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)

In [252...

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import seaborn as sns
%matplotlib inline
sns.set(color_codes=True)
import warnings
warnings.filterwarnings('ignore')
import copy
```

In [253...

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

In [254...

```
# shape of data
df.shape
```

Out[254...

```
(550068, 10)
```

In [255...

```
print("No. of Rows = ", df.shape[0])
```

```
No. of Rows =  550068
```

In [256...

```
print("No. of Columns = ", df.shape[1])
```

```
No. of Columns =  10
```

In [257...

```
# columns present in data
df.columns
```

Out[257...

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

In [258...

```
# data types of columns
df.dtypes
```

Out[258…

```
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 [259…

```
df.head()
```

Out[259…

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Year
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	
3	1000001	P00085442	F	0-17	10	A	
4	1000002	P00285442	M	55+	16	C	4

In [260…

```
df.tail()
```

Out[260…

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_Cit
550063	1006033	P00372445	M	51-55	13	B	
550064	1006035	P00375436	F	26-35	1	C	
550065	1006036	P00375436	F	26-35	15	B	
550066	1006038	P00375436	F	55+	1	C	
550067	1006039	P00371644	F	46-50	0	B	

In [261…

```
# checking for missing or null values

df.isnull().sum()
```

Out[261…
```
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
```

No null values are present in the column.

In [262…
```
# checking for duplicated values

df[df.duplicated()]
```

Out[262…

User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years

The given data does not have any duplicated values.

In [263…
```
# information about dataframe
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 [264…
```
# Converting User_ID column datatype to int32
df['User_ID'] = df['User_ID'].astype('int32')
```

In [265…
```
# Updating 'Marital_Status' column
df['Marital_Status'] = df['Marital_Status'].apply(lambda x: 'Married' if x == 1 els
```

In [266…
```
df['Marital_Status'] = df['Marital_Status'].astype('category')
```

In [267…
```
# Converting 'Age' column datatype to category
df['Age'] = df['Age'].astype('category')
```

In [268…
```
# Converting 'Product_Category' column datatype to int8
df['Product_Category'] = df['Product_Category'].astype('int8')
```

In [269…
```
# Converting 'Product_Category' column datatype to int8
df['Occupation'] = df['Occupation'].astype('int8')
```

In [270…
```
# Converting 'City_Category' column's datatype to category
df['City_Category'] = df['City_Category'].astype('category')
```

In [271…
```
# Converting 'Stay_In_Current_City_Years' column's datatype to category
df['Stay_In_Current_City_Years'] = df['Stay_In_Current_City_Years'].astype('categor
```

In [272…
```
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  int32
 1   Product_ID                  550068 non-null  object
 2   Gender                      550068 non-null  object
 3   Age                         550068 non-null  category
 4   Occupation                  550068 non-null  int8
 5   City_Category               550068 non-null  category
 6   Stay_In_Current_City_Years  550068 non-null  category
 7   Marital_Status              550068 non-null  category
 8   Product_Category            550068 non-null  int8
 9   Purchase                    550068 non-null  int64
dtypes: category(4), int32(1), int64(1), int8(2), object(2)
memory usage: 17.8+ MB
```

I have done some memory utilization here.The memory usage of the dataframe is reduced to 17.8+ MB from 42.0+ MB approx 58% reduction in the memory usage.

Basic statistical description of the dataframe

In [273…
```
df.describe(include="all")
```

Out[273...

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

- **There are 5891 unique users, and userid 1001680 being with the highest count.**
- **There are 3631 unique products in the data.**
- **City is divided into 3 unique groups.**
- **Age is divided into 7 unique bins.**
- **Out of 550068 data 414259 are male. It suggests that male purchase count is higher than female.**
- **People of age group 26-35 have most purchase count.**
- **People who made the most purchase are from city B.**
- **The most used product is having the product id P00265242.**
- **There is a huge difference between 75% percentile value and max value for Purchase column. So there might be outliers present in this column.**
- **Minimum & Maximum purchase is 12 and 23961 suggests the purchasing behaviour is quite spread over a aignificant range of values. Mean is 9264 and 75% of purchase is of less than or equal to 12054. It suggests most of the purchase is not more than 12000.**
- **There are 21 unique occupations in which people are involved.**
- **Mostly single people have made the purchase because the frequency count for single is high.**

NON VISUAL ANALYSIS

VALUE COUNTS & UNIQUE VALUES

In [274…

```
# How many unique customers' data is given in the dataset?
df['User_ID'].nunique()
```

Out[274…

```
5891
```

In [275…

```
# gender value counts
df['Gender'].value_counts()
```

Out[275…

```
M    414259
F    135809
Name: Gender, dtype: int64
```

In [276…

```
np.round(df['Occupation'].value_counts(normalize = True) * 100, 2).cumsum()
```

Out[276…

```
4     13.15
0     25.81
7     36.56
1     45.18
17    52.46
20    58.56
12    64.23
14    69.19
2     74.02
16    78.63
6     82.33
3     85.54
10    87.89
5     90.10
15    92.31
11    94.42
19    95.96
13    97.36
18    98.56
9     99.70
8     99.98
Name: Occupation, dtype: float64
```

It can be inferred from the above that 82.33%of the total transactions are made by the customers belonging to 11 occupations. These are 4, 0, 7, 1, 17, 20, 12, 14, 2, 16, 6 (Ordered in descending order of the total transactions' share.)

In [277…

```
np.round(df['Stay_In_Current_City_Years'].value_counts(normalize = True) * 100, 2)
```

Out[277…

```
1     35.24
2     18.51
3     17.32
4+    15.40
0     13.53
Name: Stay_In_Current_City_Years, dtype: float64
```

From the above result, it is clear that majority of the transactions (53.75% of total transactions) are made by the customers having 1 or 2 years of stay in the current city.

In [278…
```
np.round(df['Product_Category'].value_counts(normalize = True).head(10) * 100, 2).c
```

Out[278…
```
5     27.44
1     52.96
8     73.67
11    78.09
2     82.43
6     86.15
3     89.82
4     91.96
16    93.75
15    94.89
Name: Product_Category, dtype: float64
```

It can be inferred from the above result that 82.43% of the total transactions are made for only 5 Product Categories. These are, 5, 1, 8, 11 and 2.

In [279…
```
# No. of unique customers for each gender

df_gender_dist = pd.DataFrame(df.groupby(by = ['Gender'])['User_ID'].nunique()).res
df_gender_dist['percent_share'] = np.round(df_gender_dist['unique_customers'] / df_
df_gender_dist
```

Out[279…

	Gender	unique_customers	percent_share
0	F	1666	28.28
1	M	4225	71.72

In [280…
```
# total revenue from each gender

df_gender_revenue = df.groupby(by = ['Gender'])['Purchase'].sum().to_frame().sort_v
df_gender_revenue['percent_share'] = np.round((df_gender_revenue['Purchase'] / df_g
df_gender_revenue
```

Out[280…

	Gender	Purchase	percent_share
0	M	3909580100	76.72
1	F	1186232642	23.28

In [281…
```
# the average total purchase made by each user in each gender

df1 = pd.DataFrame(df.groupby(by = ['Gender', 'User_ID'])['Purchase'].sum()).reset_
df1.groupby(by = 'Gender')['Average_Purchase'].mean()
```

Out[281…
```
Gender
F    712024.394958
M    925344.402367
Name: Average_Purchase, dtype: float64
```

- **On an average each male makes a total purchase of 712024.394958.**

- **On an average each female makes a total purchase of 925344.402367.**

In [282…

```
# the average Revenue generated by Walmart from each Gender per transaction
pd.DataFrame(df.groupby(by = 'Gender')['Purchase'].mean()).reset_index().rename(col
```

Out[282…

	Gender	Average_Purchase
0	F	8734.565765
1	M	9437.526040

In [283…

```
# customers according to martial status
df_marital_status_dist = pd.DataFrame(df.groupby(by = ['Marital_Status'])['User_ID'
df_marital_status_dist['percent_share'] = np.round(df_marital_status_dist['unique_c
df_marital_status_dist
```

Out[283…

	Marital_Status	unique_customers	percent_share
0	Married	2474	42.0
1	Single	3417	58.0

In [284…

```
# transactions according to martial status
df.groupby(by = ['Marital_Status'])['User_ID'].count()
```

Out[284…

```
Marital_Status
Married    225337
Single     324731
Name: User_ID, dtype: int64
```

In [285…

```
print('Average number of transactions made by each user with marital status Married
print('Average number of transactions made by each with marital status Single is',
```

```
Average number of transactions made by each user with marital status Married is 91
Average number of transactions made by each with marital status Single is 95
```

In [286…

```
#the total Revenue generated by Walmart from each Marital Status

df_marital_status_revenue = df.groupby(by = ['Marital_Status'])['Purchase'].sum().t
df_marital_status_revenue['percent_share'] = np.round((df_marital_status_revenue['P
df_marital_status_revenue
```

Out[286…

	Marital_Status	Purchase	percent_share
0	Single	3008927447	59.05
1	Married	2086885295	40.95

In [287…

```
# the average total purchase made by each user in each marital status
df1 = pd.DataFrame(df.groupby(by = ['Marital_Status', 'User_ID'])['Purchase'].sum()
df1.groupby(by = 'Marital_Status')['Average_Purchase'].mean()
```

Out[287…

```
Marital_Status
Married    843526.796686
Single     880575.781972
Name: Average_Purchase, dtype: float64
```

- **On an average each Married customer makes a total purchase of 843526.796686.**
- **On an average each Single customer makes a total purchase of 880575.781972.**

In [288…

```
df_age_dist = pd.DataFrame(df.groupby(by = ['Age'])['User_ID'].nunique()).reset_ind
df_age_dist['percent_share'] =  np.round(df_age_dist['unique_customers'] / df_age_d
df_age_dist['cumulative_percent'] =  df_age_dist['percent_share'].cumsum()
df_age_dist
```

Out[288…

	Age	unique_customers	percent_share	cumulative_percent
2	26-35	2053	34.85	34.85
3	36-45	1167	19.81	54.66
1	18-25	1069	18.15	72.81
4	46-50	531	9.01	81.82
5	51-55	481	8.16	89.98
6	55+	372	6.31	96.29
0	0-17	218	3.70	99.99

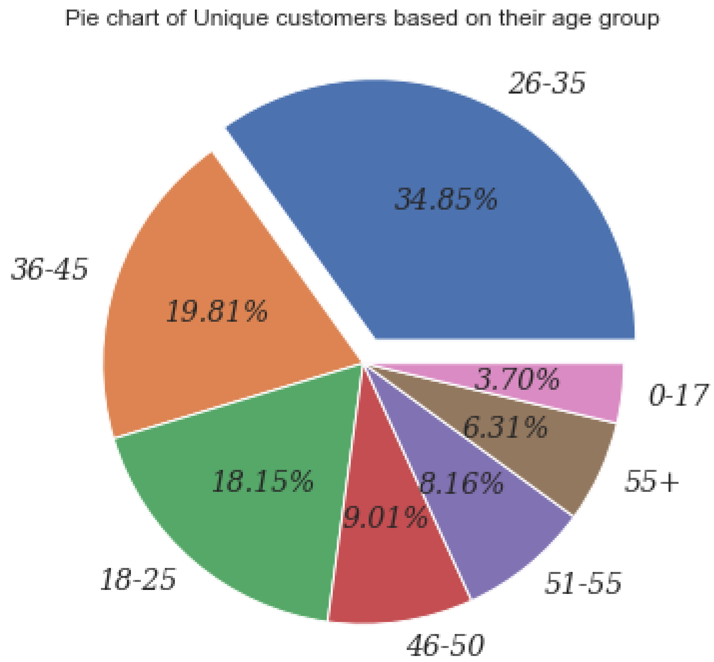
- **Majority of the transactions are made by the customers between 26 and 45 years of age.**
- **About 81.82% of the total transactions are made by customers of age between 18 and 50 years.**

VISUAL ANALYSIS

UNIVARIATE & BIVARIATE ANALYSIS

In [289…

```
plt.figure(figsize = (8, 6))
plt.title('Pie chart of Unique customers based on their age group')
plt.pie(x = df_age_dist['percent_share'], labels = df_age_dist['Age'],
        explode = [0.1] + [0] * 6, autopct = '%.2f%%',
       textprops = {'fontsize' : 14,
                   'fontstyle' : 'oblique',
                   'fontfamily' : 'serif',
                   'fontweight' : 500})
plt.show()
```



In [290…

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

Out[290…

```
26-35    219587
36-45    110013
18-25     99660
46-50     45701
51-55     38501
55+       21504
0-17      15102
Name: Age, dtype: int64
```

In [291…

```
df_age_revenue = pd.DataFrame(df.groupby(by = 'Age', as_index = False)['Purchase'].
df_age_revenue['percent_share'] = np.round((df_age_revenue['Purchase'] / df_age_rev
df_age_revenue['cumulative_percent_share'] = df_age_revenue['percent_share'].cumsum
df_age_revenue
```

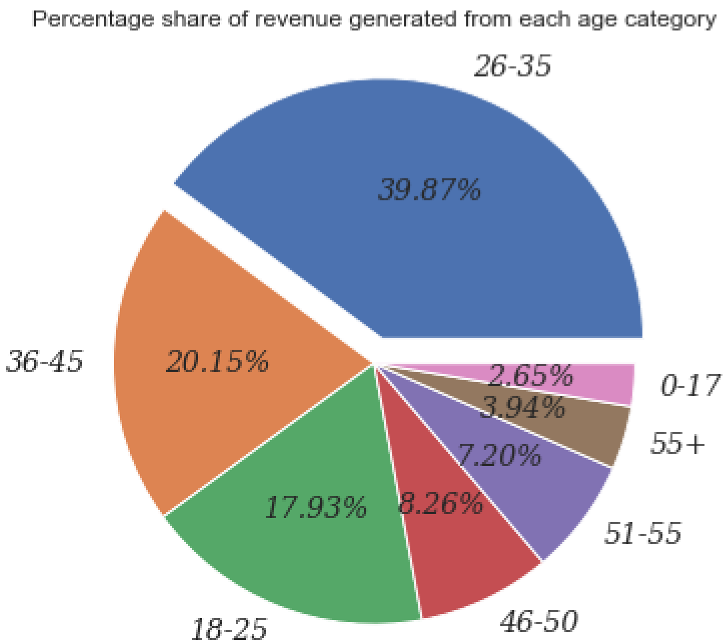
Out[291…

	Age	Purchase	percent_share	cumulative_percent_share
2	26-35	2031770578	39.87	39.87
3	36-45	1026569884	20.15	60.02
1	18-25	913848675	17.93	77.95
4	46-50	420843403	8.26	86.21
5	51-55	367099644	7.20	93.41
6	55+	200767375	3.94	97.35
0	0-17	134913183	2.65	100.00

In [292…

```
plt.figure(figsize = (8, 6))
plt.title('Percentage share of revenue generated from each age category')
plt.pie(x = df_age_revenue['percent_share'], labels = df_age_revenue['Age'],
```

```
        explode = [0.1] + [0] * 6, autopct = '%.2f%%',
        textprops = {'fontsize' : 14,
                     'fontstyle' : 'oblique',
                     'fontfamily' : 'serif',
                     'fontweight' : 500})
plt.show()
```



In [293...

```
df_city_dist = pd.DataFrame(df.groupby(by = ['City_Category'])['User_ID'].nunique()
df_city_dist['percent_share'] = np.round((df_city_dist['unique_customers'] / df_cit
df_city_dist['cumulative_percent_share'] = df_city_dist['percent_share'].cumsum()
df_city_dist
```

Out[293...

	City_Category	unique_customers	percent_share	cumulative_percent_share
0	A	1045	17.74	17.74
1	B	1707	28.98	46.72
2	C	3139	53.28	100.00

- **Majority of the total unique customers belong to the city C.**
- **82.26%of the total unique customers belong to city C and B.**

In [294...

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

Out[294...

```
B    231173
C    171175
A    147720
Name: City_Category, dtype: int64
```

In [295...

```
# average revenue from different cities
```

```
df_city_revenue = df.groupby(by = ['City_Category'])['Purchase'].sum().to_frame().s
df_city_revenue['percent_share'] = np.round((df_city_revenue['Purchase'] / df_city_
df_city_revenue['cumulative_percent_share'] = df_city_revenue['percent_share'].cums
df_city_revenue
```

Out[295…

	City_Category	Purchase	percent_share	cumulative_percent_share
0	B	2115533605	41.52	41.52
1	C	1663807476	32.65	74.17
2	A	1316471661	25.83	100.00

In [296…

```
df.groupby(by = ['Product_Category'])['Product_ID'].nunique()
```

Out[296…

```
Product_Category
1      493
2      152
3       90
4       88
5      967
6      119
7      102
8     1047
9        2
10      25
11     254
12      25
13      35
14      44
15      44
16      98
17      11
18      30
19       2
20       3
Name: Product_ID, dtype: int64
```

In [297…

