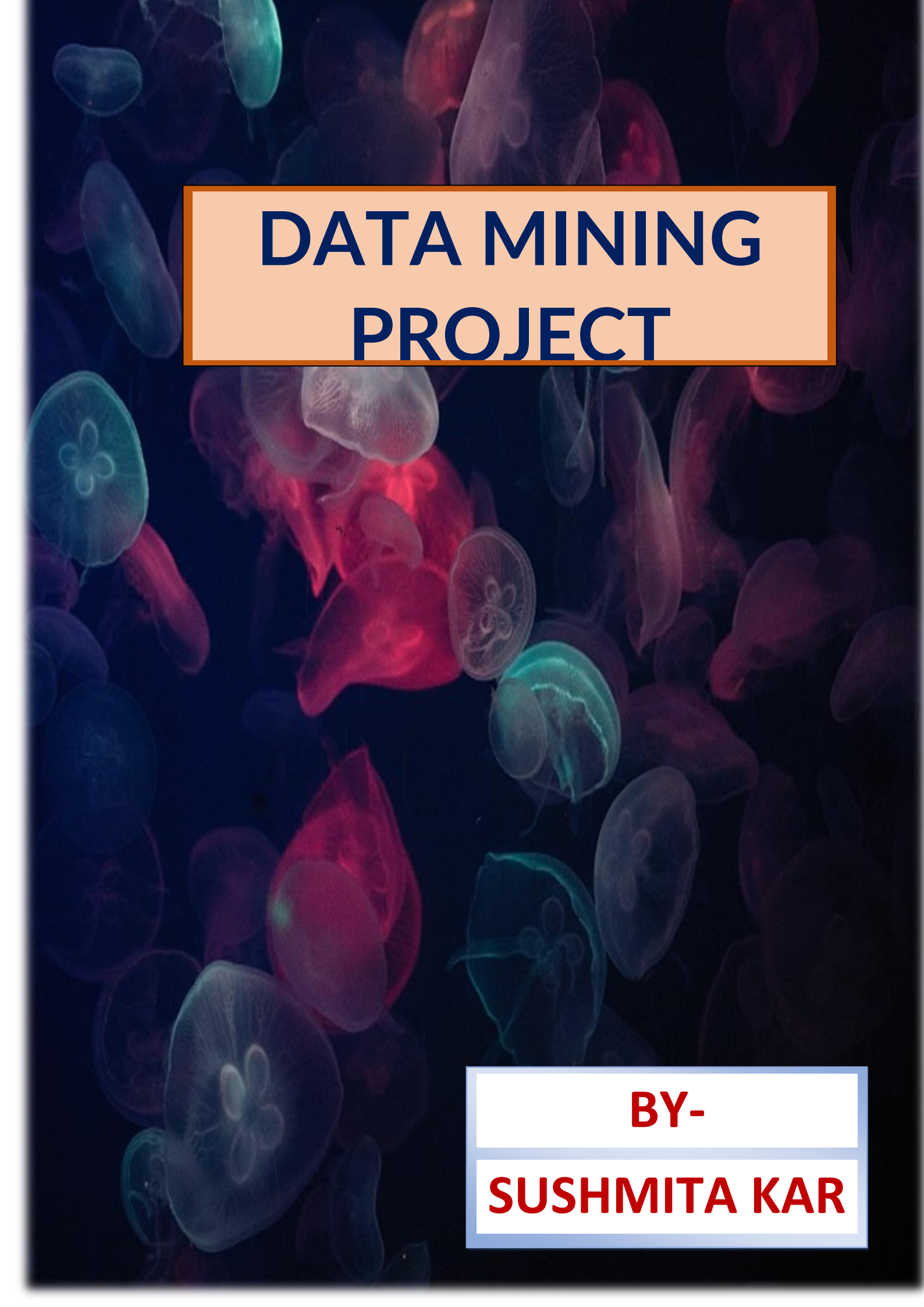


DATA MINING PROJECT

BY-

SUSHMITA KAR

Contents

List of Figure

Table

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.

Data Dictionary for Market Segmentation:

1. spending: Amount spent by the customer per month (in 1000s)
2. advance_payments: Amount paid by the customer in advance by cash (in 100s)
3. probability_of_full_payment: Probability of payment done in full by the customer to the bank
4. current_balance: Balance amount left in the account to make purchases (in 1000s)
5. credit_limit: Limit of the amount in credit card (10000s)
6. min_payment_amt : minimum paid by the customer while making payments for purchases made monthly (in 100s)
7. max_spent_in_single_shopping: Maximum amount spent in one purchase (in 1000s)

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

Dataset 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	6.55
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	5.89	3.694	2.068	5.837

There are 210 rows and 7 columns in the dataset.

Column	Non-Null Count	Dtype
spending	210 non-null	float64
advance_payments	210 non-null	float64
probability_of_full_payment	210 non-null	float64
current_balance	210 non-null	float64
credit_limit	210 non-null	float64
min_payment_amt	210 non-null	float64
max_spent_in_single_shopping	210 non-null	float64

From the above info of the data, we can infer that there are no null values in the dataset. All the variables in the dataset are continuous. Also, no duplication of data is recorded.

	count	mean	std	min	25%	50%	75%	max
spending	210	14.85	2.91	10.59	12.27	14.36	17.3	21.18
advance_payments	210	14.56	1.31	12.41	13.45	14.32	15.72	17.25
probability_of_full_payment	210	0.87	0.02	0.81	0.86	0.87	0.89	0.92
current_balance	210	5.63	0.44	4.9	5.26	5.52	5.98	6.68
credit_limit	210	3.26	0.38	2.63	2.94	3.24	3.56	4.03
min_payment_amt	210	3.7	1.5	0.77	2.56	3.6	4.77	8.46
max_spent_in_single_shopping	210	5.41	0.49	4.52	5.04	5.22	5.88	6.55

From above 5-point Summary, there are 210 customers in the data.
customer maximum spending from credit card is 21,800 to minimum spending of 10,590.
credit card limit of customer varies from minimum amount of 26,300 to maximum limit of 66,800.
There are customer who are spending minimum amount of rs. 4520 to maximum amt of rs. 6550 in single purchase.
The current credit card balance of customer varies from rs. 4900 to rs. 6680.
81% to 92% probability of customer are doing full payment towards credit card expenses.
Also, minimum amount customer is paying towards the credit card expense is rs.770 to rs.846
Currently the credit card balance maintained by the customer is minimum of 4900 to maximum of 6680.
The mean and the median appear close to each other. Showing the dataset has symmetrical distribution.

Data Visualization

Univariate Analysis/Bivariate Analysis/Multivariate Analysis

Visualization techniques are used for creating diagrams, images or animations to communicate a message. The key to perform this analysis is generating insights /inferences align with the business problems.

Spending- Amount spent by the customer per month (in 1000s)

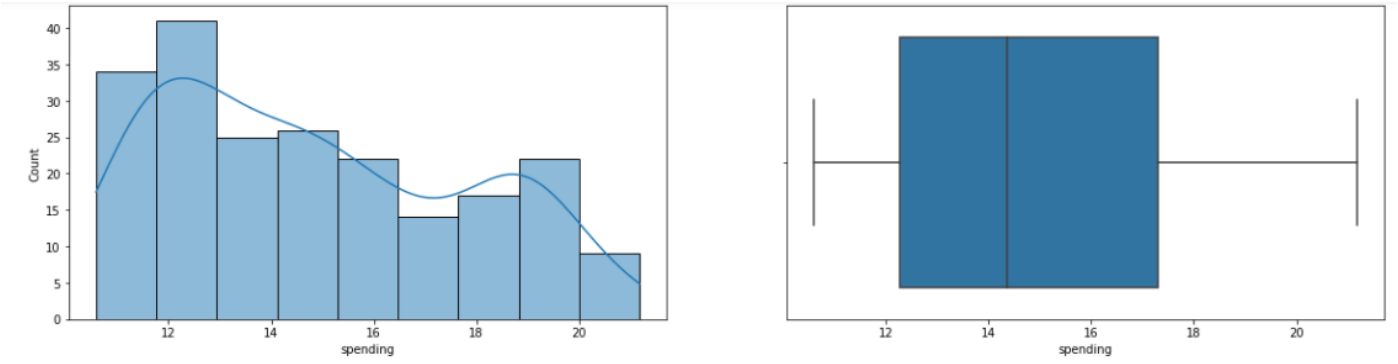


Figure1: Variable “spending”

Advance Payment- Amount paid by the customer in advance by cash (in 100s)

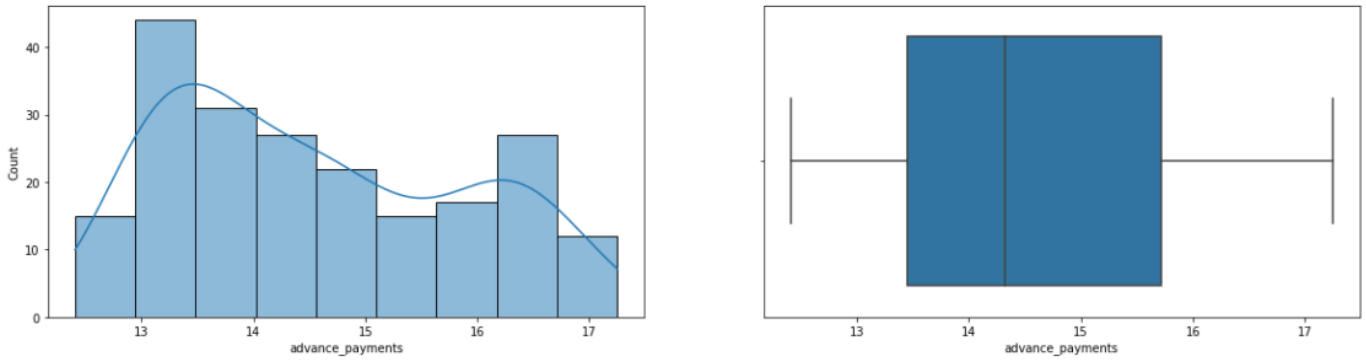


Figure2: Variable “advance_payment”

probability_of_full_payment: Probability of payment done in full by the customer to the bank

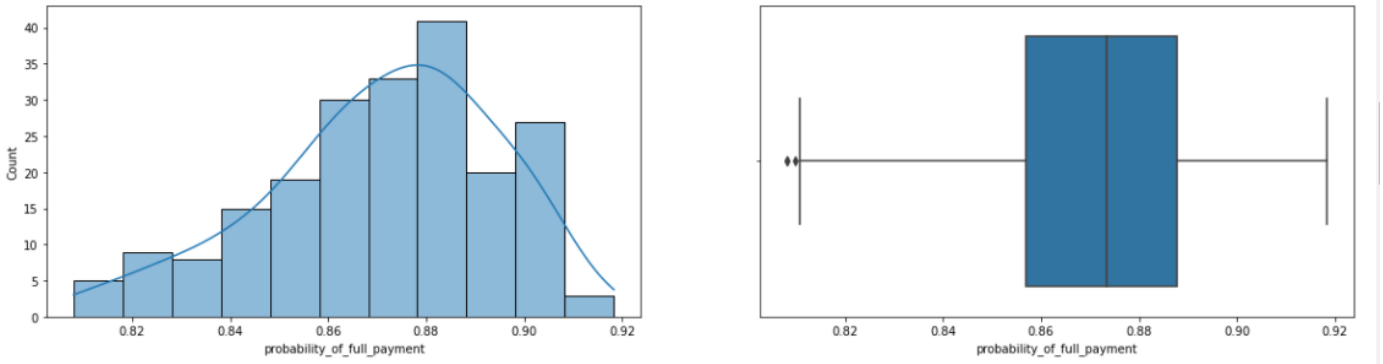


Figure3: Variable “probability_of_full_payment”

current_balance: Balance amount left in the account to make purchases (in 1000s)

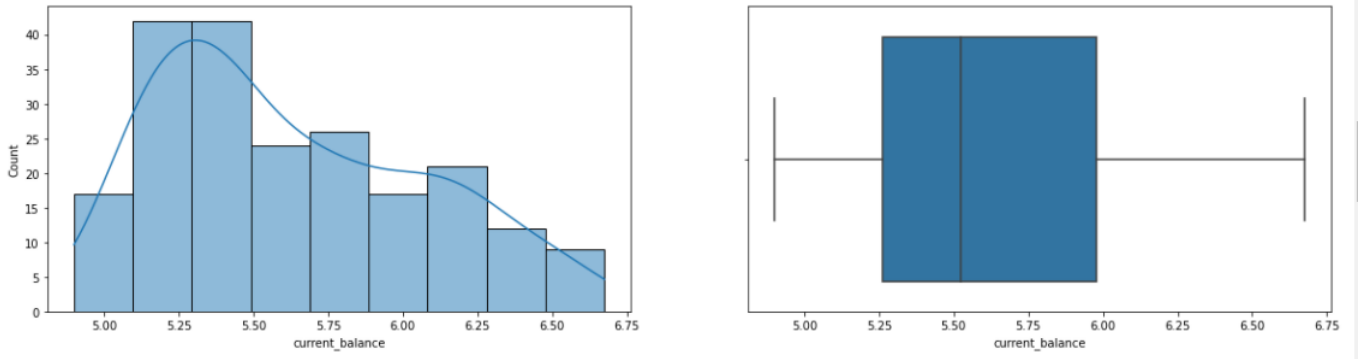


Figure4: Variable “current_balance”

credit_limit: Limit of the amount in credit card (10000s)

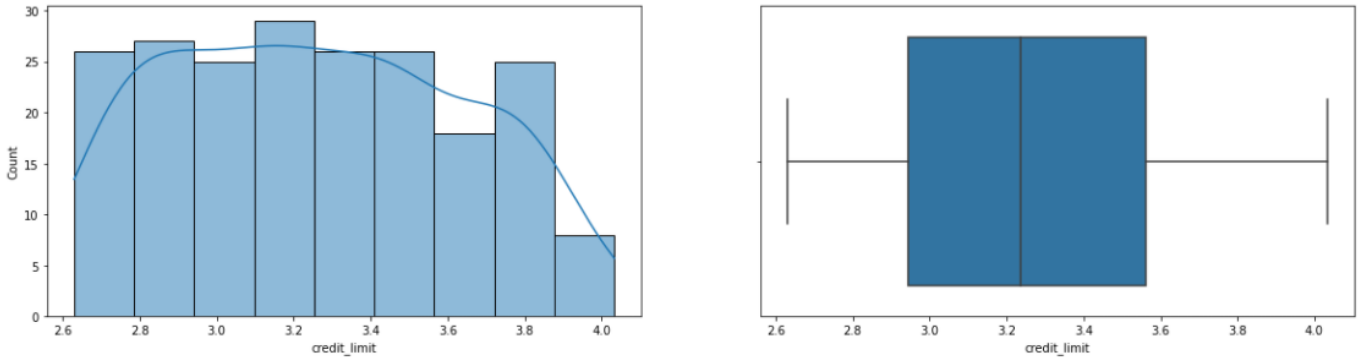


Figure 5: Variable "credit_limit"

min_payment_amt : minimum paid by the customer while making payments for purchases made monthly (in 100s)

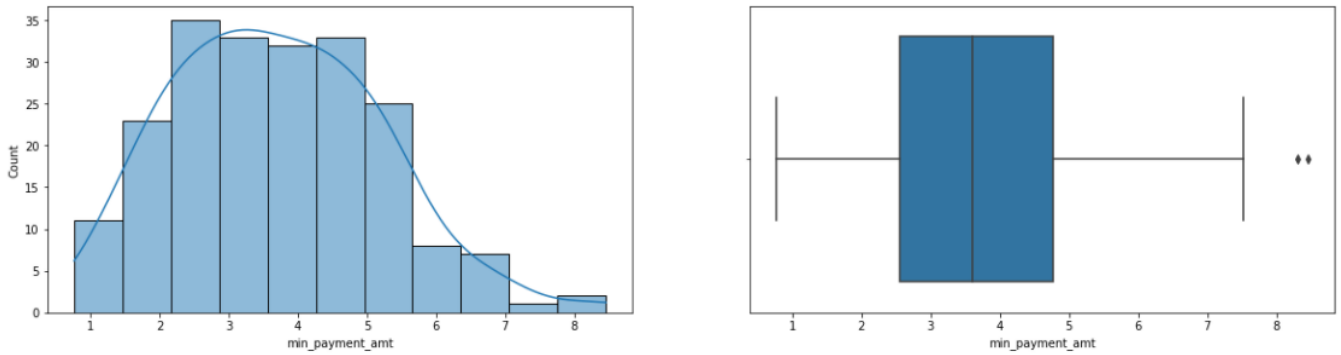


Figure 6: Variable "min_payment_amt"

max_spent_in_single_shopping: Maximum amount spent in one purchase (in 1000s)

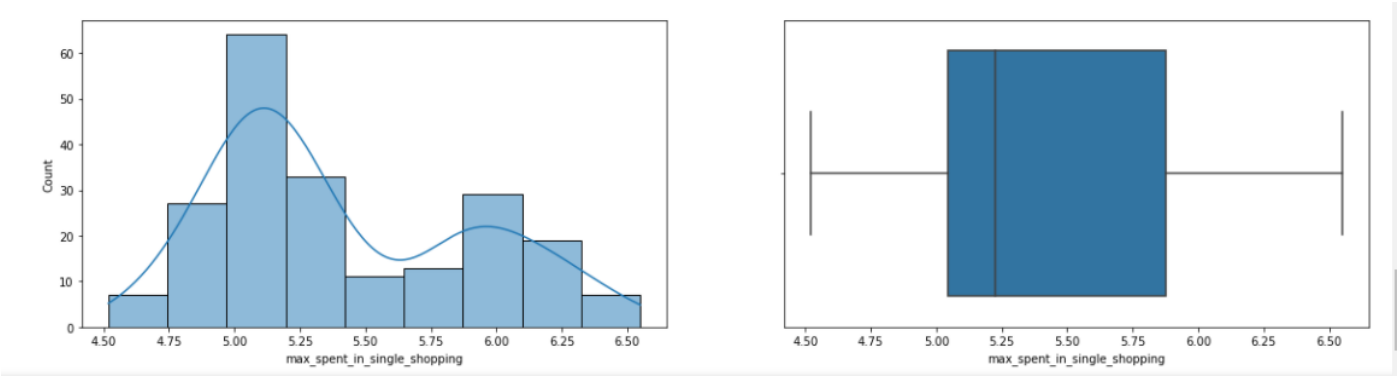


Figure 7: Variable "max_spent_in_single_shopping"

From above plots we have generated for the data it shows that spending, advance_payment, current_balance, credit_limit, and max_spent_in_single_shoppings variables are positively skewed.

Probability_of_full_payment is negatively skewed and is showing outliers.

Min_payment_amt is positively skewed and too shows outliers.

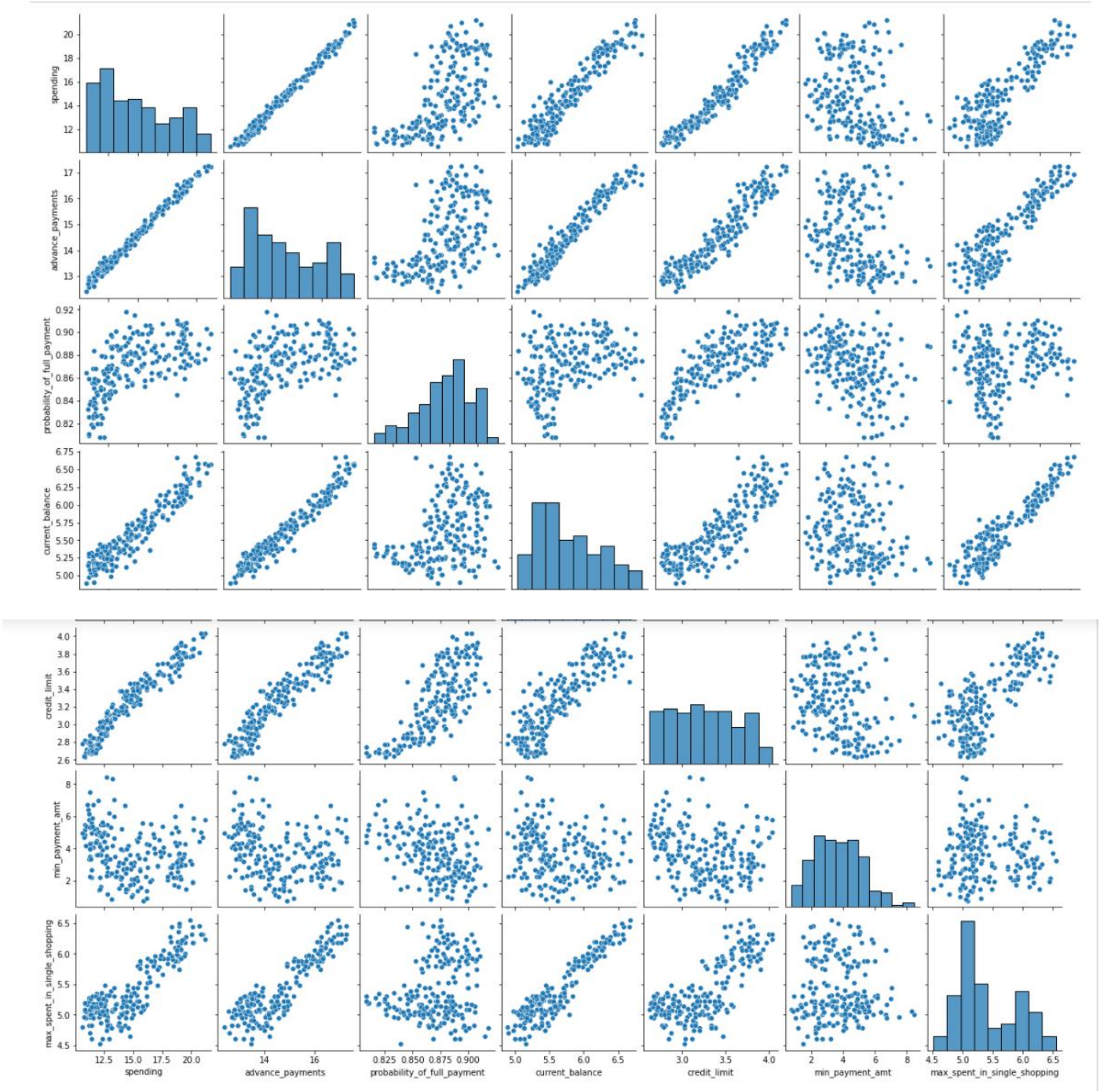


Figure 8: Pairplot of the dataset

The pairs plot is built on two basic figures, the histogram and the scatter plot. The histogram on the diagonal allows us to see the distribution of a single variable while the scatter plots on the upper and lower triangles show the relationship between two variables.

For instance, customer who spends more in a single time purchase has more monthly spending, likely to do more advance payment in cash towards the purchase. Also, their current balance and credit card limit is high.

So, from above plot we see a high correlation between spendings, advance_payments, current_balance, credit_limit, max_spent_in_single_shopping.

Figure 9: Correlation Heat Map

We can see high positive correlation among Following variables:

1. spending and advance_payment
2. spending and current_balance
3. spending and credit_limit
4. spending and max_spent_in_single_shopping
5. advance_payment and current_balance
6. advance_payment and credit_limit
7. advance_payment and max_spent-in_single_payment
8. current_balance and credit_limit
9. current_balance and max_spent_in_single_shopping
10. credit_limit and max_spent_in_single_shopping

Note: we saw outliers in probability_of_full_payment and min_payment_amt hence we treated the outliers as asked. Below is the boxplot after outlier treatment.

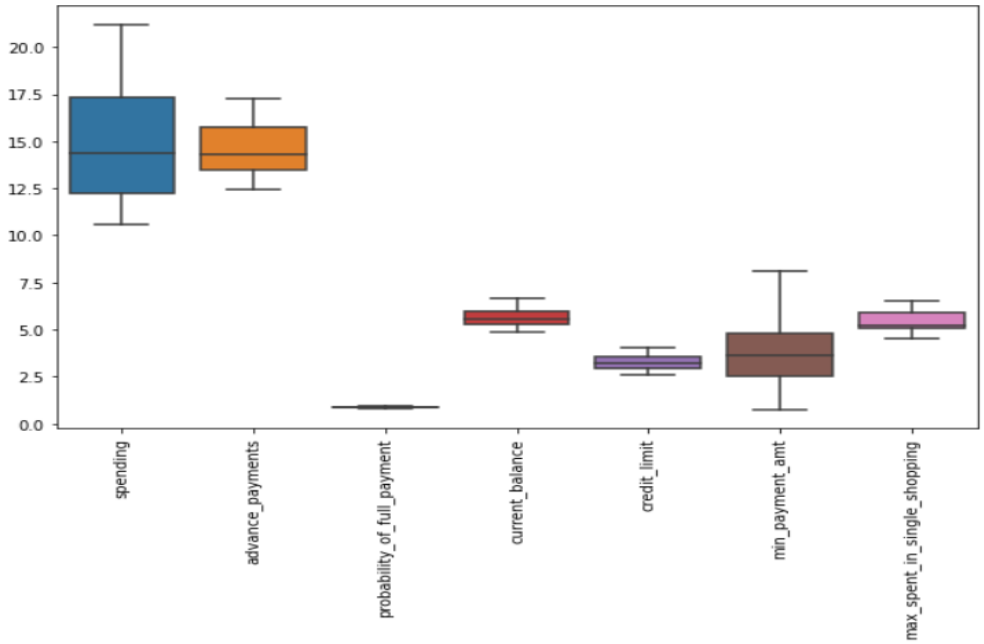


Figure 10: Outlier treated boxplot

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

Scaling of the data comes under the set of steps of data pre-processing when we are performing machine learning algorithms in the data set. As we know most of the supervised and unsupervised learning methods make decisions according to the data sets applied to them and often the algorithms calculate the distance between the data points to make better inferences out of the data.

In the machine learning algorithms if the values of the features are closer to each other there are chances for the algorithm to get trained well and faster instead of the data set where the data points or features values have high differences with each other will take more time to understand the data and the accuracy will be lower.

So, if the data in any conditions has data points far from each other, scaling is a technique to make them closer to each other or in simpler words, we can say that the scaling is used for making data points generalized so that the distance between them will be lower.

Hence, scaling is mandatory for this data set.

With the help of below library, we can scale the data.

```
from sklearn.preprocessing import StandardScaler
```

Below is the snapshot of Scaled data.

```
df_scaled.head()
```

Out[43]:

	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

As we can see from above data, after scaling all the values are transformed in such as way that the mean will be 0 and the standard deviation will be 1.

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

Hierarchical clustering is an unsupervised learning algorithm that is used to group together the unlabeled data points having similar characteristics. There are two types of Hierarchical Clustering: Agglomerative and Divisive. In Agglomerative clustering the data points are clustered using bottom-up approach starting with individual data points, while in Divisive clustering top-down approach is followed where all the data points are treated as one big cluster and the clustering process involves dividing the one big cluster into several small cluster. In this Project, we will perform agglomerative clustering.

The hierarchy of the clusters is represented as a dendrogram. A Dendrogram is a Treelike diagram that summarizes the process of clustering.

There are different ways to find the distance between the clusters. The distance itself can be Euclidean or Manhattan distance. Here we are using Euclidean distances formula to calculate the distance between each points. And to calculate the distance between these cluster we are going to use Linkage-ward method(analyse the variance of clusters). Hence creating the Dendrogram.

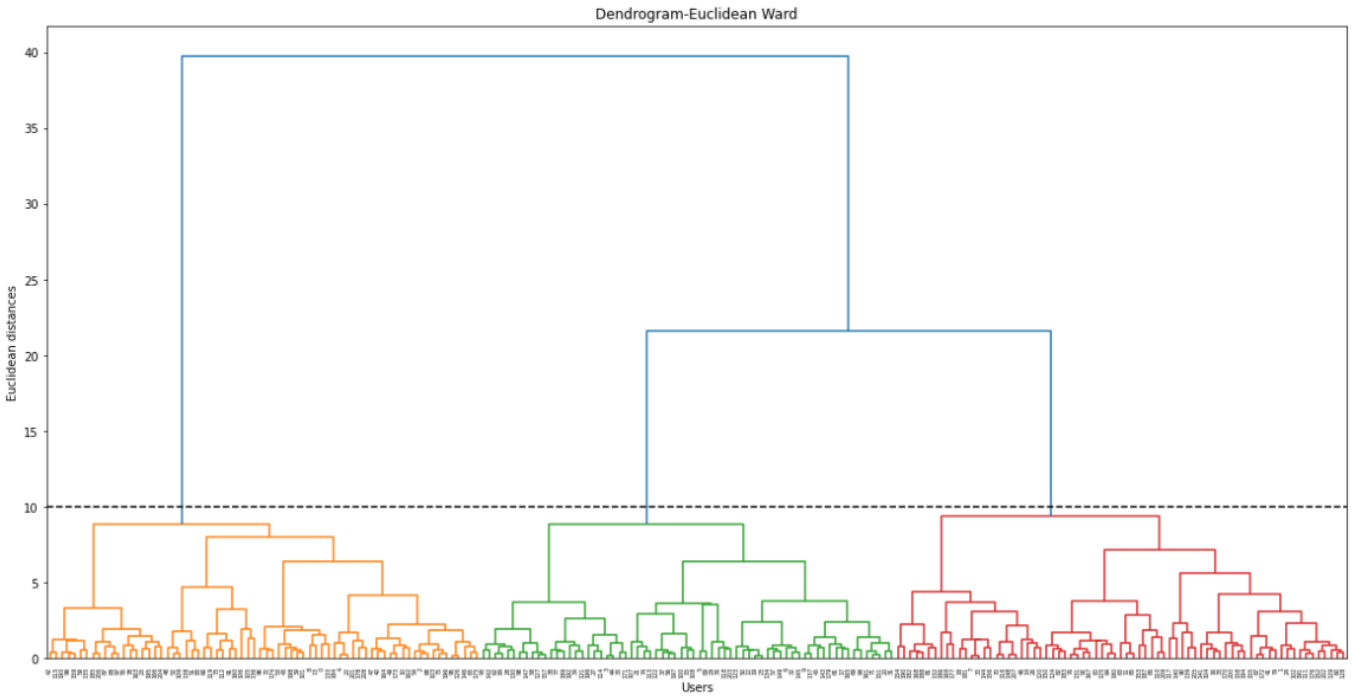


Figure 11.1: Dendrogram

In the above diagram, the x-axis contains the customer records and y-axis represents the distance between these records. The height of the horizontal lines tells us the distance at which this cluster is merged into another cluster. Vertical line tells us which cluster were part of merge forming a new cluster.

As we see that the above dendrogram is pretty big for 210 records, lets take a look at reasonable merges with the use of some other dendrogram().functions:

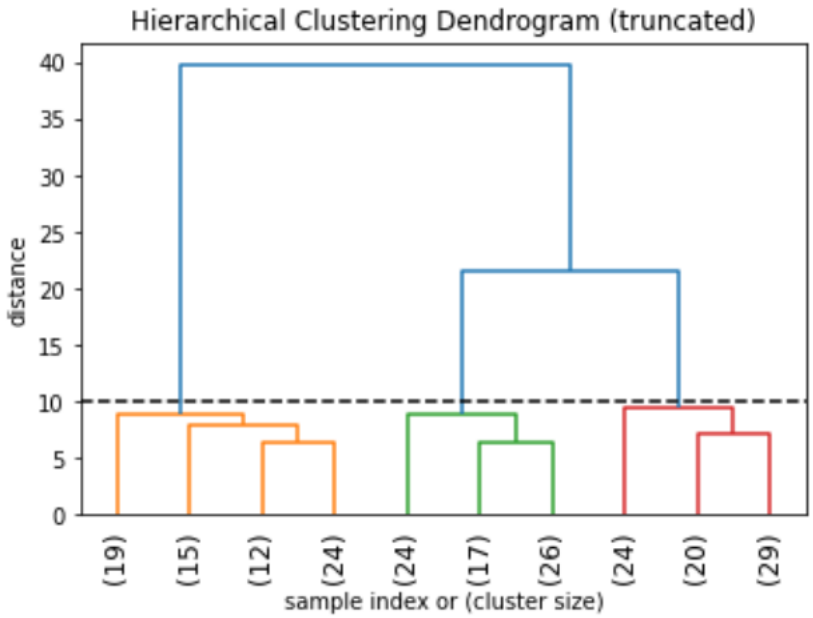


Figure 11.2: Truncated Dendrogram

The above diagram shows a truncated dendrogram, which only shows the last p=10 out of our 210 merges.

The above Dendrogram does not show the full-grown tree but just the number of Users in each clusters. Cluster size is not equal for all 12 clusters, as we can see from the above diagram.

The dotted vertical line is generally has the threshold in such a way that it cuts **the tallest vertical lines with maximum distance in the blue colour line**.

Hence, from our Dendrogram we can take decision of 3 Clusters for our Business segmentation.

After creating the above Linkage i.e. basically the distances where the merges has happened and we have visualized using the Linkage. Now, to obtain the clusters or the product which belong to these cluster for final verification, we will be forming the cluster using function "fcluster" from scientific package.

"fcluster" form flat cluster from the hierarchical clustering defined by the given linkage matrix.

The "criterion" parameter is used in forming flat clusters. Here we are using "maxclust" and below is the output:

```
clusters_1 = shc.fcluster(shc.linkage(df_scaled, method='ward'), 3, criterion='maxclust')
clusters_1
```

```
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)
```

2nd Method is using 'distance' criterion

```
## Method 2

clusters_2 = shc.fcluster(shc.linkage(df_scaled, method='ward'),10 , 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)
```

As we can see the 'maxclust' and 'distance' criterion both gives us the same output. So, these are just 2 ways, out of which we can derive the cluster from linkage method output.

From above output we can see the cluster labels from all of our data points. Since we have selected 3 clusters, we have 3 labels in the output i.e., 0,1,2.

Now, we can go ahead add these cluster in our original dataset and find out which variable belong to which cluster.

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	cluster
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

Our main goal is 'Profiling'. Profiling means how do the segments differentiate each other based on the variables. For instance, what is the average spending of each segment, and then on comparing segments, can

understand one group has a high spending, another has moderate and other has low spending. This way we can see that the group are differentiated based on spending or spending is segregating the segments.

cluster count	67	70	73
spending	11.872388	18.371429	14.199041
advance_payments	13.257015	16.145429	14.233562
probability_of_full_payment	0.848072	0.884400	0.879190
current_balance	5.238940	6.158171	5.478233
credit_limit	2.848537	3.684629	3.226452
min_payment_amt	4.949433	3.639157	2.612181
max_spent_in_single_shopping	5.122209	6.017371	5.086178
cluster count	67.000000	70.000000	73.000000

Cluster 1 -moderate

Cluster 2- low

Cluster 3- high

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

K means is one of the most popular Unsupervised Machine Learning Algorithms Used for Solving Classification Problems. ***K Means segregates the unlabeled data into various groups, called clusters, based on having similar features, common patterns***.

K, here is the pre-defined number of clusters to be formed by the Algorithm i.e., we select a value for k to decide the number of clusters to be formed.

The 'mean' in the k-mean refers to averaging of the data; that is, finding the centroid.

In other words, k-mean algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroid as small as possible.

Lets, compute k-means by considering n_cluster values as 1, 2, 3, 4, 5 and also check their inertia with the following code:

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

```
1469.999999999999
```

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

```
659.14740095485
```

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

```
430.298481751223
```

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

```
370.95298530006767
```

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

```
325.9446771140749
```

Inertia measures how well a dataset was clustered by k-means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster.

A good model is one with low Inertia and a low number of clusters. However, when k increases inertia decreases.

We can choose the right number of clusters with the help of the Within-Cluster-Sum-of-Square(WCSS) method. WCSS is the distances of the data points in each and every cluster from its centroid.

The main idea is to minimize the distance between the data points and the centroid of the clusters. The process is iterated until we reach a minimum value for the sum of distances as below:

```
wcss
```

```
[1469.999999999999,
 659.14740095485,
 430.298481751223,
 371.4400252695773,
 326.76866351004185,
 289.45524862464833,
 261.9484927636351,
 240.3798310478051,
 224.32762819205126,
 207.66534220339003]
```

To find the optimal K for a dataset, we will use the *Elbow method*; find the point where the decrease in inertia begins to slow I.e., The sharp point of bend or a point (looking like an elbow joint) of the plot like an arm, will be considered as the best/optimal value of K.

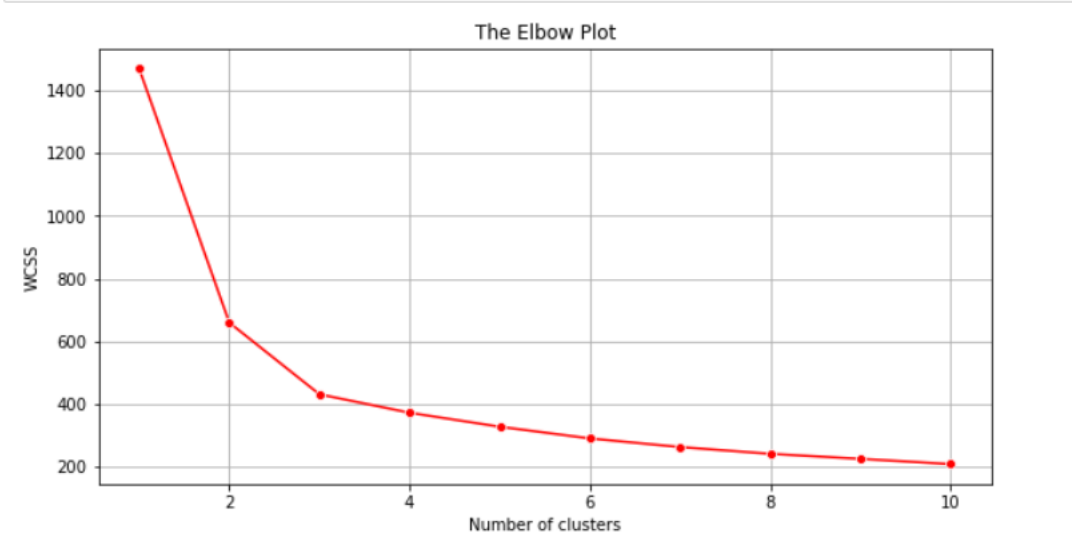


Figure 12 : K-mean Elbow Plot

K=3 is the “elbow” of this graph.

```
k_means = KMeans(n_clusters = 3)
k_means.fit(df_scaled)
labels_3 = k_means.labels_
labels_3
```

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

The above output shows us the k-mean cluster labels from all of our data points. Since we have selected 3 clusters, we have 3 labels in the output i.e., 0,1,2.

Let's profile the clusters, the way it was done in the hierarchical clustering to identify the suitable pattern for user segmentation after adding the k-mean cluster in the main dataset.

kmeans_cluster_3	0	1	2
spending	14.39	18.50	11.87
advance_payments	14.31	16.20	13.26
probability_of_full_payment	0.88	0.88	0.85
current_balance	5.51	6.18	5.24
credit_limit	3.25	3.70	2.85
min_payment_amt	2.70	3.63	4.78
max_spent_in_single_shopping	5.12	6.04	5.11
cluster	2.88	1.03	2.07
kcluster count	72.00	67.00	71.00

Cluster 0 -High

Cluster 1- low

Cluster 2- moderate

As in k-mean also our main goal is 'Profiling' i.e., how do the segments differentiate each other based on the variables, we analyse for instance, what is the average spending of each segment, and then on comparing segments, can understand one group has a high spending, another has moderate and other has low spending.

Cluster plot is a helpful way to look at the spread and overlap of clusters. Ideally, for a perfect algorithm, the clusters will be well separated with no (or minimum) overlap as below:

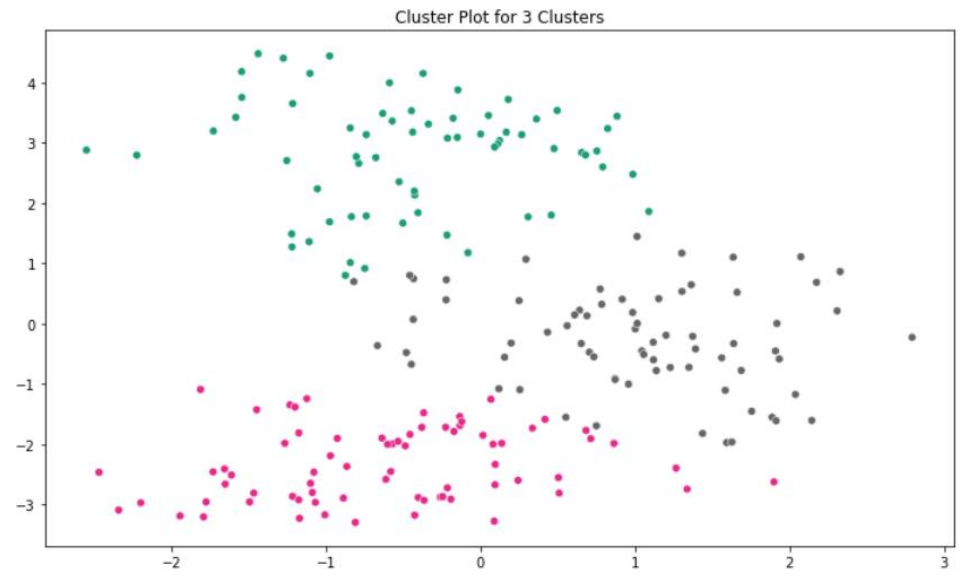


Figure 13: Cluster plot for 3 Cluster

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

Group / Cluster 1: : High Spending Group: In this group Customers spending is high.

Bank can promote more to these customer by giving them more credit points towards hotel booking, dinning, travelling and air mile offer, online shopping , electronics, home improvement, clothing, etc.

The max_spent_in_single_shopping variable is high in this group so bank can offer discounts/cash back to increase the probability_of_full_payment.

Also, increasing the credit_limit of customers.

Promoting foreclosure discount to customers for increasing the advance_payments made by the customer against their spendings.

To promote more spending Bank should also tie up with Luxury brands and promote for credit points.

Bank can also promote add-on cards for family member with no membership amounts.

Group/Cluster 2: Moderate Spending Group: They are potential customers who are paying bills and doing purchase and maintaining comparatively good credit score.

