PROGRAMMING FOR ANALYTICS

CC7182

SPRING SEMESTER 2022-2023

UNEGBU, OLISA UDOCHUKWU

22020843

MARKETING CAMPAIGN DATA ANALYSIS COURSEWORK

Table of Content

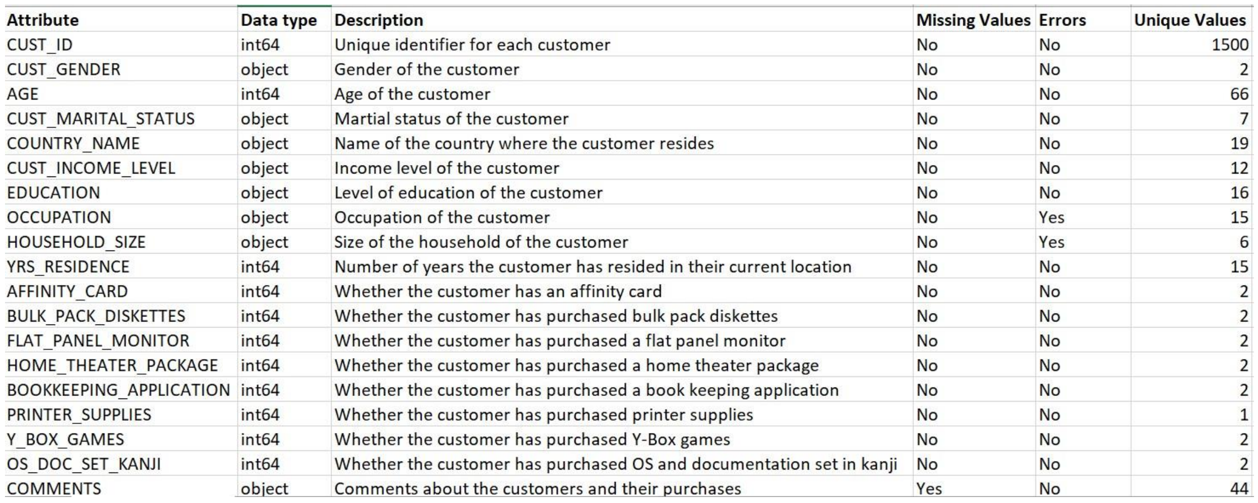
List of Tables

List of codes

List of Figures

1. Data Understanding

The Marketing campaign data is based on the case of a retail company. The dataset contains 1500 observations with 19 attributes. Attribute 11 (AFFINITY_CARD) is the target variable containing 0 and 1 where 0 represents low-value and 1 represents high-value. Table 1 below shows the metadata table marketing campaign data.

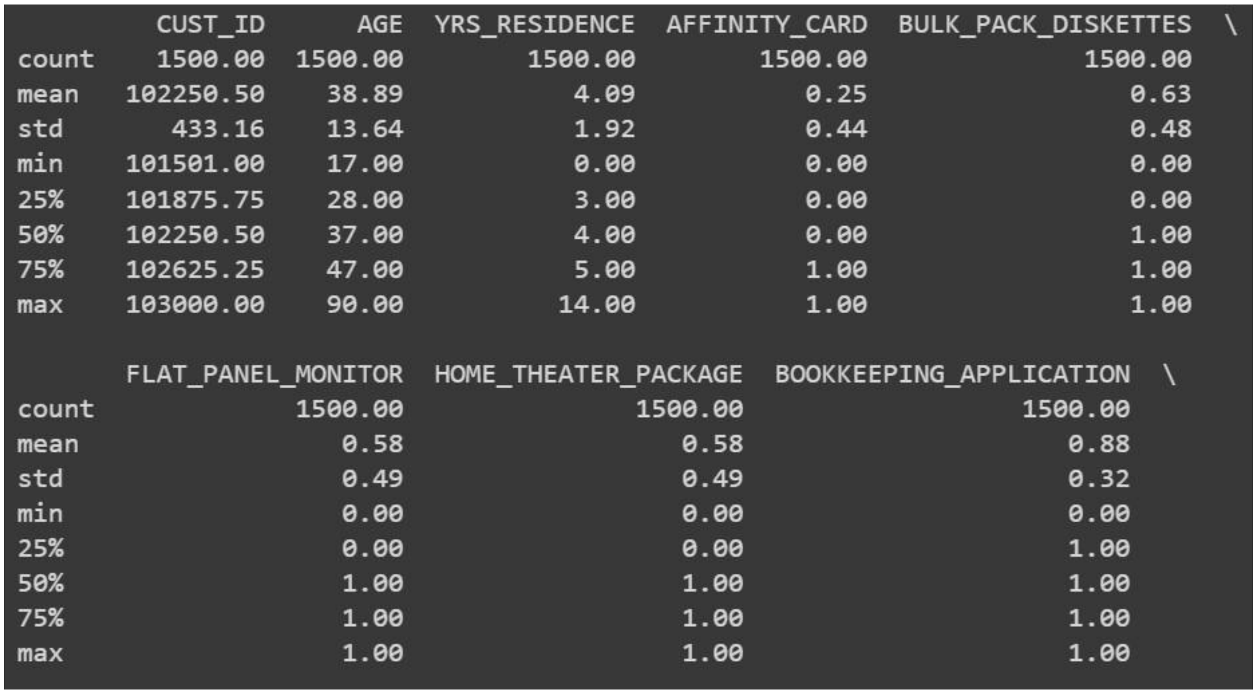


Attribute	Data type	Description	Missing Values	Errors	Unique Values
CUST_ID	int64	Unique identifier for each customer	No	No	1500
CUST_GENDER	object	Gender of the customer	No	No	2
AGE	int64	Age of the customer	No	No	66
CUST_MARITAL_STATUS	object	Martial status of the customer	No	No	7
COUNTRY_NAME	object	Name of the country where the customer resides	No	No	19
CUST_INCOME_LEVEL	object	Income level of the customer	No	No	12
EDUCATION	object	Level of education of the customer	No	No	16
OCCUPATION	object	Occupation of the customer	No	Yes	15
HOUSEHOLD_SIZE	object	Size of the household of the customer	No	Yes	6
YRS_RESIDENCE	int64	Number of years the customer has resided in their current location	No	No	15
AFFINITY_CARD	int64	Whether the customer has an affinity card	No	No	2
BULK_PACK_DISKETTES	int64	Whether the customer has purchased bulk pack diskettes	No	No	2
FLAT_PANEL_MONITOR	int64	Whether the customer has purchased a flat panel monitor	No	No	2
HOME_THEATER_PACKAGE	int64	Whether the customer has purchased a home theater package	No	No	2
BOOKKEEPING_APPLICATION	int64	Whether the customer has purchased a book keeping application	No	No	2
PRINTER_SUPPLIES	int64	Whether the customer has purchased printer supplies	No	No	1
Y_BOX_GAMES	int64	Whether the customer has purchased Y-Box games	No	No	2
OS_DOC_SET_KANJI	int64	Whether the customer has purchased OS and documentation set in kanji	No	No	2
COMMENTS	object	Comments about the customers and their purchases	Yes	No	44

table 1: metadata table.

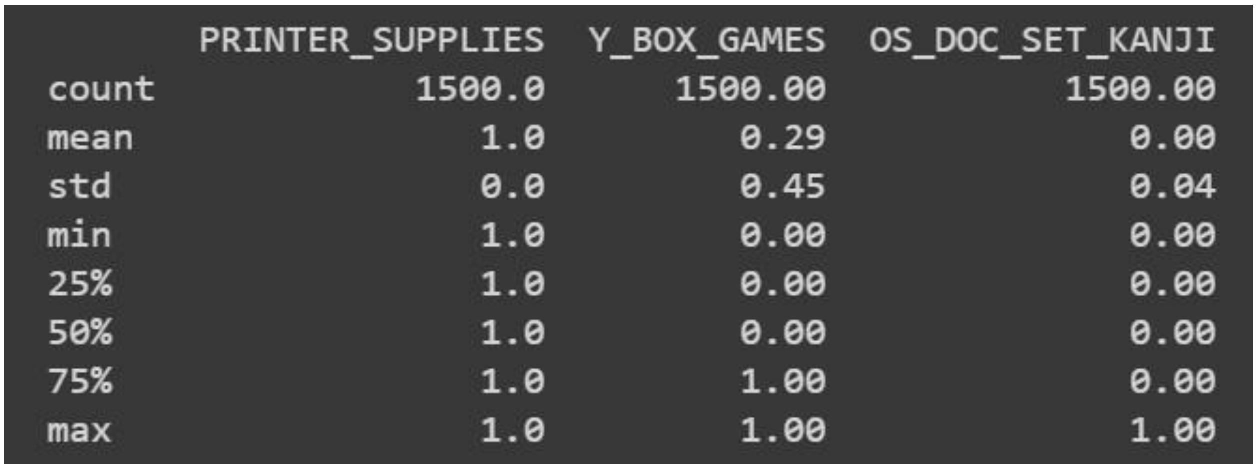
The metadata table for the marketing campaign data contains the attributes names, data type, description, missing values, errors, and unique values.

Table 2a and 2b below shows the data summary which includes count, mean, standard deviation, minimum value, 25%, 50%, 75%, and maximum value for the numeric data attributes.



```
         CUST_ID      AGE  YRS_RESIDENCE  AFFINITY_CARD  BULK_PACK_DISKETTES  \
count    1500.00  1500.00        1500.00        1500.00              1500.00
mean   102250.50    38.89           4.09           0.25                 0.63
std       433.16    13.64           1.92           0.44                 0.48
min    101501.00    17.00           0.00           0.00                 0.00
25%    101875.75    28.00           3.00           0.00                 0.00
50%    102250.50    37.00           4.00           0.00                 1.00
75%    102625.25    47.00           5.00           1.00                 1.00
max    103000.00    90.00          14.00           1.00                 1.00

       FLAT_PANEL_MONITOR  HOME_THEATER_PACKAGE  BOOKKEEPING_APPLICATION  \
count             1500.00               1500.00                  1500.00
mean                 0.58                  0.58                     0.88
std                  0.49                  0.49                     0.32
min                  0.00                  0.00                     0.00
25%                  0.00                  0.00                     1.00
50%                  1.00                  1.00                     1.00
75%                  1.00                  1.00                     1.00
max                  1.00                  1.00                     1.00
```

table 2a: data stats summary.



```
       PRINTER_SUPPLIES  Y_BOX_GAMES  OS_DOC_SET_KANJI
count            1500.0      1500.00           1500.00
mean                1.0         0.29              0.00
std                 0.0         0.45              0.04
min                 1.0         0.00              0.00
25%                 1.0         0.00              0.00
50%                 1.0         0.00              0.00
75%                 1.0         1.00              0.00
max                 1.0         1.00              1.00
```

table 2b: continuation of the data stats summary.

Fig. 1a and 1b shows a histogram chart for the AGE and YRS_RESIDENCE attributes against the target variable (AFFINITY_CARD) respectively. The AGE and YRS_RESIDENCE attributes are the only numeric attributes that are not categorical.

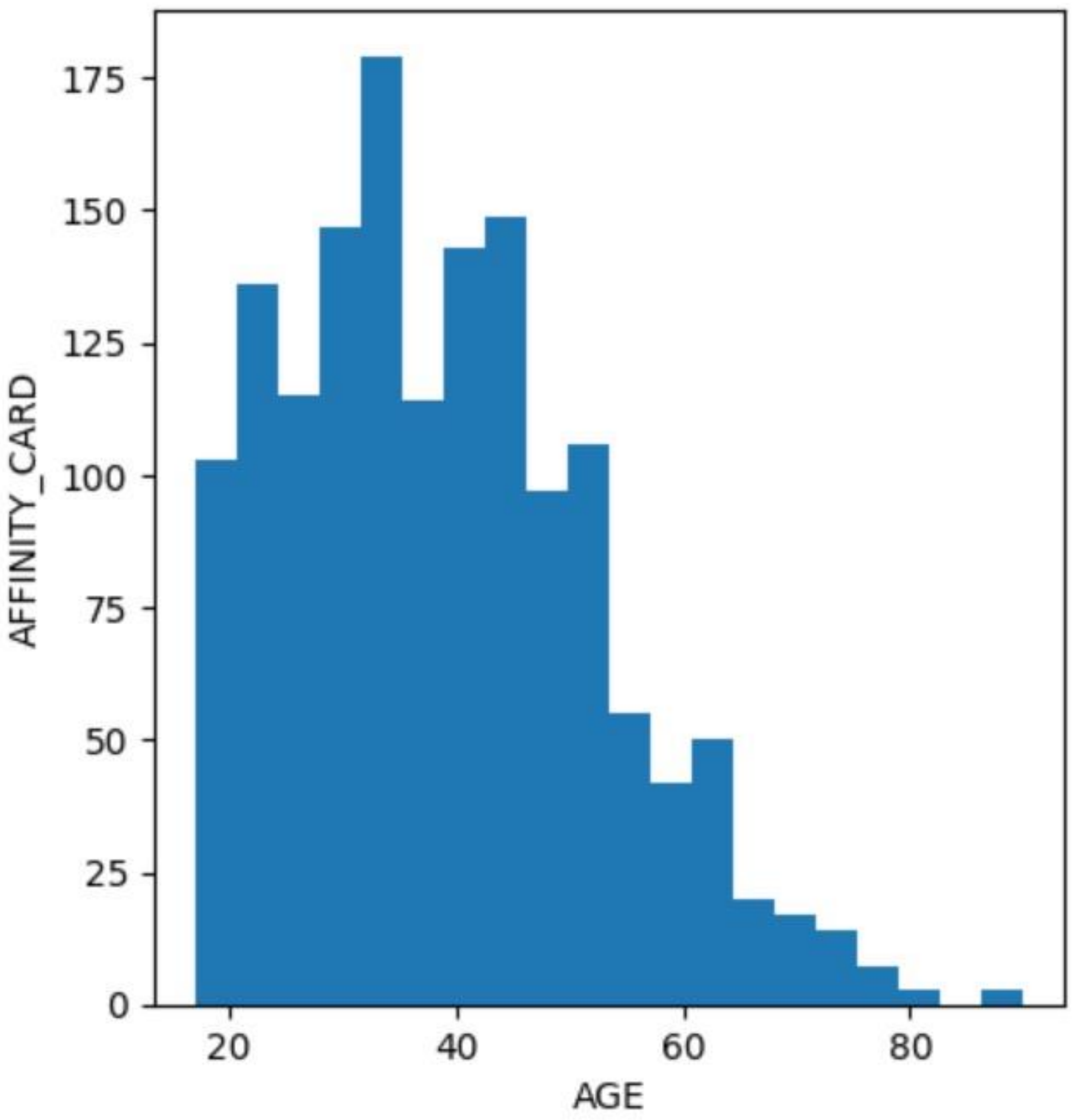


fig. 1a: Histogram chart for AGE vs AFFINITY_CARD.

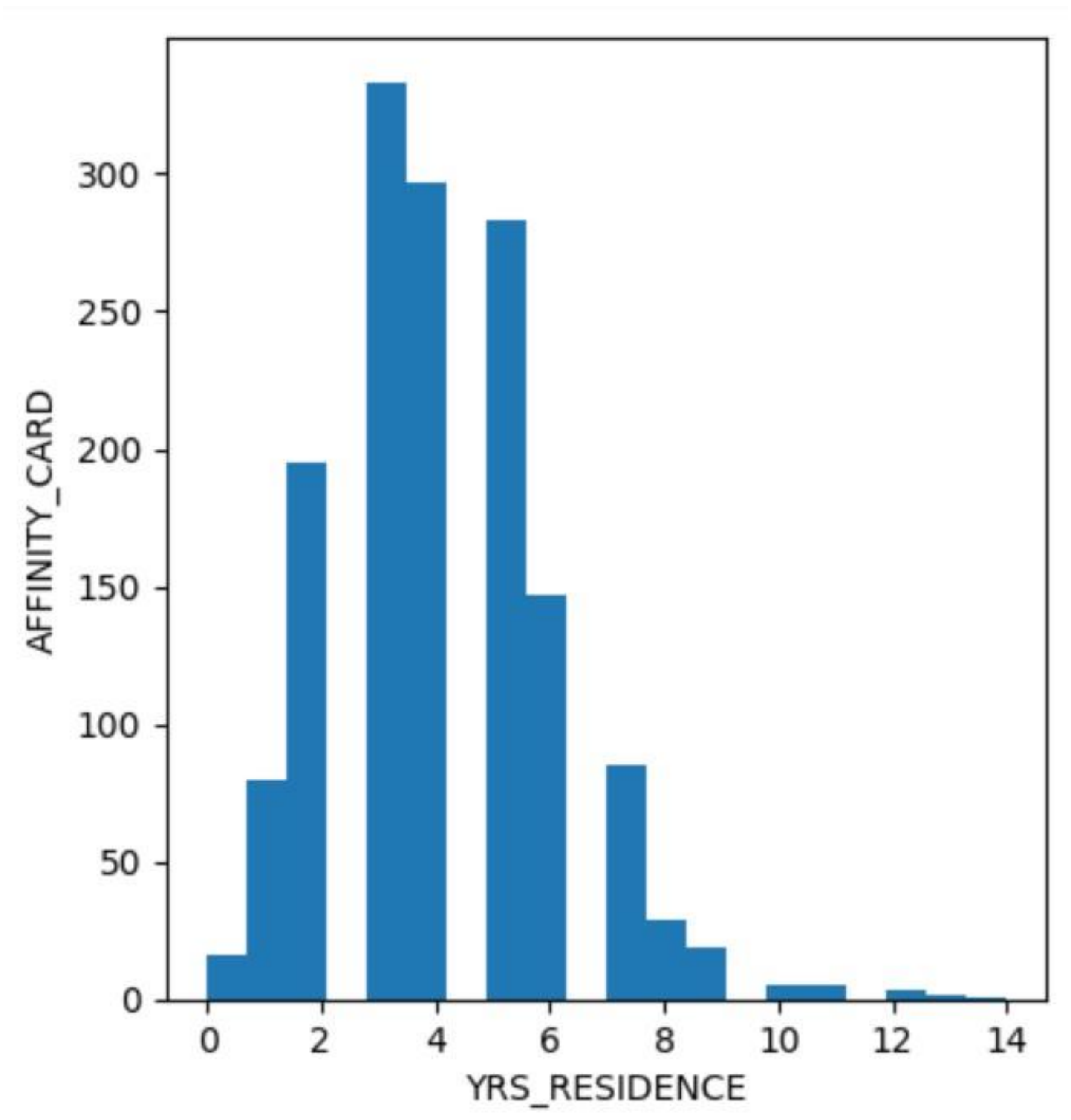
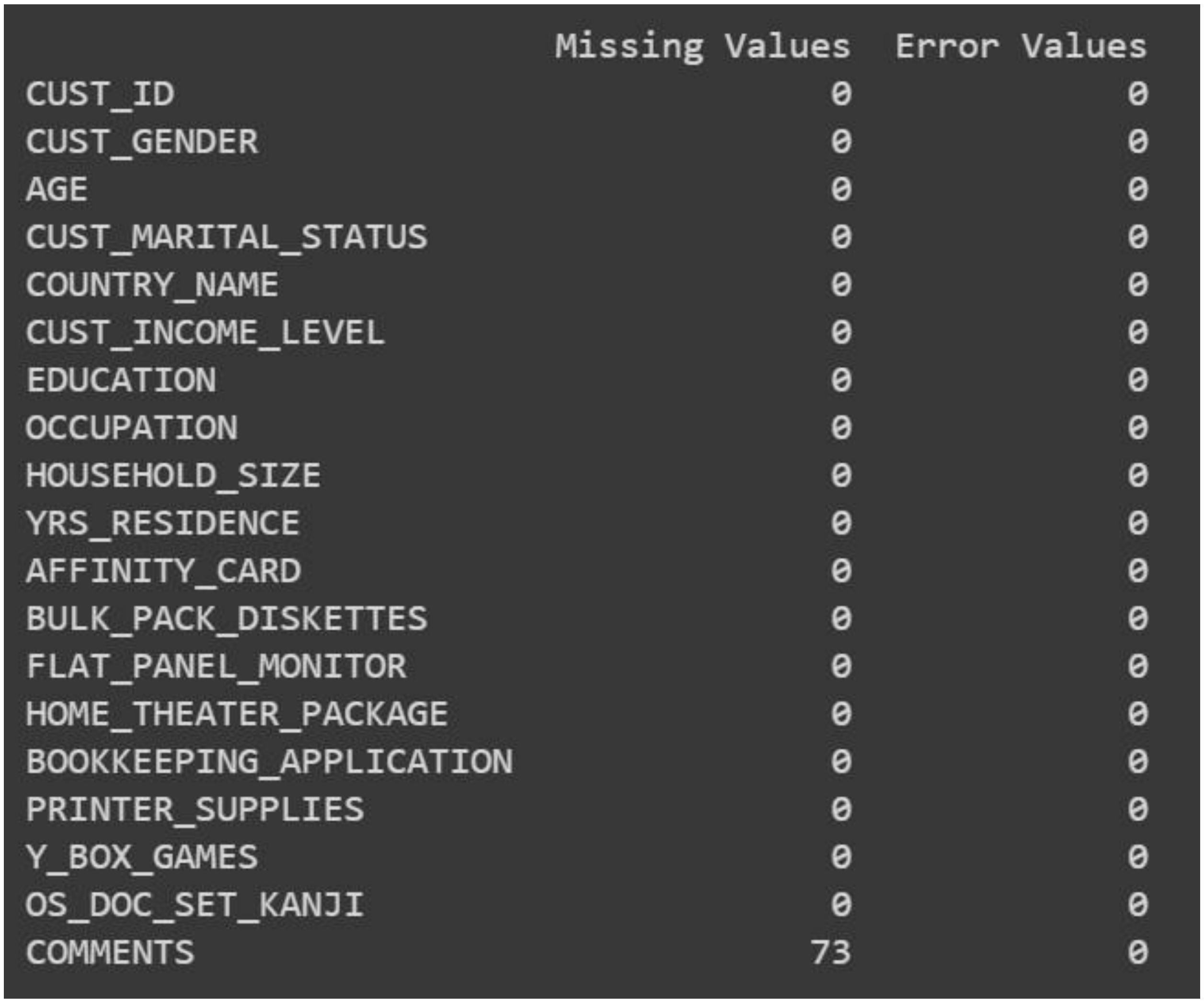


fig. 1b: Histogram chart for YRS vs AFFINITY_CARD

The marketing campaign data contains some missing values and error values. Table 3 below shows the number of missing values and error values.



```
                        Missing Values  Error Values
CUST_ID                              0             0
CUST_GENDER                          0             0
AGE                                  0             0
CUST_MARITAL_STATUS                  0             0
COUNTRY_NAME                         0             0
CUST_INCOME_LEVEL                    0             0
EDUCATION                            0             0
OCCUPATION                           0             0
HOUSEHOLD_SIZE                       0             0
YRS_RESIDENCE                        0             0
AFFINITY_CARD                        0             0
BULK_PACK_DISKETTES                  0             0
FLAT_PANEL_MONITOR                   0             0
HOME_THEATER_PACKAGE                 0             0
BOOKKEEPING_APPLICATION              0             0
PRINTER_SUPPLIES                     0             0
Y_BOX_GAMES                          0             0
OS_DOC_SET_KANJI                     0             0
COMMENTS                            73             0
```

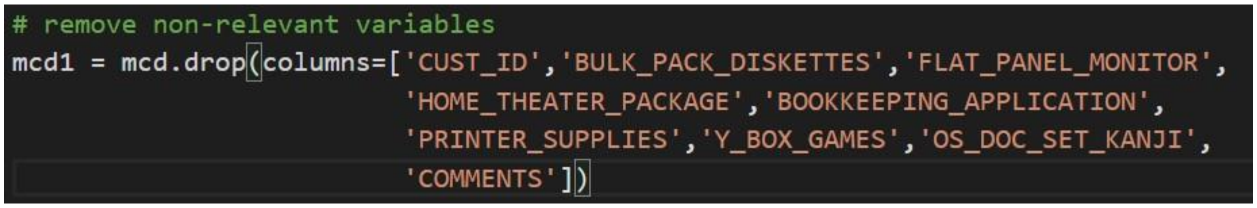
table 3: missing values and error values table.

The COMMENTS variable is the only variable showing missing values (73) and no error was discovered in all other variables as shown in Table 3 above. As I continued my exploratory data analysis (EDA) on excel spreadsheet, I observed that 80 customers are represented with "?" in the OCCUPATION column, 71, 48 and 163 customers are represented with "5-Apr", "8-Jun" and "9+" respectively in the HOUSEHOLD_SIZE column which are not numeric and the PRINTER_SUPPLIES column are represented with "1" all through which could either mean that all customers where given the printer or none has the printer.

2. Data preparation

The marketing campaign data 18 independent variables and 1 dependent variable. Some of the independent variables have little or no effect on whether a customer has a high score value or low score value for the affinity card (dependent/target variable). I dropped the "CUST_ID" attribute because it's just a unique identification number which has no effect on either the high value or low value of the affinity card which is the target variable. I also dropped the "BULK_PACK_DISKETTES", "FLAT_PANEL_MONITOR", HOME_THEATER_PACKAGE", "BOOKKEEPING_APPLICATION", "PRINTER_SUPPLIES", "Y-BOX-GAMES", and "OS_DOC_SET_KANJI" because they are just devices that I think not all customers might need and therefore should have little or no effect on the high or low value of the affinity card. The COMMENTS attribute was also dropped because it requires software programs or platforms that extract valuable insights and information from unstructured text data like RapidMiner, KNIME, IBM Watson, GATE, Lexalytics, OpenText and so on.

Code 1 below shows the python program written to reduce the variables.

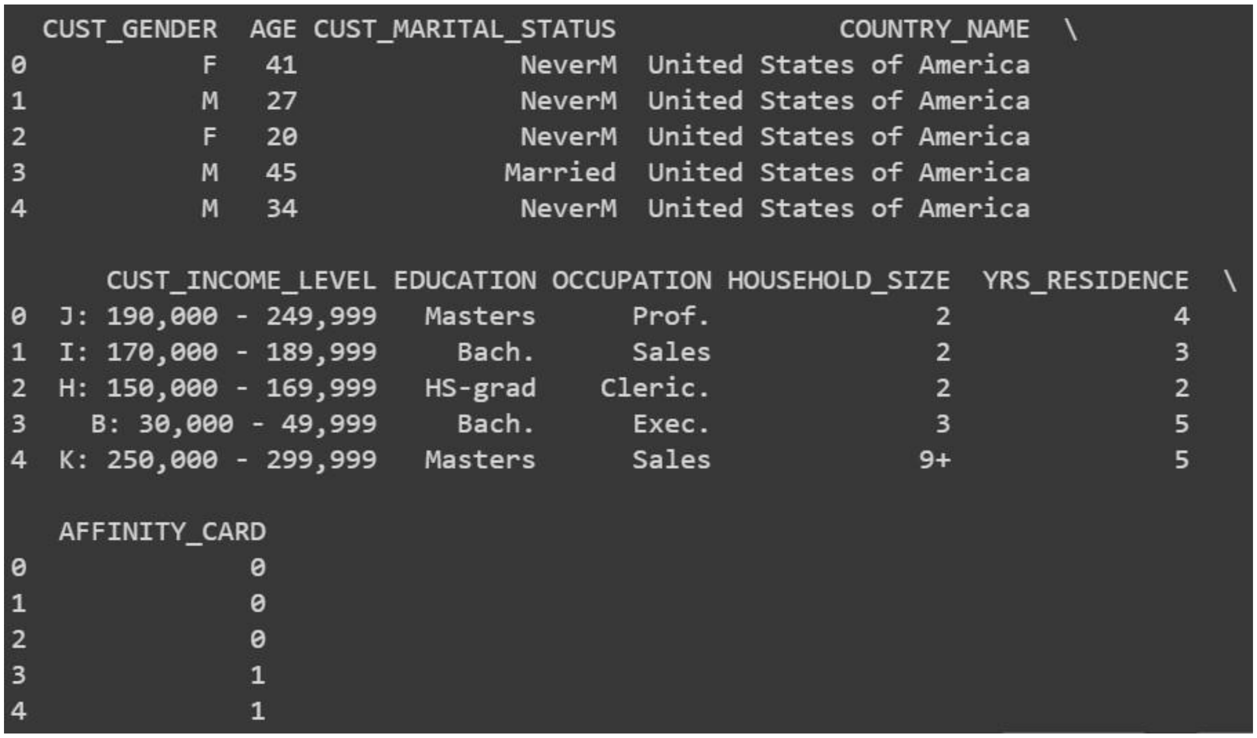


```
# remove non-relevant variables
mcd1 = mcd.drop(columns=['CUST_ID','BULK_PACK_DISKETTES','FLAT_PANEL_MONITOR',
                         'HOME_THEATER_PACKAGE','BOOKKEEPING_APPLICATION',
                         'PRINTER_SUPPLIES','Y_BOX_GAMES','OS_DOC_SET_KANJI',
                         'COMMENTS'])
```

code 1: remove non-relevant variables.

The python program above removes non-relevant variables from the dataset 'mcd'. The drop() function was used to remove the non-relevant variables from the data frame, and the 'columns' argument was used to state the variables to be dropped. The remaining variables (relevant variables) were stored in a new data frame as 'mcd1'.

Table 4 below shows the results of the first five rows of the new dataset after the reductions.

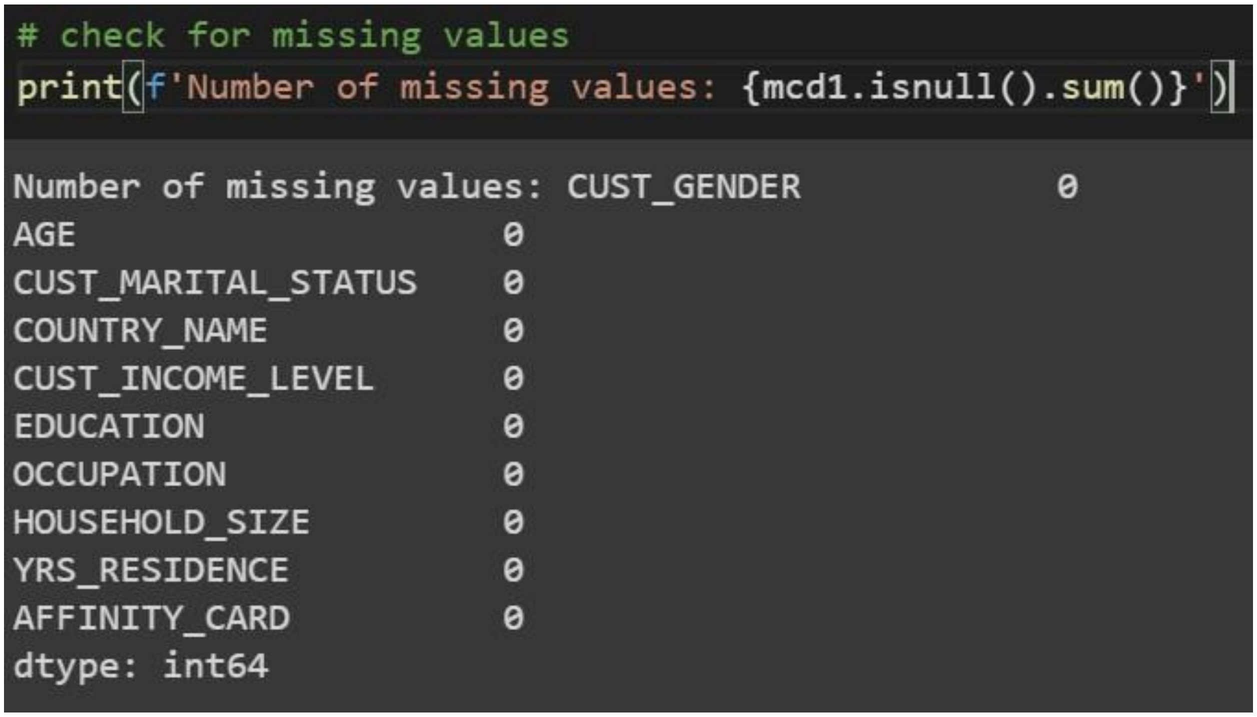


```
  CUST_GENDER  AGE CUST_MARITAL_STATUS              COUNTRY_NAME  \
0           F   41              NeverM  United States of America
1           M   27              NeverM  United States of America
2           F   20              NeverM  United States of America
3           M   45             Married  United States of America
4           M   34              NeverM  United States of America

      CUST_INCOME_LEVEL EDUCATION OCCUPATION HOUSEHOLD_SIZE  YRS_RESIDENCE  \
0  J: 190,000 - 249,999   Masters      Prof.              2              4
1  I: 170,000 - 189,999     Bach.      Sales              2              3
2  H: 150,000 - 169,999   HS-grad    Cleric.              2              2
3    B: 30,000 - 49,999     Bach.      Exec.              3              5
4  K: 250,000 - 299,999   Masters      Sales             9+              5

   AFFINITY_CARD
0              0
1              0
2              0
3              1
4              1
```

table 4: first 5 rows of the new dataset.

The new dataset is now left with CUST_GENDER, AGE, CUST_MARITAL_STATUS, COUNTRY_NAME, CUST_INCOME_LEVEL, EDUCATION, OCCUPATION, HOUSEHOLD_SIZE, YRS_RESIDENCE, and AFFINITY_CARD as shown in table 4 above.

Code 2 below shows the code and its output of the new data calculating the number of missing values.



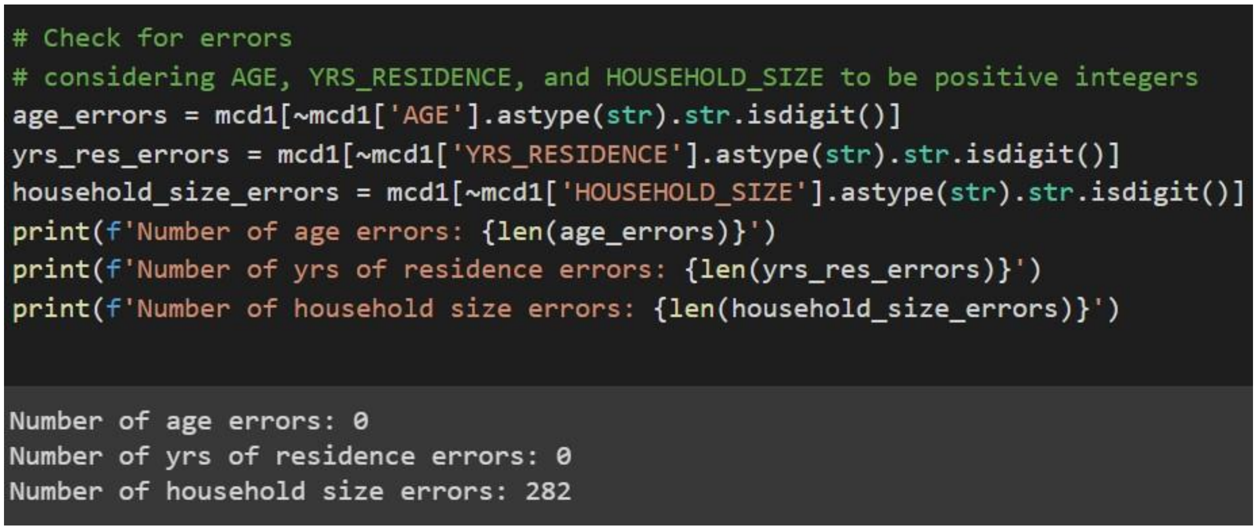
```
# check for missing values
print(f'Number of missing values: {mcd1.isnull().sum()}')
```

```
Number of missing values: 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
dtype: int64
```

code 2: number of missing values.

In calculating the number of missing values in the new dataset (mcd1), isnull() function was used the return same shape of data frame as 'mcd1' where each element is 'True' and otherwise where corresponding elements are 'False'. The sum() function was used to calculate the sum of 'True', that is, counting the number of missing values for each attribute. The 'f-string' argument was used to print the results in a formatted string including the sum of the calculated number of missing values for each variable. The output returns zero (0) missing values for all variables as shown in code 2 above.

Some errors were seen in the HOUSEHOLD_SIZE and OCCUPATION columns. Code 3 below shows the python program written to calculate the number of errors in the AGE, YRS_RESIDENCE, and HOUSEHOLD_SIZE variables since they are all supposed to be positive integers and the results.



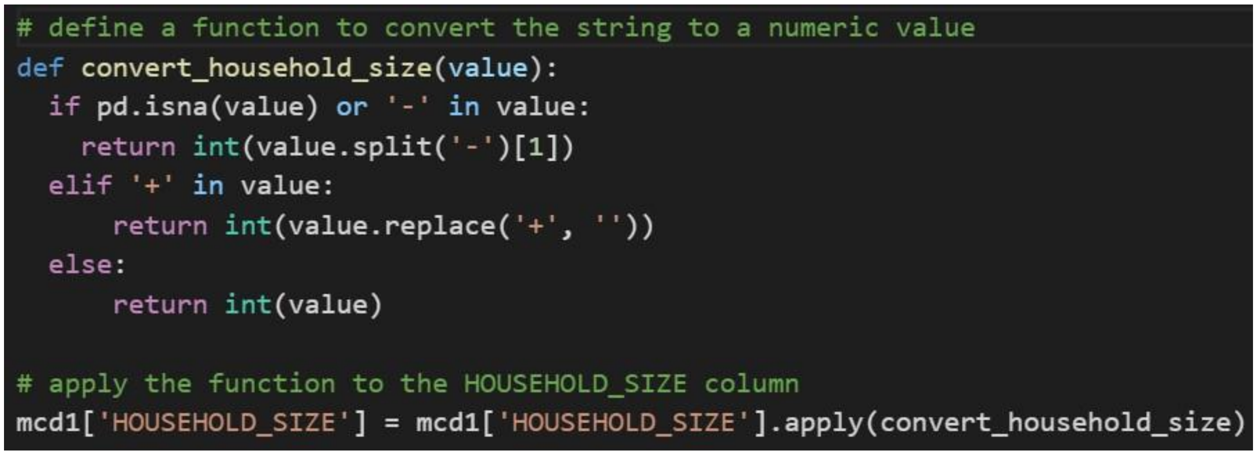
```
# Check for errors
# considering AGE, YRS_RESIDENCE, and HOUSEHOLD_SIZE to be positive integers
age_errors = mcd1[~mcd1['AGE'].astype(str).str.isdigit()]
yrs_res_errors = mcd1[~mcd1['YRS_RESIDENCE'].astype(str).str.isdigit()]
household_size_errors = mcd1[~mcd1['HOUSEHOLD_SIZE'].astype(str).str.isdigit()]
print(f'Number of age errors: {len(age_errors)}')
print(f'Number of yrs of residence errors: {len(yrs_res_errors)}')
print(f'Number of household size errors: {len(household_size_errors)}')
```

```
Number of age errors: 0
Number of yrs of residence errors: 0
Number of household size errors: 282
```

code 3: number of error values.

The python programs shown in code 3 above check for errors in the ‘mcd1’ dataset for 3 variables: AGE, YRS_RESIDENCE, and HOUSEHOLD_SIZE. The code identifies values that are not positive integers since the 3 variables are supposed to be positive integers. The results show 0 for AGE, 0 for YRS_RESIDENCE, and 282 for HOUSEHOLD_SIZE.

The HOUSEHOLD_SIZE attribute has 282 errors and removing 282 observations from 1500 records will amount to removing 18.8% of the data which is a whole lot and will have a negative effect on my result. So, I decided to extract suffix of the errors which are digits as shown in code 4 below.

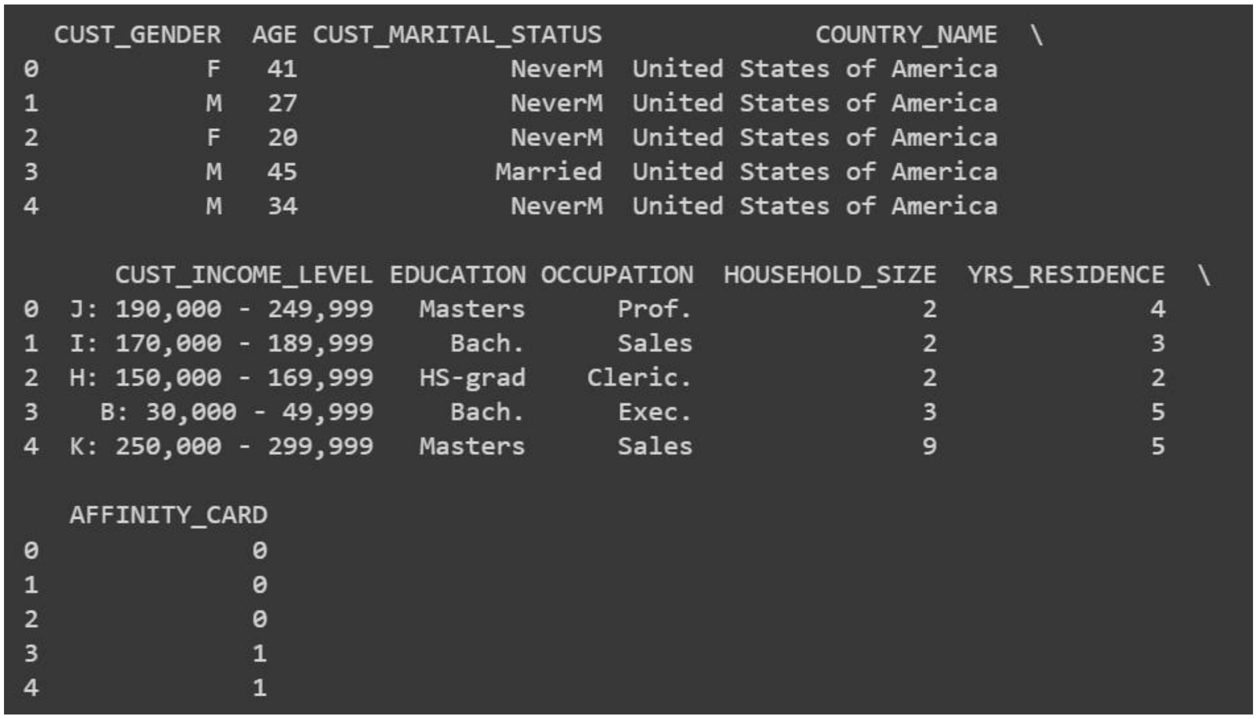


```
# define a function to convert the string to a numeric value
def convert_household_size(value):
  if pd.isna(value) or '-' in value:
    return int(value.split('-')[1])
  elif '+' in value:
      return int(value.replace('+', ''))
  else:
      return int(value)

# apply the function to the HOUSEHOLD_SIZE column
mcd1['HOUSEHOLD_SIZE'] = mcd1['HOUSEHOLD_SIZE'].apply(convert_household_size)
```

code 4: digits extraction.

The python program in code 4 above first defined a function named ‘convert_household_size’ which takes a single argument (value). The function checks whether any entry is missing or contains ‘-‘ or ‘+’ signs. If it contains ‘-‘ sign, it separates the value using ‘-‘ sign and then takes the first part of the separated string. And if it contains the ‘+’ sign, it replaces the ‘+’ sign with an empty string. If none of the signs are present, the function still converts all the values to integer. Then the apply() function was used to apply the covert_household_size() function to the HOUSEHOLD_SIZE variable of the ‘mcd1’ data frame.

Table 5 below shows the first rows of the converted new data.

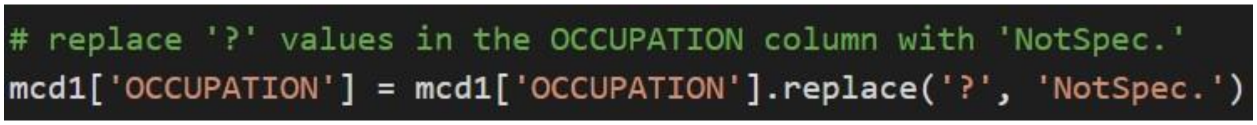


```
  CUST_GENDER  AGE CUST_MARITAL_STATUS             COUNTRY_NAME  \
0           F   41              NeverM  United States of America
1           M   27              NeverM  United States of America
2           F   20              NeverM  United States of America
3           M   45             Married  United States of America
4           M   34              NeverM  United States of America

      CUST_INCOME_LEVEL EDUCATION OCCUPATION  HOUSEHOLD_SIZE  YRS_RESIDENCE  \
0  J: 190,000 - 249,999   Masters      Prof.               2              4
1  I: 170,000 - 189,999     Bach.      Sales               2              3
2  H: 150,000 - 169,999   HS-grad    Cleric.               2              2
3    B: 30,000 - 49,999     Bach.      Exec.               3              5
4  K: 250,000 - 299,999   Masters      Sales               9              5

   AFFINITY_CARD
0              0
1              0
2              0
3              1
4              1
```

table 5: converted new data table.

As shown in table 5 above, the 5th observation of the HOUSEHOLD_SIZE variable shows ‘9’ after conversion from ‘9+’.

The OCCUPATION attribute has 80 customers represented with ‘?’ which I consider that the customers preferred not to state their occupation since there is a category for ‘Other’. I changed the question mark sign ‘?’ to ‘NotSpec.’. Which means Not Specified i.e., the customers did not disclose their occupation. The python program is shown in code 5 below.

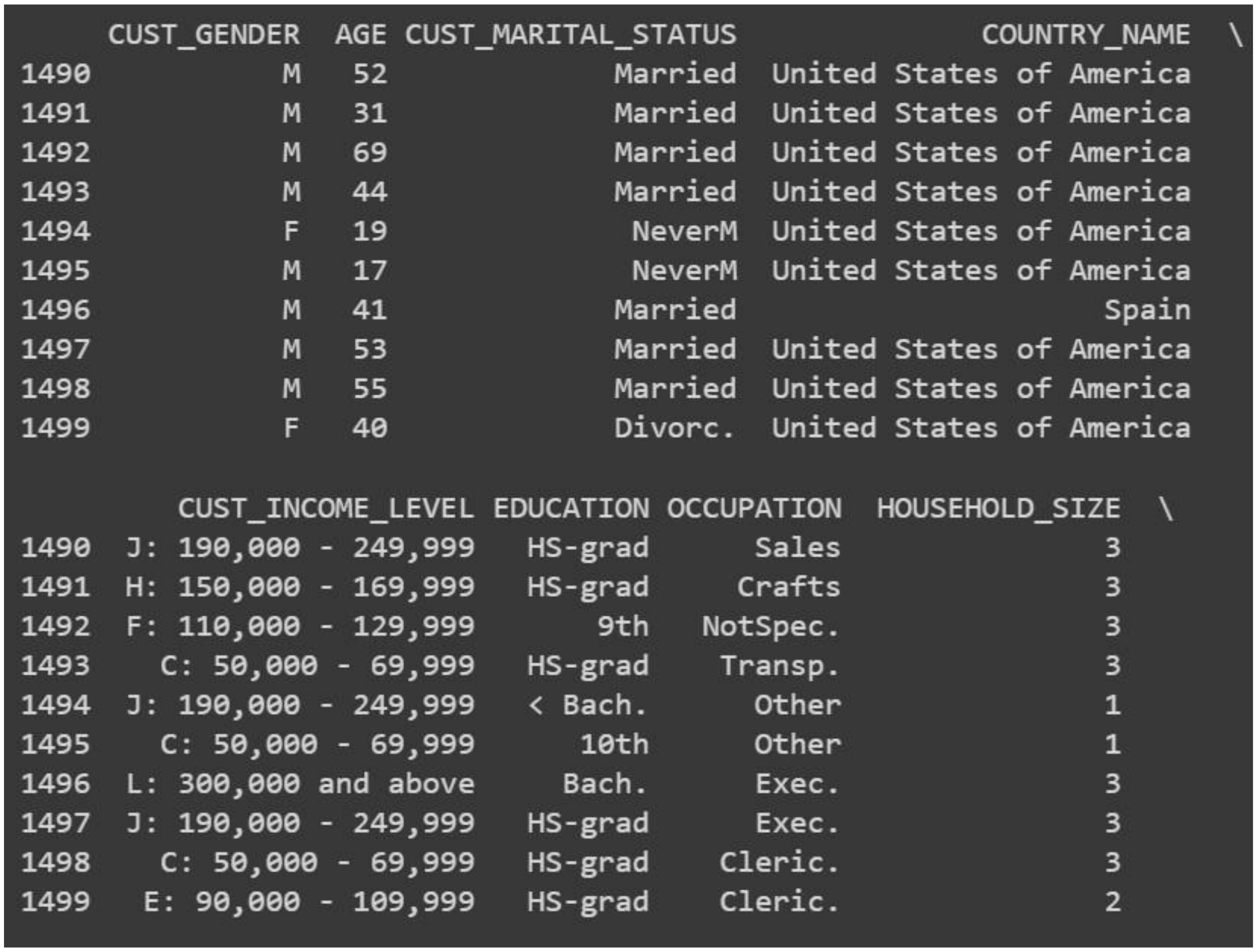


```
# replace '?' values in the OCCUPATION column with 'NotSpec.'
mcd1['OCCUPATION'] = mcd1['OCCUPATION'].replace('?', 'NotSpec.')
```

code 5: replace entry.

To avoid errors in the downstream analyses that may require complete data, I replaced the question mark ‘?’ entry in the OCCUPATION variable of the ‘mcd1’ data frame with the string ‘NotSpec.’. This allows me to handle missing data in a consistent way rather than removing the observations with errors or missing values.

Table 6 below shows the last 10 observations after replacing ‘?’ with ‘NotSpec.’

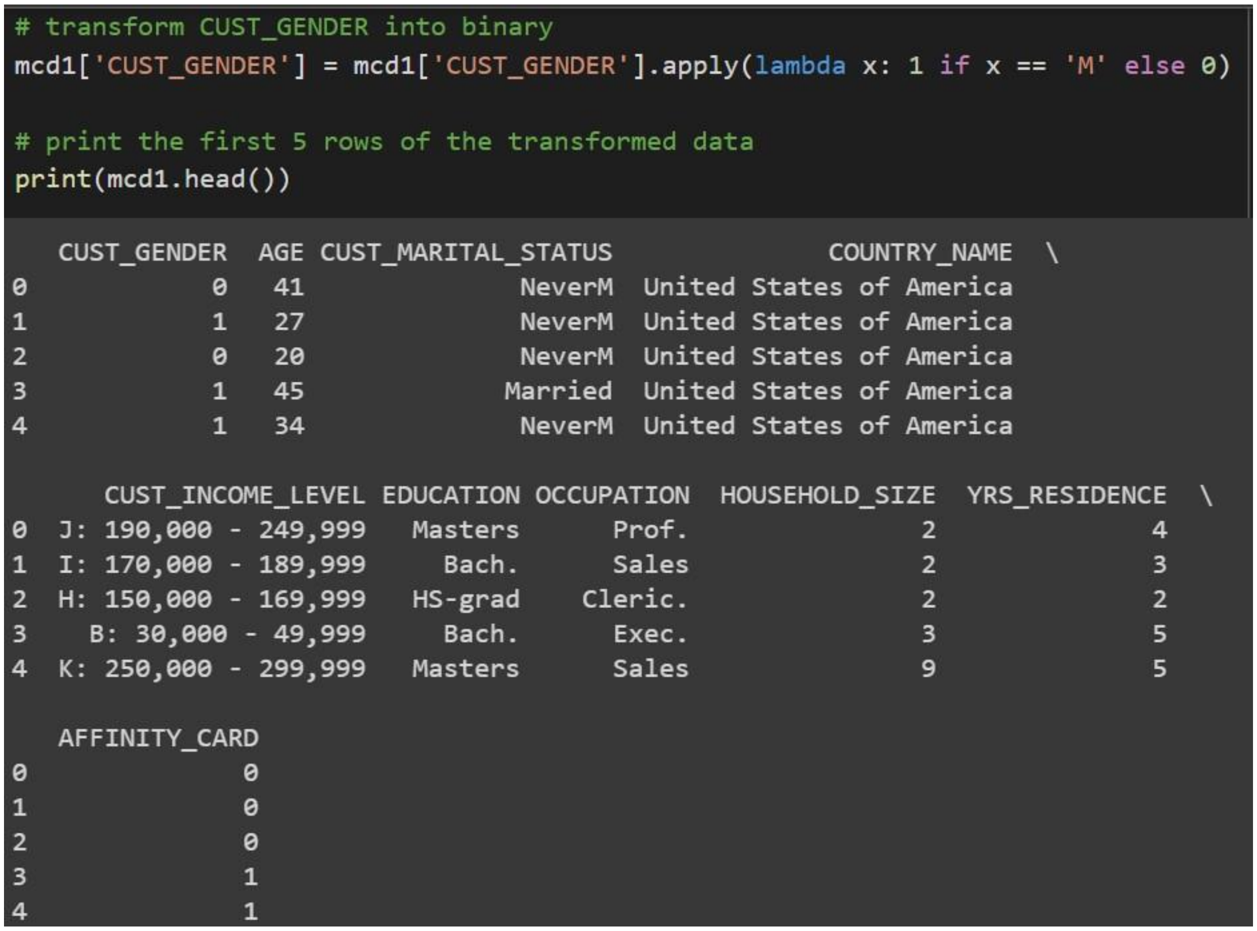


```
     CUST_GENDER  AGE CUST_MARITAL_STATUS             COUNTRY_NAME  \
1490           M   52             Married  United States of America
1491           M   31             Married  United States of America
1492           M   69             Married  United States of America
1493           M   44             Married  United States of America
1494           F   19              NeverM  United States of America
1495           M   17              NeverM  United States of America
1496           M   41             Married                     Spain
1497           M   53             Married  United States of America
1498           M   55             Married  United States of America
1499           F   40             Divorc.  United States of America

         CUST_INCOME_LEVEL EDUCATION OCCUPATION  HOUSEHOLD_SIZE  \
1490  J: 190,000 - 249,999   HS-grad      Sales               3
1491  H: 150,000 - 169,999   HS-grad     Crafts               3
1492  F: 110,000 - 129,999       9th   NotSpec.               3
1493    C: 50,000 - 69,999   HS-grad    Transp.               3
1494  J: 190,000 - 249,999   < Bach.      Other               1
1495    C: 50,000 - 69,999      10th      Other               1
1496  L: 300,000 and above     Bach.      Exec.               3
1497  J: 190,000 - 249,999   HS-grad      Exec.               3
1498    C: 50,000 - 69,999   HS-grad    Cleric.               3
1499   E: 90,000 - 109,999   HS-grad    Cleric.               2
```

table 6: new data results.

Transform Variables

2a) The python program that transforms CUST_GENDER variable into binary where F = 0 and M = 1 is shown in code 6 below together with its output.



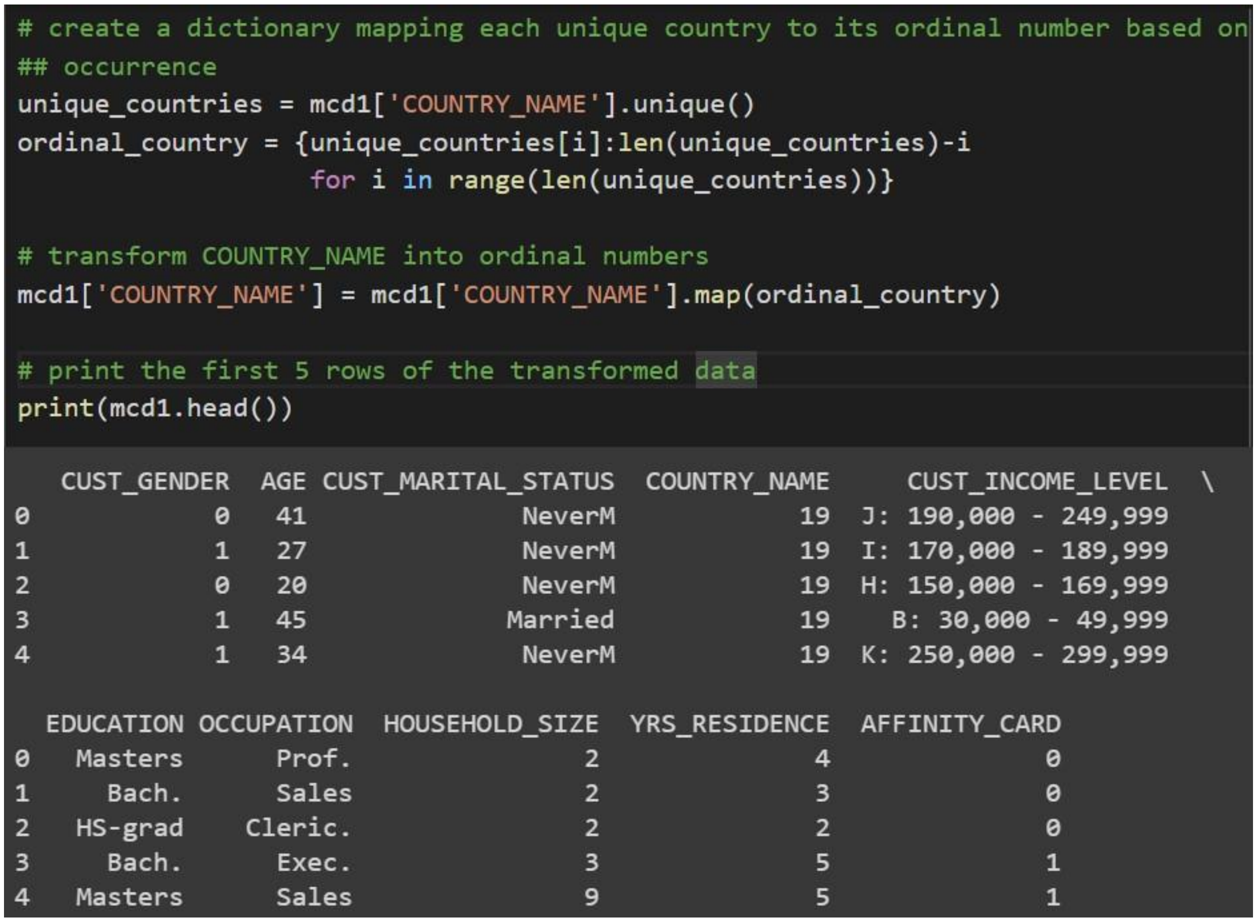
```
# transform CUST_GENDER into binary
mcd1['CUST_GENDER'] = mcd1['CUST_GENDER'].apply(lambda x: 1 if x == 'M' else 0)

# print the first 5 rows of the transformed data
print(mcd1.head())
```

```
   CUST_GENDER  AGE CUST_MARITAL_STATUS             COUNTRY_NAME  \
0            0   41              NeverM  United States of America
1            1   27              NeverM  United States of America
2            0   20              NeverM  United States of America
3            1   45             Married  United States of America
4            1   34              NeverM  United States of America

      CUST_INCOME_LEVEL EDUCATION OCCUPATION  HOUSEHOLD_SIZE  YRS_RESIDENCE  \
0  J: 190,000 - 249,999   Masters      Prof.               2              4
1  I: 170,000 - 189,999     Bach.      Sales               2              3
2  H: 150,000 - 169,999   HS-grad    Cleric.               2              2
3    B: 30,000 - 49,999     Bach.      Exec.               3              5
4  K: 250,000 - 299,999   Masters      Sales               9              5

   AFFINITY_CARD
0              0
1              0
2              0
3              1
4              1
```

code 6: transformed CUST_GENDER data.

The python program in code 6 above transforms the CUST_GENDER variable in 'mcd1' data frame into binary values. It used a lamda() function to convert 'M' values to '1' and all other values to '0' in the CUST_GENDER variable which replaces the original variable in the 'mcd1' data frame. The print() function is used to display the first 5 observations of the transformed data.

2b) Code 7 below shows the python program and its results that transforms the COUNTRY_NAME variable into ordinal number based on their occurrence in a descending order.



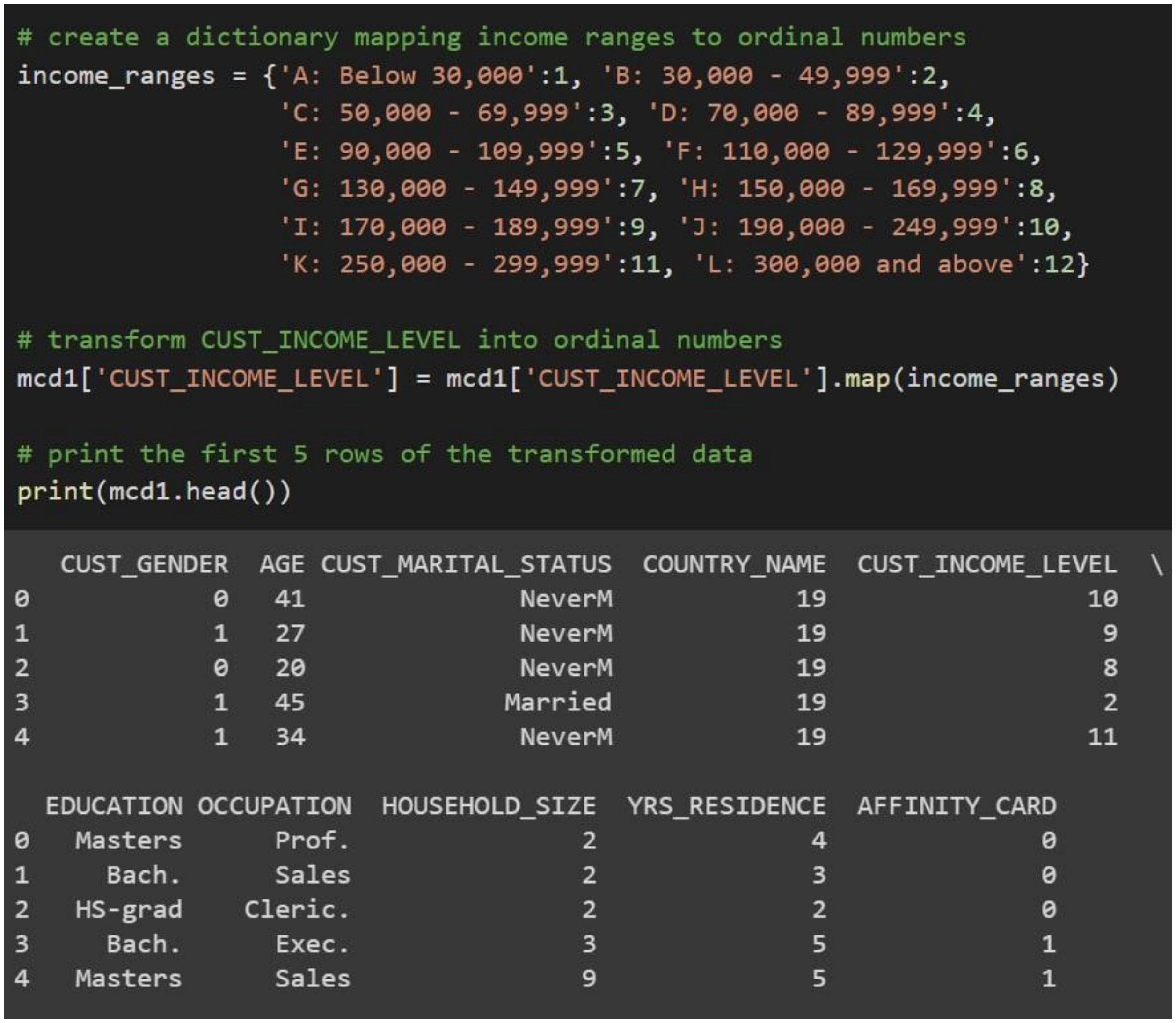
```
# create a dictionary mapping each unique country to its ordinal number based on
## occurrence
unique_countries = mcd1['COUNTRY_NAME'].unique()
ordinal_country = {unique_countries[i]:len(unique_countries)-i
                   for i in range(len(unique_countries))}

# transform COUNTRY_NAME into ordinal numbers
mcd1['COUNTRY_NAME'] = mcd1['COUNTRY_NAME'].map(ordinal_country)

# print the first 5 rows of the transformed data
print(mcd1.head())
```

```
   CUST_GENDER  AGE CUST_MARITAL_STATUS  COUNTRY_NAME     CUST_INCOME_LEVEL  \
0            0   41              NeverM            19  J: 190,000 - 249,999
1            1   27              NeverM            19  I: 170,000 - 189,999
2            0   20              NeverM            19  H: 150,000 - 169,999
3            1   45             Married            19    B: 30,000 - 49,999
4            1   34              NeverM            19  K: 250,000 - 299,999

  EDUCATION OCCUPATION  HOUSEHOLD_SIZE  YRS_RESIDENCE  AFFINITY_CARD
0   Masters      Prof.               2              4              0
1     Bach.      Sales               2              3              0
2   HS-grad    Cleric.               2              2              0
3     Bach.      Exec.               3              5              1
4   Masters      Sales               9              5              1
```

code 7: transformed COUNTRY_NAME data.

As shown in code 7 above, the unique_countries() function was used to list all the unique values in the COUNTRY_NAME variable of the 'mcd1' data frame. The 'ordinal_country' dictionary was used to assign each unique country in the unique_countries() function, an ordinal number based on its position in the list in a descending order. The map() function was then used to replace the country names in the COUNTRY_NAME variable of the 'mcd1' dataframe with their corresponding ordinal numbers from the 'ordinal_country' dictionary. The print() function was used to show the first 5 observations of the transformed data.

2c) The python program that transforms the CUST_INCOME_LEVEL variable into ordinal numbers from 1 to 12 accordingly and its output is shown in code 8 below.



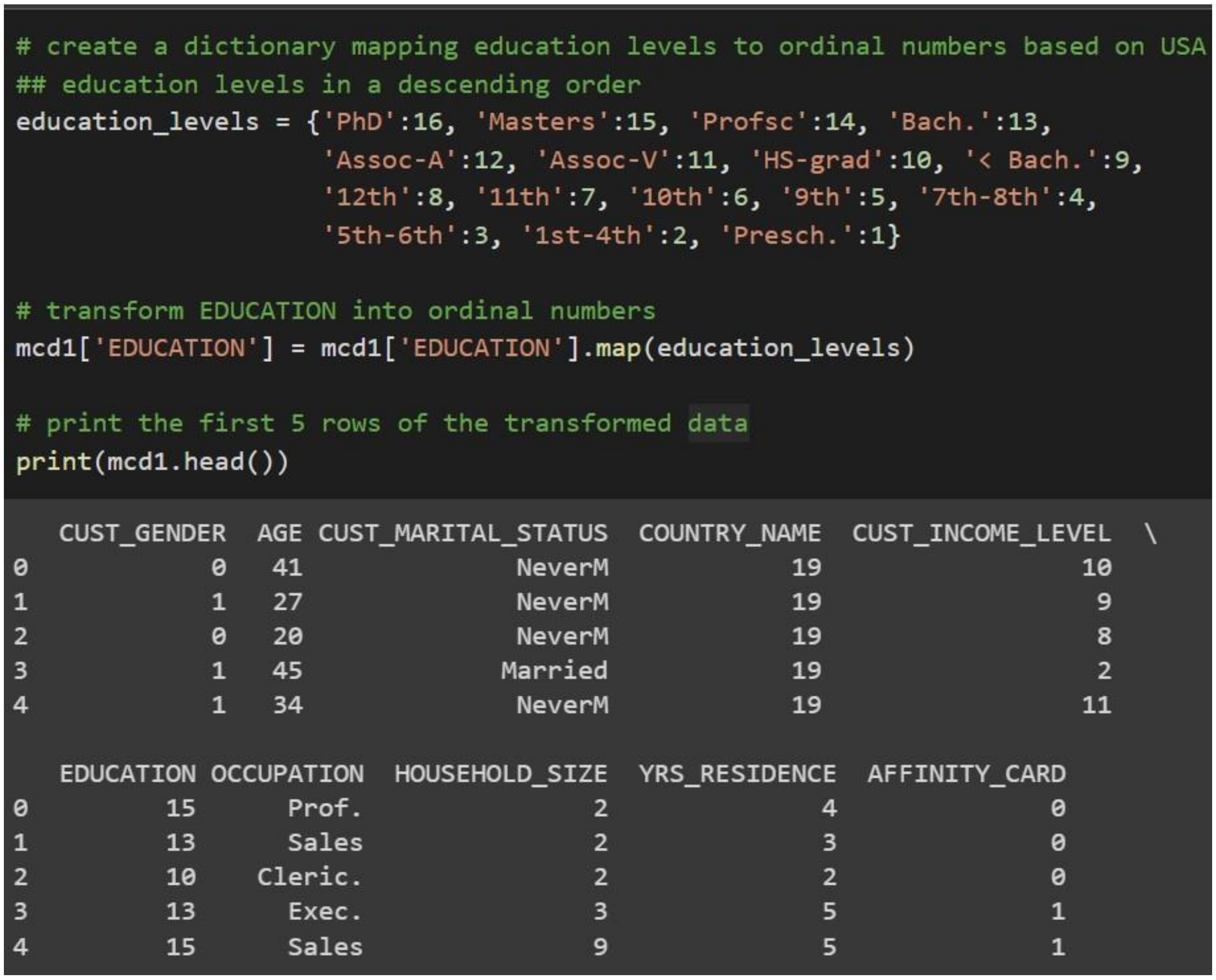
```
# create a dictionary mapping income ranges to ordinal numbers
income_ranges = {'A: Below 30,000':1, 'B: 30,000 - 49,999':2,
                 'C: 50,000 - 69,999':3, 'D: 70,000 - 89,999':4,
                 'E: 90,000 - 109,999':5, 'F: 110,000 - 129,999':6,
                 'G: 130,000 - 149,999':7, '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}

# transform CUST_INCOME_LEVEL into ordinal numbers
mcd1['CUST_INCOME_LEVEL'] = mcd1['CUST_INCOME_LEVEL'].map(income_ranges)

# print the first 5 rows of the transformed data
print(mcd1.head())
```

```
   CUST_GENDER  AGE CUST_MARITAL_STATUS  COUNTRY_NAME  CUST_INCOME_LEVEL  \
0            0   41              NeverM            19                 10
1            1   27              NeverM            19                  9
2            0   20              NeverM            19                  8
3            1   45             Married            19                  2
4            1   34              NeverM            19                 11

  EDUCATION OCCUPATION  HOUSEHOLD_SIZE  YRS_RESIDENCE  AFFINITY_CARD
0   Masters      Prof.               2              4              0
1     Bach.      Sales               2              3              0
2   HS-grad    Cleric.               2              2              0
3     Bach.      Exec.               3              5              1
4   Masters      Sales               9              5              1
```

code 8: transformed CUST_INCOME_LEVEL data.

The python program in code 8 above creates a dictionary first called ‘income_ranges’ that maps the different income ranges to ordinal numbers. Then the map() function was used to apply the mapping to the CUST_INCOME_LEVEL variable of the ‘mcd1’ data frame which replaces the original income range value with the corresponding ordinal numbers. The print() function was used to display the transformed data.

2d) Using the USA education level, code 9 below shows the python program that transforms the EDUCATION variable into ordinal numbers in a descending order and its result.



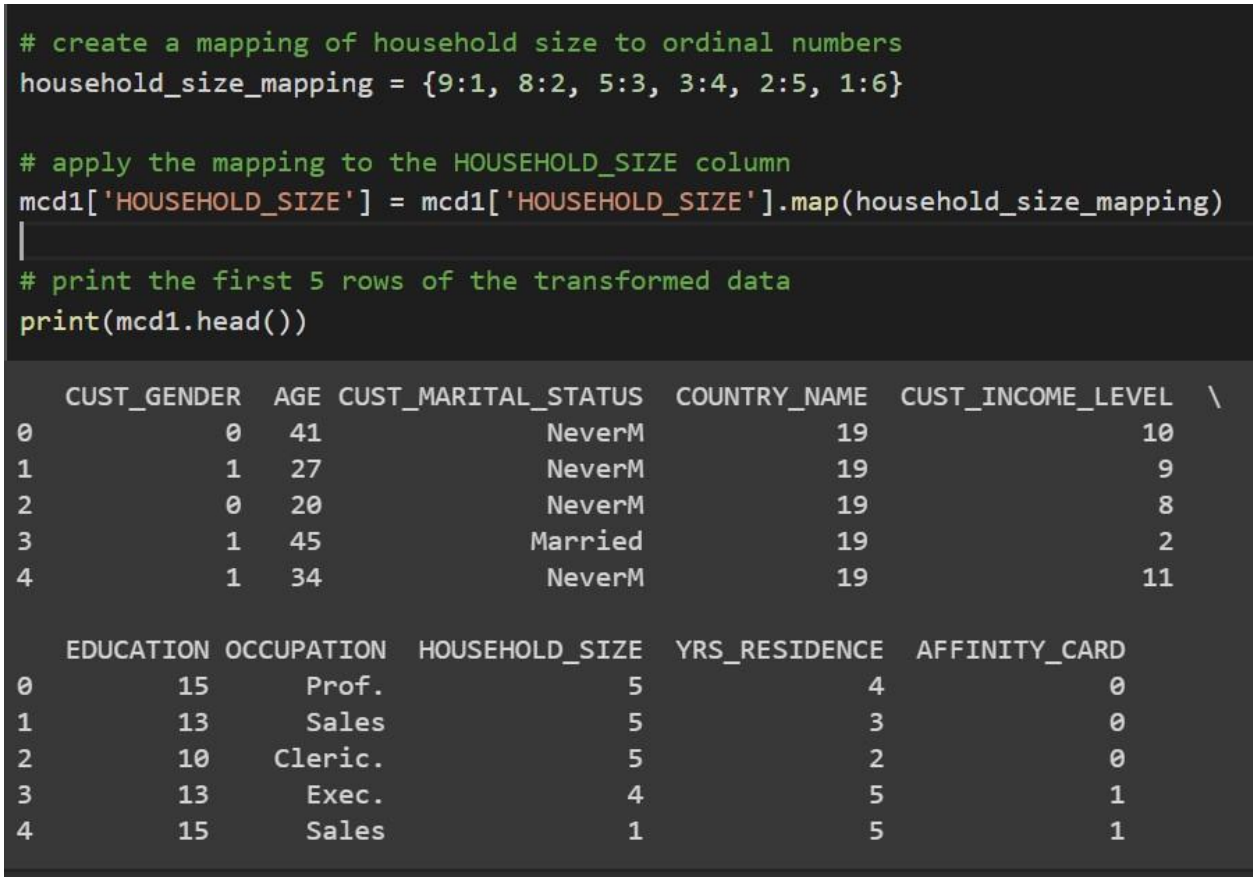
```
# create a dictionary mapping education levels to ordinal numbers based on USA
## education levels in a descending order
education_levels = {'PhD':16, 'Masters':15, 'Profsc':14, 'Bach.':13,
                    'Assoc-A':12, 'Assoc-V':11, 'HS-grad':10, '< Bach.':9,
                    '12th':8, '11th':7, '10th':6, '9th':5, '7th-8th':4,
                    '5th-6th':3, '1st-4th':2, 'Presch.':1}

# transform EDUCATION into ordinal numbers
mcd1['EDUCATION'] = mcd1['EDUCATION'].map(education_levels)

# print the first 5 rows of the transformed data
print(mcd1.head())
```

```
   CUST_GENDER  AGE CUST_MARITAL_STATUS  COUNTRY_NAME  CUST_INCOME_LEVEL  \
0            0   41              NeverM            19                 10
1            1   27              NeverM            19                  9
2            0   20              NeverM            19                  8
3            1   45             Married            19                  2
4            1   34              NeverM            19                 11

   EDUCATION OCCUPATION  HOUSEHOLD_SIZE  YRS_RESIDENCE  AFFINITY_CARD
0         15      Prof.               2              4              0
1         13      Sales               2              3              0
2         10    Cleric.               2              2              0
3         13      Exec.               3              5              1
4         15      Sales               9              5              1
```

code 9: transformed EDUCATION data.

The python program as shown in code 9 above creates a dictionary first named 'education_levels' which maps the different education levels according to the USA education levels to their corresponding ordinal numbers in a descending order. The map() function is then used to apply the 'education_levels' of the dictionary to the EDUCATION variable which replaces the original education levels with their corresponding ordinal numbers in the 'mcd1' data frame. The print() function is used to show the first 5 observations of the transformed data.

2e) During the first phase of this coursework which is data understanding, I understand the HOUSEHOLD_SIZE attribute to be the size of household of the customer as stated in my metadata table which is contrary to question 2e seeing it as number of rooms. I decided to use household size rather than number of rooms. The python program that transforms the HOUSEHOLD_SIZE into ordinal numbers based on the number of households and its result is shown in code 10 below.



```
# create a mapping of household size to ordinal numbers
household_size_mapping = {9:1, 8:2, 5:3, 3:4, 2:5, 1:6}

# apply the mapping to the HOUSEHOLD_SIZE column
mcd1['HOUSEHOLD_SIZE'] = mcd1['HOUSEHOLD_SIZE'].map(household_size_mapping)

# print the first 5 rows of the transformed data
print(mcd1.head())
```

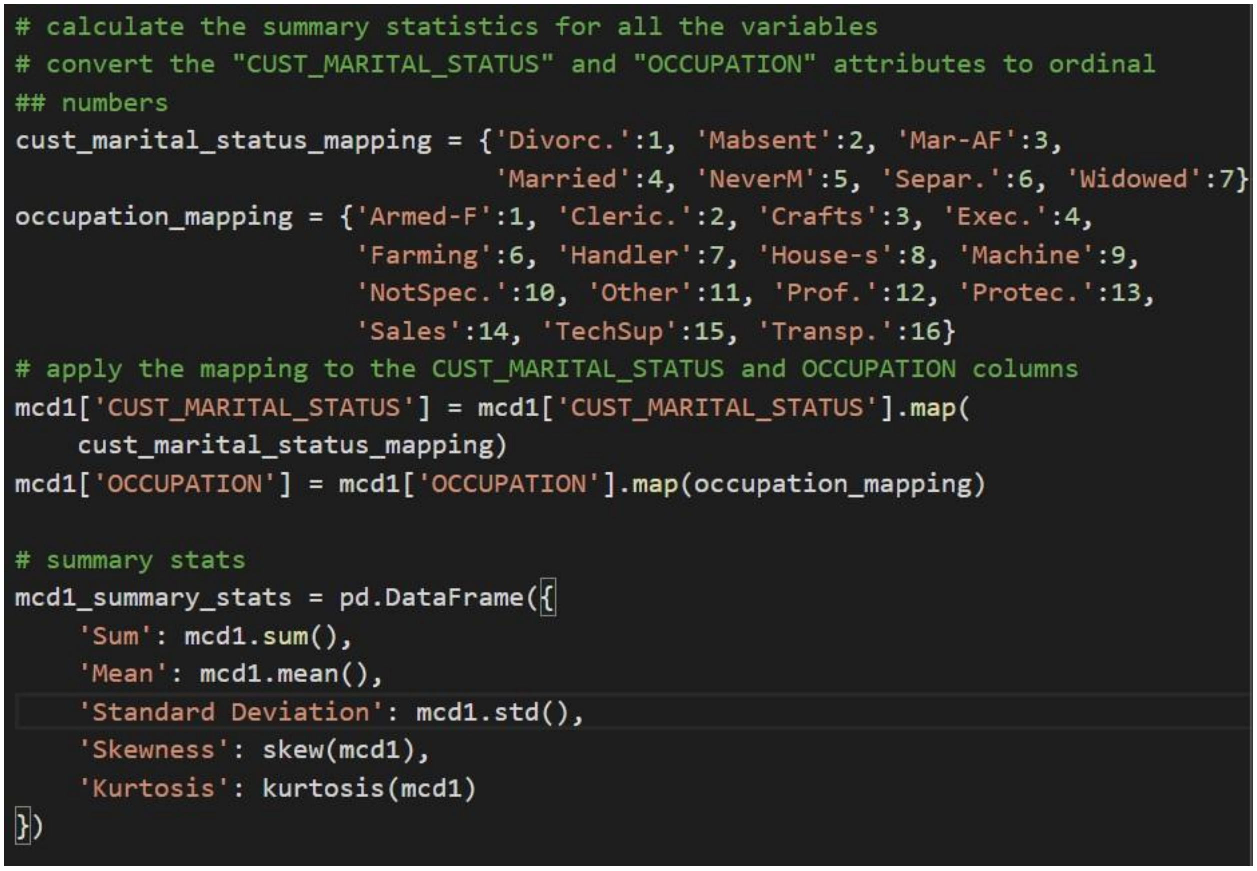
```
   CUST_GENDER  AGE CUST_MARITAL_STATUS  COUNTRY_NAME  CUST_INCOME_LEVEL  \
0            0   41              NeverM            19                 10
1            1   27              NeverM            19                  9
2            0   20              NeverM            19                  8
3            1   45             Married            19                  2
4            1   34              NeverM            19                 11

   EDUCATION OCCUPATION  HOUSEHOLD_SIZE  YRS_RESIDENCE  AFFINITY_CARD
0         15      Prof.               5              4              0
1         13      Sales               5              3              0
2         10    Cleric.               5              2              0
3         13      Exec.               4              5              1
4         15      Sales               1              5              1
```

code 10: transformed HOUSEHOLD_SIZE data.

As shown in code 10 above, the python program first creates mapping of household size to ordinal numbers where households with bigger sizes are assigned to lower numbers and vice versa. A dictionary with the household size as key and the ordinal numbers as the value was used to define the mapping. The map() function was used to apply the program to the HOUSEHOLD variable of the 'mcd1' data frame which replaces the HOUSEHOLD_SIZE values with their corresponding ordinal numbers based on the mapping.

3. Data analysis

The python program that shows the summary statistics of the sum, mean, standard deviation, skewness, and kurtosis of all variables is shown in code 11 below.

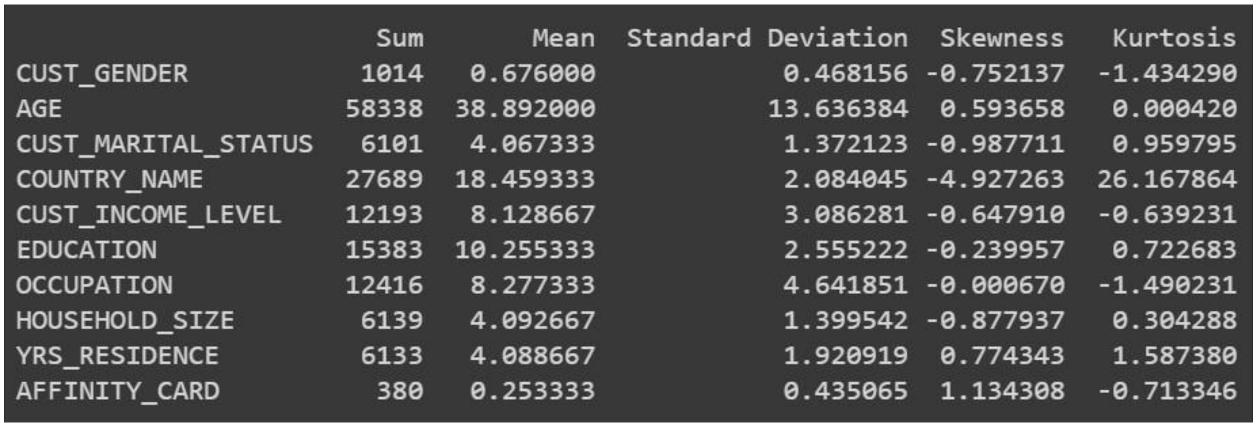


```
# calculate the summary statistics for all the variables
# convert the "CUST_MARITAL_STATUS" and "OCCUPATION" attributes to ordinal
## numbers
cust_marital_status_mapping = {'Divorc.':1, 'Mabsent':2, 'Mar-AF':3,
                               'Married':4, 'NeverM':5, 'Separ.':6, 'Widowed':7}
occupation_mapping = {'Armed-F':1, 'Cleric.':2, 'Crafts':3, 'Exec.':4,
                      'Farming':6, 'Handler':7, 'House-s':8, 'Machine':9,
                      'NotSpec.':10, 'Other':11, 'Prof.':12, 'Protec.':13,
                      'Sales':14, 'TechSup':15, 'Transp.':16}
# apply the mapping to the CUST_MARITAL_STATUS and OCCUPATION columns
mcd1['CUST_MARITAL_STATUS'] = mcd1['CUST_MARITAL_STATUS'].map(
    cust_marital_status_mapping)
mcd1['OCCUPATION'] = mcd1['OCCUPATION'].map(occupation_mapping)

# summary stats
mcd1_summary_stats = pd.DataFrame({
    'Sum': mcd1.sum(),
    'Mean': mcd1.mean(),
    'Standard Deviation': mcd1.std(),
    'Skewness': skew(mcd1),
    'Kurtosis': kurtosis(mcd1)
})
```

code 11: summary statistics.

The python program as shown in code 11 above first creates ‘cust_marital_status_mapping’ and ‘occupation_mapping’ which will be used to convert the CUST_MARITAL_STATUS and OCCUPATION variables to ordinal number without which the summary statistics cannot be calculated for the 2 variables having converted others. The map() function was then used to apply ordinal numbers to the CUST_MARITAL_STATUS and OCCUPATION variables in ‘mcd1’ data frame. The program now calculated the summary statistics for all the variables in the ‘mcd1’ data frame using the sum(), mean(), std(), skew(), and kurtosis() functions from the pandas and scipy.stats python libraries and the results was stored in a new data frame called ‘mcd1_summary_stats’.

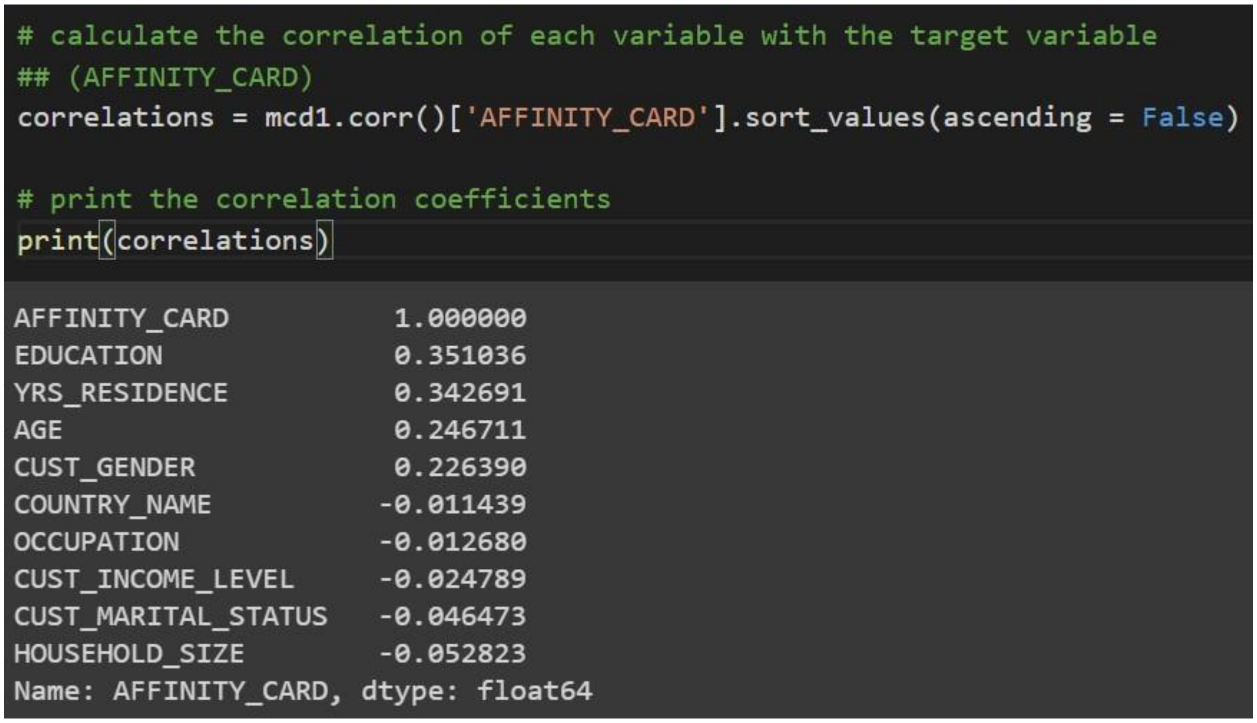
Table 7 below shows the results of the summary statistics.



```
                        Sum       Mean  Standard Deviation  Skewness   Kurtosis
CUST_GENDER            1014   0.676000            0.468156 -0.752137  -1.434290
AGE                   58338  38.892000           13.636384  0.593658   0.000420
CUST_MARITAL_STATUS    6101   4.067333            1.372123 -0.987711   0.959795
COUNTRY_NAME          27689  18.459333            2.084045 -4.927263  26.167864
CUST_INCOME_LEVEL     12193   8.128667            3.086281 -0.647910  -0.639231
EDUCATION             15383  10.255333            2.555222 -0.239957   0.722683
OCCUPATION            12416   8.277333            4.641851 -0.000670  -1.490231
HOUSEHOLD_SIZE         6139   4.092667            1.399542 -0.877937   0.304288
YRS_RESIDENCE          6133   4.088667            1.920919  0.774343   1.587380
AFFINITY_CARD           380   0.253333            0.435065  1.134308  -0.713346
```

table 7: summary statistics results.

The python program that calculates and shows the correlation of each variable with the target variable (AFFINITY_CARD) is shown in code 12 below.



```python
# calculate the correlation of each variable with the target variable
## (AFFINITY_CARD)
correlations = mcd1.corr()['AFFINITY_CARD'].sort_values(ascending = False)

# print the correlation coefficients
print(correlations)
```

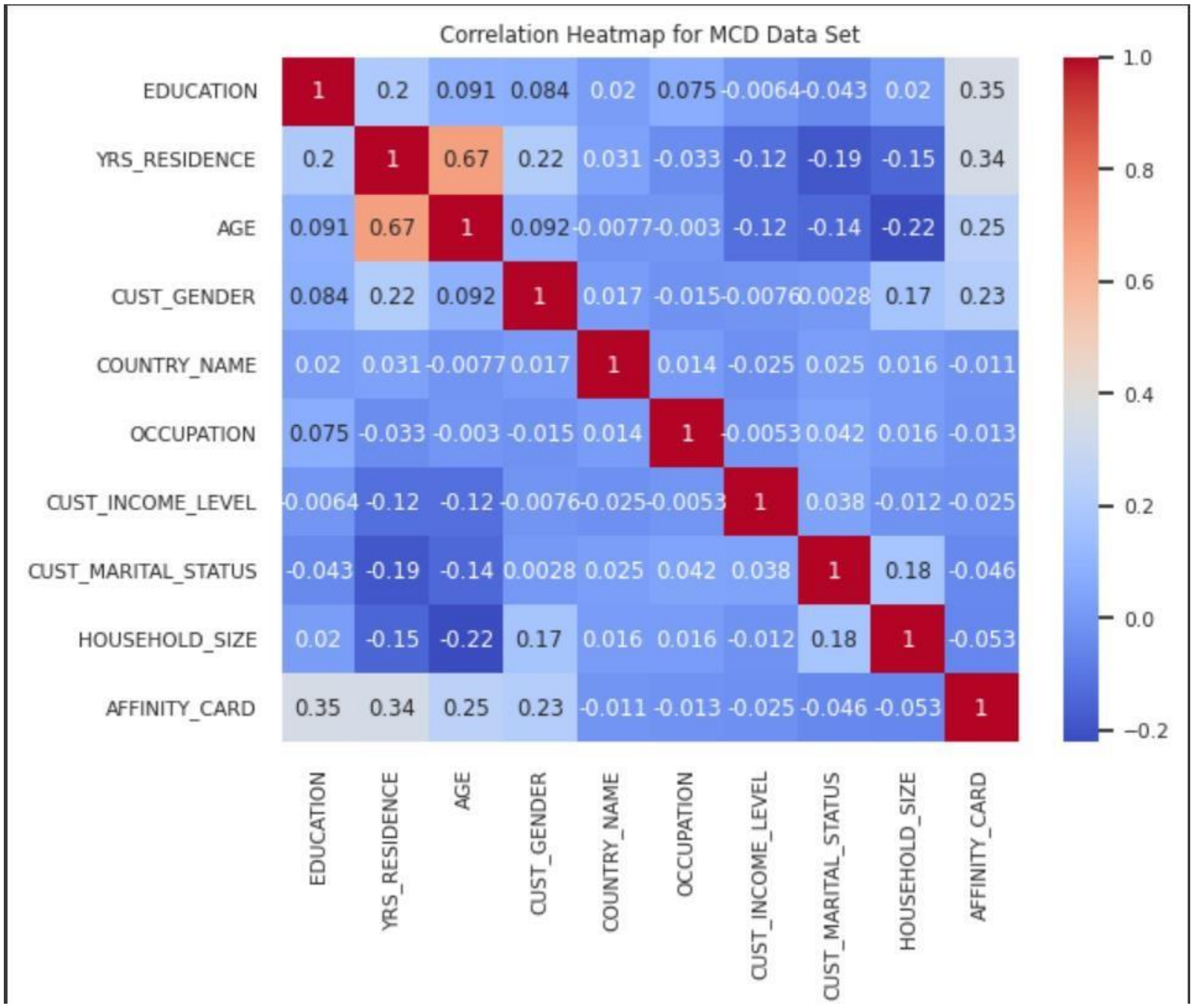
```
AFFINITY_CARD          1.000000
EDUCATION              0.351036
YRS_RESIDENCE          0.342691
AGE                    0.246711
CUST_GENDER            0.226390
COUNTRY_NAME          -0.011439
OCCUPATION            -0.012680
CUST_INCOME_LEVEL     -0.024789
CUST_MARITAL_STATUS   -0.046473
HOUSEHOLD_SIZE        -0.052823
Name: AFFINITY_CARD, dtype: float64
```

code 12: correlations of each variable with the target variable.

The python program as shown in code 12 above used the corr() function to calculate the correlations between the target variable (AFFINITY_CARD) and all other variables in the ‘mcd1’ data frame. The program then selects only the correlations between the target variable (AFFINITY_CARD) and other variables using the square bracket ‘[]’ notation. The sort_values() function was now used to arrange the results in a descending order order by setting the parameter ‘ascending’ equal ‘False’.

The results show that there is a low correlation between the target variable (AFFINITY_CARD) and other independent variables with EDUCATION having the highest correlation value of 0.351036 with the AFFINITY_CARD.

Fig. 2 below shows the correlation heatmap.

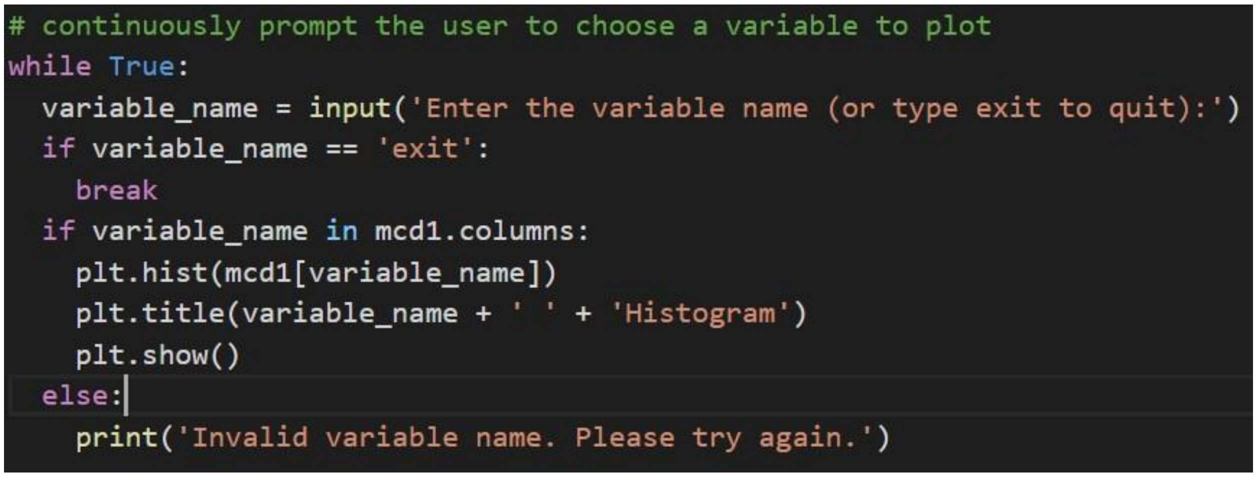


	EDUCATION	YRS_RESIDENCE	AGE	CUST_GENDER	COUNTRY_NAME	OCCUPATION	CUST_INCOME_LEVEL	CUST_MARITAL_STATUS	HOUSEHOLD_SIZE	AFFINITY_CARD
EDUCATION	1	0.2	0.091	0.084	0.02	0.075	-0.0064	-0.043	0.02	0.35
YRS_RESIDENCE	0.2	1	0.67	0.22	0.031	-0.033	-0.12	-0.19	-0.15	0.34
AGE	0.091	0.67	1	0.092	-0.0077	-0.003	-0.12	-0.14	-0.22	0.25
CUST_GENDER	0.084	0.22	0.092	1	0.017	-0.015	-0.0076	0.0028	0.17	0.23
COUNTRY_NAME	0.02	0.031	-0.0077	0.017	1	0.014	-0.025	0.025	0.016	-0.011
OCCUPATION	0.075	-0.033	-0.003	-0.015	0.014	1	-0.0053	0.042	0.016	-0.013
CUST_INCOME_LEVEL	-0.0064	-0.12	-0.12	-0.0076	-0.025	-0.0053	1	0.038	-0.012	-0.025
CUST_MARITAL_STATUS	-0.043	-0.19	-0.14	0.0028	0.025	0.042	0.038	1	0.18	-0.046
HOUSEHOLD_SIZE	0.02	-0.15	-0.22	0.17	0.016	0.016	-0.012	0.18	1	-0.053
AFFINITY_CARD	0.35	0.34	0.25	0.23	-0.011	-0.013	-0.025	-0.046	-0.053	1

fig 2: correlation heatmap.

4. Data exploration

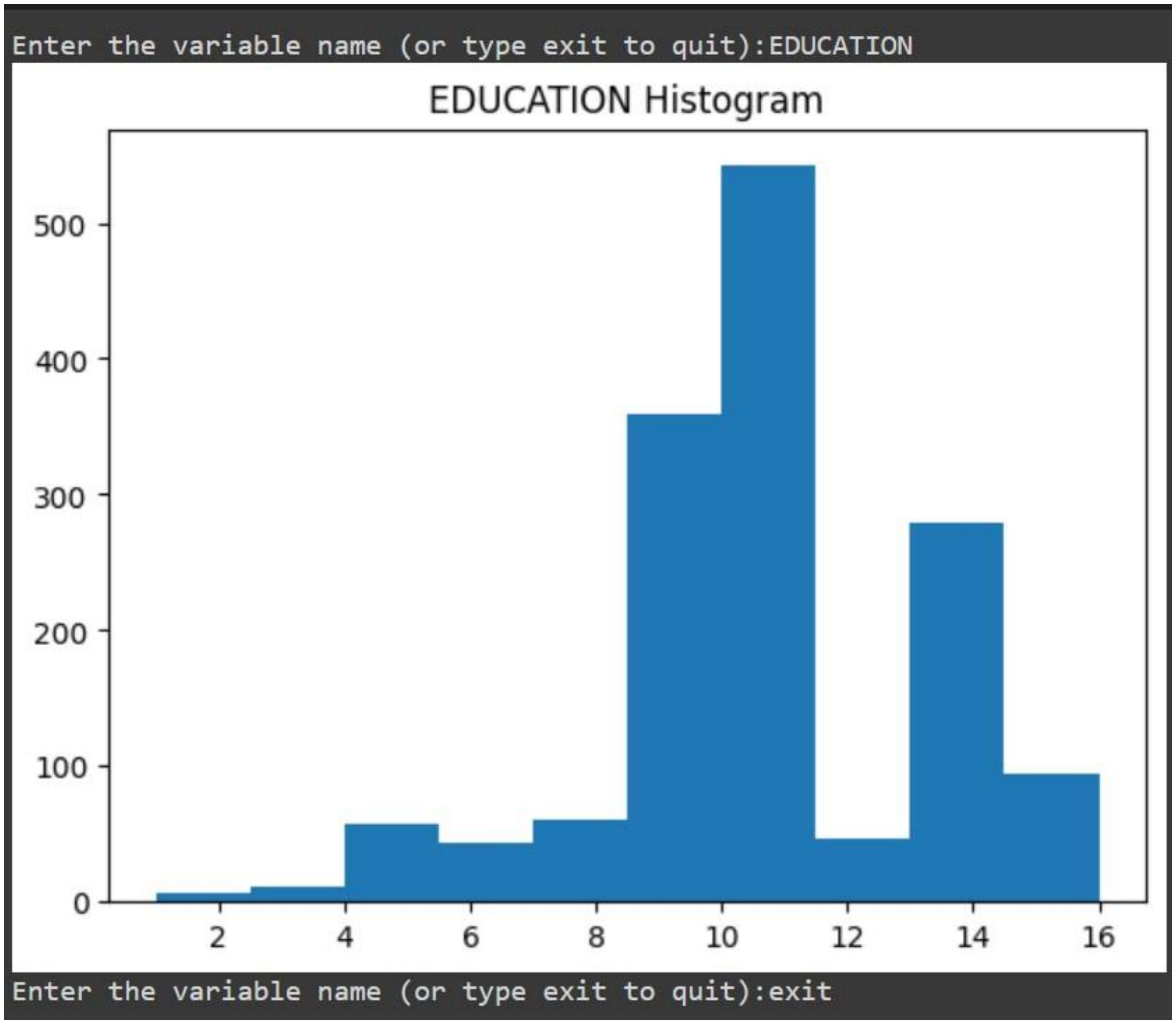
The python program that shows the histogram plot of chosen variables that will continuing running until the user chooses exit is shown in code 13 below.



```
# continuously prompt the user to choose a variable to plot
while True:
  variable_name = input('Enter the variable name (or type exit to quit):')
  if variable_name == 'exit':
    break
  if variable_name in mcd1.columns:
    plt.hist(mcd1[variable_name])
    plt.title(variable_name + ' ' + 'Histogram')
    plt.show()
  else:
    print('Invalid variable name. Please try again.')
```

code 13: continuous running program.

The python program as shown in code 13 above first starts with an infinite ‘while’ loop that continues to run until the user enters the word ‘exit’ to stop the loop. The user is prompted to enter the name of a variable in the ‘mcd1’ data frame which will plot a histogram of that variable generated using hist() function. The title of the histogram chart will include the variable name plus ‘Histogram’. If the variable name entered is not found in the ‘mcd1’ data frame, an error message will be shown.



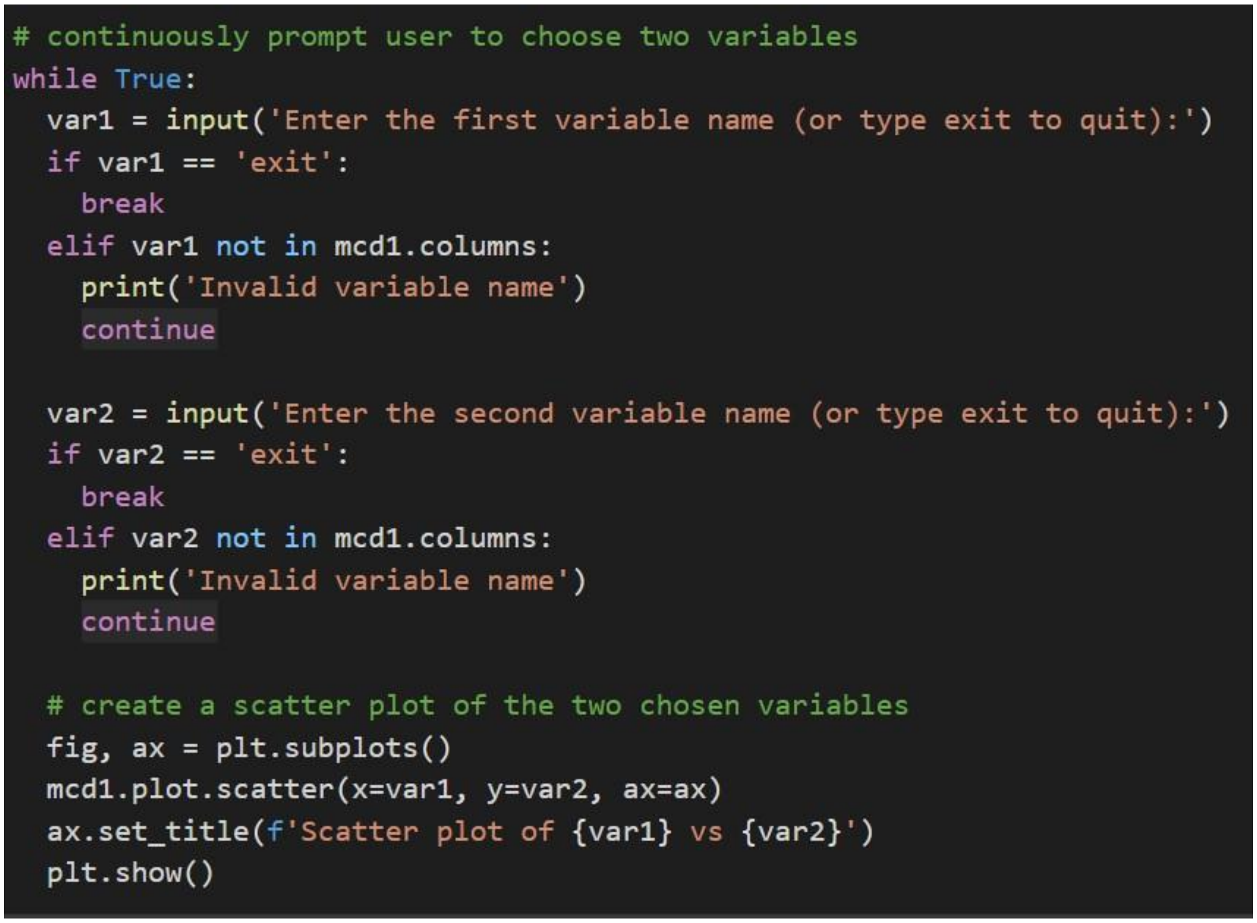
```
Enter the variable name (or type exit to quit):EDUCATION
```

```
Enter the variable name (or type exit to quit):exit
```

fig 3: education histogram.

Fig. 3 above shows that the EDUCATION variable name was entered after which the ‘exit’ word was entered to quit the loop.

The python program that shows scatter plot for any two user chosen variables that continuously runs until the user enters exit is shown in code 14 below.

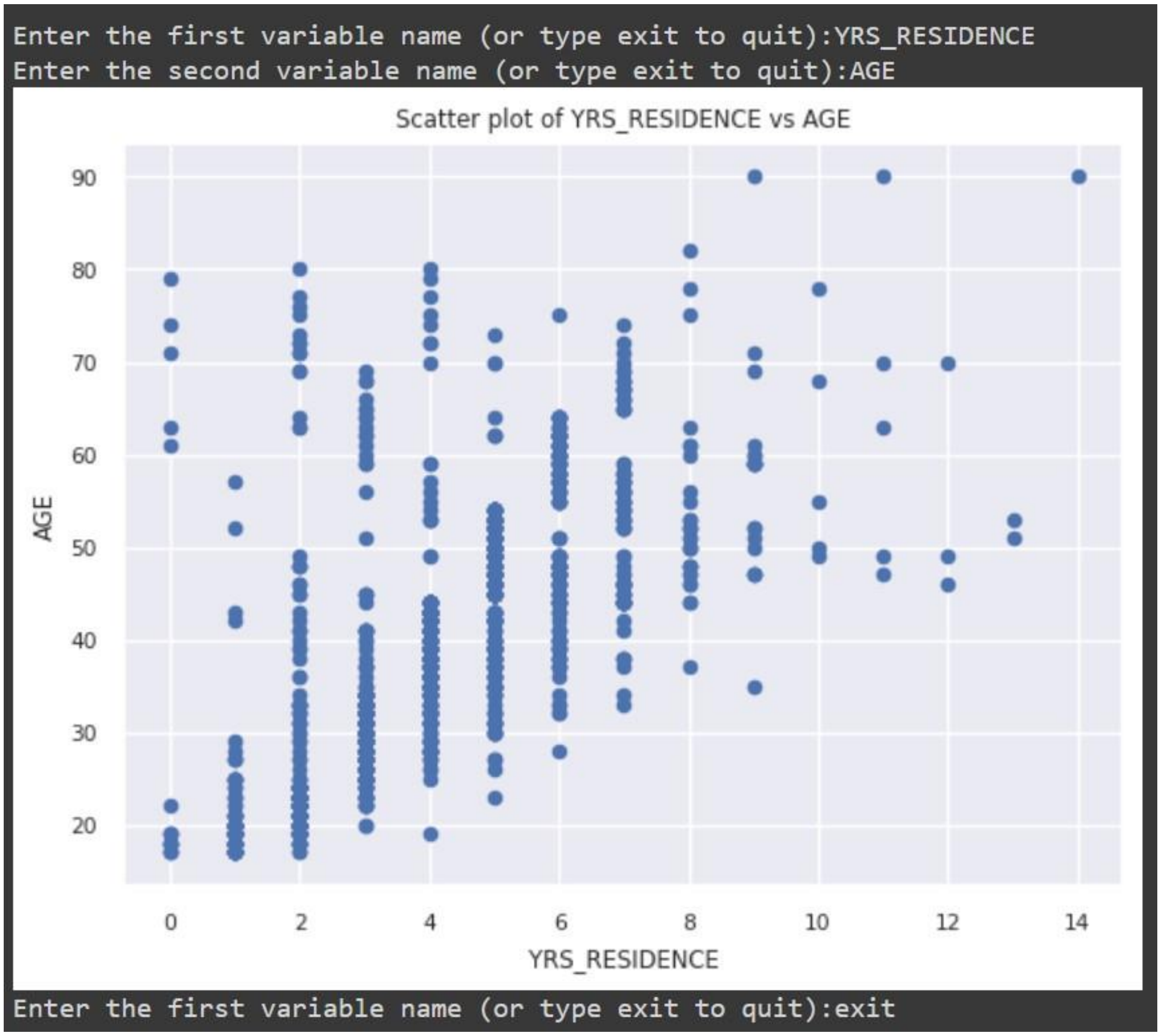


```python
# continuously prompt user to choose two variables
while True:
  var1 = input('Enter the first variable name (or type exit to quit):')
  if var1 == 'exit':
    break
  elif var1 not in mcd1.columns:
    print('Invalid variable name')
    continue

  var2 = input('Enter the second variable name (or type exit to quit):')
  if var2 == 'exit':
    break
  elif var2 not in mcd1.columns:
    print('Invalid variable name')
    continue

  # create a scatter plot of the two chosen variables
  fig, ax = plt.subplots()
  mcd1.plot.scatter(x=var1, y=var2, ax=ax)
  ax.set_title(f'Scatter plot of {var1} vs {var2}')
  plt.show()
```

code 14: continuous running two variables program.

The python program shown in code 14 above prompts the user to enter two variable names in the ‘mcd1’ data frame which creates a scatter plot of the two variables after which the user is prompted to enter word ‘exit’ to quit the loop. The while loop breaks and the program stops if the user enters ‘exit’ after the first variable name and prints ‘Invalid variable name’ if the user enters a variable name that is not found in the ‘mcd1’ data frame. If the user inputs two variable names found in the ‘mcd1’ data frame, the program creates a scatter plot of the two variables using the scatter() function. The scatter plot title shows the names of the two variables entered by the user. The loop repeats, prompting the user to enter another two valid variable names or enter ‘exit’ the quit the loop.

Fig. 4 below shows a scatter plot of YRS_RESIDENCE and AGE followed by ‘exit’ to quit the loop.



```
Enter the first variable name (or type exit to quit):YRS_RESIDENCE
Enter the second variable name (or type exit to quit):AGE
```

```
Enter the first variable name (or type exit to quit):exit
```

fig. 4: continuous scatter plot two variable program.

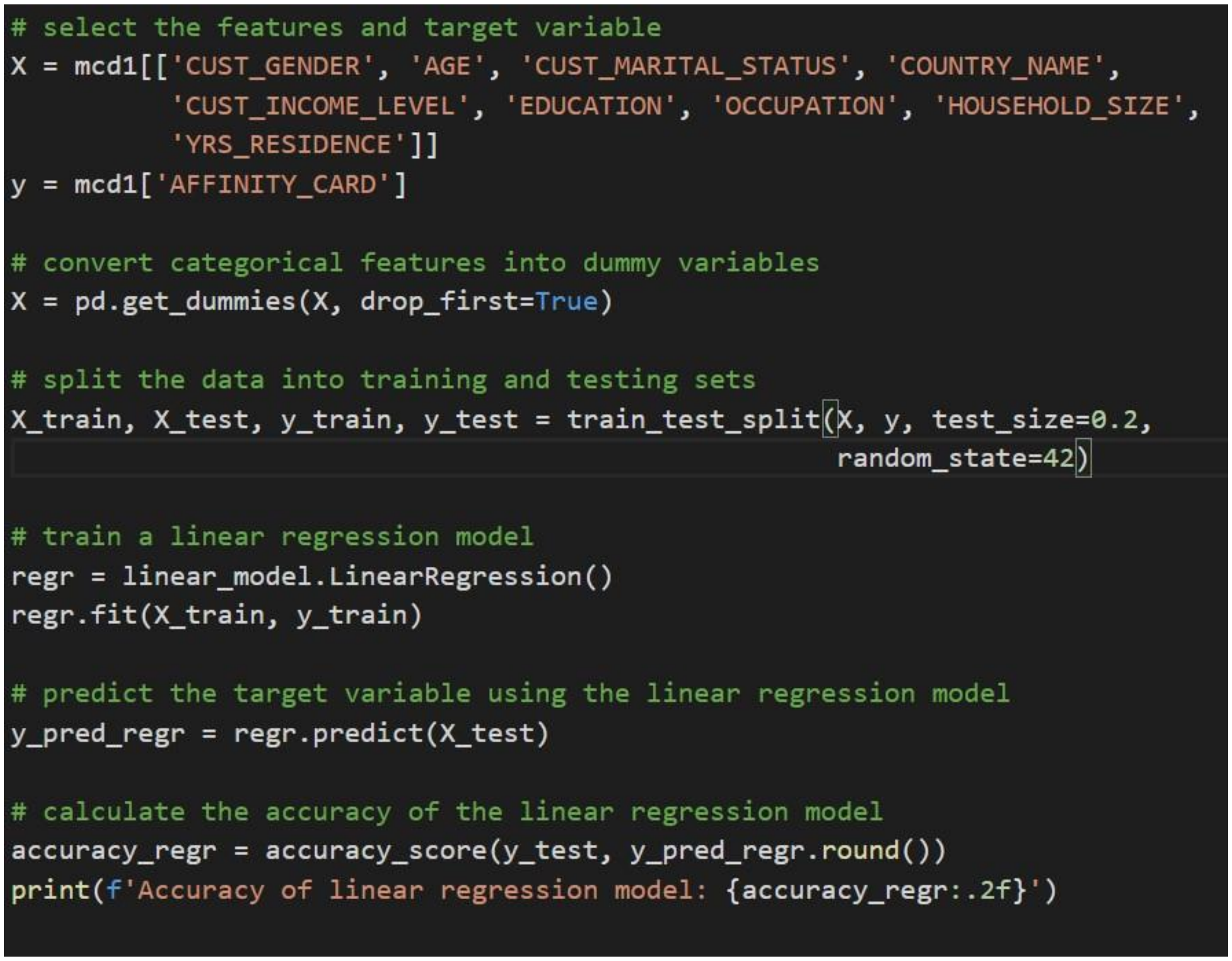
5. Data Mining

A predictive model is a statistical or mathematical tool that is used to predict future outcomes based on historical data. To build a build predictive model, several steps like data understanding, data cleaning, data exploration as already been done here, then feature selection, model selection, and model evaluation. The data is trained and validated to create a model that can accurately predict outcomes on new data. Predictive models can be used in different applications like finance, manufacturing, healthcare, marketing and so on.

After several evaluations with different predictive models, I used the Linear Regression model and Random Forest Classifier model because they gave me the highest accuracy values of 0.80 and 0.83 respectively.

The Linear Regression model is a statistical model used to establish a linear relationship between a dependent variable and one or more independent variables while the Random Forest Classifier is an ensemble learning method that builds multiple decision trees at training time and outputs the classification or regression.

The linear regression python program is shown in code 15 below.

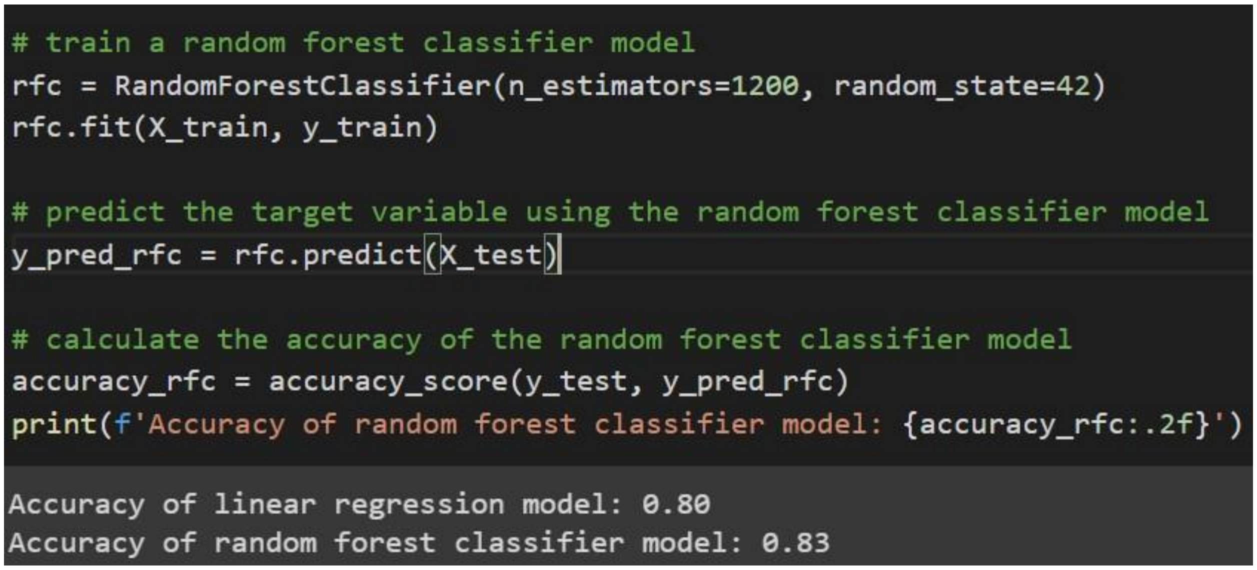


```python
# select the features and target variable
X = mcd1[['CUST_GENDER', 'AGE', 'CUST_MARITAL_STATUS', 'COUNTRY_NAME',
          'CUST_INCOME_LEVEL', 'EDUCATION', 'OCCUPATION', 'HOUSEHOLD_SIZE',
          'YRS_RESIDENCE']]
y = mcd1['AFFINITY_CARD']

# convert categorical features into dummy variables
X = pd.get_dummies(X, drop_first=True)

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)

# train a linear regression model
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)

# predict the target variable using the linear regression model
y_pred_regr = regr.predict(X_test)

# calculate the accuracy of the linear regression model
accuracy_regr = accuracy_score(y_test, y_pred_regr.round())
print(f'Accuracy of linear regression model: {accuracy_regr:.2f}')
```

code 15: linear regression model.

The python program shown in code 15 above first assigned the independent variables to variable 'X' and the dependent variable to variable 'y'. The train_test_split function from the scikit-learn python library was used to split the data into 20% testing set and 80% training set at a random state of 42. The get_dummies() function from pandas python library was then used to convert the categorical variables into dummy variables. The LinearRegression() function from scikit-learn python library was used to create the linear regression model. The predict() function was used to predict the target variable on the testing set and the accuracy_score() function from scikit-learn python library was used to calculate the accuracy which gave a score of 0.80. The model assessment for the linear regression model shows to be a good fit for the 'mcd' data having given an accuracy of 80% to predict the target variable (AFFINITY_CARD).

The python program for the random forest classifier is shown in code 16 below:



```
# train a random forest classifier model
rfc = RandomForestClassifier(n_estimators=1200, random_state=42)
rfc.fit(X_train, y_train)

# predict the target variable using the random forest classifier model
y_pred_rfc = rfc.predict(X_test)

# calculate the accuracy of the random forest classifier model
accuracy_rfc = accuracy_score(y_test, y_pred_rfc)
print(f'Accuracy of random forest classifier model: {accuracy_rfc:.2f}')
```

```
Accuracy of linear regression model: 0.80
Accuracy of random forest classifier model: 0.83
```

code 16: random forest classifier.

The python program shown in code 16 above trains a random forest classifier model with 1200 decision trees at a random state of 42. It used the trained model to predict the target variable on the testing data set. It then calculated the accuracy of the model using the accuracy_score() function, which compares the predicted values with the actual values and returns the percentage of the correct predictions. The resulting accuracy score of 0.83 shows that 83% of the testing data predicted the target variable correctly.

6. Discussion and reflection of the work

Doing this coursework has helped hone my analytical, curious-minded, attention to detail, multitasking skills among several others. I took two weeks off work and had several sleepless nights. I performed lots of research, had several productive discussions with my course mates, my workshop tutor, Ayisha Dubi and my ever willing and ready to help course leader, Dr. Qicheng.

This coursework gave me a deeper understanding of the five-step process of machine learning which involves data collection, data exploration and preparation, model training, model evaluation, and model improvement. After which the best performing model will be deployed for its intended task.