```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import t
import warnings
warnings.filterwarnings('ignore')
import copy
```

```
!gdown https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/293/original/walmart_d
```

```
Downloading...
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/293/original/walmart_d
To: /content/walmart_data.csv?1641285094
100% 23.0M/23.0M [00:00<00:00, 166MB/s]
```

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

```
df.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Mar
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	

```
df.tail()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years
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+

```
df.shape
```

```
(550068, 10)
```

```
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
```

🔍 Insights

- From the above analysis, it is clear that, data has total of 10 features with lots of mixed alpha numeric data.
- Apart from Purchase Column, all the other data types are of categorical type. We will change the datatypes of all such columns to category

🔁 Changing the Datatype of Columns

```
for i in df.columns[:-1]:
    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  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  category
  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  category
  9   Purchase                   550068 non-null  int64
 dtypes: category(9), int64(1)
 memory usage: 10.3 MB
```

📝 Statistical Summary

Satistical summary of object type columns

```
df.describe(include = 'category')
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Ye
count	550068	550068	550068	550068	550068	550068	550
unique	5891	3631	2	7	21	3	
top	1001680	P00265242	M	26-35	4	B	
freq	1026	1880	414259	219587	72308	231173	193

🔍 Insights

1. User_ID - Among 5,50,068 transactions there are 5891 unique user_id, indicating same customers buying multiple products.
2. Product_ID - Among 5,50,068 transactions there are 3631 unique products,with the product having the code P00265242 being the highest seller, with a maximum of 1,880 units sold.
3. Gender - Out of 5,50,068 transactions, 4,14,259 (nearly 75%) were done by male gender indicating a significant disparity in purchase behavior between males and females during the Black Friday event.
4. Age - We have 7 unique age groups in the dataset. 26 - 35 Age group has maximum of 2,19,587 transactions. We will analyse this feature in detail in future
5. Stay_In_Current_City_Years - Customers with 1 year of stay in current city accounted to maximum of 1,93,821 transactions among all the other customers with (0,2,3,4+) years of stay in current city
6. Marital_Status - 59% of the total transactions were done by Unmarried Customers and 41% by Married Customers.

Satistical summary of numerical data type columns

```
1 df.describe()
```

	Purchase
count	550068.000000
mean	9263.968713
std	5023.065394
min	12.000000
25%	5823.000000
50%	8047.000000
75%	12054.000000
max	23961.000000

🔍 Insights

The purchase amounts vary widely, with the minimum recorded purchase being $12andthemaximumreaching$23961. The median purchase amount of $8047isnotablylowerthanthemeanpurchaseamountof$9264, indicating a right-skewed distribution where a few high-value purchases pull up the mean

👥 Duplicate Detection

```
df.duplicated().value_counts()
```

```
False    550068
dtype: int64
```

🔍 Insights

- There are no duplicate entries in the dataset

✅ Sanity Check for columns

```
# checking the unique values for columns
for i in df.columns:
    print('Unique Values in',i,'column are :-')
    print(df[i].unique())
    print('-'*70)
```

```
Unique Values in User_ID column are :-
[1000001, 1000002, 1000003, 1000004, 1000005, ..., 1004588, 1004871, 1004113, 1005391, 1001529]
Length: 5891
Categories (5891, int64): [1000001, 1000002, 1000003, 1000004, ..., 1006037, 1006038, 1006039,
----------------------------------------------------------------------
Unique Values in Product_ID column are :-
['P00069042', 'P00248942', 'P00087842', 'P00085442', 'P00285442', ..., 'P00375436', 'P00372445'
Length: 3631
Categories (3631, object): ['P00000142', 'P00000242', 'P00000342', 'P00000442', ..., 'P0099642'
                            'P0099742', 'P0099842', 'P0099942']
----------------------------------------------------------------------
Unique Values in Gender column are :-
['F', 'M']
Categories (2, object): ['F', 'M']
----------------------------------------------------------------------
Unique Values in Age column are :-
['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+']
----------------------------------------------------------------------
Unique Values in Occupation column are :-
[10, 16, 15, 7, 20, ..., 18, 5, 14, 13, 6]
Length: 21
Categories (21, int64): [0, 1, 2, 3, ..., 17, 18, 19, 20]
----------------------------------------------------------------------
Unique Values in City_Category column are :-
['A', 'C', 'B']
Categories (3, object): ['A', 'B', 'C']
----------------------------------------------------------------------
Unique Values in Stay_In_Current_City_Years column are :-
['2', '4+', '3', '1', '0']
```

```
Categories (5, object): ['0', '1', '2', '3', '4+']
----------------------------------------------------------------------
Unique Values in Marital_Status column are :-
[0, 1]
Categories (2, int64): [0, 1]
----------------------------------------------------------------------
Unique Values in Product_Category column are :-
[3, 1, 12, 8, 5, ..., 10, 17, 9, 20, 19]
Length: 20
Categories (20, int64): [1, 2, 3, 4, ..., 17, 18, 19, 20]
----------------------------------------------------------------------
Unique Values in Purchase column are :-
[ 8370 15200  1422 ...   135   123   613]
----------------------------------------------------------------------
```

🔍 Insights

- The dataset does not contain any abnormal values.
- We will convert the 0,1 in Marital Status column as married and unmarried

```
#replacing the values in marital_status column

df['Marital_Status'] = df['Marital_Status'].replace({0:'Unmarried',1:'Married'})
df['Marital_Status'].unique()
```

```
['Unmarried', 'Married']
Categories (2, object): ['Unmarried', 'Married']
```

🧑🏽‍💻 Missing Value Analysis

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

```
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
```

🔍 Insights

- The dataset does not contain any missing values.

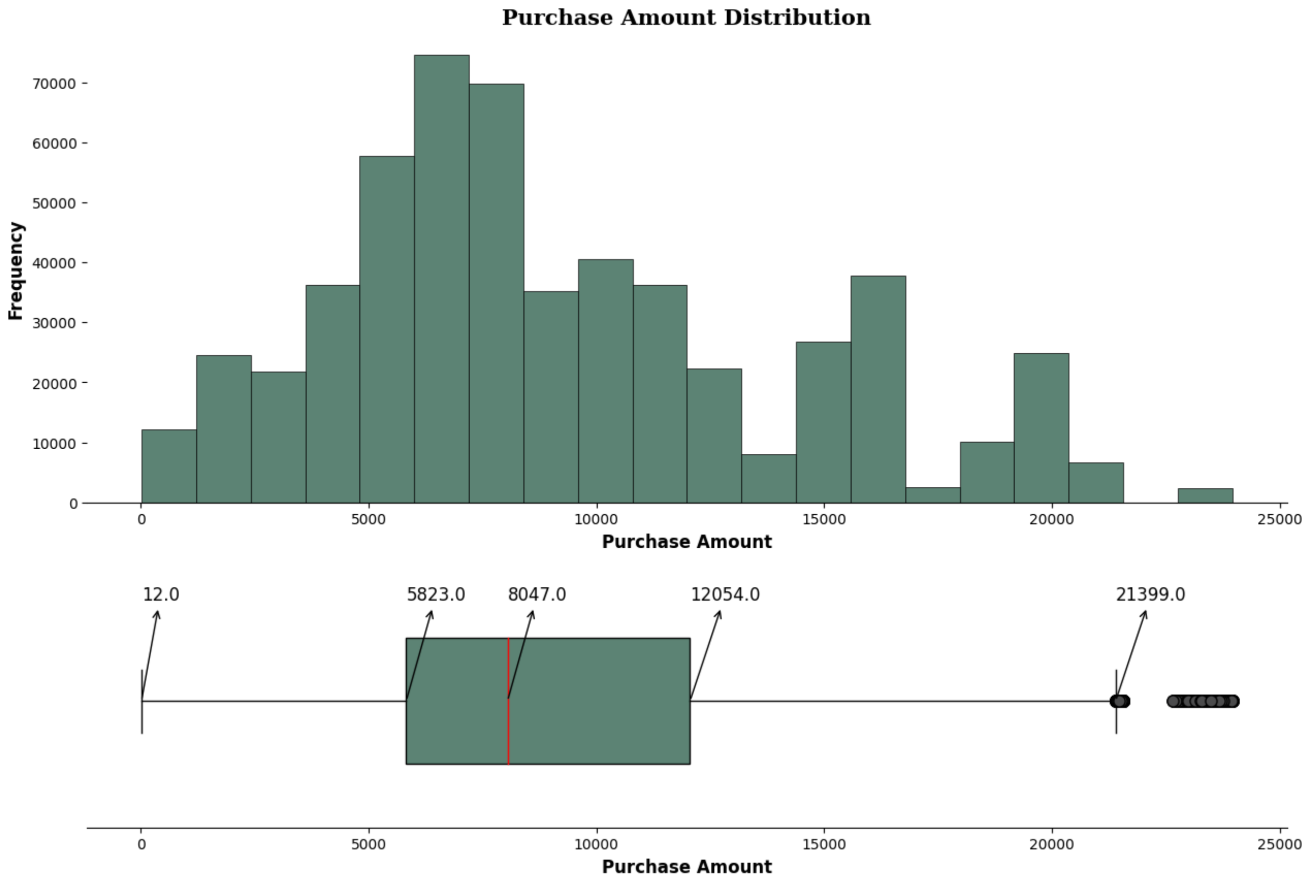
Univariate Analysis

Numerical Variables

💰 Purchase Amount Distribution

```
#setting the plot style

fig = plt.figure(figsize = (15,10))
gs = fig.add_gridspec(2,1,height_ratios=[0.65, 0.35])

                                    #creating purchase amount histogram

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

ax0.hist(df['Purchase'],color= '#5C8374',linewidth=0.5,edgecolor='black',bins = 20)
ax0.set_xlabel('Purchase Amount',fontsize = 12,fontweight = 'bold')
ax0.set_ylabel('Frequency',fontsize = 12,fontweight = 'bold')

#removing the axis lines
for s in ['top','left','right']:
    ax0.spines[s].set_visible(False)

#setting title for visual
ax0.set_title('Purchase Amount Distribution',{'font':'serif', 'size':15,'weight':'bold'})


                                     #creating box plot for purchase amount

ax1 = fig.add_subplot(gs[1,0])
boxplot = ax1.boxplot(x = df['Purchase'],vert = False,patch_artist = True,widths = 0.5)

# Customize box and whisker colors
boxplot['boxes'][0].set(facecolor='#5C8374')

# Customize median line
boxplot['medians'][0].set(color='red')

# Customize outlier markers
for flier in boxplot['fliers']:
    flier.set(marker='o', markersize=8, markerfacecolor= "#4b4b4c")

#removing the axis lines
for s in ['top','left','right']:
    ax1.spines[s].set_visible(False)

#adding 5 point summary annotations
info = [i.get_xdata() for i in boxplot['whiskers']] #getting the upperlimit,Q1,Q3 and lowerlimit

median = df['Purchase'].quantile(0.5) #getting Q2

for i,j in info: #using i,j here because of the output type of info list comprehension

    ax1.annotate(text = f"{i:.1f}", xy = (i,1), xytext = (i,1.4),fontsize = 12,
                 arrowprops= dict(arrowstyle="<-", lw=1, connectionstyle="arc,rad=0"))

    ax1.annotate(text = f"{j:.1f}", xy = (j,1), xytext = (j,1.4),fontsize = 12,
                 arrowprops= dict(arrowstyle="<-", lw=1, connectionstyle="arc,rad=0"))

