#Prepared By

Jitendra Kumar Nayak

Date:26/09/2021

#Business Report

Data Mining

Table of Contents

List of Figures

List of Tables

References

Problem 1

Executive Summary

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.

Introduction

The objective is to use unsupervised learning predictive models to give insights about data and the business problem. In specific, we will be using Agglomerative (for Hierarchical) clustering, and K-Means (for Non-Hierarchical) clustering.

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

Let us have a look at first few records of data:

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

Table 1: Sample of Data

(above table generated with .head() function of PANDAS library in Python)

Data Description

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)

Exploratory Data Analysis

Let us check the data types of variables and the missing values(if any):

```
<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
```

Table 2: General Information of Data(Problem1)

(above table generated by '.info' of 'PANDAS' in Python)

We can see that

- shape of data is (210x7)
- there are total 210 entries(rows) of individuals in data indexed from 0 to 210
- there are 7 columns in total(spending, advance_payments, probability_of_full_payment, current_balance, credit_limit, min_payment_amt, max_spent_in_single_shopping)
- there is no null value in data set as all columns have 210 non null objects
- all variables data-type float(64 bits)
- the memory usage by data is 11.6kb

(Heatmap and Pairplot provided after Univariate Analysis)

Let us now move to the general description of data(minimum values, maximum values, measures of central tendencies like mean, median, mode etc.):

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	14.85	2.91	10.59	12.27	14.36	17.30	21.18	10.59	5.04	0.20	11.23	14.11	15.38
advance_payments	14.56	1.31	12.41	13.45	14.32	15.72	17.25	4.84	2.27	0.09	13.47	NaN	NaN
probability_of_full_payment	0.87	0.02	0.81	0.86	0.87	0.89	0.92	0.11	0.03	0.03	0.88	NaN	NaN
current_balance	5.63	0.44	4.90	5.26	5.52	5.98	6.68	1.78	0.72	0.08	5.24	5.40	NaN
credit_limit	3.26	0.38	2.63	2.94	3.24	3.56	4.03	1.40	0.62	0.12	3.03	NaN	NaN
min_payment_amt	3.70	1.50	0.77	2.56	3.60	4.77	8.46	7.69	2.21	0.41	2.13	2.22	2.70
max_spent_in_single_shopping	5.41	0.49	4.52	5.04	5.22	5.88	6.55	2.03	0.83	0.09	5.00	NaN	NaN

Table 3: General Description of Data(Problem 1)

(above table generated with .describe() function of PANDAS library in Python)

From above table, we conclude:

- values in all the features seem to be correct
- the mean values are in different orders, indicating features have values in different ranges, which is also being confirmed by 'range'
- the coefficient of variation is quite high for 'min_payment_amt' (0.41)
- few features in dataset have multiple modal values

From '.info()', we already saw there are no null values in the data set. '.isnull().sum().sum()' also confirms the same yielding 0 as output, i.e., there are 0 null entries in the data set.

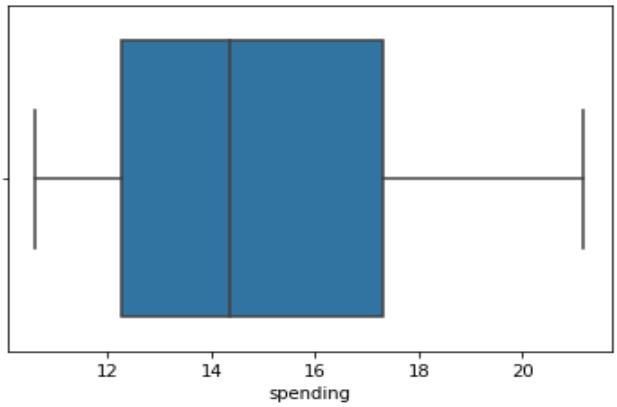
There are no duplicate entries in the data set which has been confirmed by the '.duplicated()' function of PANDAS library in Python.

Let us now go for the Univariate analysis of all features in the data set. As all features are of data type float, we will go for general descriptions, distribution plots with kernel density estimates and boxplots for all the features one by one:

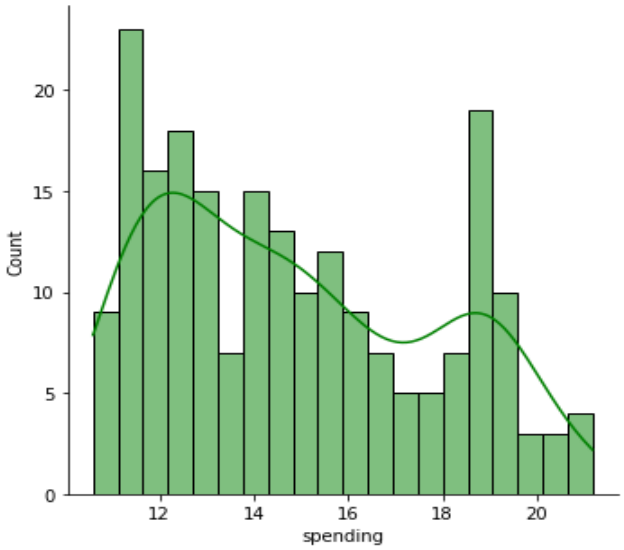
(zoom to at least 125% for better visibility)

```
Description of spending
-------------------------------
count    210.000000
mean      14.847524
std        2.909699
min       10.590000
25%       12.270000
50%       14.355000
75%       17.305000
max       21.180000
Name: spending, dtype: float64
```

BoxPlot of spending

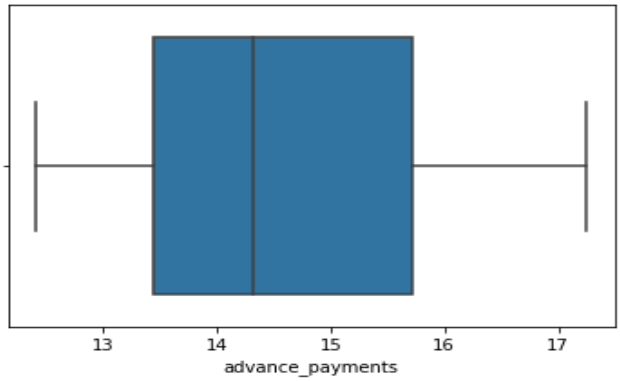


Distribution of spending

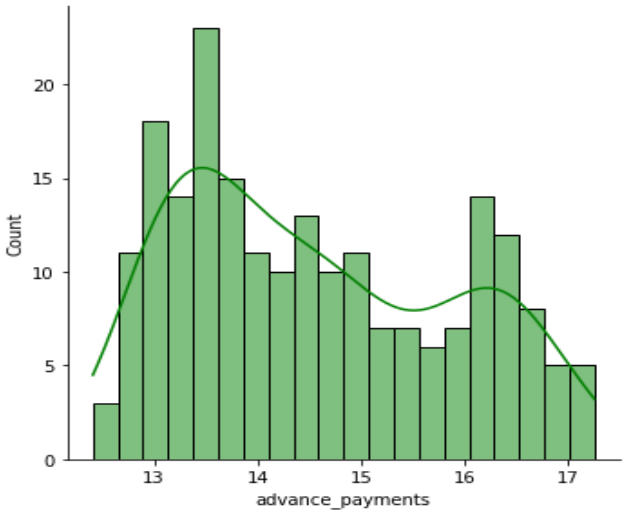



```
Description of advance_payments
--------------------------------
count    210.000000
mean      14.559286
std        1.305959
min       12.410000
25%       13.450000
50%       14.320000
75%       15.715000
max       17.250000
Name: advance_payments, dtype: float64
```

BoxPlot of advance_payments

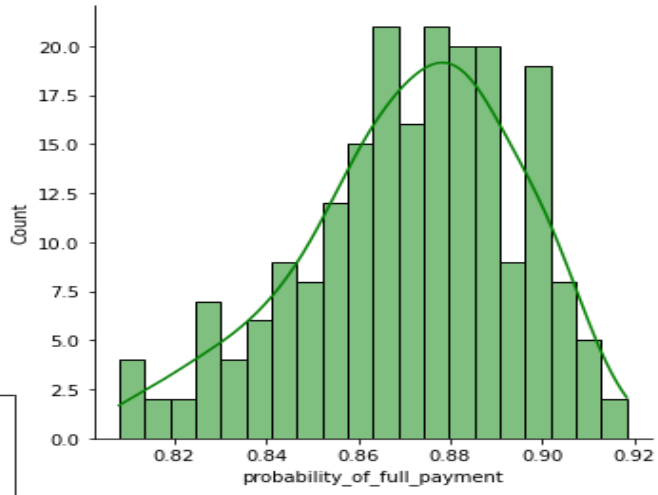


Distribution of advance_payments

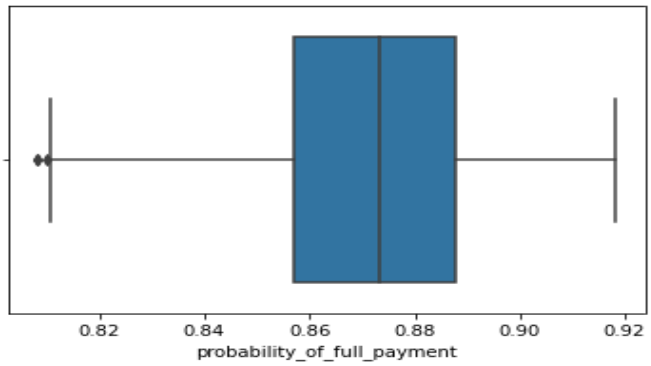


```
Description of
--------------
probability_of_full_payment
---------------------------
count    210.000000
mean       0.870999
std        0.023629
min        0.808100
25%        0.856900
50%        0.873450
75%        0.887775
max        0.918300
Name: probability_of_full_payment,
dtype: float64
```

Distribution of probability_of_full_payment

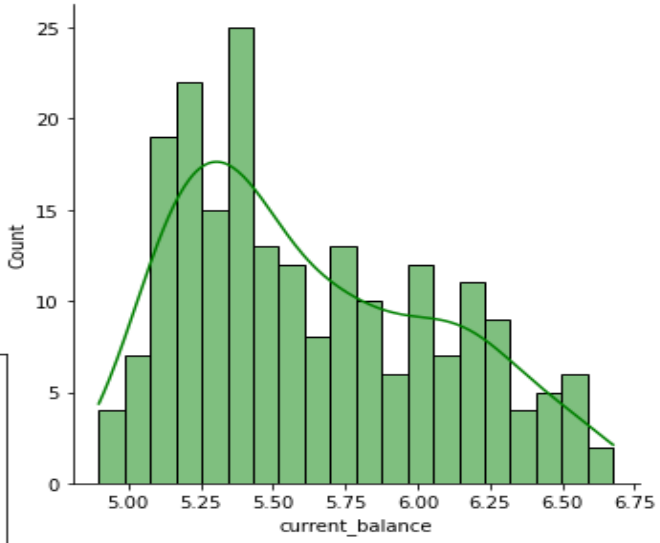


BoxPlot of probability_of_full_payment

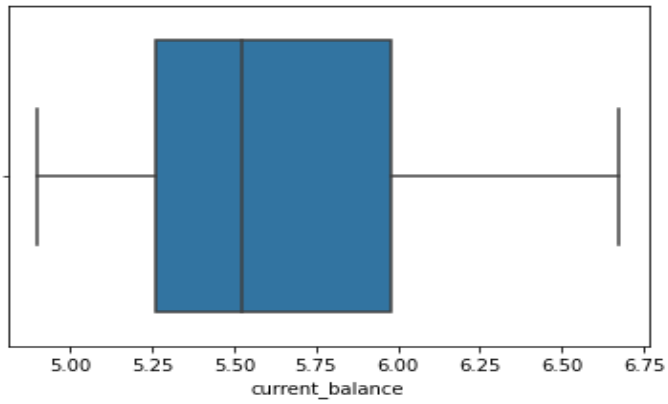



```
Description of current_balance
------------------------------
count    210.000000
mean       5.628533
std        0.443063
min        4.899000
25%        5.262250
50%        5.523500
75%        5.979750
max        6.675000
Name: current_balance, dtype: float64
```

Distribution of current_balance

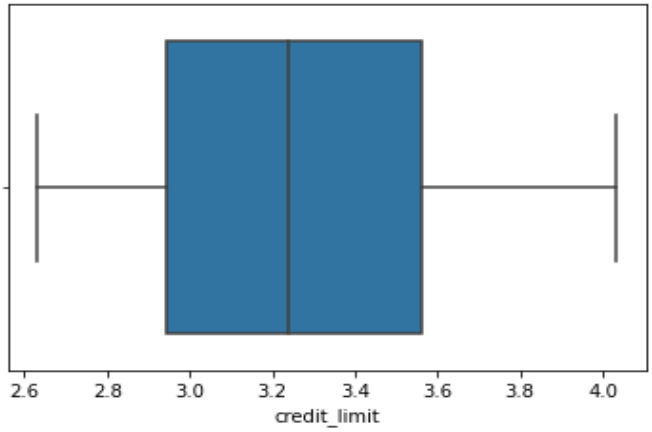


BoxPlot of current_balance

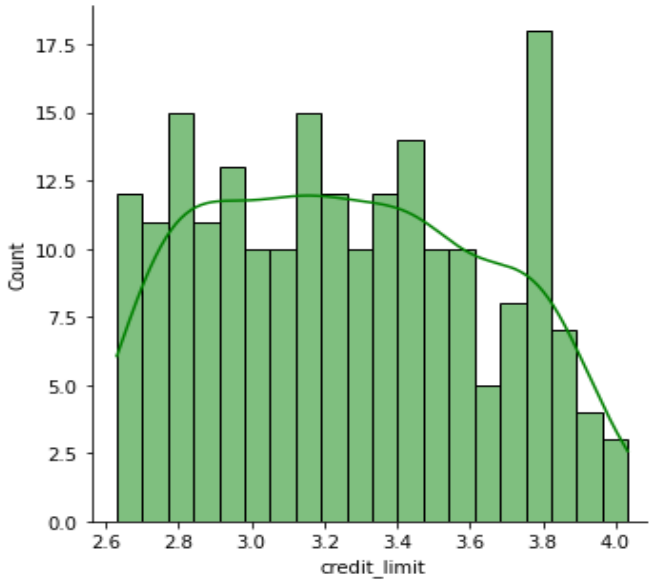


```
Description of credit_limit
---------------------------------
count    210.000000
mean       3.258605
std        0.377714
min        2.630000
25%        2.944000
50%        3.237000
75%        3.561750
max        4.033000
Name: credit_limit, dtype: float64
```

BoxPlot of credit_limit

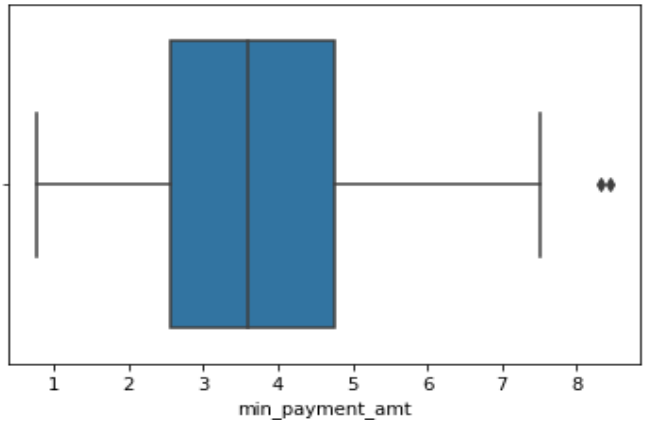


Distribution of credit_limit

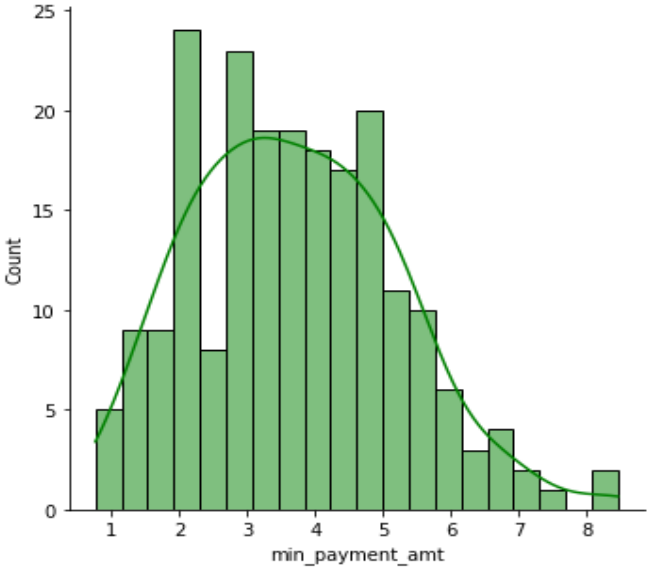



```
Description of min_payment_amt
---------------------------------
count    210.000000
mean       3.700201
std        1.503557
min        0.765100
25%        2.561500
50%        3.599000
75%        4.768750
max        8.456000
Name: min_payment_amt, dtype: float64
```

BoxPlot of min_payment_amt



Distribution of min_payment_amt

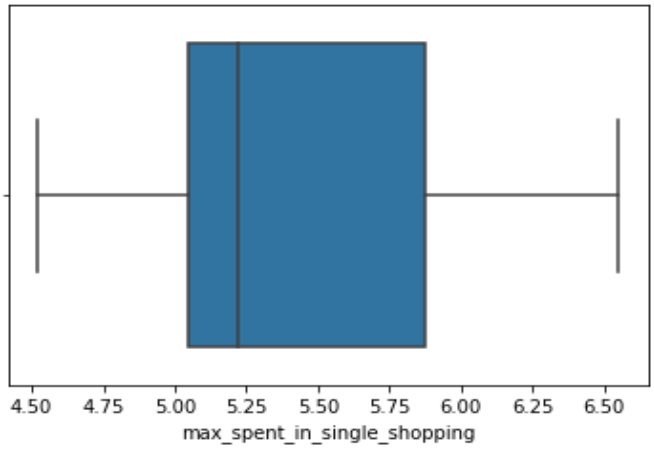


Description of
---
max_spent_in_single_shopping

count	210.000000
mean	5.408071
std	0.491480
min	4.519000
25%	5.045000
50%	5.223000
75%	5.877000
max	6.550000

Name: max spent_in_single_shopping,
dtype: float64

BoxPlot of max_spent_in_single_shopping



Distribution of max_spent_in_single_shopping

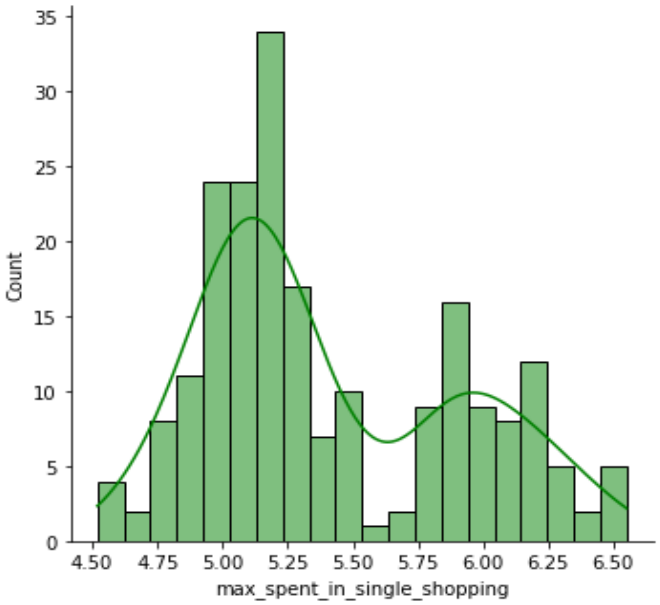


Figure 1: Univariate Analysis (Problem 1)

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

We can see:

- the general description of all features
- the variables are in different orders or ranges which is evident from 'min' and 'max' values
- none of the features are nearly-normal, all have some amount of skewness
- except for 'probability_of_full_payment' which is left skewed, all other features are right skewed
- except for 'probability_of_full_payment' and 'min_payment_amt', all other features have no outliers in them

Now for Bivariate and Multivariate analysis let us have a look at scatter plots in the pairplot and heatmap of correlations of features of the data:
(zoom to at least 150% for better visibility)

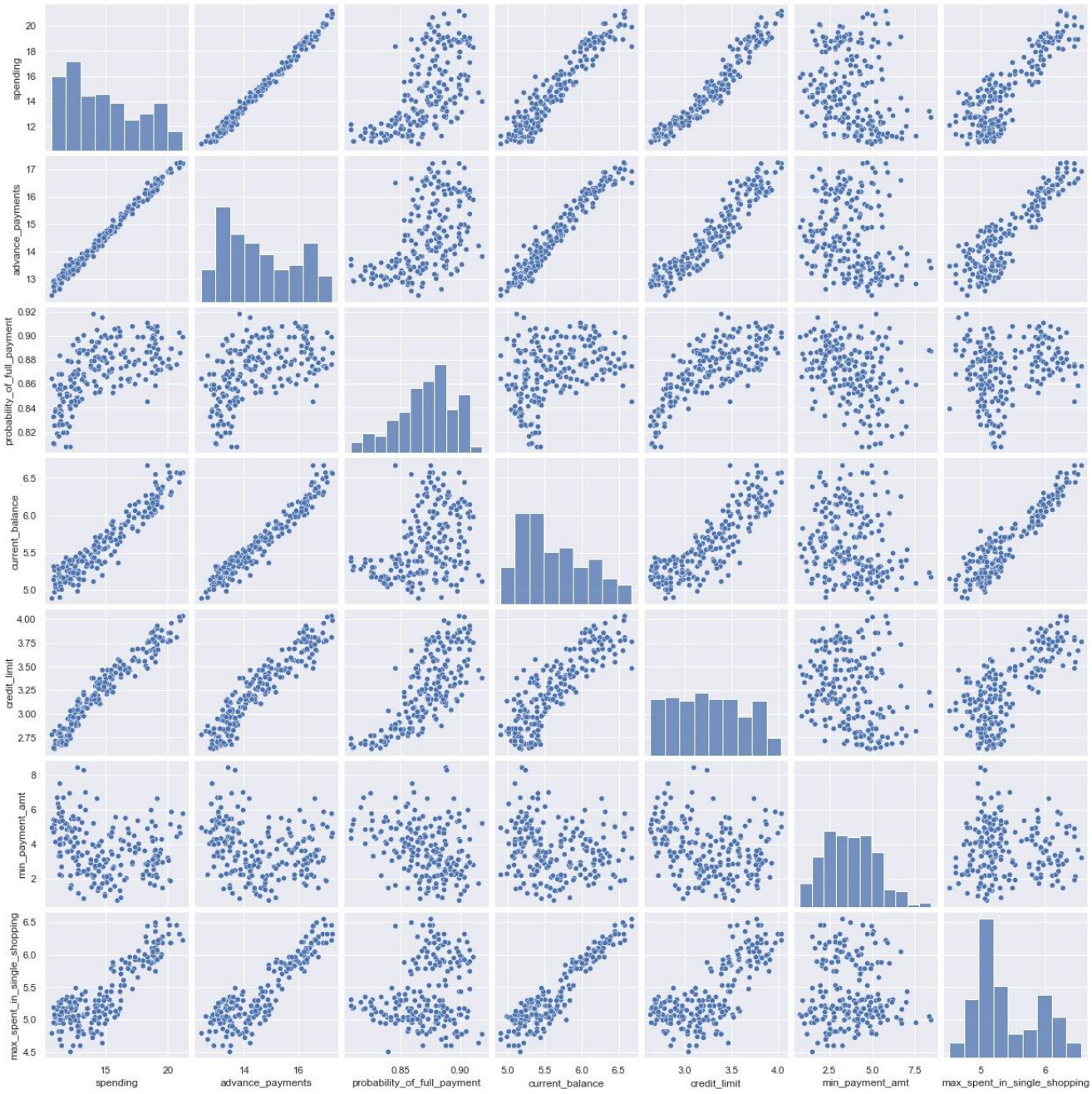


Figure 2: Pairplot(Problem 1)

(above figure generated with '.pairplot()' funtion of Seaborn library in Python)

Pairplot shows the relationship between the numeric variables in the form of scatterplot and the distribution of the variables in the form of histogram.

The pattern of clouds of points convey that there are positive correlations mostly. The cloud pattern of feature 'min_payment_amt' is suggesting negative correlation.

To verify our observations from pairplot, let us have a look at the Pearson's Correlation coefficients in the form of a heatmap:

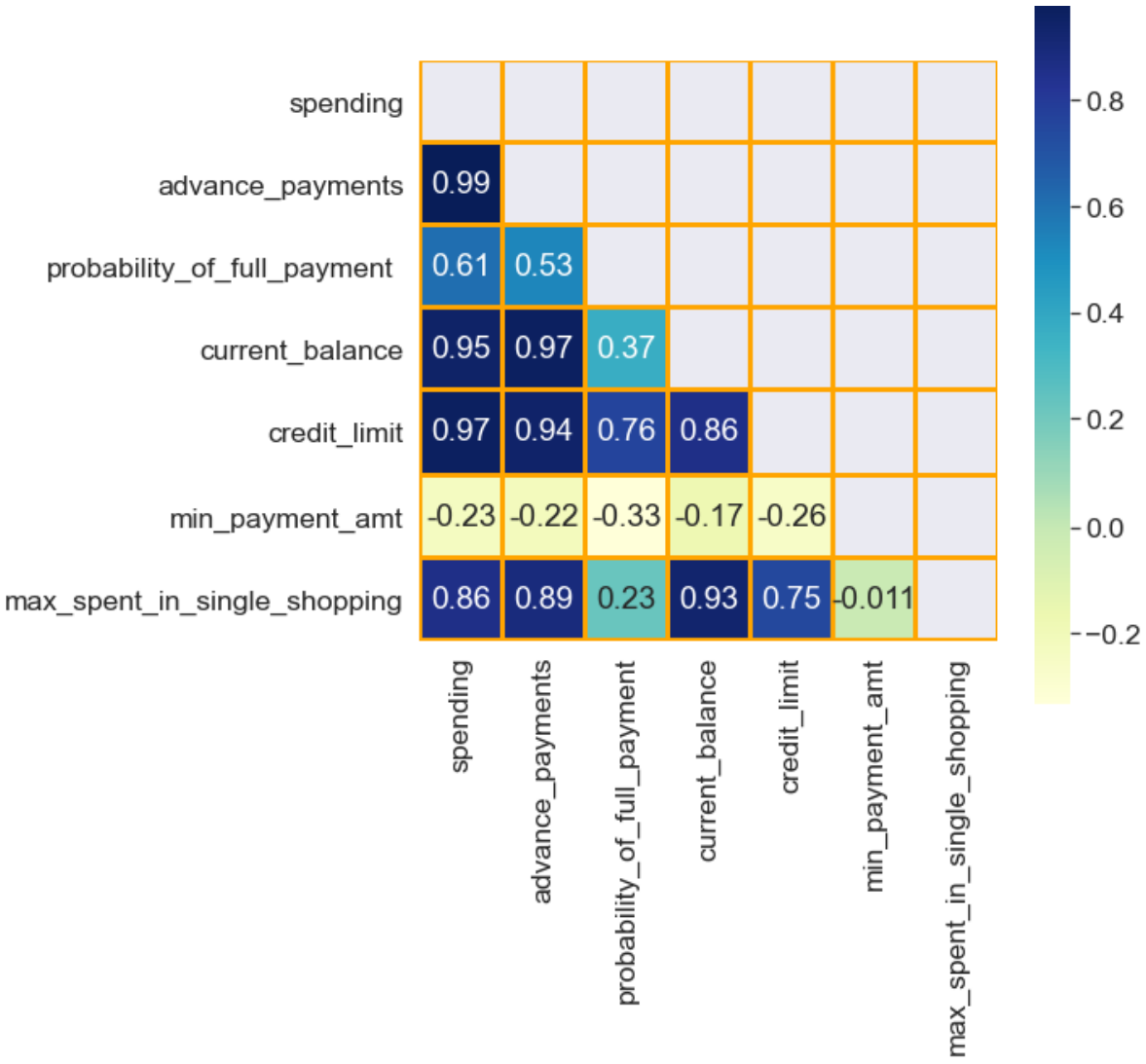


Figure 3: Correlation Heatmap (Problem 1)

(plot generated with 'heatmap()' function of 'Seaborn' library using Python)

- Pearson's Correlation coefficients near to 1 or -1 are highly positively correlated and highly negatively correlated respectively. Correlation values near to 0 are not or minimally correlated to each other.
- We see that most of Pearson's coefficients are positive or in the vicinity of 1. The only negative correlations in data are of feature 'min_payment_amt'.

Skewness

- Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean.
- For normally distributed data, the skewness should be about zero. For unimodal continuous distributions, a skewness value greater than zero means that there is more weight in the right tail of the distribution.

We did infer couple of points about skewness during univariate analysis, let us have a better insight now:

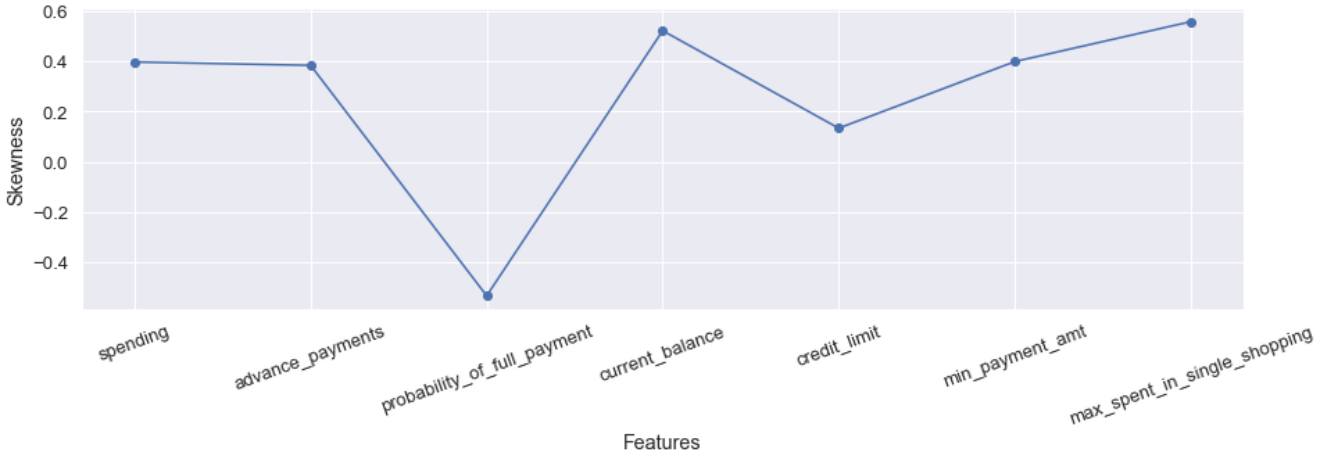


Figure 4: Skewness of features(Problem 1)

(above plot generated with '.skew()' function of scipy.stats, '.plot()' of matplotlib and PANDAS libraries in Python)

We can see:

- all features are positively skewed except for 'probability_of_full_payment'
- current_balance has high positive skewness amongst all features
- for positive or right skewness:MODE<MEDIAN<MEAN
- for negative or left skewness:MODE>MEDIAN>MEAN

Now that we are done with the univariate and multivariate analysis, let us impute the outliers in data with suitable values.

- During the univariate analysis, we saw the features 'probability_of_full_payment' and 'min_payment_amt' have outliers in them.
- We can go for the Interquartile Range(IQR) method of imputation of outliers.

Interquartile Range (IQR) = Q3 - Q1

where Q1 and Q3 are first and third quartiles respectively

- Anything point beyond 1.5 times the IQR distance on either side of the IQR is considered as an outlier.

- The data points beyond 1.5 times IQR distance to the left of first quartile(Q1) are capped with a value, let us call it lower fence, which is defined as:

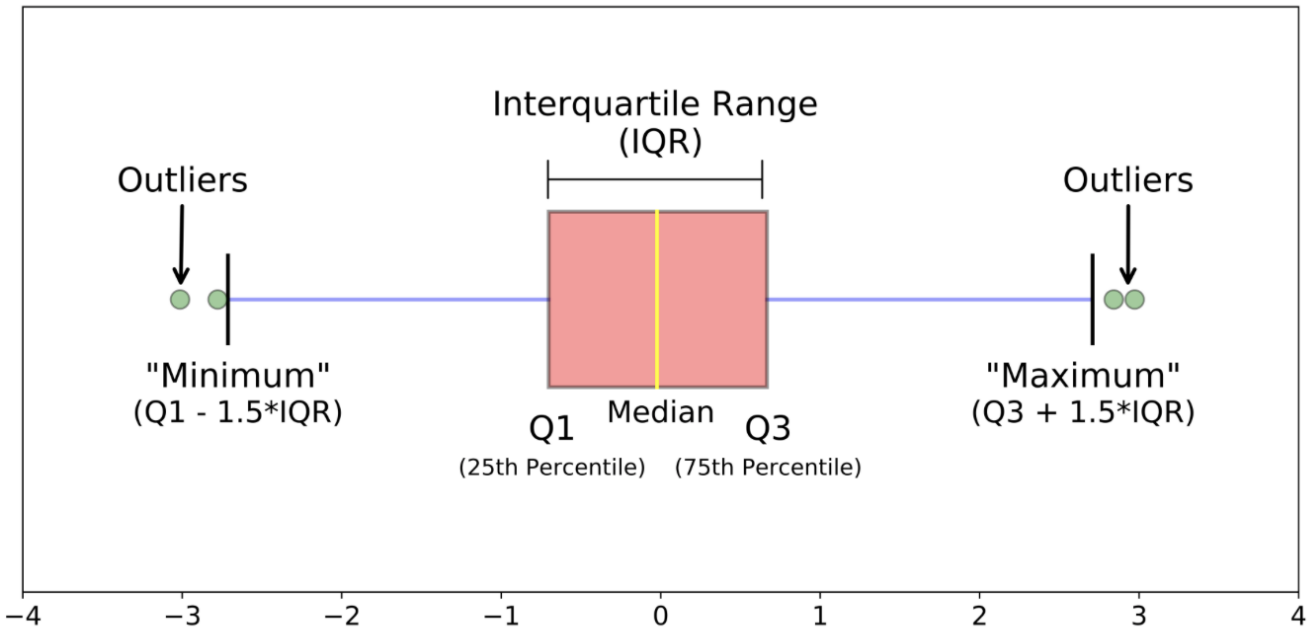
Lower Fence = Q1 - 1.5*IQR

where Q1 and IQR are first quartile and Interquartile Range respectively.

- The data points beyond 1.5 times IQR distance to the right of third quartile(Q3) are capped with a value, let us call it upper fence, which is defined as:

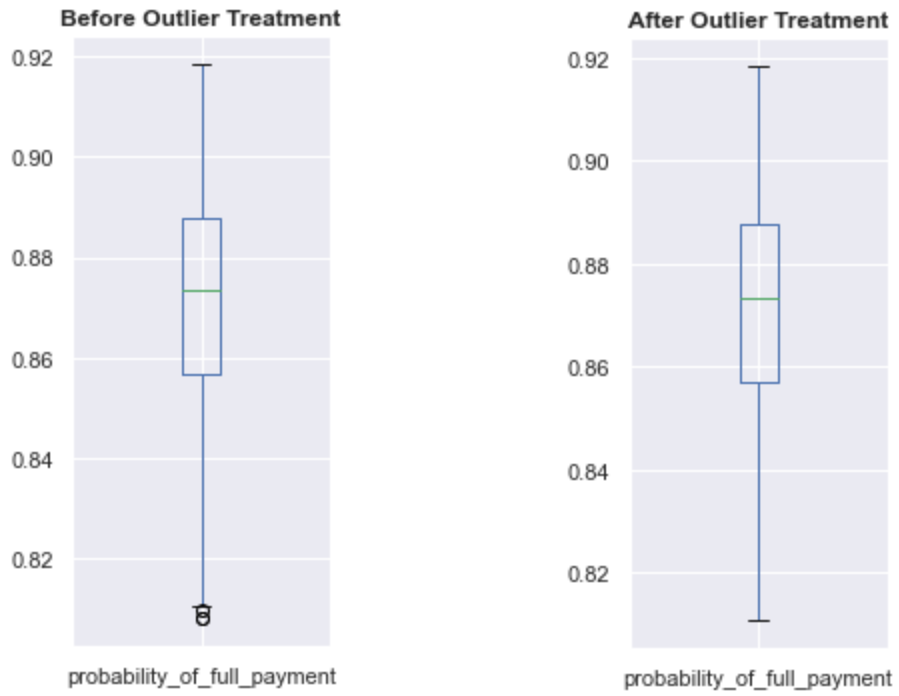
Upper Fence = Q3 + 1.5*IQR

where Q3 and IQR are first quartile and Interquartile Range respectively.



Let us view the box plots of data features before and after outlier treatment:

probability _of_full_payment:



min_payment_amt:

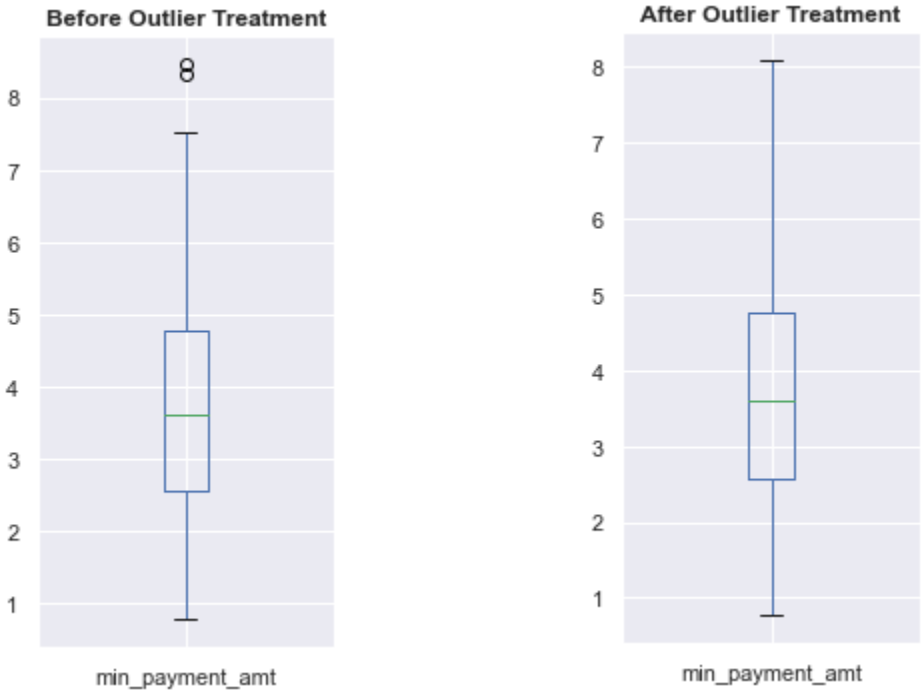


Figure 5: Outlier Treatment(Problem 1)

(above plot generated Seaborn, matplotlib and PANDAS libraries in Python)

Now that the outliers have been taken care of, we are good to proceed further.

Though we see some very high correlations amongst few features in the dataset, multicollinearity generally doesn't impact the clustering process. Also, there are only 7 features in the dataset, dropping any one of them would not be a vey great idea.

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

Let us have a look at the first and last 5 records of data set after outliers treatment:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
0	19.94	16.92	0.875200	6.675	3.763	3.252	6.550
1	15.99	14.89	0.906400	5.363	3.582	3.336	5.144
2	18.95	16.42	0.882900	6.248	3.755	3.368	6.148
3	10.83	12.96	0.810588	5.278	2.641	5.182	5.185
4	17.99	15.86	0.899200	5.890	3.694	2.068	5.837
...	...	...	...	...	...	...	...
205	13.89	14.02	0.888000	5.439	3.199	3.986	4.738
206	16.77	15.62	0.863800	5.927	3.438	4.920	5.795
207	14.03	14.16	0.879600	5.438	3.201	1.717	5.001
208	16.12	15.00	0.900000	5.709	3.485	2.270	5.443
209	15.57	15.15	0.852700	5.920	3.231	2.640	5.879

Table 4: Sample of Data after Outliers treatment(Problem 1)

(table generated using PANDAS library of Python)

Let us check the general description of data after outlier treatment:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	14.85	2.91	10.59	12.27	14.36	17.30	21.18	10.59	5.04	0.20	11.23	14.11	15.38
advance_payments	14.56	1.31	12.41	13.45	14.32	15.72	17.25	4.84	2.27	0.09	13.47	NaN	NaN
probability_of_full_payment	0.87	0.02	0.81	0.86	0.87	0.89	0.92	0.11	0.03	0.03	0.81	0.88	NaN
current_balance	5.63	0.44	4.90	5.26	5.52	5.98	6.68	1.78	0.72	0.08	5.24	5.40	NaN
credit_limit	3.26	0.38	2.63	2.94	3.24	3.56	4.03	1.40	0.62	0.12	3.03	NaN	NaN
min_payment_amt	3.70	1.49	0.77	2.56	3.60	4.77	8.08	7.31	2.21	0.40	2.13	2.22	2.70
max_spent_in_single_shopping	5.41	0.49	4.52	5.04	5.22	5.88	6.55	2.03	0.83	0.09	5.00	NaN	NaN

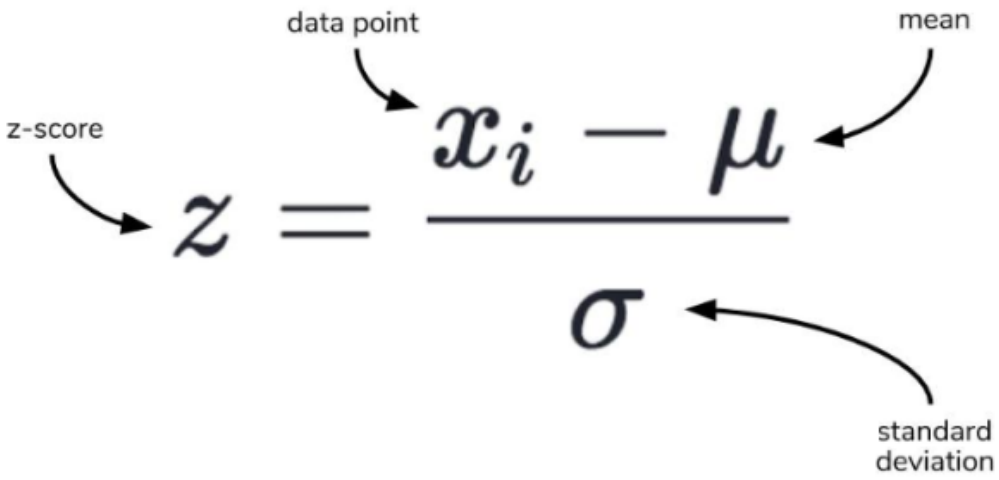
Table 5: General Description of Data(Problem 1)

(above table generated with .describe() function of PANDAS library in Python)

- We can see that all the columns have values in different scales, which is evident from data description.
- Scaling is really important to make analysis easier.
- We can see standard deviations are different for different features.
- Coefficients of Variation are also different for different features.

We can scale the data using the 'zscore' function of the 'scipy.stats' library, which converts data into z-scores:

- the z-score or standard score is the number of standard deviations by which the value of a raw score is above or below the mean value of what is being observed or measured.
- scipy.stats.zscore() function computes the relative Z-score of the input data, relative to the sample mean and standard deviation using the formula:



Let us have a look at a sample of data after scaling:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
0	1.754355	1.811968	0.177628	2.367533	1.338579	-0.298625	2.328998
1	0.393582	0.253840	1.505071	-0.600744	0.858236	-0.242292	-0.538582
2	1.413300	1.428192	0.505234	1.401485	1.317348	-0.220832	1.509107
3	-1.384034	-1.227533	-2.571391	-0.793049	-1.639017	0.995699	-0.454961
4	1.082581	0.998364	1.198738	0.591544	1.155464	-1.092656	0.874813
...	...	...	...	...	...	...	...
205	-0.329866	-0.413929	0.722220	-0.428801	-0.158181	0.193620	-1.366631
206	0.662292	0.814152	-0.307399	0.675253	0.476084	0.819993	0.789153
207	-0.281636	-0.306472	0.364831	-0.431064	-0.152873	-1.328049	-0.830235
208	0.438367	0.338271	1.232775	0.182048	0.600814	-0.957188	0.071238
209	0.248893	0.453403	-0.779662	0.659416	-0.073258	-0.709053	0.960473

Table 6: Sample of Scaled Data(Problem 1)

(table generated using PANDAS and scipy.stats libraries of Python)

Let us also check the general description of the scaled data:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	0.0	1.0	-1.47	-0.89	-0.17	0.85	2.18	3.65	1.73	1.095655e+15	-1.25	-0.25	0.18
advance_payments	0.0	1.0	-1.65	-0.85	-0.18	0.89	2.07	3.71	1.74	9.137502e+15	-0.84	NaN	NaN
probability_of_full_payment	0.0	1.0	-2.57	-0.60	0.10	0.71	2.01	4.58	1.31	6.102451e+14	-2.57	0.48	NaN
current_balance	-0.0	1.0	-1.65	-0.83	-0.24	0.79	2.37	4.02	1.62	-9.204037e+15	-0.89	-0.53	NaN
credit_limit	-0.0	1.0	-1.67	-0.83	-0.06	0.80	2.06	3.72	1.64	-3.347661e+15	-0.62	NaN	NaN
min_payment_amt	0.0	1.0	-1.97	-0.76	-0.07	0.72	2.94	4.91	1.48	6.629481e+15	-1.05	-0.99	-0.67
max_spent_in_single_shopping	-0.0	1.0	-1.81	-0.74	-0.38	0.96	2.33	4.14	1.70	-5.178999e+14	-0.83	NaN	NaN

Table 7: General Description of Scaled Data(Problem 1)

(above table generated with .describe() function of PANDAS library in Python)

- We now see the numeric data to be in all similar ranges.
- All features have mean values as 0 and standard deviation as 1.
- Coefficients of variation are also in similar orders now

The data is now ready for further analysis.

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

- From the 'scipy.cluster.hierarchy' library, we will be using the functions 'fclusters' and 'linkage' to form hierarchical agglomerative clusters. Then, later we can use the function 'dendrogram' from the same library to form dendrograms.
- We will be using Euclidean affinity in all the cases. Euclidean distance amongst n points with coordinates in x-y plane is given by:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

Let us see the dendrogram for ward linked clusters till 10 merges(Euclidean affinity has been used):

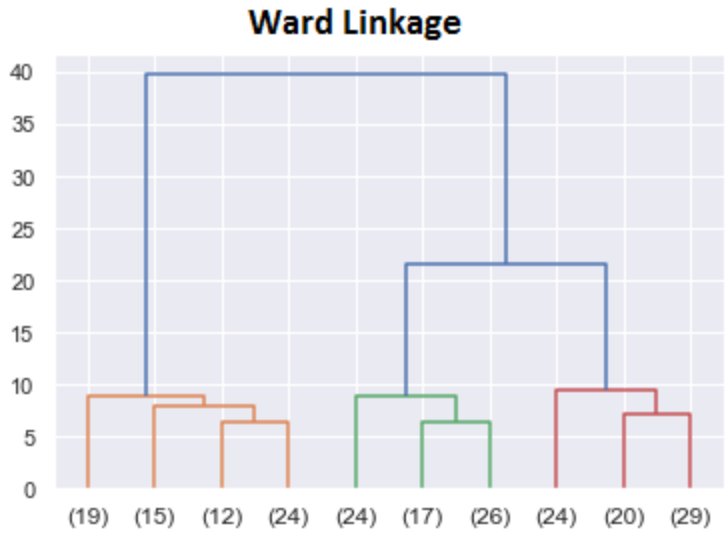
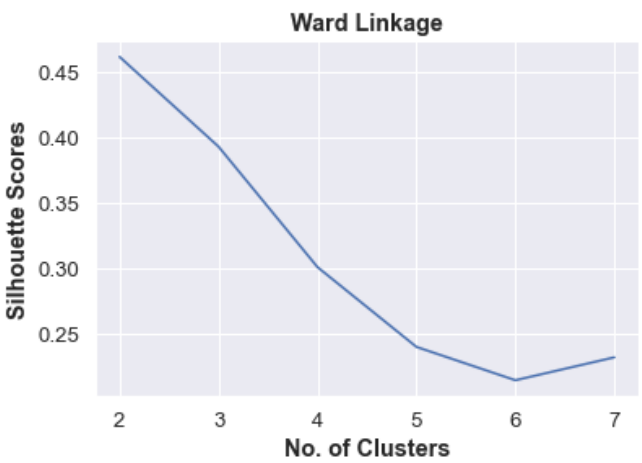


Figure 6: Dendrogram(Ward Linkage)

(generated using 'dendrogram' and 'linkage' functions of 'scipy.cluster.hierarchy' library in Python)

We can clearly see 3 would be the optimum number of clusters and based on distance criterion, cutting the dendrogram at a distance 10 on the y-axis would be the optimal solution. But, let us confirm our result, and also explore a few other options available to us, as the silhouette score is coming out to be only 0.393 for 3 clusters for Ward linkage. For well demarcated clusters silhouette score of at least 0.5 is the general acceptable standard.

There are multiple linkage options available to form clusters. Let us try to visualise the silhouette scores for different linkage methods for 2 to 7 clusters for each linkage options(Euclidean affinity has been used in all the cases):



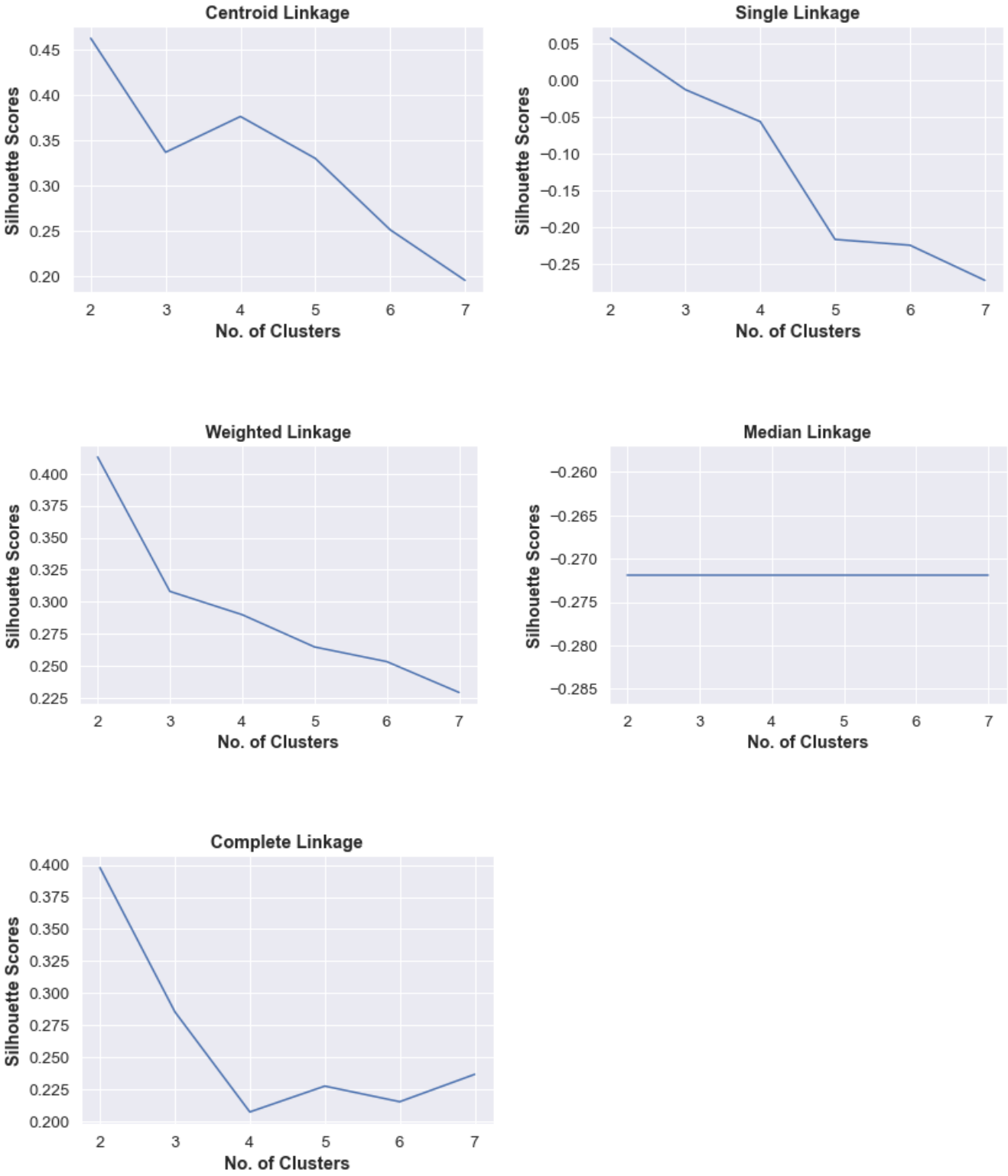


Figure 7 :Silhouette Scores for different linkages

(plots generated using matplotlib, Seaborn, PANDAS and scipy libraries in Python)

- none of the clusters are well demarcated as none have a silhouette score of 0.5 for any given number of clusters
- 2 clusters really do not make much of a business impact as it is kind of implicit
- we see silhouette scores for 3 clusters are highest in all graphs after 2 clusters, for different linkages.
- silhouette scores for ward linkage is the highest amongst 3 clusters, for different linkages.

Let us inspect the dendrograms for median and ward linkages till 10 merges:

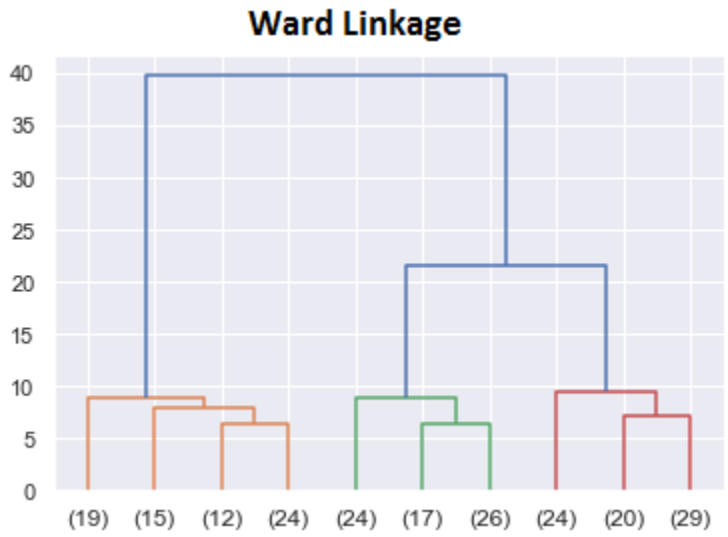


Figure 8: Dendrogram(Ward Linkage)

(generated using 'dendrogram' and 'linkage' functions of 'scipy.cluster.hierarchy' library in Python)

- There are 70, 67 and 73 observations in the 3 ward linked clusters(we get these by adding the number of samples in the leaf nodes of 10 merges in the dendrogram, shown on x-axis).
- Cutting the ward linked dendrogram at distance 10 on the y-axis might yield optimum results.

Now, we can go ahead and attach the ward linkage cluster labels to the original data.

Let us visualise the clusters using all possible combinations of 2-D scatter plots using ward linkage cluster labels as hue in a pairplot:

(zoom to at least 125% for better visibility)

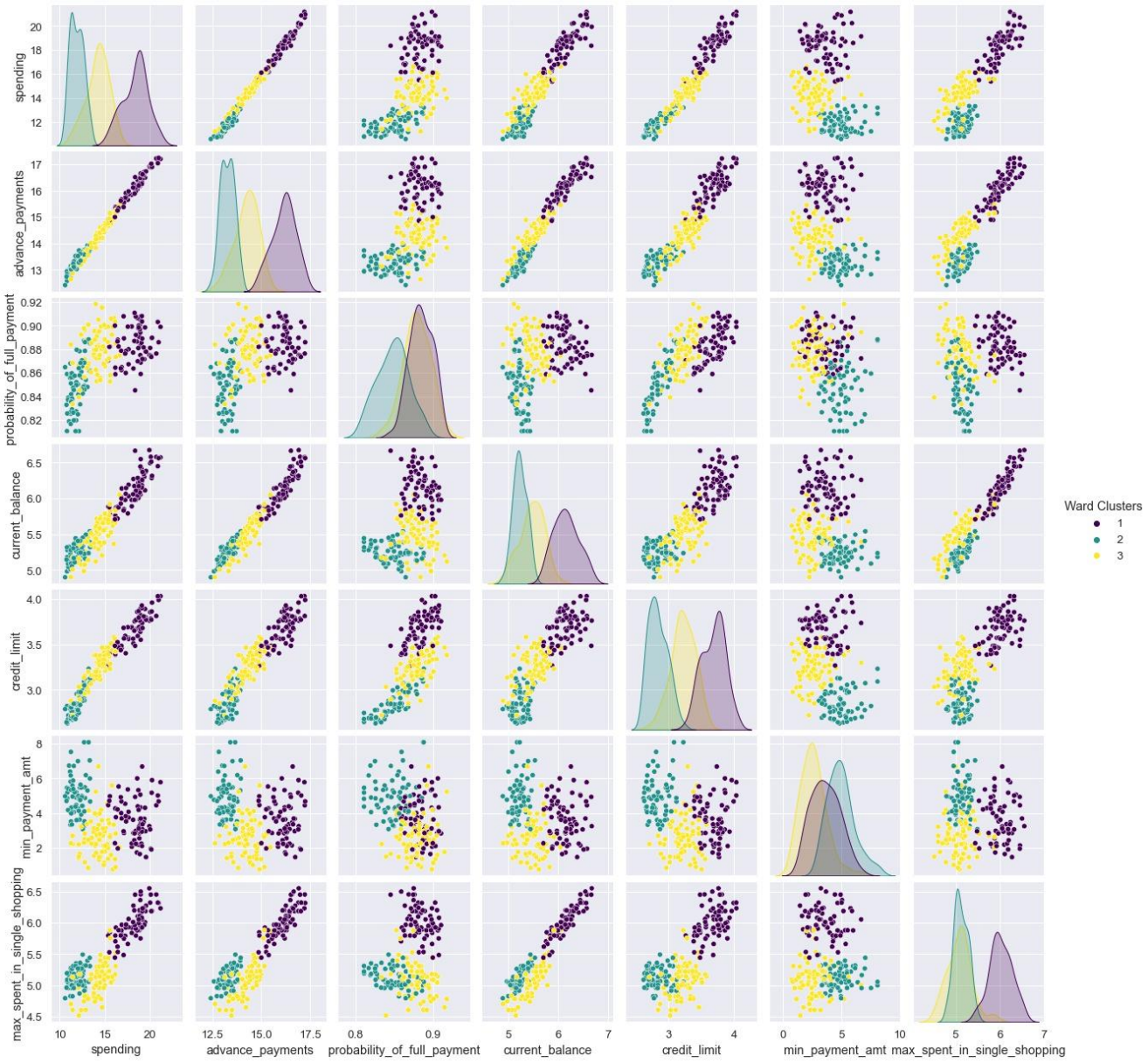


Figure 9 : Pairplot(Ward Linkage)

(generated using 'dendrogram' and 'linkage' functions of 'scipy.cluster.hierarchy' library in Python)

As the silhouette score is only 0.393, the clusters are not very well demarcated, which is also evident from the above pairplot.

Let us have a look at general descriptions for 3 clusters:

Cluster 1:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	18.37	1.38	15.38	17.33	18.72	19.14	21.18	5.80	1.81	0.08	11.23	14.11	15.38
advance_payments	16.15	0.60	14.86	15.74	16.21	16.56	17.25	2.39	0.82	0.04	13.47	NaN	NaN
probability_of_full_payment	0.88	0.01	0.85	0.87	0.88	0.90	0.91	0.07	0.02	0.02	0.81	0.88	NaN
current_balance	6.16	0.25	5.71	5.98	6.15	6.31	6.68	0.97	0.33	0.04	5.24	5.40	NaN
credit_limit	3.68	0.17	3.27	3.55	3.69	3.80	4.03	0.77	0.25	0.05	3.03	NaN	NaN
min_payment_amt	3.64	1.21	1.47	2.85	3.63	4.46	6.68	5.21	1.61	0.33	2.13	2.22	2.70
max_spent_in_single_shopping	6.02	0.25	5.44	5.88	5.98	6.19	6.55	1.11	0.31	0.04	5.00	NaN	NaN
Ward Clusters	1.00	0.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	3.00	NaN	NaN

Cluster 2:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	11.87	0.74	10.59	11.25	11.83	12.45	13.37	2.78	1.20	0.06	11.23	14.11	15.38
advance_payments	13.26	0.35	12.41	13.00	13.27	13.52	13.95	1.54	0.52	0.03	13.47	NaN	NaN
probability_of_full_payment	0.85	0.02	0.81	0.83	0.85	0.86	0.89	0.08	0.03	0.02	0.81	0.88	NaN
current_balance	5.24	0.14	4.90	5.14	5.24	5.33	5.54	0.64	0.19	0.03	5.24	5.40	NaN
credit_limit	2.85	0.14	2.63	2.73	2.83	2.97	3.23	0.60	0.24	0.05	3.03	NaN	NaN
min_payment_amt	4.94	1.14	3.08	4.12	4.86	5.47	8.08	5.00	1.35	0.23	2.13	2.22	2.70
max_spent_in_single_shopping	5.12	0.16	4.79	5.00	5.09	5.25	5.49	0.70	0.25	0.03	5.00	NaN	NaN
Ward Clusters	2.00	0.00	2.00	2.00	2.00	2.00	2.00	0.00	0.00	0.00	3.00	NaN	NaN

Cluster 3:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	14.20	1.23	11.23	13.50	14.33	15.03	16.63	5.40	1.53	0.09	11.23	14.11	15.38
advance_payments	14.23	0.60	12.63	13.85	14.28	14.67	15.46	2.83	0.82	0.04	13.47	NaN	NaN
probability_of_full_payment	0.88	0.02	0.83	0.87	0.88	0.89	0.92	0.08	0.02	0.02	0.81	0.88	NaN
current_balance	5.48	0.24	4.90	5.35	5.50	5.66	6.05	1.15	0.31	0.04	5.24	5.40	NaN
credit_limit	3.23	0.18	2.72	3.13	3.22	3.37	3.58	0.86	0.24	0.06	3.03	NaN	NaN
min_payment_amt	2.61	1.12	0.77	1.79	2.50	3.14	6.68	5.92	1.35	0.43	2.13	2.22	2.70
max_spent_in_single_shopping	5.09	0.28	4.52	4.87	5.10	5.22	5.88	1.36	0.35	0.05	5.00	NaN	NaN
Ward Clusters	3.00	0.00	3.00	3.00	3.00	3.00	3.00	0.00	0.00	0.00	3.00	NaN	NaN

Table 8: General Description of Ward Linked Clusters

(above table generated with .describe() function of PANDAS library in Python)

Let us look at the mean values for different features of various clusters in a pivot table:

Ward Clusters	advance_payments	credit_limit	current_balance	max_spent_in_single_shopping	min_payment_amt	probability_of_full_payment	spending
1	16.15	3.68	6.16	6.02	3.64	0.88	18.37
2	13.26	2.85	5.24	5.12	4.95	0.85	11.87
3	14.23	3.23	5.48	5.09	2.61	0.88	14.20

Table 9: Means of features for Ward Linked Clusters

(above table generated with .pivot_table() function of PANDAS library in Python)

- Looking at the feature 'spending' which is the amount spent by the customers per month (in 1000s), we can say cluster 2 customers are the least spending group, cluster 3 customers are the mediocre spending group and cluster 1 customers are the highest spending group.
- Looking at the feature 'min_payment_amt' which is the minimum amount paid by the customer while making payments for purchases made monthly (in 100s), we see cluster 2 customers make the highest payments though they are the least spending group. This may be because they make planned bulk purchases in order to avoid exhausting their credit limit on multiple transactions.
- Looking at the feature 'advance_payments' which is the amount paid by the customer in advance by cash (in 100s), we see cluster 2 which has the least spending customers, make the least advance payments, cluster 1 which has the highest spending customers, make the highest advance payments. This is also evident as the features 'spending' and 'advance_payments' have Pearson's correlation coefficient as 0.99.

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

From 'sklearn.cluster' we will be using the 'KMeans' function to form clusters.

The sums of squares of distance from respective centroids divided by the number of observations gives the variance of a cluster. We can check the within sums of squares variance for different clusters, which is also called inertia and then decide the optimum number of clusters.

WSS plot(Within Sums of Squares Plot) or Elbow plot will be used for the same. On y-axis we have the inertias for different number of clusters, which are on x-axis:

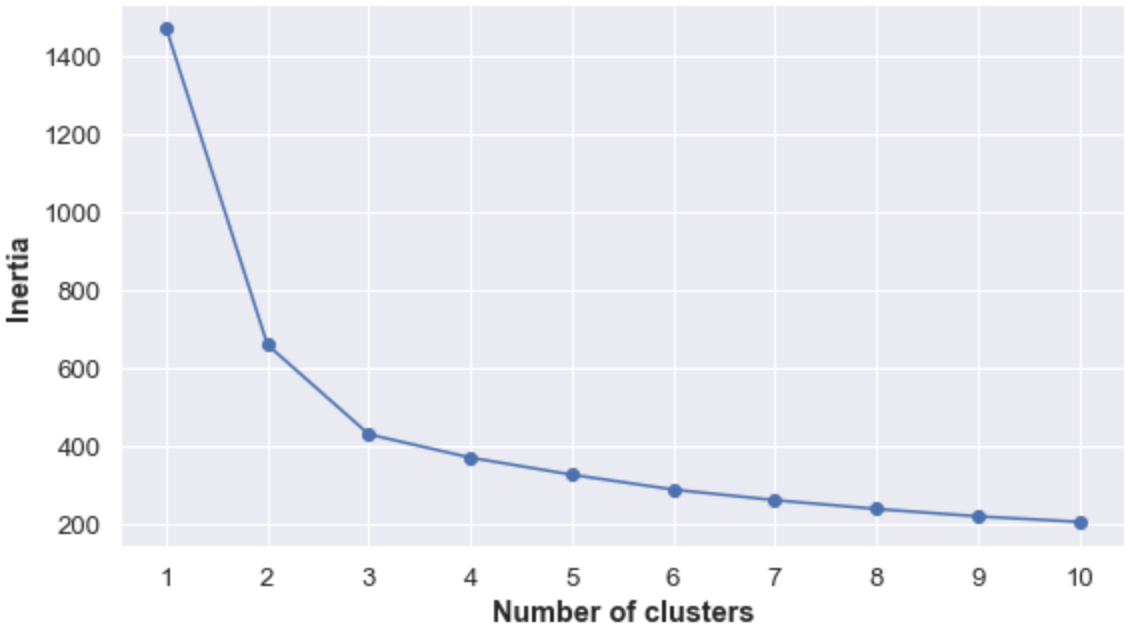


Figure 10 : WSS Plot(K-Means)

(generated using matplotlib, Seaborn and PANDAS libraries in Python)

2 clusters do not really make any business sense. Apart from that, the drop in inertia is maximum till 3 clusters, after 3 clusters is not very significant. So, we can choose the 'elbow' or 'knee' as 3 on the x-axis, i.e, 3 would be the optimum number of clusters for our business problem.

Let us also confirm above result by having a look at silhouette scores for different number of clusters(2-10):

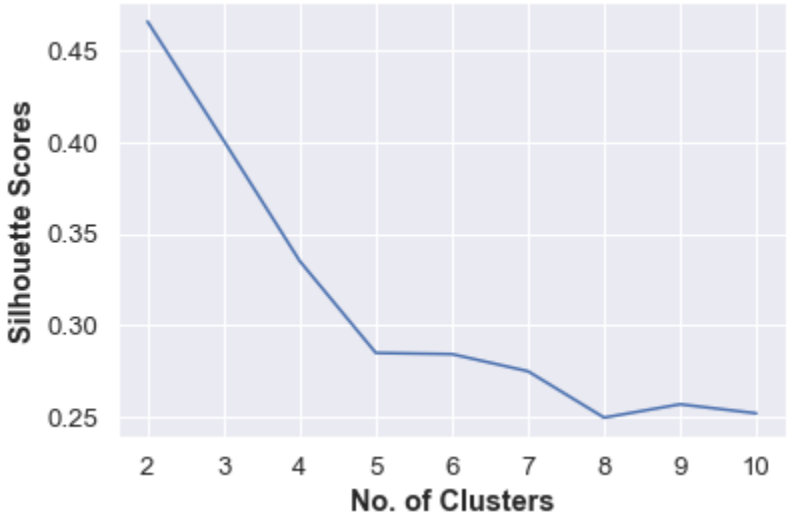


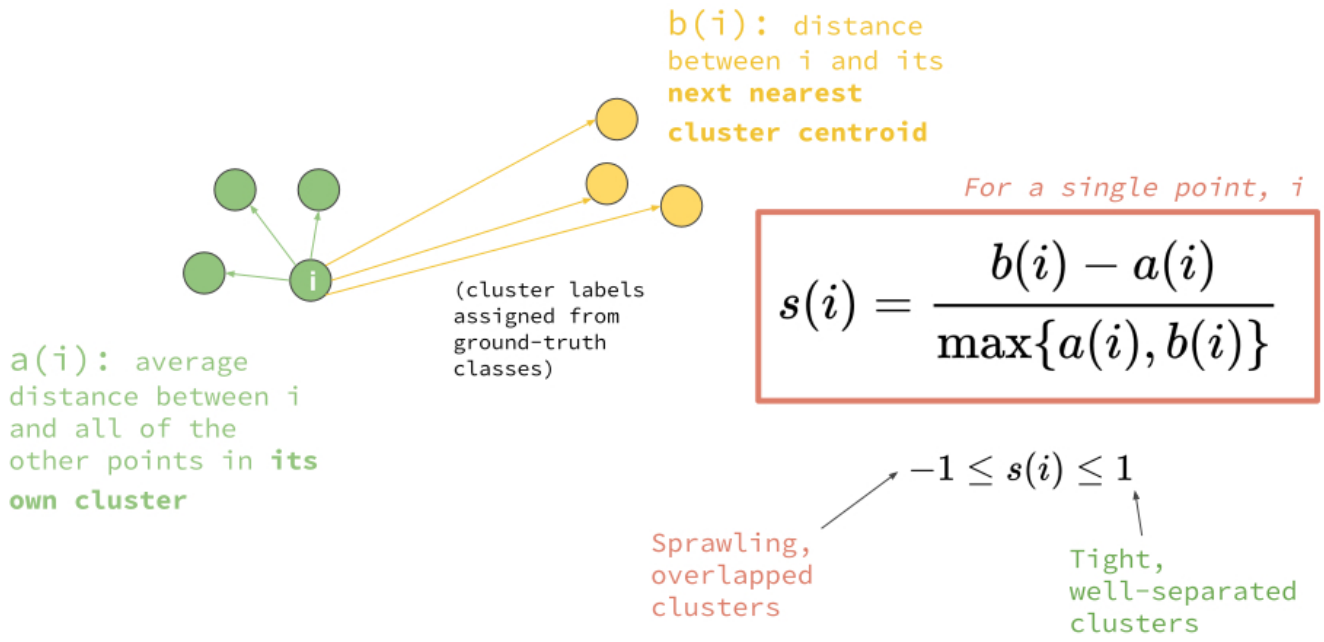
Figure 11 : Silhouette Scores(K-Means)

(generated using matplotlib, Seaborn and PANDAS libraries in Python)

As 2 clusters do not make much business sense, so apart from the silhouette score for 2 clusters, we see three clusters have the highest silhouette score among the cluster number options in the plot. Definitely, we can now go ahead with 3 clusters.

After forming clusters using the 'KMeans' function from 'sklearn.cluster' library, we can attach the cluster label to the data frame.

Silhouette Width for any row or point in data set, say i, is defined by the formula:



- 'Silhouette_samples' function from 'sklearn.metrics' library can be used to calculate silhouette widths for all rows and can be attached to the data frame.
- The minimum silhouette width is coming out to be 0.0027, indicating there is no incorrect mapping in the clustering process.
- The silhouette score for 3 clusters in K-means approach is coming out to be 0.4008, indicating the clusters are not very well demarcated. For well demarcated clusters silhouette score of at least 0.5 is the general acceptable standard.
- There are 72, 71 and 67 observations in the 3 ward linked clusters

Let us visualise the clusters using all possible combinations of 2-D scatter plots using K-means cluster labels as hue:

(zoom to at least 125% for better visibility)

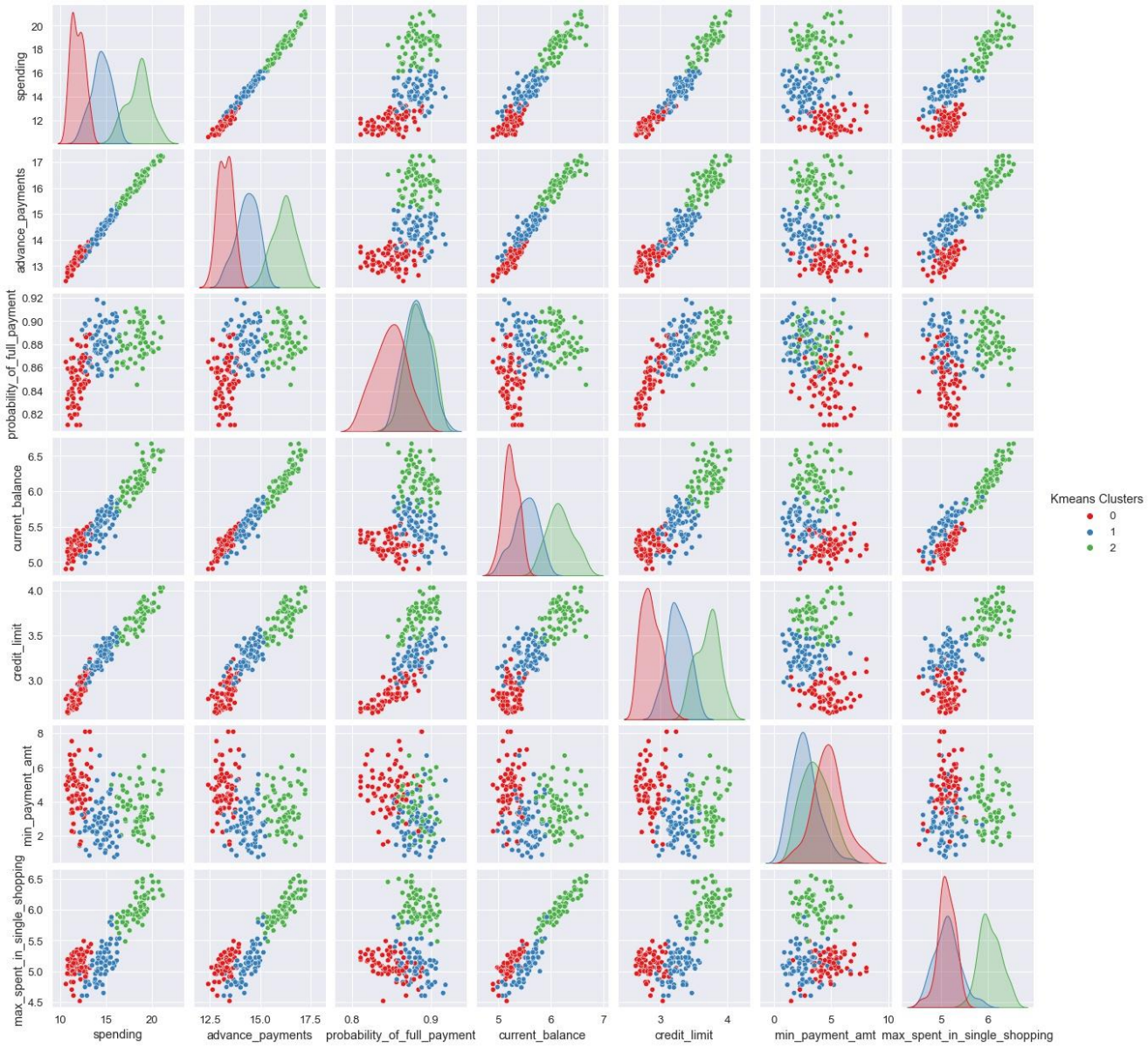


Figure 10 : Pairplot(K-Means)

(generated using '.pairplot()' function of 'Seaborn' and PANDAS libraries in Python)

As the silhouette score is only 0.4008, the clusters are not very well demarcated, which is also evident from the above pairplot.

Let us have a look at general descriptions for 3 clusters:

Cluster 0:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	11.86	0.71	10.59	11.26	11.82	12.40	13.34	2.75	1.14	0.06	11.23	14.11	15.38
advance_payments	13.25	0.36	12.41	12.99	13.25	13.48	13.95	1.54	0.49	0.03	13.47	NaN	NaN
probability_of_full_payment	0.85	0.02	0.81	0.84	0.85	0.86	0.89	0.08	0.03	0.02	0.81	0.88	NaN
current_balance	5.23	0.14	4.90	5.14	5.22	5.34	5.54	0.64	0.20	0.03	5.24	5.40	NaN
credit_limit	2.85	0.14	2.63	2.74	2.84	2.97	3.23	0.60	0.23	0.05	3.03	NaN	NaN
min_payment_amt	4.73	1.33	1.50	4.03	4.80	5.46	8.08	6.58	1.43	0.28	2.13	2.22	2.70
max_spent_in_single_shopping	5.10	0.18	4.52	5.00	5.09	5.22	5.49	0.97	0.22	0.04	5.00	NaN	NaN
Kmeans Clusters	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN	0.00	NaN	NaN
sil_width	0.40	0.16	0.00	0.32	0.46	0.52	0.59	0.59	0.20	0.40	0.00	0.00	0.01

Cluster 1:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	14.44	1.06	12.08	13.82	14.43	15.26	16.44	4.36	1.44	0.07	11.23	14.11	15.38
advance_payments	14.34	0.53	13.15	14.03	14.39	14.76	15.27	2.12	0.73	0.04	13.47	NaN	NaN
probability_of_full_payment	0.88	0.02	0.85	0.87	0.88	0.89	0.92	0.07	0.02	0.02	0.81	0.88	NaN
current_balance	5.51	0.23	4.98	5.38	5.54	5.69	5.92	0.94	0.31	0.04	5.24	5.40	NaN
credit_limit	3.26	0.15	2.94	3.16	3.26	3.38	3.58	0.65	0.22	0.05	3.03	NaN	NaN
min_payment_amt	2.71	1.18	0.77	1.95	2.64	3.33	6.68	5.92	1.38	0.43	2.13	2.22	2.70
max_spent_in_single_shopping	5.12	0.27	4.61	4.96	5.13	5.26	5.88	1.27	0.30	0.05	5.00	NaN	NaN
Kmeans Clusters	1.00	0.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	NaN	NaN
sil_width	0.34	0.17	0.00	0.23	0.37	0.48	0.55	0.55	0.25	0.49	0.00	0.00	0.01

Cluster 2:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	18.50	1.28	15.56	17.59	18.75	19.14	21.18	5.62	1.55	0.07	11.23	14.11	15.38
advance_payments	16.20	0.55	14.89	15.86	16.23	16.58	17.25	2.36	0.72	0.03	13.47	NaN	NaN
probability_of_full_payment	0.88	0.01	0.85	0.87	0.88	0.90	0.91	0.07	0.02	0.02	0.81	0.88	NaN
current_balance	6.18	0.24	5.72	6.01	6.15	6.33	6.68	0.96	0.32	0.04	5.24	5.40	NaN
credit_limit	3.70	0.17	3.39	3.56	3.72	3.81	4.03	0.65	0.24	0.04	3.03	NaN	NaN
min_payment_amt	3.63	1.21	1.47	2.85	3.62	4.42	6.68	5.21	1.57	0.33	2.13	2.22	2.70
max_spent_in_single_shopping	6.04	0.23	5.48	5.88	6.01	6.19	6.55	1.07	0.31	0.04	5.00	NaN	NaN
Kmeans Clusters	2.00	0.00	2.00	2.00	2.00	2.00	2.00	0.00	0.00	0.00	0.00	NaN	NaN
sil_width	0.47	0.15	0.03	0.42	0.52	0.57	0.64	0.61	0.15	0.33	0.00	0.00	0.01

Table 10: General Description of K-Means Clusters

(above tables generated with .describe() function of PANDAS library in Python)

Let us look at the mean values for different features of various clusters in a pivot table:

Kmeans Clusters	advance_payments	credit_limit	current_balance	max_spent_in_single_shopping	min_payment_amt	probability_of_full_payment	spending
0	13.25	2.85	5.23	5.10	4.73	0.85	11.86
1	14.34	3.26	5.51	5.12	2.71	0.88	14.44
2	16.20	3.70	6.18	6.04	3.63	0.88	18.50

Table 11: Means of features for K-Means Clusters

(above table generated with .pivot_table() function of PANDAS library in Python)

- Looking at the feature 'spending' which is the amount spent by the customers per month (in 1000s), we can say cluster 0 customers are the least spending group, cluster 1 customers are the mediocre spending group and cluster 2 customers are the highest spending group.
- Looking at the feature 'advance_payments' which is the amount paid by the customer in advance by cash (in 100s), we see cluster 0 which has the least spending customers, making the least advance payments, cluster 2 which has the highest spending customers, making the highest advance payments. This is also evident as the features 'spending' and 'advance_payments' have Pearson's correlation coefficient as 0.99.

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

For Hierarchical Agglomerative clustering:

Let us check the general description of different clusters, and a pivot table comparing the means of differents features in different clusters:

Cluster 1:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	18.37	1.38	15.38	17.33	18.72	19.14	21.18	5.80	1.81	0.08	11.23	14.11	15.38
advance_payments	16.15	0.60	14.86	15.74	16.21	16.56	17.25	2.39	0.82	0.04	13.47	NaN	NaN
probability_of_full_payment	0.88	0.01	0.85	0.87	0.88	0.90	0.91	0.07	0.02	0.02	0.81	0.88	NaN
current_balance	6.16	0.25	5.71	5.98	6.15	6.31	6.68	0.97	0.33	0.04	5.24	5.40	NaN
credit_limit	3.68	0.17	3.27	3.55	3.69	3.80	4.03	0.77	0.25	0.05	3.03	NaN	NaN
min_payment_amt	3.64	1.21	1.47	2.85	3.63	4.46	6.68	5.21	1.61	0.33	2.13	2.22	2.70
max_spent_in_single_shopping	6.02	0.25	5.44	5.88	5.98	6.19	6.55	1.11	0.31	0.04	5.00	NaN	NaN
Ward Clusters	1.00	0.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	3.00	NaN	NaN

Cluster 2:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	11.87	0.74	10.59	11.25	11.83	12.45	13.37	2.78	1.20	0.06	11.23	14.11	15.38
advance_payments	13.26	0.35	12.41	13.00	13.27	13.52	13.95	1.54	0.52	0.03	13.47	NaN	NaN
probability_of_full_payment	0.85	0.02	0.81	0.83	0.85	0.86	0.89	0.08	0.03	0.02	0.81	0.88	NaN
current_balance	5.24	0.14	4.90	5.14	5.24	5.33	5.54	0.64	0.19	0.03	5.24	5.40	NaN
credit_limit	2.85	0.14	2.63	2.73	2.83	2.97	3.23	0.60	0.24	0.05	3.03	NaN	NaN
min_payment_amt	4.94	1.14	3.08	4.12	4.86	5.47	8.08	5.00	1.35	0.23	2.13	2.22	2.70
max_spent_in_single_shopping	5.12	0.16	4.79	5.00	5.09	5.25	5.49	0.70	0.25	0.03	5.00	NaN	NaN
Ward Clusters	2.00	0.00	2.00	2.00	2.00	2.00	2.00	0.00	0.00	0.00	3.00	NaN	NaN

Cluster 3:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	14.20	1.23	11.23	13.50	14.33	15.03	16.63	5.40	1.53	0.09	11.23	14.11	15.38
advance_payments	14.23	0.60	12.63	13.85	14.28	14.67	15.46	2.83	0.82	0.04	13.47	NaN	NaN
probability_of_full_payment	0.88	0.02	0.83	0.87	0.88	0.89	0.92	0.08	0.02	0.02	0.81	0.88	NaN
current_balance	5.48	0.24	4.90	5.35	5.50	5.66	6.05	1.15	0.31	0.04	5.24	5.40	NaN
credit_limit	3.23	0.18	2.72	3.13	3.22	3.37	3.58	0.86	0.24	0.06	3.03	NaN	NaN
min_payment_amt	2.61	1.12	0.77	1.79	2.50	3.14	6.68	5.92	1.35	0.43	2.13	2.22	2.70
max_spent_in_single_shopping	5.09	0.28	4.52	4.87	5.10	5.22	5.88	1.36	0.35	0.05	5.00	NaN	NaN
Ward Clusters	3.00	0.00	3.00	3.00	3.00	3.00	3.00	0.00	0.00	0.00	3.00	NaN	NaN

Table 12: General Description of Ward Linked Clusters

(above table generated with .describe() function of PANDAS library in Python)

Let us look at the mean values for different features of various clusters in a pivot table:

Ward Clusters	advance_payments	credit_limit	current_balance	max_spent_in_single_shopping	min_payment_amt	probability_of_full_payment	spending
1	16.15	3.68	6.16	6.02	3.64	0.88	18.37
2	13.26	2.85	5.24	5.12	4.95	0.85	11.87
3	14.23	3.23	5.48	5.09	2.61	0.88	14.20

Table 13: Means of features for Ward Linked Clusters

(above table generated with .pivot_table() function of PANDAS library in Python)

- Looking at the feature ‘spending’ which is the amount spent by the customers per month (in 1000s), we can say cluster 2 customers are the least spending group, cluster 3 customers are the mediocre spending group and cluster 1 customers are the highest spending group.

- Looking at the feature ‘min_payment_amt’ which is the minimum amount paid by the customer while making payments for purchases made monthly (in 100s), we see cluster 2 customers make the highest payments though they are the least spending group. This may be because they make planned bulk purchases in order to avoid exhausting their credit limit on multiple transactions.
- Looking at the feature ‘advance_payments’ which is the amount paid by the customer in advance by cash (in 100s), we see cluster 2 which has the least spending customers, make the least advance payments, cluster 1 which has the highest spending customers, make the highest advance payments. This is also evident as the features ‘spending’ and ‘advance_payments’ have Pearson’s correlation coefficient as 0.99.

For recommendations for the business, let us have a look at the pairplot of the data:

(zoom to at least 125% for better visibility)

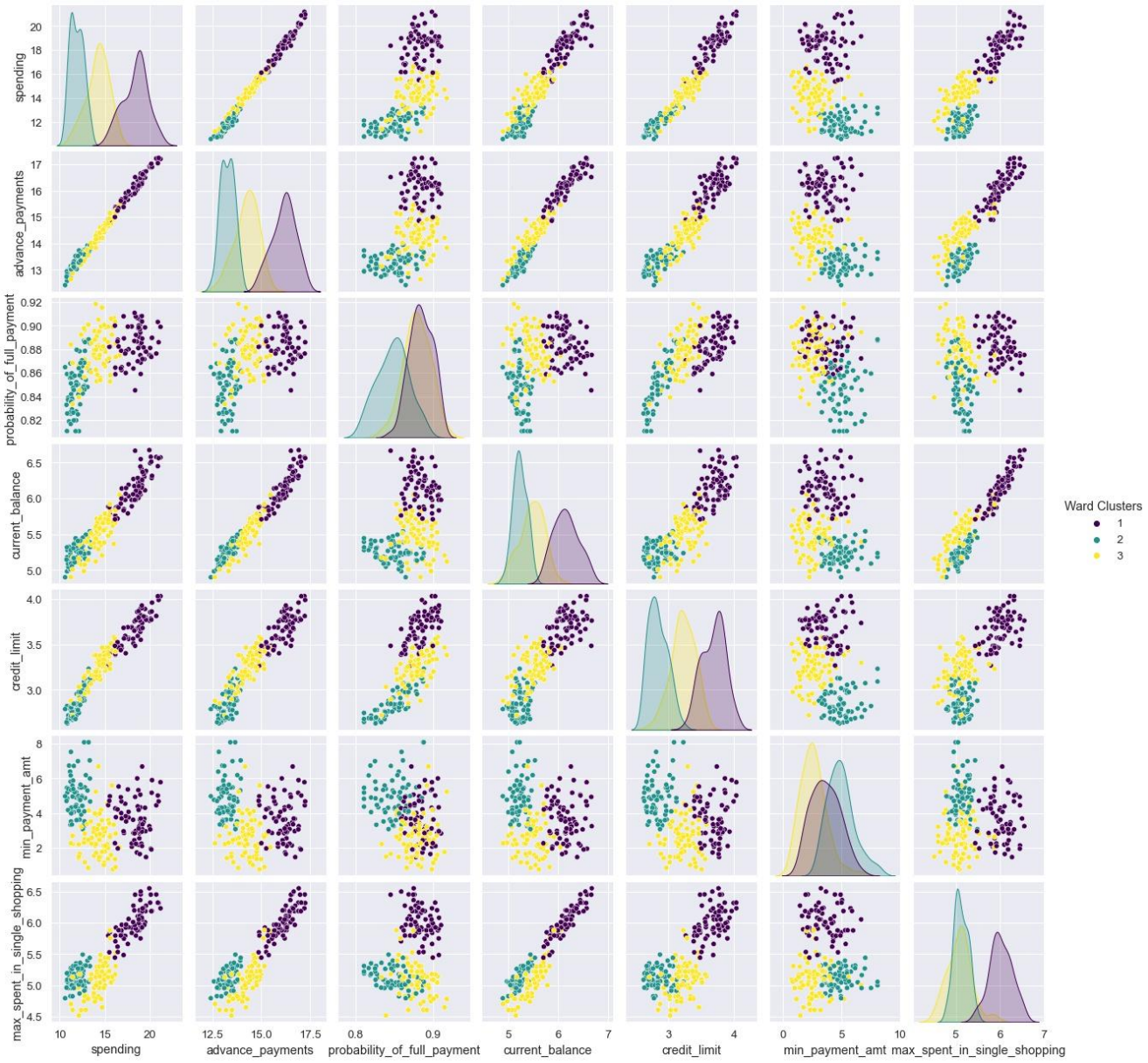


Figure 13 : Pairplot(Ward Linkage)

(generated using 'dendrogram' and 'linkage' functions of 'scipy.cluster.hierarchy' library in Python)

As the silhouette score is only 0.393, the clusters are not very well demarcated, which is also evident from the above pairplot.

- Looking at scatter plots for features 'spending', 'min_payment_amt' and 'credit_limit', we see, though cluster 2 customers are the least spending customers, they make the most of payment in single purchases. This may be because they do not wish to exhaust their credit limit. For these customers we can suggest offers so they contribute to business more by making more regular transactions rather than single bulk transactions.

- Looking at scatter plots for feature ‘probability_of_full_payment’, we see cluster 2 customers are the majority of the ones who do not believe in making full payments. We can suggest offers that benefit them if they make full payments. These offers need not be given to remaining clusters’ customers as they are already making a good amount of full payments.
- Looking at feature ‘advance_payments’ we see the high spending customers(cluster 1), tend to clear off their dues more often. We need not suggest to them any offers, rather try to withdraw extra unpaid services from them as they seem to be in no need of them.
- Looking at features ‘credit_limit’ and ‘current_balance’, we see, though cluster 1 customers are the high spending group, they do not use their credit limit to full extent, giving the bank more business. Cluster 2 and 3 customers are using their limit optimally. For them we can suggest offers or rather set a better optimal credit limit so they bring more business to the bank.
- Looking at distribution of the feature ‘max_spent_in_single_shopping’, we see the cluster 2 and 3 customers have near-identical means but vary in peaks and ranges. Cluster 2 customers who are the least spending group show erratic behavior when it comes to maximum spent in one purchase. To these customers we can suggest offers that push them to make sustained regular purchases rather than random bulk purchases.

For K-Means Clustering:

Let us check the general description of different clusters, and a pivot table comparing the means of differents features in different clusters:

Cluster 0:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	11.86	0.71	10.59	11.26	11.82	12.40	13.34	2.75	1.14	0.06	11.23	14.11	15.38
advance_payments	13.25	0.36	12.41	12.99	13.25	13.48	13.95	1.54	0.49	0.03	13.47	NaN	NaN
probability_of_full_payment	0.85	0.02	0.81	0.84	0.85	0.86	0.89	0.08	0.03	0.02	0.81	0.88	NaN
current_balance	5.23	0.14	4.90	5.14	5.22	5.34	5.54	0.64	0.20	0.03	5.24	5.40	NaN
credit_limit	2.85	0.14	2.63	2.74	2.84	2.97	3.23	0.60	0.23	0.05	3.03	NaN	NaN
min_payment_amt	4.73	1.33	1.50	4.03	4.80	5.46	8.08	6.58	1.43	0.28	2.13	2.22	2.70
max_spent_in_single_shopping	5.10	0.18	4.52	5.00	5.09	5.22	5.49	0.97	0.22	0.04	5.00	NaN	NaN
Kmeans Clusters	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN	0.00	NaN	NaN
sil_width	0.40	0.16	0.00	0.32	0.46	0.52	0.59	0.59	0.20	0.40	0.00	0.00	0.01

Cluster 1:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	14.44	1.06	12.08	13.82	14.43	15.26	16.44	4.36	1.44	0.07	11.23	14.11	15.38
advance_payments	14.34	0.53	13.15	14.03	14.39	14.76	15.27	2.12	0.73	0.04	13.47	NaN	NaN
probability_of_full_payment	0.88	0.02	0.85	0.87	0.88	0.89	0.92	0.07	0.02	0.02	0.81	0.88	NaN
current_balance	5.51	0.23	4.98	5.38	5.54	5.69	5.92	0.94	0.31	0.04	5.24	5.40	NaN
credit_limit	3.26	0.15	2.94	3.16	3.26	3.38	3.58	0.65	0.22	0.05	3.03	NaN	NaN
min_payment_amt	2.71	1.18	0.77	1.95	2.64	3.33	6.68	5.92	1.38	0.43	2.13	2.22	2.70
max_spent_in_single_shopping	5.12	0.27	4.61	4.96	5.13	5.26	5.88	1.27	0.30	0.05	5.00	NaN	NaN
Kmeans Clusters	1.00	0.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	NaN	NaN
sil_width	0.34	0.17	0.00	0.23	0.37	0.48	0.55	0.55	0.25	0.49	0.00	0.00	0.01

Cluster 2:

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode1	mode2	mode3
spending	18.50	1.28	15.56	17.59	18.75	19.14	21.18	5.62	1.55	0.07	11.23	14.11	15.38
advance_payments	16.20	0.55	14.89	15.86	16.23	16.58	17.25	2.36	0.72	0.03	13.47	NaN	NaN
probability_of_full_payment	0.88	0.01	0.85	0.87	0.88	0.90	0.91	0.07	0.02	0.02	0.81	0.88	NaN
current_balance	6.18	0.24	5.72	6.01	6.15	6.33	6.68	0.96	0.32	0.04	5.24	5.40	NaN
credit_limit	3.70	0.17	3.39	3.56	3.72	3.81	4.03	0.65	0.24	0.04	3.03	NaN	NaN
min_payment_amt	3.63	1.21	1.47	2.85	3.62	4.42	6.68	5.21	1.57	0.33	2.13	2.22	2.70
max_spent_in_single_shopping	6.04	0.23	5.48	5.88	6.01	6.19	6.55	1.07	0.31	0.04	5.00	NaN	NaN
Kmeans Clusters	2.00	0.00	2.00	2.00	2.00	2.00	2.00	0.00	0.00	0.00	0.00	NaN	NaN
sil_width	0.47	0.15	0.03	0.42	0.52	0.57	0.64	0.61	0.15	0.33	0.00	0.00	0.01

Table 14: General Description of K-Means Clusters

(above tables generated with .describe() function of PANDAS library in Python)

Let us look at the mean values for different features of various clusters in a pivot table:

Kmeans Clusters	advance_payments	credit_limit	current_balance	max_spent_in_single_shopping	min_payment_amt	probability_of_full_payment	spending
0	13.25	2.85	5.23	5.10	4.73	0.85	11.86
1	14.34	3.26	5.51	5.12	2.71	0.88	14.44
2	16.20	3.70	6.18	6.04	3.63	0.88	18.50

Table 15: Means of features for K-Means Clusters

(above table generated with .pivot_table() function of PANDAS library in Python)

- Looking at the feature ‘spending’ which is the amount spent by the customers per month (in 1000s), we can say cluster 0 customers are the least spending group, cluster 1 customers are the mediocre spending group and cluster 2 customers are the highest spending group.
- Looking at the feature ‘advance_payments’ which is the amount paid by the customer in advance by cash (in 100s), we see cluster 0 which has the least spending customers, making the least advance payments, cluster 2 which has the highest spending customers, making the highest advance payments. This is also evident as the features ‘spending’ and ‘advance_payments’ have Pearson’s correlation coefficient as 0.99.

For recommendations for the business, let us have a look at the pairplot of the data:

(zoom to at least 125% for better visibility)

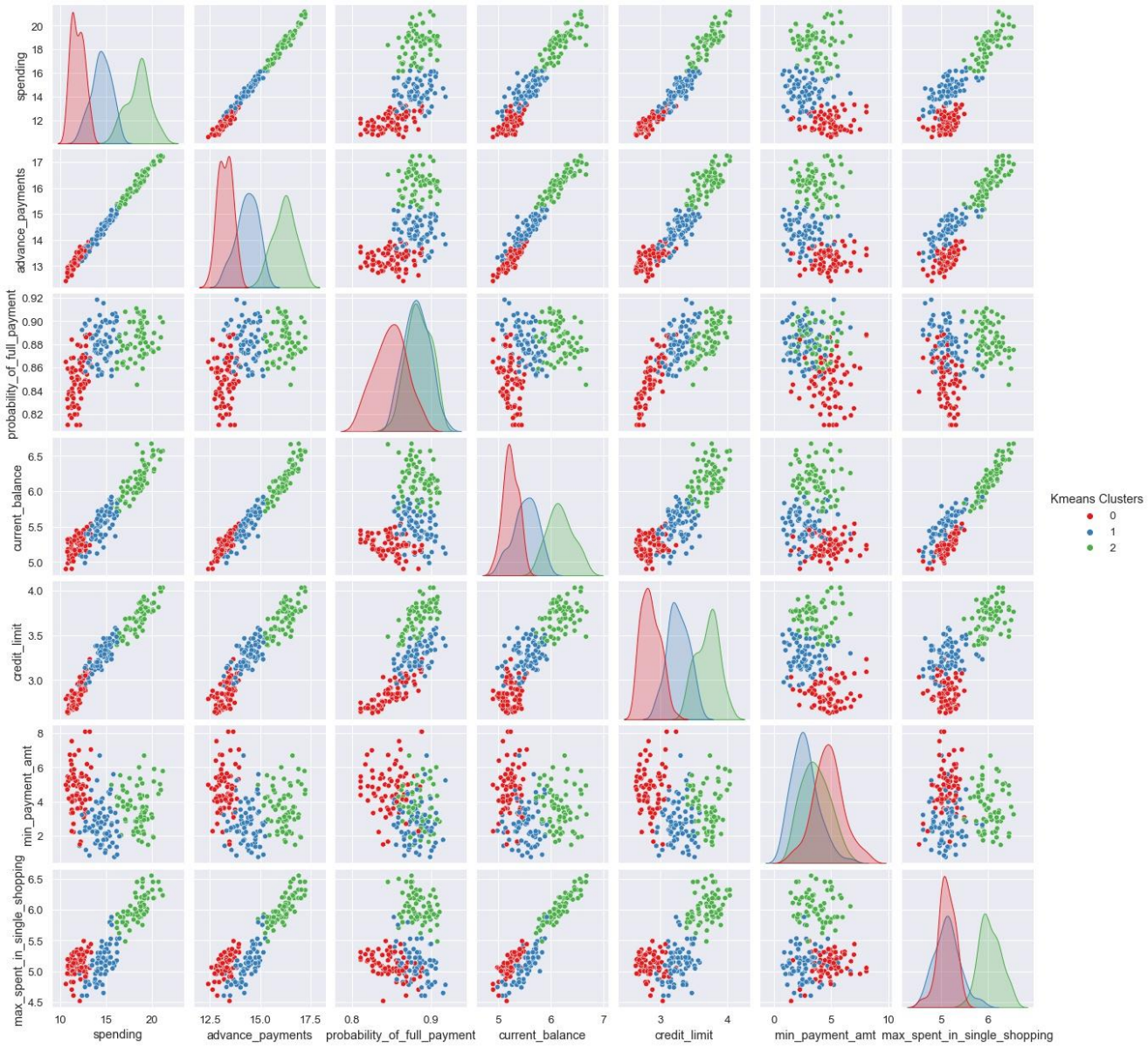


Figure 14 : Pairplot(K-Means)

(generated using '.pairplot()' function of 'Seaborn' and PANDAS libraries in Python)

As the silhouette score is only 0.4008, the clusters are not very well demarcated, which is also evident from the above pairplot.

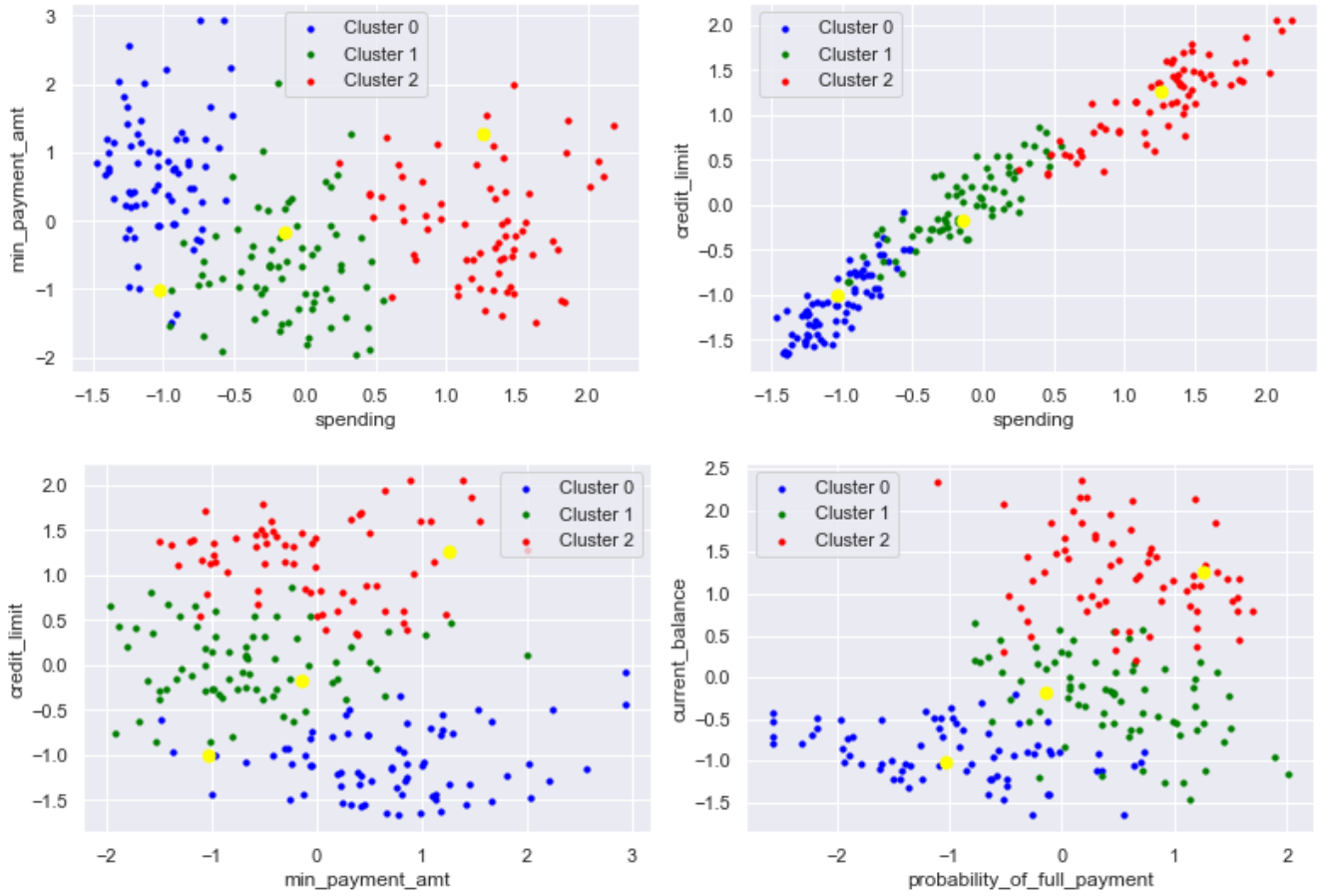
As we have applied 'fit' operation on the scaled data in the K-Means model in Python, we get centroids for scaled data from the model. Let us have a look at the centroids of the scaled data:

Kmeans Clusters	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
0	-1.030253	-1.006649	-0.965588	-0.897685	-1.085583	0.695182	-0.624809
1	-0.141119	-0.170043	0.449804	-0.257814	0.001647	-0.663892	-0.585893
2	1.256682	1.261966	0.560989	1.237883	1.164852	-0.043534	1.292308

Table 16: Centroids for scaled features(K-Means Clusters)

(above table generated with PANDAS library in Python)

Let us see scatter plots of few scaled features with centroids(marked in yellow):



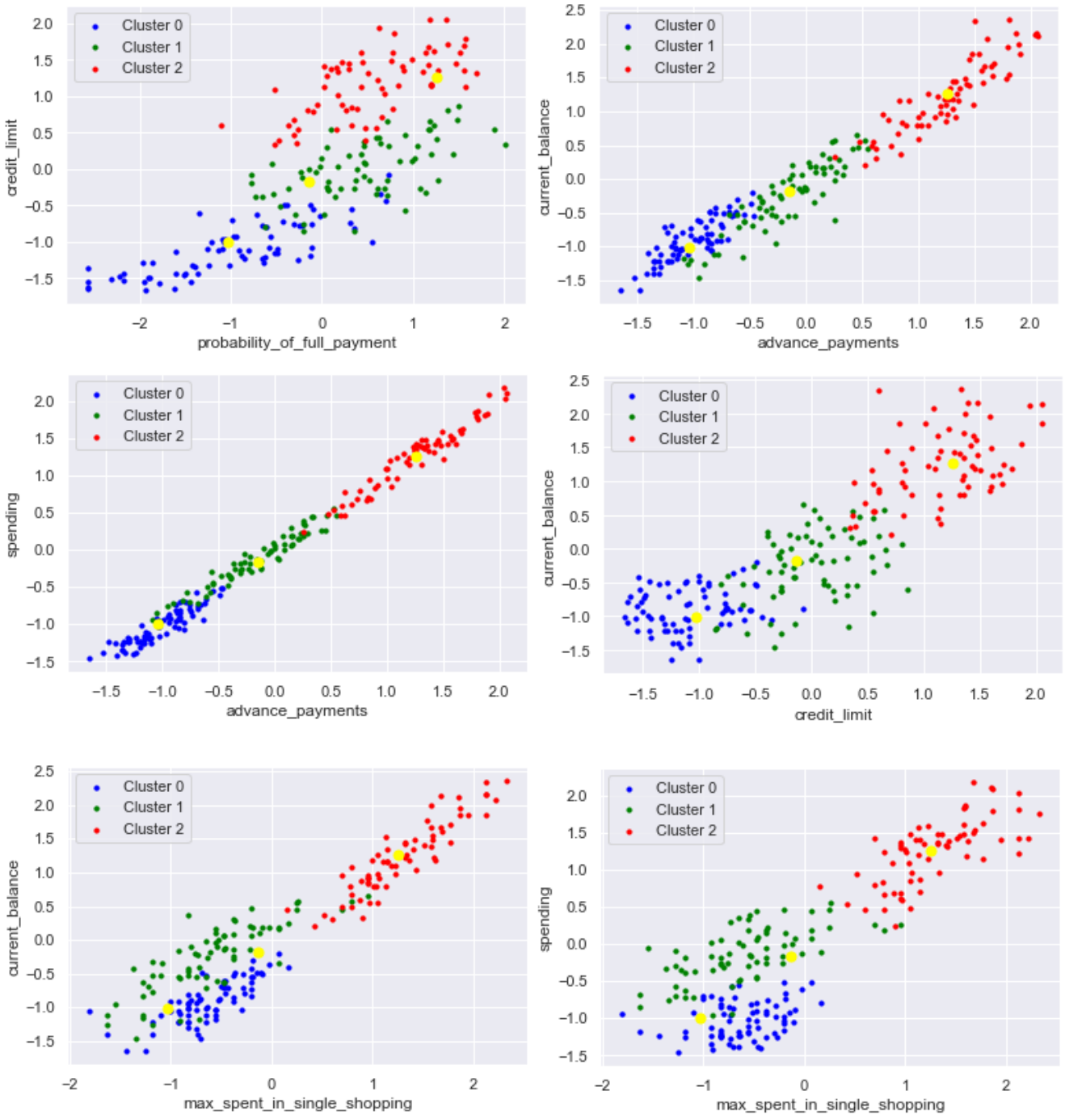


Figure 15 : 2-D scatter plots with centroids(K-Means)

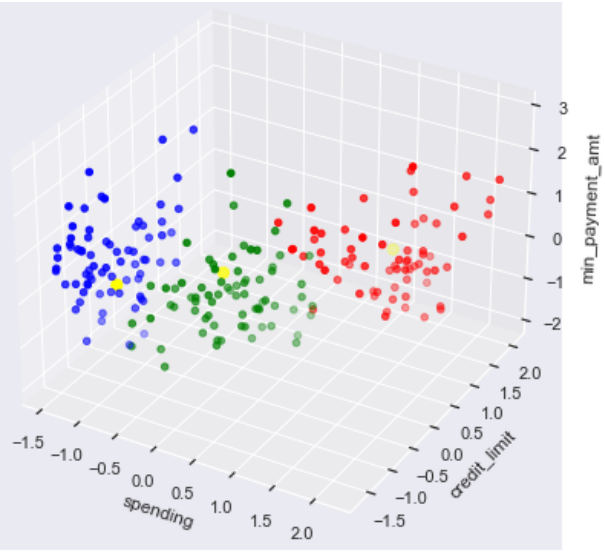
(generated using matplotlib, Seaborn and PANDAS libraries in Python)

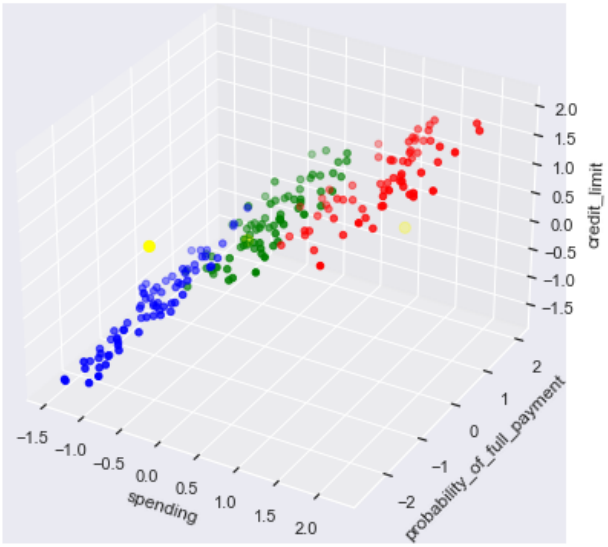
- Looking at pairplot and scatter plots for features 'spending', 'min_payment_amt' and 'credit_limit', we see, though cluster 0 customers are the least spending customers, they make the most of payment in single purchases. This may be because they do not wish to exhaust

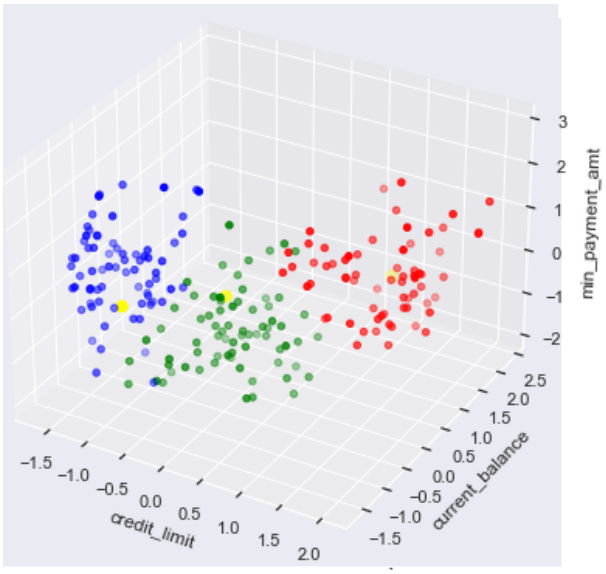
their credit limit. For these customers we can suggest offers so they contribute to business more by making more regular transactions rather than single bulk transactions.

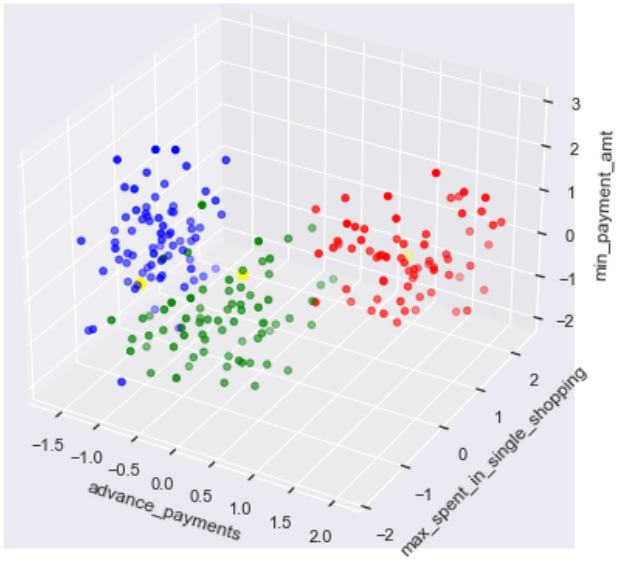
- Looking at pairplot and scatter plots for feature ‘probability_of_full_payment’, we see cluster 0 customers are the majority of the ones who do not believe in making full payments. We can suggest offers that benefit them if they make full payments. These offers need not be given to remaining clusters’ customers as they are already making a good amount of full payments.
- Looking at feature ‘advance_payments’ we see the high spending customers(cluster 2), tend to clear off their dues more often. We need not suggest to them any offers, rather try to withdraw extra unpaid services from them as they seem to be in no need of them.
- Looking at features ‘credit_limit’ and ‘current_balance’, we see, though cluster 2 customers are the high spending group, they do not use their credit limit to full extent, giving the bank more business. Cluster 0 and 1 customers are using their limit optimally. For them we can suggest offers or rather set a better optimal credit limit so they bring more business to the bank.
- Looking at distribution of the feature ‘max_spent_in_single_shopping’, we see the cluster 0 and 1 customers have near-identical means but vary in peaks and ranges. Cluster 0 customers who are the least spending group show erratic behavior when it comes to maximum spent in one purchase. To these customers we can suggest offers that push them to make sustained regular purchases rather than random bulk purchases.

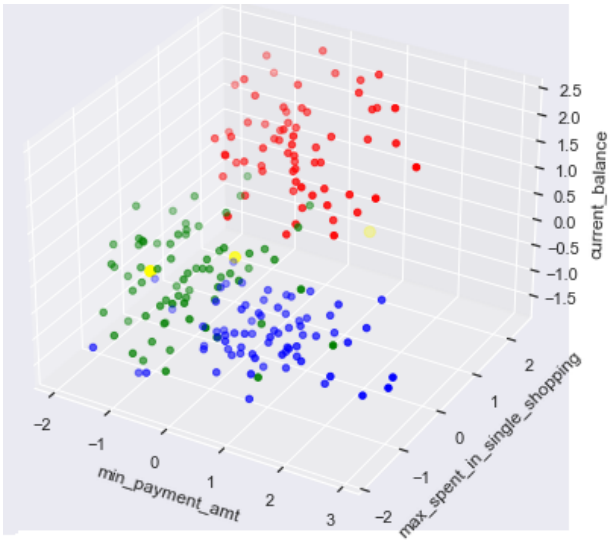
Let us now try to analyse few 3-D plots of scaled features with centroids(marked in yellow) and see how they are distributed:

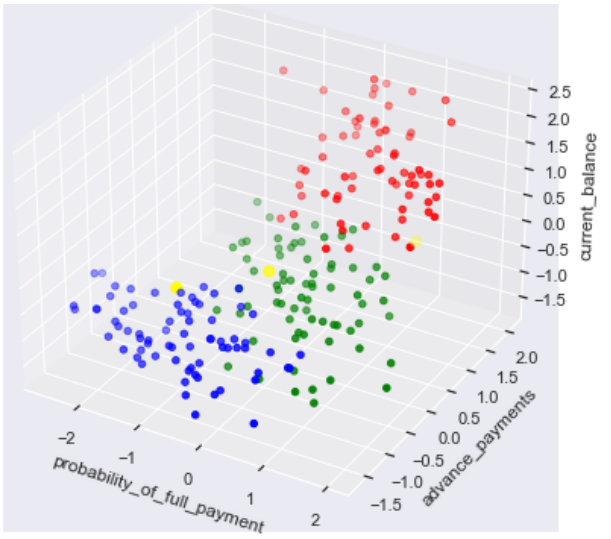





min_payment_amt
credit_limit
current_balance


min_payment_amt
advance_payments
max_spent_in_single_shopping


current_balance
min_payment_amt
max_spent_in_single_shopping


current_balance
probability_of_full_payment
advance_payments

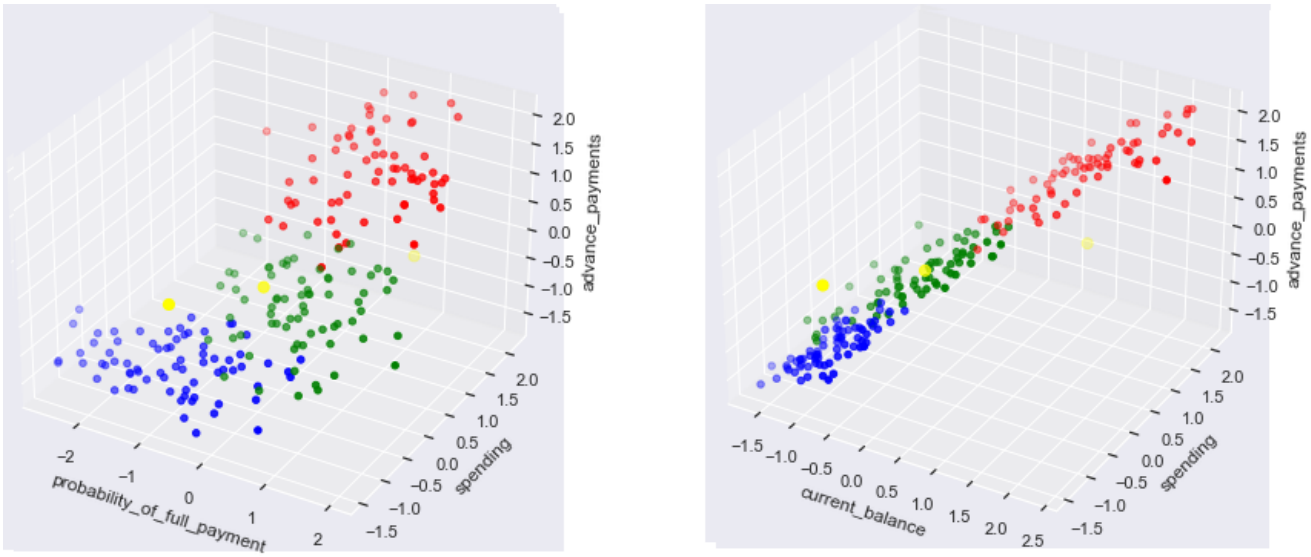


Figure 16 : 3-D scatter plots with centroids(K-Means)

(generated using matplotlib, Seaborn and PANDAS libraries in Python)

We see lots of overlapping and not so well demarcated clusters. This is also because of a silhouette score of 0.4008.

By comparing the pairplots and pivot tables of the clusters of both hierarchical and K- Means clustering, we see the three clusters in either case are identical. Silhouette scores of both the clustering methods also suggest the same as they are nearly equal.

Problem 2

Executive Summary

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

Introduction

The objective is to use supervised learning predictive models to give insights about data and the business problem. Specifically, we will be using CART(Decision Tree), Random Forest and Artificial Neural Network to predict and draw insights.

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

Let us have a look at first few records of data:

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

Table 17: Sample of Data(Problem 2)

(above table generated with .head() function of PANDAS library in Python)

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 in days(Duration)

7. Destination of the tour (Destination)

8. Amount worth of sales per customer in procuring tour insurance policies in rupees (in 100's) (Sales)

9. The commission received for tour insurance firm in percentage of sales(Commission)

10. Age of insured (Age)

Exploratory Data Analysis

Let us check the data types of variables and the missing values(if any):

```
<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
```

Table 18: General Information of Data(Problem2)

(above table generated by '.info' of 'PANDAS' in Python)

We can see that

- shape of data is (3000x10)
- there are total 3000 entries(rows) of individuals in data indexed from 0 to 2999
- there are 10 columns in total('Age', 'Agency_Code', 'Type', 'Claimed', 'Commision', 'Channel', 'Duration', 'Sales', 'Product Name', 'Destination')
- there is no null value in data set as all columns have 3000 non null objects
- variables 'Age' and 'Duration' are of data type integer(64 bits), variables 'Commision' and 'Sales' are of data type float(64 bits), and variables 'Agency_Code', 'Type', 'Claimed', 'Channel', 'Product Name' and 'Destination' are of data type object()

- the memory usage by data set is 234.5kb

(Heatmap and Pairplot provided after Univariate Analysis)

Let us now move to the general description of numeric data(minimum values, maximum values, measures of central tendencies like mean, median, mode etc.):

	mean	std	min	25%	median	75%	max	range	IQR	CV	mode	d-type
Age	38.09	10.46	8.0	32.0	36.00	42.00	84.00	76.00	10.00	0.27	36	int64
Commision	14.53	25.48	0.0	0.0	4.63	17.24	210.21	210.21	17.24	1.75	0.0	float64
Duration	70.00	134.05	-1.0	11.0	26.50	63.00	4580.00	4581.00	52.00	1.92	8	int64
Sales	60.25	70.73	0.0	20.0	33.00	69.00	539.00	539.00	49.00	1.17	20.0	float64

Table 19: General Description of Numeric Data(Problem 2)

(above table generated with .describe() function of PANDAS library in Python)

From above table, we conclude:

- values in all the features seem to be correct
- the mean values are distant apart, indicating features have values in different ranges, which is also being confirmed by 'range'
- except for the feature 'Age', the coefficient of variation is quite high for all other numeric features
- the mode of feature 'Commision' is 0

Let us now move to the general description of 'object' data:

	Agency_Code	Type	Claimed	Channel	Product Name	Destination
count	3000	3000	3000	3000	3000	3000
unique	4	2	2	2	5	3
top	EPX	Travel Agency	No	Online	Customised Plan	ASIA
freq	1365	1837	2076	2954	1136	2465

Table 20: General Description of 'Object' type data(Problem 2)

(above table generated with .describe() function of PANDAS library in Python)

The object data seems to be fine.

From .info() we already saw there are no null values in the data set. '.isnull().sum().sum()' also confirms the same yielding 0 as output, i.e., there are 0 null entries in the data set.

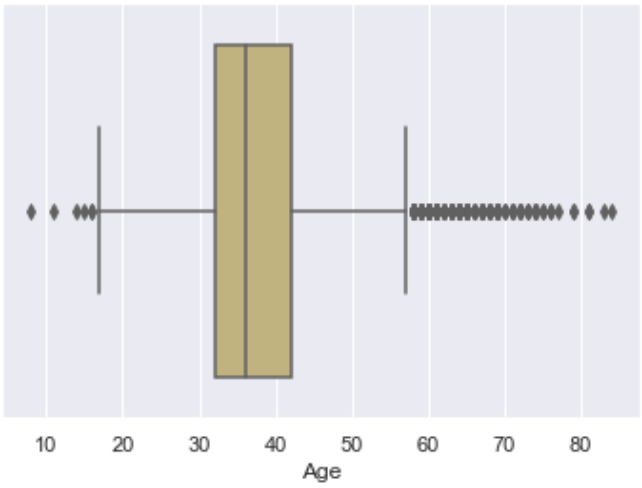
There are 139 duplicate entries in the data set which has been confirmed by the '.duplicated()' function of PANDAS library in Python. As there is no unique identifier in the data set, we can not be sure if these are actually duplicate entries or different entries with identical characters. As there are only 139 entries, and also there is a high chance of similar types of claim entries, it would be wise to retain them.

Let us now go for the Univariate analysis of numeric features in the data set. We will go for general descriptions, distribution plots with kernel density estimates and boxplots for all the features one by one:

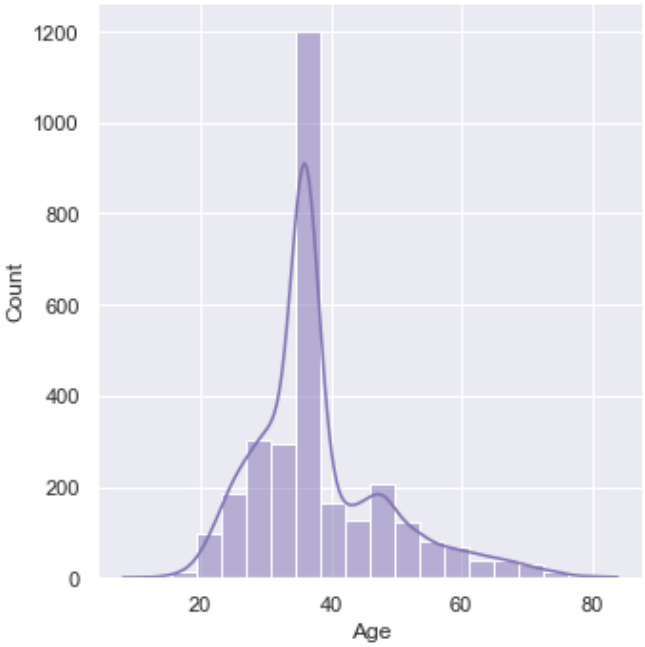
(zoom to at least 125% for better visibility)

```
Description of Age
--------------------------
count    3000.000000
mean       38.091000
std        10.463518
min         8.000000
25%        32.000000
50%        36.000000
75%        42.000000
max        84.000000
Name: Age, dtype: float64
```

BoxPlot of Age

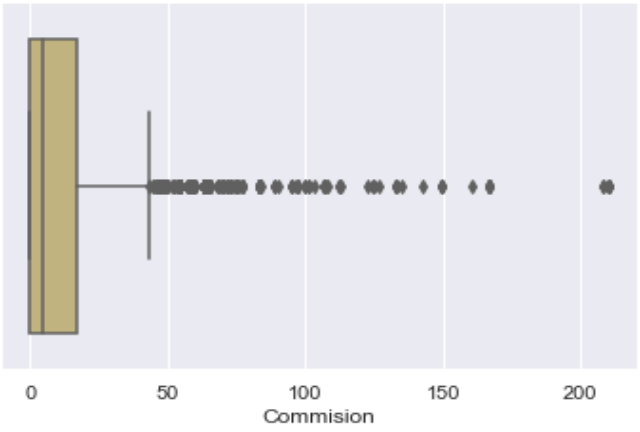


Distribution of Age

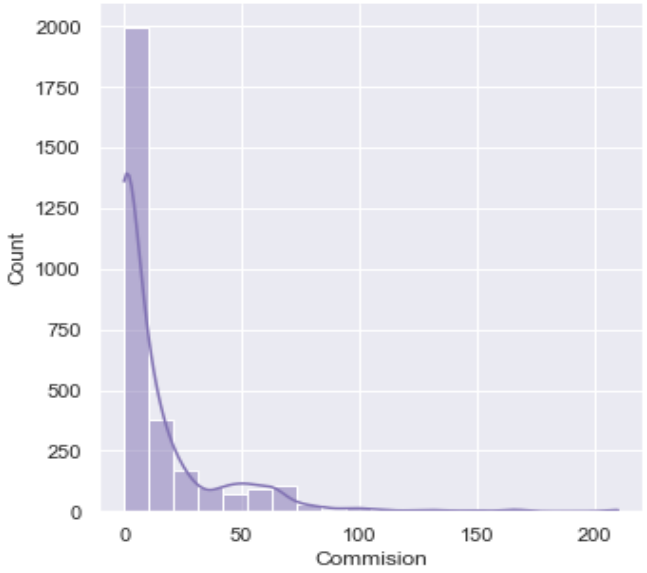


```
Description of Commision
--------------------------------
count    3000.000000
mean       14.529203
std        25.481455
min         0.000000
25%         0.000000
50%         4.630000
75%        17.235000
max       210.210000
Name: Commision, dtype: float64
```

BoxPlot of Commision

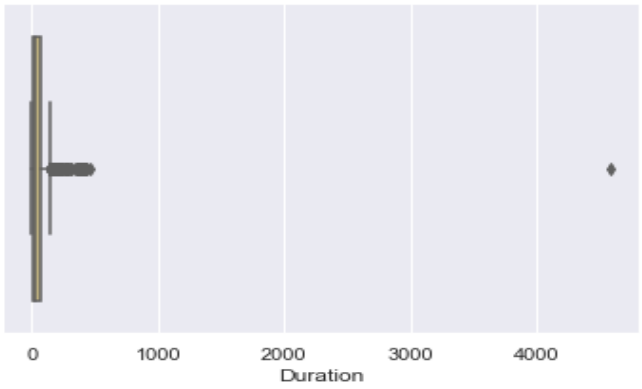


Distribution of Commision




```
Description of Duration
--------------------------------
count    3000.000000
mean       70.001333
std       134.053313
min        -1.000000
25%        11.000000
50%        26.500000
75%        63.000000
max      4580.000000
Name: Duration, dtype: float64
```

BoxPlot of Duration



Distribution of Duration

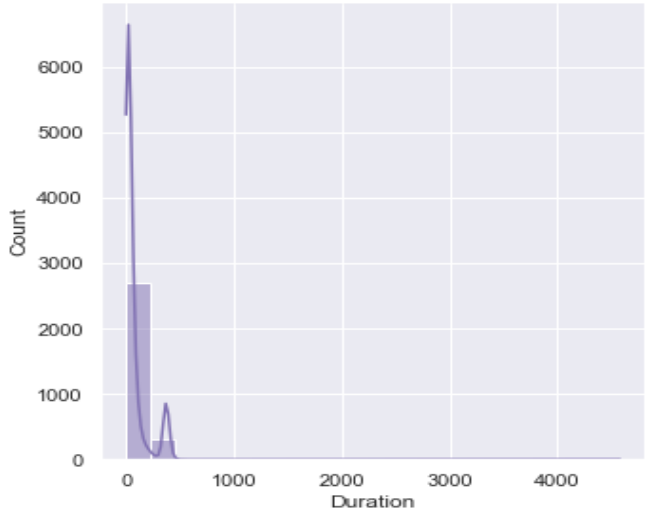


Figure 17: Univariate Analysis of Numeric Data(Problem 2)

(above descriptions generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

We can see:

- the general description of all features
- the variables are in different orders or ranges which is evident from 'min' and 'max' values
- none of the features are nearly-normal, all have some amount of skewness
- except for 'Age' which is skewed on both sides, all other features are right skewed
- except for feature 'Age' which has outliers on both sides, all other features have outliers on the right side
- there is one outlier for feature 'Duration' that is quite far from rest of the data

Let us now go for the Univariate analysis of 'object' data features in the data set. We will check the unique values, count of unique values and proportion(i.e, normalized values) of unique values in each feature:

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

 AGENCY_CODE (normalized)
--------------------
EPX    0.455000
C2B    0.308000
CWT    0.157333
JZI    0.079667
Name: Agency_Code, dtype: float64
```

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

 PRODUCT NAME (normalized)
--------------------
Customised Plan      0.378667
Cancellation Plan    0.226000
Bronze Plan          0.216667
Silver Plan          0.142333
Gold Plan            0.036333
Name: Product Name, dtype: float64
```

```
TYPE :  2 unique values
--------------------
Travel Agency    1837
Airlines         1163
Name: Type, dtype: int64

 TYPE (normalized)
--------------------
Travel Agency    0.612333
Airlines         0.387667
Name: Type, dtype: float64
```

```
CHANNEL :  2 unique values
--------------------
Online     2954
Offline      46
Name: Channel, dtype: int64

 CHANNEL (normalized)
--------------------
Online     0.984667
Offline    0.015333
Name: Channel, dtype: float64
```

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

 DESTINATION (normalized)
--------------------
ASIA        0.821667
Americas    0.106667
EUROPE      0.071667
Name: Destination, dtype: float64
```

```
CLAIMED :  2 unique values
--------------------
No     2076
Yes     924
Name: Claimed, dtype: int64

 CLAIMED (normalized)
--------------------
No     0.692
Yes    0.308
Name: Claimed, dtype: float64
```

Tables 21: Univariate Analysis of Categorical Data(Problem 2)

(above tables generated with ‘.value_counts()’ function of PANDAS library using Python)

Before drawing inferences, let us visualise above results in form of countplots:

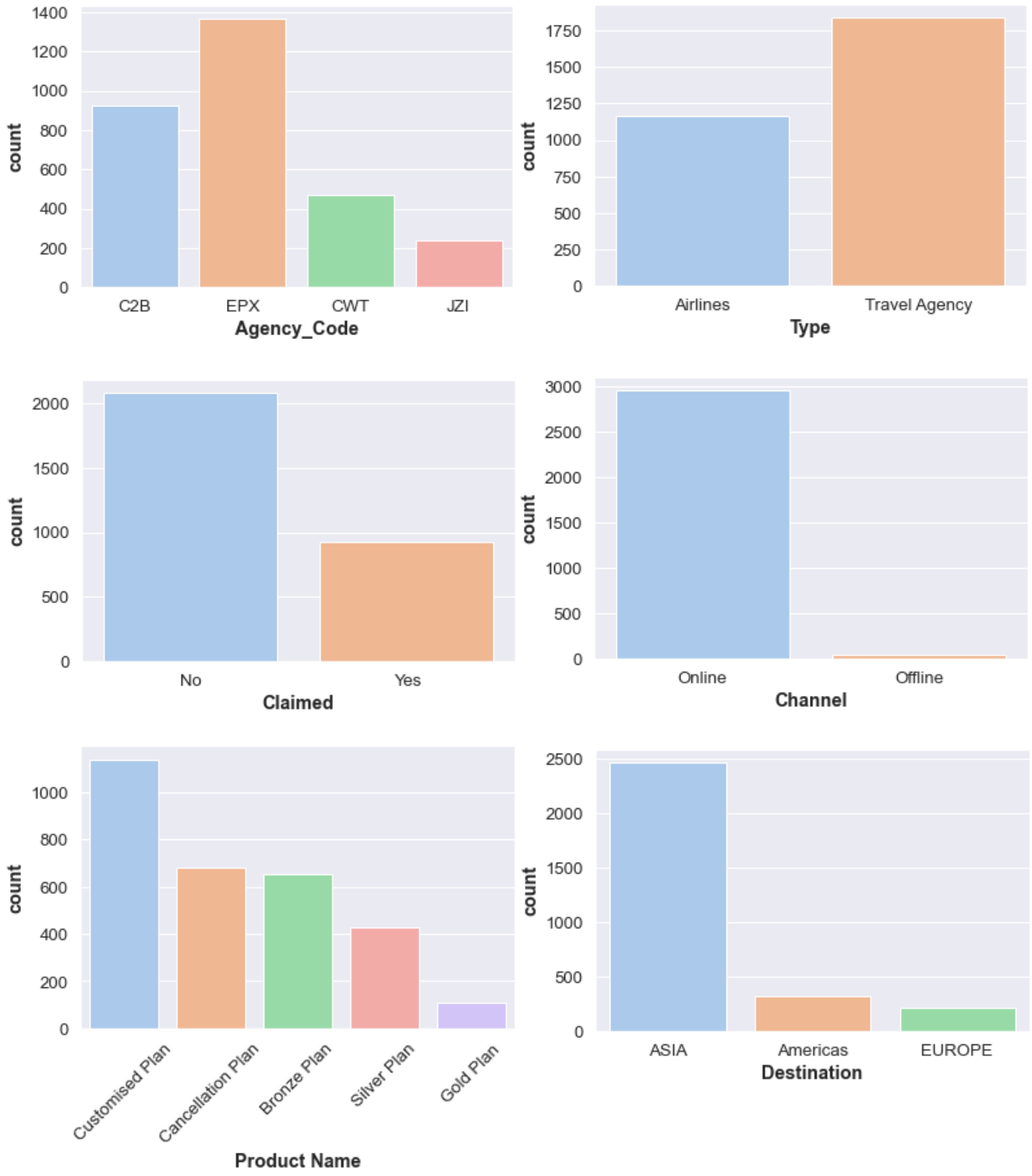


Figure 18: Count plots of Categorical Data(Problem 2)

(above plots generated using '.countplot()' function of Seaborn library in Python)

We can see:

- from the description, all the classes in the features seem to be correct without any anomalies
- target feature 'Claimed' has 30.8% observations in class 'Yes', and remaining 69.2% observations in the class 'No'
- features 'Channel' and 'Destination' are biased to at one class in the respective features, i.e, in the feature 'Channel', 98.47% of data points are in class 'Online', and in the feature 'Destination', 82.47% of the data points are in the class 'ASIA'
- in the feature 'Channel', about 98.47% of data points are in the class 'Online', only 1.53% of remaining data points are in the class 'Offline'. We can drop this feature before building our models

Let us go for bivariate and multivariate analysis now.

To check the correlations amongst numerical features, let us have a look at the Pearson's Correlation coefficients in the form of a heatmap:

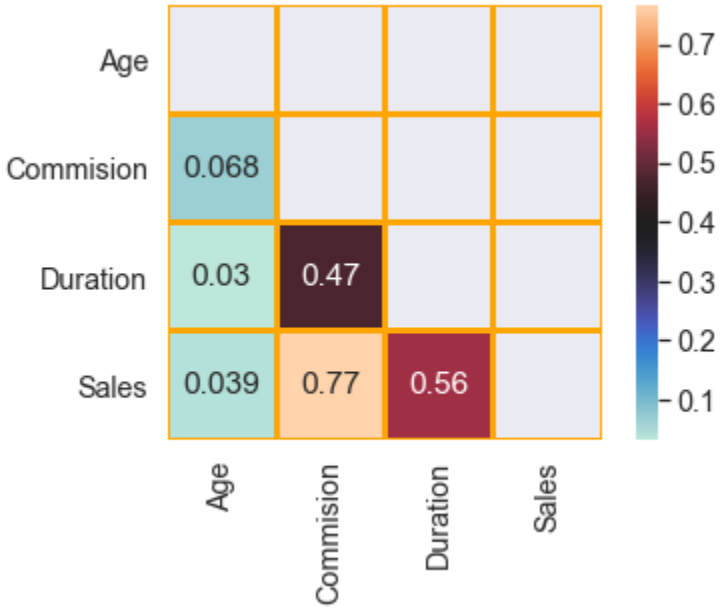


Figure 19: Correlation Heatmap(Problem 2)

(above plot generated using '.heatmap()' function of Seaborn library in Python)

- Pearson's Correlation coefficients near to 1 or -1 are highly positively correlated and highly negatively correlated respectively. Correlation values near to 0 are not or minimally correlated to each other.
- We see that most of Pearson's coefficients are positive. Features 'Commision' and 'Sales' have a correlation coefficient of 0.77, which is the highest, rest correlations are moderate or weak.

Now, let us view the pairplot of the numeric features:

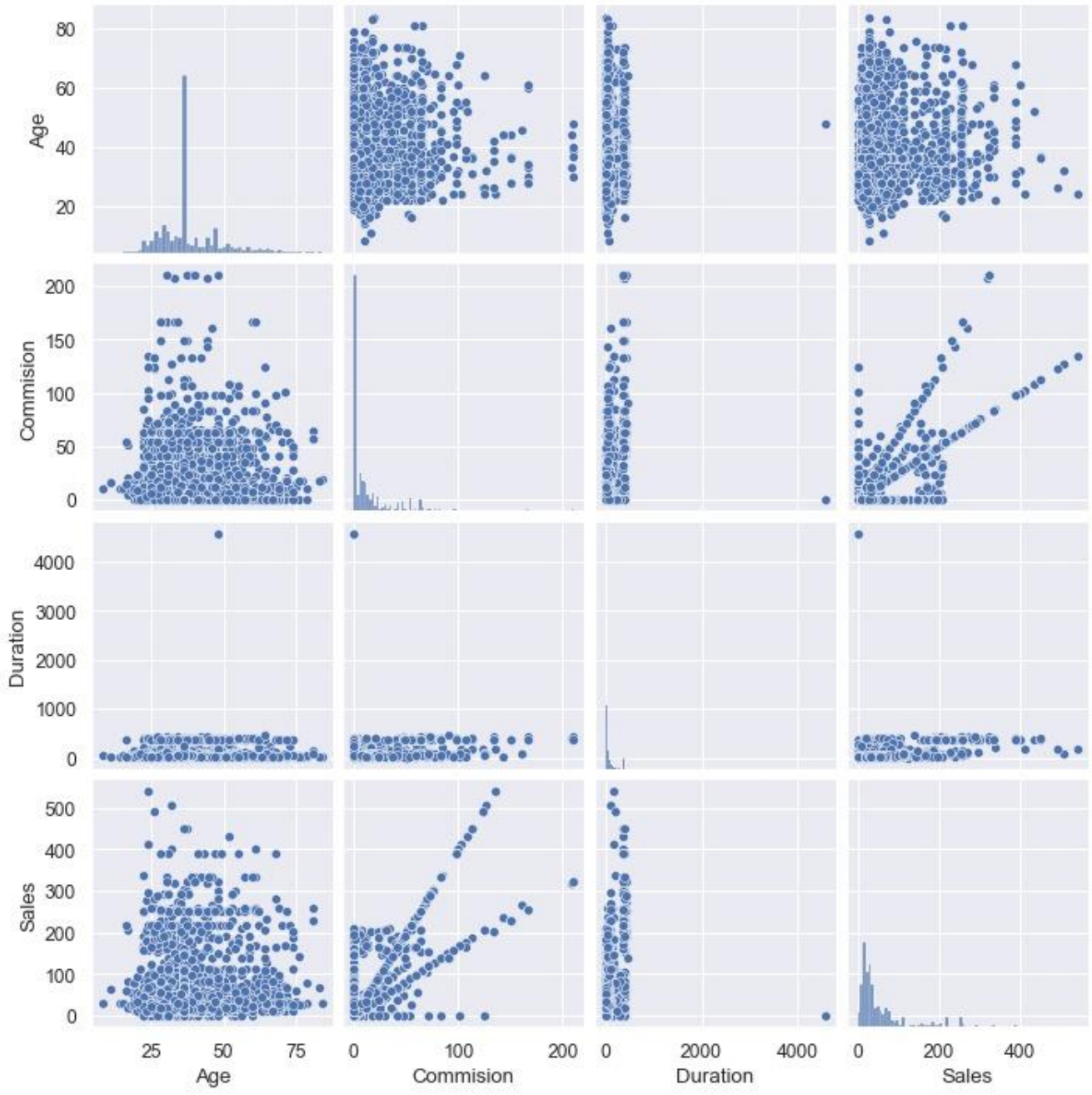


Figure 20: Pairplot(Problem 2)

(above figure generated with '.pairplot()' function of Seaborn library in Python)

Pairplot shows the relationship between the numeric variables in the form of scatterplot and the distribution of the variables in the form of histogram.

The pattern of clouds of points of ‘Commision’ and ‘Sales’ features clearly depict a high positive relationship.

Let us visualise how the target feature ‘Claimed’ is distributed over other categorical variables:

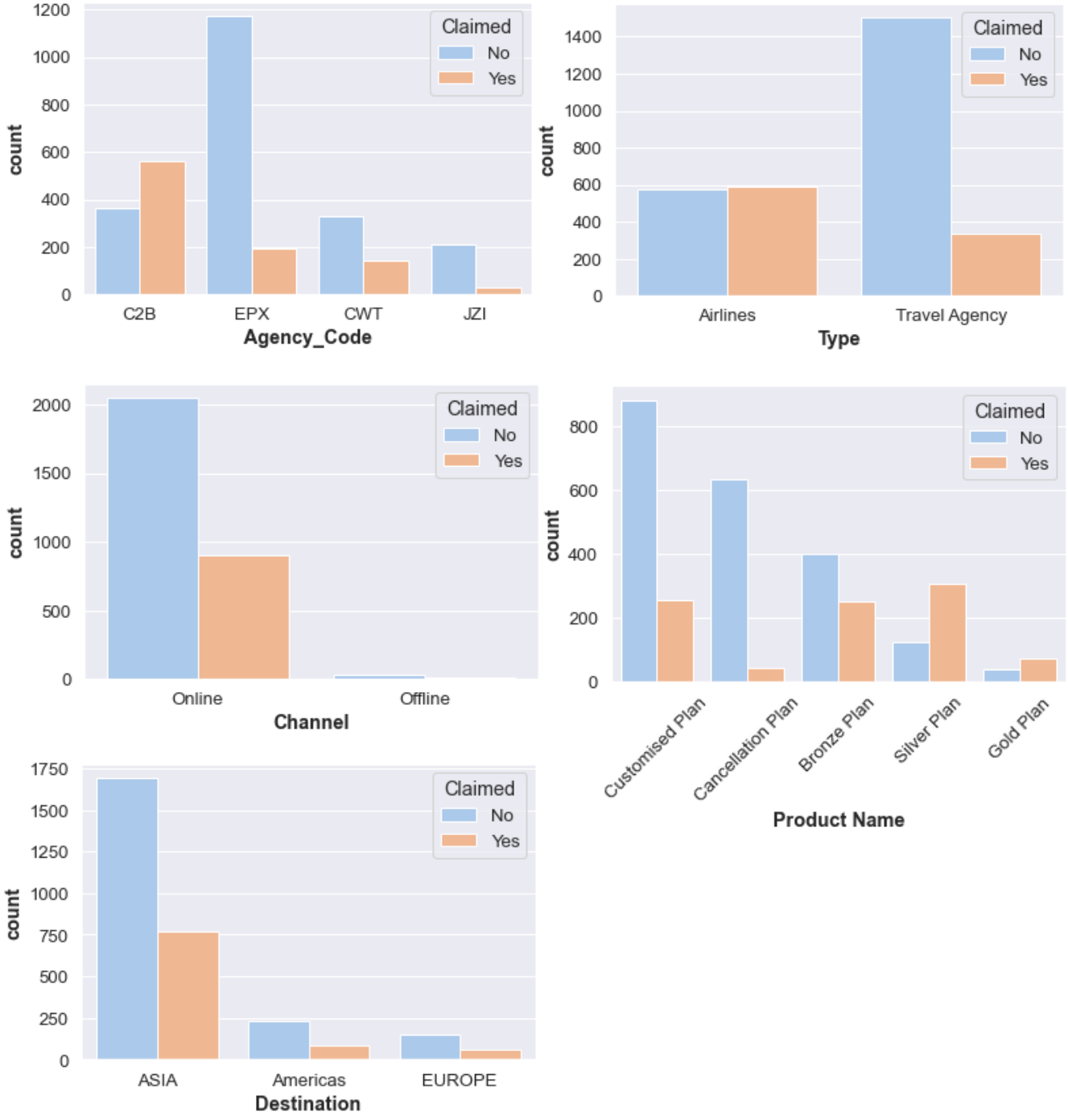


Figure 21: Distribution of Target over Categorical features(Problem 2)

(above figure generated with ‘.countplot()’ function of Seaborn library in Python)

We can see:

- target feature ‘Claimed’ has very few data points for class ‘Yes’(earlier in ‘Value_counts() also, we saw target feature ‘Claimed’ has 30.8% observations in class ‘Yes’)
- there is quite a high value of class ‘No’ in target feature ‘Claimed’ among ‘EPX’ class of feature ‘Agency_Code’
- there is quite a high value of class ‘No’ in target feature ‘Claimed’ among ‘Travel Agency’ class of feature ‘Type’
- there is quite a high value of class ‘No’ in target feature ‘Claimed’ among ‘ASIA’ class of feature ‘Destination’

Let us now check how the target ‘Claimed’ is distributed over the numeric features of the data:

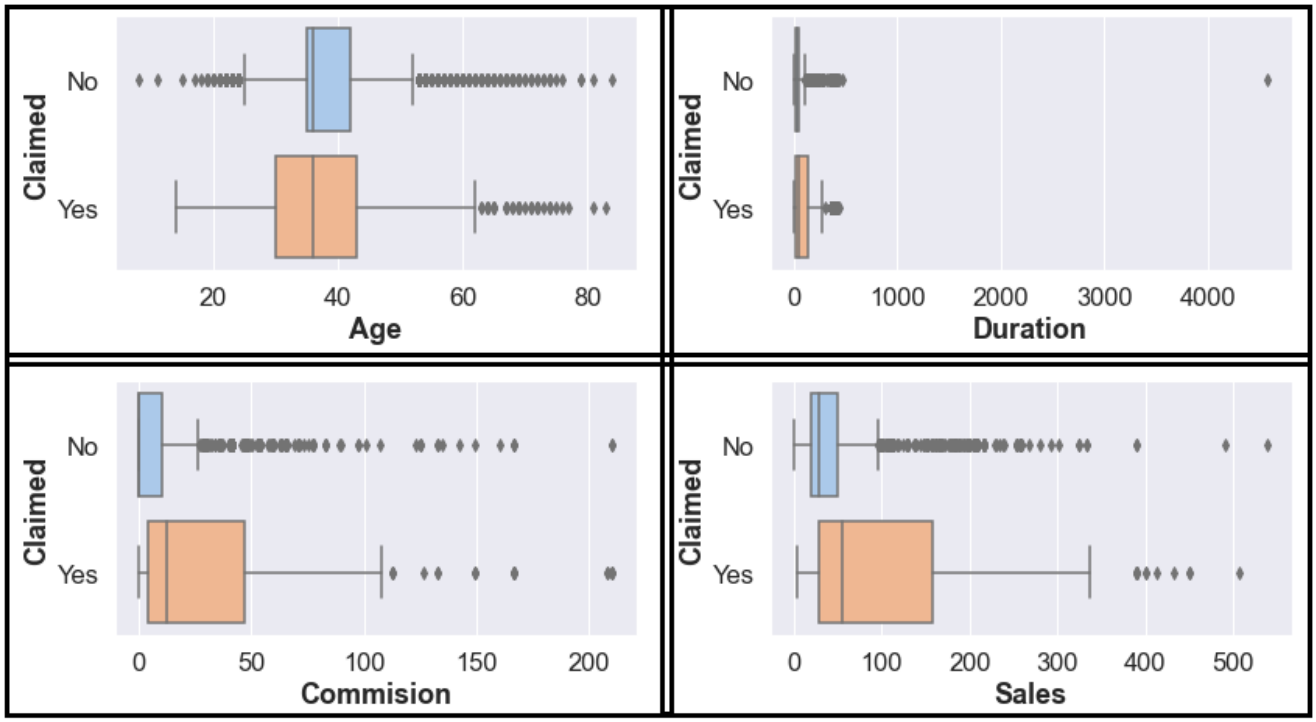


Figure 22: Distribution of Target over Numerical features(Problem 2)

(above figure generated with ‘.boxplot()’ function of Seaborn library in Python)

We can see:

- though class ‘Yes’ constitutes only 30.8% of target feature, yet the spread and range of class ‘Yes’ is more than that of class ‘No’ plotted against all numeric features
- a huge number of outliers in both classes of target feature ‘Claimed’ plotted against all numeric features
- there is one outlier that is really very far from rest of data points

Below we can see the feature ‘Sales’ versus target feature ‘Claimed’ hued over feature ‘Type’ and ‘Destination’ respectively:

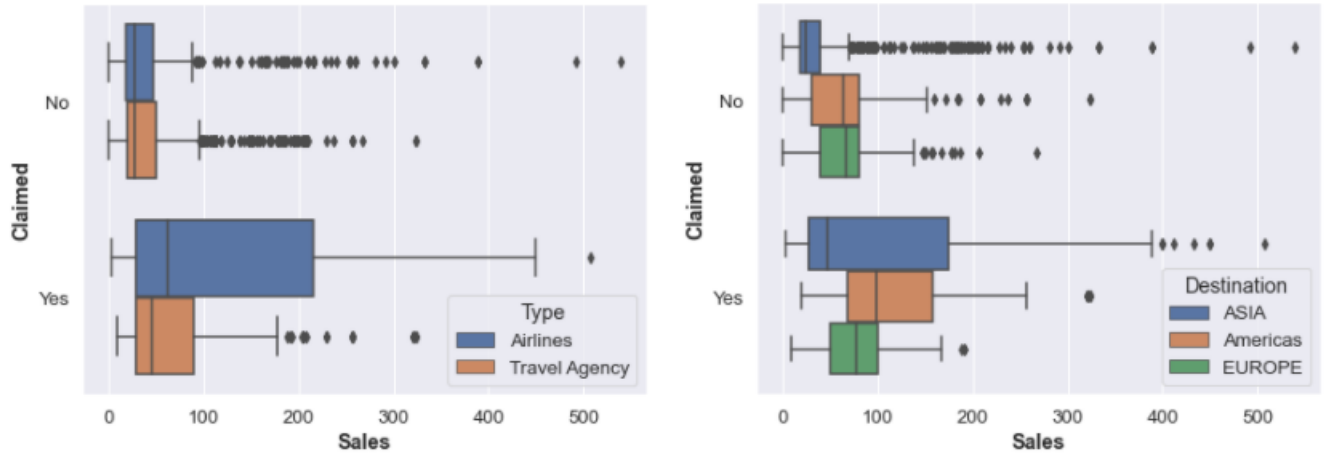


Figure 23: Multivariate Analysis(Problem 2)

(above figure generated with ‘.boxplot()’ function of Seaborn library in Python)

We can see:

- majority of claims in feature ‘Type’ comes from ‘Airlines’ class
- majority of claims in feature ‘Destination’ comes from ‘ASIA’ class

Let us now have a look at a facet grid of ‘Sales’ versus ‘Age’ hued over ‘Agency Code’ for target feature ‘Claimed’:

Figure 24: Facet Grid -1

(above figure generated with '.FacetGrid()' and '.map()' functions of Seaborn library in Python)

We can see:

- 'Cancellation Plan' has very few claims
- 'Silver Plan' has the best sales and claims
- there are very few purchases of 'Gold Plan'

Let us now have a look at a facet grid of 'Sales' versus 'Age' hued over 'Product Name' for target feature 'Claimed':

Figure 25: Facet Grid -2

(above figure generated with '.FacetGrid()' and '.map()' functions of Seaborn library in Python)

We can see:

- agency 'C2B' has the best sales and claims
- agency 'JZI' has the worst sales and claims

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

Before proceeding further, let us convert the object data types into categorical, and then further convert these categorical values into numerical codes. We will be using '.Categorical()' and '.codes()' functions of the PANDAS library in Python.

Let us have a look at the codes of the various classes of the different features in data set:

```
For feature " Agency_Code ":
['C2B', 'CWT', 'EPX', 'JZI']
[0, 1, 2, 3]

For feature " Type ":
['Airlines', 'Travel Agency']
[0, 1]

For feature " Claimed ":
['No', 'Yes']
[0, 1]

For feature " Channel ":
['Offline', 'Online']
[0, 1]

For feature " Product Name ":
['Bronze Plan', 'Cancellation Plan', 'Customised Plan', 'Gold Plan', 'Silver Plan']
[0, 1, 2, 3, 4]

For feature " Destination ":
['ASIA', 'Americas', 'EUROPE']
[0, 1, 2]
```

Let us have a look at general information of data after categorical conversion and coding:

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

Table 22: General Information of Data after Categorical conversion and coding(Problem 2)

(above table generated by '.info' of 'PANDAS' in Python)

We can see:

- 'object' data types earlier, are now 'integer'(8 bits) data types
- from 234.5 kb, the memory usage of the data set has come down to 111.5 kb after categorical conversion and coding.

We can go ahead and remove the feature 'Channel', which has 2 classes, out of which 98.47% of data points are in class 'Online', and remaining 1.53% of data points are in class 'Offline'. We can remove the feature using the '.drop()' function of the PANDAS library. Let us check the shape of data set before and after dropping the feature 'Channel', using the 'shape' function of the PANDAS library in Python:

Before Removing: (3000, 10)

After Removing: (3000, 9)

- As checked earlier with '.value_counts()' library of PANDAS library in Python, target feature 'Claimed' has 30.8% observations in class 'Yes', and remaining 69.2% observations in the class 'No'.
- We should next separate the target feature and the remaining data before proceeding further.
- From the 'model_selection' module of 'sklearn' library, we will use the 'train_test_split' function to split the data into training and testing sets. Separate labels for training and testing have been obtained from the target feature 'Claimed'.

- ‘test_size=.20’ and ‘test_size=.30’ were tried for all models. ‘test_size=.20’ gave better performance metrics for all the classification models. Hence, the data has been split into 80% training and 20% testing data with ‘test_size=.20’.
- Let us check the shape of the training and testing data and labels:

  X_train: (2400, 8)

  X_test: (600, 8)

  train_labels: (2400,)

  test_labels: (600,)

Let us now go ahead and build different classification models and check there performances by various techniques:

Decision Tree(CART model)

- For Hyper parameter tuning, we will do a grid search. From the ’model_selection’ module of ‘sklearn’ library, we will use the ‘GridSearchCV’ function to estimate the best parameters for our Decision Tree model.
- We will be passing a parameters grid for the grid search. The following is the parameter grid:

  'criterion': ['gini','entropy']

  'max_depth': [6,7,8,9,10,11,12,13,14,20,25]

  'min_samples_leaf': [25,100,150]

  'min_samples_split': [100,50,300,150,450]

  - Usually, so many parameters in one grid is not recommended, as it will burden the kernel and take longer time to process. But as the data size is small, we can experiment with more values in one go.
  - Gini criterion always gives better results.
  - The industry standard for 'max_depth' is usually 10-15, but as the data set is small, we can experiment with more values.
  - The industry standard for 'min_samples_split’ is 2%-3% of the original data(in our case we have 3000 entries. Again, as data size is small, we can experiment with more values)
  - The industry standard for 'min_samples_leaf' is usually one-third of 'min_samples_split’
- We will be using 10 fold cross validation, and ‘recall’ as scoring policy.
  - 10 fold cross validation: The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size.

The first fold is treated as a validation set, and the method is fit on the remaining (k − 1) folds. The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller.

- ❖ 'Recall' is being used as the scoring policy in the model. Scoring policy is used to evaluate the predictions on the test set. A high recall would indicate a low type 2 error, i.e, lower false negatives. False negatives are the cases where the claim has been made(insurance firm has paid) and the model predicts not claimed. It will not be very convenient if we already paid an amount and our records show that we have not paid. In this business context, optimising the recall values would be better than optimising the precision values.

- 'DecisionTreeClassifier' function of the 'tree' module of 'sklearn' library is being used to build the decision tree model.
- 'random_state' is being provided as 1 to obtain consistent output.
- The following parameters were the best parameters obtained from the grid search operation extracted using '.best_params_':
  'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 25, 'min_samples_split': 300

The following is the decision tree obtained using 'export_graphviz' function of the 'tree' module of 'sklearn' library, visualised in an online graphical editor, 'http://webgraphviz.com/':

(zoom to at least 150% for better visibility)

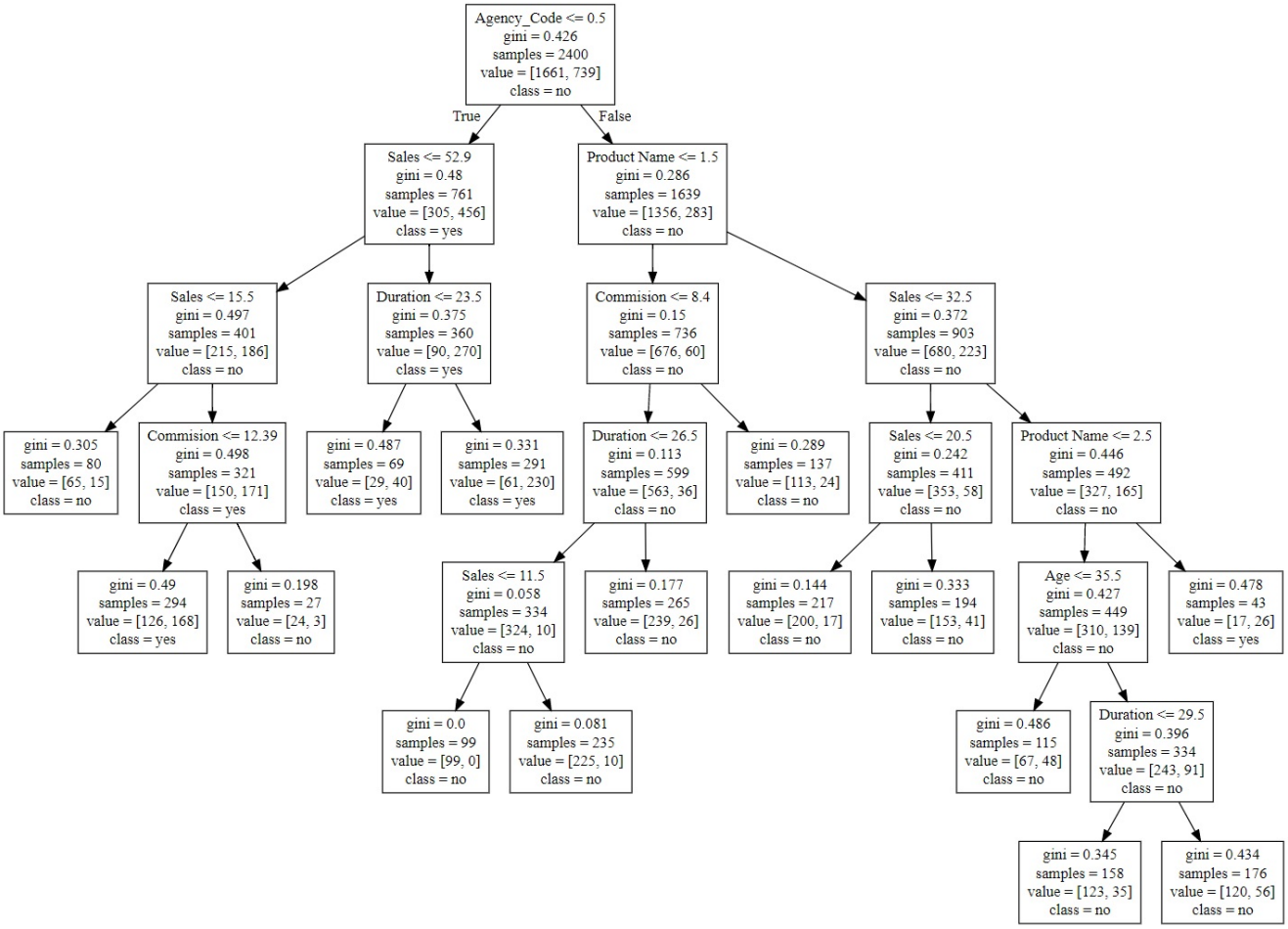
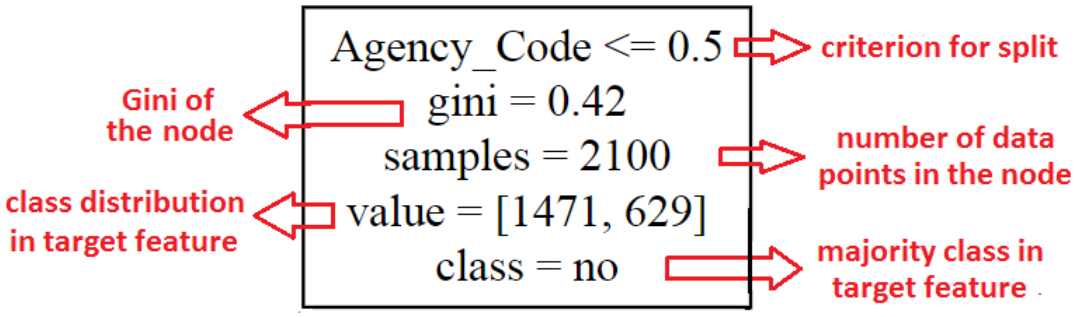


Figure 26: Decision Tree

(above tree generated using Python and 'http://webgraphviz.com/')

We can see the decision tree is pruned rightly.

Below is an example of how to infer a node in above decision tree:



- For each node we can see the feature based on which the split has taken place, Gini of the node, number of data points in the node, number of data points for each class in target feature, and the majority class in the target feature.

- The first split has taken place with respect to the feature 'Agency_Code'.

Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. The higher the value the more important the feature.

Let us now have a look at the different variable importances in the decision tree:

Feature	**Importance**
Agency_Code	0.609733
Sales	0.216365
Product Name	0.093596
Commision	0.043312
Duration	0.025407
Age	0.011588
Type	0.000000
Destination	0.000000

Table 23: Feature Importances(Decision Tree)

(above values extracted using '.feature_importances_' of 'sklearn' library in Python)

Inferences:

- Feature or variable importance is the overall proportion or percentage contribution in the Gini gain in the decision tree(CART) model.
- 'Agency_Code' has the highest feature importance of all.
- 'Agency_Code' and 'Sales' have feature importances of 60.97% and 21.64% respectively, together holding 82.61% of the total feature importances.
- 'Product Name' has a feature importance of 9.36%.
- 'Commision', 'Duration' and 'Age' have feature importances of 4.33%, 2.54% and 1.16% respectively.
- 'Type' and 'Destination' did not contribute to the splits in the Decision Tree at all.

We can next go ahead and predict for training and test sets using '.predict()'.

The probabilities of predicted classes can also be obtained by '.predict_proba()'. Probability threshold optimisation can be done if need arises.

Random Forest Classifier

- For Hyper parameter tuning, we will do a grid search. From the ’model_selection’ module of ‘sklearn’ library, we will use the ‘GridSearchCV’ function to estimate the best parameters for our Random Forest model.
- We will be passing a parameters grid for the grid search. The following is the parameter grid:

  'max_depth': [8,10,12,15,20,25]

  'max_features': [4,5,6]

  'min_samples_leaf': [35,50,100]

  'min_samples_split': [60,70,100]

  'n_estimators': [101,151,201]

  - ❖ Usually, so many parameters in one grid is not recommended, as it will burden the kernel and take longer time to process. But as the data size is small, we can experiment with more values in one go.
  - ❖ The industry standard for 'max_depth' is usually 10-15, but as the data set is small, we can experiment with more values.
  - ❖ 'max_features' usually takes the minimum value as the square root of the total number of features, and maximum value as half of the number of features. We have a small data set and can go ahead and experiment if the result can be optimised for any other values(the number of features after dropping feature ‘Channel’ is 9).
  - ❖ The industry standard for 'min_samples_split’ is 2%-3% of the original data(in our case we have 3000 entries. Again, as data size is small, we can experiment with more values)
  - ❖ The industry standard for 'min_samples_leaf' is usually one-third of 'min_samples_split’
  - ❖ 'n_estimators’ is the number of trees in the Random Forest model
- We will be using 5 fold cross validation, and ‘recall’ as scoring policy.
  - ❖ 5 fold cross validation: The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining (k − 1) folds. The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller.
  - ❖ ‘Recall’ is being used as the scoring policy in the model. Scoring policy is used to evaluate the predictions on the test set. A high recall would indicate a low type 2 error, i.e, lower false negatives. False negatives are the cases where the claim has been made(insurance firm has paid) and the model predicts not claimed. It will not be very convenient if we already paid an amount and our records show that we have not paid.

In this business context, optimising the recall values would be better than optimising the precision values.

- 'RandomForestClassifier' function of the 'ensemble' module of 'sklearn' library is being used to build the decision tree model.
- 'random_state' is being provided as 1 to obtain consistent output.
- The following parameters were the best parameters obtained from the grid search operation extracted using '.best_params_':
  'max_depth': 8, 'max_features': 5, 'min_samples_leaf': 35, 'min_samples_split': 60, 'n_estimators': 151

Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. The higher the value the more important the feature.

Let us now have a look at the different variable importances in the random forest:

Feature	**Importance**
Agency_Code	0.427460
Product Name	0.200882
Sales	0.169103
Commision	0.087551
Duration	0.061165
Age	0.035523
Type	0.013153
Destination	0.005164

Table 24: Feature Importances(Random Forest)

(above values extracted using '.feature_importances_' of 'sklearn' library in Python)

Inferences:

- 'Agency_Code' has the highest feature importance of all(42.75%).
- 'Product Name' and 'Sales' have a feature importance of 20.09% and 16.91% respectively.
- 'Agency_Code', 'Product Name' and 'Sales ' together hold 79.75% of the total feature importances.
- 'Destination' has a very low feature importance of 0.516%.

We can next go ahead and predict for training and test sets using '.predict()'.

The probabilities of predicted classes can also be obtained by '.predict_proba()'. Probability threshold optimisation can be done if need arises.

Artificial Neural Network

- As Artificial Neural Networks are weight based, scaling of features is a necessity before proceeding further.
    - ❖ We can scale the data using the ‘StandardScaler’ function of the ‘preprocessing’ module of ‘sklearn’ library.
    - ❖ We do a fit and transform operation for the train data and only transform operation for the test data. We will be using the mean and standard deviations of the train data only to transform the test data.
    - ❖ ‘fit_transform()’ is applied to the train data and ‘.transform()’ is applied to the test data.
- For Hyper parameter tuning, we will do a grid search. From the ’model_selection’ module of ‘sklearn’ library, we will use the ‘GridSearchCV’ function to estimate the best parameters for our Random Forest model.
- We will be passing a parameters grid for the grid search. The following is the parameter grid:

  'hidden_layer_sizes': [3,4,5,6,7,8,9,10,11,12,15,20,50,100] - for 1st run

  [4,70,72,75,79,81,83,87,90] - for 2nd run

  [4,[4,4],[5,5],[4,5],[5,4]] - for 3rd run

  [4,[4,80],[82,78],[78,82],[80,4],[5,3,2]] - for 4th run

  'max_iter': [2000,2500,3000,5000]

  'solver': ['adam', 'sgd']

  'tol': [0.001, 0.0001]

    - ❖ Usually, so many parameters in one grid is not recommended, as it will burden the kernel and take longer time to process. But as the data size is small, we can experiment with more values in one go.
    - ❖ For smaller data sets, 'hidden_layer_sizes' should have one hidden layer with approximately neurons equal to half the number of features. Multiple hidden layers make the smaller data sets tend to over fit the Neural Network model.
    - ❖ Some industry standards recommend 'hidden_layer_sizes' to have neurons equal to square or square root of no. of independent features.
    - ❖ We can try values for both the recommendations in multiple runs, as loading all values into one grid might over-burden the kernel and take longer time to process.
    - ❖ ‘max_iter’ is the maximum number of times the weights can be updated if the loss does not converge.
    - ❖ For 'solver', we pass 'adam'(Adam Solver) and 'sgd'(Stochastic Gradient descent Solver). Usually, Adam solver gives better results than Stochastic Gradient descent Solver.

- ❖ 'tol' is the learning rate or the estimated time of arrival(ETA). Higher the learning rate, lesser the accuracy and the processing time also. So, an optimal value needs to be passed to have a good compromise between the accuracy and processing time.

- We will be using 5 fold cross validation, and 'recall' as scoring policy.
  - ❖ 5 fold cross validation: The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining (k − 1) folds. The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller.
  - ❖ 'Recall' is being used as the scoring policy in the model. Scoring policy is used to evaluate the predictions on the test set. A high recall would indicate a low type 2 error, i.e, lower false negatives. False negatives are the cases where the claim has been made(insurance firm has paid) and the model predicts not claimed. It will not be very convenient if we already paid an amount and our records show that we have not paid. In this business context, optimising the recall values would be better than optimising the precision values.

- 'MLPClassifier' function of the 'neural_network' module of 'sklearn' library is being used to build the neural network model.
- 'random_state' is being provided as 1 to obtain consistent output.
- The following parameters were the best parameters obtained from the grid search operation extracted using '.best_params_':
  'hidden_layer_sizes': 4, 'max_iter': 2500, 'solver': 'adam', 'tol': 0.001
  Despite multiple tries of hidden layer sizes with different numbers of neurons in the grid search operation, a single hidden layer of 4 neurons gives the best result in the grid search operation, indicating an Artificial Neural Network model unsuitable for such a low number of input features.

Due to the classic problem of interpreting how model weights contribute towards classification decisions, we can not calculate the feature importances for the Artificial Neural Network Model

We can next go ahead and predict for training and test sets using '.predict()'.

The probabilities of predicted classes can also be obtained by '.predict_proba()'. Probability threshold optimisation can be done if need arises.

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

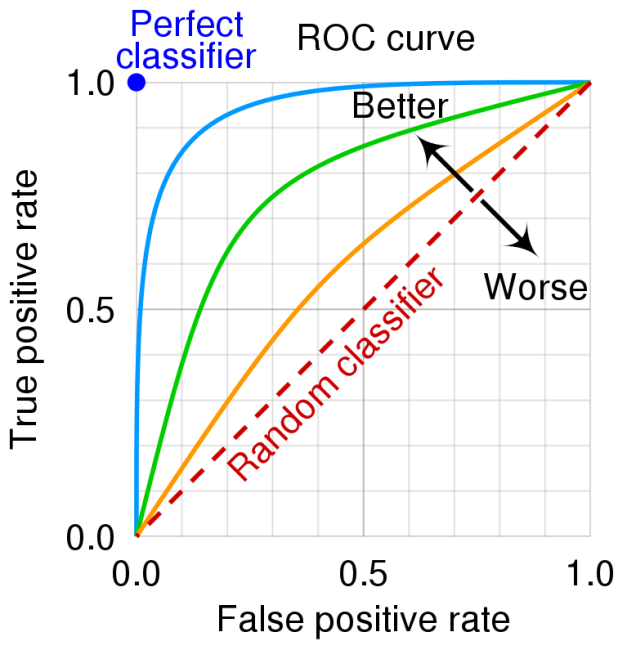
- Classification accuracy is the ratio of correct predictions to total predictions made.
- A confusion matrix is a tabular summary of the number of correct and incorrect predictions made by a classifier. It can be used to evaluate the performance of a classification model through the calculation of performance metrics like accuracy, precision, recall, and F1-score.

		Predicted	
		Negative (**N**) **-**	Positive (**P**) **+**
Actual	Negative **-**	True Negatives (T**N**)	False Positives (F**P**) **Type I error**
	Positive **+**	False Negatives (F**N**) **Type II error**	True Positives (T**P**)

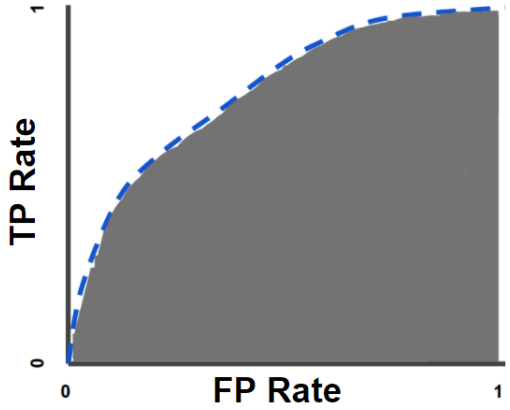
- An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:
  - ❖ True Positive Rate(TPR)
  - ❖ False Positive Rate(FPR)

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}$$

- An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows few ROC curves:

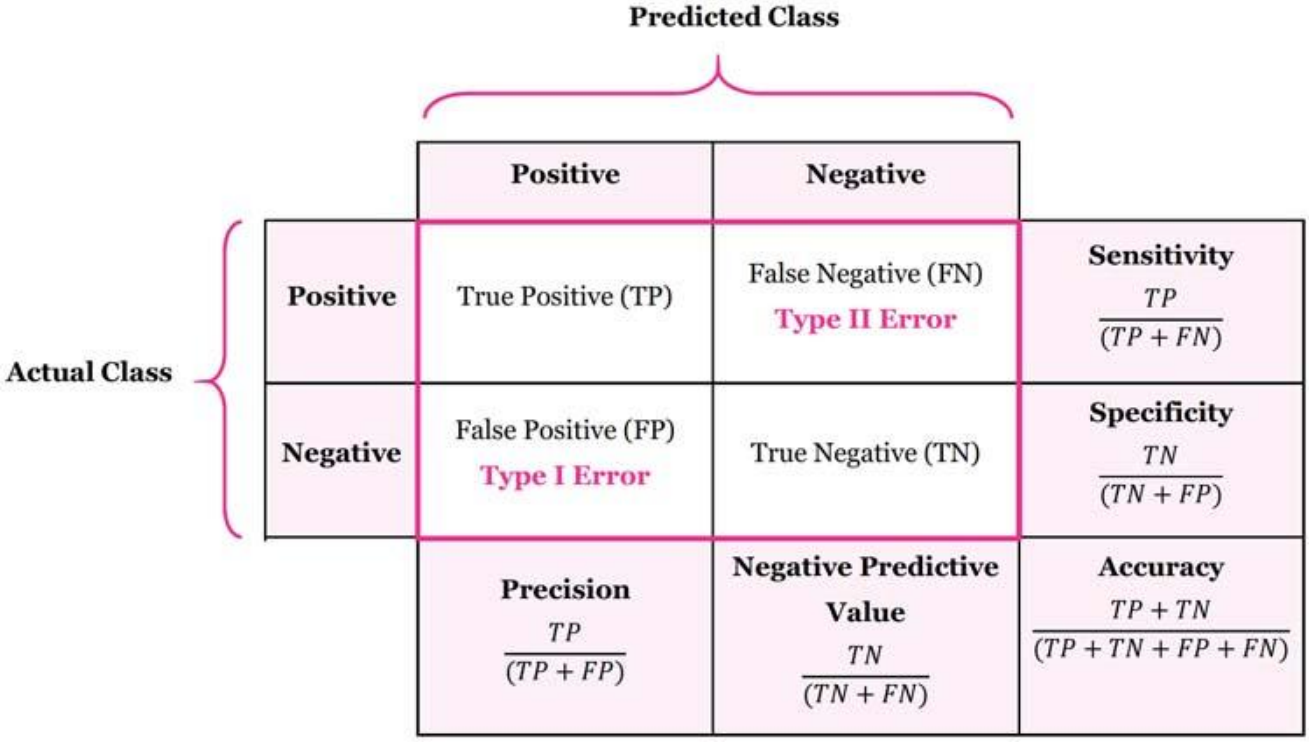


- AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1). The grey shaded portion in the below ROC curve represents the AUC for it:



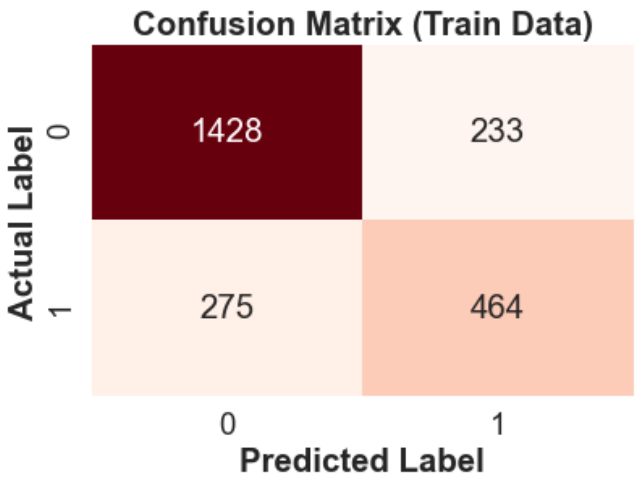
- A classification report is a performance evaluation metric in machine learning. It is used to show the precision, recall, F1 Score, and support of any trained classification model.It provides a better understanding of the overall performance of our trained model. To understand the classification report of a machine learning model, following are the metrics displayed in the report:
    - ❖ **Precision:** Precision is defined as the ratio of true positives to the sum of true and false positives. A model that produces no false positives has a precision of 1.0.

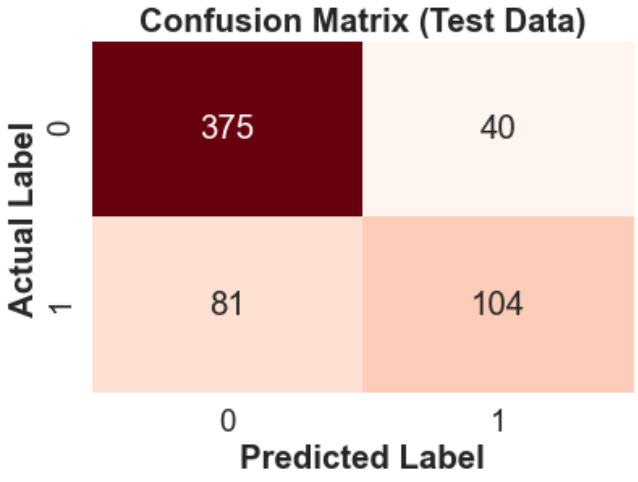
- ❖ **Recall(or Sensitivity):** Recall is defined as the ratio of true positives to the sum of true positives and false negatives. A model that produces no false negatives has a recall of 1.0.
- ❖ **F1-score:** The F1 is the weighted harmonic mean of precision and recall. The closer the value of the F1 score is to 1.0, the better the expected performance of the model is.
- ❖ **Support:** Support is the number of actual occurrences of the class in the dataset. It doesn't vary between models, it just diagnoses the performance evaluation process.



- 'Recall' is being used as the scoring policy in all the classification models. Scoring policy is used to evaluate the predictions on the test set. A high recall would indicate a low type 2 error, i.e, lower false negatives. False negatives are the cases where the claim has been made(insurance firm has paid) and the model predicts not claimed. It will not be very convenient if we already paid an amount and our records show that we have not paid. In this business context, optimising the recall values would be better than optimising the precision values.

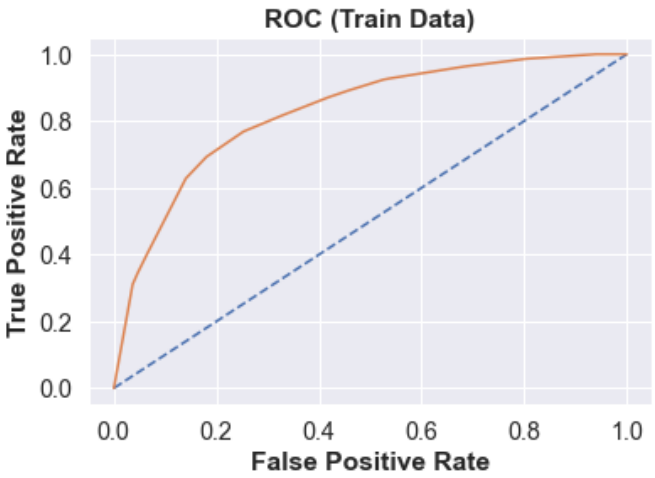
Decision Tree Classifier





Train Accuracy: 78.83%

Test Accuracy: 79.83%



Train AUC: 0.828

Test AUC: 0.807

Training Classification Report:

	precision	recall	f1-score	support
0	0.84	0.86	0.85	1661
1	0.67	0.63	0.65	739
accuracy			0.79	2400
macro avg	0.75	0.74	0.75	2400
weighted avg	0.79	0.79	0.79	2400

Testing Classification Report:

	precision	recall	f1-score	support
0	0.82	0.90	0.86	415
1	0.72	0.56	0.63	185
accuracy			0.80	600
macro avg	0.77	0.73	0.75	600
weighted avg	0.79	0.80	0.79	600

Figure 27: CART Metrics

(confusion matrix, figures, metrics and reports generated with 'roc_auc_score', 'roc_curve', 'classification_report', 'confusion_matrix', 'accuracy_score' functions of 'metrics' module of 'sklearn' library using Python)

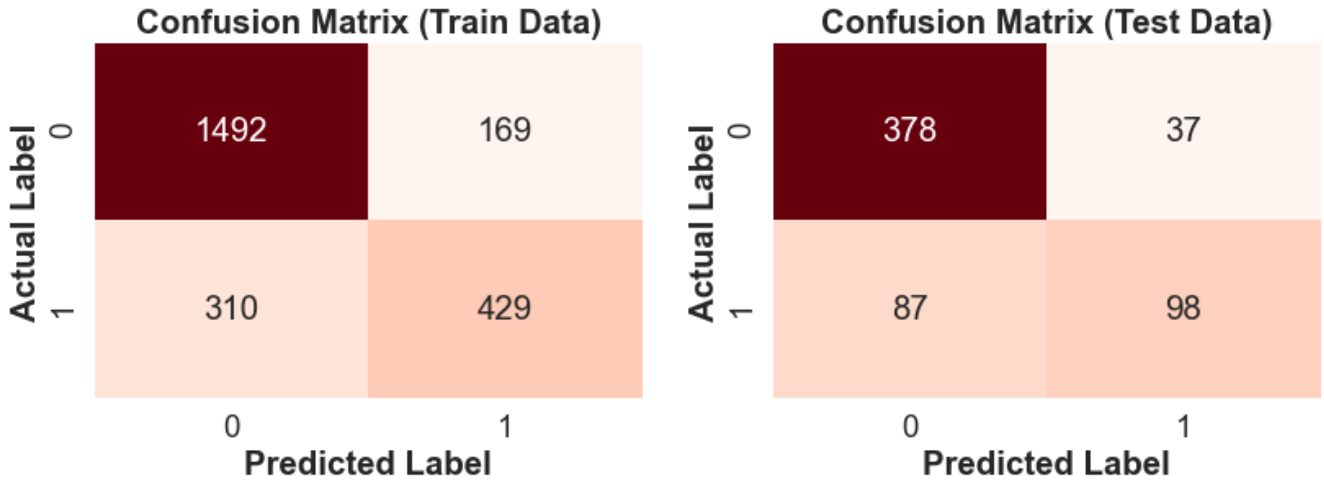
Train Data:
AUC: 82.8%
Accuracy: 78.83%
Precision(positive class): 67%
Recall(positive class): 63%
f1-Score(positive class): 65%

Test Data:
AUC: 80.7%
Accuracy:79.83%
Precision(positive class): 72%
Recall(positive class): 56%
f1-Score(positive class): 63%

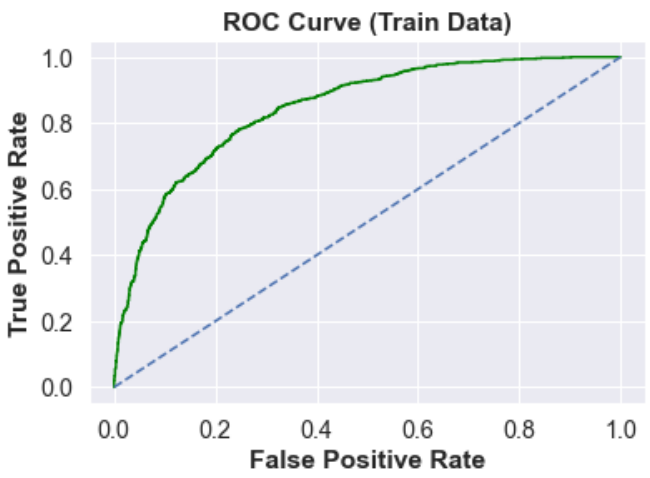
Inferences:

- Training and Test set results are almost similar, and with the overall measures at average.
- Model does not seem to have over-fitted or under-fitted as the metrics are consistent in training as well as testing data.
- As the target feature 'Claimed' has 30.8% observations in class 'Yes', and remaining 69.2% observations in the class 'No', we see the metrics for the positive class to be relatively lower than that of the negative class in the predictions as well.
- The false positives during training were 33%(1-Precision), while during testing there were false positives 28% of the time(1-Precision).
- The false negatives during training were 37%(1-Recall), while during testing, false negatives occurred 44%(1-Recall) of the time.
- Though scoring policy used while building models were for recall, a decent f1-score indicates a good balance between false positives and false negatives.
- The model is an average model.

Random Forest



Train Accuracy: 80.04%



Train AUC: 0.846

Training Classification Report:

	precision	recall	f1-score	support
0	0.83	0.90	0.86	1661
1	0.72	0.58	0.64	739
accuracy			0.80	2400
macro avg	0.77	0.74	0.75	2400
weighted avg	0.79	0.80	0.79	2400

Train Data:

AUC: 84.6%

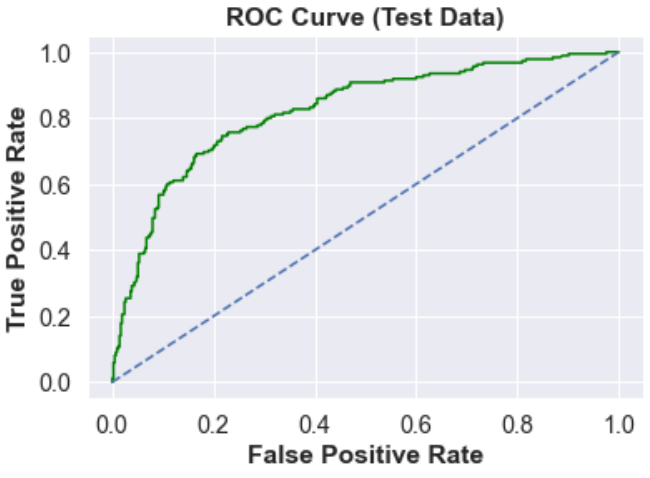
Accuracy: 80.04%

Precision(positive class): 72%

Recall(positive class): 58%

f1-Score(positive class): 64%

Test Accuracy: 79.33%



Test AUC: 0.822

Testing Classification Report:

	precision	recall	f1-score	support
0	0.81	0.91	0.86	415
1	0.73	0.53	0.61	185
accuracy			0.79	600
macro avg	0.77	0.72	0.74	600
weighted avg	0.79	0.79	0.78	600

Test Data:

AUC: 82.2%

Accuracy:79.33%

Precision(positive class): 73%

Recall(positive class): 53%

f1-Score(positive class): 61%

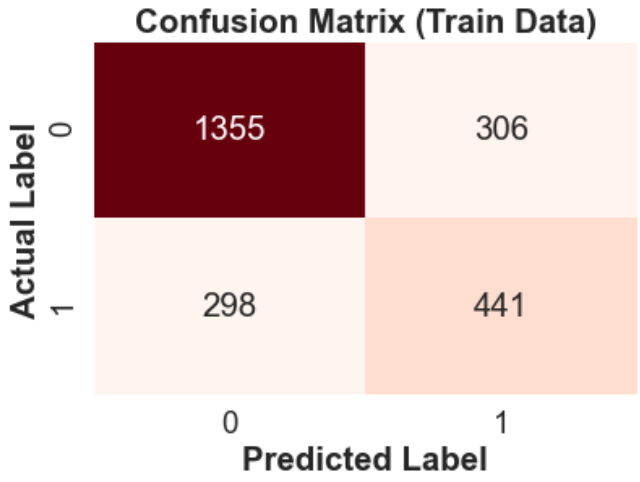
Figure 28: Random Forest Metrics

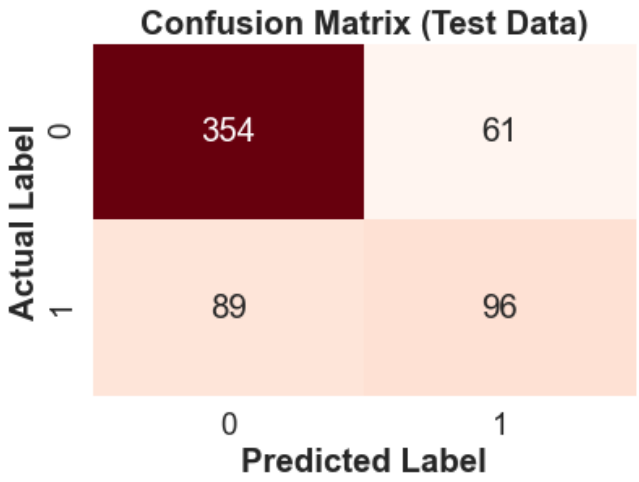
(confusion matrix, figures, metrics and reports generated with 'roc_auc_score', 'roc_curve', 'classification_report', 'confusion_matrix', 'accuracy_score' functions of 'metrics' module of 'sklearn' library using Python)

Inferences:

- Training and Test set results are almost similar, and with the overall measures at average.
- Model does not seem to have over-fitted or under-fitted as the metrics are consistent in training as well as testing data.
- As the target feature ‘Claimed’ has 30.8% observations in class ‘Yes’, and remaining 69.2% observations in the class ‘No’, we see the metrics for the positive class to be relatively lower than that of the negative class in the predictions as well.
- The false positives during training were 28%(1-Precision), while during testing there were false positives 27% of the time(1-Precision).
- The false negatives during training were 42%(1-Recall), while during testing, false negatives occurred 47%(1-Recall) of the time.
- Though scoring policy used while building models were for recall, a decent f1-score indicates a good balance between false positives and false negatives.
- The model is an average model.

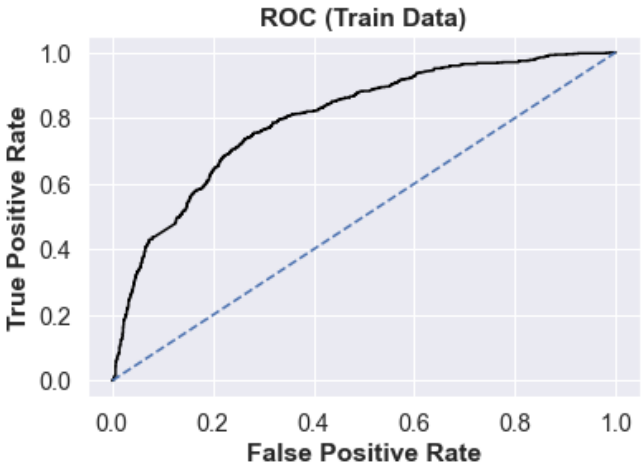
Artificial Neural Network





Train Accuracy: 74.83%

Test Accuracy: 75%



Train AUC: 79.95%

Test AUC: 79.81

Training Classification Report:

	precision	recall	f1-score	support
0	0.82	0.82	0.82	1661
1	0.59	0.60	0.59	739
accuracy			0.75	2400
macro avg	0.71	0.71	0.71	2400
weighted avg	0.75	0.75	0.75	2400

Testing Classification Report:

	precision	recall	f1-score	support
0	0.80	0.85	0.83	415
1	0.61	0.52	0.56	185
accuracy			0.75	600
macro avg	0.71	0.69	0.69	600
weighted avg	0.74	0.75	0.74	600

Train Data:
AUC: 79.95%
Accuracy: 74.83%
Precision(positive class): 59%
Recall(positive class): 60%
f1-Score(positive class): 59%

Test Data:
AUC: 79.81%
Accuracy:75%
Precision(positive class): 61%
Recall(positive class): 52%
f1-Score(positive class): 56%

Figure 29: Artificial Neural Network Metrics

(confusion matrix, figures, metrics and reports generated with 'roc_auc_score', 'roc_curve', 'classification_report', 'confusion_matrix', 'accuracy_score' functions of 'metrics' module of 'sklearn' library using Python)

Inferences:

- Training and Test set results are almost similar, and with the overall measures at average.
- Model does not seem to have over-fitted or under-fitted as the metrics are consistent in training as well as testing data.

- As the target feature ‘Claimed’ has 30.8% observations in class ‘Yes’, and remaining 69.2% observations in the class ‘No’, we see the metrics for the positive class to be relatively lower than that of the negative class in the predictions as well.
- The false positives during training were 41%(1-Precision), while during testing there were false positives 39% of the time(1-Precision).
- The false negatives during training were 40%(1-Recall), while during testing, false negatives occurred 48%(1-Recall) of the time.
- Though scoring policy used while building models were for recall, a decent f1-score indicates a good balance between false positives and false negatives.
- The model is an average model.

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

To compare the classification models, let us view a table containing all the values of accuracies, precisions, recalls, AUCs, f1 scores:

	CART Train	CART Test	Random Forest Train	Random Forest Test	Neural Network Train	Neural Network Test
Accuracy	0.79	0.80	0.80	0.79	0.75	0.75
AUC	0.83	0.81	0.85	0.82	0.80	0.80
Recall	0.63	0.56	0.58	0.53	0.60	0.52
Precision	0.67	0.72	0.72	0.73	0.59	0.61
F1 Score	0.65	0.63	0.64	0.61	0.59	0.56

Table 25: Performance Metrics Comparison

(above values extracted using sklearn and PANDAS libraries in Python)

<u>Accuracy</u>

- Accuracy speaks about the proportion of correct mapping made by the prediction model.
- Whilst accuracy score is fine for balanced classes,it can be very misleading for unbalanced classes.

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

- We can see the accuracies for CART and Random Forest are nearly the same for both training and testing data, and that for the Neural network is slightly lower than the former two for both training and testing data.

<u>AUC</u>

- AUC measures the entire two-dimensional area underneath the entire ROC curve
- AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

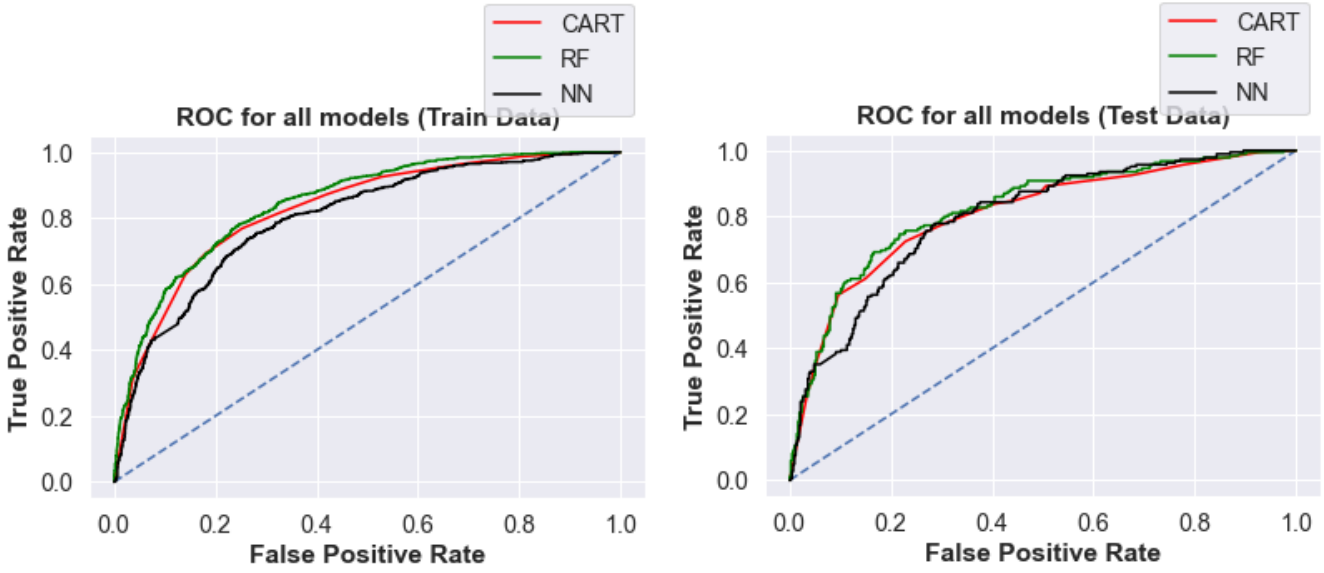


Figure 30: ROC AUC Comparison

(plots generated using 'roc_curve' function of 'metrics' module of 'sklearn' library using Python)

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.
- For training data, we see Random Forest has the highest AUC(0.85) amongst the 3 models.
- Again, for test data also, we see Random Forest has the highest AUC(0.82) amongst the 3 models.

Recall

- Recall tells what proportion of actual positives was identified correctly.
- A model that produces no false negatives has a recall of 1.0.

$$Recall = \frac{TP}{TP + FN} \text{ or } \frac{\text{True Positive}}{\text{Actual Results}}$$

- For training data, CART, Random Forest and Neural network have a recall of 0.63, 0.58 and 0.60 respectively, which means the proportion of false negatives during training for CART, Random Forest and Neural network were 0.37, 0.42 and 0.40 respectively.

- For testing data, CART, Random Forest and Neural network have a recall of 0.56, 0.53 and 0.52 respectively, which means the proportion of false negatives during testing for CART, Random Forest and Neural network were 0.44, 0.47 and 0.48 respectively.

Precision

- Precision tells what proportion of positive identifications was actually correct.
- A model that produces no false positives has a precision of 1.0.

$$\textit{Precision} = \frac{\text{TP}}{\text{TP + FP}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{Predictive Results}}$$

- For training data, CART, Random Forest and Neural network have a precision of 0.67, 0.72 and 0.59 respectively, which means the proportion of false positives during training for CART, Random Forest and Neural network were 0.33, 0.28 and 0.41 respectively.
- For testing data, CART, Random Forest and Neural network have a precision of 0.72, 0.73 and 0.61 respectively, which means the proportion of false positives during testing for CART, Random Forest and Neural network were 0.28, 0.27 and 0.39 respectively.

F1-score

- It is difficult to compare two models with different Precision and Recall. So to make them comparable, we use F1-Score.
- It is the Harmonic Mean of Precision and Recall.

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

- As compared to Arithmetic Mean, Harmonic Mean punishes the extreme values more.
- F1-score should be high.
- For training data, CART has an f1-score of 0.65 and random forest has an f1-score of 0.64 which are nearly comparable. Neural net has an f1-score of 0.59.
- Again, for testing data, CART has an f1-score of 0.63 and random forest has an f1-score of 0.61 which are nearly comparable. Neural net has an f1-score of 0.56.

Comparison

- Accuracy can be a useful measure if we have the same amount of samples per class but if we have an imbalanced set of samples accuracy isn't useful at all. Even more so, a test can have a high accuracy but actually perform worse than a test with a lower accuracy.
- Accuracies for CART and Random forest are nearly the same, and better than Neural Net.

- For imbalanced classification with a severe skew and few examples of the minority class, the ROC AUC can be misleading. This is because a small number of correct or incorrect predictions can result in a large change in the ROC Curve or ROC AUC score.
- AUC for random forest is the best for both train and test data, and poorest for neural net.
- For recall values, the CART model has best recall for both train and test data, and random forest has poorest recall for both train and test data.
- For precision values, random forest has done best for both train and test data, and neural net has done worst.
- For f1-score, the CART model has best values for both train and test data, and neural net the worst.

Conclusion

We see most of the metrics indicate random forest and the CART model to be better than neural net.

Also, most of the metrics are comparable for CART and Random Forest. But, we can go ahead with the random forest model, because of following advantages of random forests over decision trees:

- Randomness and voting mechanisms
- Trees are unpruned(unless specified)
- Trees are diverse
- Handling overfitting

Though, our focus was on recall, and random forest has poorest recall, but still we can go ahead with random forest, as the recall values are not much behind other models. Also, it won't be wise to neglect all other metrics for recall alone, when the recall values are comparable enough.

We can also see artificial neural net has given a very poor performance, majorly because of data-greediness. The data set is very small, and the number of features that were fed to the neural net were only 9.

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

The insurance firm is facing high claims. Let us see if we may draw some insights that would help the firm address the problem.
Let us visualise how the target feature 'Claimed' is distributed over other categorical variables:

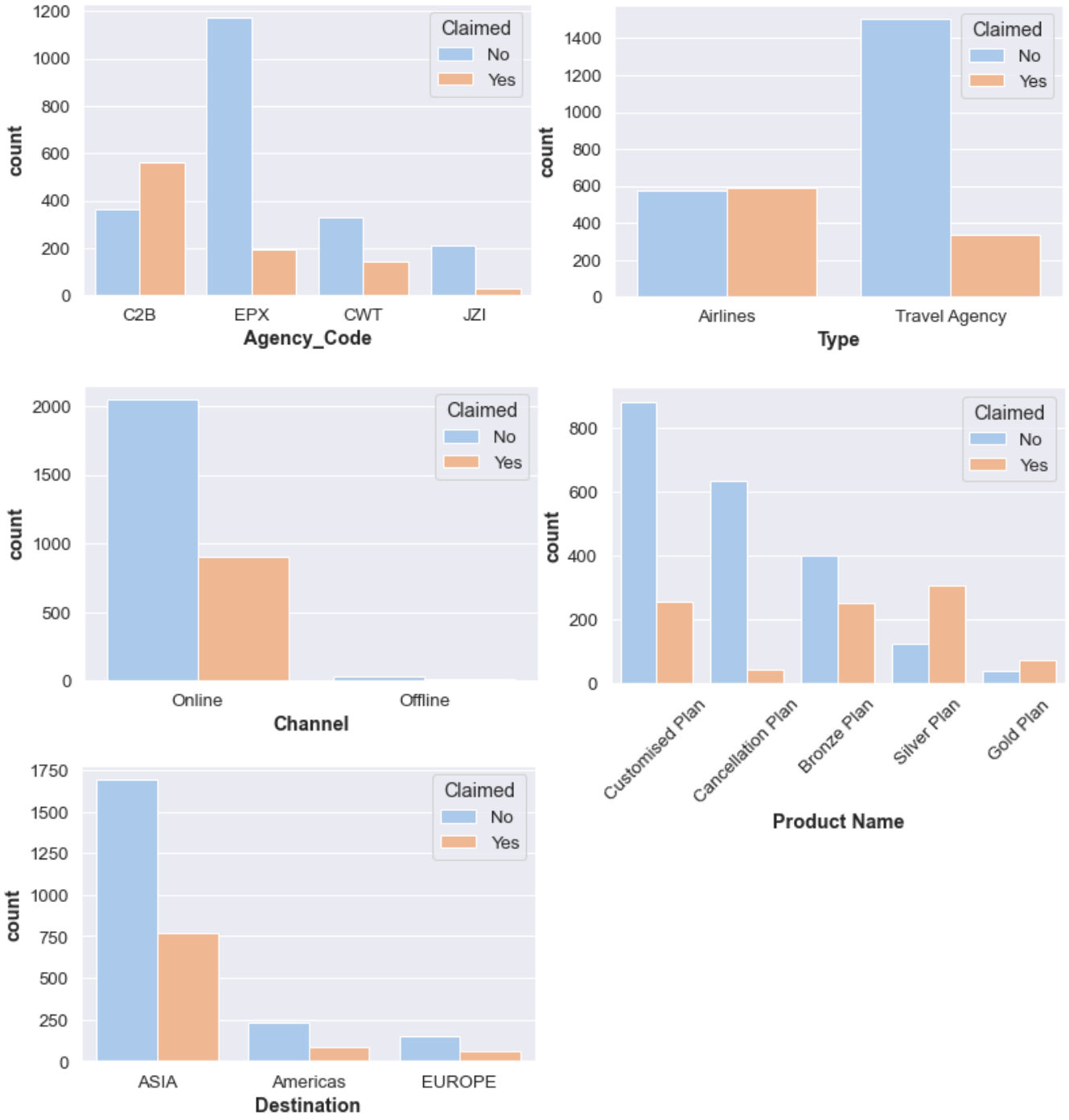


Figure 31: Distribution of target feature over categorical features

(plots generated .countplot function of Seaborn library using Python)

'Agency_Code'

- We see the tour firm 'C2B' is having a very high proportion of claims. Some kind of business restructuring needs to be done in this firm.

- The tour firm ‘EPX’ is doing the best, with a very low proportion of claims. The business strategies from this firm could be implemented across other firms as well, after generalisation, if needed.

‘Type’

Claims in ‘Airline’ are almost 50% of the total in the segment, indicating the need for improvisation.

‘Product Name’

- ‘Silver Plan’ has quite high claims(more than twice not claimed).
- ‘Bronze Plan’ also has quite high claims
- These two plans can be changed or discontinued, as these are not the most popular plans either(popular plans being ‘Customized Plan’ and ‘Cancellation Plan’)

➢ Let us see how the target feature is distributed over the feature ‘Age’ through a boxplot:

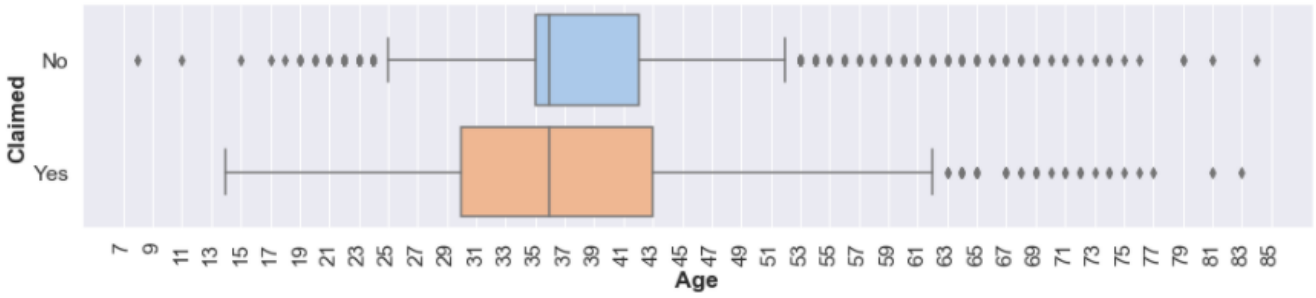


Figure 32: Boxplot of target feature and ‘Age’

(plots generated .boxplot function of Seaborn library using Python)

We can see the insured people who are in the age bracket of 30 to 35 are definitely making claims, indicating some kind of wrong doings. Perhaps this younger age group, found some loophole kind of thing, and are making fraudulent claims. A closer inspection needs to be done, or maybe some kind of reforms or new plans for this age group is required.

➢ Below we can see the feature ‘Sales’ versus target feature ‘Claimed’ hued over feature ‘Destination’:

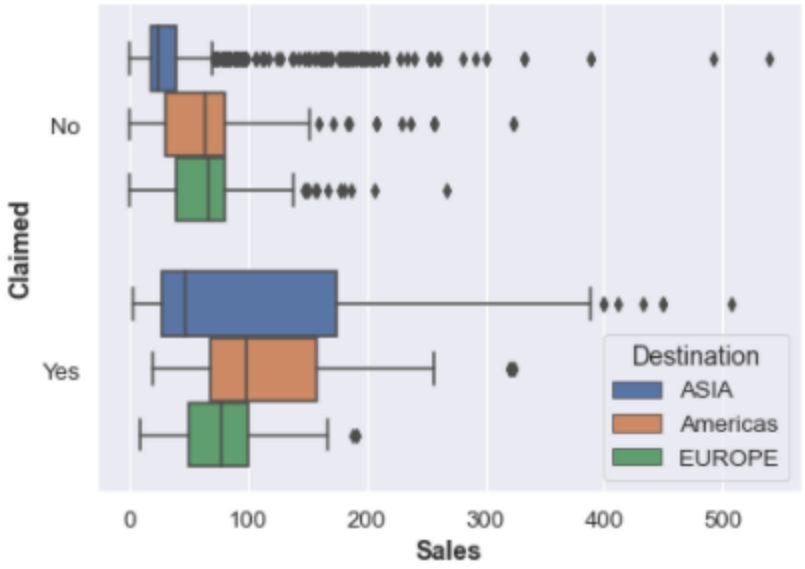


Figure 33: Boxplot of target feature and 'Sales' hued over 'Destination'

(plots generated .boxplot function of Seaborn library using Python)

The claims by the people traveling to Asia is quite high(almost 5 to 6 times that of not claimed). The claims by the people traveling to Europe are the same as not claimed. Some kind of generalization of business strategies and insurance plans from the latter case needs to apply to the former, so that the former gives similar results.

➢ Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. The higher the value the more important the feature.

Let us have a look at the feature importances of our chosen model, Random Forest:

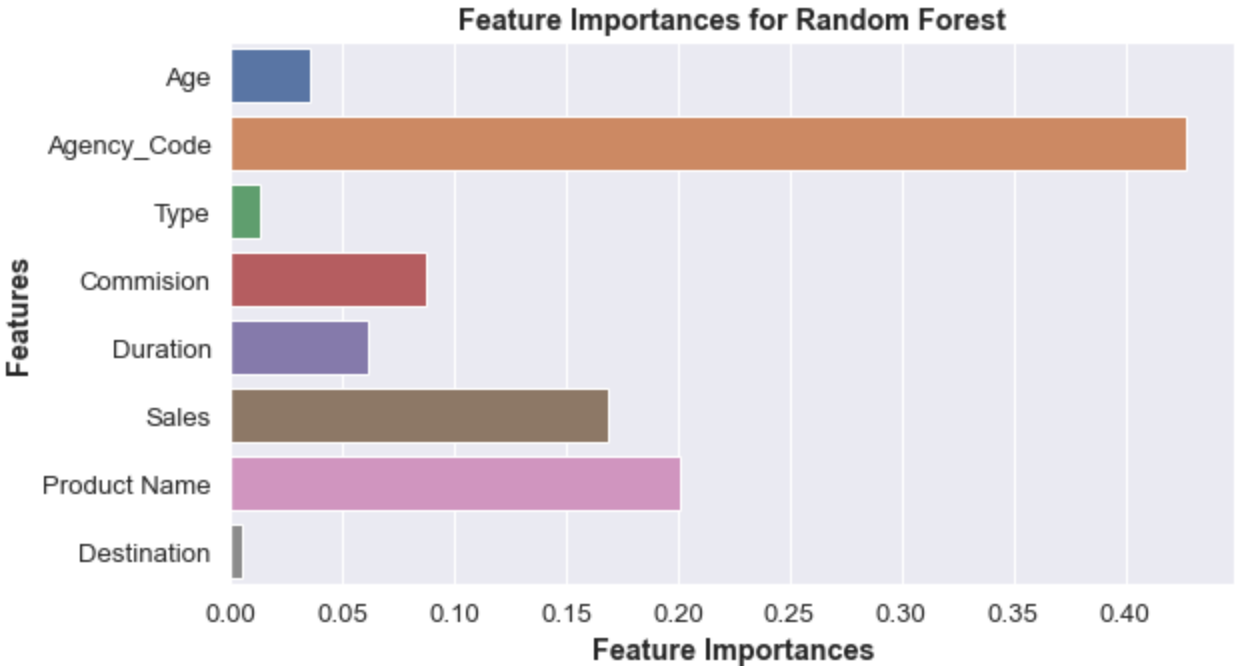


Figure 34: Feature Importances (Random Forest)

(plot generated using Python)

We can see the feature 'Agency_Code' plays the most vital role while making predictions on target feature, followed by 'Product Name' and 'Sales'. The status of these features majorly decide whether the person would claim the insurance or not.

The features 'Age', 'Type' and 'Destination' have negligible contribution in predicting the target feature.

➢ We saw the CART model also had similar metrics as the Random Forest model. We can view the feature importances for the CART model as well:

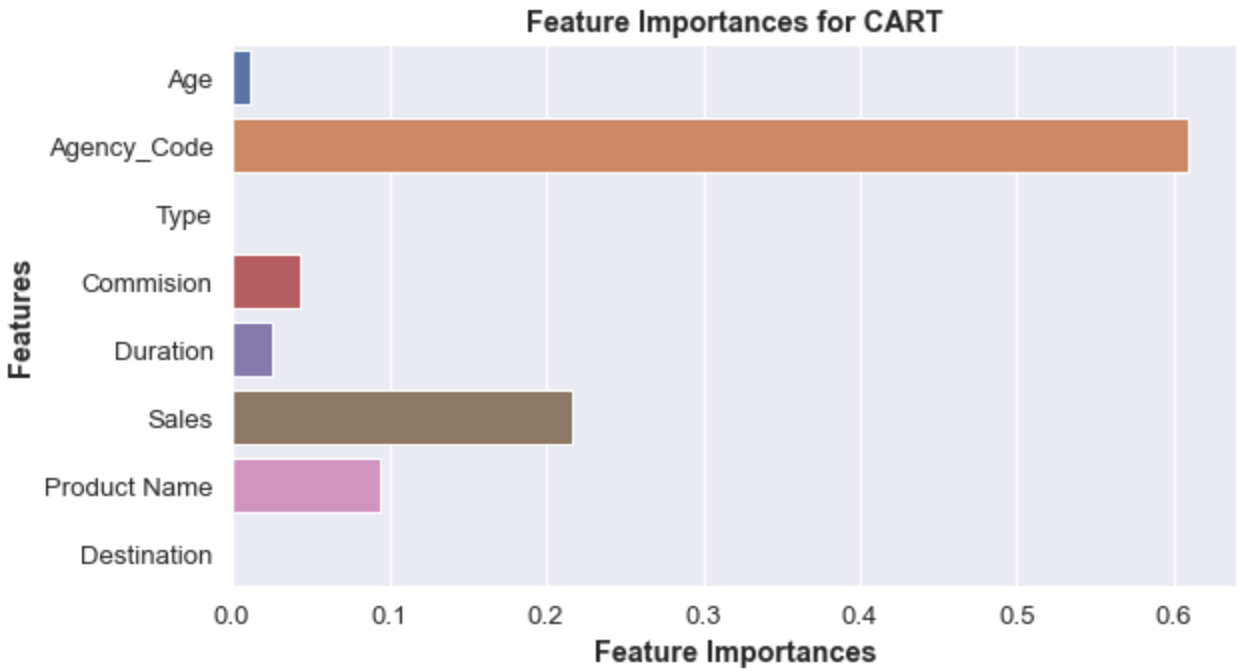


Figure 35: Feature Importances (CART)

(plot generated using Python)

We can see the feature 'Agency_Code' plays the most vital role while making predictions on target feature, followed by 'Sales' and 'Product Name'. The status of these features majorly decide whether the person would claim the insurance or not.

The features 'Age', 'Duration' and 'Commision' have negligible contribution in predicting the target feature.

The features 'Type' and 'Destination' have nil contribution in predicting the target feature.

References

- The logos for ‘Great Learning’, ‘Great Lakes’ and ‘UT Austin Texas McCombs’ School of Business’ have been taken from ‘https://olympus.greatlearning.in/’
- Author grateful to creators of:
  - https://towardsdatascience.com/
  - https://medium.com
  - https://www.geeksforgeeks.org/
  - https://developers.google.com/
- Steffen B Petersen, Aalborg University Hospital, has been a help too.