In [1]:

```
#importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Reading the dataset

In [3]:

```
#importing the dataset
data = pd.read_csv('water_potability.csv')
data.head()
```

Out[3]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279

Exploratory Data Analysis

EDA of variable

In [4]:

```
# checking for null values
data.isna().sum()
```

Out[4]:

```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

In [5]:

```
#chcking descriptive statistics results
data.describe()
```

Out[5]:

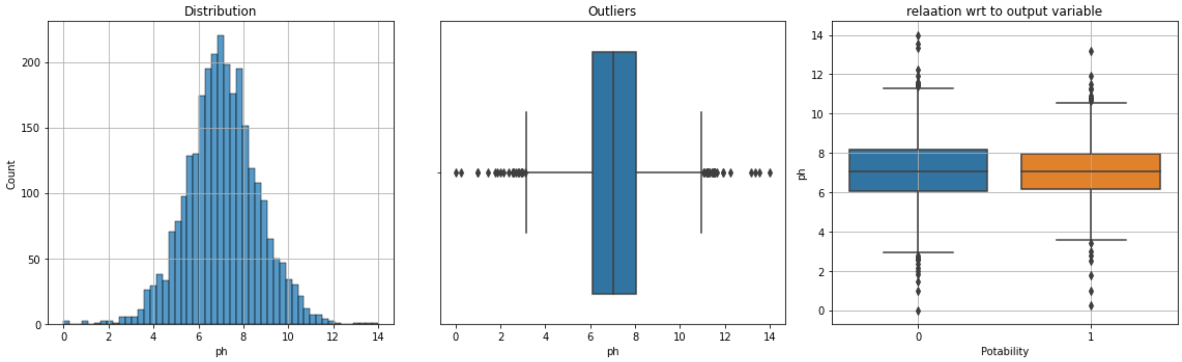
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organi
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	327
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	

In [6]:

```
# defining function for EDA
def conti_var(x):
    fig, axes = plt.subplots(nrows=1,ncols=3,figsize=(16,5),tight_layout=True)

    axes[0].set_title('Distribution')
    sns.histplot(x,ax=axes[0])
    axes[0].grid()

    axes[1].set_title('Outliers')
    sns.boxplot(x,ax=axes[1])

    axes[2].set_title('relaation wrt to output variable')
    sns.boxplot(x=data.Potability,y=x)
    axes[2].grid()
```

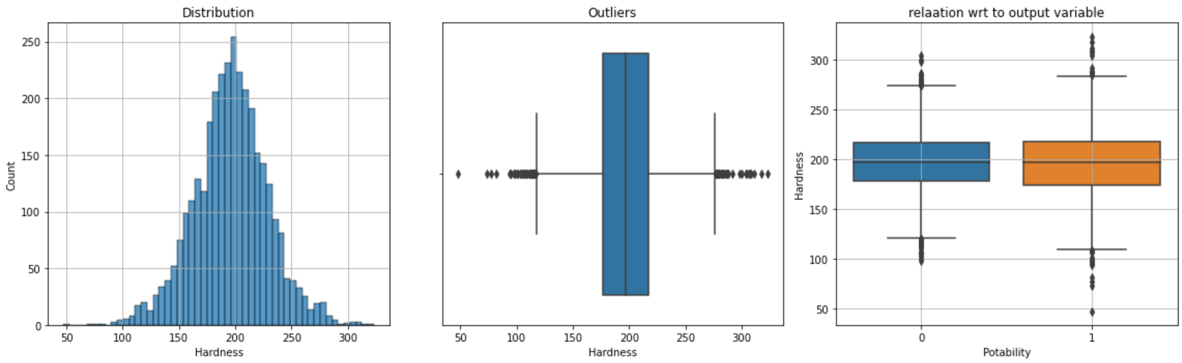
In [7]:

```
#EDA of Ph variable
conti_var(data.ph)
```



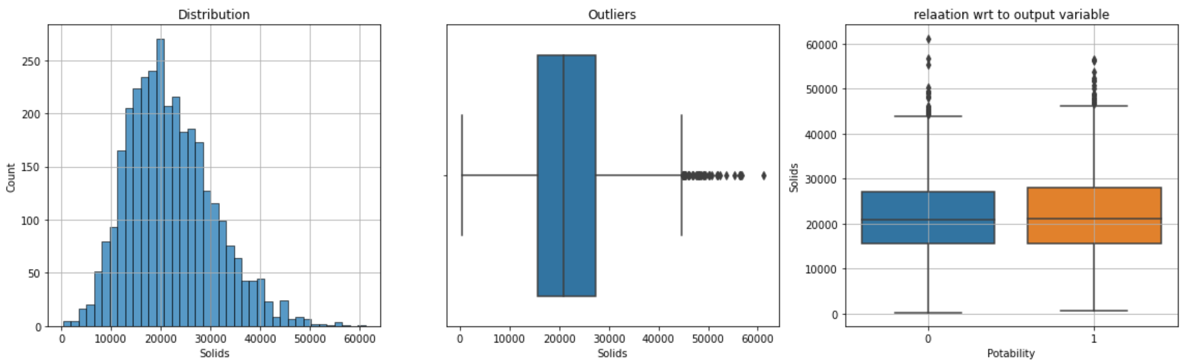
In [8]:

```
#EDA of Hardness variable
conti_var(data.Hardness)
```



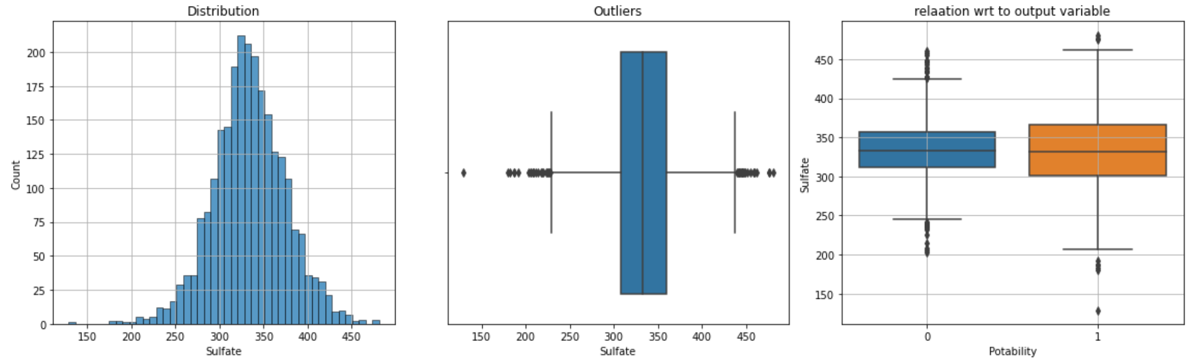
In [9]:

```
#EDA of solids
conti_var(data.Solids)
```