```
# revenue from differenr product categories

df_product_revenue = df.groupby(by = ['Product_Category'])['Purchase'].sum().to_fra
df_product_revenue['percent_share'] = np.round((df_product_revenue['Purchase'] / df
df_product_revenue['cumulative_percent_share'] = df_product_revenue['percent_share'
df_product_revenue
```

Out[297…

	Product_Category	Purchase	percent_share	cumulative_percent_share
0	1	1910013754	37.48	37.48
1	5	941835229	18.48	55.96
2	8	854318799	16.77	72.73
3	6	324150302	6.36	79.09
4	2	268516186	5.27	84.36
5	3	204084713	4.00	88.36
6	16	145120612	2.85	91.21
7	11	113791115	2.23	93.44
8	10	100837301	1.98	95.42
9	15	92969042	1.82	97.24
10	7	60896731	1.20	98.44
11	4	27380488	0.54	98.98
12	14	20014696	0.39	99.37
13	18	9290201	0.18	99.55
14	9	6370324	0.13	99.68
15	17	5878699	0.12	99.80
16	12	5331844	0.10	99.90
17	13	4008601	0.08	99.98
18	20	944727	0.02	100.00
19	19	59378	0.00	100.00

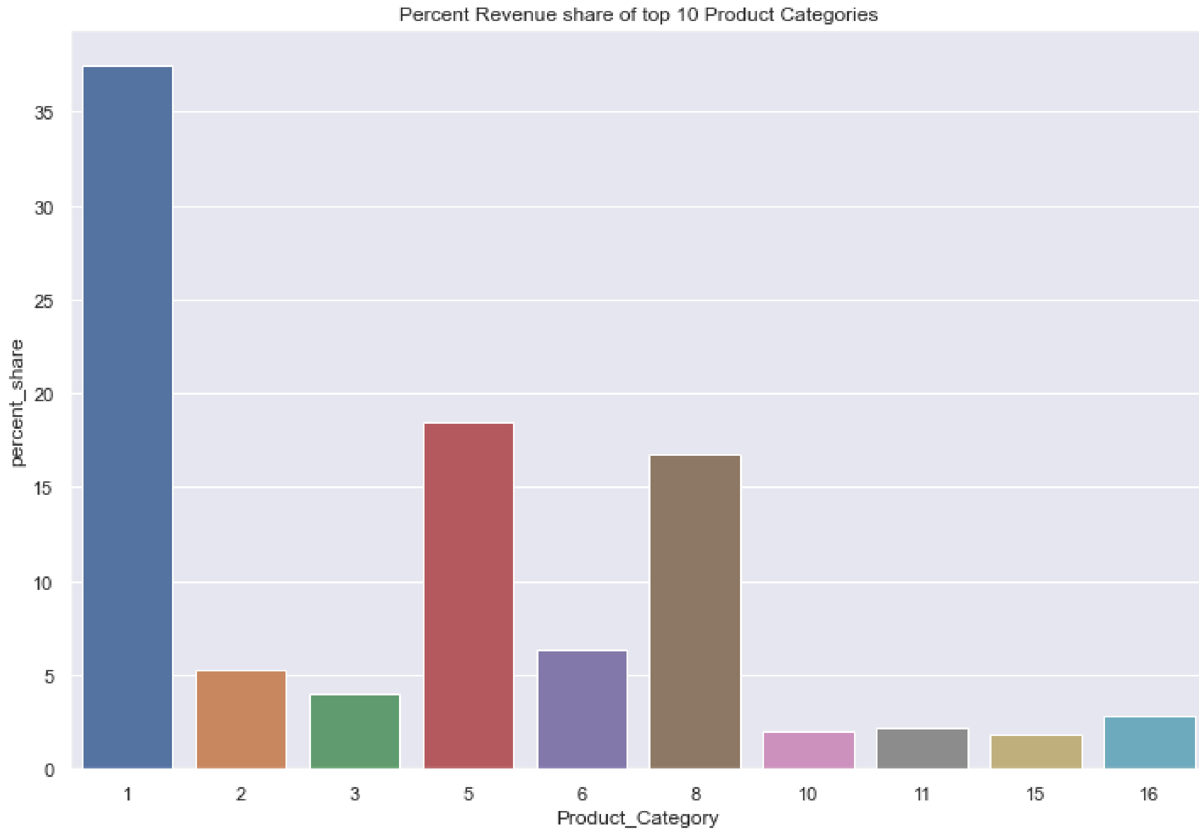
In [298…

```
top5 = df_product_revenue.head(5)['Purchase'].sum() /  df_product_revenue['Purchase
top5 = np.round(top5 * 100, 2)
print(f'Top 5 product categories from which Walmart makes {top5} % of total revenue
```

```
Top 5 product categories from which Walmart makes 84.36 % of total revenue are : [1,
5, 8, 6, 2]
```

In [299…

```
plt.figure(figsize = (12, 8))
plt.title('Percent Revenue share of top 10 Product Categories')
sns.barplot(data = df_product_revenue, x = df_product_revenue.head(10)['Product_Cat
plt.show()
```



What is the total Revenue generated by Walmart from each Gender ?

In [300…

```
# total revenue generated by Walmart from each gender.

df_gender_revenue = df.groupby(by = ['Gender'])['Purchase'].sum().to_frame().sort_v
df_gender_revenue['percent_share'] = np.round((df_gender_revenue['Purchase'] / df_g
df_gender_revenue
```

Out[300…

	Gender	Purchase	percent_share
0	M	3909580100	76.72
1	F	1186232642	23.28

What is the Average Revenue generated by Walmart from each Gender per transaction ?

In [301…

```
# average revenue from each gender per transaction

pd.DataFrame(df.groupby(by = 'Gender')['Purchase'].mean()).reset_index().rename(col
```

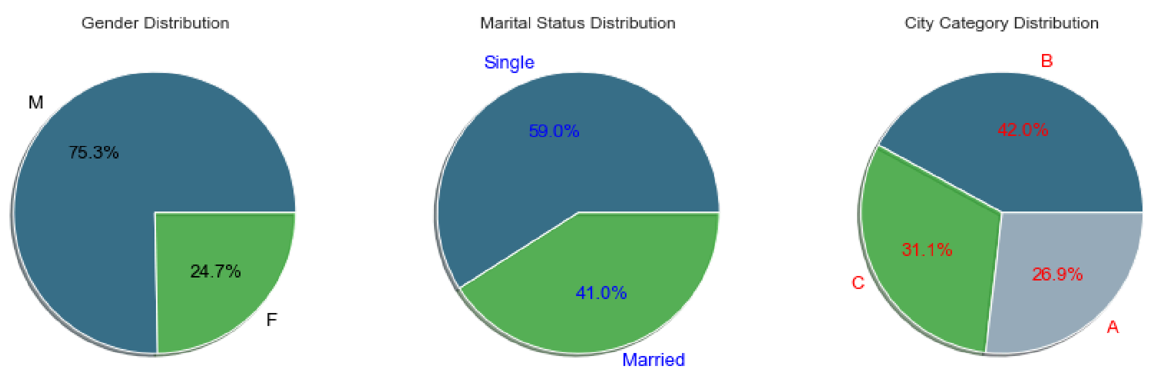
Out[301…

	Gender	Average_Purchase
0	F	8734.565765
1	M	9437.526040

Gender, Marital Status and City Category Distribution

In [302…

```
# creating pie chart for gender disribution
fig = plt.figure(figsize = (15,12))
gs = fig.add_gridspec(1,3)


ax0 = fig.add_subplot(gs[0,0])

color_map = ["#3A7089", "#4b4b"]
ax0.pie(df['Gender'].value_counts().values,labels = df['Gender'].value_counts().ind
        shadow = True,colors = color_map,textprops={'fontsize': 13, 'color': 'black

#setting title for visual
ax0.set_title('Gender Distribution')

# creating pie chart for marital status
ax1 = fig.add_subplot(gs[0,1])

color_map = ["#3A7089", "#4b4b"]
ax1.pie(df['Marital_Status'].value_counts().values,labels = df['Marital_Status'].va
        shadow = True,colors = color_map,textprops={'fontsize': 13, 'color': 'blue'

#setting title for visual
ax1.set_title('Marital Status Distribution')

# creating pie chart for city category
ax1 = fig.add_subplot(gs[0,2])

color_map = ["#3A7089", "#4b4b",'#99AEBB']
ax1.pie(df['City_Category'].value_counts().values,labels = df['City_Category'].valu
        shadow = True,colors = color_map,textprops={'fontsize': 13, 'color': 'red'}

#setting title for visual
ax1.set_title('City Category Distribution')
plt.show()
```

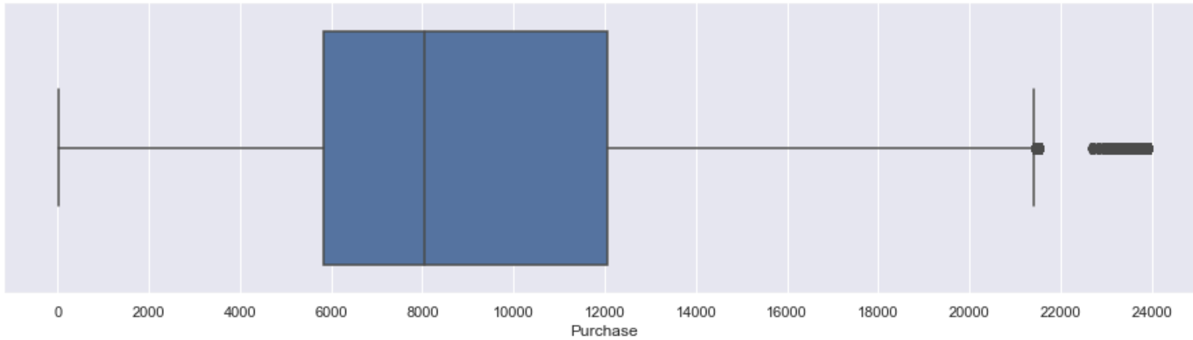


OUTLIER HANDLING

In [303…

```
# outlier checking
plt.figure(figsize = (16, 4))
sns.boxplot(data = df,
            x = 'Purchase')
```

```
plt.xticks(np.arange(0, 25001, 2000))
plt.show()
```



In [304…

```
df1=df.copy()
```

In [305…

```
q1=df1['Purchase'].quantile(0.25)
q3=df1['Purchase'].quantile(0.75)
print('The first quantile is',q1)
print('The third quantile is',q3)
```

```
The first quantile is 5823.0
The third quantile is 12054.0
```

In [306…

```
iqr=q3 - q1
print(iqr)
```

```
6231.0
```

In [307…

```
lower = q1-(1.5)*iqr
upper = q3+(1.5)*iqr
print('The lower limit for outliers are',lower)
print('The upper limit for outliers are',upper)
```

```
The lower limit for outliers are -3523.5
The upper limit for outliers are 21400.5
```

In [308…

```
outliers = df1[(df1['Purchase']<lower)|(df1['Purchase']>upper)]
outliers.head()
```

Out[308...

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_`
343	1000058	P00117642	M	26-35	2	B	
375	1000062	P00119342	F	36-45	3	A	
652	1000126	P00087042	M	18-25	9	B	
736	1000139	P00159542	F	26-35	20	C	
1041	1000175	P00052842	F	26-35	2	B	

In [309...

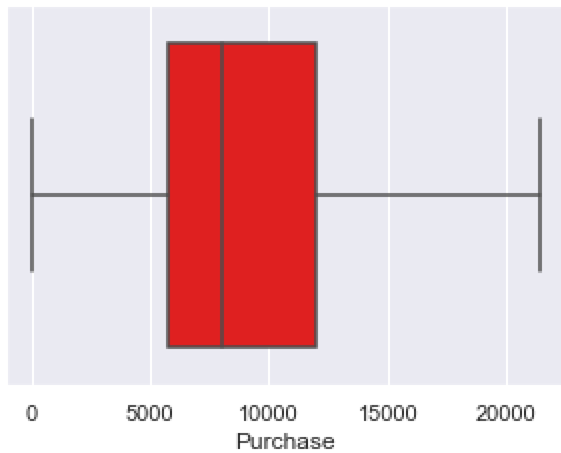
```
purchase = df1[~((df1['Purchase']<lower)|(df1['Purchase']>upper))]
purchase.head()
```

Out[309...

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Year
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	
3	1000001	P00085442	F	0-17	10	A	
4	1000002	P00285442	M	55+	16	C	4

In [310...

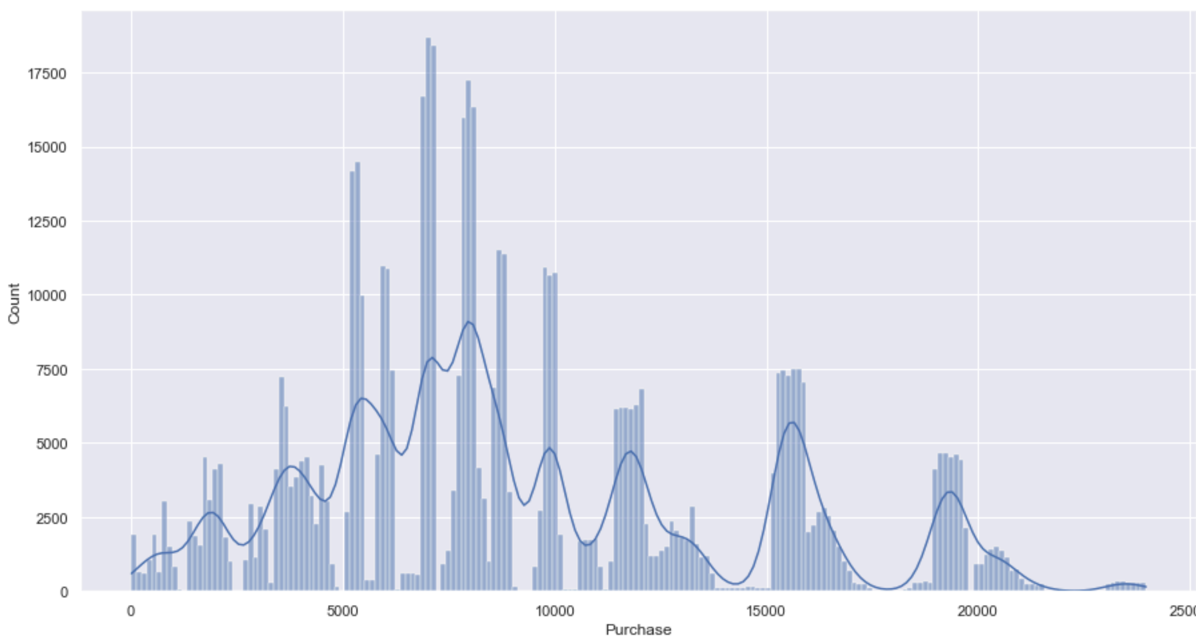
```
plt.figure(figsize=(5,3.5))
sns.boxplot(x ='Purchase', data = purchase, color="red")
plt.show()
```



No outliers are now present in the above boxplot.

In [311…

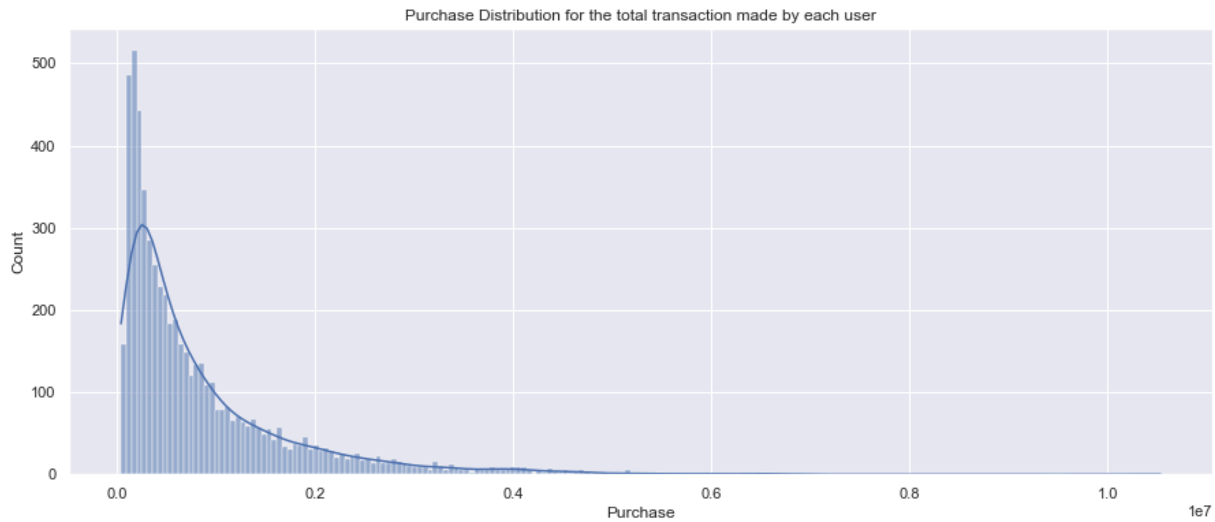
```
plt.figure(figsize = (15, 8))
sns.histplot(data = df, x = 'Purchase', kde = True, bins = 200)
plt.show()
```



In [312…

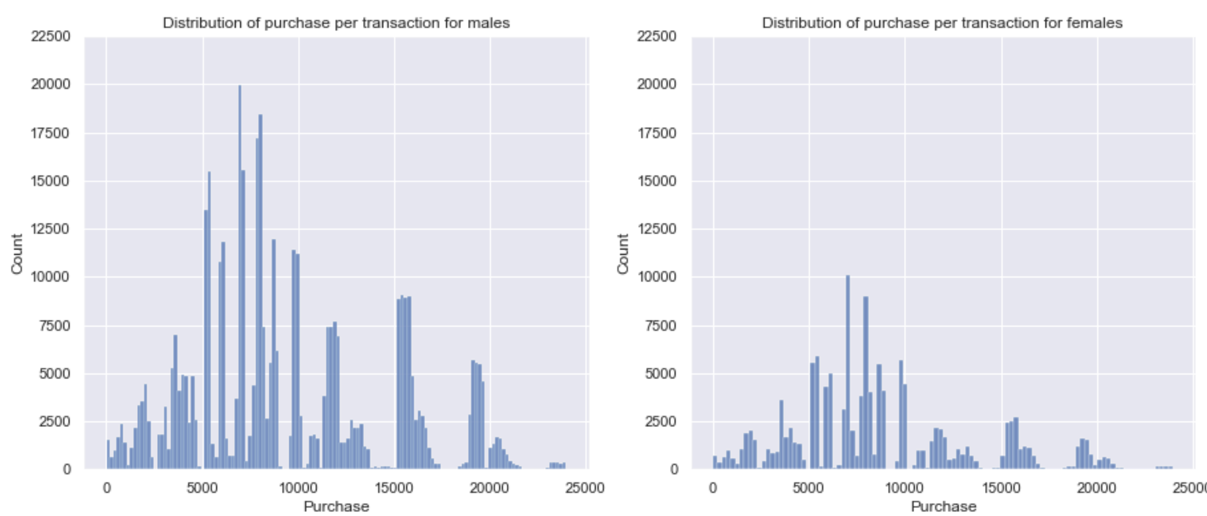
```
plt.figure(figsize = (15, 6))
plt.title('Purchase Distribution for the total transaction made by each user')
df_customer = df.groupby(by = 'User_ID')['Purchase'].sum()
sns.histplot(data = df_customer, kde = True, bins = 200)
plt.plot()
```

Out[312… []



In [313…

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.title('Distribution of purchase per transaction for males')
df_male = df[df['Gender'] == 'M']
sns.histplot(data = df_male, x = 'Purchase')
plt.yticks(np.arange(0, 22550, 2500))
plt.subplot(1, 2 ,2)
plt.title('Distribution of purchase per transaction for females')
df_female = df[df['Gender'] == 'F']
sns.histplot(data = df_female, x = 'Purchase')
plt.yticks(np.arange(0, 22550, 2500))
plt.show()
```



In [314…

```
df_cust_gender = pd.DataFrame(df.groupby(by = ['Gender', 'User_ID'])['Purchase'].su
df_cust_gender
```

Out[314…

	Gender	User_ID	Total_Purchase
0	F	1000001	334093
1	F	1000006	379930
2	F	1000010	2169510
3	F	1000011	557023
4	F	1000016	150490
...	...	...	...
5886	M	1006030	737361
5887	M	1006032	517261
5888	M	1006033	501843
5889	M	1006034	197086
5890	M	1006040	1653299

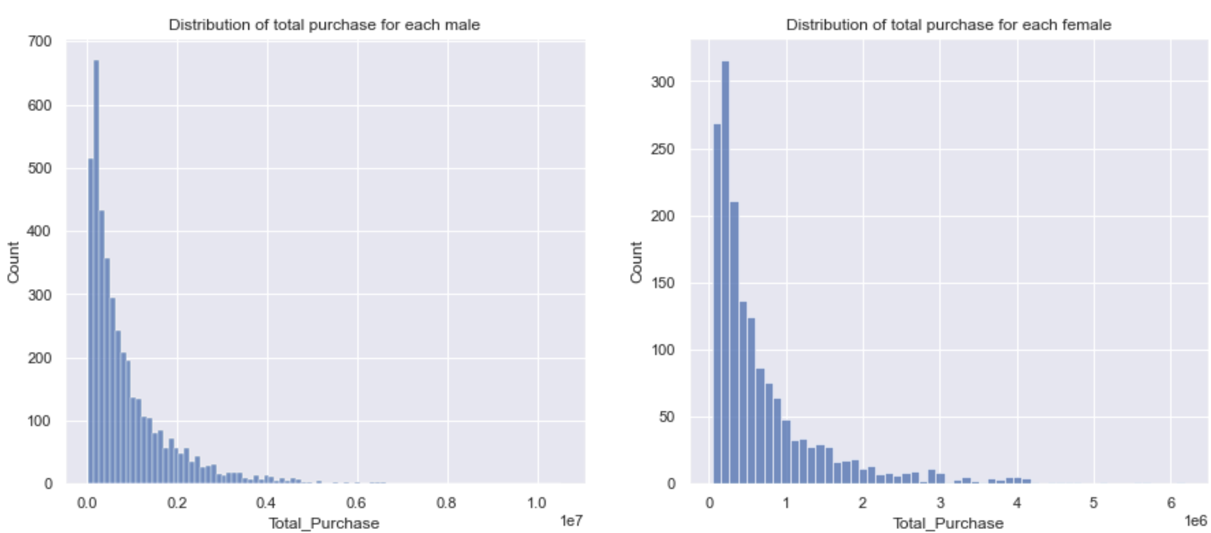
5891 rows × 3 columns

In [315…

```
df_male_customer = df_cust_gender.loc[df_cust_gender['Gender'] == 'M']
df_female_customer = df_cust_gender.loc[df_cust_gender['Gender'] == 'F']
```

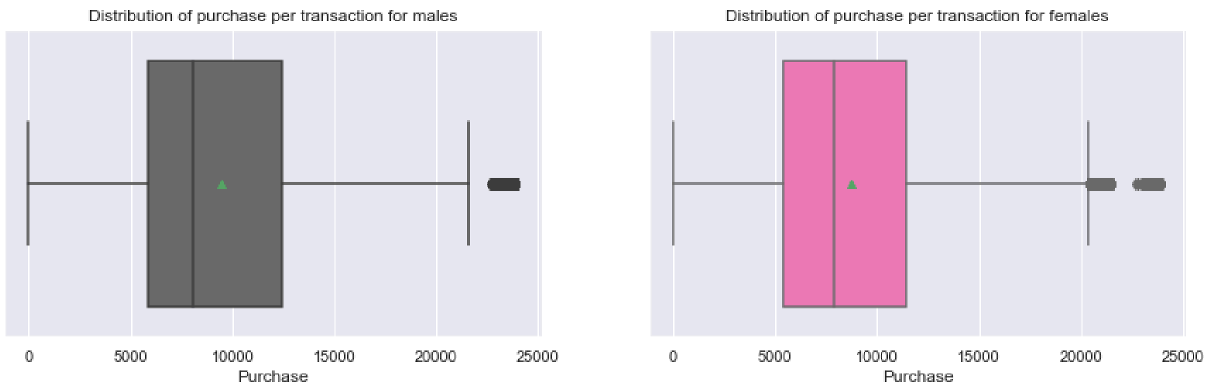
In [316…

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.title('Distribution of total purchase for each male')
sns.histplot(data = df_male_customer, x = 'Total_Purchase')
plt.subplot(1, 2 ,2)
plt.title('Distribution of total purchase for each female')
df_female = df[df['Gender'] == 'F']
sns.histplot(data = df_female_customer, x = 'Total_Purchase')
plt.show()
```



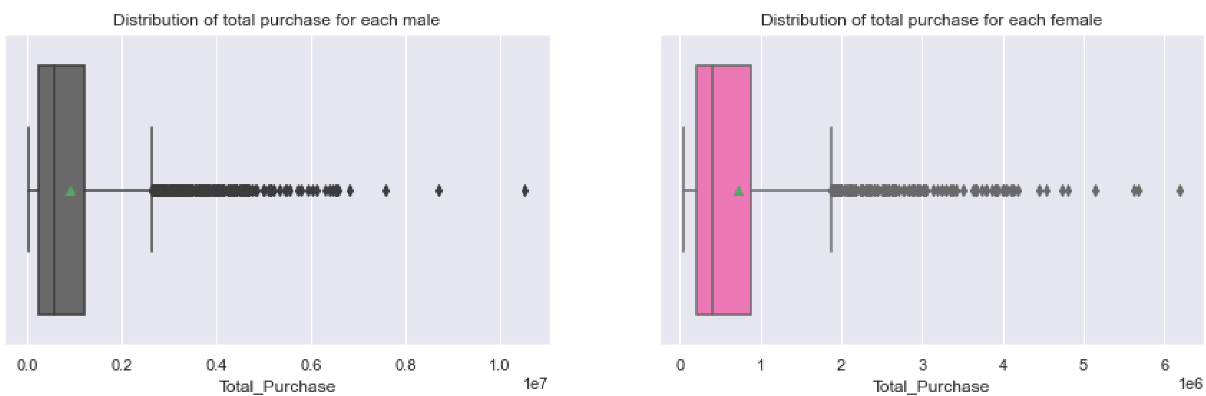
In [317…

```
plt.figure(figsize = (15, 4))
plt.subplot(1, 2, 1)
plt.title('Distribution of purchase per transaction for males')
sns.boxplot(data = df_male, x = 'Purchase', showmeans = True, color = 'dimgray')
plt.subplot(1, 2 ,2)
plt.title('Distribution of purchase per transaction for females')
sns.boxplot(data = df_female, x = 'Purchase', showmeans = True, color = 'hotpink')
plt.show()
```



In [318…

```
plt.figure(figsize = (15, 4))
plt.subplot(1, 2, 1)
plt.title('Distribution of total purchase for each male')
sns.boxplot(data = df_male_customer, x = 'Total_Purchase', showmeans = True, color
plt.subplot(1, 2 ,2)
plt.title('Distribution of total purchase for each female')
sns.boxplot(data = df_female_customer, x = 'Total_Purchase', showmeans = True, colo
plt.show()
```



In [319…

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

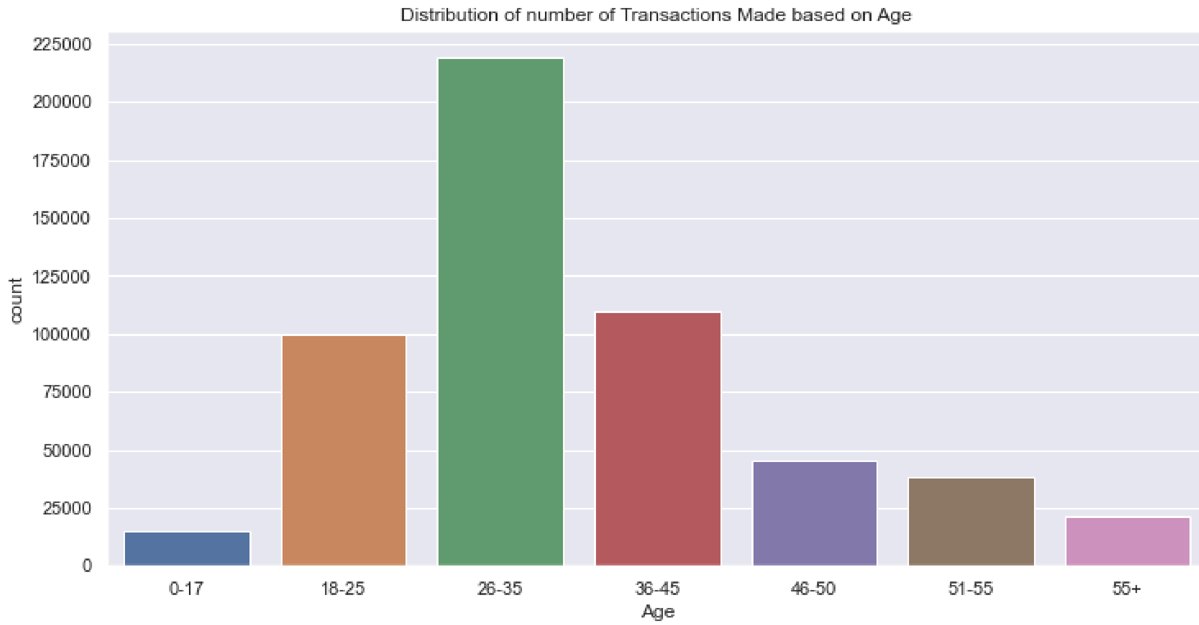
Out[319…

```
['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25']
Categories (7, object): ['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-2
5']
```

In [320…

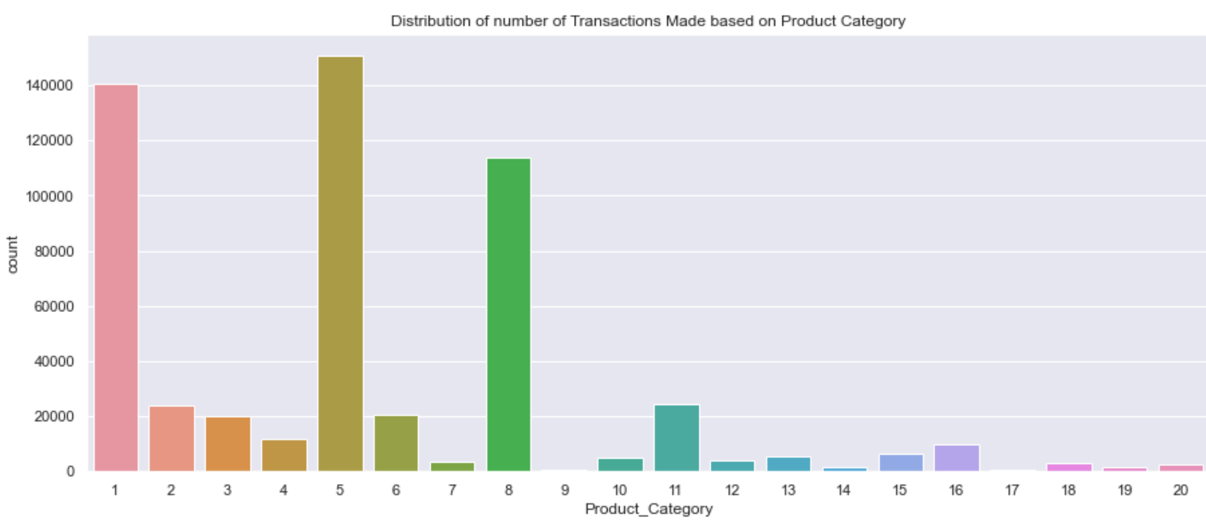
```
plt.figure(figsize = (12, 6))
plt.title('Distribution of number of Transactions Made based on Age')
plt.yticks(np.arange(0, 250001, 25000))
plt.grid('y')
sns.countplot(data = df, x = 'Age',
```

```
                order = ['0-17', '18-25', '26-35', '36-45', '46-50', '51-55', '55+'])
plt.show()
```



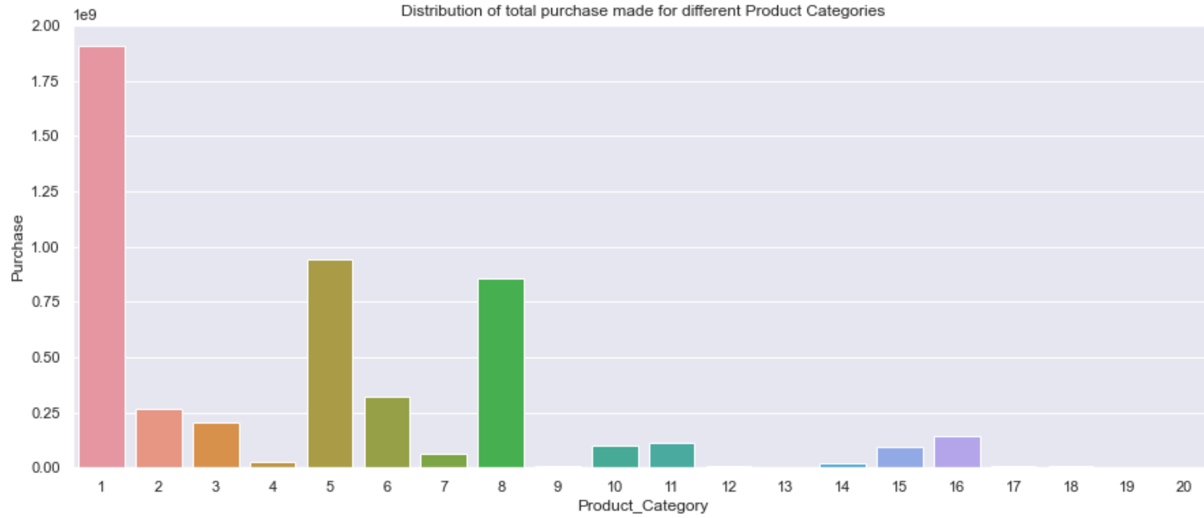
In [321…

```
plt.figure(figsize = (15, 6))
plt.title('Distribution of number of Transactions Made based on Product Category')
sns.countplot(data = df, x = 'Product_Category')
plt.show()
```



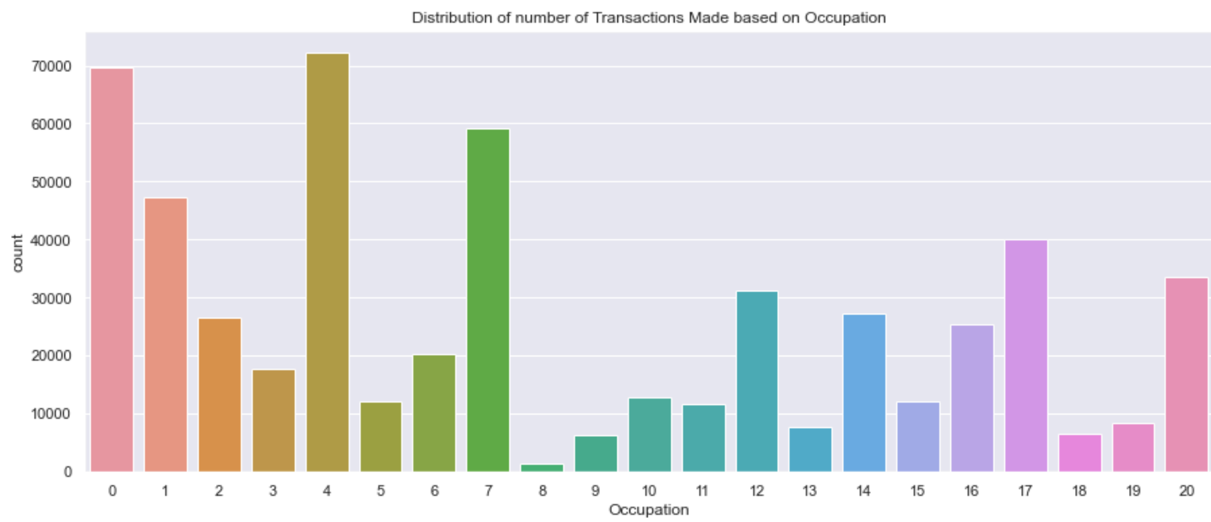
In [322…

```
df_product_category = df.groupby(by = 'Product_Category')['Purchase'].sum().to_fram
plt.figure(figsize = (15, 6))
plt.title('Distribution of total purchase made for different Product Categories')
sns.barplot(data = df_product_category, x = 'Product_Category', y = 'Purchase')
plt.show()
```



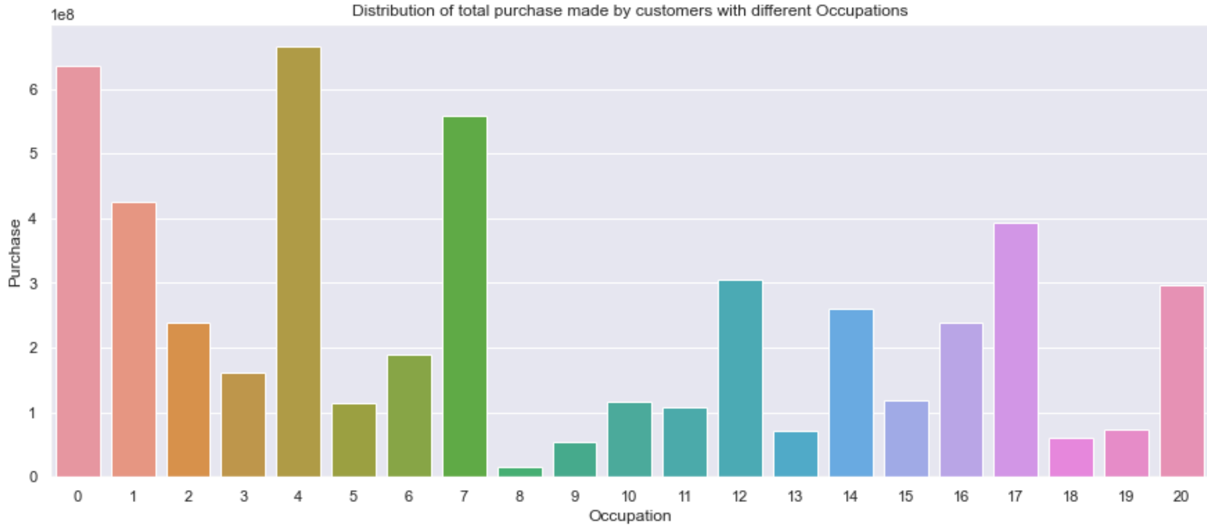
In [323…

```
plt.figure(figsize = (15, 6))
plt.title('Distribution of number of Transactions Made based on Occupation')
sns.countplot(data = df, x = 'Occupation')
plt.show()
```



In [324…

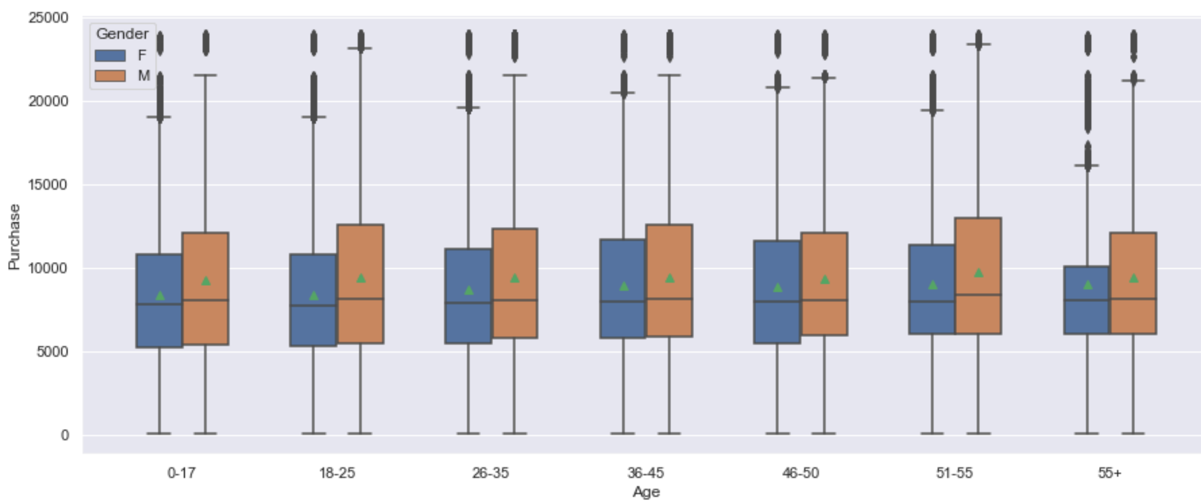
```
df_occupation = df.groupby(by = 'Occupation')['Purchase'].sum().to_frame().reset_in
plt.figure(figsize = (15, 6))
plt.title('Distribution of total purchase made by customers with different Occupati
sns.barplot(data = df_occupation, x = 'Occupation', y = 'Purchase')
plt.show()
```



In [325…

```
plt.figure(figsize = (15, 6))
sns.boxplot(data = df, x = 'Age', y = 'Purchase', hue = 'Gender', showmeans = True,
plt.plot()
```

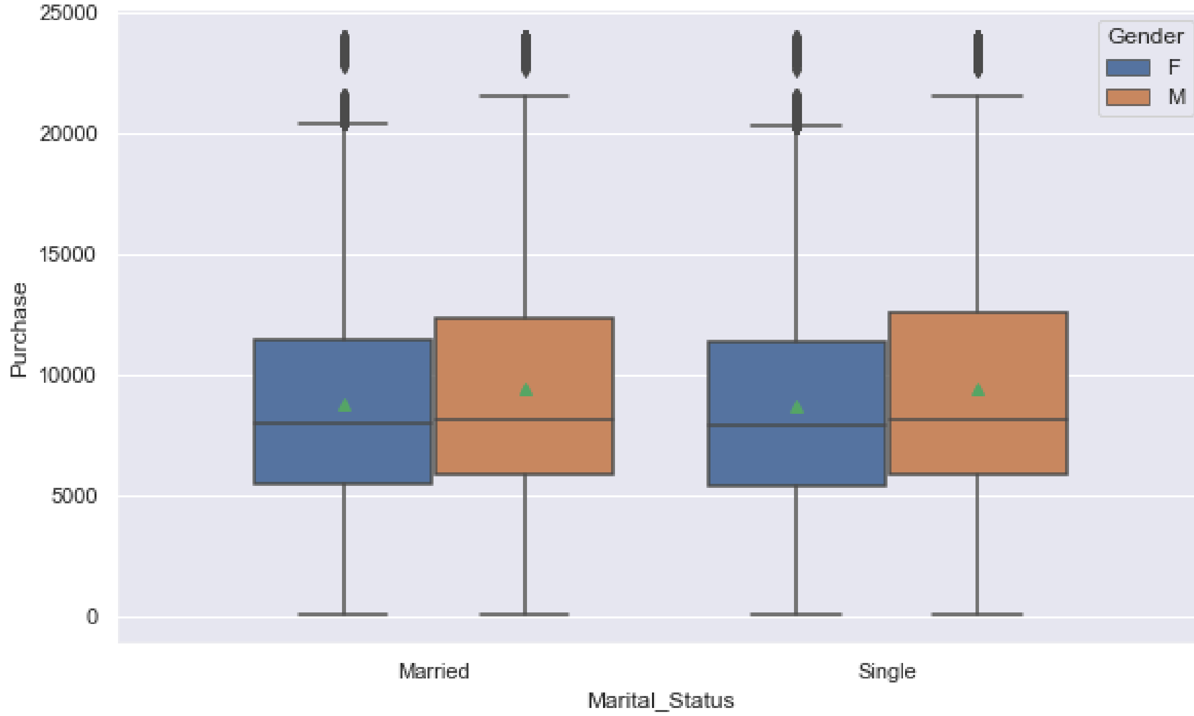
Out[325… []



In [326…

```
plt.figure(figsize = (10, 6))
sns.boxplot(data = df, x = 'Marital_Status', y = 'Purchase', hue = 'Gender', showme
plt.plot()
```

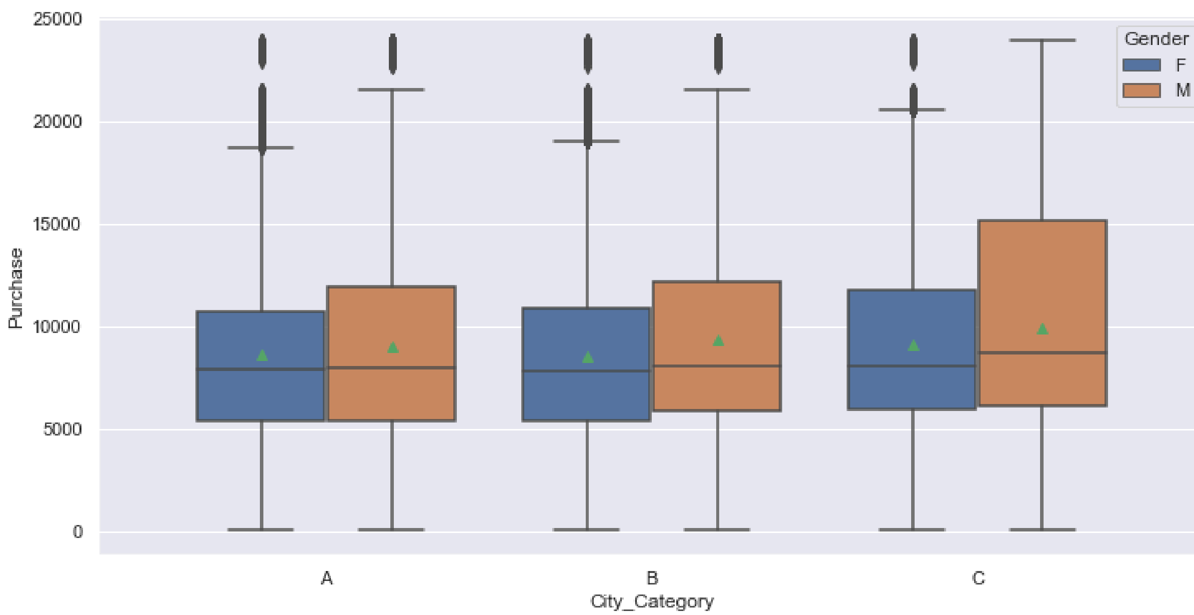
Out[326… []



In [327…

```
plt.figure(figsize = (12, 6))
sns.boxplot(data = df, x = 'City_Category', y = 'Purchase', hue = 'Gender', showmea
plt.plot()
```

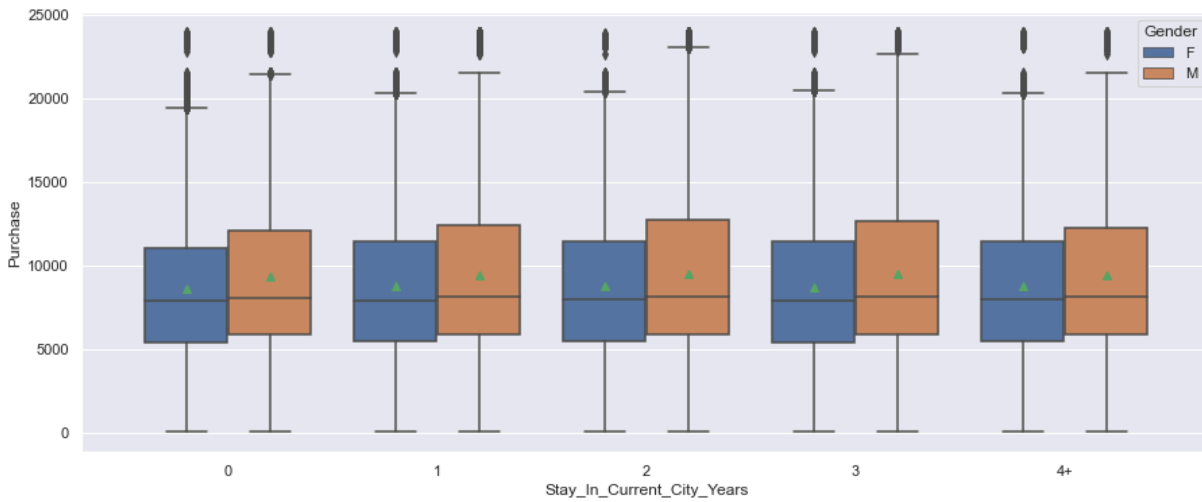
Out[327… []



In [328…

```
plt.figure(figsize = (15, 6))
sns.boxplot(data = df, x = 'Stay_In_Current_City_Years', y = 'Purchase', hue = 'Gen
plt.plot()
```

Out[328… []



Determining the mean purchase made by each user

For Males

How the deviations vary for different sample sizes ?

In [329…

```
df_male_customer
```

Out[329…

	Gender	User_ID	Total_Purchase
1666	M	1000002	810472
1667	M	1000003	341635
1668	M	1000004	206468
1669	M	1000005	821001
1670	M	1000007	234668
...	...	...	...
5886	M	1006030	737361
5887	M	1006032	517261
5888	M	1006033	501843
5889	M	1006034	197086
5890	M	1006040	1653299

4225 rows × 3 columns

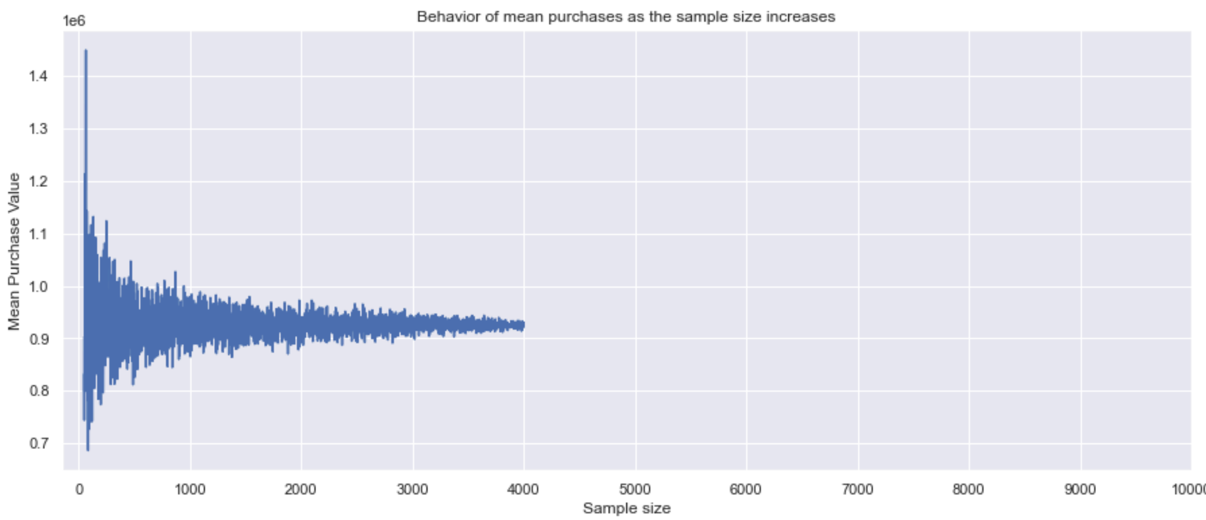
In [330…

```
# The code snippet performs a loop to calculate the mean purchase for different
    # sample sizes of male customers

mean_purchases = []
for sample_size in range(50, 4000):
    sample_mean = df_male_customer['Total_Purchase'].sample(sample_size).mean()
```

```
    mean_purchases.append(sample_mean)

# It iterates over a range of sample sizes from 50 to 4000, and for each iteration,
    # it takes a random sample of the specified size from the 'Total_Purchase' colu
    # of the 'df_male_customer' DataFrame and calculates the mean of the sampled va
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [331…

```
# Creating a plot using matplotlib to visualize the trend of the mean purchases
    # as the sample size increases

plt.figure(figsize = (15, 6))
plt.title('Behavior of mean purchases as the sample size increases')
plt.plot(np.arange(50, 4000), mean_purchases)
plt.xticks(np.arange(0, 10001, 1000))
plt.xlabel('Sample size')
plt.ylabel('Mean Purchase Value')
plt.show()
```



- **It can be inferred from the above plot that as the sample size is small the deviations are fairly high.**
- **As the sample size increases, the deviation becomes smaller and smaller.**
- **The deviations will be small if the sample size taken is greater than 2000.**

Finding the confidence interval of each male's total spending on the Black Friday

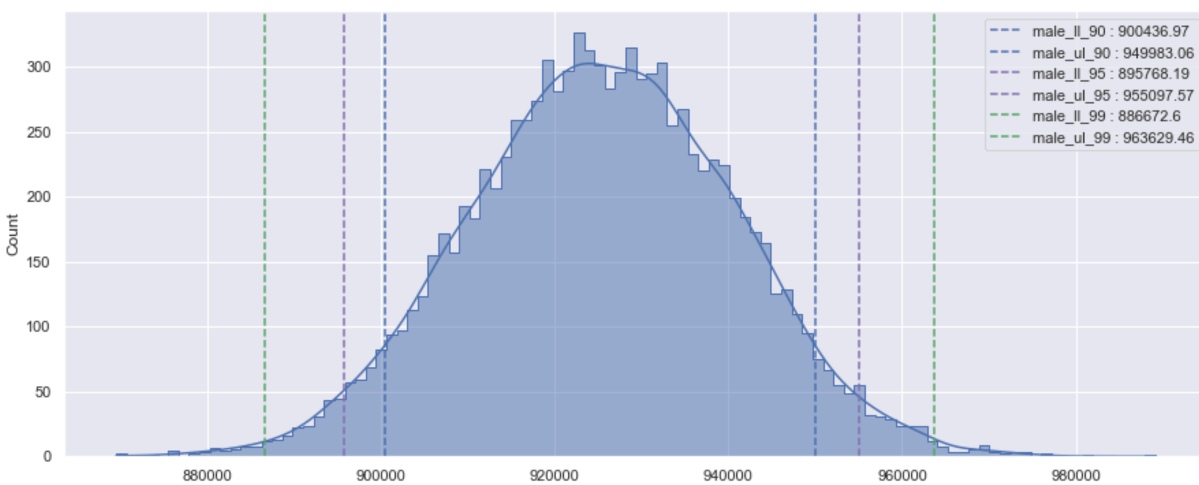
In [332…

```
means_male = []
size = df_male_customer['Total_Purchase'].shape[0]
for bootstrapped_sample in range(10000):
    sample_mean = df_male_customer['Total_Purchase'].sample(size, replace = True).m
    means_male.append(sample_mean)
```

In [333…

```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% le

plt.figure(figsize = (15, 6))     # setting the figure size of the plot

sns.histplot(means_male, kde = True, bins = 100, fill = True, element = 'step')
```

```
# Above line plots a histogram of the data contained in the `means_male` variable.
    # The `kde=True` argument adds a kernel density estimation line to the plot.
    # The `bins=100` argument sets the number of bins for the histogram


# Above line calculates the z-score corresponding to the 90% confidence level using
    # inverse of the cumulative distribution function (CDF) of a standard normal di

male_ll_90 = np.percentile(means_male, 5)
    # calculating the lower limit of the 90% confidence interval
male_ul_90 = np.percentile(means_male, 95)
    # calculating the upper limit of the 90% confidence interval
plt.axvline(male_ll_90, label = f'male_ll_90 : {round(male_ll_90, 2)}', linestyle =
    # adding a vertical line at the lower limit of the 90% confidence interval
plt.axvline(male_ul_90, label = f'male_ul_90 : {round(male_ul_90, 2)}', linestyle =
    # adding a vertical line at the upper limit of the 90% confidence interval

# Similar steps are repeated for calculating and plotting the 95% and 99% confidenc
    # with different line colors (`color='m'` for 95% and `color='g'` for 99%)

male_ll_95 = np.percentile(means_male, 2.5)
male_ul_95 = np.percentile(means_male, 97.5)
plt.axvline(male_ll_95, label = f'male_ll_95 : {round(male_ll_95, 2)}', linestyle =
plt.axvline(male_ul_95, label = f'male_ul_95 : {round(male_ul_95, 2)}', linestyle =


male_ll_99 = np.percentile(means_male, 0.5)
male_ul_99 = np.percentile(means_male, 99.5)
plt.axvline(male_ll_99, label = f'male_ll_99 : {round(male_ll_99, 2)}', linestyle =
plt.axvline(male_ul_99, label = f'male_ul_99 : {round(male_ul_99, 2)}', linestyle =

plt.legend()     # displaying a legend for the plotted lines.
plt.show()       # displaying the plot.
```



- **Through the bootstrapping method, we have been able to estimate the confidence interval for the total purchase made by each male customer on Black Friday at Walmart, despite having data for only 4225 male individuals. This provides us with**

a reasonable approximation of the range within which the total purchase of each male customer falls, with a certain level of confidence.

In [334…
```
print(f"The population mean of total spending of each male will be approximately = 
```

```
The population mean of total spending of each male will be approximately = 925457.47
```

For Females

How the deviations vary for different sample sizes ?

In [335…
```
df_female_customer
```

Out[335…

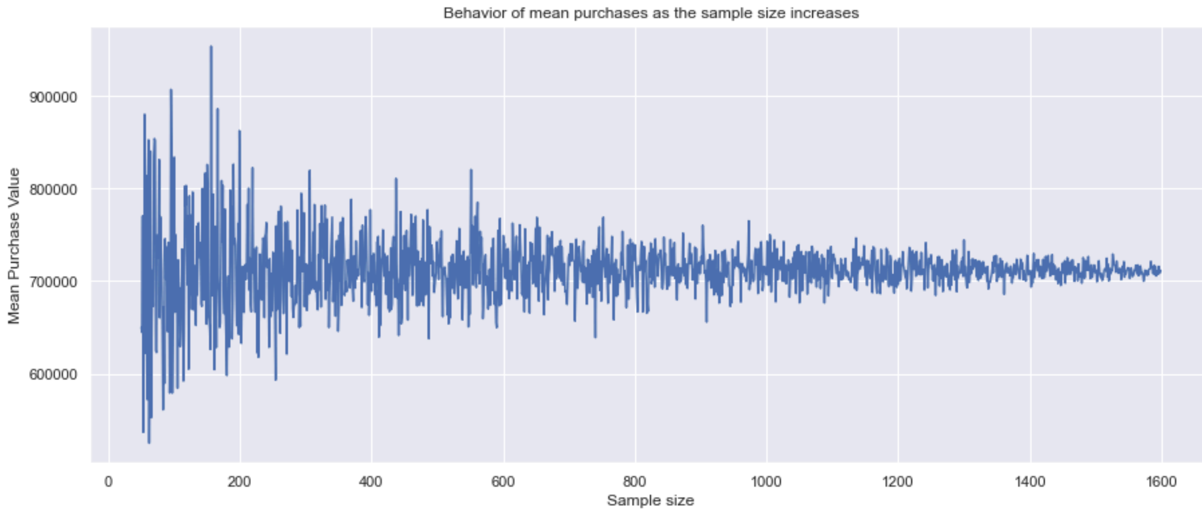
	Gender	User_ID	Total_Purchase
0	F	1000001	334093
1	F	1000006	379930
2	F	1000010	2169510
3	F	1000011	557023
4	F	1000016	150490
...	...	...	...
1661	F	1006035	956645
1662	F	1006036	4116058
1663	F	1006037	1119538
1664	F	1006038	90034
1665	F	1006039	590319

1666 rows × 3 columns

In [336…
```
# The code snippet performs a loop to calculate the mean purchase for different
    # sample sizes of female customers

mean_purchases = []
for sample_size in range(50, 1600):
    sample_mean = df_female_customer['Total_Purchase'].sample(sample_size).mean()
    mean_purchases.append(sample_mean)

# It iterates over a range of sample sizes from 50 to 1600, and for each iteration,
    # it takes a random sample of the specified size from the 'Total_Purchase' colu
    # of the 'df_female_customer' DataFrame and calculates the mean of the sampled
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [337…
```
# Creating a plot using matplotlib to visualize the trend of the mean purchases
    # as the sample size increases
```

```
plt.figure(figsize = (15, 6))
plt.title('Behavior of mean purchases as the sample size increases')
plt.plot(np.arange(50, 1600), mean_purchases)
plt.xlabel('Sample size')
plt.ylabel('Mean Purchase Value')
plt.show()
```



- **It can be inferred from the above plot that as the sample size is small the deviations are fairly high.**
- **As the sample size increases, the deviation becomes smaller and smaller.**
- **The deviations will be small if the sample size taken is greater than 1000.**

Finding the confidence interval of each female's total spending on the Black Friday

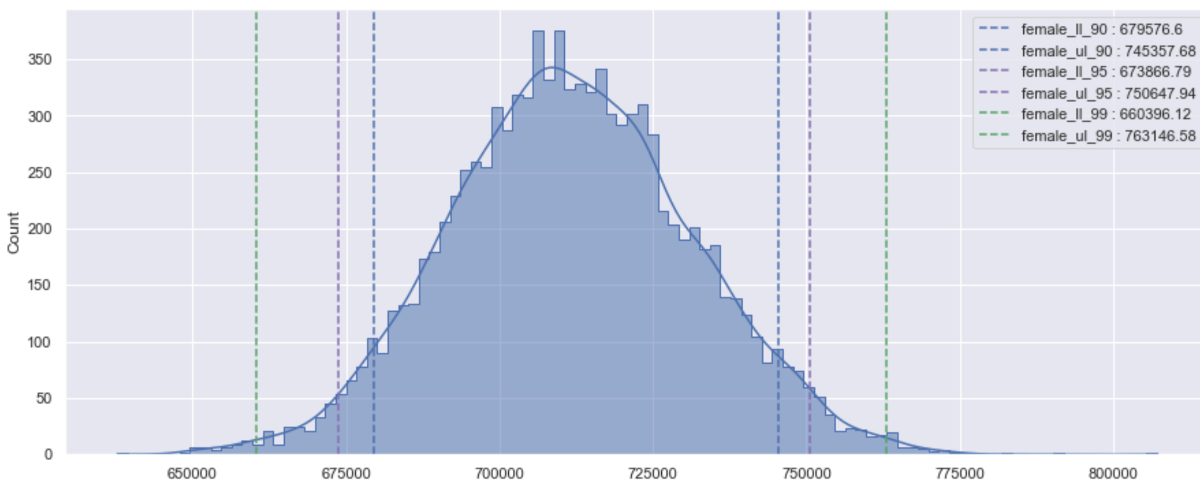
In [338…

```
means_female = []
size = df_female_customer['Total_Purchase'].shape[0]
for bootstrapped_sample in range(10000):
    sample_mean = df_female_customer['Total_Purchase'].sample(size, replace = True)
    means_female.append(sample_mean)
```

In [339…

```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% le

plt.figure(figsize = (15, 6))     # setting the figure size of the plot

sns.histplot(means_female, kde = True, bins = 100, fill = True, element = 'step')

# Above line plots a histogram of the data contained in the `means_female` variable
    # The `kde=True` argument adds a kernel density estimation line to the plot.
    # The `bins=100` argument sets the number of bins for the histogram



# Above line calculates the z-score corresponding to the 90% confidence level using
    # inverse of the cumulative distribution function (CDF) of a standard normal di

female_ll_90 = np.percentile(means_female, 5)
    # calculating the lower limit of the 90% confidence interval
```

```
female_ul_90 = np.percentile(means_female, 95)
    # calculating the upper limit of the 90% confidence interval
plt.axvline(female_ll_90, label = f'female_ll_90 : {round(female_ll_90, 2)}', lines
    # adding a vertical line at the lower limit of the 90% confidence interval
plt.axvline(female_ul_90, label = f'female_ul_90 : {round(female_ul_90, 2)}', lines
    # adding a vertical line at the upper limit of the 90% confidence interval

# Similar steps are repeated for calculating and plotting the 95% and 99% confidenc
    # with different line colors (`color='m'` for 95% and `color='g'` for 99%)

female_ll_95 = np.percentile(means_female, 2.5)
female_ul_95 = np.percentile(means_female, 97.5)
plt.axvline(female_ll_95, label = f'female_ll_95 : {round(female_ll_95, 2)}', lines
plt.axvline(female_ul_95, label = f'female_ul_95 : {round(female_ul_95, 2)}', lines


female_ll_99 = np.percentile(means_female, 0.5)
female_ul_99 = np.percentile(means_female, 99.5)
plt.axvline(female_ll_99, label = f'female_ll_99 : {round(female_ll_99, 2)}', lines
plt.axvline(female_ul_99, label = f'female_ul_99 : {round(female_ul_99, 2)}', lines

plt.legend()     # displaying a legend for the plotted lines.
plt.show()       # displaying the plot.
```



- **Through the bootstrapping method, we have been able to estimate the confidence interval for the total purchase made by each female customer on Black Friday at Walmart, despite having data for only 1666 female individuals. This provides us with a reasonable approximation of the range within which the total purchase of each female customer falls, with a certain level of confidence.**

In [340…

```
print(f"The population mean of total spending of each female will be approximately
```

```
The population mean of total spending of each female will be approximately = 711670.
54
```

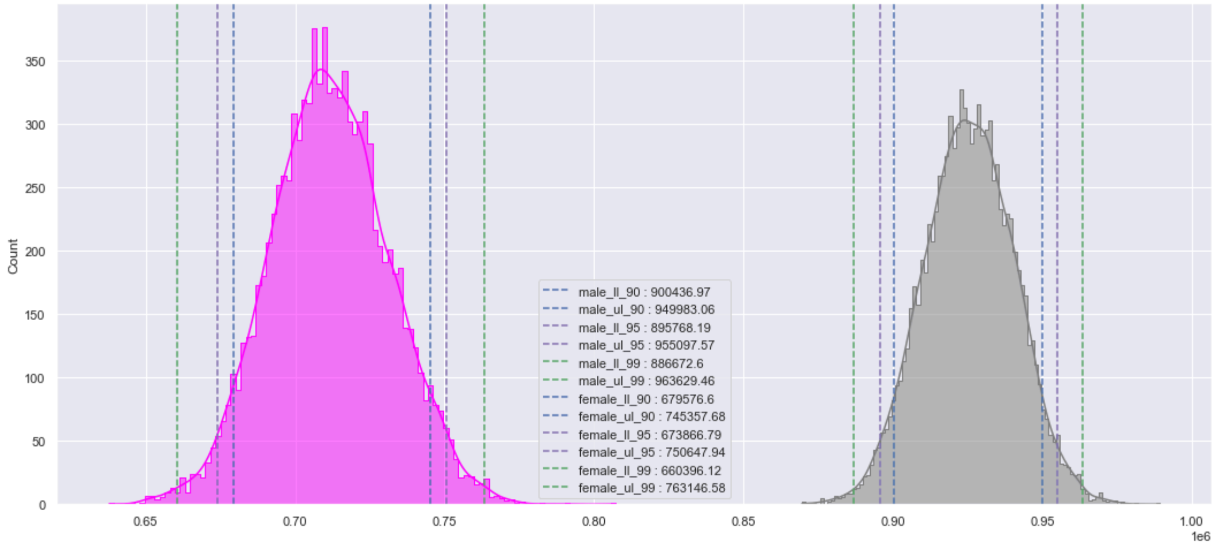
Comparison of distributions of male's total purchase amount and female's total purchase amount

In [341…

```
# The code generates a histogram plot to visualize the distributions of means_male
    # along with vertical lines indicating confidence interval limits at different

plt.figure(figsize = (18, 8))

# The first histogram represents the distribution of means_male with gray color hav
    # KDE (Kernel Density Estimation) curves enabled for smooth representation.
sns.histplot(means_male,
             kde = True,
             bins = 100,
             fill = True,
             element = 'step',
             color = 'gray',
             legend = True)

# Multiple vertical lines are plotted to represent the lower and upper limits
    # for confidence intervals at different confidence levels
plt.axvline(male_ll_90, label = f'male_ll_90 : {round(male_ll_90, 2)}', linestyle =
plt.axvline(male_ul_90, label = f'male_ul_90 : {round(male_ul_90, 2)}', linestyle =
plt.axvline(male_ll_95, label = f'male_ll_95 : {round(male_ll_95, 2)}', linestyle =
plt.axvline(male_ul_95, label = f'male_ul_95 : {round(male_ul_95, 2)}', linestyle =
plt.axvline(male_ll_99, label = f'male_ll_99 : {round(male_ll_99, 2)}', linestyle =
plt.axvline(male_ul_99, label = f'male_ul_99 : {round(male_ul_99, 2)}', linestyle =

# The second histogram represents the distribution of means_female with magenta col
    # KDE (Kernel Density Estimation) curves enabled for smooth representation.
sns.histplot(means_female,
             kde = True,
             bins = 100,
             fill = True,
             element = 'step',
             color = 'magenta',
             legend = True)

# Multiple vertical lines are plotted to represent the lower and upper limits
    # for confidence intervals at different confidence levels
plt.axvline(female_ll_90, label = f'female_ll_90 : {round(female_ll_90, 2)}', lines
plt.axvline(female_ul_90, label = f'female_ul_90 : {round(female_ul_90, 2)}', lines
plt.axvline(female_ll_95, label = f'female_ll_95 : {round(female_ll_95, 2)}', lines
plt.axvline(female_ul_95, label = f'female_ul_95 : {round(female_ul_95, 2)}', lines
plt.axvline(female_ll_99, label = f'female_ll_99 : {round(female_ll_99, 2)}', lines
plt.axvline(female_ul_99, label = f'female_ul_99 : {round(female_ul_99, 2)}', lines

plt.legend()
plt.plot()
```

Out[341…

```
[]
```



It can be clearly seen from the above chart that the distribution of males' total purchase amount lies well towards the right of females' total purchase amount. We can conclude that, *on average, males tend to spend more on purchases compared to females*. This observation suggests a potential difference in spending behavior between genders.

There could be several reasons why males are spending more than females:

- **Product preferences**: Males may have a higher tendency to purchase products that are generally more expensive or fall into higher price categories. This could include items such as electronics, gadgets, or luxury goods.
- **Income disparity**: There may be an income disparity between males and females, with males having higher earning potential or occupying higher-paying job roles. This can lead to a difference in purchasing power and ability to spend more on products.
- **Consumption patterns**: Males might exhibit different consumption patterns, such as being more inclined towards hobbies or interests that require higher spending, such as sports equipment, gaming, or collectibles.
- **Marketing and advertising targeting**: Advertisers and marketers may target males with products or services that are positioned at higher price points. This targeted marketing approach can influence purchasing decisions and contribute to males spending more.

It's important to note that these reasons are general observations and may not apply universally. Individual preferences, personal financial situations, and various other factors can also influence spending patterns.

Determining the mean purchase made by each user belonging to different Marital Status

In [342…
```
df_single = df.loc[df['Marital_Status'] == 'Single']
df_married = df.loc[df['Marital_Status'] == 'Married']
```

In [343…
```
df_single = df_single.groupby('User_ID')['Purchase'].sum().to_frame().reset_index()
df_married = df_married.groupby('User_ID')['Purchase'].sum().to_frame().reset_index
```

For Non Married

In [344…
```
df_single
```

Out[344…

	User_ID	Total_Purchase
0	1000001	334093
1	1000002	810472
2	1000003	341635
3	1000006	379930
4	1000009	594099
...	...	...
3412	1006034	197086
3413	1006035	956645
3414	1006037	1119538
3415	1006038	90034
3416	1006040	1653299

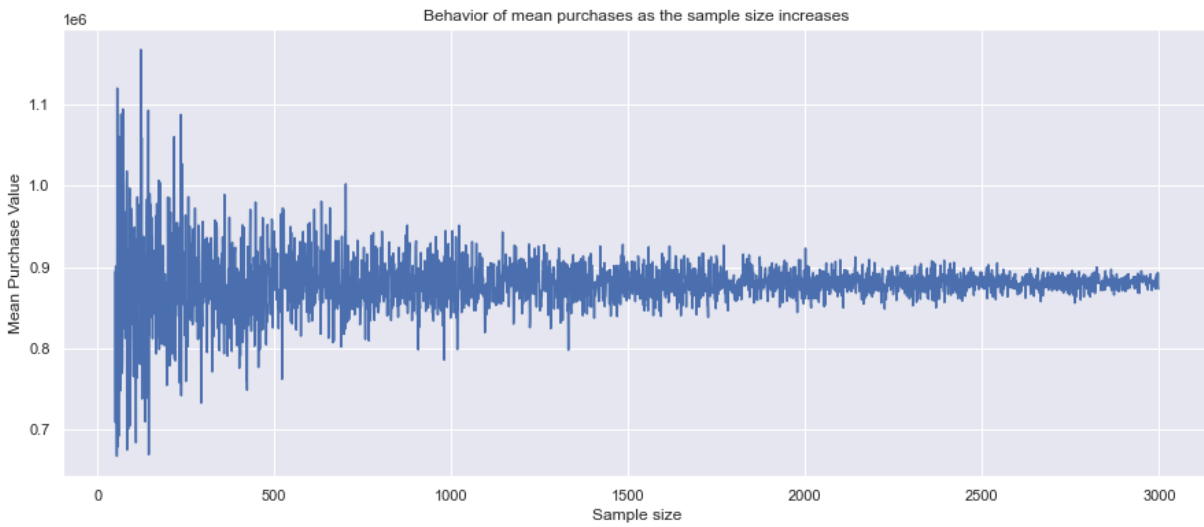
3417 rows × 2 columns

How the deviations vary for different sample sizes ?

In [345…
```
# The code snippet performs a loop to calculate the mean purchase for different
    # sample sizes of customers with matrital status as single

mean_purchases = []
for sample_size in range(50, 3000):
    sample_mean = df_single['Total_Purchase'].sample(sample_size).mean()
    mean_purchases.append(sample_mean)

# It iterates over a range of sample sizes from 50 to 3000, and for each iteration,
    # it takes a random sample of the specified size from the 'Total_Purchase' colu
    # of the 'df_single' DataFrame and calculates the mean of the sampled values.
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [346…
```
# Creating a plot using matplotlib to visualize the trend of the mean purchases
    # as the sample size increases

plt.figure(figsize = (15, 6))
```

```
plt.title('Behavior of mean purchases as the sample size increases')
plt.plot(np.arange(50, 3000), mean_purchases)
plt.xlabel('Sample size')
plt.ylabel('Mean Purchase Value')
plt.show()
```



- **It can be inferred from the above plot that as the sample size is small the deviations are fairly high.

As the sample size increases, the deviation becomes smaller and smaller. The deviations will be small if the sample size taken is greater than 2000.**

Finding the confidence interval of each single's total spending on the Black Friday

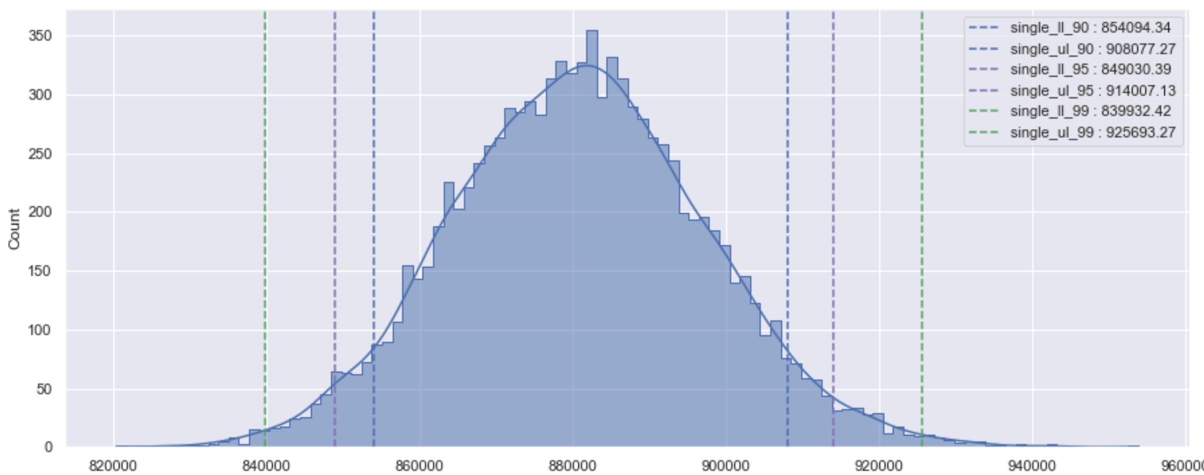
In [347…

```
single_means = []
size = df_single['Total_Purchase'].shape[0]
for bootstrapped_sample in range(10000):
    sample_mean = df_single['Total_Purchase'].sample(size, replace = True).mean()
    single_means.append(sample_mean)
```

In [348…

```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% le

plt.figure(figsize = (15, 6))     # setting the figure size of the plot

sns.histplot(single_means, kde = True, bins = 100, fill = True, element = 'step')

# Above line plots a histogram of the data contained in the `single_means` variable
    # The `kde=True` argument adds a kernel density estimation line to the plot.
    # The `bins=100` argument sets the number of bins for the histogram


# Above line calculates the z-score corresponding to the 90% confidence level using
    # inverse of the cumulative distribution function (CDF) of a standard normal di

single_ll_90 = np.percentile(single_means, 5)
    # calculating the lower limit of the 90% confidence interval
```

```
single_ul_90 = np.percentile(single_means, 95)
    # calculating the upper limit of the 90% confidence interval
plt.axvline(single_ll_90, label = f'single_ll_90 : {round(single_ll_90, 2)}', lines
    # adding a vertical line at the lower limit of the 90% confidence interval
plt.axvline(single_ul_90, label = f'single_ul_90 : {round(single_ul_90, 2)}', lines
    # adding a vertical line at the upper limit of the 90% confidence interval

# Similar steps are repeated for calculating and plotting the 95% and 99% confidenc
    # with different line colors (`color='m'` for 95% and `color='g'` for 99%)

single_ll_95 = np.percentile(single_means, 2.5)
single_ul_95 = np.percentile(single_means, 97.5)
plt.axvline(single_ll_95, label = f'single_ll_95 : {round(single_ll_95, 2)}', lines
plt.axvline(single_ul_95, label = f'single_ul_95 : {round(single_ul_95, 2)}', lines


single_ll_99 = np.percentile(single_means, 0.5)
single_ul_99 = np.percentile(single_means, 99.5)
plt.axvline(single_ll_99, label = f'single_ll_99 : {round(single_ll_99, 2)}', lines
plt.axvline(single_ul_99, label = f'single_ul_99 : {round(single_ul_99, 2)}', lines

plt.legend()     # displaying a legend for the plotted lines.
plt.show()       # displaying the plot.
```



- **Through the bootstrapping method, we have been able to estimate the confidence interval for the total purchase made by each single customer on Black Friday at Walmart, despite having data for only 3417 individuals having single as marital status. This provides us with a reasonable approximation of the range within which the total purchase of each single customer falls, with a certain level of confidence.**

In [349…

```
print(f"The population mean of total spending of each single will be approximately
```

```
The population mean of total spending of each single will be approximately = 880892.
18
```

For Married

In [350…

```
df_married
```

Out[350…

	User_ID	Total_Purchase
0	1000004	206468
1	1000005	821001
2	1000007	234668
3	1000008	796593
4	1000010	2169510
...	...	...
2469	1006029	157436
2470	1006030	737361
2471	1006033	501843
2472	1006036	4116058
2473	1006039	590319

2474 rows × 2 columns

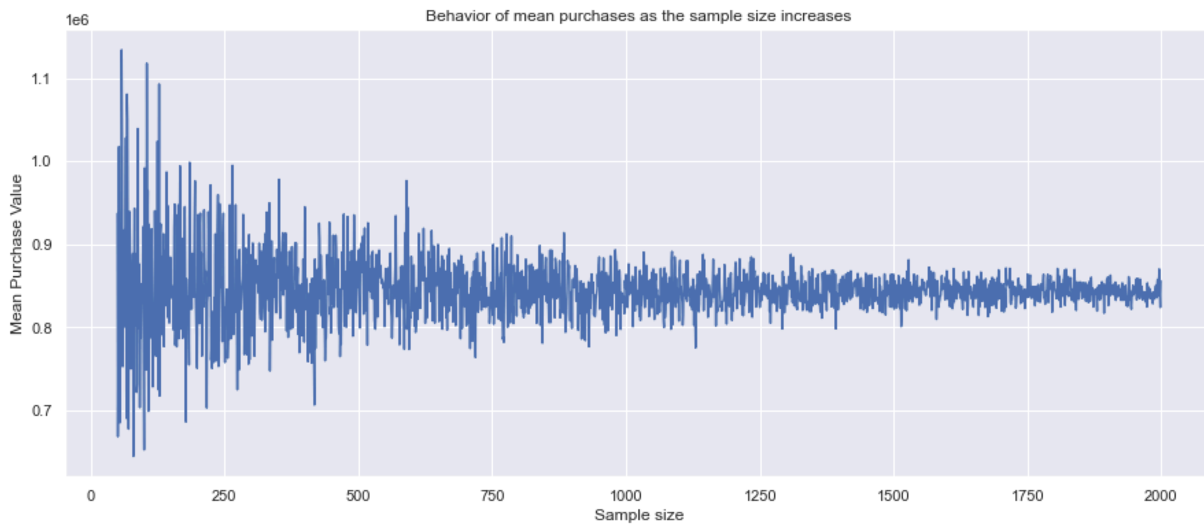
How the deviations vary for different sample sizes ?

In [351…

```
# The code snippet performs a loop to calculate the mean purchase for different
    # sample sizes of customers with matrital status as married

mean_purchases = []
for sample_size in range(50, 2000):
    sample_mean = df_married['Total_Purchase'].sample(sample_size).mean()
    mean_purchases.append(sample_mean)

# It iterates over a range of sample sizes from 50 to 2000, and for each iteration,
    # it takes a random sample of the specified size from the 'Total_Purchase' colu
    # of the 'df_married' DataFrame and calculates the mean of the sampled values.
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [352…

```
# Creating a plot using matplotlib to visualize the trend of the mean purchases
    # as the sample size increases

plt.figure(figsize = (15, 6))
plt.title('Behavior of mean purchases as the sample size increases')
plt.plot(np.arange(50, 2000), mean_purchases)
plt.xlabel('Sample size')
plt.ylabel('Mean Purchase Value')
plt.show()
```



- **It can be inferred from the above plot that as the sample size is small the deviations are fairly high.

As the sample size increases, the deviation becomes smaller and smaller. The deviations will be small if the sample size taken is greater than 1500.**

Finding the confidence interval of each married's total spending on the Black Friday

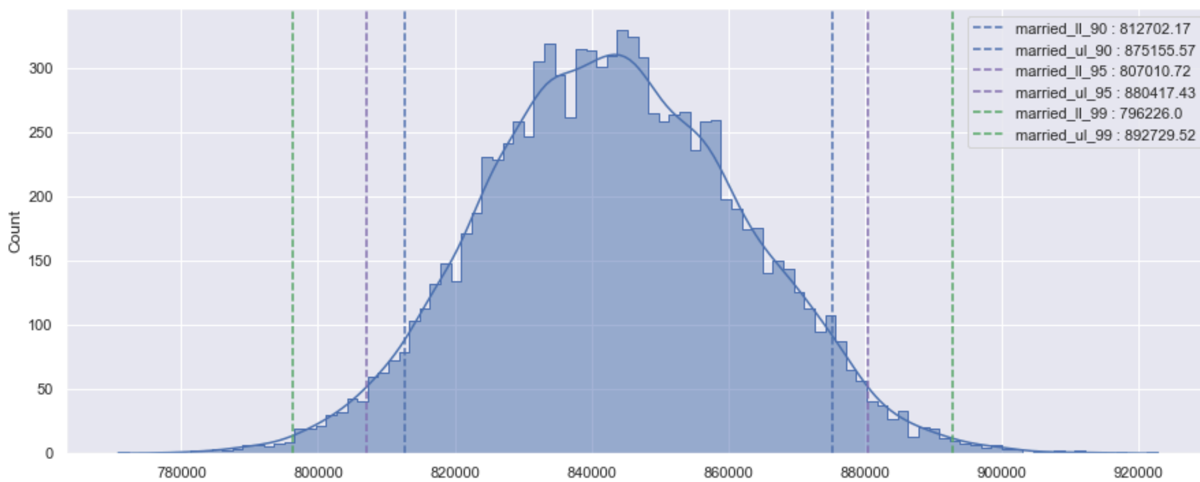
In [353…

```
married_means = []
size = df_married['Total_Purchase'].shape[0]
for bootstrapped_sample in range(10000):
    sample_mean = df_married['Total_Purchase'].sample(size, replace = True).mean()
    married_means.append(sample_mean)
```

In [354…

```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% le

plt.figure(figsize = (15, 6))     # setting the figure size of the plot

sns.histplot(married_means, kde = True, bins = 100, fill = True, element = 'step')

# Above line plots a histogram of the data contained in the `married_means` variabl
    # The `kde=True` argument adds a kernel density estimation line to the plot.
    # The `bins=100` argument sets the number of bins for the histogram


# Above line calculates the z-score corresponding to the 90% confidence level using
    # inverse of the cumulative distribution function (CDF) of a standard normal di

married_ll_90 = np.percentile(married_means, 5)
    # calculating the lower limit of the 90% confidence interval
married_ul_90 = np.percentile(married_means, 95)
    # calculating the upper limit of the 90% confidence interval
plt.axvline(married_ll_90, label = f'married_ll_90 : {round(married_ll_90, 2)}', li
    # adding a vertical line at the lower limit of the 90% confidence interval
plt.axvline(married_ul_90, label = f'married_ul_90 : {round(married_ul_90, 2)}', li
    # adding a vertical line at the upper limit of the 90% confidence interval
```

```
# Similar steps are repeated for calculating and plotting the 95% and 99% confidenc
    # with different line colors (`color='m'` for 95% and `color='g'` for 99%)

married_ll_95 = np.percentile(married_means, 2.5)
married_ul_95 = np.percentile(married_means, 97.5)
plt.axvline(married_ll_95, label = f'married_ll_95 : {round(married_ll_95, 2)}', li
plt.axvline(married_ul_95, label = f'married_ul_95 : {round(married_ul_95, 2)}', li


married_ll_99 = np.percentile(married_means, 0.5)
married_ul_99 = np.percentile(married_means, 99.5)
plt.axvline(married_ll_99, label = f'married_ll_99 : {round(married_ll_99, 2)}', li
plt.axvline(married_ul_99, label = f'married_ul_99 : {round(married_ul_99, 2)}', li

plt.legend()     # displaying a legend for the plotted lines.
plt.show()       # displaying the plot.
```



- **Through the bootstrapping method, we have been able to estimate the confidence interval for the total purchase made by each married customer on Black Friday at Walmart, despite having data for only 2474 individuals having married as marital status. This provides us with a reasonable approximation of the range within which the total purchase of each married customer falls, with a certain level of confidence.**

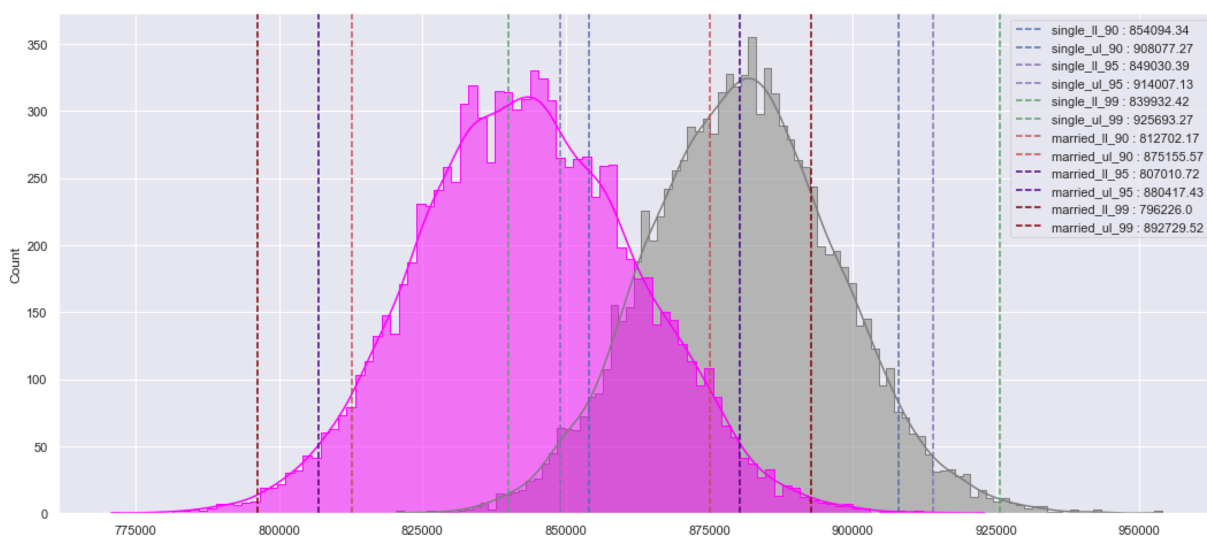
In [355…
```
print(f"The population mean of total spending of each male will be approximately =
```

```
The population mean of total spending of each male will be approximately = 843372.37
```

Comparison of distributions of single's total purchase amount and married's total purchase amount

In [356…
```
# The code generates a histogram plot to visualize the distributions of single_mean
    # along with vertical lines indicating confidence interval limits at different

plt.figure(figsize = (18, 8))

# The first histogram represents the distribution of single_means with gray color h
```

```
    # KDE (Kernel Density Estimation) curves enabled for smooth representation.
sns.histplot(single_means,
             kde = True,
             bins = 100,
             fill = True,
             element = 'step',
             color = 'gray',
             legend = True)

# Multiple vertical lines are plotted to represent the lower and upper limits
    # for confidence intervals at different confidence levels
plt.axvline(single_ll_90, label = f'single_ll_90 : {round(single_ll_90, 2)}', lines
plt.axvline(single_ul_90, label = f'single_ul_90 : {round(single_ul_90, 2)}', lines
plt.axvline(single_ll_95, label = f'single_ll_95 : {round(single_ll_95, 2)}', lines
plt.axvline(single_ul_95, label = f'single_ul_95 : {round(single_ul_95, 2)}', lines
plt.axvline(single_ll_99, label = f'single_ll_99 : {round(single_ll_99, 2)}', lines
plt.axvline(single_ul_99, label = f'single_ul_99 : {round(single_ul_99, 2)}', lines

# The second histogram represents the distribution of married_means with magenta co
    # KDE (Kernel Density Estimation) curves enabled for smooth representation.
sns.histplot(married_means,
             kde = True,
             bins = 100,
             fill = True,
             element = 'step',
             color = 'magenta',
             legend = True)

# Multiple vertical lines are plotted to represent the lower and upper limits
    # for confidence intervals at different confidence levels
plt.axvline(married_ll_90, label = f'married_ll_90 : {round(married_ll_90, 2)}', li
plt.axvline(married_ul_90, label = f'married_ul_90 : {round(married_ul_90, 2)}', li
plt.axvline(married_ll_95, label = f'married_ll_95 : {round(married_ll_95, 2)}', li
plt.axvline(married_ul_95, label = f'married_ul_95 : {round(married_ul_95, 2)}', li
plt.axvline(married_ll_99, label = f'married_ll_99 : {round(married_ll_99, 2)}', li
plt.axvline(married_ul_99, label = f'married_ul_99 : {round(married_ul_99, 2)}', li

plt.legend()
plt.show()
```



It can be inferred from the above chart that the distributions of singles' total spending and married individuals' total spending overlap. It suggests that there is no significant difference in spending habits between these two groups. Here are some possible inferences that can be drawn from this:

- **Relationship status does not strongly influence spending**: Being single or married does not appear to have a substantial impact on individuals' spending patterns. Other factors such as income, personal preferences, and financial priorities may play a more significant role in determining spending habits.

- **Similar consumption patterns**: Singles and married individuals may have similar lifestyles and consumption patterns, leading to comparable spending behaviors. They may allocate their income in comparable ways, making similar purchasing decisions and spending on similar categories of products or services.

- **Financial considerations**: Both singles and married individuals may have similar financial responsibilities and constraints, leading to similar spending levels. They may have similar obligations such as housing costs, bills, and other financial commitments, which influence their overall spending capacity.

- **Individual differences outweigh relationship status**: Other individual characteristics, such as personal values, interests, and financial habits, may have a more significant impact on spending behavior than relationship status. These factors can vary widely within each group, resulting in overlapping spending distributions.

Determining the mean purchase made by each user based on their age groups :

In [357…

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

Out[357…

```
['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25']
Categories (7, object): ['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-2
5']
```

In [358…

```
df_age_0_to_17 = df.loc[df['Age'] == '0-17']
df_age_18_to_25 = df.loc[df['Age'] == '18-25']
df_age_26_to_35 = df.loc[df['Age'] == '26-35']
df_age_36_to_45 = df.loc[df['Age'] == '36-45']
df_age_46_to_50 = df.loc[df['Age'] == '46-50']
df_age_51_to_55 = df.loc[df['Age'] == '51-55']
df_age_above_55 = df.loc[df['Age'] == '55+']
```

In [359…

```
df_age_0_to_17 = df_age_0_to_17.groupby(by = 'User_ID')['Purchase'].sum().to_frame(
df_age_18_to_25 = df_age_18_to_25.groupby(by = 'User_ID')['Purchase'].sum().to_fram
df_age_26_to_35 = df_age_26_to_35.groupby(by = 'User_ID')['Purchase'].sum().to_fram
df_age_36_to_45 = df_age_36_to_45.groupby(by = 'User_ID')['Purchase'].sum().to_fram
df_age_46_to_50 = df_age_46_to_50.groupby(by = 'User_ID')['Purchase'].sum().to_fram
```

```
df_age_51_to_55 = df_age_51_to_55.groupby(by = 'User_ID')['Purchase'].sum().to_fram
df_age_above_55 = df_age_above_55.groupby(by = 'User_ID')['Purchase'].sum().to_fram
```

For Age Group 0 - 17 years

In [360…

```
df_age_0_to_17
```

Out[360…

	User_ID	Total_Purchase
0	1000001	334093
1	1000019	1458069
2	1000051	200772
3	1000075	1035584
4	1000086	294063
...	...	...
213	1005844	476231
214	1005953	629161
215	1005973	270475
216	1005989	466195
217	1006006	514919

218 rows × 2 columns

How the deviations vary for different sample sizes ?

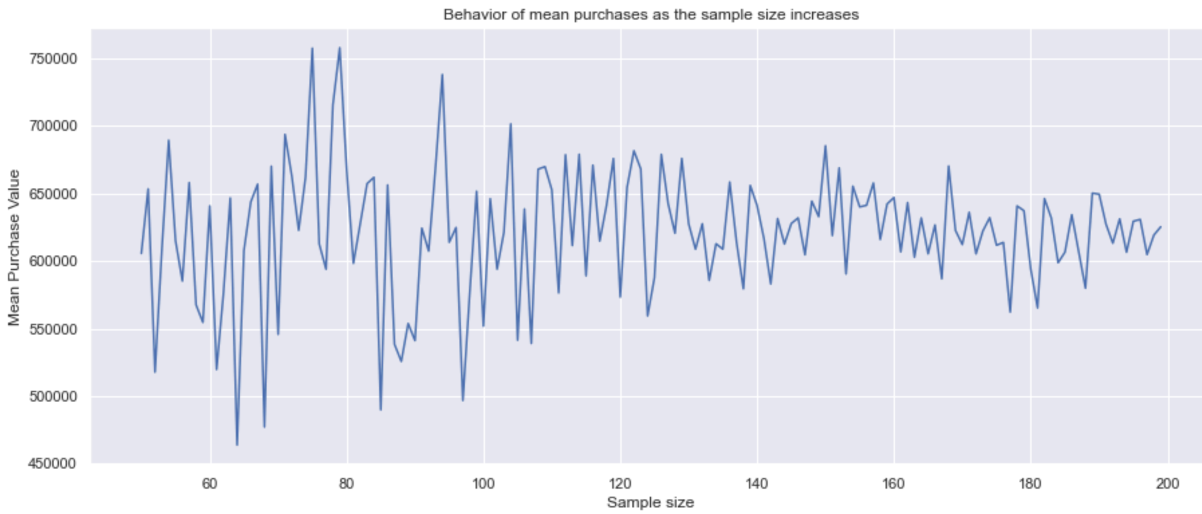
In [361…

```
# The code snippet performs a loop to calculate the mean purchase for different
    # sample sizes of customers with age group 0 - 17 yrs.

mean_purchases = []
for sample_size in range(50, 200):
    sample_mean = df_age_0_to_17['Total_Purchase'].sample(sample_size).mean()
    mean_purchases.append(sample_mean)

# It iterates over a range of sample sizes from 50 to 200, and for each iteration,
    # it takes a random sample of the specified size from the 'Total_Purchase' colu
    # of the 'df_age_0_to_17' DataFrame and calculates the mean of the sampled valu
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [362…

```
# Creating a plot using matplotlib to visualize the trend of the mean purchases
    # as the sample size increases

plt.figure(figsize = (15, 6))
plt.title('Behavior of mean purchases as the sample size increases')
plt.plot(np.arange(50, 200), mean_purchases)
plt.xlabel('Sample size')
```

```
plt.ylabel('Mean Purchase Value')
plt.show()
```



- **It can be inferred from the above plot that as the sample size is small the deviations are fairly high.

As the sample size increases, the deviation becomes smaller and smaller. The deviations will be small if the sample size taken is greater than 150.**

Finding the confidence interval of total spending for each individual in the age group 0 - 17 on the Black Friday

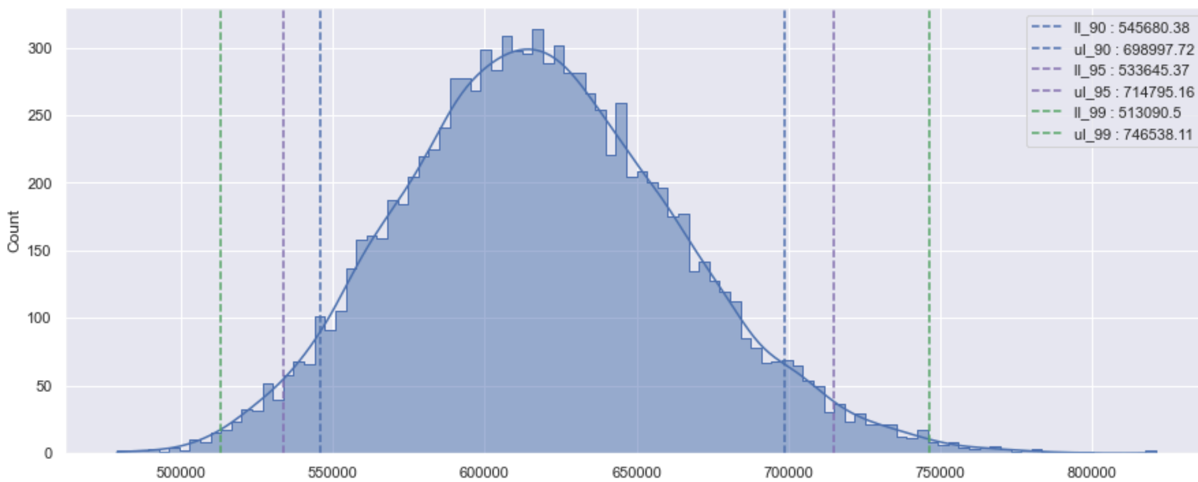
In [363…

```
means = []
size = df_age_0_to_17['Total_Purchase'].shape[0]
for bootstrapped_sample in range(10000):
    sample_mean = df_age_0_to_17['Total_Purchase'].sample(size, replace = True).mea
    means.append(sample_mean)
```

In [364…

```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% le

plt.figure(figsize = (15, 6))     # setting the figure size of the plot

sns.histplot(means, kde = True, bins = 100, fill = True, element = 'step')

# Above line plots a histogram of the data contained in the `means` variable.
    # The `kde=True` argument adds a kernel density estimation line to the plot.
    # The `bins=100` argument sets the number of bins for the histogram


# Above line calculates the z-score corresponding to the 90% confidence level using
    # inverse of the cumulative distribution function (CDF) of a standard normal di

ll_90 = np.percentile(means, 5)
    # calculating the lower limit of the 90% confidence interval
ul_90 = np.percentile(means, 95)
    # calculating the upper limit of the 90% confidence interval
plt.axvline(ll_90, label = f'll_90 : {round(ll_90, 2)}', linestyle = '--')
```

```
    # adding a vertical line at the lower limit of the 90% confidence interval
plt.axvline(ul_90, label = f'ul_90 : {round(ul_90, 2)}', linestyle = '--')
    # adding a vertical line at the upper limit of the 90% confidence interval

# Similar steps are repeated for calculating and plotting the 95% and 99% confidenc
    # with different line colors (`color='m'` for 95% and `color='g'` for 99%)

ll_95 = np.percentile(means, 2.5)
ul_95 = np.percentile(means, 97.5)
plt.axvline(ll_95, label = f'll_95 : {round(ll_95, 2)}', linestyle = '--', color =
plt.axvline(ul_95, label = f'ul_95 : {round(ul_95, 2)}', linestyle = '--', color =


ll_99 = np.percentile(means, 0.5)
ul_99 = np.percentile(means, 99.5)
plt.axvline(ll_99, label = f'll_99 : {round(ll_99, 2)}', linestyle = '--', color =
plt.axvline(ul_99, label = f'ul_99 : {round(ul_99, 2)}', linestyle = '--', color =

plt.legend()     # displaying a legend for the plotted lines.
plt.show()       # displaying the plot.
```



- **Through the bootstrapping method, we have been able to estimate the confidence interval for the total purchase made by each individual in age group 0 - 17 years on Black Friday at Walmart, despite having data for only 218 individuals having age group 0 - 17 years. This provides us with a reasonable approximation of the range within which the total purchase of each individuals having age group 0 - 17 years falls, with a certain level of confidence.**

In [365…

```
print(f"The population mean of total spending of each customer in age group 0 -17 w
```

```
The population mean of total spending of each customer in age group 0 -17 will be ap
proximately = 618567.18
```

For Age Group 18 - 25 years

In [366…

```
df_age_18_to_25
```

Out[366…

	User_ID	Total_Purchase
0	1000018	1979047
1	1000021	127099
2	1000022	1279914
3	1000025	534706
4	1000034	807983
...	...	...
1064	1005998	702901
1065	1006008	266306
1066	1006027	265201
1067	1006028	362972
1068	1006031	286374

1069 rows × 2 columns

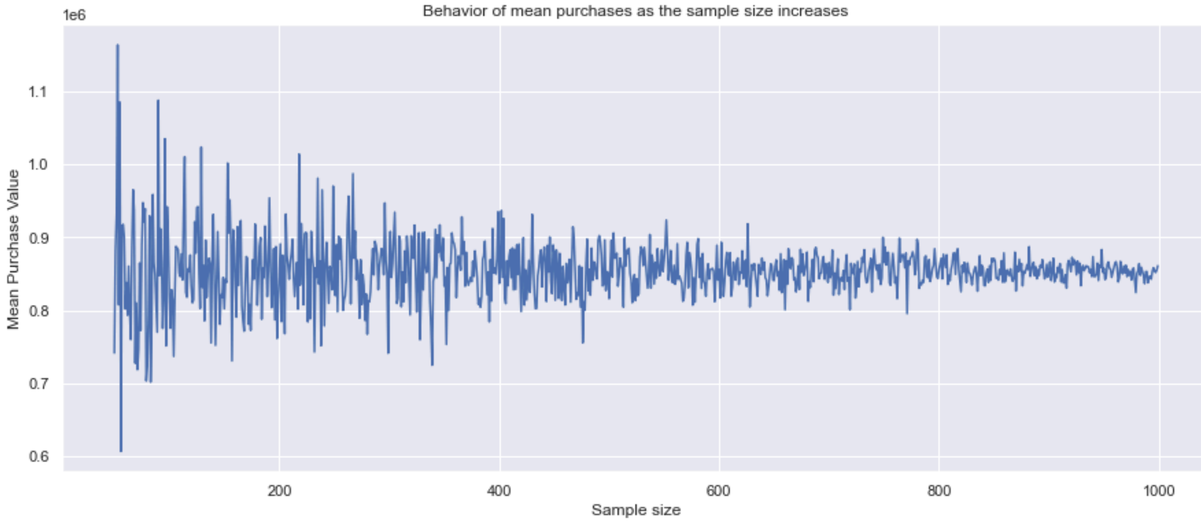
How the deviations vary for different sample sizes ?

In [367…

```
# The code snippet performs a loop to calculate the mean purchase for different
    # sample sizes of customers with age group 18 - 25 yrs.

mean_purchases = []
for sample_size in range(50, 1000):
    sample_mean = df_age_18_to_25['Total_Purchase'].sample(sample_size).mean()
    mean_purchases.append(sample_mean)

# It iterates over a range of sample sizes from 50 to 1000, and for each iteration,
    # it takes a random sample of the specified size from the 'Total_Purchase' colu
    # of the 'df_age_18_to_25' DataFrame and calculates the mean of the sampled val
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [368…

```
# Creating a plot using matplotlib to visualize the trend of the mean purchases as

plt.figure(figsize = (15, 6))
plt.title('Behavior of mean purchases as the sample size increases')
plt.plot(np.arange(50, 1000), mean_purchases)
plt.xlabel('Sample size')
plt.ylabel('Mean Purchase Value')
plt.show()
```



- **It can be inferred from the above plot that as the sample size is small the deviations are fairly high.

As the sample size increases, the deviation becomes smaller and smaller. The deviations will be small if the sample size taken is greater than 600.**

Finding the confidence interval of total spending for each individual in the age group 18 - 25 on the Black Friday

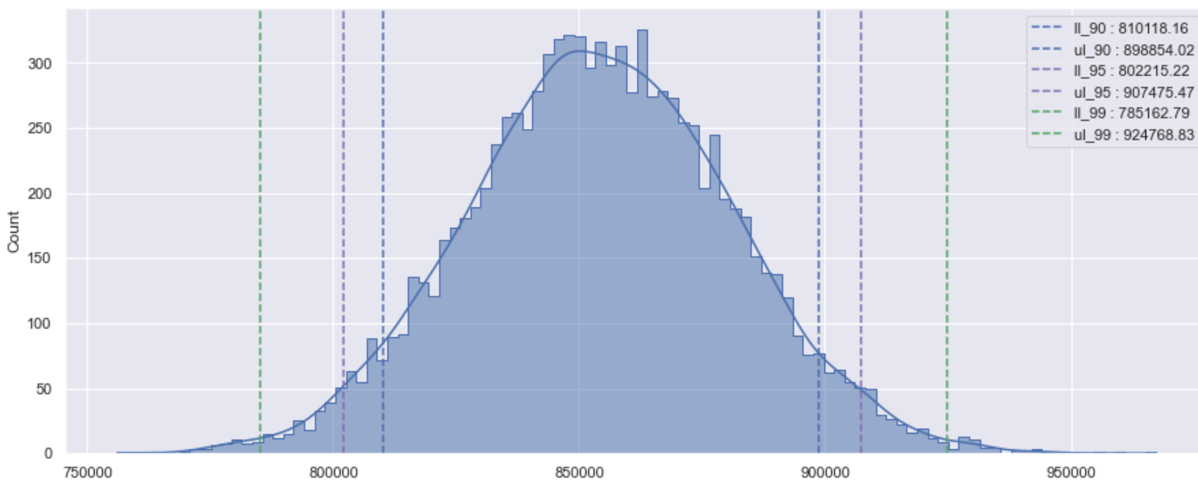
In [369…

```
means = []
size = df_age_18_to_25['Total_Purchase'].shape[0]
for bootstrapped_sample in range(10000):
    sample_mean = df_age_18_to_25['Total_Purchase'].sample(size, replace = True).me
    means.append(sample_mean)
```

In [370…

```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% le

plt.figure(figsize = (15, 6))     # setting the figure size of the plot

sns.histplot(means, kde = True, bins = 100, fill = True, element = 'step')

# Above line plots a histogram of the data contained in the `means` variable.
    # The `kde=True` argument adds a kernel density estimation line to the plot.
    # The `bins=100` argument sets the number of bins for the histogram


# Above line calculates the z-score corresponding to the 90% confidence level using
    # inverse of the cumulative distribution function (CDF) of a standard normal di

ll_90 = np.percentile(means, 5)
    # calculating the lower limit of the 90% confidence interval
ul_90 = np.percentile(means, 95)
    # calculating the upper limit of the 90% confidence interval
plt.axvline(ll_90, label = f'll_90 : {round(ll_90, 2)}', linestyle = '--')
    # adding a vertical line at the lower limit of the 90% confidence interval
plt.axvline(ul_90, label = f'ul_90 : {round(ul_90, 2)}', linestyle = '--')
```

```
    # adding a vertical line at the upper limit of the 90% confidence interval

# Similar steps are repeated for calculating and plotting the 95% and 99% confidenc
    # with different line colors (`color='m'` for 95% and `color='g'` for 99%)

ll_95 = np.percentile(means, 2.5)
ul_95 = np.percentile(means, 97.5)
plt.axvline(ll_95, label = f'll_95 : {round(ll_95, 2)}', linestyle = '--', color =
plt.axvline(ul_95, label = f'ul_95 : {round(ul_95, 2)}', linestyle = '--', color =


ll_99 = np.percentile(means, 0.5)
ul_99 = np.percentile(means, 99.5)
plt.axvline(ll_99, label = f'll_99 : {round(ll_99, 2)}', linestyle = '--', color =
plt.axvline(ul_99, label = f'ul_99 : {round(ul_99, 2)}', linestyle = '--', color =

plt.legend()     # displaying a legend for the plotted lines.
plt.show()       # displaying the plot.
```



- **Through the bootstrapping method, we have been able to estimate the confidence interval for the total purchase made by each individual in age group 18 - 25 years on Black Friday at Walmart, despite having data for only 1069 individuals having age group 18 - 25 years. This provides us with a reasonable approximation of the range within which the total purchase of each individuals having age group 18 - 25 years falls, with a certain level of confidence.**

In [371…

```
print(f"The population mean of total spending of each customer in age group 18 - 25
```

```
The population mean of total spending of each customer in age group 18 - 25 will be
approximately = 854314.88
```

For Age Group 26 - 35 years

In [372…

```
df_age_26_to_35
```

Out[372…

	User_ID	Total_Purchase
0	1000003	341635
1	1000005	821001
2	1000008	796593
3	1000009	594099
4	1000011	557023
...	...	...
2048	1006030	737361
2049	1006034	197086
2050	1006035	956645
2051	1006036	4116058
2052	1006040	1653299

2053 rows × 2 columns

How the deviations vary for different sample sizes ?

In [373…

```
# The code snippet performs a loop to calculate the mean purchase for different sam

mean_purchases = []
for sample_size in range(50, 2000):
    sample_mean = df_age_26_to_35['Total_Purchase'].sample(sample_size).mean()
    mean_purchases.append(sample_mean)

# It iterates over a range of sample sizes from 50 to 2000, and for each iteration,
    # it takes a random sample of the specified size from the 'Total_Purchase' colu
    # of the 'df_age_26_to_35' DataFrame and calculates the mean of the sampled val
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [374…

```
# Creating a plot using matplotlib to visualize the trend of the mean purchases as

plt.figure(figsize = (15, 6))
plt.title('Behavior of mean purchases as the sample size increases')
plt.plot(np.arange(50, 2000), mean_purchases)
plt.xlabel('Sample size')
plt.ylabel('Mean Purchase Value')
plt.show()
```

- **It can be inferred from the above plot that as the sample size is small the deviations are fairly high.

As the sample size increases, the deviation becomes smaller and smaller. The deviations will be small if the sample size taken is greater than 1250.**

Finding the confidence interval of total spending for each individual in the age group 26 - 35 on the Black Friday

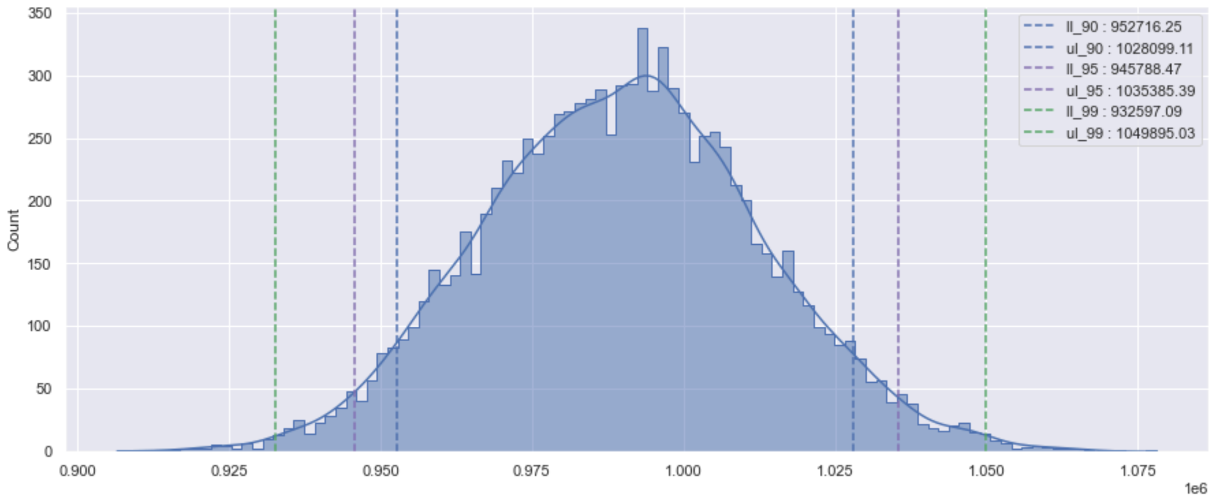
In [375…

```
means = []
size = df_age_26_to_35['Total_Purchase'].shape[0]
for bootstrapped_sample in range(10000):
    sample_mean = df_age_26_to_35['Total_Purchase'].sample(size, replace = True).me
    means.append(sample_mean)
```

In [376…

```
# The below code generates a histogram plot with kernel density estimation and adds

plt.figure(figsize = (15, 6))     # setting the figure size of the plot

sns.histplot(means, kde = True, bins = 100, fill = True, element = 'step')

# Above line plots a histogram of the data contained in the `means` variable.
    # The `kde=True` argument adds a kernel density estimation line to the plot.
    # The `bins=100` argument sets the number of bins for the histogram



# Above line calculates the z-score corresponding to the 90% confidence level using
    # inverse of the cumulative distribution function (CDF) of a standard normal di

ll_90 = np.percentile(means, 5)
    # calculating the lower limit of the 90% confidence interval
ul_90 = np.percentile(means, 95)
    # calculating the upper limit of the 90% confidence interval
plt.axvline(ll_90, label = f'll_90 : {round(ll_90, 2)}', linestyle = '--')
    # adding a vertical line at the lower limit of the 90% confidence interval
plt.axvline(ul_90, label = f'ul_90 : {round(ul_90, 2)}', linestyle = '--')
    # adding a vertical line at the upper limit of the 90% confidence interval
```

```
# Similar steps are repeated for calculating and plotting the 95% and 99% confidenc
    # with different line colors (`color='m'` for 95% and `color='g'` for 99%)

ll_95 = np.percentile(means, 2.5)
ul_95 = np.percentile(means, 97.5)
plt.axvline(ll_95, label = f'll_95 : {round(ll_95, 2)}', linestyle = '--', color =
plt.axvline(ul_95, label = f'ul_95 : {round(ul_95, 2)}', linestyle = '--', color =


ll_99 = np.percentile(means, 0.5)
ul_99 = np.percentile(means, 99.5)
plt.axvline(ll_99, label = f'll_99 : {round(ll_99, 2)}', linestyle = '--', color =
plt.axvline(ul_99, label = f'ul_99 : {round(ul_99, 2)}', linestyle = '--', color =

plt.legend()     # displaying a legend for the plotted lines.
plt.show()       # displaying the plot.
```



- **Through the bootstrapping method, we have been able to estimate the confidence interval for the total purchase made by each individual in age group 26 - 35 years on Black Friday at Walmart, despite having data for only 2053 individuals having age group 26 - 35 years. This provides us with a reasonable approximation of the range within which the total purchase of each individuals having age group 26 - 35 years falls, with a certain level of confidence.**

In [377…

```
print(f"The population mean of total spending of each customer in age group 26 - 35
```

```
The population mean of total spending of each customer in age group 26 - 35 will be
approximately = 989880.27
```

For Age Group 36 - 45 years

In [378…

```
df_age_36_to_45
```

Out[378…

	User_ID	Total_Purchase
0	1000007	234668
1	1000010	2169510
2	1000014	127629
3	1000016	150490
4	1000023	1670998
...	...	...
1162	1006011	1198714
1163	1006012	127920
1164	1006017	160230
1165	1006018	975585
1166	1006026	490768

1167 rows × 2 columns

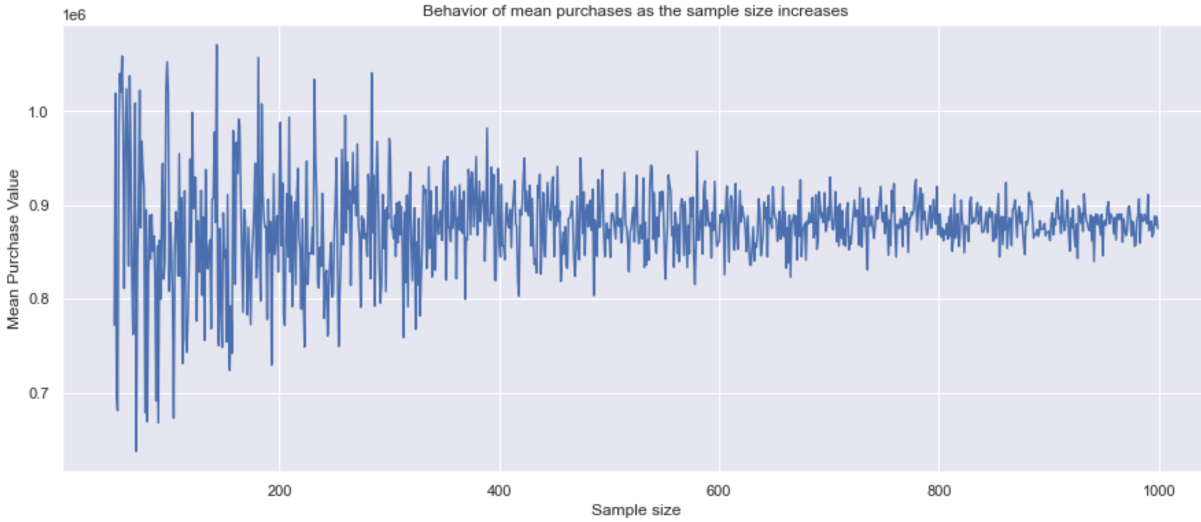
How the deviations vary for different sample sizes ?

In [379…

```
# The code snippet performs a loop to calculate the mean purchase for different
    # sample sizes of customers with age group 36 - 45 yrs.

mean_purchases = []
for sample_size in range(50, 1000):
    sample_mean = df_age_36_to_45['Total_Purchase'].sample(sample_size).mean()
    mean_purchases.append(sample_mean)

# It iterates over a range of sample sizes from 50 to 1000, and for each iteration,
    # it takes a random sample of the specified size from the 'Total_Purchase' colu
    # of the 'df_age_36_to_45' DataFrame and calculates the mean of the sampled val
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [380…

```
# Creating a plot using matplotlib to visualize the trend of the mean purchases
    # as the sample size increases

plt.figure(figsize = (15, 6))
plt.title('Behavior of mean purchases as the sample size increases')
plt.plot(np.arange(50, 1000), mean_purchases)
plt.xlabel('Sample size')
plt.ylabel('Mean Purchase Value')
plt.show()
```



- **It can be inferred from the above plot that as the sample size is small the deviations are fairly high.

As the sample size increases, the deviation becomes smaller and smaller. The deviations will be small if the sample size taken is greater than 600.**

Finding the confidence interval of total spending for each individual in the age group 36 - 45 on the Black Friday

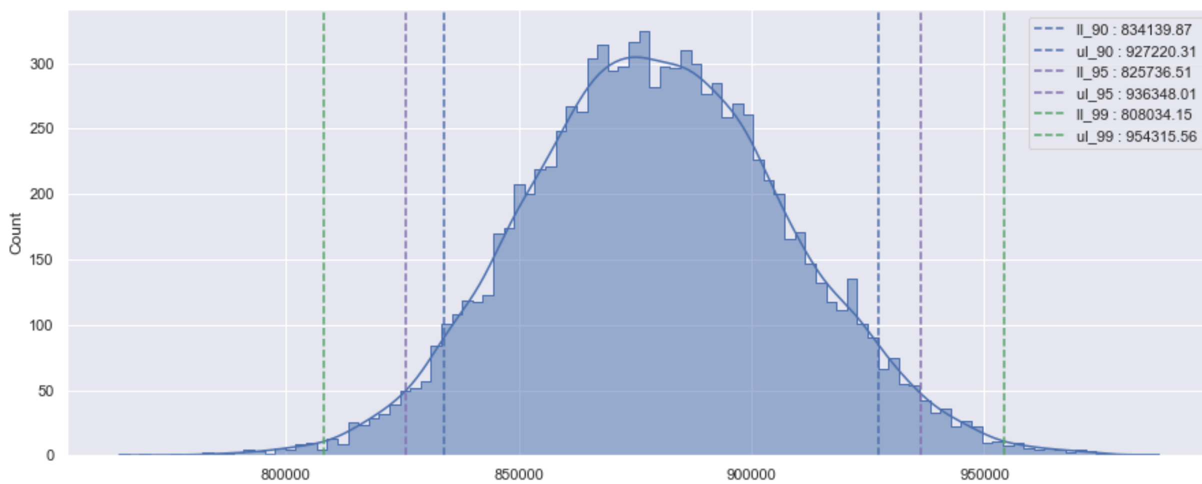
In [381…

```
means = []
size = df_age_36_to_45['Total_Purchase'].shape[0]
for bootstrapped_sample in range(10000):
    sample_mean = df_age_36_to_45['Total_Purchase'].sample(size, replace = True).me
    means.append(sample_mean)
```

In [382…

```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% le

plt.figure(figsize = (15, 6))     # setting the figure size of the plot

sns.histplot(means, kde = True, bins = 100, fill = True, element = 'step')

# Above line plots a histogram of the data contained in the `means` variable.
    # The `kde=True` argument adds a kernel density estimation line to the plot.
    # The `bins=100` argument sets the number of bins for the histogram


# Above line calculates the z-score corresponding to the 90% confidence level using
    # inverse of the cumulative distribution function (CDF) of a standard normal di

ll_90 = np.percentile(means, 5)
    # calculating the lower limit of the 90% confidence interval
ul_90 = np.percentile(means, 95)
    # calculating the upper limit of the 90% confidence interval
plt.axvline(ll_90, label = f'll_90 : {round(ll_90, 2)}', linestyle = '--')
    # adding a vertical line at the lower limit of the 90% confidence interval
plt.axvline(ul_90, label = f'ul_90 : {round(ul_90, 2)}', linestyle = '--')
```

```
    # adding a vertical line at the upper limit of the 90% confidence interval

# Similar steps are repeated for calculating and plotting the 95% and 99% confidenc
    # with different line colors (`color='m'` for 95% and `color='g'` for 99%)

ll_95 = np.percentile(means, 2.5)
ul_95 = np.percentile(means, 97.5)
plt.axvline(ll_95, label = f'll_95 : {round(ll_95, 2)}', linestyle = '--', color =
plt.axvline(ul_95, label = f'ul_95 : {round(ul_95, 2)}', linestyle = '--', color =


ll_99 = np.percentile(means, 0.5)
ul_99 = np.percentile(means, 99.5)
plt.axvline(ll_99, label = f'll_99 : {round(ll_99, 2)}', linestyle = '--', color =
plt.axvline(ul_99, label = f'ul_99 : {round(ul_99, 2)}', linestyle = '--', color =

plt.legend()     # displaying a legend for the plotted lines.
plt.show()       # displaying the plot.
```



- **Through the bootstrapping method, we have been able to estimate the confidence interval for the total purchase made by each individual in age group 36 - 45 years on Black Friday at Walmart, despite having data for only 1167 individuals having age group 36 - 45 years. This provides us with a reasonable approximation of the range within which the total purchase of each individuals having age group 36 - 45 years falls, with a certain level of confidence.**

In [383…
```
print(f"The population mean of total spending of each customer in age group 36 - 45
```

```
The population mean of total spending of each customer in age group 36 - 45 will be
approximately = 879821.37
```

For Age Group 46 - 50 years

In [384…
```
df_age_46_to_50
```

Out[384...

	User_ID	Total_Purchase
0	1000004	206468
1	1000013	713927
2	1000033	1940418
3	1000035	821303
4	1000044	1180380
...	...	...
526	1006014	528238
527	1006016	3770970
528	1006032	517261
529	1006037	1119538
530	1006039	590319

531 rows × 2 columns

How the deviations vary for different sample sizes ?

In [385...

```
# The code snippet performs a loop to calculate the mean purchase for different
    # sample sizes of customers with age group 46 - 50 yrs.

mean_purchases = []
for sample_size in range(50, 500):
    sample_mean = df_age_46_to_50['Total_Purchase'].sample(sample_size).mean()
    mean_purchases.append(sample_mean)

# It iterates over a range of sample sizes from 50 to 500, and for each iteration,
    # it takes a random sample of the specified size from the 'Total_Purchase' colu
    # of the 'df_age_46_to_50' DataFrame and calculates the mean of the sampled val
    # The calculated mean values are then stored in the 'mean_purchases' list
```

In [386...

```
# Creating a plot using matplotlib to visualize the trend of the mean purchases
    # as the sample size increases

plt.figure(figsize = (15, 6))
plt.title('Behavior of mean purchases as the sample size increases')
plt.plot(np.arange(50, 500), mean_purchases)
plt.xlabel('Sample size')
plt.ylabel('Mean Purchase Value')
plt.show()
```

- **It can be inferred from the above plot that as the sample size is small the deviations are fairly high.

As the sample size increases, the deviation becomes smaller and smaller. The deviations will be small if the sample size taken is greater than 300.**

Finding the confidence interval of total spending for each individual in the age group 46 - 50 on the Black Friday

In [387…

```
means = []
size = df_age_46_to_50['Total_Purchase'].shape[0]
for bootstrapped_sample in range(10000):
    sample_mean = df_age_46_to_50['Total_Purchase'].sample(size, replace = True).me
    means.append(sample_mean)
```

In [1]:

```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% le

plt.figure(figsize = (15, 6))     # setting the figure size of the plot

sns.histplot(means, kde = True, bins = 100, fill = True, element = 'step')

# Above line plots a histogram of the data contained in the `means` variable.
    # The `kde=True` argument adds a kernel density estimation line to the plot.
    # The `bins=100` argument sets the number of bins for the histogram


# Above line calculates the z-score corresponding to the 90% confidence level using
    # inverse of the cumulative distribution function (CDF) of a standard normal di

ll_90 = np.percentile(means, 5)
    # calculating the lower limit of the 90% confidence interval
ul_90 = np.percentile(means, 95)
    # calculating the upper limit of the 90% confidence interval
plt.axvline(ll_90, label = f'll_90 : {round(ll_90, 2)}', linestyle = '--')
    # adding a vertical line at the lower limit of the 90% confidence interval
plt.axvline(ul_90, label = f'ul_90 : {round(ul_90, 2)}', linestyle = '--')
```

```
    # adding a vertical line at the upper limit of the 90% confidence interval

# Similar steps are repeated for calculating and plotting the 95% and 99% confidenc
    # with different line colors (`color='m'` for 95% and `color='g'` for 99%)

ll_95 = np.percentile(means, 2.5)
ul_95 = np.percentile(means, 97.5)
plt.axvline(ll_95, label = f'll_95 : {round(ll_95, 2)}', linestyle = '--', color =
plt.axvline(ul_95, label = f'ul_95 : {round(ul_95, 2)}', linestyle = '--', color =


ll_99 = np.percentile(means, 0.5)
ul_99 = np.percentile(means, 99.5)
plt.axvline(ll_99, label = f'll_99 : {round(ll_99, 2)}', linestyle = '--', color =
plt.axvline(ul_99, label = f'ul_99 : {round(ul_99, 2)}', linestyle = '--', color =

plt.legend()     # displaying a legend for the plotted lines.
plt.show()       # displaying the plot.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-c4a1106ddb17> in <module>
      2     # adds vertical lines to represent confidence intervals at 90%, 95%, and
99% level
      3
----> 4 plt.figure(figsize = (15, 6))     # setting the figure size of the plot
      5
      6 sns.histplot(means, kde = True, bins = 100, fill = True, element = 'step')

NameError: name 'plt' is not defined
```

- **Through the bootstrapping method, we have been able to estimate the confidence interval for the total purchase made by each individual in age group 46 - 50 years on Black Friday at Walmart, despite having data for only 531 individuals having age group 46 - 50 years. This provides us with a reasonable approximation of the range within which the total purchase of each individuals having age group 46 - 50 years falls, with a certain level of confidence.**

In [389…

```
print(f"The population mean of total spending of each customer in age group 46 - 50
```

```
The population mean of total spending of each customer in age group 46 - 50 will be
approximately = 793105.85
```

In [ ]:

```
fig,ax = plt.subplots(2,3 , figsize = (20,10))
col =["Gender" , "Age" ,"City_Category" ,"Stay_In_Current_City_Years","Marital_Stat
k=0
for i in range(0,2):
  for j in range(0,3):
    sns.countplot(data=df.sort_values(by=col[k]) , x = col[k] , ax = ax[i,j] )
    k+=1
```

KEY TAKEAWAYS

- **Men spend more money than women, so the company can focus on retaining male customers and getting more male customers.** There are 1666 unique female customers and 4225 unique male customers. The average number of transactions made by each Male on Black Friday is 98 while for Female it is 82. Out of every four transactions made on Black Friday in Walmart stores, three are made by the males and the females make one. On average each male makes a total purchase of 925438.92 on Black Friday while for each female the figure is 712269.56.
- 82.43% of the total transactions are made for only 5 Product Categories. These are 5, 1, 8, 11 and 2. It means these are the products in these categories are in more demand. The company should focus on selling more of these products.
- **Unmarried customers spend more money than married customers, so the company should focus on the acquisition of Unmarried customers.** Out of 5891 unique customers, 42 % of them are married and 58 % of them are Single. The average number of transactions made by each user with marital status Married is 91 and for Single, it is 95. On average, each Married customer makes a total purchase of 843469.79 on Black Friday while for each Single customer, the figure is 880526.31. 59.05 % of the total revenue is generated from the Single customers.
- Customers aged 26-45 spend more money than others, So the company should focus on the acquisition of customers who are aged 26-45.
- About 81.82% of the total transactions are made by customers of age between 18 and 50 years.
- The company generated 86.21 % of total revenue from customers in the range of 18 to 50 years on Black Friday.
- The majority of the total unique customers belong to city C. 82.26 % of the total unique customers belong to cities C and B.
- The company generated 41.52 % of the total revenue from the customers belonging to City B, 32.65 % from City C, and 25.83 % from City A on Black Friday.
- The population mean of total spending of each male will be approximately = 925156.36.
- The population mean of total spending of each female will be approximately = 711789.37
- The population mean of total spending of each single will be approximately = 880356.19
- The population mean of total spending of each male will be approximately = 843632.08
- The population mean of total spending of each customer in the age group 0 -17 will be approximately = 617797.25
- The population mean of total spending of each customer in the age group 18 - 25 will be approximately = 854676.31
- The population mean of total spending of each customer in the age group 26 - 35 will be approximately = 989120.36
- The population mean of total spending of each customer in the age group 36 - 45 will be approximate = 879434.88

- The population mean of total spending of each customer in the age group 46 - 50 will be approximately = 792671.74
- For the occupations that are contributing more, the company can think of offering credit cards or other benefits to those customers by liaising with some financial partners to increase sales.
- Some of the Product categories like 19,20,13 have very less purchases. The company can think of dropping it.

Recommendations

- Since male customers account for a significant portion of Black Friday sales and tend to spend more per transaction on average, Walmart should tailor its marketing strategies and product offerings to incentivize higher spending among male customers while ensuring competitive pricing for female-oriented products.

- With the age group between 26 and 45 contributing to the majority of sales, Walmart should specifically cater to the preferences and needs of this demographic.This could include offering exclusive deals on products that are popular among this age group.

- Given that 82.33% of transactions come from customers in 11 specific occupations, it would be wise to focus marketing efforts on these occupations. Understanding the needs and preferences of individuals in these occupations can help in creating targeted marketing campaigns and customized offers.

- Since customers in the 18 - 25, 26 - 35, and 46 - 50 age groups exhibit similar buying characteristics, and so do the customers in 36 - 45 and 55+, Walmart can optimize its product selection to cater to the preferences of these age groups. Also, Walmart can use this information to adjust their pricing strategies for different age groups.

- As a significant portion of transactions (53.75%) come from customers who have recently moved to the current city, it presents an opportunity to engage with these new residents. Targeted marketing, welcoming offers, and incentives for newcomers can help capture their loyalty and increase their spending.

- The top products should be given focus in order to maintain the quality in order to further increase the sales of those products.

- Considering that customers aged 50+ have the highest spending per transaction, Walmart offer them exclusive pre-sale access, special discount or provide personalized product recommendations for this age group. Walmart can also introduce loyalty programs specifically designed to reward and retain customers in the above 50 age group.

- 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.