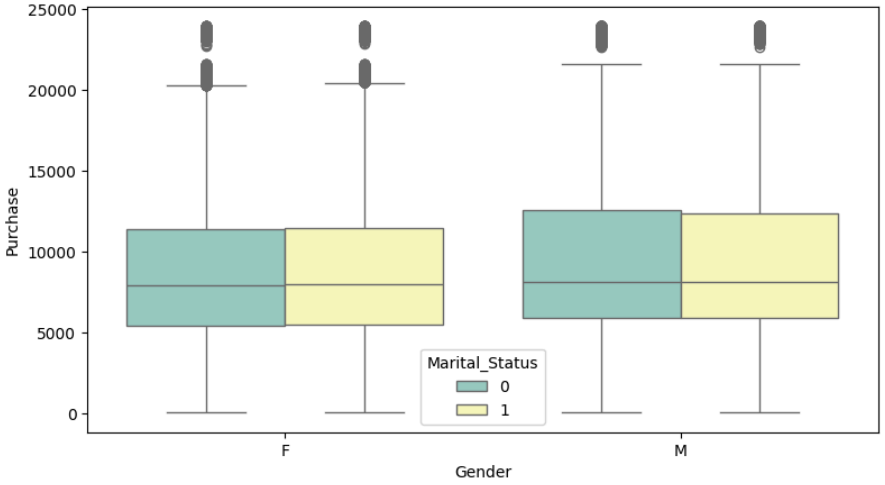
Start coding or generate with AI.

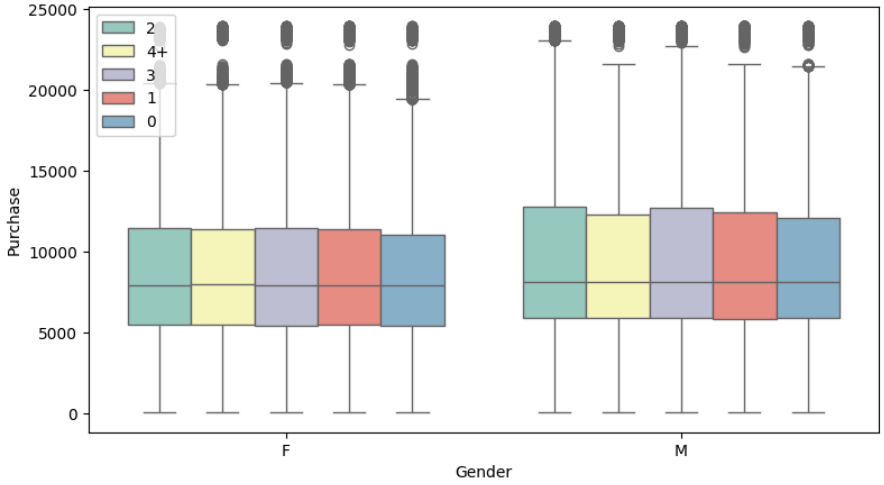
Start coding or generate with AI.

```
fig,axs=plt.subplots(nrows=2,ncols=2,figsize=(20,6))
fig.subplots_adjust(top=1.5)
sns.boxplot(data=df,y="Purchase",x="Gender",hue="Age",palette="Set3",ax=axs[0,0])
sns.boxplot(data=df,y="Purchase",x="Gender",hue="City_Category",palette="Set3",ax=axs[0,1])

sns.boxplot(data=df,y="Purchase",x="Gender",hue="Marital_Status",palette="Set3",ax=axs[1,0])
sns.boxplot(data=df,y="Purchase",x="Gender",hue="Stay_In_Current_City_Years",palette="Set3",ax=axs[1,1])
axs[1,1].legend(loc="upper left")

plt.show()
```









Start coding or generate with AI.

```
df.head(10)
```

	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
5	1000003	P00193542	M	26-35	15	A	3	0	1	15227
6	1000004	P00184942	M	46-50	7	B	2	1	1	19215

Start coding or generate with AI.

#Average amount spend per customer for Male and Female

```
amt_df=df.groupby(["User_ID","Gender"])[["Purchase"]].sum()
amt_df=amt_df.reset_index()
amt_df
```

	User_ID	Gender	Purchase
0	1000001	F	334093
1	1000002	M	810472
2	1000003	M	341635
3	1000004	M	206468
4	1000005	M	821001
...	...	...	...
5886	1006036	F	4116058
5887	1006037	F	1119538
5888	1006038	F	90034
5889	1006039	F	590319
5890	1006040	M	1653299

5891 rows × 3 columns

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

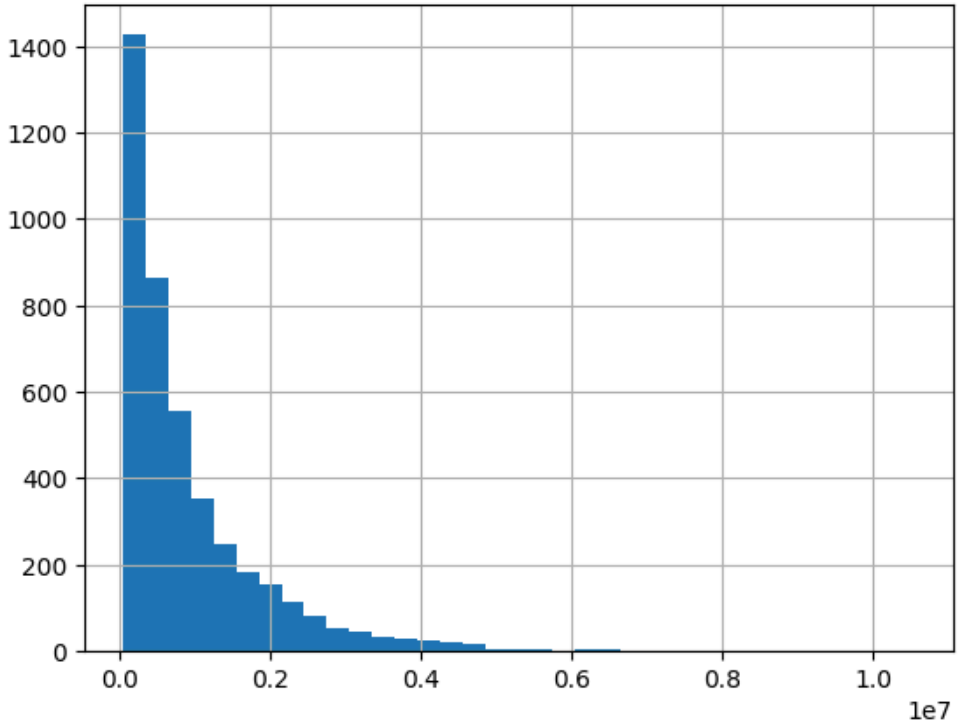
```
#Gender wise value counts in avg_amt_df
df["Gender"].value_counts()
```

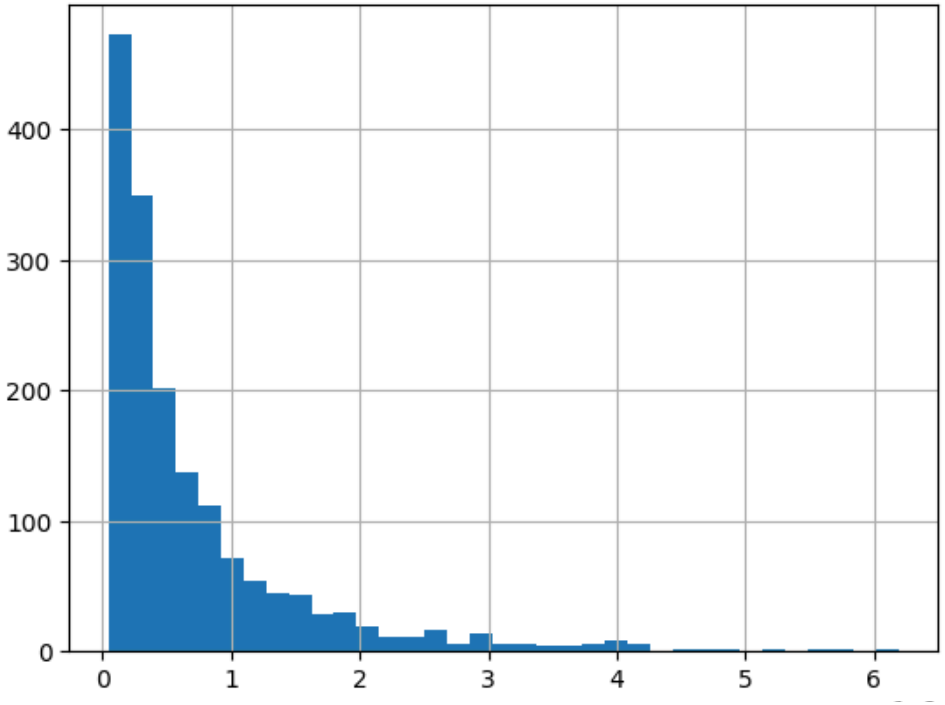
Gender	count
M	414259
F	135809

dtype: int64

```
#Histogram of average amount spend for each customer-male & female

amt_df[amt_df["Gender"]=="M"]["Purchase"].hist(bins=35)
plt.show()
```

```
amt_df[amt_df["Gender"]=="F"]["Purchase"].hist(bins=35)
plt.show()
```


1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0
1e7


400
300
200
100
0
0
1
2
3
4
5
6

1e6

Start coding or generate with AI.

```
male_avg = amt_df[amt_df["Gender"]=="M"]["Purchase"].mean()
female_avg = amt_df[amt_df["Gender"]=="M"]["Purchase"].mean()

print("Average amount spend by male customers : {:.2f}".format(male_avg))
print("Average amount spend by male customers : {:.2f}".format(female_avg))
```

```
Average amount spend by male customers : 925344.40
Average amount spend by male customers : 925344.40
```

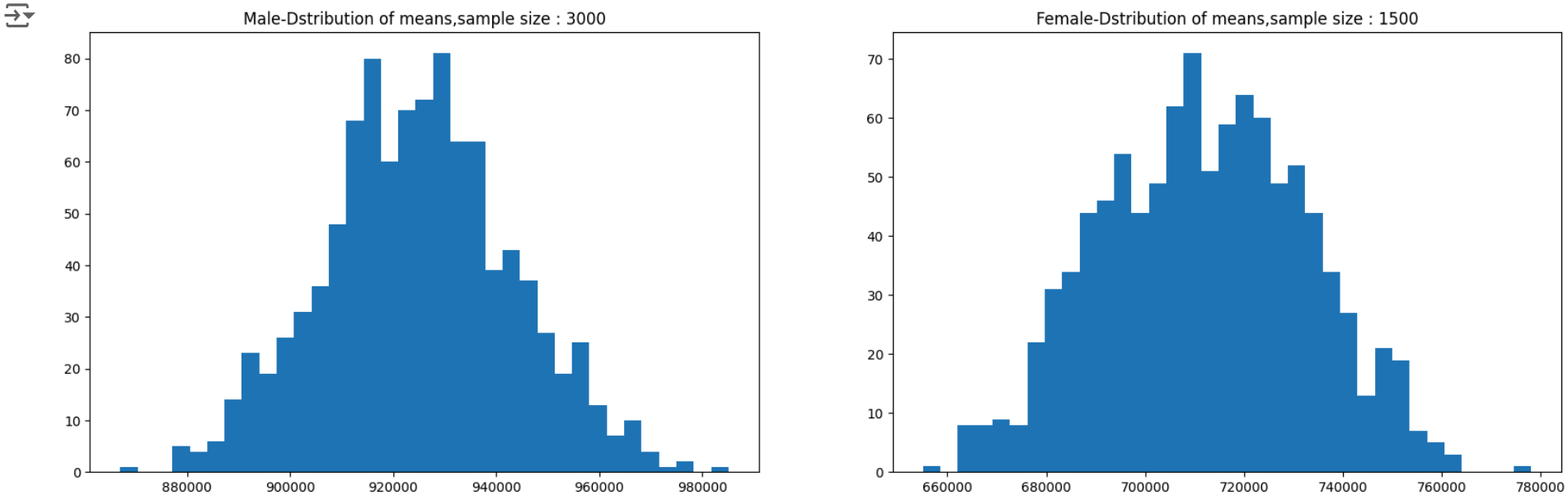
Observation:

Male customers spend more money than female customers

Start coding or generate with AI.

```
male_df = amt_df[amt_df["Gender"]=="M"]
female_df = amt_df[amt_df["Gender"]=="F"]



genders = ["M","F"]

male_sample_size=3000
female_sample_size = 1500
num_repitions=1000
male_means = []
female_means = []

for _ in range(num_repitions):
  male_mean = male_df.sample(male_sample_size,replace = True)["Purchase"].mean()
  female_mean = female_df.sample(female_sample_size,replace = True)["Purchase"].mean()

  male_means.append(male_mean)
  female_means.append(female_mean)
```

```
fig , axis = plt.subplots(nrows=1,ncols=2,figsize=(20,6))

axis[0].hist(male_means ,bins=35)
axis[1].hist(female_means ,bins=35)

axis[0].set_title("Male-Dstribution of means,sample size : 3000")
axis[1].set_title("Female-Dstribution of means,sample size : 1500")

plt.show()
```



Start coding or generate with AI.

```
import numpy as np
```

```
print("Population mean - Mean of sample means of amount spend for male :{:.2f}".format(np.mean(male_means)))
print("Population mean - Mean of sample means of amount spend for female :{:.2f}".format(np.mean(female_means)))

print("\nMale - Sample mean: {:.2f} Sample std: {:.2f}".format(male_df["Purchase"].mean(),male_df["Purchase"].std()))
print("Female - Sample mean: {:.2f} Sample std: {:.2f}".format(female_df["Purchase"].mean(),female_df["Purchase"].std()))
```

```
Population mean - Mean of sample means of amount spend for male :924897.98
Population mean - Mean of sample means of amount spend for female :712081.00

Male - Sample mean: 925344.40 Sample std: 985830.10
Female - Sample mean: 712024.39 Sample std: 807370.73
```

```
Start coding or generate with AI.
```

Observation:

Now using Central Limit Theorem for population we can say that:

Average amount spend by male customers is 926341.86

Average amount spend by female customers is 711704.09

```
male_margin_of_error_clt = 1.96*male_df['Purchase'].std()/np.sqrt(len(male_df))
male_sample_mean = male_df['Purchase'].mean()
male_lower_lim = male_sample_mean - male_margin_of_error_clt
male_upper_lim = male_sample_mean + male_margin_of_error_clt

female_margin_of_error_clt = 1.96*female_df['Purchase'].std()/np.sqrt(len(female_df))
female_sample_mean = female_df['Purchase'].mean()
female_lower_lim = female_sample_mean - female_margin_of_error_clt
female_upper_lim = female_sample_mean + female_margin_of_error_clt

print("Male confidence interval of means: ({:.2f}, {:.2f})".format(male_lower_lim, male_upper_lim))
print("Female confidence interval of means: ({:.2f}, {:.2f})".format(female_lower_lim, female_upper_lim))
```

```
Male confidence interval of means: (895617.83, 955070.97)
Female confidence interval of means: (673254.77, 750794.02)
```

Now we can infer about the population that , 95% of the times:

Average amount spend by male customers will lie between :(895617.83,9555070.97)

Average amount spend by male customers will lie between :(673254.77,750794.02)

Start coding or generate with AI.

#Doing same activity for married vs unmarried

amt_df

	User_ID	Gender	Purchase
0	1000001	F	334093
1	1000002	M	810472
2	1000003	M	341635
3	1000004	M	206468
4	1000005	M	821001
...	...	...	...
5886	1006036	F	4116058
5887	1006037	F	1119538
5888	1006038	F	90034
5889	1006039	F	590319
5890	1006040	M	1653299

5891 rows × 3 columns

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

```
amt_df = df.groupby(["User_ID","Marital_Status"])[["Purchase"]].sum()
amt_df = amt_df.reset_index()
amt_df
```

	User_ID	Marital_Status	Purchase
0	1000001	0	334093
1	1000002	0	810472
2	1000003	0	341635
3	1000004	1	206468
4	1000005	1	821001
...	...	...	...
5886	1006036	1	4116058
5887	1006037	0	1119538
5888	1006038	0	90034
5889	1006039	1	590319

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

```
amt_df["Marital_Status"].value_counts()
```

Marital_Status	count
0	3417
1	2474

dtype: int64

```
marid_samp_size = 3000
unmarid_sample_size = 2000
num_repitions = 1000
marid_means = []
unmarid_means = []

for _ in range(num_repitions):
    marid_mean = amt_df[amt_df['Marital_Status']==1].sample(marid_samp_size, replace=True)['Purchase'].mean()
```