Data MINING ASSIGNMENT

Importing the required Libariers

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.cluster import KMeans
```

Import the dataset(BankMarketing.csv):

In [2]:

```
bank=pd.read_csv('bank_marketing.csv')
```

Reading top ten Records:

In [3]:

```
bank.head(10)
```

Out[3]:

	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	

Check the dimension of data:

In [4]:

```
bank.shape
```

Out[4]:

```
(210, 7)
```

Check the Information about the data and the datatypes of each respective attributes:

In [5]:

```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210 entries, 0 to 209
Data columns (total 7 columns):
spending                      210 non-null float64
advance_payments              210 non-null float64
probability_of_full_payment   210 non-null float64
current_balance               210 non-null float64
credit_limit                  210 non-null float64
min_payment_amt               210 non-null float64
max_spent_in_single_shopping  210 non-null float64
dtypes: float64(7)
memory usage: 11.6 KB
```

Need to check description of the data

In [6]:

```
bank.describe()
```

Out[6]:

	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

Need to check Median of data

In [7]:

```
print("Data:",bank.median())  # Print the median values of the data.
```

```
Data: spending                       14.35500
advance_payments                   14.32000
probability_of_full_payment         0.87345
current_balance                     5.52350
credit_limit                        3.23700
min_payment_amt                     3.59900
max_spent_in_single_shopping        5.22300
dtype: float64
```

Checking the Null Values

In [8]:

```
bank.isnull().sum()
```

Out[8]:

```
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
```

Checking the Duplicates Values

In [9]:

```
dupes =bank.duplicated()
sum(dupes)
```

Out[9]:

```
0
```

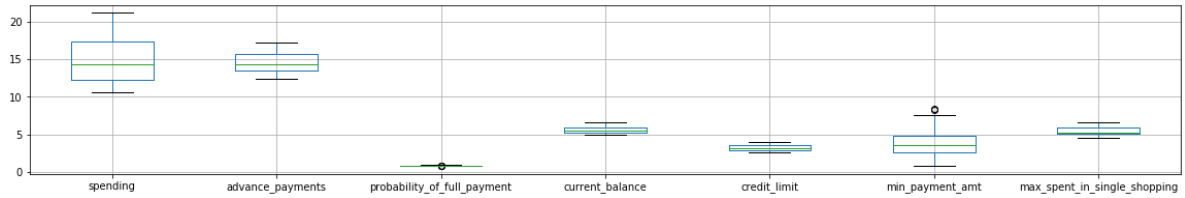
Checking the outliers through Box plot

In [10]:

```
bank.boxplot(figsize=(20,3))
```

Out[10]:

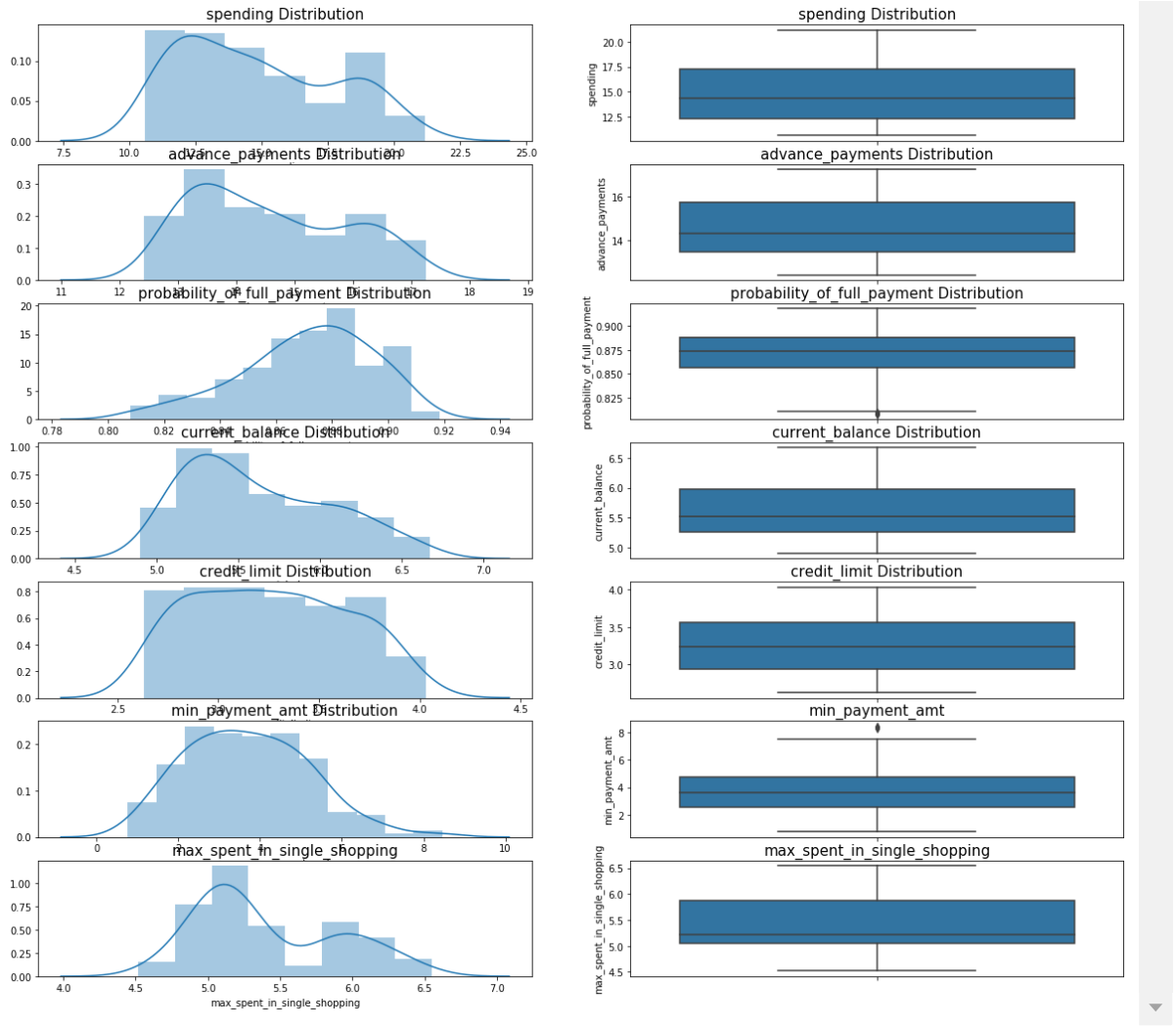
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e70af35bc8>
```



Checking the distribution of the data

In [11]:

```
fig, axes = plt.subplots(nrows=7,ncols=2)
fig.set_size_inches(20, 18)
a = sns.distplot(bank['spending'] , ax=axes[0][0])
a.set_title("spending Distribution",fontsize=15)
a = sns.boxplot(bank['spending'] , orient = "v" , ax=axes[0][1])
a.set_title("spending Distribution",fontsize=15)

a = sns.distplot(bank['advance_payments'] , ax=axes[1][0])
a.set_title("advance_payments Distribution",fontsize=15)

a = sns.boxplot(bank['advance_payments'] , orient = "v" , ax=axes[1][1])
a.set_title("advance_payments Distribution",fontsize=15)

a = sns.distplot(bank['probability_of_full_payment'] , ax=axes[2][0])
a.set_title("probability_of_full_payment Distribution",fontsize=15)

a = sns.boxplot(bank['probability_of_full_payment'] , orient = "v" , ax=axes[2][1])
a.set_title("probability_of_full_payment Distribution",fontsize=15)


a = sns.distplot(bank['current_balance'] , ax=axes[3][0])
a.set_title("current_balance Distribution",fontsize=15)

a = sns.boxplot(bank['current_balance'] , orient = "v" , ax=axes[3][1])
a.set_title("current_balance Distribution",fontsize=15)



a = sns.distplot(bank['credit_limit'] , ax=axes[4][0])
a.set_title("credit_limit Distribution",fontsize=15)

a = sns.boxplot(bank['credit_limit'] , orient = "v" , ax=axes[4][1])
a.set_title("credit_limit Distribution",fontsize=15)

a = sns.distplot(bank['min_payment_amt'] , ax=axes[5][0])
a.set_title("min_payment_amt Distribution",fontsize=15)
a = sns.boxplot(bank['min_payment_amt'] , orient = "v" , ax=axes[5][1])
a.set_title("min_payment_amt",fontsize=15)

a = sns.distplot(bank['max_spent_in_single_shopping'] , ax=axes[6][0])
a.set_title("max_spent_in_single_shopping",fontsize=15)
a = sns.boxplot(bank['max_spent_in_single_shopping'] , orient = "v" , ax=axes[6][1])
a.set_title("max_spent_in_single_shopping",fontsize=15)
plt.show()
```



In [12]:

```
# need to draw pair plot
```

Bi- Variate Analysis:

Checking the correlation of variable

In [13]:

```
bank.corr(method='pearson')
```

Out[13]:

	spending	advance_payments	probability_of_full_payment	current_
spending	1.000000	0.994341	0.608288	0
advance_payments	0.994341	1.000000	0.529244	0
probability_of_full_payment	0.608288	0.529244	1.000000	0
current_balance	0.949985	0.972422	0.367915	1
credit_limit	0.970771	0.944829	0.761635	0
min_payment_amt	-0.229572	-0.217340	-0.331471	-0
max_spent_in_single_shopping	0.863693	0.890784	0.226825	0

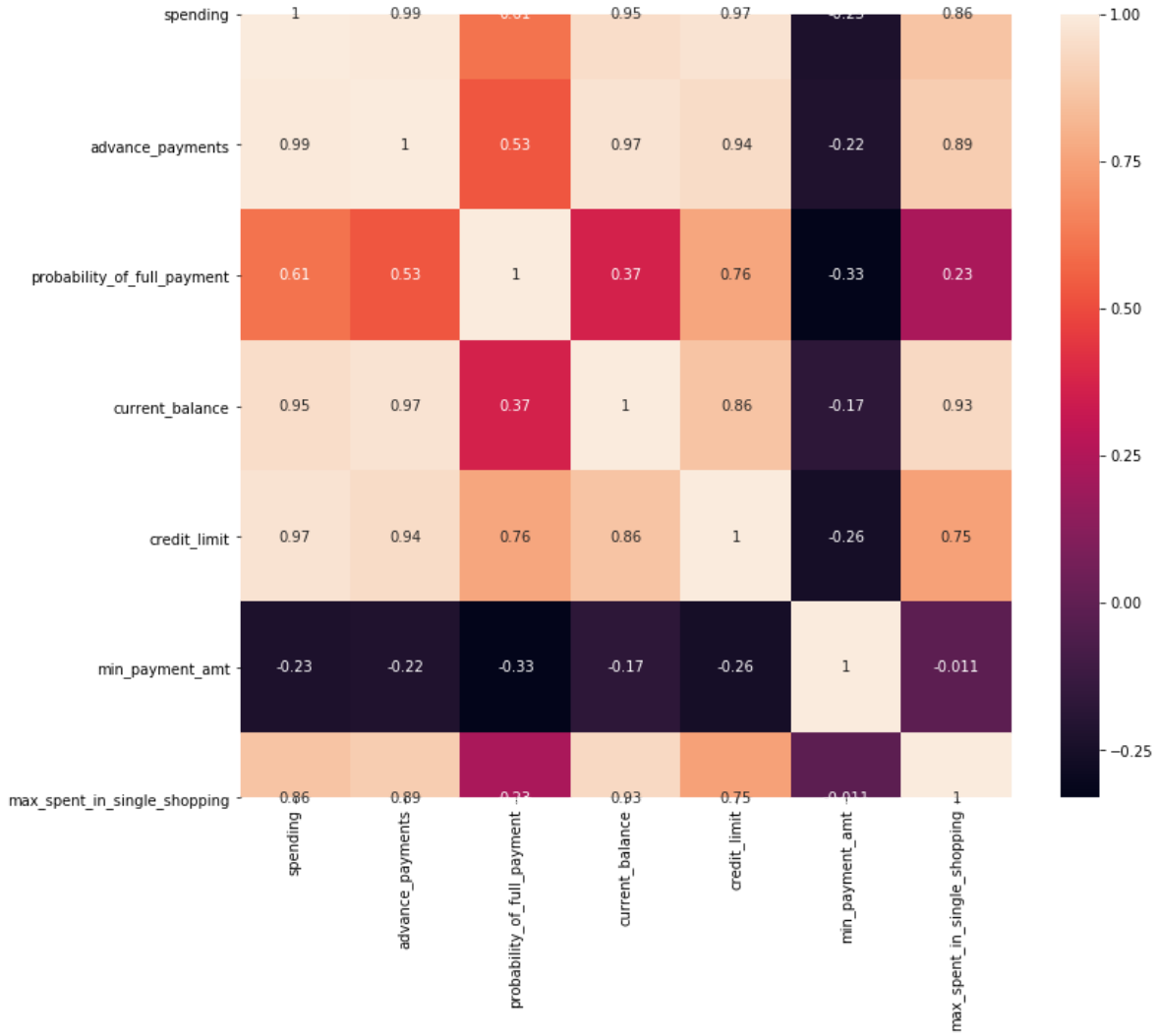
Checking the heatmap:

In [14]:

```
plt.subplots(figsize=(12,10))
sns.heatmap(bank.corr(), annot=True)  # plot the correlation coefficients as a heatmap
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e70b2b68c8>
```



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

EDA

- Bank dataset has 210 rows and 7 columns. Dataset is fairly clean no . There is no missing and duplicate rows in the dataset
- But there are outlliers in probability_of_full_payment and min_payment_amt.
- In the dataset most of the variables are not Normally distributed
- there are some variables which are high co-realted like Current balance and advance payment as well as Credit limit and Advance Payment

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

Why Scaling is requied for Clustering

- The reason this importance is particularly high in cluster analysis is because groups are defined based on the distance between points in Mathematical space.
- Standardization helps to make the relative weight of each variable equal by converting each variable to a unitless measure or relative distance.
- In the given Case Study some variable are (in 1000s) like pending,current_balance,max_spent_in_single_shopping.
- In the given Case Study some variable are (in 100s) like advance_payments,min_payment_amt.
- So Some Variables are 10 times of other variables in the measurement scale .
- Yes, We should do the scaling in the given Case Study

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

Scaling the data

In [15]:

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

In [16]:

```
X = StandardScaler()
```

In [17]:

```
scaled_bank = X.fit_transform(bank)
```

In [18]:

```
scaled_bank
```

Out[18]:

```
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]])
```

In [19]:

```
scaled_bank = pd.DataFrame(scaled_bank, index=bank.index, columns=bank.columns)
scaled_bank.head()
```

Out[19]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_
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	

Performing Hierarchical Clustering with the Ward's linkage method and plot the dendrogram.

In [20]:

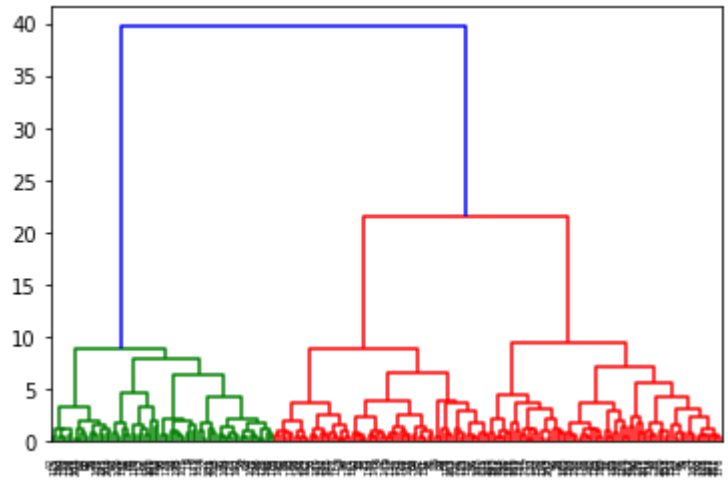
```
from scipy.cluster.hierarchy import dendrogram, linkage
```

In [21]:

```
HClust = linkage(scaled_bank, method = 'ward')
```

In [22]:

```
dend = dendrogram(HClust)
```



Identify the number of clusters based on the dendrogram

In [23]:

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

Mehtod1

In [24]:

```
clusters_1 = fcluster(HClust, 2, criterion='maxclust')
clusters_1
```

Out[24]:

```
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, 1, 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, 1,
       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, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1,
       2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2,
       1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2], dtype=int32)
```

Mehtod2

In [25]:

```
clusters_2 = fcluster(HClust, 22, criterion='distance')
clusters_2
```

Out[25]:

```
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, 1, 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, 1,
       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, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1,
       2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2,
       1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2], dtype=int32)
```

Checking both method produce same result

In [26]:

```
np.array_equal(clusters_1,clusters_2)
```

Out[26]:

```
True
```

Adding the H_Clusters column for Mapping

In [27]:

```
bank['H_clusters'] = clusters_1
```

Checking the bank dataset After Mapping the column

In [28]:

```
#ank = bank.drop('Clus_kmeans',axis=1)

bank.head()
```

Out[28]:

	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	

Optimum Number of Hierarchical Clusters

- In the above Mentiond Dendogram visual representation , Y axis mesaure the distance .
- if we take the distance 22 on y axis it is cutting the 2 vertical lines . So number of clusters would be two

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

Reading the top 5 records for checking the data

In [29]:

```

bank.head()
```

Out[29]:

	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	

In [30]:

```
from sklearn.cluster import KMeans
```

Performing the K-Means clustering with 2 clusters.

In [31]:

```
k_means = KMeans(n_clusters = 2)
```

In [32]:

```
k_means.fit(scaled_bank)
```

Out[32]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [33]:

```
k_means.labels_
```

Out[33]:

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

Calcuation of total within sum of squares for different k values

In [34]:

```
wss=[]
[]=wss
wss=[]
```

In [35]:

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

In [36]:

```
wss
```

Out[36]:

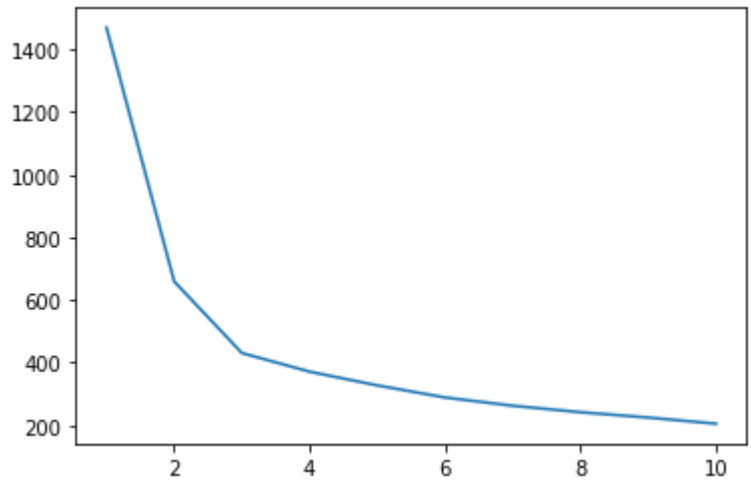
```
[1470.0,
 659.1717544870407,
 430.65897315130053,
 371.1846125351019,
 327.3281094192773,
 288.7694577022638,
 262.3667820456648,
 241.73766598079735,
 224.69807821588125,
 205.11670848885427]
```

In [37]:

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

Out[37]:

```
[<matplotlib.lines.Line2D at 0x1e70becf788>]
```



Determining the Optimum Number of K-Means clusters

- If you see total within sum of sqaure of values when k=1 (1470) and k=2(659) drop is significant.
- If we consider (k=3) there is not significant drop in total within sum of sqaure of values .
- So in this case k=2 would be optimal value in this case .That is also quite visible from the graph

In [38]:

```
k_means = KMeans(n_clusters = 2)
k_means.fit(scaled_bank)
labels = k_means.labels_
```

In [39]:

```
bank["Clus_kmeans"] = labels



bank.head(5)
```

Out[39]:

	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	

Applying the Silhouette Score

In [40]:

```


from sklearn.metrics import silhouette_samples, silhouette_score
```

In [41]:

```
silhouette_score(scaled_bank,labels)
```

Out[41]:

```
0.46577247686580914
```

Interpretation of Silhouette Score

- Average Silhouette Score of data is .46 and postive Number for 2 clusters .
- If We opt 3 clusters Average Silhouette Score of date id go down from .46 to .40 .SO We should stick to 2 Clusters.
- We can say two is the optimum number of clusters in given case study.

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

Cluster Frequency

In [42]:

```
bank.Clus_kmeans.value_counts().sort_index()
```

Out[42]:

```
0    133
1     77
Name: Clus_kmeans, dtype: int64
```

Cluster Profiles

In [43]:

```
clust_profile=bank
clust_profile=clust_profile.groupby('Clus_kmeans').mean()
clust_profile['freq']=bank.Clus_kmeans.value_counts().sort_index()
clust_profile
```

Out[43]:

Clus_kmeans	spending	advance_payments	probability_of_full_payment	current_balance	credit
0	12.930602	13.693459	0.863577	5.339699	3.0
1	18.158571	16.054805	0.883817	6.127429	3.6

Promotional Strategies for different clusters.

- There are 2 kind of Customer Segmentation in the given dataset .
- Customer Pertaining to First Cluster , who do the more shopping and do the single payment .
- Customer Pertaining to Second Cluster ,Who do the shopping by paying Minimum amount.
- There are 36.6 percent of customer belong to first cluster and 63.3 percent of customer belong to Second Cluster.
- Bank should bring offer of discount on Advance Payment to attract the customer of first cluster.
- Bank Should bring the scheme of increasing the Credit Limit of customer pertaining to 2nd cluster.

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

Import the dataset:

In [44]:

```
insur=pd.read_csv('insurance.csv')
```

Reading top ten Records:

In [45]:

```
insur.head(10)
```

Out[45]:

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

Check the dimension of data:

In [46]:

```
insur.shape
```

Out[46]:

```
(3000, 10)
```

Check the Information about the data and the datatypes of each respective attributes:

In [47]:

```
insur.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 10 columns):
Age             3000 non-null int64
Agency_Code     3000 non-null object
Type            3000 non-null object
Claimed         3000 non-null object
Commision       3000 non-null float64
Channel         3000 non-null object
Duration        3000 non-null int64
Sales           3000 non-null float64
Product Name    3000 non-null object
Destination     3000 non-null object
dtypes: float64(2), int64(2), object(6)
memory usage: 234.5+ KB
```

Need to check description of the data

In [48]:

```
insur.describe()
```

Out[48]:

	Age	Commision	Duration	Sales
count	3000.000000	3000.000000	3000.000000	3000.000000
mean	38.091000	14.529203	70.001333	60.249913
std	10.463518	25.481455	134.053313	70.733954
min	8.000000	0.000000	-1.000000	0.000000
25%	32.000000	0.000000	11.000000	20.000000
50%	36.000000	4.630000	26.500000	33.000000
75%	42.000000	17.235000	63.000000	69.000000
max	84.000000	210.210000	4580.000000	539.000000

Need to check Median of data

In [49]:

```
insur.median()
```

Out[49]:

```
Age          36.00
Commision     4.63
Duration     26.50
Sales        33.00
dtype: float64
```

Checking the Null Values

In [50]:

```
insur.isnull().sum().sum()
```

Out[50]:

```
0
```

Checking the Duplicates Values

In [51]:

```
dupes=insur.duplicated()
sum(dupes)
```

Out[51]:

```
139
```

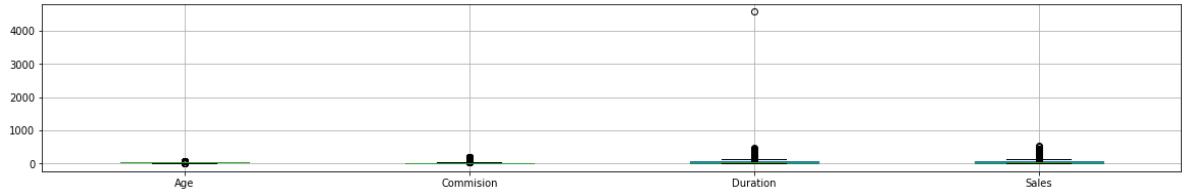
Checking the outliers through Box plot

In [52]:

```
insur.boxplot(figsize=(20,3))
```

Out[52]:

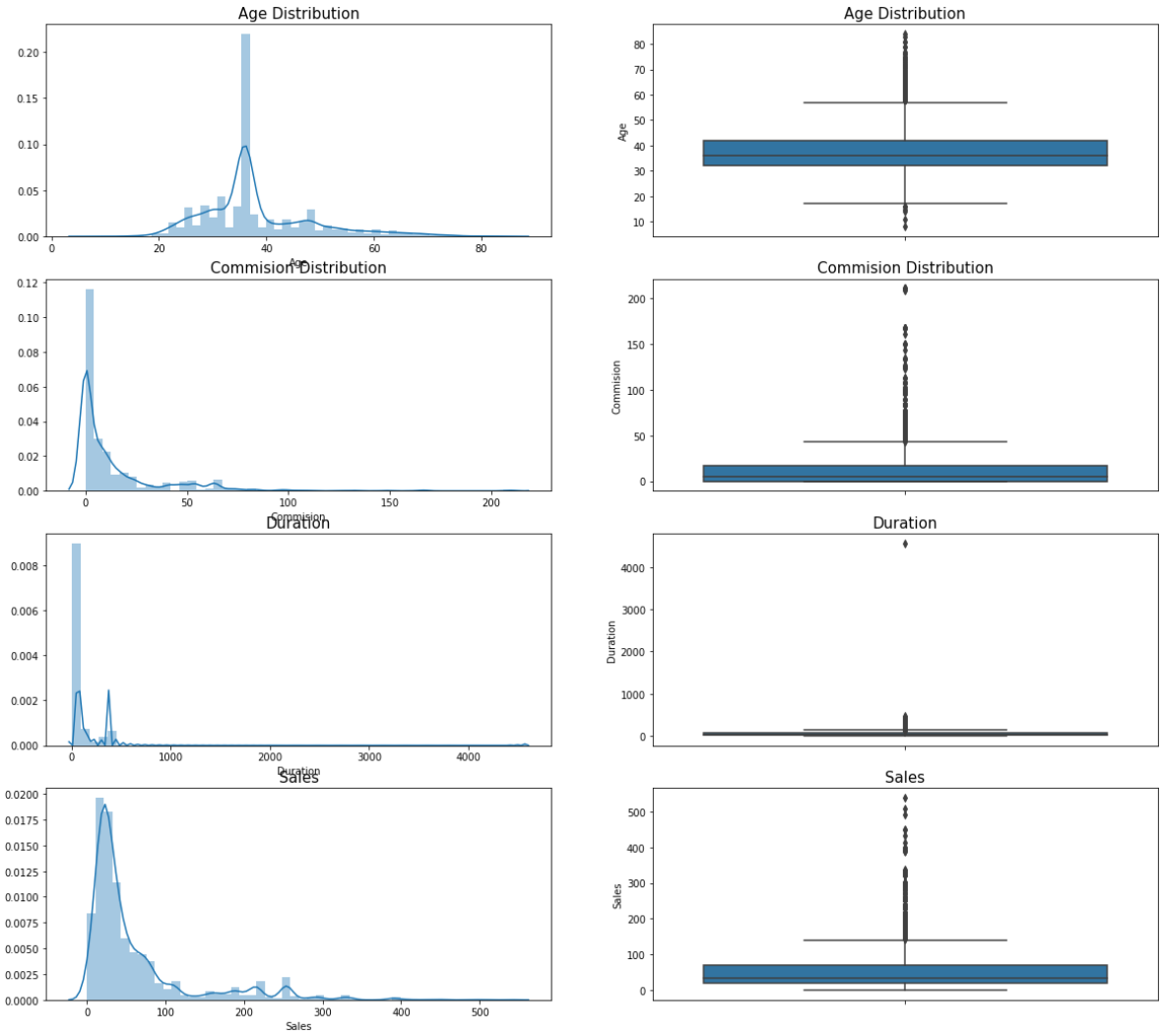
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e70bf52fc8>
```



Checking the distribution of the data

In [53]:

```
fig, axes = plt.subplots(nrows=4,ncols=2)
fig.set_size_inches(20, 18)
a = sns.distplot(insur['Age'] , ax=axes[0][0])
a.set_title("Age Distribution",fontsize=15)
a = sns.boxplot(insur['Age'] , orient = "v" , ax=axes[0][1])
a.set_title("Age Distribution",fontsize=15)

a = sns.distplot(insur['Commision'] , ax=axes[1][0])
a.set_title("Commision Distribution",fontsize=15)
a = sns.boxplot(insur['Commision'] , orient = "v" , ax=axes[1][1])
a.set_title("Commision Distribution",fontsize=15)


a = sns.distplot(insur['Duration'] , ax=axes[2][0])
a.set_title("Duration",fontsize=15)
a = sns.boxplot(insur['Duration'] , orient = "v" , ax=axes[2][1])
a.set_title("Duration",fontsize=15)


a = sns.distplot(insur['Sales'] , ax=axes[3][0])
a.set_title("Sales",fontsize=15)
a = sns.boxplot(insur['Sales'] , orient = "v" , ax=axes[3][1])
a.set_title("Sales",fontsize=15)

plt.show()
```



Bi- Variate Analysis:

Checking the correlation of variable

In [54]:

```
insur.corr(method='pearson')
```

Out[54]:

	Age	Commision	Duration	Sales
Age	1.000000	0.067717	0.030425	0.039455
Commision	0.067717	1.000000	0.471389	0.766505
Duration	0.030425	0.471389	1.000000	0.558930
Sales	0.039455	0.766505	0.558930	1.000000

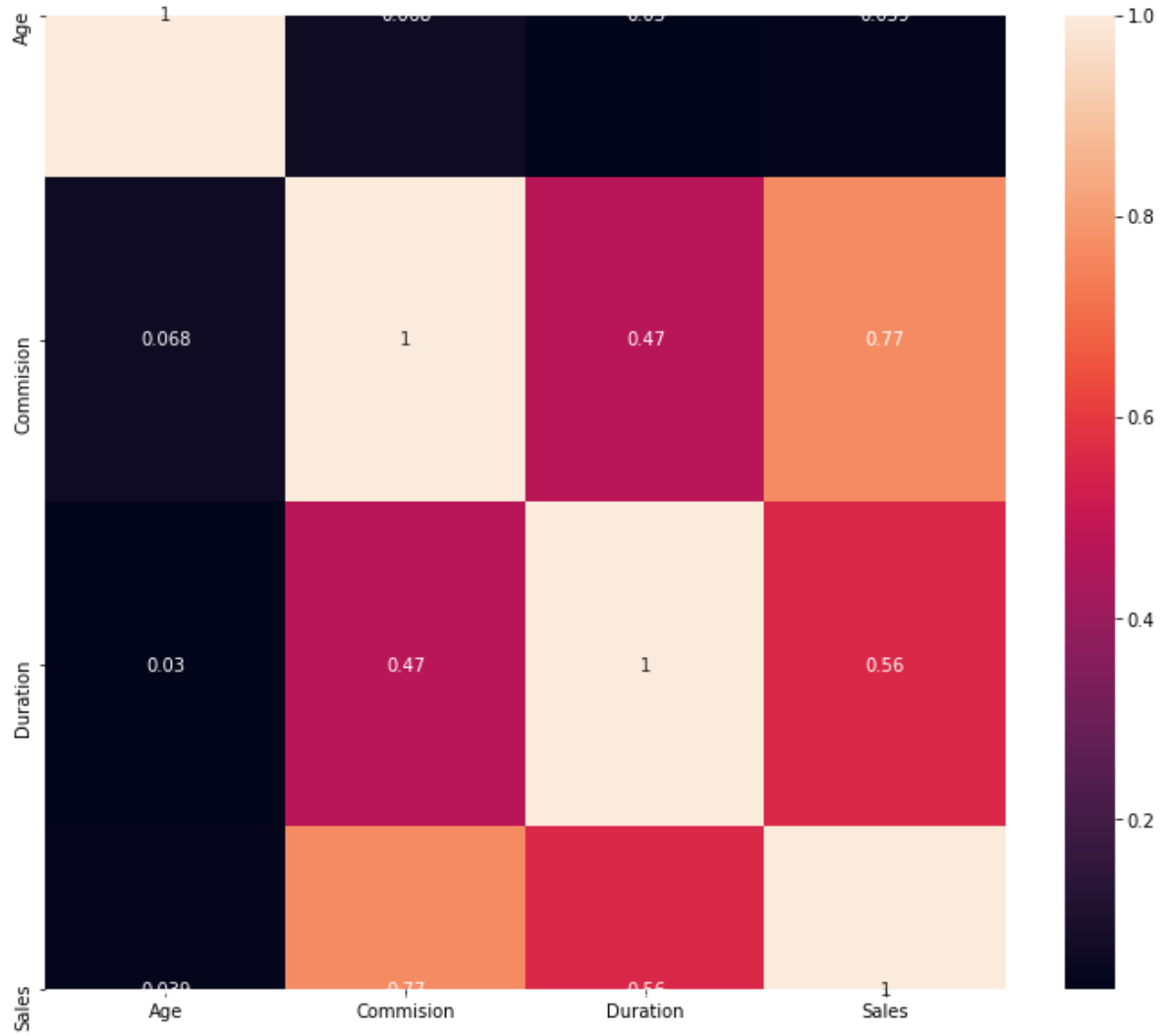
Checking the heatmap:

In [55]:

```
plt.subplots(figsize=(12,10))
sns.heatmap(insur.corr(), annot=True)
```

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e70e3516c8>
```



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

EDA of Insurance DataSet

- There are 3000 rows and 10 columns in the Insurance DataSet
- There are 158 duplicates in the dataset
- There are no null values in the dataset
- There are outliers in some variables in Age, Commision,Duration,Sales.
- Data is not normally distributed in the variables in Commision,Duration,Sales etc.
- Some Variables in the dataset are inter related like Sales and Commision ,Sales and Duration.

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

Removing the unwanted Column for preparing the Model

In [56]:

```
insur=insur.drop(['Agency_Code', 'Product Name'],axis=1)
```

Checking the data after removing the column

In [57]:

```
insur.head()
```

Out[57]:

	Age	Type	Claimed	Commision	Channel	Duration	Sales	Destination
0	48	Airlines	No	0.70	Online	7	2.51	ASIA
1	36	Travel Agency	No	0.00	Online	34	20.00	ASIA
2	39	Travel Agency	No	5.94	Online	3	9.90	Americas
3	36	Travel Agency	No	0.00	Online	4	26.00	ASIA
4	33	Airlines	No	6.30	Online	53	18.00	ASIA

Proportion of observations in Target classes

In [58]:

```
insur['Claimed'].value_counts(normalize=True)
```

Out[58]:

```
No     0.692
Yes    0.308
Name: Claimed, dtype: float64
```

Removing the duplicat values

In [59]:

```
insur=insur.drop_duplicates()
```

Crosscheck All the duplicates has been removed properly

In [60]:

```
dupes=insur.duplicated()
sum(dupes)
```

Out[60]:

```
0
```

Dimension of the data after removing the duplicates

In [61]:

```
insur.shape
```

Out[61]:

```
(2842, 8)
```

Datatypes of final dataset

In [62]:

```
insur.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2842 entries, 0 to 2999
Data columns (total 8 columns):
Age             2842 non-null int64
Type            2842 non-null object
Claimed         2842 non-null object
Commision       2842 non-null float64
Channel         2842 non-null object
Duration        2842 non-null int64
Sales           2842 non-null float64
Destination     2842 non-null object
dtypes: float64(2), int64(2), object(4)
memory usage: 199.8+ KB
```

Dealing with outliers

In [63]:

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

In [64]:

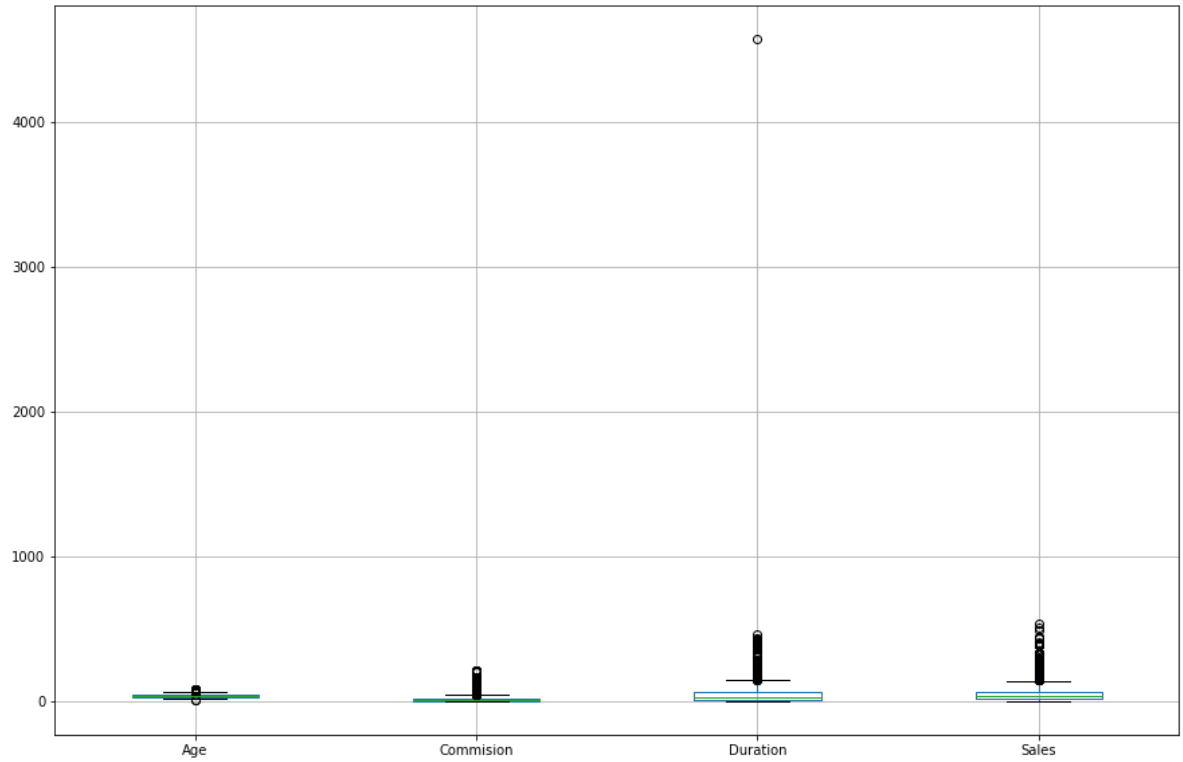
```
#for feature in insur[['Age','Commision','Duration','Sales']]:
   # lr,ur=treat_outlier(insur[feature])
   # insur[feature]=np.where(insur[feature]>ur,ur,insur[feature])
   # insur[feature]=np.where(insur[feature]<lr,lr,insur[feature])
```

In [65]:

```
plt.figure(figsize=(15,10))
insur[['Age','Commision','Duration','Sales']].boxplot()
```

Out[65]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e70bf18d48>
```



Converting the Object type variables into Categorical variables

In [66]:

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

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

Checking the data after Converting the Object type variables into Categorical variables

In [67]:

```
insur.head(10)
```

Out[67]:

	Age	Type	Claimed	Commision	Channel	Duration	Sales	Destination
0	48	0	0	0.70	1	7	2.51	0
1	36	1	0	0.00	1	34	20.00	0
2	39	1	0	5.94	1	3	9.90	1
3	36	1	0	0.00	1	4	26.00	0
4	33	0	0	6.30	1	53	18.00	0
5	45	0	1	15.75	1	8	45.00	0
6	61	1	0	35.64	1	30	59.40	1
7	36	1	0	0.00	1	16	80.00	0
8	36	1	0	0.00	1	19	14.00	0
9	36	1	0	0.00	1	42	43.00	0

In [68]:

```
insur['Claimed'].value_counts(normalize=True)
```

Out[68]:

```
0    0.678395
1    0.321605
Name: Claimed, dtype: float64
```

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

In [69]:

```
X = insur.drop(['Claimed'], axis=1)
y =insur.pop('Claimed')
X.head()
```

Out[69]:

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

Splitting data into training and test set

In [70]:

```
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 [71]:

```
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 (1989, 7)
X_test (853, 7)
train_labels (1989,)
test_labels (853,)
```

Building a Decision Tree Classifier

In [72]:

```
param_grid = {
    'criterion': ['gini'],
    'max_depth': [25,50,100],
    'min_samples_leaf': [25,50,100],
    'min_samples_split': [200,250,300,400],
}
dtcl = DecisionTreeClassifier()
grid_search = GridSearchCV(estimator = dtcl, param_grid = param_grid, cv = 10)
```

Fitting the Best parameters

In [73]:

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

```
{'criterion': 'gini', 'max_depth': 25, 'min_samples_leaf': 25, 'min_samples_
split': 200}
```

Out[73]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=25,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=25, min_samples_split=200,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

Generating Tree

In [74]:

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

tree_regularized.close()
dot_data
```

Variable Importance

In [75]:

```
print (pd.DataFrame(best_grid.feature_importances_, columns = ["Imp"], index = X_train.colu
```

```
                Imp
Type       0.430696
Sales      0.422326
Commision  0.098882
Duration   0.037620
Channel    0.007389
Destination  0.003088
Age        0.000000
```

Predicting on Training and Test dataset

In [76]:

```
ytrain_predict = best_grid.predict(X_train)
ytest_predict = best_grid.predict(X_test)
```

Getting the Predicted Classes and Probs

In [77]:

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

Out[77]:

	0	1
0	0.403670	0.596330
1	0.462121	0.537879
2	0.172619	0.827381
3	0.462121	0.537879
4	0.915385	0.084615

Model Evaluation(CART)

AUC and ROC for the training data(CART)

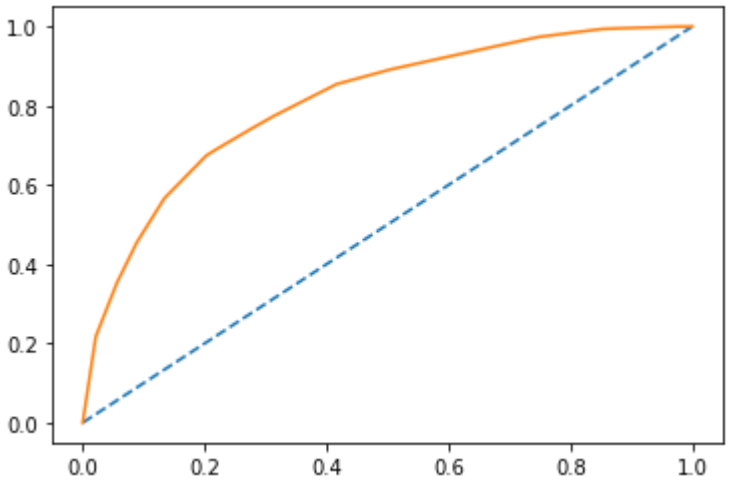
In [78]:

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

```
AUC: 0.809
```

Out[78]:

```
[<matplotlib.lines.Line2D at 0x1e70e5fa688>]
```



AUC and ROC for the test data(CART)

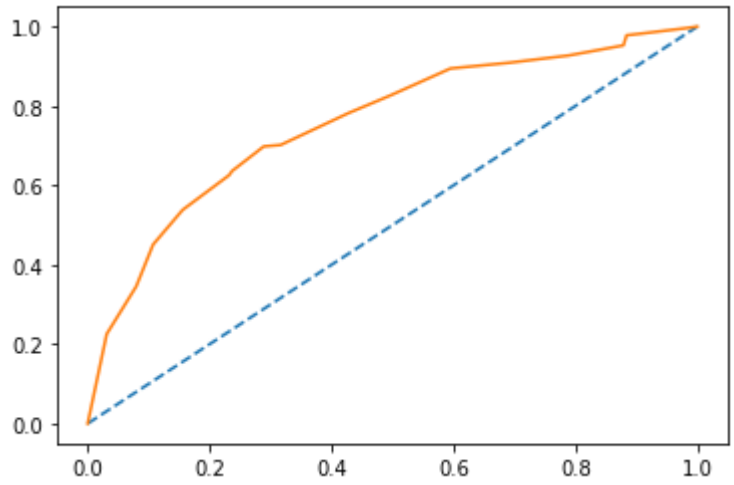
In [79]:

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

```
AUC: 0.757
```

Out[79]:

```
[<matplotlib.lines.Line2D at 0x1e70e6587c8>]
```



Confusion Matrix for the training data(CART)

In [80]:

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

Out[80]:

```
array([[1169,  181],
       [ 277,  362]], dtype=int64)
```

Train Data Accuracy(CART)

In [81]:

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

Out[81]:

```
0.7697335344394168
```

Classification Report of Training Data(CART)

In [82]:

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

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84      1350
           1       0.67      0.57      0.61       639

    accuracy                           0.77      1989
   macro avg       0.74      0.72      0.72      1989
weighted avg       0.76      0.77      0.76      1989
```

In [83]:

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

```
cart_train_precision  0.67
cart_train_recall  0.57
cart_train_f1  0.61
```

Confusion Matrix for Test data(CART)

In [84]:

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

Out[84]:

```
array([[488,  90],
       [127, 148]], dtype=int64)
```

Test Data Accuracy(CART)

In [85]:

```
cart_test_acc=best_grid.score(X_test,test_labels)
cart_test_acc
```

Out[85]:

```
0.7456037514654161
```

Classification Report of Test Data(CART)

In [86]:

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

```
              precision    recall  f1-score   support

           0       0.79      0.84      0.82       578
           1       0.62      0.54      0.58       275

    accuracy                           0.75       853
   macro avg       0.71      0.69      0.70       853
weighted avg       0.74      0.75      0.74       853
```

CART MODEL METRICS

In [87]:

```
cart_metrics=classification_report(test_labels, ytest_predict,output_dict=True)
df=pd.DataFrame(cart_metrics).transpose()
df
cart_test_precision=round(df.loc["1"][0],2)
cart_test_recall=round(df.loc["1"][1],2)
cart_test_f1=round(df.loc["1"][2],2)
print ('cart_train_precision ',cart_test_precision)
print ('cart_train_recall ',cart_test_recall)
print ('cart_train_f1 ',cart_test_f1)
```

```
cart_train_precision  0.62
cart_train_recall  0.54
cart_train_f1  0.58
```

Cart model Conclusion

In [88]:

```
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
data = pd.DataFrame({'CART Train':[cart_train_acc,cart_train_auc,cart_train_recall,cart_tra
        'CART Test':[cart_test_acc,cart_test_auc,cart_test_recall,cart_test_precision,cart_
round(data,2)
```

Out[88]:

	CART Train	CART Test
Accuracy	0.77	0.75
AUC	0.81	0.76
Recall	0.57	0.54
Precision	0.67	0.62
F1 Score	0.61	0.58

Cart Model Comments

- Training and Test set results are almost similar, and with the overall measures high, the model is a good model.
- Type, Sales and Commision are key variables to predict the Insurance

Building the Random Forest Model

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

In [89]:

```
param_grid = {
    'max_depth': [25,50,60],
    'max_features': [2,4,6],
    'min_samples_leaf': [100,150],
    'min_samples_split': [250,300,400],
    'n_estimators': [25,50,75]
}

rfcl = RandomForestClassifier()

grid_search = GridSearchCV(estimator = rfcl, param_grid = param_grid, cv = 5)
```

In [90]:

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

Out[90]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=N
one,
                                              criterion='gini', max_depth=No
ne,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=No
ne,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [25, 50, 60], 'max_features': [2, 4,
6],
                         'min_samples_leaf': [100, 150],
                         'min_samples_split': [250, 300, 400],
                         'n_estimators': [25, 50, 75]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [91]:

```
grid_search.best_params_
```

Out[91]:

```
{'max_depth': 50,
 'max_features': 4,
 'min_samples_leaf': 100,
 'min_samples_split': 300,
 'n_estimators': 75}
```

In [92]:

```
best_grid = grid_search.best_estimator_
```

In [93]:

```
best_grid
```

Out[93]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=50, max_features=4, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=100, min_samples_split=300,
                       min_weight_fraction_leaf=0.0, n_estimators=75,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Predicting the Training and Testing data(RF Model)

In [94]:

```
ytrain_predict = best_grid.predict(X_train)
ytest_predict = best_grid.predict(X_test)
```

Performance Evaluation on Training data(RF Model)

Confusion Matrix of Training Data(RF Model)

In [95]:

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

Out[95]:

```
array([[1264,   86],
       [ 398,  241]], dtype=int64)
```

Accuracy Score of Training Data(RF Model)

In [96]:

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

Out[96]:

```
0.7566616390145802
```

Classification Report of Training Data(RF Model)

In [97]:

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

```
              precision    recall  f1-score   support

           0       0.76      0.94      0.84      1350
           1       0.74      0.38      0.50       639

    accuracy                           0.76      1989
   macro avg       0.75      0.66      0.67      1989
weighted avg       0.75      0.76      0.73      1989
```

In [98]:

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

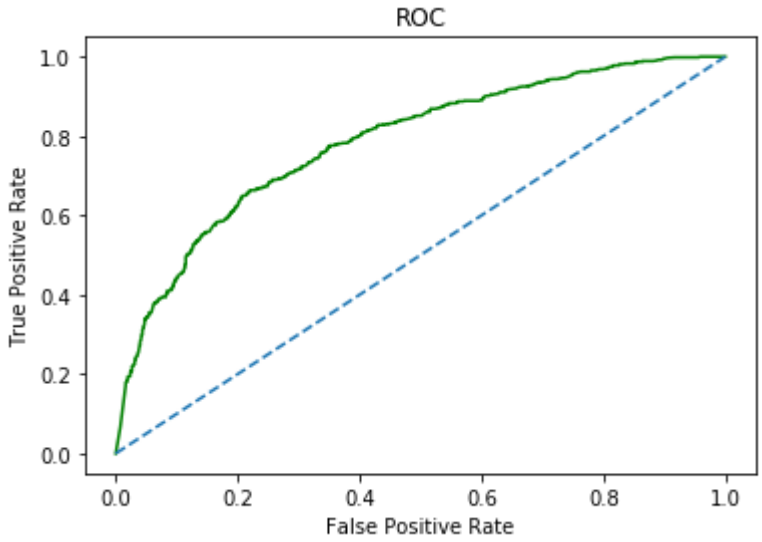
```
rf_train_precision  0.38
rf_train_recall  0.5
rf_train_f1  0.74
```

AUC and ROC of Training Data(RF MODEL)

In [99]:

```
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(X_train))
print('Area under Curve is', rf_train_auc)
```

```
Area under Curve is 0.6567240479916536
```



Performance Evaluation on Test data(RF Model)

Confusion Matrix Test Data(RF Model)

In [100]:

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

Out[100]:

```
array([[526,  52],
       [176,  99]], dtype=int64)
```

Test Data Accuracy(RF Model)

In [101]:

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

Out[101]:

```
0.7327080890973037
```

Classification Report of Test Data(RF Model)

In [102]:

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

```
              precision    recall  f1-score   support

           0       0.75      0.91      0.82       578
           1       0.66      0.36      0.46       275

    accuracy                           0.73       853
   macro avg       0.70      0.64      0.64       853
weighted avg       0.72      0.73      0.71       853
```

In [103]:

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

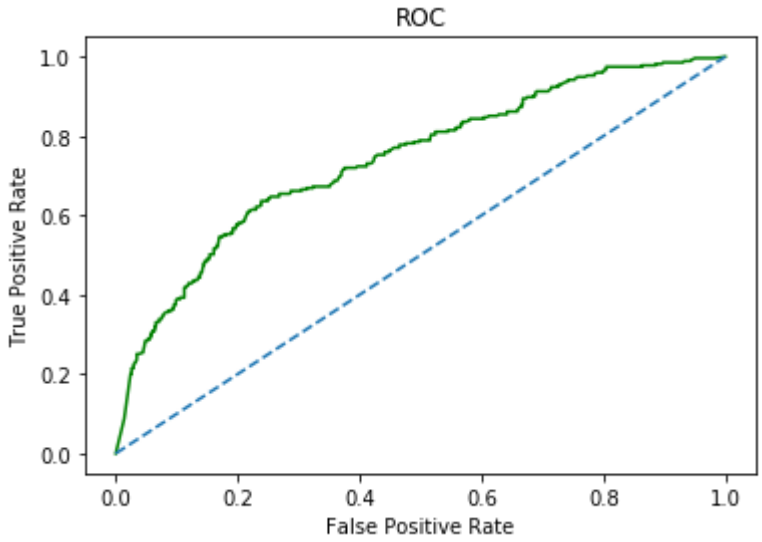
```
rf_test_precision  0.36
rf_test_recall  0.46
rf_test_f1  0.66
```

AUC and ROC for the Test data(RF MODEL)

In [104]:

```
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(X_test))
print('Area under Curve is', rf_test_auc)
```

```
Area under Curve is 0.6350173010380622
```



Variable Importance(RF Model)

In [105]:

```
print (pd.DataFrame(best_grid.feature_importances_, columns = ["Imp"], index = X_train.colu
```

```
                Imp
Type       0.338684
Sales      0.291961
Commision  0.274631
Duration   0.076046
Age        0.011198
Destination  0.007481
Channel    0.000000
```

RF Model Conclusion

In [106]:

```
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
data = pd.DataFrame({'Random Forest Train':[rf_train_acc,rf_train_auc,rf_train_recall,rf_tr
                     'Random Forest Test':[rf_test_acc,rf_test_auc,rf_test_recall,rf_test_p
round(data,2)
```

Out[106]:

	Random Forest Train	Random Forest Test
Accuracy	0.76	0.73
AUC	0.66	0.64
Recall	0.50	0.46
Precision	0.38	0.36
F1 Score	0.74	0.66

RF Model Comments

- Training and Test set results are almost similar, and with the overall Moderate results.
- Accuracy Score is good But AUC Matrics is Moderate
- Recall Matrics is Average while Precision is slightly lower side
- Type, Sales and Commision are key variables to predict the Insurance

Building the Neural Network Model

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

In [107]:

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'hidden_layer_sizes': [100,200,300,500],
    'max_iter': [5000,2500,7000,6000],
    'solver': ['sgd','adam'],
    'tol': [0.01],
}

nncl = MLPClassifier()

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

In [108]:

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

Out[108]:

```
{'hidden_layer_sizes': 200, 'max_iter': 5000, 'solver': 'sgd', 'tol': 0.01}
```

In [109]:

```
grid_search.best_params_
```

Out[109]:

```
{'hidden_layer_sizes': 200, 'max_iter': 5000, 'solver': 'sgd', 'tol': 0.01}
```

Further Optimization

In [110]:

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'hidden_layer_sizes': [500,600,700],
    'max_iter': [6000],
    'solver': ['adam'],
    'tol': [0.01],
}

nncl = MLPClassifier()

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

In [111]:

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

Out[111]:

```
{'hidden_layer_sizes': 500, 'max_iter': 6000, 'solver': 'adam', 'tol': 0.01}
```

In [112]:

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

Out[112]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.
9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=500, learning_rate='constant',
              learning_rate_init=0.001, max_iter=6000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.01,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

Predicting the Training and Testing data(ANN Model)

In [113]:

```
ytrain_predict = best_grid.predict(X_train)
ytest_predict = best_grid.predict(X_test)
```

Performance Evaluation on Training data(ANN Model)

Confusion Matrix of Train Data(ANN Model)

In [114]:

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

Out[114]:

```
array([[1310,   40],
       [ 457,  182]], dtype=int64)
```

Accuracy Score of Training data(ANN Model)

In [115]:

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

Out[115]:

```
0.7501256913021619
```

Classification Report of Training Data(ANN Model)

In [116]:

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

```
              precision    recall  f1-score   support

           0       0.74      0.97      0.84      1350
           1       0.82      0.28      0.42       639

    accuracy                           0.75      1989
   macro avg       0.78      0.63      0.63      1989
weighted avg       0.77      0.75      0.71      1989
```

In [117]:

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

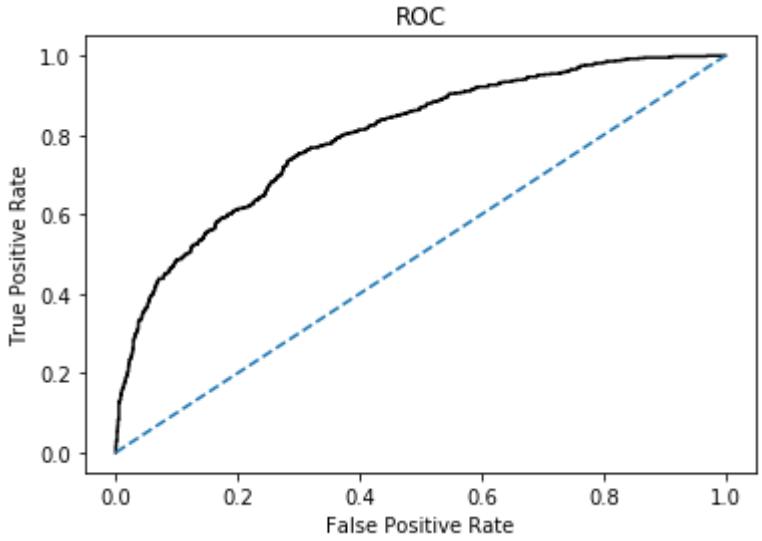
```
nn_train_precision  0.28
nn_train_recall  0.42
nn_train_f1  0.82
```

AUC and ROC for the training data(ANN Model)

In [118]:

```
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(X_train))
print('Area under Curve is', nn_train_auc)
```

```
Area under Curve is 0.6275952008346375
```



Model Evaluation on Test data (ANN Model)

Confusion of Test data(ANN Model)

In [119]:

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

Out[119]:

```
array([[554,  24],
       [195,  80]], dtype=int64)
```

Accuracy Score of Test data(ANN Model)

In [120]:

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

Out[120]:

```
0.7432590855803048
```

Classification Report of Test Data(ANN Model)

In [121]:

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

```
              precision    recall  f1-score   support

           0       0.74      0.96      0.83       578
           1       0.77      0.29      0.42       275

    accuracy                           0.74       853
   macro avg       0.75      0.62      0.63       853
weighted avg       0.75      0.74      0.70       853
```

In [122]:

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

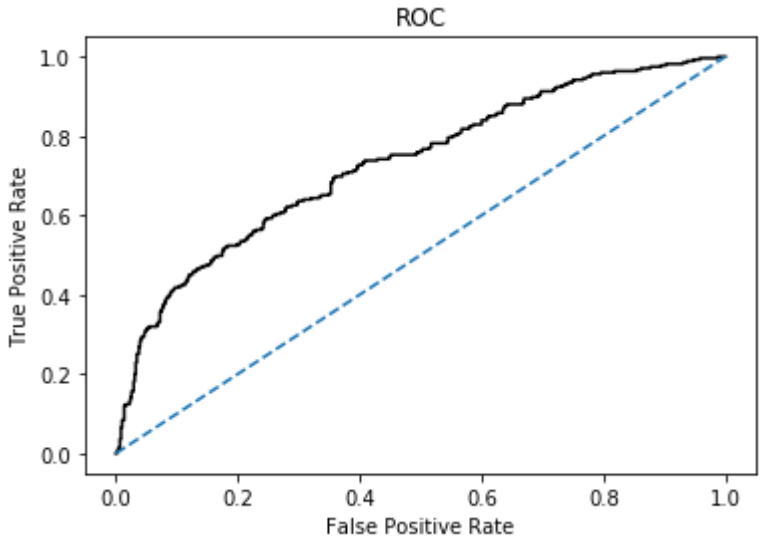
```
nn_test_precision  0.29
nn_test_recall  0.42
nn_test_f1  0.77
```

AUC and ROC for the Test data(ANN Model)

In [123]:

```
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(X_test))
print('Area under Curve is', nn_test_auc)
```

```
Area under Curve is 0.624693299779805
```



In [124]:

```
best_grid.score
```

Out[124]:

```
<bound method ClassifierMixin.score of MLPClassifier(activation='relu', alph
a=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=500, learning_rate='constant',
              learning_rate_init=0.001, max_iter=6000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.01,
              validation_fraction=0.1, verbose=False, warm_start=False)>
```

ANN Model Conclusion

In [125]:

```
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
data = pd.DataFrame({'Neural Network Train':[nn_train_acc,nn_train_auc,nn_train_recall,nn_t
                     'Neural Network Test':[nn_test_acc,nn_test_auc,nn_test_recall,nn_test_
round(data,2)
```

Out[125]:

	Neural Network Train	Neural Network Test
Accuracy	0.75	0.74
AUC	0.63	0.62
Recall	0.42	0.42
Precision	0.28	0.29
F1 Score	0.82	0.77

ANN Model Comments

- Training and Test set results are almost similar, and with the overall Moderate results.
- Accuracy Score is good But AUC Matrics is Moderate
- Recall Matrics is Average while Precision is slightly lower side
- Type, Sales and Commision are key variables to predict the Insurance

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

In [126]:

```
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
data = pd.DataFrame({'CART Train':[cart_train_acc,cart_train_auc,cart_train_recall,cart_tra
        'CART Test':[cart_test_acc,cart_test_auc,cart_test_recall,cart_test_precision,cart_
       'Random Forest Train':[rf_train_acc,rf_train_auc,rf_train_recall,rf_train_precision,
        'Random Forest Test':[rf_test_acc,rf_test_auc,rf_test_recall,rf_test_precision,rf_t
       'Neural Network Train':[nn_train_acc,nn_train_auc,nn_train_recall,nn_train_precisior
        'Neural Network Test':[nn_test_acc,nn_test_auc,nn_test_recall,nn_test_precision,nn_
round(data,2)
```

Out[126]:

	CART Train	CART Test	Random Forest Train	Random Forest Test	Neural Network Train	Neural Network Test
Accuracy	0.77	0.75	0.76	0.73	0.75	0.74
AUC	0.81	0.76	0.66	0.64	0.63	0.62
Recall	0.57	0.54	0.50	0.46	0.42	0.42
Precision	0.67	0.62	0.38	0.36	0.28	0.29
F1 Score	0.61	0.58	0.74	0.66	0.82	0.77

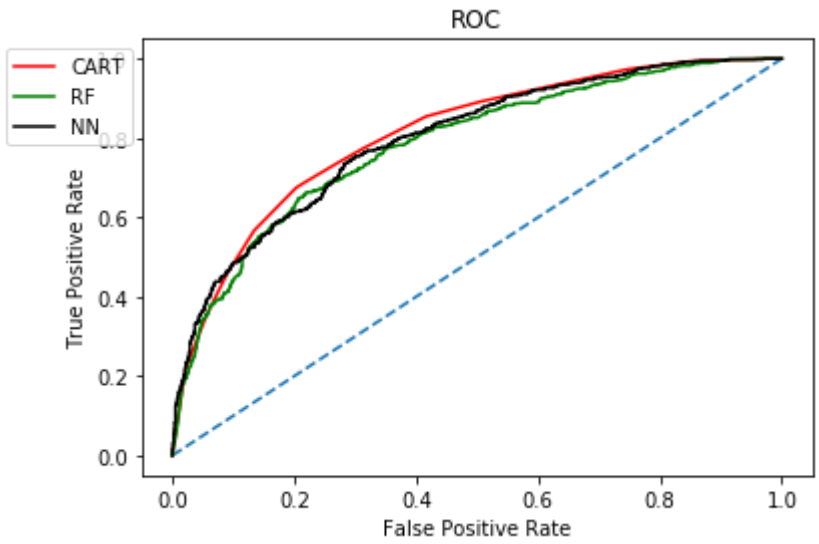
ROC Curve for the 3 models on the Training data

In [127]:

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(cart_train_fpr, cart_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.), loc='best')
```

Out[127]:

```
<matplotlib.legend.Legend at 0x1e70b6f0408>
```



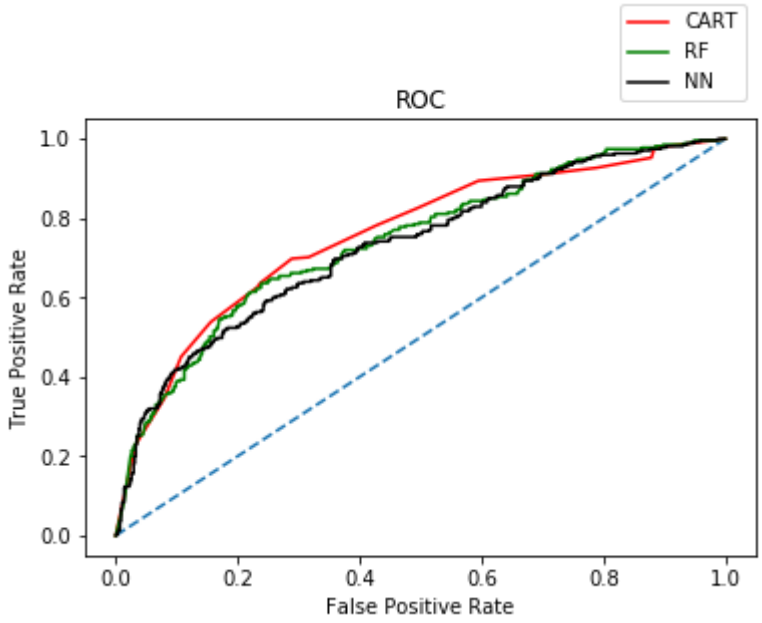
ROC Curve for the 3 models on the Test data

In [128]:

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(cart_test_fpr, cart_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[128]:

```
<matplotlib.legend.Legend at 0x1e70b2cbb48>
```



Cart,RF,ANN Model Comments

- There is accuracy is same in all three Models(Cart,RF,ANN)
- In the Cart Model Training and Test set results are almost similar, and with the overall measures high, the model is a good model.
- As Per the Cart Model, Type, Sales and Commision are key variables to predict the Insurance
- In the RF and ANN Model, Training and Test set results are almost similar, and with the overall Moderate results.
- In the RF and ANN Model, Accuracy Score is good But AUC Matrics is Moderate
- In the RF and ANN Model Recall Matrics is Average while Precision is slightly lower side
- As per the RF ModelType, Sales and Commision are key variables to predict the Insurance

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

In [129]:

```
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
data = pd.DataFrame({'CART Train':[cart_train_acc,cart_train_auc,cart_train_recall,cart_tra
        'CART Test':[cart_test_acc,cart_test_auc,cart_test_recall,cart_test_precision,cart_
       'Random Forest Train':[rf_train_acc,rf_train_auc,rf_train_recall,rf_train_precision,
        'Random Forest Test':[rf_test_acc,rf_test_auc,rf_test_recall,rf_test_precision,rf_t
       'Neural Network Train':[nn_train_acc,nn_train_auc,nn_train_recall,nn_train_precisior
        'Neural Network Test':[nn_test_acc,nn_test_auc,nn_test_recall,nn_test_precision,nn_
round(data,2)
```

Out[129]:

	CART Train	CART Test	Random Forest Train	Random Forest Test	Neural Network Train	Neural Network Test
Accuracy	0.77	0.75	0.76	0.73	0.75	0.74
AUC	0.81	0.76	0.66	0.64	0.63	0.62
Recall	0.57	0.54	0.50	0.46	0.42	0.42
Precision	0.67	0.62	0.38	0.36	0.28	0.29
F1 Score	0.61	0.58	0.74	0.66	0.82	0.77

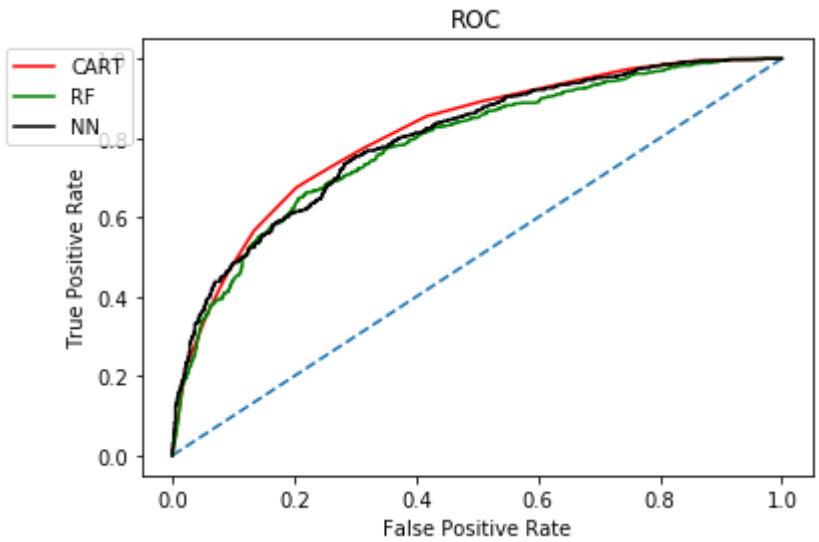
ROC Curve for the 3 models on the Training data

In [130]:

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(cart_train_fpr, cart_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.), loc='best')
```

Out[130]:

```
<matplotlib.legend.Legend at 0x1e70e6fab48>
```



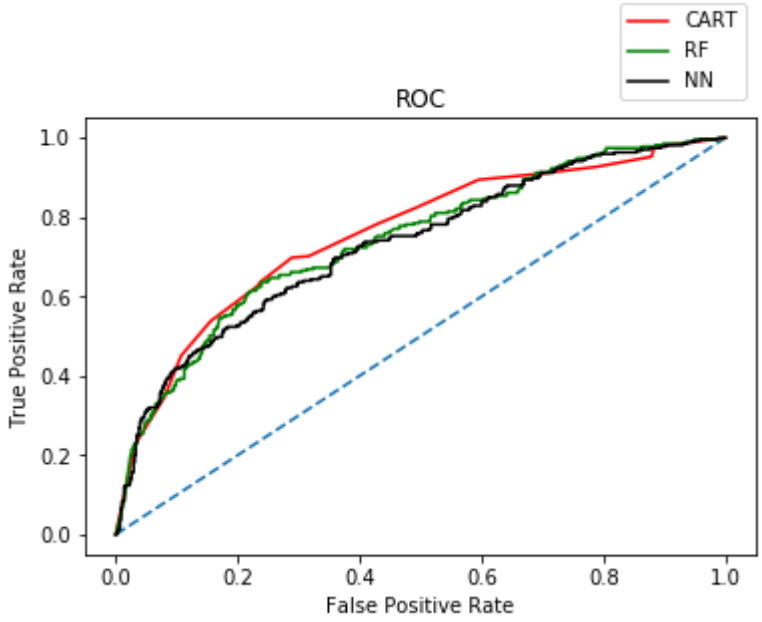
ROC Curve for the 3 models on the Test data

In [131]:

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(cart_test_fpr, cart_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[131]:

```
<matplotlib.legend.Legend at 0x1e70b66b688>
```



Cart,RF,ANN Model Comparison

- There is accuracy is same in all three Models(Cart,RF,ANN)
- AUC Score is better in Cart Model as compare to the RF and ANN.
- Recall and Precision is better in Cart Model as compare to the RF and ANN.
- If We compare the F1 score (take Recall and Precision into account) RF and ANN Model having edge over Cart Model.

2.5 Inference: Basis on these predictions, what are the business insights and recommendations

Business Insights and Recommendations.

- There are 2 variables Type(Airlines,Traval Agency) and Sales play the the key role as deciding Factor
- if Person is travellng through air and Ticket Price(Sales) is greater than 53 then 74 % probabliity is that they will claim the insurance
- if Person is travellng through air and Ticket Price(Sales) is less than 4.5 then 65% people will not claim the amount

- 83 % people will not claim insurace,If Ticket is booked through traval agency and (Ticket Price) Sales is less thant 83.5

In [ ]: