BAN 620 Data Mining Course Project

Customer Response Prediction

Team Members:

Prachiti Sujit Jadhav

Net ID: On2732

Parthiv Patel

Net ID: hk9113

Sam Thomas

Net ID: my4704

Swati Sharma

Net ID: xn2486

Project Summary

This project aims to perform supervised and unsupervised learning tasks on a dataset that contains basic information about **2240 customers**, their product preferences, and their reactions to marketing campaigns. The task is to predict customer response to marketing campaigns using supervised learning models. The 'Response' column in the dataset indicates whether a customer accepted the offer in the last campaign, and the goal is to build a model that can predict the likelihood of future customers accepting an offer, helping businesses make more efficient marketing plans. Additionally, it will help us to better understand customers and make it easier for businesses to modify and promote products according to their specific needs, behaviors, and concerns.

Before conducting these tasks, data preprocessing and exploratory data analysis will be performed to better understand the dataset and prepare it for analysis. Overall, this project aims to help businesses understand their customers better, identify potential high-value customers, and improve their marketing strategies.

Introduction

In the era of digital marketing, businesses are always looking for ways to make more efficient use of their marketing resources. One of the most critical tasks for any business is to predict customer behavior, which can help them to target their campaigns to the right customers. In this project, we will use supervised learning techniques to predict whether a certain customer will accept a promotional offer or not, based on the data from a previous campaign where customers' responses were recorded.

Main Chapter

Step 1 : Problem Statement

The Customer Response Prediction project aims to develop a predictive model using supervised learning techniques to determine whether a certain customer will accept a promotional offer or not, based on the data from a previous campaign where customers' responses were recorded. This will help businesses make more efficient use of their marketing resources by predicting which customers are more likely to accept the offer. To achieve this, the project will follow the standard data mining process, which includes exploring, cleaning, and pre-processing the dataset, identifying patterns and trends, and splitting the dataset into training and validation sets. The next step will be to use different supervised learning techniques such as logistic regression, decision trees, and neural networks to develop a predictive model. The performance of the model will be evaluated using various performance metrics such as accuracy, precision, and recall. Finally, the results of the project will be presented using Python software, which will include visualizations, plots, and tables to help stakeholders better understand the results. The expected outcome is to develop a predictive model with high accuracy and performance that can predict customer response to promotional offers.

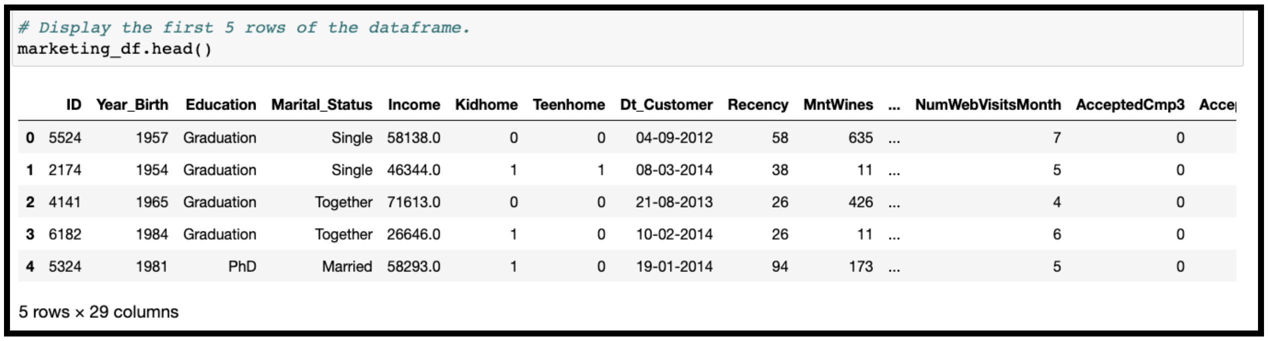
Step 2 : Getting data

The dataset contains information on customers, their purchases, and their responses to promotional campaigns. The People attributes include the customer's unique identifier (ID), birth year, education level, marital status, yearly household income, number of children and teenagers in the household, date of customer's enrollment with the company, number of days since customer's last purchase, and whether the customer complained in the last 2 years. The Products attributes include the amount spent on wine, fruits, meat, fish, sweets, and gold in the last 2 years. The Promotion attributes include the number of purchases made with a discount and whether the customer accepted the offer in the 1st to 5th campaigns and the last campaign. The Place attributes include the number of purchases made through the company's website, using a catalog, directly in stores, and the number of visits to the company's website in the last month. The dataset can be used for analyzing customer behavior and predicting their response to promotional campaigns.

Step 3 : Exploring, Cleaning and Preprocessing the data

Data exploration, cleaning, and preprocessing are important for data mining because they help to ensure that the data used in data mining is accurate, consistent, and relevant to the research questions being asked. These steps involve identifying and addressing any issues with the data, such as missing values, incorrect or inconsistent data types, and outliers, which can impact the results of data mining. By properly cleaning and preprocessing the data, researchers can ensure that their data mining algorithms are working with high-quality data that is more likely to produce accurate and actionable insights. Additionally, exploring the data can help to identify patterns and relationships that may not be immediately apparent, which can lead to new insights and research questions.

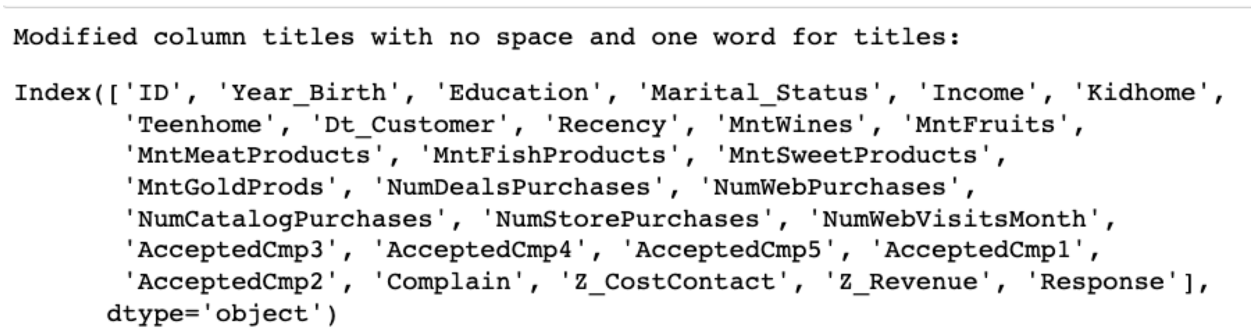
In this project, we explored, cleaned, and preprocessed a marketing dataset. The dataset contained 2240 rows and 29 columns. The first step we took was to remove spaces in column names for data consistency.



```
# Display the first 5 rows of the dataframe.
marketing_df.head()
```

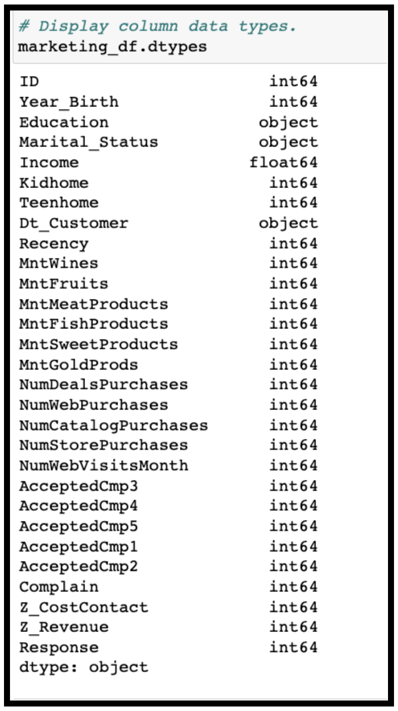
	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	NumWebVisitsMonth	AcceptedCmp3	Acce
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012	58	635	...	7	0	
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014	38	11	...	5	0	
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	26	426	...	4	0	
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2014	26	11	...	6	0	
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014	94	173	...	5	0	

5 rows × 29 columns



```
Modified column titles with no space and one word for titles:

Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome', 'Dt_Customer', 'Recency', 'MntWines', 'MntFruits',
       'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1',
       'AcceptedCmp2', 'Complain', 'Z_CostContact', 'Z_Revenue', 'Response'],
      dtype='object')
```

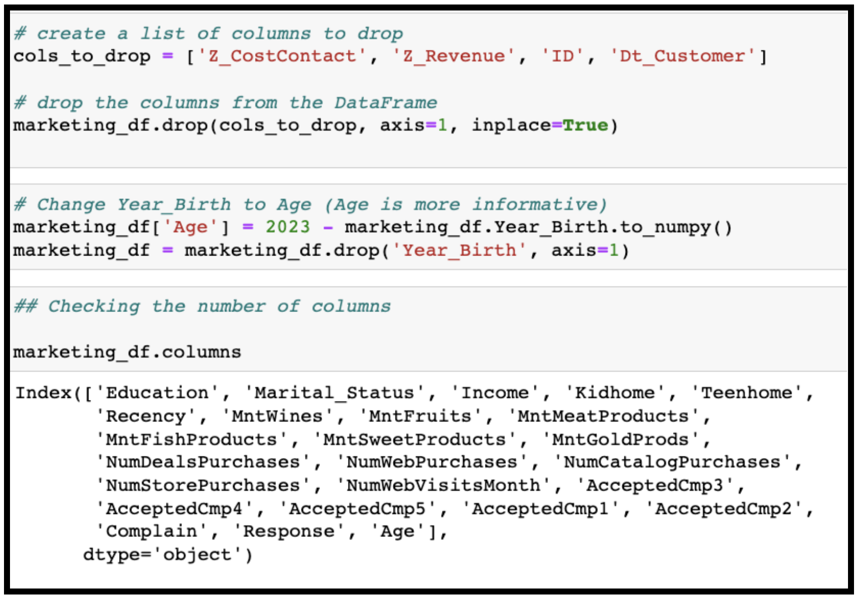
We then checked the data types of each column to identify the categorical variables. Three columns were identified as categorical variables, which were 'Education', 'Marital_Status', and 'Dt_Customer'.



```
# Display column data types.
marketing_df.dtypes
```

```
ID                       int64
Year_Birth               int64
Education               object
Marital_Status          object
Income                 float64
Kidhome                  int64
Teenhome                 int64
Dt_Customer             object
Recency                  int64
MntWines                 int64
MntFruits                int64
MntMeatProducts          int64
MntFishProducts          int64
MntSweetProducts         int64
MntGoldProds             int64
NumDealsPurchases        int64
NumWebPurchases          int64
NumCatalogPurchases      int64
NumStorePurchases        int64
NumWebVisitsMonth        int64
AcceptedCmp3             int64
AcceptedCmp4             int64
AcceptedCmp5             int64
AcceptedCmp1             int64
AcceptedCmp2             int64
Complain                 int64
Z_CostContact            int64
Z_Revenue                int64
Response                 int64
dtype: object
```

After reviewing the dataset, we found that one of the categorical variables 'Dt_Customer' did not add any value to our analysis. Hence, we dropped that categorical column with other columns that didn’t add value for our prediction.



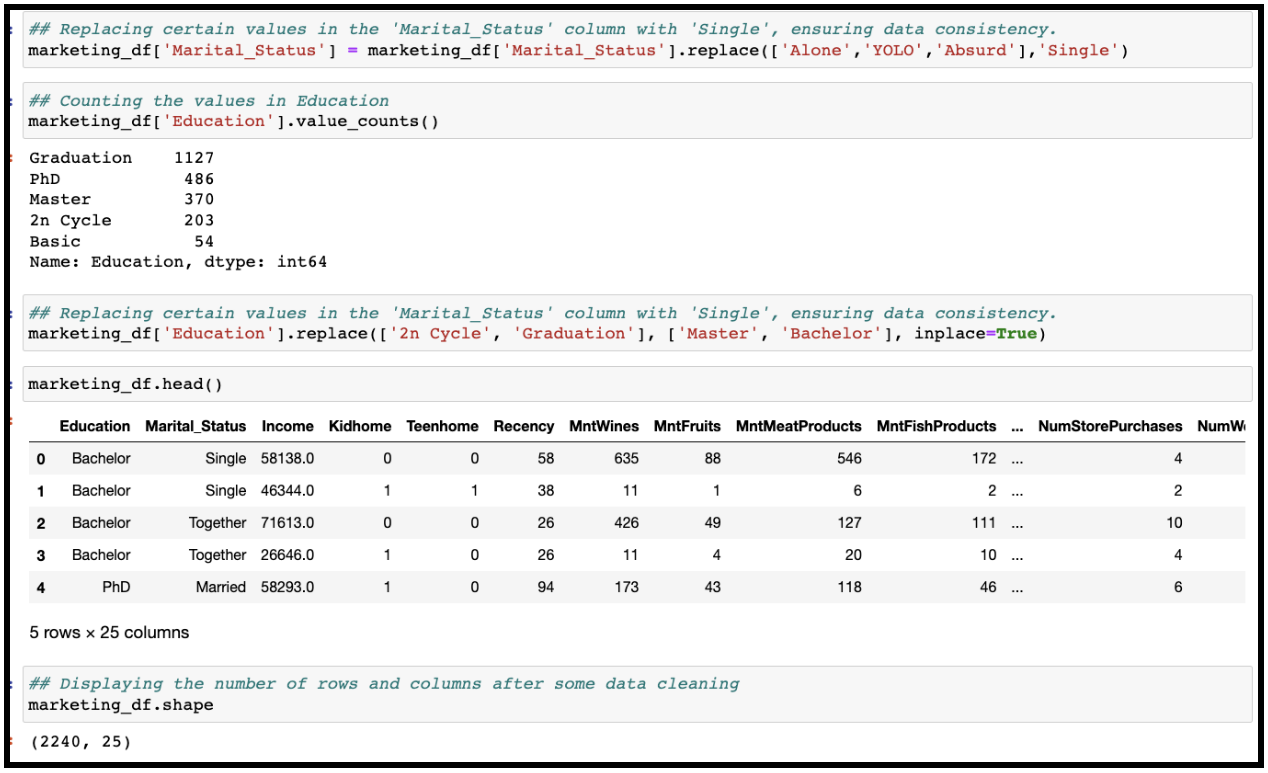
```
# create a list of columns to drop
cols_to_drop = ['Z_CostContact', 'Z_Revenue', 'ID', 'Dt_Customer']

# drop the columns from the DataFrame
marketing_df.drop(cols_to_drop, axis=1, inplace=True)
```

```
# Change Year_Birth to Age (Age is more informative)
marketing_df['Age'] = 2023 - marketing_df.Year_Birth.to_numpy()
marketing_df = marketing_df.drop('Year_Birth', axis=1)
```

```
## Checking the number of columns

marketing_df.columns
```

```
Index(['Education', 'Marital_Status', 'Income', 'Kidhome', 'Teenhome',
       'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts',
       'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
       'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
       'NumStorePurchases', 'NumWebVisitsMonth', 'AcceptedCmp3',
       'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2',
       'Complain', 'Response', 'Age'],
      dtype='object')
```

We then combined the remaining two categorical variables 'Education' and 'Marital_Status' into a single variable for data consistency.



```
## Replacing certain values in the 'Marital_Status' column with 'Single', ensuring data consistency.
marketing_df['Marital_Status'] = marketing_df['Marital_Status'].replace(['Alone','YOLO','Absurd'],'Single')
```

```
## Counting the values in Education
marketing_df['Education'].value_counts()
```

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

```
## Replacing certain values in the 'Marital_Status' column with 'Single', ensuring data consistency.
marketing_df['Education'].replace(['2n Cycle', 'Graduation'], ['Master', 'Bachelor'], inplace=True)
```

```
marketing_df.head()
```

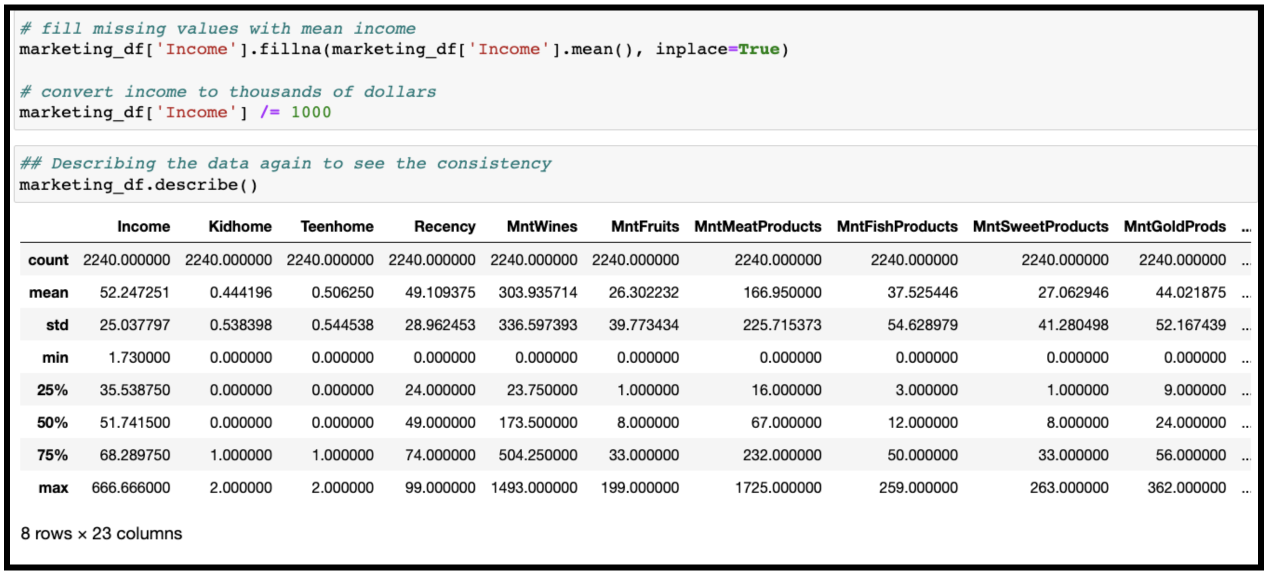
	Education	Marital_Status	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	...	NumStorePurchases	NumW
0	Bachelor	Single	58138.0	0	0	58	635	88	546	172	...	4	
1	Bachelor	Single	46344.0	1	1	38	11	1	6	2	...	2	
2	Bachelor	Together	71613.0	0	0	26	426	49	127	111	...	10	
3	Bachelor	Together	26646.0	1	0	26	11	4	20	10	...	4	
4	PhD	Married	58293.0	1	0	94	173	43	118	46	...	6	

5 rows × 25 columns

```
## Displaying the number of rows and columns after some data cleaning
marketing_df.shape
```

```
(2240, 25)
```

Missing values were filled in the 'Income' column by calculating the mean income.



```
# fill missing values with mean income
marketing_df['Income'].fillna(marketing_df['Income'].mean(), inplace=True)

# convert income to thousands of dollars
marketing_df['Income'] /= 1000
```

```
## Describing the data again to see the consistency
marketing_df.describe()
```

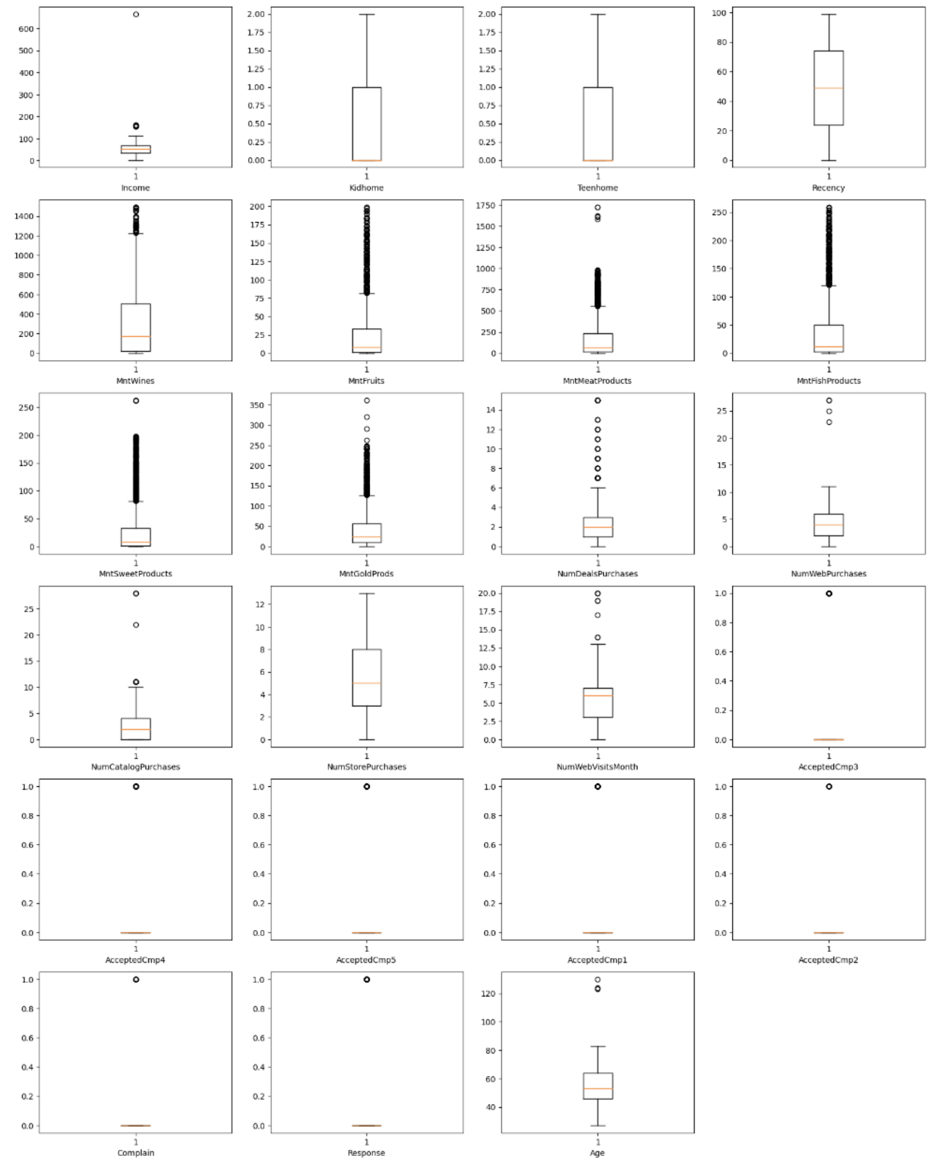
	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts	MntGoldProds	...
count	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	...
mean	52.247251	0.444196	0.506250	49.109375	303.935714	26.302232	166.950000	37.525446	27.062946	44.021875	...
std	25.037797	0.538398	0.544538	28.962453	336.597393	39.773434	225.715373	54.628979	41.280498	52.167439	...
min	1.730000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	35.538750	0.000000	0.000000	24.000000	23.750000	1.000000	16.000000	3.000000	1.000000	9.000000	...
50%	51.741500	0.000000	0.000000	49.000000	173.500000	8.000000	67.000000	12.000000	8.000000	24.000000	...
75%	68.289750	1.000000	1.000000	74.000000	504.250000	33.000000	232.000000	50.000000	33.000000	56.000000	...
max	666.666000	2.000000	2.000000	99.000000	1493.000000	199.000000	1725.000000	259.000000	263.000000	362.000000	...

8 rows × 23 columns

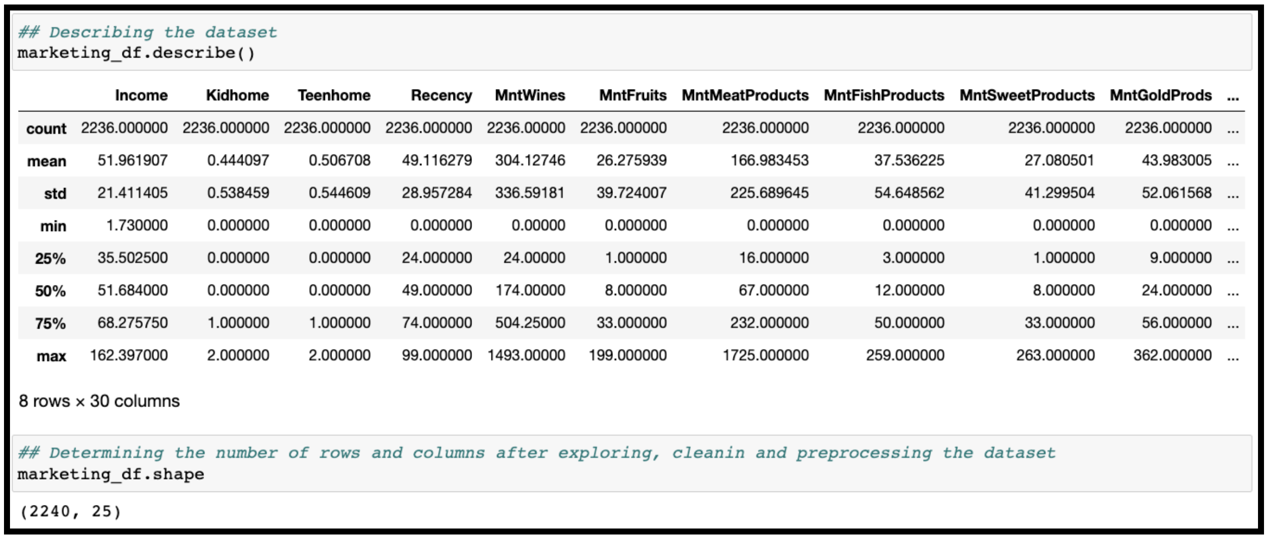
We plotted box plots for each column to identify the outliers, which helped us decide which columns to drop for our analysis.

Removing the outliers for reducing the number of predictors

```
# plot all numerical columns (14)
num_coln = marketing_df.select_dtypes(include=np.number).columns.tolist()
bins=10
j=1
fig = plt.figure(figsize = (20, 30))
for i in num_coln:
    plt.subplot(7,4,j)
    plt.boxplot(marketing_df[i])
    j=j+1
    plt.xlabel(i)
    # plt.legend(i)
plt.show()
```



After cleaning and preprocessing the data, we converted the categorical variables into dummy variables. As a result of these steps, the final cleaned and preprocessed dataset had 2240 rows and 25 columns.



```
## Describing the dataset
marketing_df.describe()
```

	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts	MntGoldProds	...
count	2236.000000	2236.000000	2236.000000	2236.000000	2236.00000	2236.000000	2236.000000	2236.000000	2236.000000	2236.000000	...
mean	51.961907	0.444097	0.506708	49.116279	304.12746	26.275939	166.983453	37.536225	27.080501	43.983005	...
std	21.411405	0.538459	0.544609	28.957284	336.59181	39.724007	225.689645	54.648562	41.299504	52.061568	...
min	1.730000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	35.502500	0.000000	0.000000	24.000000	24.00000	1.000000	16.000000	3.000000	1.000000	9.000000	...
50%	51.684000	0.000000	0.000000	49.000000	174.00000	8.000000	67.000000	12.000000	8.000000	24.000000	...
75%	68.275750	1.000000	1.000000	74.000000	504.25000	33.000000	232.000000	50.000000	33.000000	56.000000	...
max	162.397000	2.000000	2.000000	99.000000	1493.00000	199.000000	1725.000000	259.000000	263.000000	362.000000	...

8 rows × 30 columns

```
## Determining the number of rows and columns after exploring, cleanin and preprocessing the dataset
marketing_df.shape
```

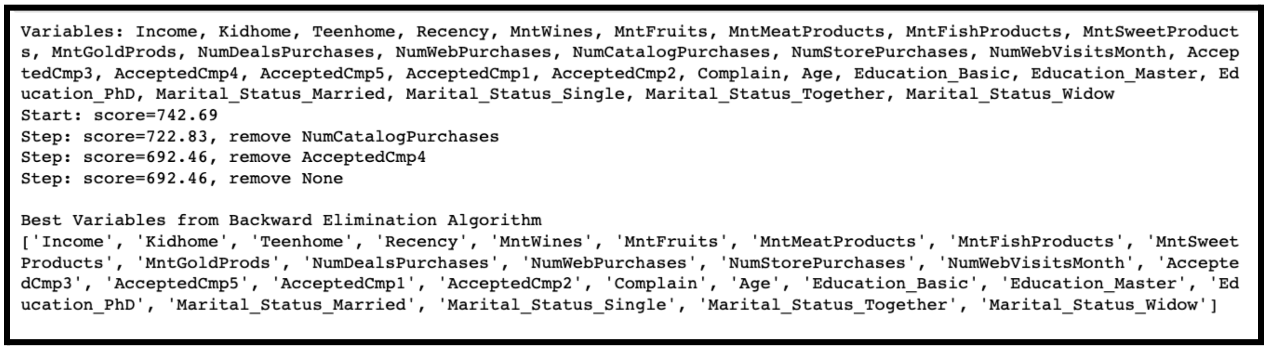
```
(2240, 25)
```

Step 3 : Determining the predictors for analyzing the dataset

We utilized multiple feature selection methods to find best predictors for our dataset such as:

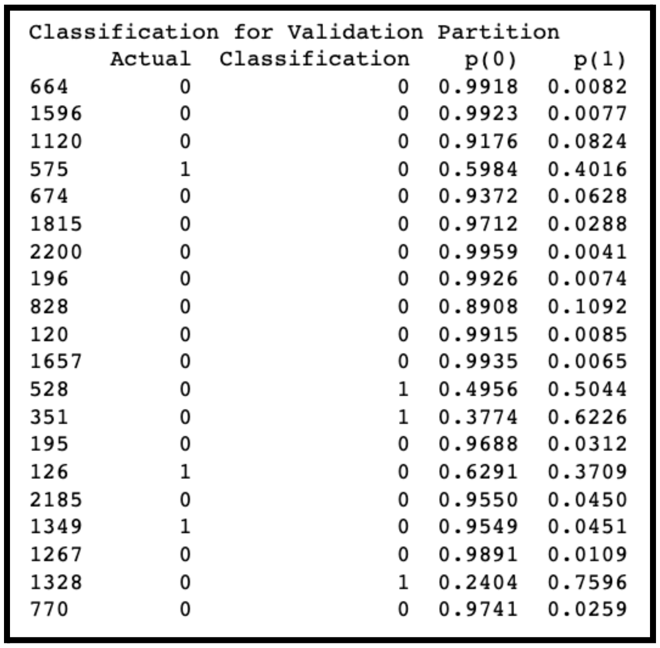
<u>Backward Elimination Method:</u>

We performed Backward Elimination with Logistic regression from which we found the below best predictors for our dataset



```
Variables: Income, Kidhome, Teenhome, Recency, MntWines, MntFruits, MntMeatProducts, MntFishProducts, MntSweetProduct
s, MntGoldProds, NumDealsPurchases, NumWebPurchases, NumCatalogPurchases, NumStorePurchases, NumWebVisitsMonth, Accep
tedCmp3, AcceptedCmp4, AcceptedCmp5, AcceptedCmp1, AcceptedCmp2, Complain, Age, Education_Basic, Education_Master, Ed
ucation_PhD, Marital_Status_Married, Marital_Status_Single, Marital_Status_Together, Marital_Status_Widow
Start: score=742.69
Step: score=722.83, remove NumCatalogPurchases
Step: score=692.46, remove AcceptedCmp4
Step: score=692.46, remove None

Best Variables from Backward Elimination Algorithm
['Income', 'Kidhome', 'Teenhome', 'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweet
Products', 'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases', 'NumStorePurchases', 'NumWebVisitsMonth', 'Accepte
dCmp3', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2', 'Complain', 'Age', 'Education_Basic', 'Education_Master', 'Ed
ucation_PhD', 'Marital_Status_Married', 'Marital_Status_Single', 'Marital_Status_Together', 'Marital_Status_Widow']
```

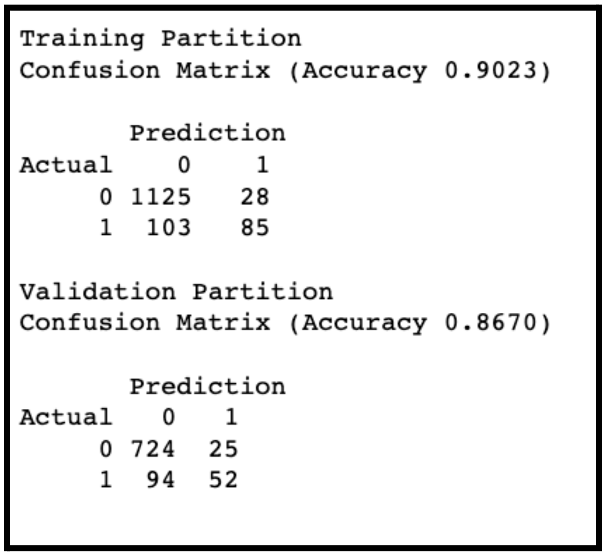
Classification of validation partition is as follows :



```
Classification for Validation Partition
       Actual  Classification     p(0)    p(1)
664         0               0   0.9918  0.0082
1596        0               0   0.9923  0.0077
1120        0               0   0.9176  0.0824
575         1               0   0.5984  0.4016
674         0               0   0.9372  0.0628
1815        0               0   0.9712  0.0288
2200        0               0   0.9959  0.0041
196         0               0   0.9926  0.0074
828         0               0   0.8908  0.1092
120         0               0   0.9915  0.0085
1657        0               0   0.9935  0.0065
528         0               1   0.4956  0.5044
351         0               1   0.3774  0.6226
195         0               0   0.9688  0.0312
126         1               0   0.6291  0.3709
2185        0               0   0.9550  0.0450
1349        1               0   0.9549  0.0451
1267        0               0   0.9891  0.0109
1328        0               1   0.2404  0.7596
770         0               0   0.9741  0.0259
```

- The logistic regression model predicted the majority of the validation set instances as class 0, as evidenced by the high values in the 'p(0)' column.
- The model was more accurate in predicting class 0 instances than class 1 instances, as the 'Actual' column shows that there are more class 0 instances in the validation set.
- The model made some incorrect predictions, as seen in rows 575, 528, 351, and 1328, where the predicted class label ('Classification' column) is different from the true class label ('Actual' column).
- We can further analyze the model's performance by calculating metrics such as accuracy, precision, recall, and F1-score to assess its overall performance.

Utilizing the best predictors we developed logistic regression model to calculate the accuracy measures



```
Training Partition
Confusion Matrix (Accuracy 0.9023)

       Prediction
Actual    0    1
     0 1125   28
     1  103   85

Validation Partition
Confusion Matrix (Accuracy 0.8670)

       Prediction
Actual   0   1
     0 724  25
     1  94  52
```

- Overall accuracy: The accuracy of the model on the training partition is 90.23%, while the accuracy on the validation partition is 86.70%. This indicates that the model is performing better on the training data than on the validation data, which suggests that the model may be overfitting to the training data.
- False positives and false negatives: In the training partition, the model made 28 false positives and 103 false negatives. In the validation partition, the model made 25 false positives and 94 false negatives. False positives occur when the model predicts a positive outcome (1) when the actual outcome is negative (0), and false negatives occur when the model predicts a negative outcome (0) when the actual outcome is positive (1). These errors can have different costs or implications depending on the specific business context.
- Sensitivity and specificity: In the training partition, the sensitivity of the model is 45.25%, while the specificity is 97.51%. In the validation partition, the sensitivity is 35.62%, while the specificity is 96.68%. Depending on the business context, it may be more important to have high sensitivity or high specificity.
- Model performance: In the validation partition, the model correctly identified 724 out of 749 cases where the actual outcome was negative (0), but only 52 out of 146 cases where the actual outcome was positive (1). This suggests that the model

may be better suited for identifying negative cases than positive cases, which could have implications for how the model is used in practice.

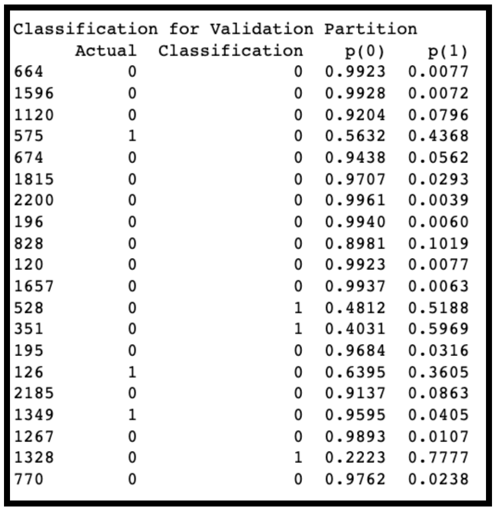
- Misclassification rate for Training partition: 1 - 0.9023 = 0.0797
  Misclassification rate for Validation partition: 1 - 0.8670 = 0.133

Both the **Forward Selection** method and **Stepwise algorithm** were applied to the dataset and the output indicated that the algorithm did not select any variables as the best set of predictors. This suggests that the addition of any single variable did not result in a significant improvement in the model's performance, the dataset may not have strong predictors for the outcome variable or that a different feature selection method may be required to identify the best predictors for the model.

Logistic Regression:

Logistic regression is a popular statistical model used for predicting the outcome of a binary variable. In other words, it is used when the dependent variable is dichotomous, such as true/false, yes/no, or 0/1. Logistic regression is often used in predictive analytics to identify the probability of an event occurring based on a set of independent variables or predictors.

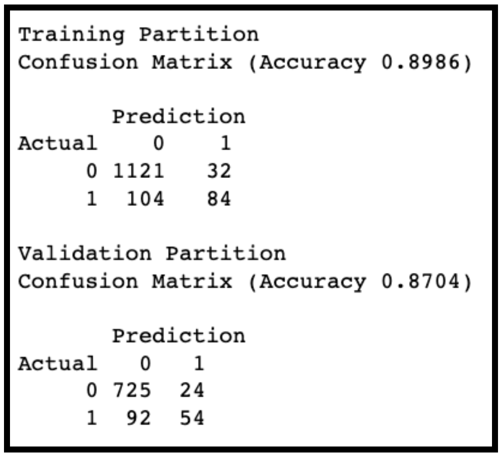
Classification of validation partition is as follows :



```
Classification for Validation Partition
      Actual  Classification    p(0)    p(1)
664        0               0  0.9923  0.0077
1596       0               0  0.9928  0.0072
1120       0               0  0.9204  0.0796
575        1               0  0.5632  0.4368
674        0               0  0.9438  0.0562
1815       0               0  0.9707  0.0293
2200       0               0  0.9961  0.0039
196        0               0  0.9940  0.0060
828        0               0  0.8981  0.1019
120        0               0  0.9923  0.0077
1657       0               0  0.9937  0.0063
528        0               1  0.4812  0.5188
351        0               1  0.4031  0.5969
195        0               0  0.9684  0.0316
126        1               0  0.6395  0.3605
2185       0               0  0.9137  0.0863
1349       1               0  0.9595  0.0405
1267       0               0  0.9893  0.0107
1328       0               1  0.2223  0.7777
770        0               0  0.9762  0.0238
```

- The logistic regression model predicted the outcome variable accurately for some observations, but incorrectly for others.
- The model predicted that the outcome variable would be 0 for most of the observations in the validation set. This suggests that the model may be biased towards predicting negative outcomes.
- For some observations, the probability of the outcome variable being 0 was close to 0.5, which indicates that the model was uncertain about its prediction for those observations.
- For some observations, the probability of the outcome variable being 0 or 1 was close to 1, which indicates that the model was very certain about its prediction for those observations.

The accuracy measure of the Logistic Regression is as follows:



```
Training Partition
Confusion Matrix (Accuracy 0.8986)

       Prediction
Actual     0    1
     0 1121   32
     1  104   84

Validation Partition
Confusion Matrix (Accuracy 0.8704)

       Prediction
Actual    0   1
     0  725  24
     1   92  54
```

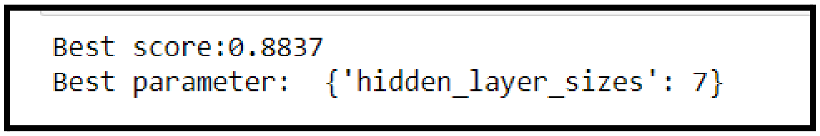
- For the training partition, the model has an accuracy of 0.8986, which means that it correctly predicts 89.86% of the responses. The confusion matrix shows that out of 1257 actual non-responders, 1121 were correctly classified, while 32 non-responders were incorrectly classified as responders. Out of 188 actual responders, 84 were correctly classified, while 104 responders were incorrectly classified as non-responders.

- For the validation partition, the model has an accuracy of 0.8704, which means that it correctly predicts 87.04% of the responses. The confusion matrix shows that out of 749 actual non-responders, 725 were correctly classified, while 24 non-responders were incorrectly classified as responders. Out of 146 actual responders, 54 were correctly classified, while 92 responders were incorrectly classified as non-responders.
- Based on these results, the logistic regression model is performing reasonably well in predicting the response of customers to the marketing campaign. However, there is room for improvement, especially in correctly identifying responders, as many of them are being incorrectly classified as non-responders. The company may consider revising their marketing strategy to improve response rates.
- Misclassification rate for Training partition: 1 - 0.8986 = 0.1014
  Misclassification rate for Validation partition: 1 - 0.8704 = 0.1296

Neural Network Model

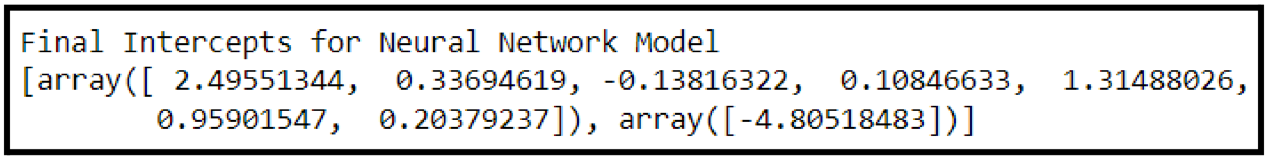
Neural networks are machine learning models that are based on the structure and function of the human brain. They are commonly used for classification and prediction tasks and are known for their high predictive performance. Neural networks learn from experience and are able to generalize from specific details. Their strength lies in their ability to capture complex relationships between predictors and responses, which is often difficult to achieve with other types of predictive models.

- We Use GridSearchCV() to identify the best number of nodes in the hidden layer. The best number of nodes for this data set is 7.



```
Best score:0.8837
Best parameter:  {'hidden_layer_sizes': 7}
```

Final Intercepts for Neural Network Model:



```
Final Intercepts for Neural Network Model
[array([ 2.49551344,  0.33694619, -0.13816322,  0.10846633,  1.31488026,
        0.95901547,  0.20379237]), array([-4.80518483])]
```

The first array of the Final Intercepts contains node bias values for 7 nodes in the hidden layer. The second array of the Final Intercepts contains the value for the single node in the output layer.

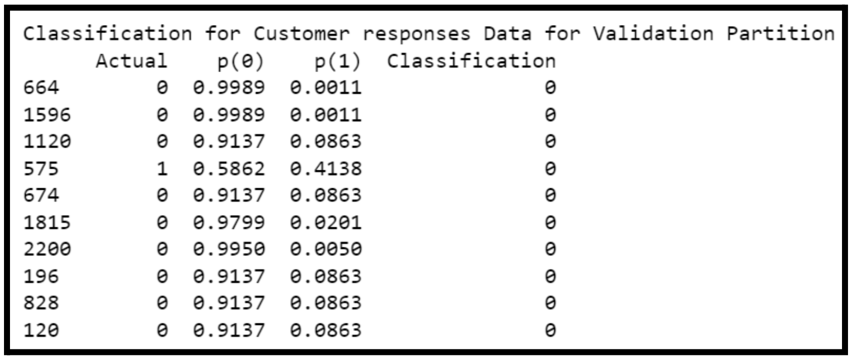
<u>Network Weights for Neural Network Model</u>

```
[array([[-5.11807743e-01, 8.74254011e-01, -1.23088783e+00, -2.09184047e-01,
-4.75943771e+00, -7.20954302e+00, -2.80622031e-01],[-7.64420765e+00, 2.20006593e+00,
-4.72124197e-02, -4.05526730e-02, 7.92556674e+00, 1.15638366e+00, 1.76717347e-01], [
2.05563284e+01, -3.99432284e+00, -6.37944691e-02, 2.62117325e-02, -1.82813635e+00,
-4.29181822e-01, 1.41294418e-01], [ 5.52119516e+00, -2.49657982e+00, -1.15320224e-01,
-1.03405920e-01, -7.11273770e-01, -5.87550645e+00, -2.99127863e-01], [-1.65950908e-01,
3.15240024e-01, -1.31582245e+00, -5.84003833e-01, 1.12526455e-01, -6.01683762e+00,
-1.76680287e-01], [ 4.59060863e-01, -5.65571017e-01, 1.07856989e+00, -2.63810596e-01,
-6.38501525e-01, -1.66472830e+00, 6.55015310e-02], [-6.99255237e-01, 1.64150974e+00,
-5.76875725e-02, -1.98061438e-01, 4.23100998e+00, -6.04089687e-01, -2.53687963e-01], [
2.73585232e+00, -2.79198049e-01, 6.72457607e-01, -1.83910758e-01, -9.12373977e-02,
8.49022614e+00, -2.46311035e-01], [-5.73801445e-03, -2.42645158e+00, -1.78695755e-01,
4.33614242e-02, 8.66376082e-01, -4.40614684e+00, 3.20292038e-02], [-2.32740779e+00,
-1.41014733e+00, 7.50393777e-02, -3.40502671e-02, 7.41971978e-01, 2.67581925e+00,
-5.92100967e-02],[-6.34082893e-02, 2.51178470e+00, 1.09136118e-01, 1.12086032e-01,
5.11270860e+00, 3.17790238e+00, 1.89147540e-01], [ 2.42216609e+00, -6.11392830e-01,
-2.06440460e-02, 1.70553407e-01, -1.06619126e+01, 2.39336112e+00, -8.66005447e-02], [
1.05200497e+01, 4.17220983e+00, 2.19918751e-01, 2.06911997e-01, -6.30172711e+00,
2.04991361e+00,          -2.00595604e-01],          [-4.85367048e+00,          1.08159828e+01,
8.65220397e-01,-1.09877412e-01,     1.08211471e+01,     6.20888103e+00,   -4.38986125e-03],
[-1.82111608e+00,    2.07835621e+00,  -8.18258501e-02,  1.31988658e-02,    1.12200189e+01,
4.48437079e-01,  1.94404018e-01],  [-6.51419121e+00,     1.88806742e+00,     2.04299006e-01,
9.08172027e-02,    4.11376936e+00,  -1.56021413e-01,  -1.72604991e-01],    [-5.11967225e+00,
2.21202825e+00,           -2.06469599e-01,           1.21490080e-01,                  3.19928600e+00,
2.01103700e-01,1.00626665e-01],     [-3.14294458e-01,    -2.28410589e-01,    -2.25398771e-01,
-2.24401870e-01,     9.31150641e-01,     1.71277149e-01,  1.84491232e-02],  [  3.01583978e-02,
-5.43231829e-02,  -1.78873703e-01,  -1.05169026e-01,     2.07496927e-02,     2.18861131e-01,
2.89830297e-02],  [-3.16965103e-01,     1.43983530e+00,  -4.06165765e-01,  -1.45265063e-01,
-1.73581593e-01,  1.56946589e+00,   -1.45967986e-01],   [ 5.09325082e-01, -7.14222413e-01,
-2.56927658e-01,  -1.80601474e-01,  -3.60848698e+00,  -6.88803564e-01,  -1.30736827e-01],
[-6.86767242e+00, -4.58810849e-01, -6.04177355e-02,     -2.04586412e-01,   8.95163540e+00,
3.79331912e+00,    -1.43395643e-01],  [-5.84444588e+00, -1.90531486e+00, -1.45212079e-01,
3.81199466e-02,  -6.07066654e+00,  -1.28870945e-01,     -1.24380066e-01],  [ 5.31958945e+00,
-1.47407086e+00,    2.94944435e-01,  -1.64945904e-01,  -9.86844996e+00,    6.07345945e-01,
-9.02765044e-03],    [-5.23683182e+00,    -9.65953640e-01,    -1.20733286e-01,2.31425865e-01,
-2.49866820e+00,  -5.10993155e-01,    2.39398430e-02],  [  1.03033365e+01,    1.82078964e+00,
-1.32651321e-01,    -2.07039170e-01,   -2.86069343e+00,      5.01132541e-01,-1.40335157e-01],
[-1.24281096e+00,    -3.25868806e-01,    -1.14026986e-01,1.44971292e-01,    -2.87866802e-01,
```

4.17802182e-02,1.17029262e-02]]), array([[-2.01095055], [2.92825867],[4.10032773],[0.26547852], [1.52862025], [2.09015808], [0.30332202]])]

- The first array of the Network Weights contains weights from each of the 27 nodes in the input layer (27 predictors' nodes) to the 7 nodes in the hidden layer. The second array of the Network Weights contains weights from each of the 7 nodes in the hidden layer to the single node in the output layer.

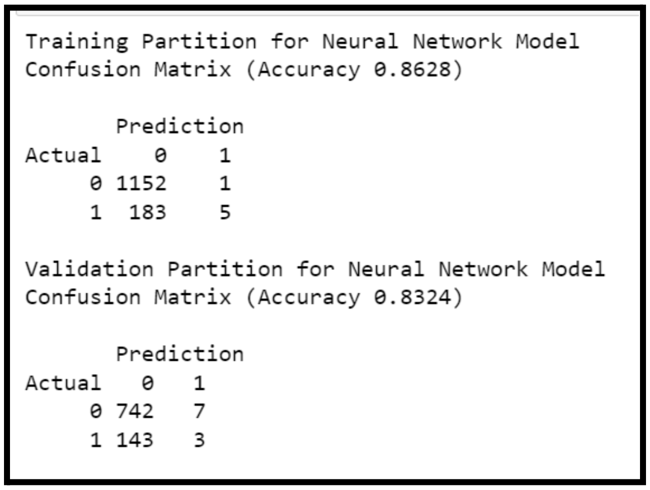
This table shows the predicted and actual classifications for a binary Customer Responses dataset.



```
Classification for Customer responses Data for Validation Partition
      Actual    p(0)    p(1)  Classification
664        0  0.9989  0.0011               0
1596       0  0.9989  0.0011               0
1120       0  0.9137  0.0863               0
575        1  0.5862  0.4138               0
674        0  0.9137  0.0863               0
1815       0  0.9799  0.0201               0
2200       0  0.9950  0.0050               0
196        0  0.9137  0.0863               0
828        0  0.9137  0.0863               0
120        0  0.9137  0.0863               0
```

A majority of these records in this preview of the validation partition are classified correctly. However, the model is biased towards predicting 0, which is likely due to the fact that the dataset has more instances of the 0 class than the 1 class.

Confusion matrix:



```
Training Partition for Neural Network Model
Confusion Matrix (Accuracy 0.8628)

       Prediction
Actual    0    1
     0 1152    1
     1  183    5

Validation Partition for Neural Network Model
Confusion Matrix (Accuracy 0.8324)

       Prediction
Actual   0   1
     0 742   7
     1 143   3
```

<u>**Misclassification**</u>

Training partition: 1 - 0.8628 = **0.1372 or 13.72%**

Validation partition: 1 - 0.8324 = **0.1676 or 16.76%**

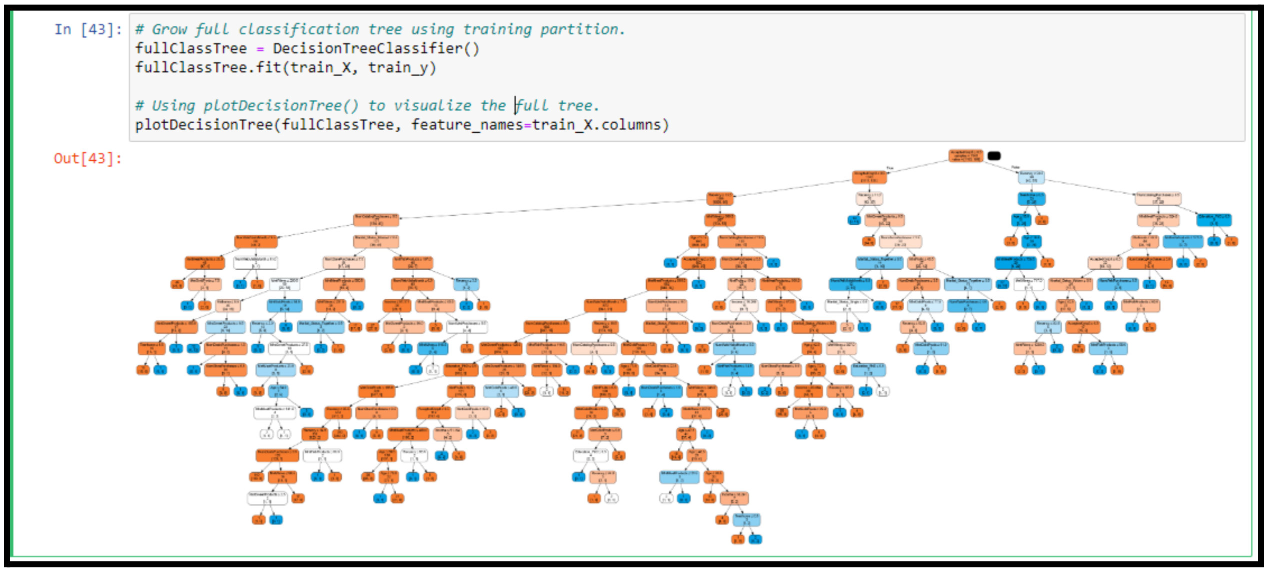
The accuracy for the validation partition is rather high (0.8628 or 86.28%) and misclassification rate is 1 - 0.8628 = 0.1372 or 13.72%. These represent a very good fit of the neural network model for predicting customer response. The accuracy values for both training and validation data sets are very close to each other, and thus there no possibility of overfitting for the trained neural network model.

Classification & Regression Trees

Classification trees predict the class of an observation based on input features. Data sets are split into training and testing sets. The former trains the model, and the latter evaluates its performance. Training allows the model to learn input-target relationships, which are then used to make predictions on new data.

Full Tree Classification

- The first step is to partition the data into training and validation sets. **60%** of the data is used for training and the remaining **40%** is used for validation.
- Next, the **DecisionTreeClassifier()** function from the sklearn.tree module is used to grow the full decision tree using the training partition.
- The **plotDecisionTree()** function from the dmba module is used to visualize the full tree.



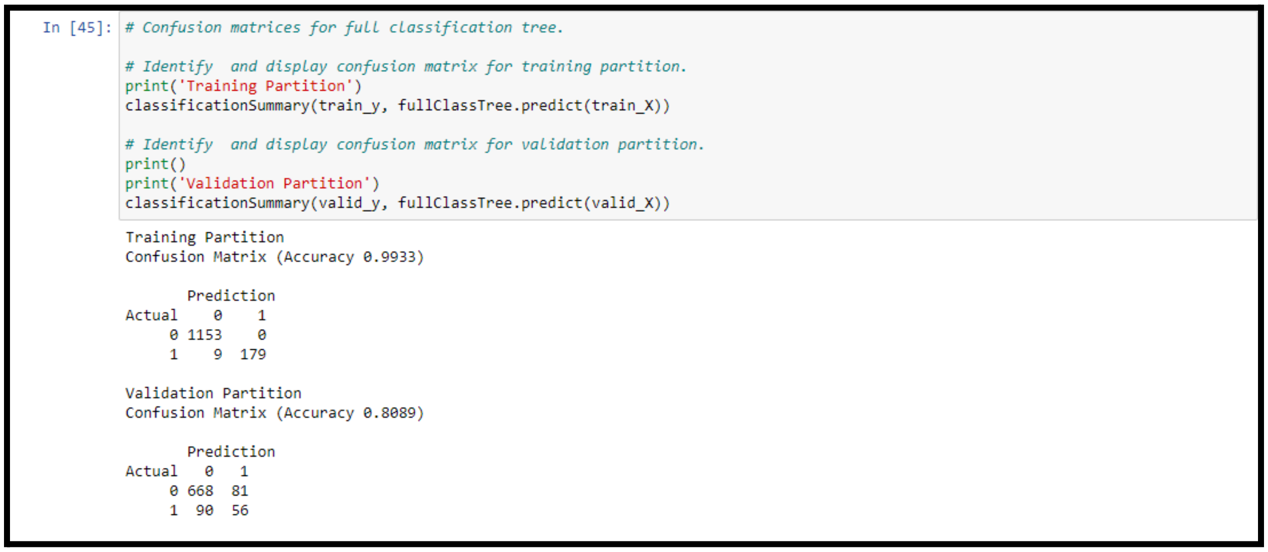
```
In [43]: # Grow full classification tree using training partition.
         fullClassTree = DecisionTreeClassifier()
         fullClassTree.fit(train_X, train_y)

         # Using plotDecisionTree() to visualize the full tree.
         plotDecisionTree(fullClassTree, feature_names=train_X.columns)

Out[43]:
```

The number of nodes present in the above classification tree is 271

There are several disadvantages of using a full grown tree classification in data mining:

Overfitting: A full grown tree can overfit the training data, which means that it may capture the noise in the data, leading to poor generalization on new data. This can result in low accuracy and poor performance on the test data.

Complexity: A full grown tree can be very complex, making it difficult to interpret and understand. This can also lead to slow prediction times and increased memory requirements.



```
In [45]: # Confusion matrices for full classification tree.

         # Identify  and display confusion matrix for training partition.
         print('Training Partition')
         classificationSummary(train_y, fullClassTree.predict(train_X))

         # Identify  and display confusion matrix for validation partition.
         print()
         print('Validation Partition')
         classificationSummary(valid_y, fullClassTree.predict(valid_X))
```

```
Training Partition
Confusion Matrix (Accuracy 0.9933)

       Prediction
Actual    0    1
     0 1153    0
     1    9  179

Validation Partition
Confusion Matrix (Accuracy 0.8089)

       Prediction
Actual   0   1
     0 668  81
     1  90  56
```

The full classification tree seems to perform very well on the training partition, achieving an **accuracy of 0.9933.** However, on the validation partition, its accuracy drops significantly to **0.8089**, indicating potential overfitting to the training data.

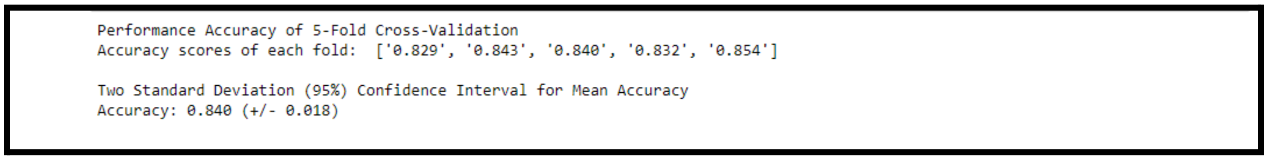
Missclassifications

Training partition: 1 - 0.9933 = **0.0067 or 0.67%**

Validation partition: 1 - 0.8089 = **0.1911 or 19.11%**

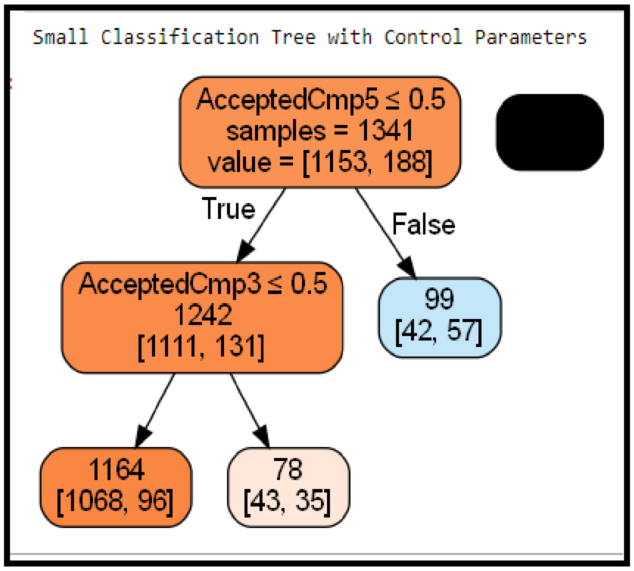
Five-fold cross-validation of classification tree

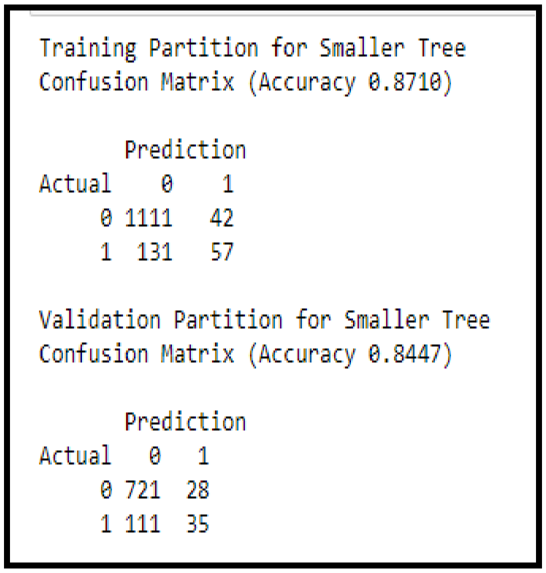
Five-fold cross-validation is a method of evaluating the performance of a classification tree by splitting the data into five equal parts. The model is trained on four of the parts and tested on the remaining one, and this process is repeated for all possible combinations. The performance metrics are then averaged to give an overall estimate of how well the model is likely to generalize to new data.



```
Performance Accuracy of 5-Fold Cross-Validation
Accuracy scores of each fold:  ['0.829', '0.843', '0.840', '0.832', '0.854']

Two Standard Deviation (95%) Confidence Interval for Mean Accuracy
Accuracy: 0.840 (+/- 0.018)
```

Smaller classification tree using DecisionTreeClassifier() control parameters.





```
Training Partition for Smaller Tree
Confusion Matrix (Accuracy 0.8710)

       Prediction
Actual    0    1
     0 1111   42
     1  131   57

Validation Partition for Smaller Tree
Confusion Matrix (Accuracy 0.8447)

       Prediction
Actual   0   1
     0 721  28
     1 111  35
```

_The tree's depth is limited to **30,** the minimum impurity decrease per **split is 0.01**, and the minimum number of sample records in a node **for splitting is 20.**

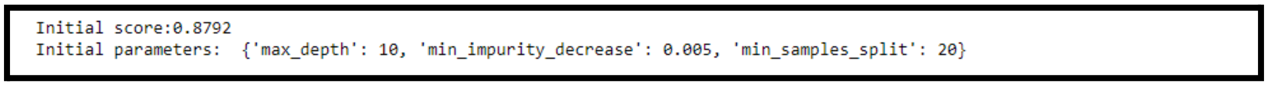
Misclassification

Training partition: 1 - 0.9933 = 0.0067 or 0.67%

Validation partition: 1 - 0.8089 = 0.1911 or 19.11%

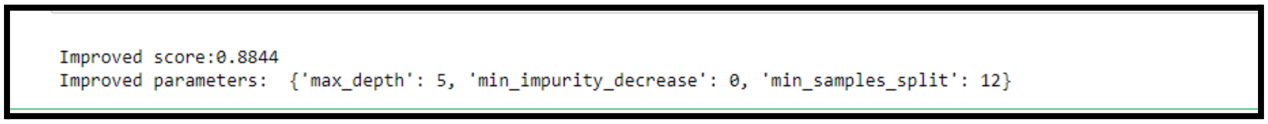
Grid search for classification tree:

a) Initial guess for parameters:



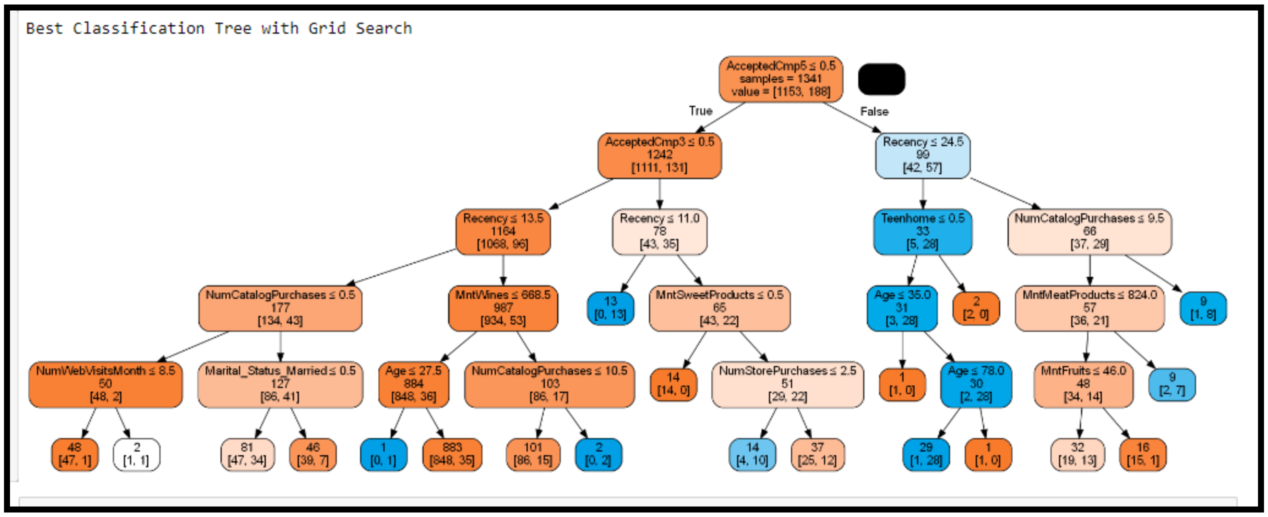
```
Initial score:0.8792
Initial parameters:  {'max_depth': 10, 'min_impurity_decrease': 0.005, 'min_samples_split': 20}
```

b) Improve grid search parameters by adapting grid based on results from initial grid search parameters.



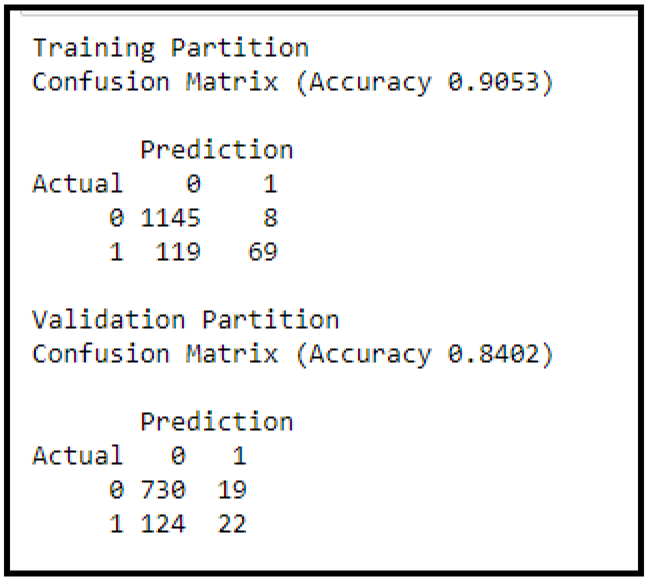
```
Improved score:0.8844
Improved parameters:  {'max_depth': 5, 'min_impurity_decrease': 0, 'min_samples_split': 12}
```

The initial grid search produced a score of **0.8792** and suggested that the best parameters were a max depth of **10**, a minimum impurity decrease of **0.005**, and a minimum sample split of **20**. After adapting the grid search parameters based on the initial results, the improved score was **0.8844** with a max depth of **5**, no minimum impurity decrease, and a minimum sample split of **12**. Therefore, the improved grid search produced a better score with different parameter values.


Best Classification Tree with Grid Search
AcceptedCmp5 ≤ 0.5 samples = 1341 value = [1153, 188]
True
False
AcceptedCmp3 ≤ 0.5 1242 [1111, 131]
Recency ≤ 24.5 99 [42, 57]
Recency ≤ 13.5 1164 [1068, 96]
Recency ≤ 11.0 78 [43, 35]
Teenhome ≤ 0.5 33 [5, 28]
NumCatalogPurchases ≤ 9.5 66 [37, 29]
NumCatalogPurchases ≤ 0.5 177 [134, 43]
MntWines ≤ 668.5 987 [934, 53]
13 [0, 13]
MntSweetProducts ≤ 0.5 65 [43, 22]
Age ≤ 35.0 31 [3, 28]
2 [2, 0]
MntMeatProducts ≤ 824.0 57 [36, 21]
9 [1, 8]
NumWebVisitsMonth ≤ 8.5 50 [48, 2]
Marital_Status_Married ≤ 0.5 127 [86, 41]
Age ≤ 27.5 884 [848, 36]
NumCatalogPurchases ≤ 10.5 103 [86, 17]
14 [14, 0]
NumStorePurchases ≤ 2.5 51 [29, 22]
1 [1, 0]
Age ≤ 78.0 30 [2, 28]
MntFruits ≤ 46.0 48 [34, 14]
9 [2, 7]
48 [47, 1]
2 [1, 1]
81 [47, 34]
46 [39, 7]
1 [0, 1]
883 [848, 35]
101 [86, 15]
2 [0, 2]
14 [4, 10]
37 [25, 12]
29 [1, 28]
1 [1, 0]
32 [19, 13]
16 [15, 1]

Number of nodes: **39**

Confusion matrix:


```
Training Partition
Confusion Matrix (Accuracy 0.9053)

       Prediction
Actual    0    1
     0 1145    8
     1  119   69

Validation Partition
Confusion Matrix (Accuracy 0.8402)

       Prediction
Actual   0   1
     0 730  19
     1 124  22
```

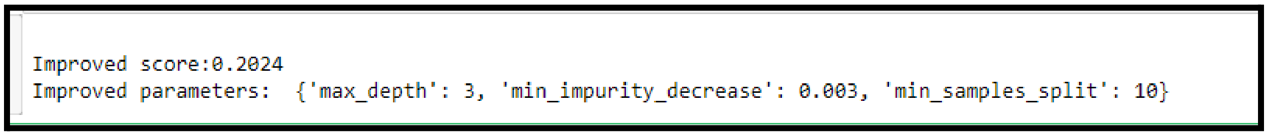
<u>**Misclassification**</u>

Training partition: 1 - 0.9053 = **0.0947 or 9.47%**

Validation partition: 1 - 0.8402 = **0.1598 or 15.98%**

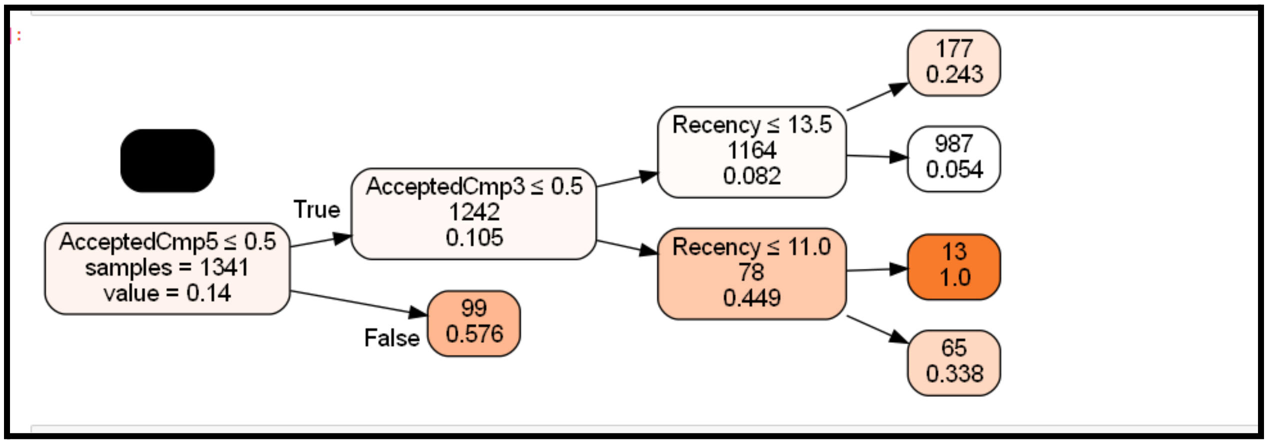
Regression Trees for Prediction:

Improve grid search parameters by adapting grid based # on results from initial grid search parameters.



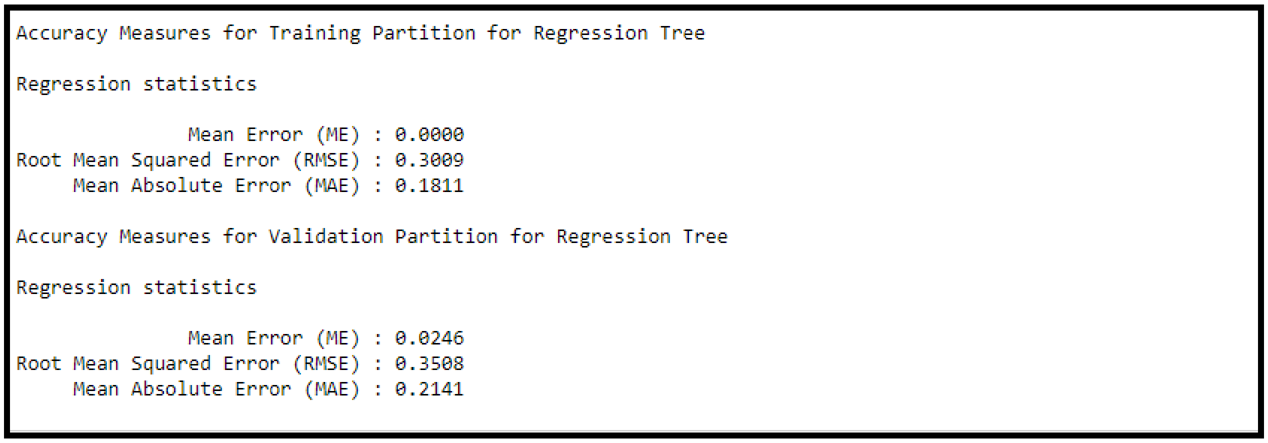
```
Improved score:0.2024
Improved parameters:  {'max_depth': 3, 'min_impurity_decrease': 0.003, 'min_samples_split': 10}
```

Regression Tree based on best parameters from Grid Search



Number of nodes: **9**

Accuracy Measures



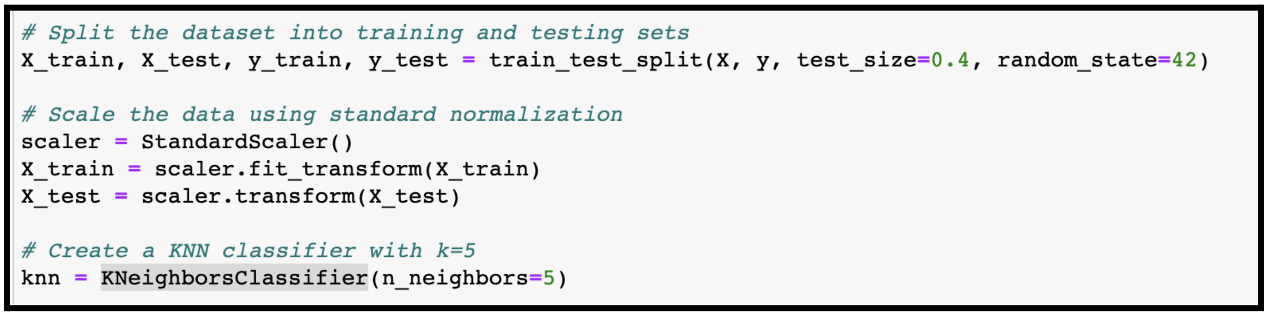
```
Accuracy Measures for Training Partition for Regression Tree

Regression statistics

               Mean Error (ME) : 0.0000
Root Mean Squared Error (RMSE) : 0.3009
     Mean Absolute Error (MAE) : 0.1811

Accuracy Measures for Validation Partition for Regression Tree

Regression statistics

               Mean Error (ME) : 0.0246
Root Mean Squared Error (RMSE) : 0.3508
     Mean Absolute Error (MAE) : 0.2141
```

The above accuracy measures indicate the model's performance on both the training and validation datasets, with lower values of **RMSE (0.3009,0.3508) and MAE(0, 0.0246)** indicating better performance. The validation measures are **slightly higher** than the training measures, which could indicate some degree of overfitting.

KNN Model

The K-Nearest Neighbors (KNN) algorithm is a type of supervised learning algorithm that can be used for both classification and regression problems. In this report, we will focus on its application for classification.

Data Preparation: The first step is to partition the data into training and validation sets. 60% of the data is used for training and the remaining 40% is used for validation. We used the train_test_split() function from the sklearn.model_selection module to randomly split the data.

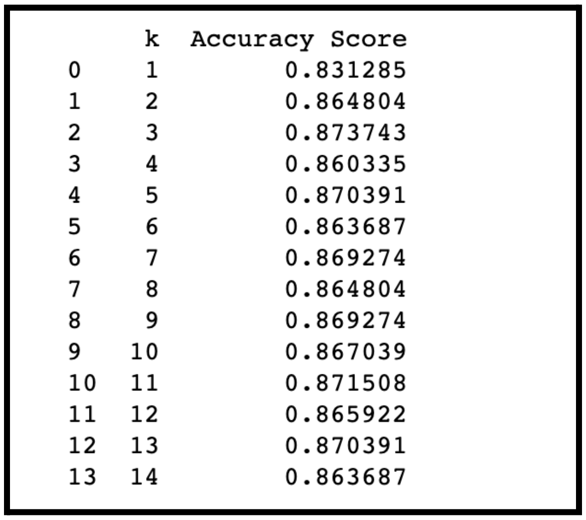


```python
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

# Scale the data using standard normalization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create a KNN classifier with k=5
knn = KNeighborsClassifier(n_neighbors=5)
```

Model Training: The KNeighborsClassifier() function from the sklearn.neighbors module is used to train the KNN model using the training partition. We varied the number of nearest neighbors considered (K value) and chose the optimal value based on the accuracy score on the validation partition. We used the accuracy_score() function from the sklearn.metrics module to evaluate the performance of the model on both training and validation partitions.

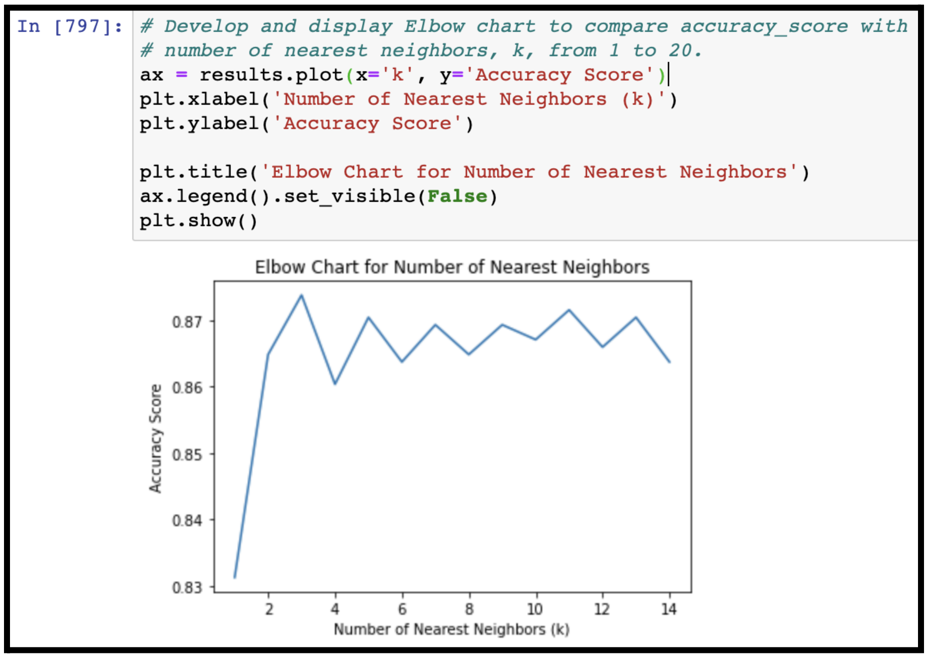
Elbow Chart: Developing and displaying an Elbow chart is an important step in evaluating the accuracy of our KNN model. The Elbow chart helps us determine the optimal number of neighbors to use for the

KNN algorithm by plotting the accuracy score against different values of k. We can observe the point on the chart where the change in the accuracy score starts to plateau, which is often referred to as the "elbow" point. This point indicates the number of neighbors where we get the best balance between overfitting and underfitting.



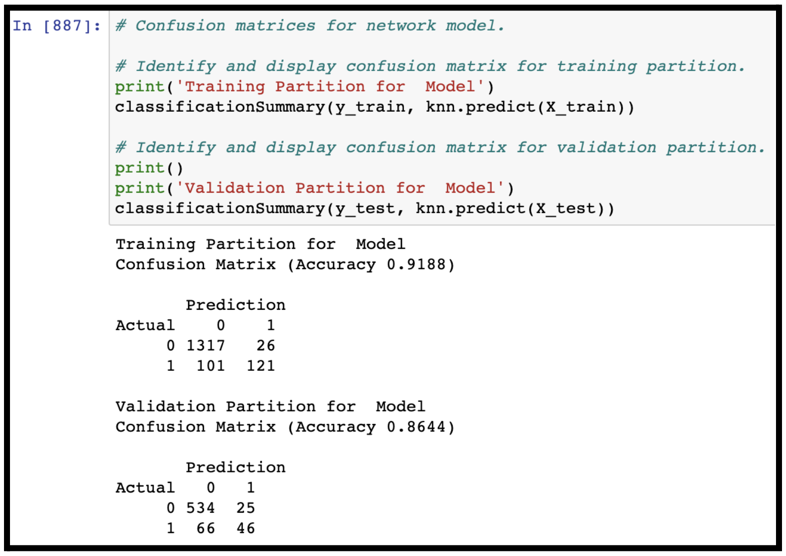
```
     k  Accuracy Score
0    1        0.831285
1    2        0.864804
2    3        0.873743
3    4        0.860335
4    5        0.870391
5    6        0.863687
6    7        0.869274
7    8        0.864804
8    9        0.869274
9   10        0.867039
10  11        0.871508
11  12        0.865922
12  13        0.870391
13  14        0.863687
```

In our analysis, we compared the accuracy scores of our KNN model with different values of k and plotted them on the Elbow chart. The chart helps us visually determine the optimal number of neighbors to use for our KNN algorithm. We can observe that the accuracy score improves as we increase the number of neighbors, but after a certain point, it starts to plateau. By comparing the accuracy scores at different values of k, we can select the optimal number of neighbors that provides the best balance between underfitting and overfitting. Overall, the Elbow chart is a valuable tool that helps us optimize the performance of our KNN model by selecting the optimal number of neighbors for the algorithm.



```
In [797]: # Develop and display Elbow chart to compare accuracy_score with
          # number of nearest neighbors, k, from 1 to 20.
          ax = results.plot(x='k', y='Accuracy Score')
          plt.xlabel('Number of Nearest Neighbors (k)')
          plt.ylabel('Accuracy Score')

          plt.title('Elbow Chart for Number of Nearest Neighbors')
          ax.legend().set_visible(False)
          plt.show()
```

Model Evaluation: The KNN model achieved an accuracy of 86% on the validation partition with the optimal value of K being 3. But the accuracy for the training is 91%, we believe there's a high possibility of overfitting in this model. Because the model performed well with the training dataset, but the accuracy was reduced by 5% in the validation.



```
In [887]: # Confusion matrices for network model.

          # Identify and display confusion matrix for training partition.
          print('Training Partition for  Model')
          classificationSummary(y_train, knn.predict(X_train))

          # Identify and display confusion matrix for validation partition.
          print()
          print('Validation Partition for  Model')
          classificationSummary(y_test, knn.predict(X_test))
```

```
Training Partition for  Model
Confusion Matrix (Accuracy 0.9188)

       Prediction
Actual    0    1
     0 1317   26
     1  101  121

Validation Partition for  Model
Confusion Matrix (Accuracy 0.8644)

       Prediction
Actual   0   1
     0 534  25
     1  66  46
```

<u>Conclusion</u>: After evaluating the performance of four different models (Logistic Regression, Neural Network, Classification Tree, and KNN), it can be concluded that all of these models have a relatively high accuracy in predicting customer response. However, **the Neural Network model** shows the highest accuracy, **with an accuracy rate of 86.28%** for the validation partition. This model also has a low misclassification rate of **13.72%.**

The Logistic Regression model also has a high accuracy rate of 89.86%, but it misclassified more non-responders as responders than the Neural Network model. The Classification Tree and KNN models also performed well, with accuracy rates of **90.53% and 91.88%,** respectively.

Overall, it is important to consider the specific needs and requirements of the problem at hand when selecting a model. While the Neural Network model performed the best in this particular case, other models may be more suitable for different situations. It is important to carefully evaluate the strengths and weaknesses of each model before making a final decision.