Heart Disease Dataset

https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

Objective :

The objective of this notebook is to predict the presence of heart disease in the patient.

About Dataset

Context : This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease.

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

Reading the data

In [2]:
```
df = pd.read_csv("C:/Users/Viraj/OneDrive/Documents/Birla 2nd Sem/ML/Files/heart.csv
```

In [3]:
```
df.head() #top 5 rows
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

Attribute Information:

```
1.age
2.sex (1= Male, 0= Female)
3.chest pain type (4 values)
4.resting blood pressure
5.serum cholestoral in mg/dl
6.fasting blood sugar > 120 mg/dl
7.resting electrocardiographic results (values 0,1,2)
8.maximum heart rate achieved
9.exercise induced angina
10.oldpeak = ST depression induced by exercise relative to rest
11.the slope of the peak exercise ST segment
12.number of major vessels (0-3) colored by flourosopy
13.thal: 0 = normal; 1 = fixed defect; 2 = reversable defect
```

In [4]:
```
df.shape
```

Out[4]:
```
(1025, 14)
```

There are total 1025 rows and 14 columns

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

If we see the datatypes of the attributes, we can notice that all datatypes are integer datatypes except the one of oldpeak attribute which is float datatype.

In [6]:
```
df.oldpeak.unique()
```

Out[6]:
```
array([1. , 3.1, 2.6, 0. , 1.9, 4.4, 0.8, 3.2, 1.6, 3. , 0.7, 4.2, 1.5,
       2.2, 1.1, 0.3, 0.4, 0.6, 3.4, 2.8, 1.2, 2.9, 3.6, 1.4, 0.2, 2. ,
       5.6, 0.9, 1.8, 6.2, 4. , 2.5, 0.5, 0.1, 2.1, 2.4, 3.8, 2.3, 1.3,
       3.5])
```

In [7]:
```
df.isna().sum()
```

Out[7]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

There is no null value present.

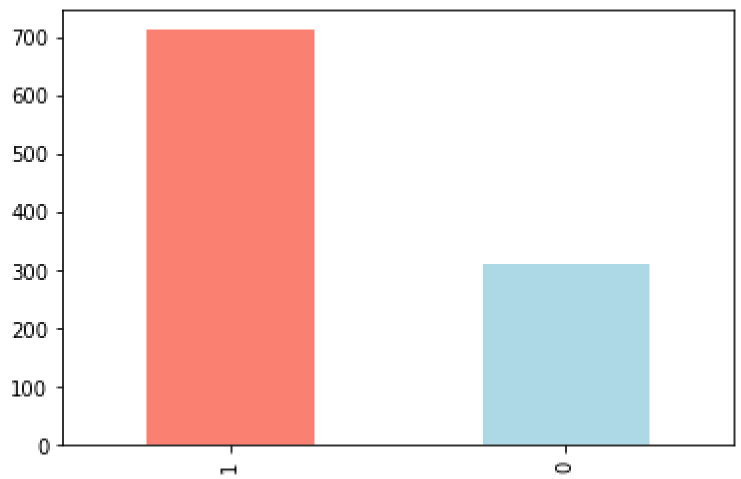
EDA

In [53]:
```
df['sex'].value_counts()
```

Out[53]:
```
1    713
0    312
Name: sex, dtype: int64
```

In [52]:
```
df['sex'].value_counts().plot(kind='bar', color=['salmon', 'lightblue'])
```

Out[52]:
```
<AxesSubplot:>
```



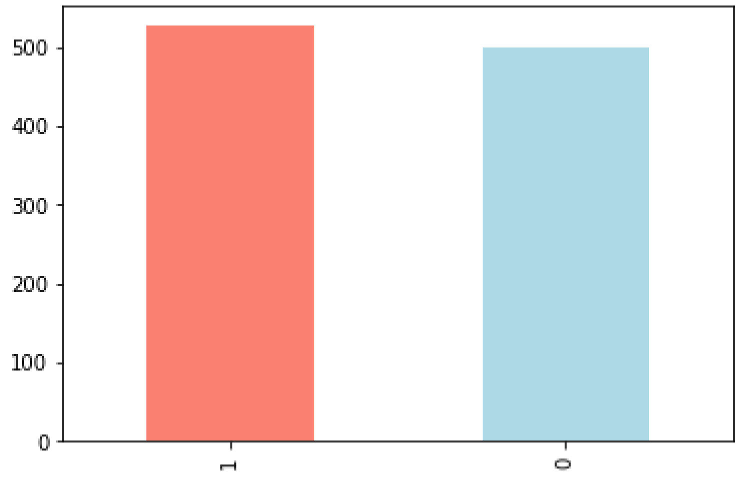
Out of 1025 records, 713 records are of males and 312 records are of females

In [48]:
```
df['target'].value_counts()
```

Out[48]:
```
1    526
0    499
Name: target, dtype: int64
```

In [50]:
```
df['target'].value_counts().plot(kind='bar', color=['salmon', 'lightblue'])
```

Out[50]:
```
<AxesSubplot:>
```

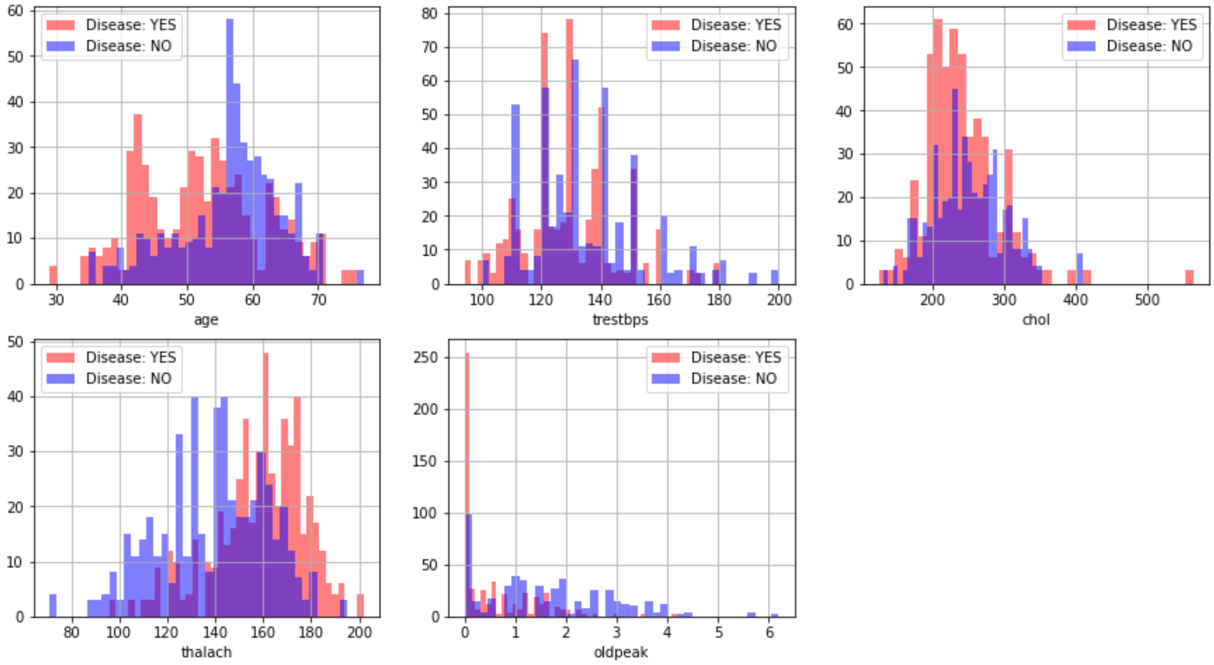


Out of 1025 records, 526 records are positive(having disease) and 499 records are negative(doesn't have disease)

In [9]:
```
cat_values = []
conti_values = []

for col in df.columns:
    if len(df[col].unique()) >= 10:
        conti_values.append(col)
    else:
        cat_values.append(col)

print("catageroy values: ", cat_values)
print("continous values: ", conti_values)
```

```
catageroy values:  ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal', 'target']
continous values:  ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
```

In [10]:
```
plt.figure(figsize=(15,8))

for i, col in enumerate(conti_values, 1):
    plt.subplot(2,3,i)
    df[df.target ==1][col].hist(bins=40, color='red', alpha=0.5,  label='Disease: YE
    df[df.target ==0][col].hist(bins=40, color='blue', alpha=0.5,  label='Disease: N
    plt.xlabel(col)
    plt.legend()
```



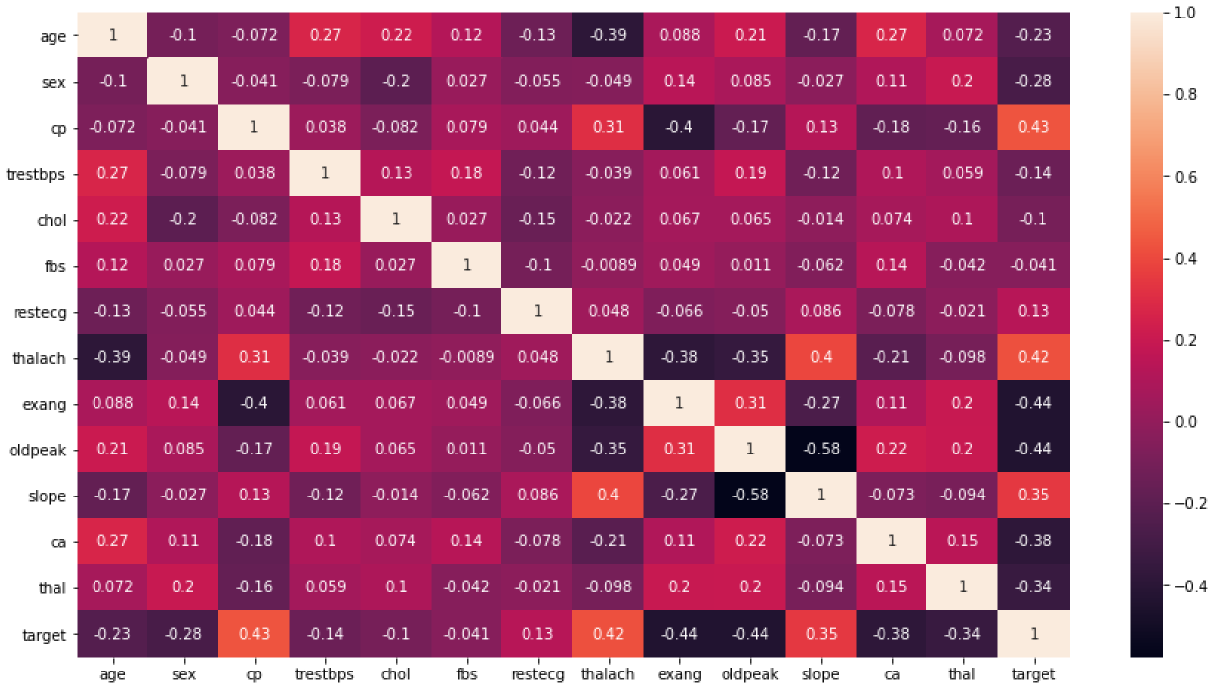

```
* trestbps[resting bp] anything above 130-140 is generally of concern
* chol[cholesterol] greater than 200 is of concern
* thalach People over 140 value are more likely to have heart disease
* oldpeak with value 0 are more than likely to have heart disease than
any other value
```

Checking Correlation using Heatmap

In [11]:

```python
x = df.corr()
plt.figure(figsize = (15,8))
sns.heatmap(x,annot = True)
```

Out[11]: <AxesSubplot:>



1. It is clearly visible that no column is a significant contributor among all the features.
2. So we are going to take all the features for the model evaluation.

In [12]:

```python
df.describe().T
```

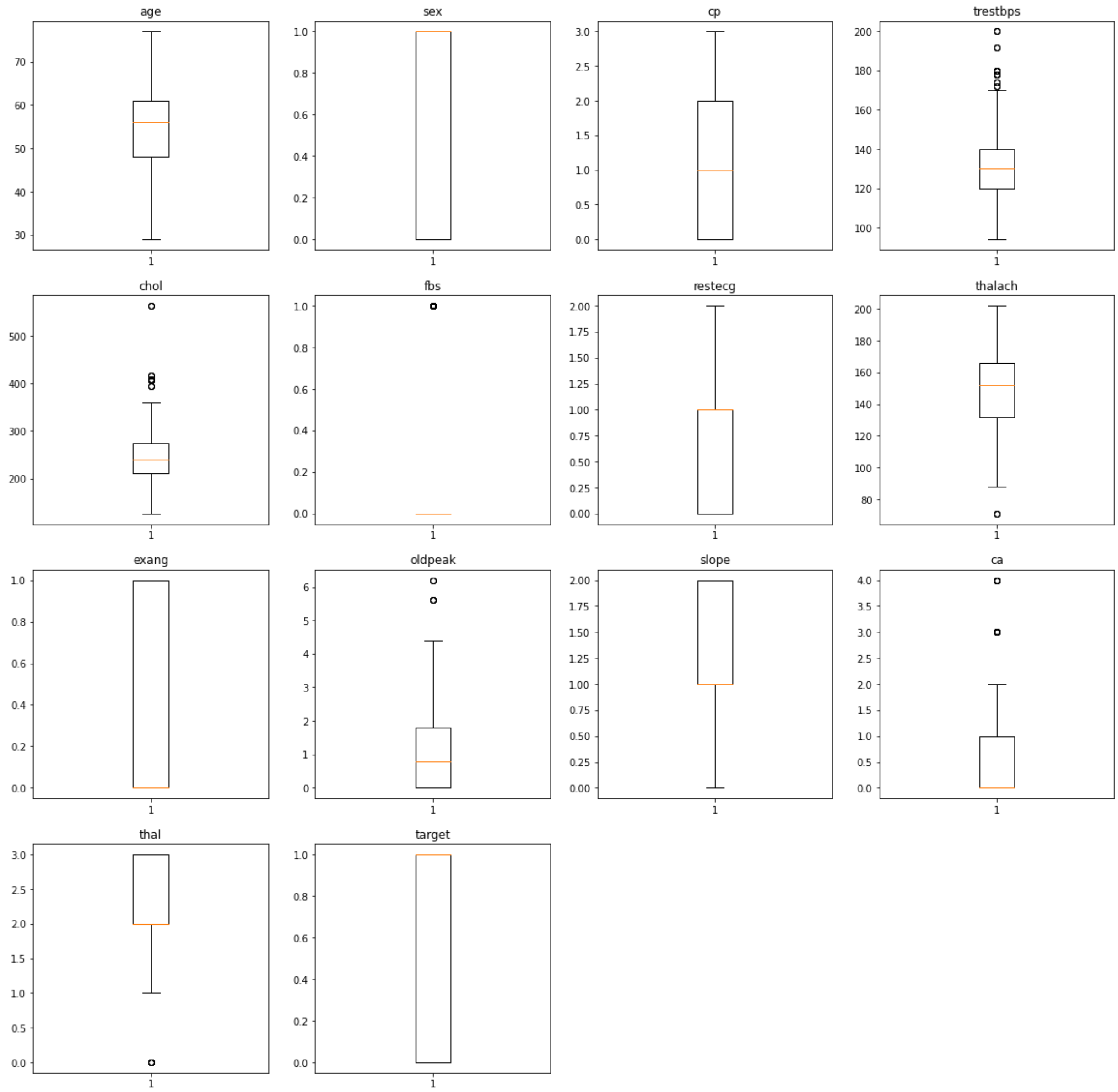
Out[12]:

	count	mean	std	min	25%	50%	75%	max
age	1025.0	54.434146	9.072290	29.0	48.0	56.0	61.0	77.0
sex	1025.0	0.695610	0.460373	0.0	0.0	1.0	1.0	1.0
cp	1025.0	0.942439	1.029641	0.0	0.0	1.0	2.0	3.0
trestbps	1025.0	131.611707	17.516718	94.0	120.0	130.0	140.0	200.0
chol	1025.0	246.000000	51.592510	126.0	211.0	240.0	275.0	564.0
fbs	1025.0	0.149268	0.356527	0.0	0.0	0.0	0.0	1.0
restecg	1025.0	0.529756	0.527878	0.0	0.0	1.0	1.0	2.0
thalach	1025.0	149.114146	23.005724	71.0	132.0	152.0	166.0	202.0
exang	1025.0	0.336585	0.472772	0.0	0.0	0.0	1.0	1.0
oldpeak	1025.0	1.071512	1.175053	0.0	0.0	0.8	1.8	6.2
slope	1025.0	1.385366	0.617755	0.0	1.0	1.0	2.0	2.0
ca	1025.0	0.754146	1.030798	0.0	0.0	0.0	1.0	4.0
thal	1025.0	2.323902	0.620660	0.0	2.0	2.0	3.0	3.0
target	1025.0	0.513171	0.500070	0.0	0.0	1.0	1.0	1.0

Outliers can be seen in various columns of the dataset. So moving onto the next method which is Boxplots for the better view of the outliers

In [13]:

```
x = 1
plt.figure(figsize = (20,20))

for i in df.columns:
    plt.subplot(4,4,x)
    plt.boxplot(df[i])
    plt.title(i)
    x = x+1
```



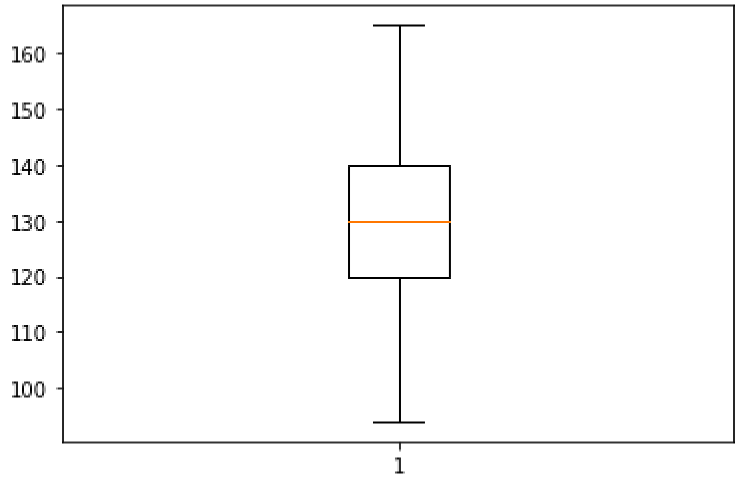
Maximum number of outliers can be seen in the column 'trestbps'.

Removing outliers from 'trestbps' column.

In [14]:
```
q1 = df['trestbps'].quantile(q = 0.25)
q3 = df["trestbps"].quantile(q = 0.75)
IQR = q3 - q1

IQR_lower_limit = int(q1 - (1.5*IQR))
IQR_upper_limit = int(q3 + (1.5*IQR))

print("Upper limit of IQR:",IQR_upper_limit)
print("Lower limit of IQR:",IQR_lower_limit)

cleaned_data = df[df["trestbps"]<IQR_upper_limit]
```

```
Upper limit of IQR: 170
Lower limit of IQR: 90
```

In [15]:
```
plt.boxplot(cleaned_data["trestbps"]);
```



Here we can see the outliers from trestbps column are removed

In [16]:
```
cleaned_data.shape
```

Out[16]:
```
(980, 14)
```

So now there are only 980 rows left in the dataset after clearing the outliers and the 14 columns as they were.

One Hot Encoding

In [17]:
```
cat_values.remove('target')
cleaned_data = pd.get_dummies(cleaned_data, columns=cat_values)
```

Train - Test Split

In [18]:
```
X = cleaned_data.drop(columns = 'target')
y = cleaned_data['target']
```

In [19]:
```
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_sta
```

Scaling

In [20]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[conti_values] = sc.fit_transform(X_train[conti_values])
X_test[conti_values] = sc.transform(X_test[conti_values])
```

Applying Logistic Regression

In [21]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

Out[21]:
```
▾ LogisticRegression

LogisticRegression()
```

In [22]:
```
y_pred_test = logreg.predict(X_test)
```

In [23]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [63]:
```
lr_acc_score=accuracy_score(y_test, y_pred_test)
lr_acc_score
```

Out[63]: 0.8673469387755102

Our model is **86.73 %** accurate by applying Logistic regeression

In [25]:
```
confusion_matrix(y_test, y_pred_test)
```

Out[25]:
```
array([[82, 13],
       [13, 88]], dtype=int64)
```

Model_prediction :

```
- Type_1 Error: 13 were diagnosed positive when they were not having
the disease.
- Type_2 Error: 13 were diagnosed negative when they actually having
the disease.
```

ROC_AUC_SCORE AND ROC_CURVE

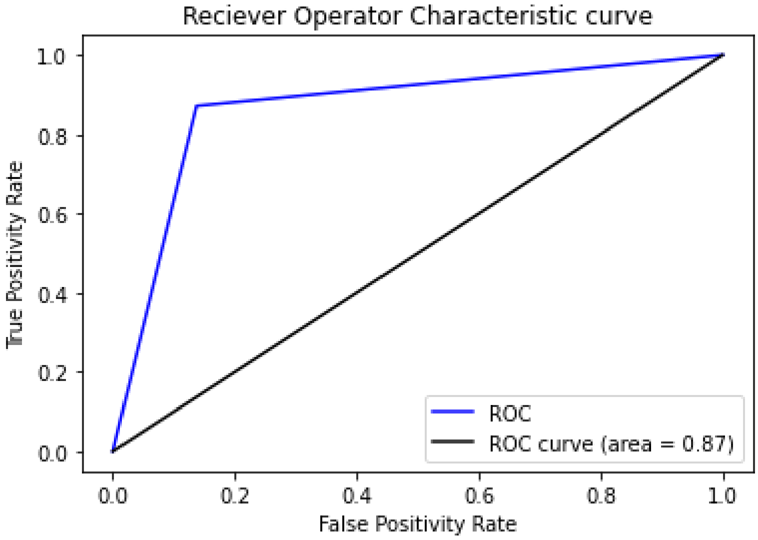
In [26]:
```
from sklearn.metrics import roc_curve, roc_auc_score
```

In [27]:
```
roc_score = roc_auc_score(y_test,y_pred_test)
roc_score
```

Out[27]: 0.8672225117248566

In [28]:
```
tpr, fpr, thresholds = roc_curve(y_test, y_pred_test)
```

In [29]:
```
plt.plot(tpr, fpr, color = 'blue', label = 'ROC')
plt.plot([0,1],[0,1],color = 'black', label = 'ROC curve (area = %0.2f)'% roc_score)
plt.xlabel("False Positivity Rate")
plt.ylabel("True Positivity Rate")
plt.title("Reciever Operator Characteristic curve")
plt.legend()
plt.show()
```



AUC : 0.87

Applying other Machine Learning Algorithms

```
In [65]: from sklearn.naive_bayes import GaussianNB
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.tree import DecisionTreeClassifier
```

```
In [66]: m2 = 'Naive Bayes'
         nb = GaussianNB()
         nb.fit(X_train,y_train)
         nbpred = nb.predict(X_test)
         nb_acc_score = accuracy_score(y_test, nbpred)
         print(nb_acc_score)
```

```
0.8418367346938775
```

Our model is **84.18 %** accurate by applying Naive Bayes

```
In [67]: m3 = 'Random Forest Classfier'
         rf = RandomForestClassifier(n_estimators=20, random_state=12,max_depth=5)
         rf.fit(X_train,y_train)
         rf_predicted = rf.predict(X_test)
         rf_acc_score = accuracy_score(y_test, rf_predicted)
         print(rf_acc_score)
```

```
0.9285714285714286
```

Our model is **92.86 %** accurate by applying Random Forest Classfier

```
In [74]: m4= 'K-Neighbors Classifier'
         knn = KNeighborsClassifier(n_neighbors=10)
         knn.fit(X_train, y_train)
         knn_predicted = knn.predict(X_test)
         knn_acc_score = accuracy_score(y_test, knn_predicted)
         print(knn_acc_score)
```

```
0.8826530612244898
```

Our model is **88.27 %** accurate by applying K-Neighbors Classifier

```
In [75]: m5 = 'Decision Tree Classifier'
         dt = DecisionTreeClassifier(criterion = 'entropy',random_state=0,max_depth = 6)
         dt.fit(X_train, y_train)
         dt_predicted = dt.predict(X_test)
         dt_acc_score = accuracy_score(y_test, dt_predicted)
         print(dt_acc_score)
```

```
0.9387755102040817
```

Our model is **93.88 %** accurate by applying Decision Tree Classifier

```
In [73]: model_ev = pd.DataFrame({'Model': ['Logistic Regression','Naive Bayes','Random Fores
                             'K-Nearest Neighbour','Decision Tree'], 'Accuracy': [lr_acc_scor
                             nb_acc_score*100,rf_acc_score*100,knn_acc_score*100,dt_acc_score
         model_ev
```

Out[73]:

	Model	Accuracy
0	Logistic Regression	86.734694
1	Naive Bayes	84.183673
2	Random Forest	92.857143
3	K-Nearest Neighbour	88.265306
4	Decision Tree	93.877551

Conclusion

Over all the Machine Learning Algorithms, **Decision Tree(93.88 %)** and **Random Forest(92.86 %)** Algorithm gives us the best Accuracy.