PGP DSBA Jan_A 22 Batch Online

Project - Data Mining

Boby Sinha

Table of Contents

Contents

Dataset for Problem 1: https://drive.google.com/file/d/1SRWZlR8Wbxt-a3uXdf5l5nTXrkodJoO4/view?usp=sharing

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)

2. Insurance Analysis

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.

Dataset for Problem 2:
https://drive.google.com/file/d/1skj1zx67g6hHi0srYkxKWiBthluHrjuK/view?usp=sharing

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 percentage of sales)
   10.Age of insured (Age)

List of Figures

List of Tables

1. Cluster Analysis

Executive Summary:

A leading bank deals with different types of buying/spending behaviour of the customers. The dataset consists of various characteristics of the customers based on the models available in the bank. Based on the different attributes/characteristics of the buying/spending behaviour of the customers is defined. In this problem statement we will explore the different attributes of the buying/spending behaviour of the customers based on credit card usage and bank will develop a customer segmentation to give promotional offers to its customers.

Introduction:

The purpose of this whole exercise is to explore the dataset. Do the exploratory data analysis. Explore the dataset using clustering techniques such as hierarchical clustering and k- means clustering. The dataset consists of 210 rows and 7 columns, 210 customers with 7 different buying/spending behaviour of the customers. Analyse the different attributes of the bank marketing dataset which can help in segmenting the customers to give promotional offers. This assignment should help the student in exploring the summary statistics, information about the dataset, shape of the dataset, missing or duplicate values, understand the distribution of the dataset and to check any outliers are there in the dataset.

Data Description:

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)

Problem 1: Clustering

1.1 Read the data and do exploratory data analysis (3 pts). Describe the data briefly. Interpret the inferences for each (3 pts). Initial steps like head() .info(), Data Types, etc . Null value check. Distribution plots(histogram) or similar plots for the continuous columns. Box plots, Correlation plots. Appropriate plots for categorical variables. Inferences on each plot. Summary stats, Skewness, Outliers proportion should be discussed, and inferences from above used plots should be there. There is no restriction on how the learner wishes to implement this but the code should be able to represent the correct output and inferences should be logical and correct.

Sol:

Read the data

```
: bank_marketing_df = pd.read_csv('bank_marketing_part1_Data.csv')
```

Pd.read_csv function is used to read the file in jupyter notebook environment.

Importing the packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set(color_codes=True)
import warnings
warnings.filterwarnings("ignore")
from sklearn.cluster import KMeans
```

To read the dataset and do the analysis and exploratory data analysis we need to import packages in python jupyter notebook so that we can read the data and perform EDA.

Exploratory Data Analysis:

Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods. It helps determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions. EDA is primarily used to see what data

can reveal beyond the formal modelling or hypothesis testing task and provides a provides a better understanding of data set variables and the relationships between them.

Sample of the dataset:

	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
5	12.70	13.41	0.8874	5.183	3.091	8.456	5.000
6	12.02	13.33	0.8503	5.350	2.810	4.271	5.308
7	13.74	14.05	0.8744	5.482	3.114	2.932	4.825
8	18.17	16.26	0.8637	6.271	3.512	2.853	6.273
9	11.23	12.88	0.8511	5.140	2.795	4.325	5.003

Table 1. Dataset Sample

Dataset consists of 210 rows and 7 unique columns with 7 different buying/spending behaviour of the customers.

Top 10 entries or head of the dataset:

```
bank_marketing_df.head(10)   #Top 10 entries
```

	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
5	12.70	13.41	0.8874	5.183	3.091	8.456	5.000
6	12.02	13.33	0.8503	5.350	2.810	4.271	5.308
7	13.74	14.05	0.8744	5.482	3.114	2.932	4.825
8	18.17	16.26	0.8637	6.271	3.512	2.853	6.273
9	11.23	12.88	0.8511	5.140	2.795	4.325	5.003

From the above tables we can see the top 10 rows or entries of the dataset.

Bottom 10 entries or tail of the dataset:

```
bank_marketing_df.tail(10)     #Bottom 10 entries
```

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
200	14.88	14.57	0.8811	5.554	3.333	1.018	4.956
201	17.08	15.38	0.9079	5.832	3.683	2.956	5.484
202	14.80	14.52	0.8823	5.656	3.288	3.112	5.309
203	11.55	13.10	0.8455	5.167	2.845	6.715	4.956
204	16.41	15.25	0.8866	5.718	3.525	4.217	5.618
205	13.89	14.02	0.8880	5.439	3.199	3.986	4.738
206	16.77	15.62	0.8638	5.927	3.438	4.920	5.795
207	14.03	14.16	0.8796	5.438	3.201	1.717	5.001
208	16.12	15.00	0.9000	5.709	3.485	2.270	5.443
209	15.57	15.15	0.8527	5.920	3.231	2.640	5.879

From the above tables we can see the bottom 10 rows or entries of the dataset.

Check the shape of the dataset:

```
bank_marketing_df.shape
```

```
(210, 7)
```

From the above figure we can say that there are 210 rows and 7 columns available in the dataset.

Check the complete information of the dataset:

```
bank_marketing_df.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
```

From the above figure we can say that there are zero null values, datatype if of float, range index starts from 0 and ends at 210, columns names and the dataframe.

To check the mathematical summary:

```
bank_marketing_df.describe().T
```

	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

From the above table we can get the 5 point summary of the dataset such as their mean value, max value, number of count, 25th percentile, 75th percentile and the maximum value.

Let us check the types of variables in the data frame

```
spending                      float64
advance_payments              float64
probability_of_full_payment   float64
current_balance               float64
credit_limit                  float64
min_payment_amt               float64
max_spent_in_single_shopping  float64
dtype: object
```

There is total 210 rows and 7 columns. And out of 7 columns all 7 columns has datatype as float.

Check for missing values in the dataset:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210 entries, 0 to 209
Data columns (total 7 columns):
 #   Column                        Non-Null Count
---  ------                        --------------
 0   spending                      210 non-null
 1   advance_payments              210 non-null
 2   probability_of_full_payment   210 non-null
 3   current_balance               210 non-null
 4   credit_limit                  210 non-null
 5   min_payment_amt               210 non-null
 6   max_spent_in_single_shopping  210 non-null
dtypes: float64(7)
memory usage: 11.6 KB
```

From the above we can say that there are no missing value present in the dataset.

To check the Index of the dataset

```
bank_marketing_df.index
```

```
RangeIndex(start=0, stop=210, step=1)
```

From the above we can say that the index starts from 0 and ends at 210 with the interval of one step.

To check the columns of the dataset:

```
bank_marketing_df.columns
```

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

From the above we can say that the columns are as follows: spending, advance_payments, probability_of_full_payment, current_balance, credit_limit, min_payment_amt, max_spent_in_single_shopping.

To check the datatypes of the variables in the dataset:

```
bank_marketing_df.dtypes
```

```
spending                      float64
advance_payments              float64
probability_of_full_payment   float64
current_balance               float64
credit_limit                  float64
min_payment_amt               float64
max_spent_in_single_shopping  float64
dtype: object
```

From the above we can say that the datatypes of the variables in the dataset are in float not in string or Boolean.

Univariate Analysis:

Univariate analysis involves analysis of single variables.

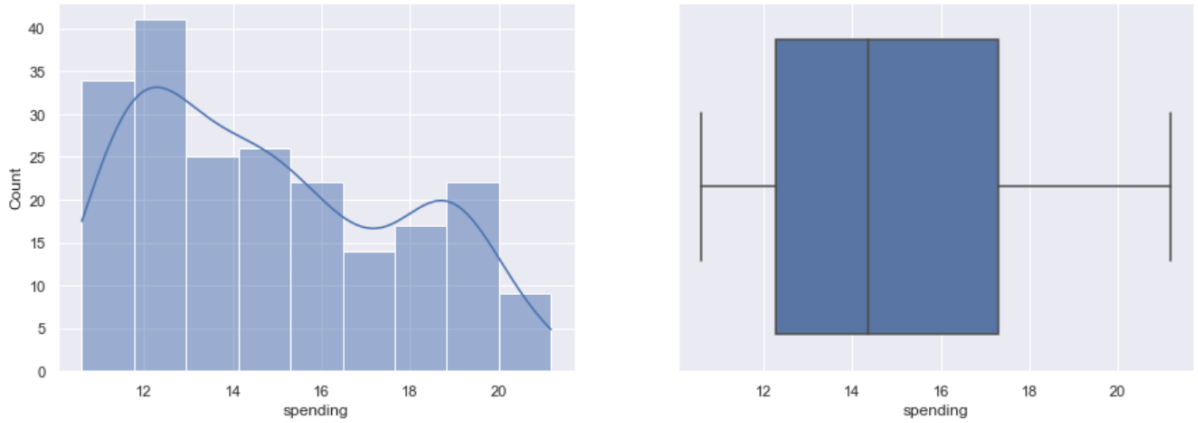


Fig 1. Spending

- The 25th quantile of spending is 12.27
- The median or 50th quantile of spending is 14.355
- The 75th quantile of spending is 17.305
- The distribution of spending is normal but little right skewed 0.3998891917177586

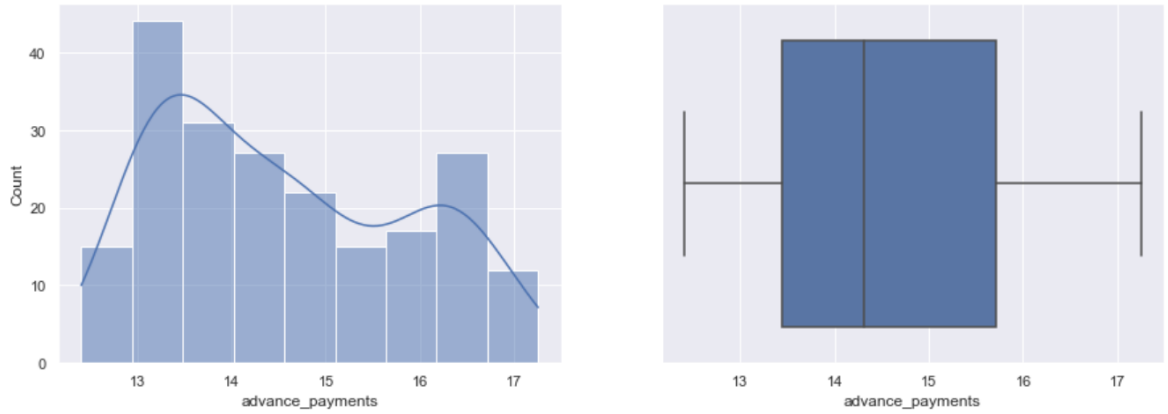


Fig 2. advance_payments

- The 25th quantile of advance_payments is 13.45
- The median or 50th quantile of advance_payments is 14.32
- The 75th quantile of advance_payments is 15.715
- The distribution of advance_payments is almost normal 0.3865727731912213

```
fig, axes = plt.subplots(nrows=1,ncols=2)
fig.set_size_inches(15,5)
sns.histplot(bank_marketing_df['probability_of_full_payment'], kde=True, ax=axes[0])
sns.boxplot(x='probability_of_full_payment', data=bank_marketing_df, ax=axes[1])
plt.show()
```

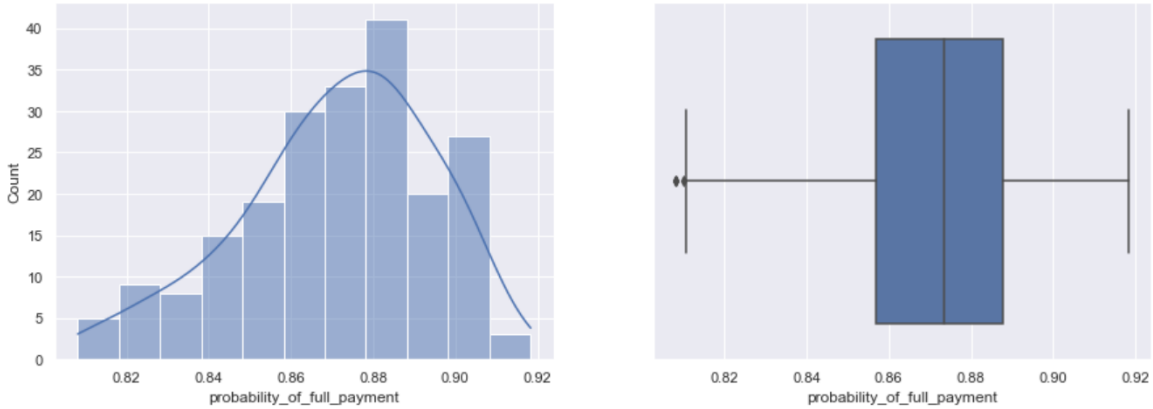


Fig 3. probability_of_full_payment

- The 25th quantile of probability_of_full_payment is 0.8569
- The median or 50th quantile of probability_of_full_payment is 0.8734500000000001
- The 75th quantile of probability_of_full_payment is 0.887775
- The distribution of probability_of_full_payment is left skewed -0.5379537283982823

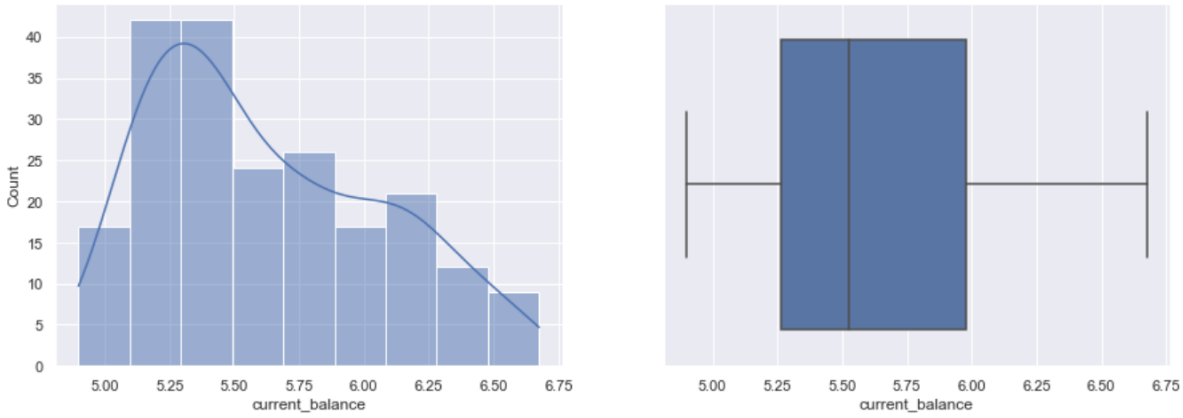


Fig 4. current_balance

- The 25th quantile of current_balance is 5.26225
- The median or 50th quantile of current_balance is 5.5235
- The 75th quantile of current_balancet is 5.97975
- The distribution of current_balance is normal but little right skewed 0.5254815601318906

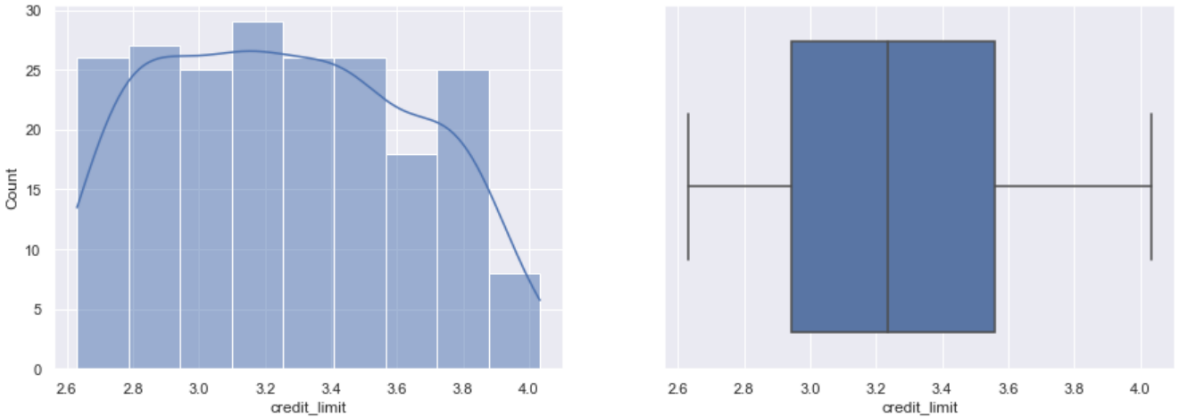


Fig 5. Univariate – credit_limit

- The 25th quantile of credit_limit is 2.944
- The median or 50th quantile of credit_limit is 3.237
- The 75th quantile of credit_limit is 3.56175
- The distribution of credit_limit is normal 0.1343782451316215

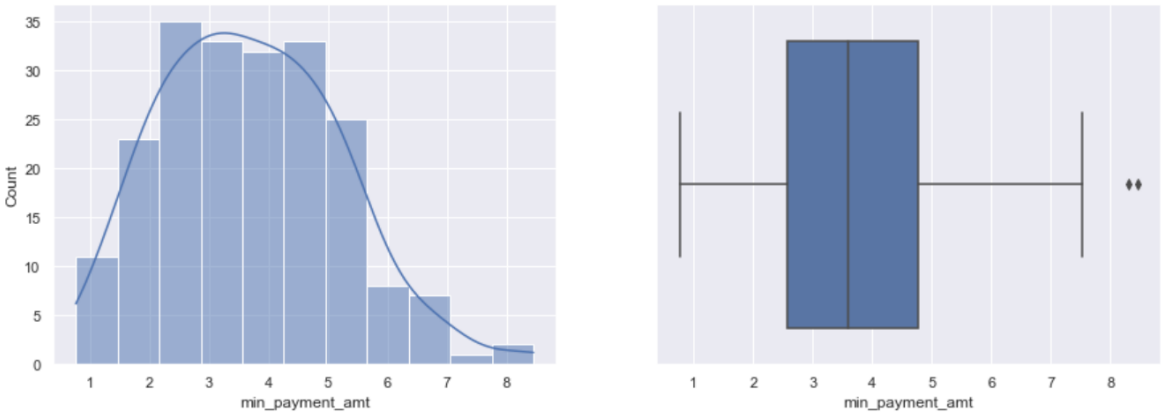


Fig 6. Univariate – min_payment_amt

- The 25th quantile of min_payment_amt is 2.5615
- The median or 50th quantile of min_payment_amt is 3.599
- The 75th quantile of min_payment_amt is 4.76875
- The distribution of min_payment_amt is normal 0.40166734329025183

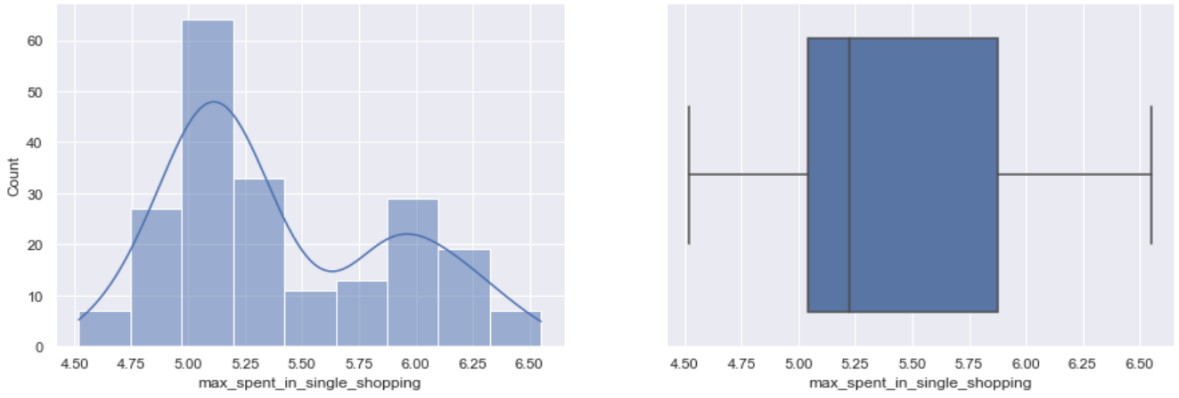


Fig 7. Univariate – max_spent_in_single_shopping

- The 25th quantile of max_spent_in_single_shopping is 5.045
- The median or 50th quantile of max_spent_in_single_shopping is 5.223000000000001
- The 75th quantile of max_spent_in_single_shopping is 5.877
- The distribution of max_spent_in_single_shopping is normal 0.561897374954866

Inference:

After plotting the Boxplots for all the variables, we can conclude that a few outliers are present in the variable namely, min_payment_amt which means that there are only a few customers whose minimum payment amount falls on the higher side on an average. We can conclude from the above graphs that most of the customers in our data have a higher spending capacity, high current balance in their bank accounts and these customers spent a higher amount during a single shopping event with their savings. Majority of the customers of the dataset have a higher probability to make full payment to the bank.

Bivariate & Multivariate Analysis:

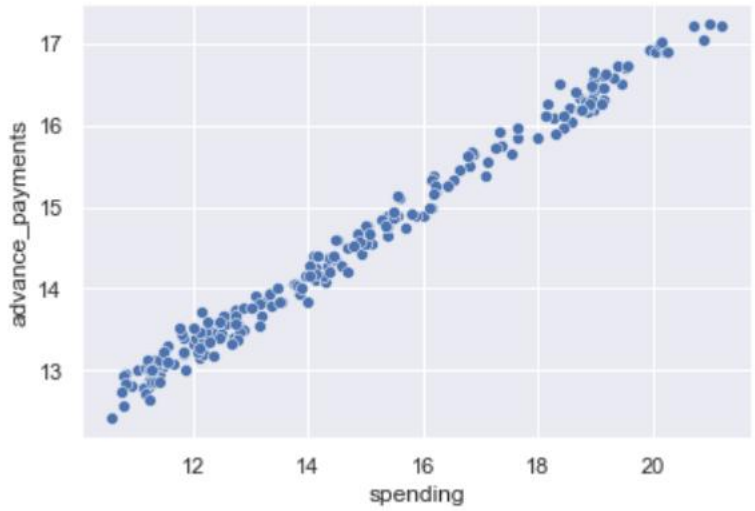


Fig 8. Bivariate – spending & advance_payments

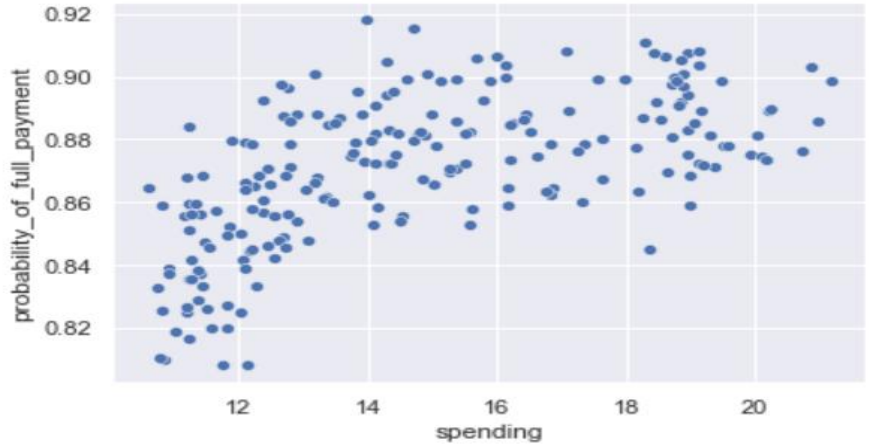


Fig 9. Bivariate – spending & probability_of_full_payment

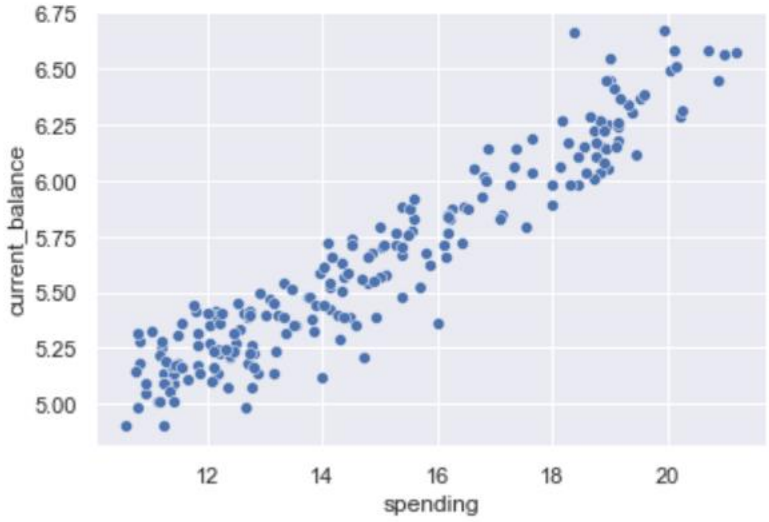


Fig 10. Bivariate – spending & current_balance

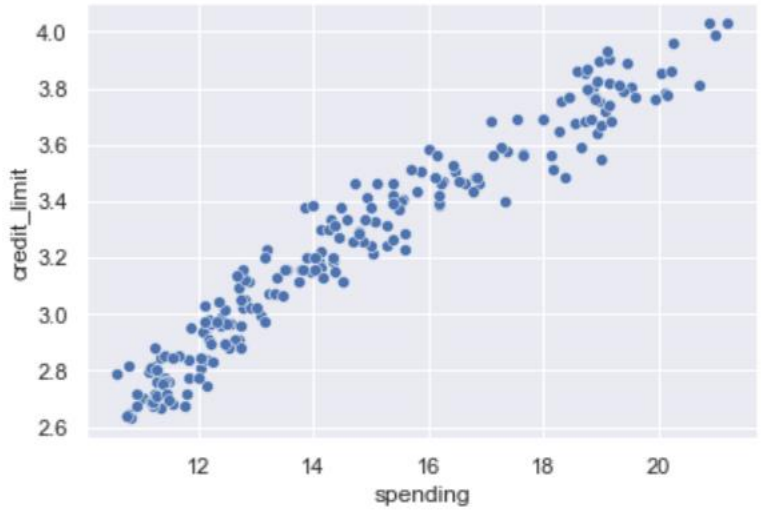


Fig 11. Bivariate – spending & credit_limit

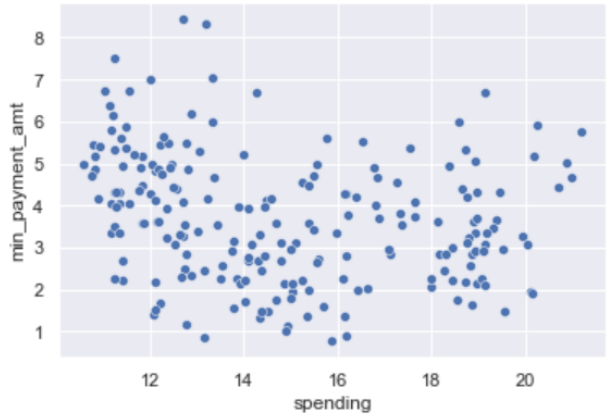


Fig 12. Bivariate – spending & min_payment_amt

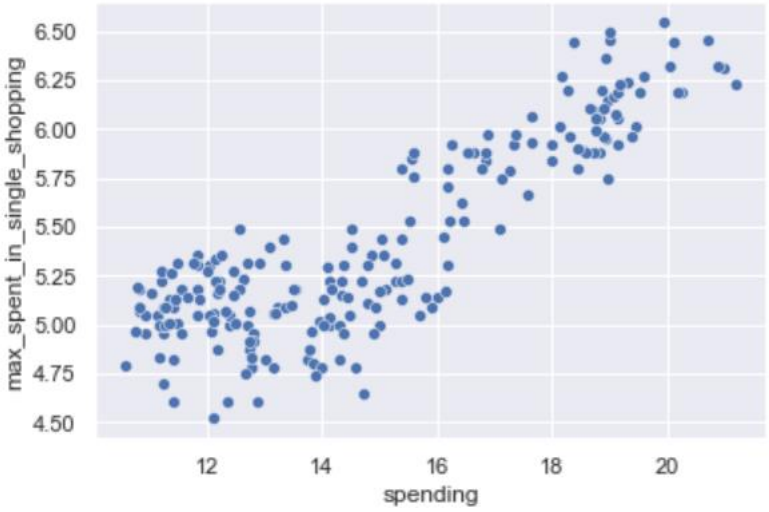
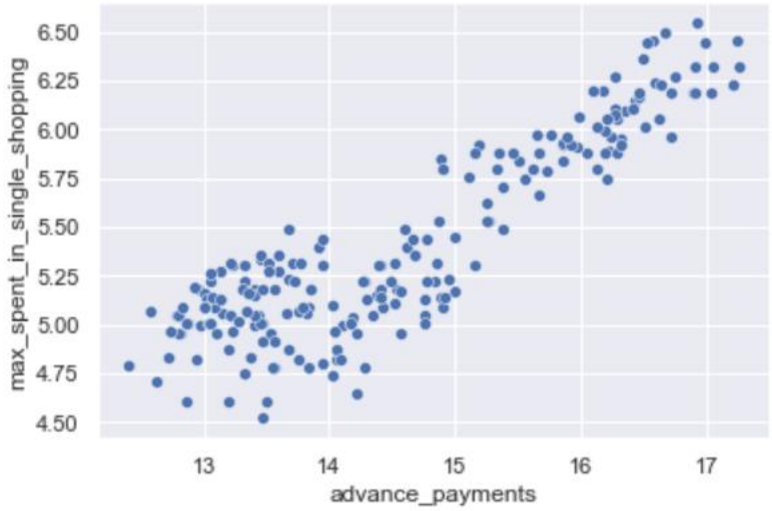


Fig 13. Bivariate – spending & max_spent_in_single_shopping



\

Fig 14 Bivariate – advance_payments & max_spent_in_single_shopping

Pair plot

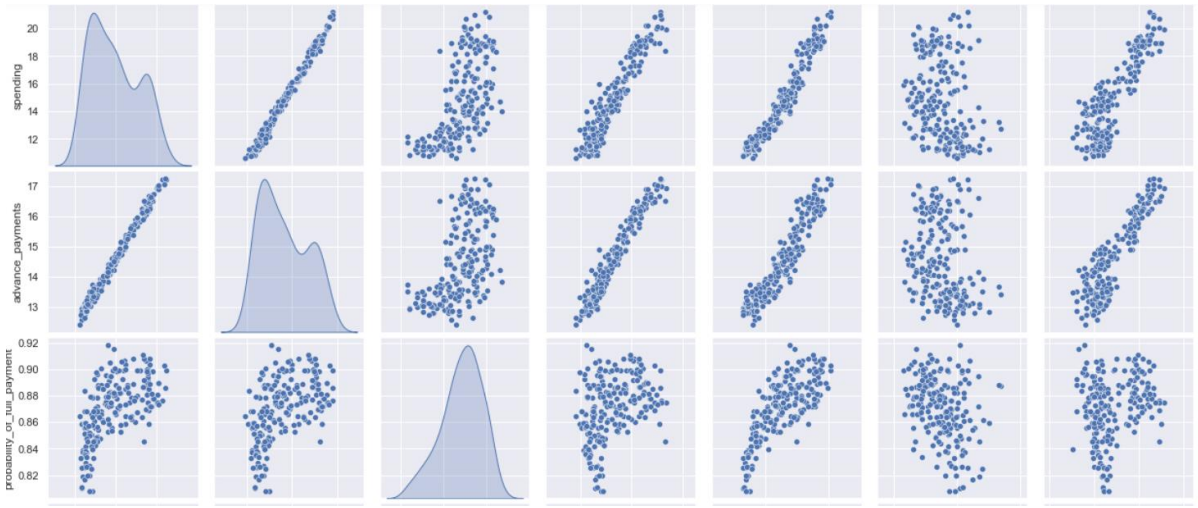


Fig15 Multivariate– PairPlot[bank_marketing_df]

HeatMap

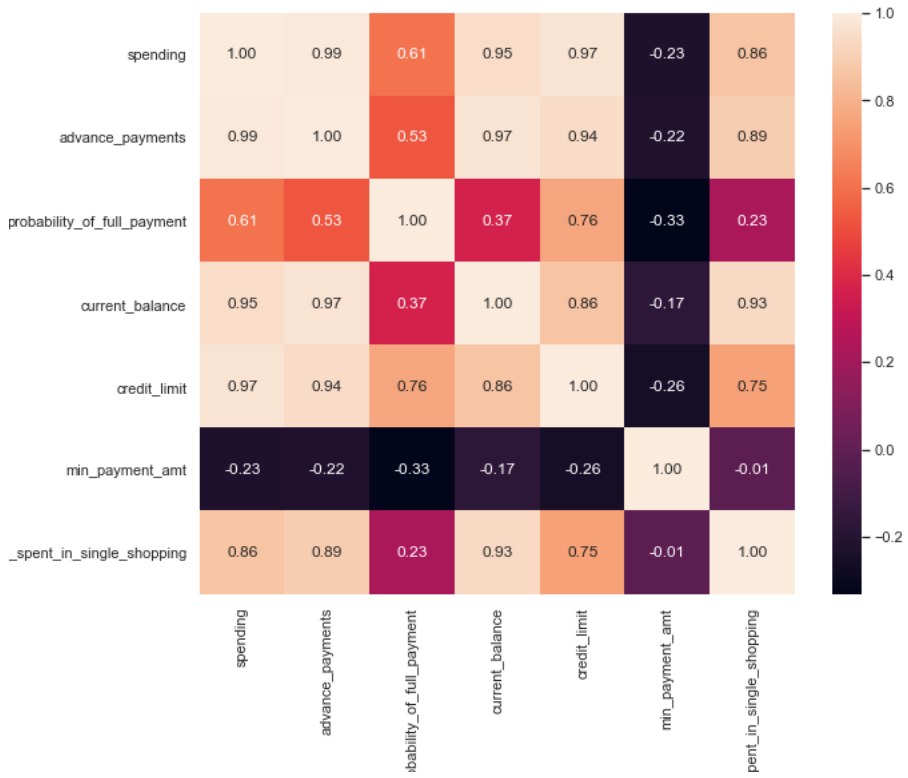


Fig16 Multivariate– HeatMap[bank_marketing_df]

Inference:

As per the Heat Map, we can conclude that the following variables are highly correlated: • Spending and advance_payments, spending and current_balance, spending and credit_limit • Advance payment and current_balance, advance payment, and credit limit • Current balance and max spent in single shopping By this we can conclude that the customers who are spending very high have a higher current balance and high credit limit. Advance payments and maximum expenditure done in single shopping are done by majority of those customers who have high current balance in their bank accounts. Probability of full payments are higher for those customers who have a higher credit limit. Minimum payment amount is not correlated to any of the variables; hence, it is not affected by any changes in the current balance or credit limit of the customers.

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

Yes, scaling is necessary for clustering in this case before building clustering algorithm scaling of data is important otherwise model will not be accurate and the values of the datapoint will differ. It is applied to independent variables which helps to normalise the data in a particular range. If feature scaling is not done,

then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values. For the data given to us, scaling is required as all the variables are expressed in different units such as spending in 1000's, advance payments in 100's and credit limit in 10000's, whereas probability is expressed as fraction or decimal values. Since the other values expressed in higher units will outweigh probabilities and can give varied results hence it is important to Scale the data using Standard Scaler and therefore normalise the values where the means will be 0 and standard deviation 1.

	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 2. Scaled Dataset

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

Cluster Analysis or Clustering is a widely accepted Unsupervised Learning technique in Machine Learning, Clustering can be divided into two categories namely, Hierarchical and K-means clustering.

Hierarchical clustering, also known as hierarchical cluster analysis, is an algorithm that groups similar objects into groups called clusters. The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly like each other. There are two types of hierarchical clustering, Divisive and Agglomerative. For the dataset in question, we will be using Agglomerative Hierarchical Clustering method to create optimum clusters and categorising the dataset on the basis of these clusters.

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

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

```
dend = dendrogram(wardlink)
```

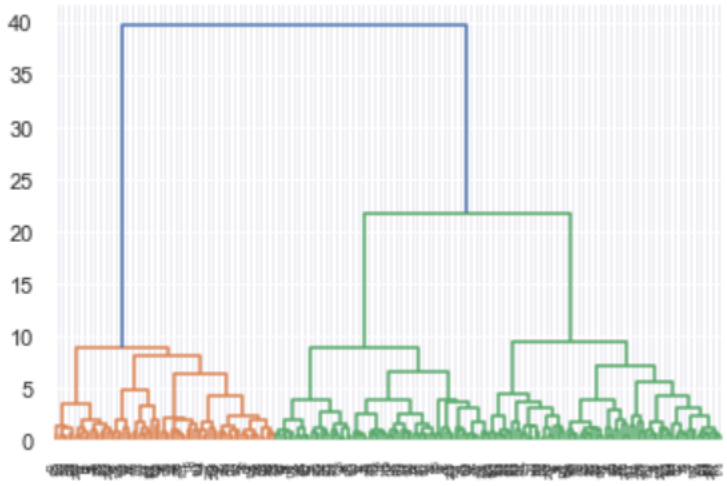


Fig16 Clustering

To create a Dendrogram using our scaled data we have firstly imported the package dendrogram, linkage from scipy.cluster.hierarchy. Using this function, we have created a dendrogram which shows two clusters very clearly. Now, we will check the make-up of these two clusters using 'maxclust' and 'distance'. As can be seen from above we will now take 2 clusters for our further analysis.

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

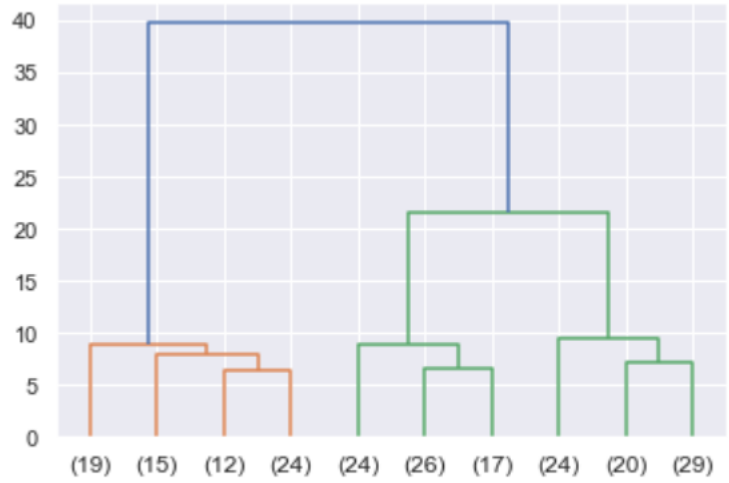


Fig17 Truncated Clustering

This above graph shows the last 10 links in the dendrogram.

```
dend = dendrogram(wardlink,
                  truncate_mode='lastp',
                  p = 5,
                  )
```

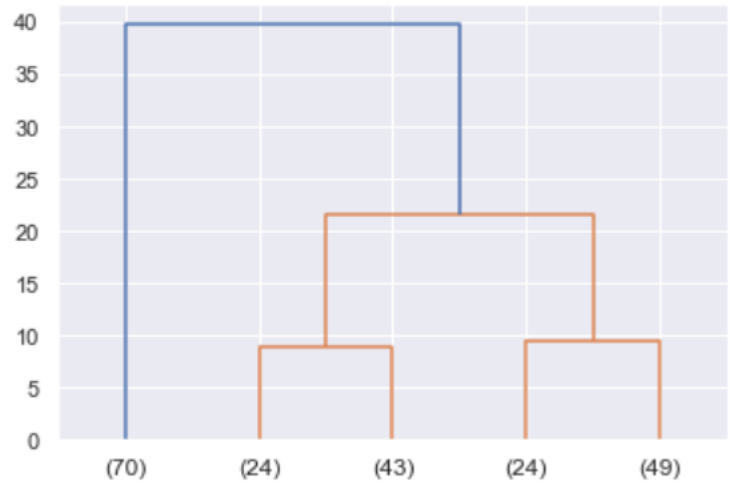


Fig17 Truncated Clustering (p=5)

This above graph shows the last 5 links in the dendrogram.

```
#Method 1

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

```
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)
```

Fig 18 Max Cluster using criterion = maxclust

```
#Method 2

clusters1 = fcluster(wardlink, 18, criterion='distance')
clusters1
```

```
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)
```

Fig 19 Max Cluster using criterion = distance

```
df.head()
```

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	clusters
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 3. Dataset with Clusters

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

Fig 20. Clustering- Sets

Cluster 1 having 70 rows, cluster 2 having 67 rows and cluster 3 having 73 rows.

Looking at the clusters and frequencies: -

• Cluster 1 has maximum power of spending and does maximum shopping in single visit as well as pays the amount in advance. The customer from has the highest probability of paying the full amount to the bank. As the current balance as well as credit limit is high, due to it increases the power of purchase. People in cluster 1 which is 70 out of 210 have more spending power as compared to other clusters.

• Overall, the three clusters are divided into high/medium/low spending based on maximum spent in single shopping.

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.

K-means clustering is one of the unsupervised machine learning algorithms. K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while

keeping the centroids as small as possible. For the dataset we will be using K-means clustering on scaled data and identify the clusters formed and use them further to devise tools to target each group separately. Firstly, we have scaled the dataset using StandardScaler package from sklearn.preprocessing and using the scaled_bank_marketing_df we will now plot two curves to determine the optimal number of clusters(k) to be used for our clustering.

The two methods are namely within sum of squares(wss) method and average silhouette scores method.

```
wss =[]
```

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

```
wss
```

```
[1469.9999999999995,
 659.1717544870411,
 430.65897315130064,
 371.301721277542,
 327.9608240079031,
 290.5900305968219,
 264.83153087478144,
 240.6837259501598,
 220.85285825594738,
 206.3829103601579]
```

Fig 21. WSS

```
a=[1,2,3,4,5,6,7,8,9,10]
sns.pointplot(a,wss);
```

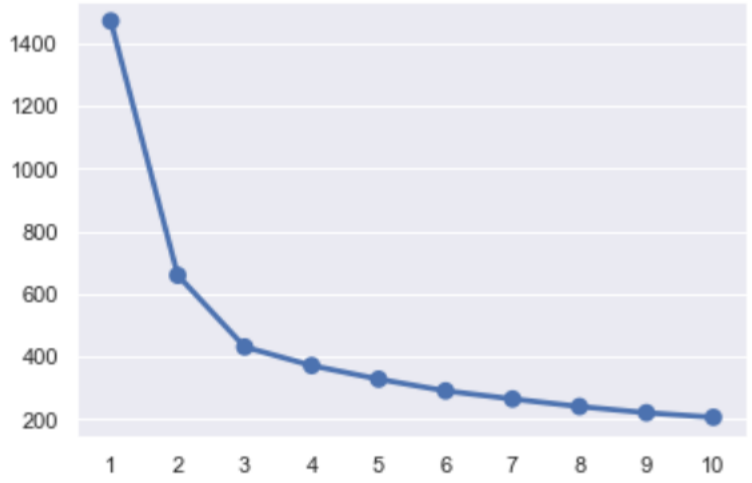


Fig 22. Elbow Curve

As per the above plot i.e. within sum of squares (wss) method we can conclude that the optimal number of clusters to be taken for k-means clustering is 3 since as per the elbow method it can be easily seen in the curve that after 3 the curve gets flat.

```
from sklearn.metrics import silhouette_samples, silhouette_score

k_means = KMeans(n_clusters = 2,random_state=1)
k_means.fit(scaled_bank_marketing_df)
labels_2 = k_means.labels_

# Cluster_2
silhouette_score(scaled_bank_marketing_df,labels_2, random_state=1)
```

```
0.46577247686580914
```

```
k_means = KMeans(n_clusters = 3,random_state=1)
k_means.fit(scaled_bank_marketing_df)
labels_3 = k_means.labels_
```

Fig 23. Silhouette score.

Average silhouette scores the highest Average score is corresponding to k=3. Hence, as per both the methods i.e. within sum of squares and silhouette method we can conclude that the optimal number of k or clusters that needs to be taken for k-means clustering is 3.

```
bank_marketing_df["kmeans3"] = labels_3
bank_marketing_df.head()
```

dvance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	clusters	Agglo_clusters	kmeans3
16.92	0.8752	6.675	3.763	3.252	6.550	1	1	1
14.89	0.9064	5.363	3.582	3.336	5.144	3	0	2
16.42	0.8829	6.248	3.755	3.368	6.148	1	1	1
12.96	0.8099	5.278	2.641	5.182	5.185	2	2	0
15.86	0.8992	5.890	3.694	2.068	5.837	1	1	1

Table 4. Dataset with kmeans

```
bank_marketing_df['sil_width'] = sil_width
bank_marketing_df.head()
```

yments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	clusters	Agglo_clusters	kmeans3	sil_width
16.92	0.8752	6.675	3.763	3.252	6.550	1	1	1	0.573699
14.89	0.9064	5.363	3.582	3.336	5.144	3	0	2	0.366386
16.42	0.8829	6.248	3.755	3.368	6.148	1	1	1	0.637784
12.96	0.8099	5.278	2.641	5.182	5.185	2	2	0	0.512458
15.86	0.8992	5.890	3.694	2.068	5.837	1	1	1	0.362276

Table 5. Dataset with sil_width

The average silhouettes score is 0.400 and minimum silhouette score is 0.002. The silhouette score ranges from -1 to +1 and higher the silhouette score better the clustering.

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

```
clust_profile=bank_marketing_df
clust_profile=clust_profile.groupby("kmeans3").mean()
clust_profile["Freq"]=bank_marketing_df['kmeans3'].value_counts().sort_index()
clust_profile
```

_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	clusters	Agglo_clusters	sil_width	Freq
13.247778	0.848253	5.231750	2.849542	4.742389	5.101722	2.083333	1.833333	0.397473	72
16.203433	0.884210	6.175687	3.697537	3.632373	6.041701	1.029851	0.985075	0.468772	67
14.337746	0.881597	5.514577	3.259225	2.707341	5.120803	2.873239	0.084507	0.339816	71

Table 6. Dataset with Frequency

Looking at the above summary, we can conclude:

K-means Cluster 0: This segment has higher spending per month, high current balance, and credit limit. This is the Prosperous or Upper class with majorly higher income. This segment can be targeted using various offers such as cards with rewards and loyalty points for every spent.

K-means Cluster 1: This segment has the lowest spending per month, lowest current balance, and credit limit. This is the Financially Stressed Class with very low income on an average. This segment can be targeted with cards with offers such as zero annual charges and lurking them with benefits such as free coupons or tickets and waivers on a variety of places.

K-means Cluster 2: This segment has must lower spending per month with low current balance and lower credit limit. This is the Middle Class with low incomes. This segment can be targeted with cards that have lower interest rates so as to encourage more spending.

2. Insurance Analysis

Executive Summary:

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.

Data Description:

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 percentage of sales)
10. Age of insured (Age)

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

Read the data

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

Pd.read_csv function is used to read the file in jupyter notebook environment.

Importing the packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set(color_codes=True)
import warnings
warnings.filterwarnings("ignore")
```

To read the dataset and do the analysis and exploratory data analysis we need to import packages in python jupyter notebook so that we can read the data and perform EDA.

Exploratory Data Analysis:

Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods. It helps determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions. EDA is primarily used to see what data can reveal beyond the formal modelling or hypothesis testing task and provides a provides a better understanding of data set variables and the relationships between them.

Sample of 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
5	45	JZI	Airlines	Yes	15.75	Online	8	45.00	Bronze Plan	ASIA
6	61	CWT	Travel Agency	No	35.64	Online	30	59.40	Customised Plan	Americas
7	36	EPX	Travel Agency	No	0.00	Online	16	80.00	Cancellation Plan	ASIA
8	36	EPX	Travel Agency	No	0.00	Online	19	14.00	Cancellation Plan	ASIA
9	36	EPX	Travel Agency	No	0.00	Online	42	43.00	Cancellation Plan	ASIA

Table 1. Dataset Sample

Dataset consists of 3000 rows and 10 unique columns.

Top 10 entries or head of the dataset:

```
: insurance_df.head(10)
```

:

	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
5	45	JZI	Airlines	Yes	15.75	Online	8	45.00	Bronze Plan	ASIA
6	61	CWT	Travel Agency	No	35.64	Online	30	59.40	Customised Plan	Americas
7	36	EPX	Travel Agency	No	0.00	Online	16	80.00	Cancellation Plan	ASIA
8	36	EPX	Travel Agency	No	0.00	Online	19	14.00	Cancellation Plan	ASIA
9	36	EPX	Travel Agency	No	0.00	Online	42	43.00	Cancellation Plan	ASIA

From the above tables we can see the top 10 rows or entries of the dataset.

Bottom 10 entries or tail of the dataset:

```
insurance_df.tail(10)
```

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	Destination
2990	51	EPX	Travel Agency	No	0.00	Online	2	20.00	Customised Plan	ASIA
2991	29	C2B	Airlines	Yes	48.30	Online	381	193.20	Silver Plan	ASIA
2992	28	CWT	Travel Agency	No	11.88	Online	389	19.80	Customised Plan	ASIA
2993	36	EPX	Travel Agency	No	0.00	Online	234	10.00	Cancellation Plan	ASIA
2994	27	C2B	Airlines	Yes	71.85	Online	416	287.40	Gold Plan	ASIA
2995	28	CWT	Travel Agency	Yes	166.53	Online	364	256.20	Gold Plan	Americas
2996	35	C2B	Airlines	No	13.50	Online	5	54.00	Gold Plan	ASIA
2997	36	EPX	Travel Agency	No	0.00	Online	54	28.00	Customised Plan	ASIA
2998	34	C2B	Airlines	Yes	7.64	Online	39	30.55	Bronze Plan	ASIA
2999	47	JZI	Airlines	No	11.55	Online	15	33.00	Bronze Plan	ASIA

From the above tables we can see the bottom 10 rows or entries of the dataset.

Check the shape of the dataset:

```
insurance_df.shape
```

```
(3000, 10)
```

From the above figure we can say that there are 3000 rows and 10 columns available in the dataset.

Check the complete information of the dataset:

```
insurance_df.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
```

From the above figure we can say that there are zero null values, datatype if of float, object, int range index starts from 0 and ends at 2999, columns names and the dataframe.

To check the mathematical summary:

	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

From the above table we can get the 5 point summary of the dataset such as their mean value, max value, number of count, 25th percentile, 75th percentile and the maximum value.

Let us check the types of variables in the data frame

```
Age              int64
Agency_Code     object
Type            object
Claimed         object
Commision      float64
Channel         object
Duration         int64
Sales          float64
Product Name    object
Destination     object
dtype: object
```

There is total 3000 rows and 10 columns. And out of 10 columns 2 columns has datatype as int, 6 columns as object type and 2 columns as float.

Check for missing values in the dataset:

```
0   Age             3000 non-null
1   Agency_Code     3000 non-null
2   Type            3000 non-null
3   Claimed         3000 non-null
4   Commision       3000 non-null
5   Channel         3000 non-null
6   Duration        3000 non-null
7   Sales           3000 non-null
8   Product Name    3000 non-null
9   Destination     3000 non-null
```

From the above we can say that there are no missing value present in the dataset.

To check the Index of the dataset

```
Index(['Age', 'Agency_Code', 'Type', 'Claimed', 'Commision', 'Channel',
       'Duration', 'Sales', 'Product Name', 'Destination'],
      dtype='object')
```

From the above we can say that the index starts from 0 and ends at 2999with the interval of one step.

To check the columns of the dataset:

```
insurance_df.columns
```

```
Index(['Age', 'Agency_Code', 'Type', 'Claimed', 'Commision', 'Channel',
       'Duration', 'Sales', 'Product Name', 'Destination'],
      dtype='object')
```

From the above we can say that the columns are as follow: Age, Agency code, Type, Claimed, Commisi on, Channel, Duration, Sales, ProductName, Destination.

Univariate Analysis:

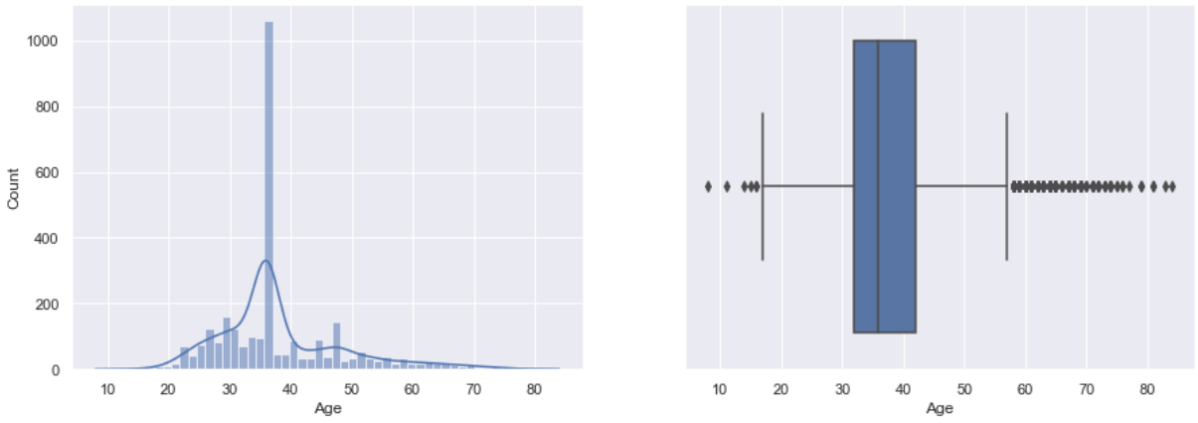


Fig 24. Univariate – Age

- The 25th quantile of Age is 32.0
- The median or 50th quantile of Age is 36.0
- The 75th quantile of Age is 42.0
- The distribution of Age is normal but little right skewed 1.149712770495169

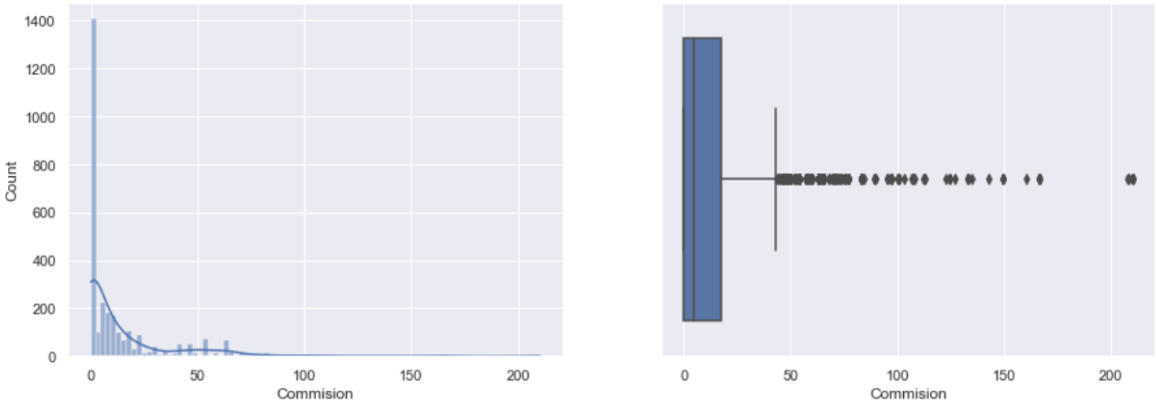


Fig 25. Univariate – Commission

```
The 25th quantile of is Commision 0.0
The median or 50th quantile of Commision is 4.63
The 75th quantile of Commision is 17.235
The distribution of Commision is normal but little right skewed 3.148857772356885
```

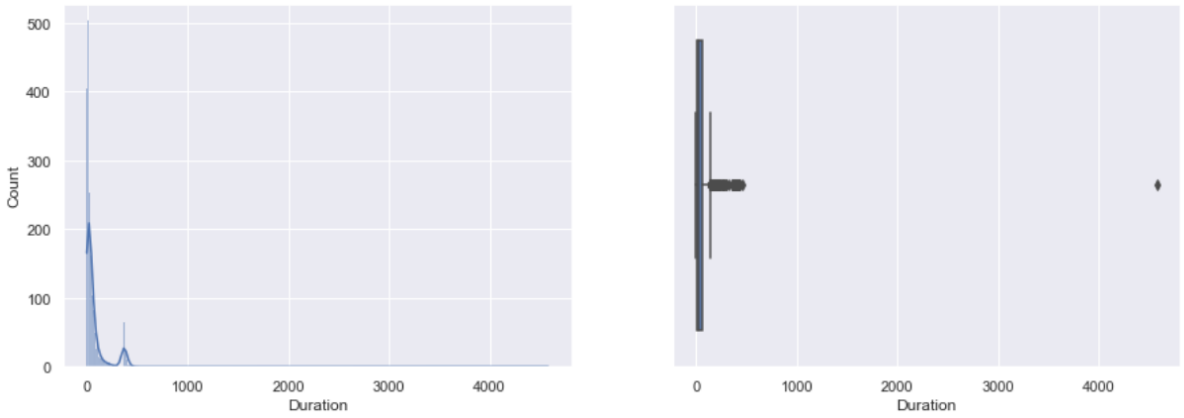


Fig 26. Univariate – Duration

```
The 25th quantile of Duration is 11.0
The median or 50th quantile of Duration is 26.5
The 75th quantile of Duration is 63.0
The distribution of Duration is normal but little right skewed 13.784681027519602
```

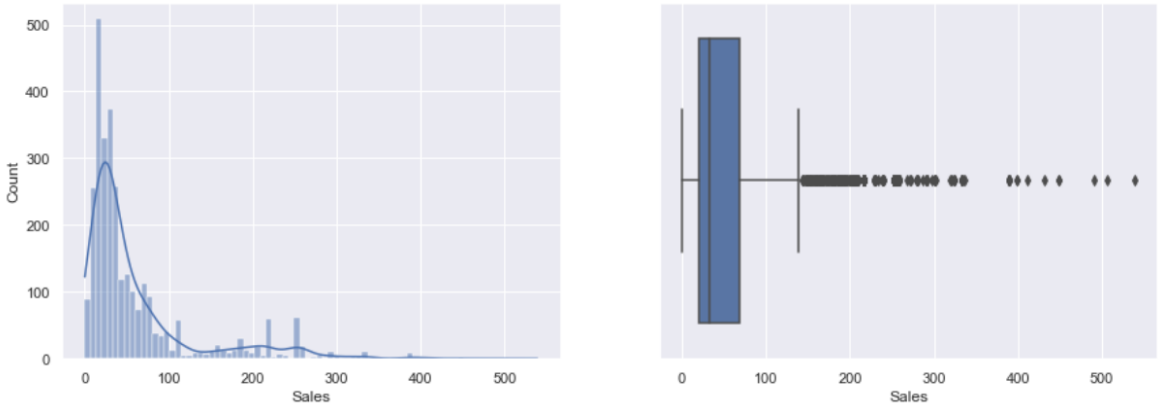


Fig 27. Univariate – Sales

```
The 25th quantile of Sales is 20.0
The median or 50th quantile of Sales is 33.0
The 75th quantile of Sales is 69.0
The distribution of Sales is normal but little right skewed 2.381148461687274
```

Bivariate Analysis:

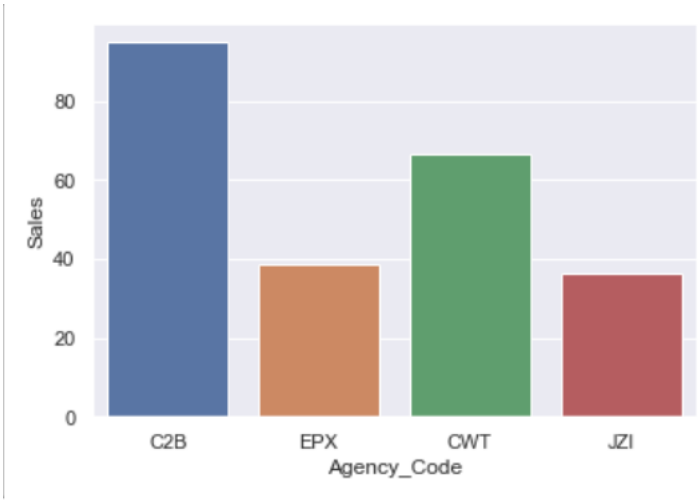


Fig 28. Agency code – Sales

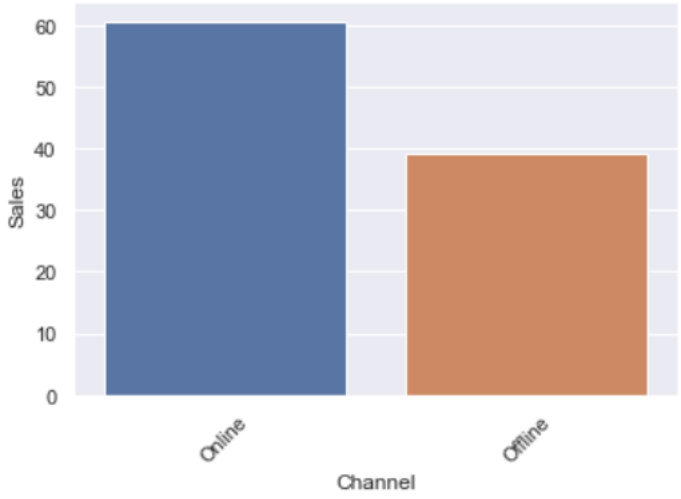


Fig 29. Channel – Sales

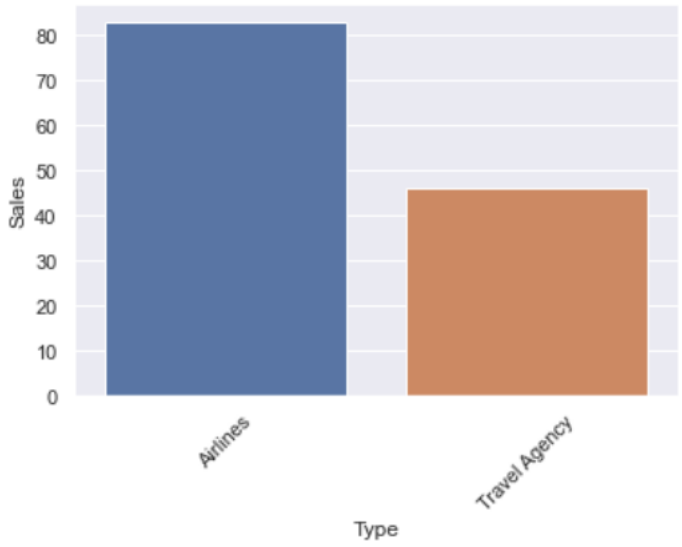


Fig 30. Type – Sales

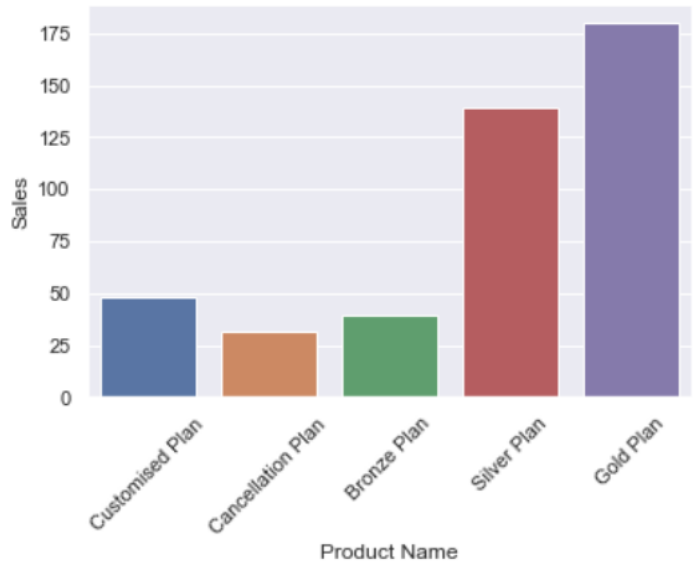


Fig 31. Product Name – Sales

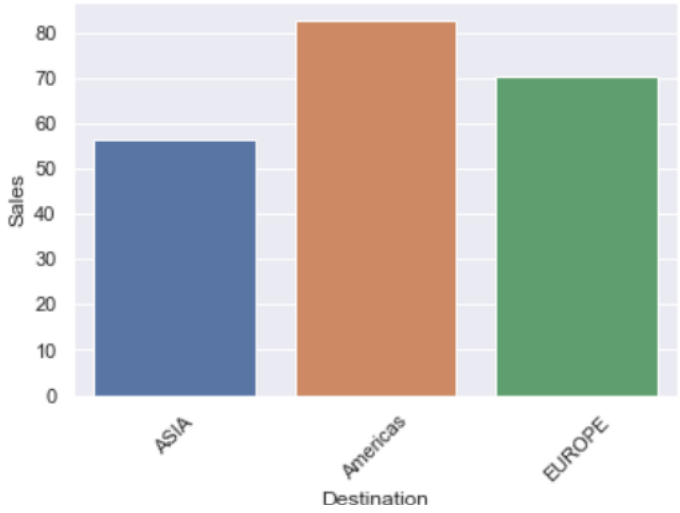


Fig 32. Destination – Sales

Multivariate Analysis:

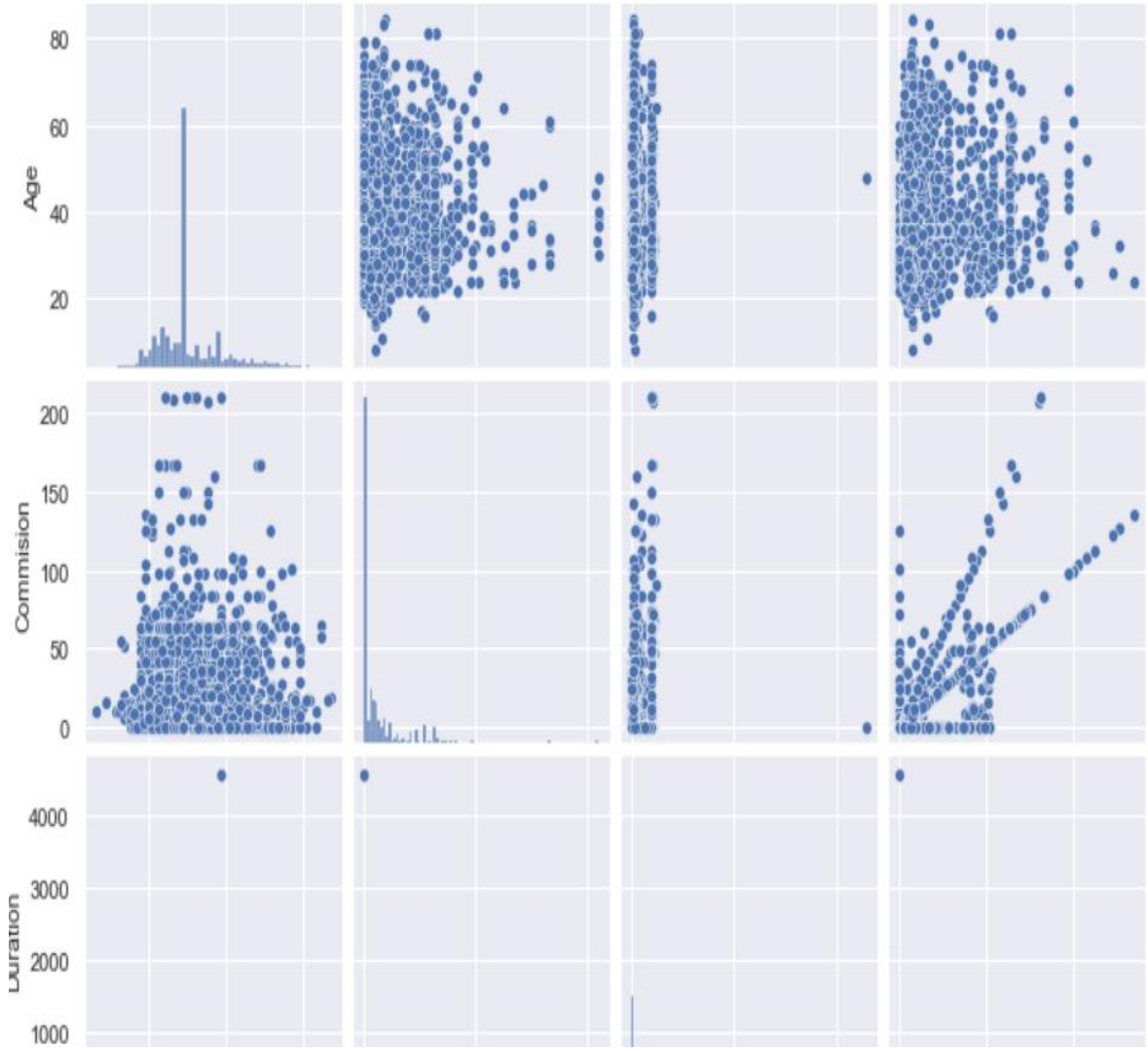


Fig 33. Pair plot

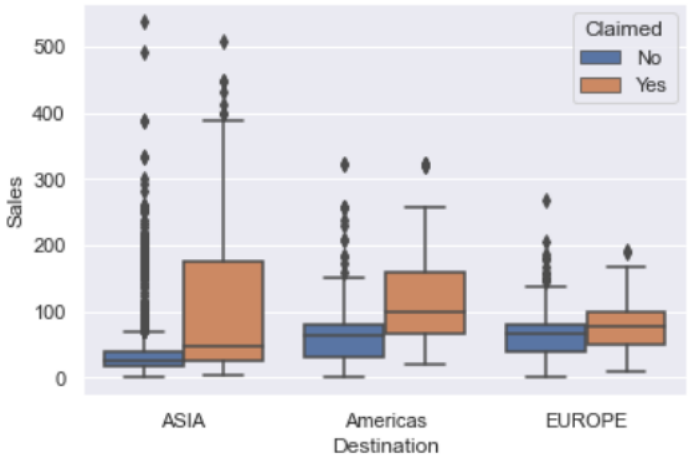


Fig 34. Destination & Sales

Inference:

After plotting the Boxplots for all the numerical variables we can conclude that a very high number of outliers are present in the variables namely, Age, Commision, Duration and Sales. We can conclude from the above graphs that most of the customers doing a claim in our data belong to age group of 25-40 with the type of Tour Agency firm being Travel Agency, Channel being Online, Product name being Customised Plan and Destination being Asia.

Heat Map

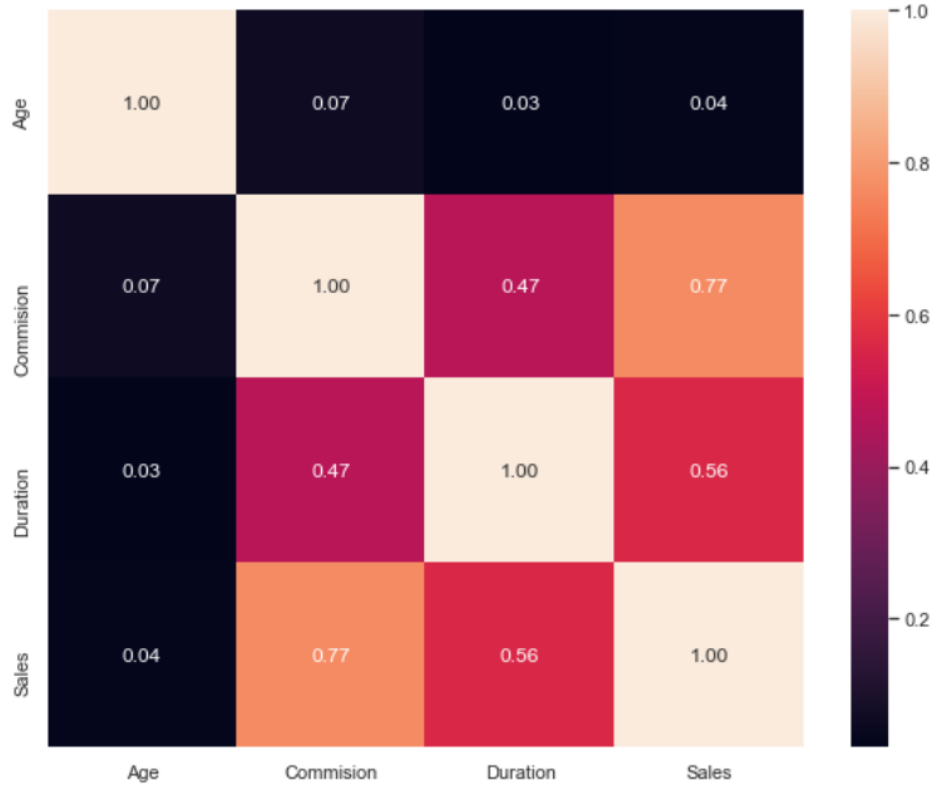


Fig 35. Heat Map

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

Converting Object datatype to Int:

```
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Age           2861 non-null   int64
 1   Agency_Code   2861 non-null   int8
 2   Type          2861 non-null   int8
 3   Claimed       2861 non-null   int8
 4   Commision     2861 non-null   float64
 5   Channel       2861 non-null   int8
 6   Duration      2861 non-null   int64
 7   Sales         2861 non-null   float64
 8   Product Name  2861 non-null   int8
 9   Destination   2861 non-null   int8
dtypes: float64(2), int64(2), int8(6)
memory usage: 208.5 KB
```

For our analysis and building Decision tree and Random Forest, we must convert the variables which have 'object' datatype and convert them into integer.

	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 8. Insurance Dataset – Conversion

Splitting Dataset in Train and Test Data (70:30)

For building the models we will now have to split the dataset into Training and Testing Data with the ratio of 70:30. These two datasets are stored in X_train and X_test with their corresponding dimensions as follows

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

X = df2.drop("Claimed" , axis=1)

y = df2.pop("Claimed")
```

```
#Splitting the data into Train & Test sets

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

```
#Checking the dimensions on train 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',2002+859)
```

```
X_train (2002, 9)
X_test (859, 9)
train_labels (2002,)
test_labels (859,)
Total Obs 2861
```

Fig 36. Train Test Data

CART MODEL:

```
#Initialise a Decission Tree
dt_model = DecisionTreeClassifier(criterion = 'gini')
```

```
#Fit model
dt_model.fit(X_train,train_labels)
```

```
3]: DecisionTreeClassifier()
```

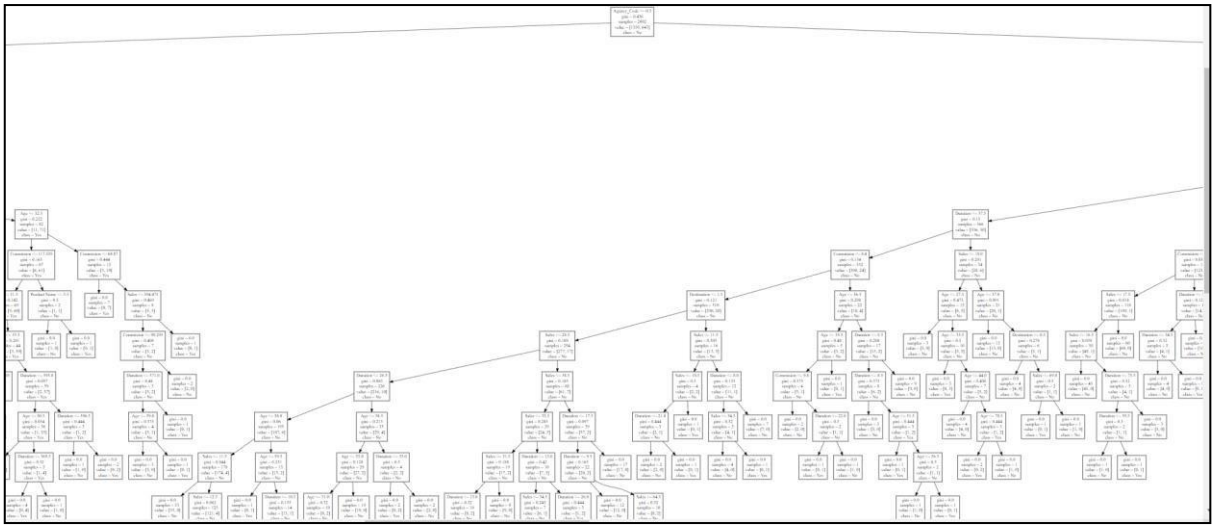


Fig 44. Decision Tree

- As we can see the decision tree has overgrown and too many branches are grown. We need toprune the model.

- The depth of the tree is large, and we are unable to predict.
- By using the Grid search method, we can find the best parameters and best estimator.

Regularising the Decision Tree

```
GridSearchCV(cv=10, estimator=RandomForestClassifier(random_state=1),
             param_grid={'criterion': ['gini'], 'max_depth': [5, 6, 8, 10],
                         'min_samples_leaf': [40, 45, 50],
                         'min_samples_split': [200, 250, 350, 450]})
```

Best Parameters

```
{'criterion': 'gini',
 'max_depth': 8,
 'min_samples_leaf': 50,
 'min_samples_split': 200}
```

These best grid parameters are henceforth used to build the regularised or pruned Decision tree.

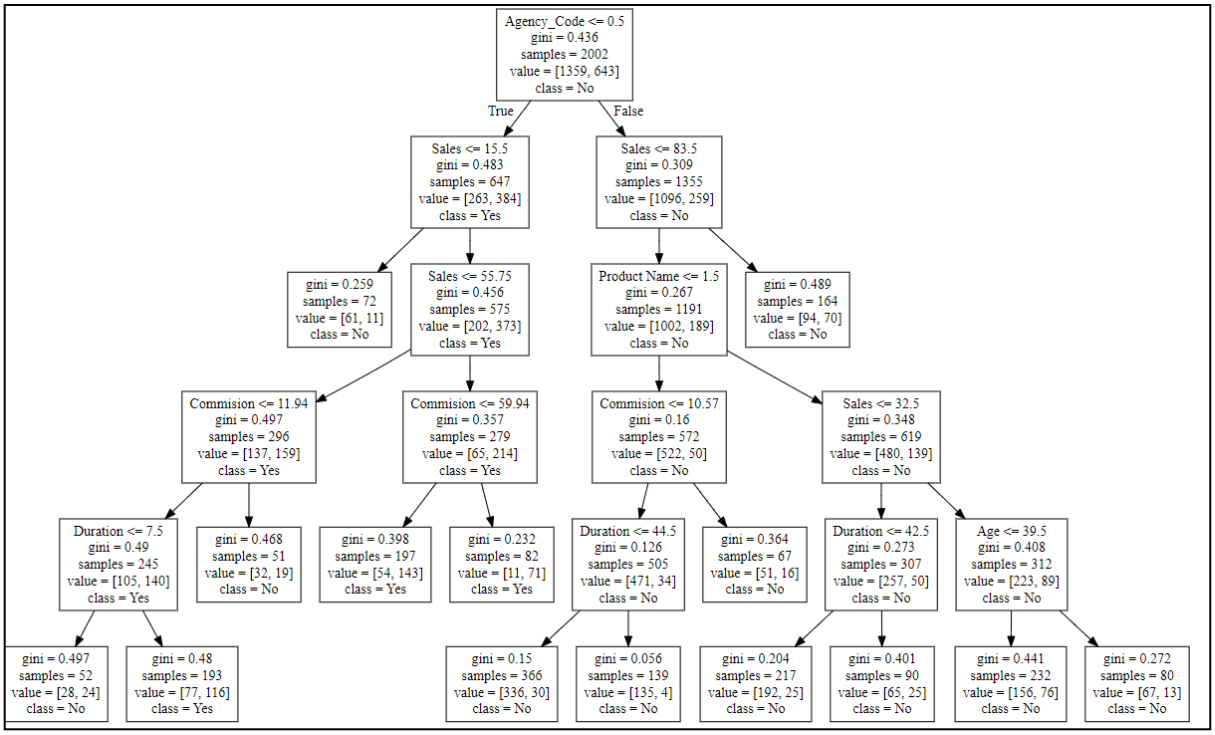


Fig 45. Regularising Decision Tree

Variable Importance

	Imp
Agency_Code	0.338088
Product Name	0.223942
Sales	0.180290
Commision	0.125910
Type	0.062099
Duration	0.040351
Age	0.019066
Destination	0.010254
Channel	0.000000

The agency code is the most important variable, and it is the root node.

Random Forest

Using the Train Dataset(X_train) we will be creating a Random Forest model and then further testing the model on Test Dataset(X_test).

Grid Search

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=1),
             param_grid={'criterion': ['gini'], 'max_depth': [3],
                         'max_features': [3, 4], 'min_samples_leaf': [8, 45],
                         'min_samples_split': [45],
                         'n_estimators': [200, 350]})
```

Best Parameters

```
{'criterion': 'gini',
 'max_depth': 3,
 'max_features': 3,
 'min_samples_leaf': 45,
 'min_samples_split': 45,
 'n_estimators': 200}
```

Using these best parameters evaluated using GridSeachCV a Random Forest Model is created which is further used for model performance evaluation.

Variable Importance

	Imp
Agency_Code	0.338088
Product Name	0.223942
Sales	0.180290
Commision	0.125910
Type	0.062099
Duration	0.040351
Age	0.019066
Destination	0.010254
Channel	0.000000

The agency code is the most important variable, and it is the root node.

Artificial Neural Network (ANN)

Firstly, we will have to Scale the two datasets.

```
array([[ 2.88764239, -1.2626112 , -1.19813318, ..., -0.65375471,
        -1.31338076, -0.44775345],
       [-0.21666128,  0.71683095,  0.83463176, ..., -0.37032806,
         0.24339146, -0.44775345],
       [ 2.04101412, -0.27289013,  0.83463176, ...,  0.11574864,
         0.24339146,  1.24676906],
       ...,
       [-0.21666128,  0.71683095,  0.83463176, ..., -0.68209737,
        -0.53499465, -0.44775345],
       [-0.21666128,  0.71683095,  0.83463176, ...,  0.72086453,
         0.24339146, -0.44775345],
       [-0.21666128,  0.71683095,  0.83463176, ...,  0.72086453,
         0.24339146,  1.24676906]])
```

Fig 46. Scaled Trained Data

```
array([[-0.68701032, -0.27289013,  0.83463176, ...,  0.50829455,
         0.24339146, -0.44775345],
       [ 2.79357258,  0.71683095,  0.83463176, ..., -0.45535606,
        -0.53499465, -0.44775345],
       [ 0.34775757, -1.2626112 , -1.19813318, ...,  0.32406723,
         1.80016368, -0.44775345],
       ...,
       [ 1.19438584, -1.2626112 , -1.19813318, ..., -0.63958338,
        -1.31338076, -0.44775345],
       [ 1.38252546,  0.71683095,  0.83463176, ..., -0.56872671,
         0.24339146, -0.44775345],
       [-0.21666128,  0.71683095,  0.83463176, ..., -0.56872671,
         0.24339146, -0.44775345]])
```

Fig 47. Scaled Test D

Grid Search

```
GridSearchCV(cv=5, estimator=MLPClassifier(random_state=1),
             param_grid={'hidden_layer_sizes': [50, 100, 150, 200],
                         'max_iter': [2500, 3000, 4000, 5000],
                         'solver': ['sgd', 'adam'], 'tol': [0.01]})
```

Best Parameters

```
{'hidden_layer_sizes': 100, 'max_iter': 2500, 'solver': 'adam', 'tol': 0.01}
```

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_AUCscore, classification reports for each model.

To check the Model Performances of the three models created above certain model evaluators are used i.e., Classification Report, Confusion Matrix, ROC_AUC Score and ROC Plot. They are calculated first for train data and then for test data.

CART Model

Classification Report

```
print(classification_report(train_labels, ytrain_predict))
```

	precision	recall	f1-score	support
0	0.80	0.89	0.84	1359
1	0.70	0.51	0.59	643
accuracy			0.77	2002
macro avg	0.75	0.70	0.72	2002
weighted avg	0.76	0.77	0.76	2002

Fig 48. Classification Report - Train

```
print(classification_report(test_labels, ytest_predict))
```

	precision	recall	f1-score	support
0	0.80	0.90	0.85	588
1	0.70	0.51	0.59	271
accuracy			0.78	859
macro avg	0.75	0.70	0.72	859
weighted avg	0.77	0.78	0.77	859

Fig 49. Classification Report - Test

Confusion Matrix

```
confusion_matrix(train_labels, ytrain_predict)
```

```
array([[1214,  145],
       [ 312,  331]], dtype=int64)
```

Fig 50. Confusion Matrix - Train

```
confusion_matrix(test_labels, ytest_predict)
```

```
array([[529,  59],
       [133, 138]], dtype=int64)
```

Fig 51. Confusion Matrix - Test

ROC_AUC Score and ROC Curve

```
# predict probabilities
probs = best_grid.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
cart_train_auc = roc_auc_score(train_labels, probs)
# print('AUC: %.3f' % cart_train_auc)
# calculate roc curve
from sklearn.metrics import roc_curve
cart_train_fpr, cart_train_tpr, cart_train_thresholds = roc_curve(train_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(cart_train_fpr,cart_train_tpr, marker='.')
# show the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
print('Area under Curve is', cart_train_auc)
plt.show()
```

```
Area under Curve is 0.810897799017437
```

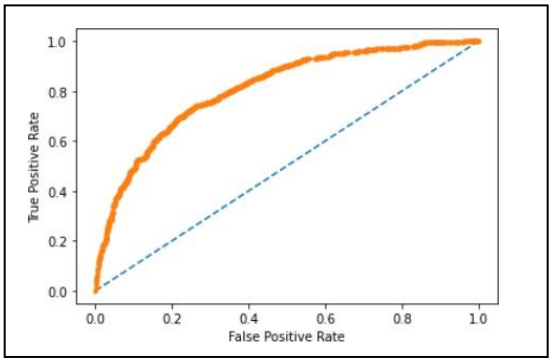


Fig 52. ROC_AUC Score and ROC Curve - Train

```
# predict probabilities
probs = best_grid.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
cart_test_auc = roc_auc_score(test_labels, probs)
# print('AUC: %.3f' % cart_test_auc)
# calculate roc curve
from sklearn.metrics import roc_curve
cart_test_fpr, cart_test_tpr, cart_test_thresholds = roc_curve(test_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(cart_test_fpr, cart_test_tpr, marker='.')
# show the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
print('Area under Curve is', cart_test_auc)
plt.show()
```

```
Area under Curve is 0.8185700479453774
```

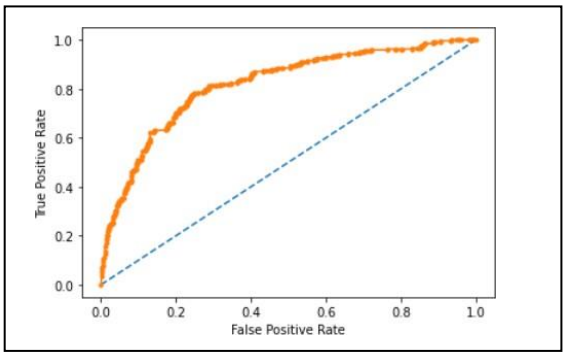


Fig 53. ROC_AUC Score and ROC Curve - Test

Cart Conclusion

```
CART Train-Accuracy 0.7717282717282717
CART Train-Recall/Sensitivity 0.5147744945567652
CART Train-Specificity 0.8933038999264165
CART Train-Precision 0.6953781512605042
CART Train-F1 Score 0.5915996425379804
```

```
CART Test-Accuracy 0.7764842840512224
CART Test-Recall/Sensitivity 0.5092250922509225
CART Test-Specificity 0.8996598639455783
CART Test-Precision 0.700507614213198
CART Test-F1 Score 0.5897435897435898
```

- Training and Testing set results are almost similar, and with the overall measures high, themodel is good.
- Agency code is the most important variable for pending “Claimed”

Random Forest Classification Report

	precision	recall	f1-score	support
0	0.79	0.90	0.84	1359
1	0.71	0.49	0.58	643
accuracy			0.77	2002
macro avg	0.75	0.70	0.71	2002
weighted avg	0.76	0.77	0.76	2002

Fig 54. Classification Report – Train

```
print(classification_report(test_labels, ytest_predict))
```

	precision	recall	f1-score	support
0	0.79	0.91	0.85	588
1	0.71	0.47	0.57	271
accuracy			0.77	859
macro avg	0.75	0.69	0.71	859
weighted avg	0.76	0.77	0.76	859

Fig 55. Classification Report – Test

Confusion Matrix

```
confusion_matrix(train_labels, ytrain_predict)
```

```
array([[1228,  131],
       [ 329,  314]], dtype=int64)
```

Fig 56. Confusion Matrix – Train

```
confusion_matrix(test_labels, ytest_predict)
```

```
array([[536,  52],
       [143, 128]], dtype=int64)
```

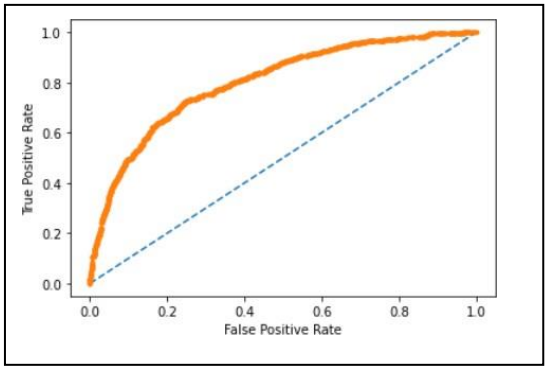
Fig 57. Confusion Matrix – Test

ROC_AUC Score and ROC Curve

Fig 58. ROC_AUC Score and ROC Curve – Train

```
# predict probabilities
probs = best_grid.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
rf_train_auc = roc_auc_score(train_labels, probs)
# print('AUC: %.3f' % rf_train_auc)
# calculate roc curve
from sklearn.metrics import roc_curve
rf_train_fpr, rf_train_tpr, rf_train_thresholds = roc_curve(train_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(rf_train_fpr,rf_train_tpr, marker='.')
# show the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
print('Area under Curve is', rf_train_auc)
plt.show()
```

```
Area under Curve is 0.8033140047857896
```




```
# predict probabilities
probs = best_grid.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
rf_test_auc = roc_auc_score(test_labels, probs)
# print('AUC: %.3f' % rf_test_auc)
# calculate roc curve
from sklearn.metrics import roc_curve
rf_test_fpr,rf_test_tpr, rf_test_thresholds = roc_curve(test_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(rf_test_fpr,rf_test_tpr, marker='.')
# show the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
print('Area under Curve is', rf_test_auc)
plt.show()
```

```
Area under Curve is 0.8120309009212541
```

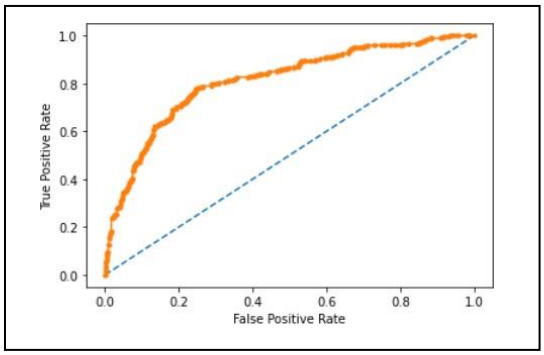


Fig 59. ROC_AUC Score and ROC Curve – Test

Random Forest Conclusion

```
Random Forest Training data-Accuracy 0.7702297702297702
Random Forest Training data-Recall/Sensitivity 0.48833592534992226
Random Forest Training data-Specificity 0.9036055923473142
Random Forest Training data-Precision 0.7056179775280899
Random Forest Training data-F1 Score 0.5772058823529411
```

```
Random Forest Test Data-Accuracy 0.7729918509895227
Random Forest Test Data-Recall/Sensitivity 0.47232472324723246
Random Forest Test Data-Specificity 0.9115646258503401
Random Forest Test Data-Precision 0.7111111111111111
Random Forest Test Data-F1 Score 0.5676274944567627
```

- Training and Testing set results are almost similar, and with the overall measures high, themodel is good.
- Agency code is the most important variable for pending “Claimed”

Artificial Neural Network (ANN)

Classification Report

```
print(classification_report(train_labels, ytrain_predict))
```

	precision	recall	f1-score	support
0	0.80	0.85	0.83	1359
1	0.64	0.56	0.60	643
accuracy			0.76	2002
macro avg	0.72	0.70	0.71	2002
weighted avg	0.75	0.76	0.75	2002

Fig 60. Classification Report– Test

```
print(classification_report(test_labels, ytest_predict))
```

	precision	recall	f1-score	support
0	0.81	0.87	0.84	588
1	0.66	0.55	0.60	271
accuracy			0.77	859
macro avg	0.73	0.71	0.72	859
weighted avg	0.76	0.77	0.76	859

Fig 61. Classification Report– Train

ROC_AUC Score and ROC Curve

```
# predict probabilities
probs = best_grid.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
nn_train_auc = roc_auc_score(train_labels, probs)
# print('AUC: %.3f' % rf_train_auc)
# calculate roc curve
from sklearn.metrics import roc_curve
nn_train_fpr, nn_train_tpr, nn_train_thresholds = roc_curve(train_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(rf_train_fpr,rf_train_tpr, marker='.')
# show the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
print('Area under Curve is', nn_train_auc)
plt.show()
```

```
Area under Curve is 0.7921265636497425
```

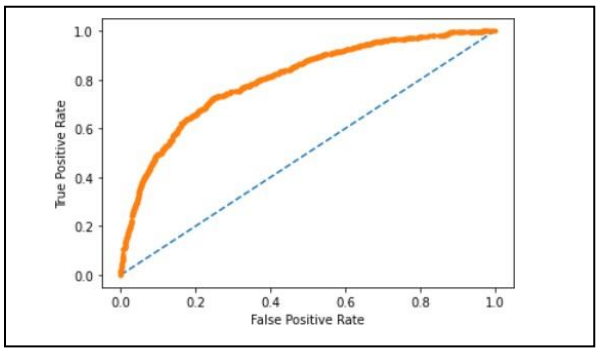


Fig 62. ROC_AUC Score and ROC Curve– Test

```
# predict probabilities
probs = best_grid.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
nn_test_auc = roc_auc_score(test_labels, probs)
# print('AUC: %.3f' % rf_test_auc)
# calculate roc curve
from sklearn.metrics import roc_curve
nn_test_fpr,nn_test_tpr, nn_test_thresholds = roc_curve(test_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(rf_test_fpr,rf_test_tpr, marker='.')
# show the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
print('Area under Curve is', nn_test_auc)
plt.show()
```

```
Area under Curve is 0.7977947636619223
```

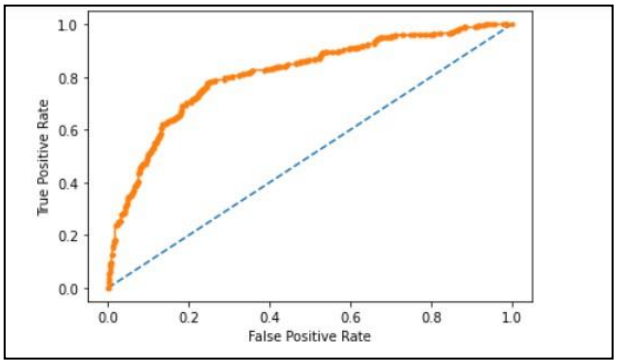


Fig 63. ROC_AUC Score and ROC Curve– Train

Confusion Matrix

```
confusion_matrix(train_labels, ytrain_predict)
```

```
array([[1159,  200],
       [ 285,  358]], dtype=int64)
```

Fig 64. Confusion Matrix– Train

```
confusion_matrix(test_labels, ytest_predict)

array([[511,  77],
       [122, 149]], dtype=int64)
```

Fig 65. Confusion Matrix– Test

Artificial Neural Network (ANN) Conclusion

```
Neutral Network Training data-Accuracy 0.7577422577422578
Neutral Network Training data-Recall/Sensitivity 0.5567651632970451
Neutral Network Training data-Specificity 0.8528329654157468
Neutral Network Training data-Precision 0.6415770609318996
Neutral Network Training data-F1 Score 0.5961698584512906
```

- Training and Testing set results are almost similar, and with the overall measures high, the model is good.
- Agency code is the most important variable for pending "Claimed"

```
Neutral Network Test data-Accuracy 0.7683352735739232
Neutral Network Test data-Recall/Sensitivity 0.5498154981549815
Neutral Network Test data-Specificity 0.8690476190476191
Neutral Network Test data-Precision 0.6592920353982301
Neutral Network Test data-F1 Score 0.5995975855130785
```

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

Comparison of all the performance evaluators for the three models are given in the following table. We are using Precision, F1 Score and AUC Score for our evaluation.

	CART Train	CART Test	Random Forest Train	Random Forest Test	Neural Network Train	Neural Network Test
Accuracy	0.77	0.78	0.77	0.77	0.76	0.77
AUC	0.81	0.82	0.80	0.81	0.79	0.80
Recall	0.51	0.51	0.49	0.47	0.56	0.55
Precision	0.70	0.70	0.71	0.71	0.64	0.66
F1 Score	0.59	0.59	0.58	0.57	0.60	0.60

Table 9. Model Comparison

From the above table, comparing the model performance evaluators for the three models it is quite clear that the Random Forest Model is performing well as compared to the other two as it has high precisions for both training and testing data and although the AUC Score is the same for all the three models for training data but for testing data it is the highest for Random Forest Model. Choosing Random Forest Model is the best option in this case as it will exhibit very less variance as compared to a single decision tree or a multi – layered Neural Network.

2.5 To provide business insights and recommendations.

For the business problem of an insurance firm providing Tour Insurance, we have attempted to make a few Data Models for predictions of probabilities. The models that are attempted are namely, CART or Classification and Regression Trees, Random Forest and Artificial Neural Network (MLP). The three models are then evaluated on training and testing datasets and their model performance scores are calculated. The Accuracy, Precision, F1 Score are computed using Classification Report. The confusion matrix, AUC_ROC Scores and ROC plot are computed for each model separately and compared. All the three models have performed well but to increase our accuracy in determining the claims made by the customers we can choose the Random Forest Model. Instead of creating a single Decision Tree it can create multiple decision trees and hence can provide the best claim status from the data. As seen from the above model performance measures, for all the models i.e. CART, Random Forest and ANN have performed exceptionally well. Hence, we can choose either of the models but choosing Random Forest Model is a great option as even though they exhibit the same accuracy but choosing Random Forest over Cart model is way better as they have much less variance than a single decision tree.