Problem 1: Clustering

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

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import dendrogram,linkage,fcluster
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

In [2]:

```
bank_df=pd.read_csv("C:\\Users\\Shubham\\Downloads\\bank_marketing_part1_Data.csv")
bank_df.head()
```

Out[2]:

	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	

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

In [3]:

```
bank_df.describe()
```

Out[3]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit
count	210.000000	210.000000	210.000000	210.000000	210.000000
mean	14.847524	14.559286	0.870999	5.628533	3.258605
std	2.909699	1.305959	0.023629	0.443063	0.377714
min	10.590000	12.410000	0.808100	4.899000	2.630000
25%	12.270000	13.450000	0.856900	5.262250	2.944000
50%	14.355000	14.320000	0.873450	5.523500	3.237000
75%	17.305000	15.715000	0.887775	5.979750	3.561750
max	21.180000	17.250000	0.918300	6.675000	4.033000

In [4]:

```
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
```

The dataset consist of 7 different attributes of credit card data.There are 210 entries,All data_types are float type and no null value present in the dataset.

In [5]:

```
bank_df.shape
```

Out[5]:

```
(210, 7)
```

In [6]:

```
bank_df.duplicated().sum()
```

Out[6]:

```
0
```

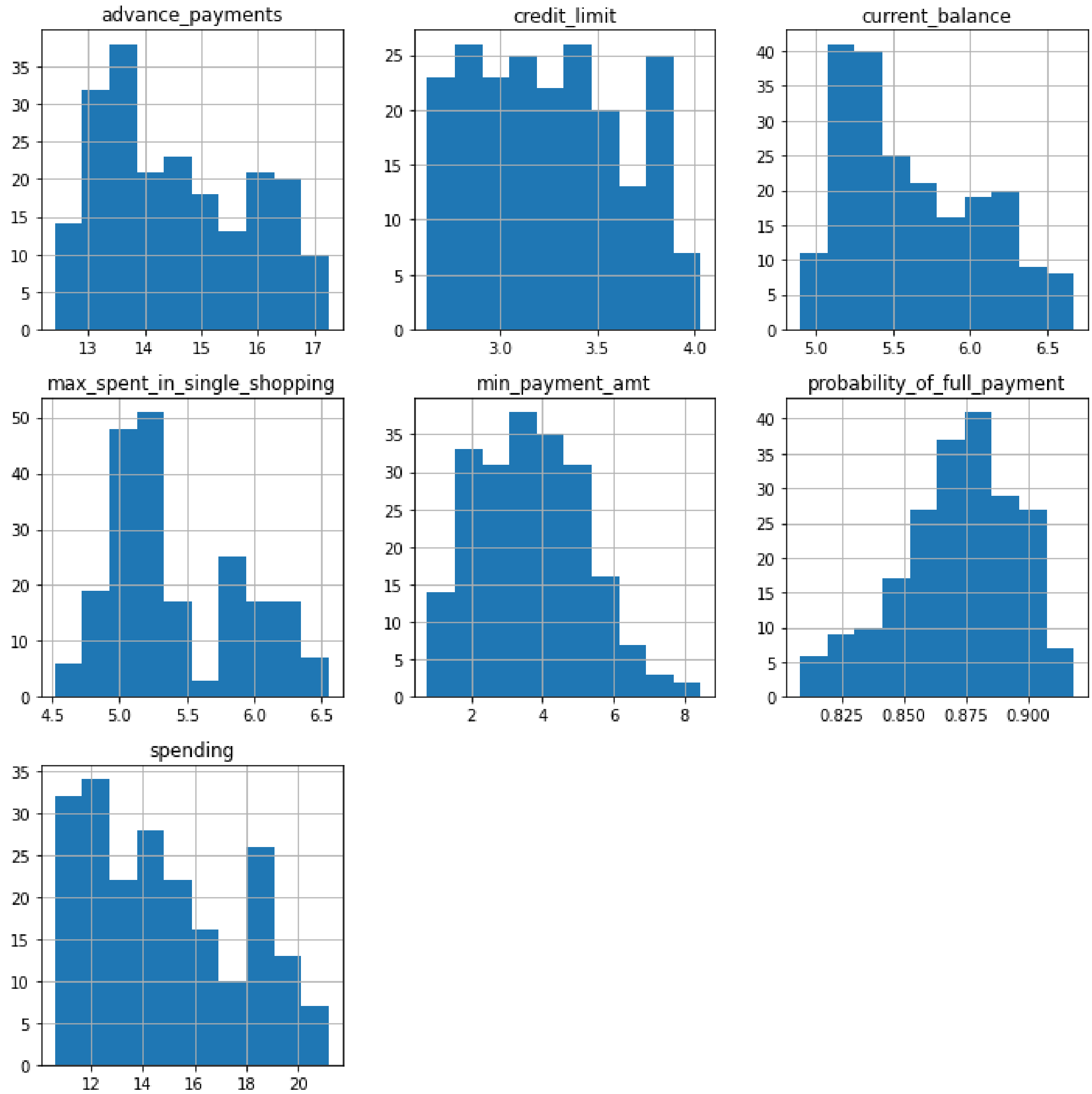
Do you think scaling is necessary for clustering in this case? Justify

Checking distribution using histogram:

In [7]:

```
df=bank_df.copy()
fig=plt.figure(figsize=(10,10))
ax=fig.gca()
df.hist(ax=ax)
plt.tight_layout()
plt.show()
```

```
<ipython-input-7-62fd5969bf53>:4: UserWarning: To output multiple subplots,
the figure containing the passed axes is being cleared
  df.hist(ax=ax)
```

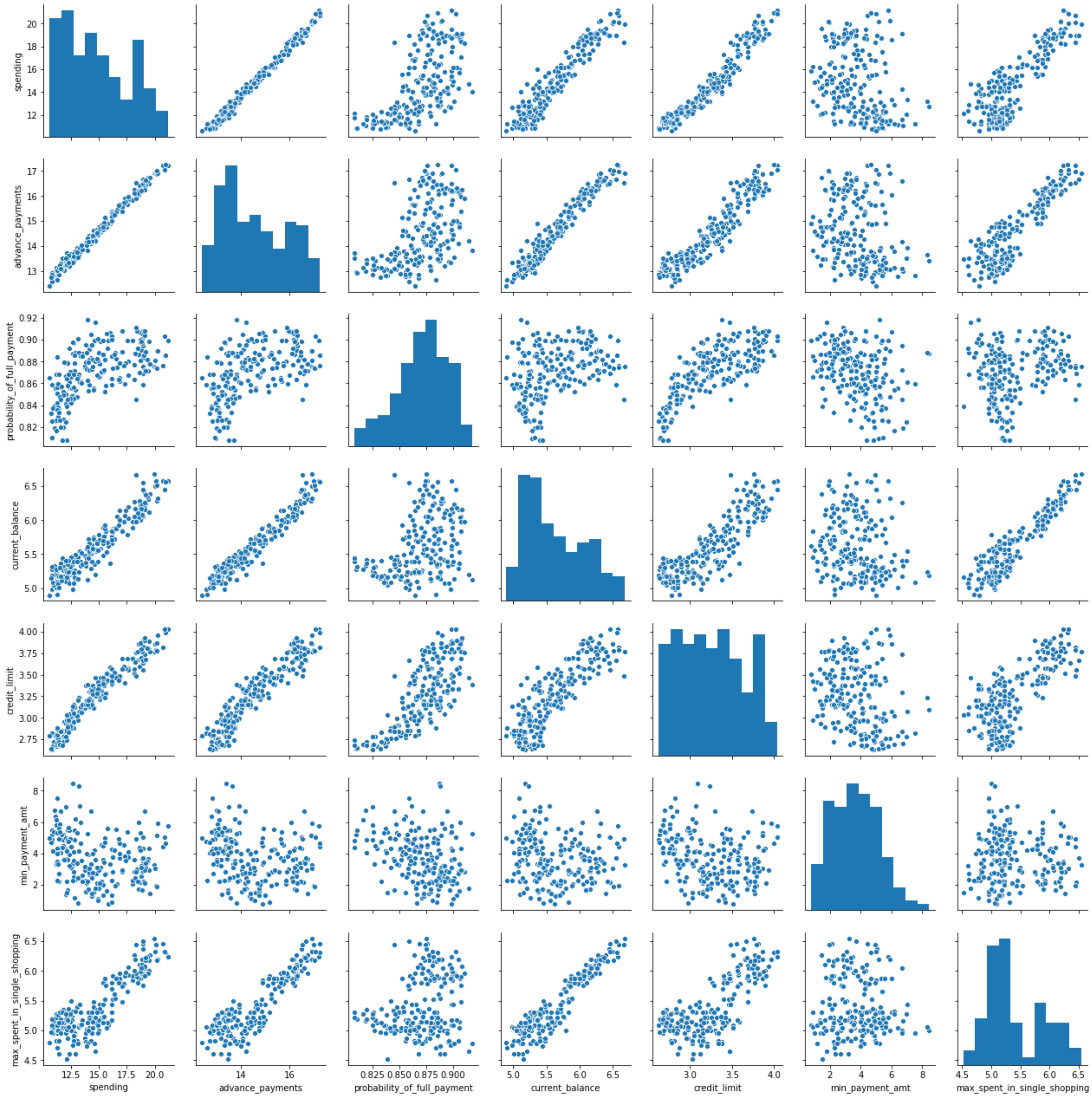


In [8]:

```
sns.pairplot(data=bank_df)
```

Out[8]:

```
<seaborn.axisgrid.PairGrid at 0x24e05647dc0>
```



clearly from the above Barplot & pairplot we can see that all the attributes are not scaled and pre scaling will be required before performing clustring.

Scaling of data:

In [9]:

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

In [10]:

```
scaler = StandardScaler()
scaled=scaler.fit_transform(bank_df)
scaled
```

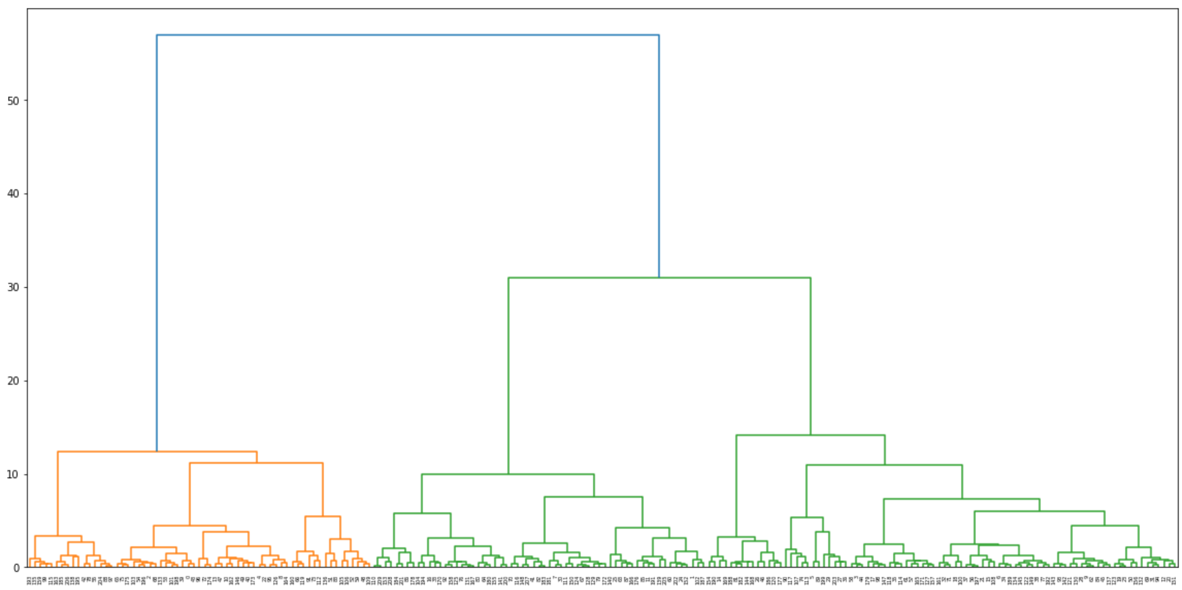
Out[10]:

```
array([[ 1.75435461,  1.81196782,  0.17822987, ...,  1.33857863,
        -0.29880602,  2.3289982 ],
       [ 0.39358228,  0.25383997,  1.501773  , ...,  0.85823561,
        -0.24280501, -0.53858174],
       [ 1.41330028,  1.42819249,  0.50487353, ...,  1.317348  ,
        -0.22147129,  1.50910692],
       ...,
       [-0.2816364 , -0.30647202,  0.36488339, ..., -0.15287318,
        -1.3221578 , -0.83023461],
       [ 0.43836719,  0.33827054,  1.23027698, ...,  0.60081421,
        -0.95348449,  0.07123789],
       [ 0.24889256,  0.45340314, -0.77624835, ..., -0.07325831,
        -0.70681338,  0.96047321]])
```

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

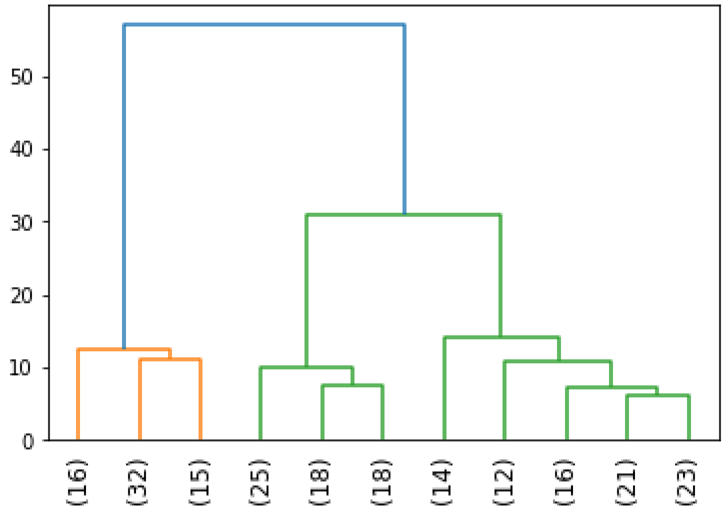
In [11]:

```
plt.figure(figsize=(20,10))
wardlink = linkage(bank_df, method = 'ward')
dend = dendrogram(wardlink)
```



In [12]:

```
dend = dendrogram(wardlink,
                  truncate_mode='lastp',
                  p = 11,
                  leaf_rotation=90,
                  leaf_font_size=12)
```



In [13]:

```
from scipy.cluster.hierarchy import fcluster
```

In [132]:

```
#Method 1
clusters=fcluster(wardlink,2,criterion="maxclust")
clusters
```

Out[132]:

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

In [133]:

```
bank_df["clusters"]=clusters
bank_df.groupby("clusters").mean()
bank_df.head(10)
```

Out[133]:

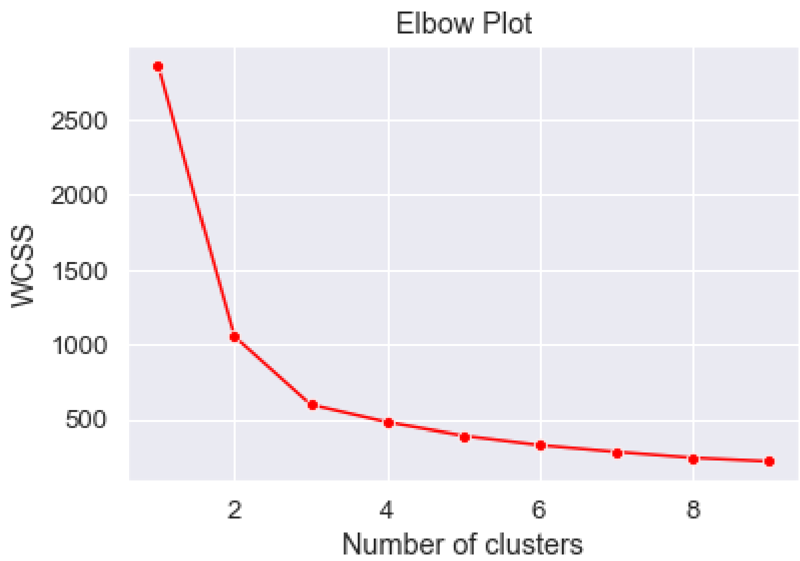
	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_
0	19.94	16.92	0.8752	6.675	3.763	
1	15.99	14.89	0.9064	5.363	3.582	
2	18.95	16.42	0.8829	6.248	3.755	
3	10.83	12.96	0.8099	5.278	2.641	
4	17.99	15.86	0.8992	5.890	3.694	
5	12.70	13.41	0.8874	5.183	3.091	
6	12.02	13.33	0.8503	5.350	2.810	
7	13.74	14.05	0.8744	5.482	3.114	
8	18.17	16.26	0.8637	6.271	3.512	
9	11.23	12.88	0.8511	5.140	2.795	

There are 2 optimum no of clusters.Cluster 1 consist of higher values of "Spending","Max_spend_in_single_shopping","advance_payments","credit_limit","current balance." Clusters 2 consist of lower values of these attributes.

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

In [128]:

```
wcss = []
for i in range(1,10):
 kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state =10)
 kmeans.fit(bank_df)
 # inertia method returns wcss for that model
 wcss.append(kmeans.inertia_)
sns.lineplot(range(1,10), wcss,marker='o',color='red')
plt.title('Elbow Plot')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



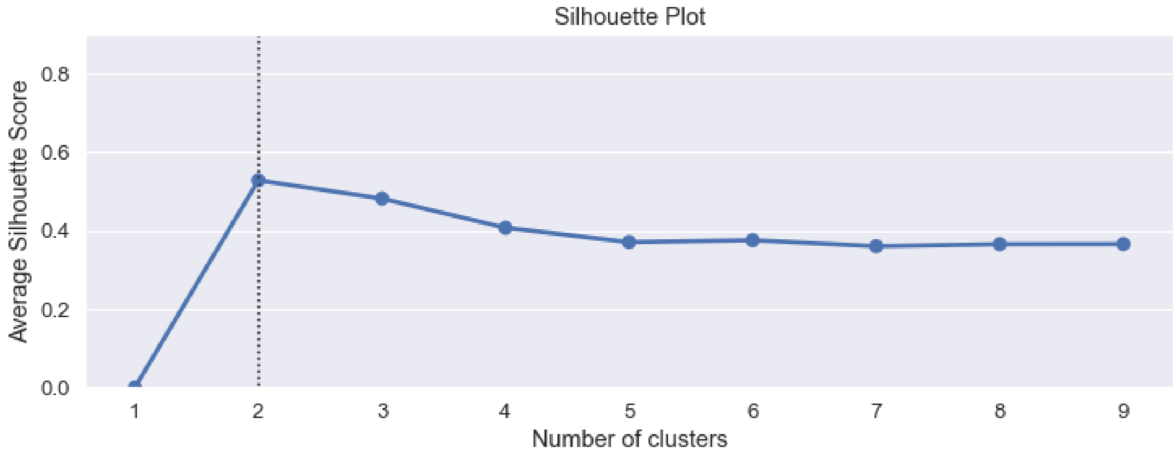
In [129]:

```
ss={1:0}
for i in range(2,10):
 clusterer = KMeans(n_clusters = i, init = 'k-means++', random_state =44)
 y=clusterer.fit_predict(bank_df)
 # The higher (up to 1) the better
 s =silhouette_score(bank_df, y )
 ss[i]=round(s,5)
 print("The Average Silhouette Score for {} clusters is {}".format(i,round(s,5)))
```

```
The Average Silhouette Score for 2 clusters is 0.5279
The Average Silhouette Score for 3 clusters is 0.4814
The Average Silhouette Score for 4 clusters is 0.40699
The Average Silhouette Score for 5 clusters is 0.37058
The Average Silhouette Score for 6 clusters is 0.3755
The Average Silhouette Score for 7 clusters is 0.3602
The Average Silhouette Score for 8 clusters is 0.36539
The Average Silhouette Score for 9 clusters is 0.3657
```

In [130]:

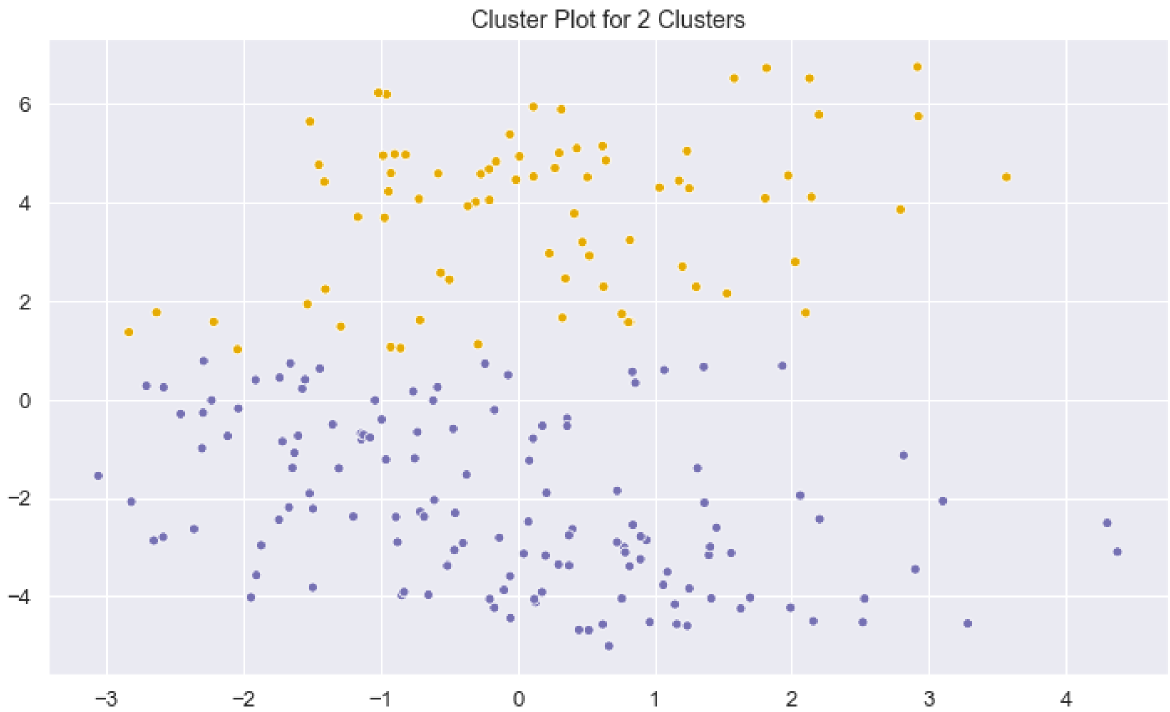
```
maxkey= [key for key, value in ss.items() if value == max(ss.values())][0]
fig,ax = plt.subplots(figsize=(12,4))
sns.pointplot(list(ss.keys()),list(ss.values()))
plt.vlines(x=maxkey-1,ymax=0,ymin=0.90,linestyles='dotted')
ax.set(ylim=(0, 0.90))
ax.set_title('Silhouette Plot')
ax.set_xlabel('Number of clusters')
ax.set_ylabel('Average Silhouette Score')
plt.show()
```



It is clear from above figure that the maximum value of average silhouette score is achieved for k =2, which, therefore, is considered to be the optimum number of clusters for this data.But statistically 2 clusters are not good for the analysis and does'nt full fill the need for clustering.Hence selecting 2 close optimum value of k other than 2.

In [134]:

```
from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(bank_df)
plt.figure(figsize=(12,7))
sns.scatterplot(x=plot_columns[:,1], y=plot_columns[:,0], hue=KMeans(n_clusters=
2, random_state=0).fit(bank_df).labels_, palette='Dark2_r',legend=False)
plt.title('Cluster Plot for 2 Clusters')
plt.show()
```



Problem 2: CART-RF-ANN

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

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

In [21]:

```
insurance_df=pd.read_csv("C:\\Users\\Shubham\\Downloads\\insurance_part2_data.csv")
insurance_df.head()
```

Out[21]:

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	D
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	

In [22]:

```
insurance_df.shape
```

Out[22]:

```
(3000, 10)
```

In [23]:

```
insurance_df.describe(include="all")
```

Out[23]:

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	
count	3000.000000	3000	3000	3000	3000.000000	3000	3000.000000	3000
unique	NaN	4	2	2	NaN	2	NaN	
top	NaN	EPX	Travel Agency	No	NaN	Online	NaN	
freq	NaN	1365	1837	2076	NaN	2954	NaN	
mean	38.091000	NaN	NaN	NaN	14.529203	NaN	70.001333	60
std	10.463518	NaN	NaN	NaN	25.481455	NaN	134.053313	70
min	8.000000	NaN	NaN	NaN	0.000000	NaN	-1.000000	0
25%	32.000000	NaN	NaN	NaN	0.000000	NaN	11.000000	20
50%	36.000000	NaN	NaN	NaN	4.630000	NaN	26.500000	33
75%	42.000000	NaN	NaN	NaN	17.235000	NaN	63.000000	69
max	84.000000	NaN	NaN	NaN	210.210000	NaN	4580.000000	539

Check for null values in columns

In [24]:

```
insurance_df.isnull().sum()
```

Out[24]:

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

There is no null value present

In [25]:

```
insurance_df.info()
```

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

Claimed is the target variable while all others are the predictors

Out of 9 datatypes 2 are integer type, 2 are float and 6 are object types

It seems there is no null values present in dataset

Getting unique counts of all columns

In [26]:

```
mn in insurance_df[["Age","Agency_Code","Type","Claimed","Commision","Channel","Duration","S
t(column.upper(),': ',insurance_df[column].nunique())
t(insurance_df[column].value_counts().sort_values())
t('\n')
```

```
AGE :  70
8       1
14      1
83      1
77      1
84      1
     ...
35     94
30     96
48    108
31    125
36    999
Name: Age, Length: 70, dtype: int64


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


TYPE :  2
Airlines          1163
Travel Agency     1837
Name: Type, dtype: int64


CLAIMED :  2
Yes     924
No     2076
Name: Claimed, dtype: int64


COMMISION :  324
126.75      1
12.45       1
46.80       1
21.35       1
17.55       1
          ...
7.70       57
23.76      61
54.00      61
63.21      62
0.00     1366
Name: Commision, Length: 324, dtype: int64


CHANNEL :  2
Offline      46
```

```
Online     2954
Name: Channel, dtype: int64


DURATION :  257
4580     1
149      1
141      1
215      1
217      1
        ..
11      81
10      81
6       81
5       82
8       83
Name: Duration, Length: 257, dtype: int64


SALES :  380
271.00      1
62.40       1
491.50      1
159.00      1
100.50      1
         ...
216.00     59
252.85     60
22.00      79
10.00     163
20.00     225
Name: Sales, Length: 380, dtype: int64


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


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

Check for duplicate data

In [27]:

```
# Are there any duplicates ?
dups = insurance_df.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
df[dups]
```

```
Number of duplicate rows = 139

<ipython-input-27-93b280984e57>:4: UserWarning: Boolean Series key will be r
eindexed to match DataFrame index.
  df[dups]
```

Out[27]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min
63	15.26	14.85	0.8696	5.714	3.242	

Removing Duplicates

In [28]:

```
insurance_df.drop_duplicates(inplace=True)
dups = insurance_df.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
print(insurance_df.shape)
```

```
Number of duplicate rows = 0
(2861, 10)
```

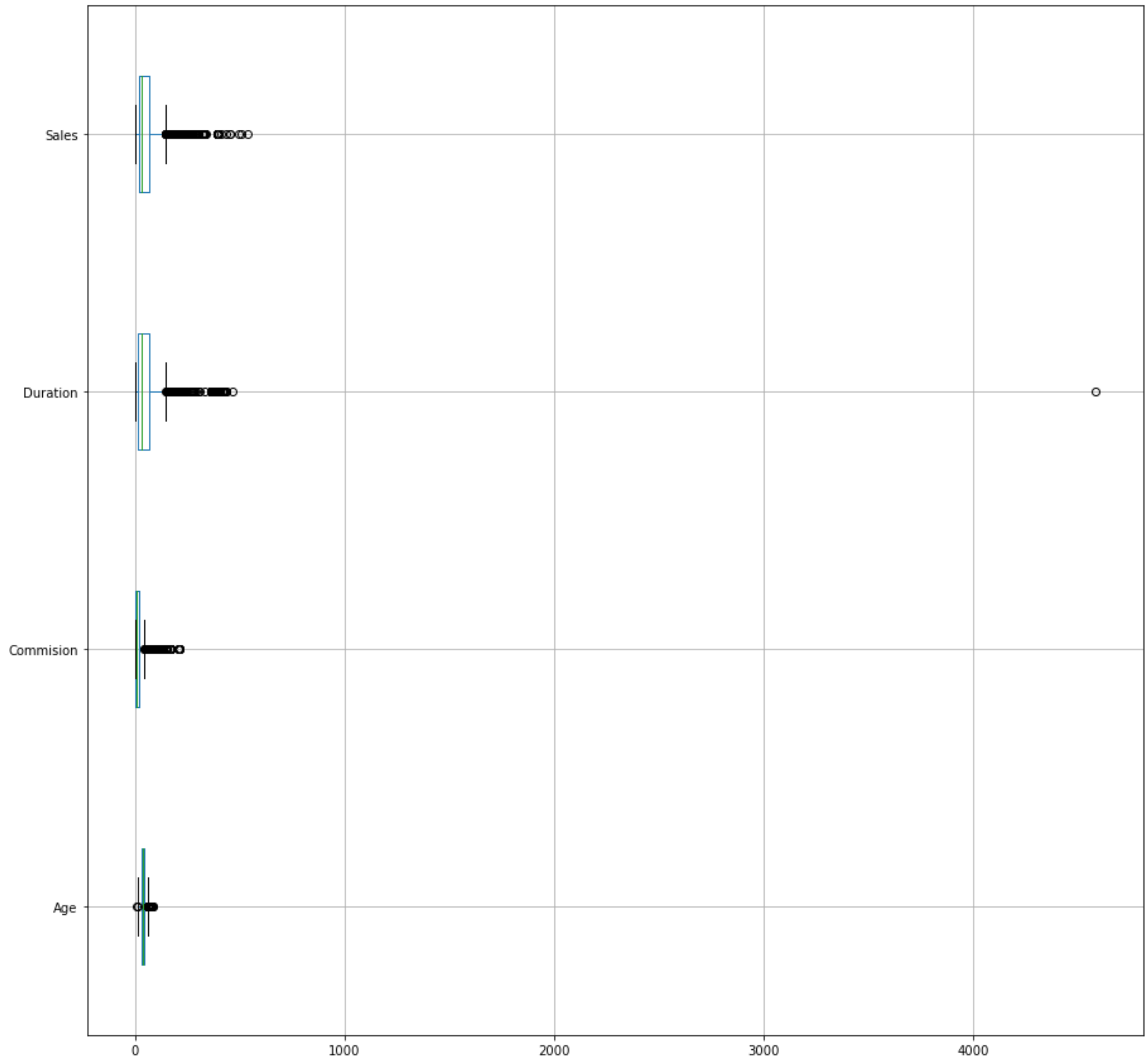
Checking for outliers

In [29]:

```
plt.figure(figsize=(15,15))
insurance_df[["Age","Commision","Duration","Sales"]].boxplot(vert=0)
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24e0b2a2fa0>
```



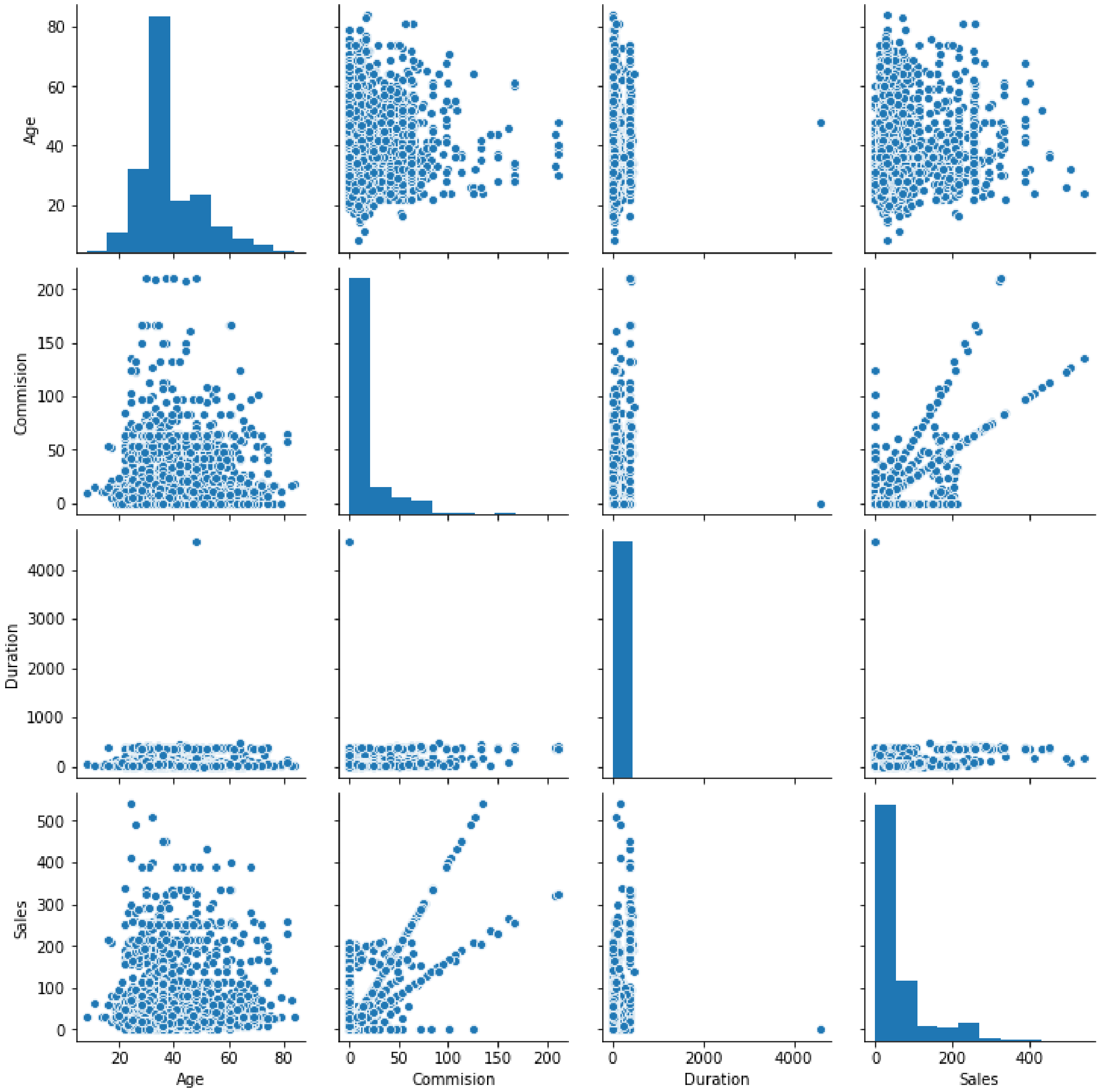
Checking pairwise distribution of the continuous variables

In [30]:

```
sns.pairplot(insurance_df[["Age","Agency_Code","Type","Claimed","Commision","Channel","Dura
```

Out[30]:

<seaborn.axisgrid.PairGrid at 0x24e0b324eb0>



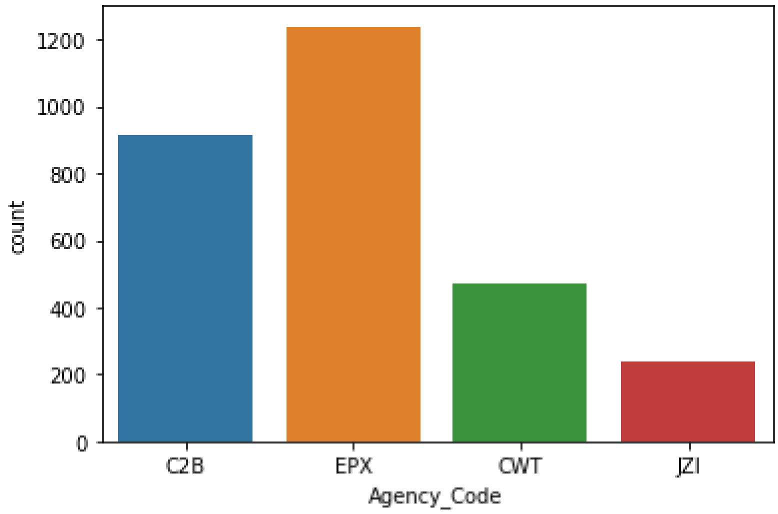
Categorical Variables:

In [31]:

```
sns.countplot(data =insurance_df, x = 'Agency_Code')
```

Out[31]:

<matplotlib.axes._subplots.AxesSubplot at 0x24e0bf11340>

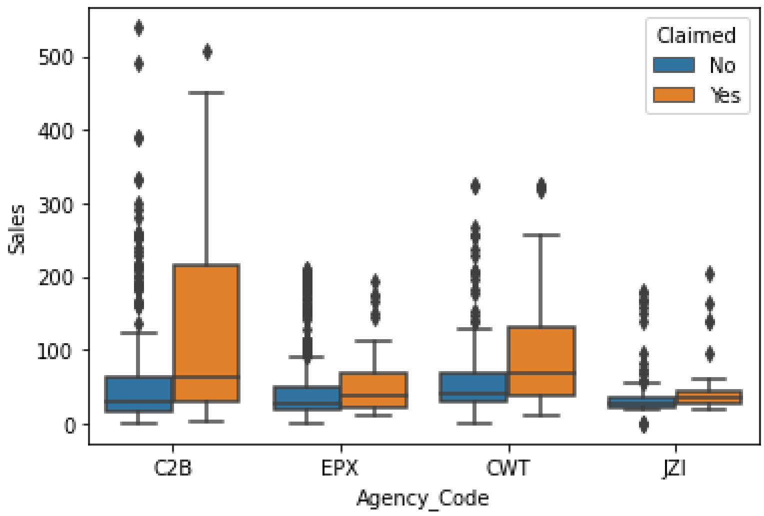


In [32]:

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

Out[32]:

<matplotlib.axes._subplots.AxesSubplot at 0x24e0befb550>

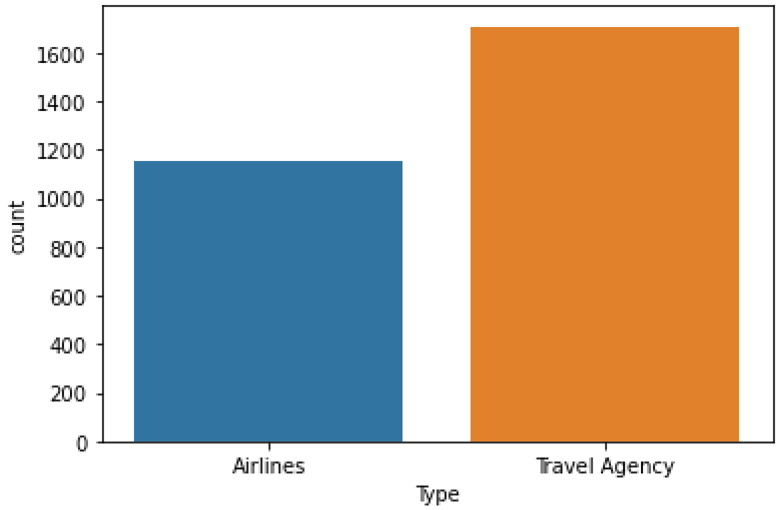


In [33]:

```
#Type:
sns.countplot(data =insurance_df, x= 'Type')
```

Out[33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24e0d228ee0>
```

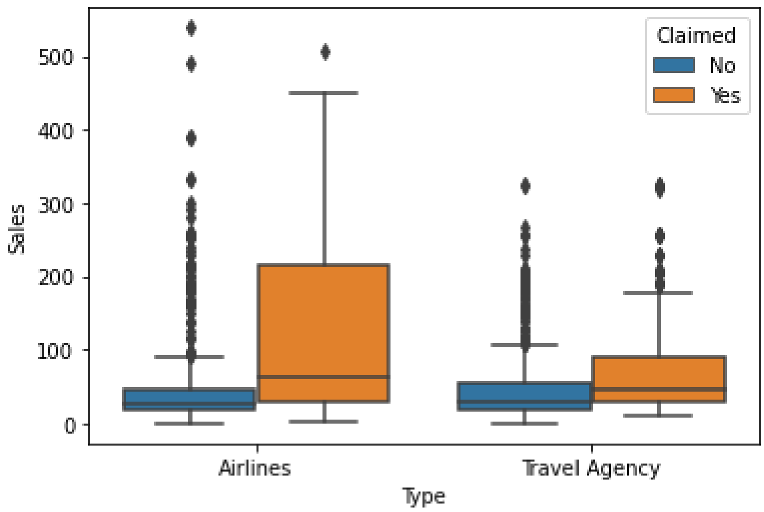


In [34]:

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

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24e0bed70d0>
```

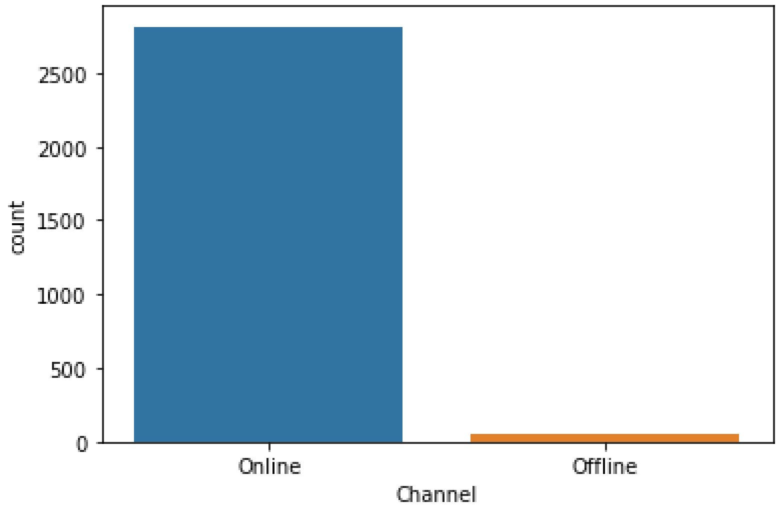


In [35]:

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

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24e0a07d700>
```

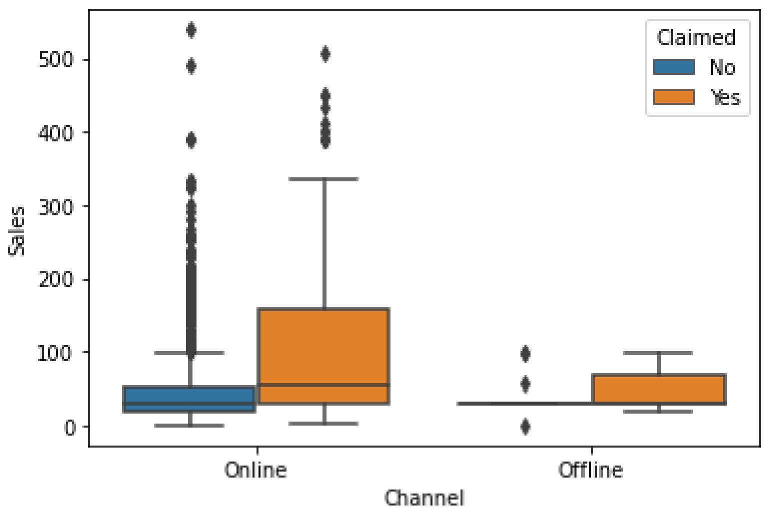


In [36]:

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

Out[36]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24e0b239100>
```

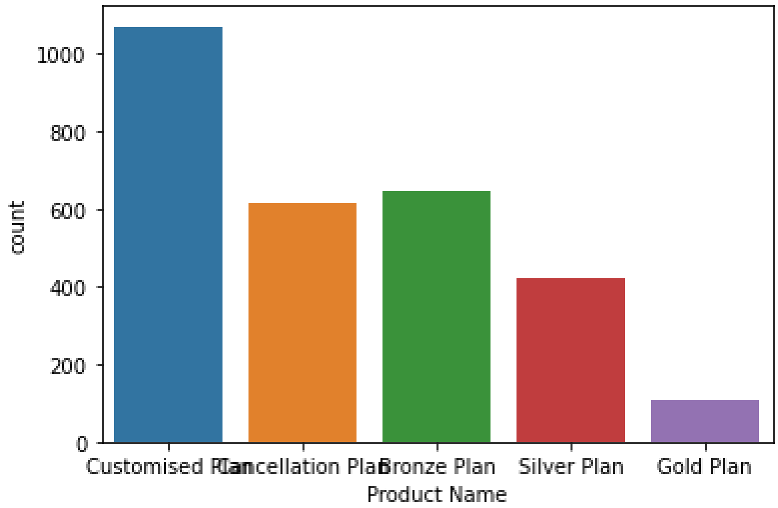


In [37]:

```
#Product
sns.countplot(data = insurance_df, x = 'Product Name')
```

Out[37]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24e08879eb0>
```

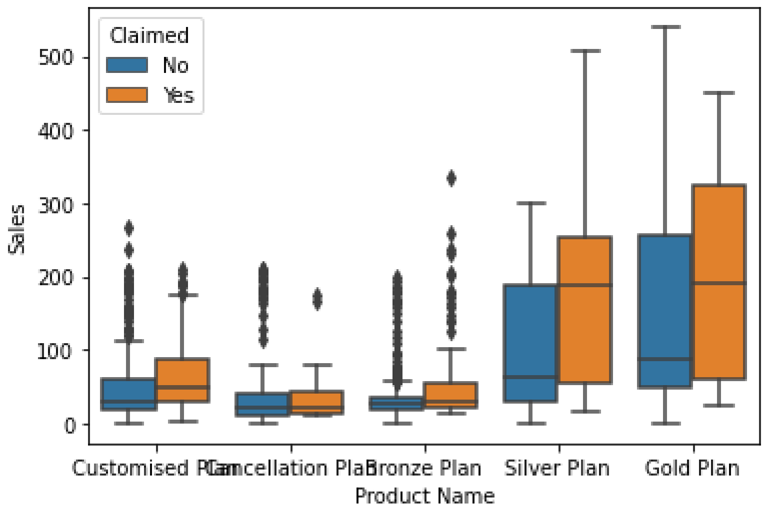


In [38]:

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

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24e088a9fd0>
```

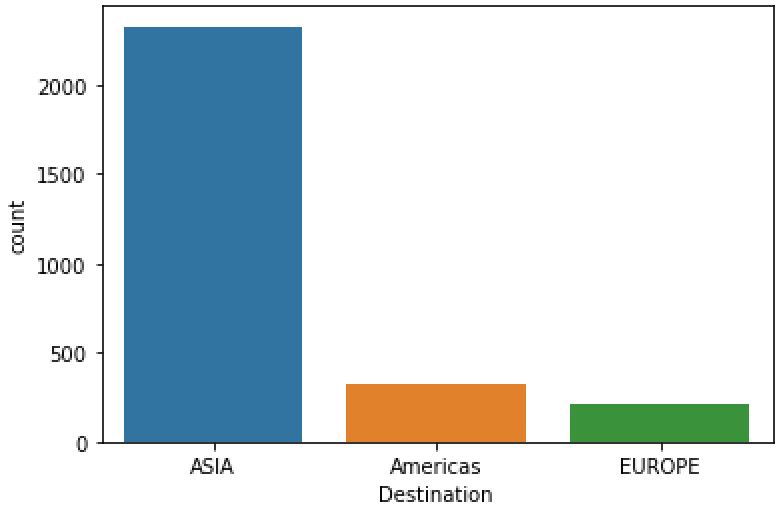


In [39]:

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

Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24e062b1d90>
```

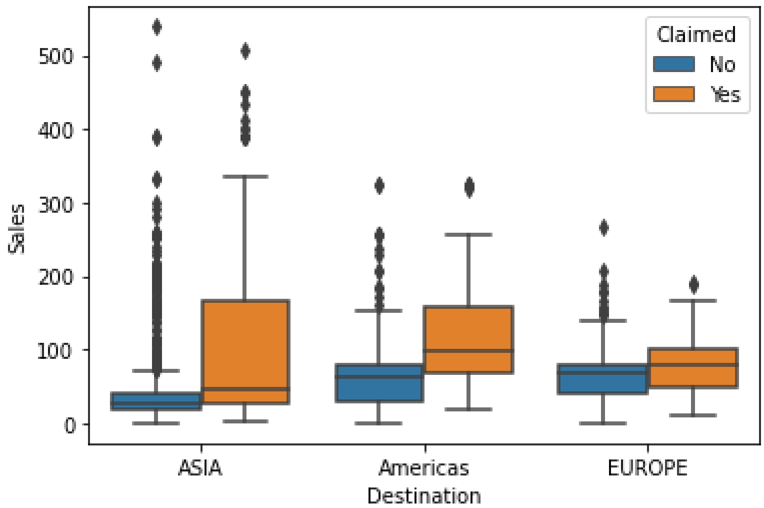


In [40]:

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

Out[40]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24e0625d9d0>
```



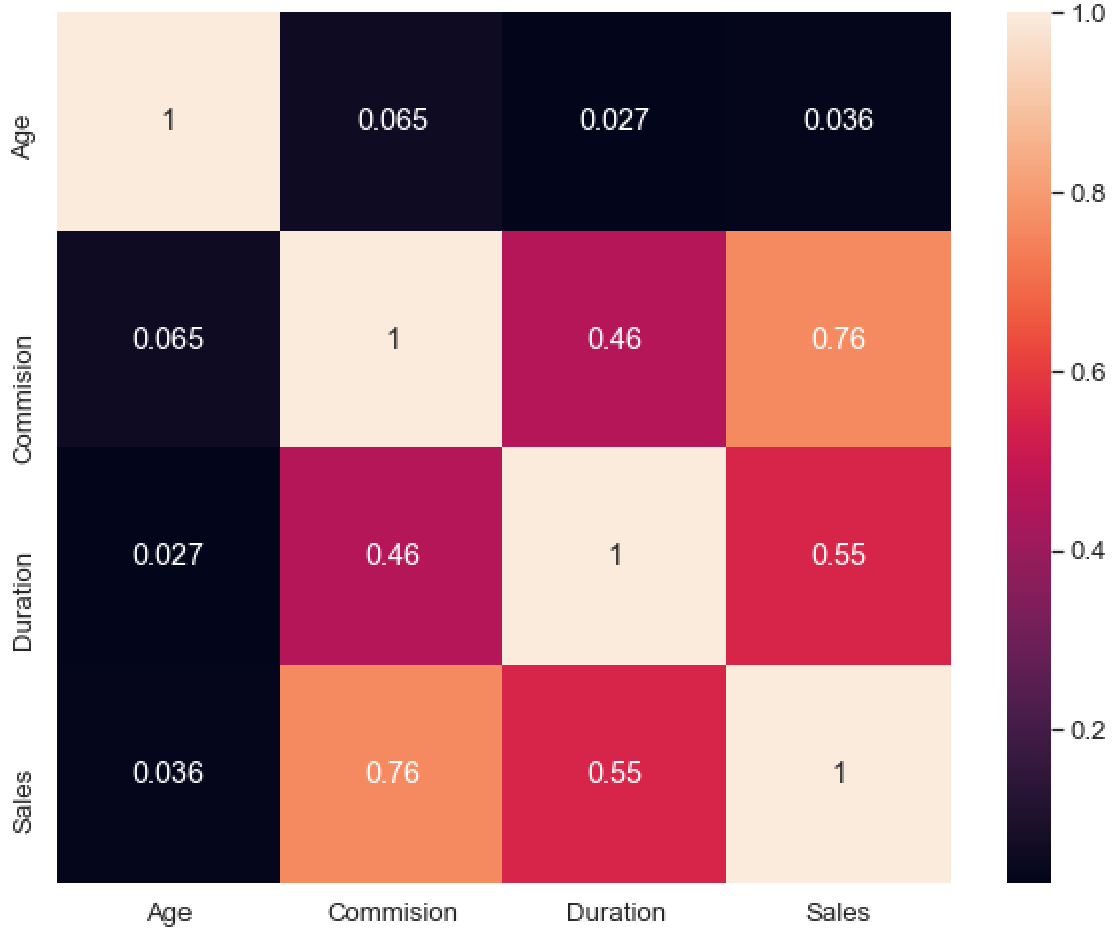
Checking for Correlations

In [41]:

```
#construct heatmap with only continuous variables
plt.figure(figsize=(10,8))
sns.set(font_scale=1.2)
sns.heatmap(insurance_df[["Age","Commision","Duration","Sales"]].corr(), annot=True)
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24e062910d0>
```



There are mostly positive correlation between different attributes. Only the "Sales" & "Commision" are higly correlated.

Converting all objects to categorical codes

In [42]:

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

```


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


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


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


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


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


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

In [43]:

```
insurance_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2861 entries, 0 to 2999
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Age           2861 non-null   int64  
 1   Agency_Code   2861 non-null   int8   
 2   Type          2861 non-null   int8   
 3   Claimed       2861 non-null   int8   
 4   Commision     2861 non-null   float64
 5   Channel       2861 non-null   int8   
 6   Duration      2861 non-null   int64  
 7   Sales         2861 non-null   float64
 8   Product Name  2861 non-null   int8   
 9   Destination   2861 non-null   int8   
dtypes: float64(2), int64(2), int8(6)
memory usage: 208.5 KB
```

In [44]:

```
insurance_df.head()
```

Out[44]:

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	Destina
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	

Proportion of 1s and 0s

In [45]:

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

Out[45]:

```
0    0.680531
1    0.319469
Name: Claimed, dtype: float64
```

So Approx 68% of customers have not claimed there insurance & There is no issue of class imbalance here as we have reasonable proportions in both the classes. The model is giving an accuracy of 68%. let see the performance of the model after using the best grid paramaters

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

In [46]:

```
x=insurance_df.drop("Claimed",axis=1)

y=insurance_df.pop("Claimed")

x.head()
```

Out[46]:

	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

Splitting data into training and test set

In [47]:

```
from sklearn.model_selection import train_test_split

x_train, x_test, train_labels, test_labels = train_test_split(x, y, test_size=.30, random_s
```

Checking the dimensions of the training and test data

In [48]:

```
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 (2002, 9)
x_test (859, 9)
train_labels (2002,)
test_labels (859,)
```

In [49]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [50]:

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

In [51]:

```
insurance_model.fit(x_train,train_labels)
```

Out[51]:

```
DecisionTreeClassifier()
```

In [52]:

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

In [53]:

```
train_char_labels=["No","Yes"]
```

In [54]:

```
claimed_tree_file=open("D:\claimed_tree_file.dot","w")
```

In [55]:

```
dot_data=tree.export_graphviz(insurance_model,out_file=claimed_tree_file,feature_names=list
```

Finding Best Parameters using best grid

In [56]:

```
param_grid = {
    'criterion': ['gini'],
    'max_depth': [10,12,14,15],
    'min_samples_leaf': [90,100,110],
    'min_samples_split': [310,300,295],
}

dtcl = DecisionTreeClassifier(random_state=1)

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

In [57]:

```
grid_search.fit(x_train, train_labels)
print(grid_search.best_params_)
best_grid = grid_search.best_estimator_
best_grid
```

```
{'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 110, 'min_samples
_split': 300}
```

Out[57]:

```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=110,
                       min_samples_split=300, random_state=1)
```

Generating Tree using best parameters

In [58]:

```
train_char_label = ['no', 'yes']
tree_regularized = open("D:\claimed_tree_file.dot","w")
dot_data = tree.export_graphviz(best_grid, out_file= tree_regularized , feature_names = lis

tree_regularized.close()
dot_data
```

Variables Importance

In [59]:

```
rint (pd.DataFrame(best_grid.feature_importances_, columns = ["Imp"], index = x_train.columr
```

```
                   Imp
Agency_Code   0.624673
Sales         0.239683
Product Name  0.098589
Duration      0.022802
Commision     0.007980
Age           0.006274
Type          0.000000
Channel       0.000000
Destination   0.000000
```

looking at the above important parameters the model higly depends upon at "Agency Code" i.e 62.46% and "Sales" i.e 23.9%

Predicting on Training and testing data

In [60]:

```
ytrain_predict = best_grid.predict(x_train)
ytest_predict = best_grid.predict(x_test)
```

Getting the Predicted Classes and Probs

In [61]:

```
ytest_predict
ytest_predict_prob=best_grid.predict_proba(x_test)
ytest_predict_prob
pd.DataFrame(ytest_predict_prob).head(10)
```

Out[61]:

	0	1
0	0.309091	0.690909
1	0.682927	0.317073
2	0.787129	0.212871
3	0.309091	0.690909
4	0.787129	0.212871
5	0.655172	0.344828
6	0.554054	0.445946
7	0.682927	0.317073
8	0.686957	0.313043
9	0.181818	0.818182

Model Evaluation

AUC and ROC for the training data

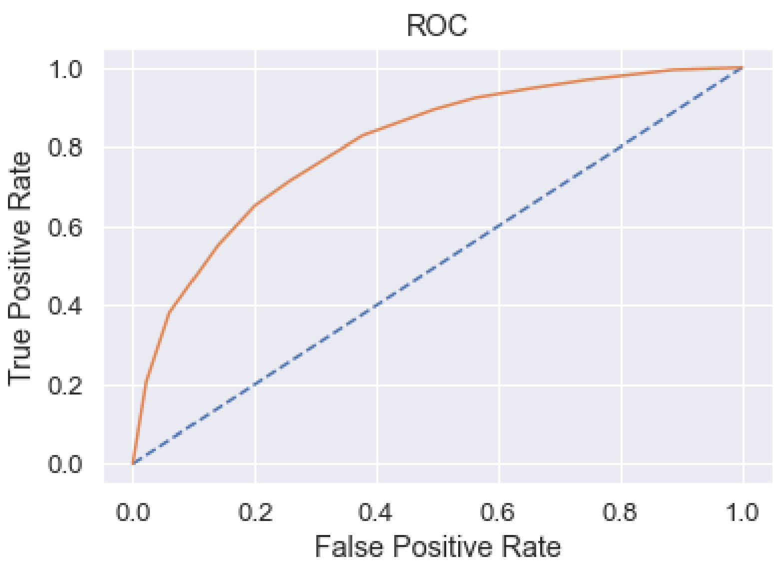
In [62]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
# predict probabilities
probs = best_grid.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
insurance_train_auc = roc_auc_score(train_labels, probs)
print('AUC: %.3f' % insurance_train_auc)
# calculate roc curve
insurance_train_fpr, insurance_train_tpr,insurance_train_thresholds = roc_curve(train_label
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
# plot the roc curve for the model
plt.plot(insurance_train_fpr, insurance_train_tpr)
```

```
AUC: 0.806
```

Out[62]:

```
[<matplotlib.lines.Line2D at 0x24e05e6a5e0>]
```



AUC and ROC for the test data

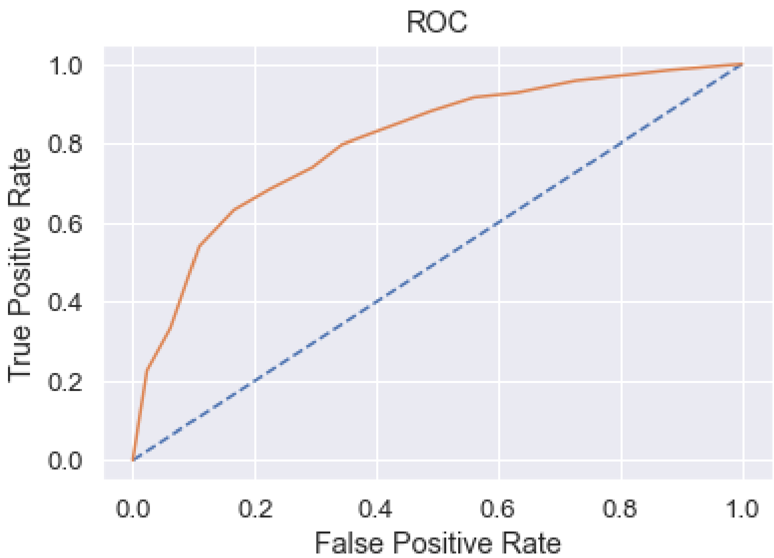
In [63]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
# predict probabilities
probs = best_grid.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
insurance_test_auc = roc_auc_score(test_labels, probs)
print('AUC: %.3f' % insurance_test_auc)
# calculate roc curve
insurance_test_fpr, insurance_test_tpr,insurance_test_thresholds = roc_curve(test_labels, p
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
# plot the roc curve for the model
plt.plot(insurance_test_fpr, insurance_test_tpr)
```

```
AUC: 0.803
```

Out[63]:

```
[<matplotlib.lines.Line2D at 0x24e09025df0>]
```



Confusion Matrix for the training data

In [64]:

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

In [65]:

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

Out[65]:

```
array([[1161,  188],
       [ 293,  360]], dtype=int64)
```

In [66]:

```
#Train Data Accuracy
insurance_train_acc=best_grid.score(x_train,train_labels)
insurance_train_acc
```

Out[66]:

```
0.7597402597402597
```

The model is tuned now and increases the accuracy from 68% to 75.9%

In [67]:

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

```
              precision    recall  f1-score   support

           0       0.80      0.86      0.83      1349
           1       0.66      0.55      0.60       653

    accuracy                           0.76      2002
   macro avg       0.73      0.71      0.71      2002
weighted avg       0.75      0.76      0.75      2002
```

In [68]:

```
insurance_metrics=classification_report(train_labels, ytrain_predict,output_dict=True)
df=pd.DataFrame(insurance_metrics).transpose()
insurance_train_f1=round(df.loc["1"][2],2)
insurance_train_recall=round(df.loc["1"][1],2)
insurance_train_precision=round(df.loc["1"][0],2)
print ('insurance_train_precision ',insurance_train_precision)
print ('insurance_train_recall ',insurance_train_recall)
print ('insurance_train_f1 ',insurance_train_f1)
```

```
insurance_train_precision  0.66
insurance_train_recall  0.55
insurance_train_f1  0.6
```

Confusion Matrix for test data

In [69]:

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

Out[69]:

```
array([[533,  65],
       [120, 141]], dtype=int64)
```

In [70]:

```
#Test Data Accuracy
insurance_test_acc=best_grid.score(x_test,test_labels)
insurance_test_acc
```

Out[70]:

```
0.7846332945285215
```

In [71]:

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

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       598
           1       0.68      0.54      0.60       261

    accuracy                           0.78       859
   macro avg       0.75      0.72      0.73       859
weighted avg       0.78      0.78      0.78       859
```

In [72]:

```
insurance_metrics=classification_report(test_labels, ytest_predict,output_dict=True)
df=pd.DataFrame(insurance_metrics).transpose()
insurance_test_f1=round(df.loc["1"][2],2)
insurance_test_recall=round(df.loc["1"][1],2)
insurance_test_precision=round(df.loc["1"][0],2)
print ('insurance_test_precision ',insurance_test_precision)
print ('insurance_test_recall ',insurance_test_recall)
print ('insurance_test_f1 ',insurance_test_f1)
```

```
insurance_test_precision  0.68
insurance_test_recall  0.54
insurance_test_f1  0.6
```

Cart Conclusion

Train Data:
AUC: 80.6%
Accuracy: 75.9%
Precision: 66%
f1-Score: 60%

Test Data:
AUC: 80.3%
Accuracy: 78.4%
Precision: 68%
f1-Score: 60%

Training and Test set results are almost similar, and with the overall measures high, the model is a good model.

Agency_code is the most important variable for predicting insurance claimed.

Building a Random Forest Classifier

In [73]:

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

In [74]:

```
rfcl=RandomForestClassifier( n_estimators=500,
                            oob_score=True,
                            max_depth=10,
                            max_features=5,
                            min_samples_leaf=21,
                            min_samples_split=60)
```

In [75]:

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

Out[75]:

```
RandomForestClassifier(max_depth=10, max_features=5, min_samples_leaf=21,
                       min_samples_split=60, n_estimators=500, oob_score=Tru
e)
```

In [76]:

```
rfcl.oob_score_
```

Out[76]:

```
0.7597402597402597
```

In [77]:

```
param_grid={'n_estimators':[301,501],
            'max_depth':[10,20],
            'min_samples_leaf':[21,22],
            'min_samples_split':[60,70],
            'max_features':[5,6],
}
```

In [78]:

```
rfcl=RandomForestClassifier()
```

In [79]:

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

In [80]:

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

Out[80]:

```
GridSearchCV(cv=3, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [10, 20], 'max_features': [5, 6],
                         'min_samples_leaf': [21, 22],
                         'min_samples_split': [60, 70],
                         'n_estimators': [301, 501]})
```

In [81]:

```
grid_search.best_params_
```

Out[81]:

```
{'max_depth': 20,
 'max_features': 6,
 'min_samples_leaf': 22,
 'min_samples_split': 60,
 'n_estimators': 301}
```

In [82]:

```
best_grid=grid_search.best_estimator_
```

In [83]:

```
best_grid
```

Out[83]:

```
RandomForestClassifier(max_depth=20, max_features=6, min_samples_leaf=22,
                       min_samples_split=60, n_estimators=301)
```

Predicting the Training and Testing data

In [84]:

```
ytrain_predict = best_grid.predict(x_train)
ytest_predict = best_grid.predict(x_test)
```

RF Model Performance Evaluation on Training data

In [85]:

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

Out[85]:

```
array([[1197,  152],
       [ 271,  382]], dtype=int64)
```

In [86]:

```
rf_train_acc=best_grid.score(x_train,train_labels)
rf_train_acc
```

Out[86]:

```
0.7887112887112887
```

In [87]:

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

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85      1349
           1       0.72      0.58      0.64       653

    accuracy                           0.79      2002
   macro avg       0.77      0.74      0.75      2002
weighted avg       0.78      0.79      0.78      2002
```

In [88]:

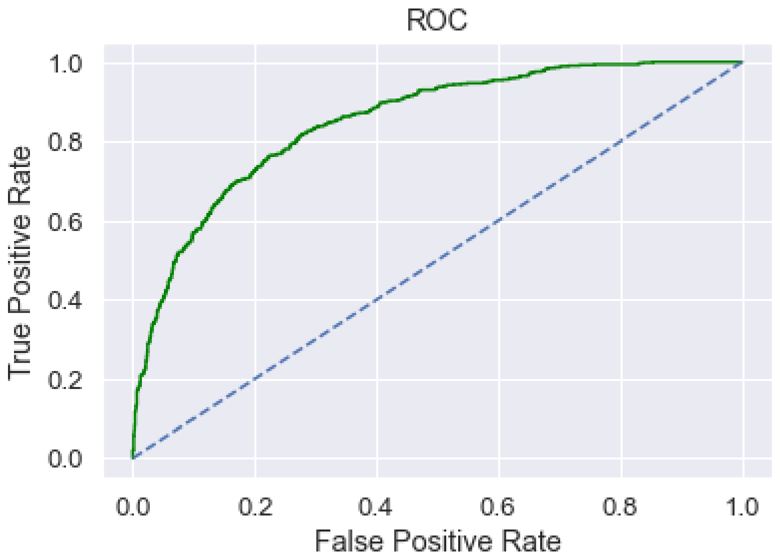
```
rf_metrics=classification_report(train_labels, ytrain_predict,output_dict=True)
df=pd.DataFrame(rf_metrics).transpose()
rf_train_precision=round(df.loc["1"][0],2)
rf_train_recall=round(df.loc["1"][1],2)
rf_train_f1=round(df.loc["1"][2],2)
print ('rf_train_precision ',rf_train_precision)
print ('rf_train_recall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

```
rf_train_precision  0.72
rf_train_recall  0.58
rf_train_f1  0.64
```

In [89]:

```
rf_train_fpr, rf_train_tpr,_=roc_curve(train_labels,best_grid.predict_proba(x_train)[:,1])
plt.plot(rf_train_fpr,rf_train_tpr,color='green')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
rf_train_auc=roc_auc_score(train_labels,best_grid.predict_proba(x_train)[:,1])
print('Area under Curve is', rf_train_auc)
```

```
Area under Curve is 0.8498848333006015
```



RF Model Performance Evaluation on Test data

In [90]:

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

Out[90]:

```
array([[534,  64],
       [128, 133]], dtype=int64)
```

In [91]:

```
rf_test_acc=best_grid.score(x_test,test_labels)
rf_test_acc
```

Out[91]:

```
0.7764842840512224
```

In [92]:

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

```
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       598
           1       0.68      0.51      0.58       261

    accuracy                           0.78       859
   macro avg       0.74      0.70      0.71       859
weighted avg       0.77      0.78      0.77       859
```

In [93]:

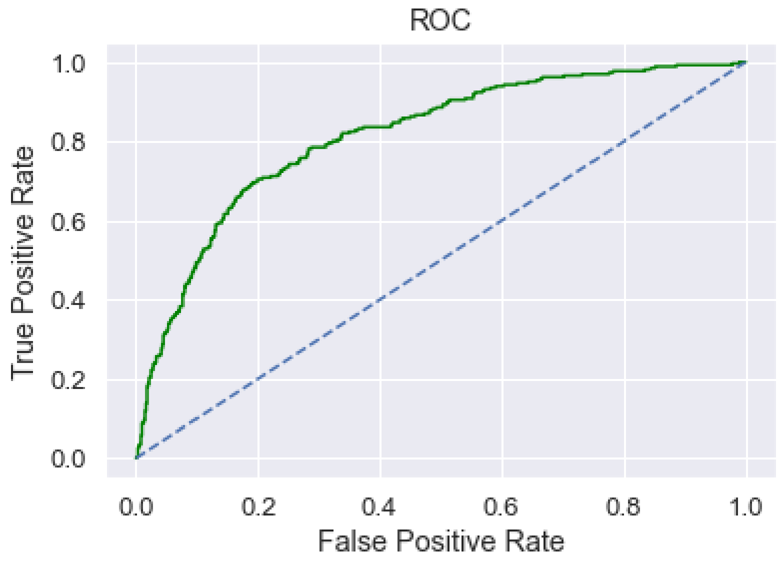
```
rf_metrics=classification_report(test_labels, ytest_predict,output_dict=True)
df=pd.DataFrame(rf_metrics).transpose()
rf_test_precision=round(df.loc["1"][0],2)
rf_test_recall=round(df.loc["1"][1],2)
rf_test_f1=round(df.loc["1"][2],2)
print ('rf_test_precision ',rf_test_precision)
print ('rf_test_recall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

```
rf_test_precision  0.68
rf_test_recall  0.51
rf_test_f1  0.58
```

In [94]:

```
rf_test_fpr, rf_test_tpr,_=roc_curve(test_labels,best_grid.predict_proba(x_test)[:,1])
plt.plot(rf_test_fpr,rf_test_tpr,color='green')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
rf_test_auc=roc_auc_score(test_labels,best_grid.predict_proba(x_test)[:,1])
print('Area under Curve is', rf_test_auc)
```

```
Area under Curve is 0.8133016824920872
```



In [95]:

```
# Variable Importance
print (pd.DataFrame(best_grid.feature_importances_, columns = ["Imp"], index = x_train.colu
```

```
                   Imp
Agency_Code   0.357309
Product Name  0.233230
Sales         0.189522
Commision     0.070150
Duration      0.067550
Age           0.052655
Destination   0.017758
Type          0.011695
Channel       0.000131
```

Random Forest Conclusion

Train Data:
AUC: 84.9%
Accuracy: 78.8%
Precision: 72%
f1-Score: 0.64%

Test Data:
AUC: 81.5%
Accuracy: 77%
Precision: 68%
f1-Score: 58%

Training and Test set results are almost similar, and with the overall measures high, the model is a good modeAgency_code is again the most important variable for predicting customer insurance claim

Building a Neural Network Classifier

In [96]:

```
from sklearn.neural_network import MLPClassifier
```

Scaling the data using standardScaler

In [97]:

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

In [98]:

```
sc=StandardScaler()
```

In [99]:

```
x_train=sc.fit_transform(x_train)
```

In [100]:

```
x_train
```

Out[100]:

```
array([[ 1.48754204, -1.24662389, -1.19074531, ...,  3.4001114 ,
         1.80654211, -0.442239  ],
       [ 2.895963  , -1.24662389, -1.19074531, ..., -0.16849295,
         1.80654211, -0.442239  ],
       [-0.01477365, -0.25585731,  0.83981015, ..., -0.31339648,
         0.25597521,  3.04344005],
       ...,
       [-1.14151041, -1.24662389, -1.19074531, ..., -0.57947912,
        -1.2945917 , -0.442239  ],
       [-0.39035257, -0.25585731,  0.83981015, ...,  0.10852848,
         0.25597521,  3.04344005],
       [-0.29645784, -1.24662389, -1.19074531, ..., -0.40005446,
        -1.2945917 , -0.442239  ]])
```

In [101]:

```
x_test=sc.transform(x_test)
```

In [102]:

```
x_test
```

Out[102]:

```
array([[ 0.36080528, -1.24662389, -1.19074531, ...,  1.16973368,
         1.80654211, -0.442239  ],
       [-0.20256311,  0.73490928,  0.83981015, ..., -0.29350776,
         0.25597521, -0.442239  ],
       [-0.20256311, -0.25585731,  0.83981015, ..., -0.31339648,
         0.25597521,  3.04344005],
       ...,
       [ 0.92417366, -1.24662389, -1.19074531, ..., -0.77651949,
         0.25597521, -0.442239  ],
       [-0.20256311,  0.73490928,  0.83981015, ..., -0.57763231,
         0.25597521, -0.442239  ],
       [-0.95372095,  1.72567587, -1.19074531, ..., -0.44977626,
        -1.2945917 , -0.442239  ]])
```

In [103]:

```
param_grid = {
    'hidden_layer_sizes': [520,100,500],
    'max_iter': [2500,3000],
    'solver': ['adam'],
    'tol': [0.01],
}

nncl = MLPClassifier(random_state=1)

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

In [104]:

```
grid_search.fit(x_train, train_labels)
grid_search.best_params_
```

Out[104]:

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

In [105]:

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

Out[105]:

```
MLPClassifier(hidden_layer_sizes=100, max_iter=2500, random_state=1, tol=0.0
1)
```

Predicting the Training and Testing data

In [106]:

```
ytrain_predict = best_grid.predict(x_train)
ytest_predict = best_grid.predict(x_test)
```

NN Model Performance Evaluation on Training data

In [107]:

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

Out[107]:

```
array([[1189,  160],
       [ 325,  328]], dtype=int64)
```

In [108]:

```
nn_train_acc=best_grid.score(x_train,train_labels)
nn_train_acc
```

Out[108]:

```
0.7577422577422578
```

In [109]:

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

```
              precision    recall  f1-score   support

           0       0.79      0.88      0.83      1349
           1       0.67      0.50      0.57       653

    accuracy                           0.76      2002
   macro avg       0.73      0.69      0.70      2002
weighted avg       0.75      0.76      0.75      2002
```

In [110]:

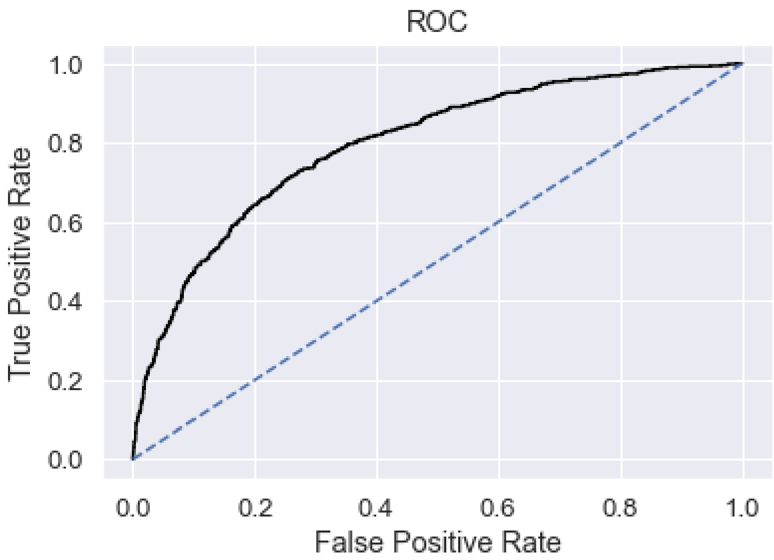
```
nn_metrics=classification_report(train_labels, ytrain_predict,output_dict=True)
df=pd.DataFrame(nn_metrics).transpose()
nn_train_precision=round(df.loc["1"][0],2)
nn_train_recall=round(df.loc["1"][1],2)
nn_train_f1=round(df.loc["1"][2],2)
print ('nn_train_precision ',nn_train_precision)
print ('nn_train_recall ',nn_train_recall)
print ('nn_train_f1 ',nn_train_f1)
```

```
nn_train_precision  0.67
nn_train_recall  0.5
nn_train_f1  0.57
```

In [111]:

```
nn_train_fpr, nn_train_tpr,_=roc_curve(train_labels,best_grid.predict_proba(x_train)[:,1])
plt.plot(nn_train_fpr,nn_train_tpr,color='black')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
nn_train_auc=roc_auc_score(train_labels,best_grid.predict_proba(x_train)[:,1])
print('Area under Curve is', nn_train_auc)
```

```
Area under Curve is 0.7958881685372977
```



NN Model Performance Evaluation on Test data

In [112]:

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

Out[112]:

```
array([[532,  66],
       [137, 124]], dtype=int64)
```

In [113]:

```
nn_test_acc=best_grid.score(x_test,test_labels)
nn_test_acc
```

Out[113]:

```
0.7636786961583236
```

In [114]:

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

```
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       598
           1       0.65      0.48      0.55       261

    accuracy                           0.76       859
   macro avg       0.72      0.68      0.69       859
weighted avg       0.75      0.76      0.75       859
```

In [115]:

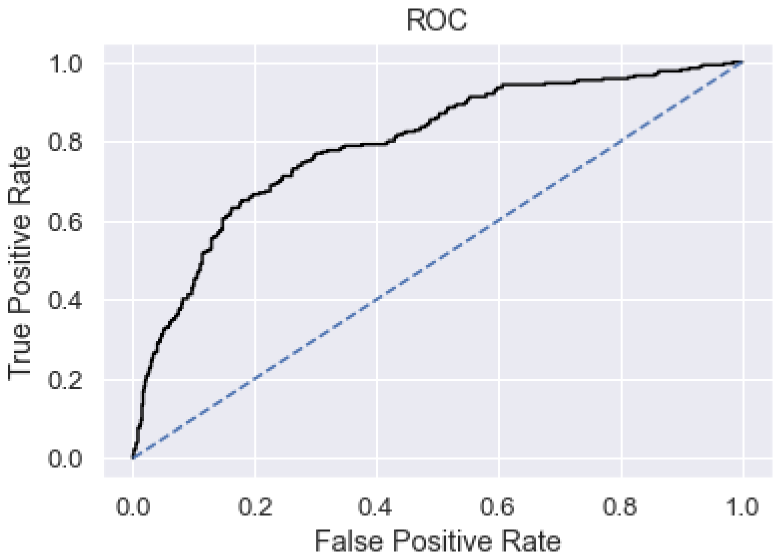
```
nn_metrics=classification_report(test_labels, ytest_predict,output_dict=True)
df=pd.DataFrame(nn_metrics).transpose()
nn_test_precision=round(df.loc["1"][0],2)
nn_test_recall=round(df.loc["1"][1],2)
nn_test_f1=round(df.loc["1"][2],2)
print ('nn_test_precision ',nn_test_precision)
print ('nn_test_recall ',nn_test_recall)
print ('nn_test_f1 ',nn_test_f1)
```

```
nn_test_precision  0.65
nn_test_recall  0.48
nn_test_f1  0.55
```

In [116]:

```
nn_test_fpr, nn_test_tpr,_=roc_curve(test_labels,best_grid.predict_proba(x_test)[:,1])
plt.plot(nn_test_fpr,nn_test_tpr,color='black')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
nn_test_auc=roc_auc_score(test_labels,best_grid.predict_proba(x_test)[:,1])
print('Area under Curve is', nn_test_auc)
```

```
Area under Curve is 0.7933276951267956
```



In [117]:

```
best_grid.score
```

Out[117]:

```
<bound method ClassifierMixin.score of MLPClassifier(hidden_layer_sizes=100,
max_iter=2500, random_state=1, tol=0.01)>
```

Neural Network Conclusion

Train Data:
AUC: 79.5%
Accuracy: 67%
Precision: 50%
f1-Score: 57%

Test Data:
AUC: 73.3%
Accuracy: 51%
Precision: 54%
f1-Score: 52%

Training and Test set results are almost similar, and with the overall measures high, the model is a good model.

Final Conclusion

Comparison of the performance metrics from the 3 models

In [118]:

```
['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
 pd.DataFrame({'CART Train':[insurance_train_acc,insurance_train_auc,insurance_train_recall,
  'CART Test':[insurance_test_acc,insurance_test_auc,insurance_test_recall,insurance_test_pr
 'Random Forest Train':[rf_train_acc,rf_train_auc,rf_train_recall,rf_train_precision,rf_trai
  'Random Forest Test':[rf_test_acc,rf_test_auc,rf_test_recall,rf_test_precision,rf_test_f1]
 'Neural Network Train':[nn_train_acc,nn_train_auc,nn_train_recall,nn_train_precision,nn_tra
  'Neural Network Test':[nn_test_acc,nn_test_auc,nn_test_recall,nn_test_precision,nn_test_f1
data,2)
```

Out[118]:

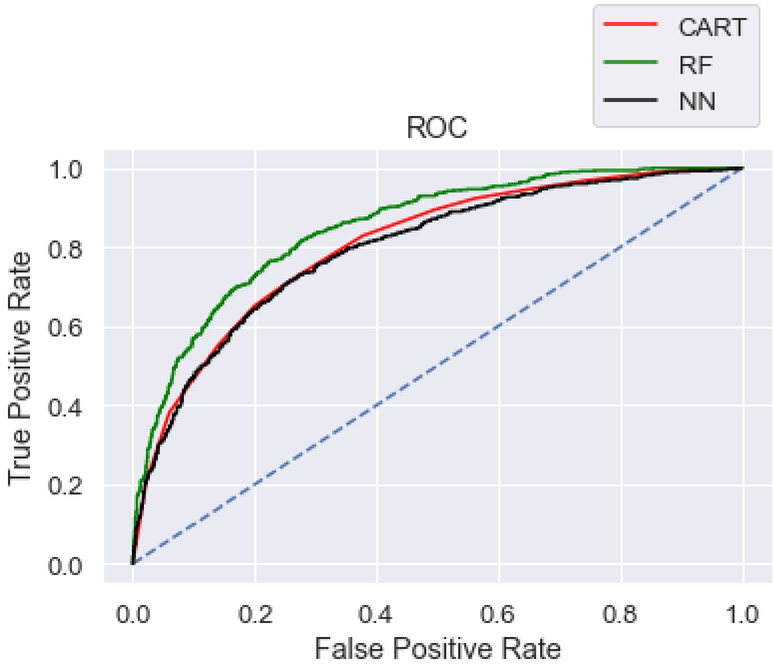
	CART Train	CART Test	Random Forest Train	Random Forest Test	Neural Network Train	Neural Network Test
Accuracy	0.76	0.78	0.79	0.78	0.76	0.76
AUC	0.81	0.80	0.85	0.81	0.80	0.79
Recall	0.55	0.54	0.58	0.51	0.50	0.48
Precision	0.66	0.68	0.72	0.68	0.67	0.65
F1 Score	0.60	0.60	0.64	0.58	0.57	0.55

In [119]:

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(insurance_train_fpr, insurance_train_tpr,color='red',label="CART")
plt.plot(rf_train_fpr,rf_train_tpr,color='green',label="RF")
plt.plot(nn_train_fpr,nn_train_tpr,color='black',label="NN")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower right')
```

Out[119]:

```
<matplotlib.legend.Legend at 0x24e091dd1c0>
```



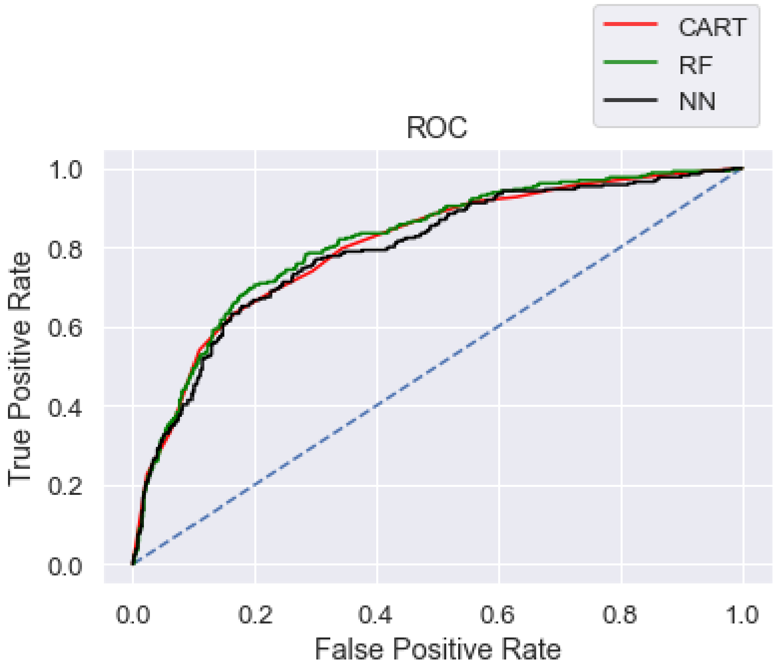
ROC Curve for the 3 models on the Test data

In [120]:

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(insurance_test_fpr,insurance_test_tpr,color='red',label="CART")
plt.plot(rf_test_fpr,rf_test_tpr,color='green',label="RF")
plt.plot(nn_test_fpr,nn_test_tpr,color='black',label="NN")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower right')
```

Out[120]:

```
<matplotlib.legend.Legend at 0x24e0922dac0>
```



In [ ]: