Business problem

Analyze the customer purchase behavior (specifically, purchase amount) against the customer's gender and the various other factors to help the business make better decisions. Understand if the spending habits differ between male and female customers

In [ ]:
```
from google.colab import drive
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
drive.mount('/content/drive')

data = pd.read_csv('/content/drive/MyDrive/Scaler notes/Business case studies/Case-
data.head()
```

```
Mounted at /content/drive
```

Out[ ]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Yea
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	4

In [ ]:
```
def basic_eda(data):
  print('Head of data')
  display(data.head())
  print('**********************************************************************
  print('Shape of data')
  print(data.shape)
  print('**********************************************************************
  print('Unique values in data')
  data.drop_duplicates(inplace = True)
  display(data.nunique())
  print('**********************************************************************
  print('Non-Null count and datatypes of columns in data')
  data.info()
  print('**********************************************************************
  print('Description of numerical data')
  display(data.describe())
  print('**********************************************************************
```

```
# Improvements: Add subplots functionality,
#               Option to select top/bottom n or given list of categories for categ
#               Can add run_all feature that runs all the functions and gives all t
#               Exception handling
class Plotter:
    def __init__(self, data):
        """
        Initializes the Plotter object.
        Parameters:
        - data (DataFrame): pandas DataFrame containing the data for plotting.
        """
        self.data = data

    def countplot(self, column, title=None, color= None, fontsize = None, bar_label
        """
        Creates a count plot for a specified column.
        Usage - Univariate categorical analysis
        If so many categories, recommended to filter for top few categories data on
        Parameters:
        - column (str): Name of the column to plot.
        - title (str): Title for the plot.
        """
        try:
          plt.figure(figsize=(10, 6))
          ax=sns.countplot(data=self.data, x=column, order=self.data[column].value_
          if bar_label:
            ax.bar_label(ax.containers[0])
          plt.title(title if title else f'Count Plot of {column}')
          plt.xlabel(column)
          plt.ylabel('Count')
          plt.xticks(rotation = 90, fontsize = fontsize)
          plt.show()
        except Exception as e:
          print(e)

    def pieplot(self, column, title=None, startangle = 90):
        """
        Creates a pie plot for a specified column.
        Usage - Univariate categorical analysis
        If so many categories, recommended to filter for top few categories data on
        Parameters:
        - column (str): Name of the column to plot.
        - title (str): Title for the plot.
        """
        try:
          plt.figure(figsize=(10, 6))
          self.data[column].value_counts().plot(kind='pie', autopct='%1.1f%%', star
          plt.title(title if title else f'Pie Plot of {column}')
          plt.ylabel('')
          plt.show()
        except Exception as e:
          print(e)
```

```
    def histogram(self, column, bins=30, use_bins = False, title=None, kde = False)
        """
        Creates a histogram for a specified column.
        Usage - Univariate numerical analysis
        Parameters:
        - column (str): Name of the column to plot.
        - bins (int): Number of bins for the histogram.
        - use_bins (Bool): if bins areto be used(use for numerical data, recommende
        - color (str): Color of the histogram bars.
        - title (str): Title for the plot.
        - kde (Bool) : Whether to include a kernel density estimate (KDE) plot.
        """
        try:
          plt.figure(figsize=(10, 6))
          if use_bins:
            sns.histplot(self.data[column], bins=bins, kde=kde)
          else:
            sns.histplot(self.data[column],  kde=kde)
          plt.title(title if title else f'Histogram of {column}')
          plt.xlabel(column)
          plt.ylabel('Frequency')
          plt.show()
        except Exception as e:
          print(e)

    def kdeplot(self, column, title=None):
        """
        Creates a kernel density plot for a specified column.
        Usage - Univariate numerical analysis
        Parameters:
        - column (str): Name of the column to plot.
        - title (str): Title for the plot.
        """
        try:
          plt.figure(figsize=(10, 6))
          sns.kdeplot(self.data[column], fill=True)
          plt.title(title if title else f'Kernel Density Plot of {column}')
          plt.xlabel(column)
          plt.ylabel('Density')
          plt.show()
        except Exception as e:
          print(e)

    def boxplot(self, column, title=None):
        """
        Creates a box plot for a specified column.
        Usage - Univariate numerical analysis
        Parameters:
        - column (str): Name of the column to plot.
        - title (str): Title for the plot.
        """
        try:
          plt.figure(figsize=(8, 6))
          sns.boxplot(y = self.data[column])
          plt.title(title if title else f'Boxplot of {column}')
          plt.ylabel(column)
```

```
            plt.show()
        except Exception as e:
            print(e)

    def lineplot(self, x_column, y_column, title=None, color = None, xlimit=None,yl
        """
        Creates a line plot between two specified columns.
        Usage - Bivariate numerical-numerical analysis(One numerical column like 'y
        Parameters:
        - x_column (str): Name of the column for the x-axis.
        - y_column (str): Name of the column for the y-axis.
        - title (str): Title for the plot.
        - xlimit (dict): Dictionary with 'left' and 'right' keys for x-axis limits.
        - ylimit (dict): Dictionary with 'left' and 'right' keys for y-axis limits.
        - color (str): Color of the line plot.
        """
        try:
            plt.figure(figsize=(10, 6))
            sns.lineplot(data=self.data, x=x_column, y=y_column, color = color)
            if xlimit:
                plt.xlim(left = xlimit['left'], right = xlimit['right'])
            if ylimit:
                plt.ylim(left = ylimit['left'], right = ylimit['right'])
            plt.title(title if title else f'Line Plot of {x_column} vs {y_column}')
            plt.xlabel(x_column)
            plt.ylabel(y_column)
            plt.show()
        except Exception as e:
            print(e)

    def scatterplot(self, x_column, y_column,  title=None):
        """
        Creates a scatter plot between two specified columns.
        Usage - Bivariate numerical-numerical analysis
        Parameters:
        - x_column (str): Name of the column for the x-axis.
        - y_column (str): Name of the column for the y-axis.
        - title (str): Title for the plot.
        """
        try:
            plt.figure(figsize=(10, 6))
            sns.scatterplot(data=self.data, x=x_column, y=y_column)
            plt.title(title if title else f'Scatterplot of {x_column} vs {y_column}')
            plt.xlabel(x_column)
            plt.ylabel(y_column)
            plt.show()
        except Exception as e:
            print(e)

    def dodgedcountplot(self, x_column, hue,  title=None):
        """
        Creates a dodged countplotplot between two specified columns.
        Usage - Bivariate categorical-categorical analysis
        If so many categories, recommended to filter for top few categories data on
        Parameters:
        - x_column (str): Name of the column for the x-axis.
```

```
        - hue (str): Column name to be used for color coding.
        - title (str): Title for the plot.
        """
        try:
          plt.figure(figsize=(10, 6))
          sns.countplot(data=self.data, x=x_column, hue=hue)
          plt.title(title if title else f'barplot distribution of {hue} for {x_colu
          plt.ylabel('Count')
          plt.show()
        except Exception as e:
          print(e)

    def stackedcountplot(self, x_column, hue,  title=None):
        """
        Creates a stacked countplot between two specified columns.
        Usage - Bivariate categorical-categorical analysis
        If so many categories, recommended to filter for top few categories data on
        Parameters:
        - x_column (str): Name of the column for the x-axis.
        - hue (str): Column name to be used for color coding.
        - title (str): Title for the plot.
        """
        try:
          plt.figure(figsize=(10, 6))
          data = pd.crosstab(index=self.data[x_column], columns=self.data[hue])
          data.plot(kind='bar', stacked=True, figsize=(10, 6))
          plt.title(title if title else f'barplot distribution of {hue} for {x_colu
          plt.xticks(rotation=90)
          plt.legend()
          plt.show()
        except Exception as e:
          print(e)

    def bivariateboxplot(self, categorical_column, numerical_column, title=None, ro
        """
        Creates a boxplot between two specified columns.
        Usage - Bivariate categorical-numerical
        If so many categories, recommended to filter for top few categories data on
        Parameters:
        - categorical_column (str): Name of the column for the x-axis.
        - numerical_column (str): Name of the column for the y-axis.
        - title (str): Title for the plot.
        - rotate_xaxis_ticks (bool): Whether to rotate x-axis ticks for better read
        """
        try:
          sns.boxplot(data=self.data, x=categorical_column, y=numerical_column)
          plt.title(title if title else f'Boxplot of {numerical_column} for {catego
          if rotate_xaxis_ticks:
            plt.xticks(rotation=90)
          plt.xlabel(categorical_column)
          plt.ylabel(numerical_column)
          plt.show()
        except Exception as e:
          print(e)

    def bivariatebarplot(self, categorical_column, numerical_column, title=None, ro
```

```
        """
        Creates a boxplot between two specified columns.
        Usage - Bivariate categorical-numerical
        If so many categories, recommended to filter for top few categories data on
        Parameters:
        - categorical_column (str): Name of the column for the x-axis.
        - numerical_column (str): Name of the column for the y-axis.
        - title (str): Title for the plot.
        """
        try:
          sns.barplot(data=self.data, x=categorical_column, y=numerical_column, est
          plt.title(title if title else f'Barplot of {numerical_column} for {catego
          if rotate_xaxis_ticks:
            plt.xticks(rotation=90)
          plt.xlabel(categorical_column)
          plt.ylabel(numerical_column)
          plt.show()
        except Exception as e:
          print(e)

    def jointplot(self, x_column, y_column, title=None):
        """
        Creates a joint plot between two specified columns.
        Usage - Bivariate numerical-numerical analysis
        Parameters:
        - x_column (str): Name of the column for the x-axis.
        - y_column (str): Name of the column for the y-axis.
        - title (str): Title for the plot.
        """
        try:
          sns.jointplot(data=self.data, x=x_column, y=y_column, kind='reg')
          plt.title(title if title else f'Joint Plot of {x_column} vs {y_column}')
          plt.xlabel(x_column)
          plt.ylabel(y_column)
          plt.show()
        except Exception as e:
          print(e)

    def trivariatescatterplot_CNN(self, numerical_column1, numerical_column2, categ
        """
        Creates a scatter plot between two specified columns.
        Usage - Trivariate CNN analysis
        Parameters:
        - numerical_column1 (str): Name of the column for the x-axis.
        - numerical_column2 (str): Name of the column for the y-axis.
        - categorical_column (str): Column name to be used for color coding.
        - title (str): Title for the plot.
        """
        try:
          plt.figure(figsize=(10, 6))
          sns.scatterplot(data=self.data, x=numerical_column1, y=numerical_column2,
          plt.title(title if title else f'Scatter plot of {numerical_column1} vs {n
          plt.xlabel(numerical_column1)
          plt.ylabel(numerical_column2)
          plt.show()
        except Exception as e:
```

```
            print(e)

    def trivariateboxplot(self, categorical_column1, categorical_column2, numerical
        """
        Creates a boxplot between two specified columns.
        Usage - Trivariate CCN analysis
        If so many categories, recommended to filter for top few categories data on
        Parameters:
        - categorical_column1 (str): Name of the column for the x-axis.
        - categorical_column2 (str): Name of the column for the x-axis.
        - numerical_column (str): Name of the column for the y-axis.
        - title (str): Title for the plot.
        - rotate_xaxis_ticks (bool): Whether to rotate x-axis ticks for better read
        """
        try:
            plt.figure(figsize=(12,8))
            sns.boxplot(x=categorical_column1,y=numerical_column,hue=categorical_colu
            plt.title(title if title else f'Boxplot of {numerical_column} for {catego
            if rotate_xaxis_ticks:
                plt.xticks(rotation=90)
            plt.xlabel(categorical_column1)
            plt.ylabel(numerical_column)
            plt.show()
        except Exception as e:
            print(e)

    #improvements : add sizes range functionality
    def trivariatescatterplot_NNN(self, x_column, y_column, size, title=None):
        """
        Creates a scatter plot between two specified columns.
        Usage - Trivariate NNN analysis
        Parameters:
        - x_column (str): Name of the column for the x-axis.
        - y_column (str): Name of the column for the y-axis.
        - size (str): Column name to be used for size coding.(rank like column)
        - title (str): Title for the plot.
        """
        try:
            plt.figure(figsize=(10, 6))
            sns.scatterplot(x=x_column, y=y_column, size=size, data=self.data)
            plt.title(title if title else f'Scatterplot of {x_column} vs {y_column} f
            plt.xlabel(x_column)
            plt.ylabel(y_column)
            plt.show()
        except Exception as e:
            print(e)

    def trivariatejointplot(self, numerical_column1, numerical_column2, categorical
        """
        Creates a joint plot between two specified columns.
        Usage - trivariate CNN analysis
        Parameters:
        - numerical_column1 (str): Name of the column for the x-axis.
        - numerical_column2 (str): Name of the column for the y-axis.
        - categorical_column (str): Column name to be used for color coding.
        - title (str): Title for the plot.
```

```
        """
        try:
            sns.jointplot(x=numerical_column1, y=numerical_column2, data=self.data, h
            # plt.title(title if title else f'Joint Plot of {numerical_column1} vs {n
            plt.show()
        except Exception as e:
            print(e)

    def pairplot(self, columns=None, vars = None,hue=None):
        """
        Creates pair plot for multiple columns.
        Usage : Multivariate numerical analysis(one category column can be added op
        Parameters:
        - columns (list): List of column names to include in the pair plot(optional
        - vars (list): dictionary of lists for x and y column names to include in t
        - hue (str): Column name to be used for color coding.
        """
        try:
            if vars:
                sns.pairplot(self.data, x_vars=vars['x'], y_vars=vars['y'], hue=hue)
            else:
                sns.pairplot(self.data if columns is None else self.data[columns], hue=
            plt.show()
        except Exception as e:
            print(e)

    def heatmap(self, columns=None, title="Correlation Heatmap"):
        """
        Creates a heatmap for correlation between specified columns.
        Usage - Multivariate numerical analysis(correlation analysis)
        Parameters:
        - columns (list): List of column names to include in the heatmap(optional.
        - title (str): Title for the heatmap.
        """
        try:
            plt.figure(figsize=(10, 8))
            corr = self.data.select_dtypes(include=[float,int]).corr() if columns is
            sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
            plt.title(title)
            plt.show()
        except Exception as e:
            print(e)

    def multivariatepieplot(self, columns, title=None, explode = None):
        """
        Creates a pie plot for a specified column.
        Usage - Multivariate numerical analysis(to check proportion for given colum
        Parameters:
        - columns (list): List of names of the columns to plot.
        - title (str): Title for the plot.
        - explode (list): List of explode values for each slice of pie chart.
        """
        try:
            plt.figure(figsize=(10, 6))
            data = self.data[columns].T.sum(axis='columns')
            plt.pie(x=data, labels=data.index,startangle=90,explode = explode,shadow=
```

```
            columns_list = ", ".join(columns)  # Join all column names with a comma a
            plt.title(title if title else f'Pie Plot of share of {columns_list}')
            plt.show()
        except Exception as e:
            print(e)
```

In [ ]:
```
basic_eda(data)
```

Head of data

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

```
**********************************************************************************
*
Shape of data
(550068, 10)
**********************************************************************************
*
Unique values in data
```

	0
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

dtype: int64

```
**********************************************************************************
*
Non-Null count and datatypes of columns in data
<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
**********************************************************************************
*
Description of numerical data
```

	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

```
**********************************************************************************
*
```

Univariate analysis

In [ ]:
```
data.nunique()
```

Out[ ]:

	0
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

dtype: int64

In [ ]:
```
user_level_data = data.groupby(['User_ID', 'Gender', 'Age', 'Occupation', 'City_Cat

plotter_obj_user_level = Plotter(user_level_data)
```

In [ ]:
```
user_level_data.head()
```

Out[ ]:

	User_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_St
0	1000001	F	0-17	10	A	2	
1	1000002	M	55+	16	C	4+	
2	1000003	M	26-35	15	A	3	
3	1000004	M	46-50	7	B	2	
4	1000005	M	26-35	20	A	1	

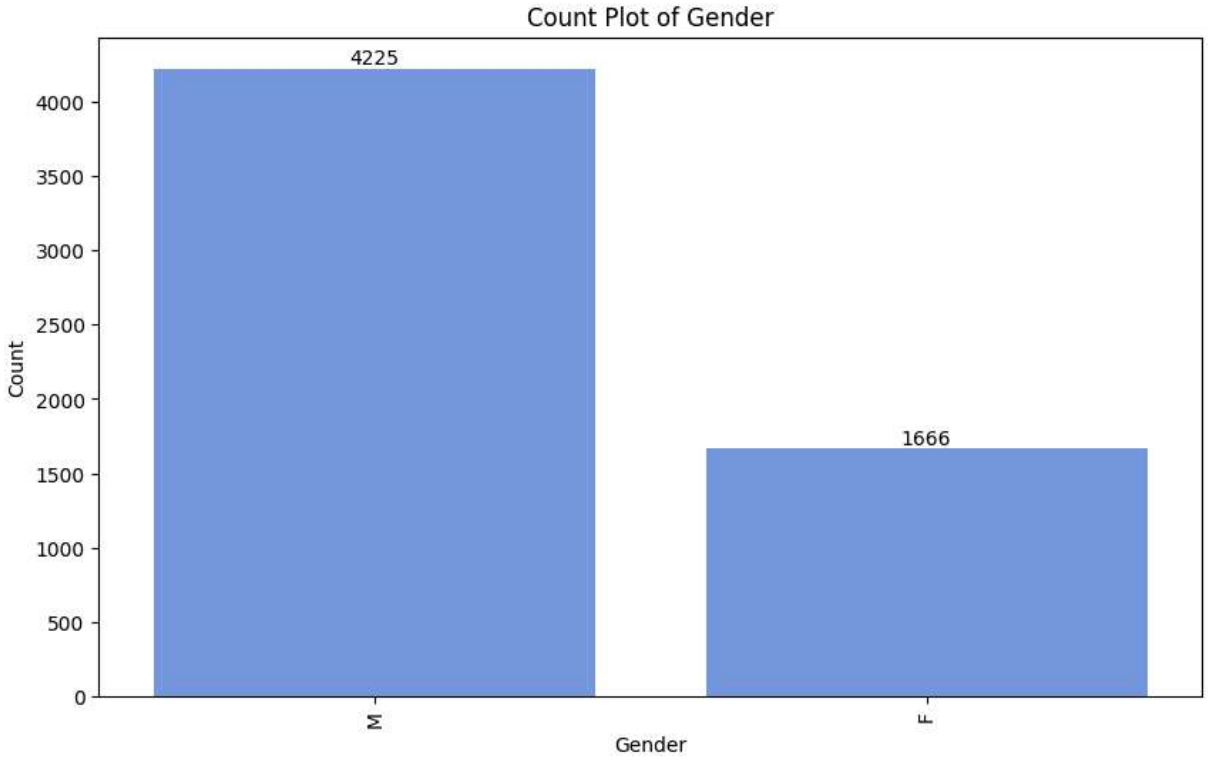
In [ ]:
```
data.drop_duplicates(inplace = True)

data.nunique()
```

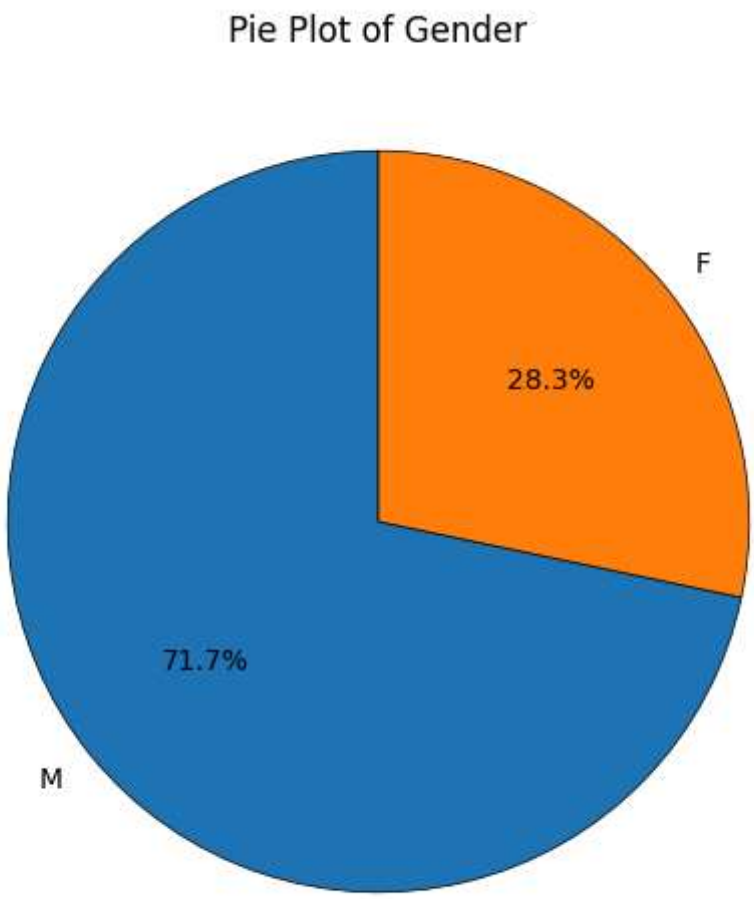
Out[ ]:

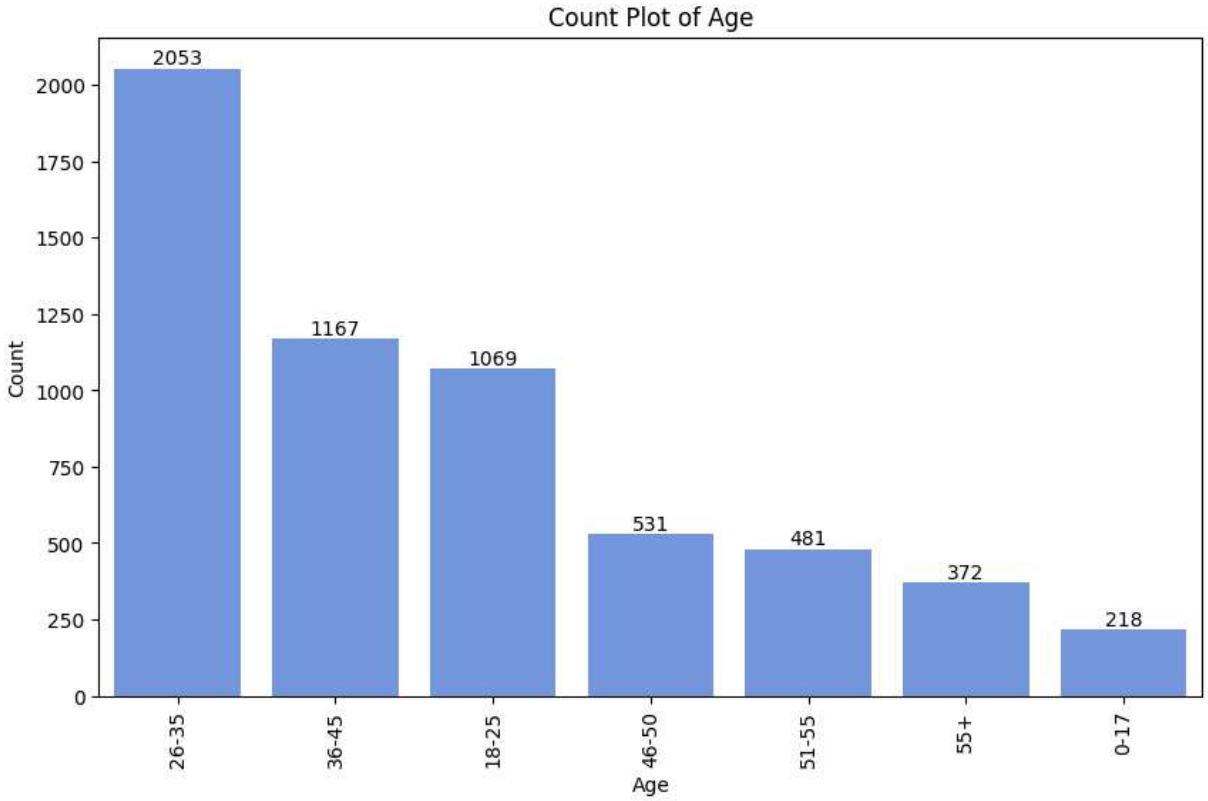
	0
User_ID	5891
Product_ID	3631
Gender	2
Age	7
Occupation	21
City_Category	3
Stay_In_Current_City_Years	5
Marital_Status	2
Product_Category	20
Purchase	18105

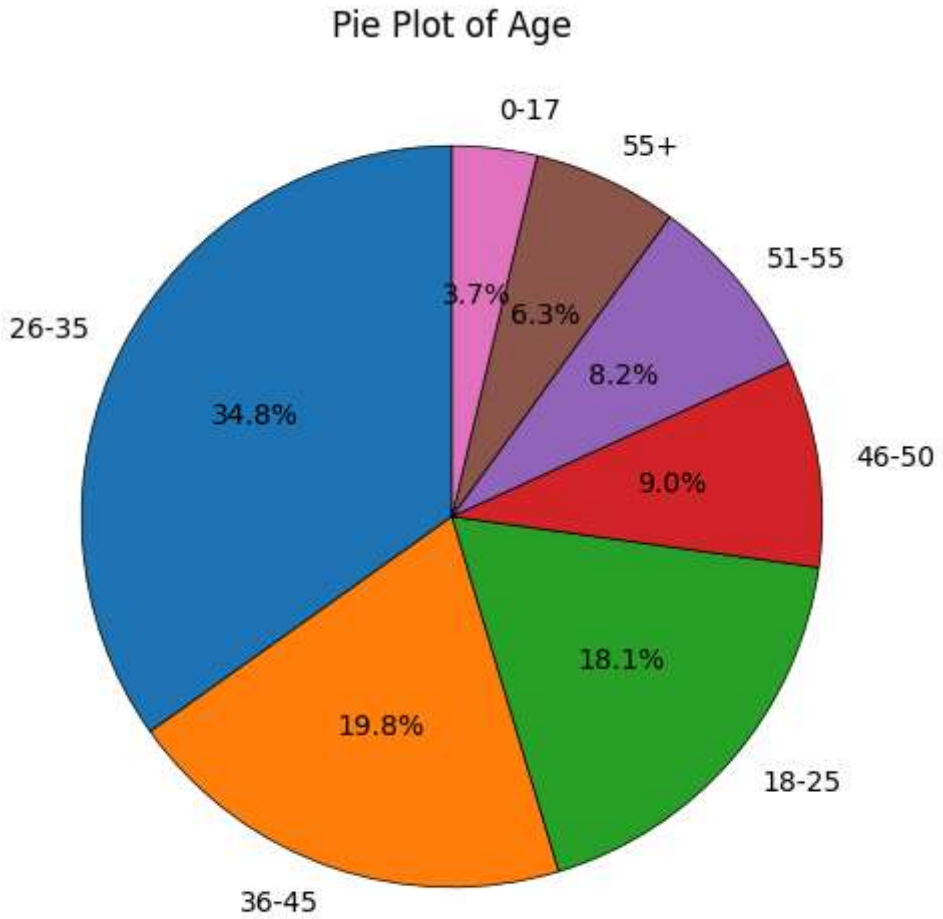
dtype: int64

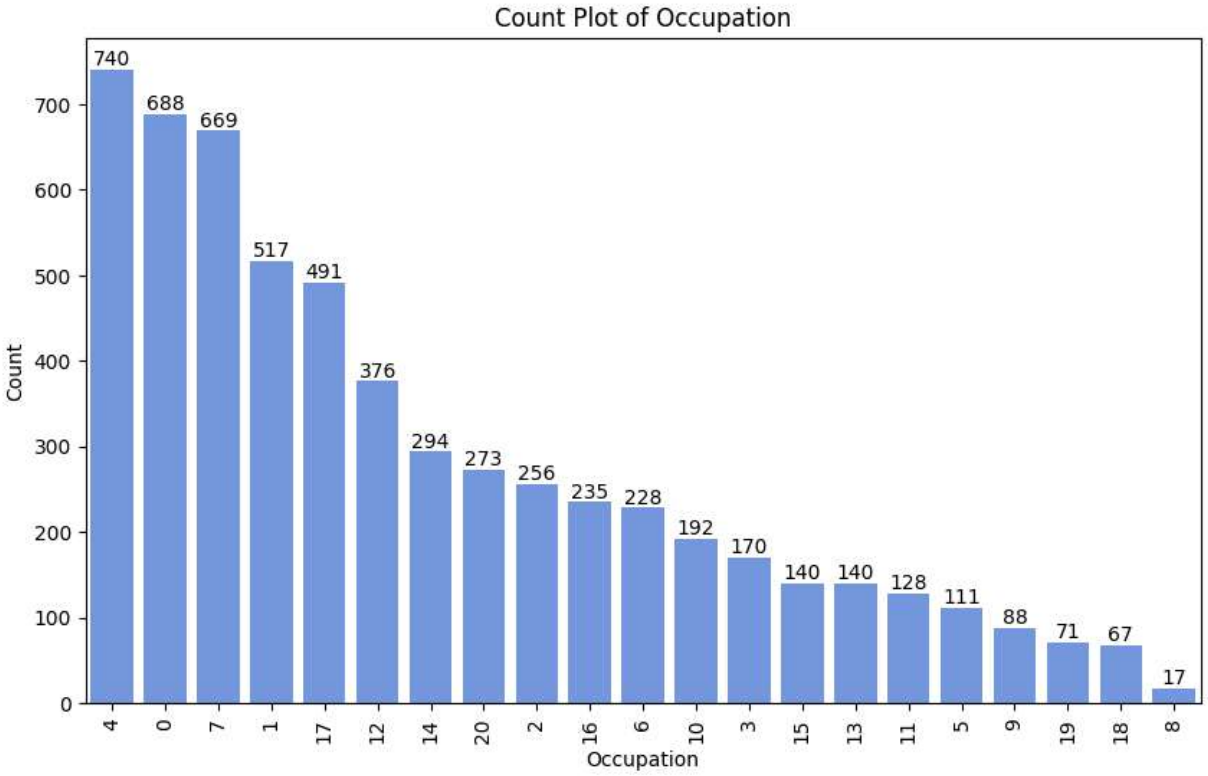
In [ ]:
```
categorical_cols = ['Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Curren
numerical_cols = ['Purchase']
for col in categorical_cols:
  plotter_obj_user_level.countplot(column = col, bar_label = True)
  plotter_obj_user_level.pieplot(column = col, startangle = 90)

for col in numerical_cols:
  plotter_obj_user_level.histogram(column = col, bins=30, use_bins = False, kde = F
  plotter_obj_user_level.kdeplot(column = col)
  plotter_obj_user_level.boxplot(column = col)
```

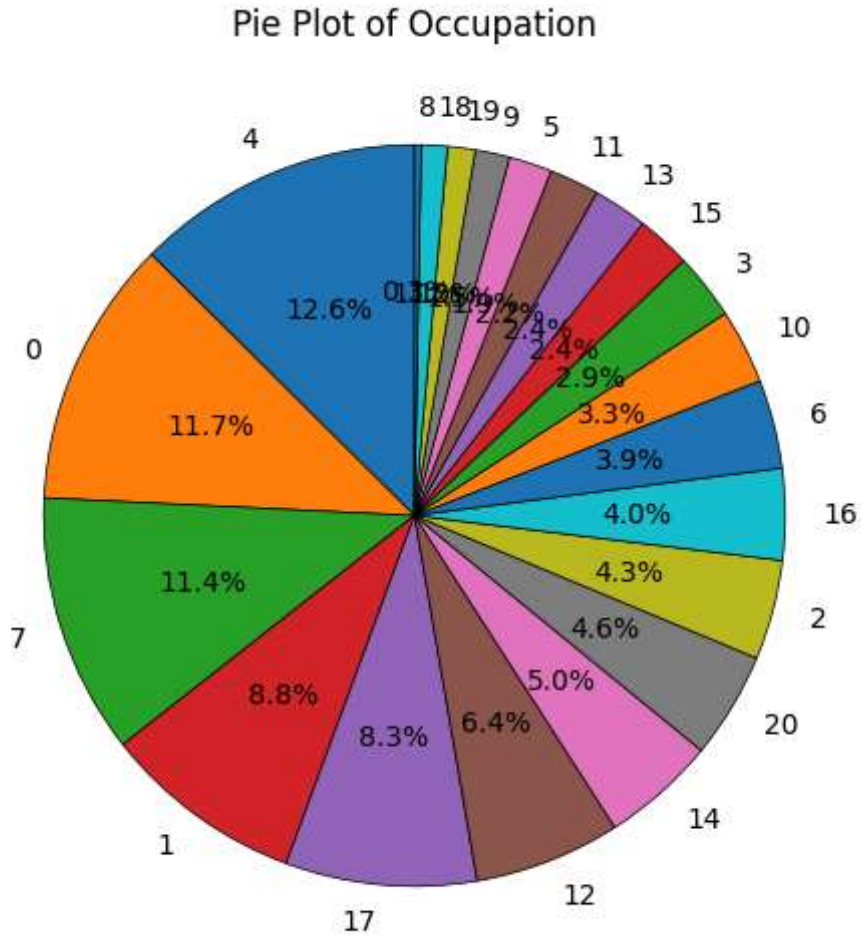

Count Plot of Gender
4225
1666
4000
3500
3000
2500
2000
1500
1000
500
0
Count
M
F
Gender

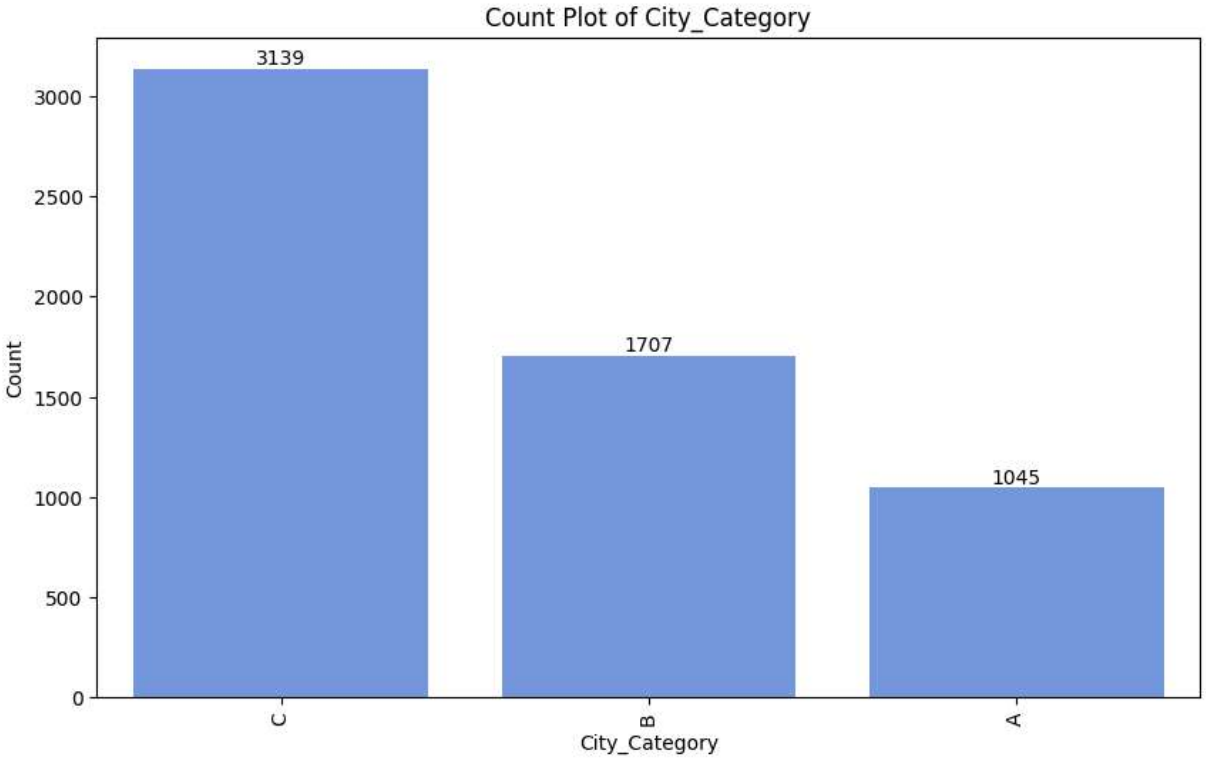

Pie Plot of Gender
F
28.3%
71.7%
M

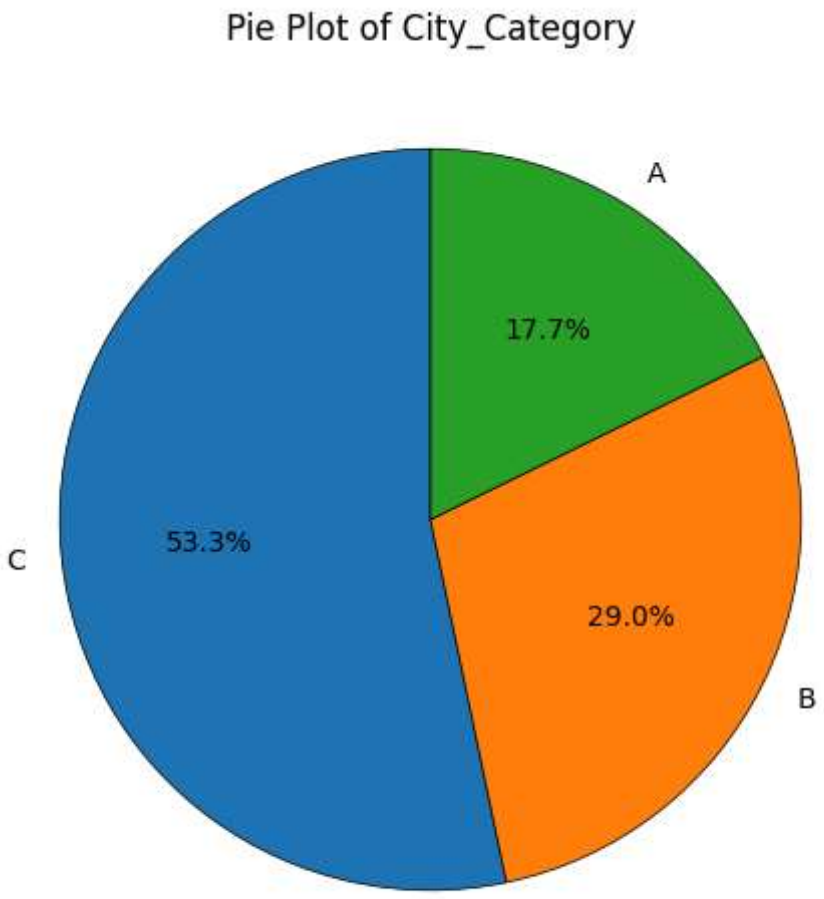

Count Plot of Age
Count
Age
2053
1167
1069
531
481
372
218
0
250
500
750
1000
1250
1500
1750
2000
26-35
36-45
18-25
46-50
51-55
55+
0-17

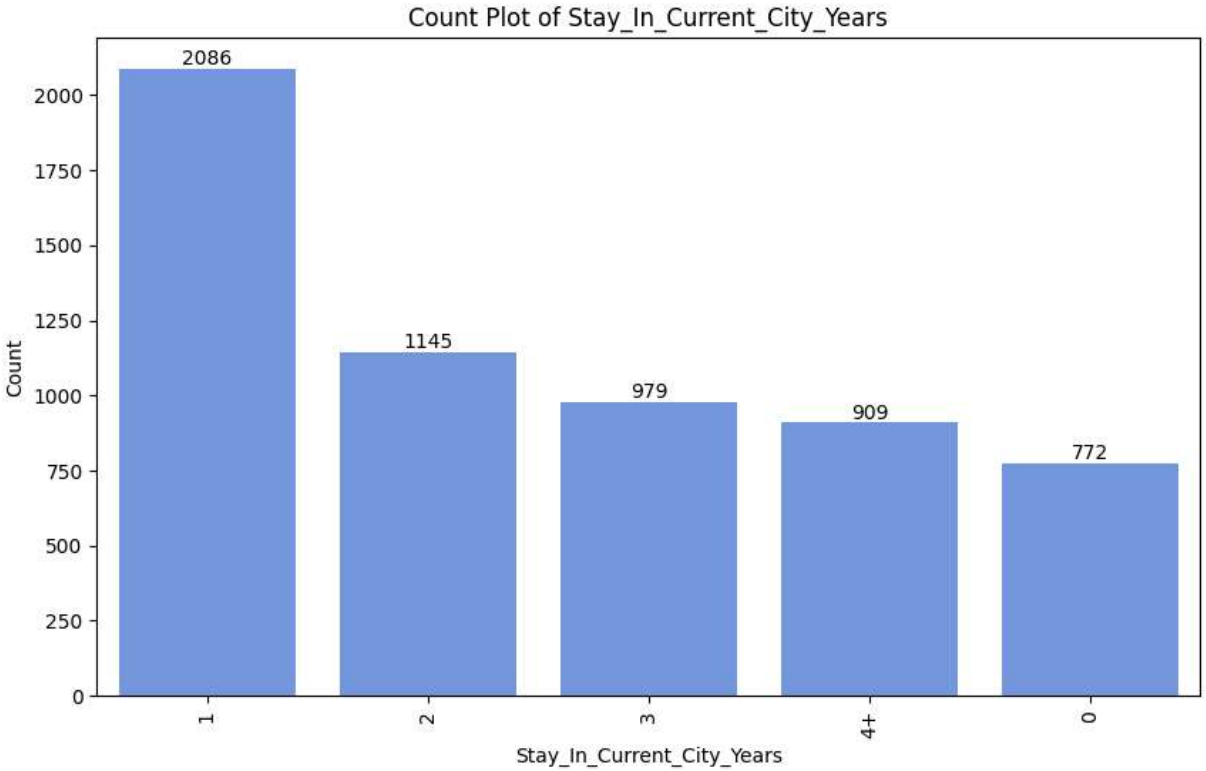

Pie Plot of Age
0-17
55+
51-55
46-50
18-25
36-45
26-35
3.7%
6.3%
8.2%
9.0%
18.1%
19.8%
34.8%

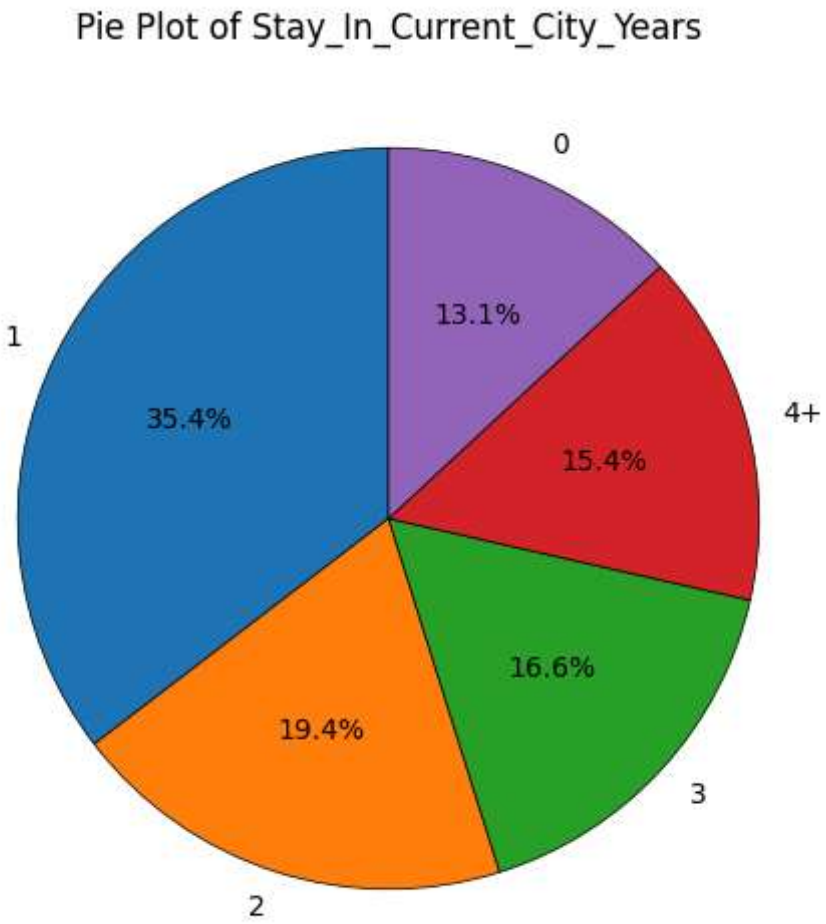

Count Plot of Occupation
Count
Occupation
740
688
669
517
491
376
294
273
256
235
228
192
170
140
140
128
111
88
71
67
17
4
0
7
1
17
12
14
20
2
16
6
10
3
15
13
11
5
9
19
18
8

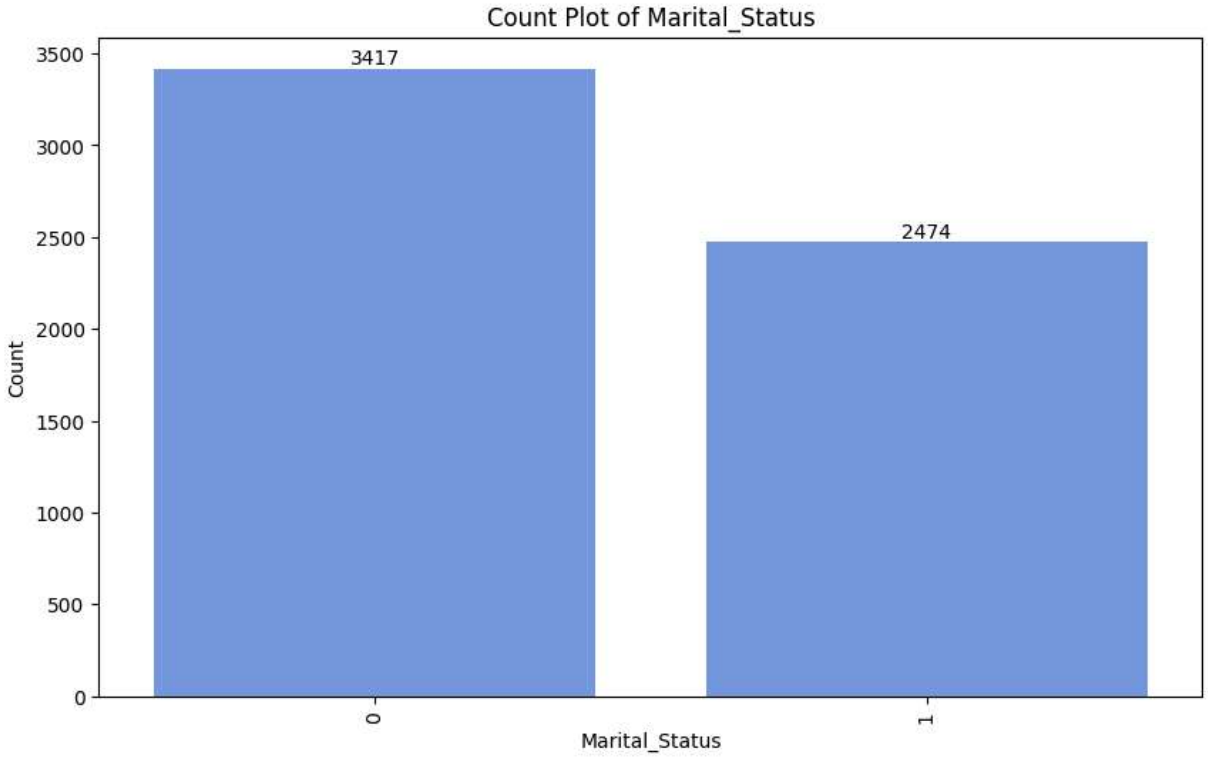

Pie Plot of Occupation
12.6%
11.7%
11.4%
8.8%
8.3%
6.4%
5.0%
4.6%
4.3%
4.0%
3.9%
3.3%
2.9%
2.4%
4
0
7
1
17
12
14
20
2
16
6
10
3
15
13
11
5
9
19
18
8

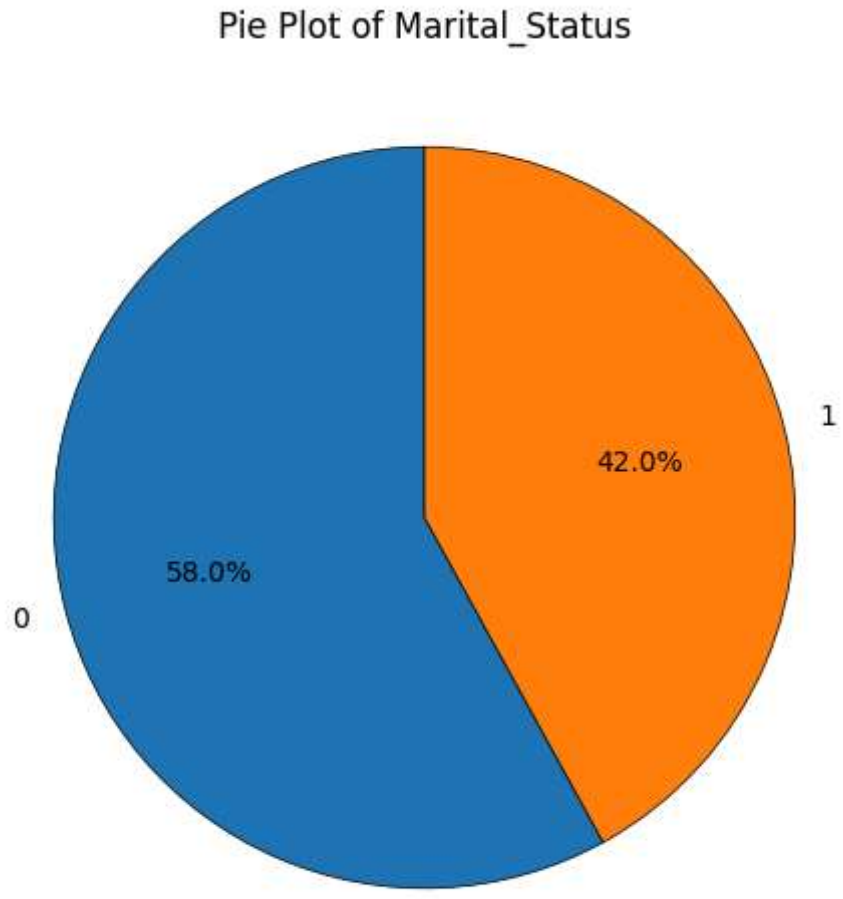

Count Plot of City_Category
3139
1707
1045
3000
2500
2000
1500
1000
500
0
Count
C
B
A
City_Category


Pie Plot of City_Category
A
17.7%
C
53.3%
29.0%
B


Count Plot of Stay_In_Current_City_Years
Count
Stay_In_Current_City_Years
2086
1145
979
909
772
0
250
500
750
1000
1250
1500
1750
2000
1
2
3
4+
0

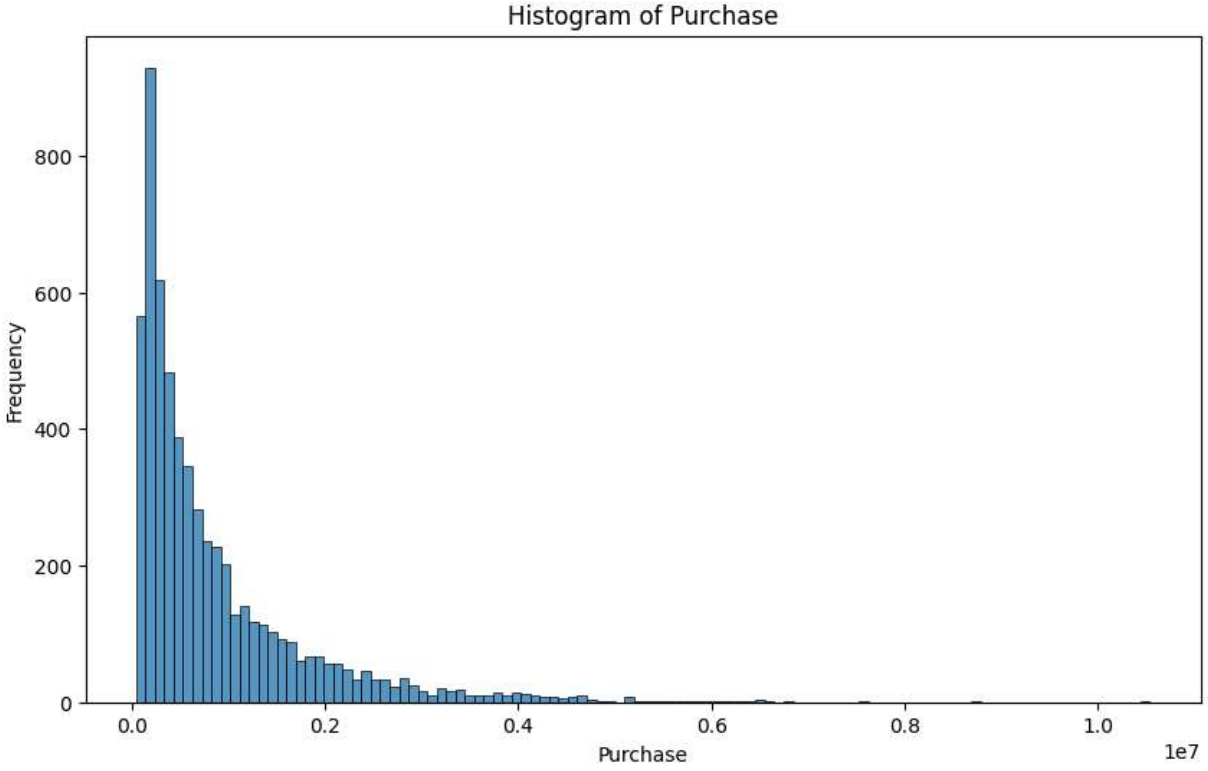

Pie Plot of Stay_In_Current_City_Years
0
13.1%
1
35.4%
4+
15.4%
16.6%
19.4%
3
2

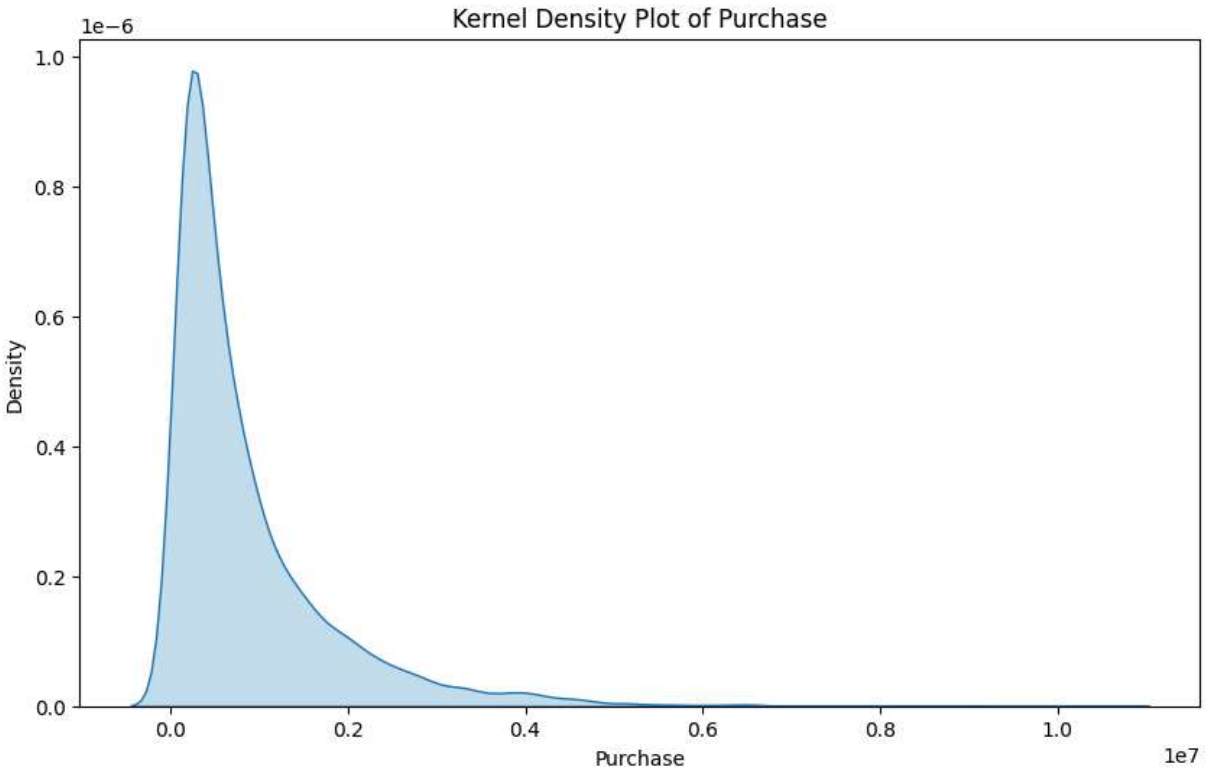


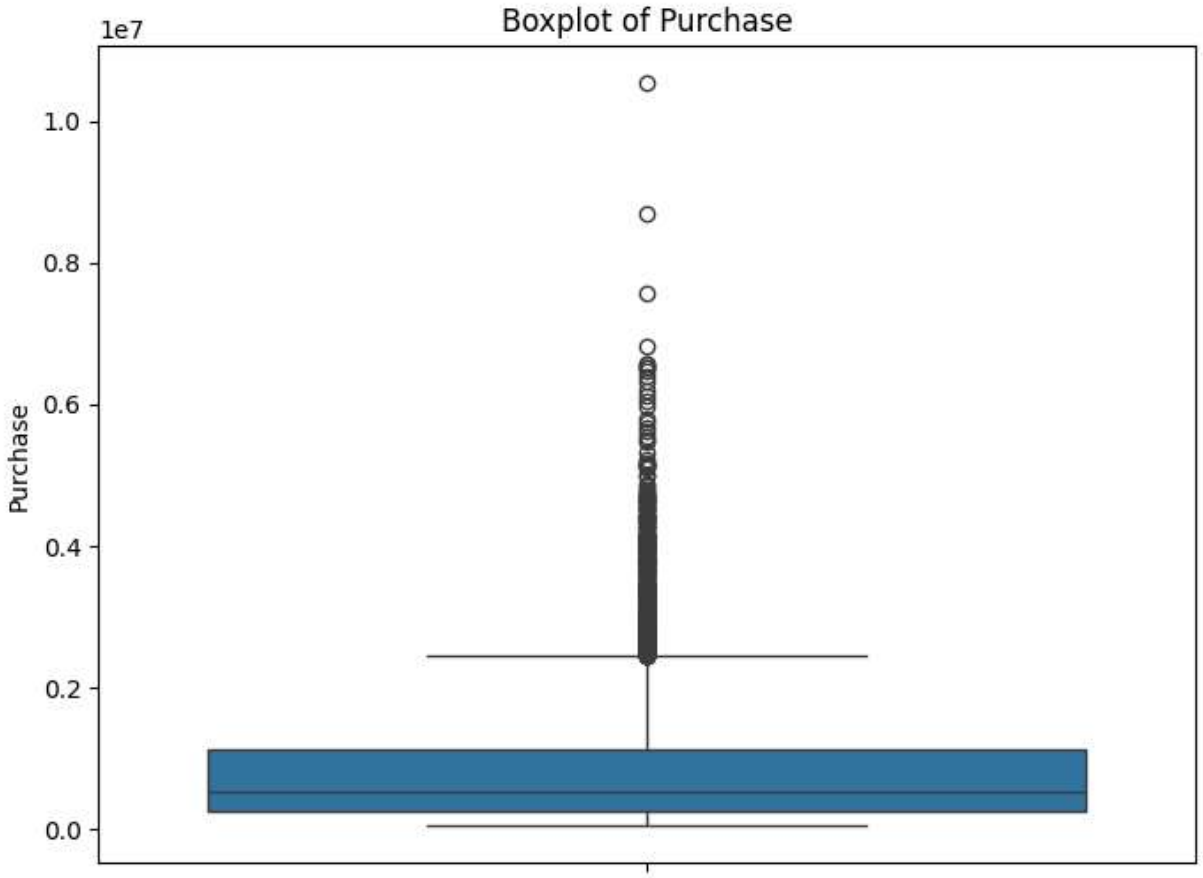



```
'Product_Category'
'Product_Category'
<Figure size 1000x600 with 0 Axes>
```

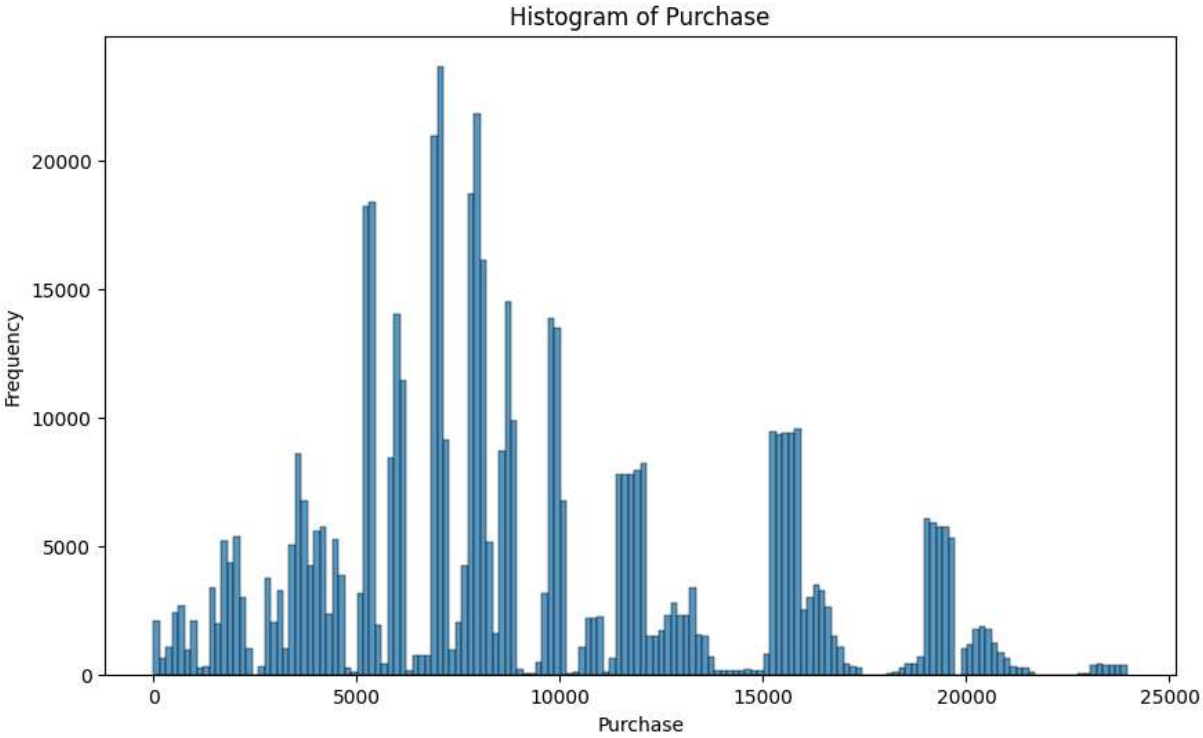
```
<Figure size 1000x600 with 0 Axes>
```

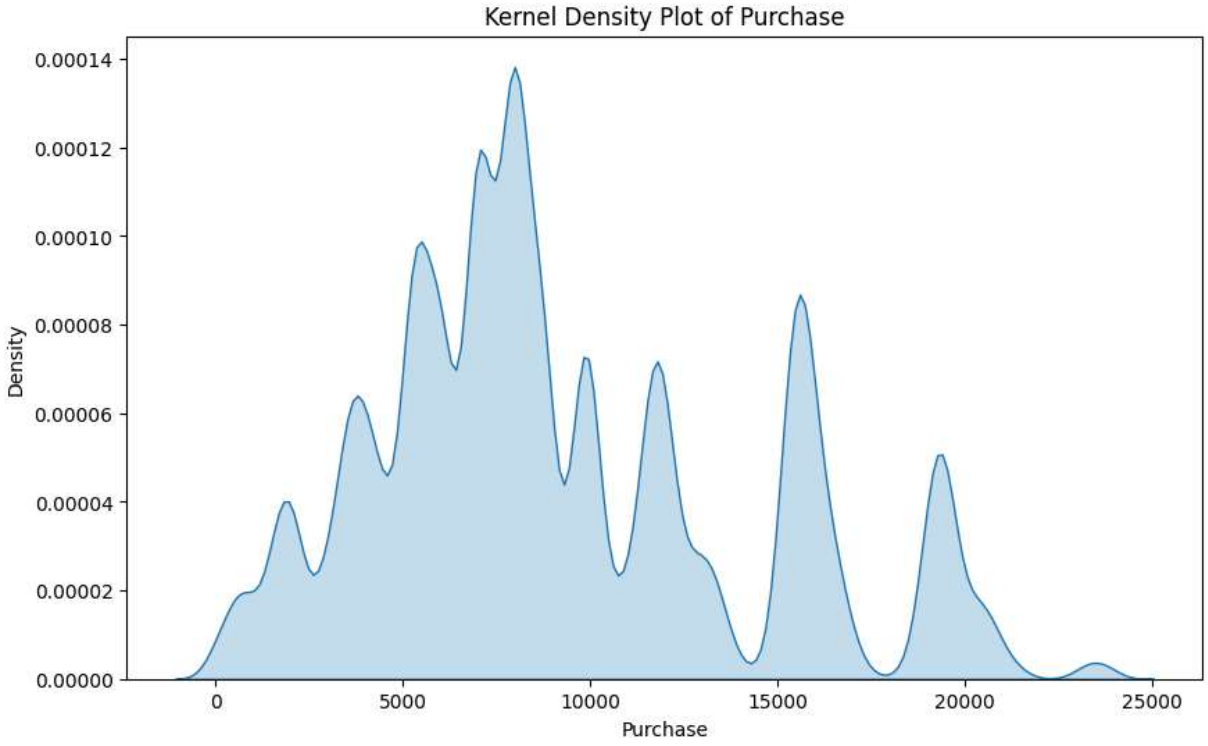


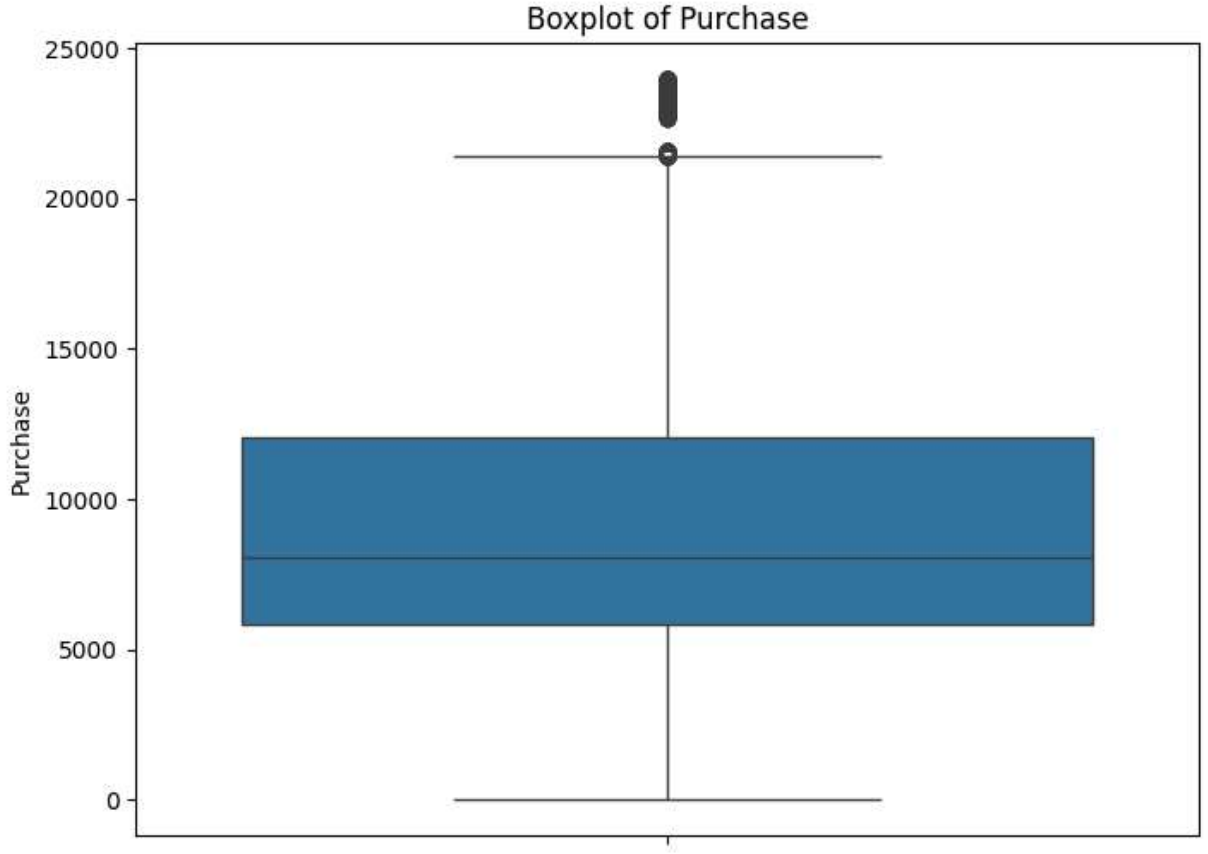




In [ ]:
```
plotter_obj = Plotter(data)
for col in numerical_cols:
  plotter_obj.histogram(column = col, bins=30, use_bins = False, kde = False)
  plotter_obj.kdeplot(column = col)
  plotter_obj.boxplot(column = col)
```



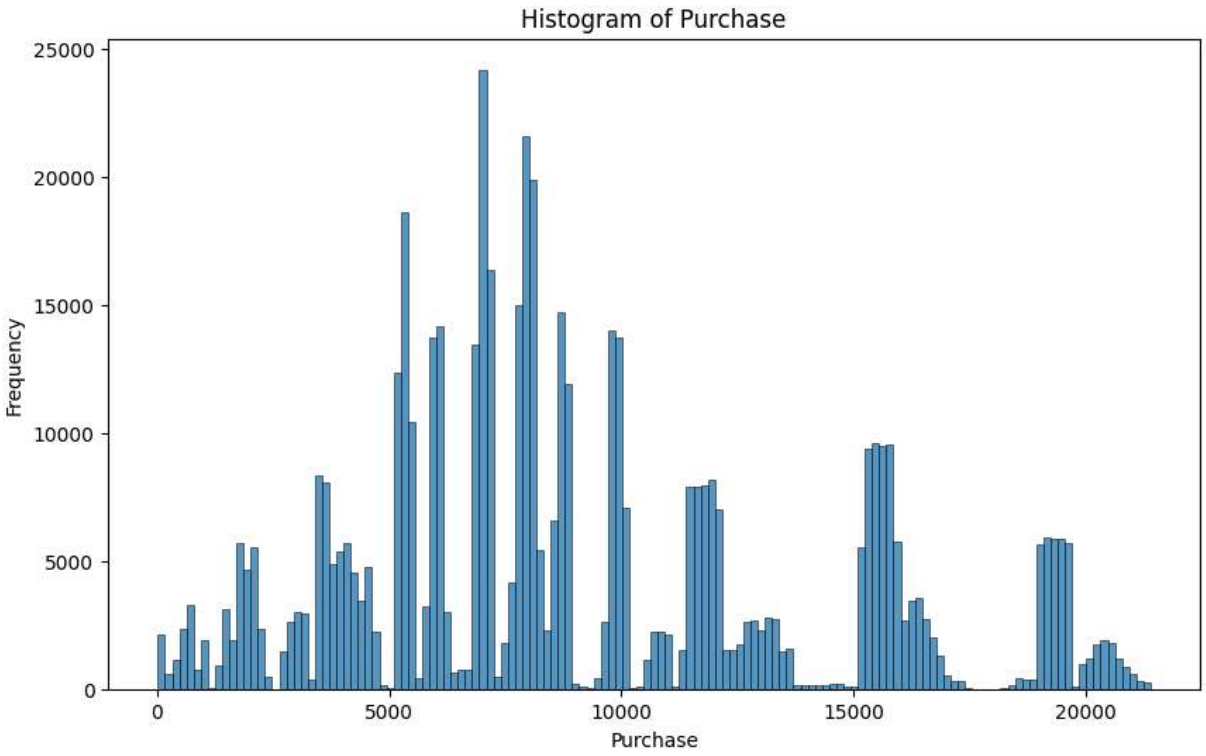


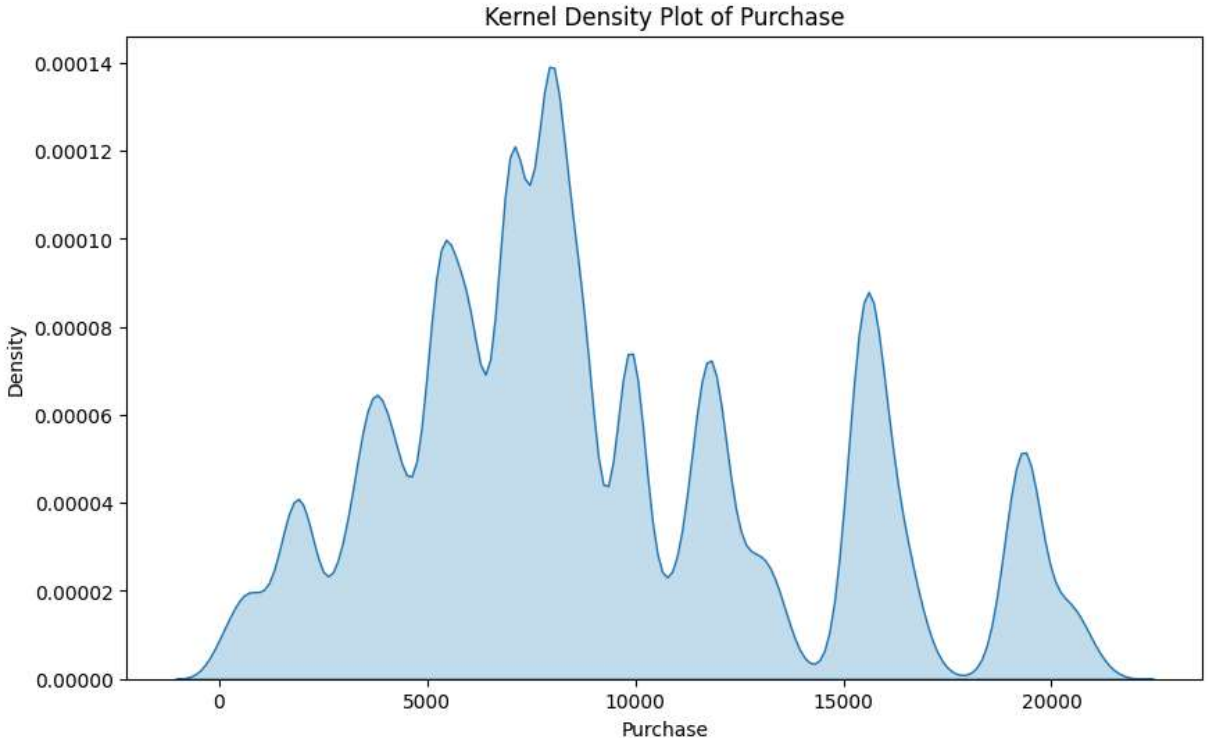


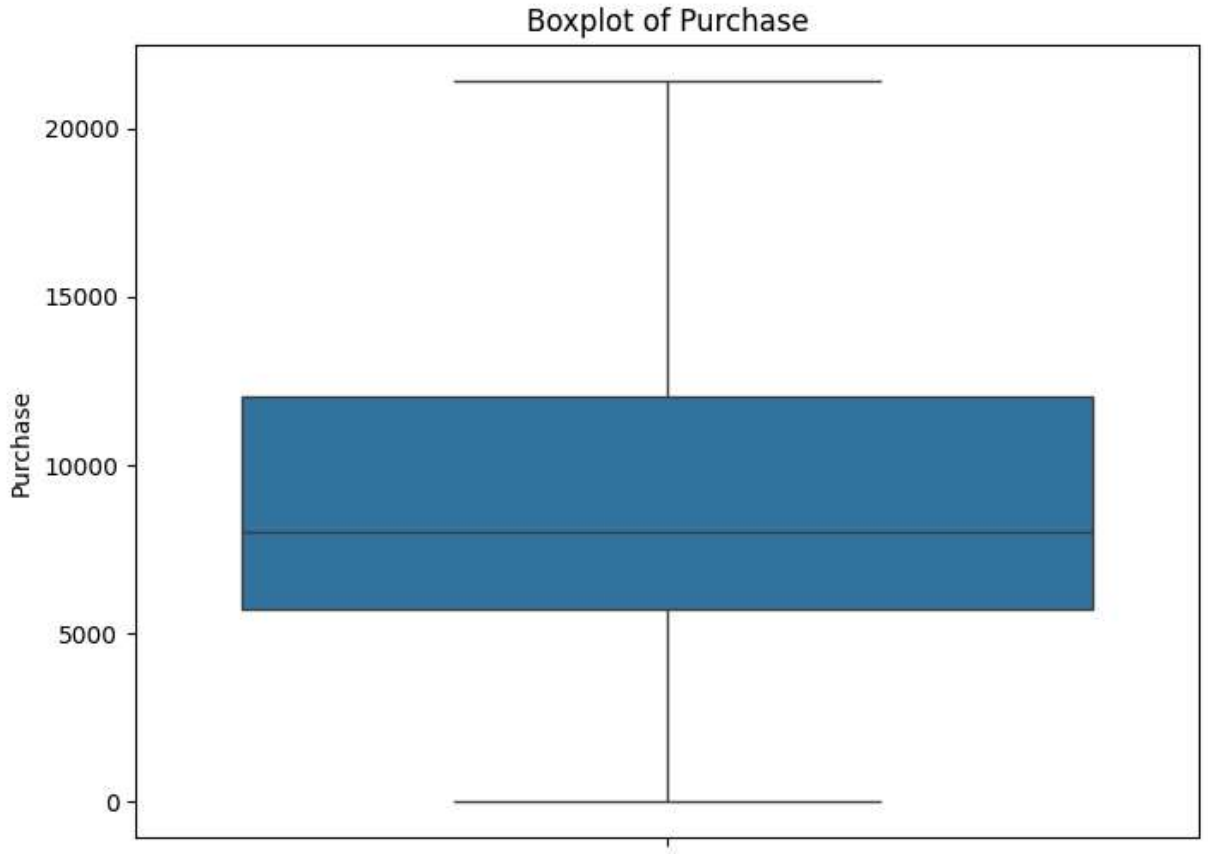
In [ ]:
```
def remove_outliers(df, column):
    """
    Removes outliers from a specific column in a DataFrame based on the 1.5*IQR met

    Parameters:
        df (pd.DataFrame): The DataFrame containing the data.
        column (str): The column name for which to remove outliers.
```

```
    Returns:
        pd.DataFrame: A filtered DataFrame with outliers removed.
    """
    # Calculate Q1 (25th percentile) and Q3 (75th percentile)
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1

    # Define the lower and upper bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    print(upper_bound)

    # Filter the DataFrame to remove outliers
    filtered_df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

    return filtered_df

outlier_removed_data = remove_outliers(data, 'Purchase')

plotter_obj = Plotter(outlier_removed_data)
for col in numerical_cols:
  plotter_obj.histogram(column = col, bins=30, use_bins = False, kde = False)
  plotter_obj.kdeplot(column = col)
  plotter_obj.boxplot(column = col)
```

```
21400.5
```







Insights

72% customers(4.2k) are Male and only 28% customers(1.6k) are Females.

35% of customers(2k) are from 26-35 age group followed by 20% from 36-45. The minumum number of customers are from age group 0-17(only 4%).

Maximum number of customers are from Occupation 4(13%) followed by Occupation 0,7 and 1.

53% customers(3k) are from city category C followedby B(29% customers) and A(18% customers).

35% of customers(2k) have been staying in the current city for only 1 year followed by 19% for 2 years, 17% for3 years, 15% for 4+ years and 13% for 0 years.

58% customers are Single and only 42% are married.

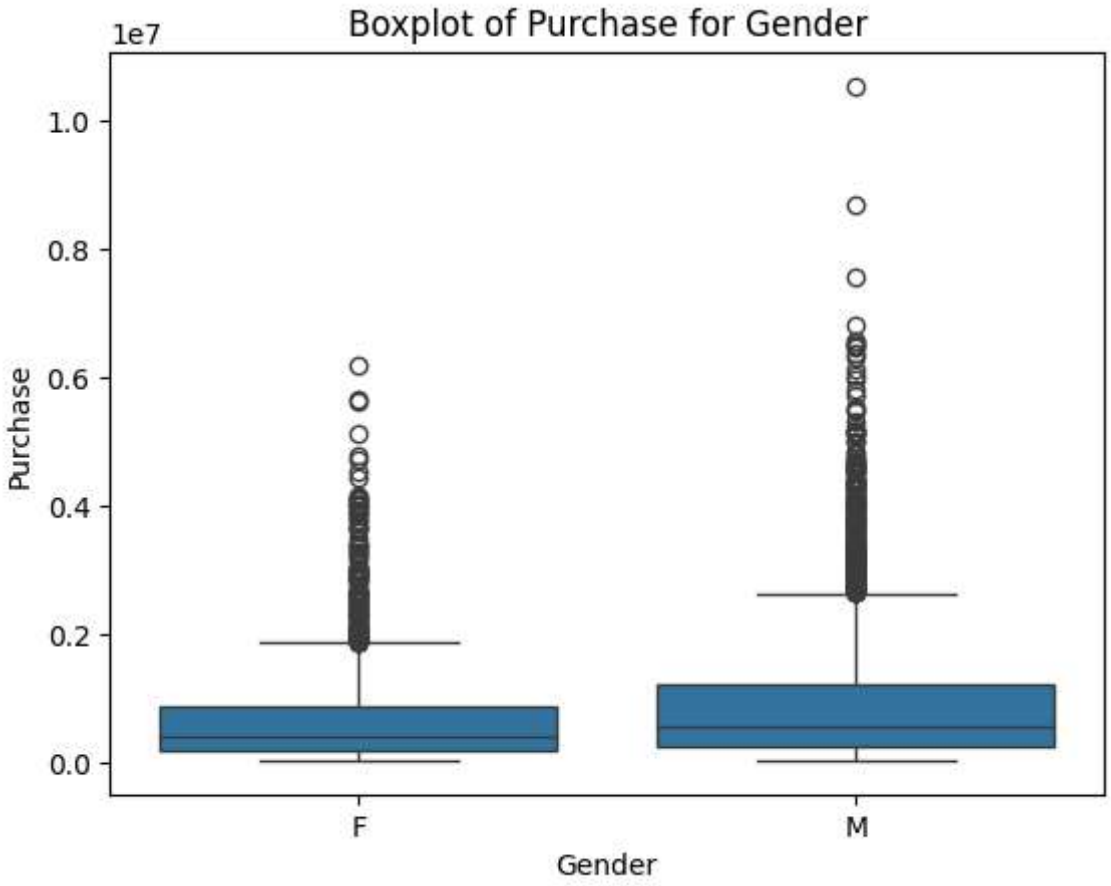
75% of customers have a purchase amount spent between 0-1000000.

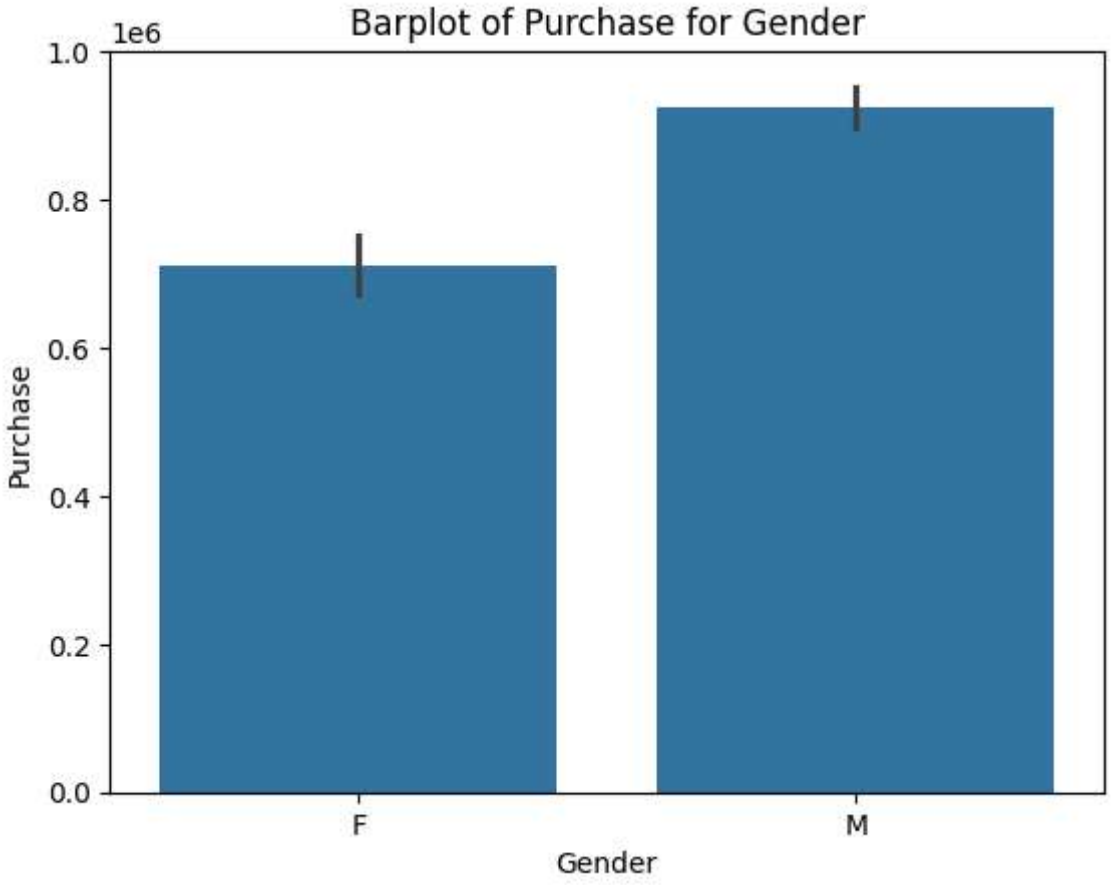
Minimum amount spent by a customer on a product is 12 and maximum is 23961. 50% customes bought the products worth less than 8k. Next 25% bought products worth 8k-12k. Next 25% bought products worth between 8k-24k indicating either outliers or high value customers.

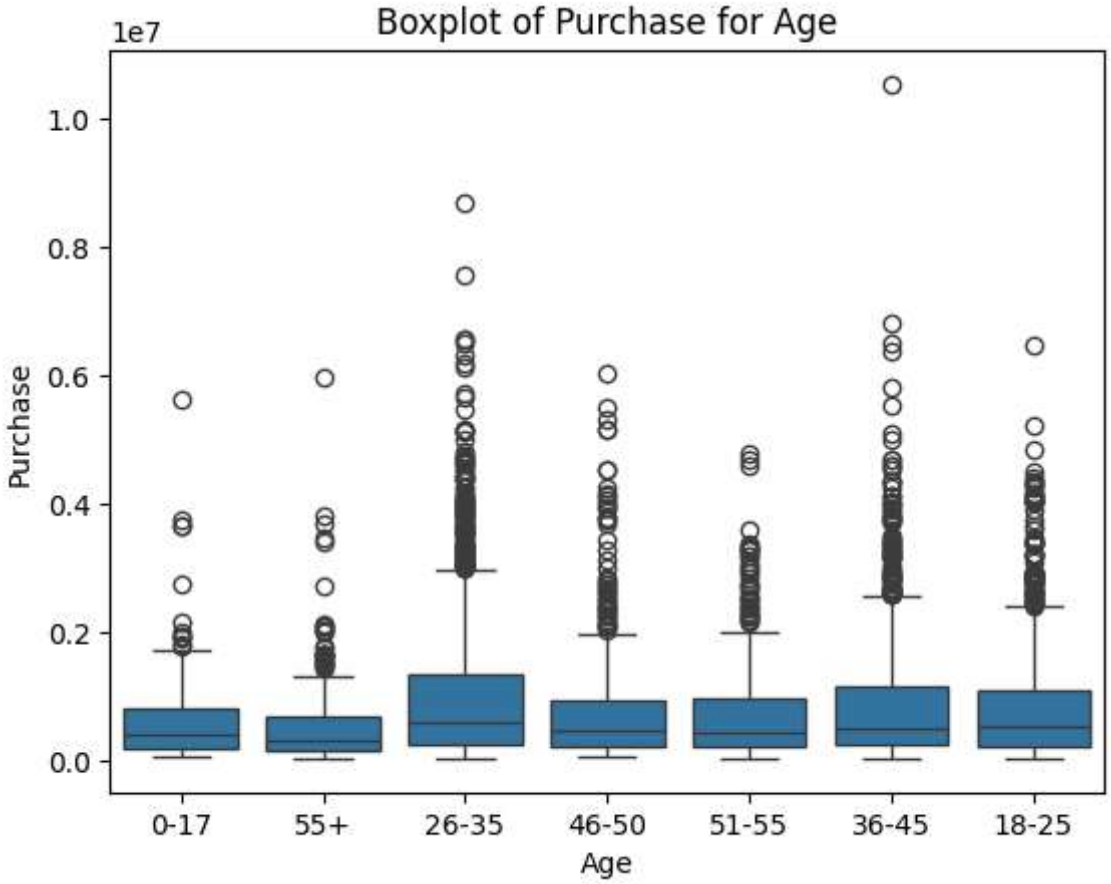
Bivariate analysis

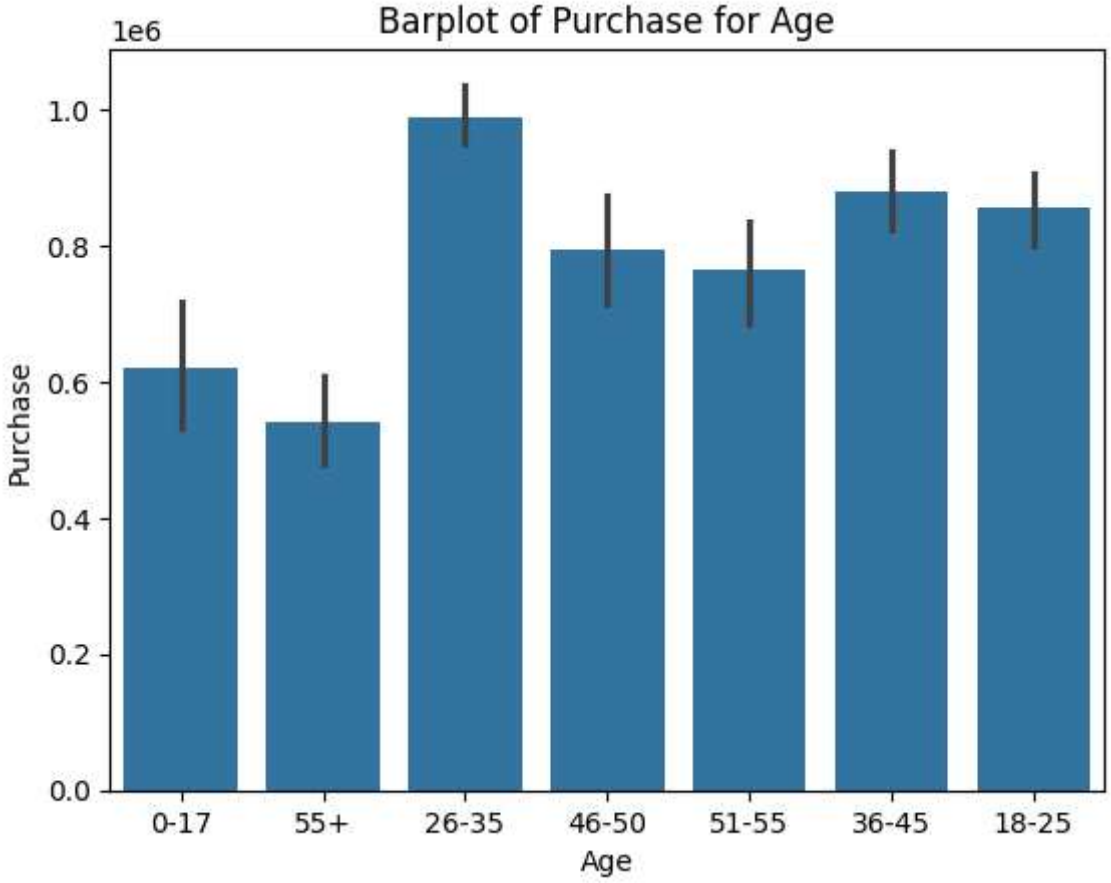
In [ ]:
```
print('User level insights')
for cat_cols in categorical_cols:
  for num_cols in numerical_cols:
    plotter_obj_user_level.bivariateboxplot(categorical_column=cat_cols, numerical_
    plotter_obj_user_level.bivariatebarplot(categorical_column=cat_cols, numerical_
```

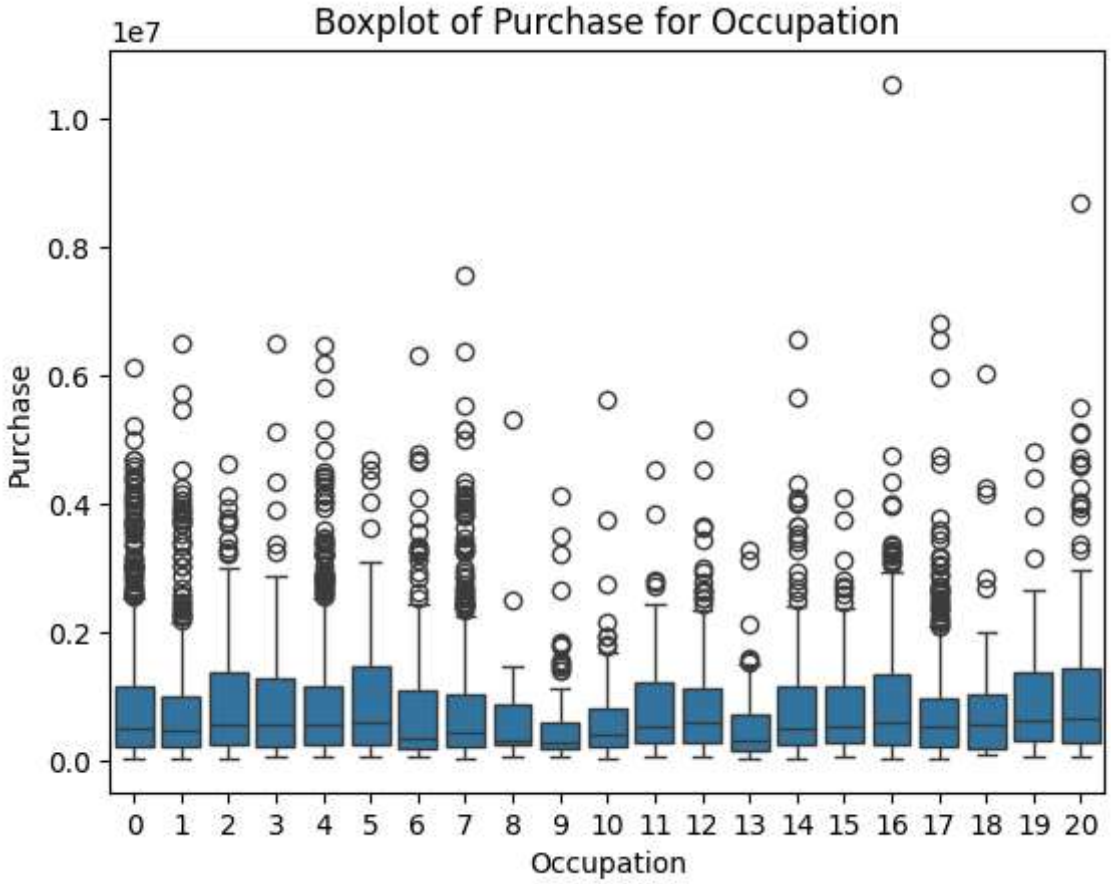
```
User level insights
```

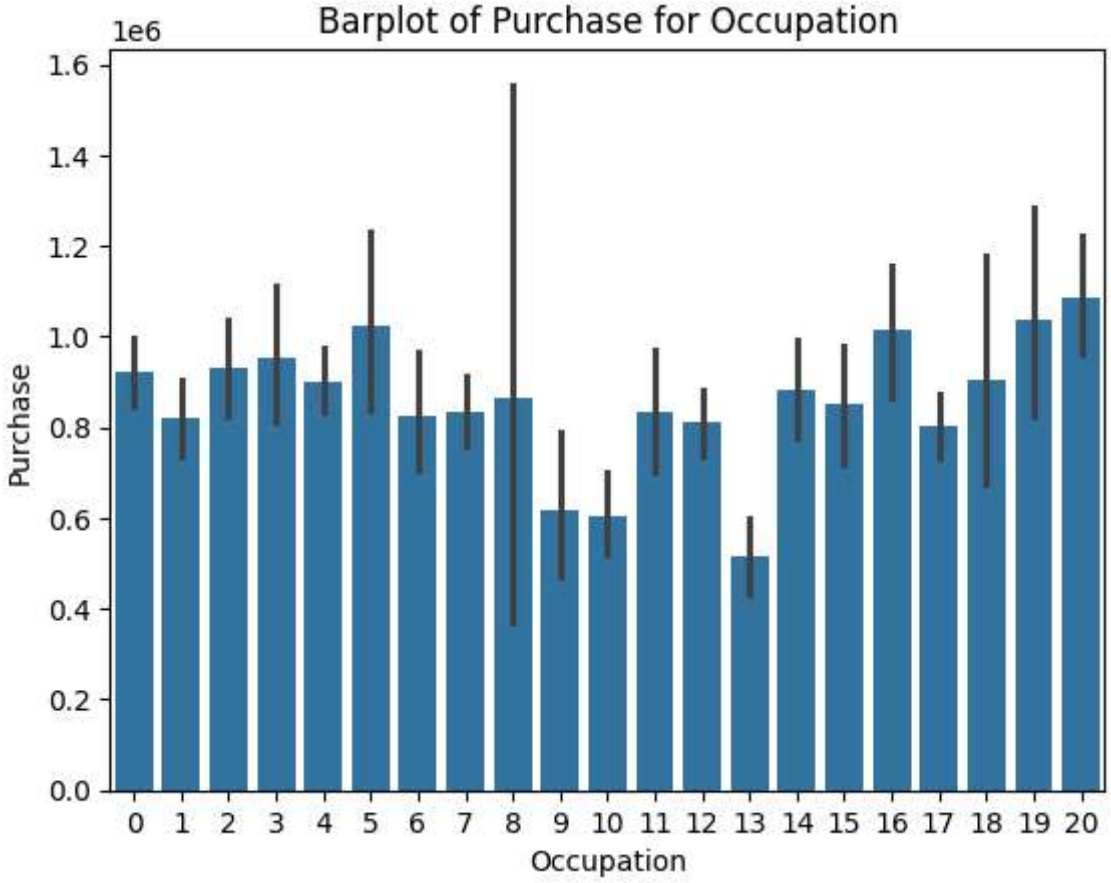

Boxplot of Purchase for Gender
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
F
M
Gender

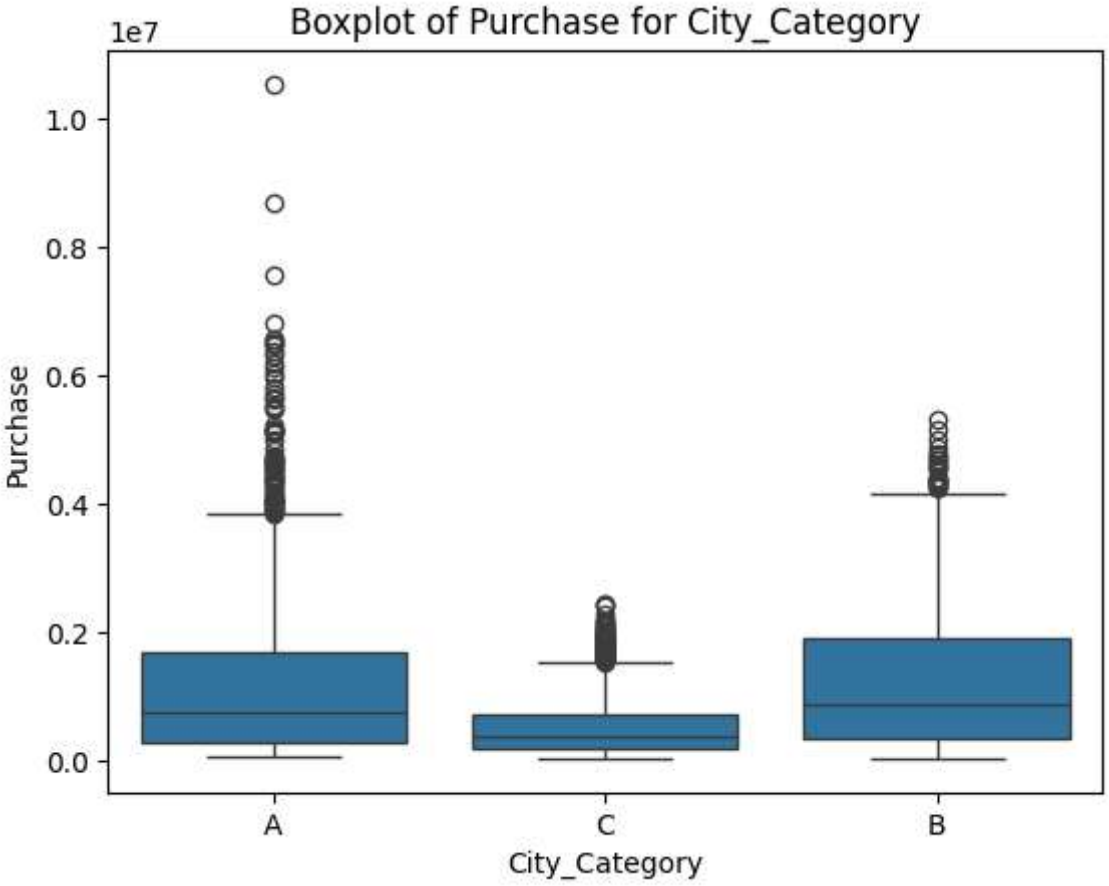

Barplot of Purchase for Gender
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
F
M
Gender

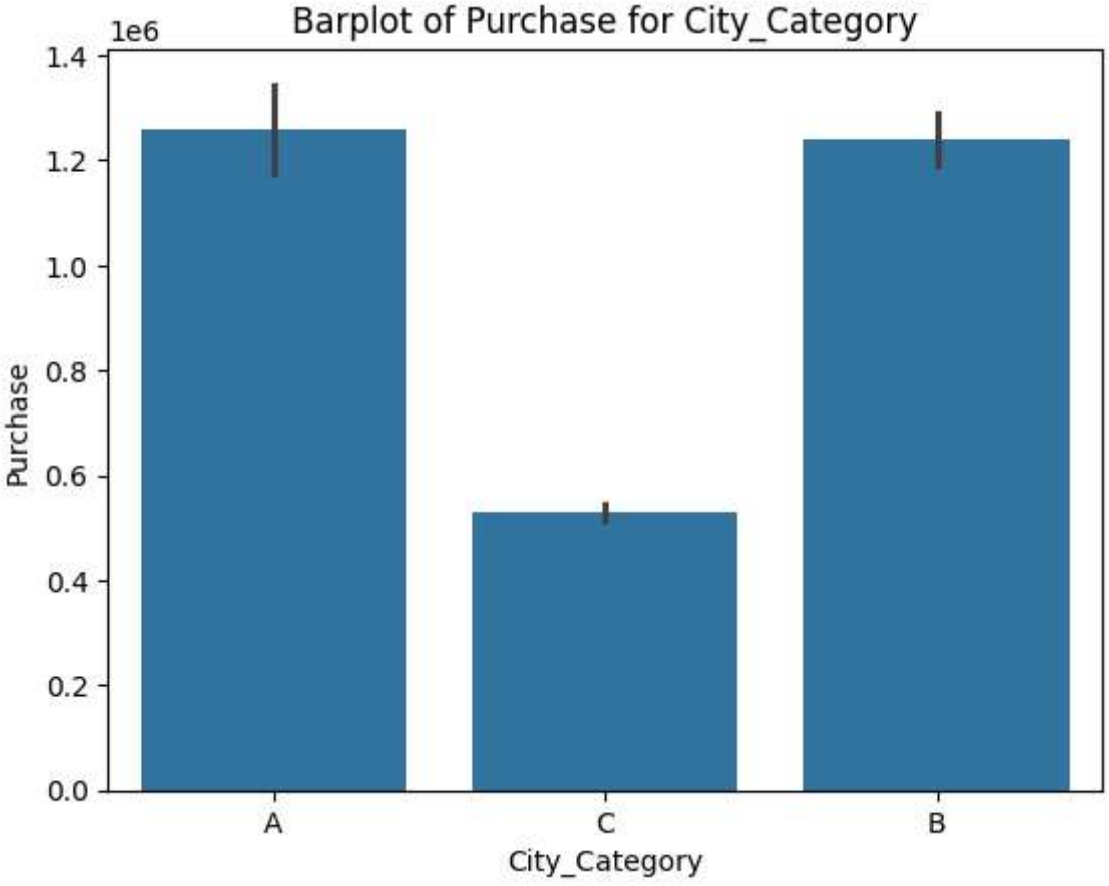

Boxplot of Purchase for Age
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
0-17
55+
26-35
46-50
51-55
36-45
18-25
Age

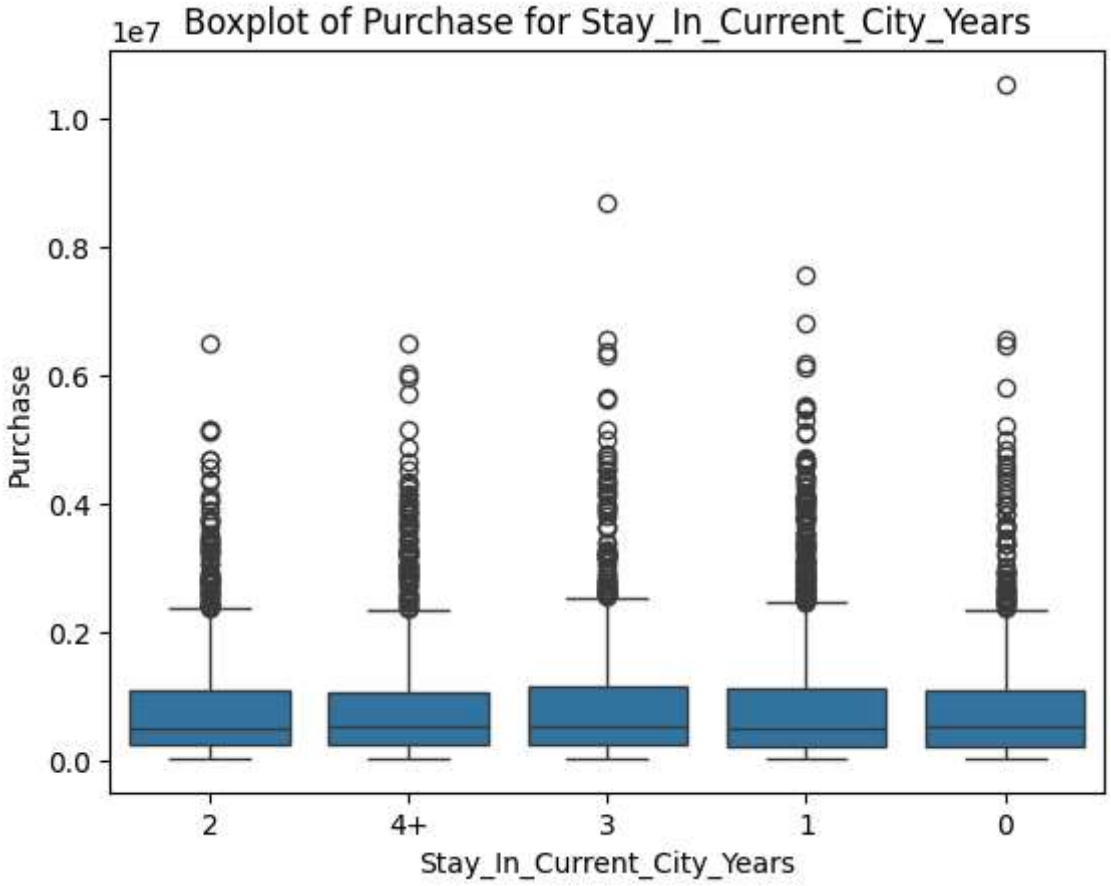

Barplot of Purchase for Age
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
0-17
55+
26-35
46-50
51-55
36-45
18-25
Age

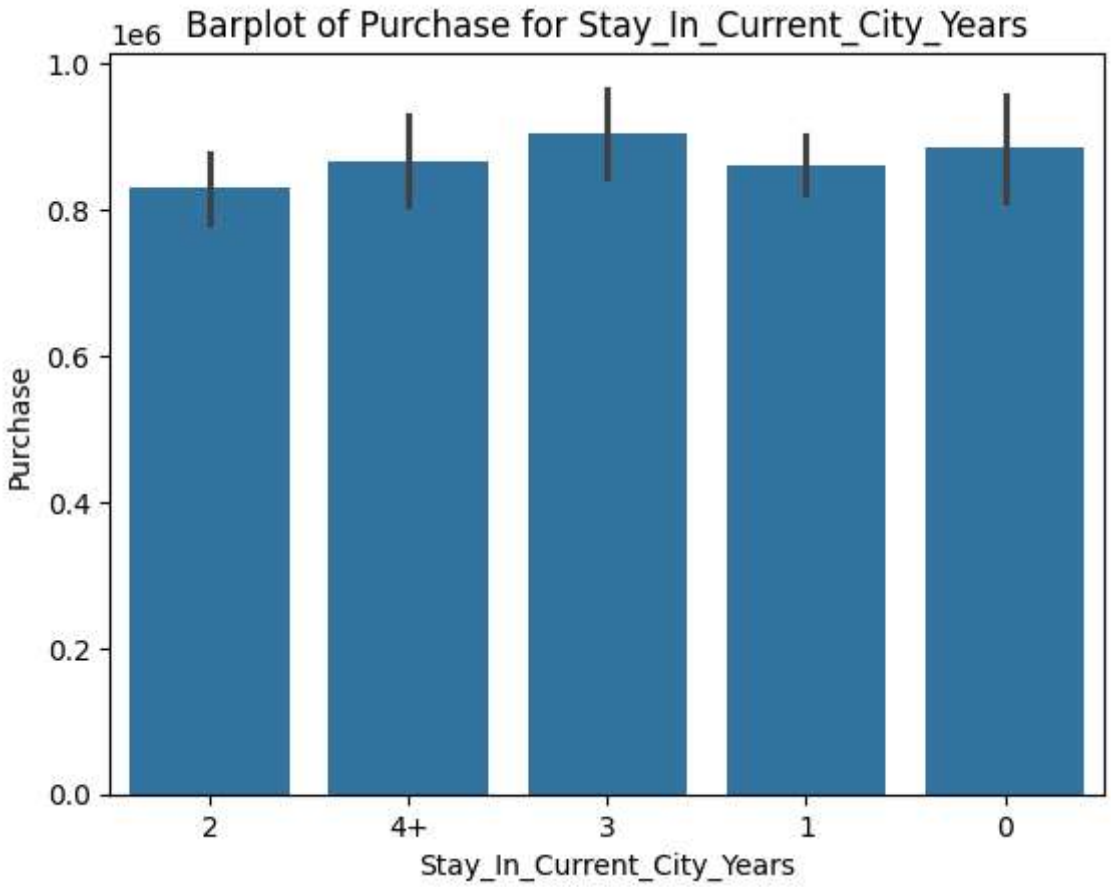

Boxplot of Purchase for Occupation
1e7
Purchase
Occupation
0.0
0.2
0.4
0.6
0.8
1.0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

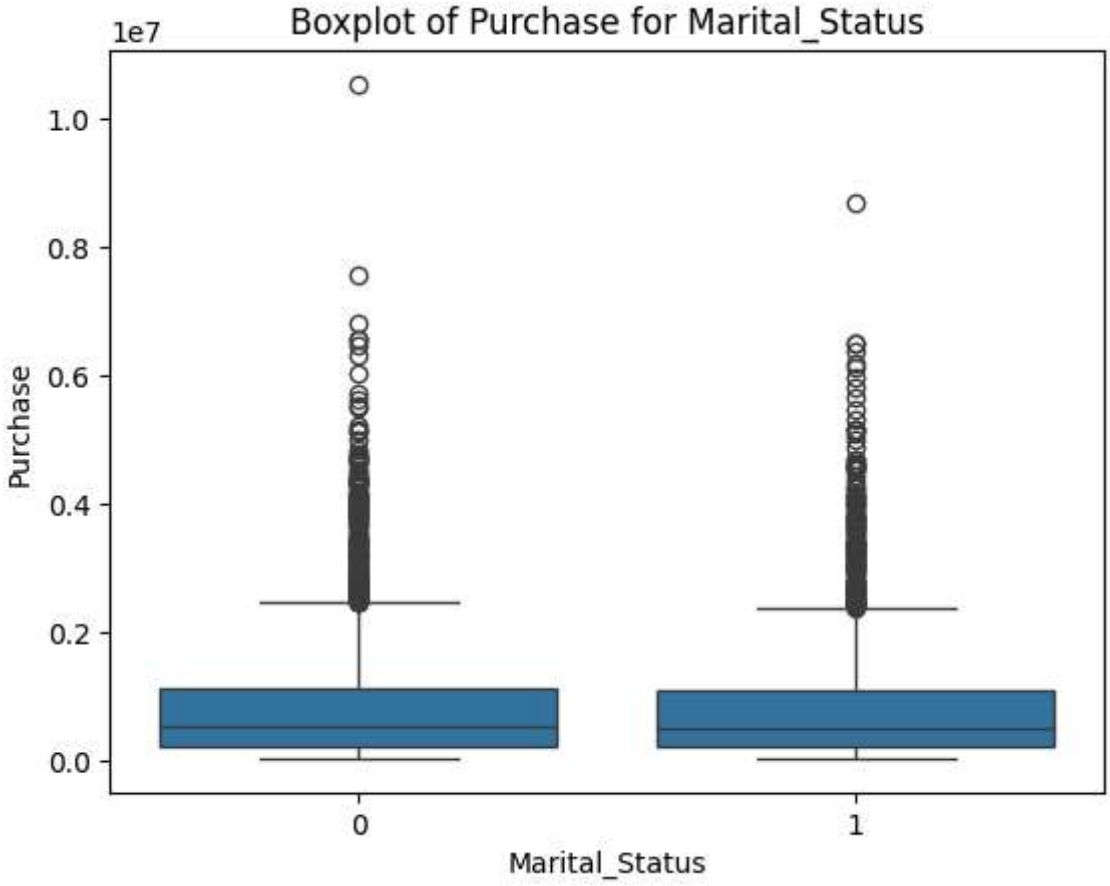

Barplot of Purchase for Occupation
1e6
Purchase
Occupation
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

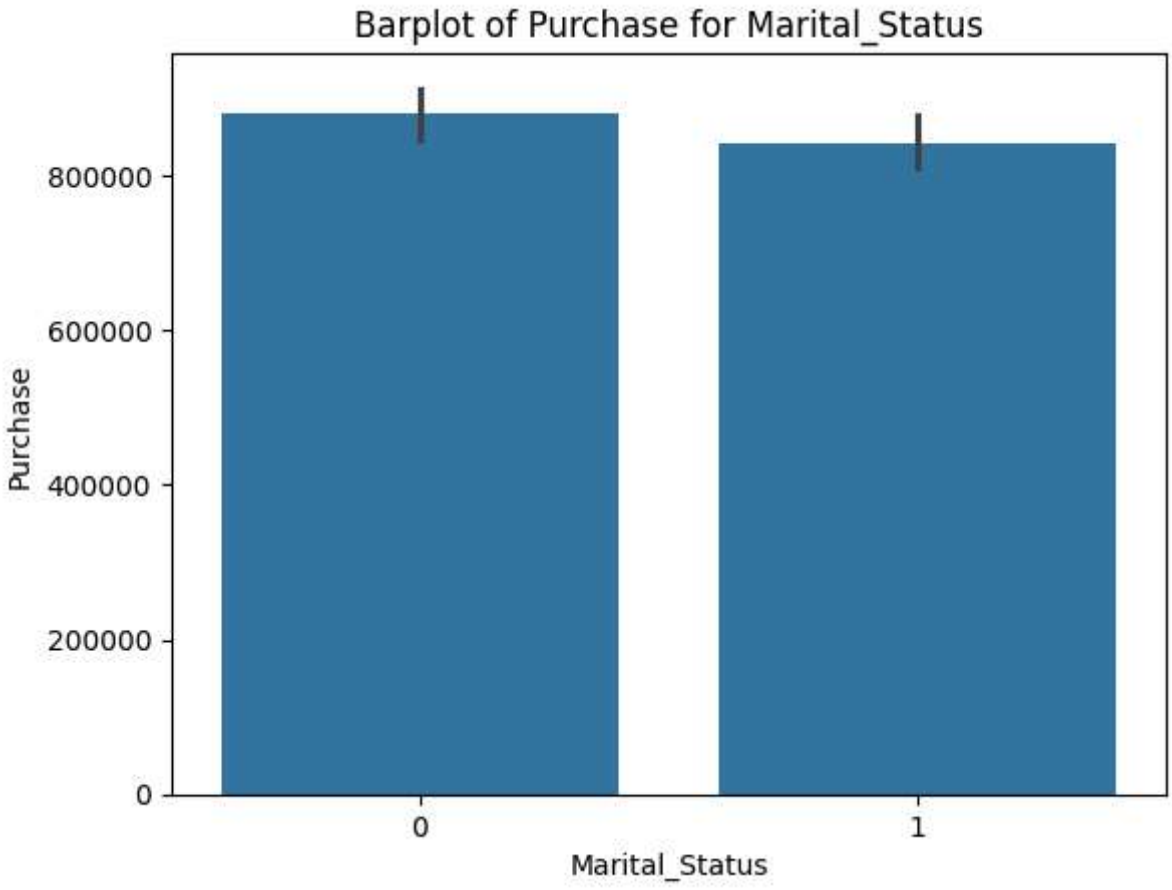

Boxplot of Purchase for City_Category
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
A
C
B
City_Category


Barplot of Purchase for City_Category
1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
A
C
B
City_Category


Boxplot of Purchase for Stay_In_Current_City_Years
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
2
4+
3
1
0
Stay_In_Current_City_Years


Barplot of Purchase for Stay_In_Current_City_Years
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
2
4+
3
1
0
Stay_In_Current_City_Years

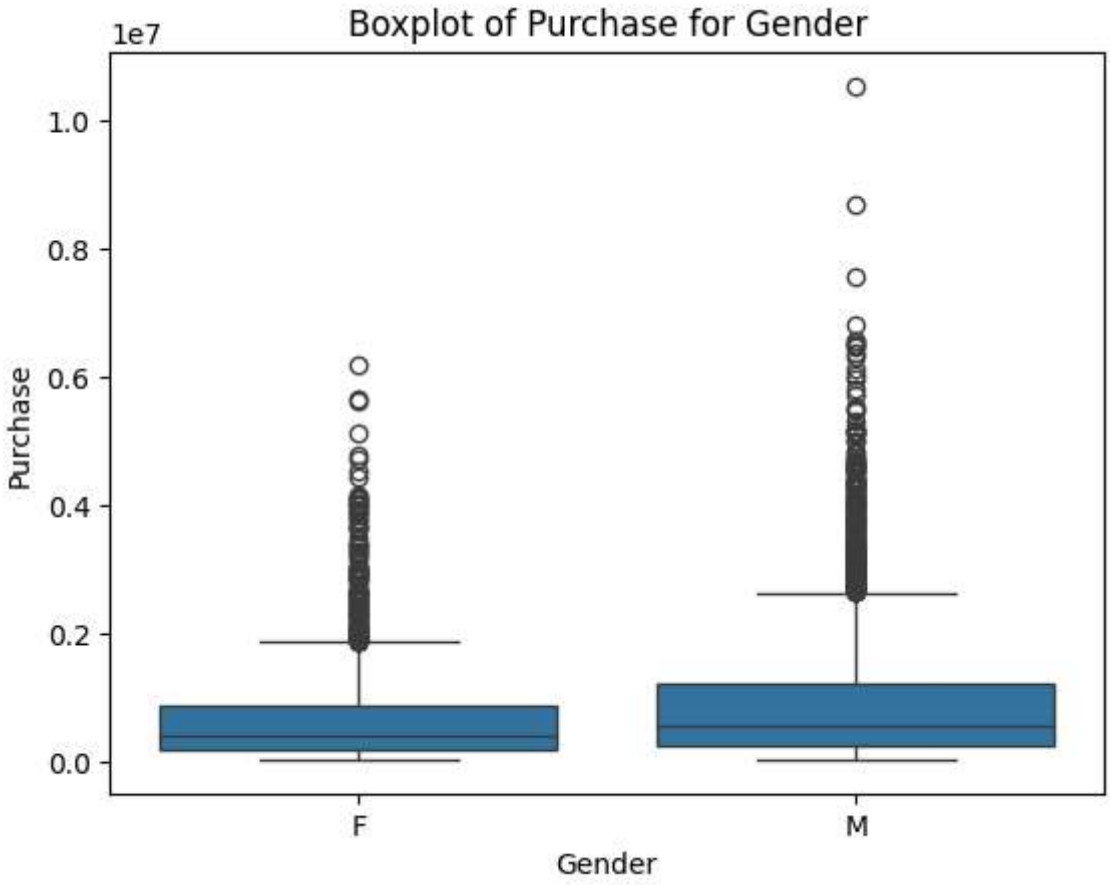

Boxplot of Purchase for Marital_Status
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
0
1
Marital_Status

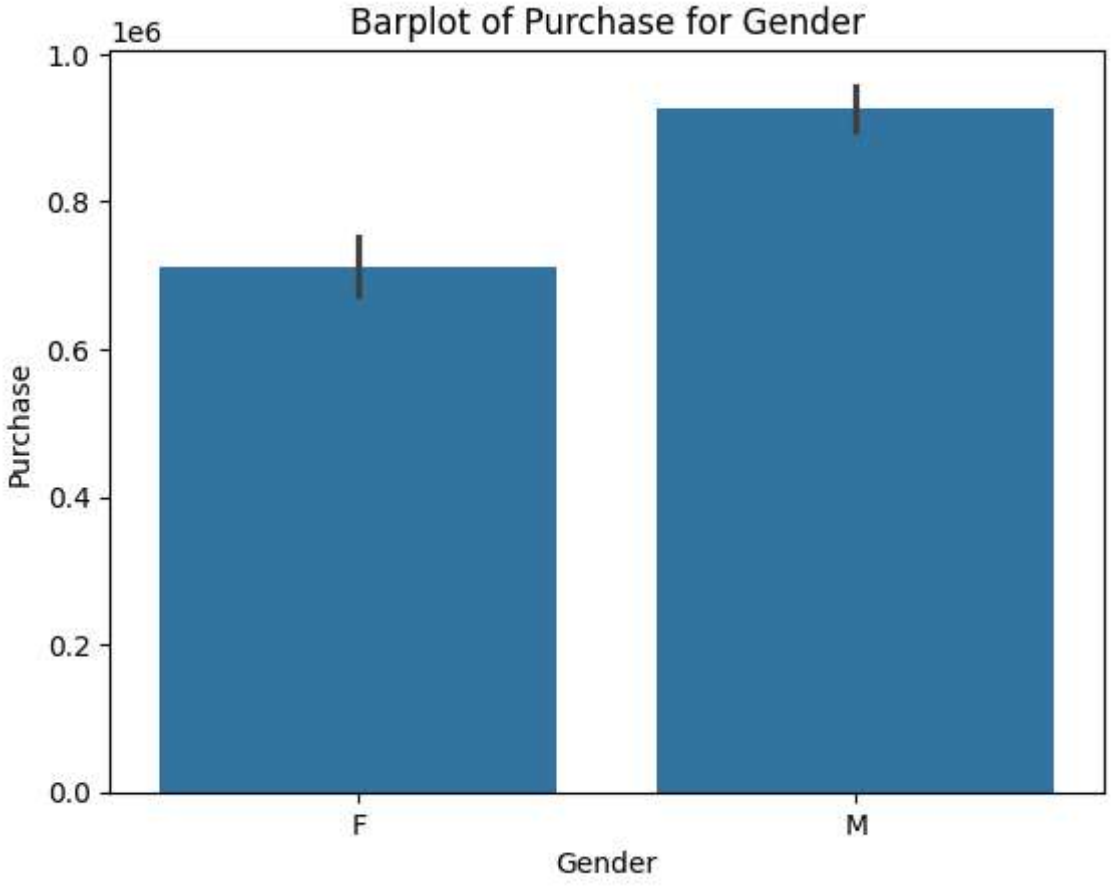

Barplot of Purchase for Marital_Status
800000
600000
400000
200000
0
Purchase
0
1
Marital_Status

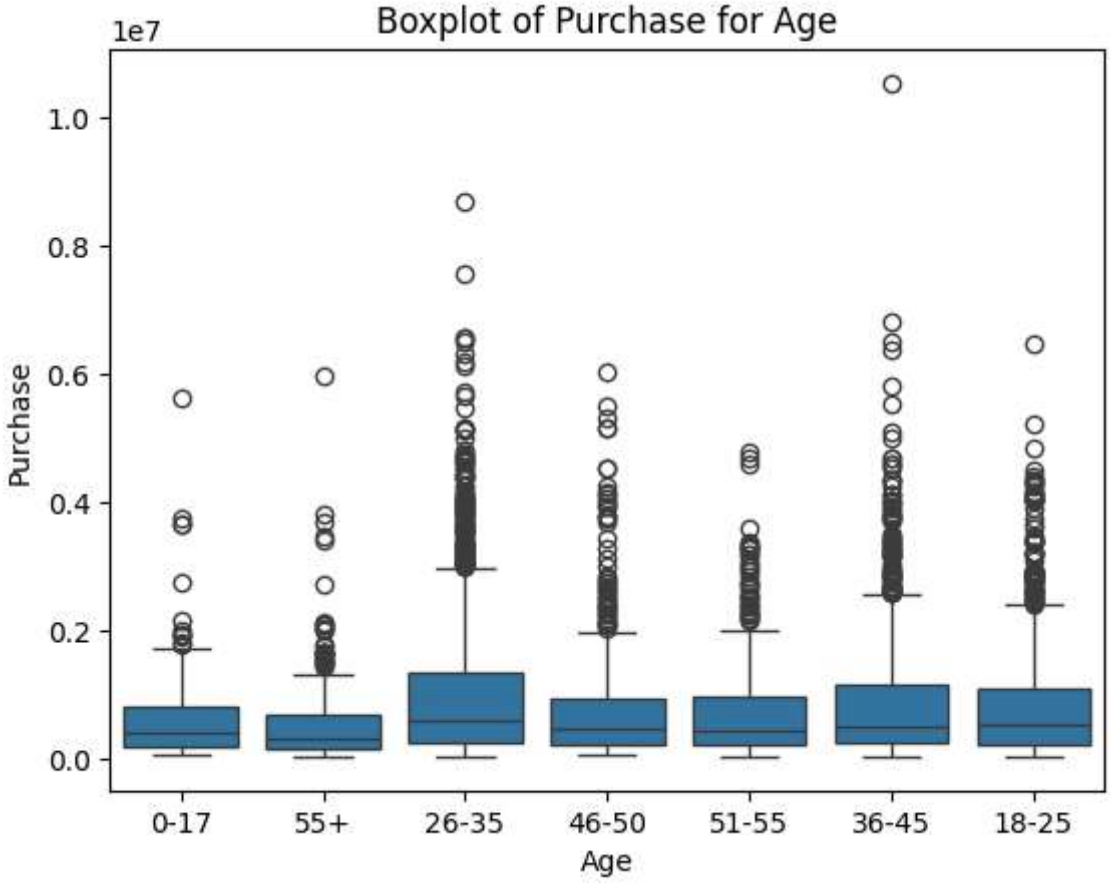
```
Could not interpret value `Product_Category` for `x`. An entry with this name does n
ot appear in `data`.
Could not interpret value `Product_Category` for `x`. An entry with this name does n
ot appear in `data`.
```

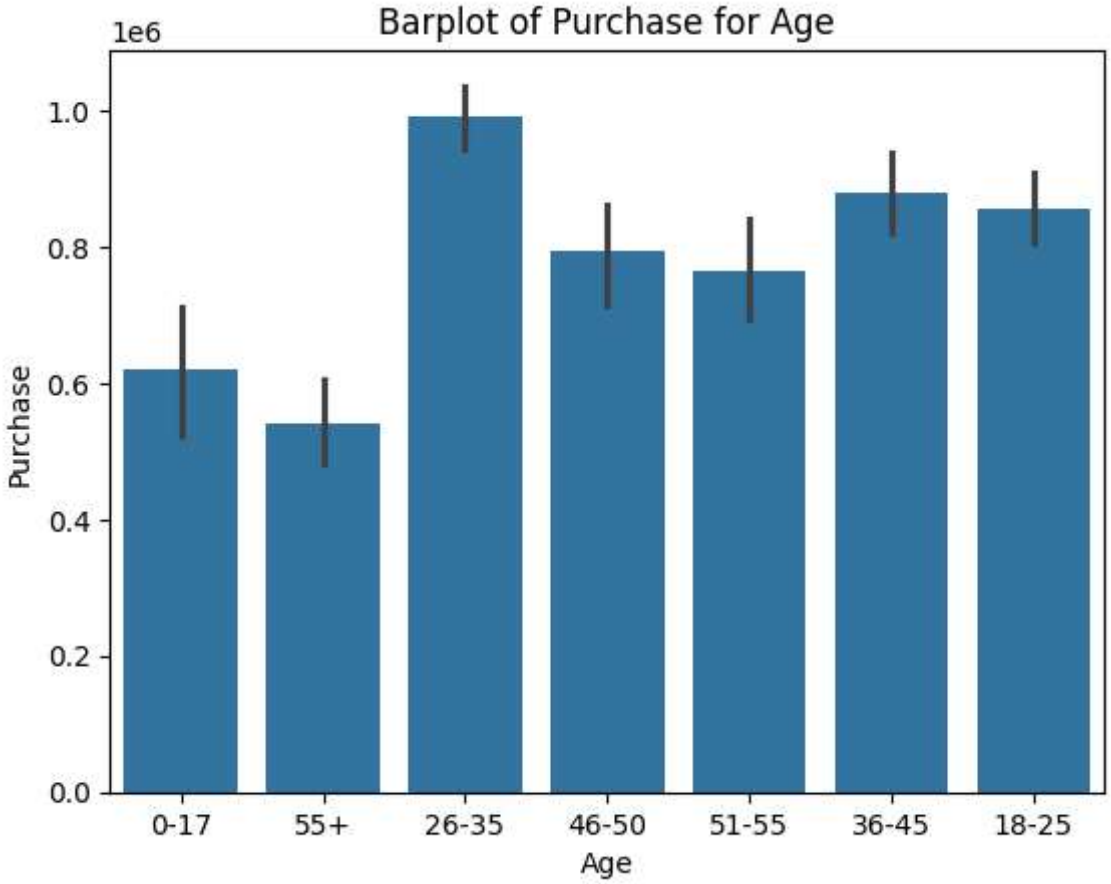
In [ ]:
```
print('Product purchase level insights')
plotter_obj = Plotter(data)
for cat_cols in categorical_cols:
  for num_cols in numerical_cols:
    plotter_obj_user_level.bivariateboxplot(categorical_column=cat_cols, numerical_
    plotter_obj_user_level.bivariatebarplot(categorical_column=cat_cols, numerical_
```

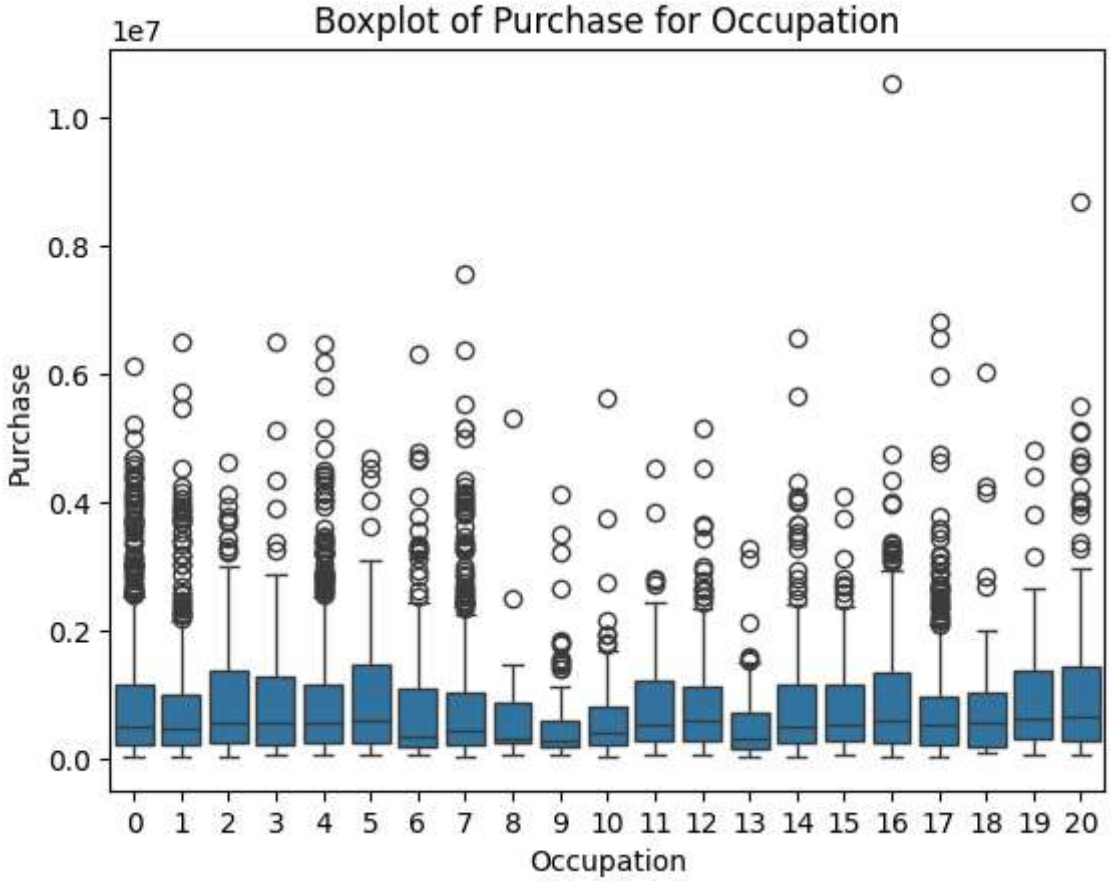
```
Product purchase level insights
```

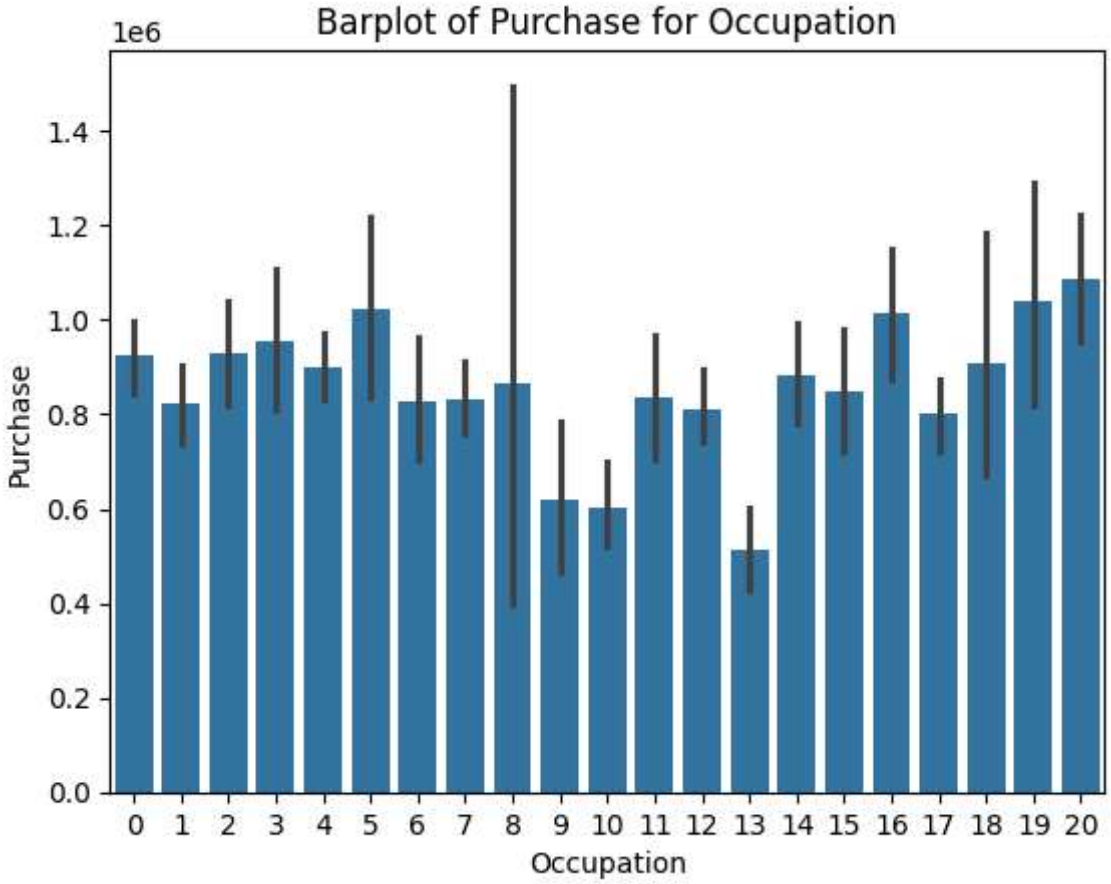


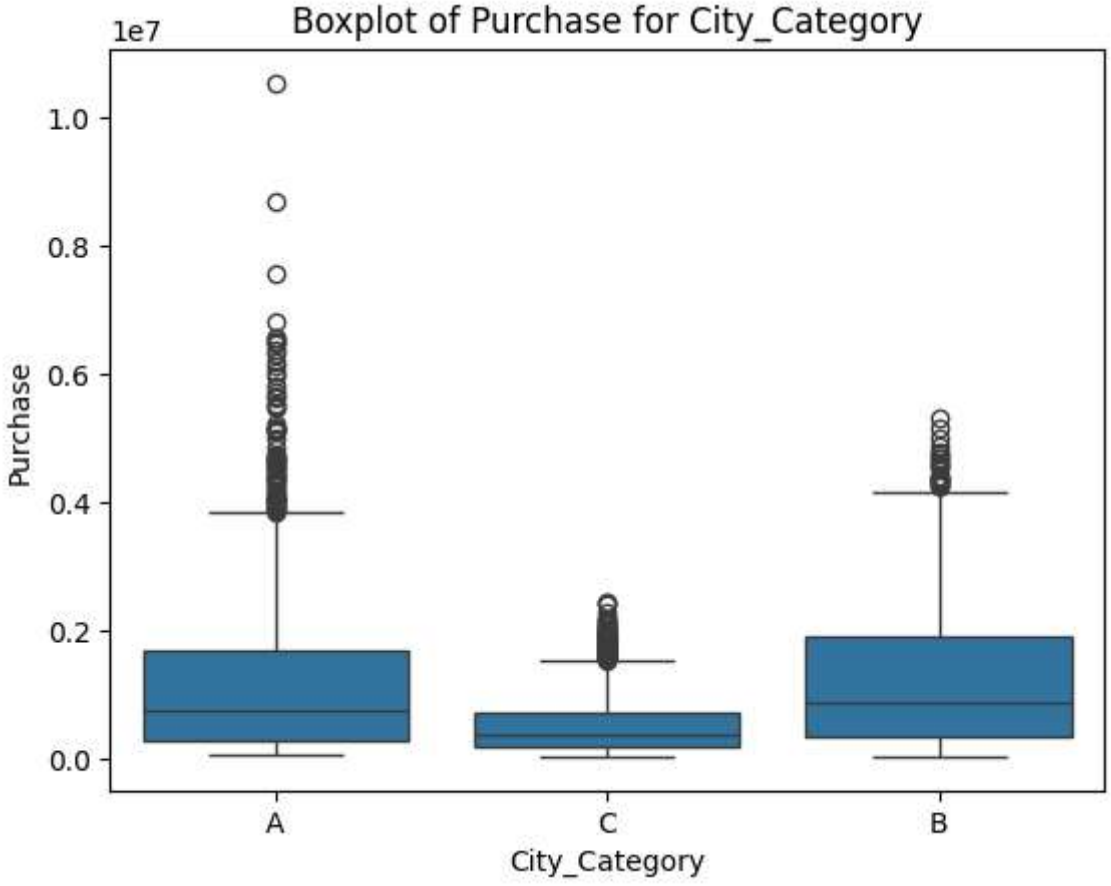

Barplot of Purchase for Gender
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
F
M
Gender

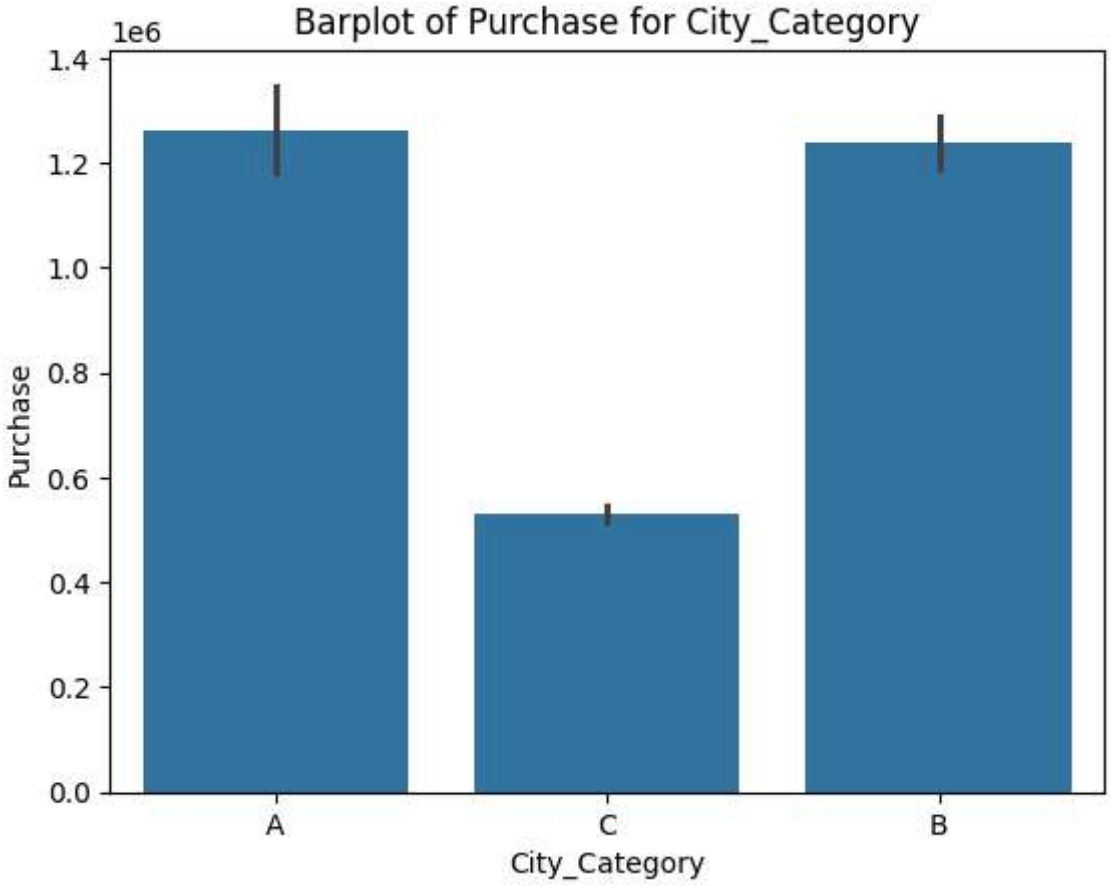

Boxplot of Purchase for Age
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
0-17
55+
26-35
46-50
51-55
36-45
18-25
Age

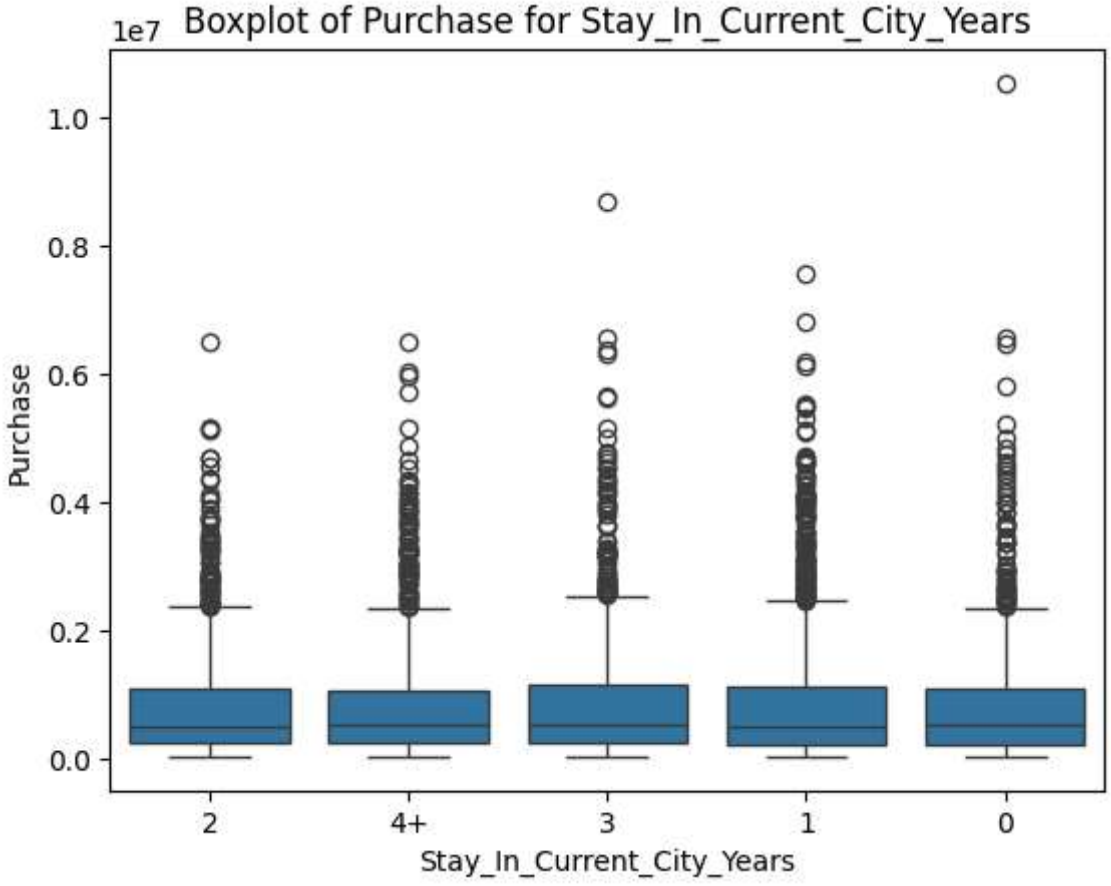

Barplot of Purchase for Age
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
0-17
55+
26-35
46-50
51-55
36-45
18-25
Age

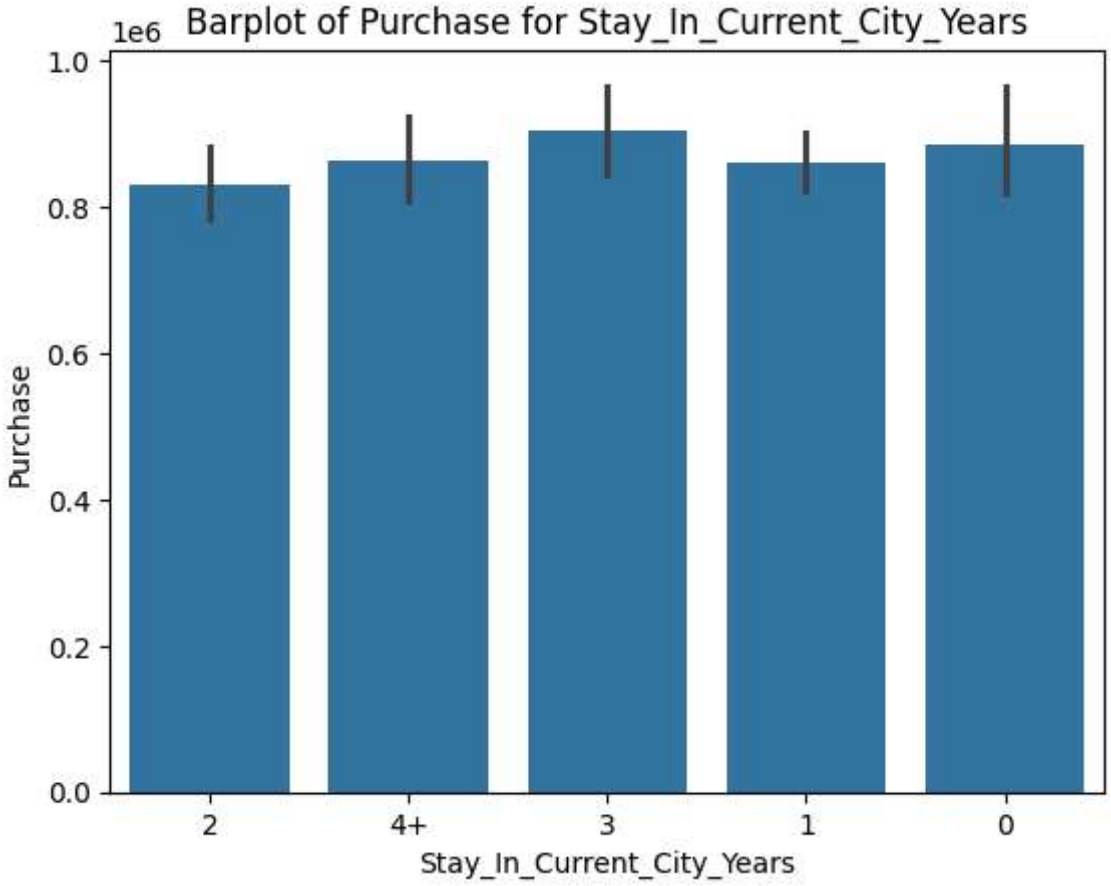

Boxplot of Purchase for Occupation
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Occupation

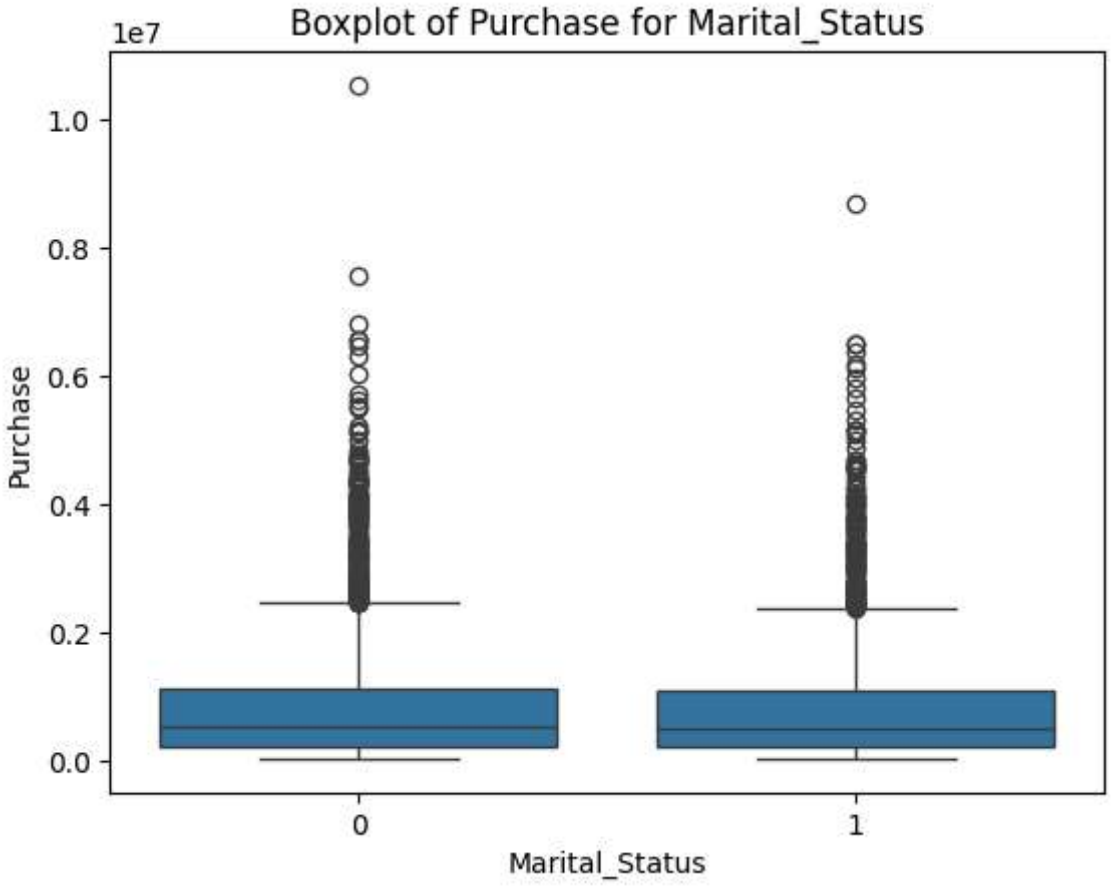

Barplot of Purchase for Occupation
1e6
Purchase
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Occupation

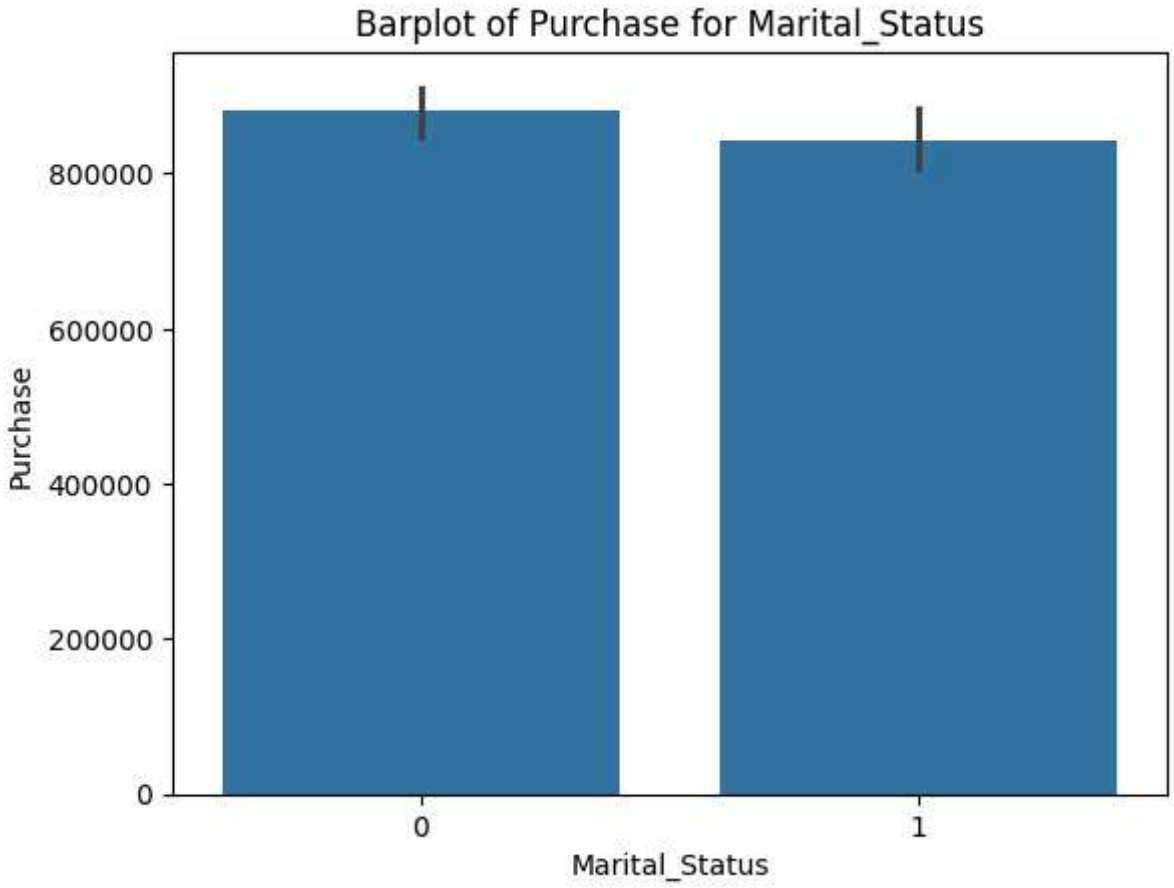

Boxplot of Purchase for City_Category
1e7
Purchase
0.0
0.2
0.4
0.6
0.8
1.0
A
C
B
City_Category


Barplot of Purchase for City_Category
1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
A
C
B
City_Category


Boxplot of Purchase for Stay_In_Current_City_Years
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
2
4+
3
1
0
Stay_In_Current_City_Years


Barplot of Purchase for Stay_In_Current_City_Years
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
2
4+
3
1
0
Stay_In_Current_City_Years


Boxplot of Purchase for Marital_Status
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Purchase
0
1
Marital_Status




```
Could not interpret value `Product_Category` for `x`. An entry with this name does n
ot appear in `data`.
Could not interpret value `Product_Category` for `x`. An entry with this name does n
ot appear in `data`.
```

Insights

Male customers generally spend more and there are more high-value customers among them than Female customers.

Customers in age group 26-35 have generally spent more compared to customers in other age groups. Also some individuals in this age group have made very large purchases, much higher than the typical spending amounts of other age groups. Customers in age group55+ have spent the least money in general.

Customers in city category C tend to spend much smaller amount than that for city A and B. Also there are many more high value customers from city category A(Purchase amount of few customers is much higher) than city B

Confidence interval analysis for mean product purchase value

In [ ]:
```
def get_confidence_interval(df, sample_size, significance_level=5):
  bootstrapped_means= []
  for i in range(10000):
    if sample_size == 'whole data':
      bootstrapped_sample = np.random.choice(df, size=len(df), replace=True)
    else:
      bootstrapped_sample = np.random.choice(df, size=sample_size, replace=True)
    bootstrapped_mean = np.mean(bootstrapped_sample)
    bootstrapped_means.append(bootstrapped_mean)
  return [np.percentile(bootstrapped_means, significance_level/2), np.percentile(bo
```

In [ ]:
```
for sample_sz in ['whole data', 300, 3000, 30000]:
  print(f'Sample size n: {sample_sz}')
  for unique_val in data.Marital_Status.unique():
    data_fltered = data[data.Marital_Status == unique_val]['Purchase'].reset_index(
    print(f'Confidence interval for mean purchase for Marital status {unique_val} :
  print()
```

```
Sample size n: whole data
Confidence interval for mean purchase for Marital status 0 : [9251.429091463395, 928
0.19787701205]
Confidence interval for mean purchase for Marital status 1 : [9243.958470868078, 927
8.315529850846]

Sample size n: 300
Confidence interval for mean purchase for Marital status 0 : [8796.510833333334, 974
1.318833333333]
Confidence interval for mean purchase for Marital status 1 : [8792.9275, 9738.130333
333333]

Sample size n: 3000
Confidence interval for mean purchase for Marital status 0 : [9116.540283333332, 941
7.368683333334]
Confidence interval for mean purchase for Marital status 1 : [9115.1993, 9413.312283
333335]

Sample size n: 30000
Confidence interval for mean purchase for Marital status 0 : [9218.592564999999, 931
4.13701]
Confidence interval for mean purchase for Marital status 1 : [9212.301894999999, 930
8.52191]
```

In [ ]:
```
for sample_sz in ['whole data', 300, 3000, 30000]:
  print(f'Sample size n: {sample_sz}')
  for unique_val in data.Gender.unique():
    data_fltered = data[data.Gender == unique_val]['Purchase'].reset_index(drop = T
    print(f'Confidence interval for mean purchase for Gender {unique_val} : {get_co
  print()
```

```
Sample size n: whole data
Confidence interval for mean purchase for Gender F : [8709.293631865341, 8760.182425
501993]
Confidence interval for mean purchase for Gender M : [9422.028984403478, 9453.275648
68838]

Sample size n: 300
Confidence interval for mean purchase for Gender F : [8204.202666666666, 9275.561583
333332]
Confidence interval for mean purchase for Gender M : [8866.452666666668, 10016.90141
6666668]

Sample size n: 3000
Confidence interval for mean purchase for Gender F : [8563.27845, 8903.309824999998]
Confidence interval for mean purchase for Gender M : [9250.343975, 9620.69440833333
3]

Sample size n: 30000
Confidence interval for mean purchase for Gender F : [8679.870495833335, 8788.51974
5]
Confidence interval for mean purchase for Gender M : [9378.487079166667, 9494.479181
666666]
```

In [ ]:
```
for sample_sz in ['whole data', 300, 3000, 30000]:
  print(f'Sample size n: {sample_sz}')
  for unique_val in data.Age.unique():
    data_fltered = data[data.Age == unique_val]['Purchase'].reset_index(drop = True
    print(f'Confidence interval for mean purchase for Age group {unique_val} : {get
  print()
```

```
Sample size n: whole data
Confidence interval for mean purchase for Age group 0-17 : [8852.699407363263, 9016.
489857303668]
Confidence interval for mean purchase for Age group 55+ : [9269.044332449777, 9403.1
28784179688]
Confidence interval for mean purchase for Age group 26-35 : [9232.486101636254, 927
3.894327191501]
Confidence interval for mean purchase for Age group 46-50 : [9164.088957572045, 925
4.043035710378]
Confidence interval for mean purchase for Age group 51-55 : [9485.03469455339, 9585.
274490272981]
Confidence interval for mean purchase for Age group 36-45 : [9302.807437075619, 936
1.411889731215]
Confidence interval for mean purchase for Age group 18-25 : [9138.546244481236, 920
1.118739965883]

Sample size n: 300
Confidence interval for mean purchase for Age group 0-17 : [8353.819916666667, 9520.
975916666666]
Confidence interval for mean purchase for Age group 55+ : [8775.308916666667, 9908.3
04583333333]
Confidence interval for mean purchase for Age group 26-35 : [8692.338833333333, 982
1.614666666666]
Confidence interval for mean purchase for Age group 46-50 : [8656.338333333333, 976
9.907083333332]
Confidence interval for mean purchase for Age group 51-55 : [8966.584583333333, 1011
6.844666666666]
Confidence interval for mean purchase for Age group 36-45 : [8764.41625, 9902.129166
666666]
Confidence interval for mean purchase for Age group 18-25 : [8605.581916666666, 974
9.736416666667]

Sample size n: 3000
Confidence interval for mean purchase for Age group 0-17 : [8747.831624999999, 9115.
377966666667]
Confidence interval for mean purchase for Age group 55+ : [9157.215274999999, 9514.3
19325]
Confidence interval for mean purchase for Age group 26-35 : [9072.009141666667, 942
9.369583333333]
Confidence interval for mean purchase for Age group 46-50 : [9029.630933333334, 938
7.69485]
Confidence interval for mean purchase for Age group 51-55 : [9355.9664, 9718.5575916
66668]
Confidence interval for mean purchase for Age group 36-45 : [9152.216058333333, 951
2.783108333333]
Confidence interval for mean purchase for Age group 18-25 : [8993.753583333335, 935
0.462041666668]

Sample size n: 30000
Confidence interval for mean purchase for Age group 0-17 : [8877.40948, 8990.904565]
Confidence interval for mean purchase for Age group 55+ : [9280.733926666666, 9393.0
13402499999]
Confidence interval for mean purchase for Age group 26-35 : [9195.571095833333, 930
9.87486]
Confidence interval for mean purchase for Age group 46-50 : [9154.096983333335, 926
4.189638333333]
```

```
Confidence interval for mean purchase for Age group 51-55 : [9479.019416666668, 959
1.708774166667]
Confidence interval for mean purchase for Age group 36-45 : [9274.174181666667, 938
9.049265]
Confidence interval for mean purchase for Age group 18-25 : [9112.794466666666, 922
6.24553]
```

Insights

- Product purchase value behavior per Gender:
  - The 95% confidence interval for mean product purchase value for Males is [9422, 9453] and tat for Females is [8709, 8760]. Since males have a higher average product purchase value, consider promoting higher-value products or premium items to them. While the average product purchase value for females is lower, they might be more value-driven shoppers.
  - Consider stocking male-preferred categories with premium options, while emphasizing practical and budget-friendly options in categories favored by females
- Product purchase value behavior per Marital Status:
  - There is overlap for 95% confidence interval for mean product purchase value for married and single users. Suggesting the marital status does not effect the mean product purchase value.
- Product purchase value behavior per Age group:
  - The high range of confidence interval ([9269.044332449777, 9403.128784179688]) for mean product purchase value suggests a higher degree of variability in the product purchase behavior within the 55+ age group. This broad interval indicates that the purchasing behavior of customers aged 55 and above may be more diverse compared to other age groups, potentially influenced by varying lifestyle needs, income levels, or product preferences.
  - Age groups 26-35 and 36-45 has the least degree of variability in the product purchase value behaviour suggesting users with consistent product purchase value behaviours.
  - Age group 36-45 has consistent and high product value purchase customers. Walmart should prioritize targeting this demographic, focusing on high purchase value products that align with their preferences
  - Age group 51-55(C.I. [9485.03469455339, 9585.274490272981]) shows the highest mean product purchase value among all age segments. Walmart should prioritize targeting this demographic, focusing on high purchase value products that align with their preferences

Recommendations

- Since males have a higher average product purchase value, consider promoting higher-value products or premium items to them. While the average product purchase value for females is lower, they might be more value-driven shoppers.
- Consider stocking male-preferred categories with premium options, while emphasizing practical and budget-friendly options in categories favoured by females.
- Age group 36-45 has consistent and high product value purchase customers. Walmart should prioritize targeting this demographic, focusing on high purchase value products that align with their preferences.
- Age group 51-55 shows the highest mean product purchase value among all age segments. Walmart should prioritize targeting this demographic, focusing on high purchase value products that align with their preferences.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: