Problem 1: Clustering

Problem Statement:

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

Data Dictionary for Market Segmentation: ¶

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

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

Importing all required libraries:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:
```
#Reading the dataset

bank_df = pd.read_csv('D:\\SHUBHANK !\\GL\\5. TOPIC 4 - Data Mining\\Final Pro
ject\\bank_marketing_part1_Data.csv')
```

In [3]:
```
#Head of the dataset

bank_df.head()
```

Out[3]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_p
0	19.94	16.92	0.8752	6.675	3.763	
1	15.99	14.89	0.9064	5.363	3.582	
2	18.95	16.42	0.8829	6.248	3.755	
3	10.83	12.96	0.8099	5.278	2.641	
4	17.99	15.86	0.8992	5.890	3.694	

In [4]:
```
#Shape of the dataset

bank_df.shape

print('Number of rows    :', bank_df.shape[0])
print('Number of columns :', bank_df.shape[1])
```

```
Number of rows    : 210
Number of columns : 7
```

In [5]:
```
#Describing the dataset

bank_df.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75
spending	210.0	14.847524	2.909699	10.5900	12.27000	14.35500	17.3050
advance_payments	210.0	14.559286	1.305959	12.4100	13.45000	14.32000	15.7150
probability_of_full_payment	210.0	0.870999	0.023629	0.8081	0.85690	0.87345	0.8877
current_balance	210.0	5.628533	0.443063	4.8990	5.26225	5.52350	5.9797
credit_limit	210.0	3.258605	0.377714	2.6300	2.94400	3.23700	3.5617
min_payment_amt	210.0	3.700201	1.503557	0.7651	2.56150	3.59900	4.7687
max_spent_in_single_shopping	210.0	5.408071	0.491480	4.5190	5.04500	5.22300	5.8770

In [6]:
```
#Checking the null values

bank_df.isnull().sum()
```

Out[6]:
```
spending                        0
advance_payments                0
probability_of_full_payment     0
current_balance                 0
credit_limit                    0
min_payment_amt                 0
max_spent_in_single_shopping    0
dtype: int64
```

There are no null values present in the dataset

In [7]:
```
#Checking duplicates:

dups = bank_df.duplicated()

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

```
Number of duplicate rows = 0
```

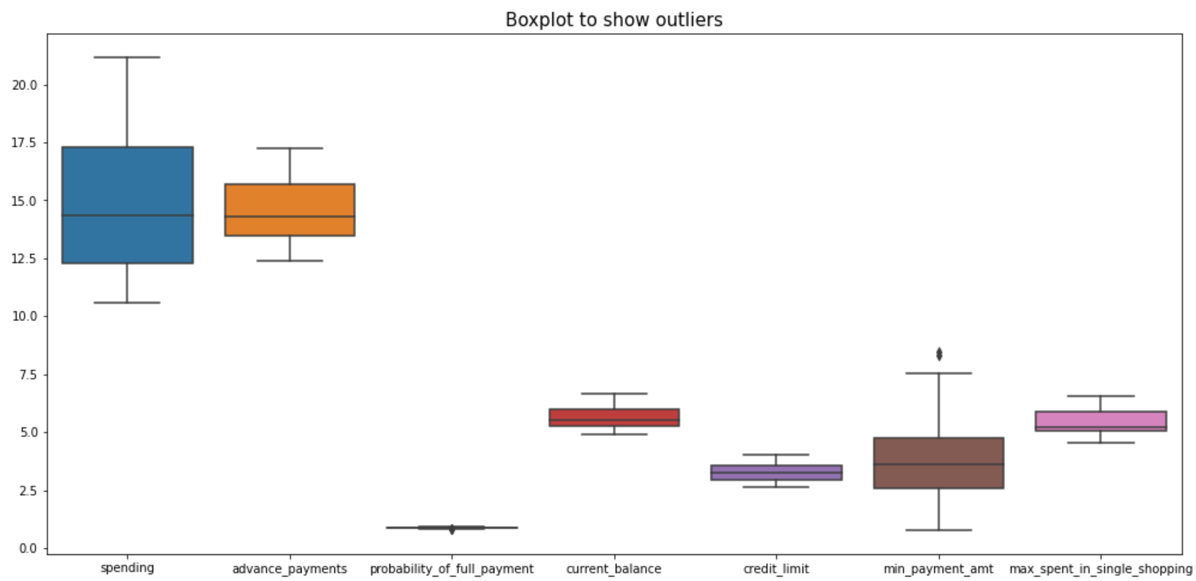
In [8]:
```
#Getting the info

bank_df.info()
```

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

There are no null values in the dataset and all the variables have the same data types(float64).

In [9]:
```python
#Boxplot

plt.figure(figsize=(17,8))

plt.title("Boxplot to show outliers", fontsize= 15)

sns.boxplot(data=bank_df)
```

Out[9]: <AxesSubplot:title={'center':'Boxplot to show outliers'}>



Since, we have performed boxplot on non-scaled data, thats why there is no proper distribution of the variables.

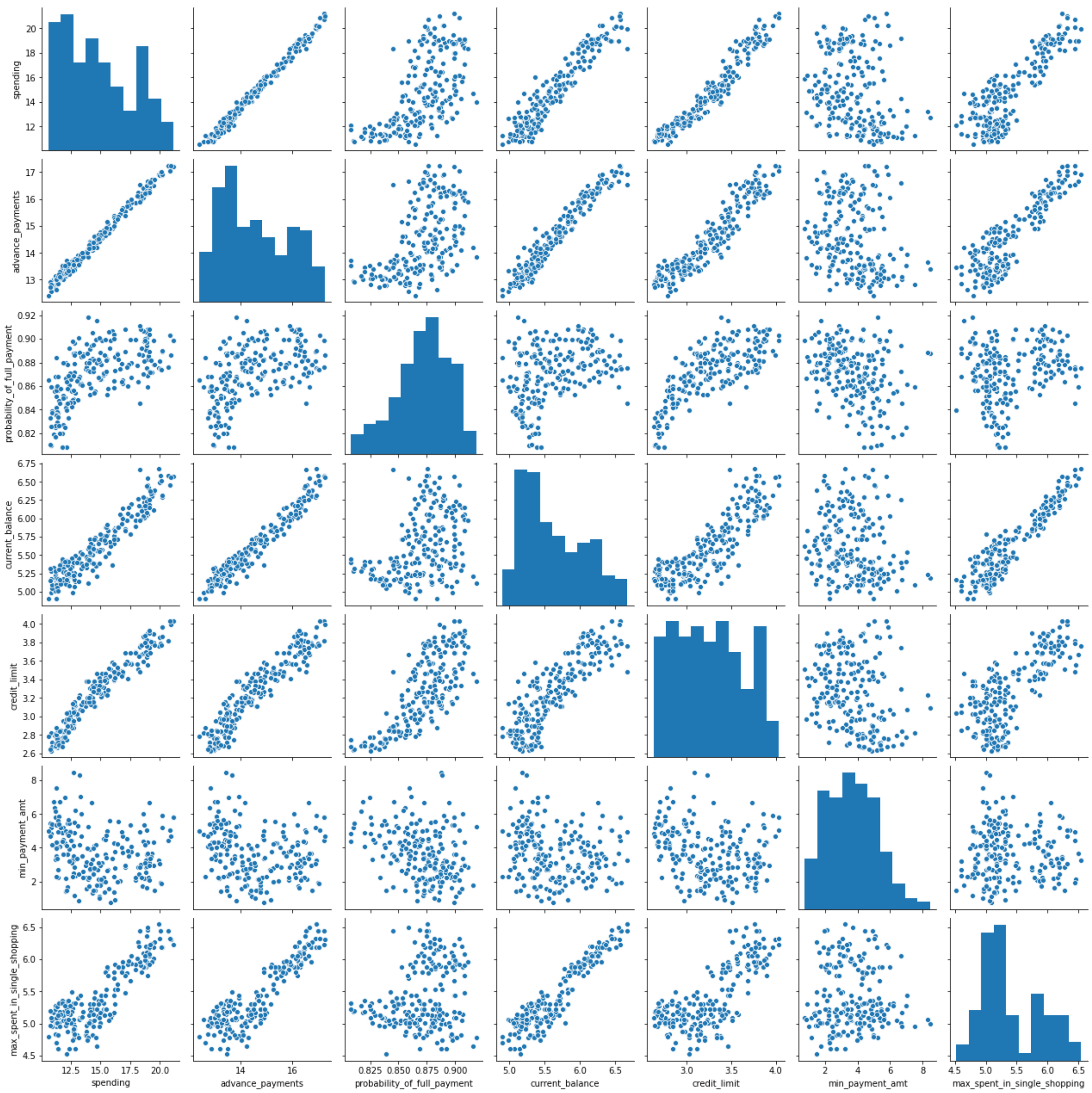
In [10]:
```python
#Correlation between the variables

corr = bank_df.corr(method='pearson')

corr
```

Out[10]:

	spending	advance_payments	probability_of_full_payment	current
spending	1.000000	0.994341	0.608288	
advance_payments	0.994341	1.000000	0.529244	
probability_of_full_payment	0.608288	0.529244	1.000000	
current_balance	0.949985	0.972422	0.367915	
credit_limit	0.970771	0.944829	0.761635	
min_payment_amt	-0.229572	-0.217340	-0.331471	
max_spent_in_single_shopping	0.863693	0.890784	0.226825	

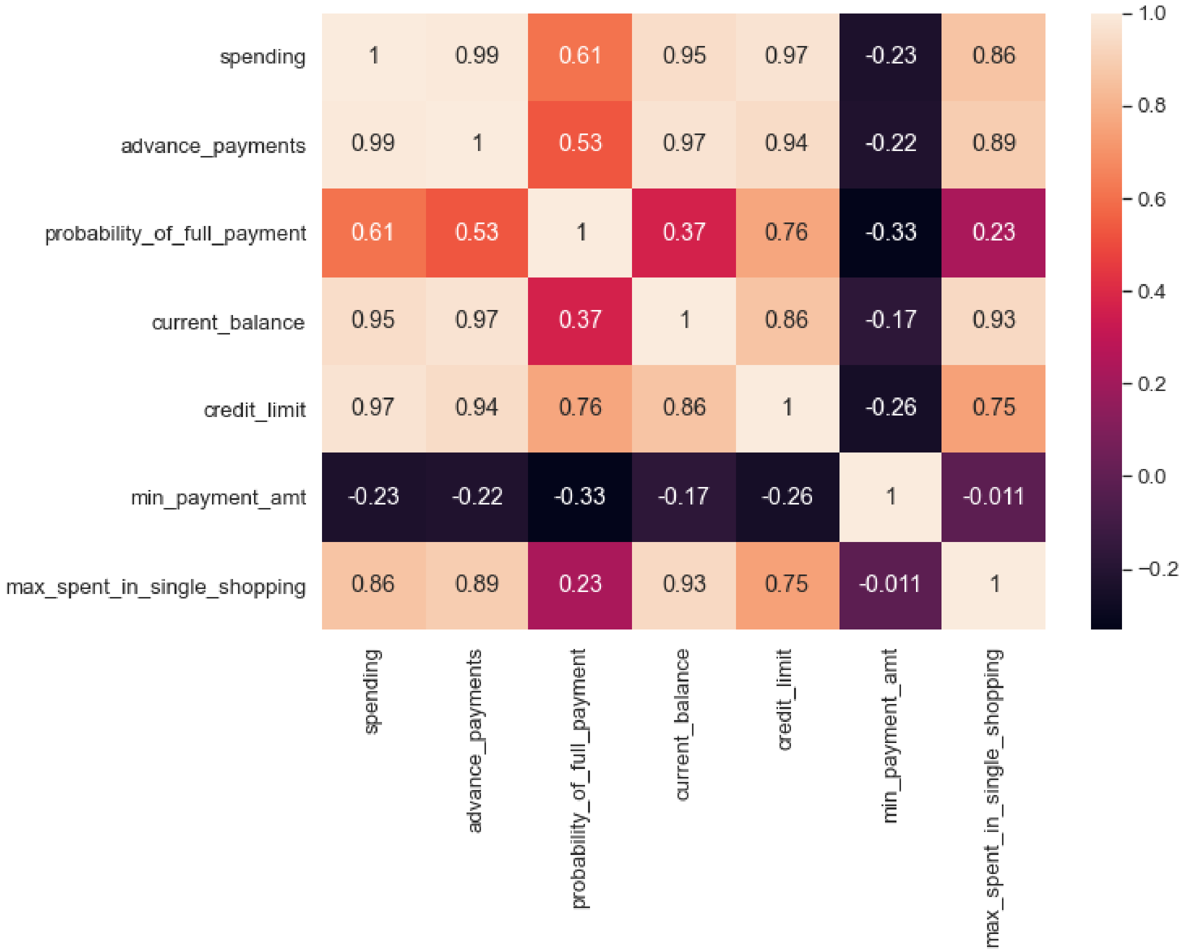
In [11]:
```
#Using Pairplot, to see how the variables correlate with each other:

sns.pairplot(data= bank_df)
```

Out[11]: <seaborn.axisgrid.PairGrid at 0x1552b4b18e0>



In [12]:
```
#Heatmap for correlation:

plt.figure(figsize=(10,7))

sns.set(font_scale=1.2)

sns.heatmap((bank_df).corr(), annot=True)
```

Out[12]: <AxesSubplot:>



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

To bring all features in the same standing, we need to do scaling in this case, so that one significant number doesn't impact the further calculations just because of their large magnitude.

