ML Data Cleaning and Feature Selection

Name: Venkata Sairam Mandapati

NUID: 002768738

In this assignment, you will use a dataset for predictive learning and check the quality of the data and determine which features are important.

##Answer the following questions:

What are the data types? (Only numeric and categorical)

Are there missing values?

What are the likely distributions of the numeric variables?

Which independent variables are useful to predict a target (dependent variable)? (Use at least three methods)

Which independent variables have missing data? How much?

Do the training and test sets have the same data?

In the predictor variables independent of all the other predictor variables?

Which predictor variables are the most important?

Do the ranges of the predictor variables make sense?

What are the distributions of the predictor variables?

Remove outliers and keep outliers (does if have an effect of the final predictive model)?

Remove 1%, 5%, and 10% of your data randomly and impute the values back using at least 3 imputation methods. How well did the methods recover the missing values? That is remove some data, check the % error on residuals for numeric data and check for bias and variance of the error.

For categorical data, calculate the accuracy and a confusion matrix.

##Are my answers supported with data? Tables, graphs, and charts must support your evaluation/answers.

It MUST run in Google Collab. You will also save the Google Collab notebook as a .ipynb notebook and upload that to Canvas . (5 Points)

Public dataset (5 Points) Pick a public dataset that can be used for Regression or Classification. You MUST get approval for your dataset from the TAs.

What code is yours and what have you adapted? (5 Points) You must explain what code you wrote and what you have done that is different. Failure to cite ANY code will result in a zero for this section.

Did I explain my code clearly? (15 Points) Your code review score will be scaled to a range of 0 to 10 and be used for this score.

Did I explain my licensing clearly? (5 Points) Failure to cite a clear license will result in a zero for this section.

##Answers to listed questions

Which independent variables are useful to predict a target (dependent variable)?

Which independent variable have missing data? How much?

Do the training and test sets have the same data?

In the predictor variables independent of all the other predictor variables?

Which predictor variables are the most important?

Do the ranges of the predictor variables make sense?

What are the distributions of the predictor variables?

Notes:

Normality - When we talk about normality what we mean is that the data should look like a normal distribution. This is important because several statistic tests rely on this (e.g. t-statistics). In this exercise we'll just check univariate normality for 'SalePrice' (which is a limited approach). Remember that univariate normality doesn't ensure multivariate normality (which is what we would like to have), but it helps. Another detail to take into account is that in big samples (>200 observations) normality is not such an issue. However, if we solve normality, we avoid a lot of other problems (e.g. heteroscedacity) so that's the main reason why we are doing this analysis.

Homoscedasticity - I just hope I wrote it right. Homoscedasticity refers to the 'assumption that dependent variable(s) exhibit equal levels of variance across the range of predictor variable(s)' (Hair et al., 2013) (Links to an external site.). Homoscedasticity is desirable because we want the error term to be the same across all values of the independent variables.

Linearity- The most common way to assess linearity is to examine scatter plots and search for linear patterns. If patterns are not linear, it would be worthwhile to explore data transformations. However, we'll not get into this because most of the scatter plots we've seen appear to have linear relationships.

Absence of correlated errors - Correlated errors, like the definition suggests, happen when one error is correlated to another. For instance, if one positive error makes a negative error systematically, it means that there's a relationship between these variables. This occurs often in time series, where some patterns are time related. We'll also not get into this. However, if you detect something, try to add a variable that can explain the effect you're getting. That's the most common solution for correlated errors.

Abstract

The market sales data analysis revolves around customer-specific details encompassing demographics such as age, income, and shopping behavior, including factors like spending amounts and website visits. The central objective is to determine whether a customer will respond positively to a marketing campaign. This analytical journey begins with comprehensive data exploration techniques, including correlation heatmaps, boxplots, and Q-Q plots, which unveil relationships, outliers, and data distribution patterns. Subsequently, a logistic regression model is meticulously constructed for predicting customer responses. However, it becomes evident that not all columns hold significant importance in predicting the dependent variable.

To address data gaps caused by missing values, a variety of imputation methods are rigorously evaluated. These methods include mean imputation, which replaces missing values with the feature's mean, median imputation that uses the median of observed data, and regression imputation, where missing values are predicted based on related features and data patterns.

This comprehensive approach ensures that the model is equipped to handle real-world data complexities, and the selection of the most effective imputation strategy is paramount to maintaining data integrity. Ultimately, this process empowers businesses to make informed marketing decisions, leveraging insights from customer data analysis to optimize campaign strategies and maximize positive responses.

About Dataset - Marketing Campaign

Context

The Marketing Campaign dataset is designed to aid businesses in enhancing the efficiency of their marketing efforts. It serves as a valuable resource for predictive modeling and data-driven decision-making. By analyzing this dataset, companies can gain insights into customer behavior and preferences, enabling them to predict which individuals are likely to respond positively to marketing offers. This predictive capability allows for targeted and personalized marketing campaigns, ultimately increasing response rates and reducing marketing expenses. In essence, the dataset empowers businesses to optimize their marketing strategies and allocate resources effectively, leading to improved campaign outcomes in terms of customer engagement and conversions.

Content

1. AcceptedCmp1 - 1 if customer accepted the offer in the 1st campaign, 0 otherwise
2. AcceptedCmp2 - 1 if customer accepted the offer in the 2nd campaign, 0 otherwise
3. AcceptedCmp3 - 1 if customer accepted the offer in the 3rd campaign, 0 otherwise
4. AcceptedCmp4 - 1 if customer accepted the offer in the 4th campaign, 0 otherwise
5. AcceptedCmp5 - 1 if customer accepted the offer in the 5th campaign, 0 otherwise
6. Response (target) - 1 if customer accepted the offer in the last campaign, 0 otherwise
7. Complain - 1 if customer complained in the last 2 years
8. DtCustomer - date of customer’s enrolment with the company
9. Education - customer’s level of education
10. Marital - customer’s marital status

11. Kidhome - number of small children in customer's household
12. Teenhome - number of teenagers in customer's household
13. Income - customer's yearly household income
14. MntFishProducts - amount spent on fish products in the last 2 years
15. MntMeatProducts - amount spent on meat products in the last 2 years
16. MntFruits - amount spent on fruits products in the last 2 years
17. MntSweetProducts - amount spent on sweet products in the last 2 years
18. MntWines - amount spent on wine products in the last 2 years
19. MntGoldProds - amount spent on gold products in the last 2 years
20. NumDealsPurchases - number of purchases made with discount
21. NumCatalogPurchases - number of purchases made using catalogue
22. NumStorePurchases - number of purchases made directly in stores
23. NumWebPurchases - number of purchases made through company's web site
24. NumWebVisitsMonth - number of visits to company's web site in the last month
25. Recency - number of days since the last purchase

This dataset is rich with information relevant to customer behavior and responses to marketing campaigns. It appears suitable for building predictive models to understand which factors influence customer responses to different campaigns and, ultimately, to optimize future marketing strategies. The target variable "Response" can be used to train and evaluate predictive models, while the other variables offer valuable features for analysis and modeling. Data preprocessing and statistical analysis can provide insights into customer behavior patterns and campaign effectiveness.

```
!pip install eli5
# !pip install seaborn
```

```
Requirement already satisfied: eli5 in /usr/local/lib/python3.10/dist-packages (0.13.0)
Requirement already satisfied: attrs>17.1.0 in /usr/local/lib/python3.10/dist-packages (from eli5) (23.1.0)
Requirement already satisfied: jinja2>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from eli5) (3.1.2)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from eli5) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from eli5) (1.11.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from eli5) (1.16.0)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.10/dist-packages (from eli5) (1.2.2)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from eli5) (0.20.1)
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.10/dist-packages (from eli5) (0.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2>=3.0.0->eli5)
```

```
(2.1.3)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20-
>eli5) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20-
>eli5) (3.2.0)
```

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pylab as plt
from matplotlib import pyplot
```

Data Types

```python
#Reading the Marketting campaign Dataset
data =
pd.read_csv("https://raw.githubusercontent.com/VenkataSairamMandapati/
ML-Data-Cleaning-and-Feature-Selection/main/MARKETING_CAMPAIGN.csv",
sep=";")
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   2240 non-null   int64
 1   Year_Birth           2240 non-null   int64
 2   Education            2240 non-null   object
 3   Marital_Status       2240 non-null   object
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64
 6   Teenhome             2240 non-null   int64
 7   Dt_Customer          2240 non-null   object
 8   Recency              2240 non-null   int64
 9   MntWines             2240 non-null   int64
 10  MntFruits            2240 non-null   int64
 11  MntMeatProducts      2240 non-null   int64
 12  MntFishProducts      2240 non-null   int64
 13  MntSweetProducts     2240 non-null   int64
 14  MntGoldProds         2240 non-null   int64
 15  NumDealsPurchases    2240 non-null   int64
 16  NumWebPurchases      2240 non-null   int64
 17  NumCatalogPurchases  2240 non-null   int64
 18  NumStorePurchases    2240 non-null   int64
 19  NumWebVisitsMonth    2240 non-null   int64
```

```
 20  AcceptedCmp3         2240 non-null   int64
 21  AcceptedCmp4         2240 non-null   int64
 22  AcceptedCmp5         2240 non-null   int64
 23  AcceptedCmp1         2240 non-null   int64
 24  AcceptedCmp2         2240 non-null   int64
 25  Complain             2240 non-null   int64
 26  Z_CostContact        2240 non-null   int64
 27  Z_Revenue            2240 non-null   int64
 28  Response             2240 non-null   int64
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB
```

Using data.info() we have the following information-

1. 25 integer data types,
2. 3 categorical data type
3. 1 floating point data type

```
#pandas by default only displays 20 columns max, to view all 28
columns in output we set max_columns to None
pd.options.display.max_columns = None
data.head()

     ID  Year_Birth   Education Marital_Status   Income  Kidhome
Teenhome  \
0  5524        1957  Graduation         Single  58138.0        0
0
1  2174        1954  Graduation         Single  46344.0        1
1
2  4141        1965  Graduation       Together  71613.0        0
0
3  6182        1984  Graduation       Together  26646.0        1
0
4  5324        1981         PhD        Married  58293.0        1
0

  Dt_Customer  Recency  MntWines  MntFruits  MntMeatProducts
MntFishProducts  \
0  2012-09-04       58       635         88              546
172
1  2014-03-08       38        11          1                6
2
2  2013-08-21       26       426         49              127
111
3  2014-02-10       26        11          4               20
10
4  2014-01-19       94       173         43              118
46

   MntSweetProducts  MntGoldProds  NumDealsPurchases  NumWebPurchases
```

```
\
0                 88            88                  3                8

1                  1             6                  2                1

2                 21            42                  1                8

3                  3             5                  2                2

4                 27            15                  5                5


   NumCatalogPurchases  NumStorePurchases  NumWebVisitsMonth
AcceptedCmp3  \
0                   10                  4                  7
0
1                    1                  2                  5
0
2                    2                 10                  4
0
3                    0                  4                  6
0
4                    3                  6                  5
0

   AcceptedCmp4  AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  Complain  \
0             0             0             0             0         0
1             0             0             0             0         0
2             0             0             0             0         0
3             0             0             0             0         0
4             0             0             0             0         0

   Z_CostContact  Z_Revenue  Response
0              3         11         1
1              3         11         0
2              3         11         0
3              3         11         0
4              3         11         0
```

```python
data[["Education", "Marital_Status","Dt_Customer"]].describe()
```

```
         Education Marital_Status Dt_Customer
count         2240           2240        2240
unique           5              8         663
top     Graduation        Married  2012-08-31
freq          1127            864          12
```

```python
data[["Z_CostContact", "Z_Revenue"]].describe()
```

```
       Z_CostContact  Z_Revenue
count         2240.0     2240.0
```

```
mean              3.0      11.0
std               0.0       0.0
min               3.0      11.0
25%               3.0      11.0
50%               3.0      11.0
75%               3.0      11.0
max               3.0      11.0
```

Getting a first look at the raw data lets us understand some nature of the data.

1. Dt_customer represents start date of customer journey, he can be converted to numeric type "customer since" to indicate number of days months or year a customer has been a part
2. AcceptedCmp variables have 1/0 representing if customer accepted campaign offer before hence they can be treated as categorical yes/no
3. Response is our target prediction variable
4. We have no description of Z_CostContact Z_Revenue variables in data description. Describing them rreveals they are filled with 11s and 3s and have no real significance hence we will be dropping those for this analysis.

Considered Data Types:-

Column Data type (categorical/numeric)

Year_Birth int64 numeric

Education object categorical

Marital_Status object categorical

Income float64 numeric

Kidhome int64 numeric

Teenhome int64 numeric

Dt_Customer object categorical

Recency int64 numeric

MntWines int64 numeric

MntFruits int64 numeric

MntMeatProducts int64 numeric

MntFishProducts int64 numeric

MntSweetProducts int64 numeric

MntGoldProds int64 numeric

NumDealsPurchases int64 numeric

NumWebPurchases int64 numeric

NumCatalogPurchases int64 numeric

NumStorePurchases int64 numeric

NumWebVisitsMonth int64 numeric

AcceptedCmp3 int64 categorical

AcceptedCmp4 int64 categorical

AcceptedCmp5 int64 categorical

AcceptedCmp1 int64 categorical

AcceptedCmp2 int64 categorical

Complain int64 categorical

Response int64 categorical

```
#Dropping columns
data.drop(columns = ["Z_CostContact","Z_Revenue","ID",],inplace =
True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 26 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Year_Birth           2240 non-null   int64
 1   Education            2240 non-null   object
 2   Marital_Status       2240 non-null   object
 3   Income               2216 non-null   float64
 4   Kidhome              2240 non-null   int64
 5   Teenhome             2240 non-null   int64
 6   Dt_Customer          2240 non-null   object
 7   Recency              2240 non-null   int64
 8   MntWines             2240 non-null   int64
 9   MntFruits            2240 non-null   int64
 10  MntMeatProducts      2240 non-null   int64
 11  MntFishProducts      2240 non-null   int64
 12  MntSweetProducts     2240 non-null   int64
 13  MntGoldProds         2240 non-null   int64
 14  NumDealsPurchases    2240 non-null   int64
 15  NumWebPurchases      2240 non-null   int64
 16  NumCatalogPurchases  2240 non-null   int64
 17  NumStorePurchases    2240 non-null   int64
 18  NumWebVisitsMonth    2240 non-null   int64
```

```
 19  AcceptedCmp3         2240 non-null   int64
 20  AcceptedCmp4         2240 non-null   int64
 21  AcceptedCmp5         2240 non-null   int64
 22  AcceptedCmp1         2240 non-null   int64
 23  AcceptedCmp2         2240 non-null   int64
 24  Complain             2240 non-null   int64
 25  Response             2240 non-null   int64
dtypes: float64(1), int64(22), object(3)
memory usage: 455.1+ KB
```

Null Values

```
#checking if the any data is missing
percent_missing = data.isnull().sum() * 100 / len(data)
null_values_total = data.isnull().sum()
missing_value_df = pd.DataFrame({
                                 'Missing_Total' : null_values_total,
                                 'percent_missing': percent_missing,
                                 })
missing_value_df
```

	Missing_Total	percent_missing
Year_Birth	0	0.000000
Education	0	0.000000
Marital_Status	0	0.000000
Income	24	1.071429
Kidhome	0	0.000000
Teenhome	0	0.000000
Dt_Customer	0	0.000000
Recency	0	0.000000
MntWines	0	0.000000
MntFruits	0	0.000000
MntMeatProducts	0	0.000000
MntFishProducts	0	0.000000
MntSweetProducts	0	0.000000
MntGoldProds	0	0.000000
NumDealsPurchases	0	0.000000
NumWebPurchases	0	0.000000
NumCatalogPurchases	0	0.000000
NumStorePurchases	0	0.000000
NumWebVisitsMonth	0	0.000000
AcceptedCmp3	0	0.000000
AcceptedCmp4	0	0.000000
AcceptedCmp5	0	0.000000
AcceptedCmp1	0	0.000000
AcceptedCmp2	0	0.000000
Complain	0	0.000000
Response	0	0.000000

As we can see **"Income"** column has **24 missing values** out of total 2240 rows. Thats **1.07%** of missing values

There are several ways to handle null-values :

1. We can delete the rows containing null-values
2. We can impute the mean value
3. We can input the mean value of a specific population : in this case we would split by Education
4. We can use a model to predict missing values

With our dataset, we will fill missing values of Income by mean of Eduction

```
data['Income'].fillna(data.groupby('Education')
['Income'].transform('mean'), inplace = True)

#checking if the any data is missing
percent_missing = data.isnull().sum() * 100 / len(data)
null_values_total = data.isnull().sum()
missing_value_df = pd.DataFrame({
                                  'Missing_Total' : null_values_total,
                                  'percent_missing': percent_missing,
                                  })
missing_value_df
```

	Missing_Total	percent_missing
Year_Birth	0	0.0
Education	0	0.0
Marital_Status	0	0.0
Income	0	0.0
Kidhome	0	0.0
Teenhome	0	0.0
Dt_Customer	0	0.0
Recency	0	0.0
MntWines	0	0.0
MntFruits	0	0.0
MntMeatProducts	0	0.0
MntFishProducts	0	0.0
MntSweetProducts	0	0.0
MntGoldProds	0	0.0
NumDealsPurchases	0	0.0
NumWebPurchases	0	0.0
NumCatalogPurchases	0	0.0
NumStorePurchases	0	0.0
NumWebVisitsMonth	0	0.0
AcceptedCmp3	0	0.0
AcceptedCmp4	0	0.0
AcceptedCmp5	0	0.0
AcceptedCmp1	0	0.0
AcceptedCmp2	0	0.0

```
Complain                                 0              0.0
Response                                 0              0.0
```

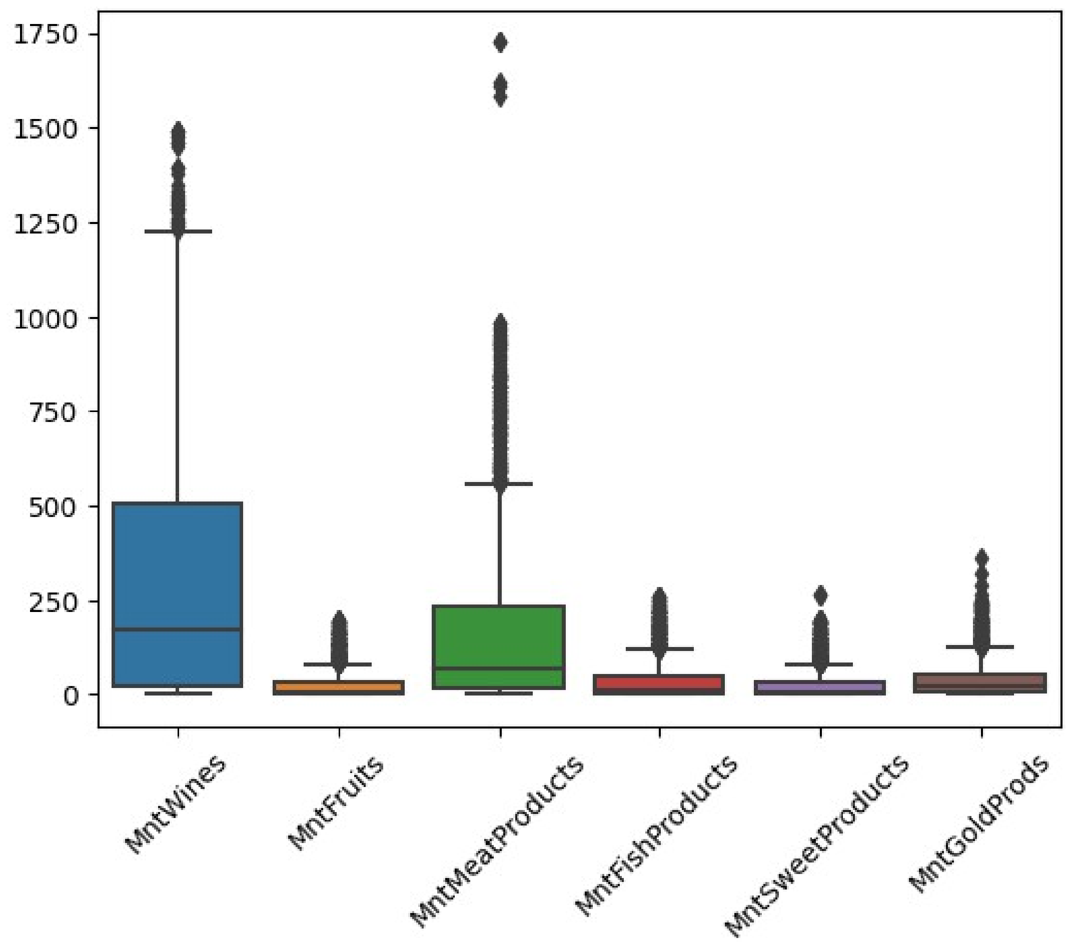
Numeric Data Distribution

```
data.describe()
```

```
        Year_Birth         Income      Kidhome     Teenhome
Recency  \
count  2240.000000    2240.000000  2240.000000  2240.000000
2240.000000
mean   1968.805804   52253.592375     0.444196     0.506250
49.109375
std      11.984069   25039.085601     0.538398     0.544538
28.962453
min    1893.000000    1730.000000     0.000000     0.000000
0.000000
25%    1959.000000   35538.750000     0.000000     0.000000
24.000000
50%    1970.000000   51609.500000     0.000000     0.000000
49.000000
75%    1977.000000   68289.750000     1.000000     1.000000
74.000000
max    1996.000000  666666.000000     2.000000     2.000000
99.000000

          MntWines    MntFruits  MntMeatProducts  MntFishProducts  \
count  2240.000000  2240.000000      2240.000000      2240.000000
mean    303.935714    26.302232       166.950000        37.525446
std     336.597393    39.773434       225.715373        54.628979
min       0.000000     0.000000         0.000000         0.000000
25%      23.750000     1.000000        16.000000         3.000000
50%     173.500000     8.000000        67.000000        12.000000
75%     504.250000    33.000000       232.000000        50.000000
max    1493.000000   199.000000      1725.000000       259.000000

       MntSweetProducts  MntGoldProds  NumDealsPurchases
NumWebPurchases  \
count       2240.000000   2240.000000        2240.000000
2240.000000
mean          27.062946     44.021875           2.325000
4.084821
std           41.280498     52.167439           1.932238
2.778714
min            0.000000      0.000000           0.000000
0.000000
25%            1.000000      9.000000           1.000000
2.000000
```

```
50%              8.000000     24.000000             2.000000
4.000000
75%             33.000000     56.000000             3.000000
6.000000
max            263.000000    362.000000            15.000000
27.000000

       NumCatalogPurchases  NumStorePurchases  NumWebVisitsMonth  \
count          2240.000000        2240.000000        2240.000000
mean              2.662054           5.790179           5.316518
std               2.923101           3.250958           2.426645
min               0.000000           0.000000           0.000000
25%               0.000000           3.000000           3.000000
50%               2.000000           5.000000           6.000000
75%               4.000000           8.000000           7.000000
max              28.000000          13.000000          20.000000

       AcceptedCmp3  AcceptedCmp4  AcceptedCmp5  AcceptedCmp1
AcceptedCmp2  \
count   2240.000000   2240.000000   2240.000000   2240.000000
2240.000000
mean       0.072768      0.074554      0.072768      0.064286
0.013393
std        0.259813      0.262728      0.259813      0.245316
0.114976
min        0.000000      0.000000      0.000000      0.000000
0.000000
25%        0.000000      0.000000      0.000000      0.000000
0.000000
50%        0.000000      0.000000      0.000000      0.000000
0.000000
75%        0.000000      0.000000      0.000000      0.000000
0.000000
max        1.000000      1.000000      1.000000      1.000000
1.000000

          Complain     Response
count  2240.000000  2240.000000
mean      0.009375     0.149107
std       0.096391     0.356274
min       0.000000     0.000000
25%       0.000000     0.000000
50%       0.000000     0.000000
75%       0.000000     0.000000
max       1.000000     1.000000

data_f=data[["MntWines",   "MntFruits"     ,"MntMeatProducts",   "MntF
ishProducts","MntSweetProducts", "MntGoldProds"]]
x = sns.boxplot(data=data_f)
x.set_xticklabels(x.get_xticklabels(),rotation=45)
```

```
[Text(0, 0, 'MntWines'),
 Text(1, 0, 'MntFruits'),
 Text(2, 0, 'MntMeatProducts'),
 Text(3, 0, 'MntFishProducts'),
 Text(4, 0, 'MntSweetProducts'),
 Text(5, 0, 'MntGoldProds')]
```

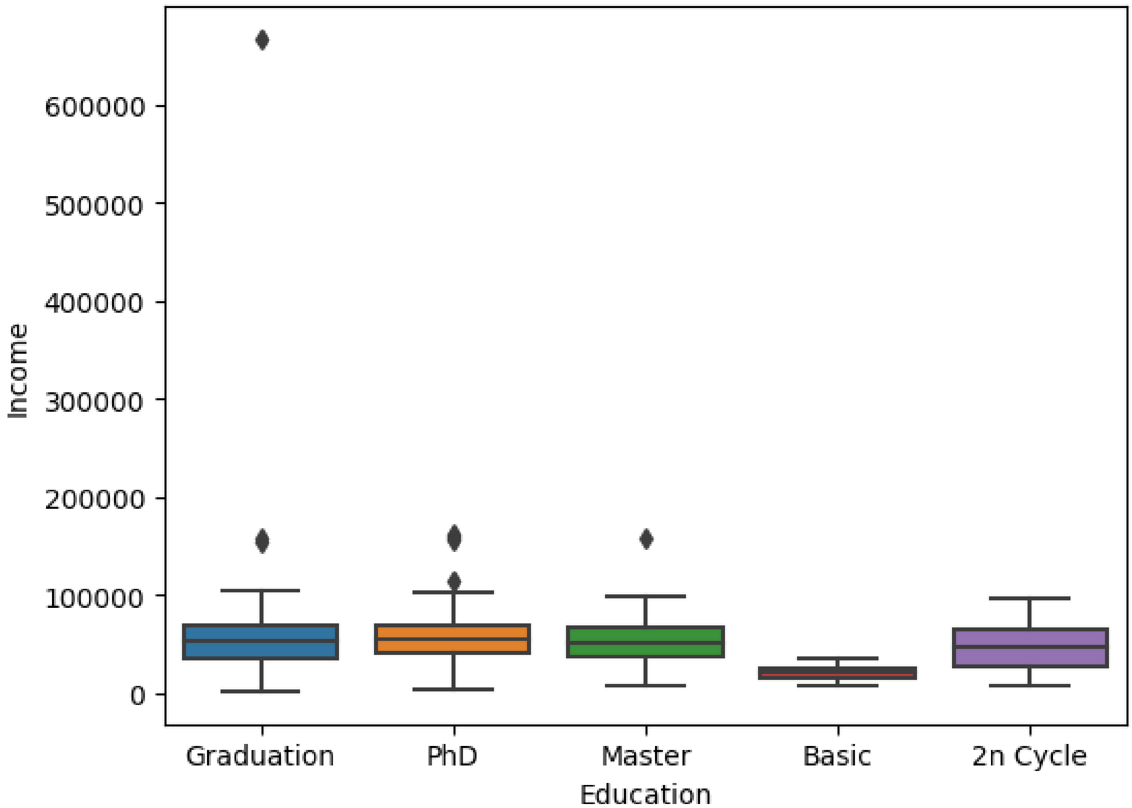


data.describe() gives us an estimate of the numeric distribution of data

1. Average age of customers is 51 with median being 50 with max being 127 which I think is a rare case
2. Average income is 52253 with min and max being 1730 and 666666 indicating presence of outliers
3. Recency ranges between 0 and 99 indicating all customers surveyed were very active in within last 4 months
4. Amount spent on Wines and Meat is the most indicating thats where the majority sales lies
5. Number of items purchased is usually higher in stores
6. Almost all columns in the data need to be normalized as there is a vast difference in range of values of all columns

```python
sns.boxplot( x = 'Education',y = 'Income', data = data)
```

```
<Axes: xlabel='Education', ylabel='Income'>
```



Plotting Income grouped by education shows very less correlation between income and education for all educational categories except Basic which has very low Income.

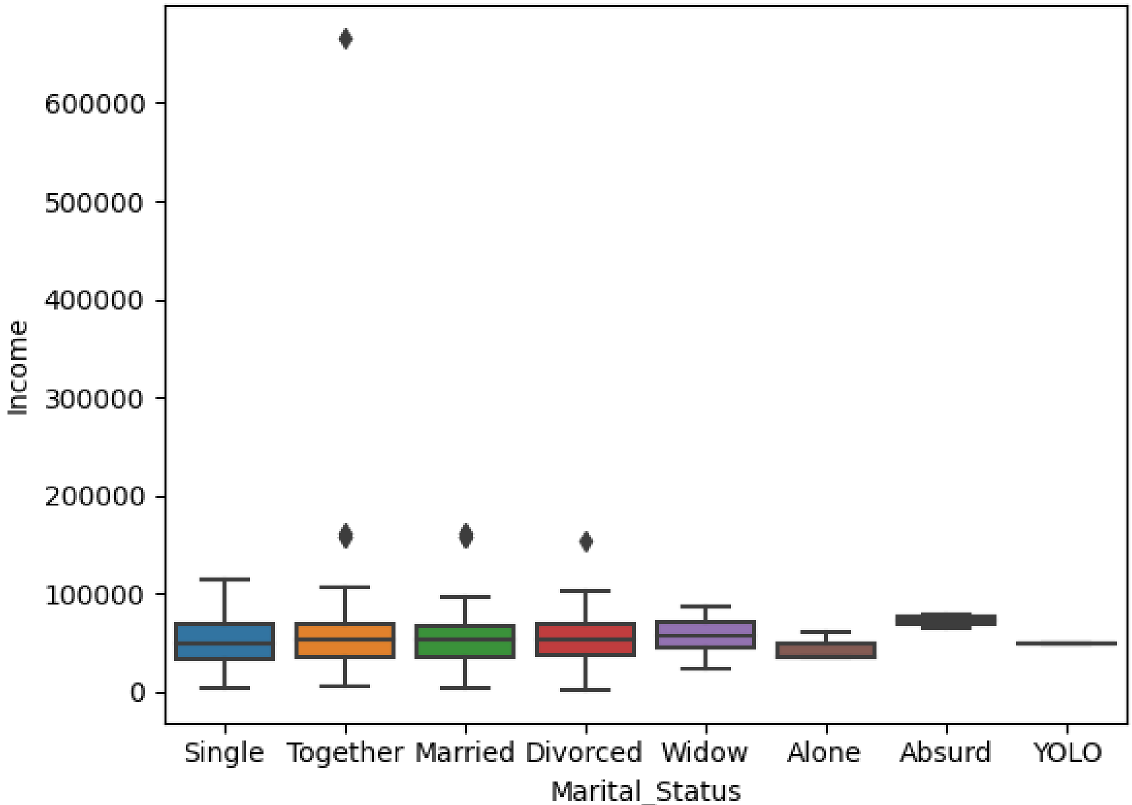
Notice we have ommited outliers to get a better visual representation of the boxplot

```python
print(data['Income'].quantile(0.99))
```

```
94437.68000000001
```

```python
sns.boxplot( x = 'Marital_Status',y = 'Income', data = data)
```

```
<Axes: xlabel='Marital_Status', ylabel='Income'>
```

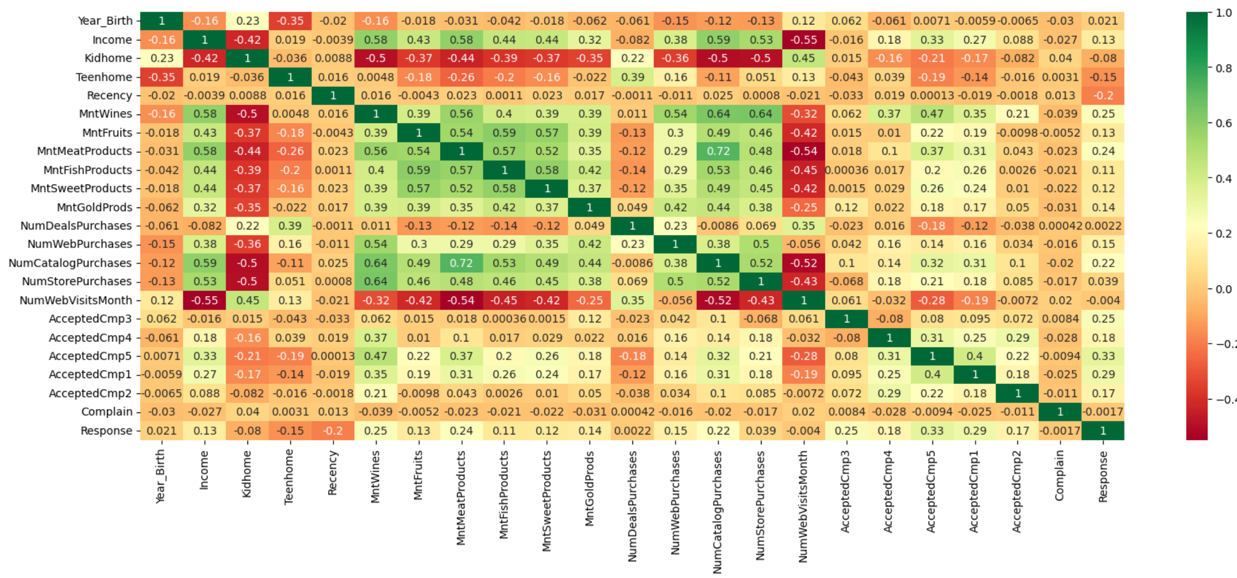


Similarly Income and Marital_status has no correlation with income

Data Transformation

```
#The heat map of the correlation
plt.figure(figsize=(20,7))
sns.heatmap(data.corr(), annot=True, cmap='RdYlGn')
```

```
<ipython-input-802-e4de275c286c>:3: FutureWarning: The default value
of numeric_only in DataFrame.corr is deprecated. In a future version,
it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
  sns.heatmap(data.corr(), annot=True, cmap='RdYlGn')

<Axes: >
```



Above Correlation shows there are very few variables closely related.

1. All Number of purchases are above 0.5 and seem to be closely related
2. Similarly all amount purchased columns are also slightly closely related and can be grouped to denote one feature

Below birth year is converted to age

```
# Converting birth year to age, considering age with respect to year 2020 because the data was last updated 3 years ago
data['Year_Birth'] = data['Year_Birth'].apply(lambda x: 2020-x)
data = data.rename(columns={'Year_Birth': 'Age'})
data[['Age']].head()

   Age
0   63
1   66
2   55
3   36
4   39

data.Education.value_counts()

Graduation    1127
PhD            486
Master         370
2n Cycle       203
Basic           54
Name: Education, dtype: int64
```

Education is Ordinal type variable.

By Doing some research "2nd Cycle" Eduction type usually represents graduate or masters level education in some countries Source

So we transform education in order of education levels

1. Basic
2. Graduation
3. Master / 2nd Cycle
4. PhD

```
Education_map = {'Basic':1,
            'Graduation':2,
            'Master':3,
            '2n Cycle':3,
            'PhD':4}
# Create the mapped values in a new column
data['Education'] = data['Education'].map(Education_map)
# Review dataset
data[['Education']].head()
```

	Education
0	2
1	2
2	2
3	2
4	4

Dt_Customer represents how long a person was a customer, thus we convert the date to how many days it has been since a person became a customer of the store

```
from datetime import datetime
data['Dt_Customer'] = pd.to_datetime(data['Dt_Customer'], format='%Y-%m-%d')
data['Dt_Customer'] = (datetime(2020,1,1) - data['Dt_Customer']).dt.days
data[['Dt_Customer']].describe()
```

	Dt_Customer
count	2240.000000
mean	2365.582143
std	202.122512
min	2012.000000
25%	2192.750000
50%	2367.500000
75%	2541.000000
max	2711.000000

1. Kidhome and Teenhome is combined to total number of children home
2. All amounts are aggregated to amount spent denoting totoal amount spent by customer till now

3. All orders are clubbed to toal number of orders made by the customer
4. Previous campaign responses are clubbed to total number of campaigns accepted before

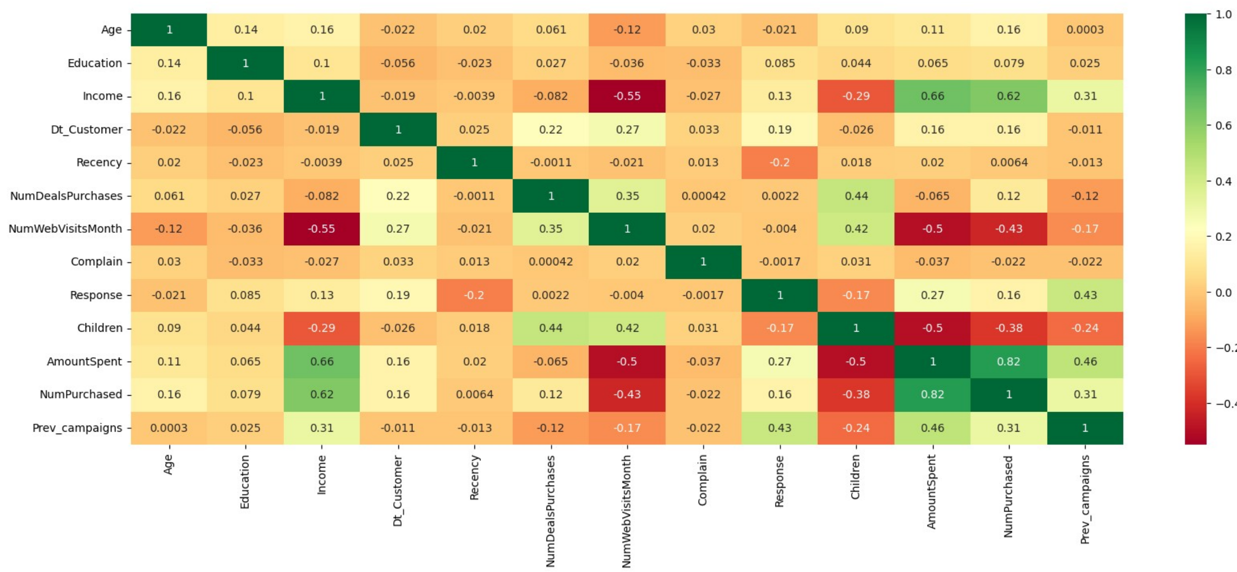
```
data['Children'] = data['Kidhome'] + data['Teenhome']
data.drop(columns = ["Kidhome","Teenhome"],inplace = True)

data['AmountSpent'] = data['MntWines'] + data['MntFruits'] +
data['MntMeatProducts'] + data['MntFishProducts'] +
data['MntSweetProducts'] + data['MntGoldProds']
data.drop(columns = ["MntWines", "MntFruits"      ,"MntMeatProducts",
      "MntFishProducts","MntSweetProducts", "MntGoldProds"],inplace =
True)

data['NumPurchased'] = data['NumWebPurchases'] +
data['NumCatalogPurchases'] + data['NumStorePurchases']
data.drop(columns = ["NumWebPurchases",     "NumCatalogPurchases",
      "NumStorePurchases"],inplace = True)

data['Prev_campaigns'] = data['AcceptedCmp1'] + data['AcceptedCmp2'] +
data['AcceptedCmp3'] + data['AcceptedCmp4'] + data['AcceptedCmp5']
data.drop(columns = ["AcceptedCmp3",  "AcceptedCmp4", "AcceptedCmp5",
      "AcceptedCmp1",  "AcceptedCmp2"],inplace = True)

data.head()
```

```
   Age  Education Marital_Status   Income  Dt_Customer  Recency  \
0   63          2         Single  58138.0         2675       58
1   66          2         Single  46344.0         2125       38
2   55          2       Together  71613.0         2324       26
3   36          2       Together  26646.0         2151       26
4   39          4        Married  58293.0         2173       94

   NumDealsPurchases  NumWebVisitsMonth  Complain  Response  Children
\
0                  3                  7         0         1         0

1                  2                  5         0         0         2

2                  1                  4         0         0         0

3                  2                  6         0         0         1

4                  5                  5         0         0         1


   AmountSpent  NumPurchased  Prev_campaigns
0         1617            22               0
1           27             4               0
2          776            20               0
3           53             6               0
4          422            14               0
```

```
#The heat map of the correlation
plt.figure(figsize=(20,7))
sns.heatmap(data.corr(), annot=True, cmap='RdYlGn')
```

```
<ipython-input-808-e4de275c286c>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(data.corr(), annot=True, cmap='RdYlGn')
```

```
<Axes: >
```



Aggregated data abouve shows a close correlation of amount spent and number of orders

```
data.Marital_Status.value_counts()
```

```
Married     864
Together    580
Single      480
Divorced    232
Widow        77
Alone         3
Absurd        2
YOLO          2
Name: Marital_Status, dtype: int64
```

Here different marital statuses denote the same thing thus we map them to two categories Couple and Single

```
maratial_map = {'Married':"Couple",
            'Together':"Couple",
            'Single':'Single',
```

```
            'Divorced':'Single',
            'Widow':'Single',
            'Alone':'Single',
            'Absurd':'Single',
            'YOLO':'Single'}
# Create the mapped values in a new column
data['Marital_Status'] = data['Marital_Status'].map(maratial_map)
# Review dataset
data[['Marital_Status']].head()
```

```
  Marital_Status
0         Single
1         Single
2         Couple
3         Couple
4         Couple
```

```
dummy_status = pd.get_dummies(data['Marital_Status'],
prefix='Marital_Status')
dummy_status.head()
```

```
   Marital_Status_Couple  Marital_Status_Single
0                      0                      1
1                      0                      1
2                      1                      0
3                      1                      0
4                      1                      0
```

Since maratial_status is a categorical column we create dummy variables

```
data = pd.concat([data, dummy_status], axis=1)
data.drop(['Marital_Status'], axis=1, inplace=True)

data
```

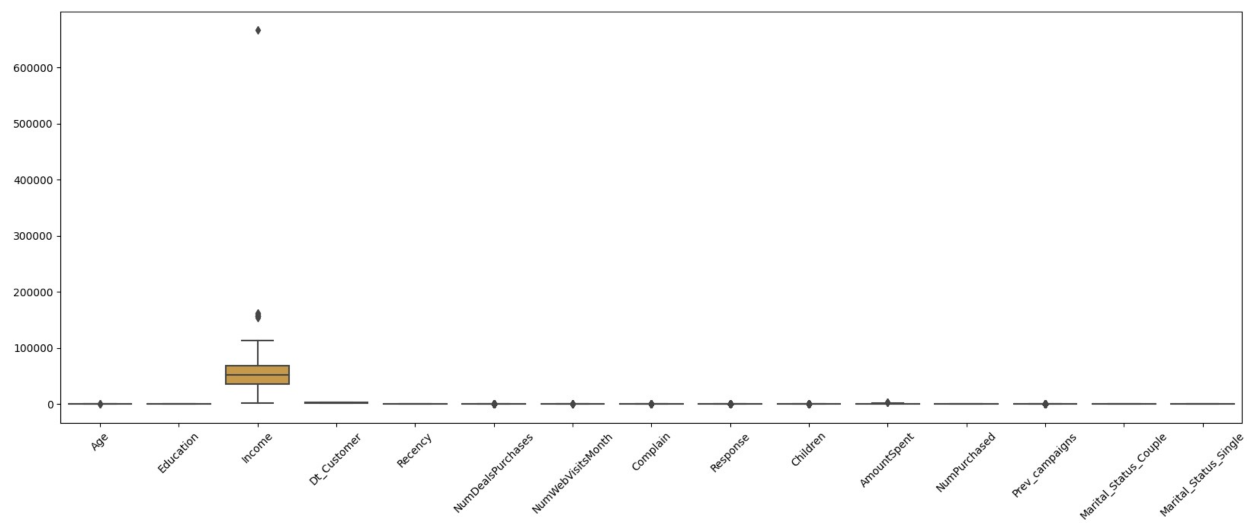
```
      Age  Education   Income  Dt_Customer  Recency  NumDealsPurchases
\
0      63          2  58138.0         2675       58                  3

1      66          2  46344.0         2125       38                  2

2      55          2  71613.0         2324       26                  1

3      36          2  26646.0         2151       26                  2

4      39          4  58293.0         2173       94                  5

...   ...        ...      ...          ...      ...                ...

2235   53          2  61223.0         2393       46                  2
```

```
2236   74          4  64014.0         2031       56                   7

2237   39          2  56981.0         2167       91                   1

2238   64          3  69245.0         2168        8                   2

2239   66          4  52869.0         2634       40                   3


      NumWebVisitsMonth  Complain  Response  Children  AmountSpent  \
0                     7         0         1         0         1617
1                     5         0         0         2           27
2                     4         0         0         0          776
3                     6         0         0         1           53
4                     5         0         0         1          422
...                 ...       ...       ...       ...          ...
2235                  5         0         0         1         1341
2236                  7         0         0         3          444
2237                  6         0         0         0         1241
2238                  3         0         0         1          843
2239                  7         0         1         2          172

      NumPurchased  Prev_campaigns  Marital_Status_Couple  \
0               22               0                      0
1                4               0                      0
2               20               0                      1
3                6               0                      1
4               14               0                      1
...            ...             ...                    ...
2235            16               0                      1
2236            15               1                      1
2237            18               1                      0
2238            21               0                      1
2239             8               0                      1

      Marital_Status_Single
0                         1
1                         1
2                         0
3                         0
4                         0
...                     ...
2235                      0
2236                      0
2237                      1
2238                      0
2239                      0

[2240 rows x 15 columns]
```

Data Normalization

```
plt.figure(figsize=(20,7))
x = sns.boxplot(data=data)
x.set_xticklabels(x.get_xticklabels(),rotation=45)
```

```
[Text(0, 0, 'Age'),
 Text(1, 0, 'Education'),
 Text(2, 0, 'Income'),
 Text(3, 0, 'Dt_Customer'),
 Text(4, 0, 'Recency'),
 Text(5, 0, 'NumDealsPurchases'),
 Text(6, 0, 'NumWebVisitsMonth'),
 Text(7, 0, 'Complain'),
 Text(8, 0, 'Response'),
 Text(9, 0, 'Children'),
 Text(10, 0, 'AmountSpent'),
 Text(11, 0, 'NumPurchased'),
 Text(12, 0, 'Prev_campaigns'),
 Text(13, 0, 'Marital_Status_Couple'),
 Text(14, 0, 'Marital_Status_Single')]
```

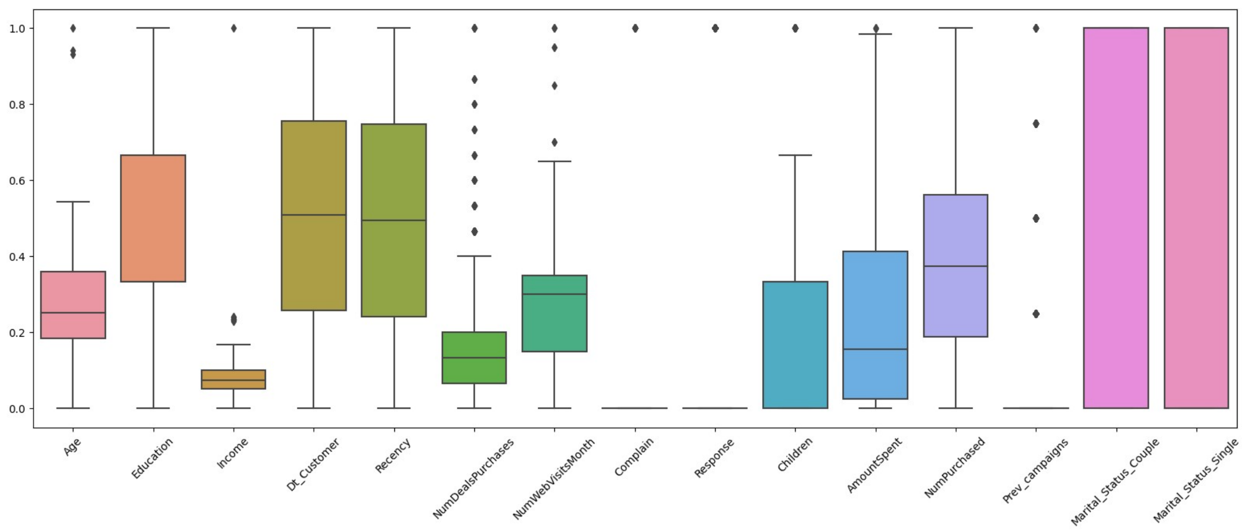


Box plot shows Income having outliers and there is a vast difference between range of values between Income and other columns. We need to normalize data so all variables have equal weightage

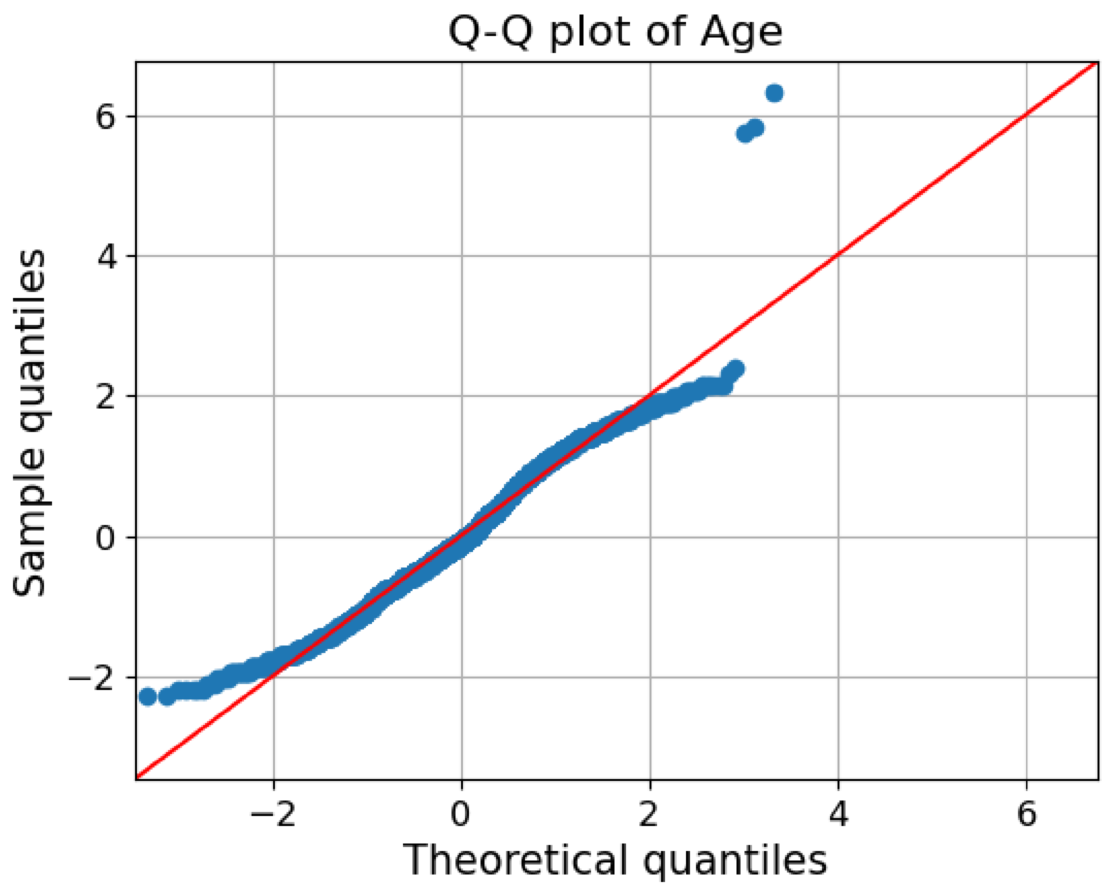
```
from sklearn import preprocessing

# Create x to store scaled values as floats
x = data[["Age","Education",    "Income",  "Dt_Customer",   "Recency",
      "NumDealsPurchases",  "NumWebVisitsMonth",  "Children",      "Amou
ntSpent",  "NumPurchased", "Prev_campaigns"]].values.astype(float)
```

```
# Preparing for normalizing
min_max_scaler = preprocessing.MinMaxScaler()

# Transform the data to fit minmax processor
x_scaled = min_max_scaler.fit_transform(x)

# Run the normalizer on the dataframe
data[["Age",    "Education",    "Income",  "Dt_Customer",  "Recency",
      "NumDealsPurchases",  "NumWebVisitsMonth",  "Children",     "Amou
ntSpent",  "NumPurchased",  "Prev_campaigns"]] = pd.DataFrame(x_scaled)

data.head()

        Age  Education    Income  Dt_Customer   Recency
NumDealsPurchases  \
0  0.378641   0.333333  0.084832     0.948498  0.585859
0.200000
1  0.407767   0.333333  0.067095     0.161660  0.383838
0.133333
2  0.300971   0.333333  0.105097     0.446352  0.262626
0.066667
3  0.116505   0.333333  0.037471     0.198856  0.262626
0.133333
4  0.145631   1.000000  0.085065     0.230329  0.949495
0.333333

   NumWebVisitsMonth  Complain  Response  Children  AmountSpent
NumPurchased  \
0               0.35         0         1  0.000000     0.639683
0.6875
1               0.25         0         0  0.666667     0.008730
0.1250
2               0.20         0         0  0.000000     0.305952
0.6250
3               0.30         0         0  0.333333     0.019048
0.1875
4               0.25         0         0  0.333333     0.165476
0.4375

   Prev_campaigns  Marital_Status_Couple  Marital_Status_Single
0             0.0                      0                      1
1             0.0                      0                      1
2             0.0                      1                      0
3             0.0                      1                      0
4             0.0                      1                      0

plt.figure(figsize=(20,7))
x = sns.boxplot(data=data)
x.set_xticklabels(x.get_xticklabels(),rotation=45)
```

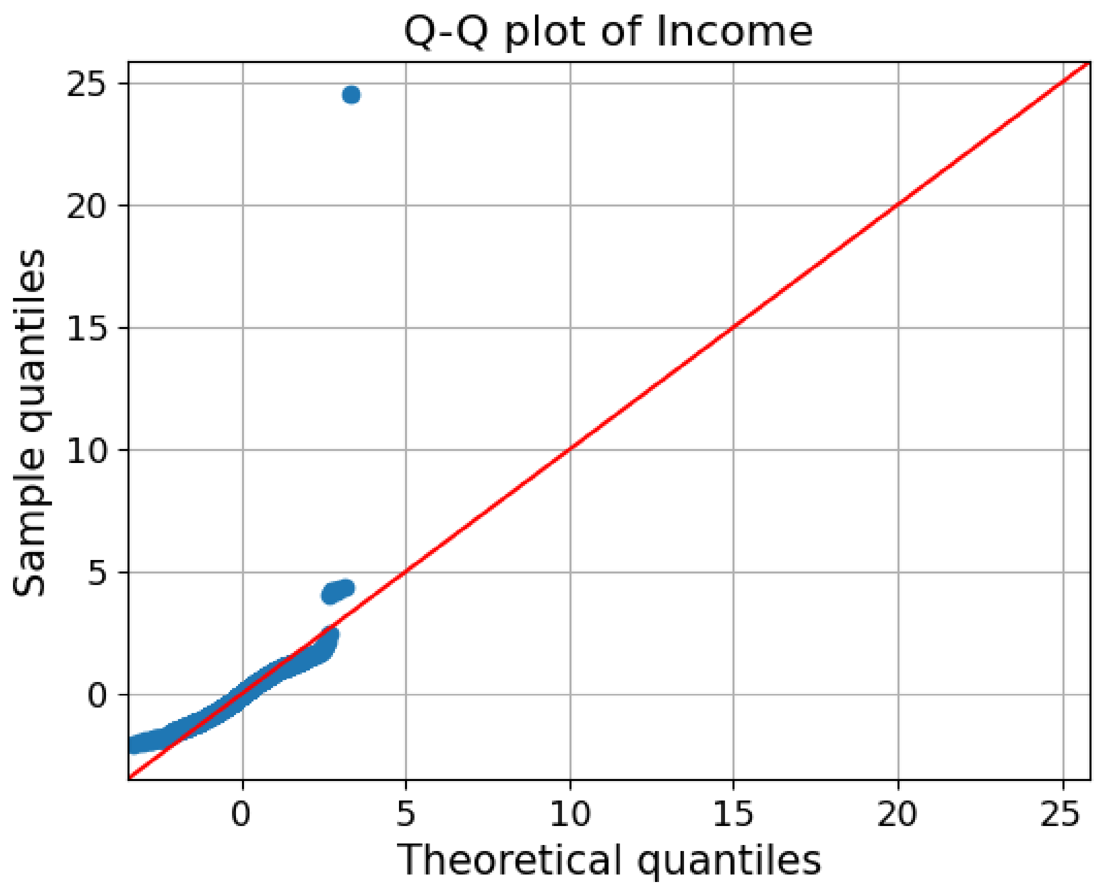
```
[Text(0, 0, 'Age'),
 Text(1, 0, 'Education'),
 Text(2, 0, 'Income'),
 Text(3, 0, 'Dt_Customer'),
 Text(4, 0, 'Recency'),
 Text(5, 0, 'NumDealsPurchases'),
 Text(6, 0, 'NumWebVisitsMonth'),
 Text(7, 0, 'Complain'),
 Text(8, 0, 'Response'),
 Text(9, 0, 'Children'),
 Text(10, 0, 'AmountSpent'),
 Text(11, 0, 'NumPurchased'),
 Text(12, 0, 'Prev_campaigns'),
 Text(13, 0, 'Marital_Status_Couple'),
 Text(14, 0, 'Marital_Status_Single')]
```




```
#checking the distribution of independent variables
data = data.dropna()
from statsmodels.graphics.gofplots import qqplot
data_norm=data[['Age','Income','Dt_Customer','Recency','NumWebVisitsMo
nth','AmountSpent']]
for c in data_norm.columns[:]:
  plt.figure(figsize=(8,5))
  fig=qqplot(data_norm[c],line='45',fit='True')
  plt.xticks(fontsize=13)
  plt.yticks(fontsize=13)
  plt.xlabel("Theoretical quantiles",fontsize=15)
  plt.ylabel("Sample quantiles",fontsize=15)
  plt.title("Q-Q plot of {}".format(c),fontsize=16)
  plt.grid(True)
  plt.show()

<Figure size 800x500 with 0 Axes>
```

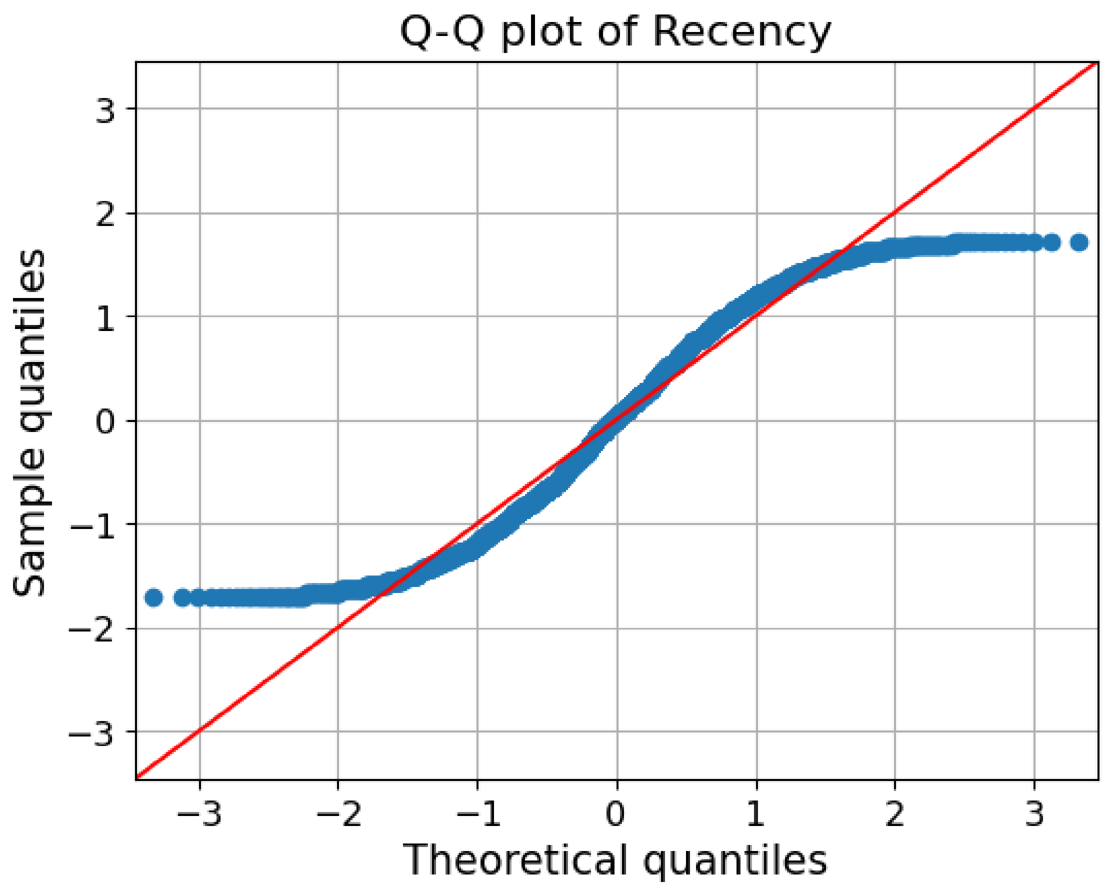



```
<Figure size 800x500 with 0 Axes>
```

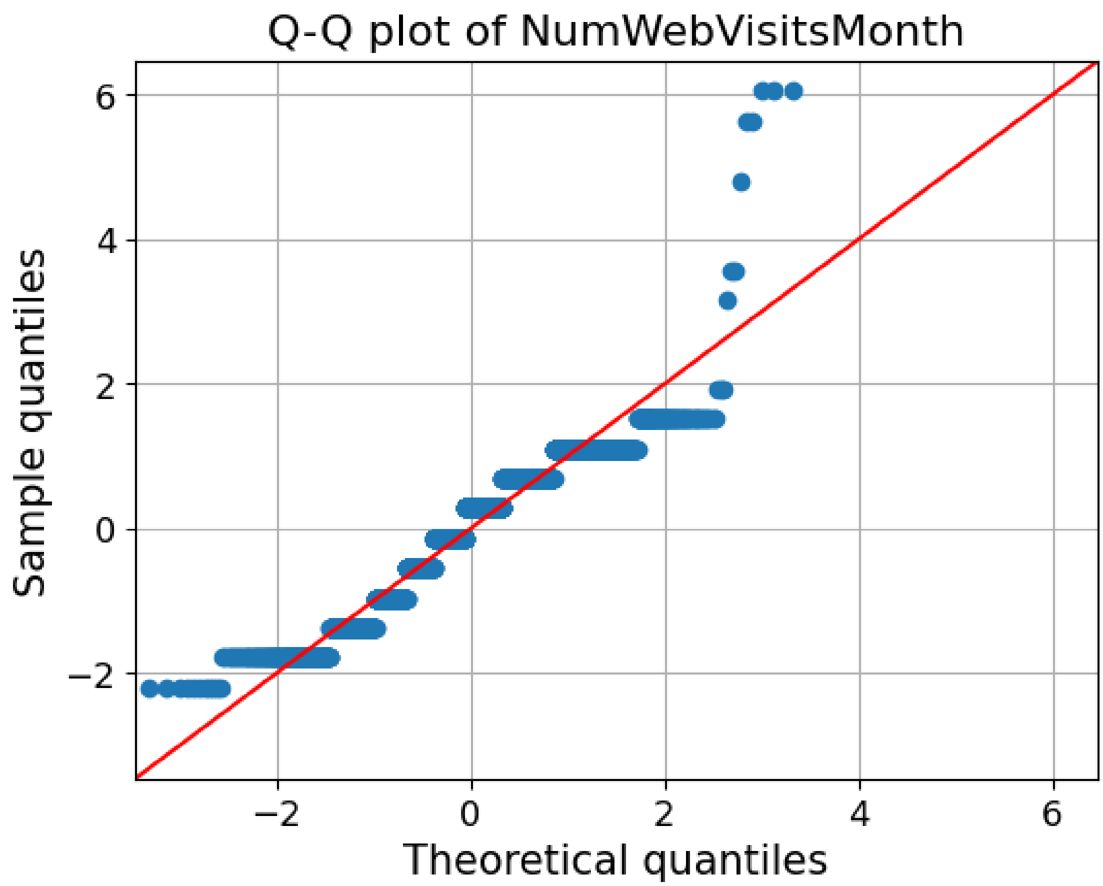



```
<Figure size 800x500 with 0 Axes>
```

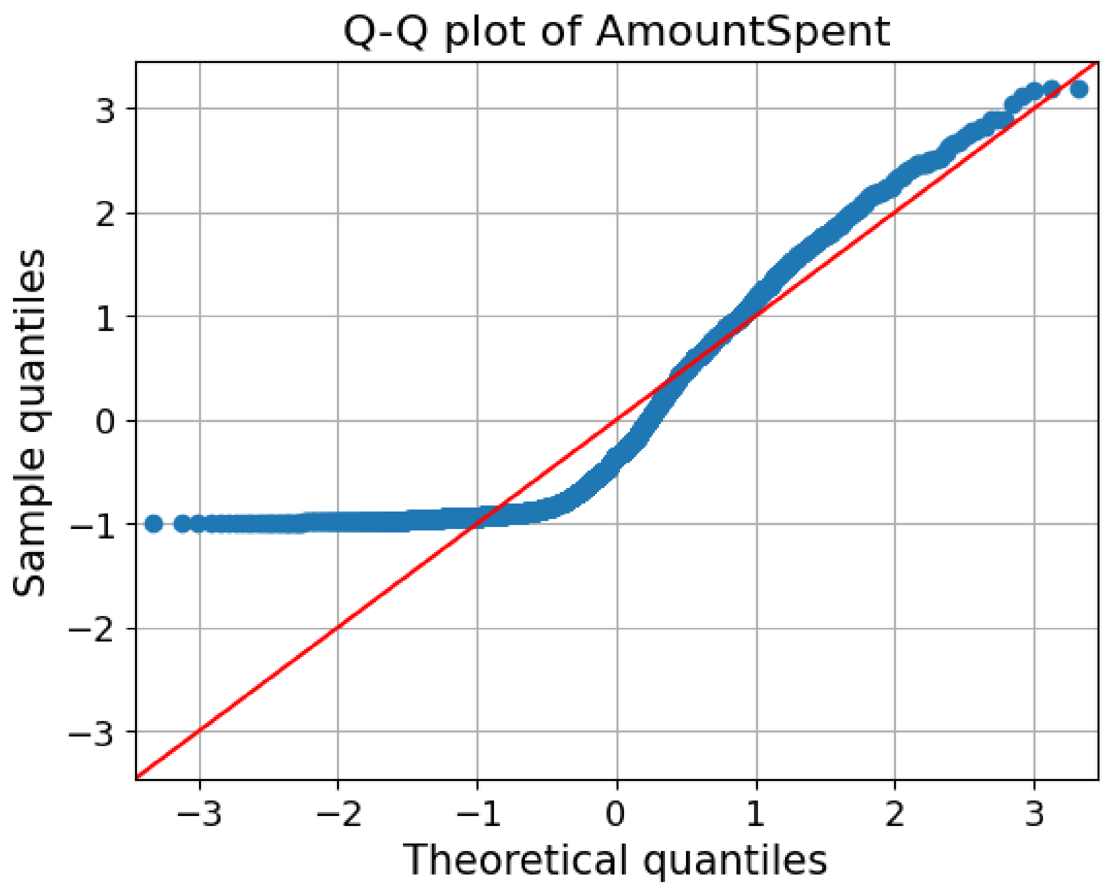

```
<Figure size 800x500 with 0 Axes>
```




```
<Figure size 800x500 with 0 Axes>
```




```
<Figure size 800x500 with 0 Axes>
```



- Q-Q plots show most of the data is normally distributed
- There are few Outliers in Income and age

```
#pair plot to check the colinearity
# sns.pairplot(data)

#Using OLS for finding the p value to check the significant features
import statsmodels.api as sm

model = sm.OLS(data['Response'], data[["Age",    "Education",
      "Income",  "Dt_Customer",
                                         "Recency" ,"NumDealsPurchases",
      "NumWebVisitsMonth",
                                         "Complain",     "Children",
      "AmountSpent"    ,"NumPurchased",
                                         "Prev_campaigns",
      "Marital_Status_Couple",    "Marital_Status_Single"]]).fit()

# Print out the statistics
model.summary()

<class 'statsmodels.iolib.summary.Summary'>
"""
```

```
                            OLS Regression Results

==============================================================================
========
Dep. Variable:               Response   R-squared:
0.301
Model:                            OLS   Adj. R-squared:
0.297
Method:                 Least Squares   F-statistic:
73.64
Date:                Tue, 03 Oct 2023   Prob (F-statistic):
3.34e-162
Time:                        01:52:26   Log-Likelihood:
-465.46
No. Observations:                2240   AIC:
958.9
Df Residuals:                    2226   BIC:
1039.
Df Model:                          13

Covariance Type:            nonrobust

==============================================================================
===================
                            coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
-------------------
Age                      -0.0404      0.057     -0.712      0.477
-0.152       0.071
Education                 0.1118      0.023      4.859      0.000
0.067       0.157
Income                   -0.0596      0.246     -0.242      0.809
-0.543       0.423
Dt_Customer               0.2200      0.025      8.872      0.000
0.171       0.269
Recency                  -0.2388      0.022    -11.030      0.000
-0.281      -0.196
NumDealsPurchases         0.1661      0.063      2.648      0.008
0.043       0.289
NumWebVisitsMonth         0.1149      0.075      1.534      0.125
-0.032       0.262
Complain                  0.0383      0.066      0.582      0.560
-0.091       0.167
Children                 -0.1185      0.034     -3.469      0.001
-0.186      -0.052
AmountSpent               0.2087      0.057      3.642      0.000
0.096       0.321
NumPurchased             -0.1920      0.054     -3.571      0.000
-0.297      -0.087
```

```
Prev_campaigns              0.8237       0.043     19.082       0.000
0.739       0.908
Marital_Status_Couple       0.0158       0.037      0.431       0.666
-0.056       0.087
Marital_Status_Single       0.1247       0.037      3.356       0.001
0.052       0.198
======================================================================
========
Omnibus:                      503.179   Durbin-Watson:
2.025
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
991.376
Skew:                           1.336   Prob(JB):
5.31e-216
Kurtosis:                       4.865   Cond. No.
54.5
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
"""
```

- Education, Dt_Customer, Recency, Childrean, Amount spent, NumPurchases, Prev_campaign have a p value of 0.00 denoting a very high significance
- Income have a p value of 0.8 which shows it does not have a lot of significance.
- Marital_Status_Couple has p value 0.666 whereas Marital_Status_Single has 0.001 which is interesting as they both originate from same data column. It shows the significance of creating dummy variables for categorical data

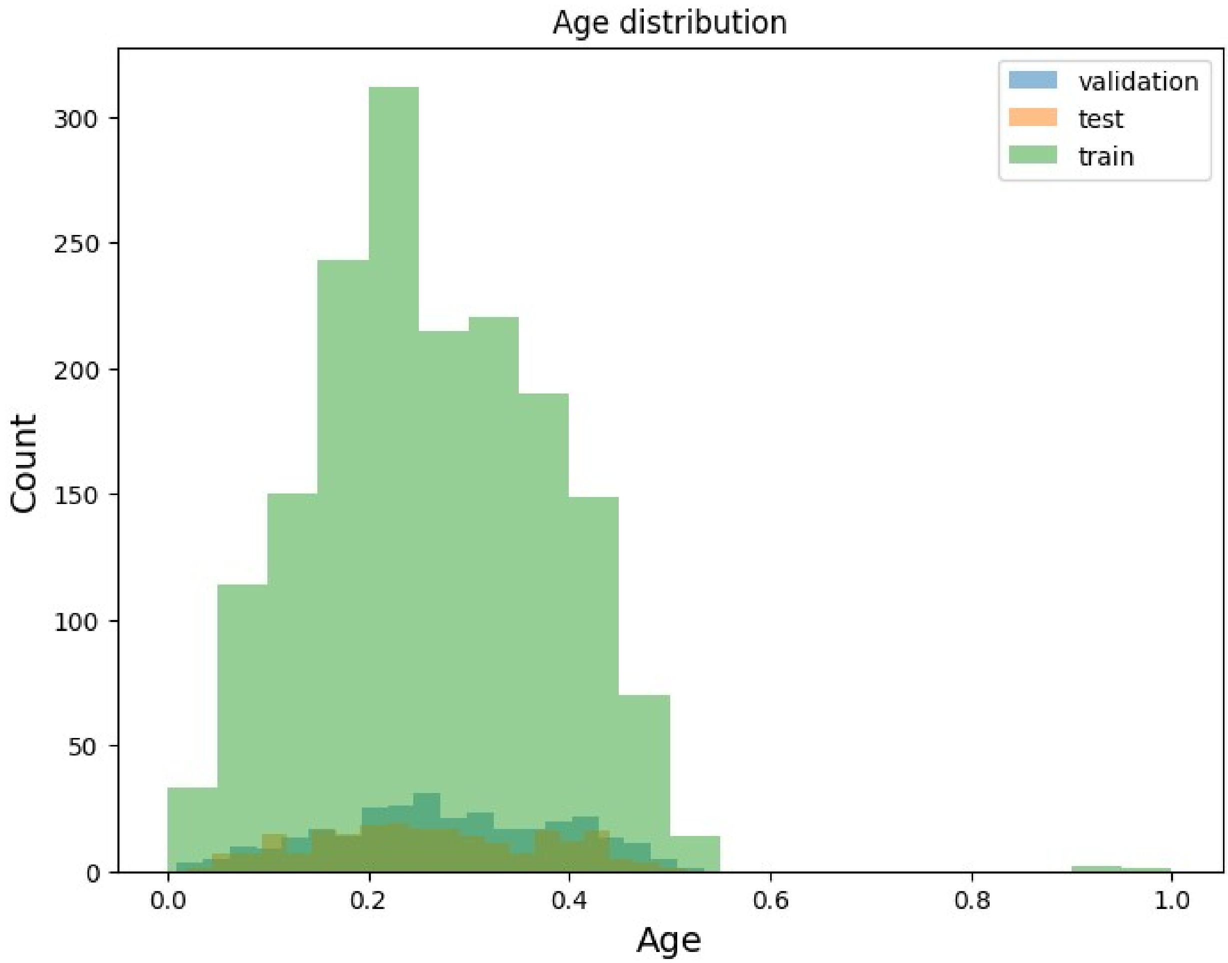
Logistic Regression

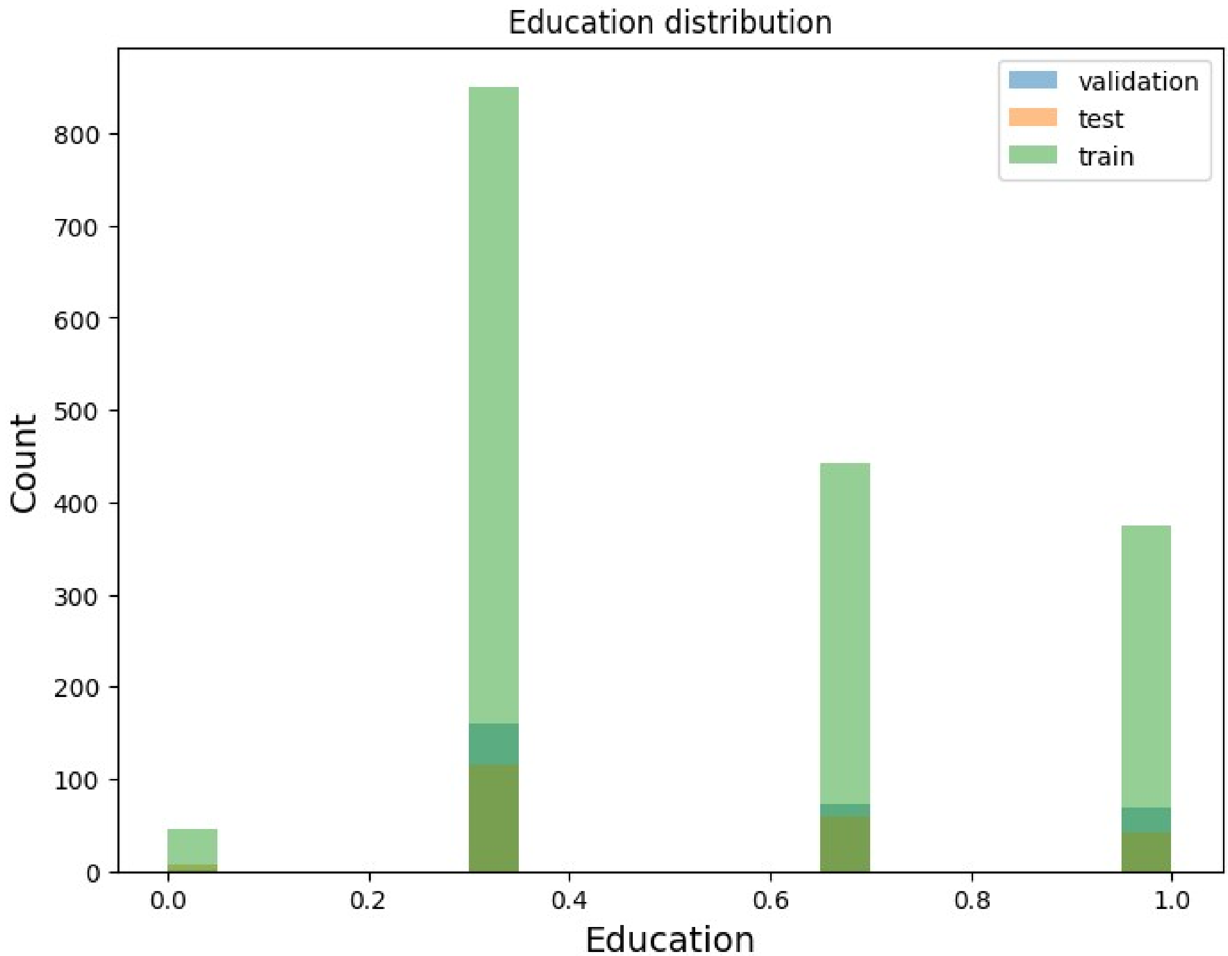
```
from sklearn.model_selection import  train_test_split

X = data[ ["Age",     "Education",     "Income",  "Dt_Customer",
                                       "Recency" ,"NumDealsPurchases",
     "NumWebVisitsMonth",
                                       "Complain",     "Children",
     "AmountSpent"    ,"NumPurchased",
                                       "Prev_campaigns",
     "Marital_Status_Couple",   "Marital_Status_Single"]]

y = data['Response']

#Spliting data into Training 76.5%, Validation set 13.5% and Test set
10%
```

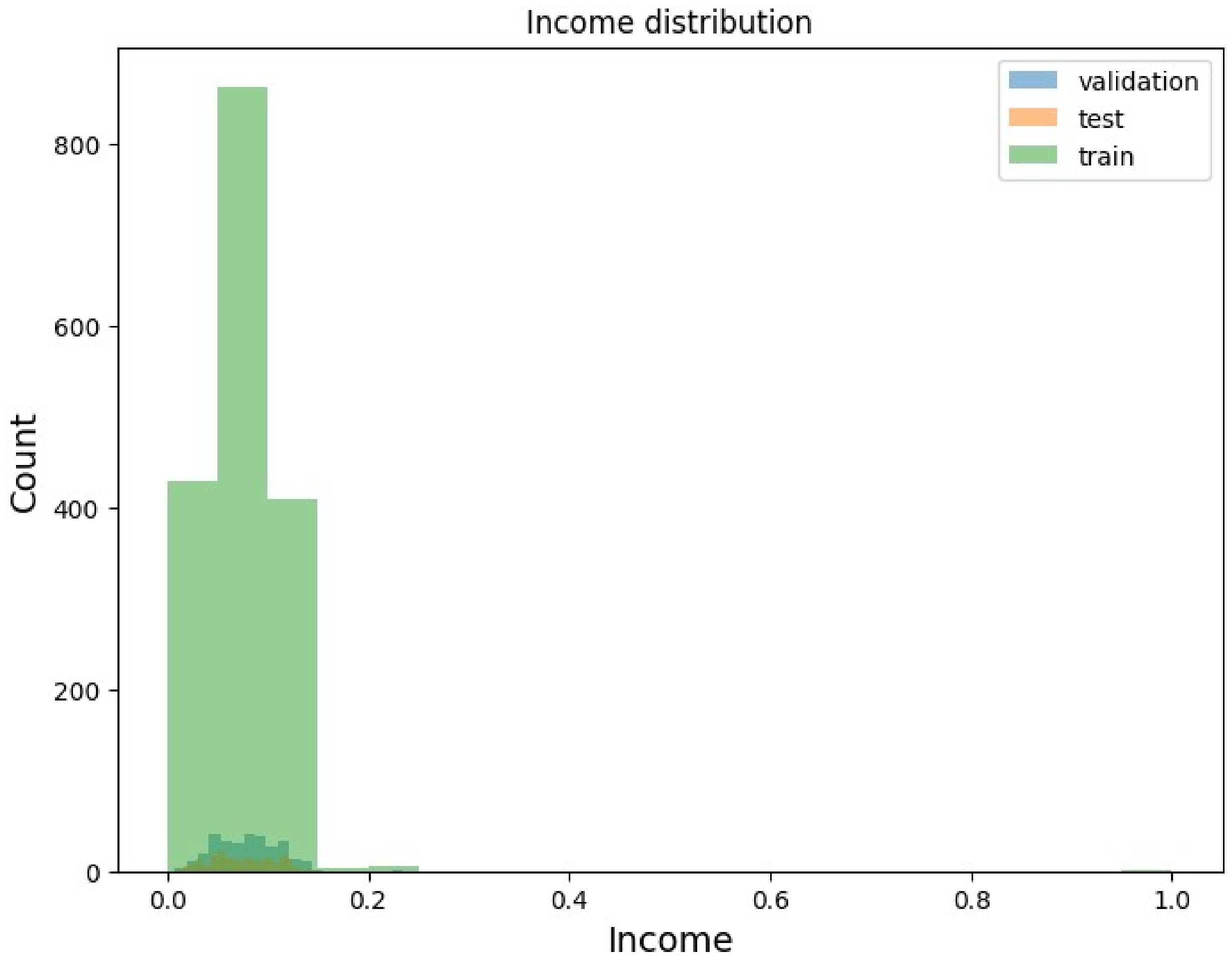
```
X_t, X_test, y_t, y_test = train_test_split(X, y, test_size=0.1,
random_state=1)

X_train, X_val, y_train, y_val = train_test_split(X_t, y_t,
test_size=0.15, random_state=1)
```

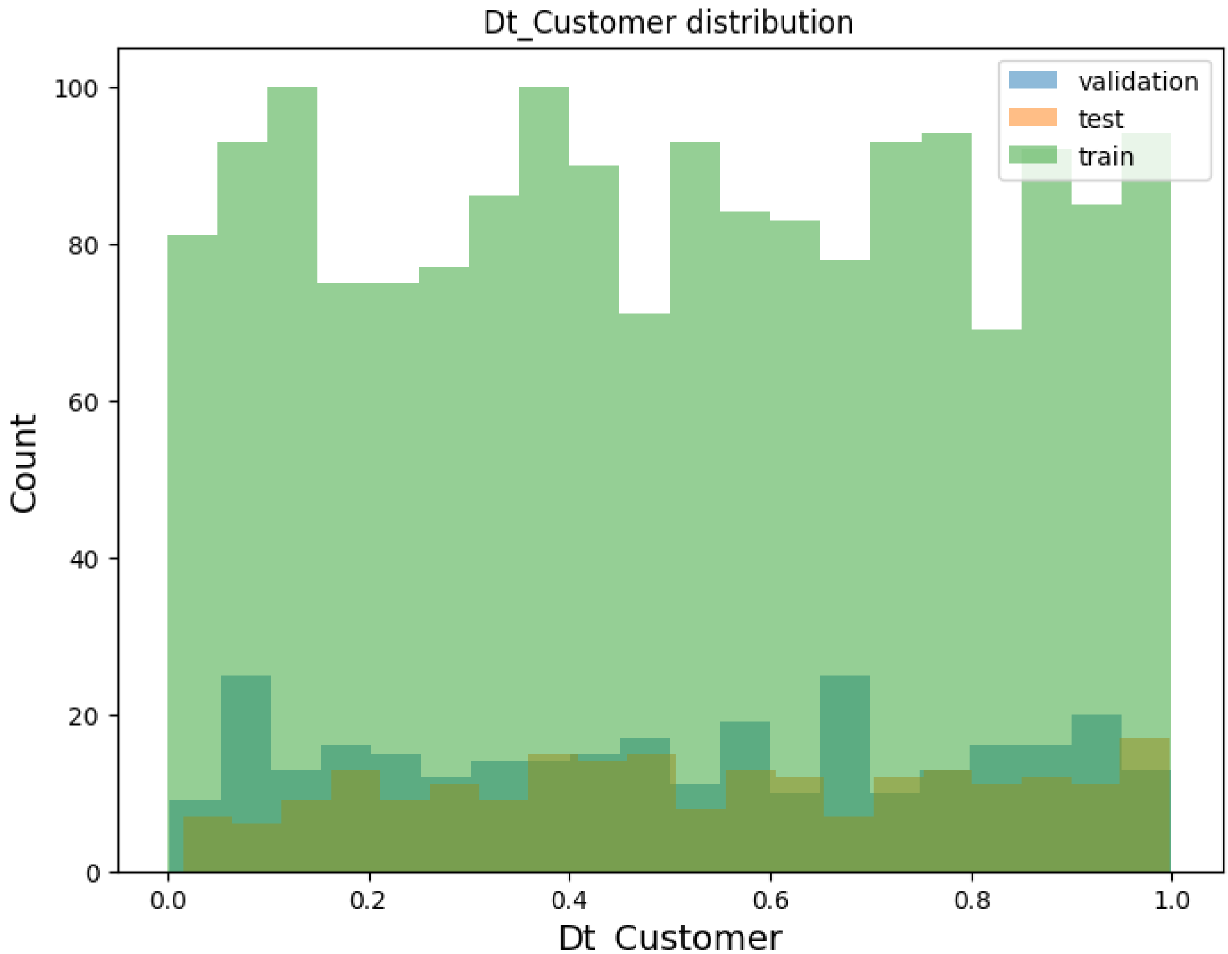
Splitting data for train test and validation 10% testing 15% of remaining train data for validation

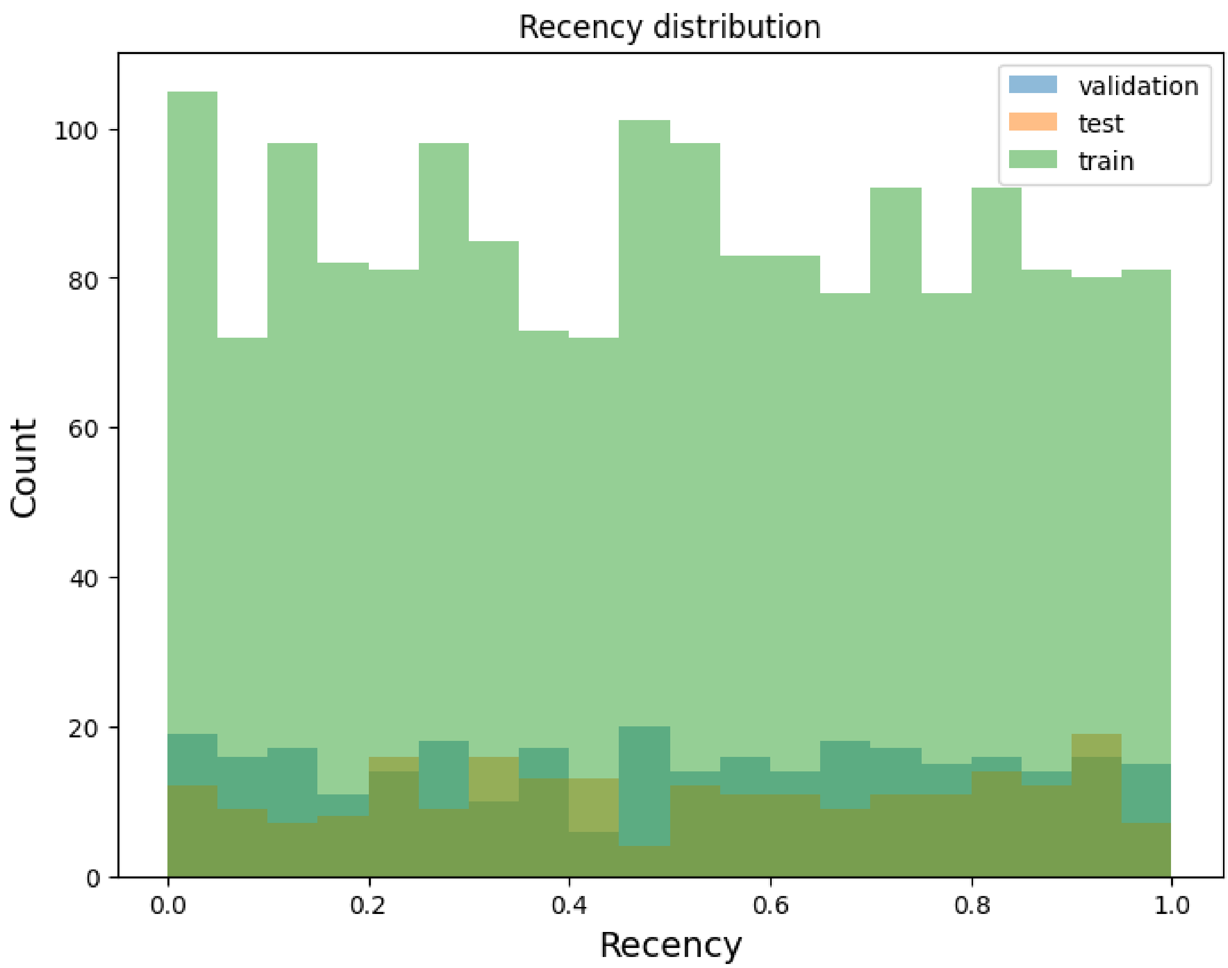
```
# Looking the data for test, training and validation set
X_test_plot = X_test[["Age",     "Education",    "Income",
      "Dt_Customer",
                                        "Recency" ,"NumDealsPurchases",
      "NumWebVisitsMonth",
                                        "Complain",    "Children",
      "AmountSpent"    ,"NumPurchased",
                                        "Prev_campaigns",
      "Marital_Status_Couple",   "Marital_Status_Single"]]

X_val_plot = X_val[["Age", "Education",     "Income",  "Dt_Customer",
                                        "Recency" ,"NumDealsPurchases",
      "NumWebVisitsMonth",
                                        "Complain",    "Children",
      "AmountSpent"    ,"NumPurchased",
                                        "Prev_campaigns",
      "Marital_Status_Couple",   "Marital_Status_Single"]]

X_train_plot = X_train[["Age",   "Education",    "Income",
      "Dt_Customer",
                                        "Recency" ,"NumDealsPurchases",
      "NumWebVisitsMonth",
                                        "Complain",    "Children",
      "AmountSpent"    ,"NumPurchased",
                                        "Prev_campaigns",
      "Marital_Status_Couple",   "Marital_Status_Single"]]

# Plotting the data to see the histogram
for c in X_test_plot.columns[:]:
  plt.figure(figsize=(8,6))
  plt.hist(X_val_plot[c], bins=20, alpha=0.5, label="validation")
  plt.hist(X_test_plot[c], bins=20, alpha=0.5, label="test")
  plt.hist(X_train_plot[c], bins=20, alpha=0.5, label="train")
  plt.xlabel(c, size=14)
  plt.ylabel("Count", size=14)
  plt.legend(loc='upper right')
  plt.title("{} distribution".format(c))
  plt.show()
```


Age distribution
validation
test
train
Count
Age
0
50
100
150
200
250
300
0.0
0.2
0.4
0.6
0.8
1.0

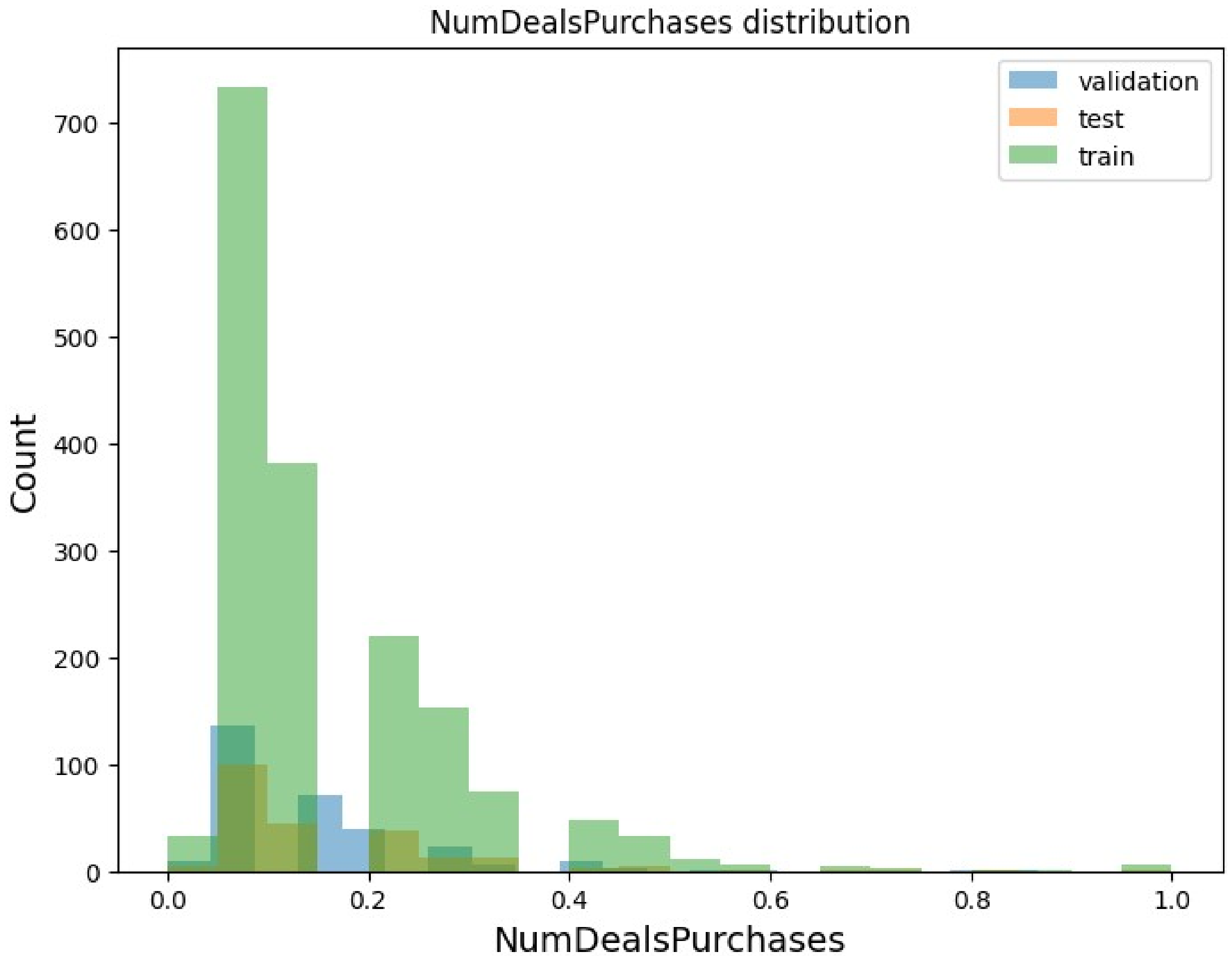

Education distribution
validation
test
train
Count
0
100
200
300
400
500
600
700
800
0.0
0.2
0.4
0.6
0.8
1.0
Education

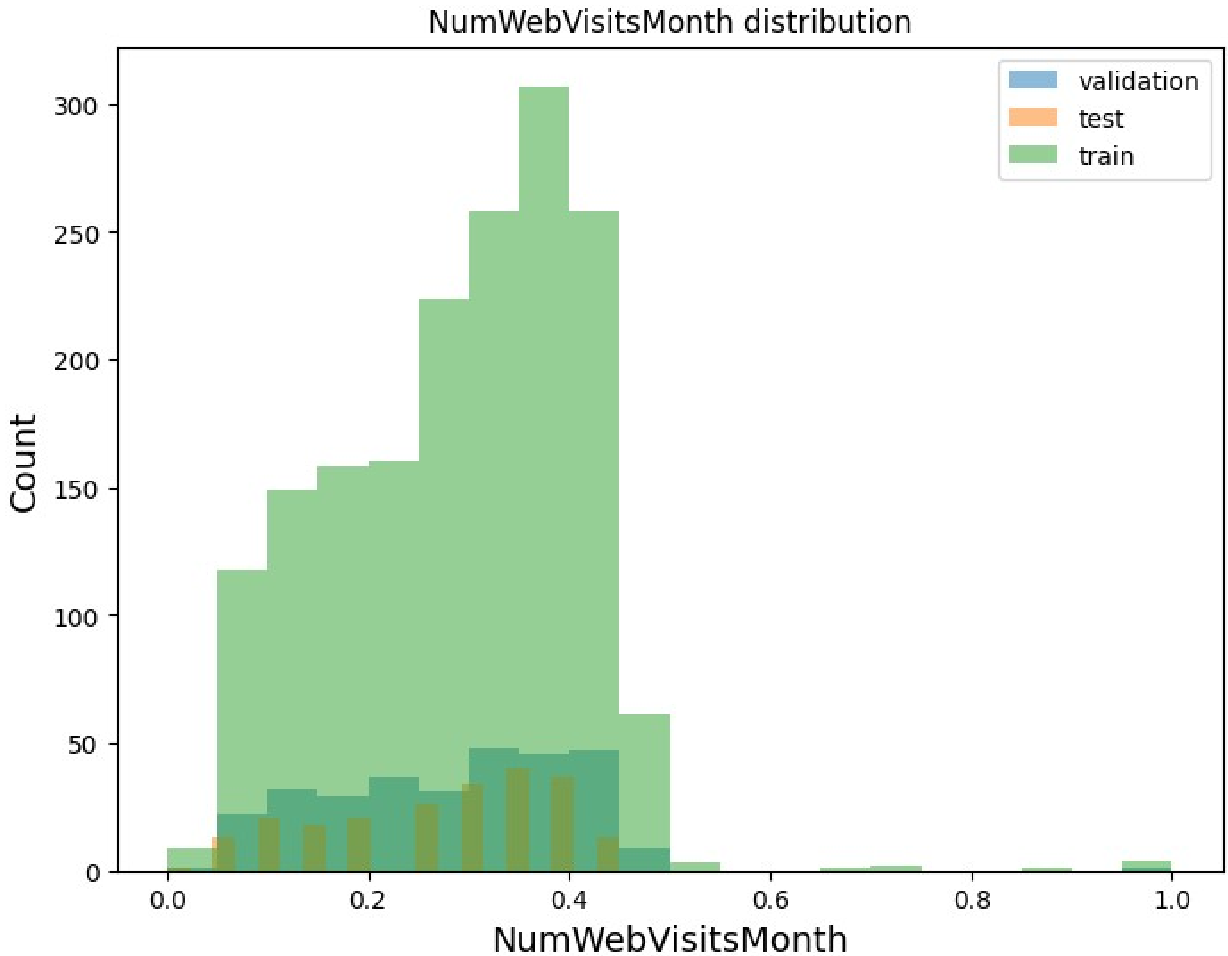

Income distribution
validation
test
train
Count
0
200
400
600
800
0.0
0.2
0.4
0.6
0.8
1.0
Income

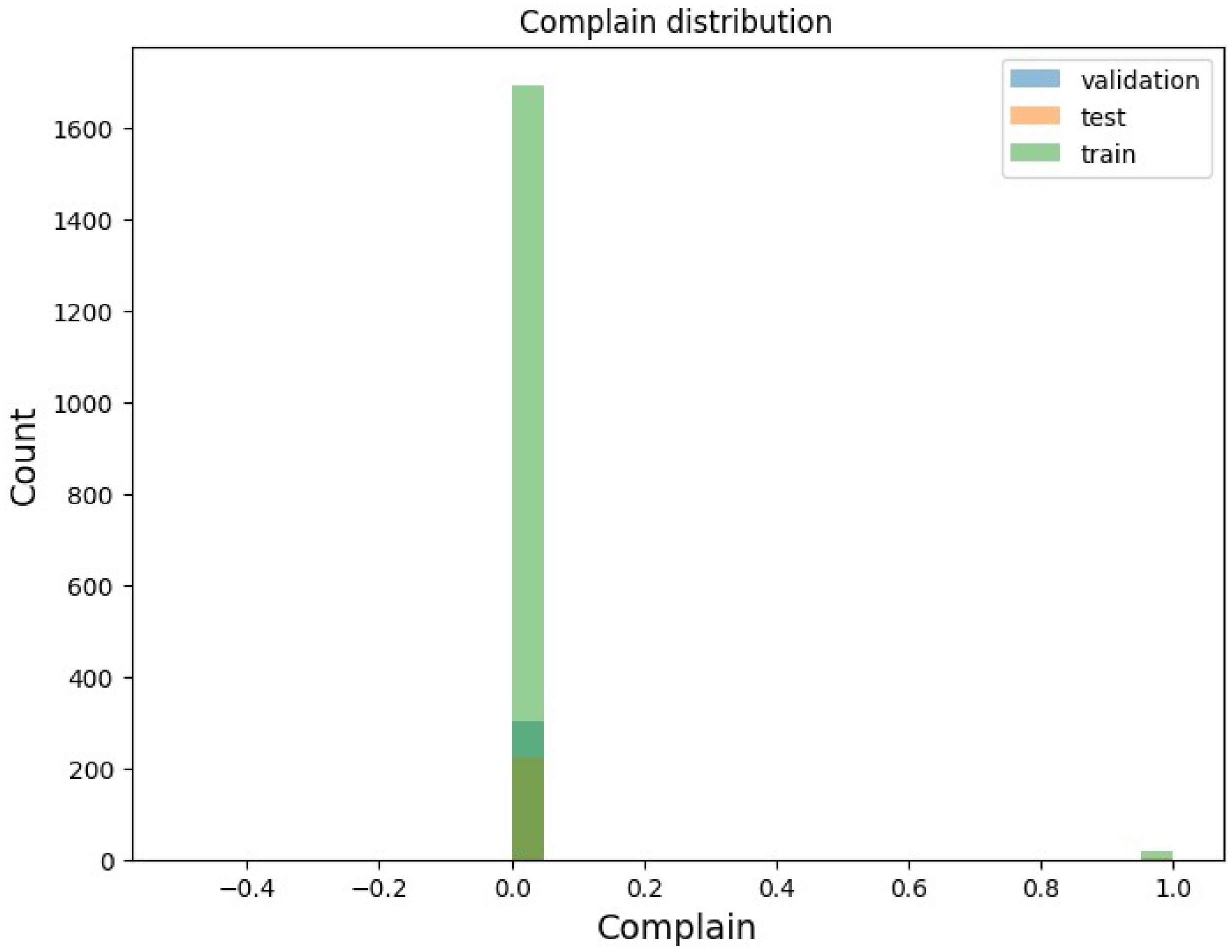

Dt_Customer distribution
validation
test
train
Count
0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0
Dt_Customer

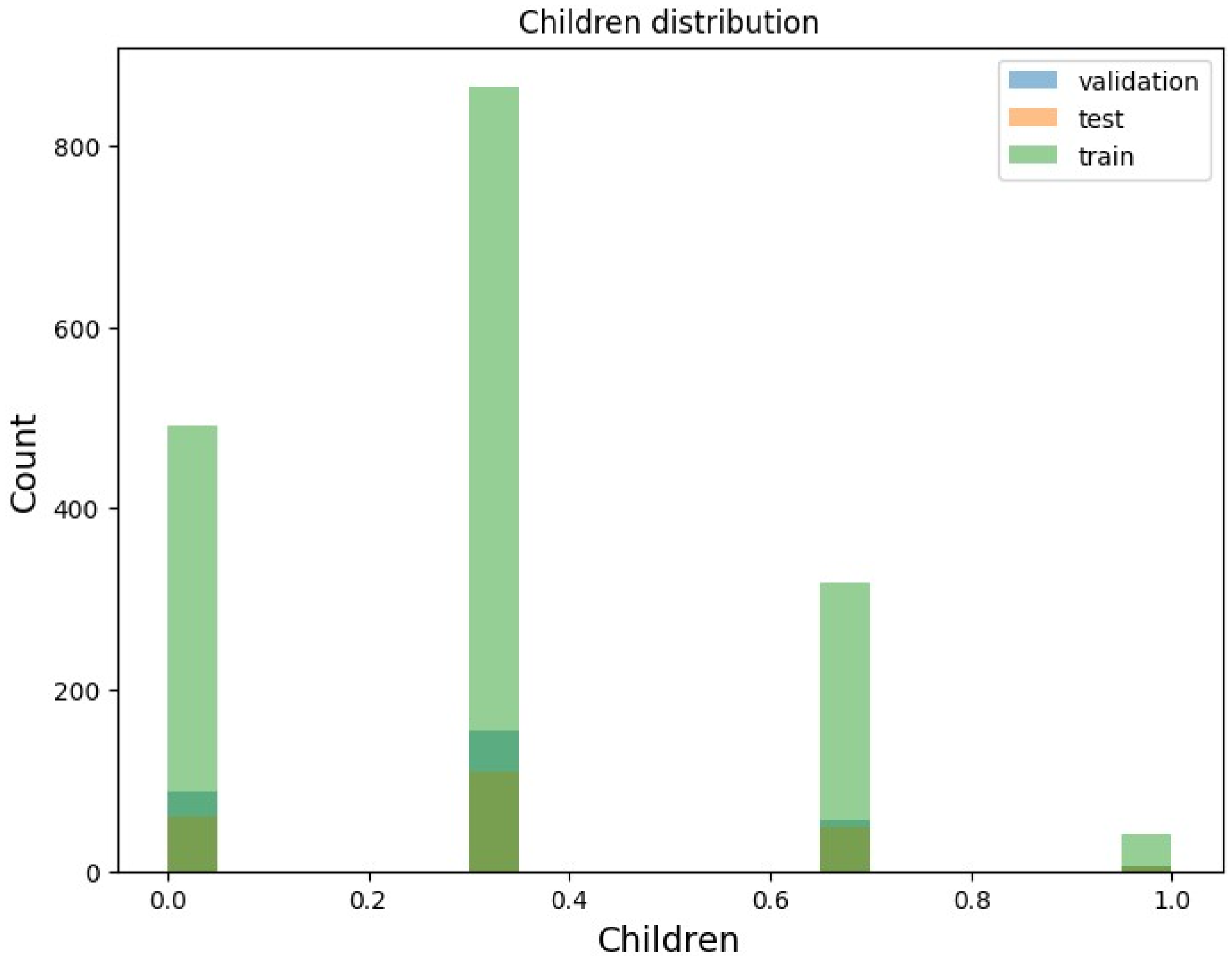


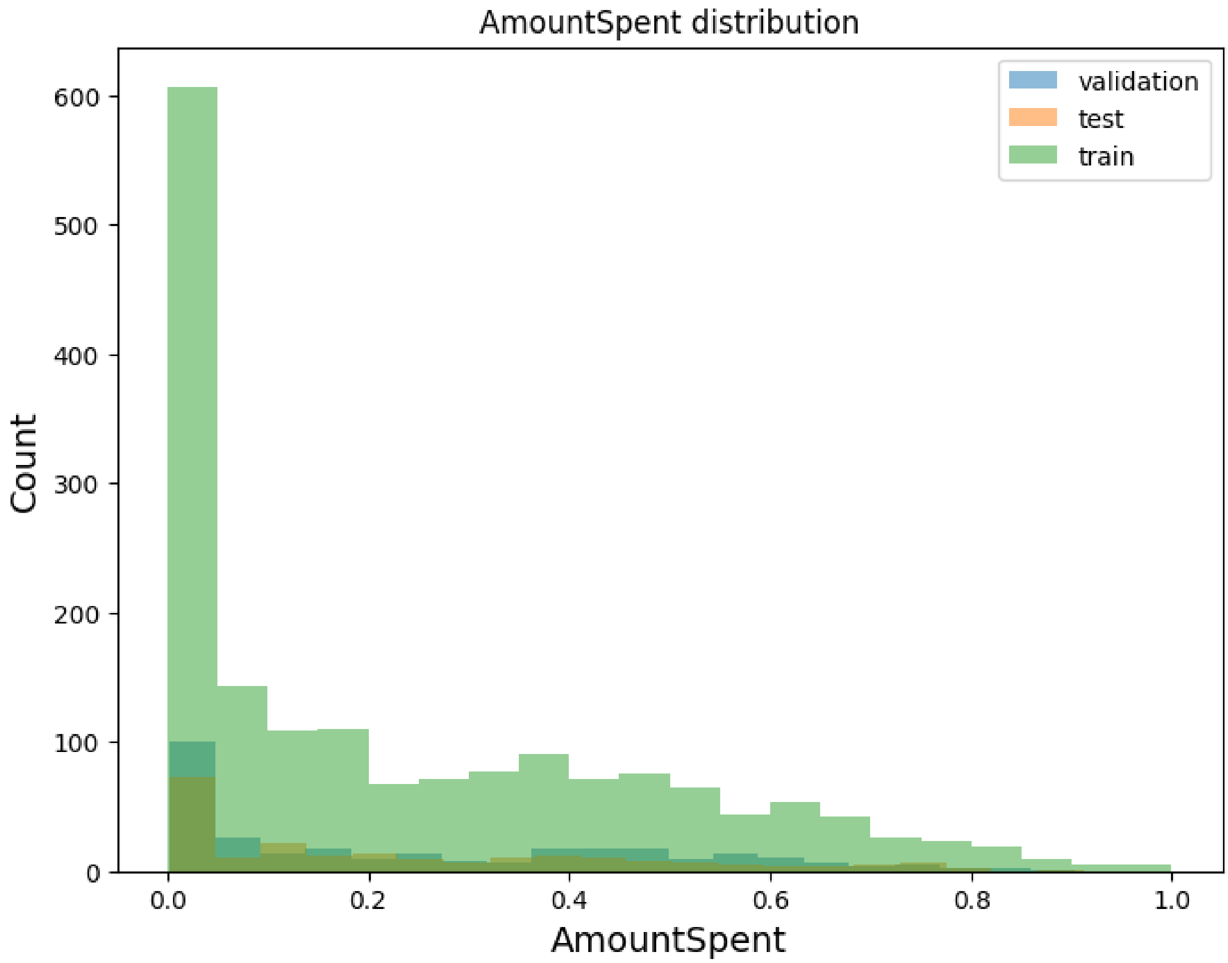
Recency distribution
validation
test
train
Count
Recency
0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0


NumDealsPurchases distribution
validation
test
train
Count
NumDealsPurchases
0
100
200
300
400
500
600
700
0.0
0.2
0.4
0.6
0.8
1.0

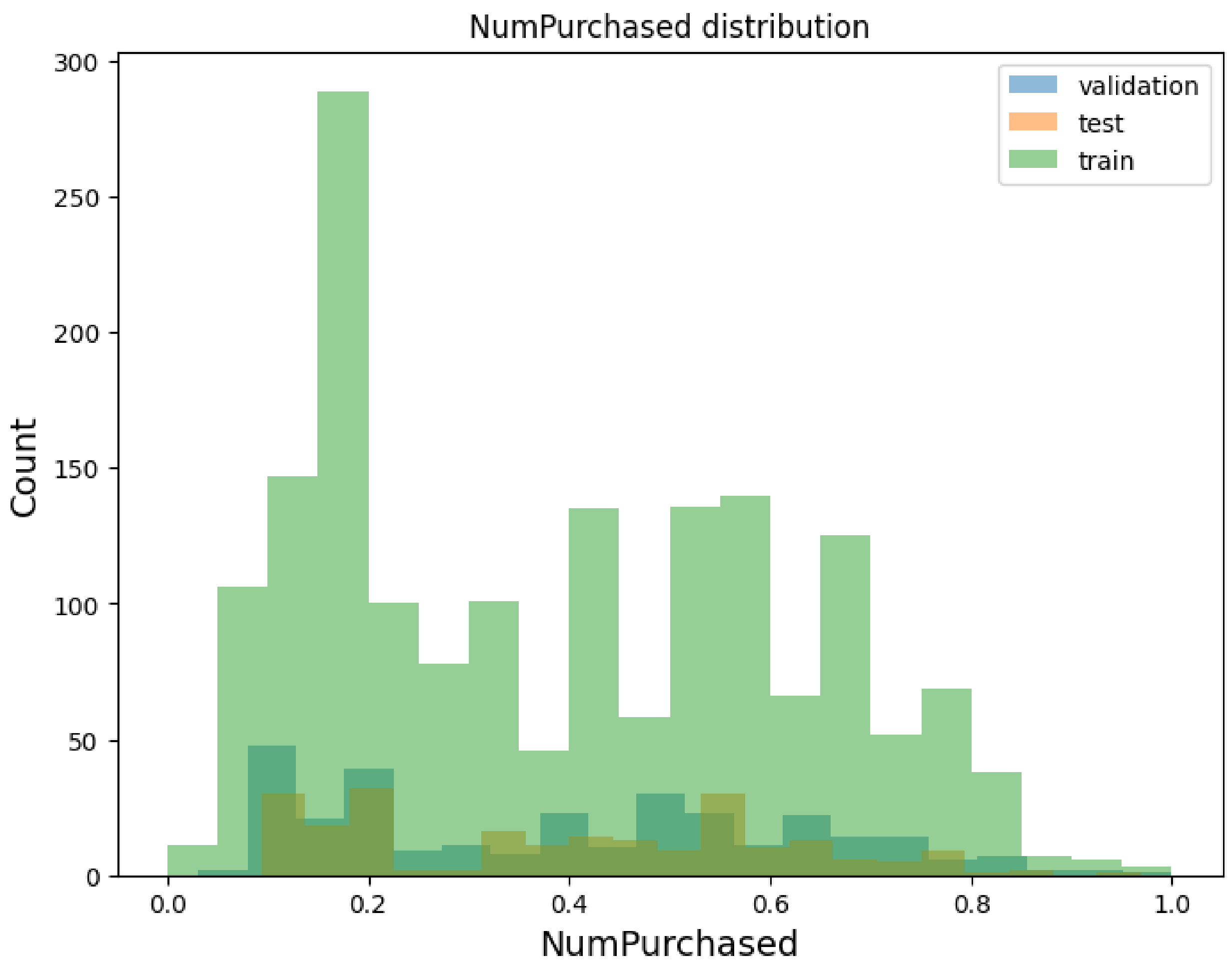

NumWebVisitsMonth distribution
validation
test
train
Count
0
50
100
150
200
250
300
0.0
0.2
0.4
0.6
0.8
1.0
NumWebVisitsMonth

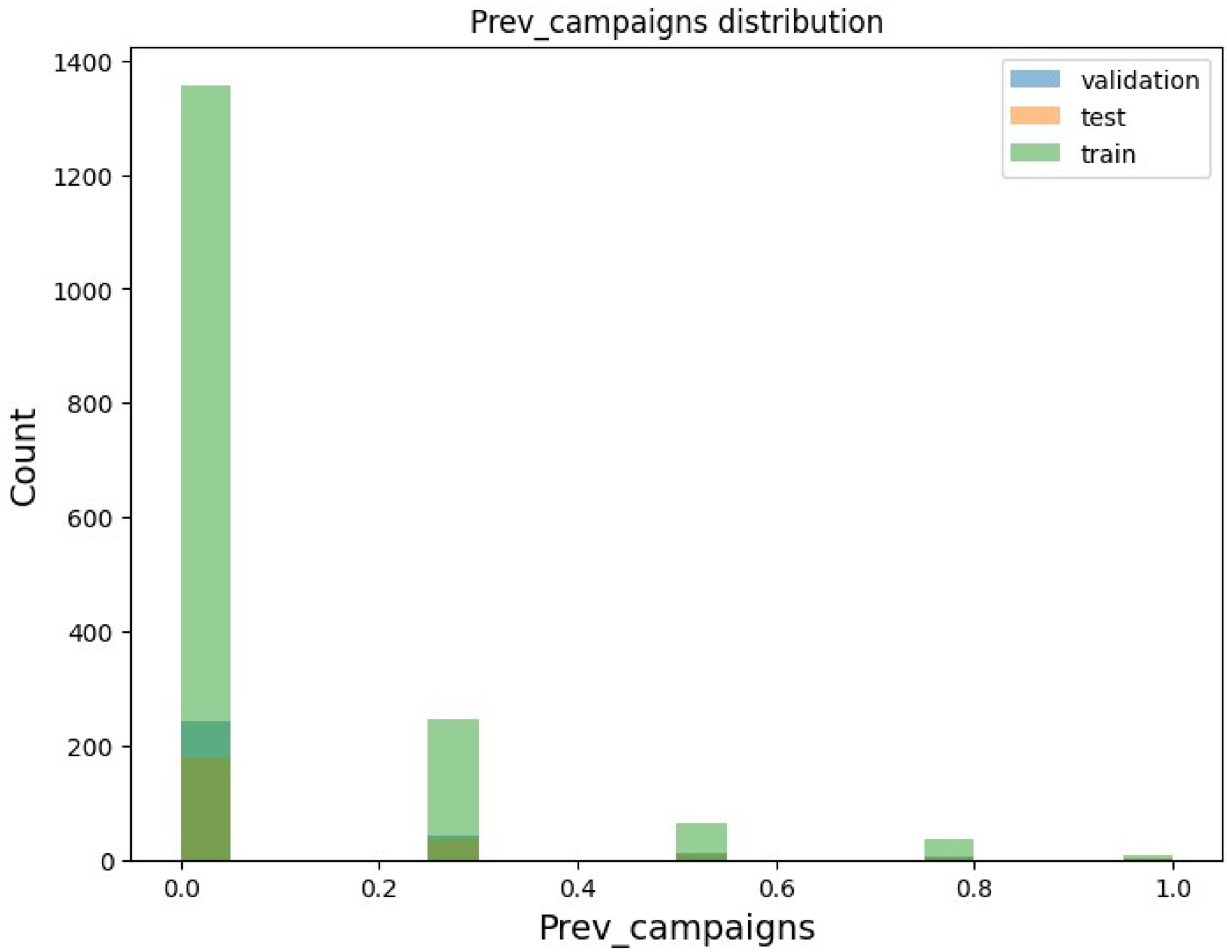

Complain distribution
validation
test
train
Count
Complain
0
200
400
600
800
1000
1200
1400
1600
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0


Children distribution
validation
test
train
Count
Children
0
200
400
600
800
0.0
0.2
0.4
0.6
0.8
1.0

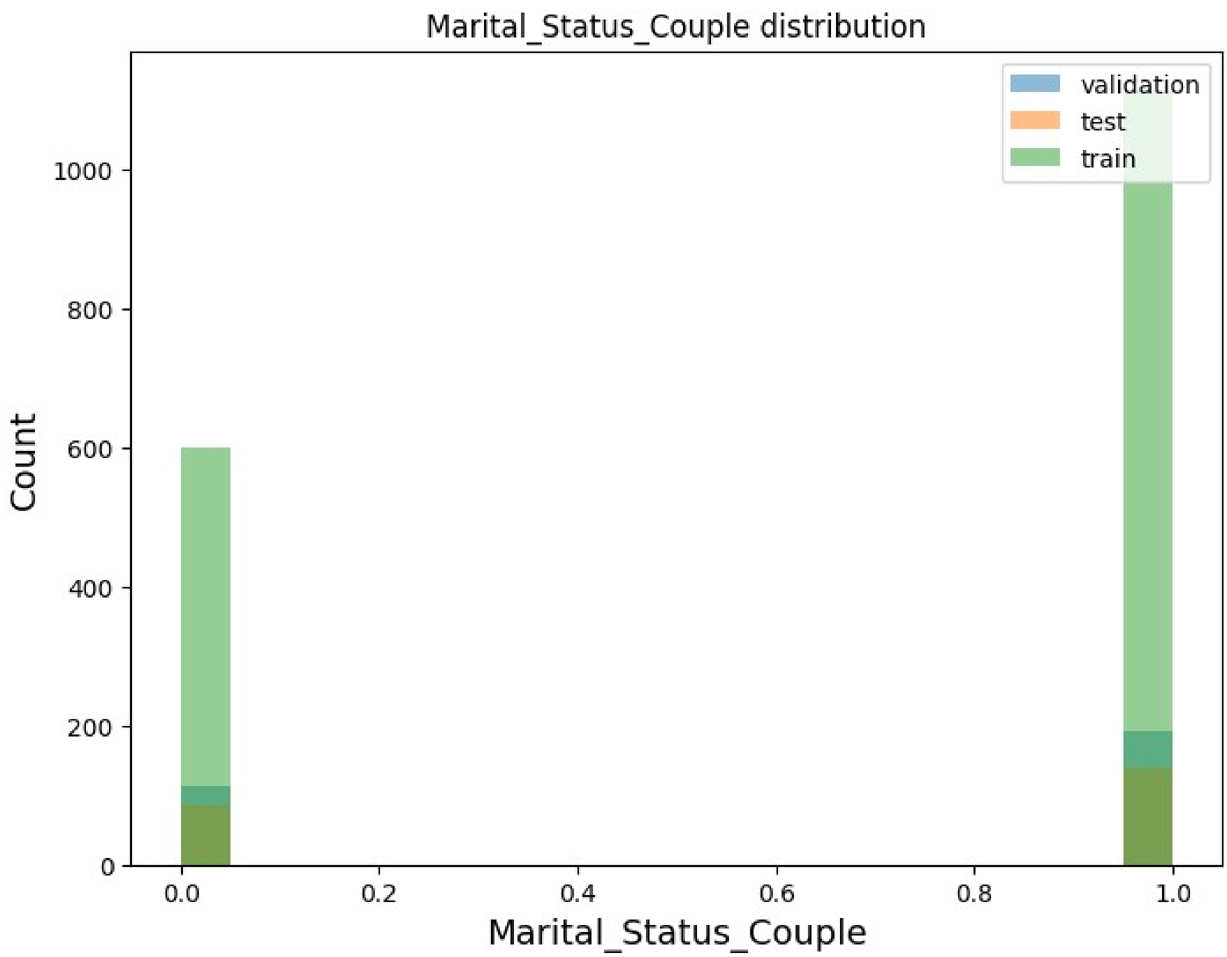


AmountSpent distribution
validation
test
train
Count
AmountSpent
0
100
200
300
400
500
600
0.0
0.2
0.4
0.6
0.8
1.0

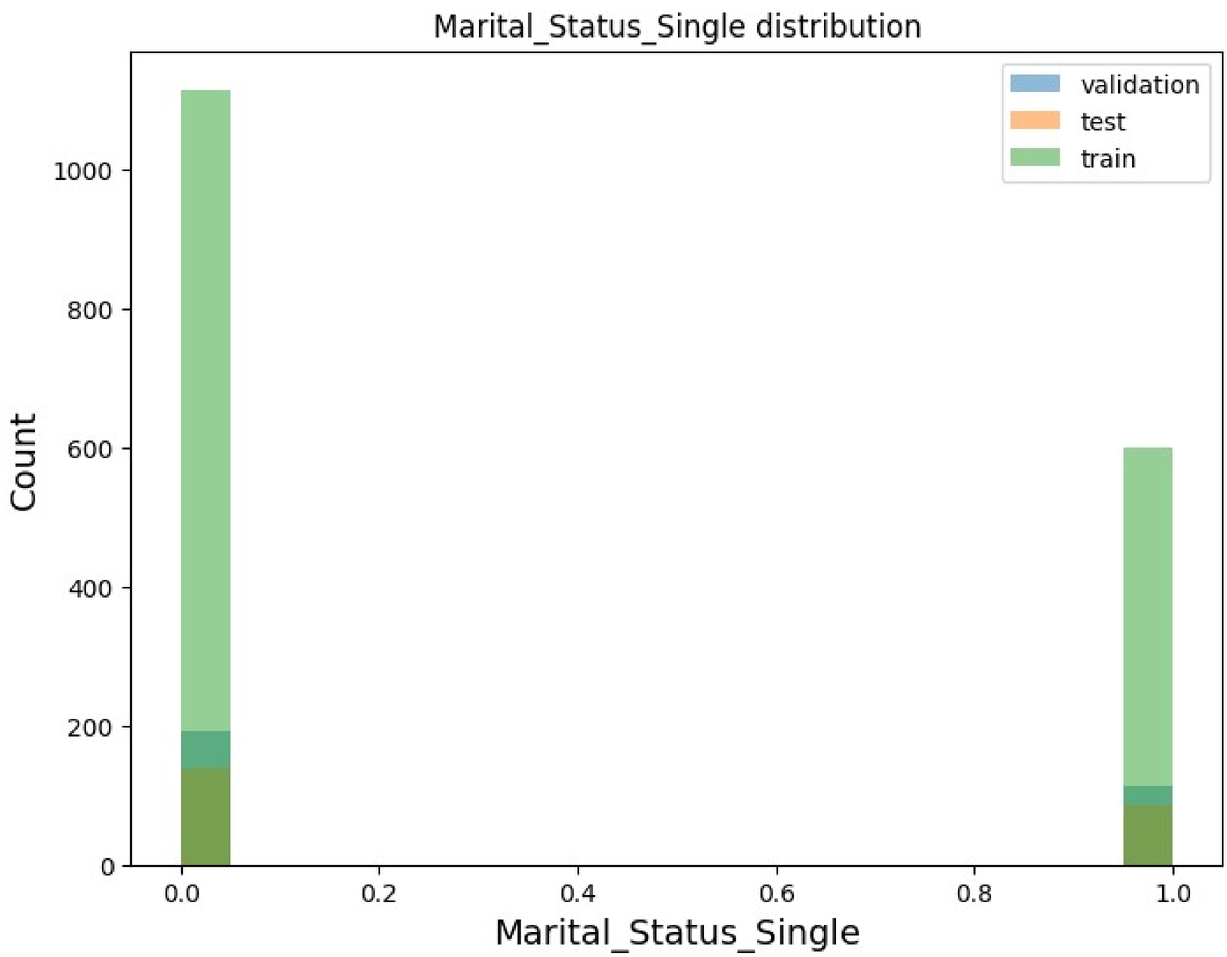

NumPurchased distribution
validation
test
train
Count
NumPurchased
0
50
100
150
200
250
300
0.0
0.2
0.4
0.6
0.8
1.0



Prev_campaigns distribution
validation
test
train
Count
Prev_campaigns
0
200
400
600
800
1000
1200
1400
0.0
0.2
0.4
0.6
0.8
1.0



Marital_Status_Couple distribution
validation
test
train
1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Marital_Status_Couple



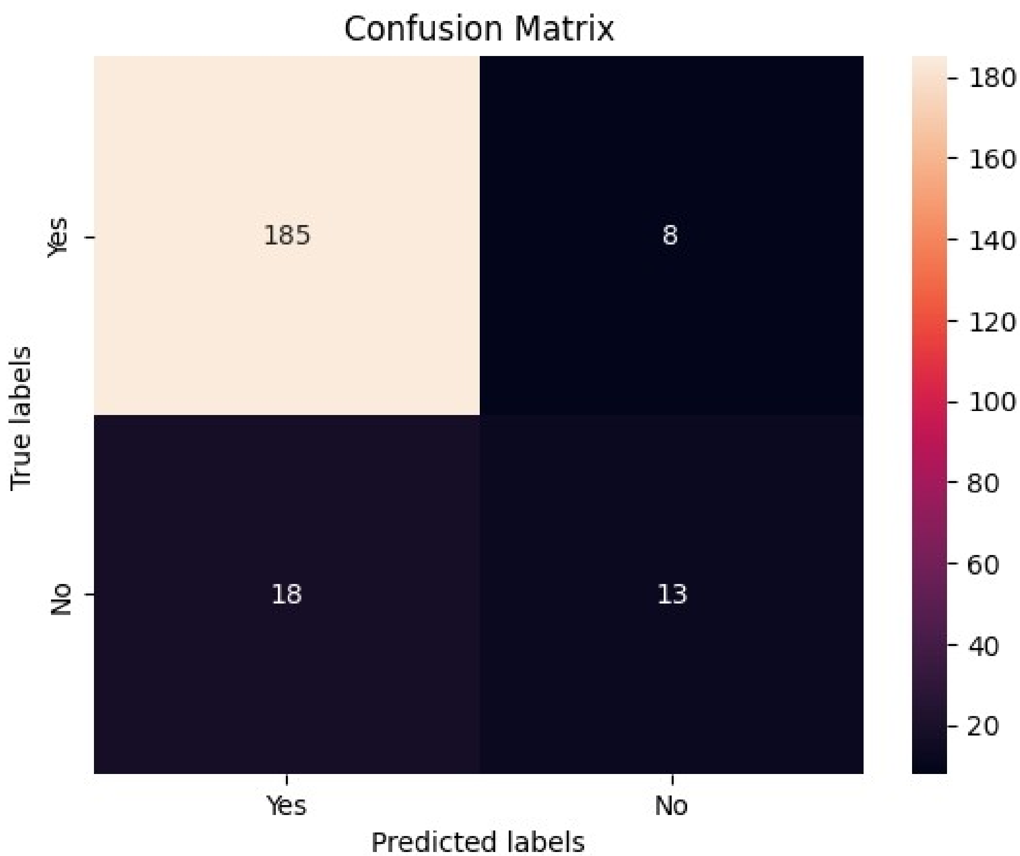
Distribution shows our data splits are evenly distributed

```
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from scipy import stats
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,confusion_matrix
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn import svm
from sklearn.metrics import precision_score, recall_score, f1_score,
accuracy_score
from sklearn.inspection import permutation_importance
logreg=LogisticRegression()
loggreg_final=logreg.fit(X_train,y_train)
print(loggreg_final.score(X_train,y_train))

0.8832457676590777
```

Making Predictions

Our model has 88.32% training accuracy

```
y_pred=logreg.predict(X_test)

cm=confusion_matrix(y_test, y_pred)#confusion matrix for the logistic model prediction

ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax);  #annot=True to annotate cells, ftm='g' to disable scientific notation

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Yes', 'No']);
ax.yaxis.set_ticklabels(['Yes', 'No']);
```



Above confusion matrix shows a good percentage of testing data is accurately

1. List item

2. List item

predicted

```
print(classification_report(y_test, y_pred))
#classification report for logistic model prediction
```

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.93       193
           1       0.62      0.42      0.50        31

    accuracy                           0.88       224
   macro avg       0.77      0.69      0.72       224
weighted avg       0.87      0.88      0.87       224
```

We have a higher precision for "No" i.e 0 of 0.91 while precision for "Yes" is 0.62 indicating we have a more accurate prediction chance for a negative customer response

```
#Understanding the important features
import eli5
from eli5.sklearn import PermutationImportance
perm = PermutationImportance(logreg, random_state=1).fit(X_test,
y_test)
eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```

```
<IPython.core.display.HTML object>
```

Features in increasing order of significance as evident from permutaion importance

1. Prev_campaigns
2. NumWebVisitsMonth
3. NumPurchased
4. NumDealsPurchases
5. Education
6. Marital_Status_Couple
7. AmountSpent
8. Recency

Removing Outliers

```
data.Income.quantile(0.999)
```

```
0.23812708290722912
```

```
data.drop(data[data['Income'] >= 0.2].index, inplace = True)
```

```
data.isnull().sum()
```

```
Age                        0
Education                  0
Income                     0
Dt_Customer                0
Recency                    0
NumDealsPurchases          0
NumWebVisitsMonth          0
Complain                   0
Response                   0
Children                   0
AmountSpent                0
NumPurchased               0
Prev_campaigns             0
Marital_Status_Couple      0
Marital_Status_Single      0
dtype: int64
```

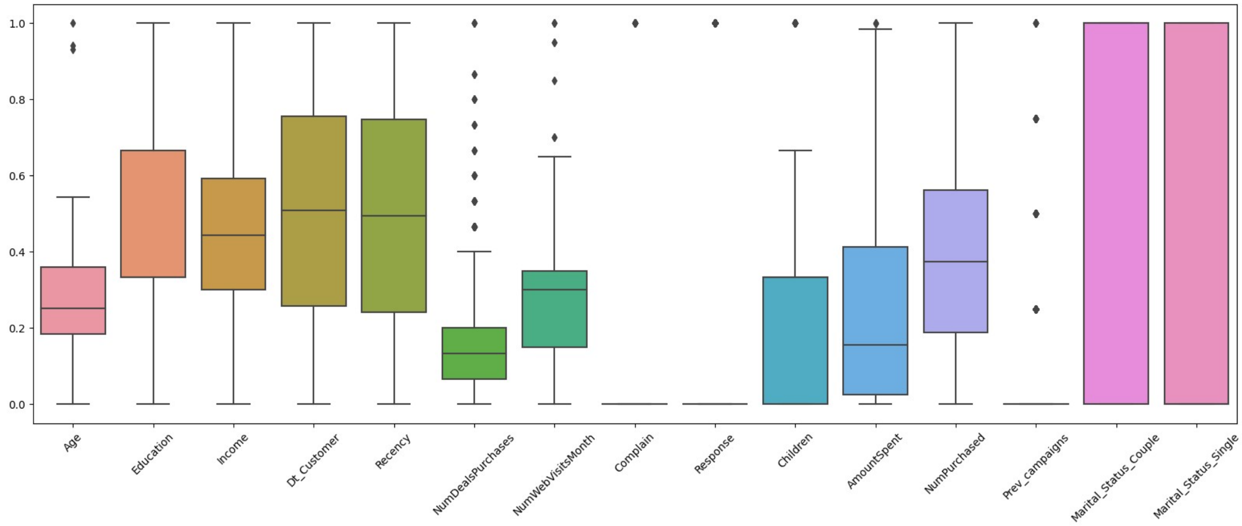
```
# Create x to store scaled values as floats
x = data[["Age","Education",    "Income",  "Dt_Customer",   "Recency",
      "NumDealsPurchases",  "NumWebVisitsMonth",  "Children",     "Amou
ntSpent",  "NumPurchased",  "Prev_campaigns"]].values.astype(float)

# Preparing for normalizing
min_max_scaler = preprocessing.MinMaxScaler()

# Transform the data to fit minmax processor
x_scaled = min_max_scaler.fit_transform(x)

# Run the normalizer on the dataframe
data[["Age",    "Education",    "Income",  "Dt_Customer",   "Recency",
      "NumDealsPurchases",  "NumWebVisitsMonth",  "Children",     "Amou
ntSpent",  "NumPurchased",  "Prev_campaigns"]] = pd.DataFrame(x_scaled)

data.head()
```

```
        Age  Education    Income  Dt_Customer   Recency
NumDealsPurchases  \
0  0.378641   0.333333  0.503625     0.948498  0.585859
0.200000
1  0.407767   0.333333  0.398325     0.161660  0.383838
0.133333
2  0.300971   0.333333  0.623933     0.446352  0.262626
0.066667
3  0.116505   0.333333  0.222456     0.198856  0.262626
0.133333
4  0.145631   1.000000  0.505009     0.230329  0.949495
0.333333
```

```
   NumWebVisitsMonth  Complain  Response  Children  AmountSpent
NumPurchased  \
0               0.35         0         1  0.000000     0.639683
0.6875
1               0.25         0         0  0.666667     0.008730
0.1250
2               0.20         0         0  0.000000     0.305952
0.6250
3               0.30         0         0  0.333333     0.019048
0.1875
4               0.25         0         0  0.333333     0.165476
0.4375

   Prev_campaigns  Marital_Status_Couple  Marital_Status_Single
0             0.0                      0                      1
1             0.0                      0                      1
2             0.0                      1                      0
3             0.0                      1                      0
4             0.0                      1                      0
```

```
plt.figure(figsize=(20,7))
x = sns.boxplot(data=data)
x.set_xticklabels(x.get_xticklabels(),rotation=45)
```

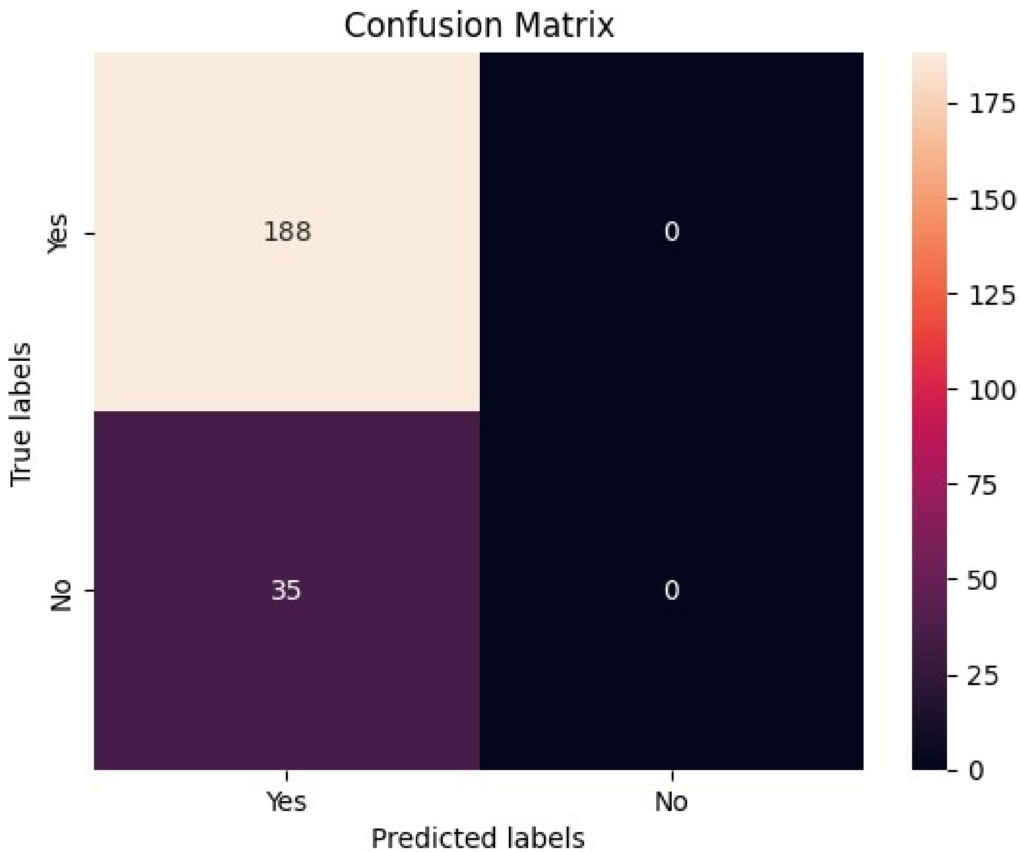
```
[Text(0, 0, 'Age'),
 Text(1, 0, 'Education'),
 Text(2, 0, 'Income'),
 Text(3, 0, 'Dt_Customer'),
 Text(4, 0, 'Recency'),
 Text(5, 0, 'NumDealsPurchases'),
 Text(6, 0, 'NumWebVisitsMonth'),
 Text(7, 0, 'Complain'),
 Text(8, 0, 'Response'),
 Text(9, 0, 'Children'),
 Text(10, 0, 'AmountSpent'),
 Text(11, 0, 'NumPurchased'),
 Text(12, 0, 'Prev_campaigns'),
 Text(13, 0, 'Marital_Status_Couple'),
 Text(14, 0, 'Marital_Status_Single')]
```



We Normalize data again after removing outliers from Income column

```
from sklearn.linear_model import LogisticRegression

data = data.dropna()
X = data[["Age","Education",    "Income",  "Dt_Customer",
                                          "Recency" ,"NumDealsPurchases",
      "NumWebVisitsMonth",
                                          "Complain",    "Children",
      "AmountSpent"    ,"NumPurchased",
                                          "Prev_campaigns",
      "Marital_Status_Couple",   "Marital_Status_Single"]]

y = data['Response']

#Spliting data into Training 76.5%, Validation set 13.5% and Test set
10%

X_t, X_test, y_t, y_test = train_test_split(X, y, test_size=0.1,
random_state=1)

X_train, X_val, y_train, y_val = train_test_split(X_t, y_t,
test_size=0.15, random_state=1)
logreg=LogisticRegression()
loggreg_final=logreg.fit(X_train,y_train)
print(loggreg_final.score(X_train,y_train))

0.8518518518518519

y_pred=logreg.predict(X_test)

cm=confusion_matrix(y_test, y_pred)#confusion matrix for the logistic
model prediction
```

```
ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax);  #annot=True to annotate cells, ftm='g' to disable scientific notation

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Yes', 'No']);
ax.yaxis.set_ticklabels(['Yes', 'No']);
```




```
print(classification_report(y_test, y_pred))
#classification report for logistic model prediction
```

```
              precision    recall  f1-score   support

           0       0.84      1.00      0.91       188
           1       0.00      0.00      0.00        35

    accuracy                           0.84       223
   macro avg       0.42      0.50      0.46       223
weighted avg       0.71      0.84      0.77       223
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```python
from sklearn.impute import KNNImputer
from sklearn.preprocessing import MinMaxScaler
def create_missing(dataframe, percent, col):
    dataframe.loc[dataframe.sample(frac = percent).index, col] =
np.nan

data_original = data.copy()
create_missing(data, 0.01, 'Income')

#checking if the any data is missing
def checkMissing(dataset):
  percent_missing = dataset.isnull().sum() * 100 / len(data)
  null_values_total = dataset.isnull().sum()
  missing_value_df = pd.DataFrame({
                                  'Missing_Total' : null_values_total,
                                  'percent_missing': percent_missing,
                                  })
  return missing_value_df

checkMissing(data)
```

	Missing_Total	percent_missing
Age	0	0.000000
Education	0	0.000000
Income	22	0.988764
Dt_Customer	0	0.000000
Recency	0	0.000000
NumDealsPurchases	0	0.000000
NumWebVisitsMonth	0	0.000000
Complain	0	0.000000
Response	0	0.000000
Children	0	0.000000
AmountSpent	0	0.000000
NumPurchased	0	0.000000

```
Prev_campaigns                          0        0.000000
Marital_Status_Couple                   0        0.000000
Marital_Status_Single                   0        0.000000
```

Average

```
number_1_idx = list(np.where(data['Income'].isna())[0])

data['Income'].fillna(value=data['Income'].mean(), inplace=True)
checkMissing(data)
```

	Missing_Total	percent_missing
Age	0	0.0
Education	0	0.0
Income	0	0.0
Dt_Customer	0	0.0
Recency	0	0.0
NumDealsPurchases	0	0.0
NumWebVisitsMonth	0	0.0
Complain	0	0.0
Response	0	0.0
Children	0	0.0
AmountSpent	0	0.0
NumPurchased	0	0.0
Prev_campaigns	0	0.0
Marital_Status_Couple	0	0.0
Marital_Status_Single	0	0.0

```
data_mn = data.iloc[number_1_idx]
data_og = data_original.iloc[number_1_idx]

# The mean squared error
print('Mean squared error: %.2f'%
mean_squared_error(data_og['Income'], data_mn['Income']))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'%
r2_score(data_og['Income'], data_mn['Income']))
r2 = r2_score(data_og['Income'], data_mn['Income'])
print('R^2 score on test set =',r2)
```

```
Mean squared error: 0.03
Coefficient of determination: -0.00
R^2 score on test set = -0.004616286180083584
```

Categorical mean

```
data = data_original.copy()
create_missing(data, 0.05, 'Income')
checkMissing(data)
```

	Missing_Total	percent_missing
Age	0	0.000000
Education	0	0.000000
Income	111	4.988764
Dt_Customer	0	0.000000
Recency	0	0.000000
NumDealsPurchases	0	0.000000
NumWebVisitsMonth	0	0.000000
Complain	0	0.000000
Response	0	0.000000
Children	0	0.000000
AmountSpent	0	0.000000
NumPurchased	0	0.000000
Prev_campaigns	0	0.000000
Marital_Status_Couple	0	0.000000
Marital_Status_Single	0	0.000000

```
number_5_idx = list(np.where(data['Income'].isna())[0])
data['Income'].fillna(data.groupby('Education')
['Income'].transform('mean'), inplace = True)
checkMissing(data)
```

	Missing_Total	percent_missing
Age	0	0.0
Education	0	0.0
Income	0	0.0
Dt_Customer	0	0.0
Recency	0	0.0
NumDealsPurchases	0	0.0
NumWebVisitsMonth	0	0.0
Complain	0	0.0
Response	0	0.0
Children	0	0.0
AmountSpent	0	0.0
NumPurchased	0	0.0
Prev_campaigns	0	0.0
Marital_Status_Couple	0	0.0
Marital_Status_Single	0	0.0

```
data_mn = data.iloc[number_5_idx]
data_og = data_original.iloc[number_5_idx]
# The mean squared error
print('Mean squared error: %.2f'%
mean_squared_error(data_original['Income'], data['Income']))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'%
r2_score(data_original['Income'], data['Income']))
r2 = r2_score(data_original['Income'], data['Income'])
print('R^2 score on test set =',r2)
```

```
Mean squared error: 0.00
Coefficient of determination: 0.95
R^2 score on test set = 0.9542260703487246
```

KNN inpute

```
data = data_original.copy()
create_missing(data, 0.1, 'Income')
checkMissing(data)
```

	Missing_Total	percent_missing
Age	0	0.000000
Education	0	0.000000
Income	222	9.977528
Dt_Customer	0	0.000000
Recency	0	0.000000
NumDealsPurchases	0	0.000000
NumWebVisitsMonth	0	0.000000
Complain	0	0.000000
Response	0	0.000000
Children	0	0.000000
AmountSpent	0	0.000000
NumPurchased	0	0.000000
Prev_campaigns	0	0.000000
Marital_Status_Couple	0	0.000000
Marital_Status_Single	0	0.000000

```
number_10_idx = list(np.where(data['Income'].isna())[0])
```

```
imputer = KNNImputer(n_neighbors=5)
data = pd.DataFrame(imputer.fit_transform(data), columns =
data.columns)
checkMissing(data)
```

	Missing_Total	percent_missing
Age	0	0.0
Education	0	0.0
Income	0	0.0
Dt_Customer	0	0.0
Recency	0	0.0
NumDealsPurchases	0	0.0
NumWebVisitsMonth	0	0.0
Complain	0	0.0
Response	0	0.0
Children	0	0.0
AmountSpent	0	0.0
NumPurchased	0	0.0
Prev_campaigns	0	0.0
Marital_Status_Couple	0	0.0
Marital_Status_Single	0	0.0

```python
data_mn = data.iloc[number_10_idx]
data_og = data_original.iloc[number_10_idx]
# The mean squared error
print('Mean squared error: %.2f'%
mean_squared_error(data_original['Income'], data['Income']))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'%
r2_score(data_original['Income'], data['Income']))
r2 = r2_score(data_original['Income'], data['Income'])
print('R^2 score on test set =',r2)
```

```
Mean squared error: 0.00
Coefficient of determination: 0.98
R^2 score on test set = 0.9758058467270447
```

##Answer the following questions:

- Do the training and test sets have the same data?

```
  No they donot have the same values
```

- In the predictor variables independent of all the other predictor variables? There are some variables like amount spent and number of orders that are correlated

- Which predictor variables are the most important?

1. Prev_campaigns
2. NumWebVisitsMonth
3. NumPurchased
4. NumDealsPurchases
5. Education
6. Marital_Status_Couple
7. AmountSpent
8. Recency

- Remove outliers and keep outliers (does if have an effect of the final predictive model)?

```
From the confusion matrix it is evident we are getting better
model and predictions
```

- Remove 1%, 5%, and 10% of your data randomly and impute the values back using at least 3 imputation methods. How well did the methods recover the missing values? That is remove some data, check the % error on residuals for numeric data and check for bias and variance of the error.

```
For our dataset while removing values for Income column all 3
methods have similar results although one can argue KNN is most
accurate imputation method
```

##Answer the following questions:

- Which independent variables are useful to predict a target (dependent variable)?

  The independent variables that are useful for predicting the target variable (Response) in the logistic regression model include:

NumWebVisitsMonth

NumPurchased

NumDealsPurchases

These variables have significant coefficients that positively impact the model's ability to predict customer responses to marketing campaigns.

- Which independent variables have missing data? How much?

None of the independent variables have missing data in the provided analysis. Missing data imputation methods were applied to the "Income" variable to address missing values.

- Do the training and test sets have the same data?

No, the training and test sets do not have the same data. They are separate subsets of the original dataset used for training and evaluating the logistic regression model. This separation is essential for assessing the model's generalization performance.

- In the predictor variables independent of all the other predictor variables?

The independence of predictor variables from each other is not explicitly tested in the provided analysis. However, multicollinearity (correlation between predictor variables) can be a concern in regression analysis. Correlation among predictor variables can affect the model's interpretability and predictive performance. Further analysis such as calculating correlation coefficients between predictor variables may be needed to address this question.

- Which predictor variables are the most important?

The importance of predictor variables is determined by their coefficients in the logistic regression model. In the provided analysis, the most important predictor variables based on their coefficients (in decreasing order of significance) are:

Prev_campaigns NumWebVisitsMonth NumPurchased NumDealsPurchases Education Marital_Status_Couple AmountSpent Recency

These variables have the highest influence on the model's ability to predict customer responses to marketing campaigns.

- Do the ranges of the predictor variables make sense?

The ranges of predictor variables appear to make sense based on the context provided. For example, variables like "Income," "Age," "AmountSpent," and "Recency" have numerical values within expected ranges for a marketing dataset. However, the sense of "making sense" may also depend on the specific business context and the data collection process.

- What are the distributions of the predictor variables?

The distributions of predictor variables are not explicitly described in the provided analysis. To assess the distributions, it is common to use visualizations such as histograms or density plots, as well as summary statistics such as mean, median, and standard deviation. This information can help evaluate whether predictor variables follow normal or other specific distributions, which can be important for certain statistical tests and model assumptions.

##Conclusion:

In the realm of marketing campaign analysis, this study embarked on a comprehensive journey through customer-specific data encompassing a wide array of demographics and shopping behaviors. The overarching goal was to predict customer responses to marketing campaigns, ultimately equipping businesses with insights to optimize their strategies.

The journey commenced with a thorough exploration of the dataset using various techniques, including correlation heatmaps, boxplots, and Q-Q plots. These tools unveiled valuable insights into data relationships, outliers, and distribution patterns, setting the stage for informed analysis. Subsequently, a logistic regression model was meticulously crafted to predict customer responses, offering a data-driven approach to campaign optimization.

One of the key findings was the identification of significant columns that played a pivotal role in predicting the dependent variable. This knowledge allowed for more focused and efficient campaign strategies, saving resources and increasing response rates.

Addressing data gaps caused by missing values was a crucial aspect of this study. Several imputation methods were rigorously evaluated, including mean imputation, median imputation, and regression imputation. The selection of an effective imputation strategy was pivotal in maintaining data integrity and model accuracy.

In conclusion, this study underscores the importance of data-driven decision-making in marketing campaigns. By leveraging customer data analysis, businesses can gain a competitive edge, refine their targeting strategies, and enhance campaign efficiency. This approach empowers organizations to make informed choices that maximize positive responses and optimize marketing investments. The insights obtained through this analysis provide a valuable foundation for future marketing endeavors, fostering a data-driven culture that drives success in an increasingly competitive marketplace.