Problem Statement:

'All You Need' Supermarket is planning for the year-end sale. They want to launch a new offer - gold membership for only $499 which is \$999 on normal days(that gives a 20% discount on all purchases).

It will be valid only for existing customers, they are planning to start a campaign through phone calls.

The best way to reduce the cost of the campaign is to make a predictive model which will classify customers who might purchase the offer, using the data they gathered during last year's campaign.

We will build a model for classifying whether customers will reply with a positive response or not.

Objective:

- What are the different factors which affect the target variable? What business recommendations can we give based on the analysis?
- How can we improve model performance using hyperparameter tuning and prevent data leakage using pipelines while building a model to predict the response of a customer?

Data Dictionary

- Response (target) - 1 if customer accepted the offer in the last campaign, 0 otherwise
- ID - Unique ID of each customer
- Year_Birth - Age of the customer
- Complain - 1 if the customer complained in the last 2 years
- Dt_Customer - date of customer's enrollment with the company
- Education - customer's level of education
- Marital - customer's marital status
- Kidhome - number of small children in customer's household
- Teenhome - number of teenagers in customer's household
- Income - customer's yearly household income
- MntFishProducts - the amount spent on fish products in the last 2 years
- MntMeatProducts - the amount spent on meat products in the last 2 years
- MntFruits - the amount spent on fruits products in the last 2 years
- MntSweetProducts - amount spent on sweet products in the last 2 years
- MntWines - the amount spent on wine products in the last 2 years
- MntGoldProds - the amount spent on gold products in the last 2 years
- NumDealsPurchases - number of purchases made with discount
- NumCatalogPurchases - number of purchases made using catalog (buying goods to be shipped through the mail)
- NumStorePurchases - number of purchases made directly in stores
- NumWebPurchases - number of purchases made through the company's website
- NumWebVisitsMonth - number of visits to company's website in the last month
- Recency - number of days since the last purchase

Import necessary libraries

In [177…

```
import warnings

warnings.filterwarnings("ignore")
# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

# libaries to help with data visualization
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

# Libraries to tune model, get different metric scores, and split data
from sklearn import metrics
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.impute import KNNImputer
from sklearn.pipeline import Pipeline, make_pipeline

#libraries to help with model building
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (
    AdaBoostClassifier,
    GradientBoostingClassifier,
    RandomForestClassifier)
from xgboost import XGBClassifier
```

Load and view the dataset

In [178…
```
data = pd.read_excel("marketing_data.xlsx")
```

In [82]:
```
data.head()
```

Out[82]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	MntFishProducts	MntSwe
0	1826	1970	Graduation	Divorced	84835.0	0	0	6/16/14	0	189	...	111	
1	1	1961	Graduation	Single	57091.0	0	0	6/15/14	0	464	...	7	
2	10476	1958	Graduation	Married	67267.0	0	1	5/13/14	0	134	...	15	
3	1386	1967	Graduation	Together	32474.0	1	1	2014-11-05 00:00:00	0	10	...	0	
4	5371	1989	Graduation	Single	21474.0	1	0	2014-08-04 00:00:00	0	6	...	11	

5 rows × 22 columns

In [179…
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 22 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  Response             2240 non-null   int64  
 21  Complain             2240 non-null   int64  
dtypes: float64(1), int64(18), object(3)
memory usage: 385.1+ KB
```

- There are a total of 22 columns and 2,240 observations in the dataset
- We can see that income column have less than 2,240 non-null values i.e. column have missing values. We'll explore this further.

Let's check the number of unique values in each column

In [180…
```
data.nunique()
```

Out[180…
```
ID                     2240
Year_Birth               59
Education                 5
Marital_Status            8
```

```
Income                 1974
Kidhome                   3
Teenhome                  3
Dt_Customer             663
Recency                 100
MntWines                776
MntFruits               158
MntMeatProducts         558
MntFishProducts         182
MntSweetProducts        177
MntGoldProds            213
NumDealsPurchases        15
NumWebPurchases          15
NumCatalogPurchases      14
NumStorePurchases        14
NumWebVisitsMonth        16
Response                  2
Complain                  2
dtype: int64
```

- We can drop the column - `ID` as it is unique for each customer and will not add value to the model.

In [181…

```
# Dropping column - ID
data.drop(columns=["ID"], inplace=True)
```

Summary of the data

In [182…

```
data.describe().T
```

Out[182…

	count	mean	std	min	25%	50%	75%	max
Year_Birth	2240.0	1968.805804	11.984069	1893.0	1959.00	1970.0	1977.00	1996.0
Income	2216.0	52247.251354	25173.076661	1730.0	35303.00	51381.5	68522.00	666666.0
Kidhome	2240.0	0.444196	0.538398	0.0	0.00	0.0	1.00	2.0
Teenhome	2240.0	0.506250	0.544538	0.0	0.00	0.0	1.00	2.0
Recency	2240.0	49.109375	28.962453	0.0	24.00	49.0	74.00	99.0
MntWines	2240.0	303.935714	336.597393	0.0	23.75	173.5	504.25	1493.0
MntFruits	2240.0	26.302232	39.773434	0.0	1.00	8.0	33.00	199.0
MntMeatProducts	2240.0	166.950000	225.715373	0.0	16.00	67.0	232.00	1725.0
MntFishProducts	2240.0	37.525446	54.628979	0.0	3.00	12.0	50.00	259.0
MntSweetProducts	2240.0	27.062946	41.280498	0.0	1.00	8.0	33.00	263.0
MntGoldProds	2240.0	44.021875	52.167439	0.0	9.00	24.0	56.00	362.0
NumDealsPurchases	2240.0	2.325000	1.932238	0.0	1.00	2.0	3.00	15.0
NumWebPurchases	2240.0	4.084821	2.778714	0.0	2.00	4.0	6.00	27.0
NumCatalogPurchases	2240.0	2.662054	2.923101	0.0	0.00	2.0	4.00	28.0
NumStorePurchases	2240.0	5.790179	3.250958	0.0	3.00	5.0	8.00	13.0
NumWebVisitsMonth	2240.0	5.316518	2.426645	0.0	3.00	6.0	7.00	20.0
Response	2240.0	0.149107	0.356274	0.0	0.00	0.0	0.00	1.0
Complain	2240.0	0.009375	0.096391	0.0	0.00	0.0	0.00	1.0

- `Year_Birth` has a large range of values i.e. 1893 to 1996.
- Columns - `MntFruits, MntWines, MntMeatProducts, MntFishProducts, MntSweetProducts` might have outliers on the right end as there is a large difference between 75th percentile and maximum values.
- Recency has an approx equal mean and median which is equal to 49.
- Highest mean amount spent in the last two years is on wines (approx 304), followed by meat products (approx 167).
- The distribution of classes in the `Response` variable is imbalanced as most of the values are 0.

Data Preprocessing

Adding age of the customers to the data using given birth years

In [183

In [183...
```python
# To calculate age we'll subtract the year 2016 because variables account for the last 2 years
# and we have customers registered till 2014 only
# We need to convert strings values to dates first to use subtraction
data["Age"] = 2016 - pd.to_datetime(data["Year_Birth"], format="%Y").apply(
    lambda x: x.year
)

data["Age"].sort_values()
```

Out[183...
```
562      20
1824     20
697      21
1468     21
964      21
       ...
1740     75
2171     76
2233    116
827     117
513     123
Name: Age, Length: 2240, dtype: int64
```

- We can see that there are 3 observations with ages greater than 100 i.e. 116, 117 and 123 which is highly unlikely to be true.
- We can cap the value for age variables to the next highest value i.e. 76.

In [184...
```python
# Capping age variable
data["Age"].clip(upper=76, inplace=True)
```

Using Dt_Customer to add features to the data

In [185...
```python
# The feature Dt_Customer represents dates of the customer’s enrollment with the company.
# Let's convert this to DateTime format
data["Dt_Customer"] = pd.to_datetime(data["Dt_Customer"])
```

In [186...
```python
# Extracting registration year from the date
data["Reg_year"] = data["Dt_Customer"].apply(lambda x: x.year)

# Extracting registration quarter from the date
data["Reg_quarter"] = data["Dt_Customer"].apply(lambda x: x.quarter)

# Extracting registration month from the date
data["Reg_month"] = data["Dt_Customer"].apply(lambda x: x.month)

# Extracting registration week from the date
data["Reg_week"] = data["Dt_Customer"].apply(lambda x: x.day // 7)
```

In [187...
```python
data.head()
```

Out[187...

	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	MntFruits	...	NumCatalogPurchases
0	1970	Graduation	Divorced	84835.0	0	0	2014-06-16	0	189	104	...	4
1	1961	Graduation	Single	57091.0	0	0	2014-06-15	0	464	5	...	3
2	1958	Graduation	Married	67267.0	0	1	2014-05-13	0	134	11	...	2
3	1967	Graduation	Together	32474.0	1	1	2014-11-05	0	10	0	...	0
4	1989	Graduation	Single	21474.0	1	0	2014-08-04	0	6	16	...	1

5 rows × 26 columns

Let's check the count of each unique category in each of the categorical variables.

In [188...
```python
# Making a list of all categorical variables
cat_col = [
    "Education",
    "Marital_Status",
    "Kidhome",
    "Teenhome",
    "Complain",
    "Response",
    "Reg_year",
    "Reg_quarter",
    "Reg_month",
```

```python
    "Reg_week",
]

# Printing number of count of each unique value in each column
for column in cat_col:
    print(data[column].value_counts())
    print("-" * 40)
```

```
Graduation    1127
PhD            486
Master         370
2n Cycle       203
Basic           54
Name: Education, dtype: int64
----------------------------------------
Married     864
Together    580
Single      480
Divorced    232
Widow        77
Alone         3
YOLO          2
Absurd        2
Name: Marital_Status, dtype: int64
----------------------------------------
0    1293
1     899
2      48
Name: Kidhome, dtype: int64
----------------------------------------
0    1158
1    1030
2      52
Name: Teenhome, dtype: int64
----------------------------------------
0    2219
1      21
Name: Complain, dtype: int64
----------------------------------------
0    1906
1     334
Name: Response, dtype: int64
----------------------------------------
2013    1189
2014     557
2012     494
Name: Reg_year, dtype: int64
----------------------------------------
4    596
1    580
2    546
3    518
Name: Reg_quarter, dtype: int64
----------------------------------------
8     211
10    209
3     202
12    202
5     192
1     191
2     187
11    185
4     184
6     170
9     166
7     141
Name: Reg_month, dtype: int64
----------------------------------------
1    523
2    500
3    490
0    462
4    265
Name: Reg_week, dtype: int64
----------------------------------------
```

- In education, 2n cycle and Master means the same thing. We can combine these two categories.
- There are many categories in marital status. We can combine the categories 'Alone', 'Absurd' and 'YOLO' with 'Single' and 'Together' categories with 'Married'.
- There are only 21 customers who complained in the last two years.

- We have 1906 observations for the 0 class but only 334 observations for class 1.
- There are only three years in the customer registration data.

In [189…

```
# Replacing 2n Cycle with Master
data["Education"] = data["Education"].replace("2n Cycle", "Master")
```

In [190…

```
# Replacing YOLO, Alone, Absurd with single and Together with Married
data["Marital_Status"] = data["Marital_Status"].replace(
    ["YOLO", "Alone", "Absurd"], "Single"
)
data["Marital_Status"] = data["Marital_Status"].replace(["Together"], "Married")
```

Imputing missing values in income column

In [192…

```
# number of missing values in each column
data.isnull().sum()
```

Out[192…

```
Year_Birth              0
Education               0
Marital_Status          0
Income                 24
Kidhome                 0
Teenhome                0
Dt_Customer             0
Recency                 0
MntWines                0
MntFruits               0
MntMeatProducts         0
MntFishProducts         0
MntSweetProducts        0
MntGoldProds            0
NumDealsPurchases       0
NumWebPurchases         0
NumCatalogPurchases     0
NumStorePurchases       0
NumWebVisitsMonth       0
Response                0
Complain                0
Age                     0
Reg_year                0
Reg_quarter             0
Reg_month               0
Reg_week                0
dtype: int64
```

In [193…

```
# Percentage of missing values in income column
round(data.isna().sum() / data.isna().count() * 100, 2)["Income"]
```

Out[193… 1.07

We can add a column - total amount spent by each customer in the last 2 years

In [194…

```
data["Total_Amount_Spent"] = data[
    [
        "MntWines",
        "MntFruits",
        "MntMeatProducts",
        "MntFishProducts",
        "MntSweetProducts",
        "MntGoldProds",
    ]
].sum(axis=1)
```

EDA

Univariate

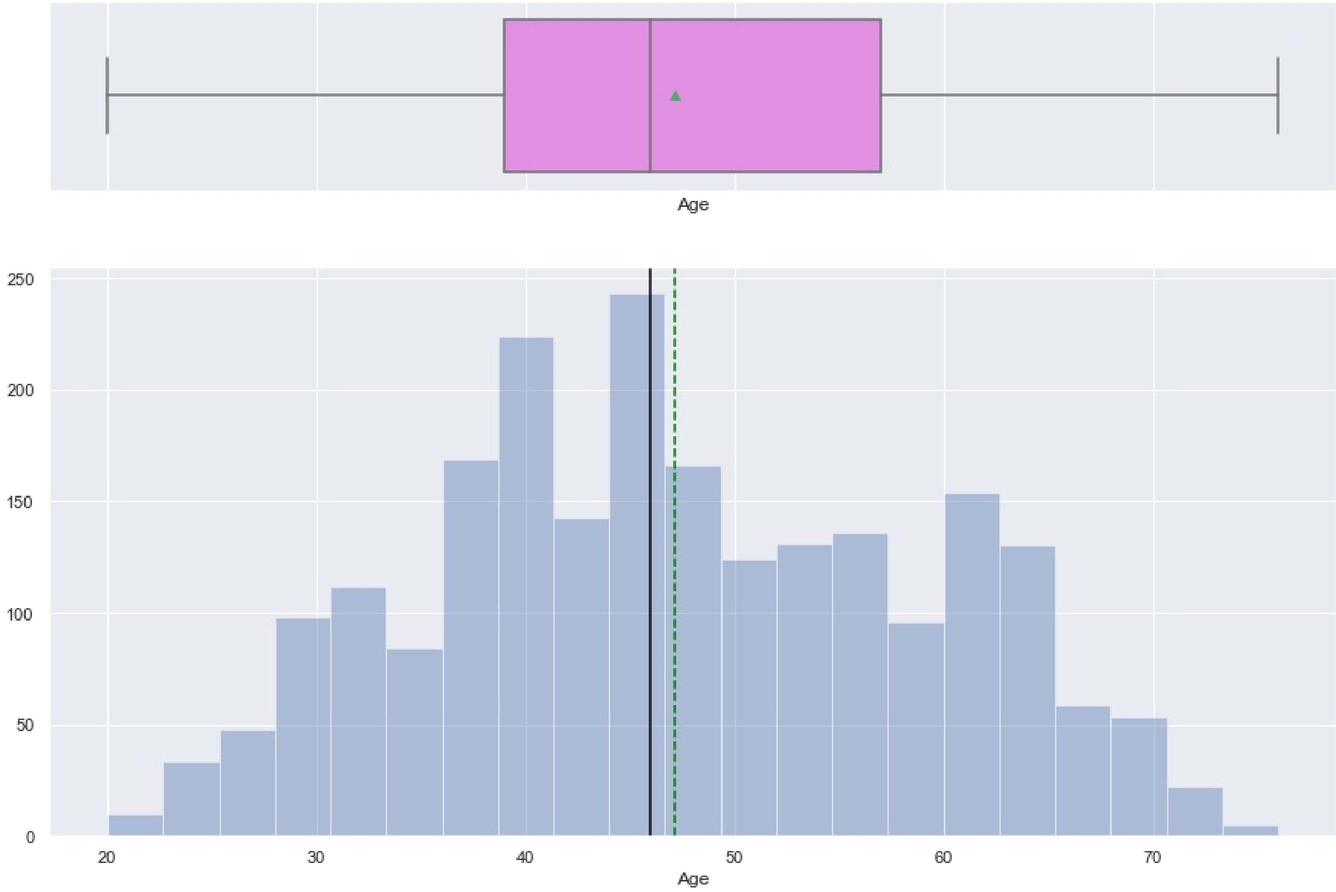
In [195…

```
# While doing uni-variate analysis of numerical variables we want to study their central tendency
# and dispersion.
```

```python
# Let us write a function that will help us create a boxplot and histogram for any input numerical
# variable.
# This function takes the numerical column as the input and returns the boxplots
# and histograms for the variable.
# Let us see if this helps us write faster and cleaner code.
def histogram_boxplot(feature, figsize=(15, 10), bins=None):
    """Boxplot and histogram combined
    feature: 1-d feature array
    figsize: size of fig (default (9,8))
    bins: number of bins (default None / auto)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.distplot(
        feature, kde=F, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.distplot(
        feature, kde=False, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        np.mean(feature), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        np.median(feature), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [196…

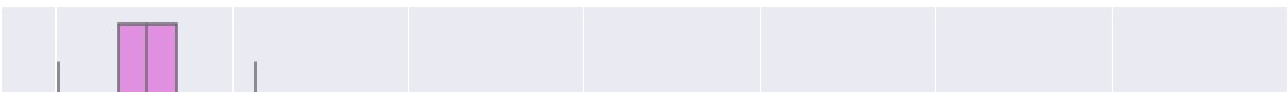
```python
# Observations on Customer_age
histogram_boxplot(data["Age"])
```

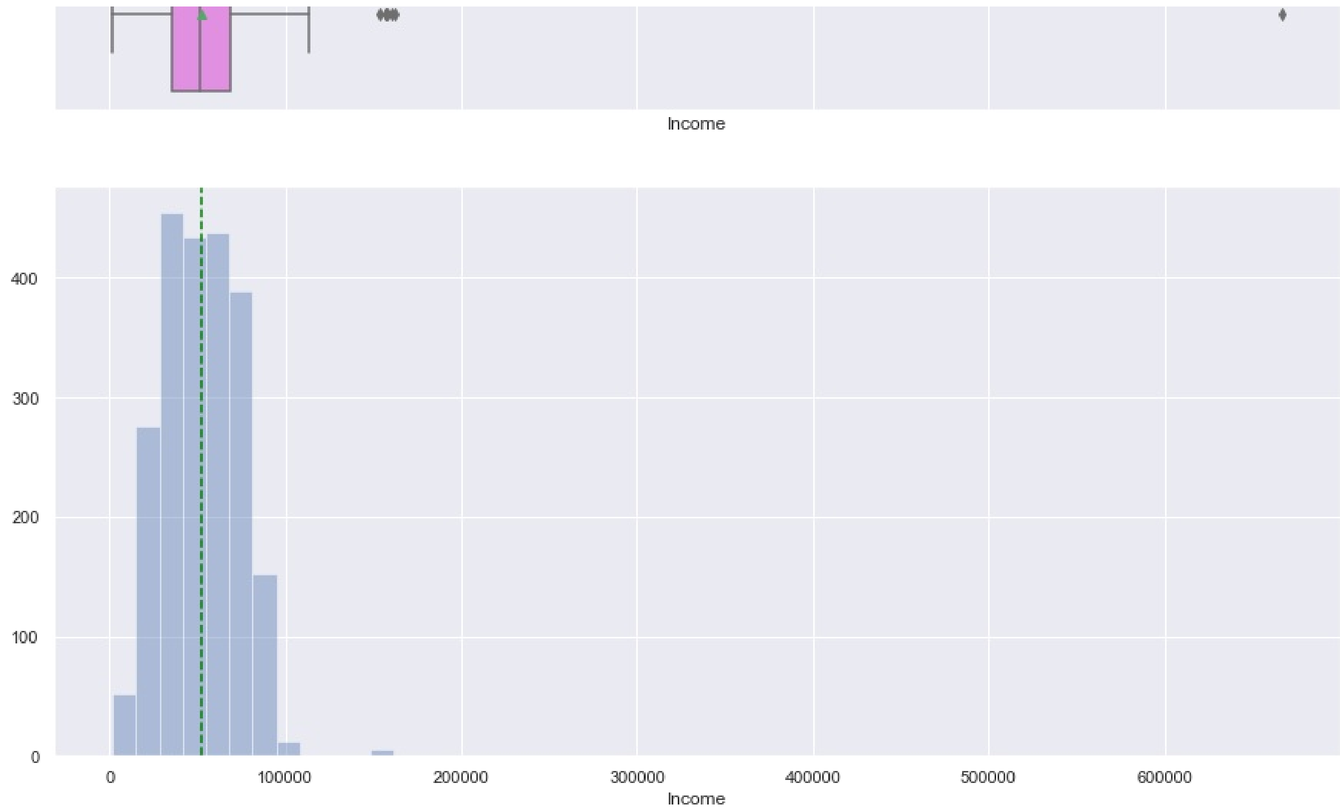


- As per the boxplot, there are no outliers in the 'Age' variable
- Age has a fairly normal distribution with approx equal mean and median

In [197…

```python
# observations on Income
histogram_boxplot(data["Income"])
```

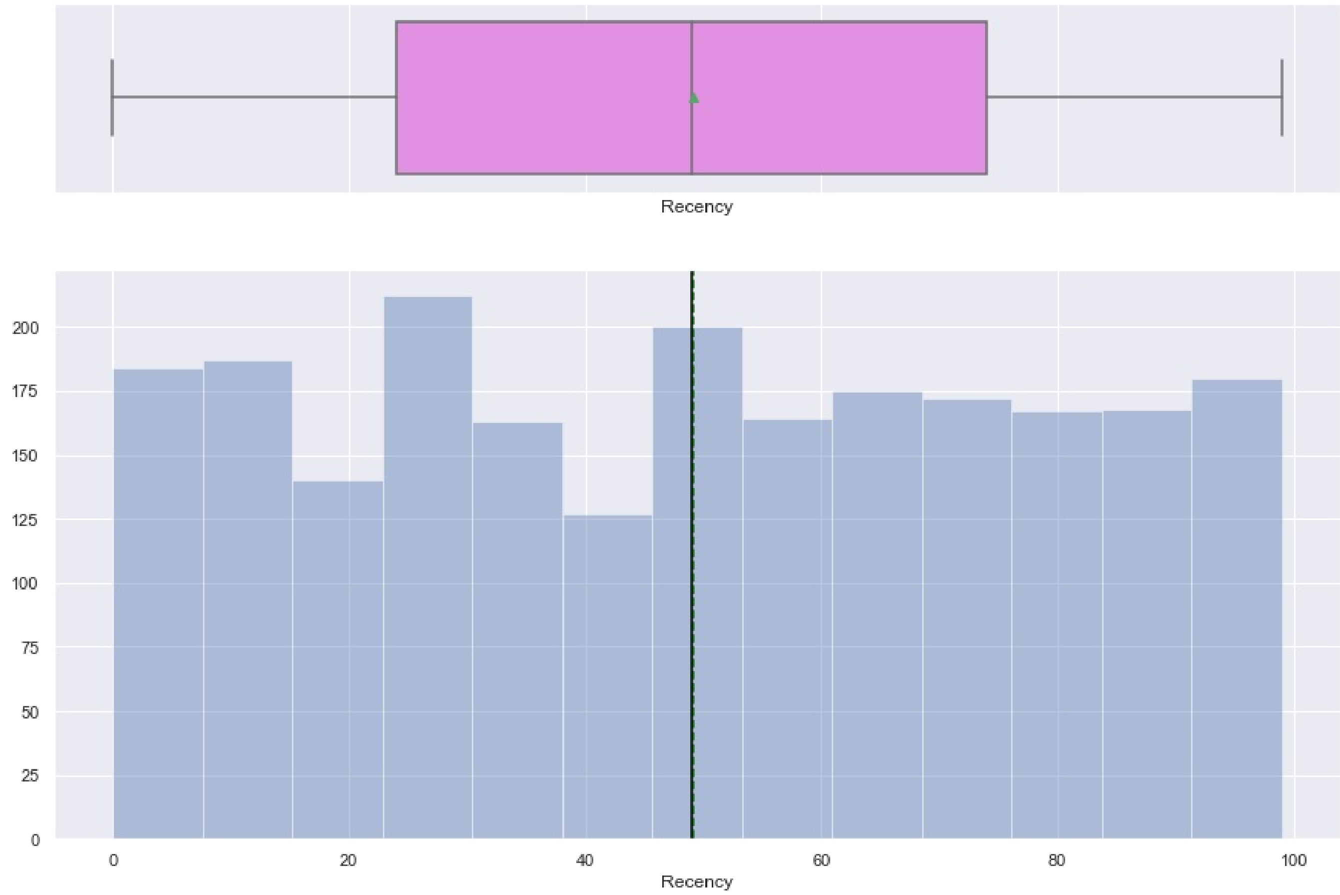




- We can see there are some outliers in the income variable.
- Some variation is always expected in real-world scenarios for the income variable but we can remove the data point on the extreme right end of the boxplot as it can be a data entry error.

In [198…
```
# Dropping observaion with income greater than 20000. There is just 1 such observation
data.drop(index=data[data.Income > 200000].index, inplace=True)
```

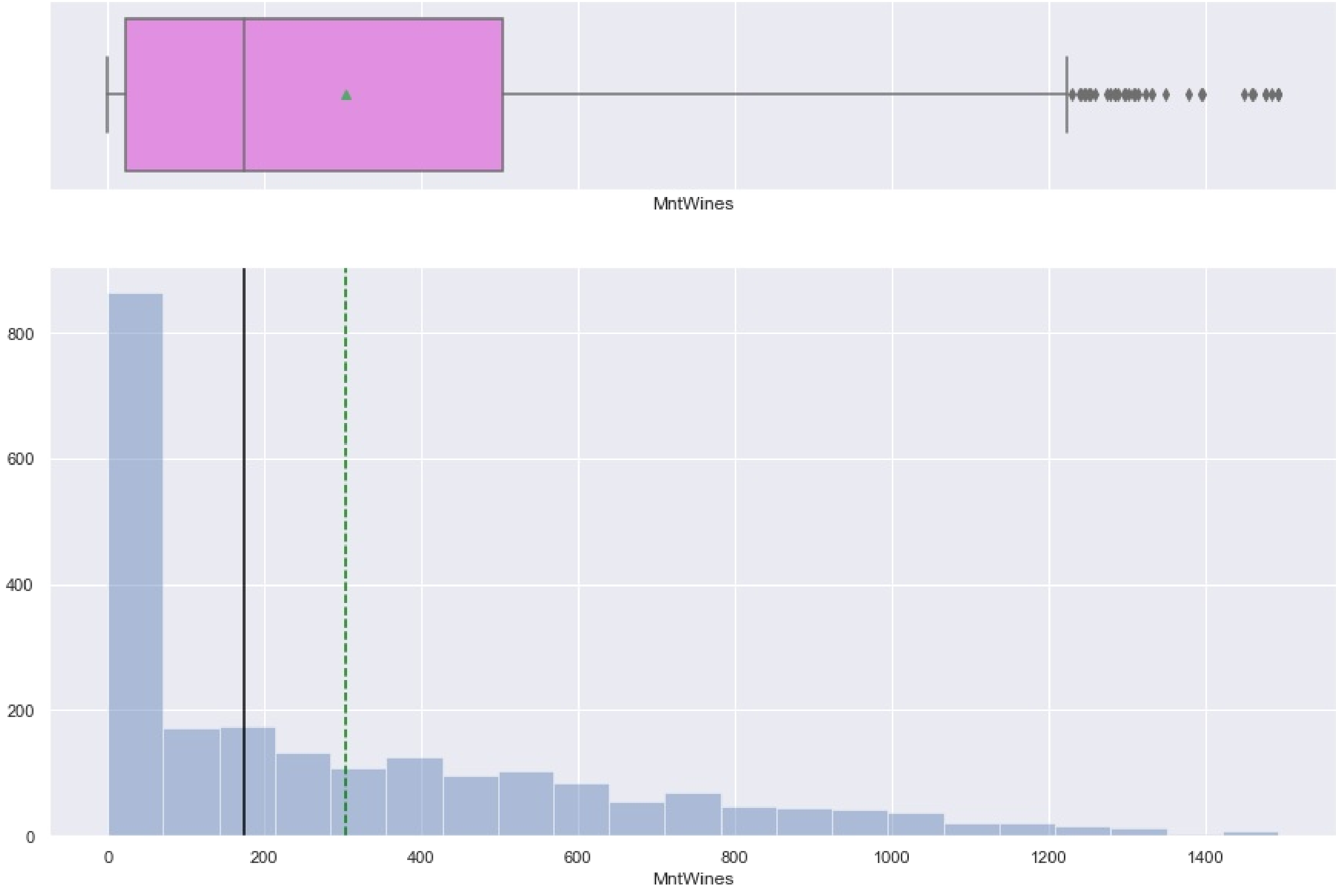
In [199…
```
# observations on Recency
histogram_boxplot(data["Recency"])
```



- There are no outliers in the 'Recency' variable
- The distribution is fairly symmetric and uniformly distributed.

In [200…

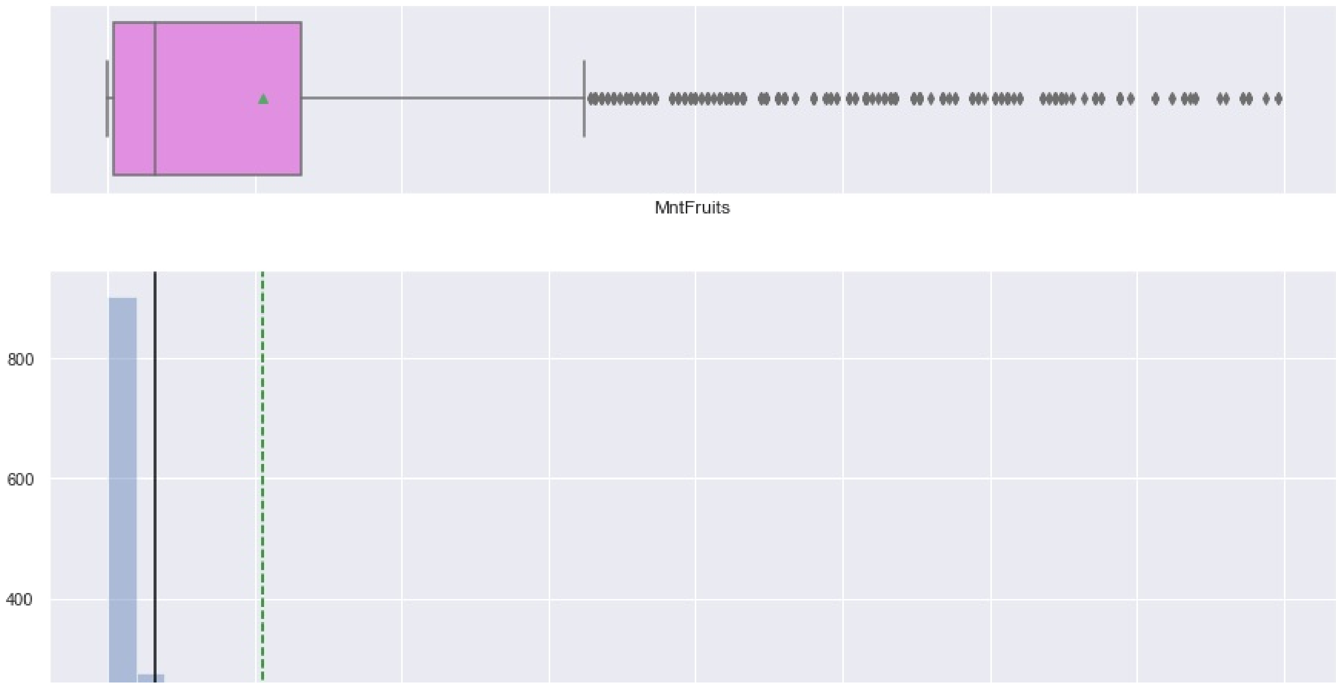
```
# observations on MntWines
histogram_boxplot(data["MntWines"])
```

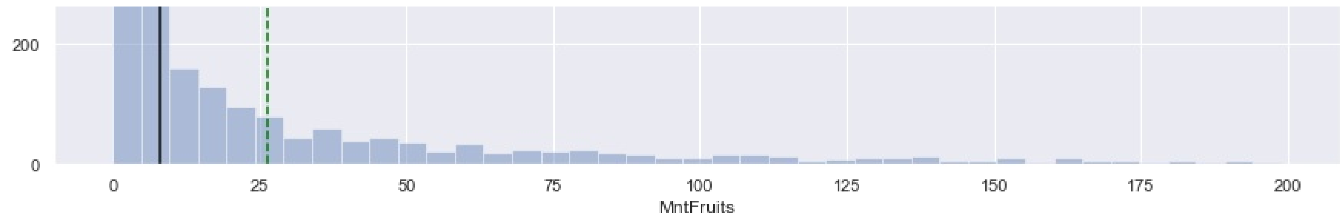


- The distribution for the amount spent on wines is highly skewed to the right
- As the median of the distribution is less than 200, more than 50% of customers have spent less than 200 on wines.
- There are some outliers on the right end of the boxplot but we will not treat them as some variation is always expected in real-world scenarios for variables like amount spent.

In [201…

```
# observations on MntFruits
histogram_boxplot(data["MntFruits"])
```

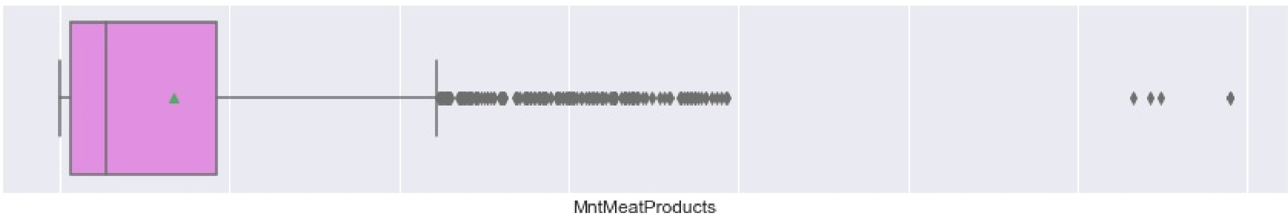


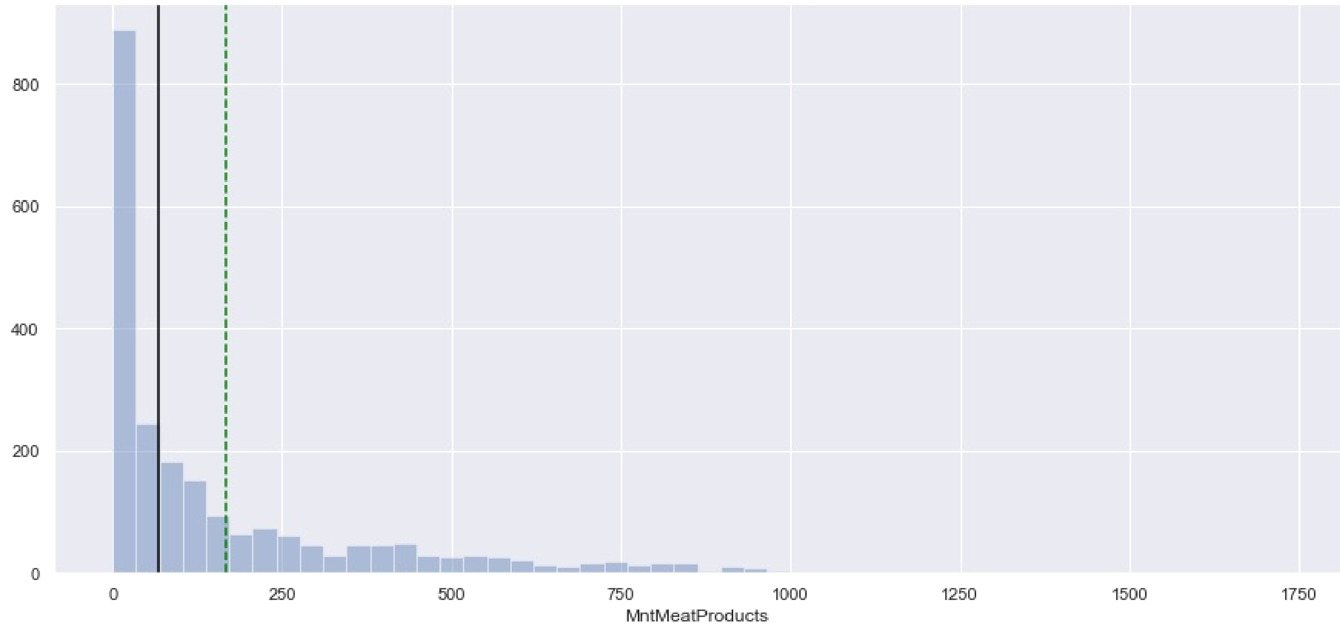


- The distribution for the amount spent on fruits is highly skewed to the right.
- As the median of the distribution is less than 20, more than 50% of customers have spent less than 20 on fruits.
- There are some outliers on the right end of the boxplot but we will not treat them as some variation is always expected in real-world scenarios for variables like amount spent.

In [202…

```
# observations on MntMeatProducts
histogram_boxplot(data["MntMeatProducts"])
```





- The distribution for the amount spent on meat products is highly skewed to the right.
- We can see that there are some extreme observations in the variable that can be considered as outliers as they very far from the rest of the values.
- We can cap the value of the variable to the next highest value.

In [106…

```
# Checking 5 largest values of amount spend on meat products
data.MntMeatProducts.nlargest(10)
```

Out[106…

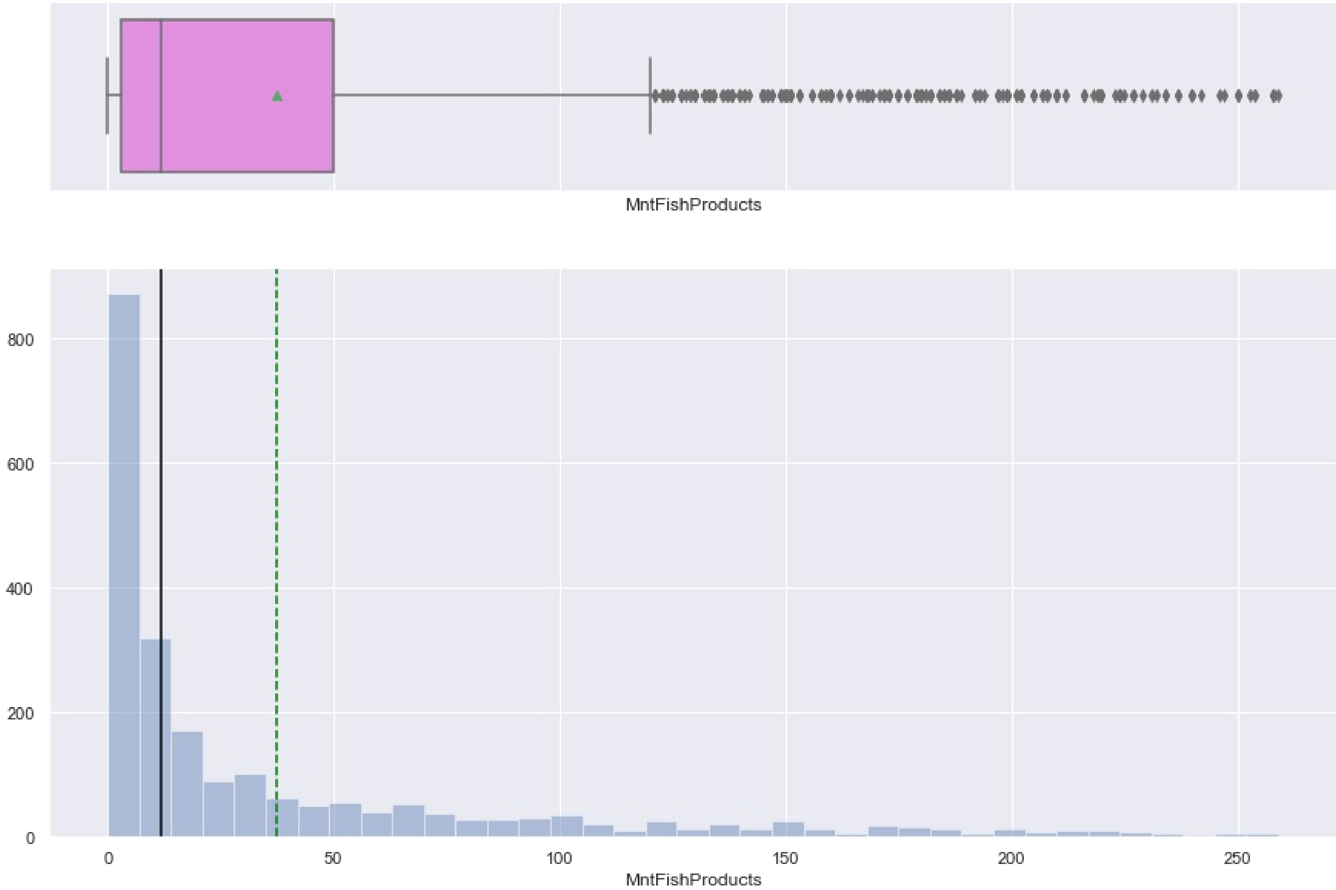
```
325     1725
961     1725
497     1622
1213    1607
2204    1582
1921     984
53       981
994      974
2021     968
1338     961
Name: MntMeatProducts, dtype: int64
```

In [203…

```
# Capping values for amount spent on meat products at next highest value i.e. 984
data["MntMeatProducts"].clip(upper=984, inplace=True)
```

In [204…

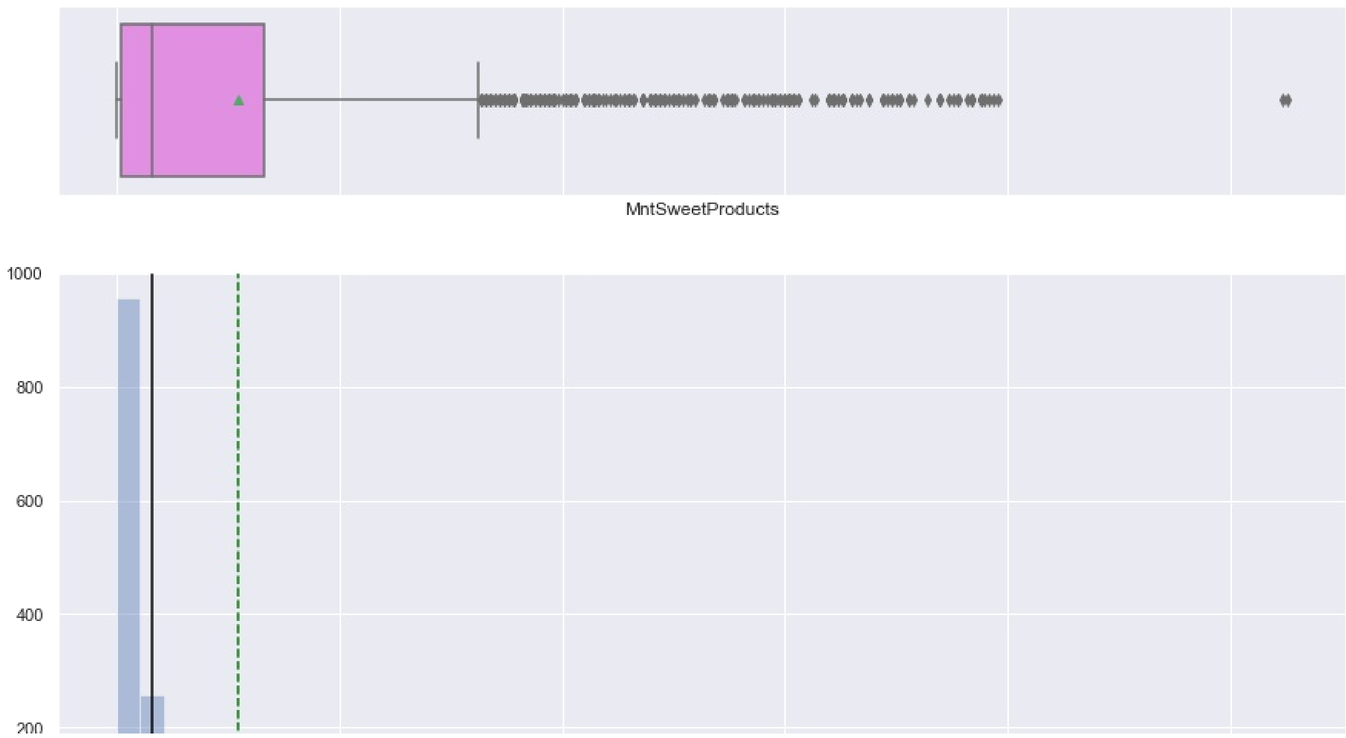
```
# observations on MntFishProducts
histogram_boxplot(data["MntFishProducts"])
```



- The distribution for the amount spent on fish products is right-skewed
- There are some outliers on the right end in the boxplot but we will not treat them as this represents a real market trend that some customers spend more on fish products than others.

In [205…

```
# observations on MntSweetProducts
histogram_boxplot(data["MntSweetProducts"])
```



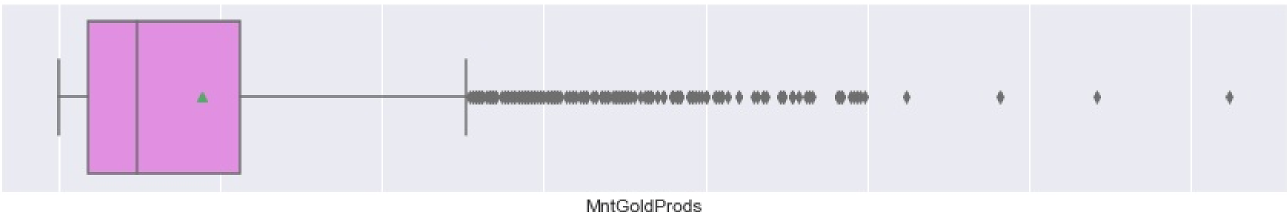
- The distribution for the amount spent on sweet products is right-skewed
- There is one observation to the right extreme which can be considered as an outlier.
- We will not remove all such data points as they represent real market trends but we can cap some of the extreme values.

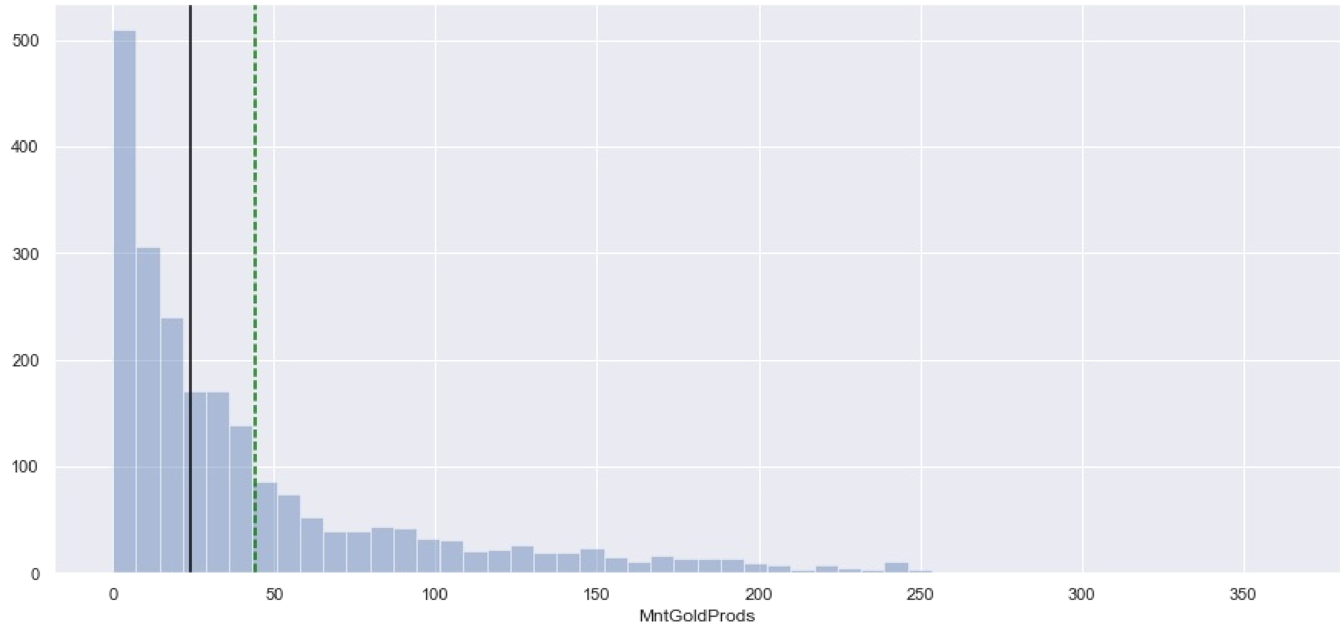
In [206…

```
# Capping values for amount spent on sweet products at 198
data["MntSweetProducts"].clip(upper=198, inplace=True)
```

In [111…

```
# observations on MntGoldProds
histogram_boxplot(data["MntGoldProds"])
```





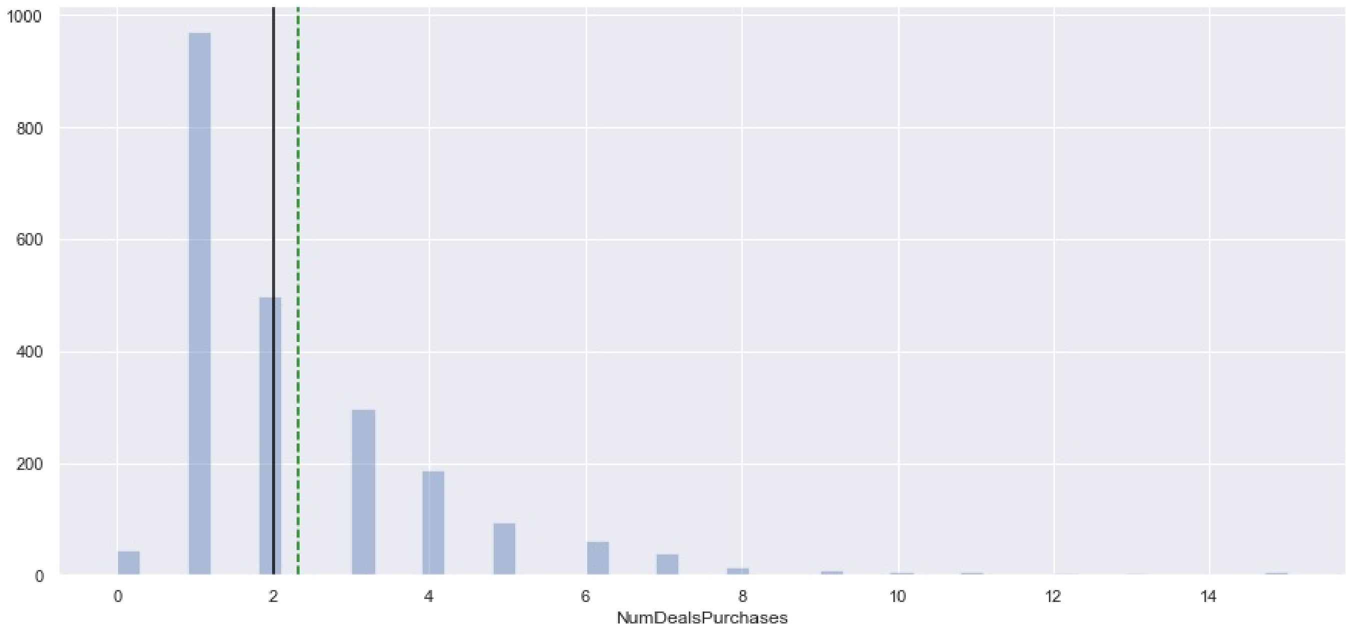
- The distribution for the amount spent on gold products is right-skewed
- There are some outliers in the amount spent on gold products. We will not remove all such data points as they represent real market trends but we can cap some of the extreme values.

In [207…

```
# Capping values for amount spent on gold products at 250
data["MntGoldProds"].clip(upper=250, inplace=True)
```

In [113…

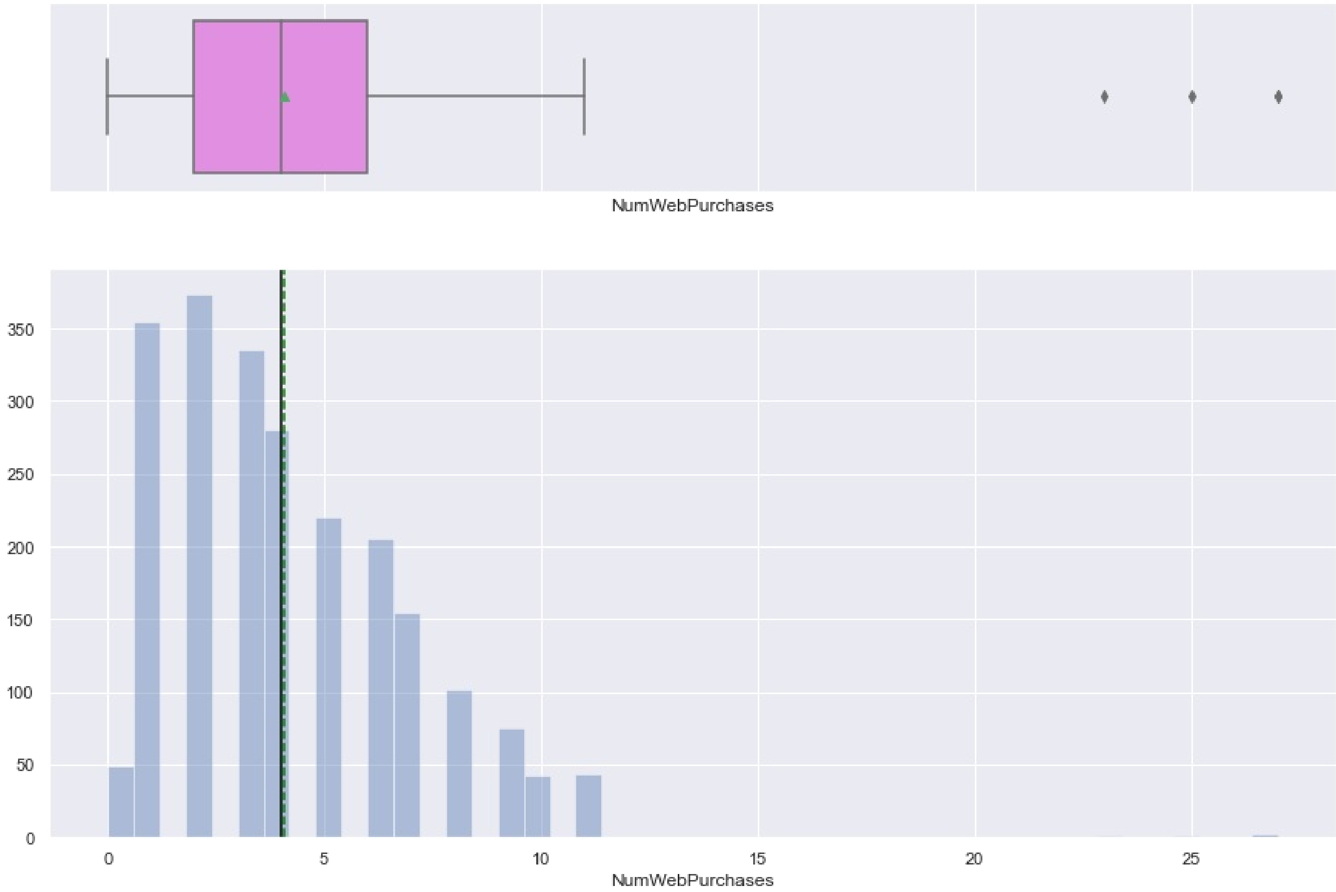
```
# observations on NumDealsPurchases
histogram_boxplot(data["NumDealsPurchases"])
```



- Majority of the customers have 2 or less than 2 deal purchases.
- We can see that there some extreme observations in the variable. This represents the real market trend.

In [114…

```
# observations on NumWebPurchases
histogram_boxplot(data["NumWebPurchases"])
```



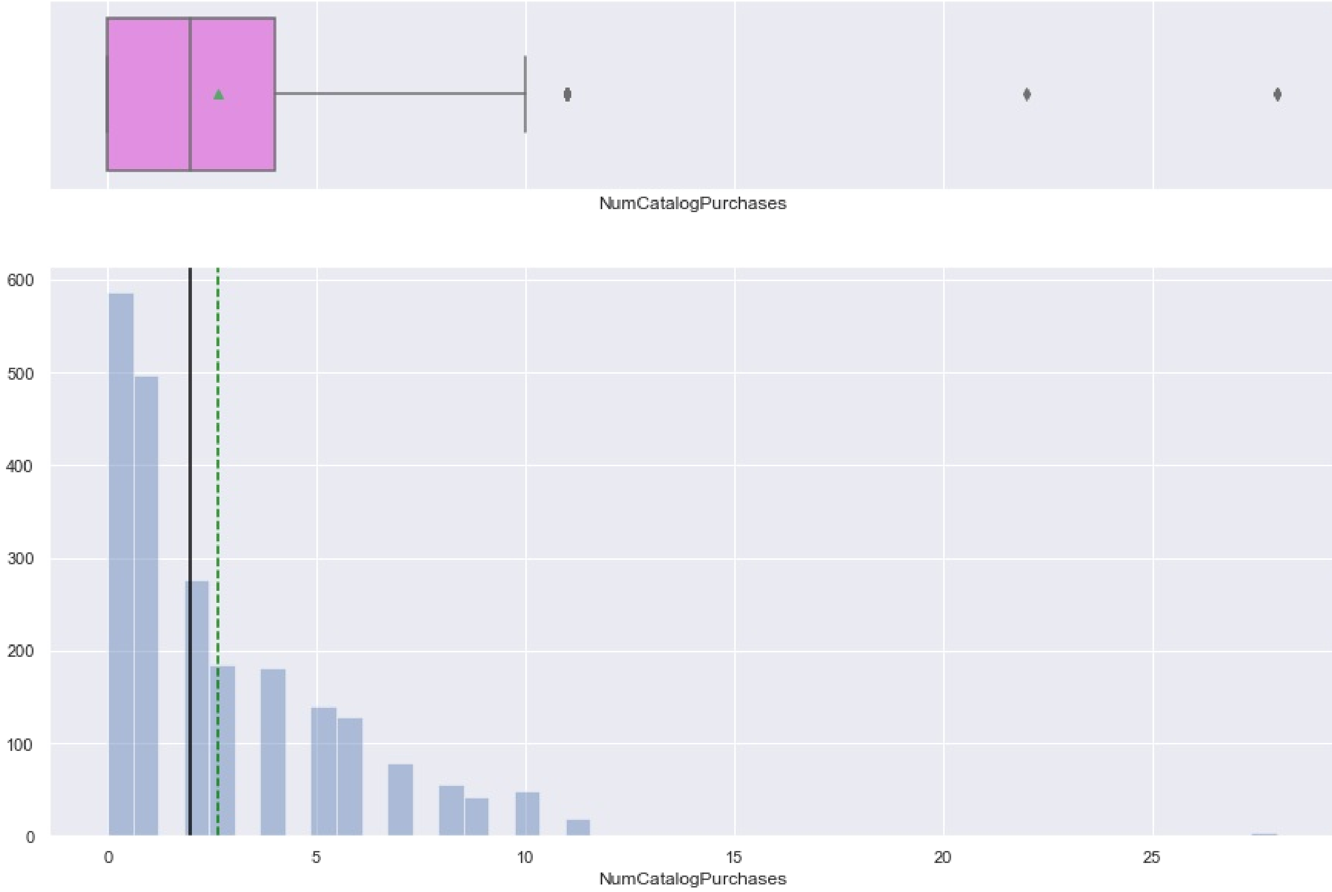
- The median of the distribution is 4 i.e. 50% of customers have 4 or less than 4 web purchases.
- We can see that there are some extreme observations in the variable. We can cap these values to the next highest number of purchases.

In [208…

```
# Capping values for number of web purchases at 11
data["NumWebPurchases"].clip(upper=11, inplace=True)
```

In [116…

```python
# observations on NumCatalogPurchases
histogram_boxplot(data["NumCatalogPurchases"])
```



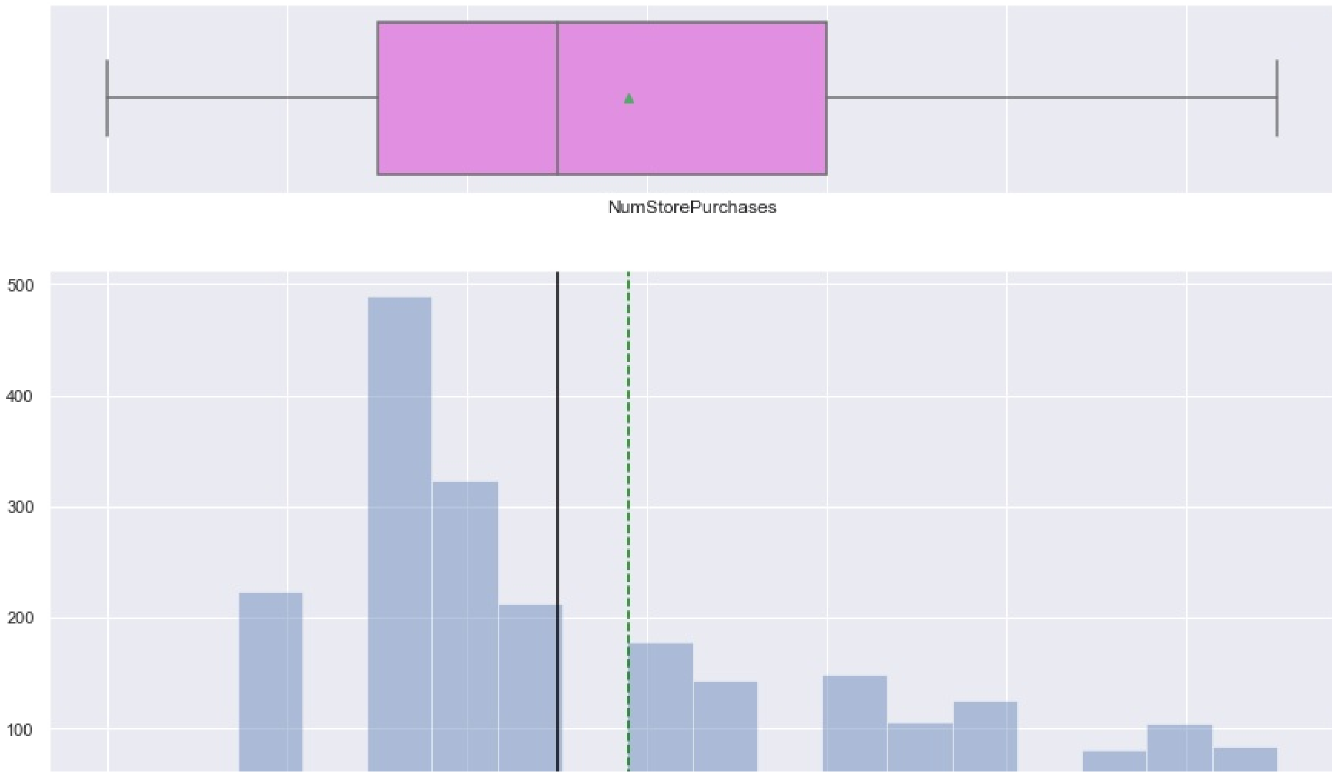
- The most number of observations are for 0 catalog purchases.
- The median of the distribution is 2 i.e. 50% of customers have 2 or less than 2 catalog purchases.
- We can see that there is two extreme observation in the variable. We can cap these values to the next highest number of purchases.

