



Learn the AI/ML Project on:  
**Heart Disease Diagnosis**



Find all projects at [@ github.com/Masterx-AI](https://github.com/Masterx-AI)

# ★ AI / ML Project - Heart Disease Diagnosis ★

**Domain: Healthcare**



## Description:

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

## Acknowledgements:

This dataset has been referred from Kaggle: \ <https://www.kaggle.com/ronitf/heart-disease-uci>

## Objective:

- Understand the Dataset & cleanup (if required).
- Build classification model to predict whether the a person will have heart disease or not.
- Also fine-tune the hyperparameters & compare the evaluation metrics of various classification algorithms.

## Strategic Plan of Action:

We aim to solve the problem statement by creating a plan of action, Here are some of the necessary steps:

1. Data Exploration
2. Exploratory Data Analysis (EDA)
3. Data Pre-processing
4. Data Manipulation
5. Feature Selection/Extraction
6. Predictive Modelling
7. Project Outcomes & Conclusion

## 1. Data Exploration

In [3]:

```
#Importing the basic libraries

import os
import math
import scipy
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import tree
from scipy.stats import randint
from scipy.stats import loguniform
from IPython.display import display
```

```

from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import RFE
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold

from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, \
f1_score, roc_auc_score, roc_curve, precision_score, recall_score

import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10, 6]

import warnings
warnings.filterwarnings('ignore')

```

In [6]:

```

#Importing the dataset

df = pd.read_csv('Heart.csv')
#df.drop(['ID', 'year'], axis=1, inplace=True)

target = 'target'
labels = ['Heart Disease', 'Not Heart Disease']
features = [i for i in df.columns.values if i not in [target]]

original_df = df.copy(deep=True)
display(df.head())

print('\n\nThe Dataset consists of {} features & {} samples.'.format(df.shape[1], df.shape[0]))

```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
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

Inference: The Dataset consists of 14 features & 303 samples.

In [7]:

```

#Checking the dtypes of all the columns

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   trestbps   303 non-null    int64  
 4   chol       303 non-null    int64  
 5   fbs        303 non-null    int64  
 6   restecg    303 non-null    int64  
 7   thalach    303 non-null    int64  
 8   exang      303 non-null    int64  
 9   oldpeak    303 non-null    float64 
 10  slope      303 non-null    int64  
 11  ca         303 non-null    int64  
 12  thal       303 non-null    int64  
 13  target     303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB

```

In [8]:

```

#Checking number of unique rows in each feature

df.nunique().sort_values()

```

sex	2
-----	---

```
Out[8]: fbs      2
exang     2
target    2
restecg   3
slope     3
cp        4
thal      4
ca        5
oldpeak   40
age       41
trestbps  49
thalach   91
chol      152
dtype: int64
```

```
In [9]: #Checking number of unique rows in each feature
```

```
nu = df[features].nunique().sort_values()
nf = []; cf = []; nnf = 0; ncf = 0; #numerical & categorical features

for i in range(df[features].shape[1]):
    if nu.values[i]<=15:cf.append(nu.index[i])
    else: nf.append(nu.index[i])

print('\nInference: The Datset has {} numerical & {} categorical features.'.format(len(nf),len(cf)))
```

Inference: The Datset has 5 numerical & 8 categorical features.

```
In [10]: #Checking the stats of all the columns
```

```
display(df.describe())
```

	age	sex	cp	trestbps	chol	fb	restecg	thalach	exang	oldpeak	slope	ca
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000

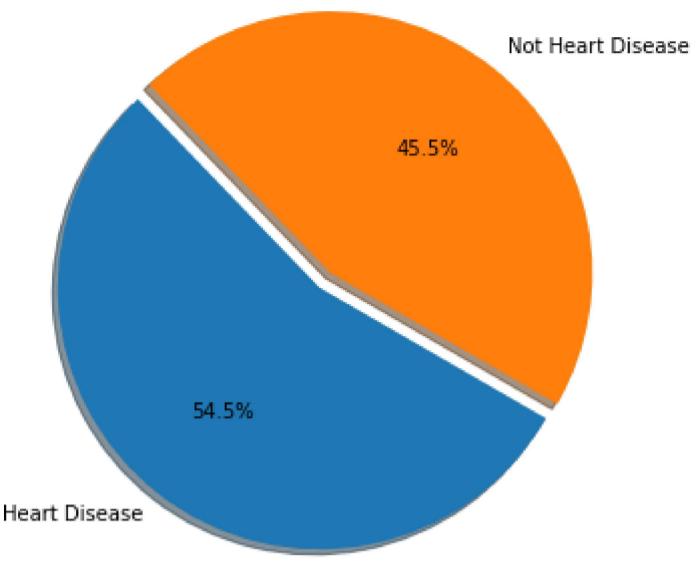
Inference: The stats seem to be fine, let us gain more understanding by visualising the dataset.

## 2. Exploratory Data Analysis (EDA)

```
In [11]: #Let us first analyze the distribution of the target variable
```

```
MAP={}
for e, i in enumerate(df[target].unique()):
    MAP[i]=labels[e]
#MAP={0:'Not-Survived',1:'Survived'}
df1 = df.copy()
df1[target]=df1[target].map(MAP)
explode=np.zeros(len(labels))
explode[-1]=0.07
print('Target Variable Distribution'.center(55))
plt.pie(df1[target].value_counts(), labels=df1[target].value_counts().index, counterclock=False, shadow=True,
         explode=explode, autopct='%1.1f%%', radius=1, startangle=-30)
plt.show()
```

Target Variable Distribution



**Inference:** The Target Variable seems to be slightly imbalanced! Hence we shall try to perform data augmentation.

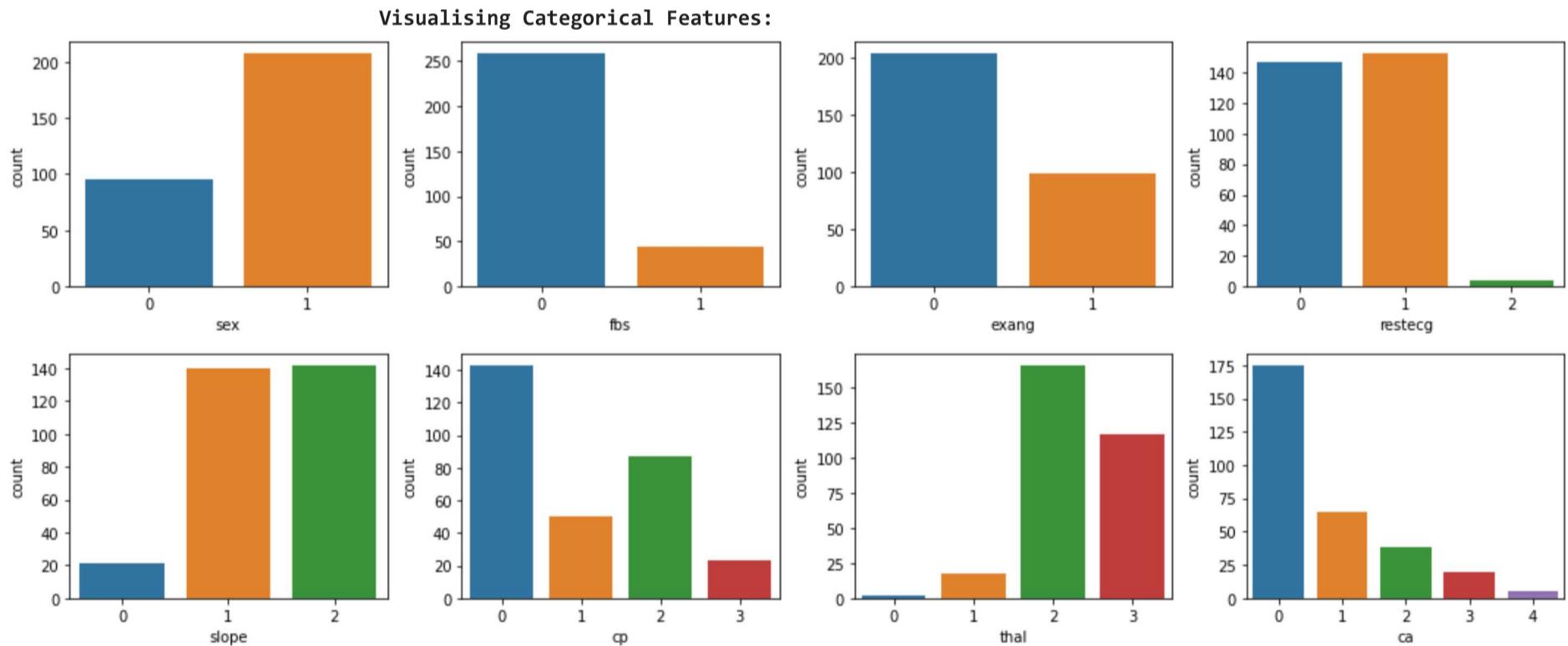
In [12]:

```
#Visualising the categorical features

print('\033[1mVisualising Categorical Features:'.center(100))

n=4
plt.figure(figsize=[15,3*math.ceil(len(cf)/n)])

for i in range(len(cf)):
    if df[cf[i]].nunique()<=15:
        plt.subplot(math.ceil(len(cf)/n),n,i+1)
        sns.countplot(df[cf[i]])
    else:
        # plt.subplot(2,2,i)
        # sns.countplot(df[cf[i]])
plt.tight_layout()
plt.show()
```



**Inference:** Visualizing the categorical features reveal lot of information about the dataset.

In [15]:

```
#Understanding the feature set

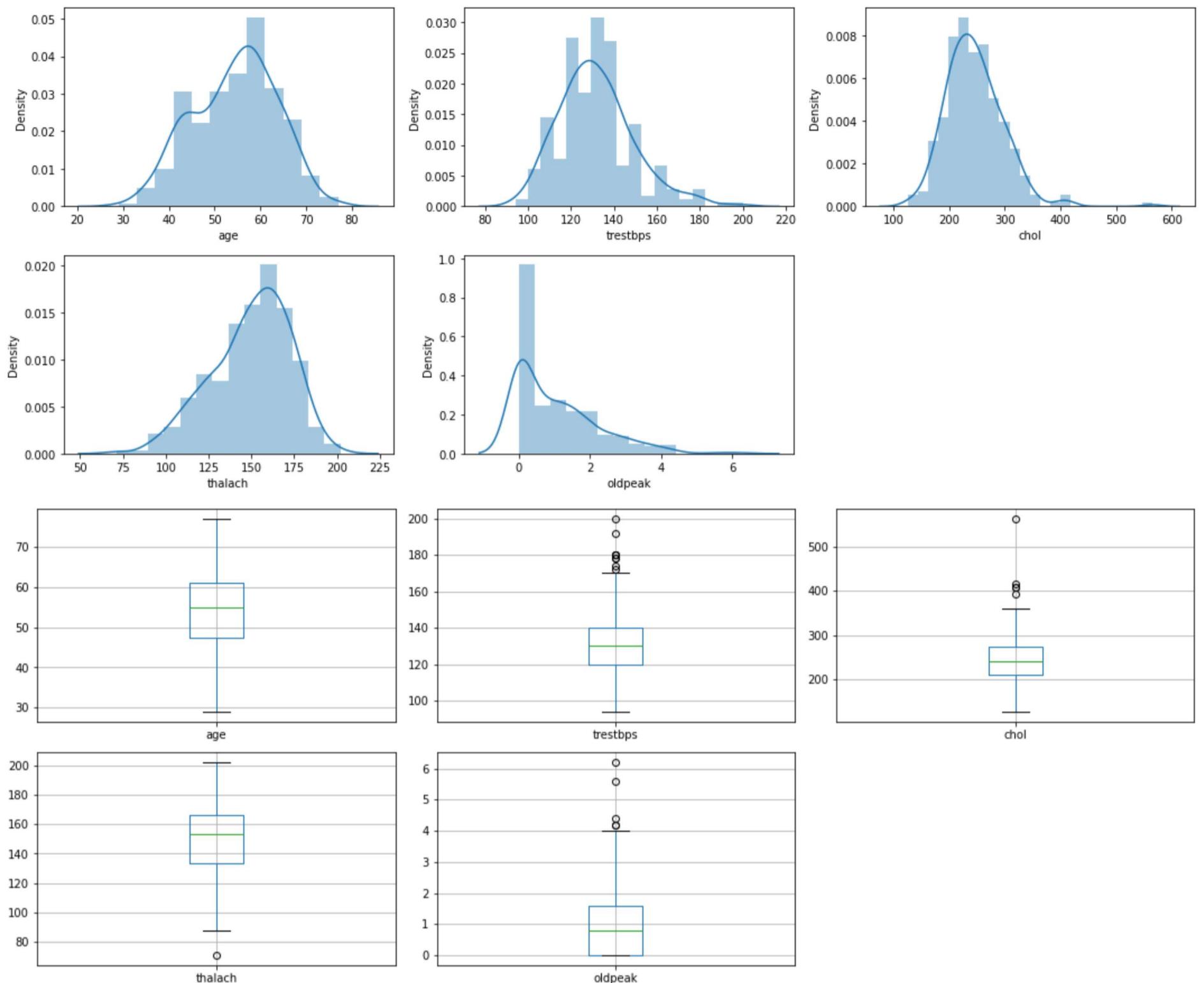
print('\033[1mFeatures Distribution'.center(100))

n=3
nf = [i for i in features if i not in cf]

plt.figure(figsize=[15,3*math.ceil(len(features)/n)])
for c in range(len(nf)):
    plt.subplot(math.ceil(len(features)/n),n,c+1)
    sns.distplot(df[nf[c]])
plt.tight_layout()
plt.show()

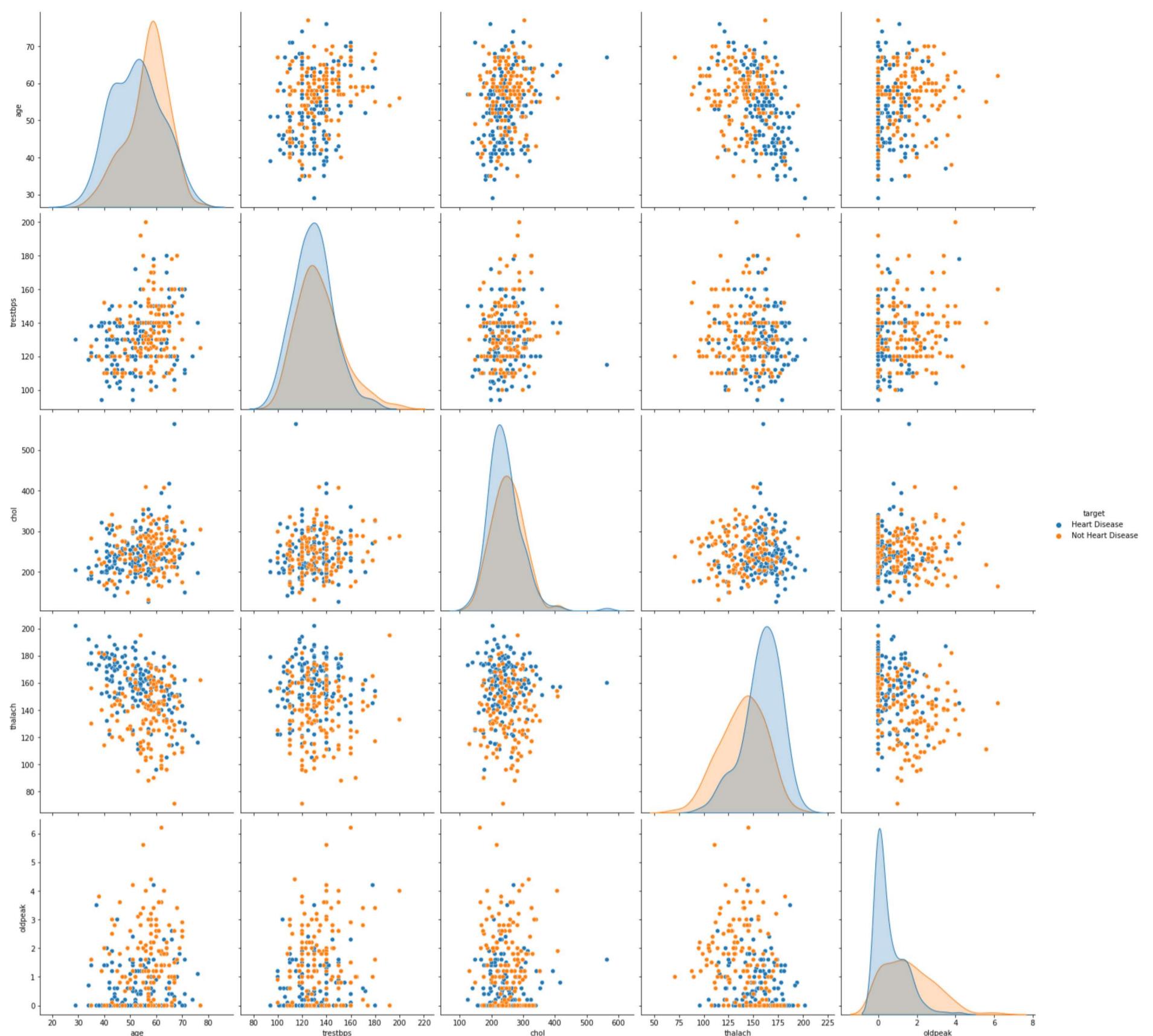
plt.figure(figsize=[15,3*math.ceil(len(features)/n)])
for c in range(len(nf)):
    plt.subplot(math.ceil(len(features)/n),n,c+1)
    df.boxplot(nf[c])
plt.tight_layout()
plt.show()
```

**Features Distribution**



**Inference:** The data is somewhat normally distributed. And there are many outliers present in the dataset. We shall fix these outliers..

```
In [16]: #Understanding the relationship between all the features
ppc=[i for i in df1.columns if i not in cf]
g=sns.pairplot(df1[ppc], hue=target, size=4)
#g.map_upper(sns.kdeplot, levels=1, color=".2")
plt.show()
```



**Inference:** The data samples of most of the features do show some patterns. Also they seem to have lot of overlap for the outcome classes, making it difficult to be distinguishable. Let us proceed to perform cleanup on the data to remove the irregularities...

### 3. Data Preprocessing

In [17]: #Check for empty elements

```
nvc = pd.DataFrame(df.isnull().sum().sort_values(), columns=['Total Null Values'])
nvc['Percentage'] = round(nvc['Total Null Values']/df.shape[0],3)*100
print(nvc)
print('\n\nInference: The dataset doesn't have any null elements')
```

	Total Null Values	Percentage
age	0	0.0
sex	0	0.0
cp	0	0.0
trestbps	0	0.0
chol	0	0.0
fbs	0	0.0
restecg	0	0.0
thalach	0	0.0
exang	0	0.0
oldpeak	0	0.0
slope	0	0.0
ca	0	0.0
thal	0	0.0
target	0	0.0

**Inference:** The dataset doesn't have any null elements

In [18]: #Removal of any Duplicate rows (if any)

```
counter = 0
r,c = original_df.shape
```

```

df1 = df.copy()
df1.drop_duplicates(inplace=True)
df1.reset_index(drop=True, inplace=True)

if df1.shape==(r,c):
    print('\n\nInference: The dataset doesn\'t have any duplicates')
else:
    print(f'\n\nInference: Number of duplicates dropped ---> {r-df1.shape[0]}')

```

Inference: Number of duplicates dropped ---> 1

```

In [19]: #Converting categorical Columns to Numeric

df1 = df.copy()

#Target Variable
#MAP={}
#for i,e in enumerate(df1[target].unique()):
#    MAP[e]=i
#df1[target]=df1[target].map(MAP)
#print('Mapping Target variable --->',MAP)

#One-Hot Binay Encoding
oh=True
dm=True
for i in cf:
    #print(i)
    if df1[i].nunique()==2:
        if oh==True: print("One-Hot Encoding on features:")
        print(i);oh=False
        df1[i]=pd.get_dummies(df1[i], drop_first=True, prefix=str(i))
    if (df1[i].nunique()>2 and df1[i].nunique()<15):
        if dm==True: print("\nDummy Encoding on features:")
        print(i);dm=False
        df1 = pd.concat([df1.drop([i], axis=1), pd.DataFrame(pd.get_dummies(df1[i], drop_first=True, prefix=str(i)))], axis=1)

```

One-Hot Encoding on features:

sex  
fbs  
exang

Dummy Encoding on features:

restecg  
slope  
cp  
thal  
ca

```

In [20]: #Removal of outlier:

df3 = df1.copy()

for i in [i for i in df3.columns]:
    if df3[i].nunique()>12:
        Q1 = df3[i].quantile(0.25)
        Q3 = df3[i].quantile(0.75)
        IQR = Q3 - Q1
        df3 = df3[df3[i] <= (Q3+(1.5*IQR))]
        df3 = df3[df3[i] >= (Q1-(1.5*IQR))]
df3 = df3.reset_index(drop=True)
display(df3.head())
print('\n\nInference: Before removal of outliers, The dataset had {} samples.'.format(df1.shape[0]))
print('\n\nInference: After removal of outliers, The dataset now has {} samples.'.format(df3.shape[0]))

```

	age	sex	trestbps	chol	fbs	thalach	exang	oldpeak	target	restecg_1	...	cp_1	cp_2	cp_3	thal_1	thal_2	thal_3	ca_1	ca_2	ca_3	ca_4
0	63	1	145	233	1	150	0	2.3	1	0	...	0	0	1	1	0	0	0	0	0	0
1	37	1	130	250	0	187	0	3.5	1	1	...	0	1	0	0	1	0	0	0	0	0
2	41	0	130	204	0	172	0	1.4	1	0	...	1	0	0	0	1	0	0	0	0	0
3	56	1	120	236	0	178	0	0.8	1	1	...	1	0	0	0	1	0	0	0	0	0
4	57	0	120	354	0	163	1	0.6	1	1	...	0	0	0	0	1	0	0	0	0	0

5 rows × 23 columns

Inference: Before removal of outliers, The dataset had 303 samples.

Inference: After removal of outliers, The dataset now has 284 samples.

```

In [21]: #Fixing the imbalance using SMOTE Technique

df4 = df3.copy()

print('Original class distribution:')
print(df4[target].value_counts())

xf = df4.columns
X = df4.drop([target],axis=1)

```

```

Y = df4[target]

smote = SMOTE()
X, Y = smote.fit_resample(X, Y)

df5 = pd.DataFrame(X, columns=xf)
df5[target] = Y

print('\nClass distribution after applying SMOTE Technique:,')
print(Y.value_counts())

```

Original class distribution:  
1 159  
0 125  
Name: target, dtype: int64

Class distribution after applying SMOTE Technique:  
1 159  
0 159  
Name: target, dtype: int64

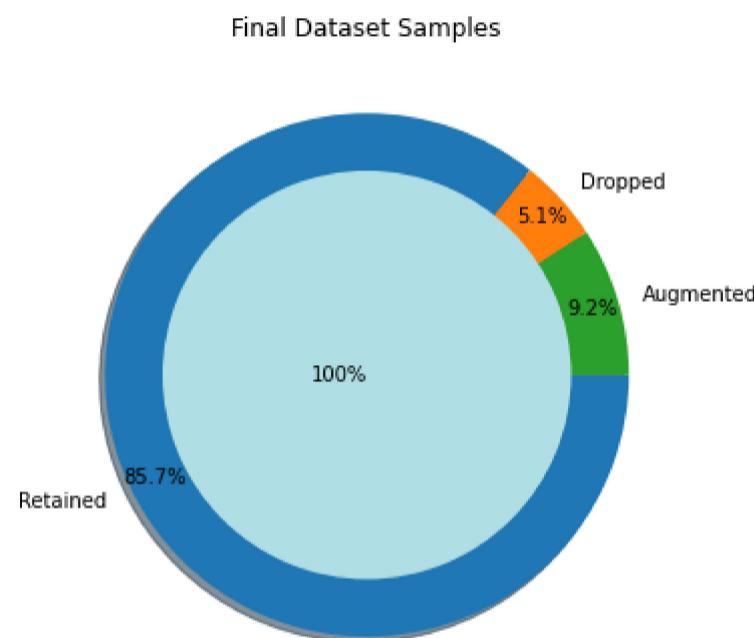
In [22]: #Final Dataset size after performing Preprocessing

```

df = df5.copy()
plt.title('Final Dataset Samples')
plt.pie([df.shape[0], original_df.shape[0]-df4.shape[0], df5.shape[0]-df4.shape[0]], radius = 1, shadow=True,
        labels=['Retained', 'Dropped', 'Augmented'], counterclock=False, autopct='%1.1f%%', pctdistance=0.9, explode=[0,0,0])
plt.pie([df.shape[0]], labels=['100%'], labeldistance=-0, radius=0.78, shadow=True, colors=['powderblue'])
plt.show()

print('\nInference:\nThe final dataset after cleanup has {} samples & {} columns.'.format(df.shape[0], df.shape[1]))

```



Inference: The final dataset after cleanup has 318 samples & 23 columns.

## 4. Data Manipulation

In [68]: #Splitting the data intro training & testing sets

```

df = df5.copy()

X = df.drop([target],axis=1)
Y = df[target]
Train_X, Test_X, Train_Y, Test_Y = train_test_split(X, Y, train_size=0.8, test_size=0.2, random_state=0)

print('Original set ---> ',X.shape,Y.shape,'Training set ---> ',Train_X.shape,Train_Y.shape,'Testing set ---> ', Test_X.sh

```

Original set ---> (318, 22) (318,)  
Training set ---> (254, 22) (254,)  
Testing set ---> (64, 22) (64,)

In [69]: #Feature Scaling (Standardization)

```

std = StandardScaler()

print('Standardization on Training set'.center(100))
Train_X_std = std.fit_transform(Train_X)
Train_X_std = pd.DataFrame(Train_X_std, columns=X.columns)
display(Train_X_std.describe())

print('\n', 'Standardization on Testing set'.center(100))
Test_X_std = std.transform(Test_X)
Test_X_std = pd.DataFrame(Test_X_std, columns=X.columns)
display(Test_X_std.describe())

```

Standardization on Training set

	age	sex	trestbps	chol	fbs	thalach	exang	oldpeak	restecg_1	restecg_2	cp
count	2.540000e+02										
mean	-1.398706e-16	-3.496765e-17	8.042560e-16	2.832380e-16	3.846442e-17	-3.566701e-16	-1.643480e-16	6.993531e-17	-5.944501e-17	-1.398706e-17	-1.398706e-17
std	1.001974e+00										
min	-2.850494e+00	-1.474788e+00	-2.402606e+00	-2.612941e+00	-3.997716e-01	-2.552199e+00	-7.029350e-01	-9.466595e-01	-9.538210e-01	-8.908708e-01	-8.908708e-01
25%	-7.518695e-01	-1.474788e+00	-6.578501e-01	-7.446171e-01	-3.997716e-01	-6.309470e-01	-7.029350e-01	-9.466595e-01	-9.538210e-01	-8.908708e-01	-8.908708e-01
50%	1.317620e-01	6.780635e-01	1.320984e-02	-7.695776e-02	-3.997716e-01	1.188099e-01	-7.029350e-01	-3.414143e-01	-9.538210e-01	-8.908708e-01	-8.908708e-01
75%	6.840317e-01	6.780635e-01	6.842698e-01	6.131439e-01	-3.997716e-01	6.694127e-01	1.422607e+00	6.145459e-01	1.048415e+00	-8.908708e-01	-8.908708e-01
max	2.451295e+00	6.780635e-01	2.697450e+00	2.638564e+00	2.501428e+00	2.461800e+00	1.422607e+00	2.761203e+00	1.048415e+00	1.122497e+01	1.122497e+01

8 rows × 22 columns

Standardization on Testing set														
	age	sex	trestbps	chol	fbs	thalach	exang	oldpeak	restecg_1	restecg_2	...	cp_1	cp_2	cp
count	64.000000	64.000000	64.000000	64.000000	64.000000	64.000000	64.000000	64.000000	64.000000	64.000000e+01	...	64.000000	64.000000	64.000000
mean	-0.254827	0.106212	0.088704	-0.002267	-0.127784	0.019965	-0.105126	0.025780	-0.015273	-8.908708e-02	...	0.062432	0.004188	0.2134
std	0.916338	0.958357	1.073263	1.020752	0.852328	1.185650	0.963218	1.026859	1.007059	1.398750e-17	...	1.066078	1.010156	1.3344
min	-2.298225	-1.474788	-1.999970	-2.096768	-0.399772	-2.880218	-0.702935	-0.946660	-0.953821	-8.908708e-02	...	-0.419435	-0.598572	-0.2505
25%	-0.889937	-1.474788	-0.657850	-0.761449	-0.399772	-0.876961	-0.702935	-0.946660	-0.953821	-8.908708e-02	...	-0.419435	-0.598572	-0.2505
50%	-0.144373	0.678064	-0.053896	0.080139	-0.399772	0.329679	-0.702935	-0.068481	-0.953821	-8.908708e-02	...	-0.419435	-0.598572	-0.2505
75%	0.380283	0.678064	0.684270	0.708524	-0.399772	1.020861	1.422607	0.787309	1.048415	-8.908708e-02	...	-0.419435	1.670642	-0.2505
max	1.567663	0.678064	2.697450	2.189718	2.501428	1.899483	1.422607	2.956354	1.048415	-8.908708e-02	...	2.384158	1.670642	3.9916

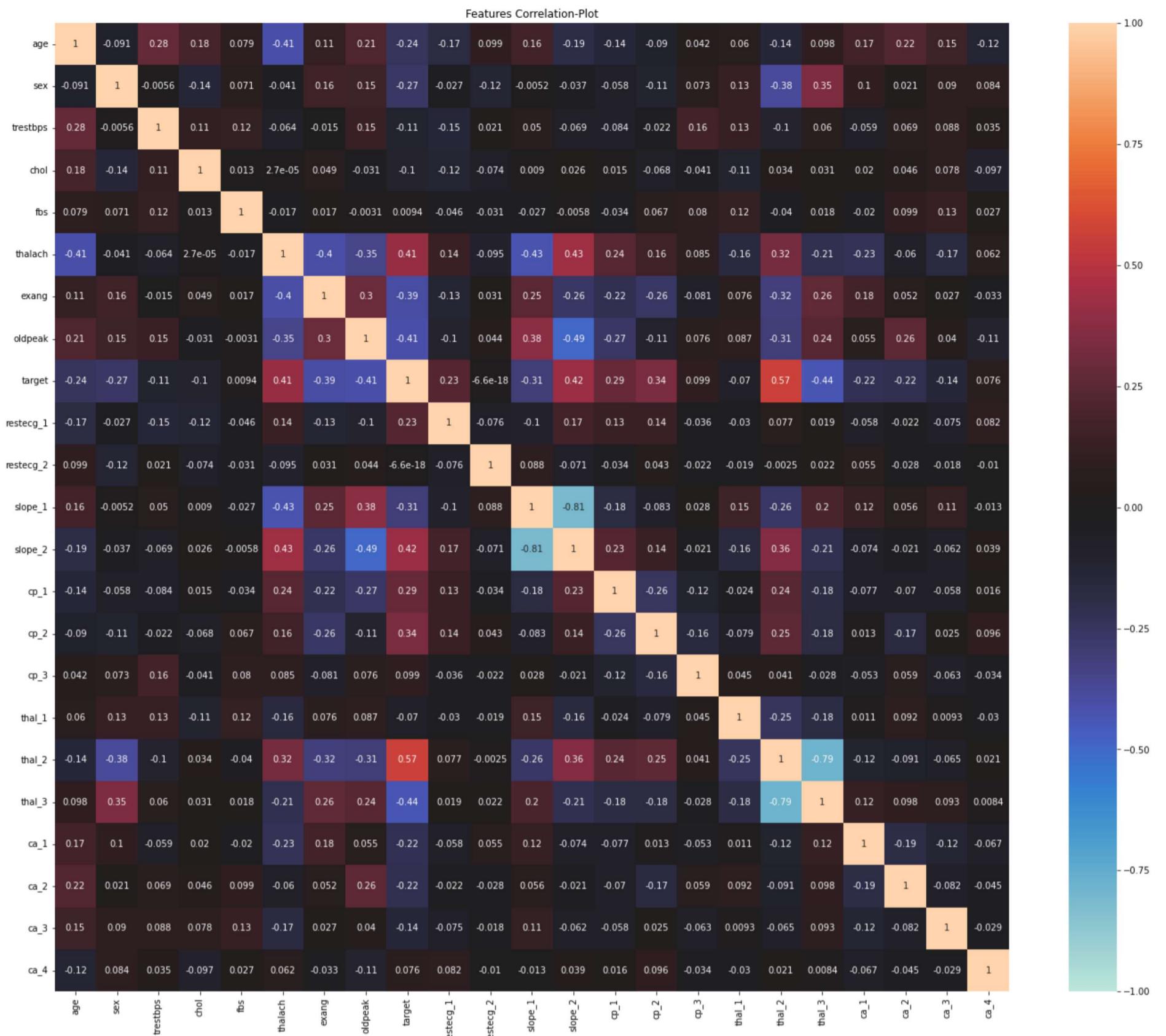
8 rows × 22 columns

## 5. Feature Selection/Extraction

In [26]:

```
#Checking the correlation

features = df.columns
plt.figure(figsize=[24,20])
plt.title('Features Correlation-Plot')
sns.heatmap(df[features].corr(), vmin=-1, vmax=1, center=0, annot=True) #,
plt.show()
```



**Inference:** Correlation plt between the variables convey lot of information about the realationship between them. Especially in case of gender & survived.Hence it is clear that probably women were given more importance to save first. Similiarly we have obvious strong correlation between fare & Passenger-Class.

Let us check with different techniques if we can improve the model's performance by performing Feature Selection/Extraction steps to take care of these multi-collinearity...

**Strategy:** We can fix these multicollinearity with two techniques:

1. Manual Method - Variance Inflation Factor (VIF)
2. Automatic Method - Recursive Feature Elimination (RFE)
3. Decomposition Method - Principle Component Analysis (PCA)

## 5a. Manual Method - VIF

In [31]:

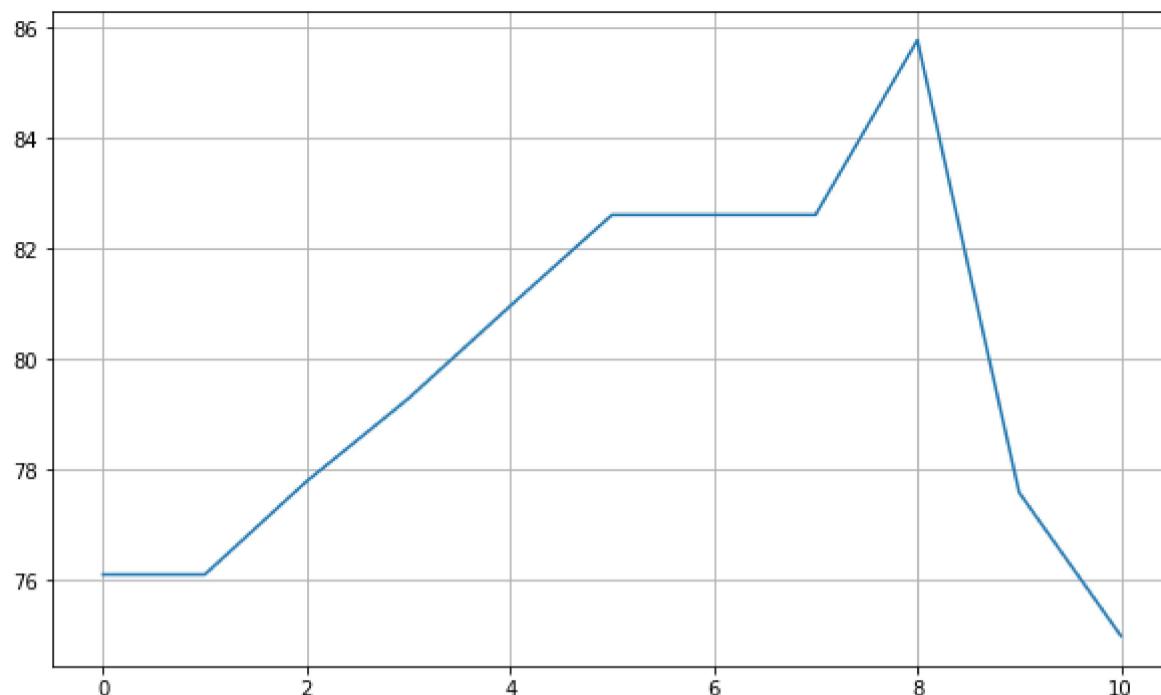
```
# Calculate the VIFs to remove multicollinearity

DROP=[]; scores=[]
#scores.append(f1_score(Test_Y,LogisticRegression().fit(Train_X_std, Train_Y).predict(Test_X_std)))

for i in range(len(X.columns.values)-10):
    vif = pd.DataFrame()
    Xs = X.drop(DROP, axis=1)
    #print(DROP)
    vif['Features'] = Xs.columns
    vif['VIF'] = [variance_inflation_factor(Xs.values, i) for i in range(Xs.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    vif.reset_index(drop=True, inplace=True)
    DROP.append(vif.Features[0])
    if vif.VIF[0]>1.5:
        scores.append(f1_score(Test_Y,LogisticRegression().fit(Train_X_std.drop(DROP, axis=1), Train_Y).predict(Test_X_std.drop(DROP, axis=1))))
        #print(scores)

plt.plot(scores)
#plt.ylim([0.7,0.85])
```

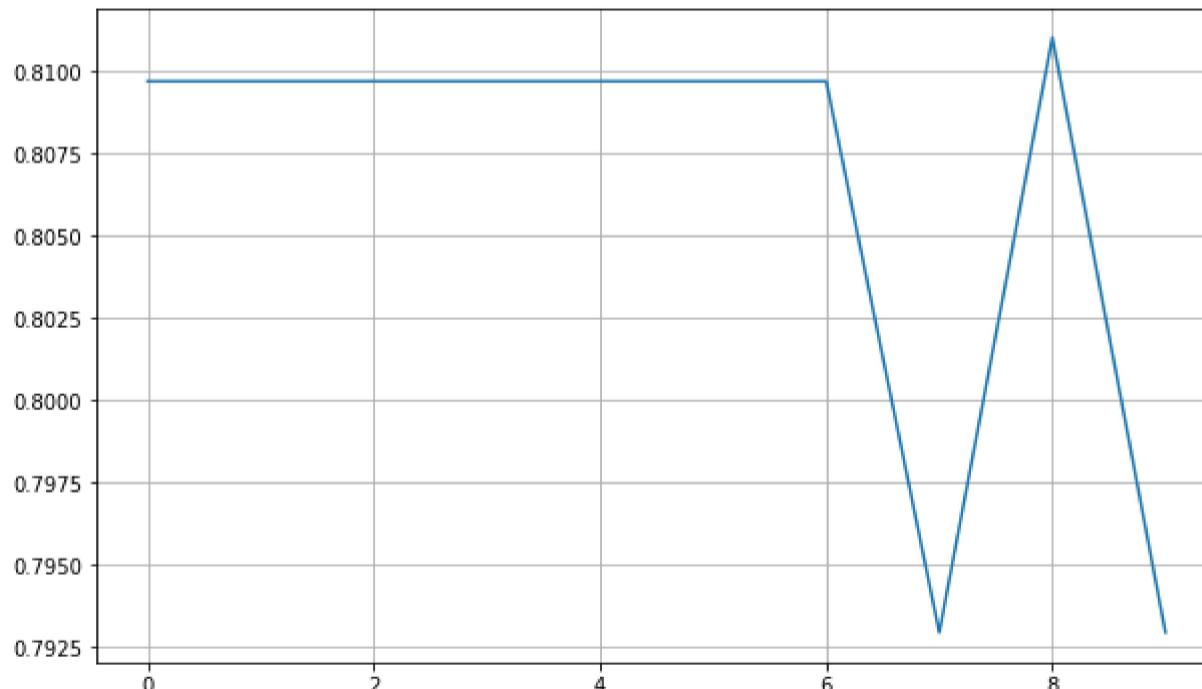
```
plt.grid()  
plt.show()
```



## 5b. Automatic Method - RFE

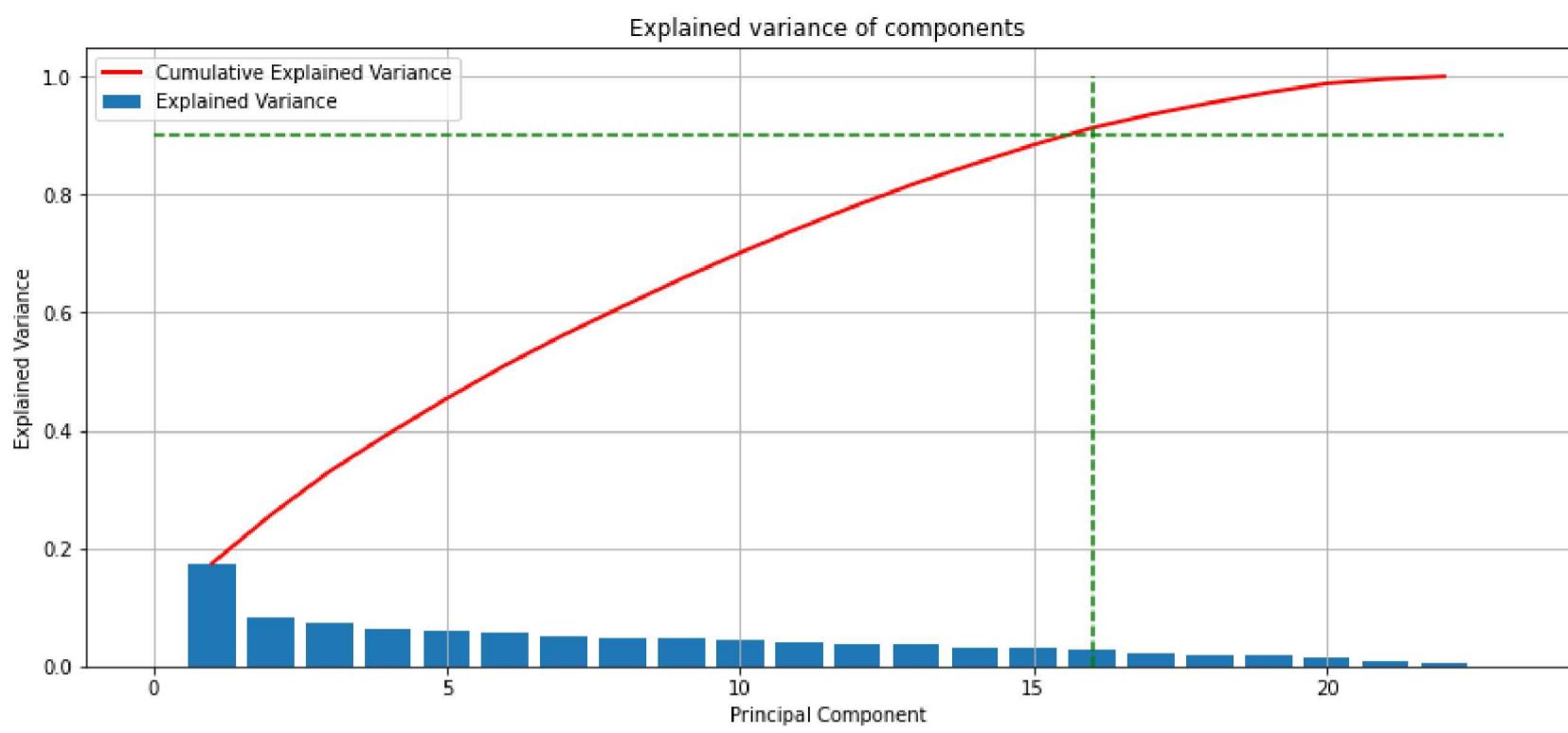
In [33]:

```
# Applying Recurrive Feature Elimination  
  
# Running RFE with the output number of the variable equal to 10  
LR = LogisticRegression().fit(Train_X_std, Train_Y)  
scores = []  
  
for i in range(10):  
    rfe = RFE(LR, n_features_to_select=len(Train_X_std.columns)-i)  
    rfe = rfe.fit(Train_X_std, Train_Y)  
    scores.append(f1_score(Test_Y, LogisticRegression().fit(Train_X_std[Train_X_std.columns[rfe.support_]]), Train_Y).predict(Test_X))  
    print(scores)  
  
plt.plot(scores)  
plt.ylim([0.80, 0.84])  
plt.grid()  
plt.show()
```



In [35]:

```
from sklearn.decomposition import PCA  
  
pca = PCA().fit(Train_X_std)  
  
fig, ax = plt.subplots(figsize=(14,6))  
x_values = range(1, pca.n_components_+1)  
ax.bar(x_values, pca.explained_variance_ratio_, lw=2, label='Explained Variance')  
ax.plot(x_values, np.cumsum(pca.explained_variance_ratio_), lw=2, label='Cumulative Explained Variance', color='red')  
plt.plot([0,pca.n_components_+1],[0.90,0.90], 'g--')  
plt.plot([16,16],[0,1], 'g--')  
ax.set_title('Explained variance of components')  
ax.set_xlabel('Principal Component')  
ax.set_ylabel('Explained Variance')  
plt.grid()  
plt.legend()  
plt.show()
```



**Inference:** We shall avoid performing dimensionality reduction for the current problem.

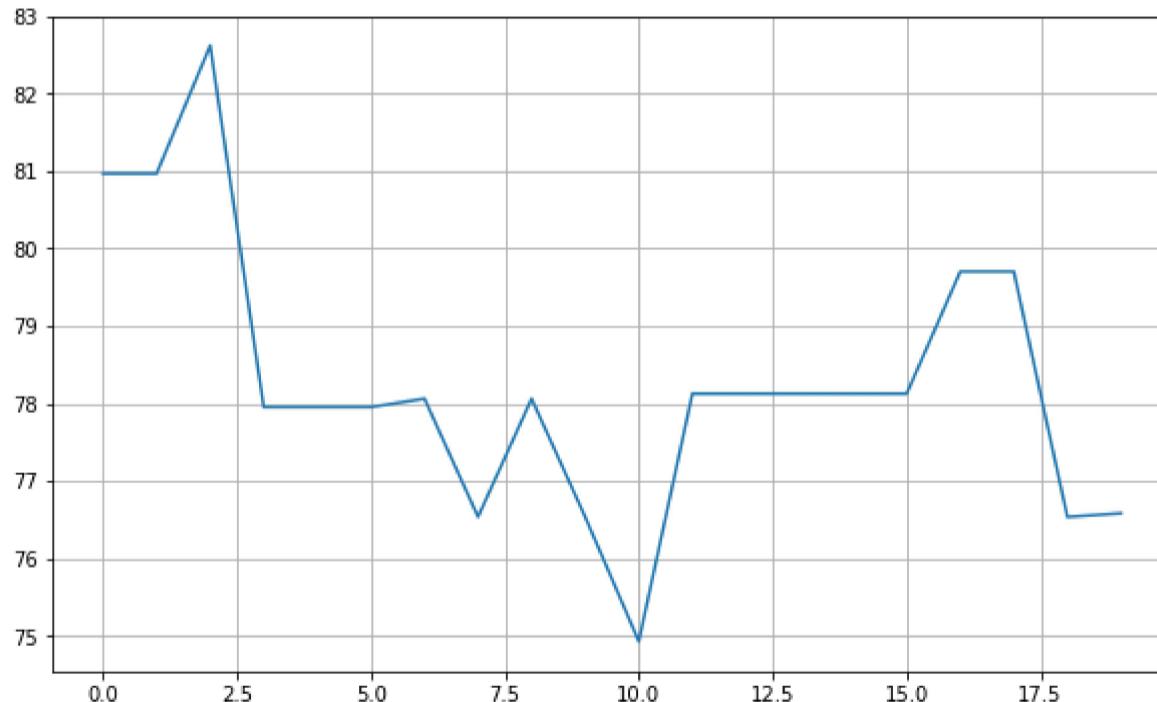
```
In [36]: #Applying PCA Transformations

scores=[]
for i in range(20):
    pca = PCA(n_components=Train_X_std.shape[1]-i)
    Train_X_std_pca = pca.fit_transform(Train_X_std)
    #print('The shape of final transformed training feature set:')
    #print(Train_X_std_pca.shape)
    Train_X_std_pca = pd.DataFrame(Train_X_std_pca)

    Test_X_std_pca = pca.transform(Test_X_std)
    #print('\nThe shape of final transformed testing feature set:')
    #print(Test_X_std_pca.shape)
    Test_X_std_pca = pd.DataFrame(Test_X_std_pca)

    scores.append(f1_score(Test_Y,LogisticRegression().fit(Train_X_std_pca, Train_Y).predict(Test_X_std_pca),average='weighted')*1

plt.plot(scores)
#plt.ylim([0.80,0.84])
plt.grid()
plt.show()
```



**Inference:** In VIF, RFE & PCA Techniques, we did notice any better scores upon dropping some multicollinear features. But in order to avoid the curse of dimensionality, we can capture top 90% of the data Variance explained by top 33 PCA components.

```
In [82]: DROP=[]; scores=[]

for i in range(len(X.columns.values)-13):
    vif = pd.DataFrame()
    Xs = X.drop(DROP, axis=1)
    vif['Features'] = Xs.columns
    vif['VIF'] = [variance_inflation_factor(Xs.values, i) for i in range(Xs.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    vif.reset_index(drop=True, inplace=True)
    DROP.append(vif.Features[0])
    #print(f1_score(Test_Y,LogisticRegression().fit(Train_X_std.drop(DROP, axis=1), Train_Y).predict(Test_X_std.drop(DROP, axis=1)),
```

```
Train_X_std = Train_X_std.drop(DROP, axis=1)
Test_X_std = Test_X_std.drop(DROP, axis=1)
```

## 6. Predictive Modeling

In [83]:

```
#Let us create first create a table to store the results of various models

Evaluation_Results = pd.DataFrame(np.zeros((8,5)), columns=['Accuracy', 'Precision', 'Recall', 'F1-score', 'AUC-ROC score'])
Evaluation_Results.index=[ 'Logistic Regression (LR)', 'Decision Tree Classifier (DT)', 'Random Forest Classifier (RF)', 'Naïve Bayes
                           'Support Vector Machine (SVM)', 'K Nearest Neighbours (KNN)', 'Gradient Boosting (GB)', 'Extreme Gradient B
Evaluation_Results
```

Out[83]:

	Accuracy	Precision	Recall	F1-score	AUC-ROC score
<b>Logistic Regression (LR)</b>	0.0	0.0	0.0	0.0	0.0
<b>Decision Tree Classifier (DT)</b>	0.0	0.0	0.0	0.0	0.0
<b>Random Forest Classifier (RF)</b>	0.0	0.0	0.0	0.0	0.0
<b>Naïve Bayes Classifier (NB)</b>	0.0	0.0	0.0	0.0	0.0
<b>Support Vector Machine (SVM)</b>	0.0	0.0	0.0	0.0	0.0
<b>K Nearest Neighbours (KNN)</b>	0.0	0.0	0.0	0.0	0.0
<b>Gradient Boosting (GB)</b>	0.0	0.0	0.0	0.0	0.0
<b>Extreme Gradient Boosting (XGB)</b>	0.0	0.0	0.0	0.0	0.0

In [84]:

```
#Let us define functions to summarise the Prediction's scores .

#Classification Summary Function
def Classification_Summary(pred,pred_prob,i):
    Evaluation_Results.iloc[i]['Accuracy']=round(accuracy_score(Test_Y, pred),3)*100
    Evaluation_Results.iloc[i]['Precision']=round(precision_score(Test_Y, pred),3)*100 #, average='weighted'
    Evaluation_Results.iloc[i]['Recall']=round(recall_score(Test_Y, pred),3)*100 #, average='weighted'
    Evaluation_Results.iloc[i]['F1-score']=round(f1_score(Test_Y, pred),3)*100 #, average='weighted'
    Evaluation_Results.iloc[i]['AUC-ROC score']=round(roc_auc_score(Test_Y, pred_prob[:, 1]),3)*100 #, multi_class='ovr'
    print('{}{}\\033[1m Evaluating {} \\033[0m{}{}\\n'.format('<'*3,'-'*35,Evaluation_Results.index[i], '-'*35,>'*3))
    print('Accuracy = {}%'.format(round(accuracy_score(Test_Y, pred),3)*100))
    print('F1 Score = {}%'.format(round(f1_score(Test_Y, pred),3)*100)) #, average='weighted'
    print('\\n \\033[1mConfusiton Matrix:\\033[0m\\n',confusion_matrix(Test_Y, pred))
    print('\\n\\033[1mClassification Report:\\033[0m\\n',classification_report(Test_Y, pred))

    auc_roc(Test_Y, pred_prob, curves=['each_class'])
    plt.show()

#Visualising Function
def AUC_ROC_plot(Test_Y, pred):
    ref = [0 for _ in range(len(Test_Y))]
    ref_auc = roc_auc_score(Test_Y, ref)
    lr_auc = roc_auc_score(Test_Y, pred)

    ns_fpr, ns_tpr, _ = roc_curve(Test_Y, ref)
    lr_fpr, lr_tpr, _ = roc_curve(Test_Y, pred)

    plt.plot(ns_fpr, ns_tpr, linestyle='--')
    plt.plot(lr_fpr, lr_tpr, marker='.', label='AUC = {}'.format(round(roc_auc_score(Test_Y, pred)*100,2)))
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()
```

### 1. Logistic Regression:

In [85]:

```
# Building Logistic Regression Classifier

LR_model = LogisticRegression()

space = dict()
space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
space['C'] = loguniform(1e-5, 100)

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(LR_model, space, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

LR = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = LR.predict(Test_X_std)
pred_prob = LR.predict_proba(Test_X_std)
Classification_Summary(pred,pred_prob,0)

print('\\n\\033[1mInterpreting the Output of Logistic Regression:\\n\\033[0m')

print('intercept ', LR.intercept_[0])
```

```

print('classes', LR.classes_)
display(pd.DataFrame({'coeff': LR.coef_[0]}, index=Train_X_std.columns))

```

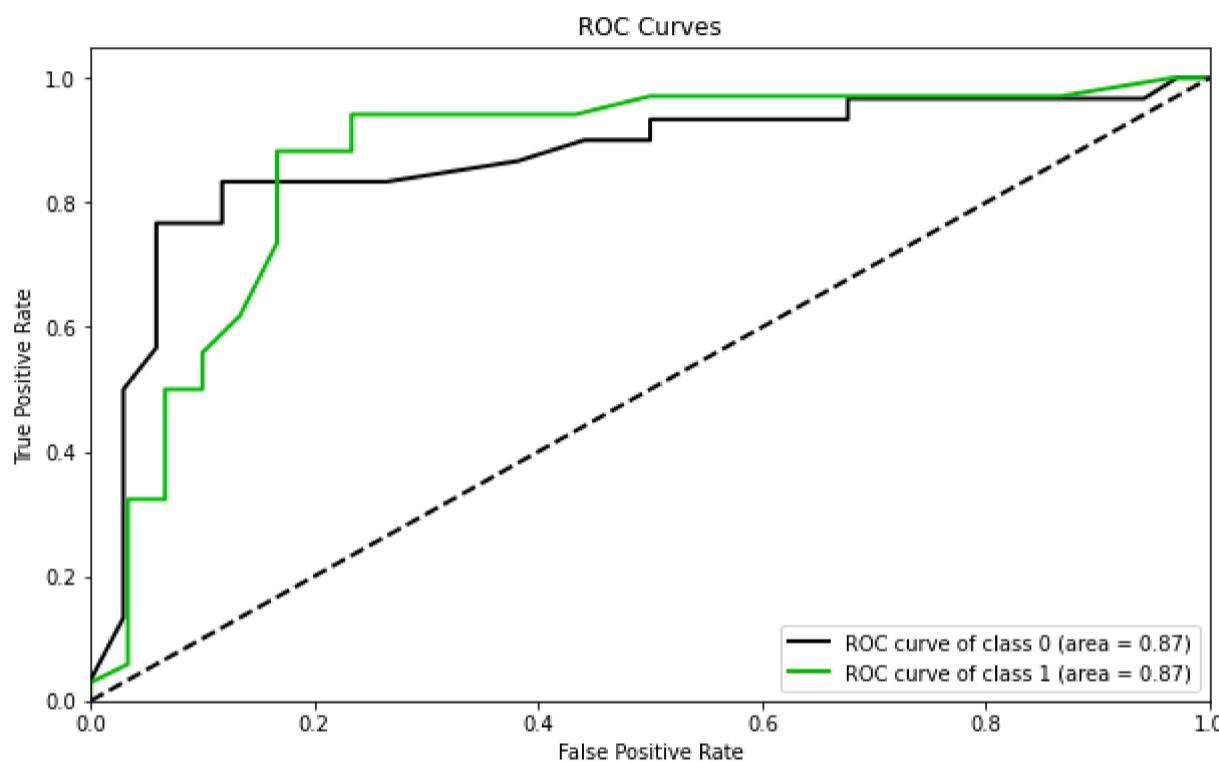
<<<----- Evaluating Logistic Regression (LR) ----->>>

Accuracy = 85.9%  
F1 Score = 87.7%

**Confusion Matrix:**  
[[23 7]  
 [ 2 32]]

**Classification Report:**

	precision	recall	f1-score	support
0	0.92	0.77	0.84	30
1	0.82	0.94	0.88	34
accuracy			0.86	64
macro avg	0.87	0.85	0.86	64
weighted avg	0.87	0.86	0.86	64



**Interpreting the Output of Logistic Regression:**

```

intercept -0.1015150345284094
classes [0 1]

```

coeff
fbs -0.014043
exang -0.441003
restecg_2 0.110824
slope_2 0.890762
cp_1 0.669963
cp_2 0.951043
cp_3 0.559555
thal_1 -0.061598
thal_3 -0.862806
ca_1 -0.712539
ca_2 -0.840116
ca_3 -0.173126
ca_4 0.188404

## 2. Decision Tree Classifier:

In [86]:

```

#Building Decision Tree Classifier

DT_model = DecisionTreeClassifier()

param_dist = {"max_depth": [3, None],
              "max_features": randint(1, 9),
              "min_samples_leaf": randint(1, 9),
              "criterion": ["gini", "entropy"]}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

```

```

RCV = RandomizedSearchCV(DT_model, param_dist, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

DT = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = DT.predict(Test_X_std)
pred_prob = DT.predict_proba(Test_X_std)
Classification_Summary(pred, pred_prob, 1)

print('\nInterpreting the output of Decision Tree:\n')
tree.plot_tree(DT)
plt.show()

```

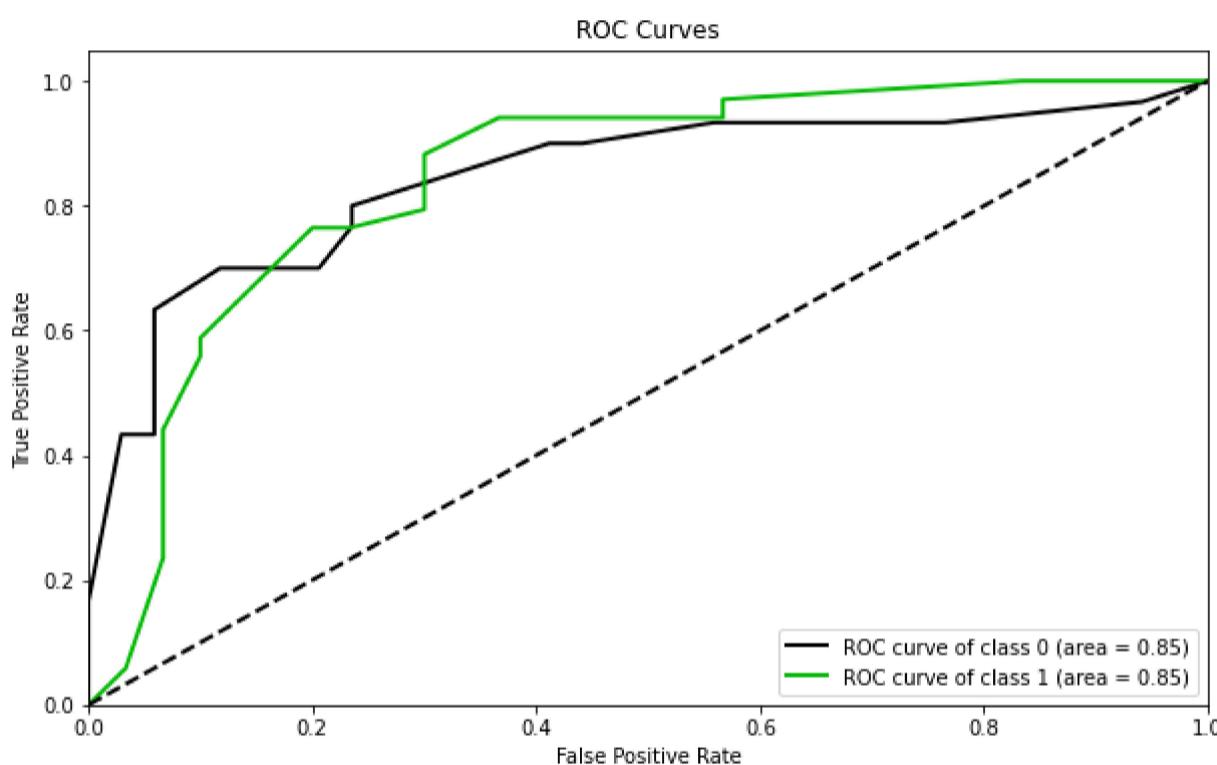
<<<----- Evaluating Decision Tree Classifier (DT) ----->>>

Accuracy = 78.10000000000001%  
F1 Score = 78.8%

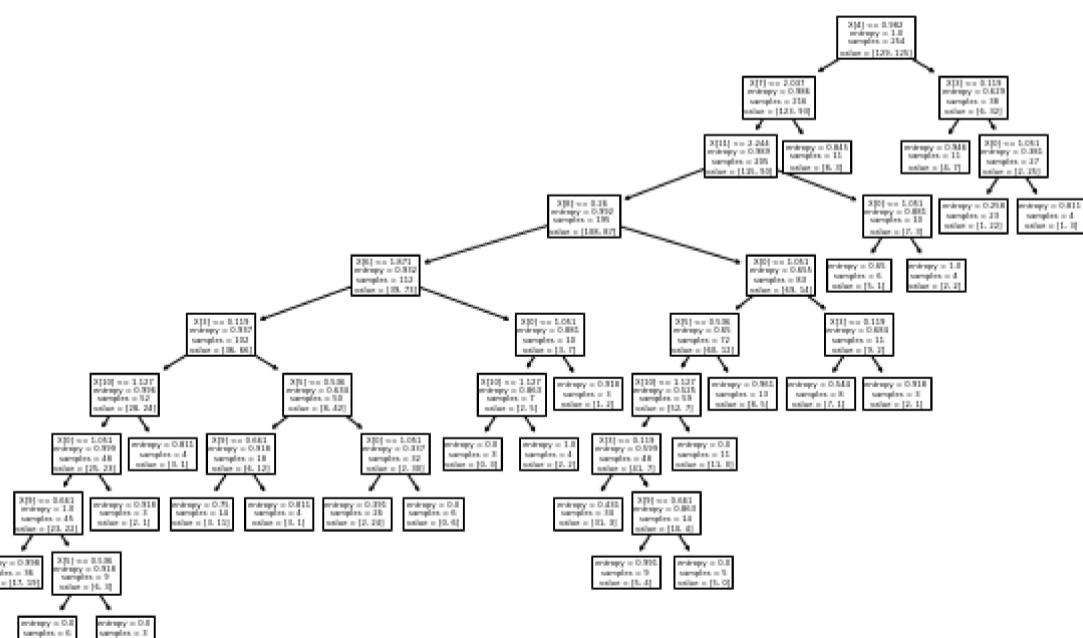
**Confusion Matrix:**  
[[24 6]  
 [ 8 26]]

**Classification Report:**

	precision	recall	f1-score	support
0	0.75	0.80	0.77	30
1	0.81	0.76	0.79	34
accuracy			0.78	64
macro avg	0.78	0.78	0.78	64
weighted avg	0.78	0.78	0.78	64



**Interpreting the output of Decision Tree:**



### 3. Random Forest Classifier:

In [87]:

```

# Building Random-Forest Classifier

RF_model = RandomForestClassifier()

param_dist = {
    'bootstrap': [True, False],
    'max_depth': [10, 20, 50, 100, None],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
}

```

```

'n_estimators': [50, 100]}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(RF_model, param_dist, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

RF = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = RF.predict(Test_X_std)
pred_prob = RF.predict_proba(Test_X_std)
Classification_Summary(pred, pred_prob, 2)

print('\n[1mInterpreting the output of Random Forest:\n[0m')
rfi=pd.Series(RF.feature_importances_, index=Train_X_std.columns).sort_values(ascending=False)
plt.barh(rfi.index, rfi.values)
plt.show()

```

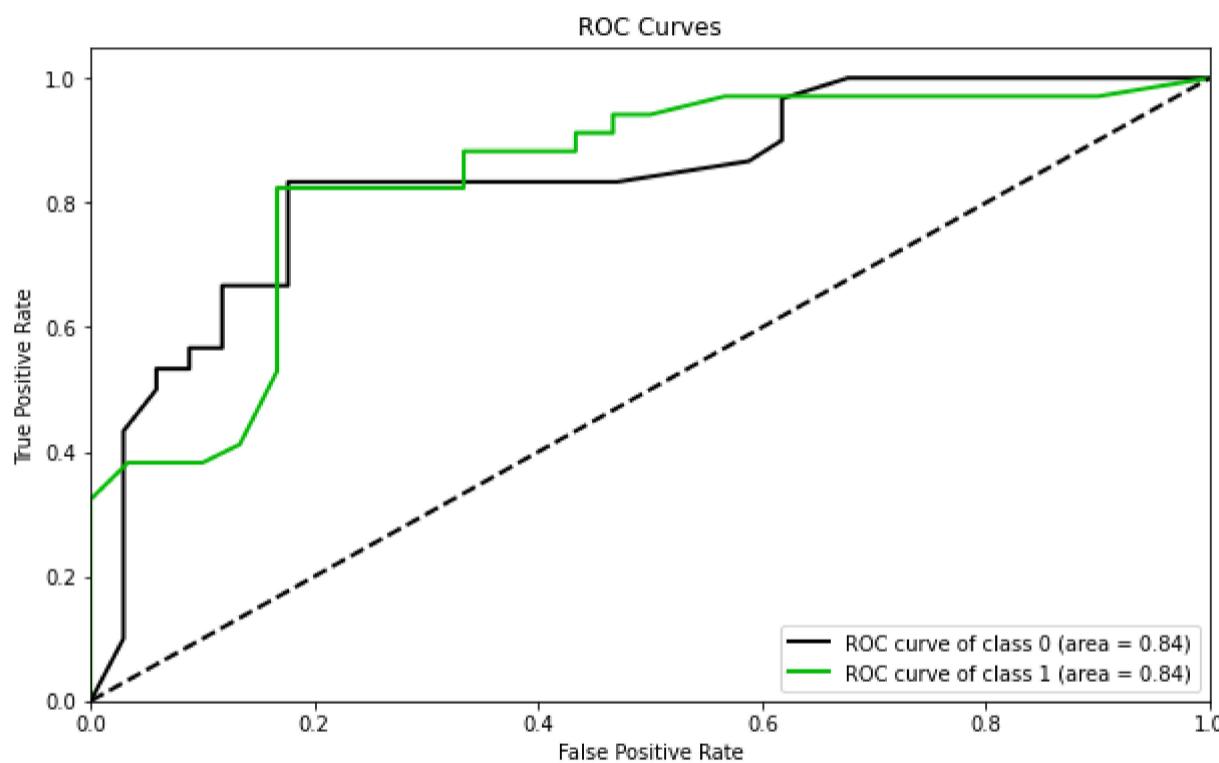
<<<----- Evaluating Random Forest Classifier (RF) ----->>>

Accuracy = 76.6%  
F1 Score = 79.5%

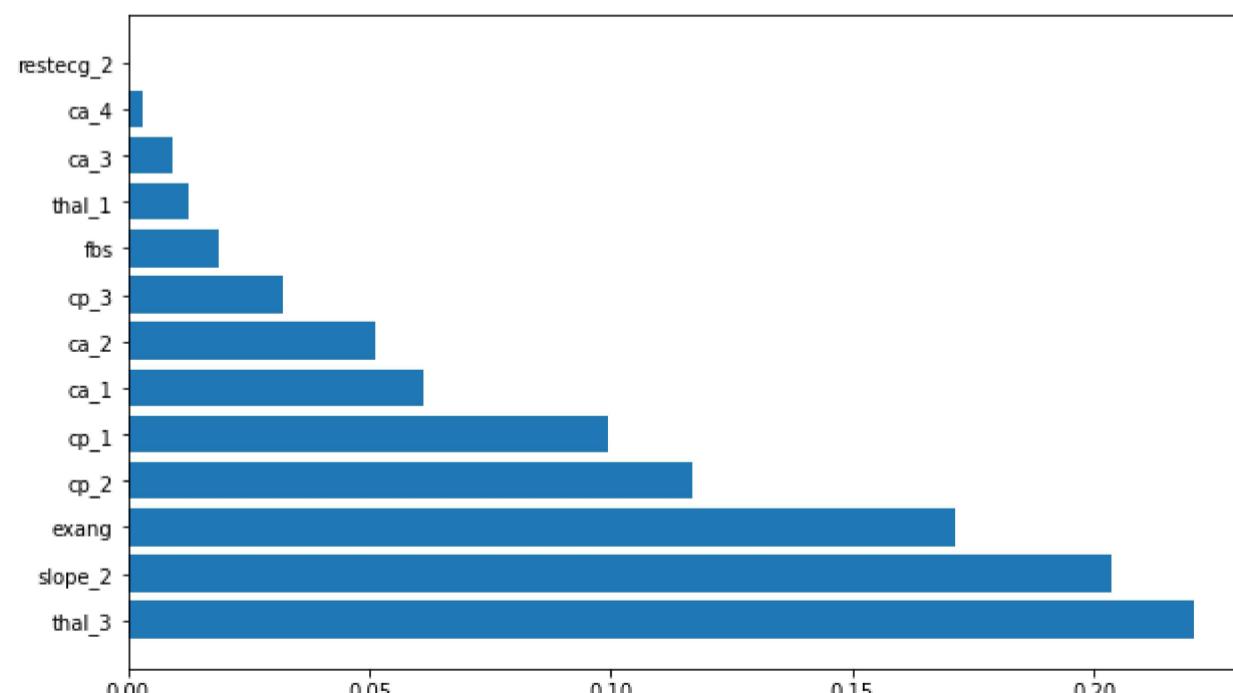
**Confusion Matrix:**  
[[20 10]  
 [ 5 29]]

**Classification Report:**

	precision	recall	f1-score	support
0	0.80	0.67	0.73	30
1	0.74	0.85	0.79	34
accuracy			0.77	64
macro avg	0.77	0.76	0.76	64
weighted avg	0.77	0.77	0.76	64



**Interpreting the output of Random Forest:**



## 4. Naive Bayes Classifier:

```
In [88]: # Building Naive Bayes Classifier
NB_model = BernoulliNB()
```

```

params = {'alpha': [0.01, 0.1, 0.5, 1.0, 10.0]}
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(NB_model, params, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

NB = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = NB.predict(Test_X_std)
pred_prob = NB.predict_proba(Test_X_std)
Classification_Summary(pred, pred_prob, 3)

```

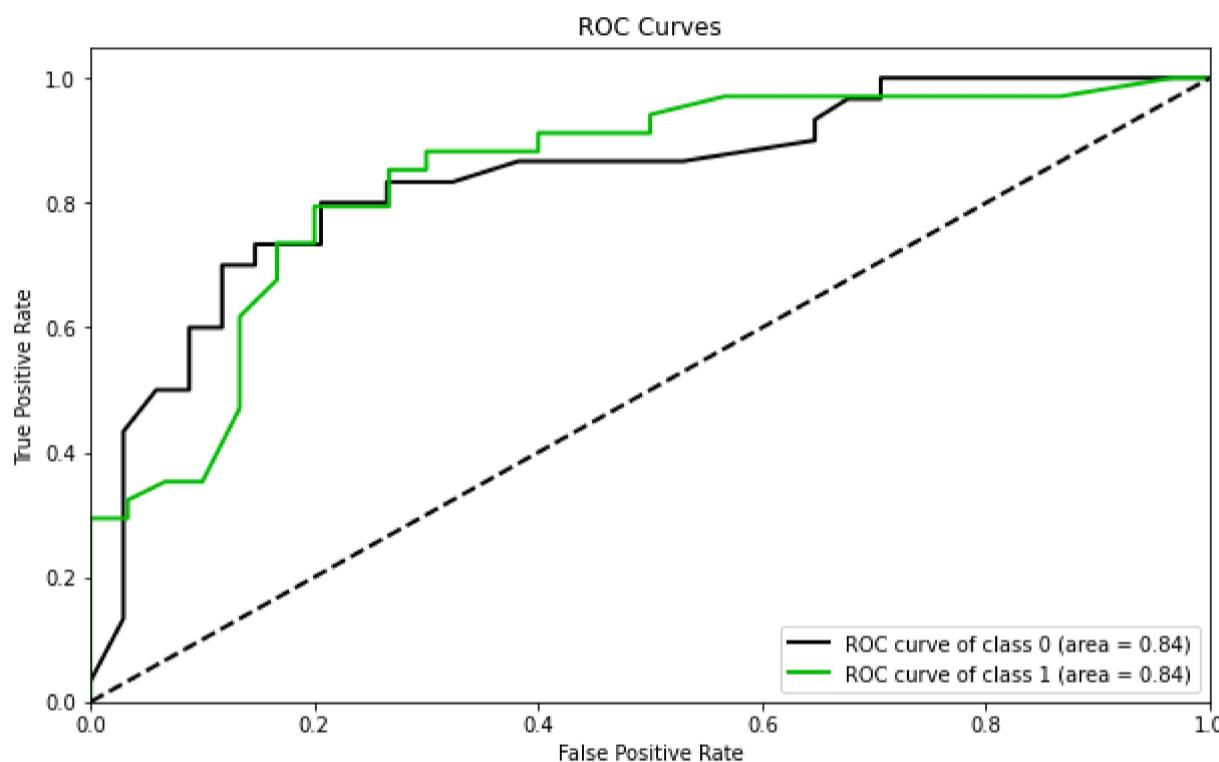
<<<----- Evaluating Naïve Bayes Classifier (NB) ----->>>

Accuracy = 76.6%  
F1 Score = 78.3%

**Confusion Matrix:**  
[[22 8]  
[ 7 27]]

**Classification Report:**

	precision	recall	f1-score	support
0	0.76	0.73	0.75	30
1	0.77	0.79	0.78	34
accuracy			0.77	64
macro avg	0.77	0.76	0.76	64
weighted avg	0.77	0.77	0.77	64



## 5. Support Vector Machine Classifier:

```

In [89]: # Building Support Vector Machine Classifier

SVM_model = SVC(probability=True).fit(Train_X_std, Train_Y)

svm_param = {"C": [.01, .1, 1, 5, 10, 100],
             "gamma": [.01, .1, 1, 5, 10, 100],
             "kernel": ["rbf"],
             "random_state": [1]}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(SVM_model, svm_param, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

SVM = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = SVM.predict(Test_X_std)
pred_prob = SVM.predict_proba(Test_X_std)
Classification_Summary(pred, pred_prob, 4)

```

<<<----- Evaluating Support Vector Machine (SVM) ----->>>

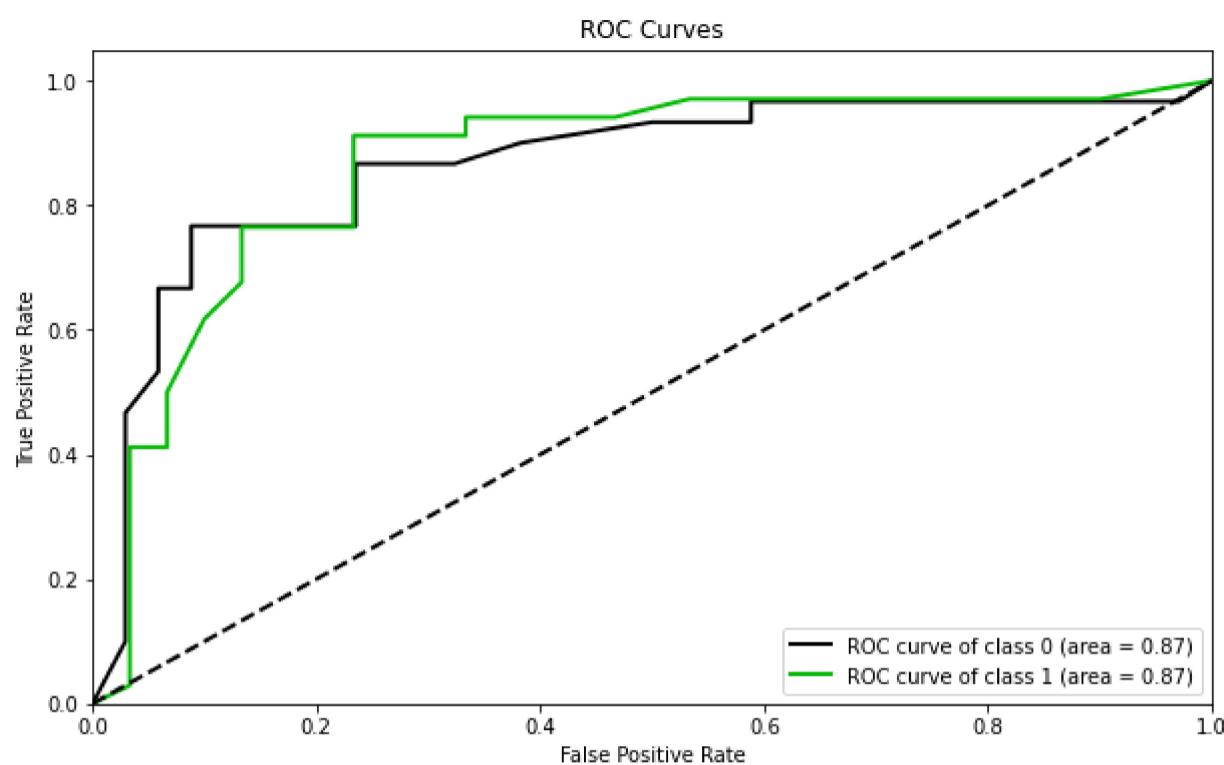
Accuracy = 81.2%  
F1 Score = 84.2%

**Confusion Matrix:**  
[[20 10]  
[ 2 32]]

**Classification Report:**

	precision	recall	f1-score	support
0	0.91	0.67	0.77	30
1	0.76	0.94	0.84	34

accuracy		0.81	64
macro avg	0.84	0.80	64
weighted avg	0.83	0.81	64



## 6. K-Nearest Neighbours Classifier:

```
In [90]: # Building K-Nearest Neighbours Classifier
KNN_model = KNeighborsClassifier()

knn_param = {"n_neighbors": [i for i in range(1,30,5)],
             "weights": ["uniform", "distance"],
             "algorithm": ["ball_tree", "kd_tree", "brute"],
             "leaf_size": [1, 10, 30],
             "p": [1,2]}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(KNN_model, knn_param, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

KNN = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = KNN.predict(Test_X_std)
pred_prob = KNN.predict_proba(Test_X_std)
Classification_Summary(pred,pred_prob,5)
```

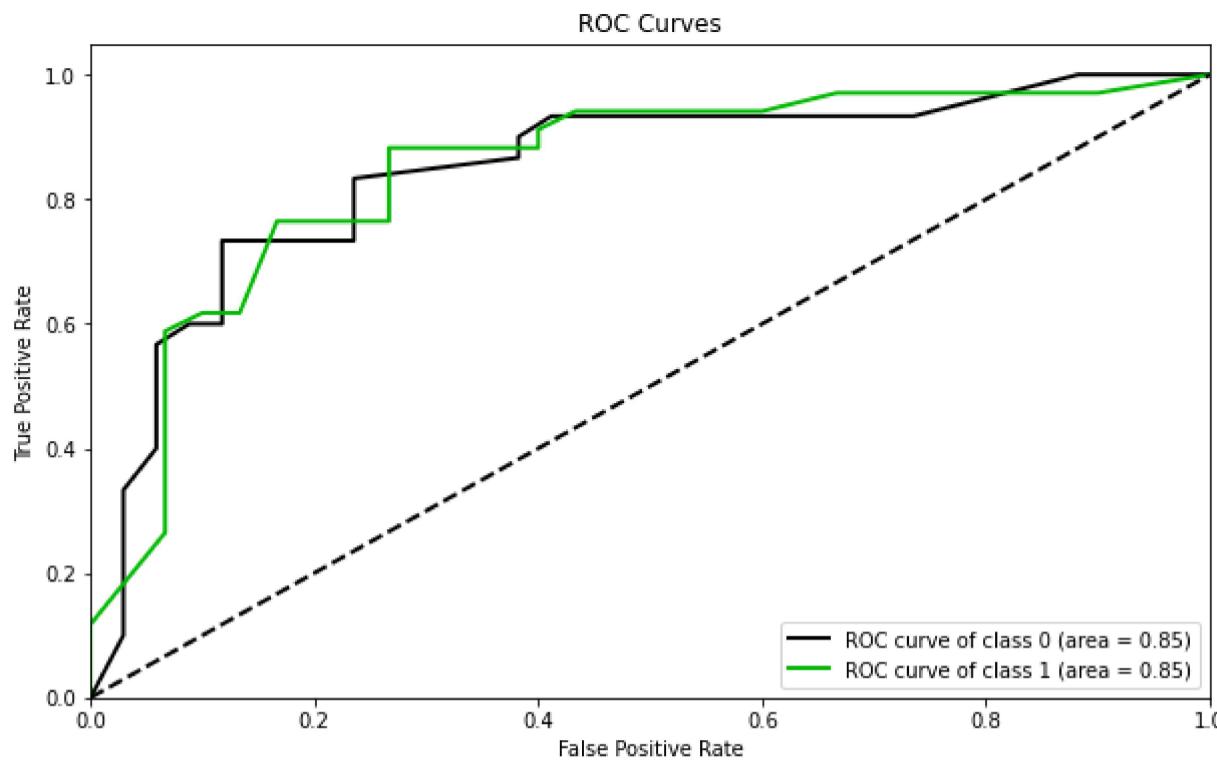
<<<----- Evaluating K Nearest Neighbours (KNN) ----->>>

Accuracy = 81.2%  
F1 Score = 83.3%

Confusion Matrix:  
[[22 8]  
 [ 4 30]]

Classification Report:

	precision	recall	f1-score	support
0	0.85	0.73	0.79	30
1	0.79	0.88	0.83	34
accuracy			0.81	64
macro avg	0.82	0.81	0.81	64
weighted avg	0.82	0.81	0.81	64



## 7. Gradient Boosting Classifier:

In [91]:

```
# Building Gradient Boosting Classifier

GB_model = GradientBoostingClassifier().fit(Train_X_std, Train_Y)
param_dist = {
    "n_estimators": [5, 20, 100, 500],
    "max_depth": [1, 3, 5, 7, 9],
    "learning_rate": [0.01, 0.1, 1, 10, 100]
}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(GB_model, param_dist, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

GB = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = GB.predict(Test_X_std)
pred_prob = GB.predict_proba(Test_X_std)
Classification_Summary(pred, pred_prob, 6)
```

<<<----- Evaluating Gradient Boosting (GB) ----->>>

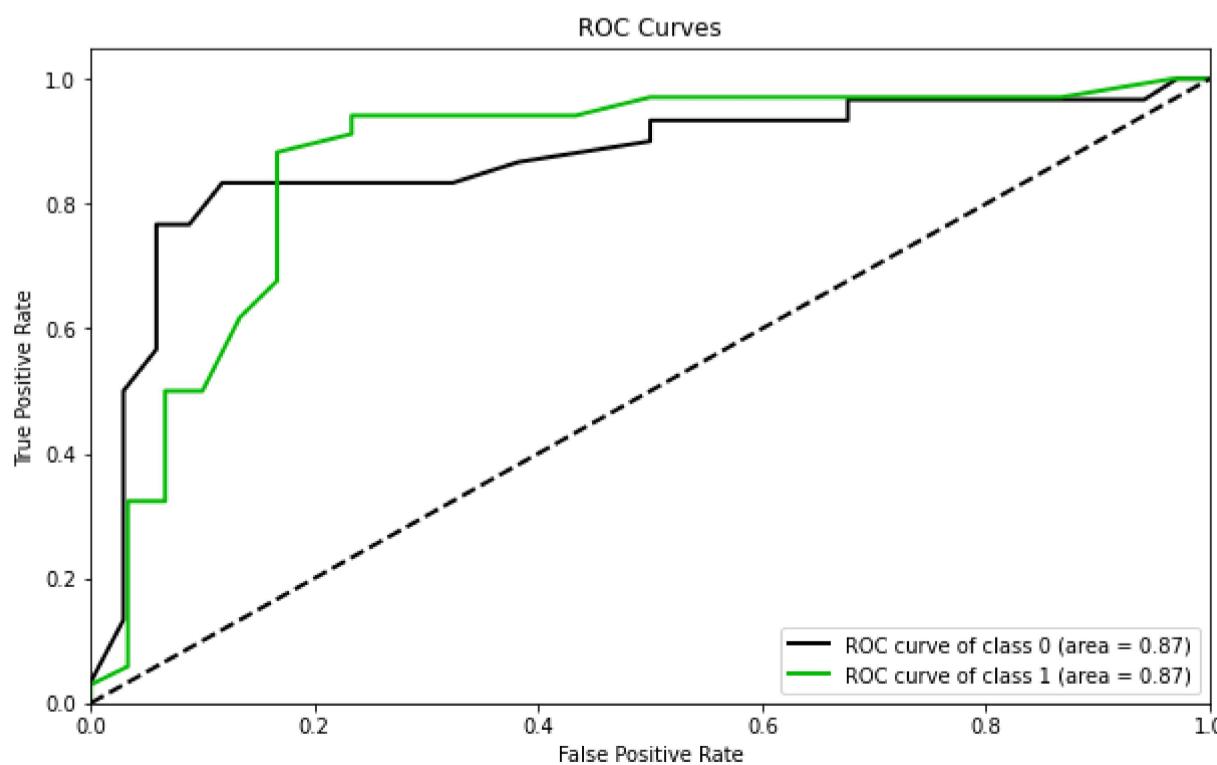
Accuracy = 85.9%  
F1 Score = 87.7%

Confusion Matrix:

```
[[23  7]
 [ 2 32]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.77	0.84	30
1	0.82	0.94	0.88	34
accuracy			0.86	64
macro avg	0.87	0.85	0.86	64
weighted avg	0.87	0.86	0.86	64



## 8. Extreme Gradient Boosting Classifier:

In [92]:

```
# Building Extreme Gradient Boosting Classifier

XGB_model = XGBClassifier().fit(Train_X_std, Train_Y)

param_dist = {
    "learning_rate" : [0.05,0.10,0.15,0.20,0.25,0.30],
    "max_depth" : [3, 4, 5, 6, 8, 10, 12, 15],
    "min_child_weight" : [1, 3, 5, 7],
    "gamma": [0.0, 0.1, 0.2, 0.3, 0.4],
    "colsample_bytree" : [0.3, 0.4, 0.5, 0.7]
}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(XGB_model, param_dist, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

XGB = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = XGB.predict(Test_X_std)
pred_prob = XGB.predict_proba(Test_X_std)
Classification_Summary(pred, pred_prob, 7)

plt.bar(Train_X_std.columns,XGB.feature_importances_)
plt.show()
```

[01:38:26] WARNING: C:/Users/Administrator/workspace/xgboost-win64\_release\_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval\_metric if you'd like to restore the old behavior.  
[01:38:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64\_release\_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval\_metric if you'd like to restore the old behavior.

<<<----- Evaluating Extreme Gradient Boosting (XGB) ----->>>

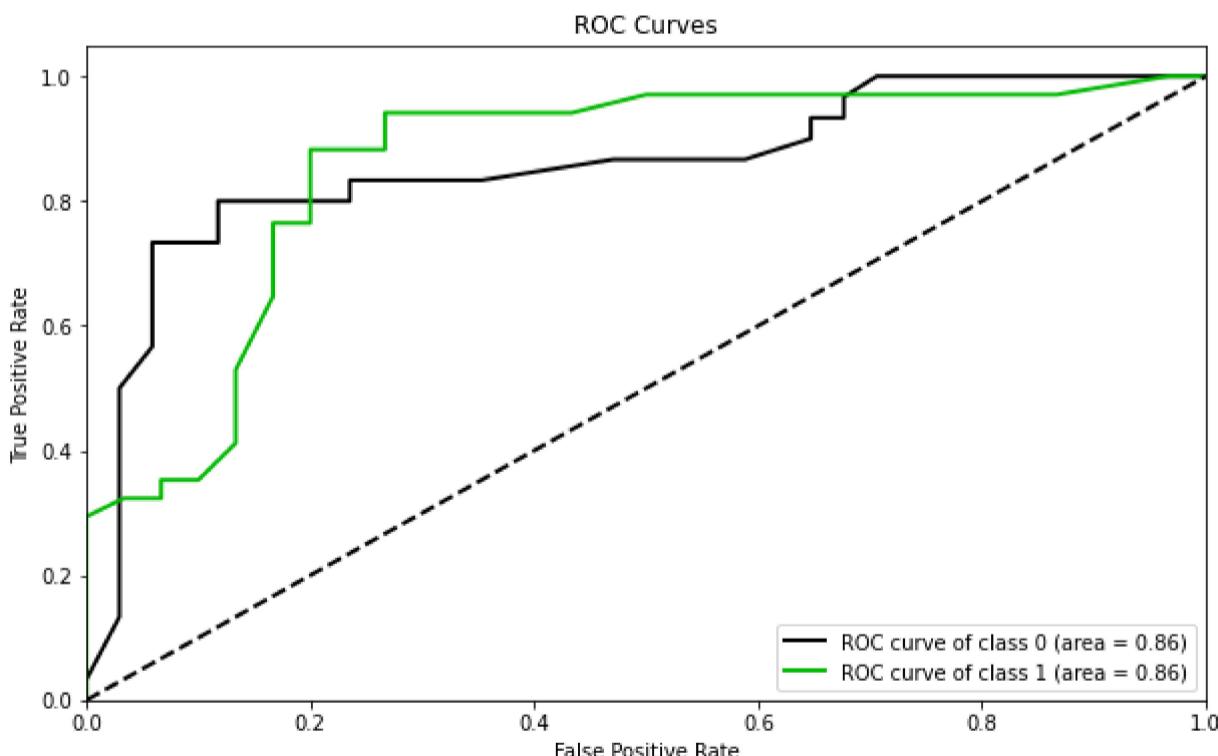
Accuracy = 82.8%  
F1 Score = 84.8999999999999%

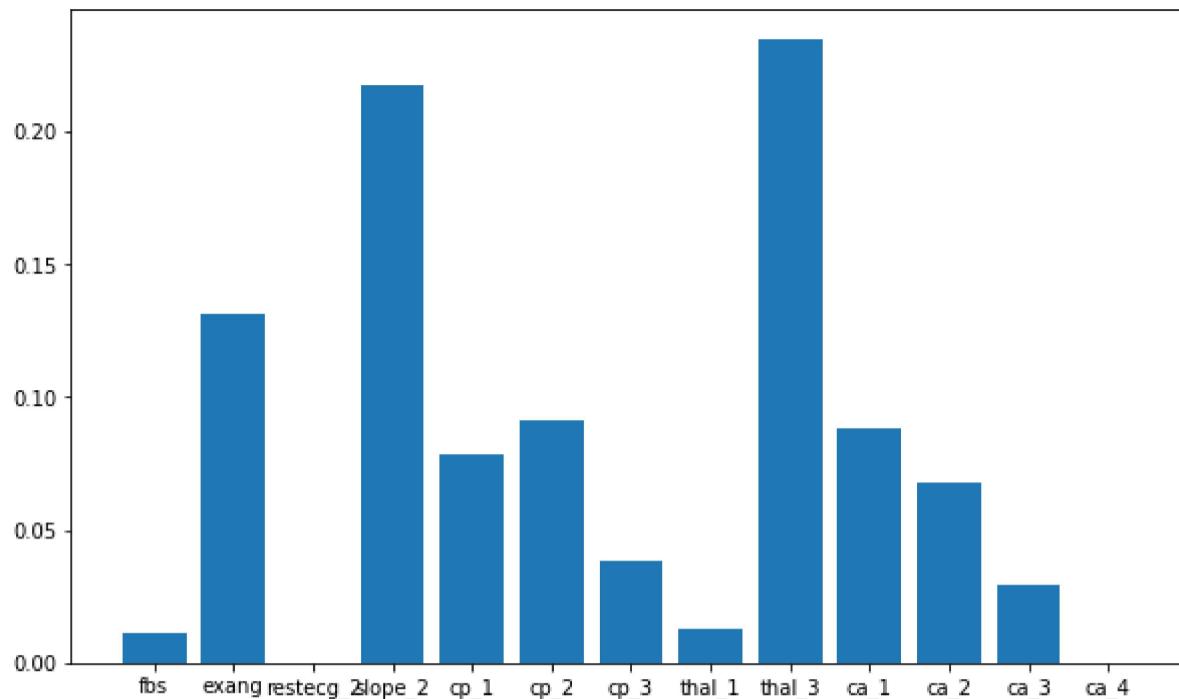
Confusion Matrix:

```
[[22  8]
 [ 3 31]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.73	0.80	30
1	0.79	0.91	0.85	34
accuracy			0.83	64
macro avg	0.84	0.82	0.82	64
weighted avg	0.83	0.83	0.83	64





In [93]: #Plotting Confusion-Matrix of all the predictive Models

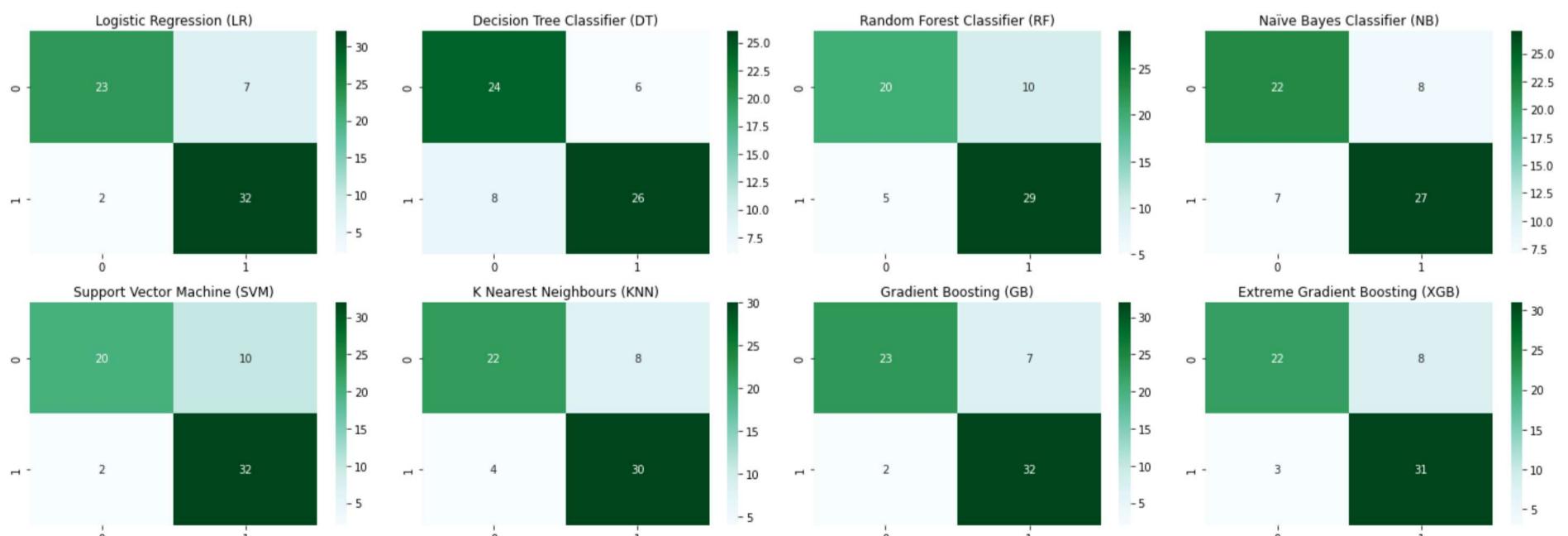
```

def plot_cm(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred, labels=np.unique(y_true))
    cm_sum = np.sum(cm, axis=1, keepdims=True)
    cm_perc = cm / cm_sum.astype(float) * 100
    annot = np.empty_like(cm).astype(str)
    nrows, ncols = cm.shape
    for i in range(nrows):
        for j in range(ncols):
            c = cm[i, j]
            p = cm_perc[i, j]
            if i == j:
                s = cm_sum[i]
                annot[i, j] = '%.1f%%\n%d/%d' % (p, c, s)
            elif c == 0:
                annot[i, j] = ''
            else:
                annot[i, j] = '%.1f%%\n%d' % (p, c)
    cm = pd.DataFrame(cm, index=np.unique(y_true), columns=np.unique(y_true))
    cm.columns=labels
    cm.index=labels
    cm.index.name = 'Actual'
    cm.columns.name = 'Predicted'
    #fig, ax = plt.subplots()
    sns.heatmap(cm, annot=annot, fmt='')# cmap= "GnBu"

def conf_mat_plot(all_models):
    plt.figure(figsize=[20,3.5*math.ceil(len(all_models)*len(labels)/14)])
    for i in range(len(all_models)):
        if len(labels)<=4:
            plt.subplot(2,4,i+1)
        else:
            plt.subplot(math.ceil(len(all_models)/3),3,i+1)
        pred = all_models[i].predict(Test_X_std)
        #plot_cm(Test_Y, pred)
        sns.heatmap(confusion_matrix(Test_Y, pred), annot=True, cmap='BuGn', fmt='%.0f') #vmin=0, vmax=5
        plt.title(Evaluation_Results.index[i])
    plt.tight_layout()
    plt.show()

conf_mat_plot([LR,DT,NB,SVM,KNN,GB,XGB])

```



In [94]: #Comparing all the models Scores

```
#plt.figure(figsize=[12,5])
```

```
sns.heatmap(Evaluation_Results, annot=True, vmin=70, vmax=90, cmap='BuGn', fmt='.1f')
plt.show()
```



**Insights:** For the current problem statement, it is more important to focus on the Recall score. We can note from the above heatmap that the Boosting & SVM Models Performed well on the current dataset...

## 7. Project Outcomes & Conclusions

Here are some of the key outcomes of the project:

- The Dataset was quiet totalling around 300 samples & after preprocessing 5.1% of the datasamples were dropped.
- The samples were slightly imbalanced after processing, hence SMOTE Technique was applied on the data to balance the classes, adding 9.2% more samples to the dataset.
- Visualising the distribution of data & their relationships, helped us to get some insights on the relationship between the feature-set.
- Feature Selection/Elimination was carried out and appropriate features were shortlisted.
- Testing multiple algorithms with fine-tuning hyperparamters gave us some understanding on the model performance for various algorithms on this specific dataset.
- The SVM, Boosting & Random Forest Classifier performed exceptionally well on the current dataset, considering Recall Score as the key-metric.
- Yet it wise to also consider simpler model like Logistic Regression as it is more generalisable & is computationally less expensive, but comes at the cost of slight misclassifications.

In [ ]:

-----><<----- THE END ----->