7/24/2021

DATA MINING PROJECT

Name: - Ruhee Ansari
PGP-DSBA Online

Table of Contents

Problem 1: Clustering

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, do the necessary initial steps, and exploratory data analysis (Univariate, Bi-variate, and multivariate analysis).

```
bank=pd.read_csv('bank_marketing_part1_Data.csv')
```

```
bank.head()
```

	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

```
bank.tail()
```

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
205	13.89	14.02	0.8880	5.439	3.199	3.986	4.738
206	16.77	15.62	0.8638	5.927	3.438	4.920	5.795
207	14.03	14.16	0.8796	5.438	3.201	1.717	5.001
208	16.12	15.00	0.9000	5.709	3.485	2.270	5.443
209	15.57	15.15	0.8527	5.920	3.231	2.640	5.879

From top and bottom i.e., head and tail function data we can say that data is healthy, or we have good data from initial records.

```
bank.shape
print('There are {} number of rows and {} number of columns'.format(bank.shape[0],bank.shape[1]))
```

```
There are 210 number of rows and 7 number of columns
```

```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210 entries, 0 to 209
Data columns (total 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)
memory usage: 11.6 KB
```

```
bank.isnull().sum()
```

```
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
dtype: int64
```

```
bank.duplicated().sum()
```

```
0
```

The data set consist of 210 rows and 7 columns. So here we have 7 different attributes, all have same datatypes as float.

There are no null entries present in it and also no duplicate values.

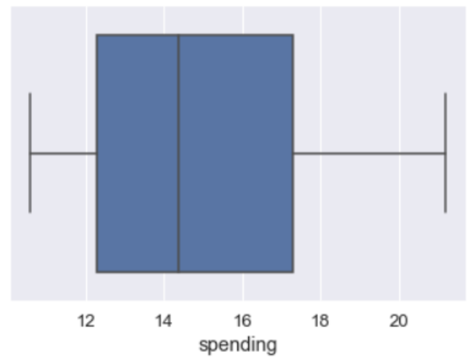
```
bank.describe()
```

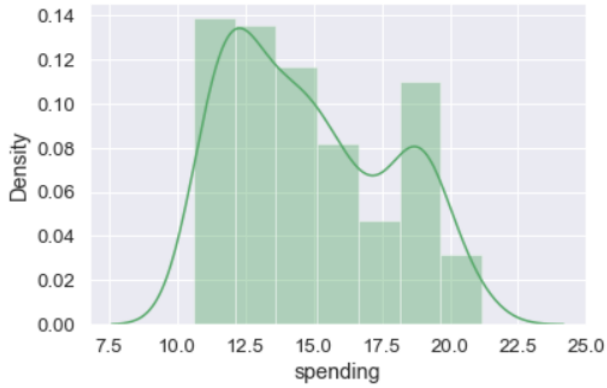
	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
count	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000
mean	14.847524	14.559286	0.870999	5.628533	3.258605	3.700201	5.408071
std	2.909699	1.305959	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

As all columns are numeric, description of all is presented here, min, max, std, 25%, 50%, 75%, total number of counts present.

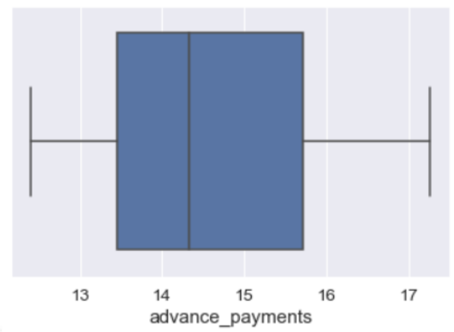
Univariant Analysis

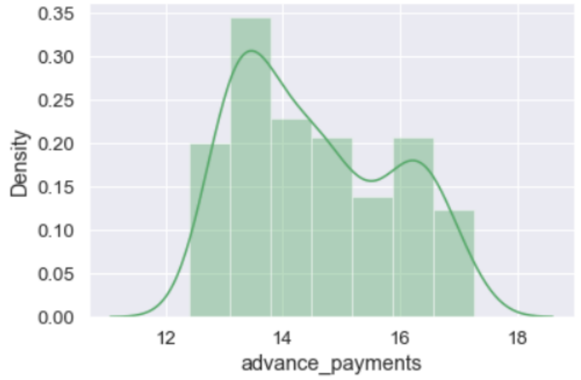
```
BoxPlot of spending
----------------------------------------------------------------
```



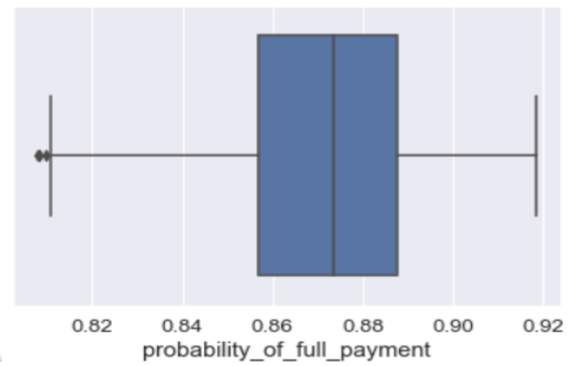


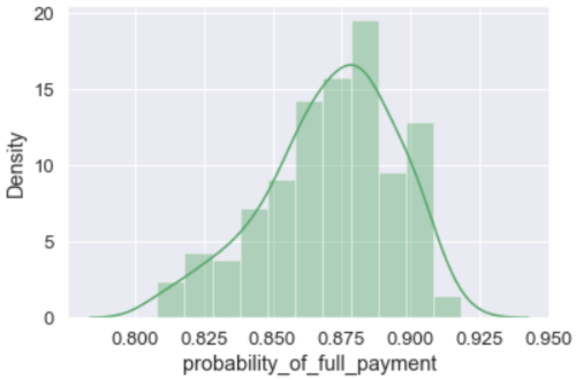

```
BoxPlot of advance_payments
----------------------------------------------------------------
```



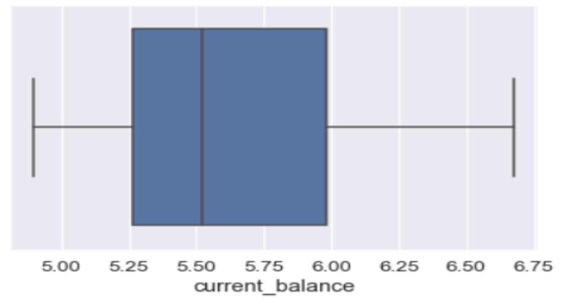


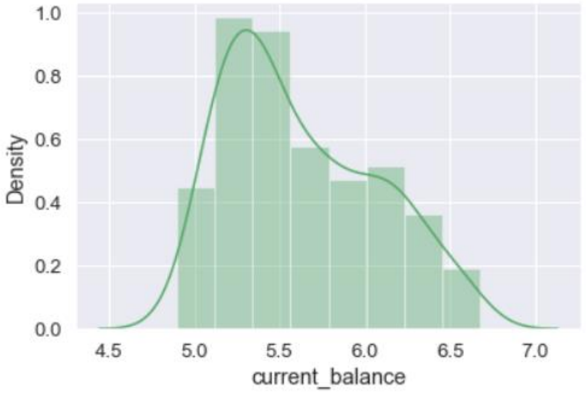

```
BoxPlot of probability_of_full_payment
----------------------------------------------------------------
```



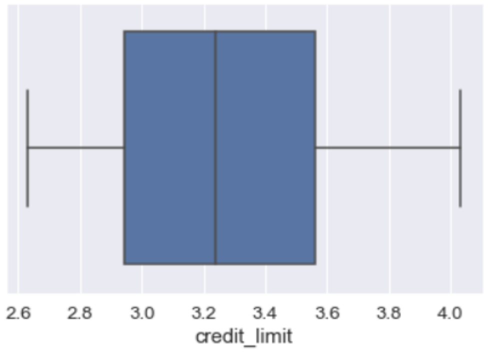


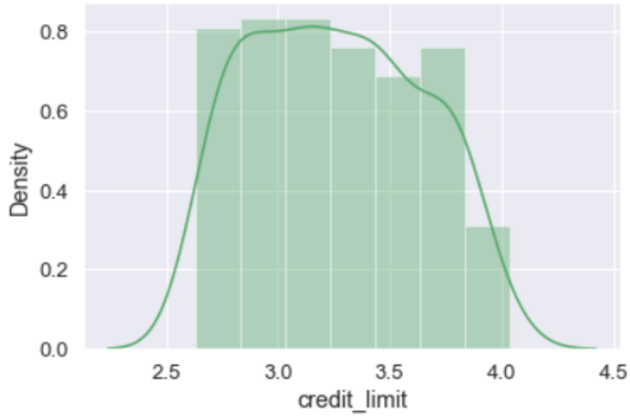

```
BoxPlot of current_balance
----------------------------------------------------------------
```



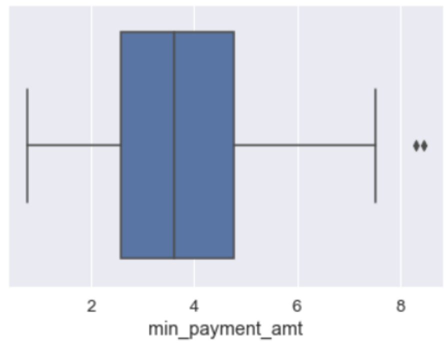


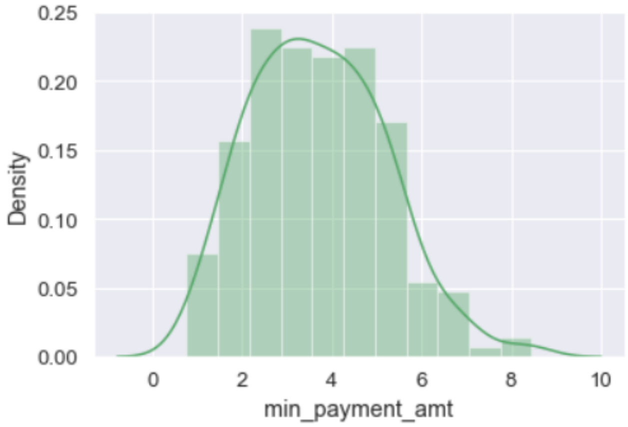
BoxPlot of credit_limit



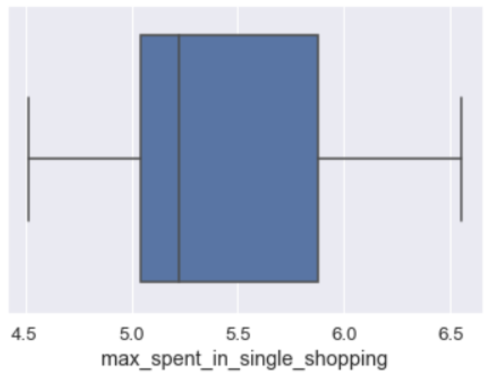


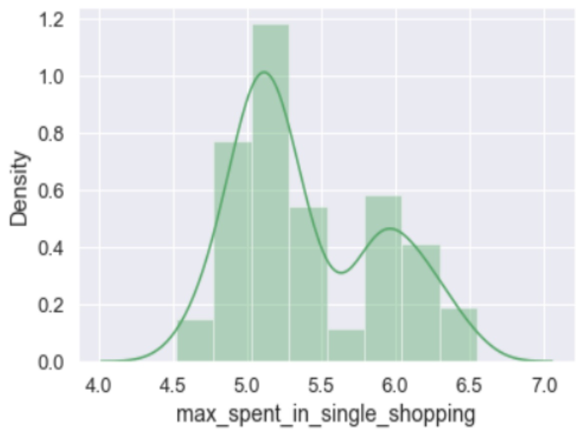
BoxPlot of min_payment_amt





BoxPlot of max_spent_in_single_shopping





From the above plots, only in min_payment_amt and probability_of_full_payment has the outliers.

I'm also checking for the lower limit, upper limit, IQR, and the percent outliers present or probability of the outlier present.

Spending:

```
spending - 1st Quartile (Q1) is:  12.27
spending - 3st Quartile (Q3) is:  17.305
Interquartile range (IQR) of spending is  5.035
Lower outliers in spending:  4.717499999999999
Upper outliers in spending:  24.8575

Number of outliers in spending upper :  0
Number of outliers in spending lower :  0
% of Outlier in spending upper:  0 %
% of Outlier in spending lower:  0 %
```

advance_payments

```
advance_payments - 1st Quartile (Q1) is:  13.45
advance_payments - 3st Quartile (Q3) is:  15.715
Interquartile range (IQR) of advance_payments is  2.2650000000000006
Lower outliers in advance_payments:  10.052499999999998
Upper outliers in advance_payments:  19.1125

Number of outliers in advance_payments upper :  0
Number of outliers in advance_payments lower :  0
% of Outlier in advance_payments upper:  0 %
% of Outlier in advance_payments lower:  0 %
```

probability_of_full_payment

```
probability_of_full_payment - 1st Quartile (Q1) is:  0.8569
probability_of_full_payment - 3st Quartile (Q3) is:  0.887775
Interquartile range (IQR) of probability_of_full_payment is  0.030874999999999986
Lower outliers in probability_of_full_payment:  0.8105875
Upper outliers in probability_of_full_payment:  0.9340875

Number of outliers in probability_of_full_payment upper :  0
Number of outliers in probability_of_full_payment lower :  3
% of Outlier in probability_of_full_payment upper:  0 %
% of Outlier in probability_of_full_payment lower:  1 %
```

current_balance

```
current_balance - 1st Quartile (Q1) is:  5.26225
current_balance - 3st Quartile (Q3) is:  5.97975
Interquartile range (IQR) of current_balance is  0.7175000000000002
Lower outliers in current_balance:  4.186
Upper outliers in current_balance:  7.056000000000001
```

```
Number of outliers in current_balance upper :  0
Number of outliers in current_balance lower :  0
% of Outlier in current_balance upper:  0 %
% of Outlier in current_balance lower:  0 %
```

credit_limit

```
credit_limit - 1st Quartile (Q1) is:  2.944
credit_limit - 3st Quartile (Q3) is:  3.56175
Interquartile range (IQR) of credit_limit is  0.61775
Lower outliers in credit_limit:  2.017375
Upper outliers in credit_limit:  4.488375
```

```
Number of outliers in credit_limit upper :  0
Number of outliers in credit_limit lower :  0
% of Outlier in credit_limit upper:  0 %
% of Outlier in credit_limit lower:  0 %
```

min_payment_amt

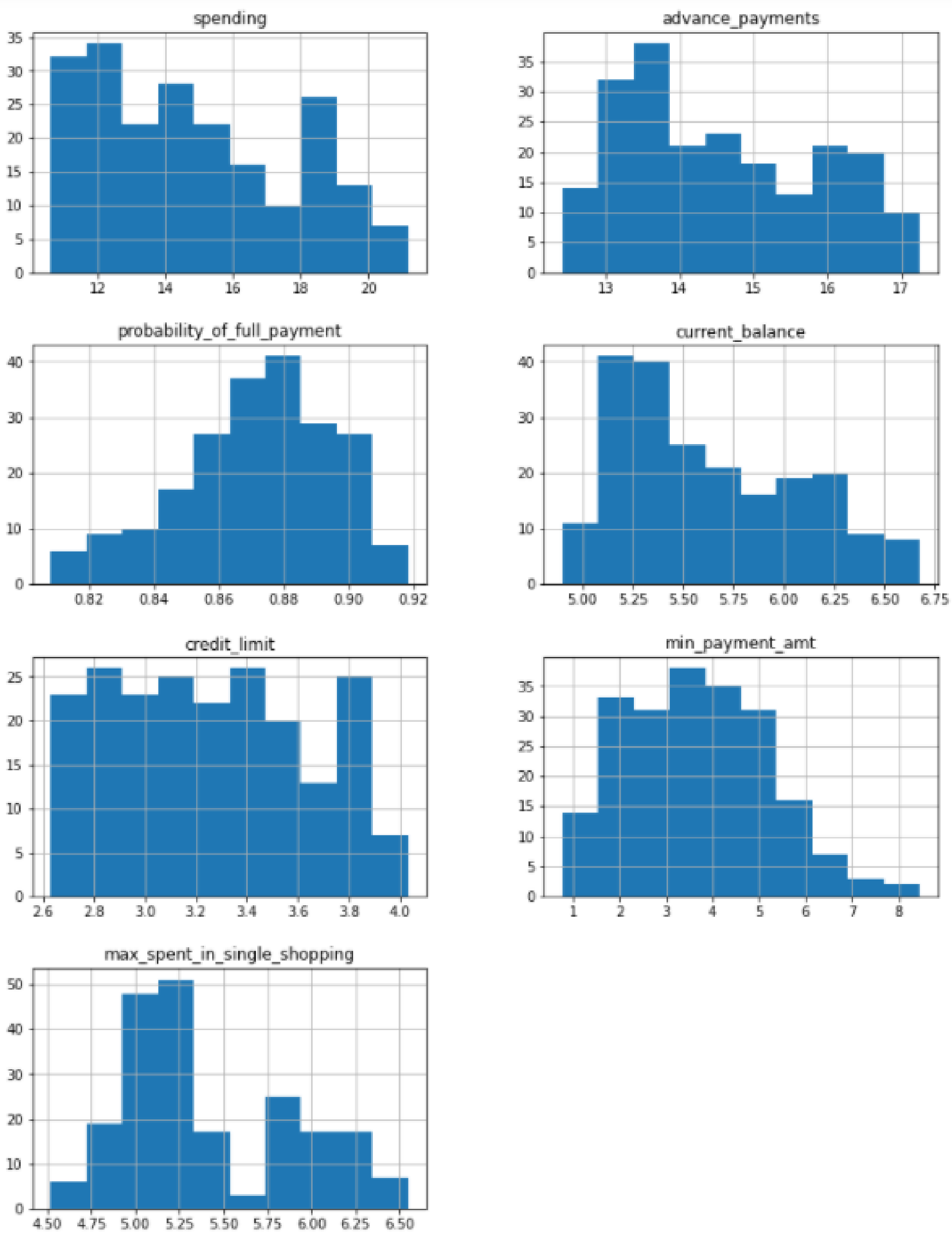
```
min_payment_amt - 1st Quartile (Q1) is:  2.5614999999999997
min_payment_amt - 3st Quartile (Q3) is:  4.76875
Interquartile range (IQR) of min_payment_amt is  2.20725
Lower outliers in min_payment_amt:  -0.7493750000000006
Upper outliers in min_payment_amt:  8.079625
```

```
Number of outliers in min_payment_amt upper :  2
Number of outliers in min_payment_amt lower :  0
% of Outlier in min_payment_amt upper:  1 %
% of Outlier in min_payment_amt lower:  0 %
```

max_spent_in_single_shopping

```
max_spent_in_single_shopping - 1st Quartile (Q1) is:  5.045
max_spent_in_single_shopping - 3st Quartile (Q3) is:  5.877000000000001
Interquartile range (IQR) of max_spent_in_single_shopping is  0.8320000000000007
Lower outliers in max_spent_in_single_shopping:  3.796999999999999
Upper outliers in max_spent_in_single_shopping:  7.125000000000002
```

```
Number of outliers in max_spent_in_single_shopping upper :  0
Number of outliers in max_spent_in_single_shopping lower :  0
% of Outlier in max_spent_in_single_shopping upper:  0 %
% of Outlier in max_spent_in_single_shopping lower:  0 %
```




```
bank.skew().sort_values(ascending=False)
```

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

```
import statistics
```

```
average=statistics.mean(bank['max_spent_in_single_shopping'])
average
```

```
5.408071428571429
```

```
average1=statistics.mean(bank['min_payment_amt'])
average1
```

```
3.7002009523809525
```

```
average2=statistics.mean(bank['credit_limit'])
average2
```

```
3.258604761904762
```

```
average3=statistics.mean(bank['current_balance'])
average3
```

```
5.628533333333333
```

```
average4=statistics.mean(bank['probability_of_full_payment'])
average4
```

```
0.8709985714285714
```

```
average5=statistics.mean(bank['spending'])
average5
```

```
14.847523809523809
```

```
average6=statistics.mean(bank['advance_payments'])
average6
```

```
14.559285714285714
```

Here we have outliers in two columns min_payment_amt and probability_of_full_payment upper as we have seen with the box plot and with the equation both.

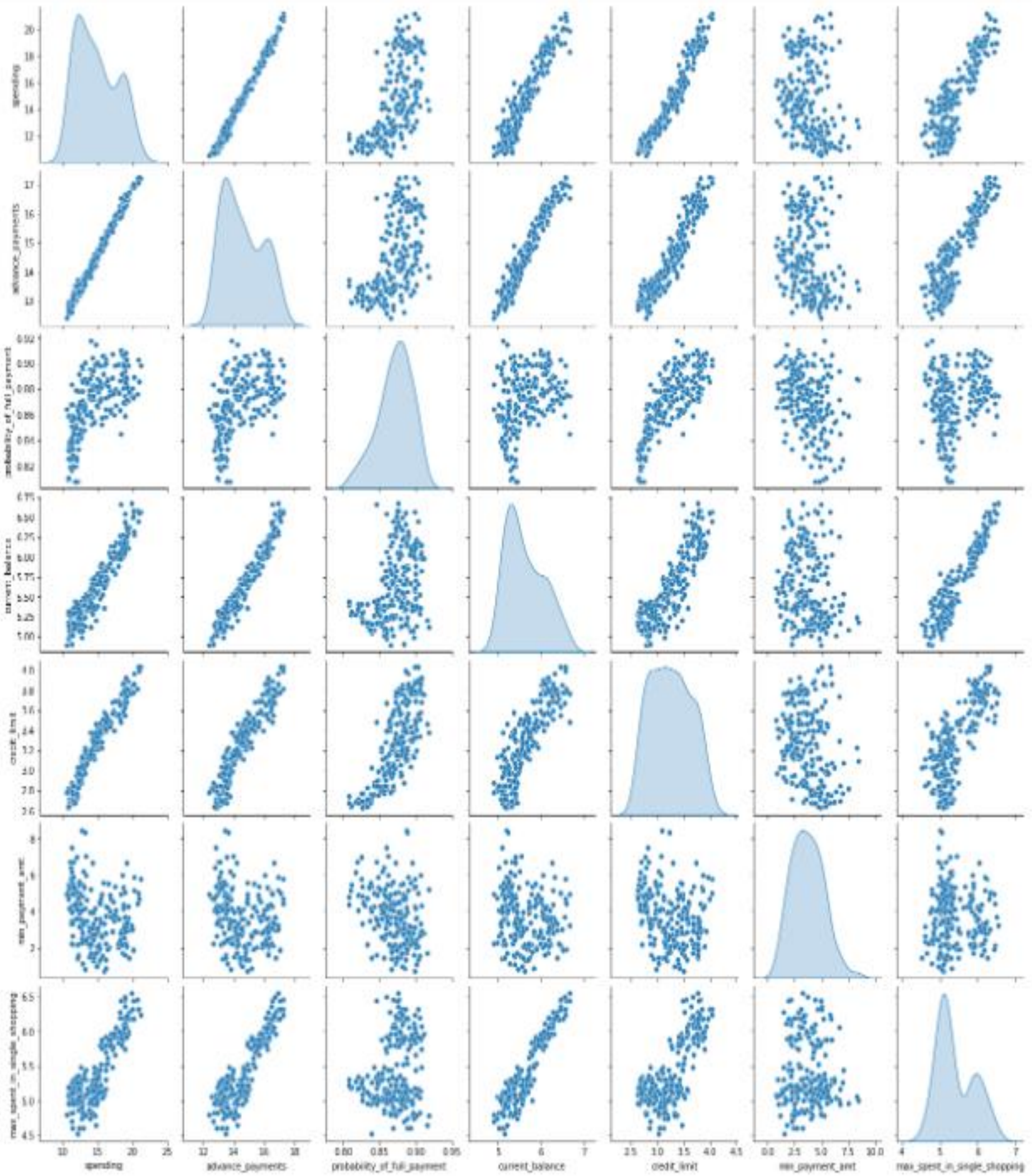
(Considering the amount in dollars)
Credit limit average is around 3.258(10000s)
max_spent_in_single_shopping average is around 5.408(1000s)
advance_payments average is around 14.559 (100s)
spending average is around 14.847 (1000s)
probability_of_full_payment average is around 87%
current_balance average is around 5.628 (1000s)
min_payment_amt average is around 3.700(100s)

Outlier in min_payment_amt upper: 1 %
Outlier in probability_of_full_payment lower: 1 %

Distribution is skewed to right tail for all the variable except probability_of_full_payment variable, which has left tail.

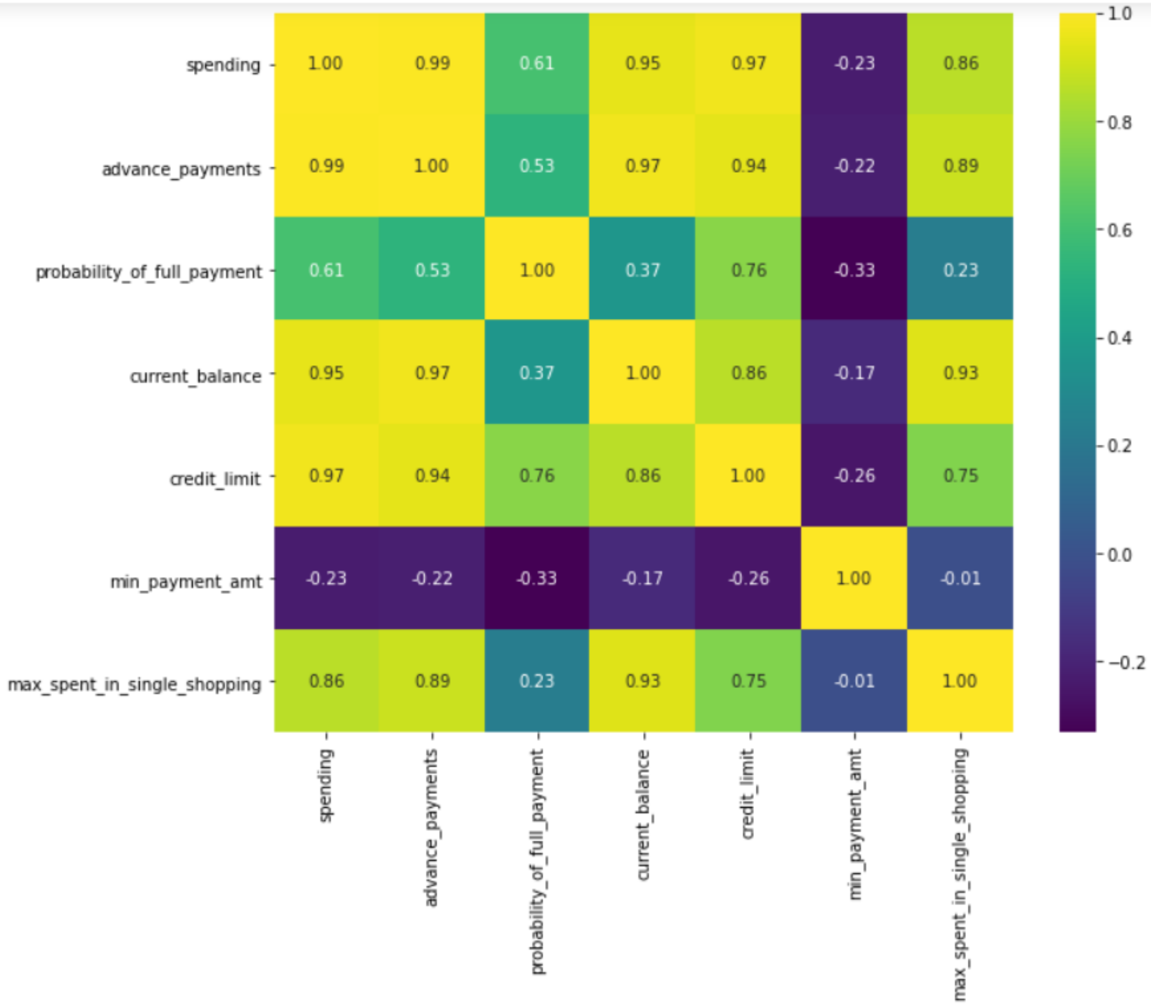
Multivariate analysis

Check for multicollinearity



```
bank.corr().T
```

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
spending	1.000000	0.994341	0.608288	0.949985	0.970771	-0.229572	0.863693
advance_payments	0.994341	1.000000	0.529244	0.972422	0.944829	-0.217340	0.890784
ability_of_full_payment	0.608288	0.529244	1.000000	0.367915	0.761635	-0.331471	0.226825
current_balance	0.949985	0.972422	0.367915	1.000000	0.860415	-0.171562	0.932806
credit_limit	0.970771	0.944829	0.761635	0.860415	1.000000	-0.258037	0.749131
min_payment_amt	-0.229572	-0.217340	-0.331471	-0.171562	-0.258037	1.000000	-0.011079
ent_in_single_shopping	0.863693	0.890784	0.226825	0.932806	0.749131	-0.011079	1.000000



Here we can see both negative and positive correlation. Listing just the strong positive correlation is between.

-spending and advance_payments
-spending and current_balance
-spending and credit_limit
-advance_payments and current_balance

As of now, for this we are not dropping the outlier values instead of dropping we will treat it with their respective medians, as mean gets affected by the outlier so, as I think median is the best option for treating it.

Only two variables have the outliers treating is the best option, so we will not loose the other relevant information which also seems important.

```
def treat_outlier(x):
    # taking 5,25,75 percentile of column
    q5= np.percentile(x,5)
    q25=np.percentile(x,25)
    q75=np.percentile(x,75)
    dt=np.percentile(x,95)
    #calculationg IQR range
    IQR=q75-q25
    #Calculating minimum threshold
    lower_bound=q25-(1.5*IQR)
    upper_bound=q75+(1.5*IQR)
    #Capping outliers
    return x.apply(lambda y: dt if y > upper_bound else y).apply(lambda y: q5 if y < lower_bound else y)
```

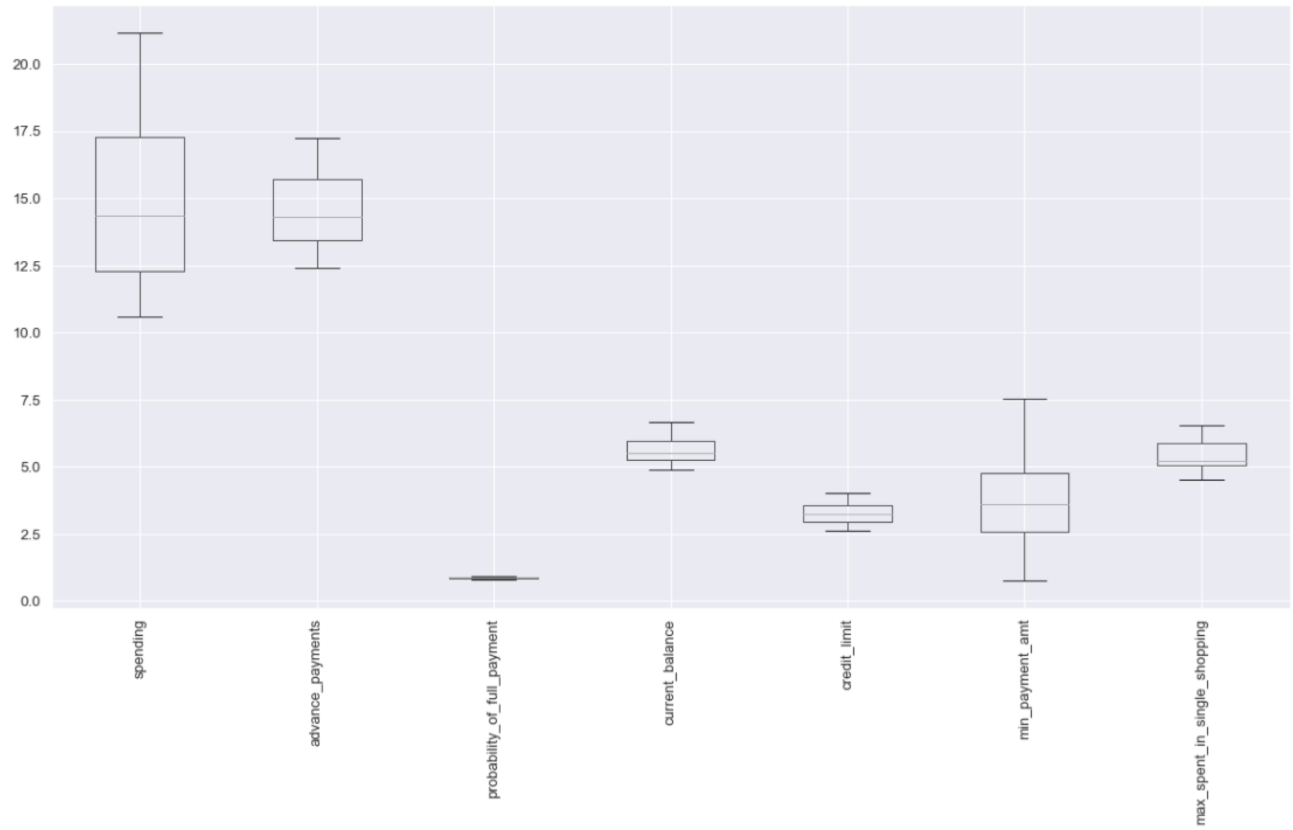
```
bank.head()
```

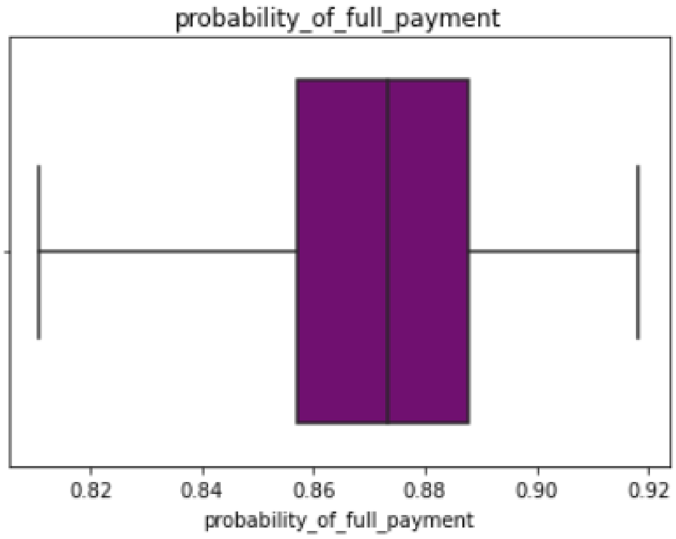
	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
0	19.94	16.92	0.8752	6.675	3.763	3.252	6.550
1	15.99	14.89	0.9064	5.363	3.582	3.336	5.144
2	18.95	16.42	0.8829	6.248	3.755	3.368	6.148
3	10.83	12.96	0.8099	5.278	2.641	5.182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

```
no_outlier = ['spending','advance_payments','current_balance','credit_limit','max_spent_in_single_shopping']
```

```
outlier_list = [x for x in df_num.columns if x not in no_outlier]
```

```
for i in df_num[outlier_list]:
    df_num[i]=treat_outlier(df_num[i])
```





Know most of the outliers have been treated, and our data is good to go for further analysis.

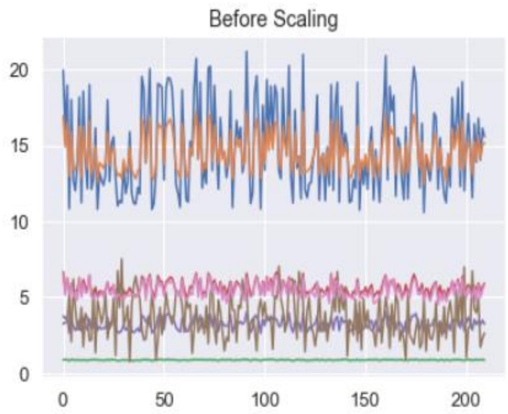
1.2- Do you think scaling is necessary for clustering in this case? Justify.

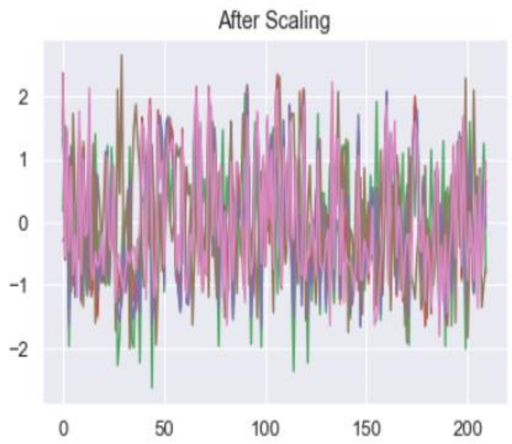
Scaling is needed to done as all variables have different values. Scaling will provide us all values with same range, that becomes more convenient for us. After scaling data become more cleaner or comes in proper manner for further analysis.

The standard normal distribution just converts the group of data in our frequency distribution such that the mean is 0 and standard deviation is 1. Normalization is used to eliminate redundant data and ensures that good quality clusters are generated which can improve the efficiency of clustering algorithms. So, it becomes essential step before clustering as Euclideandistance is very sensitive to the changes in the differences all dimensions are equally important.

Here I'm using z-score to standardize the data to relative same scale -3 to +3

<u>Data before and after scaling:</u>






```
from scipy.stats import zscore
df_num_scaled=df_num.apply(zscore)
df_num_scaled.head()
```

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
0	1.754355	1.811968	0.171955	2.367533	1.338579	-0.294861	2.328998
1	0.393582	0.253840	1.528129	-0.600744	0.858236	-0.236880	-0.538582
2	1.413300	1.428192	0.506652	1.401485	1.317348	-0.214791	1.509107
3	-1.384034	-1.227533	-1.970322	-0.793049	-1.639017	1.037338	-0.454961
4	1.082581	0.998364	1.215165	0.591544	1.155464	-1.112128	0.874813

Data looks much better after scaling.

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

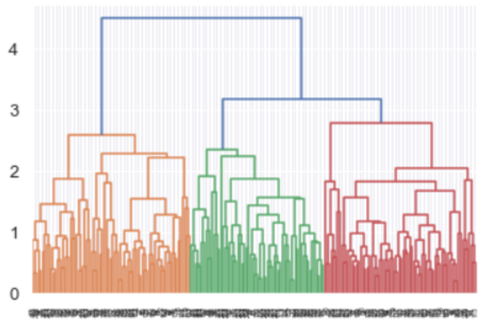
Here I'm using all the three approaches:

1- Linkage Method

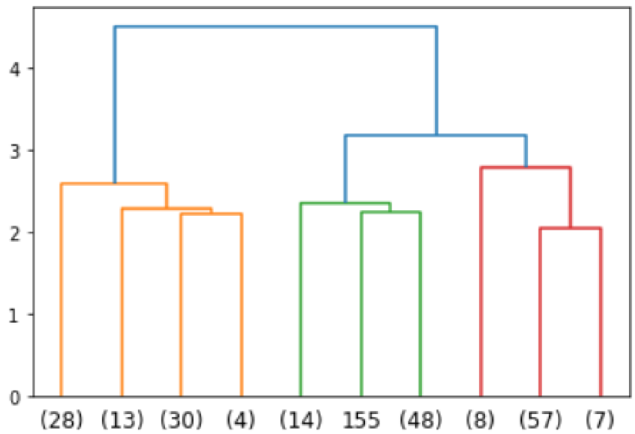
```
from scipy.cluster.hierarchy import dendrogram, linkage
```

```
link_method = linkage(df_num_scaled, method = 'average')
```

```
dend = dendrogram(link_method)
```




```
dend = dendrogram(link_method,
                  truncate_mode='lastp',
                  p = 10,
                  )
```



```
from scipy.cluster.hierarchy import fcluster
```

```
clusters_3 = fcluster(link_method, 3, criterion='maxclust')
clusters_3
```

```
array([1, 2, 1, 3, 1, 3, 3, 2, 1, 3, 1, 1, 3, 1, 2, 3, 2, 3, 3, 3, 3, 3,
       1, 3, 2, 1, 2, 3, 3, 3, 2, 3, 3, 2, 3, 3, 3, 3, 3, 1, 1, 2, 1, 1,
       3, 3, 2, 1, 1, 1, 3, 1, 1, 1, 1, 1, 3, 3, 3, 1, 2, 3, 3, 1, 2, 1,
       1, 2, 1, 2, 2, 3, 1, 1, 3, 1, 2, 3, 1, 2, 2, 2, 2, 1, 3, 1, 1, 1,
       1, 3, 3, 1, 2, 3, 2, 1, 1, 1, 3, 1, 3, 1, 2, 1, 2, 1, 1, 3, 3, 1,
       1, 2, 1, 3, 3, 1, 2, 3, 3, 1, 2, 3, 3, 3, 2, 2, 1, 3, 2, 2, 3, 2,
       2, 1, 3, 1, 1, 3, 1, 2, 3, 2, 3, 3, 2, 3, 1, 3, 2, 3, 2, 3, 2, 1,
       2, 2, 2, 3, 2, 1, 1, 3, 1, 1, 1, 3, 1, 2, 2, 3, 2, 3, 2, 1, 1, 1,
       2, 3, 2, 3, 2, 3, 2, 2, 1, 1, 2, 1, 2, 3, 2, 2, 3, 1, 2, 1, 1, 3,
       1, 3, 2, 2, 2, 3, 1, 2, 1, 2, 2, 1], dtype=int32)
```

```
cluster3_dataset=bank.copy()
```

```
cluster3_dataset['clusters-3'] = clusters_3
```

```
cluster3_dataset.head()
```

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

```
cluster3_dataset['clusters-3'].value_counts().sort_index()
```

```
1     75
2     63
3     72
Name: clusters-3, dtype: int64
```

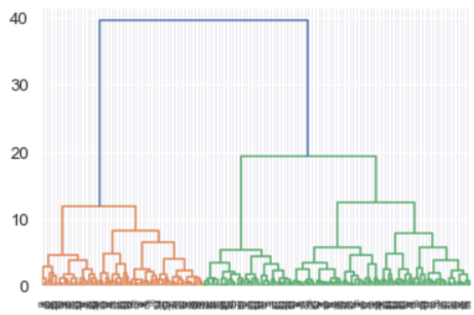
```
aggdata=cluster3_dataset.groupby('clusters-3').mean()
aggdata['Freq']=cluster3_dataset['clusters-3'].value_counts().sort_index()
aggdata
```

clusters-3	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	Freq
1	18.129200	16.058000	0.881595	6.135747	3.648120	3.650200	5.987040	75
2	14.167302	14.186190	0.882776	5.451381	3.236794	2.377956	5.048698	63
3	12.024306	13.324583	0.849656	5.255194	2.871944	4.909250	5.119431	72

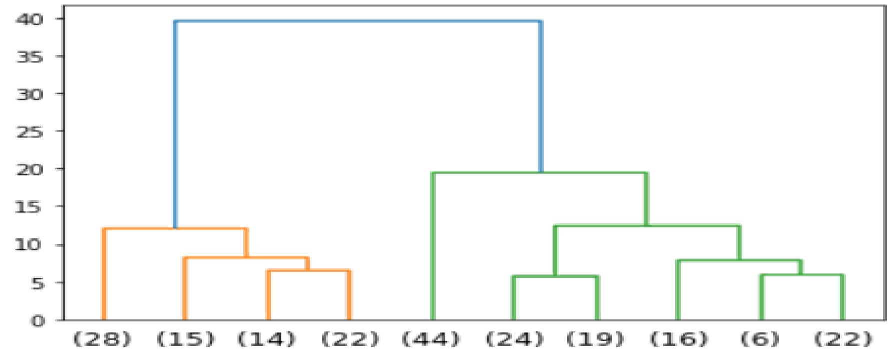
2-Ward Link Method:

```
wardlink = linkage(df_num_scaled, method = 'ward')
```

```
dend_wardlink = dendrogram(wardlink)
```



```
dend_wardlink = dendrogram(wardlink,
                           truncate_mode='lastp',
                           p = 10,
                           )
```




```
clusters_ward3 = fcluster(wardlink, 3, criterion='maxclust')
clusters_ward3
```

```
array([1, 2, 1, 3, 1, 3, 3, 2, 1, 3, 1, 1, 3, 1, 3, 3, 2, 3, 3, 3, 3, 3,
       1, 3, 2, 1, 3, 3, 3, 3, 2, 3, 3, 2, 3, 3, 3, 3, 3, 1, 1, 2, 1, 1,
       3, 3, 3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 3, 3, 3, 1, 2, 3, 3, 1, 2, 1,
       1, 2, 1, 3, 2, 3, 1, 1, 3, 1, 2, 3, 1, 2, 2, 3, 2, 1, 3, 1, 1, 1,
       1, 3, 3, 1, 2, 3, 3, 1, 1, 1, 3, 1, 3, 1, 2, 1, 2, 1, 1, 3, 3, 1,
       1, 2, 1, 3, 3, 1, 2, 3, 3, 1, 3, 3, 3, 3, 2, 2, 1, 3, 2, 2, 3, 2,
       3, 1, 3, 1, 1, 3, 1, 3, 1, 2, 3, 3, 2, 3, 1, 3, 2, 3, 2, 3, 2, 1,
       3, 2, 2, 3, 2, 1, 1, 3, 1, 1, 1, 3, 1, 2, 3, 3, 2, 3, 2, 1, 1, 1,
       2, 3, 1, 3, 2, 3, 3, 2, 1, 1, 3, 1, 3, 3, 3, 2, 3, 1, 2, 1, 1, 3,
       1, 3, 2, 1, 2, 3, 1, 3, 1, 2, 1, 1], dtype=int32)
```

```
clusters_ward3_dataset=bank.copy()
```

```
clusters_ward3_dataset['clusters_ward3'] = clusters_ward3
```

```
clusters_ward3_dataset.head()
```

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	clusters_ward3
0	19.94	16.92	0.8752	6.675	3.763	3.252	6.550	1
1	15.99	14.89	0.9064	5.363	3.582	3.336	5.144	2
2	18.95	16.42	0.8829	6.248	3.755	3.368	6.148	1
3	10.83	12.96	0.8099	5.278	2.641	5.182	5.185	3
4	17.99	15.86	0.8992	5.890	3.694	2.068	5.837	1

```
clusters_ward3_dataset['clusters_ward3'].value_counts().sort_index()
```

```
1    79
2    44
3    87
Name: clusters_ward3, dtype: int64
```

```
aggdata_ward=clusters_ward3_dataset.groupby('clusters_ward3').mean()
aggdata_ward['Freq']=clusters_ward3_dataset['clusters_ward3'].value_counts().sort_index()
aggdata_ward
```

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	Freq
clusters_ward3								
1	18.039367	16.011266	0.882377	6.117468	3.641975	3.627253	5.957266	79
2	14.582955	14.407045	0.882357	5.535318	3.283818	2.316775	5.109841	44
3	12.082989	13.317816	0.854922	5.231701	2.897736	4.466105	5.060207	87

3- Agglomerative Clustering:

```
from sklearn.cluster import AgglomerativeClustering
```

```
bank1=bank.copy()
cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='average')
Cluster_agglo=cluster.fit_predict(bank1.iloc[:,1:7])
print(Cluster_agglo)
```

```
[1 2 1 0 1 0 0 2 1 0 1 2 0 1 2 0 2 0 0 0 0 0 1 0 2 1 2 0 0 0 2 0 0 2 0 0 0
 0 0 1 1 2 1 1 0 0 2 1 1 1 0 1 1 1 1 1 0 0 0 1 2 0 0 1 2 1 1 2 1 0 2 0 1 1
 0 1 2 0 1 2 2 2 2 1 0 2 2 1 1 0 0 1 2 0 0 1 1 1 0 1 0 1 2 1 2 1 1 0 0 1 2
 2 1 0 0 1 2 0 0 1 2 0 0 0 2 2 1 0 2 2 0 2 0 1 0 1 1 0 1 2 1 2 0 0 2 0 1 0
 2 0 2 0 2 2 2 2 0 0 2 1 1 0 1 1 1 0 1 2 2 2 2 0 2 1 1 1 2 2 2 0 2 0 2 2 2
 1 2 2 2 0 2 2 0 1 2 1 1 0 1 0 2 2 2 0 1 2 1 2 2 2]
```

```
bank1["Agglo_CLusters"]=Cluster_agglo
```

```
bank1.columns
```

```
Index(['spending', 'advance_payments', 'probability_of_full_payment',
       'current_balance', 'credit_limit', 'min_payment_amt',
       'max_spent_in_single_shopping', 'Agglo_CLusters'],
      dtype='object')
```

```
agglo_data=bank1.groupby('Agglo_CLusters').mean()
agglo_data['Freq']=bank1.Agglo_CLusters.value_counts().sort_index()
agglo_data
```

Agglo_CLusters	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	Freq
0	11.996849	13.301781	0.850936	5.245301	2.873096	4.901534	5.103904	73
1	18.386471	16.158235	0.883600	6.164485	3.681779	3.747412	6.021471	68
2	14.375797	14.313913	0.879806	5.505797	3.249420	2.382699	5.125362	69

Here I have shown the results for all the approaches, we can see there is not much difference. As we know when we use different approaches minute difference/ minor variations occurs.

For cluster grouping based on dendrograms, we can say 3 looks good. It gives us the solution based on spending (high, medium, low).

We have cluster 1 as highest spending, cluster 2 as medium spending, cluster 3 as lowest spending in linkage and cluster 0 as lowest spending in Agglomerative.

1.4 Apply K-Means clustering on scaled data and determine optimum clusters. Apply elbow curve and silhouette score. Explain the results properly. Interpret and write inferences on the finalized clusters.

```
from sklearn.cluster import KMeans
```

```
k_means = KMeans(n_clusters = 1)
k_means.fit(df_num_scaled)
k_means.inertia_
```

```
1469.9999999999995
```

```
k_means = KMeans(n_clusters = 2)
k_means.fit(df_num_scaled)
k_means.inertia_
```

```
659.1308122335325
```

```
k_means = KMeans(n_clusters = 3)
k_means.fit(df_num_scaled)
k_means.inertia_
```

```
429.41517904599925
```

```
k_means = KMeans(n_clusters = 4)
k_means.fit(df_num_scaled)
k_means.inertia_
```

```
369.879109314745
```

```
k_means = KMeans(n_clusters = 5)
k_means.fit(df_num_scaled)
k_means.inertia_
```

```
322.1970030959652
```

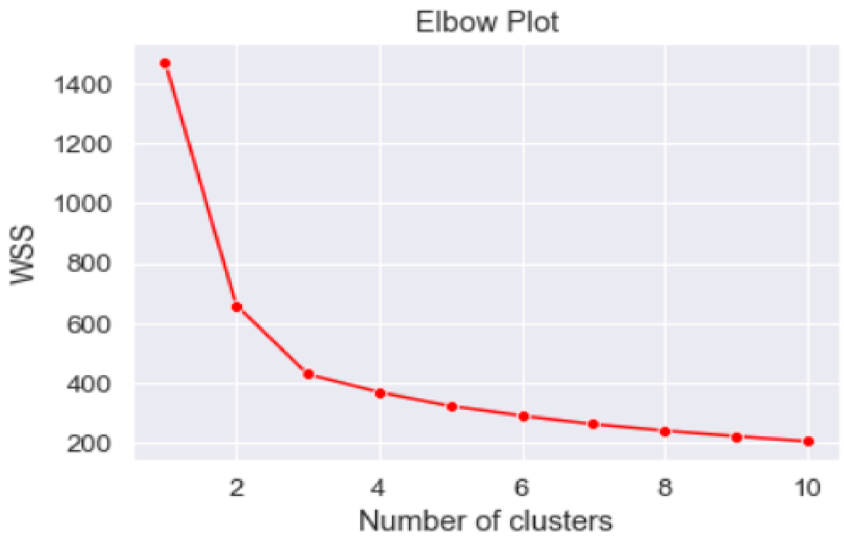
After cluster 3-4 there is a minimal drop in the values.

```
wss =[]
```

```
for i in range(1,11):
    KM = KMeans(n_clusters=i)
    KM.fit(df_num_scaled)
    wss.append(KM.inertia_)
```

```
wss
```

```
[1469.9999999999995,
 659.1308122335325,
 429.47914175239526,
 369.49394344842335,
 323.3945102124337,
 291.0580538558956,
 262.991243119072,
 241.00517359812707,
 222.57977813447366,
 205.87311597988543]
```




```
k_means_4 = KMeans(n_clusters = 4)
k_means_4.fit(df_num_scaled)
labels_4 = k_means_4.labels_
```

```
kmeans4_dataset=bank.copy()
```

```
kmeans4_dataset["Clus_kmeans"] = labels_4
kmeans4_dataset.head()
```

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

Silhouette score.

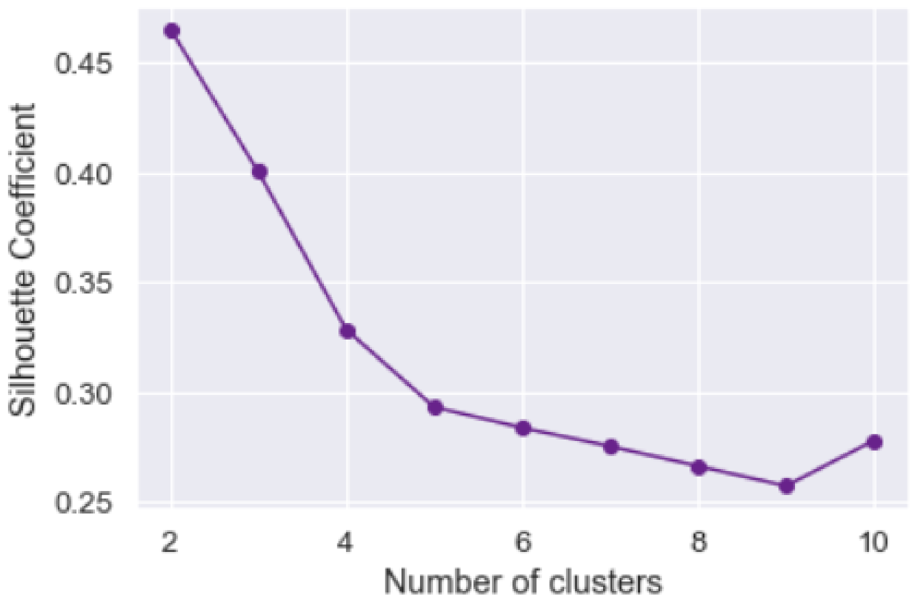
```
from sklearn import metrics
```

```
sil_scores = []
k_range = range(2, 11)
```

```
for k in k_range:
    km = KMeans(n_clusters=k, random_state=2)
    km.fit(df_num_scaled)
    sil_scores.append(metrics.silhouette_score(df_num_scaled, km.labels_))
```

```
sil_scores
```

```
[0.4653292734063001,
 0.40041910068777187,
 0.3285848463777896,
 0.29326753580142995,
 0.2838312563009059,
 0.27543009774981864,
 0.26640053752231424,
 0.2573332936643145,
 0.2780475608154224]
```



From the above graph and silhouette score 3-4 is optimal number of clustering.

```
sil_width = silhouette_samples(df_num_scaled,labels_4)
```

```
kmeans4_dataset["sil_width"] = sil_width
kmeans4_dataset.head()
```

spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	Clus_kmeans	sil_width
19.94	16.92	0.8752	6.675	3.763	3.252	6.550	0	0.468429
15.99	14.89	0.9064	5.363	3.582	3.336	5.144	2	0.053817
18.95	16.42	0.8829	6.248	3.755	3.368	6.148	0	0.487611
10.83	12.96	0.8099	5.278	2.641	5.182	5.185	1	0.593906
17.99	15.86	0.8992	5.890	3.694	2.068	5.837	0	0.155436

```
kmeans_3 = KMeans(n_clusters=3,random_state=123)
```

```
kmeans_3.fit(df_num_scaled)
kmeans_3.labels_
```

```
array([2, 0, 2, 1, 2, 1, 1, 0, 2, 1, 2, 0, 1, 2, 0, 1, 0, 1, 1, 1, 1, 1,
       2, 1, 0, 2, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 2, 2, 0, 2, 2,
       1, 1, 0, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 2, 0, 1, 1, 0, 0, 2,
       2, 0, 2, 1, 0, 1, 2, 2, 1, 2, 0, 1, 2, 0, 0, 0, 0, 2, 1, 0, 2, 0,
       2, 1, 0, 2, 0, 1, 1, 2, 2, 2, 1, 2, 0, 2, 0, 2, 0, 2, 2, 1, 1, 2,
       0, 0, 2, 1, 1, 2, 0, 0, 1, 2, 0, 1, 1, 1, 0, 0, 2, 1, 0, 0, 1, 0,
       0, 2, 1, 2, 2, 1, 2, 0, 0, 0, 1, 1, 0, 1, 2, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 2, 2, 1, 2, 2, 2, 1, 0, 0, 0, 1, 0, 1, 0, 2, 2, 2,
       0, 1, 0, 1, 0, 0, 0, 0, 2, 2, 1, 0, 0, 1, 0, 0, 1, 2, 0, 2, 2, 1,
       2, 1, 0, 2, 0, 1, 2, 0, 2, 0, 0, 0])
```

```
pd.Series(kmeans_3.labels_).value_counts()
```

```
0    72
1    71
2    67
dtype: int64
```

```
kmeansss_dataset=bank.copy()
```

```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(df_num_scaled)
```

```
y_kmeans1=y_kmeans
y_kmeans1=y_kmeans+1
cluster = pd.DataFrame(y_kmeans1)
kmeansss_dataset['cluster'] = cluster
kmeans_mean_cluster = pd.DataFrame(round(kmeansss_dataset.groupby('cluster').mean(),1))
kmeans_mean_cluster
```

cluster	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
1	14.4	14.3	0.9	5.5	3.3	2.7	5.1
2	18.4	16.2	0.9	6.2	3.7	3.6	6.0
3	11.9	13.3	0.8	5.2	2.8	4.8	5.1

```
cluster_3_T = kmeans_mean_cluster.T
```

cluster	1	2	3
spending	14.4	18.4	11.9
advance_payments	14.3	16.2	13.3
probability_of_full_payment	0.9	0.9	0.8
current_balance	5.5	6.2	5.2
credit_limit	3.3	3.7	2.8
min_payment_amt	2.7	3.6	4.8
max_spent_in_single_shopping	5.1	6.0	5.1

<u>Here I'm going with 3 group clustering via kmeans, as it makes sense based on spending pattern (high, medium, low).</u>

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

3 group cluster via Kmeans

cluster	1	2	3
spending	14.4	18.4	11.9
advance_payments	14.3	16.2	13.3
probability_of_full_payment	0.9	0.9	0.8
current_balance	5.5	6.2	5.2
credit_limit	3.3	3.7	2.8
min_payment_amt	2.7	3.6	4.8
max_spent_in_single_shopping	5.1	6.0	5.1

3 group cluster via hierarchical clustering

```
aggdata_ward.T
```

clusters_ward3	1	2	3
spending	18.039367	14.582955	12.082989
advance_payments	16.011266	14.407045	13.317816
probability_of_full_payment	0.882377	0.882357	0.854922
current_balance	6.117468	5.535318	5.231701
credit_limit	3.641975	3.283818	2.897736
min_payment_amt	3.627253	2.316775	4.466105
max_spent_in_single_shopping	5.957266	5.109841	5.060207
Freq	79.000000	44.000000	87.000000

Cluster Profile:

Group1: Highest Spending

Group2: Medium Spending

Group3: Lowest Spending.

Promotional strategies for different clusters:

Group1: Highest Spending Group

- Group 1 people are spending more money and also advance payment done is high as compared to other two clusters.so these people are the main target.
- As the advance payment is also high, Increase the credit limit, give loans on their credit cards, as they are the customers with good payment records.
- Giving reward points might attract them, and increase purchases.
- Also providing with the discounted offers on next transaction for one-time full payment will be beneficial, as max_spent_in_single_shopping is high.

Group2: Medium Spending Group

- These are potential target customers, who are paying bills, doing purchases and maintaining, good credit score. So, here we can increase the credit limit.
- Also providing some discounts / offers will increase the purchase.
- As from the cluster 3 group these set of people also have 2nd highest advanced payment done, here also we can recommend to give loans on their credit cards.

Group3: Lowest Spending Group

- Offers/discounts should be provided for early payment option.
- A gentle remainder for there payments regarding should be given.
- Also look for opportunities to cross-sell products to the customers, so as to increase the purchase.

Problem 2: CART-RF-ANN

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.

Attribute Information:

1. Target: Claim Status (Claimed)
2. Code of tour firm (Agency_Code)
3. Type of tour insurance firms (Type)
4. Distribution channel of tour insurance agencies (Channel)
5. Name of the tour insurance products (Product)
6. Duration of the tour (Duration)
7. Destination of the tour (Destination)
8. Amount of sales of tour insurance policies (Sales)
9. The commission received for tour insurance firm (Commission)
10. Age of insured (Age)

2.1 Read the data, do the necessary initial steps, and exploratory data analysis (Univariate, Bi-variate, and multivariate analysis).

```
insurance=pd.read_csv('insurance_part2_data.csv')
```

```
insurance.head()
```

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	Destination
0	48	C2B	Airlines	No	0.70	Online	7	2.51	Customised Plan	ASIA
1	36	EPX	Travel Agency	No	0.00	Online	34	20.00	Customised Plan	ASIA
2	39	CWT	Travel Agency	No	5.94	Online	3	9.90	Customised Plan	Americas
3	36	EPX	Travel Agency	No	0.00	Online	4	26.00	Cancellation Plan	ASIA
4	33	JZI	Airlines	No	6.30	Online	53	18.00	Bronze Plan	ASIA

```
insurance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 10 columns):
 #   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      3000 non-null   int64
 7   Sales         3000 non-null   float64
 8   Product Name  3000 non-null   object
 9   Destination   3000 non-null   object
dtypes: float64(2), int64(2), object(6)
memory usage: 234.5+ KB
```

```
insurance.shape
print('There are {} number of rows and {} number of columns'.format(insurance.shape[0],insurance.shape[1]))
```

```
There are 3000 number of rows and 10 number of columns
```

```
insurance.isnull().sum()
```

```
Age             0
Agency_Code     0
Type            0
Claimed         0
Commision       0
Channel         0
Duration        0
Sales           0
Product Name    0
Destination     0
dtype: int64
```

- There is total 3000 numbers of rows and 10 number of columns.
- No null entries present in it.
- Age, Commission, Duration, Sales have numeric datatypes, rest all have object datatype.
- There is total 9 independent variables and 1 target variable(claimed).

```
insurance.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Age	3000.0	38.091000	10.463518	8.0	32.0	36.00	42.000	84.00
Commision	3000.0	14.529203	25.481455	0.0	0.0	4.63	17.235	210.21
Duration	3000.0	70.001333	134.053313	-1.0	11.0	26.50	63.000	4580.00
Sales	3000.0	60.249913	70.733954	0.0	20.0	33.00	69.000	539.00

```
insurance.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Age	3000	NaN	NaN	NaN	38.091	10.4635	8	32	36	42	84
Agency_Code	3000	4	EPX	1365	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Type	3000	2	Travel Agency	1837	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Claimed	3000	2	No	2076	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Commision	3000	NaN	NaN	NaN	14.5292	25.4815	0	0	4.63	17.235	210.21
Channel	3000	2	Online	2954	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Duration	3000	NaN	NaN	NaN	70.0013	134.053	-1	11	26.5	63	4580
Sales	3000	NaN	NaN	NaN	60.2499	70.734	0	20	33	69	539
Product Name	3000	5	Customised Plan	1136	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Destination	3000	3	ASIA	2465	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Here, we have negative entries, which we can say might be a wrong entry.

Getting Unique Values For categorical variables

```
for column in insurance[['Agency_Code', 'Type', 'Claimed', 'Channel',
                  'Product Name', 'Destination']]:
    print(column.upper(),': ',insurance[column].nunique())
    print(insurance[column].value_counts().sort_values())
    print('\n')
```

```
AGENCY_CODE :  4
JZI     239
CWT     472
C2B     924
EPX    1365
Name: Agency_Code, dtype: int64


TYPE :  2
Airlines         1163
Travel Agency    1837
Name: Type, dtype: int64


CLAIMED :  2
Yes     924
No     2076
Name: Claimed, dtype: int64


CHANNEL :  2
Offline      46
Online     2954
Name: Channel, dtype: int64


PRODUCT NAME :  5
Gold Plan             109
Silver Plan           427
Bronze Plan           650
Cancellation Plan     678
Customised Plan      1136
Name: Product Name, dtype: int64


DESTINATION :  3
EUROPE      215
Americas    320
ASIA       2465
Name: Destination, dtype: int64
```

Check for duplicates:

```
dups = insurance.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
insurance[dups]
```

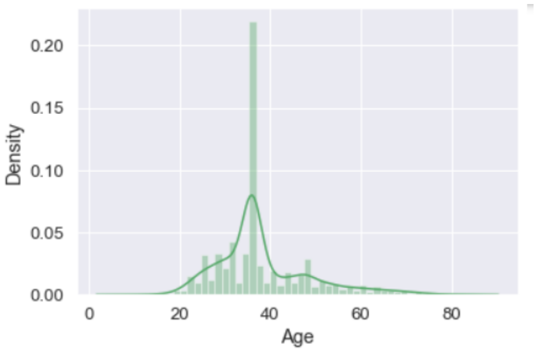
```
Number of duplicate rows = 139
```

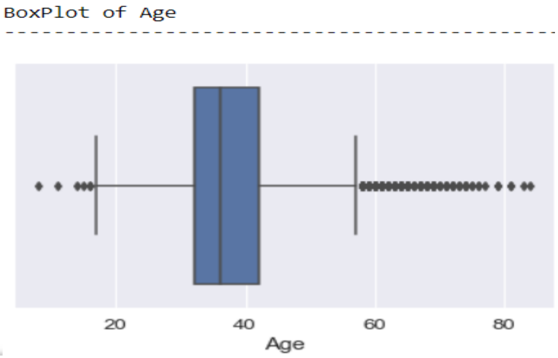
	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	Destination
63	30	C2B	Airlines	Yes	15.0	Online	27	60.0	Bronze Plan	ASIA
329	36	EPX	Travel Agency	No	0.0	Online	5	20.0	Customised Plan	ASIA
407	36	EPX	Travel Agency	No	0.0	Online	11	19.0	Cancellation Plan	ASIA
411	35	EPX	Travel Agency	No	0.0	Online	2	20.0	Customised Plan	ASIA
422	36	EPX	Travel Agency	No	0.0	Online	5	20.0	Customised Plan	ASIA
...	...	...	...	...	...	...	...	...	...	...
2940	36	EPX	Travel Agency	No	0.0	Online	8	10.0	Cancellation Plan	ASIA
2947	36	EPX	Travel Agency	No	0.0	Online	10	28.0	Customised Plan	ASIA
2952	36	EPX	Travel Agency	No	0.0	Online	2	10.0	Cancellation Plan	ASIA
2962	36	EPX	Travel Agency	No	0.0	Online	4	20.0	Customised Plan	ASIA
2984	36	EPX	Travel Agency	No	0.0	Online	1	20.0	Customised Plan	ASIA

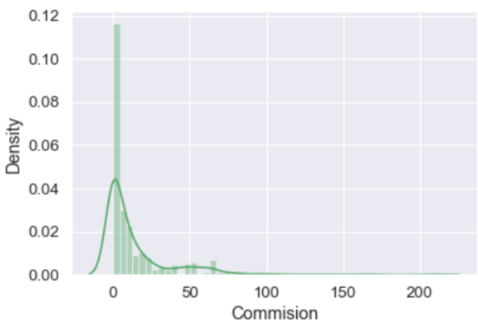
139 rows × 10 columns

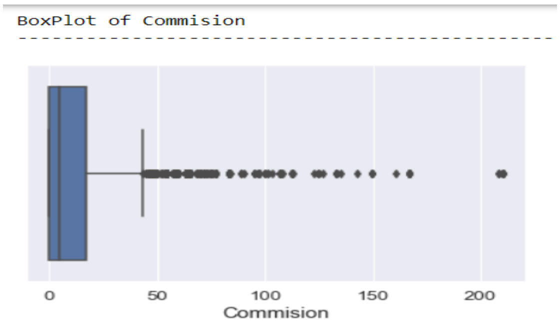
Though it shows there are 139 records, but it can be of different customers, there is no customer ID or any unique identifier, so I am not dropping them off.

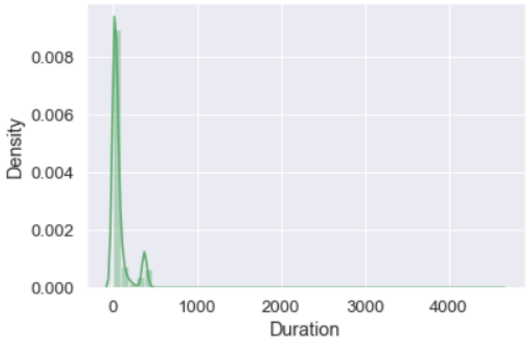
Univariate Analysis:

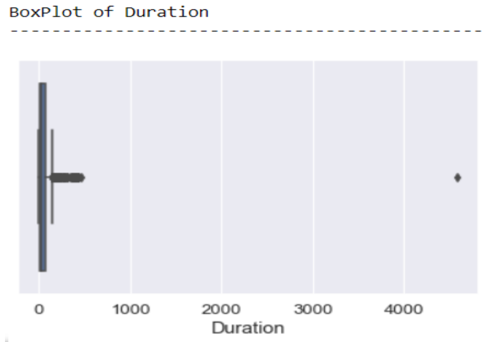


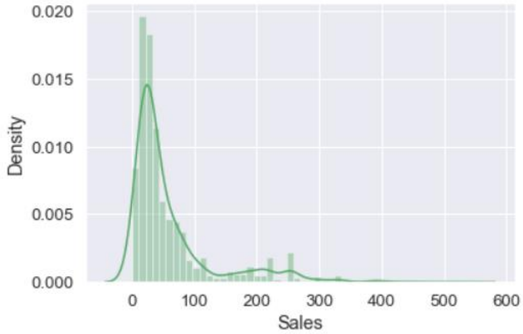


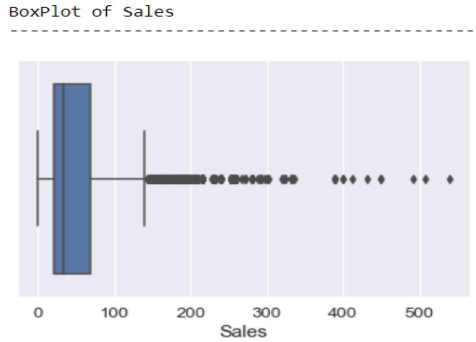












Here there is outliers in all variables , as sales and commission can have extrem values .

Random forest and Cart model can handel this , so not treating the ouliers now.

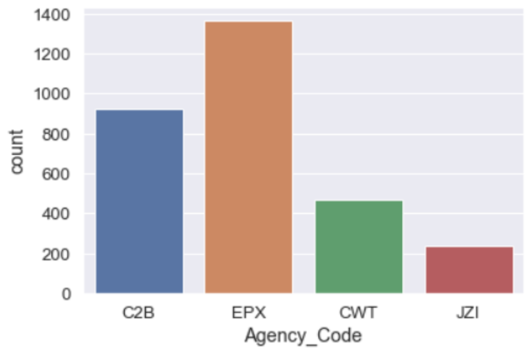
We will treat the outliers while ANN model.

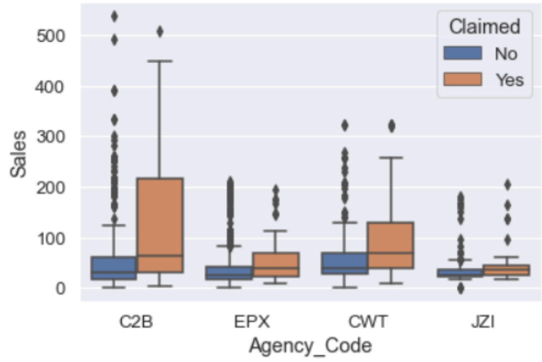
```
insurance.skew().sort_values(ascending=False)
```

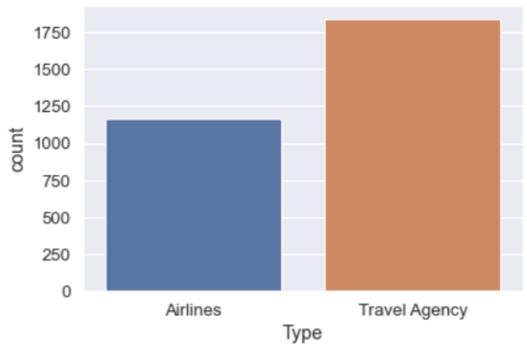
```
Duration      13.784681
Commision      3.148858
Sales          2.381148
Age            1.149713
dtype: float64
```

All the 4 variables are positively skeweed.

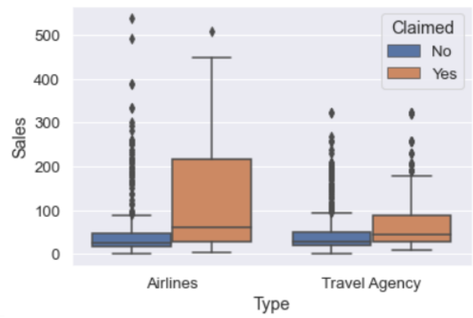
Categorical Variables:

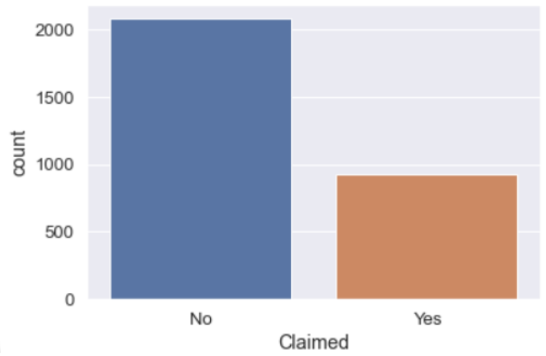




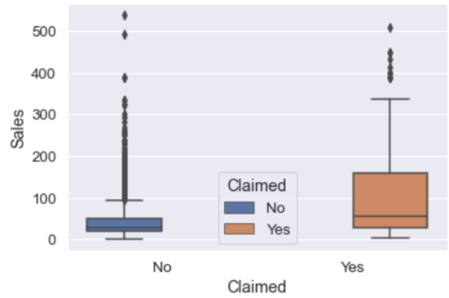


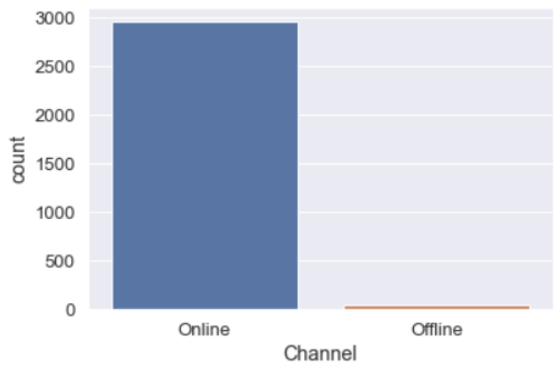
BoxPlot of Type



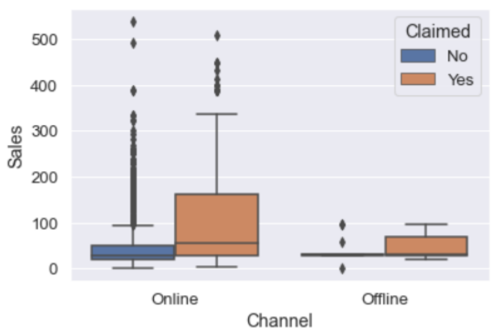


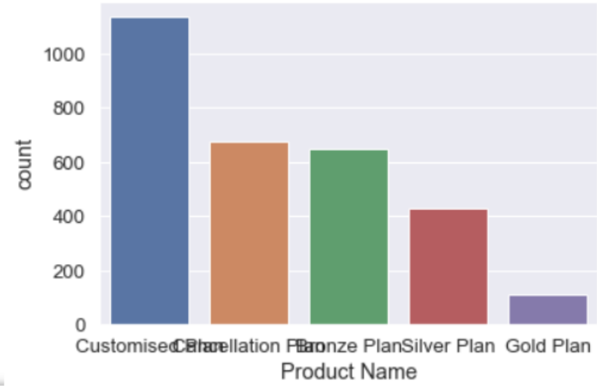
BoxPlot of Claimed

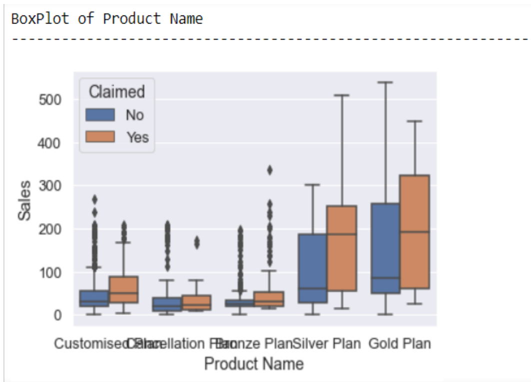


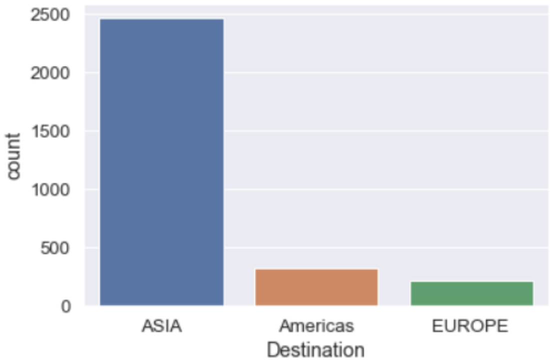


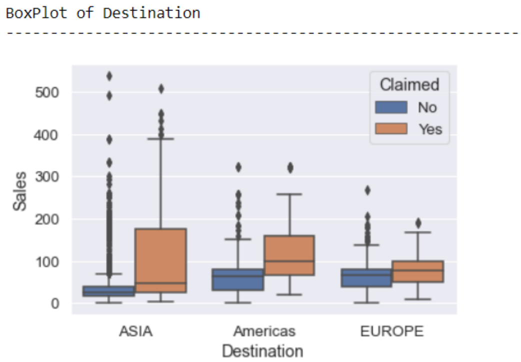
BoxPlot of Channel



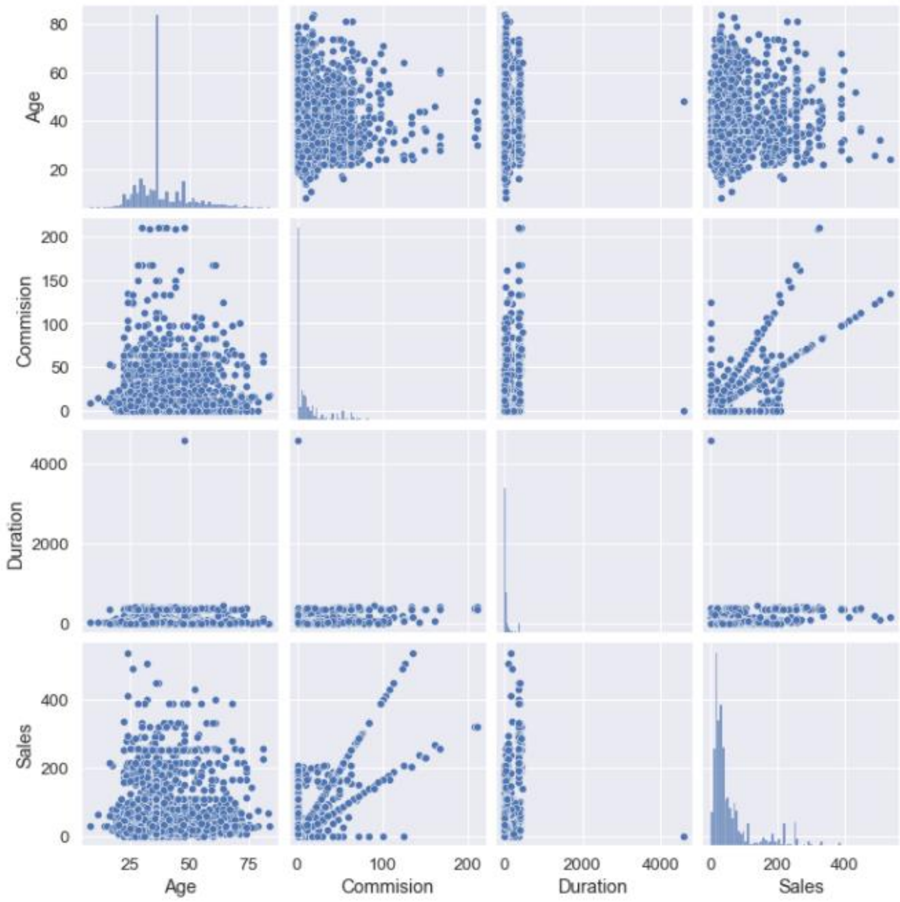




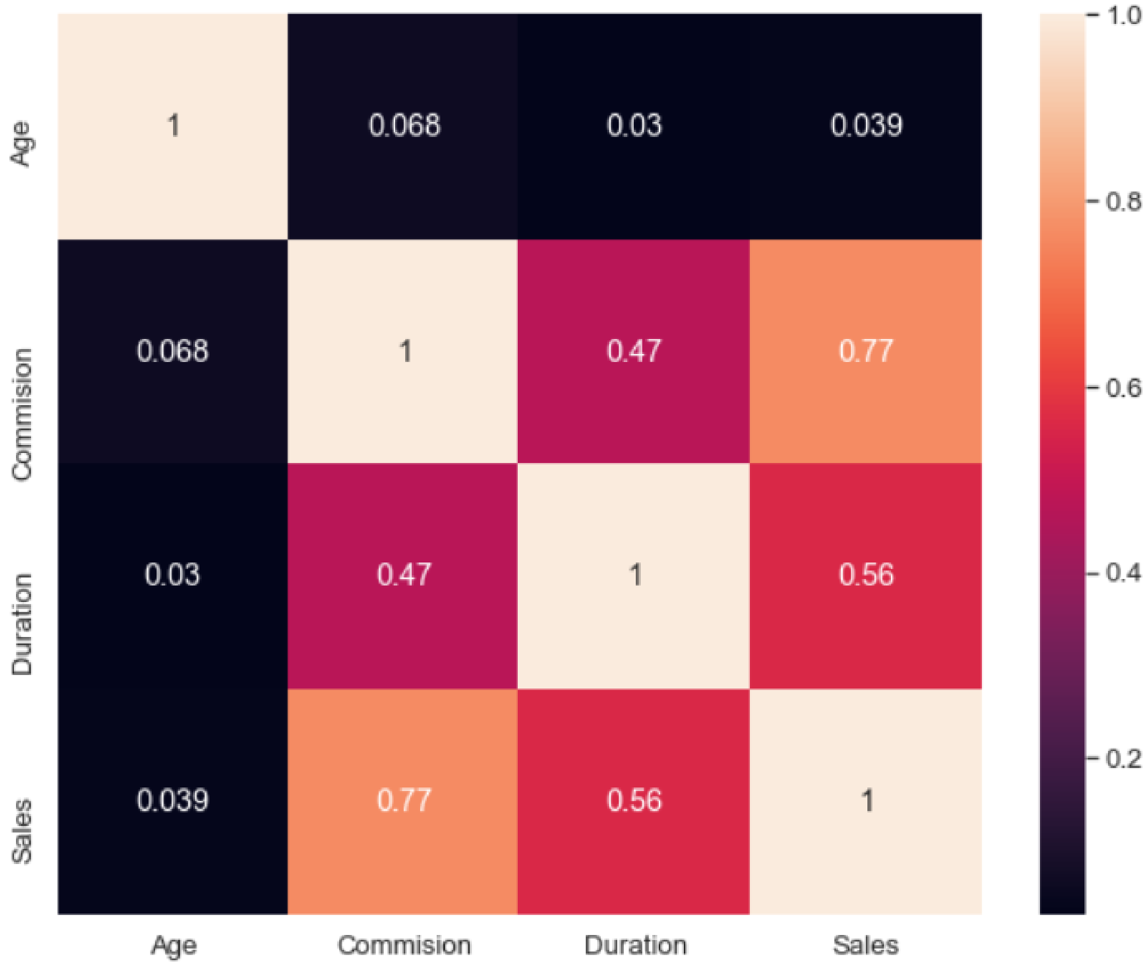




Checking pairwise distribution of the continuous variables:



Checking for Correlations:



Here we can say that there not strong correlation between the variables.

Just sales and commission have a correlation of 0.77.

As the sales increases, commission also increases.

Converting all objects to categorical codes:

```
for feature in insurance.columns:
    if insurance[feature].dtype == 'object':
        print('\n')
        print('feature:',feature)
        print(pd.Categorical(insurance[feature].unique()))
        print(pd.Categorical(insurance[feature].unique()).codes)
        insurance[feature] = pd.Categorical(insurance[feature]).codes
```

```
feature: Agency_Code
['C2B', 'EPX', 'CWT', 'JZI']
Categories (4, object): ['C2B', 'CWT', 'EPX', 'JZI']
[0 2 1 3]


feature: Type
['Airlines', 'Travel Agency']
Categories (2, object): ['Airlines', 'Travel Agency']
[0 1]


feature: Claimed
['No', 'Yes']
Categories (2, object): ['No', 'Yes']
[0 1]


feature: Channel
['Online', 'Offline']
Categories (2, object): ['Offline', 'Online']
[1 0]


feature: Product Name
['Customised Plan', 'Cancellation Plan', 'Bronze Plan', 'Silver Plan', 'Gold Plan']
Categories (5, object): ['Bronze Plan', 'Cancellation Plan', 'Customised Plan', 'Gold Plan', 'Silver Plan']
[2 1 0 4 3]


feature: Destination
['ASIA', 'Americas', 'EUROPE']
Categories (3, object): ['ASIA', 'Americas', 'EUROPE']
[0 1 2]
```

```
insurance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Age           3000 non-null   int64  
 1   Agency_Code   3000 non-null   int8   
 2   Type          3000 non-null   int8   
 3   Claimed       3000 non-null   int8   
 4   Commision     3000 non-null   float64
 5   Channel       3000 non-null   int8   
 6   Duration      3000 non-null   int64  
 7   Sales         3000 non-null   float64
 8   Product Name  3000 non-null   int8   
 9   Destination   3000 non-null   int8   
dtypes: float64(2), int64(2), int8(6)
memory usage: 111.5 KB
```

Again, checking the info (), all object datatypes are converted to numeric datatype(int).

```
insurance.head()
```

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	Destination
0	48	0	0	0	0.70	1	7	2.51	2	0
1	36	2	1	0	0.00	1	34	20.00	2	0
2	39	1	1	0	5.94	1	3	9.90	2	1
3	36	2	1	0	0.00	1	4	26.00	1	0
4	33	3	0	0	6.30	1	53	18.00	0	0

```
insurance.Claimed.value_counts(normalize=True)
```

```
0    0.692
1    0.308
Name: Claimed, dtype: float64
```

Checked for the proportion of 0s and 1s.

0=No and 1=Yes.

So here we have 69% of the data not claimed and 30% of the data with claimed.

```
insurance.skew().sort_values(ascending=False)
```

```
Duration        13.784681
Commision        3.148858
Sales            2.381148
Destination      2.188556
Age              1.149713
Claimed          0.832185
Product Name     0.432670
Agency_Code     -0.155126
Type            -0.461352
Channel         -7.892734
dtype: float64
```

After converting the datatypes to numeric again checking the skewness for all the variables.

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

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

(Answers for both the questions are given together)

```python
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
```

All the required libraries have been imported.

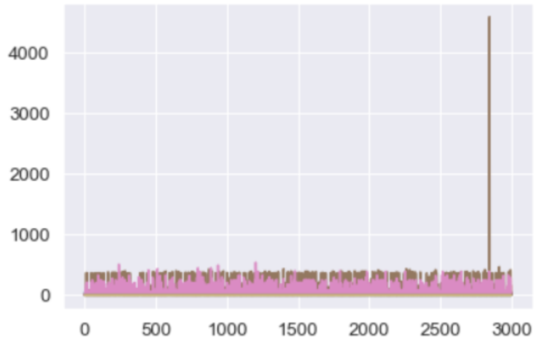
Extracting the target column into separate vectors for training set and test set

```python
x = insurance.drop("Claimed", axis=1)

y = insurance.pop("Claimed")

x.head()
```

	Age	Agency_Code	Type	Commision	Channel	Duration	Sales	Product Name	Destination
0	48	0	0	0.70	1	7	2.51	2	0
1	36	2	1	0.00	1	34	20.00	2	0
2	39	1	1	5.94	1	3	9.90	2	1
3	36	2	1	0.00	1	4	26.00	1	0
4	33	3	0	6.30	1	53	18.00	0	0

Data before scaling:

```python
plt.plot(x)
plt.show()
```

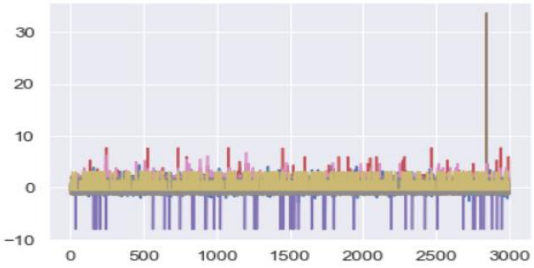



```python
from scipy.stats import zscore
x_scaled=x.apply(zscore)
x_scaled.head()
```

	Age	Agency_Code	Type	Commision	Channel	Duration	Sales	Product Name	Destination
0	0.947162	-1.314358	-1.256796	-0.542807	0.124788	-0.470051	-0.816433	0.268835	-0.434646
1	-0.199870	0.697928	0.795674	-0.570282	0.124788	-0.268605	-0.569127	0.268835	-0.434646
2	0.086888	-0.308215	0.795674	-0.337133	0.124788	-0.499894	-0.711940	0.268835	1.303937
3	-0.199870	0.697928	0.795674	-0.570282	0.124788	-0.492433	-0.484288	-0.525751	-0.434646
4	-0.486629	1.704071	-1.256796	-0.323003	0.124788	-0.126846	-0.597407	-1.320338	-0.434646

Data after scaling:

```
plt.plot(x_scaled)
plt.show()
```



Splitting data into training and test set:

```
x_train, x_test, train_labels, test_labels = train_test_split(x_scaled, y, test_size=.30, random_state=5)
```

```
#Checking the dimensions of the training and test data

print('x_train',x_train.shape)
print('x_test',x_test.shape)
print('train_labels',train_labels.shape)
print('test_labels',test_labels.shape)
```

```
x_train (2100, 9)
x_test (900, 9)
train_labels (2100,)
test_labels (900,)
```

Building a Decision Tree:

Checking for different parameters:

```
param_grid_dtcl = {
    'criterion': ['gini'],
    'max_depth': [10,20,30,50],
    'min_samples_leaf': [50,100,150],
    'min_samples_split': [150,300,450],
}

dtcl = DecisionTreeClassifier(random_state=1)

grid_search_dtcl = GridSearchCV(estimator = dtcl, param_grid = param_grid_dtcl, cv = 10)
```

```
grid_search_dtcl.fit(x_train, train_labels)
print(grid_search_dtcl.best_params_)
best_grid_dtcl = grid_search_dtcl.best_estimator_
best_grid_dtcl
```

```
{'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 50, 'min_samples_split': 450}

DecisionTreeClassifier(max_depth=10, min_samples_leaf=50, min_samples_split=450,
                       random_state=1)
```

```
param_grid_dtcl = {
    'criterion': ['gini'],
    'max_depth': [3, 5, 7, 10,12],
    'min_samples_leaf': [20,30,40,50,60],
    'min_samples_split': [150,300,450],
}

dtcl = DecisionTreeClassifier(random_state=1)

grid_search_dtcl = GridSearchCV(estimator = dtcl, param_grid = param_grid_dtcl, cv = 10)
```

```
grid_search_dtcl.fit(x_train, train_labels)
print(grid_search_dtcl.best_params_)
best_grid_dtcl = grid_search_dtcl.best_estimator_
best_grid_dtcl
```

```
{'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 20, 'min_samples_split': 150}

DecisionTreeClassifier(max_depth=5, min_samples_leaf=20, min_samples_split=150,
                       random_state=1)
```

```
param_grid_dtcl = {
    'criterion': ['gini'],
    'max_depth': [3.5,4.0,4.5, 5.0,5.5],
    'min_samples_leaf': [40, 42, 44,46,48,50,52,54],
    'min_samples_split': [250, 270, 280, 290, 300,310],
}

dtcl = DecisionTreeClassifier(random_state=1)

grid_search_dtcl = GridSearchCV(estimator = dtcl, param_grid = param_grid_dtcl, cv = 10)
```

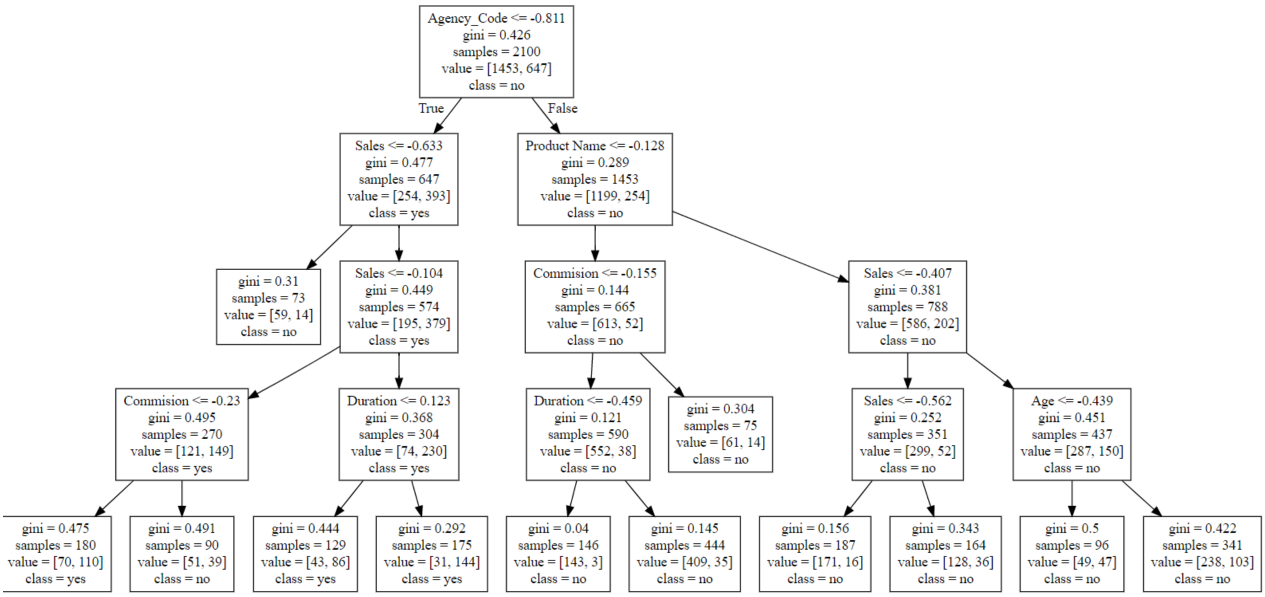
```
grid_search_dtcl.fit(x_train, train_labels)
print(grid_search_dtcl.best_params_)
best_grid_dtcl = grid_search_dtcl.best_estimator_
best_grid_dtcl
```

```
{'criterion': 'gini', 'max_depth': 3.5, 'min_samples_leaf': 44, 'min_samples_split': 250}

DecisionTreeClassifier(max_depth=3.5, min_samples_leaf=44,
                       min_samples_split=250, random_state=1)
```

Generating a Tree:

```
train_char_label = ['no', 'yes']
decision_tree_regularized = open('decision_tree_regularized.dot','w')
dot_data = tree.export_graphviz(best_grid_dtcl, out_file= decision_tree_regularized ,
                                feature_names = list(x_train),
                                class_names = list(train_char_label))

decision_tree_regularized.close()
dot_data
```

http://webgraphviz.com/



Variable Importance dtcl:

```
print (pd.DataFrame(best_grid_dtcl.feature_importances_, columns = ["Imp"], index = x_train.columns).sort_values('Imp',ascending
```

```
                  Imp
Agency_Code  0.634112
Sales        0.220899
Product Name 0.086632
Commision    0.021881
Age          0.019940
Duration     0.016536
Type         0.000000
Channel      0.000000
Destination  0.000000
```

looking at the above important parameters the model highly depends upon at "Agency Code" i.e.,63.41% and "Sales" i.e.,22%.

Predicting on Training and testing data

```
ytrain_predict_dtcl = best_grid_dtcl.predict(x_train)
ytest_predict_dtcl = best_grid_dtcl.predict(x_test)
```

Getting the Predicted Classes and Probs

```
ytest_predict_dtcl
ytest_predict_prob_dtcl=best_grid_dtcl.predict_proba(x_test)
ytest_predict_prob_dtcl
pd.DataFrame(ytest_predict_prob_dtcl).head()
```

	0	1
0	0.697947	0.302053
1	0.979452	0.020548
2	0.921171	0.078829
3	0.510417	0.489583
4	0.921171	0.078829

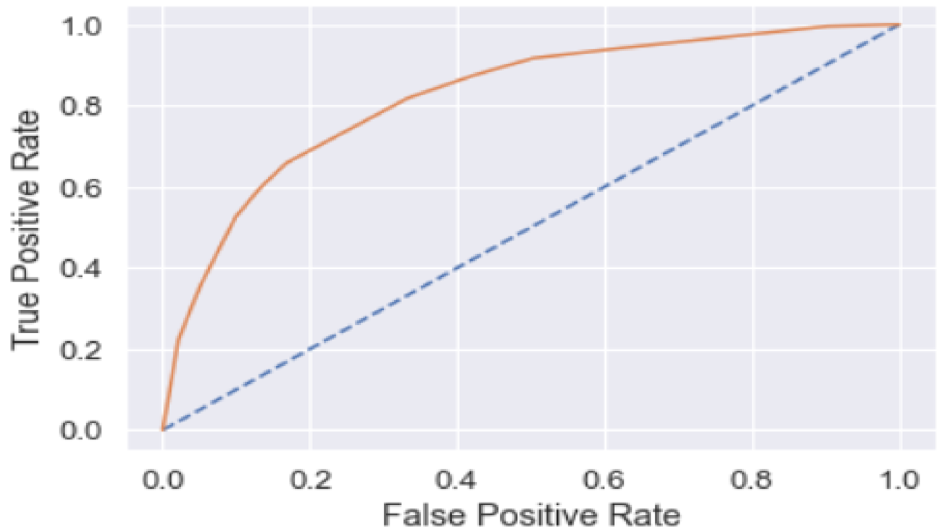
Model Evaluation

AUC and ROC for the training data

```
# predict probabilities
probs_cart = best_grid_dtcl.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs_cart = probs_cart[:, 1]
# calculate AUC
cart_train_auc = roc_auc_score(train_labels, probs_cart)
print('AUC: %.3f' % cart_train_auc)
# calculate roc curve
cart_train_fpr, cart_train_tpr, cart_train_thresholds = roc_curve(train_labels, probs_cart)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# plot the roc curve for the model
plt.plot(cart_train_fpr, cart_train_tpr)
```

AUC: 0.823

[<matplotlib.lines.Line2D at 0x264067efee0>]

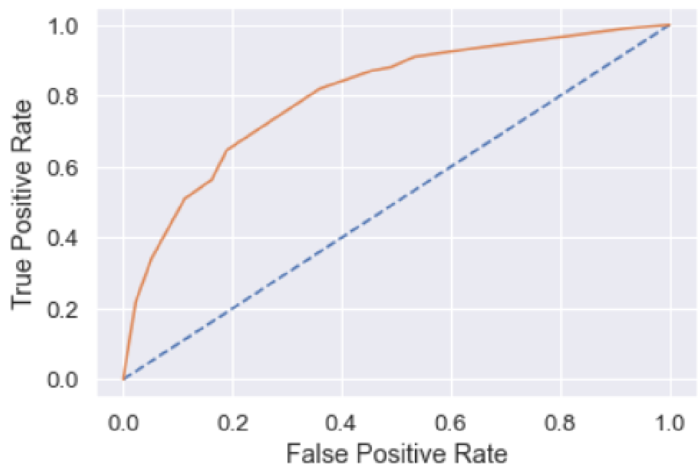


AUC and ROC for the Testing Data

```
# predict probabilities
probs_cart = best_grid_dtcl.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs_cart = probs_cart[:, 1]
# calculate AUC
cart_test_auc = roc_auc_score(test_labels, probs_cart)
print('AUC: %.3f' % cart_test_auc)
# calculate roc curve
cart_test_fpr, cart_test_tpr, cart_testthresholds = roc_curve(test_labels, probs_cart)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# plot the roc curve for the model
plt.plot(cart_test_fpr, cart_test_tpr)
```

AUC: 0.801

[<matplotlib.lines.Line2D at 0x2640660b6a0>]



Confusion Matrix for training data-dtcl

```
confusion_matrix(train_labels, ytrain_predict_dtcl)
```

```
array([[1309,  144],
       [ 307,  340]], dtype=int64)
```

```
#Training Data Accuracy
insurance_cart_train_acc=best_grid_dtcl.score(x_train,train_labels)
insurance_cart_train_acc
```

```
0.7852380952380953
```

```
print(classification_report(train_labels, ytrain_predict_dtcl))
```

```
              precision    recall  f1-score   support

           0       0.81      0.90      0.85      1453
           1       0.70      0.53      0.60       647

    accuracy                           0.79      2100
   macro avg       0.76      0.71      0.73      2100
weighted avg       0.78      0.79      0.78      2100
```

```
insurance_cart_metrics=classification_report(train_labels, ytrain_predict_dtcl,output_dict=True)
df=pd.DataFrame(insurance_cart_metrics).transpose()
insurance_cart_train_f1=round(df.loc["1"][2],2)
insurance_cart_train_recall=round(df.loc["1"][1],2)
insurance_cart_train_precision=round(df.loc["1"][0],2)
print ('insurance_cart_train_precision ',insurance_cart_train_precision)
print ('insurance_cart_train_recall ',insurance_cart_train_recall)
print ('insurance_cart_train_f1 ',insurance_cart_train_f1)
```

```
insurance_cart_train_precision  0.7
insurance_cart_train_recall  0.53
insurance_cart_train_f1  0.6
```

Confusion Matrix for test data-dtcl

```
confusion_matrix(test_labels, ytest_predict_dtcl)
```

```
array([[553,  70],
       [136, 141]], dtype=int64)
```

```
#Test Data Accuracy
insurance_cart_test_acc=best_grid_dtcl.score(x_test,test_labels)
insurance_cart_test_acc
```

```
0.7711111111111111
```

```
print(classification_report(test_labels, ytest_predict_dtcl))
```

```
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       623
           1       0.67      0.51      0.58       277

    accuracy                           0.77       900
   macro avg       0.74      0.70      0.71       900
weighted avg       0.76      0.77      0.76       900
```

```
insurance_cart_metrics_test=classification_report(test_labels, ytest_predict_dtcl,output_dict=True)
df=pd.DataFrame(insurance_cart_metrics_test).transpose()
insurance_cart_test_f1=round(df.loc["1"][2],2)
insurance_cart_test_recall=round(df.loc["1"][1],2)
insurance_cart_test_precision=round(df.loc["1"][0],2)
print ('insurance_cart_test_precision ',insurance_cart_test_precision)
print ('insurance_cart_test_recall ',insurance_cart_test_recall)
print ('insurance_cart_test_f1 ',cart_test_f1)
```

```
insurance_cart_test_precision  0.67
insurance_cart_test_recall  0.51
insurance_cart_test_f1  0.58
```

Cart Conclusion:

Train Data:

AUC:82%

Accuracy:79%

Precision:70%

F1-score:60%

Test Data:

AUC:80%

Accuracy:77%

Precision:67%

F1-score:58%

Training and Test set results are almost similar, and with the overall measures high, the model is a good model. Agency_code is the most important variable for predicting insurance claimed.

Building a Random Forest Classifier:

```
rfcl=RandomForestClassifier(n_estimators=500,
                            oob_score=True,
                            max_depth=10,
                            max_features=5,
                            min_samples_leaf=21,
                            min_samples_split=60)
```

```
rfcl.fit(x_train,train_labels)
```

```
RandomForestClassifier(max_depth=10, max_features=5, min_samples_leaf=21,
                       min_samples_split=60, n_estimators=500, oob_score=True)
```

```
rfcl.oob_score_
```

```
0.7823809523809524
```

```
param_grid={'n_estimators':[301,501,450],
 'max_depth':[10,20],
 'min_samples_leaf':[21,22],
 'min_samples_split':[60,70],
 'max_features':[5,6],
          }
```

```
rfcl=RandomForestClassifier()
```

```
grid_search=GridSearchCV(estimator=rfcl,param_grid=param_grid,cv=3)
```

```
grid_search.fit(x_train,train_labels)
```

```
GridSearchCV(cv=3, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [10, 20], 'max_features': [5, 6],
                         'min_samples_leaf': [21, 22],
                         'min_samples_split': [60, 70],
                         'n_estimators': [301, 501, 450]})
```

```
grid_search.best_params_
```

```
{'max_depth': 10,
 'max_features': 6,
 'min_samples_leaf': 22,
 'min_samples_split': 60,
 'n_estimators': 501}
```

```
best_grid_rfcl=grid_search.best_estimator_
```

```
best_grid_rfcl
```

```
RandomForestClassifier(max_depth=10, max_features=6, min_samples_leaf=22,
                       min_samples_split=60, n_estimators=501)
```

Predicting the Training and Testing data:

```
ytrain_predict_rfcl = best_grid_rfcl.predict(x_train)
ytest_predict_rfcl = best_grid_rfcl.predict(x_test)
```

```
ytest_predict_rfcl
ytest_predict_prob_rfcl=best_grid_rfcl.predict_proba(x_test)
ytest_predict_prob_rfcl
pd.DataFrame(ytest_predict_prob_rfcl).head()
```

	0	1
0	0.764837	0.235163
1	0.992648	0.007352
2	0.885095	0.114905
3	0.570183	0.429817
4	0.869180	0.130820

```
# Variable Importance via RF
print (pd.DataFrame(best_grid_rfcl.feature_importances_,
                    columns = ["Imp"],
                    index = x_train.columns).sort_values('Imp',ascending=False))
```

```
                   Imp
Agency_Code   0.390246
Product Name  0.208315
Sales         0.176787
Commision     0.090872
Duration      0.072130
Age           0.041476
Type          0.014091
Destination   0.005327
Channel       0.000756
```

RF Model Performance Evaluation on Training data:

```
confusion_matrix(train_labels,ytrain_predict_rfcl)
```

```
array([[1296,  157],
       [ 261,  386]], dtype=int64)
```

```
rf_train_acc=best_grid_rfcl.score(x_train,train_labels)
rf_train_acc
```

```
0.8009523809523809523
```

```
print(classification_report(train_labels,ytrain_predict_rfcl))
```

```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1453
           1       0.71      0.60      0.65       647

    accuracy                           0.80      2100
   macro avg       0.77      0.74      0.75      2100
weighted avg       0.79      0.80      0.80      2100
```

```
rf_metrics=classification_report(train_labels, ytrain_predict_rfcl,output_dict=True)
df=pd.DataFrame(rf_metrics).transpose()
rf_train_precision=round(df.loc["1"][0],2)
rf_train_recall=round(df.loc["1"][1],2)
rf_train_f1=round(df.loc["1"][2],2)
print ('rf_train_precision ',rf_train_precision)
print ('rf_train_recall ',rf_train_recall)
print ('rf_train_f1 ',rf_train_f1)
```

```
rf_train_precision  0.71
rf_train_recall  0.6
rf_train_f1  0.65
```

```
rf_train_fpr, rf_train_tpr,_=roc_curve(train_labels,best_grid_rfcl.predict_proba(x_train)[:,1])
plt.plot(rf_train_fpr,rf_train_tpr,color='blue')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
rf_train_auc=roc_auc_score(train_labels,best_grid_rfcl.predict_proba(x_train)[:,1])
print('Area under Curve is', rf_train_auc)
```

```
Area under Curve is 0.8612474749784862
```

RF Model Performance Evaluation on Test data:

```
confusion_matrix(test_labels,ytest_predict_rfcl)
```

```
array([[547,  76],
       [129, 148]], dtype=int64)
```

```
rf_test_acc=best_grid_rfcl.score(x_test,test_labels)
rf_test_acc
```

```
0.7722222222222223
```

```
print(classification_report(test_labels,ytest_predict_rfcl))
```

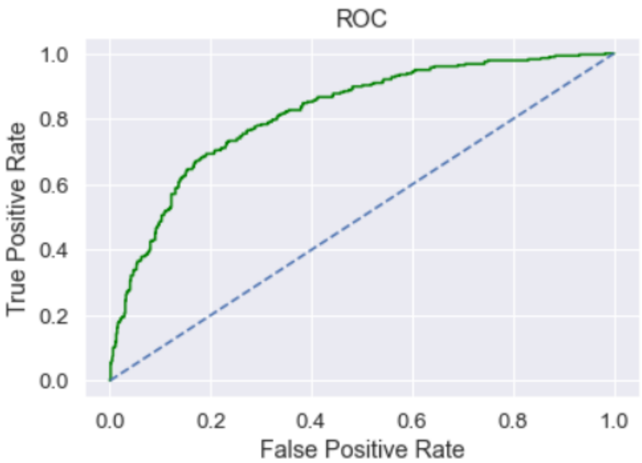
```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84       623
           1       0.66      0.53      0.59       277

    accuracy                           0.77       900
   macro avg       0.73      0.71      0.72       900
weighted avg       0.76      0.77      0.76       900
```

```
rf_metrics=classification_report(test_labels, ytest_predict_rfcl,output_dict=True)
df=pd.DataFrame(rf_metrics).transpose()
rf_test_precision=round(df.loc["1"][0],2)
rf_test_recall=round(df.loc["1"][1],2)
rf_test_f1=round(df.loc["1"][2],2)
print ('rf_test_precision ',rf_test_precision)
print ('rf_test_recall ',rf_test_recall)
print ('rf_test_f1 ',rf_test_f1)
```

```
rf_test_precision  0.66
rf_test_recall  0.53
rf_test_f1  0.59
```

```
rf_test_fpr, rf_test_tpr,_=roc_curve(test_labels,best_grid_rfcl.predict_proba(x_test)[:,1])
plt.plot(rf_test_fpr,rf_test_tpr,color='green')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
rf_test_auc=roc_auc_score(test_labels,best_grid_rfcl.predict_proba(x_test)[:,1])
print('Area under Curve is', rf_test_auc)
```

```
Area under Curve is 0.8178807563263816
```



Random Forest Conclusion

Train Data:

AUC:86%

Accuracy:80%

Precision:71%

F1-score:65%

Test Data:

AUC:82%

Accuracy:77%

Precision:66%

F1-score:59%

Training and Test set results are almost similar, and with the overall measures high, the model is a good model. Agency_code is the most important variable for predicting insurance claimed.

Building a Neural Network Classifier:

```
param_grid_nncl = {
    'hidden_layer_sizes': [50,100,200],
    'max_iter': [2500,3000,4000],
    'solver': ['adam'],
    'tol': [0.01],
}

nncl = MLPClassifier(random_state=1)

grid_search_nncl = GridSearchCV(estimator = nncl, param_grid = param_grid_nncl, cv = 10)
```

```
grid_search_nncl.fit(x_train, train_labels)
grid_search_nncl.best_params_
best_grid_nncl = grid_search_nncl.best_estimator_
best_grid_nncl
```

```
MLPClassifier(hidden_layer_sizes=200, max_iter=2500, random_state=1, tol=0.01)
```

Predicting the Training and Testing data:

```
ytrain_predict_nncl = best_grid_nncl.predict(x_train)
ytest_predict_nncl = best_grid_nncl.predict(x_test)
```

```
ytest_predict_nncl
ytest_predict_prob_nncl=best_grid_nncl.predict_proba(x_test)
ytest_predict_prob_nncl
pd.DataFrame(ytest_predict_prob_nncl).head()
```

	0	1
0	0.822676	0.177324
1	0.933407	0.066593
2	0.918772	0.081228
3	0.688933	0.311067
4	0.913425	0.086575

```
confusion_matrix(train_labels,ytrain_predict_nncl)
```

```
array([[1298,  155],
       [ 315,  332]], dtype=int64)
```

```
nncl_train_acc=best_grid_nncl.score(x_train,train_labels)
nncl_train_acc
```

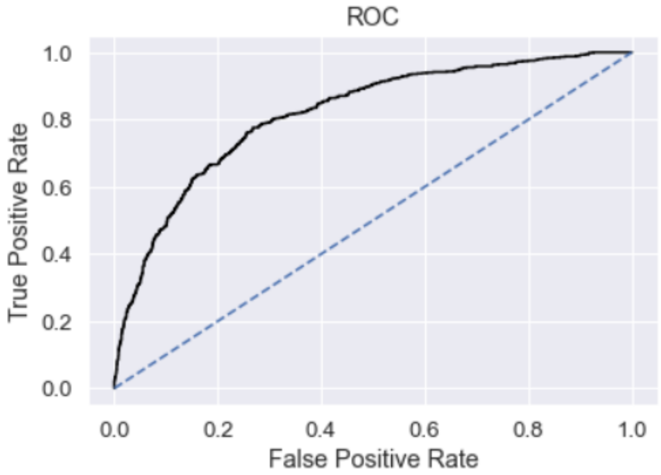
```
0.7761904761904762
```

```
print(classification_report(train_labels,ytrain_predict_nncl))
```

```
              precision    recall  f1-score   support

           0       0.80      0.89      0.85      1453
           1       0.68      0.51      0.59       647

    accuracy                           0.78      2100
   macro avg       0.74      0.70      0.72      2100
weighted avg       0.77      0.78      0.77      2100
```

```
nncl_train_precision  0.68
nncl_train_recall  0.51
nncl_train_f1  0.59
```

```
Area under Curve is 0.8166831721609928
```



NN Model Performance Evaluation on Test data:

```
confusion_matrix(test_labels,ytest_predict_nncl)
```

```
array([[553,  70],
       [138, 139]], dtype=int64)
```

```
nncl_test_acc=best_grid_nncl.score(x_test,test_labels)
nncl_test_acc
```

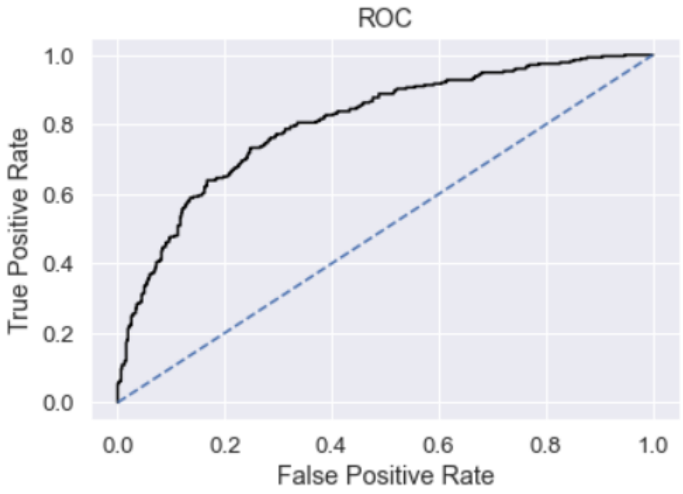
```
0.7688888888888888
```

```
print(classification_report(test_labels,ytest_predict_nncl))
```

```
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       623
           1       0.67      0.50      0.57       277

    accuracy                           0.77       900
   macro avg       0.73      0.69      0.71       900
weighted avg       0.76      0.77      0.76       900
```

```
nncl_test_precision  0.67
nncl_test_recall  0.5
nncl_test_f1  0.57
```

```
Area under Curve is 0.8044225275393896
```



<u>Neural Network Conclusion:</u>

<u>Train Data:</u>

AUC:82%

Accuracy:78%

Precision:68%

F1-score:59%

<u>Test Data:</u>

AUC:80%

Accuracy:77%

Precision:67%

F1-score:57%

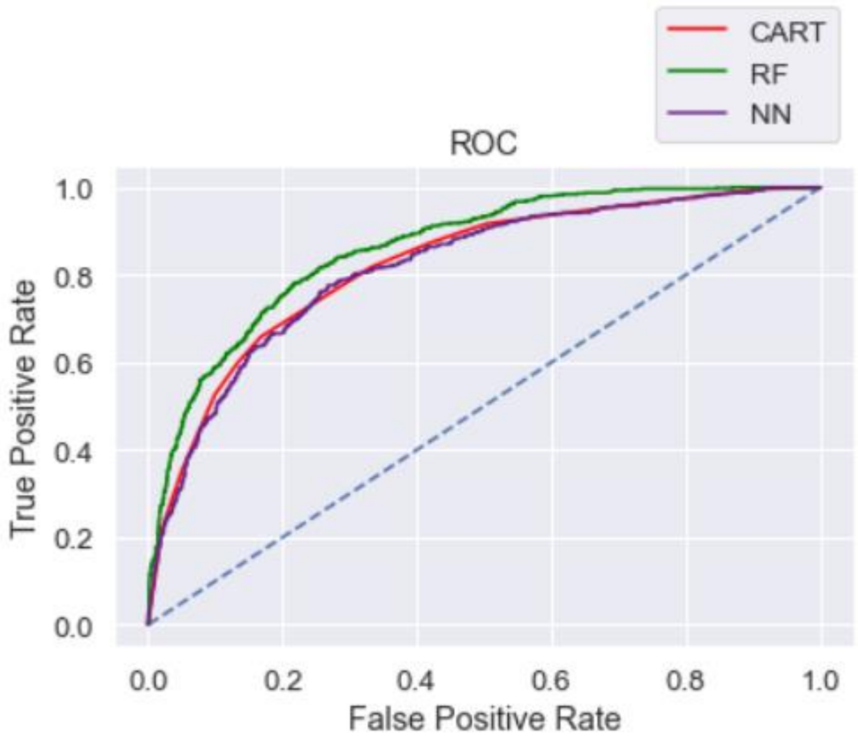
Training and Test set results are almost similar, and with the overall measures high, the model is a good model.

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

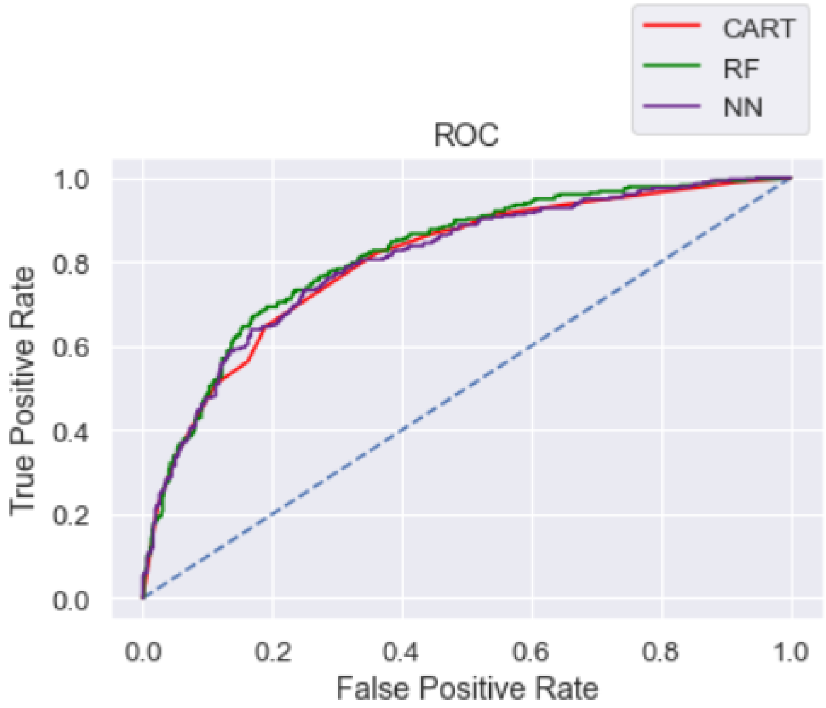
```
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
data = pd.DataFrame({'CART Train':[insurance_cart_train_acc,cart_train_auc,insurance_cart_train_recall,insurance_cart_train_preci
        'CART Test':[insurance_cart_test_acc,cart_test_auc,insurance_cart_test_recall,insurance_cart_test_precision,insurance_car
       'Random Forest Train':[rf_train_acc,rf_train_auc,rf_train_recall,rf_train_precision,rf_train_f1],
        'Random Forest Test':[rf_test_acc,rf_test_auc,rf_test_recall,rf_test_precision,rf_test_f1],
       'Neural Network Train':[nncl_train_acc,nncl_train_auc,nncl_train_recall,nncl_train_precision,nncl_train_f1],
        'Neural Network Test':[nncl_test_acc,nncl_test_auc,nncl_test_recall,nncl_test_precision,nncl_test_f1]},index=index)
round(data,2)
```

	CART Train	CART Test	Random Forest Train	Random Forest Test	Neural Network Train	Neural Network Test
Accuracy	0.79	0.77	0.80	0.77	0.78	0.77
AUC	0.82	0.80	0.86	0.82	0.82	0.80
Recall	0.53	0.51	0.60	0.53	0.51	0.50
Precision	0.70	0.67	0.71	0.66	0.68	0.67
F1 Score	0.60	0.58	0.65	0.59	0.59	0.57

ROC Curve for the 3 models on the Training data



ROC Curve for the 3 models on the Test data:



Here I'm selecting Random Forest model, as it has better Accuracy, precision, f1-score, recall other than Cart and Neural networks. That we can see from the above table and also from graph.

2.5 Inference: Based on the whole Analysis, what are the business insights and recommendations.

The main objective of the project was to develop a predictive model to predict if An Insurance firm providing tour insurance is facing higher claim frequency or not.

As per the data 90% of insurance is done by online channel. almost all the offline business has a claimed associated. JZI agency resources need to pick up sales as they are in bottom, need to run promotional marketing campaign or evaluate if we need to tie up with alternate agency, also can provide reward points or discounts accordingly.

As per our model we have accuracy of approx. 80%, so on the selling or purchase of airline tickets we can provide cross selling of insurance claim pattern, so increase in profit.

Also, we can say that the claims are processed more by airlines then the travel agency, and as per sales pattern the sales made are high at travel agencies.

Increase customer satisfaction. Reduce claim handling costs.