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.

2.1 Data Ingestion: Read the dataset. Do the descriptive statistics and do null value condition check, write an inference on it. 2.2 Data Split: Split the data into test and train, build classification model CART, Random Forest, Artificial Neural Network 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 2.4 Final Model: Compare all the model and write an inference which model is best/optimized. 2.5 Inference: Based on the whole Analysis, what are the business insights and recommendations

Dataset for Problem 2: insurance_part2_data-1.csv

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 [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
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_matri
x
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
```

Loading the data

In [2]:

```
dfi = pd.read_csv("insurance_part2_data.csv")
```

In [3]:

```
dfi.head()
```

Out[3]:

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	Destination

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

In [4]:

```
dfi.shape
```

Out[4]:

```
(3000, 10)
```

In [5]:

```
dfi.info()
```

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

In [6]:

```
dfi.describe()
```

Out[6]:

	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

In [7]:

```
dfi.isnull().sum()
```

Out[7]:

```
Age             0
Agency_Code     0
Type            0
Claimed         0
Commision       0
Channel         0
```

```
Duration        0
Sales           0
Product Name    0
Destination     0
dtype: int64
```

In [8]:

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

Out[8]:

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	Destinati
count	3000.000000	3000	3000	3000	3000.000000	3000	3000.000000	3000.000000	3000	30
unique	NaN	4	2	2	NaN	2	NaN	NaN	5	
top	NaN	EPX	Travel Agency	No	NaN	Online	NaN	NaN	Customised Plan	AS
freq	NaN	1365	1837	2076	NaN	2954	NaN	NaN	1136	24
mean	38.091000	NaN	NaN	NaN	14.529203	NaN	70.001333	60.249913	NaN	Na
std	10.463518	NaN	NaN	NaN	25.481455	NaN	134.053313	70.733954	NaN	Na
min	8.000000	NaN	NaN	NaN	0.000000	NaN	-1.000000	0.000000	NaN	Na
25%	32.000000	NaN	NaN	NaN	0.000000	NaN	11.000000	20.000000	NaN	Na
50%	36.000000	NaN	NaN	NaN	4.630000	NaN	26.500000	33.000000	NaN	Na
75%	42.000000	NaN	NaN	NaN	17.235000	NaN	63.000000	69.000000	NaN	Na
max	84.000000	NaN	NaN	NaN	210.210000	NaN	4580.000000	539.000000	NaN	Na

Getting unique counts of all nominal variables

In [9]:

```
dfi.columns
```

Out[9]:

```
Index(['Age', 'Agency_Code', 'Type', 'Claimed', 'Commision', 'Channel',
       'Duration', 'Sales', 'Product Name', 'Destination'],
      dtype='object')
```

In [10]:

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

```
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


CHANNEL :  2
Offline      46
Online     2954
Name: Channel, 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
```

In [11]:

```
dups=dfi.duplicated()
dups
dups.sum()
```

Out[11]:

```
139
```

In [12]:

```
dfi.drop_duplicates(inplace=True)
```

In [13]:

```
dups=dfi.duplicated()
dups
dups.sum()
```

Out[13]:

```
0
```

In [14]:

```
dfi.shape
```

Out[14]:

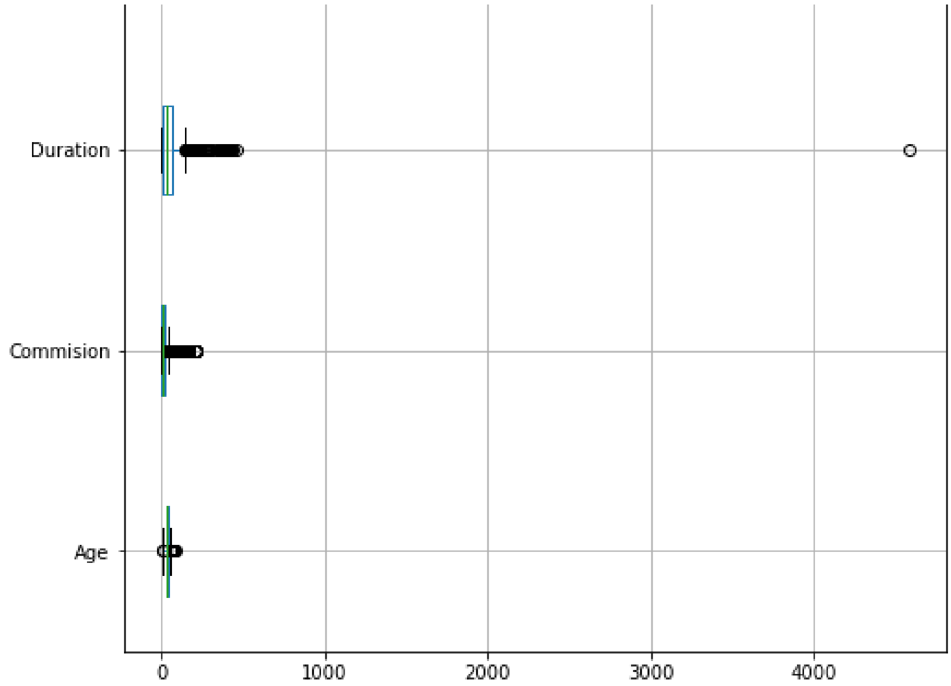
```
(2861, 10)
```

In [15]:

```
plt.figure(figsize=(8,8))
dfi[['Age', 'Commision', 'Duration', 'Sales']].boxplot(vert=0)
```

Out[15]:

```
<AxesSubplot:>
```

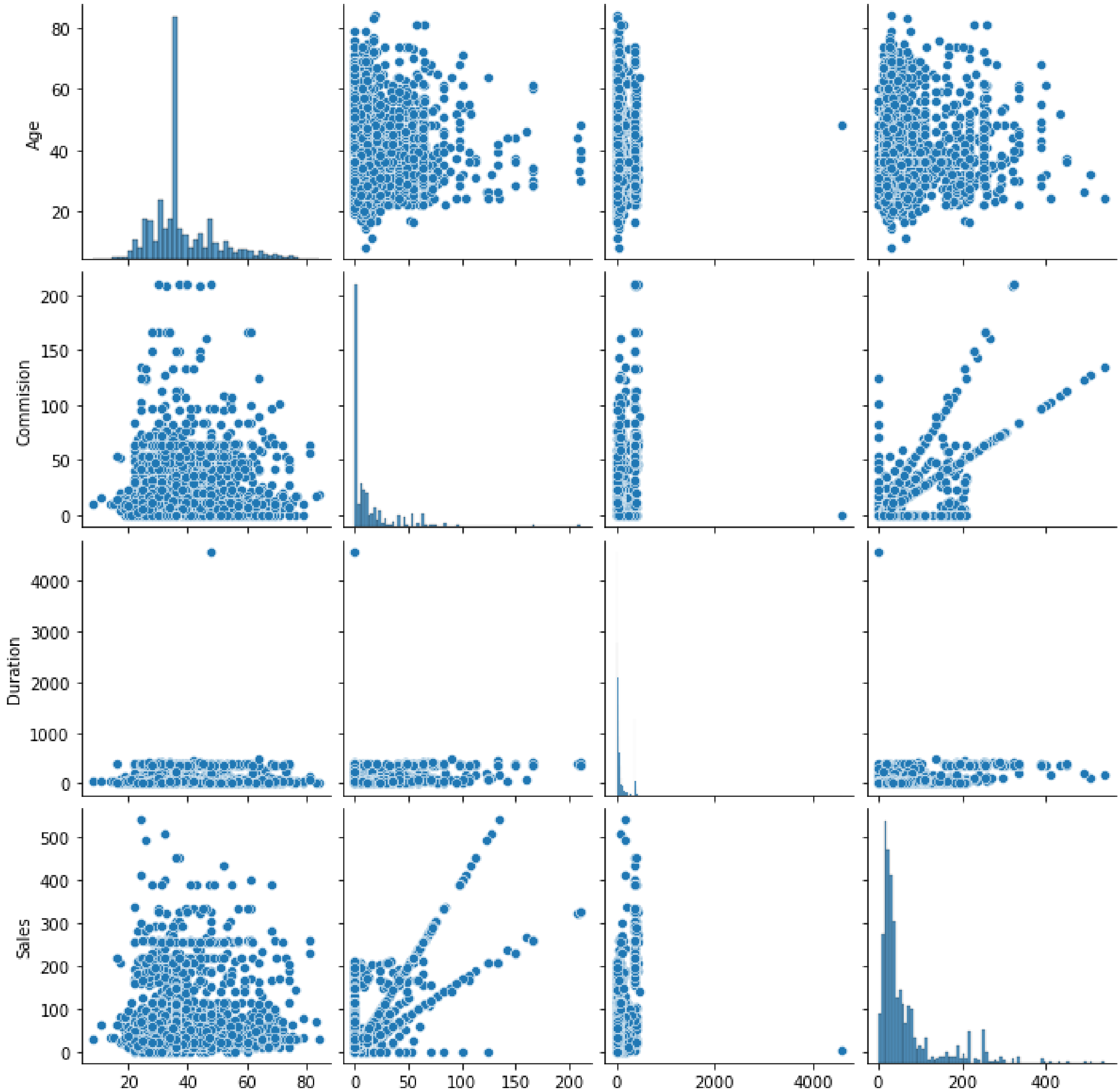


In [16]:

```
sns.pairplot(dfi)
```

Out[16]:

<seaborn.axisgrid.PairGrid at 0x1b041442c40>



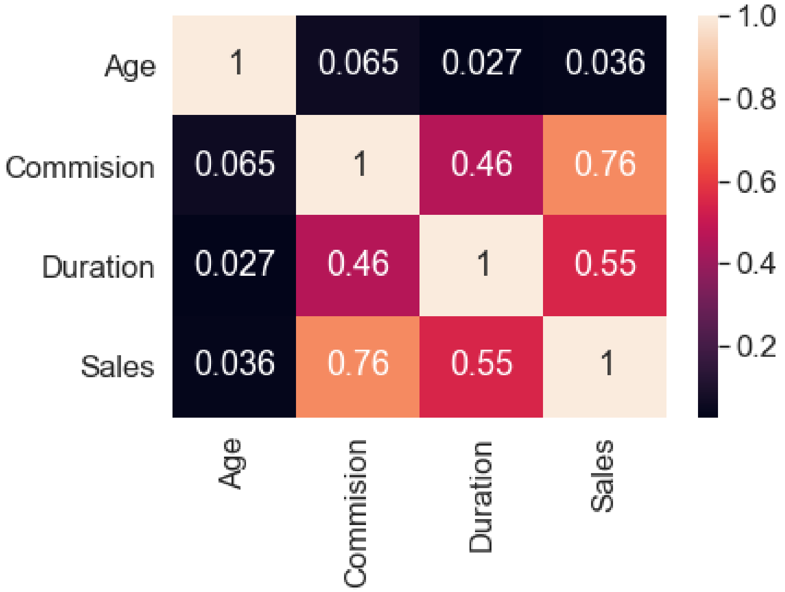
Age Commision Duration Sales

In [17]:

```
plt.figure(figsize=(6,4))
sns.set(font_scale=1.5)
sns.heatmap((dfi).corr(), annot=True)
```

Out[17]:

```
<AxesSubplot:>
```



In [18]:

```
for feature in dfi.columns:
    if dfi[feature].dtype == 'object':
        print('\n')
        print('feature:',feature)
        print(pd.Categorical(dfi[feature].unique()))
        print(pd.Categorical(dfi[feature].unique()).codes)
        dfi[feature] = pd.Categorical(dfi[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', 'Gold Pla
n', 'Silver Plan']
[2 1 0 4 3]
```

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

In [19]:

```
dfi.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: 128.5 KB
```

In [20]:

```
dfi.head()
```

Out[20]:

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

proportion of 1s and 0s

In [21]:

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

Out[21]:

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

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

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

In [22]:

```
X=dfi.drop("Claimed", axis=1)
Y=dfi.pop("Claimed")
X.head()
```

Out[22]:

	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 test and train

In [23]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, train_labels, test_labels = train_test_split(X, Y, test_size=.30, rando
m_state=1)
```

Checking the dimensions of the test and train data

In [24]:

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

```
X_test (859, 9)
X_train (2002, 9)
test_labels (859,)
train_labels (2002,)
```

Building a Decision Tree Classifier

In [25]:

```
param_grid = {
    'criterion': ['gini'],
    'max_depth': [5,6,7,8,9,10],
    'min_samples_leaf': [25,50,75,100,125,150],
    'min_samples_split': [75,150,225,300,375,450],
}

dtcl = DecisionTreeClassifier(random_state=1)

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

In [26]:

```
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': 50, 'min_samples_split': 450}
```

```
{'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 50, 'min_samples_split': 300}
```

Out[26]:

```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=50, min_samples_split=300,
                       random_state=1)
```

Generating Tree

In [27]:

```
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 = l
ist(X_train), class_names = list(train_char_label))

tree_regularized.close()
dot_data
```

http://webgraphviz.com/

Variable Importance

In [28]:

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

```
                   Imp
Agency_Code   0.601720
Sales         0.305611
Product Name  0.047457
Duration      0.016689
Commision     0.014763
Age           0.013760
Type          0.000000
Channel       0.000000
Destination   0.000000
```

Predicting on Training and Test dataset

In [29]:

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

Getting the Predicted Classes and Probs

In [30]:

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

Out[30]:

	0	1
0	0.573171	0.426829
1	0.971223	0.028777
2	0.232975	0.767025
3	0.837500	0.162500
4	0.837500	0.162500

Model Evaluation

AUC and ROC for the training data

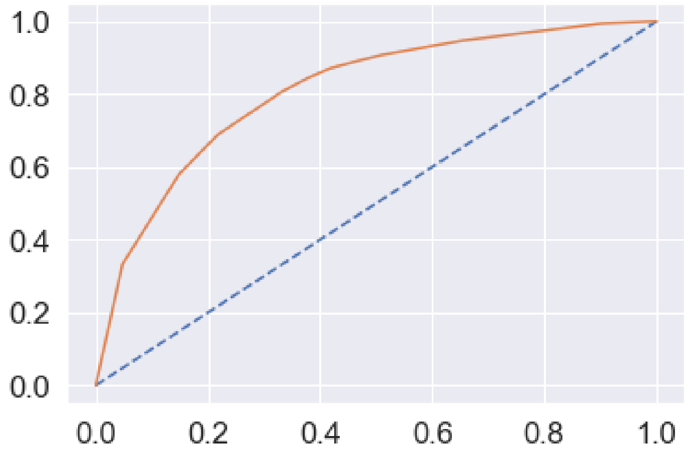
In [31]:

```
# 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[31]:

```
[<matplotlib.lines.Line2D at 0x1b0438bfeb0>]
```



AUC and ROC for the test data

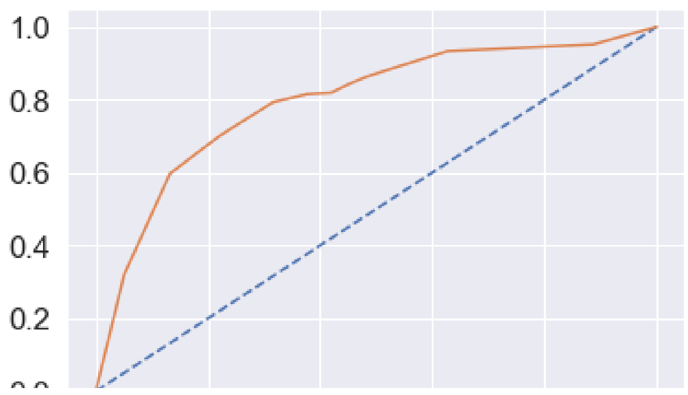
In [32]:

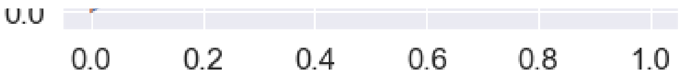
```
# predict probabilities
probs = best_grid.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
cart_test_auc = roc_auc_score(test_labels, probs)
print('AUC: %.3f' % cart_test_auc)
# calculate roc curve
cart_test_fpr, cart_test_tpr, cart_testthresholds = 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: 0.796
```

Out[32]:

```
[<matplotlib.lines.Line2D at 0x1b043a2d610>]
```





Confusion Matrix for the training data

In [33]:

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

Out[33]:

```
array([[1157,  202],
       [ 270,  373]], dtype=int64)
```

In [34]:

```
#Train Data Accuracy
cart_train_acc=best_grid.score(X_train,train_labels)
cart_train_acc
```

Out[34]:

```
0.7642357642357642
```

In [35]:

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

```
              precision    recall  f1-score   support

           0       0.81      0.85      0.83      1359
           1       0.65      0.58      0.61       643

    accuracy                           0.76      2002
   macro avg       0.73      0.72      0.72      2002
weighted avg       0.76      0.76      0.76      2002
```

In [36]:

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

```
cart_train_precision  0.65
cart_train_recall  0.58
cart_train_f1  0.61
```

Confusion Matrix for test data

In [37]:

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

Out[37]:

```
array([[510,  78],
       [109, 162]], dtype=int64)
```

In [38]:

```
#Test Data Accuracy
cart_test_acc=best_grid.score(X_test,test_labels)
cart_test_acc
```

Out[38]:

```
0.7823050058207218
```

In [39]:

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

```
              precision    recall  f1-score   support

           0       0.82      0.87      0.85       588
           1       0.68      0.60      0.63       271

    accuracy                           0.78       859
   macro avg       0.75      0.73      0.74       859
weighted avg       0.78      0.78      0.78       859
```

In [40]:

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

```
cart_test_precision  0.68
cart_test_recall  0.6
cart_test_f1  0.63
```

Cart Conclusion

CART CONCLUSION Train Data: AUC: 81% Accuracy: 76% Precision: 65% f1-Score: 61% recall: 58%

Test Data: AUC: 79.6% Accuracy: 78.2% Precision: 68% f1-Score: 63% recall: 60%

Training and Test set results somewhat closer, and with the overall measures high, the model is a good model.

Agency_Code is the most important variable for Claimed.

Building a Random Forest Classifier

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

In [41]:

```
param_grid = {
    'max_depth': [10],## 20,30,40
    'max_features': [6],## 7,8,9
    'min_samples_leaf': [10],## 50,100
    'min_samples_split': [50], ## 60,70
    'n_estimators': [300] ## 100,200
}

rfcl = RandomForestClassifier(random_state=1)

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

In [42]:

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

Out[42]:

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=1),
             param_grid={'max_depth': [10], 'max_features': [6],
                         'min_samples_leaf': [10], 'min_samples_split': [50],
                         'n_estimators': [300]})
```

In [43]:

```
grid_search.best_params_
```

Out[43]:

```
{'max_depth': 10,
 'max_features': 6,
 'min_samples_leaf': 10,
 'min_samples_split': 50,
 'n_estimators': 300}
```

In [44]:

```
best_grid = grid_search.best_estimator_
```

In [45]:

```
best_grid
```

Out[45]:

```
RandomForestClassifier(max_depth=10, max_features=6, min_samples_leaf=10,
                       min_samples_split=50, n_estimators=300, random_state=1)
```

Predicting the Training and Testing data

In [46]:

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

RF Model Performance Evaluation on Training data

In [47]:

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

Out[47]:

```
array([[1228,  131],
       [ 258,  385]], dtype=int64)
```

In [48]:

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

Out[48]:

```
0.8056943056943057
```

In [49]:

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

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      1359
           1       0.75      0.60      0.66       643

    accuracy                           0.81      2002
   macro avg       0.79      0.75      0.76      2002
weighted avg       0.80      0.81      0.80      2002
```

In [50]:

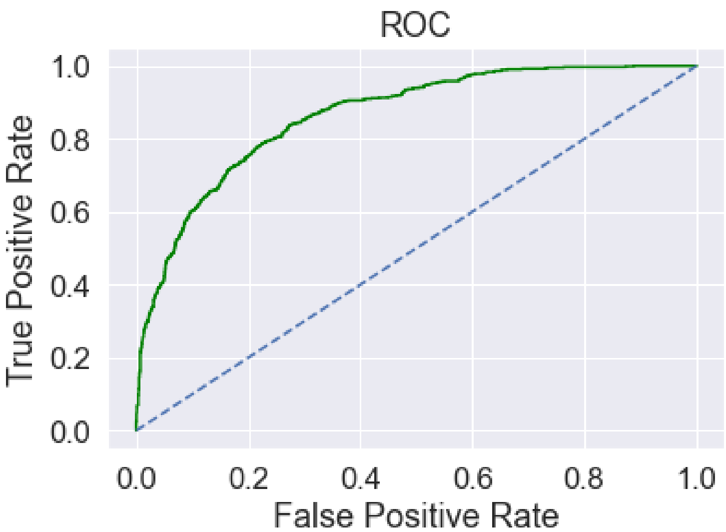
```
rf_metrics=classification_report(train_labels, ytrain_predict,output_dict=True)
df=pd.DataFrame(rf_metrics).transpose()
rf_train_precision=round(df.loc["1"][0],2)
rf_train_recall=round(df.loc["1"][1],2)
rf_train_f1=round(df.loc["1"][2],2)
print ('rf_train_precision ',rf_train_precision)
print ('rf_train_recall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

```
rf_train_precision  0.75
rf_train_recall  0.6
rf_train_f1  0.66
```

In [51]:

```
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 0.8639002468423744
```



RF Model Performance Evaluation on Test data

In [52]:

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

Out[52]:

```
array([[522,  66],
       [118, 153]], dtype=int64)
```

In [53]:

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

Out[53]:

```
0.7857974388824214
```

In [54]:

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

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       588
           1       0.70      0.56      0.62       271

    accuracy                           0.79       859
   macro avg       0.76      0.73      0.74       859
weighted avg       0.78      0.79      0.78       859
```

In [55]:

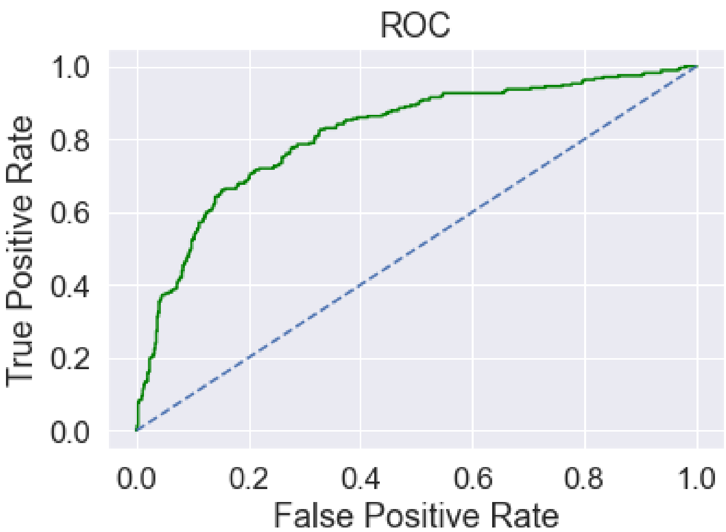
```
rf_metrics=classification_report(test_labels, ytest_predict,output_dict=True)
df=pd.DataFrame(rf_metrics).transpose()
rf_test_precision=round(df.loc["1"][0],2)
rf_test_recall=round(df.loc["1"][1],2)
rf_test_f1=round(df.loc["1"][2],2)
print ('rf_test_precision ',rf_test_precision)
print ('rf_test_recall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

```
rf_test_precision  0.7
rf_test_recall  0.56
rf_test_f1  0.62
```

In [56]:

```
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 0.8136186208800863
```



In [57]:

```
# Variable Importance
print (pd.DataFrame(best_grid.feature_importances_, columns = ["Imp"], index = X_train.c
olumns).sort_values('Imp',ascending=False))
```

```
                   Imp
Agency_Code   0.325075
Sales         0.207197
Product Name  0.169962
Duration      0.101869
Commision     0.095711
Age           0.069549
Type          0.015744
```

```
Destination     0.013474
Channel         0.001419
```

Random Forest Conclusion

RF conclusion Train Data: AUC: 86.3% Accuracy: 80.6% Precision: 75% f1-Score: 66% Recall: 60%

Test Data: AUC: 81.3% Accuracy: 78.6% Precision: 70% f1-Score: 62% Recall: 56% Training and Test set results are almost similar, and with the overall measures high, the model is a good model. Agency_Code is again the most important variable for predicting claimed

Building a Neural Network Classifier

In [58]:

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

In [59]:

```
#Initialize an object for StandardScaler
sc = StandardScaler()
```

In [60]:

```
#Scale the training data
X_train = sc.fit_transform(X_train)
```

In [61]:

```
X_train
```

Out[61]:

```
array([[ 2.88764239, -1.2626112 , -1.19813318, ..., -0.65375471,
        -1.31338076, -0.44775345],
       [-0.21666128,  0.71683095,  0.83463176, ..., -0.37032806,
         0.24339146, -0.44775345],
       [ 2.04101412, -0.27289013,  0.83463176, ...,  0.11574864,
         0.24339146,  1.24676906],
       ...,
       [-0.21666128,  0.71683095,  0.83463176, ..., -0.68209737,
        -0.53499465, -0.44775345],
       [-0.21666128,  0.71683095,  0.83463176, ...,  0.72086453,
         0.24339146, -0.44775345],
       [-0.21666128,  0.71683095,  0.83463176, ...,  0.72086453,
         0.24339146,  1.24676906]])
```

In [62]:

```
# Apply the transformation on the test data
X_test = sc.transform(X_test)
```

In [63]:

```
X_test
```

Out[63]:

```
array([[-0.68701032, -0.27289013,  0.83463176, ...,  0.50829455,
         0.24339146, -0.44775345],
       [ 2.79357258,  0.71683095,  0.83463176, ..., -0.45535606,
        -0.53499465, -0.44775345],
       [ 0.34775757, -1.2626112 , -1.19813318, ...,  0.32406723,
         1.80016368, -0.44775345],
       ...,
       [ 1.19438584, -1.2626112 , -1.19813318, ..., -0.63958338,
        -1.31338076, -0.44775345],
```

```
        [ 1.38252546,  0.71683095,  0.83463176, ..., -0.56872671,
          0.24339146, -0.44775345],
        [-0.21666128,  0.71683095,  0.83463176, ..., -0.56872671,
          0.24339146, -0.44775345]])
```

In [64]:

```
param_grid = {
    'hidden_layer_sizes': [100], # 50, 200
    'max_iter': [2500], #5000,2500
    'solver': ['adam'], #sgd
    'tol': [0.01],
}

nncl = MLPClassifier(random_state=1)

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

In [65]:

```
grid_search.fit(X_train, train_labels)
grid_search.best_params_
#{'hidden_layer_sizes': 100, 'max_iter': 2500, 'solver': 'adam', 'tol': 0.01}
```

Out[65]:

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

In [66]:

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

Out[66]:

```
MLPClassifier(hidden_layer_sizes=100, max_iter=2500, random_state=1, tol=0.01)
```

Predicting the Training and Testing data

In [67]:

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

NN Model Performance Evaluation on Training data

In [68]:

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

Out[68]:

```
array([[1159,  200],
       [ 285,  358]], dtype=int64)
```

In [69]:

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

Out[69]:

```
0.7577422577422578
```

In [70]:

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

```
              precision    recall  f1-score   support
```

```
           0       0.80      0.85      0.83      1359
           1       0.64      0.56      0.60       643

    accuracy                           0.76      2002
   macro avg       0.72      0.70      0.71      2002
weighted avg       0.75      0.76      0.75      2002
```

In [71]:

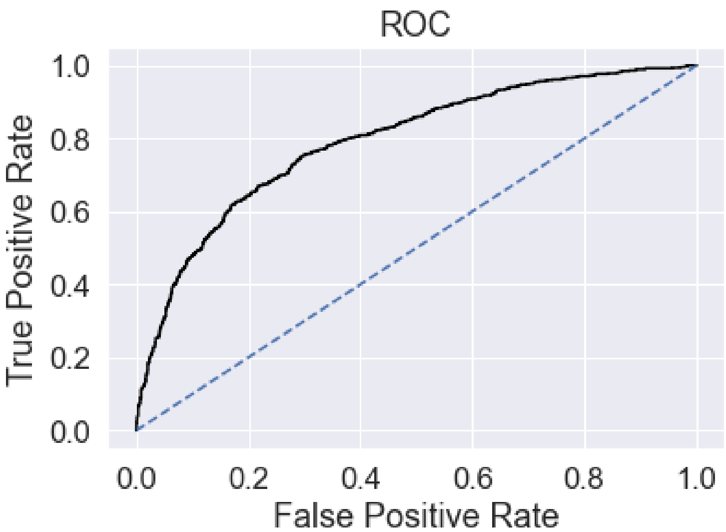
```
nn_metrics=classification_report(train_labels, ytrain_predict,output_dict=True)
df=pd.DataFrame(nn_metrics).transpose()
nn_train_precision=round(df.loc["1"][0],2)
nn_train_recall=round(df.loc["1"][1],2)
nn_train_f1=round(df.loc["1"][2],2)
print ('nn_train_precision ',nn_train_precision)
print ('nn_train_recall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

```
nn_train_precision  0.64
nn_train_recall  0.56
nn_train_f1  0.6
```

In [72]:

```
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 0.7921265636497425
```



NN Model Performance Evaluation on Test data

In [73]:

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

Out[73]:

```
array([[511,  77],
       [122, 149]], dtype=int64)
```

In [74]:

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

```
nn_test_acc
```

Out[74]:

```
0.7683352735739232
```

In [75]:

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

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84       588
           1       0.66      0.55      0.60       271

    accuracy                           0.77       859
   macro avg       0.73      0.71      0.72       859
weighted avg       0.76      0.77      0.76       859
```

In [76]:

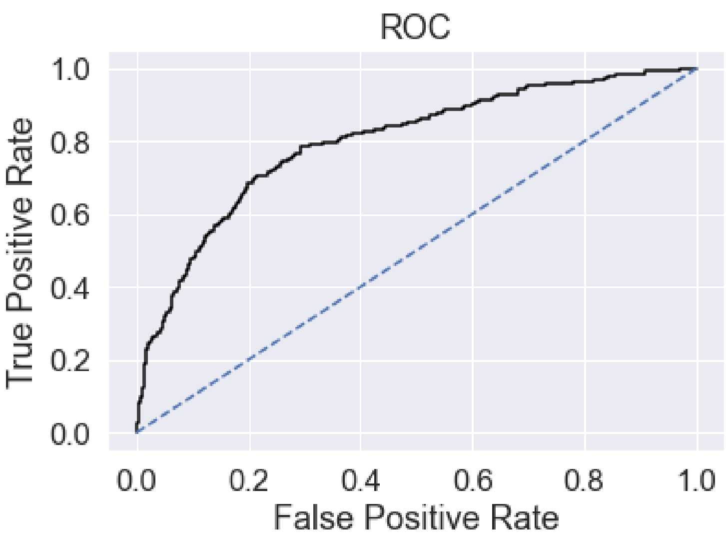
```
nn_metrics=classification_report(test_labels, ytest_predict,output_dict=True)
df=pd.DataFrame(nn_metrics).transpose()
nn_test_precision=round(df.loc["1"][0],2)
nn_test_recall=round(df.loc["1"][1],2)
nn_test_f1=round(df.loc["1"][2],2)
print ('nn_test_precision ',nn_test_precision)
print ('nn_test_recall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

```
nn_test_precision  0.66
nn_test_recall  0.55
nn_test_f1  0.6
```

In [77]:

```
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 0.7977947636619223
```



In [78]:

```
best_grid.score
```

Out[78]:

```
<bound method ClassifierMixin.score of MLPClassifier(hidden_layer_sizes=100, max_iter=250
0, random_state=1, tol=0.01)>
```

In [ ]:

```

```

Neural Network Conclusion

Train Data:

AUC: 79.15% Accuracy: 75.77% Precision: 64% f1-Score: 60% Recall : 56%

Test Data: AUC: 79.8% Accuracy: 76.8% Precision: 66% f1-Score: 60% Recall : 55% Training and Test set results are almost similar, and with the overall measures high, the model is a good model

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 [79]:

```
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
data = pd.DataFrame({'CART Train':[cart_train_acc,cart_train_auc,cart_train_recall,cart_
train_precision,cart_train_f1],
         'CART Test':[cart_test_acc,cart_test_auc,cart_test_recall,cart_test_precision,car
t_test_f1],
        'Random Forest Train':[rf_train_acc,rf_train_auc,rf_train_recall,rf_train_precisi
on,rf_train_f1],
         'Random Forest Test':[rf_test_acc,rf_test_auc,rf_test_recall,rf_test_precision,r
f_test_f1],
        'Neural Network Train':[nn_train_acc,nn_train_auc,nn_train_recall,nn_train_precis
ion,nn_train_f1],
         'Neural Network Test':[nn_test_acc,nn_test_auc,nn_test_recall,nn_test_precision,
nn_test_f1]},index=index)
round(data,2)
```

Out[79]:

	CART Train	CART Test	Random Forest Train	Random Forest Test	Neural Network Train	Neural Network Test
Accuracy	0.76	0.78	0.81	0.79	0.76	0.77
AUC	0.81	0.80	0.86	0.81	0.79	0.80
Recall	0.58	0.60	0.60	0.56	0.56	0.55
Precision	0.65	0.68	0.75	0.70	0.64	0.66
F1 Score	0.61	0.63	0.66	0.62	0.60	0.60

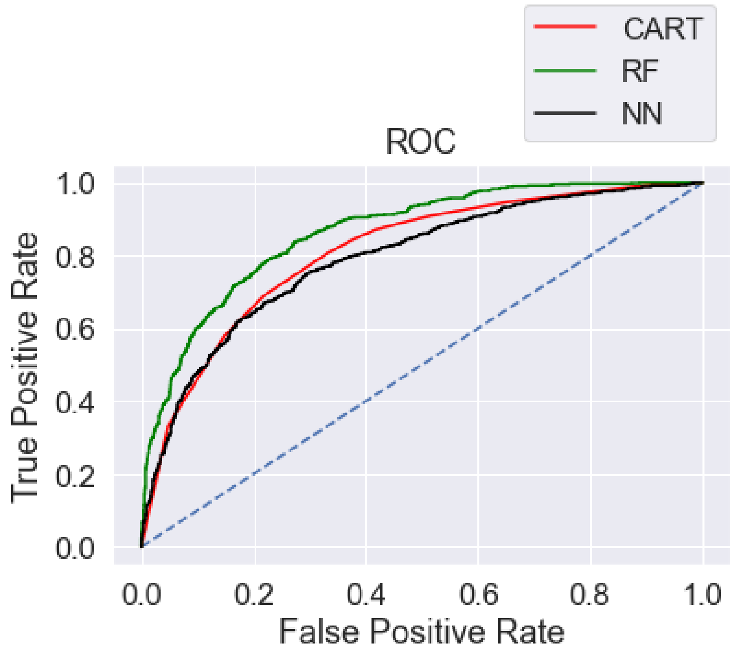
ROC Curve for the 3 models on the Training data

In [80]:

```
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., 1.02, 1., .102), loc='lower right')
```

Out[80]:

```
<matplotlib.legend.Legend at 0x1b04395ab50>
```



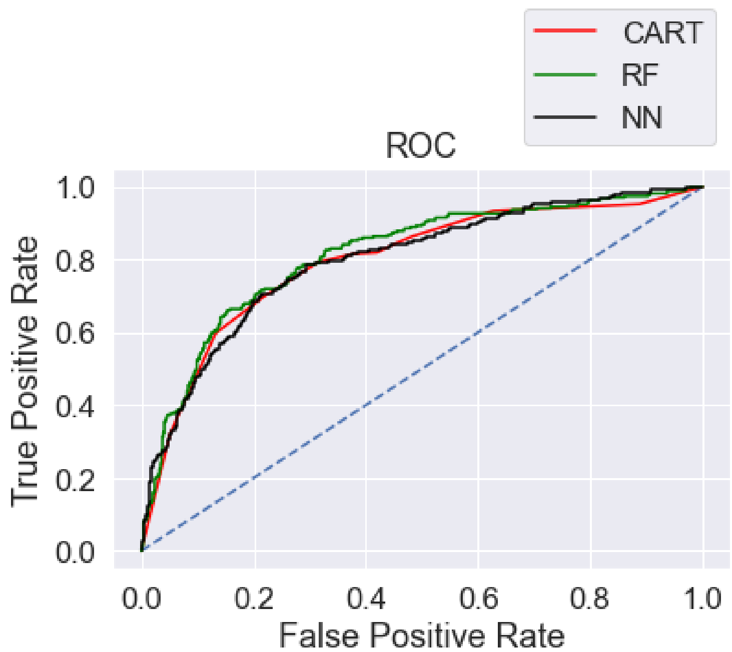
ROC Curve for the 3 models on the Test data

In [81]:

```
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[81]:

```
<matplotlib.legend.Legend at 0x1b04399dd00>
```



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

Out of the 3 models, CART has slightly better performance than the RF and Neural network model based on

recall. Based on precision, we prefer RF over CART and ANN. Overall all the 3 models are reasonaly stable enough to be used for making any future predictions. From Cart and Random Forest Model, the variable Agency code is found to be the most useful feature amongst all other features for predicting the target claimed .

In [ ]:

```

```

In [ ]:

```

```