Programming for Data Analytics Technical Report

1 - Data Understanding

1.1 - Metadata Table

A metadata table was produced to better understand the marketing campaign dataset. The mean, minimum, maximum and standard deviation values were recorded in the table of each attribute. All necessary modules were imported to achieve this. Using the function describe() allowed created a table which presented all the required information. Using this function was a quick way of getting all the required values using a simple function already in Google Colab.

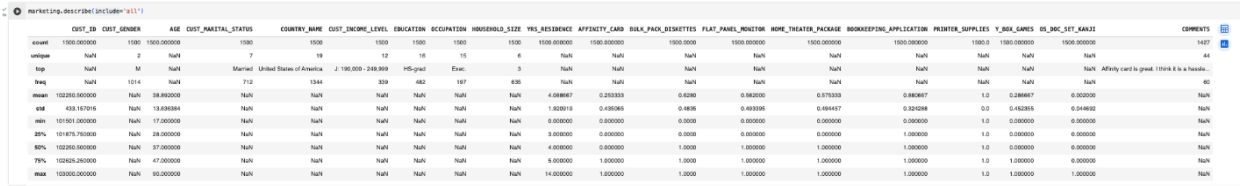
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
marketing = pd.read_csv('/content/drive/MyDrive/Marketing Campaign data.csv')
marketing.shape[1]
```

```
19
```

```
print(marketing.columns)
```

```
Index(['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_APPLICATION',
       'PRINTER_SUPPLIES', 'Y_BOX_GAMES', 'OS_DOC_SET_KANJI', 'COMMENTS'],
      dtype='object')
```

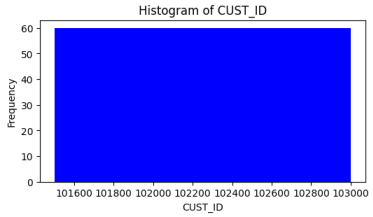
Table 1: Metatable showing mean, minimum, maximum and standard deviation values using describe().

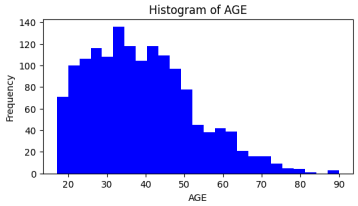


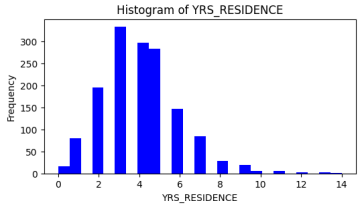
	CUST_ID	CUST_GENDER	AGE	CUST_MARITAL_STATUS	COUNTRY_NAME	CUST_INCOME_LEVEL	EDUCATION	OCCUPATION	HOUSEHOLD_SIZE	YRS_RESIDENCE	AFFINITY_CARD	BULK_PACK_DISKETTES	FLAT_PANEL_MONITOR	HOME_THEATER_PACKAGE	BOOKKEEPING_APPLICATION	PRINTER_SUPPLIES	Y_BOX_GAMES	OS_DOC_SET_KANJI	COMMENTS
count	1500.000000	1500	1500.000000	1500	1500	1500	1500	1500	1500	1500.000000	1500.000000	1500.0000	1500.000000	1500.000000	1500.000000	1500.0	1500.000000	1500.000000	1427
unique	NaN	2	NaN	7	19	12	16	15	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	44
top	NaN	M	NaN	Married	United States of America	J: 190,000 - 249,999	HS-grad	Exec.	3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Affinity card is great. I think it is a hassle...
freq	NaN	1014	NaN	712	1344	339	482	197	635	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	60
mean	102250.500000	NaN	38.892000	NaN	NaN	NaN	NaN	NaN	NaN	4.088667	0.253333	0.8280	0.582000	0.575333	0.880667	1.0	0.286667	0.002000	NaN
std	433.167015	NaN	13.636384	NaN	NaN	NaN	NaN	NaN	NaN	1.920919	0.435065	0.4835	0.493395	0.494457	0.324288	0.0	0.452355	0.044692	NaN
min	101501.000000	NaN	17.000000	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	0.0000	0.000000	0.000000	0.000000	1.0	0.000000	0.000000	NaN
25%	101875.750000	NaN	28.000000	NaN	NaN	NaN	NaN	NaN	NaN	3.000000	0.000000	0.0000	0.000000	0.000000	1.000000	1.0	0.000000	0.000000	NaN
50%	102250.500000	NaN	37.000000	NaN	NaN	NaN	NaN	NaN	NaN	4.000000	0.000000	1.0000	1.000000	1.000000	1.000000	1.0	0.000000	0.000000	NaN
75%	102625.250000	NaN	47.000000	NaN	NaN	NaN	NaN	NaN	NaN	5.000000	1.000000	1.0000	1.000000	1.000000	1.000000	1.0	1.000000	0.000000	NaN
max	103000.000000	NaN	90.000000	NaN	NaN	NaN	NaN	NaN	NaN	14.000000	1.000000	1.0000	1.000000	1.000000	1.000000	1.0	1.000000	1.000000	NaN

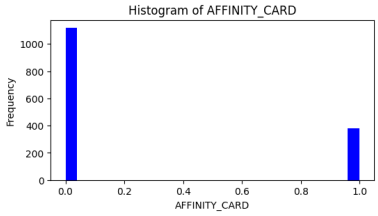
```
marketing.describe(include='all')
```

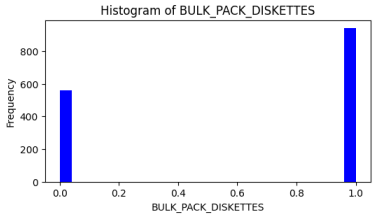
	CUST_ID	CUST_GENDER	AGE	CUST_MARITAL_STATUS	COUNTRY_NAME	CUST_INCOME_LEVEL	EDUCATION	OCCUPATION	HOUSEHOLD_SIZE	YRS_RESIDENCE	AFFIN
count	1500.000000	1500	1500.000000	1500	1500	1500	1500	1500	1500	1500.000000	15
unique	NaN	2	NaN	7	19	12	16	15	6	NaN	
top	NaN	M	NaN	Married	United States of America	J: 190,000 - 249,999	HS-grad	Exec.	3	NaN	
freq	NaN	1014	NaN	712	1344	339	482	197	635	NaN	
mean	102250.500000	NaN	38.892000	NaN	NaN	NaN	NaN	NaN	NaN	4.088667	
std	433.157015	NaN	13.636384	NaN	NaN	NaN	NaN	NaN	NaN	1.920919	
min	101501.000000	NaN	17.000000	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	
25%	101875.750000	NaN	28.000000	NaN	NaN	NaN	NaN	NaN	NaN	3.000000	
50%	102250.500000	NaN	37.000000	NaN	NaN	NaN	NaN	NaN	NaN	4.000000	
75%	102625.250000	NaN	47.000000	NaN	NaN	NaN	NaN	NaN	NaN	5.000000	
max	103000.000000	NaN	90.000000	NaN	NaN	NaN	NaN	NaN	NaN	14.000000	

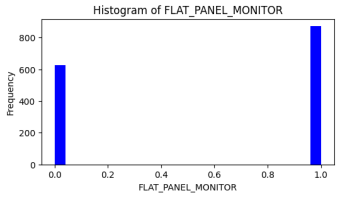


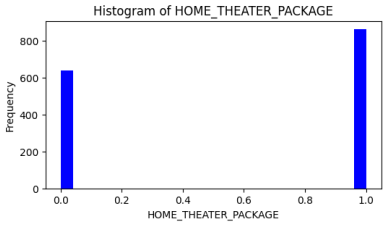


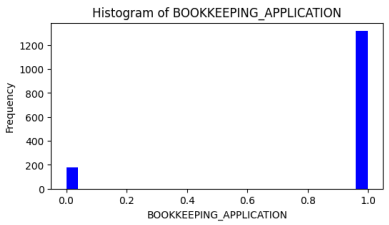


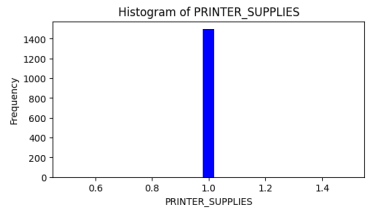


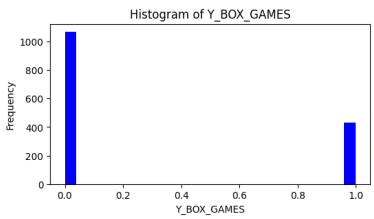


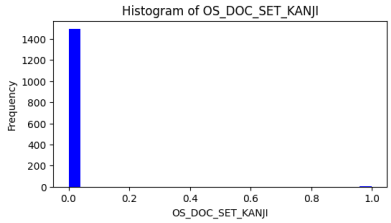












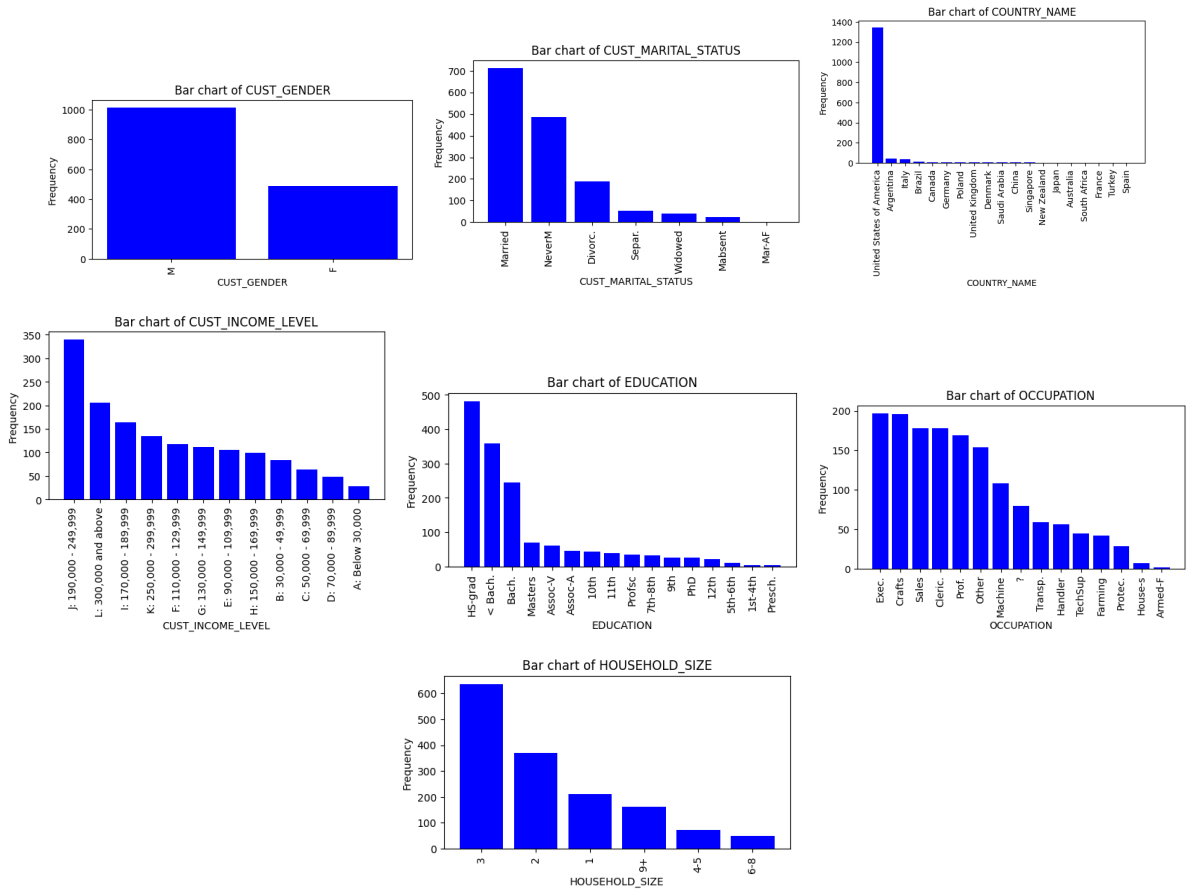


Figure 1: Histograms for variables with numerical data and bar charts for variables with nominal data

The figures in Figure 1 were produced with the following code:

```
numerical_columns = marketing.select_dtypes(include=['int',
'float']).columns
for column in numerical_columns:
   plt.figure(figsize=(6, 3))
   plt.hist(marketing[column], bins=25, color='blue')
   plt.title(f'Histogram of {column}')
   plt.xlabel(column)
   plt.ylabel('Frequency')
   plt.show()
```

```
nominal_columns = marketing.select_dtypes(include=['object']).columns
for column in nominal_columns:
   plt.figure(figsize=(6, 3))
   value_counts = marketing[column].value_counts()
   plt.bar(value_counts.index, value_counts, color='blue')
```

```
    plt.title(f'Bar chart of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.xticks(rotation=90)
    plt.show()
```

The histograms in Figure 1 show the frequency of each value present in the variable. As many variables contained binary values (1 or 0), only two values were present in the figure. For variables such as age, there were many different values present as shown in the figure.

1.2 - Missing or Error Data

Error data occurred due to some of the attributes in the original data being string values as seen in table 1. The returned values had to be numerical (floats or integers) for the mean, minimum, maximum and standard deviation to be calculated. The missing data were present in the attributes CUST_GENDER, CUST_MARITAL_STATUS, COUNTRY_NAME, CUST_INCOME_LEVEL, EDUCATION, OCCUPATION, HOUSEHOLD_SIZE and COMMENTS. Therefore in the later stages of the data analysis process, these values will change into ordinal numerical values. Within the data no anomalies were found and all rows had data present in each variable.

2 - Data Preparation

2.1 - Reduce Variables

Certain variables which did not influence the target variable were reduced before data analysis and mining. Based on the target variable AFFINITY_CARD, the attributes COMMENTS, OS_DOC_SET_KANJI and PRINTER_SUPPLIES were dropped. The variables OS_DOC_SET_KANJI and PRINTER_SUPPLIES both contained only one value as seen in the histogram which means they will have no effect on the target variable. The variable COMMENTS contains customer comments which cannot be analysed along with the other variables as they are not possible to infer and are not numerical values. The following code drops the desired variables and also checks if they were taken away. It can be seen that the updated dataset now has 16 variables instead of 19.

```
newmarketing = marketing.drop(["COMMENTS", "OS_DOC_SET_KANJI",
"PRINTER_SUPPLIES"], axis='columns')
newmarketing.shape
```

(1500, 16)

2.2 - Clean Data

Cleaning data is crucial before analysis so the data is as reliable as possible. The data was checked for duplicated rows and any duplicates were removed from the dataset. The following code was used in which the output is shown below and the function drop_duplicates() ensured that the duplicates were dropped.

```
newmarketing.duplicated()
```

output

```
0      False
1      False
2      False
3      False
4      False
      ...
1495   False
1496   False
1497   False
1498   False
1499   False
Length: 1500, dtype: bool
```

```
newmarketing.drop_duplicates()
```

All null values were also dropped within the data as they would create errors during analysis.

```
newmarketing.dropna(how="all")
```

The following code and output showed that there were no null values present in the dataset:

```
newmarketing.isnull().sum()
```

output

```
CUST_ID                0
CUST_GENDER            0
AGE                    0
CUST_MARITAL_STATUS    0
COUNTRY_NAME           0
CUST_INCOME_LEVEL      0
EDUCATION              0
OCCUPATION             0
HOUSEHOLD_SIZE         0
YRS_RESIDENCE          0
AFFINITY_CARD          0
BULK_PACK_DISKETTES    0
FLAT_PANEL_MONITOR     0
HOME_THEATER_PACKAGE   0
BOOKKEEPING_APPLICATION  0
```

```
Y_BOX_GAMES          0
dtype: int64
```

2.3 - CUST_GENDER into Binary F -0, M-1 Reduce Variables

Two methods were used to change CUST_GENDER from F and M to 0 and 1 representing genders. The first method used a dictionary method in Python to make 0 and 1 represent F and M respectively. Mapping was then used to link the dictionary with the variable in the dataset. Dictionaries store a collection of key-value pairs called the key and value. Each key is associated with the value, in which the value can be modified and changed whenever or retrieved. The code below and other codes for changing variables into ordinary numerical values during the preparation phase use a dictionary method to adjust the dataset to its requirements.

```
gender_m = {'F': 0, 'M': 1}
newmarketing['CUST_GENDER'] = newmarketing['CUST_GENDER'].map(gender_m)
```

The second method included using the replace function method. This replaces all rows with F to 0 and all M rows to 1 within the variables. The replace method is different to the dictionary method as it does not contain a key and value pair. Therefore the old values in the dataset will not be linked with the new numerical values.

```
newmarketing['CUST_GENDER_id'] = newmarketing['CUST_GENDER'].replace({'F':
0, 'M': 1})
```

2.4 - COUNTRY_NAME into Ordinal Number

Based on the country's occurrence in the data set, their names were assigned a number to represent them as a numerical value. The first task was to use code to find the order in which countries occurred the most in the dataset. The following code was used.

```
value_counts = newmarketing['COUNTRY_NAME'].value_counts()
```

```
print(value_counts)
```

```
output
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: count, dtype: int64
```

The output shows that the United States of America has the highest value which means it will be represented with the value 1. The rest of the countries were given a value based on the order of amount of times they appeared in the dataset.

The first method to assign numerical values to the countries was by creating a dictionary. The code is shown below:

```
no_to_country = {
  "United States of America" : "1",
  "Argentina" : '2',
  "Italy" : "3",
  "Brazil" : "4",
  "Canada" : "5",
  "Germany" : "6",
  "Poland" : "7",
  "United Kingdom" : "8",
  "Denmark" : "9",
  "Saudi Arabia" : "10",
  "China" : "11",
  "Singapore" : "12",
  "New Zealand" : "13",
  "Japan" : "14",
  "Australia" : "15",
  "South Africa" : "16",
  "France" : "17",
  "Turkey" : "18",
  "Spain" : "19"
}
```

```
newmarketing['COUNTRY_id'] =
newmarketing['COUNTRY_NAME'].map(no_to_country)
```

The dictionary was mapped to the original variable in the dataset so that all rows were given the correct value.

The second method was done using the replace function which replaced all country names with the chosen numerical value.

```
newmarketing['COUNTRY_NAME_id']=
newmarketing['COUNTRY_NAME'].replace(["United States of America",
"Argentina", "Italy", "Brazil",  "Canada",
                    "Germany", "Poland", "United Kingdom", "Denmark",
"Saudi Arabia",
                    "China",  "Singapore", "New Zealand",  "Japan",
"Australia", "South Africa",
                    "France", "Turkey",  "Spain"],

[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]).astype(int)
```

The astype(int) function ensured that the code recognised the new values as integers and not objects so that data analysis could be carried out. The following code shows that this is the case with this variable:

```
column_dtype = newmarketing['COUNTRY_NAME_id'].dtype
print(column_dtype)
int64
```

2.5 - CUST_INCOME_LEVEL into 3 Ordinal Levels

The CUST_INCOME_LEVEL variable was changed into three ordinal variables. A value of 1 represented low income, 2 represented middle income and 3 represented high income. First, it was looked at how many different values were present in this variable using the following code:

```
unique_income_levels = newmarketing['CUST_INCOME_LEVEL'].nunique()
print(unique_income_levels)
12
```

```
all_values = newmarketing['CUST_INCOME_LEVEL'].unique()
all_value_asc = sorted(all_values)
print(all_value_asc)
```

```
['A: Below 30,000', 'B: 30,000 - 49,999', 'C: 50,000 - 69,999', 'D: 70,000
- 89,999', 'E: 90,000 - 109,999', 'F: 110,000 - 129,999', 'G: 130,000 -
149,999', 'H: 150,000 - 169,999', 'I: 170,000 - 189,999', 'J: 190,000 -
249,999', 'K: 250,000 - 299,999', 'L: 300,000 and above']
```

Based on the results from the code and research carried out on the average, low and high-income levels, the unique values were categorised into the three required values. The following code is the first method which categorises all the values into one of three numerical values:

```
customer_income = {
   'A: Below 30,000' : "1",
   'B: 30,000 - 49,999' : "1",
   'C: 50,000 - 69,999' : "2",
   'D: 70,000 - 89,999' : "2",
   'E: 90,000 - 109,999' : "2",
   'F: 110,000 - 129,999' : "2",
   'G: 130,000 - 149,999' : "3",
   'H: 150,000 - 169,999' : "3",
   'I: 170,000 - 189,999' : "3",
   'J: 190,000 - 249,999' : "3",
   'K: 250,000 - 299,999' : "3",
   'L: 300,000 and above' : "3"

}
newmarketing['income_levels'] =
newmarketing['CUST_INCOME_LEVEL'].map(customer_income)
```

The code above uses a dictionary method which categorises A and B as low-income (1), C, D, E and F as medium-income (2) and G, H, I, J, K and L as high-income (3).

The second method used the replace function to change the original values into the required category values.

```
newmarketing['INCOME_LEVEL_id']=
newmarketing['CUST_INCOME_LEVEL'].replace(['A: Below 30,000', 'B: 30,000 -
49,999', 'C: 50,000 - 69,999',
                    'D: 70,000 - 89,999', 'E: 90,000 - 109,999', 'F:
110,000 - 129,999',
                    'G: 130,000 - 149,999', 'H: 150,000 - 169,999',
                    'I: 170,000 - 189,999', 'J: 190,000 - 249,999',
                    'K: 250,000 - 299,999', 'L: 300,000 and above'],
```

```
                        [1,1,2,2,2,2,3,3,3,3,3,3])
column_dtype = newmarketing['INCOME_LEVEL_id'].dtype
print(column_dtype)
```

```
int64
```

2.6 - EDUCATION into ordinal numbers

The EDUCATION variable values were changed based on the USA education level in ascending order of lowest education level to highest education level. The following code was used to see which education levels were present in the original dataset:

```
unique_education = newmarketing['EDUCATION'].nunique()
print(unique_education)
```

```
16
```

```
edu_level = newmarketing['EDUCATION'].unique()
education_level = sorted(edu_level)
print(education_level)
```

```
['10th', '11th', '12th', '1st-4th', '5th-6th', '7th-8th', '9th', '<
Bach.', 'Assoc-A', 'Assoc-V', 'Bach.', 'HS-grad', 'Masters', 'PhD',
'Presch.', 'Profsc']
```

The first method uses a dictionary code to change the explicit education levels into numerical values:

```
level_of_education = {
  'Presch.' : "1",
  '1st-4th' : "2",
  '5th-6th' : '3',
  '7th-8th' : "4",
  '9th' : "5",
  '10th' : "6",
  '11th' : "7",
  '12th' : "8",
  'HS-grad' : "9",
  'Assoc-A' : "10",
  'Assoc-V' : "11",
  'Bach.' : "12",
  '< Bach.' : "13",
  'Masters' : "14",
  'PhD' : "15",
  'Profsc' : "16"
}
```

```
newmarketing['education_lev'] =
newmarketing['EDUCATION'].map(level_of_education)
newmarketing
```

The second method used the replace function to change education levels into numerical values:

```
newmarketing['EDUCATION_LEV_id']=
newmarketing['EDUCATION'].replace(['Presch.', '1st-4th' , '5th-6th',
                     '7th-8th', '9th', '10th',
                     '11th', '12th',
                     'HS-grad', 'Assoc-A',
                     'Assoc-V', 'Bach.', '< Bach.',
                      'Masters', 'PhD', 'Profsc'],
                      [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16])
column_dtype = newmarketing['EDUCATION_LEV_id'].dtype
print(column_dtype)
int64
```

2.7 - HOUSEHOLD_SIZE into ordinal numbers

The HOUSEHOLD_SIZE variable contained six different values in the dataset and was changed into ordinal numbers using two different methods. The sorted function was used along with the unique function to see all values present in the original dataset and the number of unique values.

```
unique_household = newmarketing['HOUSEHOLD_SIZE'].nunique()
print(unique_household)
6
house_size = newmarketing['HOUSEHOLD_SIZE'].unique()
household_number = sorted(house_size)
print(household_number)
['1', '2', '3', '4-5', '6-8', '9+']
```

The first method uses a dictionary to assign the HOUSEHOLD_SIZE variable values into ordinal numbers. Even though from the code above there seem to be numerical values, python will not recognise them as this due to there being other symbols in certain values. Therefore this variable must be changed into ordinal numerical types. The following code changed the six values into the correct ordinal numbers:

```
household_number = {
  '1' : "1",
  '2' : "2",
```

```
  '3' : '3',
  '4-5' : "4",
  '6-8' : "5",
  '9+' : "6"
}
newmarketing['household_amount'] =
newmarketing['HOUSEHOLD_SIZE'].map(household_number)
```

The second method used the replace function to change the values into the required ordinal numbers:

```
newmarketing['HOUSEHOLD_SIZE_id']=
newmarketing['HOUSEHOLD_SIZE'].replace(['1', '2', '3', '4-5', '6-8',
'9+'],
                    [1,2,3,4,5,6])
column_dtype = newmarketing['HOUSEHOLD_SIZE_id'].dtype
print(column_dtype)
int64
```

3 - Data Analysis:

Summary statistics of sum, mean, standard deviation, skewness, and kurtosis of all variables were produced during the data analytics phase. This was achieved using the following code:

```
summary_data = {"sum" : newmarketing_2.sum(),
                "mean " : newmarketing_2.mean(),
                "std dev" : newmarketing_2.std(),
                "Skewness" : newmarketing_2.skew(),
                "kurtosis" : newmarketing_2.kurtosis(),
                 }
summary_dataframe = pd.DataFrame(summary_data)
summary_dataframe
```

Table 2: All variables in the dataset showing sum, mean, standard deviation, skewness and kurtosis values

	sum	mean	std dev	Skewness	kurtosis
CUST_ID	153375750	102250.500000	433.157015	0.000000	-1.200000
AGE	58338	38.892000	13.636384	0.594253	0.004432
YRS_RESIDENCE	6133	4.088667	1.920919	0.775118	1.596695
AFFINITY_CARD	380	0.253333	0.435065	1.135444	-0.711719
BULK_PACK_DISKETTES	942	0.628000	0.483500	-0.530180	-1.721206
FLAT_PANEL_MONITOR	873	0.582000	0.493395	-0.332835	-1.891745
HOME_THEATER_PACKAGE	863	0.575333	0.494457	-0.305118	-1.909451
BOOKKEEPING_APPLICATION	1321	0.880667	0.324288	-2.350839	3.531149
Y_BOX_GAMES	430	0.286667	0.452355	0.944471	-1.109456
CUST_GENDER_id	1014	0.676000	0.468156	-0.752890	-1.435072
COUNTRY_NAME_id	2147	1.431333	1.788692	5.643832	36.017806
INCOME_LEVEL_id	3941	2.627333	0.619173	-1.437659	0.902407
EDUCATION_LEV_id	15973	10.648667	2.705694	-0.575723	0.206557
HOUSEHOLD_SIZE_id	4361	2.907333	1.399542	0.878816	0.309316

The results from Table 2 show that some variables had slightly skewed and kurtosis data, especially the variable COUNTRY_NAME_id having the most. The correlation values between all independent variables were measured against the target variable AFFINITY_CARD using the code below:

```
target_variable = 'AFFINITY_CARD'
correlations = newmarketing_2.corr()[target_variable]

correlations
```

```
CUST_ID                   -0.025969
AGE                        0.246711
YRS_RESIDENCE              0.342691
AFFINITY_CARD              1.000000
BULK_PACK_DISKETTES       -0.017887
FLAT_PANEL_MONITOR        -0.028467
HOME_THEATER_PACKAGE       0.283358
BOOKKEEPING_APPLICATION    0.162404
Y_BOX_GAMES               -0.281121
CUST_GENDER_id             0.226390
COUNTRY_NAME_id            0.014653
INCOME_LEVEL_id           -0.025722
EDUCATION_LEV_id           0.272877
HOUSEHOLD_SIZE_id          0.052823
Name: AFFINITY_CARD, dtype: float64
```

Figure 2: The correlation between the independent variables and the target variable AFFINITY_CARD

Figure 2 shows that the variable with the strongest correlation with the target variable is YRS_RESIDENCE which has a value of 0.34. This suggests that the more years someone is as a resident in their area, the more likely they are to have a high-value affinity card.

4 - Data Exploration

During the data exploration phase, a histogram was produced in the AGE variable using the following code:

```
print("Choose variable out of the following:")
print(newmarketing_2.columns)
variable = input("Enter variable name: ")

if variable not in newmarketing_2.columns:
   print("Variable not available.")
else:
    newmarketing_2[variable].plot(kind='hist', bins=25, color='lightblue',
edgecolor ='black')
   plt.title(f'Histogram of {variable}')
   plt.xlabel(variable)
   plt.ylabel('Frequency')
   plt.show()
```

```
•••  Choose variable out of the following
     Index(['CUST_ID', 'AGE', 'YRS_RESIDENCE', 'AFFINITY_CARD',
            'BULK_PACK_DISKETTES', 'FLAT_PANEL_MONITOR', 'HOME_THEATER_PACKAGE',
            'BOOKKEEPING_APPLICATION', 'Y_BOX_GAMES', 'CUST_GENDER_id',
            'COUNTRY_NAME_id', 'INCOME_LEVEL_id', 'EDUCATION_LEV_id',
            'HOUSEHOLD_SIZE_id'],
           dtype='object')
     name
```

Figure 3: Output of which a user can select which variable they would like to select to produce a histogram

The result in Figure 3 shows how the user will input their chosen variable to produce a histogram. If the user chooses a variable that is not part of the dataset or not spelt correctly, it will print as seen in Figure 4.

```
Choose variable out of the following:
Index(['CUST_ID', 'AGE', 'YRS_RESIDENCE', 'AFFINITY_CARD',
       'BULK_PACK_DISKETTES', 'FLAT_PANEL_MONITOR', 'HOME_THEATER_PACKAGE',
       'BOOKKEEPING_APPLICATION', 'Y_BOX_GAMES', 'CUST_GENDER_id',
       'COUNTRY_NAME_id', 'INCOME_LEVEL_id', 'EDUCATION_LEV_id',
       'HOUSEHOLD_SIZE_id'],
      dtype='object')
Enter variable name: house
Variable not available.
```

Figure 4: Result from the user putting in an invalid variable name

Figure 5 shows the output of a histogram when a user puts the variable AGE in the input. All different variables available will print out different results of histograms.

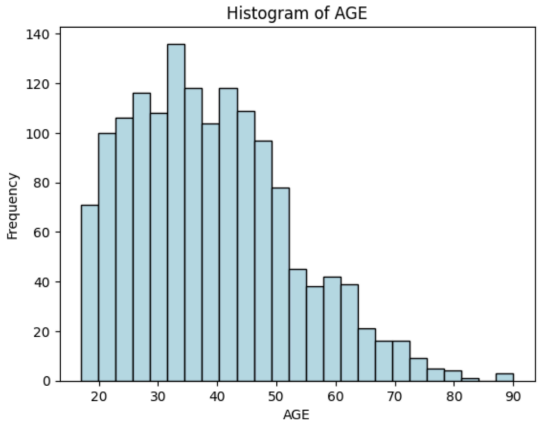


Figure 5: Histogram of the variable AGE using the code script

Figure 5 shows that the AGE variable is heavily skewed to the right which suggests that there are more younger aged people present in the data than older aged people. Each variable can be seen through visualisation whether the data is skewed or kurtosis.

Code was produced to allow users to pick two independent variables to create a scatter plot. The following code allows the user to pick the x and y axis from the available variables in the dataset to see how they correlate to each other:

```
print("Choose variable out of the following:")
print(newmarketing_2.columns)
x_variable = input("Enter x variable name: ")
y_variable = input("Enter y variable name: ")

if x_variable not in newmarketing_2.columns or y_variable not in
newmarketing_2.columns:
   print("One or more variables not available.")
else:
        newmarketing_2.plot(kind='scatter', x=x_variable, y=y_variable,
color='lightblue', edgecolor ='black')
   plt.title(f'Scatter plot of {x_variable} and {y_variable}')
   plt.xlabel(x_variable)
   plt.ylabel(y_variable)
   plt.show()
```

Figure 6 shows how the user can input their chosen variables and Figure 7 shows an example of a scatter plot generated using the variables INCOME_LEVEL_id and AGE.

```
Choose variable out of the following:
Index(['CUST_ID', 'AGE', 'YRS_RESIDENCE', 'AFFINITY_CARD',
       'BULK_PACK_DISKETTES', 'FLAT_PANEL_MONITOR', 'HOME_THEATER_PACKAGE',
       'BOOKKEEPING_APPLICATION', 'Y_BOX_GAMES', 'CUST_GENDER_id',
       'COUNTRY_NAME_id', 'INCOME_LEVEL_id', 'EDUCATION_LEV_id',
       'HOUSEHOLD_SIZE_id'],
      dtype='object')
Enter x variable name: INCOME_LEVEL_id
Enter y variable name:
```

Figure 6: Output of which a user can select which variables they would like to select to produce a scatter plot

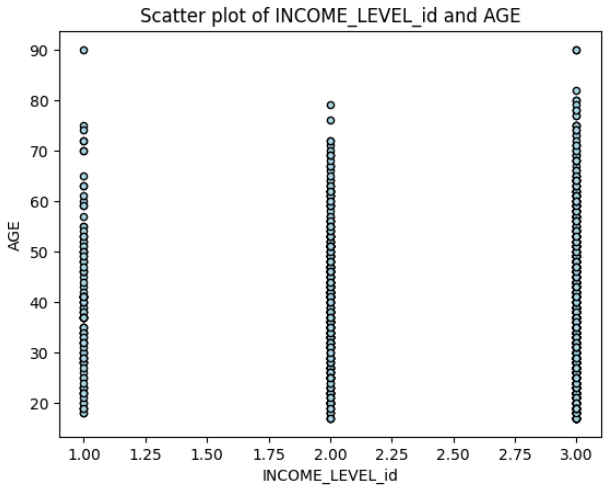


Figure 7: Scatter plot of the variables INCOME_LEVEL_id and AGE using the code script

5 - Data Mining

A random sample of 1000 variables was used during the data mining phase which included producing a Logistical Regression Model and a Support Vector Machine Model. The random variables were selected using the following code:

```
random_marketing = newmarketing_2.sample(n=1000)
random_marketing.head()
random_marketing.shape
```

(1000, 14)

From using the function it could be seen through the CUST_ID variable that the rows were now randomly selected. The shape() function also showed that there were 1000 rows and 14 variables in the dataset now named random_marketing.

The variables were first sorted into independent variables and one target variable being AFFINITY_CARD. This was achieved with the following code:

```
feature_cols = ['CUST_ID', 'AGE', 'YRS_RESIDENCE', 'BULK_PACK_DISKETTES',
```

```
                'FLAT_PANEL_MONITOR', 'HOME_THEATER_PACKAGE',
                'BOOKKEEPING_APPLICATION', 'Y_BOX_GAMES', 'CUST_GENDER_id',
                'COUNTRY_NAME_id', 'INCOME_LEVEL_id', 'EDUCATION_LEV_id',
                'HOUSEHOLD_SIZE_id']
X = random_marketing[feature_cols]
y = random_marketing.AFFINITY_CARD
```

The appropriate libraries were installed over the course of the Logistical Regression Model process. Before the data was partitioned into training and testing sets, the data was normalised so that all the variables had equal values due to there being different measurements such as age. Train and test split was implemented in the data. The training set allows the machine model to be trained whilst the testing data evaluates the performance of unseen data. This was carried out using the following:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y,
test_size=0.25, random_state=30)
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(random_state=16, max_iter=1000)
logreg.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000, random_state=16)
```

```
y_pred = logreg.predict(X_test)
```

A confusion matrix was created which showed the actual and predicted true positive, true negative, false positive and false negative values. Figure 8 was created using the following code:

```
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cnf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
```

```
plt.text(0.5, 0.1, 'True Negative', ha='center', va='center',
color='white')
plt.text(0.5, 1.1, 'False Positive', ha='center', va='center')
plt.text(1.5, 0.1, 'False Negative', ha='center', va='center')
plt.text(1.5, 1.1, 'True Positive', ha='center', va='center')

plt.show()
```

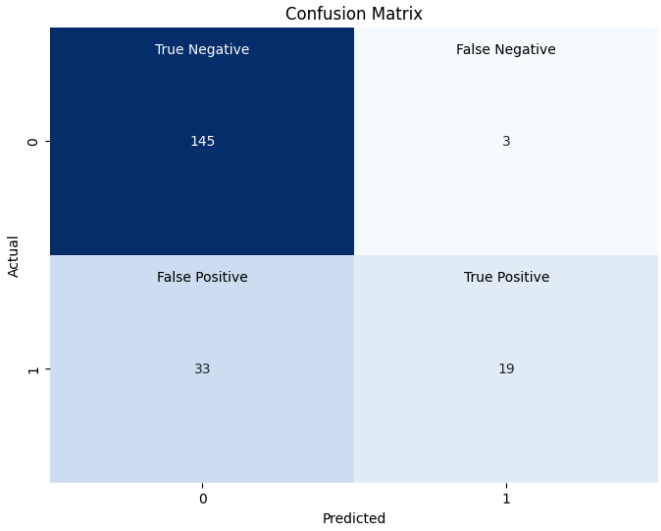


Figure 8: Confusion matrix results using the test and train target variable

More libraries were installed to complete the rest of the Logistical Regression Model. A Roc Curve was produced using the following code which shows a visual representation of the performance of the used mode, plotting the true positive against the false positive:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
logreg.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000, random_state=16)
```

```
y_prob = logreg.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
```

```
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

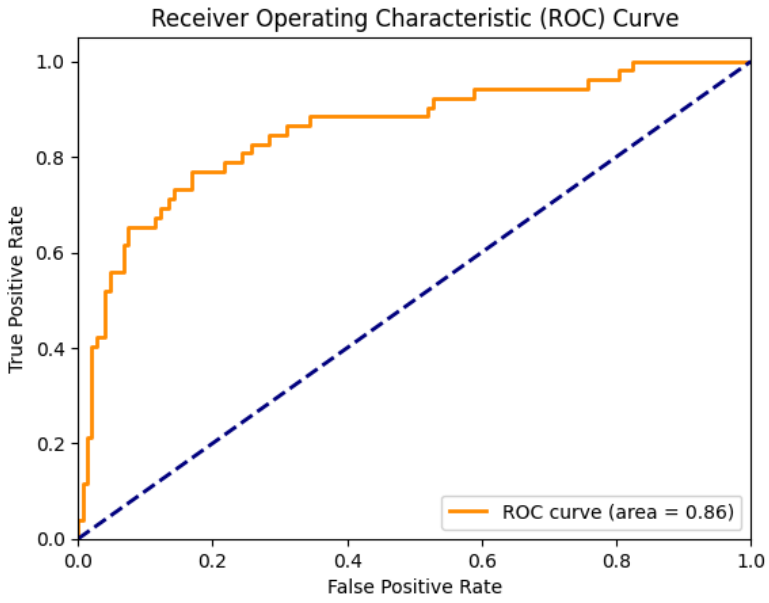


Figure 9: Roc Curve of the Logistical Regression Model

Results from Figure 9 show that the model shows a good discrimination ability. A higher value of up to 1 means the model performs better in discriminating between positive and negative classes. 0.86 represents that the model presents good results better than random guessing.

```
from sklearn.metrics import classification_report
target_names = ['without AFFINITY_CARD', 'with AFFINITY_CARD']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
                     precision    recall  f1-score   support

 low value AFFINITY_CARD       0.81      0.98      0.89       148
high value AFFINITY_CARD       0.86      0.37      0.51        52

               accuracy                           0.82       200
              macro avg       0.84      0.67      0.70       200
           weighted avg       0.83      0.82      0.79       200
```

Results from the code above show that when the model predicts low-value affinity cards, 81% is correct and 86% are correct when predicting high-value affinity cards. The recall scores show that all the models did well at identifying all instances of low-value affinity cards at a rate of 98%

however high-value affinity cards were only at 37%. The overall accuracy of the classifier model is 0.82 shown in the code below:

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

0.82

The following code creates a sigmoid curve on one independent variable, in this case, the AGE variable and the dependent variable:

```
def sigmoid(x):
 return 1 / (1 + np.exp(-x))
feature_name = 'AGE'
X_feature = random_marketing[feature_name].values.reshape(-1, 1)

logreg.fit(X_feature, y)
x_values = np.linspace(np.min(X_feature), np.max(X_feature),
100).reshape(-1, 1)
y_values = sigmoid(logreg.coef_ * x_values + logreg.intercept_).ravel()
plt.figure(figsize=(8, 6))
plt.plot(x_values, y_values, color='red', label='Sigmoid Curve')
plt.scatter(X_feature, y, color='black', label='Data Points')
plt.xlabel(feature_name)
plt.ylabel('Predicted Probability')
plt.title('Logistic Regression Sigmoid Function with Data Points for ' +
feature_name)
plt.legend()
plt.grid(True)
plt.show()
```

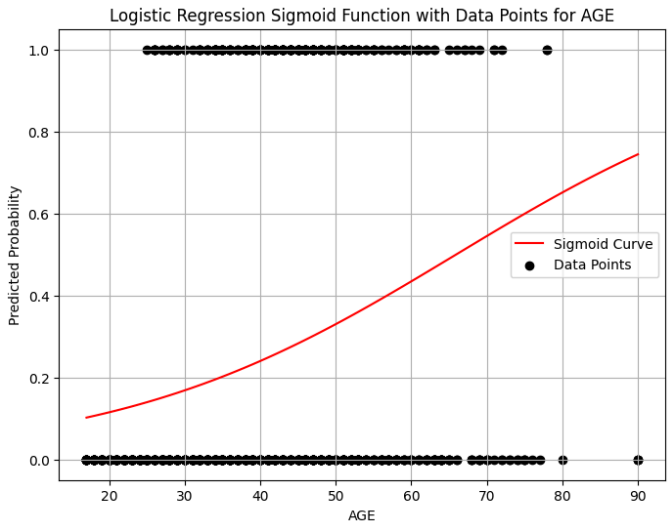


Figure 10: Logistic Regression Sigmoid Function with Data Points for AGE

The second method used was the Support Vector Machine Model. First, it was checked how many rows contained customers with a high-value affinity card (1) and a low-value affinity card (0) using the following code:

```
class_counts = random_marketing['AFFINITY_CARD'].value_counts()
class_counts
```

```
AFFINITY_CARD
0    756
1    244
Name: count, dtype: int64
```

It can be seen that there are significantly more customers present in the dataset with a low-value affinity card compared with a high-value affinity card shown in Figure 11.

```
plt.figure(figsize=(6, 4))
plt.pie(class_counts,     labels=class_counts.index,     autopct='%1.1f%%',
startangle=140)
plt.axis('equal')
plt.show()
```

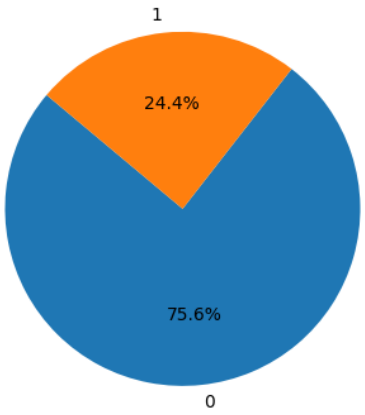


Figure 11: Pie chart of the Affinity card's original occurrence in the dataset

The following code changes the number of affinity cards so that both high-value and low-value affinity cards have the same amount within the data (572 each). This is produced through the code creating more rows based on the information within the original rows.

```
X1 = random_marketing[feature_cols]
y1 = random_marketing.AFFINITY_CARD
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X1 = sc.fit_transform(X)
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1,
test_size=0.25, random_state= 30)
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy=1.0, random_state=42)
X1_train_resampled, y1_train_resampled = smote.fit_resample(X1_train,
y1_train)
print(y1_train_resampled.value_counts())
```

```
AFFINITY_CARD
1    566
0    566
Name: count, dtype: int64
```

The variable AFFINITY_CARD now has an equal amount of outcomes for 1 or 0. The code below creates the classification report in which the results show that the results have lower accuracy than the Logistical Regressions Model, however, the recall and f1-score are significantly higher:

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X1_train_resampled, y1_train_resampled)
y1_pred = svm_model.predict(X1_test)
```

```
print(classification_report(y1_test, y1_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.64      0.74       184
           1       0.43      0.76      0.55        66

    accuracy                           0.67       250
   macro avg       0.66      0.70      0.65       250
weighted avg       0.76      0.67      0.69       250
```

A ROC Curve was then made using the following code:

```
svm_model.fit(X1_train_resampled, y1_train_resampled)
y1_prob = svm_model.decision_function(X1_test)
fpr1, tpr1, thresholds1 = roc_curve(y1_test, y1_prob)
roc_auc = auc(fpr1, tpr1)
plt.figure()
plt.plot(fpr1, tpr1, color='green', lw=2, label='ROC curve (area = %0.2f)'
% roc_auc)
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

Results in Figure 12 show that the ROC Curve has a value of 0.76 which is very similar but slightly lower than the ROC Curve for the Logistical Regressions Model. This suggests that the results in the Logistical Regressions Model are better and more reliable than the Support Vector Machine Model results.

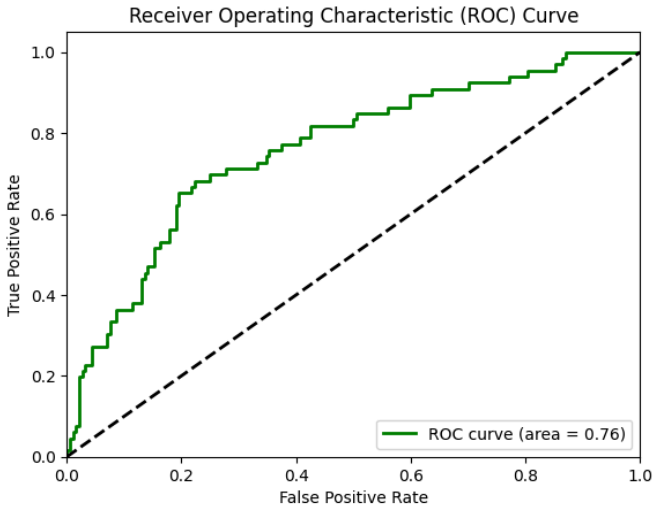


Figure 12: Figure 9: Roc Curve of the Support Vector Machine Model

The accuracy score produced from the code below was also less than the previous model suggesting it does a worse job at predicting the data:

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y1_test, y1_pred)
accuracy
```

0.74

6 - Discussion and Reflection of the Work

The marketing campaign dataset was able to be carefully analysed and predicted using coding in Python. The successful code allowed the data to be cleaned and normalised so that reliable data could be analysed and used to predict the target variable. The partitioning of the data during data mining allowed the data to predict low-value and high-value affinity cards. Based on the accuracy score, the Logistical Regressions model was the best model to predict the target variable and produce a high percentage of correct predictions based on the dataset.

Upon reflection of the work, the assignment provides a good structure to use during the process of analysing data. All the processes during this work were necessary to produce a quality and reliable piece of analysis and should be done using any dataset to ensure it is analysed the best as possible. The data could perhaps be made more reliable at predicting the target variable outcome with some more precise or a change in the coding, creating better results.