INDEX

- Observing the Data given
- Detecting Null values and outliers
- Handling the outliers in Purchase
- Data Exploration
- Analysing the relation between Gender and Purchase
- Analysing the relation between Marital Status and Purchase
- Analysing the relation between Age group and Purchase

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import poisson, binom, expon, geom
from scipy.stats import norm
```

```
df=pd.read_csv("walmart_data.csv")
df
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	P
0	1000001	P00069042	F	0-17	10	A	2	0	3	
1	1000001	P00248942	F	0-17	10	A	2	0	1	
2	1000001	P00087842	F	0-17	10	A	2	0	12	
3	1000001	P00085442	F	0-17	10	A	2	0	12	
4	1000002	P00285442	M	55+	16	C	4+	0	8	
...	...	...	...	...	...	...	...	...	...	
550063	1006033	P00372445	M	51-55	13	B	1	1	20	
550064	1006035	P00375436	F	26-35	1	C	3	0	20	

Observing the Data given

```
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
```

```
df.describe()
```

	User_ID	Occupation	Marital_Status	Product_Category	Purchase
count	5.500680e+05	550068.000000	550068.000000	550068.000000	550068.000000
mean	1.003029e+06	8.076707	0.409653	5.404270	9263.968713
std	1.727592e+03	6.522660	0.491770	3.936211	5023.065394
min	1.000001e+06	0.000000	0.000000	1.000000	12.000000
25%	1.001516e+06	2.000000	0.000000	1.000000	5823.000000
50%	1.003077e+06	7.000000	0.000000	5.000000	8047.000000
75%	1.004478e+06	14.000000	1.000000	8.000000	12054.000000
max	1.006040e+06	20.000000	1.000000	20.000000	23961.000000

```
df.dtypes
```

	0
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

- table consist of data types "object" and "int"

Detecting Null values and outliers

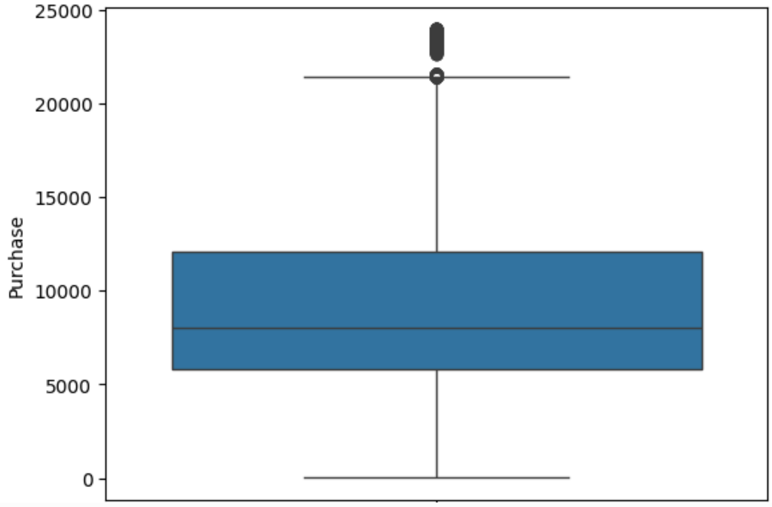
```
# Checking the missing values in the dataset
df.loc[pd.isna(df["User_ID"])].count()
```

	0
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

- No null values in the table

Handling the outliers in Purchase

```
sns.boxplot(y=df["Purchase"])
plt.show()
```




```
df_clippedPurchase = df[["User_ID"  ,"Product_ID"   ,"Gender","Age" ,"Occupation","Purchase","Marital_Status"]].copy()

# Clip the 'Purchase' values at 5th and 95th percentiles
lower_bound = df['Purchase'].quantile(0.05)
upper_bound = df['Purchase'].quantile(0.95)

df_clippedPurchase["Purchase"] = np.clip(df_clippedPurchase["Purchase"], a_min=lower_bound, a_max=upper_bound)

# Output the dataframe
df_clippedPurchase
```

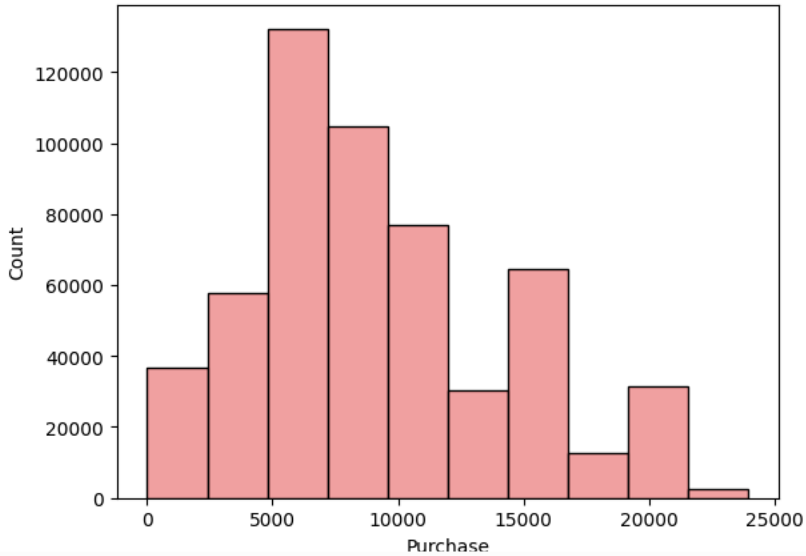
	User_ID	Product_ID	Gender	Age	Occupation	Purchase	Marital_Status
0	1000001	P00069042	F	0-17	10	8370	0
1	1000001	P00248942	F	0-17	10	15200	0
2	1000001	P00087842	F	0-17	10	1984	0
3	1000001	P00085442	F	0-17	10	1984	0
4	1000002	P00285442	M	55+	16	7969	0
...	...	...	...	...	...	...	...
550063	1006033	P00372445	M	51-55	13	1984	1
550064	1006035	P00375436	F	26-35	1	1984	0
550065	1006036	P00375436	F	26-35	15	1984	1
550066	1006038	P00375436	F	55+	1	1984	0
550067	1006039	P00371644	F	46-50	0	1984	1

550068 rows × 7 columns

Data Exploration

Analysing which purchase amount is most prevalent

```
sns.histplot(df["Purchase"], bins=10, color='lightcoral')
# plt.hist(df["Purchase"], bins=30, color='lightcoral')
plt.show()
```



Interpretation

- Most customers make low to mid-range purchases (₹2,000 to ₹10,000).
- here are some purchases reaching up to ₹25,000, but they're relatively rare.

Business Insights for Walmart

- Position more products, bundles, and offers in ₹5,000–₹10,000 range
- Highlight these items more during Christmas,new years, Diwali sales, etc.
- Walmart should optimize inventory, promotions, and discount strategies around this dominant mid-price range while
- Customers who spend ₹15,000–₹25,000 are rare but valuable. Offer them VIP programs, cashback etc

```
users_by_age=df.groupby("Age")["User_ID"].count().reset_index()
users_by_age = users_by_age.rename(columns={"User_ID":"no_of_users"})
```

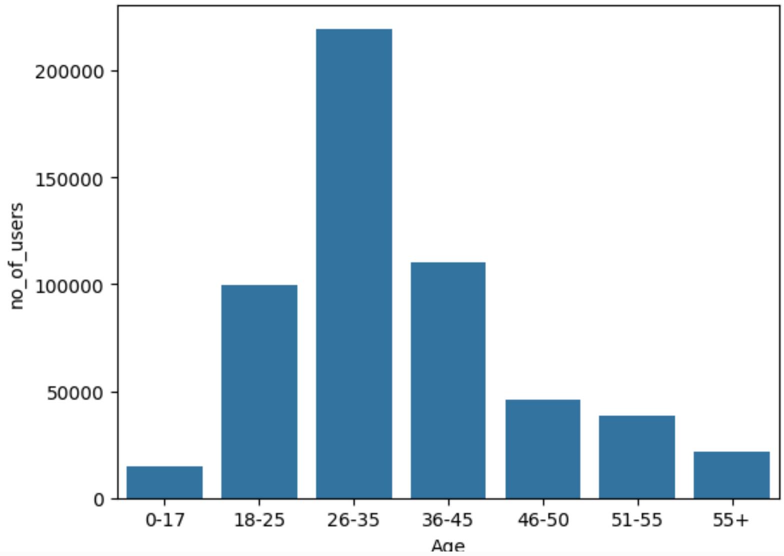
```
users_by_age
```

	Age	no_of_users
0	0-17	15102
1	18-25	99660
2	26-35	219587
3	36-45	110013
4	46-50	45701
5	51-55	38501
6	55+	21504

Next steps: Generate code with `users_by_age` View recommended plots New interactive sheet

Analysing which Age Group and purchase amount

```
sns.barplot(data=users_by_age, x="Age", y="no_of_users")
plt.show()
```



Interpretation

- 26–35 is the Core Customer Base
- 18–25 and 36–45 are Secondary Priority Groups
- Minimal Contribution from 0–17 and 55+

Business Insights for Walmart

- Focus on 26–35 Group . Likely consists of Working professionals,Young parents,Digitally active users etc
- Prioritize this group in Product targeting , App UI/UX decisions,mail/SMS campaigns etc

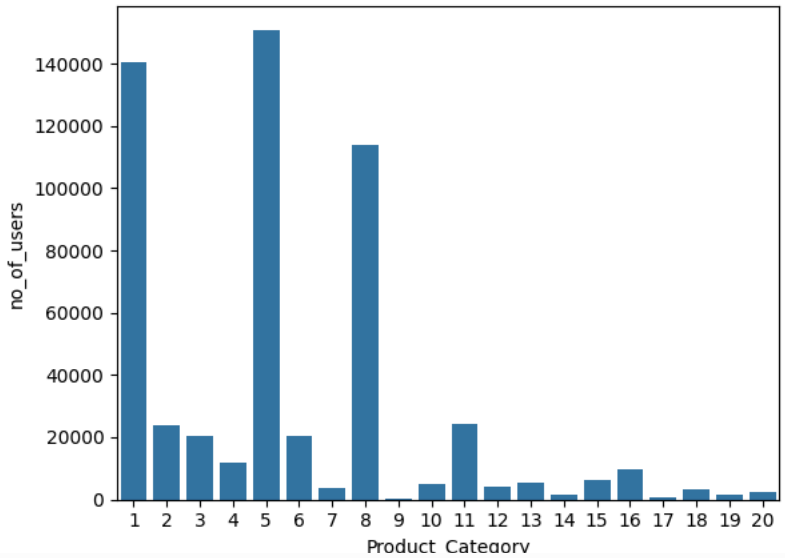
Finding most wanted Product Category

```
productcat_df = df.groupby("Product_Category")["User_ID"].count().reset_index()
productcat_df=productcat_df.rename(columns={"User_ID":"no_of_users"})
productcat_df.head(5)
```

	Product_Category	no_of_users
0	1	140378
1	2	23864
2	3	20213
3	4	11753
4	5	150933

Next steps: Generate code with productcat_df View recommended plots New interactive sheet

```
sns.barplot(x='Product_Category', y="no_of_users",data=productcat_df)
plt.show()
```



Interpretation

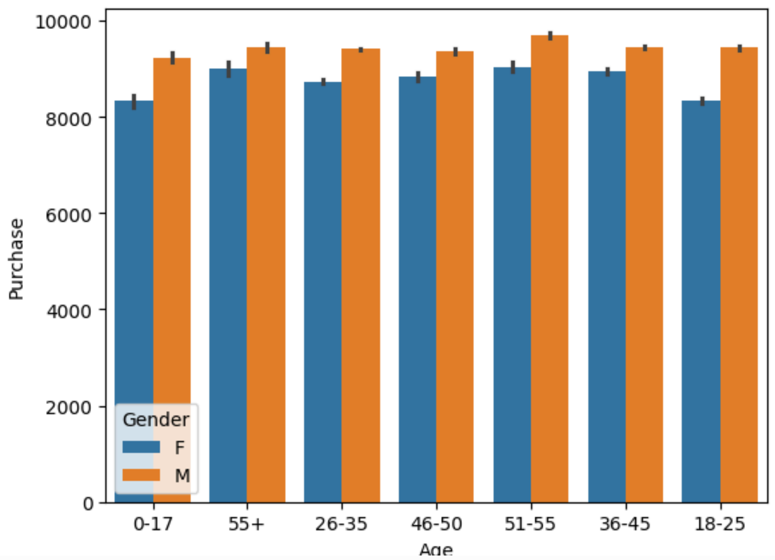
- Top Product Categories -Category 5 has the highest user count (~150,000+) , Followed by Category 1 and 8
- Moderately Purchased Categories - Categories like 2, 3, 6, 11
- Low Engagement Categories - Categories 9, 12–20 have very low user counts, suggesting niche or underperforming segments.

Business Insights for Walmart

- A small number of product categories (especially 1, 5, and 8) drive the majority of user engagement. Walmart should optimize promotions, inventory, and UX around these categories
- rethinking or repositioning the low-performing ones. Segmenting marketing efforts by category performance will improve both conversion and inventory turnover."

Exploring data for the relation among age , gender and purchase amount

```
sns.barplot(x='Age',y='Purchase',hue='Gender',data=df)
plt.show()
```



Interpretation

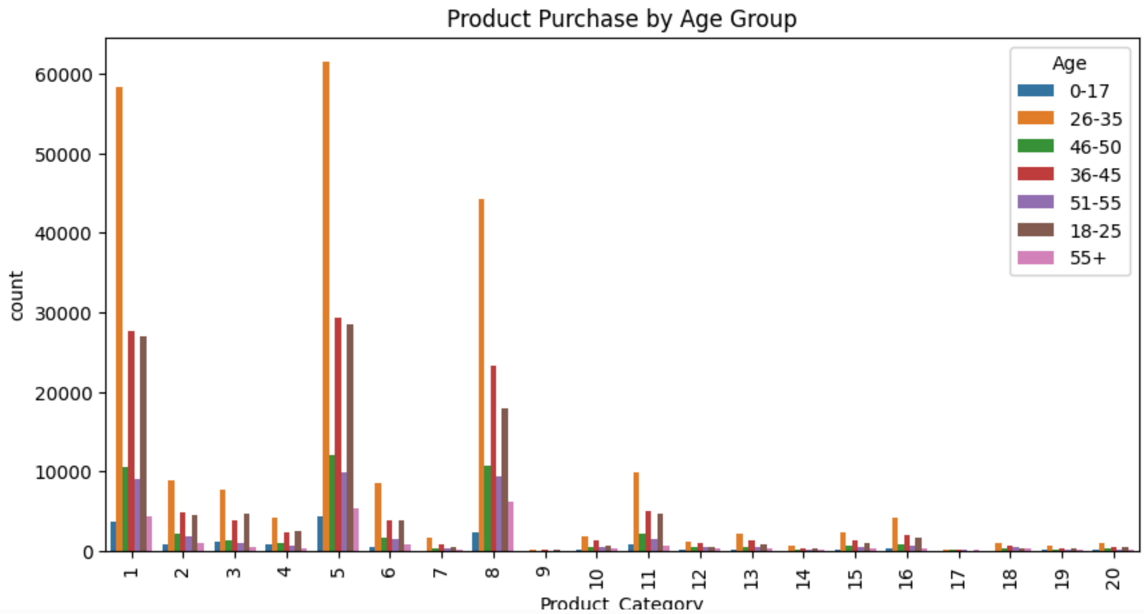
- The chart reveals that males consistently spend more than females across all age groups
- Female spend is more consistent across age

Business Insights for Walmart

- For males in high-spend age groups, promote EMIs, cashback, and premium memberships.
- For younger females, use value-based messaging and combo offers.

How various age groups are related to the various product category

```
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x="Product_Category", hue="Age")
plt.title("Product Purchase by Age Group")
plt.xticks(rotation=90)
plt.show()
```

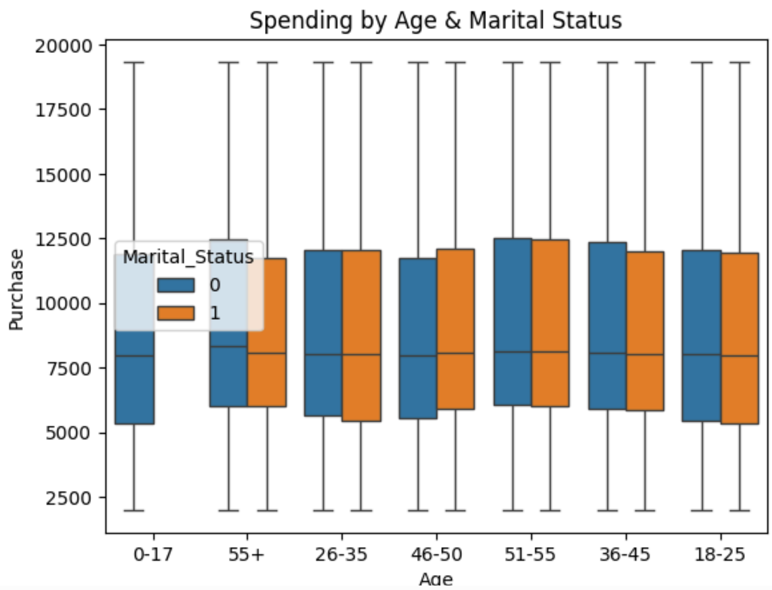


Business Insights for Walmart

- This chart reveals that the 26–35 age group is the most active shopper across almost all product categories, especially Categories 1, 5, 7, and 8
- Walmart should prioritize this group for high-margin offers, customized digital experiences, and loyalty programs.
- Walmart should prioritize this group for high-margin offers, customized digital experiences, and loyalty programs.

How Spending by Age & Marital Status are related

```
sns.boxplot(data=df_clippedPurchase, x="Age", y="Purchase", hue="Marital_Status")
plt.title("Spending by Age & Marital Status")
plt.show()
```



- Spending behavior is not significantly influenced by marital status across any age group. Therefore, Walmart should deprioritize marital status as a primary segmentation variable.

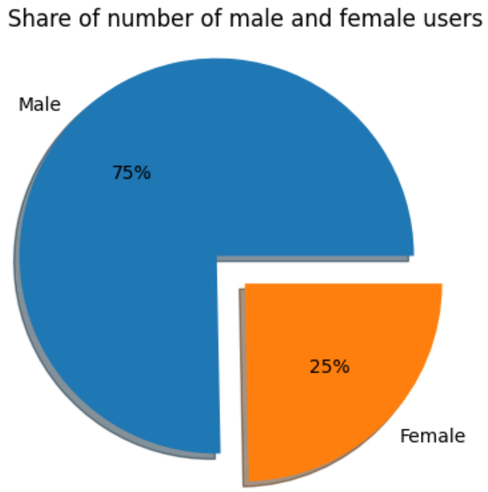
- Marketing strategies should instead focus on stronger predictors like age, gender, and product category preferences to optimize impact and efficiency.

Analysing the relation between Gender and Purchase

```
pie_data_gender = df["Gender"].value_counts()
pie_data_gender
```

Gender	count
M	414259
F	135809

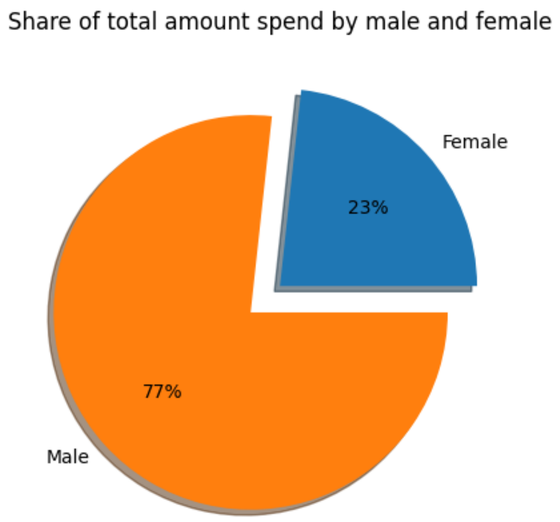
```
plt.pie(pie_data_gender ,labels =["Male","Female"] , explode=[0.2, 0] ,shadow = True , autopct='%1.0f%%')
plt.title("Share of number of male and female users")
plt.show()
```




```
purchase_by_gender=df.groupby("Gender")["Purchase"].sum()
purchase_by_gender
```

Gender	Purchase
F	1186232642
M	3909580100

```
plt.pie(purchase_by_gender , labels=["Female" , "Male"] , explode=[0,0.2] , shadow=True , autopct='%1.0f%%')
plt.title("Share of total amount spend by male and female")
plt.show()
```




```
df_male_purchase_table=df.loc[df["Gender"]=="M"]
df_male_purchase = df_male_purchase_table["Purchase"]
df_male_purchase.head(5)
```

	Purchase
4	7969
5	15227
6	19215
7	15854
8	15686

Using the central limit theorem to compute the 95% confidence intervals for the average amount spent per gender.

```
#FINDING FOR 30
n=30
z1=norm.ppf(.025)
z2=norm.ppf(1-.025)
mean_t=np.mean(df_male_purchase)  # Mean of the total Population

sample_30=[np.mean(df_male_purchase.sample(30)) for i in range(1000)]
mean_s = np.mean(sample_30)  # Mean of the sample Population
print("mean of the sample - " , mean_s)
std_dev=df_male_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(30)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9434.493266666666
std deviation -  5092.186209777949
std error -  929.7017513707056
lower limit -  7612.3113176162715
upper limit -  11256.675215717061
range -  3644.36389810079
```

```
#FINDING FOR 300
n=300

sample_300=[np.mean(df_male_purchase.sample(300)) for i in range(1000)]
mean_s = np.mean(sample_300) # Mean of the sample Population
print("mean of the sample - " , mean_t , mean_s)
std_dev=df_male_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(300)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
```

```
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9437.526040472265 9429.74575
std deviation -  5092.186209777949
std error -  293.9975078978999
lower limit -  8853.521222975585
upper limit -  10005.970277024415
range -  1152.4490540488296
```

```
#FINDING FOR 3000
n=3000
sample_3000=[np.mean(df_male_purchase.sample(3000)) for i in range(1000)]
mean_s = np.mean(sample_3000)
print("mean of the sample - " , mean_t , mean_s)
std_dev=df_male_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(3000)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9437.526040472265 9436.909349666666
std deviation -  5092.186209777949
std error -  92.97017513707056
lower limit -  9254.691154761627
upper limit -  9619.127544571706
range -  364.43638981007825
```

```
# FINDING FOR ENTIRE
n=len(df_male_purchase)

sample_entire=[np.mean(df_male_purchase) for i in range(1000)]
mean_s = np.mean(sample_entire)
print("mean of the sample - " , mean_t , mean_s)
std_dev=df_male_purchase.std()
# print("std dev " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9437.526040472265 9437.526040472265
std error -  7.91167247562093
lower limit -  9422.01944736257
upper limit -  9453.032633581959
range -  31.013186219388444
```

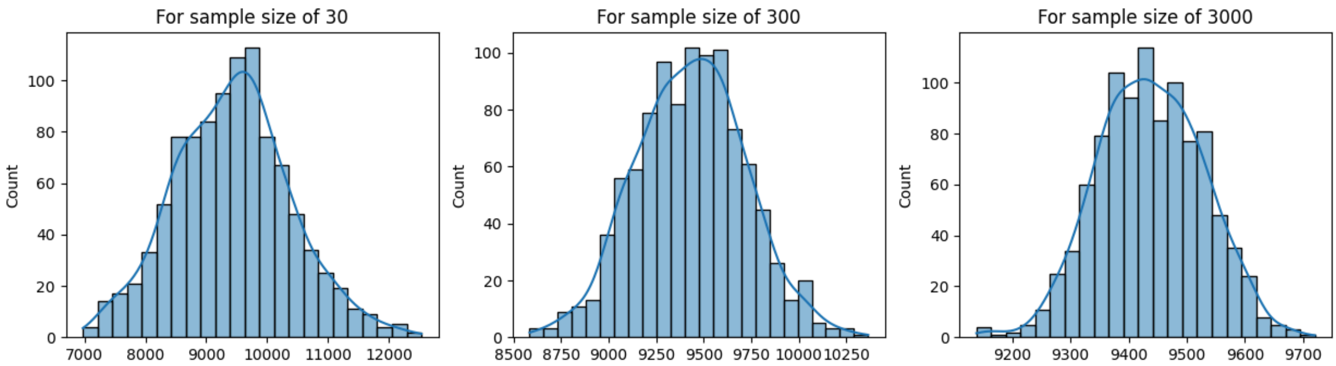
```
plt.figure(figsize=(15, 8))

plt.subplot(2, 3, 4)
plt.title("For sample size of 30")
sample_30=[np.mean(df_male_purchase.sample(30))for i in range(1000)]
sns.histplot(sample_30 , kde=True)

plt.subplot(2, 3, 5)
plt.title("For sample size of 300")
sample_300=[np.mean(df_male_purchase.sample(300)) for i in range(1000)]
sns.histplot(sample_300 , kde=True)

plt.subplot(2, 3, 6)
plt.title("For sample size of 3000")
sample_3000=[np.mean(df_male_purchase.sample(3000)) for i in range(1000)]
sns.histplot(sample_3000 , kde=True)


plt.show()
```




```
# ==============================          FOR FEMALE            =====================================================


df_female_purchase_table=df.loc[df["Gender"]=="F"]
df_female_purchase = df_female_purchase_table["Purchase"]


#FINDING FOR 30
n=30
z1=norm.ppf(.025)
z2=norm.ppf(1-.025)

mean_t=np.mean(df_female_purchase)
sample_30=[np.mean(df_female_purchase.sample(30)) for i in range(1000)]
mean_s = np.mean(sample_30)
print("mean of the sample - " , mean_s)
std_dev=df_female_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(30)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  8725.486066666666
std deviation -  4767.233289291444
std error -  870.3737364781583
lower limit -  7019.58489007992
upper limit -  10431.387243253412
range -  3411.8023531734916
```

```
#FINDING FOR 300
n=300
sample_300=[np.mean(df_female_purchase.sample(300)) for i in range(1000)]
mean_s = np.mean(sample_300)
print("mean of the sample - " ,mean_s)
std_dev=df_female_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(300)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  8740.7094
std deviation -  4767.233289291444
std error -  275.236342286216
lower limit -  8201.256081882477
upper limit -  9280.16271811752
range -  1078.906636235044
```

```
#FINDING FOR 3000
n=3000
sample_3000=[np.mean(df_female_purchase.sample(3000)) for i in range(1000)]
mean_s = np.mean(sample_3000)
print("mean of the sample - " , mean_s)
std_dev=df_female_purchase.std()
print("std deviation -  " , std_dev)
std_error=std_dev/np.sqrt(3000)
print("std error - " , std_error)
```

```
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  8734.523680333336
std deviation -   4767.233289291444
std error -  87.03737364781583
lower limit -  8563.933562674662
upper limit -  8905.11379799201
range -  341.180235317348
```

```
# FINDING FOR ENTIRE
n=len(df_female_purchase)

sample_entire=[np.mean(df_female_purchase) for i in range(1000)]
mean_s = np.mean(sample_entire)
print("mean of the sample -" , mean_t , mean_s)
std_dev=df_female_purchase.std()
# print("std dev " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample - 8734.565765155476 8734.565765155477
std error -  12.936063220950688
lower limit -  8709.211547140681
upper limit -  8759.919983170274
range -  50.70843602959212
```

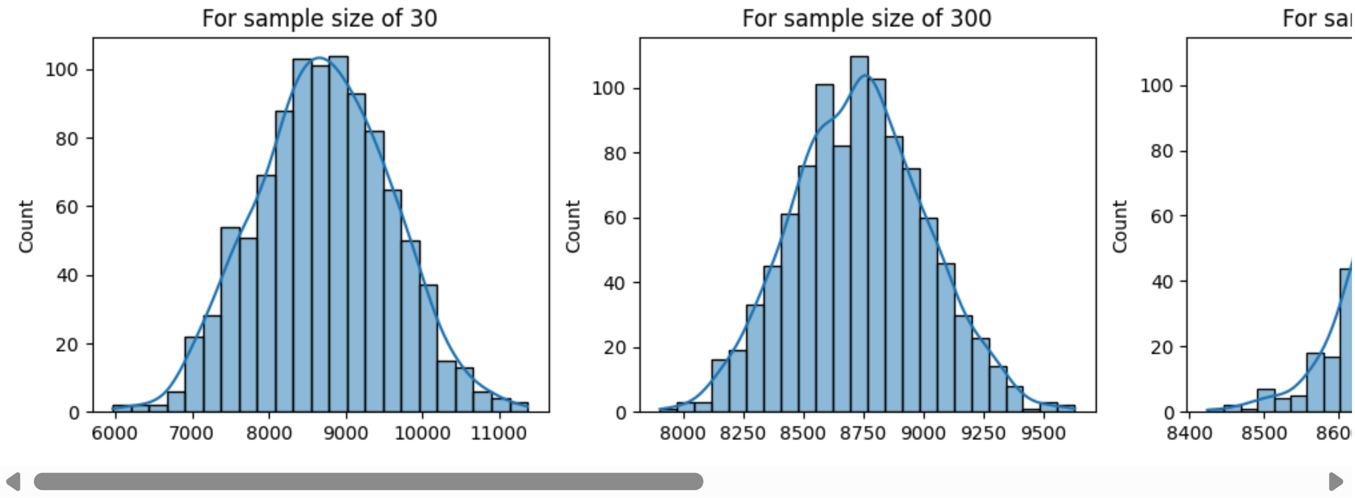
```
plt.figure(figsize=(15, 8))

plt.subplot(2, 3, 4)
plt.title("For sample size of 30")
sample_30=[np.mean(df_female_purchase.sample(30))for i in range(1000)]
sns.histplot(sample_30 , kde=True)

plt.subplot(2, 3, 5)
plt.title("For sample size of 300")
sample_300=[np.mean(df_female_purchase.sample(300))for i in range(1000)]
sns.histplot(sample_300 , kde=True)

plt.subplot(2, 3, 6)
plt.title("For sample size of 3000")
sample_3000=[np.mean(df_female_purchase.sample(3000))for i in range(1000)]
sns.histplot(sample_3000 , kde=True)
```

```
<Axes: title={'center': 'For sample size of 3000'}, ylabel='Count'>
```



From the above calculated CLT following is the analysis.

1. Is the confidence interval computed using the entire dataset wider for one of the genders? Why is this the case?
   - confidence interval computed using the entire dataset for Female = 50 and male =31
   - This is because there are more number of male users (414259) than female users (135809) ,this results in lesser deviation in means of male users than female users

2. How is the width of the confidence interval affected by the sample size?
   - As the sample size increases the range of the upper and lower limit of the means decreases as the value of standard error also decrease with increase in sample size
3. Do the confidence intervals for different sample sizes overlap?
   - Yes the sample size overlaps however range becomming more and more narrow with increase in sample size
4. How does the sample size affect the shape of the distributions of the means?
   - Keeping the values in x axis unchanged , the graph becommes more and more narrow as sample size increases

Interpretation of These Confidence Intervals:

1. The intervals do not overlap, which means?
   - There is statistically significant difference in average spending between male and female customers.
   - Males spend more than females on average (by approx ₹700–₹1,000).
2. Reliability:
   - The sample size is large (3000), and with CLT, the estimate is reliable.
   - We are 95% confident that the true population mean lies within those ranges.

Business Implications for Walmart:

- Male customers are higher spenders
  - Target them with premium product bundles, high-ticket items.
  - high-end electronics, gadgets, or hobby categories.
  - Feature more male-targeted items (electronics, tools, fitness gear) at prominent store positions or online banners
- Female customers may be more price-sensitive
  - Use value-oriented offers.
  - Highlight savings and essential items.

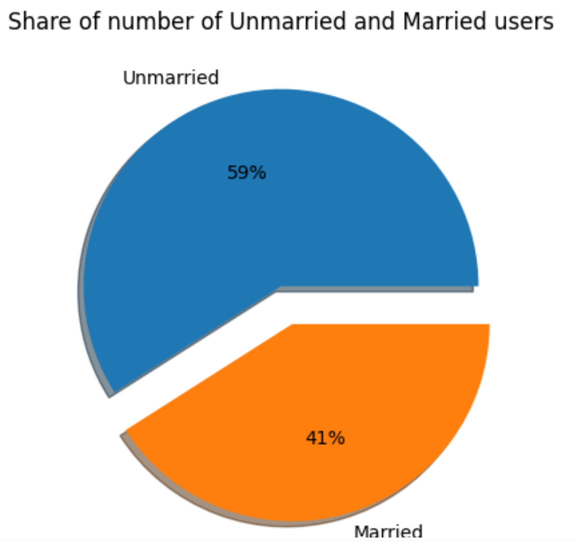
Walmart should segment customers by gender when planning promotional campaigns and inventory strategies. Males show a statistically higher average spend, making them ideal targets higher end products. Meanwhile, female-oriented campaigns should focus on value, deals, and utility-based product placements to drive engagement.

Analysing the relation between Marital Status and Purchase

```
pie_data_marstatus = df["Marital_Status"].value_counts()
pie_data_marstatus
```

Marital_Status	count
0	324731
1	225337

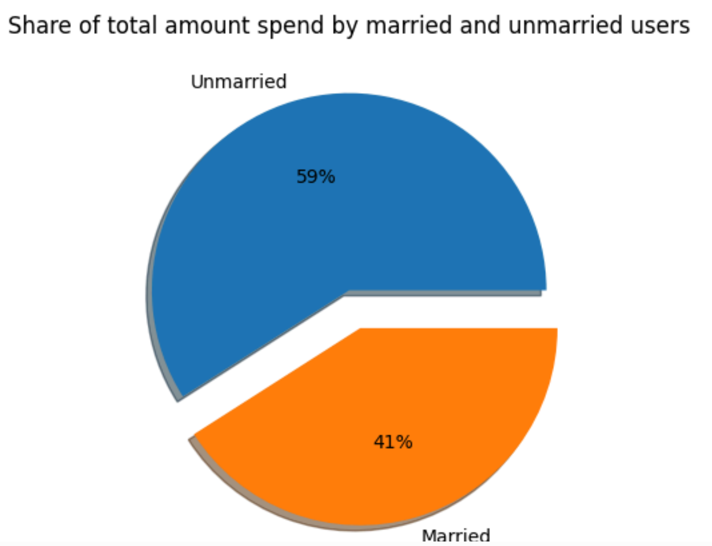
```
plt.pie(pie_data_marstatus , labels=[ "Unmarried" , "Married" ] , explode=[0,0.2] , shadow=True , autopct='%1.0f%%')
plt.title("Share of number of Unmarried and Married users")
plt.show()
```




```
purchase_by_marstatus=df.groupby("Marital_Status")["Purchase"].sum()
purchase_by_marstatus
```

Marital_Status	Purchase
0	3008927447
1	2086885295

```
plt.pie(purchase_by_marstatus , labels=[ "Unmarried" , "Married" ] , explode=[0,0.2] , shadow=True , autopct='%1.0f%%')
plt.title("Share of total amount spend by married and unmarried users")
plt.show()
```



Finding CLT for Unmarried

```
df_unmaried_purchase_table=df.loc[df["Marital_Status"]==0]
df_unmaried_purchase = df_unmaried_purchase_table["Purchase"]
df_unmaried_purchase.head(5)
```

	Purchase
0	8370
1	15200
2	1422
3	1057
4	7969

```
#FINDING FOR 30
n=30
z1=norm.ppf(.025)
z2=norm.ppf(1-.025)

mean_t=np.mean(df_unmaried_purchase)
print(mean_t)
sample_30=[np.mean(df_unmaried_purchase.sample(30)) for i in range(1000)]
mean_s = np.mean(sample_30)
print("mean of the sample - " ,mean_s)
std_dev=df_unmaried_purchase.std()
print("std deviation -  " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
9265.907618921507
mean of the sample -  9221.221066666667
std deviation -   5027.347858674457
std error -  917.8639422070979
lower limit -  7422.240797232801
upper limit -  11020.201336100532
range -  3597.960538867731
```

```
#FINDING FOR 300
n=300
sample_300=[np.mean(df_unmaried_purchase.sample(300)) for i in range(1000)]
mean_s = np.mean(sample_300)
print("mean of the sample - " , mean_s)
std_dev=df_unmaried_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9262.95027
std deviation -  5027.347858674457
std error -  290.2540639515586
lower limit -  8694.062758288559
upper limit -  9831.83778171144
range -  1137.7750234228806
```

```
#FINDING FOR 3000
n=3000
sample_3000=[np.mean(df_unmaried_purchase.sample(3000)) for i in range(1000)]
mean_s = np.mean(sample_3000)
print("mean of the sample - " , mean_t , mean_s)
std_dev=df_unmaried_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9265.907618921507 9268.65244
std deviation -  5027.347858674457
std error -  91.7863942207098
lower limit -  9088.754413056613
```

```
upper limit -  9448.550466943387
range -  359.79605388677373
```

```
# FINDING FOR ENTIRE
n=len(df_unmaried_purchase)
sample_entire=[np.mean(df_unmaried_purchase) for i in range(1000)]
mean_s = np.mean(sample_entire)
print("mean" , mean_s)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean 9265.907618921508
std error -  8.82220330129379
lower limit -  9248.616418186682
upper limit -  9283.198819656334
range -  34.58240146965181
```

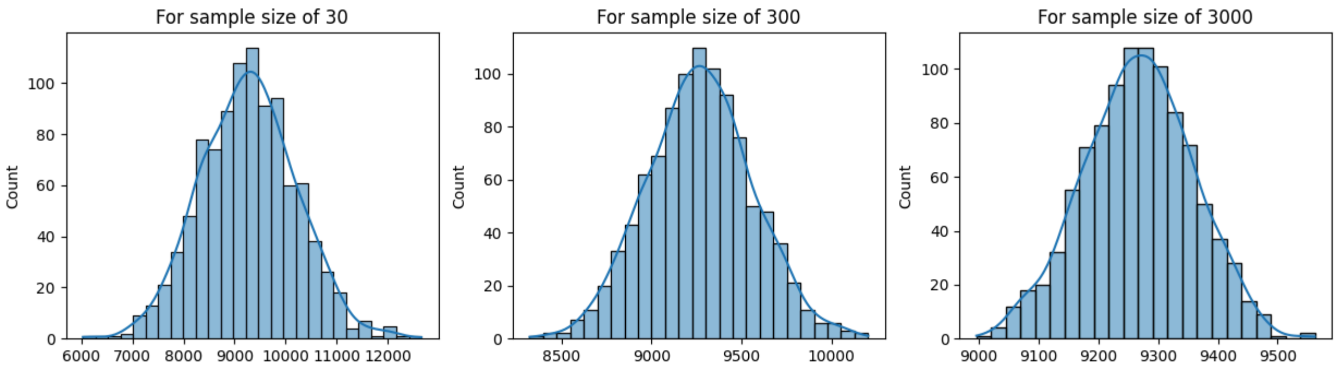
```
plt.figure(figsize=(15, 8))

plt.subplot(2, 3, 4)
plt.title("For sample size of 30")
sample_30=[np.mean(df_unmaried_purchase.sample(30))for i in range(1000)]
sns.histplot(sample_30 , kde=True)

plt.subplot(2, 3, 5)
plt.title("For sample size of 300")
sample_300=[np.mean(df_unmaried_purchase.sample(300))for i in range(1000)]
sns.histplot(sample_300 , kde=True)

plt.subplot(2, 3, 6)
plt.title("For sample size of 3000")
sample_3000=[np.mean(df_unmaried_purchase.sample(3000))for i in range(1000)]
sns.histplot(sample_3000 , kde=True)
```

```
<Axes: title={'center': 'For sample size of 3000'}, ylabel='Count'>
```




```
df_married_purchase_table=df.loc[df["Marital_Status"]==1]
df_married_purchase = df_married_purchase_table["Purchase"]
df_married_purchase
```

	Purchase
6	19215
7	15854
8	15686
9	7871
10	5254
...	...
550060	494
550061	599
550063	368
550065	137
550067	490

225337 rows × 1 columns

```
#FINDING FOR 30
n=30
z1=norm.ppf(.025)
z2=norm.ppf(1-.025)

mean_t=np.mean(df_married_purchase)
print(mean_t)
sample_30=[np.mean(df_married_purchase.sample(30)) for i in range(1000)]
mean_s = np.mean(sample_30)
print("mean of the sample - " ,mean_s)
std_dev=df_married_purchase.std()
print("std deviation -  " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
9261.174574082374
mean of the sample -  9215.718866666668
std deviation -   5016.89737779313
std error -  915.9559541686049
lower limit -  7420.4781850711815
upper limit -  11010.959548262153
range -  3590.481363190972
```

```
#FINDING FOR 300
n=300
sample_300=[np.mean(df_married_purchase.sample(300)) for i in range(1000)]
mean_s = np.mean(sample_300)
print("mean of the sample - " , mean_s)
std_dev=df_married_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9256.748366666667
std deviation -  5016.89737779313
std error -  289.65070515655907
lower limit -  8689.04341646318
upper limit -  9824.453316870153
range -  1135.4099004069722
```

```
#FINDING FOR 3000
n=3000
sample_3000=[np.mean(df_married_purchase.sample(3000)) for i in range(1000)]
mean_s = np.mean(sample_3000)
print("mean of the sample - " , mean_s)
std_dev=df_married_purchase.std()
print("std deviation - " , std_dev)
```

```
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9257.562018333332
std deviation -  5016.89737779313
std error -  91.59559541686048
lower limit -  9078.037950173784
upper limit -  9437.08608649288
range -  359.04813631909565
```

```
# FINDING FOR ENTIRE
n=len(df_married_purchase)
sample_entire=[np.mean(df_married_purchase) for i in range(1000)]
mean_s = np.mean(sample_entire)
print("mean" , mean_s)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean 9261.174574082372
std error -  10.568636561021444
lower limit -  9240.460427057076
upper limit -  9281.888721107667
range -  41.42829405059092
```

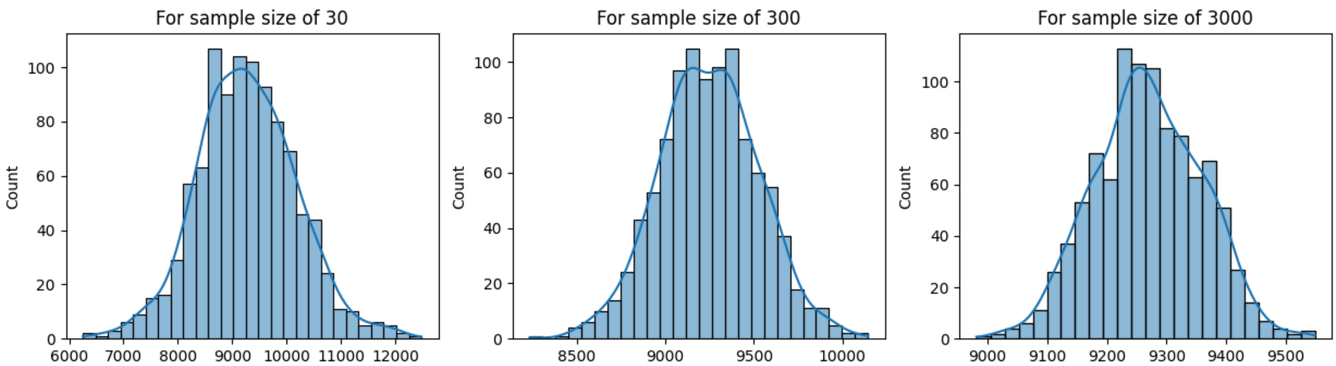
```
plt.figure(figsize=(15, 8))

plt.subplot(2, 3, 4)
plt.title("For sample size of 30")
sample_30=[np.mean(df_married_purchase.sample(30))for i in range(1000)]
sns.histplot(sample_30 , kde=True)

plt.subplot(2, 3, 5)
plt.title("For sample size of 300")
sample_300=[np.mean(df_married_purchase.sample(300))for i in range(1000)]
sns.histplot(sample_300 , kde=True)

plt.subplot(2, 3, 6)
plt.title("For sample size of 3000")
sample_3000=[np.mean(df_married_purchase.sample(3000))for i in range(1000)]
sns.histplot(sample_3000 , kde=True)
```

```
<Axes: title={'center': 'For sample size of 3000'}, ylabel='Count'>
```



From the above calculated CLT following is the analysis.

1. Is the confidence interval computed using the entire dataset wider for Married or Unmarried? Why is this the case?
    - confidence interval computed using the entire dataset for Unmarried = 34 and married =41
    - This is because there are more number of Unmarried users than Married users ,this results in lesser deviation in means of Unmarried users than Married users
2. How is the width of the confidence interval affected by the sample size?

- As the sample size increases the range of the upper and lower limit of the means decreases as the value of standard error also decrease with increase in sample size

3. Do the confidence intervals for different sample sizes overlap?
    - Yes the sample size overlaps with range becomming more and more narrow with increase in sample size
4. How does the sample size affect the shape of the distributions of the means?
    - Keeping the values in x axis unchanged , the graph becommes more and more narrow as sample size increases , due to lesser statndard error

Interpretation of These Confidence Intervals:

1. The CIs for married and unmarried users are almost identical, and they significantly overlap.
    - no statistically significant difference in average spending between married and unmarried users at the 95% confidence level.
    - Marital status does not influence purchase amount in a meaningful way.
2. Reliability:
    - The sample size is large (3000), and with CLT, the estimate is reliable.
    - We are 95% confident that the true population mean lies within those ranges.

Significance for Walmart

Since married and unmarried customers spend similarly, Walmart:

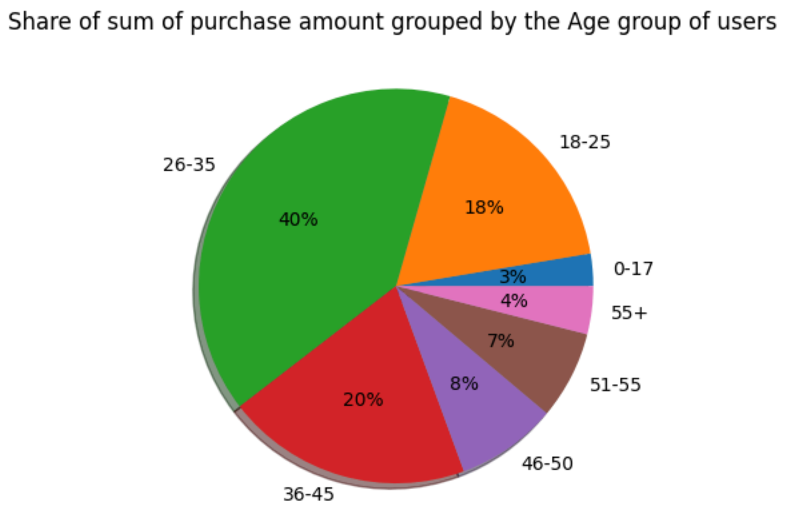
- Should not use marital status alone to segment for promotions. There's no clear economic advantage in tailoring campaigns purely based on this variable.
- Instead, combine marital status with other factors like Age , Product , category , City , Purchase frequency etc

Analysing the relation between Age group and Purchase

```
pie_data_age = df.groupby("Age")["Purchase"].sum()
pie_data_age
```

Age	Purchase
0-17	134913183
18-25	913848675
26-35	2031770578
36-45	1026569884
46-50	420843403
51-55	367099644
55+	200767375

```
plt.pie(pie_data_age ,labels=["0-17","18-25","26-35","36-45","46-50","51-55","55+"], shadow=True , autopct='%1.0f%%')
plt.title("Share of sum of purchase amount grouped by the Age group of users ")
plt.show()
```




```
df_age_0to17_purchase_table=df.loc[df["Age"]=="0-17"]
df_age_0to17_purchase = df_age_0to17_purchase_table["Purchase"]
df_age_0to17_purchase.head(5)

#FINDING FOR 3000
n=3000
sample_3000=[np.mean(df_age_0to17_purchase.sample(3000)) for i in range(1000)]
mean_s = np.mean(sample_3000)
print("mean of the sample - " , mean_s)
std_dev=df_age_0to17_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  8936.129114
std deviation -  5111.11404600277
std error -  93.31574856590714
lower limit -  8753.233607620426
upper limit -  9119.024620379572
range -  365.7910127591458
```

```
df_age_18to25_purchase_table=df.loc[df["Age"]=="18-25"]
df_age_18to25_purchase = df_age_18to25_purchase_table["Purchase"]
df_age_18to25_purchase.head(5)

#FINDING FOR 3000
n=3000
sample_3000=[np.mean(df_age_18to25_purchase.sample(3000)) for i in range(1000)]
mean_s = np.mean(sample_3000)
print("mean of the sample - " , mean_s)
std_dev=df_age_18to25_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9170.842647999998
std deviation -  5034.321997176577
std error -  91.91372398660234
lower limit -  8990.695059301303
upper limit -  9350.990236698693
range -  360.29517739739094
```

```
df_age_26to35_purchase_table=df.loc[df["Age"]=="26-35"]
df_age_26to35_purchase = df_age_26to35_purchase_table["Purchase"]
df_age_26to35_purchase
#FINDING FOR 3000
n=3000
sample_3000=[np.mean(df_age_26to35_purchase.sample(3000)) for i in range(1000)]
```

```
mean_s = np.mean(sample_3000)
print("mean of the sample - " , mean_s)
std_dev=df_age_26to35_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9249.721937333332
std deviation -  5010.527303002927
std error -  91.4792942950075
lower limit -  9070.425815183977
upper limit -  9429.018059482687
range -  358.5922442987103
```

```
df_age_36to45_purchase_table=df.loc[df["Age"]=="36-45"]
df_age_36to45_purchase = df_age_36to45_purchase_table["Purchase"]
df_age_36to45_purchase

#FINDING FOR 3000
n=3000
sample_3000=[np.mean(df_age_36to45_purchase.sample(3000)) for i in range(1000)]
mean_s = np.mean(sample_3000)
print("mean of the sample - " , mean_s)
std_dev=df_age_36to45_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9328.837956999998
std deviation -  5022.923879204652
std error -  91.70562377572473
lower limit -  9149.098237219798
upper limit -  9508.577676780198
range -  359.47943956040035
```

```
df_age_46to50_purchase_table=df.loc[df["Age"]=="46-50"]
df_age_46to50_purchase = df_age_46to50_purchase_table["Purchase"]
df_age_46to50_purchase

#FINDING FOR 3000
n=3000
sample_3000=[np.mean(df_age_46to50_purchase.sample(3000)) for i in range(1000)]
mean_s = np.mean(sample_3000)
print("mean of the sample - " , mean_s)
std_dev=df_age_46to50_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9207.100094333335
std deviation -  4967.216367142921
std error -  90.68854840976802
lower limit -  9029.353805639972
upper limit -  9384.846383026697
range -  355.4925773867253
```

```
df_age_51to55_purchase_table=df.loc[df["Age"]=="51-55"]
df_age_51to55_purchase = df_age_51to55_purchase_table["Purchase"]
df_age_51to55_purchase.head(5)

#FINDING FOR 3000
n=3000
sample_3000=[np.mean(df_age_51to55_purchase.sample(3000)) for i in range(1000)
mean_s = np.mean(sample_3000)
print("mean of the sample - " , mean_s)
std_dev=df_age_51to55_purchase.std()
print("std deviation - " , std_dev)
```

```
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9529.056419333336
std deviation -  5087.368079602116
std error -  92.88220851766054
lower limit -  9347.010635834182
upper limit -  9711.10220283249
range -  364.0915669983078
```

```
df_age_55plus_purchase_table=df.loc[df["Age"]=="55+"]
df_age_55plus_purchase = df_age_55plus_purchase_table["Purchase"]
df_age_55plus_purchase.head(5)

#FINDING FOR 3000
n=3000
sample_3000=[np.mean(df_age_55plus_purchase.sample(3000)) for i in range(1000)]
mean_s = np.mean(sample_3000)
print("mean of the sample - " , mean_s)
std_dev=df_age_55plus_purchase.std()
print("std deviation - " , std_dev)
std_error=std_dev/np.sqrt(n)
print("std error - " , std_error)
val1 = mean_s + z1*(std_error)
print("lower limit - " , val1)
val2 = mean_s + z2*(std_error)
print("upper limit - " , val2)
print("range - " , val2 - val1)
```

```
mean of the sample -  9334.395127999998
std deviation -  5011.493995603418
std error -  91.49694360645626
lower limit -  9155.064413835851
upper limit -  9513.725842164145
range -  358.6614283282943
```

Interpretation of These Confidence Intervals and Significance for Walmart

- The intervals overlap, which means?
  - There is statistically no significant difference in average spending among various age group customers.
- Reliability:
  - The sample size is large (3000), and with CLT, the estimate is reliable.
  - We are 95% confident that the true population mean lies within those ranges.
- All age groups spend fairly close amounts The ranges overlap significantly, meaning spending does not vary sharply by age.
- 51–55 age group spends the most suggests stable financial standing and fewer dependents.Ideal group to target for: Luxury items,Health & wellness ,Home improvement products
- 0–17 and 18–25 spend the least Likely dependents or early career customers with:Lower income,Limited decision-making power . Wallmart should Focus on: Budget-friendly deals , Education-related items,Digital products (gaming, headphones, etc.)

Note : All the detailed Business recommendation are written within the sections.