# Heart Attack Prediction

*The main aim of this project is to predict the chances of heart attack by analysing some medical properties like blood pressure,chest pain,sugar,cholestoral,maximum heart rate achieved*

```
#Importing all the necessary librarys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Dropout
from sklearn.metrics import classification_report,accuracy_score,roc_curve,roc_auc_score
```

```
#Loading dataset and Reading
df = pd.read_csv('heart.csv')
df.head()
```

| | age | sex | cp | trtbps | chol | fbs | restecg | thalachh | exng | oldpeak | slp | caa | tha |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | |

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trtbps    303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalachh  303 non-null    int64
 8   exng      303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64
 11  caa       303 non-null    int64
 12  thall     303 non-null    int64
 13  output    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

# Data Preprocessing

```
df.nunique()
```

```
age         41
sex          2
cp           4
trtbps      49
```

```
chol        152
fbs           2
restecg       3
thalachh     91
exng          2
oldpeak      40
slp           3
caa           5
thall         4
output        2
dtype: int64
```

We have 5 numerical feature and 8 categorical feature in our data set

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trtbps    303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalachh  303 non-null    int64  
 8   exng      303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64  
 11  caa       303 non-null    int64  
 12  thall     303 non-null    int64  
 13  output    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

All our categorical nature column has integer type of data type so first we convert into an object data type

```
cat_columns = ['sex', 'cp', 'fbs', 'restecg', 'exng', 'slp', 'caa', 'thall', 'output']
num_columns = ['age', 'trtbps', 'oldpeak', 'chol', 'thalachh']
df[cat_columns] = df[cat_columns].astype(str)
```

```
df.describe()
```

| | age | trtbps | chol | thalachh | oldpeak |
|---|---|---|---|---|---|
| **count** | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 |
| **mean** | 54.366337 | 131.623762 | 246.264026 | 149.646865 | 1.039604 |
| **std** | 9.082101 | 17.538143 | 51.830751 | 22.905161 | 1.161075 |
| **min** | 29.000000 | 94.000000 | 126.000000 | 71.000000 | 0.000000 |
| **25%** | 47.500000 | 120.000000 | 211.000000 | 133.500000 | 0.000000 |
| **50%** | 55.000000 | 130.000000 | 240.000000 | 153.000000 | 0.800000 |
| **75%** | 61.000000 | 140.000000 | 274.500000 | 166.000000 | 1.600000 |
| **max** | 77.000000 | 200.000000 | 564.000000 | 202.000000 | 6.200000 |

Median and mean values of all the numerical features are comparable. Therefore, I don't think there are any outliers in our numerical features

## Handling Missing values

```
df.isnull().sum()
```

```
df.isnull().sum()
```

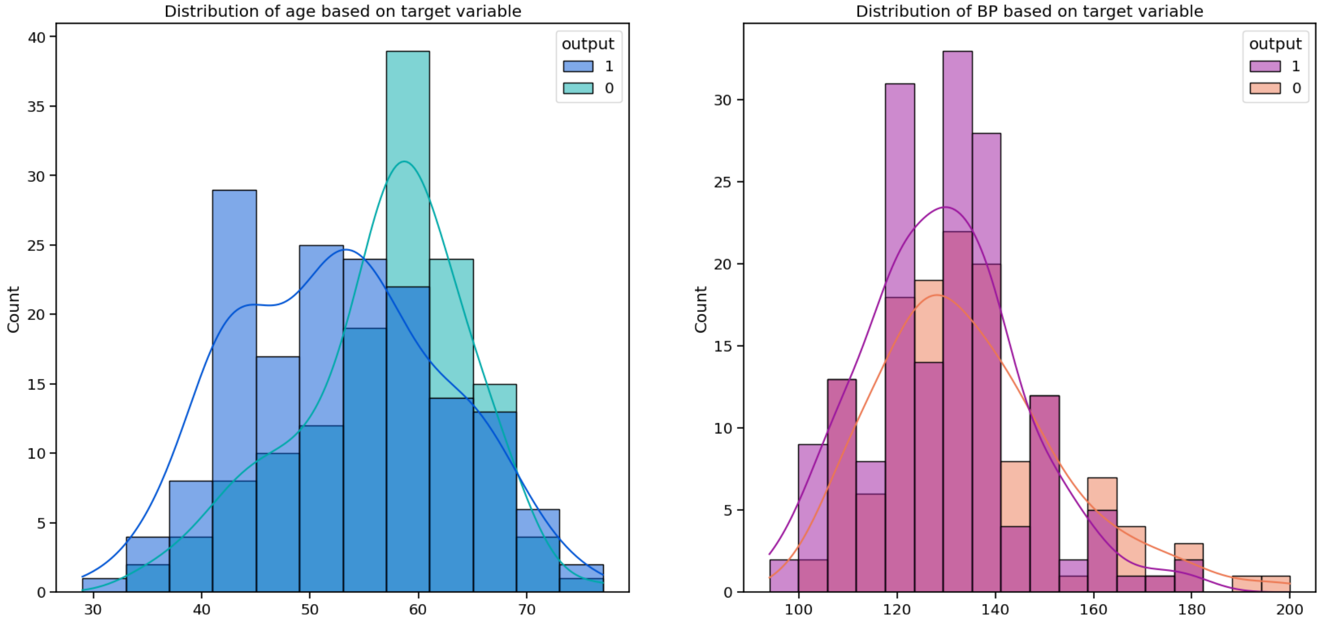
```
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
output      0
dtype: int64
```

As there is no null value our data is clean for preprocessing

# Visualization

Let us check how our numerical features change based on target columns

```
sns.set_context('notebook',font_scale=1.2)
fig, ax=plt.subplots(2,2,figsize=(20,20))
plt.title('Distribution of various feature based on target ')
a1 = sns.histplot(x ='age', data= df, hue= 'output',kde=True, ax=ax[0, 0], palette='winter')
a1.set(xlabel = 'Age', title= 'Distribution of age based on target variable')

a2 = sns.histplot(x ='trtbps', data= df, hue= 'output', kde= True, ax= ax[0, 1], palette='plasma')
a2.set(xlabel = 'Resting blood pressure (in mm Hg)', title= 'Distribution of BP based on target variable')

a3 = sns.histplot(x ='chol', data= df, hue= 'output', kde= True, ax= ax[1, 0], palette='winter')
a3.set(xlabel = 'Cholestoral in mg/dl', title= 'Distribution of Cholestrol based on target variable')

a4 = sns.histplot(x ='thalachh', data=df, hue= 'output', kde= True, ax= ax[1, 1], palette='plasma')
a4.set(xlabel = 'Max Heart Rate Achieved', title= 'Distribution of maximum heart rate achieved based on target variable'

plt.show()
```




We can clearly seen that the person who had maximum heart rate achieved having a great chance of heart attack

Distribution of Cholestrol based on target variable

Distribution of maximum heart rate achieved based on target variable

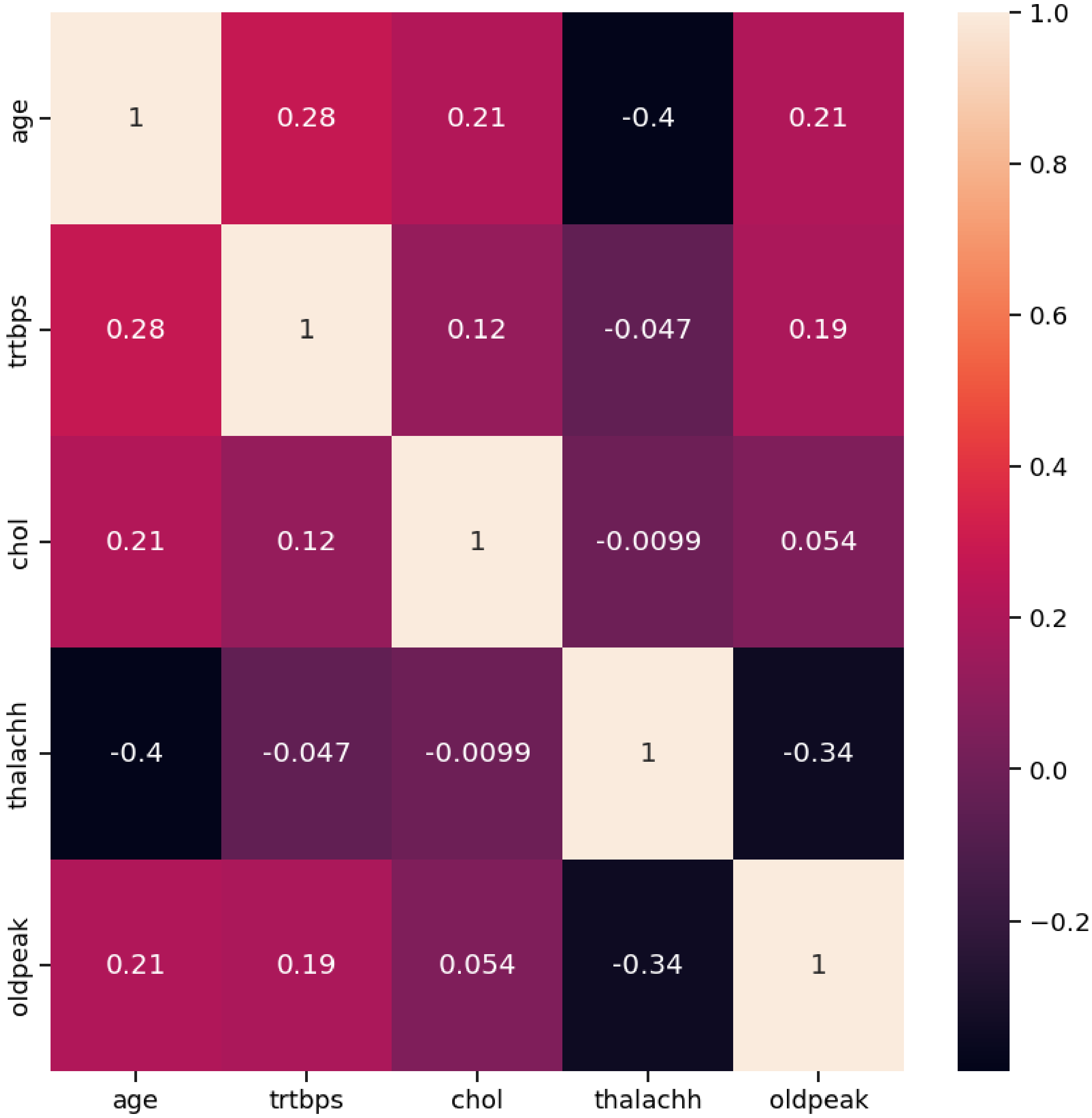
```
# To check outlier is present or not

sns.set_context('notebook', font_scale= 1.2)
fig, ax = plt.subplots(2, 2, figsize = (20, 10))

plt.suptitle('Boxplot of various features based on target variable', fontsize = 20)

b1 = sns.boxplot(x ='age', data=df, ax= ax[0, 0], color = '#40bf80')
b1.set(xlabel = 'Age')

b2 = sns.boxplot(x ='trtbps', data=df, ax= ax[0, 1], color='#40bf80')
b2.set(xlabel = 'Resting blood pressure (in mm Hg)')

b3 = sns.boxplot(x ='chol', data=df, ax= ax[1, 0], color= '#40bf80')
b3.set(xlabel = 'Cholestoral in mg/dl')

b4 = sns.boxplot(x ='thalachh', data=df, ax= ax[1, 1], color = '#40bf80')
b4.set(xlabel = 'Max Heart Rate Achieved')

plt.show()
```

Boxplot of various features based on target variable

As we clearly seen that there are outlier present in cholestoral and blood pressure column.

```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot=True)
```

```
<Axes: >
```



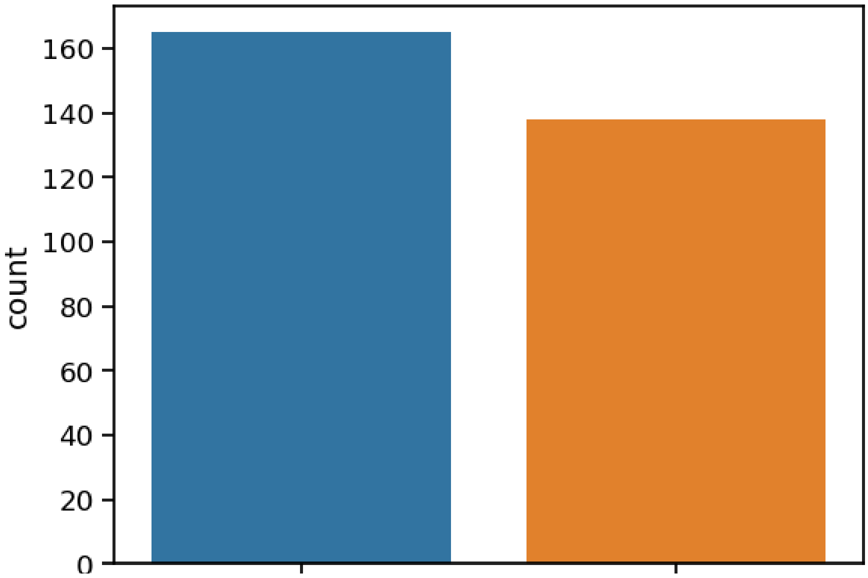

As with the help of heatmap we see that there is no correlation in our numerical columns

```
df['output'].value_counts()
```

```
1    165
0    138
Name: output, dtype: int64
```

```
sns.countplot(data=df,x=df['output'])
```

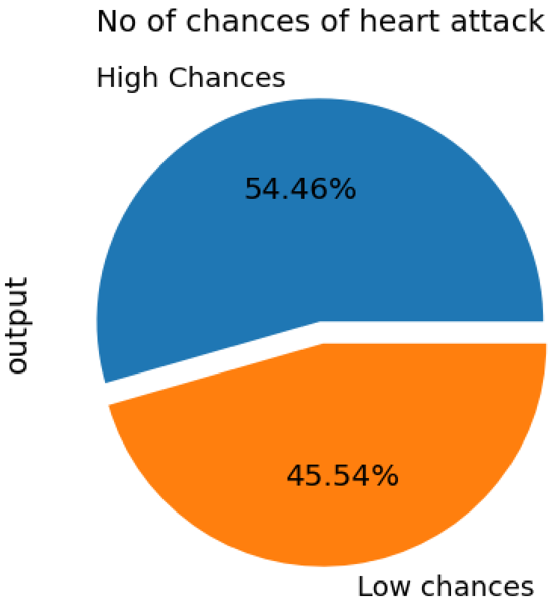
```
<Axes: xlabel='output', ylabel='count'>
```




Hence with above countplot we can clearly seen that our target column is balanced in nature.

```
plt.title('No of chances of heart attack')
df['output'].value_counts().plot.pie(autopct='%1.2f%%',labels=['High Chances','Low chances'],explode=(0,0.1))
```

```
<Axes: title={'center': 'No of chances of heart attack'}, ylabel='output'>
```




```
df.head()
```

| | age | sex | cp | trtbps | chol | fbs | restecg | thalachh | exng | oldpeak | slp | caa | thall | output |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | 1 |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 | 1 |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 | 1 |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | 2 | 1 |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | 2 | 1 |

```
cat_columns = ['sex', 'cp', 'fbs', 'restecg', 'exng', 'slp', 'caa', 'thall', 'output']
num_columns = ['age', 'trtbps', 'oldpeak', 'chol', 'thalachh']
df[cat_columns] = df[cat_columns].astype(float)
```

```
#To seperate x and y
x=df.iloc[::-1].values
```

```
x
```

```
array([[57.,  0.,  1., ...,  1.,  2.,  0.],
       [57.,  1.,  0., ...,  1.,  3.,  0.],
       [68.,  1.,  0., ...,  2.,  3.,  0.],
       ...,
       [41.,  0.,  1., ...,  0.,  2.,  1.],
       [37.,  1.,  2., ...,  0.,  2.,  1.],
       [63.,  1.,  3., ...,  0.,  1.,  1.]])
```

```
y=df['output'].values
y
```

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

```
#Split the train and testing part
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.20,random_state=1)
```

## Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
xtrain=sc.fit_transform(xtrain)
xtest=sc.transform(xtest)
```

## Early Stopping

```
#Early stopping
from tensorflow.keras.callbacks import EarlyStopping
early_stop=EarlyStopping(monitor="val_loss",mode="min",verbose=1,patience=10)
```

## Model Building

```
ann=Sequential()
#Hidden Layer
ann.add(Dense(units=100,activation="relu"))
ann.add(Dropout(0.20))
ann.add(Dense(units=50,activation="relu"))
#ann.add(Dense(units=100,activation="relu"))
#ann.add(Dropout(0.20))
#ann.add(Dense(units=50,activation="relu"))

ann.add(Dense(units=1,activation="sigmoid"))
```

```
ann.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"])
```

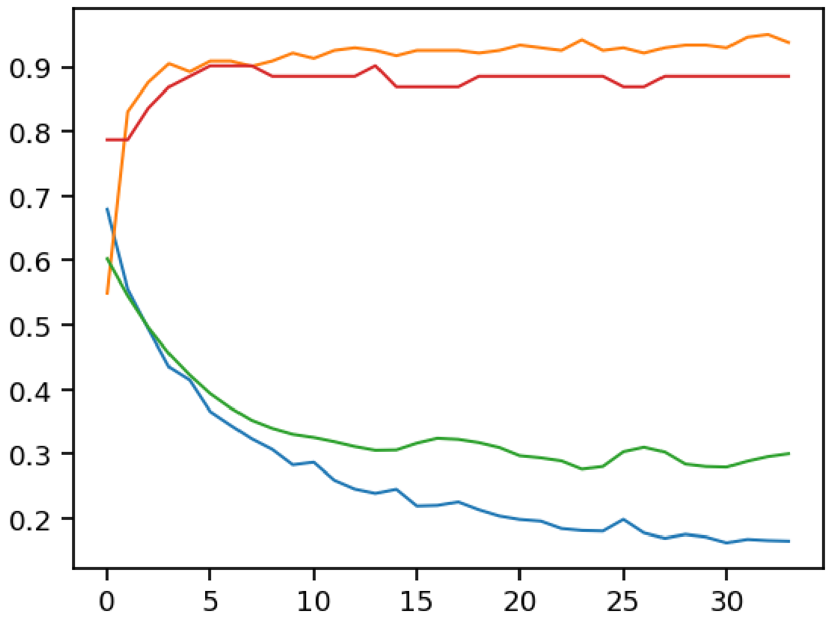
```
ann.fit(xtrain,ytrain,validation_data=(xtest,ytest),verbose=1,callbacks=[early_stop],batch_size=40,epochs=100)
```

```
=============================] - 0s 9ms/step - loss: 0.3444 - accuracy: 0.9091 - val_loss: 0.3712 - val_accuracy:
8/100
=============================] - 0s 9ms/step - loss: 0.3239 - accuracy: 0.9008 - val_loss: 0.3525 - val_accuracy:
9/100
=============================] - 0s 9ms/step - loss: 0.3074 - accuracy: 0.9091 - val_loss: 0.3399 - val_accuracy:
10/100
=============================] - 0s 8ms/step - loss: 0.2833 - accuracy: 0.9215 - val_loss: 0.3307 - val_accuracy:
11/100
=============================] - 0s 13ms/step - loss: 0.2873 - accuracy: 0.9132 - val_loss: 0.3258 - val_accuracy
12/100
=============================] - 0s 12ms/step - loss: 0.2596 - accuracy: 0.9256 - val_loss: 0.3193 - val_accuracy
13/100
=============================] - 0s 11ms/step - loss: 0.2459 - accuracy: 0.9298 - val_loss: 0.3117 - val_accuracy
14/100
=============================] - 0s 9ms/step - loss: 0.2392 - accuracy: 0.9256 - val_loss: 0.3059 - val_accuracy:
15/100
=============================] - 0s 15ms/step - loss: 0.2457 - accuracy: 0.9174 - val_loss: 0.3064 - val_accuracy
16/100
=============================] - 0s 10ms/step - loss: 0.2196 - accuracy: 0.9256 - val_loss: 0.3170 - val_accuracy
17/100
=============================] - 0s 10ms/step - loss: 0.2205 - accuracy: 0.9256 - val_loss: 0.3246 - val_accuracy
18/100
=============================] - 0s 11ms/step - loss: 0.2259 - accuracy: 0.9256 - val_loss: 0.3229 - val_accuracy
19/100
=============================] - 0s 9ms/step - loss: 0.2137 - accuracy: 0.9215 - val_loss: 0.3179 - val_accuracy:
20/100
=============================] - 0s 9ms/step - loss: 0.2040 - accuracy: 0.9256 - val_loss: 0.3102 - val_accuracy:
21/100
=============================] - 0s 9ms/step - loss: 0.1986 - accuracy: 0.9339 - val_loss: 0.2973 - val_accuracy:
22/100
=============================] - 0s 13ms/step - loss: 0.1961 - accuracy: 0.9298 - val_loss: 0.2941 - val_accuracy
23/100
=============================] - 0s 9ms/step - loss: 0.1846 - accuracy: 0.9256 - val_loss: 0.2894 - val_accuracy:
24/100
=============================] - 0s 11ms/step - loss: 0.1816 - accuracy: 0.9421 - val_loss: 0.2766 - val_accuracy
25/100
=============================] - 0s 8ms/step - loss: 0.1808 - accuracy: 0.9256 - val_loss: 0.2808 - val_accuracy:
26/100
=============================] - 0s 14ms/step - loss: 0.1988 - accuracy: 0.9298 - val_loss: 0.3036 - val_accuracy
27/100
=============================] - 0s 10ms/step - loss: 0.1779 - accuracy: 0.9215 - val_loss: 0.3107 - val_accuracy
28/100
=============================] - 0s 9ms/step - loss: 0.1697 - accuracy: 0.9298 - val_loss: 0.3031 - val_accuracy:
29/100
=============================] - 0s 11ms/step - loss: 0.1757 - accuracy: 0.9339 - val_loss: 0.2844 - val_accuracy
30/100
=============================] - 0s 8ms/step - loss: 0.1715 - accuracy: 0.9339 - val_loss: 0.2806 - val_accuracy:
31/100
=============================] - 0s 9ms/step - loss: 0.1627 - accuracy: 0.9298 - val_loss: 0.2797 - val_accuracy:
32/100
=============================] - 0s 11ms/step - loss: 0.1680 - accuracy: 0.9463 - val_loss: 0.2889 - val_accuracy
33/100
=============================] - 0s 11ms/step - loss: 0.1663 - accuracy: 0.9504 - val_loss: 0.2960 - val_accuracy
34/100
=============================] - 0s 9ms/step - loss: 0.1654 - accuracy: 0.9380 - val_loss: 0.3005 - val_accuracy:
34: early stopping
.callbacks.History at 0x7f30b2bf9b50>
```

```
ann.history.history  #data points of loss with respect to epochs
```

```
        0.27663564682006836,
        0.2807615399360657,
        0.3036169111728668,
        0.3106541931629181,
        0.30313342809677124,
        0.2843846380710602,
        0.28060442209243774,
        0.2797313630580902,
        0.28886181116104126,
        0.29600051045417786,
        0.3004968464374542],
       'val_accuracy': [0.7868852615356445,
        0.7868852615356445,
        0.8360655903816223,
        0.868852436542511,
        0.8852459192276001,
        0.9016393423080444,
        0.9016393423080444,
        0.9016393423080444,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.9016393423080444,
        0.868852436542511,
        0.868852436542511,
        0.868852436542511,
        0.868852436542511,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.868852436542511,
        0.868852436542511,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001,
        0.8852459192276001]}
```

```
plt.plot(pd.DataFrame(ann.history.history))
plt.show()
```




```
ypred=ann.predict(xtest)
```

```
2/2 [==============================] - 0s 8ms/step
```

```
ypred=np.where(ypred<0.5,0,1)
ypred
```

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

```
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

         0.0       0.83      0.97      0.89        30
         1.0       0.96      0.81      0.88        31

    accuracy                           0.89        61
   macro avg       0.90      0.89      0.88        61
weighted avg       0.90      0.89      0.88        61
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(ytest,ypred))
```

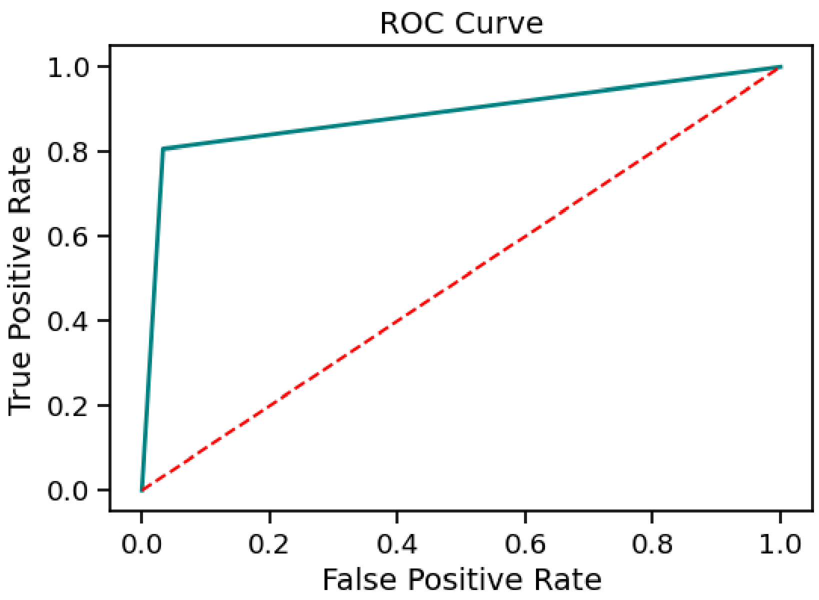
```
[[29  1]
 [ 6 25]]
```

As you can clearly seen by our matrix that our model classification is good

## Roc Auc Curve

```
from sklearn.metrics import roc_curve,roc_auc_score
```

```
fpr, tpr, thresholds = roc_curve(ytest,ypred)

plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, linewidth=2, color= 'teal')
plt.plot([0,1], [0,1], 'r--' )
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.show()
```




```
print(roc_auc_score(ytest,ypred))
```

```
0.8865591397849462
```

Here you can see that our roc auc curve is closest to 1,Hence our model is performing good while doing prediction whether the person has heart attack chances or not.

✓ 0s completed at 12:33