Data Mining:

Business Report

By Utkarsh Sharma

Problem Statement 1:

A leading bank wants to develop a customer segmentation to give promotional offers to its customers. They collected a sample that summarizes the activities of users during the past few months. You are given the task to identify the segments based on credit card usage.

1.1 Read the data and do exploratory data analysis. Describe the data briefly.

1.2 Do you think scaling is necessary for clustering in this case? Justify

1.3 Apply hierarchical clustering to scaled data. Identify the number of optimum clusters using Dendrogram and briefly describe them

1.4 Apply K-Means clustering on scaled data and determine optimum clusters. Apply elbow curve and silhouette score.

1.5 Describe cluster profiles for the clusters defined. Recommend different promotional strategies for different clusters.

Solution:

1.1 Read the data and do exploratory data analysis. Describe the data briefly.

Data Analysis:

We are given a sample of customer data of a leading bank, let us look at some sample records and then proceed with analysing this data:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
0	19.94	16.92	0.8752	6.675	3.763	3.252	6.550
1	15.99	14.89	0.9064	5.363	3.582	3.336	5.144
2	18.95	16.42	0.8829	6.248	3.755	3.368	6.148
3	10.83	12.96	0.8099	5.278	2.641	5.182	5.185
4	17.99	15.86	0.8992	5.890	3.694	2.068	5.837

The dataset has 210 records and 7 columns:

```
#   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   spending                      210 non-null    float64
 1   advance_payments              210 non-null    float64
 2   probability_of_full_payment   210 non-null    float64
 3   current_balance               210 non-null    float64
 4   credit_limit                  210 non-null    float64
 5   min_payment_amt               210 non-null    float64
 6   max_spent_in_single_shopping  210 non-null    float64
dtypes: float64(7)
```

Here we noticed that all the columns are of 'float' datatype and there are no 'NULL' values in the data as shown below:

```
spending                      0
advance_payments              0
probability_of_full_payment   0
current_balance               0
credit_limit                  0
min_payment_amt               0
max_spent_in_single_shopping  0
```

Exploratory Data Analysis:

Now getting the insights in to the data:

Univariate Analysis:

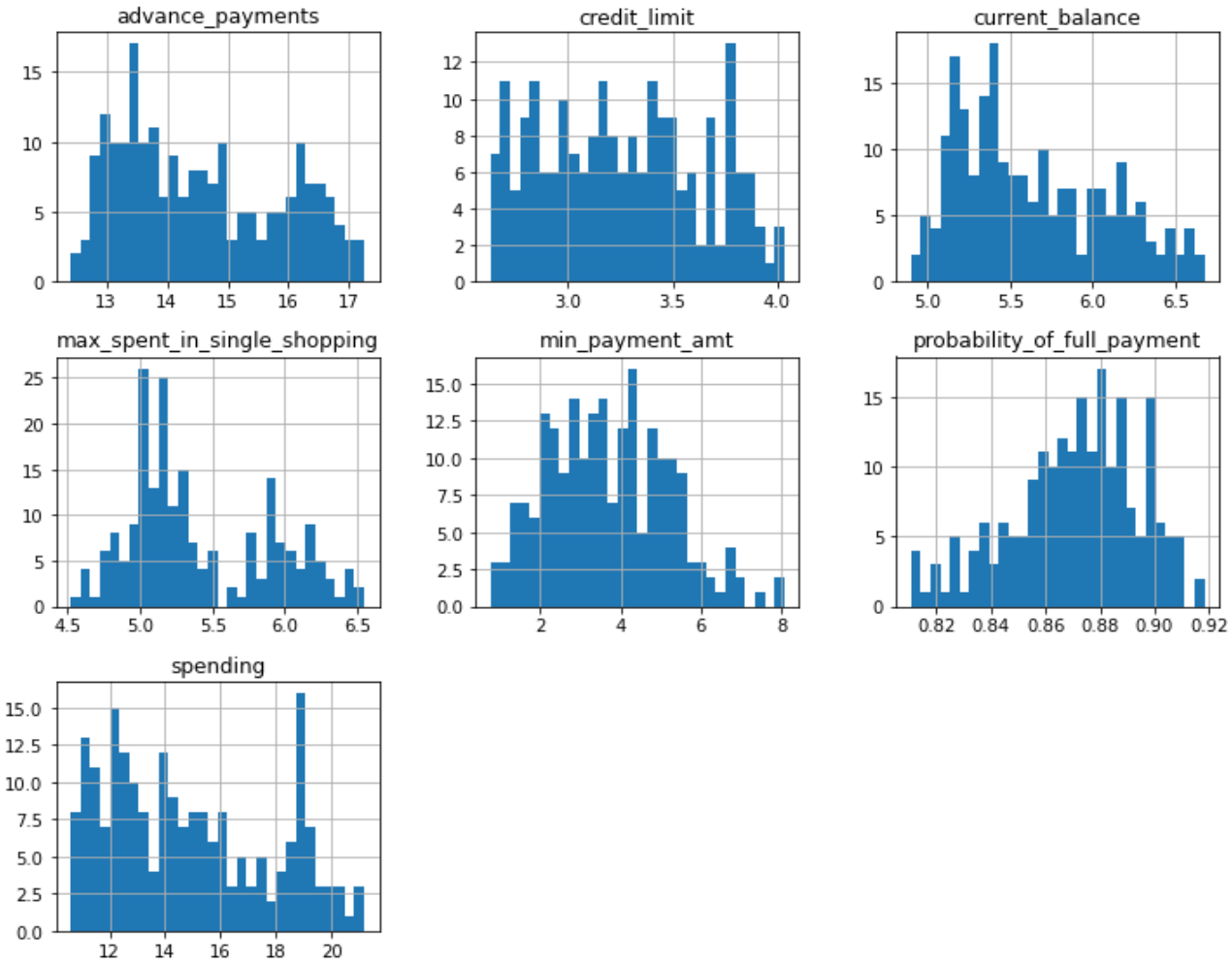
Once we load data we are good to go for the first type of EDA called as univariate analysis. “Uni” means one and “Variate” means variable hence univariate analysis means analysis of one variable or one feature. Univariate basically tells us how data in each feature is distributed and also tells us about central tendencies like mean, median, and mode.

On performing some statistical analysis on the data to get more insights:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
count	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000
mean	14.847524	14.559286	0.870999	5.628533	3.258605	3.700201	5.408071
std	2.909699	1.305959	0.023629	0.443063	0.377714	1.503557	0.491480
min	10.590000	12.410000	0.808100	4.899000	2.630000	0.765100	4.519000
25%	12.270000	13.450000	0.856900	5.262250	2.944000	2.561500	5.045000
50%	14.355000	14.320000	0.873450	5.523500	3.237000	3.599000	5.223000
75%	17.305000	15.715000	0.887775	5.979750	3.561750	4.768750	5.877000
max	21.180000	17.250000	0.918300	6.675000	4.033000	8.456000	6.550000

Seeing this data we can say that this data is pretty much normal, and by observing the minimum, maximum and median values of every column, not much skewness can be find in the data. Also, we can observe that the average probability of customer making full payment is 87.1% and average max spent in a single shopping is 5.41K. We can see that the average credit limit of customers is 32.59K and average spending is 14.85K.

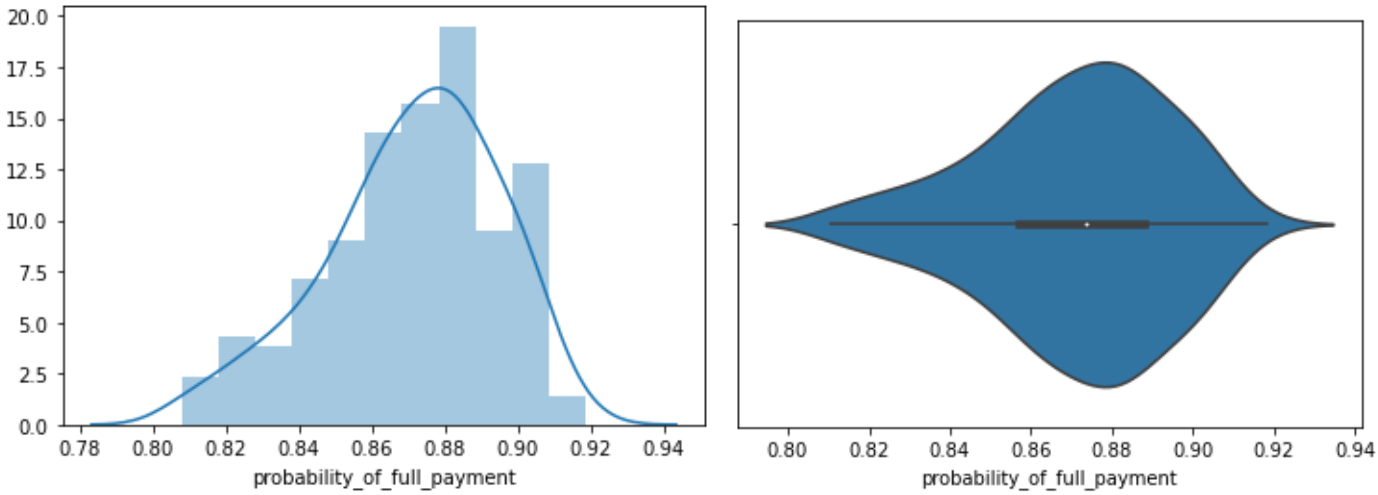
To get further details about the data, lets plot the histograms for each filed:



Upon seeing this histogram plots, some positive skewness can be seen for 'current_balance' column and data for 'probability_of_full_payment' looks a bit negatively skewed, below is the table to see the skewness for every filed.

```
spending                      0.399889
advance_payments              0.386573
probability_of_full_payment  -0.537954
current_balance               0.525482
credit_limit                  0.134378
min_payment_amt               0.401667
max_spent_in_single_shopping  0.561897
dtype: float64
```

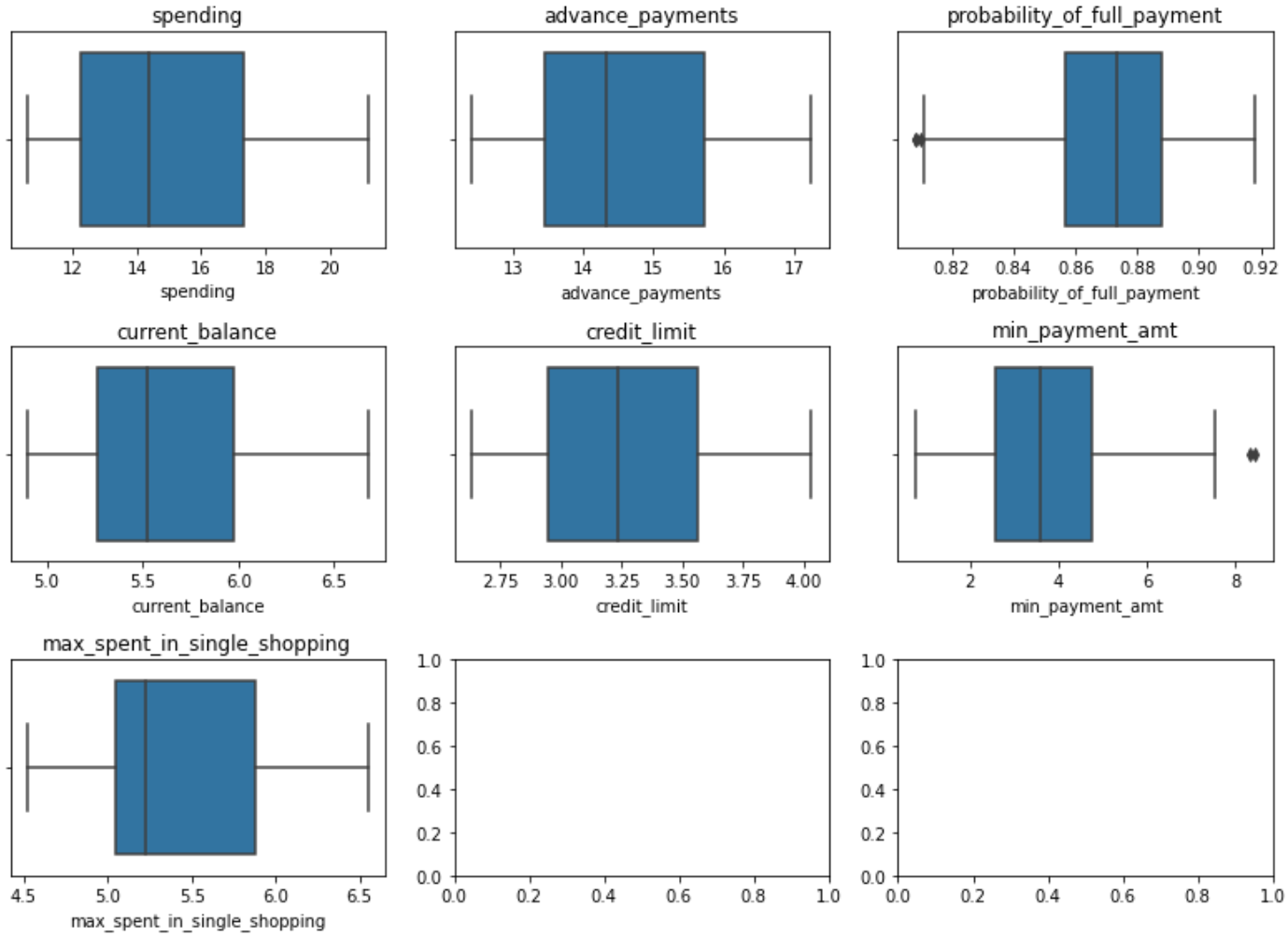
Getting into the details of 'probability_of_full_payment' field:



The negatively skewed character of this field suggests that the probability of people making full payment is on the higher side.

Note: For the details of other columns, please refer python notebook.

Checking for the outliers:



We can see that there are couple of outliers for variables 'probability_of_full_payment' and 'min_payment_amount'. Machine learning algorithms are sensitive to the range and distribution of attribute values, especially when it comes to unsupervised learning technique we are more focused on calculating the distance between different datapoints, thus it becomes important to treat these outliers.

Here we are using IQR method to treat these outliers: The interquartile range (IQR), also called the mid-spread or middle 50%, or technically H-spread, is a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, i.e. between upper and lower quartiles.

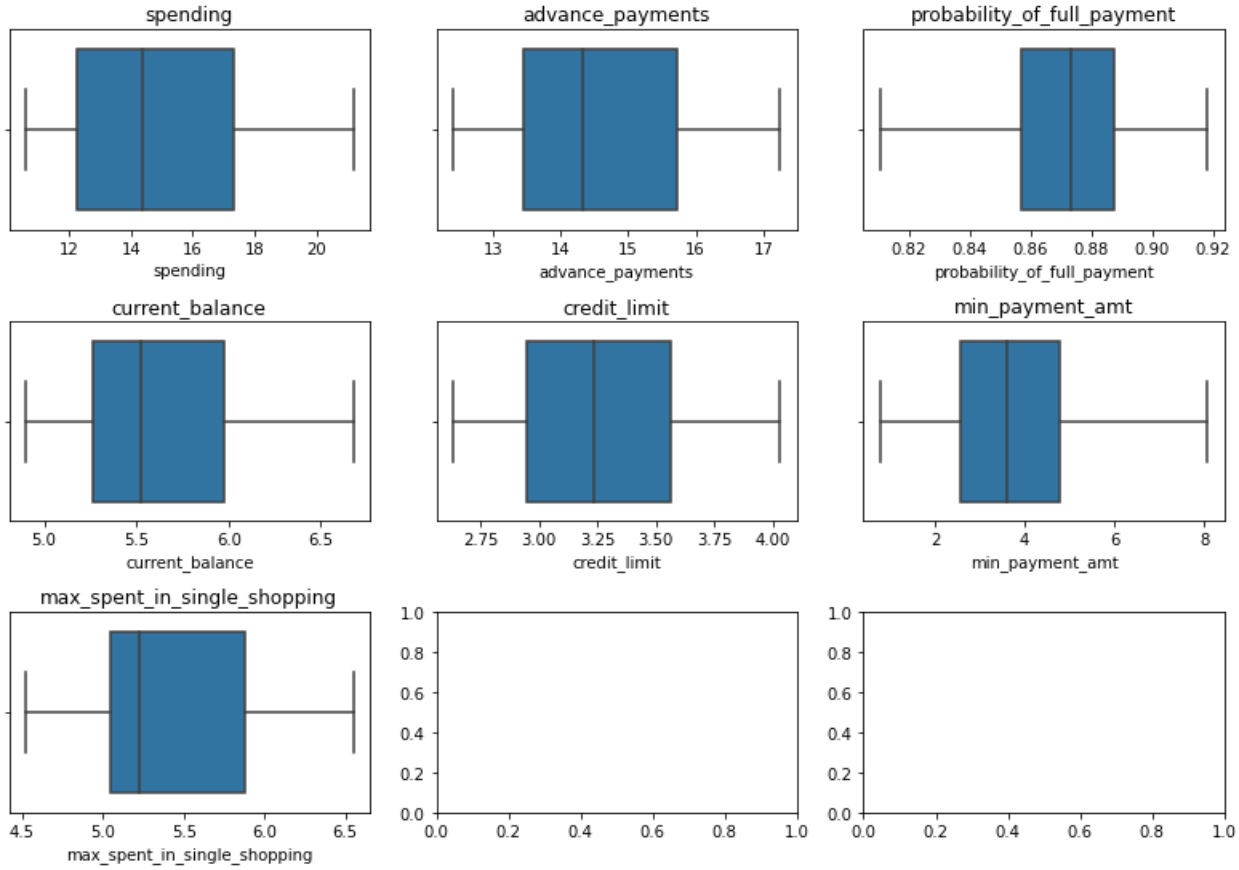
$$IQR = Q3 - Q1.$$

We define the upper and the lower limit as below and if any data point has value greater than the upper limit we will replace it with the upper range value itself and if any data point has value less than the lower limit then we will replace it with the lower range value itself.

lower_range= Q1-(1.5 * IQR)

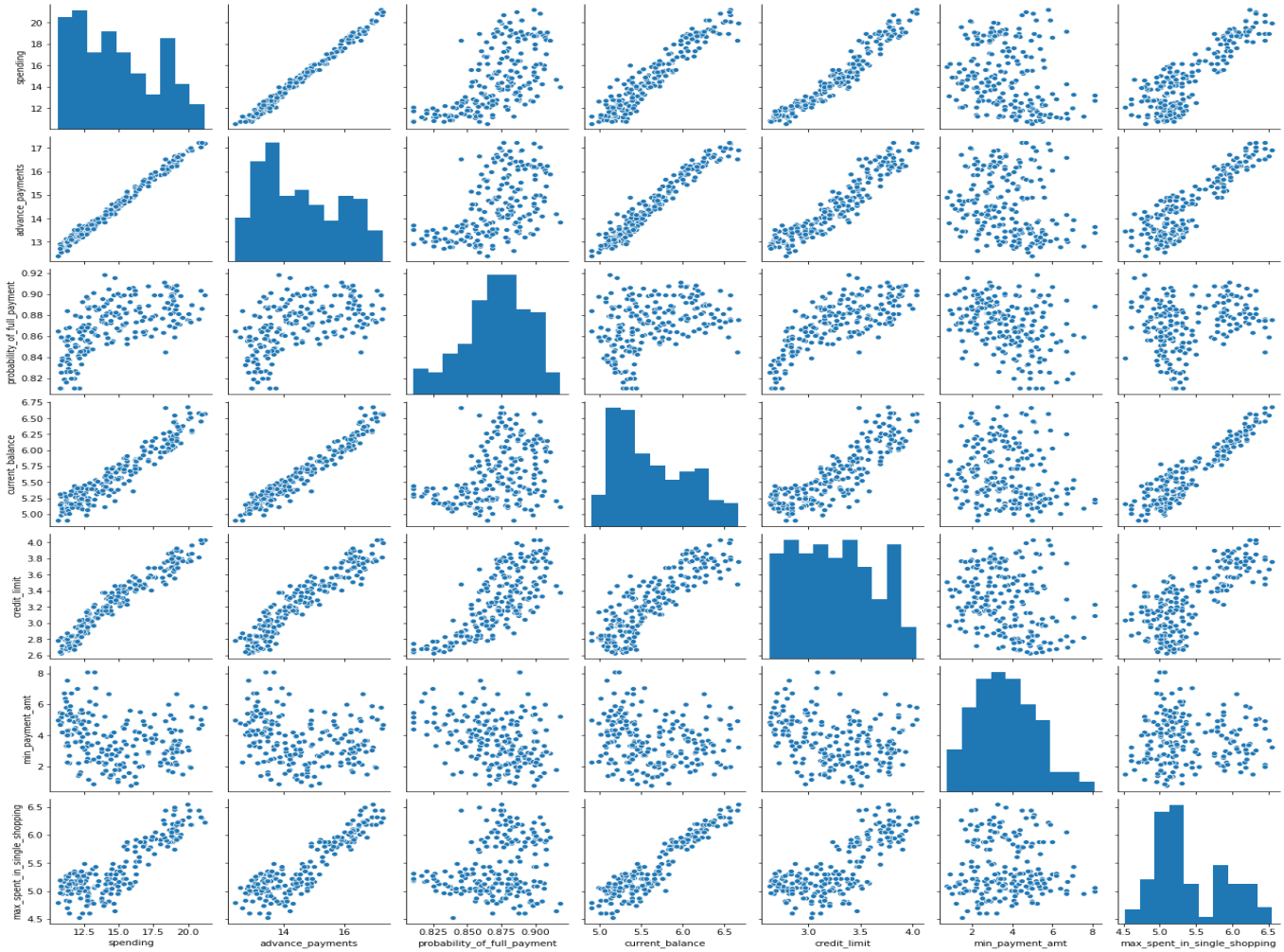
upper_range= Q3+(1.5 * IQR)

We can see that the outliers are now removed from the variables. Below are the boxplots of every variable after treating the outliers:



Multivariate Analysis:

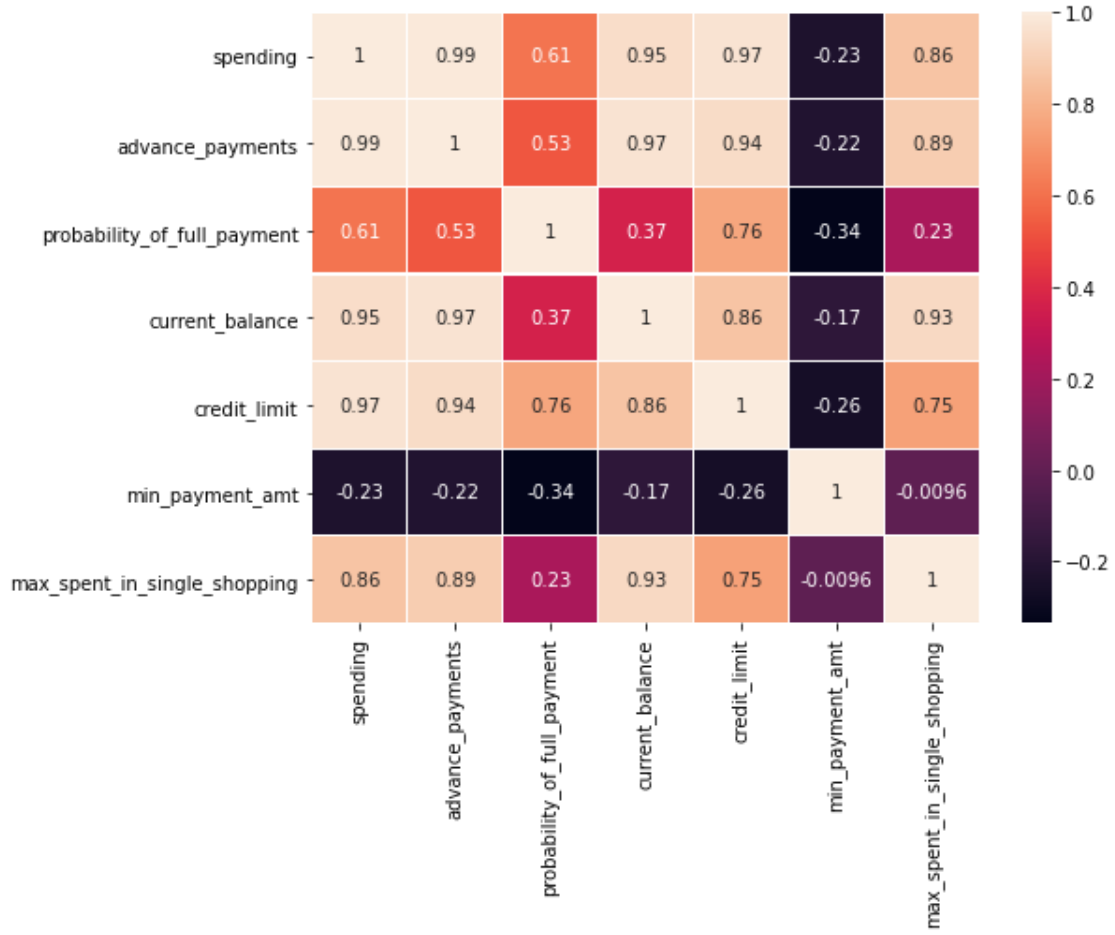
Multivariate analysis help us to study the relationship among multiple variables. Let look into the details by creating a scatter plot and see if there is any relation among the variables:



Here we can observe a strong positive correlation between many variables like ‘advance_payments’ and ‘spending’, ‘current_balance’ and ‘advance_payments’, ‘credit_limit’ and ‘spending’ etc. Also, there is very less correlation between ‘min_payment_amt’ and ‘credit_limit’.
Let’s get into the details and check the magnitude of this correlation and then plot the heat-map for the variables:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
spending	1	0.994341	0.6089	0.949985	0.970771	-0.229619	0.863693
advance_payments	0.994341	1	0.529925	0.972422	0.944829	-0.217051	0.890784
probability_of_full_payment	0.6089	0.529925	1	0.368419	0.762218	-0.335071	0.22714
current_balance	0.949985	0.972422	0.368419	1	0.860415	-0.170701	0.932806
credit_limit	0.970771	0.944829	0.762218	0.860415	1	-0.25898	0.749131
min_payment_amt	-0.229619	-0.217051	-0.335071	-0.170701	-0.25898	1	-0.009605
max_spent_in_single_shopping	0.863693	0.890784	0.22714	0.932806	0.749131	-0.009605	1

Heat-map:



The correlation between the variables ‘advance_payment’ & ‘spending’ and ‘spending’ & ‘credit_limit’ is 0.99 and 0.97 respectively which is very strong and positive. On the other hand, there can be seen some negative correlation between ‘min_payment_amt’ and ‘probability_of_full_payment’ (-0.34), however we can say that there is very small or no correlation between the variables ‘min_payment_amt’ and ‘max_spent_in_single_shopping’ (-0.0096).

1.2 Do you think scaling is necessary for clustering in this case? Justify

Yes! There may be the case where dataset contains features that highly vary in magnitudes, units, and range. Normalisation should be performed when the scale of a feature is irrelevant or misleading. ML algorithm works better when features are relatively on a similar scale and close to Normal Distribution.

From the statistical description of the data set (shown above) we can see that the values in column 'spending' ranges from 10.59 to 21.18 whereas for 'probability_of_full_payment' have values from 0.808 to 0.918, which shows a huge difference in the magnitude of the values the variables can have.

When we work with distance based models (Agglomerative clustering or K-means), scaling or normalization is a requirement.

When doing clustering, we calculate the distance between the data points and while doing so we don't want one variable to have more impact than the other(s) thus, to avoid overpowering of any variable in distance calculation, scaling is necessary.

1.3 Apply hierarchical clustering to scaled data. Identify the number of optimum clusters using Dendrogram and briefly describe them

To scale the data we will use Standard Scaler in this case:

Standard Scaler standardizes a feature by subtracting the mean and then scaling to unit variance. Unit variance means dividing all the values by the standard deviation. Standard Scaler normalizes the data using the formula **(x-mean)/standard deviation.**

After scaling, below is our scaled dataset:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
0	1.754355	1.811968	0.177628	2.367533	1.338579	-0.298625	2.328998
1	0.393582	0.253840	1.505071	-0.600744	0.858236	-0.242292	-0.538582
2	1.413300	1.428192	0.505234	1.401485	1.317348	-0.220832	1.509107
3	-1.384034	-1.227533	-2.571391	-0.793049	-1.639017	0.995699	-0.454961
4	1.082581	0.998364	1.198738	0.591544	1.155464	-1.092656	0.874813

Please refer python note book for the steps involved in scaling.

For Hierarchical clustering, we will be using '**hierarchy'** from **scipy.cluster** library and for diatance calculation we will be using '**ward's'** method.

The concept of Agglomerative clustering: This is a hierarchical clustering which follows- bottom-up approach i.e. it starts with each object forming a separate group and keeps on merging the objects that are closer to one another.

Ward's Linkage: Also known as minimum variance clustering method. It is an iterative method where after every merge, the distances are updated successively. Ward's method often creates compact and even-sized clusters.

Below is the distance matrix obtained using agglomerative clustering and Ward's linkage method:

```
array([[1.90000000e+01, 2.30000000e+01, 1.93561325e-01, 2.00000000e+00],
       [3.00000000e+00, 4.40000000e+01, 2.10880708e-01, 2.00000000e+00],
       [7.00000000e+00, 3.00000000e+01, 2.11416263e-01, 2.00000000e+00],
       [9.50000000e+01, 1.26000000e+02, 2.19575820e-01, 2.00000000e+00],
       [7.00000000e+01, 1.16000000e+02, 2.41240086e-01, 2.00000000e+00],
       [1.48000000e+02, 2.07000000e+02, 2.59139938e-01, 2.00000000e+00],
       [1.27000000e+02, 1.57000000e+02, 2.71769302e-01, 2.00000000e+00],
       [7.60000000e+01, 1.31000000e+02, 2.87275484e-01, 2.00000000e+00],
       [6.70000000e+01, 1.72000000e+02, 2.91364436e-01, 2.00000000e+00],
       [4.00000000e+00, 2.20000000e+01, 2.91569402e-01, 2.00000000e+00],
       [7.10000000e+01, 1.51000000e+02, 2.97134664e-01, 2.00000000e+00],
       [1.85000000e+02, 2.06000000e+02, 3.19058136e-01, 2.00000000e+00],
       [9.90000000e+01, 1.59000000e+02, 3.34862969e-01, 2.00000000e+00],
       [9.00000000e+00, 1.37000000e+02, 3.37994916e-01, 2.00000000e+00],
       [7.50000000e+01, 1.96000000e+02, 3.45544469e-01, 2.00000000e+00],
       [5.60000000e+01, 1.97000000e+02, 3.55807106e-01, 2.00000000e+00],
       [8.80000000e+01, 9.70000000e+01, 3.59654615e-01, 2.00000000e+00],
       [6.40000000e+01, 1.80000000e+02, 3.64523798e-01, 2.00000000e+00],
       [1.11000000e+02, 1.76000000e+02, 3.64843360e-01, 2.00000000e+00],
       [5.20000000e+01, 1.09000000e+02, 3.67595472e-01, 2.00000000e+00],
       [4.90000000e+01, 1.73000000e+02, 3.70274426e-01, 2.00000000e+00],
       [1.10000000e+01, 8.50000000e+01, 3.71467423e-01, 2.00000000e+00],
       [2.00000000e+00, 6.80000000e+01, 3.74171243e-01, 2.00000000e+00],
       [6.20000000e+01, 8.40000000e+01, 3.77607236e-01, 2.00000000e+00],
```

[3.50000000e+01, 1.71000000e+02, 4.03661273e-01, 2.00000000e+00],
[3.70000000e+01, 2.25000000e+02, 4.10157885e-01, 3.00000000e+00],
[3.90000000e+01, 1.01000000e+02, 4.12485772e-01, 2.00000000e+00],
[4.20000000e+01, 1.15000000e+02, 4.21020451e-01, 2.00000000e+00],
[1.68000000e+02, 1.88000000e+02, 4.23041645e-01, 2.00000000e+00],
[1.93000000e+02, 2.22000000e+02, 4.23628537e-01, 3.00000000e+00],
[6.00000000e+01, 1.28000000e+02, 4.25302852e-01, 2.00000000e+00],
[1.10000000e+02, 2.09000000e+02, 4.27995857e-01, 2.00000000e+00],
[9.80000000e+01, 1.47000000e+02, 4.32080673e-01, 2.00000000e+00],
[5.80000000e+01, 2.16000000e+02, 4.41905278e-01, 3.00000000e+00],
[6.50000000e+01, 1.75000000e+02, 4.44084333e-01, 2.00000000e+00],
[3.20000000e+01, 1.45000000e+02, 4.53607736e-01, 2.00000000e+00],
[9.20000000e+01, 1.67000000e+02, 4.54918691e-01, 2.00000000e+00],
[7.70000000e+01, 1.89000000e+02, 4.59442178e-01, 2.00000000e+00],
[4.00000000e+01, 1.64000000e+02, 4.64071400e-01, 2.00000000e+00],
[5.70000000e+01, 1.49000000e+02, 4.68077527e-01, 2.00000000e+00],
[1.61000000e+02, 2.20000000e+02, 4.71872878e-01, 3.00000000e+00],
[1.58000000e+02, 1.94000000e+02, 4.75685420e-01, 2.00000000e+00],
[2.70000000e+01, 1.14000000e+02, 4.81115717e-01, 2.00000000e+00],
[5.40000000e+01, 2.32000000e+02, 4.82966246e-01, 3.00000000e+00],
[5.00000000e+00, 8.90000000e+01, 4.85178021e-01, 2.00000000e+00],
[1.82000000e+02, 2.38000000e+02, 4.92196547e-01, 3.00000000e+00],
[7.40000000e+01, 1.13000000e+02, 4.96391842e-01, 2.00000000e+00],
[1.52000000e+02, 2.02000000e+02, 4.99393082e-01, 2.00000000e+00],
[1.60000000e+01, 3.30000000e+01, 5.16185972e-01, 2.00000000e+00],
[3.40000000e+01, 1.21000000e+02, 5.16886631e-01, 2.00000000e+00],
[8.20000000e+01, 1.83000000e+02, 5.17972600e-01, 2.00000000e+00],
[2.00000000e+01, 3.10000000e+01, 5.27347021e-01, 2.00000000e+00],
[1.20000000e+01, 2.10000000e+02, 5.32506373e-01, 3.00000000e+00],
[7.80000000e+01, 1.63000000e+02, 5.38635152e-01, 2.00000000e+00],
[9.30000000e+01, 1.42000000e+02, 5.43501371e-01, 2.00000000e+00],
[4.10000000e+01, 7.90000000e+01, 5.54427768e-01, 2.00000000e+00],
[1.29000000e+02, 2.40000000e+02, 5.56661616e-01, 3.00000000e+00],
[7.30000000e+01, 1.12000000e+02, 5.57851921e-01, 2.00000000e+00],
[2.80000000e+01, 1.30000000e+02, 5.61301940e-01, 2.00000000e+00],
[1.22000000e+02, 2.35000000e+02, 5.64623262e-01, 4.00000000e+00],
[7.20000000e+01, 1.74000000e+02, 5.64859577e-01, 2.00000000e+00],
[1.98000000e+02, 2.36000000e+02, 5.70368447e-01, 3.00000000e+00],
[6.90000000e+01, 9.40000000e+01, 5.70573619e-01, 2.00000000e+00],
[3.80000000e+01, 2.47000000e+02, 5.71580779e-01, 3.00000000e+00],
[1.91000000e+02, 2.28000000e+02, 5.74221417e-01, 3.00000000e+00],
[1.50000000e+01, 1.08000000e+02, 5.78047334e-01, 2.00000000e+00],
[1.03000000e+02, 2.24000000e+02, 5.82699616e-01, 3.00000000e+00],
[5.90000000e+01, 1.35000000e+02, 5.90406324e-01, 2.00000000e+00],
[5.10000000e+01, 8.30000000e+01, 6.06687454e-01, 2.00000000e+00],
[5.00000000e+01, 2.62000000e+02, 6.07738937e-01, 4.00000000e+00],
[1.24000000e+02, 2.60000000e+02, 6.13437151e-01, 3.00000000e+00],
[1.80000000e+01, 1.81000000e+02, 6.17754970e-01, 2.00000000e+00],
[1.18000000e+02, 2.03000000e+02, 6.22480416e-01, 2.00000000e+00],
[1.04000000e+02, 2.58000000e+02, 6.35283014e-01, 3.00000000e+00],
[9.10000000e+01, 1.60000000e+02, 6.56772401e-01, 2.00000000e+00],
[1.70000000e+01, 1.65000000e+02, 6.57754557e-01, 2.00000000e+00],
[1.34000000e+02, 2.49000000e+02, 6.59112609e-01, 3.00000000e+00],
[2.40000000e+01, 1.02000000e+02, 6.70800967e-01, 2.00000000e+00],
[2.50000000e+01, 1.95000000e+02, 6.71728550e-01, 2.00000000e+00],
[4.70000000e+01, 2.48000000e+02, 6.76753291e-01, 3.00000000e+00],
[9.60000000e+01, 2.70000000e+02, 6.79963312e-01, 3.00000000e+00],
[2.17000000e+02, 2.46000000e+02, 6.81938204e-01, 4.00000000e+00],
[1.00000000e+01, 1.62000000e+02, 6.97793573e-01, 2.00000000e+00],
[6.60000000e+01, 1.19000000e+02, 7.04145534e-01, 2.00000000e+00],
[6.10000000e+01, 2.85000000e+02, 7.04622461e-01, 3.00000000e+00],
[1.44000000e+02, 1.56000000e+02, 7.09647002e-01, 2.00000000e+00],
[6.00000000e+00, 2.45000000e+02, 7.14743093e-01, 3.00000000e+00],
[4.30000000e+01, 2.71000000e+02, 7.15606085e-01, 4.00000000e+00],

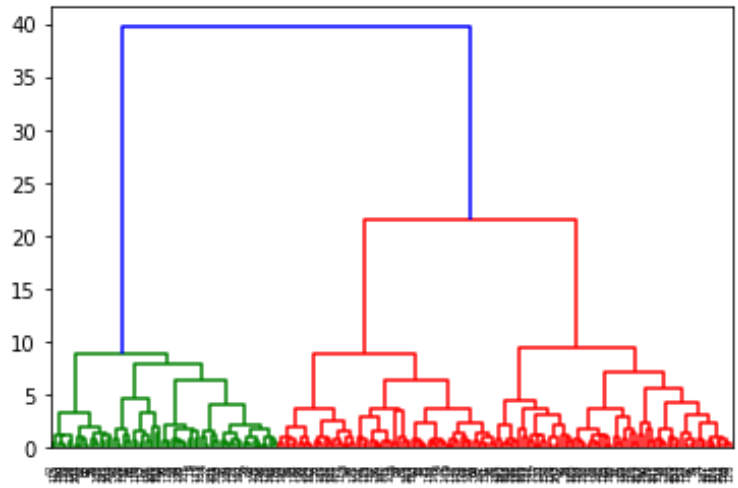
[3.60000000e+01, 2.82000000e+02, 7.15798971e-01, 3.00000000e+00],
[4.50000000e+01, 1.43000000e+02, 7.17589982e-01, 2.00000000e+00],
[1.39000000e+02, 2.05000000e+02, 7.21609790e-01, 2.00000000e+00],
[8.10000000e+01, 1.32000000e+02, 7.24357083e-01, 2.00000000e+00],
[2.60000000e+01, 1.20000000e+02, 7.26095461e-01, 2.00000000e+00],
[1.00000000e+02, 2.75000000e+02, 7.37288009e-01, 3.00000000e+00],
[1.78000000e+02, 2.08000000e+02, 7.37327455e-01, 2.00000000e+00],
[4.80000000e+01, 2.29000000e+02, 7.57309988e-01, 3.00000000e+00],
[2.18000000e+02, 2.65000000e+02, 7.59119652e-01, 4.00000000e+00],
[1.23000000e+02, 2.79000000e+02, 7.66110329e-01, 5.00000000e+00],
[1.66000000e+02, 2.04000000e+02, 7.85008170e-01, 2.00000000e+00],
[1.54000000e+02, 1.90000000e+02, 8.16674515e-01, 2.00000000e+00],
[2.33000000e+02, 2.68000000e+02, 8.19883067e-01, 4.00000000e+00],
[8.70000000e+01, 2.26000000e+02, 8.21222498e-01, 3.00000000e+00],
[1.46000000e+02, 2.44000000e+02, 8.23169354e-01, 3.00000000e+00],
[2.10000000e+01, 2.56000000e+02, 8.31659561e-01, 3.00000000e+00],
[2.23000000e+02, 2.99000000e+02, 8.43221484e-01, 4.00000000e+00],
[8.00000000e+00, 1.30000000e+01, 8.45004874e-01, 2.00000000e+00],
[1.99000000e+02, 2.52000000e+02, 8.57831776e-01, 3.00000000e+00],
[1.53000000e+02, 1.87000000e+02, 8.59641466e-01, 2.00000000e+00],
[2.30000000e+02, 2.92000000e+02, 8.60896384e-01, 4.00000000e+00],
[5.50000000e+01, 2.63000000e+02, 8.70703397e-01, 3.00000000e+00],
[2.57000000e+02, 2.66000000e+02, 8.72661995e-01, 5.00000000e+00],
[1.00000000e+00, 2.87000000e+02, 8.73695188e-01, 3.00000000e+00],
[1.92000000e+02, 2.59000000e+02, 8.84097349e-01, 3.00000000e+00],
[1.50000000e+02, 2.80000000e+02, 8.86133961e-01, 4.00000000e+00],
[2.00000000e+02, 2.51000000e+02, 9.05829857e-01, 3.00000000e+00],
[2.86000000e+02, 2.96000000e+02, 9.13432840e-01, 6.00000000e+00],
[2.55000000e+02, 3.01000000e+02, 9.23096790e-01, 5.00000000e+00],
[5.30000000e+01, 2.97000000e+02, 9.26739401e-01, 5.00000000e+00],
[0.00000000e+00, 1.33000000e+02, 9.29711393e-01, 2.00000000e+00],
[1.41000000e+02, 2.83000000e+02, 9.34660962e-01, 4.00000000e+00],
[1.79000000e+02, 2.94000000e+02, 9.50920852e-01, 4.00000000e+00],
[2.64000000e+02, 3.10000000e+02, 9.60451767e-01, 6.00000000e+00],
[1.69000000e+02, 1.77000000e+02, 9.60789558e-01, 2.00000000e+00],
[1.25000000e+02, 2.27000000e+02, 9.83600152e-01, 3.00000000e+00],
[2.53000000e+02, 2.76000000e+02, 9.88674756e-01, 6.00000000e+00],
[1.84000000e+02, 2.19000000e+02, 1.00808757e+00, 3.00000000e+00],
[2.42000000e+02, 2.43000000e+02, 1.02382306e+00, 5.00000000e+00],
[1.40000000e+01, 3.02000000e+02, 1.02800319e+00, 3.00000000e+00],
[2.11000000e+02, 2.34000000e+02, 1.04154707e+00, 4.00000000e+00],
[2.50000000e+02, 2.61000000e+02, 1.04385754e+00, 5.00000000e+00],
[6.30000000e+01, 2.31000000e+02, 1.06250145e+00, 3.00000000e+00],
[1.70000000e+02, 3.24000000e+02, 1.06514217e+00, 4.00000000e+00],
[2.14000000e+02, 2.15000000e+02, 1.08489012e+00, 4.00000000e+00],
[2.73000000e+02, 3.22000000e+02, 1.10024176e+00, 6.00000000e+00],
[2.21000000e+02, 3.11000000e+02, 1.10446422e+00, 5.00000000e+00],
[2.13000000e+02, 3.12000000e+02, 1.10855900e+00, 5.00000000e+00],
[1.07000000e+02, 3.13000000e+02, 1.12039625e+00, 4.00000000e+00],
[2.74000000e+02, 3.20000000e+02, 1.13047231e+00, 8.00000000e+00],
[2.90000000e+01, 2.98000000e+02, 1.13322610e+00, 4.00000000e+00],
[2.88000000e+02, 3.08000000e+02, 1.13874313e+00, 4.00000000e+00],
[1.38000000e+02, 2.78000000e+02, 1.15107616e+00, 3.00000000e+00],
[2.01000000e+02, 3.04000000e+02, 1.17425244e+00, 3.00000000e+00],
[8.00000000e+01, 3.33000000e+02, 1.18965003e+00, 4.00000000e+00],
[2.39000000e+02, 2.77000000e+02, 1.19447631e+00, 5.00000000e+00],
[2.67000000e+02, 2.84000000e+02, 1.19762593e+00, 4.00000000e+00],
[2.91000000e+02, 3.52000000e+02, 1.21086663e+00, 8.00000000e+00],
[2.12000000e+02, 2.95000000e+02, 1.22430084e+00, 4.00000000e+00],
[2.37000000e+02, 3.53000000e+02, 1.25558804e+00, 7.00000000e+00],
[4.60000000e+01, 3.37000000e+02, 1.26747080e+00, 4.00000000e+00],
[2.81000000e+02, 3.56000000e+02, 1.27040407e+00, 6.00000000e+00],
[1.17000000e+02, 1.40000000e+02, 1.31633698e+00, 2.00000000e+00],
[2.89000000e+02, 3.18000000e+02, 1.35853968e+00, 7.00000000e+00],

```
[1.05000000e+02, 1.36000000e+02, 1.35894197e+00, 2.00000000e+00],
[8.60000000e+01, 2.41000000e+02, 1.42996578e+00, 3.00000000e+00],
[3.14000000e+02, 3.30000000e+02, 1.44143326e+00, 8.00000000e+00],
[2.72000000e+02, 3.39000000e+02, 1.44367456e+00, 7.00000000e+00],
[3.19000000e+02, 3.49000000e+02, 1.49048772e+00, 7.00000000e+00],
[1.55000000e+02, 3.06000000e+02, 1.51266508e+00, 5.00000000e+00],
[3.16000000e+02, 3.38000000e+02, 1.53863258e+00, 7.00000000e+00],
[3.15000000e+02, 3.28000000e+02, 1.59728762e+00, 4.00000000e+00],
[2.93000000e+02, 3.54000000e+02, 1.66251848e+00, 6.00000000e+00],
[2.69000000e+02, 3.03000000e+02, 1.68272090e+00, 7.00000000e+00],
[3.35000000e+02, 3.51000000e+02, 1.69186688e+00, 6.00000000e+00],
[1.86000000e+02, 3.32000000e+02, 1.71860425e+00, 3.00000000e+00],
[9.00000000e+01, 3.00000000e+02, 1.73171721e+00, 3.00000000e+00],
[3.23000000e+02, 3.55000000e+02, 1.75179707e+00, 1.20000000e+01],
[3.29000000e+02, 3.41000000e+02, 1.75861311e+00, 8.00000000e+00],
[3.05000000e+02, 3.50000000e+02, 1.80923988e+00, 6.00000000e+00],
[3.34000000e+02, 3.45000000e+02, 1.84988922e+00, 1.10000000e+01],
[3.17000000e+02, 3.63000000e+02, 1.87275987e+00, 5.00000000e+00],
[3.27000000e+02, 3.69000000e+02, 1.87419256e+00, 9.00000000e+00],
[3.44000000e+02, 3.66000000e+02, 1.93391093e+00, 1.20000000e+01],
[3.31000000e+02, 3.36000000e+02, 1.93449551e+00, 1.10000000e+01],
[1.06000000e+02, 3.62000000e+02, 1.98269631e+00, 3.00000000e+00],
[2.90000000e+02, 3.80000000e+02, 2.14511547e+00, 1.20000000e+01],
[3.42000000e+02, 3.58000000e+02, 2.19789326e+00, 8.00000000e+00],
[3.09000000e+02, 3.26000000e+02, 2.23014274e+00, 7.00000000e+00],
[3.61000000e+02, 3.78000000e+02, 2.23360274e+00, 1.80000000e+01],
[3.21000000e+02, 3.47000000e+02, 2.33952440e+00, 1.10000000e+01],
[3.60000000e+02, 3.74000000e+02, 2.35166683e+00, 5.00000000e+00],
[3.07000000e+02, 3.25000000e+02, 2.42385036e+00, 1.10000000e+01],
[3.64000000e+02, 3.65000000e+02, 2.44016775e+00, 1.50000000e+01],
[3.43000000e+02, 3.68000000e+02, 2.64015506e+00, 1.30000000e+01],
[3.40000000e+02, 3.79000000e+02, 2.89543899e+00, 8.00000000e+00],
[3.46000000e+02, 3.71000000e+02, 2.95830200e+00, 1.10000000e+01],
[3.67000000e+02, 3.88000000e+02, 3.13497264e+00, 1.60000000e+01],
[3.59000000e+02, 3.85000000e+02, 3.13968327e+00, 1.40000000e+01],
[3.70000000e+02, 3.83000000e+02, 3.28298077e+00, 9.00000000e+00],
[3.57000000e+02, 3.81000000e+02, 3.35569600e+00, 1.90000000e+01],
[2.54000000e+02, 3.48000000e+02, 3.55592895e+00, 6.00000000e+00],
[3.94000000e+02, 3.99000000e+02, 3.65396128e+00, 1.70000000e+01],
[3.82000000e+02, 3.92000000e+02, 3.68795481e+00, 2.40000000e+01],
[3.73000000e+02, 3.96000000e+02, 3.71191340e+00, 1.70000000e+01],
[3.75000000e+02, 3.93000000e+02, 3.76632419e+00, 2.00000000e+01],
[3.90000000e+02, 3.91000000e+02, 3.81120326e+00, 2.60000000e+01],
[3.72000000e+02, 3.87000000e+02, 4.19413130e+00, 2.40000000e+01],
[3.76000000e+02, 3.95000000e+02, 4.22607600e+00, 2.40000000e+01],
[3.86000000e+02, 4.02000000e+02, 4.42405017e+00, 2.40000000e+01],
[3.77000000e+02, 3.97000000e+02, 4.72736356e+00, 1.50000000e+01],
[3.89000000e+02, 4.06000000e+02, 5.61484242e+00, 2.90000000e+01],
[3.84000000e+02, 4.05000000e+02, 6.38141312e+00, 3.60000000e+01],
[4.00000000e+02, 4.04000000e+02, 6.43252343e+00, 4.30000000e+01],
[4.03000000e+02, 4.09000000e+02, 7.15098291e+00, 4.90000000e+01],
[4.08000000e+02, 4.10000000e+02, 7.99378855e+00, 5.10000000e+01],
[4.01000000e+02, 4.11000000e+02, 8.83472517e+00, 6.70000000e+01],
[3.98000000e+02, 4.13000000e+02, 8.85110324e+00, 7.00000000e+01],
[4.07000000e+02, 4.12000000e+02, 9.41341726e+00, 7.30000000e+01],
[4.14000000e+02, 4.16000000e+02, 2.16202394e+01, 1.40000000e+02],
[4.15000000e+02, 4.17000000e+02, 3.97908992e+01, 2.10000000e+02]])
```

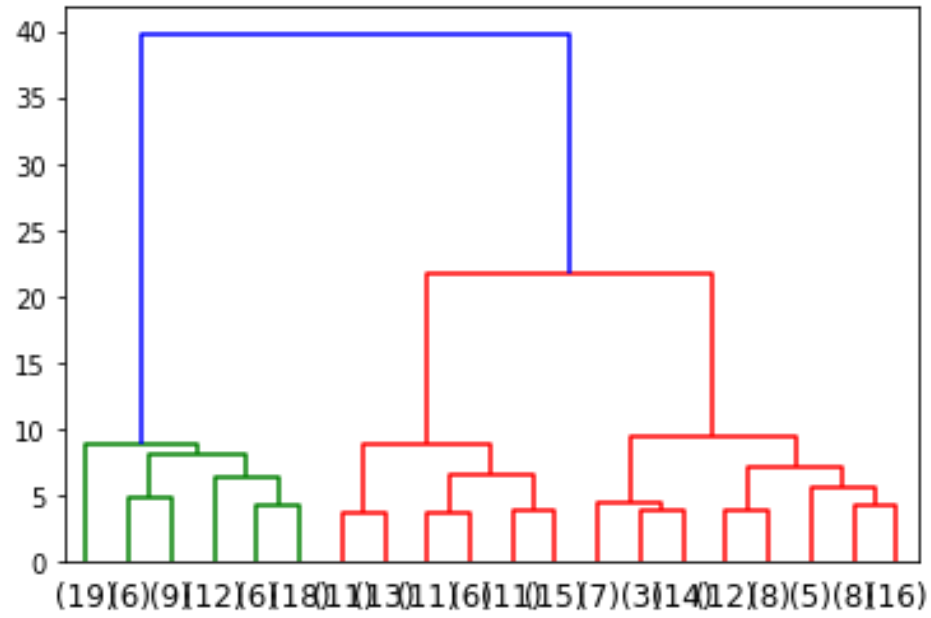
Dendrogram: A dendrogram is a tree like structure that summarizes the process of clustering. On x-axis are the records and y-axis represents the distance. Similar records are joined by the lines whose vertical length represents the distance between the records.

The greater the distance the more the dissimilarities. Bu closing the cut-off distance on y-axis a set of clusters is created.

Below is the dendrogram for our dataset obtained by using the distance matrix generated above:



To have a clear view of this graph, we gave a cut-off range of last 20 clusters i.e. p=20:



Upon observing this dendrogram it is observed that two clusters are suggested for our dataset (one in green and the other in red). Thus the no. of optimum clusters = 2.

We are able to identify the no. of clusters, now our next step is to form the clusters:

We will be using 'fcluster' in scipy to form our clusters in python (refer python notebook for details). Giving our dataset and no. of clusters we want to make as the parameter, we have separated our data points into two separate clusters and attached the same with our primary data as a column (H_cluster), please refer below sample:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	H_clusters
0	19.94	16.92	0.875200	6.675	3.763	3.252000	6.550	1
1	15.99	14.89	0.906400	5.363	3.582	3.336000	5.144	2
2	18.95	16.42	0.882900	6.248	3.755	3.368000	6.148	1
3	10.83	12.96	0.810588	5.278	2.641	5.182000	5.185	2
4	17.99	15.86	0.899200	5.890	3.694	2.068000	5.837	1
5	12.70	13.41	0.887400	5.183	3.091	8.079625	5.000	2
6	12.02	13.33	0.850300	5.350	2.810	4.271000	5.308	2
7	13.74	14.05	0.874400	5.482	3.114	2.932000	4.825	2
8	18.17	16.26	0.863700	6.271	3.512	2.853000	6.273	1
9	11.23	12.88	0.851100	5.140	2.795	4.325000	5.003	2

By analysing the data in the two clusters (1 and 2), we noticed that the cluster 1 comprises of higher values i.e. it consists of customers whose monthly spending is on higher side, amount paid by these customers is on the higher side, more spending in single shopping etc., refer below screenshot for the same:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	H_clusters
0	19.94	16.92	0.8752	6.675	3.763	3.252	6.550	1
159	17.36	15.76	0.8785	6.145	3.574	3.526	5.971	1
73	20.20	16.89	0.8894	6.285	3.864	5.173	6.187	1
72	20.10	16.99	0.8746	6.581	3.785	1.955	6.449	1
160	20.88	17.05	0.9031	6.450	4.032	5.016	6.321	1
162	18.88	16.26	0.8969	6.084	3.764	1.649	6.109	1
68	19.15	16.45	0.8890	6.245	3.815	3.084	6.185	1
163	17.26	15.73	0.8763	5.978	3.594	4.539	5.791	1
66	20.71	17.23	0.8763	6.579	3.814	4.451	6.451	1
65	18.85	16.17	0.9056	6.152	3.806	2.843	6.200	1

Cluster 2 on the other hand comprises of customers whose spending and expanses are on a lower side, below are few such customer records:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	H_clusters
209	15.57	15.15	0.8527	5.920	3.231	2.640	5.879	2
176	14.69	14.49	0.8799	5.563	3.259	3.586	5.219	2
100	13.37	13.78	0.8849	5.320	3.128	4.670	5.091	2
102	15.38	14.66	0.8990	5.477	3.465	3.600	5.439	2
172	14.38	14.21	0.8951	5.386	3.312	2.462	4.956	2
104	15.69	14.75	0.9058	5.527	3.514	1.599	5.046	2
1	15.99	14.89	0.9064	5.363	3.582	3.336	5.144	2
171	11.49	13.22	0.8263	5.304	2.695	5.388	5.310	2
107	13.32	13.94	0.8613	5.541	3.073	7.035	5.440	2
108	12.80	13.47	0.8860	5.160	3.126	4.873	4.914	2

We have categorised our data into two separate clusters and imported the data into a csv which will be use full for the business in decision making.

1.4 Apply K-Means clustering on scaled data and determine optimum clusters. Apply elbow curve and silhouette score.

K-Means clustering:

A non-hierarchical approach to forming a good cluster is to pre-specify a desired no. of clusters, *k*. Assign each record to one of the k clusters according to their distance from each cluster so as to minimize the measure of dispersion within the cluster. The '*means*' in k-means clustering refers to averaging of data i.e. finding the centroid.
Before starting with the model we have to determine how many clusters to make i.e. the value of k.
Given an initial set of k means m1(1), …, mk(1) , the algorithm proceeds by alternating between two steps:
Assignment step: Assign each observation to the cluster whose mean has the least squared
Euclidean distance; this is intuitively the "nearest" mean.
Update step: Calculate the new means (centroids) of the observations in the new clusters.
The algorithm has converged when the assignments no longer changes.

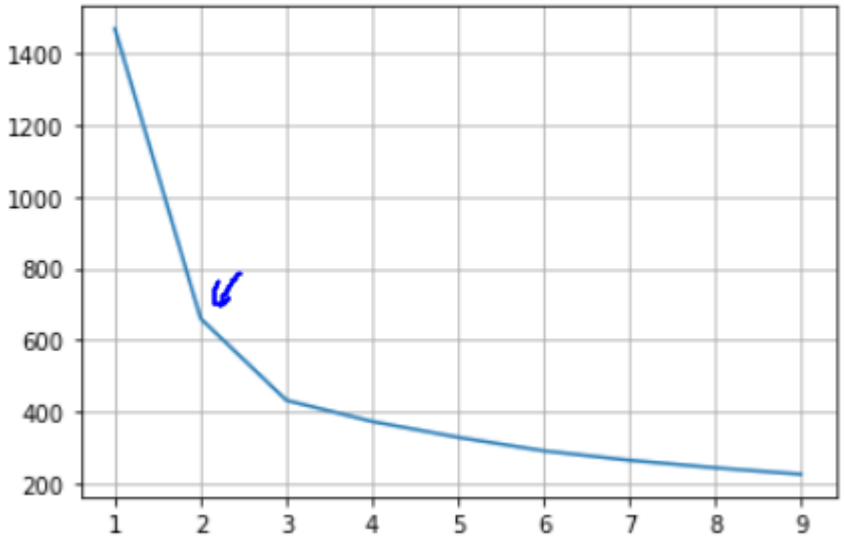
We will be using sklearn.cluster library in python for k-means clustering. As we have already analysed and scaled our data, let's proceed with the method:
To perform k-means clustering, we need to identify the value of k first. Let's k = 2 (say). Thus we will give no. of clusters, n_cluster = 2 and fit our model with the dataset. We will obtain an array of no. of elements = no. of data points (rows in the dataset) which will indicate which element belongs to which cluster. Similarly, we will do the same for k=3, k=4 and so on and will analyse their within sum of square values. The within sum of square value for k = 2 is 659.1717 and for k = 3 is 430.6589 and so on.

Elbow method:

For a given number of clusters, the total within-cluster sum of squares (WSS) is computed. That value of k is chosen to be optimum, where addition of one more cluster does not lower the value of total WSS appreciably. The Elbow method looks at the total WSS as a function of the number of clusters.
We have created a list of within cluster sum of squares (WSS) for different values of k:

```
[1470.0,
 659.1717544870407,
 430.65897315130053,
 371.38509060801096,
 327.2127816566136,
 289.315995389595,
 262.98186570162255,
 241.81894656086018,
 223.9125422100272]
```

1470.0 corresponds to WSS for k = 1, 659.1717 corresponds to WSS for k = 2 and so on…
Plotting these WSS values against different values of k we get the below elbow ploy:



Here we can find an elbow in the graph for k = 2 showing a clear break in the plot at k =2 and the dip is not too significant after this point, and thus k=2 are the optimal no. of clusters for out dataset.

Silhouette Method:

This method measures how tightly the observations are clustered and the average distance between clusters. For each observation a silhouette score is constructed which is a function of the average distance between the point and all other points in the cluster to which it belongs, and the distance between the point and all other points in all other clusters, that it does not belong to. The maximum value of the statistic indicates the optimum value of k.
Now verifying this using Silhouette width, this is used to check how much the clusters are separated from each other.

S_Width = $(d_2 – d_1)/\max(d_1, d_2)$

Here, S_width ➔ Silhouette Width

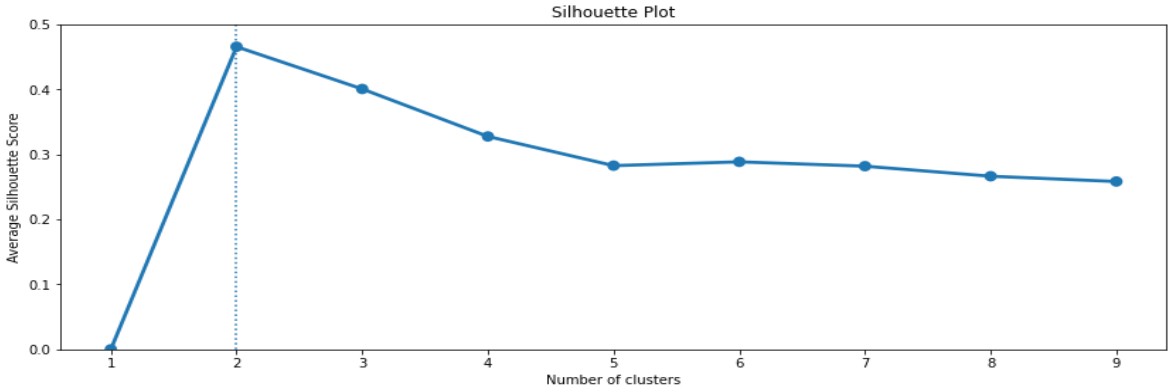
d_1 ➔ Distance of a point (p) to the centroid of the cluster it belongs

d_2 ➔ Distance of point (p) to the centroid of nearby cluster.

When we calculate the Silhouette width of every element and then take the average of it, then it is called ‘Silhouette Score’. If the Silhouette score is close to 1 then we can say that the clusters are clearly separated, if the score is close to 0 then we can say that the clusters are not clearly separated and if the score is close to -1 then we can conclude that the model is wrongly setup.
Calculating Silhouette score for different values of k:

```
The Average Silhouette Score for 2 clusters is 0.46577
The Average Silhouette Score for 3 clusters is 0.40073
The Average Silhouette Score for 4 clusters is 0.32765
The Average Silhouette Score for 5 clusters is 0.28273
The Average Silhouette Score for 6 clusters is 0.2886
The Average Silhouette Score for 7 clusters is 0.28191
The Average Silhouette Score for 8 clusters is 0.26644
The Average Silhouette Score for 9 clusters is 0.25831
```

Plotting these average Silhouette Score:



Here we can see that the average Silhouette Score is maximum for 2 clusters, thus we will proceed with no. of clusters =2 for our dataset.
We have also calculated the min silhouette width for different values of 'k' and observed that the **min S_Width for k = 2 is -0.006** (negative and almost close to zero) however, the average Silhouette Score for 2 clusters is maximum hence we still consider that the overall two clusters are separated from each other (except for one individual record).
Thus, from both Elbow and Silhouette Score method we can conclude that optimum no. of clusters, *k* = 2.

1.5 Describe cluster profiles for the clusters defined. Recommend different promotional strategies for different clusters.

Conclusion:

Hierarchical Clustering:

Using agglomerative clustering, we have categorised our data in two separate clusters:

- Cluster 1: It consists of higher magnitude values of various features (independent variables) for our bank marketing dataset. It has customers whose monthly spending is high, amount paid by these customers is on the higher side and are spending more in single shopping etc.,
- Cluster 2: It consist of customers whose spending is on lower side and thus the amount paid by them is comparatively small, also these are the customers who do not spend much on single shopping.

K-Means Clustering:

We have categorised the records in two clusters, cluster 0 and cluster 1, below are the count of records in each cluster:

```
0    133
1     77
Name: Clus_kmeans, dtype: int64
```

The Cluster profile is given as:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	freq
Clus_kmeans								
0	12.9306	13.693459	0.863619	5.339699	3.025917	3.822845	5.081737	133
1	18.15857	16.054805	0.883817	6.127429	3.660519	3.480417	5.97174	77

Cluster 0 in k-means is similar to cluster 2 of hierarchical clustering and consists of customers with low spending and lower average spending in single shopping.
Cluster 1 in k-means is similar to cluster 1 in hierarchical clustering, consists of higher magnitude values of various features (independent variables) for our bank marketing dataset. It has customers whose monthly spending is high, amount paid by these customers is on the higher side and are spending more in single shopping etc.

Recommendations and Promotional strategies:
For the customers in cluster 1 (both k-means and hierarchical), as they have spending on the higher side and are open to spend significant amount in single shopping, bank can actually increase their credit limits so that they can use this limit and spend more every month.
Also, banks can give various offers to these customers on expansive/luxurious items so that they are motivated to spend on these items.
For the customers in cluster 2 (which is cluster 0 in k-means), as their spending is on the lower side, the bank can target these customers and give them various offers if they choose to make advance payments.
Also, bank can provide offers/cashbacks if they spend more than some cut-off amount in a month, so that these customers are motivated to spend more every month.

Problem Statement 2:

An Insurance firm providing tour insurance is facing higher claim frequency. The management decides to collect data from the past few years. You are assigned the task to make a model which predicts the claim status and provide recommendations to management. Use CART, RF & ANN and compare the models' performances in train and test sets.

2.1 Data Ingestion: Read the dataset. Do the descriptive statistics and do null value condition check, write an inference on it.
2.2 Data Split: Split the data into test and train, build classification model CART, Random Forest, Artificial Neural Network
2.3 Performance Metrics: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score for each model
2.4 Final Model: Compare all the model and write an inference which model is best/optimized.
2.5 Inference: Basis on these predictions, what are the business insights and recommendations.

Solution:

2.1 Data Ingestion: Read the dataset. Do the descriptive statistics and do null value condition check, write an inference on it.

Data Analysis:

Here we are given the data of an Insurance firm providing tour insurance, below are some sample records from the dataset:

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	Destination
0	48	C2B	Airlines	No	0.70	Online	7.0	2.51	Customised Plan	ASIA
1	36	EPX	Travel Agency	No	0.00	Online	34.0	20.00	Customised Plan	ASIA
2	39	CWT	Travel Agency	No	5.94	Online	3.0	9.90	Customised Plan	Americas
3	36	EPX	Travel Agency	No	0.00	Online	4.0	26.00	Cancellation Plan	ASIA
4	33	JZI	Airlines	No	6.30	Online	53.0	18.00	Bronze Plan	ASIA

On analysing the data, we get that there are a total of 3000 rows and 10 columns present in the dataset and there are no missing values present. However **we can see that the column 'Duration' has values like '-1' and '0'.** Considering '0' as a valid input and since 'Duration' of the journey can't be negative thus we need to clean this data where Duration = -1. Replacing this -1 with the NULL value in our data set, we can see that there is one such record which is now showing a NULL value in column Duration:

```
Column Name     Count of Null
Age              0
Agency_Code      0
Type             0
```

```
Claimed          0
Commision        0
Channel          0
Duration         1
Sales            0
Product Name     0
Destination      0
dtype: int64
```

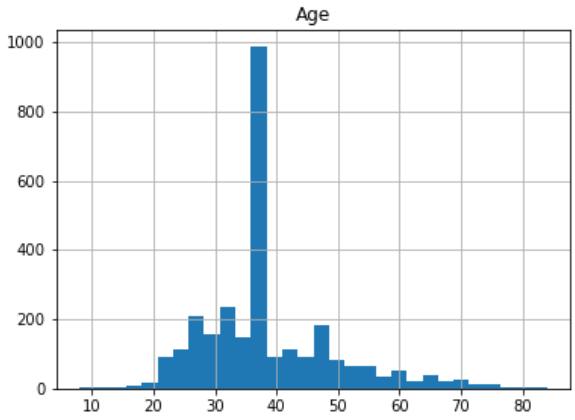
Now looking more into the info about the data, we can see that Age, Commission, Duration and Sales are numeric co lumns and rest are object type:

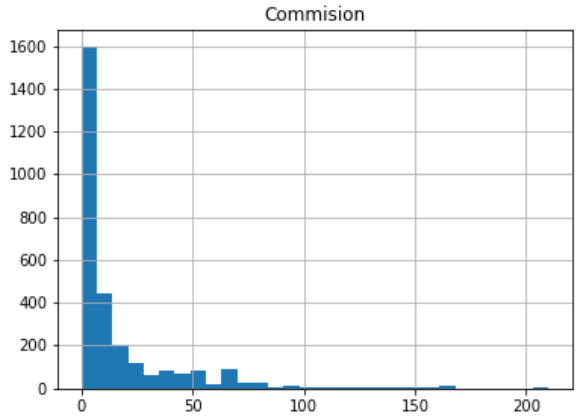
```
#   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Age            3000 non-null   int64
 1   Agency_Code    3000 non-null   object
 2   Type           3000 non-null   object
 3   Claimed        3000 non-null   object
 4   Commision      3000 non-null   float64
 5   Channel        3000 non-null   object
 6   Duration       2999 non-null   float64
 7   Sales          3000 non-null   float64
 8   Product Name   3000 non-null   object
 9   Destination    3000 non-null   object
dtypes: float64(3), int64(1), object(6)
```

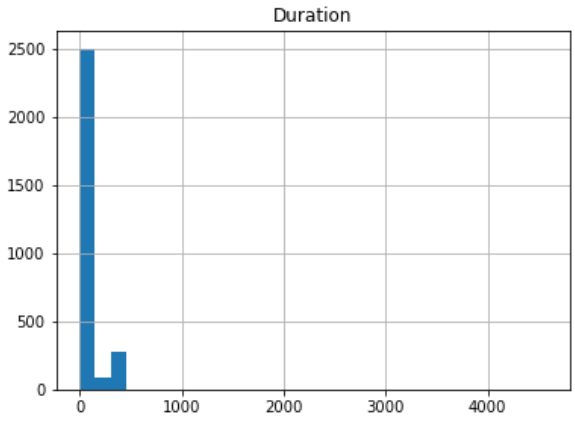
We can also see that there are 139 duplicate records in our data, and we need to remove these duplicate records before proceeding, thus deleting these duplicate. Also, there is a NULL value for column 'Duration' and we have replaced this null value with the mean (average journey duration) of the Duration column.
Since we have removed the duplicates and replaced the Null values, let's get into the statistical description of this dataset:

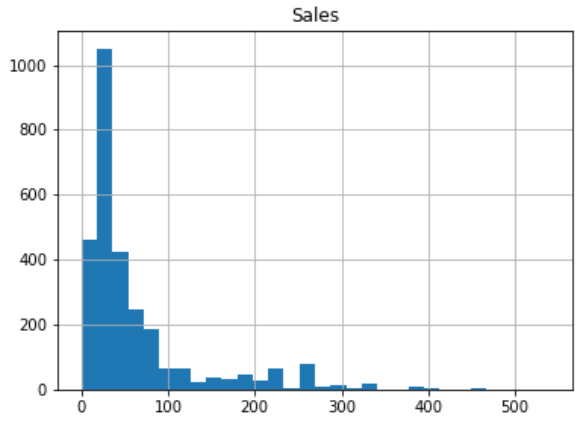
	Age	Commision	Duration	Sales
count	2861.000000	2861.000000	2861.000000	2861.000000
mean	38.204124	15.080996	72.145063	61.757878
std	10.678106	25.826834	135.970329	71.399740
min	8.000000	0.000000	0.000000	0.000000
25%	31.000000	0.000000	12.000000	20.000000
50%	36.000000	5.630000	28.000000	33.500000
75%	43.000000	17.820000	66.000000	69.300000
max	84.000000	210.210000	4580.000000	539.000000

Now the points to note here are, that all these numerical columns have quite large range (large (max - min)). The average Sale we can see is 61.76 with the min value of 0 and max value of 539, and for duration column as well the min and the average values are 0 and 75.14 respectively, however the max value is 4580, which is quite large thus **suggesting that the data is skewed for these variables.**
Let's plot the histogram for these columns and see the variation:





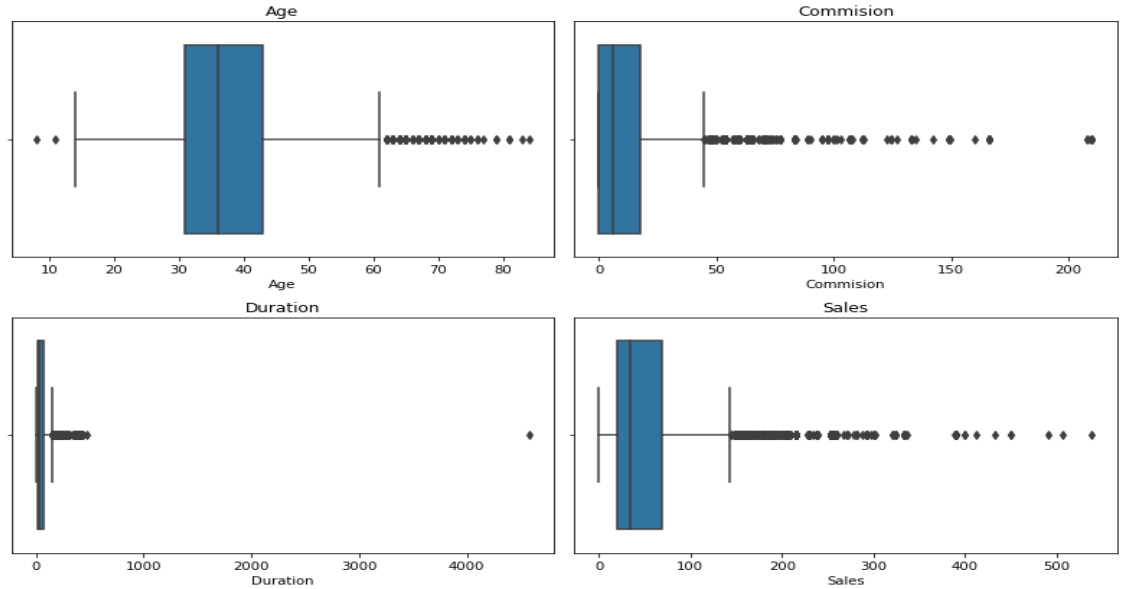




Below is the skewness shown by these variables:

```
Age           1.103145
Commision     3.104741
Duration     13.787692
Sales         2.344643
dtype: float64
```

The above plot and table suggests that out data is highly skewed and these might be the presence of large no. of outl iers too. Checking the presence of outliers using box and whiskers plot:



Before proceeding with the model preparation, there is no harm is treating these outliers, and we will be using IQR method for the same:

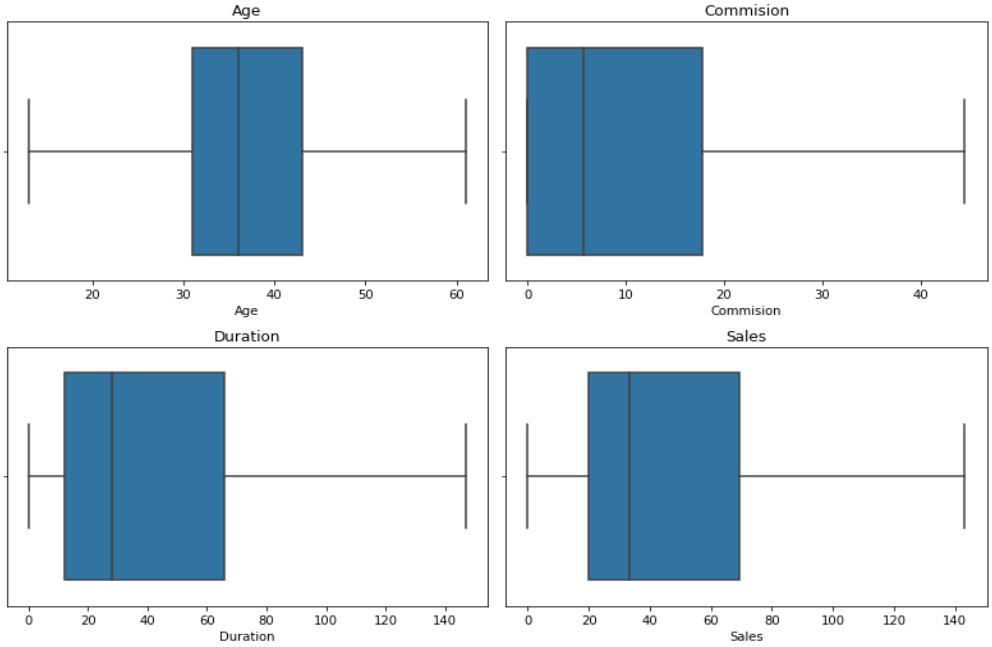
$$IQR = Q3 - Q1.$$

We define the upper and the lower limit as below and if any data point has value greater than the upper limit we will replace it with the upper range value itself and if any data point has value less than the lower limit then we will replace it with the lower range value itself.

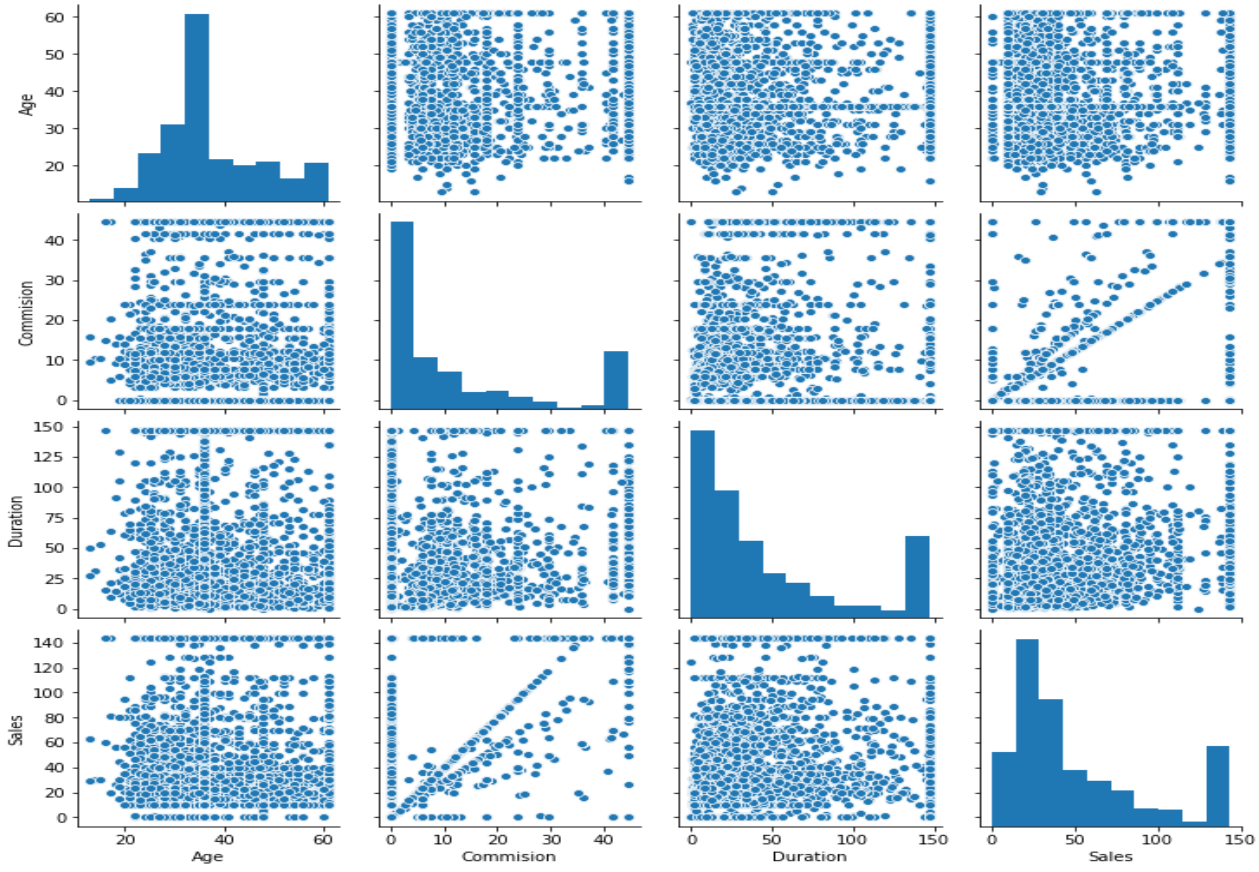
lower_range= Q1-(1.5 * IQR)

upper_range= Q3+(1.5 * IQR)

Box and whiskers plot after treating the outliers:



Getting into the multivariate analysis of our data and plotting the pair plot and correlation matrix (heat map):



Correlation Table and heat map:

	Age	Commision	Duration	Sales
Age	1.000000	0.071246	0.008528	0.021450
Commision	0.071246	1.000000	0.453104	0.682537
Duration	0.008528	0.453104	1.000000	0.534180
Sales	0.021450	0.682537	0.534180	1.000000

Here, we can see that there is very small or no correlation between many variables, however significant amount of correlation can be seen between '*Sales'* and '*Commision'* and also '*Sales'* and '*Duration'*.

2.2 Data Split: Split the data into test and train, build classification model CART, Random Forest, Artificial Neural Network.

Before be prepare our model, we split our data into 'training and testing' sets, the training data is fitted to the model and the test data is used to check how our model is performing. Usually we split our data as 70% training and 30% testing.

As we saw earlier that few of our variables are on 'Object' data type, thus there is a need to convert them into categorical variables (*int8*). Please refer python notebook file for conversion process:

Once we convert our variables to categorical variables:

```
#   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Age            2861 non-null   float64
 1   Agency_Code    2861 non-null   int8
 2   Type           2861 non-null   int8
 3   Claimed        2861 non-null   int8
 4   Commision      2861 non-null   float64
 5   Channel        2861 non-null   int8
 6   Duration       2861 non-null   float64
 7   Sales          2861 non-null   float64
 8   Product Name   2861 non-null   int8
 9   Destination    2861 non-null   int8
dtypes: float64(4), int8(6)
```

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	Destination
count	2861.000000	2861.000000	2861.000000	2861.000000	2861.000000	2861.000000	2861.000000	2861.000000	2861.000000	2861.000000
mean	37.896190	1.280671	0.597344	0.319469	11.756865	0.983922	47.363868	51.085089	1.666550	0.261797
std	9.821593	1.003773	0.490518	0.466352	15.502632	0.125799	47.291025	42.604294	1.277822	0.586239
min	13.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	31.000000	0.000000	0.000000	0.000000	0.000000	1.000000	12.000000	20.000000	1.000000	0.000000
50%	36.000000	2.000000	1.000000	0.000000	5.630000	1.000000	28.000000	33.500000	2.000000	0.000000
75%	43.000000	2.000000	1.000000	1.000000	17.820000	1.000000	66.000000	69.300000	2.000000	0.000000
max	61.000000	3.000000	1.000000	1.000000	44.550000	1.000000	147.000000	143.250000	4.000000	2.000000

Now here we observe that our target variable '**Claimed**' is now categorical where '0' represents 'No' and '1' represents 'Yes'. On further analysis, out of 2861 unique records, there are **914 people who has claimed for the insurance** (i.e. 914 one's are there in column Claimed) which makes it **31.94%** of the total data we have. Separating this dependent variable (Claimed) from the independent variable so that we can split out independent features into training and testing.
To split the data we will be using '**train_test_split'** functionality from '**model_selection'** sub-library in **sklearn**.

CART Model (Classification And Regression Trees):

Decision Tree: Decision trees are commonly used in data mining with the objective of creating model that predicts the value of target (dependent variable) based on the values of the several inputs (independent variables).
In our case the target variable (Claimed) is categorical (Yes/No) and the tree in this model is used to identify the 'Class' within which the target variable is likely to fall into, thus the tree here is **Classification Tree.**
We will be using '*DecisionTreeClassifier*' from sklearn.tree library to create out CART model and the criterion we will be using is 'Gini impurities', please refer python notebook file for details:
Criterion: 'gini' ➔ The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Calculated as:

$$Gini\,(D) = 1 - \sum_{i=1}^{m} p_i^{\,2}$$

m ➔ No. of classes; p➔ Probability that a record in D belongs to class Ci

Randon_state = 1 ➔ To ensure that the split (data) doesn't change every time the model runs.

Now fitting this model with the training data and importing the .dot file in our local using 'tree.export_graphviz'. Once we have generated this .dot file, we have to copy the content of this file and paste it to http://webgraphviz.com/ in our web browser to generate the tree. As this is a fully grown tree, there is a need to prone this tree with various parameters:
GridSearchCV: This is used to give multiple values of parameter like max_depth, min_sample_leaf, etc. and using GridSearchCV we can find which combination of variables gives the best output. Below is the with different values that we have provided for our parameters:

```
param_grid = {
    'max_depth': [5, 6, 7, 8],
    'min_samples_leaf': [20, 25, 30, 35],
    'min_samples_split': [60, 75, 90, 105]
}
```

Here,
max_depth ➔ Max no. of levels the decision tree can have
Min_sample_leaf ➔ min no. of elements in the leaf node, usually we give this as 1-3% of our training data.
Min_sample_split ➔ min no. of elements to split a node

After fitting out model with the training data and using the above mentioned parameters, we get the combination of best parameters as:

```
grid_search.best_params_
```

```
{'max_depth': 5, 'min_samples_leaf': 25, 'min_samples_split': 75}
```

Creating the Classification report to check the performance of our model, using training and test data:
For training data:

	precision	recall	f1-score	support
0	0.82	0.87	0.84	1359
1	0.69	0.58	0.63	643
accuracy			0.78	2002
macro avg	0.75	0.73	0.74	2002
weighted avg	0.77	0.78	0.78	2002

For test data:

	precision	recall	f1-score	support
0	0.81	0.87	0.84	588
1	0.66	0.54	0.60	271
accuracy			0.77	859
macro avg	0.73	0.71	0.72	859
weighted avg	0.76	0.77	0.76	859

Random Forest:

Random forest, the ensemble learning technique that combines several base models (decision trees) in order to produce one optimal predictive model.
For random forest we will be using, '*RandomForestClassifier*' from *sklearn.ensemble*. Same we have split our data as 70% training and 30% testing.

Below are the set of parameters that we have provided to build our model:

```
param_grid = {
   'max_depth': [6, 7],
   'max_features': [4, 5],
   'min_samples_leaf': [20, 25, 30],
   'min_samples_split': [45, 60, 75],
   'n_estimators': [101, 301]
}
```

After fitting out model with the training data and using the above mentioned parameters, we get the combination of best parameters as:

```
grid_search.best_params_
```

```
{'max_depth': 7,
 'max_features': 4,
 'min_samples_leaf': 20,
 'min_samples_split': 45,
 'n_estimators': 101}
```

Classification report for training data:

	precision	recall	f1-score	support
0	0.82	0.89	0.85	1359
1	0.72	0.58	0.64	643
accuracy			0.79	2002
macro avg	0.77	0.74	0.75	2002
weighted avg	0.79	0.79	0.79	2002

Classification report for test data:

	precision	recall	f1-score	support
0	0.82	0.89	0.85	588
1	0.70	0.57	0.63	271
accuracy			0.79	859
macro avg	0.76	0.73	0.74	859
weighted avg	0.78	0.79	0.78	859

Artificial Neural Network:

ANN, a machine learning algorithm that is roughly modelled around what is currently known about how the human brain functions.

We will be using '*MLPClassifier*' from *sklearn.neural_network* library to create out artificial neural network. Before we create out model let's scale our data using standard scaler function.

Standard Scaler standardizes a feature by subtracting the mean and then scaling to unit variance. Unit variance means dividing all the values by the standard deviation. Standard Scaler normalizes the data using the formula **(x-mean)/standard deviation**.

Same we have split our data as 70% training and 30% testing. Below are the set of parameters that we have provided to build our model:

```
param_grid = {
  'hidden_layer_sizes': [(100,100,100,100)],
  'activation': ['logistic', 'relu'],
  'solver': ['sgd', 'adam'],
  'tol': [0.1,0.01],
  'max_iter' : [500]
}
```

Here,

Hidden_layer ➔ We are passing a tuple with 3 elements, it means that there will be 3 hidden layers with 100 nodes each.

Max_iter ➔ max iterations to update the synaptic weights, this should not go beyond data count and usually its value is 10-15% of our data size.

Solver ➔ Stochastic Gradient or ADAM whichever suits the best.

Tol ➔ It's the tolerance indicating how precise the output should be, the smaller the tolerance value the longer the model will take time to give output however the more precise it will be.

After fitting out model with the training data and using the above mentioned parameters, we get the combination of best parameters as:

```
grid_search.best_params_
```

```
{'activation': 'relu',
 'hidden_layer_sizes': (100, 100, 100, 100),
 'max_iter': 500,
 'solver': 'adam',
 'tol': 0.01}
```

Classification report for training data:

	precision	recall	f1-score	support
0	0.81	0.85	0.83	1359
1	0.65	0.59	0.62	643
accuracy			0.77	2002
macro avg	0.73	0.72	0.72	2002
weighted avg	0.76	0.77	0.76	2002

Classification report for test data:

	precision	recall	f1-score	support
0	0.82	0.85	0.84	588
1	0.65	0.59	0.62	271
accuracy			0.77	859
macro avg	0.74	0.72	0.73	859
weighted avg	0.77	0.77	0.77	859

We will discuss about the classification report and model performance in detail in our later section.

2.3 Performance Metrics: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score for each model

Performance Metrics: Now since we have built the model, the next step is to check how our model is performing so that we can identify the accuracy and precision with which the model is supposed to give the output. For that we will be discussing about the classification report (which we created earlier), the confusion matrix, ROC curve and ROC_AUC score.

Confusion Matrix: It's a 2x2 tabular structure reflecting the performance of the model:

Confusion Matrix	Predicted Positive	Predicted Negative
Actual Positive	True Positive	False Negative
Actual Negative	False Positive	True Negative

Accuracy – How accurately / cleanly does the model classify the data-points. Lesser the false predictions, more the accuracy.

Accuracy = (TP + TN)/(TP + TN + FP + FN)

Sensitivity / Recall – How many of the actual True data points are identified as True data points by the model . Remember, False Negatives are those data points which should have been identified as True.

Recall = TP / TP + FN

Precision – Among the points identified as Positive by the model, how many are really Positive

Precision = TP / TP + FP

CART Model performance measures:

As we have learnt about the confusion matrix, let's proceed and create one for out CART model and for that we will be using **confusion_matrix** from **sklearn.matrices**. Below is the confusion matrix for our model:

```
array([[1189,  170],
       [ 268,  375]], dtype=int64)
```

Calculating the accuracy of our model for training data:

Accuracy_CART_training = Total True/Total population = (1189 + 375)/(1189 + 170 + 268 + 375) = **0.7812**

Thus the CART model we created has the accuracy of 78.12% for the training data.

Similarly, for test data:

```
array([[513,  75],
       [124, 147]], dtype=int64)
```

Accuracy_CART_test = (513 + 147)/(513 + 147 + 75 + 124) = **0.7683**

Classification Report: Training Data

	precision	recall	f1-score	support
0	0.82	0.87	0.84	1359
1	0.69	0.58	0.63	643
accuracy			0.78	2002
macro avg	0.75	0.73	0.74	2002
weighted avg	0.77	0.78	0.78	2002

Classification Report: Test Data

	precision	recall	f1-score	support
0	0.81	0.87	0.84	588
1	0.66	0.54	0.60	271
accuracy			0.77	859
macro avg	0.73	0.71	0.72	859
weighted avg	0.76	0.77	0.76	859

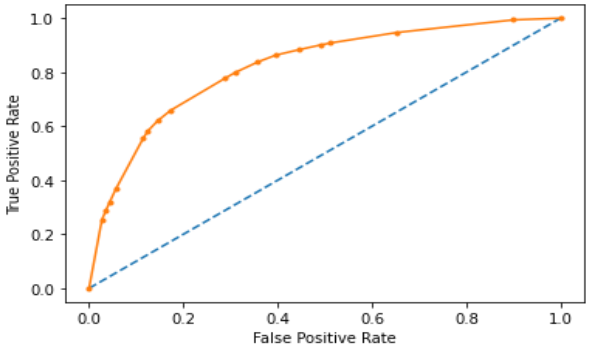
Comparing the two reports, we can say that the model did pretty well with 78% and 77% of accuracy for training and testing data respectively. Also the Recall value (sensitivity) of the model for 1's (i.e. for predicting the Claimed as Yes) is 0.58 and 0.54 for training and testing which is quite good (although on the lower side) as the test data prediction is very close to the training one. Thus we can say that there is no case of over fitting with this model.

ROC Curve and ROC_AUC score

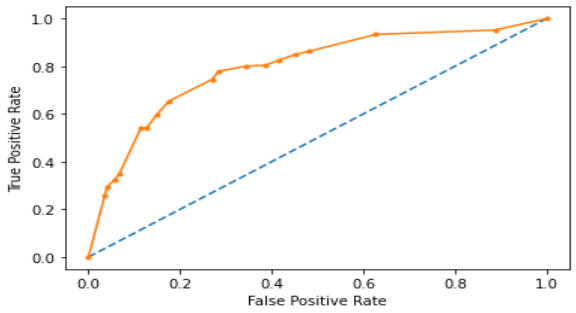
It is a graph between True Positive Rates and False Positive Rates.
TP rate = TP / total positive
FP rate = FP / total negative

ROC graph is a trade off between benefits (TP) and costs (FP)
Classifiers very close to Y axis and lower (nearer to x axis) are conservative models and strict in classifying positives (low TP rate). Classifiers on top right are liberal in classifying positives hence higher TP rate and FP rate.
It's the visual representation of model performance, the steeper the ROC curve; the stronger is the performance.

ROC Curve with training data



ROC Curve with test data



For training: Area Under the Curve, AUC = 0.820 i.e. 82%
For testing: Area Under the Curve, AUC = 0.795 i.e. 79.5%
Larger the AUC better the model will perform.

Random Forest Model performance measures:

Confusion Matrix for training and testing data:

For training data	For test data
`array([[1214, 145], [ 270, 373]], dtype=int64)`	`array([[519, 69], [117, 154]], dtype=int64)`

Accuracy_RF_Train = (1214 + 373)/(1214 + 145 + 270 + 373) = **0.792 = 79.2%**
Accuracy_RF_Test = (519 + 154)/(519 + 69 + 117 + 154) = **0.783 =78.3%**

Classification Report: Training Data

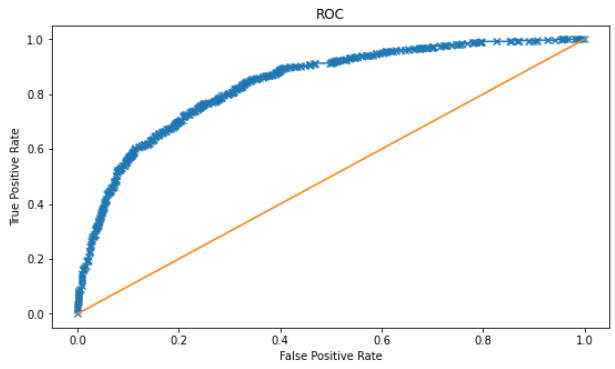
	precision	recall	f1-score	support
0	0.82	0.89	0.85	1359
1	0.72	0.58	0.64	643
accuracy			0.79	2002
macro avg	0.77	0.74	0.75	2002
weighted avg	0.79	0.79	0.79	2002

Classification Report: Test Data

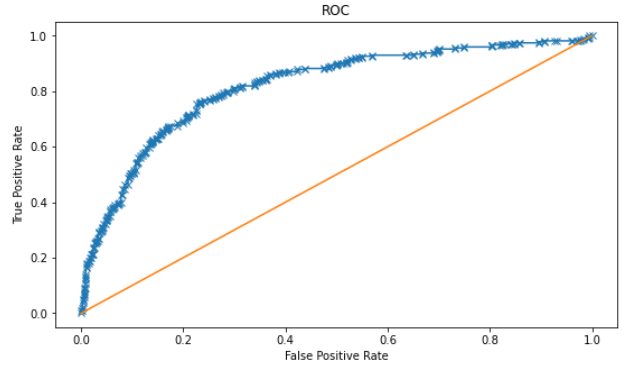
	precision	recall	f1-score	support
0	0.82	0.88	0.85	588
1	0.69	0.56	0.62	271
accuracy			0.78	859
macro avg	0.75	0.72	0.74	859
weighted avg	0.78	0.78	0.78	859

The accuracy for training and test data are 79% and 78% respectively for our Random Forest model. The recall values are 0.58 and 0.56 respectively and thus shows that the model did well to predict our test data and there are no signs of over fitting.

ROC Curve with training data



ROC Curve with test data



For training: Area under Curve is 0.8379 i.e. 83.8%
For testing: Area under Curve is 0.8183 i.e. 81.8%

ANN model performance:

Confusion Matrix for training and testing data:

For training data

```
array([[682, 677],
       [147, 496]], dtype=int64)
```

For test data

```
array([[293, 295],
       [ 61, 210]], dtype=int64)
```

Accuracy_ANN_Train = (682 + 496)/(682 + 677 + 147 + 496) = **0.5884 = 58.84%**
Accuracy_ANN_Test = (293 + 210)/(293 + 210 + 61 + 295) = **0.5855 = 58.55%**

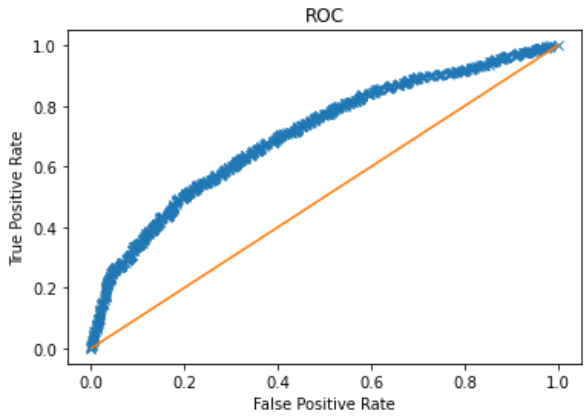
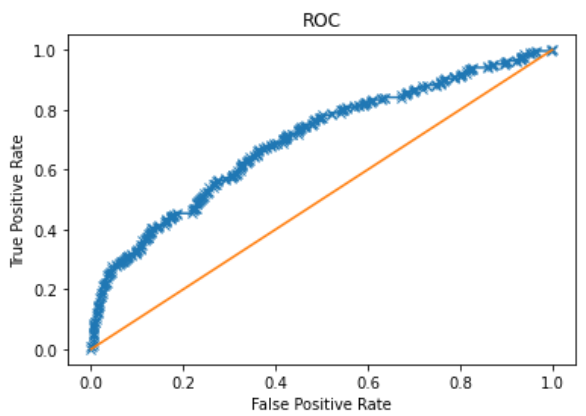
Classification Report: Training Data

	precision	recall	f1-score	support
0	0.82	0.50	0.62	1359
1	0.42	0.77	0.55	643
accuracy			0.59	2002
macro avg	0.62	0.64	0.58	2002
weighted avg	0.69	0.59	0.60	2002

Classification Report: Test Data

	precision	recall	f1-score	support
0	0.83	0.50	0.62	588
1	0.42	0.77	0.54	271
accuracy			0.59	859
macro avg	0.62	0.64	0.58	859
weighted avg	0.70	0.59	0.60	859

The accuracy for ANN model is quite low 59% approx. for both training and testing data. **The recall value is significant at 0.77** however the precision is on the lower side.

ROC Curve with training data	ROC Curve with test data
	

For training: Area under Curve is 0.7089 i.e. 70.9%
For testing: Area under Curve is 0.6989 i.e. 69.9%

2.4 Final Model: Compare all the model and write an inference which model is best/optimized.

As we checked the performance of all the models, we can see that the **CART** model has performed pretty well with **77% of accuracy and 79.5% of area under the curve**. Also the behaviour of model with the test data is very much similar to that of the training data, which is a good sign and we can say that the model is not over fitted.

On the other hand, **Random Forest** model has done even better with **accuracy and area under the curve of 78.3% and 81.8% respectively.** This model is also slightly better in predicting both the classes (0 and 1). With the below set of parameters, the model has given the best results with cross validation of 3 (cv = 3):

```
{'max_depth': 6,
 'max_features': 5,
 'min_samples_leaf': 20,
 'min_samples_split': 60,
 'n_estimators': 101}
```

However, Artificial Neural Network shows the below average results of 56% accuracy approx. and 70% of area under the curve, as the **recall value (sensitivity) is good with 0.77** we can say that the model is good in predicting the true value as true but the overall **precision of the model of 0.42** is the disadvantage of this model. Thus, we can say ANN is not the best model for the Insurance Dataset we have.

Conclusion: Although the CART model did a decent job with the accuracy and the prediction with its True Positive Rate against False Positive Rate, and its recall value of 0.56 is very much same as that of Random Forest model. But still the Random Forest model shows slightly better accuracy, precision and AUC score than the CART model, thus **Random Forest can be the best suited model for our scenario of insurance data set.**

2.5 Inference: Basis on these predictions, what are the business insights and recommendations

As suggested earlier, Random Forest can be the best suited model for our scenario of insurance data set. Using this model, it can be predicted that whether the customer will claim for the insurance or not with the accuracy of 78.3%, however the sensitivity of the model is on the lower side that is 56% and the f1 score is 0.62, which can be improved by offsetting this by predicting the probabilities and choosing a different cut-off, by default the cut off is 0.5.

Using this technique of random forest, business can look into the details of the customers (i.e. the independent variables), feed this data to the model and can know whether the customer will claim the insurance or not.
This way the Insurance firm providing tour insurance can deal with the higher claim frequency.

By Utkarsh Sharma
