Name	**Tonny K Podiyan**
Student ID	**21052135**
Email ID	**Tok0165@my.londonmet.ac.uk**
Module Code	**CC7182**
Module Title	**Programming for Data Analytics**
Date	**29th April 2023**

Introduction

The purpose of this report is to present the results of the analysis of the marketing campaign dataset provided for the CC7182 Programming for Analytics coursework. The report provides an overview of the dataset, outlines the steps taken to clean and prepare the data, presents summary statistics and correlation analysis, explores the data using histograms and scatter plots, and builds predictive models to predict whether a customer will take the AFFINITY_CARD in the marketing campaign.

Data Description

The dataset provided for this coursework is a marketing campaign dataset from a retail company. The dataset contains 1500 customer records, each consisting of 19 variables. The variables include socio-demographic and product ownership information, with AFFINITY_CARD being the target variable.

Data Preparation

To prepare the data for analysis, we first produced a meta data table showing the characteristics of each attribute. The table was used to identify missing or erroneous data, and the necessary steps were taken to clean the data using Python programming. This included removing records with missing values or errors and transforming some variables into ordinal numbers. The in-depth details of the cleaning methods are given below:

1. Describing meta data to show characteristics of each attribute:

Attribute	Description	Data Type	Example
CUST_ID	Unique identifier for each customer	Numeric	101501
CUST_GENDER	Gender of the customer	Categorical (Binary)	M, F
AGE	Age of the customer	Numeric	41
CUST_MARITAL_STATUS	Marital status of the customer	Categorical	Married, Divorc
COUNTRY_NAME	Country of residence of the customer	Categorical	Spain
CUST_INCOME_LEVEL	Income level of the customer	Categorical (Ordinal)	J: 190,000 - 249,999
EDUCATION	Education level of the customer	Categorical	Masters
OCCUPATION	Occupation of the customer	Categorical	Sales, Other
HOUSEHOLD_SIZE	Size of the customer's household	Numeric	2
YRS_RESIDENCE	Number of years the customer has been living in their current residence	Numeric	4
AFFINITY_CARD	Target variable: Whether the customer holds a affinity card or not	Categorical (Binary)	1 (Yes), 0 (No)

BULK_PACK_DISKETTES	Whether the customer owns bulk pack diskettes or not	Categorical (Binary)	1 (Yes), 0 (No)
FLAT_PANEL_MONITOR	Whether the customer owns a flat panel monitor or not	Categorical (Binary)	1 (Yes), 0 (No)
HOME_THEATER_PACKAGE	Whether the customer owns a home theatre package or not	Categorical (Binary)	1 (Yes), 0 (No)
BOOKKEEPING_APPLICATION	Whether the customer owns a bookkeeping application or not	Categorical (Binary)	1 (Yes), 0 (No)
PRINTER_SUPPLIES	Whether the customer buys printer supplies or not	Categorical (Binary)	1 (Yes), 0 (No)
Y_BOX_GAMES	Whether the customer buys Y-box games or not	Categorical (Binary)	1 (Yes), 0 (No)
OS_DOC_SET_KANJI	Whether the customer owns an operating system documentation set in Kanji or not	Categorical (Binary)	1 (Yes), 0 (No)
COMMENTS	Any comments or additional information related to the customer	Text	"Very satisfied with the product."

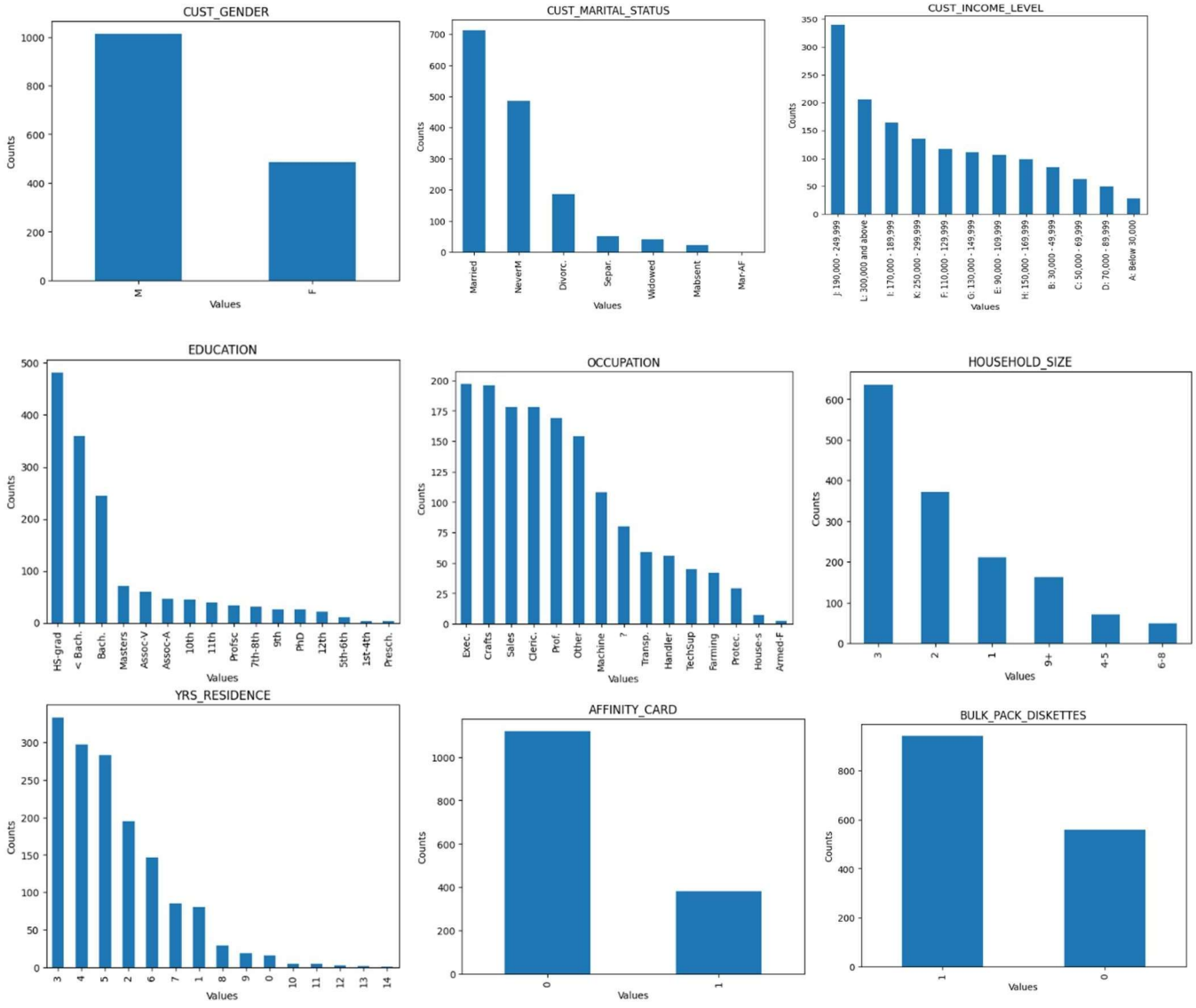
The above metadata table was built based on the information obtained by using the following code:

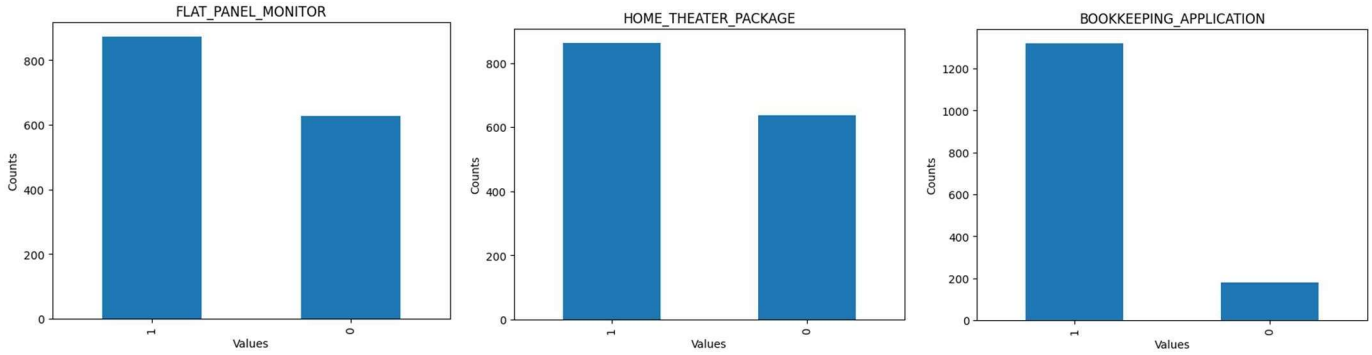
```
# create metadata
metadata = pd.DataFrame({
    'Column': df.columns,
    'Data Type': df.dtypes,
    'Number of Null Values': df.isnull().sum(),
    'Number of Unique Values': df.nunique()
})
metadata
```

	Column	Data Type	Number of Null Values	Number of Unique Values
AGE	AGE	int64	0	66
CUST_MARITAL_STATUS	CUST_MARITAL_STATUS	int64	0	6
COUNTRY_NAME	COUNTRY_NAME	int64	0	19
OCCUPATION	OCCUPATION	int64	0	15
YRS_RESIDENCE	YRS_RESIDENCE	int64	0	15
AFFINITY_CARD	AFFINITY_CARD	int64	0	2
HOME_THEATER_PACKAGE	HOME_THEATER_PACKAGE	int64	0	2
BOOKKEEPING_APPLICATION	BOOKKEEPING_APPLICATION	int64	0	2
CUST_GENDER_BIN	CUST_GENDER_BIN	int64	0	2
COUNTRY_RANK	COUNTRY_RANK	int64	0	19
EDUCATION_ORD	EDUCATION_ORD	int64	0	14
CUST_INCOME_LEVEL_ORD	CUST_INCOME_LEVEL_ORD	int64	0	12
HOUSEHOLD_SIZE_ORD	HOUSEHOLD_SIZE_ORD	int64	0	6

```
for col in df.columns:
    if col in ['CUST_ID', 'AGE', 'COMMENTS']:
        continue
    fig, ax = plt.subplots()
    df[col].value_counts().plot(kind='bar', ax=ax)
    ax.set_title(col)
    ax.set_xlabel('Values')
    ax.set_ylabel('Counts')
    plt.show()
```

We also visualized all the attributes using a bar plot.





Count of the Individual Values of each attribute was determined using the below code:

```
for col in df.columns:
 if col in ['CUST_ID','AGE','COMMENTS']:
  continue
 print(f"{col}:")
 print(df[col].value_counts())
 print()
```

```
CUST_GENDER:
M    1014
F     486
Name: CUST_GENDER, dtype: int64

CUST_MARITAL_STATUS:
Married    712
NeverM     485
Divorc.    187
Separ.      52
Widowed     40
Mabsent     23
Mar-AF       1
Name: CUST_MARITAL_STATUS, dtype: int64

COUNTRY_NAME:
United States of America    1344
Argentina                     46
Italy                         37
Brazil                        14
Canada                         9
Germany                        8
Poland                         7
United Kingdom                 6
Denmark                        5
Saudi Arabia                   5
China                          4
Singapore                      4
New Zealand                    3
Japan                          2
Australia                      2
South Africa                   1
France                         1
Turkey                         1
Spain                          1
Name: COUNTRY_NAME, dtype: int64
```

```
CUST_INCOME_LEVEL:
J: 190,000 - 249,999    339
L: 300,000 and above    205
I: 170,000 - 189,999    164
K: 250,000 - 299,999    135
F: 110,000 - 129,999    117
G: 130,000 - 149,999    111
E: 90,000 - 109,999     106
H: 150,000 - 169,999     99
B: 30,000 - 49,999       84
C: 50,000 - 69,999       63
D: 70,000 - 89,999       49
A: Below 30,000          28
Name: CUST_INCOME_LEVEL, dtype: int64

EDUCATION:
HS-grad    482
< Bach.    359
Bach.      245
Masters     70
Assoc-V     60
Assoc-A     46
10th        44
11th        39
Profsc      34
7th-8th     31
9th         26
PhD         25
12th        22
5th-6th     11
1st-4th      3
Presch.      3
Name: EDUCATION, dtype: int64
```

```
AFFINITY_CARD:
0   1120
1    380
Name: AFFINITY_CARD, dtype: int64

BULK_PACK_DISKETTES:
1   942
0   558
Name: BULK_PACK_DISKETTES, dtype: int64

FLAT_PANEL_MONITOR:
1   873
0   627
Name: FLAT_PANEL_MONITOR, dtype: int64

HOME_THEATER_PACKAGE:
1   863
0   637
Name: HOME_THEATER_PACKAGE, dtype: int64

BOOKKEEPING_APPLICATION:
1   1321
0    179
Name: BOOKKEEPING_APPLICATION, dtype: int64

PRINTER_SUPPLIES:
1   1500
Name: PRINTER_SUPPLIES, dtype: int64

Y_BOX_GAMES:
0   1070
1    430
Name: Y_BOX_GAMES, dtype: int64

OS_DOC_SET_KANJI:
0   1497
1      3
Name: OS_DOC_SET_KANJI, dtype: int64
```

```
HOUSEHOLD_SIZE:
3     635
2     371
1     212
9+    163
4-5    71
6-8    48
Name: HOUSEHOLD_SIZE, dtype: int64

YRS_RESIDENCE:
3    333
4    297
5    283
2    195
6    147
7     85
1     80
8     29
9     19
0     16
10     5
11     5
12     3
13     2
14     1
Name: YRS_RESIDENCE, dtype: int64

AFFINITY_CARD:
```

```
OCCUPATION:
Exec.      197
Crafts     196
Sales      178
Cleric.    178
Prof.      169
Other      154
Machine    108
?           80
Transp.     59
Handler     56
TechSup     45
Farming     42
Protec.     29
House-s      7
Armed-F      2
Name: OCCUPATION, dtype: int64

HOUSEHOLD_SIZE:
3     635
2     371
1     212
9+    163
4-5    71
6-8    48
Name: HOUSEHOLD_SIZE, dtype: int64
```

2. Describing missing or error data of each attribute based on above analysis.

- There are missing values in the 'COMMENTS' column as it has a non-null count of 1427, whereas all the other columns have a non-null count of 1500 (73 null values).
- The HOUSEHOLD_SIZE column in the dataframe is currently stored as an object datatype, whereas it would be preferred as an integer datatype for further analysis. Also, data is stored in the form of both individual numbers as well as slab/ ranges. It would be best to transform them to ordinal values for better analysis.
- In the OCCUPATION column we have 80 unrecorded occupations which are denoted using '?'
- In the CUST_MARITAL_STATUS column there is one value as Mar-AF , considering that it's just a single entry in 1500 data points , we are assuming that it's been wrongly recorded so we would be dropping this entry.
- In the PRINTER_SUPPLIES column the only recorded value is 1
- In the CUST_INCOME_LEVEL column the range for most values is defined using the '-' separator for e.g., J: 190,000 - 249,999 , however there are two values which do not follow this format L: 300,000 and above , Below 30,000
- The dataset exhibits a significant degree of geographical bias, as out of the 1500 records, a predominant majority of 1344 records originate from the United States of America.
- The datapoints in the EDUCATION column need to be generalized for better understanding and analysis.

3. Data Cleaning for further analysis.

To remove variables with little or no influence we find the correlation of other variables with our target variable 'AFFINITY_CARD' for that we use the below code.

```
# Checking correlation with target variables to drop attributes
corr = df.corr()['AFFINITY_CARD'].sort_values(ascending=False)
print(corr)
```

```
AFFINITY_CARD           1.000000
YRS_RESIDENCE           0.342691
HOME_THEATER_PACKAGE    0.283358
AGE                     0.246711
CUST_GENDER_BIN         0.226390
BOOKKEEPING_APPLICATION 0.162404
BULK_PACK_DISKETTES     -0.017887
CUST_ID                 -0.025969
OS_DOC_SET_KANJI        -0.026075
FLAT_PANEL_MONITOR      -0.028467
Y_BOX_GAMES             -0.281121
PRINTER_SUPPLIES        NaN
Name: AFFINITY_CARD, dtype: float64
```

As per the correlation values we have dropped the below columns :

- 'CUST_ID'
- 'BULK_PACK_DISKETTES'
- 'OS_DOC_SET_KANJI',
- 'FLAT_PANEL_MONITOR'
- 'Y_BOX_GAMES'
- 'PRINTER_SUPPLIES'

```
[70] # Dropping specific column due to very low correlation to target variable AFFINITY_CARD
     df.drop(['CUST_ID','COMMENTS','BULK_PACK_DISKETTES',
     'OS_DOC_SET_KANJI',
     'FLAT_PANEL_MONITOR',
     'Y_BOX_GAMES',
     'PRINTER_SUPPLIES' ], axis=1, inplace=True)
```

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 19 columns):
 #  Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0  CUST_ID               1500 non-null   int64
 1  AGE                   1500 non-null   int64
 2  CUST_MARITAL_STATUS   1500 non-null   object
 3  COUNTRY_NAME          1500 non-null   object
 4  CUST_INCOME_LEVEL     1500 non-null   object
 5  EDUCATION             1500 non-null   object
 6  OCCUPATION            1500 non-null   object
 7  HOUSEHOLD_SIZE        1500 non-null   object
 8  YRS_RESIDENCE         1500 non-null   int64
 9  AFFINITY_CARD         1500 non-null   int64
 10 BULK_PACK_DISKETTES   1500 non-null   int64
 11 FLAT_PANEL_MONITOR    1500 non-null   int64
 12 HOME_THEATER_PACKAGE  1500 non-null   int64
 13 BOOKKEEPING_APPLICATION 1500 non-null  int64
 14 PRINTER_SUPPLIES      1500 non-null   int64
 15 Y_BOX_GAMES           1500 non-null   int64
 16 OS_DOC_SET_KANJI      1500 non-null   int64
 17 COMMENTS              1427 non-null   object
 18 CUST_GENDER_BIN       1500 non-null   int64
dtypes: int64(12), object(7)
memory usage: 222.8+ KB
```

```
[198] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 12 columns):
 #  Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0  AGE                   1500 non-null   int64
 1  CUST_MARITAL_STATUS   1500 non-null   object
 2  COUNTRY_NAME          1500 non-null   object
 3  CUST_INCOME_LEVEL     1500 non-null   object
 4  EDUCATION             1500 non-null   object
 5  OCCUPATION            1500 non-null   object
 6  HOUSEHOLD_SIZE        1500 non-null   object
 7  YRS_RESIDENCE         1500 non-null   int64
 8  AFFINITY_CARD         1500 non-null   int64
 9  HOME_THEATER_PACKAGE  1500 non-null   int64
 10 BOOKKEEPING_APPLICATION 1500 non-null  int64
 11 CUST_GENDER_BIN       1500 non-null   int64
dtypes: int64(6), object(6)
memory usage: 140.8+ KB
```

We have also dropped the COMMENTS column as we would require dedicated Text mining tools for detailed analysis. We have dropped the necessary columns using the below code:

In the CUST_MARITAL_STATUS there was one value recorded as 'Mar-AF' considering that it's just a single entry in 1500 data points, we are assuming that it's been wrongly recorded so we are dropping this entry.

```
df['CUST_MARITAL_STATUS'] = df['CUST_MARITAL_STATUS'].replace('Mar-AF', np.nan)
df=df.dropna()
```

We also did the following transformations:

- CUST_GENDER into binary F - 0, M -1

```
# Normalising Gender Column
df['CUST_GENDER_BIN'] = df['CUST_GENDER'].replace({'F': 0, 'M': 1})
df.drop('CUST_GENDER',axis=1,inplace=True)
```

- COUNTRY_NAME into ordinal number based on their occurrence in the data set in descending order.

```
# Normalizing Country Names
# Get the count of each unique value in the COUNTRY_NAME column
country_counts = df['COUNTRY_NAME'].value_counts()
# Create a dictionary mapping each country to its rank based on the occurrence count
country_rank = {country: rank for rank, country in enumerate(country_counts.index, 1)}
# Replace the country names with their corresponding rank based on the occurrence count
df['COUNTRY_RANK'] = df['COUNTRY_NAME'].map(country_rank)
```

- CUST_INCOME_LEVEL into ordinal numbers 1 - 12 accordingly.

```
# Define a dictionary mapping the income level codes to their corresponding ranges
income_dict = {'A: Below 30,000': 1,
               'B: 30,000 - 49,999': 2,
               'C: 50,000 - 69,999': 3,
               'D: 70,000 - 89,999': 4,
               'E: 90,000 - 109,999': 5,
               'F: 110,000 - 129,999': 6,
               'G: 130,000 - 149,999': 7,
               'H: 150,000 - 169,999': 8,
               'I: 170,000 - 189,999': 9,
               'J: 190,000 - 249,999': 10,
               'K: 250,000 - 299,999': 11,
               'L: 300,000 and above': 12}

df['CUST_INCOME_LEVEL_ORD'] = df['CUST_INCOME_LEVEL'].map(income_dict)
df.drop('CUST_INCOME_LEVEL',axis=1,inplace=True)
```

- OCCUPATION into Normalized form

```
# Map the 'OCCUPATION' column using the occupation_dict

occupation_dict = {
   '?': 'Not Recorded',
   'Armed-F':'Army',
   'Cleric.':'Clerk',
   'Crafts': 'Craftsman',
   'Exec.': 'Executive',
   'Farming': 'Farmer',
   'Handler': 'Handler',
   'House-s': 'House Service',
   'Machine': 'Machine Operator',
   'Other': 'Other',
   'Prof.': 'Professor',
   'Protec.': 'Security',
   'Sales': 'Sales',
   'TechSup': 'Tech Support',
   'Transp.': 'Transportation'}


df['OCCUPATION'] = df['OCCUPATION'].map(occupation_dict)
```

- EDUCATION into ordinal numbers based on USA education level.

```
# Normalizing remaining Attributes
# Define a dictionary mapping the original values to their corresponding ordinal numbers
education_levels = {
   'Presch.': 1,
   '1st-4th': 2,
   '5th-6th': 3,
   '7th-8th': 4,
   '9th': 5,
   '< Bach.': 6,
   'Assoc-A': 7,
   'Assoc-V': 8,
   'Bach.': 9,
   'HS-grad': 10,
   '10th': 10,
   '11th': 11,
   '12th': 12,
   'Profsc': 12,
   'Masters': 13,
   'PhD': 14
}


# Map the 'Education' column using the education_dict
df['EDUCATION_ORD'] = df['EDUCATION'].map(education_levels)
df.drop('EDUCATION',axis=1,inplace=True)
```

- HOUSEHOLD_SIZE into ordinal numbers based on number of rooms.

```
# Define a dictionary mapping the household sizes  to their corresponding ranges
size_dict = {'1': 1, '2': 2, '3': 3, '4-5': 4, '6-8': 5, '9+': 6}
# Map the 'HOUSEHOLD_SIZE' column using the size_dict_dict
df['HOUSEHOLD_SIZE_ORD'] = df['HOUSEHOLD_SIZE'].map(size_dict)
df.drop('HOUSEHOLD_SIZE',axis=1,inplace=True)
```

- CUST_MARITAL_STATUS into Normalised form

```
marital_status_dict={'Divorc.':'Divorced', 'Mabsent':'Not Recorded', 'Married':'Married','NeverM':'Unmarried', 'Separ.':'Separated','Widowed':'Widowed'}
# Separated means a couple is living apart but is not legally divorced, while divorced means the marriage has been legally terminated.
df['CUST_MARITAL_STATUS'] = df['CUST_MARITAL_STATUS'].map(marital_status_dict)
```

Before above Transformations:

	CUST_ID	CUST_GENDER	AGE	CUST_MARITAL_STATUS	COUNTRY_NAME	CUST_INCOME_LEVEL	EDUCATION	OCCUPATION	HOUSEHOLD_SIZE	YRS_RESIDENCE	AFFINITY_CARD	BULK_PACK_DISKETTES	FLAT_PANEL_MONITOR	HOME_THEATER_PACKAGE	BOOKKEEPING_APPLICA
0	101501	F	41	NeverM	United States of America	J: 190,000 - 249,999	Masters	Prof.	2	4	0	1	1	1	
1	101502	M	27	NeverM	United States of America	I: 170,000 - 189,999	Bach.	Sales	2	3	0	1	1	0	
2	101503	F	20	NeverM	United States of America	H: 150,000 - 169,999	HS-grad	Cleric.	2	2	0	1	0	0	

After transformation:

	AGE	CUST_MARITAL_STATUS	COUNTRY_NAME	OCCUPATION	YRS_RESIDENCE	AFFINITY_CARD	HOME_THEATER_PACKAGE	BOOKKEEPING_APPLICATION	CUST_GENDER_BIN	COUNTRY_RANK	EDUCATION_ORD	CUST_INCOME_LEVEL_ORD	HOUSEHOLD_SIZE_ORD
0	41	Unmarried	United States of America	Professor	4	0	1	1	0	1	13	10	2
1	27	Unmarried	United States of America	Sales	3	0	0	1	1	1	9	9	2
2	20	Unmarried	United States of America	Clerk	2	0	0	1	0	1	10	8	2
3	45	Married	United States of America	Executive	5	1	1	1	1	1	9	2	3
4	34	Unmarried	United States of America	Sales	5	1	0	1	1	1	13	11	6

- **Data Analysis**

After cleaning and preparing the data, we calculated summary statistics for all variables using Python programming. To do so we converted categorical variables into numerical values using label encoder. We also calculated the correlation of each variable with the target variable. The results showed that the variables with the highest correlation with AFFINITY_CARD were YRS_RESIDENCE , HOME_THEATER_PACKAGE and AGE.

Converting categorical variables into numerical values

```
from sklearn.preprocessing import LabelEncoder

# create a label encoder object
le = LabelEncoder()

# fit and transform the categorical columns
df['CUST_MARITAL_STATUS'] = le.fit_transform(df['CUST_MARITAL_STATUS'])
df['COUNTRY_NAME'] = le.fit_transform(df['COUNTRY_NAME'])
df['OCCUPATION'] = le.fit_transform(df['OCCUPATION'])
```

Calculating summary statistics for all variables:

```
# Compute summary statistics using describe
summary = df.describe()

# Add additional summary statistics (skewness and kurtosis)
summary.loc['skew'] = df.skew()
summary.loc['kurtosis'] = df.kurtosis()

# Print the summary statistics
summary
```

	AGE	CUST_MARITAL_STATUS	COUNTRY_NAME	OCCUPATION	YRS_RESIDENCE	AFFINITY_CARD	HOME_THEATER_PACKAGE	BOOKKEEPING_APPLICATION
count	1499.000000	1499.000000	1499.000000	1499.000000	1499.000000	1499.000000	1499.000000	1499.000000
mean	38.867912	2.037358	16.687125	6.559039	4.090060	0.253502	0.575050	0.880587
std	13.608974	1.594782	4.162925	4.099523	1.920802	0.435161	0.494500	0.324382
min	17.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	28.000000	1.000000	18.000000	2.000000	3.000000	0.000000	0.000000	1.000000
50%	37.000000	1.000000	18.000000	7.000000	4.000000	0.000000	1.000000	1.000000
75%	47.000000	4.000000	18.000000	10.000000	5.000000	1.000000	1.000000	1.000000
max	90.000000	5.000000	18.000000	14.000000	14.000000	1.000000	1.000000	1.000000
skew	0.591082	0.391724	-3.137268	0.062577	0.774459	1.134416	-0.303944	-2.349671
kurtosis	0.001751	-1.517185	8.540501	-1.405254	1.597708	-0.714054	-1.910168	3.525657

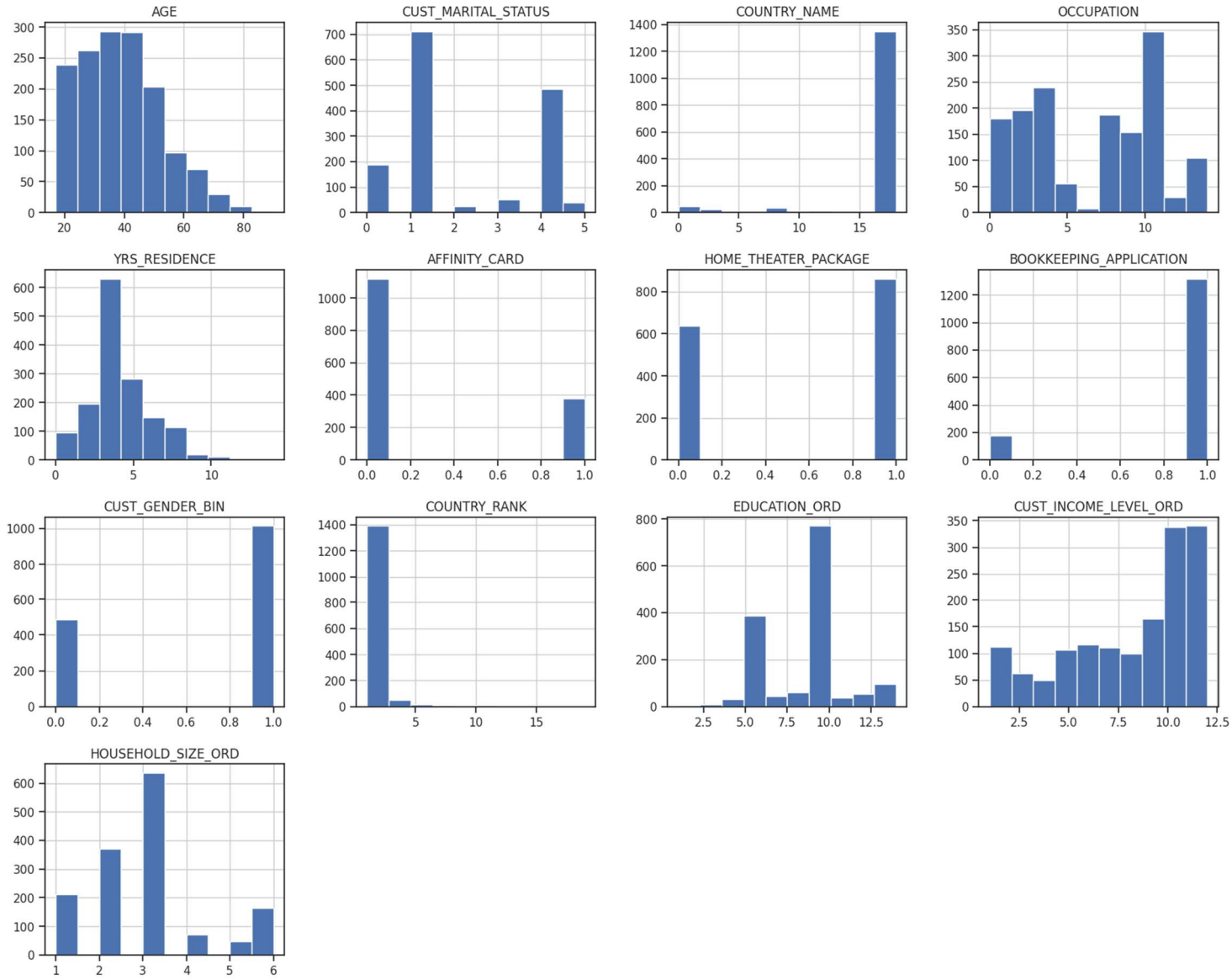
	CUST_GENDER_BIN	COUNTRY_RANK	EDUCATION_ORD	CUST_INCOME_LEVEL_ORD	HOUSEHOLD_SIZE_ORD
count	1499.000000	1499.000000	1499.000000	1499.000000	1499.000000
mean	0.676451	1.431621	8.717812	8.127418	2.906604
std	0.467986	1.789254	2.341310	3.086932	1.399725
min	0.000000	1.000000	1.000000	1.000000	1.000000
25%	0.000000	1.000000	6.000000	6.000000	2.000000
50%	1.000000	1.000000	9.000000	9.000000	3.000000
75%	1.000000	1.000000	10.000000	10.000000	3.000000
max	1.000000	19.000000	14.000000	12.000000	6.000000
skew	-0.755094	5.641809	-0.190707	-0.647516	0.880306
kurtosis	-1.431746	35.991259	-0.311256	-0.638914	0.311395

Visualizing Summary Statistics by using a histogram and Box Plot

1. Histogram

To better understand the distribution of our numeric data, we created a histogram for each column using the hist() function from Matplotlib. The code used to generate the histograms is shown below:

```
import matplotlib.pyplot as plt

# Plot histograms for all numeric columns in the dataframe
fig, ax = plt.subplots(figsize=(20, 16))
df.hist(ax=ax,bins=10)
plt.show()
```

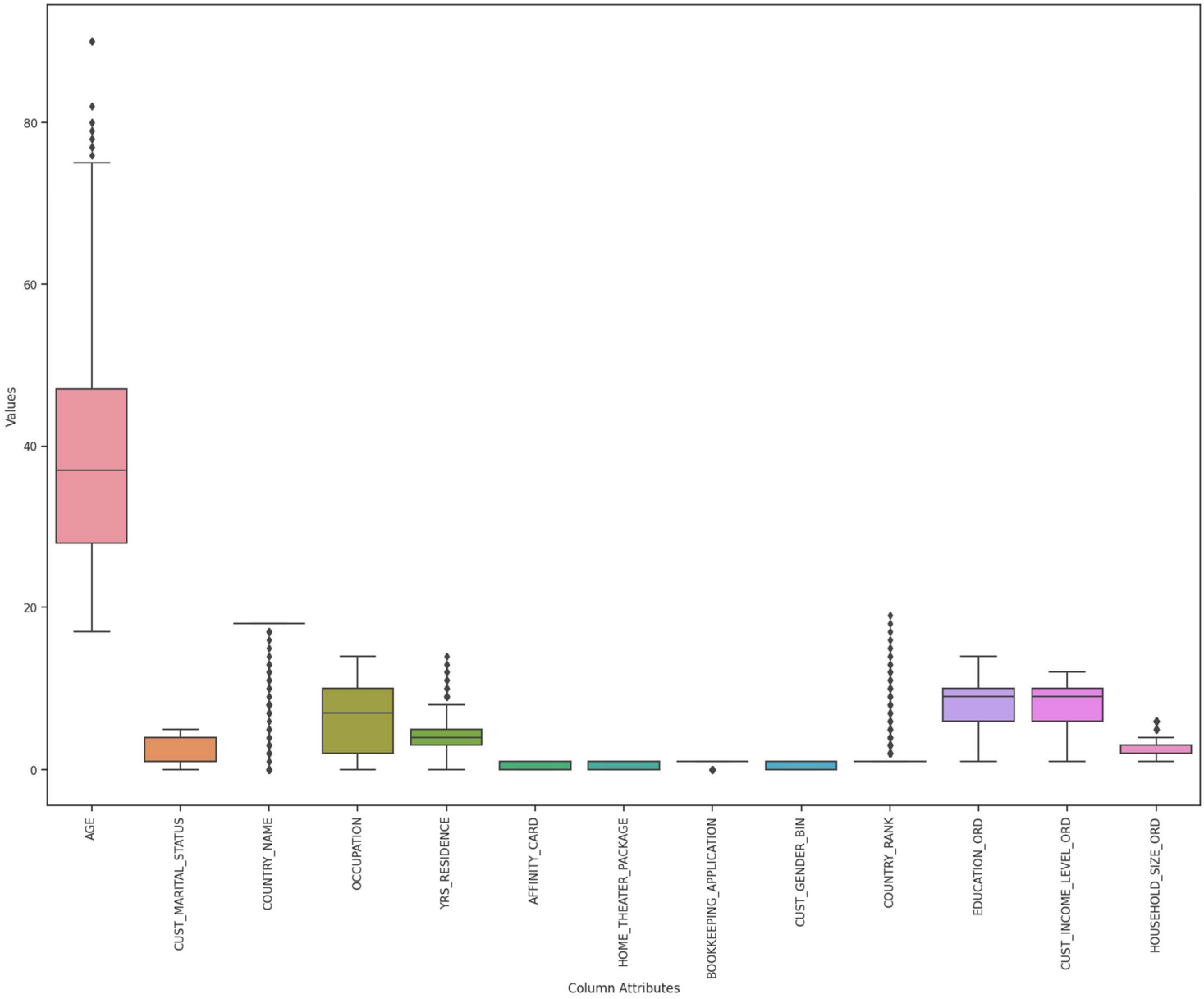


As we can see from the histograms, 'AGE 'appears to be skewed to the right and CUST_INCOME_LEVEL_ORD is left heavy tailed. This information can be useful as we continue our analysis.

2. Boxplot

To better understand the distribution and variability of our data, we created a boxplot using the boxplot() function from Seaborn. The code used to generate the boxplot is shown below:

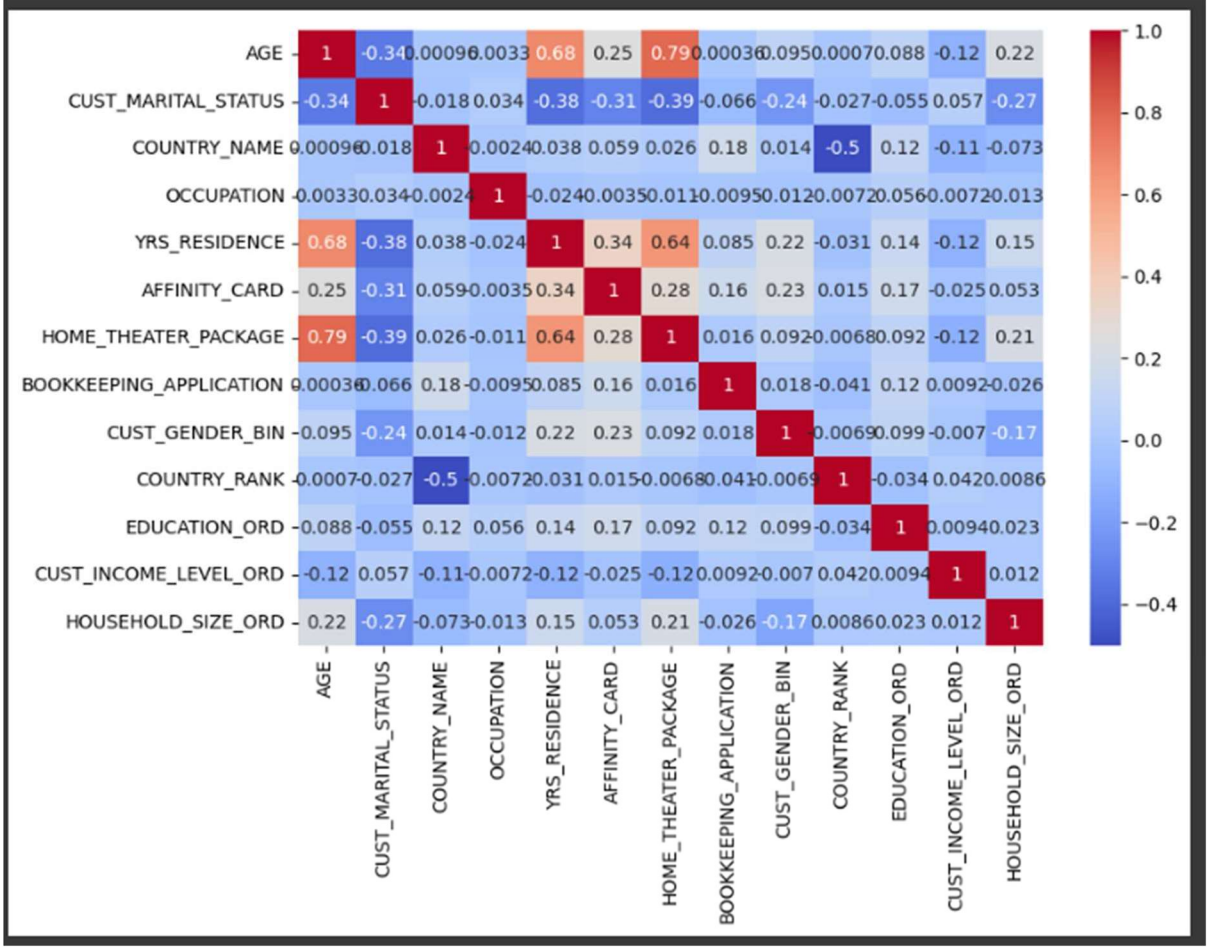
```
# @title Creating a Boxplot
plt.figure(figsize=(20, 7))
sns.boxplot(x='Borough', y='Price', data=merged_df)
plt.xticks(rotation=90)
plt.show()
```



From the box plot we can see that there are some potential outliers in the AGE column, also in COUNTRY_NAME/ COUNTRY_RANK, it seems one value has been repeated, which has led to a biased data set favoring a specific region. We can see some outliers in YRS_RESIDENCE and HOUSEHOLD_SIZE_ORD column. In addition to identifying outliers and biased data, the box plot also provides summary statistics for each column. These include the minimum and maximum values (represented by the whiskers), the lower and upper quartiles (the box itself), and the median (the line inside the box).

Plotting correlation Matrix and checking its values

```
# @title Plotting Correlation Matrix
import seaborn as sns
plt.figure(figsize=(9, 6))
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```




```
corr = df.corr()['AFFINITY_CARD'].sort_values(ascending=False)
print(corr)
```

```
AFFINITY_CARD           1.000000
YRS_RESIDENCE           0.342442
HOME_THEATER_PACKAGE    0.283793
AGE                     0.248351
CUST_GENDER_BIN         0.226012
EDUCATION_ORD           0.174437
BOOKKEEPING_APPLICATION 0.162573
COUNTRY_NAME            0.058552
HOUSEHOLD_SIZE_ORD      0.053144
COUNTRY_RANK            0.014561
OCCUPATION             -0.003531
CUST_INCOME_LEVEL_ORD  -0.024559
CUST_MARITAL_STATUS    -0.308963
Name: AFFINITY_CARD, dtype: float64
```

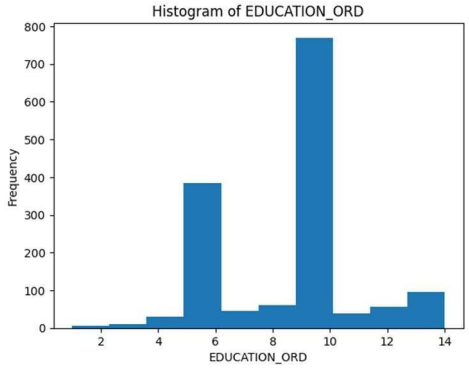
Data Exploration

We explored the data using histograms and scatter plots. Below is the python code to show:

a. Histogram plot of any user chosen variables.

```
# Define a list of variable names to choose from
var_list = ['AGE', 'YRS_RESIDENCE', 'EDUCATION_ORD', 'CUST_INCOME_LEVEL_ORD', 'HOUSEHOLD_SIZE_ORD']

# Keep running the program until the user chooses to exit
while True:
    # Print the list of variables to choose from
    print("Choose a variable to plot (or enter 'exit' to quit):")
    for i, var in enumerate(var_list):
        print(f"{i+1}. {var}")

    # Prompt the user for their choice
    choice = input()

    # Check if the user chose to exit
    if choice == "exit":
        break

    # Check if the user entered a valid choice
    try:
        choice = int(choice)
        var_name = var_list[choice-1]
    except:
        print("Invalid choice. Please try again.")
        continue

    # Plot the histogram of the selected variable
    plt.hist(df[var_name])
    plt.title(f"Histogram of {var_name}")
    plt.xlabel(var_name)
    plt.ylabel("Frequency")
    plt.show()
```

```
Choose a variable to plot (or enter 'exit' to quit):
1. AGE
2. YRS_RESIDENCE
3. EDUCATION_ORD
4. CUST_INCOME_LEVEL_ORD
5. HOUSEHOLD_SIZE_ORD
3
```

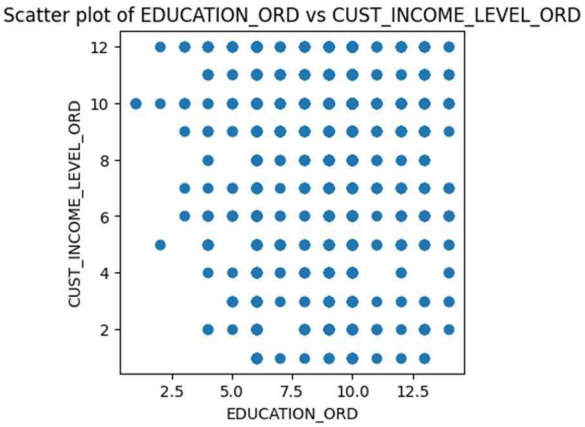


We begin the program by defining a list of variable names and entering an infinite loop using the while True statement. Within each iteration of the loop, we print out the list of variables using the print statement and a for loop that uses the enumerate function to get the index and value of each variable in the list. Then, we prompt the user to choose a variable by asking for input using the input function, and we store the input in a variable called choice. If the user enters the word "exit", we stop the loop using the break statement. Next, we try to convert the user's input to an integer using the int function, and if successful, we use the resulting integer to retrieve the corresponding variable name from the list and store it in a variable called var_name. If the conversion is not successful, we print an error message and continue to the next iteration of the loop using the continue statement. Finally, we display a histogram of the selected variable using the plt.hist function from the matplotlib library and add a title and labels to the histogram.

b. A scatter plot for any two-user chosen variables.

```
exit_flag = False

while not exit_flag:

    # Print the list of variables to choose from
    print("Choose two variables to plot (or enter 'exit' to quit)(Use space to separate numbers):")
    for i, var in enumerate(var_list):
        print(f"{i+1}. {var}")

    # Prompt the user for their choices
    choices = input().split()

    # Check if the user chose to exit
    if 'exit' in choices:
        print("Program terminated.")
        exit_flag = True
        continue

    # Check if the user entered valid choices
    if len(choices) != 2:
        print("Invalid choice. Please enter two choices separated by a space.")
        continue

    try:
        choice1 = int(choices[0])
        choice2 = int(choices[1])
        var1 = var_list[choice1-1]
        var2 = var_list[choice2-1]
    except:
        print("Invalid choice. Please try again.")
        continue

    # Plot the scatter plot of the selected variables
    plt.figure(figsize=(4,4))
    plt.scatter(df[var1], df[var2])
    plt.title(f"Scatter plot of {var1} vs {var2}")
    plt.xlabel(var1)
    plt.ylabel(var2)
    plt.show()

# End of program
```

```
Choose two variables to plot (or enter 'exit' to quit)(Use space to separate numbers):
1. AGE
2. YRS_RESIDENCE
3. EDUCATION_ORD
4. CUST_INCOME_LEVEL_ORD
5. HOUSEHOLD_SIZE_ORD
3 4
```



A list of variable names called var_list is defined. The program enters a while loop that will continue until the user decides to exit. Within each iteration of the loop, the program displays the list of variables to choose from using the print statement and a for loop with the enumerate function to get the index and value of each variable in var_list. The user is prompted to choose two variables by entering their corresponding numbers separated by a space. If the user enters "exit", the loop is terminated. The program then checks if the user entered valid choices and converts the input to integers. If successful, the corresponding variable names are retrieved from var_list and a scatter plot of those variables is displayed using the plt.scatter function from the matplotlib library. The plot is given a title and labels using the plt.title, plt.xlabel, and plt.ylabel functions. The plt.show function is used to show the plot to the user.

Data Mining

Two predictive models were built to predict whether a customer would take the AFFINITY_CARD in the marketing campaign. The first model used logistic regression, while the second model used SVM. Both models were trained using the prepared data, and their performance was evaluated using the ROC Curve

To determine the best attributes to choose for our model we used RFE (Recursive Feature Elimination).It Is a feature selection method used in machine learning that recursively removes attributes and builds a model on those attributes that remain. The main objective of RFE is to select the subset of features that lead to the best predictive performance for the model.

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# Define the target variable and predictor variables
target = df['AFFINITY_CARD']
predictors = df.drop('AFFINITY_CARD', axis=1)
# Create a  model
model1 = LogisticRegression()
model2 = svm.SVC(kernel='linear')
# Create an RFE selector with 5 features
selector1 = RFE(model1)
selector2 = RFE(model2)
# Fit the selector to the data
selector1.fit(predictors, target)
selector2.fit(predictors, target)
# Get the selected features
selected_features1 = predictors.columns[selector1.support_]
selected_features2 = predictors.columns[selector2.support_]
print(selected_features1)
print(selected_features2)
```

Output:

```
Index(['CUST_MARITAL_STATUS', 'YRS_RESIDENCE', 'HOME_THEATER_PACKAGE',
       'BOOKKEEPING_APPLICATION', 'CUST_GENDER_BIN', 'EDUCATION_ORD'],
      dtype='object')
Index(['CUST_MARITAL_STATUS', 'YRS_RESIDENCE', 'HOME_THEATER_PACKAGE',
       'BOOKKEEPING_APPLICATION', 'CUST_GENDER_BIN', 'EDUCATION_ORD'],
      dtype='object')
```

Now performing logistic regression based on the above attributes:

We have decided to go for logistic regression since our target variable is binary in nature, in the below model we defined the target and predictor variables and then use Standard scaler to scale the data so that one attribute does not influence our model , we then split our data into training and testing sets so that we can check for accuracy.

```
# Perfroming Logistic Regression

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[['CUST_MARITAL_STATUS', 'YRS_RESIDENCE', 'HOME_THEATER_PACKAGE',
      'BOOKKEEPING_APPLICATION', 'CUST_GENDER_BIN', 'EDUCATION_ORD']], df['AFFINITY_CARD'], test_size=0.2, random_state=42)

# preprocess the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# define the logistic regression model
model = LogisticRegression(random_state=42)

# define k-fold cross-validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# perform cross-validation
scores = []
for train_index, test_index in kfold.split(X_train_scaled):
    Xl_train, Xl_test = X_train_scaled[train_index], X_train_scaled[test_index]
    yl_train, yl_test = y_train.iloc[train_index], y_train.iloc[test_index]
    model.fit(Xl_train, yl_train)
    yl_pred = model.predict(Xl_test)
    score = accuracy_score(yl_test, yl_pred)
    scores.append(score)

# evaluate the logistic regression model on the test set
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
test_accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

Performing SVM based on the above attributes:

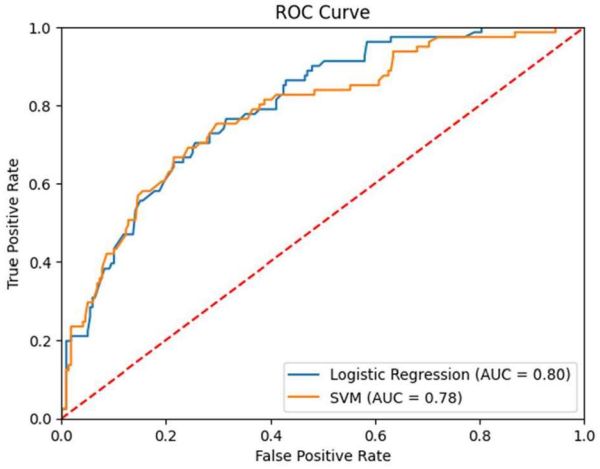
```
# Performing SVM

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[['CUST_MARITAL_STATUS', 'YRS_RESIDENCE', 'HOME_THEATER_PACKAGE',
    'BOOKKEEPING_APPLICATION', 'CUST_GENDER_BIN', 'EDUCATION_ORD']], df['AFFINITY_CARD'], test_size=0.2, random_state=42)

# preprocess the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# define the SVM model
clf = svm.SVC()

# define k-fold cross-validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# perform cross-validation
scores = []
for train_index, test_index in kfold.split(X_train_scaled):
    Xs_train, Xs_test = X_train_scaled[train_index], X_train_scaled[test_index]
    ys_train, ys_test = y_train.iloc[train_index], y_train.iloc[test_index]
    clf.fit(Xs_train, ys_train)
    ys_pred = clf.predict(Xs_test)
    score = accuracy_score(ys_test, ys_pred)
    scores.append(score)

# evaluate the SVM model on the test set
clf.fit(X_train_scaled, y_train)
y_pred = clf.predict(X_test_scaled)
test_accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

For the second model we used SVM, we carried our similar steps like splitting the training and testing data and using standard scaler as we did with the logistic regression and built our SVM model. In the next stage we would check different evaluation metrics for both the models.

Evaluation:

For visual inspection and evaluation, we plotted an ROC Curve for both the models:

```
from sklearn.metrics import roc_curve, roc_auc_score

# calculate the predicted probabilities for logistic regression model
y_pred_prob_lr = model.predict_proba(X_test_scaled)[:,1]

# calculate the predicted probabilities for SVM model
y_pred_prob_svm = clf.decision_function(X_test_scaled)

# calculate false positive rate (fpr), true positive rate (tpr) and thresholds for logistic regression model
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, y_pred_prob_lr)

# calculate false positive rate (fpr), true positive rate (tpr) and thresholds for SVM model
fpr_svm, tpr_svm, thresholds_svm = roc_curve(y_test, y_pred_prob_svm)

# calculate AUC score for logistic regression model
auc_lr = roc_auc_score(y_test, y_pred_prob_lr)

# calculate AUC score for SVM model
auc_svm = roc_auc_score(y_test, y_pred_prob_svm)

# plot the ROC curves for both models
plt.plot(fpr_lr, tpr_lr, label='Logistic Regression (AUC = %0.2f)' % auc_lr)
plt.plot(fpr_svm, tpr_svm, label='SVM (AUC = %0.2f)' % auc_svm)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



Metrics	Model	
	Logistic Regression	**SVM**
Mean Accuracy	0.7631	0.7798
Test Accuracy	0.7667	0.7833
Precision	0.6000	0.6290
Recall	0.4074	0.4815
F1 Score	0.4853	0.5455

Table 1: Evaluation Metrics of Machine Learning Models

In the above scenario, two machine learning models were evaluated, based on the output we can see that the SVM model outperformed the logistic regression model in all metrics. The mean accuracy of the SVM model was 0.7798 compared to 0.7631 for the logistic regression model. The SVM model also had a higher test accuracy of 0.7833 compared to 0.7667 for the logistic regression model. In terms of precision, recall, and F1 score, the SVM model achieved higher scores of 0.6290, 0.4815, and 0.5455, respectively, compared to 0.6000, 0.4074, and 0.4853 for the logistic regression model.

Discussion and Reflection

Based on the evaluation results, we conclude that the SVM model is more suitable for the task of binary classification on this dataset. The results of the analysis showed that years of residence, age , gender and education were the variables with the highest correlation with AFFINITY_CARD. The predictive models built using logistic regression and SVM showed promising results, with accuracies of 76% and 77%, respectively.

Conclusion

In conclusion, the analysis of the marketing campaign dataset provided insights into the factors that influence whether a customer will take the AFFINITY_CARD in the marketing campaign. The use of Python programming facilitated the cleaning, preparation, and analysis of the data, and the results of the analysis can be used to inform marketing strategies for the retailer company.

Code Repository

The .ipynb file is available in the zip file or the link to access the Google Colab Notebook is:
https://colab.research.google.com/drive/1If7yTB2FOQf8wf-iFGWNPcGlUvYMaOyY?usp=sharing

References

1. Pandas: Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)
2. NumPy: Travis E. Oliphant. A guide to NumPy, USA: Trelgol Publishing (2006)
3. Matplotlib: J. D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007)
4. Seaborn: Michael Waskom. Seaborn: statistical data visualization, Journal of Open Source Software, 6(60), 3021 (2021)