Mojeeb Alrahmaan Oshaibat : 2141987

Hatem Yousef Mohammad : 2132159

Malek Ghassan Ghalib : 2135729

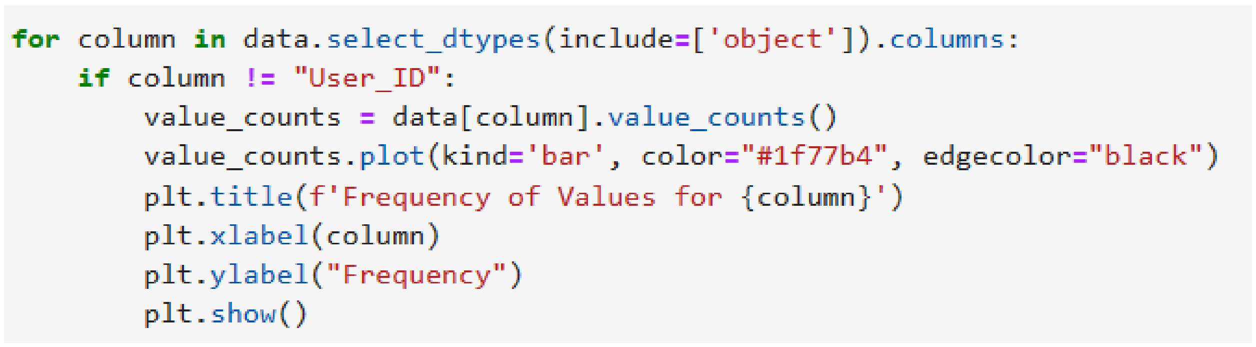
Data Mining Assignment

A1. Identify the type of each attribute (nominal,ordinal, interval or ratio).

	Attributes	Attribute Type
1	User_ID	Nominal
2	Gender	Nominal(binary(symmetric))
3	Age	Numeric(ratio)
4	Marital_Status	Nominal(binary(symmetric))
5	Website_Activity	Nominal(ordinal)
6	Browsed_Electronics_12Mo	Nominal((Asymmetric)binary)
7	Bought_Electronoics_12Mo	Nominal((Asymmetric)binary)
8	Bought_Digital_Media_18Mo	Nominal((Asymmetric)binary)
9	Bought_Digital_Books	Nominal((Asymmetric)binary)
10	Payment_Method	Nominal

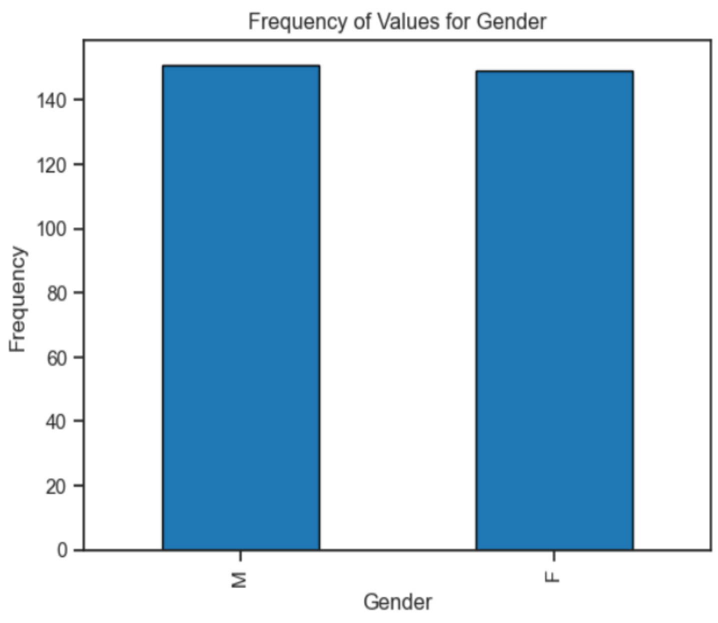
A2. In the next step we want to identify the values of the summarizing properties for each attribute including frequency, location and spread.

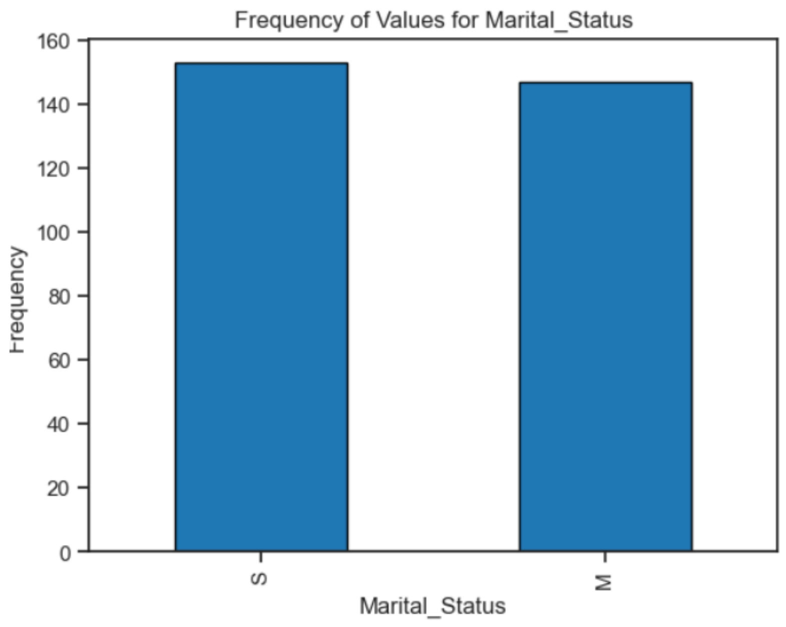
And we did it with the following code:

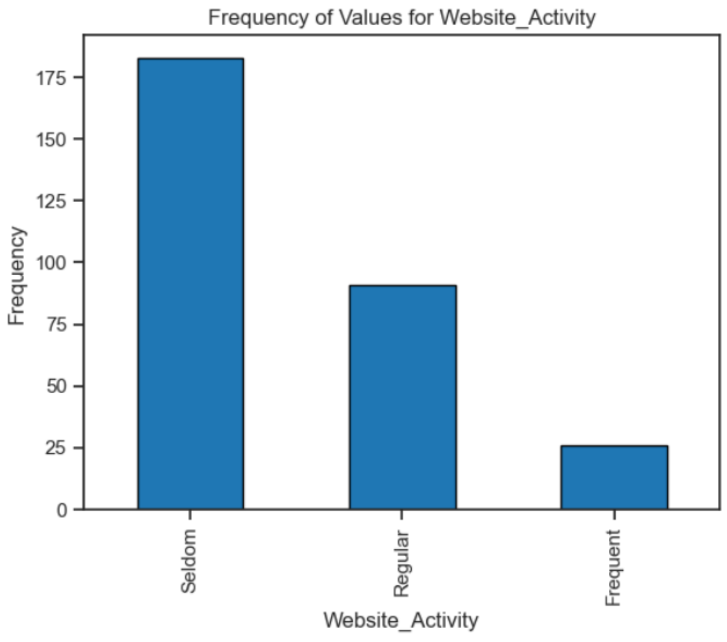


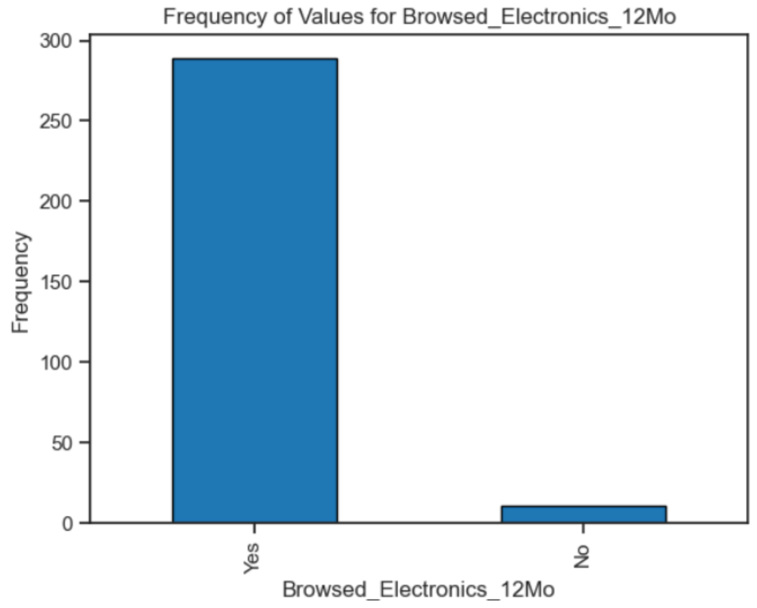
```
for column in data.select_dtypes(include=['object']).columns:
    if column != "User_ID":
        value_counts = data[column].value_counts()
        value_counts.plot(kind='bar', color="#1f77b4", edgecolor="black")
        plt.title(f'Frequency of Values for {column}')
        plt.xlabel(column)
        plt.ylabel("Frequency")
        plt.show()
```

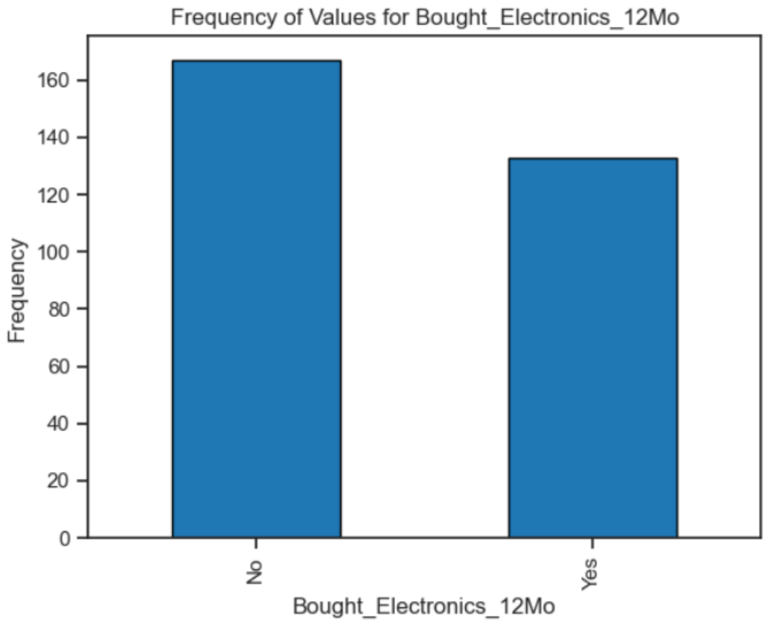
This code divides each attribute with its columns and it shows you the frequency of each column using plots like in the following slide:

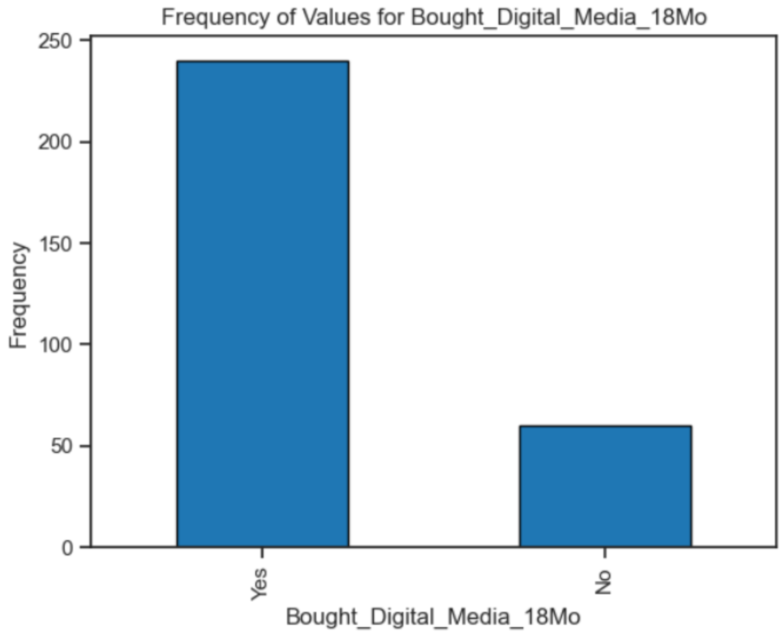

Frequency of Values for Gender
Frequency
0
20
40
60
80
100
120
140
M
F
Gender

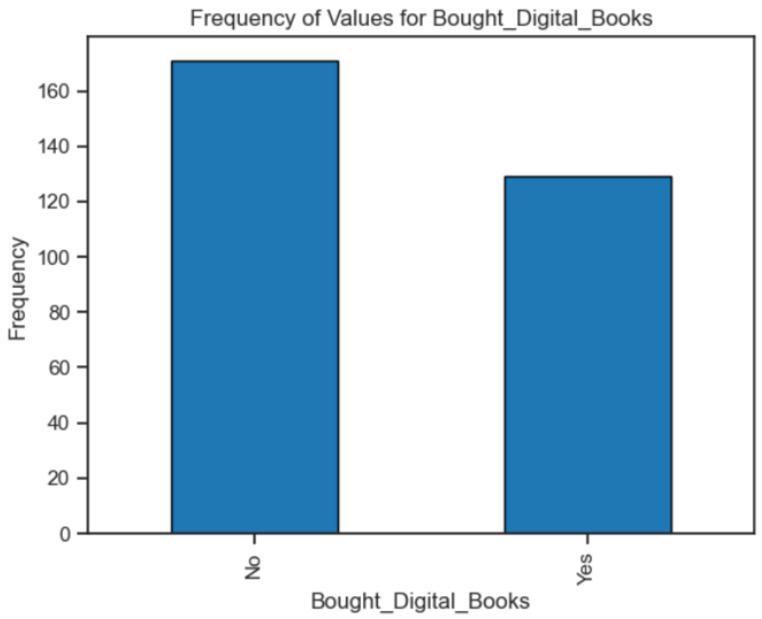

Frequency of Values for Marital_Status
Frequency
0
20
40
60
80
100
120
140
160
S
M
Marital_Status

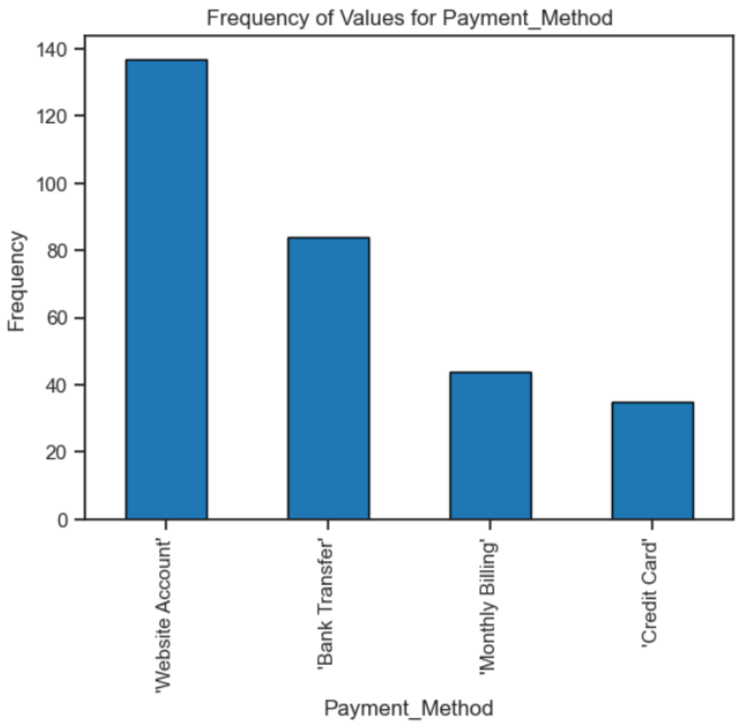

Frequency of Values for Website_Activity
Frequency
0
25
50
75
100
125
150
175
Seldom
Regular
Frequent
Website_Activity


Frequency of Values for Browsed_Electronics_12Mo
Frequency
0
50
100
150
200
250
300
Yes
No
Browsed_Electronics_12Mo


Frequency of Values for Bought_Electronics_12Mo
Frequency
0
20
40
60
80
100
120
140
160
No
Yes
Bought_Electronics_12Mo


Frequency of Values for Bought_Digital_Media_18Mo
Frequency
0
50
100
150
200
250
Yes
No
Bought_Digital_Media_18Mo

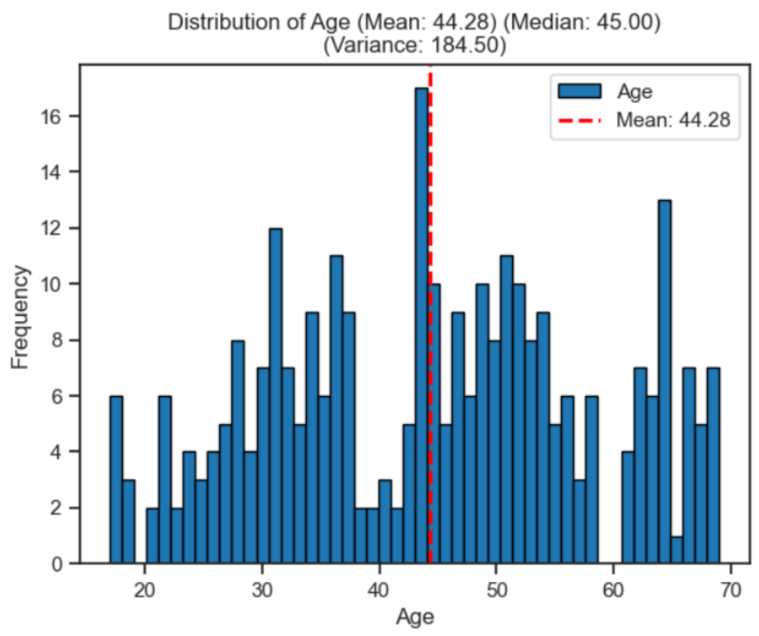

Frequency of Values for Bought_Digital_Books
Frequency
0
20
40
60
80
100
120
140
160
No
Yes
Bought_Digital_Books


Frequency of Values for Payment_Method
Frequency
0
20
40
60
80
100
120
140
'Website Account'
'Bank Transfer'
'Monthly Billing'
'Credit Card'
Payment_Method

Now, we needed to calculate the distribution, Measures of Central Tendency & Measures of Spread such like (Mean, Median, Variance and percentiles) and we were able to calculate these for the column Age only because it's the only Numeric data that exists in our csv file.

```
for column in data.select_dtypes(include=['int64', 'float64']).columns:
    if column != "User_ID":
        mean_value = data[column].mean()
        median_value = data[column].median()
        variance_value = data[column].var()
        percentiles_value = data[column].quantile([0.25, 0.5, 0.75])
        data[column].plot(kind='hist', bins=50, color="#1f77b4", edgecolor="black")
        plt.axvline(mean_value, color='red', linestyle='dashed', linewidth=2, label=f'Mean: {mean_value:.2f}')
        plt.title(f"Distribution of {column} (Mean: {mean_value:.2f}) (Median: {median_value:.2f})\n(Variance: {variance_value:.2f})")
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.legend()
        plt.show()
        print(percentiles_value)
```

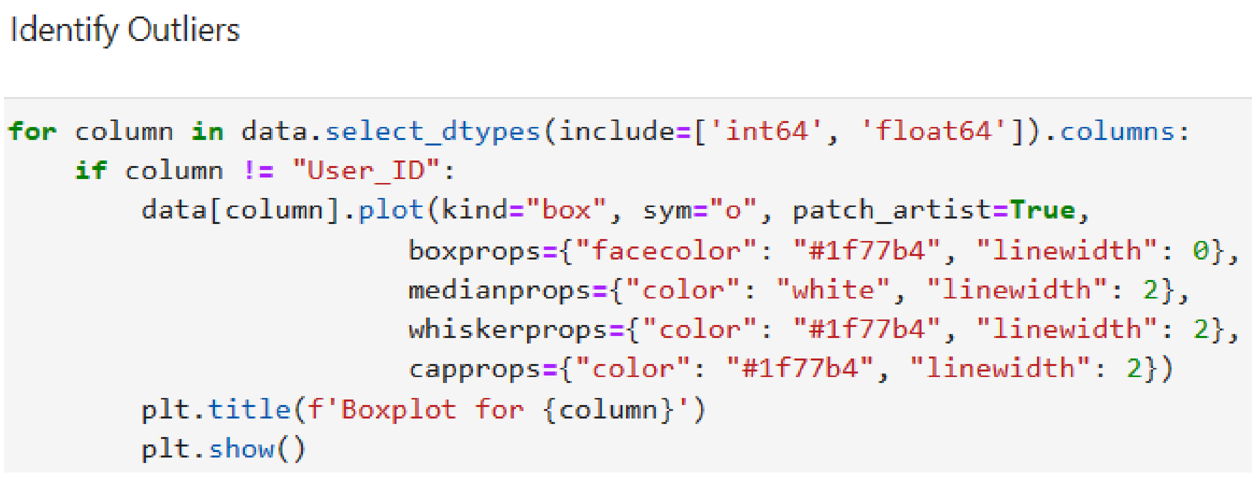
This code finds the mean of each numeric column , it also calculates (Median, Mean, Variance and percentiles) of each column , and at the end it build a histogram graph , and it draws a dashed red line showing where the Mean is. Below are some examples of what the code gives :



```
0.25    33.0
0.50    45.0
0.75    54.0
Name: Age, dtype: float64
```

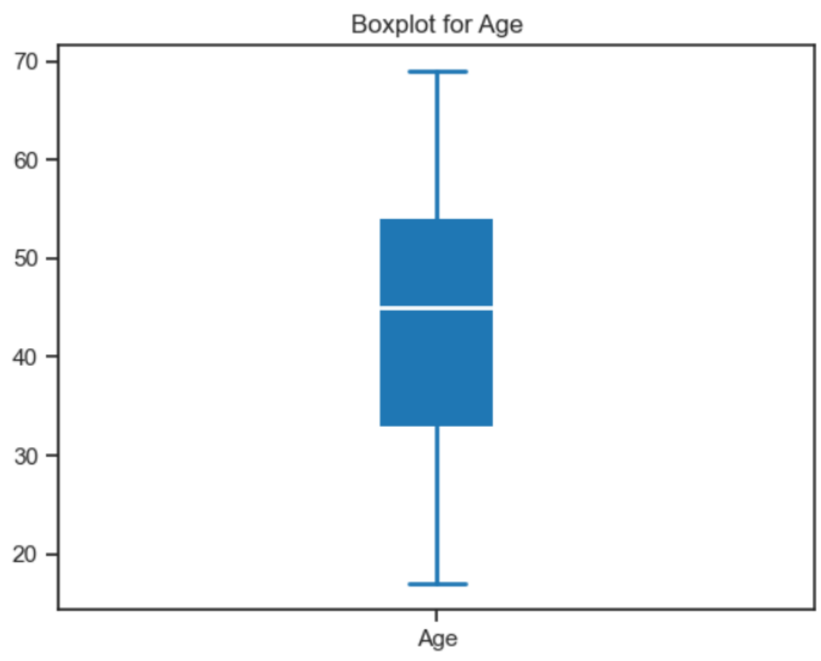
attribute	Mean	Median	Variance	Q1	Q2	Q3
Age	44.28	45	184.5	33.0	45	54

A3. in this task we had to find the Outliers for each numeric attribute but we noted before that the only numeric attribute we had was “Age” only, so we wrote the following code that helps us find the Outliers and shows it in a Boxplot to show us the Outliers.