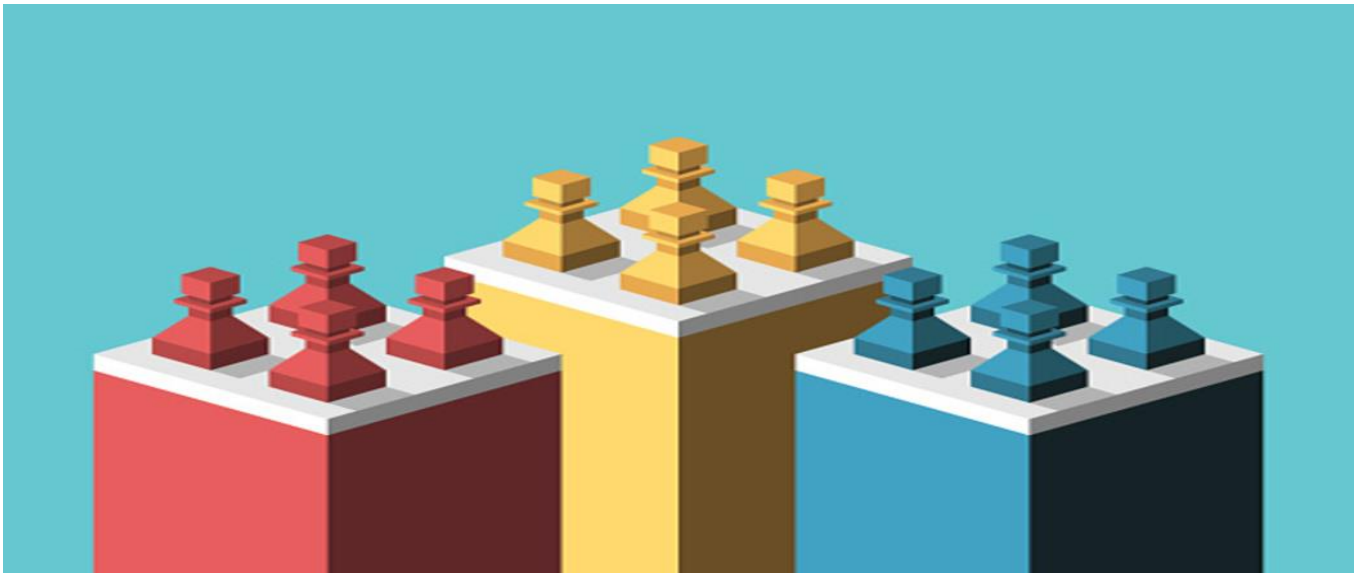
Bank can promote more by increasing the credit limit also decreasing interest rate on any loan/emi payment against the credit card. Promote add-on cards for family members 'One year membership amount free'.

To encourage spending more bank should promote more to these customers by giving them more credit points towards hotel booking, dinning, travelling and air mile offer, online shopping, electronics, home improvement, clothing, etc.

Group/Cluster 3: **Low Spending Group: Customer should be given remainders for payments and promote foreclosure discounts to promote advance_payment and min_payment_amt.**

Also, bank should promote more to these customers by giving them more credit points towards hotel booking, dinning, travelling and air mile offer, online shopping, electronics, home improvement, clothing, etc.

Also, should promote cash back and discounts for utilising the credit card purchase more efficiently.



Problem 2: CART-RF

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 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 in days)
7. Destination of the tour (Destination)
8. Amount worth of sales per customer in procuring tour insurance policies in rupees (in 100's)
9. The commission received for tour insurance firm (Commission is in the percentage of sales)
10. Age of insured (Age)

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

Some important libraries to be install for CART and RF analysis:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
```

Dataset 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

A Quick review of the Summary of the dataframe:

```
<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
```

Observation

- 10 variables
- Age, Commision, Duration, Sales are numeric variable
- rest are categorial variables
- 3000 records, no missing one
- 9 independent variable and one target variable – Clamied

*** The summary also shows there is NO NULL values.

Statistical description of dataframe:

	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

Observation:

- There are 6 categorical and 4 numeric variables.
- Variable ‘duration’ is showing min value as -1. This could be a data entry error.
- ‘Agency_Code’ EPX has frequency value of 1365.
- The most preferred ‘type’ seems to be Travel Agency.
- The most selected ‘Channel’ is Online.
- Also, the most preferred ‘Product Name’ is Customised Plan.
- And, the frequent travel destination is ASIA.

*After checking for Duplicates in the dataset, we see that there are 139 duplicate rows as below:

```
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

Note: No unique identifiers in the dataset above. These can be different customers so I will not drop them. Not treating the duplicate records here and keeping them as it is.

Displaying the Unique Counts of All Categorical Variables below:

```
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
```

```
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

Visualization of Numerical variable.

```
Age
Skew: 1.15
```

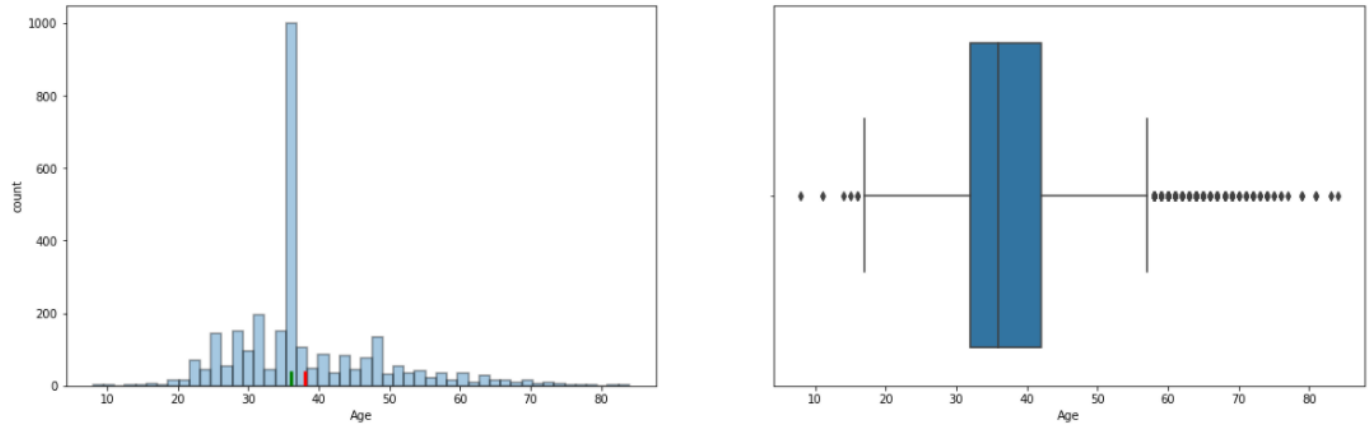


Figure 14: Variable 'Age'

Commision
Skew: 3.15

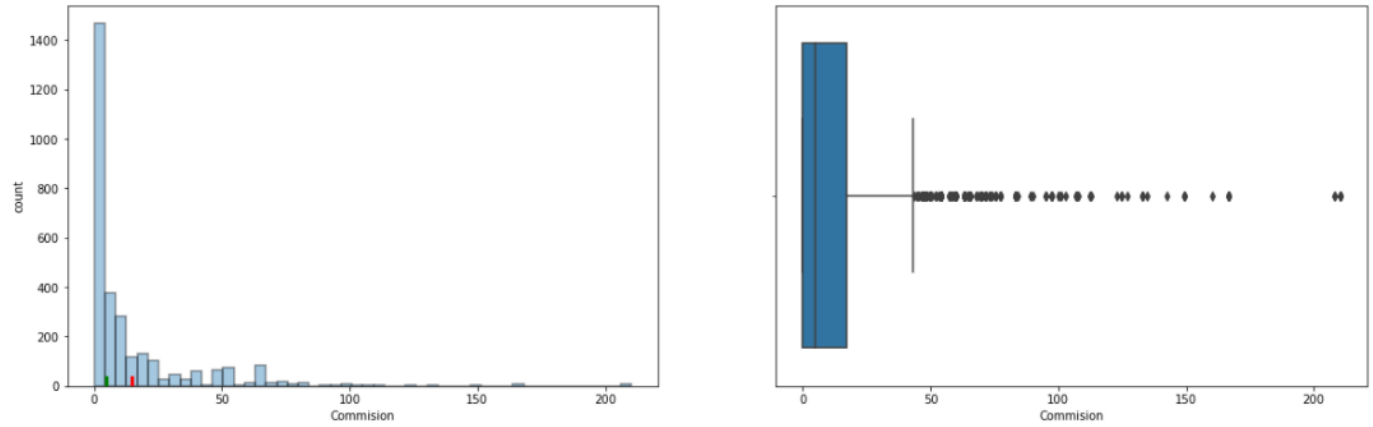


Figure 15: Variable 'Commision'

Duration
Skew: 13.78

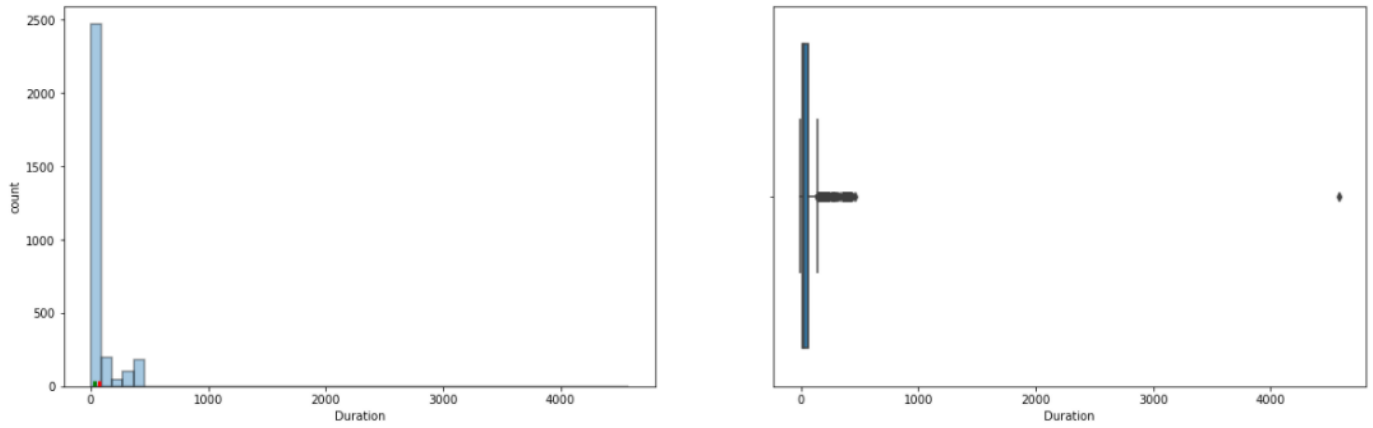


Figure 16: Variable 'Duration'

Sales
Skew: 2.38

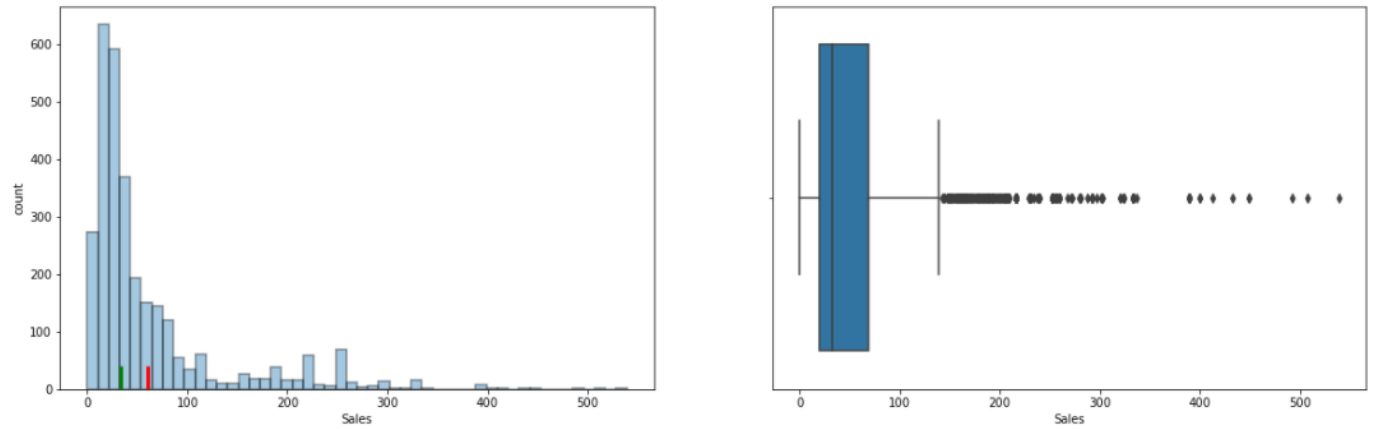


Figure 17: Variable 'Sales'

Observation:

All the above variables are rightly skewed.

Above continuous variable also shows outliers.

There are outliers in all the variables, but the 'sales' and 'commision' can be a genuine business value.

*Note: Random Forest and CART can handle the outliers. Hence, Outliers are not treated for now, we will keep the data as it is.

Visualizing Categorical Variables.

Agency_Code

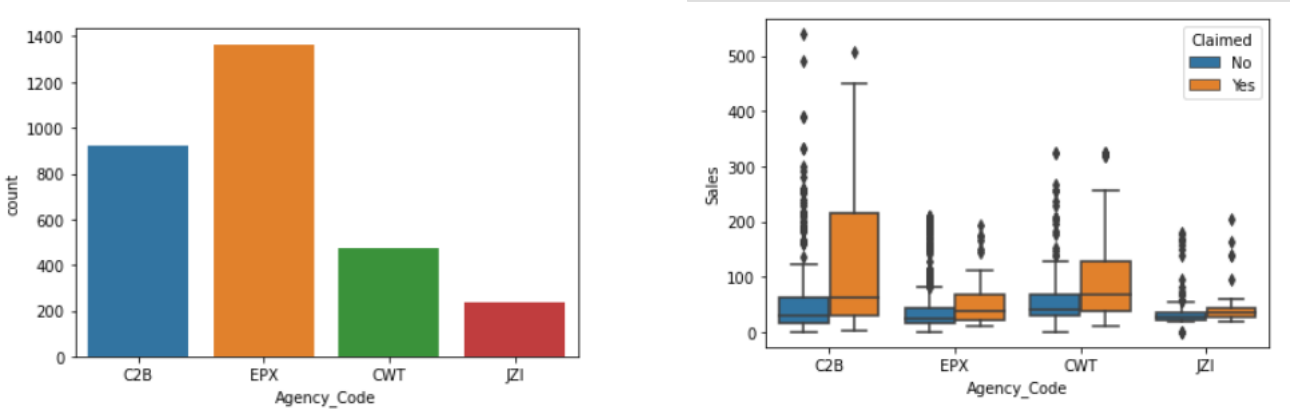


Figure 18: **Variable 'Agency_Code'**

-EPX in Agency_code is showing high count.

-In the boxplot above C2B is showing the highest claim.

Type

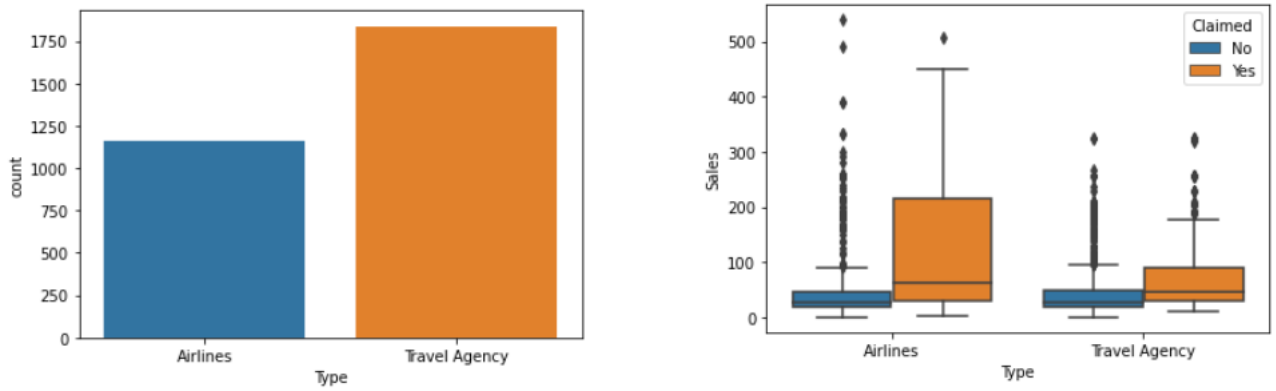


Figure 19: Variable 'Type'

-Travel Agency count is high

-Airline shows high claims than Travel Agency.

Channel

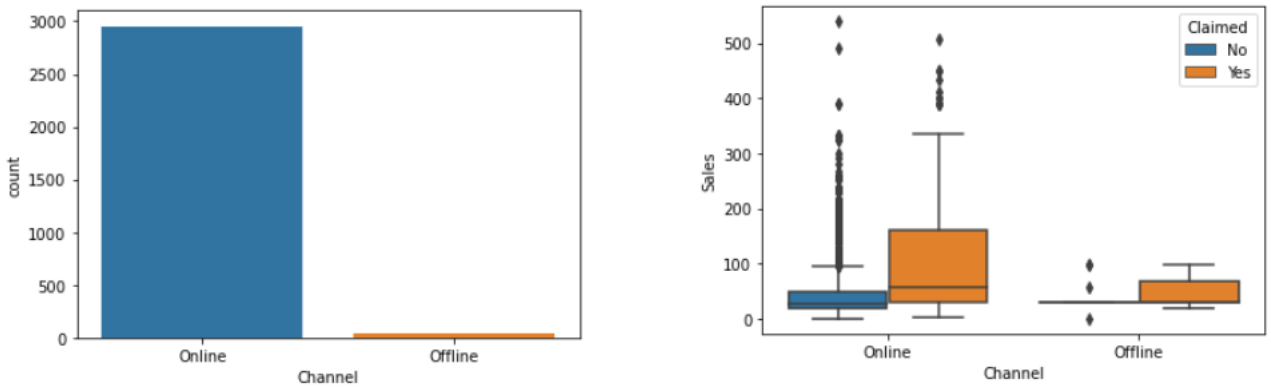


Figure 20: Variable 'Channel'

-Customer use Online channel more than offline.

-Boxplot shows sales split for both the channel and Online claim is more.

Product Name

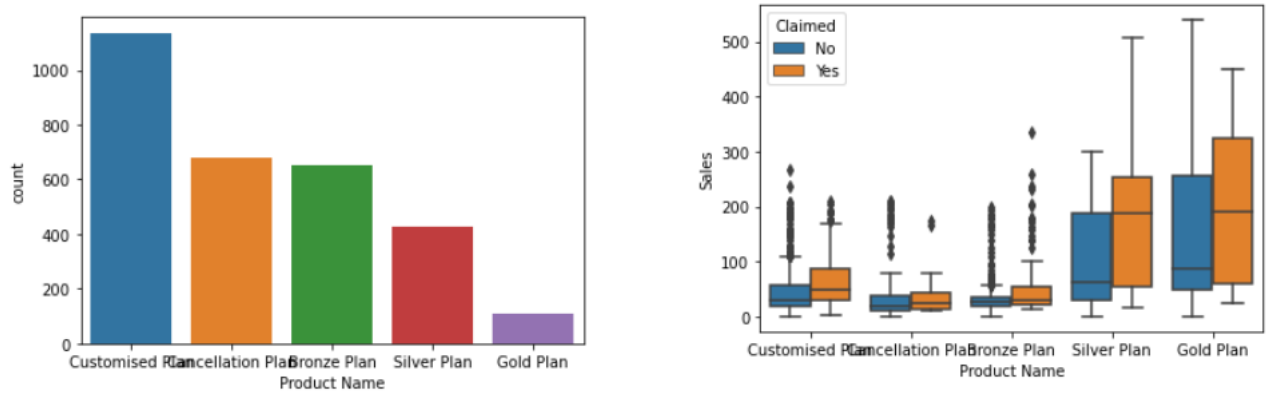


Figure 21: Variable 'Product Name'

-Product Name 'Customised Plan' shows most preferred.

-boxplot showing sales split against product name. Also, customer opting for Silver and Gold plan claim is high as compared to others.

Destination

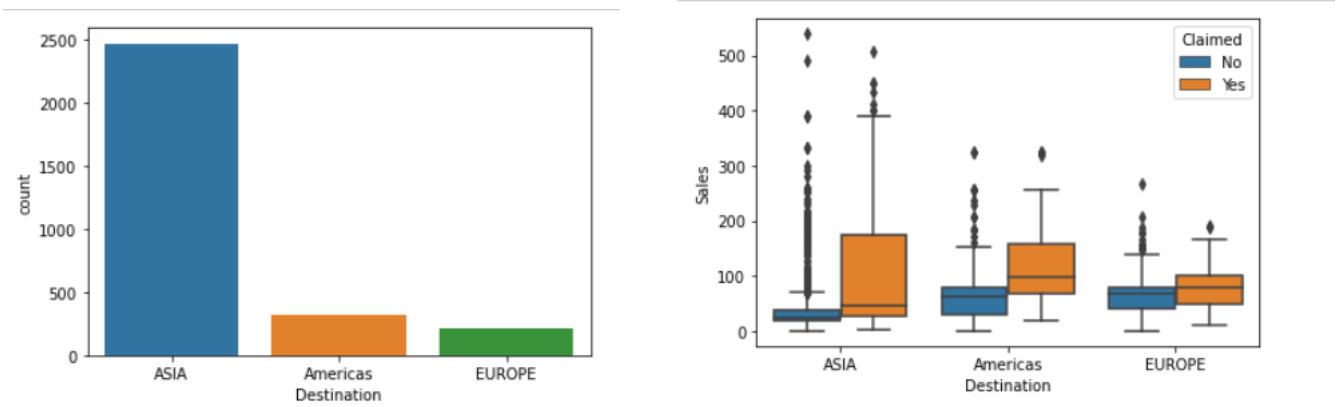


Figure 22: Variable 'Destination'

-'ASIA' is most visited destination opted by customer as compared to Americas and Europe destination.

-Boxplot shows sales split against the destinations. Also, customer opting Asia as destination has high claim as compared to others.

Checking pairwise distribution of the continuous variables

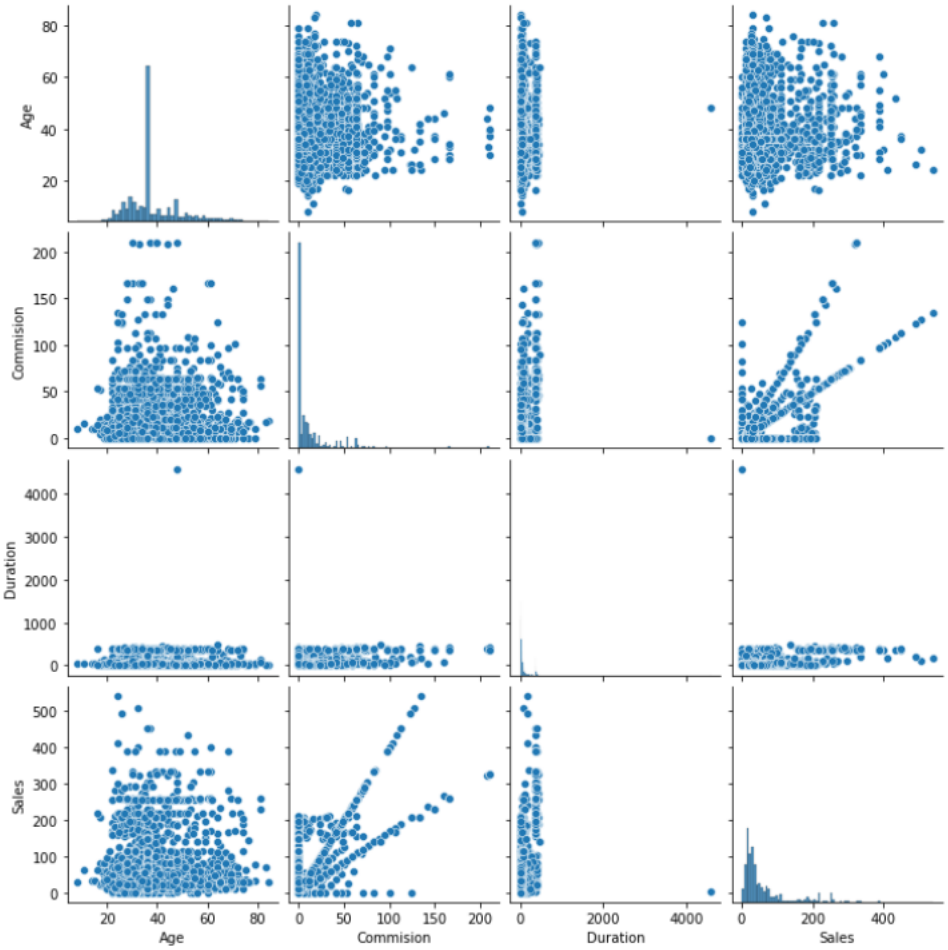


Figure 23: Pairplot of Numeric Variables

Checking for Correlation

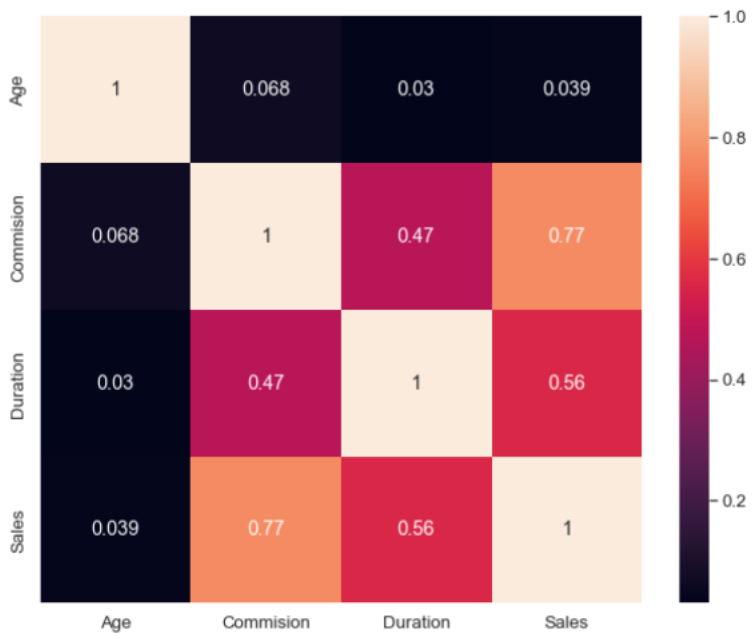


Figure 24: Correlation Matrix

Observation:

I don't observe a very strong correlation between the variables. Though to an extent commission and sales variable are correlated.

There is lack of multicollinearity observed.

Also, no negative correlation all positive correlation.

Converting all object datatype to categorical datatype

Why Categorical datatype? The category data type in pandas is **a hybrid data type**. It looks and behaves like a string in many instances but internally it is represented by an array of integers. Some of the python visualization libraries can interpret the categorical data type to apply appropriate statistical models or plot types. Hence here converting all object types to categorical datatypes. Here is a snapshot of the code used and the output of conversion as below:

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

Proportion of 0's and 1's in Target Column

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

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

CART, Classification and Regression Tree is a algorithm which is used to build a Decision Tree.

A Decision Tree is a Supervised Learning Technique that can be used for both Classification and Regression problems, but most preferred for solving Classification problems.

Decision Tree is a tree structured classifier, where internal nodes represent the feature of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision Tree there are 2 nodes, Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas leaf nodes are output of those decisions and do not contain any further branches.

In simple terms, It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

A decision tree simply asks a question and based on the answer (Yes/No).

The representation for the CART model is a binary tree.

With the binary tree representation of the CART model, making predictions in Decision Tree is relatively straightforward.

Let’s go ahead and build the Decision Tree by applying CART algorithm…..

Extracting the Target Column into separate vector for train and test set.

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

y = Ins_df.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

```
y.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: Claimed, dtype: int8
```

The train-test split is **a technique for evaluating the performance of a machine learning algorithm**. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets.

Why do we split the dataset into training and test data?

Separating data into training and testing sets is an **important part of evaluating data mining models**. … Because the data in the testing set already contains known values for the attribute that we want to predict, it is easy to determine whether the model's guesses are correct.
As the name also suggest the Training set is used for training the model and testing set is used for testing the accuracy of the model.

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

Here we are using the split ratio of 70:30. *The 30% testing data set is represented by the 0.3 at the end.*
**random_state* variable is a pseudo-random number generator state used for random sampling.

To compare the shape of different testing and training sets, below is the following piece of code:

```
# Checking the dimensions of the training and test data
print("shape of original dataset :", Ins_df.shape)
print("shape of input - training set",X_train.shape)
print("shape of input - testing set",X_test.shape)
print("shape of output - training set",train_labels.shape)
print("shape of output - testing set",test_labels.shape)
```

```
shape of original dataset : (3000, 9)
shape of input - training set (2100, 9)
shape of input - testing set (900, 9)
shape of output - training set (2100,)
shape of output - testing set (900,)
```

Method 1:

Following steps involve creating the Decision Tree Classifier and fitting the Decision Tree classifier into the Insurance dataset.

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

```
Ins_dt_model.fit(X_train, train_labels)
```

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

In order to check the goodness of splitting criterion, here Gini index criterion is used for Node splitting.

Gini says, if we select two items from a population at random then they must be of same class and probability for this is 1 if the population is pure.

- Gini usually works with categorical target variable “Success” and “Failure”.
- Gini performs Binary splits.
- Higher the value of Gini higher the homogeneity.
- CART used Gini method to create binary splits.

Steps to Calculate Gini for a split

1. Calculate Gini for sub-nodes, using formula sum of square of probability for success and failure (p^2+q^2).
2. Calculate Gini for split using weighted Gini score of each node of that split

Now that we have a fitted decision tree model and we can proceed to visualize the tree. We start with the easiest approach — using inbuilt library from sklearn import tree function from scikit-learn.

```
from sklearn import tree

train_char_label = ['No', 'Yes']
Ins_Tree_File = open('d:\Ins_tree.dot','w')
dot_data = tree.export_graphviz(Ins_dt_model, out_file=Ins_Tree_File, feature_names = list(X_train),
                                class_names = list(train_char_label))

Ins_Tree_File.close()
```

In above code, we have created a ‘dot’ file which will contains all the instructions on how to build this graphical visualization of the Classification Tree that we had built.

We first open the dot file, pass all the instructions and write in the 'dot' file, then we go ahead and close the file.

We can go ahead and open a webgraph editor in a browser and paste the contents of the 'dot' file that has been created to visualize the Classification Tree.

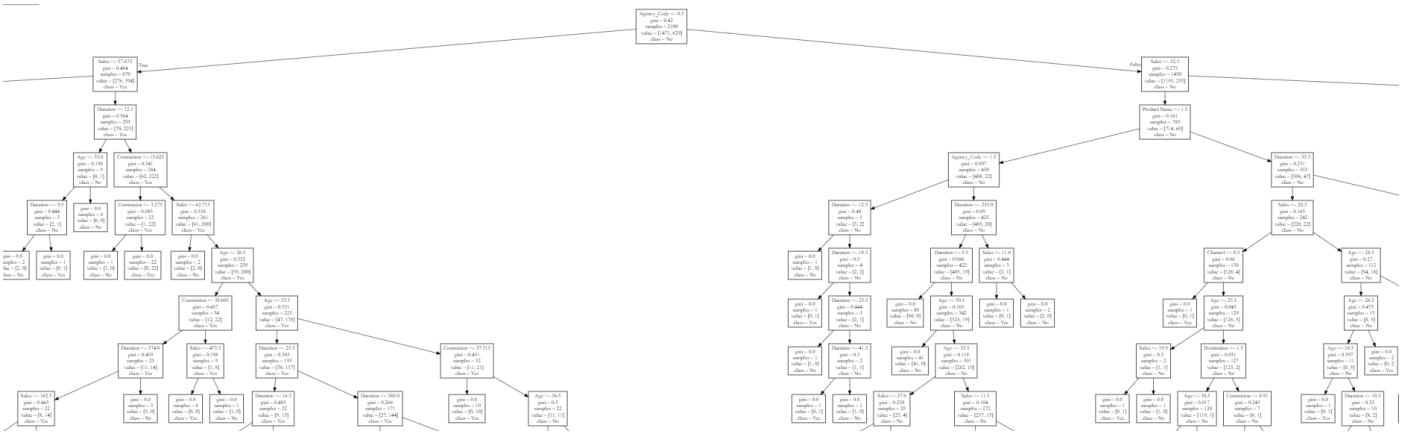


Figure 25: Decision Tree before Pruning

It is very evident from above Figure of Decision Tree that my tree is overgrown.

Lets check the important variables in the dataset.

Feature importance provides a score that indicates how useful or valuable each feature was in the construction of the boosted decision trees within the model. The more an attribute is used to make key decisions with decision trees, the higher its relative importance. Here we see that our importance feature of the model is Duration, Sales, Agency_code, and Age.

```
print (pd.DataFrame(Ins_dt_model.feature_importances_, columns = ["Imp"], index = X_train.columns))
```

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

Graphical representation of feature importance.

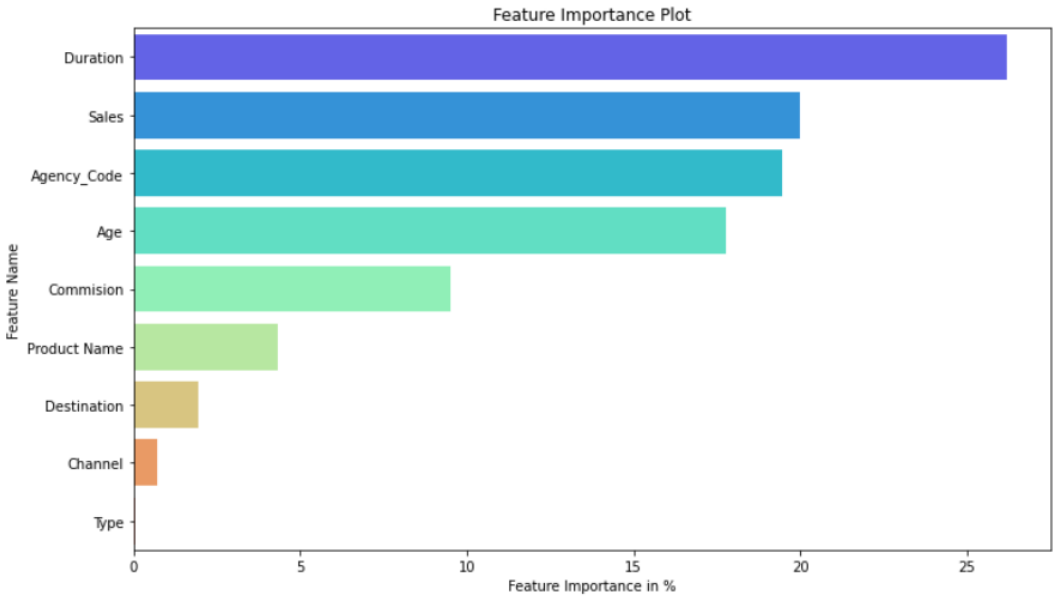


Figure 26: Feature Importance before Pruning

Lets go ahead and prune the Decision Tree. This process of trimming trees is called “Pruning”.

Note: Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting. By default, the Decision Tree function doesn’t perform any pruning and allows the tree to grow as much as it can.

We get an accuracy score of **0.99** and **0.71** on the train and test part respectively as shown below. We can say that our model is Overfitting i.e. memorizing the train part but is not able to perform equally well on the test part.

```
Ins_dt_model.score(X_train,train_labels)
```

```
0.9947619047619047
```

```
Ins_dt_model.score(X_test,test_labels)
```

```
0.7177777777777777
```

Overfitting: Over-fitting is the phenomenon in which the learning system tightly fits the given training data so much that it would be inaccurate in predicting the outcomes of the untrained data.

In decision trees, over-fitting occurs when the tree is designed so as to perfectly fit all samples in the training data set.

Overfitting is easy to diagnose with the accuracy visualizations is available. If "Accuracy" (measured against the training set) is very good.

We can combat overfitting by *reducing the complexity* of our model (i.e. reducing the number of trainable parameters).

Method2: We can also build DT Classifier using **GridSearchCV** to find best hyperparameters.

GridSearchCV is **a library function** that is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters

GridSearchCv which is the main component which will help us find the best hyperparameters. We simply create a tuple (kind of non edit list) of hyperparameters we want the machine to test with as save them as params.

We then give the classifier and list of params as parameters to GridSearchCV as below:

```
from sklearn.model_selection import GridSearchCV
```

```
params = {'criterion':['gini'],
          'max_depth':[4,5,7,9,10,11,13,15],
           'min_samples_split':[150,300,450],
          "min_samples_leaf":[50,100,150]}
```

Max_depth: This **determines the maximum depth of the tree**. The deeper the tree, the more splits it has and it captures more information about the data.

Min_samples_leaf: specifies **the minimum number of samples required to be at a leaf node**.

Min_samples_split: specifies **the minimum number of samples required to split an internal node**

The next step is to fit the grid search cv model with training data, the output will look something like this-

```
grid_Ins_dt_model = GridSearchCV(DecisionTreeClassifier(),param_grid=params,refit=True,verbose=1)
grid_Ins_dt_model.fit(X_train, train_labels)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits

GridSearchCV(estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini'],
                         'max_depth': [4, 5, 7, 9, 10, 11, 13, 15],
                         'min_samples_leaf': [50, 100, 150],
                         'min_samples_split': [150, 300, 450]},
             verbose=1)
```

Now these below parameters are the best hyperparameter for this algorithm.

```
grid_Ins_dt_model.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 5,
 'min_samples_leaf': 50,
 'min_samples_split': 150}
```

Next step is to **Pruning/Regularizing the Decision Tree** with appropriate parameters.

i.e., regularizing the Decision Tree with best parameters and fitting the model train and test data with the dataset.

```
reg_Ins_dt_model = DecisionTreeClassifier(criterion='gini',max_depth= 5,min_samples_leaf=50,min_samples_split=150)
reg_Ins_dt_model.fit(X_train, train_labels)
```

```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=50, min_samples_split=150)
```

Once we have fitted the data we can again visualize our DT using online webvizgraph link.

```
train_char_label = ['No', 'Yes']
Ins_Tree_regularized = open('d:\Ins_tree_regularised.dot','w')
dot_data = tree.export_graphviz(reg_Ins_dt_model, out_file=Ins_Tree_regularized, feature_names = list(X_train),
                                class_names = list(train_char_label))
Ins_Tree_regularized.close()
```

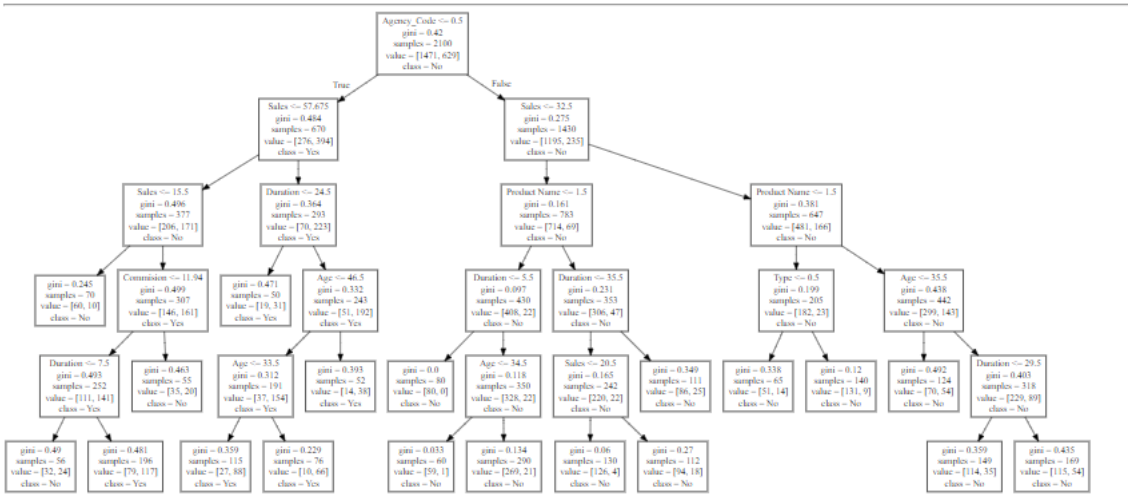


Figure 27: Pruned/Regularised DT

The above image shows a improved DT. Also, the branches are not coming out in haphazard manner.

we also see that the max_depth level has decreased, and a good amount of pruning has happened.

We will go ahead and follow the below steps now.

Let's the check the feature importance, Feature importance refers to a class of techniques for assigning scores to input features to a predictive model that indicates the relative importance of each feature when making a prediction.

Inspecting the importance score provides insight into that specific model and which features are the most important and least important to the model when making a prediction.

```
print (pd.DataFrame(reg_Ins_dt_model.feature_importances_, columns = ["Imp"], index = X_train.columns))
```

```
                 Imp
Age         0.022509
Agency_Code 0.607479
Type        0.007516
Commision    0.012847
Channel     0.000000
Duration    0.034352
Sales       0.259249
Product Name 0.056048
Destination  0.000000
```

Note: After pruning we see that, now our very important variable used in the model are Agency_code, sales with high importance ratio as compared to other variable with less importance ratio.

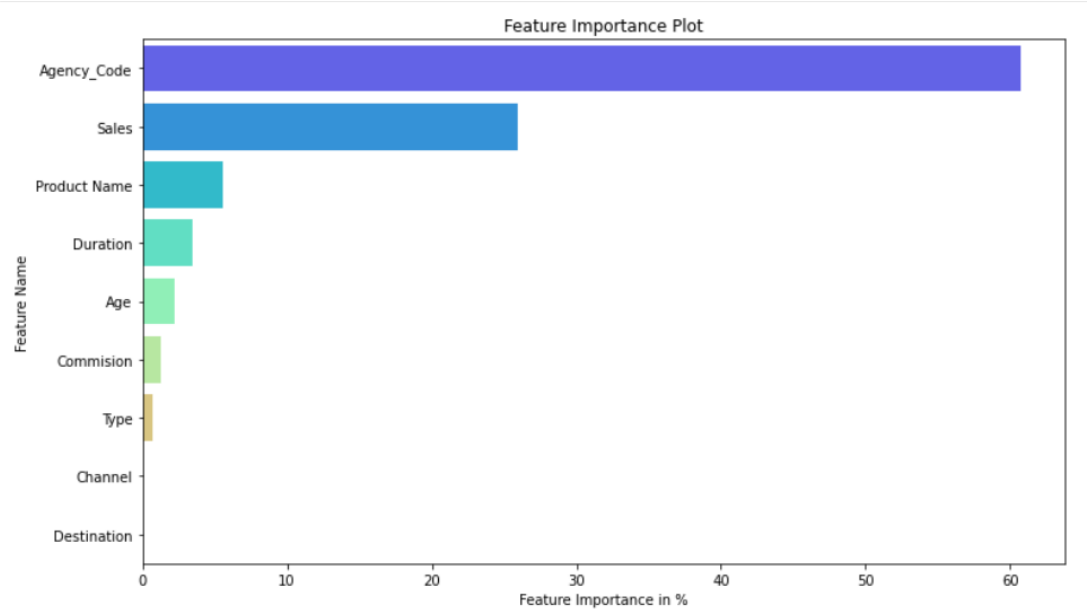


Figure 28: Feature Importance plot DT

The variable with '0'(zero) that independent variable was never used in splitting the nodes within the DT. They are not used in Training and will never be used in testing. Based on this we can keep and delete the variables.

As, we have trained the model, checked the feature importance lets go ahead and predict on the test data.

Next step is "Prediction".

```
ytrain_predict_cart1 = reg_Ins_dt_model.predict(X_train)
ytest_predict_cart1 = reg_Ins_dt_model.predict(X_test)
```

```
ytrain_predict_cart1
```

```
array([0, 0, 1, ..., 0, 0, 1], dtype=int8)
```

*Output is in array. This gives us the prediction of Train and test data.

We can also, check the probability of the predict train and test data as below:

```
ytrain_predict_cart2 = reg_Ins_dt_model.predict_proba(X_train)
ytest_predict_cart2 = reg_Ins_dt_model.predict_proba(X_test)
```

```
ytrain_predict_cart2
```

```
array([[0.96923077, 0.03076923],
       [0.68047337, 0.31952663],
       [0.23478261, 0.76521739],
       ...,
       [0.68047337, 0.31952663],
       [0.92758621, 0.07241379],
       [0.40306122, 0.59693878]])
```

Further we will do Evaluation of Train and Test model by calculating the Accuracy, classification report and confusion matrix. (Refer to 2.3)

MODEL 2: Random Forest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning**, which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

Advantage of Random Forest

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

Implementing RF for our Insurance dataset, the first step is to do data pre-processing as we have done in problem no. 2.1.

Next step is splitting the dataset into Training and Testing set.

```
## Splitting the dataset into training and test set.
X_train, X_test, train_labels, test_labels = train_test_split(X, y, test_size=.30, random_state=1)
```

Now we will fit the Random Forest algorithm to the training set. To fit it, we will import the **RandomForestClassifier** class from the **sklearn.ensemble** library. The code is given below:

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

```
rfcl = RandomForestClassifier(n_estimators = 501,criterion='gini',
                              random_state=1,
                              oob_score=True,
                              max_features=4)
```

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

```
RandomForestClassifier(max_features=4, n_estimators=501, oob_score=True,
                       random_state=1)
```

In the above code, the classifier objects take below parameters:

- **n_estimators=** The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.
- **criterion=** It is a function to analyze the accuracy of the split. Here we have taken "gini" for the gain index/gini impurity.
- **Oob_score=**Whether to use out-of-bag samples to estimate the generalization score. Only available if bootstrap=True. In this one-third of the sample is not used to train the data instead used to evaluate its performance. These samples are called out of bag samples.
- **max_features=** The number of features to consider when looking for the best split.

Since our model is fitted to the training set, lets go ahead and check the oob_score.

```
rfcl.oob_score_
```

```
: 0.7542857142857143
```

Now, we check the feature importance of the dataset.

```
pd.Series(rfcl.feature_importances_,index=X_train.columns).sort_values(ascending=False)
```

```
Duration        0.250725
Sales           0.202638
Age             0.175082
Commision       0.122420
Agency_Code     0.112718
Product Name    0.088615
Type            0.020950
Destination     0.020598
Channel         0.006254
dtype: float64
```

The above result shows us the variable which are important and is arranged in descending order. Duration, sales, Age, Commision, Agency_code and so on.

Next step is to check the Accuracy of the Training and Test data.

```
rfcl.score(X_train,train_labels)
```

```
0.9947619047619047
```

```
rfcl.score(X_test,test_labels)
```

```
0.7555555555555555
```

We see that the accuracy on the Training data is 0.99 approximately 1 and the accuracy on the test data is 0.76. This shows a case of slight overfitting.

But in Random Forests training uses bootstrap aggregation (or sampling with replacement) along with random selection of features for a split, the correlation between the trees would be low. That means although individual trees have high variance, but the ensemble output will be appropriate because the tree are not correlated.
So, unlike the tree, no pruning takes place in Random Forest.

Here, in the case of Random Forest, Let's do hyperparameter tuning for Random Forest using **GridSearchCV** and fit the data.

What the command **GridSearchCV** does is build multiple model based on a range of parameters specified by us. Then it ultimately returns best parameters on the basis of which we can go ahead and build our model.

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [5,10],
    'max_features': [4,6],
    'min_samples_leaf': [50,150],
    'min_samples_split': [150,450],
    'n_estimators': [301,401]
}

rfcl = RandomForestClassifier(random_state=1,oob_score=True)

grid_search = GridSearchCV(estimator = rfcl, param_grid = param_grid, cv = 3, n_jobs=-1)
```

Important Hyperparameters

Hyperparameters are used in random forests to either enhance the performance and predictive power of models or to make the model faster.

Following hyperparameters increases the predictive power:

1. **n_estimators**– number of trees the algorithm builds before averaging the predictions.

2. ***max_features***– maximum number of features random forest considers splitting a node.

3. ***mini_sample_leaf***– determines the minimum number of leaves required to split an internal node.

Following hyperparameters increases the speed:

1. ***n_jobs***– it tells the engine how many processors it is allowed to use. If the value is 1, it can use only one processor but if the value is -1 there is no limit.

2. ***random_state***– controls randomness of the sample. The model will always produce the same results if it has a definite value of random state and if it has been given the same hyperparameters and the same training data.

3. ***oob_score*** *– OOB* means out of the bag. It is a random forest cross-validation method. In this one-third of the sample is not used to train the data instead used to evaluate its performance. These samples are called out of bag samples.

Next step is to fit the model as below:

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

```
GridSearchCV(cv=10,
             estimator=RandomForestClassifier(oob_score=True, random_state=1),
             n_jobs=-1,
             param_grid={'max_depth': [3, 6, 10], 'max_features': [2, 4, 6],
                         'min_samples_leaf': [10, 50, 100],
                         'min_samples_split': [50, 300],
                         'n_estimators': [101, 501, 701]})
```

From hyperparameter tuning, we can fetch the best estimator as shown.

```
grid_search.best_params_
```

```
{'max_depth': 6,
 'max_features': 2,
 'min_samples_leaf': 10,
 'min_samples_split': 50,
 'n_estimators': 101}
```

```
best_grid=grid_search.best_estimator_
best_grid
```

```
RandomForestClassifier(max_depth=6, max_features=2, min_samples_leaf=10,
                       min_samples_split=50, n_estimators=101, oob_score=True,
                       random_state=1)
```

Next step is to check the feature importance as below:

```
pd.Series(best_grid.feature_importances_,index=X_train.columns).sort_values(ascending=False)
```

```
Agency_Code     0.258772
Commision       0.176513
Sales           0.159931
Product Name    0.155005
Type            0.094890
Duration        0.090983
Age             0.051663
Destination     0.011707
Channel         0.000536
dtype: float64
```

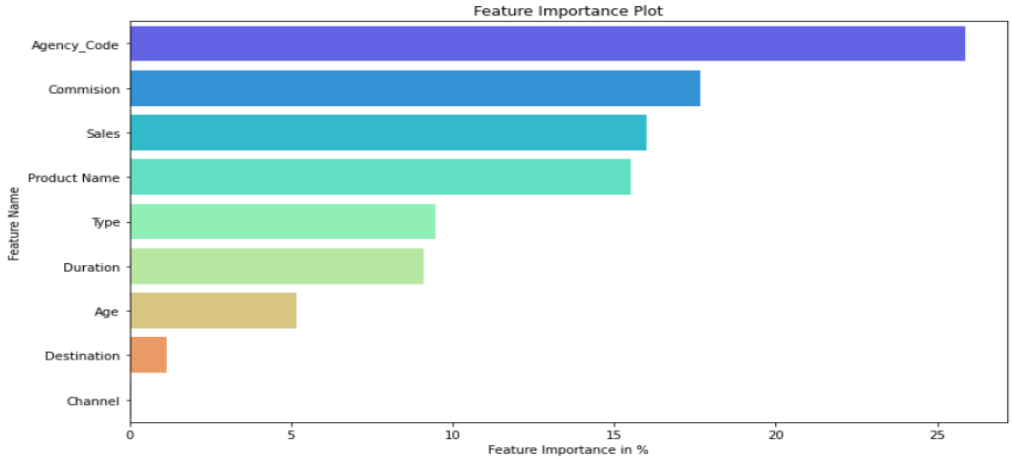


Figure 29: Feature Importance plot RF

We will go ahead with our model prediction now

```
ytrain_predict_rf1 = best_grid.predict(X_train)
ytest_predict_rf1 = best_grid.predict(X_test)
```

Also, to put our prediction in perspective, we will evaluate the accuracy, classification and confusion matrix of the Training and Test model. (Refer to problem no. 2.3)

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

Classification is a Supervised leaning approach in which a target variable is discrete (or categorical). Evaluating a machine learning model is as important as building it. There are many ways for evaluating the performance of the Model. Here, we are going to see 3 Performance metrics i.e. Accuracy, Confusion matrix and Classification report. Let's go through it one by one.

Accuracy (overall, how often is the classifier is correct)

Accuracy simple measures how often the classifier correctly predicts. We can define accuracy as the ratio of the number of correct predictions and the total number of predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Below is the Comparison of metric Accuracy of Train and Test Model of Decision Tree and Random Forest Classifiers.

```
Accuracy Score for Train set for DecisionTreeClassifier is 0.7914285714285715
Accuracy Score for Test set for DecisionTreeClassifier is 0.7588888888888888
Accuracy Score for Train set for RandomForestClassifier is 0.8080952380952381
Accuracy Score for Test set for RandomForestClassifier is 0.7677777777777778
```

The Accuracy of the Training data of Decision Tree model is 0.79%.
The Accuracy of the Test data of Decision Tree model is 0.76%.

The Accuracy of the Training data of Random Forest model is 0.81%.
The Accuracy of the Test data of Random Forest model is 0.77%.

Training and Test data, after the data has been pre-processed and is ready for training, the data is split into Training Data, and Testing Data in the ratio of 70:30 (in this problem). This ratio varies depending on the availability of data and on the application. This is done to ensure that the model does not unnecessarily "Overfit" or "Underfit" and performs equally well when deployed in the real world.

Finally, as the last step, the Training Data is fed into the model to train upon. the training of data is the most crucial step of any Machine Learning Algorithm. Improper training can lead to drastic performance degradation of the model on deployment. On a high level, there are two main types of outcomes of Improper Training: Underfitting and Overfitting.

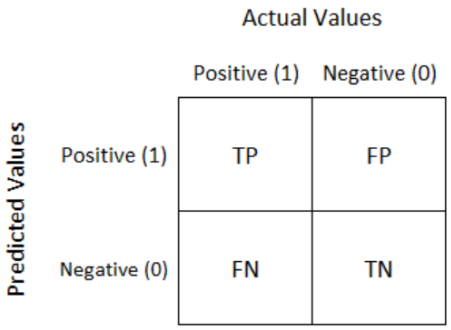
How do we identify overfitting and Underfitting?

If both the training accuracy and test accuracy are close, then the model has **not overfit**. If the training result is very good and the test result is poor, then the model has **overfitted**. If the training accuracy and test accuracy is low, then the model has **underfit.**

Confusion Matrix

This is a performance measurement for the machine learning classification problems where the output can be two or more classes. It is a table with combinations of predicted and actual value.

A confusion matrix is a summary of prediction results on a classification problem.
The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.



TP (True Positive): We predicted positive (1) and its actual value is also Positive (1).

TN (True Negative): We predicted Negative (0) and its true (1).

False Positive (type1 error) (FP): We predicted positive (1) and its false (0).

False Negative (type2 error) (FN): We predicted Negative (0) and its false (0).

Precision:

Precision explains how many of the correctly predicted cases turned out to be positive. Precision is useful in the cases where False Negative is a higher concern than False Negatives.

$$Precision = \frac{TP}{TP + FP}$$

Recall (Sensitivity):

Recall explains how many of the actual positive cases we were able to predict correctly with our model. It is a useful metric in case where False Negative is of higher concern than False Positive.

$$Recall = \frac{TP}{TP + FN}$$

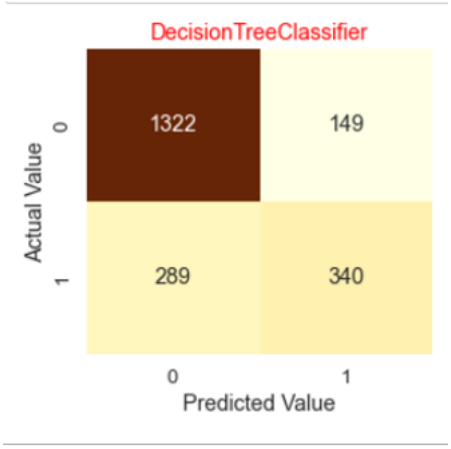
F1 Score (Harmonic Mean):

It gives a combined idea about Precision and Recall metrics. It is maximum when Precision is equal to Recall.

$$F\text{-}measure = \frac{2*Recall*Precision}{Recall + Precision}$$

F1 Score is effective when False Positive and False Negative are equally costly. Adding more data doesn't effectively change the outcome. True Negative is high.

Confusion Matrix of Train and Test data of DT model.



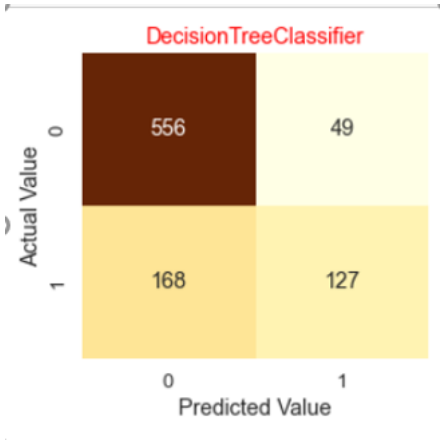


Figure 30.1: Confusion Matrix of Train data of DT *Figure 30.2*: Confusion Matrix of Test data of DT

What does the above matrix tell us:

1. **Decision Tree Classifier**

- The **rows** represent the **actual values** of the target variable
- The **column** represent the **predicted values** of the target variable
- There are 2 possible prediction classes "0" and "1".
- When we are predicting status of Policy Claimed by the customer, for example, "0" would mean customer has "Not Claimed" and "1" would mean customer has "Claimed".
- The classifier has made total 2100 cases in Train data and 900 cases in Test data .
- In Train data out of these 2100 cases the classifier predicted "No(Not Claimed)" 1611 times and "Yes(Claimed)" 489 times. Likewise, in test data where we have 900 cases the classifier predicted "No" 724 times and "Yes" 176 times.
- In reality, in the Train data "Not Claimed" customer were 1471 and "Claimed" customers were 629.
- Also, in Test data "Not Claimed" customer were 605 and "Claimed" customer were 295.

- For Train data: TN=1322, FP=149, FN=289 and TP=340.
- For Test data: TN=556, FP=49, FN=168 and TP=127.

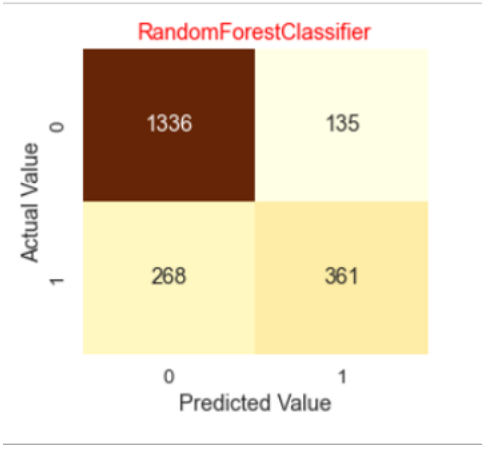
For Train data

- True Positive (TP) = 340; meaning 340 positive class data points were correctly classified by the model
- True Negative (TN) = 1322; meaning 1322 negative class data points were correctly classified by the model
- False Positive (FP) = 149; meaning 149 negative class data points were incorrectly classified as belonging to the positive class by the model
- False Negative (FN) = 289; meaning 289 positive class data points were incorrectly classified as belonging to the negative class by the model

For Test data

- True Positive (TP) = 127; meaning 127 positive class data points were correctly classified by the model
- True Negative (TN) = 556; meaning 556 negative class data points were correctly classified by the model
- False Positive (FP) = 49; meaning 49 negative class data points were incorrectly classified as belonging to the positive class by the model
- False Negative (FN) = 168; meaning 168 positive class data points were incorrectly classified as belonging to the negative class by the model

Confusion Matrix of Train and Test data of Random Forest.



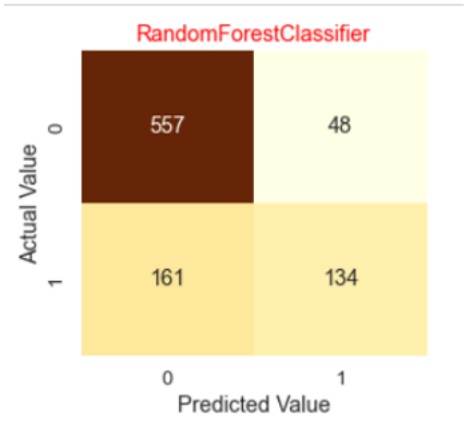


Figure 31.1: Confusion Matrix of Train data of RF *Figure 31.2*: Confusion Matrix of Test data of RF

Random Forest Classifier

- The **rows** represent the **actual values** of the target variable.
- The **column** represents the **predicted values** of the target variable.
- There are 2 possible prediction classes "0" and "1".
- When we are predicting status of Policy Claimed by the customer, for example, "0" would mean customer has "No(Not Claimed)" and "1" would mean customer has "Yes(Claimed)".
- The classifier has made total 2100 cases in Train data and 900 cases in Test data.
- In Train data out of these 2100 cases the classifier predicted "Not Claimed" 1604 times and "Claimed" 496 times. Likewise, in test data where we have 900 cases the classifier predicted "No" 718 times and "Yes" 182 times.
- In reality, "Not Claimed" customer were 1471 and "Claimed" customers were 629.
- Also, in Test data "Not Claimed" customer were 605 and "Claimed" customer were 295.
- For Train data: TN=1336, FP=135, FN=268 and TP=361.

- For Test data: TN=557, FP=48, FN=161 and TP=134.

For Train data

- True Positive (TP) = 361; meaning 361 positive class data points were correctly classified by the model
- True Negative (TN) = 1336; meaning 1336 negative class data points were correctly classified by the model
- False Positive (FP) = 135; meaning 135 negative class data points were incorrectly classified as belonging to the positive class by the model
- False Negative (FN) = 268; meaning 268 positive class data points were incorrectly classified as belonging to the negative class by the model

For Test data

- True Positive (TP) = 134; meaning 134 positive class data points were correctly classified by the model
- True Negative (TN) = 557; meaning 557 negative class data points were correctly classified by the model
- False Positive (FP) = 48; meaning 48 negative class data points were incorrectly classified as belonging to the positive class by the model
- False Negative (FN) = 161; meaning 161 positive class data points were incorrectly classified as belonging to the negative class by the model

Some Important values of the Confusion Matrix would be as follows:

For instance, let's take the Train data of DT model and calculate them manually.

True Positive rate: When its actually "Yes", how often does it predict "Yes"? is also known as "**Sensitivity/Recall**".
In Train Model, the total actual "Yes" cases are 629 and total predicted "Yes" cases are 340.
i.e., the True Positive rate will be =TP/actual= 340/629= 0.5405405405405405 which is our Recall value of "Yes" in the model.

False Positive rate: When its's actual showing "No" and it predicts "Yes".
i.e. among 1417 "No" cases how often the model predict "Yes" i.e. 149
FP/actual=149/1417=0.101291638341

True Negative Rate: When its actually "No" how often the model predicts "No".
True Negative rate is actually 1-False Positive rate.
Also known as "**Specificity**"
TN/actual=1322/1417=0.898708361658

Precision: When it predicts "Yes" how often it is correct?
i.e. from total 489 "Yes" predicted cases there is 340 actual "Yes" cases.
TP/actual=340/489=0.695296523517

F1score: How good and complete are the prediction.
=2*((0.695296523517*0.54054054054054)/(0.54054054054054+0.695296523517))=0.608229

Likewise, we can calculate the precision and recall for the other class.

Precision of "No" Class : TN/actual= 1322/1611= 0.8206083178150217
Recall for the "No" class: 1322/1471= 0.8987083616587356

*If we want our model to be smart, then our model has to predict correctly.

This means our **True Positives** and **True Negatives** should be as high as possible, and at the same time, we need to minimize our mistakes for which our **False Positives** and **False Negatives** should be low as possible.
Also, in terms of ratios, our TPR & TNR should be very high whereas FPR & FNR should be very low,
i.e. A Smart Model: TPR ↑ , TNR ↑, FPR ↓, FNR ↓

Classification Report

An Sklearn library which is important to execute these reports is

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

DT-CART Classification Report-Train Data				
	precision	recall	f1-score	support
0	0.82	0.90	0.86	1471.00
1	0.70	0.54	0.61	629.00
accuracy			0.79	2100
macro avg	0.76	0.72	0.73	2100
weighted avg	0.78	0.79	0.78	2100

Table 1: Classification Report of Train Data of DT Model

Here, Precision=0.70 and Recall=0.54
70% percent of the correctly predicted cases turned out to be positive cases. Whereas 54% of the positives were successfully predicted by our model.

DT-CART Classification Report-Test Data				
	precision	recall	f1-score	support
0	0.77	0.92	0.84	605
1	0.72	0.43	0.54	295
accuracy			0.76	900
macro avg	0.74	0.67	0.69	900
weighted avg	0.75	0.76	0.74	900

Table 2: Classification Report of Test Data of DT Model

Here, Precision=0.72 and Recall=0.43
72% percent of the correctly predicted cases turned out to be positive cases. Whereas 43% of the positives were successfully predicted by our model.

RF Classification Report-Train Data				
	precision	recall	f1-score	support
0	0.83	0.91	0.87	1471
1	0.73	0.57	0.64	629
accuracy			0.81	2100
macro avg	0.78	0.74	0.76	2100
weighted avg	0.8	0.81	0.8	2100

Table 3: Classification Report of Train Data of RF Model

Here, Precision=0.73 and Recall=0.57
73% percent of the correctly predicted cases turned out to be positive cases. Whereas 57% of the positives were successfully predicted by our model.

RF Classification Report-Test Data				
	precision	recall	f1-score	support
0	0.78	0.92	0.84	605
1	0.74	0.45	0.56	295
accuracy			0.77	900
macro avg	0.76	0.69	0.7	900
weighted avg	0.76	0.77	0.75	900

Table 4: Classification Report of Test Data of RF Model

Here, Precision=0.74 and Recall=0.45
74% percent of the correctly predicted cases turned out to be positive cases. Whereas 45% of the positives were successfully predicted by our model.

Above, we saw the precision and recall values of both Training and Test data of both Decision Tree and Random Forest. It is still difficult to say which model is the winner on these two metrics. Hence lets combine them, I mean lets see our f1 scores.

[Precision is a useful metric in cases where False Positive is a higher concern than False Negatives.]
[Recall is a useful metric in cases where False Negative trumps False Positive.]

In practice, when we try to increase the precision of our model, the recall goes down, and vice-versa. The F1-score captures both the trends in a single value:

2*((precision*recall)/(precision+recall))

F1-score is a harmonic mean of Precision and Recall, and so it gives a combined idea about these two metrics. It is maximum when Precision is equal to Recall.

Let's go ahead and compare all these important values/metrics in problem no. 2.4.

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

Let us see some Pros and Cons of DT and RF before we compare the model values we have achieved while coding in python.

Pros & Cons of Decision Trees

Pros

- Easy to interpret
- Handles both categorical and continuous data well.
- Works well on a large dataset.
- Not sensitive to outliers.
- Non-parametric in nature.

Cons

- These are prone to overfitting.
- It can be quite large, thus making pruning necessary.
- It can't guarantee optimal trees.
- It gives low prediction accuracy for a dataset as compared to other machine learning algorithms.
- Calculations can become complex when there are many class variables.
- High Variance(Model is going to change quickly with a change in training data)

Pros & Cons of Random Forest

Pros:

- Robust to outliers.
- Works well with non-linear data.
- Lower risk of overfitting.
- Runs efficiently on a large dataset.
- Better accuracy than other classification algorithms.

Cons:

- Random forests are found to be biased while dealing with categorical variables.
- Slow Training.
- Not suitable for linear methods with a lot of sparse features

Comparison:

	DT CART Train	DT CART Test	Random Forest Train	Random Forest Test
Accuracy	0.791429	0.758889	0.808095	0.767778
Recall	0.540541	0.430508	0.573927	0.454237
Precision	0.695297	0.721591	0.727823	0.736264
F1 Score	0.608229	0.539278	0.641778	0.561845

Table 5: Model Comparison DT/RF

To compare the models i.e Decision Tree and Random Forest I have prepared a tabular comparison.

The table shows the value of Accuracy, Recall, Precision and F1score of both the DT and RF model.

To see whether a model is working properly and going to give us efficient result it is important to consider these factors.

As we see above and compare (also refer to 2.3 classification report), we see that,

DT has 76% of accuracy against 77% against RF. However, that information alone is not enough to determine which model is best.

Let's, Look at the f1score column. RF has f1score of 56% and DT has f1score of 54%. Here we can see that RF outperformed DT. That means that RF can identify more accurately the True Positives among these classes.

Now, looking at Precision and Recall values of DT i.e., 43% and 72%, our RF model shows 74% and 45% against the same. Here we see again that RF is outperforming DT.

So, the quality, condition, or the fact of being exact and accurate our RF model is outperforming than DT.

Hence, in this problem, I will be selecting **Random Forest** as my winning Model.

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

Insurance is an information-based industry where data is the key ingredient. Looking at our data I feel we need more data variables to understand where the organization is trying to create an opportunity based on our insight. Our model in turn will help the organization to create the right product with optimum risk.

With the current dataset, below are my insights:

The distribution channel of travel insurance is more online (more than 90%) than offline. Which is leading to good number of sales per customer, procured by the tour insurance policy company.

Type 'Airline' of tour insurance firm seems doing pretty good policy sale as compared to 'Travel agency' firm.

And claim percentage is good via type 'Airline' than 'Travel Agency'.

Type 'Travel Agency' tour insurance firm 'No claim' towards insurance policy is quite hight. Need to dig deep and understand why this tour insurance firm is lagging towards policy claim.

JZI agency firm need to have resources to pick up sales as they are underperforming compared to other firms. They need to strengthen their marketing campaign and customer experience for cross selling, reference selling, add more customer to their data.

EPX/Travel agent selling insurance policy count is good but need to understand why number of customer not claiming for policy is high. Need more customer interaction for pushing customers to claim more.

With reference to our winning Model performance, I can see that "Agency_code" is playing a major role and has a great role in the model for predicting to determine the claims made by the customer.

Our RF model shows accuracy of 81%, shows the organization that customer needs to book airline tickets more and plan trips more. This can be used wisely based on claimed data patterns.

Recommendation for insurance organization for policy claims are:

Insurance companies and agencies must track cost, time, and productivity-related metrics to ensure their business growth. Some of the important ones are expense ratio, renewal or retention rate, quota vs. production, and average policy size.

Sales are the backbone of the insurance industry. We can have all the products we want, but without someone selling them, we can't make a profit.

Insurers should focus on integrating their systems and platforms to create the ability for agents to have seamless conversations across multiple channels with customers.

Insurer also change and tie up with other alternative agencies/channels.

Adding new channels to communicate with customer for policy questions and claims.

Channing language in documents and communications to use less insurance jargon.

Offering hybrid experiences (human and artificial intelligence, physical and virtual, direct and agent-based).

Engaging with customers daily or throughout the year instead of only at renewal time.

Measuring how may new clients were referred by existing clients.

The policy sales growth can be defined based on the number of new clients, a measure of number of new policies sold, or a combination of the two. So, its important to increase new customers.

By incorporating process automation, insurer can optimise insurance claim processes hence minimizing error on claims.

Investigating Insurance claims to avoid fraud. Follow up with the account/insurance firm for improperly processed claims.

Also, organization should look into the area of reducing the cost of claim handling.

THANK YOU