1. Import Necessary Libraries

In [1]:

```
import pandas as pd
```

2.Import Dataset

In [2]:

```
claimants_data=pd.read_csv('claimants.csv')
claimants_data
```

Out[2]:

	CASENUM	ATTORNEY	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
0	5	0	0.0	1.0	0.0	50.0	34.940
1	3	1	1.0	0.0	0.0	18.0	0.891
2	66	1	0.0	1.0	0.0	5.0	0.330
3	70	0	0.0	1.0	1.0	31.0	0.037
4	96	1	0.0	1.0	0.0	30.0	0.038
...	...	...	...	...	...	...	...
1335	34100	1	0.0	1.0	0.0	NaN	0.576
1336	34110	0	1.0	1.0	0.0	46.0	3.705
1337	34113	1	1.0	1.0	0.0	39.0	0.099
1338	34145	0	1.0	0.0	0.0	8.0	3.177
1339	34153	1	1.0	1.0	0.0	30.0	0.688

1340 rows × 7 columns

In [3]:

```
claimants_data['ATTORNEY'].unique()
```

Out[3]:

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

3. Data Understanding

3.1 Initial Analysis

In [4]:

```
claimants_data.shape
```

Out[4]:

```
(1340, 7)
```

In [5]:

```
claimants_data.isna().sum()
```

Out[5]:

```
CASENUM       0
ATTORNEY      0
CLMSEX       12
CLMINSUR     41
SEATBELT     48
CLMAGE      189
LOSS          0
dtype: int64
```

In [6]:

```
claimants_data.dtypes
```

Out[6]:

```
CASENUM       int64
ATTORNEY      int64
CLMSEX      float64
CLMINSUR    float64
SEATBELT    float64
CLMAGE      float64
LOSS        float64
dtype: object
```

4.Data Prepartion

In [7]:

```
del claimants_data['CASENUM']
```

In [8]:

```
claimants_data.head()
```

Out[8]:

	ATTORNEY	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
0	0	0.0	1.0	0.0	50.0	34.940
1	1	1.0	0.0	0.0	18.0	0.891
2	1	0.0	1.0	0.0	5.0	0.330
3	0	0.0	1.0	1.0	31.0	0.037
4	1	0.0	1.0	0.0	30.0	0.038

In [9]:

```
claimants_data.dropna(axis=0,inplace=True)
```

In [10]:

```
claimants_data.isna().sum()
```

Out[10]:

```
ATTORNEY    0
CLMSEX      0
CLMINSUR    0
SEATBELT    0
CLMAGE      0
LOSS        0
dtype: int64
```

In [11]:

```
claimants_data.shape
```

Out[11]:

```
(1096, 6)
```

In [12]:

```
#Seeing balanced or unbalanced class
claimants_data['ATTORNEY'].value_counts()
```

Out[12]:

```
ATTORNEY
0    578
1    518
Name: count, dtype: int64
```

5.model Building

2 steps process:

1.Separate Input and Output 2.Perform some Model Validation Techniques:

```
        *Train-Test Split
        *K-Fold Cross Validation
        *Leave One Out Cross Validation
```

In [13]:

```
X = claimants_data.drop('ATTORNEY',axis=1)
Y = claimants_data[['ATTORNEY']]
```

In [14]:

```
X.shape,Y.shape
```

Out[14]:

```
((1096, 5), (1096, 1))
```

In [15]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.20,random_state=123,shuffle=True)
```

In [16]:

```
x_train.shape,y_train.shape
```

Out[16]:

```
((876, 5), (876, 1))
```

In [17]:

```
x_test.shape,y_test.shape
```

Out[17]:

```
((220, 5), (220, 1))
```

6.Model Training

In [18]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [19]:

```
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression()
logistic_model.fit(x_train,y_train)
```

Out[19]:

```
▾ LogisticRegression
LogisticRegression()
```

In [20]:

```
# from sklearn.tree import DecisionTreeClassifier
# dt = DecisionTreeClassifier()
# dt.fit(x_train,y_train)
```

In [21]:

```
%%time
logistic_model.fit(x_train,y_train)
```

```
CPU times: total: 31.2 ms
Wall time: 57.1 ms
```

Out[21]:

```
▾ LogisticRegression
LogisticRegression()
```

7.Model Testing

In [22]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,roc_auc_score,roc_curve
```

In [23]:

```
y_pred_train=logistic_model.predict(x_train)
```

In [24]:

```
accuracy_score(y_train,y_pred_train)
```

Out[24]:

```
0.7134703196347032
```

In [25]:

```
y_pred_test=logistic_model.predict(x_test)
accuracy_score(y_test,y_pred_test)
```

Out[25]:

```
0.6863636363636364
```

In [26]:

```
confusion_matrix(y_train,y_pred_train)
```

Out[26]:

```
array([[315, 151],
       [100, 310]])
```

In [27]:

```
#Accuracy
(315+310)/(315+151+100+310)
```

Out[27]:

```
0.7134703196347032
```

In [28]:

```
#Recall Zero row wise
rzero = 315/(315+151)
rzero
```

Out[28]:

```
0.6759656652360515
```

In [29]:

```
#Recall One
rone = 310/(100+310)
rone
```

Out[29]:

```
0.7560975609756098
```

In [30]:

```
#Precision zer0
pzero = 315/(315+100)
pzero
```

Out[30]:

```
0.7590361445783133
```

In [31]:

```
#Precision one
pone = 310/(151+310)
pone
```

Out[31]:

```
0.6724511930585684
```

In [32]:

```
avgr = (0.6759656652360515+0.7560975609756098)/2
avgr
```

Out[32]:

```
0.7160316131058306
```

In [33]:

```
avgp = (0.7590361445783133+0.6724511930585684)/2
avgp
```

Out[33]:

```
0.7157436688184409
```

```
In [34]: #F1 Score
         f1 = (2*avgr*avgp)/(avgr+avgp)
         f1
```

```
Out[34]: 0.715887612007902
```

```
In [35]: #Dorect function
         print(classification_report(y_train,y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.76      0.68      0.72       466
           1       0.67      0.76      0.71       410

    accuracy                           0.71       876
   macro avg       0.72      0.72      0.71       876
weighted avg       0.72      0.71      0.71       876
```

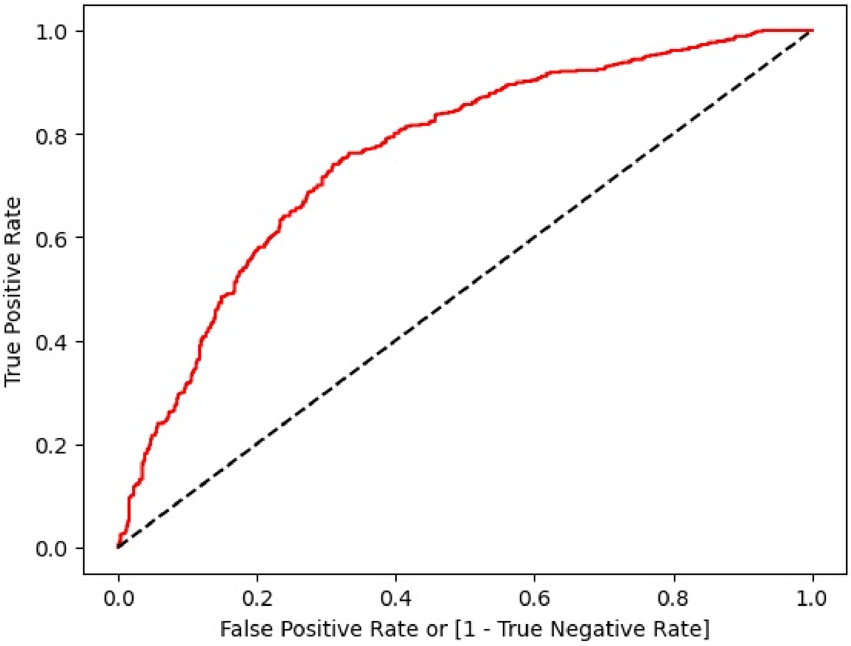
```
In [36]: from sklearn.metrics import roc_curve,roc_auc_score
         fpr, tpr, thresholds = roc_curve(Y,logistic_model.predict_proba(X)[:,1])

         auc = roc_auc_score(y_train,y_pred_train)
         print(auc)


         import matplotlib.pyplot as plt
         plt.plot(fpr, tpr, color = 'red', label = 'logit model(area =%0.2f)'%auc)
         plt.plot([0,1],[0,1],'k--')
         plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
         plt.ylabel('True Positive Rate')
```

```
0.7160316131058306
```

```
Out[36]: Text(0, 0.5, 'True Positive Rate')
```



```
In [37]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [38]: knn = KNeighborsClassifier() #build model
         knn.fit(x_train,y_train)
         y_pred_train= knn.predict(x_train)
         y_pred_test= knn.predict(x_test)
```

```
In [39]: confusion_matrix(y_train,y_pred_train)
```

```
Out[39]: array([[361, 105],
                [ 98, 312]])
```

```
In [40]: print(classification_report(y_train,y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.79      0.77      0.78       466
           1       0.75      0.76      0.75       410

    accuracy                           0.77       876
   macro avg       0.77      0.77      0.77       876
weighted avg       0.77      0.77      0.77       876
```

In [41]:
```
confusion_matrix(y_test,y_pred_test)
```

Out[41]:
```
array([[67, 45],
       [38, 70]])
```

In [42]:
```
print(classification_report(y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.64      0.60      0.62       112
           1       0.61      0.65      0.63       108

    accuracy                           0.62       220
   macro avg       0.62      0.62      0.62       220
weighted avg       0.62      0.62      0.62       220
```

In [43]:
```
# how to find best optimal k value
```

In [ ]:

8.Test Accuracy

In [44]:
```
y_pred_test=logistic_model.predict(x_test)
```

In [45]:
```
accuracy_score(y_test,y_pred_test)
print(classification_report(y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.74      0.59      0.66       112
           1       0.65      0.79      0.71       108

    accuracy                           0.69       220
   macro avg       0.70      0.69      0.68       220
weighted avg       0.70      0.69      0.68       220
```

In [46]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [47]:
```
base_model = DecisionTreeClassifier(max_depth=1)
abm = AdaBoostClassifier(estimator=base_model,n_estimators=1000,random_state=32,learning_rate=1.0)
abm.fit(x_train,y_train)
```

Out[47]:
```
▸ AdaBoostClassifier

▸ estimator: DecisionTreeClassifier

  ▸ DecisionTreeClassifier
```

In [48]:
```
y_pred_train = abm.predict(x_train)
print(classification_report(y_train,y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.75      0.78      0.76       466
           1       0.74      0.70      0.72       410

    accuracy                           0.74       876
   macro avg       0.74      0.74      0.74       876
weighted avg       0.74      0.74      0.74       876
```

In [49]:
```
y_pred_test = abm.predict(x_test)
print(classification_report(y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.69      0.80      0.74       112
           1       0.76      0.63      0.69       108

    accuracy                           0.72       220
   macro avg       0.72      0.72      0.72       220
weighted avg       0.72      0.72      0.72       220
```

```
In [50]: from sklearn.ensemble import GradientBoostingClassifier
```

```
In [51]: base_model = DecisionTreeClassifier()
         model = GradientBoostingClassifier()
         model.fit(x_train,y_train)
         y_pred_train = model.predict(x_train)
         y_pred_test = model.predict(x_test)
```

```
In [52]: print(classification_report(y_train,y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       466
           1       0.80      0.76      0.78       410

    accuracy                           0.80       876
   macro avg       0.80      0.79      0.79       876
weighted avg       0.80      0.80      0.80       876
```

```
In [53]: print(classification_report(y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.67      0.76      0.71       112
           1       0.71      0.62      0.66       108

    accuracy                           0.69       220
   macro avg       0.69      0.69      0.69       220
weighted avg       0.69      0.69      0.69       220
```

9.Model Deployment

```
In [56]: # from pickle import dump
```

```
In [57]: # dump(logistic_model,open('claimants_intelligene.pkl','wb'))
```

```
In [58]: # from pickle import load
```

```
In [59]: # loaded_intelligence=load(open('claimants_intelligene.pkl','rb'))
```

```
In [60]: # y_pred_deployment=loaded_intelligence.predict(x_test)
```

1.STANDARDIZE AND CHECK THE ACCURACY

```
In [61]: from sklearn.preprocessing import StandardScaler
         std_scaler=StandardScaler()
         scaled_x=std_scaler.fit_transform(X)
         scaled_x
```

```
Out[61]: array([[-1.13916369,  0.32550512, -0.13633547,  1.05048704,  2.96924493],
                [ 0.87783697, -3.07214831, -0.13633547, -0.51942439, -0.28328699],
                [-1.13916369,  0.32550512, -0.13633547, -1.15720091, -0.33687653],
                ...,
                [ 0.87783697,  0.32550512, -0.13633547,  0.51082998, -0.35894281],
                [ 0.87783697, -3.07214831, -0.13633547, -1.01002171, -0.06491676],
                [ 0.87783697,  0.32550512, -0.13633547,  0.0692924 , -0.30267857]])
```

```
In [62]: x_train,x_test,y_train,y_test=train_test_split(scaled_x,Y,test_size=0.20,random_state=123)
```

```
In [63]: %%time
         logistic_model.fit(x_train,y_train)
```

```
CPU times: total: 15.6 ms
Wall time: 64 ms
```

Out[63]:

```
▾ LogisticRegression ⓘ ?
LogisticRegression()
```

In [64]:

```
y_pred_train = logistic_model.predict(x_train)
```

In [65]:

```
accuracy_score(y_train,y_pred_train)
```

Out[65]:

```
0.7180365296803652
```

In [66]:

```
y_pred_test = logistic_model.predict(x_test)
```

In [67]:

```
accuracy_score(y_test,y_pred_test)
```

Out[67]:

```
0.6681818181818182
```