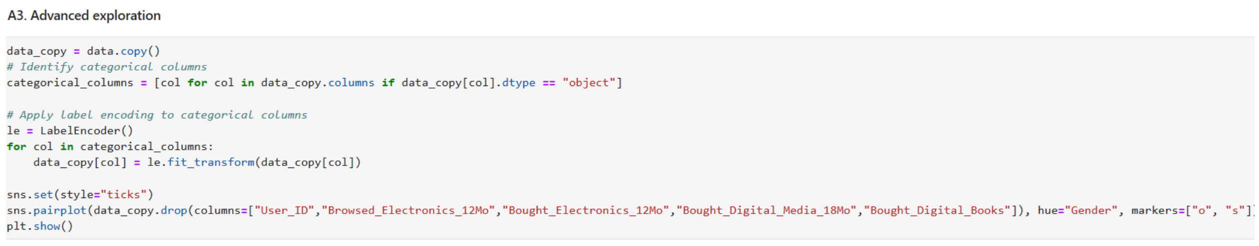


Identify Outliers

```python
for column in data.select_dtypes(include=['int64', 'float64']).columns:
    if column != "User_ID":
        data[column].plot(kind="box", sym="o", patch_artist=True,
                          boxprops={"facecolor": "#1f77b4", "linewidth": 0},
                          medianprops={"color": "white", "linewidth": 2},
                          whiskerprops={"color": "#1f77b4", "linewidth": 2},
                          capprops={"color": "#1f77b4", "linewidth": 2})
        plt.title(f'Boxplot for {column}')
        plt.show()
```



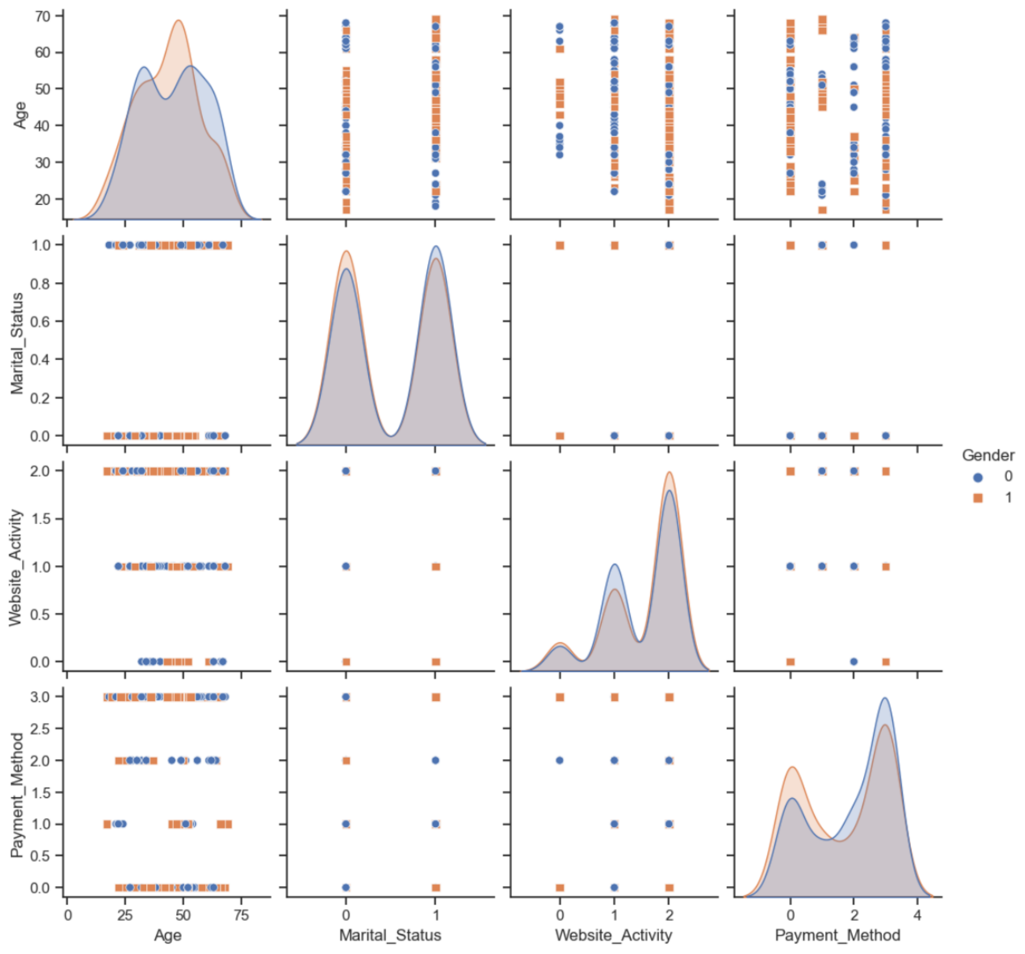
A3(2). We visualized the data in a scatterplot to derive meaningful insight



A3. Advanced exploration

```
data_copy = data.copy()
# Identify categorical columns
categorical_columns = [col for col in data_copy.columns if data_copy[col].dtype == "object"]

# Apply label encoding to categorical columns
le = LabelEncoder()
for col in categorical_columns:
    data_copy[col] = le.fit_transform(data_copy[col])

sns.set(style="ticks")
sns.pairplot(data_copy.drop(columns=["User_ID","Browsed_Electronics_12Mo","Bought_Electronics_12Mo","Bought_Digital_Media_18Mo","Bought_Digital_Books"]), hue="Gender", markers=["o", "s"]
plt.show()
```

This code prepares messy data for exploration by making a safe copy, finding categorial columns, and transforming them into numeric codes using a label encoding. Then, it visualizes connections between different variables using scatterplot matrix, highlighting how things might differ for Male and Female. Below we can see the scatter plot matrix it gives and we did drop some columns that does not provide meaningful insights into relationships between features, it might be excluded to simplify the visualization.

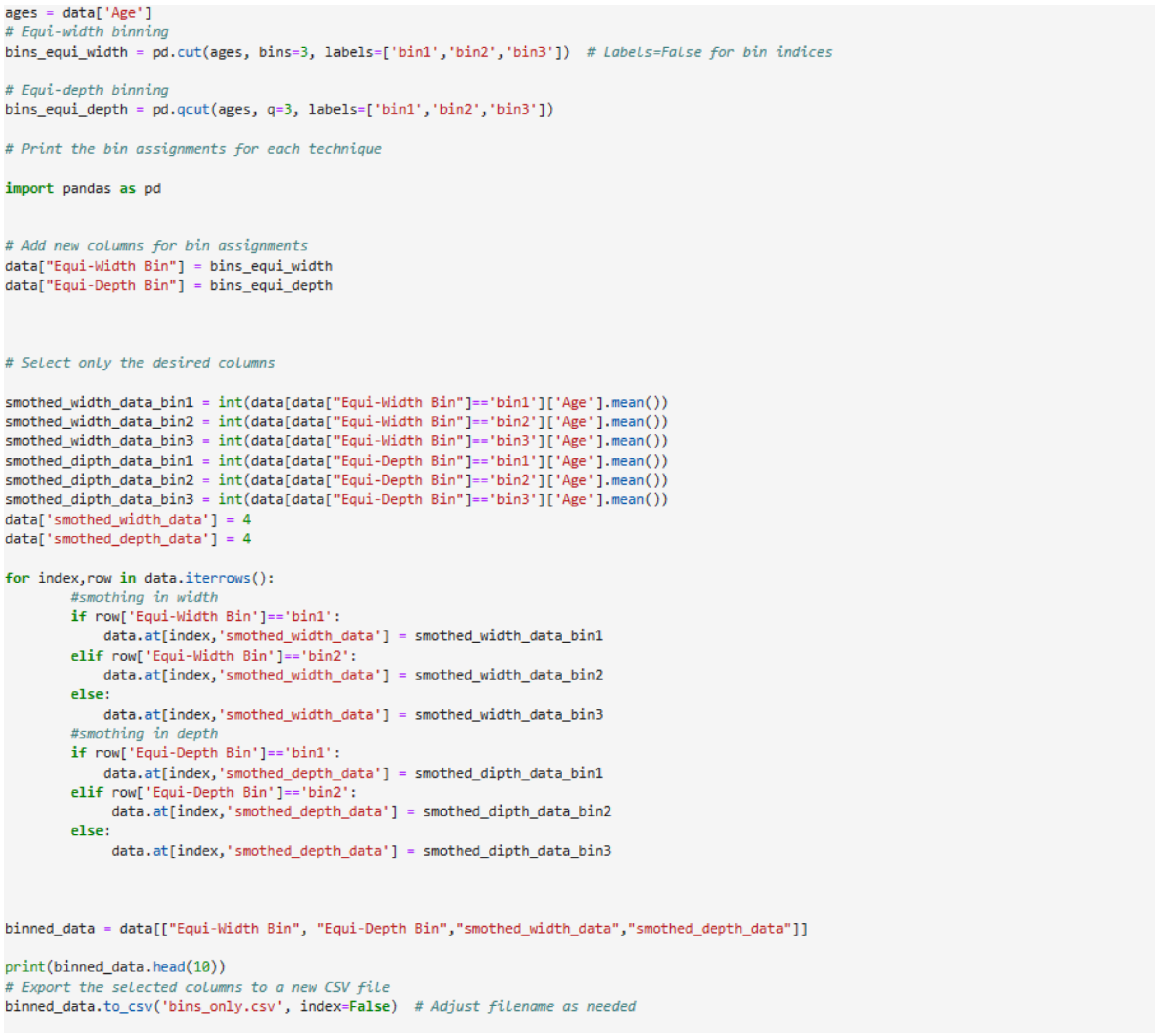


B. Data pre-pocessing.

B1. Using binning techniques to smooth the values of the “Age” attribute.

- **equal-width binning(3bins)**

- **equal-depth binning(3bins).**



```
ages = data['Age']
# Equi-width binning
bins_equi_width = pd.cut(ages, bins=3, labels=['bin1','bin2','bin3'])  # labels=False for bin indices

# Equi-depth binning
bins_equi_depth = pd.qcut(ages, q=3, labels=['bin1','bin2','bin3'])

# Print the bin assignments for each technique

import pandas as pd


# Add new columns for bin assignments
data["Equi-Width Bin"] = bins_equi_width
data["Equi-Depth Bin"] = bins_equi_depth



# Select only the desired columns

smothed_width_data_bin1 = int(data[data["Equi-Width Bin"]=='bin1']['Age'].mean())
smothed_width_data_bin2 = int(data[data["Equi-Width Bin"]=='bin2']['Age'].mean())
smothed_width_data_bin3 = int(data[data["Equi-Width Bin"]=='bin3']['Age'].mean())
smothed_dipth_data_bin1 = int(data[data["Equi-Depth Bin"]=='bin1']['Age'].mean())
smothed_dipth_data_bin2 = int(data[data["Equi-Depth Bin"]=='bin2']['Age'].mean())
smothed_dipth_data_bin3 = int(data[data["Equi-Depth Bin"]=='bin3']['Age'].mean())
data['smothed_width_data'] = 4
data['smothed_depth_data'] = 4

for index,row in data.iterrows():
        #smothing in width
        if row['Equi-Width Bin']=='bin1':
            data.at[index,'smothed_width_data'] = smothed_width_data_bin1
        elif row['Equi-Width Bin']=='bin2':
            data.at[index,'smothed_width_data'] = smothed_width_data_bin2
        else:
            data.at[index,'smothed_width_data'] = smothed_width_data_bin3
        #smothing in depth
        if row['Equi-Depth Bin']=='bin1':
            data.at[index,'smothed_depth_data'] = smothed_dipth_data_bin1
        elif row['Equi-Depth Bin']=='bin2':
             data.at[index,'smothed_depth_data'] = smothed_dipth_data_bin2
        else:
             data.at[index,'smothed_depth_data'] = smothed_dipth_data_bin3



binned_data = data[["Equi-Width Bin", "Equi-Depth Bin","smothed_width_data","smothed_depth_data"]]

print(binned_data.head(10))
# Export the selected columns to a new CSV file
binned_data.to_csv('bins_only.csv', index=False)  # Adjust filename as needed
```

In this code we are calculating the equal width and equal depth and we are smoothing each one after the calculation showing the results for each one before and after smoothing.

Here are the results after running the code we have showed in the previous slide

1. **Bins using Equal width**
2. **Bins using Equal depth**
3. **Equal width with smoothing**
4. **Equal depth with smoothing**

```
  Equi-Width Bin Equi-Depth Bin  smothed_width_data  smothed_depth_data
0           bin2           bin1                  43                  28
1           bin3           bin3                  59                  59
2           bin3           bin3                  59                  59
3           bin1           bin1                  27                  28
4           bin2           bin2                  43                  45
5           bin1           bin1                  27                  28
6           bin3           bin3                  59                  59
7           bin3           bin3                  59                  59
8           bin2           bin2                  43                  45
9           bin3           bin3                  59                  59
```

B2. In this step we are calculating the following on the attribute "Age":

1. min-max normalization to transform the values onto the range [0.0-1.0].

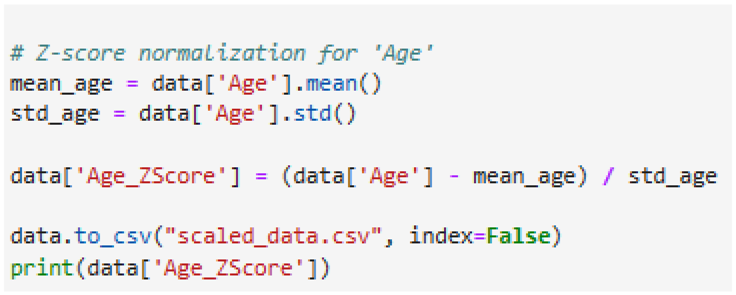
2. z-score normalization to transform the values.

1-Min-max normalization code:

```
# Get the minimum and maximum values of the "ages" column
min_age = data["Age"].min()
max_age = data["Age"].max()

# Scale the "ages" column using min-max scaling
data["Ages_scaled"] = (data["Age"] - min_age) / (max_age - min_age)

# Save the scaled data to a new CSV file (optional)
data.to_csv("scaled_data.csv", index=False)
print(data["Ages_scaled"])
```

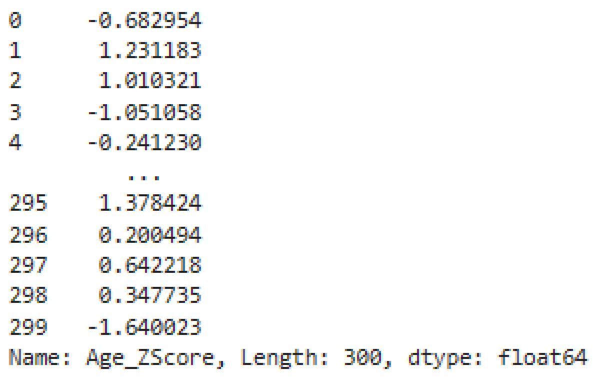
Results of the code:

```
0      0.346154
1      0.846154
2      0.788462
3      0.250000
4      0.461538
         ...
295    0.884615
296    0.576923
297    0.692308
298    0.615385
299    0.096154
Name: Ages_scaled, Length: 300, dtype: float64
```



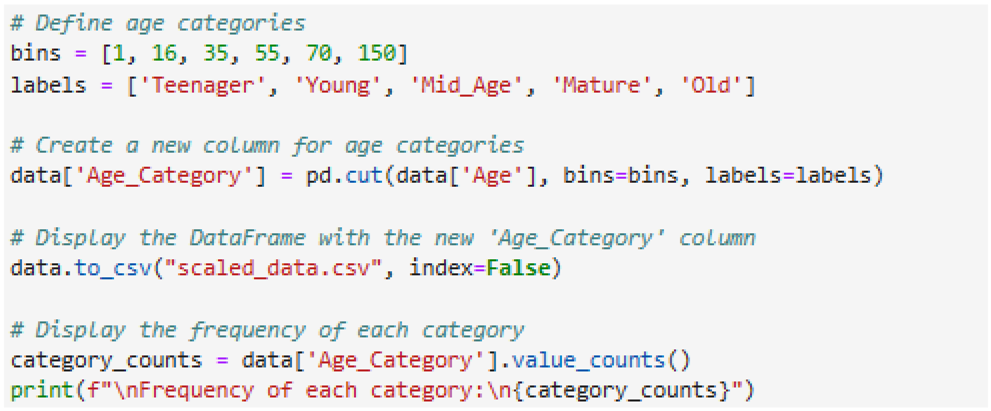
```python
# Z-score normalization for 'Age'
mean_age = data['Age'].mean()
std_age = data['Age'].std()

data['Age_ZScore'] = (data['Age'] - mean_age) / std_age

data.to_csv("scaled_data.csv", index=False)
print(data['Age_ZScore'])
```

Results of the code:



```
0     -0.682954
1      1.231183
2      1.010321
3     -1.051058
4     -0.241230
         ...
295    1.378424
296    0.200494
297    0.642218
298    0.347735
299   -1.640023
Name: Age_ZScore, Length: 300, dtype: float64
```

B3. In this step we are discretizing the "Age" attribute into categories



```python
# Define age categories
bins = [1, 16, 35, 55, 70, 150]
labels = ['Teenager', 'Young', 'Mid_Age', 'Mature', 'Old']

# Create a new column for age categories
data['Age_Category'] = pd.cut(data['Age'], bins=bins, labels=labels)

# Display the DataFrame with the new 'Age_Category' column
data.to_csv("scaled_data.csv", index=False)

# Display the frequency of each category
category_counts = data['Age_Category'].value_counts()
print(f"\nFrequency of each category:\n{category_counts}")
```

This code divides the continuous "Age" into five categories which are:

.teenager = 1-16 .Young=17-35 .Mid_Age = 36-55 .Mature = 56-70 .Old = 71+

So when we run the code we get the following results:

```
Frequency of each category:
Mid_Age     142
Young        93
Mature       65
Teenager      0
Old           0
Name: Age_Category, dtype: int64
```

according to our results we got 142 Middle age people, 93 Young people, 65 Mature people, Zero Teenager people and Zero Old people.

B4. In this step we are converting the attributes “Gender” into binary variables with values [0,1].

```
# Map 'Gender' to binary values
gender_mapping = {'M': 1, 'F': 0}
data['Gender_Binary'] = data['Gender'].map(gender_mapping)

data.to_csv("scaled_data.csv", index=False)

print(data[['Gender','Gender_Binary']])
```

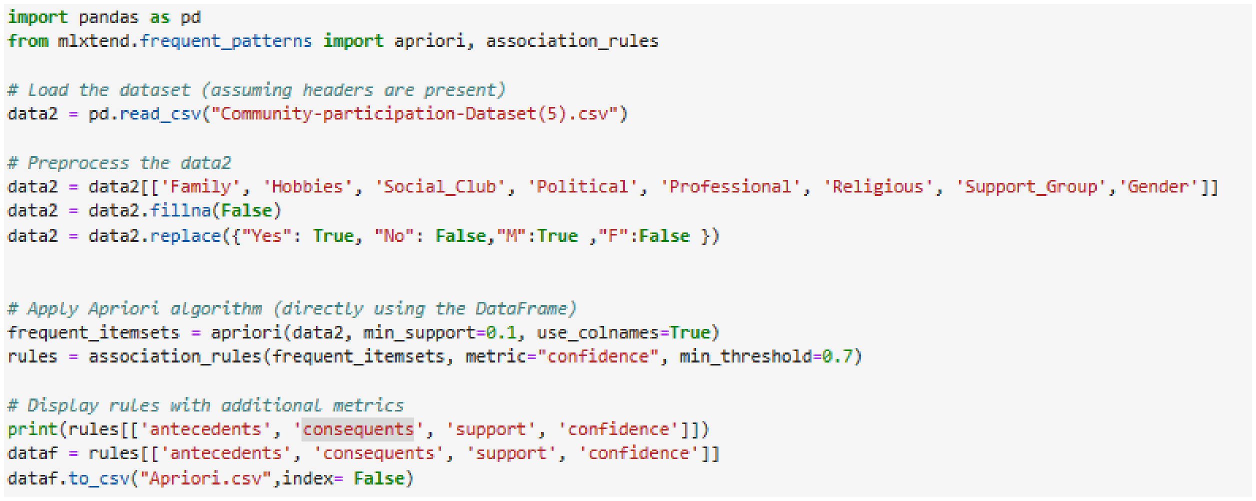
This code converts the categorical attribute “Gender” values into numeric values(binary).

The results after running the code:

```
    Gender  Gender_Binary
0        F              0
1        M              1
2        M              1
3        M              1
4        F              0
..     ...            ...
295      F              0
296      M              1
297      M              1
298      F              0
299      F              0

[300 rows x 2 columns]
```

C. Association Rules Mining

In this task we are going to create an association rule model to try to find linkages across types of community organization. to do this task we need to use The Association rule techniques to Extract and evaluate possible associations between the attributes by using this code.



```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Load the dataset (assuming headers are present)
data2 = pd.read_csv("Community-participation-Dataset(5).csv")

# Preprocess the data2
data2 = data2[['Family', 'Hobbies', 'Social_Club', 'Political', 'Professional', 'Religious', 'Support_Group','Gender']]
data2 = data2.fillna(False)
data2 = data2.replace({"Yes": True, "No": False,"M":True ,"F":False })


# Apply Apriori algorithm (directly using the DataFrame)
frequent_itemsets = apriori(data2, min_support=0.1, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# Display rules with additional metrics
print(rules[['antecedents', 'consequents', 'support', 'confidence']])
dataf = rules[['antecedents', 'consequents', 'support', 'confidence']]
dataf.to_csv("Apriori.csv",index= False)
```

1- we imported apriori and association_rules so we can apply the apriori algorithm and use the association rules then we read the csv file we are going to run the code on.

we preprocessed our data so we can select only the specified columns from the DataFrame.

3-we used fillna method to fill any missing value with a Boolean value “false” and we converted categorical variables to Boolean values if its “yes” then we convert it to **‘True’**, if it’s “No” then we convert it to ‘False’, if it’s “M” we convert it to **‘True’**, and if it’s “F” we convert it to ‘False’.

in the code line

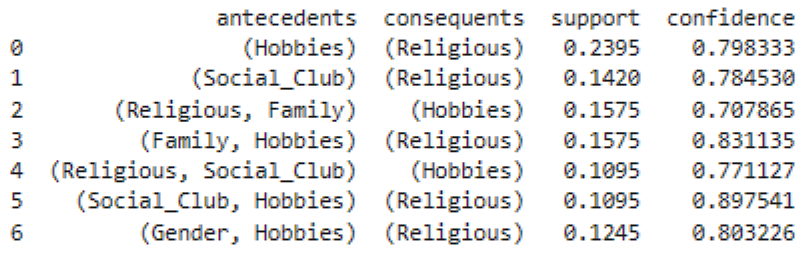
frequent_itemsets = apriori(data2, min_support=0.1, use_colnames=True)

we applied the ‘apriori’ algorithm to find the frequent itemset and in the second line

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7) the association_rules function generates association rules from the frequent itemsets. It takes the ‘frequent_itemsets’ generated by Apriori, specifies the metric to evaluate the rules (‘confidence’). The results are stored in the ‘rules’ variable. We print the itemset after that which are 'antecedents', 'consequents', 'support', and 'confidence'. Then we exported the results to the csv file named “Apriori.csv”.

note that we set the minimum support threshold of 0.1 and a minimum confidence threshold of 0.7.

after running the code you will be able to see the results here:



```
               antecedents  consequents  support  confidence
0                (Hobbies)  (Religious)   0.2395    0.798333
1            (Social_Club)  (Religious)   0.1420    0.784530
2       (Religious, Family)    (Hobbies)   0.1575    0.707865
3         (Family, Hobbies)  (Religious)   0.1575    0.831135
4  (Religious, Social_Club)    (Hobbies)   0.1095    0.771127
5    (Social_Club, Hobbies)  (Religious)   0.1095    0.897541
6         (Gender, Hobbies)  (Religious)   0.1245    0.803226
```

Or you can just enter the file “Apriori.csv” and you will be able to see the results there too.