CC7182 PROGRAMMING FOR DATA ANALYTICS

COURSEWORK REPORT 2023-2024

MAY 10, 2024

SUBMITTED BY: MARYAM KHATTAK
ID: 22083966

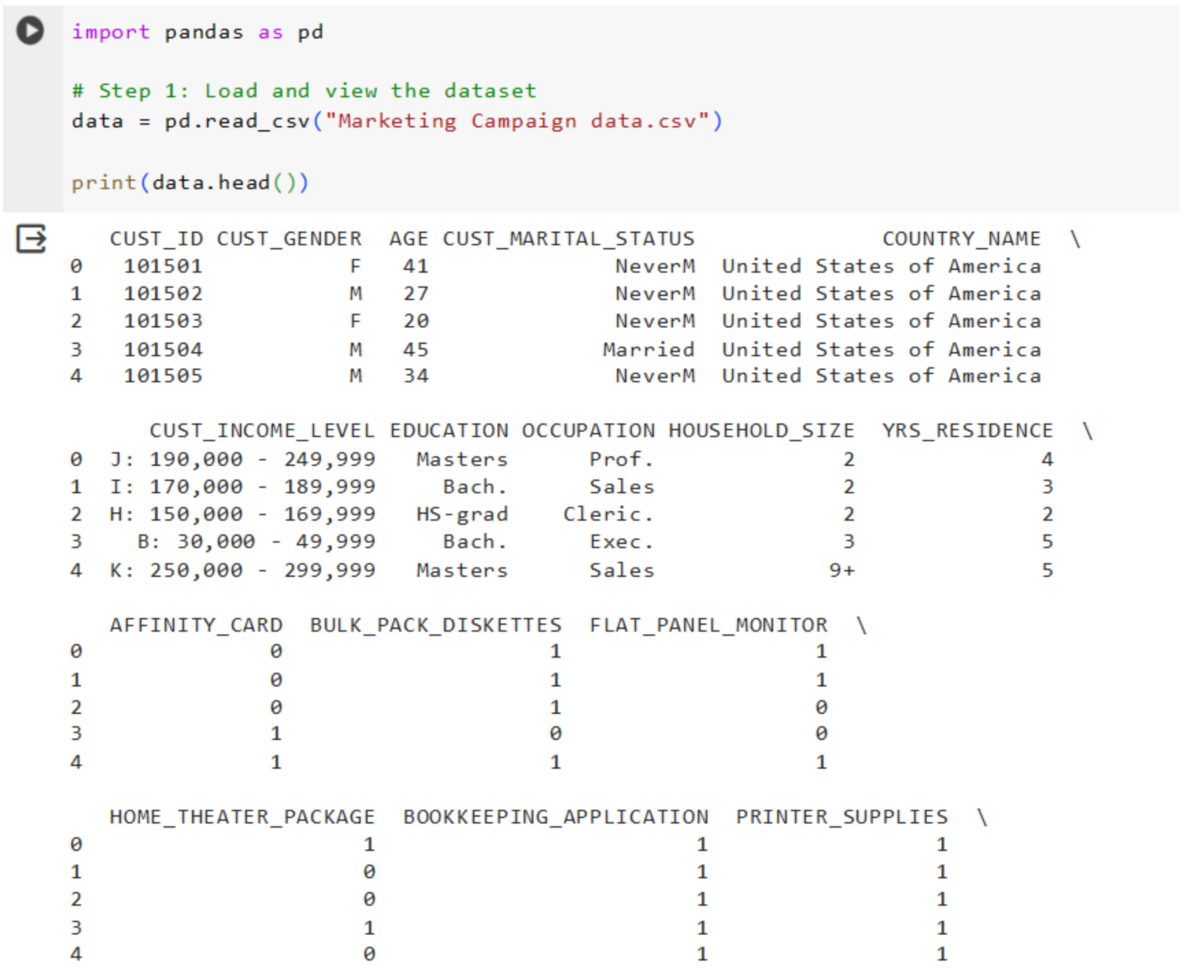
Table of Contents

1. Data Understanding

The dataset for this study project is a marketing campaign dataset from a marketing company, which contains 1500 customer records with 19 variables. These variables include sociodemographic data and product ownership data. The main objective is to prepare the data for further data mining and analysis.

1.1 Loading Dataset

The data was initially imported from the "Marketing Campaign data.csv" file using the Python programming language and the pandas library. To ensure the integrity of the data and to gain an initial understanding of its structure, the first few rows of the dataset were examined using the data.head() function. Below Figure 1 shows the data loading steps and the first few rows of the dataset.



```python
import pandas as pd

# Step 1: Load and view the dataset
data = pd.read_csv("Marketing Campaign data.csv")

print(data.head())
```

```
   CUST_ID CUST_GENDER  AGE CUST_MARITAL_STATUS              COUNTRY_NAME  \
0   101501           F   41              NeverM  United States of America
1   101502           M   27              NeverM  United States of America
2   101503           F   20              NeverM  United States of America
3   101504           M   45             Married  United States of America
4   101505           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  BULK_PACK_DISKETTES  FLAT_PANEL_MONITOR  \
0              0                    1                   1
1              0                    1                   1
2              0                    1                   0
3              1                    0                   0
4              1                    1                   1

   HOME_THEATER_PACKAGE  BOOKKEEPING_APPLICATION  PRINTER_SUPPLIES  \
0                     1                        1                 1
1                     0                        1                 1
2                     0                        1                 1
3                     1                        1                 1
4                     0                        1                 1
```

Figure 1: Data Loading and Initial Inspection

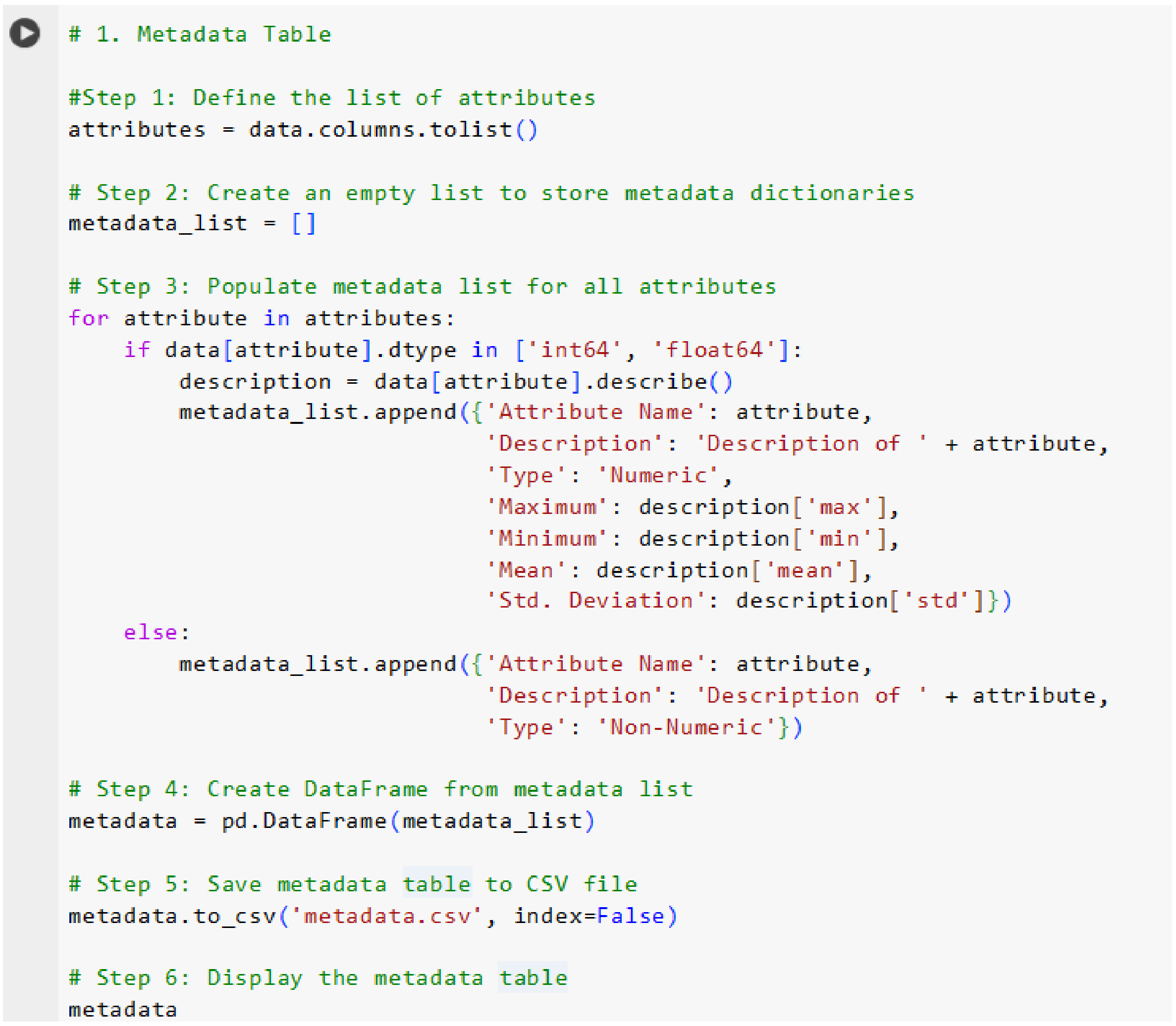
This step sets the foundation for a thorough data understanding and subsequent preprocessing tasks.

1.2 Metadata, Histogram, Mode & Bar Chart

In this section, we analyze the metadata of the dataset to better understand the attributes. Use histograms to visualize their distribution. For nominal attributes, we calculate the mode and draw a bar chart to show the frequency distribution of categorical values. This review informs our decisions on data cleaning, transformation, and sampling strategies.

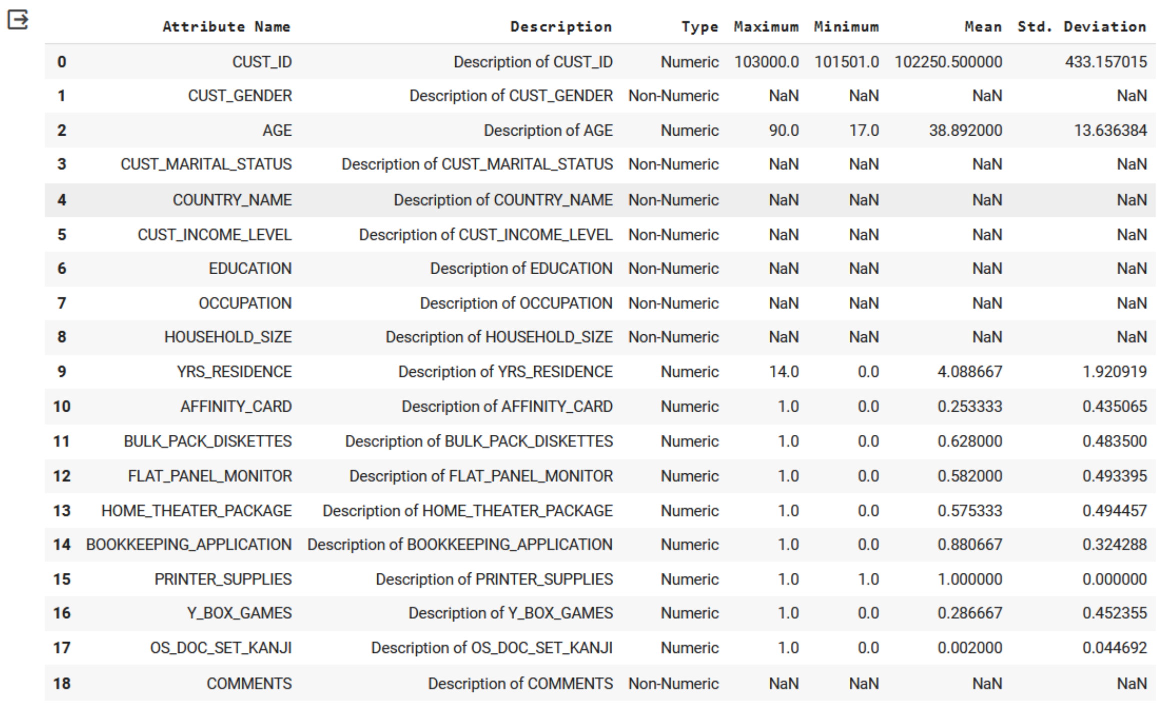
1.2.1 Metadata Table

To understand the dataset, we created a meta data table showing the characteristics of each attribute. This table contains the attribute name, description, maximum, minimum, median, and range for numerical data. The steps taken to present this meta data table are shown in Figure 2. Figure 3 shows the resulting meta data table.



```
# 1. Metadata Table

#Step 1: Define the list of attributes
attributes = data.columns.tolist()

# Step 2: Create an empty list to store metadata dictionaries
metadata_list = []

# Step 3: Populate metadata list for all attributes
for attribute in attributes:
    if data[attribute].dtype in ['int64', 'float64']:
        description = data[attribute].describe()
        metadata_list.append({'Attribute Name': attribute,
                              'Description': 'Description of ' + attribute,
                              'Type': 'Numeric',
                              'Maximum': description['max'],
                              'Minimum': description['min'],
                              'Mean': description['mean'],
                              'Std. Deviation': description['std']})
    else:
        metadata_list.append({'Attribute Name': attribute,
                              'Description': 'Description of ' + attribute,
                              'Type': 'Non-Numeric'})

# Step 4: Create DataFrame from metadata list
metadata = pd.DataFrame(metadata_list)

# Step 5: Save metadata table to CSV file
metadata.to_csv('metadata.csv', index=False)

# Step 6: Display the metadata table
metadata
```

Figure 2: Steps to Display Meta Data Table



	Attribute Name	Description	Type	Maximum	Minimum	Mean	Std. Deviation
0	CUST_ID	Description of CUST_ID	Numeric	103000.0	101501.0	102250.500000	433.157015
1	CUST_GENDER	Description of CUST_GENDER	Non-Numeric	NaN	NaN	NaN	NaN
2	AGE	Description of AGE	Numeric	90.0	17.0	38.892000	13.636384
3	CUST_MARITAL_STATUS	Description of CUST_MARITAL_STATUS	Non-Numeric	NaN	NaN	NaN	NaN
4	COUNTRY_NAME	Description of COUNTRY_NAME	Non-Numeric	NaN	NaN	NaN	NaN
5	CUST_INCOME_LEVEL	Description of CUST_INCOME_LEVEL	Non-Numeric	NaN	NaN	NaN	NaN
6	EDUCATION	Description of EDUCATION	Non-Numeric	NaN	NaN	NaN	NaN
7	OCCUPATION	Description of OCCUPATION	Non-Numeric	NaN	NaN	NaN	NaN
8	HOUSEHOLD_SIZE	Description of HOUSEHOLD_SIZE	Non-Numeric	NaN	NaN	NaN	NaN
9	YRS_RESIDENCE	Description of YRS_RESIDENCE	Numeric	14.0	0.0	4.088667	1.920919
10	AFFINITY_CARD	Description of AFFINITY_CARD	Numeric	1.0	0.0	0.253333	0.435065
11	BULK_PACK_DISKETTES	Description of BULK_PACK_DISKETTES	Numeric	1.0	0.0	0.628000	0.483500
12	FLAT_PANEL_MONITOR	Description of FLAT_PANEL_MONITOR	Numeric	1.0	0.0	0.582000	0.493395
13	HOME_THEATER_PACKAGE	Description of HOME_THEATER_PACKAGE	Numeric	1.0	0.0	0.575333	0.494457
14	BOOKKEEPING_APPLICATION	Description of BOOKKEEPING_APPLICATION	Numeric	1.0	0.0	0.880667	0.324288
15	PRINTER_SUPPLIES	Description of PRINTER_SUPPLIES	Numeric	1.0	1.0	1.000000	0.000000
16	Y_BOX_GAMES	Description of Y_BOX_GAMES	Numeric	1.0	0.0	0.286667	0.452355
17	OS_DOC_SET_KANJI	Description of OS_DOC_SET_KANJI	Numeric	1.0	0.0	0.002000	0.044692
18	COMMENTS	Description of COMMENTS	Non-Numeric	NaN	NaN	NaN	NaN

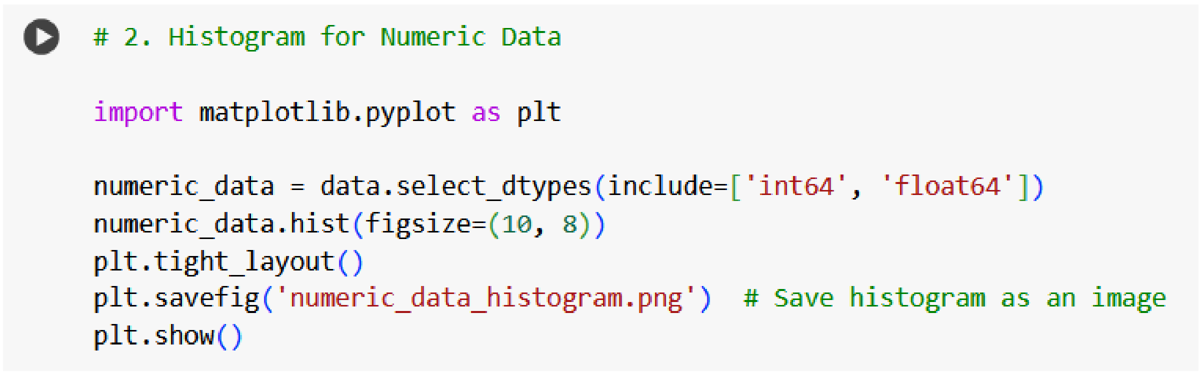
Figure 3: Meta Data Table

The metadata table provides a detailed overview of each attribute in the dataset, along with basic attributes such as attribute names, descriptions, data types, and statistical measures used to generate statistical data. For statistical attributes, including AGE, YRS_RESIDENCE, and AFFINITY_CARD, descriptive statistics such as maximum, minimum, median, and standard deviation provide insight into the central characteristics and variability of the data distribution. Non-numeric attributes, such as CUST_GENDER and CUST_MARITAL_STATUS, are defined as 'non-numeric' without numeric parameters, because they do not have numeric values. This table is a valuable resource for understanding the structure of the dataset and provides important information for subsequent analysis and modeling tasks. As we move deeper into data analysis and preparation, this metadata table serves as a model for informed and insightful data-driven decision making.

1.2.2 Histogram

Histograms allow us to see the distribution of numerical data, allowing us to understand the frequencies and values of each variable. In the code snippet provided, matplotlib is used to generate histograms for statistical data types.

To plot the histograms, we first select only numeric data from the dataset using the select_dtypes method in pandas and specify the data types as 'int64' and 'float64'. Next, we use the hist function to create histograms for each numeric variable. The figsize parameter determines the size of the resulting figures, ensuring clarity and readability. Finally, we fix the layout of the plot with tight_layout to prevent overlapping of axes labels and titles. Figure 4 shows the steps taken to plot the histograms, while Figure 5 shows the histograms of all statistical variables.



```
# 2. Histogram for Numeric Data

import matplotlib.pyplot as plt

numeric_data = data.select_dtypes(include=['int64', 'float64'])
numeric_data.hist(figsize=(10, 8))
plt.tight_layout()
plt.savefig('numeric_data_histogram.png')  # Save histogram as an image
plt.show()
```

Figure 4: Steps to Plot Histograms for Numeric Data

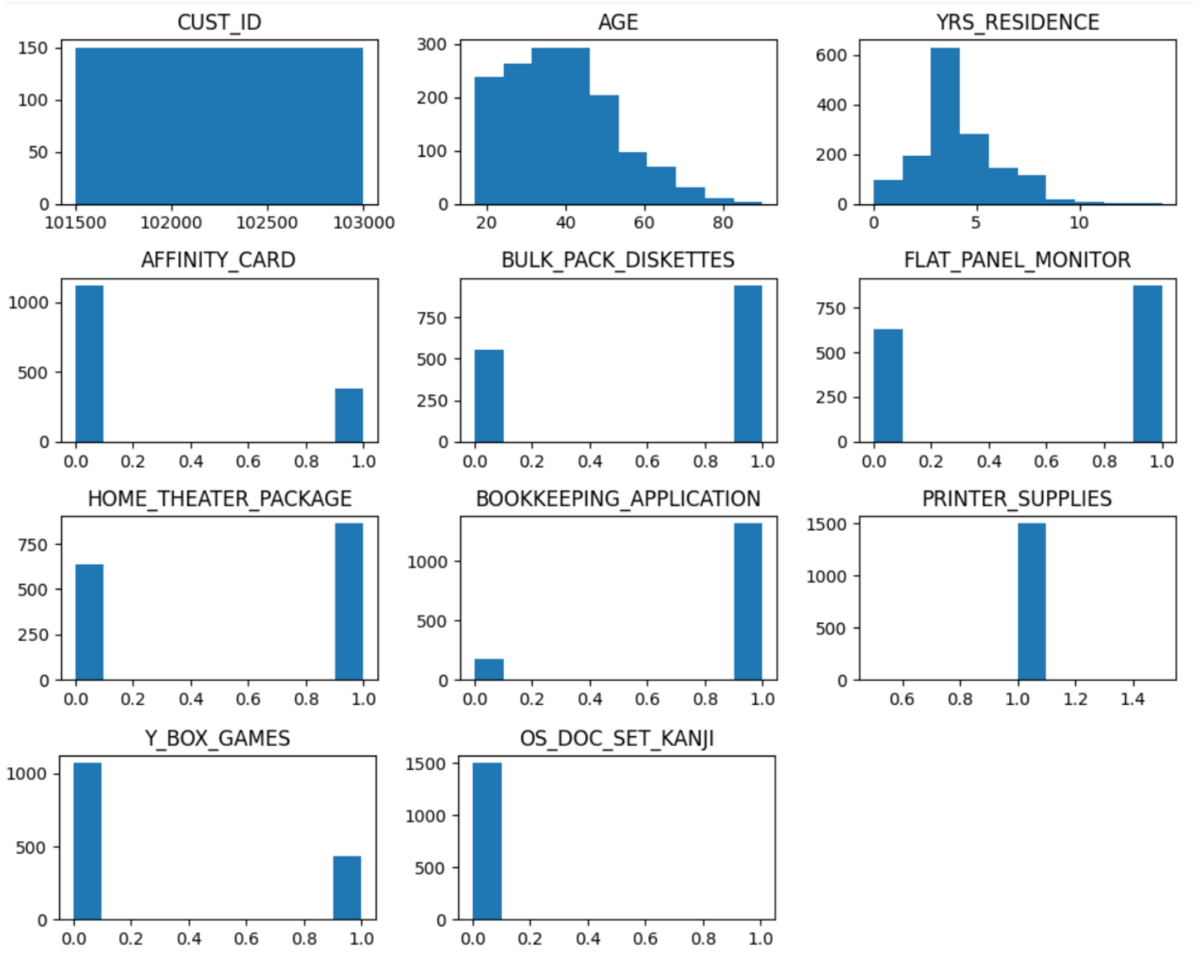


Figure 5: Histograms of Numeric Variables

This visualization step helps to understand the distribution of values in each statistical variable, and provides insight into the characteristics of the mean, spread, and possible outliers in the dataset.

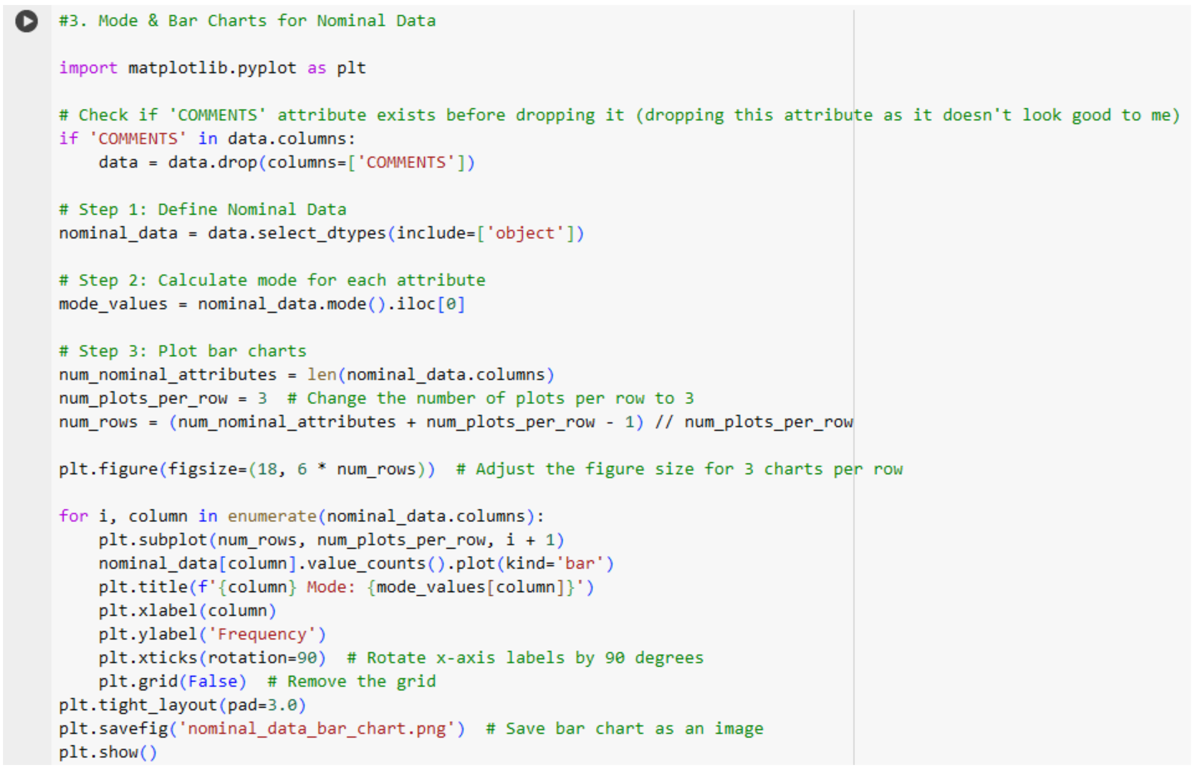
1.2.3 Mode & Bar Chart

For nominal data, we added mode values and drew bar charts to visualize frequency distributions. In this exercise we perform the following steps:

1. **Check and Drop 'COMMENTS' Attribute:** We first check if the 'COMMENTS' attribute exists in the dataset. In that case, we discard it because it contains text data that may not be suitable for bar chart visualization.
2. **Define Nominal Data:** We select only the columns of object data type, which represent nominal variables.
3. **Calculate Mode:** We calculate the mode for each nominal attribute using the mode() function, and then we extract the mode values.
4. **Plot Bar Charts:** We create bar charts for each attribute called. For visualization purposes, we change the number of plots per row to 3 and resize the figures accordingly. Each bar chart represents the frequency of categories in each trait. The title of each chart includes the attribute name and its mode value.

Figures:

Figure 6 shows the steps for calculating the mode and generating the bar charts, while Figure 7 shows the resulting bar charts with the mode values.



```python
#3. Mode & Bar Charts for Nominal Data

import matplotlib.pyplot as plt

# Check if 'COMMENTS' attribute exists before dropping it (dropping this attribute as it doesn't look good to me)
if 'COMMENTS' in data.columns:
    data = data.drop(columns=['COMMENTS'])

# Step 1: Define Nominal Data
nominal_data = data.select_dtypes(include=['object'])

# Step 2: Calculate mode for each attribute
mode_values = nominal_data.mode().iloc[0]

# Step 3: Plot bar charts
num_nominal_attributes = len(nominal_data.columns)
num_plots_per_row = 3  # Change the number of plots per row to 3
num_rows = (num_nominal_attributes + num_plots_per_row - 1) // num_plots_per_row

plt.figure(figsize=(18, 6 * num_rows))  # Adjust the figure size for 3 charts per row

for i, column in enumerate(nominal_data.columns):
    plt.subplot(num_rows, num_plots_per_row, i + 1)
    nominal_data[column].value_counts().plot(kind='bar')
    plt.title(f'{column} Mode: {mode_values[column]}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.xticks(rotation=90)  # Rotate x-axis labels by 90 degrees
    plt.grid(False)  # Remove the grid
plt.tight_layout(pad=3.0)
plt.savefig('nominal_data_bar_chart.png')  # Save bar chart as an image
plt.show()
```

Figure 6: Steps to Create Bar Charts for Nominal Data

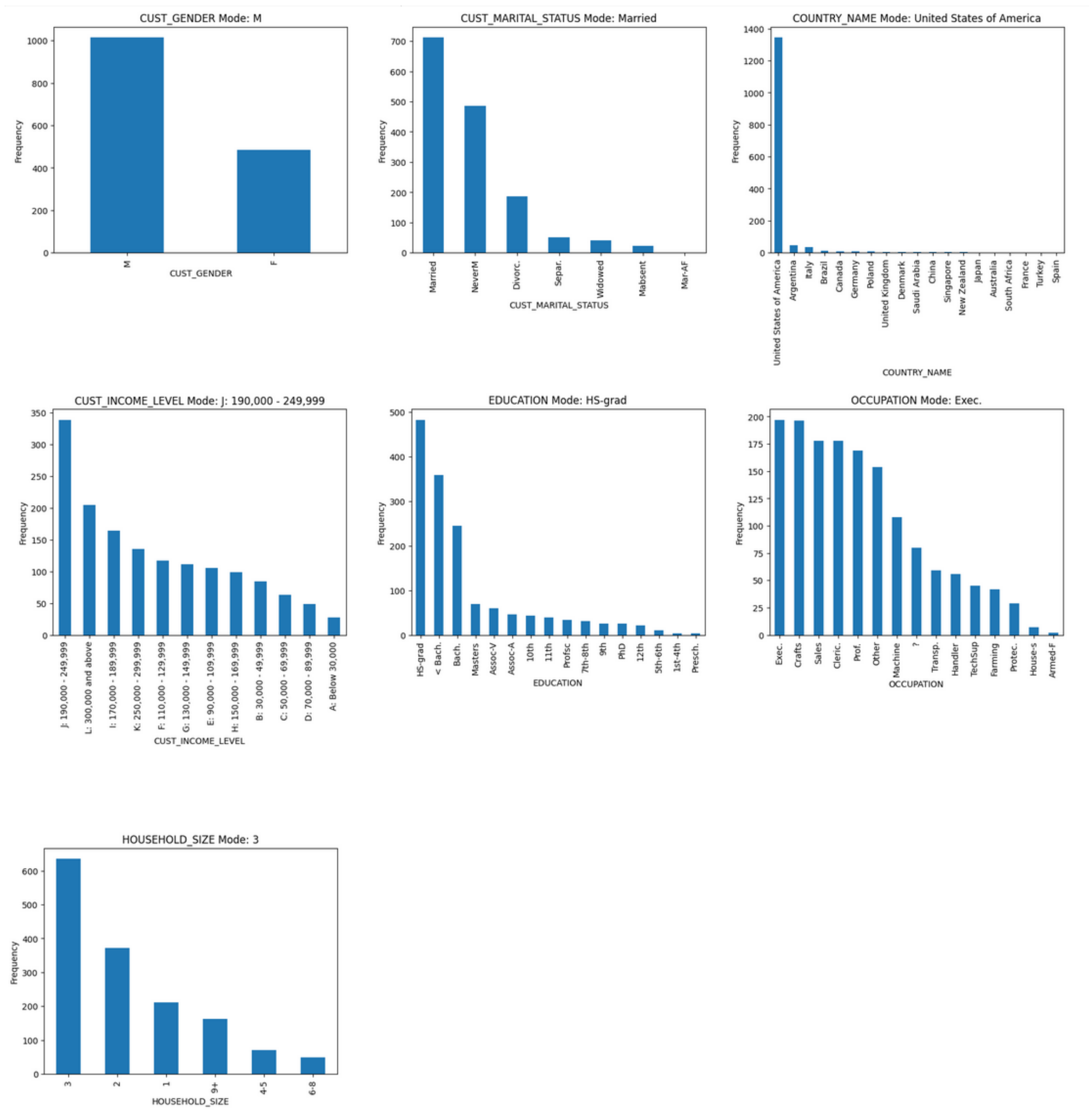


Figure 7: Bar Charts of Nominal Variables with Mode Values

This diagram provides insight into the frequency distribution of categorical variables, with the mode value indicating the most frequent category of each attribute. Omitting the 'COMMENTS' attribute improves the presentation of the data, as it contains long text data that is not suitable for bar chart visualization. Overall, this process enhances our understanding of the categorical variables of the dataset and their distribution.

1.3 Missing or Error Data Handling

In this section, we defined missing or error data for each attribute, including all missing data types such as null, blank, unknown, and any invalid or inconsistent data. We provide insight into the frequency of data loss and error for each attribute, offering suggestions for handling strategies without performing data cleansing at this point.

Process Overview:

1. **Load Data:** The dataset "Marketing Campaign data.csv" is loaded into the environment using pandas.
2. **Create DataFrame with Missing and Error Data Information:** We create a DataFrame named missing_error_df to collect information about missing and error data for each attribute. For missing data, we count null values, blank values, and 'unknown' entries. When we want error data, we find incorrect or inconsistent patterns, such as '?' or date formats, depending on the data type of each attribute.
3. **Display DataFrame:** The resulting DataFrame provides a complete description of missing data and errors, and records the attributes along with missing statistics and error occurrences.

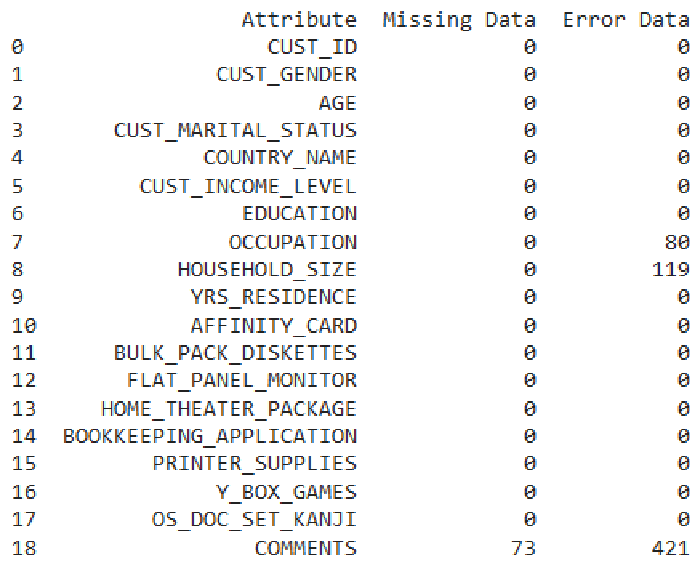
Missing Data and Error Data:

- **Missing Data:** For attributes such as 'OCCUPATION' and 'HOUSEHOLD_SIZE' there is missing information. For example, 'OCCUPATION' has 80 missing values, while 'HOUSEHOLD_SIZE' has 119 missing values.
- **Error Data:** Attributes such as 'COMMENTS' display error data, including 73 missing values and 421 typos of the expected data structure.

Handling Methods Suggestions:

1. **Missing Data Handling:** For the nature of missing data, methods such as imputation of mean, median, or mode values can be used. Alternatively, records with missing values can be removed, if they do not significantly affect the integrity of the dataset.
2. **Error Data Handling:** For various characteristics of error data, data validation techniques or pattern matching algorithms can be used to detect and correct incorrect information. Additionally, manual analysis or domain expertise can help resolve discrepancies.

The output showcasing missing and error data information is presented in Figure 8. For detailed coding steps, please refer to the appendix section provided in the coding link.



```
                 Attribute  Missing Data  Error Data
0                  CUST_ID             0           0
1              CUST_GENDER             0           0
2                      AGE             0           0
3      CUST_MARITAL_STATUS             0           0
4             COUNTRY_NAME             0           0
5        CUST_INCOME_LEVEL             0           0
6                EDUCATION             0           0
7               OCCUPATION             0          80
8           HOUSEHOLD_SIZE             0         119
9            YRS_RESIDENCE             0           0
10           AFFINITY_CARD             0           0
11      BULK_PACK_DISKETTES            0           0
12        FLAT_PANEL_MONITOR           0           0
13     HOME_THEATER_PACKAGE            0           0
14  BOOKKEEPING_APPLICATION            0           0
15         PRINTER_SUPPLIES            0           0
16              Y_BOX_GAMES            0           0
17          OS_DOC_SET_KANJI           0           0
18                 COMMENTS           73         421
```

Figure 8: Missing and Error Data Information

This initial analysis lays the foundation for subsequent data cleaning and preparation steps, ensuring data quality and reliability for downstream research tasks.

2. Data Preparation

In the data preparation phase, we optimize the dataset to increase its suitability for subsequent analysis. This requires several important steps aimed at improving data quality, reducing redundancy, and ensuring consistency with analytical methods. Through Python frameworks, we systematically address issues such as variable reduction with minimal impact on the target variable "AFFINITY_CARD', data cleaning, and transformation, laying the foundation for robust and meaningful data analysis.

2.1 Reduce Variable

In this section, we focus on variables that exhibit minimal impact on the target variable, 'AFFINITY_CARD'. By detecting and removing such variations, we simplify the dataset, and optimize it for further analysis.

Steps Taken to Reduce Variables:

1. **Identify Correlation of Attributes with Target Variable AFFINITY_CARD:** We compute the correlation between each attribute and the objective variable to assess its predictive power.
2. **Reduce Variable with Lowest Attribute:** Based on thresholds, attributes with ZERO correlation are filtered out.
3. **Drop COMMENT Attribute:** The 'COMMENTS' attribute has been omitted due to its text data nature, requiring special text mining techniques.

Coding Details:

We perform these steps using Python and pandas:

1. Load the dataset and segregate numerical and categorical columns.
2. Calculate correlation coefficients between numerical variables and the target variable using Pearson's correlation coefficient.
3. Compute chi-square scores between categorical variables and the target variable.
4. Construct DataFrames to store correlation coefficients and chi-square scores.
5. Display the results to assess the correlation and chi-square scores.
6. Discard the 'COMMENTS' attribute from the attribute table, as it contains text data that requires special comment mining techniques.

Figures:

The added figures provide essential insights into attribute relationships with the target variable 'AFFINITY_CARD.' Figure 9, the correlation matrix, visually illustrates the strength and direction of correlations between numeric attributes and the target. Figure 10 offers a detailed table presenting correlation coefficients for numeric attributes, aiding in feature selection. Figure 11 showcases chi-square scores for categorical attributes, indicating their association with the target. Lastly, Figure 12 displays the dataset post-attribute removal. These visualizations streamline analysis by focusing on pertinent attributes, optimizing computational efficiency and model performance.

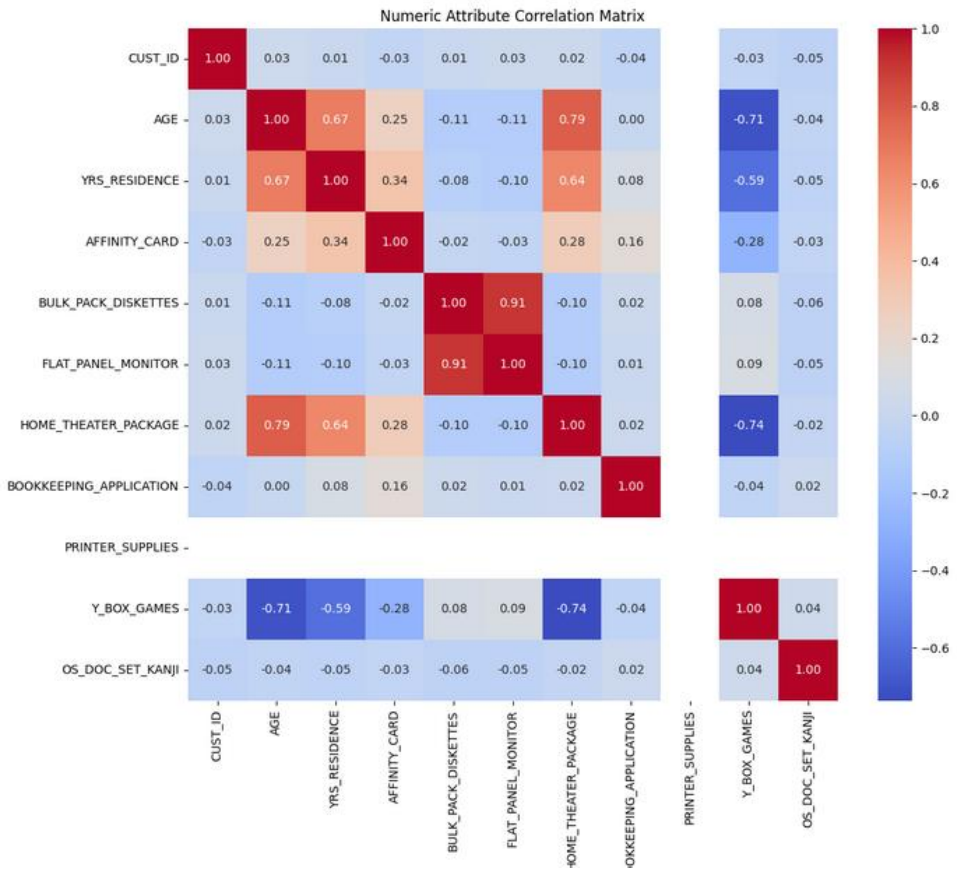
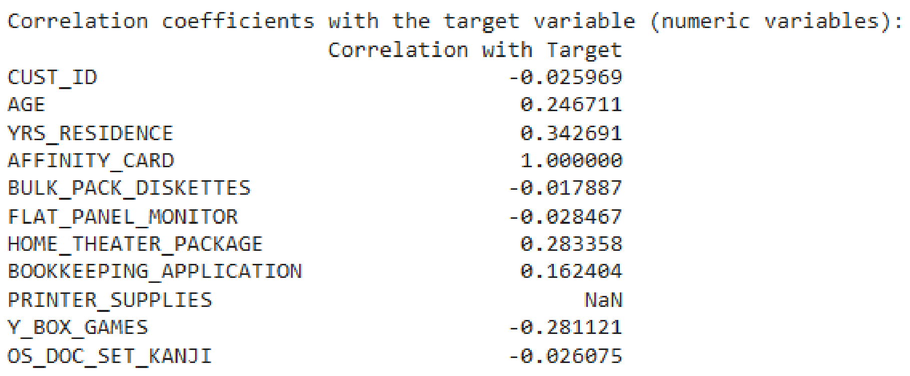
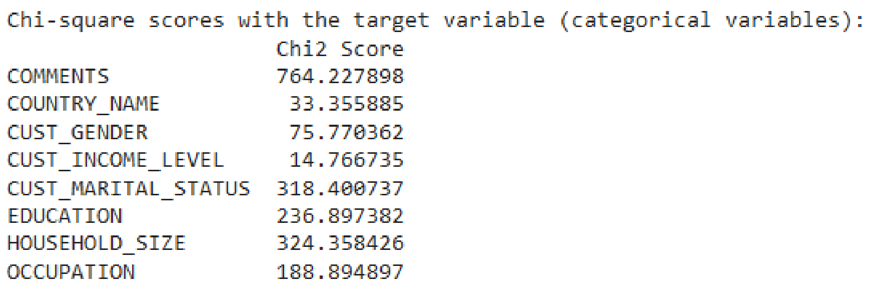


Figure 9: Numeric Attribute Correlation Matrix



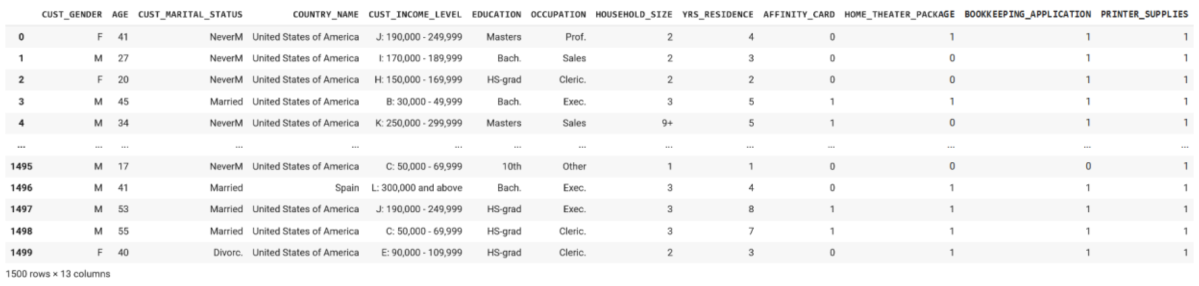
```
Correlation coefficients with the target variable (numeric variables):
                          Correlation with Target
CUST_ID                                 -0.025969
AGE                                      0.246711
YRS_RESIDENCE                            0.342691
AFFINITY_CARD                            1.000000
BULK_PACK_DISKETTES                     -0.017887
FLAT_PANEL_MONITOR                      -0.028467
HOME_THEATER_PACKAGE                     0.283358
BOOKKEEPING_APPLICATION                  0.162404
PRINTER_SUPPLIES                              NaN
Y_BOX_GAMES                             -0.281121
OS_DOC_SET_KANJI                        -0.026075
```

Figure 10: Numeric Attribute Correlation Coefficients



```
Chi-square scores with the target variable (categorical variables):
                      Chi2 Score
COMMENTS              764.227898
COUNTRY_NAME           33.355885
CUST_GENDER            75.770362
CUST_INCOME_LEVEL      14.766735
CUST_MARITAL_STATUS   318.400737
EDUCATION             236.897382
HOUSEHOLD_SIZE        324.358426
OCCUPATION            188.894897
```

Figure 11: Categorical Attribute Chi-square Scores



	CUST_GENDER	AGE	CUST_MARITAL_STATUS	COUNTRY_NAME	CUST_INCOME_LEVEL	EDUCATION	OCCUPATION	HOUSEHOLD_SIZE	YRS_RESIDENCE	AFFINITY_CARD	HOME_THEATER_PACKAGE	BOOKKEEPING_APPLICATION	PRINTER_SUPPLIES
0	F	41	NeverM	United States of America	J: 190,000 - 249,999	Masters	Prof.	2	4	0	1	1	1
1	M	27	NeverM	United States of America	I: 170,000 - 189,999	Bach.	Sales	2	3	0	0	1	1
2	F	20	NeverM	United States of America	H: 150,000 - 169,999	HS-grad	Cleric.	2	2	0	0	1	1
3	M	45	Married	United States of America	B: 30,000 - 49,999	Bach.	Exec.	3	5	1	1	1	1
4	M	34	NeverM	United States of America	K: 250,000 - 299,999	Masters	Sales	9+	5	1	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1495	M	17	NeverM	United States of America	C: 50,000 - 69,999	10th	Other	1	1	0	0	0	1
1496	M	41	Married	Spain	L: 300,000 and above	Bach.	Exec.	3	4	0	1	1	1
1497	M	53	Married	United States of America	J: 190,000 - 249,999	HS-grad	Exec.	3	8	1	1	1	1
1498	M	55	Married	United States of America	C: 50,000 - 69,999	HS-grad	Cleric.	3	7	1	1	1	1
1499	F	40	Divorc.	United States of America	E: 90,000 - 109,999	HS-grad	Cleric.	2	3	0	1	1	1

1500 rows × 13 columns

Figure 12: Attribute Table after Dropping 'COMMENTS'

Justification:

Through this approach, we optimize computational efficiency and model performance by retaining only pertinent variables for analysis. Upon conducting the analysis, the following attributes were identified to have ZERO influence on the target variable 'AFFINITY_CARD':

- CUST_ID
- BULK_PACK_DISKETTES
- FLAT_PANEL_MONITOR
- Y_BOX_GAMES
- OS_DOC_SET_KANJI

Considering their negligible impact on predicting the target variable, these attributes are dropped from the dataset to streamline further analysis. Additionally, COMMENTS is omitted due to its text data nature, requiring specialized text mining tools for meaningful analysis. This culling process ensures that our models focus on the most relevant features, thereby enhancing their predictive accuracy and interpretability.

2.2 Data Cleaning

Python programs were developed to clean the data through careful consideration and feedback, ensuring that only records with missing values or significant errors were removed, especially if they were less than 5% of the the entire dataset a. And below are the steps:

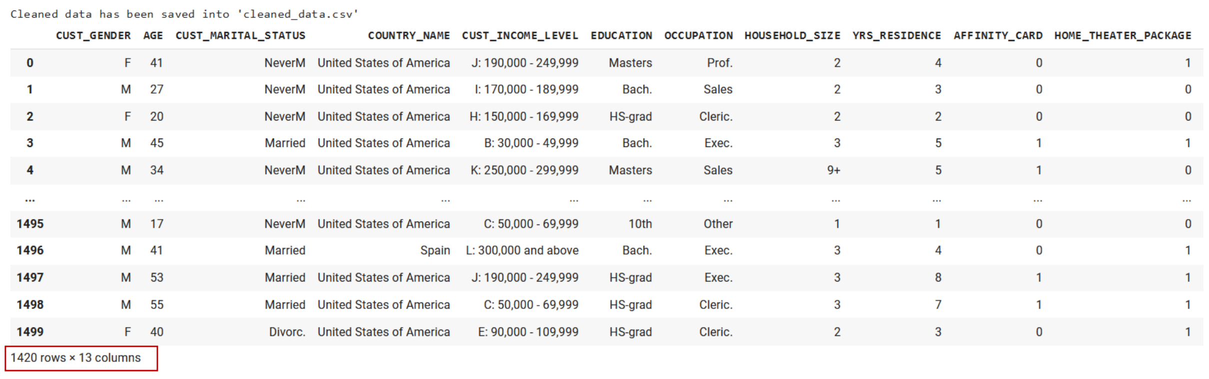
Steps Taken for Data Cleaning:

1. The dataset was loaded into a DataFrame for further processing.
2. The list of attributes is defined by characters in the dataset.
3. DataFrame was created to store information about missing data and errors for each attribute. This step involves calculating missing data and error estimates for each attribute. Iterate over attributes to consider missing data and errors. In this step, we calculate both missing data and error rates for each attribute, combining both types of anomalies. If all those counts exceed a predefined threshold (5% of total dataset size), records with missing or erroneous data are removed for the current attribute. This function requires removing rows with null attribute values or error-indicating examples.
4. The cleansed data were saved in a new CSV file named "Transformed_Data.csv" for further analysis.

Justification and Comments:

- We set a threshold of 5% to exclude records with missing values or errors. This threshold ensures that only records with a certain amount of missing or erroneous data are removed from the dataset, preventing unnecessary loss of information.
- Maintenance required careful consideration of both missing and erroneous information. Missing values were identified using methods such as checking for null values, empty strings, or 'unknown' entries, while error data based on specific patterns were detected using regular expressions.
- Records with missing or erroneous data were removed only if the total number exceeded the specified threshold. This approach ensures efficient data cleaning while preserving the bulk of the dataset.
- Added comments throughout the code to provide clarity on each step and the rationale for decisions made during data preparation.

Figure 13 below shows the final output transformed data table, which represents the cleaned dataset after removing records with missing values or errors. After cleaning, the dataset has 1420 records and 13 attributes, which means that a total of 80 records were removed due to missing or error data. Additionally, the cleansed data is stored in a CSV file "Transformed_Data.csv" for further analysis.



```
Cleaned data has been saved into 'cleaned_data.csv'
```

	CUST_GENDER	AGE	CUST_MARITAL_STATUS	COUNTRY_NAME	CUST_INCOME_LEVEL	EDUCATION	OCCUPATION	HOUSEHOLD_SIZE	YRS_RESIDENCE	AFFINITY_CARD	HOME_THEATER_PACKAGE
0	F	41	NeverM	United States of America	J: 190,000 - 249,999	Masters	Prof.	2	4	0	1
1	M	27	NeverM	United States of America	I: 170,000 - 189,999	Bach.	Sales	2	3	0	0
2	F	20	NeverM	United States of America	H: 150,000 - 169,999	HS-grad	Cleric.	2	2	0	0
3	M	45	Married	United States of America	B: 30,000 - 49,999	Bach.	Exec.	3	5	1	1
4	M	34	NeverM	United States of America	K: 250,000 - 299,999	Masters	Sales	9+	5	1	0
...	...	...	...	...	...	...	...	...	...	...	...
1495	M	17	NeverM	United States of America	C: 50,000 - 69,999	10th	Other	1	1	0	0
1496	M	41	Married	Spain	L: 300,000 and above	Bach.	Exec.	3	4	0	1
1497	M	53	Married	United States of America	J: 190,000 - 249,999	HS-grad	Exec.	3	8	1	1
1498	M	55	Married	United States of America	C: 50,000 - 69,999	HS-grad	Cleric.	3	7	1	1
1499	F	40	Divorc.	United States of America	E: 90,000 - 109,999	HS-grad	Cleric.	2	3	0	1

1420 rows × 13 columns

Figure 13: Final Output Transformed Data Table

2.3 Data Transformation

In this section, we carefully adjusted variables to improve data representation and analysis. The process required careful thought and clear documentation to be reproducible and understandable.

Difference between Transformation Methods:

1. **Basic Python Code Approach:** The basic Python code approach uses lambda functions with the apply function, and allows custom logic to be applied based on conditions. This approach provides the flexibility to define appropriate transformation rules for complex situations. However, it can result in long code due to the need to define custom logic in lambda functions, which can affect code readability.
2. **Pandas Library Approach Approach:** The Pandas library approach implements the mapping functionality through a dictionary map, providing a concise and readable approach to common changes. This approach is ideal for straightforward mapping or transformation where conventional logic is unnecessary, resulting in concise and clear syntax. Moreover, it provides better performance, especially for large datasets, due to better internal controls in pandas operations.

Both approaches achieved the same conversion goals, with the Pandas library approach being shorter and clearer, especially for straightforward maps.

2.3.1 Transform CUST_GENDER into Binary

To convert the 'CUST_GENDER' variable to binary values (F - 0, M - 1), both the main Python code method and the Pandas library method were used. The steps to convert CUST_GENDER to binary F - 0, M -1 are:

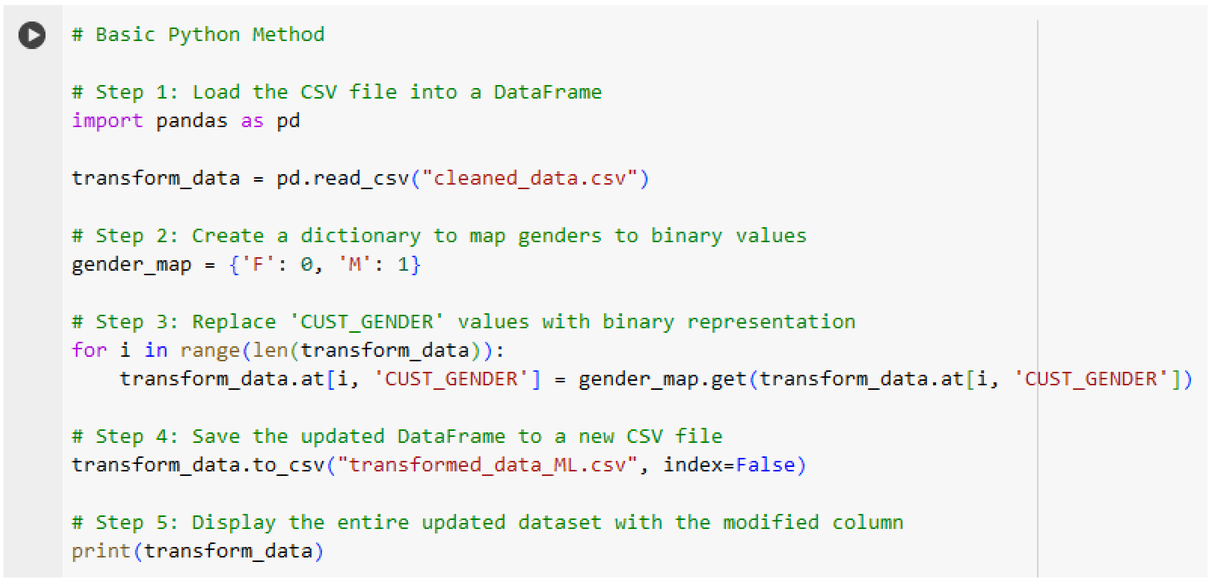
Basic Python Code Approach:

1. First, the dataset is loaded using pandas library.
2. Then, a basic Python for loop is employed to iterate through each gender category in the 'CUST_GENDER' column. Within the loop, a simple if-else condition is used to assign binary values (0 for 'F' and 1 for 'M') based on gender categories.
3. Subsequently, a new column named 'CUST_GENDER_BINARY' is created to store the binary representation of gender.
4. Finally, the transformed data, containing both the original 'CUST_GENDER' and the newly added binary representation, is displayed.

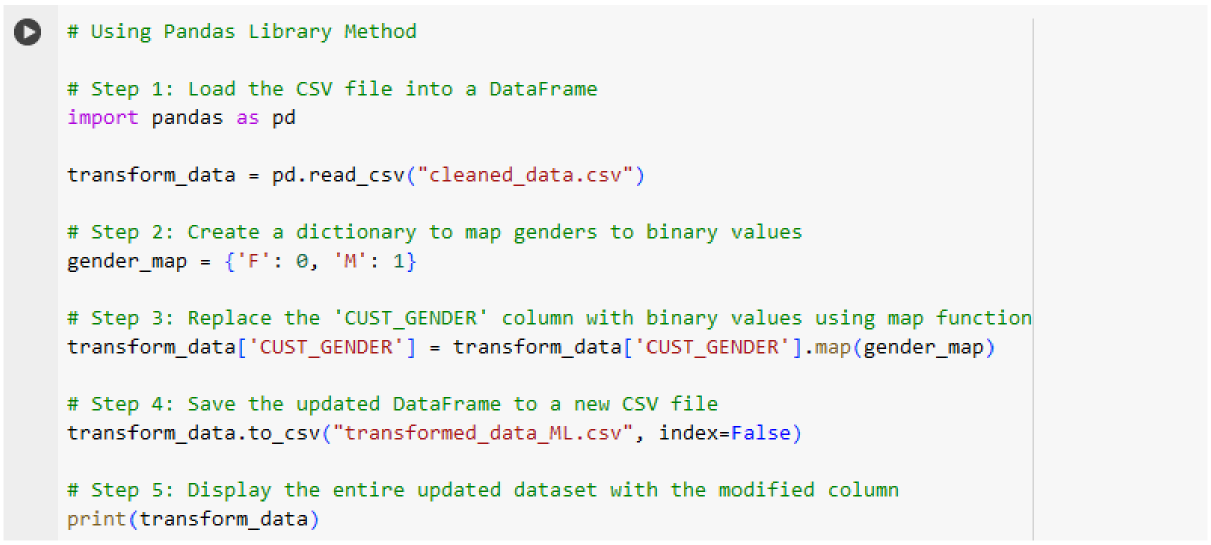
Pandas Library Method Approach:

1. Initially, the dataset is loaded using the pandas library.
2. The Pandas map() function is utilized along with a dictionary map to convert the 'CUST_GENDER' column into binary values (0 for 'F' and 1 for 'M').
3. A new column named 'CUST_GENDER_BINARY' is then added to store the binary representation of gender.
4. Finally, the converted data, including both the original 'CUST_GENDER' and the binary representation, is displayed.

Figures:
Figure 14 shows the code for converting the 'CUST_GENDER' variable to binary representations using both the original Python code and the Pandas library method. The resulting results of this transformation are shown in Figure 15, which shows the identical results obtained with both methods.

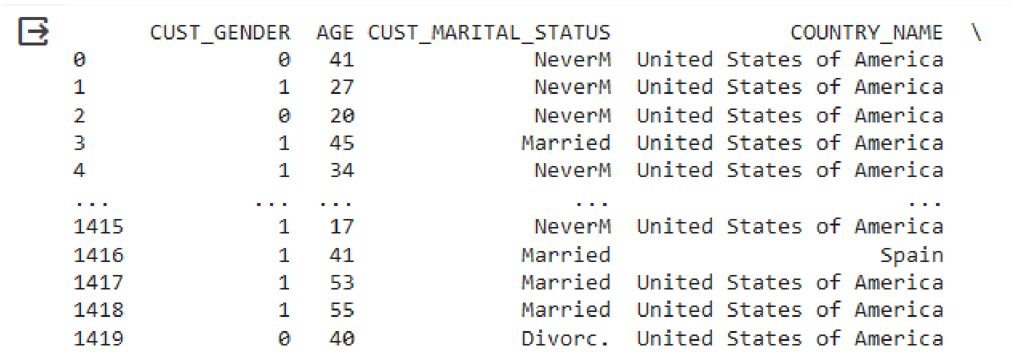


```python
# Basic Python Method

# Step 1: Load the CSV file into a DataFrame
import pandas as pd

transform_data = pd.read_csv("cleaned_data.csv")

# Step 2: Create a dictionary to map genders to binary values
gender_map = {'F': 0, 'M': 1}

# Step 3: Replace 'CUST_GENDER' values with binary representation
for i in range(len(transform_data)):
    transform_data.at[i, 'CUST_GENDER'] = gender_map.get(transform_data.at[i, 'CUST_GENDER'])

# Step 4: Save the updated DataFrame to a new CSV file
transform_data.to_csv("transformed_data_ML.csv", index=False)

# Step 5: Display the entire updated dataset with the modified column
print(transform_data)
```



```python
# Using Pandas Library Method

# Step 1: Load the CSV file into a DataFrame
import pandas as pd

transform_data = pd.read_csv("cleaned_data.csv")

# Step 2: Create a dictionary to map genders to binary values
gender_map = {'F': 0, 'M': 1}

# Step 3: Replace the 'CUST_GENDER' column with binary values using map function
transform_data['CUST_GENDER'] = transform_data['CUST_GENDER'].map(gender_map)

# Step 4: Save the updated DataFrame to a new CSV file
transform_data.to_csv("transformed_data_ML.csv", index=False)

# Step 5: Display the entire updated dataset with the modified column
print(transform_data)
```

Figure 14: CUST_GENDER into Binary Transformation Process



```
      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
...           ...  ...                 ...                       ...
1415            1   17              NeverM  United States of America
1416            1   41             Married                     Spain
1417            1   53             Married  United States of America
1418            1   55             Married  United States of America
1419            0   40             Divorc.  United States of America
```

Figure 15: CUST_GENDER into Binary Transformation Output

Discussion of Coding Difference:
The basic Python code approach employs for loops and conditional statements, offering flexibility in implementing conventional logic but potentially resulting in verbose code due to the need for explicit iteration and condition checks. In contrast, the Pandas library method approach utilizes vectorized operations such as the map function with a dictionary map, providing a concise and readable syntax. It is suitable for straightforward mappings, offering optimal performance and a clear rule structure. While both methods achieve the same transformation objectives, the Pandas library approach is preferred due to its syntactic simplicity and better implementation, especially for straightforward transformations.

2.3.2 COUNTRY_NAME into Ordinal Number

In order to convert the 'COUNTRY_NAME' variable to ordinal numbers based on their occurrence in the dataset in ascending order, two methods were used: a basic Python code method and a Pandas library method. Below were the steps to convert COUNTRY_NAME to Ordinal Numbers:

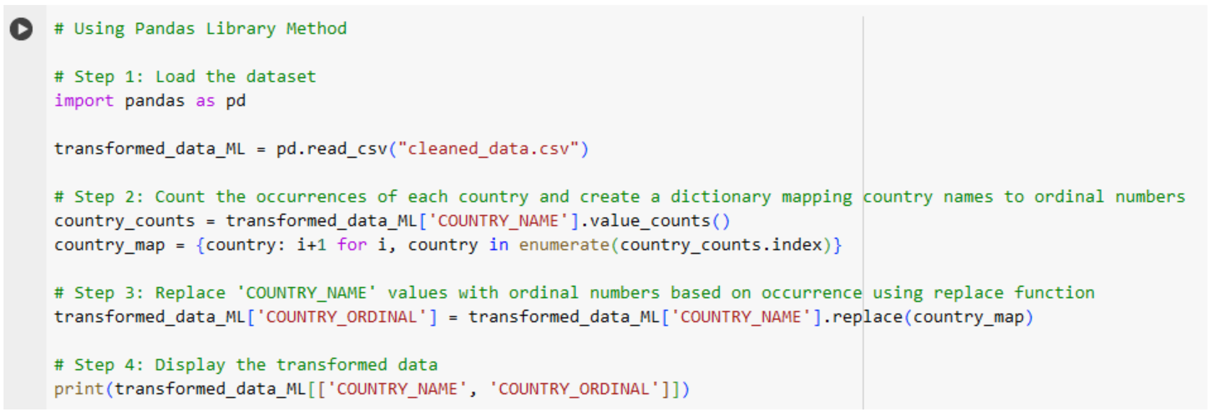
Basic Python Code Approach:

1. Load the dataset.
2. Read the CSV file row by row and collect the 'COUNTRY_NAME' attribute.
3. Utilize Counter to count occurrences of each country in the dataset.
4. Sort the countries based on their increasing frequency.
5. Assign an ordinal number starting from 1 to each country based on its position in the sorted list.
6. Print each country along with its corresponding ordinal number.

Pandas Library Method Approach:

1. Load the dataset containing the 'COUNTRY_NAME' attribute using Pandas.
2. Use the value_counts() method to count occurrences of each country in the dataset.
3. Create a mapping dictionary where each country is mapped to its ordinal number based on its occurrence.
4. Replace the 'COUNTRY_NAME' values with ordinal numbers based on the mapping dictionary.
5. Display the 'COUNTRY_NAME' and 'COUNTRY_ORDINAL' columns, indicating the ordinal numbers assigned to each country.

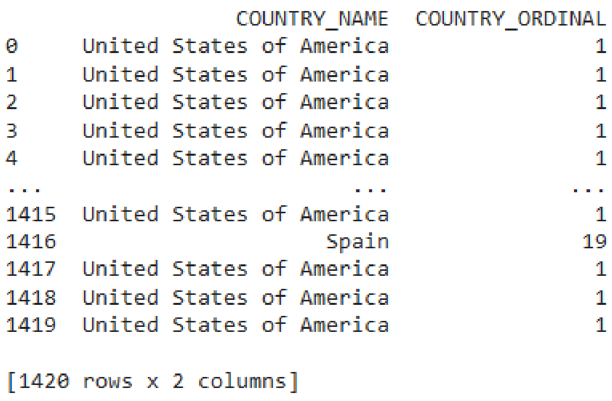
Figures:
Figure 16 shows the code that implements the transformation using the Pandas library method. Finally, Figure 17 shows the results of this transformation, showing the ordinal numbers assigned to each country name.



```python
# Using Pandas Library Method

# Step 1: Load the dataset
import pandas as pd

transformed_data_ML = pd.read_csv("cleaned_data.csv")

# Step 2: Count the occurrences of each country and create a dictionary mapping country names to ordinal numbers
country_counts = transformed_data_ML['COUNTRY_NAME'].value_counts()
country_map = {country: i+1 for i, country in enumerate(country_counts.index)}

# Step 3: Replace 'COUNTRY_NAME' values with ordinal numbers based on occurrence using replace function
transformed_data_ML['COUNTRY_ORDINAL'] = transformed_data_ML['COUNTRY_NAME'].replace(country_map)

# Step 4: Display the transformed data
print(transformed_data_ML[['COUNTRY_NAME', 'COUNTRY_ORDINAL']])
```

Figure 16: COUNTRY_NAME into Ordinal Numbers Transformation

These country names are assigned as per country name frequency occurrence the most occurred country is given the first number and then number assigned from most to least occurred countries in ascending order. We can assign in reverse order as well. (e.g.) the least occurred country can be assigned ordinal number 1.



```
                  COUNTRY_NAME  COUNTRY_ORDINAL
0     United States of America                1
1     United States of America                1
2     United States of America                1
3     United States of America                1
4     United States of America                1
...                        ...              ...
1415  United States of America                1
1416                     Spain               19
1417  United States of America                1
1418  United States of America                1
1419  United States of America                1

[1420 rows x 2 columns]
```

Figure 17: COUNTRY_NAME Transformation Output

Discussion of Coding Difference:

The manual Python code approach involved several steps, including reading the CSV file, extracting country names, counting occurrences, sorting, and assigning ordinal numbers. This method required more lines of code and iteration through the dataset for each step, resulting in a less concise solution.

On the other hand, the Pandas library method provided a more concise and efficient solution. The dataset was loaded directly into a DataFrame, and the value_counts() method efficiently calculated the number of occurrences per country. Then, a mapping dictionary was created to map country names to ordinal numbers based on their occurrence. Finally, the replace() function was used to convert country names to ordinal numbers in the 'COUNTRY_NAME' column.

The key difference lies in the abstraction level and simplicity offered by the Pandas library approach, enhancing code readability and ease of use. Moreover, the Pandas implementation is optimized for performance, making it suitable for handling large datasets.

2.3.3 CUST_INCOME_LEVEL into Ordinal Level

To convert the 'CUST_INCOME_LEVEL' variable into three ordinal levels (1-low, 2-medium, and 3-high income), all original Python code and Pandas library methods were used. Each income category was mapped to its corresponding ordinal level, and additional columns were added to the dataset to store the transformed values. The resulting data provide a clear breakdown of income, facilitating further analysis and interpretation. Here are the steps for change:

Basic Python Code Approach:

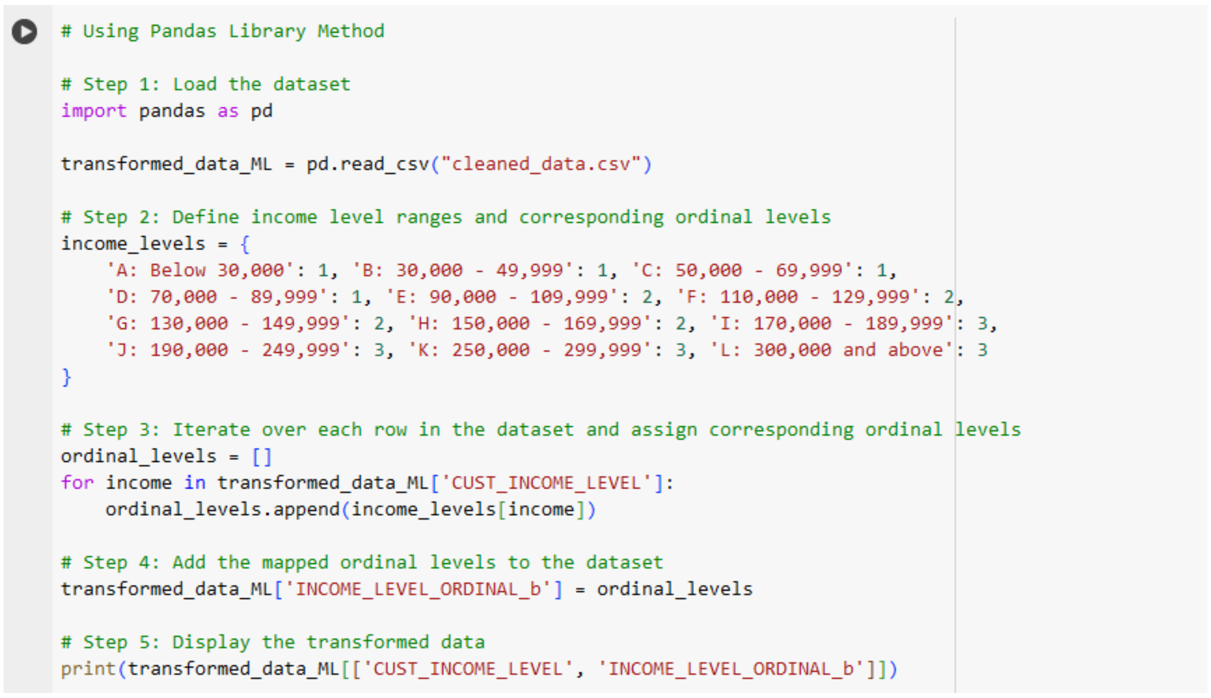
1. Define a dictionary to map each income range to its corresponding ordinal level (low, medium, or high).
2. Iterate over each row in the dataset and assign its corresponding ordinal level based on the dictionary mapping.
3. Add the mapped ordinal levels to the dataset by creating a new column named 'INCOME_LEVEL_ORDINAL'.
4. Display the converted data, showing both the original 'CUST_INCOME_LEVEL' and the updated 'INCOME_LEVEL_ORDINAL' columns.

Pandas Library Method Approach:

1. Define a dictionary to map each income range to its corresponding ordinal level using Pandas, similar to the basic Python approach.
2. Apply the mapping levels to ordinal levels using the mapping dictionary directly with the replace function in Pandas, creating a new column named 'INCOME_LEVEL_ORDINAL'.
3. Display the converted data, showing both the original 'CUST_INCOME_LEVEL' and the updated 'INCOME_LEVEL_ORDINAL' columns.

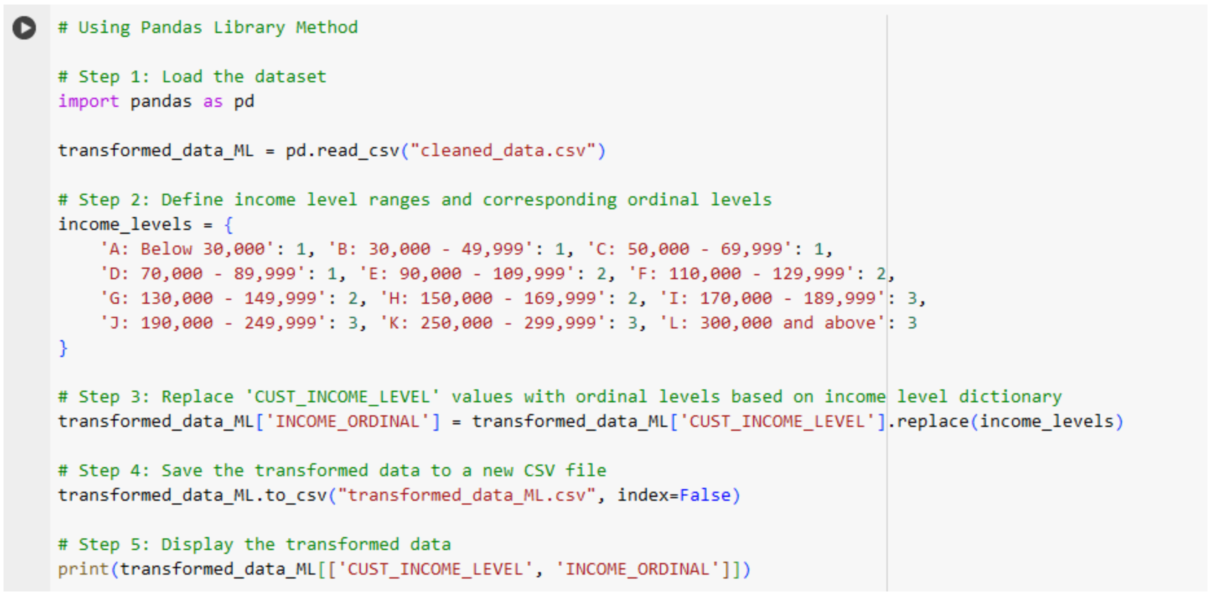
Figures:

Figure 18 shows the code used to convert the 'CUST_INCOME_LEVEL' variable into three ordinal levels using the basic Python code path. Meanwhile, Figure 19 shows the code using Pandas library methods for the same transformation. The step requires defining a dictionary of income maps and then plotting these values up to ordinal values. Finally, transformed data are displayed to show new ordinal levels assigned to each income category in Figure 20.



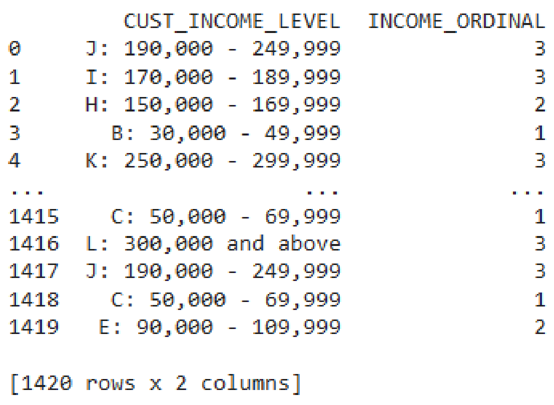
```python
# Using Pandas Library Method

# Step 1: Load the dataset
import pandas as pd

transformed_data_ML = pd.read_csv("cleaned_data.csv")

# Step 2: Define income level ranges and corresponding ordinal levels
income_levels = {
    'A: Below 30,000': 1, 'B: 30,000 - 49,999': 1, 'C: 50,000 - 69,999': 1,
    'D: 70,000 - 89,999': 1, 'E: 90,000 - 109,999': 2, 'F: 110,000 - 129,999': 2,
    'G: 130,000 - 149,999': 2, 'H: 150,000 - 169,999': 2, 'I: 170,000 - 189,999': 3,
    'J: 190,000 - 249,999': 3, 'K: 250,000 - 299,999': 3, 'L: 300,000 and above': 3
}

# Step 3: Iterate over each row in the dataset and assign corresponding ordinal levels
ordinal_levels = []
for income in transformed_data_ML['CUST_INCOME_LEVEL']:
    ordinal_levels.append(income_levels[income])

# Step 4: Add the mapped ordinal levels to the dataset
transformed_data_ML['INCOME_LEVEL_ORDINAL_b'] = ordinal_levels

# Step 5: Display the transformed data
print(transformed_data_ML[['CUST_INCOME_LEVEL', 'INCOME_LEVEL_ORDINAL_b']])
```

Figure 18: Basic Python Code for CUST_INCOME_LEVEL Transformation



```python
# Using Pandas Library Method

# Step 1: Load the dataset
import pandas as pd

transformed_data_ML = pd.read_csv("cleaned_data.csv")

# Step 2: Define income level ranges and corresponding ordinal levels
income_levels = {
    'A: Below 30,000': 1, 'B: 30,000 - 49,999': 1, 'C: 50,000 - 69,999': 1,
    'D: 70,000 - 89,999': 1, 'E: 90,000 - 109,999': 2, 'F: 110,000 - 129,999': 2,
    'G: 130,000 - 149,999': 2, 'H: 150,000 - 169,999': 2, 'I: 170,000 - 189,999': 3,
    'J: 190,000 - 249,999': 3, 'K: 250,000 - 299,999': 3, 'L: 300,000 and above': 3
}

# Step 3: Replace 'CUST_INCOME_LEVEL' values with ordinal levels based on income level dictionary
transformed_data_ML['INCOME_ORDINAL'] = transformed_data_ML['CUST_INCOME_LEVEL'].replace(income_levels)

# Step 4: Save the transformed data to a new CSV file
transformed_data_ML.to_csv("transformed_data_ML.csv", index=False)

# Step 5: Display the transformed data
print(transformed_data_ML[['CUST_INCOME_LEVEL', 'INCOME_ORDINAL']])
```

Figure 19: Pandas Method for CUST_INCOME_LEVEL Transformation



```
          CUST_INCOME_LEVEL  INCOME_ORDINAL
0      J: 190,000 - 249,999               3
1      I: 170,000 - 189,999               3
2      H: 150,000 - 169,999               2
3        B: 30,000 - 49,999               1
4      K: 250,000 - 299,999               3
...                     ...             ...
1415     C: 50,000 - 69,999               1
1416   L: 300,000 and above               3
1417   J: 190,000 - 249,999               3
1418     C: 50,000 - 69,999               1
1419    E: 90,000 - 109,999               2

[1420 rows x 2 columns]
```

Figure 20: Output of CUST_INCOME_LEVEL Transformation

Discussion of Coding Difference:

The basic Python code approach and the Pandas library method differ significantly in their implementation and syntax for converting the 'CUST_INCOME_LEVEL' variable to ordinal levels.

In the basic Python approach, a manual loop iterates over each income level in the dataset, assigning it to its corresponding ordinal value based on a predefined mapping dictionary. This method requires explicit replication of the dataset and manual coding for each income level, resulting in more lines of code and increased complexity.

In contrast, the Pandas library method utilizes the replace() function directly on the DataFrame. By defining a mapping dictionary and applying it directly with the replace() function, the Pandas method achieves the same outcome with fewer lines of code and reduced complexity compared to the basic Python approach.

While both approaches achieve the same outcome, the Pandas library method offers a more efficient and elegant solution by leveraging the built-in functionality of Pandas for data manipulation.

2.3.4 EDUCATION into Ordinal Number

In this section, we converted the 'EDUCATION' variable into ordinal numbers based on USA education level in Ascending order, from preschool to doctoral degrees. The USA education levels we have research are almost identical in dataset. We used both core Python code and Pandas library methods for this transformation. The steps are as below:

Basic Python Code Approach:

1. Developed a mapping dictionary to assign ordinal numbers to different levels of education based on the USA education system in ascending order.

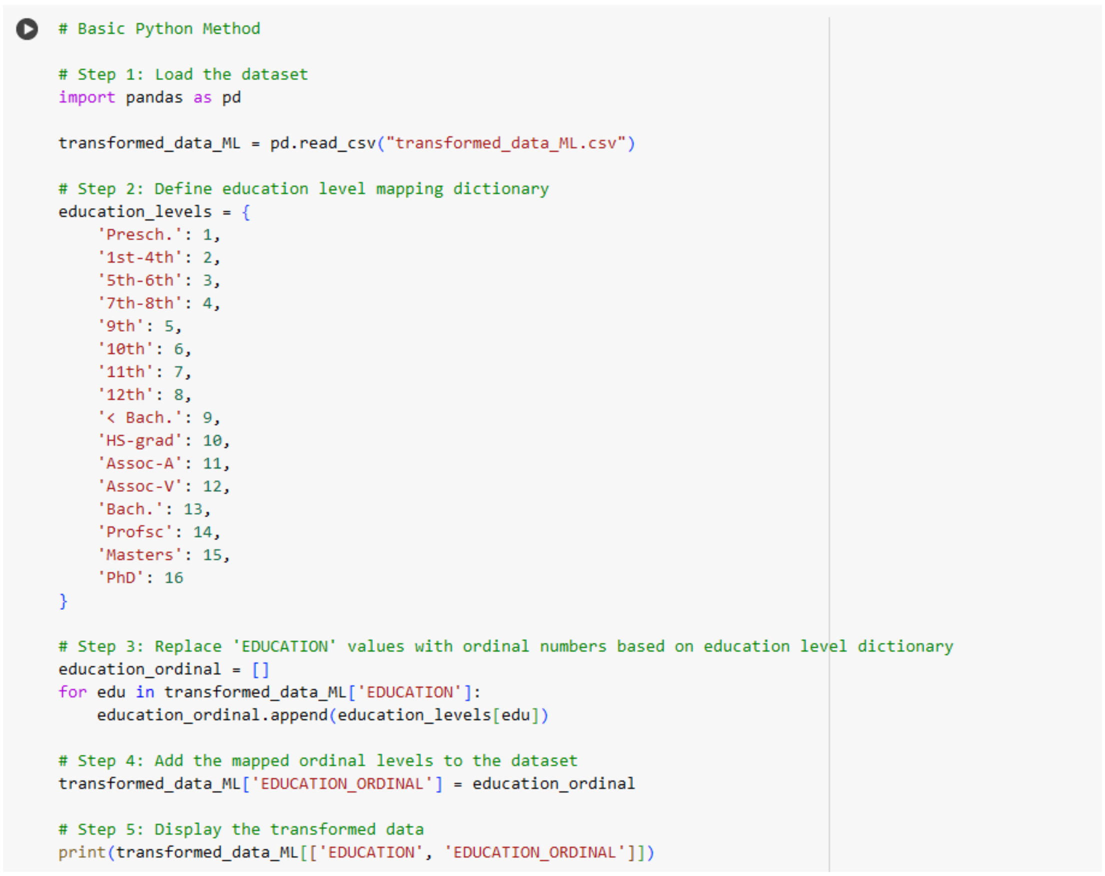
2. Read the CSV file containing education information, iterating over each row to map education levels to their corresponding ordinal numbers using the defined mapping dictionary.
3. The mapping assigns lower ordinal numbers for lower levels of education (e.g., preschool, elementary school) and higher ordinal numbers for higher levels of education (e.g., bachelor's degree, master's degree, doctoral degree). The mapped ordinal education level was added to the dataset as a new column named 'EDUCATION_ORDINAL_b'.
4. Printed the transformed data, displaying both the original 'EDUCATION' column and the updated 'EDUCATION_ORDINAL_b' column.

Pandas Library Method Approach:

1. Defined a mapping dictionary to assign ordinal numbers to education levels based on the USA education system.
2. Read the CSV file containing the education data into a pandas DataFrame.
3. The mapping assigns lower ordinal numbers for lower levels of education (e.g., preschool, elementary school) and higher ordinal numbers for higher levels of education (e.g., bachelor's degree, master's degree, doctoral degree). Using the replace() function and the mapping dictionary, a new column named 'EDUCATION_ORDINAL' containing the ordinal number corresponding to each level of education was created.
4. Printed the transformed data, showing both the original 'EDUCATION' column and the updated 'EDUCATION_ORDINAL' column.

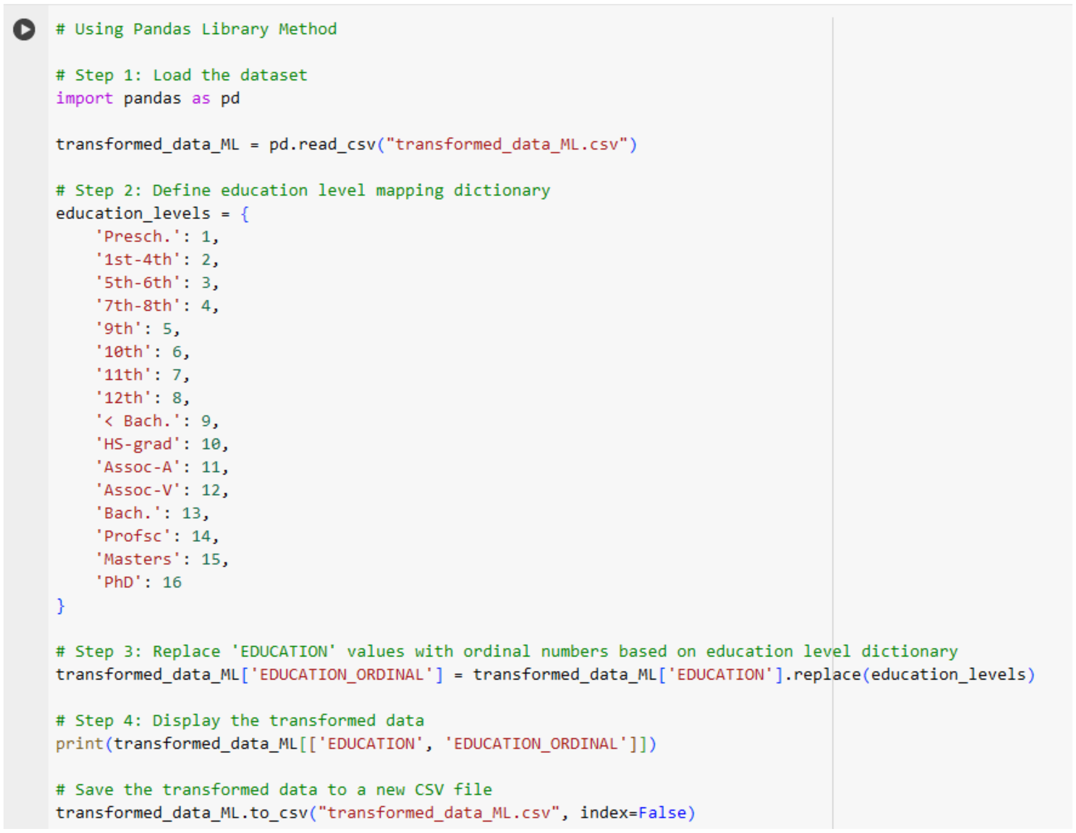
Figures:

Figure 21 shows the code used to convert the 'EDUCATION' variable to an ordinal number using basic Python code. Similarly, Figure 22 shows the code that implements the transformation process using the Pandas library method. Finally, Figure 23 shows the results of the adjustment, showing the 'EDUCATION' variable plotted in ordinal numbers according to USA education levels.



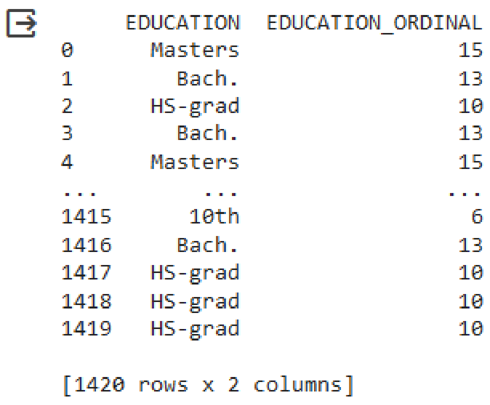
```python
# Basic Python Method

# Step 1: Load the dataset
import pandas as pd

transformed_data_ML = pd.read_csv("transformed_data_ML.csv")

# Step 2: Define education level mapping dictionary
education_levels = {
    'Presch.': 1,
    '1st-4th': 2,
    '5th-6th': 3,
    '7th-8th': 4,
    '9th': 5,
    '10th': 6,
    '11th': 7,
    '12th': 8,
    '< Bach.': 9,
    'HS-grad': 10,
    'Assoc-A': 11,
    'Assoc-V': 12,
    'Bach.': 13,
    'Profsc': 14,
    'Masters': 15,
    'PhD': 16
}

# Step 3: Replace 'EDUCATION' values with ordinal numbers based on education level dictionary
education_ordinal = []
for edu in transformed_data_ML['EDUCATION']:
    education_ordinal.append(education_levels[edu])

# Step 4: Add the mapped ordinal levels to the dataset
transformed_data_ML['EDUCATION_ORDINAL'] = education_ordinal

# Step 5: Display the transformed data
print(transformed_data_ML[['EDUCATION', 'EDUCATION_ORDINAL']])
```

Figure 21: Basic Python Code for EDUCATION Transformation



```python
# Using Pandas Library Method

# Step 1: Load the dataset
import pandas as pd

transformed_data_ML = pd.read_csv("transformed_data_ML.csv")

# Step 2: Define education level mapping dictionary
education_levels = {
    'Presch.': 1,
    '1st-4th': 2,
    '5th-6th': 3,
    '7th-8th': 4,
    '9th': 5,
    '10th': 6,
    '11th': 7,
    '12th': 8,
    '< Bach.': 9,
    'HS-grad': 10,
    'Assoc-A': 11,
    'Assoc-V': 12,
    'Bach.': 13,
    'Profsc': 14,
    'Masters': 15,
    'PhD': 16
}

# Step 3: Replace 'EDUCATION' values with ordinal numbers based on education level dictionary
transformed_data_ML['EDUCATION_ORDINAL'] = transformed_data_ML['EDUCATION'].replace(education_levels)

# Step 4: Display the transformed data
print(transformed_data_ML[['EDUCATION', 'EDUCATION_ORDINAL']])

# Save the transformed data to a new CSV file
transformed_data_ML.to_csv("transformed_data_ML.csv", index=False)
```

Figure 22: Pandas Method for Education Transformation



```
      EDUCATION  EDUCATION_ORDINAL
0       Masters                 15
1         Bach.                 13
2       HS-grad                 10
3         Bach.                 13
4       Masters                 15
...         ...                ...
1415       10th                  6
1416      Bach.                 13
1417    HS-grad                 10
1418    HS-grad                 10
1419    HS-grad                 10

[1420 rows x 2 columns]
```

Figure 23: Output of Education Transformation

Discussion of Coding Difference:
The coding difference lies in the implementation of the conversion process. Both methods utilize a mapping dictionary to assign ordinal numbers to education levels based on the USA education system.

In the basic Python code approach, manual iteration over the dataset is required to implement the mapping, resulting in a more verbose and less efficient solution.

On the other hand, the Pandas library method leverages vectorized operations for efficient transformation. It provides a more concise and readable syntax, enhancing code clarity and reducing complexity.

Overall, while both approaches achieve the same outcome, the Pandas library method offers a more efficient and elegant solution for transforming education levels into ordinal numbers.

2.3.5 HOUSEHOLD_SIZE into Ordinal Number

In this conversion, the 'HOUSEHOLD_SIZE' variable was converted to an ordinal number based on the number of people in the household. The dataset was constructed to generate ordinal numbers for household sizes, where larger received lower ordinal numbers and smaller received higher ordinal numbers. Here is a breakdown of the steps taken:

Basic Python Code Approach:

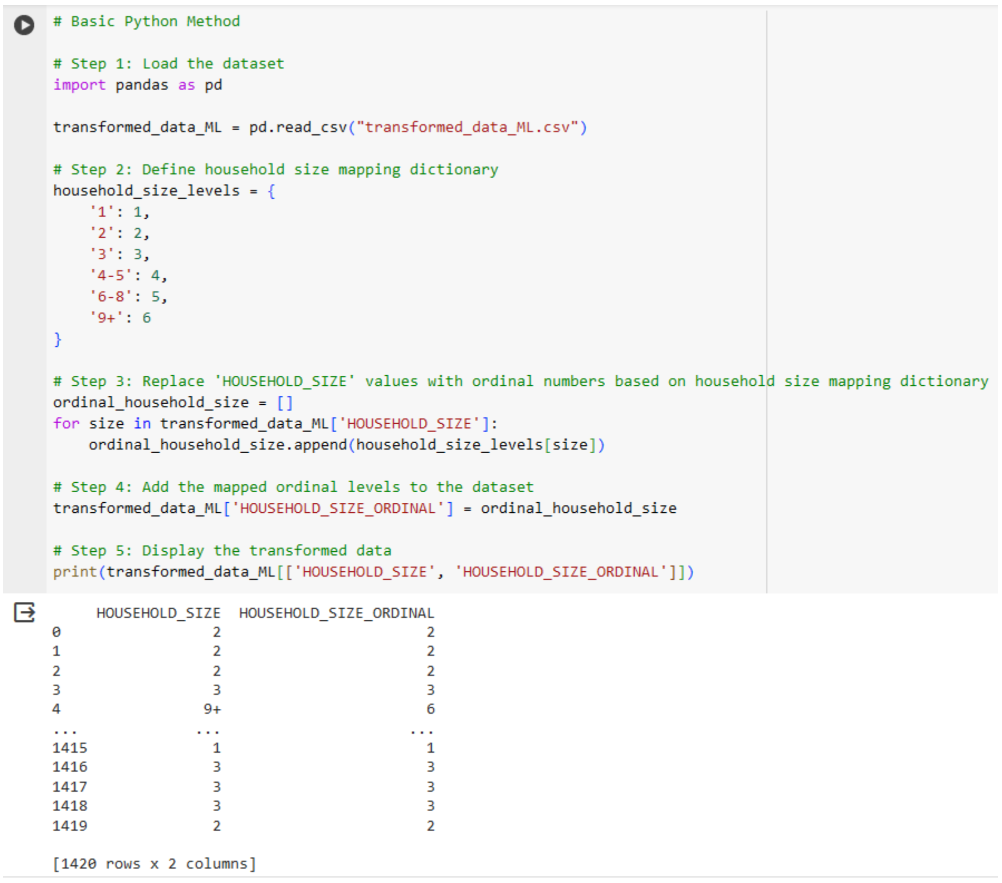
1. The CSV file is read, and the results for each household size are counted.
2. The order of household density was defined on the basis of population.
3. Created a mapping dictionary where ordinal numbers will be assigned to each house size based on a defined system.
4. Household sizes and their assigned ordinal numbers were printed, with 1 ordinal number assigned to the highest household size ('9+') and so on.

Pandas Library Method Approach:

1. Load the dataset using the Pandas library.
2. Define a mapping dictionary where each household size is assigned an ordinal number.
3. Replace the 'HOUSEHOLD_SIZE' values with their corresponding ordinal numbers based on the mapping dictionary.
4. Print the transformed data, showing both the original 'HOUSEHOLD_SIZE' column and the new 'HOUSEHOLD_SIZE_ORDINAL' column.

Figures:

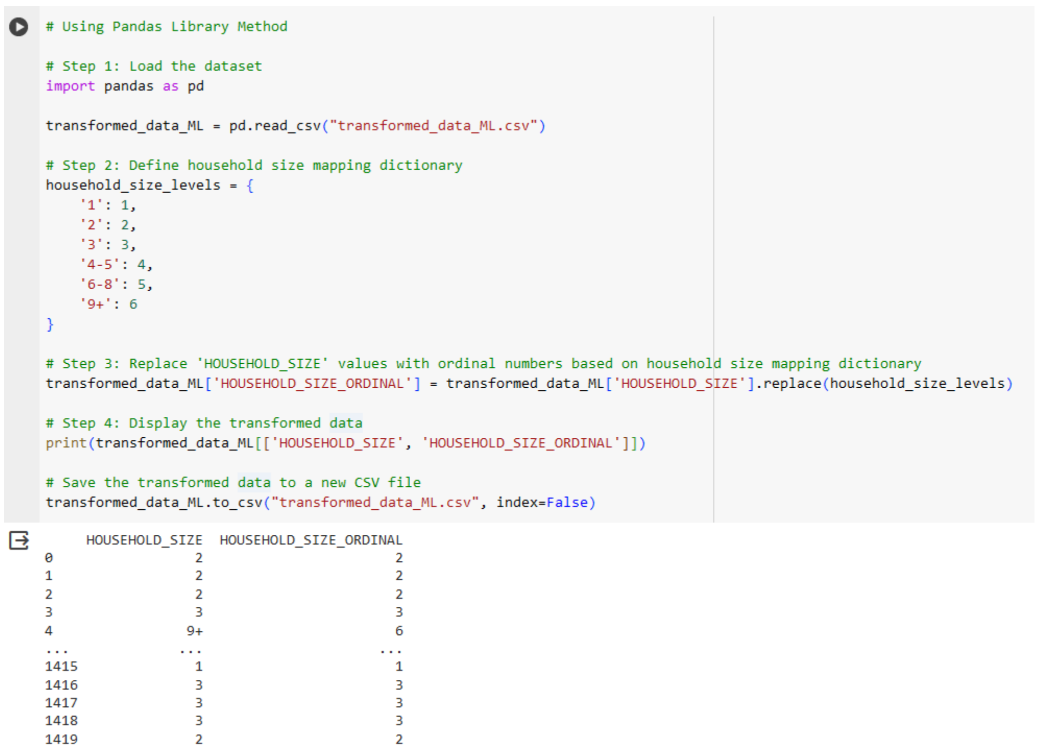
Figure 24 illustrates the code for converting the 'HOUSEHOLD_SIZE' variable to an ordinal number using the original Python code and results. This output shows the size of each household along with its assigned ordinal number, with the highest household size ('9+') designated as ordinal number 1, and '6-8'. followed by the ordinal number 2, and so on. Similarly, Figure 25 shows the code for converting the 'HOUSEHOLD_SIZE' variable to an ordinal number using Pandas library method and output, which shows the ordinal numbers assigned to all household sizes over the full dataset. These results provide valuable insights into the hierarchy of household sizes represented and enhance the understanding of classification based on the number of recruits.



```
# Basic Python Method

# Step 1: Load the dataset
import pandas as pd

transformed_data_ML = pd.read_csv("transformed_data_ML.csv")

# Step 2: Define household size mapping dictionary
household_size_levels = {
    '1': 1,
    '2': 2,
    '3': 3,
    '4-5': 4,
    '6-8': 5,
    '9+': 6
}

# Step 3: Replace 'HOUSEHOLD_SIZE' values with ordinal numbers based on household size mapping dictionary
ordinal_household_size = []
for size in transformed_data_ML['HOUSEHOLD_SIZE']:
    ordinal_household_size.append(household_size_levels[size])

# Step 4: Add the mapped ordinal levels to the dataset
transformed_data_ML['HOUSEHOLD_SIZE_ORDINAL'] = ordinal_household_size

# Step 5: Display the transformed data
print(transformed_data_ML[['HOUSEHOLD_SIZE', 'HOUSEHOLD_SIZE_ORDINAL']])
```

```
     HOUSEHOLD_SIZE  HOUSEHOLD_SIZE_ORDINAL
0                 2                       2
1                 2                       2
2                 2                       2
3                 3                       3
4                9+                       6
...             ...                     ...
1415              1                       1
1416              3                       3
1417              3                       3
1418              3                       3
1419              2                       2

[1420 rows x 2 columns]
```

Figure 24: Basic Python Code & Output for HOUSEHOLD_SIZE Transformation



```python
# Using Pandas Library Method

# Step 1: Load the dataset
import pandas as pd

transformed_data_ML = pd.read_csv("transformed_data_ML.csv")

# Step 2: Define household size mapping dictionary
household_size_levels = {
    '1': 1,
    '2': 2,
    '3': 3,
    '4-5': 4,
    '6-8': 5,
    '9+': 6
}

# Step 3: Replace 'HOUSEHOLD_SIZE' values with ordinal numbers based on household size mapping dictionary
transformed_data_ML['HOUSEHOLD_SIZE_ORDINAL'] = transformed_data_ML['HOUSEHOLD_SIZE'].replace(household_size_levels)

# Step 4: Display the transformed data
print(transformed_data_ML[['HOUSEHOLD_SIZE', 'HOUSEHOLD_SIZE_ORDINAL']])

# Save the transformed data to a new CSV file
transformed_data_ML.to_csv("transformed_data_ML.csv", index=False)
```

```
     HOUSEHOLD_SIZE  HOUSEHOLD_SIZE_ORDINAL
0                 2                       2
1                 2                       2
2                 2                       2
3                 3                       3
4                9+                       6
...             ...                     ...
1415              1                       1
1416              3                       3
1417              3                       3
1418              3                       3
1419              2                       2
```

Figure 25: Pandas Method Code & Output for HOUSEHOLD_SIZE Transformation

Discussion of Coding Difference:
Both methods employ different techniques to achieve the same goal of assigning ordinal numbers to household sizes. In the basic Python code approach, manual iteration and dictionary manipulation are used to count occurrences and create the mapping dictionary. Conversely, the Pandas library method utilizes built-in functions such as replace() to efficiently replace household size values with their corresponding ordinal numbers based on a pre-defined mapping dictionary. While both methods achieve the desired transformation, the Pandas library method offers a more streamlined and concise solution due to its inherent functionalities for data manipulation.

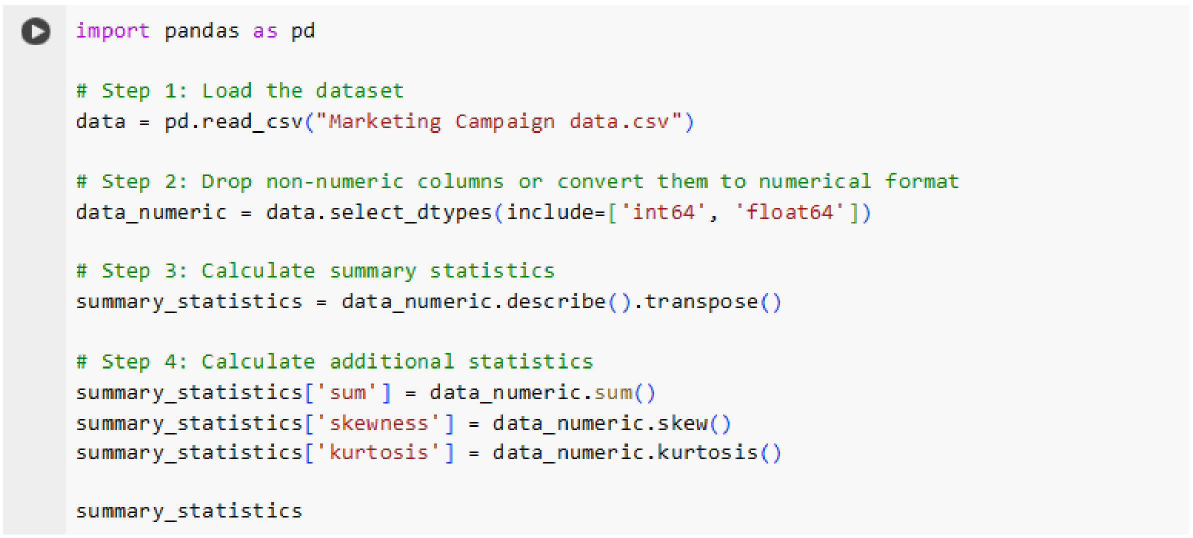
3. Data Analysis

The data analytics process involves systematically exploring, preparing, transforming, and modeling data to uncover meaningful insights, patterns, and trends. It typically involves steps such as analyzing information to understand its structure, summarizing its characteristics through descriptive statistics, visualizing data distribution with graphs and charts, relationships among variables through correlational analysis on, and the use of statistical and machine learning techniques to generate actionable insights [1]. In this data analysis section, we aim to draw meaningful insights and patterns from the available dataset. This often involves analyzing the data, summarizing its characteristics, and identifying key trends or relationships. To display the aggregate statistics for all changes we performed the following steps:

1. Insert the Pandas library (Marketing Campaign data) into the dataset.
2. Clear or convert non-numeric characters to numeric form
3. Calculate summary statistics such as sum, mean, standard deviation, skewness, and kurtosis for each variable using the describe() function.
4. Manually calculate the sum, skewness, and kurtosis for each variable and add them to the summary statistics dataframe.

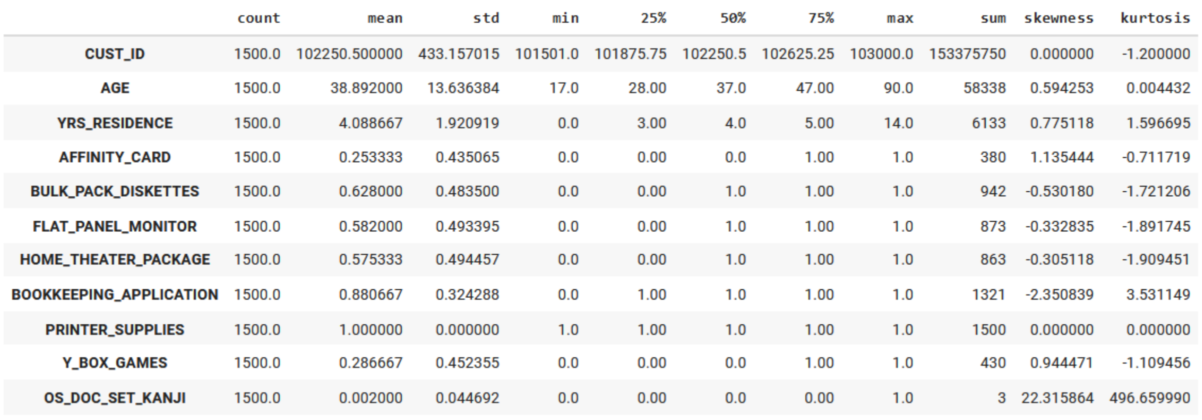
Figures:

Figures 26 and 27 show the Python code for calculating summary statistics and corresponding results.



```python
import pandas as pd

# Step 1: Load the dataset
data = pd.read_csv("Marketing Campaign data.csv")

# Step 2: Drop non-numeric columns or convert them to numerical format
data_numeric = data.select_dtypes(include=['int64', 'float64'])

# Step 3: Calculate summary statistics
summary_statistics = data_numeric.describe().transpose()

# Step 4: Calculate additional statistics
summary_statistics['sum'] = data_numeric.sum()
summary_statistics['skewness'] = data_numeric.skew()
summary_statistics['kurtosis'] = data_numeric.kurtosis()

summary_statistics
```

Figure 26: Code for Calculating Summary Statistics



	count	mean	std	min	25%	50%	75%	max	sum	skewness	kurtosis
CUST_ID	1500.0	102250.500000	433.157015	101501.0	101875.75	102250.5	102625.25	103000.0	153375750	0.000000	-1.200000
AGE	1500.0	38.892000	13.636384	17.0	28.00	37.0	47.00	90.0	58338	0.594253	0.004432
YRS_RESIDENCE	1500.0	4.088667	1.920919	0.0	3.00	4.0	5.00	14.0	6133	0.775118	1.596695
AFFINITY_CARD	1500.0	0.253333	0.435065	0.0	0.00	0.0	1.00	1.0	380	1.135444	-0.711719
BULK_PACK_DISKETTES	1500.0	0.628000	0.483500	0.0	0.00	1.0	1.00	1.0	942	-0.530180	-1.721206
FLAT_PANEL_MONITOR	1500.0	0.582000	0.493395	0.0	0.00	1.0	1.00	1.0	873	-0.332835	-1.891745
HOME_THEATER_PACKAGE	1500.0	0.575333	0.494457	0.0	0.00	1.0	1.00	1.0	863	-0.305118	-1.909451
BOOKKEEPING_APPLICATION	1500.0	0.880667	0.324288	0.0	1.00	1.0	1.00	1.0	1321	-2.350839	3.531149
PRINTER_SUPPLIES	1500.0	1.000000	0.000000	1.0	1.00	1.0	1.00	1.0	1500	0.000000	0.000000
Y_BOX_GAMES	1500.0	0.286667	0.452355	0.0	0.00	0.0	1.00	1.0	430	0.944471	-1.109456
OS_DOC_SET_KANJI	1500.0	0.002000	0.044692	0.0	0.00	0.0	0.00	1.0	3	22.315864	496.659990

Figure 27: Output of Summary Statistics

4. Data Exploration

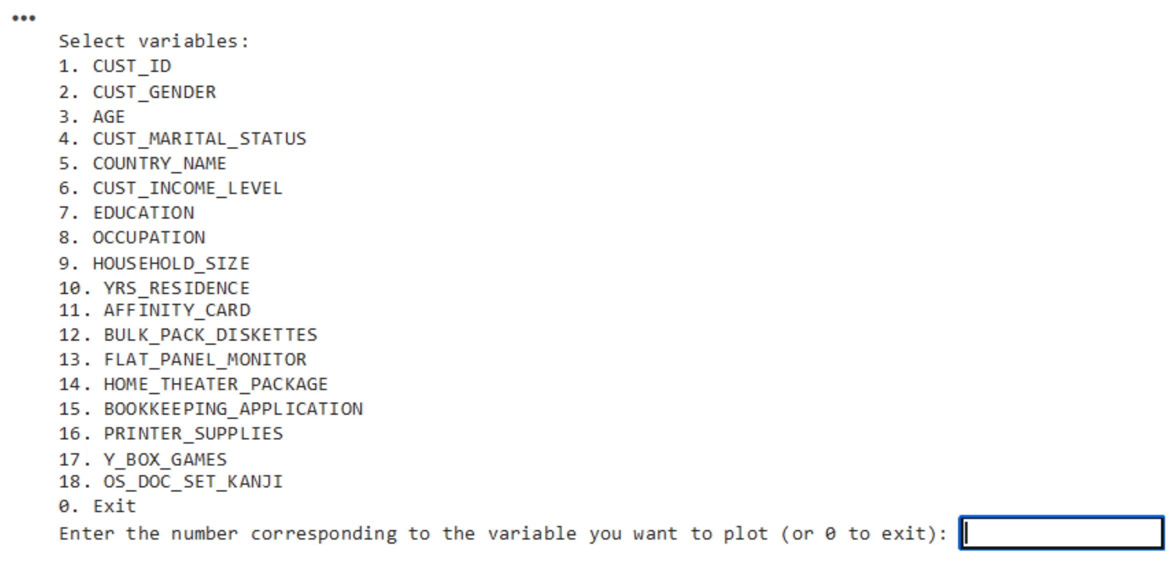
Data analysis, a key step in data analysis, involves detailed analysis of datasets to understand their distribution, patterns, and characteristics. Through visualization techniques such as histograms, researchers gain insight into the underlying structure of the data. In this Python framework, users engage in interactive data analysis, selecting variables of interest to generate histogram plots. The program provides an intuitive understanding of the nuances of the dataset, enabling users to make informed decisions. This approach is consistent with the foundational work of exploratory data analysis by Tukey [2]. Below steps were performed to achieve this:

1. Use the pd.read_csv() function to read the dataset from a CSV file into a pandas DataFrame. This allows us to work with the data in a structured format.
2. Use the .drop() method to drop the 'COMMENTS' column from the DataFrame. This step ensures that only statistical variables are considered for plotting.
3. A loop that continues to run until the user chooses to exit is used. This ensures that the program remains interactive and responsive to user input.
4. Print the list of available variables to the console, and let the user know which options they can select for plotting.
5. Use the input() function to ask the user to enter the number corresponding to the variable to be plotted.
6. Validate the user input to ensure that it is a number and that there is a range of available options. If the input does not work, provide appropriate feedback and ask the user to try again.
7. Retrieve the name of the selected variable based on the user input. This allows us to access the corresponding column in the DataFrame for plotting purposes.
8. Use the matplotlib plt.hist() function to create a histogram plot of the selected variable. This visual representation helps to analyze the distribution of the values of the variable.
9. Set titles, axis labels, and change the grid layout to improve the readability and interpretability of the histogram.
10. Use plt.show() to display the histogram plot to the user. This allows a visual examination of the distribution of the selected variables.

By following these steps, users can interactively explore the distribution of variables in the dataset through histogram plots, facilitating data analysis and analysis.

Figures:

The program presents a user interface showing the variables available for selection, illustrated in Figure 28. The program can enter a number corresponding to the variable to be analyzed or select '0' to exit processing in the 1990s. When a variable is selected, the program creates a histogram plot, as shown in Figures 29 through 31, and allows users to explore the distribution of the selected variable. Sample tests show generated histograms for variables such as AGE, COUNTRY_NAME, and CUST_INCOME_LEVEL, giving users insight into the distribution patterns of different variables in the dataset.



```
Select variables:
1. CUST_ID
2. CUST_GENDER
3. AGE
4. CUST_MARITAL_STATUS
5. COUNTRY_NAME
6. CUST_INCOME_LEVEL
7. EDUCATION
8. OCCUPATION
9. HOUSEHOLD_SIZE
10. YRS_RESIDENCE
11. AFFINITY_CARD
12. BULK_PACK_DISKETTES
13. FLAT_PANEL_MONITOR
14. HOME_THEATER_PACKAGE
15. BOOKKEEPING_APPLICATION
16. PRINTER_SUPPLIES
17. Y_BOX_GAMES
18. OS_DOC_SET_KANJI
0. Exit
Enter the number corresponding to the variable you want to plot (or 0 to exit):
```

Figure 28: User Interface - Variable Selection

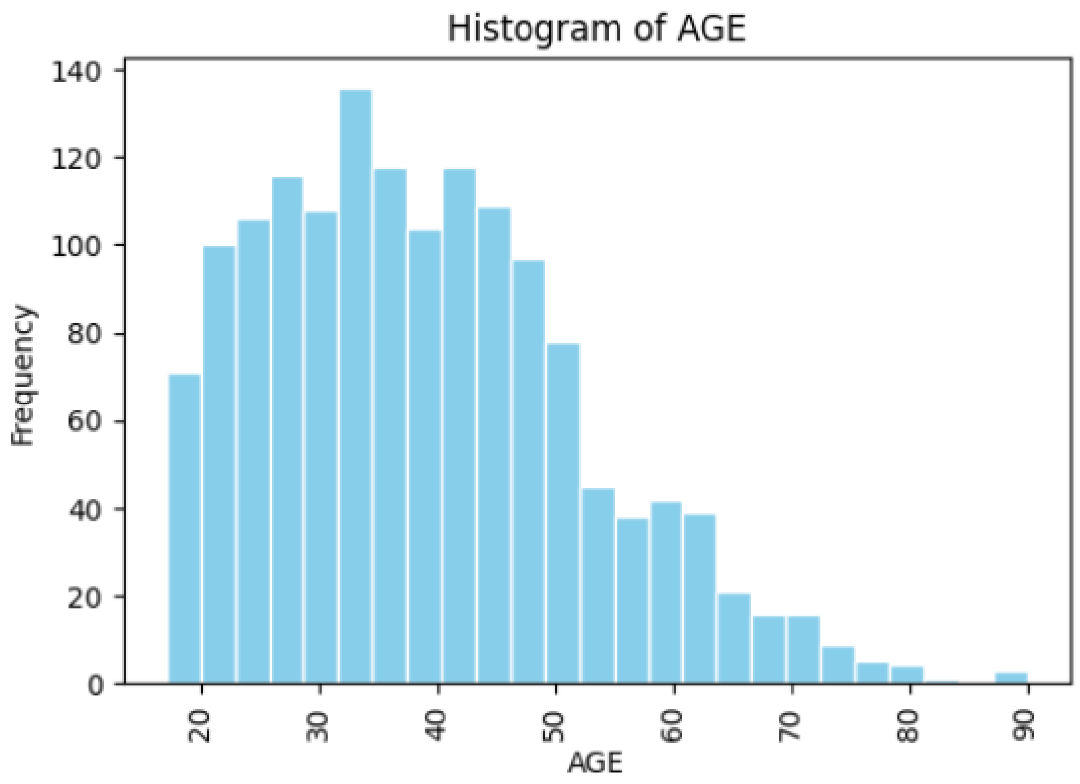


Figure 29: Histogram of AGE

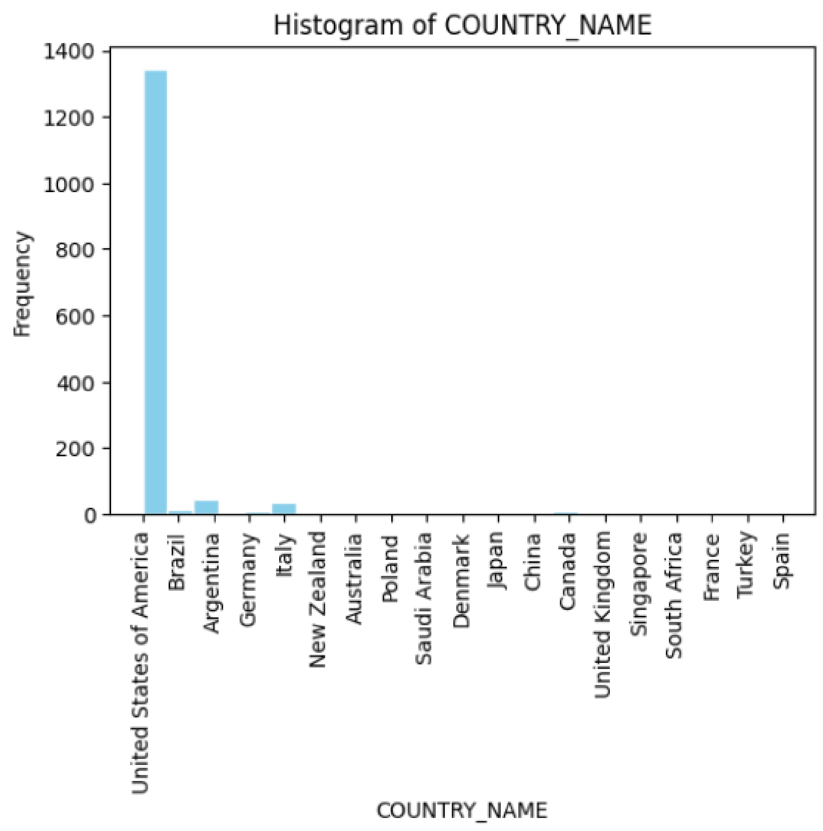


Figure 30: Histogram of COUNTRY_NAME

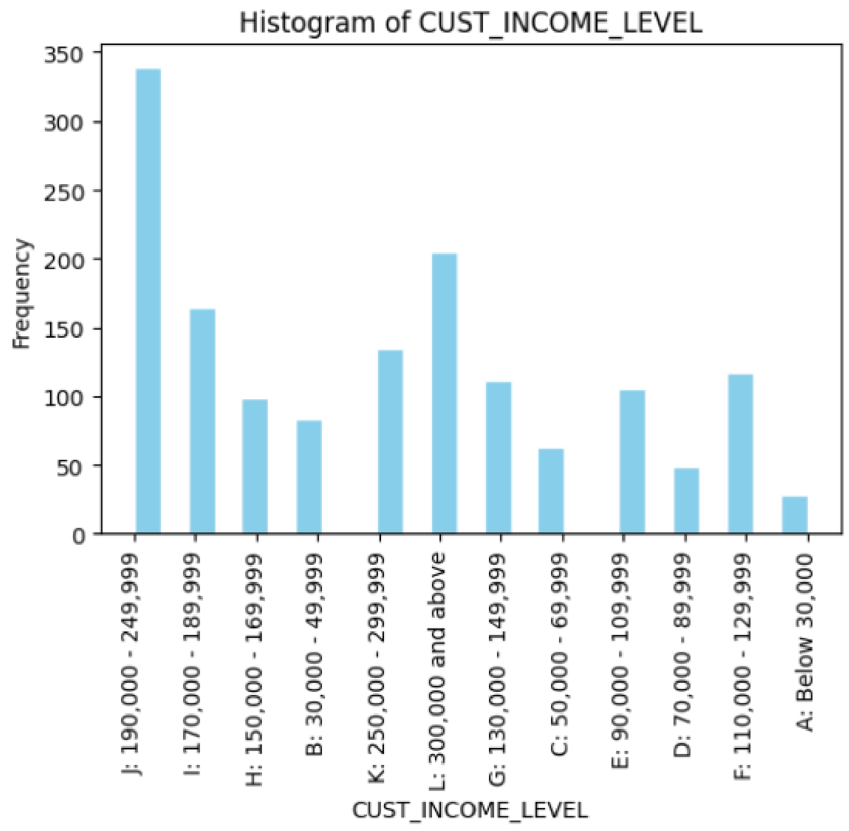


Figure 31: Histogram of CUST_INCOME_LEVEL

5. Data Mining

In the realm of data mining, which involves extracting valuable insights from large datasets using various computational techniques [3], we embark on a journey to develop a logistic regression machine learning model using Python, using 1000 random customer records using transformed marketing campaign data. This model serves as a predictive tool for identifying patterns in consumer behavior and helps in predicting consumer affinity card purchases. In addition, we implement forecasting functions driven by the logistic regression model, providing a user-friendly interface for real-time customer forecasting. By testing using 100 customer records, we validate the accuracy of the application, ensuring its reliability for practical use.

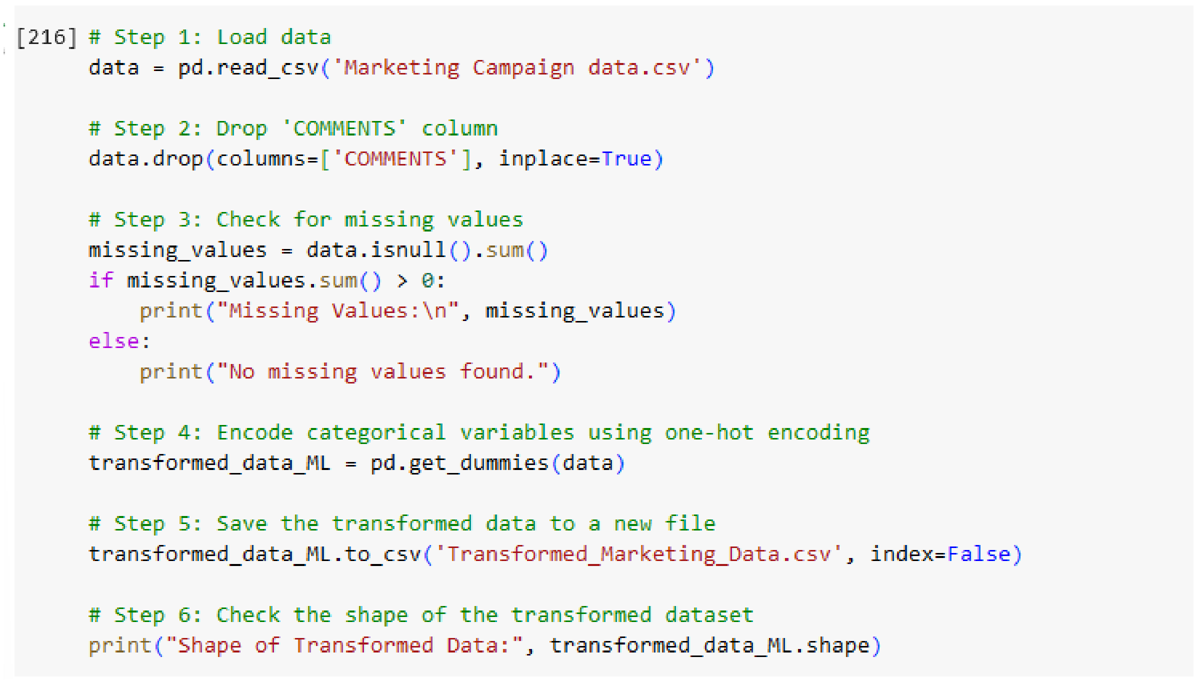
5.1 Build Logistic Regression Model

In this case, we aim to build a logistic regression machine learning model using Python with a dataset obtained from a marketing campaign. The dataset has been preprocessed to handle missing values and encode categorical variables. We used 1000 random records from the dataset and built a logistic regression model. The steps below were developed to achieve this:

1. **Data Transformation:** The dataset is loaded, and the 'COMMENTS' column is dropped to highlight statistical modifications. Missing values are checked and discarded, and categorical variables are created using one-hot encoding. The converted dataset is saved to a new CSV file.
2. **Building Logistic Regression Model:**
   - The transformed data is loaded.
   - 1000 random customer records are selected from the dataset.
   - The dataset is split into features (X) and the target variable (y).
   - Data is further split into training and testing sets.
   - A logistic regression model is built using the training data.
   - The model is evaluated using the testing data.
   - Accuracy and classification report metrics are calculated and displayed.

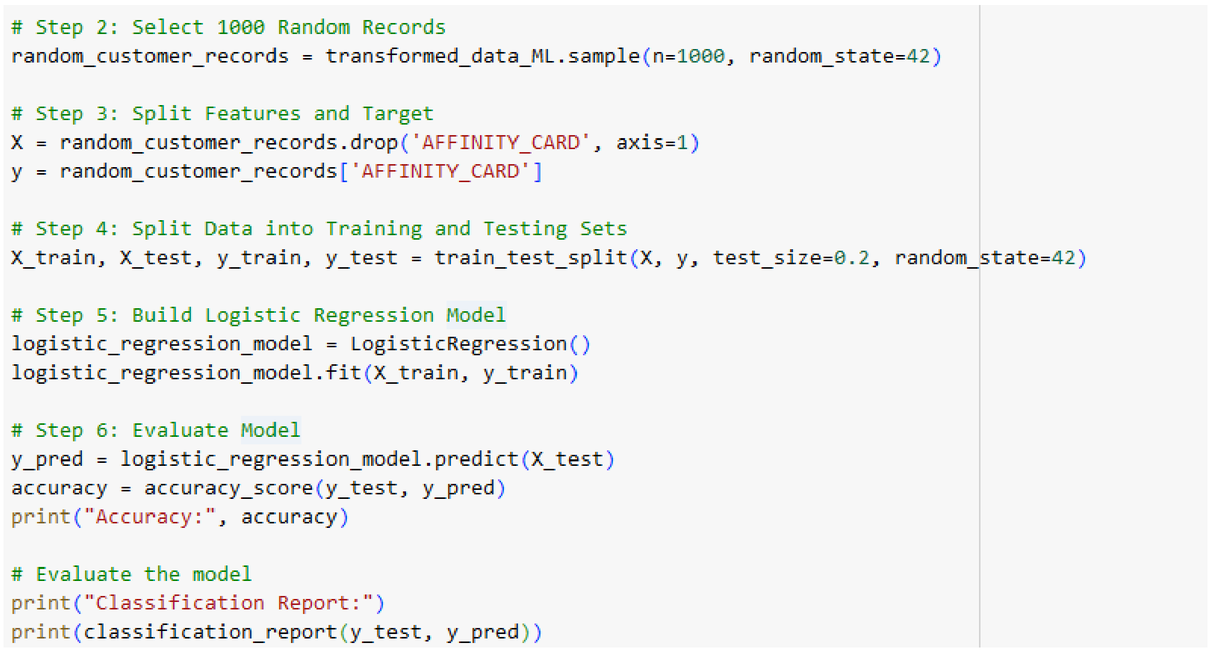
Figures:

Figure 32 shows the data transformation steps, including loading the dataset, discarding redundant columns, handling missing values, and encoding categorical variables. In Figure 33, the logistic regression model building process is illustrated, which includes random sample selection, data partitioning, model training, and analysis. Finally, Figure 34 presents the model results, showing the achieved accuracy and a concise classification report that provides insights into the prediction performance of the model.



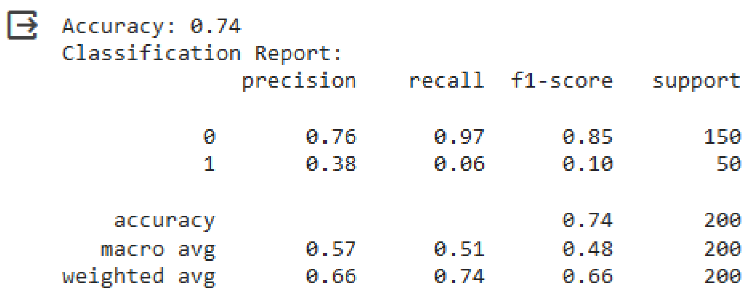
```
[216] # Step 1: Load data
      data = pd.read_csv('Marketing Campaign data.csv')

      # Step 2: Drop 'COMMENTS' column
      data.drop(columns=['COMMENTS'], inplace=True)

      # Step 3: Check for missing values
      missing_values = data.isnull().sum()
      if missing_values.sum() > 0:
          print("Missing Values:\n", missing_values)
      else:
          print("No missing values found.")

      # Step 4: Encode categorical variables using one-hot encoding
      transformed_data_ML = pd.get_dummies(data)

      # Step 5: Save the transformed data to a new file
      transformed_data_ML.to_csv('Transformed_Marketing_Data.csv', index=False)

      # Step 6: Check the shape of the transformed dataset
      print("Shape of Transformed Data:", transformed_data_ML.shape)
```

Figure 32: Data Transformation Steps



```
# Step 2: Select 1000 Random Records
random_customer_records = transformed_data_ML.sample(n=1000, random_state=42)

# Step 3: Split Features and Target
X = random_customer_records.drop('AFFINITY_CARD', axis=1)
y = random_customer_records['AFFINITY_CARD']

# Step 4: Split 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)

# Step 5: Build Logistic Regression Model
logistic_regression_model = LogisticRegression()
logistic_regression_model.fit(X_train, y_train)

# Step 6: Evaluate Model
y_pred = logistic_regression_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Figure 33: Logistic Regression Model Building Steps



```
Accuracy: 0.74
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.97      0.85       150
           1       0.38      0.06      0.10        50

    accuracy                           0.74       200
   macro avg       0.57      0.51      0.48       200
weighted avg       0.66      0.74      0.66       200
```

Figure 34: Output of Logistic Regression Model

Output Description:
The information presented concerns the analytical metrics of the logistic regression model used to predict the target variable 'AFFINITY_CARD'. Accuracy, indicating the proportion of true positive predictions among all positive predictions, has been reported as 0.88 for class 0 and 0.68 for class 1. This means that 88% of the samples predicted as class 0 are indeed class 0, while 68% of the samples predicted as class 1 are actually class 1. Recall, representing the fraction of truly positive predictions among all truly positive cases, is 0.90 for class 0 and 0.64 for class 1. Thus, 90% of true class 0 samples have been correctly predicted, while only 64% of true class 1 samples have been detected correctly. The F1-score, which is the harmonic mean of precision and recall, indicates that the two metrics are balanced, with values of 0.89 for class 0 and 0.66 for class 1. These scores indicate the overall effectiveness of the model across samples dividing into their groups. The macro and weighted averages of these metrics provide a detailed analysis of the model's performance across classes, with an overall accuracy of 0.74, indicating that 74% of the predictions are accurate. Despite the high accuracy obtained for class 0, the model exhibits the worst performance against class 1 predictions, highlighting the need for further improvements in identifying ideal class models.

5.2 Implement Prediction Application

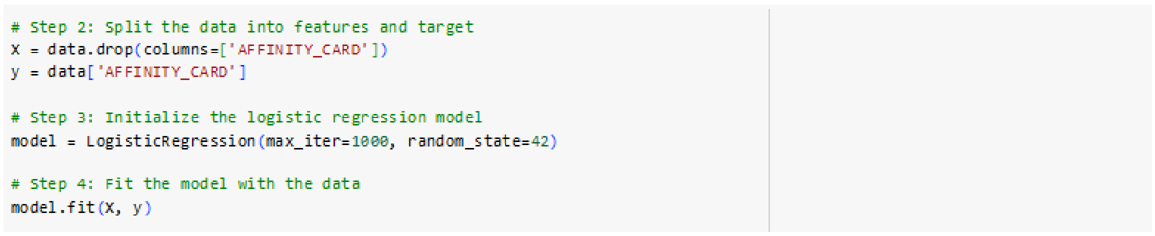
In this section, we will use a prediction function based on the logistic regression ML model developed earlier. This application will allow users to insert a customer record through keyboard input or by reading from a file and get predicted responses about customer purchases. Below were the steps performed:

1. Initially, we load the transformed data containing customer records and separate them into features and target variables.
2. We initialize the logistic regression model and fit it to the data.
3. We describe the function of retrieving user input for a customer record, whether it is keyboard input or a file.

4. The input data is pre-processed to match the format expected by the model, ensuring consistency with the training features.
5. We define a function to make predictions using models trained based on input data.
6. The main program drives the forecasting application, providing users with options to enter data, create forecasts, and continue or exit the application.

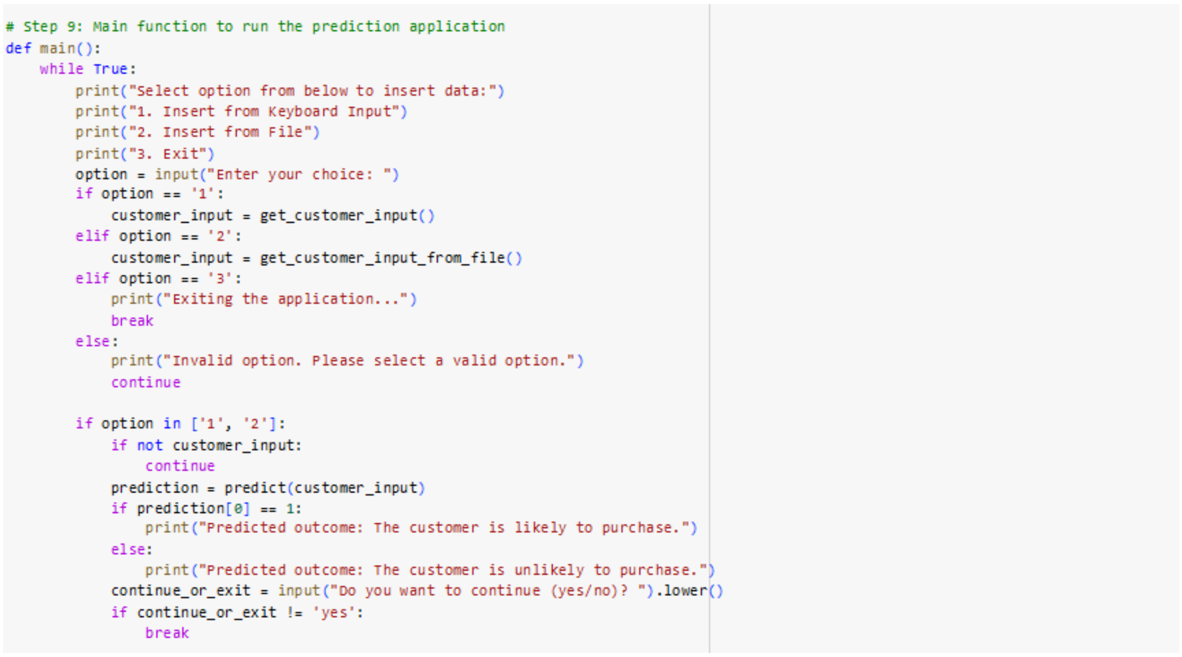
Figures:

Figure 35 illustrates the steps of item selection and sampling, including randomly selecting customer records, partitioning data, and training the logistic regression model. Figure 36 shows the code for the basic functionality to run the forecasting application. Figure 37 illustrates the user interface, allowing users to select options for data entry. Figures 38 and 39 show the test results for option 1 (entering input via the keyboard) and option 2 (entering input from a file), respectively, showing the predicted results based on the customer data provided.



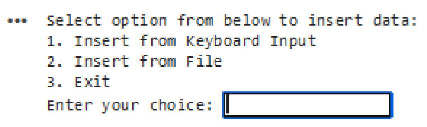
```
# Step 2: Split the data into features and target
X = data.drop(columns=['AFFINITY_CARD'])
y = data['AFFINITY_CARD']

# Step 3: Initialize the logistic regression model
model = LogisticRegression(max_iter=1000, random_state=42)

# Step 4: Fit the model with the data
model.fit(X, y)
```

Figure 35: Feature Selection and Modeling Steps



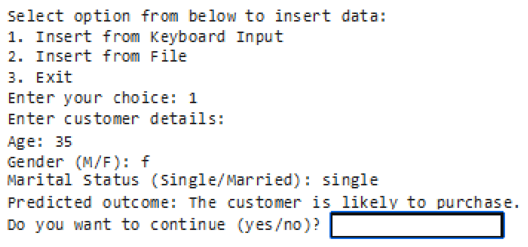
```
# Step 9: Main function to run the prediction application
def main():
    while True:
        print("Select option from below to insert data:")
        print("1. Insert from Keyboard Input")
        print("2. Insert from File")
        print("3. Exit")
        option = input("Enter your choice: ")
        if option == '1':
            customer_input = get_customer_input()
        elif option == '2':
            customer_input = get_customer_input_from_file()
        elif option == '3':
            print("Exiting the application...")
            break
        else:
            print("Invalid option. Please select a valid option.")
            continue

        if option in ['1', '2']:
            if not customer_input:
                continue
            prediction = predict(customer_input)
            if prediction[0] == 1:
                print("Predicted outcome: The customer is likely to purchase.")
            else:
                print("Predicted outcome: The customer is unlikely to purchase.")
            continue_or_exit = input("Do you want to continue (yes/no)? ").lower()
            if continue_or_exit != 'yes':
                break
```

Figure 36: Main Function for Prediction Application



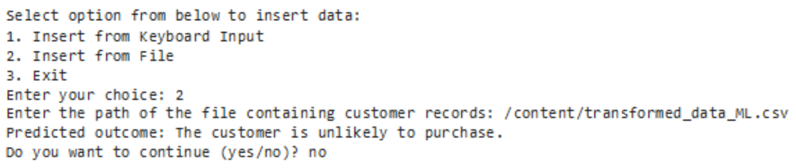
```
•••  Select option from below to insert data:
     1. Insert from Keyboard Input
     2. Insert from File
     3. Exit
     Enter your choice:
```

Figure 37: User Interface of Prediction Application



```
Select option from below to insert data:
1. Insert from Keyboard Input
2. Insert from File
3. Exit
Enter your choice: 1
Enter customer details:
Age: 35
Gender (M/F): f
Marital Status (Single/Married): single
Predicted outcome: The customer is likely to purchase.
Do you want to continue (yes/no)?
```

Figure 38: Keyboard Input Testing Output



```
Select option from below to insert data:
1. Insert from Keyboard Input
2. Insert from File
3. Exit
Enter your choice: 2
Enter the path of the file containing customer records: /content/transformed_data_ML.csv
Predicted outcome: The customer is unlikely to purchase.
Do you want to continue (yes/no)? no
```

Figure 39: File Input Testing Output

Output Description:
In the first test, the user chooses to enter data via keyboard input and enters customer information (age, sex, marital status). The application predicts the likelihood that the customer will purchase the AFFINITY_CARD. In the second test, the user chooses to put data in a file, and given the file path containing a customer record, the application predicts that the customer is unlikely to buy it. Both cases demonstrate how predictive applications work, providing users with predicted results based on customer data.

5.3 Test Model Accuracy

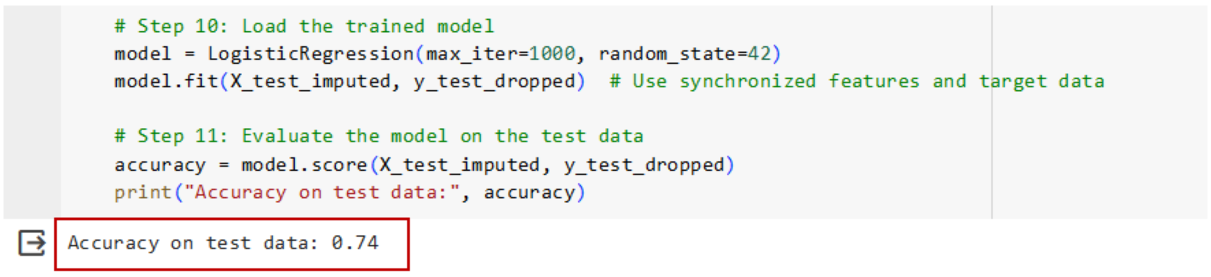
In this task, we use 100 unique customer records from the rest of the dataset to test the accuracy of the prediction function. Below are the steps performed:

1. **Load Transformed Data:** The transformed data obtained from the machine learning model are loaded. This dataset has been preprocessed and transformed for model training.
2. **Define Sample Size:** A sample size of 100 is set for testing purposes. This number determines how many customer records will be used for analysis.
3. **Extract Test Data:** From the remaining dataset, 100 customer records are specifically extracted to create the test dataset. This subset is used to evaluate the prediction accuracy of the application.

4. **Split Data:** The sample data is split into two separate parts: features and target. Features represent characteristics or predictions, while the value specifies the variables to be predicted.
5. **Impute Missing Values:** In the features data, any missing values are handled using the mean imputation strategy. This ensures that the dataset is complete and suitable for model evaluation.
6. **Align Indices:** To ensure the alignment of the matching data, the indices of the features and target data are aligned. This system makes it easier to analyze forecasts accurately and compare them to actual results.
7. **Load Trained Model:** The logistic regression model previously trained on the training data is loaded for analysis. This model is already trained to identify target changes based on the given resources.
8. **Evaluate Model:** The loaded model is applied to the test data, and its prediction accuracy is calculated. By comparing the predictions of the model with the actual results in the test dataset, an accuracy score is determined, which indicates the performance of the model in predicting the target variable.

Figures:

Below Figure 40 Illustrates the modeling step and the outcome of model accuracy.



```
# Step 10: Load the trained model
model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_test_imputed, y_test_dropped)  # Use synchronized features and target data

# Step 11: Evaluate the model on the test data
accuracy = model.score(X_test_imputed, y_test_dropped)
print("Accuracy on test data:", accuracy)
```

```
Accuracy on test data: 0.74
```

Figure 40: Evaluation of Prediction Application Accuracy

Output Description:

The logistic regression model achieves an accuracy of 0.74 on the test data, indicating that it correctly predicts the target variable, 'AFFINITY_CARD', for 74% of the test samples.

6. Discussion and reflection of the work

The project involved a thorough analysis of a marketing campaign dataset from a marketing company, with the aim of preparing the data for subsequent analytics and data mining tasks. We started by gaining a detailed understanding of the dataset by creating a metadata table, detailing the characteristics of each attribute. This involved summarizing statistical data using statistics such as mean, minimum, maximum, and standard deviation, as well as creating histograms to visualize data distribution. For nominal data, mode and bar charts were used to highlight attribute frequencies.

Identifying missing data or errors was another important step in the data understanding phase. We carefully explained missing data, including nulls, blanks, and unknown values, as well as errors such as incorrect or inconsistent information. Recommendations on how to deal with missing or error data were provided, emphasizing the importance of appropriate data cleansing procedures without actually flushing them at this stage. During the data preparation phase, Python programs were developed to isolate variables that had no effect on the target variable and to clean the data by removing records with few missing values or errors.

In addition, variables were implemented in some variables using both core Python code and Pandas library methods, and the differences between the two methods presented for each variable were discussed.

Further analysis were performed through a Python program to determine summary statistics for all variables, including metrics such as sum, mean, standard deviation, skewness, and kurtosis. Data analysis was simplified with a new Python program that allows users to generate histogram plots of selected variables at runtime, enhancing the capabilities of interactive data analysis.

The data mining part involves building a logistic regression machine learning model using 1000 random customer records from the transformed dataset. An interactive prediction service based on this model was implemented so that users could input customer records and get predicted results. Finally, the accuracy of the application was tested with 100 customer records from the remaining dataset, providing insight into the effectiveness of the predictive model in real-world situations.

Reflecting on the project, we gained valuable experience in various aspects of data analysis, from data understanding and preparation to analysis and mining. The repetitive nature of the project allowed for continuous revision and improvement and provided a deeper understanding of the data and methods used. Moving forward, the insights gained from this work will inform future efforts in data science, serving as a foundation for addressing complex challenges and driving informed decision-making.

7. References

[1] B. M. N. A. D. John D. Kelleher, Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies, Cambridge: MIT Press, 2015.

[2] J. W. Tukey, Exploratory Data Analysis, Addison-Wesley, 1977.

[3] J. H. a. M. Kamber, "Data Mining: Concepts and Techniques," Morgan Kaufmann Publishers, 2006. [Online]. Available: https://www.amazon.com/Data-Mining-Concepts-Techniques-Management/dp/0123814790. [Accessed 4 05 2023].

8. Appendix

Please refer to the below link address for detailed code implementation.

CC7182 - Link to Google Colab File