walmart-business-case-study

August 14, 2024

Welcome to Colab!

Explore the Gemini API

The Gemini API gives you access to Gemini models created by Google DeepMind. Gemini models are built from the ground up to be multimodal, so you can reason seamlessly across text, images, code, and audio.

How to get started

Go to Google AI Studio and log in with your Google account.

Create an API key.

Use a quickstart for Python, or call the REST API using curl.

Explore use cases

Create a marketing campaign

Analyze audio recordings

Use System instructions in chat

To learn more, check out the Gemini cookbook or visit the Gemini API documentation.

Colab now has AI features powered by Gemini. The video below provides information on how to use these features, whether you're new to Python, or a seasoned veteran.

```
[ ]:
```

What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with - Zero configuration required - Access to GPUs free of charge - Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab or Colab Features You May Have Missed to learn more, or just get started below!

0.1 Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

[ ]:
```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

[ ]:
```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

[ ]:
```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

[ ]:
```
604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: create a new Colab notebook.

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see jupyter.org.

0.2 Data science

With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

[ ]:
```
import numpy as np
import IPython.display as display
from matplotlib import pyplot as plt
import io
import base64

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

fig = plt.figure(figsize=(4, 3), facecolor='w')
plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)
plt.title("Sample Visualization", fontsize=10)

data = io.BytesIO()
plt.savefig(data)
image = F"data:image/png;base64,{base64.b64encode(data.getvalue()).decode()}"
alt = "Sample Visualization"
display.display(display.Markdown(F"""![{alt}]({image})"""))
plt.close(fig)
```

Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

For example, if you find yourself waiting for **pandas** code to finish running and want to go faster, you can switch to a GPU Runtime and use libraries like RAPIDS cuDF that provide zero-code-change acceleration.

To learn more about accelerating pandas on Colab, see the 10 minute guide or US stock market data analysis demo.

0.3 Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code.

Colab is used extensively in the machine learning community with applications including: - Getting started with TensorFlow - Developing and training neural networks - Experimenting with TPUs - Disseminating AI research - Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the machine learning examples below.

0.4 More Resources

0.4.1 Working with Notebooks in Colab

- Overview of Colab
- Guide to Markdown
- Importing libraries and installing dependencies
- Saving and loading notebooks in GitHub
- Interactive forms
- Interactive widgets

Working with Data

- Loading data: Drive, Sheets, and Google Cloud Storage
- Charts: visualizing data
- Getting started with BigQuery

0.4.2 Machine Learning Crash Course

These are a few of the notebooks from Google's online Machine Learning course. See the full course website for more. - Intro to Pandas DataFrame - Intro to RAPIDS cuDF to accelerate pandas - Linear regression with tf.keras using synthetic data

Using Accelerated Hardware

- TensorFlow with GPUs
- TensorFlow with TPUs

0.4.3 Featured examples

- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

```
[ ]: import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
```

Basic Data Analysis

```
[ ]: df = pd.read_csv("walmart_data.txt")
     df.head()
```

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

       Stay_In_Current_City_Years  Marital_Status  Product_Category  Purchase
     0                          2               0                 3      8370
     1                          2               0                 1     15200
     2                          2               0                12      1422
     3                          2               0                12      1057
     4                         4+               0                 8      7969
```

```
[ ]: 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.columns
```

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

Count of Unique Values in Coloumns

```
[ ]: for col in df.columns:
       print(f"{col} : {df[col].nunique()}")
```

```
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
```

Detecting Null Values

```
[ ]: df.isna().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
```

```
[ ]: df_copy = df[["Gender" , "Age" , "Occupation" , "City_Category" ,␣
     ↪"Stay_In_Current_City_Years" , "Marital_Status" , "Product_Category" ,␣
     ↪"Purchase"]]

     df_copy["Gender"].replace(["M" , "F"] , [1 , 0] , inplace = True)

     df_copy["City_Category"].replace(["A" , "B" , "C"] , [0 , 1 , 2] , inplace =␣
     ↪True)

     df_copy["Age"].replace(["0-17" , "18-25" , "26-35" , "36-45" , "46-50" ,␣
     ↪"51-55" , "55+"] , [0,1,2,3,4,5,6] , inplace = True)

     df_copy["Stay_In_Current_City_Years"].replace(["0" , "1" , "2" , "3" , "4+"] ,␣
     ↪[0,1,2,3,4] , inplace = True)

     df_copy.corr()
```

```
<ipython-input-11-16ac5788cd43>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_copy["Gender"].replace(["M" , "F"] , [1 , 0] , inplace = True)
<ipython-input-11-16ac5788cd43>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_copy["City_Category"].replace(["A" , "B" , "C"] , [0 , 1 , 2] , inplace =
True)
<ipython-input-11-16ac5788cd43>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_copy["Age"].replace(["0-17" , "18-25" , "26-35" , "36-45" , "46-50" ,
"51-55" , "55+"] , [0,1,2,3,4,5,6] , inplace = True)
<ipython-input-11-16ac5788cd43>:9: SettingWithCopyWarning:
```

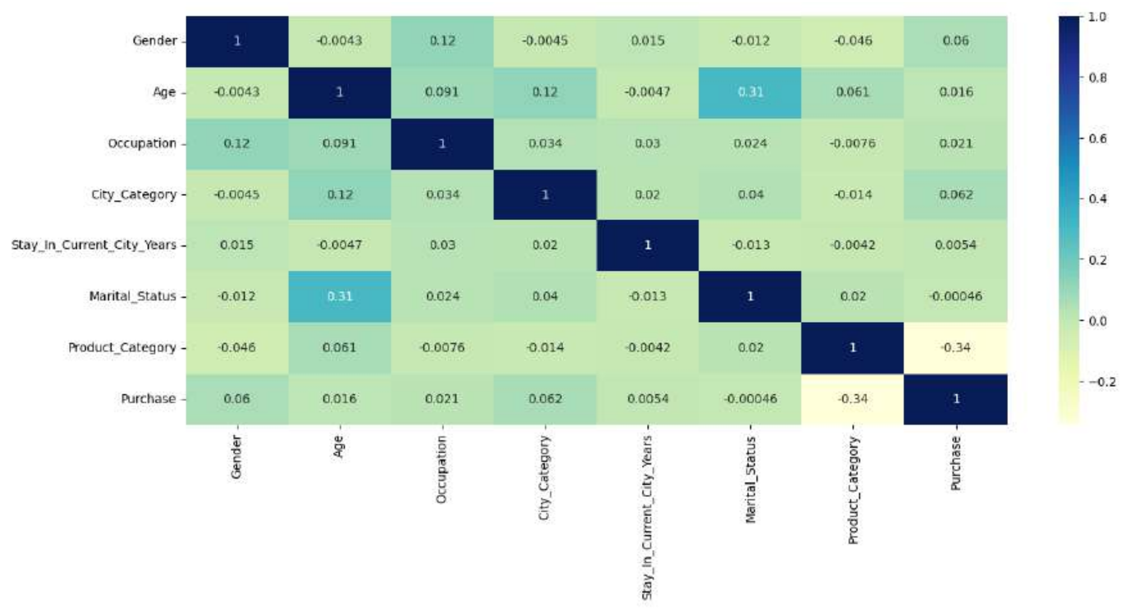
```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_copy["Stay_In_Current_City_Years"].replace(["0" , "1" , "2" , "3" , "4+"] ,
[0,1,2,3,4] , inplace = True)
```

[ ]:

```
                              Gender       Age  Occupation  City_Category  \
Gender                      1.000000 -0.004262    0.117291      -0.004515
Age                        -0.004262  1.000000    0.091463       0.123079
Occupation                  0.117291  0.091463    1.000000       0.034479
City_Category              -0.004515  0.123079    0.034479       1.000000
Stay_In_Current_City_Years  0.014660 -0.004712    0.030005       0.019946
Marital_Status             -0.011603  0.311738    0.024280       0.039790
Product_Category           -0.045594  0.061197   -0.007618      -0.014364
Purchase                    0.060346  0.015839    0.020833       0.061914

                            Stay_In_Current_City_Years  Marital_Status  \
Gender                                        0.014660       -0.011603
Age                                          -0.004712        0.311738
Occupation                                    0.030005        0.024280
City_Category                                 0.019946        0.039790
Stay_In_Current_City_Years                    1.000000       -0.012819
Marital_Status                               -0.012819        1.000000
Product_Category                             -0.004213        0.019888
Purchase                                      0.005422       -0.000463

                            Product_Category  Purchase
Gender                             -0.045594  0.060346
Age                                 0.061197  0.015839
Occupation                         -0.007618  0.020833
City_Category                      -0.014364  0.061914
Stay_In_Current_City_Years         -0.004213  0.005422
Marital_Status                      0.019888 -0.000463
Product_Category                    1.000000 -0.343703
Purchase                           -0.343703  1.000000
```

[ ]:

```python
plt.figure(figsize=(15,6))
sns.heatmap(df_copy.corr(), cmap="YlGnBu", annot=True)
plt.show()
```



Insights from the Heatmap:

Gender and Purchase: There is a slight positive correlation (0.06) between gender and purchase behavior, indicating that **gender may have a minor influence on purchasing decisions.**

Age and Purchase: The correlation between age and purchase is very low (0.016), suggesting that **age might not be a significant factor in predicting purchase behavior.**

Occupation and Purchase: The correlation between occupation and purchase behavior (0.021) is also minimal, **indicating that occupation does not have a strong impact on purchasing decisions.**

Product Category and Purchase: There is a more noticeable negative correlation (-0.34) between product category and purchase, indicating that certain product categories might be less favored by consumers.

Recommendations:

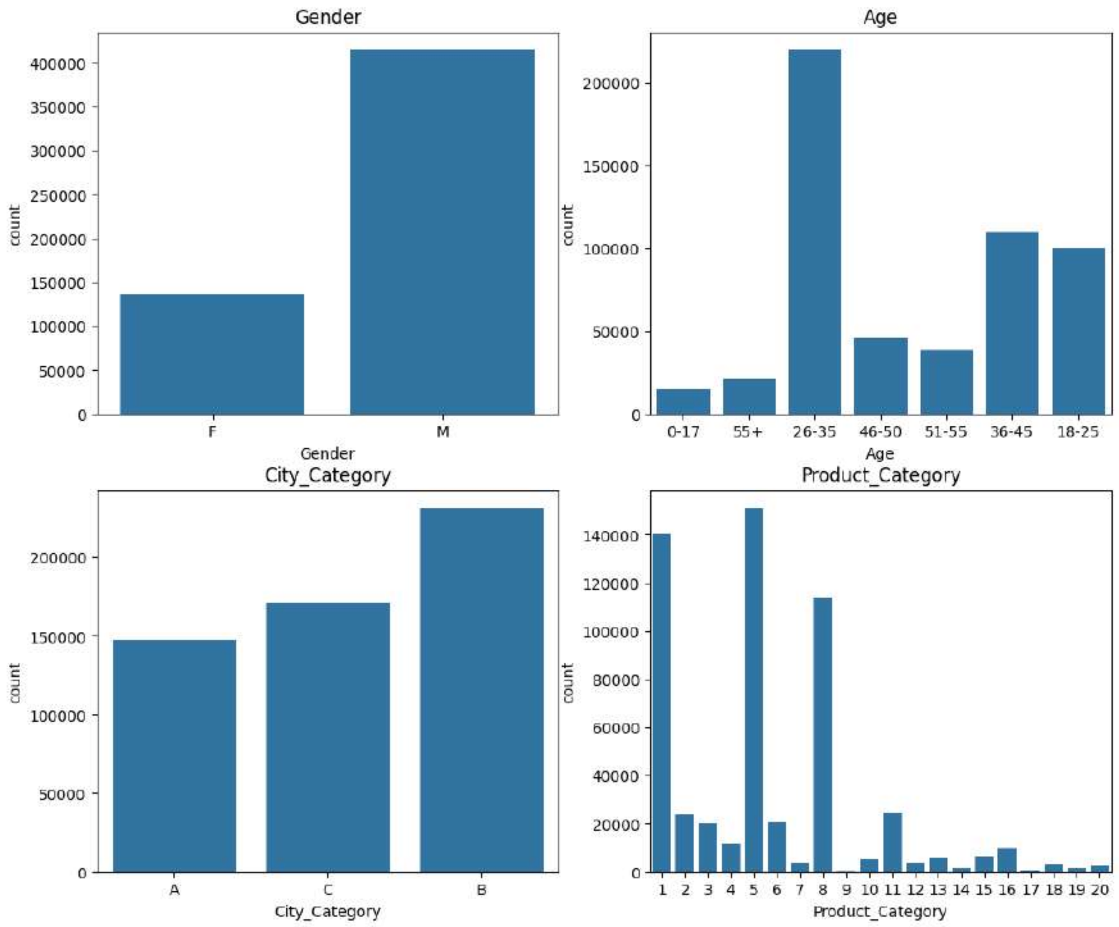
Targeted Marketing Based on Gender: Since gender shows a slight correlation with purchase behavior, consider creating gender-specific marketing campaigns to enhance sales.

Product Category Analysis: Given the significant negative correlation between product category and purchase, further investigation into the product categories is needed. Consider discontinuing or rebranding products that are less favored to align better with customer preferences.

Count Plots

```
[ ]: col = ["Gender" , "Age" , "City_Category" , "Product_Category"]

     plt.figure(figsize = (12 , 10))
```

```
m=1
for i in col:
  plt.subplot(2,2,m)
  sns.countplot(x = df[i])
  m += 1
  plt.title(f"{i}")
```



Insights and Recommendation from the Count Plots:

Gender Distribution:

- The majority of the customers are male, with a count significantly higher than female customers.
- Recommendation: Marketing strategies can be more tailored towards male customers, considering they form the larger segment of the customer base. However, efforts should also be made to engage more female customers to balance the gender distribution.

Age Distribution:

- The age group of 26-35 years is the most prominent, followed by 36-45 years and 18-25 years.

Very few customers fall into the 0-17 and 55+ age brackets.
- Recommendation: Products and promotions should primarily target the 26-35 age group, as they represent the largest customer segment.
- however Wallmart should also consider launching specific campaigns to attract younger and older age groups.

City Category:

- Customers from city category B are the most frequent, followed by category C and A.
- Recommendation: Retail strategies should focus on city category B, which has the highest customer base. However, marketing efforts in city categories A and C should not be neglected, as they still represent a significant portion of the customers.

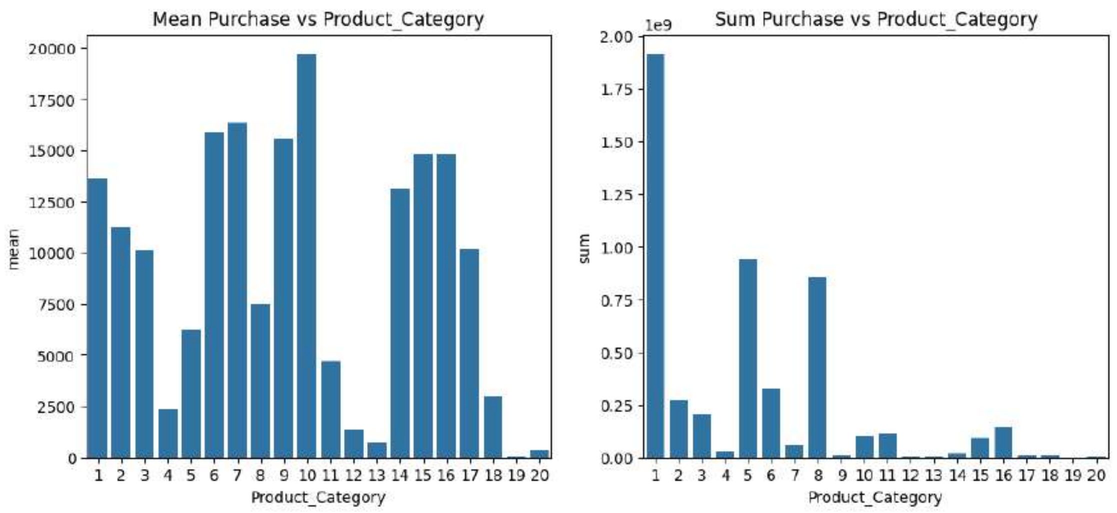
Product Category:

- Product categories 1 and 5 have the highest counts, indicating they are the most popular among customers. Other categories, especially 9 to 20, have much lower counts.
- Recommendation: Continue to stock and promote products in categories 1 and 5 as they are in high demand. For categories with lower counts, consider analyzing whether to discontinue certain products, reposition them, or improve their marketing to boost their sales.

Product based Analysis

```
[ ]: df_product = df.groupby("Product_Category")["Purchase"].agg(["mean" , "sum"]).
      ↪reset_index()
     plt.figure(figsize = (12 , 5))
     plt.subplot(1,2,1)
     sns.barplot(data = df_product , x = "Product_Category" , y = "mean")
     plt.title("Mean Purchase vs Product_Category")

     plt.subplot(1,2,2)
     sns.barplot(data = df_product , x = "Product_Category" , y = "sum")
     plt.title("Sum Purchase vs Product_Category")
```

```
[ ]: Text(0.5, 1.0, 'Sum Purchase vs Product_Category')
```



Creating a seprate DataFrame, which include datas for age group between 18 to 45 and product category of 5 , 1 and 8 Only for analysis purpuse

[ ]:
```
df_sep = df[(df["Age"] == "18-25") | (df["Age"] == "26-35") | (df["Age"] ==␣
 ↪"36-45")]
df_sep = df_sep[(df_sep["Product_Category"] == 5) | (df_sep["Product_Category"]␣
 ↪== 1) | (df_sep["Product_Category"] == 8)]
df_sep.head()
```

[ ]:
```
    User_ID Product_ID Gender    Age  Occupation City_Category  \
5   1000003  P00193542      M  26-35          15             A
9   1000005  P00274942      M  26-35          20             A
10  1000005  P00251242      M  26-35          20             A
11  1000005  P00014542      M  26-35          20             A
12  1000005  P00031342      M  26-35          20             A

   Stay_In_Current_City_Years Marital_Status  Product_Category  Purchase
5                           3         Single                 1     15227
9                           1        Married                 8      7871
10                          1        Married                 5      5254
11                          1        Married                 8      3957
12                          1        Married                 8      6073
```

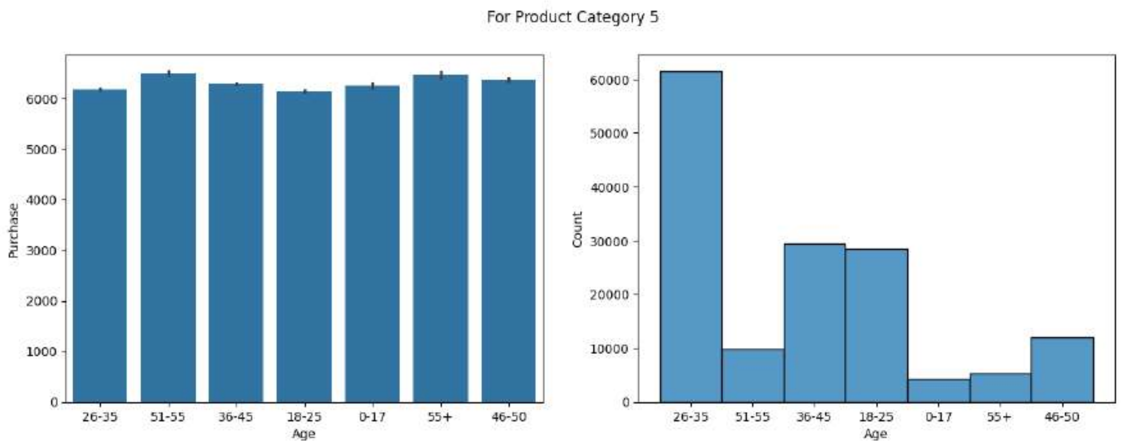
[ ]:
```
df.groupby("Product_Category")["Purchase"].mean().sort_values(ascending =␣
 ↪False).head(10)
```

[ ]:
```
Product_Category
10    19675.570927
7     16365.689600
6     15838.478550
9     15537.375610
15    14780.451828
16    14766.037037
1     13606.218596
14    13141.625739
2     11251.935384
17    10170.759516
Name: Purchase, dtype: float64
```

[ ]:
```
df_sep1 = df[df["Product_Category"] == 5]

plt.figure(figsize = (15,5)).suptitle("For Product Category 5")
plt.subplot(1,2,1)
sns.barplot(data = df_sep1 , x = "Age" , y = "Purchase")
```

```
plt.subplot(1,2,2)
sns.histplot(data = df_sep1 , x = "Age")
```

```
[ ]: <Axes: xlabel='Age', ylabel='Count'>
```



Insights

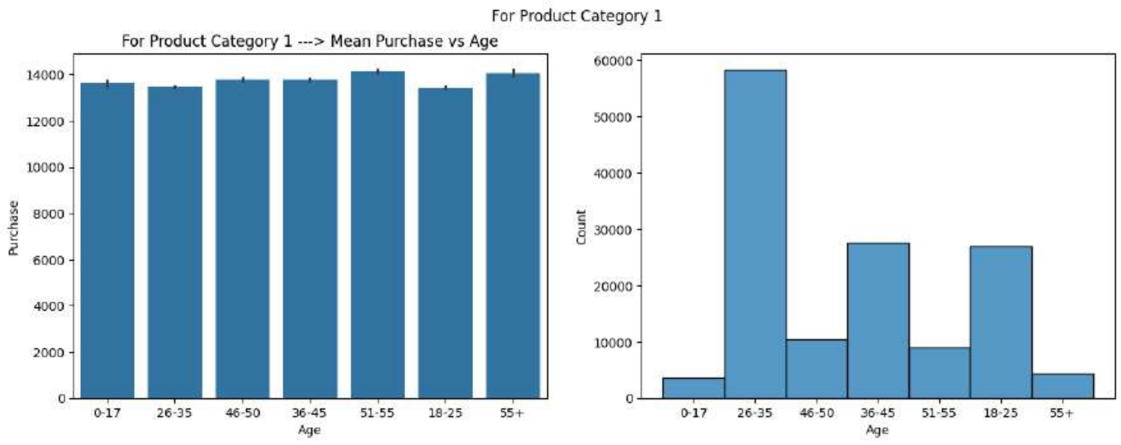
- Mean Purchase is same in all age group.
- Age group 26-35 is purchasing more than any other age group.

Recomendation

- For age group 26-35 addition membership plan can be introduce so that wallmart keep its most buying caterogy to itself.
- Wallmart should also promote this caterogy of products to other age groups for more purchases.

```
[ ]: df_sep2 = df[df["Product_Category"] == 1]
     plt.figure(figsize = (15,5)).suptitle("For Product Category 1")

     plt.subplot(1,2,1)
     sns.barplot(data = df_sep2 , x = "Age" , y = "Purchase")
     plt.title("For Product Category 1 ---> Mean Purchase vs Age")

     plt.subplot(1,2,2)
     sns.histplot(data = df_sep2 , x = "Age")
```

```
[ ]: <Axes: xlabel='Age', ylabel='Count'>
```



Insights

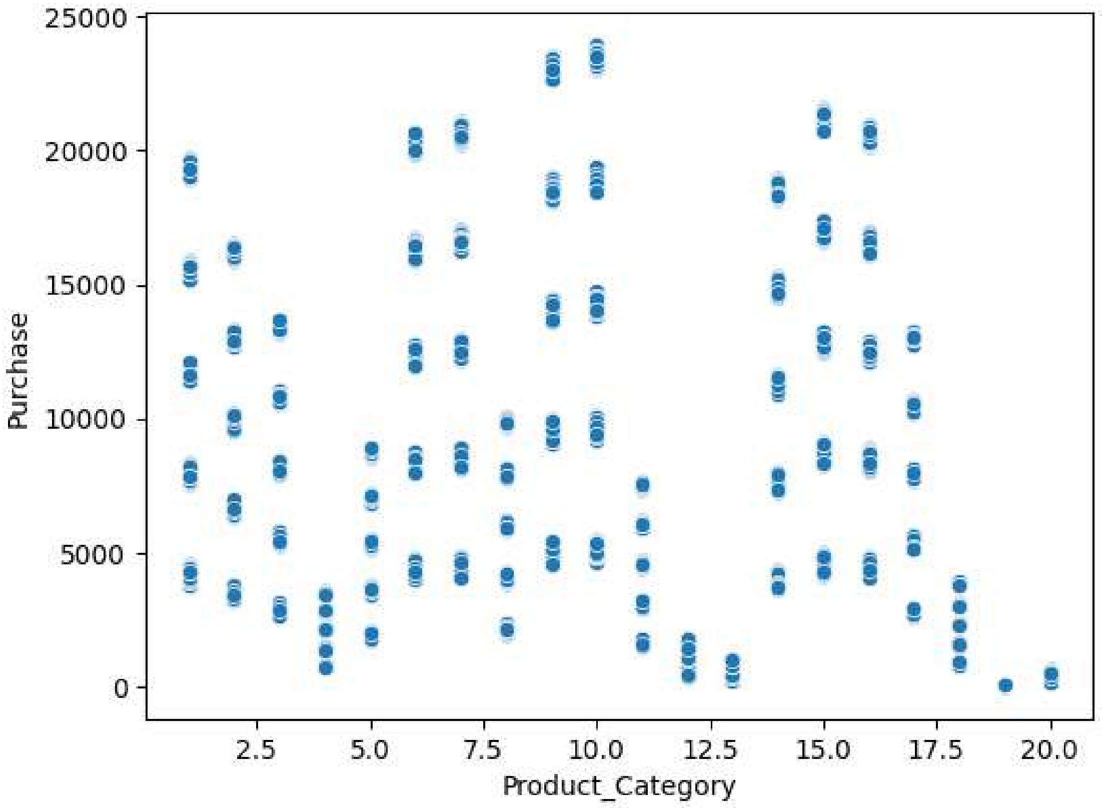
- This caterogy also show same trends as of category 5.

Recomendation

- Any sceme for buying caterogy 1 with category 5 can be indroduce.
- Same promotion can be implemented on both the product Cetegory

[ ]:
```
sns.scatterplot(data = df , x = "Product_Category" , y = "Purchase")
```

[ ]:
```
<Axes: xlabel='Product_Category', ylabel='Purchase'>
```



Insights

- Product Category 1 is the most frequently purchased, followed closely by Categories 5 and 8. This indicates that these categories have broad appeal among customers.
- The mean purchase amount is highest for Product Category 10, followed by Categories 7, 6, 15, and 16. Despite their high average purchase values, these categories are not as frequently bought. less.
- Product Categories 10, 7, 6, 15, and 16 have relatively low purchase frequencies, despite their higher mean purchase amounts. Additionally, Product Categories 4, 7, 9, 12, 13, 14, 17, 18, 19, and 20 are also less frequently purchased compared to other products.

Recommendations

- Product Categories 10, 7, 6, 15, and 16 should be promoted more aggressively, especially targeting segments that can afford higher-end products.
- Walmart should consider reducing the inventory levels of Product Categories 4, 9, 12, 13, 14, 17, 18, 19, and 20 in their stores due to their lower sales volume. In particular, for Categories 19 and 20, they might even consider discontinuing these items in stores altogether, as they may not be worth the shelf space and inventory costs.
- ince Product Categories 1, 5, and 8 are popular, consider cross-selling related products or introducing loyalty programs to capitalize on the existing demand.

Detecting Outlinners

```
[ ]: df_age = df["Age"].value_counts()/df["Age"].count()
     df_age
```

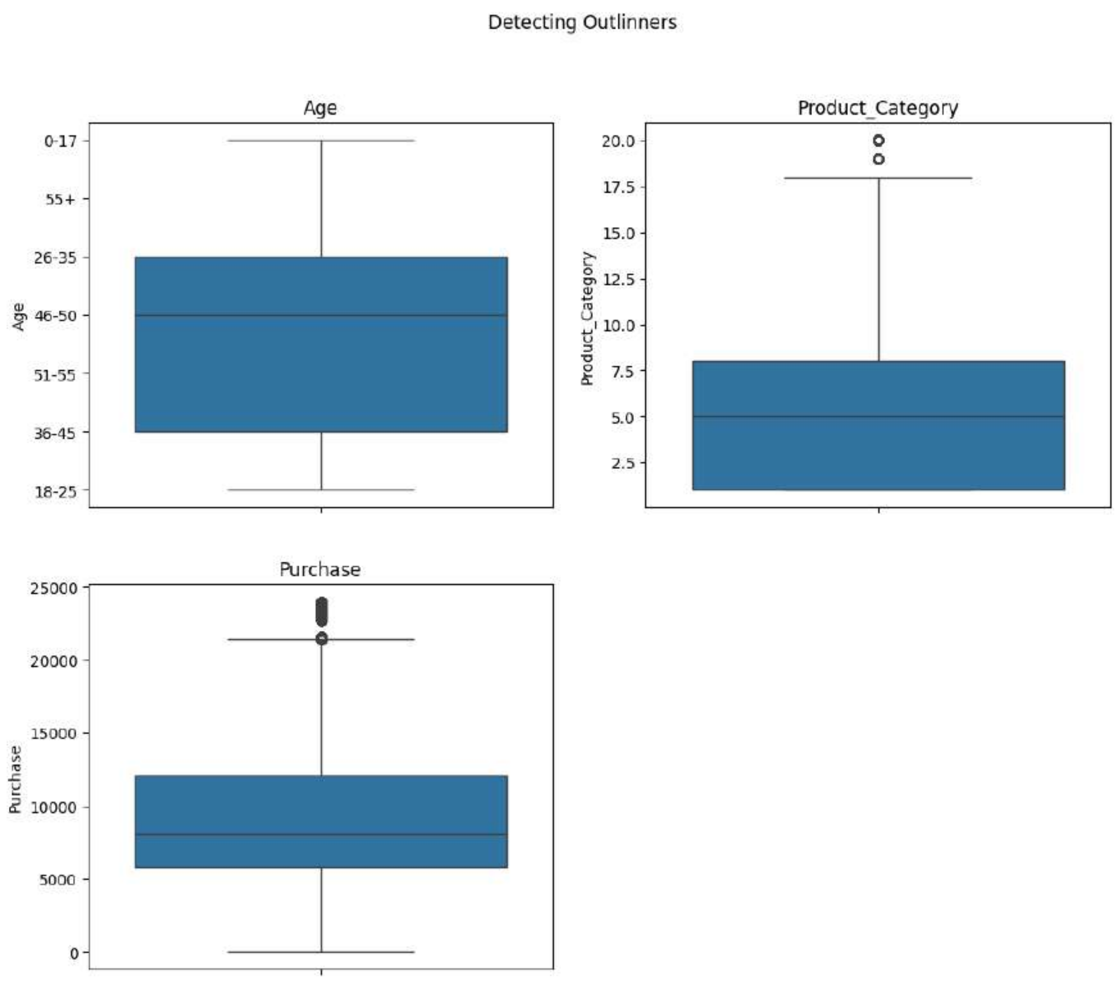
```
[ ]: Age
     26-35    0.399200
     36-45    0.199999
     18-25    0.181178
     46-50    0.083082
     51-55    0.069993
     55+      0.039093
     0-17     0.027455
     Name: count, dtype: float64
```

```
[ ]: df_prod_cat = df["Product_Category"].value_counts()/df["Product_Category"].
       ↪count()
     df_prod_cat.head(5)
```

```
[ ]: Product_Category
     5     0.274390
     1     0.255201
     8     0.207111
     11    0.044153
     2     0.043384
     Name: count, dtype: float64
```

```
[ ]: plt.figure(figsize = (12,10)).suptitle(f"Detecting Outlinners")

     plt.subplot(2,2,1)
     sns.boxplot(df["Age"])
     plt.title("Age")

     plt.subplot(2,2,2)
     sns.boxplot(df["Product_Category"])
     plt.title("Product_Category")

     plt.subplot(2,2,3)
     sns.boxplot(df["Purchase"])
     plt.title("Purchase")
```

```
[ ]: Text(0.5, 1.0, 'Purchase')
```



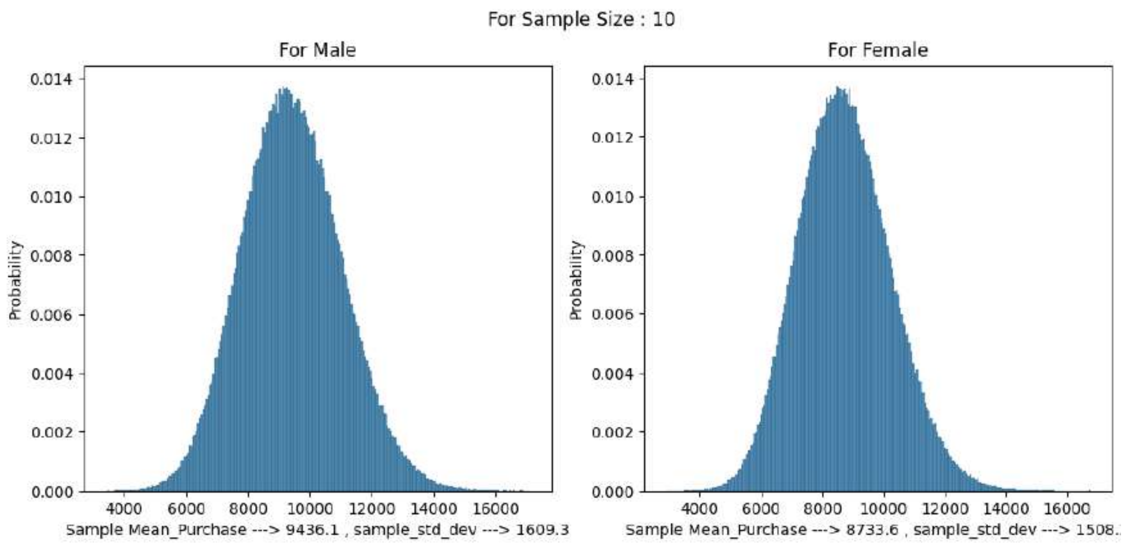
- Product_Category 5 , 1 and 8 makes more than 70% of the sales.
- Age Group of 26-35 has the higest purchase percentage with almost 40%
- People from age 18 to 45 purchase aprox to 80% out of total sales.
- There are no significant outliers in the age distribution, indicating a consistent customer age demographic.
- The product category box plot shows some notable outliers. Categories around 19 and 20 are outliers, indicating that they are less frequently purchased.
- The box plot for purchase amount reveals that the majority of purchase values are concentrated between 5,000 to 15,000, with a few significant outliers exceeding 20,000.

Analysis Based on Gender

Making a prediction about the mean Purchase and CI Based on Gender

Sample Size —> 10

```
[ ]: n = 10
plt.figure(figsize = (12 , 5)).suptitle(f"For Sample Size : {n}")
plt.subplot(1,2,1)
sample_size = np.random.choice(df[df["Gender"] == "M"]["Purchase"] , size =␣
 ↪[500000 , n])
a = np.mean(sample_size , axis = 1)
sns.histplot(a , stat = "probability")
plt.title(f"For Male")
plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
 ↪{np.std(a):.1f}")
m_m = np.mean(a)
s_m = np.std(a)

plt.subplot(1,2,2)
sample_size = np.random.choice(df[df["Gender"] == "F"]["Purchase"] , size =␣
 ↪[500000 , n])
a = np.mean(sample_size , axis = 1)
sns.histplot(a , stat = "probability")
plt.title(f"For Female")
plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
 ↪{np.std(a):.1f}")
m_f = np.mean(a)
s_f = np.std(a)
```



```
[ ]: m = m_m
s = s_m

SE = s / np.sqrt(n)
```

```
# Z_value = 1.64 for 90% CI
print(f"Confidence Interval for **90 percentage** of the mean purchase by␣
↪**Male**\n[{m - 1.64*SE:.2f} , {m + 1.64*SE:.2f}]")
print("-"*50)

# Z_value = 2 for 95% CI
print(f"Confidence Interval for **95 percentage** of the mean purchase by␣
↪**Male**\n[{m - 2*SE:.2f} , {m + 2*SE:.2f}]")
print("-"*50)

# Z_value = 3 for 99.7% CI
print(f"Confidence Interval for **99.7 percentage** of the mean purchase by␣
↪**Male**\n[{m - 3*SE:.2f} , {m + 3*SE:.2f}]")
```

```
Confidence Interval for **90 percentage** of the mean purchase by **Male**
[8601.51 , 10270.68]
--------------------------------------------------
Confidence Interval for **95 percentage** of the mean purchase by **Male**
[8418.31 , 10453.88]
--------------------------------------------------
Confidence Interval for **99.7 percentage** of the mean purchase by **Male**
[7909.41 , 10962.78]
```

[ ]:
```
m = m_f
s = s_f

SE = s / np.sqrt(n)

# Z_value = 1.64 for 90% CI
print(f"Confidence Interval for **90 percentage** of the mean purchase by␣
↪**Female**\n[{m - 1.64*SE:.2f} , {m + 1.64*SE:.2f}]")
print("-"*50)

# Z_value = 2 for 95% CI
print(f"Confidence Interval for **95 percentage** of the mean purchase by␣
↪**Female**\n[{m - 2*SE:.2f} , {m + 2*SE:.2f}]")
print("-"*50)

# Z_value = 3 for 99.7% CI
print(f"Confidence Interval for **99.7 percentage** of the mean purchase by␣
↪**Female**\n[{m - 3*SE:.2f} , {m + 3*SE:.2f}]")
```

```
Confidence Interval for **90 percentage** of the mean purchase by **Female**
[7951.47 , 9515.77]
--------------------------------------------------
Confidence Interval for **95 percentage** of the mean purchase by **Female**
```

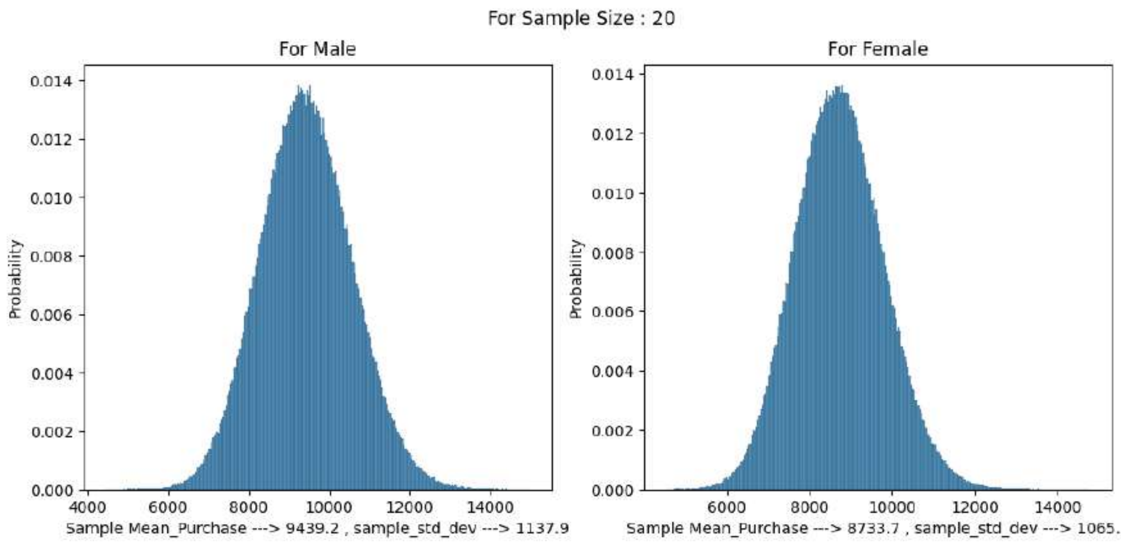
```
[7779.78 , 9687.46]
---------------------------------------------------
Confidence Interval for **99.7 percentage** of the mean purchase by **Female**
[7302.85 , 10164.39]
```

Clearly, the range is overlapping. Hence it would be difficult to concule seperatly that who on average purchase more than the other

Therefore increasing the sample size

Sample Size —> 20

```
[ ]: n = 20
     plt.figure(figsize = (12 , 5)).suptitle(f"For Sample Size : {n}")
     plt.subplot(1,2,1)
     sample_size = np.random.choice(df[df["Gender"] == "M"]["Purchase"] , size =␣
      ↪[500000 , n])
     a = np.mean(sample_size , axis = 1)
     sns.histplot(a , stat = "probability")
     plt.title(f"For Male")
     plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
      ↪{np.std(a):.1f}")
     m_m = np.mean(a)
     s_m = np.std(a)

     plt.subplot(1,2,2)
     sample_size = np.random.choice(df[df["Gender"] == "F"]["Purchase"] , size =␣
      ↪[500000 , n])
     a = np.mean(sample_size , axis = 1)
     sns.histplot(a , stat = "probability")
     plt.title(f"For Female")
     plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
      ↪{np.std(a):.1f}")
     m_f = np.mean(a)
     s_f = np.std(a)
```



```
[ ]: m = m_m
     s = s_m

     SE = s / np.sqrt(n)

     # Z_value = 1.64 for 90% CI
     print(f"Confidence Interval for **90 percentage** of the mean purchase by␣
      ↪**Male**\n[{m - 1.64*SE:.2f} , {m + 1.64*SE:.2f}]")
     print("-"*50)

     # Z_value = 2 for 95% CI
     print(f"Confidence Interval for **95 percentage** of the mean purchase by␣
      ↪**Male**\n[{m - 2*SE:.2f} , {m + 2*SE:.2f}]")
     print("-"*50)

     # Z_value = 3 for 99.7% CI
     print(f"Confidence Interval for **99.7 percentage** of the mean purchase by␣
      ↪**Male**\n[{m - 3*SE:.2f} , {m + 3*SE:.2f}]")
```

```
Confidence Interval for **90 percentage** of the mean purchase by **Male**
[9021.87 , 9856.47]
--------------------------------------------------
Confidence Interval for **95 percentage** of the mean purchase by **Male**
[8930.27 , 9948.08]
--------------------------------------------------
Confidence Interval for **99.7 percentage** of the mean purchase by **Male**
[8675.82 , 10202.53]
```

```
[ ]: m = m_f
     s = s_f

     SE = s / np.sqrt(n)

     # Z_value = 1.64 for 90% CI
     print(f"Confidence Interval for **90 percentage** of the mean purchase by␣
      ↪**Female**\n[{m - 1.64*SE:.2f} , {m + 1.64*SE:.2f}]")
     print("-"*50)

     # Z_value = 2 for 95% CI
     print(f"Confidence Interval for **95 percentage** of the mean purchase by␣
      ↪**Female**\n[{m - 2*SE:.2f} , {m + 2*SE:.2f}]")
     print("-"*50)

     # Z_value = 3 for 99.7% CI
     print(f"Confidence Interval for **99.7 percentage** of the mean purchase by␣
      ↪**Female**\n[{m - 3*SE:.2f} , {m + 3*SE:.2f}]")
```

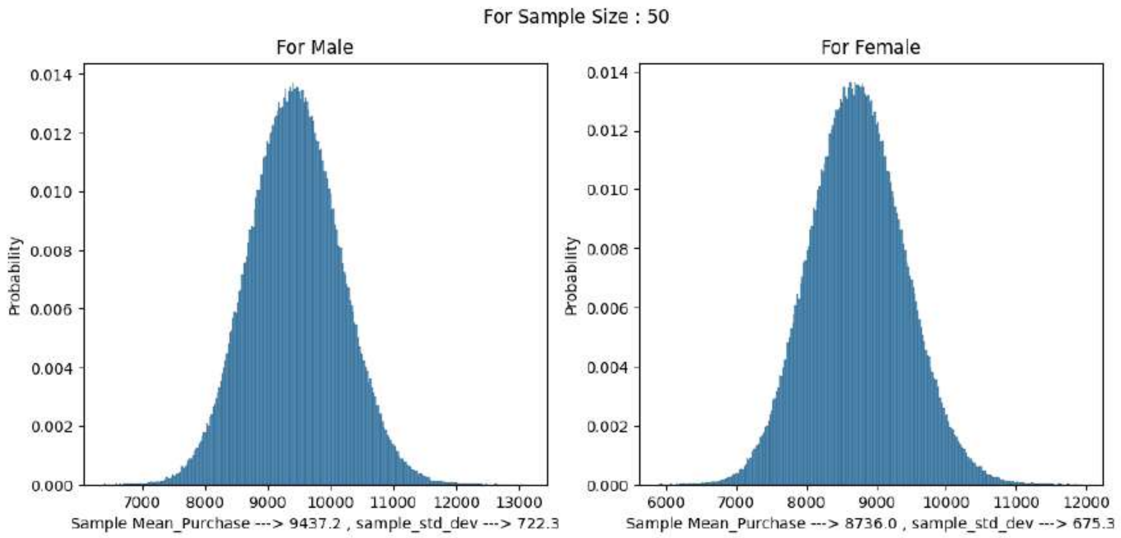
```
Confidence Interval for **90 percentage** of the mean purchase by **Female**
[8342.85 , 9124.46]
--------------------------------------------------
Confidence Interval for **95 percentage** of the mean purchase by **Female**
[8257.06 , 9210.25]
--------------------------------------------------
Confidence Interval for **99.7 percentage** of the mean purchase by **Female**
[8018.76 , 9448.54]
```

Clearly overlapping again , hence increasing the sample size again

Sample Size —> 50

```
[ ]: n = 50
     plt.figure(figsize = (12 , 5)).suptitle(f"For Sample Size : {n}")
     plt.subplot(1,2,1)
     sample_size = np.random.choice(df[df["Gender"] == "M"]["Purchase"] , size =␣
      ↪[500000 , n])
     a = np.mean(sample_size , axis = 1)
     sns.histplot(a , stat = "probability")
     plt.title(f"For Male")
     plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
      ↪{np.std(a):.1f}")
     m_m = np.mean(a)
     s_m = np.std(a)

     plt.subplot(1,2,2)
     n = 50
```

```
sample_size = np.random.choice(df[df["Gender"] == "F"]["Purchase"] , size =␣
 ↪[500000 , n])
a = np.mean(sample_size , axis = 1)
sns.histplot(a , stat = "probability")
plt.title(f"For Female")
plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
 ↪{np.std(a):.1f}")
m_f = np.mean(a)
s_f = np.std(a)
```



Confidence Interval

```
[ ]: from scipy.stats import norm

z_90 = norm.ppf(0.90)
z_95 = norm.ppf(0.95)
z_99 = norm.ppf(0.99)
z_97_5 = norm.ppf(0.975)

print(f"Z-value for 90% CI : {z_90}")
print(f"Z-value for 95% CI : {z_95}")
print(f"Z-value for 99% CI : {z_99}")
print(f"Z-value for 97.5% CI : {z_97_5}")
```

```
Z-value for 90% CI : 1.2815515655446004
Z-value for 95% CI : 1.6448536269514722
Z-value for 99% CI : 2.3263478740408408
Z-value for 97.5% CI : 1.959963984540054
```

[ ]:
```
m = 9438
s = 720.6

SE = s / np.sqrt(n)

# Z_value = 1.64 for 90% CI
print(f"Confidence Interval for **90 percentage** of the mean purchase by␣
 ↪**Male**\n[{m - 1.64*SE:.2f} , {m + 1.64*SE:.2f}]")
print("-"*50)

# Z_value = 2 for 90% CI
print(f"Confidence Interval for **95 percentage** of the mean purchase by␣
 ↪**Male**\n[{m - 2*SE:.2f} , {m + 2*SE:.2f}]")
print("-"*50)

# Z_value = 3 for 99.7% CI
print(f"Confidence Interval for **99.7 percentage** of the mean purchase by␣
 ↪**Male**\n[{m - 3*SE:.2f} , {m + 3*SE:.2f}]")
```

```
Confidence Interval for **90 percentage** of the mean purchase by **Male**
[9270.87 , 9605.13]
--------------------------------------------------
Confidence Interval for **95 percentage** of the mean purchase by **Male**
[9234.18 , 9641.82]
--------------------------------------------------
Confidence Interval for **99.7 percentage** of the mean purchase by **Male**
[9132.28 , 9743.72]
```

[ ]:
```
m = 8733
s = 674.3

SE = s / np.sqrt(n)

# Z_value = 1.64 for 90% CI
print(f"Confidence Interval for **90 percentage** of the mean purchase by␣
 ↪**Female**\n[{m - 1.64*SE:.2f} , {m + 1.64*SE:.2f}]")
print("-"*50)

# Z_value = 2 for 90% CI
print(f"Confidence Interval for **95 percentage** of the mean purchase by␣
 ↪**Female**\n[{m - 2*SE:.2f} , {m + 2*SE:.2f}]")
print("-"*50)

# Z_value = 3 for 99.7% CI
print(f"Confidence Interval for **99.7 percentage** of the mean purchase by␣
 ↪**Female**\n[{m - 3*SE:.2f} , {m + 3*SE:.2f}]")
```

```
Confidence Interval for **90 percentage** of the mean purchase by **Female**
[8576.61 , 8889.39]
--------------------------------------------------
Confidence Interval for **95 percentage** of the mean purchase by **Female**
[8542.28 , 8923.72]
--------------------------------------------------
Confidence Interval for **99.7 percentage** of the mean purchase by **Female**
[8446.92 , 9019.08]
```

Let us test if Gender is independent of Purchase or not

For this we will use 2 sample t-test for testing our Hypothesis

[ ]:
```
male = df[df["Gender"] == "M"]["Purchase"].sample(50000)
female = df[df["Gender"] == "F"]["Purchase"].sample(40000)

"""
H0 (Genaral Belief) : Males and Female are puchasing equally
H1 (Alternative Hypothesis) : Male are purchasing more
"""
#let us consider confidence Region as 90%


from scipy.stats import ttest_ind

alpha = 0.1

z_score , pvalue = ttest_ind(male , female , alternative = "greater")
print(f"P-Value : {pvalue}")

if pvalue < alpha:
  print("Reject the Null Hypothesis , Males are purchasing more than Female")
else:
  print("Accept the Null Hypothesis, Gender does not impact Purchase")
```

```
P-Value : 2.91898748791032 65e-102
Reject the Null Hypothesis , Males are purchasing more than Female
```

Insights

- From the analysis it reveals that males are purchasing significantly more than females. The proportion of purchases by males is around 75%, compared to 25% for females. This suggests a strong gender disparity in purchasing behavior.
- Confidence Intervals For Males:

90% Confidence Interval: [9270.87, 9605.13]

For Females:

90% Confidence Interval: [8576.61, 8889.39]

- The confidence intervals indicate the range within which the true mean purchase amount is likely to fall for each gender.

Males have higher average purchase amounts across all confidence levels compared to Females. The confidence intervals for males are consistently higher than those for females, reinforcing the observed disparity in purchasing behavior. * T-Test Results The t-test results indicate that gender significantly affects purchasing behavior. This statistical evidence supports the observation that there is a meaningful difference between male and female purchasing patterns

Recommendations

- For Males ,consider tailoring marketing strategies to capitalize on their higher purchasing power. This could involve promoting products like men clothing , heavy lifting gym equipments , Bikes accessories etc.
- Explore opportunities to increase engagement and spending among female customers. Wallmart can give attractive offors on Female products.
- Wallmart should develop gender-specific campaigns and initiatives to address the needs and preferences of each gender
- A special team for male and female for their likes and dislikse shall be made.

Analysis Based on Marrital Status

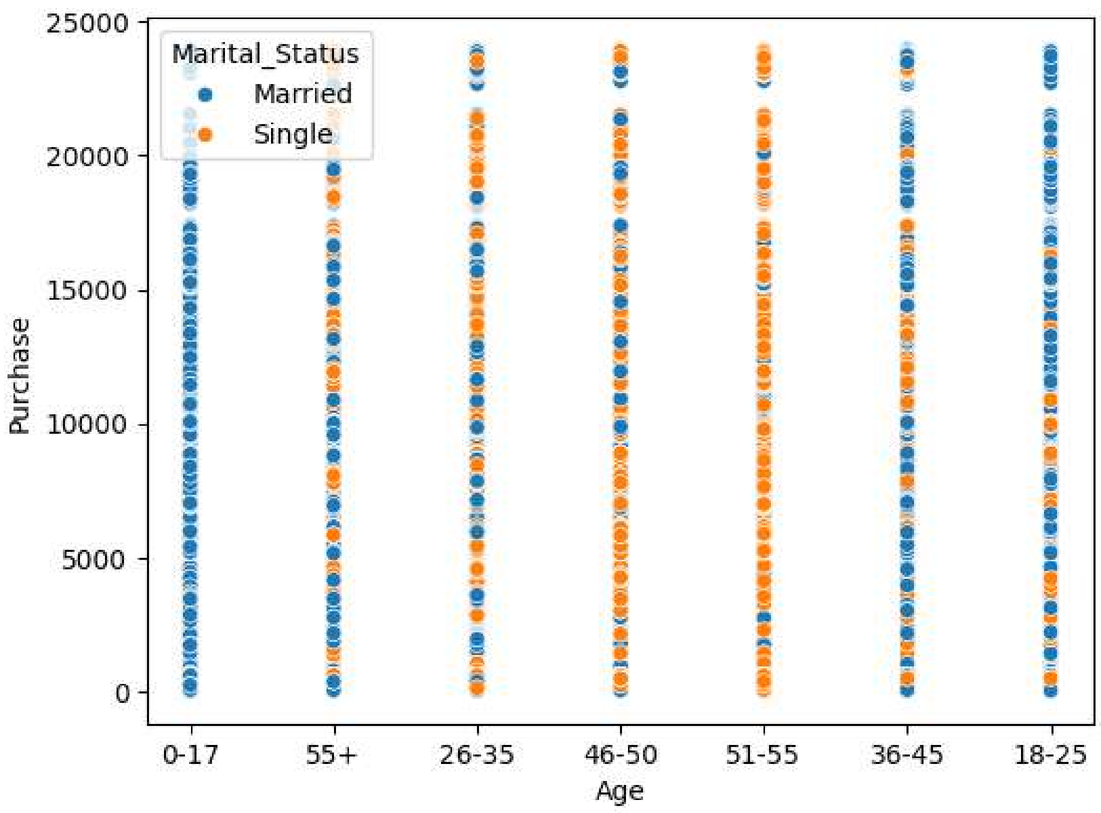
[ ]:
```
df["Marital_Status"].unique()
```

[ ]:
```
array([0, 1])
```

[ ]:
```
#Considering 0 for Married and 1 for Single

def encode(x):
  if x == 0:
    return "Married"
  else:
    return "Single"

df["Marital_Status"] = df["Marital_Status"].apply(encode)
```

[ ]:
```
sns.scatterplot(data = df , x = "Age" , y = "Purchase" , hue = "Marital_Status")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151:
UserWarning: Creating legend with loc="best" can be slow with large amounts of
data.
  fig.canvas.print_figure(bytes_io, **kw)
```



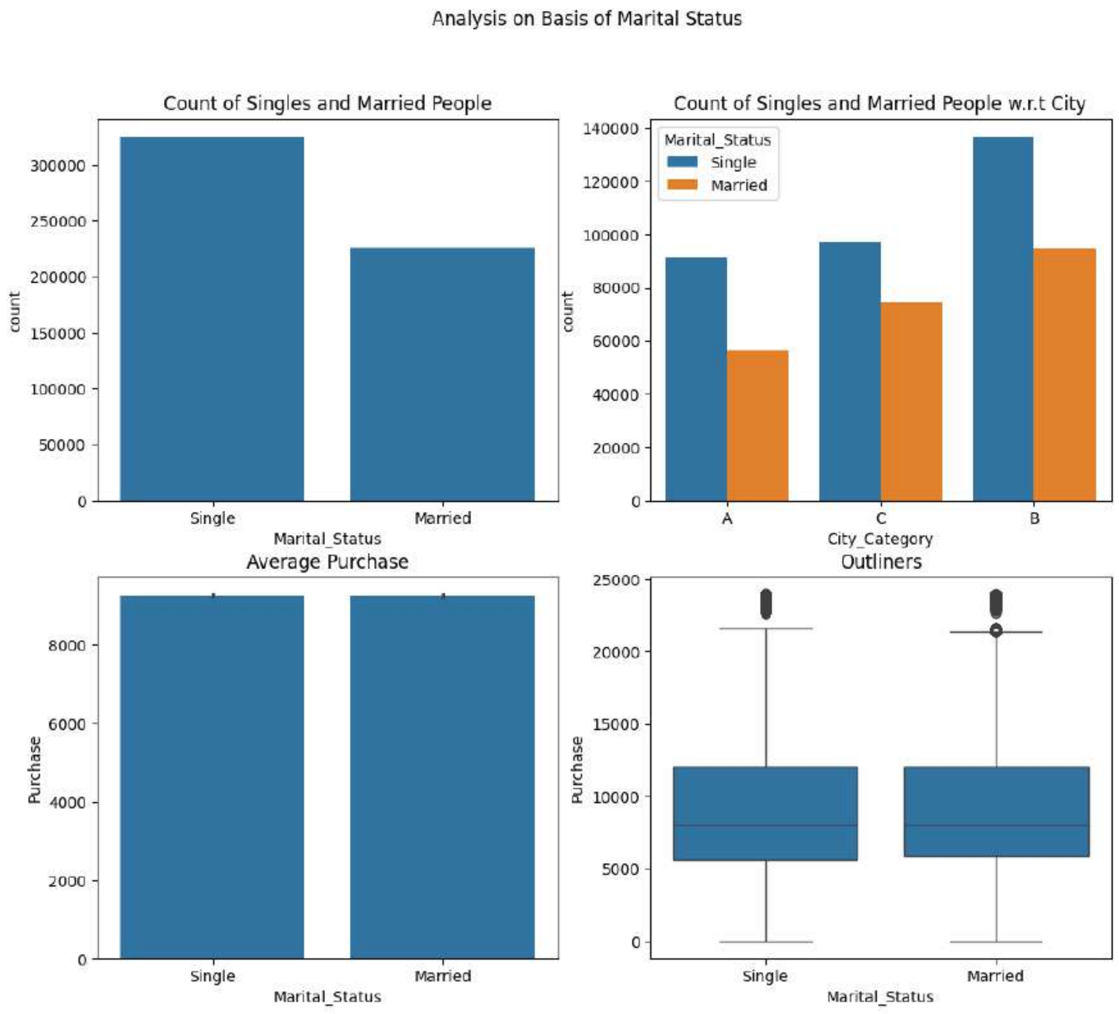
Looking at the graph, we can surely say **our consideration is wrong** and we shall now reverse our study

- Walmart has successfully attracted customers across all age groups, a noteworthy achievement that highlights its broad appeal. It is crucial for Walmart to continue focusing on strategies that resonate with diverse demographics, ensuring sustained engagement and growth in the future.

[ ]:
```
def encode(x):
  if x == "Married":
    return "Single"
  else:
    return "Married"

df["Marital_Status"] = df["Marital_Status"].apply(encode)
```

[ ]:
```
plt.figure(figsize = (12 , 10)).suptitle("Analysis on Basis of Marital Status")

plt.subplot(2,2,1)
plt.title("Count of Singles and Married People")
sns.countplot(x = df["Marital_Status"])
```

```
plt.subplot(2,2,2)
plt.title("Count of Singles and Married People w.r.t City")
sns.countplot(data = df , x = "City_Category" , hue = "Marital_Status")

plt.subplot(2,2,3)
plt.title("Average Purchase")
sns.barplot(x = df["Marital_Status"] , y = df["Purchase"])

plt.subplot(2,2,4)
plt.title("Outliners")
sns.boxplot(x = df["Marital_Status"] , y = df["Purchase"])

plt.show()
```



Insights

- Even Though, more number of singles have bought but the *purchasing power is same of*

Singles and Married people.

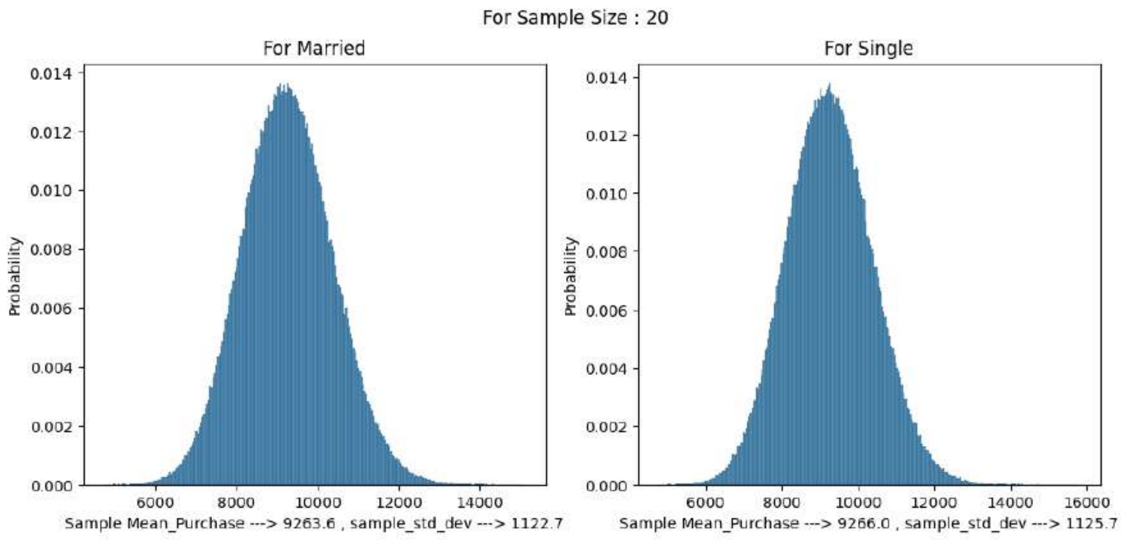
- Outliner is very less and there can be taken out of the data for analysis

Recommendation

- In City_category B , Number of singles purchasings is more, Wallmart can give offers on household items to attract more Married people.
- Develop and promote products and services that cater to both singles and married people, ensuring they meet the needs and preferences of both groups.
- Implement targeted promotions and discounts for singles to capitalize on their higher purchasing frequency.

Making a prediction about the mean Purchase of Married and Single Customers

```
[ ]: n = 20
     plt.figure(figsize = (12 , 5)).suptitle(f"For Sample Size : {n}")
     plt.subplot(1,2,1)
     sample_size = np.random.choice(df[df["Marital_Status"] ==␣
      ↪"Married"]["Purchase"] , size = [500000 , n])
     a = np.mean(sample_size , axis = 1)
     sns.histplot(a , stat = "probability")
     plt.title(f"For Married")
     plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
      ↪{np.std(a):.1f}")
     m_m = np.mean(a)
     s_m = np.std(a)

     plt.subplot(1,2,2)
     sample_size = np.random.choice(df[df["Marital_Status"] == "Single"]["Purchase"]␣
      ↪, size = [500000 , n])
     a = np.mean(sample_size , axis = 1)
     sns.histplot(a , stat = "probability")
     plt.title(f"For Single")
     plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
      ↪{np.std(a):.1f}")
     m_s = np.mean(a)
     s_s = np.std(a)
```



[ ]:
```
m = m_m
s = s_m

SE = s / np.sqrt(n)

# Z_value = 1.64 for 90% CI
print(f"Confidence Interval for **90 percentage** of the mean purchase by␣
 ↪**Married**\n[{m - 1.64*SE:.2f} , {m + 1.64*SE:.2f}]")
print("-"*50)

# Z_value = 2 for 95% CI
print(f"Confidence Interval for **95 percentage** of the mean purchase by␣
 ↪**Married**\n[{m - 2*SE:.2f} , {m + 2*SE:.2f}]")
print("-"*50)

# Z_value = 3 for 99.7% CI
print(f"Confidence Interval for **99.7 percentage** of the mean purchase by␣
 ↪**Married**\n[{m - 3*SE:.2f} , {m + 3*SE:.2f}]")
```

```
Confidence Interval for **90 percentage** of the mean purchase by **Married**
[8851.93 , 9675.35]
--------------------------------------------------
Confidence Interval for **95 percentage** of the mean purchase by **Married**
[8761.56 , 9765.72]
--------------------------------------------------
Confidence Interval for **99.7 percentage** of the mean purchase by **Married**
[8510.52 , 10016.76]
```

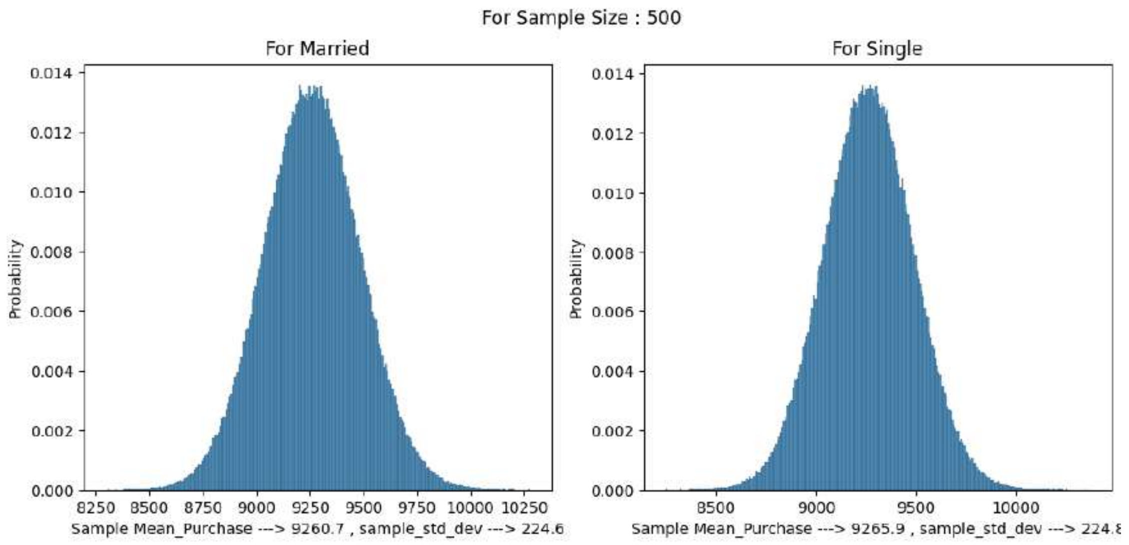
```
[ ]: m = m_s
     s = s_s

     SE = s / np.sqrt(n)

     # Z_value = 1.64 for 90% CI
     print(f"Confidence Interval for **90 percentage** of the mean purchase by␣
      ↪**Single**\n[{m - 1.64*SE:.2f} , {m + 1.64*SE:.2f}]")
     print("-"*50)

     # Z_value = 2 for 95% CI
     print(f"Confidence Interval for **95 percentage** of the mean purchase by␣
      ↪**Single**\n[{m - 2*SE:.2f} , {m + 2*SE:.2f}]")
     print("-"*50)

     # Z_value = 3 for 99.7% CI
     print(f"Confidence Interval for **99.7 percentage** of the mean purchase by␣
      ↪**Single**\n[{m - 3*SE:.2f} , {m + 3*SE:.2f}]")
```

```
Confidence Interval for **90 percentage** of the mean purchase by **Single**
[8853.19 , 9678.83]
--------------------------------------------------
Confidence Interval for **95 percentage** of the mean purchase by **Single**
[8762.57 , 9769.45]
--------------------------------------------------
Confidence Interval for **99.7 percentage** of the mean purchase by **Single**
[8510.84 , 10021.17]
```

For sample size of 20 , CI is overlapping. let us take a larger sample to compare

```
[ ]: n = 500
     plt.figure(figsize = (12 , 5)).suptitle(f"For Sample Size : {n}")
     plt.subplot(1,2,1)
     sample_size = np.random.choice(df[df["Marital_Status"] ==␣
      ↪"Married"]["Purchase"] , size = [500000 , n])
     a = np.mean(sample_size , axis = 1)
     sns.histplot(a , stat = "probability")
     plt.title(f"For Married")
     plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
      ↪{np.std(a):.1f}")
     m_m = np.mean(a)
     s_m = np.std(a)

     plt.subplot(1,2,2)
     sample_size = np.random.choice(df[df["Marital_Status"] == "Single"]["Purchase"]␣
      ↪, size = [500000 , n])
     a = np.mean(sample_size , axis = 1)
```

```
sns.histplot(a , stat = "probability")
plt.title(f"For Single")
plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
↪{np.std(a):.1f}")
m_s = np.mean(a)
s_s = np.std(a)
```



```
[ ]: m = m_m
     s = s_m

     SE = s / np.sqrt(n)

     # Z_value = 1.64 for 90% CI
     print(f"Confidence Interval for **90 percentage** of the mean purchase by␣
     ↪**Married**\n[{m - 1.64*SE:.2f} , {m + 1.64*SE:.2f}]")
     print("-"*50)

     # Z_value = 2 for 95% CI
     print(f"Confidence Interval for **95 percentage** of the mean purchase by␣
     ↪**Married**\n[{m - 2*SE:.2f} , {m + 2*SE:.2f}]")
     print("-"*50)

     # Z_value = 3 for 99.7% CI
     print(f"Confidence Interval for **99.7 percentage** of the mean purchase by␣
     ↪**Married**\n[{m - 3*SE:.2f} , {m + 3*SE:.2f}]")
```

```
Confidence Interval for **90 percentage** of the mean purchase by **Married**
[9244.28 , 9277.22]
--------------------------------------------------
```

```
Confidence Interval for **95 percentage** of the mean purchase by **Married**
[9240.66 , 9280.83]
--------------------------------------------------
Confidence Interval for **99.7 percentage** of the mean purchase by **Married**
[9230.62 , 9290.88]
```

[ ]:
```
m = m_s
s = s_s

SE = s / np.sqrt(n)

# Z_value = 1.64 for 90% CI
print(f"Confidence Interval for **90 percentage** of the mean purchase by␣
 ↪**Single**\n[{m - 1.64*SE:.2f} , {m + 1.64*SE:.2f}]")
print("-"*50)

# Z_value = 2 for 95% CI
print(f"Confidence Interval for **95 percentage** of the mean purchase by␣
 ↪**Single**\n[{m - 2*SE:.2f} , {m + 2*SE:.2f}]")
print("-"*50)

# Z_value = 3 for 99.7% CI
print(f"Confidence Interval for **99.7 percentage** of the mean purchase by␣
 ↪**Single**\n[{m - 3*SE:.2f} , {m + 3*SE:.2f}]")
```

```
Confidence Interval for **90 percentage** of the mean purchase by **Single**
[9249.41 , 9282.39]
--------------------------------------------------
Confidence Interval for **95 percentage** of the mean purchase by **Single**
[9245.79 , 9286.01]
--------------------------------------------------
Confidence Interval for **99.7 percentage** of the mean purchase by **Single**
[9235.74 , 9296.07]
```

Let us test if Marital Status is impacting the purchase

For this, we will use 2 sample t-test for testing our Hypothesis

[ ]:
```
Married = df[df["Marital_Status"] == "Married"]["Purchase"].sample(8000)
Single = df[df["Marital_Status"] == "Single"]["Purchase"].sample(10000)

"""
H0 (Genaral Belief) : Marital Status are independent of puchase.
H1 (Alternative Hypothesis) : Marital Status does impact the Purchase
"""
#let us consider confidence Region as 90%
```

```python
from scipy.stats import ttest_ind

alpha = 0.1

z_score , pvalue = ttest_ind(Married , Single , alternative = "two-sided")
print(f"P-Value : {pvalue}")

if pvalue < alpha:
  print("Reject the Null Hypothesis , Marital Status does impact the Purchase")
else:
  print("Accept the Null Hypothesis, Marital Status are independent of puchase")
```

```
P-Value : 0.8713090618386401
Accept the Null Hypothesis, Marital Status are independent of puchase
```

Insights

- For Singles at 90% CI, Purchase population mean falls in the range of [9249.92 , 9282.95]
- For Couples at 90% CI, Purchase population mean falls in the range of [9244.69 , 9277.57]
- Clearly the range of Confidence Interval is overlapping by a huge proportion, even if we take a sample size of 500.
- Therefore we cant compare the Purchasing Amount on the basis of Marrital Status as also concluded by our Hypothesis testing.

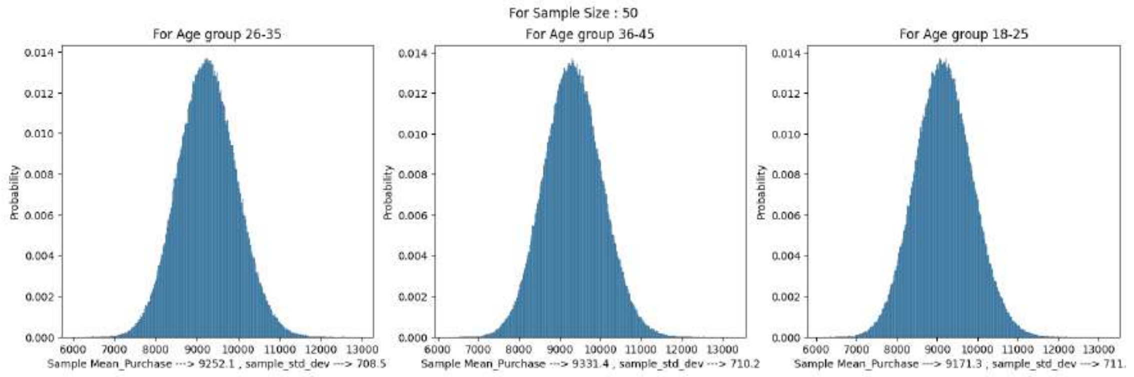
Recommendations

- Products that cater specifically to the needs of singles and couples should be available at Walmart to capture this segment of the market. However, the company should ensure that this focus does not overshadow broader, more inclusive strategies that target larger, more diverse customer bases.

Let us make CLT and CI for age group of 26-35 , 36-45 , 18-25 to get some insights

```python
[ ]: n = 50
plt.figure(figsize = (18 , 5)).suptitle(f"For Sample Size : {n}")
plt.subplot(1,3,1)
sample_size = np.random.choice(df[df["Age"] == "26-35"]["Purchase"] , size =␣
  ↪[500000 , n])
a = np.mean(sample_size , axis = 1)
sns.histplot(a , stat = "probability")
plt.title(f"For Age group 26-35")
plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
  ↪{np.std(a):.1f}")
m_a1 = np.mean(a)
s_a1 = np.std(a)

plt.subplot(1,3,2)
```

```
sample_size = np.random.choice(df[df["Age"] == "36-45"]["Purchase"] , size =␣
↪[500000 , n])
a = np.mean(sample_size , axis = 1)
sns.histplot(a , stat = "probability")
plt.title(f"For Age group 36-45")
plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
↪{np.std(a):.1f}")
m_a2 = np.mean(a)
s_a2 = np.std(a)

plt.subplot(1,3,3)
sample_size = np.random.choice(df[df["Age"] == "18-25"]["Purchase"] , size =␣
↪[500000 , n])
a = np.mean(sample_size , axis = 1)
sns.histplot(a , stat = "probability")
plt.title(f"For Age group 18-25")
plt.xlabel(f"Sample Mean_Purchase ---> {np.mean(a):.1f} , sample_std_dev --->␣
↪{np.std(a):.1f}")
m_a3 = np.mean(a)
s_a3 = np.std(a)
```



[ ]:
```
m = m_a1
s = s_a1

SE = s / np.sqrt(n)

# Z_value = 1.64 for 90% CI
print(f"Confidence Interval for **90 percentage** for Age group 26-35\n[{m - 1.
↪64*SE:.2f} , {m + 1.64*SE:.2f}]")
print("-"*50)

m = m_a2
s = s_a2
```

```
SE = s / np.sqrt(n)

# Z_value = 1.64 for 90% CI
print(f"Confidence Interval for **90 percentage** for Age group 36-45\n[{m - 1.
↪64*SE:.2f} , {m + 1.64*SE:.2f}]")
print("-"*50)

m = m_a3
s = s_a3

SE = s / np.sqrt(n)

# Z_value = 1.64 for 90% CI
print(f"Confidence Interval for **90 percentage** for Age group 18-25\n[{m - 1.
↪64*SE:.2f} , {m + 1.64*SE:.2f}]")
```

```
Confidence Interval for **90 percentage** for Age group 26-35
[9087.80 , 9416.43]
--------------------------------------------------
Confidence Interval for **90 percentage** for Age group 36-45
[9166.69 , 9496.12]
--------------------------------------------------
Confidence Interval for **90 percentage** for Age group 18-25
[9006.23 , 9336.40]
```

Insights:

- Since the 90% confidence intervals for the mean purchase amounts across the age groups of 18-25, 26-35, and 36-45 overlap significantly, **it suggests that there is no clear distinction in purchasing behavior based on age among these groups.** But still we can draw some insights from the data.
- Age Group 26-35: The 90% confidence interval for this age group's average purchase amount is between $9,087.58 and $9,416.
- Age Group 36-45: The confidence interval for this group is slightly higher, ranging from $9,166.80 to $9,495.75. This suggests that buyers in this age group generally spend a bit more compared to the 26-35 age group.
- Age Group 18-25: This group has a confidence interval ranging from $9,004.20 to $9,334.49. **Although younger, their spending habits are somewhat close to the 26-35 age group.**

Recommendations:

- For the 26-35 and 18-25 age groups, consider implementing loyalty programs that offer rewards or discounts on repeated purchases. This could encourage more frequent purchases and increase overall spending within these groups.
- As for Age group of 18-25 shows a huge potential and therefore, a special team promoting

segment of products like sports , adventure should introduce if not done already.

- The most frequent buyers of Wallmart is in the age gap of 26-35 and with the highest purchase. Wallmart should focus on tailored marketing strategies and product offerings that cater specifically to this demographic to maximize engagement and drive sales.

**

```
[ ]: a = df[(df["Purchase"] >= 9087) & (df["Purchase"] <= 9416)]["Product_Category"].
      ↪unique()
     print(f"For Age group 26-35 , Product_Category that falls under:-")
     for i in range(len(a)):
       print(f"Prod_Category : {a[i]}" , end = "    ")

     print("\n")

     print("-"*50)
     a = df[(df["Purchase"] >= 9166) & (df["Purchase"] <= 9495)]["Product_Category"].
      ↪unique()
     print(f"For Age group 36-45 , Product_Category that falls under:-")
     for i in range(len(a)):
       print(f"Prod_Category : {a[i]}" , end = "    ")

     print("\n")

     print("-"*50)
     a = df[(df["Purchase"] >= 9004) & (df["Purchase"] <= 9334)]["Product_Category"].
      ↪unique()
     print(f"For Age group 18-25 , Product_Category that falls under:-")
     for i in range(len(a)):
       print(f"Prod_Category : {a[i]}" , end = "    ")
```

```
For Age group 26-35 , Product_Category that falls under:-
Prod_Category : 10    Prod_Category : 15    Prod_Category : 9

--------------------------------------------------
For Age group 36-45 , Product_Category that falls under:-
Prod_Category : 10    Prod_Category : 9

--------------------------------------------------
For Age group 18-25 , Product_Category that falls under:-
Prod_Category : 10    Prod_Category : 15    Prod_Category : 9
```

Insights

- Above are the Product Category that falls under the Age group mean Purchase.
- Product Category 9 , 10 and 15 are found to fall under these Category.
- While these categories show higher mean purchases, the total volume of purchases for these products is relatively lower compared to other categories

Recommendation

- Focus marketing efforts on Categories 9, 10, and 15 specifically for the age group of 18 to 45 years. This demographic has shown significant potential in terms of purchasing power and interest in these categories.
- Implement promotional strategies such as discounts, bundles, and loyalty programs targeted at this age group. This could help boost the total purchase volume of these categories.
- Ensure that products in Categories 9, 10, and 15 are prominently featured in both online and in-store settings, particularly during peak shopping seasons when the 18-45 age group is most active.

Let us check does Age impact Purchase or nor

For this we will use , one-way ANOVA Testing (f-testing)

```
[ ]: df["Age"].unique()
```

```
[ ]: array(['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25'],
           dtype=object)
```

```
[ ]: age_1 = df[df["Age"] == "0-17"]["Purchase"].sample(5000)
     age_2 = df[df["Age"] == "18-25"]["Purchase"].sample(5000)
     age_3 = df[df["Age"] == "26-35"]["Purchase"].sample(5000)
     age_4 = df[df["Age"] == "36-45"]["Purchase"].sample(5000)
     age_5 = df[df["Age"] == "46-50"]["Purchase"].sample(5000)
     age_6 = df[df["Age"] == "51-55"]["Purchase"].sample(5000)
     age_7 = df[df["Age"] == "55+"]["Purchase"].sample(5000)

     from scipy.stats import f_oneway

     alpha = 0.1

     f_score , pvalue = f_oneway(age_1 , age_2 , age_3 , age_4 , age_5 , age_6 ,␣
      ↪age_7)
     print(f"P-Value : {pvalue}")

     if pvalue < alpha:
       print("Reject the Null Hypothesis , Age does impact the Purchase")
     else:
       print("Accept the Null Hypothesis, Age does not impact the Purchase")
```

```
P-Value : 2.3875727407629604e-09
Reject the Null Hypothesis , Age does impact the Purchase
```

Age 18 to 45 forms the most percentage of the purchase. let remove those age group and see if still age is depended on purchase or not

```
[ ]: f_score , pvalue = f_oneway(age_1 , age_5 , age_6 , age_7)
print(f"P-Value : {pvalue}")

if pvalue < alpha:
  print("Reject the Null Hypothesis , Age does impact the Purchase")
else:
  print("Accept the Null Hypothesis, Age does not impact the Purchase")
```

```
P-Value : 3.5391888045947554e-10
Reject the Null Hypothesis , Age does impact the Purchase
```

Summary of Insights and Recommendations

Insights

- Product Category 1 is the most purchased, followed by Categories 5 and 8. Categories 10, 7, 6, 15, and 16 have the highest mean purchase values but are purchased less frequently.

Categories 4, 7, 9, 12, 13, 14, 17, 18, 19, and 20 have relatively low purchase frequencies compared to other categories.

- The most frequent buyers are in the age group of 26-35, who also make the highest purchases.
- The 18-25 age group shows significant potential as future customers, particularly for products related to sports and adventure.
- Confidence intervals for the mean purchase amounts across the 18-25, 26-35, and 36-45 age groups overlap, indicating that purchasing behavior does not significantly differ among these group
- Males make up the majority of Walmart's customers, particularly in the 26-35 age group.
- There is an equal purchasing power between singles and married individuals, even though more singles make purchases.
- The majority of Walmart's customers are male, significantly outnumbering female customers.
- Despite singles making more purchases, the purchasing power between singles and married individuals is relatively equal. This indicates that both groups have similar spending capabilities, though they may prioritize different products or categories.

Recommendations

- Walmart should reduce the inventory levels of Product Categories 4, 9, 12, 13, 14, 17, 18, 19, and 20 due to their lower sales volume. Consider discontinuing Categories 19 and 20 from store shelves to optimize space and reduce costs.
- Increase promotional efforts for Product Categories 10, 7, 6, 15, and 16, particularly targeting the 18-45 age group, as these products have high mean purchase values but low sales volume.
- Focus on the 26-35 age group, who are the most frequent buyers with the highest purchase amounts, by offering products and services that cater to their preferences.
- Create specialized marketing campaigns for the 18-25 age group, emphasizing products related to sports, adventure, and lifestyle, as this group shows high potential for future growth.

- Offer products that cater specifically to singles and couples, but ensure that this is not the primary focus of the company's product strategy.
- Enhance in-store and online shopping experiences to cater to the purchasing behaviors of both singles and married customers, leveraging the equal purchasing power across these groups.
- Walmart should consider launching gender-specific marketing campaigns to capitalize on the existing male customer base while working to attract more female customers. This can be achieved by promoting products that appeal to different genders through targeted ads, product placements, and store layouts.

[71]:
```
df.head()
```

[71]:
```
   User_ID Product_ID Gender   Age  Occupation City_Category  \
0  1000001  P00069042      F  0-17          10             A
1  1000001  P00248942      F  0-17          10             A
2  1000001  P00087842      F  0-17          10             A
3  1000001  P00085442      F  0-17          10             A
4  1000002  P00285442      M   55+          16             C

  Stay_In_Current_City_Years Marital_Status  Product_Category  Purchase
0                          2         Single                 3      8370
1                          2         Single                 1     15200
2                          2         Single                12      1422
3                          2         Single                12      1057
4                         4+         Single                 8      7969
```

[ ]:
```

```