Business Case: Walmart - Confidence Interval and CLT

Submitted by: Archana Bharti

Problem Statement:

The Management team at Walmart Inc. wants to analyze the customer purchase behavior (specifically, purchase amount) against the customer's gender and the various other factors to help the business make better decisions. They want to understand if the spending habits differ between male and female customers: Do women spend more on Black Friday than men? (Assume 50 million customers are male and 50 million are female).

In [80]:
```
#Importing packages
import numpy as np
import pandas as pd

# Importing matplotlib and seaborn for graphs
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='whitegrid')

import warnings
warnings.filterwarnings('ignore')

from scipy import stats
from scipy.stats import kstest
import statsmodels.api as sm

# Importing Date & Time util modules
from dateutil.parser import parse

import statistics
from scipy.stats import norm
```

In [13]:
```
#Reading input file

df = pd.read_csv('D:\\Scaler\\Scaler\\Probability & Stats\\Business Case\\walmart_
```

In [3]:
```
df
```

Out[3]:

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

550068 rows × 10 columns

In [38]:
```
df.columns
```
Out[38]:
```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category',
       'Purchase'],
      dtype='object')
```

In [39]:
```
df.duplicated().sum()
```
Out[39]:
```
0
```

1. Analyzing basic metrics

1.1 Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required), statistical summary

In [7]:
```
df.shape
```
Out[7]:
```
(550068, 10)
```

In [9]:
```
df.dtypes
```

Out[9]:
```
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 [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 10 columns):
 #   Column                      Non-Null Count   Dtype 
---  ------                      --------------   ----- 
 0   User_ID                     550068 non-null  int64 
 1   Product_ID                  550068 non-null  object
 2   Gender                      550068 non-null  object
 3   Age                         550068 non-null  object
 4   Occupation                  550068 non-null  int64 
 5   City_Category               550068 non-null  object
 6   Stay_In_Current_City_Years  550068 non-null  object
 7   Marital_Status              550068 non-null  int64 
 8   Product_Category            550068 non-null  int64 
 9   Purchase                    550068 non-null  int64 
dtypes: int64(5), object(5)
memory usage: 42.0+ MB
```

In [14]:
```
# Converting gender, age, city_category, stay_in_current_city_years and marital_st
```

In [45]:
```
obj_to_cat = ['Gender','Age','City_Category','Stay_In_Current_City_Years','Marital_
for i in obj_to_cat:
    df[i] = df[i].astype('category')
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  category
 3   Age                         550068 non-null  category
 4   Occupation                  550068 non-null  int64   
 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  int64   
 9   Purchase                    550068 non-null  int64   
dtypes: category(5), int64(4), object(1)
memory usage: 23.6+ MB
```

In [48]:
```
cols = ['User_ID','Product_ID']
for col_name in cols:
    df[col_name] = df[col_name].astype("category")

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  category
 1   Product_ID                  550068 non-null  category
 2   Gender                      550068 non-null  category
 3   Age                         550068 non-null  category
 4   Occupation                  550068 non-null  int64   
 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  int64   
 9   Purchase                    550068 non-null  int64   
dtypes: category(7), int64(3)
memory usage: 17.6 MB
```

In [50]:

```
#Statistical Summary

df.describe()
```

Out[50]:

	Occupation	Product_Category	Purchase
count	550068.000000	550068.000000	550068.000000
mean	8.076707	5.404270	9263.968713
std	6.522660	3.936211	5023.065394
min	0.000000	1.000000	12.000000
25%	2.000000	1.000000	5823.000000
50%	7.000000	5.000000	8047.000000
75%	14.000000	8.000000	12054.000000
max	20.000000	20.000000	23961.000000

In [49]:

```
df.describe(include=['object','category']).T
```

Out[49]:

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

1.1 Observations

1. There are 5,50,068 rows and 10 columns in the data.
2. There are no null values.
3. Range of purchase amount is 12 dollars to 23961 dollars.
4. Mean purchase amount is 9264 dollars.

5. Median purchase amount is 8047 dollars.
6. Standard deviation of purchase amount is 5023 dollars.
7. Inter quartile range of purchase amount is 5823 to 12054 dollars.

1.2 Non-Graphical Analysis: Value counts and unique attributes

In [17]:
```
# Unique Atrributes

df.nunique()
```

Out[17]:
```
User_ID                        5891
Product_ID                     3631
Gender                            2
Age                               7
Occupation                       21
City_Category                     3
Stay_In_Current_City_Years        5
Marital_Status                    2
Product_Category                 20
Purchase                      18105
dtype: int64
```

In [19]:
```
# Value_counts for Gender, Age, Occupation, City_Category, Stay_In_Current_City_Yea

Categorical_Columns = ['Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Cu
                       'Marital_Status', 'Product_Category']
df[Categorical_Columns].melt().groupby(['variable', 'value'])[['value']].count()/le
```

Out[19]:

variable	value	value
Age	0-17	0.027455
	18-25	0.181178
	26-35	0.399200
	36-45	0.199999
	46-50	0.083082
	51-55	0.069993
	55+	0.039093
City_Category	A	0.268549
	B	0.420263
	C	0.311189
Gender	F	0.246895
	M	0.753105
Marital_Status	0	0.590347
	1	0.409653
Occupation	0	0.126599
	1	0.086218
	2	0.048336
	3	0.032087
	4	0.131453
	5	0.022137
	6	0.037005
	7	0.107501
	8	0.002811
	9	0.011437
	10	0.023506
	11	0.021063
	12	0.056682
	13	0.014049
	14	0.049647
	15	0.022115
	16	0.046123
	17	0.072796
	18	0.012039
	19	0.015382
	20	0.061014

variable	value	value
Product_Category	**1**	0.255201
	2	0.043384
	3	0.036746
	4	0.021366
	5	0.274390
	6	0.037206
	7	0.006765
	8	0.207111
	9	0.000745
	10	0.009317
	11	0.044153
	12	0.007175
	13	0.010088
	14	0.002769
	15	0.011435
	16	0.017867
	17	0.001051
	18	0.005681
	19	0.002914
	20	0.004636
Stay_In_Current_City_Years	**0**	0.135252
	1	0.352358
	2	0.185137
	3	0.173224
	4+	0.154028

1.2 Observations:

1. ~ 80% of the users are between the age 18-50 (40%: 26-35, 18%: 18-25, 20%: 36-45) 1.1 People in age group 26–35 make more purchases than any other age group.
2. 75% of the users are Male and 25% are Female
3. 60% Single, 40% Married 3.1 Unmarried people make more purchases than married people
4. 35% Staying in the city from 1 year, 18% from 2 years, 17% from 3 years
5. People of city category B make more purchases than other city categories
6. Total of 20 product categories are there 6.1 Product categories 5, 1 and 8 sell more than other categories 6.2 Product categories 17 and 9 sell the least
7. There are 20 diffrenent types of occupations in the city

Observations:

1. Mostly features are categorical and not much correlation can be observed from above graphs
2. There’s a weak negative correlation between product category and purchase amount.

**

2.Missing Value & Outlier Detection

In [55]:
```
# Finding outliers using IQR method
for i in ['Purchase']:
    outliers = []
    p25 = np.percentile(df[i], 25)
    p75 = np.percentile(df[i], 75)
    iqr = p75 - p25
    max_cut = p75 + iqr*1.5
    min_cut = max(0, p25 - iqr*1.5)
    outliers = df.loc[(df[i]<min_cut) | (df[i]>max_cut),i]
    print('Outliers for the column',i,'-')
    print(outliers)
    print('Number of outliers-', len(outliers))
    print('Percentage of outliers =', round((len(outliers)/len(df[i]))*100,2),'%')
```

```
Outliers for the column Purchase -
343       23603
375       23792
652       23233
736       23595
1041      23341
          ...
544488    23753
544704    23724
544743    23529
545663    23663
545787    23496
Name: Purchase, Length: 2677, dtype: int64
Number of outliers- 2677
Percentage of outliers = 0.49 %
```

In [56]:
```
# Checking for missing values
df.isna().sum()
```

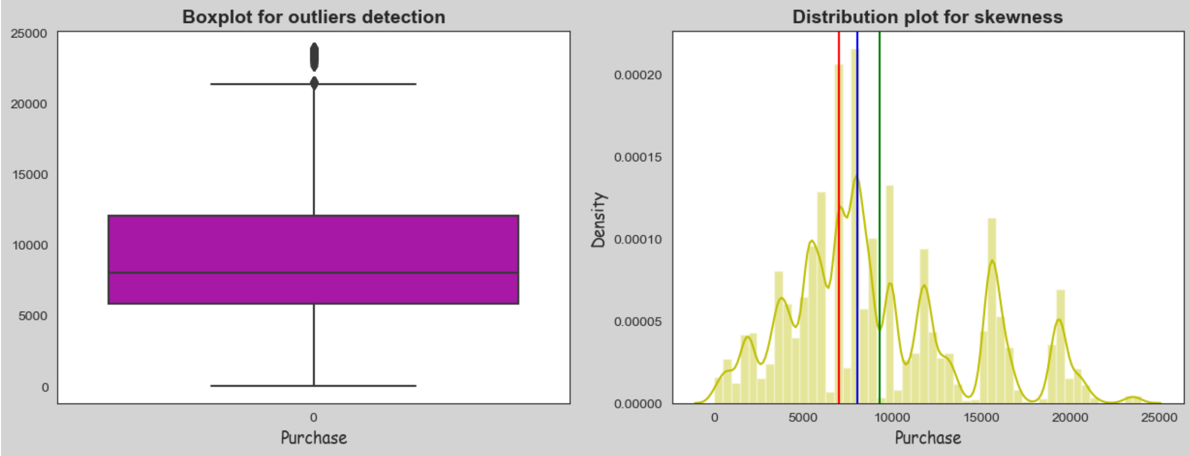
Out[56]:
```
User_ID                       0
Product_ID                    0
Gender                        0
Age                           0
Occupation                    0
City_Category                 0
Stay_In_Current_City_Years    0
Marital_Status                0
Product_Category              0
Purchase                      0
dtype: int64
```

Observations:

1. Purchase columns contains 2677 outliers. This is 0.49% of total number of entries.
2. There are no missing values in any column.

In [63]:
```
# Visualizing our dependent variable for Outliers and Skewness
fig = plt.figure(figsize=(15,5))
fig.set_facecolor("lightgrey")

plt.subplot(1,2,1)
sns.boxplot(df["Purchase"],color='m')
plt.title("Boxplot for outliers detection", fontweight="bold",fontsize=14)
plt.xlabel('Purchase', fontsize=12,family = "Comic Sans MS")

plt.subplot(1,2,2)
sns.distplot(df["Purchase"],color='y')

plt.title("Distribution plot for skewness", fontweight="bold",fontsize=14)
plt.ylabel('Density', fontsize=12,family = "Comic Sans MS")
plt.xlabel('Purchase', fontsize=12,family = "Comic Sans MS")
plt.axvline(df["Purchase"].mean(),color="g")
plt.axvline(df["Purchase"].median(),color="b")
plt.axvline(df["Purchase"].mode()[0],color="r")

plt.show()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_30772\2356310139.py:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Purchase"],color='y')
```



Observations:

1. 'Purchase' feature has outliers

Above graphs, it looks like "right-skewed distribution" which means the mass of the distribution is concentrated on the left of the figure.

- Majority of Customers purchase within the 5,000 - 20,000 range.

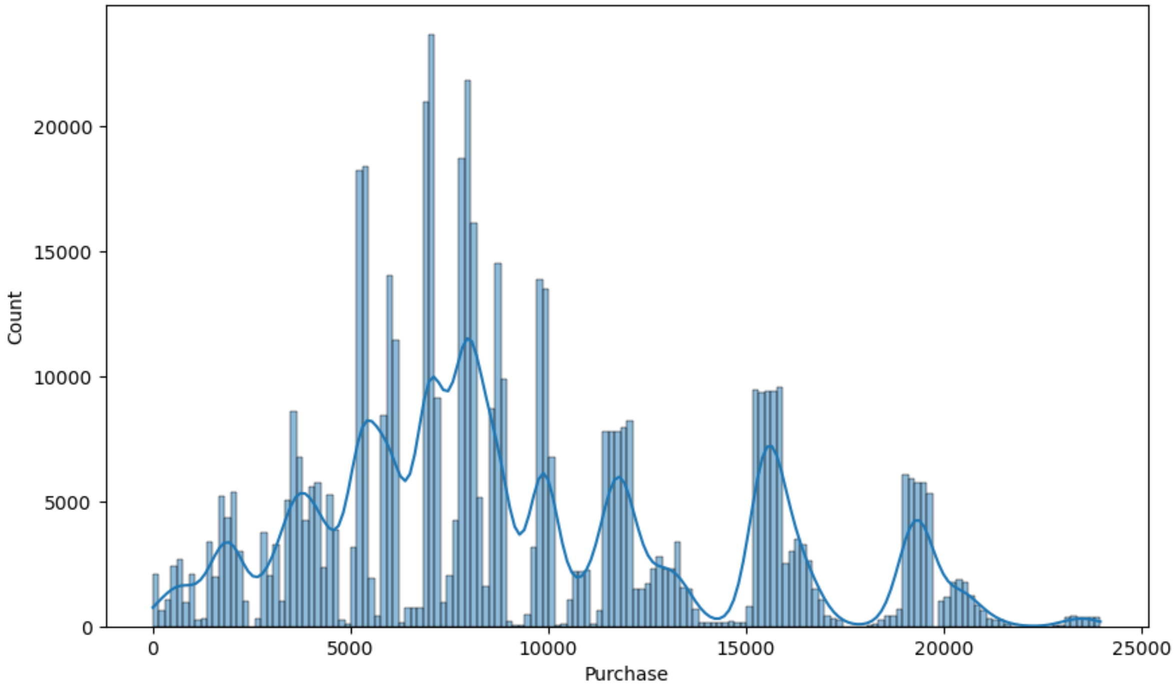
1.3 Visual Analysis - Univariate & Bivariate

```
  a) For continuous variable(s): Distplot, countplot, histogram
for univariate analysis
  b) For categorical variable(s): Boxplot
  c) For correlation: Heatmaps, Pairplots
```

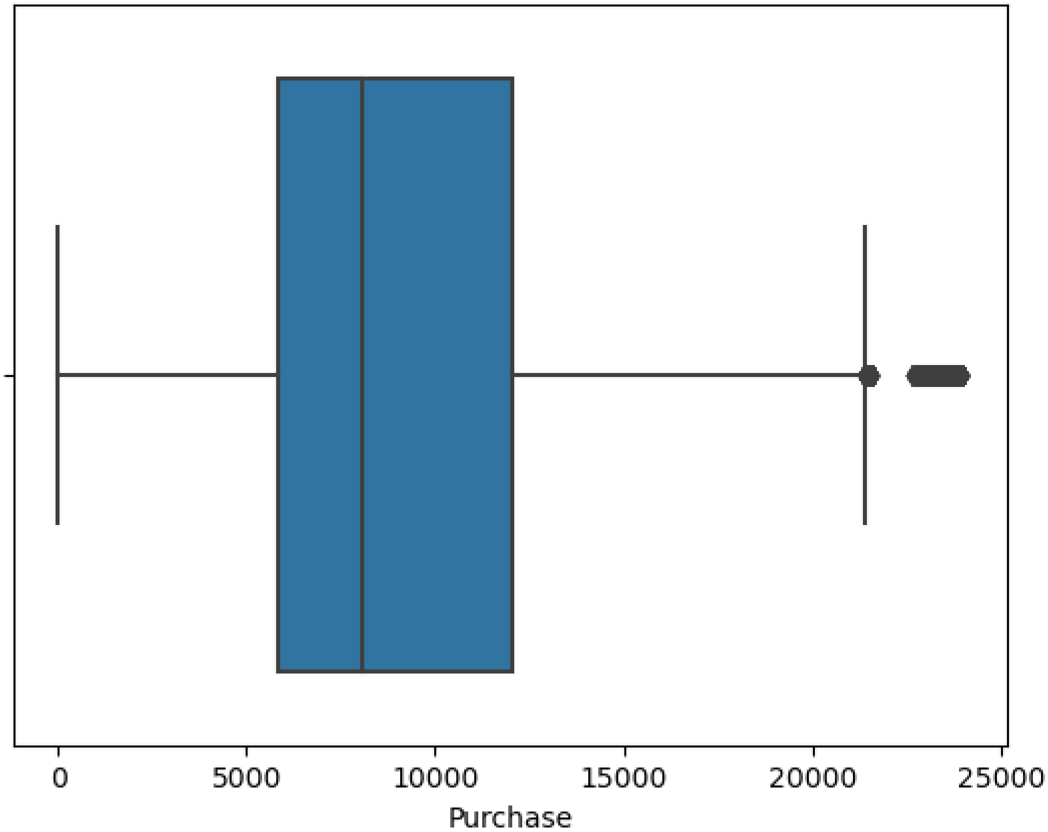
a) For continuous variable(s): Distplot, countplot, histogram for univariate analysis

Understanding the distribution of data and detecting outlies for continuous variables

In [23]:
```
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Purchase', kde=True)
plt.show()
```



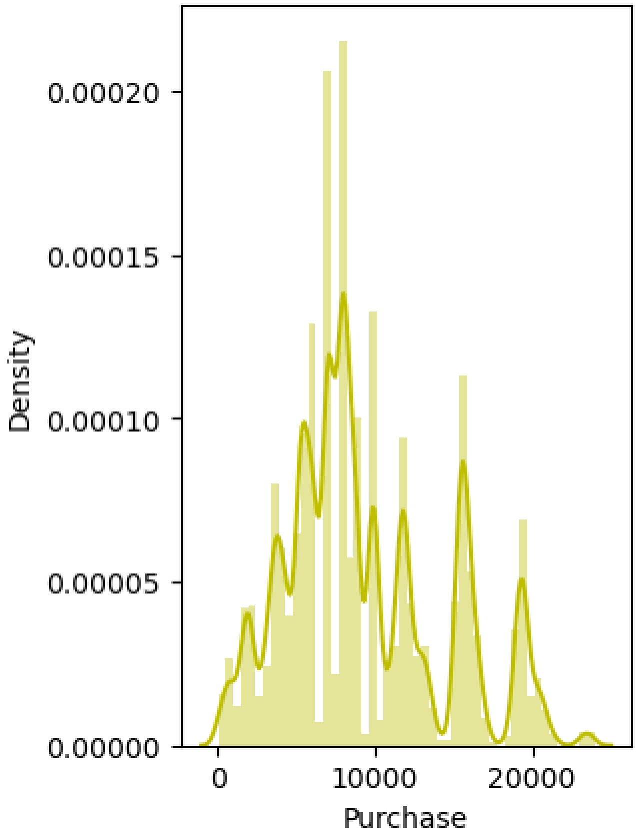
In [21]:
```
sns.boxplot(data=df, x='Purchase', orient='h')
plt.show()
```



In [22]:
```
plt.subplot(1,2,2)
sns.distplot(df["Purchase"],color='y')
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_30772\3824452842.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Purchase"],color='y')
```

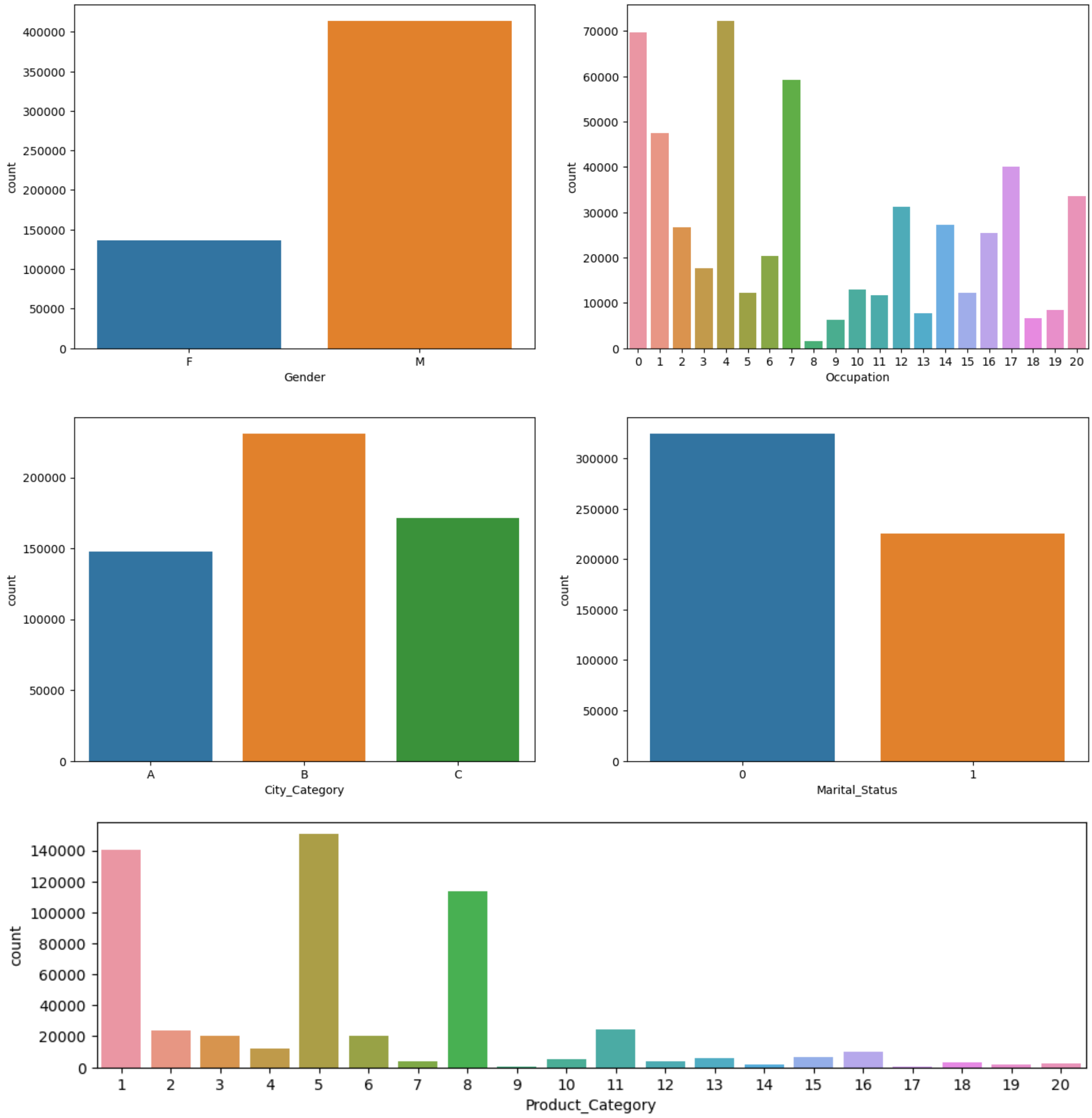
Out[22]: `<Axes: xlabel='Purchase', ylabel='Density'>`



b) For categorical variable(s): Boxplot

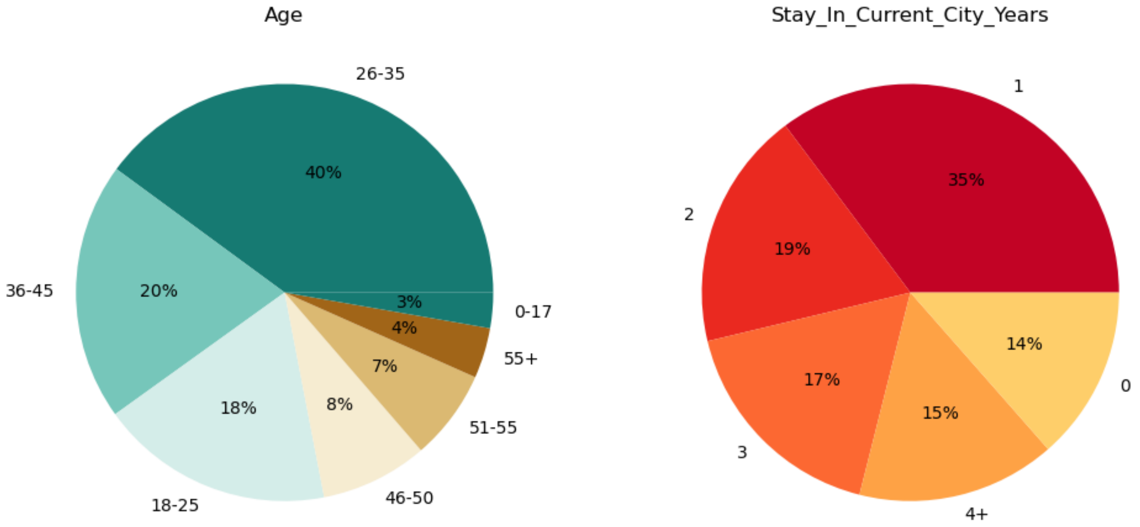
Understanding the distribution of data for the categorical variables - Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status and Product_Category

In [27]:
```
Categorical_Columns = ['Gender', 'Occupation','City_Category','Marital_Status','Pr

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

plt.figure(figsize=(12, 3))
sns.countplot(data=df, x='Product_Category')
plt.show()
```



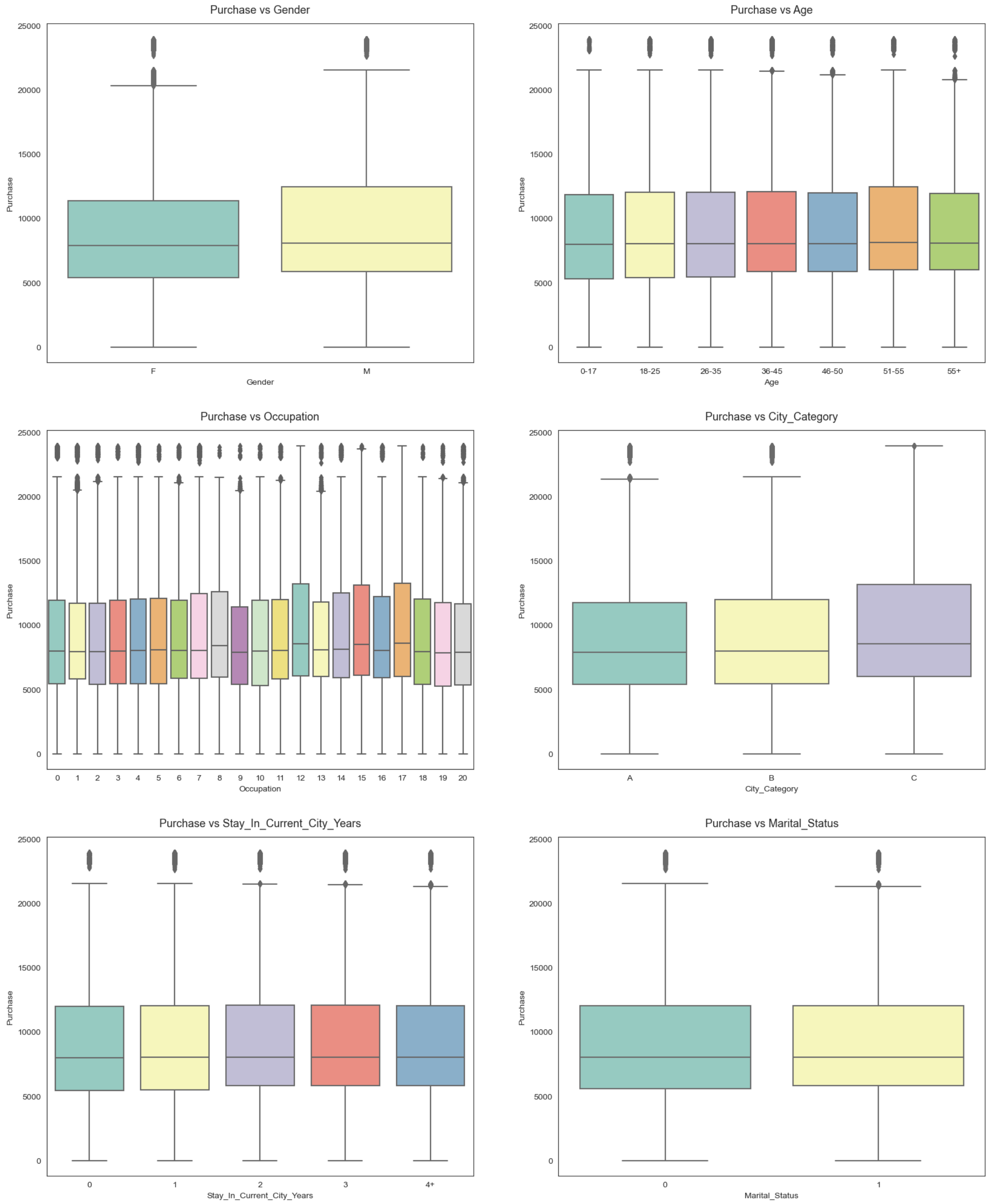
In [28]:

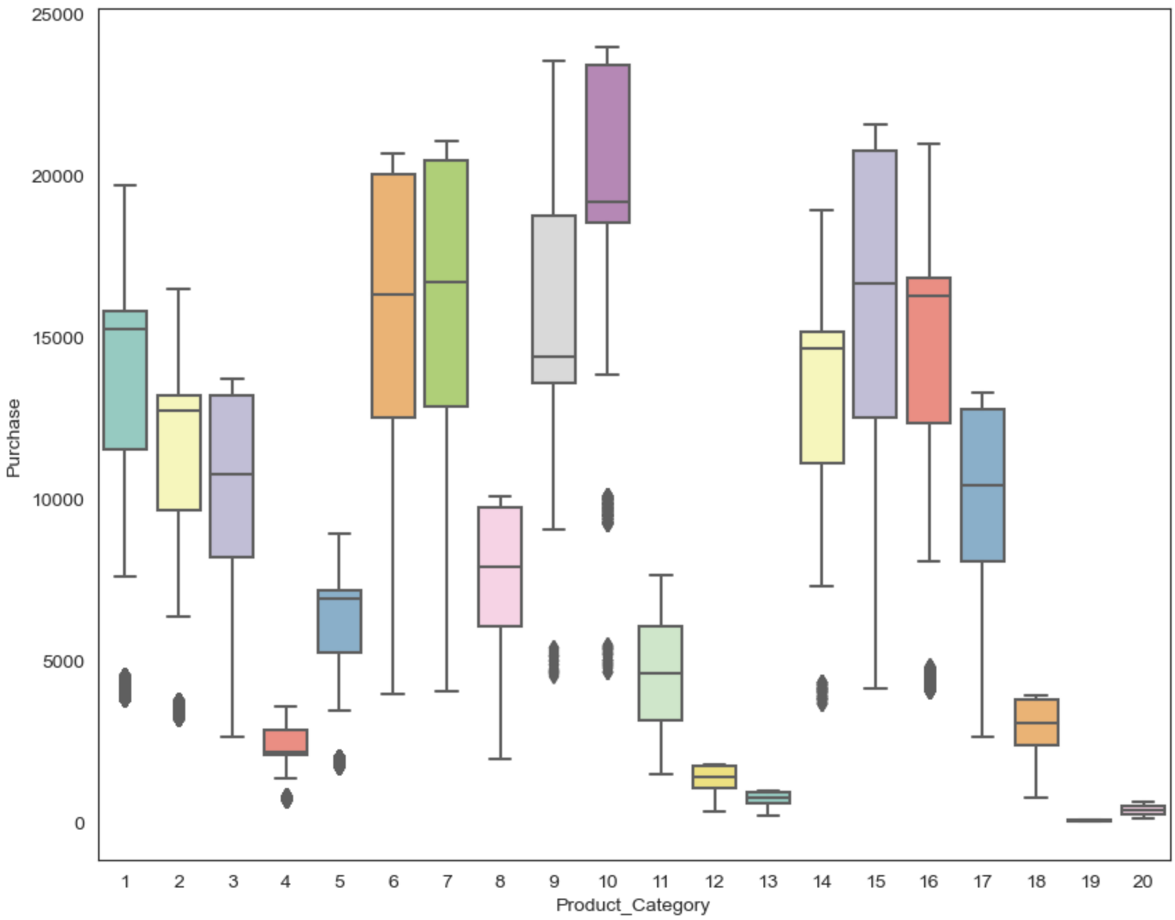
```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(12, 8))

data = df['Age'].value_counts(normalize=True)*100
palette_color = sns.color_palette('BrBG_r')
axs[0].pie(x=data.values, labels=data.index, autopct='%.0f%%', colors=palette_colo
axs[0].set_title("Age")

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


plt.show()
```



In [60]:

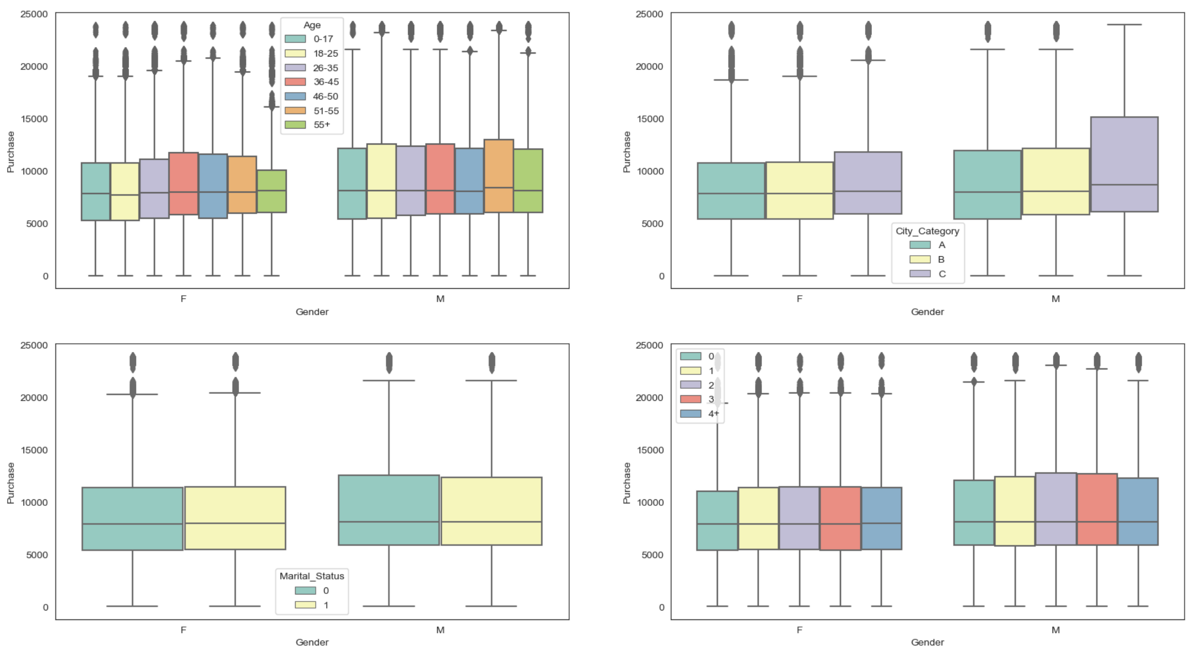
```
#Bi-variate Analysis

attrs = ['Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Current_City_Year
sns.set_style("white")

fig, axs = plt.subplots(nrows=3, ncols=2, figsize=(20, 16))
fig.subplots_adjust(top=1.3)
count = 0
for row in range(3):
    for col in range(2):
        sns.boxplot(data=df, y='Purchase', x=attrs[count], ax=axs[row, col], palet
        axs[row,col].set_title(f"Purchase vs {attrs[count]}", pad=12, fontsize=13)
        count += 1
plt.show()

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


Purchase vs Gender
Purchase
Gender
F
M
Purchase vs Age
Age
0-17
18-25
26-35
36-45
46-50
51-55
55+
Purchase vs Occupation
Occupation
Purchase vs City_Category
City_Category
A
B
C
Purchase vs Stay_In_Current_City_Years
Stay_In_Current_City_Years
0
1
2
3
4+
Purchase vs Marital_Status
Marital_Status
0
1



In [62]:

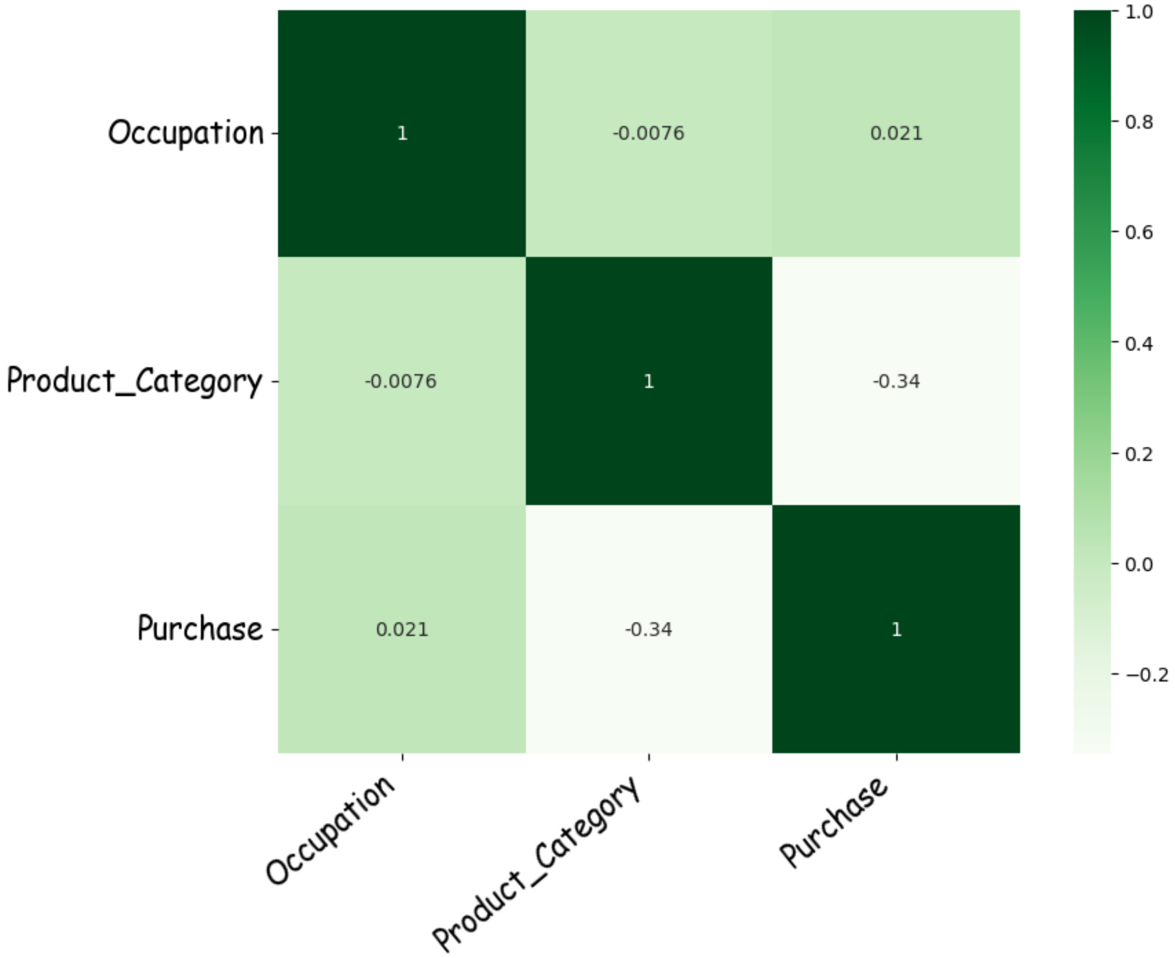
```
#Multivariate Analysis

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

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

plt.show()
```



c) For correlation: Heatmaps, Pairplots

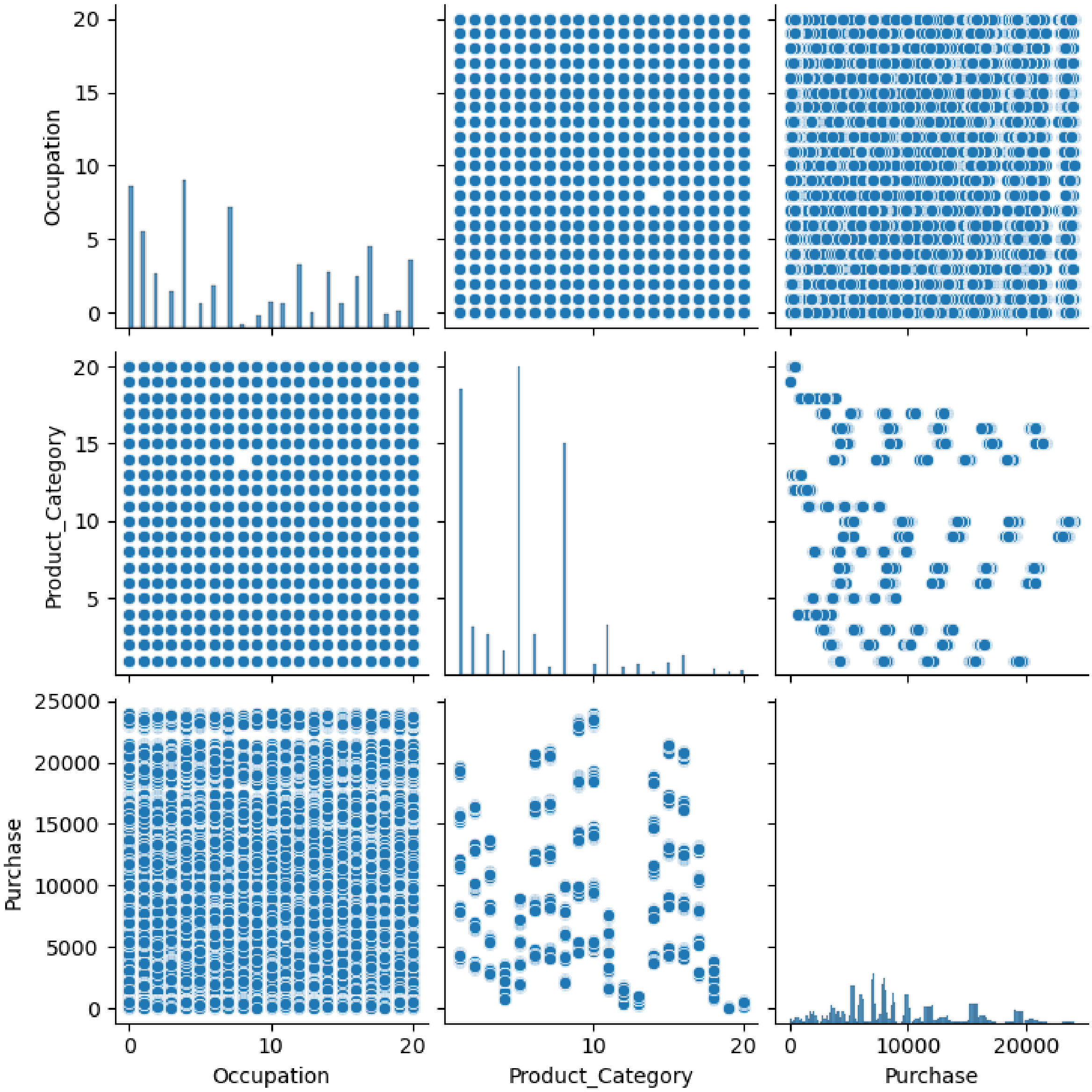
In [51]:
```
plt.figure(figsize = (10, 7))
ax = sns.heatmap(df.corr(),
            annot=True,cmap='Greens',square=True)

ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=40,fontsize=16,family = "Comic Sans MS",
    horizontalalignment='right')

ax.set_yticklabels(
    ax.get_yticklabels(),
    rotation=0,fontsize=16,family = "Comic Sans MS",
    horizontalalignment='right')

plt.show()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_30772\2082957.py:2: FutureWarning: The de
fault value of numeric_only in DataFrame.corr is deprecated. In a future version,
it will default to False. Select only valid columns or specify the value of numeri
c_only to silence this warning.
  ax = sns.heatmap(df.corr(),
```



In [52]:
```
sns.pairplot(df)
```

Out[52]: <seaborn.axisgrid.PairGrid at 0x2630f46bf40>



Observations:

1. Most of the users are Male
2. There are 20 different types of Occupation and Product_Category
3. More users belong to B City_Category
4. More users are Single as compare to Married
5. Product_Category: 1, 5, 8, & 11 have highest purchasing frequency
6. More puchases have been made by males than females.
7. People of age group 26–35 have made the maximum number of purchases.
8. People in cities of category B have made maximum number of purchases.
9. People who have stayed in their city for a year have made the maximum number of purchases.
10. Unmarried people have made more purchases than married peole.
11. Products of category 1, 5 and 8 sold most frequently.
12. Purchases of amount (5000, 10000] were maximum in number.
13. People of occupation 0,4 and 7 have made more purchases than other occupations.
14. People of occupation 8 have made least purchases.
15. Both males and females of city category B make more purchases compared to city categories A and C.

16. Females purchase products of category 4, 11, 15, 17 and 18 less often.
17. Most popular product category among males is 1.
18. Most popular product category among females is 5. It is popular among male customers as well.
19. Females with occupation 0–10 made more purchases than females with occupations 11–20.

4.1 Are women spending more money per transaction than men? Why or Why not?

In [64]:
```
# Average amount spend per customer for Male and Female
amt_df = df.groupby(['User_ID', 'Gender'])[['Purchase']].sum()
amt_df = amt_df.reset_index()
amt_df
```

Out[64]:

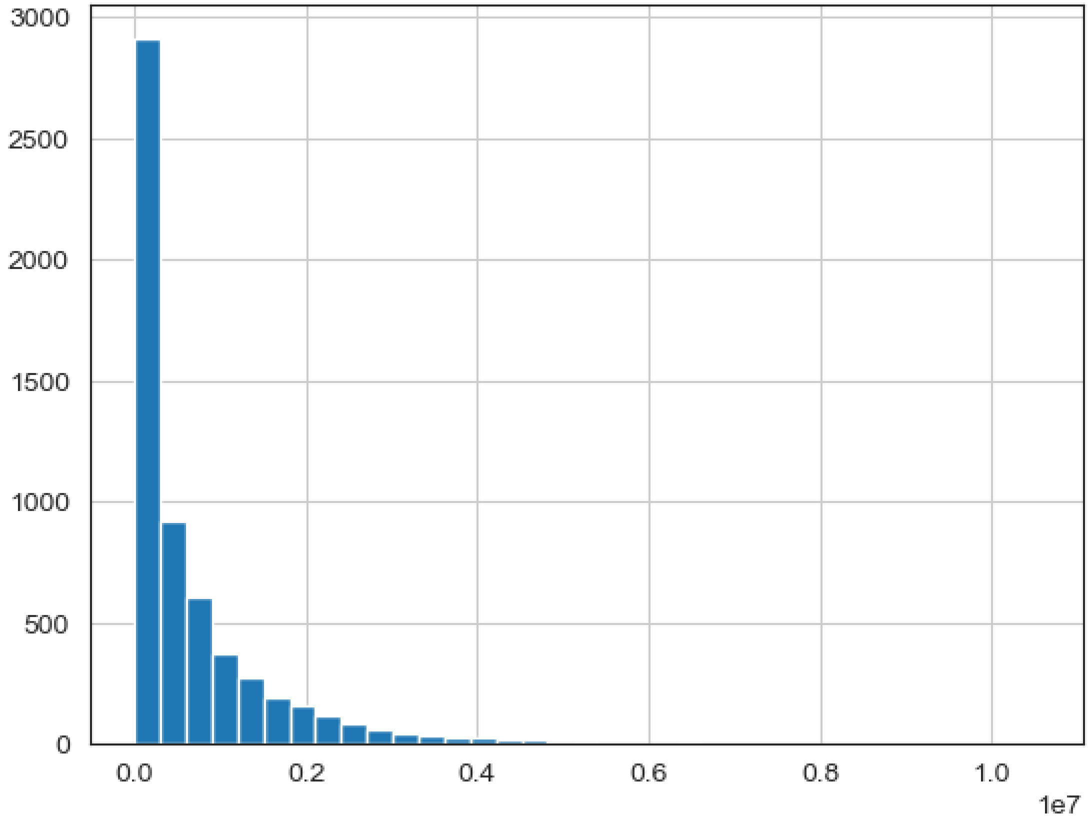
	User_ID	Gender	Purchase
0	1000001	F	334093
1	1000001	M	0
2	1000002	F	0
3	1000002	M	810472
4	1000003	F	0
...	...	...	...
11777	1006038	M	0
11778	1006039	F	590319
11779	1006039	M	0
11780	1006040	F	0
11781	1006040	M	1653299

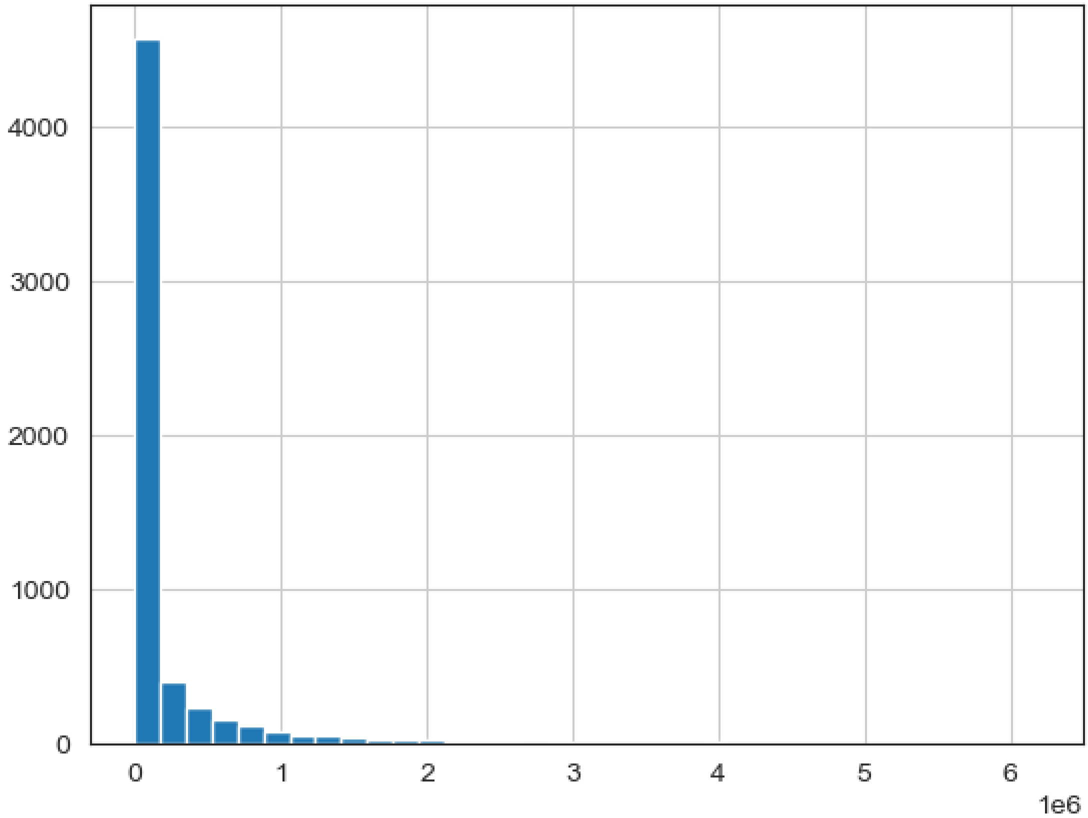
11782 rows × 3 columns

In [66]:
```
# Gender wise value counts in avg_amt_df
amt_df['Gender'].value_counts()
```

Out[66]:
```
F    5891
M    5891
Name: Gender, dtype: int64
```

In [67]:
```
# histogram of average amount spend for each customer - Male & Female
amt_df[amt_df['Gender']=='M']['Purchase'].hist(bins=35)
plt.show()

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





In [68]:
```
male_avg = amt_df[amt_df['Gender']=='M']['Purchase'].mean()
female_avg = amt_df[amt_df['Gender']=='F']['Purchase'].mean()

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

```
Average amount spend by Male customers: 663653.05
Average amount spend by Female customers: 201363.54
```

Observation:

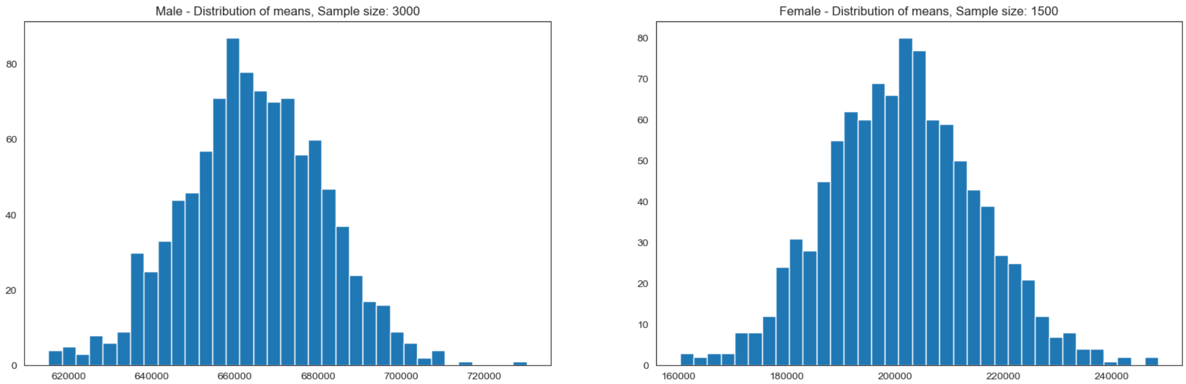
As Average amount spend by Male customers is more than that of female customers, Male customers spend more money per transaction than female customers

4.2 Confidence intervals and distribution of the mean of the expenses by female and male customers

In [69]:
```
male_df = amt_df[amt_df['Gender']=='M']
female_df = amt_df[amt_df['Gender']=='F']
```

In [70]:
```
genders = ["M", "F"]

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

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

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

In [71]:
```
fig, axis = plt.subplots(nrows=1, ncols=2, figsize=(20, 6))

axis[0].hist(male_means, bins=35)
axis[1].hist(female_means, bins=35)
axis[0].set_title("Male - Distribution of means, Sample size: 3000")
axis[1].set_title("Female - Distribution of means, Sample size: 1500")

plt.show()
```



In [72]:
```
print("Population mean - Mean of sample means of amount spend for Male: {:.2f}".for
print("Population mean - Mean of sample means of amount spend for Female: {:.2f}".

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

```
Population mean - Mean of sample means of amount spend for Male: 664341.60
Population mean - Mean of sample means of amount spend for Female: 201424.37

Male - Sample mean: 663653.05 Sample std: 933096.80
Female - Sample mean: 201363.54 Sample std: 535828.17
```

Observation

Using the Central Limit Theorem for the population we can say that:

Average amount spend by male customers is 9,26,341.86 Average amount spend by female customers is 7,11,704.09

In [73]:
```
male_margin_of_error_clt = 1.96*male_df['Purchase'].std()/np.sqrt(len(male_df))
male_sample_mean = male_df['Purchase'].mean()
male_lower_lim = male_sample_mean - male_margin_of_error_clt
male_upper_lim = male_sample_mean + male_margin_of_error_clt

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

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

```
Male confidence interval of means: (639825.01, 687481.08)
Female confidence interval of means: (187680.36, 215046.73)
```

Observations:

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

Average amount spend by male customer will lie in between: (895617.83, 955070.97)
Average amount spend by female customer will lie in between: (673254.77, 750794.02)

1. Mean purchase amount for females = 8734.56
2. Mean purchase amount for males = 9437.52
3. 95% confidence interval for purchase amounts of females is less than males without any intersection.
4. We can say with 95% confidence that females spend less than males.

4.3 Are confidence intervals of average male and female spending overlapping? How can Walmart leverage this conclusion to make changes or improvements?

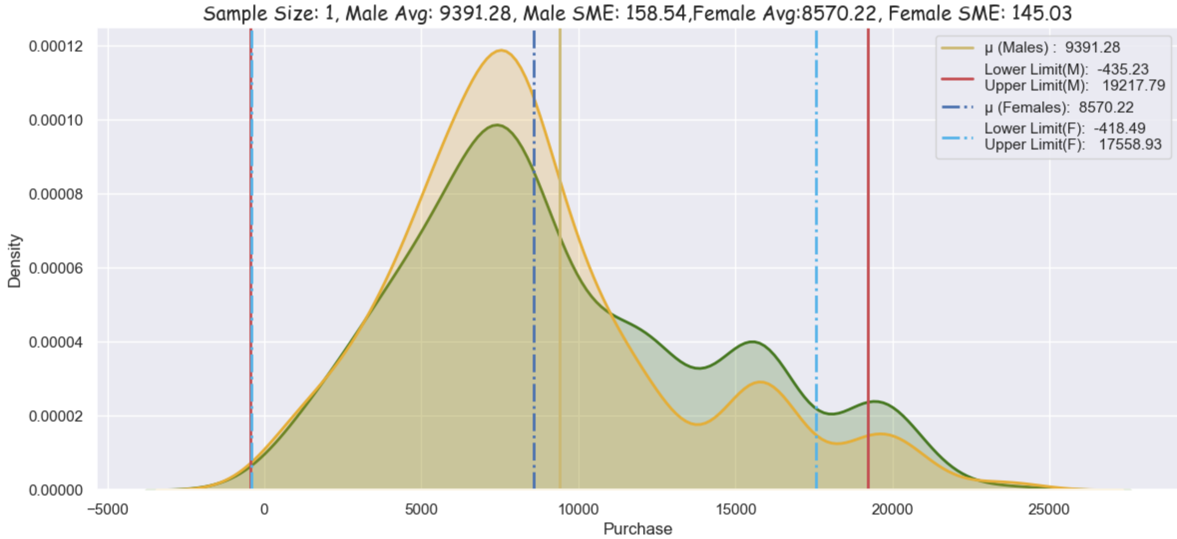
In [83]:
```
def bootstrapping(sample1,sample2,smp_siz=500,itr_size=5000,confidence_level=0.95,

    smp1_means_m = np.empty(itr_size)
    smp2_means_m = np.empty(itr_size)
    for i in range(itr_size):
        smp1_n = np.empty(smp_siz)
        smp2_n = np.empty(smp_siz)
        smp1_n = np.random.choice(sample1, size = smp_siz,replace=True)
        smp2_n = np.random.choice(sample2, size = smp_siz,replace=True)
        smp1_means_m[i] = np.mean(smp1_n)
        smp2_means_m[i] = np.mean(smp2_n)

    #Calcualte the Z-Critical value
    alpha = (1 - confidence_level)/no_of_tails
    z_critical = stats.norm.ppf(1 - alpha)

    # Calculate the mean, standard deviation & standard Error of sampling distribut
    mean1  = np.mean(smp1_means_m)
    sigma1 = statistics.stdev(smp1_means_m)
    sem1   = stats.sem(smp1_means_m)

    lower_limit1 = mean1 - (z_critical * sigma1)
```

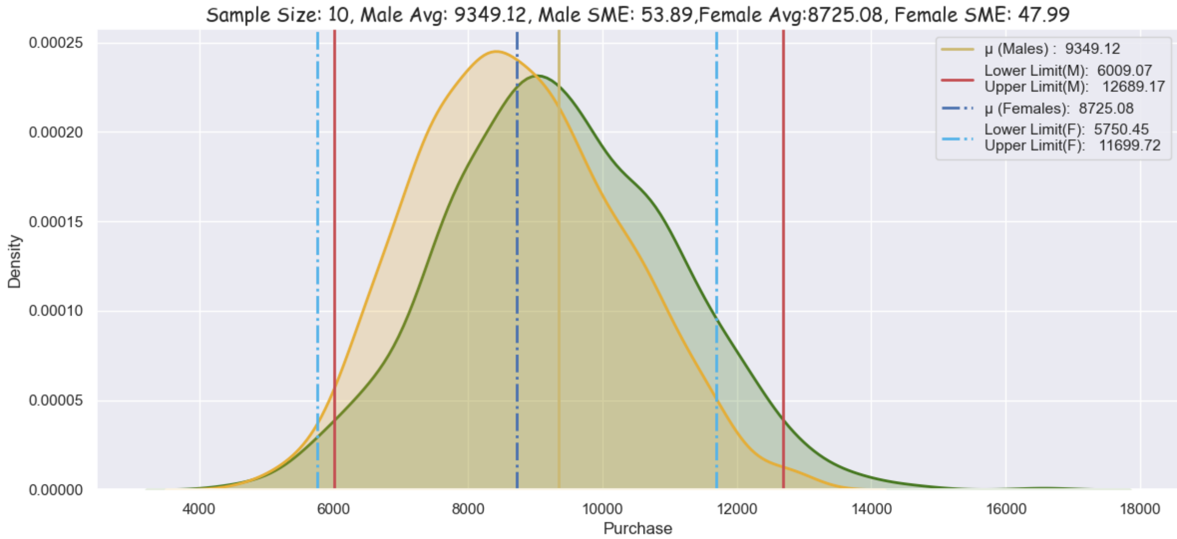
```
    upper_limit1 = mean1 + (z_critical * sigma1)

    # Calculate the mean, standard deviation & standard Error of sampling distribut
    mean2  = np.mean(smp2_means_m)
    sigma2 = statistics.stdev(smp2_means_m)
    sem2   = stats.sem(smp2_means_m)

    lower_limit2 = mean2 - (z_critical * sigma2)
    upper_limit2 = mean2 + (z_critical * sigma2)

    fig, ax = plt.subplots(figsize=(14,6))
    sns.set_style("darkgrid")

    sns.kdeplot(data=smp1_means_m,color="#467821",fill=True,linewidth=2)
    sns.kdeplot(data=smp2_means_m,color='#e5ae38',fill=True,linewidth=2)

    label_mean1=("μ (Males) :  {:.2f}".format(mean1))
    label_ult1=("Lower Limit(M):  {:.2f}\nUpper Limit(M):   {:.2f}".format(lower_l
    label_mean2=("μ (Females):  {:.2f}".format(mean2))
    label_ult2=("Lower Limit(F):  {:.2f}\nUpper Limit(F):   {:.2f}".format(lower_l

    plt.title(f"Sample Size: {smp_siz}, Male Avg: {np.round(mean1, 2)}, Male SME: 
              fontsize=14,family = "Comic Sans MS")
    plt.xlabel('Purchase')
    plt.axvline(mean1, color = 'y', linestyle = 'solid', linewidth = 2,label=label
    plt.axvline(upper_limit1, color = 'r', linestyle = 'solid', linewidth = 2,label
    plt.axvline(lower_limit1, color = 'r', linestyle = 'solid', linewidth = 2)
    plt.axvline(mean2, color = 'b', linestyle = 'dashdot', linewidth = 2,label=lab
    plt.axvline(upper_limit2, color = '#56B4E9', linestyle = 'dashdot', linewidth 
    plt.axvline(lower_limit2, color = '#56B4E9', linestyle = 'dashdot', linewidth 
    plt.legend(loc='upper right')

    plt.show()

    return smp1_means_m,smp2_means_m ,np.round(lower_limit1,2),np.round(upper_limi
```

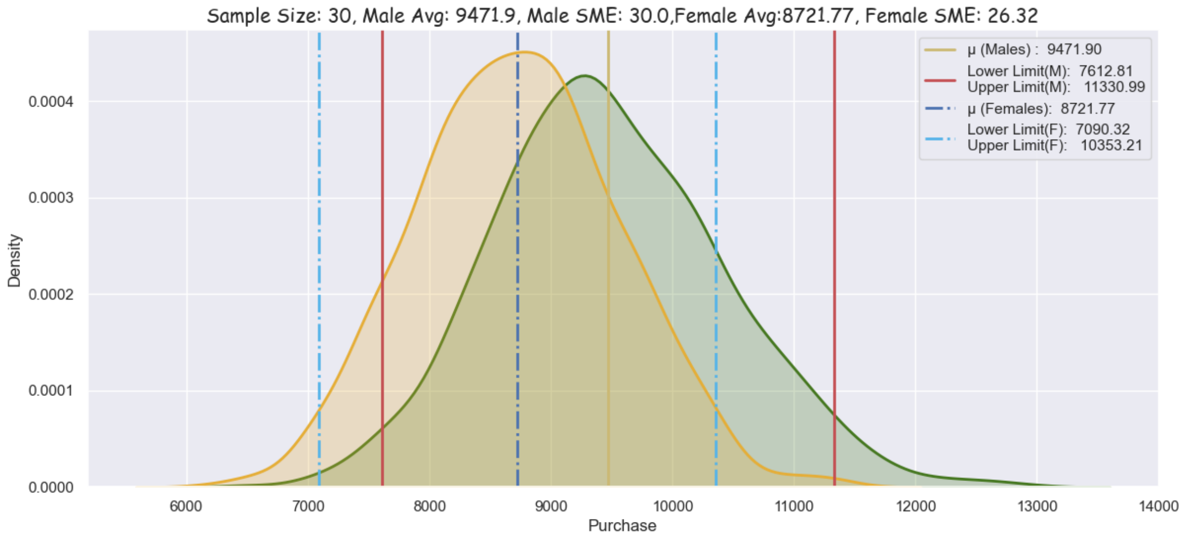
In [79]:
```
retail_data_smp_male = df[df['Gender'] == 'M']['Purchase']
retail_data_smp_female = df[df['Gender'] == 'F']['Purchase']
print("Male Customers : ",retail_data_smp_male.shape[0])
print("Female Customers : ",retail_data_smp_female.shape[0])
```

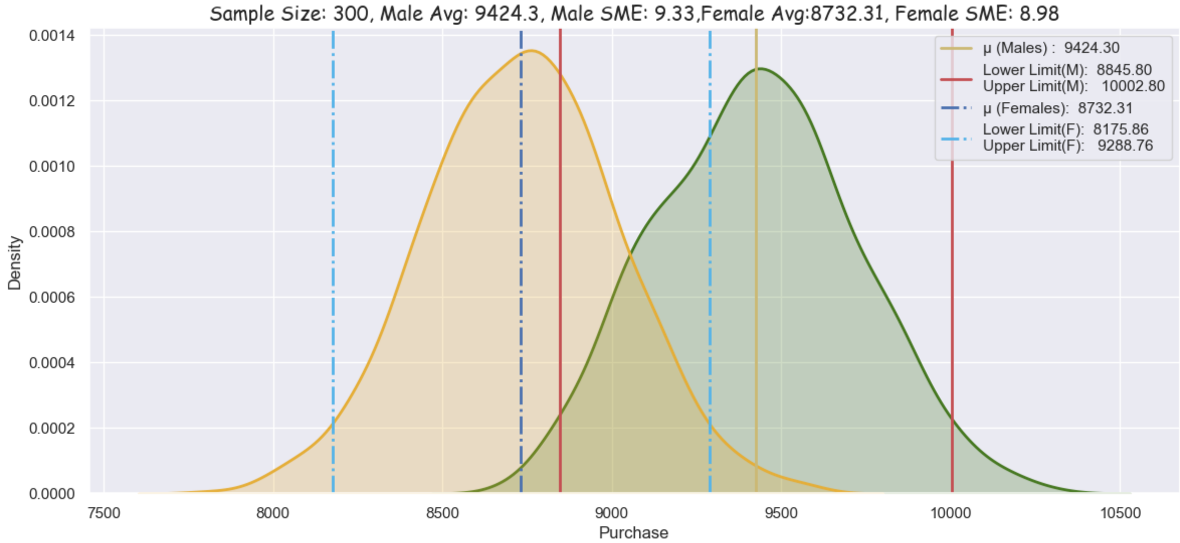
```
Male Customers :  414259
Female Customers :  135809
```

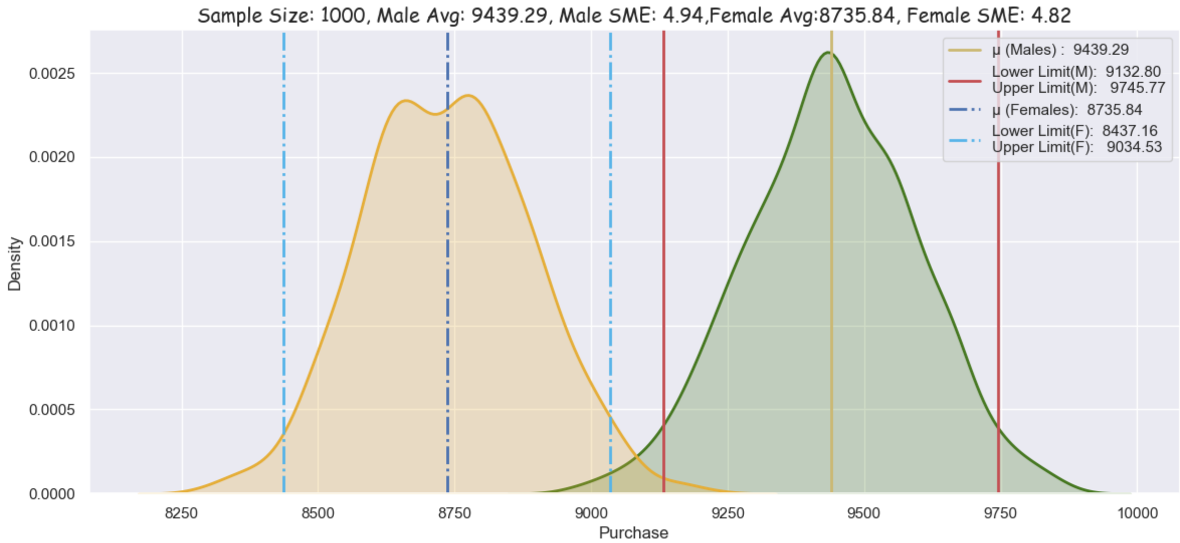
In [85]:
```
# CLT Analysis for mean purchase with confidence 95% - Based on Gender

itr_size = 1000
size_list = [1, 10, 30, 300, 1000, 100000]
ci = 0.95

array = np.empty((0,7))

for smp_siz in size_list:
    m_avg, f_avg, ll_m, ul_m, ll_f, ul_f = bootstrapping(retail_data_smp_male,reta

    array = np.append(array, np.array([['M', ll_m, ul_m, smp_siz, ([ll_m,ul_m]) ,(
    array = np.append(array, np.array([['F', ll_f, ul_f, smp_siz, ([ll_f,ul_f]) ,(

overlap_95 = pd.DataFrame(array, columns = ['Gender','Lower_limit','Upper_limit','
overlap = pd.concat([overlap, overlap_95], axis=0)
```


Sample Size: 1, Male Avg: 9391.28, Male SME: 158.54,Female Avg:8570.22, Female SME: 145.03
μ (Males) : 9391.28
Lower Limit(M): -435.23
Upper Limit(M): 19217.79
μ (Females): 8570.22
Lower Limit(F): -418.49
Upper Limit(F): 17558.93
Density
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
−5000
0
5000
10000
15000
20000
25000
Purchase


Sample Size: 10, Male Avg: 9349.12, Male SME: 53.89,Female Avg:8725.08, Female SME: 47.99
μ (Males) : 9349.12
Lower Limit(M): 6009.07
Upper Limit(M): 12689.17
μ (Females): 8725.08
Lower Limit(F): 5750.45
Upper Limit(F): 11699.72
Density
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
4000
6000
8000
10000
12000
14000
16000
18000
Purchase


Sample Size: 30, Male Avg: 9471.9, Male SME: 30.0,Female Avg:8721.77, Female SME: 26.32
μ (Males) : 9471.90
Lower Limit(M): 7612.81
Upper Limit(M): 11330.99
μ (Females): 8721.77
Lower Limit(F): 7090.32
Upper Limit(F): 10353.21
Density
0.0004
0.0003
0.0002
0.0001
0.0000
6000
7000
8000
9000
10000
11000
12000
13000
14000
Purchase





In [86]:
```
overlap_95.loc[(overlap_95['Gender'] == 'M') & (overlap_95['Sample_Size'] >= 300)]
```

Out[86]:

	Gender	Lower_limit	Upper_limit	Sample_Size	CI	Range	Confidence_pct
6	M	8845.8	10002.8	300	[8845.8, 10002.8]	1157.0	95
8	M	9132.8	9745.77	1000	[9132.8, 9745.77]	612.97	95
10	M	9406.95	9470.25	100000	[9406.95, 9470.25]	63.3	95

In [87]:
```
overlap_95.loc[(overlap_95['Gender'] == 'F') & (overlap_95['Sample_Size'] >= 300)]
```

Out[87]:

	Gender	Lower_limit	Upper_limit	Sample_Size	CI	Range	Confidence_pct
7	F	8175.86	9288.76	300	[8175.86, 9288.76]	1112.9	95
9	F	8437.16	9034.53	1000	[8437.16, 9034.53]	597.37	95
11	F	8704.13	8765.52	100000	[8704.13, 8765.52]	61.39	95

Observations:

Using confidence interval 95%: As the sample size increases, the Male and female groups start to become distinct With increasing sample size, Standard error of the mean in the samples decreases. For sample size 100000 is 0.47 For Female (sample size 100000) range for mean purchase with confidence interval 90% is [8642.58, 8701.58] For Male range for mean purchase with confidence interval 95% is [9336.23, 9397.53] Overlappings are increasing with a confidence interval of 95%. Due to the increasing CI, we consider higher ranges within which the actual population might fall, so that both mean purchase are more likely to fall within the same range.

4.4 Analysis based on Married vs Unmarried

In [90]:
```
amt_df = df.groupby(['User_ID', 'Marital_Status'])[['Purchase']].sum()
amt_df = amt_df.reset_index()
amt_df
```

Out[90]:

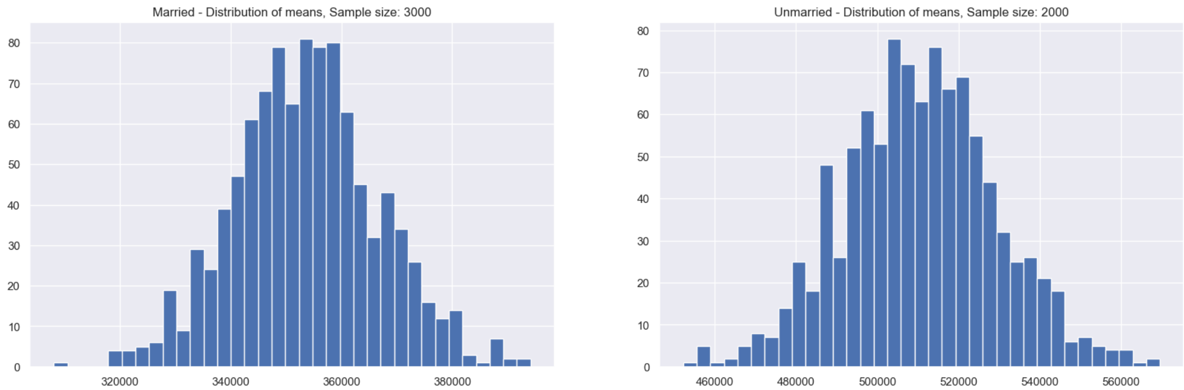
	User_ID	Marital_Status	Purchase
0	1000001	0	334093
1	1000001	1	0
2	1000002	0	810472
3	1000002	1	0
4	1000003	0	341635
...	...	...	...
11777	1006038	1	0
11778	1006039	0	0
11779	1006039	1	590319
11780	1006040	0	1653299
11781	1006040	1	0

11782 rows × 3 columns

In [91]:
```
amt_df['Marital_Status'].value_counts()
```

Out[91]:
```
0    5891
1    5891
Name: Marital_Status, dtype: int64
```

In [92]:
```
marid_samp_size = 3000
unmarid_sample_size = 2000
num_repitions = 1000
marid_means = []
unmarid_means = []

for _ in range(num_repitions):
    marid_mean = amt_df[amt_df['Marital_Status']==1].sample(marid_samp_size, repla
    unmarid_mean = amt_df[amt_df['Marital_Status']==0].sample(unmarid_sample_size,

    marid_means.append(marid_mean)
    unmarid_means.append(unmarid_mean)


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

axis[0].hist(marid_means, bins=35)
axis[1].hist(unmarid_means, bins=35)
axis[0].set_title("Married - Distribution of means, Sample size: 3000")
axis[1].set_title("Unmarried - Distribution of means, Sample size: 2000")

plt.show()

print("Population mean - Mean of sample means of amount spend for Married: {:.2f}"
print("Population mean - Mean of sample means of amount spend for Unmarried: {:.2f

print("\nMarried - Sample mean: {:.2f} Sample std: {:.2f}".format(amt_df[amt_df['Ma
print("Unmarried - Sample mean: {:.2f} Sample std: {:.2f}".format(amt_df[amt_df['Ma
```




```
Population mean - Mean of sample means of amount spend for Married: 353676.93
Population mean - Mean of sample means of amount spend for Unmarried: 510595.49

Married - Sample mean: 354249.75 Sample std: 735314.88
Unmarried - Sample mean: 510766.84 Sample std: 843632.94
```

In [93]:
```
for val in ["Married", "Unmarried"]:

    new_val = 1 if val == "Married" else 0

    new_df = amt_df[amt_df['Marital_Status']==new_val]

    margin_of_error_clt = 1.96*new_df['Purchase'].std()/np.sqrt(len(new_df))
    sample_mean = new_df['Purchase'].mean()
    lower_lim = sample_mean - margin_of_error_clt
    upper_lim = sample_mean + margin_of_error_clt

    print("{} confidence interval of means: ({:.2f}, {:.2f})".format(val, lower_li
```

```
Married confidence interval of means: (335472.38, 373027.13)
Unmarried confidence interval of means: (489223.40, 532310.28)
```

In [114…

```
def bootstrapping_m_vs_um(sample1,sample2,smp_siz=500,itr_size=5000,confidence_leve

    smp1_means_m = np.empty(itr_size)
    smp2_means_m = np.empty(itr_size)
    for i in range(itr_size):
        smp1_n = np.empty(smp_siz)
        smp2_n = np.empty(smp_siz)
        smp1_n = np.random.choice(sample1, size = smp_siz,replace=True)
        smp2_n = np.random.choice(sample2, size = smp_siz,replace=True)
        smp1_means_m[i] = np.mean(smp1_n)
        smp2_means_m[i] = np.mean(smp2_n)

#     std_dev1 = np.std(sample1)
#     std_err1 = np.std(sample1,ddof=1)/np.sqrt(smp_siz)
#     std_dev2 = np.std(sample2)
#     std_err2 = np.std(sample2,ddof=1)/np.sqrt(smp_siz)

    #Calcualte the Z-Critical value
    alpha = (1 - confidence_level)/no_of_tails
    z_critical = stats.norm.ppf(1 - alpha)

    # Calculate the mean, standard deviation & standard Error of sampling distribut
    mean1  = np.mean(smp1_means_m)
    sigma1 = statistics.stdev(smp1_means_m)
    sem1   = stats.sem(smp1_means_m)

    lower_limit1 = mean1 - (z_critical * sigma1)
    upper_limit1 = mean1 + (z_critical * sigma1)

    # Calculate the mean, standard deviation & standard Error of sampling distribut
    mean2  = np.mean(smp2_means_m)
    sigma2 = statistics.stdev(smp2_means_m)
    sem2   = stats.sem(smp2_means_m)

#     print(smp_siz,std_dev1,std_err1,sem1)
#     print(smp_siz,std_dev2,std_err2,sem2)

    lower_limit2 = mean2 - (z_critical * sigma2)
    upper_limit2 = mean2 + (z_critical * sigma2)

    fig, ax = plt.subplots(figsize=(14,6))
    sns.set_style("darkgrid")

    sns.kdeplot(data=smp1_means_m,color="#467821",fill=True,linewidth=2)
    sns.kdeplot(data=smp2_means_m,color='#e5ae38',fill=True,linewidth=2)

    label_mean1=("μ (Married) :  {:.2f}".format(mean1))
    label_ult1=("Lower Limit(M):  {:.2f}\nUpper Limit(M):   {:.2f}".format(lower_l
    label_mean2=("μ (Unmarried):  {:.2f}".format(mean2))
    label_ult2=("Lower Limit(F):  {:.2f}\nUpper Limit(F):   {:.2f}".format(lower_l

    plt.title(f"Sample Size: {smp_siz}, Married Avg: {np.round(mean1, 2)}, Married
              fontsize=14,family = "Comic Sans MS")
    plt.xlabel('Purchase')
    plt.axvline(mean1, color = 'y', linestyle = 'solid', linewidth = 2,label=label_
    plt.axvline(upper_limit1, color = 'r', linestyle = 'solid', linewidth = 2,label
    plt.axvline(lower_limit1, color = 'r', linestyle = 'solid', linewidth = 2)
    plt.axvline(mean2, color = 'b', linestyle = 'dashdot', linewidth = 2,label=lab
    plt.axvline(upper_limit2, color = '#56B4E9', linestyle = 'dashdot', linewidth 
    plt.axvline(lower_limit2, color = '#56B4E9', linestyle = 'dashdot', linewidth 
    plt.legend(loc='upper right')

    plt.show()
```

```
    return smp1_means_m,smp2_means_m ,np.round(lower_limit1,2),np.round(upper_limi
```

In [116…

```
def bootstrapping_age(sample,smp_siz=500,itr_size=5000,confidence_level=0.99,no_of_

    smp_means_m = np.empty(itr_size)
    for i in range(itr_size):
        smp_n = np.empty(smp_siz)
        smp_n = np.random.choice(sample, size = smp_siz,replace=True)
        smp_means_m[i] = np.mean(smp_n)

    #Calcualte the Z-Critical value
    alpha = (1 - confidence_level)/no_of_tails
    z_critical = stats.norm.ppf(1 - alpha)

    # Calculate the mean, standard deviation & standard Error of sampling distribut
    mean  = np.mean(smp_means_m)
    sigma = statistics.stdev(smp_means_m)
    sem   = stats.sem(smp_means_m)

    lower_limit = mean - (z_critical * sigma)
    upper_limit = mean + (z_critical * sigma)

    fig, ax = plt.subplots(figsize=(14,6))
    sns.set_style("darkgrid")

    sns.kdeplot(data=smp_means_m,color="#7A68A6",fill=True,linewidth=2)

    label_mean=("μ :  {:.2f}".format(mean))
    label_ult=("Lower Limit:  {:.2f}\nUpper Limit:   {:.2f}".format(lower_limit,up

    plt.title(f"Sample Size: {smp_siz},Mean:{np.round(mean,2)}, SME:{np.round(sem,
    plt.xlabel('Purchase')
    plt.axvline(mean, color = 'y', linestyle = 'solid', linewidth = 2,label=label_
    plt.axvline(upper_limit, color = 'r', linestyle = 'solid', linewidth = 2,label
    plt.axvline(lower_limit, color = 'r', linestyle = 'solid', linewidth = 2)
    plt.legend(loc='upper right')

    plt.show()

    return smp_means_m ,np.round(lower_limit,2),np.round(upper_limit,2)
```

In [117…

```
df['Marital_Status'].replace(to_replace = 0, value = 'Unmarried', inplace = True)
df['Marital_Status'].replace(to_replace = 1, value = 'Married', inplace = True)
```

In [118…

```
df.sample(500,replace=True).groupby(['Marital_Status'])['Purchase'].describe()
```

Out[118]:

	count	mean	std	min	25%	50%	75%	max
Marital_Status								
Unmarried	305.0	9550.4	4812.162596	124.0	6018.0	8658.0	12349.0	21205.0
Married	195.0	9649.8	5016.382768	558.0	6025.0	8641.0	11995.5	23650.0

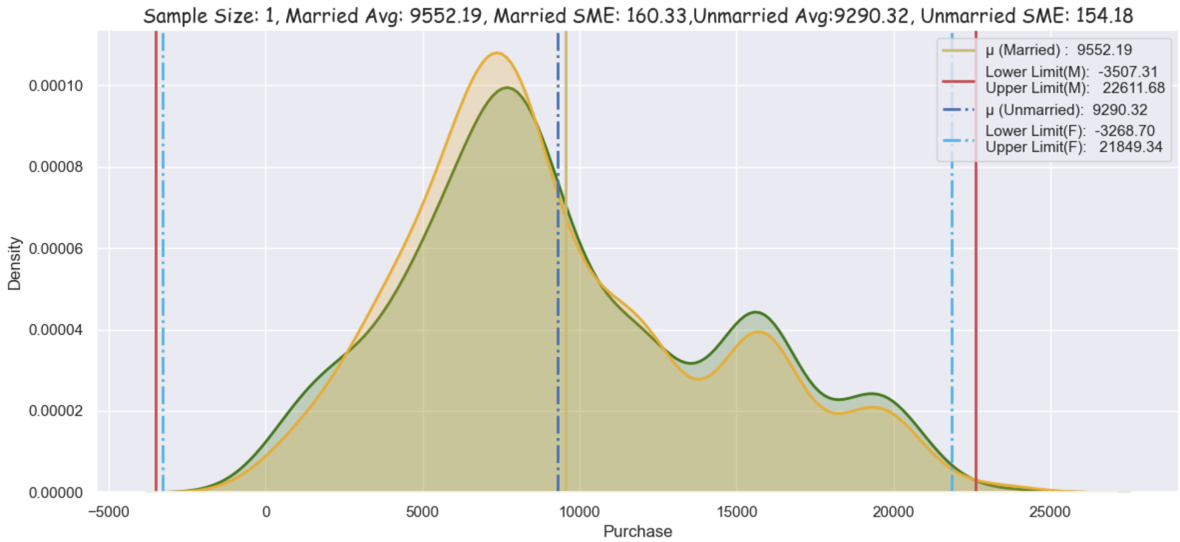
In [119…

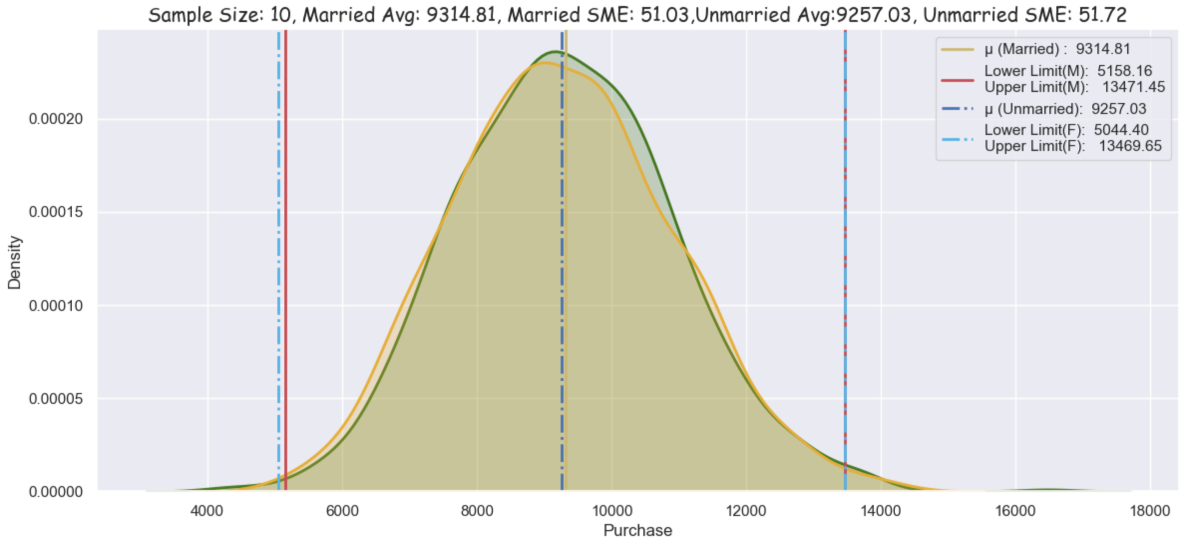
```
retail_data_smp_married = df[df['Marital_Status'] == 'Married']['Purchase']
retail_data_smp_unmarried = df[df['Marital_Status'] == 'Unmarried']['Purchase']
```

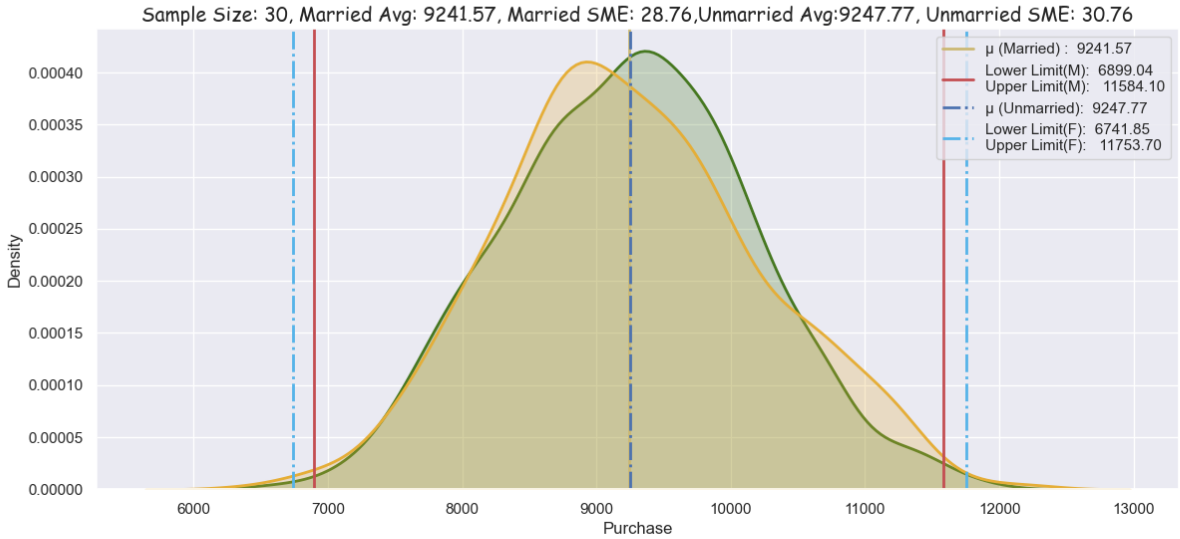
In [120…

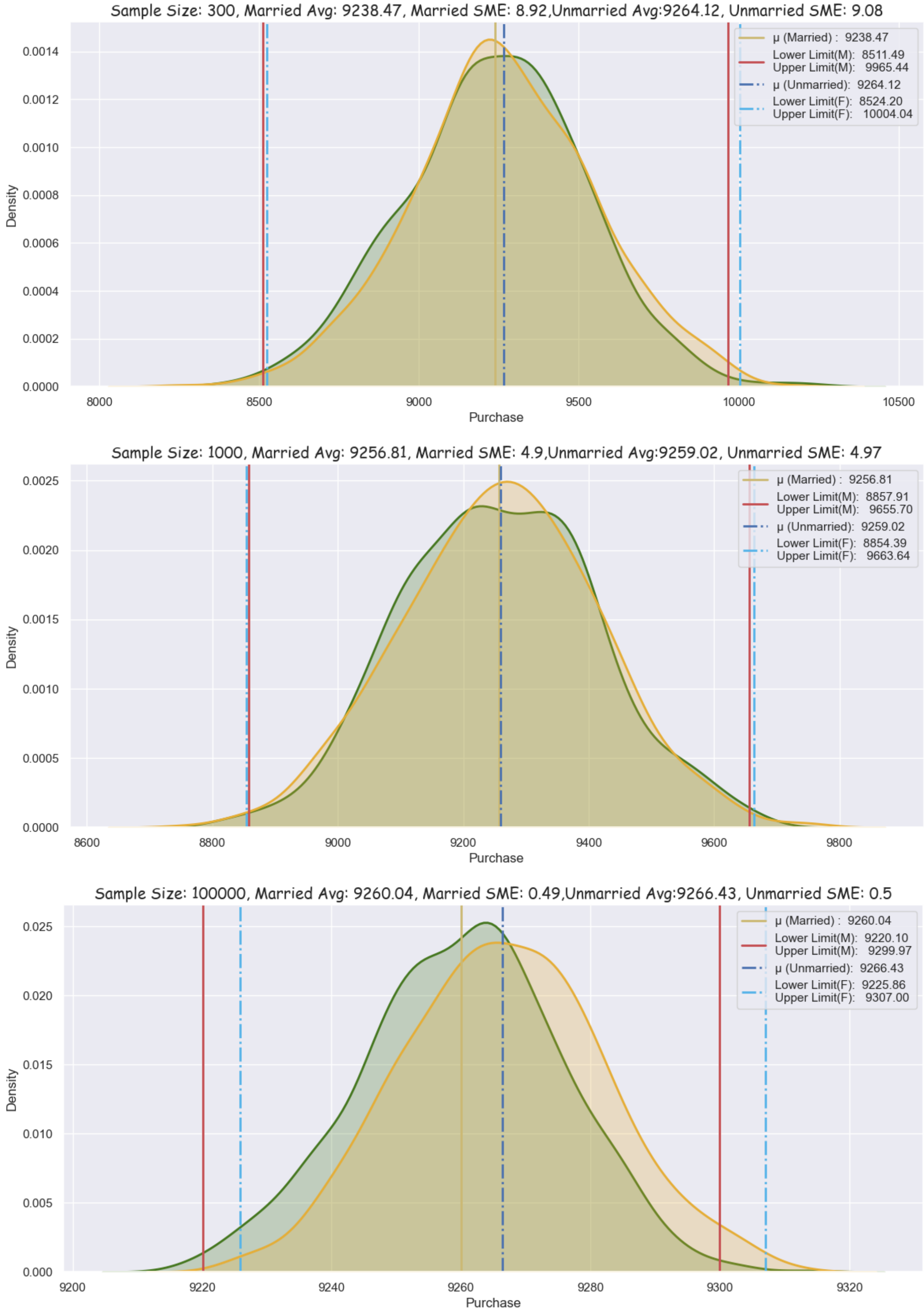
```
itr_size = 1000
size_list = [1, 10, 30, 300, 1000, 100000]
ci = 0.99
```

```
array = np.empty((0,7))

for smp_siz in size_list:
    m_avg, f_avg, ll_m, ul_m, ll_u, ul_u = bootstrapping_m_vs_um(retail_data_smp_ma

    array = np.append(array, np.array([['Married', ll_m, ul_m, smp_siz, ([ll_m,ul_
    array = np.append(array, np.array([['Unmarried', ll_u, ul_u, smp_siz, ([ll_u,u

overlap = pd.DataFrame(array, columns = ['Marital_Status','Lower_limit','Upper_lim
```









In [111…

```
overlap.head()
```

Out[111]:

	Marital_Status	Lower_limit	Upper_limit	Sample_Size	CI	Range	Confidence_pct
0	Married	-3856.03	22009.58	1	[-3856.03, 22009.58]	25865.61	99
1	Unmarried	-3719.03	22274.37	1	[-3719.03, 22274.37]	25993.4	99
2	Married	4946.45	13422.1	10	[4946.45, 13422.1]	8475.65	99
3	Unmarried	5210.28	13302.72	10	[5210.28, 13302.72]	8092.44	99
4	Married	6936.06	11647.62	30	[6936.06, 11647.62]	4711.56	99

In [112…

```
overlap.loc[(overlap['Marital_Status'] == 'Married') & (overlap['Sample_Size'] >= 
```

Out[112]:

	Marital_Status	Lower_limit	Upper_limit	Sample_Size	CI	Range	Confidence_pct
6	Married	8496.19	10020.12	300	[8496.19, 10020.12]	1523.93	99
8	Married	8852.64	9659.28	1000	[8852.64, 9659.28]	806.64	99
10	Married	9219.88	9301.77	100000	[9219.88, 9301.77]	81.89	99

In [113…

```
overlap.loc[(overlap['Marital_Status'] == 'Unmarried') & (overlap['Sample_Size'] >
```

Out[113]:

	Marital_Status	Lower_limit	Upper_limit	Sample_Size	CI	Range	Confidence_pct
7	Unmarried	8505.85	10031.68	300	[8505.85, 10031.68]	1525.83	99
9	Unmarried	8838.89	9680.32	1000	[8838.89, 9680.32]	841.43	99
11	Unmarried	9224.98	9306.32	100000	[9224.98, 9306.32]	81.34	99

Observations:

Overlapping is evident for married vs single customer spend even when more samples are analyzed, which indicates that customers spend the same regardless of whether they are single or married.

4.5 Analysis based on Age

In [122…

```
amt_df = df.groupby(['User_ID', 'Age'])[['Purchase']].sum()
amt_df = amt_df.reset_index()
amt_df
```

Out[122]:

	User_ID	Age	Purchase
0	1000001	0-17	334093
1	1000001	18-25	0
2	1000001	26-35	0
3	1000001	36-45	0
4	1000001	46-50	0
...	...	...	...
41232	1006040	26-35	1653299
41233	1006040	36-45	0
41234	1006040	46-50	0
41235	1006040	51-55	0
41236	1006040	55+	0

41237 rows × 3 columns

In [123…

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

Out[123]:

```
0-17     5891
18-25    5891
26-35    5891
36-45    5891
46-50    5891
51-55    5891
55+      5891
Name: Age, dtype: int64
```

In [124…

```
sample_size = 200
num_repitions = 1000

all_means = {}

age_intervals = ['26-35', '36-45', '18-25', '46-50', '51-55', '55+', '0-17']
for age_interval in age_intervals:
    all_means[age_interval] = []

for age_interval in age_intervals:
    for _ in range(num_repitions):
        mean = amt_df[amt_df['Age']==age_interval].sample(sample_size, replace=Tru
        all_means[age_interval].append(mean)
```

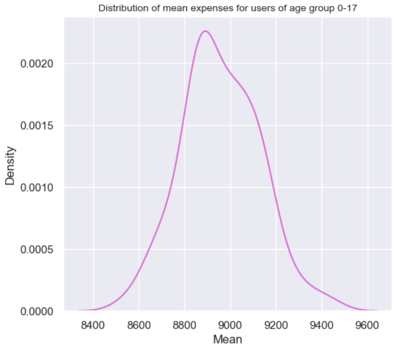
In [125…

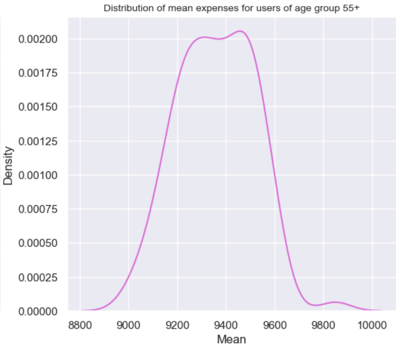
```
for val in ['26-35', '36-45', '18-25', '46-50', '51-55', '55+', '0-17']:

    new_df = amt_df[amt_df['Age']==val]

    margin_of_error_clt = 1.96*new_df['Purchase'].std()/np.sqrt(len(new_df))
    sample_mean = new_df['Purchase'].mean()
    lower_lim = sample_mean - margin_of_error_clt
    upper_lim = sample_mean + margin_of_error_clt

    print("For age {} --> confidence interval of means: ({:.2f}, {:.2f})".format(va
```

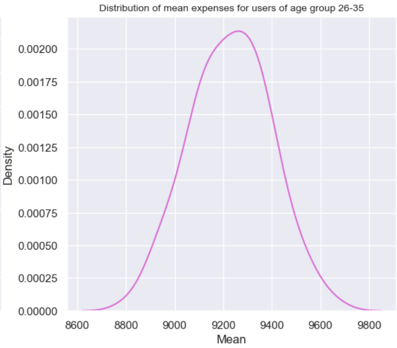
```
For age 26-35 --> confidence interval of means: (325226.35, 364561.66)
For age 36-45 --> confidence interval of means: (159958.40, 188563.04)
For age 18-25 --> confidence interval of means: (142318.86, 167933.62)
For age 46-50 --> confidence interval of means: (62258.26, 80618.47)
For age 51-55 --> confidence interval of means: (54450.95, 70179.72)
For age 55+ --> confidence interval of means: (28893.83, 39266.89)
For age 0-17 --> confidence interval of means: (18402.36, 27400.79)
```

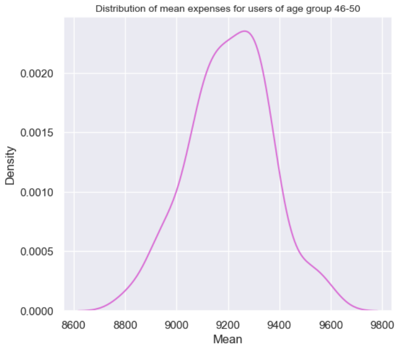
In [128…

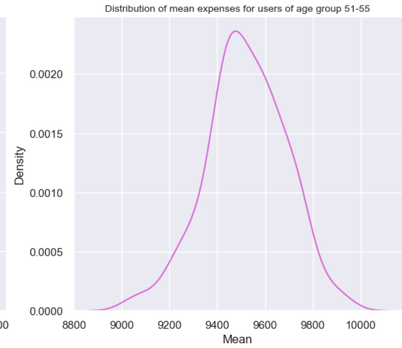
```
# Taking 100 samples of 1000 entries for each age group and
# Plotting KDE plots to see if their distribution looks gaussian
plt.figure(figsize=(20,18))
x = 1
for j in ['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25']:
    means = []
    for i in range(100):
        temp = df.loc[df['Age']==j,'Purchase'].sample(1000)
        avg = temp.mean()
        means.append(avg)
    plt.subplot(3,3,x)
    sns.kdeplot(x = means, color = 'orchid')
    if j == '0-17':
        means_0 = means
    elif j == '55+':
        means_55 = means
    elif j == '26-35':
        means_26 = means
    elif j == '46-50':
        means_46 = means
    elif j == '51-55':
        means_51 = means
    elif j == '36-45':
        means_36 = means
    else:
        means_18 = means
    plt.title('Distribution of mean expenses for users of age group {a}'.format(a
    plt.xlabel('Mean')
    x += 1
plt.show()
```

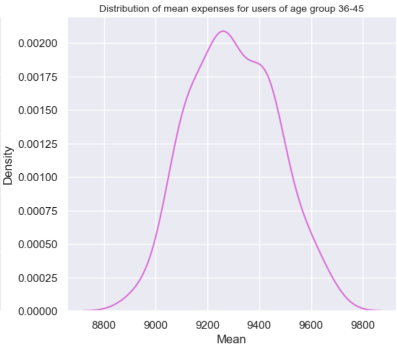


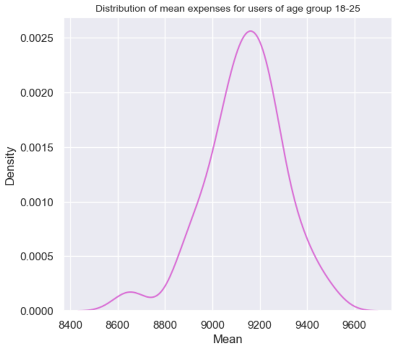












In [129…

```
# Finding confidence intervals for mean purchase for each age group
for i in ['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25']:
    print('For {m}-'.format(m = i))
    if i == '0-17':
        means = means_0
    elif i == '55+':
        means = means_55
    elif i == '26-35':
        means = means_26
    elif i == '46-50':
        means = means_46
    elif i == '51-55':
        means = means_51
    elif i == '36-45':
        means = means_36
    else:
        means = means_18

    print('Mean of sample means =',np.mean(means))
    print('Population mean =', np.mean(df.loc[df['Age']==i, 'Purchase']))
    print('Standard deviation of means (Standard Error) =', np.std(means))
    print('Standard deviation of population =',df.loc[df['Age']==i, 'Purchase'].st
    print('99% CONFIDENCE INTERVAL for mean expense by users of age group {a}-'.fo
    print((np.percentile(means, 0.5).round(2), np.percentile(means, 99.5).round(2)
    print('95% CONFIDENCE INTERVAL for mean expense by users of age group {a}-'.fo
    print((np.percentile(means, 2.5).round(2), np.percentile(means, 97.5).round(2)
    print('90% CONFIDENCE INTERVAL for mean expense by users of age group {a}-'.fo
    print((np.percentile(means, 5).round(2), np.percentile(means, 95).round(2)))
    print('-'*50)
```

```
For 0-17-
Mean of sample means = 8958.924439999999
Population mean = 8933.464640444974
Standard deviation of means (Standard Error) = 169.13667550092848
Standard deviation of population = 5111.11404600277
99% CONFIDENCE INTERVAL for mean expense by users of age group 0-17-
(8579.44, 9410.82)
95% CONFIDENCE INTERVAL for mean expense by users of age group 0-17-
(8646.73, 9299.8)
90% CONFIDENCE INTERVAL for mean expense by users of age group 0-17-
(8684.76, 9213.9)
---------------------------------------------------
For 55+-
Mean of sample means = 9359.08287
Population mean = 9336.280459449405
Standard deviation of means (Standard Error) = 156.81968408944425
Standard deviation of population = 5011.493995603418
99% CONFIDENCE INTERVAL for mean expense by users of age group 55+-
(9017.02, 9737.18)
95% CONFIDENCE INTERVAL for mean expense by users of age group 55+-
(9071.38, 9603.31)
90% CONFIDENCE INTERVAL for mean expense by users of age group 55+-
(9114.73, 9588.83)
---------------------------------------------------
For 26-35-
Mean of sample means = 9229.8788
Population mean = 9252.690632869888
Standard deviation of means (Standard Error) = 164.83426852017152
Standard deviation of population = 5010.527303002927
99% CONFIDENCE INTERVAL for mean expense by users of age group 26-35-
(8861.53, 9610.07)
95% CONFIDENCE INTERVAL for mean expense by users of age group 26-35-
(8922.04, 9547.84)
90% CONFIDENCE INTERVAL for mean expense by users of age group 26-35-
(8948.68, 9474.16)
---------------------------------------------------
For 46-50-
Mean of sample means = 9206.63968
Population mean = 9208.625697468327
Standard deviation of means (Standard Error) = 157.578319943378
Standard deviation of population = 4967.216367142921
99% CONFIDENCE INTERVAL for mean expense by users of age group 46-50-
(8818.86, 9580.86)
95% CONFIDENCE INTERVAL for mean expense by users of age group 46-50-
(8915.74, 9530.96)
90% CONFIDENCE INTERVAL for mean expense by users of age group 46-50-
(8939.3, 9479.92)
---------------------------------------------------
For 51-55-
Mean of sample means = 9520.912779999999
Population mean = 9534.808030960236
Standard deviation of means (Standard Error) = 164.29302717148894
Standard deviation of population = 5087.368079602116
99% CONFIDENCE INTERVAL for mean expense by users of age group 51-55-
(9070.64, 9905.2)
95% CONFIDENCE INTERVAL for mean expense by users of age group 51-55-
(9200.12, 9804.3)
90% CONFIDENCE INTERVAL for mean expense by users of age group 51-55-
(9229.91, 9752.7)
---------------------------------------------------
For 36-45-
Mean of sample means = 9298.4161
Population mean = 9331.350694917874
Standard deviation of means (Standard Error) = 163.24484124164536
```

```
Standard deviation of population = 5022.923879204652
99% CONFIDENCE INTERVAL for mean expense by users of age group 36-45-
(8937.98, 9667.0)
95% CONFIDENCE INTERVAL for mean expense by users of age group 36-45-
(9026.36, 9604.66)
90% CONFIDENCE INTERVAL for mean expense by users of age group 36-45-
(9065.3, 9583.67)
--------------------------------------------------
For 18-25-
Mean of sample means = 9137.0125
Population mean = 9169.663606261289
Standard deviation of means (Standard Error) = 163.2560818611975
Standard deviation of population = 5034.321997176577
99% CONFIDENCE INTERVAL for mean expense by users of age group 18-25-
(8636.44, 9496.4)
95% CONFIDENCE INTERVAL for mean expense by users of age group 18-25-
(8766.68, 9435.32)
90% CONFIDENCE INTERVAL for mean expense by users of age group 18-25-
(8864.59, 9378.44)
--------------------------------------------------
```

Observations:

1. 99% Confidence Interval for 0–17 is less than 51–55 without overlap.
2. We can say with 99% confidence that expense of 0–17 is less compared to expense of 51–55 ages.

Confidence Interval by Age

For age 26-35 --> confidence interval of means: (945034.42, 1034284.21) For age 36-45 --> confidence interval of means: (823347.80, 935983.62) For age 18-25 --> confidence interval of means: (801632.78, 908093.46) For age 46-50 --> confidence interval of means: (713505.63, 871591.93) For age 51-55 --> confidence interval of means: (692392.43, 834009.42) For age 55+ --> confidence interval of means: (476948.26, 602446.23) For age 0-17 --> confidence interval of means: (527662.46, 710073.17)

5. Overall Insights:

1. Walmart can keep products like P00265242 and P00025442 (which are selling a lot) in the inventory. Products like P00056342 P00350742 (which are not selling) need not be kept in store.
2. Ads can be targeted towards people of age group 26–35, since they are making maximum purchases. Walmart can also include new products required by people of this age group.
3. Ads can be targeted towards people of city category B. Inventory in these cities can be replenished.
4. Ads can be targeted towards people who have spent between 1 to 2 years in their cities.
5. Ads can be targeted towards unmarried people.
6. Products of categories 1, 5 and 8 can be kept in inventory as well as made easily visible in the stores.
7. Offers/rewards can be given on purchases above 12000 dollars to nudge customers to make more purchases.
8. More products popular among people with occupations 0, 4 and 7 can be kept in store.

9. Ads for slightly expensive products can be targetted towards people with occupation 12 and 17. (See median expenses of all occupations below)
10. Ads for products which cost between 9151 and 9790 can be targetted towards males.
11. Ads for products which cost between 8507 and 9051 can be targetted towards females.
12. Ads for products which cost between 9225 to 9908 can be targetted towards 51–55 year old customers.
13. Ads for products which cost between 8611 to 9235 can be targetted towards 0–17 year old customers.

6. Recommendations:

1. Walmart can give offers/rewards on purchases above 12000 to nudge customers to spend more.
2. Ads can be targeted towards people of city category B.
3. Ads should be targeted towards people who have spent between 1 to 2 years in their city.
4. Target ads towards unmarried people.
5. Target ads for products which cost between 9151 and 9790 towards males.
6. Target ads for products which cost between 8507 and 9051 towards females.
7. Target ads for products which cost between 9225 to 9908 towards 51–55 year old people.
8. Target ads for products which cost between 8611 to 9235 towards 0–17 year old people.

**** END OF PROJECT *****

In [ ]:

```

```