About Walmart

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.

Business Problem

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).

Importing Python Libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
```

############ to display the output rows in single line

In [ ]:
```
pd.set_option('display.max_columns',None)
pd.set_option('display.width',1000)
pd.set_option('display.max_colwidth',50)
```

Loading Dataset

In [ ]:
```
df = pd.read_csv('C:\\Users\\emahsla\\python practice\\dataScience\\walmart\\Walmart.csv')
print(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        10.0             A                          2             0.0               3.0    8370.0
1  1000001  P00248942      F  0-17        10.0             A                          2             0.0               1.0   15200.0
2  1000001  P00087842      F  0-17        10.0             A                          2             0.0              12.0    1422.0
3  1000001  P00085442      F  0-17        10.0             A                          2             0.0              12.0    1057.0
4  1000002  P00285442      M   55+        16.0             C                         4+             0.0               8.0    7969.0
```

data analysis steps like checking the structure & characteristics of the dataset

In [ ]:
```
print(df.columns)
print(df.shape)
```

```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category', 'Purchase'], dtype='object')
(400699, 10)
```

In [ ]:
```
print(df.info())
print(df.describe(include="all"))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400699 entries, 0 to 400698
Data columns (total 10 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   User_ID                     400699 non-null  int64
 1   Product_ID                  400699 non-null  object
 2   Gender                      400698 non-null  object
 3   Age                         400698 non-null  object
 4   Occupation                  400698 non-null  float64
 5   City_Category               400698 non-null  object
 6   Stay_In_Current_City_Years  400698 non-null  object
 7   Marital_Status              400698 non-null  float64
 8   Product_Category            400698 non-null  float64
 9   Purchase                    400698 non-null  float64
dtypes: float64(4), int64(1), object(5)
memory usage: 30.6+ MB
None
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purchase
count	4.006990e+05	400699	400698	400698	400698.000000	400698	400698	400698.000000	400698.000000	400698.000000
unique	NaN	3575	2	7	NaN	3	5	NaN	NaN	NaN
top	NaN	P00265242	M	26-35	NaN	B	1	NaN	NaN	NaN
freq	NaN	1353	301958	159712	NaN	168466	140929	NaN	NaN	NaN
mean	1.002971e+06	NaN	NaN	NaN	8.077011	NaN	NaN	0.409867	5.295335	9325.345145
std	1.741129e+03	NaN	NaN	NaN	6.526370	NaN	NaN	0.491810	3.747050	4975.676726
min	1.000001e+06	NaN	NaN	NaN	0.000000	NaN	NaN	0.000000	1.000000	185.000000
25%	1.001425e+06	NaN	NaN	NaN	2.000000	NaN	NaN	0.000000	1.000000	5866.000000
50%	1.003001e+06	NaN	NaN	NaN	7.000000	NaN	NaN	0.000000	5.000000	8061.000000
75%	1.004446e+06	NaN	NaN	NaN	14.000000	NaN	NaN	1.000000	8.000000	12065.000000
max	1.006040e+06	NaN	NaN	NaN	20.000000	NaN	NaN	1.000000	18.000000	23961.000000

In [ ]:
```
print(df.isnull().sum())
print(df.nunique())
```

```
User_ID                     0
Product_ID                  0
Gender                      1
Age                         1
Occupation                  1
City_Category               1
Stay_In_Current_City_Years  1
Marital_Status              1
Product_Category            1
Purchase                    1
dtype: int64
User_ID                      5891
Product_ID                   3575
Gender                          2
Age                             7
Occupation                     21
City_Category                   3
Stay_In_Current_City_Years      5
Marital_Status                  2
Product_Category               18
Purchase                    17392
dtype: int64
```

Insights

1 - A quick look at the information of the data reveals that there are 400699 rows and 10 columns implying 400699 items have been sold to customers with information of each customer like User_ID, Gender, Age, Occupation to name a few. The datatype of Product_ID, Gender, Age, City_Category and Stay_In_Current_City_Years is “object” and rest is of int64 datatype.

2 - We can also infer that there are 1 missing values or nulls in the dataset.

In [ ]:
```
print(df[df['Age'].isnull()])
print(df[df['User_ID']==1001648])
print(df[df['Product_ID']=='P001'])
```

```
        User_ID Product_ID Gender  Age  Occupation City_Category Stay_In_Current_City_Years  Marital_Status  Product_Category  Purchase
400698  1001648       P001    NaN  NaN         NaN           NaN                        NaN             NaN               NaN       NaN
        User_ID Product_ID Gender    Age  Occupation City_Category Stay_In_Current_City_Years  Marital_Status  Product_Category  Purchase
10813   1001648  P00080342      M  51-55        14.0             B                          3             1.0               1.0   15313.0
10814   1001648  P00195242      M  51-55        14.0             B                          3             1.0               8.0    2233.0
10815   1001648  P00116542      M  51-55        14.0             B                          3             1.0               5.0    7132.0
10816   1001648  P00186142      M  51-55        14.0             B                          3             1.0               1.0   11830.0
10817   1001648  P00295342      M  51-55        14.0             B                          3             1.0               2.0    9954.0
...         ...        ...    ...    ...         ...           ...                        ...             ...               ...       ...
361480  1001648  P00187442      M  51-55        14.0             B                          3             1.0               2.0    6743.0
361481  1001648  P00102042      M  51-55        14.0             B                          3             1.0               5.0    3560.0
400696  1001648  P00086442      M  51-55        14.0             B                          3             1.0               8.0    5847.0
400697  1001648  P00111542      M  51-55        14.0             B                          3             1.0               2.0    9830.0
400698  1001648       P001    NaN    NaN         NaN           NaN                        NaN             NaN               NaN       NaN

[67 rows x 10 columns]
        User_ID Product_ID Gender  Age  Occupation City_Category Stay_In_Current_City_Years  Marital_Status  Product_Category  Purchase
400698  1001648       P001    NaN  NaN         NaN           NaN                        NaN             NaN               NaN       NaN
```

########## Since there is only 1 row which contains all the null values we can drop that particular row becasue it doesnt conatin any useful information.

In [ ]:
```
df = df.drop(400698)
print(df[df['User_ID']==1001648])
```

```
        User_ID Product_ID Gender    Age  Occupation City_Category Stay_In_Current_City_Years  Marital_Status  Product_Category  Purchase
10813   1001648  P00080342      M  51-55        14.0             B                          3             1.0               1.0   15313.0
10814   1001648  P00195242      M  51-55        14.0             B                          3             1.0               8.0    2233.0
10815   1001648  P00116542      M  51-55        14.0             B                          3             1.0               5.0    7132.0
10816   1001648  P00186142      M  51-55        14.0             B                          3             1.0               1.0   11830.0
10817   1001648  P00295342      M  51-55        14.0             B                          3             1.0               2.0    9954.0
...         ...        ...    ...    ...         ...           ...                        ...             ...               ...       ...
361479  1001648  P00089542      M  51-55        14.0             B                          3             1.0               5.0    6956.0
361480  1001648  P00187442      M  51-55        14.0             B                          3             1.0               2.0    6743.0
361481  1001648  P00102042      M  51-55        14.0             B                          3             1.0               5.0    3560.0
400696  1001648  P00086442      M  51-55        14.0             B                          3             1.0               8.0    5847.0
400697  1001648  P00111542      M  51-55        14.0             B                          3             1.0               2.0    9830.0

[66 rows x 10 columns]
```

In [ ]:
```
print(df.isnull().sum())
print(df.nunique())
```

```
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
User_ID                     5891
Product_ID                  3574
Gender                         2
Age                            7
Occupation                    21
City_Category                  3
Stay_In_Current_City_Years     5
Marital_Status                 2
Product_Category              18
Purchase                   17392
dtype: int64
```

1 - There are 2 genders, 7 age groups, 21 occupations, 3 city categories, 5 year groups of stay, 2 marital status and 20 categories of product.

2 - There are no duplicate entries.

3 - It makes sense to convert all columns except Purchase to "category" datatype.

In [ ]:
```
print(df.duplicated().sum())
```

```
0
```

In [ ]:
```
df["Gender"]=df["Gender"].astype("category")
df["Marital_Status"]=df["Marital_Status"].replace({0:"Unmarried", 1:"Married"})
df["Age"]=df["Age"].astype("category")
df["Stay_In_Current_City_Years"]=df["Stay_In_Current_City_Years"].astype("category")
df["Product_Category"]=df["Product_Category"].astype("category")
df["City_Category"]=df["City_Category"].astype("category")
df["Occupation"]=df["Occupation"].astype("category")
```

In [ ]:
```
columns = [ 'Gender', 'Marital_Status', 'Age', 'Stay_In_Current_City_Years', 'Product_Category', 'City_Category', 'Occupation']
for column in columns:
    grouped = df.groupby(column)['Purchase'].sum().apply(int).sort_values()
    print("grouping and their share of purchases ---"+ str(grouped))
```

```
grouping and their share of purchases ---Gender
F     869526775
M    2867120374
Name: Purchase, dtype: int64
grouping and their share of purchases ---Marital_Status
Married      1532427741
Unmarried    2204219408
Name: Purchase, dtype: int64
grouping and their share of purchases ---Age
0-17      100244962
55+       147157532
51-55     269718129
46-50     306873122
18-25     675466231
36-45     751802372
26-35    1485384801
Name: Purchase, dtype: int64
grouping and their share of purchases ---Stay_In_Current_City_Years
0      502675668
4+     577834440
3      649073674
2      695072461
1     1311990906
Name: Purchase, dtype: int64
grouping and their share of purchases ---Product_Category
13.0       2950728
12.0       3885754
17.0       4359305
9.0        4558275
18.0       6670168
14.0      14578398
4.0       20254040
7.0       44813209
15.0      68131175
10.0      74102115
11.0      82959006
16.0     106296417
3.0      150006350
2.0      197177877
6.0      235840721
8.0      628300972
5.0      691748753
1.0     1400013886
Name: Purchase, dtype: int64
grouping and their share of purchases ---City_Category
A     968436027
C    1220640145
B    1547570977
Name: Purchase, dtype: int64
grouping and their share of purchases ---Occupation
8.0      10839226
9.0      40305152
18.0     43794362
13.0     53526653
19.0     53811207
11.0     79116763
5.0      82016771
10.0     86654359
15.0     86723873
3.0     119658561
6.0     138576195
2.0     174808325
16.0    174836677
14.0    189832983
20.0    218103105
12.0    222418718
17.0    288294895
1.0     310916364
7.0     407268136
0.0     466742760
```

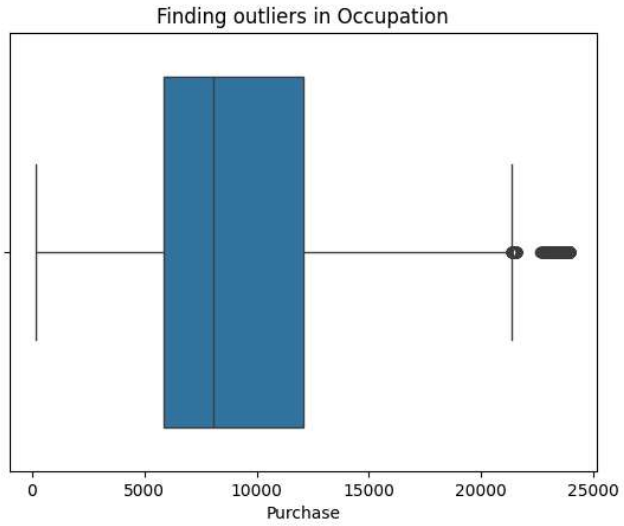
```
4.0     488402064
Name: Purchase, dtype: int64
```

Insights

1 - How much each category groups category is contributing to the purchase share.

2 - for example males purchases are more than 3 times then females. Unmarried people spend more amount of money as compared to married people. Age group 26-35 are the people who spend most amount of money. and much more.

In [ ]:
```
sns.boxplot(x='Purchase', data=df)
plt.title(f'Finding outliers in {column}')
plt.show()
```



In [ ]:
```
Purchase_25 = df.Purchase.quantile(0.25)
Purchase_75 = df.Purchase.quantile(0.75)
Purchase_median = df.Purchase.median()
Purchase_mean = df.Purchase.mean()
lower_limit = Purchase_25 - 1.5*(Purchase_75 - Purchase_25)
upper_limit = Purchase_75 + 1.5*(Purchase_75 - Purchase_25)
print(f'Lower limit: {lower_limit}\nUpper limit: {upper_limit}\nMedian: {Purchase_median}\nMean: {Purchase_mean}')
print(f"Outliers: {round((len(df[df['Purchase']>upper_limit])/len(df))*100,2)}%")
```

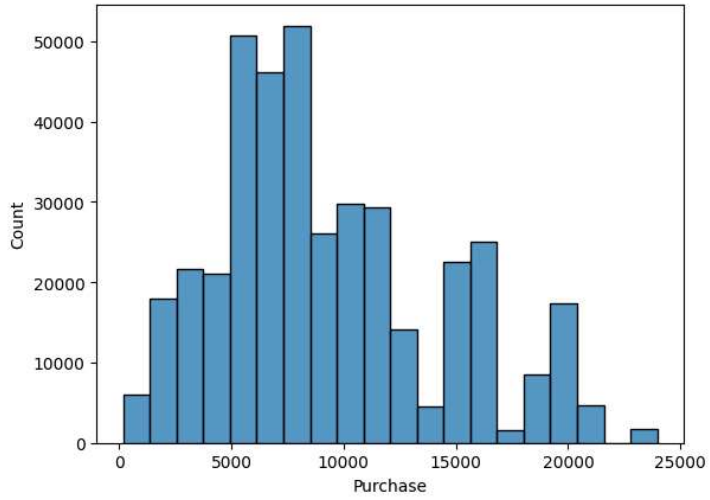
```
Lower limit: -3432.5
Upper limit: 21363.5
Median: 8061.0
Mean: 9325.345145221589
Outliers: 0.5%
```

Insights

Since there are only 0.5% of outliers in purchases column, we can ignore that and let us go ahead with the present data. These outliers are also not a very large value as such compared which might deflect the outcome.

In [ ]:
```
sns.histplot(data=df, x = "Purchase", bins=20)
```

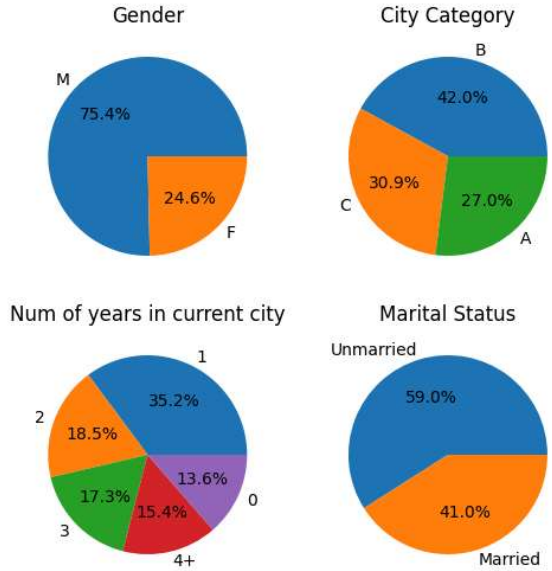
Out[ ]:
```
<Axes: xlabel='Purchase', ylabel='Count'>
```



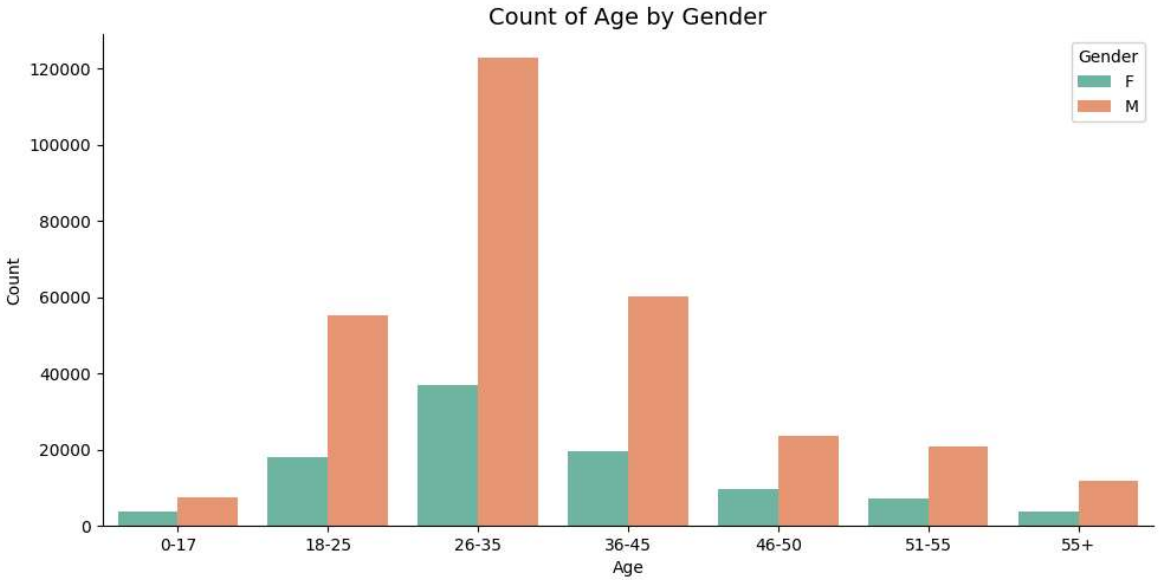
Insights

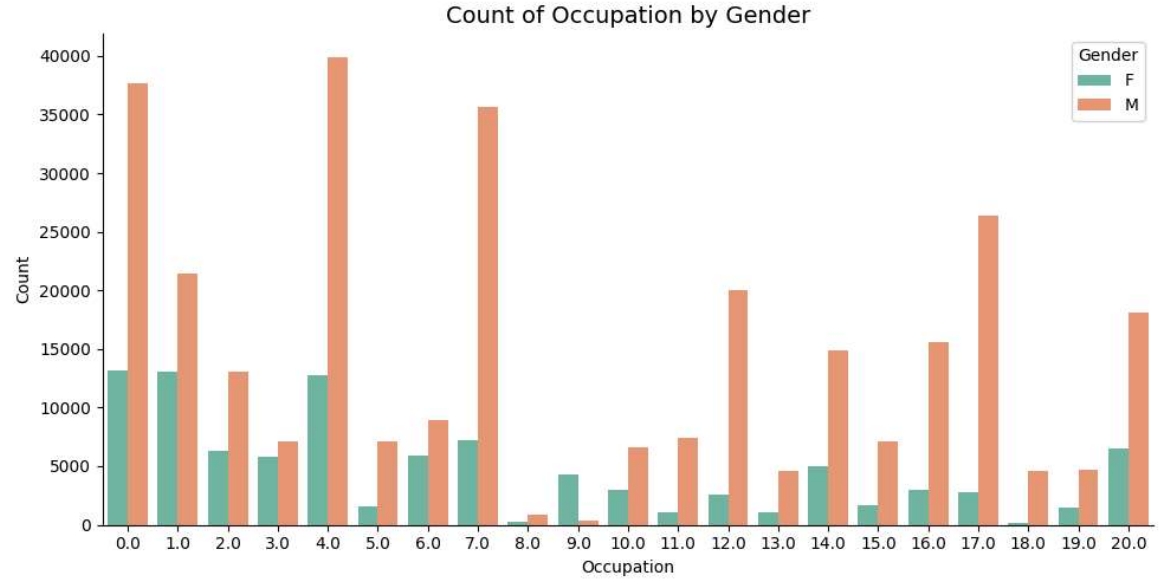
maximum number of individuals spent mostly in 5000 - 8000 of range

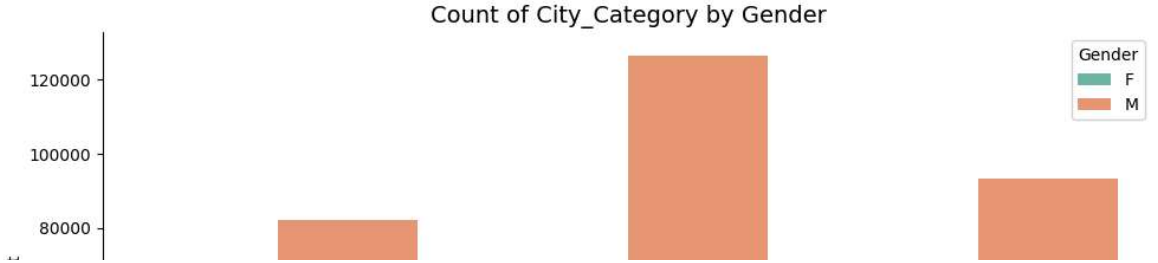
In [ ]:
```
plt.figure(figsize=(6,6))
plt.subplot(2,2,1)
data = df["Gender"].value_counts()
plt.pie(data.values, labels = data.index, autopct='%.1f%%')
plt.title("Gender")
plt.subplot(2,2,2)
data = df["City_Category"].value_counts()
plt.pie(data.values, labels = data.index, autopct='%.1f%%')
plt.title("City Category")
plt.subplot(2,2,3)
data = df["Stay_In_Current_City_Years"].value_counts()
plt.pie(data.values, labels = data.index, autopct='%.1f%%')
plt.title("Num of years in current city")
plt.subplot(2,2,4)
data = df["Marital_Status"].value_counts()
plt.pie(data.values, labels = data.index, autopct='%.1f%%')
plt.title("Marital Status")
plt.show()
```

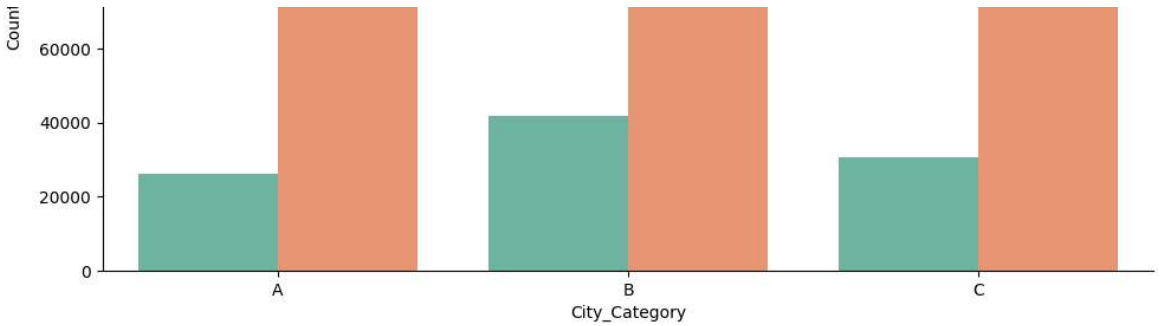


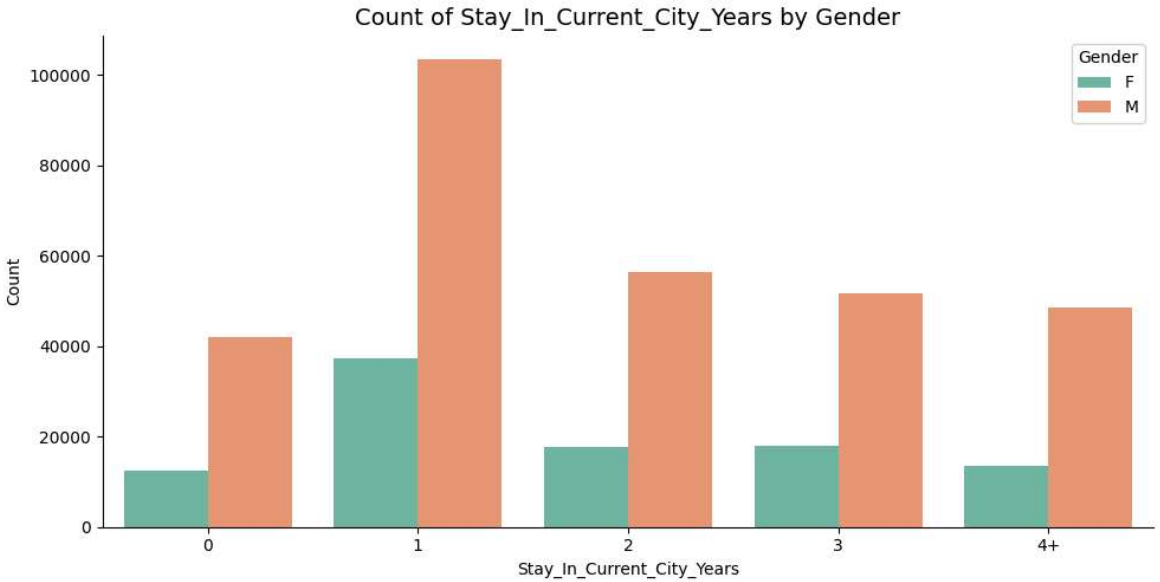
In [ ]:
```
category = ['Age', 'Occupation', 'City_Category', 'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category']
plt.figure(figsize=(10, 30))

for i, col in enumerate(category, 1):
    plt.subplot(6, 1, i)
    ax = sns.countplot(data=df, x=col, hue='Gender', palette='Set2')
    sns.despine()

    plt.title(f'Count of {col} by Gender', fontsize=14)
    plt.xlabel(col)
    plt.ylabel('Count')

    plt.tight_layout()

plt.show()
```

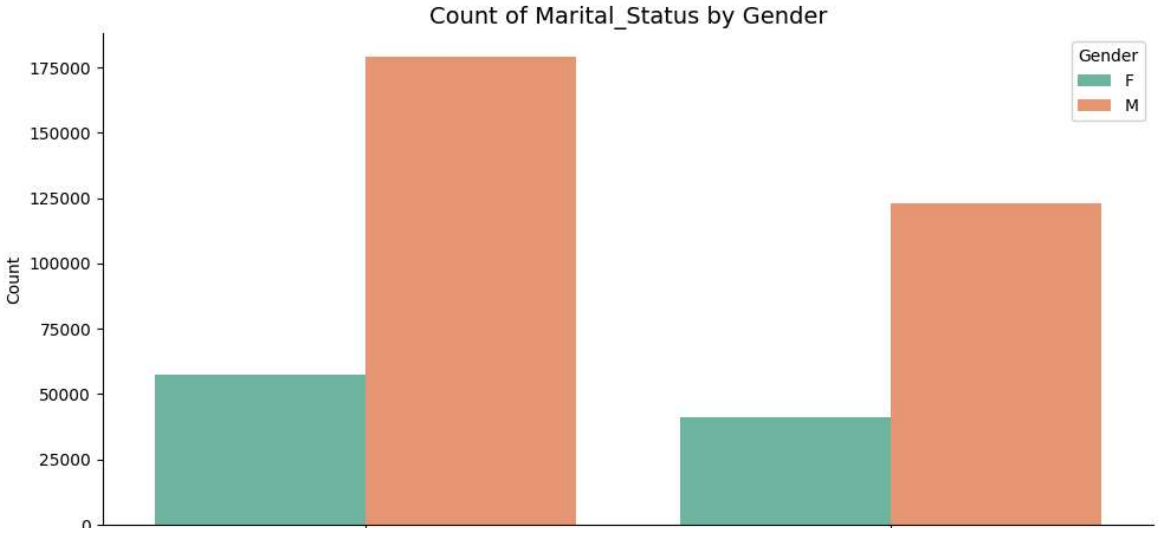

Count of Age by Gender
Gender
F
M
120000
100000
80000
60000
40000
20000
0
Count
0-17
18-25
26-35
36-45
46-50
51-55
55+
Age

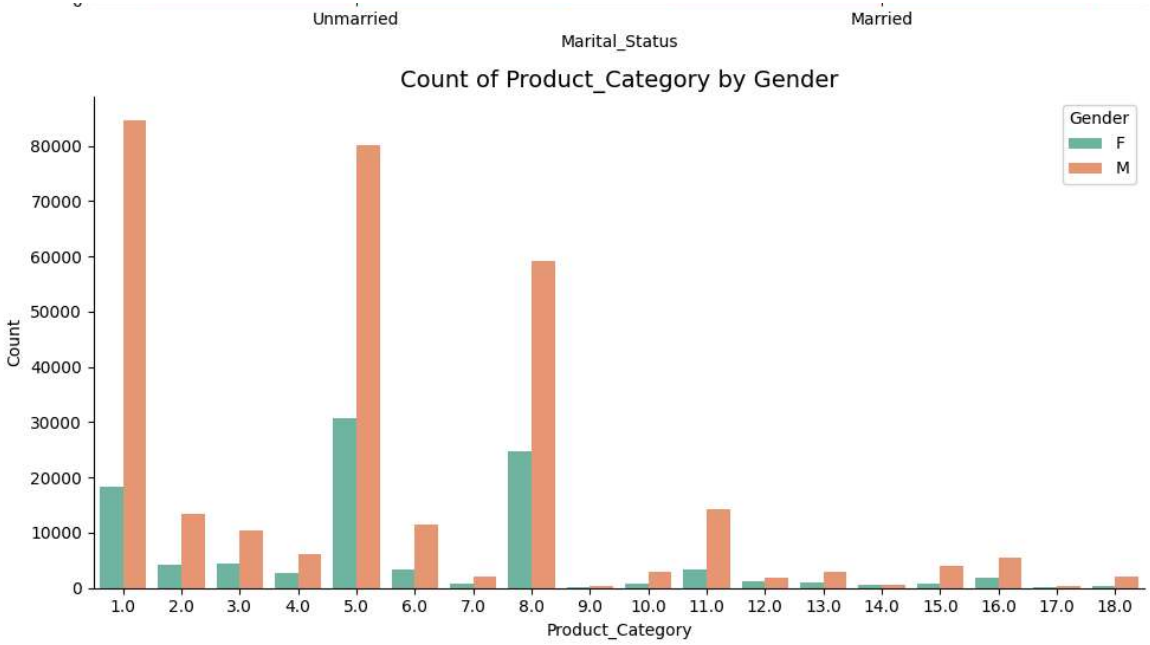

Count of Occupation by Gender
Gender
F
M
40000
35000
30000
25000
20000
15000
10000
5000
0
Count
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
Occupation


Count of City_Category by Gender
Gender
F
M
120000
100000
80000


Count
60000
40000
20000
0
A
B
C
City_Category


Count of Stay_In_Current_City_Years by Gender
Gender
F
M
Count
100000
80000
60000
40000
20000
0
0
1
2
3
4+
Stay_In_Current_City_Years


Count of Marital_Status by Gender
Gender
F
M
Count
175000
150000
125000
100000
75000
50000
25000
0



Insights

Occupation-Related Purchase Analysis:

Occupations '0' and '4' show the highest purchase counts, suggesting that individuals in these occupations contribute significantly to overall sales, with '4' having notably higher purchases than others.

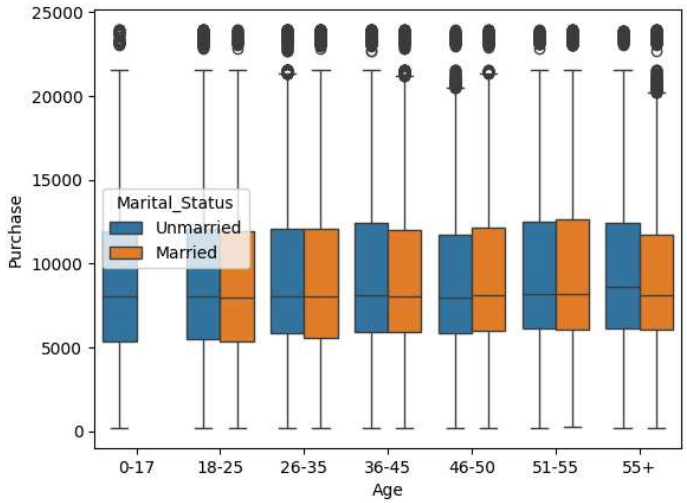
City Category-Related Purchase Analysis:

City_Category 'B' has the highest purchase counts for both genders, indicating that customers residing in City_Category 'B' contribute significantly to overall sales compared to 'A' and 'C'.

Stay in Current City Duration Impact:

Customers who have stayed in their current city for 1 year exhibit the highest purchase counts, suggesting that individuals with a 1-year residence duration have a higher tendency to make purchases compared to other durations.

In [ ]:
```
sns.boxplot(data=df, x = 'Age', y='Purchase', hue='Marital_Status')
plt.show()
```

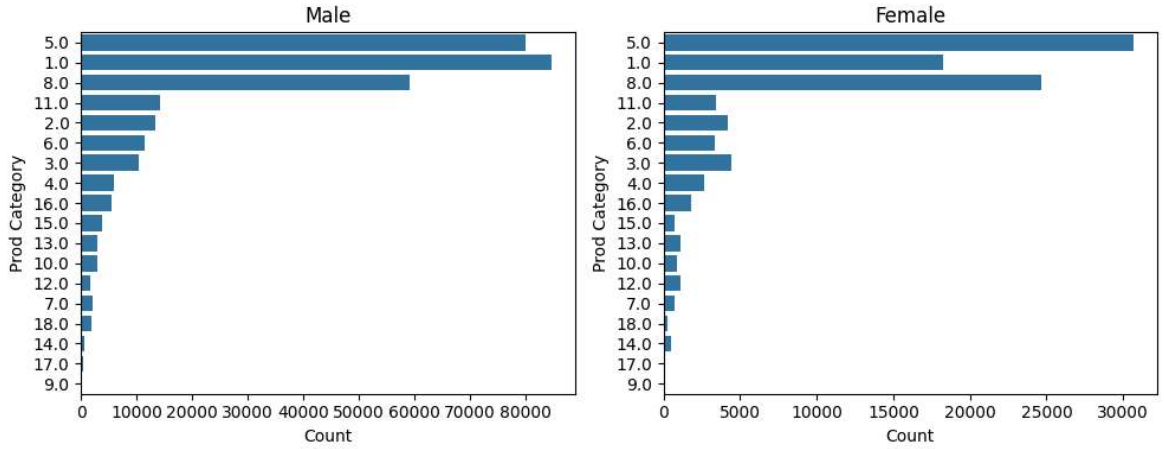


Insights

We can observe that the people of different age groups, mariied or unmarried, spend almost the same amount of money.

In [ ]:
```
avg_purchase = df.groupby('Gender')[['Purchase']].mean().reset_index().round(2)
print(avg_purchase)
df_male = df[df['Gender']=='M']
df_female = df[df['Gender']=='F']
print(f'Male customers count- {len(df_male)}')
print(f'Female customers count - {len(df_female)}')

fig, axes = plt.subplots(1, 2, figsize=(10, 4))
sns.countplot(data=df_male, y='Product_Category', ax=axes[0], order=df['Product_Category'].value_counts().index)
axes[0].set_title('Male')
axes[0].set_xlabel('Count')
axes[0].set_ylabel('Prod Category')

sns.countplot(data=df_female, y='Product_Category', ax=axes[1], order=df['Product_Category'].value_counts().index)
axes[1].set_title('Female')
axes[1].set_xlabel('Count')
axes[1].set_ylabel('Prod Category')

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

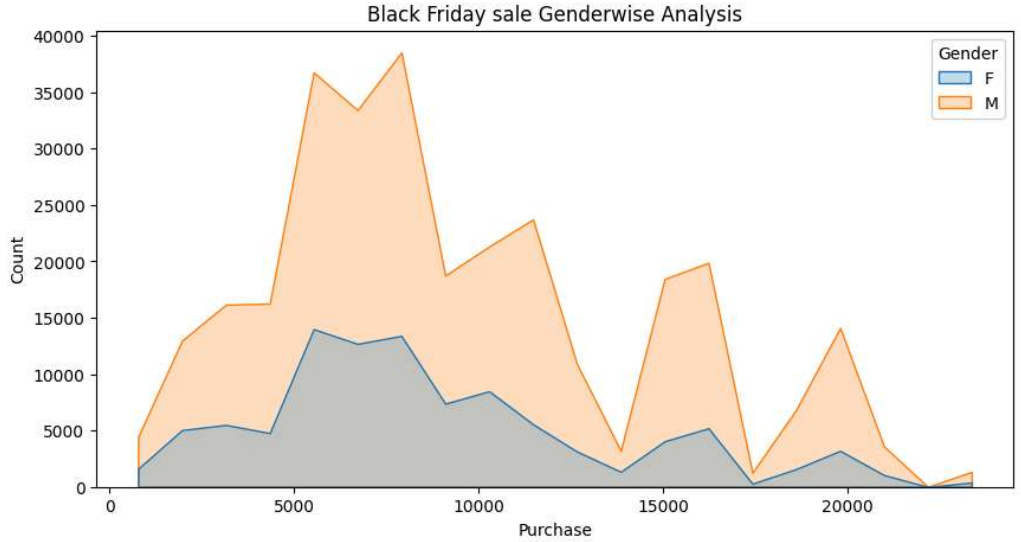
```
  Gender  Purchase
0      F   8806.23
1      M   9495.10
Male customers count- 301958
Female customers count - 98740
```



Insights

Males prefer to buy product category 1 more and female prefer to buy product category 5. The top 3 product categories prefered by both male and female are 1, 5 and 8.

In [ ]:
```
plt.figure(figsize=(10,5))
sns.histplot(data=df, x = "Purchase", bins=20, hue = "Gender",element='poly')
plt.title('Black Friday sale Genderwise Analysis')
plt.show()
```



Men spent more money than women during the Black Friday sale.

The total number of male customers (4225) exceeds the total number of female customers (1666).

The average amount spent by male customers (9437) is higher than the average amount spent by female customers (8734).

With a larger male customer base, it is likely that men will make more purchases compared to females.

The higher sales among male customers could be attributed to a product range better suited to their preferences, leading to increased sales of products targeted towards men.

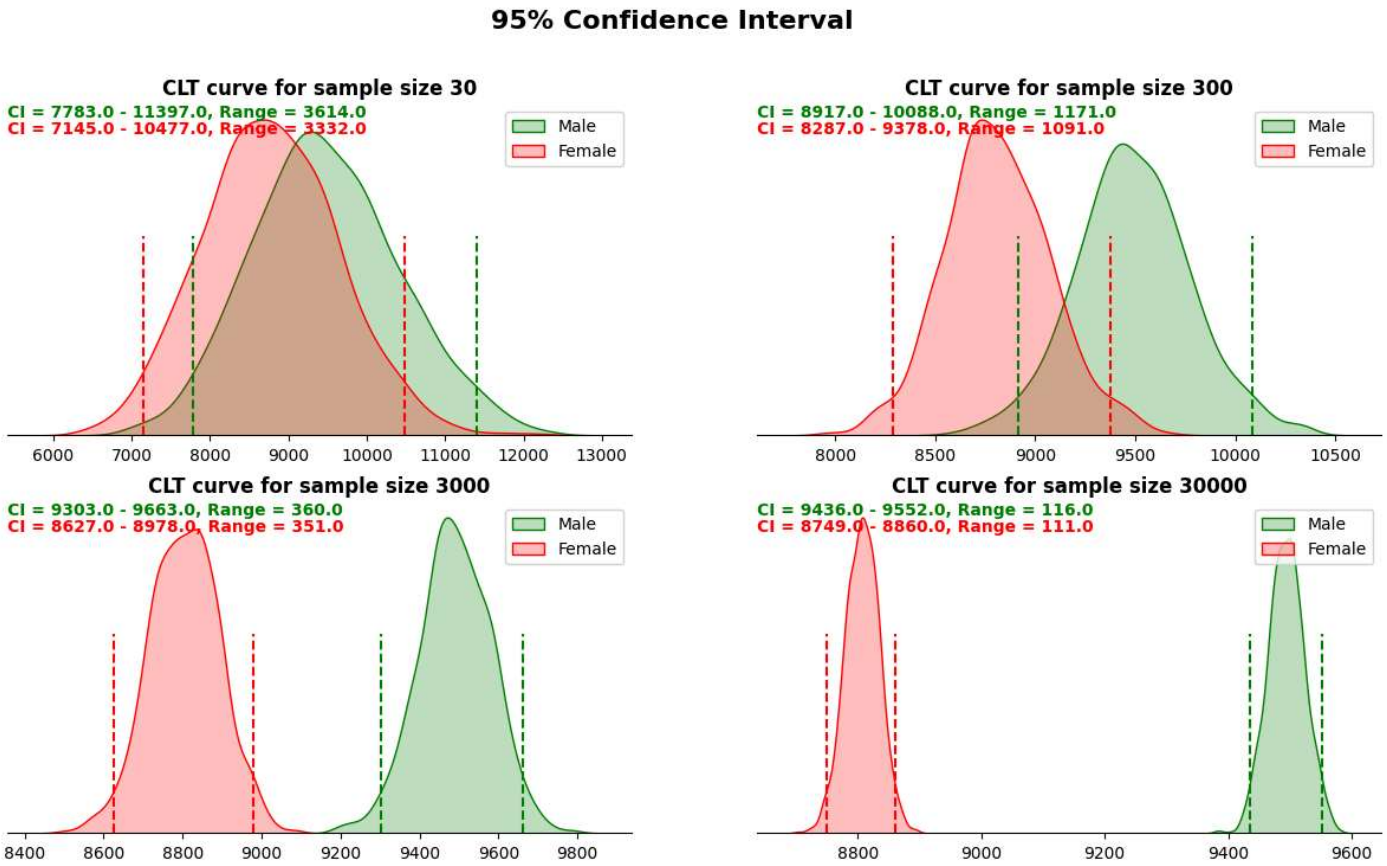
CLT ANALYSIS :

How does gender affect the amount spent?

In [ ]:
```
def confidence_interval(data, ci):
    l_ci = (100-ci)/2
    u_ci = (100+ci)/2
    interval = np.percentile(data, [l_ci, u_ci]).round(0)
    return interval
sample_sizes = [(30, 0, 0), (300, 0, 1), (3000, 1, 0), (30000, 1, 1)]
#sample_sizes = [(30, 0, 0), (30, 0, 1), (30, 1, 0), (30, 1, 1)]
bootstrap_samples = 1000
df_1 = df[df['Gender'] == 'M']['Purchase'].reset_index(drop=True)
df_2 = df[df['Gender'] == 'F']['Purchase'].reset_index(drop=True)
interval_1 = confidence_interval(df_1, 95)
interval_2 = confidence_interval(df_2, 95)
range_1 = interval_1[1] - interval_1[0]
range_2 = interval_2[1] - interval_2[0]
print(f'Male  : CI = {interval_1}, Range = {range_1} \nFemale: CI = {interval_2}, Range = {range_2}')
```

```
Male  : CI = [ 1704. 19925.], Range = 18221.0
Female: CI = [ 1574. 19652.], Range = 18078.0
```

In [ ]:
```
label1 = 'Male'
label2 = 'Female'
data1_color = 'g'
data2_color = 'r'
data1_text_y_pos = 1
data2_text_y_pos = 0.95

fig = plt.figure(figsize=(15,8))
gs = fig.add_gridspec(2,2)
ci_percent = 95

for sample_size, row, col in sample_sizes:
    data1_means = [np.mean(random.choices(df_1, k=sample_size)) for idx in range(bootstrap_samples)]
    data2_means = [np.mean(random.choices(df_2, k=sample_size)) for idx in range(bootstrap_samples)]

    ax = fig.add_subplot(gs[row, col])

    for means, color, label in [(data1_means, data1_color, label1), (data2_means, data2_color, label2)]:
        sns.kdeplot(ax = ax, x=means, color=color, fill=True, label = label)

    data1_ci = confidence_interval(data1_means, ci_percent)
    data2_ci = confidence_interval(data2_means, ci_percent)
    for ci, color in [(data1_ci, data1_color), (data2_ci, data2_color)]:
        for k in ci:
            ax.axvline(x=k, ymax=0.6, color=color, linestyle='--')

    for s in ['top','left','right']:
        ax.spines[s].set_visible(False)

    for ci, color, text_pos in [(data1_ci, data1_color, data1_text_y_pos), (data2_ci, data2_color, data2_text_y_pos)]:
        ax.text(0, text_pos, f"CI = {ci[0]} - {ci[1]}, Range = {ci[1]-ci[0]}", transform=ax.transAxes, size=10, weight='bold', verticalalignment='top', color=color)
    ax.set_yticks([])
    ax.set_ylabel('')
    ax.set_xlabel('')
    ax.set_title(f'CLT curve for sample size {sample_size}', size = 12, weight = 'bold')
    ax.legend()

fig.suptitle(f'{ci_percent}% Confidence Interval', size = 16, weight = 'bold')
plt.show()
```



Confidence interval calculated using the entire dataset for both the genders is almost same. [1704. 19925.] for male and [1574. 19652.] for female.

As the sample size increases the width of the confidence interval decreases.

The confidence intervals for males and females overlap for sample sizes 30 and 300 but then they start moving apart for sample sizes 3000 and 30000.

The shape of the distribution of the means gets closer to the normal distribution as the same size increases.

How does Marital_Status affect the amount spent?

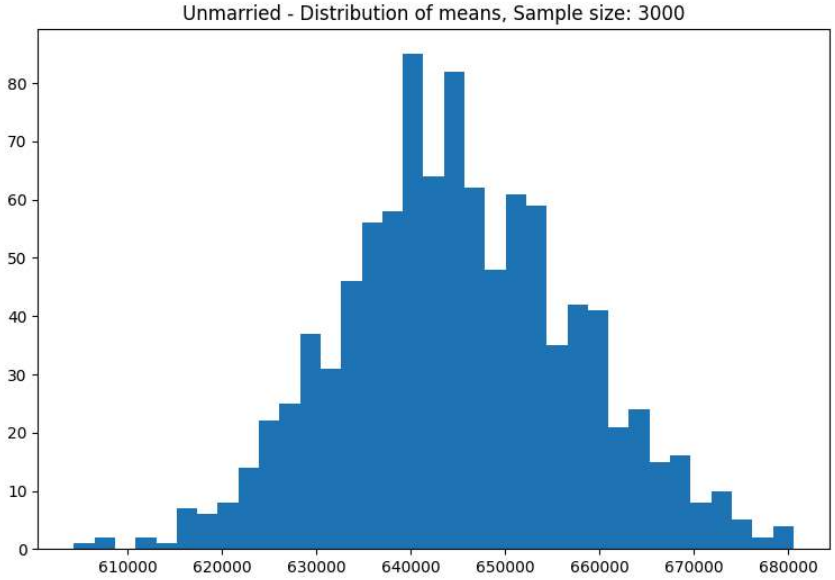
In [ ]:
```
sum_by_Marital_Status = df.groupby(['User_ID', 'Marital_Status'])['Purchase'].sum()
# print(sum_by_Marital_Status)
sum_by_Marital_Status = sum_by_Marital_Status.reset_index()
sum_by_Marital_Status = sum_by_Marital_Status.sort_values(by='User_ID', ascending=False)
# print(sum_by_Marital_Status)
Married_cust_avg = sum_by_Marital_Status[sum_by_Marital_Status['Marital_Status']=='Married']['Purchase'].mean()
print(f'Married customer average spent amount: {Married_cust_avg}')

sum_by_Marital_Status = df.groupby(['User_ID', 'Marital_Status'])['Purchase'].sum()
sum_by_Marital_Status = sum_by_Marital_Status.reset_index()
sum_by_Marital_Status = sum_by_Marital_Status.sort_values(by='User_ID', ascending=False)
Unmarried_cust_avg = sum_by_Marital_Status[sum_by_Marital_Status['Marital_Status']=='Unmarried']['Purchase'].mean()
print(f'Unmarried customer average spent amount: {Unmarried_cust_avg}')
```

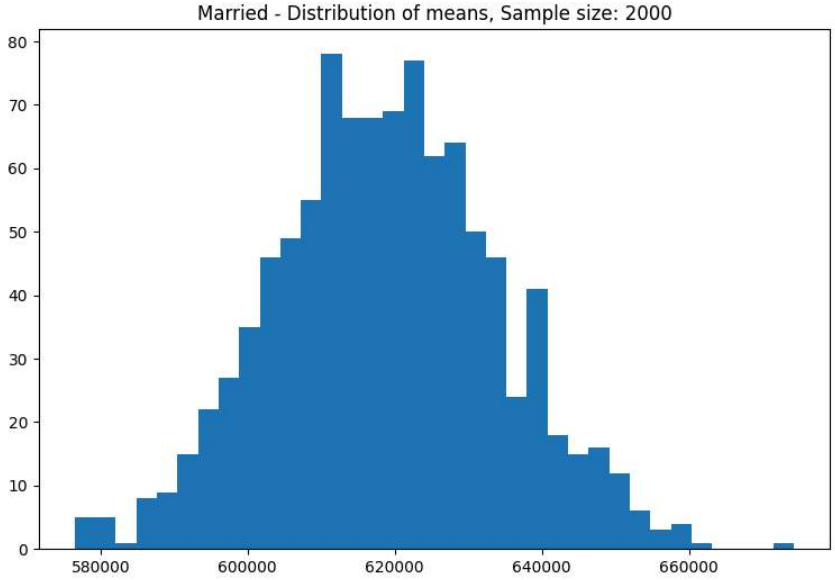
```
Married customer average spent amount: 619412.9915117219
Unmarried customer average spent amount: 645074.4536142815
```

In [ ]:

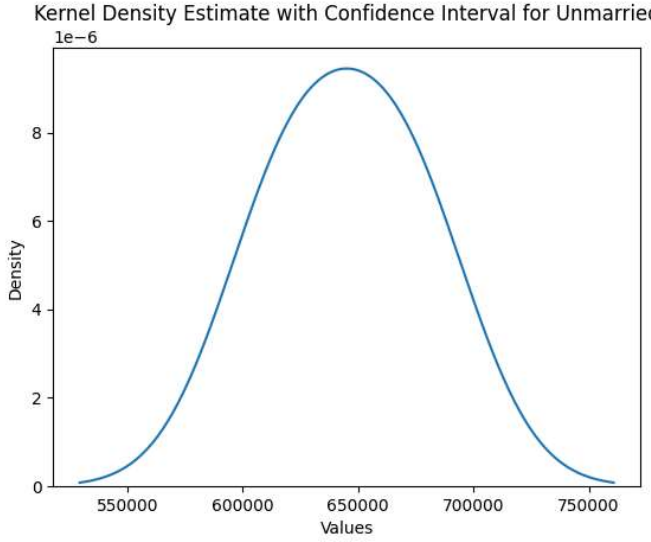
```
import scipy.stats as stats

Unmarried_df = sum_by_Marital_Status[sum_by_Marital_Status['Marital_Status']=='Unmarried']
Married_df = sum_by_Marital_Status[sum_by_Marital_Status['Marital_Status']=='Married']
# Taking random sample size from dataframe
Unmarried_sample_size = 3000
Married_sample_size = 2000
num_repitions = 1000
# Taking random sample from unmarried and married dataframe
random_sample_Unmarried = Unmarried_df.sample(n=Unmarried_sample_size)
random_sample_Married = Married_df.sample(n=Married_sample_size)
# Taking mean value from random sample unmarried and married dataframe
Unmarried_means = random_sample_Unmarried['Purchase'].mean()
print(f'Population mean: random Unmarried samples mean purchase value: {Unmarried_means}')
Married_means = random_sample_Married['Purchase'].mean()
print(f'Population mean: random Married samples mean purchase value : {Married_means}')
# Taking sample mean from filtered unmarried dataframe
Unmarried_sample_mean = round(Unmarried_df['Purchase'].mean(),2)
print(f'Sample means of Unmarried purchase : {Unmarried_sample_mean}')
Unmarried_std_value = round(Unmarried_df['Purchase'].std(),2)
print(f'Sample STD of Unmarried purchase : {Unmarried_std_value}')
# Taking sample mean from filtered Married dataframe
Married_sample_mean = round(Married_df['Purchase'].mean(),2)
print(f'Sample means of Married purchase : {Married_sample_mean}')
Married_std_value = round(Married_df['Purchase'].std(),2)
print(f'Sample STD of Married purchase : {Married_std_value}')
# taking blank list to creat histogram
Unmarried_means1 = []
Married_means1 = []
# using for loop to create again mean value for histogram
for _ in range(num_repitions):
    Unmarried_mean2 = Unmarried_df.sample(Unmarried_sample_size,replace=True)['Purchase'].mean()
    Married_mean2 = Married_df.sample(Married_sample_size,replace=True)['Purchase'].mean()
    Unmarried_means1.append(Unmarried_mean2)
    Married_means1.append(Married_mean2)
fig, axis = plt.subplots(nrows=1, ncols=2, figsize=(20, 6))
axis[0].hist(Unmarried_means1, bins=35)
axis[1].hist(Married_means1, bins=35)
axis[0].set_title("Unmarried - Distribution of means, Sample size: 3000")
axis[1].set_title("Married - Distribution of means, Sample size: 2000")
plt.show()
sample_size = 3000
# Confidence level ( 95% confidence interval)
confidence_level = 0.95
# Calculate the margin of error using the z-distribution for male
z_critical = stats.norm.ppf((1 + confidence_level) / 2) # Z-score for the desired confidence level
margin_of_error = z_critical * (Unmarried_std_value / np.sqrt(sample_size))
# Calculate the margin of error using the z-distribution for female
z_critical = stats.norm.ppf((1 + confidence_level) / 2) # Z-score for the desired confidence level
margin_of_error = z_critical * (Married_std_value / np.sqrt(sample_size))
# Calculate the confidence interval for Unmarried and presenting it on the graph
Unmarried_confidence_interval = (Unmarried_sample_mean - margin_of_error, Unmarried_sample_mean + margin_of_error)
print("Confidence Interval 95% Unmarried:", Unmarried_confidence_interval)
sns.kdeplot(Unmarried_confidence_interval)
plt.xlabel('Values')
plt.ylabel('Density')
plt.title('Kernel Density Estimate with Confidence Interval for Unmarried')
plt.show()

Married_confidence_interval = (Married_sample_mean - margin_of_error, Married_sample_mean + margin_of_error)
print("Confidence Interval 95% Married:", Married_confidence_interval)
sns.kdeplot(Married_confidence_interval)
plt.xlabel('Values')
plt.ylabel('Density')
plt.title('Kernel Density Estimate with Confidence Interval for Married')
plt.show()
```

```
Population mean: random Unmarried samples mean purchase value: 648522.4063333333
Population mean: random Married samples mean purchase value : 615529.9475
Sample means of Unmarried purchase : 645074.45
Sample STD of Unmarried purchase : 698938.43
Sample means of Married purchase : 619412.99
Sample STD of Married purchase : 689234.66
```

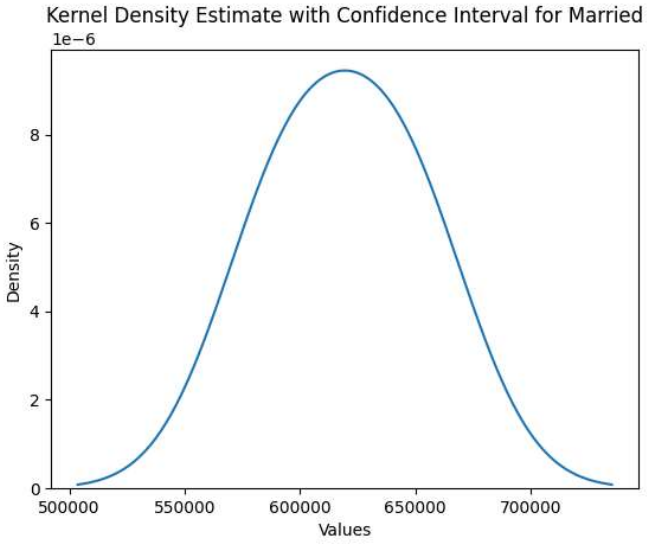





```
Confidence Interval 95% Unmarried: (620410.9576536223, 669737.9423463776)
```




```
Confidence Interval 95% Married: (594749.4976536223, 644076.4823463777)
```



Insight

1. With reference to the above data, at a 95% confidence interval:

a) The average amount spent by an unmarried customer will lie between (620410.95, 669737.94).

b) The average amount spent by a married customer will lie between (594749.49, 644076.48).

2. Confidence intervals for average unmarried and married spending are overlapping.

3. With respect to the above data, company should target more unmarried customers.

Business Recommendations:

1. Focus marketing efforts on individuals in the age group '26-35', as they demonstrate the highest purchase counts. Tailor promotions and advertisements to resonate with this demographic.
2. Since Occupation '4' has the highest representation and notable purchases, consider customizing product offerings or promotions to cater specifically to individuals in this occupation.
3. The male customers spend more on average than the female customers, so Walmart should reward the customers who spend more so that male customers continue to spend more and it also encourages female customers on spendign more. Additional discounts can also be provided for female customers
4. Design promotions or incentives based on the top occupations, such as '0' and '4', to further boost sales from these occupational groups.
5. Analyze the product preferences of male customers to inform product development. Ensure that the product range aligns with the preferences of the larger male customer base, leading to increased sales.
6. After Black Friday, walmart should engage with customers who made purchases by sending follow-up emails or offers for related products. This can help increase customer retention and encourage repeat business throughout the holiday season and beyond. While Company is making shopping better for different groups, it's important to make sure that everyone has a good shopping experience, no matter who they are.