BUSINESS CASE: WALMART- CONFIDENCE INTERVAL AND CLT

```
! pip install pandas

Requirement already satisfied: pandas in e:\rasa\lib\site-packages
(2.1.4)
Requirement already satisfied: numpy<2,>=1.23.2 in e:\rasa\lib\site-
packages (from pandas) (1.24.3)
Requirement already satisfied: python-dateutil>=2.8.2 in e:\rasa\lib\
site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in e:\rasa\lib\site-
packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in e:\rasa\lib\site-
packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in e:\rasa\lib\site-packages
(from python-dateutil>=2.8.2->pandas) (1.16.0)

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

import scipy.stats as stats

#IMPORTING THE DATASET
data = pd.read_csv('G:/dsml-scaler/probability and
stats/casestudy/walmart_data.txt')
```

Observing the dataset

```
data.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
```

```python
print("The shape of dataset is:",data.shape)
```

```
The shape of dataset is: (550068, 10)
```

```python
print("Datatypes of all attributes:")
print(data.dtypes)
```

```
Datatypes of all attributes:
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
```

```python
data.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
```

```python
# converting object data type to categorical data type
# Convert object columns to categorical data type
data['Product_ID'] = data['Product_ID'].astype('category')
data['Gender'] = data['Gender'].astype('category')
data['Age'] = data['Age'].astype('category')
data['City_Category'] = data['City_Category'].astype('category')
data['Stay_In_Current_City_Years'] =
data['Stay_In_Current_City_Years'].astype('category')
```

```
# Check the data types after conversion
print(data.dtypes)
```

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

```
#checking null and duplicates

data.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
```

```
# Checking for duplicate rows
duplicate_rows = data[data.duplicated()]
print(duplicate_rows.shape[0])
```

```
0
```

```
# Print unique values of the columns before label encoding
print("Unique values of Gender:", data['Gender'].unique())
print("Unique values of Age:", data['Age'].unique())
print("Unique values of City_Category:",
data['City_Category'].unique())
print("Unique values of Stay_In_Current_City_Years:",
data['Stay_In_Current_City_Years'].unique())
```

```
Unique values of Gender: ['F', 'M']
Categories (2, object): ['F', 'M']
Unique values of Age: ['0-17', '55+', '26-35', '46-50', '51-55', '36-
45', '18-25']
Categories (7, object): ['0-17', '18-25', '26-35', '36-45', '46-50',
```

```
'51-55', '55+']
Unique values of City_Category: ['A', 'C', 'B']
Categories (3, object): ['A', 'B', 'C']
Unique values of Stay_In_Current_City_Years: ['2', '4+', '3', '1',
'0']
Categories (5, object): ['0', '1', '2', '3', '4+']
```

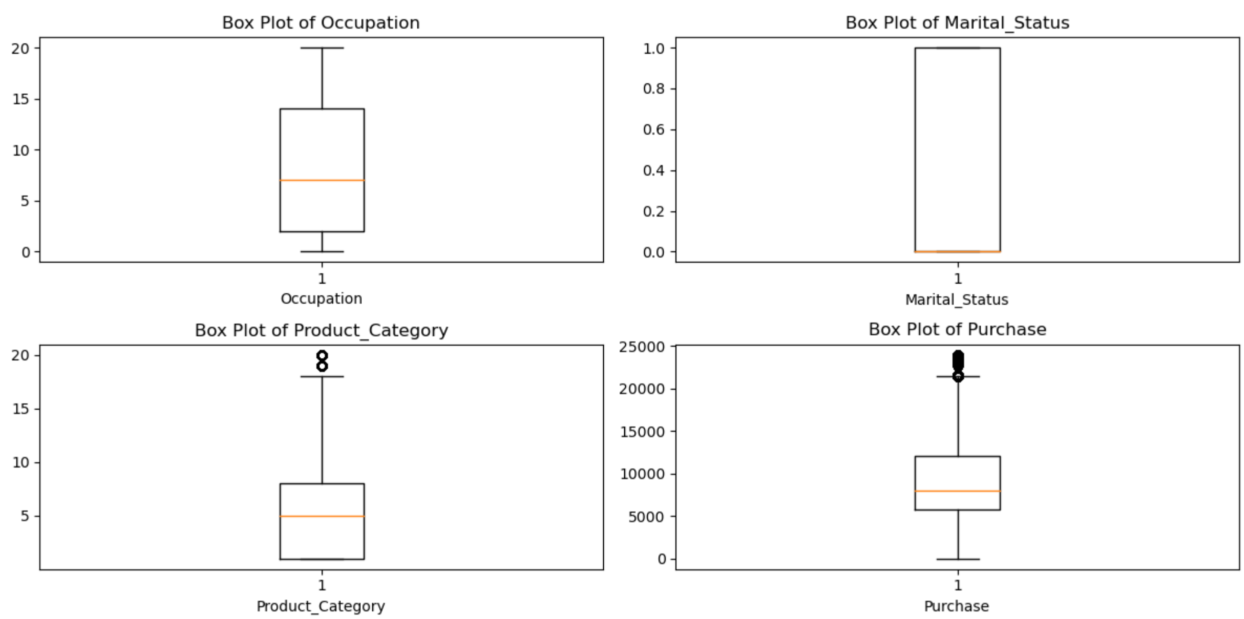
Label Encoding

```
from sklearn.preprocessing import LabelEncoder

# Assuming df is your DataFrame containing the dataset
# Replace df with your actual DataFrame name

# Define columns to be label encoded
columns_to_encode = ['Gender', 'Age', 'City_Category',
'Stay_In_Current_City_Years']

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Iterate through each column and perform label encoding
for column in columns_to_encode:
    data[column] = label_encoder.fit_transform(data[column])

# Displaying the encoded DataFrame
print(data.head())

   User_ID Product_ID  Gender  Age  Occupation  City_Category  \
0  1000001  P00069042       0    0          10              0
1  1000001  P00248942       0    0          10              0
2  1000001  P00087842       0    0          10              0
3  1000001  P00085442       0    0          10              0
4  1000002  P00285442       1    6          16              2

   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

data.dtypes
```

```
User_ID                          int64
Product_ID                    category
Gender                           int32
Age                              int32
Occupation                       int64
City_Category                    int32
Stay_In_Current_City_Years       int32
Marital_Status                   int64
Product_Category                 int64
Purchase                         int64
dtype: object
```

Outlier Detection

```python
import matplotlib.pyplot as plt
# Select numerical features for which you want to identify outliers
numerical_features = ['Occupation', 'Marital_Status',
'Product_Category', 'Purchase']

# Create box plots for each numerical feature
plt.figure(figsize=(12, 6))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(2, 2, i)
    plt.boxplot(data[feature])
    plt.title(f'Box Plot of {feature}')
    plt.xlabel(feature)

plt.tight_layout()
plt.show()
```



```
q1= data.Purchase.quantile(0.25)
q3= data.Purchase.quantile(0.75)
iqr=q3-q1
outliers = data[((data.Purchase < (q1-1.5*iqr)) | (data.Purchase >
(q3+1.5*iqr)))]
print("number of outliers:",len(outliers))
print("percentage of outliers:", len(outliers)/data.shape[0])
```

```
number of outliers: 2677
percentage of outliers: 0.004866671029763593
```

Non Graphical Analysis

```
# Analyzing value counts and uniqueness of the data
for column in data.columns:
    value_counts =
data[column].value_counts().sort_values(ascending=False)
    uniqueness = len(value_counts)

    print(f"Column: {column}")
    print(f"Number of unique values: {uniqueness}")
    print("Value counts (sorted in descending order):")
    print(value_counts)
    print()
```

```
Column: User_ID
Number of unique values: 5891
Value counts (sorted in descending order):
User_ID
1001680    1026
1004277     979
1001941     898
1001181     862
1000889     823
           ...
1002690       7
1002111       7
1005810       7
1004991       7
1000708       6
Name: count, Length: 5891, dtype: int64

Column: Product_ID
Number of unique values: 3631
Value counts (sorted in descending order):
Product_ID
P00265242    1880
P00025442    1615
P00110742    1612
P00112142    1562
```

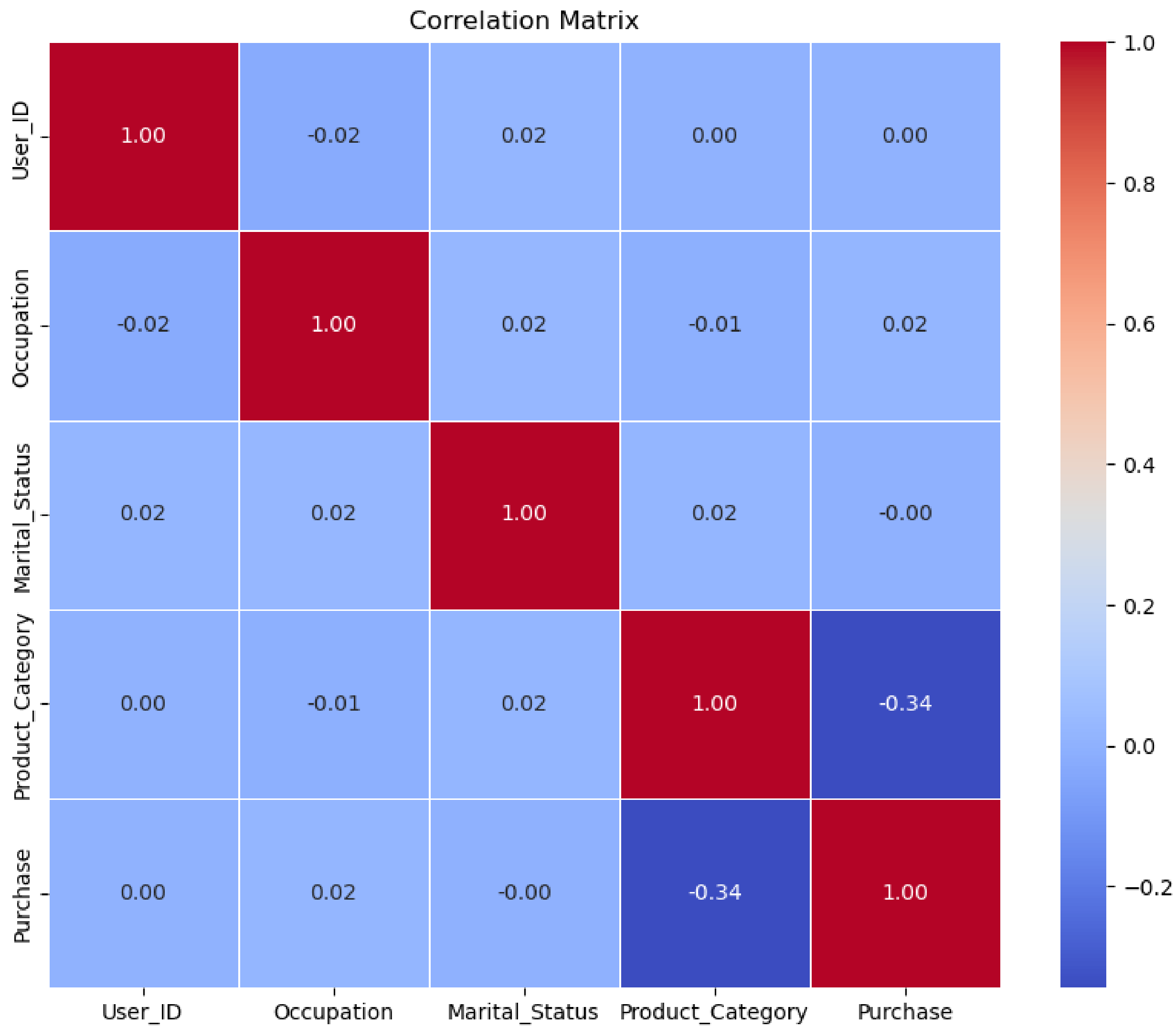
```
P00057642    1470
             ...
P00065542       1
P00107442       1
P00135942       1
P00065942       1
P00231642       1
Name: count, Length: 3631, dtype: int64

Column: Gender
Number of unique values: 2
Value counts (sorted in descending order):
Gender
1    414259
0    135809
Name: count, dtype: int64

Column: Age
Number of unique values: 7
Value counts (sorted in descending order):
Age
2    219587
3    110013
1     99660
4     45701
5     38501
6     21504
0     15102
Name: count, dtype: int64

Column: Occupation
Number of unique values: 21
Value counts (sorted in descending order):
Occupation
4     72308
0     69638
7     59133
1     47426
17    40043
20    33562
12    31179
14    27309
2     26588
16    25371
6     20355
3     17650
10    12930
5     12177
15    12165
11    11586
```

```
19     8461
13     7728
18     6622
9      6291
8      1546
Name: count, dtype: int64

Column: City_Category
Number of unique values: 3
Value counts (sorted in descending order):
City_Category
1    231173
2    171175
0    147720
Name: count, dtype: int64

Column: Stay_In_Current_City_Years
Number of unique values: 5
Value counts (sorted in descending order):
Stay_In_Current_City_Years
1    193821
2    101838
3     95285
4     84726
0     74398
Name: count, dtype: int64

Column: Marital_Status
Number of unique values: 2
Value counts (sorted in descending order):
Marital_Status
0    324731
1    225337
Name: count, dtype: int64

Column: Product_Category
Number of unique values: 20
Value counts (sorted in descending order):
Product_Category
5     150933
1     140378
8     113925
11     24287
2      23864
6      20466
3      20213
4      11753
16      9828
15      6290
13      5549
```

```
10       5125
12       3947
7        3721
18       3125
20       2550
19       1603
14       1523
17        578
9         410
Name: count, dtype: int64

Column: Purchase
Number of unique values: 18105
Value counts (sorted in descending order):
Purchase
7011     191
7193     188
6855     187
6891     184
7012     183
        ...
14837      1
14890      1
4849       1
4852       1
21489      1
Name: count, Length: 18105, dtype: int64
```

Visual Graphic Analysis

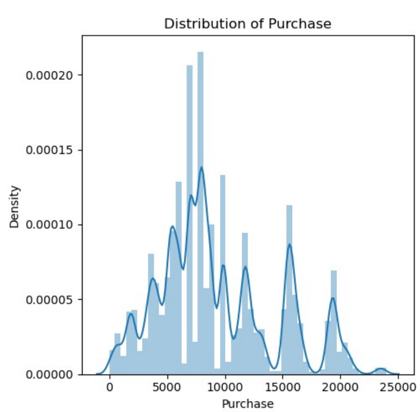
```
# Selecting only numeric columns
numeric_data = data.select_dtypes(include=['int64', 'float64'])

# Calculate correlation matrix
corr_matrix = numeric_data.corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f",
linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

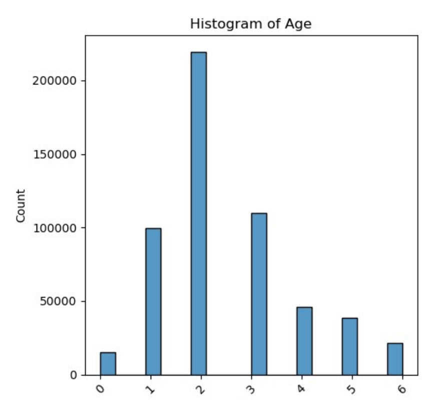



```
# Univariate analysis
# Continuous variables: Distplot, countplot, histogram
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
sns.distplot(data['Purchase'])
plt.title('Distribution of Purchase')

plt.subplot(1, 3, 3)
sns.histplot(data['Age'], bins=20)
plt.title('Histogram of Age')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

plt.subplot(1, 3, 3)
sns.histplot(data['Occupation'], bins=20)
plt.title('Histogram of Occupation')
```

```
plt.xlabel('Occupation')
plt.ylabel('Count')
plt.show()

# Categorical variable: Boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(x='Gender', y='Purchase', data=data)
plt.title('Boxplot of Purchase by Gender')
plt.show()
```

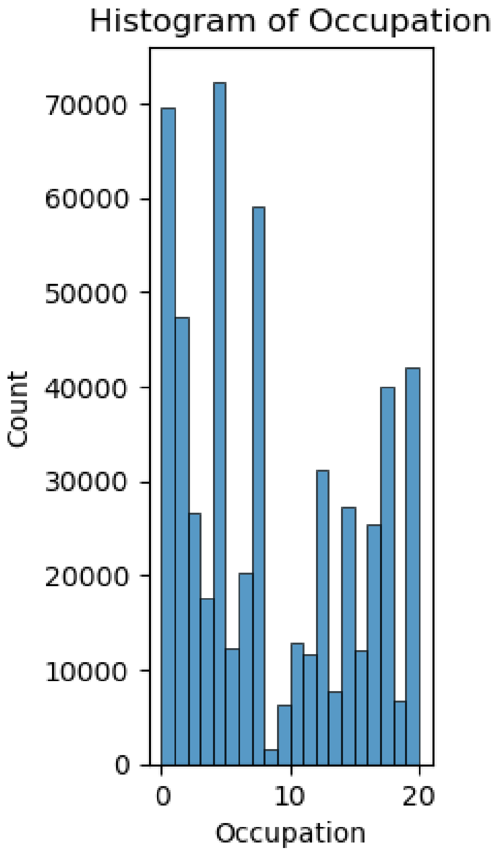
```
C:\Users\user\AppData\Local\Temp\ipykernel_2032\3437351054.py:5:
UserWarning:

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

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

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

  sns.distplot(data['Purchase'])
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future
version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future
version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

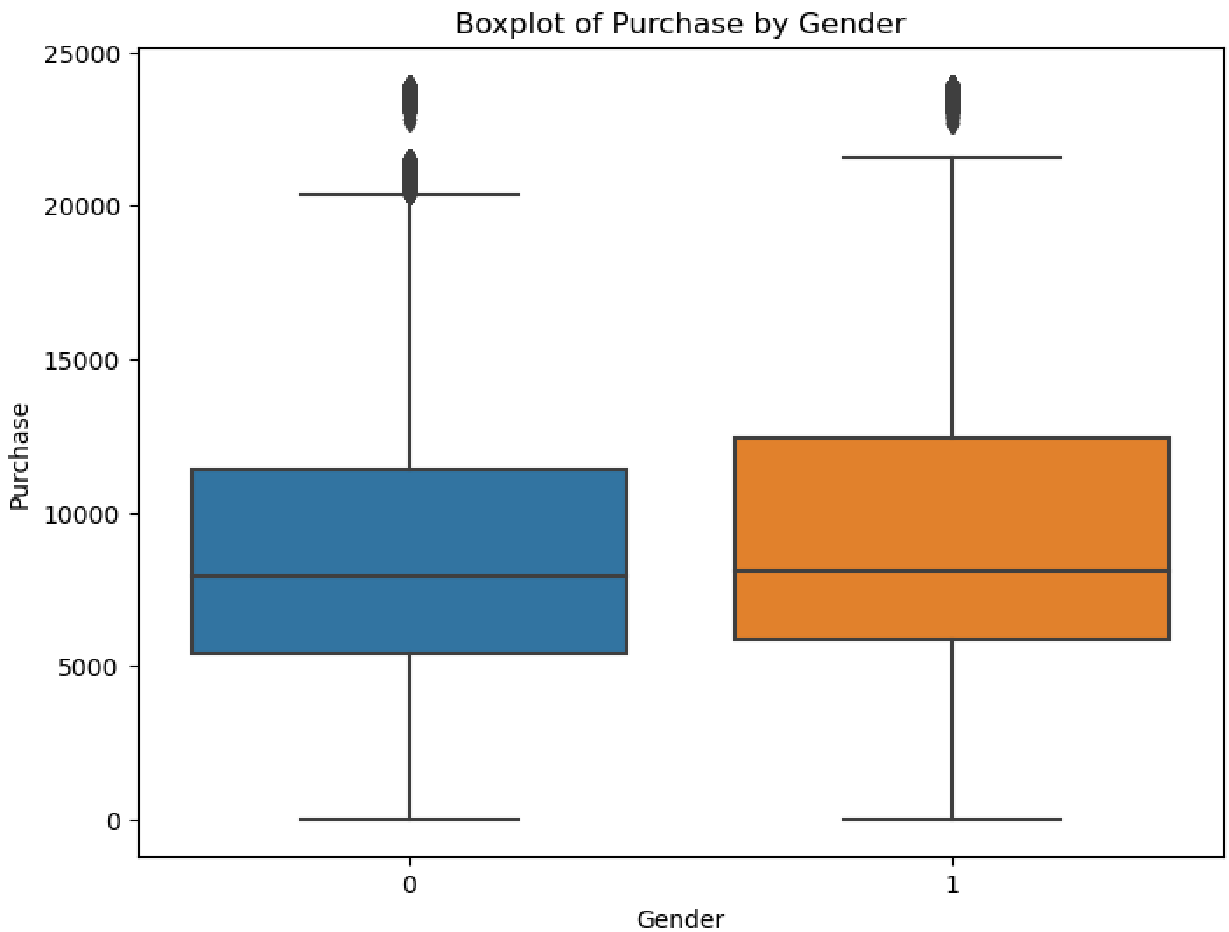





```
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future
```

```
version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```





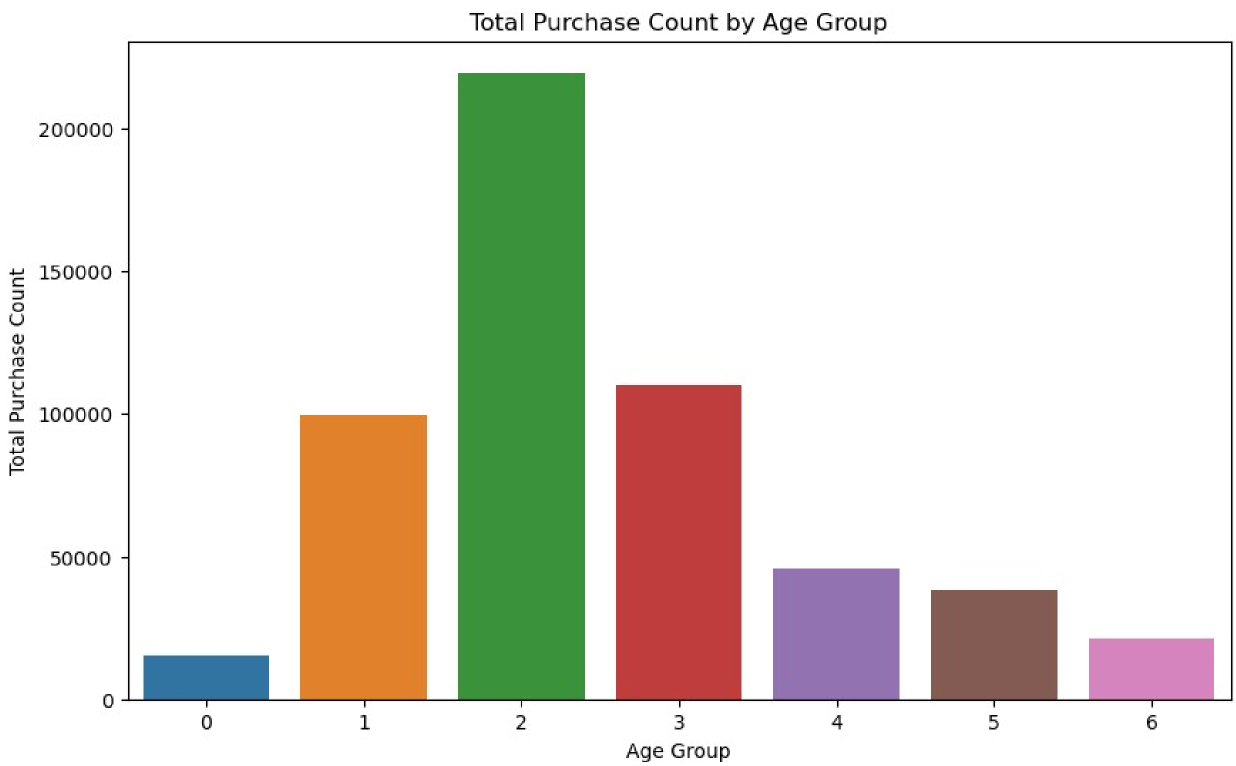

```
# Group the data by 'Age' and calculate the count of purchases for
each age group
purchase_counts_by_age = data.groupby('Age')['Purchase'].count()

# Display the purchase counts by age group
print("Purchase counts by age group:")
print(purchase_counts_by_age)

Purchase counts by age group:
Age
0     15102
1     99660
2    219587
3    110013
4     45701
5     38501
6     21504
Name: Purchase, dtype: int64

# Group the data by 'Age' and calculate the count of purchases for
each age group
```

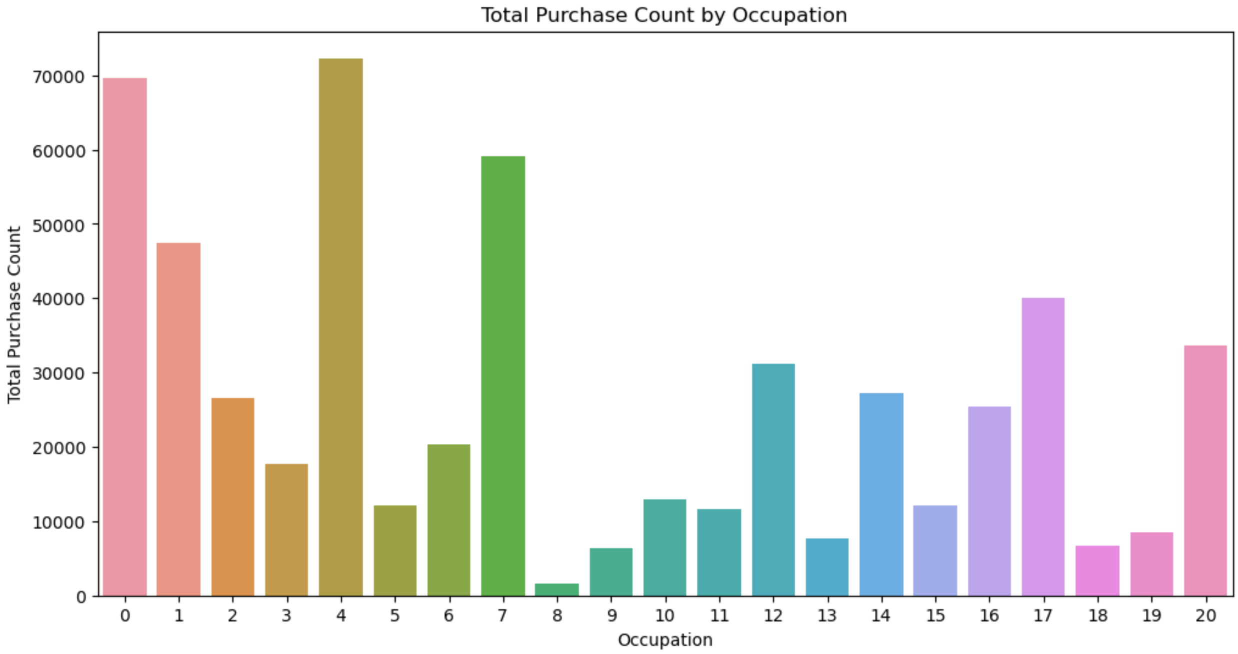
```
purchase_counts_by_occupation = data.groupby('Occupation')
['Purchase'].count()

# Display the purchase counts by age group
print("Purchase counts by age group:")
print(purchase_counts_by_occupation)
```

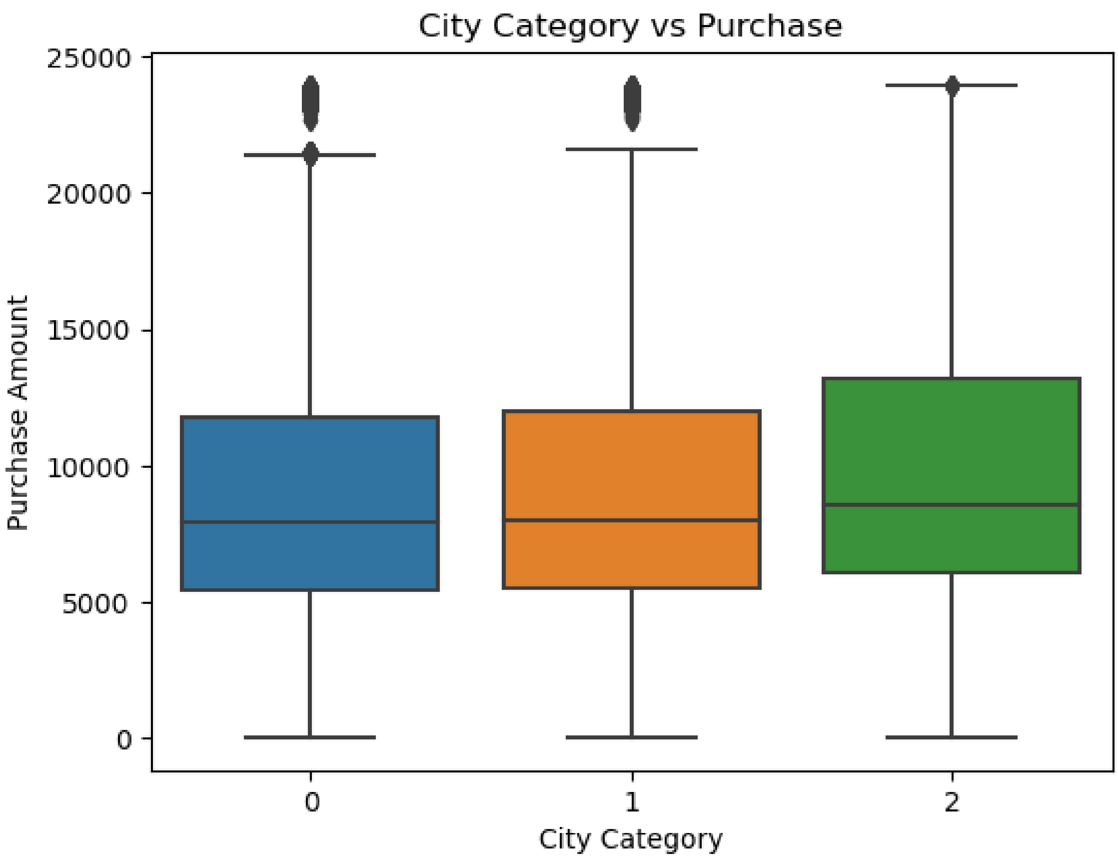
```
Purchase counts by age group:
Occupation
0     69638
1     47426
2     26588
3     17650
4     72308
5     12177
6     20355
7     59133
8      1546
9      6291
10    12930
11    11586
12    31179
13     7728
14    27309
15    12165
16    25371
17    40043
18     6622
19     8461
20    33562
Name: Purchase, dtype: int64
```

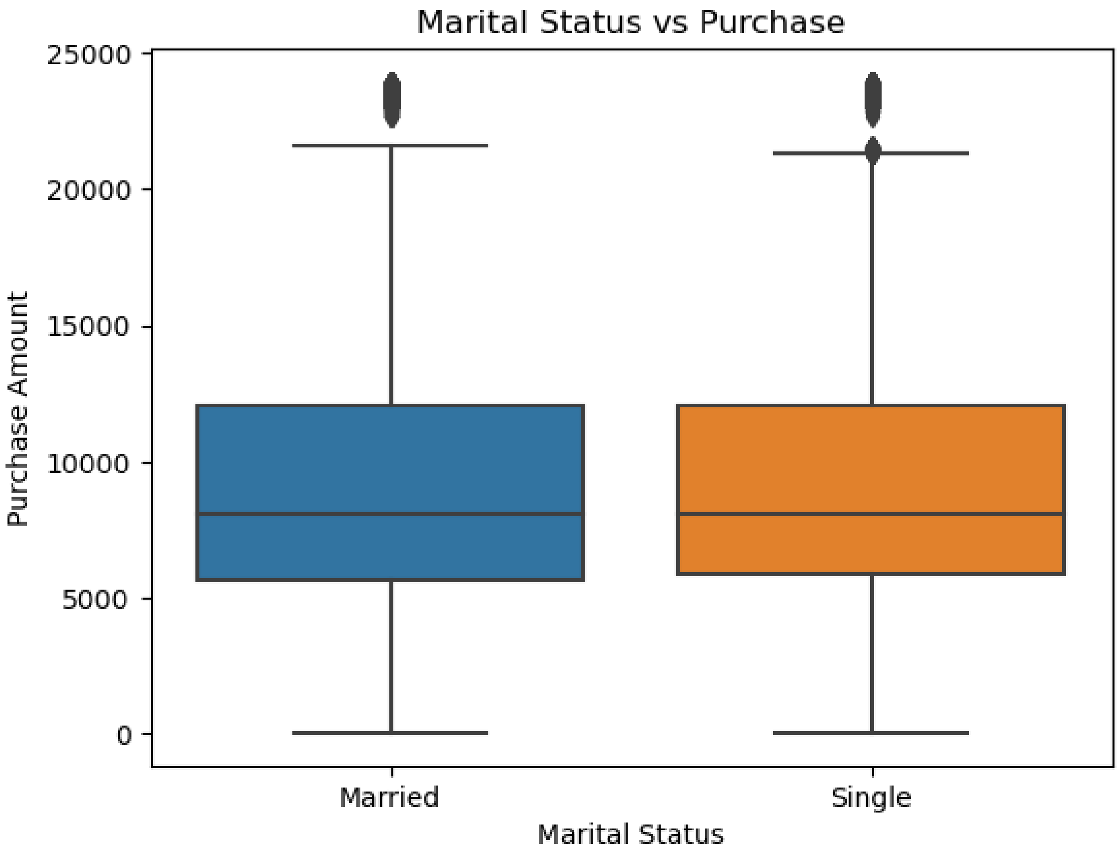
```
# Pairwise bivariate analysis
# Plotting the total purchase amount for each age group
plt.figure(figsize=(10, 6))
sns.barplot(x=purchase_counts_by_age.index,
y=purchase_counts_by_age.values)
plt.title('Total Purchase Count by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Total Purchase Count')
plt.show()

# Plotting the total purchase amount for each occupation
plt.figure(figsize=(12, 6))
sns.barplot(x=purchase_counts_by_occupation.index,
y=purchase_counts_by_occupation.values)
plt.title('Total Purchase Count by Occupation')
plt.xlabel('Occupation')
plt.ylabel('Total Purchase Count')
plt.show()
```

```
sns.boxplot(x='City_Category', y='Purchase', data=data)
plt.title('City Category vs Purchase')
plt.xlabel('City Category')
plt.ylabel('Purchase Amount')
plt.show()

sns.boxplot(x='Marital_Status', y='Purchase', data=data)
plt.title('Marital Status vs Purchase')
plt.xlabel('Marital Status')
plt.ylabel('Purchase Amount')
plt.xticks(ticks=[0, 1], labels=['Married', 'Single'])
plt.show()
```




Total Purchase Count by Occupation
Total Purchase Count
0
10000
20000
30000
40000
50000
60000
70000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Occupation


City Category vs Purchase
Purchase Amount
0
5000
10000
15000
20000
25000
0 1 2
City Category



Applying CLT

```
# Filter the dataset for male and female customers
male_customers = data[data['Gender'] == 1]['Purchase']
female_customers = data[data['Gender'] == 0]['Purchase']

# Calculate average spending for male and female customers
male_avg_spending = male_customers.mean()
female_avg_spending = female_customers.mean()

# Print the average spending for male and female customers
print("Average spending per transaction for male customers:",
male_avg_spending)
print("Average spending per transaction for female customers:",
female_avg_spending)

# Infer the results
if male_avg_spending > female_avg_spending:
    print("On average, male customers spend more per transaction than
female customers.")
elif male_avg_spending < female_avg_spending:
    print("On average, female customers spend more per transaction
than male customers.")
else:
```

```
    print("There is no significant difference in the average spending
per transaction between male and female customers.")

# Calculate sample standard deviation for male and female customers
male_std = male_customers.std()
female_std = female_customers.std()

# Calculate the sample size
sample_size_male = len(male_customers)
sample_size_female = len(female_customers)

# Calculate the standard error of the mean for male and female
customers
male_se = male_std / (sample_size_male ** 0.5)
female_se = female_std / (sample_size_female ** 0.5)

# Set the confidence level
confidence_level = 0.95

# Calculate the margin of error
margin_of_error_male = male_se * 1.96  # 1.96 is the z-value for 95%
confidence interval
margin_of_error_female = female_se * 1.96

# Calculate the confidence interval for male and female customers
ci_male = (male_avg_spending - margin_of_error_male, male_avg_spending
+ margin_of_error_male)
ci_female = (female_avg_spending - margin_of_error_female,
female_avg_spending + margin_of_error_female)

print("95% Confidence Interval for average spending of male
customers:", ci_male)
print("95% Confidence Interval for average spending of female
customers:", ci_female)
```

```
Average spending per transaction for male customers: 9437.526040472265
Average spending per transaction for female customers:
8734.565765155476
On average, male customers spend more per transaction than female
customers.
95% Confidence Interval for average spending of male customers:
(9422.019162420047, 9453.032918524483)
95% Confidence Interval for average spending of female customers:
(8709.211081242413, 8759.920449068539)
```

```
# Changing the confidence interval to see the impact
# Print the average spending for male and female customers
print("Average spending per transaction for male customers:",
male_avg_spending)
print("Average spending per transaction for female customers:",
```

```
female_avg_spending)

# Infer the results
if male_avg_spending > female_avg_spending:
    print("On average, male customers spend more per transaction than
female customers.")
elif male_avg_spending < female_avg_spending:
    print("On average, female customers spend more per transaction
than male customers.")
else:
    print("There is no significant difference in the average spending
per transaction between male and female customers.")


# Set the confidence level
confidence_level = 0.90

# Calculate the margin of error
margin_of_error_male = male_se * 1.64  # 1.64 is the z-value for 95%
confidence interval
margin_of_error_female = female_se * 1.64

# Calculate the confidence interval for male and female customers
ci_male = (male_avg_spending - margin_of_error_male, male_avg_spending
+ margin_of_error_male)
ci_female = (female_avg_spending - margin_of_error_female,
female_avg_spending + margin_of_error_female)

print("90% Confidence Interval for average spending of male
customers:", ci_male)
print("90% Confidence Interval for average spending of female
customers:", ci_female)
```

```
Average spending per transaction for male customers: 9437.526040472265
Average spending per transaction for female customers:
8734.565765155476
On average, male customers spend more per transaction than female
customers.
90% Confidence Interval for average spending of male customers:
(9424.550897612247, 9450.501183332282)
90% Confidence Interval for average spending of female customers:
(8713.350621473117, 8755.780908837834)
```

```
# taking various samples
# Define a function to generate multiple samples and compute sample
means
def compute_sample_means(data, sample_size, num_samples):
    sample_means = []
    for _ in range(num_samples):
        sample = np.random.choice(data, size=sample_size,
```

```
replace=False)
        sample_mean = np.mean(sample)
        sample_means.append(sample_mean)
    return sample_means

# Define parameters
sample_size = 1000  # You can adjust the sample size as needed
num_samples = 1000  # Number of samples to generate

# Generate sample means for male and female customers
male_sample_means = compute_sample_means(male_customers, sample_size,
num_samples)
female_sample_means = compute_sample_means(female_customers,
sample_size, num_samples)

avg_male_sample_means=np.mean(male_sample_means)
avg_female_sample_means = np.mean(female_sample_means)

# Print the average spending for male and female customers
print("Average spending per transaction for male
customers:",avg_male_sample_means )
print("Average spending per transaction for female customers:",
avg_female_sample_means )

# Calculate standard deviation of the sample means
male_sample_means_std = np.std(male_sample_means)
female_sample_means_std = np.std(female_sample_means)

# Calculate standard error of the mean
male_se1 = male_sample_means_std / np.sqrt(sample_size)
female_se1 = female_sample_means_std / np.sqrt(sample_size)

# Set confidence level = 95%
confidence_level = 0.95

# Calculate z-score based on confidence level
z_score = 1.96  # For 95% confidence level

# Calculate margin of error
male_margin_of_error = z_score * male_se1
female_margin_of_error = z_score * female_se1

# Calculate confidence interval
male_ci = (avg_male_sample_means - male_margin_of_error,
avg_male_sample_means + male_margin_of_error)
female_ci = (avg_female_sample_means - female_margin_of_error,
avg_female_sample_means + female_margin_of_error)

print("95% Confidence Interval for average spending of male customers
(using CLT):", male_ci)
```

```python
print("95% Confidence Interval for average spending of female
customers (using CLT):", female_ci)

# Set confidence level = 99%
confidence_level = 0.99

# Calculate z-score based on confidence level
z_score = 2.57  # For 99% confidence level

# Calculate margin of error
male_margin_of_error = z_score * male_se1
female_margin_of_error = z_score * female_se1

# Calculate confidence interval
male_ci = (np.mean(male_sample_means) - male_margin_of_error,
np.mean(male_sample_means) + male_margin_of_error)
female_ci = (np.mean(female_sample_means) - female_margin_of_error,
np.mean(female_sample_means) + female_margin_of_error)

print("99% Confidence Interval for average spending of male customers
(using CLT):", male_ci)
print("99% Confidence Interval for average spending of female
customers (using CLT):", female_ci)

# Set confidence level = 90%
confidence_level = 0.90

# Calculate z-score based on confidence level
z_score = 1.64  # For 90% confidence level

# Calculate margin of error
male_margin_of_error = z_score * male_se1
female_margin_of_error = z_score * female_se1

# Calculate confidence interval
male_ci = (np.mean(male_sample_means) - male_margin_of_error,
np.mean(male_sample_means) + male_margin_of_error)
female_ci = (np.mean(female_sample_means) - female_margin_of_error,
np.mean(female_sample_means) + female_margin_of_error)

print("90% Confidence Interval for average spending of male customers
(using CLT):", male_ci)
print("90% Confidence Interval for average spending of female
customers (using CLT):", female_ci)
```

```
Average spending per transaction for male customers: 9445.657505
Average spending per transaction for female customers: 8735.489376
95% Confidence Interval for average spending of male customers (using
CLT): (9435.830506466697, 9455.484503533302)
95% Confidence Interval for average spending of female customers
```

```
(using CLT): (8726.194516667312, 8744.784235332687)
99% Confidence Interval for average spending of male customers (using
CLT): (9432.772103861946, 9458.542906138053)
99% Confidence Interval for average spending of female customers
(using CLT): (8723.301728813773, 8747.677023186227)
90% Confidence Interval for average spending of male customers (using
CLT): (9437.434914390502, 9453.880095609496)
90% Confidence Interval for average spending of female customers
(using CLT): (8727.71204472163, 8743.26670727837)

### CLT for Married and single
# Filter the dataset for male and female customers
single_customers = data[data['Marital_Status'] == 1]['Purchase']
married_customers = data[data['Marital_Status'] == 0]['Purchase']

# Define a function to generate multiple samples and compute sample
means
def compute_sample_means(data, sample_size, num_samples):
    sample_means = []
    for _ in range(num_samples):
        sample = np.random.choice(data, size=sample_size,
replace=False)
        sample_mean = np.mean(sample)
        sample_means.append(sample_mean)
    return sample_means

# Define parameters
sample_size = 1000  # You can adjust the sample size as needed
num_samples = 1000  # Number of samples to generate

# Generate sample means for male and female customers
single_sample_means = compute_sample_means(single_customers,
sample_size, num_samples)
married_sample_means = compute_sample_means(married_customers,
sample_size, num_samples)

avg_single_sample_means=np.mean(single_sample_means)
avg_married_sample_means = np.mean(married_sample_means)

# Print the average spending for male and female customers
print("Average spending per transaction for male
customers:",avg_single_sample_means )
print("Average spending per transaction for female customers:",
avg_married_sample_means )

# Calculate standard deviation of the sample means
single_sample_means_std = np.std(single_sample_means)
married_sample_means_std = np.std(married_sample_means)

# Calculate standard error of the mean
```

```
single_se = single_sample_means_std / np.sqrt(sample_size)
married_se = married_sample_means_std / np.sqrt(sample_size)

# List to store confidence levels
confidence_levels = [0.90, 0.95, 0.99]

# Calculate confidence intervals for the current age group
for confidence_level in confidence_levels:
    # Calculate z-score based on confidence level
    if confidence_level == 0.90:
        z_score = 1.64
        # Calculate margin of error
        single_margin_of_error = z_score * single_se
        married_margin_of_error = z_score * married_se

        # Calculate confidence interval
        single_ci = (avg_single_sample_means - single_margin_of_error,
avg_single_sample_means + single_margin_of_error)
        married_ci = (avg_married_sample_means  -
married_margin_of_error, avg_married_sample_means  +
married_margin_of_error)

        print("90% Confidence Interval for average spending of single
customers (using CLT):", single_ci)
        print("90% Confidence Interval for average spending of married
customers (using CLT):", married_ci)



    elif confidence_level == 0.95:
        z_score = 1.96
        # Calculate margin of error
        single_margin_of_error = z_score * single_se
        married_margin_of_error = z_score * married_se

        # Calculate confidence interval
        single_ci = (avg_single_sample_means - single_margin_of_error,
avg_single_sample_means + single_margin_of_error)
        married_ci = (avg_married_sample_means  -
married_margin_of_error, avg_married_sample_means  +
married_margin_of_error)

        print("95% Confidence Interval for average spending of single
customers (using CLT):", single_ci)
        print("95% Confidence Interval for average spending of married
customers (using CLT):", married_ci)

    elif confidence_level == 0.99:
        z_score = 2.57
         # Calculate margin of error
```

```
        single_margin_of_error = z_score * single_se
        married_margin_of_error = z_score * married_se

        # Calculate confidence interval
        single_ci = (avg_single_sample_means - single_margin_of_error, 
avg_single_sample_means + single_margin_of_error)
        married_ci = (avg_married_sample_means  - 
married_margin_of_error, avg_married_sample_means  + 
married_margin_of_error)

        print("99% Confidence Interval for average spending of single 
customers (using CLT):", single_ci)
        print("99% Confidence Interval for average spending of married 
customers (using CLT):", married_ci)
```

```
Average spending per transaction for male customers: 9260.981264
Average spending per transaction for female customers: 
9262.720624000001
90% Confidence Interval for average spending of single customers 
(using CLT): (9252.87659065089, 9269.08593734911)
90% Confidence Interval for average spending of married customers 
(using CLT): (9254.299177987099, 9271.142070012904)
95% Confidence Interval for average spending of single customers 
(using CLT): (9251.295190973016, 9270.667337026984)
95% Confidence Interval for average spending of married customers 
(using CLT): (9252.655969008973, 9272.78527899103)
99% Confidence Interval for average spending of single customers 
(using CLT): (9248.280647837068, 9273.681880162932)
99% Confidence Interval for average spending of married customers 
(using CLT): (9249.523601894418, 9275.917646105585)
```

```
# Filter the dataset for different age groups
age_group_0_17 = data[data['Age'] == 0]['Purchase']
age_group_18_25 = data[data['Age'] == 1]['Purchase']
age_group_26_35 = data[data['Age'] == 2]['Purchase']
age_group_36_45 = data[data['Age'] == 3]['Purchase']
age_group_46_50 = data[data['Age'] == 4]['Purchase']
age_group_51_55 = data[data['Age'] == 5]['Purchase']
age_group_55plus = data[data['Age'] == 6]['Purchase']

# Define age group datasets
age_group_datasets = {
    '0-17': age_group_0_17,
    '18-25': age_group_18_25,
    '26-35': age_group_26_35,
    '36-45': age_group_36_45,
    '46-50': age_group_46_50,
    '51-55': age_group_51_55,
    '55+': age_group_55plus
}
```

```
# Define confidence levels
confidence_levels = [0.90, 0.95, 0.99]

# Function to calculate confidence interval
def calculate_confidence_interval(sample_mean, sample_se,
confidence_level):
    z_score = stats.norm.ppf((1 + confidence_level) / 2)
    margin_of_error = z_score * sample_se
    confidence_interval = (sample_mean - margin_of_error, sample_mean
+ margin_of_error)
    return confidence_interval

# Iterate over each age group dataset
for age_group, dataset in age_group_datasets.items():
    print(f"Age Group: {age_group}")

    # Calculate sample statistics
    sample_mean = dataset.mean()
    sample_std = dataset.std()
    sample_size = len(dataset)

    # Calculate standard error of the mean
    sample_se = sample_std / np.sqrt(sample_size)

    # Generate multiple random samples and compute sample means
    num_samples = 1000
    sample_means = [np.mean(np.random.choice(dataset,
size=sample_size, replace=True)) for _ in range(num_samples)]

    # Calculate confidence intervals for each confidence level
    for confidence_level in confidence_levels:
        confidence_interval =
calculate_confidence_interval(np.mean(sample_means), sample_se,
confidence_level)
        print(f"{confidence_level*100}% Confidence Interval for
average spending (using CLT):", confidence_interval)
```

```
Age Group: 0-17
90.0% Confidence Interval for average spending (using CLT):
(8865.053163406192, 9001.875055240343)
95.0% Confidence Interval for average spending (using CLT):
(8851.94743942098, 9014.980779225556)
99.0% Confidence Interval for average spending (using CLT):
(8826.333045325011, 9040.595173321524)
Age Group: 18-25
90.0% Confidence Interval for average spending (using CLT):
(9143.136932601223, 9195.598081908107)
95.0% Confidence Interval for average spending (using CLT):
(9138.111849746818, 9200.623164762512)
```

```
99.0% Confidence Interval for average spending (using CLT):
(9128.290610359902, 9210.444404149428)
Age Group: 26-35
90.0% Confidence Interval for average spending (using CLT):
(9235.06682796617, 9270.242092543696)
95.0% Confidence Interval for average spending (using CLT):
(9231.697503785075, 9273.61141672479)
99.0% Confidence Interval for average spending (using CLT):
(9225.112350800851, 9280.196569709015)
Age Group: 36-45
90.0% Confidence Interval for average spending (using CLT):
(9306.24855918068, 9356.067196611817)
95.0% Confidence Interval for average spending (using CLT):
(9301.47659394369, 9360.839161848808)
99.0% Confidence Interval for average spending (using CLT):
(9292.150058581701, 9370.165697210796)
Age Group: 46-50
90.0% Confidence Interval for average spending (using CLT):
(9170.911455764683, 9247.349132537543)
95.0% Confidence Interval for average spending (using CLT):
(9163.589739331537, 9254.670848970689)
99.0% Confidence Interval for average spending (using CLT):
(9149.279859893431, 9268.980728408795)
Age Group: 51-55
90.0% Confidence Interval for average spending (using CLT):
(9493.200889922367, 9578.494089896341)
95.0% Confidence Interval for average spending (using CLT):
(9485.030931725696, 9586.664048093013)
99.0% Confidence Interval for average spending (using CLT):
(9469.063211878, 9602.63176794071)
Age Group: 55+
90.0% Confidence Interval for average spending (using CLT):
(9280.139039838683, 9392.564543308643)
95.0% Confidence Interval for average spending (using CLT):
(9269.370166541989, 9403.333416605337)
99.0% Confidence Interval for average spending (using CLT):
(9248.323014556927, 9424.380568590399)
```