We have variables in our dataset which are measured in different scales, some are in 100's , some in 1000's and some in 10000's, so it is useful to scale the data.

We are using 'zscore' from 'scipy.stats' to scale our dataset.

z scores indicate how many standard deviations an observation is above or below the mean. These scores are a useful way of putting data from different sources onto the same scale.

In [13]:
```
#Scaling the 'bank_df' dataset using zscore

from scipy.stats import zscore

scaled_df = bank_df.apply(zscore)
```

In [14]:
```
#Checking the head of the scaled data

scaled_df.head()
```

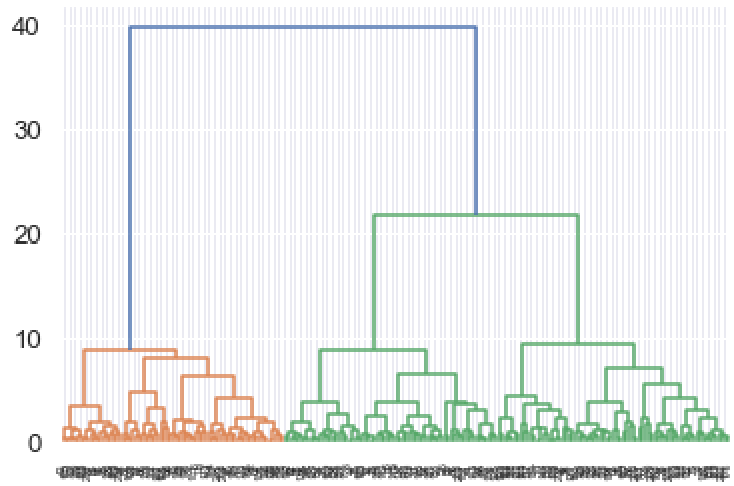
Out[14]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_p
0	1.754355	1.811968	0.178230	2.367533	1.338579	
1	0.393582	0.253840	1.501773	-0.600744	0.858236	
2	1.413300	1.428192	0.504874	1.401485	1.317348	
3	-1.384034	-1.227533	-2.591878	-0.793049	-1.639017	
4	1.082581	0.998364	1.196340	0.591544	1.155464	

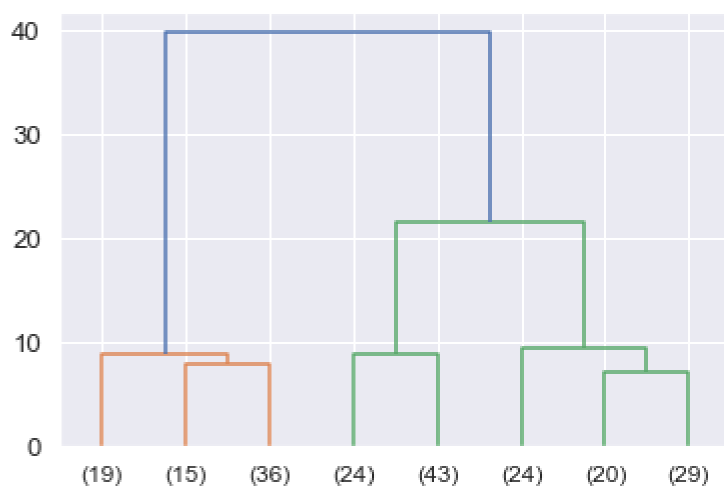
1.3 Apply hierarchical clustering to scaled data. Identify the number of optimum clusters using Dendrogram and briefly describe them.

Applying Hierarchical clustering to scaled data 'scaled_df'

In [15]:
```
#First Importing 'Dendrogram' and 'Linkage' from scipy.cluster.hierarchy

from scipy.cluster.hierarchy import dendrogram, linkage

#Using 'ward' method

wardlink = linkage(scaled_df, method = 'ward')

#Creating Dendrogram

dend = dendrogram(wardlink)
```



In [16]:
```
dend = dendrogram(wardlink, truncate_mode='lastp', p = 8)
```



Here, truncate_mode='lastp', shows only the last p merged clusters and p=8, shows only the last 8 merged clusters.

In [17]:
```
from scipy.cluster.hierarchy import fcluster
```

In [18]:
```
#We are using 'maxclust' as criterion  for making clusters

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

clusters
```

Out[18]:
```
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)
```

In [19]:
```
#Adding the clusters columns in the dataset

bank_df['clusters'] = clusters
```

In [20]:
```
bank_df
```

Out[20]:

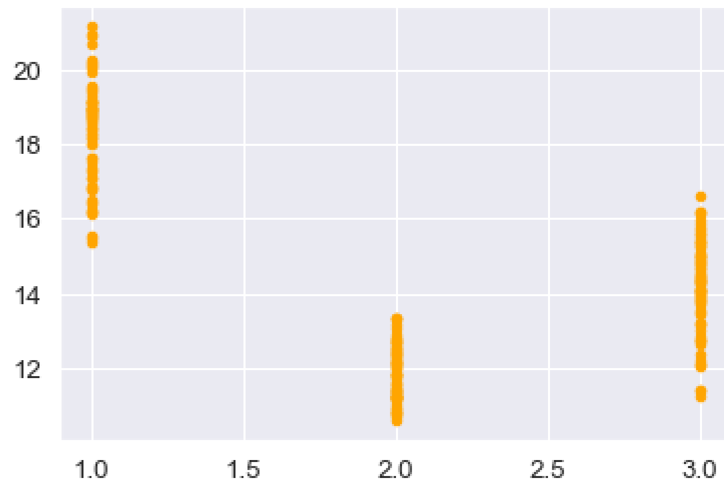
	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min
0	19.94	16.92	0.8752	6.675	3.763	
1	15.99	14.89	0.9064	5.363	3.582	
2	18.95	16.42	0.8829	6.248	3.755	
3	10.83	12.96	0.8099	5.278	2.641	
4	17.99	15.86	0.8992	5.890	3.694	
...	...	...	...	...	...	
205	13.89	14.02	0.8880	5.439	3.199	
206	16.77	15.62	0.8638	5.927	3.438	
207	14.03	14.16	0.8796	5.438	3.201	
208	16.12	15.00	0.9000	5.709	3.485	
209	15.57	15.15	0.8527	5.920	3.231	

210 rows × 8 columns

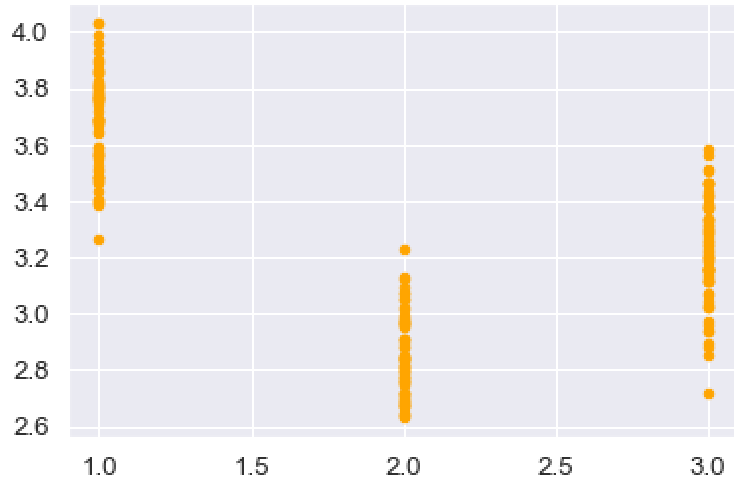
In [21]:
```
x = bank_df.clusters

def customer_segmentation(column,nbins):
    print("Description of :" + column)
    print("-------------------------------------------")
    print(bank_df[column].describe(),end=' ')


    plt.figure()
    print("\n\nCustomer segmentation with respect to :" + column)
    print("-------------------------------------------")
    plt.scatter(x,bank_df[column], s=20, c='orange');
    plt.show()
```

In [22]:
```
customer_segmentation('spending',20)


print('\n\n_____________________________________________________________________________________
_____\n\n')

customer_segmentation('credit_limit',20)

print('\n\n_____________________________________________________________________________________
_____\n\n')

customer_segmentation('max_spent_in_single_shopping',20)
```

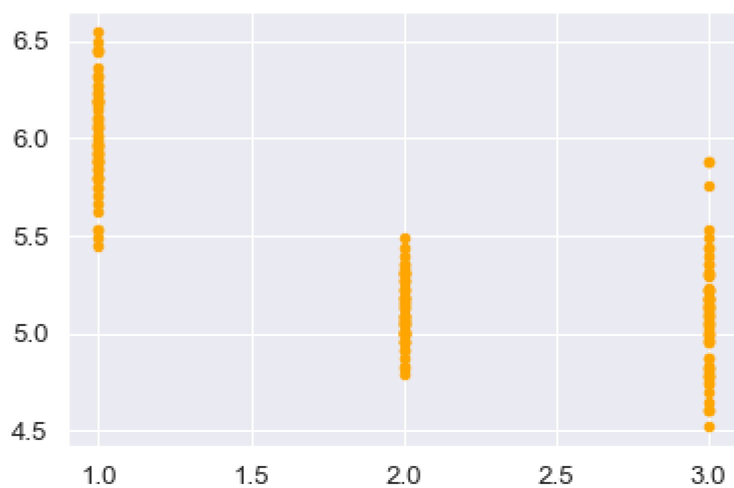
```
Description of :spending
-------------------------------------------
count    210.000000
mean      14.847524
std        2.909699
min       10.590000
25%       12.270000
50%       14.355000
75%       17.305000
max       21.180000
Name: spending, dtype: float64

Customer segmentation with respect to :spending
-------------------------------------------
```



```
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

Customer segmentation with respect to :credit_limit
-------------------------------------------
```



```
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

Customer segmentation with respect to :max_spent_in_single_shopping
--------------------------------------------
```



We can see maximum monthly expenditure is done by the customers of cluster 1. The maximum amount that spent monthly is around 21000 by the user from cluster 1. The minimum amount spent monthly is around 10000 from the user of cluster 2.

Maximum amount spent in single shopping is also done by the user of cluster 1 that is around 6500. Minimum amount spent in single shopping is done by the user of cluster 3 that is around 4500.

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.

In [23]:

```
#First, Importing KMeans from sklearn.cluster

from sklearn.cluster import KMeans
```

In [24]:

```
#Now we need to check inertia for different number of clusters and appending t
hem into the variable 'wss',
#to get the optimul numbers of clusters using KMeans

#within sum of squares(wss)

wss =[]

#We iterate the values of k from 1 to 7 and calculate the values of distortion
s for each value of k
#and calculate the distortion and inertia for each value of k in the given ran
ge

for k in range(1,8):
    k_means = KMeans(n_clusters=k)
    k_means.fit(scaled_df)
    wss.append(k_means.inertia_)

#NOTE: Inertia is the sum of squared distances of samples to their closest clu
ster center.
```

In [25]:
```
#Checking all the inertia(within sum of squares) from 1 to 7 clusters for the
 scaled data

#Within Sum of Squares(wss)

wss
```

Out[25]:
```
[1469.9999999999998,
 659.171754487041,
 430.6589731513006,
 371.1846125351018,
 327.3281094192775,
 289.46717056412893,
 264.3052234087485]
```

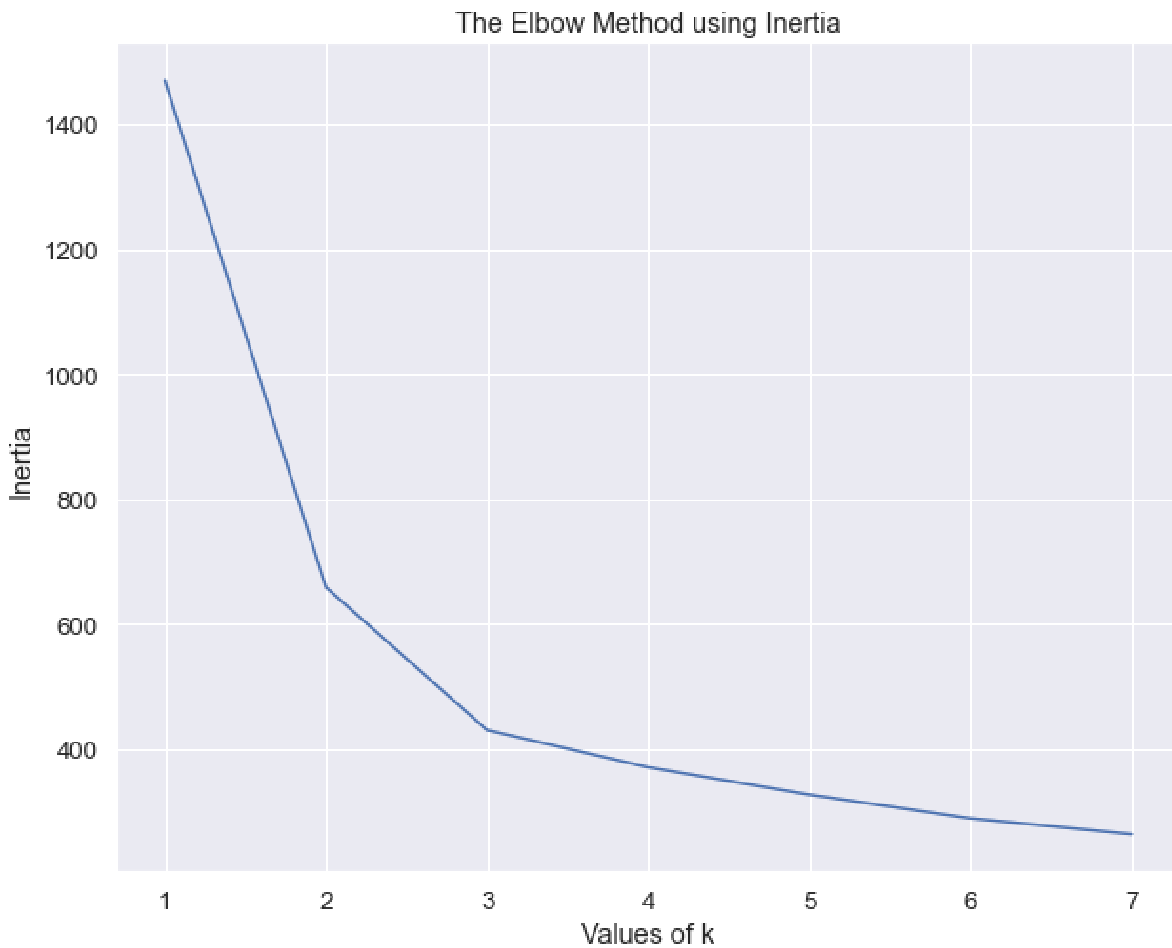
In [26]:
```
print('For k=1, the inertia is', wss[0])
print('For k=2, the inertia is', wss[1])
print('For k=3, the inertia is', wss[2])
print('For k=4, the inertia is', wss[3])
print('For k=5, the inertia is', wss[4])
print('For k=6, the inertia is', wss[5])
print('For k=7, the inertia is', wss[6])
```

```
For k=1, the inertia is 1469.9999999999998
For k=2, the inertia is 659.171754487041
For k=3, the inertia is 430.6589731513006
For k=4, the inertia is 371.1846125351018
For k=5, the inertia is 327.3281094192775
For k=6, the inertia is 289.46717056412893
For k=7, the inertia is 264.3052234087485
```

We can see, there is significant drop in the values from 'k=1' till 'k=3', and after that there is not much difference in the values as we go further from 4th to 7th.

But still we see it on the elbow method using inertia.

In [27]:
```
#Now, we will use elbow method to select the optimum numbers of clusters

plt.figure(figsize=(10,8))

plt.plot(range(1,8), wss)

plt.xlabel('Values of k')

plt.ylabel('Inertia')

plt.title('The Elbow Method using Inertia')

plt.show()
```



To get the optimal number of clusters, we have to select the value of k at the 'elbow' that we can see in the above plot. Generally, 'elbow' is the point after which the values of the inertia start decreasing in a linear fashion.

Similarly, in our above elbow plot/method we can see that the values of inertia dropping slowly after k=3.

Therefore for our dataset, we conclude that the optimal number of clusters for the data is 3.

In [28]:
```
#taking 3 as number of clusters and defining the labels to our dataset

k_means = KMeans(n_clusters = 3)
k_means.fit(scaled_df)
labels = k_means.labels_
```

In [29]:
```
#alloting the column 'k_means_clusters' our dataset and labelling it

bank_df["k_means_clusters"] = labels
```

In [30]:
```
bank_df.head()
```

Out[30]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_p
0	19.94	16.92	0.8752	6.675	3.763	
1	15.99	14.89	0.9064	5.363	3.582	
2	18.95	16.42	0.8829	6.248	3.755	
3	10.83	12.96	0.8099	5.278	2.641	
4	17.99	15.86	0.8992	5.890	3.694	

In [31]:
```
# Silhouette Score

from sklearn.metrics import silhouette_samples, silhouette_score
```

In [32]:
```
silhouette_score(scaled_df,labels)
```

Out[32]: 0.4007270552751299

As we can see that the Silhouette score for our data is positive. This means that the clusters are well separated.

In [33]:
```
sil_width = silhouette_samples(scaled_df,labels)
```

In [34]:
```
sil_width
```

Out[34]:
```
array([0.57369874, 0.36638639, 0.63778363, 0.51245819, 0.36227633,
       0.21844638, 0.4728666 , 0.36181217, 0.52028453, 0.5325168 ,
       0.46759191, 0.13224116, 0.38966769, 0.5247812 , 0.11221528,
       0.22129574, 0.33795723, 0.49990157, 0.03155344, 0.2357566 ,
       0.35903729, 0.36612754, 0.43277307, 0.26136159, 0.47570507,
       0.06575223, 0.2717924 , 0.50389413, 0.55352814, 0.43430599,
       0.37707319, 0.42823997, 0.38827268, 0.39498208, 0.5345933 ,
       0.55628078, 0.50760384, 0.42334973, 0.50496507, 0.62241469,
       0.56053376, 0.48652307, 0.39923175, 0.61098901, 0.51352958,
       0.37606912, 0.30715373, 0.58258949, 0.48825724, 0.53403992,
       0.31448221, 0.49548458, 0.58601272, 0.59926567, 0.61967102,
       0.23378798, 0.44189877, 0.5384123 , 0.57674252, 0.57696905,
       0.55410258, 0.51383032, 0.55412974, 0.28131787, 0.49622138,
       0.56495699, 0.57828489, 0.5237842 , 0.63205238, 0.08288516,
       0.44353914, 0.32042362, 0.54187254, 0.58284321, 0.29226419,
       0.58740222, 0.45274186, 0.45864864, 0.36031781, 0.47235547,
       0.35417435, 0.2831762 , 0.47203593, 0.43332917, 0.54185487,
       0.11223661, 0.22242271, 0.00545677, 0.02979192, 0.16646164,
       0.20517965, 0.5183525 , 0.48637841, 0.46183334, 0.11885986,
       0.47957255, 0.52478745, 0.12866857, 0.5607693 , 0.50116166,
       0.07635312, 0.63928523, 0.35654605, 0.59044189, 0.43933781,
       0.57027048, 0.44769618, 0.27027543, 0.04661235, 0.57498168,
       0.13233096, 0.46436826, 0.53800318, 0.3679253 , 0.51909228,
       0.37156469, 0.4551955 , 0.02350739, 0.55969347, 0.57258487,
       0.09100925, 0.49344017, 0.31608966, 0.23522984, 0.45363846,
       0.47464838, 0.46014082, 0.58243476, 0.5138668 , 0.51914758,
       0.53329198, 0.49191608, 0.126471  , 0.54960064, 0.55440964,
       0.5234821 , 0.46225939, 0.47523201, 0.29475089, 0.3672136 ,
       0.21082087, 0.5124197 , 0.49210569, 0.36077109, 0.00758394,
       0.47903987, 0.50875345, 0.56149935, 0.4665377 , 0.49796266,
       0.2933896 , 0.33914323, 0.55061142, 0.11954055, 0.15438564,
       0.43772026, 0.0147342 , 0.58727725, 0.49253235, 0.50982822,
       0.55039802, 0.16465047, 0.4923075 , 0.40688005, 0.56328221,
       0.52812855, 0.08356374, 0.4883814 , 0.28327002, 0.31463815,
       0.29994534, 0.55293118, 0.5327705 , 0.48314156, 0.54160451,
       0.55177632, 0.45981976, 0.0473607 , 0.08235604, 0.44057444,
       0.48352558, 0.08180233, 0.27528811, 0.405653  , 0.24838999,
       0.34038446, 0.04968614, 0.40448831, 0.36979337, 0.44827555,
       0.00271309, 0.37107701, 0.49526093, 0.54780938, 0.48791268,
       0.26514219, 0.59782639, 0.39559692, 0.6139783 , 0.47242729,
       0.52434091, 0.09698616, 0.51856563, 0.51075769, 0.04663163,
       0.31052936, 0.26754472, 0.5067837 , 0.25736883, 0.04169976])
```

Sil_width is positive, that means the mapping is correct to its centroid.

In [35]:
```
bank_df["sil_width"] = sil_width

bank_df.head(5)
```

Out[35]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_p
0	19.94	16.92	0.8752	6.675	3.763	
1	15.99	14.89	0.9064	5.363	3.582	
2	18.95	16.42	0.8829	6.248	3.755	
3	10.83	12.96	0.8099	5.278	2.641	
4	17.99	15.86	0.8992	5.890	3.694	

In [36]:
```
#Saving and Storing this above dataset into a seperate csv file

bank_df.to_csv('bank_Clustering1.csv')
```

Inference:

We can see that there is not much decline in the values as we go further from 4th to 7th clusters, that's why we have decided to keep the optimum number of cluster equal to 3.

By using KMeans we got the number of clusters is 3 but still for more clarity we can see it on the elbow method using inertia. In the above elbow plot/method also, we can see that the values of inertia dropping significantly after k=3.

The data is well separated within these 3 clusters. This we can say by directly looking at the silhouette score.

Therefore for our dataset, we conclude that the optimal number of clusters for the data is 3.

NOTE: Inertia is the sum of squared distances of samples to their closest cluster center.

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

Observation:

Once the dataset is clustered properly, we can look into the data and say that:

In first cluster, there are the people with credit limit between 30,000 and 40,000 are spending around 6000 to 7000 every time they go to shopping and also their monthly spending is high around 19,000.

In second cluster, there are the people with credit limit between 25,000 and 30,000(around) are spending around 4000 to 5500 every time they go to shopping and also their monthly spending is moderate around 11,000.

In third cluster, there are the people with credit limit around 30,000 are spending around 4000 to 5000 everytime they go to shopping but their monthly spending is less in compared to the people from first cluster, around 14,000 and their probability of full payment is higher than others.

Recommendation:

So based on above observations, my recommmendation is that bank should give attractive offers to the customers present in the third cluster, so that they can utilise their credit limit and spend somewhere around or more than the people of first cluster. This we can say because customers in first and third clusters have approximately same amount of credit limit but there is differnece in their monthly spending.

Also, bank should give promotional offers to the second cluster members, because their monthly spend is less. The bank should collect information about their area of interest, like where they usually visit, which category they often chose to buy while shopping, so that bank can give some descent offers them as per their requirement.

The bank should provide some easy to pay EMI options to pay their credit card bill to the members of second and third clusters, so that they can utilise their credit card amount to some further extent and can pay the bill without any fear.

As the per records after clustering, the bank should focus more on members of third clusters and should give them the promotional offers more often.

-----------------------------------*END---

Problem 2: CART-RF-ANN

Problem Statement:

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.

Attribute Information:

1. Target: Claim Status (Claimed)
2. Code of tour firm (Agency_Code)
3. Type of tour insurance firms (Type)
4. Distribution channel of tour insurance agencies (Channel)
5. Name of the tour insurance products (Product)
6. Duration of the tour (Duration)
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 Read the data, do the necessary initial steps, and exploratory data analysis (Univariate, Bi-variate, and multivariate analysis).

In [37]:
```
#Importing all required libraries and packages

import numpy as np, pandas as pd
import seaborn as sns, matplotlib.pyplot as plt
```

In [38]:
```
#Reading csv file

ins_df = pd.read_csv('D:\\SHUBHANK !\\GL\\5. TOPIC 4 - Data Mining\\Final Proj
ect\\insurance_part2_data.csv')
```

In [39]:
```
#Checking the head of the dataset

ins_df.head()
```

Out[39]:

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

Understanding the data:

In [40]:
```
#Checking the shape of the data

print('The number of rows:', ins_df.shape[0] )
print('\nThe number of columns:', ins_df.shape[1])
```

```
The number of rows: 3000

The number of columns: 10
```

In [41]:
```
#Describing the dataset and looking at the five number summary

ins_df.describe().T
```

Out[41]:

	count	mean	std	min	25%	50%	75%	max
Age	3000.0	38.091000	10.463518	8.0	32.0	36.00	42.000	84.00
Commision	3000.0	14.529203	25.481455	0.0	0.0	4.63	17.235	210.21
Duration	3000.0	70.001333	134.053313	-1.0	11.0	26.50	63.000	4580.00
Sales	3000.0	60.249913	70.733954	0.0	20.0	33.00	69.000	539.00

From the above summary of the dataset, we can observe few details about the data like,

1. Maximum age of the insured is 84 years and minimum age is 8 years.
2. Mean age is around 38.
3. One odd thing we can observe here is that the minimum duration of the tour is -1., but duration can't be negative, so this means it is an outlier, and also maximum duration is 4580, that can also be an outlier in our dataset, that we will treat later.
4. Maximum commision taken by agency is 210.21 and the minimum is 0.

In [42]:
```
#Checking the info about the data

ins_df.info()
```

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

In the dataset, we can see from the info that there are 2 variables with float datatype, 2 with integer datatype, 6 with object or string datatype. There are no null values present.

In [43]:
```
#Checking the null/missing values if any

ins_df.isnull().sum()
```

Out[43]:
```
Age             0
Agency_Code     0
Type            0
Claimed         0
Commision       0
Channel         0
Duration        0
Sales           0
Product Name    0
Destination     0
dtype: int64
```

In [44]:
```
# We need to check for duplicates if any

duplicate_records = ins_df.duplicated()

print('Number of duplicate rows = %d' % (duplicate_records.sum()))

ins_df[duplicate_records]
```

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

Out[44]:

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name
63	30	C2B	Airlines	Yes	15.0	Online	27	60.0	Bronze Plan
329	36	EPX	Travel Agency	No	0.0	Online	5	20.0	Customised Plan
407	36	EPX	Travel Agency	No	0.0	Online	11	19.0	Cancellation Plan
411	35	EPX	Travel Agency	No	0.0	Online	2	20.0	Customised Plan
422	36	EPX	Travel Agency	No	0.0	Online	5	20.0	Customised Plan
...	...	...	...	...	...	...	...	...	...
2940	36	EPX	Travel Agency	No	0.0	Online	8	10.0	Cancellation Plan
2947	36	EPX	Travel Agency	No	0.0	Online	10	28.0	Customised Plan
2952	36	EPX	Travel Agency	No	0.0	Online	2	10.0	Cancellation Plan
2962	36	EPX	Travel Agency	No	0.0	Online	4	20.0	Customised Plan
2984	36	EPX	Travel Agency	No	0.0	Online	1	20.0	Customised Plan

139 rows × 10 columns

As we can see there are 139 duplicate records are present in the dataset.

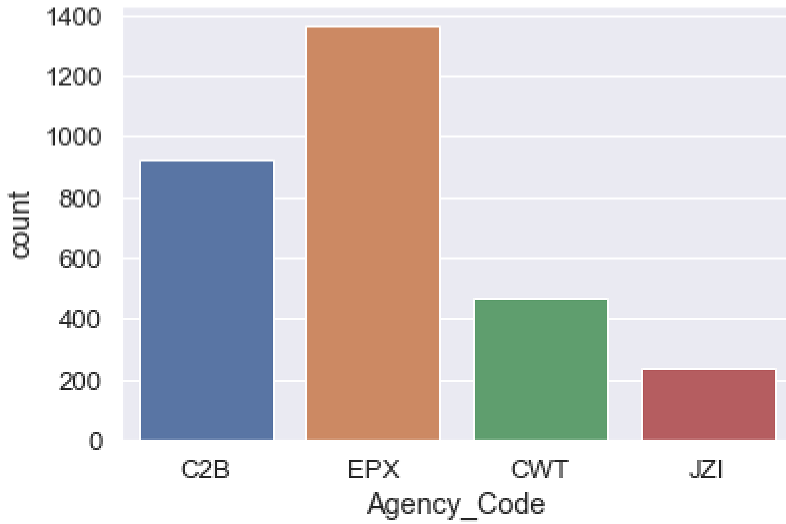
But no need to remove those records because as we can see that there is difference in the values of few variables in every record, not everything is same, for example; there is difference in person's age or the duration of the tour, or tour insurance firms are different or the differnce in product name that one has choosen.

So we are not removing the duplicates from our dataset.

Few Countplot for measure counts

In [45]:
```
sns.countplot(data = ins_df, x = 'Agency_Code')
```

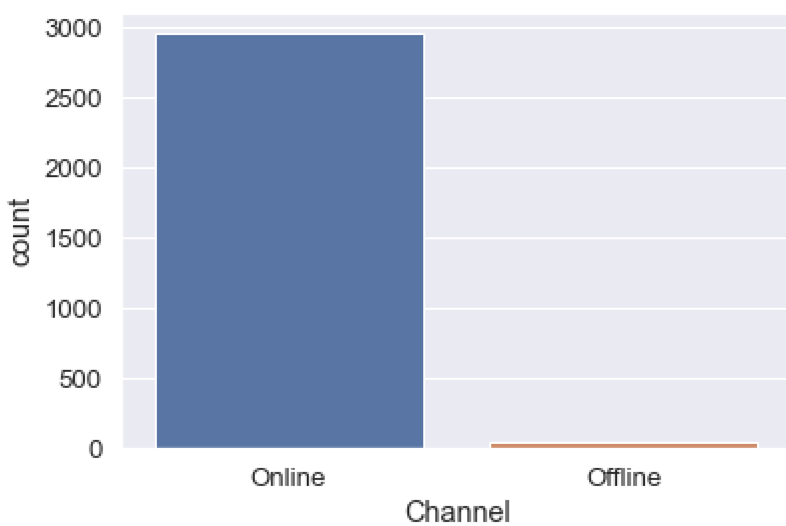
Out[45]: <AxesSubplot:xlabel='Agency_Code', ylabel='count'>



EPX (Agency_Code) has highest number of customers, and JZI has the lowest number of customers.

In [46]:
```
#Countplot for 'Channel'
sns.countplot(data = ins_df, x = 'Channel')
```

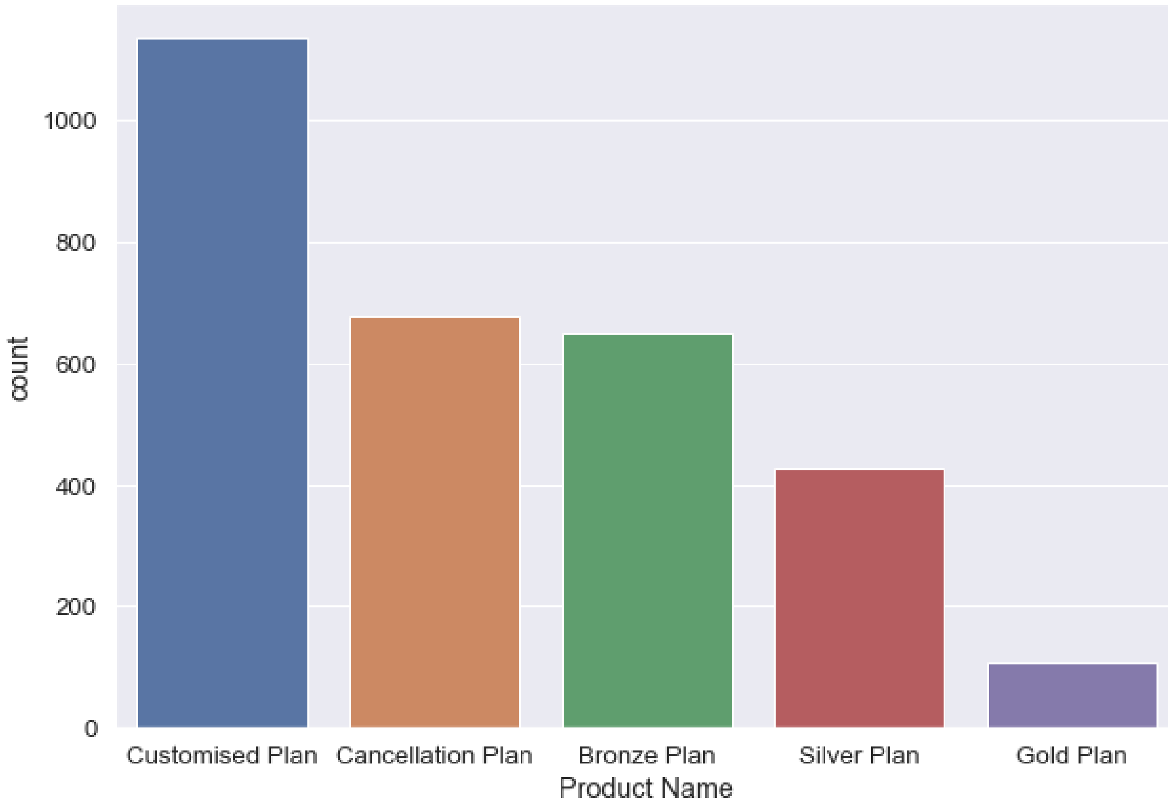
Out[46]: <AxesSubplot:xlabel='Channel', ylabel='count'>



We can see that the majority of the customers used 'Online' Channel for the insurance services.

In [47]:
```
#Countplot for 'Product Name':

size_of_plot = (10, 7)
fig, ax = plt.subplots(figsize=size_of_plot)
sns.countplot(ax=ax, data=ins_df, x = 'Product Name')
```

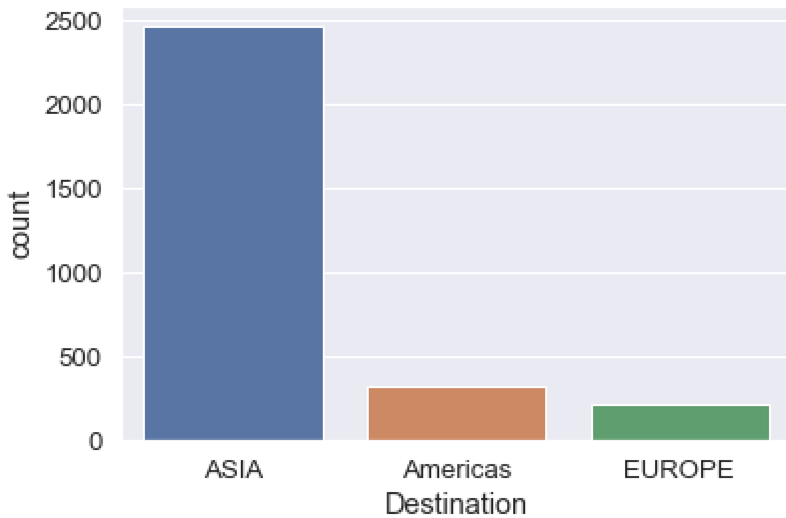
Out[47]: <AxesSubplot:xlabel='Product Name', ylabel='count'>



As per the above shown figure, we can observe that most of the customers want to customize their plan as per their requirement and so the customers who opt for the 'Customised Plan' are more than 1000.

In [48]:
```
#Countplot for 'Destination'

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

Out[48]: <AxesSubplot:xlabel='Destination', ylabel='count'>



Majority of the customers choose 'ASIA' as their preferred Destination.

Performing Crosstab validations just to get the claimed count with respect to few variables

In [49]:
```
#Crosstab validation between 'Type' and 'Claimed'

print('Count of different type of tour insurance firms:\n',ins_df.Type.value_c
ounts())

crosstab_1 = pd.crosstab(ins_df['Type'], ins_df['Claimed'], margins = False)

crosstab_1
```

```
Count of different type of tour insurance firms:
 Travel Agency    1837
Airlines         1163
Name: Type, dtype: int64
```

Out[49]:

Claimed	No	Yes
Type		
Airlines	573	590
Travel Agency	1503	334

In Airlines tour insurance firm, out of total 1163 people, 590 people has claimed and 573 did not claimed their insurance.

In Travel Agency, out of total 1837 people, only 334 people has taken the claim and 1503 did not go for it.

In [50]:
```
#Crosstab validation between 'Channel' and 'Claimed'


print('Count of different Distribution channel of tour insurance agencies:\n',
ins_df.Channel.value_counts())

crosstab_2 = pd.crosstab(ins_df['Channel'], ins_df['Claimed'], margins = False
)

crosstab_2
```

```
Count of different Distribution channel of tour insurance agencies:
 Online     2954
Offline      46
Name: Channel, dtype: int64
```

Out[50]:

Claimed	No	Yes
Channel		
Offline	29	17
Online	2047	907

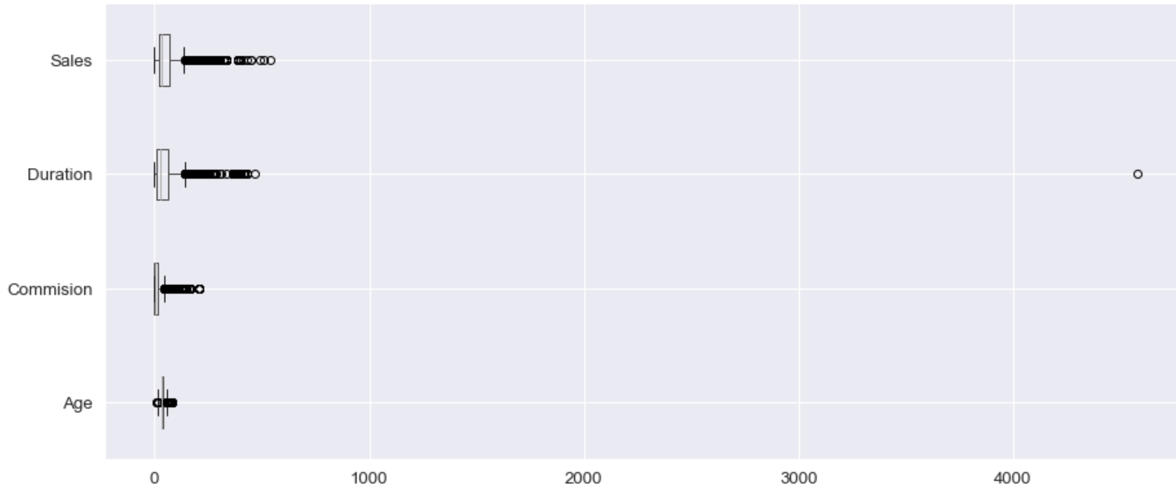
Through online channel, out of total 2954 members, only 907 claimed their insurance and 2047 did not go for it.

Through offline channel, out of total 46, only 17 claimed and 29 did not claimed.

Checking for Outliers for Continuous variables using Boxplot

In [51]:
```
plt.figure(figsize=(15,6.5))

ins_df[['Age','Commision', 'Duration', 'Sales']].boxplot(vert=0)
```

Out[51]: <AxesSubplot:>

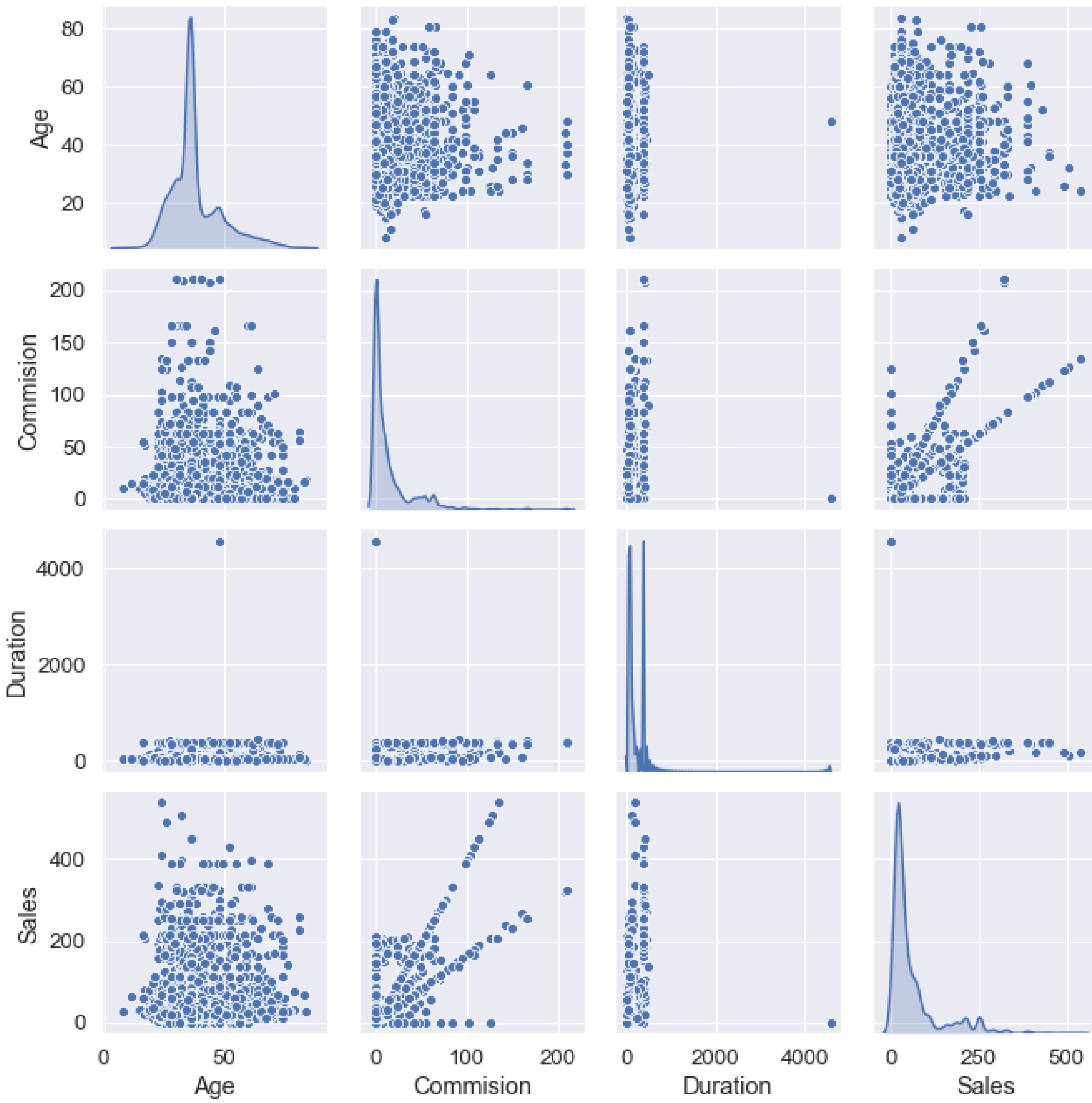


As we can see that there are outliers present in all 4 features of our data.

Displaying Pairplot to see pair wise distribution of continuous variables:

In [52]:
```
sns.pairplot(ins_df[['Age', 'Commision', 'Duration', 'Sales']], height=2.3, di
ag_kind="kde")
```

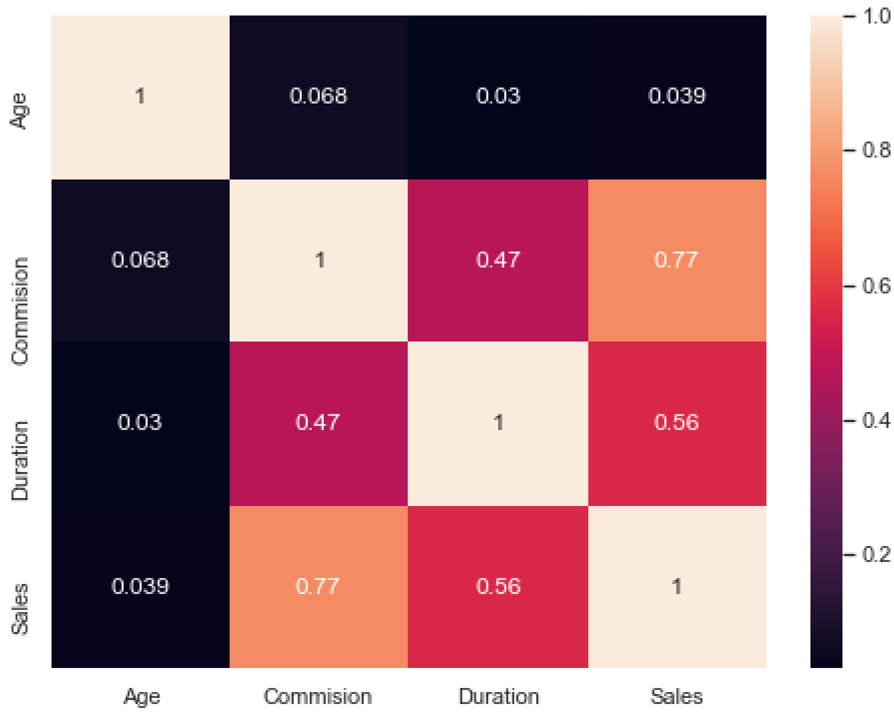
Out[52]: <seaborn.axisgrid.PairGrid at 0x155309221c0>



Displaying Correlations with heatmap for only continuous variables

In [53]:
```
#Displaying Correlations with heatmap for only continuous variables

plt.figure(figsize=(8,6))
sns.set(font_scale=1)
sns.heatmap(ins_df[['Age', 'Commision', 'Duration', 'Sales']].corr(), annot=True)
```

Out[53]: <AxesSubplot:>



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

In [54]:
```
# Decision tree in Python can take only numerical(categorical) columns. It can
not take string(object) datatypes.

# So, we are converting object datatypes into categorical with each distinct v
alue becoming a category or code.

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

```
feature: Agency_Code
Unique Categories:
 [C2B, EPX, CWT, JZI]
Categories (4, object): [C2B, CWT, EPX, JZI]
[0 2 1 3]


feature: Type
Unique Categories:
 [Airlines, Travel Agency]
Categories (2, object): [Airlines, Travel Agency]
[0 1]


feature: Claimed
Unique Categories:
 [No, Yes]
Categories (2, object): [No, Yes]
[0 1]


feature: Channel
Unique Categories:
 [Online, Offline]
Categories (2, object): [Offline, Online]
[1 0]


feature: Product Name
Unique Categories:
 [Customised Plan, Cancellation Plan, Bronze Plan, Silver Plan, Gold Plan]
Categories (5, object): [Bronze Plan, Cancellation Plan, Customised Plan, Gol
d Plan, Silver Plan]
[2 1 0 4 3]


feature: Destination
Unique Categories:
 [ASIA, Americas, EUROPE]
Categories (3, object): [ASIA, Americas, EUROPE]
[0 1 2]
```

In [55]:
```
#After changing object datatypes into categorical, checking the info again

ins_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Age           3000 non-null   int64  
 1   Agency_Code   3000 non-null   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
```

We can see that all the object data types variables are now changed into categorical values.

In [56]:
```
#Checking the head again

ins_df.head()
```

Out[56]:

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

In [57]:
```
#Checking the Proportion of 1s and 0s:

print(ins_df.Type.value_counts(normalize=True))
print('\n--------------------------------------------\n')
print(ins_df.Claimed.value_counts(normalize=True))
print('\n--------------------------------------------\n')
print(ins_df.Channel.value_counts(normalize=True))
print('\n--------------------------------------------\n')
print(ins_df.Agency_Code.value_counts(normalize=True))
print('\n--------------------------------------------\n')
print(ins_df.Destination.value_counts(normalize=True))
```

```
1    0.612333
0    0.387667
Name: Type, dtype: float64

--------------------------------------------

0    0.692
1    0.308
Name: Claimed, dtype: float64

--------------------------------------------

1    0.984667
0    0.015333
Name: Channel, dtype: float64

--------------------------------------------

2    0.455000
0    0.308000
1    0.157333
3    0.079667
Name: Agency_Code, dtype: float64

--------------------------------------------

0    0.821667
1    0.106667
2    0.071667
Name: Destination, dtype: float64
```

In [58]:
```
#Checking the value counts of 'Claimed' individuals

ins_df.Claimed.value_counts()
```

Out[58]:
```
0    2076
1     924
Name: Claimed, dtype: int64
```

Here, 1 indicates the number of customers who claimed the insurance, and 0 indicates the number f customer who did not claimed their insurance.

Splitting the data into training and test set

In [149]:
```
#First step is to separate the dependent variable(target column/variable) and
 independent variable

#Target column is 'Claimed' for this data

X = ins_df.drop('Claimed', axis=1)

y = ins_df['Claimed']
```

In [150]:
```
X.head()
```

Out[150]:

	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

In [151]:
```
from sklearn.model_selection import train_test_split


# We are splitting train set and test set to 70% and 30% of the data respectiv
ely.

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

In [152]:
```
print('X_train has 70% of the data, it has {} rows and {} columns'.format(X_train.shape[0],X_train.shape[1]))

X_train
```

```
X_train has 70% of the data, it has 2100 rows and 9 columns
```

Out[152]:

	Age	Agency_Code	Type	Commision	Channel	Duration	Sales	Product Name	Destination
1045	36	2	1	0.00	1	30	20.00	2	0
2717	36	2	1	0.00	1	139	42.00	2	1
2835	28	0	0	46.96	1	367	187.85	4	0
2913	28	0	0	12.13	1	29	48.50	4	0
959	48	1	1	18.62	1	53	49.00	3	0
...	...	...	...	...	...	...	...	...	...
2763	39	0	0	34.13	1	55	136.50	3	0
905	41	0	0	6.00	1	9	15.00	0	0
1096	36	2	1	0.00	1	131	63.00	2	0
235	44	3	0	6.30	1	6	18.00	0	0
1061	36	0	0	5.63	1	85	22.50	4	0

2100 rows × 9 columns

In [153]:
```
print('X_test has 30% of the data, it has {} rows and {} columns'.format(X_test.shape[0],X_test.shape[1]))

X_test
```

```
X_test has 30% of the data, it has 900 rows and 9 columns
```

Out[153]:

	Age	Agency_Code	Type	Commision	Channel	Duration	Sales	Product Name	Destination
1957	22	1	1	28.50	1	28	75.0	0	2
2087	55	0	0	6.63	1	24	26.5	0	0
1394	29	0	0	4.00	1	33	16.0	0	0
1520	27	0	0	15.88	1	40	63.5	4	0
1098	36	2	1	0.00	1	35	27.0	1	0
...	...	...	...	...	...	...	...	...	...
2363	36	2	1	0.00	1	3	29.0	1	0
270	35	2	1	0.00	1	2	20.0	2	0
517	36	0	0	6.75	1	20	27.0	0	0
2383	49	3	0	10.50	1	57	30.0	0	0
2201	35	3	0	9.10	1	7	26.0	0	0

900 rows × 9 columns

After splitting the data, now we will build a Decision Tree Classifier using 'Gini' criterion and also we will apply grid search for each model and make the models on best_params:

In [154]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV


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

In [155]:
```
#fitting the X_train and train_lables into the dt_model

dt_model.fit(X_train, train_labels)
```

Out[155]:
```
DecisionTreeClassifier()
```

In [156]:
```
from sklearn import tree

train_char_label = ['No','Yes']
```

Creating and opening a dot file 'insurance_tree'

In [157]:
```
#Now we will create a dot file and open it to write(w), so that we can put few
information into this file


file_of_insurance_tree = open('D:\\SHUBHANK !\\GL\\5. TOPIC 4 - Data Mining\\i
nsurance_tree.dot','w')
```

In [158]:
```
#Now writing into the dot file 'insurance_tree.dot' and exporting a graphviz

dot_data = tree.export_graphviz(dt_model,
                                out_file=file_of_insurance_tree,
                                feature_names=list(X_train),
                                class_names=train_char_label)
```

In [159]:
```
#Clsoing the file

file_of_insurance_tree.close()
```

After this we need to open 'webgraphviz.com' in the browser and paste the content from the dot file.

Webgraphviz is actually a Graphviz in a browser. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks.

After checking the tree on the webgraphviz, we can see that the graph is overfitting, so we are defining some parametres using param_grid and using GridSearchCV to loop through predefined hyperparameters,and rebuilding the tree.

Pruning needs to be done to avoid overgrowing of sub-trees/branches

In [160]:
```
#min_samples_split specifies the minimum number of samples required to split a
n internal node,

#min_samples_leaf specifies the minimum number of samples required to be at a
 leaf node.

#GridSearchCV is a function that:
#helps to loop through predefined hyperparameters,
#to fit your estimator (model) on your training set.
#So, in the end, we can select the best parameters from the listed hyperparame
ters.

#CV: (integer value). It Determines the cross-validation splitting strategy.

param_grid = {
    'criterion': ['gini'],
    'max_depth': [5,10,15,20,25,40,50,60,70],
    'min_samples_leaf': [5,8,12,15,30,45,60,],
    'min_samples_split': [20,30,40,50,60,70,80,100,200,300,400]
}

DecTree = DecisionTreeClassifier(random_state=1)

grid_search = GridSearchCV(estimator = DecTree, param_grid = param_grid, cv =
10)

grid_search
```

Out[160]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(random_state=1),
             param_grid={'criterion': ['gini'],
                         'max_depth': [5, 10, 15, 20, 25, 40, 50, 60, 70],
                         'min_samples_leaf': [5, 8, 12, 15, 30, 45, 60],
                         'min_samples_split': [20, 30, 40, 50, 60, 70, 80, 10
0,
                                               200, 300, 400]})
```

In [161]:
```
grid_search.fit(X_train, train_labels)

print(grid_search.best_params_)  #getting the best parameters using grid_searc
h.best_params_
```

```
{'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 5, 'min_samples_spl
it': 60}
```

In [162]:
```
best_grid_dtcl = grid_search.best_estimator_    #getting the best estimator us
ing grid_search.best_estimator_

best_grid_dtcl
```

Out[162]:
```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=5, min_samples_split=60,
                       random_state=1)
```

In [163]:
```
#Creating a dot file named 'insurance_tree_2' and opening it to write

file_of_insurance_tree_2 = open('D:\\SHUBHANK !\\GL\\5. TOPIC 4 - Data Mining
\\insurance_tree_2.dot','w')


#writing or exporting Graphviz into that file

dot_data = tree.export_graphviz(best_grid_dtcl, out_file= file_of_insurance_tr
ee_2 ,
                                feature_names = list(X_train),
                                class_names = list(train_char_label))


#Closing the file

file_of_insurance_tree_2.close()
```

After this, again we need to open 'webgraphviz.com' in the browser and paste the content from the above dot file.

After checking the new tree on the webgraphviz, we can see that the graph is now fitted to the model with the specified parameters(max_depth=5, min_samples_leaf=5, min_samples_split=60) and by using 'gini' criterion.

Getting the 'feature_importances'

In [164]:
```
print (pd.DataFrame(best_grid_dtcl.feature_importances_, columns = ['Imp'],
                    index = X_train.columns).sort_values('Imp',ascending=False
))
```

```
                   Imp
Agency_Code   0.573512
Sales         0.251454
Product Name  0.071827
Duration      0.054815
Commision     0.027938
Channel       0.015191
Age           0.005262
Type          0.000000
Destination   0.000000
```

We can see the 'Agency_Code' holds the most importance of all the features.

Now we have build our final training model(Decision Tree), we can now use it to predict class or value of the target variable.

In [165]:
```
#Our target variable/column is 'Claimed'

ytrain_predict_1 = best_grid_dtcl.predict(X_train)
ytest_predict_1 = best_grid_dtcl.predict(X_test)
```

In [166]:
```
#Probability

ytrain_predict_proba = best_grid_dtcl.predict_proba(X_train)
ytest_predict_proba=best_grid_dtcl.predict_proba(X_test)
```

Building Random Forest Classifier

It is a Supervised Classification Algorithm, as the name suggests, this algorithm creates the forest with a number of trees in random order. Random forest classifier can handle the missing values.

When we have more trees in the forest, random forest classifier won't over fit the model.

In [167]:
```
from sklearn.ensemble import RandomForestClassifier

rfcl = RandomForestClassifier(n_estimators = 501,
                              oob_score = True,
                              max_depth = 10,
                              max_features = 5,
                              min_samples_leaf = 50,
                              min_samples_split = 100,
                              random_state=1)
```

NOTE:

1. n_estimators is the number of trees, that we want to build within the RandomForestClassifier.
2. Out of bag (OOB) score is a way of validating the Random forest model. OOB score is computed as the number of correctly predicted rows from the out of bag sample.
3. max_features will be used to select randomly features from the available number of features.

In [168]:
```
#Fit the X_train and train_labels

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

In [169]:
```
#Checking the OOB Score:

rfcl.oob_score_
```

Out[169]: 0.7847619047619048

In [170]:
```
#Checking the error rate in %:

print('error_rate(in %) :', 1- (rfcl.oob_score_))

#NOTE: OOB Error is the number of wrongly classifying the OOB Sample.
```

```
error_rate(in %) : 0.21523809523809523
```

Error rate is 21.5 %

For the above specified grid parameters, we got the rfcl model with oob score 78.4% and error rate 21.5%, which shows the model is good to go for further evaluation but this time again we will set grid parameters with different values and see how the model works and what is the error rate for the new parameters.

Changing the Grid Parameters using param_grid

In [171]:
```
#Making the param_grid

param_grid_rfcl = {'max_depth':[7,8,9,10],
             'max_features':[4,5,6],
             'min_samples_leaf':[30,50,100],
             'min_samples_split':[50,100,150,300],
             'n_estimators':[301,401,501]}
```

In [172]:
```
rfcl = RandomForestClassifier(random_state=1, oob_score = True)
```

In [173]:
```
grid_search_rfcl = GridSearchCV(estimator = rfcl, param_grid = param_grid_rfcl
, cv=3)
```

In [174]:
```
grid_search_rfcl
```

Out[174]:
```
GridSearchCV(cv=3,
             estimator=RandomForestClassifier(oob_score=True, random_state=
1),
             param_grid={'max_depth': [7, 8, 9, 10], 'max_features': [4, 5,
6],
                         'min_samples_leaf': [30, 50, 100],
                         'min_samples_split': [50, 100, 150, 300],
                         'n_estimators': [301, 401, 501]})
```

In [176]:
```
#Fit the X_train and train_labels into the grid_search

grid_search_rfcl.fit(X_train,train_labels)
```

Out[176]:
```
GridSearchCV(cv=3,
             estimator=RandomForestClassifier(oob_score=True, random_state=
1),
             param_grid={'max_depth': [7, 8, 9, 10], 'max_features': [4, 5,
6],
                         'min_samples_leaf': [30, 50, 100],
                         'min_samples_split': [50, 100, 150, 300],
                         'n_estimators': [301, 401, 501]})
```

In [177]:
```
#Now, Checking the best parameters from the grid_search by using best_params_

grid_search_rfcl.best_params_
```

Out[177]:
```
{'max_depth': 7,
 'max_features': 5,
 'min_samples_leaf': 30,
 'min_samples_split': 50,
 'n_estimators': 401}
```

In [178]:
```
#Now, Checking the best estimator from the grid_search by using best_estimator
_

grid_search_rfcl.best_estimator_
```

Out[178]:
```
RandomForestClassifier(max_depth=7, max_features=5, min_samples_leaf=30,
                       min_samples_split=50, n_estimators=401, oob_score=Tru
e,
                       random_state=1)
```

In [179]:
```
best_grid_rfcl = grid_search_rfcl.best_estimator_
```

In [180]:
```
ytrain_predict = best_grid_rfcl.predict(X_train)
ytest_predict = best_grid_rfcl.predict(X_test)
```

In [181]:
```
ytrain_predict
```

Out[181]:
```
array([0, 0, 1, ..., 0, 0, 1], dtype=int8)
```

In [93]:
```
ytest_predict
```

Out[93]:
```
array([0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
      dtype=int8)
```

In [182]:
```
#Predicting probability

ytest_predict
ytest_predict_prob = best_grid_rfcl.predict_proba(X_test)

ytest_predict_prob
pd.DataFrame(ytest_predict_prob).head()
```

Out[182]:

	0	1
0	0.739077	0.260923
1	0.510523	0.489477
2	0.492637	0.507363
3	0.268578	0.731422
4	0.930667	0.069333

Getting the Variable Importance

In [95]:
```
# Variable Importance

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

```
                   Imp
Agency_Code   0.382244
Product Name  0.212220
Sales         0.169139
Commision     0.096392
Duration      0.059979
Age           0.040173
Type          0.033776
Destination   0.006078
Channel       0.000000
```

Building a Neural Network Classifier

In [184]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
```

In [185]:
```
param_grid_neural = {'hidden_layer_sizes': [50,100,200,250],
                     'max_iter': [2000,2500,3000,4000],
                     'solver': ['sgd'],
                     'tol': [0.01]}

neural_clf = MLPClassifier(random_state=1)

grid_search_neural = GridSearchCV(estimator = neural_clf, param_grid = param_g
rid_neural, cv = 5)
```

In [188]:
```
#Getting the best_params

grid_search_neural.fit(X_train, train_labels)
grid_search_neural.best_params_

#Getting the best_grid

best_grid_neural = grid_search_neural.best_estimator_
best_grid_neural
```

Out[188]:
```
MLPClassifier(hidden_layer_sizes=100, max_iter=2000, random_state=1,
              solver='sgd', tol=0.01)
```

Predicting for the training data and the testing data

In [118]:
```
ytrain_predict_neural = best_grid_neural.predict(X_train)
ytest_predict_neural = best_grid_neural.predict(X_test)
```

Predicting the probability

In [189]:
```
ytest_predict_prob_neural = best_grid_neural.predict_proba(X_test)
ytrain_predict_prob_neural = best_grid_neural.predict_proba(X_train)


ytest_predict_prob_neural
pd.DataFrame(ytest_predict_prob_neural).head()
```

Out[189]:

	0	1
0	0.516012	0.483988
1	0.690558	0.309442
2	0.667939	0.332061
3	0.273701	0.726299
4	0.817098	0.182902

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.

In [123]:
```
#Importing Libraries and packages required to check Performance metrics

from sklearn.metrics import confusion_matrix , classification_report, roc_auc_
score, roc_curve
```

Confusion Matrix for Train and Test Data for Decision Tree Classifier

In [124]:
```
#Confusion Matrix for train data prediction for Decision Tree Classifier

confusion_matrix(train_labels, ytrain_predict_1)
```

Out[124]:
```
array([[1296,  175],
       [ 241,  388]], dtype=int64)
```

In [125]:
```
#Confusion Matrix for test data prediction for Decision Tree Classifier

confusion_matrix(test_labels, ytest_predict_1)
```

Out[125]:
```
array([[544,  61],
       [142, 153]], dtype=int64)
```

Confusion Matrix for Train and Test Data for Random Forest Classifier

In [126]:
```
#Confusion Matrix for train data prediction for Random Forest Classifier

confusion_matrix(train_labels, ytrain_predict)
```

Out[126]:
```
array([[1331,  140],
       [ 270,  359]], dtype=int64)
```

In [127]:
```
#Confusion Matrix for test data prediction for Random Forest Classifier

confusion_matrix(test_labels, ytest_predict)
```

Out[127]:
```
array([[552,  53],
       [156, 139]], dtype=int64)
```

Confusion Matrix for Train and Test Data for Neural Network Classifier

In [129]:
```
#Confusion Matrix for train data prediction for Neural Network Classifier

confusion_matrix(train_labels, ytrain_predict_neural)
```

Out[129]:
```
array([[1350,  121],
       [ 368,  261]], dtype=int64)
```

In [130]:
```
#Confusion Matrix for test data prediction for Neural Neywork Classifier

confusion_matrix(test_labels, ytest_predict_neural)
```

Out[130]:
```
array([[570,  35],
       [194, 101]], dtype=int64)
```

Classification Reports for all 3 Classifiers

In [194]:
```
#Classification Report for train and test data of Decision Tree Classifier

print('Classification Report for train and test data of Decision Tree Classifi
er\n')

print(classification_report(train_labels, ytrain_predict_1))

print(classification_report(test_labels, ytest_predict_1))

print('\n---------------------------------------------------------------------
---')


#Classification Report for train and test data of Random Forest Classifier

print('Classification Report for train and test data of Random Forest Classifi
er\n')

print(classification_report(train_labels, ytrain_predict))

print(classification_report(test_labels, ytest_predict))

print('\n---------------------------------------------------------------------
---')


#Classification Report for train and test data of Neural Network Classifier

print('Classification Report for train and test data of Neural Network Classif
ier\n')

print(classification_report(train_labels, ytrain_predict_neural))

print(classification_report(test_labels, ytest_predict_neural))
```

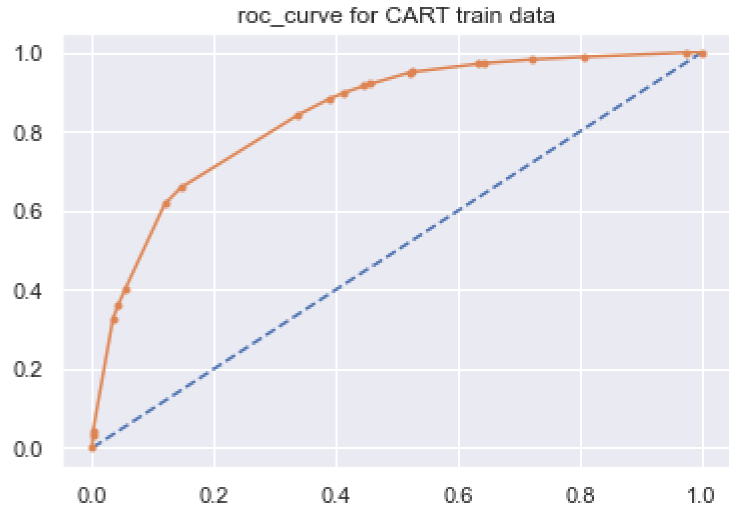
```
Classification Report for train and test data of Decision Tree Classifier

              precision    recall  f1-score   support

           0       0.84      0.88      0.86      1471
           1       0.69      0.62      0.65       629

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

              precision    recall  f1-score   support

           0       0.79      0.90      0.84       605
           1       0.71      0.52      0.60       295

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


-------------------------------------------------------------------------
Classification Report for train and test data of Random Forest Classifier

              precision    recall  f1-score   support

           0       0.83      0.90      0.87      1471
           1       0.72      0.57      0.64       629

    accuracy                           0.80      2100
   macro avg       0.78      0.74      0.75      2100
weighted avg       0.80      0.80      0.80      2100

              precision    recall  f1-score   support

           0       0.78      0.91      0.84       605
           1       0.72      0.47      0.57       295

    accuracy                           0.77       900
   macro avg       0.75      0.69      0.71       900
weighted avg       0.76      0.77      0.75       900


-------------------------------------------------------------------------
Classification Report for train and test data of Neural Network Classifier

              precision    recall  f1-score   support

           0       0.79      0.92      0.85      1471
           1       0.68      0.41      0.52       629

    accuracy                           0.77      2100
   macro avg       0.73      0.67      0.68      2100
weighted avg       0.76      0.77      0.75      2100

              precision    recall  f1-score   support
```

```
              0       0.75      0.94      0.83       605
              1       0.74      0.34      0.47       295

       accuracy                           0.75       900
      macro avg       0.74      0.64      0.65       900
   weighted avg       0.74      0.75      0.71       900
```

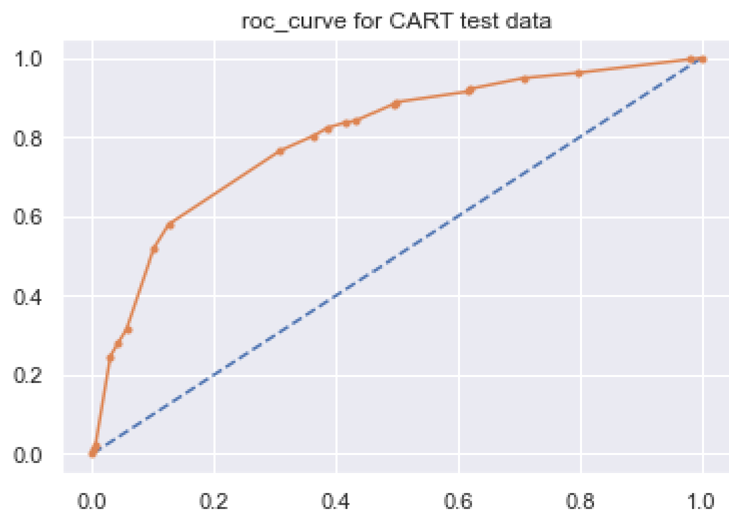
ROC_AUC_SCORE and ROC_CURVE for CART

In [191]:
```
ytrain_predict_proba = best_grid_dtcl.predict_proba(X_train)

ytrain_predict_proba = ytrain_predict_proba[:,1]

auc_dtcl_train = roc_auc_score(train_labels, ytrain_predict_proba)

print('roc_auc_score for train data for CART: ',auc_dtcl_train)

fpr_dtcl_train,tpr_dtcl_train,thresholds_dtcl_train = roc_curve(train_labels,
ytrain_predict_proba)

plt.plot([0,1],[0,1],linestyle='--')

plt.plot(fpr_dtcl_train,tpr_dtcl_train,marker='.')
plt.title('roc_curve for CART train data')

plt.show()

print('-------------------------------------------')

ytest_predict_proba = best_grid_dtcl.predict_proba(X_test)

ytest_predict_proba = ytest_predict_proba[:,1]

auc_dtcl_test = roc_auc_score(test_labels, ytest_predict_proba)

print('roc_auc_score for test data for CART: ',auc_dtcl_test)


fpr_dtcl_test,tpr_dtcl_test,thresholds_dtcl_test = roc_curve(test_labels, ytes
t_predict_proba)

plt.plot([0,1],[0,1],linestyle='--')

plt.plot(fpr_dtcl_test,tpr_dtcl_test,marker='.')
plt.title('roc_curve for CART test data')

plt.show()
```

```
roc_auc_score for train data for CART:  0.8434233009351976
```



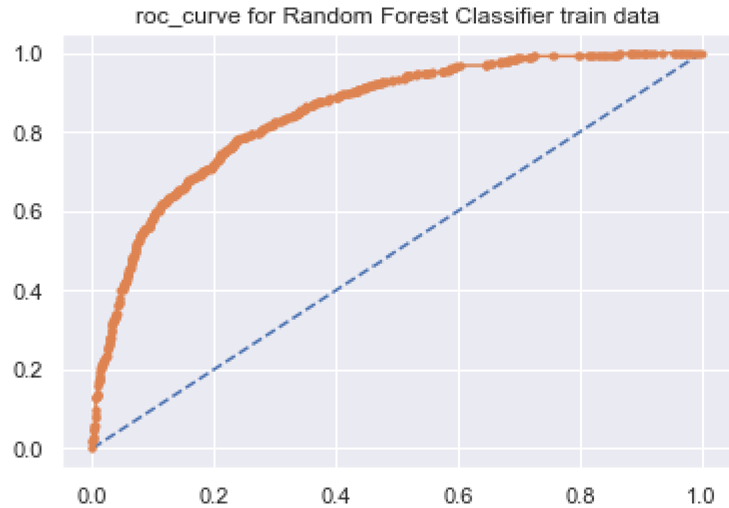