#adding the median separately because it was included in info list
ax1.annotate(text = f"{median:.1f}",xy = (median,1),xytext = (median + 1,1.4),fontsize = 12,
            arrowprops= dict(arrowstyle="<-", lw=1, connectionstyle="arc,rad=0"))

#removing y-axis ticks
ax1.set_yticks([])

#adding axis label
```

```
ax1.set_xlabel('Purchase Amount',fontweight = 'bold',fontsize = 12)

plt.show()
```



Calculating the Number of Outliers

- As seen above, Purchase amount over 21399 is considered as outlier. We will count the number of outliers as below

```
len(df.loc[df['Purchase'] > 21399,'Purchase'])
```

2677

🔍 Insights

Outliers

- There are total of 2677 outliers which is roughly 0.48% of the total data present in purchase amount. We will not remove them as it indicates a broad range of spending behaviors during the sale, highlighting the importance of tailoring marketing strategies to both regular and high-value customers to maximize revenue.

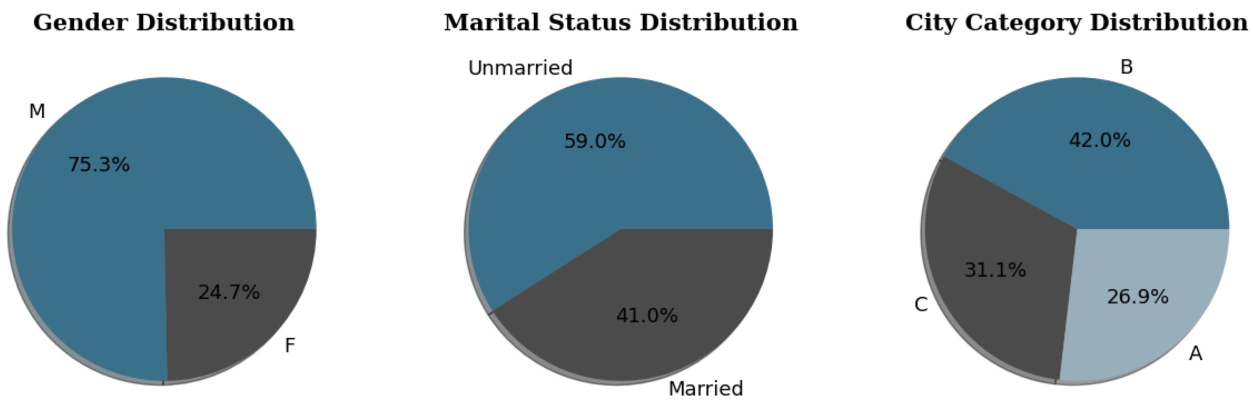
Distribution

- Data suggests that the majority of customers spent between 5,823 USD and 12,054 USD, with the median purchase amount being 8,047 USD.
- The lower limit of 12 USD while the upper limit of 21,399 USD reveal significant variability in customer spending

Categorical Variables

🙍‍♀️ 🙍‍♂️ Gender, 👫 Marital Status and 🌎 City Category Distribution

```
#setting the plot style
fig = plt.figure(figsize = (15,12))
gs = fig.add_gridspec(1,3)

                                        # creating pie chart for gender disribution
ax0 = fig.add_subplot(gs[0,0])

color_map = ["#3A7089", "#4b4b4c"]
ax0.pie(df['Gender'].value_counts().values,labels = df['Gender'].value_counts().index,autopct =
        shadow = True,colors = color_map,textprops={'fontsize': 13, 'color': 'black'})

#setting title for visual
ax0.set_title('Gender Distribution',{'font':'serif', 'size':15,'weight':'bold'})

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

color_map = ["#3A7089", "#4b4b4c"]
ax1.pie(df['Marital_Status'].value_counts().values,labels = df['Marital_Status'].value_counts().
        shadow = True,colors = color_map,textprops={'fontsize': 13, 'color': 'black'})

#setting title for visual
ax1.set_title('Marital Status Distribution',{'font':'serif', 'size':15,'weight':'bold'})

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

color_map = ["#3A7089", "#4b4b4c",'#99AEBB']
ax1.pie(df['City_Category'].value_counts().values,labels = df['City_Category'].value_counts().in
        shadow = True,colors = color_map,textprops={'fontsize': 13, 'color': 'black'})

#setting title for visual
ax1.set_title('City Category Distribution',{'font':'serif', 'size':15,'weight':'bold'})
plt.show()
```



🔍 Insights

1. Gender Distribution - Data indicates a significant disparity in purchase behavior between males and females during the Black Friday event.
2. Marital Status - Given that unmarried customers account for a higher percentage of transactions, it may be worthwhile to consider specific marketing campaigns or promotions that appeal to this group.
3. City Category - City B saw the most number of transactions followed by City C and City A respectively

Confidence Interval Construction: Estimating Average Purchase Amount per Transaction

1. **Step 1 - Building CLT Curve**

As seen above, the purchase amount distribution is not Normal. So we need to use Central Limit Theorem. It states the distribution of sample means will approximate a normal distribution, regardless of the underlying population distribution 2.** Step 2 - Building Confidence Interval**

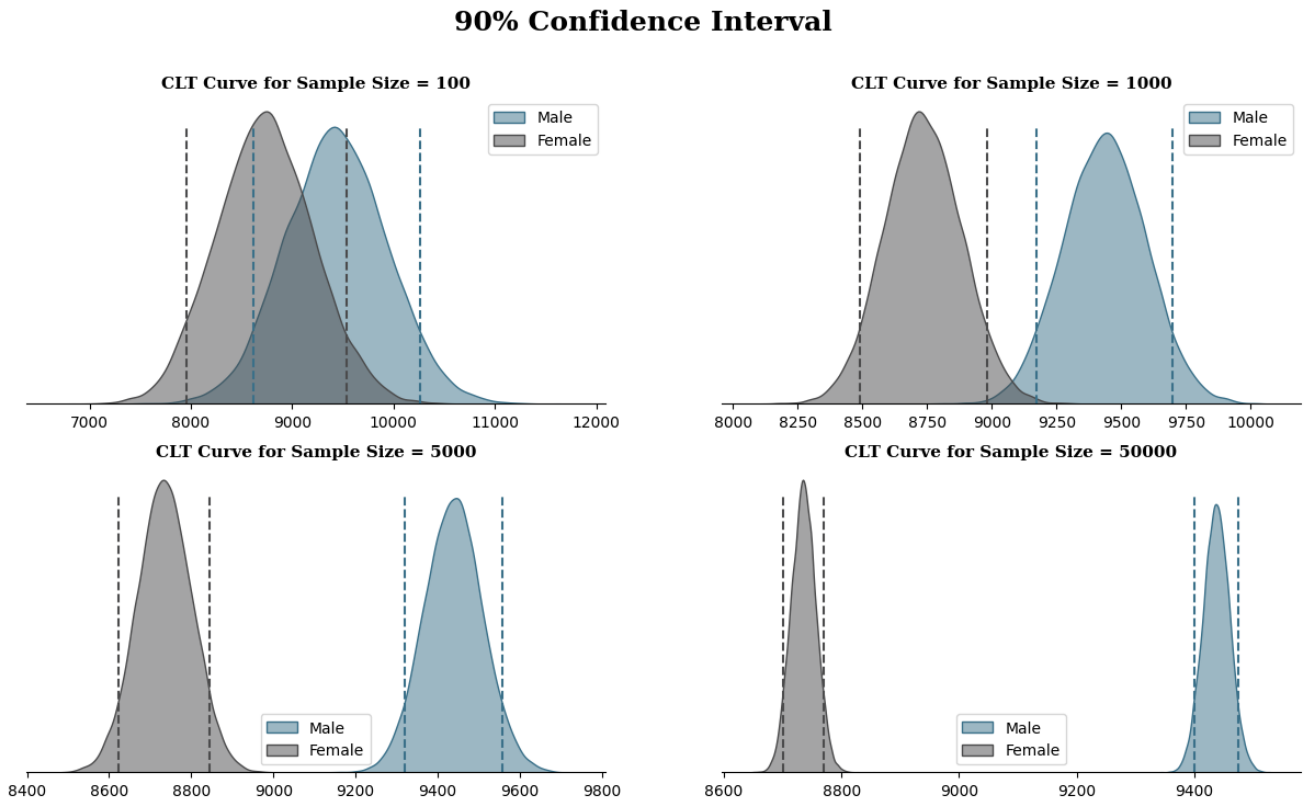
After building CLT curve, we will create a confidence interval predicting population mean at 99%,95% and 90% Confidence level.

```
#creating a function to calculate confidence interval

def confidence_interval(data,ci):
    #converting the list to series
    l_ci = (100-ci)/2
    u_ci = (100+ci)/2

    #calculating lower limit and upper limit of confidence interval
    interval = np.percentile(data,[l_ci,u_ci]).round(0)

    return interval
```

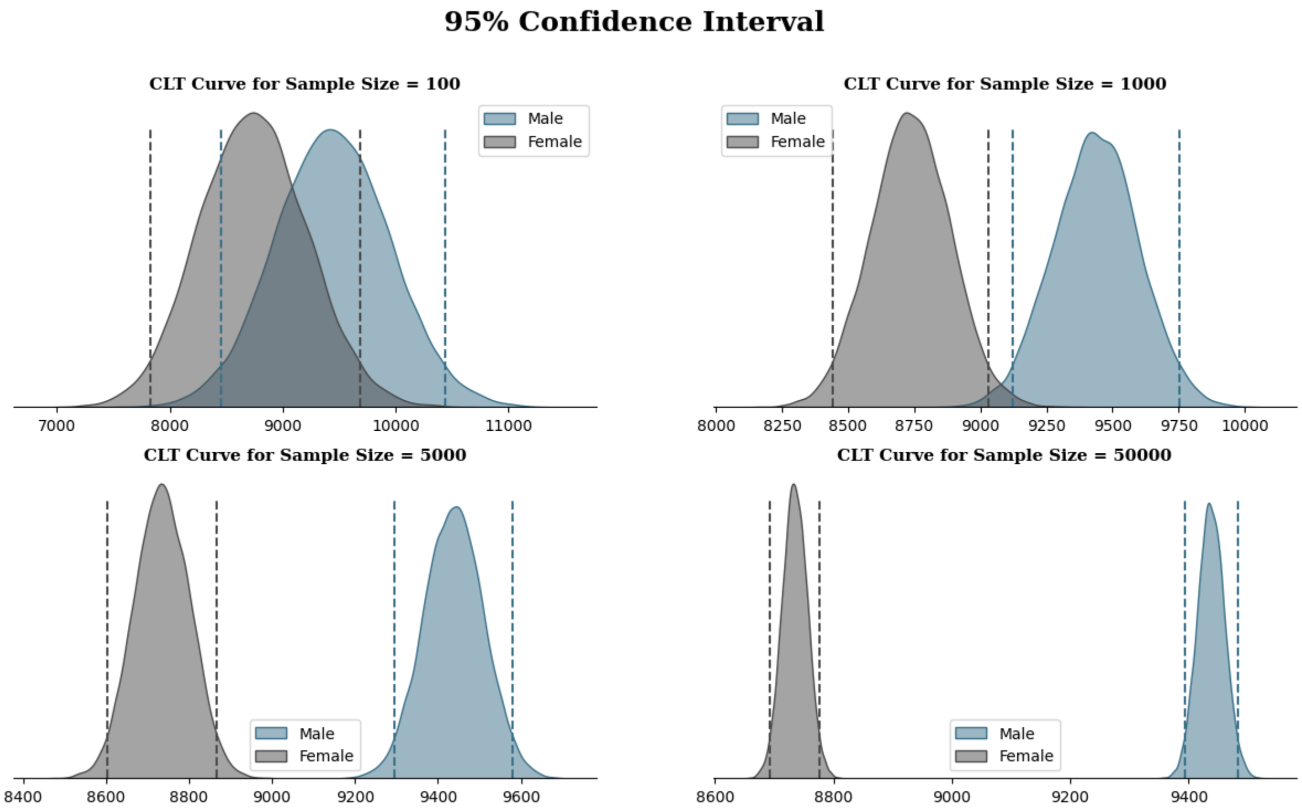
```
#defining a function for plotting the visual for given confidence interval

def plot(ci):

    #setting the plot style
    fig = plt.figure(figsize = (15,8))
    gs = fig.add_gridspec(2,2)

    #creating separate data frames for each gender
    df_male = df.loc[df['Gender'] == 'M','Purchase']
    df_female = df.loc[df['Gender'] == 'F','Purchase']

    #sample sizes and corresponding plot positions
    sample_sizes = [(100,0,0),(1000,0,1),(5000,1,0),(50000,1,1)]

    #number of samples to be taken from purchase amount
    bootstrap_samples = 20000

    male_samples = {}
    female_samples = {}

    for i,x,y in sample_sizes:
        male_means = [] #list for collecting the means of male sample
        female_means = [] #list for collecting the means of female sample

        for j in range(bootstrap_samples):

            #creating random 5000 samples of i sample size
            male_bootstrapped_samples = np.random.choice(df_male,size = i)
            female_bootstrapped_samples = np.random.choice(df_female,size = i)

            #calculating mean of those samples
            male_sample_mean = np.mean(male_bootstrapped_samples)
            female_sample_mean = np.mean(female_bootstrapped_samples)

            #appending the mean to the list
            male_means.append(male_sample_mean)
            female_means.append(female_sample_mean)

        #storing the above sample generated
        male_samples[f'{ci}%_{i}'] = male_means
        female_samples[f'{ci}%_{i}'] = female_means

        #creating a temporary dataframe for creating kdeplot
        temp_df = pd.DataFrame(data = {'male_means':male_means,'female_means':female_means})

                                                        #plotting kdeplots
        #plot position
        ax = fig.add_subplot(gs[x,y])

        #plots for male and female
        sns.kdeplot(data = temp_df,x = 'male_means',color ="#3A7089" ,fill = True, alpha = 0.5,a
        sns.kdeplot(data = temp_df,x = 'female_means',color ="#4b4b4c" ,fill = True, alpha = 0.5

        #calculating confidence intervals for given confidence level(ci)
        m_range = confidence_interval(male_means,ci)
        f_range = confidence_interval(female_means,ci)

        #plotting confidence interval on the distribution
        for k in m_range:
            ax.axvline(x = k,ymax = 0.9, color ="#3A7089",linestyle = '--')
```

```

        for k in f_range:
            ax.axvline(x = k,ymax = 0.9, color ="#4b4b4c",linestyle = '--')


        #removing the axis lines
        for s in ['top','left','right']:
            ax.spines[s].set_visible(False)

        # adjusting axis labels
        ax.set_yticks([])
        ax.set_ylabel('')
        ax.set_xlabel('')

        #setting title for visual
        ax.set_title(f'CLT Curve for Sample Size = {i}',{'font':'serif', 'size':11,'weight':'bol

        plt.legend()

    #setting title for visual
    fig.suptitle(f'{ci}% Confidence Interval',font = 'serif', size = 18, weight = 'bold')

    plt.show()

    return male_samples,female_samples
```

```
m_samp_90,f_samp_90 = plot(90)
```




```
1 m_samp_95,f_samp_95 = plot(95)
```



🔍 Insights

1. Sample Size

The analysis highlights the importance of sample size in estimating population parameters. It suggests that as the sample size increases, the confidence intervals become narrower and more precise. In business, this implies that larger sample sizes can provide more reliable insights and estimates.

2. Confidence Intervals

From the above analysis, we can see that except for the Sample Size of 100, the confidence interval do not overlap as the sample size increases. This means that there is a statistically significant difference between the average spending per transaction for men and women within the given samples.

3. Population Average

We are 95% confident that the true population average for males falls between $9,393 and$9,483, and for females, it falls between $8,692 and$8,777.

4. Women spend less

Men tend to spend more money per transaction on average than women, as the upper bounds of the confidence intervals for men are consistently higher than those for women across different sample sizes.

👫 Marital Status VS 💰 Purchase Amount

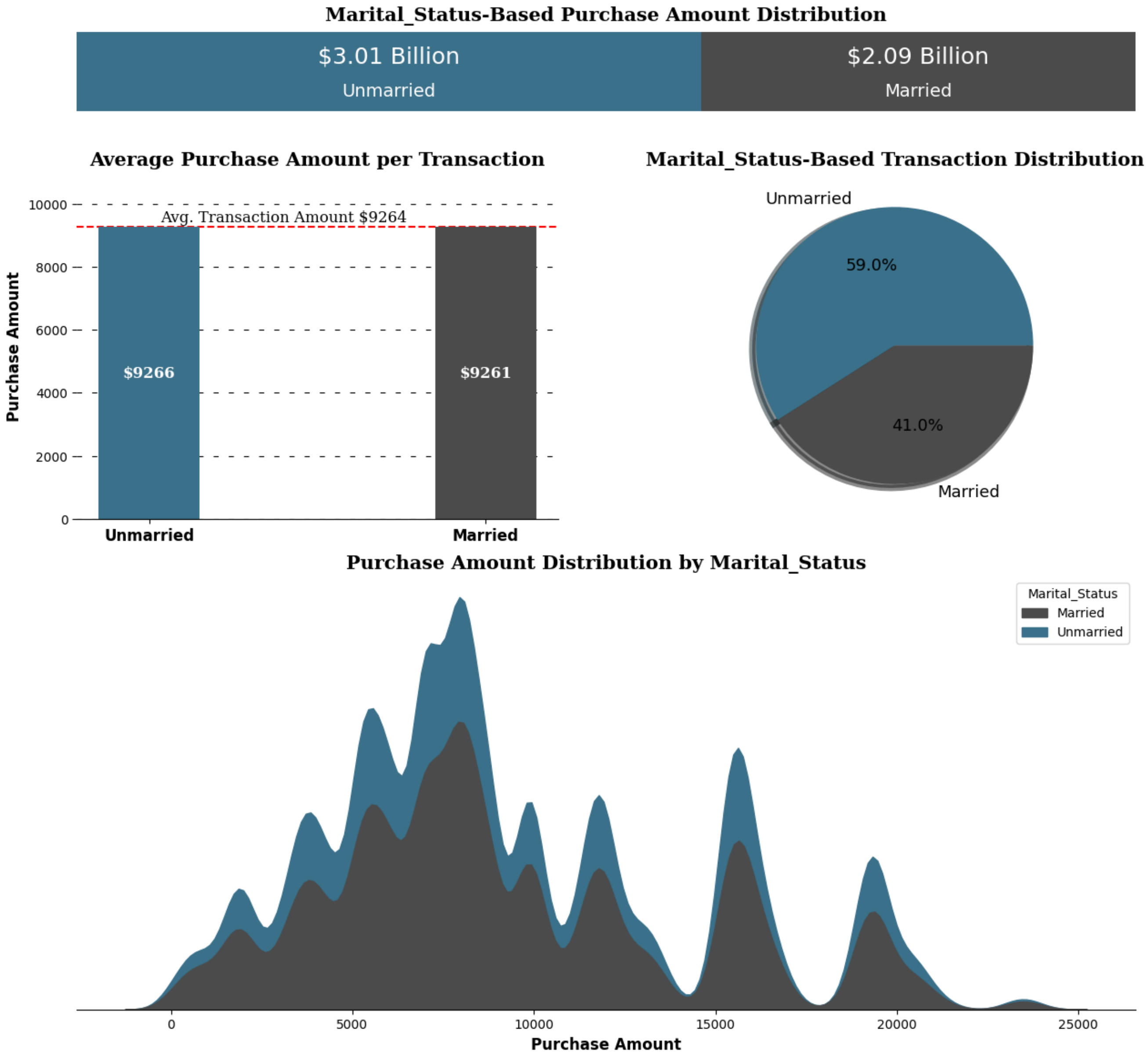
📊 Data Visualization

```
#creating a df for purchase amount vs marital status
temp = df.groupby('Marital_Status')['Purchase'].agg(['sum','count']).reset_index()

#calculating the amount in billions
temp['sum_in_billions'] = round(temp['sum'] / 10**9,2)

#calculationg percentage distribution of purchase amount
temp['%sum'] = round(temp['sum']/temp['sum'].sum(),3)

#calculationg per purchase amount
temp['per_purchase'] = round(temp['sum']/temp['count'])

temp
```

	Marital_Status	sum	count	sum_in_billions	%sum	per_purchase
0	Unmarried	3008927447	324731	3.01	0.59	9266.0
1	Married	2086885295	225337	2.09	0.41	9261.0

```
#setting the plot style
fig = plt.figure(figsize = (15,14))
gs = fig.add_gridspec(3,2,height_ratios =[0.10,0.4,0.5])


                                        #Distribution of Purchase Amount
ax = fig.add_subplot(gs[0,:])

#plotting the visual
ax.barh(temp.loc[0,'Marital_Status'],width = temp.loc[0,'%sum'],color = "#3A7089",label = 'Unmar
ax.barh(temp.loc[0,'Marital_Status'],width = temp.loc[1,'%sum'],left =temp.loc[0,'%sum'], color

#inserting the text
txt = [0.0] #for left parameter in ax.text()

for i in temp.index:
    #for amount
    ax.text(temp.loc[i,'%sum']/2 + txt[0],0.15,f"${temp.loc[i,'sum_in_billions']} Billion",
           va = 'center', ha='center',fontsize=18, color='white')

    #for marital status
    ax.text(temp.loc[i,'%sum']/2 + txt[0],- 0.20 ,f"{temp.loc[i,'Marital_Status']}",
           va = 'center', ha='center',fontsize=14, color='white')

    txt += temp.loc[i,'%sum']

#removing the axis lines
for s in ['top','left','right','bottom']:
    ax.spines[s].set_visible(False)

#customizing ticks
ax.set_xticks([])
ax.set_yticks([])
ax.set_xlim(0,1)

#plot title
ax.set_title('Marital_Status-Based Purchase Amount Distribution',{'font':'serif', 'size':15,'wei


                                            #Distribution of Purchase Amount per Transaction

ax1 = fig.add_subplot(gs[1,0])

color_map = ["#3A7089", "#4b4b4c"]

#plotting the visual
ax1.bar(temp['Marital_Status'],temp['per_purchase'],color = color_map,zorder = 2,width = 0.3)

#adding average transaction line
avg = round(df['Purchase'].mean())

ax1.axhline(y = avg, color ='red', zorder = 0,linestyle = '--')

#adding text for the line
ax1.text(0.4,avg + 300, f"Avg. Transaction Amount ${avg:.0f}",
         {'font':'serif','size' : 12},ha = 'center',va = 'center')

#adjusting the ylimits
ax1.set_ylim(0,11000)

#adding the value_counts
```

```
for i in temp.index:
    ax1.text(temp.loc[i,'Marital_Status'],temp.loc[i,'per_purchase']/2,f"${temp.loc[i,'per_purch
             {'font':'serif','size' : 12,'color':'white','weight':'bold' },ha = 'center',va = 'c

#adding grid lines
ax1.grid(color = 'black',linestyle = '--', axis = 'y', zorder = 0, dashes = (5,10))

#removing the axis lines
for s in ['top','left','right']:
    ax1.spines[s].set_visible(False)

#adding axis label
ax1.set_ylabel('Purchase Amount',fontweight = 'bold',fontsize = 12)
ax1.set_xticklabels(temp['Marital_Status'],fontweight = 'bold',fontsize = 12)

#setting title for visual
ax1.set_title('Average Purchase Amount per Transaction',{'font':'serif', 'size':15,'weight':'bol

                                        # creating pie chart for Marital_Status disribution
ax2 = fig.add_subplot(gs[1,1])

color_map = ["#3A7089", "#4b4b4c"]
ax2.pie(temp['count'],labels = temp['Marital_Status'],autopct = '%.1f%%',
        shadow = True,colors = color_map,wedgeprops = {'linewidth': 5},textprops={'fontsize': 13

#setting title for visual
ax2.set_title('Marital_Status-Based Transaction Distribution',{'font':'serif', 'size':15,'weight

                                        # creating kdeplot for purchase amount distribution

ax3 = fig.add_subplot(gs[2,:])
color_map = [ "#4b4b4c","#3A7089"]

#plotting the kdeplot
sns.kdeplot(data = df, x = 'Purchase', hue = 'Marital_Status', palette = color_map,fill = True,
            ax = ax3,hue_order = ['Married','Unmarried'])

#removing the axis lines
for s in ['top','left','right']:
    ax3.spines[s].set_visible(False)

# adjusting axis labels
ax3.set_yticks([])
ax3.set_ylabel('')
ax3.set_xlabel('Purchase Amount',fontweight = 'bold',fontsize = 12)

#setting title for visual
ax3.set_title('Purchase Amount Distribution by Marital_Status',{'font':'serif', 'size':15,'weigh

plt.show()
```



🔍 Insights

1. Total Sales and Transactions Comparison

The total purchase amount and number of transactions by Unmarried customers was more than 20% the amount and transactions by married customers indicating that they had a more significant impact on the Black Friday sales.

2. Average Transaction Value

The average purchase amount per transaction was almost similar for married and unmarried customers ($9261 vs$9266).

3. Distribution of Purchase Amount

As seen above, the purchase amount for both married and unmarried customers is not normally distributed

Confidence Interval Construction: Estimating Average Purchase Amount per Transaction

1. Step 1 - Building CLT Curve

As seen above, the purchase amount distribution is not Normal. So we need to use Central Limit Theorem. It states the distribution of sample means will approximate a normal distribution, regardless of the underlying population distribution

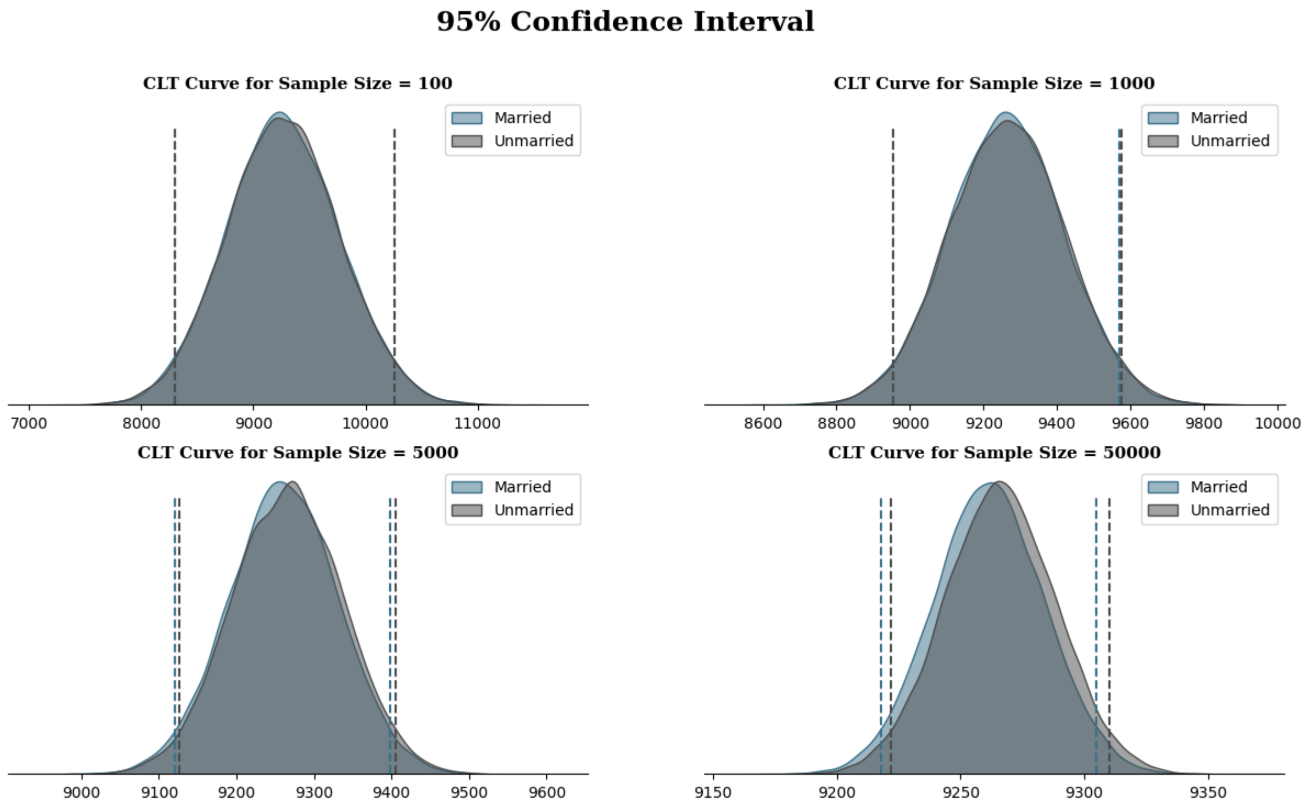
2. Step 2 - Building Confidence Interval

After building CLT curve, we will create a confidence interval predicting population mean at 95% Confidence level. Note - We will use different sample sizes of [100,1000,5000,50000]

```
#defining a function for plotting the visual for given confidence interval

def plot(ci):

    #setting the plot style
    fig = plt.figure(figsize = (15,8))
    gs = fig.add_gridspec(2,2)

    #creating separate data frames
    df_married = df.loc[df['Marital_Status'] == 'Married','Purchase']
    df_unmarried = df.loc[df['Marital_Status'] == 'Unmarried','Purchase']

    #sample sizes and corresponding plot positions
    sample_sizes = [(100,0,0),(1000,0,1),(5000,1,0),(50000,1,1)]

    #number of samples to be taken from purchase amount
    bootstrap_samples = 20000

    married_samples = {}
    unmarried_samples = {}

    for i,x,y in sample_sizes:
        married_means = [] #list for collecting the means of married sample
        unmarried_means = [] #list for collecting the means of unmarried sample

        for j in range(bootstrap_samples):

            #creating random 5000 samples of i sample size
            married_bootstrapped_samples = np.random.choice(df_married,size = i)
            unmarried_bootstrapped_samples = np.random.choice(df_unmarried,size = i)

            #calculating mean of those samples
            married_sample_mean = np.mean(married_bootstrapped_samples)
            unmarried_sample_mean = np.mean(unmarried_bootstrapped_samples)

            #appending the mean to the list
            married_means.append(married_sample_mean)
            unmarried_means.append(unmarried_sample_mean)

        #storing the above sample generated
        married_samples[f'{ci}%_{i}'] = married_means
        unmarried_samples[f'{ci}%_{i}'] = unmarried_means

        #creating a temporary dataframe for creating kdeplot
        temp_df = pd.DataFrame(data = {'married_means':married_means,'unmarried_means':unmarried

                                                        #plotting kdeplots
        #plot position
        ax = fig.add_subplot(gs[x,y])

        #plots for married and unmarried
        sns.kdeplot(data = temp_df,x = 'married_means',color ="#3A7089" ,fill = True, alpha = 0.
        sns.kdeplot(data = temp_df,x = 'unmarried_means',color ="#4b4b4c" ,fill = True, alpha =

        #calculating confidence intervals for given confidence level(ci)
        m_range = confidence_interval(married_means,ci)
        u_range = confidence_interval(unmarried_means,ci)

        #plotting confidence interval on the distribution
        for k in m_range:
            ax.axvline(x = k,ymax = 0.9, color ="#3A7089",linestyle = '--')
```

```
        for k in u_range:
            ax.axvline(x = k,ymax = 0.9, color ="#4b4b4c",linestyle = '--')


        #removing the axis lines
        for s in ['top','left','right']:
            ax.spines[s].set_visible(False)

        # adjusting axis labels
        ax.set_yticks([])
        ax.set_ylabel('')
        ax.set_xlabel('')

        #setting title for visual
        ax.set_title(f'CLT Curve for Sample Size = {i}',{'font':'serif', 'size':11,'weight':'bol

        plt.legend()

    #setting title for visual
    fig.suptitle(f'{ci}% Confidence Interval',font = 'serif', size = 18, weight = 'bold')

    plt.show()

    return married_samples,unmarried_samples
```

```
m_samp_95,u_samp_95 = plot(95)
```



🔍 Insights

1. Sample Size

The analysis highlights the importance of sample size in estimating population parameters. It suggests that as the sample size increases, the confidence intervals become narrower and more precise. In business, this implies that larger sample sizes can provide more reliable insights and estimates.

2. Confidence Intervals

From the above analysis, we can see that the confidence interval overlap for all the sample sizes. This means that there is no statistically significant difference between the average spending per transaction for married and unmarried customers within the given samples.

3. Population Average

We are 95% confident that the true population average for married customers falls between $9,217and$ 9,305, and for unmarried customers, it falls between $9,222and$9,311.

4. Both the customers spend equal

The overlapping confidence intervals of average spending for married and unmarried customers indicate that both married and unmarried customers spend a similar amount per transaction. This implies a resemblance in spending behavior between the two groups

📅 Customer Age VS 💰 Purchase Amount.

📊 Data Visualization

```
#creating a df for purchase amount vs age group
temp = df.groupby('Age')['Purchase'].agg(['sum','count']).reset_index()

#calculating the amount in billions
temp['sum_in_billions'] = round(temp['sum'] / 10**9,2)

#calculationg percentage distribution of purchase amount
temp['%sum'] = round(temp['sum']/temp['sum'].sum(),3)

#calculationg per purchase amount
temp['per_purchase'] = round(temp['sum']/temp['count'])

temp
```

	Age	sum	count	sum_in_billions	%sum	per_purchase
0	0-17	134913183	15102	0.13	0.026	8933.0
1	18-25	913848675	99660	0.91	0.179	9170.0
2	26-35	2031770578	219587	2.03	0.399	9253.0
3	36-45	1026569884	110013	1.03	0.201	9331.0
4	46-50	420843403	45701	0.42	0.083	9209.0
5	51-55	367099644	38501	0.37	0.072	9535.0
6	55+	200767375	21504	0.20	0.039	9336.0

```
#setting the plot style
fig = plt.figure(figsize = (20,14))
gs = fig.add_gridspec(3,1,height_ratios =[0.10,0.4,0.5])
```