In [209…

```python
# Capping values for number of catalog purchases at 11
data["NumCatalogPurchases"].clip(upper=11, inplace=True)
```

In [118…

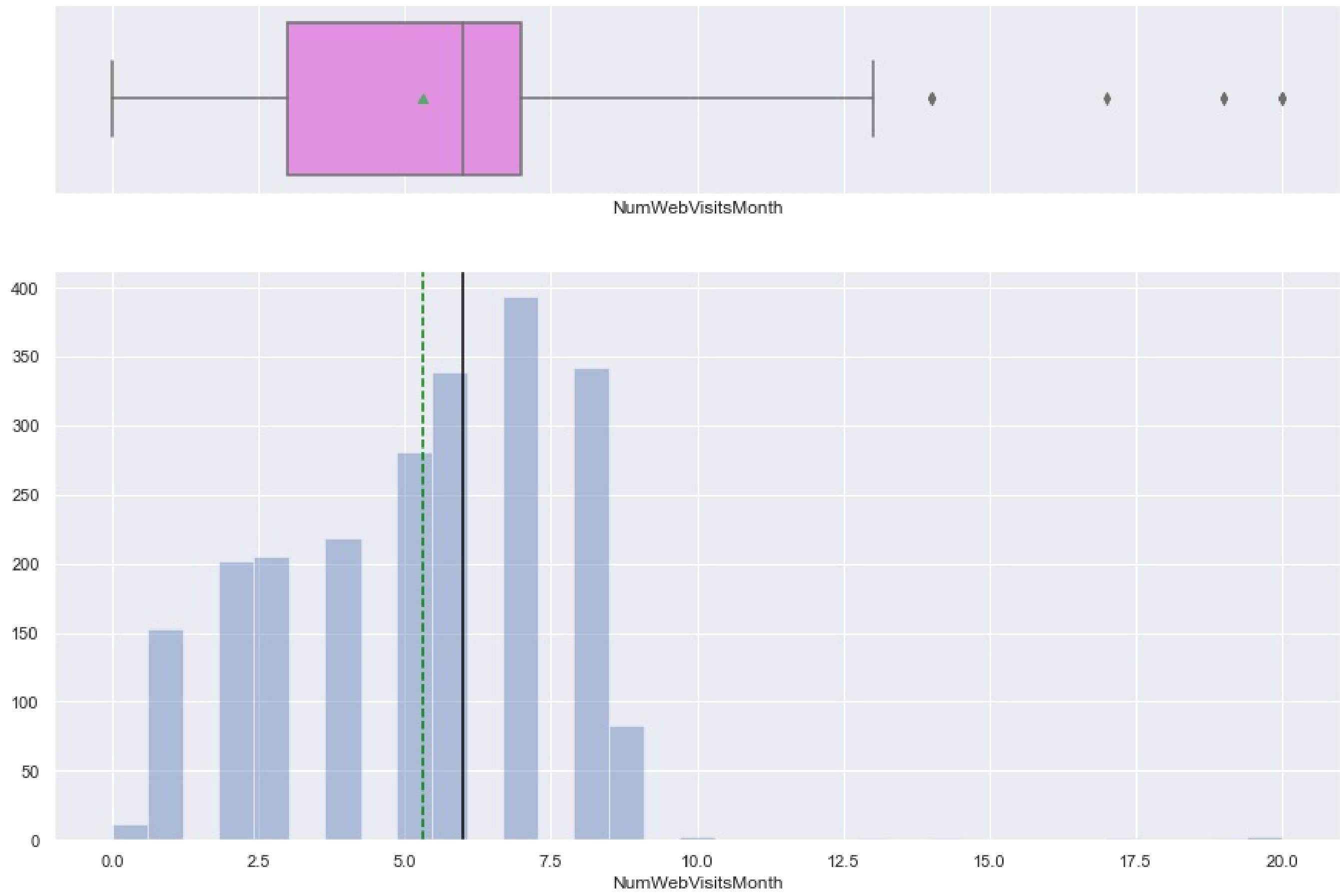
```python
# observations on NumStorePurchases
histogram_boxplot(data["NumStorePurchases"])
```



- There are very few observations with less than 2 purchases from the store
- Most of the customers have 4 or 5 purchases from the store
- There are no outliers in this variable

In [119…

```
# observations on NumWebVisitsMonth
histogram_boxplot(data["NumWebVisitsMonth"])
```



- The distribution for the number of visits in a month is skewed and has some outliers at the right end.
- We will not treat this as this represents a general market trend

In [120…

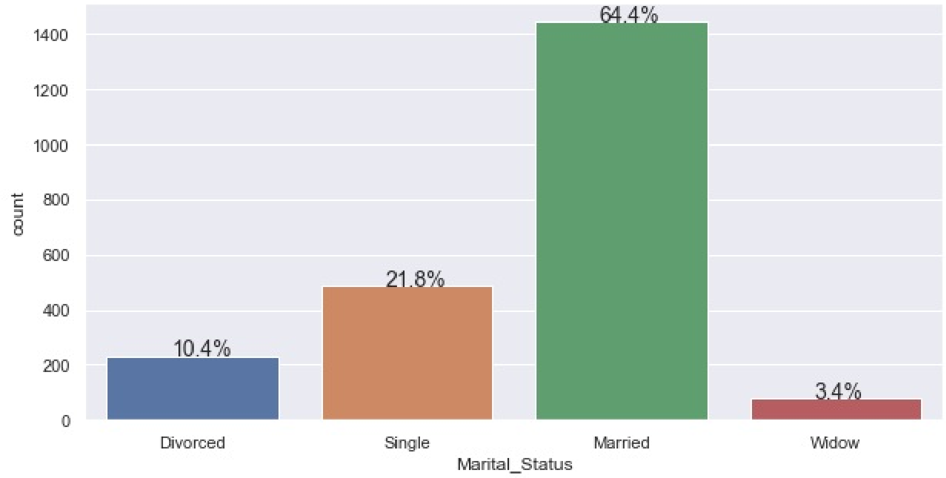
```
def perc_on_bar(feature):
    """
    plot
    feature: categorical feature
    the function won't work if a column is passed in the hue parameter
    """
    # Creating a countplot for the feature
    sns.set(rc={"figure.figsize": (10, 5)})
    ax = sns.countplot(x=feature, data=data)

    total = len(feature)  # length of the column
    for p in ax.patches:
        percentage = "{:.1f}%".format(
            100 * p.get_height() / total
        )  # percentage of each class of the category
        x = p.get_x() + p.get_width() / 2 - 0.1  # width of the plot
        y = p.get_y() + p.get_height()  # hieght of the plot
        ax.annotate(percentage, (x, y), size=14)  # annotate the percantage

    plt.show()  # show the plot
```

In [121…

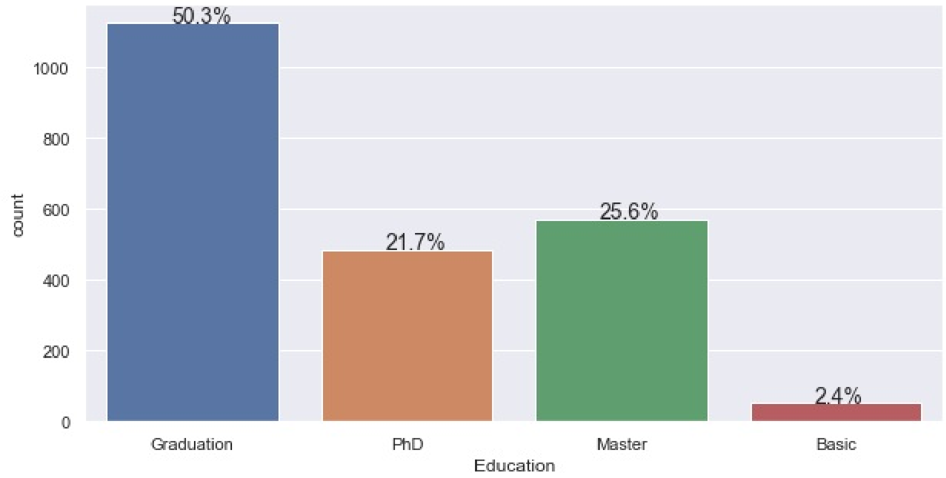
```
# observations on Marital_Status
perc_on_bar(data["Marital_Status"])
```



- Majority of the customers are married comprising approx 64% of total customers.

In [122…

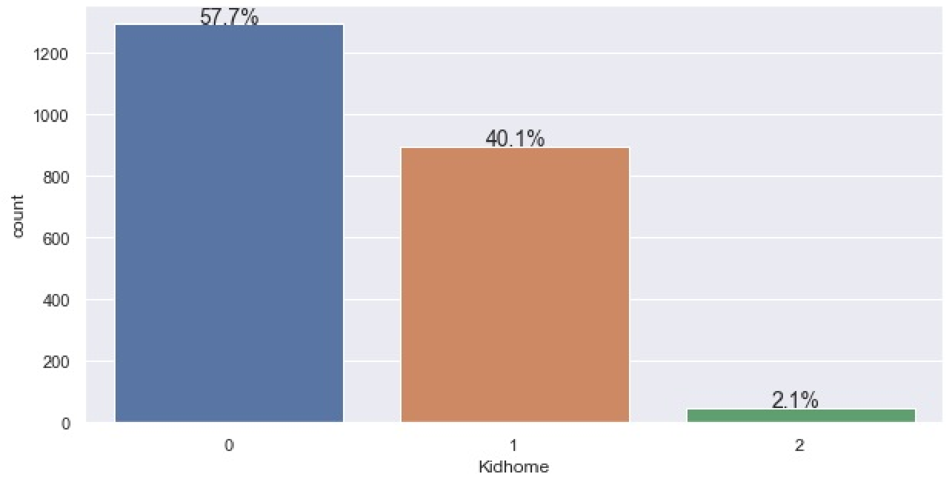
```
# observations on Education
perc_on_bar(data["Education"])
```



- Education of approx 50% of customers is at graduation level.
- Very few observations i.e. ~2% for customers with basic level education

In [123…

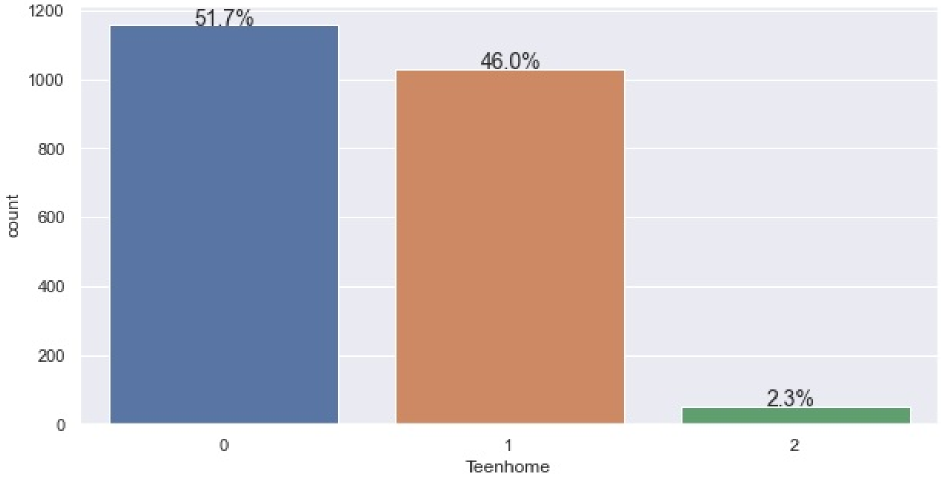
```
# observations on Kidhome
perc_on_bar(data["Kidhome"])
```



- ~40% of customers have 1 kid and ~58% of customers have no kids at home
- There are very few customers, approx 2%, with a number of kids greater than 1

In [124…

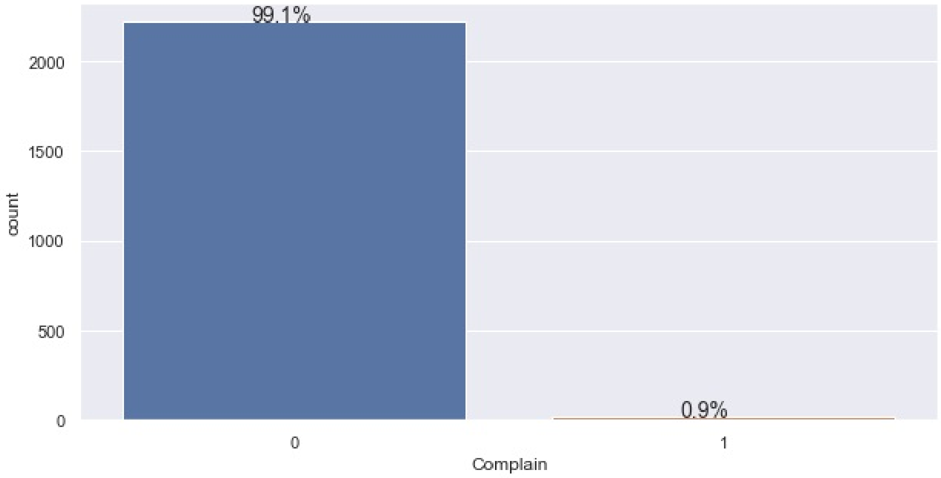
```
# observations on Teenhome
perc_on_bar(data["Teenhome"])
```



- Majority of the customers i.e. ~52% customers have no teen at home
- There are very few customers, only ~2%, with a number of teens greater than 1

In [125…

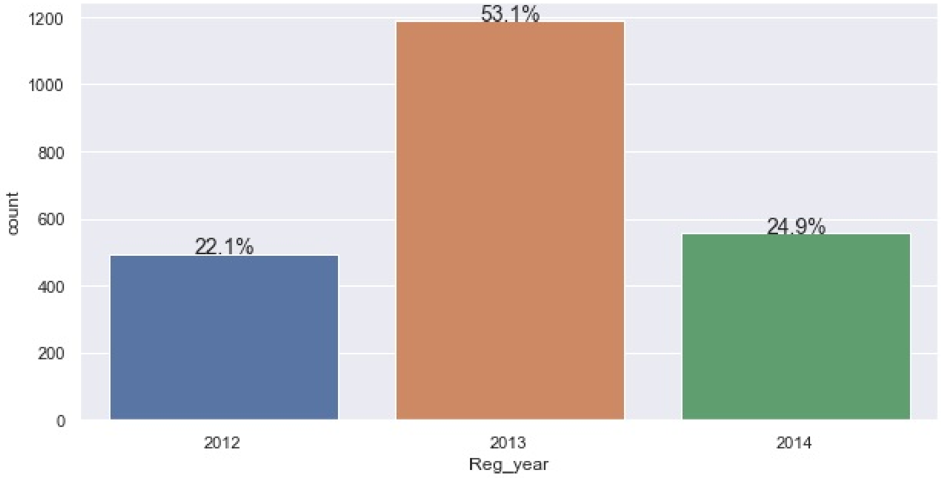
```
# observations on Complain
perc_on_bar(data["Complain"])
```



- Approx 99% of customers had no complaint in the last 2 years. This might be because the company provides good services or might be due to the lack of feedback options for customers.

In [126…

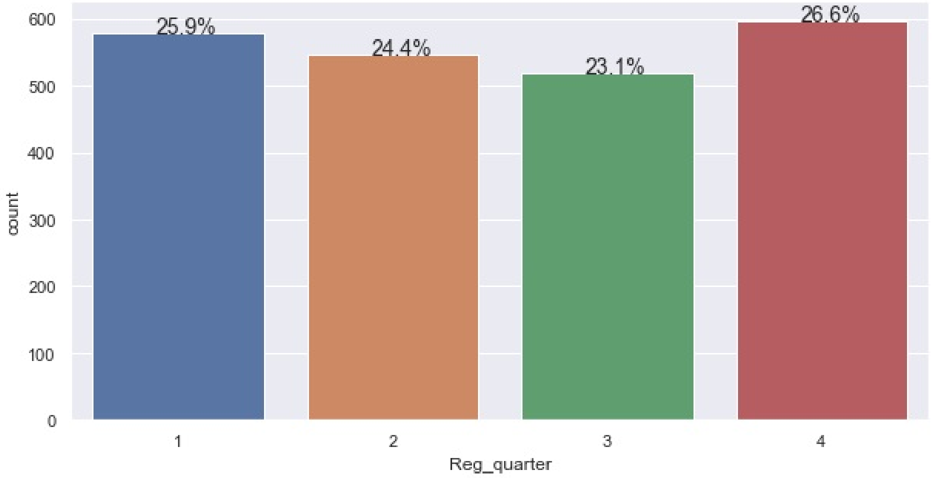
```
# observations on Registration year
perc_on_bar(data["Reg_year"])
```



- The number of customers registered is highest in the year 2013.

In [127…

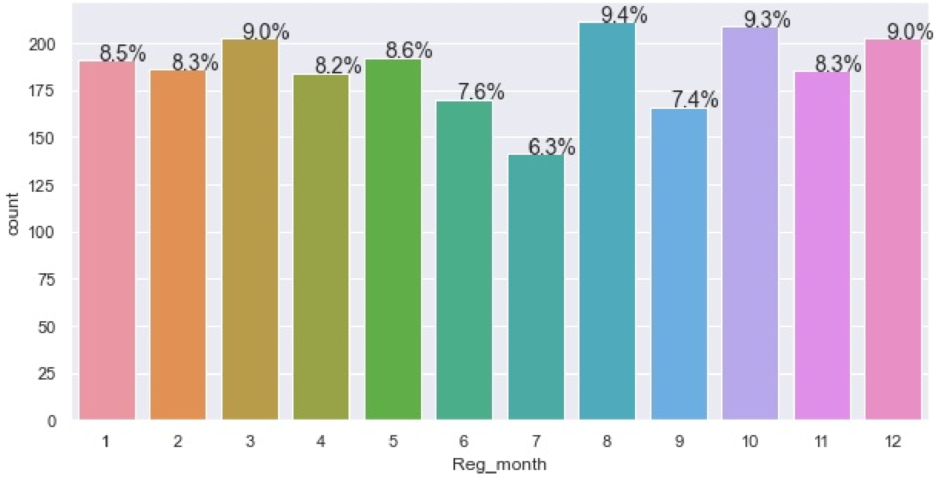
```
# observations on Registration quarter
perc_on_bar(data["Reg_quarter"])
```



- There is no significant difference in the number of registrations for each quarter.
- The number of registrations is slightly higher for the 1st and the 4th quarter. This can be due to the festival season in these months.
- Let's explore this further by plotting the count of registration per month.

In [128…

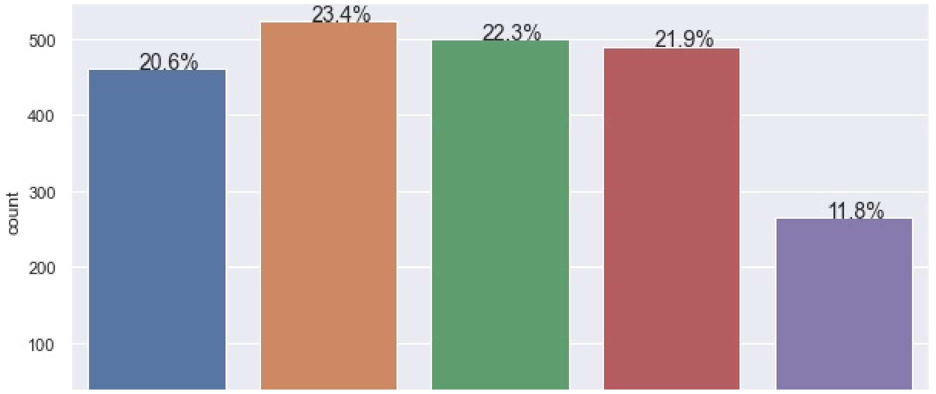
```
# observations on Registration month
perc_on_bar(data["Reg_month"])
```



- This shows that the highest number of registration is in the months of winters i.e. March, August, October & December
- There is approx 3% reduction in the number of registrations from June to July.

In [129…

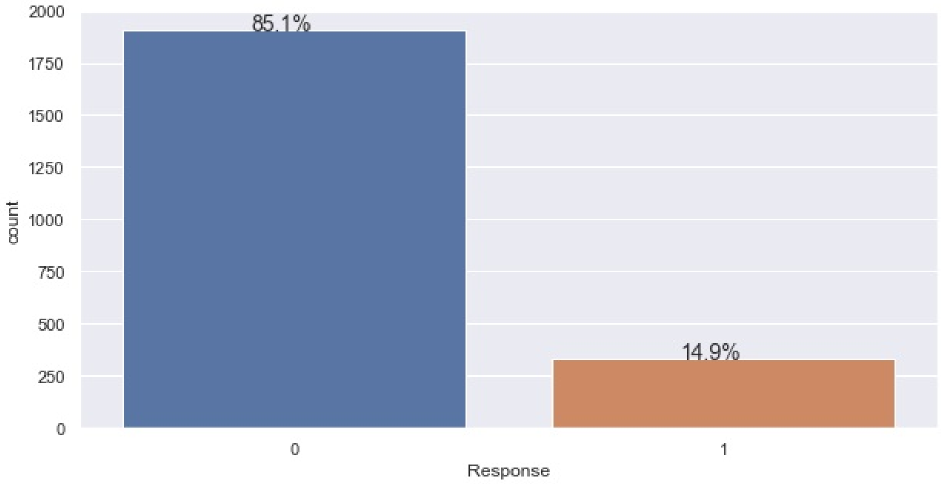
```
# observations on Registration week
perc_on_bar(data["Reg_week"])
```



- This shows that the number of registrations declines at the end of the month i.e. in the last two weeks.
- This can be because most people get salaries on the last day or first day of the month.

In [130…

```
# observations on Response
perc_on_bar(data["Response"])
```



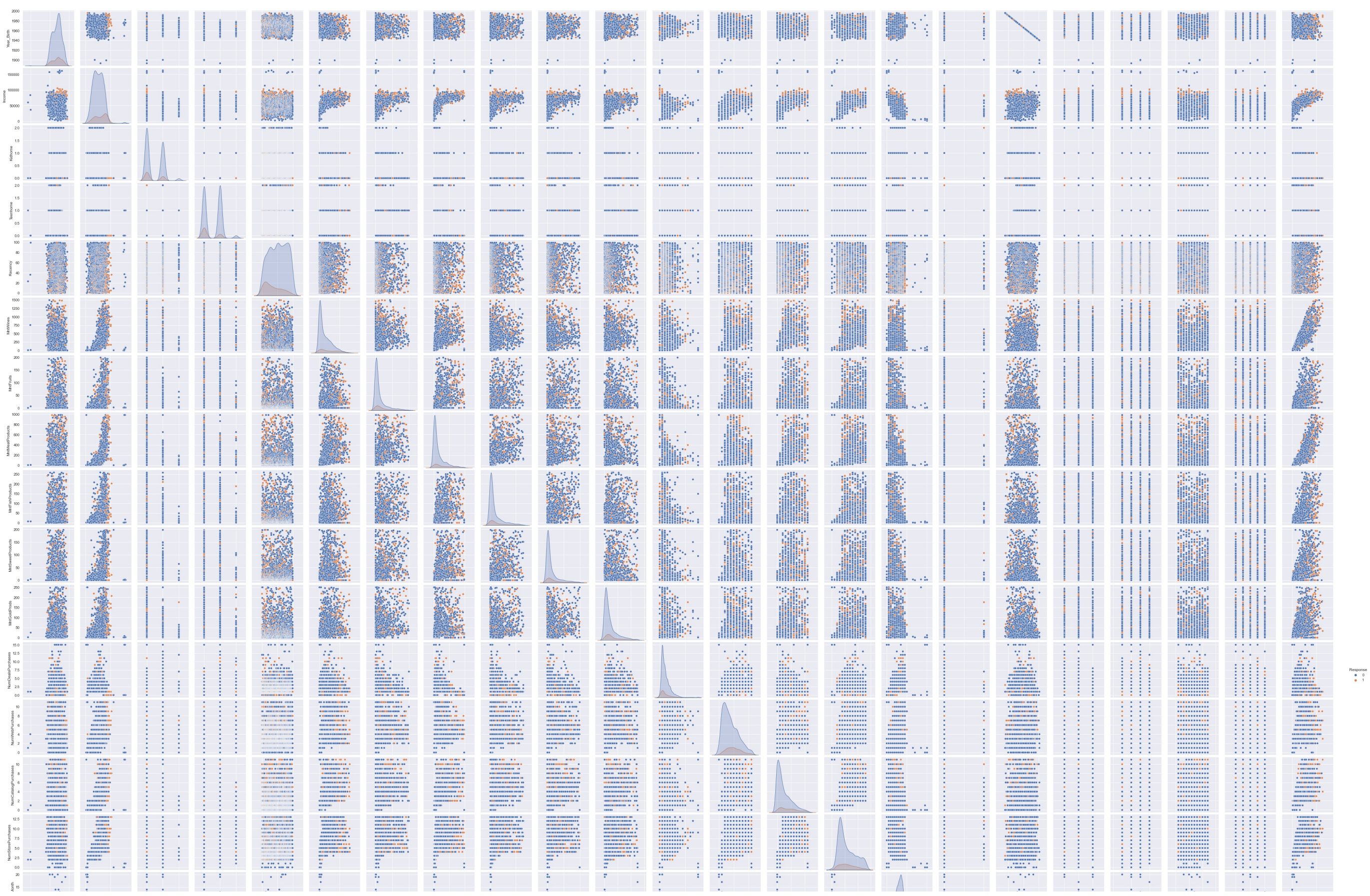
- Approx 85% customer's response was NO in the last campaign.
- This shows that the distribution of classes in the target variable is imbalanced. We have only ~15% observations where the response is YES.

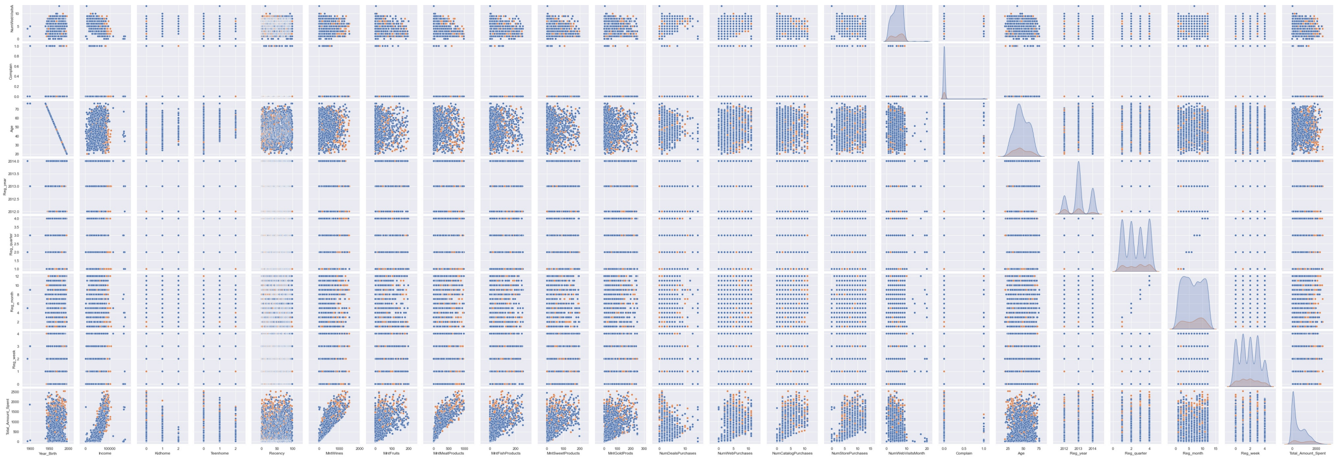
Bivariate Analysis

In [131…

```
sns.pairplot(data, hue="Response")
```

Out[131… <seaborn.axisgrid.PairGrid at 0x167f26990>

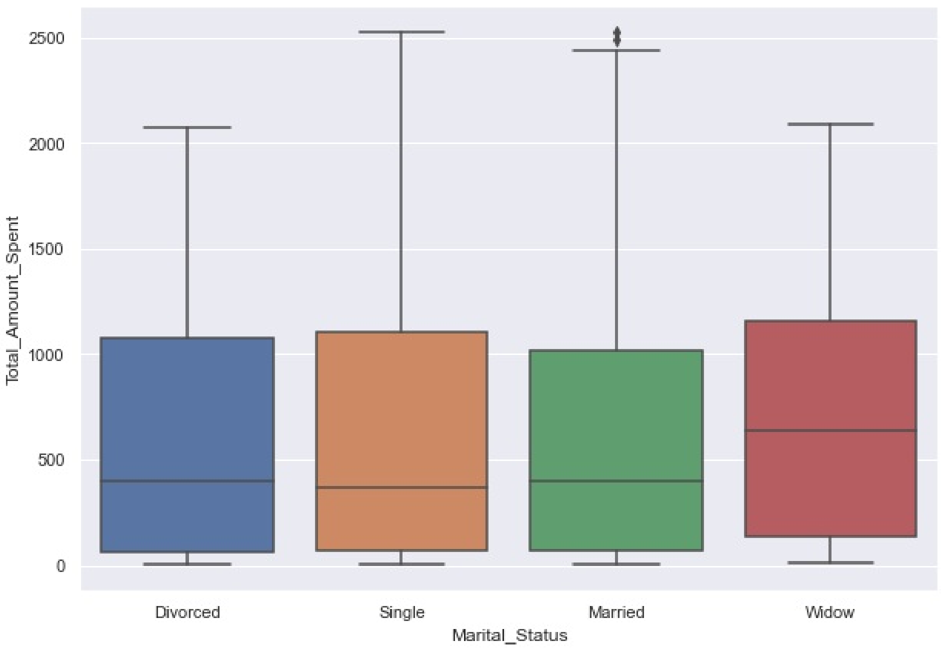




- There are overlaps i.e. no clear distinction in the distribution of variables for people who have taken the product and did not take the product.
- Let's explore this further with the help of other plots.

In [132…
```
sns.set(rc={"figure.figsize": (10, 7)})
sns.boxplot(y="Total_Amount_Spent", x="Marital_Status", data=data, orient="vertical")
```

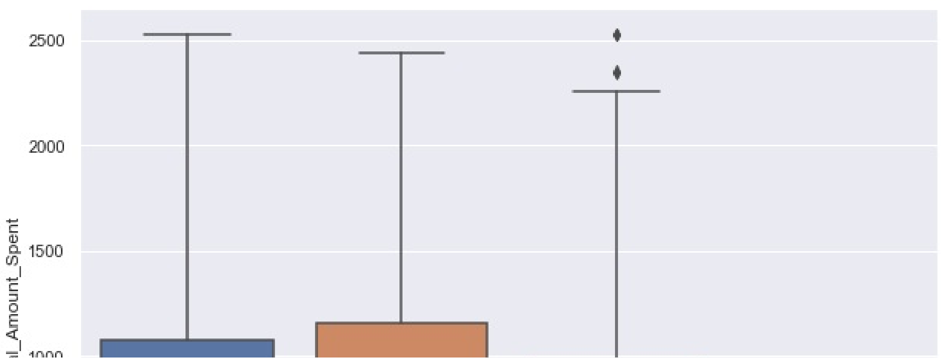
Out[132… <AxesSubplot:xlabel='Marital_Status', ylabel='Total_Amount_Spent'>

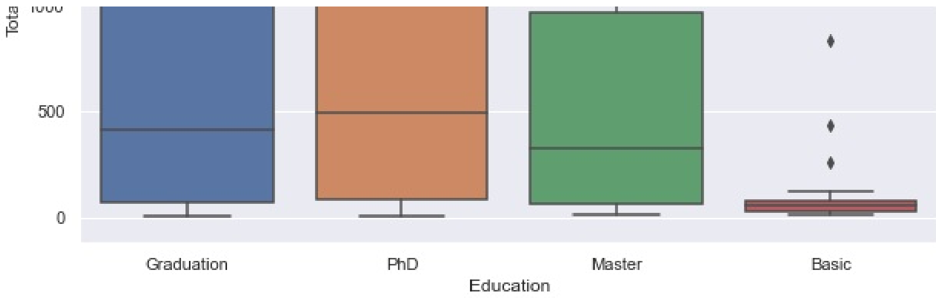


- We can see that the total amount spent is higher for widowed customers.
- No significant difference in the amount spent by single, married or divorced customers.

In [133…
```
sns.boxplot(y="Total_Amount_Spent", x="Education", data=data, orient="vertical")
```

Out[133… <AxesSubplot:xlabel='Education', ylabel='Total_Amount_Spent'>



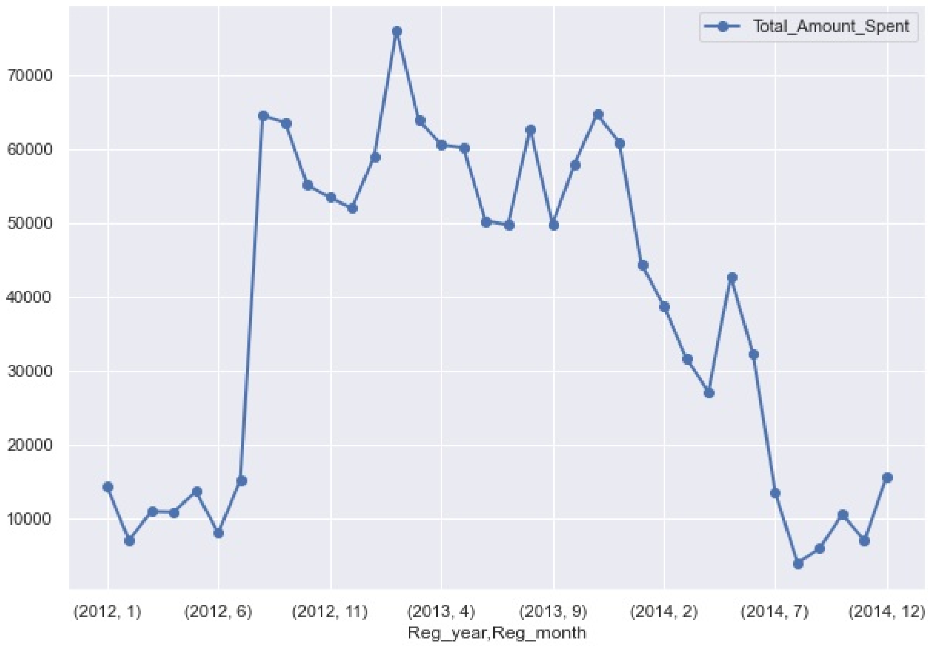


- As expected, the amount spent increases with the increase in education level.
- Customers with graduate-level education spend slightly more than customers with master-level education.

In [134…

```
pd.pivot_table(
    data=data,
    index=["Reg_year", "Reg_month"],
    values="Total_Amount_Spent",
    aggfunc=np.sum,
).plot(kind="line", marker="o", linewidth=2)
```

Out[134… <AxesSubplot:xlabel='Reg_year,Reg_month'>

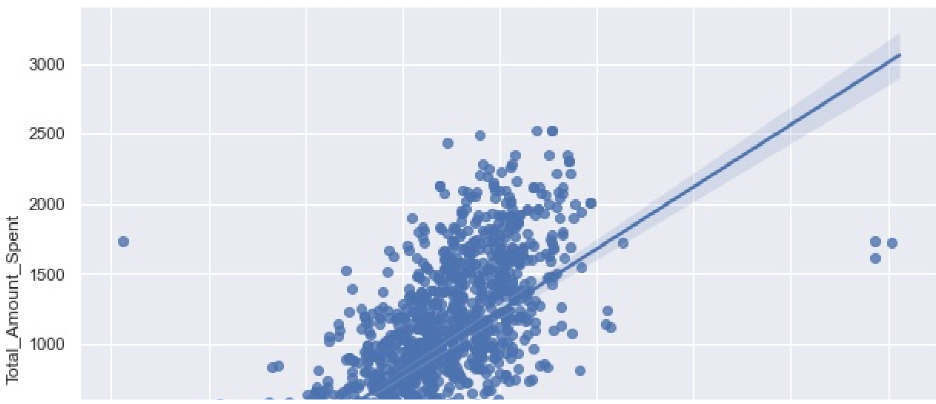


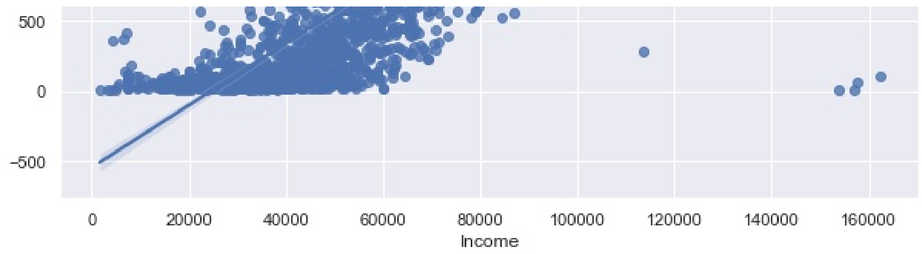
- The plot clearly shows that the total amount spent has declined over the years.
- The plot shows the highest increase in the amount spent from August to September 2012.

In [135…

```
sns.regplot(y=data.Total_Amount_Spent, x=data.Income)
```

Out[135… <AxesSubplot:xlabel='Income', ylabel='Total_Amount_Spent'>



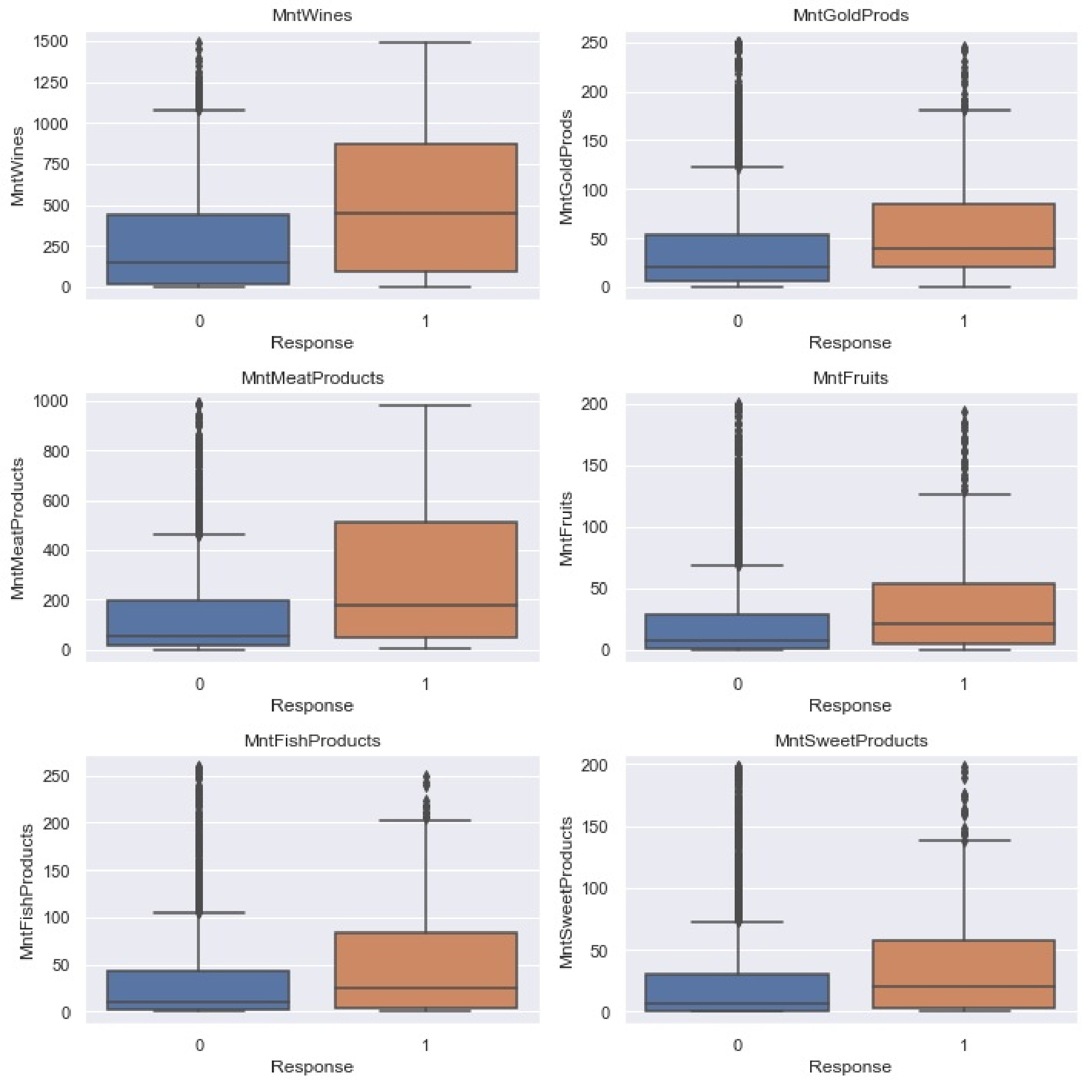


- We can see that income and the total amount spent have a positive correlation.
- The total amount spent is not much different for customers with income in the range of 20K to 60K but the difference is significant for customers in the range of 60K to 100K.

In [136…

```
cols = data[
    [
        "MntWines",
        "MntGoldProds",
        "MntMeatProducts",
        "MntFruits",
        "MntFishProducts",
        "MntSweetProducts",
    ]
].columns.tolist()
plt.figure(figsize=(10, 10))

for i, variable in enumerate(cols):
    plt.subplot(3, 2, i + 1)
    sns.boxplot(data["Response"], data[variable])
    plt.tight_layout()
    plt.title(variable)
plt.show()
```

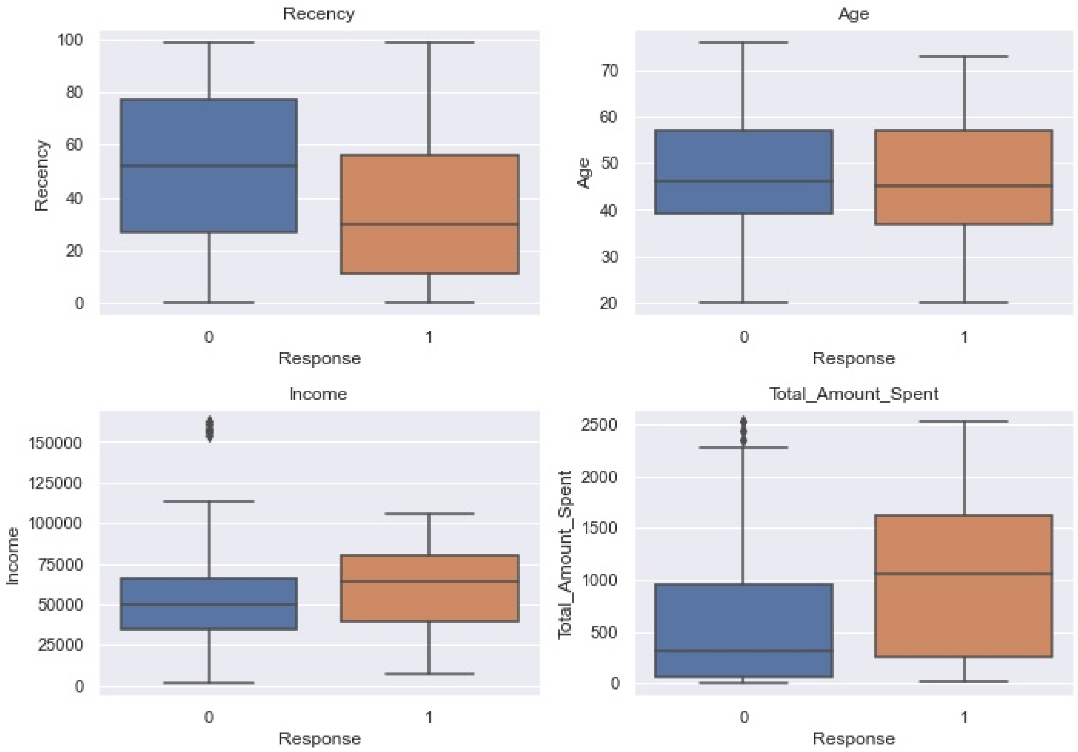


- Each plot shows that customer spending more on any product is more likely to take the offer.

In [137…

```
cols = data[["Recency", "Age", "Income", "Total_Amount_Spent"]].columns.tolist()
plt.figure(figsize=(10, 10))

for i, variable in enumerate(cols):
    plt.subplot(3, 2, i + 1)
    sns.boxplot(data["Response"], data[variable])
    plt.tight_layout()
    plt.title(variable)
plt.show()
```



- Customers with lower recency i.e. less number of days since the last purchase, are more likely to take the offer.
- Response does not depend much on age.
- Customers with higher income are more likely to take the offer.
- Customers who spent more in the last 2 years are more likely to take the offer.

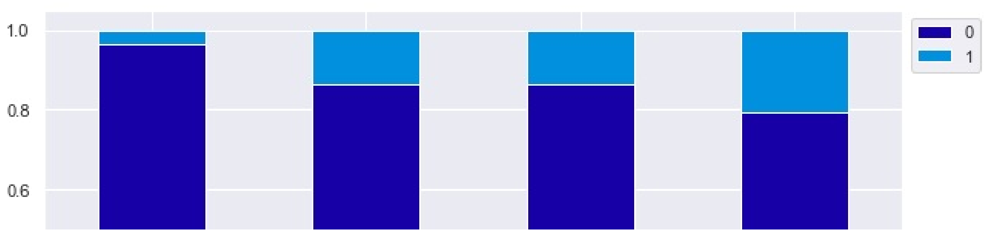
In [138…

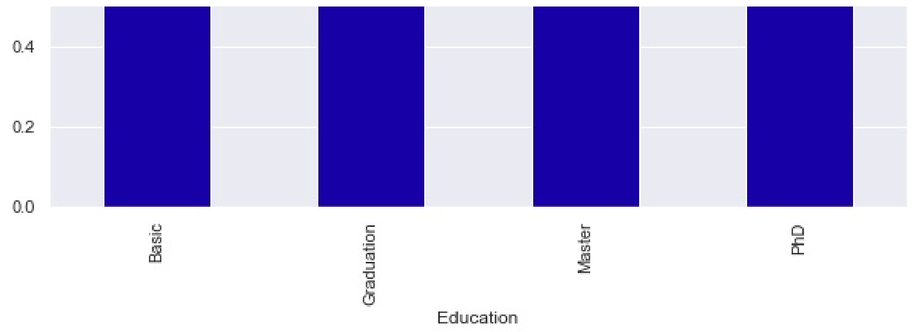
```
### Function to plot stacked bar charts for categorical columns
def stacked_plot(x):
    sns.set(palette="nipy_spectral")
    tab1 = pd.crosstab(x, data["Response"], margins=True)
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(x, data["Response"], normalize="index")
    tab.plot(kind="bar", stacked=True, figsize=(10, 5))
    plt.legend(loc="lower left", frameon=False)
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

In [139…

```
stacked_plot(data["Education"])
```

```
Response       0    1   All
Education
Basic         52    2    54
Graduation   974  152  1126
Master       494   79   573
PhD          385  101   486
All         1905  334  2239
------------------------------------------------------------------------------------------------------------------------
```



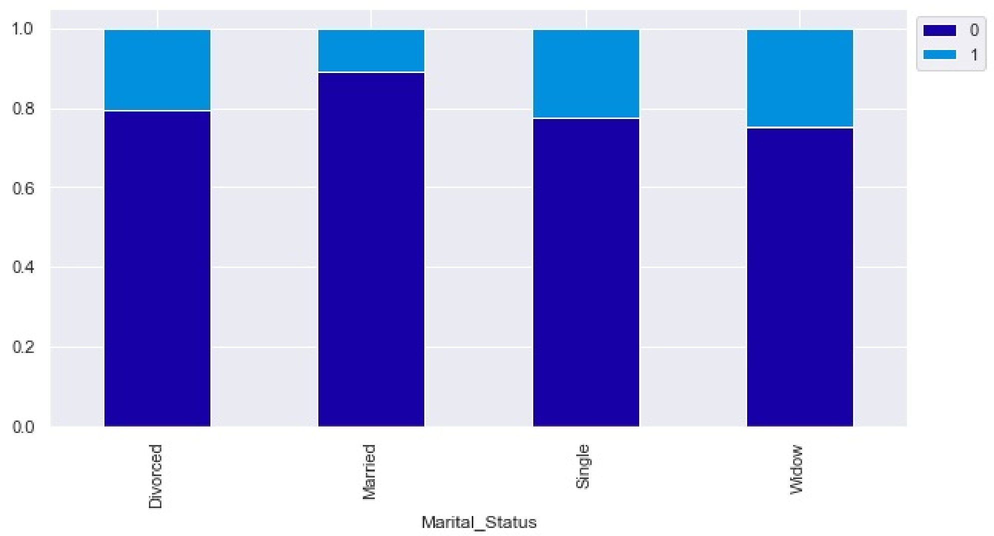


- We can see a clear trend here that customers with higher education are more likely to take the offer.

In [140…

```
stacked_plot(data["Marital_Status"])
```

```
Response           0    1   All
Marital_Status
Divorced         184   48   232
Married         1285  158  1443
Single           378  109   487
Widow             58   19    77
All             1905  334  2239
------------------------------------------------------------------------------------------------------------------------
-------
```

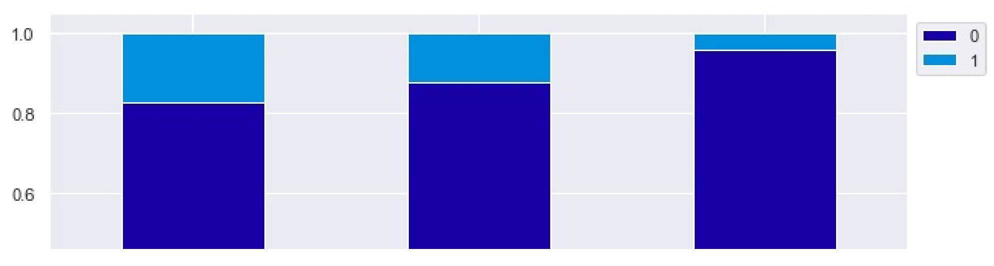


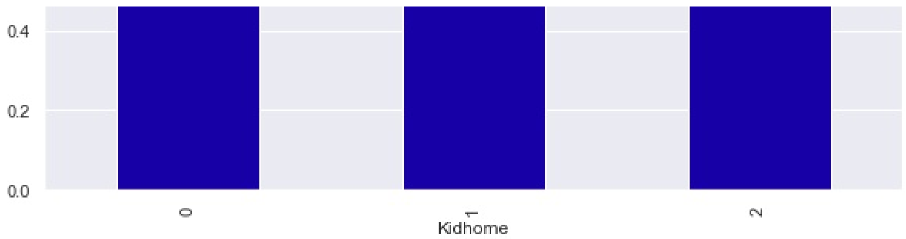
- We saw earlier that number of married customers is much more than single or divorced but divorced/widow customers are more likely to take the offer.
- Single customers are more likely to take the offer than married customers.

In [141…

```
stacked_plot(data["Kidhome"])
```

```
Response     0    1   All
Kidhome
0         1071  222  1293
1          788  110   898
2           46    2    48
All       1905  334  2239
------------------------------------------------------------------------------------------------------------------------
-------
```



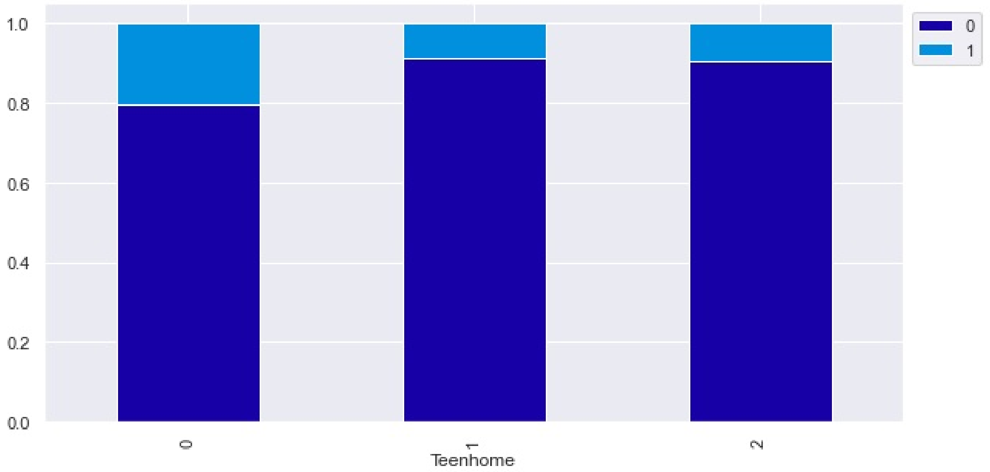


- We can see that the number of kids increases, chances of customers taking the offer decreases.
- Customers with no kids at home are more likely to take the offer which can be expected as this includes single customers as well.

In [142…

```
stacked_plot(data["Teenhome"])
```

```
Response     0    1   All
Teenhome
0          920  237  1157
1          938   92  1030
2           47    5    52
All       1905  334  2239
------------------------------------------------------------------------------------------------------------------------
-------
```

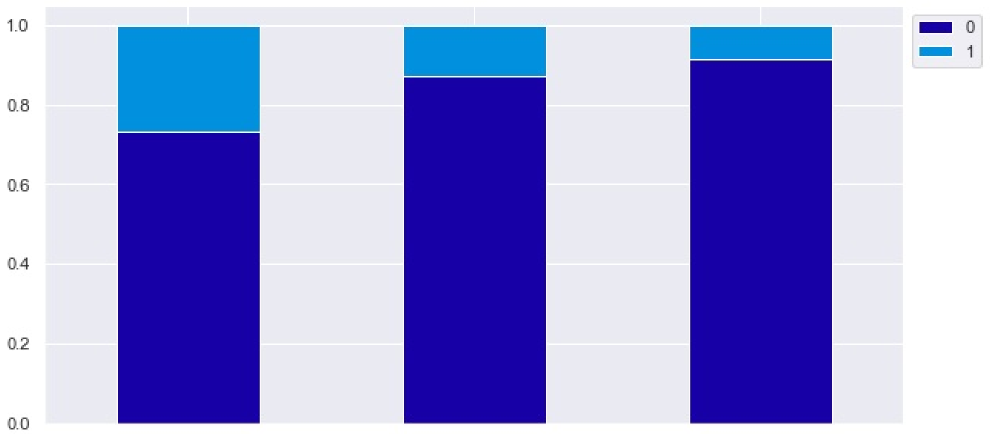


- Customers with no teens at home are most likely to take the offer.
- Customers with two teens are more likely to take the offer than customers with 1 teenager.

In [143…

```
stacked_plot(data["Reg_year"])
```

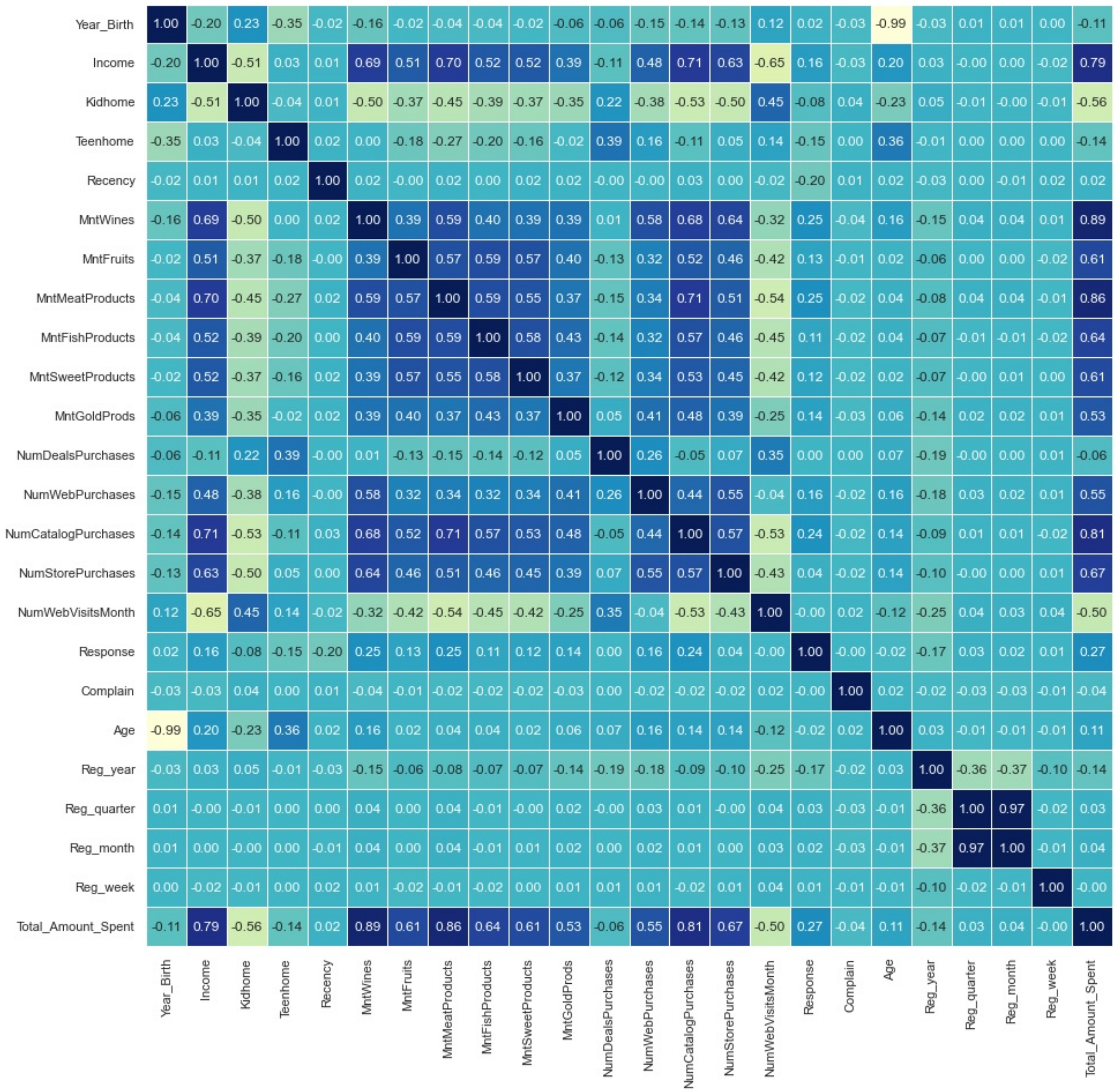
```
Response     0    1   All
Reg_year
2012       362  132   494
2013      1034  154  1188
2014       509   48   557
All       1905  334  2239
------------------------------------------------------------------------------------------------------------------------
-------
```



- Number of customers taking the offer is decreasing each subsequent year.
- Let's explore this further for month-wise distribution for each of the year.

In [144…

```
sns.set(rc={"figure.figsize": (15, 15)})
sns.heatmap(
    data.corr(),
    annot=True,
    linewidths=0.5,
    center=0,
    cbar=False,
    cmap="YlGnBu",
    fmt="0.2f",
)
plt.show()
```



- As expected, age and year of birth have a high negative correlation. We can drop one of them.
- Registration month, quarter and year columns are highly correlated which can be expected as we extracted these columns from the same column.
- We can drop one of the columns in a quarter or month as they are almost perfectly correlated.
- Total amount spent is correlated with variables they are associated with. We can drop this column.

Data Preparation

In [210…
```
education = {'Basic':1, 'Graduation':2, 'Master':3, 'PhD':4}
data['Education']=data['Education'].map(education)
marital_status = {'Married':1,'Single':2, 'Divorced':3, 'Widow':4}
data['Marital_Status']=data['Marital_Status'].map(marital_status)
```

Split the data into train and test sets

In [211…
```
# Separating target variable and other variables
X = data.drop(columns="Response")
Y = data["Response"]
```

In [212…
```
# Dropping birth year and Dt_Customer columns
X.drop(
    columns=[
        "Year_Birth",
        "Dt_Customer",
        "Reg_quarter",
        "Total_Amount_Spent",
    ],
    inplace=True,
)
```

In [213…
```
# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.30, random_state=1, stratify=Y
)
print(X_train.shape, X_test.shape)
```

```
(1567, 22) (672, 22)
```

Missing-Value Treatment

- We will use KNN imputer to impute missing values.
- `KNNImputer` : Each sample's missing values are imputed by looking at the n_neighbors nearest neighbors found in the training set. Default value for n_neighbors=5.
- KNN imputer replaces missing values using the average of k nearest non-missing feature values.
- Nearest points are found based on euclidean distance.

In [214…
```
imputer = KNNImputer(n_neighbors=5)
```

In [216…
```
#Fit and transform the train data
X_train=pd.DataFrame(imputer.fit_transform(X_train),columns=X_train.columns)

#Transform the test data
X_test=pd.DataFrame(imputer.fit_transform(X_test),columns=X_test.columns)
```

In [217…
```
#Checking that no column has missing values in train or test sets
print(X_train.isna().sum())
print('-'*30)
print(X_test.isna().sum())
```

```
Education              0
Marital_Status         0
Income                 0
Kidhome                0
Teenhome               0
Recency                0
MntWines               0
MntFruits              0
MntMeatProducts        0
MntFishProducts        0
MntSweetProducts       0
MntGoldProds           0
NumDealsPurchases      0
NumWebPurchases        0
NumCatalogPurchases    0
NumStorePurchases      0
NumWebVisitsMonth      0
```

```
Complain                  0
Age                       0
Reg_year                  0
Reg_month                 0
Reg_week                  0
dtype: int64
------------------------------
Education                 0
Marital_Status            0
Income                    0
Kidhome                   0
Teenhome                  0
Recency                   0
MntWines                  0
MntFruits                 0
MntMeatProducts           0
MntFishProducts           0
MntSweetProducts          0
MntGoldProds              0
NumDealsPurchases         0
NumWebPurchases           0
NumCatalogPurchases       0
NumStorePurchases         0
NumWebVisitsMonth         0
Complain                  0
Age                       0
Reg_year                  0
Reg_month                 0
Reg_week                  0
dtype: int64
```

- All missing values have been treated.
- Let's inverse map the encoded values.

In [218…

```
## Function to inverse the encoding
def inverse_mapping(x,y):
    inv_dict = {v: k for k, v in x.items()}
    X_train[y] = np.round(X_train[y]).map(inv_dict).astype('category')
    X_test[y] = np.round(X_test[y]).map(inv_dict).astype('category')
```

In [219…

```
inverse_mapping(education,'Education')
inverse_mapping(marital_status,'Marital_Status')
```

- Checking inverse mapped values/categories.

In [220…

```
cols = X_train.select_dtypes(include=['object','category'])
for i in cols.columns:
    print(X_train[i].value_counts())
    print('*'*30)
```

```
Graduation    793
Master        420
PhD           316
Basic          38
Name: Education, dtype: int64
******************************
Married     993
Single      346
Divorced    168
Widow        60
Name: Marital_Status, dtype: int64
******************************
```

In [221…

```
cols = X_test.select_dtypes(include=['object','category'])
for i in cols.columns:
    print(X_test[i].value_counts())
    print('*'*30)
```

```
Graduation    333
PhD           170
Master        153
Basic          16
Name: Education, dtype: int64
```

```
******************************
Married     450
Single      141
Divorced     64
Widow        17
Name: Marital_Status, dtype: int64
******************************
```

- Inverse mapping returned original labels.

Encoding categorical varaibles

In [223…

```
X_train=pd.get_dummies(X_train,drop_first=True)
X_test=pd.get_dummies(X_test,drop_first=True)
print(X_train.shape, X_test.shape)
```

```
(1567, 26) (672, 26)
```

- After encoding there are 26 columns.

Building the model

Model evaluation criterion:

Model can make wrong predictions as:

1. Predicting a customer will buy the product and the customer doesn't buy - Loss of resources
2. Predicting a customer will not buy the product and the customer buys - Loss of opportunity

Which case is more important?

- Predicting that customer will not buy the product but he buys i.e. losing on a potential source of income for the company because that customer will not be targeted by the marketing team when he should be targeted.

How to reduce this loss i.e need to reduce False Negatives?

- Company wants Recall to be maximized, greater the Recall lesser the chances of false negatives.

Let's start by building different models using KFold and cross_val_score with pipelines and tune the best model using GridSearchCV and RandomizedSearchCV

- `Stratified K-Folds cross-validation` provides dataset indices to split data into train/validation sets. Split dataset into k consecutive folds (without shuffling by default) keeping the distribution of both classes in each fold the same as the target variable. Each fold is then used once as validation while the k - 1 remaining folds form the training set.

In [157…

```
models = []  # Empty list to store all the models

# Appending pipelines for each model into the list
models.append(
    (
        "LR",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("log_reg", LogisticRegression(random_state=1)),
            ]
        ),
    )
)
models.append(
    (
        "RF",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("random_forest", RandomForestClassifier(random_state=1)),
            ]
        ),
    )
```

```python
)
models.append(
    (
        "GBM",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("gradient_boosting", GradientBoostingClassifier(random_state=1)),
            ]
        ),
    )
)
models.append(
    (
        "ADB",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("adaboost", AdaBoostClassifier(random_state=1)),
            ]
        ),
    )
)
models.append(
    (
        "XGB",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("xgboost", XGBClassifier(random_state=1,eval_metric='logloss')),
            ]
        ),
    )
)
models.append(
    (
        "DTREE",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("decision_tree", DecisionTreeClassifier(random_state=1)),
            ]
        ),
    )
)

results = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models

# loop through all models to get the mean cross validated score
for name, model in models:
    scoring = "recall"
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  # Setting number of splits equal to 5
    cv_result = cross_val_score(
        estimator=model, X=X_train, y=y_train, scoring=scoring, cv=kfold
    )
    results.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean() * 100))
```

```
LR: 30.814061054579096
RF: 19.22294172062905
GBM: 30.74005550416281
ADB: 37.6040703052729
XGB: 29.92599444958372
DTREE: 38.880666049953746
```

In [224…

```python
models = []  # Empty list to store all the models

# Appending pipelines for each model into the list
models.append(
    (
        "LR",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("log_reg", LogisticRegression(random_state=1)),
            ]
        ),
    )
)
models.append(
    (
```

```
            "RF",
            Pipeline(
                steps=[
                    ("scaler", StandardScaler()),
                    ("random_forest", RandomForestClassifier(random_state=1)),
                ]
            ),
        )
)
models.append(
        (
            "GBM",
            Pipeline(
                steps=[
                    ("scaler", StandardScaler()),
                    ("gradient_boosting", GradientBoostingClassifier(random_state=1)),
                ]
            ),
        )
)
models.append(
        (
            "ADB",
            Pipeline(
                steps=[
                    ("scaler", StandardScaler()),
                    ("adaboost", AdaBoostClassifier(random_state=1)),
                ]
            ),
        )
)
models.append(
        (
            "XGB",
            Pipeline(
                steps=[
                    ("scaler", StandardScaler()),
                    ("xgboost", XGBClassifier(random_state=1,eval_metric='logloss')),
                ]
            ),
        )
)
models.append(
        (
            "DTREE",
            Pipeline(
                steps=[
                    ("scaler", StandardScaler()),
                    ("decision_tree", DecisionTreeClassifier(random_state=1)),
                ]
            ),
        )
)

results = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models

# loop through all models to get the mean cross validated score
for name, model in models:
    scoring = "recall"
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  # Setting number of splits equal to 5
    cv_result = cross_val_score(
        estimator=model, X=X_train, y=y_train, scoring=scoring, cv=kfold
    )
    results.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean() * 100))
```

```
LR: 30.814061054579096
RF: 19.22294172062905
GBM: 30.74005550416281
ADB: 37.6040703052729
XGB: 29.92599444958372
DTREE: 38.880666049953746
```

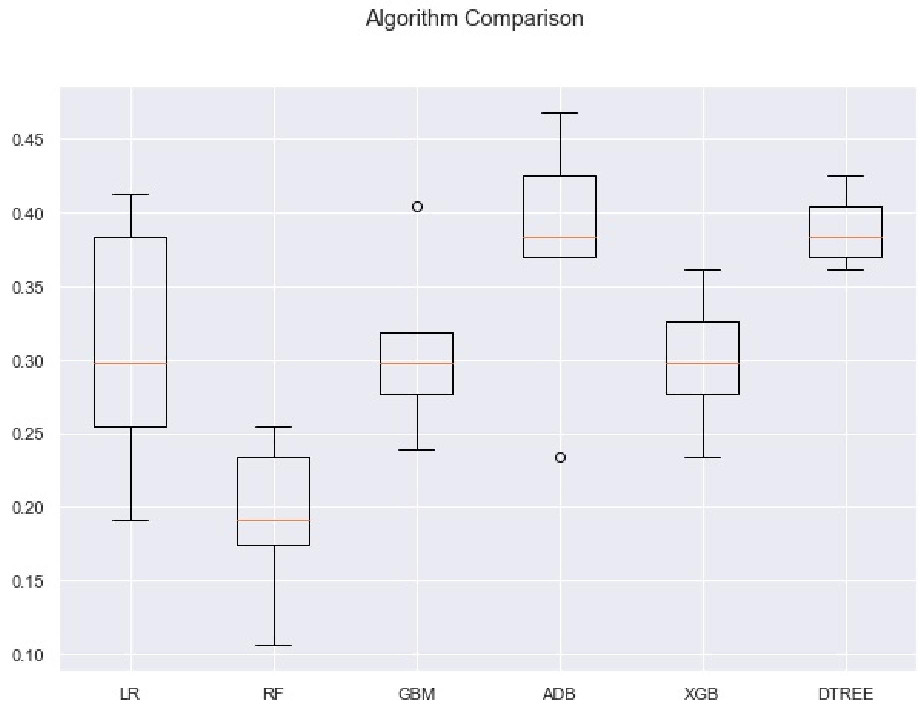
In [158…

```
# Plotting boxplots for CV scores of all models defined above
fig = plt.figure(figsize=(10, 7))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results)
ax.set_xticklabels(names)
```

```python
plt.show()
```



- We can see that AdaBoost is giving the highest cross-validated recall followed by XGBoost
- The boxplot shows that the performance of both the models is consistent with just one outlier for AdaBoost.
- We will tune both models - AdaBoost and XGBoost and see if the performance improves.

Hyperparameter Tuning

We will use pipelines with StandardScaler and AdaBoost model and tune the model using GridSearchCV and RandomizedSearchCV. We will also compare the performance and time taken by these two methods - grid search and randomized search.

We can also use the make_pipeline function instead of Pipeline to create a pipeline.

`make_pipeline` **: This is a shorthand for the Pipeline constructor; it does not require and does not permit, naming the estimators. Instead, their names will be set to the lowercase of their types automatically.**

First, let's create two functions to calculate different metrics and confusion matrix so that we don't have to use the same code repeatedly for each model.

In [225…

```python
##  Function to calculate different metric scores of the model - Accuracy, Recall and Precision
def get_metrics_score(model, flag=True):
    """
    model: classifier to predict values of X

    """
    # defining an empty list to store train and test results
    score_list = []

    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)

    train_acc = model.score(X_train, y_train)
    test_acc = model.score(X_test, y_test)

    train_recall = metrics.recall_score(y_train, pred_train)
    test_recall = metrics.recall_score(y_test, pred_test)

    train_precision = metrics.precision_score(y_train, pred_train)
    test_precision = metrics.precision_score(y_test, pred_test)

    score_list.extend(
        (
            train_acc,
            test_acc,
            train_recall,
            test_recall,
```

```python
            train_precision,
            test_precision,
        )
    )

    # If the flag is set to True then only the following print statements will be dispayed. The default value is
    if flag == True:
        print("Accuracy on training set : ", model.score(X_train, y_train))
        print("Accuracy on test set : ", model.score(X_test, y_test))
        print("Recall on training set : ", metrics.recall_score(y_train, pred_train))
        print("Recall on test set : ", metrics.recall_score(y_test, pred_test))
        print(
            "Precision on training set : ", metrics.precision_score(y_train, pred_train)
        )
        print("Precision on test set : ", metrics.precision_score(y_test, pred_test))

    return score_list  # returning the list with train and test scores
```

In [226…

```python
## Function to create confusion matrix
def make_confusion_matrix(model, y_actual, labels=[1, 0]):
    """
    model: classifier to predict values of X
    y_actual: ground truth

    """
    y_predict = model.predict(X_test)
    cm = metrics.confusion_matrix(y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(
        cm,
        index=[i for i in ["Actual - No", "Actual - Yes"]],
        columns=[i for i in ["Predicted - No", "Predicted - Yes"]],
    )
    group_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cm.flatten() / np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in zip(group_counts, group_percentages)]
    labels = np.asarray(labels).reshape(2, 2)
    plt.figure(figsize=(10, 7))
    sns.heatmap(df_cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

AdaBoost

GridSearchCV

In [161…

```python
%%time

# Creating pipeline
pipe = make_pipeline(StandardScaler(), AdaBoostClassifier(random_state=1))

# Parameter grid to pass in GridSearchCV
param_grid = {
    "adaboostclassifier__n_estimators": np.arange(10, 110, 10),
    "adaboostclassifier__learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "adaboostclassifier__base_estimator": [
        DecisionTreeClassifier(max_depth=1, random_state=1),
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

# Calling GridSearchCV
grid_cv = GridSearchCV(estimator=pipe, param_grid=param_grid, scoring=scorer, cv=5, n_jobs = -1)

# Fitting parameters in GridSeachCV
grid_cv.fit(X_train, y_train)

print(
    "Best Parameters:{} \nScore: {}".format(grid_cv.best_params_, grid_cv.best_score_)
)
```

```
Best Parameters:{'adaboostclassifier__base_estimator': DecisionTreeClassifier(max_depth=2, random_state=1), 'adab
oostclassifier__learning_rate': 1, 'adaboostclassifier__n_estimators': 100}
Score: 0.4830712303422756
CPU times: user 6.04 s, sys: 635 ms, total: 6.68 s
Wall time: 51.6 s
```

In [227…

```python
%%time

# Creating pipeline
pipe = make_pipeline(StandardScaler(), AdaBoostClassifier(random_state=1))

# Parameter grid to pass in GridSearchCV
param_grid = {
    "adaboostclassifier__n_estimators": np.arange(10, 110, 10),
    "adaboostclassifier__learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "adaboostclassifier__base_estimator": [
        DecisionTreeClassifier(max_depth=1, random_state=1),
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

# Calling GridSearchCV
grid_cv = GridSearchCV(estimator=pipe, param_grid=param_grid, scoring=scorer, cv=5, n_jobs = -1)

# Fitting parameters in GridSeachCV
grid_cv.fit(X_train, y_train)

print(
    "Best Parameters:{} \nScore: {}".format(grid_cv.best_params_, grid_cv.best_score_)
)
```

```
Best Parameters:{'adaboostclassifier__base_estimator': DecisionTreeClassifier(max_depth=2, random_state=1), 'adab
oostclassifier__learning_rate': 1, 'adaboostclassifier__n_estimators': 100}
Score: 0.4830712303422756
CPU times: user 5.97 s, sys: 638 ms, total: 6.61 s
Wall time: 58.5 s
```

In [162…

```python
# Creating new pipeline with best parameters
abc_tuned1 = make_pipeline(
    StandardScaler(),
    AdaBoostClassifier(
        base_estimator=DecisionTreeClassifier(max_depth=2, random_state=1),
        n_estimators=100,
        learning_rate=1,
        random_state=1,
    ),
)

# Fit the model on training data
abc_tuned1.fit(X_train, y_train)
```

Out[162…

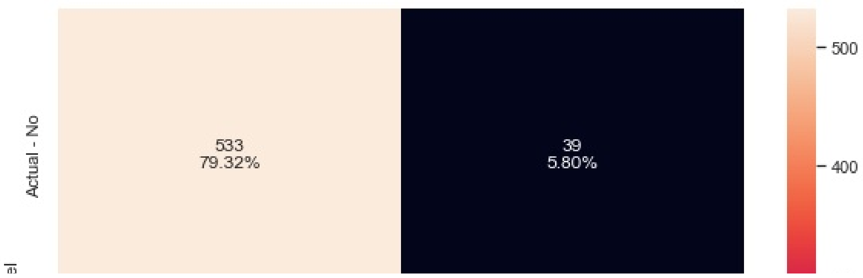
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('adaboostclassifier',
                 AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2,
                                                                          random_state=1),
                                    learning_rate=1, n_estimators=100,
                                    random_state=1))])
```

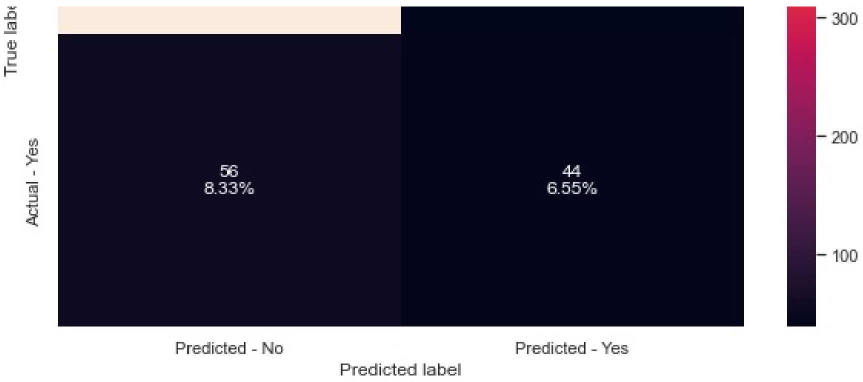
In [163…

```python
# Calculating different metrics
get_metrics_score(abc_tuned1)

# Creating confusion matrix
make_confusion_matrix(abc_tuned1, y_test)
```

```
Accuracy on training set :  0.9929802169751116
Accuracy on test set :  0.8586309523809523
Recall on training set :  0.9786324786324786
Recall on test set :  0.44
Precision on training set :  0.9744680851063829
Precision on test set :  0.5301204819277109
```





- The test recall has increased by ~7% as compared to cross-validated recall
- The tuned Adaboost model is slightly overfitting the training data
- The test recall is still less than 50% i.e. the model is not good at identifying potential customers who would take the offer.

RandomizedSearchCV

In [164…

```
%%time

# Creating pipeline
pipe = make_pipeline(StandardScaler(), AdaBoostClassifier(random_state=1))

# Parameter grid to pass in RandomizedSearchCV
param_grid = {
    "adaboostclassifier__n_estimators": np.arange(10, 110, 10),
    "adaboostclassifier__learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "adaboostclassifier__base_estimator": [
        DecisionTreeClassifier(max_depth=1, random_state=1),
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}
# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
abc_tuned2 = RandomizedSearchCV(estimator=pipe, param_distributions=param_grid, n_iter=50, scoring=scorer, cv=5,

#Fitting parameters in RandomizedSearchCV
abc_tuned2.fit(X_train,y_train)

print("Best parameters are {} with CV score={}:" .format(abc_tuned2.best_params_,abc_tuned2.best_score_))
```

```
Best parameters are {'adaboostclassifier__n_estimators': 100, 'adaboostclassifier__learning_rate': 1, 'adaboostcl
assifier__base_estimator': DecisionTreeClassifier(max_depth=2, random_state=1)} with CV score=0.4830712303422756:
CPU times: user 59 s, sys: 459 ms, total: 59.5 s
Wall time: 1min
```

In [228…

```
%%time

# Creating pipeline
pipe = make_pipeline(StandardScaler(), AdaBoostClassifier(random_state=1))

# Parameter grid to pass in RandomizedSearchCV
param_grid = {
    "adaboostclassifier__n_estimators": np.arange(10, 110, 10),
    "adaboostclassifier__learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "adaboostclassifier__base_estimator": [
        DecisionTreeClassifier(max_depth=1, random_state=1),
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}
# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
abc_tuned2 = RandomizedSearchCV(estimator=pipe, param_distributions=param_grid, n_iter=50, scoring=scorer, cv=5,

#Fitting parameters in RandomizedSearchCV
abc_tuned2.fit(X_train,y_train)

print("Best parameters are {} with CV score={}:" .format(abc_tuned2.best_params_,abc_tuned2.best_score_))
```

```
Best parameters are {'adaboostclassifier__n_estimators': 100, 'adaboostclassifier__learning_rate': 1, 'adaboostcl
assifier__base_estimator': DecisionTreeClassifier(max_depth=2, random_state=1)} with CV score=0.4830712303422756:
CPU times: user 2.17 s, sys: 174 ms, total: 2.34 s
Wall time: 18.9 s
```

In [229…

```
# Creating new pipeline with best parameters
abc_tuned2 = make_pipeline(
    StandardScaler(),
    AdaBoostClassifier(
        base_estimator=DecisionTreeClassifier(max_depth=2, random_state=1),
        n_estimators=100,
        learning_rate=1,
        random_state=1,
    ),
)

# Fit the model on training data
abc_tuned2.fit(X_train, y_train)
```

Out[229…

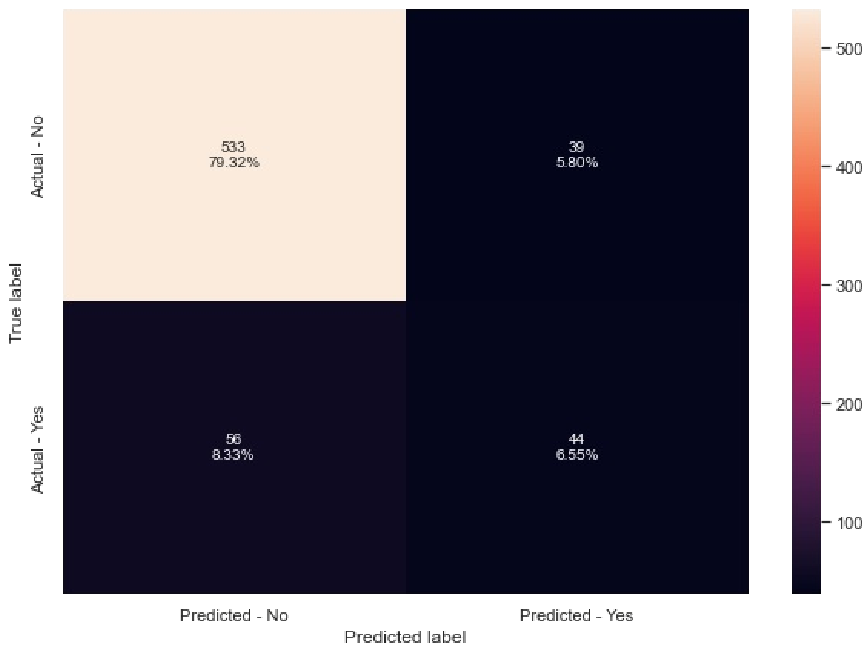
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('adaboostclassifier',
                 AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2,
                                                                          random_state=1),
                                    learning_rate=1, n_estimators=100,
                                    random_state=1))])
```

In [230…

```
# Calculating different metrics
get_metrics_score(abc_tuned2)

# Creating confusion matrix
make_confusion_matrix(abc_tuned2, y_test)
```

```
Accuracy on training set :  0.9929802169751116
Accuracy on test set :  0.8586309523809523
Recall on training set :  0.9786324786324786
Recall on test set :  0.44
Precision on training set :  0.9744680851063829
Precision on test set :  0.5301204819277109
```



- Grid search took a significantly longer time than random search. This difference would further increase as the number of parameters increases but the parameters from random search are the same as compared grid search.
- This can happen by chance but it is not guaranteed to happen for each algorithm.

XGBoost

GridSearchCV

In [167…

```
%%time

#Creating pipeline
pipe=make_pipeline(StandardScaler(), XGBClassifier(random_state=1,eval_metric='logloss'))

#Parameter grid to pass in GridSearchCV
param_grid={'xgbclassifier__n_estimators':np.arange(50,300,50),'xgbclassifier__scale_pos_weight':[0,1,2,5,10],
            'xgbclassifier__learning_rate':[0.01,0.1,0.2,0.05], 'xgbclassifier__gamma':[0,1,3,5],
            'xgbclassifier__subsample':[0.7,0.8,0.9,1]}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling GridSearchCV
grid_cv = GridSearchCV(estimator=pipe, param_grid=param_grid, scoring=scorer, cv=5, n_jobs = -1)

#Fitting parameters in GridSeachCV
grid_cv.fit(X_train,y_train)


print("Best parameters are {} with CV score={}:" .format(grid_cv.best_params_,grid_cv.best_score_))
```

```
Best parameters are {'xgbclassifier__gamma': 3, 'xgbclassifier__learning_rate': 0.01, 'xgbclassifier__n_estimator
s': 150, 'xgbclassifier__scale_pos_weight': 10, 'xgbclassifier__subsample': 0.8} with CV score=0.8765032377428307
:
CPU times: user 59.6 s, sys: 5.28 s, total: 1min 4s
Wall time: 11min 47s
```

In [168…

```
%%time

#Creating pipeline
pipe=make_pipeline(StandardScaler(), XGBClassifier(random_state=1,eval_metric='logloss'))

#Parameter grid to pass in GridSearchCV
param_grid={'xgbclassifier__n_estimators':np.arange(50,300,50),'xgbclassifier__scale_pos_weight':[0,1,2,5,10],
            'xgbclassifier__learning_rate':[0.01,0.1,0.2,0.05], 'xgbclassifier__gamma':[0,1,3,5],
            'xgbclassifier__subsample':[0.7,0.8,0.9,1]}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling GridSearchCV
grid_cv = GridSearchCV(estimator=pipe, param_grid=param_grid, scoring=scorer, cv=5, n_jobs = -1)

#Fitting parameters in GridSeachCV
grid_cv.fit(X_train,y_train)


print("Best parameters are {} with CV score={}:" .format(grid_cv.best_params_,grid_cv.best_score_))
```

```
Best parameters are {'xgbclassifier__gamma': 3, 'xgbclassifier__learning_rate': 0.01, 'xgbclassifier__n_estimator
s': 150, 'xgbclassifier__scale_pos_weight': 10, 'xgbclassifier__subsample': 0.8} with CV score=0.8765032377428307
:
CPU times: user 57.8 s, sys: 5.06 s, total: 1min 2s
Wall time: 10min 2s
```

In [169…

```
# Creating new pipeline with best parameters
xgb_tuned1 = make_pipeline(
    StandardScaler(),
    XGBClassifier(
        random_state=1,
        n_estimators=150,
        scale_pos_weight=10,
        subsample=0.8,
        learning_rate=0.01,
        gamma=3,
        eval_metric='logloss',
    ),
)

# Fit the model on training data
xgb_tuned1.fit(X_train, y_train)
```

Out[169…

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('xgbclassifier',
                 XGBClassifier(eval_metric='logloss', gamma=3,
                               learning_rate=0.01, n_estimators=150,
                               random_state=1, scale_pos_weight=10,
```

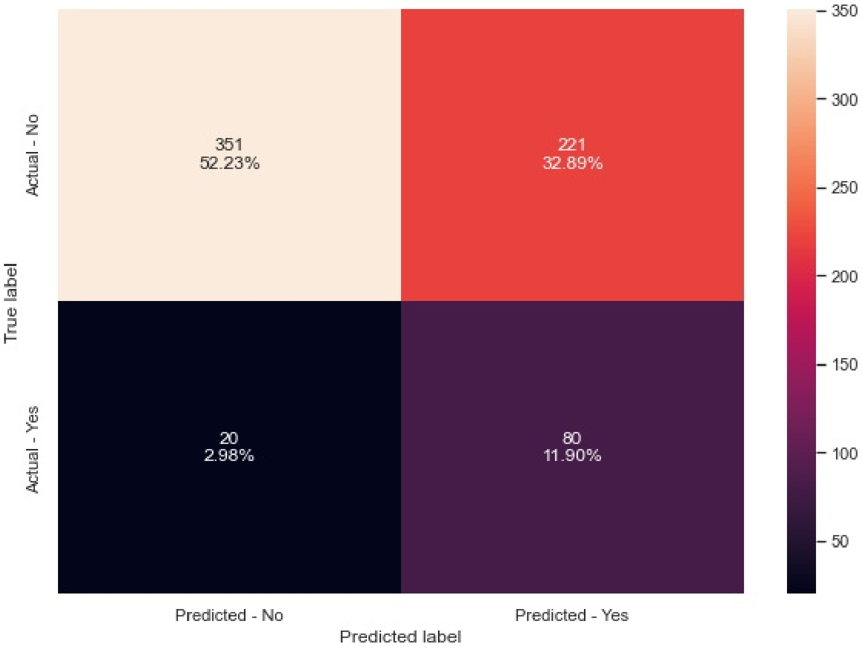
```
                                  subsample=0.8))])
```

In [170…

```
# Calculating different metrics
get_metrics_score(xgb_tuned1)

# Creating confusion matrix
make_confusion_matrix(xgb_tuned1, y_test)
```

```
Accuracy on training set :  0.6617740906190173
Accuracy on test set :  0.6413690476190477
Recall on training set :  0.9658119658119658
Recall on test set :  0.8
Precision on training set :  0.30213903743315507
Precision on test set :  0.26578073089701
```



- The test recall has increased by ~40% as compared to the result from cross-validation with default parameters.

RandomizedSearchCV

In [171…

```
%%time

#Creating pipeline
pipe=make_pipeline(StandardScaler(),XGBClassifier(random_state=1,eval_metric='logloss', n_estimators = 50))

#Parameter grid to pass in RandomizedSearchCV
param_grid={'xgbclassifier__n_estimators':np.arange(50,300,50),
            'xgbclassifier__scale_pos_weight':[0,1,2,5,10],
            'xgbclassifier__learning_rate':[0.01,0.1,0.2,0.05],
            'xgbclassifier__gamma':[0,1,3,5],
            'xgbclassifier__subsample':[0.7,0.8,0.9,1],
           'xgbclassifier__max_depth':np.arange(1,10,1),
            'xgbclassifier__reg_lambda':[0,1,2,5,10]}




# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=pipe, param_distributions=param_grid, n_iter=50, scoring=scorer, cv=

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train,y_train)

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,randomized_cv.best_score_))
```

```
Best parameters are {'xgbclassifier__subsample': 0.8, 'xgbclassifier__scale_pos_weight': 10, 'xgbclassifier__reg_
lambda': 2, 'xgbclassifier__n_estimators': 50, 'xgbclassifier__max_depth': 2, 'xgbclassifier__learning_rate': 0.0
```

```
5, 'xgbclassifier__gamma': 3} with CV score=0.9148011100832563:
CPU times: user 1min 32s, sys: 211 ms, total: 1min 32s
Wall time: 1min 33s
```

In [231…

```
%%time

#Creating pipeline
pipe=make_pipeline(StandardScaler(),XGBClassifier(random_state=1,eval_metric='logloss', n_estimators = 50))

#Parameter grid to pass in RandomizedSearchCV
param_grid={'xgbclassifier__n_estimators':np.arange(50,300,50),
            'xgbclassifier__scale_pos_weight':[0,1,2,5,10],
            'xgbclassifier__learning_rate':[0.01,0.1,0.2,0.05],
            'xgbclassifier__gamma':[0,1,3,5],
            'xgbclassifier__subsample':[0.7,0.8,0.9,1],
           'xgbclassifier__max_depth':np.arange(1,10,1),
            'xgbclassifier__reg_lambda':[0,1,2,5,10]}




# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=pipe, param_distributions=param_grid, n_iter=50, scoring=scorer, cv=

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train,y_train)

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,randomized_cv.best_score_))
```

```
Best parameters are {'xgbclassifier__subsample': 0.8, 'xgbclassifier__scale_pos_weight': 10, 'xgbclassifier__reg_
lambda': 2, 'xgbclassifier__n_estimators': 50, 'xgbclassifier__max_depth': 2, 'xgbclassifier__learning_rate': 0.0
5, 'xgbclassifier__gamma': 3} with CV score=0.9148011100832563:
CPU times: user 1min 41s, sys: 735 ms, total: 1min 42s
Wall time: 1min 43s
```

In [233…

```
randomized_cv.best_params_
```

Out[233…

```
{'xgbclassifier__subsample': 0.8,
 'xgbclassifier__scale_pos_weight': 10,
 'xgbclassifier__reg_lambda': 2,
 'xgbclassifier__n_estimators': 50,
 'xgbclassifier__max_depth': 2,
 'xgbclassifier__learning_rate': 0.05,
 'xgbclassifier__gamma': 3}
```

In [172…

```
# Creating new pipeline with best parameters
xgb_tuned2 = Pipeline(
    steps=[
        ("scaler", StandardScaler()),
        (
            "XGB",
            XGBClassifier(
                random_state=1,
                n_estimators=randomized_cv.best_params_['xgbclassifier__n_estimators'],
                scale_pos_weight=10,
                gamma=3,
                subsample=0.9,
                learning_rate= 0.05,
                eval_metric='logloss', max_depth = 2, reg_lambda = 2
            ),
        ),
    ]
)
# Fit the model on training data
xgb_tuned2.fit(X_train, y_train)
```

Out[172…

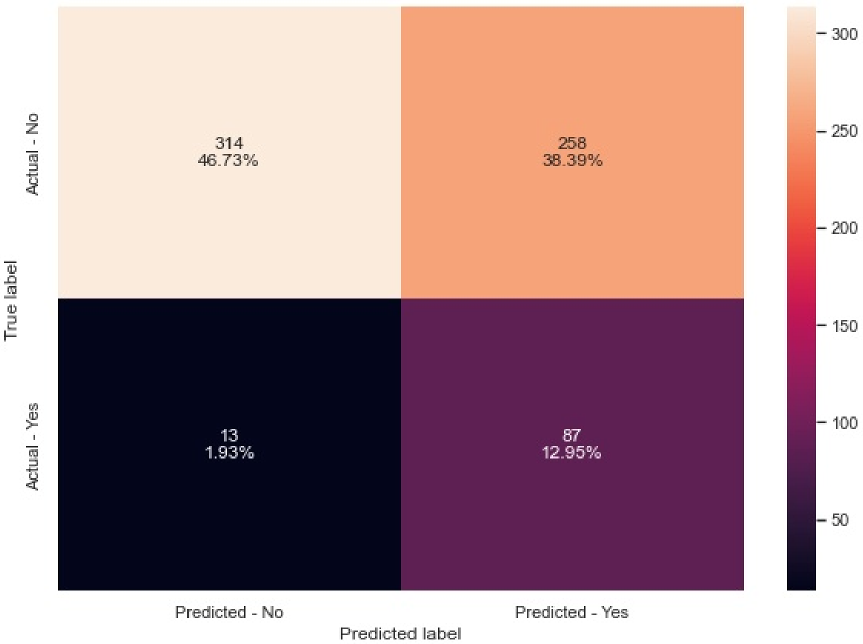
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('XGB',
                 XGBClassifier(eval_metric='logloss', gamma=3,
                               learning_rate=0.05, max_depth=2, n_estimators=50,
                               random_state=1, reg_lambda=2,
                               scale_pos_weight=10, subsample=0.9))])
```

In [173…

```
# Calculating different metrics
get_metrics_score(xgb_tuned2)

# Creating confusion matrix
make_confusion_matrix(xgb_tuned2, y_test)
```

```
Accuracy on training set :  0.5851946394384173
Accuracy on test set :  0.5967261904761905
Recall on training set :  0.9529914529914529
Recall on test set :  0.87
Precision on training set :  0.25870069605568446
Precision on test set :  0.25217391304347825
```



- Random search is giving better results than Grid search.
- The test recall has increased as compared to the test recall from grid search but the accuracy and precision have decreased.
- The overfitting in the model has also decreased

Comparing all models

In [234…

```
# defining list of models
models = [abc_tuned1, abc_tuned2, xgb_tuned1, xgb_tuned2]

# defining empty lists to add train and test results
acc_train = []
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []

# looping through all the models to get the metrics score - Accuracy, Recall and Precision
for model in models:

    j = get_metrics_score(model, False)
    acc_train.append(j[0])
    acc_test.append(j[1])
    recall_train.append(j[2])
    recall_test.append(j[3])
    precision_train.append(j[4])
    precision_test.append(j[5])
```

In [235…

```
comparison_frame = pd.DataFrame(
    {
        "Model": [
            "Adaboost with GridSearchCV",
            "Adaboost with RandomizedSearchCV",
            "XGBoost with GridSearchCV",
            "XGBoost with RandomizedSearchCV",
```

```
        ],
        "Train_Accuracy": acc_train,
        "Test_Accuracy": acc_test,
        "Train_Recall": recall_train,
        "Test_Recall": recall_test,
        "Train_Precision": precision_train,
        "Test_Precision": precision_test,
    }
)

# Sorting models in decreasing order of test recall
comparison_frame.sort_values(by="Test_Recall", ascending=False)
```

Out[235…

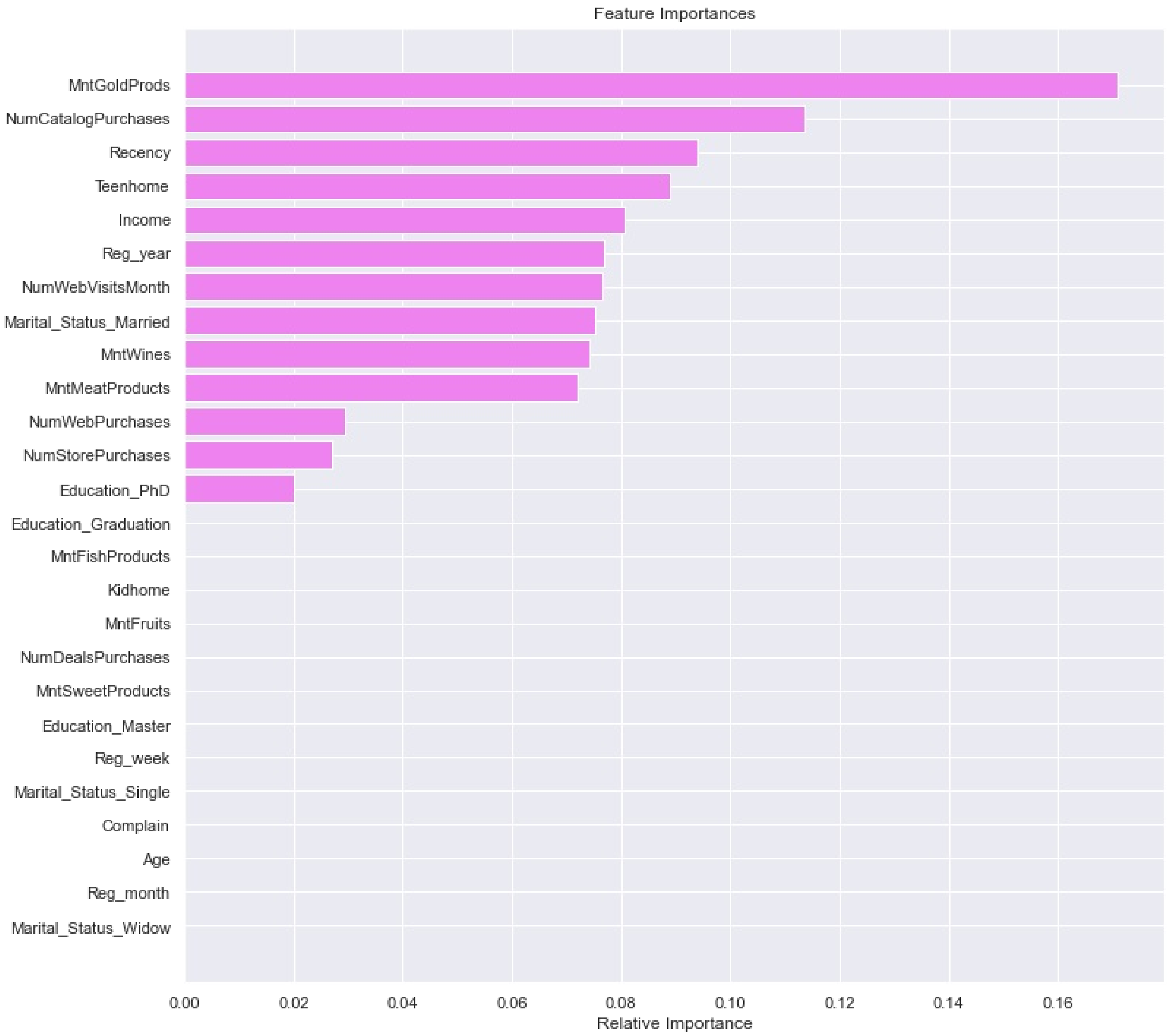
	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision
3	XGBoost with RandomizedSearchCV	0.585195	0.596726	0.952991	0.87	0.258701	0.252174
2	XGBoost with GridSearchCV	0.661774	0.641369	0.965812	0.80	0.302139	0.265781
0	Adaboost with GridSearchCV	0.992980	0.858631	0.978632	0.44	0.974468	0.530120
1	Adaboost with RandomizedSearchCV	0.992980	0.858631	0.978632	0.44	0.974468	0.530120

- The xgboost model tuned using randomised search is giving the best test recall of 0.87 but it has the least train and test precision.
- Let's see the feature importance from the tuned xgboost model

In [236…

```
feature_names = X_train.columns
importances = xgb_tuned2[1].feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12, 12))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



- Amount spent on gold products is the most important feature, followed by NumCatalogPurchases and the Recency of the customer.

Business Recommendations

- Company should target customers who buy premium products - gold products or high-quality wines - as these customers can spend more and are more likely to purchase the offer. The company should further launch premium offers for such customers. Such offers can also be extended to customers with higher income.
- We observed in our analysis that ~64% of customers are married but single customers, including divorced and widowed, are more likely to take the offer. The company should expand its customers by customizing offers to attract more single customers.
- Customers who are frequent buyers, should be targeted more by the company and offer them added benefits.
- Total amount spent has decreased over the years which shows that either our product qualities have declined or the company lacks marketing strategies. The company should constantly improve its marketing strategies to address such issues.
- Our analysis showed that ~99% of customers had no complaints in the last two years which can be due to the lack of feedback options for customers. The company should create easy mechanisms to gather feedback from the customers and use it to identify major concerns if any.
- The number of web visits is an important feature and the company should work on customizing its website to allow more traffic on the website. The company can improve the interface and provide easy check-in, check-out and delivery options.

In [ ]:

```
```