```
------------------------------------------
roc_auc_score for test data for CART:  0.7995293458467572
```



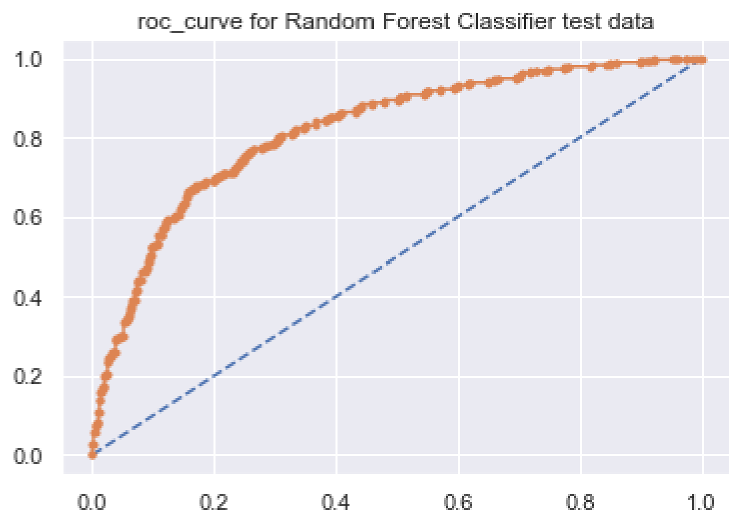
ROC_AUC_SCORE and ROC_CURVE for RANDOM FOREST Classifier

In [192]:
```
ytrain_predict_prob = best_grid.predict_proba(X_train)

ytrain_predict_prob = ytrain_predict_prob[:,1]

auc_rfcl_train = roc_auc_score(train_labels, ytrain_predict_proba)

print('roc_auc_score for train data for Random Forest Classifier: ',auc_rfcl_t
rain)

fpr_rfcl_train,tpr_rfcl_train,thresholds_rfcl_train = roc_curve(train_labels,
ytrain_predict_prob)

plt.plot([0,1],[0,1],linestyle='--')

plt.plot(fpr_rfcl_train,tpr_rfcl_train,marker='.')
plt.title('roc_curve for Random Forest Classifier train data')

plt.show()

print('-------------------------------------------')

ytest_predict_prob = best_grid.predict_proba(X_test)

ytest_predict_prob = ytest_predict_prob[:,1]

auc_rfcl_test = roc_auc_score(test_labels, ytest_predict_prob)

print('roc_auc_score for test data for Random Forest Classifier: ',auc_rfcl_te
st)


fpr_rfcl_test,tpr_rfcl_test,thresholds_rfcl_test = roc_curve(test_labels, ytes
t_predict_prob)

plt.plot([0,1],[0,1],linestyle='--')

plt.plot(fpr_rfcl_test,tpr_rfcl_test,marker='.')
plt.title('roc_curve for Random Forest Classifier test data')

plt.show()
```

```
roc_auc_score for train data for Random Forest Classifier:  0.843423300935197
6
```



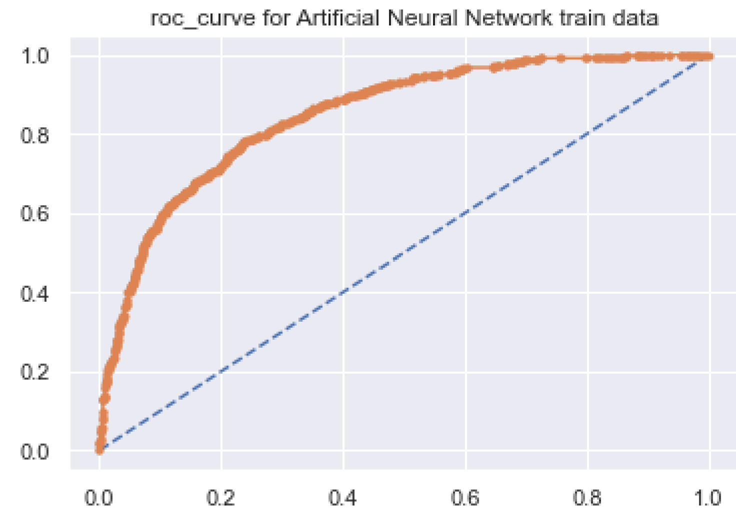
```
-------------------------------------------
roc_auc_score for test data for Random Forest Classifier:  0.8200112060512676
```



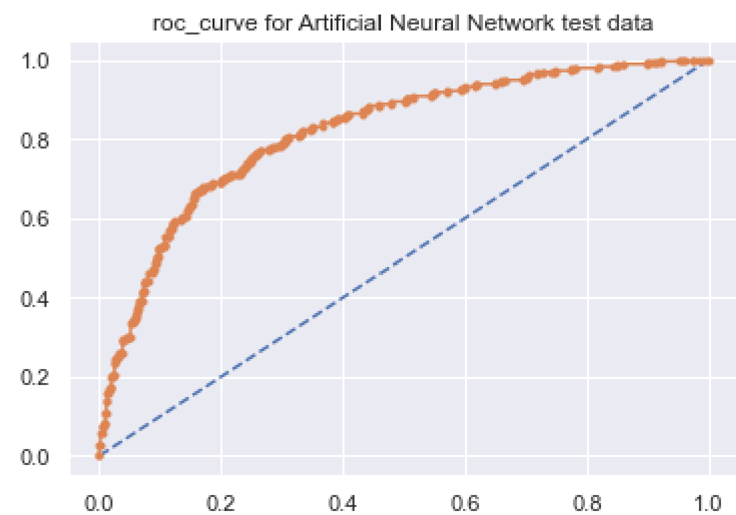
ROC_AUC_SCORE and ROC_CURVE for Artificial Neural Network

In [193]:
```
ytrain_predict_prob_neural = best_grid.predict_proba(X_train)

ytrain_predict_prob_neural = ytrain_predict_prob_neural[:,1]

auc_neural_train = roc_auc_score(train_labels, ytrain_predict_prob_neural)

print('roc_auc_score for train data for Artificial Neural Network: ',auc_neura
l_train)

fpr_neural_train,tpr_neural_train,thresholds_neural_train = roc_curve(train_la
bels, ytrain_predict_prob_neural)

plt.plot([0,1],[0,1],linestyle='--')

plt.plot(fpr_neural_train,tpr_neural_train,marker='.')
plt.title('roc_curve for Artificial Neural Network train data')

plt.show()

print('-------------------------------------------')

ytest_predict_prob_neural = best_grid.predict_proba(X_test)

ytest_predict_prob_neural = ytest_predict_prob_neural[:,1]

auc_neural_test = roc_auc_score(test_labels, ytest_predict_prob_neural)

print('roc_auc_score for test data for Artificial Neural Network: ',auc_neural
_test)


fpr_neural_test,tpr_neural_test,thresholds_neural_test = roc_curve(test_labels
, ytest_predict_prob_neural)

plt.plot([0,1],[0,1],linestyle='--')

plt.plot(fpr_neural_test,tpr_neural_test,marker='.')
plt.title('roc_curve for Artificial Neural Network test data')

plt.show()
```

```
roc_auc_score for train data for Artificial Neural Network:  0.85022247824663
14
```




```
-------------------------------------------
roc_auc_score for test data for Artificial Neural Network:  0.820011206051267
6
```



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

Comparison of AUC, Accuracy, Precision, Recall and F1 Score between all 3 models:

As we can see from the above calculations and classification report, below are some values that we got:

In [195]:
```
#AUC, Accuracy, Precision, Recall and F1 Score for CART

#train data
cart_train_acc=0.80
cart_train_recall=0.62
cart_train_precision=0.69
cart_train_f1=0.65

#test data
cart_test_acc=0.77
cart_test_recall=0.52
cart_test_precision=0.71
cart_test_f1=0.60

#AUC, Accuracy, Precision, Recall and F1 Score for RFCL


#train data
rf_train_acc=0.80
rf_train_recall=0.57
rf_train_precision=0.72
rf_train_f1=0.64

#test data
rf_test_acc=0.77
rf_test_recall=0.47
rf_test_precision=0.72
rf_test_f1=0.57

#AUC, Accuracy, Precision, Recall and F1 Score for ANN

#train data
neural_train_acc=0.77
neural_train_recall=0.41
neural_train_precision=0.68
neural_train_f1=0.52

#test data
neural_test_acc=0.75
neural_test_recall=0.34
neural_test_precision=0.74
neural_test_f1=0.47
```

Table for Comparison of all values:

In [197]:
```
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']

data = pd.DataFrame({'CART_train':[cart_train_acc,auc_dtcl_train,cart_train_re
call,cart_train_precision,cart_train_f1],
        'CART_test':[cart_test_acc,auc_dtcl_test,cart_test_recall,cart_test_pr
ecision,cart_test_f1],
       'Random_Forest_train':[rf_train_acc,auc_rfcl_train,rf_train_recall,rf_t
rain_precision,rf_train_f1],
        'Random_Forest_test':[rf_test_acc,auc_rfcl_test,rf_test_recall,rf_test
_precision,rf_test_f1],
       'Neural_Network_train':[neural_train_acc,auc_neural_train,neural_train_
recall,neural_train_precision,neural_train_f1],
        'Neural_Network_test':[neural_test_acc,auc_neural_test,neural_test_rec
all,neural_test_precision,neural_test_f1]},index=index)
round(data,2)
```

Out[197]:

	CART_train	CART_test	Random_Forest_train	Random_Forest_test	Neural_Network_tr
Accuracy	0.80	0.77	0.80	0.77	0
AUC	0.84	0.80	0.84	0.82	0
Recall	0.62	0.52	0.57	0.47	0
Precision	0.69	0.71	0.72	0.72	0
F1 Score	0.65	0.60	0.64	0.57	0

Final Model:

I prefer to go with Random Forest Classifier Model because, based on above performance metrics, we can see that the accuracy , f1 score and AUC score is better than the CART and ANN based models. Due to high accuracy and better AUC score, we can select RF model confidently.

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

Recommendations:

1. As we can see that majority of the insurance is done through 'Online' channel because it is easy to get the benefits online instead of choosing Offline channel. Also, it is easy to process claim online, So, due to these reasons, there is increase in the claim process.
2. I have also observed that JZI agency has lesser number of sales, they need to boost up their sales by applying some kind of promotional strategies to attract the customers. They can also take help and support from other successful agencies or resources.
3. JZI agency need to add more number of working and active resources to increase their sales.
4. Agencies have more number of sales than Airlines, but number of insured is more by Airlines than agencies.
5. One thing we have observed in the dataset is, people opted for customized plan more than any other plans offered by agencies and airlines. By looking at this situation, I recommend that agencies should gather more information from the customers who took customized plan and ask them what they liked most and in which of the situations they felt to take the insurance plan.
6. If customers buying insurance offline, we must make sure that the process is simple and smooth so that in future, the customers who claimed their insurance suggest others to take insurance from us. In this way we are not only making our active customers happy but also doubling our sales.
7. We can add-up few more extra risk covers which does happen generally with all kind of people, like loss of personal objects, damage due to the services provided while travelling.
8. We can opt for traditional ways to boost up the sales and also for making the process safe and secure.

-----------------------------------"END"--