Project: DATA MINING

DSBA

Submitted by:

Name: Hima Jose Paliakkara

Batch: 2021-2022

Table of Contents

Contents

Problem 1: Clustering

1.1 Read the data, do the necessary initial steps, and exploratory data analysis (Univariate, Bivariate, and multivariate analysis)……..5
1.2 Do you think scaling is necessary for clustering in this case? Justify……..12
1.3 Apply hierarchical clustering to scaled data. Identify the number of optimum clusters using Dendrogram and briefly describe them……13
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……...18
1.5 Describe cluster profiles for the clusters defined. Recommend different promotional strategies for different clusters……22

Problem 2: CART-RF-ANN

2.1 Read the data, do the necessary initial steps, and exploratory data analysis (Univariate, Bivariate, and multivariate analysis)……24
2.2 Data Split: Split the data into test and train, build classification model CART, Random Forest, Artificial Neural Network……36
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, classificationreports foreachmodel……45
2.4 Final Model: Compare all the models and write an inference which model is best/optimized……..53
2.5 Inference: Based on the whole Analysis, what are the business insights and recommendations……54

List of figures

Figure 1.1.1 Spending distribution…… 6
Figure 1.1.2 Advance Payments distribution ……7
Figure 1.1.3 Probablity_of_full_payment distribution……7
Figure 1.1.4 current_balance distribution……8
Figure 1.1.5 credit_limit distribution……9
Figure 1.1.6 min_payment_amt distribution……9
Figure 1.1.7 max_spent_in_single_shopping distribution……10
Figure 1.1.8 Multivariate analysis- Pairplot……11
Figure 1.1.9 Heatmap to check correlation……11
Figure 1.1.10 Boxplot after outlier treatment……12
Figure 1.2 Plots of data before and after scaling……13
Figure 1.3.1 Dendogram - Ward method……13
Figure 1.3.2 Ward Dendogram using truncate mode……14
Figure 1.3.4 Dendogram – average linkage method……16
Figure 1.3.5 Average Dendogram using truncate mode……16
Figure 1.4.4 K-Elbow plot with WSS values……20
Figure 1.4.5 Silhouette Plot……21
Figure 2.1.1 Age Distribution……26
Figure 2.1.2 Commision Distribution ……27
Figure 2.1.2 Duration Distribution……27
Figure 2.1.3 Sales Distribution……28
Figure 2.1.4 Agency_code Countplot……29
Figure 2.1.5 Agency_code boxplot……29
Figure 2.1.6 Type boxplot……30
Figure 2.1.6 Type boxplot……30
Figure 2.1.7 Channel countplot……31
Figure 2.1.8 Channel boxplot……31
Figure 2.1.7 Product name countplot……32
Figure 2.1.8 Product name boxplot……33
Figure 2.1.6 Destination boxplot……33
Figure 2.1.6 Destination boxplot……33
Figure 2.1.7 Pairplot for continuous variables……34
Figure 2.1.8 Heatmap for continuous variables……34
Figure 2.2.1 Decision Tree after pruning……40
Figure 2.2.2 Decision Tree- Variable Importance……40
Figure 2.2.3 Random Forest – Variable Importance……42
Figure 2.3.1 DT- Train data Confusion Matrix……46
Figure 2.3.2 DT- Train data ROC curve……46
Figure 2.3.2 DT- Test data Confusion Matrix……47
Figure 2.3.3 DT- Test data ROC curve……47
Figure 2.3.4 RF- Train data Confusion Matrix……48
Figure 2.3.5 RF- Train data ROC curve……49
Figure 2.3.6 RF- Test data Confusion Matrix……49
Figure 2.3.7 RF- Test data ROC curve……50
Figure 2.3.8 NN – Train data Confusion Matrix……50
Figure 2.3.9 NN- Train data ROC curve……51

Figure 2.3.10 NN- Test data Confusion Matrix……51
Figure 2.3.2 NN - Test data ROC curve……52
Figure 2.4.1 Model Comparision ROC curve for Train data……53
Figure 2.4.1 Model Comparision ROC curve for Test data……54

List of Tables

Table 1.1.1 Reading Dataset……5
Table 1.1.2 Summary of dataset……6
Table 1.3.1 Dataset with fcluster=3……15
Table 1.3.2 Dataset with fcluster frequency……15
Table 1.3.3 Dataset with Agglo cluster frequency……16
Table 1.3.3 Dataset with fcluster=3……17
Table 1.3.4 Dataset with fcluster frequency……17
Table 1.3.5 Dataset with Agglo cluster frequency……18
Table 1.4.1 Dataset with k-means cluster=3……22
Table 1.4.2 Dataset with k-means cluster frequency……22
Table 1.5 Cluster Profile……22
Table 2.1.1 Reading dataset……24
Table 2.1.2 Summary dataset……24
Table 2.1.3 Dataset after converting objects to categorical……35
Table 2.2.1 Pred Probs of DT Test……41
Table 2.2.2 RF Pred Probs……43
Table 2.2.3 NN Pred Probs……44
Table 2.4 Model Comparison……53

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)

Answer:

Checking the 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

Table 1.1.1 Reading Dataset

Dataset 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
```

Observations:

- Dataset has 210 entries and 7 columns
- All the variables are numeric type
- There are no missing values in the dataset
- No duplicates found in the dataset.

Summary of data:

	count	mean	std	min	25%	50%	75%	max
spending	210.0	14.847524	2.909699	10.5900	12.27000	14.35500	17.305000	21.1800
advance_payments	210.0	14.559286	1.305959	12.4100	13.45000	14.32000	15.715000	17.2500
probability_of_full_payment	210.0	0.870999	0.023629	0.8081	0.85690	0.87345	0.887775	0.9183
current_balance	210.0	5.628533	0.443063	4.8990	5.26225	5.52350	5.979750	6.6750
credit_limit	210.0	3.258605	0.377714	2.6300	2.94400	3.23700	3.561750	4.0330
min_payment_amt	210.0	3.700201	1.503557	0.7651	2.56150	3.59900	4.768750	8.4560
max_spent_in_single_shopping	210.0	5.408071	0.491480	4.5190	5.04500	5.22300	5.877000	6.5500

Table 1.1.2 Summary of dataset

Observations:

- We have 7 variables
- No null values present in any variables
- The mean and median seem to be almost equal
- Standard deviation for Spending is high compared to other variables.

EDA

Univariate Analysis

Univariate analysis is the simplest form of analysing data. It helps to understand the distribution of data in dataset.

Spending

```
The skewness for Spending is: 0.3998891917177586
```

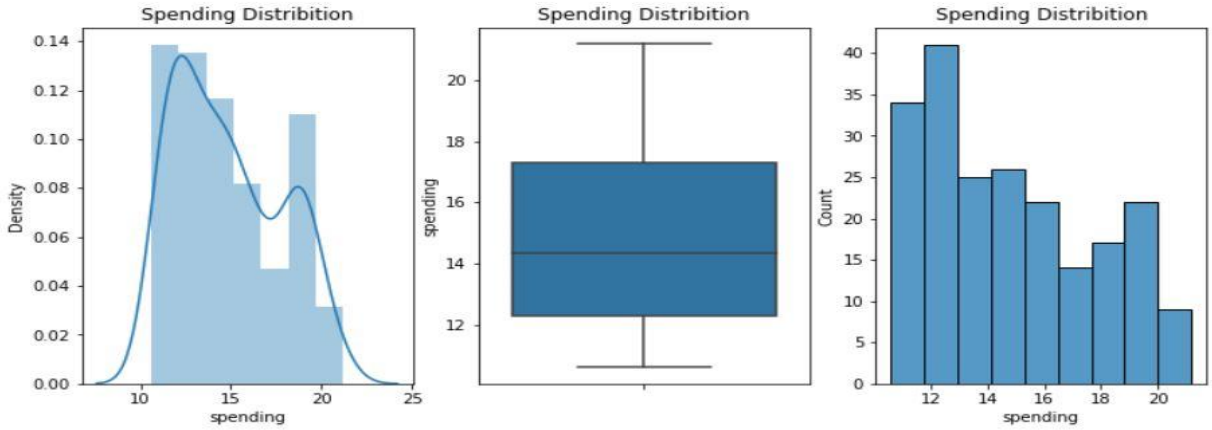


Figure 1.1.1 Spending distribution

The boxplot shows there are no outliers and the distribution seems to be positively skewed. The distribution of data is from 10 to 22.

Minimum spending	: 10.59
Maximum spending	: 21.18
Mean value	: 14.847523809523818
Median value	: 14.355
Standard deviation	: 2.909699430687361
Null values	: False

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

Advance Payments

```
The skewness for advance_payments is: 0.3865727731912213
```

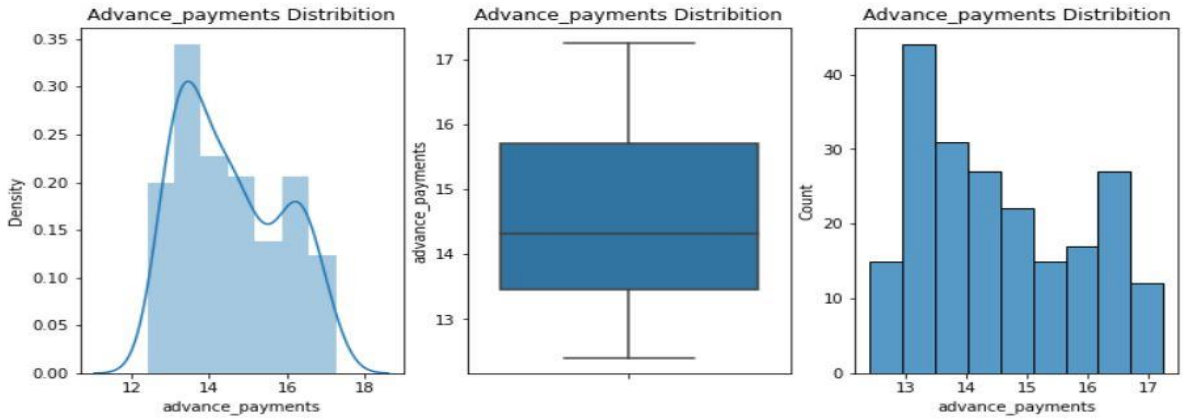


Figure 1.1.2 Advance Payments distribution

The boxplot show no outliers for the variable and the distribution seems to be positively skewed. The distplot shows the data distribution from 12 to 17

Minimum advance_payments : 12.41
Maximum advance_payments : 17.25
Mean value : 14.559285714285727
Median value : 14.32
Standard deviation : 1.305958726564022
Null values : False
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

Probability of full payment

```
The skewness for probability_of_full_payment is: -0.5379537283982823
```

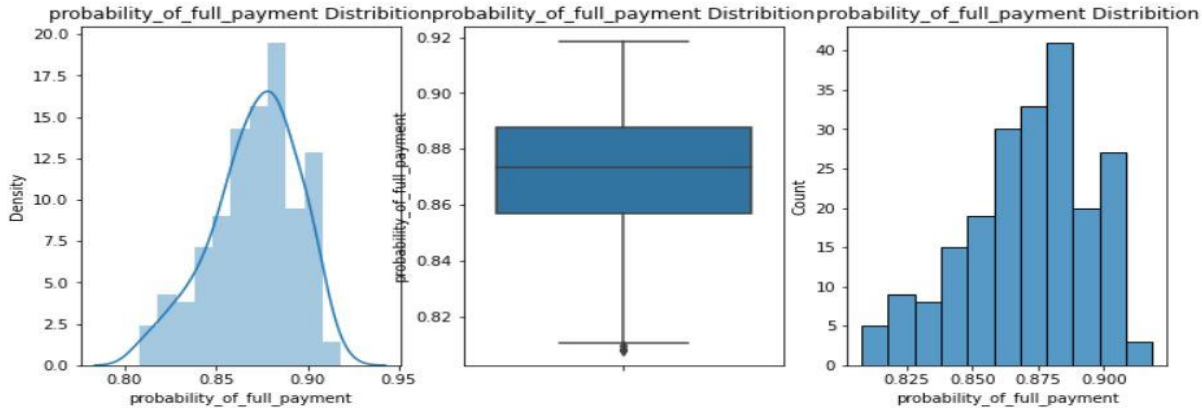


Figure 1.1.3 Probablity_of_full_payment distribution

The boxplot show there are few outliers for the variable and the distribution seems to be negatively skewed.

Minimum probability_of_full_payment	:	0.8081
Maximum probability_of_full_payment	:	0.9183
Mean value	:	0.8709985714285714
Median value	:	0.8734500000000001
Standard deviation	:	0.0236294165838465
Null values	:	False
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_paymentlower	:	3

Current Balance

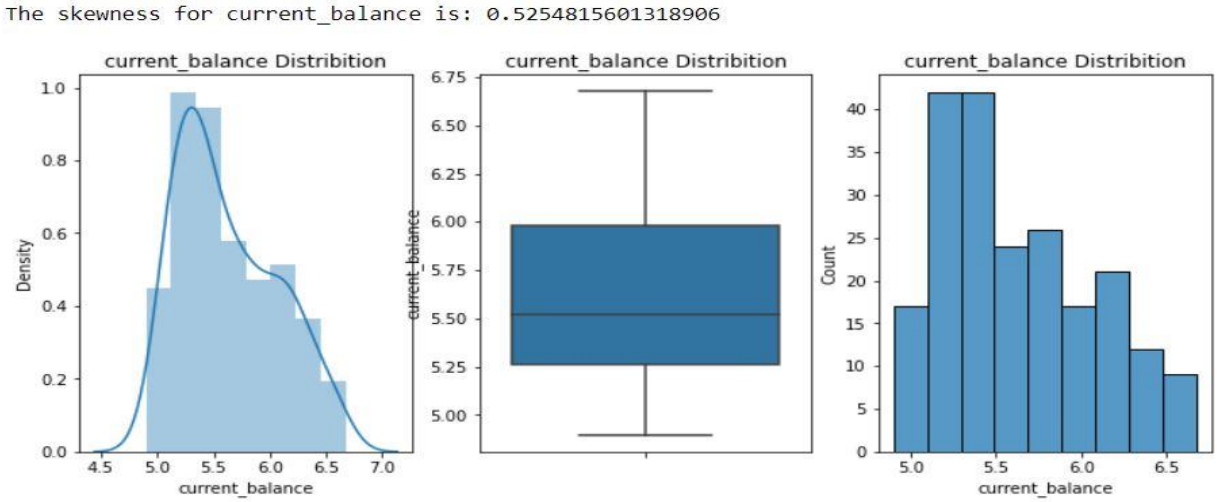


Figure 1.1.4 current_balance distribution

The boxplot show there are no outliers present for the variable and the distribution seems to be positively skewed.

Minimum current_balance	:	4.899
Maximum current_balance	:	6.675
Mean value	:	5.628533333333335
Median value	:	5.5235
Standard deviation	:	0.44306347772644944
Null values	:	False
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

Credit Limit

```
The skewness for credit_limit is: 0.1343782451316215
```

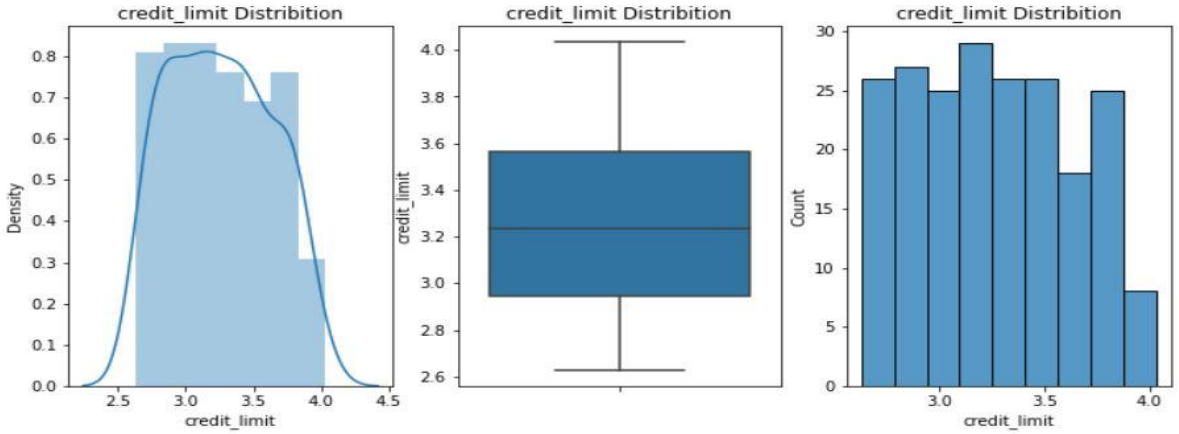


Figure 1.1.5 credit_limit distribution

The boxplot show there are no outliers for the variable and the distribution seems to be positively skewed.

Minimum credit_limit	: 2.63
Maximum credit_limit	: 4.033
Mean value	: 3.258604761904763
Median value	: 3.237
Standard deviation	: 0.37771444490658734
Null values	: False
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

Min payment amount

```
The skewness for min_payment_amt is: 0.40166734329025183
```

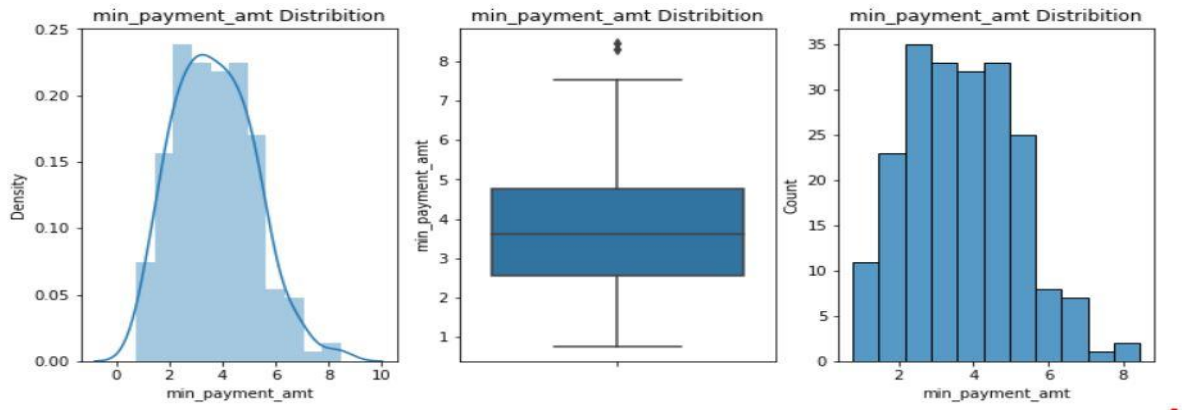


Figure 1.1.6 min_payment_amt distribution

The boxplot show there are outliers present in the variable and the distribution seems to be almost normally distributed.

Minimum min_payment_amt : 0.7651
Maximum min_payment_amt : 8.456
Mean value : 3.7002009523809503
Median value : 3.599
Standard deviation : 1.5035571308217792
Null values : False
Lower outliers in min_payment_amt : -0.7493749999999992
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

Max Spend

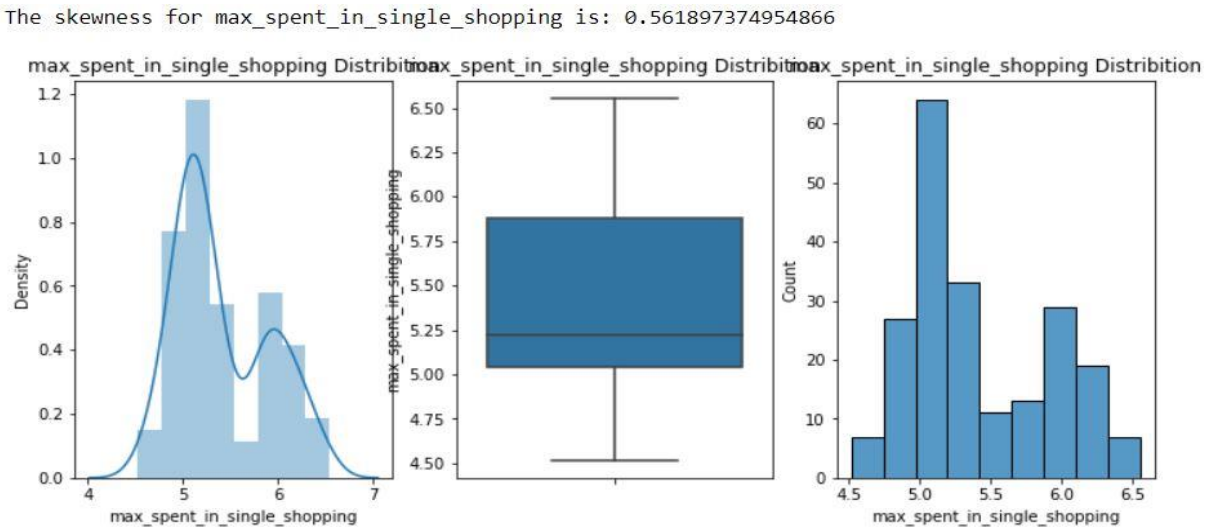


Figure 1.1.7 max_spent_in_single_shopping distribution

The boxplot shows there are no outliers in the variable and the distribution seems to be positively skewed.

Minimum max_spent_in_single_shopping : 4.519
Maximum max_spent_in_single_shopping : 6.55
Mean value : 5.408071428571429
Median value : 5.223000000000001
Standard deviation : 0.49148049910240543
Null values : False
Lower outliers in max_spent_in_single_shopping : 3.797
Upper outliers in max_spent_in_single_shopping : 7.125
Number of outliers in max_spent_in_single_shopping upper : 0
Number of outliers in max_spent_in_single_shopping lower : 0

Multivariate Analysis

Pair plot to view the relationship of all variables among each other

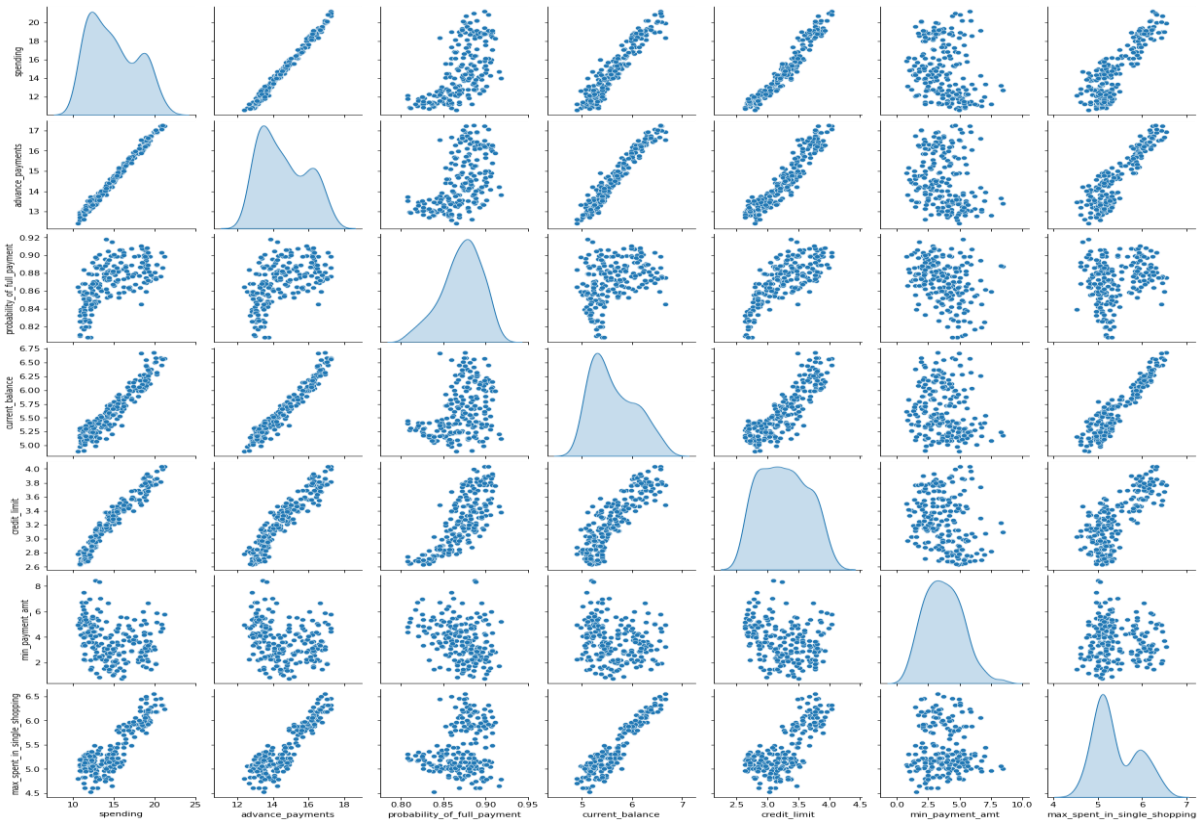


Figure 1.1.8 Multivariate analysis- Pairplot

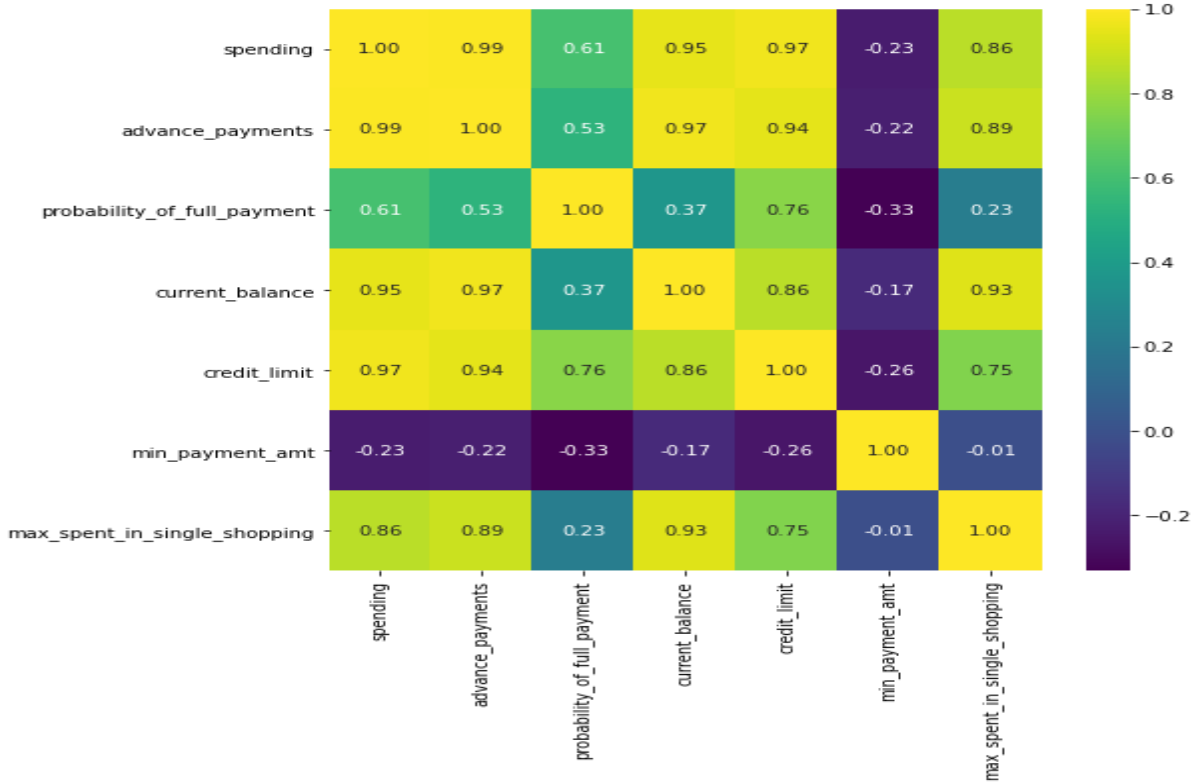


Figure 1.1.9 Heatmap to check correlation

Observations:

- Strong positive correlation between
  - Spending & advance_payments
  - Advance_payments & current_balance
  - Credit_limit & spending
  - Spending & current_balance
  - Credit_limit & advance_payments
  - Max_spend_in_single_shopping & current_balance

Removing outliers:

We choose to treat the outliers with Inter quartile range (IQR) method, instead of dropping them, as we will lose other column info and also outlier are present only in two variables.

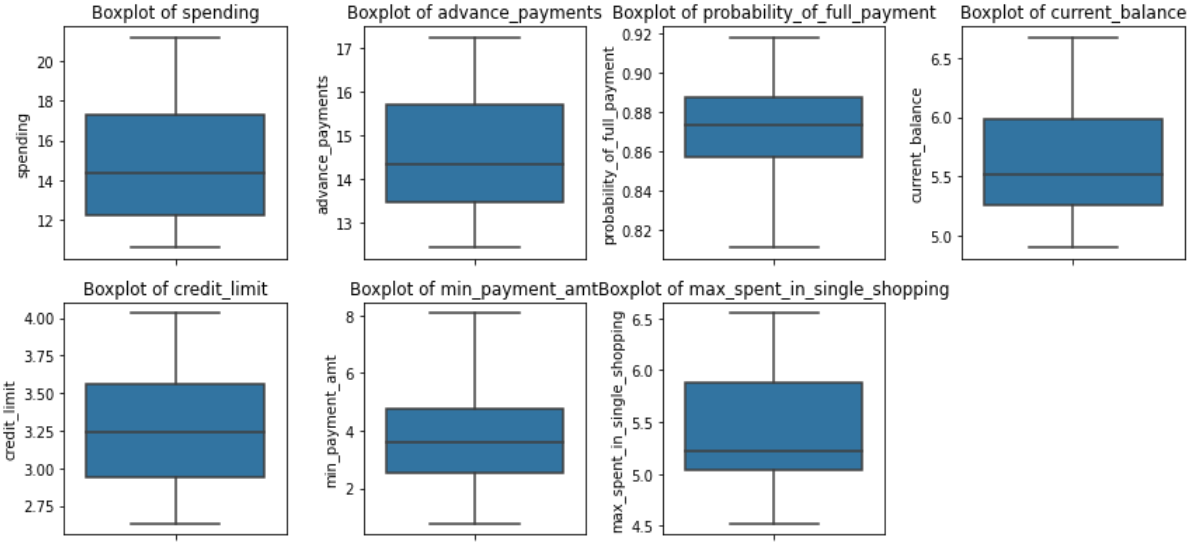


Figure 1.1.10 Boxplot after outlier treatment

All the outlier has been treated now and we are good to go for scaling the data.

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

Answer:

- Yes, scaling need to be done as the values of the variables are different.
- Scaling is important to be applied with K-means. Most of the time, the standard Euclidean distance is used (as a distance function of K-means) with the assumption that the attributes are normalized.
- spending, advance_payments are in different values and this may get more weightage.
- Scaling will have all the values in the relative same range.
- I have used zscore to standardize the data

Below are the plots of data prior and after scaling

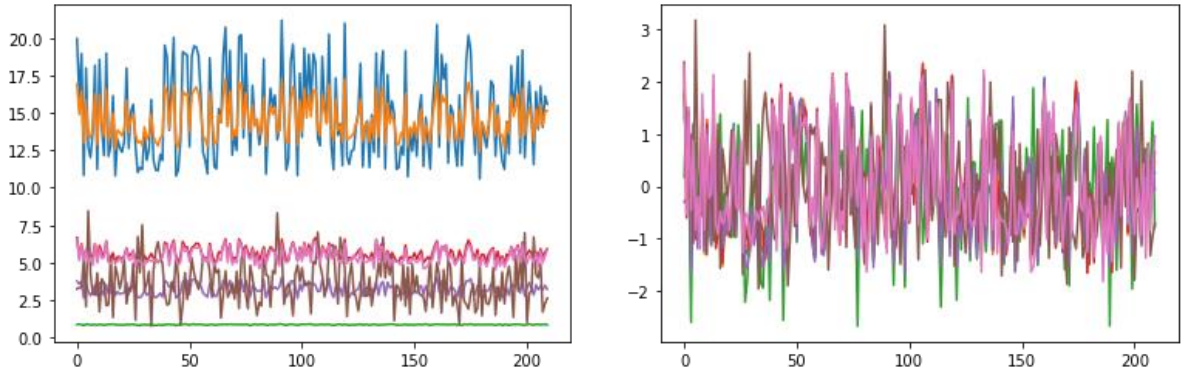


Figure 1.2 Plots of data before and after scaling

Dataset after scaling

	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

Table 1.2 Scaled Dataset

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

Answer:

Hierarchical clustering – ward's method & average method

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters.

Dendrogram of the scaled data (using Ward method)

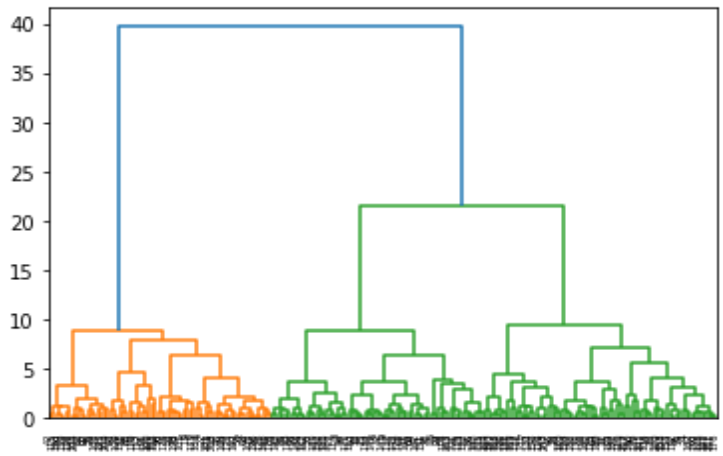


Figure 1.3.1 Dendogram - Ward method

A dendrogram is a visual representation of cluster-making. On the x-axis are the item names or item numbers. On the y-axis is the distance or height. The vertical straight lines denote the height where two items or two clusters combine. The dendrogram shows all the data points have clustered to different clusters by wards method. To find the optimal number cluster to solve the business objective we use truncate mode = lastp, where the lastp=10 according to industry set value.

Dendrogram after using the optimal value lastp=15

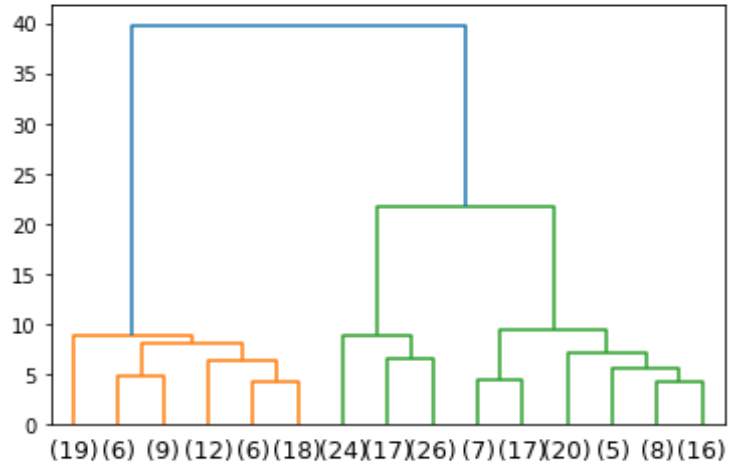


Figure 1.3.2 Ward Dendogram using truncate mode

Now the datapoints have clustered into 2 clusters. 2 clusters do not make much business sense. Therefore, selecting the clusters=3.

Finding the number of clusters using fcluster

```
#Method 1
clusters_ward = fcluster(wardlink, 3, criterion='maxclust')
clusters_ward
```

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

```
# 2
clusters_2 = fcluster(wardlink, 15, criterion='distance')
clusters_2
```

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

Now, let us go ahead and check whether the number of clusters generated by the 'maxclust' criterion is same as the number of clusters generated by the 'distance' criterion.

```
np.array_equal(clusters_1,clusters_2)
```

```
True
```

To map these clusters to the dataset we use fcluster =*3* with criterion = “maxclust”.

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	clusters_ward
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	3
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	2
4	17.99	15.86	0.8992	5.890	3.694	2.068	5.837	1

Table 1.3.1 Dataset with fcluster=3

Fcluster frequency

```
#Cluster Frequency
df_cluster['clusters_ward'].value_counts().sort_index()
```

```
1    70
2    67
3    73
Name: clusters_ward, dtype: int64
```

Appending fcluster frequency to cluster profile

clusters_ward	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	Freq
1	18.371429	16.145429	0.884400	6.158171	3.684629	3.639157	6.017371	70
2	11.872388	13.257015	0.848072	5.238940	2.848537	4.949433	5.122209	67
3	14.199041	14.233562	0.879190	5.478233	3.226452	2.612181	5.086178	73

Table 1.3.2 Dataset with fcluster frequency

Agglomerative Clustering

The agglomerative clustering is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. The algorithm starts by treating each object as a singleton cluster. Next, pairs of clusters are successively merged until all clusters have been merged into one big cluster containing all objects. The result is a tree-based representation of the objects, named dendrogram.

Checking the clusters using Agglomerative Clustering with n_cluster=3, affinity=Euclidean, linkage=ward.

```
cluster_w = AgglomerativeClustering(n_clusters=3,affinity='euclidean',linkage='ward')
Cluster_agglo_w=cluster_w.fit_predict(df_bank_Scaled.iloc[:,:8])
print(Cluster_agglo_w)
```

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

Agglomerative Cluster Frequency

```
# Agglomerative Cluster Frequency
df_cluster.Agglo_clustward.value_counts().sort_index()
```

```
0    73
1    70
2    67
Name: Agglo_clustward, dtype: int64
```

Cluster profile for Agglomerative Cluster

Agglo_clustward	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	Freq
0	14.199041	14.233562	0.879190	5.478233	3.226452	2.612181	5.086178	73
1	18.371429	16.145429	0.884400	6.158171	3.684629	3.639157	6.017371	70
2	11.872388	13.257015	0.848155	5.238940	2.848537	4.940302	5.122209	67

Table 1.3.3 Dataset with Agglo cluster frequency

Dendrogram of the scaled data (using average linkage method)

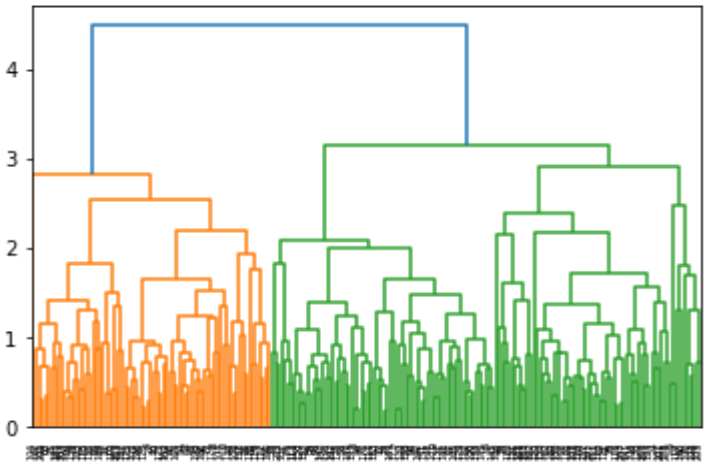


Figure 1.3.4 Dendogram – average linkage method

To find the optimal number cluster to solve the business objective we use truncate mode = lastp, where the lastp=10 according to industry set value.

Cutting the dendrogram with suitable clusters (with last p=15)

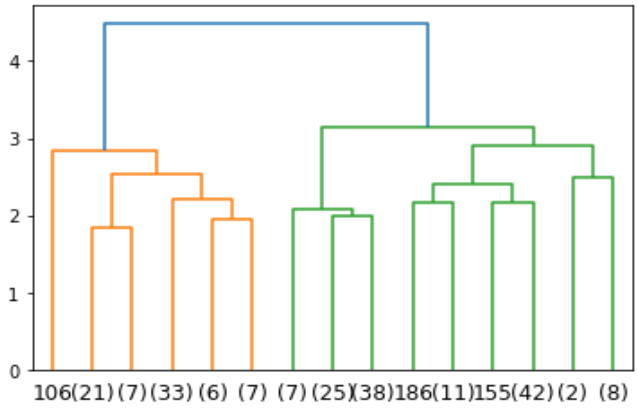


Figure 1.3.5 Average Dendogram using truncate mode

Setting criterion as maxclust, with 3 clusters

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

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

Appending fcluster to the original dataset

	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	3
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	2
4	17.99	15.86	0.8992	5.890	3.694	2.068	5.837	1

Table 1.3.3 Dataset with fcluster=3

Cluster profile for fcluster

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	11.916857	13.291000	0.846766	5.258300	2.846000	4.619000	5.115071	70
3	14.217077	14.195846	0.884869	5.442000	3.253508	2.768418	5.055569	65

Table 1.3.4 Dataset with fcluster frequency

Agglomerative Clustering

Checking the clusters using Agglomerative Clustering with n_cluster=3, affinity=Euclidean, linkage=average.

```
# Set n_clusters=3,affinity='euclidean', linkage='average' and store the result in another object 'Cluster_agglo'

cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='average')
Cluster_agglo=cluster.fit_predict(df_bank_Scaled.iloc[:,:8])
print(Cluster_agglo)
```

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

Agglomerative Cluster Frequency

```
# Agglomerative Cluster Frequency
df_cluster1.Agglo_CLusters.value_counts().sort_index()
```

```
0    65
1    75
2    70
Name: Agglo_CLusters, dtype: int64
```

Cluster profile for Agglomerative Cluster

Agglo_CLusters	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	Freq
0	14.217077	14.195846	0.884869	5.442000	3.253508	2.768418	5.055569	65
1	18.129200	16.058000	0.881595	6.135747	3.648120	3.650200	5.987040	75
2	11.916857	13.291000	0.846766	5.258300	2.846000	4.619000	5.115071	70

Table 1.3.5 Dataset with Agglo cluster frequency

Observations:

- Both the method used for clustering are almost similar means, minor variations which is known to us that occurs.
- Cluster grouping 3 or 4 based on dendrogram looks good, further analysis based on dataset I had gone ahead with 3 group cluster.
- Based on the 3 clusters formed it shows the patterns of high (clsuter1)/ medium(cluster2)/ low (cluster3) spending with high value item as max_spend_in_single_shopping and payment mode as probability_of_full_payment.

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.

Answer:

K means clustering

k-means clustering is the most used non-hierarchical clustering technique. It aims to partition n observations into kclusters in which each observation belongs to the cluster whose mean (centroid) is nearest to it, serving as a prototype of the cluster.

Randomly decided to give n_clusters=2 ,3 & 4 and check the distribution of clusters accordingly.

Apply K-means technique to the scaled data

```
k_means2 = KMeans(n_clusters = 2,random_state=1)
k_means2.fit(df_bank_Scaled)
k_means2.inertia_
```

```
659.14740095485
```

```
k_means3 = KMeans(n_clusters = 3,random_state=1)
k_means3.fit(df_bank_Scaled)
k_means3.inertia_
```

```
430.298481751223
```

```
k_means4 = KMeans(n_clusters = 4,random_state=1)
k_means4.fit(df_bank_Scaled)
k_means4.inertia_
```

```
371.2217639268478
```

For n_cluster=3 , below is the cluster output

```
K-means.inertia_ for n_cluster=3 is: 430.298481751223
```

```
K-means.labels_ for n_cluster=3:
 [0 2 0 1 0 1 1 2 0 1 0 2 1 0 2 1 2 1 1 1 1 1 0 1 2 0 2 1 1 1 2 1 1 2 1 1 1
 1 1 0 0 2 0 0 1 1 2 0 0 0 1 0 0 0 0 0 1 1 1 0 2 1 1 2 2 0 0 2 0 1 2 1 0 0
 1 0 2 1 0 2 2 2 2 0 1 2 0 2 0 1 2 0 2 1 1 0 0 0 1 0 2 0 2 0 2 0 0 1 1 0 2
 2 0 1 1 0 2 2 1 0 2 1 1 1 2 2 0 1 2 2 1 2 2 0 1 0 0 1 0 2 2 2 1 1 2 1 0 1
 2 1 2 1 2 2 1 2 2 1 2 0 0 1 0 0 0 1 2 2 2 1 2 1 2 0 0 0 2 1 2 1 2 2 2 2 0
 0 1 2 2 1 1 2 1 0 2 0 0 1 0 1 2 0 2 1 0 2 0 2 2 2]
```

There are some other techniques from which can be used to find the approximate or optimal value of k. These techniques include:

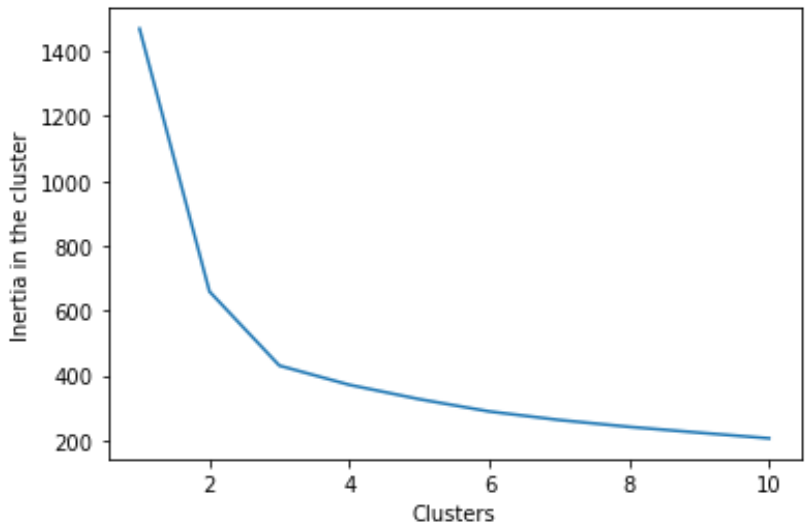
1.Elbow Method

2.Silhouette method

1.Elbow Method

It is most popular and well-known method to find the optimal no. of clusters or the value of k in the process of clustering. This method is based of plotting the value of cost function against different values of k. As the number of clusters (k) increase lesser number of points fall within clusters or around the centroids. Hence the average distortion decreases with the increase of number of clusters. The point where the distortion declines most is said to be the elbow point and define the optimal number of clusters for dataset.

To find the optimal number of clusters we use K-elbow method




```
wss
```

```
[1469.999999999999,
 659.14740095485,
 430.298481751223,
 371.2217639268478,
 326.88464076818576,
 290.1513312373964,
 263.0291032947616,
 242.81070733171134,
 221.48759698221107,
 206.3290465077041]
```

Figure 1.4.4 K-Elbow plot with WSS values

2.Silhouette method

Silhouette is a different method to determine optimal number of clusters for given dataset. It defines as a coefficient of measure of how similar an observation to its own cluster compared to that of other clusters. The range of silhouette coefficient varies between -1 to 1.1 value indicate that an observation is far from its neighbouring cluster and close to its own whereas-1 denotes that an observation is close to neighbouring cluster than its own cluster. The 0 value indicate the presence of observation on boundary of two clusters.

Finding the silhouette_score to choose the optimal number of clusters.

```
ss={1:0}
for i in range(2,11):
    clusterer=KMeans(n_clusters=i,init='k-means++',random_state=1)
    y=clusterer.fit_predict(x_full)
    # The higher (up to 1) the better
    s=silhouette_score(x_full,y)
    ss[i]=round(s,5)
    print("The Average Silhouette Score for {}clusters is {}".format(i,round(s,5)))
```

```
The Average Silhouette Score for 2clusters is 0.4656
The Average Silhouette Score for 3clusters is 0.40081
The Average Silhouette Score for 4clusters is 0.32944
The Average Silhouette Score for 5clusters is 0.28649
The Average Silhouette Score for 6clusters is 0.28466
The Average Silhouette Score for 7clusters is 0.2747
The Average Silhouette Score for 8clusters is 0.26874
The Average Silhouette Score for 9clusters is 0.26594
The Average Silhouette Score for 10clusters is 0.25381
```

Silhouette Plot

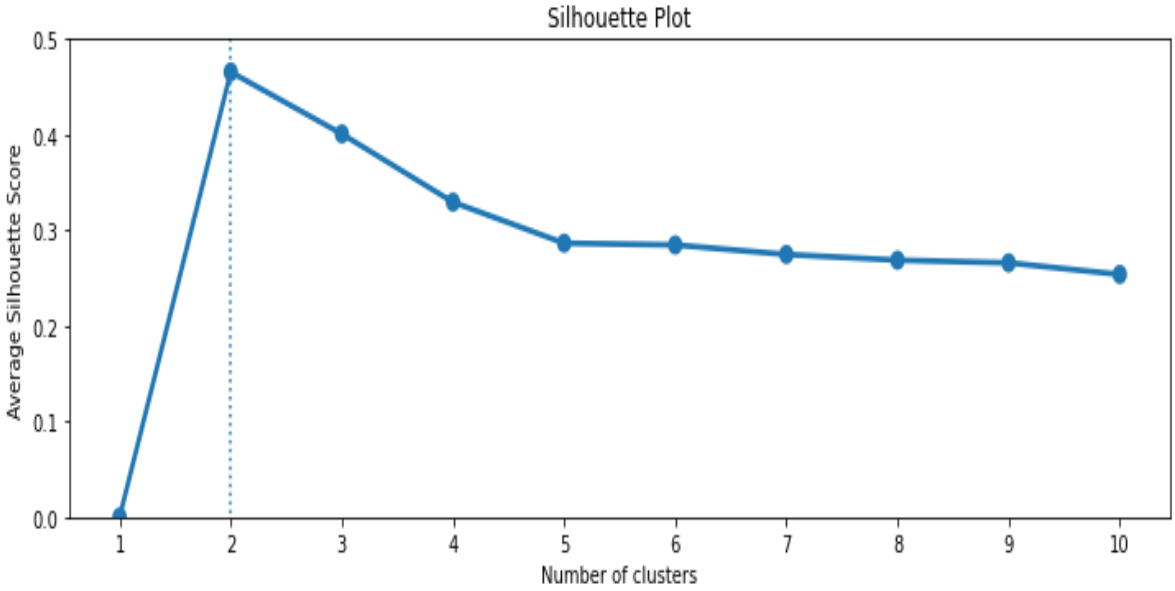


Figure 1.4.5 Silhouette Plot

Observations:

It is clear from above figure that the maximum value of average silhouette score is achieved for k =2, which, therefore, is considered to be the optimum number of clusters for this data. But statistically 2 clusters are not good for the analysis and doesn't full fill the need for clustering. The elbow curve seen above also shows that after 3 clusters there is no huge drop in the values. Hence selecting **k=3** as there is no much drop in values after that.

Checking the silhouette_samples for cluster 2,3 & 4

```
#cluster 2
print('silhouette_samples for n_clusters=2 is:',silhouette_samples(df_bank_Scaled,labels2).min())
```

```
silhouette_samples for n_clusters=2 is: -0.005677379727717533
```

```
#cluster 3
print('silhouette_samples for n_clusters=3 is:',silhouette_samples(df_bank_Scaled,labels3).min())
```

```
silhouette_samples for n_clusters=3 is: 0.0027685411286160638
```

```
#cluster 4
print('silhouette_samples for n_clusters=4 is:',silhouette_samples(df_bank_Scaled,labels4).min())
```

```
silhouette_samples for n_clusters=4 is: -0.051445326522361244
```

silhouette score & silhouette sample is good for 3 clusters than for 2 & 4 clusters. So, final clusters will be 3

Adding cluster to the dataset

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	Clus_kmeans3
0	19.94	16.92	0.875200	6.675	3.763	3.252	6.550	0
1	15.99	14.89	0.906400	5.363	3.582	3.336	5.144	2
2	18.95	16.42	0.882900	6.248	3.755	3.368	6.148	0
3	10.83	12.96	0.810588	5.278	2.641	5.182	5.185	1
4	17.99	15.86	0.899200	5.890	3.694	2.068	5.837	0

Table 1.4.1 Dataset with k-means cluster=3

Cluster Profile

Clus_kmeans3	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	freq
0	18.495373	16.203433	0.884210	6.175687	3.697537	3.632373	6.041701	67
1	11.856944	13.247778	0.848330	5.231750	2.849542	4.733892	5.101722	72
2	14.437887	14.337746	0.881597	5.514577	3.259225	2.707341	5.120803	71

Table 1.4.2 Dataset with k-means cluster frequency

Observations:

Cluster 0 – High

Cluster 1 – Low

Cluster 2 – Medium

- Using K-means cluster 3 we can find it optimal as there is no huge drop in inertia values, the elbow curve also shows the same result.
- The silhouette width score of the K – means also seems to be very less value which indicates all the data points are properly clustered to the cluster. There is no mismatch in the data points with regards to clustering.
- Three group clusters give the pattern of low, medium, high spending.

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

Answer:

Cluster Group Profiles

Clus_kmeans3	0	1	2
spending	18.5	11.9	14.4
advance_payments	16.2	13.2	14.3
probability_of_full_payment	0.9	0.8	0.9
current_balance	6.2	5.2	5.5
credit_limit	3.7	2.8	3.3
min_payment_amt	3.6	4.7	2.7
max_spent_in_single_shopping	6.0	5.1	5.1

Table 1.5 Cluster Profile

Cluster 0: High Spending

Cluster 1: Low Spending

Cluster 2: Medium Spending

High Spending

- Maximum max_spent_in_single_shopping is high for this group, can be offered discount/offer on next transactions upon full payment
- Offering reward points might increases the purchases.
- Increase their credit limit which will eventually increase their spending
- As they are customers with good repayment record, can try giving loan against the credit card.
- Partner with high-end brands to increase one_time_maximum expenditure.

Medium Spending

- Customers that pay their bills, make purchases, and sustaining a good credit score. We can increase their credit limit or reduce the interest rate.
- Promote premium cards/loyalty cards to increase transaction
- Try premium ecommerce sites, travel portals, and trip airlines/hotels to increase spending habits, as this will encourage people to spend more.

Low Spending

- Customers should be offered payment remainders. Offers can be provided on early payments to enhance their payment rate.
- Increase their spending habits by tie up with grocery stores, utilities (electricity, phone , gas etc)

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.

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

Answer:

Reading the dataset

	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

Table 2.1.1 Reading dataset

Summary of the dataset

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Age	3000.0	NaN	NaN	NaN	38.091	10.463518	8.0	32.0	36.0	42.0	84.0
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	3000.0	NaN	NaN	NaN	14.529203	25.481455	0.0	0.0	4.63	17.235	210.21
Channel	3000	2	Online	2954	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Duration	3000.0	NaN	NaN	NaN	70.001333	134.053313	-1.0	11.0	26.5	63.0	4580.0
Sales	3000.0	NaN	NaN	NaN	60.249913	70.733954	0.0	20.0	33.0	69.0	539.0
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

Table 2.1.2 Summary dataset

Checking duplicates

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

Checking null values

```
df_cart.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
```

Observations:

- Dataset has 3000 entries and 10 columns
- No missing values present in the dataset
- From INFO, found that the dataset has object, integer and float, we have to change the object datatype to numeric value.
- 139 duplicate records are found, but it can be of different customers, there is no customer ID or any unique identifier, so I am not dropping them off.
- Categorial code variable maximum unique count is 5
- Channel is online
- Destination ASIA seems to be most destination place by customers.
- Customised plan is the most Product chose by the customers
- Most preferred type seems to be Travel Agency
- Duration has a negative value, it’s not possible, wrong entry.

Getting unique counts of all Nominal Variables

```
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
```

Univariate/Bivariate analysis

Age

```
The skewness for Age is: 1.149712770495169
```

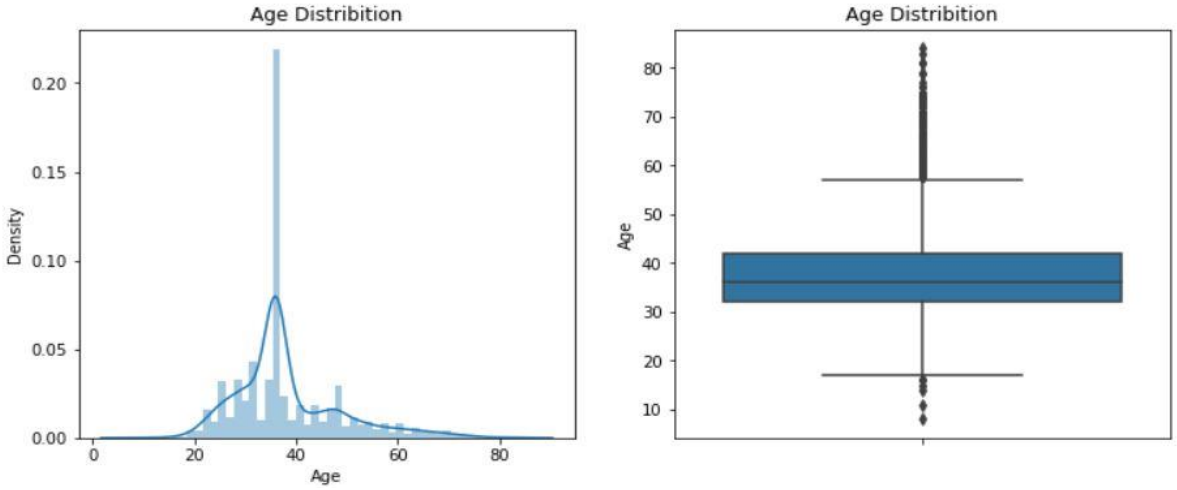


Figure 2.1.1 Age Distribution

The boxplot show outliers for the variable and the distribution seems to be positively skewed. The distplot shows the data distribution from 20 to 80.

Minimum Age	: 8
Maximum Age	: 84
Mean value	: 38.091
Median value	: 36.0
Standard deviation	: 10.463518245377944
Lower outliers in Age	: 17.0
Upper outliers in Age	: 57.0
Number of outliers in Age upper	: 198
Number of outliers in Age lower	: 6

Commision

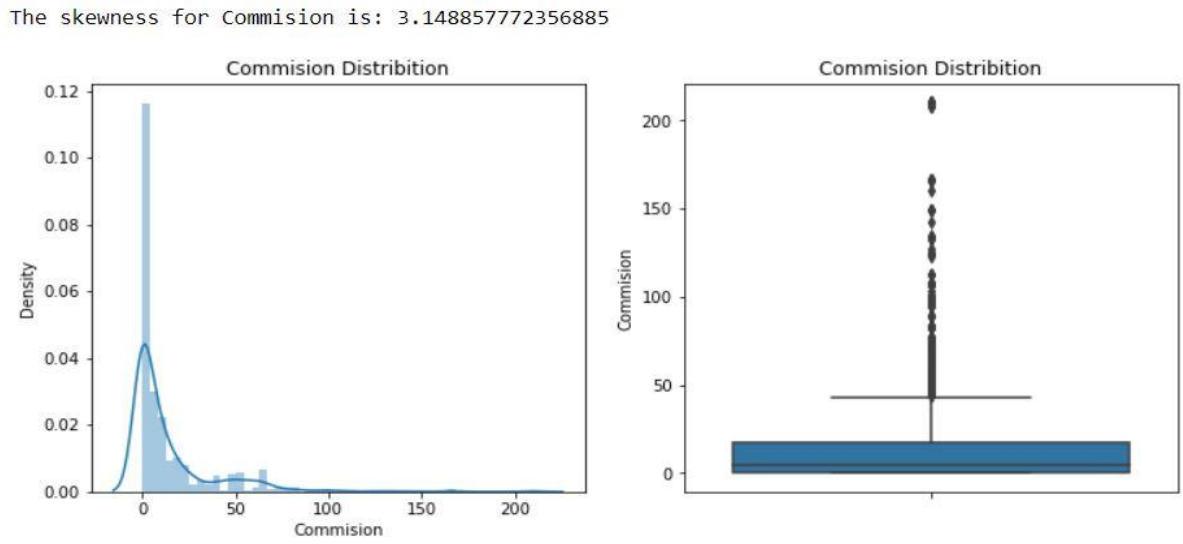


Figure 2.1.2 Commision Distribution

The boxplot show outliers for the variable and the distribution seems to be positively skewed.

Minimum Age	:	0.0
Maximum Age	:	210.21
Mean value	:	14.529203333333266
Median value	:	4.63
Standard deviation	:	25. 48145450662553
Lower outliers in Commision	:	-25.8525
Upper outliers in Commision	:	43.0875
Number of outliers in Commision upper	:	362
Number of outliers in Commision lower	:	0

Duration

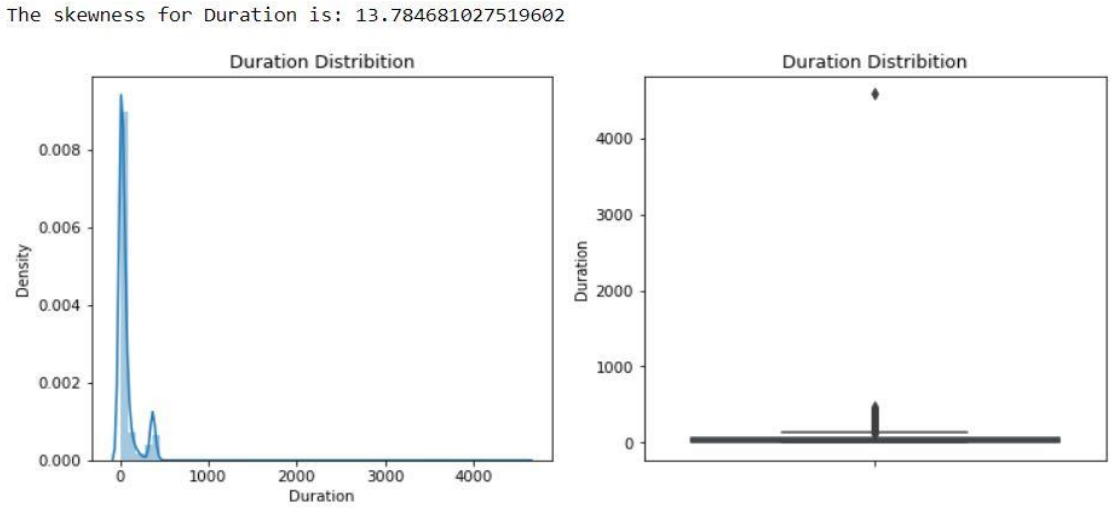


Figure 2.1.2 Duration Distribution

The boxplot show outliers for the variable and the distribution seems to be positively skewed. The dist plot shows the distribution of data from 0 to 100

Minimum Age	:	-1
Maximum Age	:	4580
Mean value	:	70.00133333333333
Median value	:	26.5
Standard deviation	:	134.05331313253495
Lower outliers in Duration	:	-67.0
Upper outliers in Duration	:	141.0
Number of outliers in Duration upper	:	382
Number of outliers in Duration lower	:	0

Sales

```
The skewness for Sales is: 2.381148461687274
```

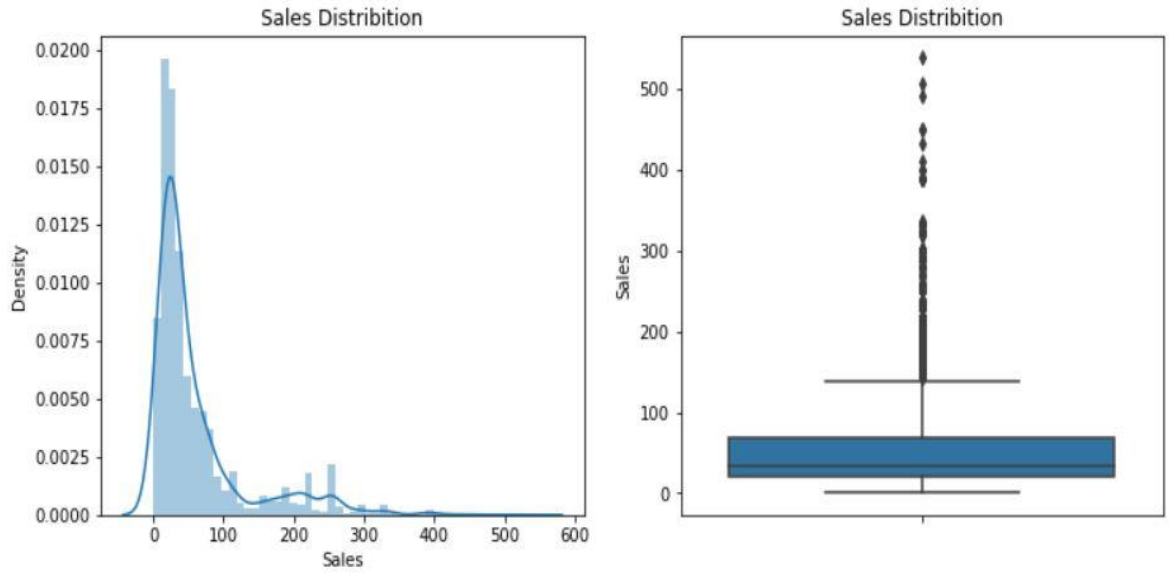


Figure 2.1.3 Sales Distribution

The boxplot show outliers for the variable and the distribution seems to be positively skewed. The dist plot shows the distribution of data from 0 to 300

Minimum Age	:	0.0
Maximum Age	:	539.0
Mean value	:	60.24991333333344
Median value	:	33.0
Standard deviation	:	70.73395353143047
Lower outliers in Sales	:	-53.5
Upper outliers in Sales	:	142.5
Number of outliers in Sales upper	:	353
Number of outliers in Sales lower	:	0

Categorical Variables

Agency Code

Countplot

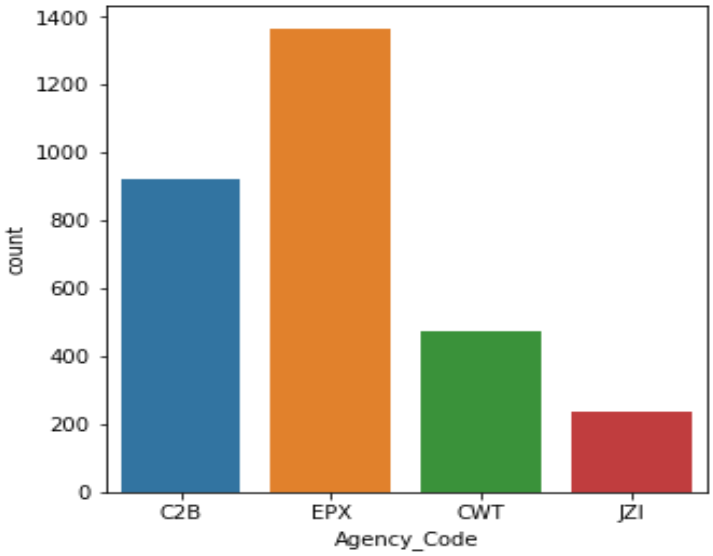


Figure 2.1.4 Agency_code Countplot

From the countplot distribution we can find that EPX has maximum frequency.

Boxplot

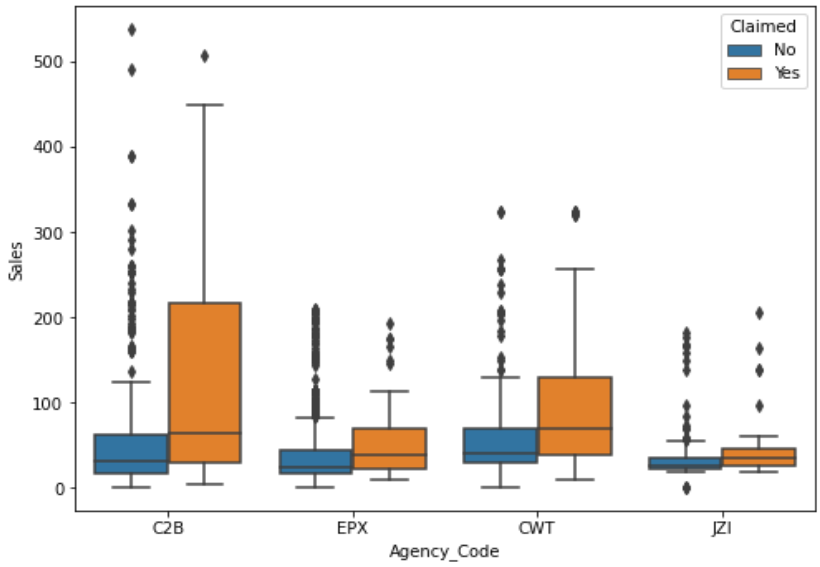


Figure 2.1.5 Agency_code boxplot

The boxplot above shows the split of Sales with different Agency code and hue with claimed. It shows that C2B has claimed more claims than other Agency.

Type

Countplot

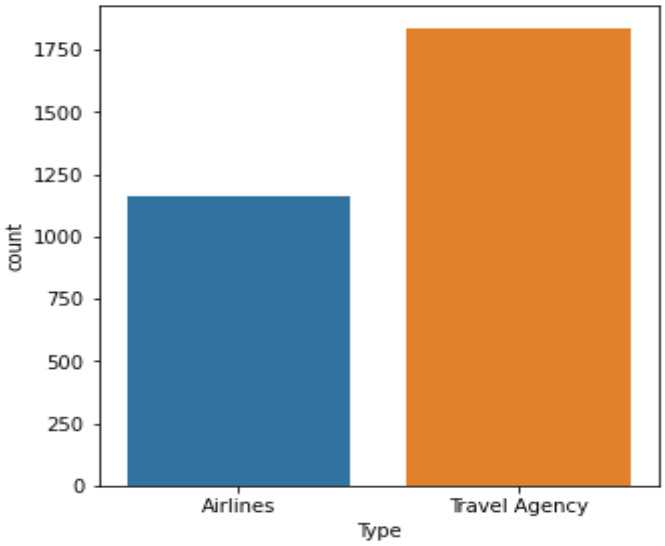


Figure 2.1.6 Type boxplot

From the countplot we can find that the most preferred Type mode is Travel Agency

Boxplot

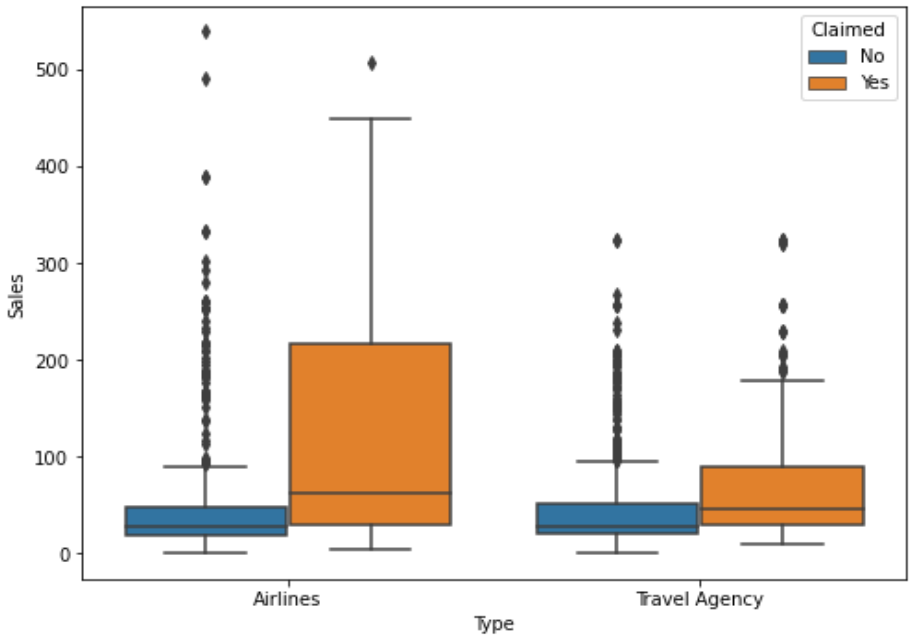


Figure 2.1.6 Type boxplot

The boxplot above shows the split of Sales with different Type and hue with claimed.

It shows that Airlines has more claimed.

Channel

Countplot

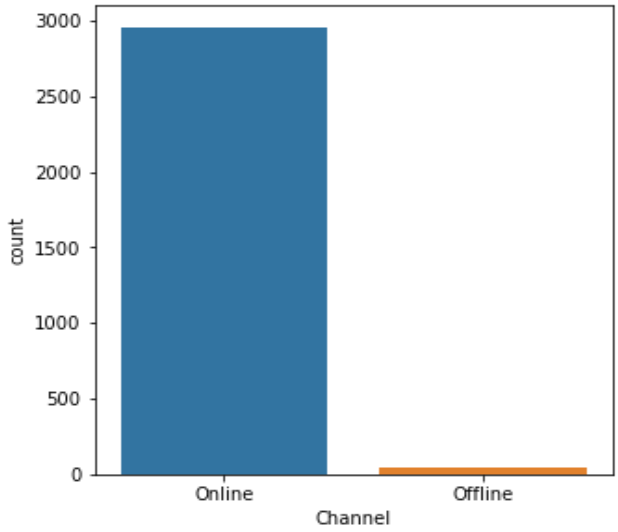


Figure 2.1.7 Channel countplot

From the countplot we can find that the most preferred Channel is Online

Boxplot

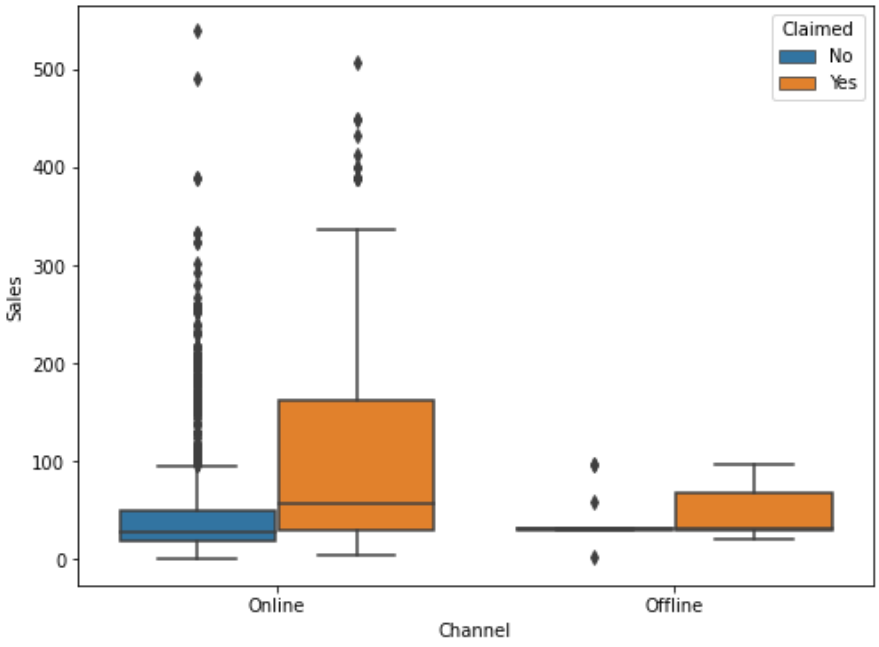


Figure 2.1.8 Channel boxplot

The boxplot above shows the split of Sales with different Channel and hue with claimed.

Product Name

Countplot

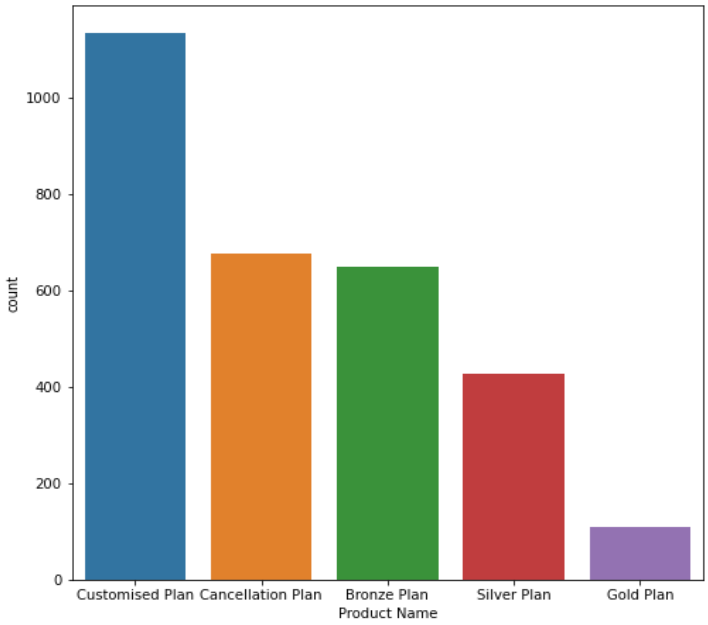


Figure 2.1.7 Product name countplot

Customised plan is the most preferred plan by customers compared to the other plans.

Boxplot

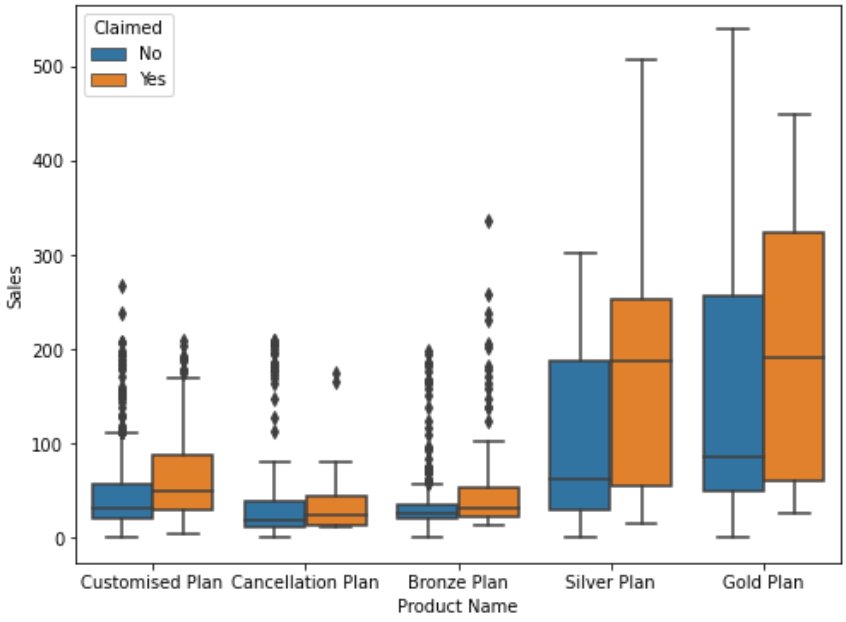


Figure 2.1.8 Product name boxplot

The boxplot above shows the split of Sales with different Product name and hue with claimed.

It shows that Gold plan has more claimed.

Destination

Countplot

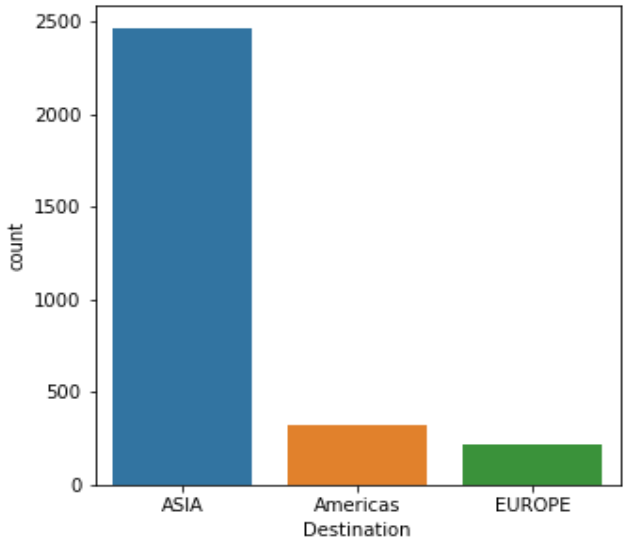


Figure 2.1.6 Destination boxplot

Asia is the most preferred destination by the customers.

Boxplot

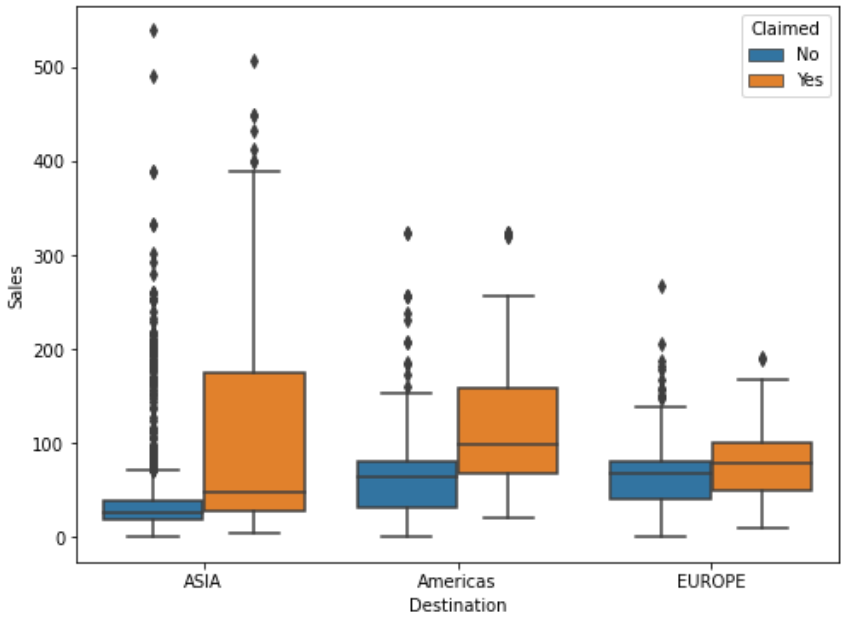


Figure 2.1.6 Destination boxplot

The boxplot above shows the split of Sales with different Destination and hue with claimed. It shows that Asia has more claimed.

Checking Pairwise distribution for continuous variables

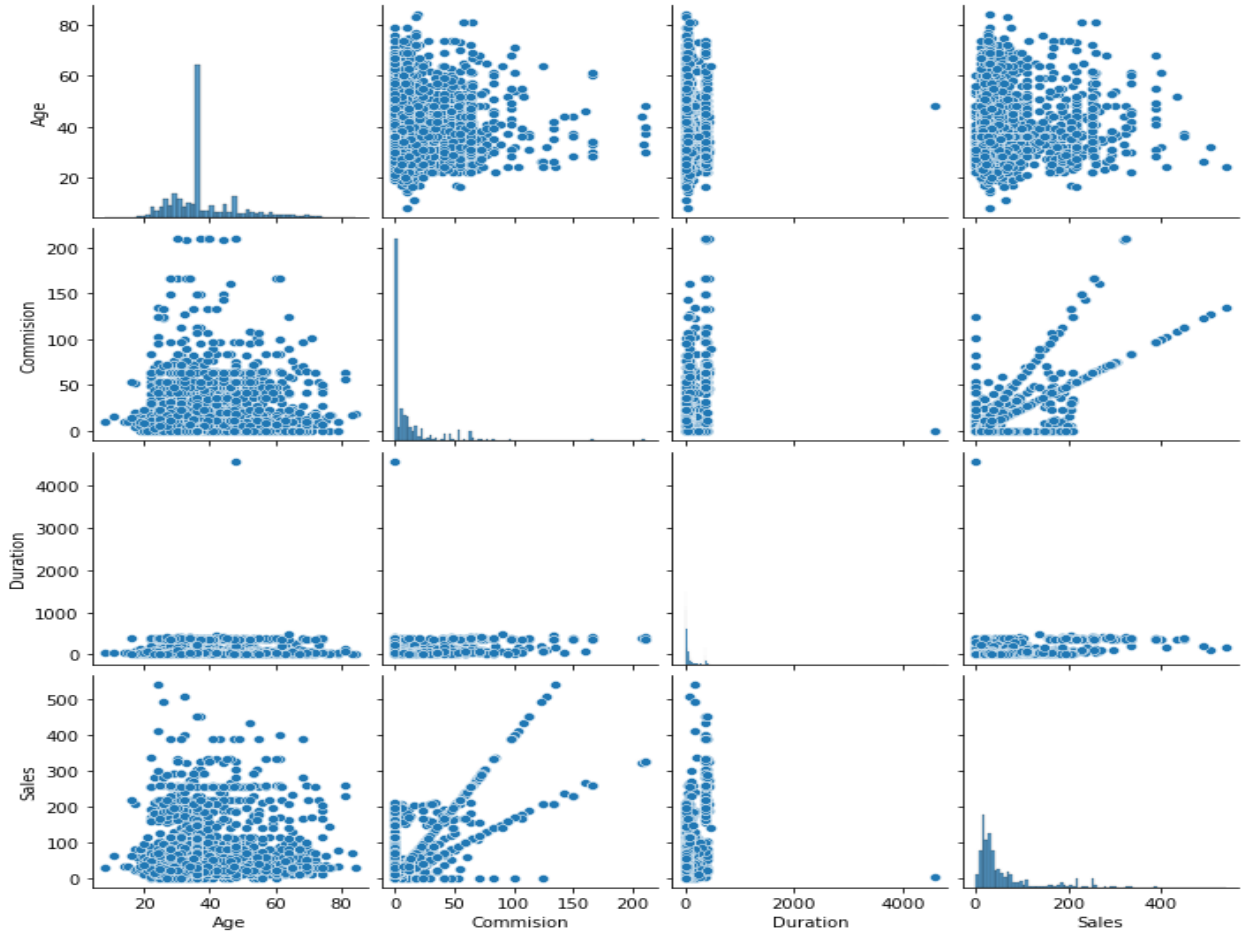


Figure 2.1.7 Pairplot for continuous variables

Checking for Correlation

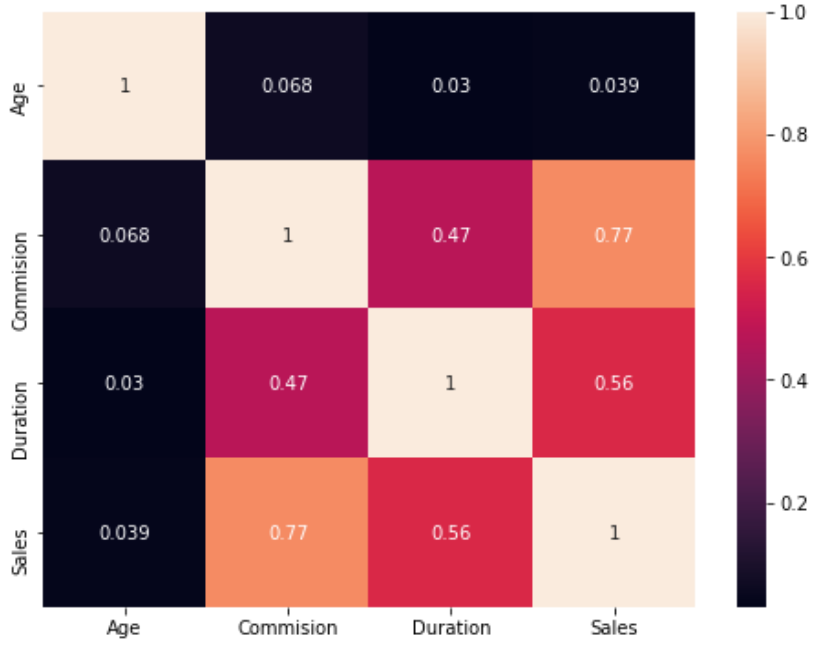


Figure 2.1.8 Heatmap for continuous variables

There is mostly positive correlation between different attributes. Only the ‘Sales’ & ‘Commision’ are highly correlated.

Converting all objects to categorical codes

To build our models we are changing the object data type to numeric values.

```
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]
```

Reading dataset

	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

Table 2.1.3 Dataset after converting objects to categorical

Checking the proportion of 1s and 0s in the dataset. That is our target column “Claimed”

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

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

So around 69 percent of customers have not claimed their insurance, and there is no class imbalance here because both classes have adequate numbers. The model has a 69 percent accuracy rate. Let's look at the model's performance after we've used the best grid parameters.

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

Answer:

Converting all objects to categorical codes

To build our models we are changing the object data type to numeric values.

```
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]
```

Checking 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
```

Reading dataset

	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

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

```
X = df_cart.drop("Claimed", axis=1)

y = df_cart.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

Splitting data into training and test set

For training and testing purpose we are splitting the dataset into train and test data in the ratio 70:30. Very often, the proportion chosen is 70% for the training set and 30% for the test. The idea is that more training data is a good thing because it makes the classification model better whilst more test data makes the error estimate more accurate.

```
from sklearn.model_selection import train_test_split

X_train, X_test, train_labels, test_labels = train_test_split(X, y, test_size=.30, random_state=1)
```

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)
print('Total Obs    :',2100+900)
```

```
X_train      : (2100, 9)
X_test       : (900, 9)
train_labels : (2100,)
test_labels  : (900,)
Total Obs    : 3000
```

Scaling the data

Feature scaling is performed for any machine learning/neural network so that sample data does not take relatively large values and remains homogeneous. Simply fit() method does is create a model that extracts the various parameters from your training samples to do the neccessary transformation later on. transform() on the other hand is doing the actual transformation to the data itself returning a standardized or scaled form. fit_transform() is just a faster way of doing the operations of fit() and transform() consequently.

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

```
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

Model 1: Building a Decision Tree Classifier

```
# Initialise a Decision Tree Classifier

dt_model = DecisionTreeClassifier(criterion = 'gini',random_state=1 )
```

```
# Fit the model

dt_model.fit(X_train, train_labels)
```

```
DecisionTreeClassifier(random_state=1)
```

```
from sklearn import tree

train_char_label = ['No', 'Yes']
DT_file = open('d:\DT_123.dot','w')
dot_data = tree.export_graphviz(dt_model, out_file=DT_file, feature_names = list(df_cart), class_names = list(train_char_label))

DT_file.close()
```

```
#Checking the features

print (pd.DataFrame(dt_model.feature_importances_, columns = ["Imp"], index = df_cart.columns).sort_values('Imp',ascending=False)
```

```
                 Imp
Duration    0.262122
Sales       0.199864
Agency_Code 0.194770
Age         0.177894
Commision   0.095127
Product Name 0.043258
Destination 0.019321
Channel     0.007262
Type        0.000383
```

To get the best set of hyperparameters we can use Grid Search. Grid Search passes all combinations of hyperparameters one by one into the model and check the result. Finally, it gives us the set of hyperparameters which gives the best result after passing in the model.

Finding Best Parameters using GridSearch

```
from sklearn.model_selection import GridSearchCV

param_grid_dtcl = {
    'criterion': ['gini'],
    'max_depth': [4,5,6,7],
    'min_samples_leaf': [20,30,40,50],
    'min_samples_split': [60,90,120,150],
}

dtcl = DecisionTreeClassifier(random_state=1)

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

- **criterion:** string, optional (default="gini"). The function to measure the quality of a split. criteria are "gini" for the Gini impurity
- **max_depth:** int or None, optional (default=None). The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure. The max_depth value should be adjusted according no to high/low to avoid overfitting/underfitting.
- **min_samples_split:** int, float, optional (default=2). min_samples_split is used to control over-fitting. depending on the level of underfitting or overfitting, you can tune the values for min_samples_split.
- **min_samples_leaf:** int, float, optional (default=1). The minimum number of samples required to be at a leaf node. min_samples_leaf is also used to control over-fitting by defining that each leaf has more than one element. Thus ensuring that the tree cannot overfit the training dataset.
- **random_state:** int, RandomState instance or None, optional (default=None). Random state ensures that the splits that you generate are reproducible. Scikit-learn uses random permutations to generate the splits. The random state that you provide is used as a seed to the random number generator. This ensures that the random numbers are generated in the same order.
- **cv:** number of cross-validation you have to try for each selected set of hyperparameters.

Fitting the model using the optimal values from Grid Search

```
grid_search_dtcl.fit(X_train, train_labels)
print(grid_search_dtcl.best_params_)
print('\n')

best_grid_dtcl = grid_search_dtcl.best_estimator_
print('BEST GRID:','\n',best_grid_dtcl)
```

```
{'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 20, 'min_samples_split': 60}


BEST GRID:
 DecisionTreeClassifier(max_depth=4, min_samples_leaf=20, min_samples_split=60,
                        random_state=1)
```

Regularising the Decision Tree & Generating Tree with best grid

```
reg_dt_model= DecisionTreeClassifier(criterion = 'gini', max_depth =4,min_samples_leaf=20,min_samples_split=60)
reg_dt_model.fit(X_train, train_labels)
```

```
DecisionTreeClassifier(max_depth=4, min_samples_leaf=20, min_samples_split=60)
```

Gernerating Tree with best grid

```
DTcart_tree_regularized = open('d:\DTcart_tree_regularized.dot','w')
dot_data = tree.export_graphviz(reg_dt_model, out_file= DTcart_tree_regularized , feature_names = list(df_cart), class_names = li

DTcart_tree_regularized.close()
```

Decision Tree after pruning

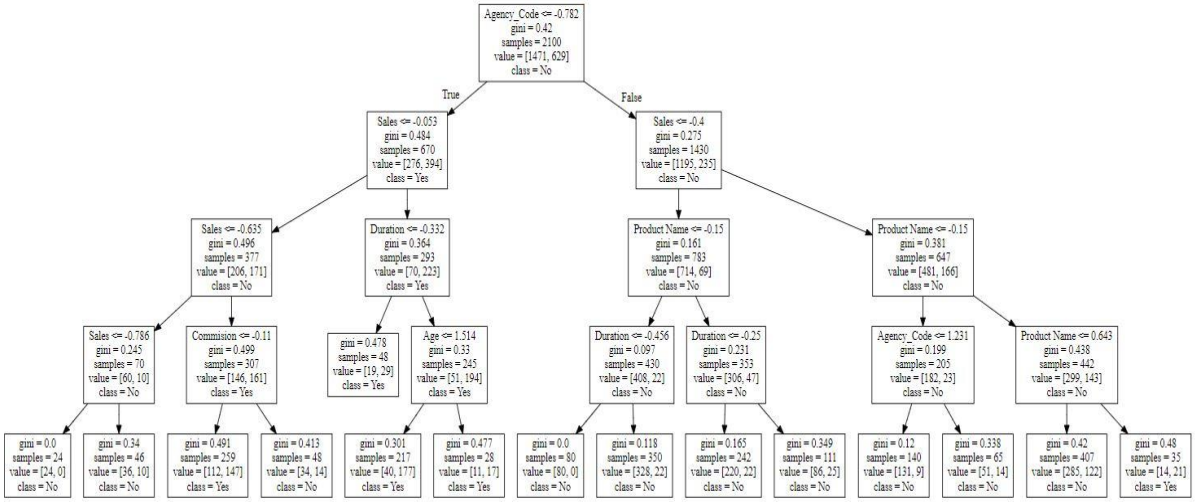
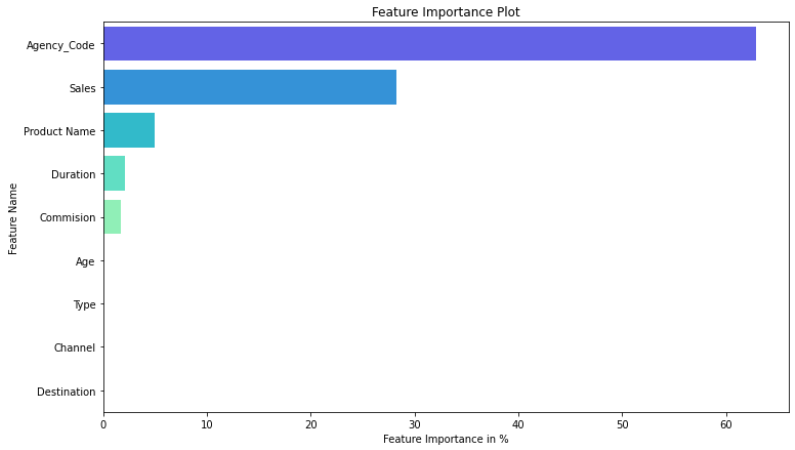


Figure 2.2.1 Decision Tree after pruning

Variable Importance



	Imp
Agency_Code	0.612996
Sales	0.256406
Product Name	0.077342
Commision	0.022786
Duration	0.022499
Age	0.007971
Type	0.000000
Channel	0.000000
Destination	0.000000

Figure 2.2.2 Decision Tree- Variable Importance

From the above important parameters, the model highly depends upon at "Agency Code" i.e 61.3% and "Sales" i.e 25.64%

Predicting on Training and Test dataset

ytrain_predict = reg_dt_model.predict(X_train)

ytest_predict = reg_dt_model.predict(X_test)

Getting the Predicted Probabilities

```
Predicted Probabilities of Test data
```

	0	1
0	0.935714	0.064286
1	0.432432	0.567568
2	0.432432	0.567568
3	0.184332	0.815668
4	0.937143	0.062857

Table 2.2.1 Pred Probs of DT Test

Model 2: Building a Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
```

```
rfcl = RandomForestClassifier(n_estimators = 200,random_state=1)
rfcl = rfcl.fit(X_train, train_labels)
```

```
rfcl
```

```
RandomForestClassifier(n_estimators=200, random_state=1)
```

To get the best set of hyperparameters we can use Grid Search. Grid Search passes all combinations of hyperparameters one by one into the model and check the result. Finally, it gives us the set of hyperparameters which gives the best result after passing in the model.

Finding Best Parameters using GridSearch

```
param_grid = {
    'max_depth': [4,5,6],
    'max_features': [4,5,6],
    'min_samples_leaf': [8,10,15],
    'min_samples_split': [40,50,60],
    'n_estimators': [300,]#100,300
}

rfcl = RandomForestClassifier(random_state=1)

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

- **random_state:** int, RandomState instance or None, optional (default=None). Random state ensures that the splits that you generate are reproducible. Scikit-learn uses random permutations to generate the splits. The random state that you provide is used as a seed

to the random number generator. This ensures that the random numbers are generated in the same order.

- **cv:** number of cross-validation you have to try for each selected set of hyperparameters.
- **max_depth:** int or None, optional (default=None). The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure. The max_depth value should be adjusted according no to high/low to avoid overfitting/underfitting.
- **min_samples_split:** int, float, optional (default=2). min_samples_split is used to control over-fitting. depending on the level of underfitting or overfitting, you can tune the values for min_samples_split.
- **min_samples_leaf:** int, float, optional (default=1). The minimum number of samples required to be at a leaf node. min_samples_leaf is also used to control over-fitting by defining that each leaf has more than one element. Thus ensuring that the tree cannot overfit the training dataset.
- **max_features:** These are the maximum number of features Random Forest is allowed to try in individual tree. Increasing max_features generally improve the performance of the model as at each node now we have a higher number of options to be considered.
- **n_estimators:** This is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher number of trees give you better performance but makes your code slower. You should choose as high value as your processor can handle because this makes your predictions stronger and more stable.

Fitting the model using the optimal values from Grid Search

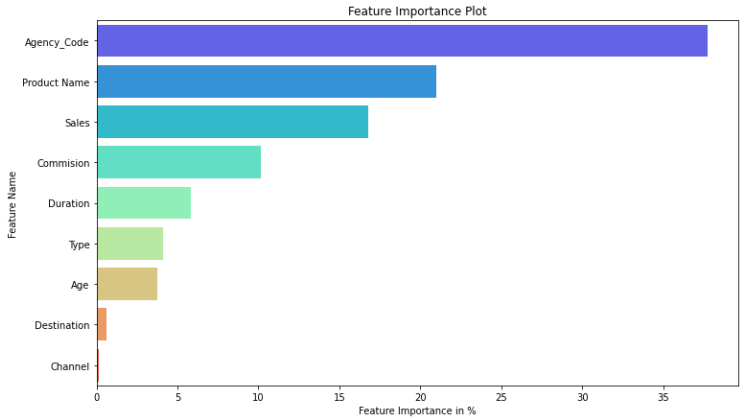
```
grid_search_rfcl.fit(X_train, train_labels)
print(grid_search_rfcl.best_params_)
print('\n')

best_grid_rfcl = grid_search_rfcl.best_estimator_
print('BEST GRID:','\n',best_grid_rfcl)
```

```
{'max_depth': 5, 'max_features': 5, 'min_samples_leaf': 8, 'min_samples_split': 50, 'n_estimators': 300}


BEST GRID:
 RandomForestClassifier(max_depth=5, max_features=5, min_samples_leaf=8,
                        min_samples_split=50, n_estimators=300, random_state=1)
```

Variable Importance



	Imp
Agency_Code	0.377515
Product Name	0.209599
Sales	0.167533
Commision	0.101448
Duration	0.058145
Type	0.041071
Age	0.037318
Destination	0.006171
Channel	0.001200

Figure 2.2.3 Random Forest – Variable Importance

From the above important parameters, the model highly depends upon at "Agency Code" i.e 37.75% and "Product Name" i.e 20.95%

Predicting on Training and Test dataset

ytrain_predict_rfcl = best_grid_rfcl.predict(X_train)

ytest_predict_rfcl = best_grid_rfcl.predict(X_test)

Getting the Predicted Probabilities

```
Predicted Probabilities of Test data
```

	0	1
0	0.754774	0.245226
1	0.497115	0.502885
2	0.471964	0.528036
3	0.246076	0.753924
4	0.932714	0.067286

Table 2.2.2 RF Pred Probs

Model 3: Building a Neural Network Classifier

```
clf = MLPClassifier(hidden_layer_sizes=100, max_iter=5000, #200,1000
                    solver='sgd', verbose=True,  random_state=1,tol=0.01)
```

```
# Fit the model on the training data

clf.fit(X_train, train_labels)
```

```
Iteration 1, loss = 0.73190828
Iteration 2, loss = 0.70481842
Iteration 3, loss = 0.67377712
Iteration 4, loss = 0.64619472
Iteration 5, loss = 0.62272023
Iteration 6, loss = 0.60330223
Iteration 7, loss = 0.58732612
Iteration 8, loss = 0.57422402
Iteration 9, loss = 0.56310704
Iteration 10, loss = 0.55382327
Iteration 11, loss = 0.54599574
Iteration 12, loss = 0.53911673
Iteration 13, loss = 0.53335898
Iteration 14, loss = 0.52839899
Iteration 15, loss = 0.52391237
Iteration 16, loss = 0.51999360
Iteration 17, loss = 0.51649432
Iteration 18, loss = 0.51346008
Iteration 19, loss = 0.51074817
Iteration 20, loss = 0.50822930
Training loss did not improve more than tol=0.010000 for 10 consecutive epochs. Stopping.

MLPClassifier(hidden_layer_sizes=100, max_iter=5000, random_state=1,
              solver='sgd', tol=0.01, verbose=True)
```

train =77 accu,auc 78, test = 74 accu auc 75

Finding Best Parameters using GridSearch

```
param_grid_nncl = {
    'hidden_layer_sizes': [200],#350
    'activation': ['logistic', 'relu'],
    'solver': ['sgd', 'adam'],
    'tol': [0.1,0.01],#0.1
    'max_iter' : [300]#250,300
}

nncl = MLPClassifier(random_state=1)

grid_search_nncl = GridSearchCV(estimator = nncl, param_grid = param_grid_nncl, cv = 3)
```

- **random_state:** int, RandomState instance or None, optional (default=None). Random state ensures that the splits that you generate are reproducible. Scikit-learn uses random permutations to generate the splits. The random state that you provide is used as a seed to the random number generator. This ensures that the random numbers are generated in the same order.
- **cv:** number of cross-validation you have to try for each selected set of hyperparameters.
- **hidden_layer_sizes** : This parameter allows us to set the number of layers and the number of nodes we wish to have in the Neural Network Classifier. Each element in the tuple represents the number of nodes at the ith position where i is the index of the tuple. Thus the length of tuple denotes the total number of hidden layers in the network.
- **max_iter:** It denotes the number of epochs.
- **activation:** The activation function for the hidden layers.
- **solver:** This parameter specifies the algorithm for weight optimization across the nodes.
- **tol:** tolerance for the stopping criteria. This tells scikit to stop searching for a minimum (or maximum) once some tolerance is achieved, i.e. once you're close enough.

Fitting the model using the optimal values from Grid Search

```
grid_search_nncl.fit(X_train, train_labels)
print(grid_search_nncl.best_params_)
print('\n')

best_grid_nncl = grid_search_nncl.best_estimator_
print('BEST GRID:','\n',best_grid_nncl)
```

```
{'activation': 'relu', 'hidden_layer_sizes': 200, 'max_iter': 300, 'solver': 'adam', 'tol': 0.01}


BEST GRID:
 MLPClassifier(hidden_layer_sizes=200, max_iter=300, random_state=1, tol=0.01)
```

Predicting on Training and Test dataset

ytrain_predict_nncl = best_grid_nncl.predict(X_train)

ytest_predict_nncl = best_grid_nncl.predict(X_test)

Getting the Predicted Probabilities

```
Predicted Probabilities of Test data
```

	0	1
0	0.387241	0.612759
1	0.930650	0.069350
2	0.351693	0.648307
3	0.758975	0.241025
4	0.717447	0.282553

Table 2.2.3 NN Pred Probs

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.

Answer:

		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

TP: True Positives, these are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes.

TN: True Negatives, these are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no.

FP: False Positives, when actual class is no and predicted class is yes.

FN: False Negatives, when actual class is yes but predicted class in no.

Accuracy: Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best.

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

Precision: Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

Precision = TP/TP+FP

Recall (Sensitivity) : Recall is the ratio of correctly predicted positive observations to the all observations in actual class – yes

Recall = TP/TP+FN

F1 score: It is the harmonic mean of precision and recall. It takes both false positive and false negatives into account.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Metrics for Predictions on the Training set of Decision Tree

[Accuracy Score,Precision,Recall,f1-score, Classification Report, Confusion Matrix]

```
Accuracy  for DecisionTreeClassifier model is 0.793
Precision for DecisionTreeClassifier model is 0.67
Recall    for DecisionTreeClassifier model is 0.62
f1-score  for DecisionTreeClassifier model is 0.64

Classification report for DecisionTreeClassifier model is
              precision    recall  f1-score   support

           0       0.84      0.87      0.85      1471
           1       0.67      0.62      0.64       629

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

Confusion Matrix for DecisionTreeClassifier model is

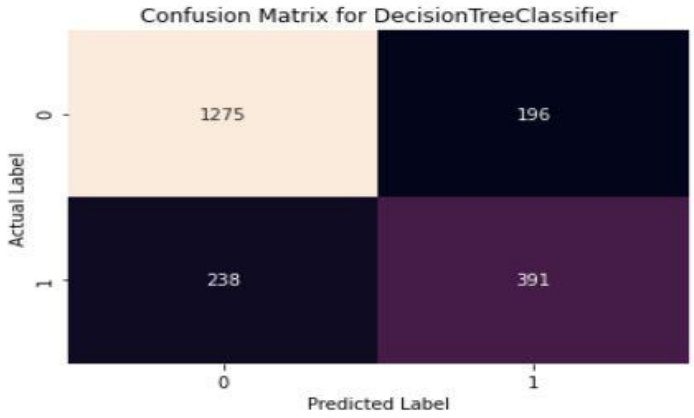


Figure 2.3.1 DT- Train data Confusion Matrix

Score of AUC and Plotting roc_curve for Training set of Decision Tree

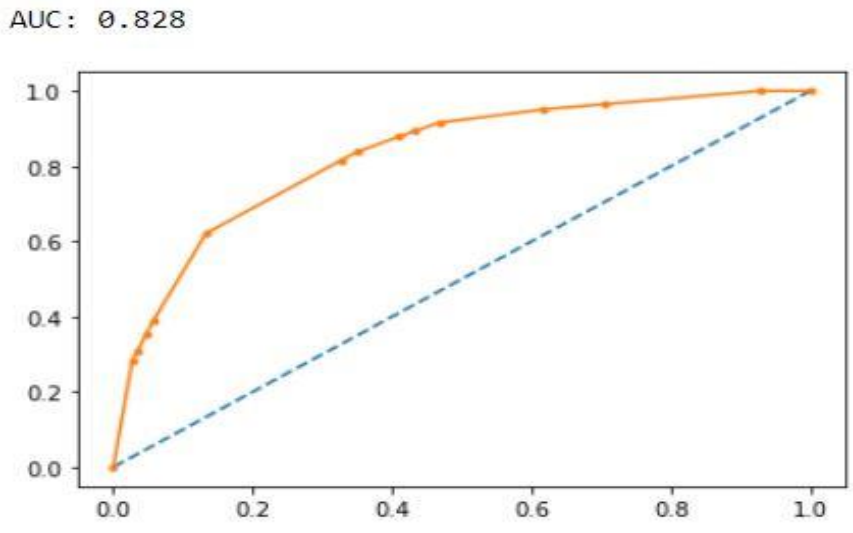


Figure 2.3.2 DT- Train data ROC curve

Metrics for Predictions on the Testing set of Decision Tree

[Accuracy Score,Precision,Recall,f1-score, Classification Report, Confusion Matrix]

```
Accuracy  for DecisionTreeClassifier model is 0.773
Precision for DecisionTreeClassifier model is 0.71
Recall    for DecisionTreeClassifier model is 0.53
f1-score  for DecisionTreeClassifier model is 0.6
```

```
Classification report for DecisionTreeClassifier model is
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       605
           1       0.71      0.53      0.60       295

    accuracy                           0.77       900
   macro avg       0.75      0.71      0.72       900
weighted avg       0.77      0.77      0.76       900
```

```
Confusion Matrix for DecisionTreeClassifier model is
```

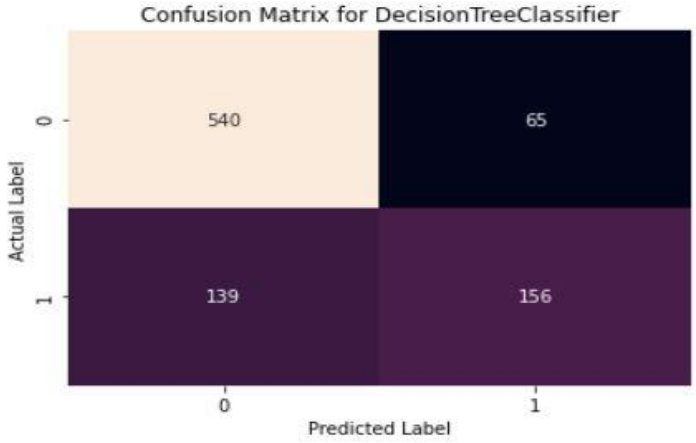


Figure 2.3.2 DT- Test data Confusion Matrix

Score of AUC and Plotting roc_curve for Testing set of Decision Tree

```
AUC: 0.790
```

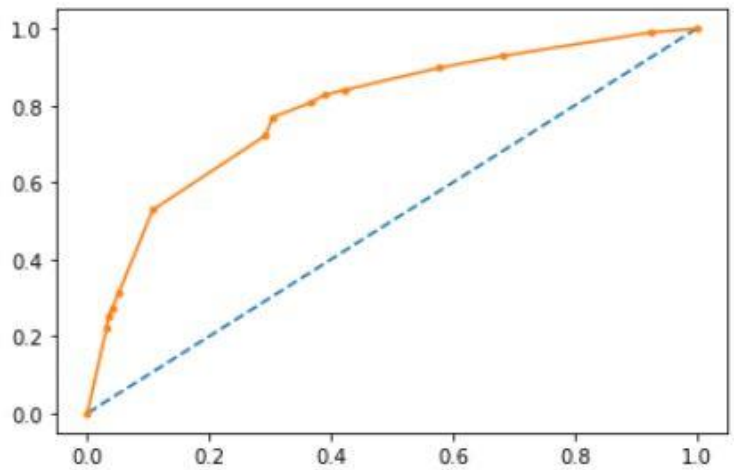


Figure 2.3.3 DT- Test data ROC curve

Decision Tree Conclusion

Train Data:		Test Data:	
AUC	83%	AUC	79%
Accuracy	79%	Accuracy	77%
Precision	67%	Precision	71%
Recall	62%	Recall	53%
f1-score	64%	f1-score	60%

Metrics for Predictions on the Training set of Random Forest

[Accuracy Score,Precision,Recall,f1-score, Classification Report, Confusion Matrix]

```
Accuracy  for RandomForestClassifier model is 0.806
Precision for RandomForestClassifier model is 0.71
Recall    for RandomForestClassifier model is 0.61
f1-score  for RandomForestClassifier model is 0.65
```

```
Classification report for RandomForestClassifier model is
              precision    recall  f1-score   support

           0       0.84      0.89      0.87      1471
           1       0.71      0.61      0.65       629

    accuracy                           0.81      2100
   macro avg       0.77      0.75      0.76      2100
weighted avg       0.80      0.81      0.80      2100
```

```
Confusion Matrix for RandomForestClassifier model is
```

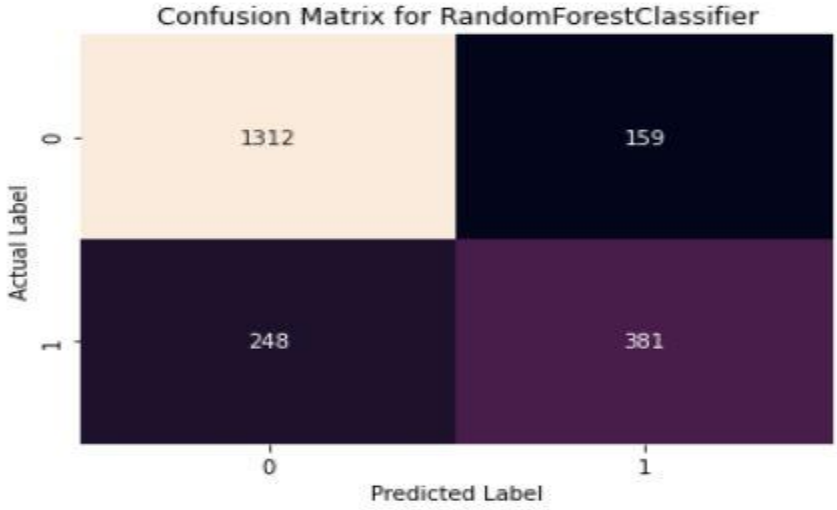


Figure 2.3.4 RF- Train data Confusion Matrix

Score of AUC and Plotting roc_curve for Training set of Random Forest

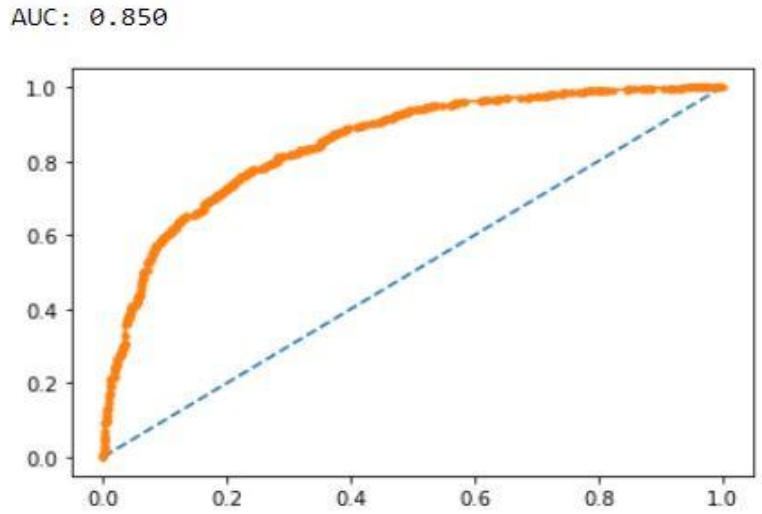


Figure 2.3.5 RF- Train data ROC curve

Metrics for Predictions on the Testing set of Random Forest

[Accuracy Score,Precision,Recall,f1-score, Classification Report, Confusion Matrix]

```
Accuracy  for RandomForestClassifier model is 0.781
Precision for RandomForestClassifier model is 0.74
Recall    for RandomForestClassifier model is 0.51
f1-score  for RandomForestClassifier model is 0.61


Classification report for RandomForestClassifier model is
               precision    recall  f1-score   support

           0       0.79      0.91      0.85       605
           1       0.74      0.51      0.61       295

    accuracy                           0.78       900
   macro avg       0.77      0.71      0.73       900
weighted avg       0.78      0.78      0.77       900


Confusion Matrix for RandomForestClassifier model is
```

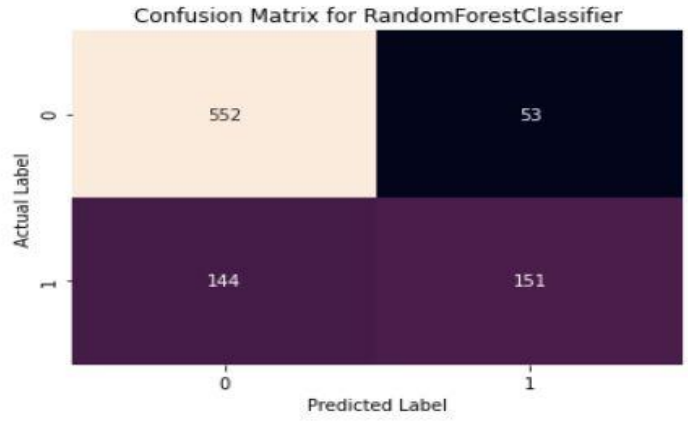


Figure 2.3.6 RF- Test data Confusion Matrix

Score of AUC and Plotting roc_curve for Testing set of Random Forest

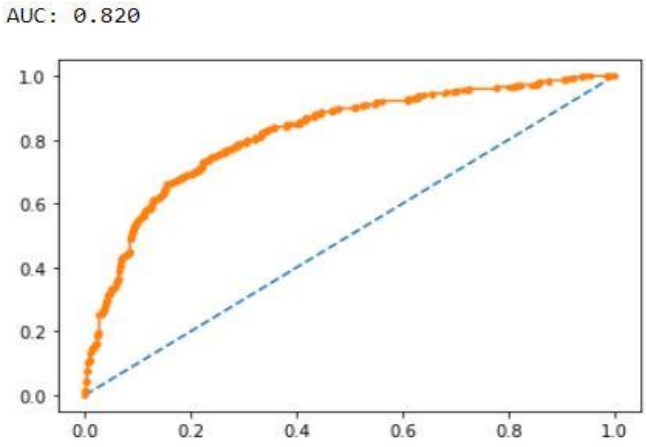


Figure 2.3.7 RF- Test data ROC curve

Random Forest Conclusion

Train Data:		**Test Data:**	
AUC	85%	AUC	82%
Accuracy	80%	Accuracy	78%
Precision	71%	Precision	74%
Recall	61%	Recall	51%
f1-score	65%	f1-score	61%

Metrics for Predictions on the Training set of Neural Network

[Accuracy Score,Precision,Recall,f1-score, Classification Report, Confusion Matrix]

```
Accuracy  for ANN model is 0.776
Precision for ANN model is 0.67
Recall    for ANN model is 0.51
f1-score  for ANN model is 0.57
```

```
Classification report for ANN model is
              precision    recall  f1-score   support

           0       0.81      0.89      0.85      1471
           1       0.67      0.51      0.57       629

    accuracy                           0.78      2100
   macro avg       0.74      0.70      0.71      2100
weighted avg       0.77      0.78      0.77      2100
```

```
Confusion Matrix for ANN model is
```

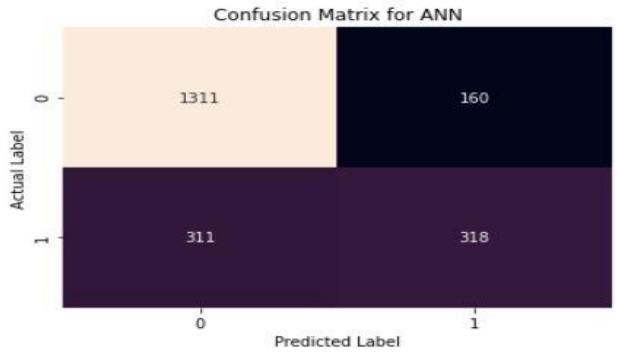


Figure 2.3.8 NN – Train data Confusion Matrix

Score of AUC and Plotting roc_curve for Training set of Neural Network

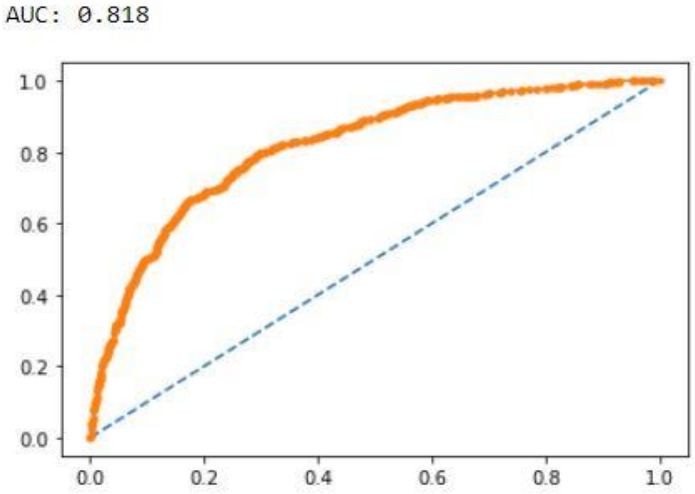


Figure 2.3.9 NN- Train data ROC curve

Metrics for Predictions on the Testing set of Neural Network

[Accuracy Score,Precision,Recall,f1-score, Classification Report, Confusion Matrix]

```
Accuracy  for ANN model is 0.76
Precision for ANN model is 0.67
Recall    for ANN model is 0.51
f1-score  for ANN model is 0.57
```

```
Classification report for ANN model is
              precision    recall  f1-score   support

           0       0.81      0.89      0.85      1471
           1       0.67      0.51      0.57       629

    accuracy                           0.78      2100
   macro avg       0.74      0.70      0.71      2100
weighted avg       0.77      0.78      0.77      2100
```

```
Confusion Matrix for ANN model is
```

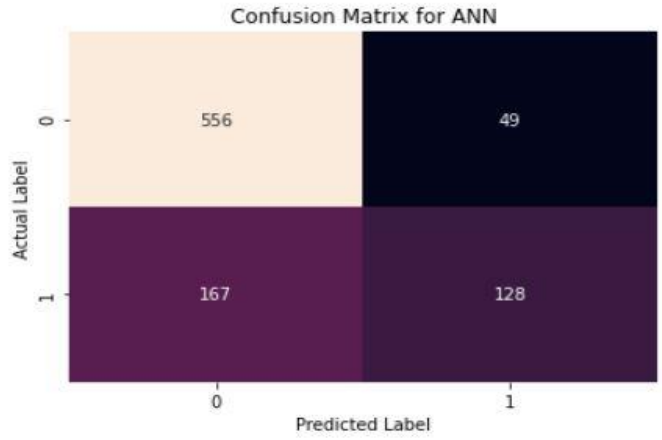


Figure 2.3.10 NN- Test data Confusion Matrix

Score of AUC and Plotting roc_curve for Testing set of Neural Network

```
AUC: 0.804
```

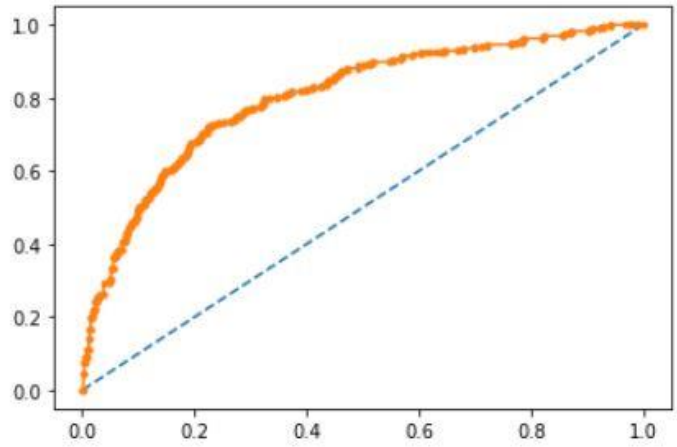


Figure 2.3.2 NN - Test data ROC curve

Neural Netwok Conclusion

Train Data:		**Test Data:**	
AUC	82%	AUC	80%
Accuracy	78%	Accuracy	76%
Precision	67%	Precision	67%
Recall	51%	Recall	51%
f1-score	57%	f1-score	57%

Inferences:

In our business scenario, we have to reduce the claims frequencies coming in the insurance firm. Here we are considering accepted claim as '1' and denied claim as '0'.

As per the business insights there is increase in claim frequency. Algorithm should be designed in such a way that it should not give False Positive, i.e it should not predict claims wrongly. So, in total we want to reduce False positive. As we know Precision is inversely proportional to False Positive. When False positive reduces, Precision increases. In our model we give more importance to Precision. From the above models test data we can find that the precision in high. It concludes that our model is good one.

But if we look in a business perspective, increase in the number of False Negative (predicting the actual cases as negative), will affect the reputation of the firm as the claims were not properly claimed. So here comes the importance of F1 score, where we give the equal weightage to both precision and recall.

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

Answer:

	CART Train	CART Test	Random Forest Train	Random Forest Test	Neural Network Train	Neural Network Test
Accuracy	0.79	0.77	0.81	0.78	0.78	0.76
AUC	0.83	0.79	0.85	0.82	0.82	0.80
Recall	0.62	0.53	0.61	0.51	0.51	0.51
Precision	0.67	0.71	0.71	0.74	0.67	0.67
F1 Score	0.64	0.60	0.65	0.61	0.57	0.57

Table 2.4 Model Comparison

ROC curve: The ROC curve is a plot of True Positive Rate (TPR) on the y-axis vs False Positive Rate (FPR) on the x-axis. The more that the ROC curve hugs the top left corner of the plot, the better the model does at classifying the data into categories. To quantify this, we can calculate the **AUC** (area under the curve) which tells us how much of the plot is located under the curve. The closer AUC is to 1, the better the model.

ROC Curve for the 3 models on the Train data

```
Area under the curve for Decision Tree Classification Model is 0.8284453326041682
Area under the curve for Random Forest Classification Model is 0.8495886016780168
Area under the curve for Artificial Neural Network Model is 0.8182460262477858
```

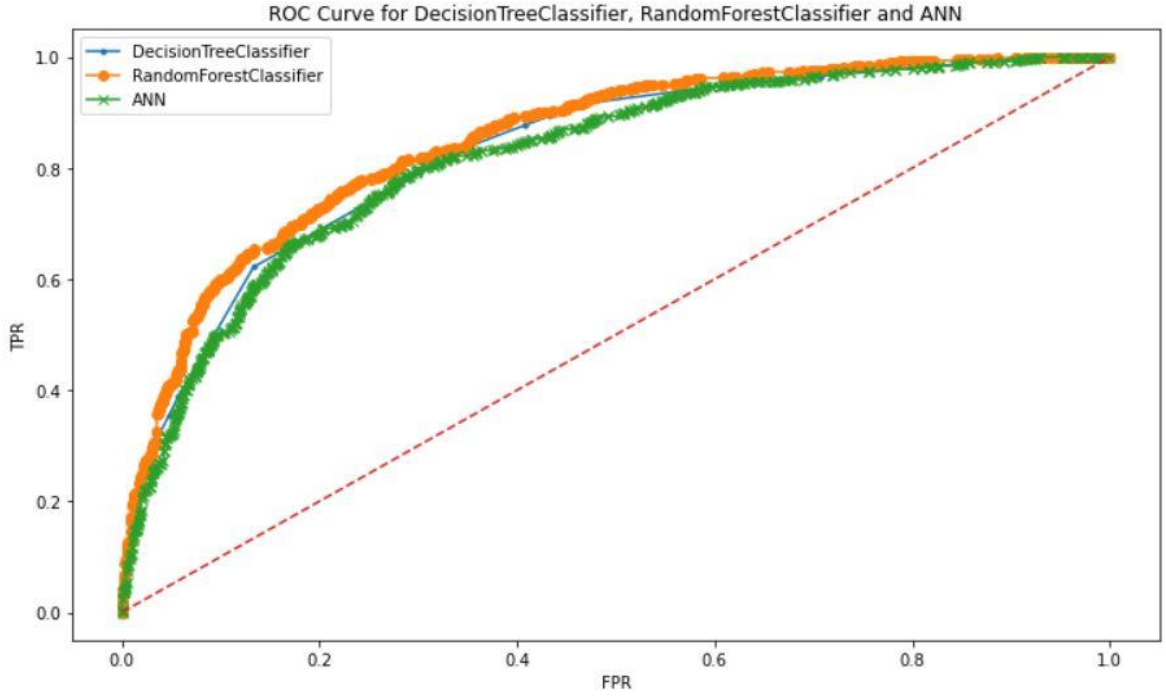


Figure 2.4.1 Model Comparision ROC curve for Train data

ROC Curve for the 3 models on the Test data

```
Area under the curve for Decision Tree Classification Model is 0.7904188261661298
Area under the curve for Random Forest Classification Model is 0.82003641966662
Area under the curve for Artificial Neural Network Model is 0.8038464770976328
```

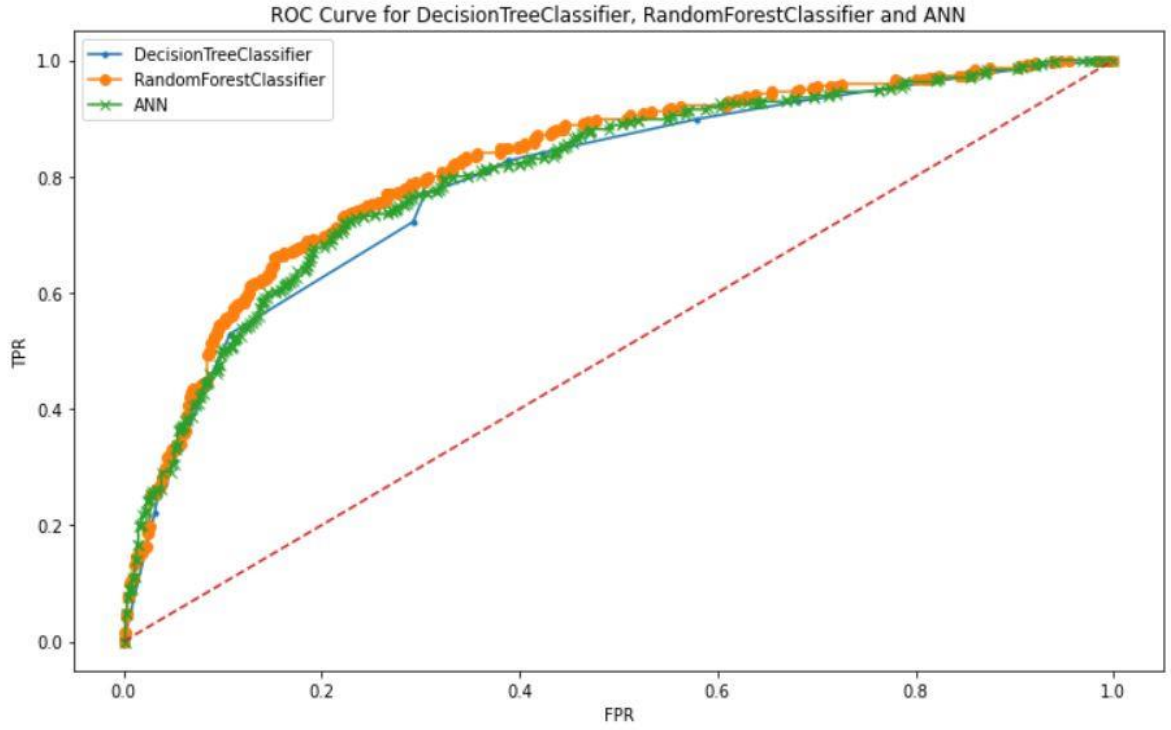


Figure 2.4.1 Model Comparision ROC curve for Test data

Conclusion

In our model we give more importance to Precision. From the above models test data we can find that the precision in high. It concludes that our model is good one.

Also, from the above ROC cure and AUC values we can find that all the models are performing well. Here, I am selecting the **RF model**, as it has better accuracy, precision, recall, and f1 score, AUC than other two CART & NN

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

Answer:

Business Insights

- When we look at the model, more data will help us to better understand and predict models.
- Customers benefited from streamlining online experiences, which resulted in a rise in conversions and, as a result, increased profitability.
- As per the data, 90% of insurance is done by channel – Online. Need to find why almost all offline business has claim associated.

- From the dataset we can find Agency JZI sales in on low side, need to provide proper training to the agency to increase the sales also need to run promotional marketing campaigns. Else, need to find another agency which can perform well.
- Based on the model we get accuracy almost 80%. So based on the claim data pattern of customers booking of tickets, plans we need to sell the insurance plans.
- From the plot found that more sales happened through Agency than Alirlines and the trend shows claim are processed through Airline, need to check the process and understand the workflow why it's happening.

Recommendations

- Increase customer satisfaction which will give more revenue
- Reduce claim handling costs
- Optimize claim recovery method
- Reduce fraudulent transactions and take immediate steps to avoid fraudulent transactions.

----------END OF REPORT----------