Business Case: Walmart - Confidence Interval and CLT ¶

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 libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
import scipy.stats as spy
```

Loading the dataset

In [2]:
```
df = pd.read_csv(r"https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/293/original/walmart_data.c
```

shape of data

In [3]:
```
df.shape
```

Out[3]:
```
(550068, 10)
```

columns present in the data

In [4]:
```
df.columns
```

Out[4]:
```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category',
       'Purchase'],
      dtype='object')
```

datatype of the each column

In [5]:
```
df.dtypes
```

Out[5]:
```
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 [6]:
```python
df.head()
```

Out[6]:

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

In [7]:
```python
df.tail()
```

Out[7]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purcha
550063	1006033	P00372445	M	51-55	13	B	1	1	20	
550064	1006035	P00375436	F	26-35	1	C	3	0	20	
550065	1006036	P00375436	F	26-35	15	B	4+	1	20	
550066	1006038	P00375436	F	55+	1	C	2	0	20	
550067	1006039	P00371644	F	46-50	0	B	4+	1	20	

Is there any missing value in the dataset?

In [8]:
```python
np.any(df.isna())
```

Out[8]: False

Is there any duplicate value in the dataset ?

In [9]:
```python
np.any(df.duplicated())
```

Out[9]: False

Basic information about the dataset

In [10]:
```python
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
```

Memory Optimization

Converting User_ID column datatype to int32

In [11]:
```python
df['User_ID'] = df['User_ID'].astype('int32')
```

Updating 'Marital_Status' column

In [12]:
```python
df['Marital_Status'] = df['Marital_Status'].apply(lambda x: 'Married' if x == 1 else 'Single')
```

In [13]:
```python
df['Marital_Status'] = df['Marital_Status'].astype('category')
```

Converting 'Age' column datatype to category

In [14]:
```python
df['Age'] = df['Age'].astype('category')
```

Converting 'Product_Category' column datatype to int8

In [15]:
```python
df['Product_Category'] = df['Product_Category'].astype('int8')
```

Converting 'Occupation' column's datatype to int8

In [16]:
```python
df['Occupation'] = df['Occupation'].astype('int8')
```

Converting 'City_Category' column's datatype to category

In [17]:
```python
df['City_Category'] = df['City_Category'].astype('category')
```

Converting 'Stay_In_Current_City_Years' column's datatype to category

In [18]:
```python
df['Stay_In_Current_City_Years'] = df['Stay_In_Current_City_Years'].astype('category')
```

In [19]:
```python
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
```

Earlier the dataframe took 42.0+ MB of memory but the memory usage is reduced to 17.8+ MB (57.62% reduction in the memory usage).

Basic statistical description of the dataset

In [20]:
```python
# For measurable quantities
df.describe()
```

Out[20]:

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

The dataset provides information on the following variables:

- **User_ID**: It contains unique identification numbers assigned to each user. The dataset includes a total of 550,068 user records.
- **Occupation**: This variable represents the occupation of the users. The dataset includes values ranging from 0 to 20, indicating different occupations.
- **Product_Category**: It indicates the category of the products purchased by the users. The dataset includes values ranging from 1 to 20, representing different product categories.
- **Purchase**: This variable represents the purchase amount made by each user. The dataset includes purchase values ranging from 12 to 23,961.

In [21]:
```
# description of columns with 'object' datatype
df.describe(include = 'object')
```

Out[21]:

	Product_ID	Gender
count	550068	550068
unique	3631	2
top	P00265242	M
freq	1880	414259

The provided data represents summary statistics for two variables: Product_ID and Gender. Here is a breakdown of the information:

- **Product_ID**: There are 3,631 unique values observed in this variable, indicating that there are 3,631 different products. The top value, which appears most frequently, is 'P00265242'. This value occurs 1,880 times in the dataset.
- **Gender**: There are 2 unique values in this variable, which suggests that it represents a binary category. The top value is 'M', indicating that 'M' is the most common gender category. It appears 414,259 times in the dataset.

These summary statistics provide insights into the distribution and frequency of the Product_ID and Gender variables. They give an understanding of the number of unique products, the most common product, and the dominant gender category in the dataset.

value_counts and unique attributes

In [22]:
```
# How many unique customers' data is given in the dataset?
df['User_ID'].nunique()
```

Out[22]:
```
5891
```

- We have the data of 5891 customers who made at least one purchase on Black Friday in Walmart.

In [23]:
```
# Total number of transactions made by each gender
np.round(df['Gender'].value_counts(normalize = True) * 100, 2)
```

Out[23]:
```
M    75.31
F    24.69
Name: Gender, dtype: float64
```

- It is clear from the above that out of every four transactions, three are made by males.

In [24]:
```
np.round(df['Occupation'].value_counts(normalize = True) * 100, 2).cumsum()
```

Out[24]:
```
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 [25]:
```
np.round(df['Stay_In_Current_City_Years'].value_counts(normalize = True) * 100, 2)
```

Out[25]:
```
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 [26]:
```
np.round(df['Product_Category'].value_counts(normalize = True).head(10) * 100, 2).cumsum()
```

Out[26]:
```
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.

How many unique customers are there for each gender

In [27]:
```
df_gender_dist = pd.DataFrame(df.groupby(by = ['Gender'])['User_ID'].nunique()).reset_index().rename(columns = {'
df_gender_dist['percent_share'] = np.round(df_gender_dist['unique_customers'] / df_gender_dist['unique_customers'
df_gender_dist
```

Out[27]:

	Gender	unique_customers	percent_share
0	F	1666	28.28
1	M	4225	71.72

How many transactions are made by each gender category ?

In [28]:
```
df.groupby(by = ['Gender'])['User_ID'].count()
```

Out[28]:
```
Gender
F    135809
M    414259
Name: User_ID, dtype: int64
```

In [29]:
```
print('Average number of transactions made by each Male on Black Friday is', round(414259 / 4225))
print('Average number of transactions made by each Female on Black Friday is', round(135809 / 1666))
```

```
Average number of transactions made by each Male on Black Friday is 98
Average number of transactions made by each Female on Black Friday is 82
```

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

In [30]:
```
df_gender_revenue = df.groupby(by = ['Gender'])['Purchase'].sum().to_frame().sort_values(by = 'Purchase', ascendi
df_gender_revenue['percent_share'] = np.round((df_gender_revenue['Purchase'] / df_gender_revenue['Purchase'].sum(
df_gender_revenue
```

Out[30]:

	Gender	Purchase	percent_share
0	M	3909580100	76.72
1	F	1186232642	23.28

What is the average total purchase made by each user in each gender ?

In [31]:
```
df1 = pd.DataFrame(df.groupby(by = ['Gender', 'User_ID'])['Purchase'].sum()).reset_index().rename(columns = {'Pur
df1.groupby(by = 'Gender')['Average_Purchase'].mean()
```

Out[31]:
```
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.
```

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

In [32]:
```
pd.DataFrame(df.groupby(by = 'Gender')['Purchase'].mean()).reset_index().rename(columns = {'Purchase' : 'Average_
```

Out[32]:

	Gender	Average_Purchase
0	F	8734.565765
1	M	9437.526040

How many unique customers are there for each Marital Status ?

In [33]:
```
df_marital_status_dist = pd.DataFrame(df.groupby(by = ['Marital_Status'])['User_ID'].nunique()).reset_index().ren
df_marital_status_dist['percent_share'] = np.round(df_marital_status_dist['unique_customers'] / df_marital_status_
df_marital_status_dist
```

Out[33]:

	Marital_Status	unique_customers	percent_share
0	Married	2474	42.0
1	Single	3417	58.0

How many transactions are made by each Marital Status category ?

In [34]:
```
df.groupby(by = ['Marital_Status'])['User_ID'].count()
```

Out[34]:
```
Marital_Status
Married    225337
Single     324731
Name: User_ID, dtype: int64
```

In [35]:
```
print('Average number of transactions made by each user with marital status Married is', round(225337 / 2474))
print('Average number of transactions made by each with marital status Single is', round(324731 / 3417))
```

```
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
```

What is the total Revenue generated by Walmart from each Marital Status ?

In [36]:
```
df_marital_status_revenue = df.groupby(by = ['Marital_Status'])['Purchase'].sum().to_frame().sort_values(by = 'Pu
df_marital_status_revenue['percent_share'] = np.round((df_marital_status_revenue['Purchase'] / df_marital_status_
df_marital_status_revenue
```

Out[36]:

	Marital_Status	Purchase	percent_share
0	Single	3008927447	59.05
1	Married	2086885295	40.95

What is the average total purchase made by each user in each marital status ?

In [37]:
```
df1 = pd.DataFrame(df.groupby(by = ['Marital_Status', 'User_ID'])['Purchase'].sum()).reset_index().rename(columns
df1.groupby(by = 'Marital_Status')['Average_Purchase'].mean()
```

Out[37]:
```
Marital_Status
Married    354249.753013
Single     510766.838737
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 [38]:
```
df_age_dist = pd.DataFrame(df.groupby(by = ['Age'])['User_ID'].nunique()).reset_index().rename(columns = {'User_I
df_age_dist['percent_share'] =  np.round(df_age_dist['unique_customers'] / df_age_dist['unique_customers'].sum()
df_age_dist['cumulative_percent'] =  df_age_dist['percent_share'].cumsum()
df_age_dist
```

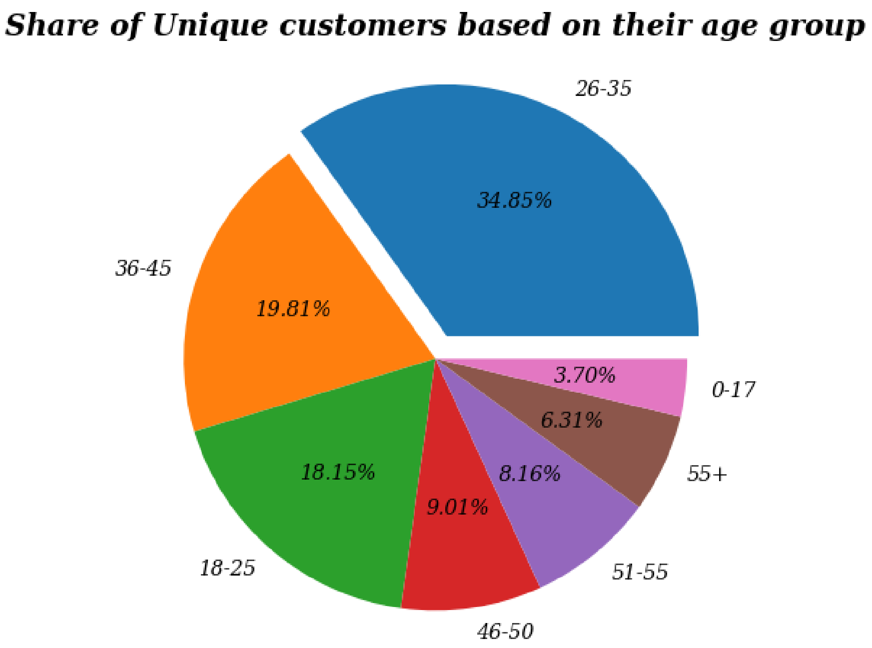
Out[38]:

	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.
```

In [39]:
```
plt.figure(figsize = (8, 8))
plt.title('Share of Unique customers based on their age group', fontdict = {'fontsize' : 20,
                    'fontstyle' : 'oblique',
                    'fontfamily' : 'serif',
                    'fontweight' : 600} )
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.plot()
```

Out[39]: []



In [40]:
```
df['Age'].value_counts()
```

Out[40]:
```
26-35    219587
36-45    110013
18-25     99660
46-50     45701
51-55     38501
55+       21504
0-17      15102
Name: Age, dtype: int64
```

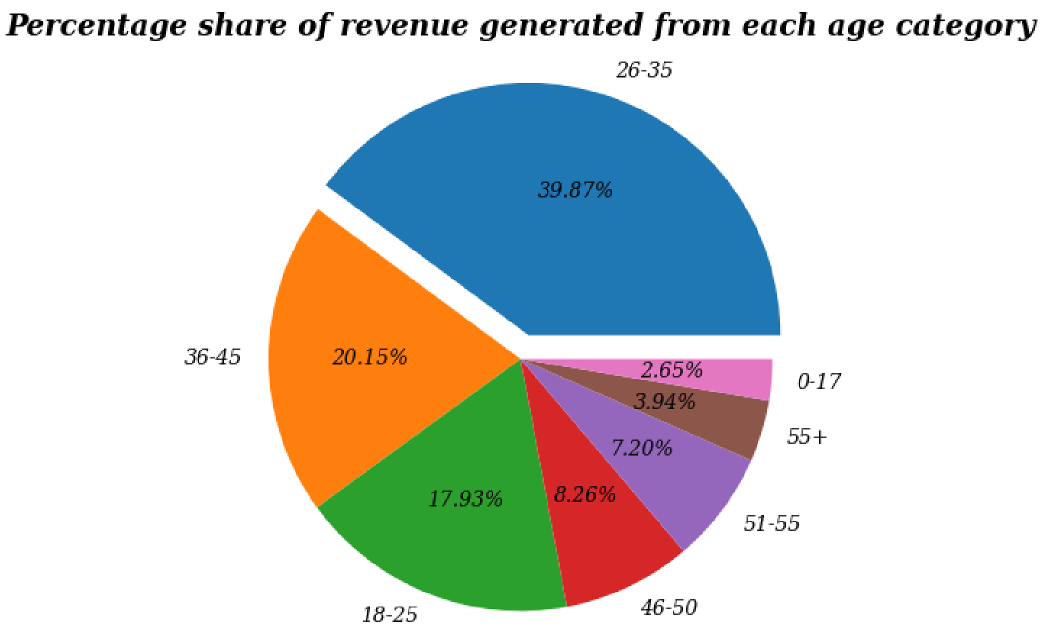
In [41]:
```
df_age_revenue = pd.DataFrame(df.groupby(by = 'Age', as_index = False)['Purchase'].sum()).sort_values(by = 'Purch
df_age_revenue['percent_share'] = np.round((df_age_revenue['Purchase'] / df_age_revenue['Purchase'].sum()) * 100,
df_age_revenue['cumulative_percent_share'] = df_age_revenue['percent_share'].cumsum()
df_age_revenue
```

Out[41]:

	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 [42]: plt.figure(figsize = (8, 8))
         plt.title('Percentage share of revenue generated from each age category', fontdict = {'fontsize' : 20,
                           'fontstyle' : 'oblique',
                           'fontfamily' : 'serif',
                           'fontweight' : 600} )
         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.plot()
```

```
Out[42]: []
```




```
In [43]: df_city_dist = pd.DataFrame(df.groupby(by = ['City_Category'])['User_ID'].nunique()).reset_index().rename(columns
         df_city_dist['percent_share'] = np.round((df_city_dist['unique_customers'] / df_city_dist['unique_customers'].sum
         df_city_dist['cumulative_percent_share'] = df_city_dist['percent_share'].cumsum()
         df_city_dist
```

Out[43]:

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

```
Out[44]: B    231173
         C    171175
         A    147720
         Name: City_Category, dtype: int64
```

What is the revenue generated from different cities ?

In [45]:
```
df_city_revenue = df.groupby(by = ['City_Category'])['Purchase'].sum().to_frame().sort_values(by = 'Purchase', as
df_city_revenue['percent_share'] = np.round((df_city_revenue['Purchase'] / df_city_revenue['Purchase'].sum()) * 10
df_city_revenue['cumulative_percent_share'] = df_city_revenue['percent_share'].cumsum()
df_city_revenue
```

Out[45]:

	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 [46]:
```
df.groupby(by = ['Product_Category'])['Product_ID'].nunique()
```

Out[46]:
```
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
```

What is the revenue generated from different product categories ?

In [47]:
```
df_product_revenue = df.groupby(by = ['Product_Category'])['Purchase'].sum().to_frame().sort_values(by = 'Purchas
df_product_revenue['percent_share'] = np.round((df_product_revenue['Purchase'] / df_product_revenue['Purchase'].s
df_product_revenue['cumulative_percent_share'] = df_product_revenue['percent_share'].cumsum()
df_product_revenue
```

Out[47]:

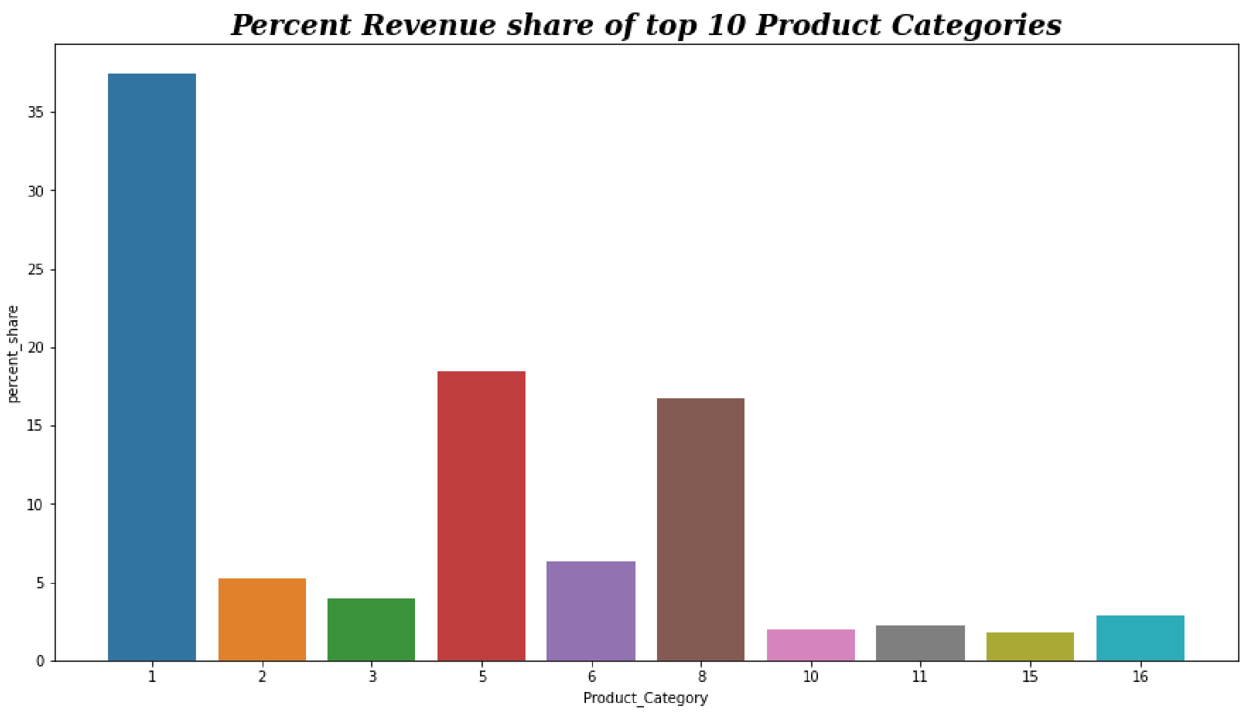
	Product_Category	Purchase	percent_share	cumulative_percent_share
0	1	1910013754	37.48	37.48
1	5	941835229	18.48	55.96
2	8	854318799	16.77	72.73
3	6	324150302	6.36	79.09
4	2	268516186	5.27	84.36
5	3	204084713	4.00	88.36
6	16	145120612	2.85	91.21
7	11	113791115	2.23	93.44
8	10	100837301	1.98	95.42
9	15	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 [48]:
```python
top5 = df_product_revenue.head(5)['Purchase'].sum() /  df_product_revenue['Purchase'].sum()
top5 = np.round(top5 * 100, 2)
print(f'Top 5 product categories from which Walmart makes {top5} % of total revenue are : {list(df_product_revenu
```

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

In [49]:
```python
plt.figure(figsize = (15, 8))
plt.title('Percent Revenue share of top 10 Product Categories', fontsize = 20, fontweight = 600, fontfamily = 'se
sns.barplot(data = df_product_revenue, x = df_product_revenue.head(10)['Product_Category'], y = df_product_revenu
plt.plot()
```

Out[49]: []



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

In [50]:
```python
df_gender_revenue = df.groupby(by = ['Gender'])['Purchase'].sum().to_frame().sort_values(by = 'Purchase', ascendi
df_gender_revenue['percent_share'] = np.round((df_gender_revenue['Purchase'] / df_gender_revenue['Purchase'].sum(
df_gender_revenue
```

Out[50]:

	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 [51]:
```python
pd.DataFrame(df.groupby(by = 'Gender')['Purchase'].mean()).reset_index().rename(columns = {'Purchase' : 'Average_
```

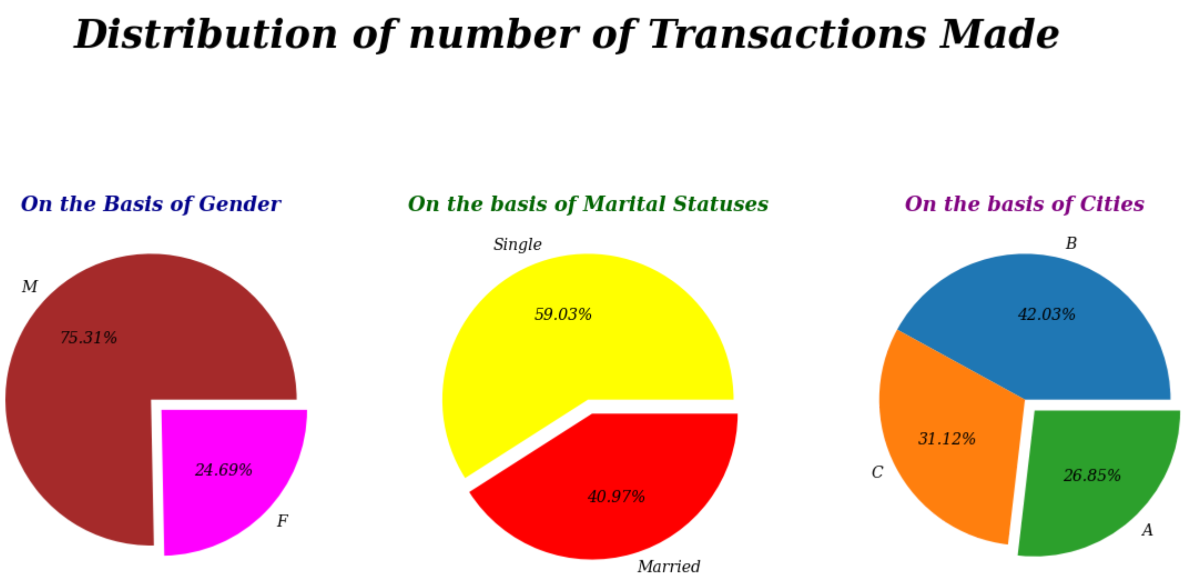
Out[51]:

	Gender	Average_Purchase
0	F	8734.565765
1	M	9437.526040

Distribution of number of Transactions :

In [52]:
```
plt.figure(figsize = (20, 10))
plt.suptitle('Distribution of number of Transactions Made', fontsize = 35, fontweight = 600, fontfamily = 'serif'
plt.subplot(1, 3, 1)
plt.title('On the Basis of Gender', color = 'darkblue', fontdict = {'fontsize' : 18,
                                                  'fontweight' : 600,
                                                  'fontstyle' : 'oblique',
                                                  'fontfamily' : 'serif'})
df_gender_dist = np.round(df['Gender'].value_counts(normalize = True) * 100, 2)
plt.pie(x = df_gender_dist.values, labels = df_gender_dist.index,
        explode = [0, 0.1], autopct = '%.2f%%',
       textprops = {'fontsize' : 14,
                    'fontstyle' : 'oblique',
                    'fontfamily' : 'serif',
                    'fontweight' : 500},
       colors = ['brown', 'magenta'])
plt.plot()
plt.subplot(1, 3, 2)
plt.title('On the basis of Marital Statuses', color = 'darkgreen', fontdict = {'fontsize' : 18,
                                                  'fontweight' : 600,
                                                  'fontstyle' : 'oblique',
                                                  'fontfamily' : 'serif'})
df_Marital_Status_dist = np.round(df['Marital_Status'].value_counts(normalize = True) * 100, 2)
plt.pie(x = df_Marital_Status_dist.values, labels = df_Marital_Status_dist.index,
        explode = [0, 0.1], autopct = '%.2f%%',
       textprops = {'fontsize' : 14,
                    'fontstyle' : 'oblique',
                    'fontfamily' : 'serif',
                    'fontweight' : 500},
       colors = ['yellow', 'red'])
plt.plot()
plt.subplot(1, 3, 3)
plt.title("On the basis of Cities", color = 'purple', fontdict = {'fontsize' : 18,
                                                  'fontweight' : 555,
                                                  'fontstyle' : 'oblique',
                                                  'fontfamily' : 'serif'})
df_City_Category_dist = np.round(df['City_Category'].value_counts(normalize = True) * 100, 2)
plt.pie(x = df_City_Category_dist.values, labels = df_City_Category_dist.index,
        explode = [0, 0, 0.1], autopct = '%.2f%%',
       textprops = {'fontsize' : 14,
                    'fontstyle' : 'oblique',
                    'fontfamily' : 'serif',
                    'fontweight' : 500})
plt.plot()
```

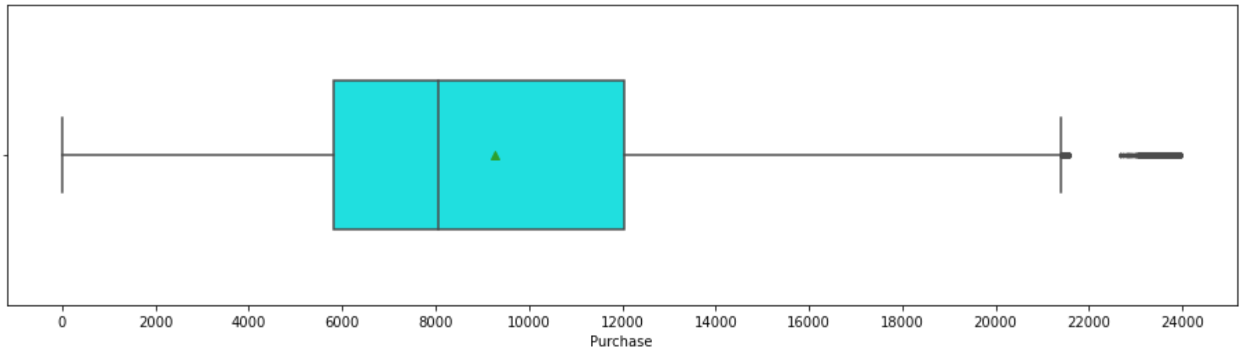
Out[52]: []



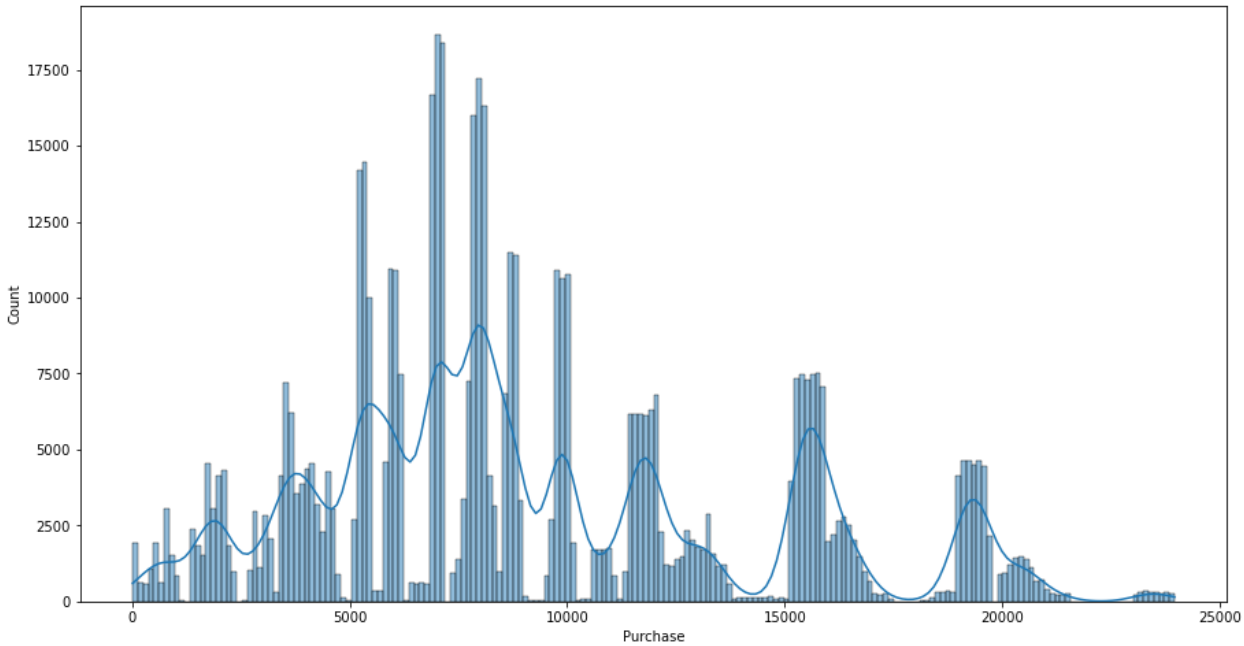
Univariate Analysis

In [53]:
```
plt.figure(figsize = (16, 4))
sns.boxplot(data = df,
            x = 'Purchase',
            showmeans = True,
            fliersize = 2,
            width = 0.5,
            color = np.random.choice(['magenta', 'lightgreen', 'cyan']))
plt.xticks(np.arange(0, 25001, 2000))
plt.plot()
```

Out[53]: []

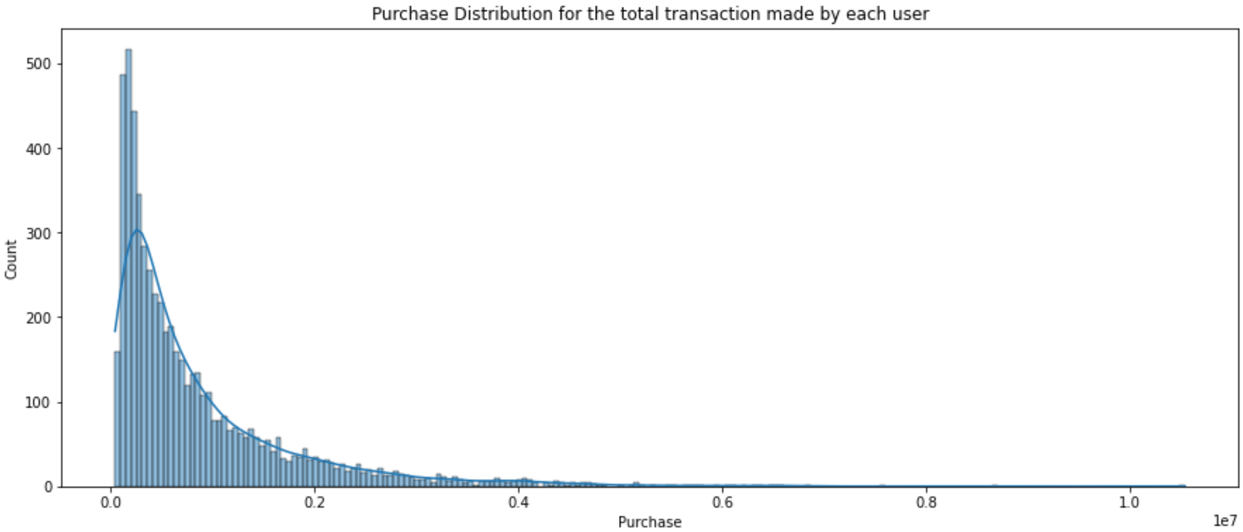


In [54]:
```
plt.figure(figsize = (15, 8))
sns.histplot(data = df, x = 'Purchase', kde = True, bins = 200)
plt.show()
```

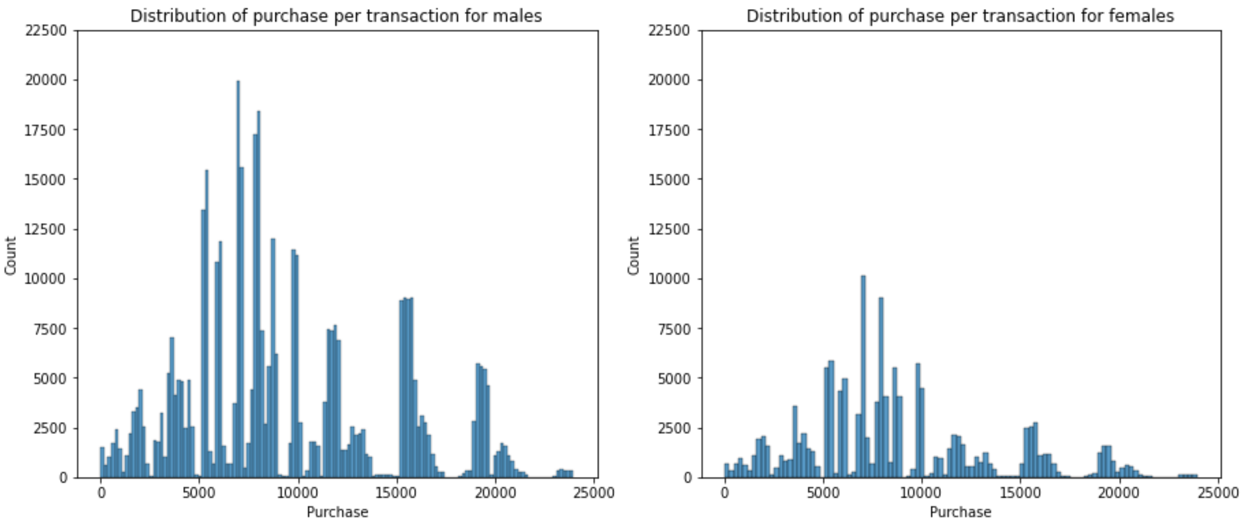


In [55]:
```
plt.figure(figsize = (15, 6))
plt.title('Purchase Distribution for the total transaction made by each user')
df_customer = df.groupby(by = 'User_ID')['Purchase'].sum()
sns.histplot(data = df_customer, kde = True, bins = 200)
plt.plot()
```

Out[55]: []



In [56]:
```
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 [57]:
```
df_cust_gender = pd.DataFrame(df.groupby(by = ['Gender', 'User_ID'])['Purchase'].sum()).reset_index().rename(colu
df_cust_gender
```

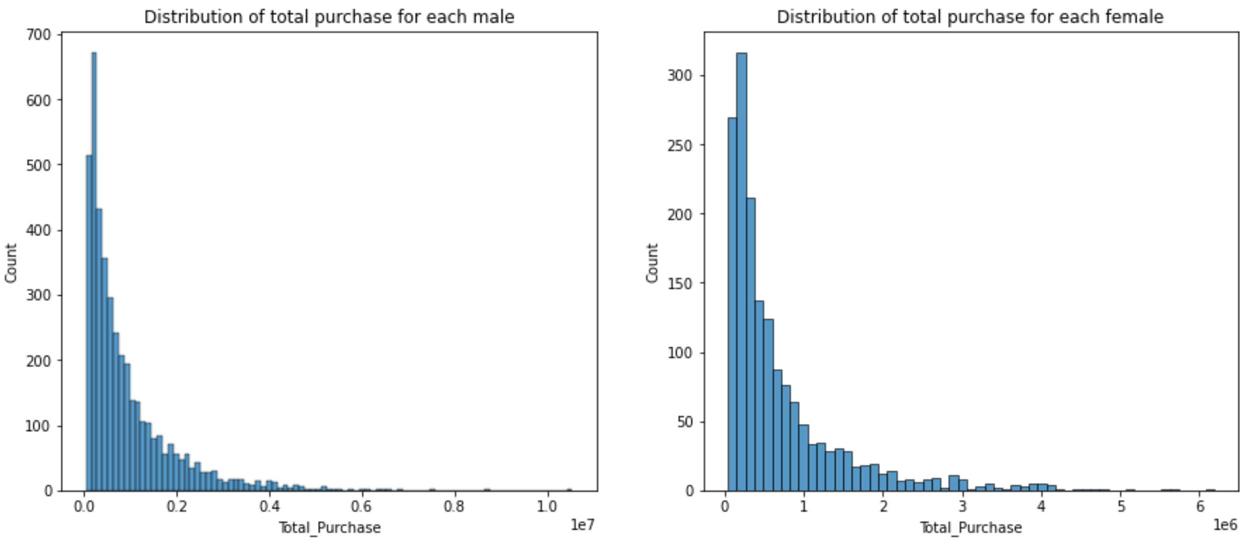
Out[57]:

	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

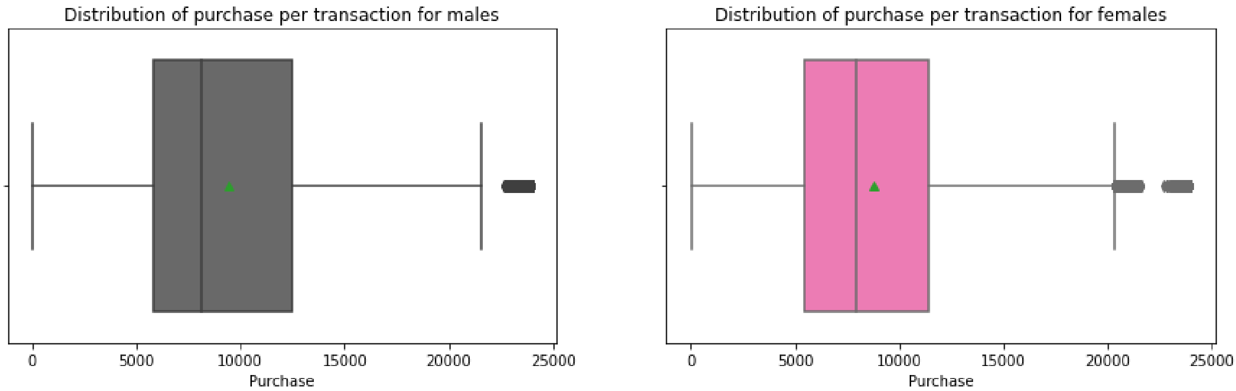
5891 rows × 3 columns

In [58]:
```
df_male_customer = df_cust_gender.loc[df_cust_gender['Gender'] == 'M']
df_female_customer = df_cust_gender.loc[df_cust_gender['Gender'] == 'F']
```

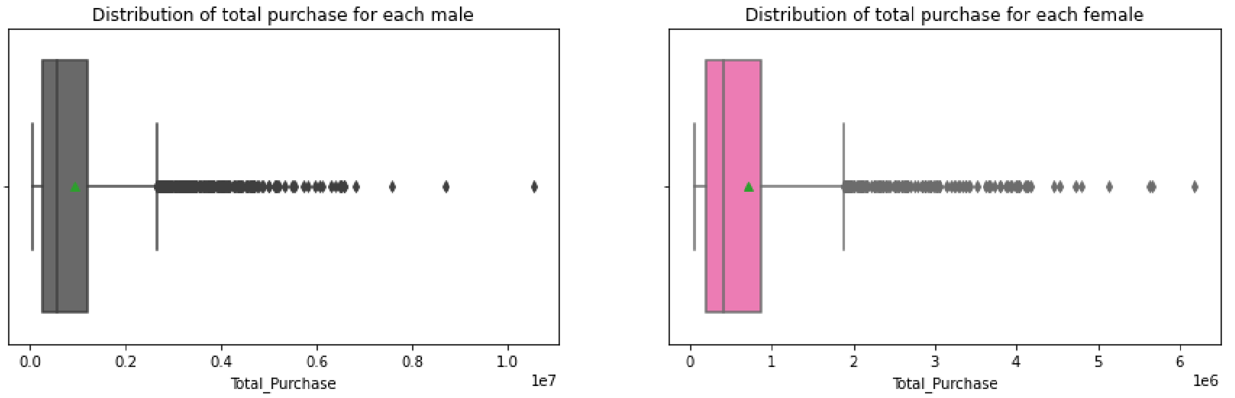
In [59]:
```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.title('Distribution of total purchase for each male')
sns.histplot(data = df_male_customer, x = 'Total_Purchase')
plt.subplot(1, 2 ,2)
plt.title('Distribution of total purchase for each female')
df_female = df[df['Gender'] == 'F']
sns.histplot(data = df_female_customer, x = 'Total_Purchase')
plt.show()
```



In [60]:
```
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 [61]:
```
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 = 'dimgray')
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, color = 'hotpink')
plt.show()
```

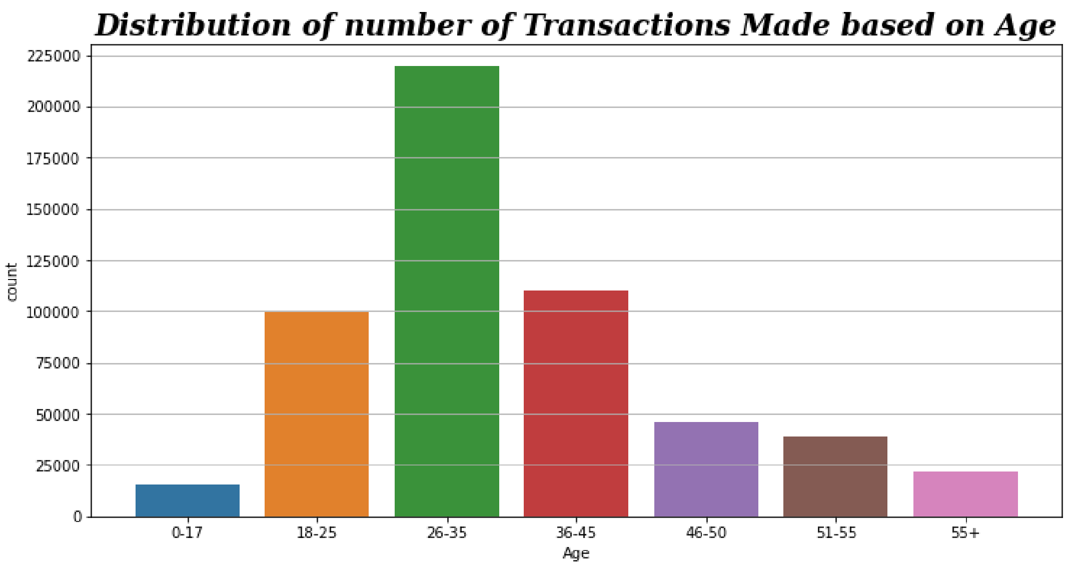


In [62]:
```
df['Age'].unique()
```

Out[62]:
```
['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25']
Categories (7, object): ['0-17', '18-25', '26-35', '36-45', '46-50', '51-55', '55+']
```

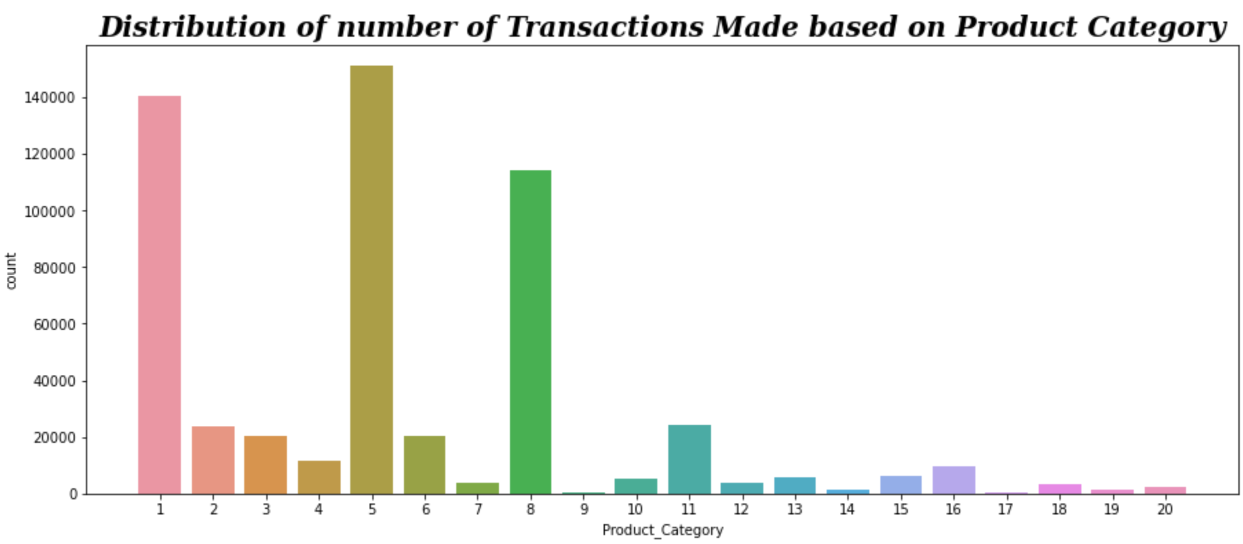
In [63]:
```
plt.figure(figsize = (12, 6))
plt.title('Distribution of number of Transactions Made based on Age',
          fontsize = 20,
          fontweight = 600,
          fontstyle = 'oblique',
          fontfamily = 'serif')
plt.yticks(np.arange(0, 250001, 25000))
plt.grid('y')
sns.countplot(data = df, x = 'Age',
              order = ['0-17', '18-25', '26-35', '36-45', '46-50', '51-55', '55+'])
plt.plot()
```

Out[63]: []



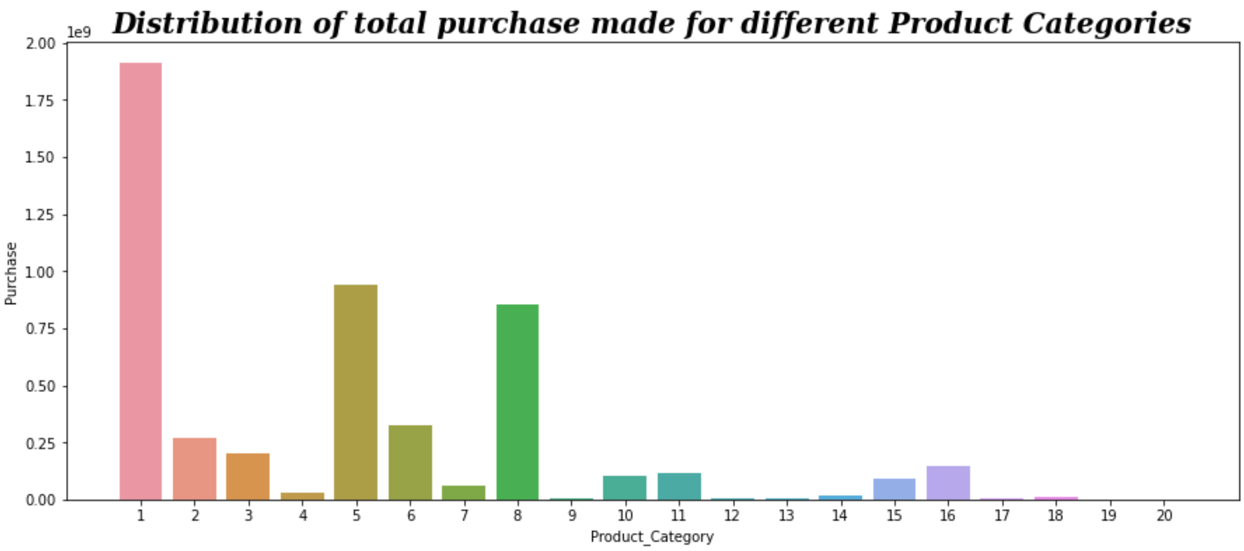
In [64]:
```
plt.figure(figsize = (15, 6))
plt.title('Distribution of number of Transactions Made based on Product Category', fontsize = 20, fontweight = 60
sns.countplot(data = df, x = 'Product_Category')
plt.plot()
```

Out[64]: []



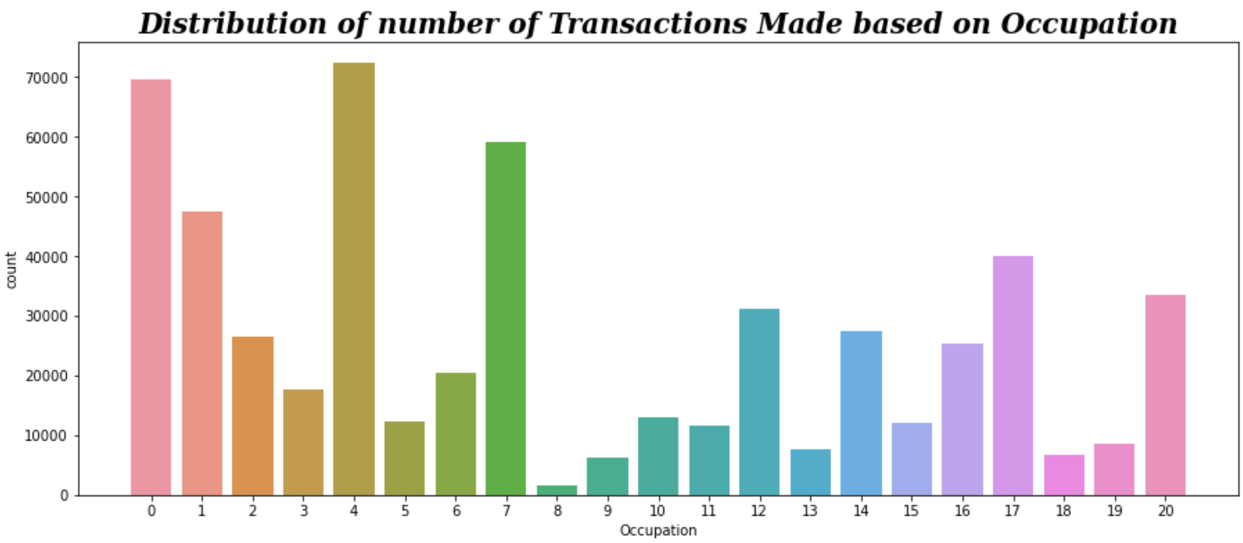
In [65]:
```
df_product_category = df.groupby(by = 'Product_Category')['Purchase'].sum().to_frame().reset_index()
plt.figure(figsize = (15, 6))
plt.title('Distribution of total purchase made for different Product Categories',
          fontsize = 20,
          fontweight = 600,
          fontstyle = 'oblique',
          fontfamily = 'serif')
sns.barplot(data = df_product_category, x = 'Product_Category', y = 'Purchase')
plt.plot()
```

Out[65]: []



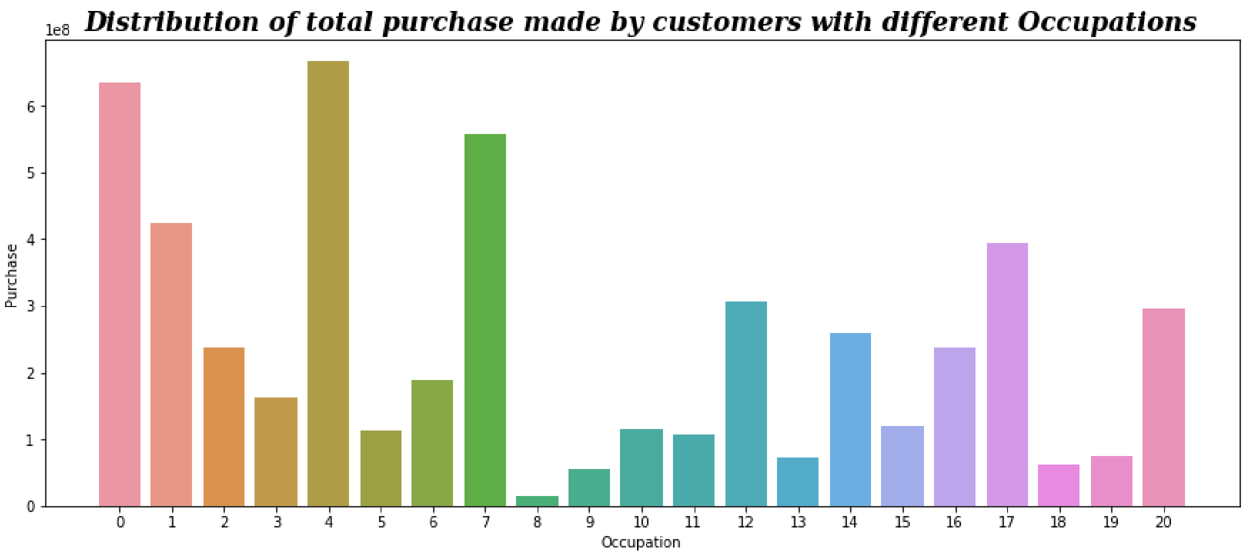
In [66]:
```
plt.figure(figsize = (15, 6))
plt.title('Distribution of number of Transactions Made based on Occupation',
          fontsize = 20,
          fontweight = 600,
          fontstyle = 'oblique',
          fontfamily = 'serif')
sns.countplot(data = df, x = 'Occupation')
plt.plot()
```

Out[66]: []



In [67]:
```
df_occupation = df.groupby(by = 'Occupation')['Purchase'].sum().to_frame().reset_index()
plt.figure(figsize = (15, 6))
plt.title('Distribution of total purchase made by customers with different Occupations',
          fontsize = 18,
          fontweight = 600,
          fontstyle = 'oblique',
          fontfamily = 'serif')
sns.barplot(data = df_occupation, x = 'Occupation', y = 'Purchase')
plt.plot()
```

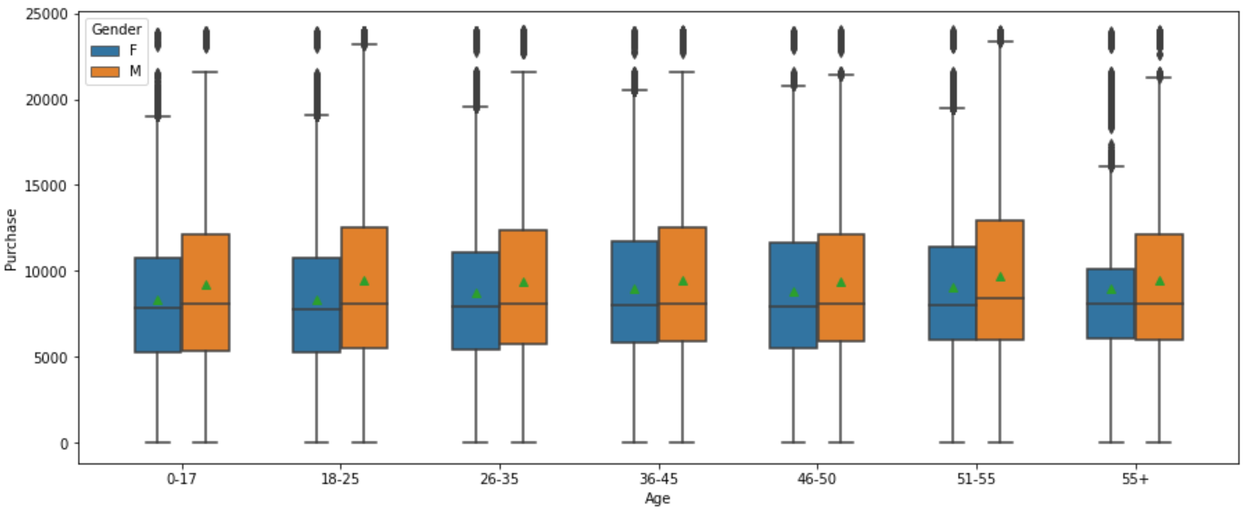
Out[67]: []



Bivariate Analysis

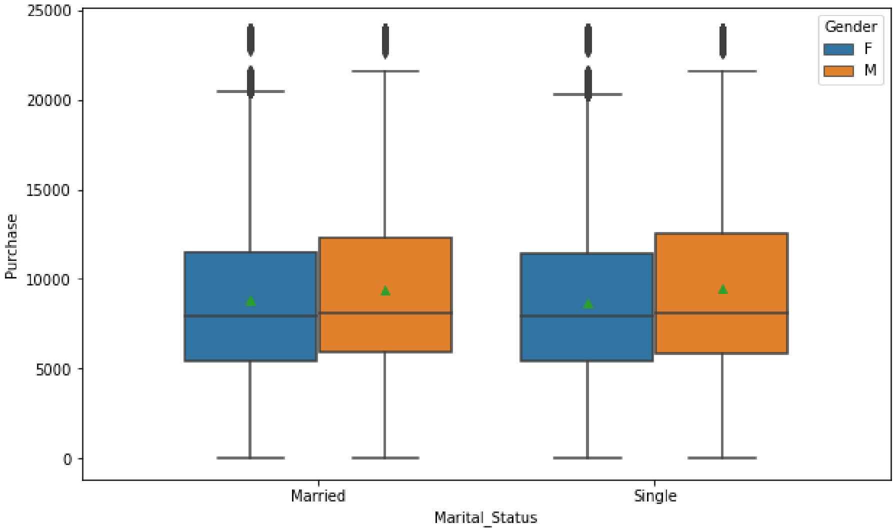
In [68]:
```
plt.figure(figsize = (15, 6))
sns.boxplot(data = df, x = 'Age', y = 'Purchase', hue = 'Gender', showmeans = True, width = 0.6)
plt.plot()
```

Out[68]: []



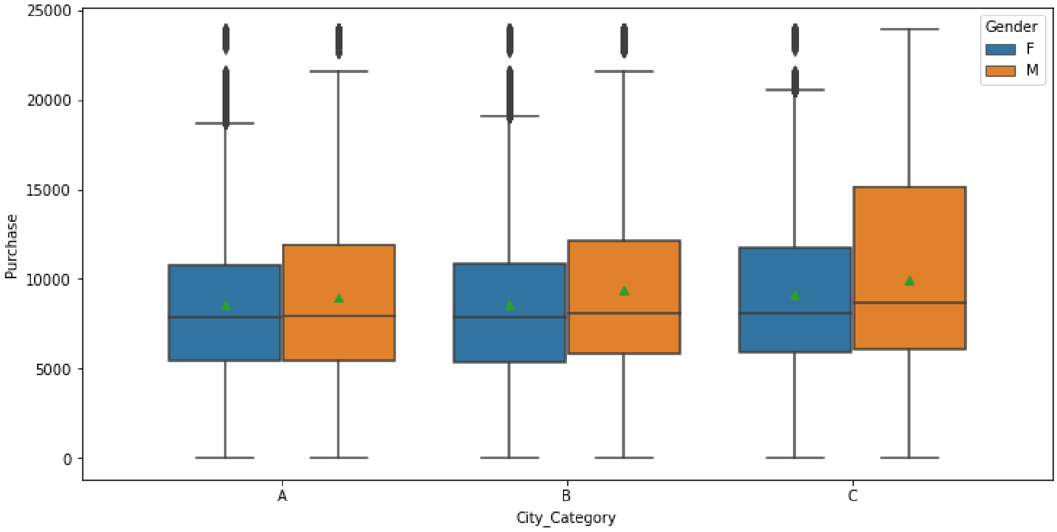
In [69]:
```python
plt.figure(figsize = (10, 6))
sns.boxplot(data = df, x = 'Marital_Status', y = 'Purchase', hue = 'Gender', showmeans = True, width  = 0.8)
plt.plot()
```

Out[69]: []



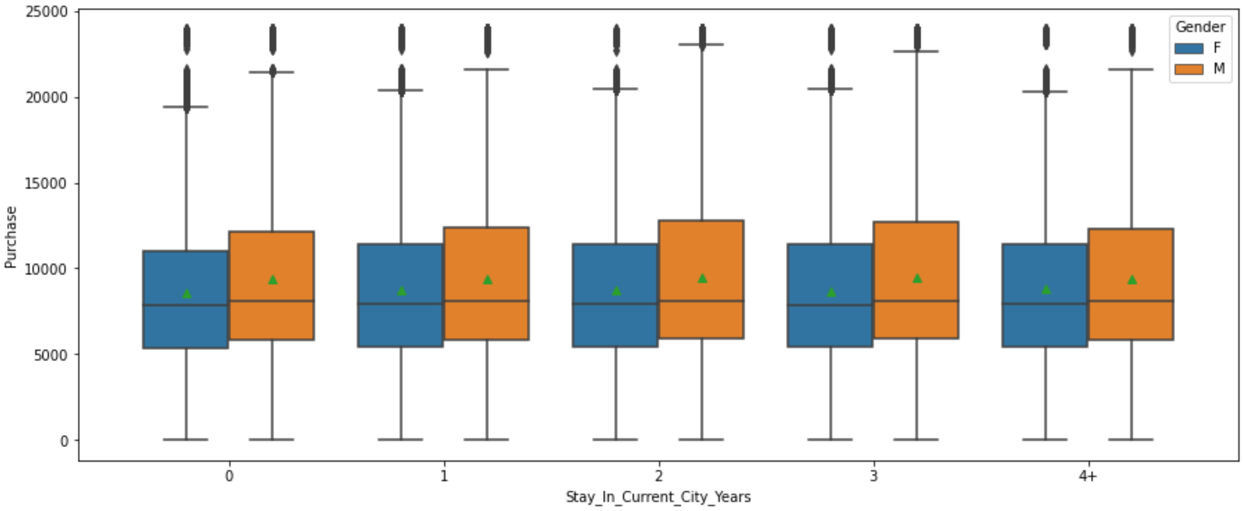
In [70]:
```python
plt.figure(figsize = (12, 6))
sns.boxplot(data = df, x = 'City_Category', y = 'Purchase', hue = 'Gender', showmeans = True)
plt.plot()
```

Out[70]: []



In [71]:
```
plt.figure(figsize = (15, 6))
sns.boxplot(data = df, x = 'Stay_In_Current_City_Years', y = 'Purchase', hue = 'Gender', showmeans = True)
plt.plot()
```

Out[71]: []



Determining the mean purchase made by each user

For Males

How the deviations vary for different sample sizes ?

In [72]:
```
df_male_customer
```

Out[72]:

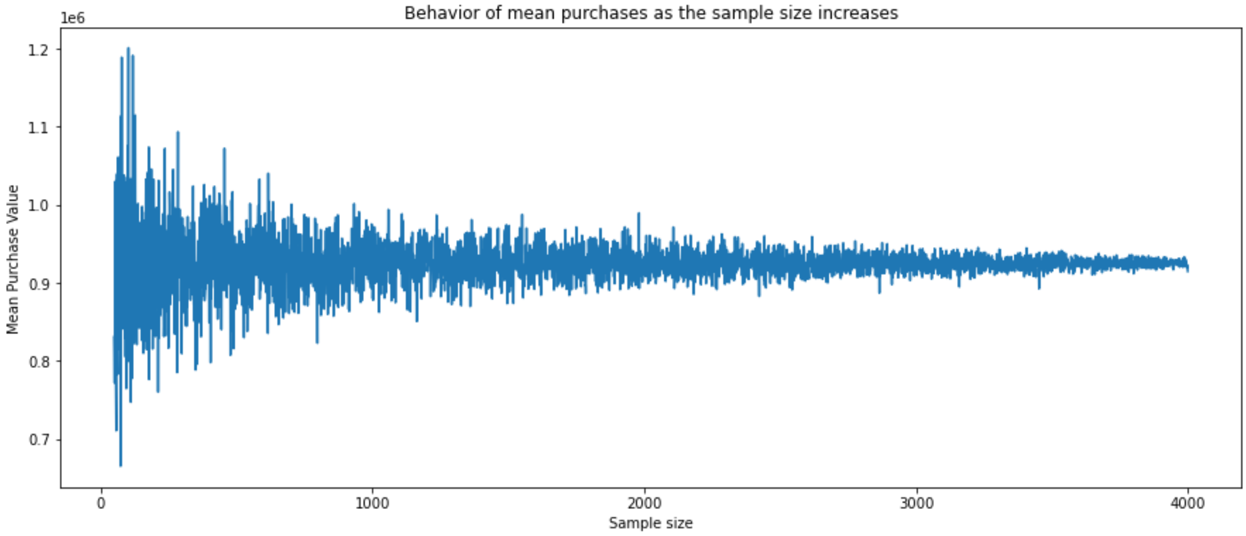
	Gender	User_ID	Total_Purchase
1666	M	1000002	810472
1667	M	1000003	341635
1668	M	1000004	206468
1669	M	1000005	821001
1670	M	1000007	234668
...	...	...	...
5886	M	1006030	737361
5887	M	1006032	517261
5888	M	1006033	501843
5889	M	1006034	197086
5890	M	1006040	1653299

4225 rows × 3 columns

In [73]:
```
# 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' column
    # of the 'df_male_customer' DataFrame and calculates the mean of the sampled values.
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [74]:
```
# 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.plot()
```

Out[74]: []



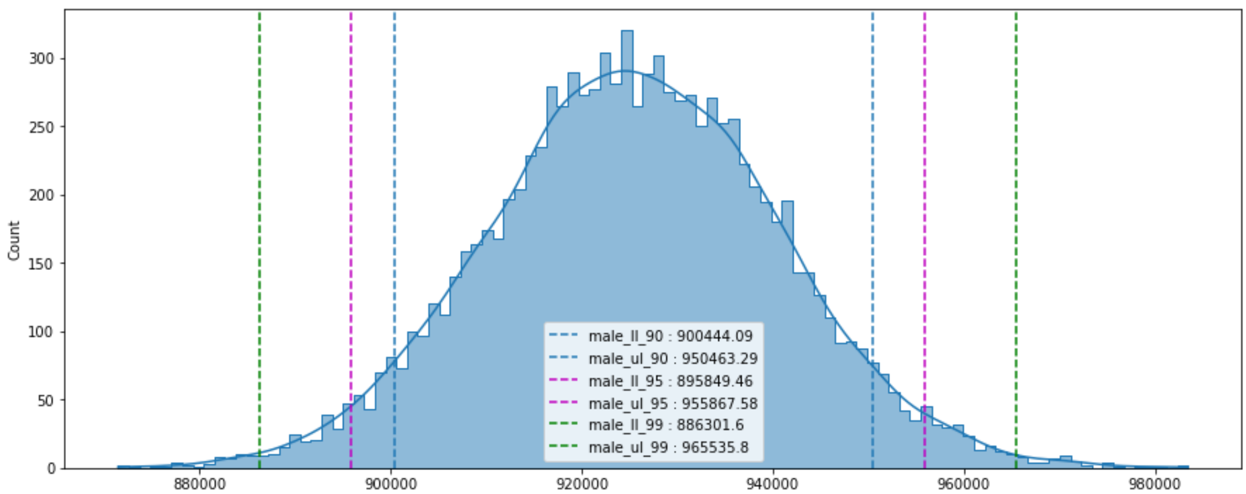

```
It can be inferred from the above plot that as the sample size is small the deviations are fairly high.
As the sample size increases, the deviation becomes smaller and smaller.
The deviations will be small if the sample size taken is greater than 2000.
```

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

In [75]:
```
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).mean()
    means_male.append(sample_mean)
```

In [76]:
```python
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% level

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 the
    # inverse of the cumulative distribution function (CDF) of a standard normal distribution

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% confidence intervals,
    # 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 = '--', color = 'm')
plt.axvline(male_ul_95, label = f'male_ul_95 : {round(male_ul_95, 2)}', linestyle = '--', color = 'm')


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 = '--', color = 'g')
plt.axvline(male_ul_99, label = f'male_ul_99 : {round(male_ul_99, 2)}', linestyle = '--', color = 'g')

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

Out[76]: []



- 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 [77]:
```python
The population mean of total spending of each male will be approximately = {np.round(np.mean(means_male), 2)} ")
```

```
The population mean of total spending of each male will be approximately = 925499.44
```

For Females

How the deviations vary for different sample sizes ?

In [78]:
```
df_female_customer
```

Out[78]:

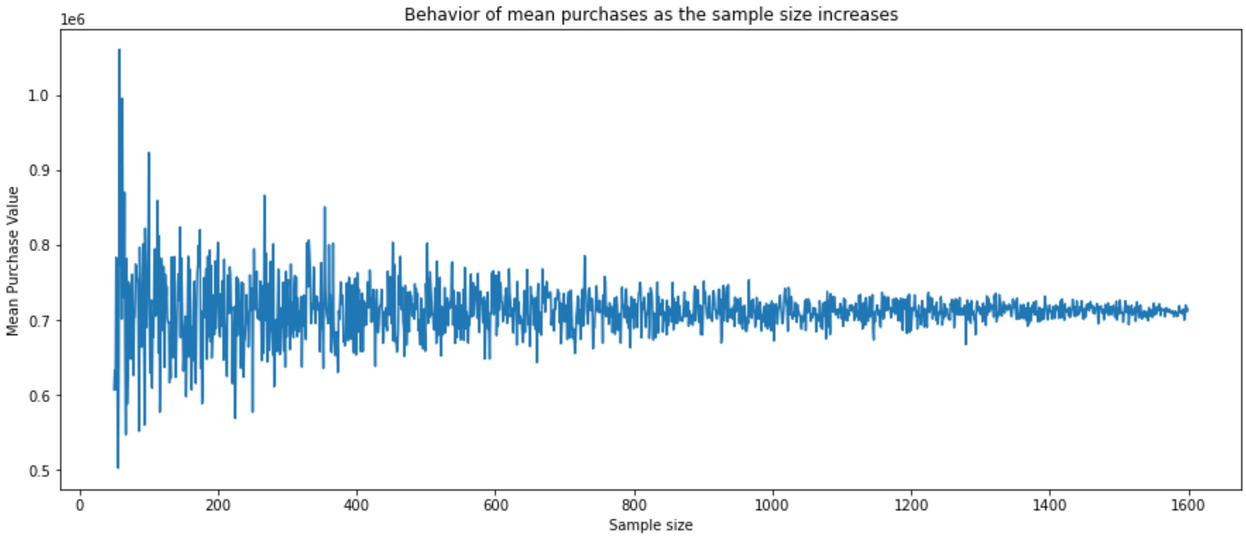
	Gender	User_ID	Total_Purchase
0	F	1000001	334093
1	F	1000006	379930
2	F	1000010	2169510
3	F	1000011	557023
4	F	1000016	150490
...	...	...	...
1661	F	1006035	956645
1662	F	1006036	4116058
1663	F	1006037	1119538
1664	F	1006038	90034
1665	F	1006039	590319

1666 rows × 3 columns

In [79]:
```
# 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' column
    # of the 'df_female_customer' DataFrame and calculates the mean of the sampled values.
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [80]:
```
# 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.plot()
```

Out[80]: []



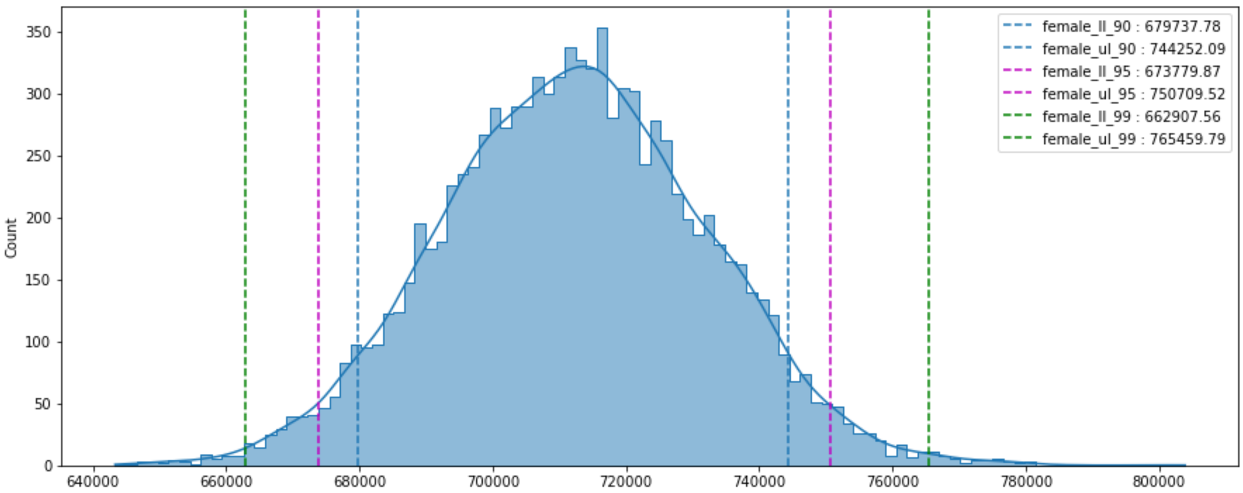

```
It can be inferred from the above plot that as the sample size is small the deviations are fairly high.
As the sample size increases, the deviation becomes smaller and smaller.
The deviations will be small if the sample size taken is greater than 1000.
```

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

In [81]:
```
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).mean()
    means_female.append(sample_mean)
```

In [82]:
```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% level

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 the
    # inverse of the cumulative distribution function (CDF) of a standard normal distribution

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)}', linestyle = '--')
    # 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)}', 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% confidence intervals,
    # 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)}', linestyle = '--', color = 'm')
plt.axvline(female_ul_95, label = f'female_ul_95 : {round(female_ul_95, 2)}', linestyle = '--', color = 'm')


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)}', linestyle = '--', color = 'g')
plt.axvline(female_ul_99, label = f'female_ul_99 : {round(female_ul_99, 2)}', linestyle = '--', color = 'g')

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

Out[82]: []



- 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.

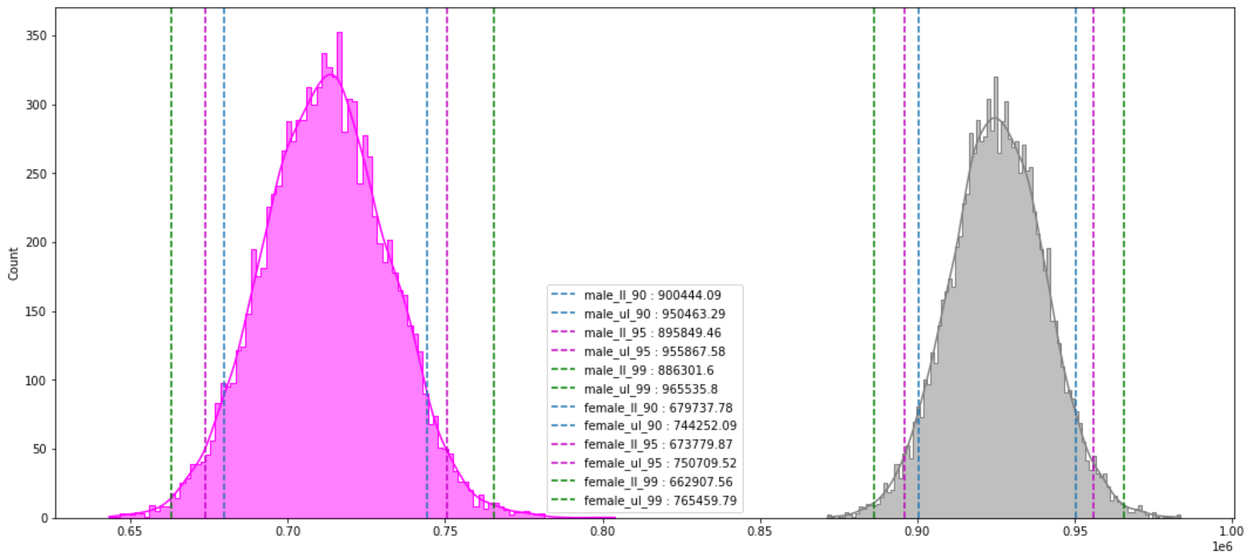
In [83]:
```
print(f"The population mean of total spending of each female will be approximately = {np.round(np.mean(means_fema
```

```
The population mean of total spending of each female will be approximately = 712205.09
```

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

In [84]:
```
# The code generates a histogram plot to visualize the distributions of means_male and means_female,
    # along with vertical lines indicating confidence interval limits at different confidence levels

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

# The first histogram represents the distribution of means_male with gray color having
    # 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 = '--', color = 'm')
plt.axvline(male_ul_95, label = f'male_ul_95 : {round(male_ul_95, 2)}', linestyle = '--', color = 'm')
plt.axvline(male_ll_99, label = f'male_ll_99 : {round(male_ll_99, 2)}', linestyle = '--', color = 'g')
plt.axvline(male_ul_99, label = f'male_ul_99 : {round(male_ul_99, 2)}', linestyle = '--', color = 'g')

# The second histogram represents the distribution of means_female with magenta color
    # 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)}', linestyle = '--')
plt.axvline(female_ul_90, label = f'female_ul_90 : {round(female_ul_90, 2)}', linestyle = '--')
plt.axvline(female_ll_95, label = f'female_ll_95 : {round(female_ll_95, 2)}', linestyle = '--', color = 'm')
plt.axvline(female_ul_95, label = f'female_ul_95 : {round(female_ul_95, 2)}', linestyle = '--', color = 'm')
plt.axvline(female_ll_99, label = f'female_ll_99 : {round(female_ll_99, 2)}', linestyle = '--', color = 'g')
plt.axvline(female_ul_99, label = f'female_ul_99 : {round(female_ul_99, 2)}', linestyle = '--', color = 'g')

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

Out[84]: []



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

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

In [85]:
```
df_single = df.loc[df['Marital_Status'] == 'Single']
df_married = df.loc[df['Marital_Status'] == 'Married']
```

In [86]:
```
df_single = df_single.groupby('User_ID')['Purchase'].sum().to_frame().reset_index().rename(columns = {'Purchase'
df_married = df_married.groupby('User_ID')['Purchase'].sum().to_frame().reset_index().rename(columns = {'Purchase
```

For Singles

In [87]:
```
df_single
```

Out[87]:

	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

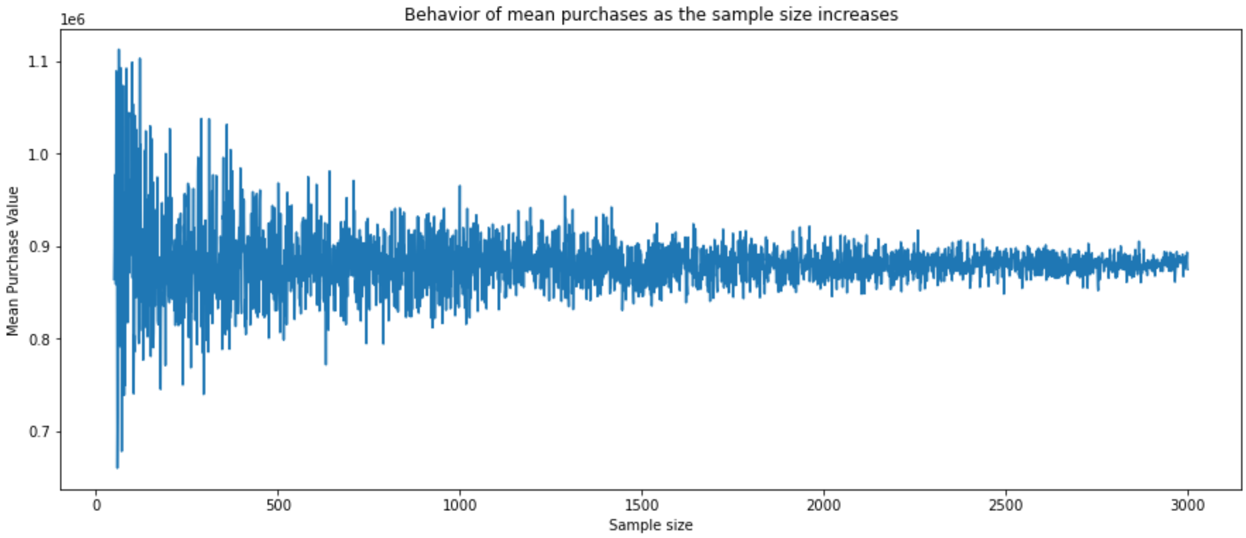
3417 rows × 2 columns

How the deviations vary for different sample sizes ?

In [88]:
```
# 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' column
    # 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 [89]:
```
# 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.plot()
```

Out[89]: []




```
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.
```

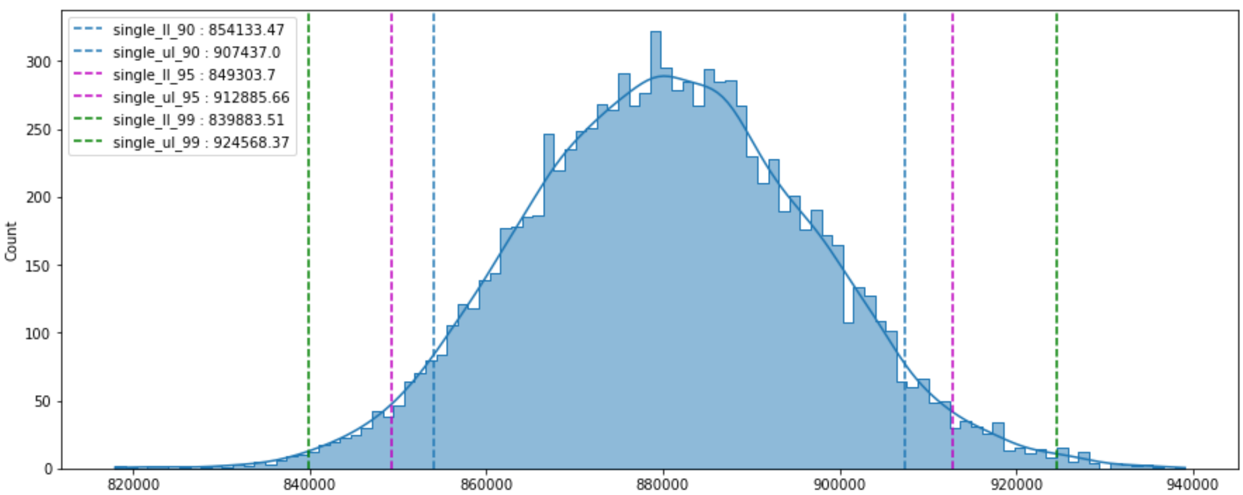
Finding the confidence interval of each single's total spending on the Black Friday

In [90]:
```
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 [91]:

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

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 the
    # inverse of the cumulative distribution function (CDF) of a standard normal distribution

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)}', linestyle = '--')
    # 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)}', 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% confidence intervals,
    # 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)}', linestyle = '--', color = 'm')
plt.axvline(single_ul_95, label = f'single_ul_95 : {round(single_ul_95, 2)}', linestyle = '--', color = 'm')


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)}', linestyle = '--', color = 'g')
plt.axvline(single_ul_99, label = f'single_ul_99 : {round(single_ul_99, 2)}', linestyle = '--', color = 'g')

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

Out[91]: []



- 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 [92]:

```python
print(f"The population mean of total spending of each single will be approximately = {np.round(np.mean(single_mea
```

```
The population mean of total spending of each single will be approximately = 880575.44
```

For Marrieds

In [93]:
```
df_married
```

Out[93]:

	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

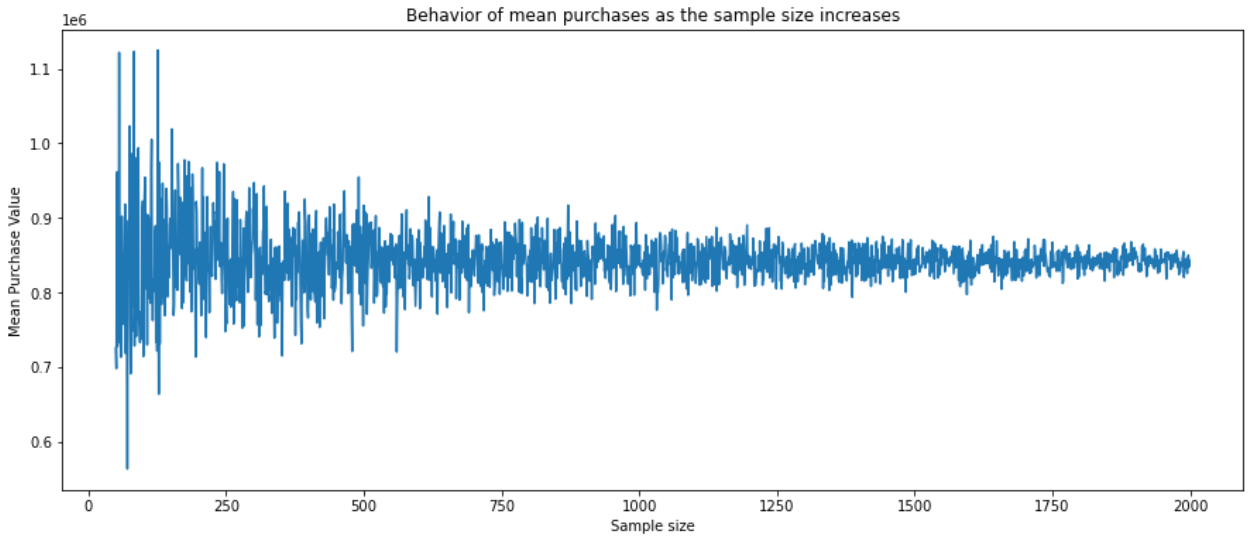
2474 rows × 2 columns

How the deviations vary for different sample sizes ?

In [94]:
```
# 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' column
    # 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 [95]:
```
# 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.plot()
```

Out[95]: []




```
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.
```

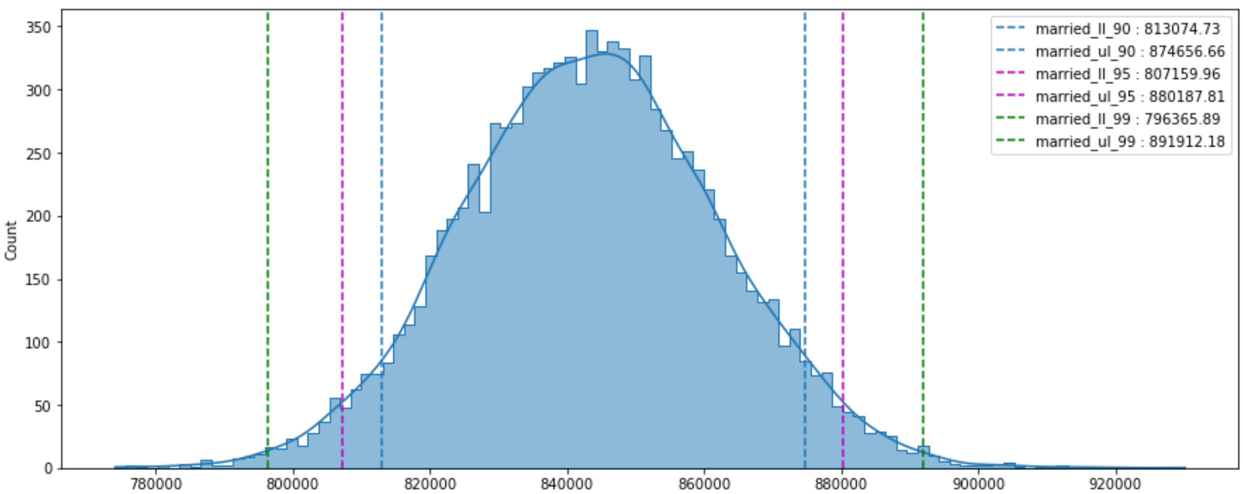
```
The deviations will be small if the sample size taken is greater than 1500.
```

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

In [96]:
```python
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 [97]:
```python
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% level

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` 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 the
    # inverse of the cumulative distribution function (CDF) of a standard normal distribution

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)}', linestyle = '--')
    # 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)}', 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% confidence intervals,
    # 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)}', linestyle = '--', color = 'm')
plt.axvline(married_ul_95, label = f'married_ul_95 : {round(married_ul_95, 2)}', linestyle = '--', color = 'm')


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)}', linestyle = '--', color = 'g')
plt.axvline(married_ul_99, label = f'married_ul_99 : {round(married_ul_99, 2)}', linestyle = '--', color = 'g')

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

Out[97]: []



- 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.

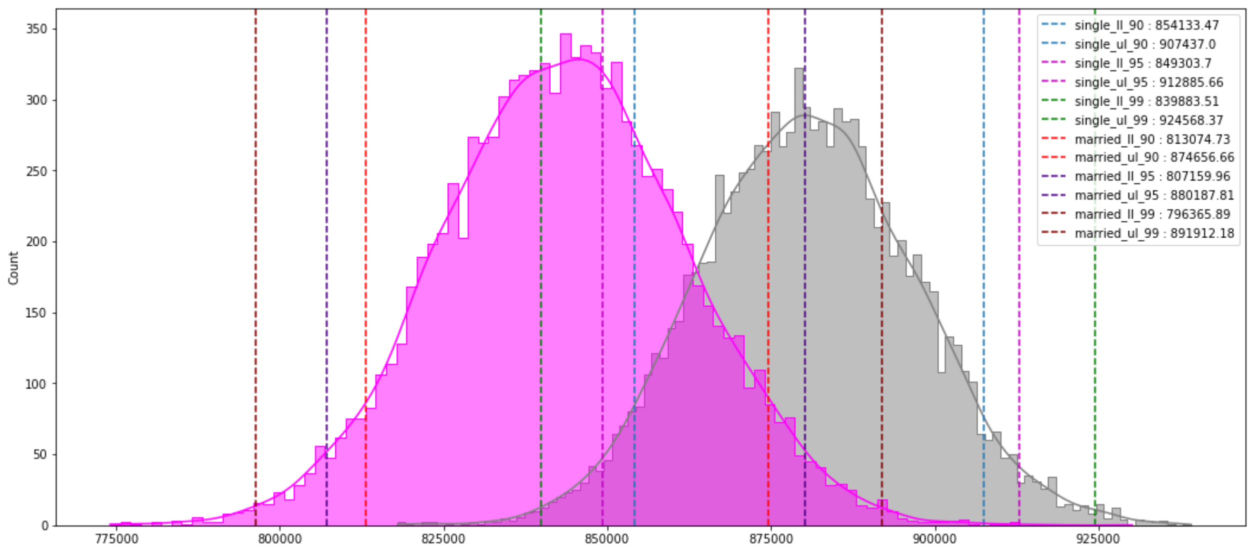
```
In [98]: print(f"The population mean of total spending of each male will be approximately = {np.round(np.mean(married_mean
```

```
The population mean of total spending of each male will be approximately = 843521.34
```

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

```
In [99]: # The code generates a histogram plot to visualize the distributions of single_means and married_means,
             # along with vertical lines indicating confidence interval limits at different confidence levels

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

         # The first histogram represents the distribution of single_means with gray color having
             # 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)}', linestyle = '--')
         plt.axvline(single_ul_90, label = f'single_ul_90 : {round(single_ul_90, 2)}', linestyle = '--')
         plt.axvline(single_ll_95, label = f'single_ll_95 : {round(single_ll_95, 2)}', linestyle = '--', color = 'm')
         plt.axvline(single_ul_95, label = f'single_ul_95 : {round(single_ul_95, 2)}', linestyle = '--', color = 'm')
         plt.axvline(single_ll_99, label = f'single_ll_99 : {round(single_ll_99, 2)}', linestyle = '--', color = 'g')
         plt.axvline(single_ul_99, label = f'single_ul_99 : {round(single_ul_99, 2)}', linestyle = '--', color = 'g')

         # The second histogram represents the distribution of married_means with magenta color
             # 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)}', linestyle = '--', color = 'r')
         plt.axvline(married_ul_90, label = f'married_ul_90 : {round(married_ul_90, 2)}', linestyle = '--', color = 'r')
         plt.axvline(married_ll_95, label = f'married_ll_95 : {round(married_ll_95, 2)}', linestyle = '--', color = 'indig
         plt.axvline(married_ul_95, label = f'married_ul_95 : {round(married_ul_95, 2)}', linestyle = '--', color = 'indig
         plt.axvline(married_ll_99, label = f'married_ll_99 : {round(married_ll_99, 2)}', linestyle = '--', color = 'maroo
         plt.axvline(married_ul_99, label = f'married_ul_99 : {round(married_ul_99, 2)}', linestyle = '--', color = 'maroo

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

```
Out[99]: []
```



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 [100]:
```
df['Age'].unique()
```

Out[100]:
```
['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25']
Categories (7, object): ['0-17', '18-25', '26-35', '36-45', '46-50', '51-55', '55+']
```

In [101]:
```
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 [102]:
```
df_age_0_to_17 = df_age_0_to_17.groupby(by = 'User_ID')['Purchase'].sum().to_frame().reset_index().rename(columns
df_age_18_to_25 = df_age_18_to_25.groupby(by = 'User_ID')['Purchase'].sum().to_frame().reset_index().rename(colum
df_age_26_to_35 = df_age_26_to_35.groupby(by = 'User_ID')['Purchase'].sum().to_frame().reset_index().rename(colum
df_age_36_to_45 = df_age_36_to_45.groupby(by = 'User_ID')['Purchase'].sum().to_frame().reset_index().rename(colum
df_age_46_to_50 = df_age_46_to_50.groupby(by = 'User_ID')['Purchase'].sum().to_frame().reset_index().rename(colum
df_age_51_to_55 = df_age_51_to_55.groupby(by = 'User_ID')['Purchase'].sum().to_frame().reset_index().rename(colum
df_age_above_55 = df_age_above_55.groupby(by = 'User_ID')['Purchase'].sum().to_frame().reset_index().rename(colum
```

For Age Group 0 - 17 years

In [103]:
```
df_age_0_to_17
```

Out[103]:

	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

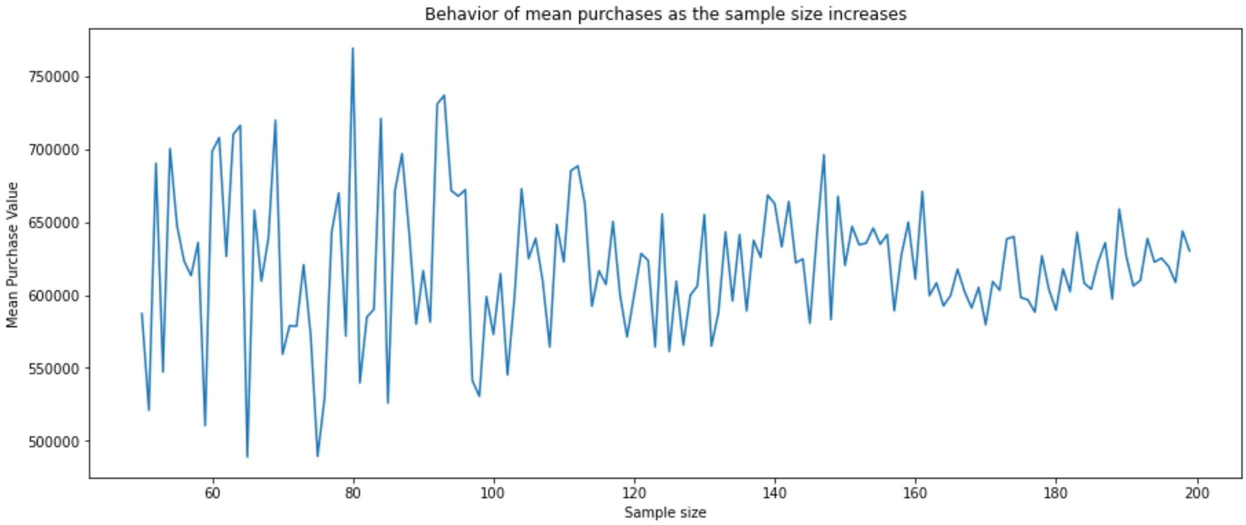
218 rows × 2 columns

How the deviations vary for different sample sizes ?

In [104]:
```python
# 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' column
    # of the 'df_age_0_to_17' DataFrame and calculates the mean of the sampled values.
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [105]:
```python
# 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.plot()
```

Out[105]: []




```
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.
```

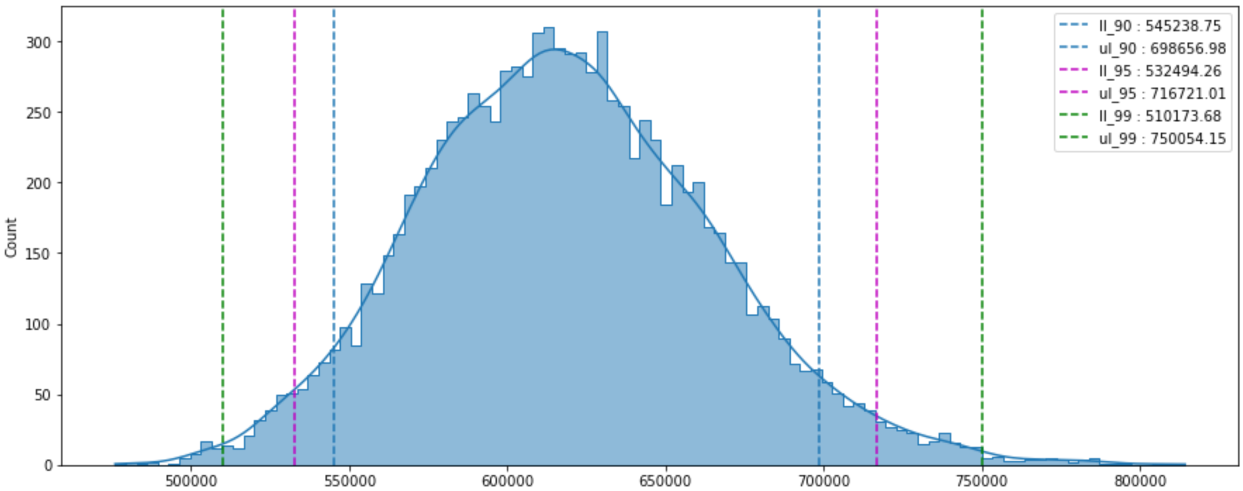
Finding the confidence interval of total spending for each individual in the age group 0 - 17 on the Black Friday

In [106]:
```python
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).mean()
    means.append(sample_mean)
```

In [107]:

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

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 the
    # inverse of the cumulative distribution function (CDF) of a standard normal distribution

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% confidence intervals,
    # 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 = 'm')
plt.axvline(ul_95, label = f'ul_95 : {round(ul_95, 2)}', linestyle = '--', color = 'm')


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 = 'g')
plt.axvline(ul_99, label = f'ul_99 : {round(ul_99, 2)}', linestyle = '--', color = 'g')

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

Out[107]: []



- 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 [108]:

```
n of total spending of each customer in age group 0 -17 will be approximately = {np.round(np.mean(means), 2)} ")
```

```
The population mean of total spending of each customer in age group 0 -17 will be approximately = 618801.27
```

For Age Group 18 - 25 years

In [109]:
```
df_age_18_to_25
```

Out[109]:

	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

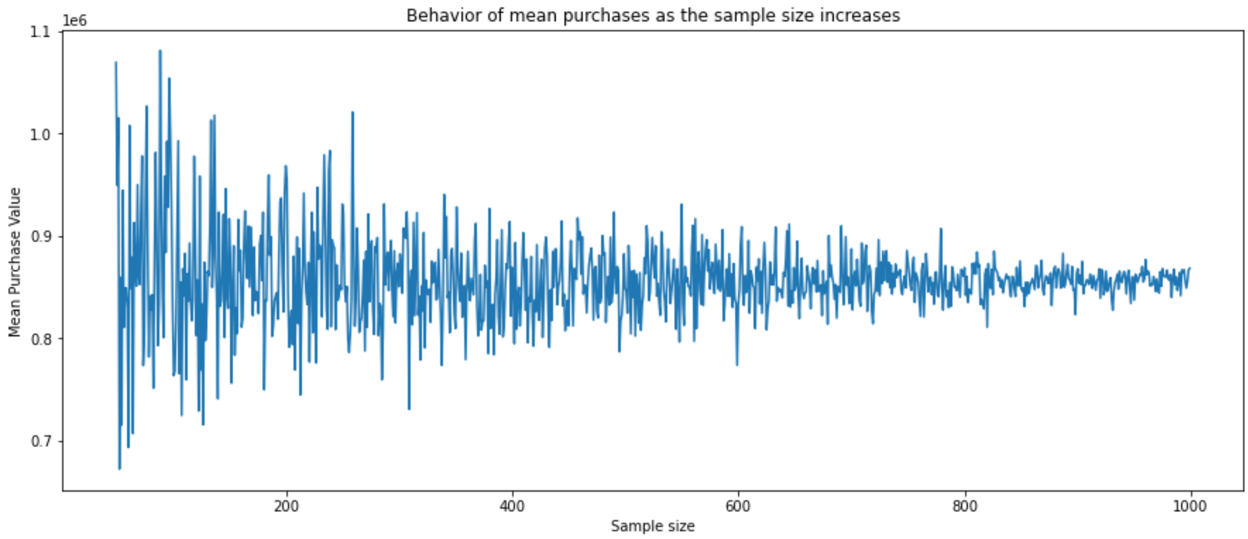
1069 rows × 2 columns

How the deviations vary for different sample sizes ?

In [110]:
```
# 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' column
    # of the 'df_age_18_to_25' DataFrame and calculates the mean of the sampled values.
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [111]:
```
# 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.plot()
```

Out[111]: []




```
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.
```

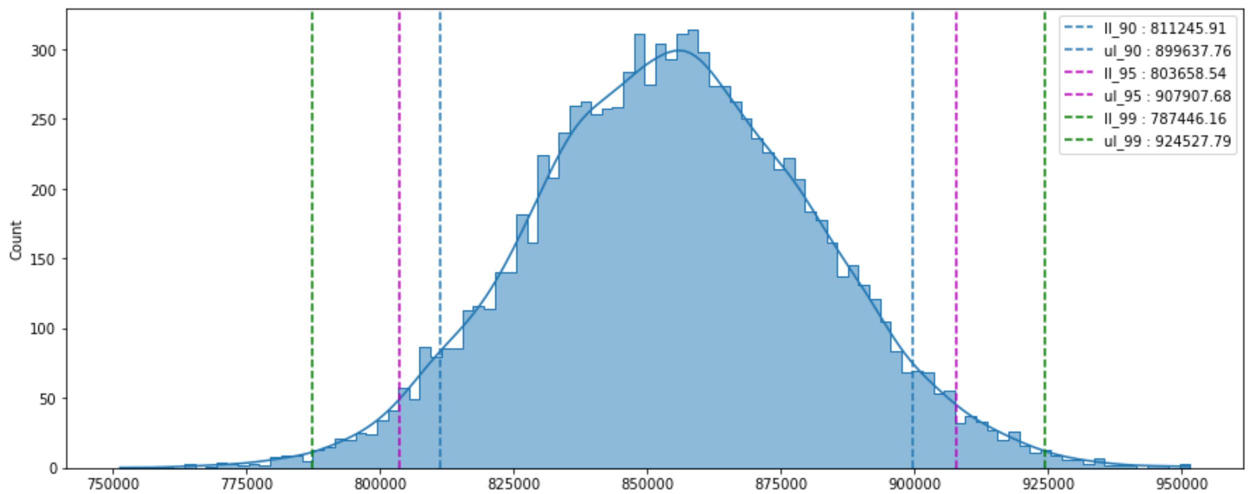
```
The deviations will be small if the sample size taken is greater than 600.
```

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

In [112]:
```
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).mean()
    means.append(sample_mean)
```

In [113]:
```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% level

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 the
    # inverse of the cumulative distribution function (CDF) of a standard normal distribution

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% confidence intervals,
    # 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 = 'm')
plt.axvline(ul_95, label = f'ul_95 : {round(ul_95, 2)}', linestyle = '--', color = 'm')


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 = 'g')
plt.axvline(ul_99, label = f'ul_99 : {round(ul_99, 2)}', linestyle = '--', color = 'g')

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

Out[113]: []



- 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 [114]:
```
print(f"The population mean of total spending of each customer in age group 18 - 25 will be approximately = {np.r
```

```
The population mean of total spending of each customer in age group 18 - 25 will be approximately = 855102.7
```

For Age Group 26 - 35 years

In [115]:
```
df_age_26_to_35
```

Out[115]:

	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

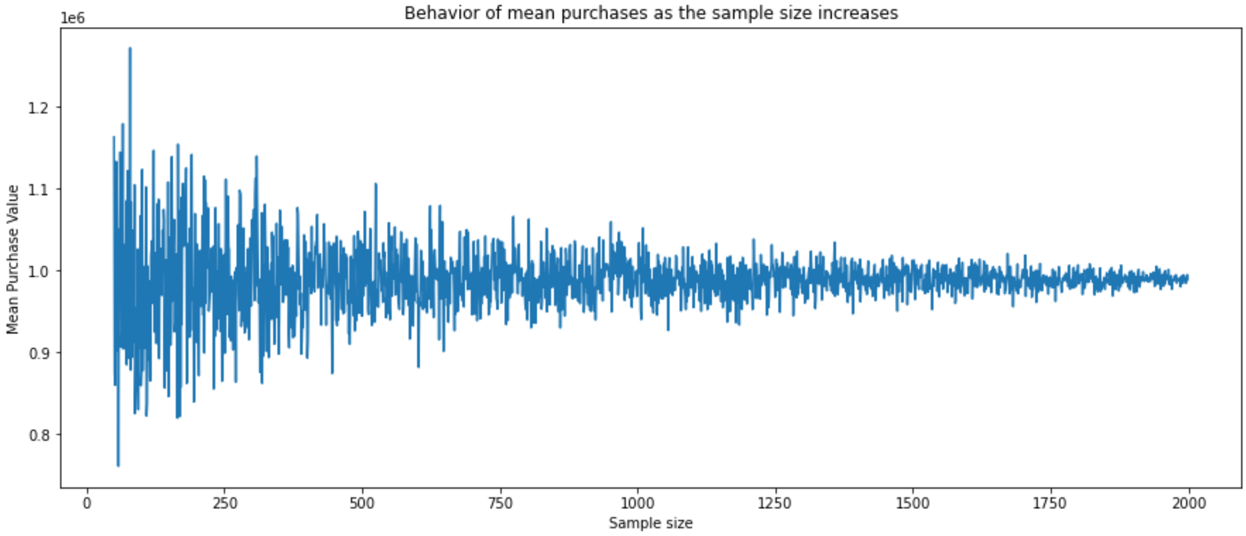
2053 rows × 2 columns

How the deviations vary for different sample sizes ?

In [116]:
```
# The code snippet performs a loop to calculate the mean purchase for different
    # sample sizes of customers with age group 26 - 35 yrs.

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' column
    # of the 'df_age_26_to_35' DataFrame and calculates the mean of the sampled values.
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [117]:
```python
# 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.plot()
```

Out[117]: []



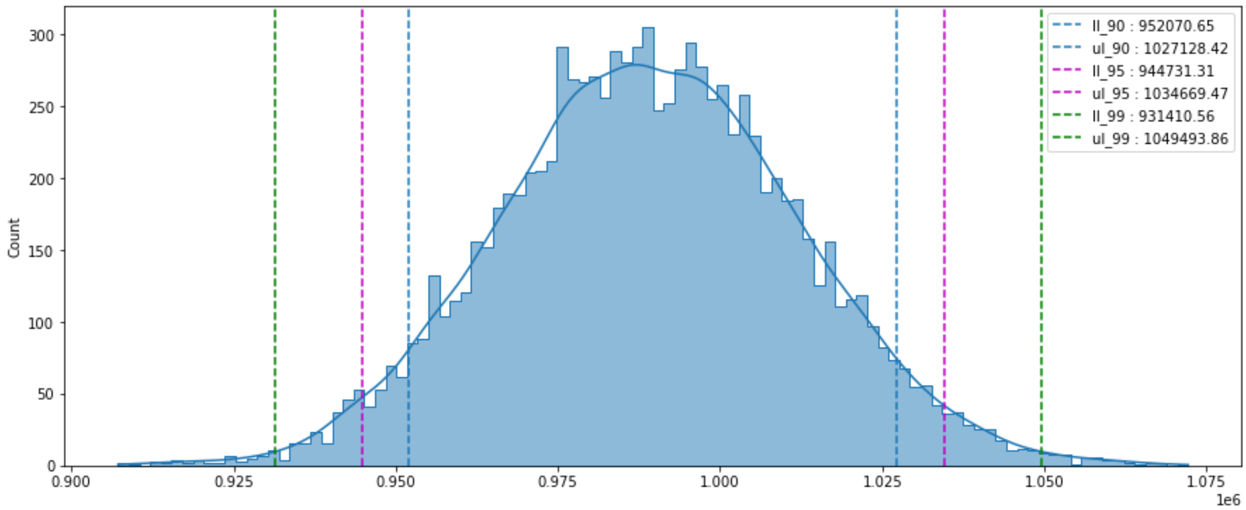

```
It can be inferred from the above plot that as the sample size is small the deviations are fairly high.
As the sample size increases, the deviation becomes smaller and smaller.
The deviations will be small if the sample size taken is greater than 1250.
```

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

In [118]:
```python
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).mean()
    means.append(sample_mean)
```

In [119]:
```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% level

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 the
    # inverse of the cumulative distribution function (CDF) of a standard normal distribution

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% confidence intervals,
    # 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 = 'm')
plt.axvline(ul_95, label = f'ul_95 : {round(ul_95, 2)}', linestyle = '--', color = 'm')


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 = 'g')
plt.axvline(ul_99, label = f'ul_99 : {round(ul_99, 2)}', linestyle = '--', color = 'g')

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

Out[119]: []



- 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 [120]:
```
print(f"The population mean of total spending of each customer in age group 26 - 35 will be approximately = {np.r
```

```
The population mean of total spending of each customer in age group 26 - 35 will be approximately = 989223.39
```

For Age Group 36 - 45 years

In [121]:
```
df_age_36_to_45
```

Out[121]:

	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

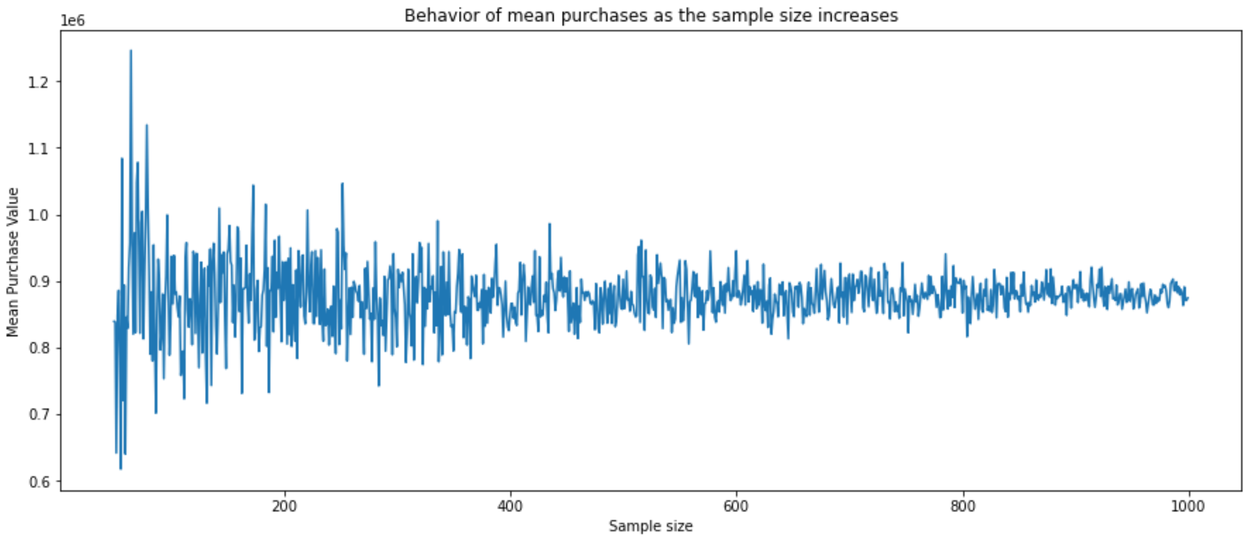
1167 rows × 2 columns

How the deviations vary for different sample sizes ?

In [122]:
```
# 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' column
    # of the 'df_age_36_to_45' DataFrame and calculates the mean of the sampled values.
    # The calculated mean values are then stored in the 'mean_purchases' list.
```

In [123]:
```
# 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.plot()
```

Out[123]: []




```
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.
```

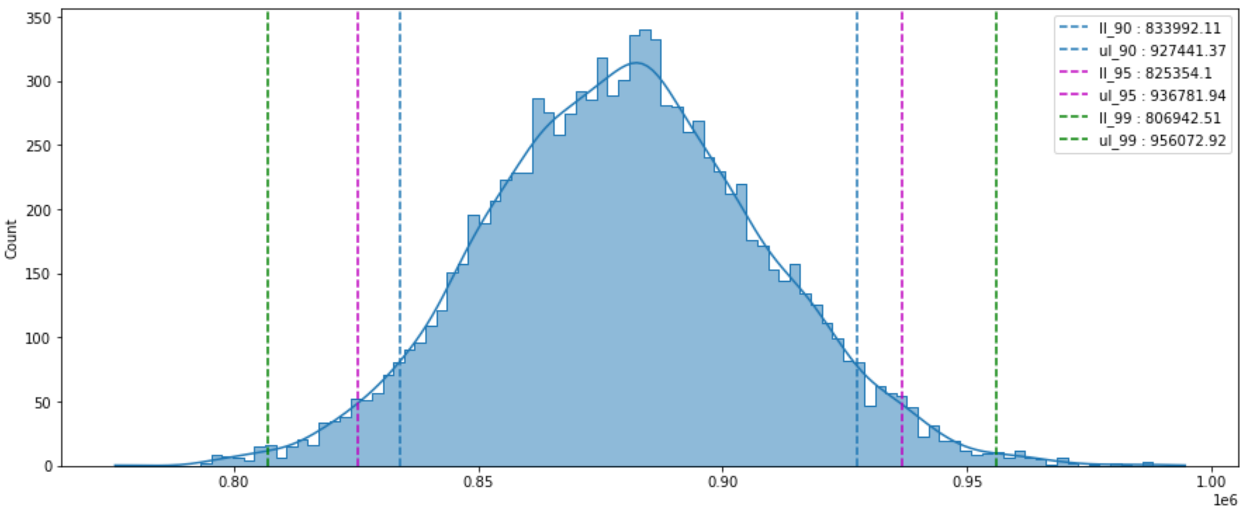
```
The deviations will be small if the sample size taken is greater than 600.
```

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

In [124]:
```python
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).mean()
    means.append(sample_mean)
```

In [125]:
```python
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% level

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 the
    # inverse of the cumulative distribution function (CDF) of a standard normal distribution

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% confidence intervals,
    # 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 = 'm')
plt.axvline(ul_95, label = f'ul_95 : {round(ul_95, 2)}', linestyle = '--', color = 'm')


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 = 'g')
plt.axvline(ul_99, label = f'ul_99 : {round(ul_99, 2)}', linestyle = '--', color = 'g')

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

Out[125]: []



- 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 [126]:
```
of total spending of each customer in age group 36 - 45 will be approximately = {np.round(np.mean(means), 2)} ")
```

```
The population mean of total spending of each customer in age group 36 - 45 will be approximately = 880002.81
```

For Age Group 46 - 50 years

In [127]:
```
df_age_46_to_50
```

Out[127]:

	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

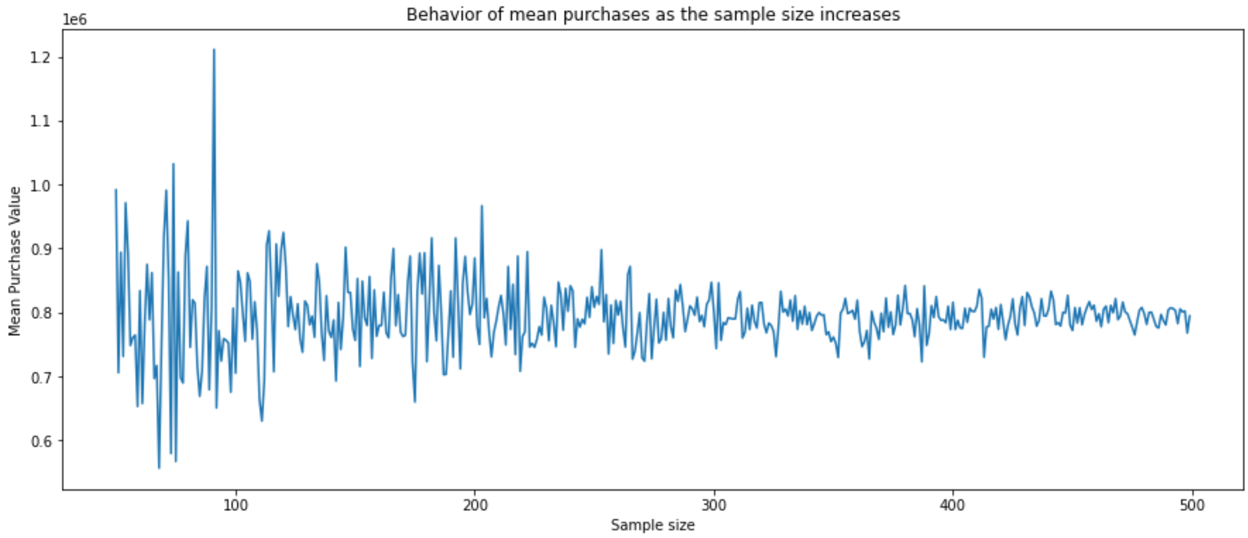
531 rows × 2 columns

How the deviations vary for different sample sizes ?

In [128]:
```
# 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' column
    # of the 'df_age_46_to_50' DataFrame and calculates the mean of the sampled values.
    # The calculated mean values are then stored in the 'mean_purchases' list
```

In [129]:
```
# 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.plot()
```

Out[129]: []



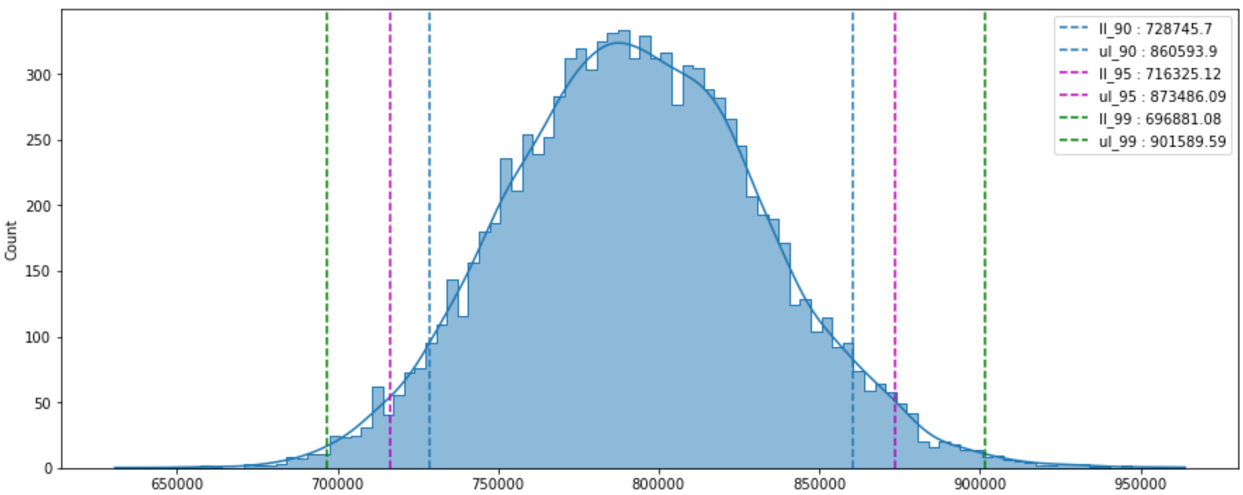

```
It can be inferred from the above plot that as the sample size is small the deviations are fairly high.
As the sample size increases, the deviation becomes smaller and smaller.
The deviations will be small if the sample size taken is greater than 300.
```

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

In [130]:
```
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).mean()
    means.append(sample_mean)
```

In [131]:
```
# The below code generates a histogram plot with kernel density estimation and
    # adds vertical lines to represent confidence intervals at 90%, 95%, and 99% level

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 the
    # inverse of the cumulative distribution function (CDF) of a standard normal distribution

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% confidence intervals,
    # 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 = 'm')
plt.axvline(ul_95, label = f'ul_95 : {round(ul_95, 2)}', linestyle = '--', color = 'm')


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 = 'g')
plt.axvline(ul_99, label = f'ul_99 : {round(ul_99, 2)}', linestyle = '--', color = 'g')

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

Out[131]: []



- 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 [132]:
```
print(f"The population mean of total spending of each customer in age group 46 - 50 will be approximately = {np.r
```

```
The population mean of total spending of each customer in age group 46 - 50 will be approximately = 793101.72
```

Actionable insights

- Out of every four transactions made on Black Friday in the Walmart stores, three are made by the males and one is made by the females.

- 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.)
- 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.
- 82.43% of the total transactions are made for only 5 Product Categories. These are, 5, 1, 8, 11 and 2.
- There are 1666 unique female customers and 4225 unique male customers. Average number of transactions made by each Male on Black Friday is 98 while for Female it is 82.
- On an average each male makes a total purchase of 925438.92 on Black Friday while for each female the figure is 712269.56.
- 76.72 % of the total revenue is generated from males.
- Out of 5891 unique customers, 42 % of them are Married and 58 % of them are Single.
- Average number of transactions made by each user with marital status Married is 91 and for Single it is 95.
- On an 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 customers who are Single.
- Majority of the transactions are made by the customers whose age is between 26 and 45 years.
- About 81.82% of the total transactions are made by customers of age between 18 and 50 years.
- 81.82 % of total unique customers have age between 18 and 50 years.
- Out of all unique customers, 35.85 % belong to the age group of 26 - 35 years, 19.81 % belong to the age group of 36 - 45 years, 18.15 % belong to the age group of 18 - 25 years, 9.01 % belong to the age group of 46 - 50 years.
- Walmart generated 86.21 % of total revenue from customers in range 18 to 50 years on Black Friday.
- 39.87 % of the total revenue is generated from the customers having age group of 26 - 35 years, 20.15 % is generated from 36 - 45 years, 17.93 % from 18 - 25 years, 8.26 % from 46 - 50 years.
- Majority of the total unique customers belong to the city C. 82.26 % of the total unique customers belong to city C and B.
- Walmart generated 41.52 % of the total revenue from the customers belonging to the city B, 32.65 % from city C and 25.83 % from city A on Black Friday.
- Top 5 product categories from which Walmart made 84.36 % of total revenue on Black Friday are 1, 5, 8, 6 and 2.
- 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 age group 0 -17 will be approximately = 617797.25
- The population mean of total spending of each customer in age group 18 - 25 will be approximately = 854676.31
- The population mean of total spending of each customer in age group 26 - 35 will be approximately = 989120.36
- The population mean of total spending of each customer in age group 36 - 45 will be approximately = 879434.88
- The population mean of total spending of each customer in age group 46 - 50 will be approximately = 792671.74

Recommendations

- **Targeted marketing**: Since the majority of transactions are made by males, it would be beneficial to tailor marketing strategies to cater to their preferences and needs. This could include specific promotions, product offerings, or advertising campaigns designed to attract male customers.
- **Focus on popular occupations**: 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.
- **Engage with new residents**: 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.
- **Emphasize popular product categories**: Since 82.43% of transactions are concentrated in just five product categories, allocating resources and promotions towards these categories can maximize sales potential. Highlighting these popular categories and offering attractive deals can encourage more purchases.
- **Increase focus on single customers**: Given that 59.05% of total revenue is generated by single customers, dedicating efforts to cater to their needs and preferences can help drive more sales. Understanding their motivations and targeting them with personalized offers can enhance their shopping experience and loyalty.
- **Optimize revenue from specific age groups**: Since a majority of transactions are made by customers between the ages of 26 and 45, it is important to focus marketing efforts on this demographic. Offering products and services that align with their interests and values can maximize revenue generation.
- **Location-based marketing**: With a significant number of customers belonging to specific cities, tailoring marketing strategies to target these locations can lead to better results. Allocating resources, promotions, and events based on the customer concentration in each city can help drive sales.
- **Emphasize top-selling product categories**: The top five product categories generate a substantial portion of total revenue. Investing in these categories, ensuring a wide range of options and competitive pricing, can capitalize on customer demand and drive overall sales.
- **Personalized offers for high spenders**: Identifying customers with high total spending, such as males or customers in specific age groups, allows for targeted marketing and personalized offers. Providing exclusive discounts, loyalty rewards, or special privileges to these customers can encourage repeat purchases and increase customer satisfaction.
- **Implement loyalty program**: Implementating a loyalty program that offers incentives, rewards, and exclusive deals to encourage repeat purchases and increase customer retention. Targeted loyalty programs can be designed for male customers, single customers, and customers in specific age groups.
- **Enhance product offerings**: Analyze the popular product categories and identify opportunities to expand the product range within those categories. This can attract more customers and increase sales. Additionally, identify complementary products or cross-selling opportunities to encourage customers to make additional purchases.

- **Customer engagement**: Implement targeted marketing campaigns and communication strategies to engage customers regularly. This can include personalized email campaigns, social media engagement, and special promotions tailored to different customer segments. Keeping customers informed about new products, offers, and events can increase their engagement and encourage them to make more purchases.
- **Collaborations and partnerships**: Explore collaborations with popular brands or influencers that resonate with the target customer segments. These collaborations can help attract new customers, create buzz, and increase brand visibility. It can also provide opportunities for joint promotions or exclusive offers.
- **Seasonal and event-based promotions**: Leverage seasonal events, holidays, and special occasions to offer targeted promotions and discounts. Aligning marketing campaigns and product offerings with these events can create a sense of urgency and drive sales.
- **Customer feedback and reviews**: Actively seek feedback from customers to understand their preferences, pain points, and suggestions for improvement. Encourage customers to leave reviews and ratings to build social proof and credibility. Utilize this feedback to make necessary improvements and refine the customer experience.
- **Personalization and customization**: Invest in technology and data analytics to provide personalized recommendations, product suggestions, and customized offers based on individual customer preferences and past purchase history. This level of personalization can enhance the customer experience and increase conversion rates.
- **Competitive pricing and promotions**: Continuously monitor competitors' pricing and promotional activities to ensure competitiveness. Offer price-match guarantees or price comparison tools to instill confidence in customers that they are getting the best value for their purchases

In [ ]: