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

file_path = '/content/walmart_data.txt'

df = pd.read_csv(file_path)

df.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product
0	1000001	P00069042	F	0-17	10	A	2	0	
1	1000001	P00248942	F	0-17	10	A	2	0	
2	1000001	P00087842	F	0-17	10	A	2	0	
3	1000001	P00085442	F	0-17	10	A	2	0	
4	1000002	P00285442	M	55+	16	C	4+	0	

1. Import the dataset and do usual data analysis steps like checking the structure & characteristics of the dataset.

```
# getting the summary of the data

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
```

```
# Missing values :

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

	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

dtype: int64

```
# Summary statistics

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

```
# Checking dataset characteristics :

# column names and datatypes

print(df.columns)
print(df.dtypes)
```

```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category',
       'Purchase'],
      dtype='object')
User_ID                        int64
Product_ID                    object
Gender                        object
Age                           object
Occupation                     int64
City_Category                 object
Stay_In_Current_City_Years    object
Marital_Status                 int64
Product_Category               int64
Purchase                       int64
```

```
dtype: object
```

```
# unique values in each column :

for col in df.columns:

  print(f"{col} : {df[col].nunique()} unique values")
```

```
User_ID : 5891 unique values
Product_ID : 3631 unique values
Gender : 2 unique values
Age : 7 unique values
Occupation : 21 unique values
City_Category : 3 unique values
Stay_In_Current_City_Years : 5 unique values
Marital_Status : 2 unique values
Product_Category : 20 unique values
Purchase : 18105 unique values
```

```
# Shape of the matrix :

print(df.shape)
```

```
(550068, 10)
```

```
# Duplicates :

df.duplicated().sum()
```

```
0
```

INSIGHTS :

1. Null values in any of the column is 0.
2. dtype in this dataset consist of "object" and "int64"
3. All the columns in the dataset consist of unique values
4. Shape of the dataset is (550068, 10).

RECOMMENDATIONS : None

2. **Detect Null values & Outliers (using boxplot, "describe" method by checking the difference between mean and median, isnull etc.**) **bold text**

```
# Detect null values :

null_counts = df.isnull().sum()
print("Missing values in each column:\n",null_counts)
```

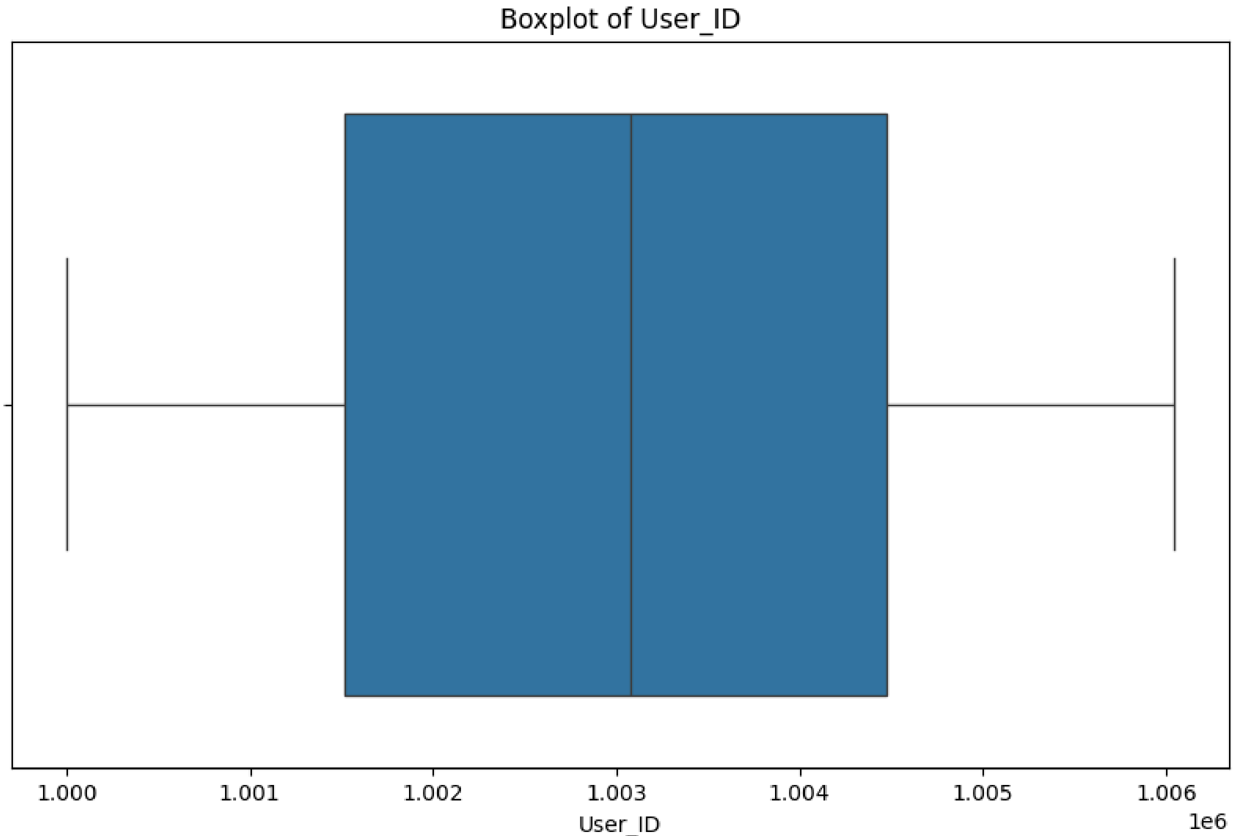
```
Missing values in each column:
 User_ID                       0
Product_ID                    0
Gender                        0
Age                           0
Occupation                    0
City_Category                 0
Stay_In_Current_City_Years    0
Marital_Status                0
Product_Category              0
Purchase                      0
```

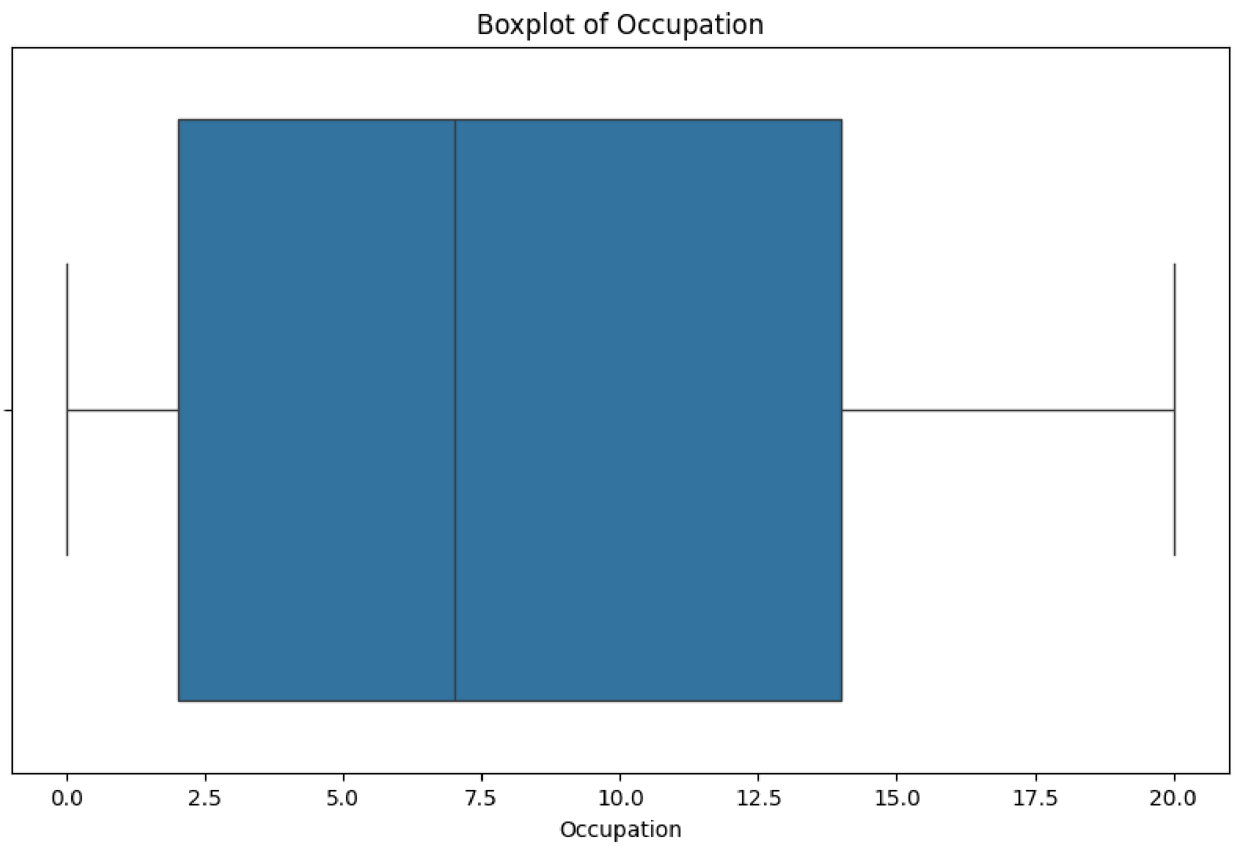
```
dtype: int64
```

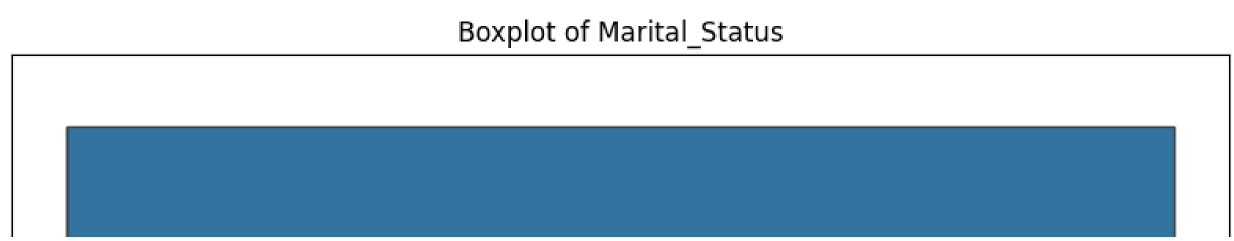
```
# Summary of rows with missing values :

missing_values = df[df.isnull().any(axis=1)]

print("rows with missing values:\n",missing_values)
```

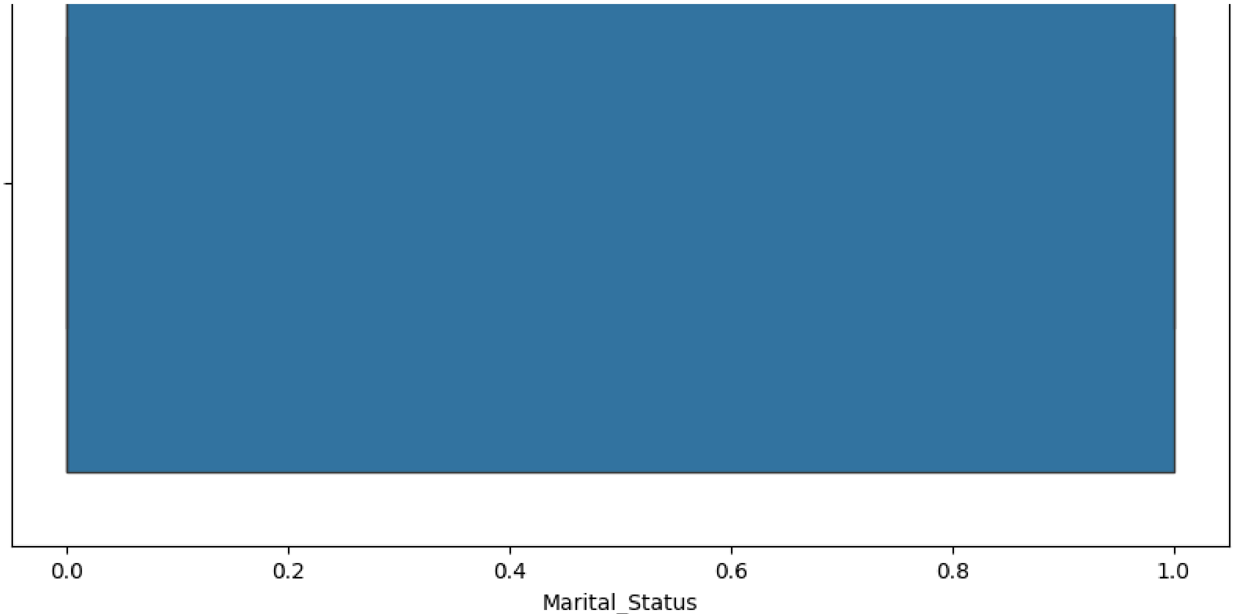
```
rows with missing values:
 Empty DataFrame
Columns: [User_ID, Product_ID, Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status,
Index: []
```

```
# Detecting outliers :

# Create boxplots for each numerical column to detect outliers
for column in df.select_dtypes(include=['float64', 'int64']).columns:
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=df[column])
    plt.title(f'Boxplot of {column}')
    plt.show()
```

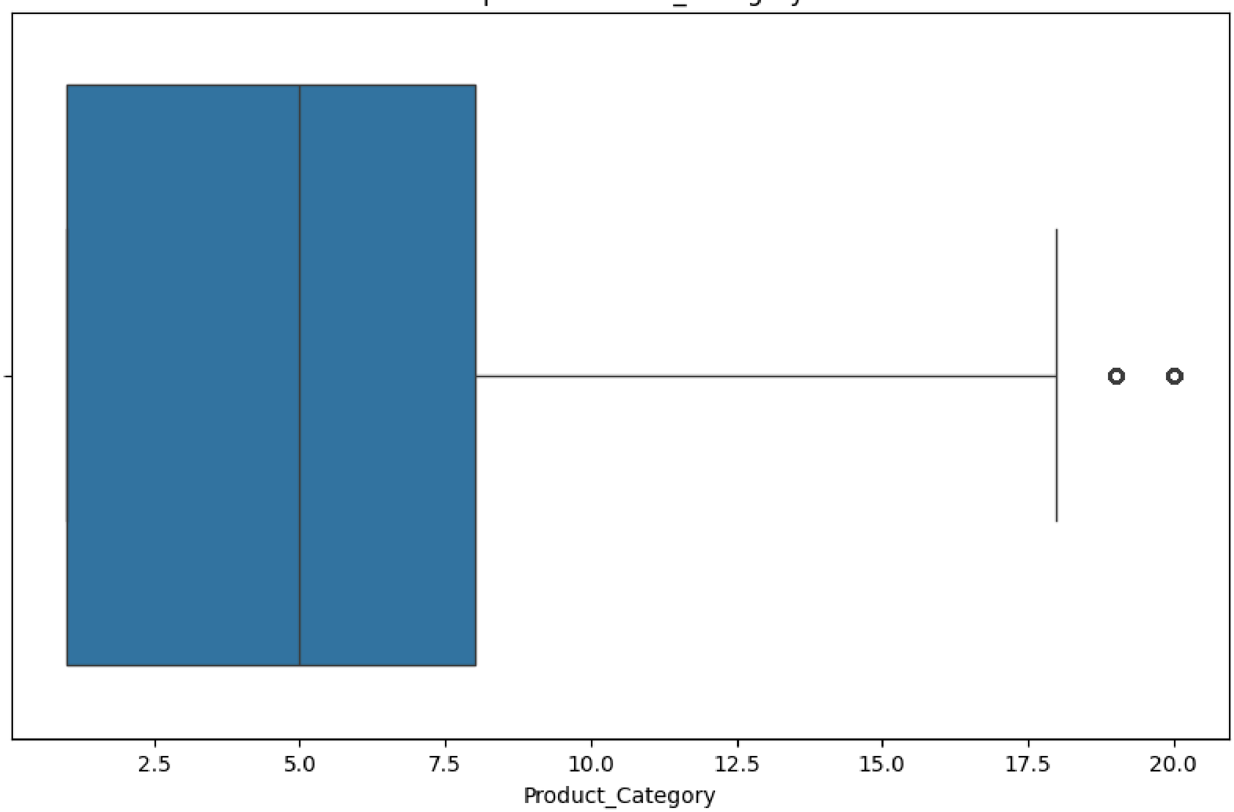

Boxplot of User_ID
1.000
1.001
1.002
1.003
1.004
1.005
1.006
1e6
User_ID


Boxplot of Occupation
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Occupation

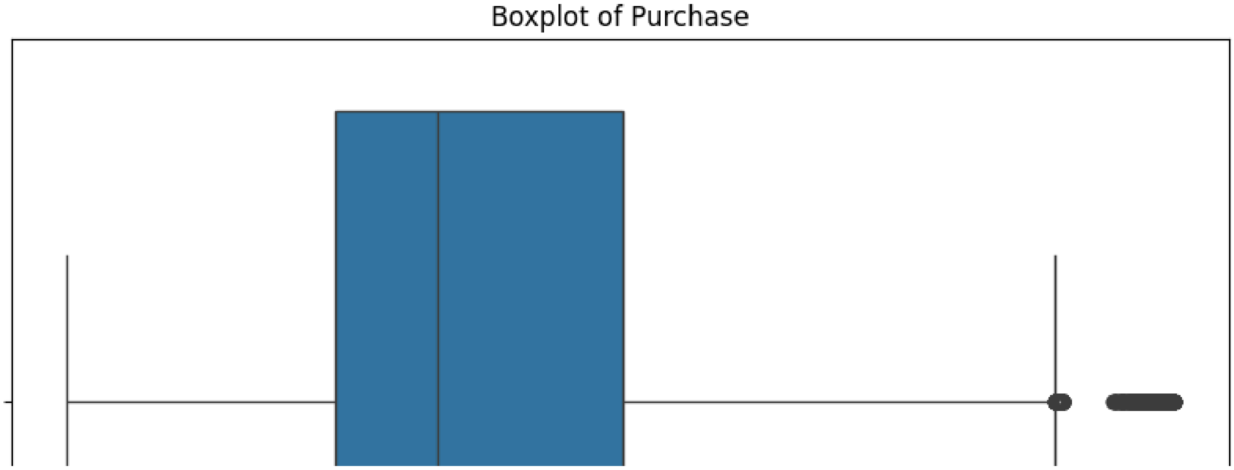

Boxplot of Marital_Status

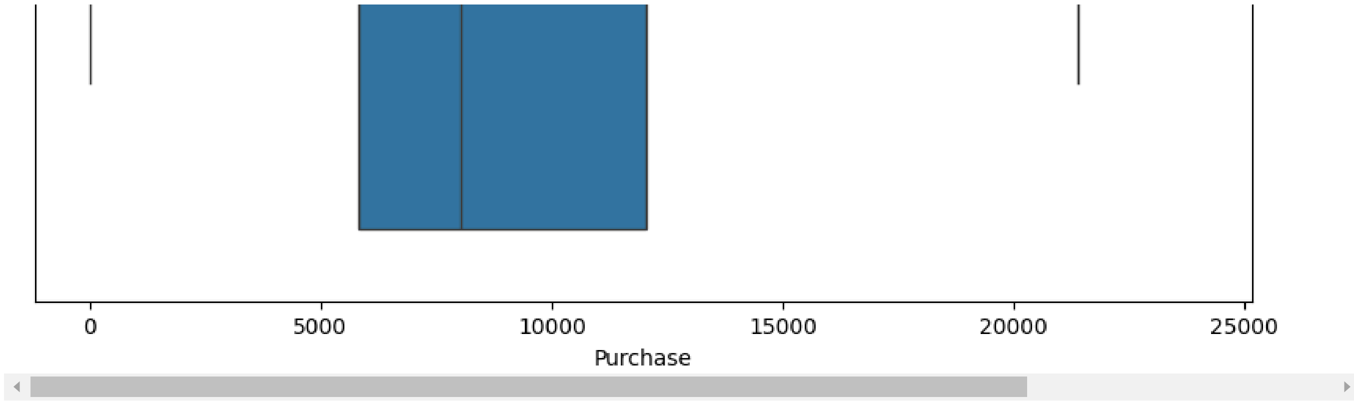


Boxplot of Product_Category



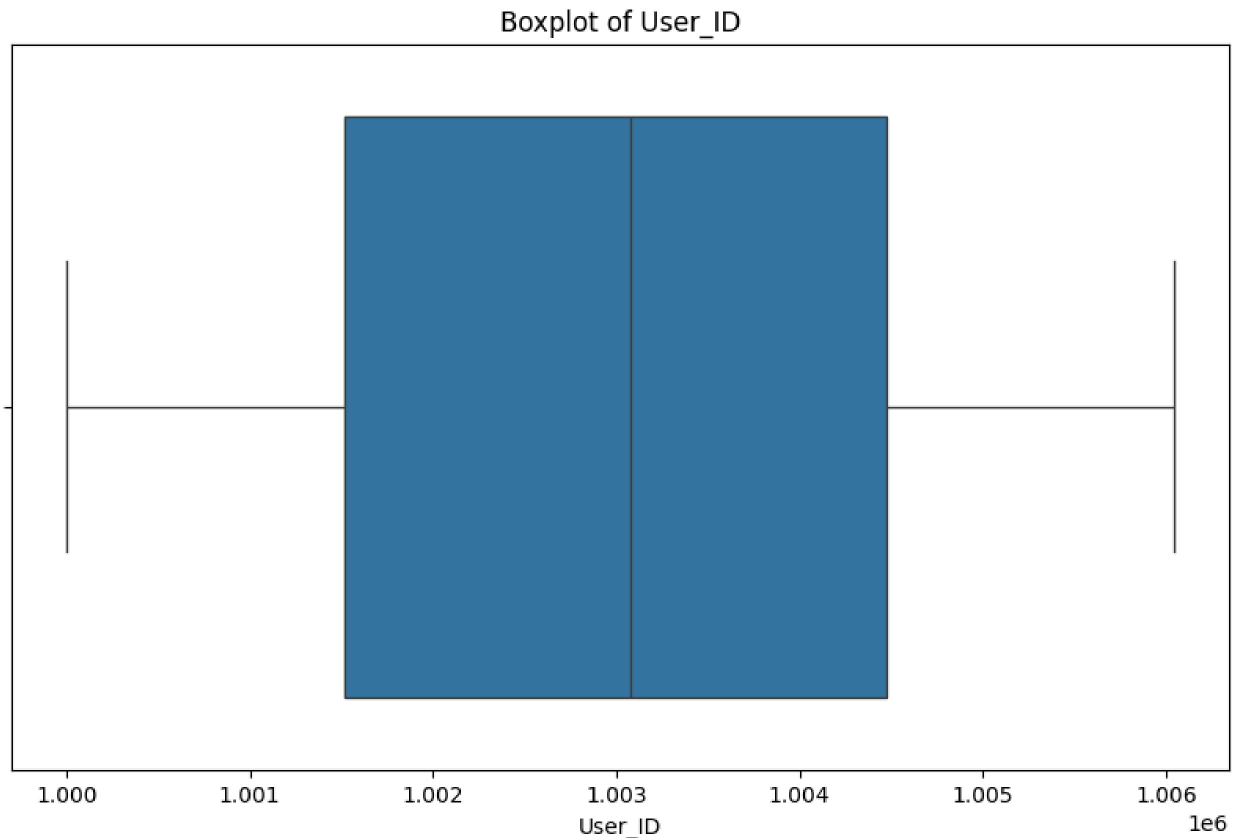
Boxplot of Purchase



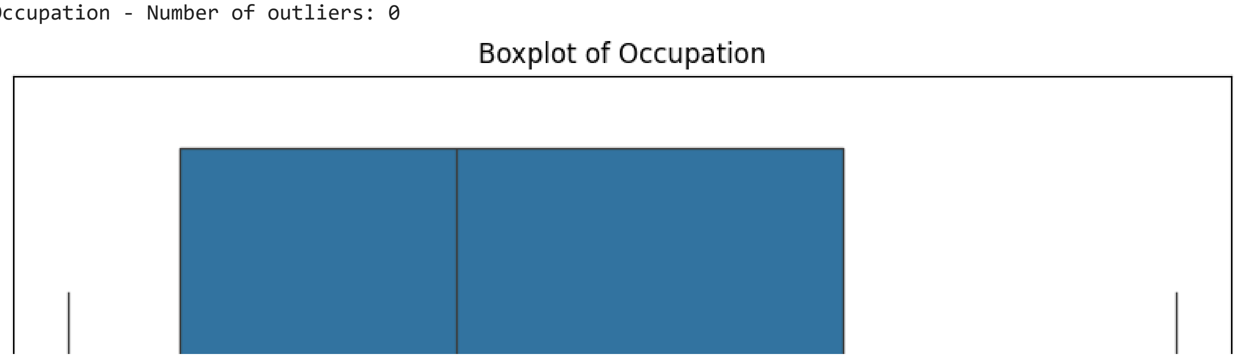

0
5000
10000
15000
20000
25000
Purchase

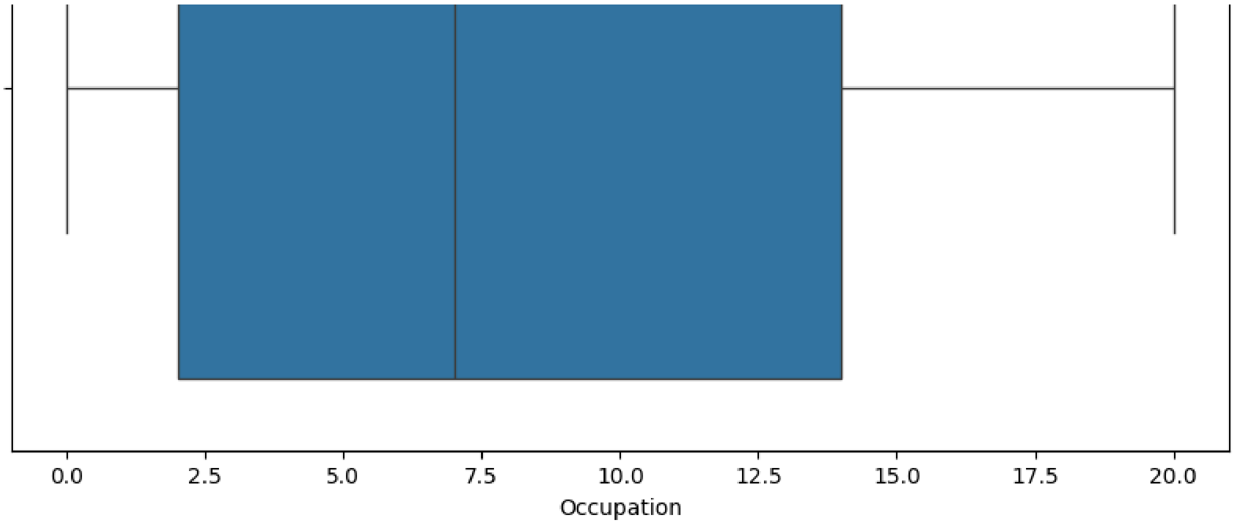
```
# descriptive statistics
desc_stats = df.describe()
print("Descriptive statistics:\n", desc_stats)

# Check for outliers
for column in df.select_dtypes(include=['float64', 'int64']).columns:
    mean = desc_stats.loc['mean', column]
    median = desc_stats.loc['50%', column]
    std_dev = desc_stats.loc['std', column]
    print(f'{column} - Mean: {mean}, Median: {median}, Std Dev: {std_dev}')

    # Check for skewness
    skewness = df[column].skew()
    print(f'{column} skewness: {skewness}')

    # Compute IQR and identify outliers
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    print(f'{column} - IQR: {IQR}')

    # Identify outliers
    outliers = df[(df[column] < (Q1 - 1.5 * IQR)) | (df[column] > (Q3 + 1.5 * IQR))]
    print(f'{column} - Number of outliers: {len(outliers)}')

    # Plot boxplot
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=df[column])
    plt.title(f'Boxplot of {column}')
    plt.show()
```

```
Descriptive statistics:
            User_ID     Occupation  Marital_Status  Product_Category  \
count  5.500680e+05  550068.000000   550068.000000     550068.000000
mean   1.003029e+06       8.076707        0.409653          5.404270
std    1.727592e+03       6.522660        0.491770          3.936211
min    1.000001e+06       0.000000        0.000000          1.000000
25%    1.001516e+06       2.000000        0.000000          1.000000
50%    1.003077e+06       7.000000        0.000000          5.000000
75%    1.004478e+06      14.000000        1.000000          8.000000
max    1.006040e+06      20.000000        1.000000         20.000000

            Purchase
count  550068.000000
mean     9263.968713
std      5023.065394
min        12.000000
25%      5823.000000
50%      8047.000000
75%     12054.000000
max     23961.000000
User_ID - Mean: 1003028.8424013031, Median: 1003077.0, Std Dev: 1727.5915855305516
User_ID skewness: 0.0030655518513462644
User_ID - IQR: 2962.0
User_ID - Number of outliers: 0
```

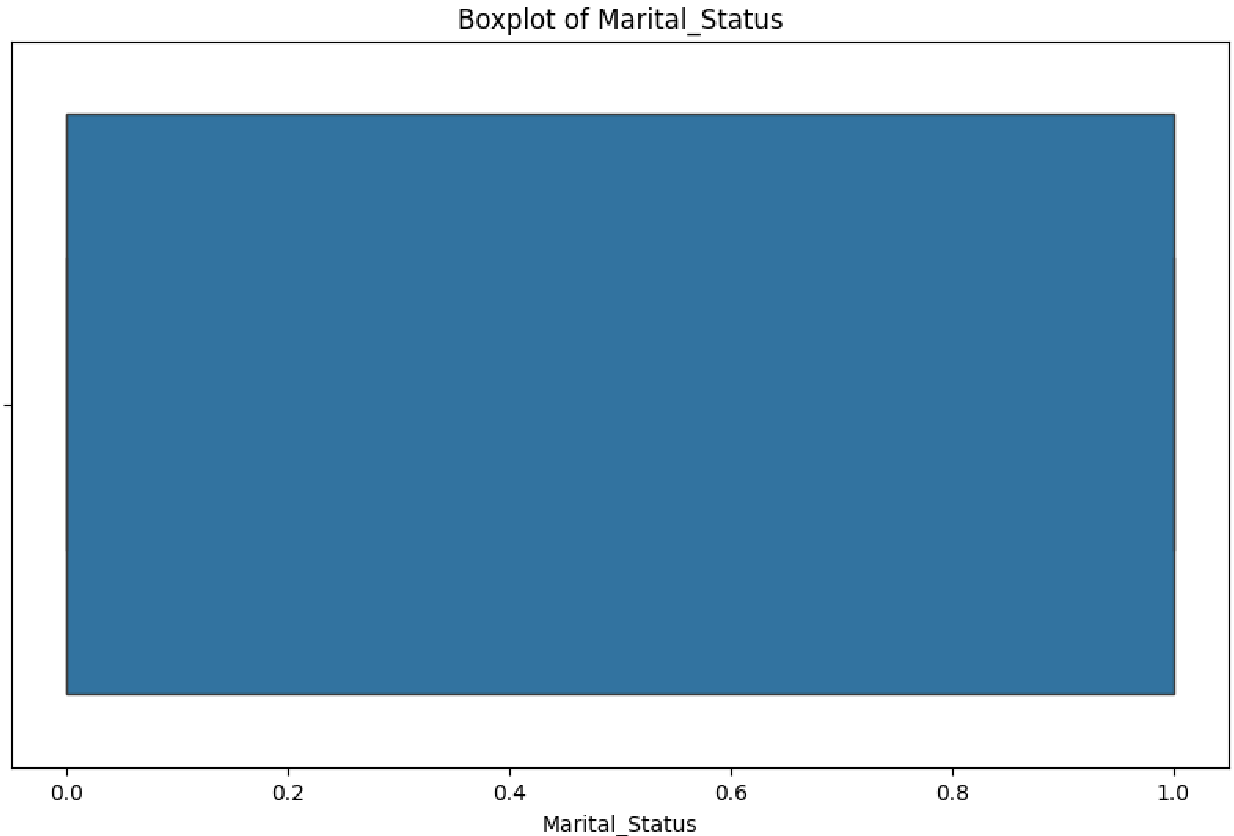



```
Occupation - Mean: 8.076706879876669, Median: 7.0, Std Dev: 6.522660487341824
Occupation skewness: 0.40014010986184784
Occupation - IQR: 12.0
Occupation - Number of outliers: 0
```

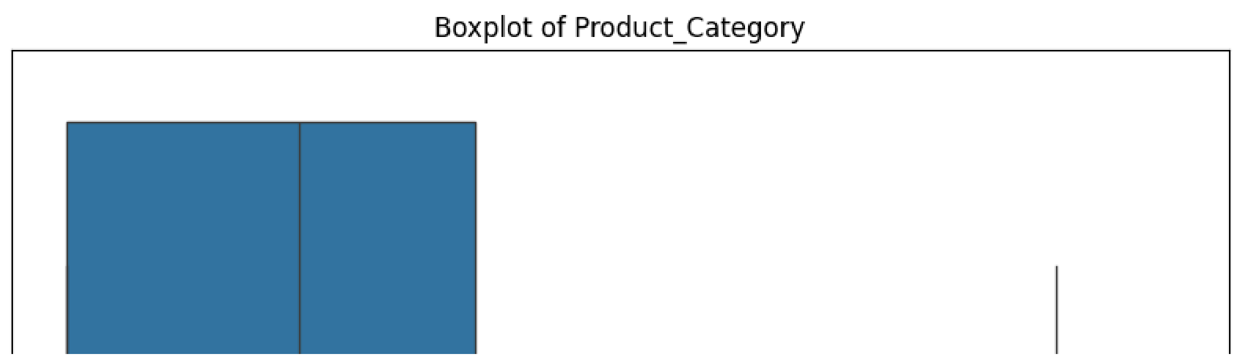


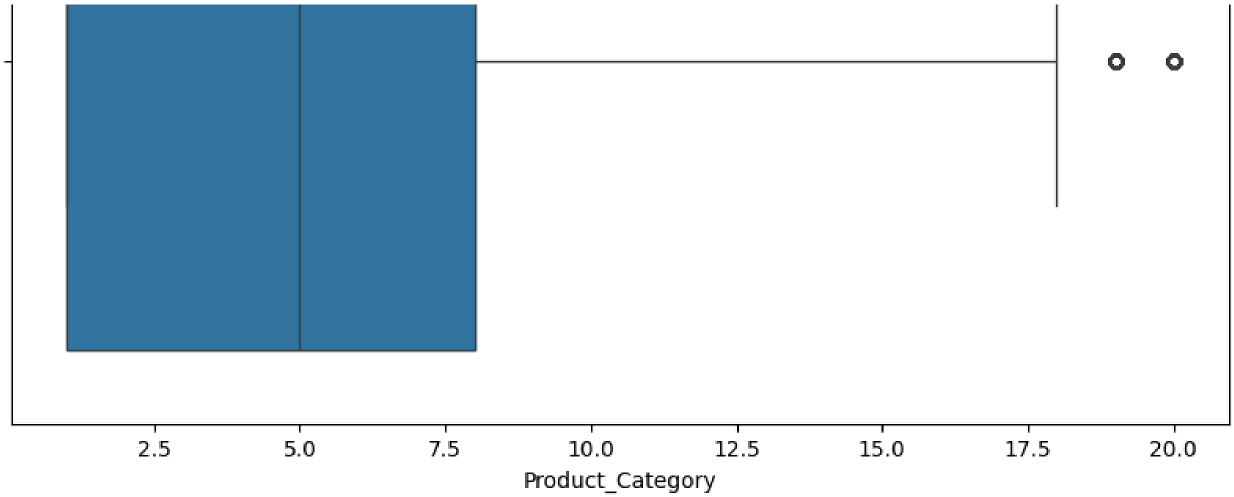



```
Marital_Status - Mean: 0.40965298835780306, Median: 0.0, Std Dev: 0.49177012631733
Marital_Status skewness: 0.3674372854404167
Marital_Status - IQR: 1.0
Marital_Status - Number of outliers: 0
```

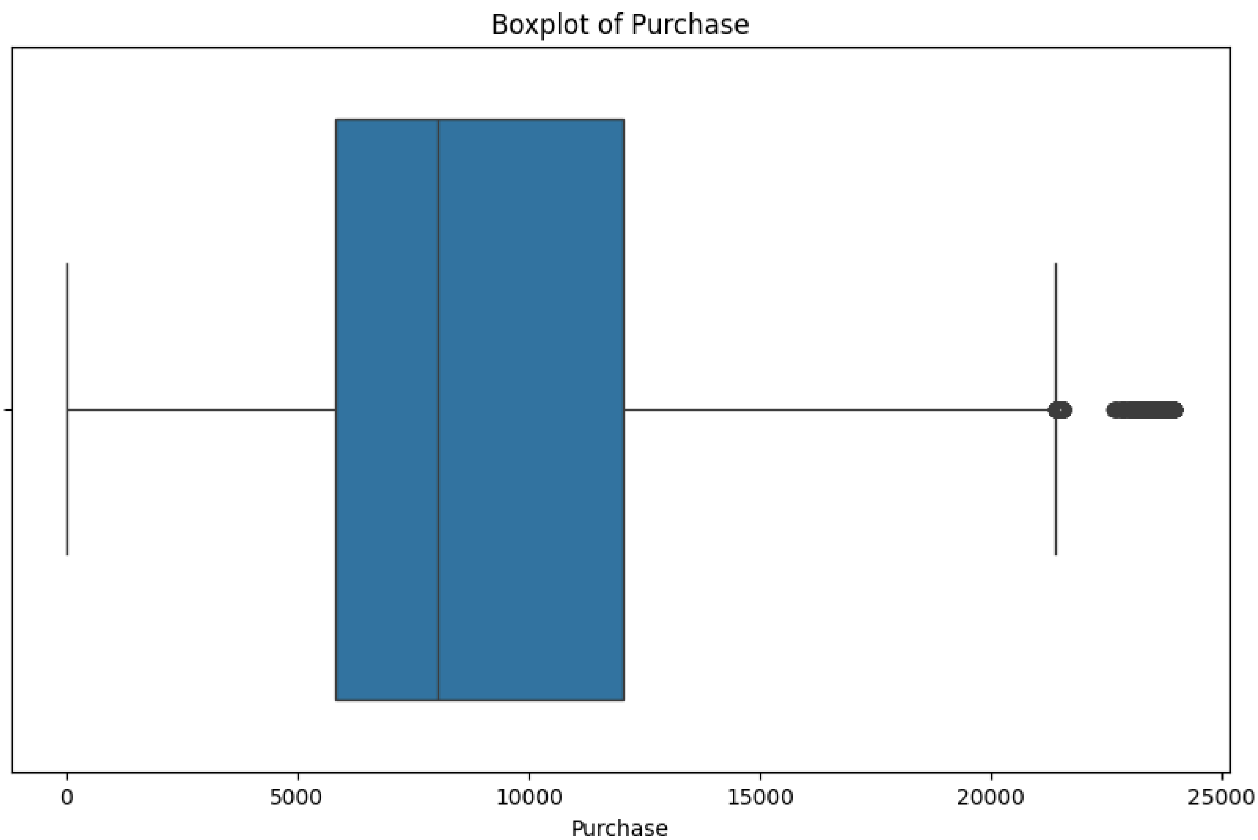



```
Product_Category - Mean: 5.404270017525106, Median: 5.0, Std Dev: 3.936211369201389
Product_Category skewness: 1.0257349338538029
Product_Category - IQR: 7.0
Product_Category - Number of outliers: 4153
```






```
Purchase - Mean: 9263.968712959126, Median: 8047.0, Std Dev: 5023.065393820582
Purchase skewness: 0.6001400037087128
Purchase - IQR: 6231.0
Purchase - Number of outliers: 2677
```



**INSIGHTS : **

There are only outliers in Purchase and Product Category.

Mean Purchase Value (₹9263.97): The average purchase amount is relatively high, indicating that customers are spending a significant amount on each transaction.

Median Purchase Value (₹8047.00): The median is lower than the mean, which suggests that the data might be right-skewed, meaning there are a few larger purchases that are pulling the average upwards.

Standard Deviation (₹5023.07): The wide standard deviation indicates a large variation in purchase amounts, suggesting that customers are spending very differently.

Skewness (0.60): A positive skewness indicates that there are more frequent smaller purchases, with some large purchases pulling the average up.

IQR (₹6231): The interquartile range shows the middle 50% of purchase amounts is between ₹6231, implying that most purchases fall within a moderately wide range.

Number of Outliers (2677): There are a significant number of outliers, suggesting that many purchases are either significantly lower or higher than the typical range.

FOR PRODUCT CATEGORY :

Mean Category (5.40): On average, customers are purchasing items from a product category rated around 5, which may suggest mid-range products.

Median Category (5.0): The median aligns with the mean, indicating a relatively balanced distribution, though not perfectly symmetric.

Standard Deviation (3.94): The variation across product categories is fairly high, showing that customers are buying across a wide range of categories.

Skewness (1.03): A stronger positive skewness compared to purchases indicates that customers tend to buy more lower-category products, with fewer high-category purchases.

IQR (7.0): The middle 50% of customers purchase items within a broad range of 7 product categories, indicating a variety of preferences.

Number of Outliers (4153): A high number of outliers suggests a few product categories are being purchased much more or much less frequently than the majority.

RECOMMEDATIONS :

Target High-Spending Customers:

The presence of large outliers in purchase amounts suggests that some customers are making significantly larger purchases. Focus on personalized offers or loyalty programs for high-value customers to increase their lifetime value.

Address Purchase Variability:

With a wide variation in purchase amounts, consider segmenting customers based on purchase behavior (e.g., low, medium, high spenders) and tailor marketing strategies for each group. Lower spenders can be encouraged with discounts, while high spenders can be targeted with premium products or exclusive deals.

3.Do some data exploration steps like:

Tracking the amount spent per transaction of all the 50 million female customers, and all the 50 million male customers, calculate the average, and conclude the results.

Inference after computing the average female and male expenses.

Use the sample average to find out an interval within which the population average will lie. Using the sample of female customers you will calculate the interval within which the average spending of 50 million male and female customers may lie.

```
# Filter data by gender :

df_female = df[df['Gender'] == 'F']
df_male = df[df['Gender'] == 'M']
```

```
# Calculate average transaction amount per gender :

avg_transaction_amount_female = df_female['Purchase'].mean()
avg_transaction_amount_male = df_male['Purchase'].mean()

print(f'Average purchase amount of female customers : {avg_transaction_amount_female}')
print(f'Average purchase amount of male customers: {avg_transaction_amount_male}')
```

```
Average purchase amount of female customers : 8734.565765155476
Average purchase amount of male customers: 9437.526040472265
```

INSIGHTS :

Male customers purchase amount is slightly higher than female. It could be either male customer purchase more goods or expensive products than females.

RECOMMENDATIONS :

1. Different marketing strategies for male and female customers.
2. Gender specific promotions
3. Analyse purchase trends by Gender

```
# sample statistics for female :

mean_female = df_female['Purchase'].mean()
std_female = df_female['Purchase'].std()
n_female = len(df_female)

# sample statistics for male :

mean_male = df_male['Purchase'].mean()
std_male = df_male['Purchase'].std()
n_male = len(df_male)

print(f'Female - Mean : {mean_female}, Standard_deviation : {std_female}, Sample_size : {n_female}')
print(f'male - Mean : {mean_male}, Standard_deviation : {std_male}, Sample_size : {n_male}')
```

```
Female - Mean : 8734.565765155476, Standard_deviation : 4767.233289291458, Sample_size : 135809
male - Mean : 9437.526040472265, Standard_deviation : 5092.18620977797, Sample_size : 414259
```

4. Use the Central limit theorem to compute the interval. Change the sample size to observe the distribution of the mean of the expenses by female and male customers.

The interval that you calculated is called Confidence Interval. The width of the interval is mostly decided by the business: Typically 90%, 95%, or 99%. Play around with the width parameter and report the observations.

```
import numpy as np
import scipy.stats as stats

def compute_confidence_interval(mean, std_dev, n, confidence_level=0.95):
    """
    Computes the confidence interval for the mean using the Central Limit Theorem.

    Parameters:
        mean (float): Sample mean.
        std_dev (float): Sample standard deviation.
        n (int): Sample size.
        confidence_level (float): Confidence level (e.g., 0.90, 0.95, 0.99).

    Returns:
        tuple: Lower and upper bounds of the confidence interval.
    """
    # Z-score for the confidence level
    z = stats.norm.ppf((1 + confidence_level) / 2)

    # Margin of error
    margin_of_error = z * (std_dev / np.sqrt(n))

    # Confidence interval
    lower_bound = mean - margin_of_error
    upper_bound = mean + margin_of_error

    return lower_bound, upper_bound

# Provided statistics
mean_female = 8734.565765155476
std_female = 4767.233289291458
n_female = 135809

mean_male = 9437.526040472265
std_male = 5092.18620977797
n_male = 414259

# Different sample sizes to test
sample_sizes = [1000, 5000, 10000, 20000, 50000]
confidence_levels = [0.90, 0.95, 0.99]

print("Confidence Intervals for Female and Male Customers")
print("--------------------------------------------------")

for size in sample_sizes:
    print(f"Sample Size: {size}")

    for level in confidence_levels:
        # Compute confidence intervals for the given sample size and confidence level
        female_ci = compute_confidence_interval(mean_female, std_female, size, confidence_level=level)
        male_ci = compute_confidence_interval(mean_male, std_male, size, confidence_level=level)

        print(f"  Confidence Level: {int(level * 100)}%")
        print(f"    Female Expenses CI: {female_ci[0]:,.2f} to {female_ci[1]:,.2f}")
        print(f"    Male Expenses CI: {male_ci[0]:,.2f} to {male_ci[1]:,.2f}")
    print()
```

```
Confidence Intervals for Female and Male Customers
--------------------------------------------------
Sample Size: 1000
  Confidence Level: 90%
    Female Expenses CI: 8,486.60 to 8,982.53
    Male Expenses CI: 9,172.66 to 9,702.40
  Confidence Level: 95%
    Female Expenses CI: 8,439.10 to 9,030.04
    Male Expenses CI: 9,121.91 to 9,753.14
```

```
    Confidence Level: 99%
      Female Expenses CI: 8,346.25 to 9,122.88
      Male Expenses CI: 9,022.74 to 9,852.31

  Sample Size: 5000
    Confidence Level: 90%
      Female Expenses CI: 8,623.67 to 8,845.46
      Male Expenses CI: 9,319.07 to 9,555.98
    Confidence Level: 95%
      Female Expenses CI: 8,602.43 to 8,866.70
      Male Expenses CI: 9,296.38 to 9,578.67
    Confidence Level: 99%
      Female Expenses CI: 8,560.91 to 8,908.23
      Male Expenses CI: 9,252.03 to 9,623.02

  Sample Size: 10000
    Confidence Level: 90%
      Female Expenses CI: 8,656.15 to 8,812.98
      Male Expenses CI: 9,353.77 to 9,521.29
    Confidence Level: 95%
      Female Expenses CI: 8,641.13 to 8,828.00
      Male Expenses CI: 9,337.72 to 9,537.33
    Confidence Level: 99%
      Female Expenses CI: 8,611.77 to 8,857.36
      Male Expenses CI: 9,306.36 to 9,568.69

  Sample Size: 20000
    Confidence Level: 90%
      Female Expenses CI: 8,679.12 to 8,790.01
      Male Expenses CI: 9,378.30 to 9,496.75
    Confidence Level: 95%
      Female Expenses CI: 8,668.50 to 8,800.64
      Male Expenses CI: 9,366.95 to 9,508.10
    Confidence Level: 99%
      Female Expenses CI: 8,647.74 to 8,821.40
      Male Expenses CI: 9,344.78 to 9,530.27

  Sample Size: 50000
    Confidence Level: 90%
      Female Expenses CI: 8,699.50 to 8,769.63
      Male Expenses CI: 9,400.07 to 9,474.98
    Confidence Level: 95%
      Female Expenses CI: 8,692.78 to 8,776.35
      Male Expenses CI: 9,392.89 to 9,482.16
    Confidence Level: 99%
      Female Expenses CI: 8,679.65 to 8,789.48
      Male Expenses CI: 9,378.87 to 9,496.19
```

5. Conclude the results and check if the confidence intervals of average male and female spends are overlapping or not overlapping. How can Walmart leverage this conclusion to make changes or improvements?

```
import numpy as np
import scipy.stats as stats

def compute_confidence_interval(mean, std_dev, n, confidence_level=0.95):
    """
    Computes the confidence interval for the mean using the Central Limit Theorem.

    Parameters:
        mean (float): Sample mean.
        std_dev (float): Sample standard deviation.
        n (int): Sample size.
        confidence_level (float): Confidence level (e.g., 0.90, 0.95, 0.99).

    Returns:
        tuple: Lower and upper bounds of the confidence interval.
    """
```

```python
    # Z-score for the confidence level
    z = stats.norm.ppf((1 + confidence_level) / 2)

    # Margin of error
    margin_of_error = z * (std_dev / np.sqrt(n))

    # Confidence interval
    lower_bound = mean - margin_of_error
    upper_bound = mean + margin_of_error

    return lower_bound, upper_bound

# Provided statistics
mean_female = 8734.565765155476
std_female = 4767.233289291458
n_female = 135809

mean_male = 9437.526040472265
std_male = 5092.18620977797
n_male = 414259

# Different sample sizes to test
sample_sizes = [1000, 5000, 10000, 20000, 50000]
confidence_levels = [0.90, 0.95, 0.99]

print("Confidence Intervals for Female and Male Customers")
print("--------------------------------------------------")

# Iterate over sample sizes and confidence levels
for size in sample_sizes:
    print(f"\nSample Size: {size}")

    for level in confidence_levels:
        # Compute confidence intervals for the given sample size and confidence level
        female_ci = compute_confidence_interval(mean_female, std_female, size, confidence_level=level)
        male_ci = compute_confidence_interval(mean_male, std_male, size, confidence_level=level)

        # Print confidence intervals
        print(f"Confidence Level: {level}")
        print(f"Female CI: {female_ci}")
        print(f"Male CI: {male_ci}")

        # Check for overlap
        def intervals_overlap(ci1, ci2):
            return not (ci1[1] < ci2[0] or ci2[1] < ci1[0])

        overlap = intervals_overlap(female_ci, male_ci)
        print(f"Do the confidence intervals overlap? {'Yes' if overlap else 'No'}")
```

```
Male CI: (9022.74265116118, 9852.30942978335)
Do the confidence intervals overlap? Yes
```

```
Sample Size: 10000
Confidence Level: 0.9
Female CI: (8656.151755491328, 8812.979774819623)
Male CI: (9353.767030909608, 9521.285050034921)
Do the confidence intervals overlap? No
Confidence Level: 0.95
Female CI: (8641.129709626359, 8828.001820684593)
Male CI: (9337.721024734901, 9537.331056209629)
Do the confidence intervals overlap? No
Confidence Level: 0.99
Female CI: (8611.769973121369, 8857.361557189583)
Male CI: (9306.360015889528, 9568.692065055002)
Do the confidence intervals overlap? No

Sample Size: 20000
Confidence Level: 0.9
Female CI: (8679.118687181928, 8790.012843129023)
Male CI: (9378.299476825043, 9496.752604119487)
Do the confidence intervals overlap? No
Confidence Level: 0.95
Female CI: (8668.496496683514, 8800.635033627437)
Male CI: (9366.953237047945, 9508.098843896585)
Do the confidence intervals overlap? No
Confidence Level: 0.99
Female CI: (8647.736027906985, 8821.395502403966)
Male CI: (9344.77765502853, 9530.274425915999)
Do the confidence intervals overlap? No

Sample Size: 50000
Confidence Level: 0.9
Female CI: (8699.497953956003, 8769.633576354949)
Male CI: (9400.067872650234, 9474.984208294296)
Do the confidence intervals overlap? No
Confidence Level: 0.95
Female CI: (8692.779890812966, 8776.351639497985)
Male CI: (9392.891880535428, 9482.160200409102)
Do the confidence intervals overlap? No
Confidence Level: 0.99
Female CI: (8679.649817487638, 8789.481712823314)
Male CI: (9378.866811011183, 9496.185269933347)
Do the confidence intervals overlap? No
```

1. At smaller sample sizes (1000), confidence intervals for male and female spending overlap at a 99% confidence level but do not overlap at 90% and 95% confidence levels.

2. As the sample size increases, the confidence intervals become narrower. At larger sample sizes (50000), the confidence intervals for male and female spending do not overlap at any confidence level (90%, 95%, 99%).

Smaller Sample Sizes: At lower sample sizes, there is some overlap, indicating that there might be some statistical uncertainty in distinguishing between male and female spending habits.

Larger Sample Sizes: At higher sample sizes, the confidence intervals do not overlap, which suggests that the average spending between males and females is more distinct.

What can walmart do?

Efficient resource Allocation: Utilize insights from large samples to allocate marketing budgets and resources more efficiently. For instance, if male spending is higher, Walmart might choose to allocate more resources to campaigns targeting male consumers in certain product categories.

Targetted marketting strategies :

For Smaller Samples: When operating with smaller datasets or in cases where precise segment data is unavailable, Walmart should consider broader marketing strategies that do not rely heavily on gender-specific spending patterns.

For Larger Samples: As data grows and becomes more precise, Walmart can leverage the distinction between male and female spending habits to tailor marketing efforts more effectively. For example, targeted promotions or advertisements based on gender-specific preferences might be more effective.

Product Placement and Inventory:

General Placement: With overlapping intervals at smaller sample sizes, Walmart should focus on a more generalized product placement strategy that appeals to both genders.

Specific Targeting: With larger datasets showing distinct differences, Walmart can optimize product placement and inventory based on gender-specific trends. For example, products that are more popular among one gender could be placed in more prominent locations or given special promotions.

6. Perform the same activity for Married vs Unmarried and Age For Age, you can try bins based on life stages: 0-17, 18-25, 26-35, 36-50, 51+ years.

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

file_path = '/content/walmart_data.csv'

df = pd.read_csv(file_path)

df.head()
```

Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product
P00069042	F	0-17	10	A	2	0	
P00248942	F	0-17	10	A	2	0	
P00087842	F	0-17	10	A	2	0	
P00085442	F	0-17	10	A	2	0	
P00285442	M	55+	16	C	4+	0	

```
print("DataFrame Overview:")
print(df.head())
print("\nUnique values in Marital_Status:")
print(df['Marital_Status'].unique())

print("\nCount of entries for each Marital_Status:")
print(df['Marital_Status'].value_counts())

# Check for NaNs in the Purchase column
print("\nMissing values in Purchase column:")
print(df['Purchase'].isnull().sum())
```

```
DataFrame Overview:
   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

Unique values in Marital_Status:
[0 1]

Count of entries for each Marital_Status:
Marital_Status
0    324731
1    225337
Name: count, dtype: int64

Missing values in Purchase column:
0
```

```
print("\nData types:")
print(df.dtypes)

# Check the unique values and data types in the Marital_Status column
print("\nMarital_Status values and data types:")
print(df['Marital_Status'].unique())
print(df['Marital_Status'].dtype)

# Convert Marital_Status to string if it is not already
df['Marital_Status'] = df['Marital_Status'].astype(str)
```

```
Data types:
User_ID                        int64
Product_ID                    object
Gender                        object
Age                           object
Occupation                     int64
City_Category                 object
Stay_In_Current_City_Years    object
Marital_Status                 int64
Product_Category               int64
Purchase                       int64
dtype: object

Marital_Status values and data types:
[0 1]
int64
```

```
# Filter data by marital status :

df_married = df[df['Marital_Status'] == '1']
df_unmarried = df[df['Marital_Status'] == '0']

# Sample statistics for married :

mean_married = df_married['Purchase'].mean()
std_married = df_married['Purchase'].std()
n_married = len(df_married)

# Sample statistics for unmarried :

mean_unmarried = df_unmarried['Purchase'].mean()
std_unmarried = df_unmarried['Purchase'].std()
n_unmarried = len(df_unmarried)
```