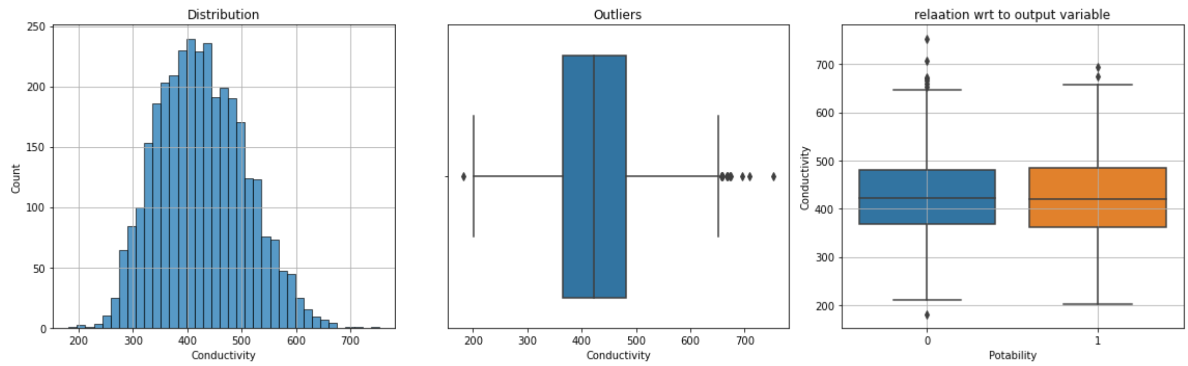
In [10]:

```
#EDA of sulfates
conti_var(data.Sulfate)
```



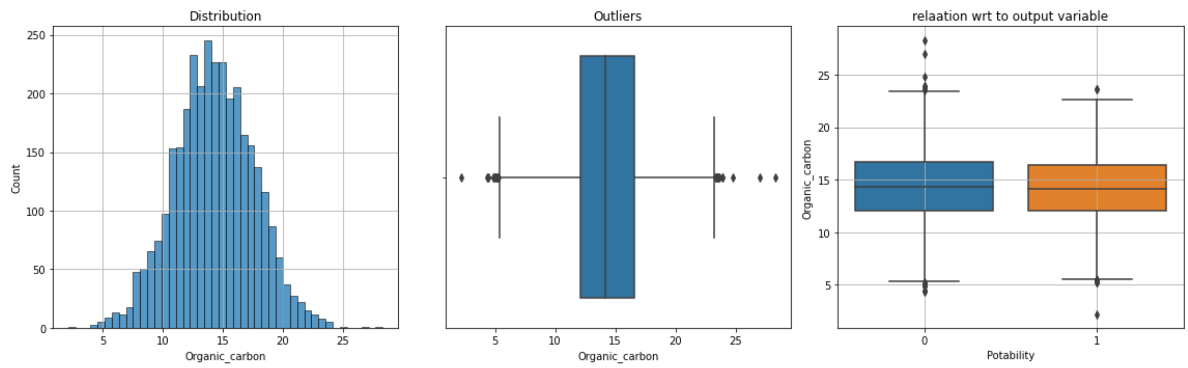
In [11]:

```
#EDA of Conductivity variable
conti_var(data.Conductivity)
```



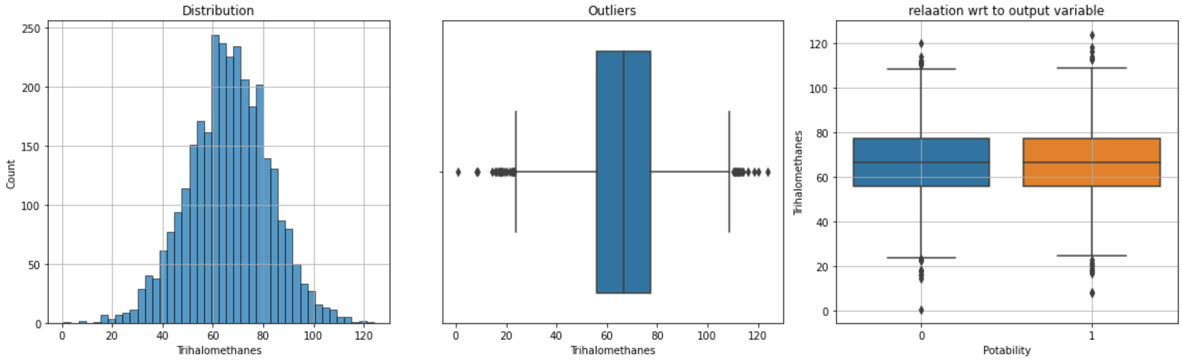
In [13]:

```
# EDA of Organic_carbon variable
conti_var(data.Organic_carbon)
```



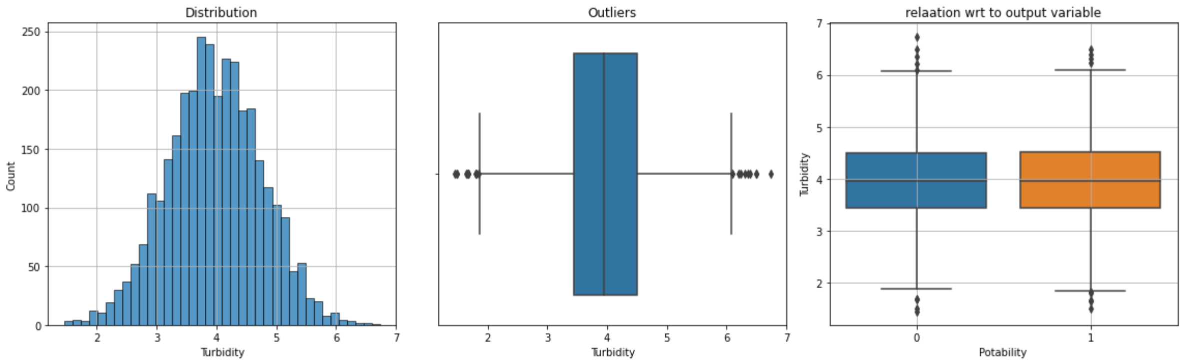
In [14]:

```
# EDA of Trihalomethanes variable
conti_var(data.Trihalomethanes)
```



In [15]:

```
# EDA of Turbidity variable
conti_var(data.Turbidity)
```



In [16]:

```

#checking output variable for unbalanced dataset
data.Potability.value_counts()
```

Out[16]:

```
0    1998
1    1278
Name: Potability, dtype: int64
```

From the above EDA, observations are

1) Its not a unbalanced dataset.

2) Almost all the input variables are normally distributed

Imputing missing values

In [17]:

```
# Since Trihalomethanes and pH has less number of missing values, they are imputed with med
data.ph.fillna(data.ph.median(),inplace=True)
data.Trihalomethanes.fillna(data.Trihalomethanes.median(),inplace=True)
```

In [18]:

```
#splitting the data
test_x = data[data.Sulfate.isna()].drop('Sulfate',axis=1)
train_x = data[data.Sulfate.notna()].drop('Sulfate',axis=1)
train_y = data.Sulfate[data.Sulfate.notna()]

#splitting the shape of splitted data
print('train_x = {}, train_y={}, test_x={}'.format(train_x.shape,train_y.shape,test_x.shape
```

```
train_x = (2495, 9), train_y=(2495,), test_x=(781, 9)
```

In [19]:

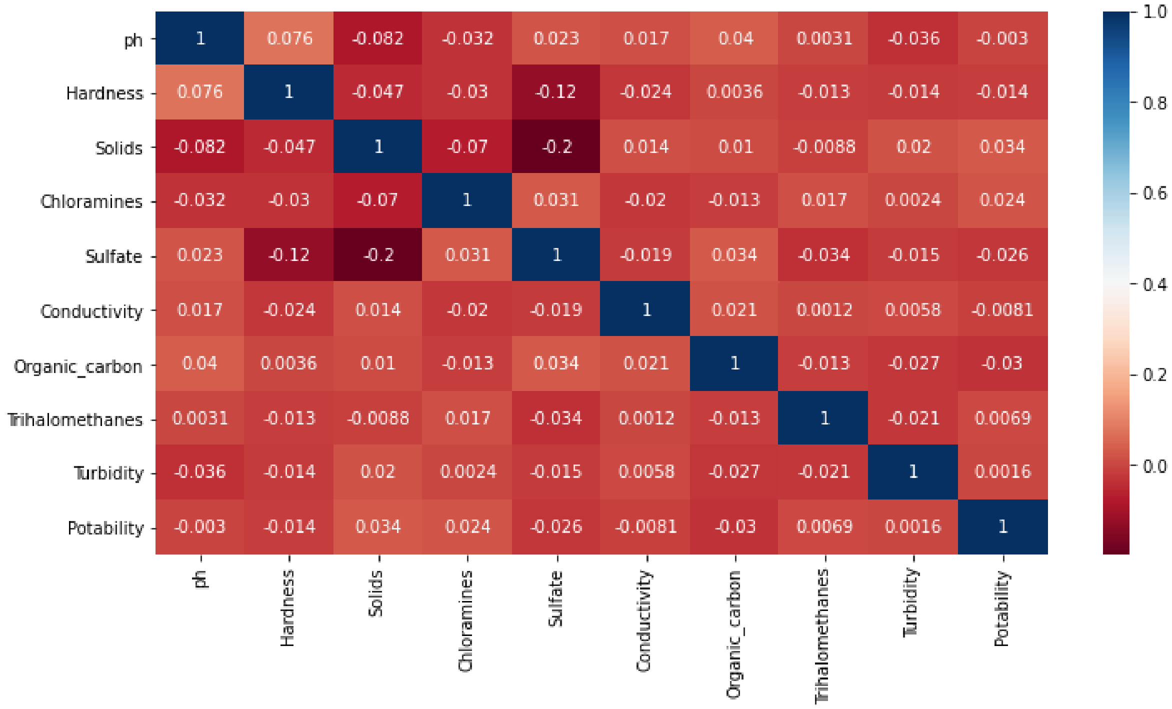
```
#since sulfate variable has more missing values, they are filled with linear regression alg
#importing missing values
from sklearn.linear_model import LinearRegression

#initializing the model
lin = LinearRegression()

#fitting the model
lin.fit(train_x,train_y)

#predicting the missing values
for i in data[data.Sulfate.isna()].index:
    data.Sulfate[i] = lin.predict([data.loc[i,data.columns != 'Sulfate']])
```

In [22]:

```
# checking the corealtion
plt.figure(figsize=(12,6))
sns.heatmap(data.corr(),annot=True,cmap='RdBu')
```

Out[22]:

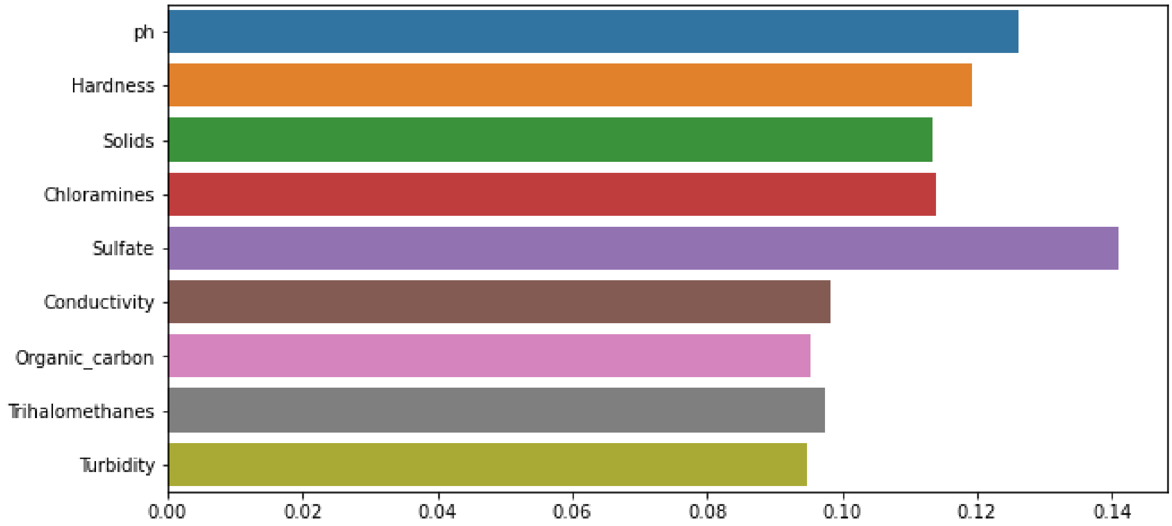
```
<AxesSubplot:>
```



Feature Selection

In [23]:

```
#Feature selection using random forest feature importance
#importing the libraries
from sklearn.ensemble import RandomForestClassifier

#initializing the model
ran = RandomForestClassifier()

#fitting the model
ran.fit(data.drop('Potability',axis=1),data.Potability)

plt.figure(figsize=(10,5))
sns.barplot(x=ran.feature_importances_,y=data.drop('Potability',axis=1).columns)
plt.show()
```



Since there is not much significance will be derived from above plot all the input variables are considered for prediction

In [24]:

```
#splitting the data into input and output
x = data.drop(['Potability','Organic_carbon'],axis=1)
y = data.Potability

print('input shape={}, output shape={}'.format(x.shape,y.shape))
```

```
input shape=(3276, 8), output shape=(3276,)
```

In [25]:

```
#Standard scalar is used to avoid scaling effect
#importing the libraries
from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()

#fitting scalar model for input data
x = pd.DataFrame(scalar.fit_transform(x),columns=x.columns)
```

In [26]:

```
#splitting entire data into 80% train and 20% test
# importing the libraries
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=1)

print('Shape of Splitting:')
print('x_train={},y_train={},x_test={},y_test={}'.format(x_train.shape,y_train.shape,x_test
```

```
Shape of Splitting:
x_train=(2620, 8),y_train=(2620,),x_test=(656, 8),y_test=(656,)
```

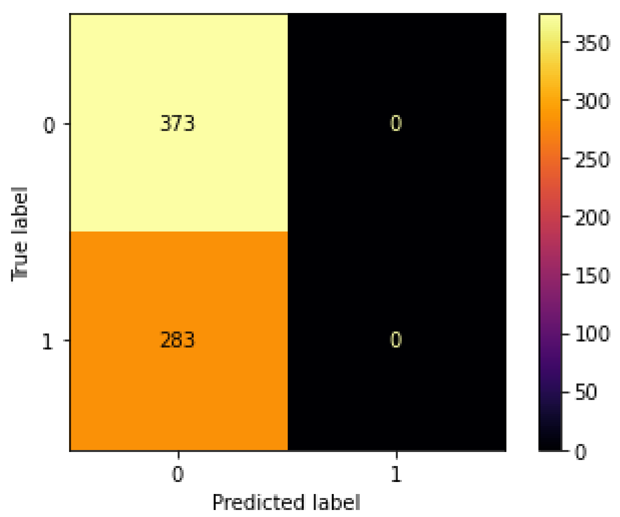
Building the model

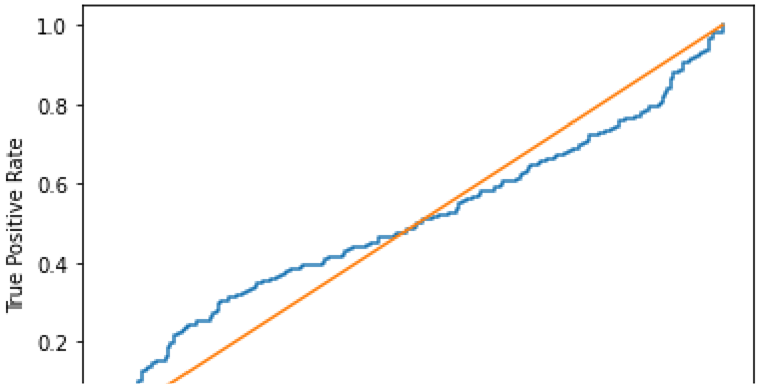
Logistic regression model

In [27]:

```
#importing libraries
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, plot_confusion_matrix, plot_roc_curve, a

#initializing the model
logis = LogisticRegression()

#fitting and predicting for test data
pred_logis = logis.fit(x_train,y_train).predict(x_test)

#printing the report
print('Report: \n',classification_report(y_test,pred_logis))

#confusion matrix
print('confusion matrix:')
plot_confusion_matrix(logis,x_test,y_test,cmap='inferno')
plt.show()

#plotting the ROC curve
plot_roc_curve(logis,x_test,y_test)
plt.plot([0,1],[0,1])
plt.show()

#accuracy score
acc_logis = accuracy_score(y_test,pred_logis)
```

```
Report: 
               precision    recall  f1-score   support

           0       0.57      1.00      0.72       373
           1       0.00      0.00      0.00       283

    accuracy                           0.57       656
   macro avg       0.28      0.50      0.36       656
weighted avg       0.32      0.57      0.41       656

confusion matrix:
```



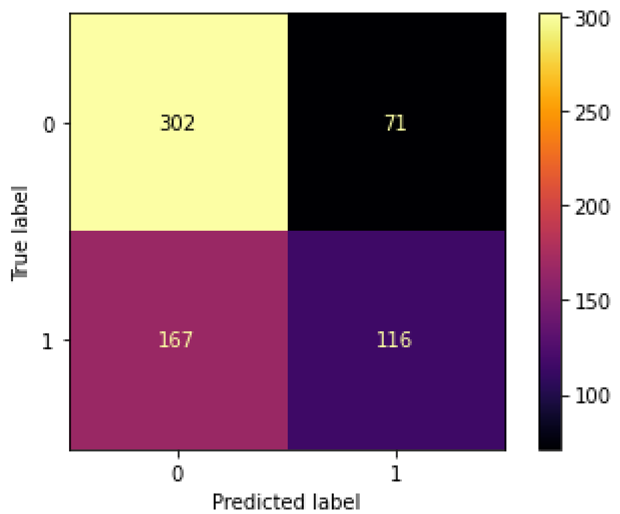


KNN MODEL

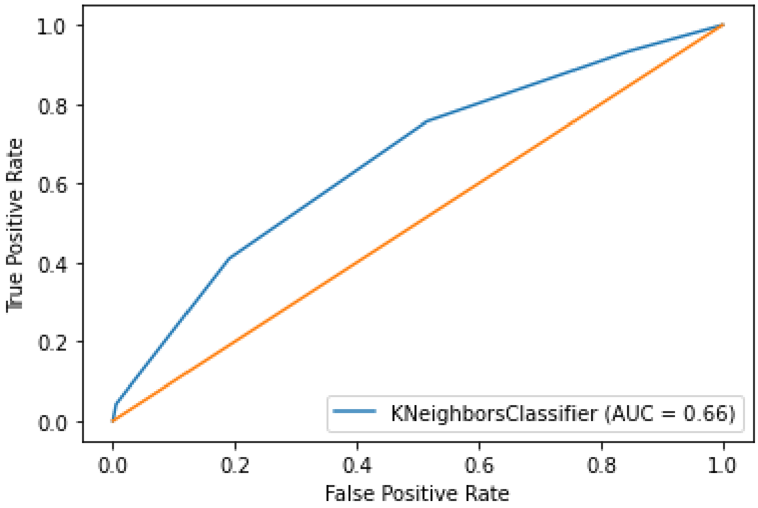
In [28]:

```
#importing libraries
from sklearn.neighbors import KNeighborsClassifier

#initializing the model
knn = KNeighborsClassifier()

#fitting and predicting for test data
pred_knn = knn.fit(x_train,y_train).predict(x_test)

#printing the report
print('Report: \n',classification_report(y_test,pred_knn))

#confusion matrix
print('confusion matrix:')
plot_confusion_matrix(knn,x_test,y_test,cmap='inferno')
plt.show()

#plotting the ROC curve
print('ROC curve :')
plot_roc_curve(knn,x_test,y_test)
plt.plot([0,1],[0,1])
plt.show()
```

```
Report:
               precision    recall  f1-score   support

           0       0.64      0.81      0.72       373
           1       0.62      0.41      0.49       283

    accuracy                           0.64       656
   macro avg       0.63      0.61      0.61       656
weighted avg       0.63      0.64      0.62       656

confusion matrix:
```




```
ROC curve :
```



In [29]:

```
#checking hyper parameters
knn.get_params().keys()
```

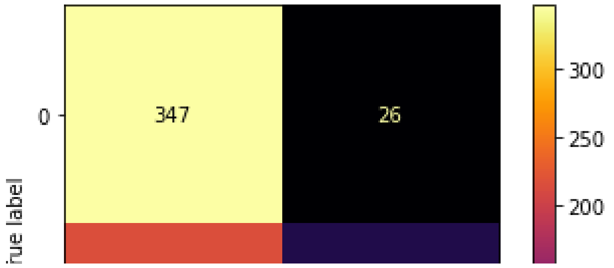
Out[29]:

```
dict_keys(['algorithm', 'leaf_size', 'metric', 'metric_params', 'n_jobs', 'n
_neighbors', 'p', 'weights'])
```

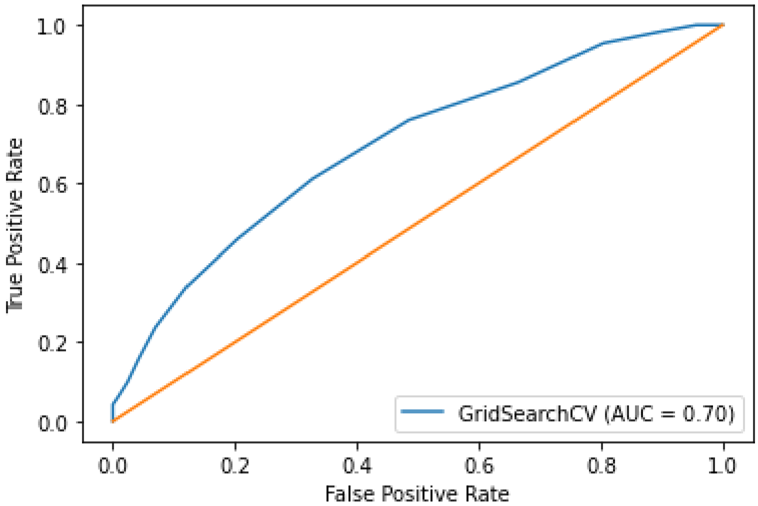
In [32]:

```
#hyper parameters
params = {'n_neighbors':range(1,25)}

#initializing the grid
grid_knn = GridSearchCV(estimator=knn,param_grid=params,cv=3,verbose=3,n_jobs=-1)

#fitting for test data
pred_knn = grid_knn.fit(x_train,y_train).predict(x_test)

#printing best score and parameters
print('Best score = {}\nBest params = {}'.format(grid_knn.best_score_,grid_knn.best_params_

#printing the report
print('Report: \n',classification_report(y_test,pred_knn))

#confusion matrix
print('confusion matrix:')
plot_confusion_matrix(grid_knn,x_test,y_test,cmap='inferno')
plt.show()

#plotting the ROC curve
print('ROC curve :')
plot_roc_curve(grid_knn,x_test,y_test)
plt.plot([0,1],[0,1])
plt.show()

#accuracy score
acc_knn = accuracy_score(y_test,pred_knn)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    0.3s
[Parallel(n_jobs=-1)]: Done  57 out of  72 | elapsed:    1.2s remaining:
0.2s
[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed:    1.4s finished

Best score = 0.6503831794237324
Best params = {'n_neighbors': 20}
Report:
               precision    recall  f1-score   support

           0       0.62      0.93      0.74       373
           1       0.72      0.24      0.36       283

    accuracy                           0.63       656
   macro avg       0.67      0.58      0.55       656
weighted avg       0.66      0.63      0.58       656

confusion matrix:
```



ROC curve :

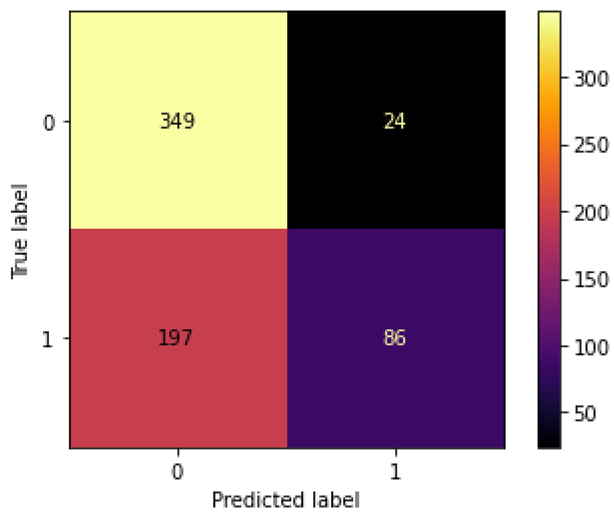


SVM model

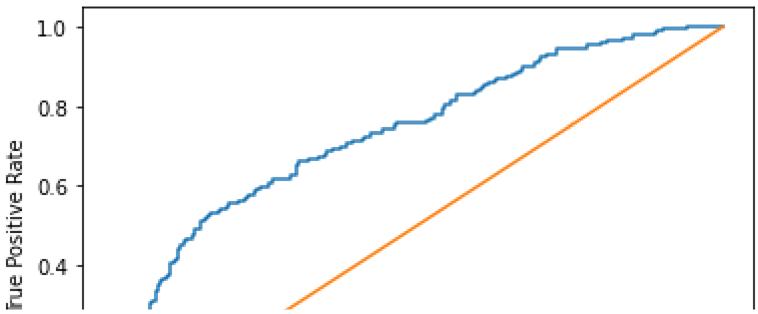
In [33]:

```
#importing libraries
from sklearn.svm import SVC

#initializing the model
svm = SVC()

#fitting and predicting for test data
pred_svm = svm.fit(x_train,y_train).predict(x_test)

#printing the report
print('Report: \n',classification_report(y_test,pred_svm))

#confusion matrix
print('confusion matrix:')
plot_confusion_matrix(svm,x_test,y_test,cmap='inferno')
plt.show()

#plotting the ROC curve
print('ROC curve :')
plot_roc_curve(svm,x_test,y_test)
plt.plot([0,1],[0,1])
plt.show()
```

```
Report: 
               precision    recall  f1-score   support

           0       0.64      0.94      0.76       373
           1       0.78      0.30      0.44       283

    accuracy                           0.66       656
   macro avg       0.71      0.62      0.60       656
weighted avg       0.70      0.66      0.62       656

confusion matrix:
```




```
ROC curve :
```



In [34]:

```
#checking hyper parameters
svm.get_params().keys()
```

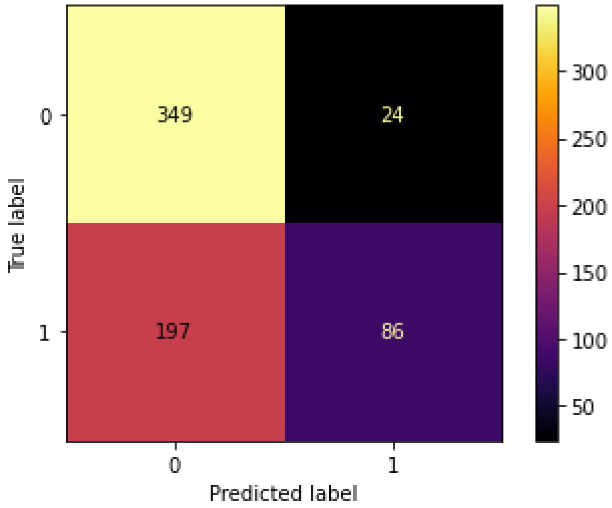
Out[34]:

```
dict_keys(['C', 'break_ties', 'cache_size', 'class_weight', 'coef0', 'decisi
on_function_shape', 'degree', 'gamma', 'kernel', 'max_iter', 'probability',
'random_state', 'shrinking', 'tol', 'verbose'])
```

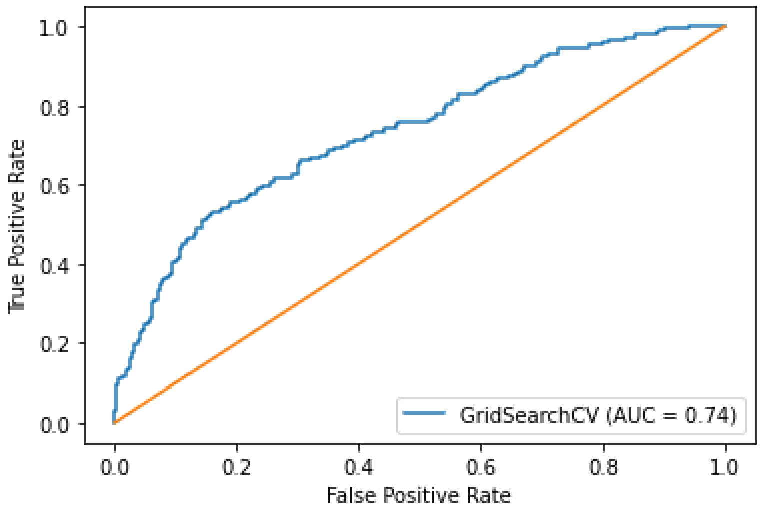
In [35]:

```
#hyper parameters
params = {'C':[0.001,0.01,0.1,1,10],
          'kernel':['linear', 'poly', 'rbf']}

#initializing the grid
grid_svm = GridSearchCV(estimator=svm,param_grid=params,cv=3,verbose=3,n_jobs=-1)

#fitting for test data
pred_svm = grid_svm.fit(x_train,y_train).predict(x_test)

#printing best score and parameters
print('Best score = {}\nBest params = {}'.format(grid_svm.best_score_,grid_svm.best_params_

#printing the report
print('Report: \n',classification_report(y_test,pred_svm))

#confusion matrix
print('confusion matrix:')
plot_confusion_matrix(grid_svm,x_test,y_test,cmap='inferno')
plt.show()

#plotting the ROC curve
print('ROC curve :')
plot_roc_curve(grid_svm,x_test,y_test)
plt.plot([0,1],[0,1])
plt.show()

#accuracy score
acc_svm = accuracy_score(y_test,pred_svm)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed:    3.4s finished

Best score = 0.6736649008346856
Best params = {'C': 1, 'kernel': 'rbf'}
Report:
               precision    recall  f1-score   support

           0       0.64      0.94      0.76       373
           1       0.78      0.30      0.44       283

    accuracy                           0.66       656
   macro avg       0.71      0.62      0.60       656
weighted avg       0.70      0.66      0.62       656

confusion matrix:
```




```
ROC curve :
```

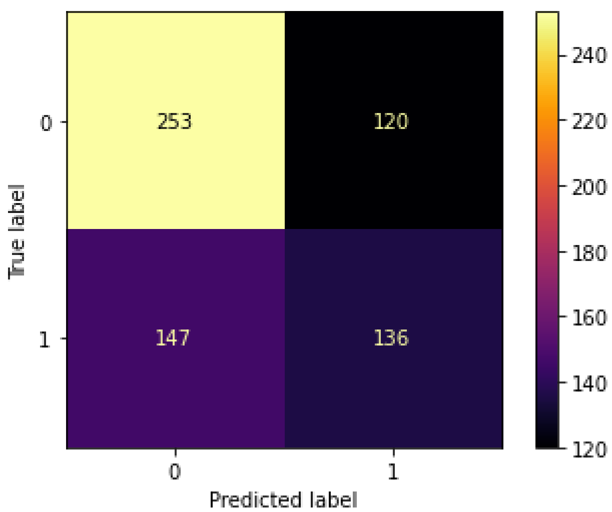


Decision tree model

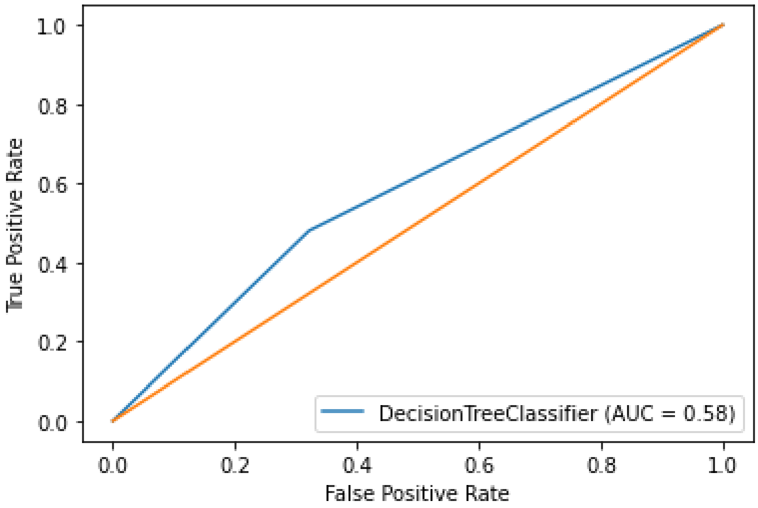
In [36]:

```
#importing libraries
from sklearn.tree import DecisionTreeClassifier

#initializing the model
deci = DecisionTreeClassifier()

#fitting and predicting for test data
pred_deci = deci.fit(x_train,y_train).predict(x_test)

#printing the report
print('Report: \n',classification_report(y_test,pred_deci))

#confusion matrix
print('confusion matrix:')
plot_confusion_matrix(deci,x_test,y_test,cmap='inferno')
plt.show()

#plotting the ROC curve
print('ROC curve :')
plot_roc_curve(deci,x_test,y_test)
plt.plot([0,1],[0,1])
plt.show()
```

```
Report: 
               precision    recall  f1-score   support

           0       0.63      0.68      0.65       373
           1       0.53      0.48      0.50       283

    accuracy                           0.59       656
   macro avg       0.58      0.58      0.58       656
weighted avg       0.59      0.59      0.59       656

confusion matrix:
```




```
ROC curve :
```

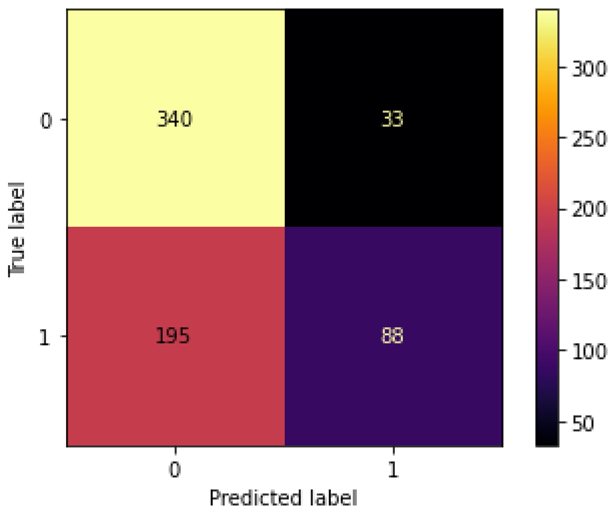


Random Forest Model

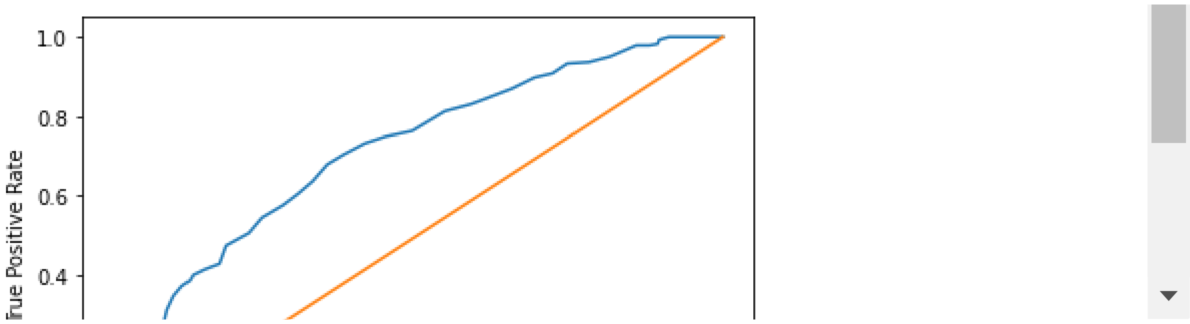
In [37]:

```
#importing libraries
from sklearn.ensemble import RandomForestClassifier

#initializing the model
rand = RandomForestClassifier()

#fitting and predicting for test data
pred_rand = rand.fit(x_train,y_train).predict(x_test)

#printing the report
print('Report: \n',classification_report(y_test,pred_rand))

#confusion matrix
print('confusion matrix:')
plot_confusion_matrix(rand,x_test,y_test,cmap='inferno')
plt.show()

#plotting the ROC curve
print('ROC curve :')
plot_roc_curve(rand,x_test,y_test)
plt.plot([0,1],[0,1])
plt.show()
```

```
Report: 
               precision    recall  f1-score   support

           0       0.64      0.91      0.75       373
           1       0.73      0.31      0.44       283

    accuracy                           0.65       656
   macro avg       0.68      0.61      0.59       656
weighted avg       0.68      0.65      0.61       656

confusion matrix:
```




```
ROC curve :
```



In [38]:

```
#checking for hyper parameters
rand.get_params().keys()
```

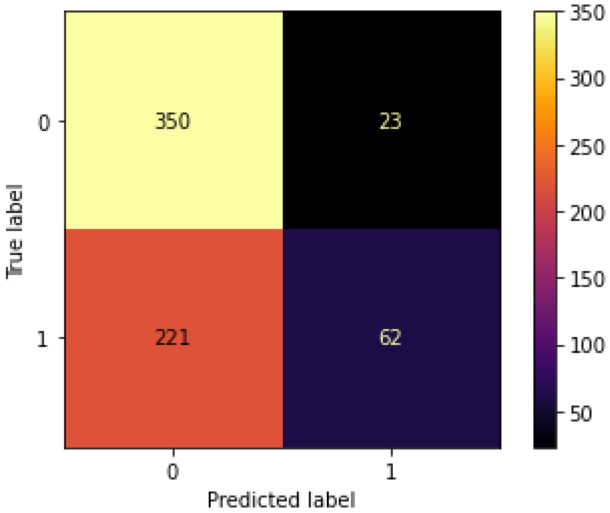
Out[38]:

```
dict_keys(['bootstrap', 'ccp_alpha', 'class_weight', 'criterion', 'max_dept
h', 'max_features', 'max_leaf_nodes', 'max_samples', 'min_impurity_decreas
e', 'min_impurity_split', 'min_samples_leaf', 'min_samples_split', 'min_weig
ht_fraction_leaf', 'n_estimators', 'n_jobs', 'oob_score', 'random_state', 'v
erbose', 'warm_start'])
```

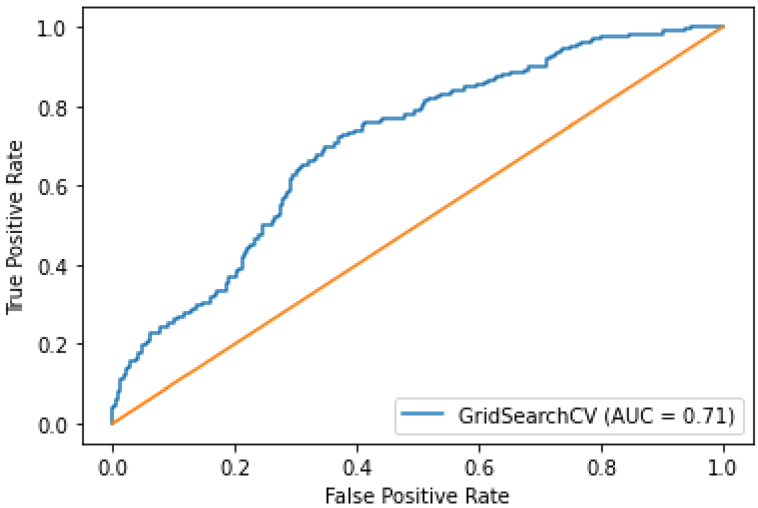
In [42]:

```
#hyper parameters
params = {'max_depth':[15,20,25],
          'min_samples_leaf':[10,20,30],
          'min_samples_split':[10,20,30],
          'n_estimators' : [200,250,300]
         }
#initializing the grid
grid_rand = GridSearchCV(estimator=rand,param_grid=params,cv=3,verbose=3,n_jobs=-1)

#fitting for test data
pred_rand = grid_rand.fit(x_train,y_train).predict(x_test)

#printing best score and parameters
print('Best score = {}\nBest params = {}'.format(grid_rand.best_score_,grid_rand.best_param

#printing the report
print('Report: \n',classification_report(y_test,pred_rand))

#confusion matrix
print('confusion matrix:')
plot_confusion_matrix(grid_rand,x_test,y_test,cmap='inferno')
plt.show()

#plotting the ROC curve
print('ROC curve :')
plot_roc_curve(grid_rand,x_test,y_test)
plt.plot([0,1],[0,1])
plt.show()

#accuracy score
acc_rand = accuracy_score(y_test,pred_rand)
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    6.0s
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:   33.2s
[Parallel(n_jobs=-1)]: Done 243 out of 243 | elapsed:  1.2min finished

Best score = 0.6694670088239173
Best params = {'max_depth': 15, 'min_samples_leaf': 10, 'min_samples_split':
20, 'n_estimators': 250}
Report:
               precision    recall  f1-score   support

           0       0.61      0.94      0.74       373
           1       0.73      0.22      0.34       283

    accuracy                           0.63       656
   macro avg       0.67      0.58      0.54       656
weighted avg       0.66      0.63      0.57       656

confusion matrix:
```




```
ROC curve :
```



Conclusion

Prediction is very diffcult since records are close to each other prediction becomes difficult

All the built models produce less accuracy

In [ ]: