Walmart Case Study on Black Fridays

```
- The case study is to help the team at Walmart Inc. to analyze the
customer purchase behaviour against the customer’s gender, marital
status and the age group they belong to.
- Help the team to make better business decisions.
- They want to understand if the spending habits differ between male
and female customers.
- Which Age groups are likely to spend more, and which marital status
category spends more?
- This helps Walmart to take necessary actions to gain more business,
attract more customers and come up with
more such products, which the loyal customers prefer to buy more.
```

- Defining Problem Statement and Analyzing basic metrics

- To help Walmart with this analysis, we will analyze the given CSV file and try to generate meaningful insights and recommendations.
- We will start with creating a dataframe out of the given CSV. Analyze each attribute, their type, if any null value is present, and if yes, how we can find a solution to fill those null values and describe each attribute for statistical analysis.
- Non-graphical analysis
  - For all numerical values, we can describe the field to get the mean, max, min, 25%, 50% and 75% and standard deviation to create IQR for analyzing the outliers.
  - For categorical values, we can get unique value counts for each field and then major the mean, max, and other statistical data for analyzing outliers.
- Graphical Analysis
  - We can analyze the above statistical data by creating frequency graphs for each attribute(pie charts, histplot, countplot). This is for continuous variables.
  - We can draw a box plot or dist plot to analyze the outliers for discrete variables such as categorical variables.
  - To determine the relationship between two or more fields, we can create bivariate plots(box plot, barplot, scatter plots, line plot).
  - To determine the correlation between fields, we can plot heatmaps.
- Once the fundamental analysis is done, we will do some statistical analysis to find the Central Limit Theorem for normally distributed graphs and use it to find the confidence interval to determine the lower and upper limits within which the population average purchase lies.

```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm

# Read the Walmart csv file

walmart_df =
pd.read_csv('/Users/parimitarath/workspace/DSML/case_studies/walmart_c
ase_study/walmart.csv')
```

Define three different classes to create generic functions which can be reused by multiple columns of the dataframe.

```
- Walmart
    - Holds all essential functionalities, such as creating a copy of
the original dataframe.
    - Convert object attributes to categories when an object attribute
has finite choices
    - Giving labels to each age group.
    - Giving categorical labels to integer Marital_status.
    - Non graphical analysis
- WalmartCLT
    - This class holds all functionalities, which is responsible for
    - calculating aggregate functions
    - find the lower and upper confidence interval for 90%, 95% and
99% confidence level
- WalmartPlot
    - Holds all kinds of plots
    - univariate plots
    - bivariate plots
    - multivariate plots
    - plots to determine outliers
    - heatmap
    - special plots to create distribution plot for different age
groups.

# Class definition for all standard methods for different
dataframes/series

class Walmart:
    def __init__(self):
        pass
```

```python
    # create a copy of dataframe to maintain originality
    def copy_df(self, df):
        return df.copy()

    # Convert object/int to categorical variable whenever required
    def to_category(self, df, pattern):
        df[pattern] = df[pattern].astype('category')
        return df

    # When required, convert string to number for Gender and Marital
Status
    def string_to_num_gender(self, gender):
        if gender == 'M':
            return 0
        return 1

    # When required, convert string to number for Gender and Marital
Status
    def string_to_num_maritalStatus(self, status):
        if status == 'Single':
            return 0
        return 1

    # Value counts for each column
    def value_counts(self, df, pattern):
        print(f'Value counts for field {pattern}:\
n{df[pattern].value_counts()}')

    # Unique values for each column
    def nunique(self, df, pattern):
        print(f'Unique values for field {pattern}:
{df[pattern].nunique()}')

    # When required, convert the number to a string for Marital Status
    def maritalStatusLabel(self, num):
        if num == 0:
            return 'Single'
        return 'Partnered'

    # define a label for attributes
    def define_range(self, df, column, bins, labels, string):
        df[string] = pd.cut(df[column],bins=bins,labels=labels)
        return df

    # City Stay Range
    def cityStayRange(self, df):
        # Update cityStay dataframe object with labels
        bins = ['0', "1", "2", "3", "4", "5"]
        labels = ['<1','1-2','2-3','3-4','>4']
        column = 'Stay_In_Current_City_Years'
```

```
        string = 'Stay_In_Current_City_Range'
        df = define_range(df, column, bins, labels, string)

        return df

    # Give a label to each age category

    # 0   0-17
    # 1  18-25
    # 2  26-35
    # 3  36-45
    # 4  46-50
    # 5  51-55
    # 6    55+

    def age_to_label(self, values):
        if values == '0-17':
            return 'Teen'
        elif values == '18-25':
            return 'Young Adults'
        elif values == '26-35':
            return 'Mid Adults'
        elif values == '36-45':
            return 'Adults'
        elif values == '46-50':
            return 'Mid-Age'
        elif values == '51-55':
            return 'Late Mid-Age'
        else:
            return 'Senior Citizens'

    # Create cross tab between two fields
    def cross_tab_2fields(self, df, col1, col2):
        crossTab = pd.crosstab(df[col1], df[col2], margins = True)
        print(crossTab)

    # Create cross tab between more than two fields
    def cross_tab_multiFields(self, df, col1, col2, agg_col, aggFunc):
        crossTab = pd.crosstab(df[col1], df[col2],
                               values = df[agg_col], aggfunc =
aggFunc, margins = True)
        print(crossTab)

    # univariate non-graphical presentation
    def univariate_nongraphical(self, df, pattern, aggFunc):
        grouped_df = df.groupby(pattern)[[pattern]].aggregate(
        total = (pattern, aggFunc)
        )
        grouped_df.reset_index(inplace = True)
        return grouped_df
```

```python
    # Outlier Detection for individual fields against purchase
    def outlier_detection_non_graph(self, df, pattern1, pattern2,
*args):
        # non-graphical representation using groupby method
        df_groupby = df.groupby(pattern1)[[pattern2]].aggregate(
            Mean = (pattern2, args[0]),
            Median = (pattern2, args[1])
        )
        df_groupby.reset_index(inplace = True)
        return df_groupby


# A class definition for all aggregate functions, CLT and CI
class WalMartCLT:
    def __init__(self):
        pass

    def aggregate_function(self, df, agg_col, aggr, columns = []):
        df_copy = df.groupby(columns).agg(
            {agg_col: aggr}
        ).reset_index()
        return df_copy

    def attribute_df(self, df, column, pattern):
        col_df = df.loc[df[column] == pattern]
        return col_df

    # Analyze the data with different sample size
    def sample_size_mean(self, df, size, agg_col, sample_range):

        sample_mean = [df.sample(size, replace = True)[agg_col].mean()
for i in range(sample_range)]
        return sample_mean

    def sample_mean_std(self, df1, df2, pattern1, pattern2):

        # Find the mean and standard deviation of the sample mean for
each category of an attribute
        sample1_mean = np.mean(df1).round(4)
        sample2_mean = np.mean(df2).round(4)

        sample1_std = np.std(df1).round(4)
        sample2_std = np.std(df2).round(4)

        print(f'Sample mean for {pattern1} is {sample1_mean}')
        print(f'Sample Standard Deviation For {pattern1} is
{sample1_std}')

        print(f'Sample mean for {pattern2} is {sample2_mean}')
```

```
        print(f'Sample Standard Deviation For {pattern2} is
{sample2_std}')


    # Calculate Confidence interval
    def find_CI(self, mean, std, sample_count, prob):
        range = []
        std_err = std / np.sqrt(sample_count)
        slice = (1 - (prob / 100)) / 2 # Get the area out side of 95%
probability on both side to get the lower and upper mean
        z1 = norm.ppf(slice)
        z2 = norm.ppf(1 - slice)
        lower_range = z1 * std_err + mean
        higher_range = z2 * std_err + mean
        range.extend([lower_range, higher_range])
        return range

    # Confidence Level 90
    def CI_90(self, df1, df2, agg_col, value1, value2):

        confidence_level = 90

        # Length of Male and Female sample
        df1_length = len(df1)
        df2_length = len(df2)

        # Mean for each sample

        df1_mean = df1[agg_col].mean()
        df2_mean = df2[agg_col].mean()

        # Standard Error for each sample

        df1_stdErr = df1[agg_col].std() / np.sqrt(df1_length)
        df2_stdErr = df2[agg_col].std() / np.sqrt(df2_length)


        df1_CI = self.find_CI(df1_mean, df1_stdErr, df1_length,
confidence_level)
        df2_CI = self.find_CI(df2_mean, df2_stdErr, df2_length,
confidence_level)

        print(f'With 90% confidence level {value1} purchase Confidence
level lies between {df1_CI}')
        print(f'With 90% confidence level {value2} purchase Confidence
level lies between {df2_CI}')

    # Confidence Level 95
    def CI_95(self, df1, df2, agg_col, value1, value2):
```

```
        confidence_level = 95

        # Length of Male and Female sample
        df1_length = len(df1)
        df2_length = len(df2)

        # Mean for each sample

        df1_mean = df1[agg_col].mean()
        df2_mean = df2[agg_col].mean()

        # Standard Error for each sample

        df1_stdErr = df1[agg_col].std() / np.sqrt(df1_length)
        df2_stdErr = df2[agg_col].std() / np.sqrt(df2_length)


        df1_CI = self.find_CI(df1_mean, df1_stdErr, df1_length,
confidence_level)
        df2_CI = self.find_CI(df2_mean, df2_stdErr, df2_length,
confidence_level)

        print(f'With 95% confidence level {value1} purchase Confidence
level lies between {df1_CI}')
        print(f'With 95% confidence level {value2} purchase Confidence
level lies between {df2_CI}')

    # Confidence Level 99
    def CI_99(self, df1, df2, agg_col, value1, value2):

        confidence_level = 99

        # Length of Male and Female sample
        df1_length = len(df1)
        df2_length = len(df2)

        # Mean for each sample

        df1_mean = df1[agg_col].mean()
        df2_mean = df2[agg_col].mean()

        # Standard Error for each sample

        df1_stdErr = df1[agg_col].std() / np.sqrt(df1_length)
        df2_stdErr = df2[agg_col].std() / np.sqrt(df2_length)


        df1_CI = self.find_CI(df1_mean, df1_stdErr, df1_length,
confidence_level)
```

```
        df2_CI = self.find_CI(df2_mean, df2_stdErr, df2_length,
confidence_level)

        print(f'With 99% confidence level {value1} purchase Confidence
interval lies between {df1_CI}')
        print(f'With 99% confidence level {value2} purchase Confidence
interval lies between {df2_CI}')

    # Calculate Central Limit Theorem
    def CLT(self, df):
        pass

# A class definition for all different plots

class WalmartPlots:

    def __init__(self):
        pass

    def distribution(self, df, column, pattern, agg_col, type, bins,
color, saveAs = '', rotate = False):
        new_df = df.loc[df[column] == pattern, [agg_col]]
        self.univariate_plot(new_df, agg_col, type, bins, color,
saveAs, rotate)

    # Outlier detection
    def graphical_outlier_detection(self, df, pattern1, pattern2,
type, saveAs= ''):

        # Graphical outlier distribution
        plt.figure(figsize = (8, 3))
        if type == 'violin':
            sns.violinplot(x = pattern1, y = pattern2, data = df)
        elif type == 'line':
            sns.lineplot(x = pattern1, y = pattern2, data = df)
        elif type == 'bar':
            plt.barh(df[pattern1], df[pattern2], color = ['indigo'])
        else:
            sns.boxplot(x = df[pattern1], y = df[pattern2])

        plt.title(saveAs)
        plt.savefig(saveAs)

    # Univariate plots for frequency distribution
    def univariate_plot(self, df, pattern, type = 'hist', bins = 0,
color = '', saveAs = '', rotate = False):

        plt.figure(figsize = (6, 3))
        if type == 'count':
            sns.countplot(x = df[pattern], color = color)
```

```
        elif type == 'pie':
            plt.pie(df['total'], labels = df[pattern],
                    colors =  sns.color_palette('vlag'),
autopct='%1.1f%%')
        else:
            if rotate:
                sns.histplot(df[pattern], bins = bins, color = color,
kde = True)
                plt.xticks(rotation = 90)
            else:
                sns.histplot(df[pattern], bins = bins, color = color,
kde = True)

        plt.title(saveAs)
        plt.savefig(saveAs)

    # Bivariate plots for frequency distribution
    def bivariate_plot(self, df, col1, col2, type = 'bar', color = '',
saveAs = ''):

        plt.figure(figsize = (6, 3))
        if type == 'line':
            sns.lineplot(x = col1, y = col2, data = df, color = color)
        elif type == 'scatter':
            sns.scatterplot(x = col1, y = col2, data = df, color =
color)
        else:
            sns.barplot(x = col1, y = col2, data = df, color = color)

        plt.title(saveAs)
        plt.savefig(saveAs)

    # Multivariate plots
    def multivariate_plots(self, df, pattern1, pattern2, hue_pattern,
type, saveAs= ''):
        plt.figure(figsize = (6, 3))
        if type == 'violin':
            sns.violinplot(x = pattern1, y = pattern2, data = df, hue
= hue_pattern)
        elif type == 'line':
            sns.lineplot(x = pattern1, y = pattern2, data = df, hue =
hue_pattern)
        elif type == 'bar':
            sns.barplot(x = df[pattern1], y = df[pattern2], data = df,
hue = hue_pattern)
        elif type == 'scatter':
            sns.scatterplot(x = df[pattern1], y = df[pattern2], data =
df, hue = hue_pattern)
        else:
```

```
            sns.boxplot(x = df[pattern1], y = df[pattern2], data = df,
hue = hue_pattern)

        plt.title(saveAs)
        plt.savefig(saveAs)

    # HeatMap for Important Attributes
    def heatMap(self, df, pattern, clrMap, saveAs):
        df_dummy = pd.get_dummies(df, columns=[pattern])
        plt.figure(figsize = (10, 4))
        sns.heatmap(df_dummy.corr(numeric_only = True), annot=True,
cmap = clrMap)

        plt.title(saveAs)
        plt.savefig(saveAs)

    # Gaussian distribution graph for individual group of an attribute
    def gaussian_distribution_2var(self, means1, means2, bins, color,
title1, title2, saveAs):

        plt.figure(figsize = (25, 6))
        plt.subplot(1, 3, 1)
        sns.histplot(means1, bins=bins, color = color)
        plt.title(title1)

        plt.subplot(1, 3, 3)
        sns.histplot(means2, bins=bins, color = color)
        plt.title(title2)
        plt.savefig(saveAs)

    # A special case for each age group. This can be done using a
single-function
    # But to accommodate all the subplots in a single matrix. The
function is created.
    def gaussian_distribution_multiVar(self, mean1, mean2, mean3,
mean4, mean5,
                                       mean6, mean7, bins, color,
title1, title2,
                                       title3, title4, title5, title6,
title7, saveAs):

        plt.figure(figsize = (25, 12))
        plt.subplot(3, 3, 1)
        sns.histplot(mean1, bins=bins, color = color)
        plt.title(title1)

        plt.subplot(3, 3, 2)
        sns.histplot(mean2, bins=bins, color = color)
        plt.title(title2)
```

```
        plt.subplot(3, 3, 3)
        sns.histplot(mean3, bins=bins, color = color)
        plt.title(title3)

        plt.subplot(3, 3, 4)
        sns.histplot(mean4, bins=bins, color = color)
        plt.title(title4)

        plt.subplot(3, 3, 6)
        sns.histplot(mean5, bins=bins, color = color)
        plt.title(title5)

        plt.subplot(3, 3, 7)
        sns.histplot(mean6, bins=bins, color = color)
        plt.title(title6)

        plt.subplot(3, 3, 9)
        sns.histplot(mean7, bins=bins, color = color)
        plt.title(title7)
        plt.savefig(saveAs)

# Object creation for each class which can be used for all use cases
wmt = Walmart()
wmtCLT = WalMartCLT()
wmtPlot = WalmartPlots()
```

Observations on shape of data, data types of all the attributes,

- conversion of categorical attributes to 'category' (If required), statistical summary
- Non-Graphical Analysis: Value counts and unique attributes

```
# Print the first five rows of Walmart dataframe
# Print the information, related to variables and their types from
Walmart dataframe
# Describe the dataframe
# Detect null values

walmart_copy = wmt.copy_df(walmart_df)

# print first five rows
print(walmart_copy.head())

# print shape of the dataframe
print(walmart_df.shape)

# print info
print(walmart_copy.info())
```

```
# Describe all numeric variables
print(walmart_copy.describe())

# Detect null values
print(walmart_df.isnull().sum())

   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
(550068, 10)
<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
None
            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                       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
```

```
# Value counts for each fields
wmt.value_counts(walmart_copy, 'User_ID')
wmt.value_counts(walmart_copy, 'Product_ID')
wmt.value_counts(walmart_copy, 'Gender')
wmt.value_counts(walmart_copy, 'Age')
wmt.value_counts(walmart_copy, 'Occupation')
wmt.value_counts(walmart_copy, 'City_Category')
wmt.value_counts(walmart_copy, 'Stay_In_Current_City_Years')
wmt.value_counts(walmart_copy, 'Marital_Status')
wmt.value_counts(walmart_copy, 'Product_Category')
```

```
Value counts for field User_ID:
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
Value counts for field Product_ID:
Product_ID
P00265242    1880
P00025442    1615
P00110742    1612
P00112142    1562
P00057642    1470
             ...
P00314842       1
P00298842       1
P00231642       1
P00204442       1
P00066342       1
Name: count, Length: 3631, dtype: int64
Value counts for field Gender:
Gender
M    414259
F    135809
Name: count, dtype: int64
Value counts for field Age:
Age
26-35    219587
36-45    110013
18-25     99660
46-50     45701
51-55     38501
55+       21504
0-17      15102
Name: count, dtype: int64
Value counts for field Occupation:
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
Value counts for field City_Category:
City_Category
B    231173
C    171175
A    147720
Name: count, dtype: int64
Value counts for field Stay_In_Current_City_Years:
Stay_In_Current_City_Years
1     193821
2     101838
3      95285
4+     84726
0      74398
Name: count, dtype: int64
Value counts for field Marital_Status:
Marital_Status
0    324731
1    225337
Name: count, dtype: int64
Value counts for field Product_Category:
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
```

```
# unique counts for each fields
wmt.nunique(walmart_copy, 'User_ID')
wmt.nunique(walmart_copy, 'Product_ID')
wmt.nunique(walmart_copy, 'Gender')
wmt.nunique(walmart_copy, 'Age')
wmt.nunique(walmart_copy, 'Occupation')
wmt.nunique(walmart_copy, 'City_Category')
wmt.nunique(walmart_copy, 'Stay_In_Current_City_Years')
wmt.nunique(walmart_copy, 'Marital_Status')
wmt.nunique(walmart_copy, 'Product_Category')
```

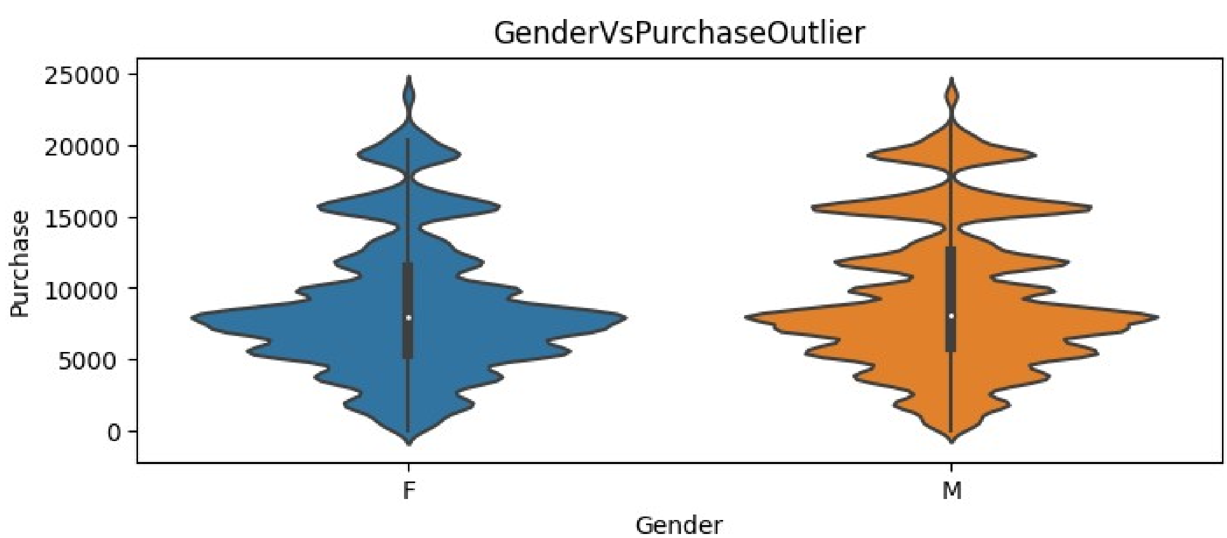
```
Unique values for field User_ID: 5891
Unique values for field Product_ID: 3631
Unique values for field Gender: 2
Unique values for field Age: 7
Unique values for field Occupation: 21
Unique values for field City_Category: 3
Unique values for field Stay_In_Current_City_Years: 5
Unique values for field Marital_Status: 2
Unique values for field Product_Category: 20
```

Business Insights based on Non- Graphical and Visual Analysis

```
- Comments on the range of attributes
    - From the above function, we find that there are 10 attributes,
among which 5 are integers, and 5 are objects
    - Some object variables such as Gender, Age, City_Category and
Marital_Status can be later converted to categorical depending on the
use cases.
    - There are no null values in any of the fields.
    - From the unique counts, many values are repeated, though there
are around 5.5L rows.
    - The total number of unique user IDs is only 5891. Similarly, the
unique product ID count is 3631.
    - We have also determined some statistical values, such as mean,
max, mean, median, and std, for all numeric fields.
```

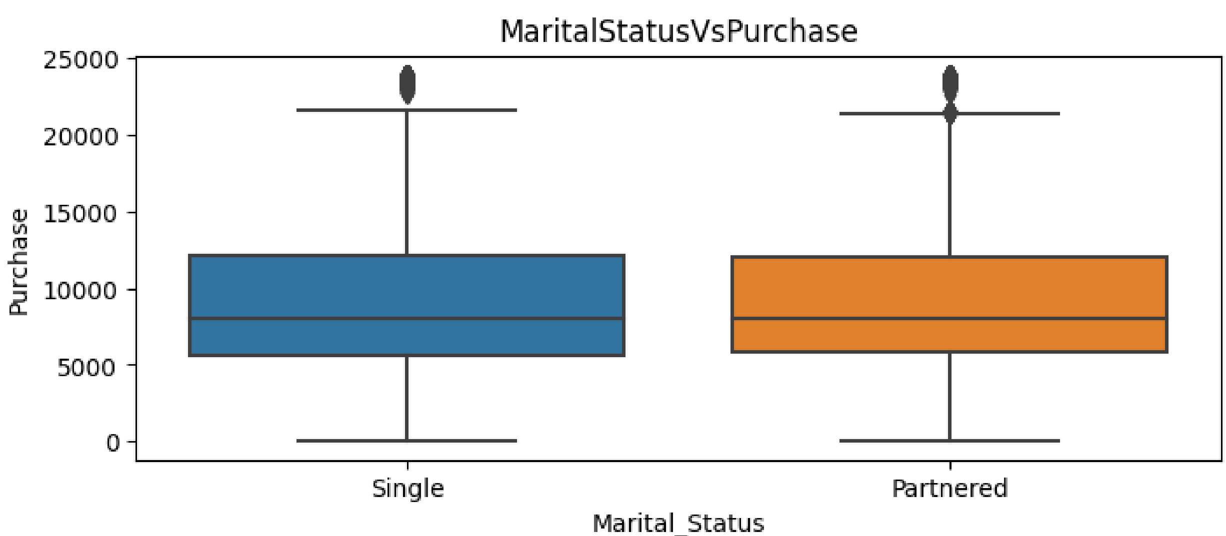
Visual Analysis

```
- Univariate
- Bivariate
- Multivariate
- Outlier detection
- HeatMap

# Individual dataframe for each required field
age_df = wmt.copy_df(walmart_copy)
maritalStatus_df = wmt.copy_df(walmart_copy)
gender_df = wmt.copy_df(walmart_copy)
cityCategory_df = wmt.copy_df(walmart_copy)
occupation_df = wmt.copy_df(walmart_copy)
productCategory_df = wmt.copy_df(walmart_copy)
cityStay_df = wmt.copy_df(walmart_copy)

# Outlier detection for Gender
# Create walmart instance for gender

gender_category_df = wmt.to_category(gender_df, 'Gender')
gender_grouped_df =
wmt.outlier_detection_non_graph(gender_category_df, 'Gender',
'Purchase', 'mean', 'median')
print(gender_grouped_df)
wmtPlot.graphical_outlier_detection(gender_category_df, 'Gender',
'Purchase', 'violin', 'GenderVsPurchaseOutlier')

  Gender         Mean  Median
0      F  8734.565765  7914.0
1      M  9437.526040  8098.0
```



- There is no significant difference in the mean and median purchase for Females and Males.
- However, from the above non-graphical and graphical analysis, we can see that male customers have a higher mean and median value.
- This indicates that Male customers purchase more compared to Female customers.
- Outlier is more for Female customers than male customers.

```
# Outlier detection for Marital Status
# First convert marital status to category
# Assume 0 as Single and 1 as married/partnered
maritalStatus_df = wmt.to_category(maritalStatus_df, 'Marital_Status')
maritalStatus_df['Marital_Status'] =
maritalStatus_df['Marital_Status'].apply(lambda x:
wmt.maritalStatusLabel(x))
maritalStatusGrouped_df =
wmt.outlier_detection_non_graph(maritalStatus_df, 'Marital_Status',
'Purchase', 'mean', 'median')
print(maritalStatusGrouped_df)
wmtPlot.graphical_outlier_detection(maritalStatus_df,
'Marital_Status', 'Purchase', 'box', 'MaritalStatusVsPurchase')

  Marital_Status         Mean   Median
0         Single  9265.907619   8044.0
1      Partnered  9261.174574   8051.0
```



- A similar pattern can be seen for Marital Status Outlier analysis, where there is no significant difference in the mean and median purchase for Single and Partnered.
- However, from the above non-graphical and graphical analysis, we can see that single customers tend to have a higher mean value than partnered customers.
- This indicates that Single customers purchase more than Partnered customers.
- Outlier is more for Partner customers than Single customers.

```
# Outlier detection for City Category
# First, convert City Category to category
cityCategory_df = wmt.to_category(cityCategory_df, 'City_Category')
cityCategoryGrouped_df =
wmt.outlier_detection_non_graph(cityCategory_df, 'City_Category',
'Purchase', 'mean', 'median')
print(cityCategoryGrouped_df)
wmtPlot.graphical_outlier_detection(cityCategory_df, 'City_Category',
'Purchase', 'box', 'CityCategoryVSPurchase')
```

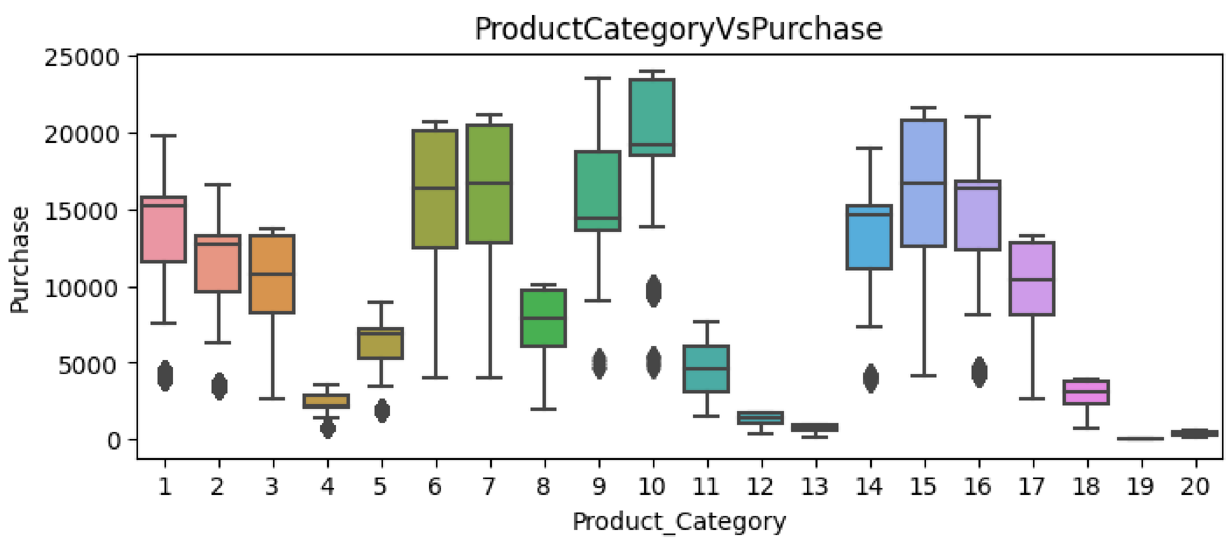
```
  City_Category         Mean  Median
0             A  8911.939216  7931.0
1             B  9151.300563  8005.0
2             C  9719.920993  8585.0
```

- City Category Outlier analysis shows a subtle difference in the cities' mean and median values. However, City C has a higher median and mean than the other two categories, which indicates that customers from City C purchase more than customers of A and B.
- Outlier is more for City A and B, whereas City C has no noticeable outlier.

```
# Outlier detection for Product Category
productCategoryGrouped_df =
wmt.outlier_detection_non_graph(productCategory_df,
'Product_Category', 'Purchase', 'mean', 'median')
print(productCategoryGrouped_df.sort_values('Mean', ascending =
False))
wmtPlot.graphical_outlier_detection(productCategory_df,
'Product_Category', 'Purchase', 'box', 'ProductCategoryVsPurchase')
```

```
   Product_Category          Mean   Median
9                10  19675.570927  19197.0
6                 7  16365.689600  16700.0
5                 6  15838.478550  16312.0
```

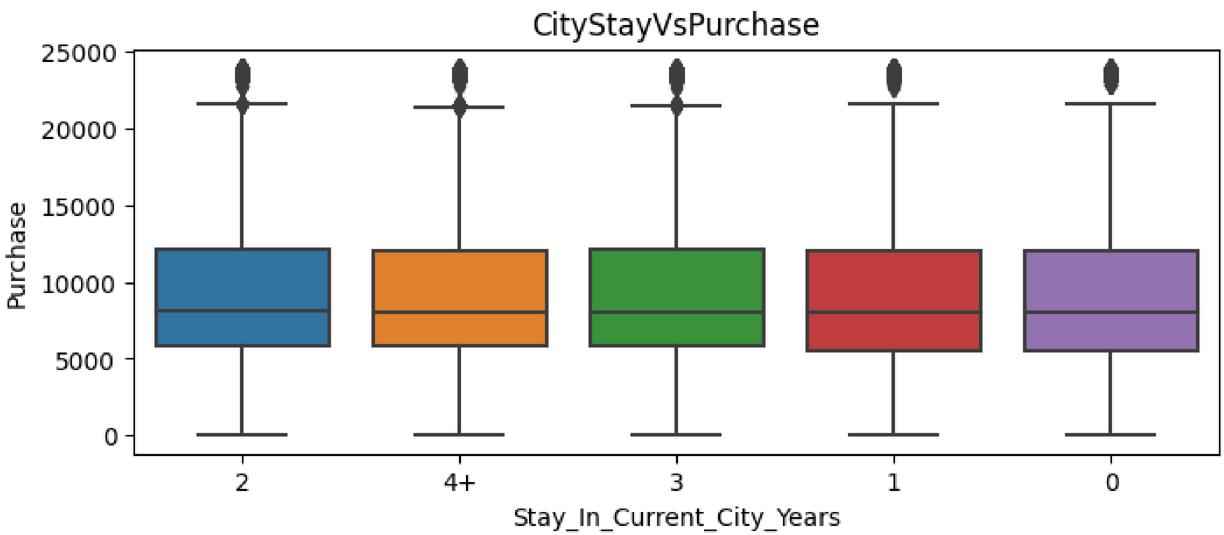
```
8                   9  15537.375610  14388.5
14                 15  14780.451828  16660.0
15                 16  14766.037037  16292.5
0                   1  13606.218596  15245.0
13                 14  13141.625739  14654.0
1                   2  11251.935384  12728.5
16                 17  10170.759516  10435.5
2                   3  10096.705734  10742.0
7                   8   7498.958078   7905.0
4                   5   6240.088178   6912.0
10                 11   4685.268456   4611.0
17                 18   2972.864320   3071.0
3                   4   2329.659491   2175.0
11                 12   1350.859894   1401.0
12                 13    722.400613    755.0
19                 20    370.481176    368.0
18                 19     37.041797     37.0
```




```python
# Outlier detection for Stay in City
cityStayGrouped = wmt.outlier_detection_non_graph(cityStay_df,
'Stay_In_Current_City_Years', 'Purchase',
                                                  'mean',
'median').sort_values(by = 'Mean', ascending = False)
print(cityStayGrouped)
wmtPlot.graphical_outlier_detection(cityStay_df,
'Stay_In_Current_City_Years', 'Purchase', 'box', 'CityStayVsPurchase')
```

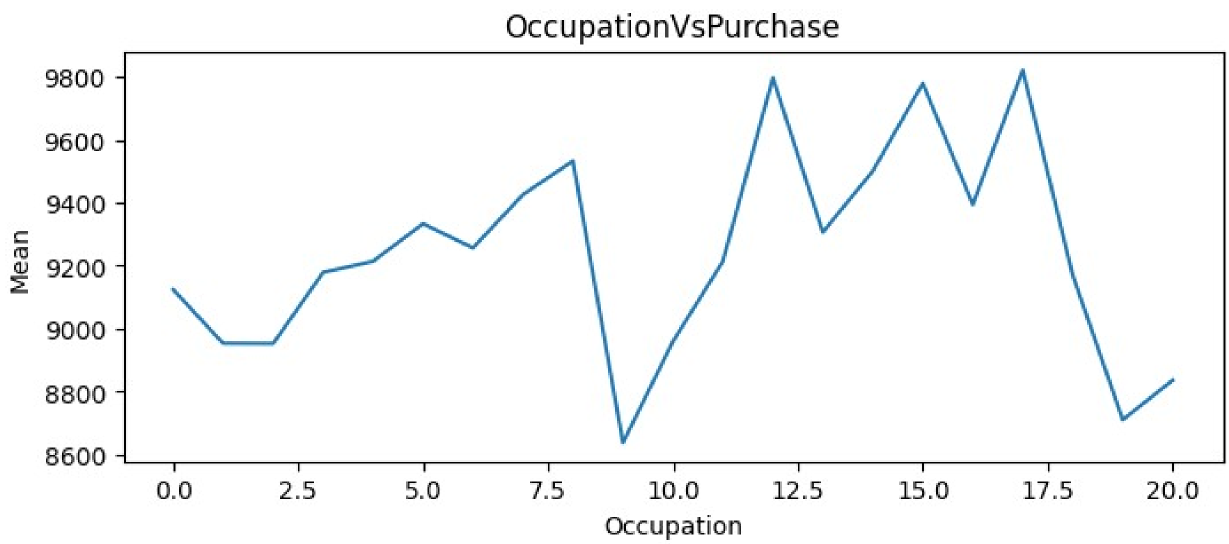
```
  Stay_In_Current_City_Years         Mean  Median
2                          2  9320.429810  8072.0
3                          3  9286.904119  8047.0
4                         4+  9275.598872  8052.0
```

```
1                           1  9250.145923  8041.0
0                           0  9180.075123  8025.0
```



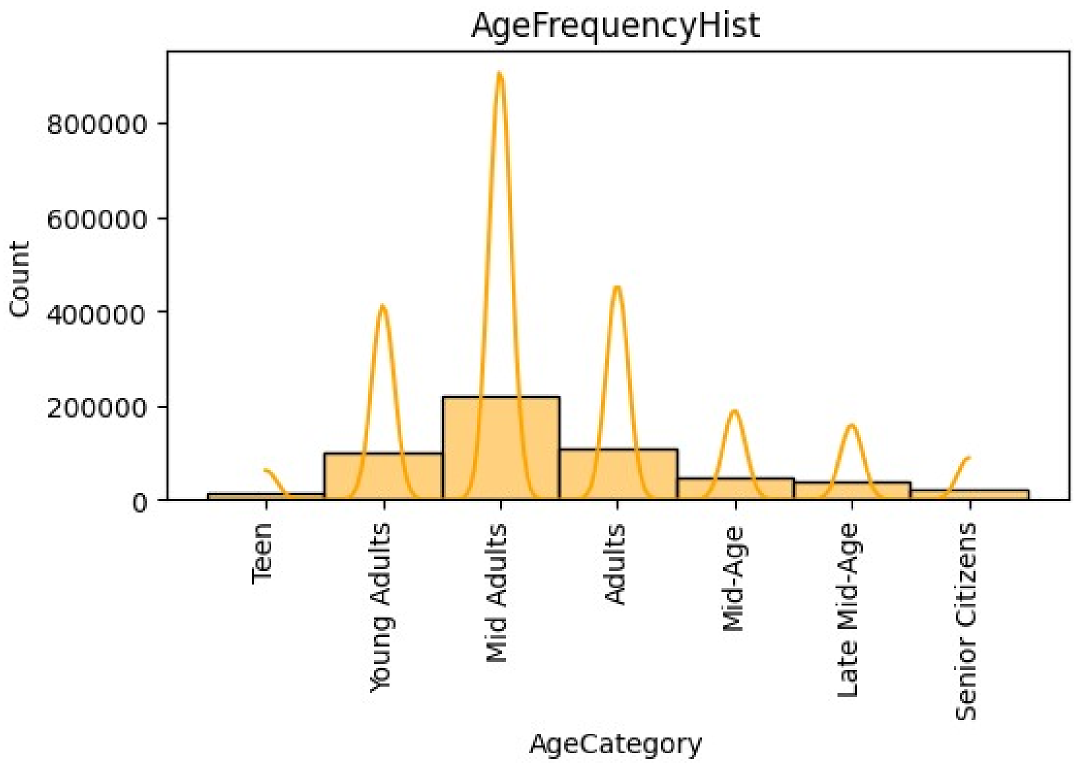

```
# Outlier detection for Occupation
occupationGrouped_df = wmt.outlier_detection_non_graph(occupation_df,
'Occupation', 'Purchase', 'mean', 'median')
print(occupationGrouped_df.sort_values('Mean', ascending = False))
wmtPlot.graphical_outlier_detection(occupationGrouped_df,
'Occupation', 'Mean', 'line', 'OccupationVsPurchase')
```

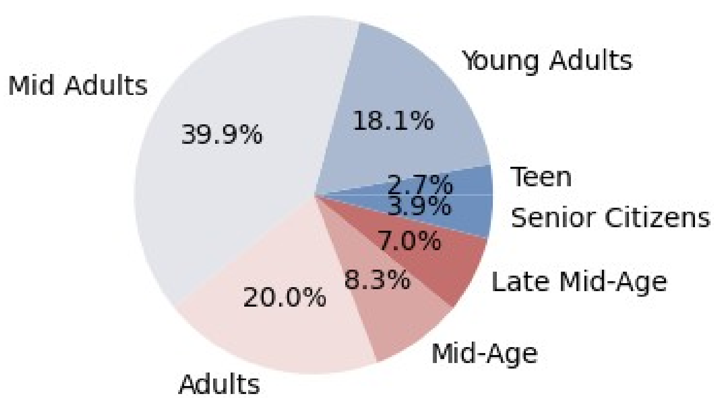
```
    Occupation         Mean  Median
17          17  9821.478236  8635.0
12          12  9796.640239  8569.0
15          15  9778.891163  8513.0
8            8  9532.592497  8419.5
14          14  9500.702772  8122.0
7            7  9425.728223  8069.0
16          16  9394.464349  8070.0
5            5  9333.149298  8080.0
13          13  9306.351061  8090.5
6            6  9256.535691  8050.0
4            4  9213.980251  8043.0
11          11  9213.845848  8041.5
3            3  9178.593088  8008.0
18          18  9169.655844  7955.0
0            0  9124.428588  8001.0
10          10  8959.355375  8012.5
1            1  8953.193270  7966.0
2            2  8952.481683  7952.0
20          20  8836.494905  7903.5
19          19  8710.627231  7840.0
9            9  8637.743761  7886.0
```



General outlier detection analysis

```
- Similarly, outlier detection can be done for product category,
occupation, and stay in the city year
- For occupation, we see 17, 12, and 15 have higher mean and median
values than other occupation
- Customers who have stayed two years in a city have a higher median
followed by 3 years and 4+ years

# Frequency Detection for each Age groups
age_df = wmt.to_category(age_df, 'Age')
age_df['AgeCategory'] = age_df['Age'].apply(lambda x:
wmt.age_to_label(x))
age_grouped_df = wmt.univariate_nongraphical(age_df, 'AgeCategory',
'count')
set_bins = age_grouped_df['AgeCategory'].nunique()
print(age_grouped_df.sort_values('total', ascending = False))
wmtPlot.univariate_plot(age_df, 'AgeCategory', 'hist', set_bins,
'orange', 'AgeFrequencyHist', True)
wmtPlot.univariate_plot(age_grouped_df, 'AgeCategory', 'pie',
'AgeFrequencyPie')

       AgeCategory   total
2       Mid Adults  219587
3           Adults  110013
1     Young Adults   99660
4          Mid-Age   45701
5     Late Mid-Age   38501
6  Senior Citizens   21504
0             Teen   15102
```

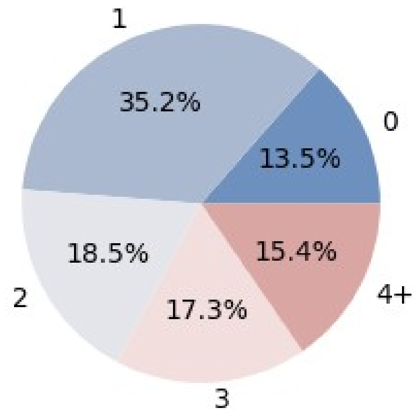





```
# Frequency Detection for each City Stay Range
cityStay_grouped_df = wmt.univariate_nongraphical(cityStay_df,
'Stay_In_Current_City_Years', 'count')
print(cityStay_grouped_df)
wmtPlot.univariate_plot(cityStay_grouped_df,
'Stay_In_Current_City_Years', 'pie', 'StayInCityYearFrequency')
```

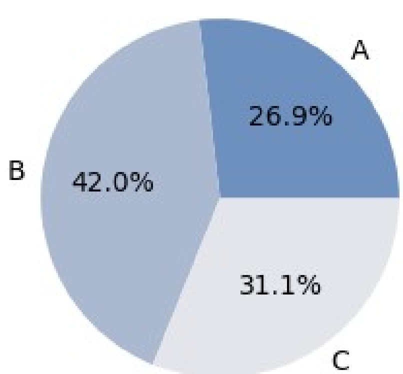
	Stay_In_Current_City_Years	total
0	0	74398
1	1	193821
2	2	101838

```
3                                   3   95285
4                                  4+   84726
```




```
# Frequency Detection for city category
city_category_grouped_df =
wmt.univariate_nongraphical(cityCategory_df, 'City_Category', 'count')
print(city_category_grouped_df.sort_values('total', ascending =
False))
wmtPlot.univariate_plot(city_category_grouped_df, 'City_Category',
'pie', 'CityCategoryFrequency')
```

	City_Category	total
1	B	231173
2	C	171175
0	A	147720



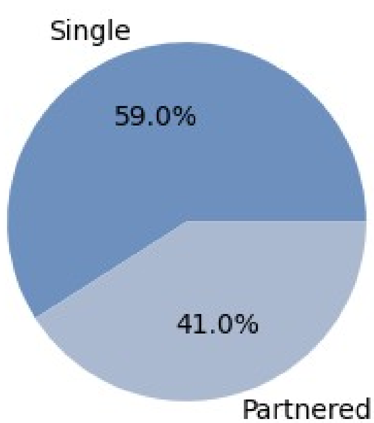

```
# Frequency detection for Gender
gender_grouped_df = wmt.univariate_nongraphical(gender_df, 'Gender',
```

```
'count')
print(gender_grouped_df.sort_values('total', ascending = False))
wmtPlot.univariate_plot(gender_grouped_df, 'Gender', 'pie',
'GenderFrequency')
```

```
  Gender   total
1      M  414259
0      F  135809
```

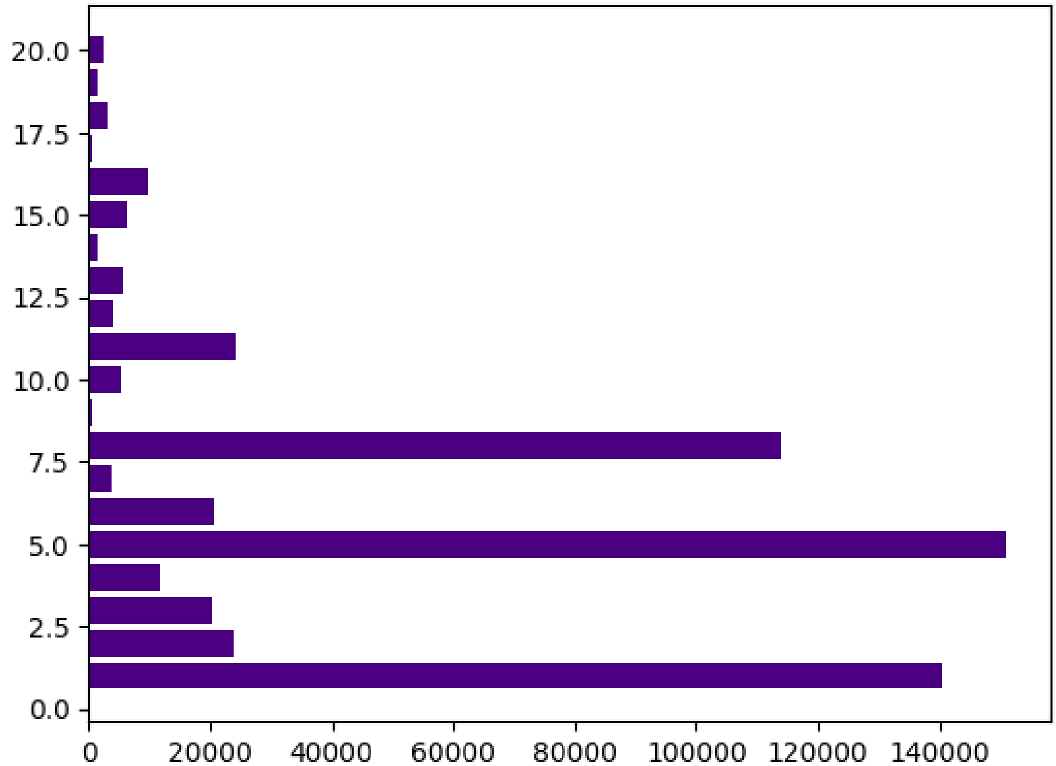

```
# Frequency detection for Marital Status
ms_grouped_df = wmt.univariate_nongraphical(maritalStatus_df,
'Marital_Status', 'count')
print(ms_grouped_df.sort_values('total', ascending = False))
wmtPlot.univariate_plot(ms_grouped_df, 'Marital_Status', 'pie',
'MaritalStatusFrequency')
```

```
  Marital_Status   total
0         Single  324731
1      Partnered  225337
```



```python
# Frequency detection for Product Category
pc_grouped_df = wmt.univariate_nongraphical(productCategory_df,
'Product_Category', 'count').sort_values(by = ['total'], ascending =
False)
print(pc_grouped_df)
plt.barh(pc_grouped_df['Product_Category'], pc_grouped_df['total'],
color = 'indigo')
plt.savefig('ProductCategoryFrequency')
```

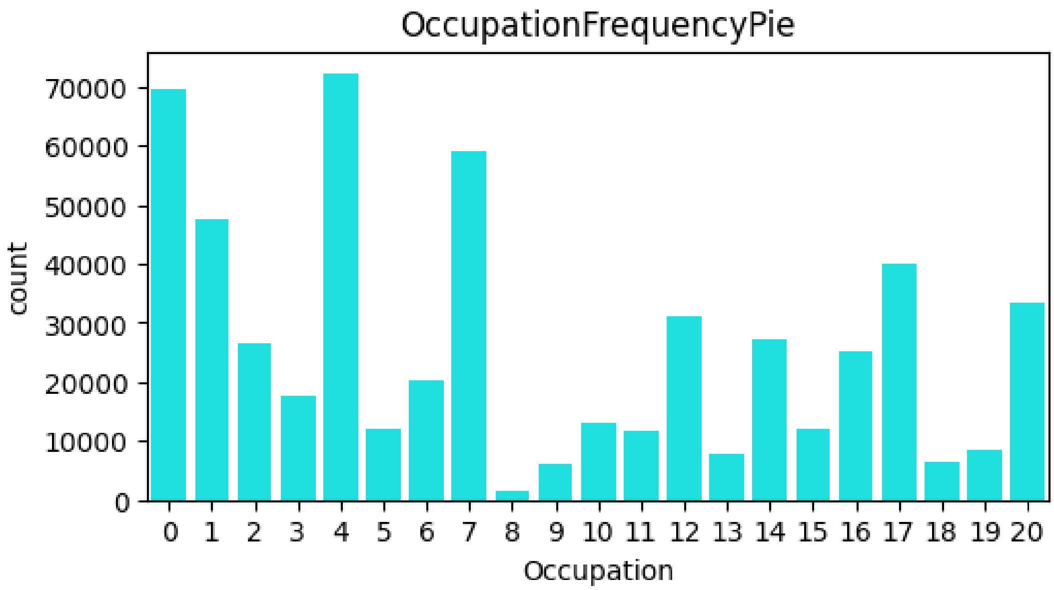
```
    Product_Category   total
4                  5  150933
0                  1  140378
7                  8  113925
10                11   24287
1                  2   23864
5                  6   20466
2                  3   20213
3                  4   11753
15                16    9828
14                15    6290
12                13    5549
9                 10    5125
11                12    3947
6                  7    3721
17                18    3125
19                20    2550
18                19    1603
13                14    1523
16                17     578
8                  9     410
```




```
# Frequency Detection for each Occupation
occ_grouped_df = wmt.univariate_nongraphical(occupation_df,
'Occupation', 'count')
print(occ_grouped_df.sort_values('total', ascending = False))
wmtPlot.univariate_plot(occupation_df, 'Occupation', 'count', color =
'cyan', saveAs = 'OccupationFrequencyPie')
```

	Occupation	total
4	4	72308
0	0	69638
7	7	59133
1	1	47426
17	17	40043
20	20	33562
12	12	31179
14	14	27309
2	2	26588
16	16	25371
6	6	20355
3	3	17650
10	10	12930
5	5	12177
15	15	12165
11	11	11586
19	19	8461

```
13          13    7728
18          18    6622
9            9    6291
8            8    1546
```



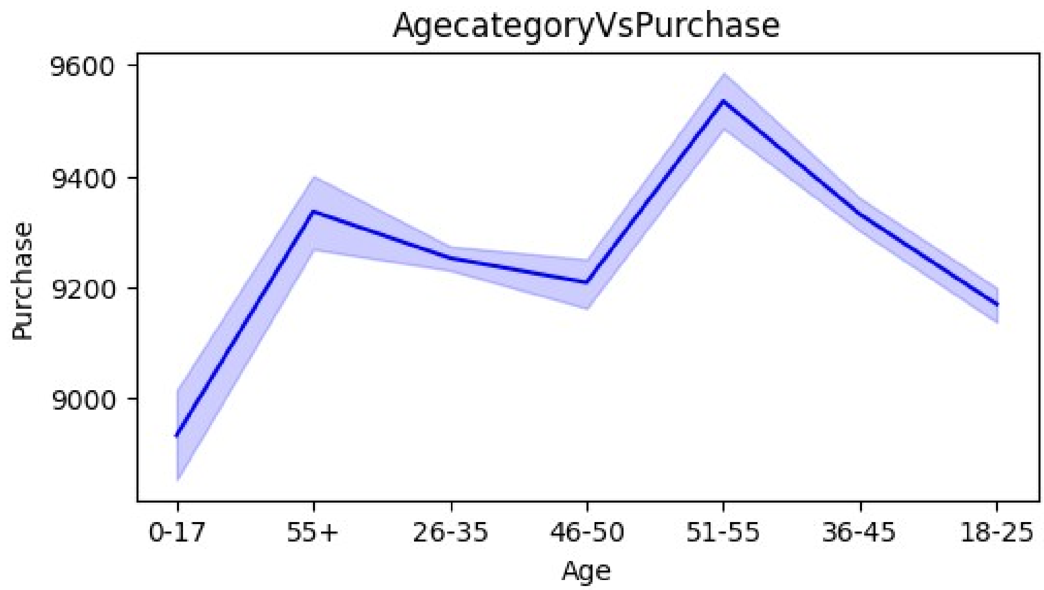
Univariate Graph(Frequency) Analysis

```
- Frequency plots are drawn using three different plots(hist, count,
pie).
- From Age Frequency, we see MidAdults(26-35), Young Adults(18-25) and
Adults(36-45) have higher frequency which indicates
Adult people purchase more than Teens, Mid-age and senior citizens.
- 78% of the customers belong to the age range 18 to 45 who purchase
more.
- Similarly, a marginal difference can be seen from Gender frequency,
which suggests that 75% of Male customers purchase products from
Walmart.
- Single people tend to purchase more, which stands around 59%.
- Customers from City B(42%) spends more than other two Cities.
- Many customers buy from product categories 5, 1, 8, 11 and 2, which
are the top 5 product categories.
- Similar conclusions can be drawn by plotting visualization for
occupation, which shows customers with Occupation categories 4, 0 and
7 purchase more from Walmart.
```

Bivariate Graphs

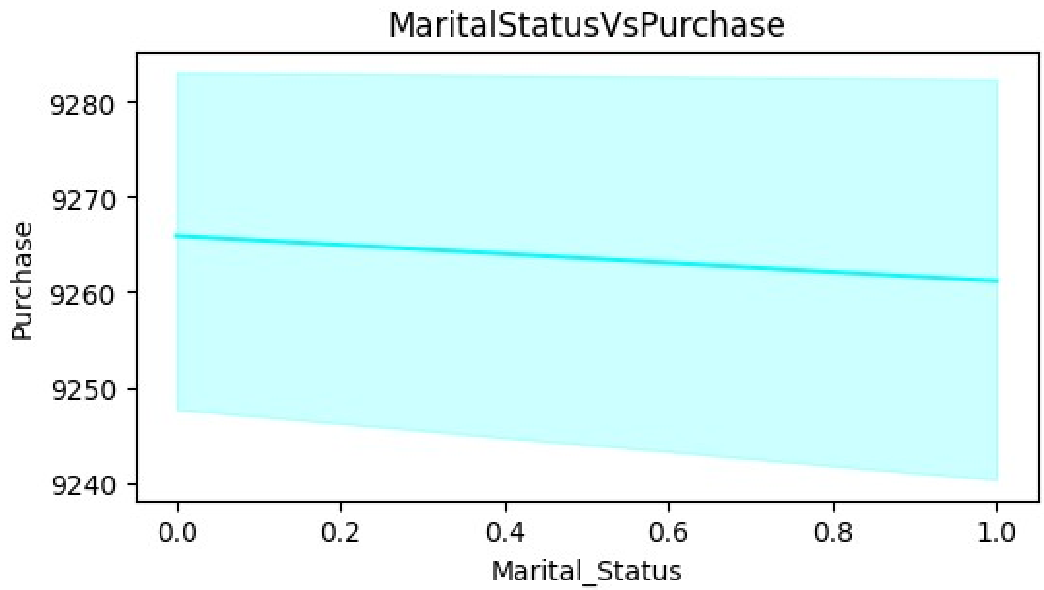
```
# Bivarite Relationship between City Stay and Purchase
def cityStay_vs_purchase(df):
    wmtPlot.bivariate_plot(df, 'City_Category', 'Purchase', 'bar', 
'orange', 'CitycategoryVsPurchase')
cityStay_vs_purchase(walmart_copy)
```


```
# Bivarite Relationship between Age group and Purchase
def ageGroup_vs_purchase(df):
    wmtPlot.bivariate_plot(df, 'Age', 'Purchase', 'line', 'blue', 
'AgecategoryVsPurchase')
ageGroup_vs_purchase(walmart_copy)
```

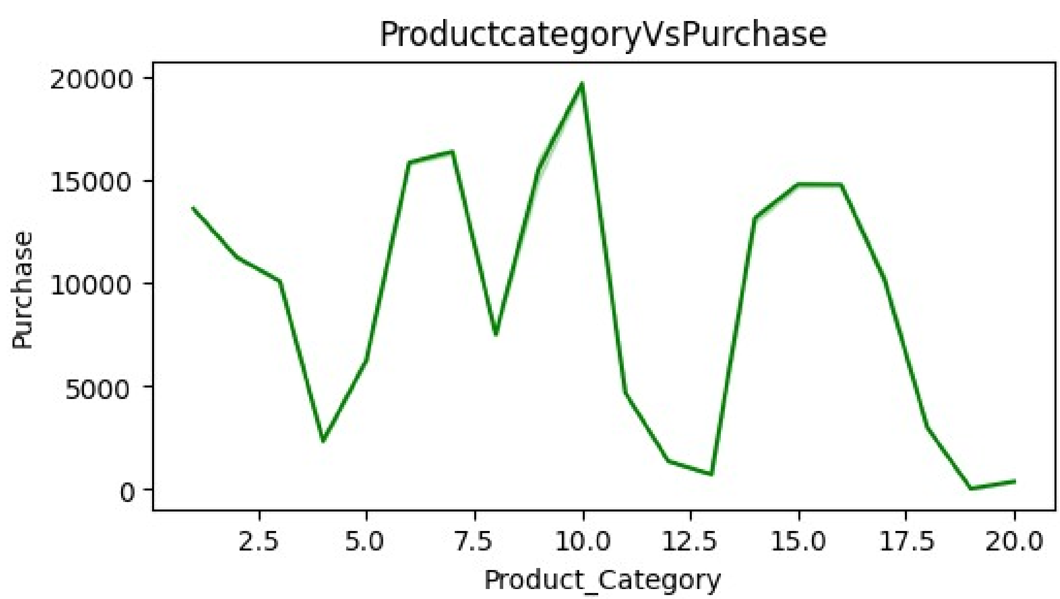



```
# Bivarite Relationship between gender and Purchase
def gender_vs_purchase(df):
    wmtPlot.bivariate_plot(df, 'Gender', 'Purchase', 'bar', 'g',
'GenderVsPurchase')
gender_vs_purchase(walmart_copy)
```

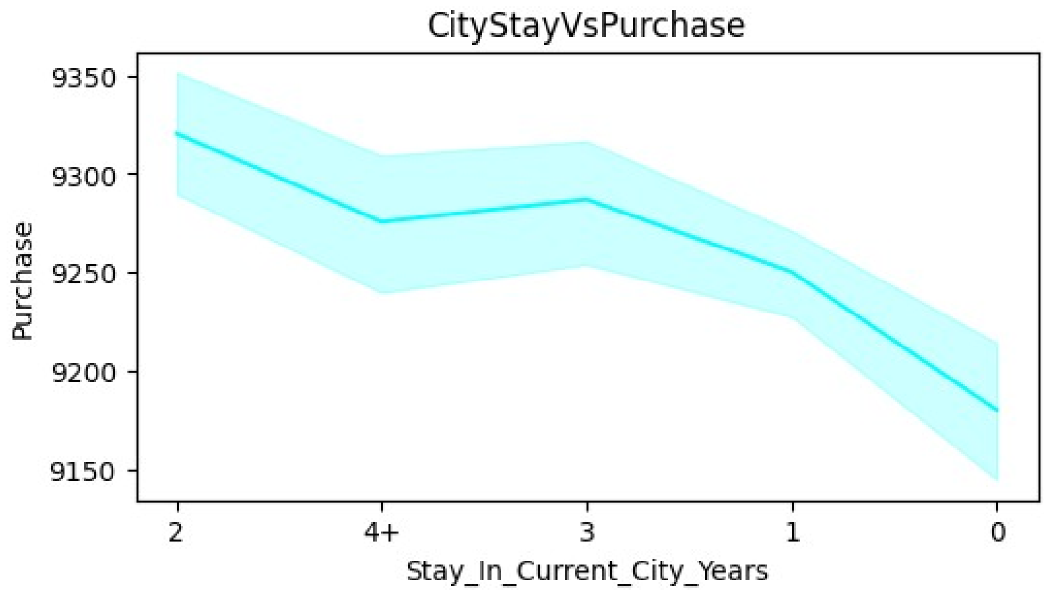

```
# Bivarite Relationship between Marital Status group and Purchase
def ms_vs_purchase(df):
    wmtPlot.bivariate_plot(df, 'Marital_Status', 'Purchase', 'line',
'cyan', 'MaritalStatusVsPurchase')
ms_vs_purchase(walmart_copy)
```



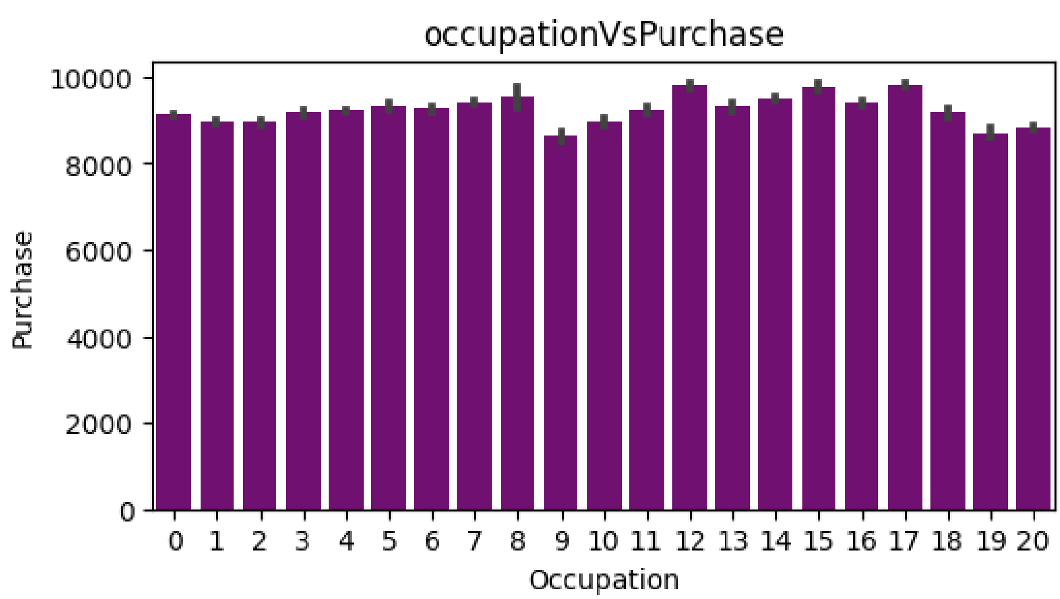

```
# Bivarite Relationship between Product category and Purchase
def PC_vs_purchase(df):
    wmtPlot.bivariate_plot(df, 'Product_Category', 'Purchase', 'line',
'green', 'ProductcategoryVsPurchase')
PC_vs_purchase(walmart_copy)
```




```
# Bivarite Relationship between City Stay vs and Purchase
def cityStay_vs_purchase(df):
    wmtPlot.bivariate_plot(df, 'Stay_In_Current_City_Years',
'Purchase', 'line', 'cyan', 'CityStayVsPurchase')
cityStay_vs_purchase(walmart_copy)
```




```python
# Bivarite Relationship between Occupation vs and Purchase
def occup_vs_purchase(df):
    wmtPlot.bivariate_plot(df, 'Occupation', 'Purchase', 'bar', 
'purple', 'occupationVsPurchase')
occup_vs_purchase(walmart_copy)
```

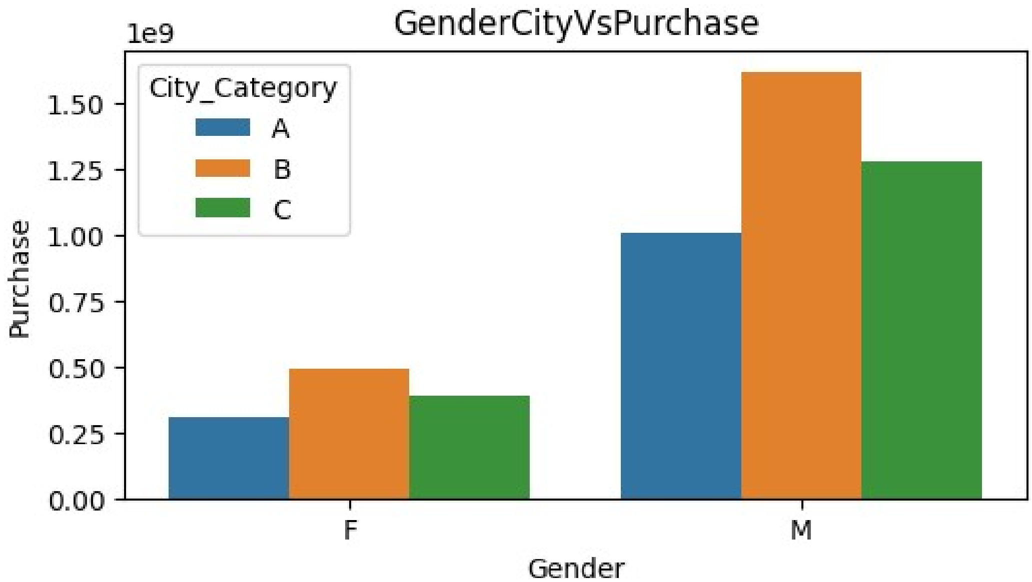


Multi variate Graphs

```python
# Find out the purchase amount for each gender in each city
def gender_vs_city_vs_purchase(df):
```

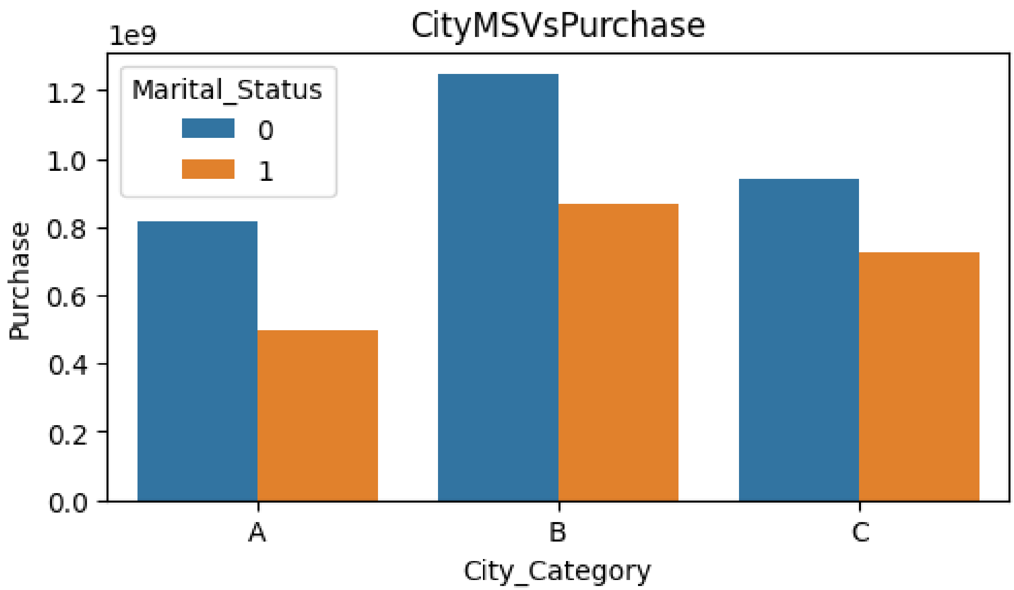
```
    # Male and Females purchase Vs City
    wmt.cross_tab_multiFields(df, 'City_Category', 'Gender',
'Purchase', 'sum')

    gender_grouped_df = df.groupby(['City_Category', 'Gender']).agg({
        'Purchase': 'sum'
    }).reset_index()

    # For each gender
    wmtPlot.multivariate_plots(gender_grouped_df, 'City_Category',
'Purchase', 'Gender', 'bar', 'CityGenderVsPurchase')

    # For each City
    wmtPlot.multivariate_plots(gender_grouped_df, 'Gender',
'Purchase', 'City_Category', 'bar', 'GenderCityVsPurchase')

gender_vs_city_vs_purchase(walmart_copy)
```

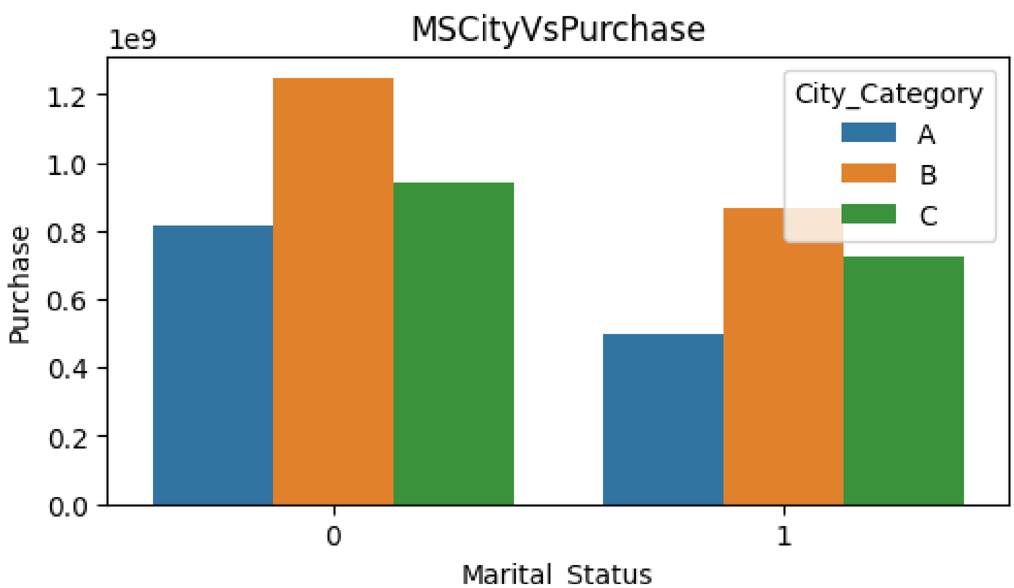
Gender	F	M	All
City_Category			
A	306329915	1010141746	1316471661
B	493617008	1621916597	2115533605
C	386285719	1277521757	1663807476
All	1186232642	3909580100	5095812742




```python
# Find out the purchase amount for each Marital Status in each city
def maritalStatus_vs_city_vs_purchase(df):
    # Male and Females purchase Vs City
    wmt.cross_tab_multiFields(df, 'Marital_Status', 'City_Category',
'Purchase', 'sum')

    ms_grouped_df = df.groupby(['City_Category',
'Marital_Status']).agg({
        'Purchase': 'sum'
    }).reset_index()

    # For each marital status
    wmtPlot.multivariate_plots(ms_grouped_df, 'City_Category',
'Purchase', 'Marital_Status', 'bar', 'CityMSVsPurchase')

    # For each City
    wmtPlot.multivariate_plots(ms_grouped_df, 'Marital_Status',
'Purchase', 'City_Category', 'bar', 'MSCityVsPurchase')

maritalStatus_vs_city_vs_purchase(walmart_copy)
```

City_Category	A	B	C	All
Marital_Status				
0	818350626	1250605488	939971333	3008927447
1	498121035	864928117	723836143	2086885295
All	1316471661	2115533605	1663807476	5095812742

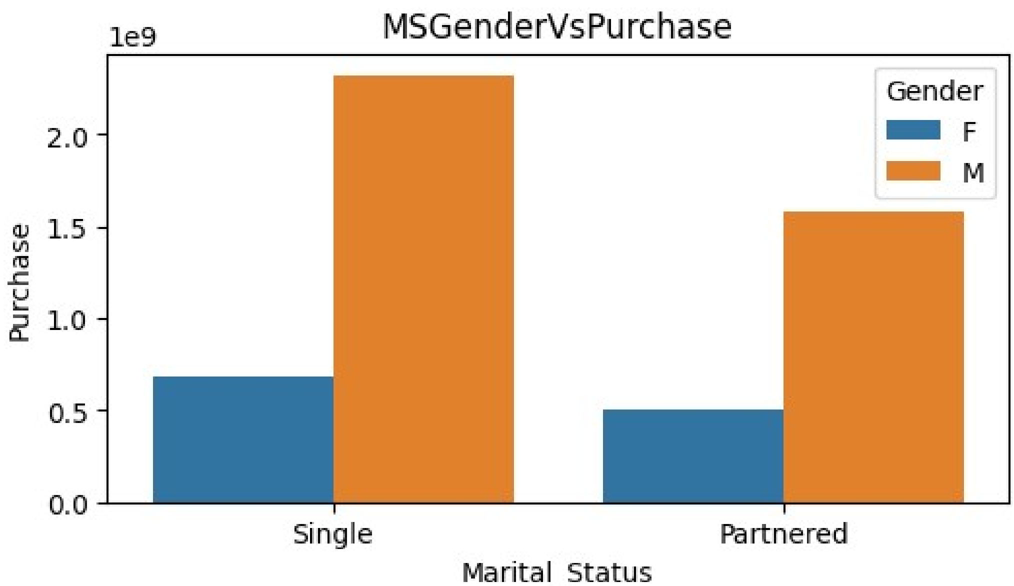


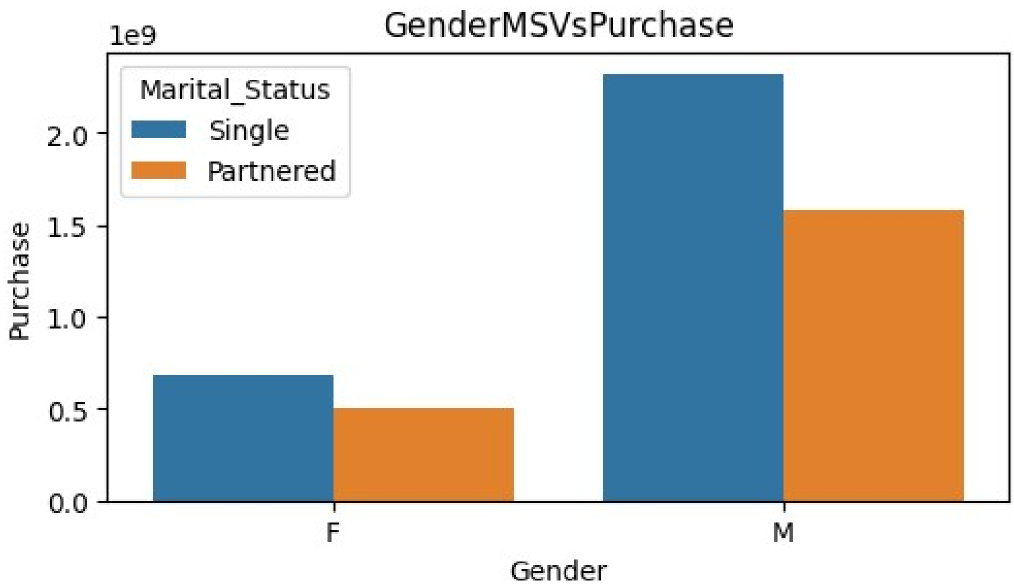



```
# Find out the purchase amount for each gender in each marital status
def gender_vs_ms_vs_purchase(df):
    # Male and Females purchase Vs Marital Status
    wmt.cross_tab_multiFields(df, 'Marital_Status', 'Gender',
'Purchase', 'sum')


    gender_ms_grouped_df = df.groupby(['Marital_Status',
'Gender']).agg({
        'Purchase': 'sum'
    }).reset_index()
```

```
    # For each gender
    wmtPlot.multivariate_plots(gender_ms_grouped_df, 'Marital_Status',
'Purchase', 'Gender', 'bar', 'MSGenderVsPurchase')

    # For each MS
    wmtPlot.multivariate_plots(gender_ms_grouped_df, 'Gender',
'Purchase', 'Marital_Status', 'bar', 'GenderMSVsPurchase')

gender_vs_ms_vs_purchase(maritalStatus_df)
```

Gender	F	M	All
Marital_Status			
Single	684154127	2324773320	3008927447
Partnered	502078515	1584806780	2086885295
All	1186232642	3909580100	5095812742

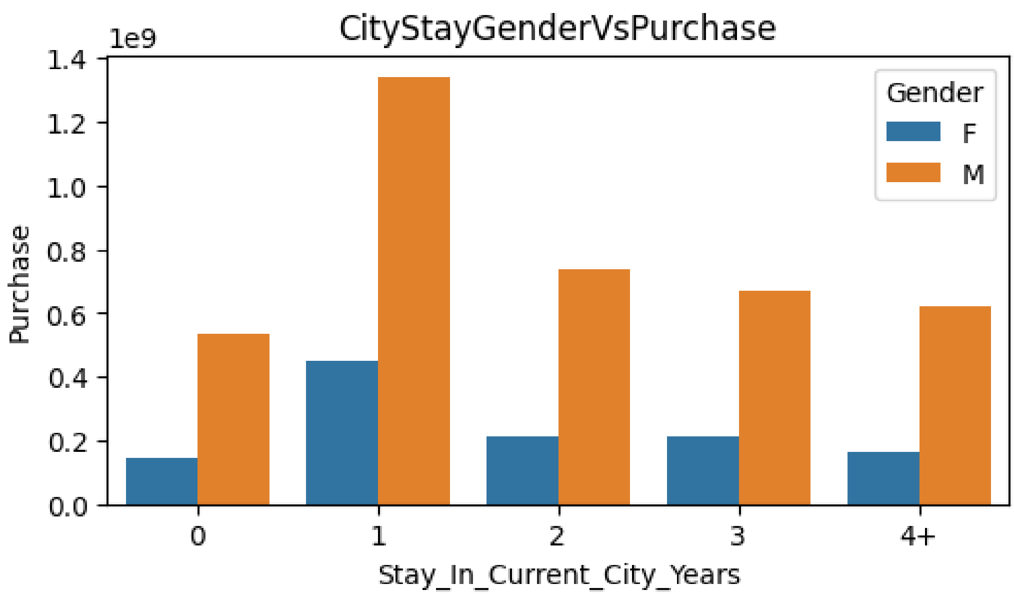


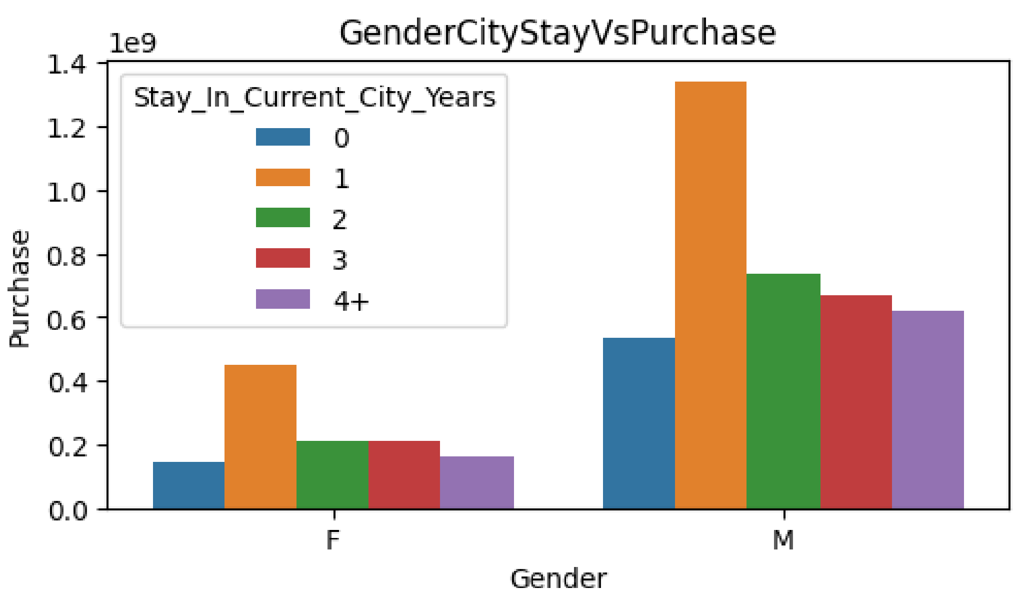



```
# Find out the purchase amount for each gender and their stay in each
city
def gender_vs_city_stay_purchase(df):
    # Male and Females purchase Vs City Stay
    wmt.cross_tab_multiFields(df, 'Stay_In_Current_City_Years',
'Gender', 'Purchase', 'sum')


    gender_grouped_df = df.groupby(['Stay_In_Current_City_Years',
'Gender']).agg({
        'Purchase': 'sum'
    }).reset_index()

    # For each gender
    wmtPlot.multivariate_plots(gender_grouped_df,
'Stay_In_Current_City_Years', 'Purchase', 'Gender', 'bar',
'CityStayGenderVsPurchase')

    # For each City
    wmtPlot.multivariate_plots(gender_grouped_df, 'Gender',
'Purchase', 'Stay_In_Current_City_Years', 'bar',
'GenderCityStayVsPurchase')

gender_vs_city_stay_purchase(walmart_copy)
```

Gender	F	M	All
Stay_In_Current_City_Years			
0	146844869	536134360	682979229
1	450142630	1342729903	1792872533
2	212674244	736499687	949173931
3	213207201	671695458	884902659

```
4+                                  163363698   622520692   785884390
All                                1186232642  3909580100  5095812742
```

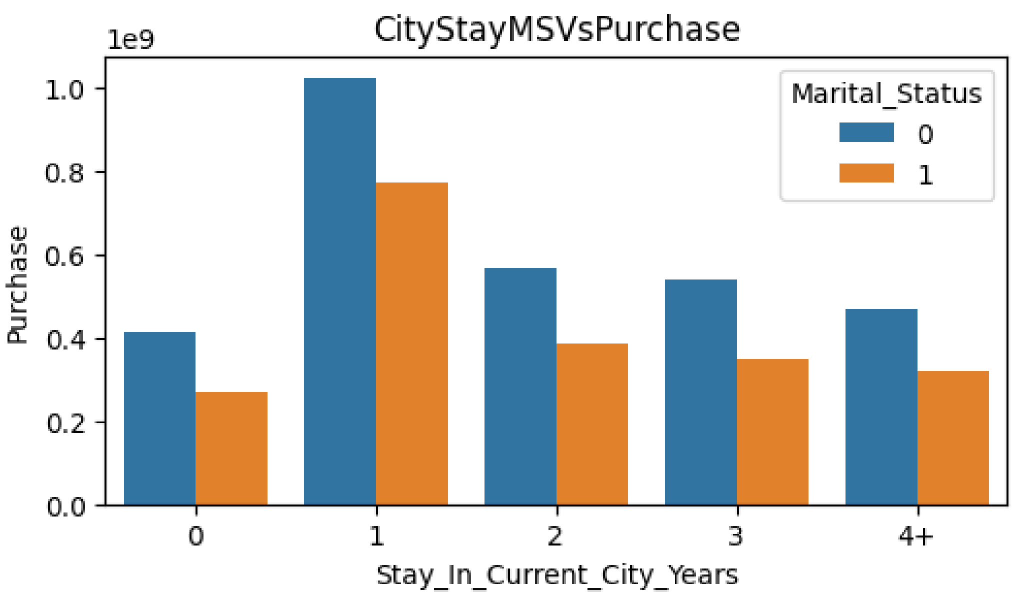


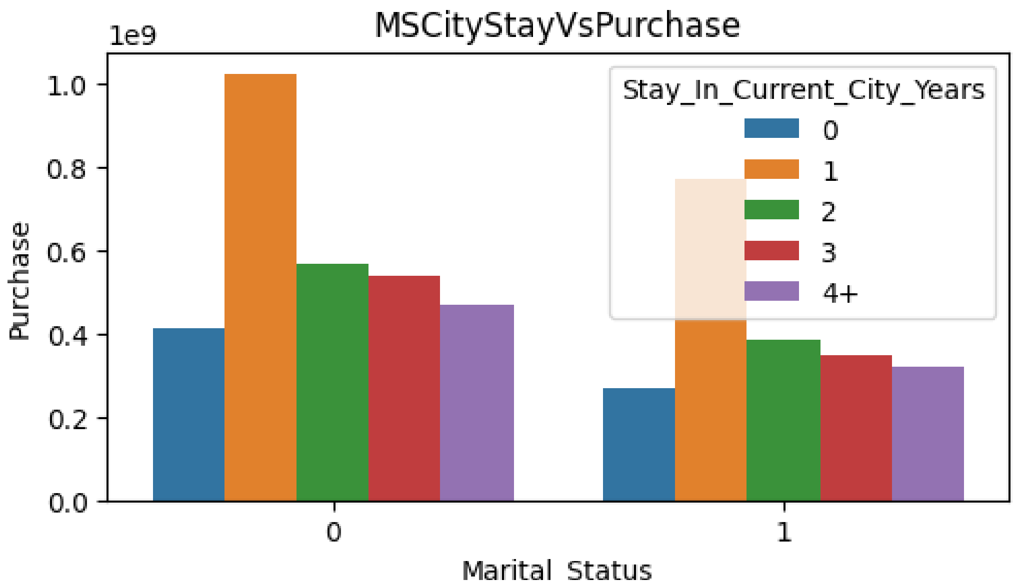



```
# Find out the purchase amount for each gender and their stay in each
city
def ms_vs_city_stay_purchase(df):
    # Single and Partnered purchase Vs City Stay
    wmt.cross_tab_multiFields(df, 'Stay_In_Current_City_Years',
'Marital_Status', 'Purchase', 'sum')

    ms_grouped_df = df.groupby(['Stay_In_Current_City_Years',
```

```
'Marital_Status']).agg({
        'Purchase': 'sum'
    }).reset_index()

    # For each Marital Status
    wmtPlot.multivariate_plots(ms_grouped_df,
'Stay_In_Current_City_Years', 'Purchase', 'Marital_Status', 'bar',
'CityStayMSVsPurchase')

    # For each City
    wmtPlot.multivariate_plots(ms_grouped_df, 'Marital_Status',
'Purchase', 'Stay_In_Current_City_Years', 'bar',
'MSCityStayVsPurchase')

ms_vs_city_stay_purchase(walmart_copy)
```

Marital_Status	0	1	All
Stay_In_Current_City_Years			
0	413140099	269839130	682979229
1	1023036909	769835624	1792872533
2	565881440	383292491	949173931
3	539741219	345161440	884902659
4+	467127780	318756610	785884390
All	3008927447	2086885295	5095812742

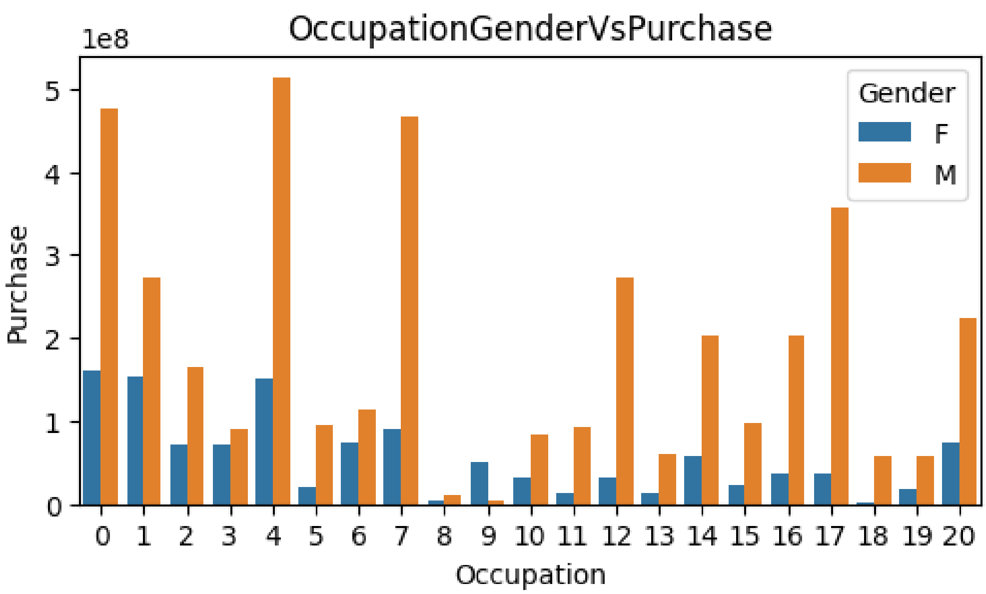





```
# Find out which occupation has the highest purchase and the gender
category
def gender_vs_occup_purchase(df):
    # Male and Females purchase Vs Marital Status
    wmt.cross_tab_multiFields(df, 'Occupation', 'Gender', 'Purchase',
'sum')

    gender_occup_grouped_df = df.groupby(['Gender',
'Occupation']).agg({
        'Purchase': 'sum'
    }).reset_index()

    # For each gender
    wmtPlot.multivariate_plots(gender_occup_grouped_df, 'Occupation',
'Purchase', 'Gender', 'bar', 'OccupationGenderVsPurchase')

gender_vs_occup_purchase(occupation_df)
```

Gender	F	M	All
Occupation			
0	159883833	475523125	635406958
1	152806726	271807418	424614144
2	72569470	165459113	238028583
3	71707639	90294529	162002168
4	152264321	513980163	666244484
5	19595050	94054709	113649759
6	74079792	114336992	188416784
7	91177610	466193977	557371587
8	3379484	11357904	14737388
9	50206487	4133559	54340046
10	32803589	83040876	115844465

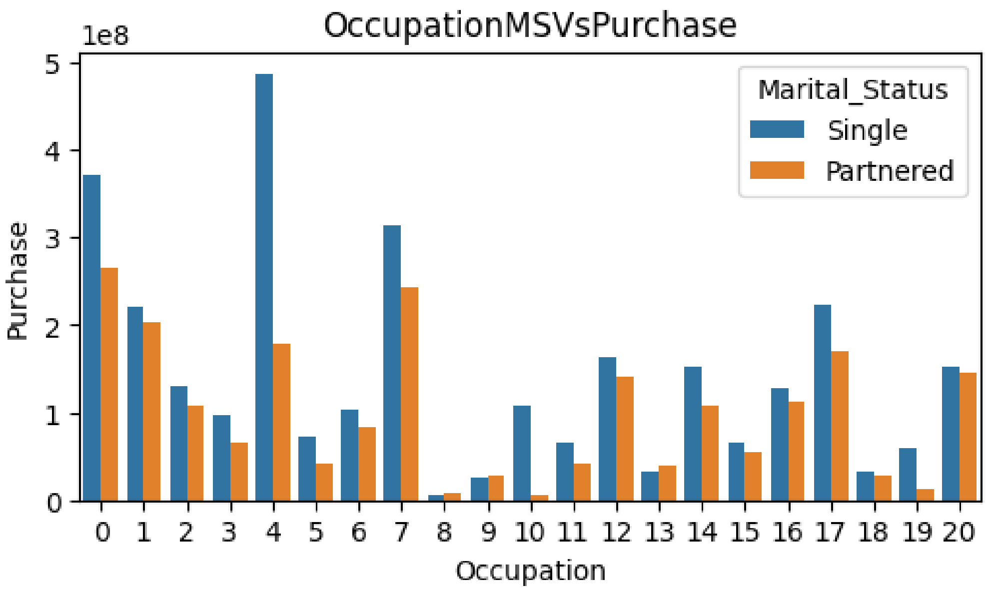
```
11             13636200    93115418   106751618
12             31762002   273687444   305449446
13             12827008    59092473    71919481
14             58010060   201444632   259454692
15             22453799    96506412   118960211
16             36820127   201526828   238346955
17             37496159   355785294   393281453
18              2317160    58404301    60721461
19             17007150    56693467    73700617
20             73428976   223141466   296570442
All          1186232642  3909580100  5095812742
```




```
# Find out which occupation has the highest purchase and the gender
category
def ms_vs_occup_purchase(df):

    # marital status as category
    df = wmt.to_category(df, 'Marital_Status')
    df['Marital_Status'] = df['Marital_Status'].apply(lambda x:
wmt.maritalStatusLabel(x))

    # Married or Single
    wmt.cross_tab_multiFields(df, 'Occupation', 'Marital_Status',
'Purchase', 'sum')

    ms_occup_cross_grouped_df = df.groupby(['Marital_Status',
'Occupation']).agg({
        'Purchase': 'sum'
    }).reset_index()
```

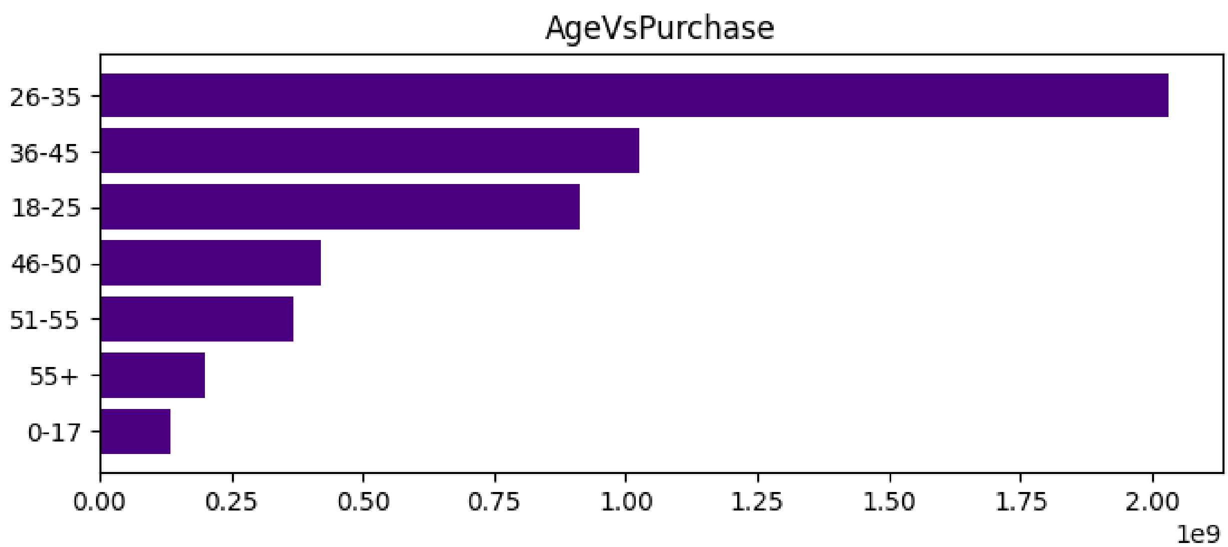
```
    # For each MS
    wmtPlot.multivariate_plots(ms_occup_cross_grouped_df,
'Occupation', 'Purchase',
                       'Marital_Status', 'bar',
'OccupationMSVsPurchase')

ms_vs_occup_purchase(walmart_copy)
```

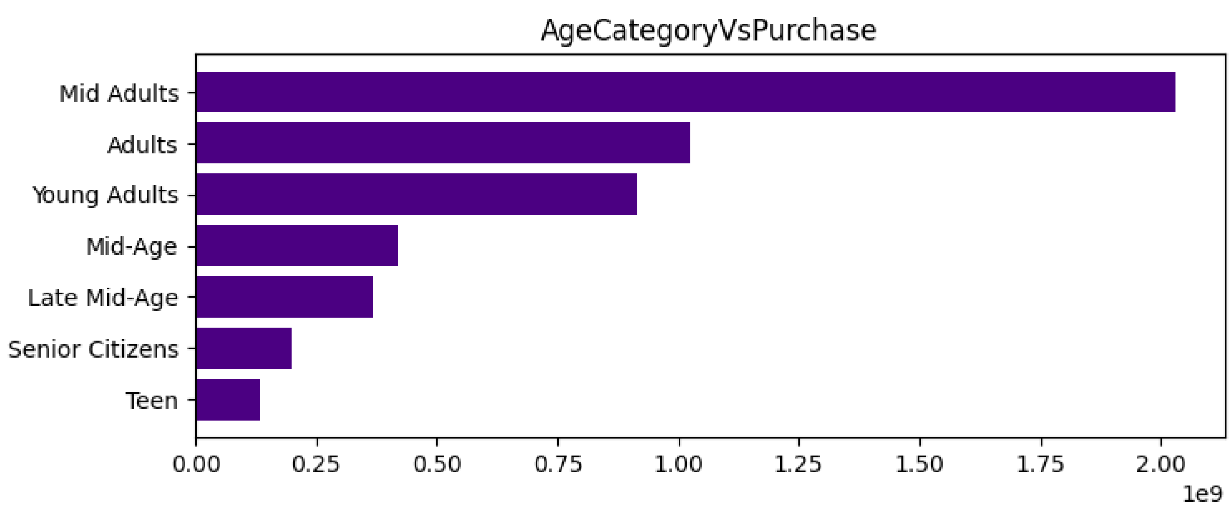
Marital_Status	Single	Partnered	All
Occupation			
0	370825372	264581586	635406958
1	221595254	203018890	424614144
2	130715770	107312813	238028583
3	95988995	66013173	162002168
4	487595558	178648926	666244484
5	71453497	42196262	113649759
6	103806681	84610103	188416784
7	313327193	244044394	557371587
8	6357455	8379933	14737388
9	25676420	28663626	54340046
10	108779592	7064873	115844465
11	64769488	41982130	106751618
12	164023062	141426384	305449446
13	33349086	38570395	71919481
14	151230666	108224026	259454692
15	65142251	53817960	118960211
16	127031633	111315322	238346955
17	222402436	170879017	393281453
18	32842665	27878796	60721461
19	60186842	13513775	73700617
20	151827531	144742911	296570442
All	3008927447	2086885295	5095812742




```
# Find out which occupation has the highest purchase and the gender
category
def ageVsPurchase(df):

    ageCat_purchase_grouped_df = df.groupby('AgeCategory').agg({
        'Purchase': 'sum'
    }).reset_index().sort_values('Purchase', ascending = False)

    age_purchase_grouped_df = df.groupby('Age').agg({
        'Purchase': 'sum'
    }).reset_index().sort_values('Purchase', ascending = False)

    print(age_purchase_grouped_df)
    print(ageCat_purchase_grouped_df)

    # For each age group
    wmtPlot.graphical_outlier_detection(age_purchase_grouped_df[::-1],
                                        'Age', 'Purchase',
                                            'bar',
                                            'AgeVsPurchase')

    wmtPlot.graphical_outlier_detection(ageCat_purchase_grouped_df[::-
1],
                                        'AgeCategory', 'Purchase',
                                            'bar',
                                            'AgeCategoryVsPurchase')

ageVsPurchase(age_df)

     Age     Purchase
2  26-35  2031770578
```

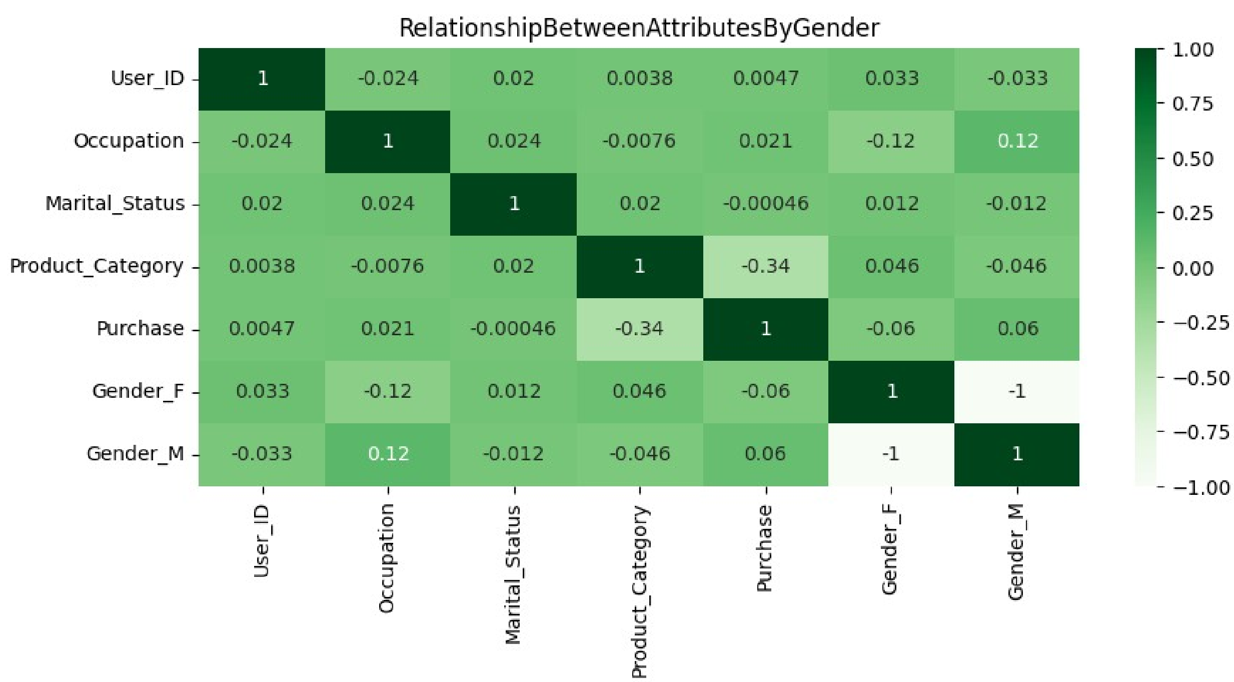
```
3  36-45  1026569884
1  18-25   913848675
4  46-50   420843403
5  51-55   367099644
6    55+   200767375
0   0-17   134913183
        AgeCategory    Purchase
2       Mid Adults  2031770578
3           Adults  1026569884
1     Young Adults   913848675
4          Mid-Age   420843403
5     Late Mid-Age   367099644
6  Senior Citizens   200767375
0             Teen   134913183
```



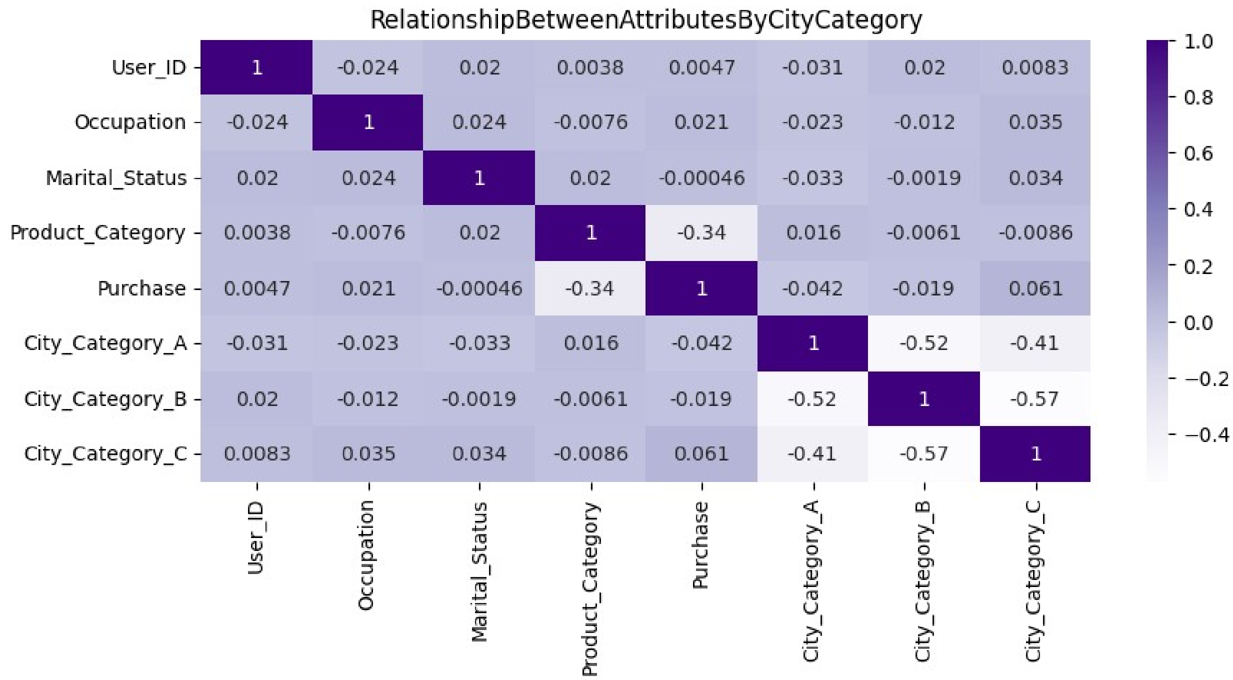


Bivariate/Multivariate Analysis

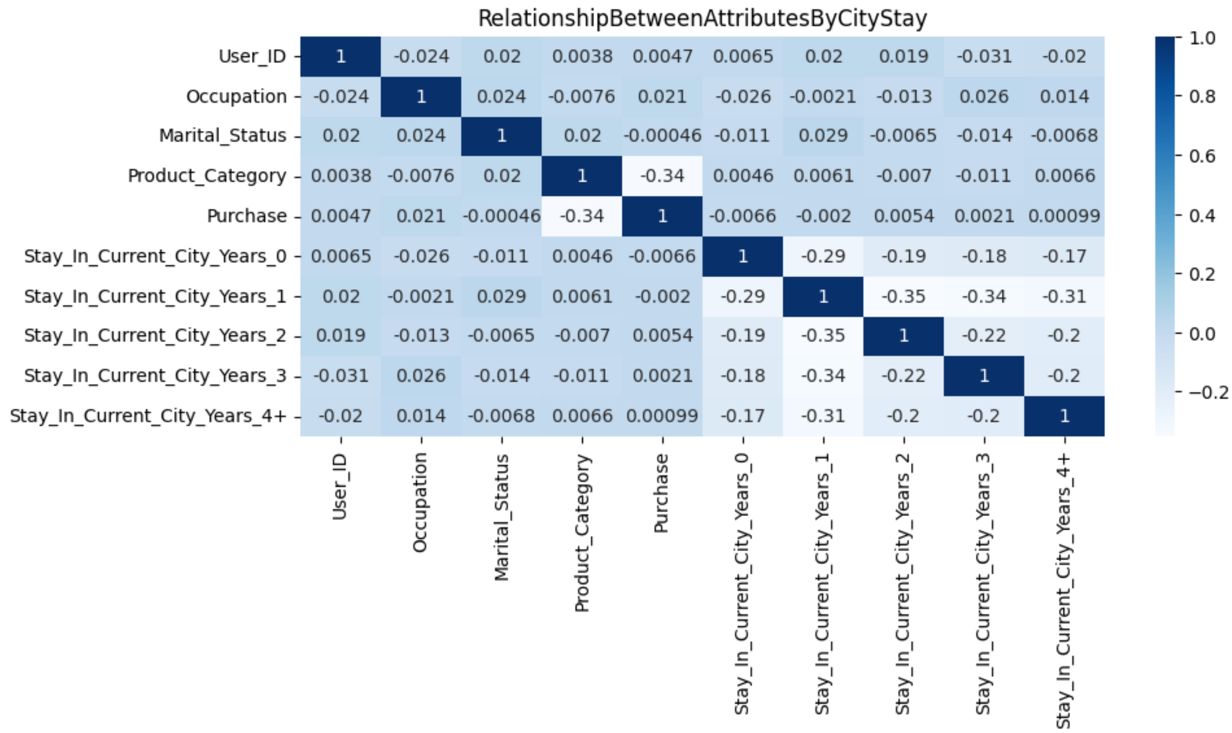
```
- The graphs plot can also do bivariate graph analysis for outlier
detection
- From the outlier detection, we found that Male customers purchase
more than female customers on black Friday
sales.
- Similarly, single customers buy more than partnered customers.
- Customers who belong to the Age range of (26-35) have the highest
purchase, followed by young adults(18-25),
and Adults(36-45).
- Let's analyse some multivariate visualization
The above graphs between City Category and Gender and City category
and Marital Status suggest that single
people who have stayed in City B purchase more on Black Friday.
- We can further deduct from the above visualization that Most single
people who spend more are Male.
A similar deduction can be made for Years of stay in a City, which
shows customers staying in a city for a year purchase
more than other customers.
- Some more deduction can be made by plotting graphs between
Occupation vs. Gender vs. Purchase and Occupation vs.
Marital Status vs. Purchase.

# Correlation between individual attributes by Gender
wmtPlot.heatMap(gender_df, 'Gender', 'Greens',
'RelationshipBetweenAttributesByGender')
```



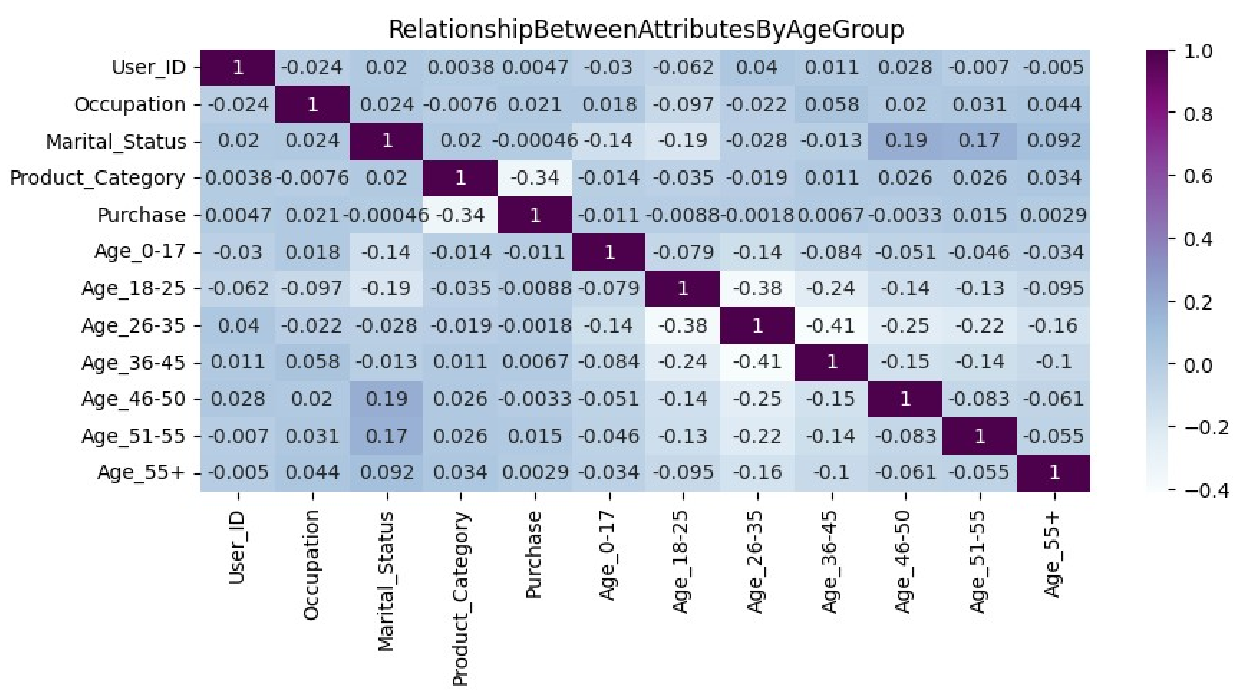
```
# Correlation between individual attributes by City_Category
wmtPlot.heatMap(cityCategory_df, 'City_Category', 'Purples',
'RelationshipBetweenAttributesByCityCategory')
```




```
# Correlation between individual attributes by Stay In Current City
wmtPlot.heatMap(cityStay_df, 'Stay_In_Current_City_Years', 'Blues',
'RelationshipBetweenAttributesByCityStay')
```




```python
# Correlation between individual attributes by age

wmtPlot.heatMap(age_df, 'Age', 'BuPu',
'RelationshipBetweenAttributesByAgeGroup')
```



Confidence Interval and Central Limit Theorem Analysis

```
- We will try to analyze this for each category and provide insights
and recommendations while answering questions
for each category
- We will start with Gender with 90%, 95% and 99% confidence levels.
- Then, we will proceed with similar calculations for each Marital
Status and Age Group.
- Similar plots can be drawn for other variables as well. However, we
will focus more on the above three categories.
```

Purchase Distribution and confidence interval for each Gender

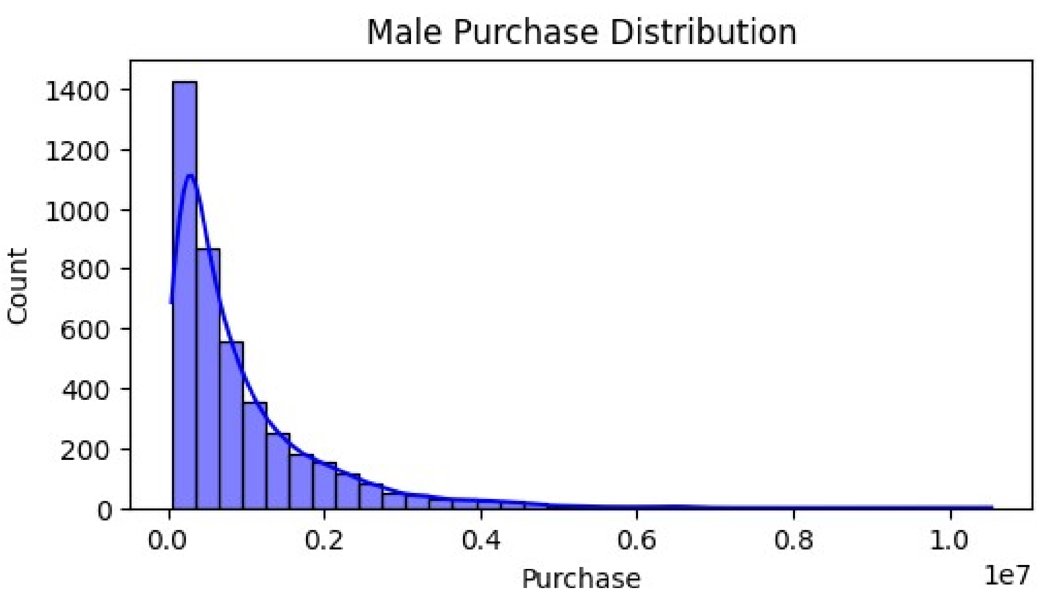
```
- The above graphs and cross-tab functions show a significant
difference in purchases for each gender.
- We have graphical proof that Male customers buy more during Black
Friday than Female customers, which is deducted from only sample data
of 5000 customers. Among these, around 4,000 customers are male, and
1,500 customers are females
- We will prove this theory by calculating CLT and Confidence interval
using 90%, 95% and 99% Confidence levels for all 100 million
customers.
- We will generate a sample size of 3,000 for Males and 1,500 for
Females for a range of 20,000.
- We will calculate the sample mean and standard error for the sample
mean and the range.
```

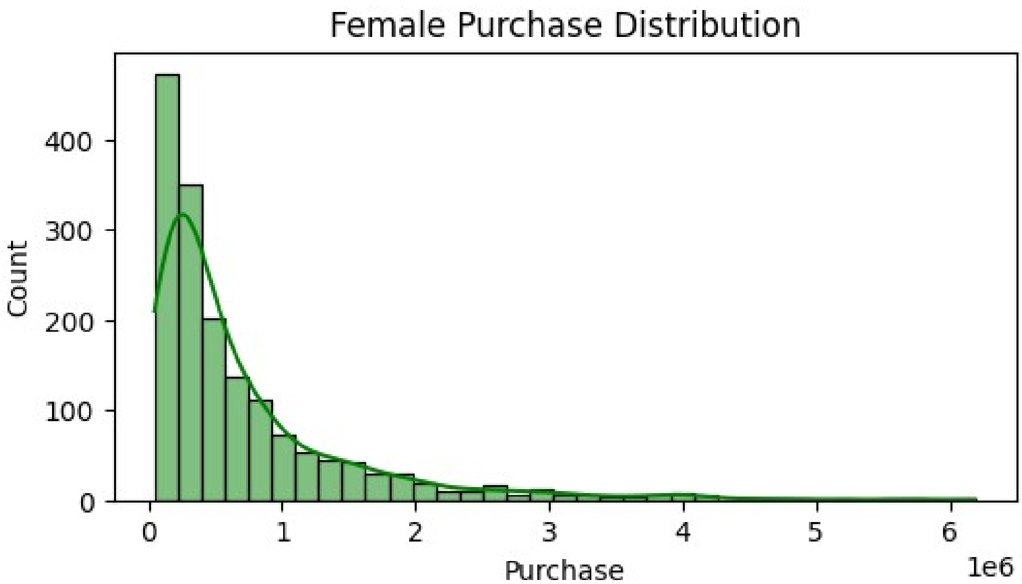
```python
'''
    - Tracking the amount spent per transaction of all 50 million
female and 50 million male customers,
    - calculate the average, and conclude the results.
'''

def gender_CI_CLT(df):

    userId_grouped = wmtCLT.aggregate_function(df, 'Purchase', 'sum',
['User_ID', 'Gender'])
    print(userId_grouped)
    print()

    # Mean purchase for each Gender Type
    gender_mean_purchase_grouped = wmtCLT.aggregate_function(df,
'Purchase', 'mean', ['Gender'])
    print(gender_mean_purchase_grouped)
```

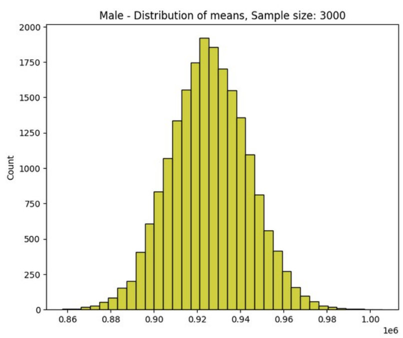
```
    print()

    # Hist plot to display distribution of purchase for Males
    wmtPlot.distribution(userId_grouped, 'Gender', 'M', 'Purchase', 
'hist', 35, 'b', 'Male Purchase Distribution')
    wmtPlot.distribution(userId_grouped, 'Gender', 'F', 'Purchase', 
'hist', 35, 'g', 'Female Purchase Distribution')

    # Total male and female counts for mean distribution
    male_df = wmtCLT.attribute_df(userId_grouped, 'Gender', 'M')
    female_df = wmtCLT.attribute_df(userId_grouped, 'Gender', 'F')

    # Create Purchase sample for Male and Females
    male_purchase_sample_mean = wmtCLT.sample_size_mean(male_df, 3000, 
'Purchase', 20000)
    female_purchase_sample_mean = wmtCLT.sample_size_mean(female_df, 
1500, 'Purchase', 20000)


    # Plot graph to display mean distribution for males and Females
    wmtPlot.gaussian_distribution_2var(
        means1 = male_purchase_sample_mean,
        means2 = female_purchase_sample_mean,
        bins = 35,
        color = 'y',
        title1 = 'Male - Distribution of means, Sample size: 3000',
        title2 = 'Female - Distribution of means, Sample size: 1500',
        saveAs = 'Gender Wise Purchase Mean Distribution')

    # Sample Purchase Mean and standard deviation for each gender

    wmtCLT.sample_mean_std(male_purchase_sample_mean, 
                        female_purchase_sample_mean, 'Male', 'Female')

    print()

    # Confidence Interval for Each category in gender

    wmtCLT.CI_90(male_df, female_df, 'Purchase', 'Male', 'Female')
    wmtCLT.CI_95(male_df, female_df, 'Purchase', 'Male', 'Female')
    wmtCLT.CI_99(male_df, female_df, 'Purchase', 'Male', 'Female')

gender_CI_CLT(walmart_df)
```

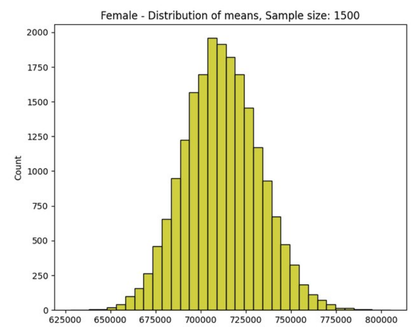
	User_ID	Gender	Purchase
0	1000001	F	334093
1	1000002	M	810472
2	1000003	M	341635
3	1000004	M	206468
4	1000005	M	821001

```
...        ...    ...       ...
5886  1006036      F   4116058
5887  1006037      F   1119538
5888  1006038      F     90034
5889  1006039      F    590319
5890  1006040      M   1653299

[5891 rows x 3 columns]

  Gender     Purchase
0      F  8734.565765
1      M  9437.526040

Sample mean for Male is 925502.3882
Sample Standard Deviation For Male is 18087.886
Sample mean for Female is 711727.0982
Sample Standard Deviation For Female is 20882.3428

With 90% confidence level Male purchase Confidence level lies between
[924960.6044457157, 925728.2002880122]
With 90% confidence level Female purchase Confidence level lies
between [711227.2721085255, 712821.5178074408]
With 95% confidence level Male purchase Confidence level lies between
[924887.0789367019, 925801.725797026]
With 95% confidence level Female purchase Confidence level lies
between [711074.564498548, 712974.2254174183]
With 99% confidence level Male purchase Confidence interval lies
between [924743.3775001303, 925945.4272335975]
With 99% confidence level Female purchase Confidence interval lies
between [710776.1061373135, 713272.6837786528]
```









Insights and Recommendation

```
- From the above distribution graphs, mean and standard deviation
calculation, we have the following observations
    - Sample Purchase Mean for Males - 925353.57
    - Sample Purchase Mean for Females - 712049.4862
    - Sample Purchase STD for Males - 18190.9945
    - Sample Purchase STD for Females - 21122.8525
    - With the above sample mean and STD, we calculated CI for 90%,
95% and 99% Confidence levels.
    - Male Purchase mean interval with 90% CL lies between 924960.60
to 925728.20
    - Female Purchase mean interval with 90% CL lies between 711227.27
to 712821.51
    - Similar observations can be seen for 95% and 99% Confidence
levels.
    - We have also observed in no condition Male mean confidence
interval and the Female mean confidence interval overlap.
    - The male purchase mean is much higher than the female purchase
```

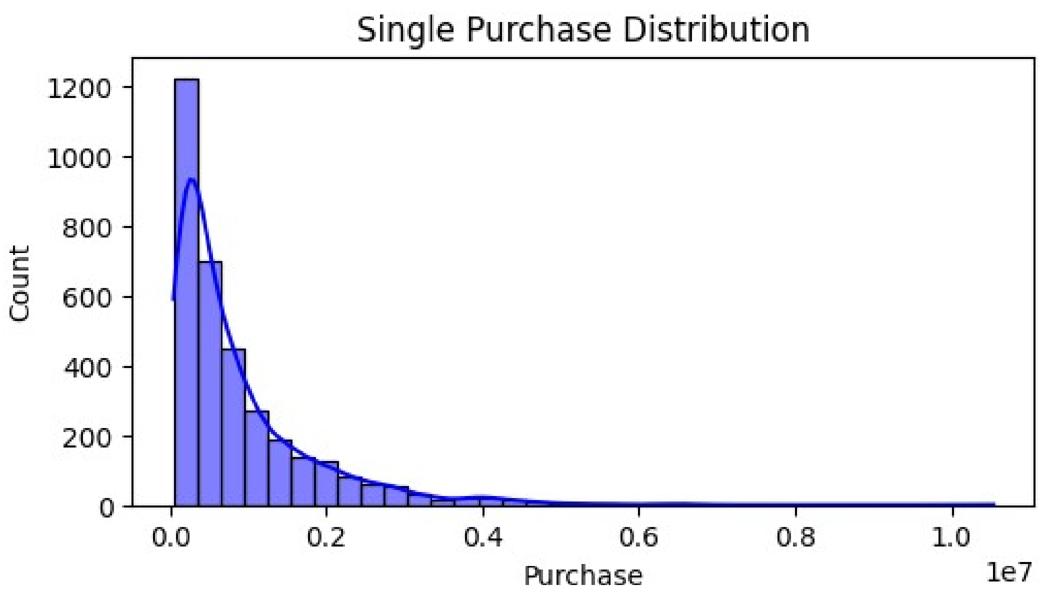
```
mean, which indicates Male customers purchase more.
- Recommendation
    - Walmart need to focus on attracting more male customers.
    - Walmart should update their inventory with products more
preferred by male customers to gain more business.
```

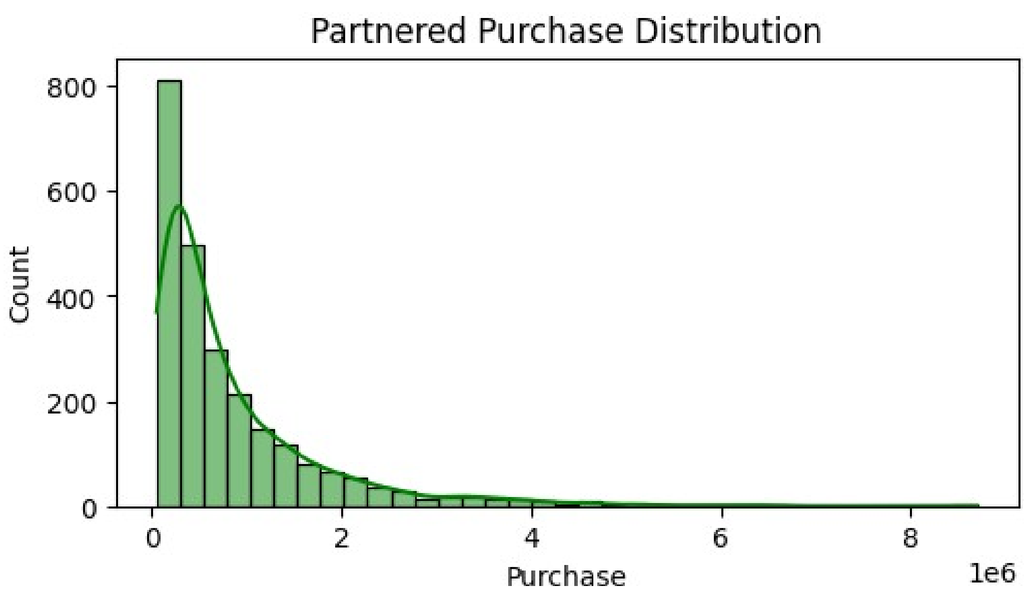
Purchase Distribution and confidence interval for each Marital Status

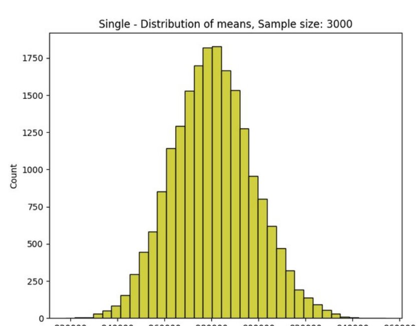
```
 - The above graphs and cross-tab functions show a significant
difference in purchases for each marital status.
- We have graphical proof that Single customers buy more during Black
Friday than Partnered customers, which is deducted from only sample
data of 5000 customers. Among these, around 3,417 customers are
Single, and 2,474 customers are Partnered
- We will prove this theory by calculating CLT and Confidence interval
using 90%, 95% and 99% Confidence levels for all 100 million
customers.
- We will generate a sample size of 3,000 for Single and 2,000 for
Partner of a range of 20,000.
- We will calculate the sample mean and standard error for the sample
mean and the range.

# Group by user id and marital status

def maritalStatus_CLT_CI(df):

    ms_df = wmt.copy_df(df)

    userId_grouped = wmtCLT.aggregate_function(ms_df, 'Purchase',
'sum', ['User_ID', 'Marital_Status'])
    print(userId_grouped)

    # Mean purchase for each Marital Status Type
    ms_mean_purchase_grouped = wmtCLT.aggregate_function(ms_df,
'Purchase', 'mean', ['Marital_Status'])
    print(ms_mean_purchase_grouped)
    print()

    userId_grouped['MS_Label'] =
userId_grouped['Marital_Status'].apply(lambda x :
wmt.maritalStatusLabel(x))

    # Hist plot to display distribution of purchase for Each Marital
Status
    wmtPlot.distribution(userId_grouped, 'MS_Label', 'Single',
'Purchase',
```

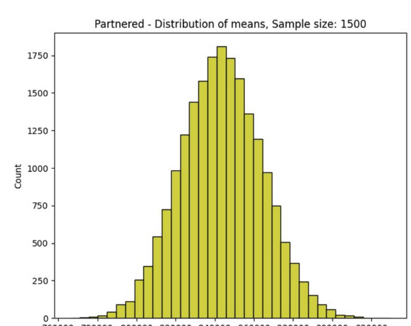
```
                         'hist', 35, 'b', 'Single Purchase
Distribution')
    wmtPlot.distribution(userId_grouped, 'MS_Label', 'Partnered',
'Purchase',
                         'hist', 35, 'g', 'Partnered Purchase
Distribution')

    #Total single and partnered counts for mean distribution
    single_df = wmtCLT.attribute_df(userId_grouped, 'MS_Label',
'Single')
    partnered_df = wmtCLT.attribute_df(userId_grouped, 'MS_Label',
'Partnered')

    # Create Purchase sample for Single and Partnered
    single_mean = wmtCLT.sample_size_mean(single_df, 3000, 'Purchase',
20000)
    partnered_mean = wmtCLT.sample_size_mean(partnered_df, 2000,
'Purchase', 20000)


    # Plot graph to display mean distribution for single and partners
    wmtPlot.gaussian_distribution_2var(
        means1 = single_mean,
        means2 = partnered_mean,
        bins = 35,
        color = 'y',
        title1 = 'Single - Distribution of means, Sample size: 3000',
        title2 = 'Partnered - Distribution of means, Sample size:
1500',
        saveAs = 'Marital Status Wise Purchase Mean Distribution')

    # Sample Purchase Mean and standard deviation for each Marital
Status
    wmtCLT.sample_mean_std(single_mean,
                        partnered_mean, 'Single', 'Partnered')

    print()

    # Confidence Interval for Each category in Marital Status

    wmtCLT.CI_90(single_df, partnered_df, 'Purchase', 'Single',
'Partnered')
    wmtCLT.CI_95(single_df, partnered_df, 'Purchase', 'Single',
'Partnered')
    wmtCLT.CI_99(single_df, partnered_df, 'Purchase', 'Single',
'Partnered')

maritalStatus_CLT_CI(walmart_df)
```

```
      User_ID  Marital_Status  Purchase
0     1000001               0    334093
1     1000002               0    810472
2     1000003               0    341635
3     1000004               1    206468
4     1000005               1    821001
...       ...             ...       ...
5886  1006036               1   4116058
5887  1006037               0   1119538
5888  1006038               0     90034
5889  1006039               1    590319
5890  1006040               0   1653299

[5891 rows x 3 columns]
   Marital_Status     Purchase
0               0  9265.907619
1               1  9261.174574

Sample mean for Single is 880544.6493
Sample Standard Deviation For Single is 17381.4063
Sample mean for Partnered is 843391.9394
Sample Standard Deviation For Partnered is 20804.1959

With 90% confidence level Single purchase Confidence level lies
between [880118.7484171378, 881032.8155278432]
With 90% confidence level Partnered purchase Confidence level lies
between [842904.9222634633, 844148.6711075958]
With 95% confidence level Single purchase Confidence level lies
between [880031.1929017428, 881120.3710432382]
With 95% confidence level Partnered purchase Confidence level lies
between [842785.7876071621, 844267.8057638969]
With 99% confidence level Single purchase Confidence interval lies
between [879860.0706135628, 881291.4933314181]
With 99% confidence level Partnered purchase Confidence interval lies
between [842552.9456794345, 844500.6476916246]
```


Single Purchase Distribution
Count
Purchase
1e7


Partnered Purchase Distribution
Count
Purchase
1e6


Single - Distribution of means, Sample size: 3000
Count


Partnered - Distribution of means, Sample size: 1500
Count

Insights and Recommendation

```
- From the above distribution graphs, mean and standard deviation
calculation, we have the following observations
    - Sample Purchase Mean for Singles - 880573.72
    - Sample Purchase Mean for Partnered - 843585.27
    - Sample Purchase STD for Singles - 17274.1709
    - Sample Purchase STD for Partnered - 20830.2717
    - With the above sample mean and STD, we calculated CI for 90%,
95% and 99% Confidence levels.
    - Single Customers Purchase mean interval with 90% CL lies between
880118.74 to 881032.81
    - Partnered Purchase mean interval with 90% CL lies between
842904.92 to 844148.67
    - Similar observations can be seen for 95% and 99% Confidence
levels.
    - We have also observed that the mean confidence interval for each
group does not overlap.
    - The single purchase mean is much higher than the partnered
purchase means, which indicates single customers purchase more.
- Recommendation
    - Walmart need to focus on attracting more Single customers.
    - We have also noticed from multivariate graphs that Male single
customers purchase more than
    single females and partnered males and females, which Walmart
needs to consider while bringing new
    products to the market and the price of products its loyal
customers prefer.
```

Purchase Distribution and confidence interval for each Age Group

```
- '0-17' : 'Teen'
- '18-25' : 'Young Adults'
- '26-35' : 'Mid Adults'
- '36-45' : 'Adults'
- '46-50' : 'Mid-Age'
- '51-55' : 'Late Mid-Age'
- 55+ : 'Senior Citizens'

 - The above graphs and cross-tab functions show a significant
purchase difference for each age category.
- We have graphical proof that Customers aged 18 to 45 buy more during
Black Friday than Female customers, which is deducted from only sample
data of 5000 customers. Among these, around 218 are Teens, 1069 are
young adults, 2053 are mid-adults, 1167 are adults, 531 are mid-age,
481 are late mid-age, and 372 are senior citizens
```

```
- We will prove this theory by calculating CLT and Confidence interval
using 90%, 95% and 99% Confidence levels for all 100 million
customers.
- We will generate different sample sizes for different age groups for
a range of 20,000.
- We will calculate the sample mean and standard error for the sample
mean and the range.

# Group by user id and Age Category

def ageGroup_CLT_CI(df):

    age_df = wmt.copy_df(df)

    userId_grouped = wmtCLT.aggregate_function(age_df, 'Purchase',
'sum', ['User_ID', 'Age'])
    print(userId_grouped.value_counts('Purchase', ascending = False))

    # Mean purchase for each age group
    age_purchase_grouped = wmtCLT.aggregate_function(age_df,
'Purchase', 'mean', ['Age'])
    print(age_purchase_grouped.sort_values('Purchase', ascending =
False))
    print()

    userId_grouped['Age_Label'] = userId_grouped['Age'].apply(lambda x
: wmt.age_to_label(x))

    # Hist plot to display distribution of purchase for Each Age
Category
    wmtPlot.distribution(userId_grouped, 'Age_Label', 'Teen',
'Purchase',
                         'hist', 35, 'b', 'Teen(0-17) Purchase
Distribution')
    wmtPlot.distribution(userId_grouped, 'Age_Label', 'Young Adults',
'Purchase',
                         'hist', 35, 'g', 'Young Adult(18-25) Purchase
Distribution')
    wmtPlot.distribution(userId_grouped, 'Age_Label', 'Mid Adults',
'Purchase',
                         'hist', 35, 'purple', 'Mid Adults(26-35)
Purchase Distribution')
    wmtPlot.distribution(userId_grouped, 'Age_Label', 'Adults',
'Purchase',
                         'hist', 35, 'r', 'Adult(36-45) Purchase
Distribution')
    wmtPlot.distribution(userId_grouped, 'Age_Label', 'Mid-Age',
'Purchase',
                         'hist', 35, 'orange', 'Mid Age(46-50)
Purchase Distribution')
```

```
    wmtPlot.distribution(userId_grouped, 'Age_Label', 'Late Mid-Age',
'Purchase',
                         'hist', 35, 'yellow', 'Late Mid Age(51-55)
Purchase Distribution')
    wmtPlot.distribution(userId_grouped, 'Age_Label', 'Senior
Citizens', 'Purchase',
                         'hist', 35, 'cyan', 'Senior Citizens(55+)
Purchase Distribution')

    #Total age category counts for mean distribution
    teen_df = wmtCLT.attribute_df(userId_grouped, 'Age_Label', 'Teen')
    youngAdult_df = wmtCLT.attribute_df(userId_grouped, 'Age_Label',
'Young Adults')
    midAdult_df = wmtCLT.attribute_df(userId_grouped, 'Age_Label',
'Mid Adults')
    adult_df = wmtCLT.attribute_df(userId_grouped, 'Age_Label',
'Adults')
    midAge_df = wmtCLT.attribute_df(userId_grouped, 'Age_Label', 'Mid-
Age')
    lateMidAge_df = wmtCLT.attribute_df(userId_grouped, 'Age_Label',
'Late Mid-Age')
    sc_df = wmtCLT.attribute_df(userId_grouped, 'Age_Label', 'Senior
Citizens')

    # Create Purchase sample for Single and Partnered
    teen_df_mean = wmtCLT.sample_size_mean(teen_df, 200, 'Purchase',
20000)
    youngAdult_df_mean = wmtCLT.sample_size_mean(youngAdult_df, 1000,
'Purchase', 20000)
    midAdult_df_mean = wmtCLT.sample_size_mean(midAdult_df, 2000,
'Purchase', 20000)
    adult_df_mean = wmtCLT.sample_size_mean(adult_df, 1000,
'Purchase', 20000)
    midAge_df_mean = wmtCLT.sample_size_mean(midAge_df, 500,
'Purchase', 20000)
    lateMidAge_df_mean = wmtCLT.sample_size_mean(lateMidAge_df, 400,
'Purchase', 20000)
    sc_df_mean = wmtCLT.sample_size_mean(sc_df, 300, 'Purchase',
20000)


    # Plot graph to display mean distribution for males and Females
    wmtPlot.gaussian_distribution_multiVar(
        mean1 = teen_df_mean,
        mean2 = youngAdult_df_mean,
        mean3 = midAdult_df_mean,
        mean4 = adult_df_mean,
        mean5 = midAge_df_mean,
        mean6 = lateMidAge_df_mean,
        mean7 = sc_df_mean,
```

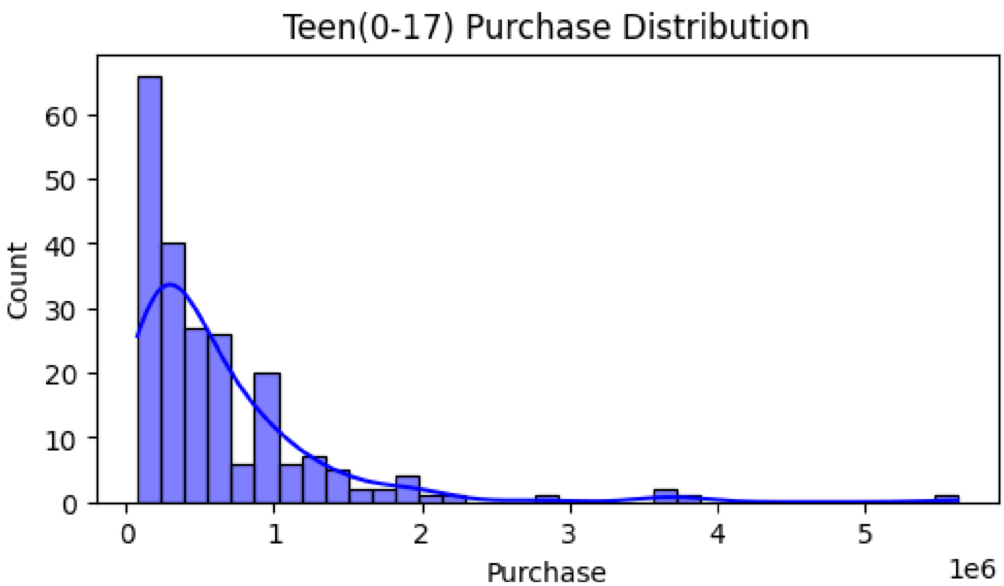
```
        bins = 35,
        color = 'purple',
        title1 = 'Teen(0-17) - Distribution of means, Sample size:
200',
        title2 = 'Young Adults(18-25) - Distribution of means, Sample
size: 1000',
        title3 = 'Mid Aults(26-35) - Distribution of means, Sample
size: 2000',
        title4 = 'Adults(36-45) - Distribution of means, Sample size:
1000',
        title5 = 'Mid Age(46-50) - Distribution of means, Sample size:
500',
        title6 = 'Late Mid Age(51-55) - Distribution of means, Sample
size: 400',
        title7 = 'Senior Citizens(55+) - Distribution of means, Sample
size: 300',
        saveAs = 'Age Group Wise Purchase Mean Distribution')

    # Sample Purchase Mean and standard deviation for each Age Group
    wmtCLT.sample_mean_std(teen_df_mean,
                        youngAdult_df_mean, 'Teens', 'Young Adults')
    wmtCLT.sample_mean_std(midAdult_df_mean,
                        adult_df_mean, 'Mid Aults', 'Adults')
    wmtCLT.sample_mean_std(midAge_df_mean,
                        lateMidAge_df_mean, 'Mid-Age', 'Late Mid-Age')
    wmtCLT.sample_mean_std(lateMidAge_df_mean,
                        sc_df_mean, 'Late Mid-Age', 'Senior Citizens')

    print()

    # Confidence Interval for Each category in Marital Status

    wmtCLT.CI_90(teen_df, youngAdult_df, 'Purchase', 'Teen', 'Young
Adults')
    wmtCLT.CI_95(teen_df, youngAdult_df, 'Purchase', 'Teen', 'Young
Adults')
    wmtCLT.CI_99(teen_df, youngAdult_df, 'Purchase', 'Teen', 'Young
Adults')

    print()

    wmtCLT.CI_90(midAdult_df, adult_df, 'Purchase', 'Mid Adults',
'Adults')
    wmtCLT.CI_95(midAdult_df, adult_df, 'Purchase', 'Mid Adults',
'Adults')
    wmtCLT.CI_99(midAdult_df, adult_df, 'Purchase', 'Mid Adults',
'Adults')

    print()
```

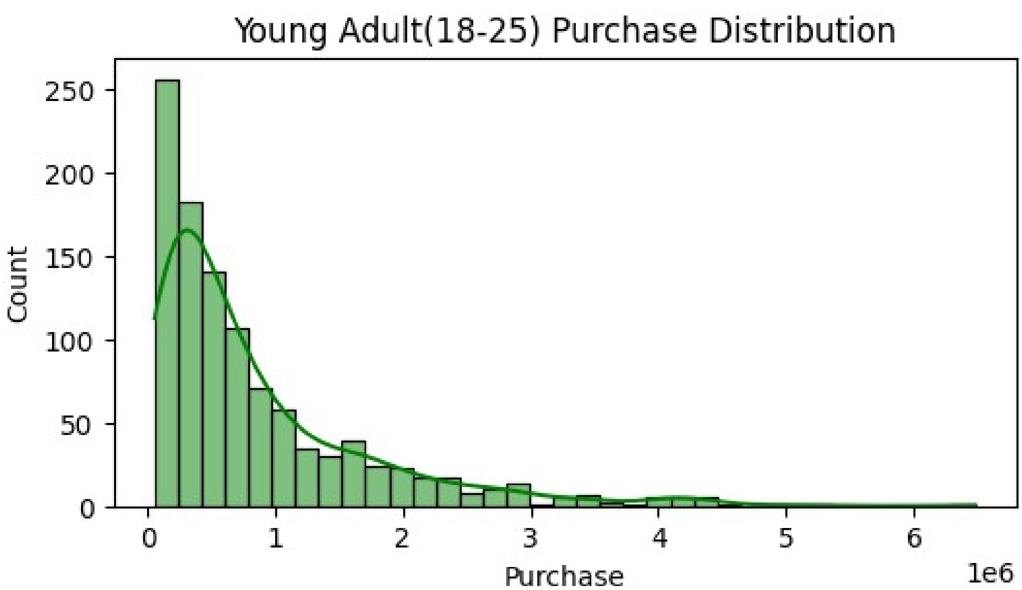
```
    wmtCLT.CI_90(midAge_df, lateMidAge_df, 'Purchase', 'Mid-Age',
'Late Mid-Age')
    wmtCLT.CI_95(midAge_df, lateMidAge_df, 'Purchase', 'Mid-Age',
'Late Mid-Age')
    wmtCLT.CI_99(midAge_df, lateMidAge_df, 'Purchase', 'Mid-Age',
'Late Mid-Age')

    print()

    wmtCLT.CI_90(lateMidAge_df, sc_df, 'Purchase', 'Late Mid-Age',
'Senior Citizens')
    wmtCLT.CI_95(lateMidAge_df, sc_df, 'Purchase', 'Late Mid-Age',
'Senior Citizens')
    wmtCLT.CI_99(lateMidAge_df, sc_df, 'Purchase', 'Late Mid-Age',
'Senior Citizens')

ageGroup_CLT_CI(walmart_df)
```

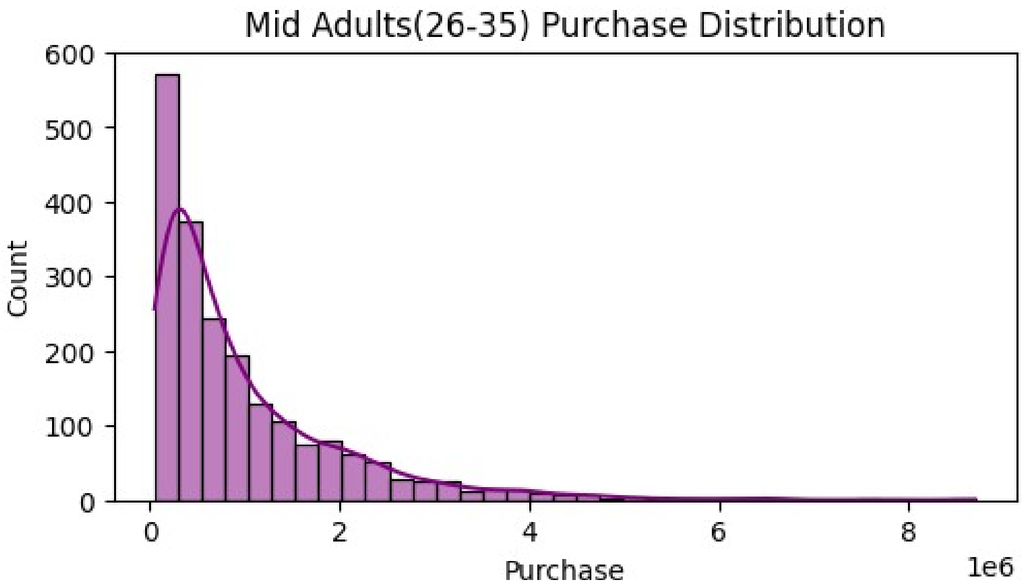
```
Purchase
689454      2
325558      2
227662      2
784192      2
203258      2
           ..
314423      1
314220      1
314108      1
313566      1
10536909    1
Name: count, Length: 5876, dtype: int64
     Age     Purchase
5  51-55  9534.808031
6    55+  9336.280459
3  36-45  9331.350695
2  26-35  9252.690633
4  46-50  9208.625697
1  18-25  9169.663606
0   0-17  8933.464640

Sample mean for Teens is 618849.4948
Sample Standard Deviation For Teens is 48754.9957
Sample mean for Young Adults is 854555.7647
Sample Standard Deviation For Young Adults is 28202.1553
Sample mean for Mid Aults is 989679.8572
Sample Standard Deviation For Mid Aults is 23057.2682
Sample mean for Adults is 879558.8968
Sample Standard Deviation For Adults is 31276.9445
Sample mean for Mid-Age is 792064.1816
Sample Standard Deviation For Mid-Age is 41074.0475
```

```
Sample mean for Late Mid-Age is 762893.457
Sample Standard Deviation For Late Mid-Age is 39409.004
Sample mean for Late Mid-Age is 762893.457
Sample Standard Deviation For Late Mid-Age is 39409.004
Sample mean for Senior Citizens is 539233.8955
Sample Standard Deviation For Senior Citizens is 35609.6065

With 90% confidence level Teen purchase Confidence level lies between
[613683.8323992885, 624051.7914539225]
With 90% confidence level Young Adults purchase Confidence level lies
between [853496.8337636532, 856229.4057124928]
With 95% confidence level Teen purchase Confidence level lies between
[612690.7193247761, 625044.904528435]
With 95% confidence level Young Adults purchase Confidence level lies
between [853235.0895847711, 856491.1498913749]
With 99% confidence level Teen purchase Confidence interval lies
between [610749.7361558096, 626985.8876974015]
With 99% confidence level Young Adults purchase Confidence interval
lies between [852723.525434803, 857002.714041343]

With 90% confidence level Mid Adults purchase Confidence level lies
between [988832.7960770902, 990485.8381167724]
With 90% confidence level Adults purchase Confidence level lies
between [878282.2004619493, 881049.2202749829]
With 95% confidence level Mid Adults purchase Confidence level lies
between [988674.4565568744, 990644.1776369881]
With 95% confidence level Adults purchase Confidence level lies
between [878017.1566342029, 881314.2641027293]
With 99% confidence level Mid Adults purchase Confidence interval lies
between [988364.9909465542, 990953.6432473083]
With 99% confidence level Adults purchase Confidence interval lies
between [877499.1435077048, 881832.2772292274]

With 90% confidence level Mid-Age purchase Confidence level lies
between [789670.1362918321, 795427.4267966802]
With 90% confidence level Late Mid-Age purchase Confidence level lies
between [760491.4550604789, 765910.3910933674]
With 95% confidence level Mid-Age purchase Confidence level lies
between [789118.6641608175, 795978.8989276948]
With 95% confidence level Late Mid-Age purchase Confidence level lies
between [759972.3928037849, 766429.4533500613]
With 99% confidence level Mid-Age purchase Confidence interval lies
between [788040.8431634663, 797056.719925046]
With 99% confidence level Late Mid-Age purchase Confidence interval
lies between [758957.9150673165, 767443.9310865297]

With 90% confidence level Late Mid-Age purchase Confidence level lies
between [760491.4550604789, 765910.3910933674]
With 90% confidence level Senior Citizens purchase Confidence level
lies between [536966.9694443106, 542427.5198030012]
```

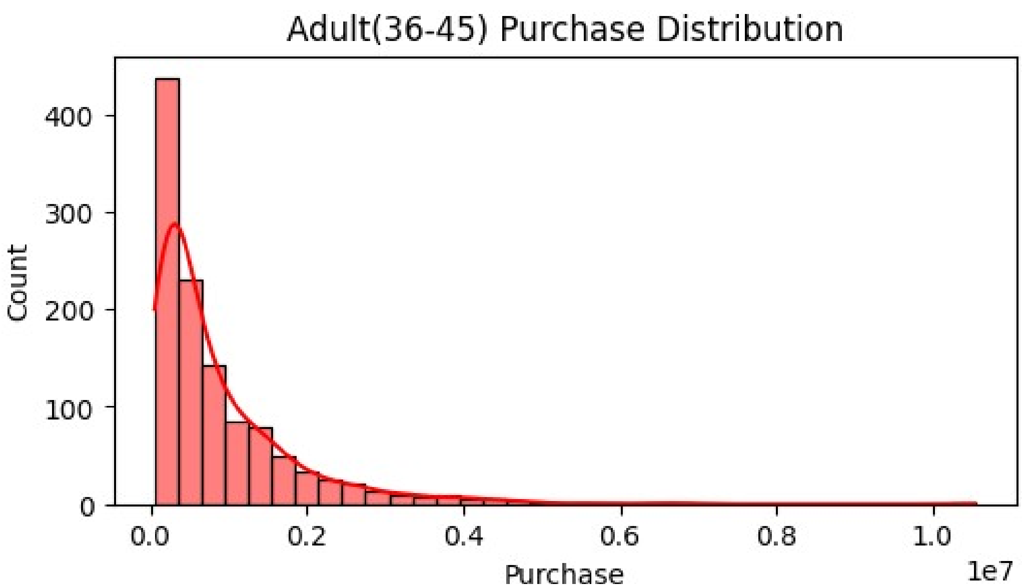
```
With 95% confidence level Late Mid-Age purchase Confidence level lies
between [759972.3928037849, 766429.4533500613]
With 95% confidence level Senior Citizens purchase Confidence level
lies between [536443.921086705, 542950.5681606068]
With 99% confidence level Late Mid-Age purchase Confidence interval
lies between [758957.9150673165, 767443.9310865297]
With 99% confidence level Senior Citizens purchase Confidence interval
lies between [535421.6527421194, 543972.8365051924]
```



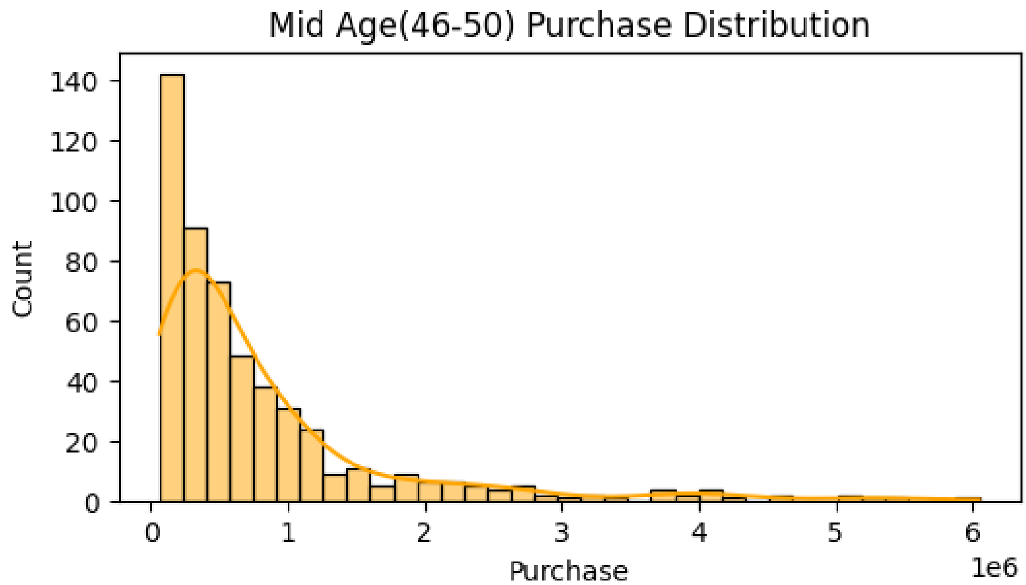




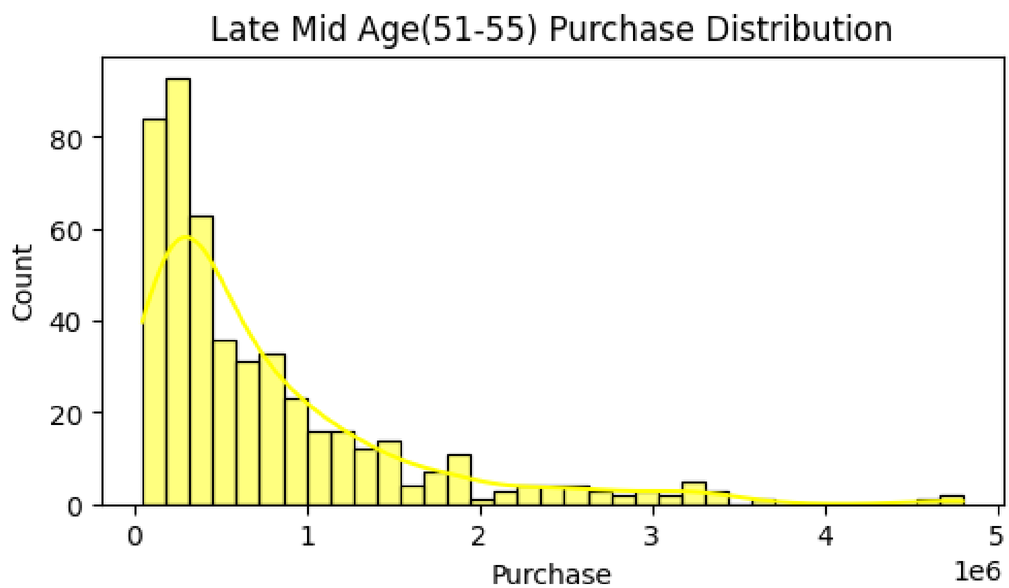
Mid Adults(26-35) Purchase Distribution
Count
Purchase
1e6
0
100
200
300
400
500
600
0
2
4
6
8



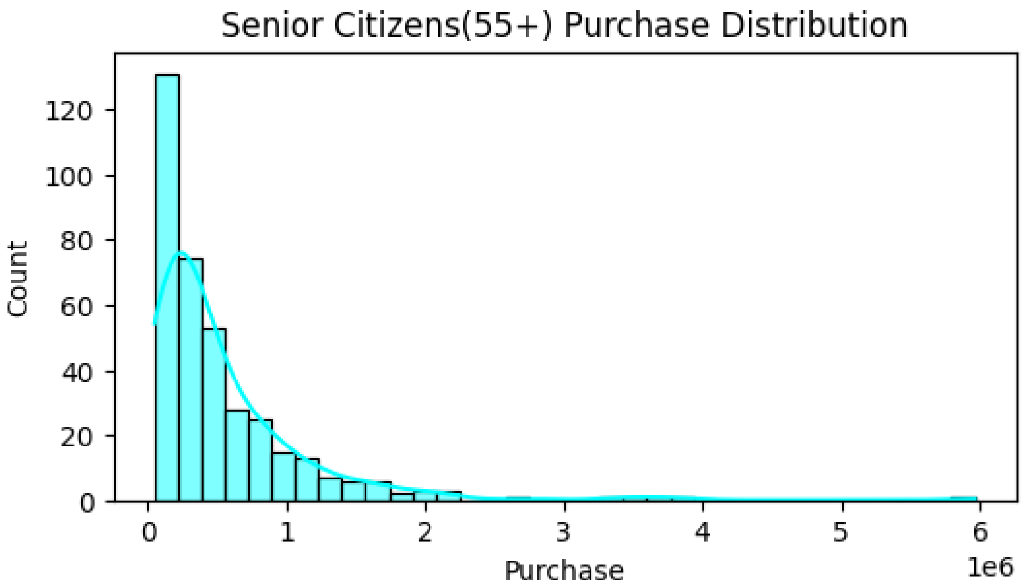
Adult(36-45) Purchase Distribution
Count
Purchase
1e7
0
100
200
300
400
0.0
0.2
0.4
0.6
0.8
1.0

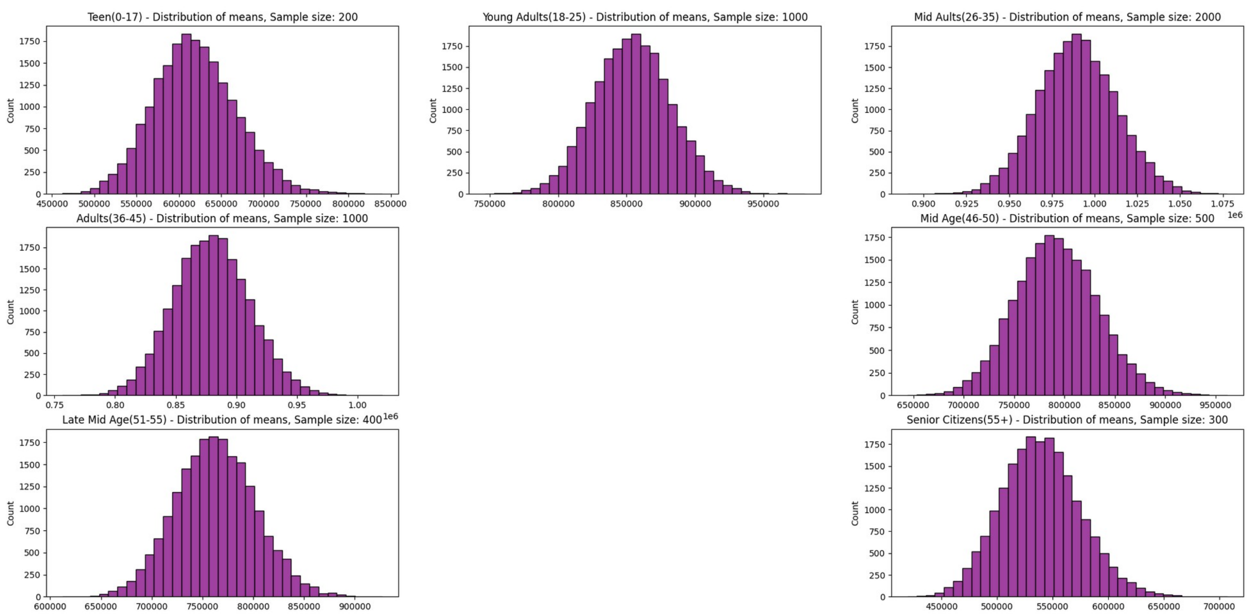


Mid Age(46-50) Purchase Distribution
Count
140
120
100
80
60
40
20
0
0
1
2
3
4
5
6
Purchase
1e6



Late Mid Age(51-55) Purchase Distribution
Count
80
60
40
20
0
0
1
2
3
4
5
Purchase
1e6





Insights and Recommendation

```
- From the above distribution graphs, mean and standard deviation
calculation, we have the following observations
    - The sample Purchase mean for the age group 18-45 lies between
854907.74 and 989731.32.
    - The sample Purchase Mean for teenagers and senior citizens is
less than any other category.
    - The sample purchase CI  for age group 18-45 with 90% CL lies
between 853496.83.74 and 990485.83
    - Similar observations can be seen for 95% and 99% Confidence
```

```
levels.
    - We have also observed that the mean confidence interval for each group does not overlap.
- Recommendation
    - Walmart need to focus on retaining customers from the range of 18 to 45 and bring new products suitable for
    customers in this age range
```

General Insights and Recommendations

```
- Insights
    - We have plotted many multivariate and bivariate plots between different categories vs. purchases.
    - We have seen Customers from City C buy more than other cities.
    - Customers who stay for only a year in a specific city purchase more than customers who have stayed longer in a specific city.
    - Most customers have purchased from product categories 1, 5, 8, 11, and 2.
    - Similar deduction can be found for occupation as well.
- Recommendations
    - Walmart should focus on the issues due to which fewer sales are happening in Cities A and B
    surveying to understand people's needs in those cities will help them grow their business.
    - The company must bring new products suitable for each occupation category.
    - The company should stock their inventory with 1, 5, 8, 11, and 2 product categories. In the meantime, they should
    also, brainstorm on how to increase sales of other product categories.
```