1. Import Necessary libraries

In [4]:

```
import pandas as pd
```

2. Import Data

In [5]:

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

Out[5]:

	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

3. Data understanding

In [6]:

```
claimants_data.shape
```

Out[6]:

```
(1340, 7)
```

In [7]:

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

Out[7]:

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

In [8]:

```
claimants_data.head(50)
```

Out[8]:

	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
5	97	0	1.0	1.0	0.0	35.0	0.309
6	10	0	0.0	1.0	0.0	9.0	3.538
7	36	0	1.0	1.0	0.0	34.0	4.881
8	51	1	1.0	1.0	0.0	60.0	0.874
9	55	1	0.0	1.0	0.0	NaN	0.350
10	61	0	1.0	1.0	0.0	37.0	6.190
11	148	0	0.0	1.0	0.0	41.0	19.610
12	150	1	0.0	1.0	0.0	7.0	1.678
13	150	0	1.0	1.0	0.0	40.0	0.673
14	169	1	1.0	1.0	0.0	37.0	0.143
15	171	1	1.0	0.0	0.0	9.0	0.053
16	334	1	1.0	1.0	0.0	58.0	0.050
17	360	0	0.0	1.0	0.0	58.0	0.758
18	376	1	0.0	1.0	0.0	3.0	0.000
19	401	0	1.0	1.0	0.0	38.0	4.754
20	479	0	0.0	NaN	0.0	37.0	3.100
21	480	1	1.0	1.0	0.0	39.0	0.130
22	550	0	0.0	0.0	0.0	38.0	16.161
23	569	0	0.0	NaN	0.0	30.0	0.609
24	580	1	0.0	1.0	0.0	54.0	10.040
25	608	1	1.0	1.0	0.0	3.0	0.787
26	603	1	0.0	1.0	0.0	61.0	0.150
27	606	0	0.0	1.0	0.0	0.0	0.405
28	607	1	0.0	1.0	0.0	40.0	0.405
29	630	0	0.0	1.0	0.0	NaN	0.595
30	640	1	1.0	1.0	0.0	16.0	3.994
31	640	1	0.0	1.0	0.0	NaN	0.337
32	685	1	0.0	1.0	0.0	9.0	0.603
33	700	0	1.0	1.0	0.0	35.0	1.673

	CASENUM	ATTORNEY	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
34	710	0	1.0	1.0	0.0	35.0	4.751
35	766	0	1.0	1.0	0.0	17.0	3.538
36	775	0	1.0	0.0	0.0	41.0	3.185
37	805	0	1.0	1.0	0.0	50.0	3.700
38	819	1	0.0	1.0	0.0	33.0	0.386
39	838	0	0.0	1.0	0.0	9.0	1.970
40	851	1	0.0	1.0	0.0	34.0	0.904
41	871	0	1.0	0.0	0.0	33.0	8.090
42	905	0	1.0	1.0	0.0	46.0	8.090
43	941	1	1.0	1.0	0.0	55.0	13.100
44	971	1	0.0	1.0	0.0	5.0	0.460
45	61	1	1.0	1.0	0.0	NaN	3.981
46	63	0	0.0	0.0	0.0	16.0	1.740
47	78	1	0.0	1.0	0.0	1.0	0.000
48	91	0	0.0	1.0	0.0	NaN	1.003
49	0	0	0.0	1.0	0.0	40.0	3.100

In [9]:

```
claimants_data.dtypes
```

Out[9]:

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

4. Data Preprocessing

In [10]:

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

In [11]:

```
claimants_data.shape
```

Out[11]:

```
(1340, 6)
```

In [12]:

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

In [13]:

```
claimants_data.shape
```

Out[13]:

```
(1096, 6)
```

In [14]:

```
1340-1096
```

Out[14]:

```
244
```

5. Model Building

In [15]:

```
claimants_data
```

Out[15]:

	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
...	...	...	...	...	...	...
1334	1	1.0	1.0	0.0	16.0	0.060
1336	0	1.0	1.0	0.0	46.0	3.705
1337	1	1.0	1.0	0.0	39.0	0.099
1338	0	1.0	0.0	0.0	8.0	3.177
1339	1	1.0	1.0	0.0	30.0	0.688

1096 rows × 6 columns

In [16]:

```
x=claimants_data.drop('ATTORNEY',axis=1)
y=claimants_data[['ATTORNEY']]
```

In [17]:

```
x.shape,y.shape
```

Out[17]:

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

In [18]:

```
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)
```

In [30]:

```
x_train
```

Out[30]:

	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
575	0.0	1.0	0.0	70.0	3.706
1257	0.0	1.0	0.0	7.0	0.453
804	1.0	1.0	0.0	47.0	5.004
349	1.0	1.0	0.0	34.0	4.349
251	1.0	1.0	0.0	50.0	6.608
...	...	...	...	...	...
777	1.0	1.0	0.0	6.0	0.046
143	0.0	1.0	0.0	4.0	8.616
120	1.0	1.0	0.0	10.0	0.305
134	1.0	1.0	0.0	31.0	4.460
1272	1.0	1.0	0.0	34.0	0.150

876 rows × 5 columns

K-fold CV

In [102]:

```
from sklearn.model_selection import KFold,cross_val_score

cv_scores=cross_val_score(estimator=logistic_model,X=x,y=y,cv=5)
print('CV Scores :',cv_scores)
print('Mean accuracy :',cv_scores.mean())
print('Std.Deviation:',cv_scores.std())
```

```
CV Scores : [0.71363636 0.69863014 0.69863014 0.70319635 0.70776256]
Mean accuracy : 0.7043711083437111
Std.Deviation: 0.005738369952269936
```

In [105]:

```
from sklearn.model_selection import KFold,cross_val_score

kfold_split = KFold(n_splits=5,shuffle=True,random_state=12)
print('CV Scores :',cv_scores)
print('Mean accuracy :',cv_scores.mean())
print('Std.Deviation:',cv_scores.std())
```

```
CV Scores : [0.71363636 0.69863014 0.69863014 0.70319635 0.70776256]
Mean accuracy : 0.7043711083437111
Std.Deviation: 0.005738369952269936
```

LOOCV

In [108]:

```
from sklearn.model_selection import LeaveOneOut
loo_split=LeaveOneOut()
cv_scores=cross_val_score(estimator=logistic_model,X=x,y=y,cv=loo_split)
print('CV Scores :',cv_scores)
print('Mean accuracy :',cv_scores.mean())
print('Std.Deviation:',cv_scores.std())
```

```
CV Scores : [1. 1. 1. ... 1. 1. 1.]
Mean accuracy : 0.7025547445255474
Std.Deviation: 0.4571340891578643
```

In [ ]:

```

```

In [ ]:

```

```

In [100]:

```
x_train.shape,x_test.shape
```

Out[100]:

```
((876, 5), (220, 5))
```

6. Model Training

In [32]:

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

In [54]:

```
from sklearn.linear_model import LogisticRegression
logistic_model=LogisticRegression() #Model initialization/object creation/Estimator
logistic_model.fit(x_train,y_train )

from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(max_depth=5)
dt_model.fit(x_train,y_train )

from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
rf_classifier=RandomForestClassifier(n_estimators=100,max_depth=4)
rf_classifier.fit(x_train,y_train)

gb_classifier=GradientBoostingClassifier(max_depth=3)
gb_classifier.fit(x_train,y_train)
```

Out[54]:

```
GradientBoostingClassifier()
```

Automate the way for getting optimal values for Hyperparameters

In [55]:

```
from sklearn.model_selection import GridSearchCV

grid_search_cv=GridSearchCV(estimator=dt_model,param_grid={'criterion':['entropy','gini'],
                                                           'max_depth':[1,2,3,4,5,6,7,8,9,1
                            cv=5)
grid_search_cv.fit(x,y)
print(grid_search_cv.best_params_)
print(grid_search_cv.best_score_)
```

```
{'criterion': 'gini', 'max_depth': 3}
0.7326400996264011
```

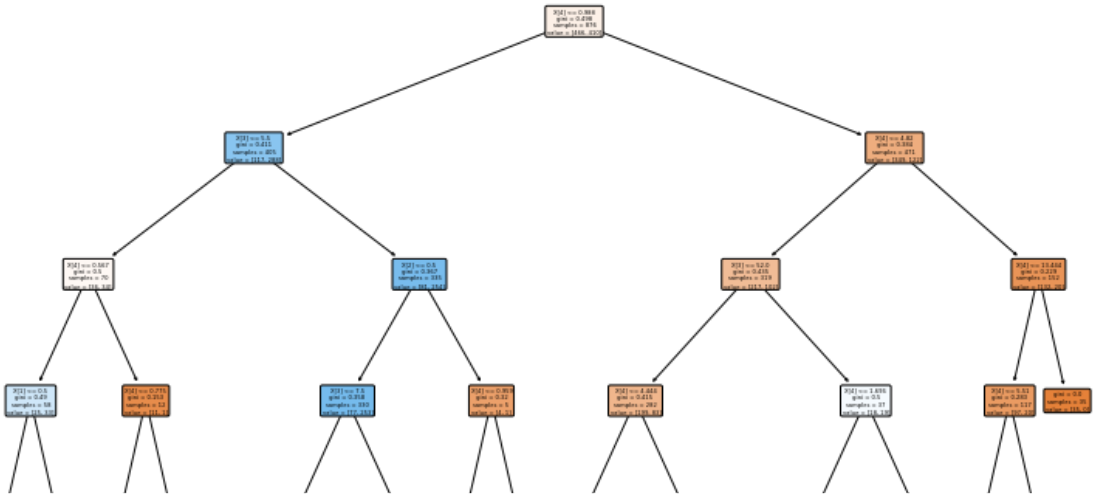
Plot The Tree

In [56]:

```
from sklearn.tree import plot_tree
from matplotlib import pyplot as plt
plt.figure(figsize=(13,9))
plot_tree(decision_tree=dt_model,filled=True,rounded=True)
plt.show
```

Out[56]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [ ]:

```

```

In [57]:

```
logistic_model.intercept_
```

Out[57]:

```
array([-0.20480482])
```

In [58]:

```
logistic_model.coef_
```

Out[58]:

```
array([[ 0.40131585,  0.54554594, -0.78114554,  0.01003038, -0.39685852]])
```

7. Model Testing || 8.Model Evaluation

Training Data

In [59]:

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

In [60]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_pred_train)
```

Out[60]:

```
0.75
```

In [ ]:

```

```

In [61]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
print('Accuracy Score :',accuracy_score(y_train,y_pred_train))
print('Confusion Matrix:\n',confusion_matrix(y_train,y_pred_train))
print('-----------------------------------------------------------')
print('Classification Report:\n',classification_report(y_train,y_pred_train))
```

```
Accuracy Score : 0.75
Confusion Matrix:
 [[363 103]
 [116 294]]
-----------------------------------------------------------
Classification Report:
               precision    recall  f1-score   support

           0       0.76      0.78      0.77       466
           1       0.74      0.72      0.73       410

    accuracy                           0.75       876
   macro avg       0.75      0.75      0.75       876
weighted avg       0.75      0.75      0.75       876
```

In [62]:

```
from sklearn.metrics import roc_curve,roc_auc_score
fpr,tpr,thresholds = roc_curve(y,rf_classifier.predict_proba (x)[:,1])

auc=roc_auc_score(y_train,y_pred_train)
print('Area Under the Curve[AUC]:',auc)
```

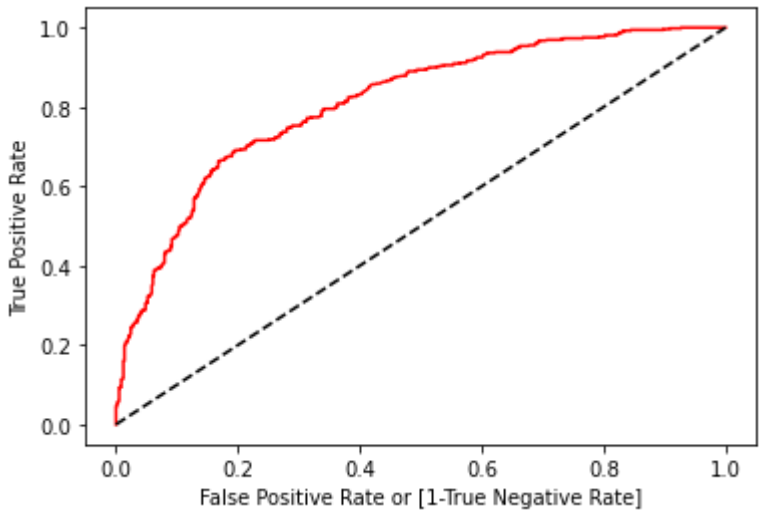
```
Area Under the Curve[AUC]: 0.7480215639066262
```

In [63]:

```
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')
```

Out[63]:

```
Text(0, 0.5, 'True Positive Rate')
```



Test data

In [64]:

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

In [65]:

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

Out[65]:

```
0.7181818181818181
```

9.Model deployment

In [66]:

```
from pickle import dump
```

In [67]:

```
dump(logistic_model,open('log_model_intelligence.pkl','wb'))
```

In [68]:

```
from pickle import load
```

In [69]:

```
loaded_model=load(open('log_model_intelligence.pkl','rb'))
```

In [70]:

```
y_pred=loaded_model.predict(x_test)
```

In [71]:

```
accuracy_score(y_test,y_pred)
```

Out[71]:

```
0.6863636363636364
```

==========

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [44]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
print('Accuracy Score :',accuracy_score(y_train,y_pred_train))
print('---------------------------------------')
print('Confusion Matrix:\n',confusion_matrix(y_train,y_pred_train))
print('---------------------------------------')
print('Classification Report :\n',classification_report(y_train,y_pred_train))
```

```
Accuracy Score : 0.9965753424657534
---------------------------------------
Confusion Matrix:
 [[466   0]
 [  3 407]]
---------------------------------------
Classification Report :
               precision    recall  f1-score   support

           0       0.99      1.00      1.00       466
           1       1.00      0.99      1.00       410

    accuracy                           1.00       876
   macro avg       1.00      1.00      1.00       876
weighted avg       1.00      1.00      1.00       876
```

In [ ]:

```
```

Test data

In [45]:

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

In [46]:

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

Out[46]:

```
0.6863636363636364
```

Model deployment

In [47]:

```
from pickle import dump
```

In [48]:

```
dump(logistic_model,open('log_model_intelligence.pkl','wb'))
```

In [49]:

```
from pickle import load
```

In [50]:

```
loaded_model=load(open('log_model_intelligence.pkl','rb'))
```

In [52]:

```
y_pred
```

Out[52]:

```
array([0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1],
      dtype=int64)
```

==========

Stacking Technique

In [91]:

```
from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC

log_model=LogisticRegression()
dt_model=DecisionTreeClassifier()
svc_model=SVC()

voting_classifier=VotingClassifier(estimators=[('logistic_model',log_model),
                                               ('decision_tree',dt_model),
                                               ('svclassifier',svc_model)])

voting_classifier.fit(x_train,y_train)
y_pred=voting_classifier.predict(x_test)
accuracy_score(y_test,y_pred)
```

Out[91]:

```
0.6681818181818182
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: