TECHNICAL REPORT FOR PROJECT

Contents

1 Data Understanding

The data for this coursework is a marketing campaign dataset of a retailer company and it contains 19 variables and 1500 customer records that hold socio-demographic and product ownership information. The dataset uses the Python programming language and libraries like pandas.

1.1 Loading the libraries and CSV file

The python libraries like pandas, NumPy, Matplotlib, seaborn, SciPy, and Scikit-Learn are loaded into the data frame.

NumPy: A Python library used for working with large, multi-dimensional arrays and matrices.

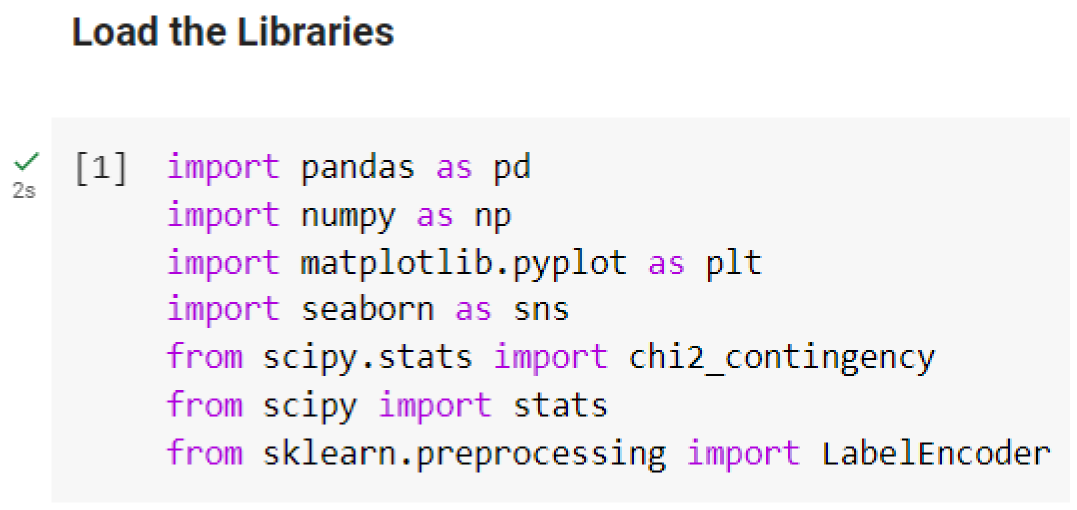
Matplotlib: A Python library used for creating static, interactive, and animated visualizations.

Seaborn: A Python library based on Matplotlib used for data visualization.

SciPy: An open-source Python library used for manipulating and visualizing.

Scikit-Learn: An open-source Python library used for machine learning models that involve classification, regressing, and clustering.

Note: Features, variables, and columns shall be used interchangeably in this project.,

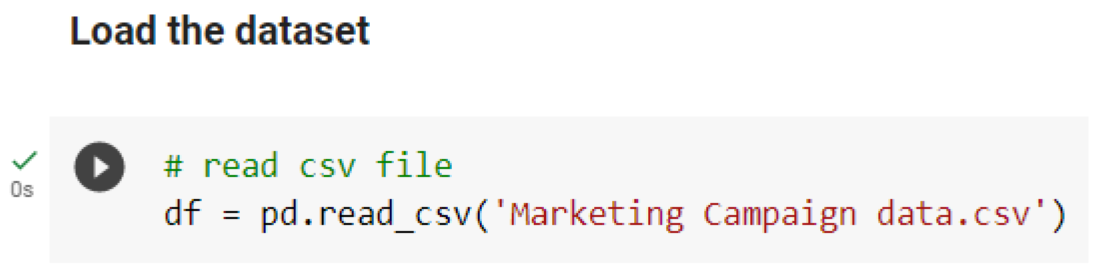


Load the Libraries

```
[1] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from scipy.stats import chi2_contingency
    from scipy import stats
    from sklearn.preprocessing import LabelEncoder
```

Figure 1: Loading all Python libraries

In Figure 1, all the libraries are loaded into Google Colab. After this, data manipulation, mining, and analysis can then be started.



Load the dataset

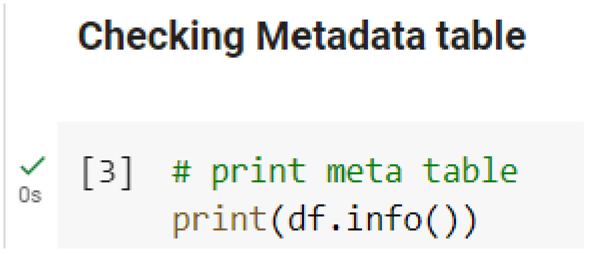
```
# read csv file
df = pd.read_csv('Marketing Campaign data.csv')
```

Figure 2: Reading the Marketing Campaign data csv

In Figure 2, the dataset is loaded into the data frame.

1.2 Metadata Tables

A metadata table is a table that describes the features of a dataset. It gives information and details about a particular piece of the dataset. It can include information such as file name, file type, count, etc.

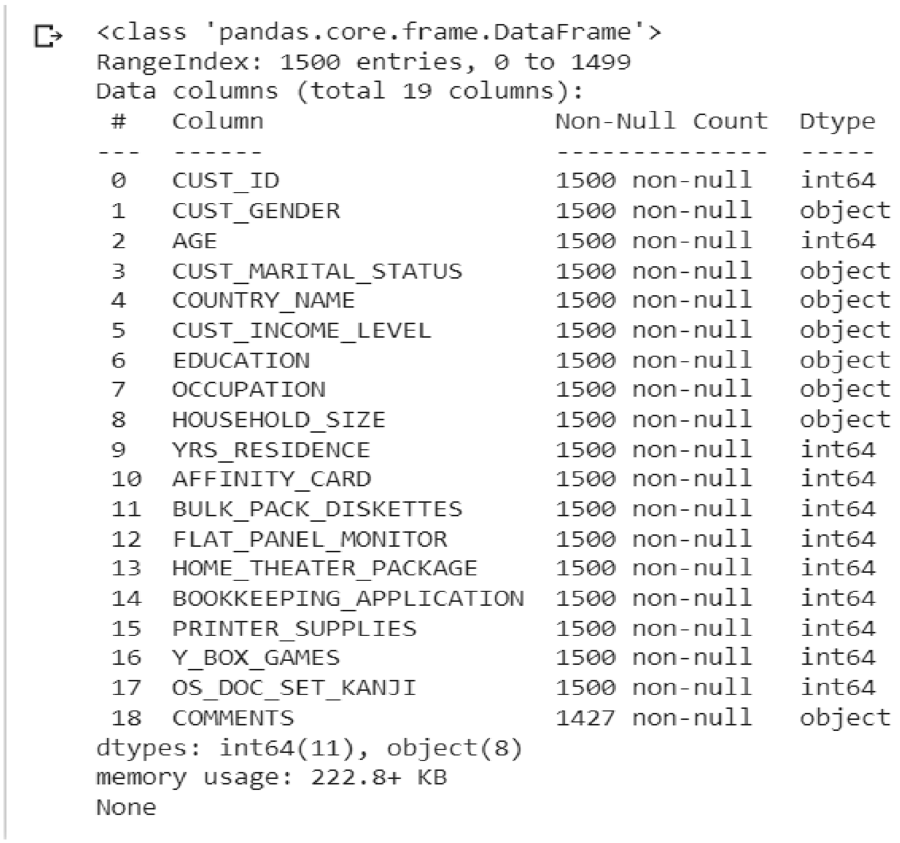


Checking Metadata table

```
[3]  # print meta table
     print(df.info())
```

Figure 3: code using Python libraries to check metadata table

The metadata table is generated by calling the df.info() function in Figure 3 and the output is printed out in Figure 4.



```
<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   CUST_GENDER               1500 non-null   object
 2   AGE                       1500 non-null   int64 
 3   CUST_MARITAL_STATUS       1500 non-null   object
 4   COUNTRY_NAME              1500 non-null   object
 5   CUST_INCOME_LEVEL         1500 non-null   object
 6   EDUCATION                 1500 non-null   object
 7   OCCUPATION                1500 non-null   object
 8   HOUSEHOLD_SIZE            1500 non-null   object
 9   YRS_RESIDENCE             1500 non-null   int64 
 10  AFFINITY_CARD             1500 non-null   int64 
 11  BULK_PACK_DISKETTES       1500 non-null   int64 
 12  FLAT_PANEL_MONITOR        1500 non-null   int64 
 13  HOME_THEATER_PACKAGE      1500 non-null   int64 
 14  BOOKKEEPING_APPLICATION   1500 non-null   int64 
 15  PRINTER_SUPPLIES          1500 non-null   int64 
 16  Y_BOX_GAMES               1500 non-null   int64 
 17  OS_DOC_SET_KANJI          1500 non-null   int64 
 18  COMMENTS                  1427 non-null   object
dtypes: int64(11), object(8)
memory usage: 222.8+ KB
None
```

Figure 4: The metadata table

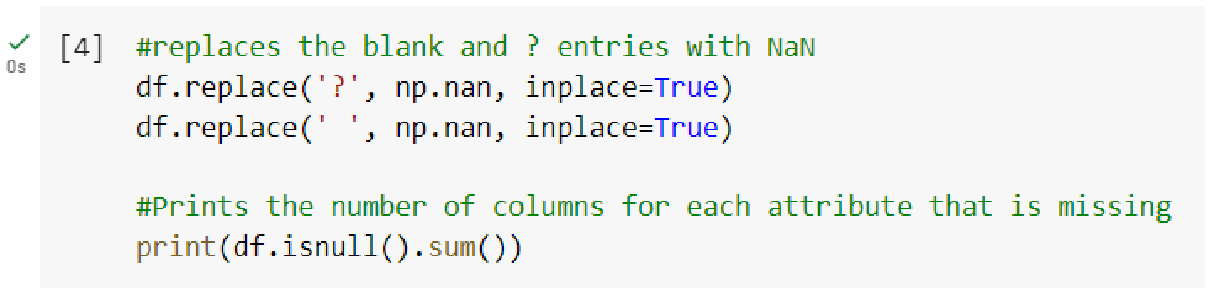
There are 1500 rows of the dataset. There are 19 features of which some features will drop later in this project and the remaining used for mining and analysis. The features are either int64 or object data types. The COMMENTS column is missing some values.

1.3 Missing or error data of each attribute

1.3.1 Missing Data

It is important to set straight what exactly is missing data. In this scenario, anything that is “?” or has a blank entry is said to be missing. These missing inputs will then be unified and the blank spaces and “?” are replaced with “NaN” (Not a Number).

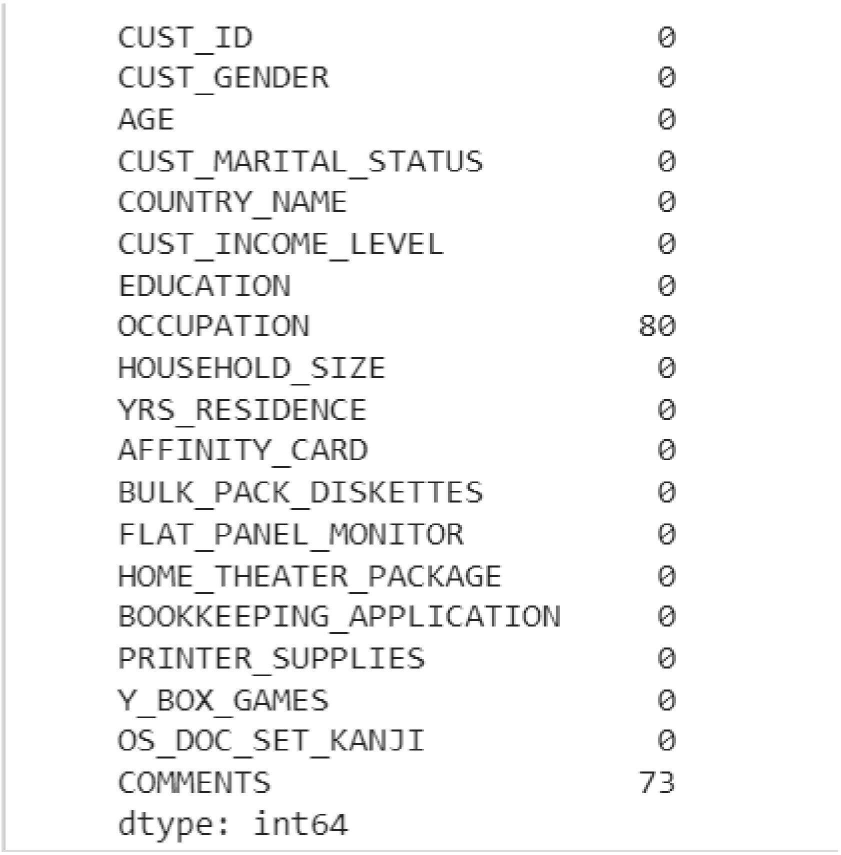
Check for missing data



```
[4] #replaces the blank and ? entries with NaN
    df.replace('?', np.nan, inplace=True)
    df.replace(' ', np.nan, inplace=True)

    #Prints the number of columns for each attribute that is missing
    print(df.isnull().sum())
```

Figure 5: Code to check missing data in the dataset

The code above reads the CSV file, replaces the "?" and blank spaces with NaN, and then looks for the total of all entries that is null for each variable.



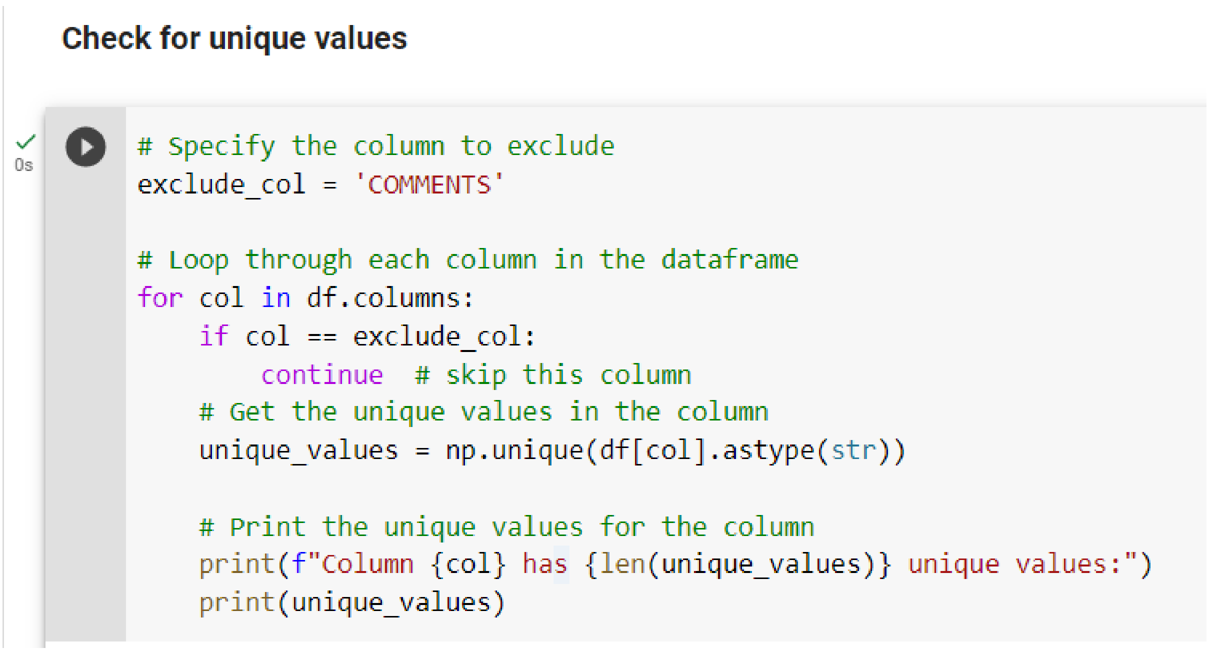
```
CUST_ID                    0
CUST_GENDER                0
AGE                        0
CUST_MARITAL_STATUS        0
COUNTRY_NAME               0
CUST_INCOME_LEVEL          0
EDUCATION                  0
OCCUPATION                80
HOUSEHOLD_SIZE             0
YRS_RESIDENCE              0
AFFINITY_CARD              0
BULK_PACK_DISKETTES        0
FLAT_PANEL_MONITOR         0
HOME_THEATER_PACKAGE       0
BOOKKEEPING_APPLICATION    0
PRINTER_SUPPLIES           0
Y_BOX_GAMES                0
OS_DOC_SET_KANJI           0
COMMENTS                  73
dtype: int64
```

Figure 6: Output of missing data code

The code above shows the number of entries that are null for each attribute. It can be seen that the occupation and comments variables have 80 and 73 null variables respectively. One thing to know is that fields that have "?" values are not denoted by Python as nulls until they are manipulated and replaced with NaN.

1.3.2 Error Data

Checking for error data can be a little bit tricky as error checking involves using the data types to check for errors. That's why the code below is used to check for unique values and then if any irregular data is found, provision can be made when writing the error-checking code.


Check for unique values

```python
# Specify the column to exclude
exclude_col = 'COMMENTS'

# Loop through each column in the dataframe
for col in df.columns:
    if col == exclude_col:
        continue  # skip this column
    # Get the unique values in the column
    unique_values = np.unique(df[col].astype(str))

    # Print the unique values for the column
    print(f"Column {col} has {len(unique_values)} unique values:")
    print(unique_values)
```

Figure 7: Code to check for unique columns

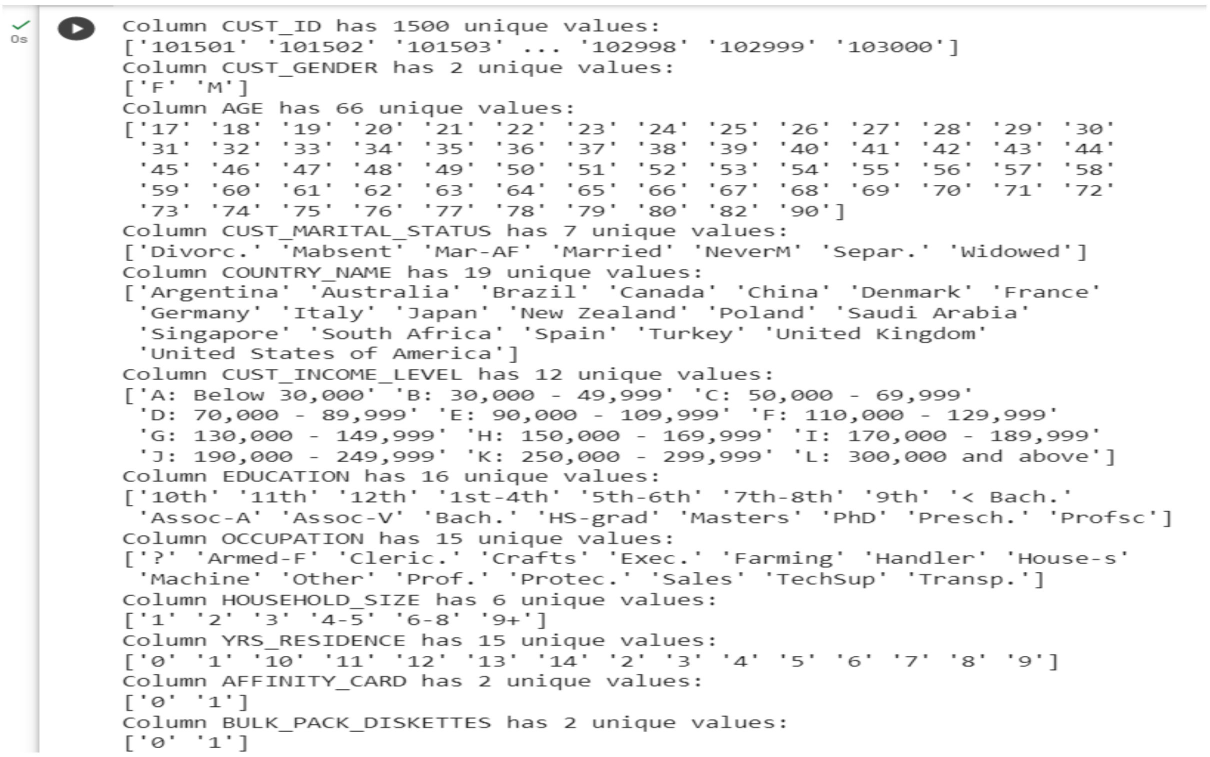

```
Column CUST_ID has 1500 unique values:
['101501' '101502' '101503' ... '102998' '102999' '103000']
Column CUST_GENDER has 2 unique values:
['F' 'M']
Column AGE has 66 unique values:
['17' '18' '19' '20' '21' '22' '23' '24' '25' '26' '27' '28' '29' '30'
 '31' '32' '33' '34' '35' '36' '37' '38' '39' '40' '41' '42' '43' '44'
 '45' '46' '47' '48' '49' '50' '51' '52' '53' '54' '55' '56' '57' '58'
 '59' '60' '61' '62' '63' '64' '65' '66' '67' '68' '69' '70' '71' '72'
 '73' '74' '75' '76' '77' '78' '79' '80' '82' '90']
Column CUST_MARITAL_STATUS has 7 unique values:
['Divorc.' 'Mabsent' 'Mar-AF' 'Married' 'NeverM' 'Separ.' 'Widowed']
Column COUNTRY_NAME has 19 unique values:
['Argentina' 'Australia' 'Brazil' 'Canada' 'China' 'Denmark' 'France'
 'Germany' 'Italy' 'Japan' 'New Zealand' 'Poland' 'Saudi Arabia'
 'Singapore' 'South Africa' 'Spain' 'Turkey' 'United Kingdom'
 'United States of America']
Column CUST_INCOME_LEVEL has 12 unique values:
['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']
Column EDUCATION has 16 unique values:
['10th' '11th' '12th' '1st-4th' '5th-6th' '7th-8th' '9th' '< Bach.'
 'Assoc-A' 'Assoc-V' 'Bach.' 'HS-grad' 'Masters' 'PhD' 'Presch.' 'Profsc']
Column OCCUPATION has 15 unique values:
['?' 'Armed-F' 'Cleric.' 'Crafts' 'Exec.' 'Farming' 'Handler' 'House-s'
 'Machine' 'Other' 'Prof.' 'Protec.' 'Sales' 'TechSup' 'Transp.']
Column HOUSEHOLD_SIZE has 6 unique values:
['1' '2' '3' '4-5' '6-8' '9+']
Column YRS_RESIDENCE has 15 unique values:
['0' '1' '10' '11' '12' '13' '14' '2' '3' '4' '5' '6' '7' '8' '9']
Column AFFINITY_CARD has 2 unique values:
['0' '1']
Column BULK_PACK_DISKETTES has 2 unique values:
['0' '1']
```

Figure 8: Result of the code used to check unique data

From the above, there is no incident or unique value that has an error or irregular data. But there are some suspect data, especially in the CUST_MARITAL_STATUS feature and they will be discussed in detail below.

- CUST_ID

  The Customer ID has no missing or error data, and all 1500 rows are complete.
- CUST_GENDER

  The Customer Gender has no missing or error data, and all entries are correct and complete.
- AGE

  The Age has no missing or error data, and all entries are correct and complete.

- CUST_MARITAL_STATUS
  The Customer's Marital Status has no missing data. There seems to be suspected error data, but they were cleared up after consulting the necessary links. Mar-AF and Mabsent were the data entries that were considered unique error data, but they were cleared by Unmarried Equality; an organization that stands up for fairness and equal treatment of all people regardless of marital status [1]. Mabsent which is considered as Married but absent and Mar-AF, which is considered as Married, Armed Forces were grouped as “other married, spouse absent”.
- COUNTRY_NAME
  The Country name has no missing or error data, and all entries are correct and complete.
- CUST_INCOME_LEVEL
  The customer income level has no missing or error data, and all entries are correct and complete.
- EDUCATION
  The education level of the customer has no missing data, and all entries are complete.
- OCCUPATION
  The occupation of the customer has about 80 missing data entries which are represented with ‘?” but with no error data.
- HOUSEHOLD_SIZE
  The household size has no missing or error data.
- YRS_RESIDENCE
  The years spent in the residence variable have no missing or error data and all entries are correct and complete.
- AFFINITY_CARD
  The affinity card variable has no missing or error data entry, and all entries are correct and complete.
- BULK_PACK_DISKETTES
  The Bulk pack diskette product has no missing or error data entry, and all entries are correct and complete.
- FLAT_PANEL_MONITOR
  The flat panel monitor product has no missing or error data entry and all entries are correct and complete.
- HOME_THEATER_PACKAGE
  The home theatre package product has no missing or error data entry, and all entries are correct and complete.
- BOOKKEEPING_APPLICATION
  The bookkeeping application variable has no missing or error data entry, and all entries are correct and complete.
- PRINTER_SUPPLIES
  The printer supplies have no missing or error data entries, and all entries are correct and complete.
- Y_BOX_GAMES
  The Y_Box games product has no missing or error data entry, and all entries are correct and complete.
- OS_DOC_SET_KANJI
  This variable has no error or missing data entry, and all entries are correct and complete.
- COMMENTS
  The comments variable has 73 missing data entries which are left blank, and it is difficult to know what is considered error data as it requires dedicated mining tools.

2 Data Preparation

Data preparation involves reducing variables to only those needed for modelling. Various tests are performed on the categorical and numerical data to find the usefulness of each feature to the target variable to prevent bias.

2.1 Reducing Variables.

Reducing variables involves removing the variables that originally do not correlate with the target variable and may inject bias into the results. It also helps to reduce the computational cost of modelling and improve the model.

As stated, there are 19 variables in the CSV file. The **Customer ID** variable will be removed because it's just an identifier and it has no effect on the target variable. The **Comments** variable will also be removed as stated in the instructions.

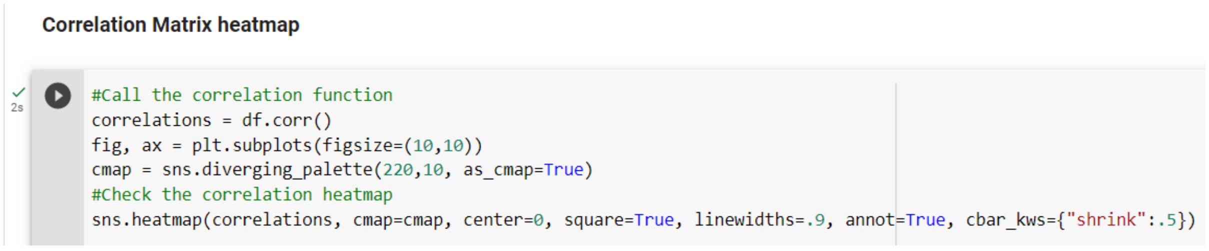
Since there are two types of variables in the data given which are the **categorical** and **numerical** data, a **chi-squared test** and **correlation matrix** shall be performed to reduce variables. Chi-squared tests are useful for categorical data and correlation matrices for numerical data to measure the degree of a linear relationship between pairs.

The categorical variables are

- CUST_GENDER
- CUST_MARITAL_STATUS
- COUNTRY_NAME
- CUST_INCOME_LEVEL
- EDUCATION
- OCCUPATION

The numerical variables are

- AGE
- 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


Correlation Matrix heatmap

```
#Call the correlation function
correlations = df.corr()
fig, ax = plt.subplots(figsize=(10,10))
cmap = sns.diverging_palette(220,10, as_cmap=True)
#Check the correlation heatmap
sns.heatmap(correlations, cmap=cmap, center=0, square=True, linewidths=.9, annot=True, cbar_kws={"shrink":.5})
```

Figure 9: Code to check correlation using correlation heatmap

The code above uses pandas to get the data in the data frame. The df. corr() function calculates the correlation between the variables and stores the correlation matrix in 'correlations'. The cmap creates a color map after which a heatmap is created using the Seaborn library. The result is displayed below

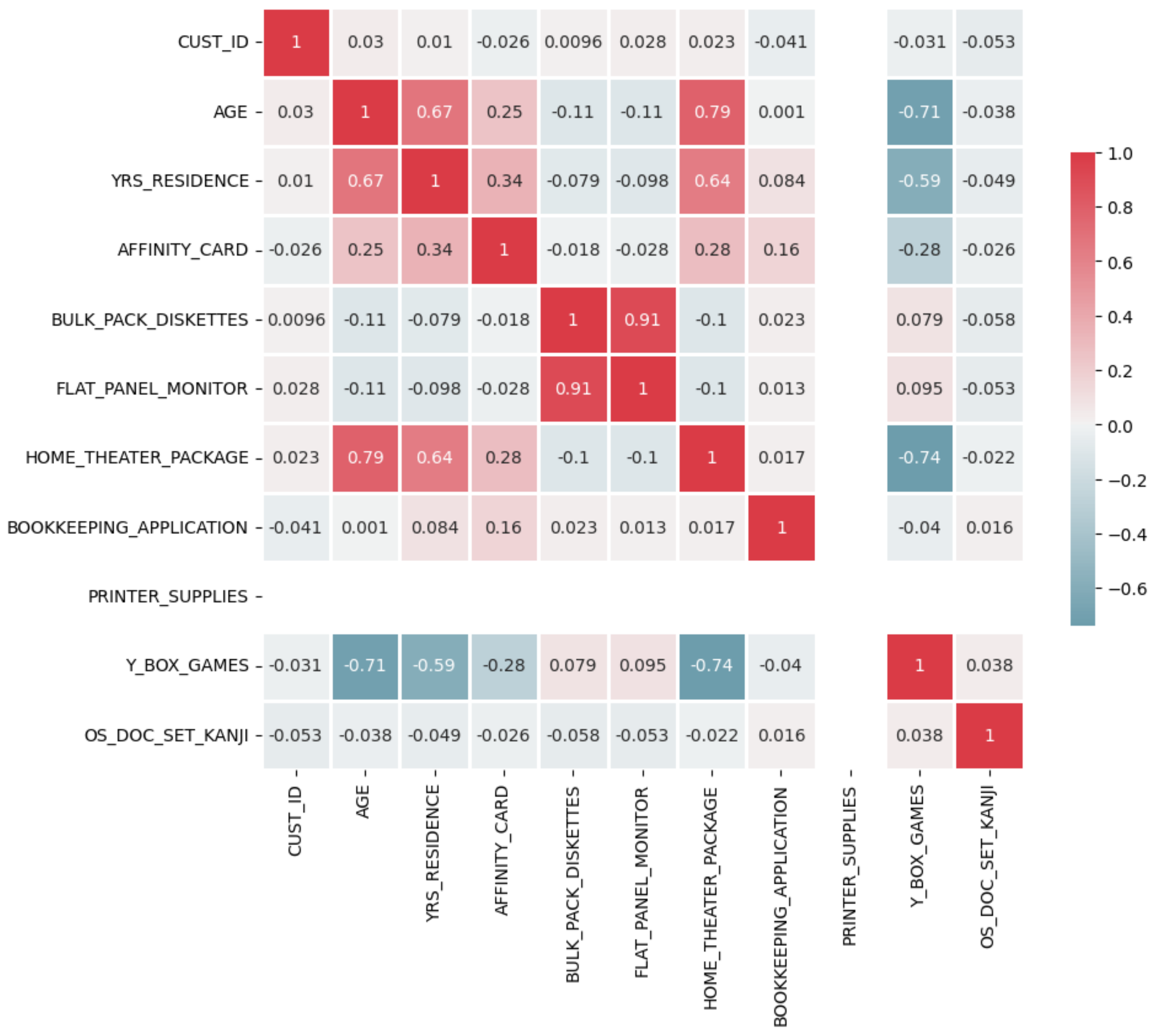
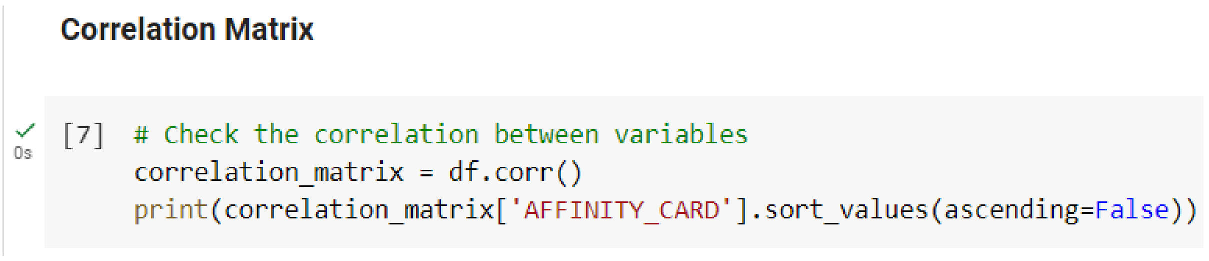


Figure 10: Results showing the correlation matrix between each variable

From the above, we shall only consider the correlations between AFFINITY_CARD and other numerical variables. One can deduct from the above that the PRINTER_SUPPLIES variable will be removed as it has no correlation to the target variable (AFFINITY_CARD) and is unlikely to have any impact.

The AGE, YRS_RESIDENCE, AFFINITY_CARD, FLAT_PANEL_MONITOR, HOME_THEATER_PACKAGE, BOOKKEEPING_APPLICATION, and Y_BOX_GAMES will be considered because they have a certain level of correlation with AFFINITY_CARD.

Also, Correlation can be calculated as shown below


Correlation Matrix

```
[7] # Check the correlation between variables
    correlation_matrix = df.corr()
    print(correlation_matrix['AFFINITY_CARD'].sort_values(ascending=False))
```

Figure 11: Code to show correlation with AFFINITY_CARD

It uses pandas to print the correlation in descending order.

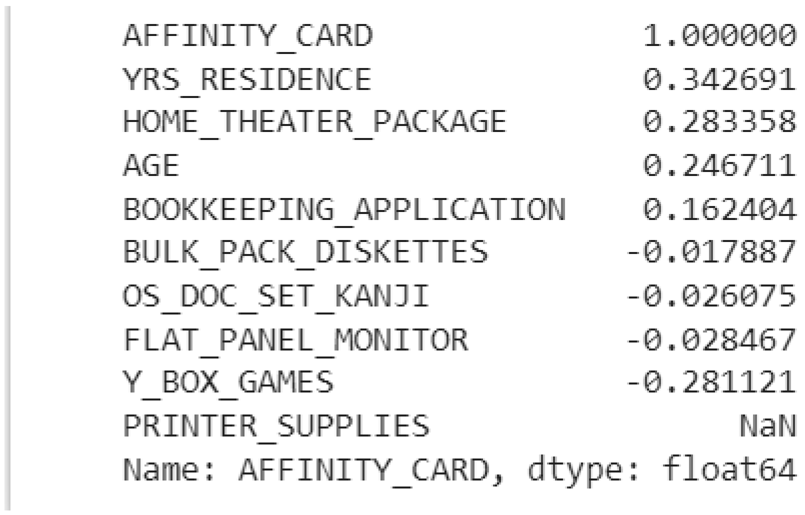
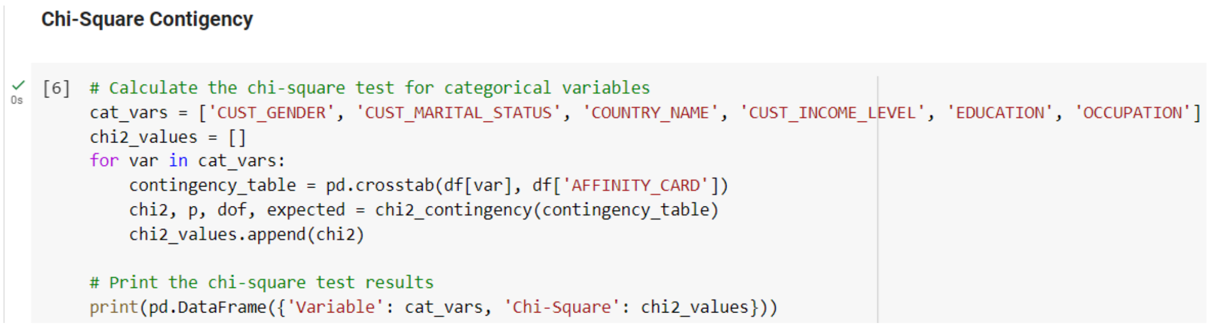

```
AFFINITY_CARD             1.000000
YRS_RESIDENCE             0.342691
HOME_THEATER_PACKAGE      0.283358
AGE                       0.246711
BOOKKEEPING_APPLICATION   0.162404
BULK_PACK_DISKETTES      -0.017887
OS_DOC_SET_KANJI         -0.026075
FLAT_PANEL_MONITOR       -0.028467
Y_BOX_GAMES     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

Figure 12: Results showing the correlation matrix between each variable

The issue of correlation with numerical data has been solved with Correlation Matrix. The chi-squared test is now applied to the categorical variables.


Chi-Square Contigency

```
[6]  # Calculate the chi-square test for categorical variables
     cat_vars = ['CUST_GENDER', 'CUST_MARITAL_STATUS', 'COUNTRY_NAME', 'CUST_INCOME_LEVEL', 'EDUCATION', 'OCCUPATION']
     chi2_values = []
     for var in cat_vars:
         contingency_table = pd.crosstab(df[var], df['AFFINITY_CARD'])
         chi2, p, dof, expected = chi2_contingency(contingency_table)
         chi2_values.append(chi2)

     # Print the chi-square test results
     print(pd.DataFrame({'Variable': cat_vars, 'Chi-Square': chi2_values}))
```

Figure 13: Code to show the chi-squared test for categorical variables

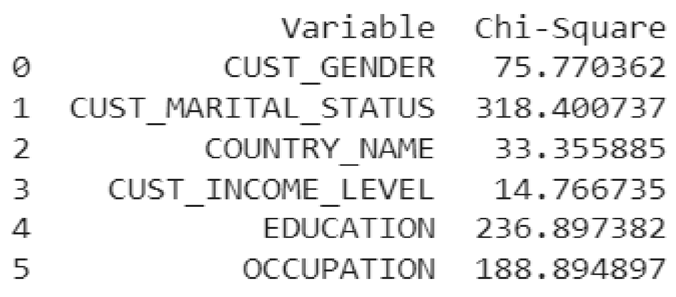

```
              Variable  Chi-Square
0          CUST_GENDER   75.770362
1  CUST_MARITAL_STATUS  318.400737
2         COUNTRY_NAME   33.355885
3    CUST_INCOME_LEVEL   14.766735
4            EDUCATION  236.897382
5           OCCUPATION  188.894897
```

Figure 14: Chi-Square for the categorical variables

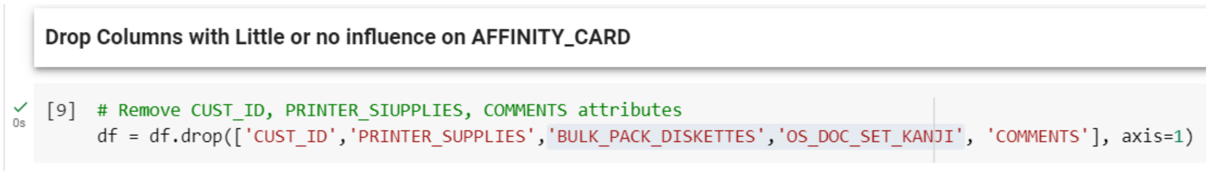
One can deduct from the above that all the variables above have a certain high level of influence on the target variable (AFFINITY_CARD) and BULK_PACK_DISKETTES and OS_DOC_SET_KANJI will be dropped and In conclusion, only three variables will be dropped, and they are:

- CUST_ID: This variable is an identifier and as such as no influence on the target variable.
- PRINTER_SUPPLIES: This variable has no influence on or correlation with the target variable.
- COMMENTS: It requires dedicated text-mining tools.
- BULK_PACK_DISKETTES: It has a very low correlation with the target variable
- OS_DOC_SET_KANJI: It has a very low correlation with the target variable.

Every other variable will be left as they are, and they are:

- CUST_GENDER

- AGE
- CUST_MARITAL_STATUS
- COUNTRY_NAME
- CUST_INCOME_LEVEL
- EDUCATION
- OCCUPATION
- HOUSEHOLD_SIZE
- YRS_RESIDENCE
- AFFINITY_CARD
- FLAT_PANEL_MONITOR
- HOME_THEATER_PACKAGE
- BOOKKEEPING_APPLICATION
- Y_BOX_GAMES



Drop Columns with Little or no influence on AFFINITY_CARD

```
[9] # Remove CUST_ID, PRINTER_SIUPPLIES, COMMENTS attributes
    df = df.drop(['CUST_ID','PRINTER_SUPPLIES','BULK_PACK_DISKETTES','OS_DOC_SET_KANJI', 'COMMENTS'], axis=1)
```

Figure 15: Code to drop CUST_ID, PRINTER_SUPPLIES, BULK_PACK_DISKETTES, OS_DOC_SET_KANJI and COMMENTS

These columns are dropped, and the remaining data are saved to the data frame

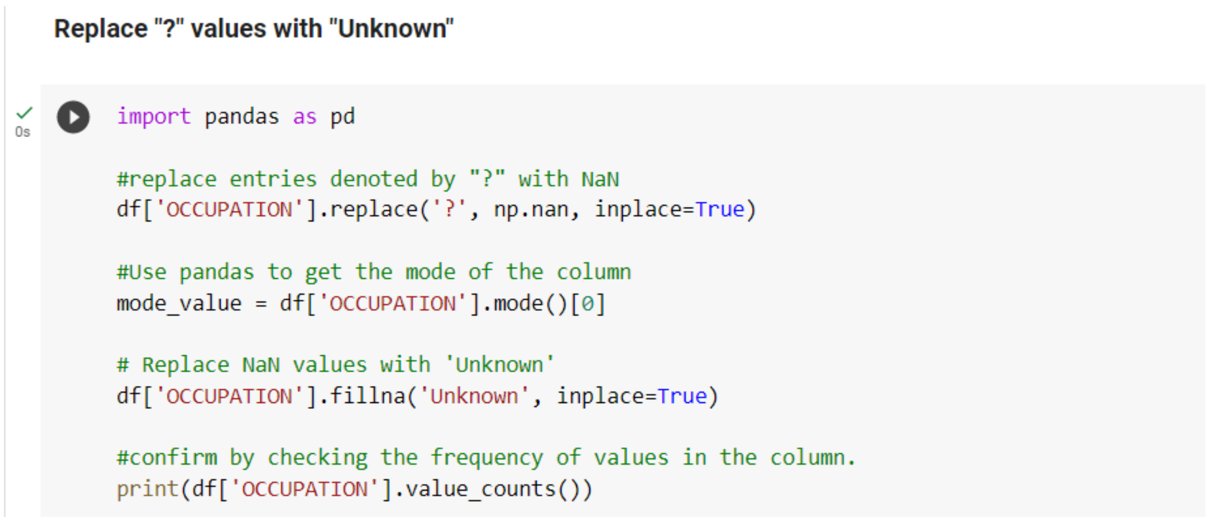
2.2 Cleaning Data

2.2.1 Missing Data

Cleaning data involves removing or altering records with missing values or error data. In this instance, missing data will not be removed because there are a lot of missing data in the OCCUPATION variable (80 records out of 1500) and they represent about 5.3% of the feature values. Removing these records can lead to biased or inaccurate results. It can also lead to loss of information and the ability to detect important patterns in the data. In the case of missing data, we shall be replacing the "?" value with "Unknown " because we don't want to delete the data to avoid loss of information. We shall be creating another category for it. Using the mode would have been another option but not suitable because the number of occurrences of each value is close one another, and it could be any of them.

The figure below uses pandas to replace the "?" entries with NaN (Not a Number) for easy manipulation. Then pandas get the mode of the variable (OCCUPATION) and then replace the NaN entries with "UNKNOWN".

We can make a confirmation by printing out the frequency of values in the variable.


Replace "?" values with "Unknown"

```
import pandas as pd

#replace entries denoted by "?" with NaN
df['OCCUPATION'].replace('?', np.nan, inplace=True)

#Use pandas to get the mode of the column
mode_value = df['OCCUPATION'].mode()[0]

# Replace NaN values with 'Unknown'
df['OCCUPATION'].fillna('Unknown', inplace=True)

#confirm by checking the frequency of values in the column.
print(df['OCCUPATION'].value_counts())
```

Figure 16: Code to replace the missing data with “Unknown”

The figure shows the frequency of the values of the variable. They all add up to 1500.

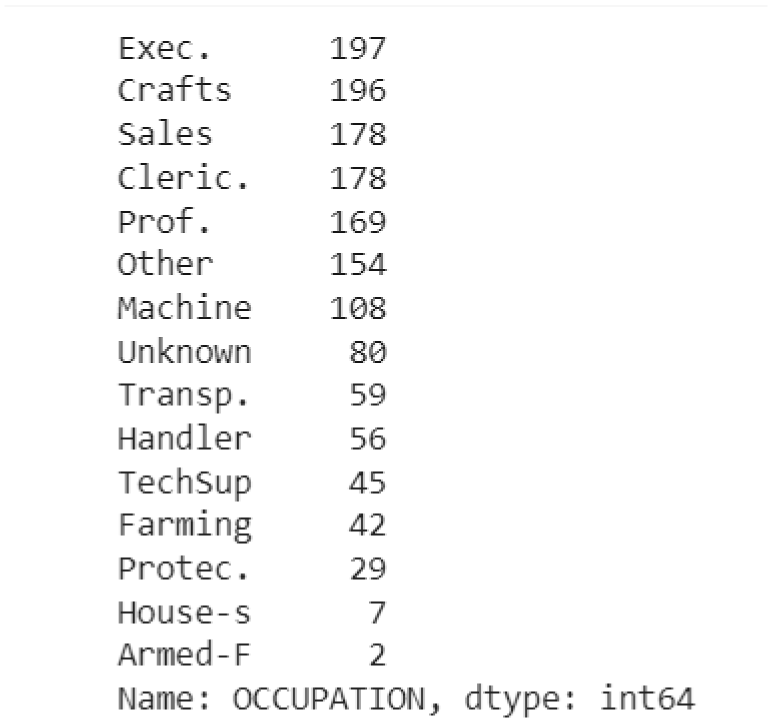
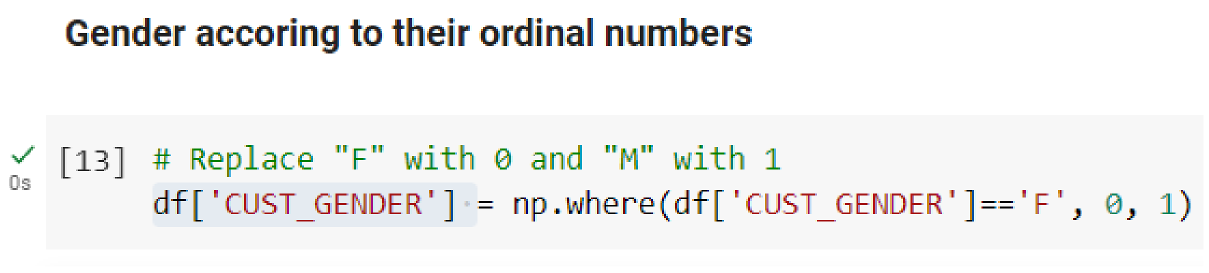

```
Exec.       197
Crafts      196
Sales       178
Cleric.     178
Prof.       169
Other       154
Machine     108
Unknown      80
Transp.      59
Handler      56
TechSup      45
Farming      42
Protec.      29
House-s       7
Armed-F       2
Name: OCCUPATION, dtype: int64
```

Figure 17: Frequency of values of the variable

2.3 Data Transformation

2.3.1 CUST_GENDER into binary F - 0, M -1

Transforming data of the CUST_GENDER variable involves changing the “F” entries to 0 and changing the “M” entries to 1. This can be done with Panda and NumPy.

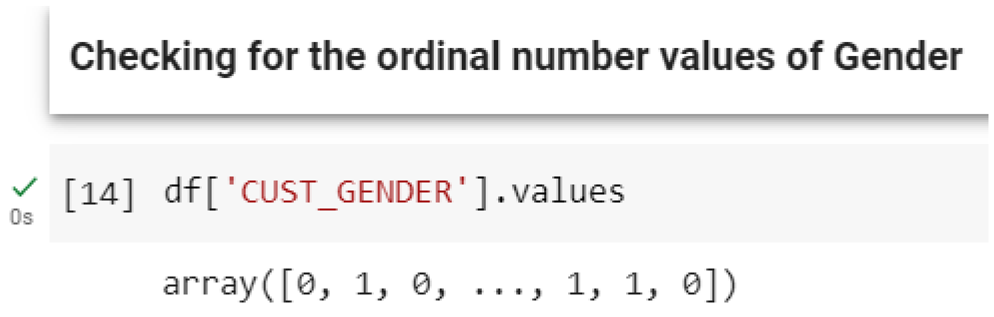


Gender accoring to their ordinal numbers

```
[13] # Replace "F" with 0 and "M" with 1
     df['CUST_GENDER'] = np.where(df['CUST_GENDER']=='F', 0, 1)
```

Figure 18: Code to transform the CUST_GENDER variable entries ”F” to 0 and “M” to 1

The figure above uses NumPy to transform the CUST_GENDER variable uses NumPy to assign “F” to 0 and “M” to 1. It then saves the modified data frame to a new CSV file



Checking for the ordinal number values of Gender

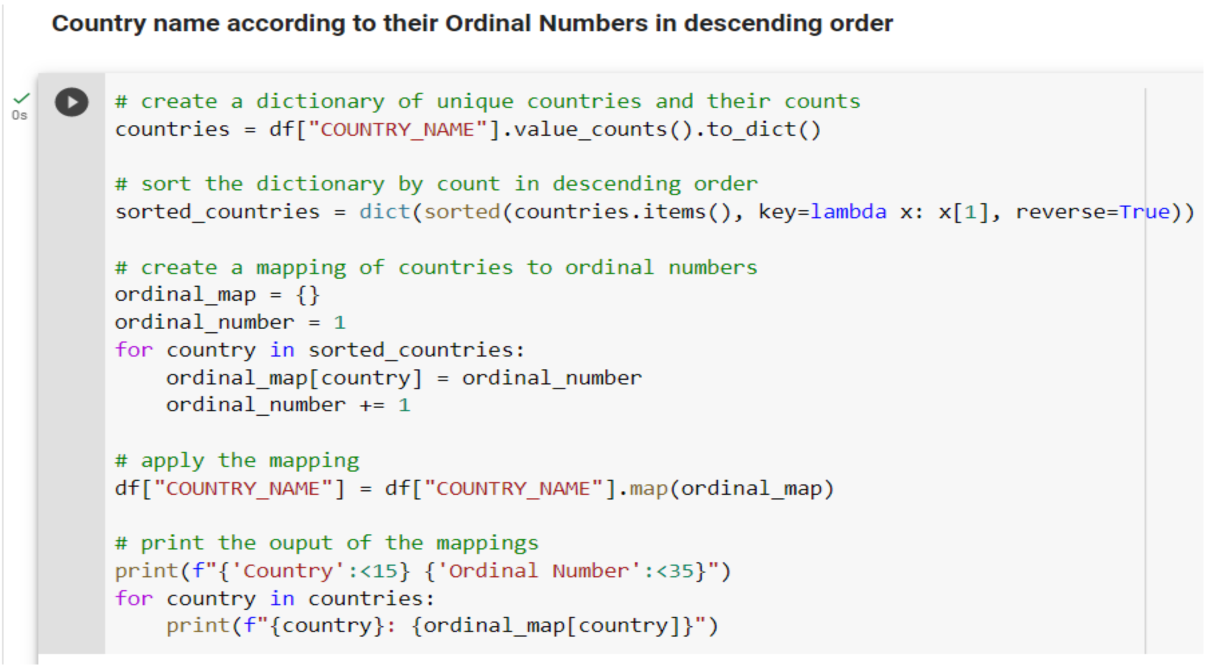
```
[14] df['CUST_GENDER'].values

     array([0, 1, 0, ..., 1, 1, 0])
```

Figure 19: Snapshot showing the ordinal values of the Gender variable

The figure above shows the results of the code when deployed in Google Colab. One can notice the change in the gender values that have occurred.

2.3.2 COUNTRY_NAME into ordinal numbers based on their occurrence in the data set in descending order.

The code is for dividing country names into ordinal numbers based on their occurrence in the data set in descending order.



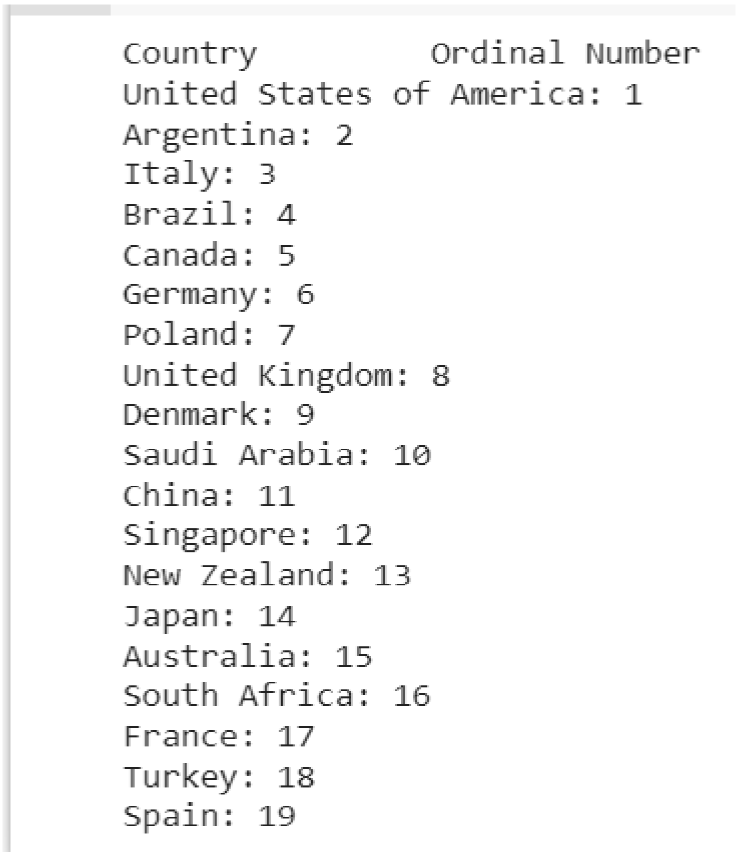
Country name according to their Ordinal Numbers in descending order

```
# create a dictionary of unique countries and their counts
countries = df["COUNTRY_NAME"].value_counts().to_dict()

# sort the dictionary by count in descending order
sorted_countries = dict(sorted(countries.items(), key=lambda x: x[1], reverse=True))

# create a mapping of countries to ordinal numbers
ordinal_map = {}
ordinal_number = 1
for country in sorted_countries:
    ordinal_map[country] = ordinal_number
    ordinal_number += 1

# apply the mapping
df["COUNTRY_NAME"] = df["COUNTRY_NAME"].map(ordinal_map)

# print the ouput of the mappings
print(f"{'Country':<15} {'Ordinal Number':<35}")
for country in countries:
    print(f"{country}: {ordinal_map[country]}")
```

Figure 20: Code to categorize the COUNTRY_NAME into ordinal numbers

The code above creates a dictionary of each unique country and its total number of occurrences and then sorts out the dictionary by count in descending order using the sorted() built-in function. It maps each country to the ordinal number which is its position in the dictionary. The mapping is applied, and the output is printed.

The result of the code is shown below.



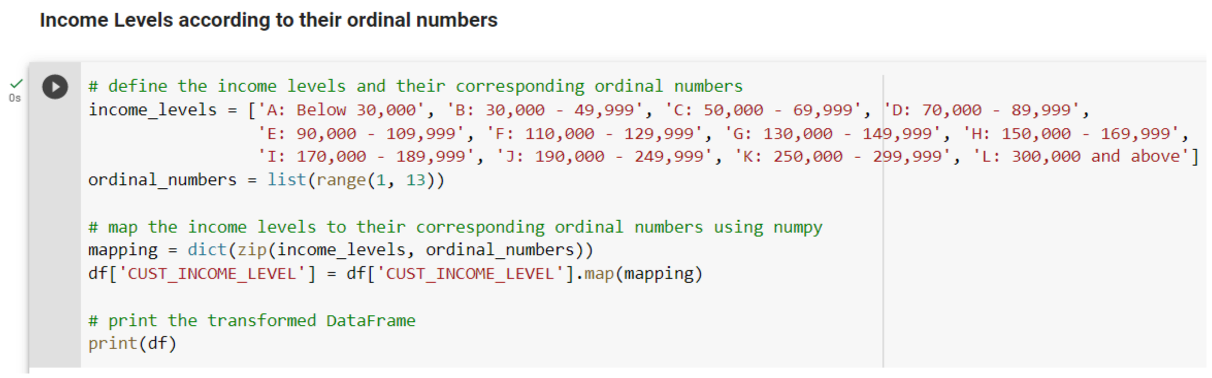
```
Country         Ordinal Number
United States of America: 1
Argentina: 2
Italy: 3
Brazil: 4
Canada: 5
Germany: 6
Poland: 7
United Kingdom: 8
Denmark: 9
Saudi Arabia: 10
China: 11
Singapore: 12
New Zealand: 13
Japan: 14
Australia: 15
South Africa: 16
France: 17
Turkey: 18
Spain: 19
```

Figure 21: Results showing each country with its ordinal number

From the figure above, one can deduce that the United States of America has the highest number of occurrences in the COUNTRY_NAME feature and Spain has the lowest number of occurrences.

2.3.3 CUST_INCOME_LEVEL into ordinal numbers 1 - 12 accordingly.

Transforming CUST_INCOME_LEVEL involves sorting it into the ordinal numbers 1-12 depending on a given range of income.



Income Levels according to their ordinal numbers

```
# define the income levels and their corresponding ordinal numbers
income_levels = ['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']
ordinal_numbers = list(range(1, 13))

# map the income levels to their corresponding ordinal numbers using numpy
mapping = dict(zip(income_levels, ordinal_numbers))
df['CUST_INCOME_LEVEL'] = df['CUST_INCOME_LEVEL'].map(mapping)

# print the transformed DataFrame
print(df)
```

Figure 22: Code to transform ordinal numbers to 1-12 accordingly

The block of code above defines the income levels into the given range and a corresponding ordinal number is assigned to it ranging from 1 to 12. The ordinal number is then mapped to its corresponding income level using a dictionary. The transformed data is then saved to the dataframe.

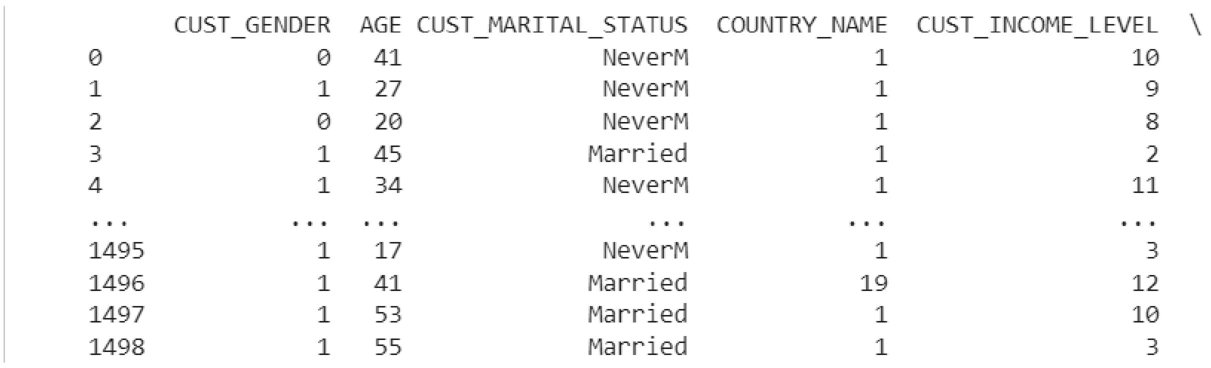
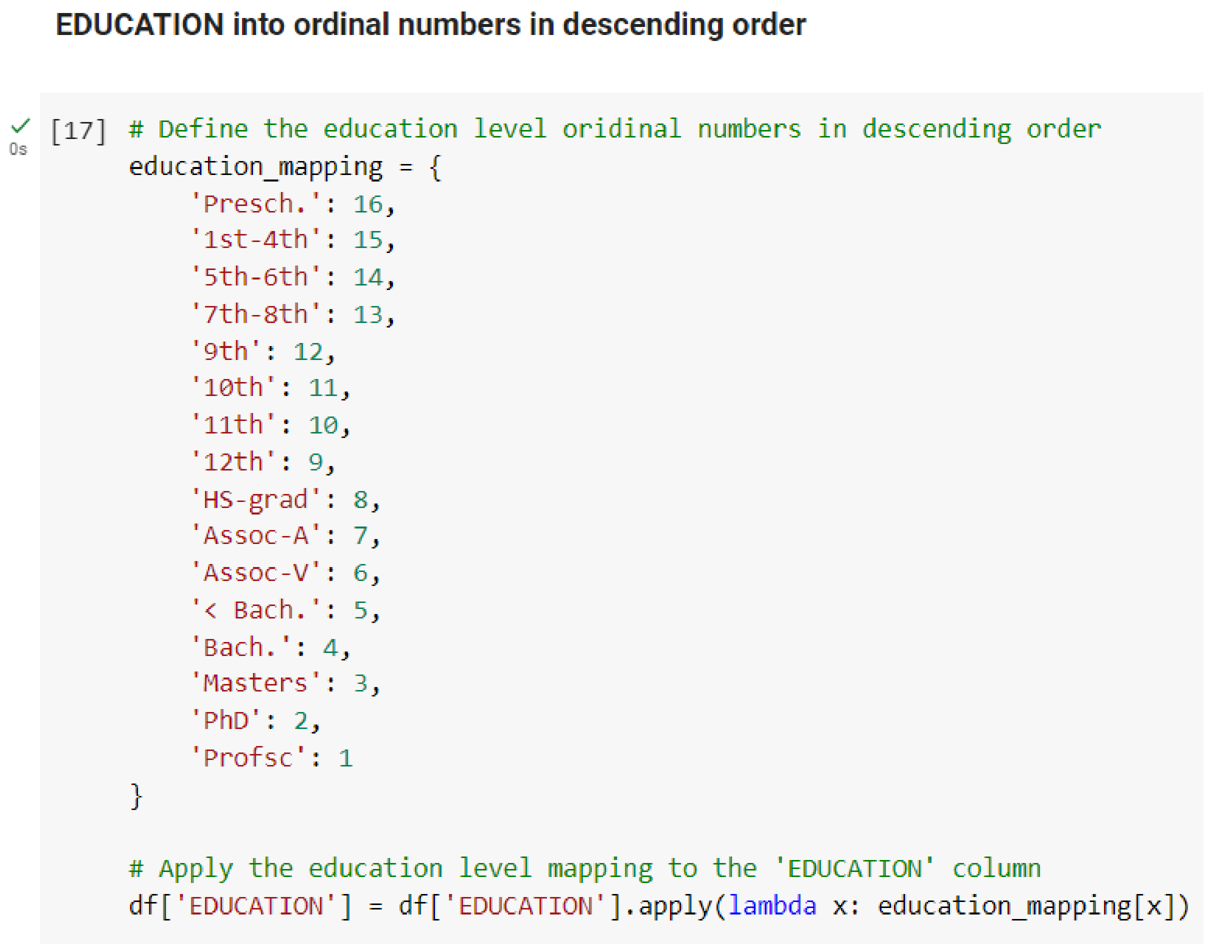

```
      CUST_GENDER  AGE CUST_MARITAL_STATUS  COUNTRY_NAME  CUST_INCOME_LEVEL  \
0               0   41              NeverM             1                 10   
1               1   27              NeverM             1                  9   
2               0   20              NeverM             1                  8   
3               1   45             Married             1                  2   
4               1   34              NeverM             1                 11   
...           ...  ...                 ...           ...                ...   
1495            1   17              NeverM             1                  3   
1496            1   41             Married            19                 12   
1497            1   53             Married             1                 10   
1498            1   55             Married             1                  3   
```

Figure 23: A snapshot to show the transformed income level.

The results of the transformed income level are shown above. There are 13 levels of income. The highest paying income level 'L: 300,000 and above has the ordinal number 1 and the lowest paying income level 'A: Below 30,000' has the ordinal number 13

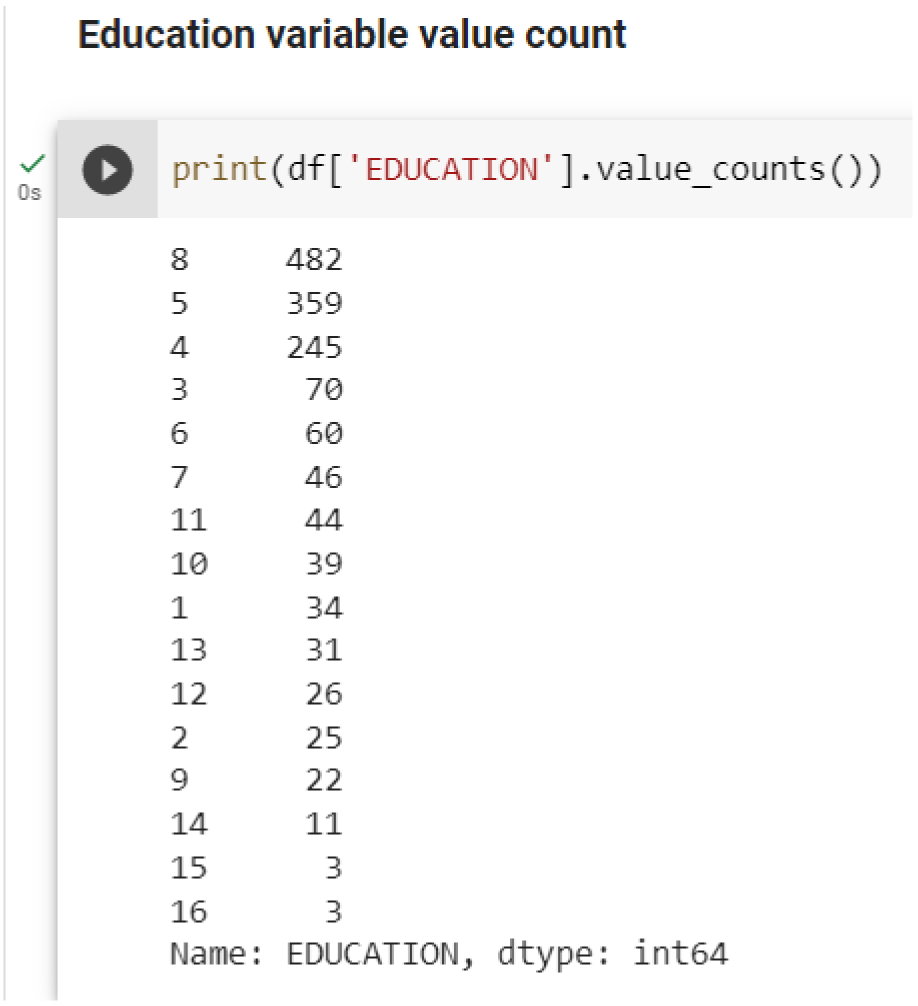
2.3.4 EDUCATION into ordinal numbers based on USA education level in descending order.


EDUCATION into ordinal numbers in descending order

```
[17] # Define the education level oridinal numbers in descending order
     education_mapping = {
         'Presch.': 16,
         '1st-4th': 15,
         '5th-6th': 14,
         '7th-8th': 13,
         '9th': 12,
         '10th': 11,
         '11th': 10,
         '12th': 9,
         'HS-grad': 8,
         'Assoc-A': 7,
         'Assoc-V': 6,
         '< Bach.': 5,
         'Bach.': 4,
         'Masters': 3,
         'PhD': 2,
         'Profsc': 1
     }

     # Apply the education level mapping to the 'EDUCATION' column
     df['EDUCATION'] = df['EDUCATION'].apply(lambda x: education_mapping[x])
```

Figure 24: Organising the US Educational System into ordinal numbers

The above shows the code for organizing the US Educational system into ordinal numbers in descending order. The educational levels are mapped to ordinal numbers in descending order in a dictionary. The mapping is then applied to the EDUCATION feature in the data frame.



Education variable value count

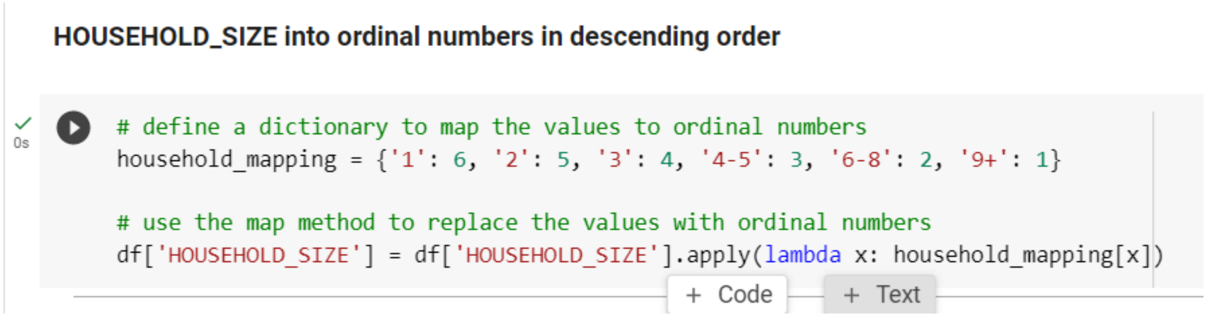
```
print(df['EDUCATION'].value_counts())
```

```
8     482
5     359
4     245
3      70
6      60
7      46
11     44
10     39
1      34
13     31
12     26
2      25
9      22
14     11
15      3
16      3
Name: EDUCATION, dtype: int64
```

Figure 25: Snapshot of the Education value count after being organized into Ordinal numbers

The snapshot above shows the value count of the feature after it has been organized based on the ordinal number. Notice that it has been replaced with numerical values.

2.3.5 HOUSEHOLD_SIZE into ordinal numbers based on the number of rooms.



HOUSEHOLD_SIZE into ordinal numbers in descending order

```
# define a dictionary to map the values to ordinal numbers
household_mapping = {'1': 6, '2': 5, '3': 4, '4-5': 3, '6-8': 2, '9+': 1}

# use the map method to replace the values with ordinal numbers
df['HOUSEHOLD_SIZE'] = df['HOUSEHOLD_SIZE'].apply(lambda x: household_mapping[x])
```

Figure 26: Organizing HOUSEHOLD_SIZE into ordinal numbers based on the number of rooms

The pre-processed data is then saved alongside other columns into the dataframe.

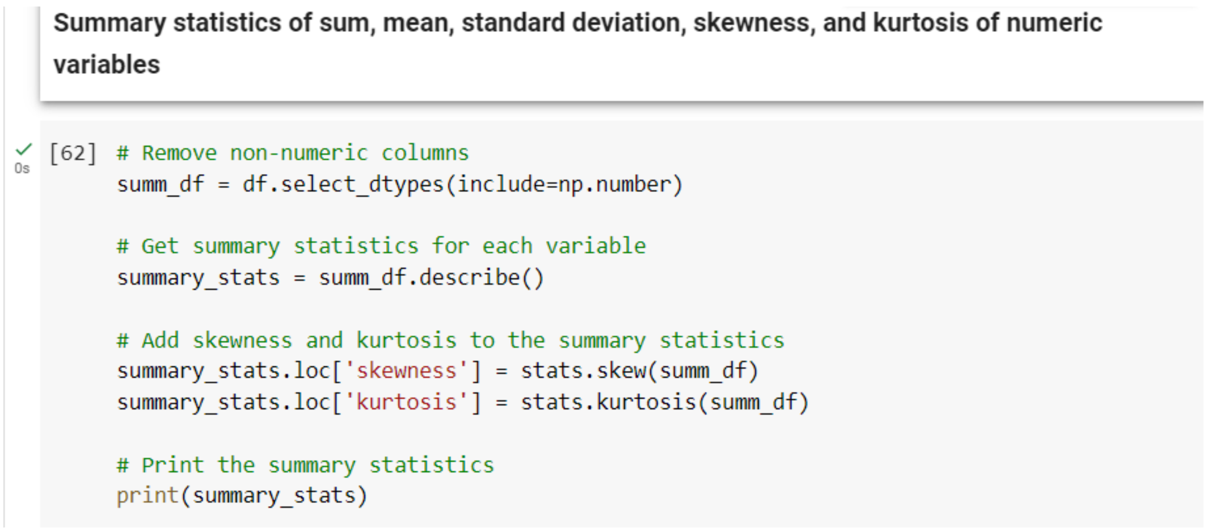
3 Data Analysis

Data analysis involves collecting data and using analytical tools to extract meaningful information to enhance and help make better decisions for users. One can examine various parts of data such as the sum, mean, standard deviation, and other properties. It helps show meaningful trends and the direction the data is headed. These operations can be performed with the help of the Pandas library.

3.1 Summary Statistics

They are numerical values that help to describe the main features of a dataset. They help to understand the characteristics of the data without having to examine all individual observations.

Summary statistics are used in the analysis for a summary of the main features of the dataset. They can then be used to compare different datasets, identify outliers, and detect trends in the data.



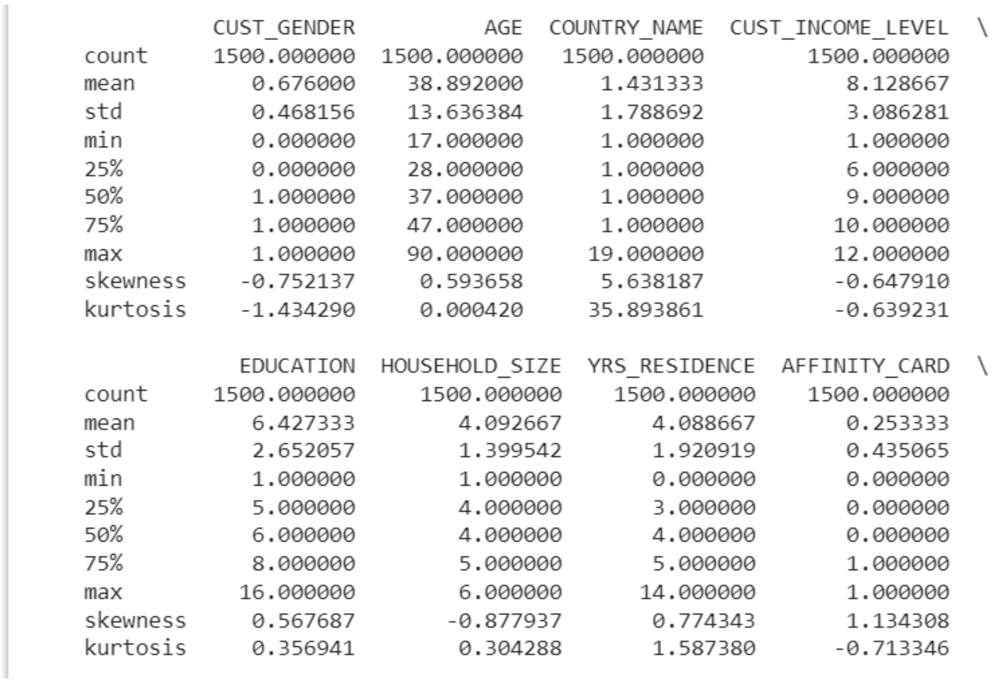
Summary statistics of sum, mean, standard deviation, skewness, and kurtosis of numeric variables

✓ 0s [62]

```
# Remove non-numeric columns
summ_df = df.select_dtypes(include=np.number)

# Get summary statistics for each variable
summary_stats = summ_df.describe()

# Add skewness and kurtosis to the summary statistics
summary_stats.loc['skewness'] = stats.skew(summ_df)
summary_stats.loc['kurtosis'] = stats.kurtosis(summ_df)

# Print the summary statistics
print(summary_stats)
```

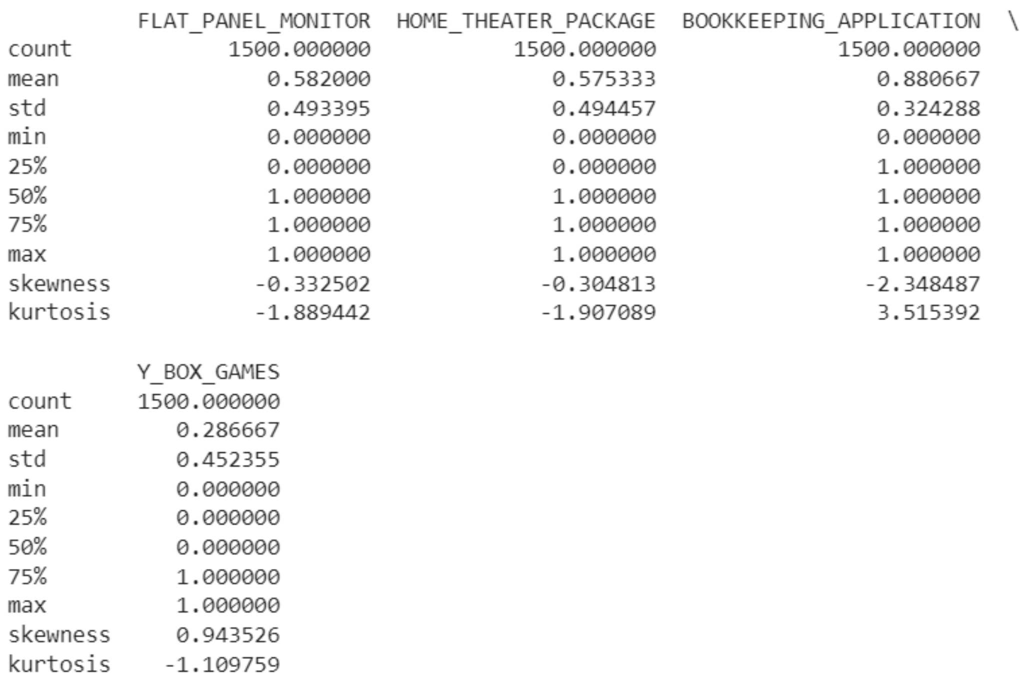
Figure 27: Code to show sum, mean, standard deviation, skewness, and kurtosis of all variables

The snapshot above uses pandas to call the describe() function which is the sum, mean, and standard deviation of the dataset. It then adds skewness and kurtosis to the summary and prints the outputs.



```
          CUST_GENDER          AGE  COUNTRY_NAME  CUST_INCOME_LEVEL  \
count     1500.000000  1500.000000   1500.000000        1500.000000
mean         0.676000    38.892000      1.431333           8.128667
std          0.468156    13.636384      1.788692           3.086281
min          0.000000    17.000000      1.000000           1.000000
25%          0.000000    28.000000      1.000000           6.000000
50%          1.000000    37.000000      1.000000           9.000000
75%          1.000000    47.000000      1.000000          10.000000
max          1.000000    90.000000     19.000000          12.000000
skewness    -0.752137     0.593658      5.638187          -0.647910
kurtosis    -1.434290     0.000420     35.893861          -0.639231

            EDUCATION  HOUSEHOLD_SIZE  YRS_RESIDENCE  AFFINITY_CARD  \
count     1500.000000     1500.000000    1500.000000    1500.000000
mean         6.427333        4.092667       4.088667       0.253333
std          2.652057        1.399542       1.920919       0.435065
min          1.000000        1.000000       0.000000       0.000000
25%          5.000000        4.000000       3.000000       0.000000
50%          6.000000        4.000000       4.000000       0.000000
75%          8.000000        5.000000       5.000000       1.000000
max         16.000000        6.000000      14.000000       1.000000
skewness     0.567687       -0.877937       0.774343       1.134308
kurtosis     0.356941        0.304288       1.587380      -0.713346
```

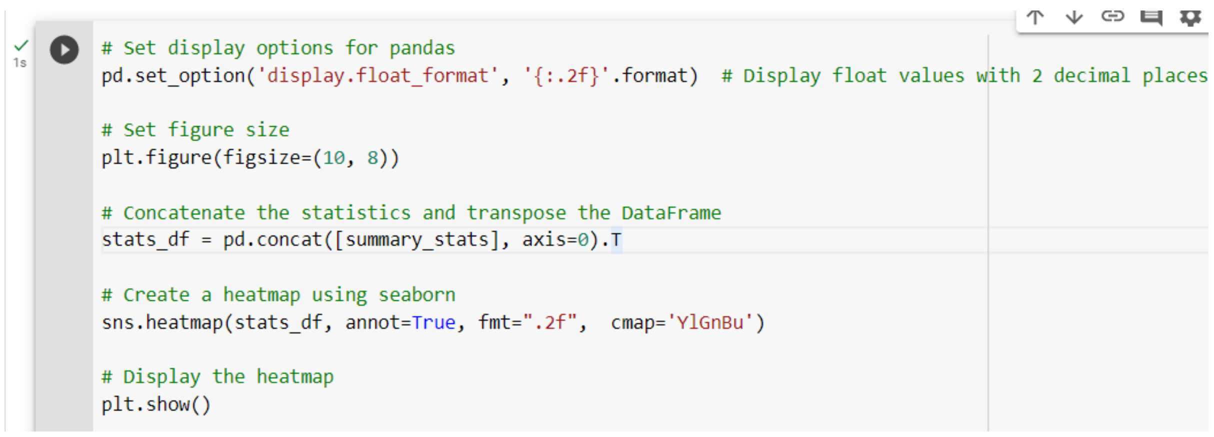
Figure 28: sum, mean, standard deviation, skewness, and kurtosis of some variables



```
          FLAT_PANEL_MONITOR  HOME_THEATER_PACKAGE  BOOKKEEPING_APPLICATION  \
count            1500.000000           1500.000000              1500.000000
mean                0.582000              0.575333                 0.880667
std                 0.493395              0.494457                 0.324288
min                 0.000000              0.000000                 0.000000
25%                 0.000000              0.000000                 1.000000
50%                 1.000000              1.000000                 1.000000
75%                 1.000000              1.000000                 1.000000
max                 1.000000              1.000000                 1.000000
skewness           -0.332502             -0.304813                -2.348487
kurtosis           -1.889442             -1.907089                 3.515392

          Y_BOX_GAMES
count     1500.000000
mean         0.286667
std          0.452355
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          1.000000
skewness     0.943526
kurtosis    -1.109759
```

Figure 29: sum, mean, standard deviation, skewness, and kurtosis of other variables

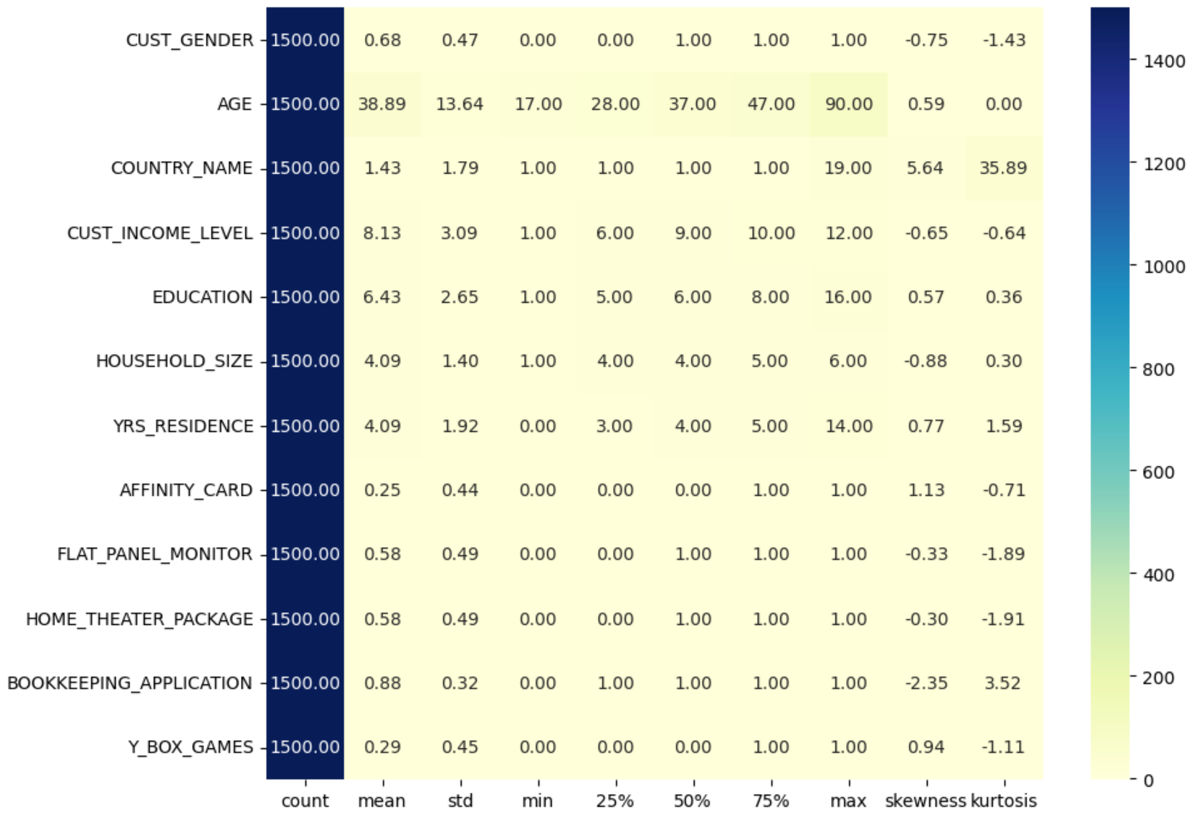
The results in Figures 28 and 29 show the summary statistics of all the features.



```
# Set display options for pandas
pd.set_option('display.float_format', '{:.2f}'.format)  # Display float values with 2 decimal places

# Set figure size
plt.figure(figsize=(10, 8))

# Concatenate the statistics and transpose the DataFrame
stats_df = pd.concat([summary_stats], axis=0).T

# Create a heatmap using seaborn
sns.heatmap(stats_df, annot=True, fmt=".2f",  cmap='YlGnBu')

# Display the heatmap
plt.show()
```

Figure 30: Code to get Summary statistics heatmap

The code above uses seaborn to visualize the heatmap of the summary statistics and the heatmap is displayed below.

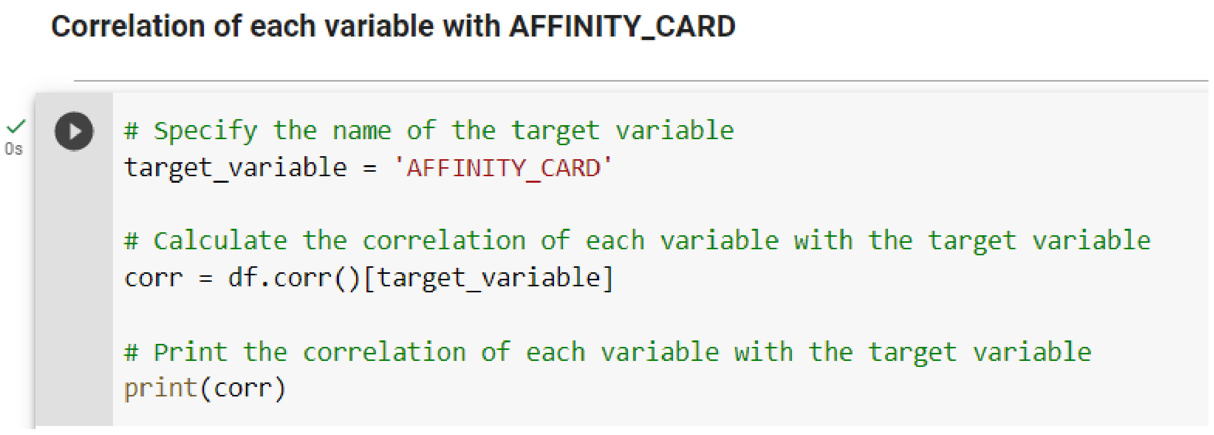


	count	mean	std	min	25%	50%	75%	max	skewness	kurtosis
CUST_GENDER	1500.00	0.68	0.47	0.00	0.00	1.00	1.00	1.00	-0.75	-1.43
AGE	1500.00	38.89	13.64	17.00	28.00	37.00	47.00	90.00	0.59	0.00
COUNTRY_NAME	1500.00	1.43	1.79	1.00	1.00	1.00	1.00	19.00	5.64	35.89
CUST_INCOME_LEVEL	1500.00	8.13	3.09	1.00	6.00	9.00	10.00	12.00	-0.65	-0.64
EDUCATION	1500.00	6.43	2.65	1.00	5.00	6.00	8.00	16.00	0.57	0.36
HOUSEHOLD_SIZE	1500.00	4.09	1.40	1.00	4.00	4.00	5.00	6.00	-0.88	0.30
YRS_RESIDENCE	1500.00	4.09	1.92	0.00	3.00	4.00	5.00	14.00	0.77	1.59
AFFINITY_CARD	1500.00	0.25	0.44	0.00	0.00	0.00	1.00	1.00	1.13	-0.71
FLAT_PANEL_MONITOR	1500.00	0.58	0.49	0.00	0.00	1.00	1.00	1.00	-0.33	-1.89
HOME_THEATER_PACKAGE	1500.00	0.58	0.49	0.00	0.00	1.00	1.00	1.00	-0.30	-1.91
BOOKKEEPING_APPLICATION	1500.00	0.88	0.32	0.00	1.00	1.00	1.00	1.00	-2.35	3.52
Y_BOX_GAMES	1500.00	0.29	0.45	0.00	0.00	0.00	1.00	1.00	0.94	-1.11

Figure 31: Heatmap of the Summary Statistics

3.2 Correlation

Finding the relationship between variables is important as it can be used to understand the strength and direction of the relationship between the variables.

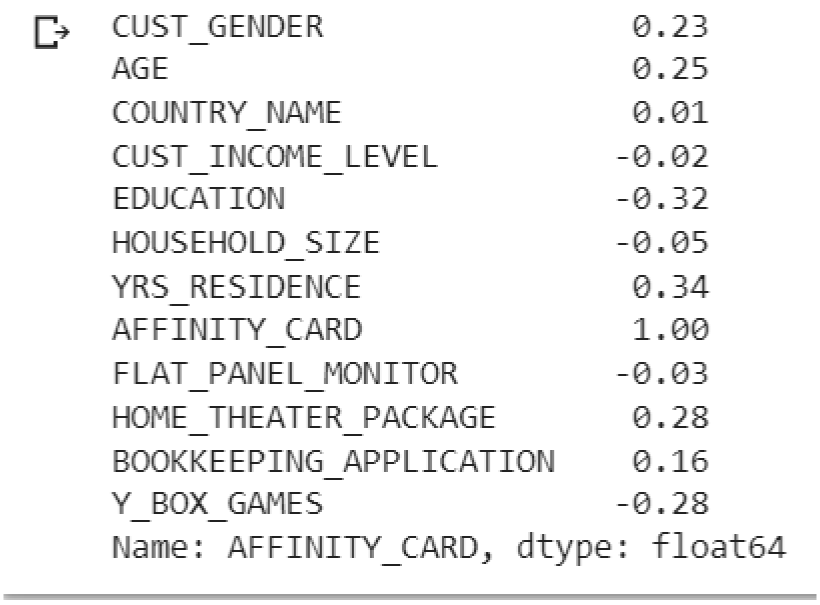


Correlation of each variable with AFFINITY_CARD

```
# Specify the name of the target variable
target_variable = 'AFFINITY_CARD'

# Calculate the correlation of each variable with the target variable
corr = df.corr()[target_variable]

# Print the correlation of each variable with the target variable
print(corr)
```

Figure 32: Code to show the correlation of each variable with AFFINITY_CARD

The above snapshot specifies that AFFINITY_CARD is the target variable. it then stores the correlation of each variable with the target variable and stores it in color and then prints the result.

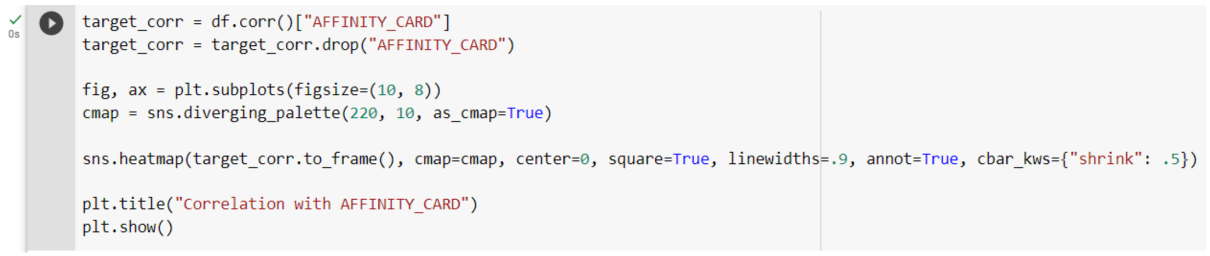


```
CUST_GENDER                 0.23
AGE                         0.25
COUNTRY_NAME                0.01
CUST_INCOME_LEVEL          -0.02
EDUCATION                  -0.32
HOUSEHOLD_SIZE             -0.05
YRS_RESIDENCE               0.34
AFFINITY_CARD               1.00
FLAT_PANEL_MONITOR         -0.03
HOME_THEATER_PACKAGE        0.28
BOOKKEEPING_APPLICATION     0.16
Y_BOX_GAMES                -0.28
Name: AFFINITY_CARD, dtype: float64
```

Figure 33: Correlation of each variable with AFFINITY_CARD

The output of the correlation is displayed in the figure above. It can be discovered that some of the variables are negatively correlated which means that as the value of the target variable increases, the value of that specific variable reduces and vice versa. The variable which is the most positively correlated with the target variable is the YRS_RESIDENCE variable.

The correlation heatmap of the variables with the target variable can also be visualized below. It helps give a better understanding.



```
target_corr = df.corr()["AFFINITY_CARD"]
target_corr = target_corr.drop("AFFINITY_CARD")

fig, ax = plt.subplots(figsize=(10, 8))
cmap = sns.diverging_palette(220, 10, as_cmap=True)

sns.heatmap(target_corr.to_frame(), cmap=cmap, center=0, square=True, linewidths=.9, annot=True, cbar_kws={"shrink": .5})

plt.title("Correlation with AFFINITY_CARD")
plt.show()
```

Figure 34: Code to show correlation heatmap of each variable with AFFINITY_CARD

The heatmap visualizes the correlations that have been calculated in Pandas.

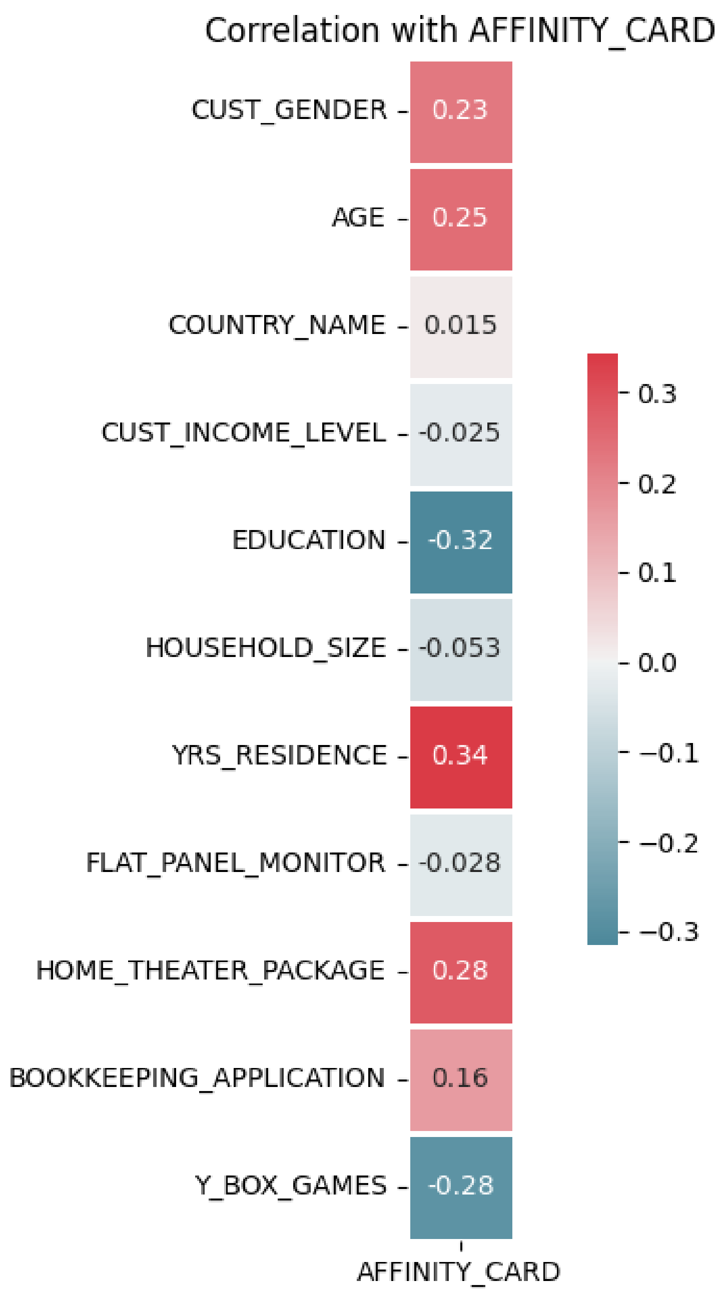


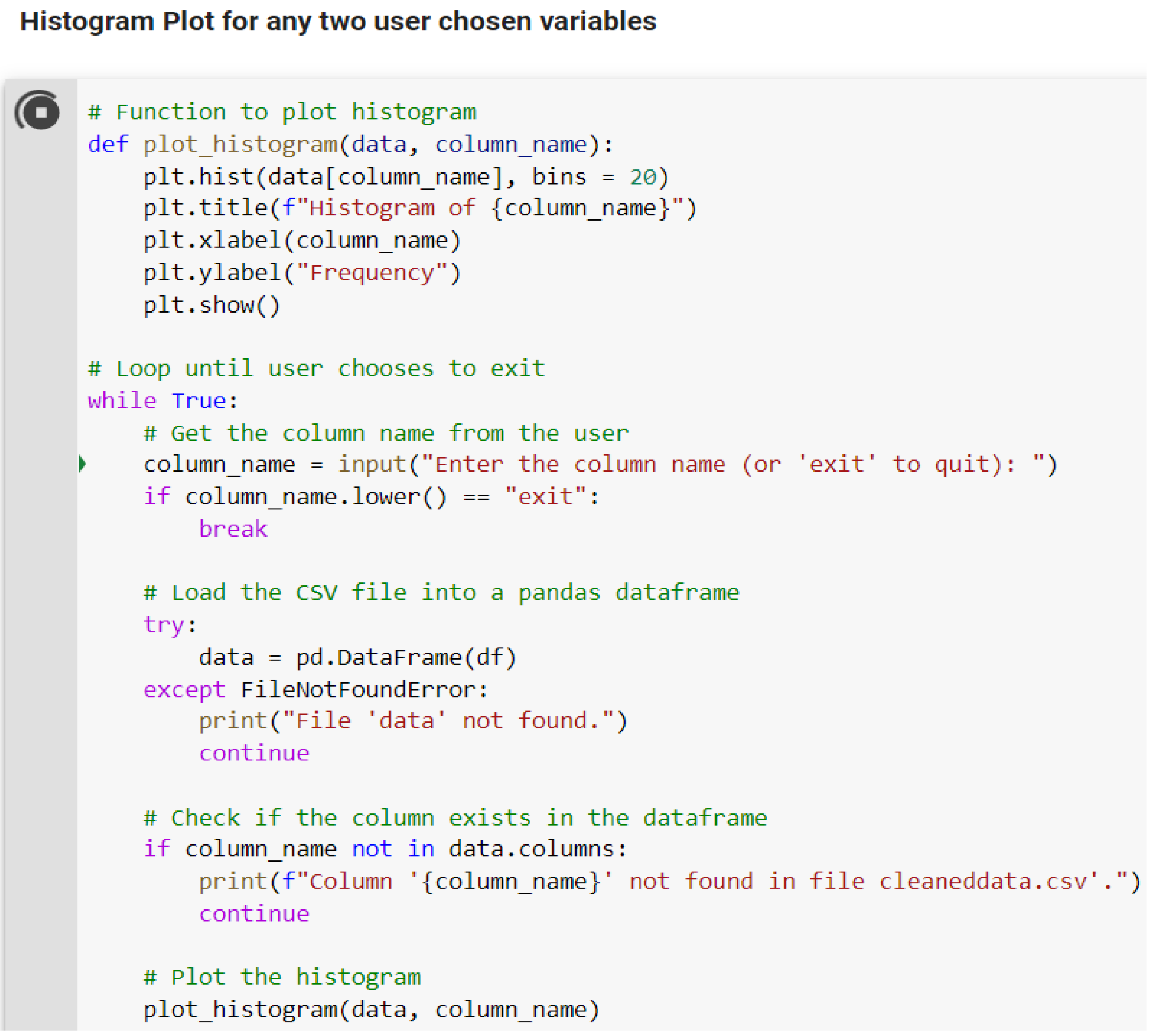
Figure 35: Correlation matrix heatmap of each variable with AFFINITY_CARD

4 Data Exploration

This is the stage where there is data understanding by visualization which involves bar charts, scatter plots, and histograms. The specification at this stage is to plot scatter plots and histograms of any two user-chosen variables.

4. 1 Histograms Plot

Histogram is a graphical representation of the number of occurrences of a dataset. It provides a visualization of the summary of the variables. Matplotlib is most times used for plotting histograms.



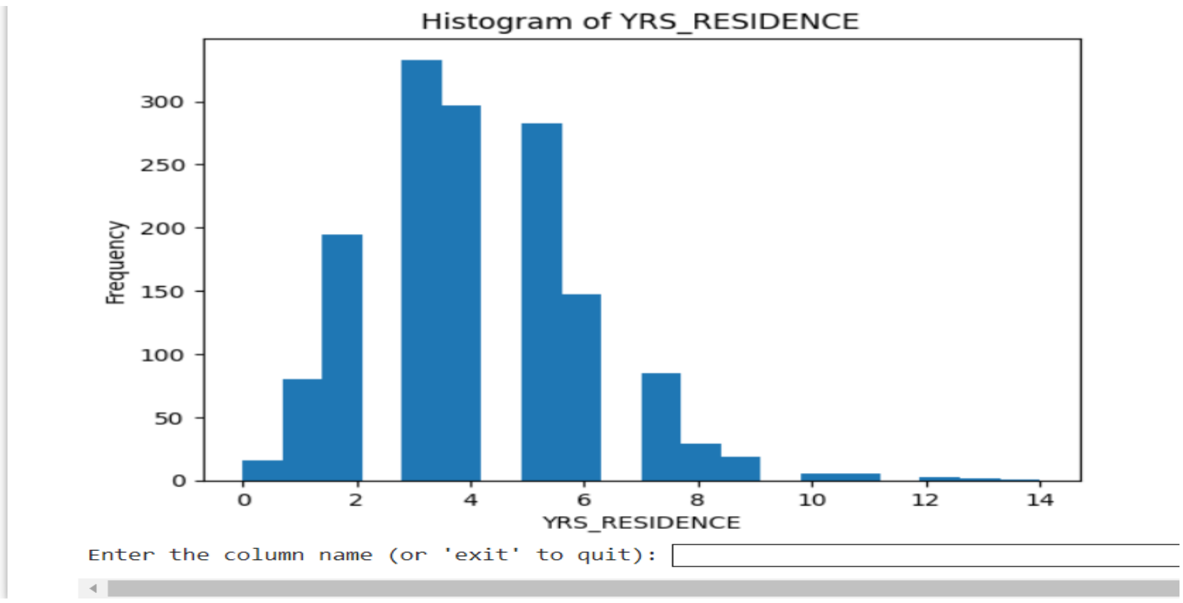
Histogram Plot for any two user chosen variables

```
# Function to plot histogram
def plot_histogram(data, column_name):
    plt.hist(data[column_name], bins = 20)
    plt.title(f"Histogram of {column_name}")
    plt.xlabel(column_name)
    plt.ylabel("Frequency")
    plt.show()

# Loop until user chooses to exit
while True:
    # Get the column name from the user
    column_name = input("Enter the column name (or 'exit' to quit): ")
    if column_name.lower() == "exit":
        break

    # Load the CSV file into a pandas dataframe
    try:
        data = pd.DataFrame(df)
    except FileNotFoundError:
        print("File 'data' not found.")
        continue

    # Check if the column exists in the dataframe
    if column_name not in data.columns:
        print(f"Column '{column_name}' not found in file cleaneddata.csv'.")
        continue

    # Plot the histogram
    plot_histogram(data, column_name)
```

Figure 36: Code to show the histogram of any user-chosen variable

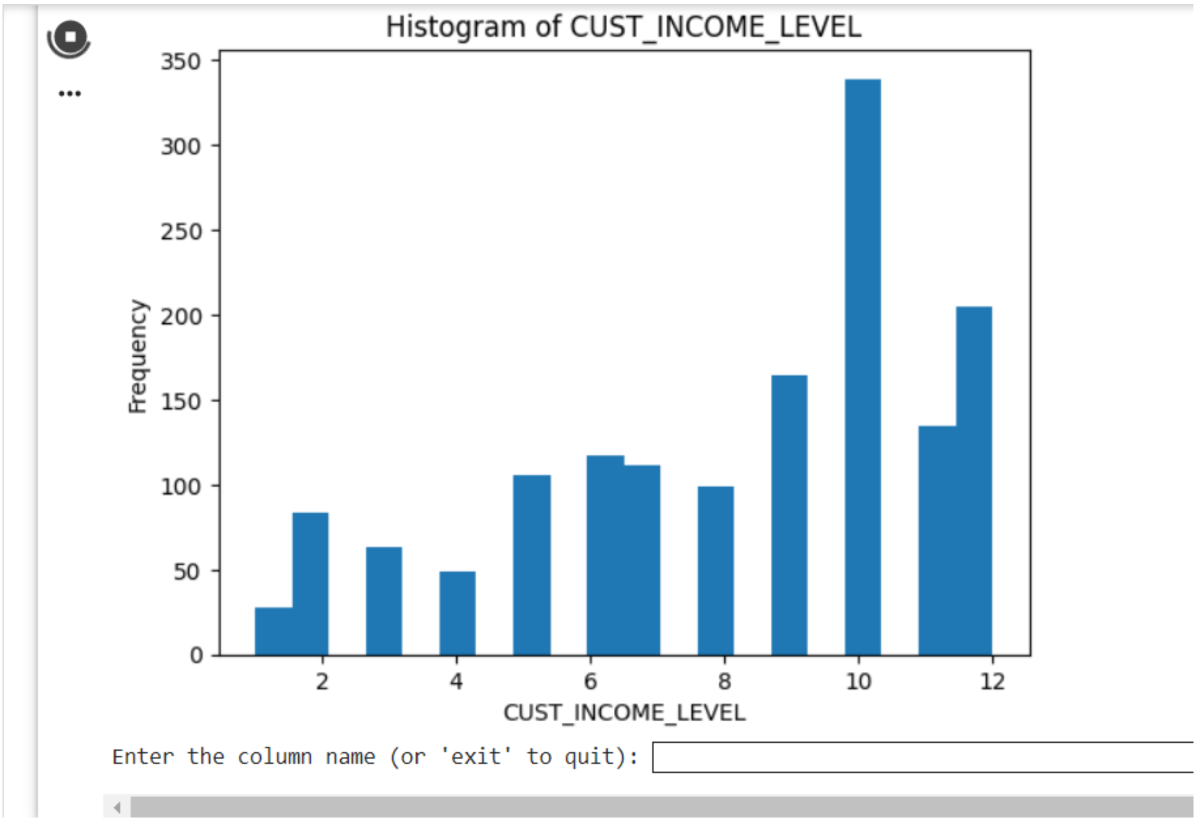
The code above contains the function plot_histogram which holds the properties data and column name. The necessary properties of the histogram are listed, and a while loop is constructed that asks for variables as inputs and plots them against each other. This function only terminates when the user types ‘exit’ in the text box.

The outputs of the code are shown below.



```
Enter the column name (or 'exit' to quit):
```

Figure 37: Histogram of YRS_RESIDENCE



```
Enter the column name (or 'exit' to quit):
```

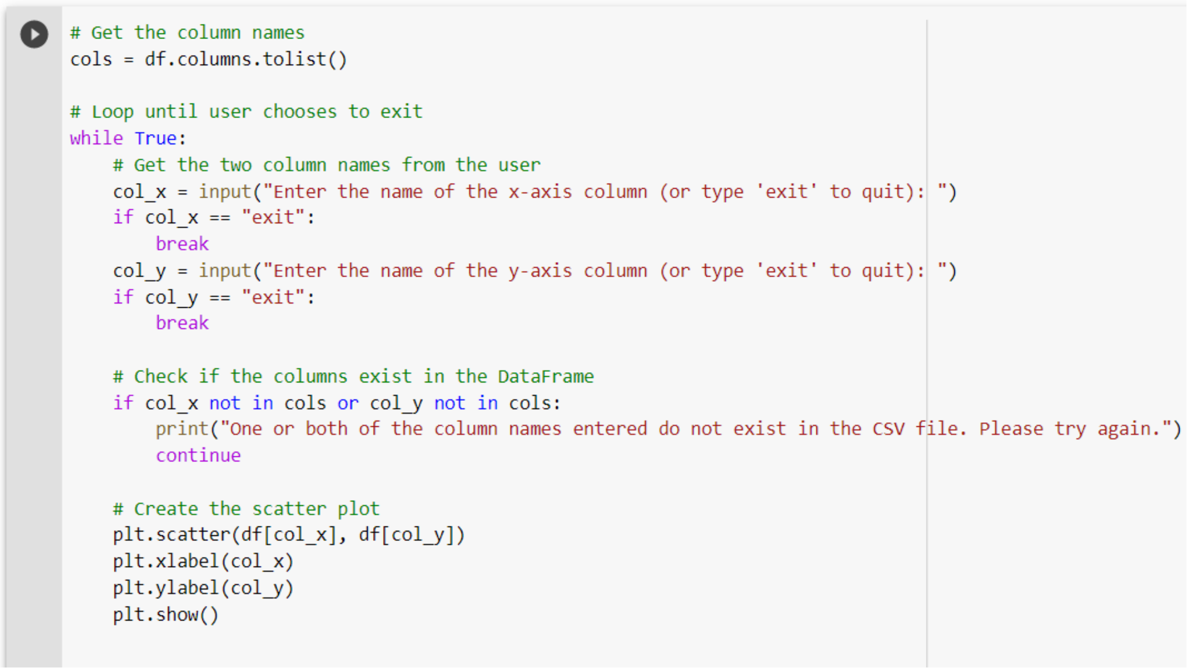
Figure 38: Histogram of CUST_INCOME_LEVEL

4.2 Scatter Plot

A Scatter plot shows the relationship between two continuous variables which are on the x and y axes. The position of each value represents the corresponding position on both axes. The Matplotlib library also provides functionality for scatter plots.

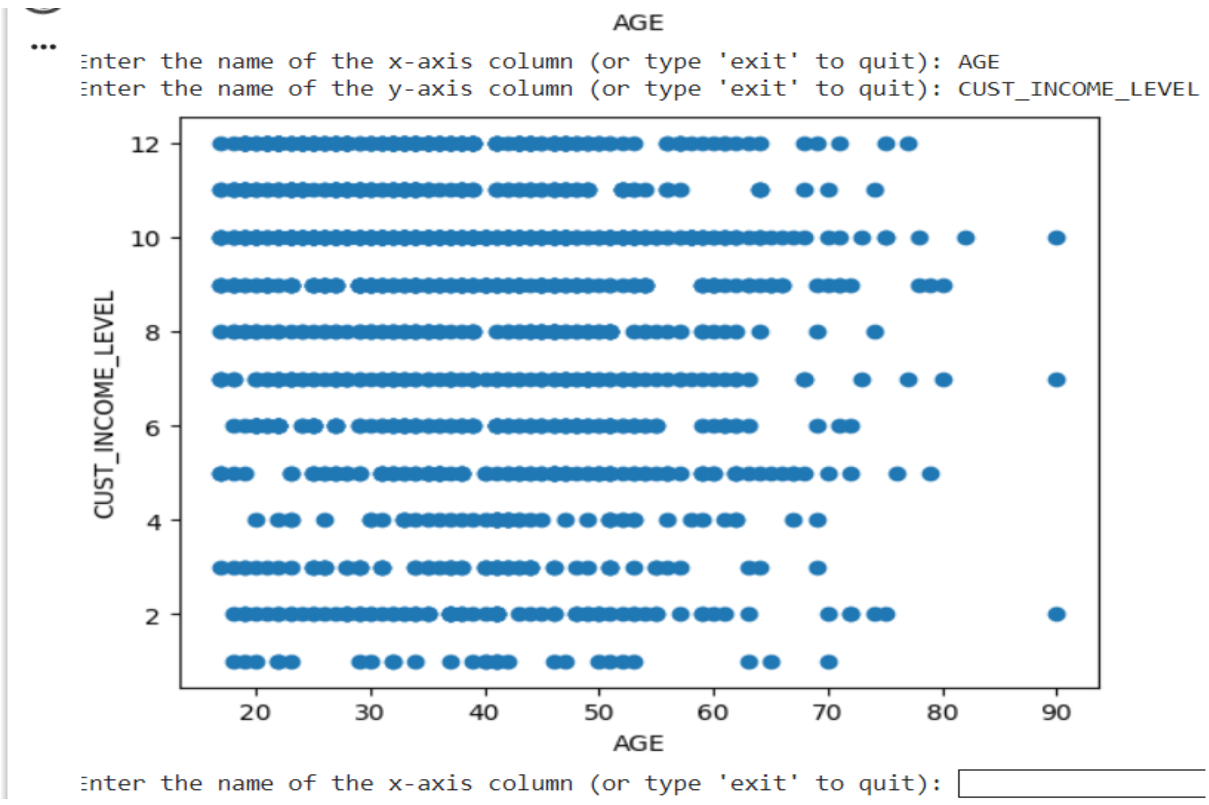
The code below like the histogram lists out all the properties of the scatter plot concerning the matplotlib library. It then loops until a user exits.

Scatter Plot for any two user chosen variables



```python
# Get the column names
cols = df.columns.tolist()

# Loop until user chooses to exit
while True:
    # Get the two column names from the user
    col_x = input("Enter the name of the x-axis column (or type 'exit' to quit): ")
    if col_x == "exit":
        break
    col_y = input("Enter the name of the y-axis column (or type 'exit' to quit): ")
    if col_y == "exit":
        break

    # Check if the columns exist in the DataFrame
    if col_x not in cols or col_y not in cols:
        print("One or both of the column names entered do not exist in the CSV file. Please try again.")
        continue

    # Create the scatter plot
    plt.scatter(df[col_x], df[col_y])
    plt.xlabel(col_x)
    plt.ylabel(col_y)
    plt.show()
```

Figure 39: Code to show a scatter plot of any user of two chosen variables.

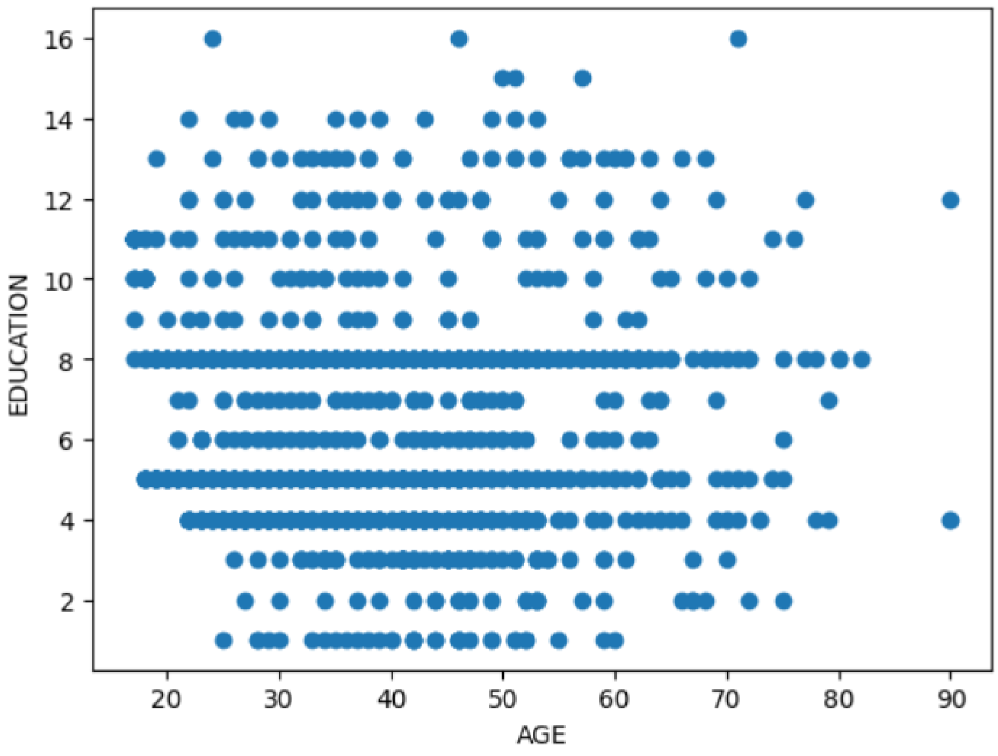


```
AGE
Enter the name of the x-axis column (or type 'exit' to quit): AGE
Enter the name of the y-axis column (or type 'exit' to quit): CUST_INCOME_LEVEL
```

```
Enter the name of the x-axis column (or type 'exit' to quit):
```

Figure 40: Scatter plot of AGE against CUST_INCOME_LEVEL

```
Enter the name of the x-axis column (or type 'exit' to quit): AGE
Enter the name of the y-axis column (or type 'exit' to quit): EDUCATION
```



```
Enter the name of the x-axis column (or type 'exit' to quit):
```

Figure 41: Scatter plot of AGE against EDUCATION

5 Data Mining

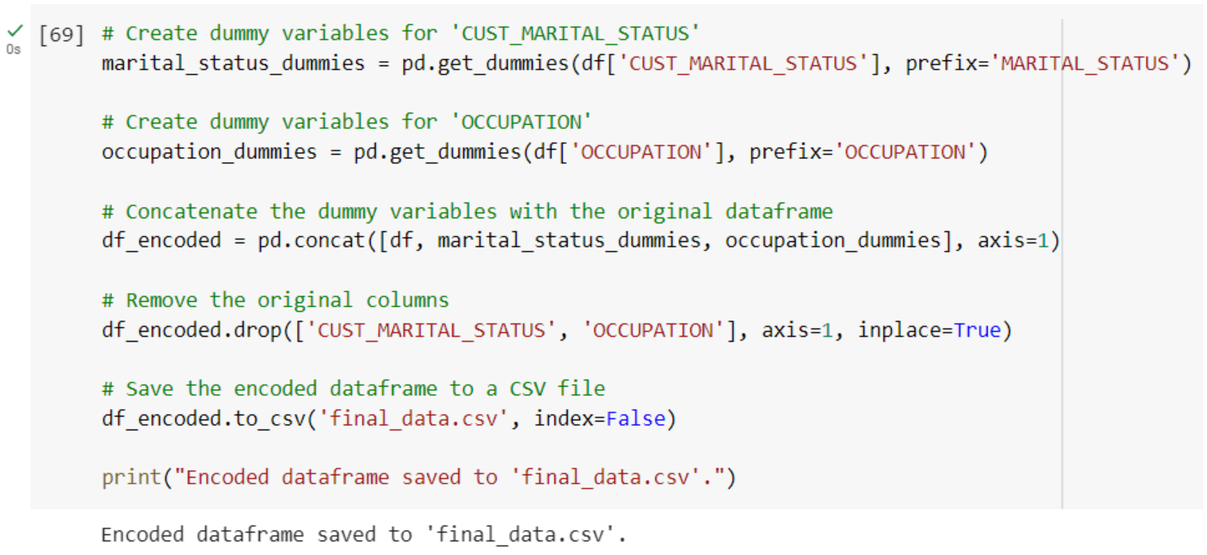
Data Mining involves discovering patterns and trends from large datasets. It involves extracting useful information from cleaned data using various algorithms. In this project, the Random Forest classifier and Logistic Regression are the models used in this analysis. For each model, there will be accuracy calculation, precision, recall, F1 score, specificity, and log loss.

Even though there seems to be a heavy bias towards 0s than 1s for the target variable (AFFINITY_CARD). Undersampling is going to be applied to the dataset before splitting it into training and test data and also before the model is fitted and predicted. Undersampling is a technique to address class imbalance problems by removing instances from the majority class. The model is going to be trained on a baseline and there will be an assessment of its performance metrics such as accuracy, precision, recall, and F1-score.

The data will be split into 80% training and 20% testing data. The models to be considered are Logistic Regression (a popular ML algorithm known for its effectiveness when solving a range of classification and regression problems) and Random Forest Classifier (an ML algorithm used for classification and known for its robustness and effectiveness and uses multiple decision trees to make predictions). Both Logistic Regression and Random Forest Classifier are supervised machine learning models with AFFINITY_CARD as the target variable.

5.1 One-Hot Encoding

Before proceeding, One-hot encoding is going to be applied to the dataset as it still contains categorical data. One-hot encoding is used to transform categorical variables into binary vectors. In this encoding scheme, each category is represented by a binary vector. The vector contains all zeros except for one which contains one indicating that it belongs to that category.

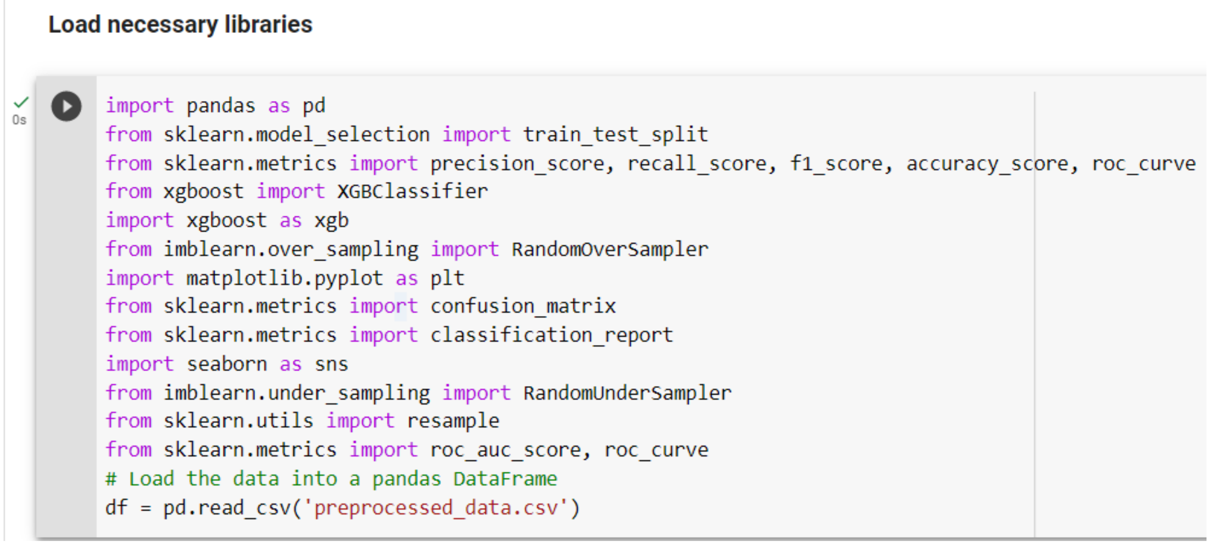


```
[69] # Create dummy variables for 'CUST_MARITAL_STATUS'
     marital_status_dummies = pd.get_dummies(df['CUST_MARITAL_STATUS'], prefix='MARITAL_STATUS')

     # Create dummy variables for 'OCCUPATION'
     occupation_dummies = pd.get_dummies(df['OCCUPATION'], prefix='OCCUPATION')

     # Concatenate the dummy variables with the original dataframe
     df_encoded = pd.concat([df, marital_status_dummies, occupation_dummies], axis=1)

     # Remove the original columns
     df_encoded.drop(['CUST_MARITAL_STATUS', 'OCCUPATION'], axis=1, inplace=True)

     # Save the encoded dataframe to a CSV file
     df_encoded.to_csv('final_data.csv', index=False)

     print("Encoded dataframe saved to 'final_data.csv'.")

     Encoded dataframe saved to 'final_data.csv'.
```

Figure 42: One-Hot Encoding.

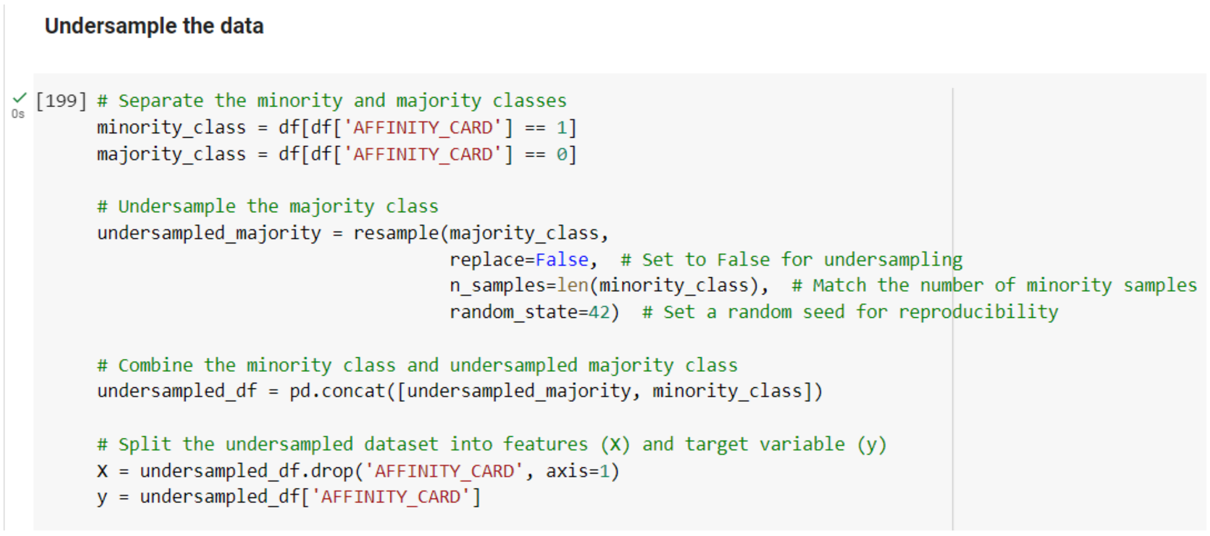
The encoding is saved to a ‘final_data.csv’ file.

5.2 Model 1: Logistic Regression Model


Load necessary libraries

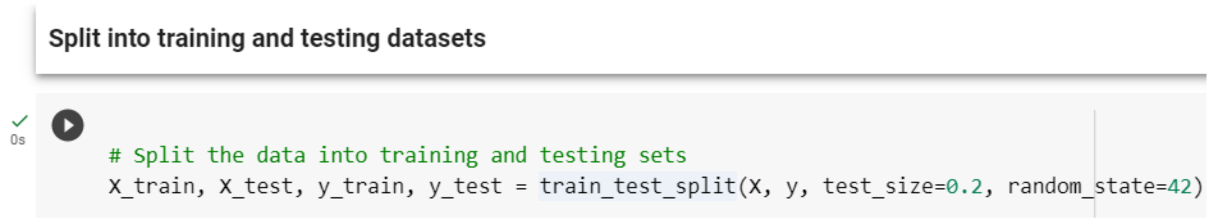
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, roc_curve
from xgboost import XGBClassifier
import xgboost as xgb
from imblearn.over_sampling import RandomOverSampler
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns
from imblearn.under_sampling import RandomUnderSampler
from sklearn.utils import resample
from sklearn.metrics import roc_auc_score, roc_curve
# Load the data into a pandas DataFrame
df = pd.read_csv('preprocessed_data.csv')
```

Figure 43: Loading the dataset and machine learning libraries into the DataFrame


Undersample the data

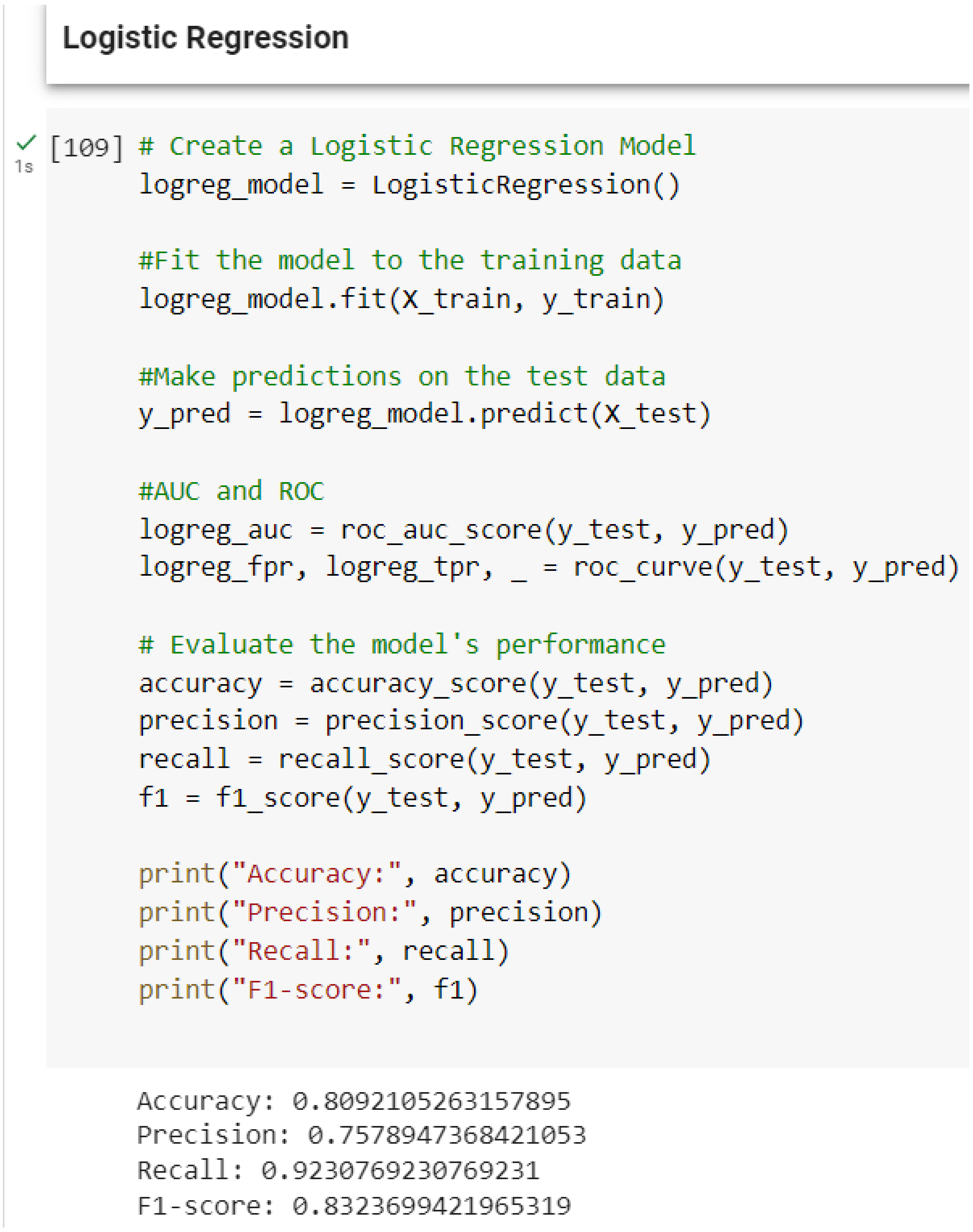
```
[199] # Separate the minority and majority classes
      minority_class = df[df['AFFINITY_CARD'] == 1]
      majority_class = df[df['AFFINITY_CARD'] == 0]

      # Undersample the majority class
      undersampled_majority = resample(majority_class,
                                       replace=False,  # Set to False for undersampling
                                       n_samples=len(minority_class),  # Match the number of minority samples
                                       random_state=42)  # Set a random seed for reproducibility

      # Combine the minority class and undersampled majority class
      undersampled_df = pd.concat([undersampled_majority, minority_class])

      # Split the undersampled dataset into features (X) and target variable (y)
      X = undersampled_df.drop('AFFINITY_CARD', axis=1)
      y = undersampled_df['AFFINITY_CARD']
```

Figure 44: Undersampling the data


Split into training and testing datasets

```
# 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)
```

Figure 45: splitting into training and testing data


Logistic Regression

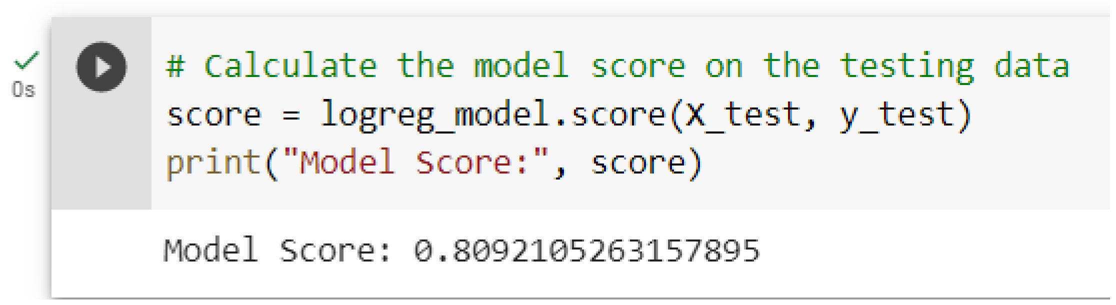
[109]
```
# Create a Logistic Regression Model
logreg_model = LogisticRegression()

#Fit the model to the training data
logreg_model.fit(X_train, y_train)

#Make predictions on the test data
y_pred = logreg_model.predict(X_test)

#AUC and ROC
logreg_auc = roc_auc_score(y_test, y_pred)
logreg_fpr, logreg_tpr, _ = roc_curve(y_test, y_pred)

# Evaluate the model's performance
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)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Accuracy: 0.8092105263157895
Precision: 0.7578947368421053
Recall: 0.9230769230769231
F1-score: 0.8323699421965319
```

Figure 46: The Logistic Regression Model with the results

The accuracy is 0.809. The precision has a score of 0.758, a recall of 0.923, and an F1 score of 0.832.

Now it’s important to calculate the model score of the algorithm. The model score is used to evaluate the performance of the machine learning model.



Calculate the Model Score

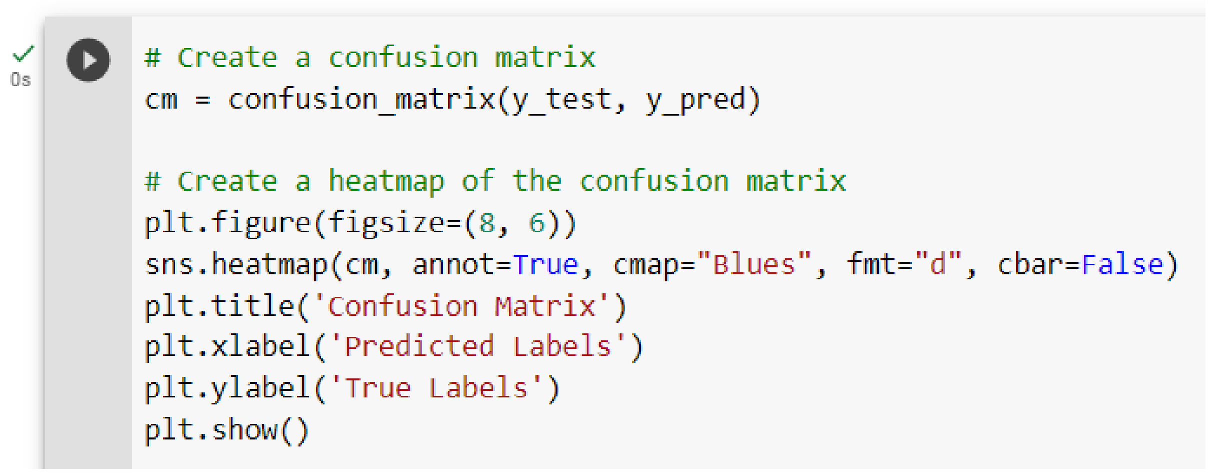
```
# Calculate the model score on the testing data
score = logreg_model.score(X_test, y_test)
print("Model Score:", score)
```

```
Model Score: 0.8092105263157895
```

Figure 47: Model score of the Logistic Regression model

From the snapshot above, the model score is 0.809 or 81% which means the model is 81% good and is performing well.

The next thing is the correlation matrix which is used to evaluate the performance of a classification model.



Create a Confusion Matrix

```
# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Figure 48: Code to create confusion matrix heatmap

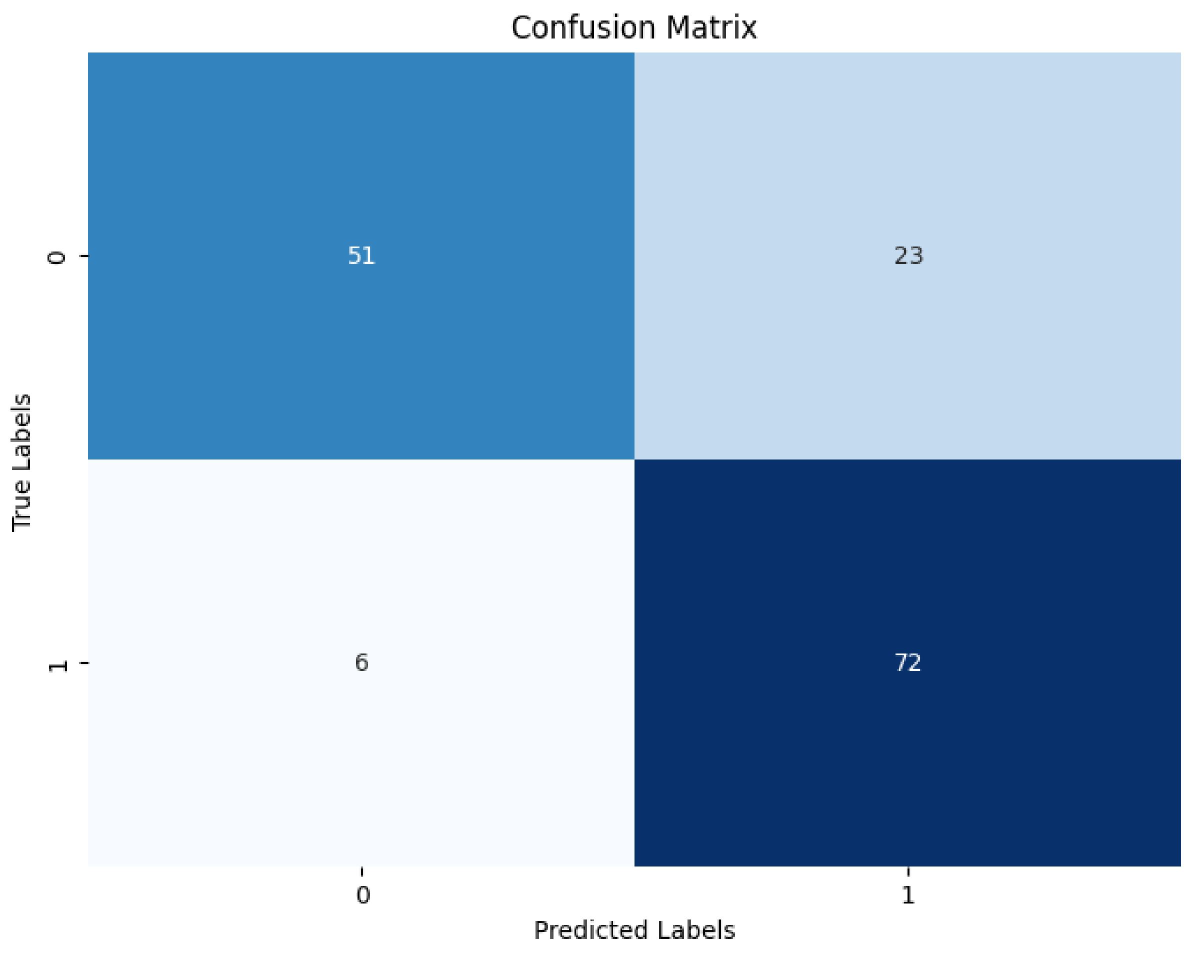
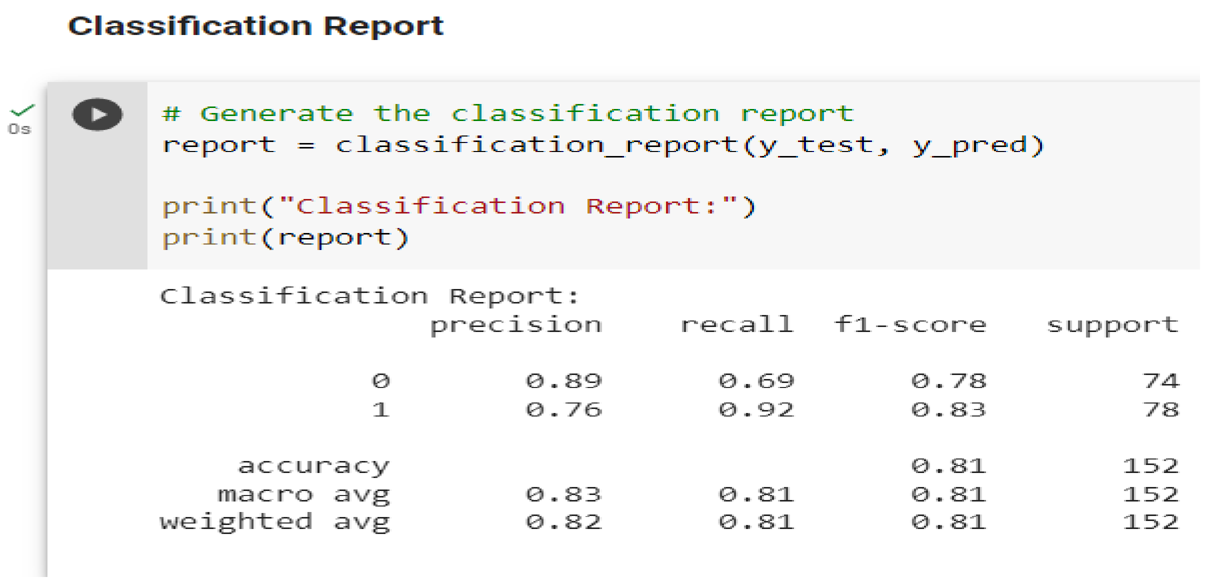


Figure 49 The confusion matrix for the Linear Regression model

From the above, the True Negative (TN) is 51, False Negative (FN) is 6, True Positive (TP) is 72, False Positive (FP) is 23.

The last thing is now to generate a classification report to see how the 1s and 0s performed individually during training



Classification Report

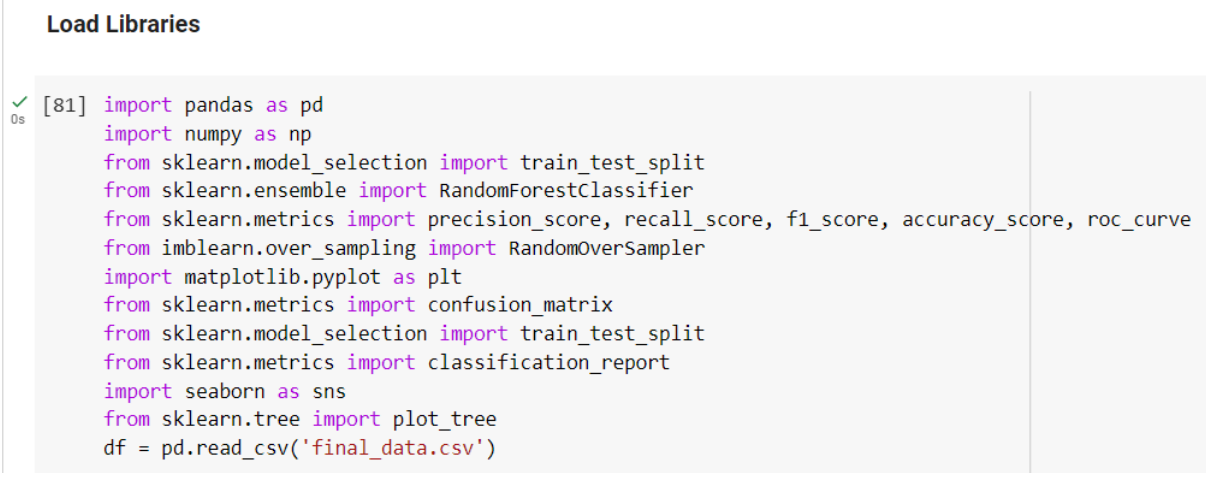
```
# Generate the classification report
report = classification_report(y_test, y_pred)

print("Classification Report:")
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.69      0.78        74
           1       0.76      0.92      0.83        78

    accuracy                           0.81       152
   macro avg       0.83      0.81      0.81       152
weighted avg       0.82      0.81      0.81       152
```

Figure 50: The classification report

From the snapshot, one can deduce that class 0 (0.89 compared to 0.76) has a higher precision indicating that the model is more accurate in predicting class 0 instances. Class 1 has a higher recall (0.92 compared to 0.69) indicating that the model is better at capturing actual positive instances in class 1.

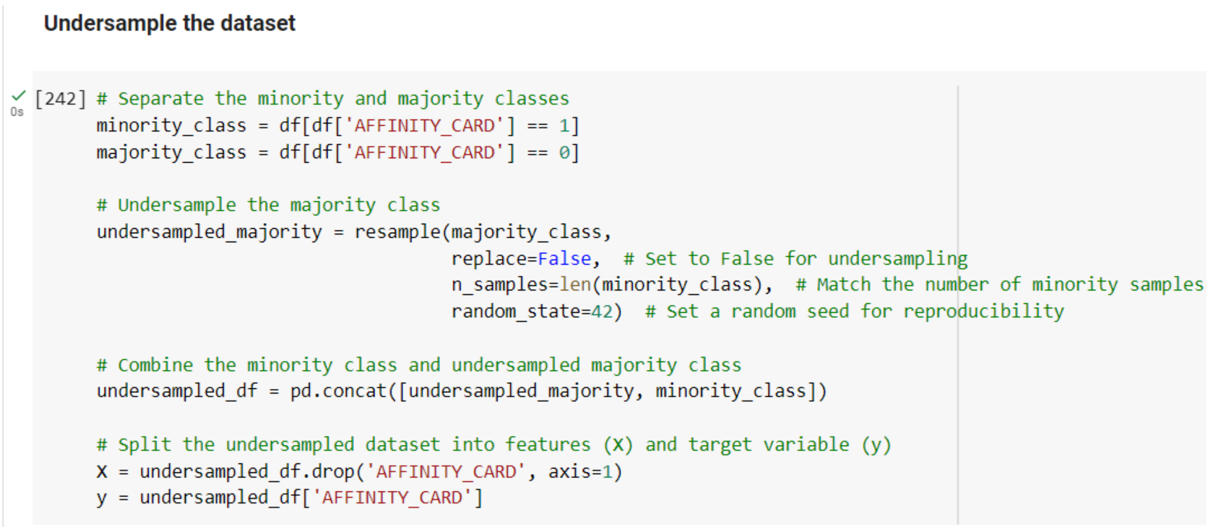
5.3 Model 2: Random Forest Classifier



Load Libraries

```
[81] import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, roc_curve
     from imblearn.over_sampling import RandomOverSampler
     import matplotlib.pyplot as plt
     from sklearn.metrics import confusion_matrix
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import classification_report
     import seaborn as sns
     from sklearn.tree import plot_tree
     df = pd.read_csv('final_data.csv')
```

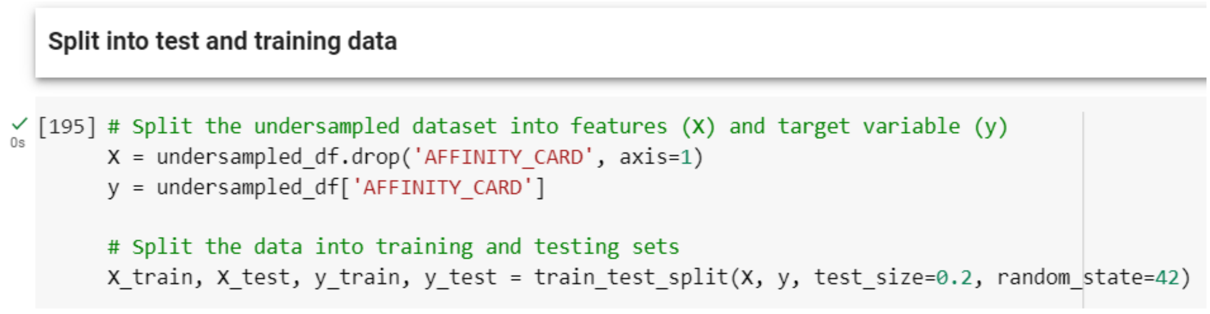
Figure 51: Loading the Libraries and dataset into the data frame



Undersample the dataset

```
[242] # Separate the minority and majority classes
      minority_class = df[df['AFFINITY_CARD'] == 1]
      majority_class = df[df['AFFINITY_CARD'] == 0]

      # Undersample the majority class
      undersampled_majority = resample(majority_class,
                                       replace=False,  # Set to False for undersampling
                                       n_samples=len(minority_class),  # Match the number of minority samples
                                       random_state=42)  # Set a random seed for reproducibility

      # Combine the minority class and undersampled majority class
      undersampled_df = pd.concat([undersampled_majority, minority_class])

      # Split the undersampled dataset into features (X) and target variable (y)
      X = undersampled_df.drop('AFFINITY_CARD', axis=1)
      y = undersampled_df['AFFINITY_CARD']
```

Figure 52: Undersampling the data

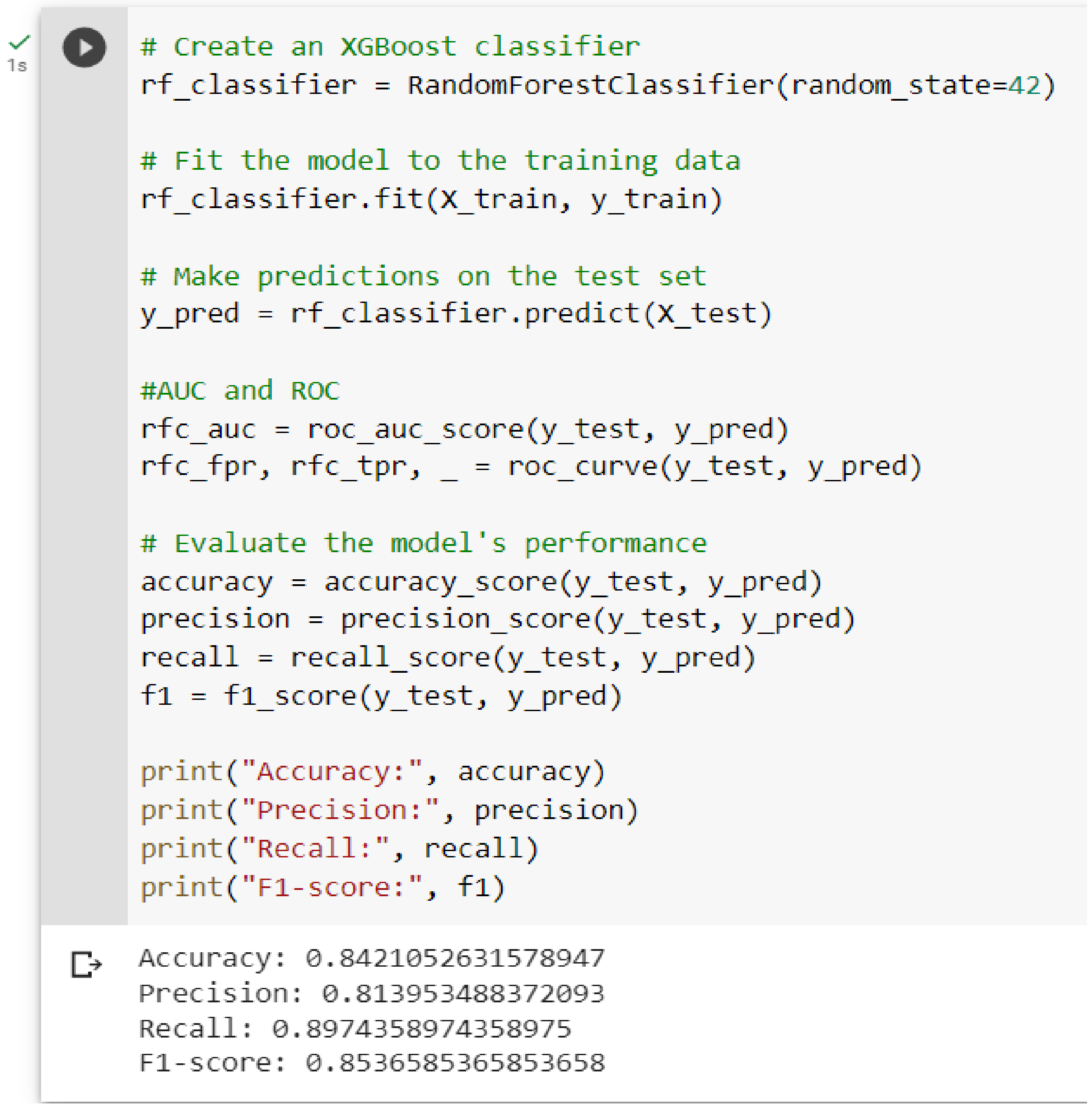


Split into test and training data

```
[195] # Split the undersampled dataset into features (X) and target variable (y)
      X = undersampled_df.drop('AFFINITY_CARD', axis=1)
      y = undersampled_df['AFFINITY_CARD']

      # 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)
```

Figure 53: Splitting the data into training and test data

Model and Predict using Random Forest Classifier



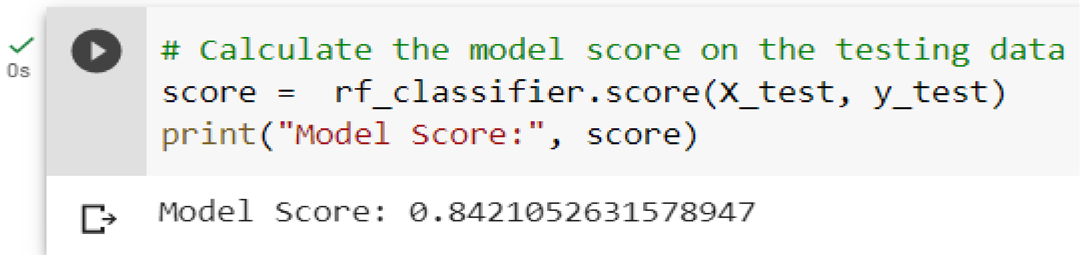
```
# Create an XGBoost classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Fit the model to the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

#AUC and ROC
rfc_auc = roc_auc_score(y_test, y_pred)
rfc_fpr, rfc_tpr, _ = roc_curve(y_test, y_pred)

# Evaluate the model's performance
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)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Accuracy: 0.8421052631578947
Precision: 0.813953488372093
Recall: 0.8974358974358975
F1-score: 0.8536585365853658
```

Figure 54: Model and predict using Random Forest Classifier

The accuracy is high at 0.842. The precision has a score of 0.814, a recall of 0.897, and an F1 score of 0.854.

Model Score

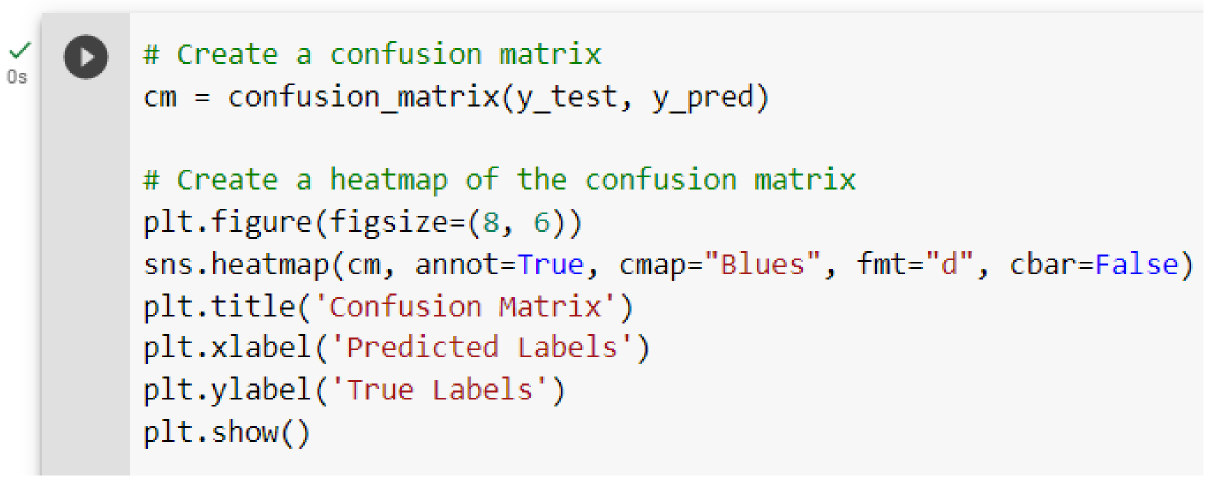


```
# Calculate the model score on the testing data
score =  rf_classifier.score(X_test, y_test)
print("Model Score:", score)
```

```
Model Score: 0.8421052631578947
```

Figure 55: The mode score of the Random Forest Classifier

Confusion Matrix



```
# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Figure 56: Code for the confusion matrix

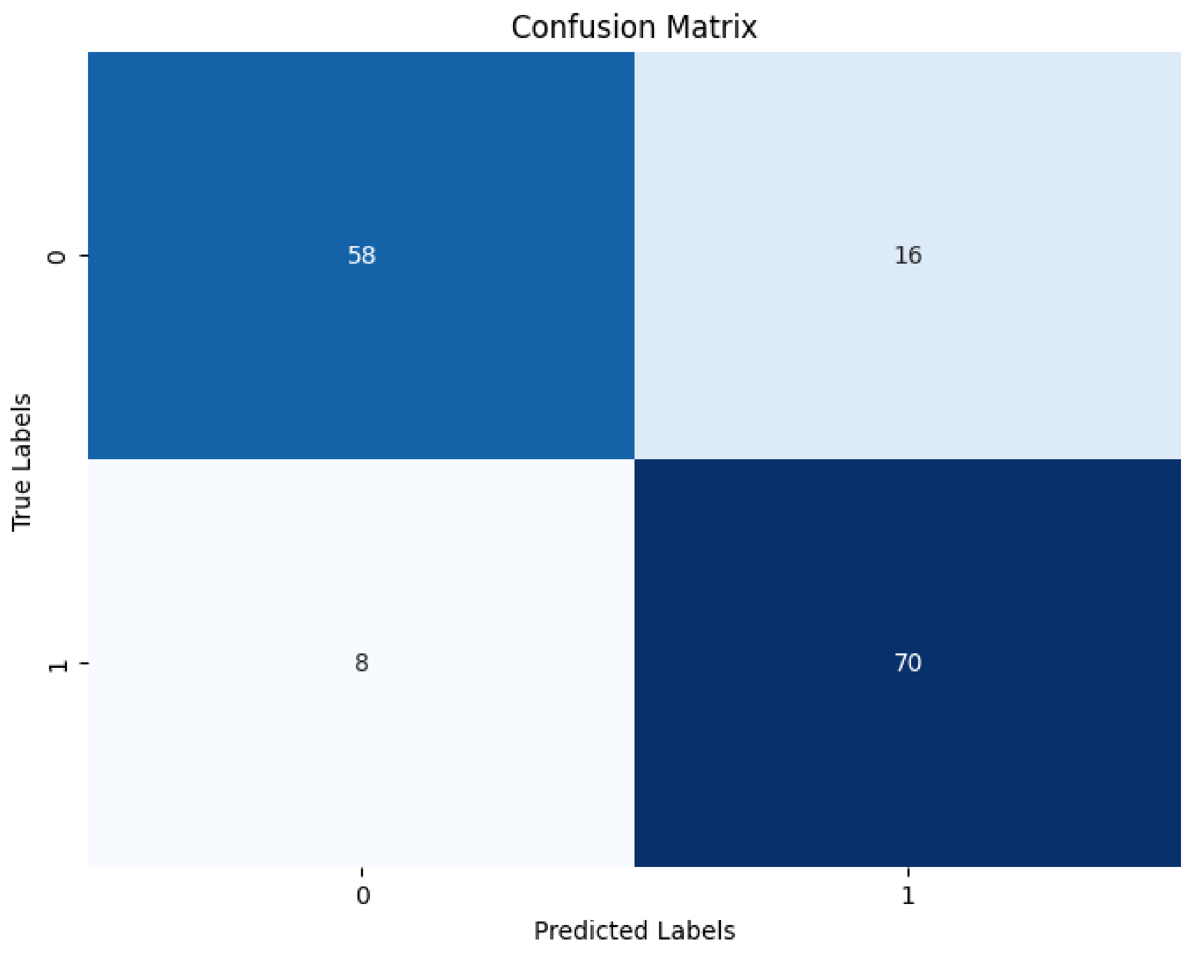
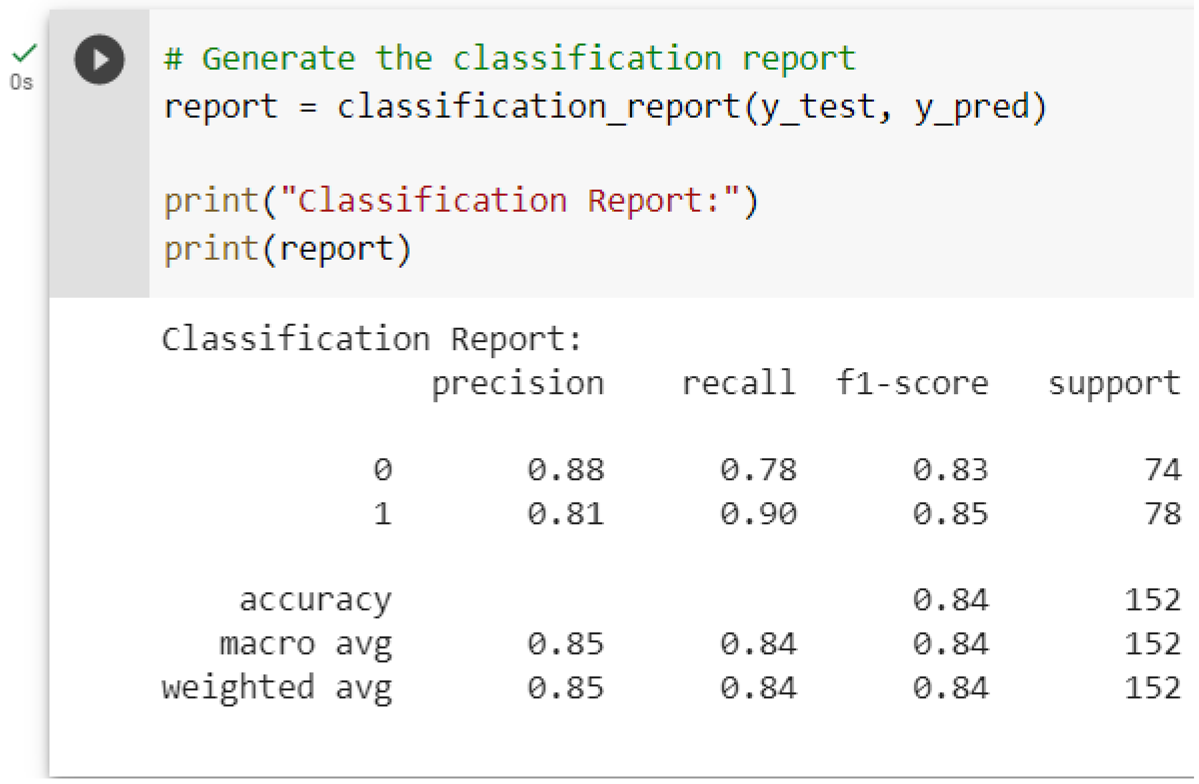


Figure 56: The confusion matrix of the Random Forest Classifier.

Classification Report



```
# Generate the classification report
report = classification_report(y_test, y_pred)

print("Classification Report:")
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.78      0.83        74
           1       0.81      0.90      0.85        78

    accuracy                           0.84       152
   macro avg       0.85      0.84      0.84       152
weighted avg       0.85      0.84      0.84       152
```

Figure 57: The classification report for Random Forest Classifier

From the above, one can deduce that class 0 (0.88 compared to 0.81) has a higher precision than class 1 indicating that the model is more accurate in predicting class 0 instances. Class 1 has a higher recall (0.90 compared to 0.78) indicating that the model is better at capturing actual positive instances in class 1.

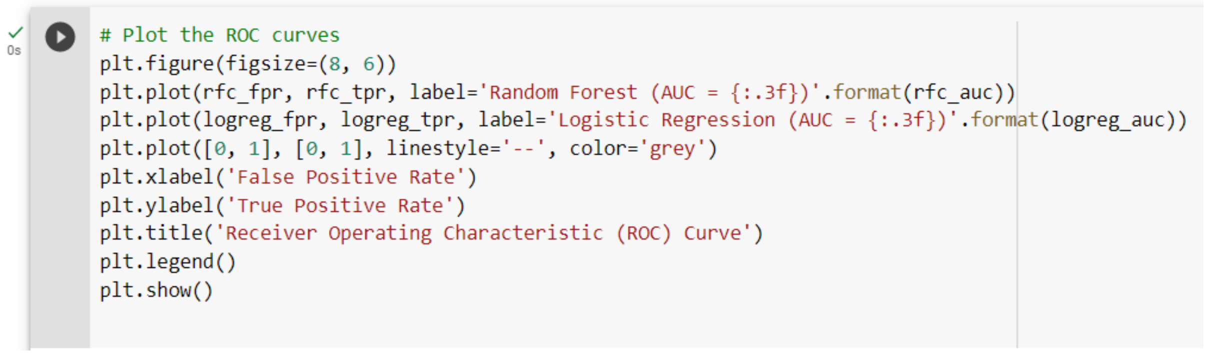
5.4 Comparison of Models

There is a need to assess the performance of the models against each other. This is done using the Receiver Operating Characteristic Curve (ROC). It illustrates the performance of a binary classifier system. It is used to select the model with the best performance.

5.4.1 ROC Curves

Receiver Operating Characteristic Curve plots the true positive rate (TPR) against the false positive rate (FPR) for various thresholds. The ROC curve can be used to calculate the area under the curve (AUC). The AUC measures the overall performance of a model. AUC values range from 0.5 to 1 where 1 is perfect and 0.5 is random guessing. The higher the Area Under the Curve, the better its performance. The code to plot the ROC Curve is shown below

ROC Curve



```
# Plot the ROC curves
plt.figure(figsize=(8, 6))
plt.plot(rfc_fpr, rfc_tpr, label='Random Forest (AUC = {:.3f})'.format(rfc_auc))
plt.plot(logreg_fpr, logreg_tpr, label='Logistic Regression (AUC = {:.3f})'.format(logreg_auc))
plt.plot([0, 1], [0, 1], linestyle='--', color='grey')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

Figure 58: ROC Curve Code

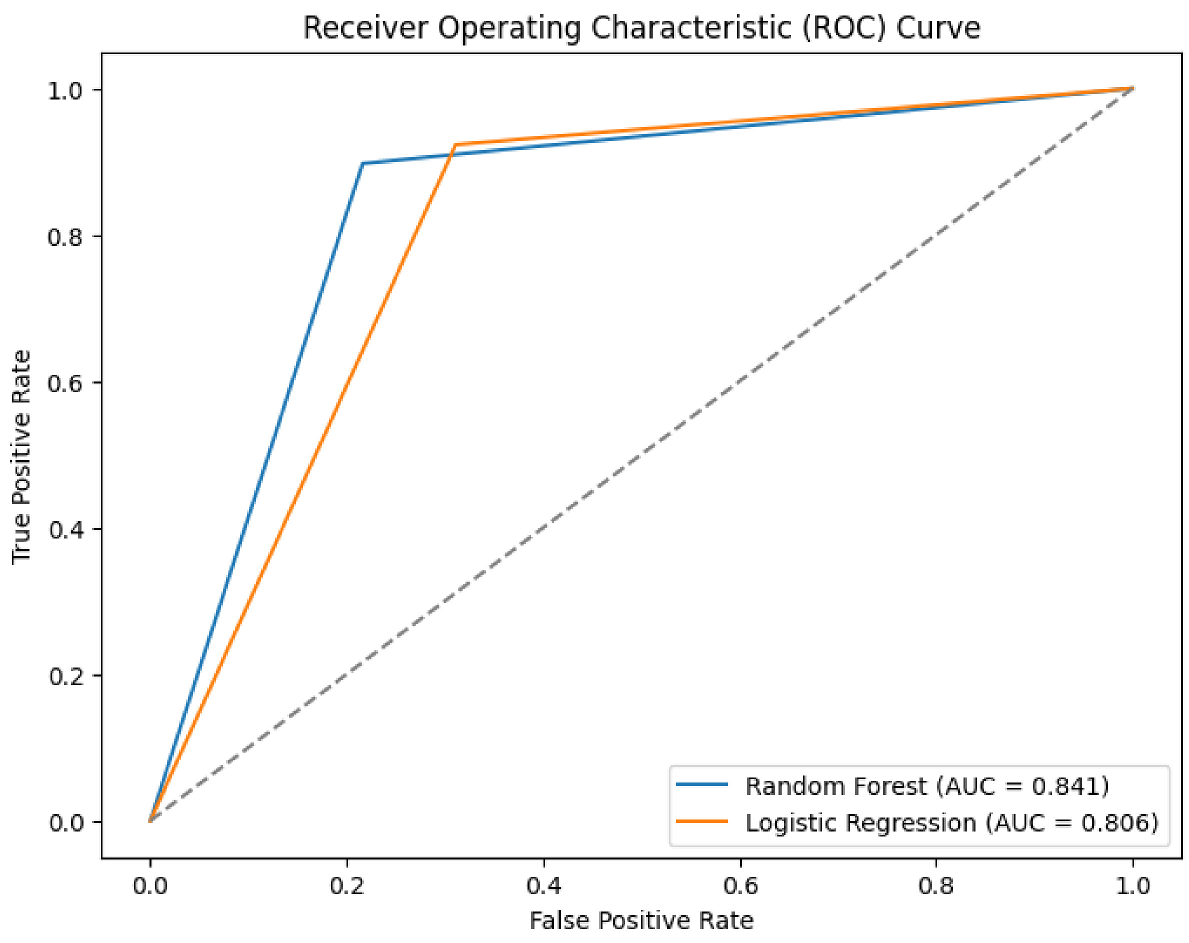


Figure 59: Receiver Operating Characteristic (ROC) Curve

From the above, The AUC of Random Forest Classifier is 0.841 and it shows that it has a better performance than Logistic Regression with an AUC of 0.806. It is safe to conclude that Random Forest Classifier has a higher ability to correctly classify positive and negative instances than Logistic Regression.

6 Discussion and Reflection

The project follows the CRISP-DM Methodology as it starts with Business Understanding before it talks about data understanding where there was an in-depth understanding of the data to understand the data type, data count, and column name. In this stage, there was a check for missing and error data in the dataset.

The next stage which is Data preparation involved removing and correcting missing and error data. Some of the variables removed are This stage takes about 60-70% of the total time to implement the CRISP-DM methodology. In this stage, some uncorrelated variables are removed. These are variables with very little or no influence on the target variable. Also, transforming categorical variables into ordinal numbers happens at this stage. This is to make the data suitable for prediction by machine learning models.

Data Analysis is where the summary statistics like Skewness and Kurtosis of all variables happen. It shows a correlation. Data exploration involves how each variable affects another or itself.

Data Mining is where predictions are done with machine learning models. In this stage, One-hot encoding is done to convert categorical information into numerical variables. The Logistic Regression and Random Forest Classifier models are used here for prediction. The dataset is under-sampled and then split into training and test data before predictions are made. This is to remove the bias by the majority class (in this case 0) by removing values from the 0s to balance it with the 1s.

The Random Forest Classifier (RFC) has a precision of 0.88 predicting class 0 instances than class 1 with a precision of 0.81 indicating that the classifier had a relatively lower rate of false positives for class 0. This means that class 0 has a better rate of accuracy for positive predictions. RF Classifier also has a recall of 0.78 for class 0 and 0.90 for class 1 indicating that the classifier had a relatively low rate of false negatives for class 1. This means that class 1 can correctly identify actual positive instances better. The accuracy of the model is 0.84.

The Logistic Regression model has a higher precision for class 0 at 0.89 than class 1 at 0.76 which shows that the classifier has a relatively lower rate of false positives. It also has a recall of 0.69 for class 0 and 0.92 for class 1 indicating that the classifier has a relatively low rate of false negatives for class 1. This implies that class 1 can correctly identify actual positive instances better. The accuracy of the model is 0.81.

Overall, the Random Forest Classifier is a better model than the Logistic Regression. The Random Forest Classifier has higher precision, recall, and f1-score and a better accuracy based on the classification reports.

In conclusion, Random Forest Classifier is a better model for predicting the target variable which is AFFINITY_CARD. This means that we should expect an 84% accuracy when predicting AFFINITY_CARD.

7 References

[1] H. McCord, “Thoughts on Government Terminology,” Unmarried Equality, August 2009. [Online]. Available: https://www.unmarried.org/government-terminology/. [Accessed 04 May 2023].