Name: Gauri Surti
Net ID: gas210004

MKT-6373.501 : Introduction to Programming for Analytics

Class Project - Report

Steps followed for the analysis and processing of the data are as below:

1. Imported the required packages in python.

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
import scipy.cluster.hierarchy as sch
import tensorflow as tf
from sklearn.metrics import confusion_matrix, classification_report
```

2. Loaded the data from marketing.csv file using read_csv function from pandas and added age and customer_hist columns in the dataset

3. Basic statistics of the data was found using describe function and below is the output.(Question 1)

```
In [183]: df.describe()
Out[183]:
                 ID   Year_Birth         Income      Kidhome     Teenhome  \
count   2240.000000  2240.000000    2216.000000  2240.000000  2240.000000
mean    5592.159821  1968.805804   52247.251354     0.444196     0.506250
std     3246.662198    11.984069   25173.076661     0.538398     0.544538
min        0.000000  1893.000000    1730.000000     0.000000     0.000000
25%     2828.250000  1959.000000   35303.000000     0.000000     0.000000
50%     5458.500000  1970.000000   51381.500000     0.000000     0.000000
75%     8427.750000  1977.000000   68522.000000     1.000000     1.000000
max    11191.000000  1996.000000  666666.000000     2.000000     2.000000

           Recency     MntWines    MntFruits  MntMeatProducts  \
count  2240.000000  2240.000000  2240.000000      2240.000000
mean     49.109375   303.935714    26.302232       166.950000
std      28.962453   336.597393    39.773434       225.715373
min       0.000000     0.000000     0.000000         0.000000
25%      24.000000    23.750000     1.000000        16.000000
50%      49.000000   173.500000     8.000000        67.000000
75%      74.000000   504.250000    33.000000       232.000000
max      99.000000  1493.000000   199.000000      1725.000000
```

	MntFishProducts	MntSweetProducts	MntGoldProds	NumDealsPurchases
count	2240.000000	2240.000000	2240.000000	2240.000000
mean	37.525446	27.062946	44.021875	2.325000
std	54.628979	41.280498	52.167439	1.932238
min	0.000000	0.000000	0.000000	0.000000
25%	3.000000	1.000000	9.000000	1.000000
50%	12.000000	8.000000	24.000000	2.000000
75%	50.000000	33.000000	56.000000	3.000000
max	259.000000	263.000000	362.000000	15.000000

	NumWebPurchases	NumCatalogPurchases	NumStorePurchases
count	2240.000000	2240.000000	2240.000000
mean	4.084821	2.662054	5.790179
std	2.778714	2.923101	3.250958
min	0.000000	0.000000	0.000000
25%	2.000000	0.000000	3.000000
50%	4.000000	2.000000	5.000000
75%	6.000000	4.000000	8.000000
max	27.000000	28.000000	13.000000

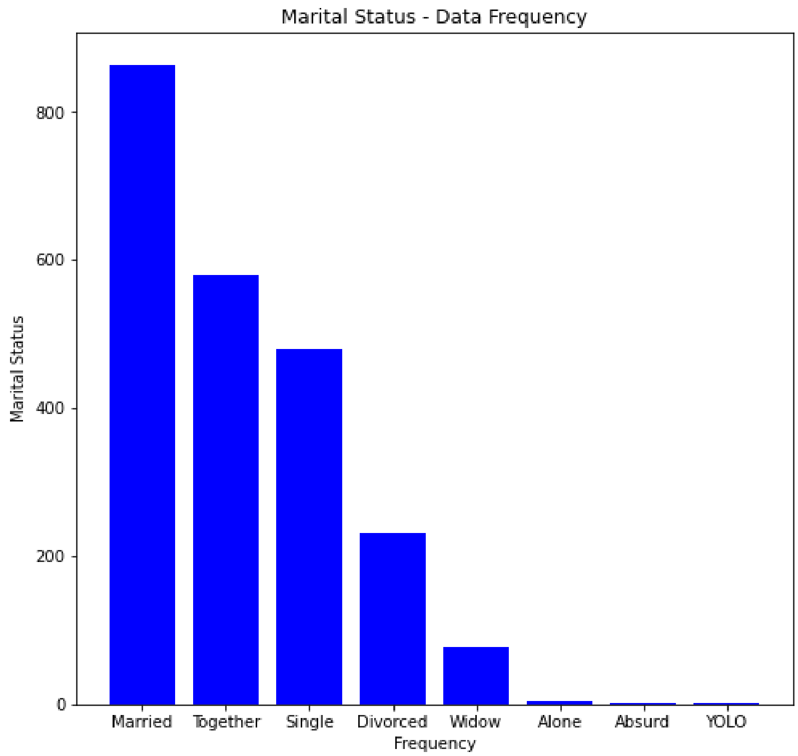
	NumWebVisitsMonth	AcceptedCmp3	AcceptedCmp4	AcceptedCmp5
count	2240.000000	2240.000000	2240.000000	2240.000000
mean	5.316518	0.072768	0.074554	0.072768
std	2.426645	0.259813	0.262728	0.259813
min	0.000000	0.000000	0.000000	0.000000
25%	3.000000	0.000000	0.000000	0.000000
50%	6.000000	0.000000	0.000000	0.000000
75%	7.000000	0.000000	0.000000	0.000000
max	20.000000	1.000000	1.000000	1.000000

	AcceptedCmp1	AcceptedCmp2	Complain	Z_CostContact	Z_Revenue
count	2240.000000	2240.000000	2240.000000	2240.0	2240.0
mean	0.064286	0.013393	0.009375	3.0	11.0
std	0.245316	0.114976	0.096391	0.0	0.0
min	0.000000	0.000000	0.000000	3.0	11.0
25%	0.000000	0.000000	0.000000	3.0	11.0
50%	0.000000	0.000000	0.000000	3.0	11.0
75%	0.000000	0.000000	0.000000	3.0	11.0
max	1.000000	1.000000	1.000000	3.0	11.0

	Response	Customer_hist	Age
count	2240.000000	2240.000000	2240.000000
mean	0.149107	8.915625	53.194196
std	0.356274	0.683191	11.984069
min	0.000000	8.000000	26.000000
25%	0.000000	8.000000	45.000000
50%	0.000000	9.000000	52.000000
75%	0.000000	9.000000	63.000000
max	1.000000	10.000000	129.000000

4. Analysis was done for the provided data using plots and below are some of the observations.(Question 2)

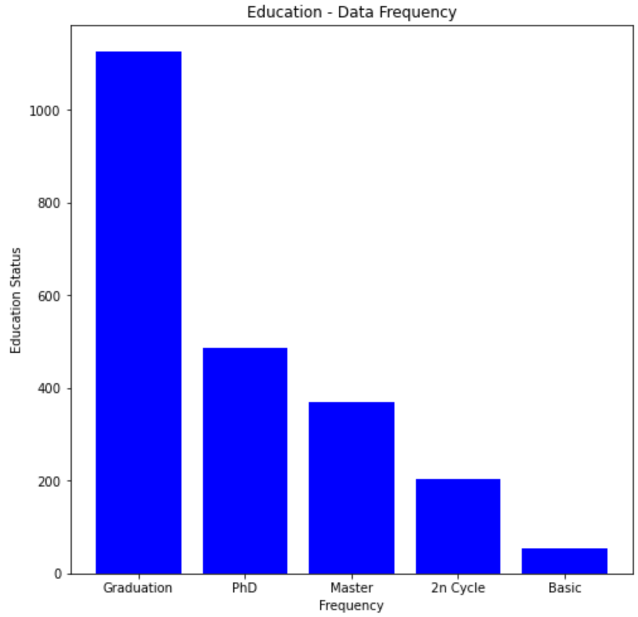
- Number of samples according to Marital Status



Number of Samples According to Marital Status

Married	864
Together	580
Single	480
Divorced	232
Widow	77
Alone	3
Absurd	2
YOLO	2

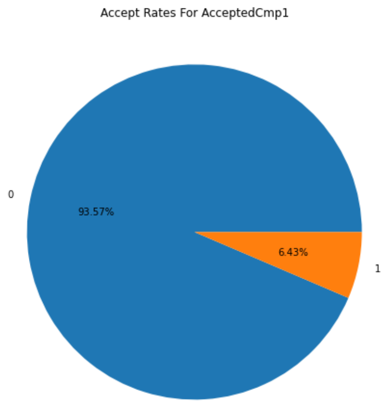
- Number of samples according to education

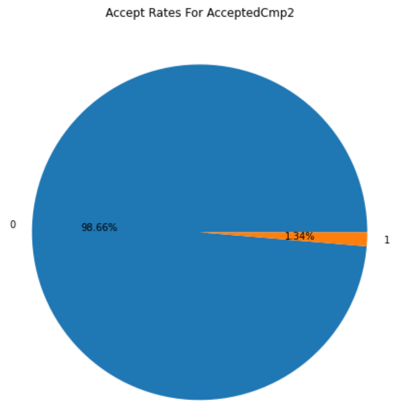


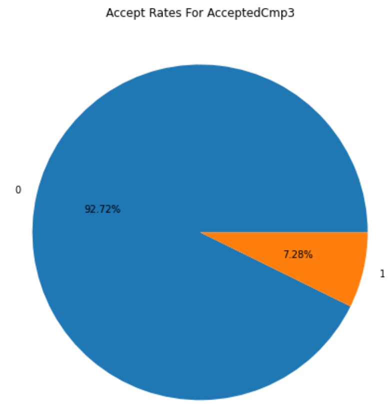
Number of Samples According to Education

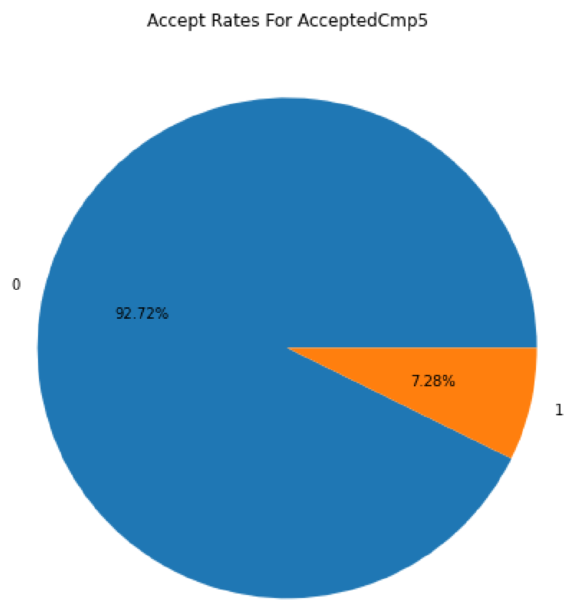
Graduation	1127
PhD	486
Master	370
2n Cycle	203
Basic	54

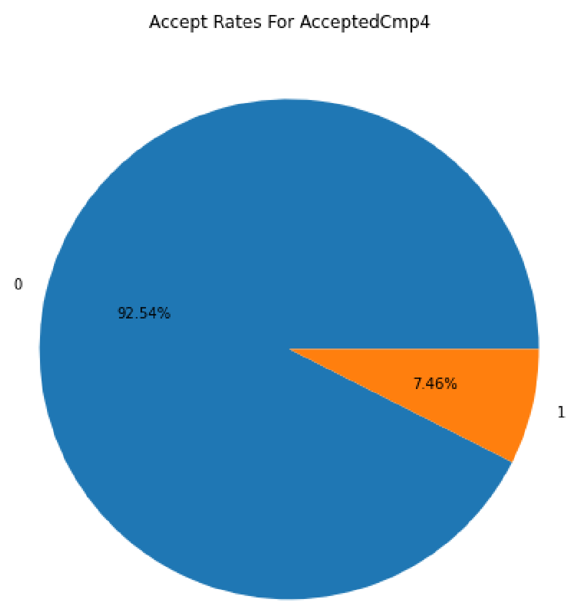
- Campaign Accept Rates





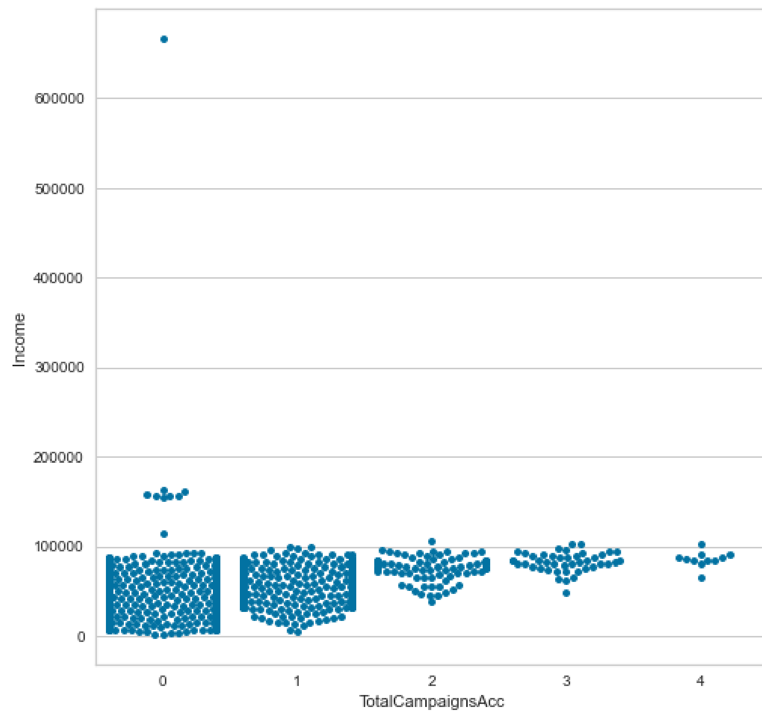






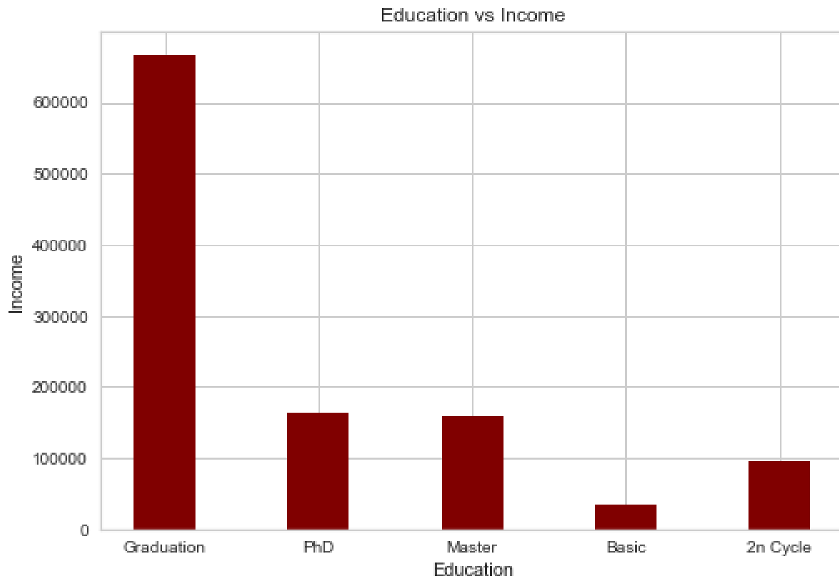
- The acceptance rate for the campaign "AcceptedCmp1" is around 6.43 %
- The acceptance rate for the campaign "AcceptedCmp2" is around 1.34 %
- The acceptance rate for the campaign "AcceptedCmp3" is around 7.28 %
- The acceptance rate for the campaign "AcceptedCmp4" is around 7.46 %
- The acceptance rate for the campaign "AcceptedCmp5" is around 7.28 %

- No of campaigns accepted Vs Income



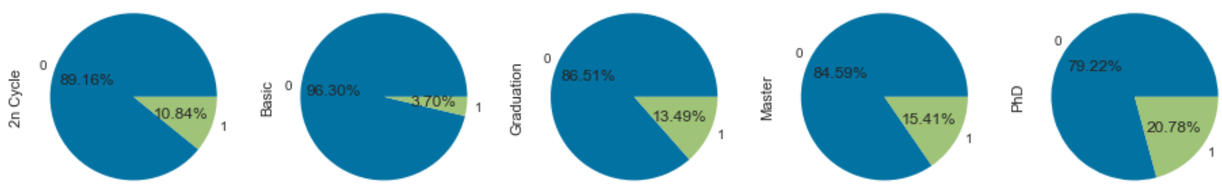
- Higher the income higher the number of campaigns accepted.

- Education Vs Income



  - ➢ Customer's with Graduation level of education have the highest yearly household income followed by customer's with Master's degree and PhD, followed by others.

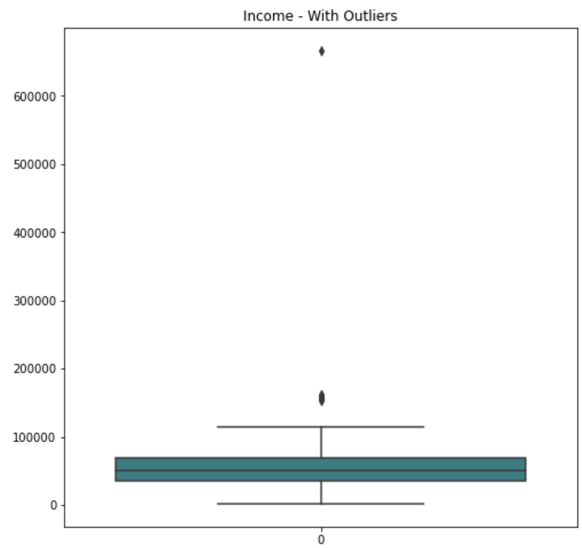
- Response Rate according to education

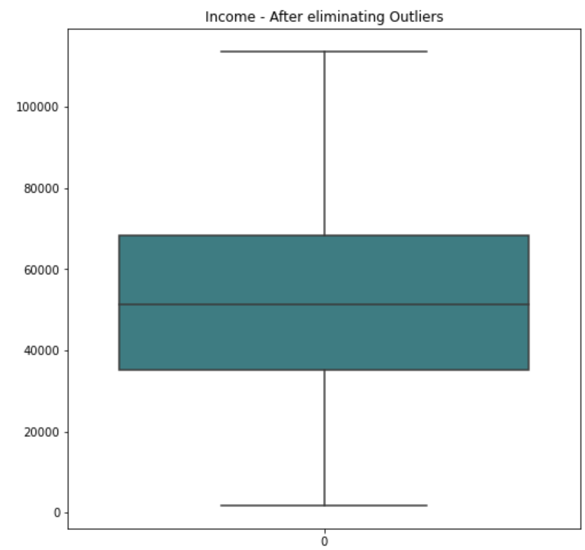


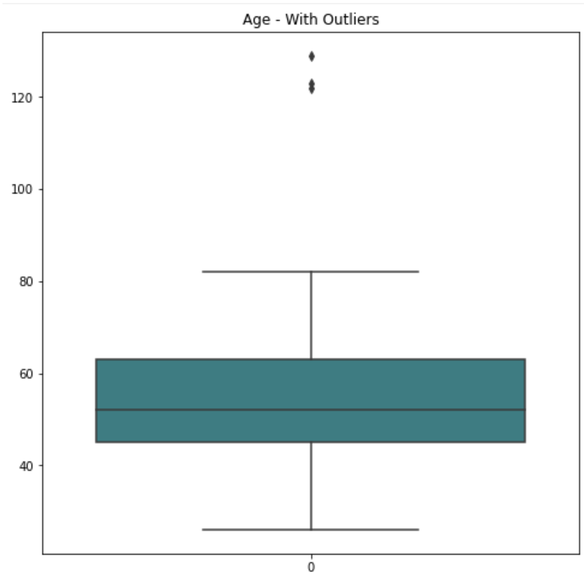
  - ➢ 13.49% of the customers with Graduation degree accepted the offer in last campaign.
  - ➢ 15.41% of the customers with Masters degree accepted the offer in last campaign.
  - ➢ 20.78% of the customers with PhD degree accepted the offer in last campaign.
  - ➢ 3.70% of the customers with Basic degree accepted the offer in last campaign.
  - ➢ 10.84% of the customers with 2^{nd} Cycle degree accepted the offer in last campaign.

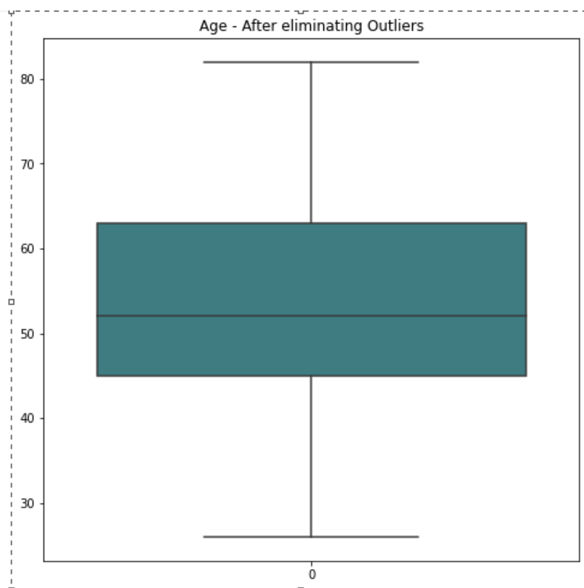
5. Data Pre-processing, Cleaning and Manipulation:
   - We first made a copy of existing dataset and named it new_df.
   - Dropping of 'NA' and duplicate valued rows from the existing rows
   - Converted categorical valued rows like Marital_Status and Education into Numerical values.
   - Merged some columns as below into a single column.
     - ➢ Created a column 'Spent' by adding columns MntWines, MntFruits, MntMeatProducts, MntFishProducts, MntSweetProducts, MntGoldProds.
     - ➢ Created column 'Purchase' by adding NumStorePurchases, NumWebPurchases, NumCatalogPurchases, NumDealsPurchases

- Dropped below columns which were not required and were creating semantic confusion.
  ID, Year_Birth, Education, Marital_Status, Kidhome, Teenhome, Dt_Customer, MntWines, MntFruits, MntMeatProducts, MntFishProducts, MntSweetProducts, MntGoldProds, NumStorePurchases, NumWebPurchases, NumCatalogPurchases, NumDealsPurchases, NumWebVisitsMonth, AcceptedCmp3,AcceptedCmp4,AcceptedCmp5,AcceptedCmp1, AcceptedCmp2, Complain, Z_CostContact, Z_Revenue, Response, Customer_hist
- Detected and eliminated outliers present in Income and Age column.









- Basic statistics of the newly transformed dataset were found.

```
In [216]: new_df.head()
Out[216]:
    Income  Recency  Age  Spent  Purchase
0  58138.0       58   65   1617        25
1  46344.0       38   68     27         6
2  71613.0       26   57    776        21
3  26646.0       26   38     53         8
4  58293.0       94   41    422        19
```

```
In [211]: new_df.describe()
Out[211]:
              Income      Recency          Age        Spent     Purchase
count    2205.000000  2205.000000  2205.000000  2205.000000  2205.000000
mean    51622.094785    49.009070    53.095692   606.821769    14.887982
std     20713.063826    28.932111    11.705801   601.675284     7.615277
min      1730.000000     0.000000    26.000000     5.000000     0.000000
25%     35196.000000    24.000000    45.000000    69.000000     8.000000
50%     51287.000000    49.000000    52.000000   397.000000    15.000000
75%     68281.000000    74.000000    63.000000  1047.000000    21.000000
max    113734.000000    99.000000    82.000000  2525.000000    43.000000
```

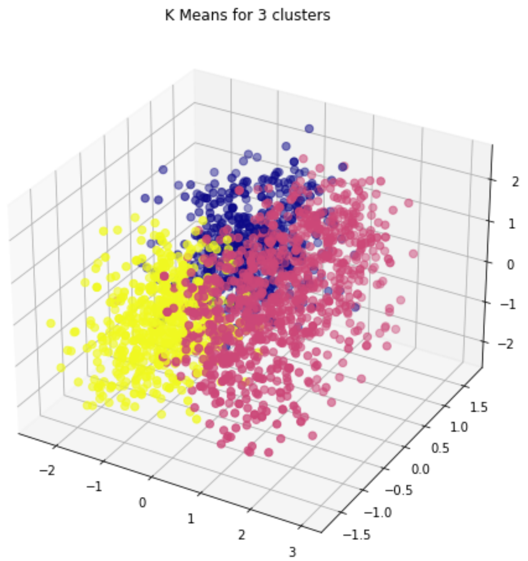
- Feature scaling of the dataset was done using standard scaler transformation to normalize the range of independent variables or features of data and newly formed dataset was named as 'scaled_df'.

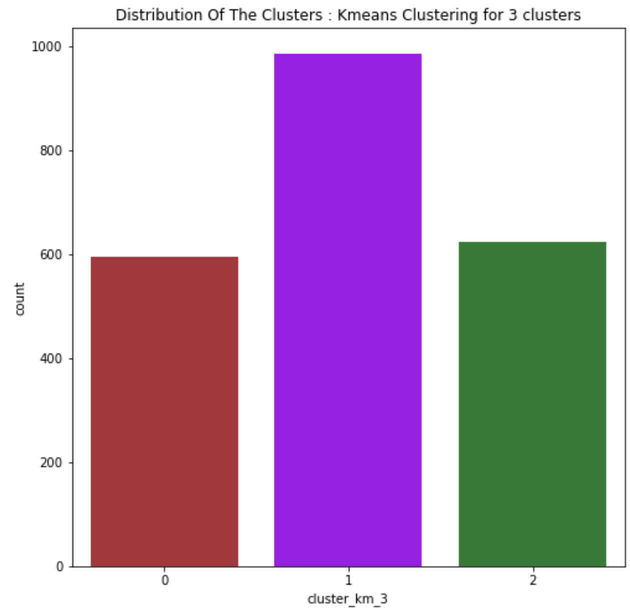
```
In [215]: scaled_df.head()
Out[215]:
     Income   Recency       Age     Spent  Purchase
0  0.314651  0.310830  1.017189  1.679323  1.328161
1 -0.254877 -0.380600  1.273530 -0.963897 -1.167390
2  0.965354 -0.795458  0.333612  0.281242  0.802782
3 -1.206087 -0.795458 -1.289883 -0.920675 -0.904700
4  0.322136  1.555404 -1.033542 -0.307248  0.540092
```

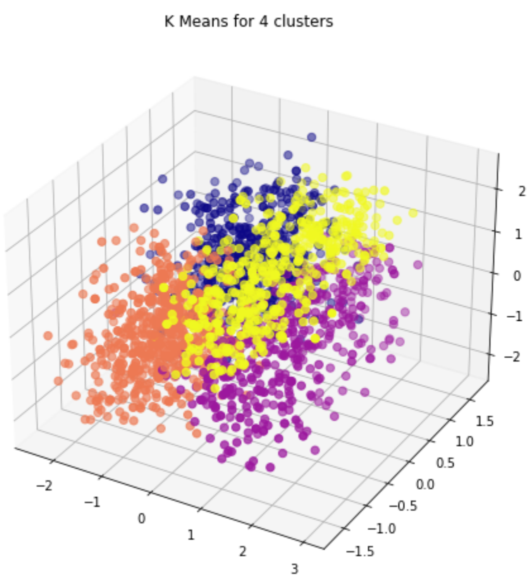
```
In [214]: scaled_df.describe()
Out[214]:
             Income       Recency           Age         Spent      Purchase
count  2.205000e+03  2.205000e+03  2.205000e+03  2.205000e+03  2.205000e+03
mean   2.255691e-17  7.975480e-17  1.643432e-16  9.667248e-18  2.255691e-17
std    1.000227e+00  1.000227e+00  1.000227e+00  1.000227e+00  1.000227e+00
min   -2.409272e+00 -1.694318e+00 -2.315248e+00 -1.000470e+00 -1.955459e+00
25%   -7.932106e-01 -8.646014e-01 -6.917534e-01 -8.940766e-01 -9.047005e-01
50%   -1.618161e-02 -3.135738e-04 -9.362368e-02 -3.488084e-01  1.471300e-02
75%    8.044529e-01  8.639742e-01  8.462945e-01  7.317536e-01  8.027817e-01
max    2.999363e+00  1.728262e+00  2.469790e+00  3.188785e+00  3.692367e+00
```

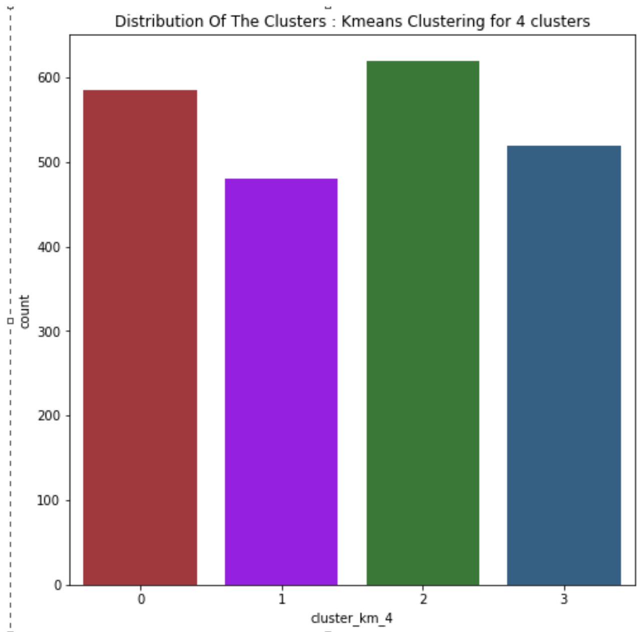
6. Clustering techniques performed on the data to identify similar clusters.(Question 3)
   - Clustering is basically a task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different.
   - Here, we have performed two techniques for clustering : KMeans Clustering and Agglomerative Clustering.
   - KMeans Clustering:

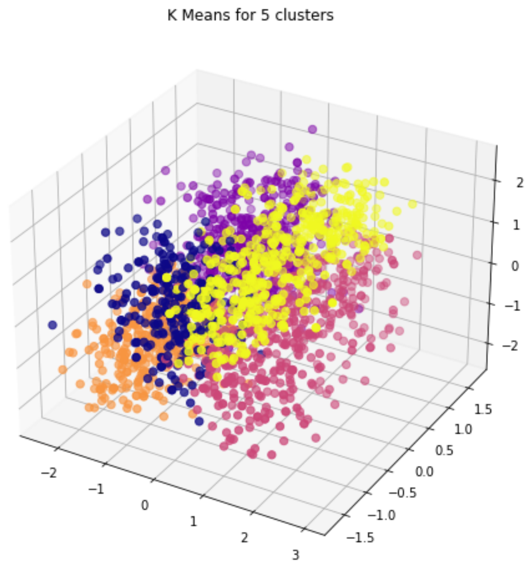
- KMeans algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group.
- We have defined a function to perform KMeans clustering and created cluster(scatter) plots and count plots for the same through it.
- We have then executed the function by entering required parameters and number of clusters.
- Below are the plots for the same.

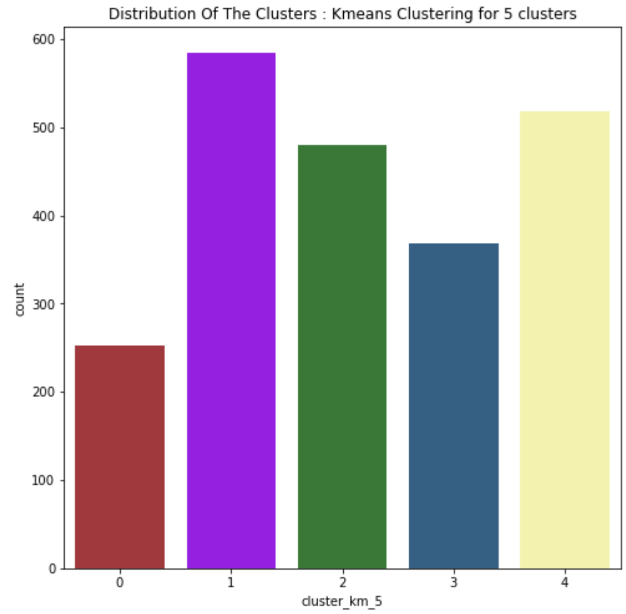




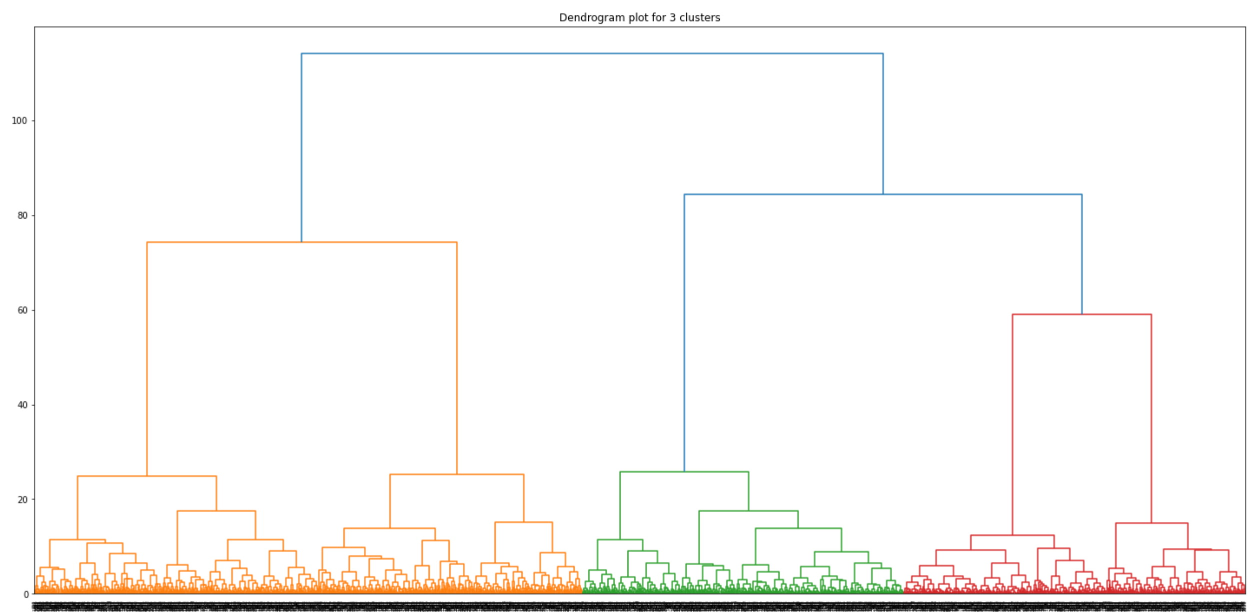


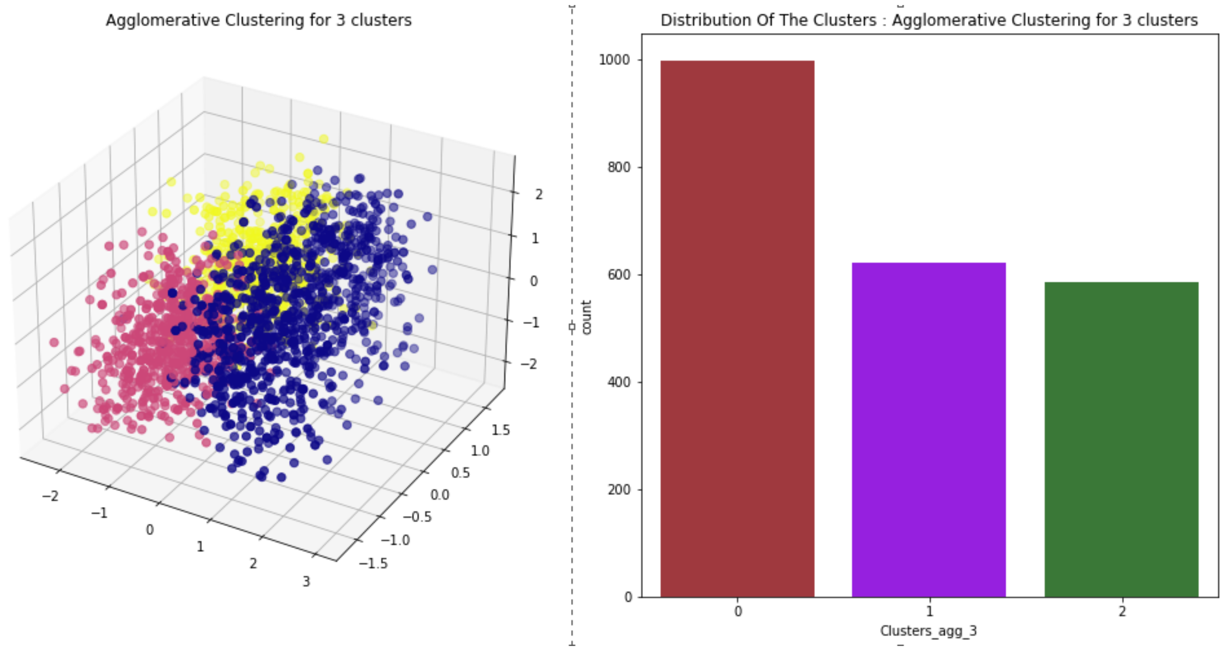


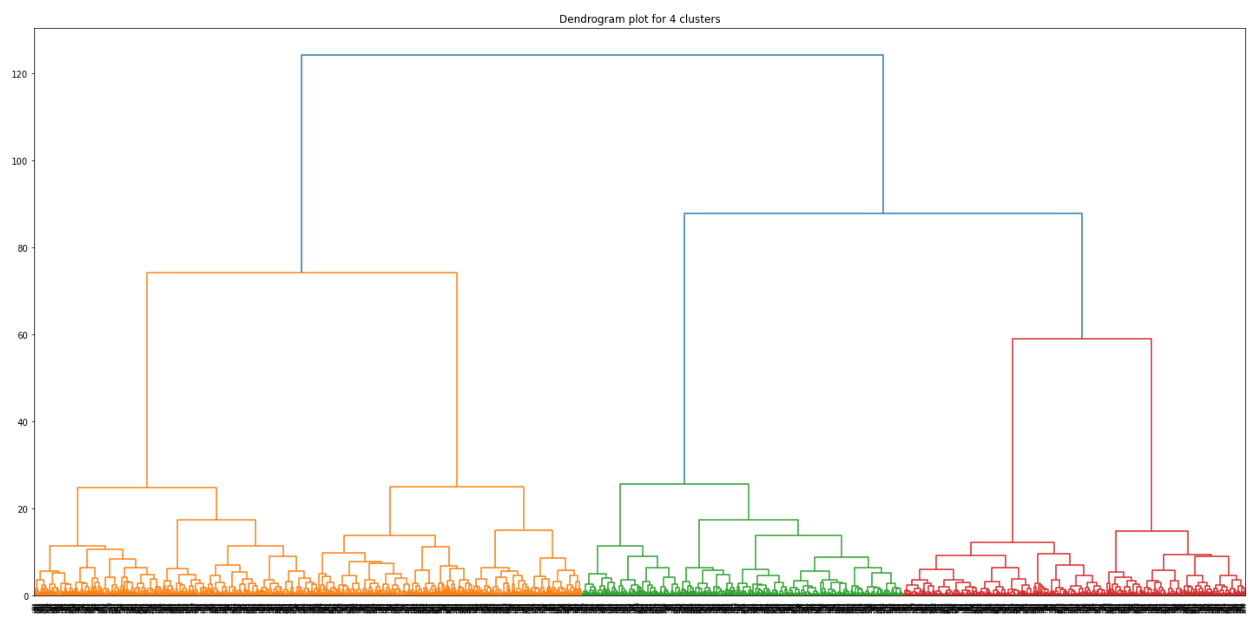


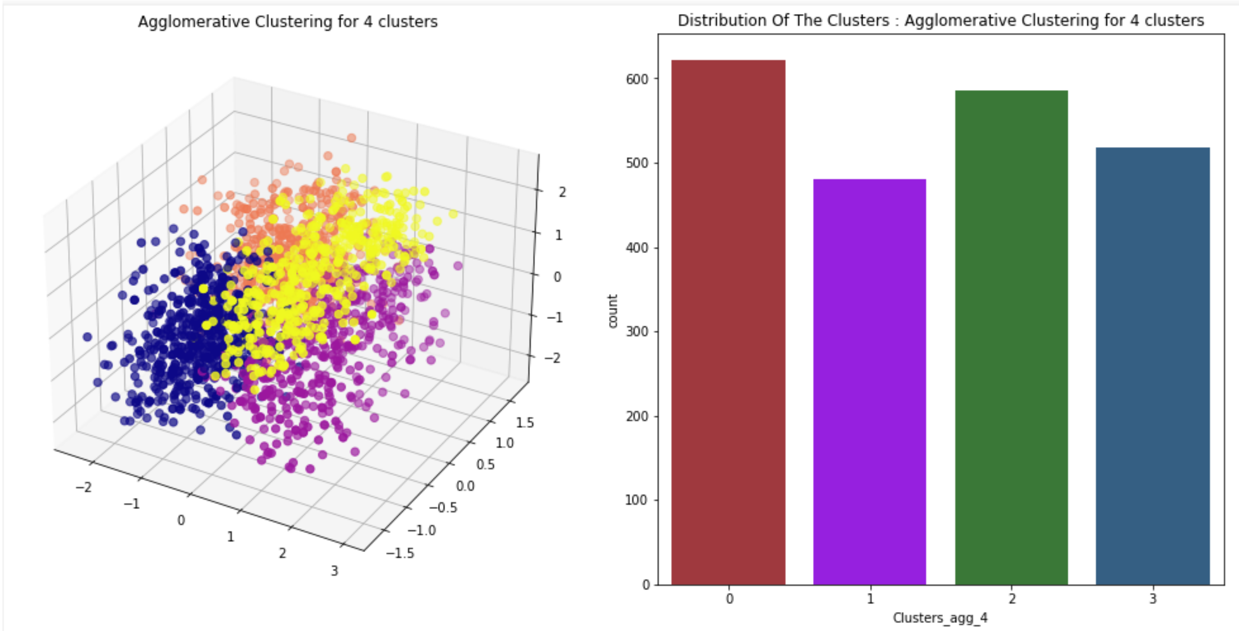


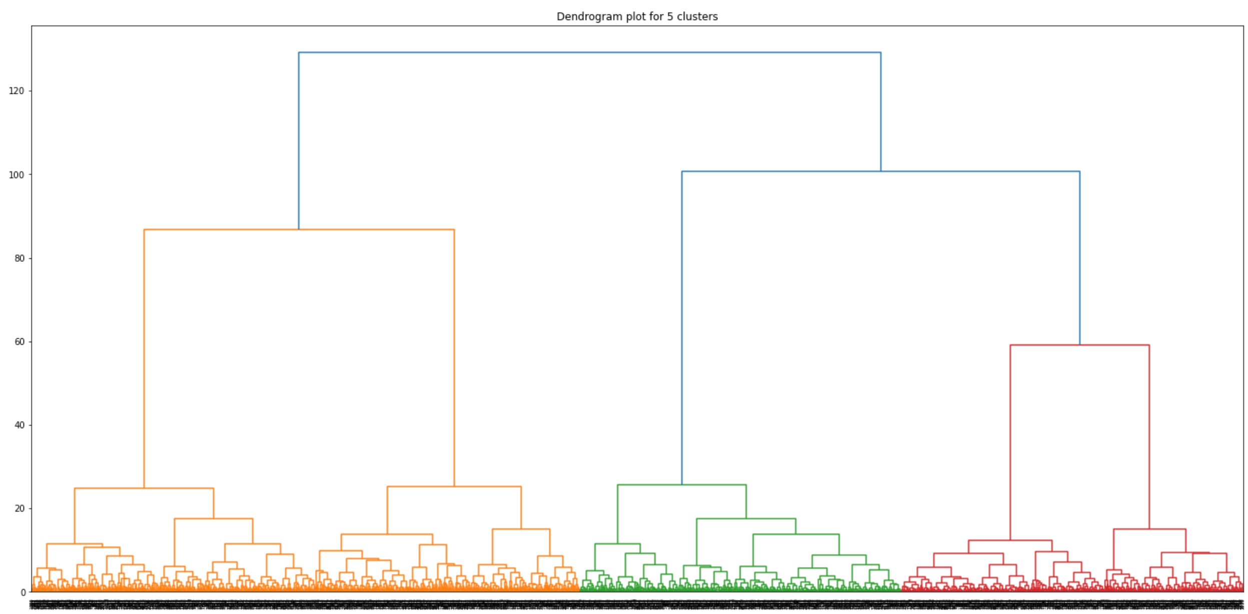
- Agglomerative Clustering:
  - Agglomerative Clustering is a specific type of hierarchical clustering which merges the closest pair of clusters based on the distance among centroids and repeats this step until only a single cluster is left.
  - We have defined a function to perform Agglomerative clustering and created cluster(scatter) plots and count plots for the same through it.
  - We have then executed the function by entering required parameters and number of clusters.
  - Below are the plots for the same.

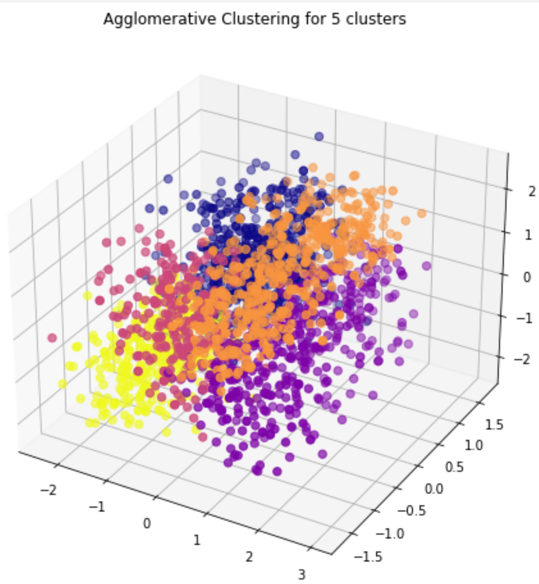


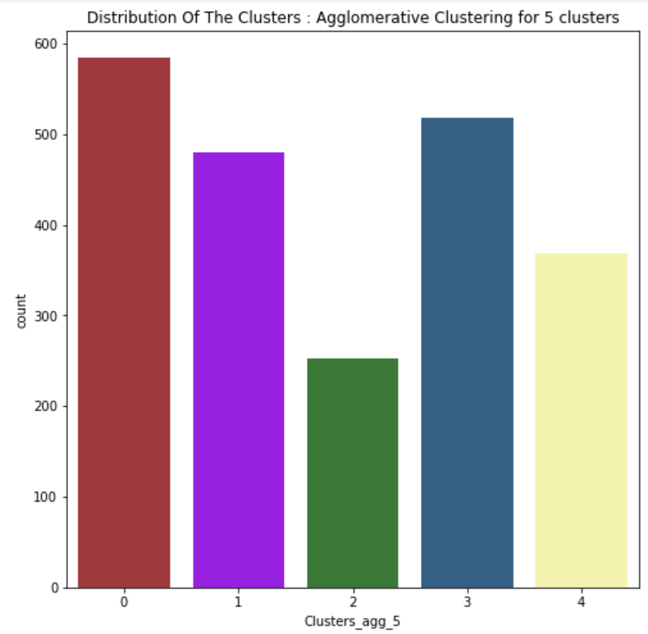

Agglomerative Clustering for 3 clusters
Distribution Of The Clusters : Agglomerative Clustering for 3 clusters
count
Clusters_agg_3


Dendrogram plot for 4 clusters

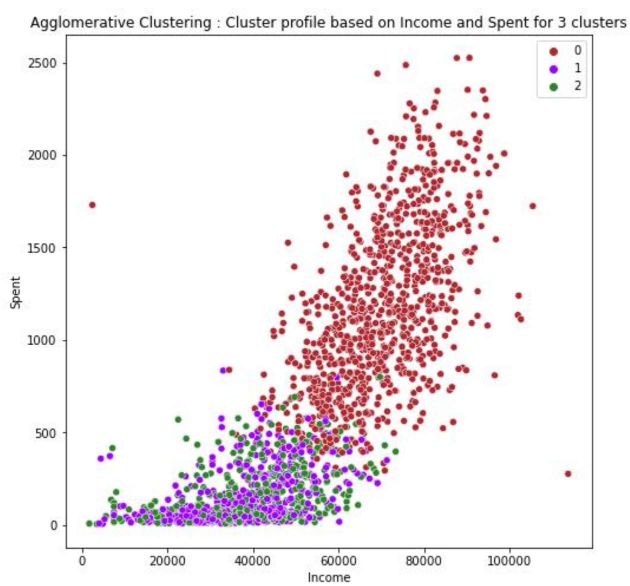

Agglomerative Clustering for 4 clusters
Distribution Of The Clusters : Agglomerative Clustering for 4 clusters
count
Clusters_agg_4
0
1
2
3
600
500
400
300
200
100


Dendrogram plot for 5 clusters
120
100
80
60
40
20
0





- We have identified below clusters.

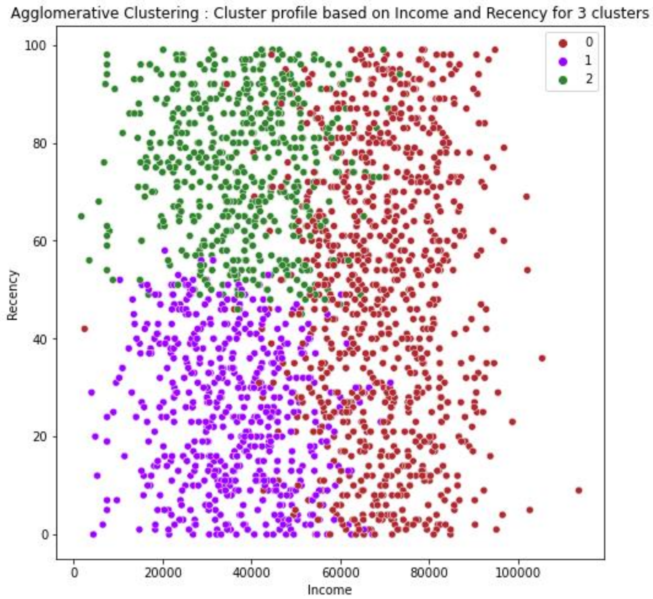


Spent Vs Income :

Group 0 : High Income, High Expenditure(Spent)

Group 1 : Low Income, Low Expenditure(Spent)

Group 2 : Average Income, Average Expenditure(Spent)

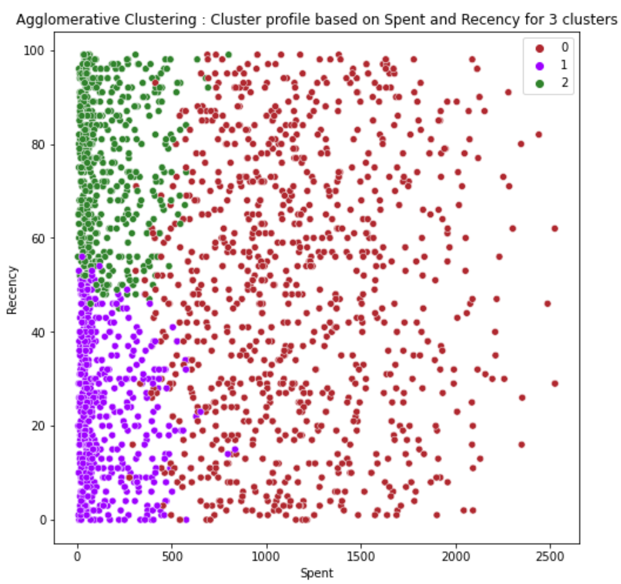


Recency Vs Income :

Group 0 : High Income, High Recency

Group 1 : Low Income, Low Recency

Group 2 : Low Income, High Recency



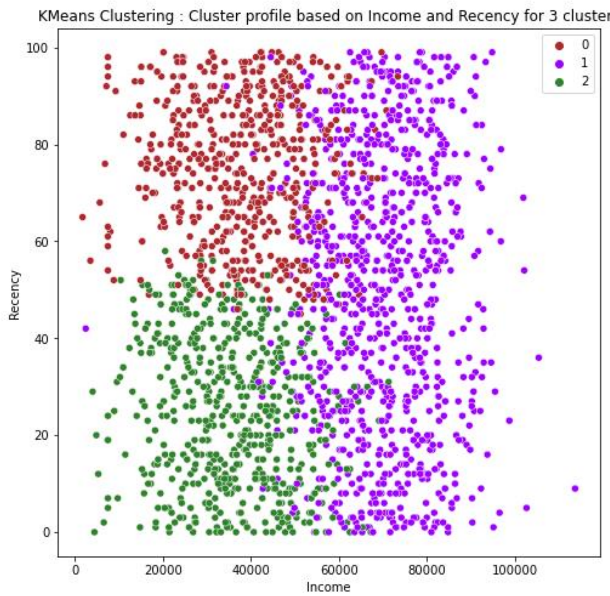
Recency Vs Age :

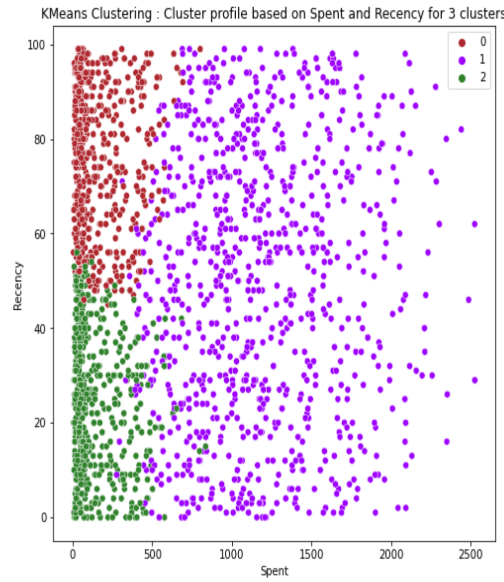
Group 0 : High Spent, High Recency

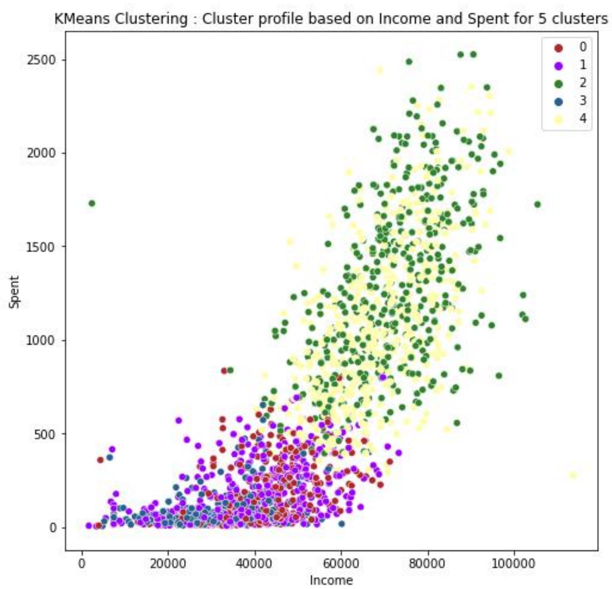
Group 1 : Low Spent, Low Recency

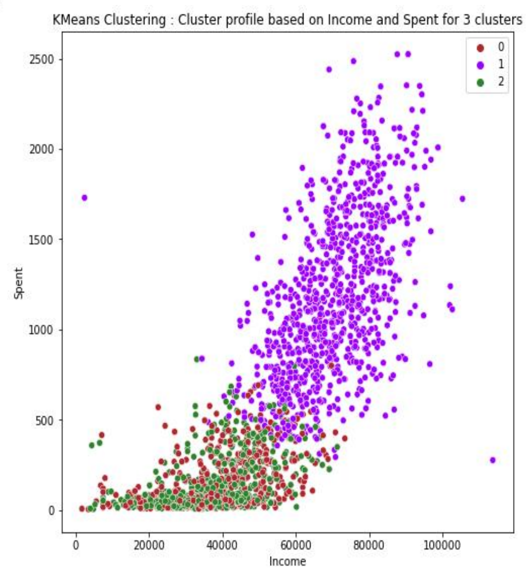
Group 2 : Low Spent, High Recency

- Other similar charts are as follows :





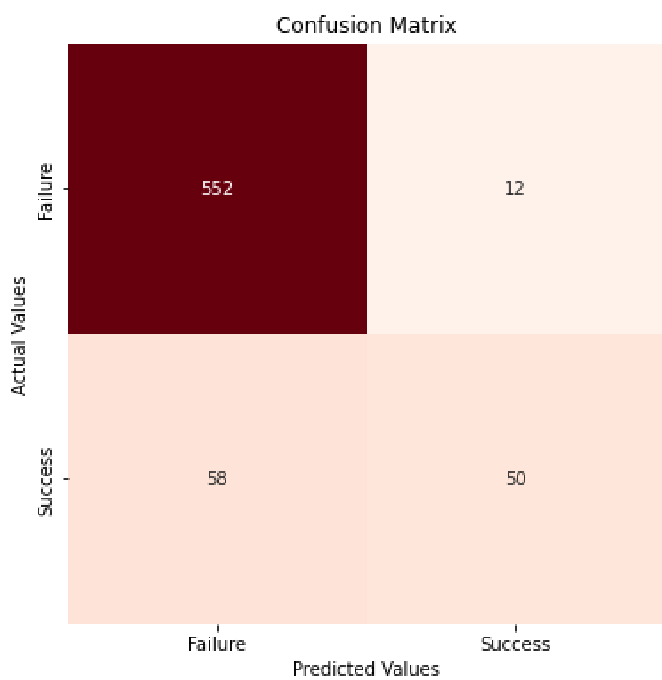




7. Binary Classification ML approach to identify what customers will respond to the next marketing campaign ("Response" attribute).(Question 4)
   - For performing Binary classification, we have first split the dataset df in to X and y (Response).
   - We have then used train_test_split function from sklearn.model_selection to get the train and test component of X and y.
   - Post that, we have performed scalar transformation on X_test and X_train.
   - We used Tensor flow to train the data and then found the model (model1) to be evaluated.
   - On evaluating X_test and y_test of the model, we found the test accuracy, test loss and test AUC.
   - Predict function was used for X_test to find the predicted value of y and corresponding confusion matrix and classification report were determined as below.

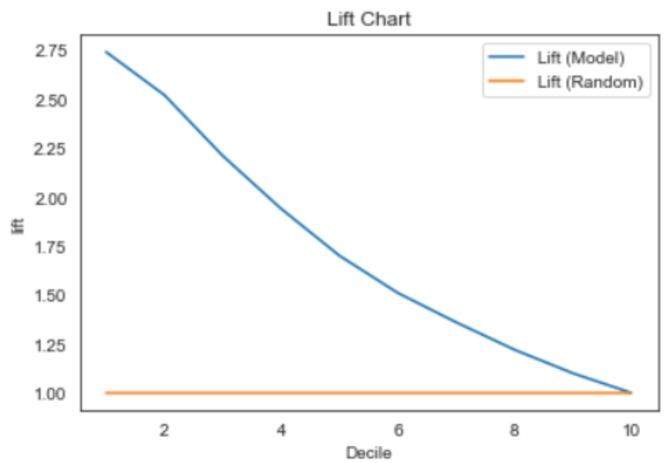
```
  return t[start:end]
Epoch 1/100
40/40 [==============================] - 1s 13ms/step - loss: 0.4172 - accuracy: 0.8349 - auc: 0.6463 - val_loss: 0.3503 -
val_accuracy: 0.8822 - val_auc: 0.8292
Epoch 2/100
40/40 [==============================] - 0s 2ms/step - loss: 0.2848 - accuracy: 0.8907 - auc: 0.8663 - val_loss: 0.2972 -
val_accuracy: 0.8917 - val_auc: 0.8916
Epoch 3/100
40/40 [==============================] - 0s 2ms/step - loss: 0.2468 - accuracy: 0.8987 - auc: 0.9066 - val_loss: 0.2904 -
val_accuracy: 0.9108 - val_auc: 0.8964
Epoch 4/100
40/40 [==============================] - 0s 2ms/step - loss: 0.2224 - accuracy: 0.9083 - auc: 0.9269 - val_loss: 0.2790 -
val_accuracy: 0.9076 - val_auc: 0.9081
Epoch 5/100
40/40 [==============================] - 0s 2ms/step - loss: 0.2052 - accuracy: 0.9139 - auc: 0.9384 - val_loss: 0.2835 -
val_accuracy: 0.8949 - val_auc: 0.9053
Epoch 6/100
40/40 [==============================] - 0s 2ms/step - loss: 0.1926 - accuracy: 0.9203 - auc: 0.9483 - val_loss: 0.2826 -
val_accuracy: 0.9076 - val_auc: 0.9115
Epoch 7/100
40/40 [==============================] - 0s 2ms/step - loss: 0.1741 - accuracy: 0.9290 - auc: 0.9573 - val_loss: 0.2874 -
val_accuracy: 0.9045 - val_auc: 0.9083
```

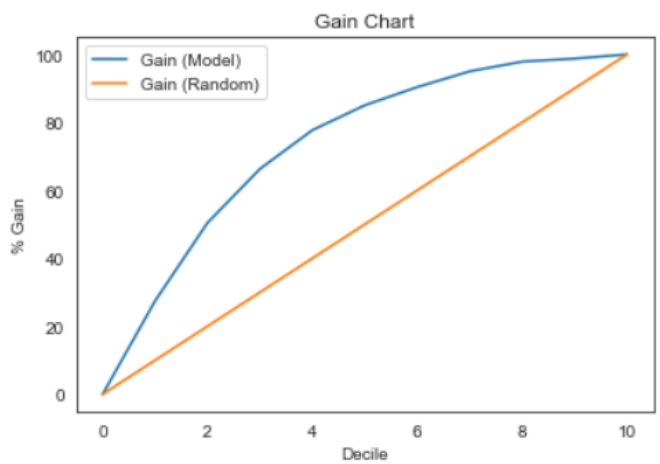
```
In [246]: print("    Test Loss : {:.5f}".format(result1[0]))
     ...: print("Test Accuracy : {:.2f}%".format(result1[1] * 100))
     ...: print("     Test AUC : {:.5f}".format(result1[2]))
    Test Loss : 0.25977
Test Accuracy : 89.58%
     Test AUC : 0.90638
```



```
In [249]: print("Report for Classification :\n \n", classrep)
Report for Classification :

              precision    recall  f1-score   support

     Failure       0.90      0.98      0.94       564
     Success       0.81      0.46      0.59       108

    accuracy                           0.90       672
   macro avg       0.86      0.72      0.76       672
weighted avg       0.89      0.90      0.88       672
```

8. Calculation of Lift metric for 10th percentile.(Question 5)
    - Lift and Gain analysis is an analysis to evaluate the model prediction and the benefit to the business. It is often used in the marketing target analysis but not restricted.
    - Gain and lift charts are visual aids for evaluating the performance of classification models.





Project Code:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 24 10:08:28 2022

@author: gauri
"""

import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
import scipy.cluster.hierarchy as sch
import tensorflow as tf
from sklearn.metrics import confusion_matrix, classification_report

########################## Functions ###############################

#Defining function for calculating Age
def age_in_years(joined_date):
    '''
    Function to calculate age in years

    Parameters
    ----------
    joined_date : Date

    Returns
    -------
    Customer Age in years

    '''
    today = date.today()
    return today.year - joined_date.year - ((today.month, today.day) < (joined_date.month,
                                        joined_date.day))

#Defining function for KMeans Clustering
def KMeans_Cluster(x,data,n_clusters):

    sdx= x.iloc[:,0]
    sdy= x.iloc[:,1]
    sdz= x.iloc[:,2]
```

```
    kmns= KMeans(n_clusters, init='k-means++',
            n_init=10, max_iter=100, random_state=0)
    ypredicted= kmns.fit_predict(x)
    #Adding the Clusters feature to the orignal dataframe.
    x["cluster_km_{}".format(n_clusters)] = kmns.labels_
    data["cluster_km_{}".format(n_clusters)] = kmns.labels_

    #Plotting the clusters
    fig = plt.figure(figsize=(10,8))
    ax = plt.subplot(111, projection='3d', label="bla")
    ax.scatter(sdx,sdy,sdz, s=40, c=kmns.labels_, marker='o' ,cmap = 'plasma' )
    ax.set_title("K Means for {} clusters".format(n_clusters))
    plt.show()

    #Plotting count plot for Clusters
    pal = ["#b0282f","#9E00FF", "#30832c","#286090","#fffea3"]
    pl = sns.countplot(x=x["cluster_km_{}".format(n_clusters)], palette= pal)
    pl.set_title("Distribution Of The Clusters : Kmeans Clustering for {} clusters".format(n_clusters))
    plt.show()

#Defining function for Agglomerative Clustering
def Agglomerative_Cluster(x,data,n_clusters):

    adx= x.iloc[:,0]
    ady= x.iloc[:,1]
    adz= x.iloc[:,2]

    plt.figure(figsize=(25,12))
    dendrogram=sch.dendrogram(sch.linkage(x,method = 'ward'))
    plt.title("Dendrogram plot for {} clusters".format(n_clusters))
    plt.show()

    agg = AgglomerativeClustering(n_clusters)
    y_pred_agg = agg.fit_predict(x)
    x["Clusters_agg_{}".format(n_clusters)] = y_pred_agg
    data["Clusters_agg_{}".format(n_clusters)] = y_pred_agg

    #Plotting the clusters
    fig = plt.figure(figsize=(10,8))
    ax = plt.subplot(111, projection='3d', label="bla")
    ax.scatter(adx,ady,adz, s=40, c= y_pred_agg, marker='o', cmap = 'plasma' )
    ax.set_title("Agglomerative Clustering for {} clusters".format(n_clusters))
    plt.show()

    #Plotting count plot for Clusters
    pal = ["#b0282f","#9E00FF", "#30832c","#286090","#fffea3"]
    pl = sns.countplot(x=x["Clusters_agg_{}".format(n_clusters)], palette= pal)
```

```
    pl.set_title("Distribution Of The Clusters : Agglomerative Clustering for {} clusters".format(n_clusters))
    plt.show()

#Defining function for plotting clusters wrt columns
def plotXYKmeans(x,n_clusters,x_axis,y_axis):
        pal = ["#b0282f","#9E00FF", "#30832c","#286090","#fffea3"]
        plt.rcParams['figure.figsize'] = [8,8]
        plt1 = sns.scatterplot(data = x,x=x[x_axis],
y=x[y_axis],hue=x["cluster_km_{}".format(n_clusters)],palette=pal)
        plt1.set_title("KMeans Clustering : Cluster profile based on {} and {} for {}
clusters".format(x_axis,y_axis,n_clusters))
        plt1.legend()
        plt1.show()

def plotXYAgg(x,n_clusters,x_axis,y_axis):
        pal = ["#b0282f","#9E00FF", "#30832c","#286090","#fffea3"]
        plt.rcParams['figure.figsize'] = [8,8]
        plt1 = sns.scatterplot(data = x,x=x[x_axis],
y=x[y_axis],hue=x["Clusters_agg_{}".format(n_clusters)],palette=pal)
        plt1.set_title("Agglomerative Clustering : Cluster profile based on {} and {} for {}
clusters".format(x_axis,y_axis,n_clusters))
        plt1.legend()
        plt1.show()

#Defining function to convert categorical variable into numeric
def onehot_encode(df, column):
    df = df.copy()
    dummies = pd.get_dummies(df[column], prefix = column)
    df = pd.concat([df, dummies], axis=1)
    df = df.drop(column, axis = 1)
    return df

#Defining function for pre-processing data
def preprocess_inputs(df):
    df = df.copy()

    # Drop ID column
    df = df.drop('ID', axis=1)

    # Fill missing Income values with column mean
    df['Income'] = df['Income'].fillna(df['Income'].mean())

    # Date encoding
    df['Dt_Customer'] = pd.to_datetime(df['Dt_Customer'])
    df['Year_Customer'] = df['Dt_Customer'].apply(lambda x: x.year)
    df['Month_Customer'] = df['Dt_Customer'].apply(lambda x: x.month)
    df['Day_Customer'] = df['Dt_Customer'].apply(lambda x: x.day)
    df = df.drop('Dt_Customer', axis=1)
```

```
    # One-hot encoding
    for column in ['Education', 'Marital_Status']:
        df = onehot_encode(df, column=column)

    # Split df into X and y
    y = df['Response']
    X = df.drop('Response', axis=1)

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, shuffle=True, random_state=1)

    scaler = StandardScaler()
    scaler.fit(X_train)
    X_train = pd.DataFrame(scaler.transform(X_train), index = X_train.index, columns = X_train.columns)
    X_test = pd.DataFrame(scaler.transform(X_test), index = X_test.index, columns = X_train.columns)

    return X_train, X_test, y_train, y_test


######################### Data Loading ############################
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
from datetime import date

df = pd.read_csv("Downloads/marketing_campaign.csv",sep=";")

# Cast variable to date format
df['Dt_Customer'] = pd.to_datetime(df["Dt_Customer"], infer_datetime_format=True)


df['Customer_hist'] = df['Dt_Customer'].apply(age_in_years)
df['Age'] = 2022 - df['Year_Birth']

df.head()

######################################### Qs 1 ##########################################
#Basic statistics of every column
df.describe()

######################################## Qs 2 #########################################

#Graphs for dataset attributes
# Number of sample according to MaritalStatus

plt.figure()
plt.bar(df["Marital_Status"].value_counts().index, df["Marital_Status"].value_counts(), color = "b")
plt.xlabel("Frequency")
```

```
plt.ylabel("Marital Status")
plt.title("Marital Status - Data Frequency")
plt.show()
print(f"Number of Samples According to Marital Status \n{df['Marital_Status'].value_counts()}")

# Number of samples according to Education

plt.figure()
plt.bar(df["Education"].value_counts().index, df["Education"].value_counts(), color = "b")
plt.xlabel("Frequency")
plt.ylabel("Education Status")
plt.title("Education - Data Frequency")
plt.show()
print(f"Number of Samples According to Education \n{df['Education'].value_counts()}")


# campaign accept rates
campaigns = ["AcceptedCmp1", "AcceptedCmp2", "AcceptedCmp3", "AcceptedCmp4",
"AcceptedCmp5",]
for i in campaigns:
   accept_rate = (df.groupby(i).size() / df[i].count()) * 100
   plt.title(f"Accept Rates For {i}")
   plt.pie(accept_rate, labels=df[i].unique(), autopct='%1.2f%%')
   plt.show()


#Total no. of campaign accepted by a customer vs Income
campaigns_cols = [col for col in df.columns if 'Cmp' in col]
df['TotalCampaignsAcc'] = df[campaigns_cols].sum(axis=1)
plt.figure(figsize=(8,8))
sns.swarmplot(x='TotalCampaignsAcc', y='Income', data=df)
plt.show()

#Education Vs income
plt.bar('Education','Income',data= df, color ='maroon',width = 0.4)
plt.xlabel("Education")
plt.ylabel("Income")
plt.title("Education vs Income")
plt.show()

#Response Rate according to education
pd.crosstab(index=df['Response'], columns=df['Education']).plot(kind="pie", figsize=(16, 8),
subplots=True, autopct='%1.2f%%', legend=False)
plt.show()

########################################### Qs 3
############################################
```

```
#Data Manipulation

#Copy this dataset and create new dataset

new_df = df.copy()
new_df.head()

#Drop duplicate rows and rows with NA values
new_df.dropna(inplace=True)
new_df.drop_duplicates(inplace=True)

#Adding new columns
new_df["Spent"] = df["MntWines"]+ df["MntFruits"]+ df["MntMeatProducts"]+ df["MntFishProducts"]+
df["MntSweetProducts"]+ df["MntGoldProds"]
new_df["Purchase"] = df["NumDealsPurchases"]+ df["NumCatalogPurchases"]+
df["NumStorePurchases"]+ df["NumWebPurchases"]


#Drop columns which are not required or might create semantic confusion in analysis
new_df.drop(["Z_CostContact", "Z_Revenue","ID",
"Education","Marital_Status","Dt_Customer","AcceptedCmp3", "AcceptedCmp4", "AcceptedCmp5",
"AcceptedCmp1","AcceptedCmp2", "Complain",
"Response","MntFruits","MntWines","MntMeatProducts","MntFishProducts","MntSweetProducts"
,"MntGoldProds" ], axis=1 , inplace=True) #drop columns due to same values for all datapoints

new_df.drop(["NumWebPurchases","NumDealsPurchases","NumStorePurchases","NumCatalogPurchas
es","NumWebVisitsMonth","Customer_hist","Year_Birth","Kidhome","Teenhome"], axis=1 ,
inplace=True)

#Eliminating outliers

#Income
sns.boxplot(new_df.Income, palette='crest')
plt.title("Income - With Outliers")

Q1 = new_df['Income'].quantile(0.25)
Q3 = new_df['Income'].quantile(0.75)
IQR = Q3 - Q1

lower_lim = Q1 - 1.5 * IQR
upper_lim = Q3 + 1.5 * IQR

outliers_low = (new_df['Income'] < lower_lim)
outliers_up = (new_df['Income'] > upper_lim)
len(new_df['Income'] - (len(new_df['Income'][outliers_low] + len(new_df['Income'][outliers_up]))))
new_df['Income'][(outliers_low | outliers_up)]
new_df['Income'][~(outliers_low | outliers_up)]
new_df = new_df[~(outliers_low | outliers_up)]
```

```
sns.boxplot(new_df.Income, palette='crest')
plt.title("Income - After eliminating Outliers")

#Age
sns.boxplot(new_df.Age, palette='crest')
plt.title("Age - With Outliers")

Q1_1 = new_df['Age'].quantile(0.25)
Q3_1 = new_df['Age'].quantile(0.75)
IQR_1 = Q3_1 - Q1_1

lower_lim_1 = Q1_1 - 1.5 * IQR_1
upper_lim_1 = Q3_1 + 1.5 * IQR_1

outliers_low_1 = (new_df['Age'] < lower_lim_1)
outliers_up_1 = (new_df['Age'] > upper_lim_1)
len(new_df['Age'] - (len(new_df['Age'][outliers_low_1] + len(new_df['Age'][outliers_up_1]))))
new_df['Age'][(outliers_low_1 | outliers_up_1)]
new_df['Age'][~(outliers_low_1 | outliers_up_1)]
new_df = new_df[~(outliers_low_1 | outliers_up_1)]

sns.boxplot(new_df.Age, palette='crest')
plt.title("Age - After eliminating Outliers")

#Basic statistics of the data
new_df.describe()
new_df.head()

#Feature scaling
sc = StandardScaler()
sc.fit(new_df)
scaled_df = pd.DataFrame(sc.transform(new_df), columns = new_df.columns )

scaled_df.head()

scaled_df.describe()

#Clustering

#Kmeans Clustering

#KMeans Clustering with cluster 3,4,5
#KMeans_Cluster(Dataset,number_of_Clusters)

KMeans_Cluster(scaled_df,new_df,3)
KMeans_Cluster(scaled_df,new_df,4)
KMeans_Cluster(scaled_df,new_df,5)
```

```
#Hierarchial Clustering : Agglomerative Clustering

#Agglomerative Clustering with cluster 3,4,5
#Agglomerative_Cluster(Dataset,number_of_Clusters)

Agglomerative_Cluster(scaled_df,new_df,3)
Agglomerative_Cluster(scaled_df,new_df,4)
Agglomerative_Cluster(scaled_df,new_df,5)

#Interpretation

#Spent Vs Income
plotXYKmeans(new_df,3,'Income','Spent')
plotXYKmeans(new_df,4,'Income','Spent')
plotXYKmeans(new_df,5,'Income','Spent')


plotXYAgg(new_df,3,'Income','Spent')
plotXYAgg(new_df,4,'Income','Spent')
plotXYAgg(new_df,5,'Income','Spent')


#Recency Vs Income
plotXYKmeans(new_df,4,'Income','Recency')
plotXYAgg(new_df,4,'Income','Recency')

#Recency Vs Age
plotXYKmeans(new_df,5,'Spent','Recency')
plotXYAgg(new_df,5,'Spent','Recency')


##################################### Qs 4 ##########################################

#Binary Classification ML Approach

#Pre-processing

X_train, X_test, y_train, y_test = preprocess_inputs(df)

#Training

ip = tf.keras.Input(shape = (X_train.shape[1], ))
x = tf.keras.layers.Dense(128, activation = 'relu')(ip)
x = tf.keras.layers.Dense(128, activation = 'relu')(x)
op = tf.keras.layers.Dense(1, activation = 'sigmoid')(x)
```

```
model1 = tf.keras.Model(inputs=ip, outputs=op)

model1.compile(optimizer = 'adam',
        loss = 'binary_crossentropy',
        metrics = ['accuracy',
              tf.keras.metrics.AUC(name='auc')])

history = model1.fit(X_train,
          y_train,
          validation_split = 0.2,
          epochs  = 100,
          callbacks = [
            tf.keras.callbacks.EarlyStopping(
            monitor = 'val_loss',
            patience = 3,
            restore_best_weights = True)
          ])


result1 = model1.evaluate(X_test, y_test, verbose = 0)

print("    Test Loss : {:.5f}".format(result1[0]))
print("Test Accuracy : {:.2f}%".format(result1[1] * 100))
print("     Test AUC : {:.5f}".format(result1[2]))


y_prediction = np.array(model1.predict(X_test) >= 0.5, dtype=np.int)

confmat = confusion_matrix(y_test, y_prediction)
classrep = classification_report(y_test, y_prediction, target_names = ['Failure', 'Success'])


plt.figure(figsize = (6,6))
sns.heatmap(confmat, annot = True, fmt = 'g', vmin = 0, cmap = 'Reds', cbar = False)
plt.xticks(ticks = np.arange(2) + 0.5, labels=["Failure", "Success"])
plt.yticks(ticks = np.arange(2) + 0.5, labels=["Failure", "Success"])
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title("Confusion Matrix")
plt.show()


print("Report for Classification :\n \n", classrep)


########################################### Qs 5 ##################################
```

```
#Getting the prediction probability of class 1 and order it by descending order
X_test['Probability']  = model1.predict_proba(X_test)[:,1]
X_test = X_test.sort_values(by = 'Probability', ascending = False)
X_test['Response'] = y_test

#Divide the data into decile
X_test['decile'] = pd.qcut(X_test['Probability'], 10, labels=[i for i in range (10, 0, -1)])

#Calculate the actual response in each decile
response1 = pd.crosstab(X_test['decile'], X_test['Response'])[1].reset_index().rename(columns = {1: 'No of Responses'})
liftgain = X_test['decile'].value_counts(sort = False).reset_index().rename(columns = {'decile': 'No of Cases', 'index': 'decile'})
liftgain = pd.merge(liftgain, response1, on = 'decile').sort_values(by = 'decile', ascending = False).reset_index(drop = True)

#Calculate the cumulative
liftgain['Cumulative Response'] = liftgain['No of Responses'].cumsum()
#Calculate the percentage of positive in each decile
liftgain['Percentage of Events'] = np.round(((liftgain['No of Responses']/liftgain['No of Responses'].sum())*100),2)
#Calculate the Gain in each decile
liftgain['gain'] = liftgain['Percentage of events'].cumsum()

liftgain['decile'] = liftgain['decile'].astype('int')
liftgain['lift'] = np.round((liftgain['gain']/(liftgain['decile']*10)),2)
```