CLUSTERING ANALYSIS

&

CART-RF-ANN

BY

ISSAC ABRAHAM

CLUSTERING ANALYSIS

A leading bank wants to develop a customer segmentation to give promotional offers to its customers. They collected a sample that summarizes the activities of users during the past few months. You are given the task to identify the segments based on credit card usage.

Data Dictionary for Market Segmentation:

1. spending: Amount spent by the customer per month (in 1000s)
2. advance payments: Amount paid by the customer in advance by cash (in 100s)
3. probability_of_full_payment: Probability of payment done in full by the customer tothe 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 forpurchases made monthly (in 100s)
7. max_spent_in_single_shopping: Maximum amount spent in one purchase (in 1000s)

1.1 Read the data and do exploratory data analysis. Describe the data briefly.

So, we will import all the necessary libraries for cluster analysis,

Import numpy as np

Import pandas as pd

Import matplotlib.pyplot as plt

Import seaborn as sns

From sklearn.cluster import KMeans

From sklearn.metrics import silhouette samples, silhouette score

Reading the data,

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

- The data seems to be perfect
- The shape of the data is (210, 7)
- The info of the data indicates that all values are float

- No Null values in the data
- No missing values in the data

Description of the Data

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

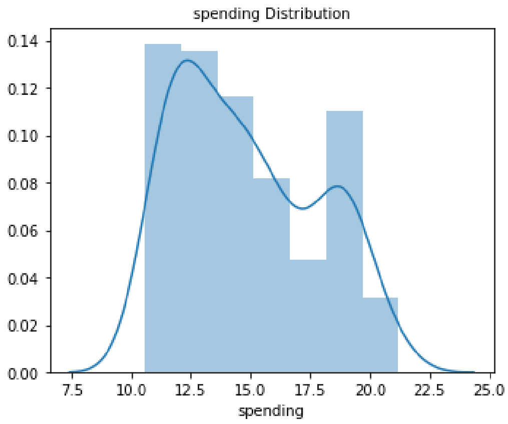
We have 7 variables,

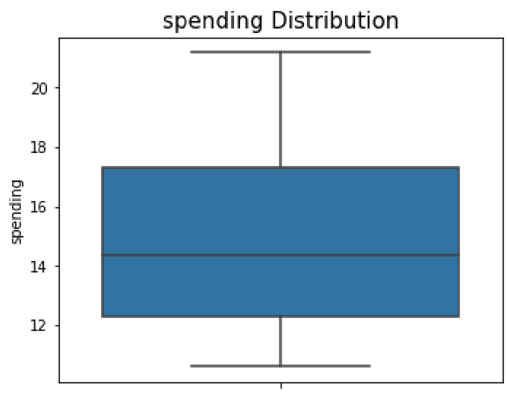
- No null values present in any variables.
- The mean and median values seems to be almost equal.
- The standard deviation for spending is high when compared to other variables.
- No duplicates in the dataset

Exploratory Data Analysis

Univariate / Bivariate analysis

Helps us to understand the distribution of data in the dataset. With univariate analysis we can find patterns and we can summarize the data and have understanding about the data to solve our business problem.



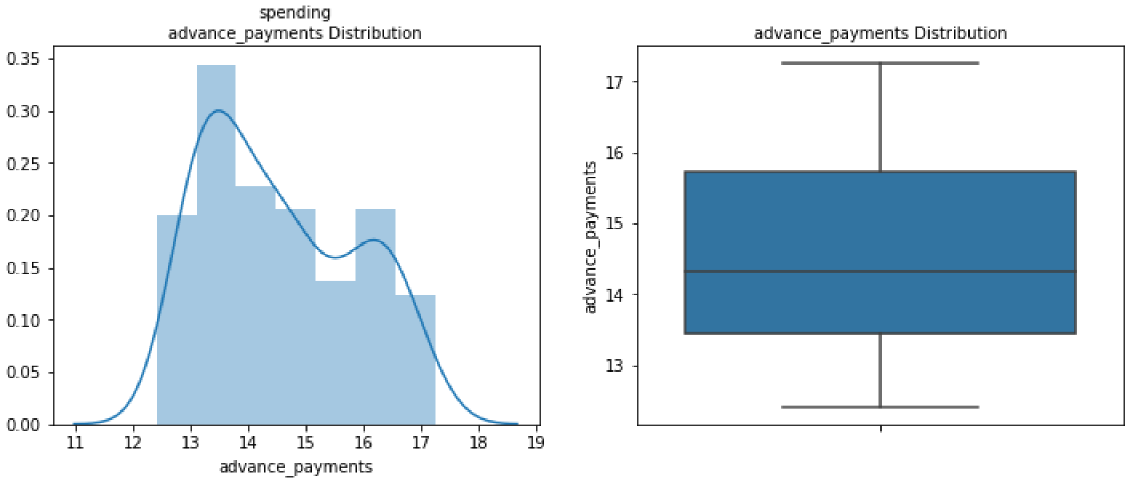


The box plot of the spending variable shows no outliers.

Spending is positively skewed - 0.399889.

We could also understand there could be chance of multi modes in the dataset.

The dist plot shows the distribution of data from 10 to 22

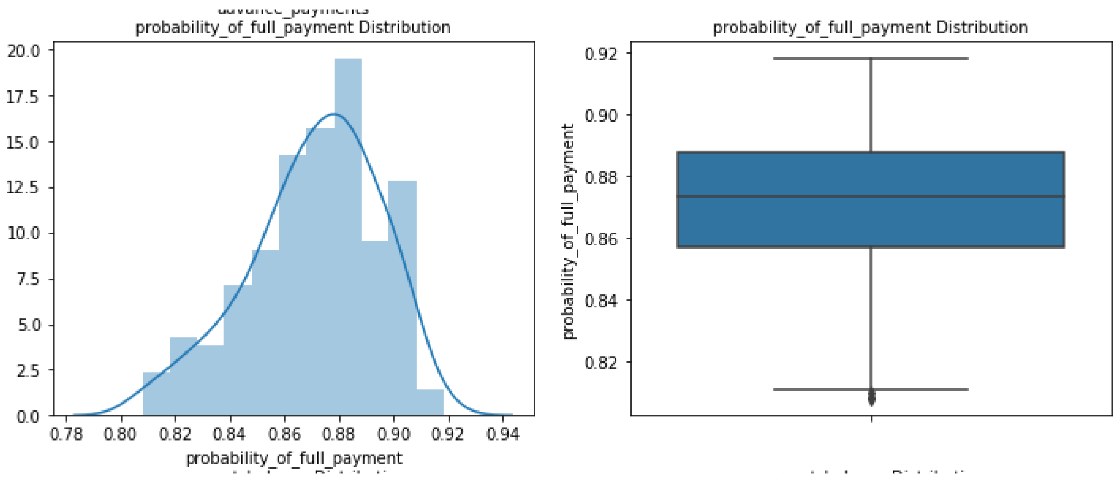


The box plot of the advance payments variable shows no outliers.

advance payments is positively skewed - 0.386573.

We could also understand there could be chance of multi modes in the dataset.

The dist plot shows the distribution of data from 12 to 17

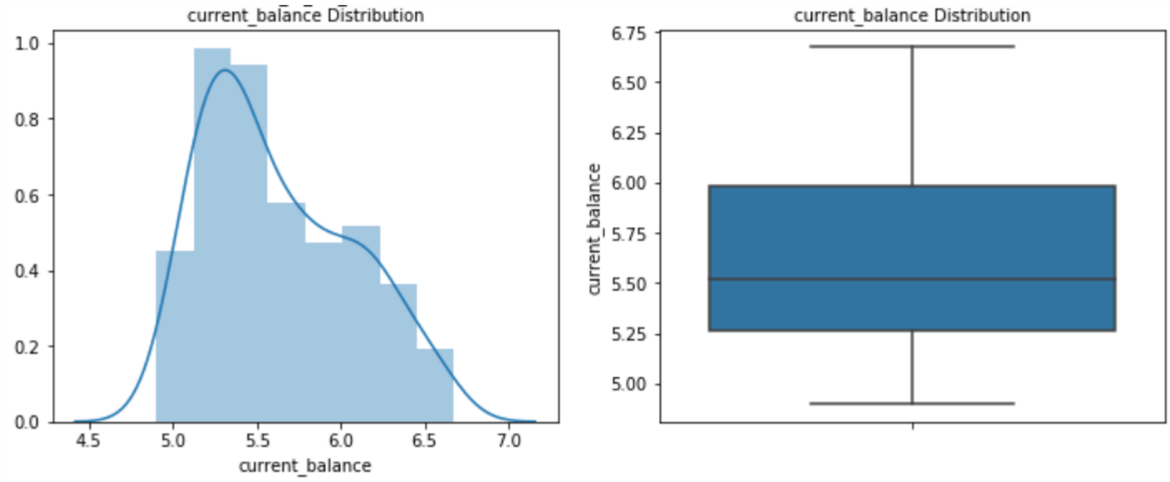


The box plot of the probability of full payment variable shows few outliers.

Probability of full payment is negatively skewed - -0.537954

The dist plot shows the distribution of data from 0.80 to 0.92.

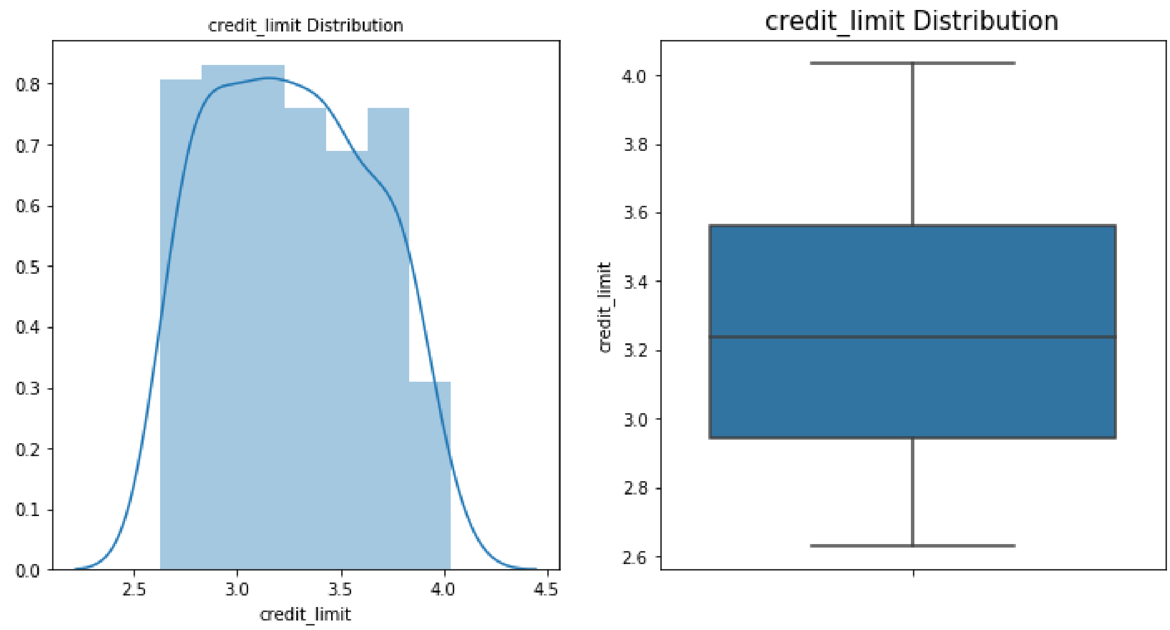
The Probability values is good above 80%



The box plot of the current balance variable shows no outliers.

Current balance is positively skewed - 0.525482

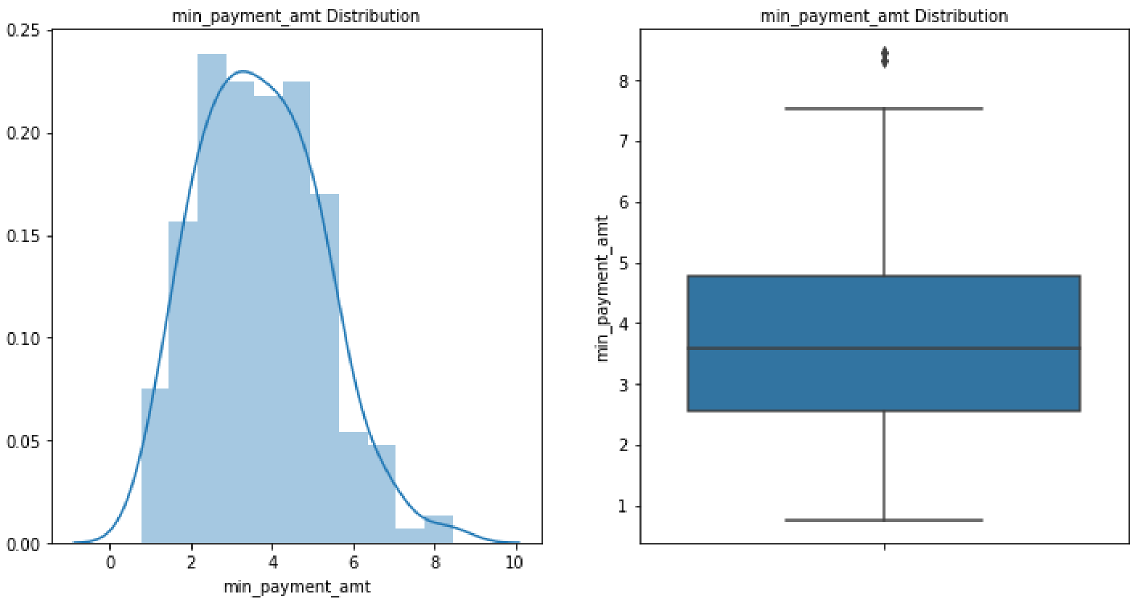
The dist plot shows the distribution of data from 5.0 to 6.5.



The box plot of the credit limit variable shows no outliers.

Credit limit is positively skewed - 0.134378

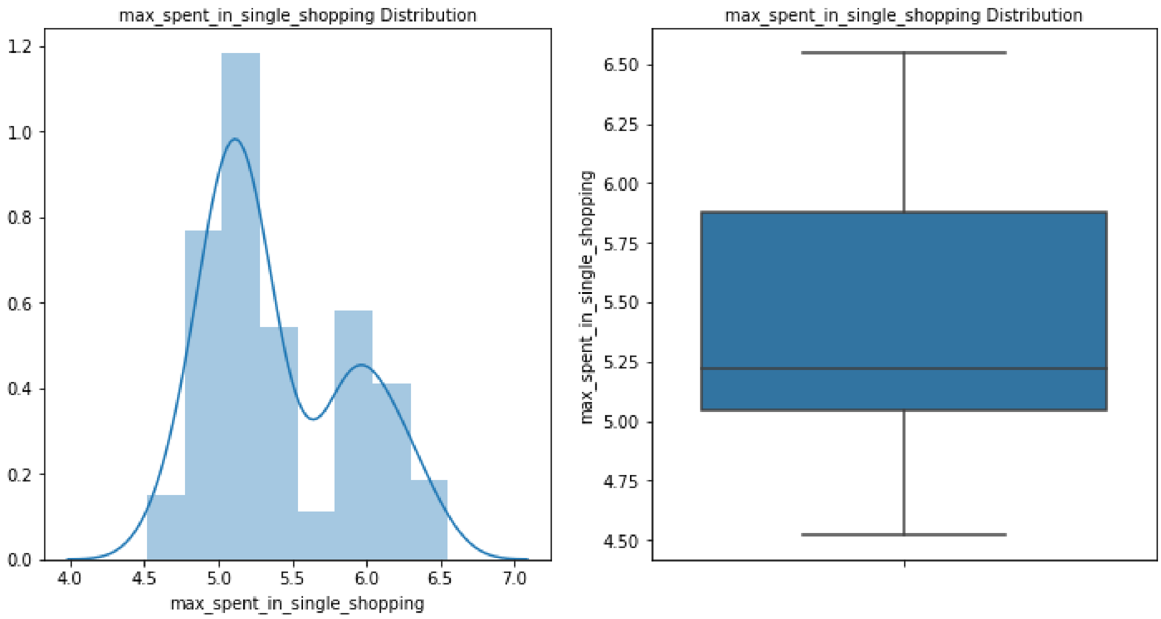
The dist plot shows the distribution of data from 2.5 to 4.0



The box plot of the min payment amount variable shows few outliers.

Min payment amount is positively skewed - 0.401667

The dist plot shows the distribution of data from 2 to 8



The box plot of the max spent in single shopping variable shows no outliers.

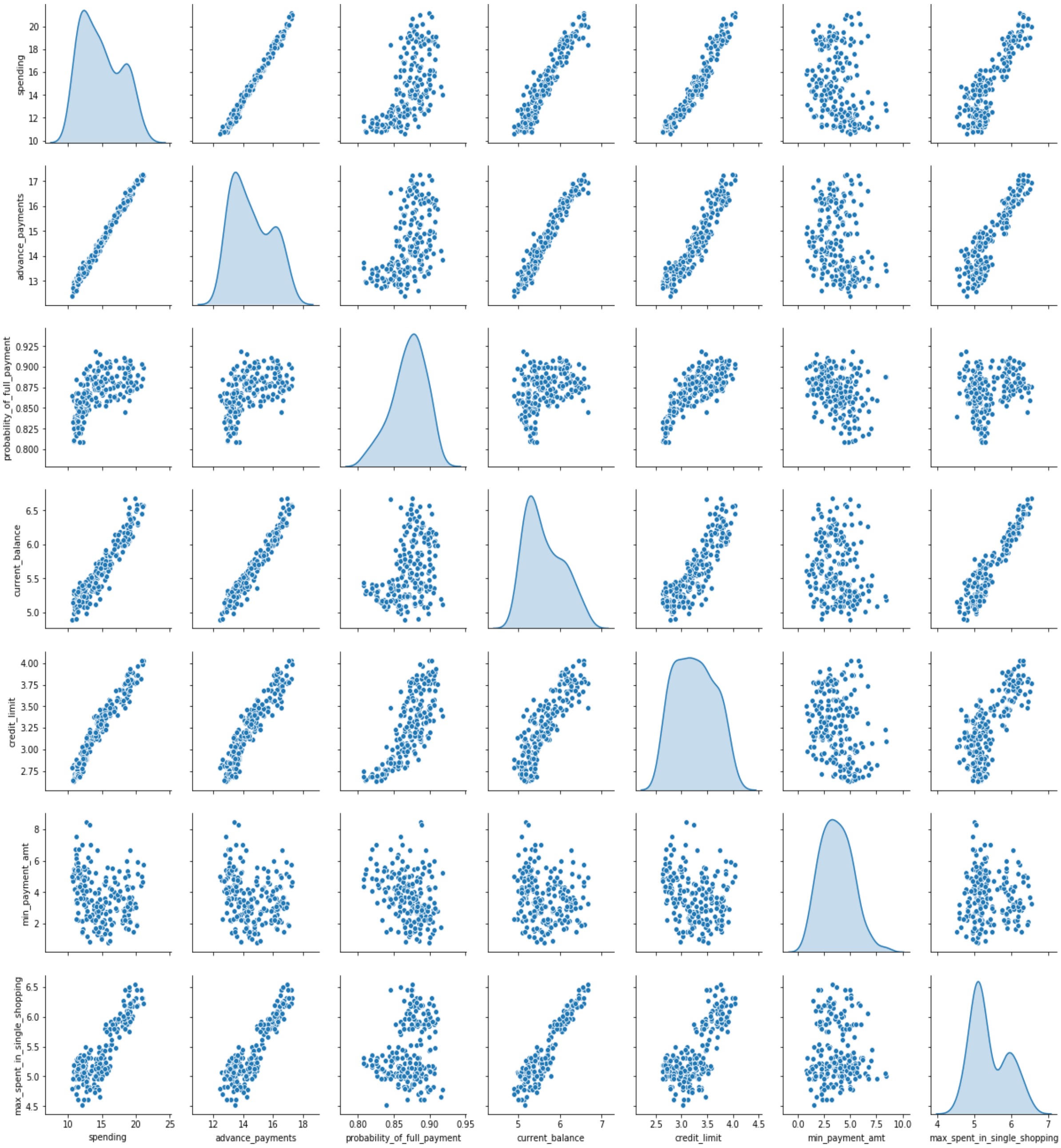
Max spent in single shopping is positively skewed - 0.561897

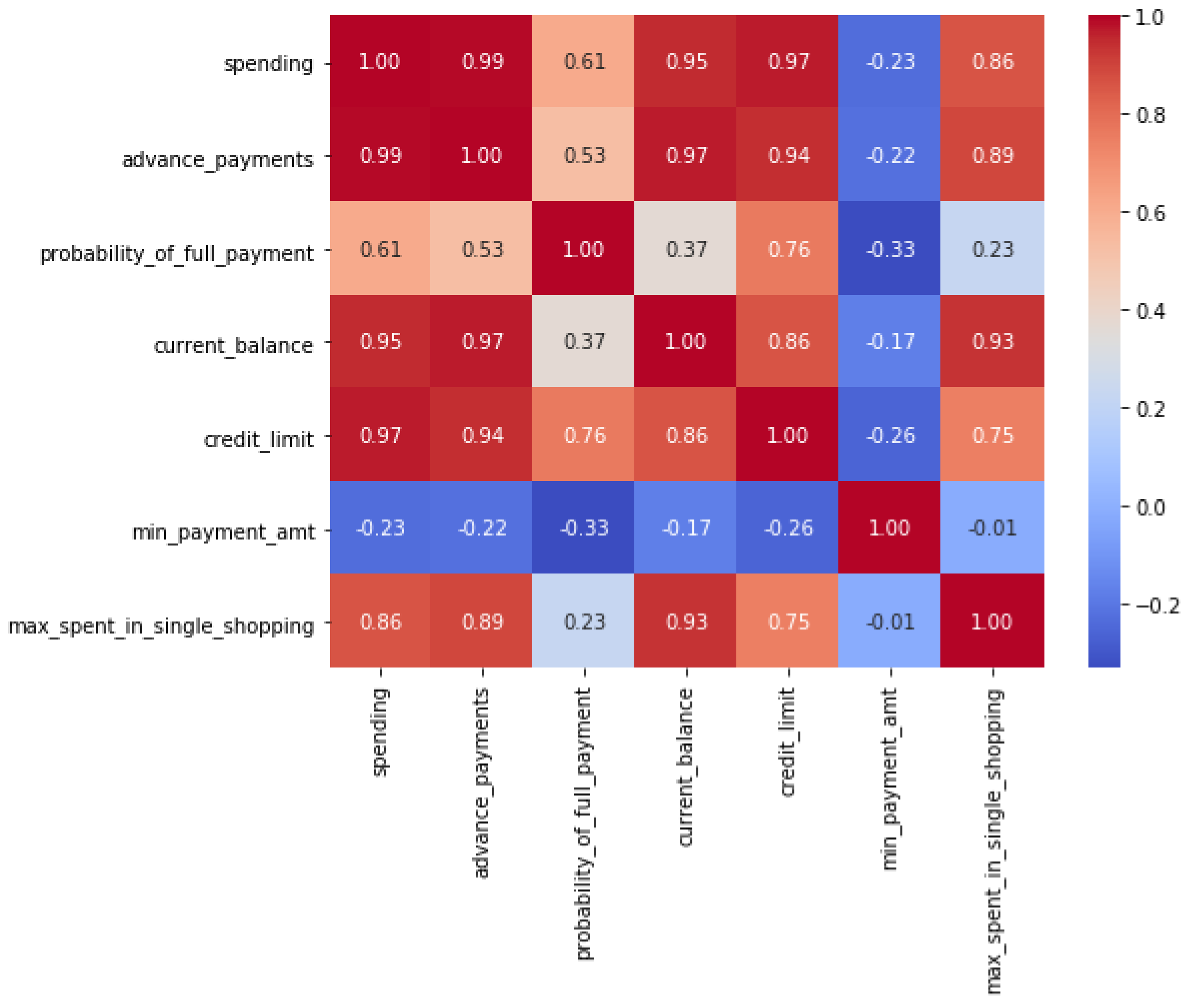
The dist plot shows the distribution of data from 4.5 to 6.5

No outlier treatment – only 3 to 4 values re observed has outlier we are treating them

Multivariate analysis

Check for multicollinearity





Heatmap for Better Visualization

Observations

Strong positive correlation

Between - spending & advance payments,

-advance payments & current balance,

- Credit limit & spending

- Spending & current balance

- credit limit & advance payments

- Max_spent_in_single_shopping current balance

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

Yes, scaling is very important as the model works based on the distance based computations scaling is necessary for unscaled data.

Scaling needs to be done as the values of the variables are in different scales.

Spending, advance payments are in different values and this may get more weightage.

Scaling will have all the values in the relative same range.

I have used standard scalar for scaling

Below is the snapshot of scaled data**.**

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping
0	1.754355	1.811968	0.178230	2.367533	1.338579	-0.298806	2.328998
1	0.393582	0.253840	1.501773	-0.600744	0.858236	-0.242805	-0.538582
2	1.413300	1.428192	0.504874	1.401485	1.317348	-0.221471	1.509107
3	-1.384034	-1.227533	-2.591878	-0.793049	-1.639017	0.987884	-0.454961
4	1.082581	0.998364	1.196340	0.591544	1.155464	-1.088154	0.874813

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

Hierarchical clustering – ward's method & average method

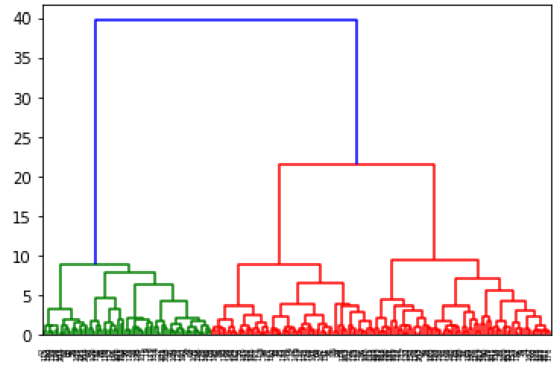
By choosing ward's method to the scaled data,

For visualization purposes I have used to Dendrogram

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

```
: wardlink = linkage(scaled_clust, method = 'ward')
```

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

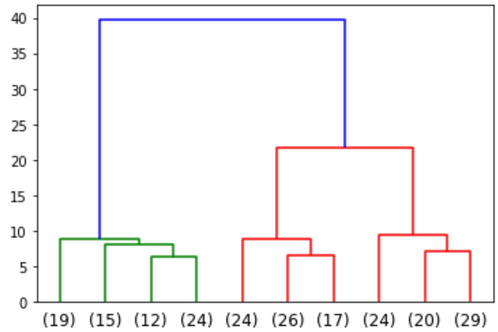


The above dendrogram indicates all the data points have clustered to different clusters by wards method.

To find the optimal number cluster through which we can solve our business objective we use truncate mode = lastp.

Wherein we can give last p = 10 according to industry set base value.

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



Now, we can understand all the data points have clustered into 3 clusters.

Next to map these clusters to our dataset we can use fclusters

Criterion we can give "maxclust"

```
]: clusters_ward = fcluster(wardlink, 3, criterion='maxclust')
```

```
]: clusters_ward
```

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

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

Now, we can look at the cluster frequency in our dataset,

Cluster Frequency

```
]: cluster_ward_dataset.clusters_ward.value_counts().sort_index()
```

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

Cluster profiling to understand the business problem.

```
aggdata=cluster_ward_dataset.groupby('clusters_ward').mean()
aggdata['Freq']=cluster_ward_dataset.clusters_ward.value_counts().sort_index()
aggdata
```

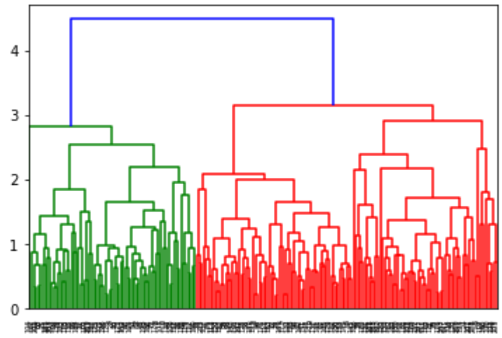
clusters_ward	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	Freq
1	18.371429	16.145429	0.884400	6.158171	3.684629	3.639157	6.017371	70
2	11.872388	13.257015	0.848072	5.238940	2.848537	4.949433	5.122209	67
3	14.199041	14.233562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

By choosing average method to the scaled data,

Choosing ward linkage method

```
[39]: wardlink_1 = linkage(scaled_clust, method = 'average')
```

```
[40]: dend_1 = dendrogram(wardlink_1)
```

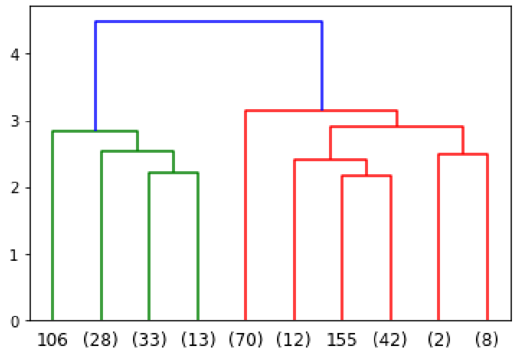


The above dendrogram indicates all the data points have clustered to different clusters by average method.

To find the optimal number cluster through which we can solve our business objective we use truncate mode = lastp.

Wherein we can give last p = 10 according to industry set base value.

```
]: dend_1= dendrogram(wardlink_1,
                       truncate_mode='lastp',
                       p = 10,
                       )
```



Now, we can understand all the data points have clustered into 3 clusters.

Next to map these clusters to our dataset we can use fclusters

Criterion we can give "maxclust"

```
clusters_average = fcluster(wardlink_1, 3, criterion='maxclust')
```

```
clusters_average
```

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

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

Now, we can look at the cluster frequency in our dataset,

Cluster Frequency

```
cluster_average_dataset.clusters_average.value_counts().sort_index()
```

```
1    75
2    70
3    65
Name: clusters_average, dtype: int64
```

Cluster Profiles

```
aggdata_1=cluster_average_dataset.groupby('clusters_average').mean()
aggdata_1['Freq']=cluster_average_dataset.clusters_average.value_counts().sort_index()
aggdata_1
```

lusters_average	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	Freq
1	18.129200	16.058000	0.881595	6.135747	3.648120	3.650200	5.987040	75
2	11.916857	13.291000	0.846766	5.258300	2.846000	4.619000	5.115071	70
3	14.217077	14.195846	0.884869	5.442000	3.253508	2.768418	5.055569	65

Observation

Both the method are almost similar means, minor variation, which we know it occurs.
There was not too much variations from both methods

Cluster grouping based on the dendrogram, 3 or 4 looks good. Did the further analysis, and based on the dataset had gone for 3 group cluster

And three group cluster solution gives a pattern based on high/medium/low spending with max_spent_in_single_shopping (high value item) and probability_of_full_payment (payment made).

1.4 Apply K-Means clustering on scaled data and determine optimum clusters. Apply elbow curve and silhouette score.

K-means clustering,

Randomly we decide to give n_clusters = 3 and we look at the distribution of clusters according to the n_clusters.

We apply K-means technique to the scaled data.

Cluster output for all the observations in the dataset,

Cluster Output for all the observations

```
k_means.labels_
```

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

We have 3 clusters 0,1,2

To find the optimal number of clusters, we can use k-elbow method

Calculating WSS for other values of K - Elbow Method

```
wss =[]
```

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

```
wss
```

```
[1469.9999999999998,
 659.171754487041,
 430.6589731513006,
 371.38509060801096,
 327.21278165661346,
 289.31599538959495,
 262.98186570162267,
 241.81894656086033,
 223.91254221002725,
 206.39612184786694]
```

To find the inertia value for all the clusters from 1 to 11, I used a for loop to find the optimal number of clusters.

The silhouette score for 3 clusters is good

```
k_means = KMeans(n_clusters = 3,random_state=1)
k_means.fit(scaled_clust)
labels = k_means.labels_
```

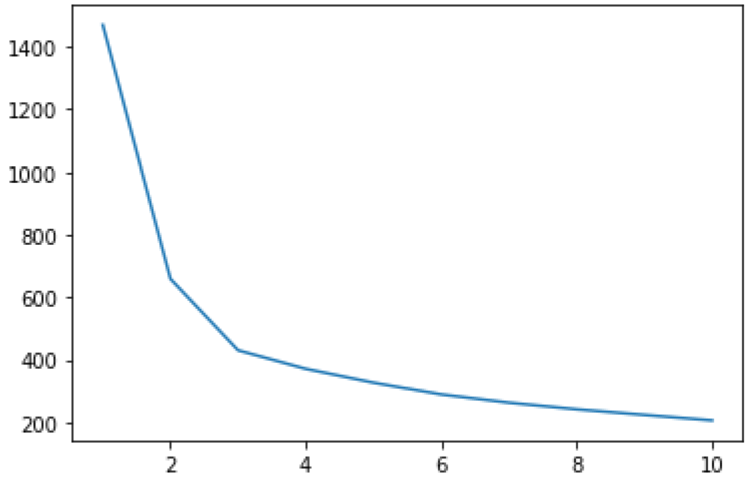
```
silhouette_score(scaled_clust,labels,random_state=1)
```

```
0.4007270552751299
```

The elbow curve seen here also shows us after 3 clusters there is no huge drop in the values, so we select 3 clusters.

```
plt.plot(range(1,11), wss)
```

```
[<matplotlib.lines.Line2D at 0x1df0fbc2908>]
```



So adding the cluster results to our dataset to solve our business objective.

pending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	sil_width	Clus_kmeans
19.94	16.92	0.8752	6.675	3.763	3.252	6.550	0.573699	1
15.99	14.89	0.9064	5.363	3.582	3.336	5.144	0.366386	2
18.95	16.42	0.8829	6.248	3.755	3.368	6.148	0.637784	1
10.83	12.96	0.8099	5.278	2.641	5.182	5.185	0.512458	0
17.99	15.86	0.8992	5.890	3.694	2.068	5.837	0.362276	1

This table shows the clusters to the dataset and also individual sil_width score.

Cluster frequency

Cluster Profiling

```
3]: cluster_kmeans_dataset.Clus_kmeans.value_counts().sort_index()
```

```
3]: 0    71
    1    72
    2    67
    Name: Clus_kmeans, dtype: int64
```

This frequency shows frequency of clusters to the dataset.

eans	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_single_shopping	sil_width	Freq
0	14.437887	14.337746	0.881597	5.514577	3.259225	2.707341	5.120803	0.272895	71
1	11.856944	13.247778	0.848253	5.231750	2.849542	4.742389	5.101722	0.384945	72
2	18.495373	16.203433	0.884210	6.175687	3.697537	3.632373	6.041701	0.324118	67

3-Group clusters via K- Means has equal split of percentage of results.

Cluster 0 – Medium

Cluster 1 – low

Cluster 2 – High

Observation

By K- Mean's method we can infer that at cluster 3 there is no huge drop in inertiavalues. Also the elbow curve seems to show similar results. The silhouette width score indicates all the data points are properly clustered to the cluster. There is no mismatch in the data points with regards toclustering.. Did the further analysis, andbased on the dataset had gone for 3 group cluster. The three group cluster solution gives a pattern based on high/medium/low spending with max_spent_in_single_shopping (high value item) and probability_of_full_payment (paymentmade).

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

Group 1: High Spending Group –
Giving any reward points might increase their purchases. –

Maximum max_spent_in_single_shopping is high for this group, so can be offereddiscount/offer on next transactions upon full payment –
Increase their credit limit and –
Increase spending habits –

Give loan against the credit card, as they are customers with good repayment record. –
Tie up with luxury brands, which will drive more one_time_maximun spending

Group 2: Low Spending Group - customers should be given remainders for payments. Offers can be provided on early payments to improve their payment rate.
- Increase their spending habits by tying up with grocery stores, utilities (electricity, phone, gas, others)

Group 3: Medium Spending Group - They are potential target customers who are paying bills and doing purchases and maintaining comparatively good credit score. So we can increase credit limit or can lower down interest rate. - Promote premiumcards/loyalty cars to increase transactions. - Increase spending habits by trying withpremium ecommerce sites, travel portal, travel airlines/hotel, as this will encourage them to spend more

CART-RF-ANN

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

Data Dictionary

1. Target: Claim Status (Claimed)
2. Code of tour firm (Agency Code)
3. Type of tour insurance firms (Type)
4. Distribution channel of tour insurance agencies (Channel)
5. Name of the tour insurance products (Product)

6. Duration of the tour (Duration)
7. Destination of the tour (Destination)
8. Amount of sales of tour insurance policies (Sales)
9. The commission received for tour insurance firm (Commission)
10. Age of insured (Age)

2.1 Data Ingestion: Read the dataset. Do the descriptive statistics and do null value condition check, write an inference on it.

Importing all required libraries,

Import numpy as np
Import pandas as Pd
Import matplotlib.pyplot as plt
Import seaborn as sns
From sklearn import tree
From sklearn.tree import DecisionTreeClassifier
From sklearn.ensemble import RandomForestClassifier
From sklearn.neural_network import MLPClassifier
From sklearn.model_selection import train_test_split
From sklearn.metrics import roc_auc_score, roc_curve,classification_report,confusion_matrix
From sklearn.preprocessing import StandardScaler
From sklearn.model_selection import GridSearchCV
From scipy import stats

Reading the dataset,

Checking the data

```
df_insured.head()
```

[5]:

	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

The data has read successfully,

The shape of the dataset is (3000, 10)

Info function clearly indicates the dataset has object, integer and float so we have to change the object data type to numeric value.

```
df_insured.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Age           3000 non-null   int64
 1   Agency_Code   3000 non-null   object
 2   Type          3000 non-null   object
 3   Claimed       3000 non-null   object
 4   Commision     3000 non-null   float64
 5   Channel       3000 non-null   object
 6   Duration      3000 non-null   int64
 7   Sales         3000 non-null   float64
 8   Product Name  3000 non-null   object
 9   Destination   3000 non-null   object
dtypes: float64(2), int64(2), object(6)
memory usage: 234.5+ KB
```

No missing values in the dataset,

Check for missing value in any column

```
]:  df_insured.isnull().sum()
```

```
ıt[8]: Age             0
       Agency_Code     0
       Type            0
       Claimed         0
       Commision       0
       Channel         0
       Duration        0
       Sales           0
       Product Name    0
       Destination     0
       dtype: int64
```

Summary of the dataset,

Summary of the data

]:
```
df_insured.describe(include="all").T
```

ut[9]:

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

We have 4 numeric values and 6 categorical values,

Agency code EPX has a frequency of 1365,

The most preferred type seems to be travel agency

Channel is online

Customized plan is the most sought plan by customers

Destination ASIA seems to be most sought destination place by customers.

We will further look at the distribution of dataset in univarite and bivariate analysis

Checking for duplicates in the dataset,

Check for duplicate data

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

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

Removing Duplicates

since i don't find any unique identifier in the dataset to remove these duplicates these duplicates can be different customers so i'm not dropping these duplicates.

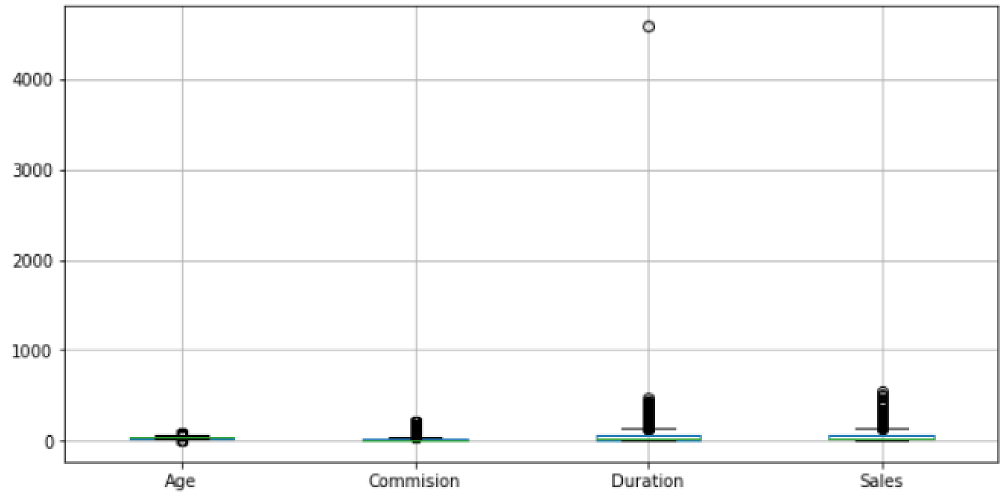
Checking for Outliers

As there is no unique identifier I'm not dropping the duplicates it may be different customer's data.

Checking for Outliers

```
[11]:  plt.figure(figsize=(10,5))
       df_insured[['Age','Commision', 'Duration', 'Sales']].boxplot()
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x22d338cce88>
```

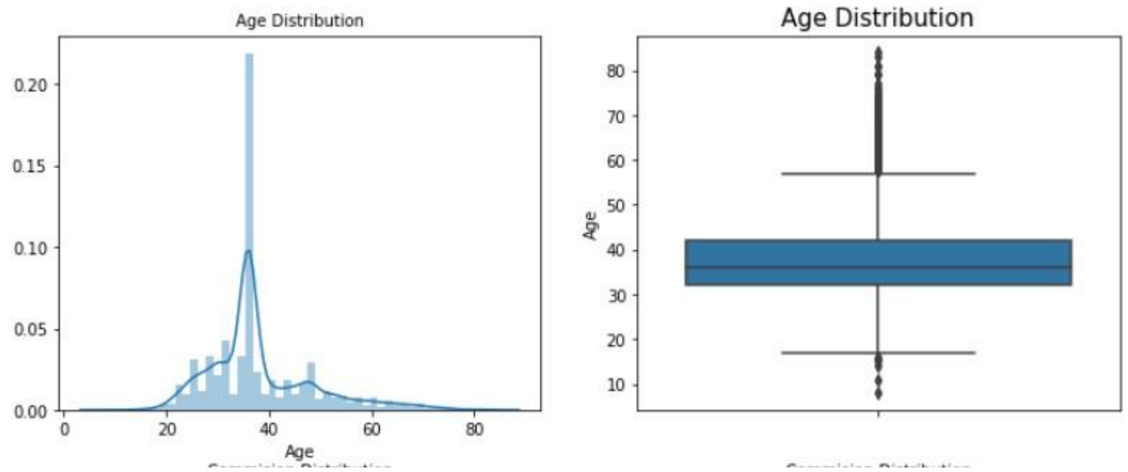


Outliers exist in almost all the numeric values.

We can treat outliers in random forest classification.

Geting unique counts of all Nominal Variables

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



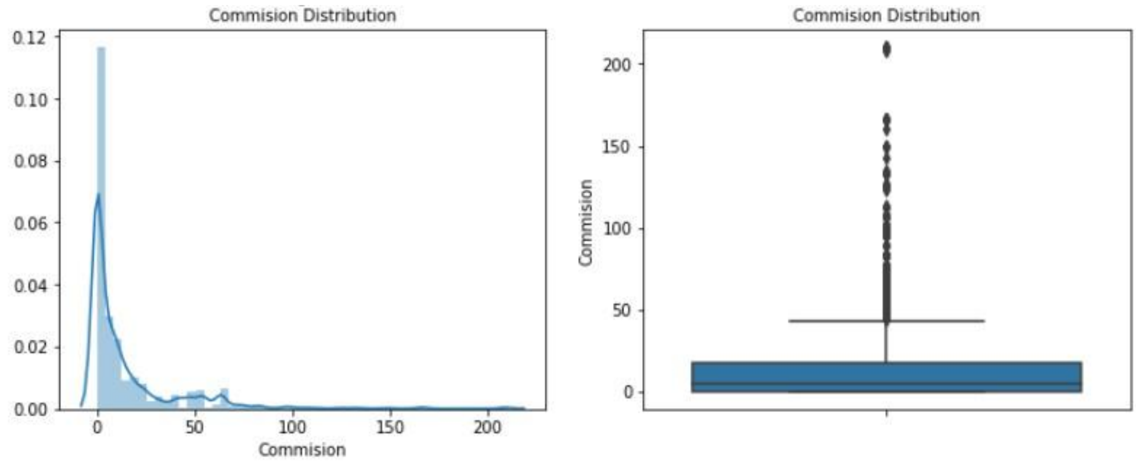
Univariate / Bivariate analysis

The box plot of the age variable shows outliers.

Spending is positively skewed - 1.149713

The dist plot shows the distribution of data from 20 to 80

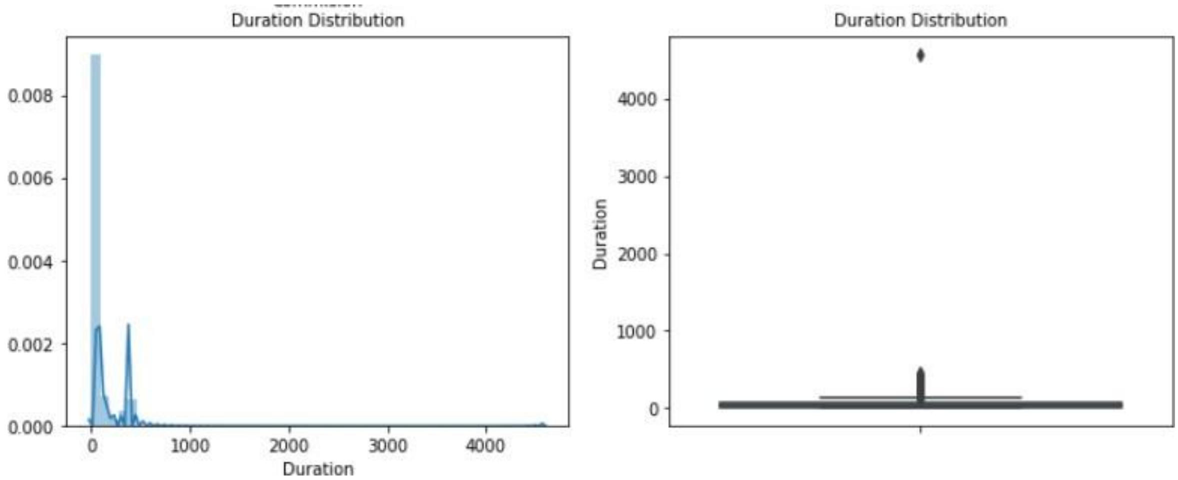
In the range of 30 to 40 is where the majority of the distribution lies.



The box plot of the commission variable shows outliers.

Spending is positively skewed - 3.148858

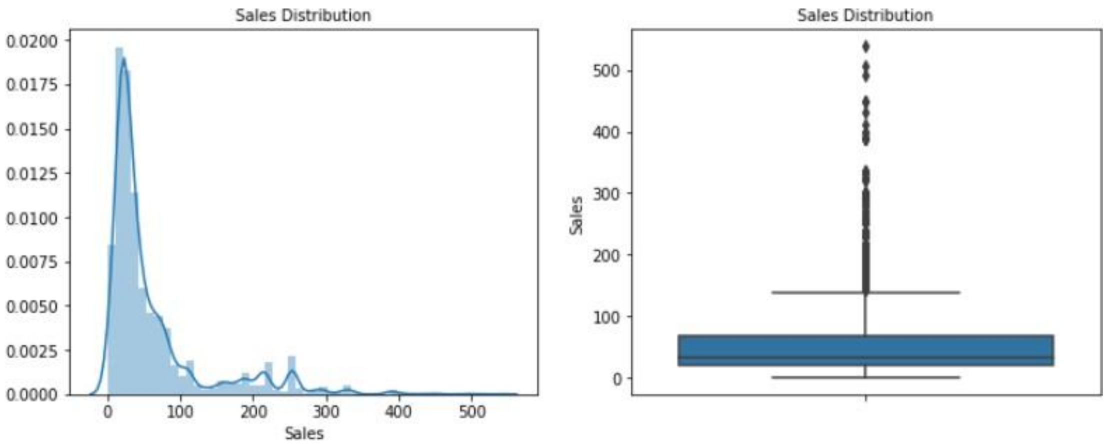
The dist plot shows the distribution of data from 0 to 30



The box plot of the duration variable shows outliers.

Spending is positively skewed - 13.784681

The dist plot shows the distribution of data from 0 to 100



The box plot of the sales variable shows outliers.

Spending is positively skewed - 2.381148

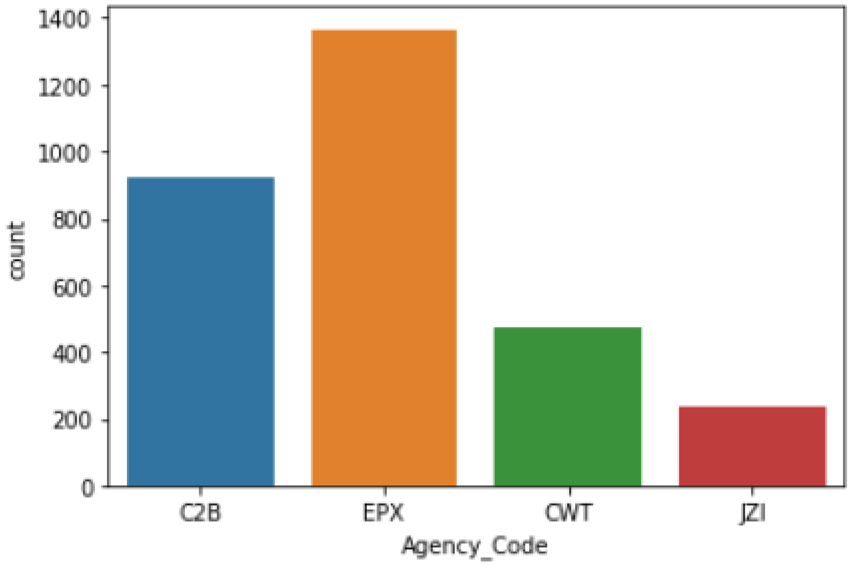
The dist plot shows the distribution of data from 0 to 300

Categorical Variables

Agency Code

```
sns.countplot(df_insured['Agency_Code'])
```

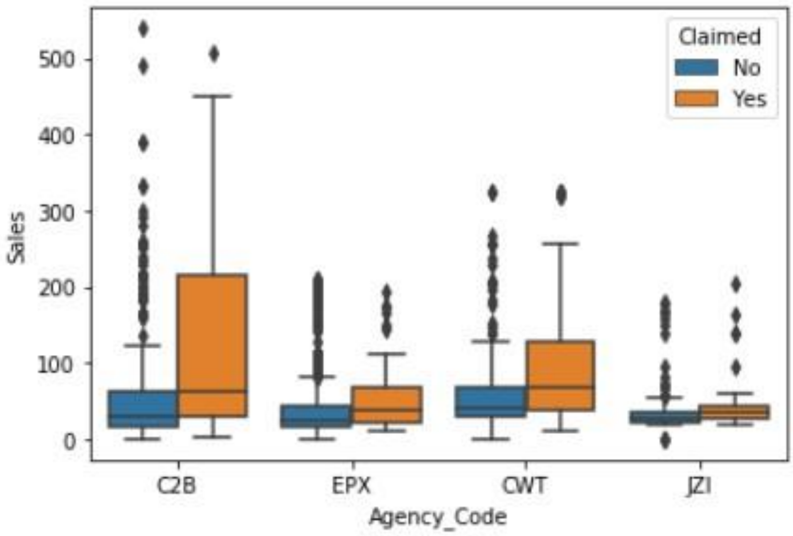
]: <matplotlib.axes._subplots.AxesSubplot at 0x22d34aa6448>



The distribution of the agency code, shows us EPX with maximum frequency

```
sns.boxplot(data = df_insured, x='Agency_Code',y='Sales', hue='Claimed')
```

<matplotlib.axes._subplots.AxesSubplot at 0x22d34b391c8>



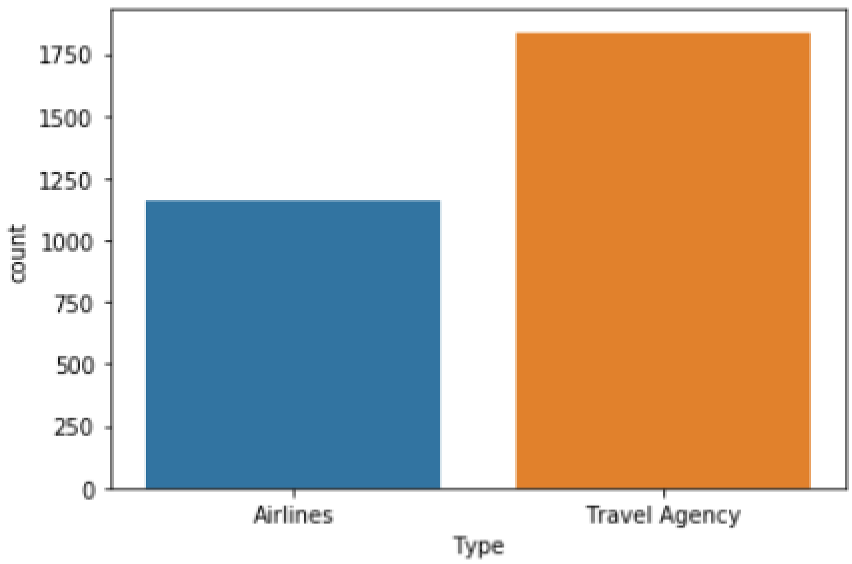
The box plot shows the split of sales with different agency code and also hue having claimed column.

It seems that C2B have claimed more claims than other agency.

Type

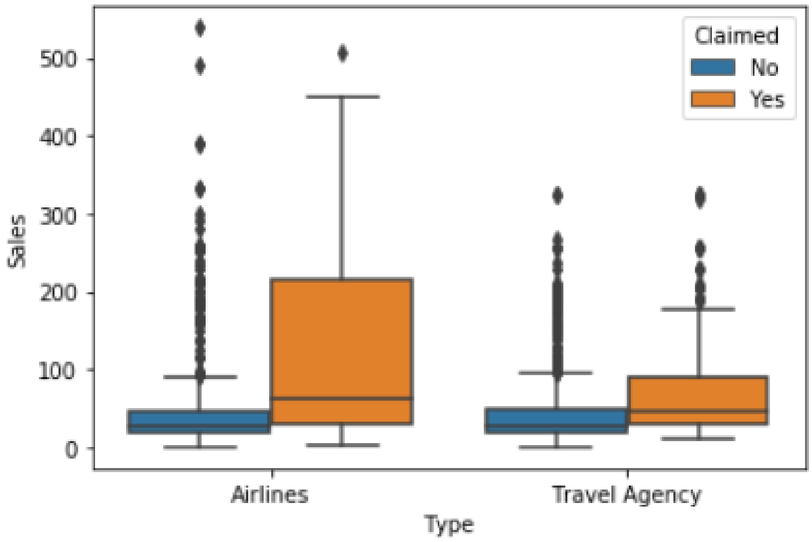
```
sns.countplot(data = df_insured, x = 'Type')
```

7]: <matplotlib.axes._subplots.AxesSubplot at 0x22d3471c788>



```
sns.boxplot(data = df_insured, x='Type',y='Sales', hue='Claimed')
```

]: <matplotlib.axes._subplots.AxesSubplot at 0x22d34763288>



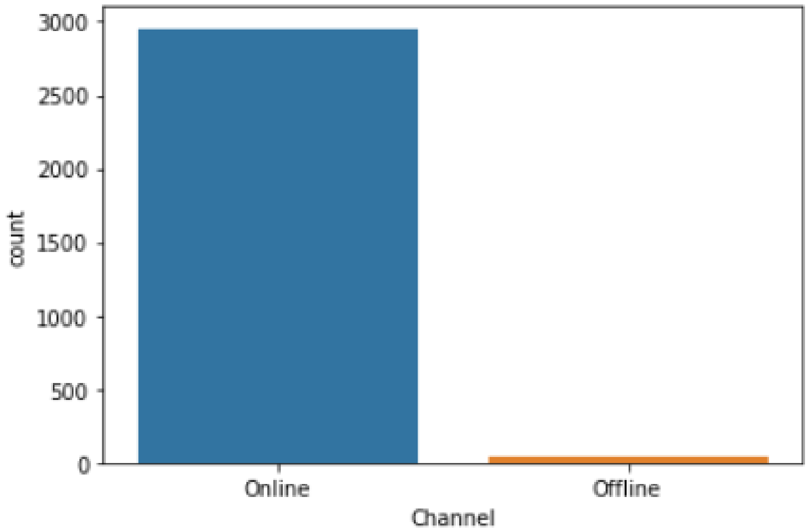
The box plot shows the split of sales with different type and also hue having claimed column.

We could understand airlines type has more claims.

Channel

```
sns.countplot(data = df_insured, x = 'Channel')
```

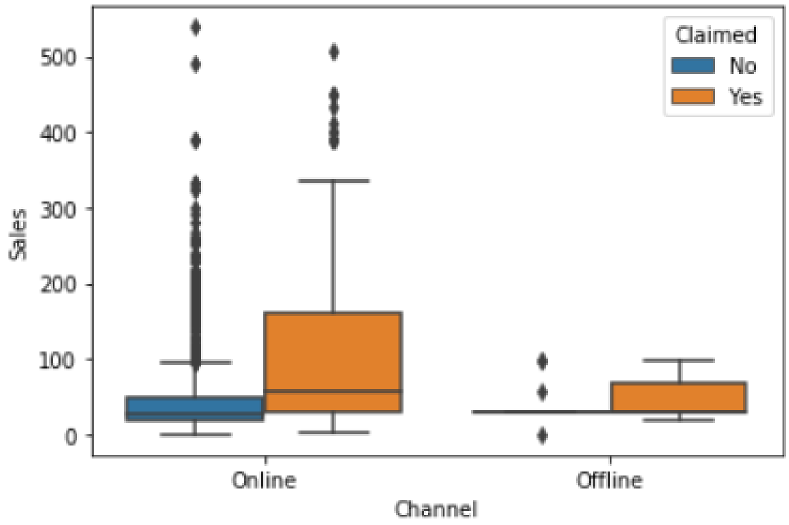
.9]: <matplotlib.axes._subplots.AxesSubplot at 0x22d34827e48>



The majority of customers have used online medium, very less with offline medium

```
sns.boxplot(data = df_insured, x='Channel',y='Sales', hue='Claimed')
```

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x22d34883588>
```

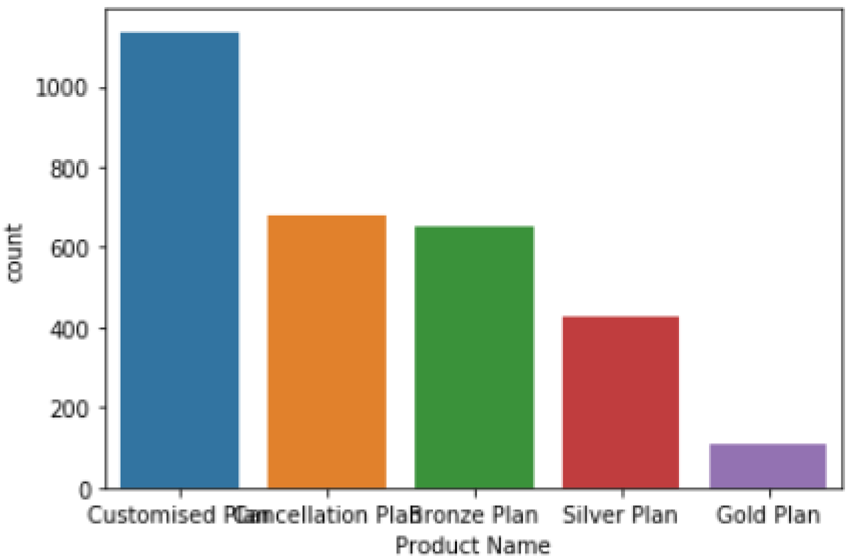


The box plot shows the split of sales with different channel and also hue having claimed column.

Product Name

```
1]: sns.countplot(data = df_insured, x = 'Product Name')
```

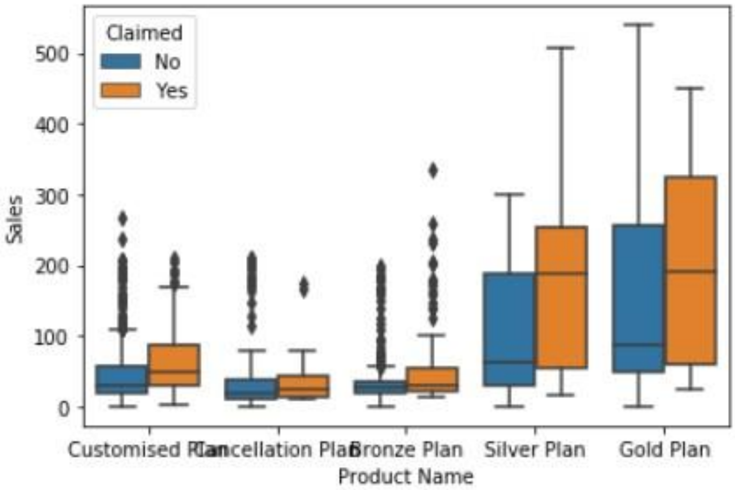
```
ut[21]: <matplotlib.axes._subplots.AxesSubplot at 0x22d348ce308>
```



Customized plan seems to be most liked plan by customers when compared to all other plans.

```
sns.boxplot(data = df_insured, x='Product Name',y='Sales', hue='Claimed')
```

]: <matplotlib.axes._subplots.AxesSubplot at 0x22d349a2408>

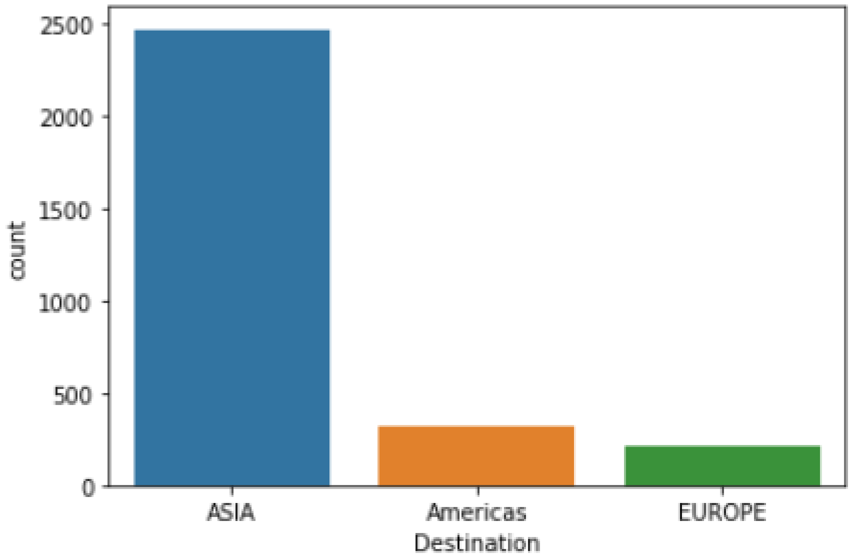


The box plot shows the split of sales with different product name and also hue having claimed column.

Destination

```
sns.countplot(data = df_insured, x = 'Destination')
```

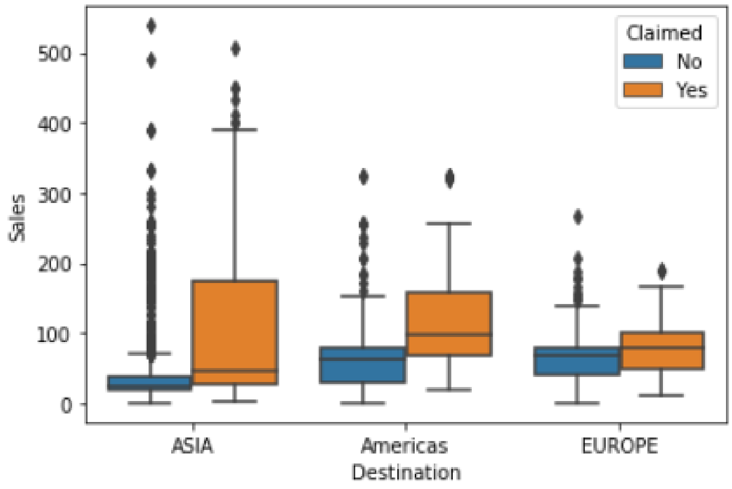
3]: <matplotlib.axes._subplots.AxesSubplot at 0x22d35eba388>



Asia is where customers choose when compared with other destination places.

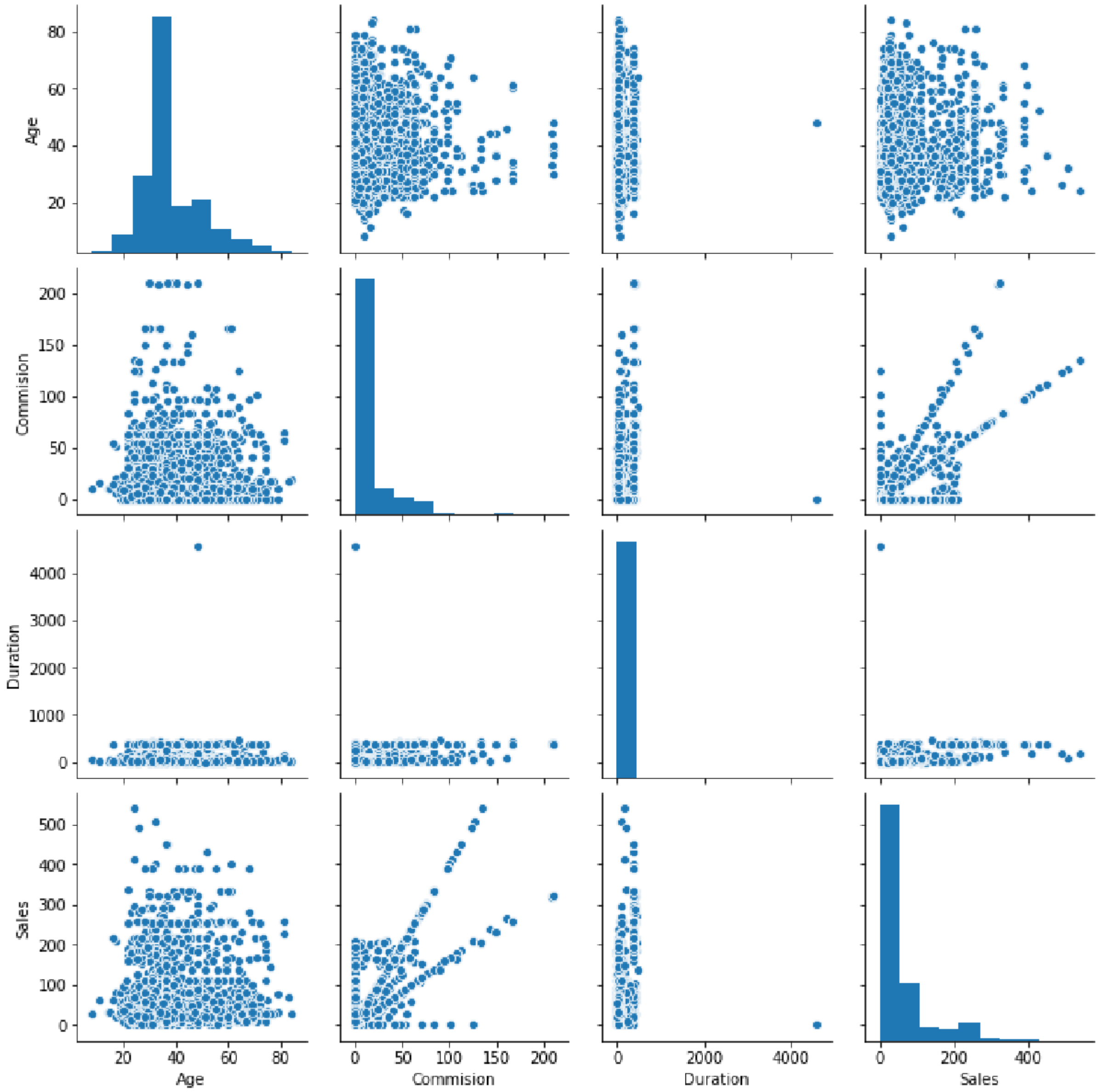
```
sns.boxplot(data = df_insured, x='Destination',y='Sales', hue='Claimed')
```

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x22d35ef1ec8>
```

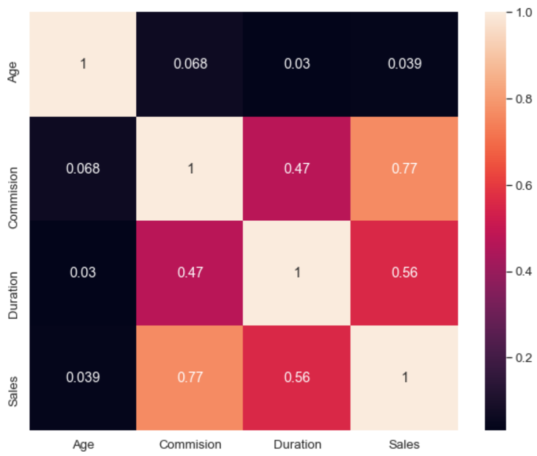


The box plot shows the split of sales with different destination and also hue having claimed column.

Checking pairwise distribution of the continuous variables



Checking for Correlations



Not much of multi collinearity observed

No negative correlation

Only positive correlation

Converting all objects to categorical codes

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

Checking the info

```
df_insured.info()
```

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

```
df_insured.head()
```

]:

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

Proportion of 1s and 0s

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

```
30]: 0    0.692
     1    0.308
     Name: Claimed, dtype: float64
```

Checking the proportion of 1s and 2s in the dataset. That is our target column.

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

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

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

y = df_insured.pop("Claimed")

X.head()
```

31]:

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

For training and testing purpose we are splitting the dataset into train and test data in the ratio 70:30.

Splitting data into training and test set

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

Checking the dimensions of the training and test data

```
print('X_train',X_train.shape)
print('X_test',X_test.shape)
print('train_labels',train_labels.shape)
print('test_labels',test_labels.shape)
```

```
X_train (2100, 9)
X_test (900, 9)
train_labels (2100,)
test_labels (900,)
```

We have bifurcated the dataset into train and test.

We have also taken out the target column out of train and test data into separate vector for evaluation purposes.

MODEL 1

Building a Decision Tree Classifier

```
dt_model = DecisionTreeClassifier(criterion = 'gini' )
```

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

```
35]: DecisionTreeClassifier()
```

CHECKING THE FEATURE

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

```
                   Imp
Duration      0.276811
Agency_Code   0.194356
Sales         0.194228
Age           0.163714
Commision     0.102841
Product Name  0.038334
Destination   0.019359
Channel       0.007262
Type          0.003095
```

OPTIMAL VALUES FOR DECISSION TREE,GRID

SEARCH FOR FINDING,

Grid Search for finding out the optimal values for the hyper parameters

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [4, 5,6],
    'min_samples_leaf': [20, 40, 60, 70],
    'min_samples_split': [150, 200, 250, 300,]
}

dt_model = DecisionTreeClassifier()

grid_search = GridSearchCV(estimator = dt_model, param_grid = param_grid, cv = 10)
```

FITTING THE OPTMAL VALUES TO THE TRAINING DATASET

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

```
: GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [4, 5, 6],
                         'min_samples_leaf': [20, 40, 60, 70],
                         'min_samples_split': [150, 200, 250, 300]})
```

BEST GRID

```
best_grid
```

```
reg_dt_model = DecisionTreeClassifier(criterion = 'gini', max_depth = 4,min_samples_leaf=20,min_samples_split=150)
```

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

```
DecisionTreeClassifier(max_depth=4, min_samples_leaf=20, min_samples_split=150)
```

Generating New Tree

```
insurance_prediction_tree_regularized = open('C:\\Users\\WELCOME\\Downloads\\PYTHON FILES\\4.Data Mining\Project\\insurance_p
dot_data = tree.export_graphviz(reg_dt_model, out_file= insurance_prediction_tree_regularized , feature_names = list(X_train)

insurance_prediction_tree_regularized.close()
dot_data
```

Variable Importance

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

```
                   Imp
Agency_Code   0.616392
Sales         0.252286
Product Name  0.077771
Commision     0.022912
Duration      0.022624
Age           0.008015
Type          0.000000
Channel       0.000000
Destination   0.000000
```

Predicting on Training dataset for Decission Tree

```
ytrain_predict_dt = reg_dt_model.predict(X_train)
```

```
ytest_predict_dt = reg_dt_model.predict(X_test)
```

MODEL 2

Building a Ensemble RandomForest Classifier

```
df_insured_rf=df_original.copy()
df_insured_rf.head()
```

9]:

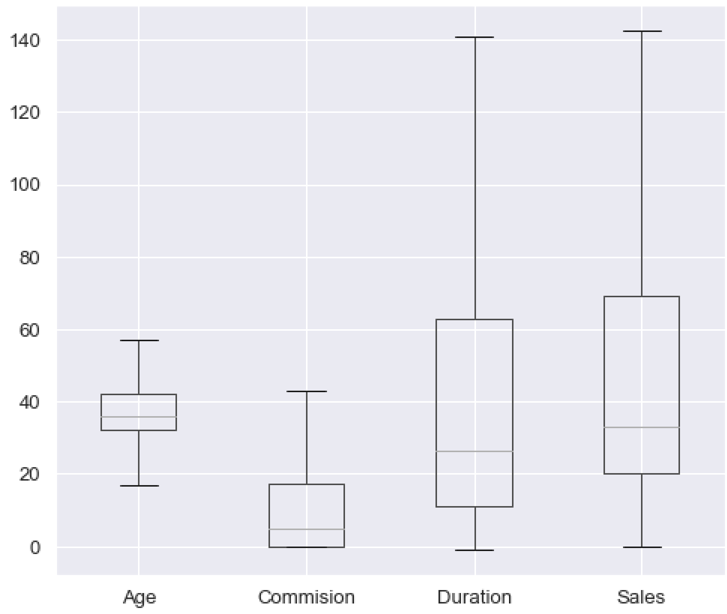
	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	Destination
0	48	C2B	Airlines	No	0.70	Online	7	2.51	Customised Plan	ASIA
1	36	EPX	Travel Agency	No	0.00	Online	34	20.00	Customised Plan	ASIA
2	39	CWT	Travel Agency	No	5.94	Online	3	9.90	Customised Plan	Americas
3	36	EPX	Travel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

TREATING OUTLIERS FOR RANDOM FOREST

```
def treat_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

```
for feature in df_insured_rf[['Age','Commision', 'Duration', 'Sales']]:
    lr,ur=treat_outlier(df_insured_rf[feature])
    df_insured_rf[feature]=np.where(df_insured_rf[feature]>ur,ur,df_insured_rf[feature])
    df_insured_rf[feature]=np.where(df_insured_rf[feature]<lr,lr,df_insured_rf[feature])
```

BOX PLOT TO CHECK PRESENCE OF OUTLIERS



RANDOM FOREST CLASSIFIER

```
X_train, X_test, train_labels, test_labels = train_test_split(X_rf, y_rf, test_size=.30, random_state=1)
```

```
rfcl = RandomForestClassifier(n_estimators = 100,max_features=6,random_state=1)
rfcl = rfcl.fit(X_train, train_labels)
```

```
rfcl
```

TO FIND OPTIMAL NUMBERS USING GRID SEARCH

Grid Search for finding out the optimal values for the hyper parameters

```
param_grid_rfcl = {
    'max_depth': [6],#20,30,40
    'max_features': [4],## 7,8,9
    'min_samples_leaf': [8],## 50,100
    'min_samples_split': [45], ## 60,70
    'n_estimators': [100] ## 100,200
}

rfcl = RandomForestClassifier(random_state=1)

grid_search_rfcl = GridSearchCV(estimator = rfcl, param_grid = param_grid_rfcl, cv = 10)
```

FIFTING THE MODEL TO RFCL VALUES OBTAINED BY OPTIMAL GRIDSEARCH METHOD

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

```
GridSearchCV(cv=10, estimator=RandomForestClassifier(random_state=1),
             param_grid={'max_depth': [6], 'max_features': [4],
                         'min_samples_leaf': [8], 'min_samples_split': [45],
                         'n_estimators': [100]})
```

BEST GRID VALUES

```
best_grid_rf
```

```
]: RandomForestClassifier(max_depth=6, max_features=4, min_samples_leaf=8,
                          min_samples_split=45, random_state=1)
```

Predicting on Training dataset for Random Forest

```
ytrain_predict_rf = best_grid_rf.predict(X_train)
```

```
ytest_predict_rf = best_grid_rf.predict(X_test)
```

MODEL 3

Building a Neural Network Classifier

BEFORE BUILDING THE MODEL

WE SCALE THE VALUES, TO STANDARD SCALE USING MINMAXSCALER

```
sc = StandardScaler()
```

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

AFTER SCALING WE ARE TRANSFORMING THE SAME TO THE TEST DATA

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

MLP CLASSIFIER

```
clf = MLPClassifier(hidden_layer_sizes=100, max_iter=5000,
                    solver='sgd', verbose=True,  random_state=21,tol=0.01)
```

TRAINING THE MODEL

```
clf.fit(X_trains, train_labels)
```

```
Iteration 1, loss = 0.64244509
Iteration 2, loss = 0.62392631
Iteration 3, loss = 0.60292414
Iteration 4, loss = 0.58458220
Iteration 5, loss = 0.56914550
Iteration 6, loss = 0.55651481
Iteration 7, loss = 0.54598011
Iteration 8, loss = 0.53752961
Iteration 9, loss = 0.53051147
Iteration 10, loss = 0.52440802
Iteration 11, loss = 0.51934384
Iteration 12, loss = 0.51483466
Iteration 13, loss = 0.51108343
Iteration 14, loss = 0.50763356
Iteration 15, loss = 0.50476577
Iteration 16, loss = 0.50218466
Iteration 17, loss = 0.49989583
Iteration 18, loss = 0.49786338
Training loss did not improve more than tol=0.010000 for 10 consecutive epochs. Stopping.
```

GRID SEARCH

Grid Search for finding out the optimal values for the hyper parameters

```
param_grid = {
    'hidden_layer_sizes': [200], # 50, 200
    'max_iter': [2500], #5000,2500
    'solver': ['adam'], #sgd
    'tol': [0.01],
}

nncl = MLPClassifier(random_state=1)

grid_search = GridSearchCV(estimator = nncl, param_grid = param_grid, cv = 10)
```

FITTING THE MODEL USING THE OPTIMAL VALUES FROM GRID SEARCH

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

```
GridSearchCV(cv=10, estimator=MLPClassifier(random_state=1),
             param_grid={'hidden_layer_sizes': [200], 'max_iter': [2500],
                         'solver': ['adam'], 'tol': [0.01]})
```

BEST GRID VALUES,

```
best_grid_ann= grid_search.best_estimator_
best_grid_ann
```

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

Predicting on Training dataset for Neural Network Classifier

```
ytrain_predict = best_grid_ann.predict(X_trains)
ytest_predict = best_grid_ann.predict(X_tests)
```

2.3 Performance Metrics: Check the performance of Predictions onTrain and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score for each model

DECISSION TREE PREDICTION

Predicting on Training dataset for Decission Tree

```
: ytrain_predict_dt = reg_dt_model.predict(X_train)
```

```
: print('ytrain_predict',ytrain_predict_dt.shape)
```

```
ytrain_predict (2100,)
```

ACCURACY

```
cart_train_acc = reg_dt_model.score(X_train,train_labels)
cart_train_acc
```

```
4]: 0.7933333333333333
```

CONFUSION MATRIX

```
print(classification_report(train_labels, ytrain_predict_dt))
```

```
              precision    recall  f1-score   support

           0       0.84      0.87      0.85      1471
           1       0.67      0.62      0.64       629

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

```
confusion_matrix(train_labels, ytrain_predict_dt)
```

```
3]: array([[1275,  196],
           [ 238,  391]], dtype=int64)
```

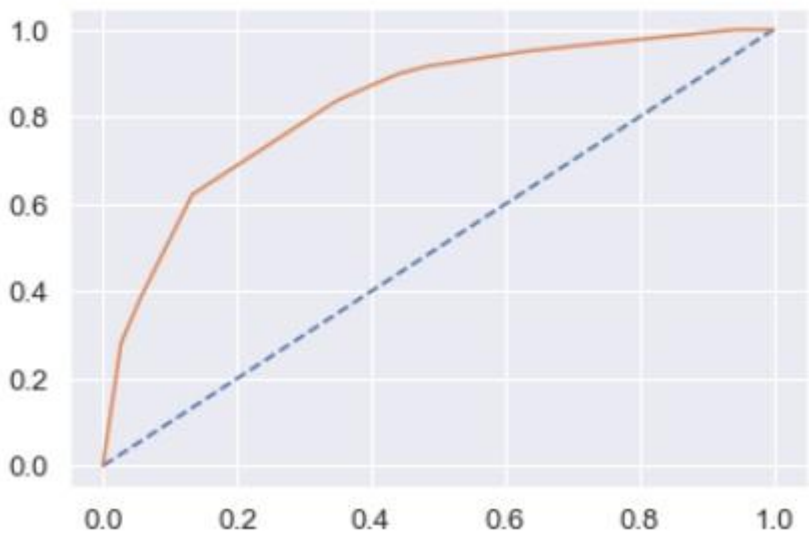
Model Evaluation for Decision Tree

AUC and ROC for the training data for Decision Tree

```
# predict probabilities
probs = reg_dt_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
cart_train_auc = roc_auc_score(train_labels, probs)
print('AUC: %.3f' % cart_train_auc)
# calculate roc curve
cart_train_fpr, cart_train_tpr, cart_train_thresholds = roc_curve(train_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(cart_train_fpr, cart_train_tpr)
```

```
AUC: 0.827
```

```
[<matplotlib.lines.Line2D at 0x22d36d29fc8>]
```




```
cart_metrics=classification_report(train_labels, ytrain_predict_dt,output_dict=True)
df=pd.DataFrame(cart_metrics).transpose()
cart_train_f1=round(df.loc["1"][2],2)
cart_train_recall=round(df.loc["1"][1],2)
cart_train_precision=round(df.loc["1"][0],2)
print ('cart_train_precision ',cart_train_precision)
print ('cart_train_recall ',cart_train_recall)
print ('cart_train_f1 ',cart_train_f1)
```

```
cart_train_precision  0.67
cart_train_recall  0.62
cart_train_f1  0.64
```

AUC and ROC for the test data for Decission Tree

```
# predict probabilities
probs = reg_dt_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
cart_test_auc = roc_auc_score(test_labels, probs)
print('AUC: %.3f' % cart_test_auc)
# calculate roc curve
cart_test_fpr, cart_test_tpr, cart_testthresholds = roc_curve(test_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(cart_test_fpr, cart_test_tpr)
```

```
AUC: 0.790
```

```
cart_metrics=classification_report(test_labels, ytest_predict_dt,output_dict=True)
df=pd.DataFrame(cart_metrics).transpose()
cart_test_precision=round(df.loc["1"][0],2)
cart_test_recall=round(df.loc["1"][1],2)
cart_test_f1=round(df.loc["1"][2],2)
print ('cart_test_precision ',cart_test_precision)
print ('cart_test_recall ',cart_test_recall)
print ('cart_test_f1 ',cart_test_f1)
```

```
cart_test_precision  0.71
cart_test_recall  0.53
cart_test_f1  0.6
```

MODEL 2 PREDICTION RANDOM FOREST

Predicting on Training dataset for Random Forest

```
ytrain_predict_rf = best_grid_rf.predict(X_train)
```

```
print('ytrain_predict',ytrain_predict_rf.shape)
```

```
ytrain_predict (2100,)
```

ACCURACY

```
rf_train_acc = best_grid_rf.score(X_train,train_labels)
rf_train_acc
```

```
5]: 0.8123809523809524
```

CONFUSION MATRIX

```
print(classification_report(train_labels, ytrain_predict_rf))
```

```
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1471
           1       0.73      0.60      0.66       629

    accuracy                           0.81      2100
   macro avg       0.78      0.75      0.76      2100
weighted avg       0.81      0.81      0.81      2100
```

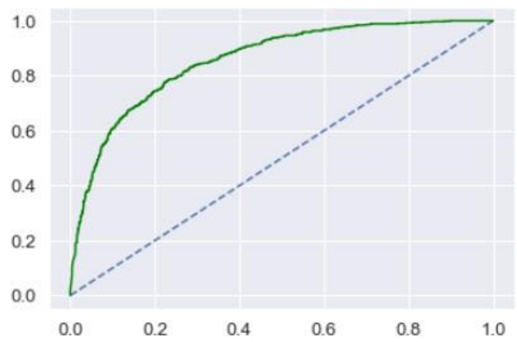
```
confusion_matrix(train_labels, ytrain_predict_rf)
```

```
array([[1331,  140],
       [ 254,  375]], dtype=int64)
```

Model Evaluation for Random Forest

AUC and ROC for the training data for Random Forest

```
# predict probabilities
probs = best_grid_rf.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
rf_train_auc = roc_auc_score(train_labels, probs)
print('AUC: %.3f' % rf_train_auc)
# calculate roc curve
rf_train_fpr, rf_train_tpr, rf_train_thresholds = roc_curve(train_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(rf_train_fpr, rf_train_tpr, color='green')
```



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

```
rf_train_precision  0.73
rf_train_recall  0.6
rf_train_f1  0.66
```

Predicting on Test dataset for Random Forest

```
ytest_predict_rf = best_grid_rf.predict(X_test)
```

```
print('ytest_predict_rf',ytest_predict_rf.shape)
```

```
ytest_predict_rf (900,)
```

ACCURACY

```
rf_test_acc = best_grid_rf.score(X_test,test_labels)
rf_test_acc
```

```
0.7733333333333333
```

CONFUSION MATRIX

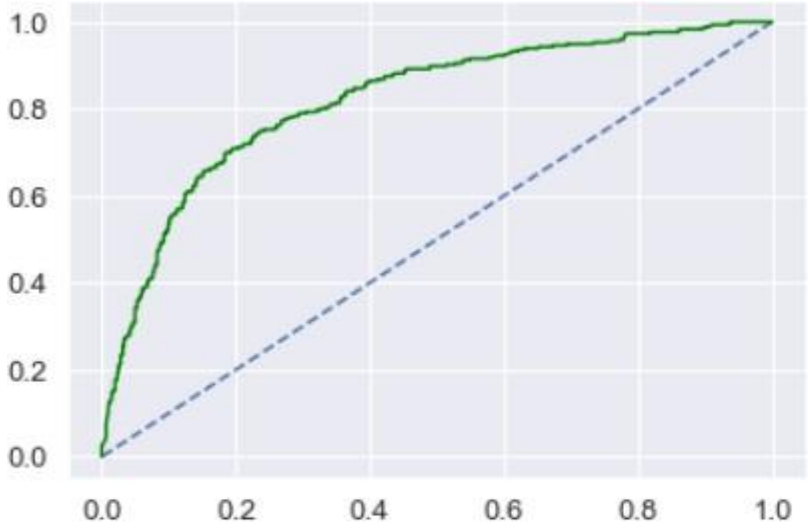
```
print(classification_report(test_labels, ytest_predict_rf))
```

```
              precision    recall  f1-score   support

           0       0.79      0.91      0.84       605
           1       0.73      0.49      0.59       295

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

AUC and ROC for the test data for Random Forest

```
# predict probabilities
probs = best_grid_rf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
rf_test_auc = roc_auc_score(test_labels, probs)
print('AUC: %.3f' % rf_test_auc)
# calculate roc curve
rf_test_fpr, rf_test_tpr, rf_testthresholds = roc_curve(test_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(rf_test_fpr, rf_test_tpr, color='green')
```

```
AUC: 0.818
```




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

```
rf_test_precision  0.73
rf_test_recall  0.49
rf_test_f1  0.59
```

MODEL 3

ANN

Predicting on Training dataset for Neural Network Classifier

```
ytrain_predict_ann = best_grid_ann.predict(X_trains)
```

```
print('ytrain_predict',ytrain_predict_ann.shape)
```

```
ytrain_predict (2100,)
```

CONFUSION MATRIX

```
confusion_matrix(train_labels,ytrain_predict_ann)
```

```
]: array([[1317,  154],
          [ 303,  326]], dtype=int64)
```

ACCURACY

```
ann_train_acc=best_grid_ann.score(X_trains,train_labels)
ann_train_acc
```

```
: 0.7823809523809524
```

```
print(classification_report(train_labels,ytrain_predict_ann))
```

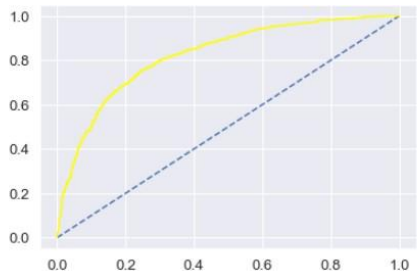
```
              precision    recall  f1-score   support

           0       0.81      0.90      0.85      1471
           1       0.68      0.52      0.59       629

    accuracy                           0.78      2100
   macro avg       0.75      0.71      0.72      2100
weighted avg       0.77      0.78      0.77      2100
```

Model Evaluation for Neural Network Classifier

AUC and ROC for the training data for Neural Network Classifier

```
# predict probabilities
probs = best_grid_ann.predict_proba(X_trains)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
ann_train_auc = roc_auc_score(train_labels, probs)
print('AUC: %.3f' % ann_train_auc)
# calculate roc curve
ann_train_fpr, ann_train_tpr, ann_train_thresholds = roc_curve(train_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(ann_train_fpr, ann_train_tpr, color='yellow')
```

```
AUC: 0.823
```



Predicting on Test dataset for Neural Network Classifier

```
ytest_predict_ann = best_grid_ann.predict(X_tests)
```

```
print('ytest_predict_ann',ytest_predict_ann.shape)
```

```
ytest_predict_ann (900,)
```

ACCURACY

```
ann_test_acc = best_grid_ann.score(X_tests,test_labels)
ann_test_acc
```

```
0.7622222222222222
```

CONFUSION MATRIX

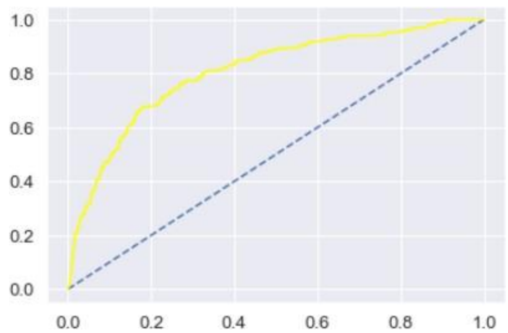
```
confusion_matrix(test_labels, ytest_predict_ann)
```

```
]: array([[557,  48],
          [166, 129]], dtype=int64)
```

AUC and ROC for the test data for Neural Network Classifier

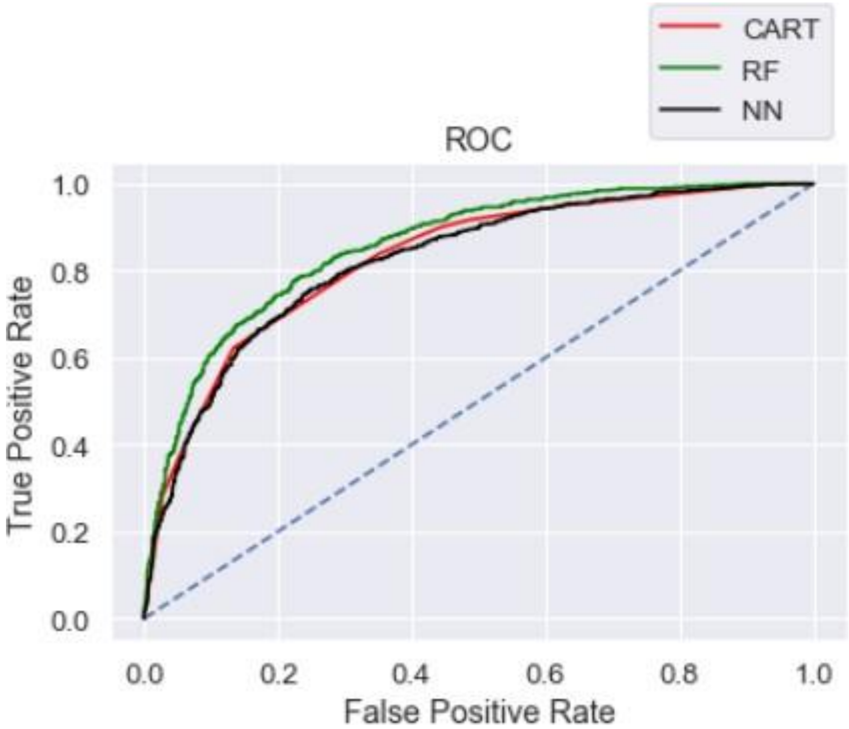
```
# predict probabilities
probs = best_grid_ann.predict_proba(X_tests)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
ann_test_auc = roc_auc_score(test_labels, probs)
print('AUC: %.3f' % ann_test_auc)
# calculate roc curve
ann_test_fpr, ann_test_tpr, ann_testthresholds = roc_curve(test_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(ann_test_fpr, ann_test_tpr, color='yellow')
```

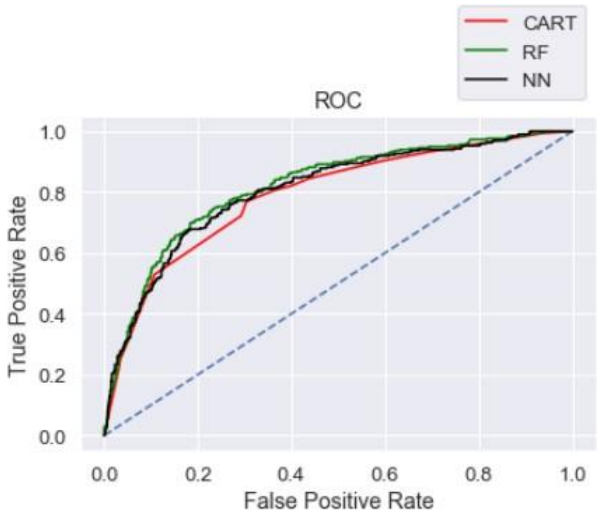
```
AUC: 0.806
```



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

	CART Train	CART Test	Random Forest Train	Random Forest Test	Neural Network Train	Neural Network Test
Accuracy	0.79	0.77	0.81	0.77	0.78	0.76
AUC	0.83	0.79	0.86	0.82	0.82	0.81
Recall	0.62	0.53	0.60	0.49	0.52	0.44
Precision	0.67	0.71	0.73	0.73	0.68	0.73
F1 Score	0.64	0.60	0.66	0.59	0.59	0.55





Conclusion:

The RF model has better accuracy, precision, recall, and f1 score better than CART & NN.

2.5 Inference: Based on the whole Analysis, what are the businessinsights and recommendations?

Looking at the model, more data will help us understand and predict models better. Streamlining online experiences benefitted customers, leading to an increase in conversions,which subsequently raised profits. As per the data 90% of insurance is done by online channel. There is a need to train the JZI agency resources to pick up sales as they are in bottom, need to runpromotional marketing campaign or evaluate if we need to tie up with alternate agency. Based on the model we are getting 80%accuracy, so we need customer books airlinetickets or plans, cross sell the insurance based on the claim data pattern. There are more sales happen via Agency than Airlines and the trend shows theclaim are processed more at Airline.

Key performance indicators (KPI) The KPI's of insurance claims are:

Reduce claim handling costs.
Increase customer satisfaction which in fact will give more revenue
Combat fraud transactions, deploy measures to avoid fraudulent transactions at earliest
Optimize claims recovery method

END