```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from datetime import datetime

from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
from sklearn.model_selection import GridSearchCV,train_test_split,cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, auc
import os
import warnings
warnings.filterwarnings('ignore')
```

```
data=pd.read_csv('/content/heart.csv')
```

```
#Now,I will check null on all data and If data has null, I will sum of null data's. In this w
print('Data Sum of Null Values \n')
```



```
Data Sum of Null Values

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
```

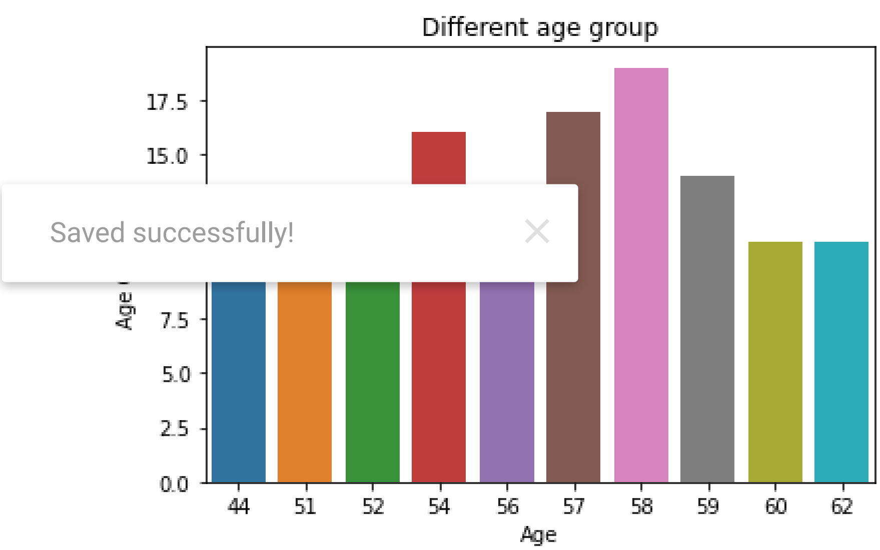
```
#all rows control for null values
print(data.isnull().values.any())
```

```
False
```

```
# TODO:- Age Analysis
print(data.age.value_counts()[:10])
#data age show value counts for age least 10
```

```
58    19
57    17
54    16
59    14
52    13
51    12
62    11
44    11
60    11
56    11
Name: age, dtype: int64
```

```
sns.barplot(x=data.age.value_counts()[:10].index,y=data.age.value_counts()[:10].values)
plt.xlabel('Age')
plt.ylabel('Age Counter')
plt.title('Different age group')
plt.show()
```



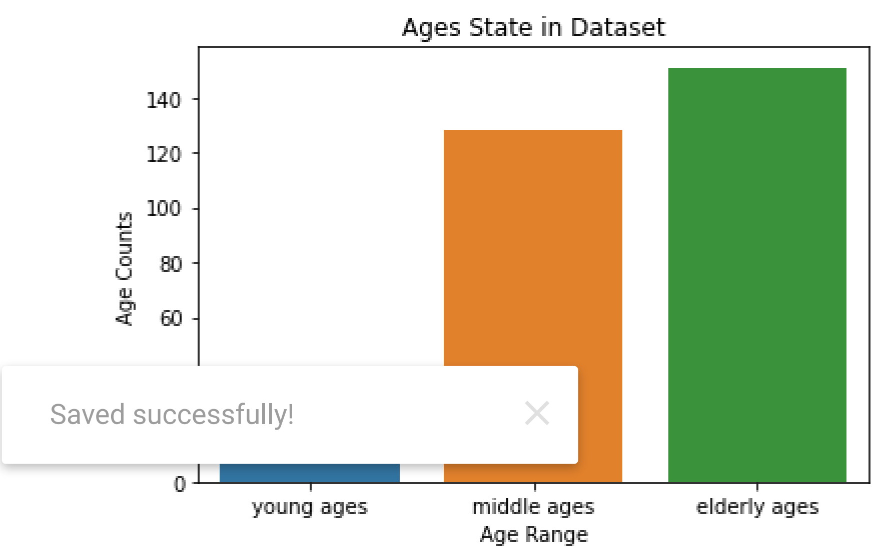

```
#firstly find min and max ages
minage=min(data.age)
maxage=max(data.age)
meanage=data.age.mean()
print('Min Age :',minage)
print('Max Age :',maxage)
print('Mean Age :',meanage)
```

```
Min Age : 29
Max Age : 77
Mean Age : 54.366336633663366
```

```
young_ages=data[(data.age>=29)&(data.age<40)]
middle_ages=data[(data.age>=40)&(data.age<55)]
elderly_ages=data[(data.age>55)]
print('Young Ages :',len(young_ages))
print('Middle Ages :',len(middle_ages))
print('Elderly Ages :',len(elderly_ages))
```

```
Young Ages : 16
Middle Ages : 128
Elderly Ages : 151
```

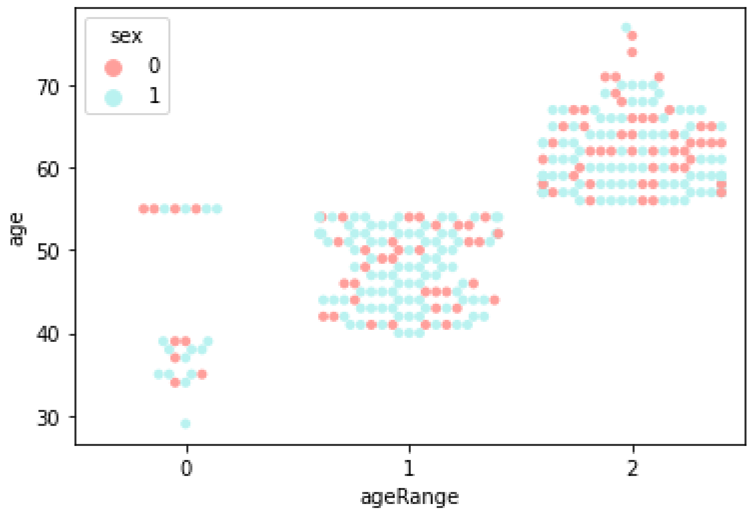
```
sns.barplot(x=['young ages','middle ages','elderly ages'],y=[len(young_ages),len(middle_ages)
plt.xlabel('Age Range')
plt.ylabel('Age Counts')
plt.title('Ages State in Dataset')
plt.show()
```



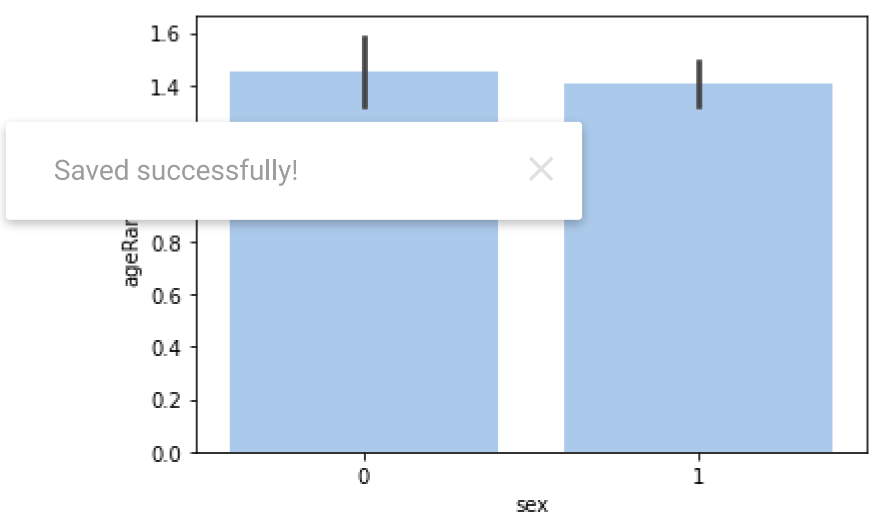

```
data['ageRange']=0
youngage_index=data[(data.age>=29)&(data.age<40)].index
middleage_index=data[(data.age>=40)&(data.age<55)].index
elderlyage_index=data[(data.age>55)].index


for index in elderlyage_index:
    data.loc[index,'ageRange']=2

for index in middleage_index:
    data.loc[index,'ageRange']=1
```

```
for index in youngage_index:
```

```
# Draw a categorical scatterplot to show each observation
sns.swarmplot(x="ageRange", y="age",hue='sex',
              palette=["r", "c", "y"], data=data)
plt.show()
```



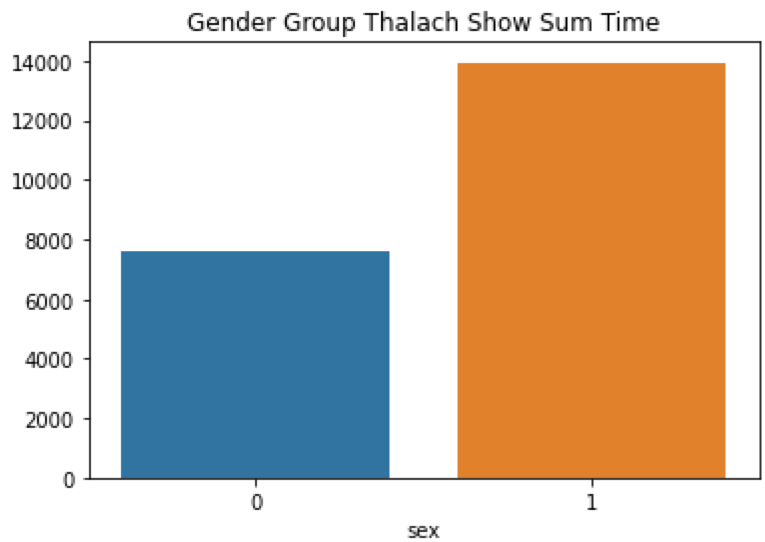

```
# Plot the total crashes
sns.set_color_codes("pastel")
sns.barplot(y="ageRange", x="sex", data=data,
            label="Total", color="b")
plt.show()
```



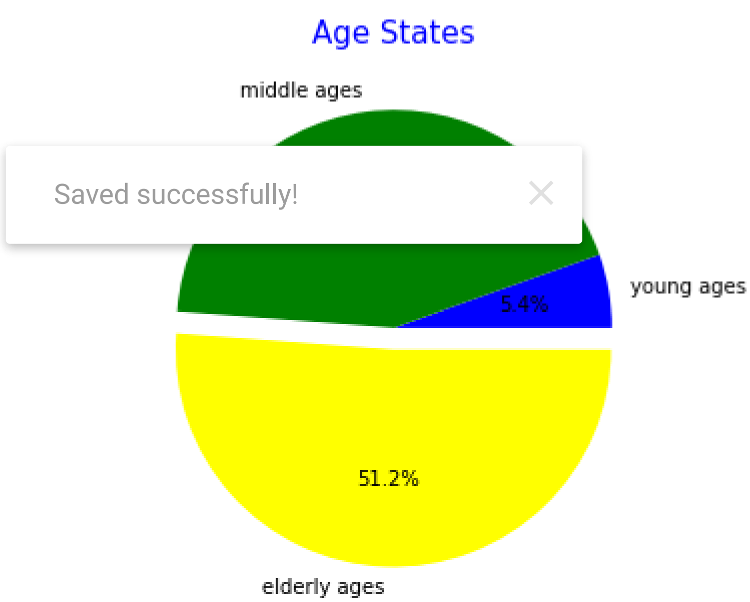

```
print(elderly_ages.groupby(elderly_ages['sex'])['thalach'].agg('sum'))
```

```
sex
0     7578
1    13948
Name: thalach, dtype: int64
```

```
sns.barplot(x=elderly_ages.groupby(elderly_ages['sex'])['thalach'].agg('sum').index,y=elderly_
plt.title("Gender Group Thalach Show Sum Time")
plt.show()
```




```
colors = ['blue','green','yellow']
explode = [0,0,0.1]
plt.figure(figsize = (5,5))
#plt.pie([target_0_agerang_0,target_1_agerang_0], explode=explode, labels=['Target 0 Age Rang
plt.pie([len(young_ages),len(middle_ages),len(elderly_ages)],labels=['young ages','middle age
plt.title('Age States',color = 'blue',fontsize = 15)
plt.show()
```



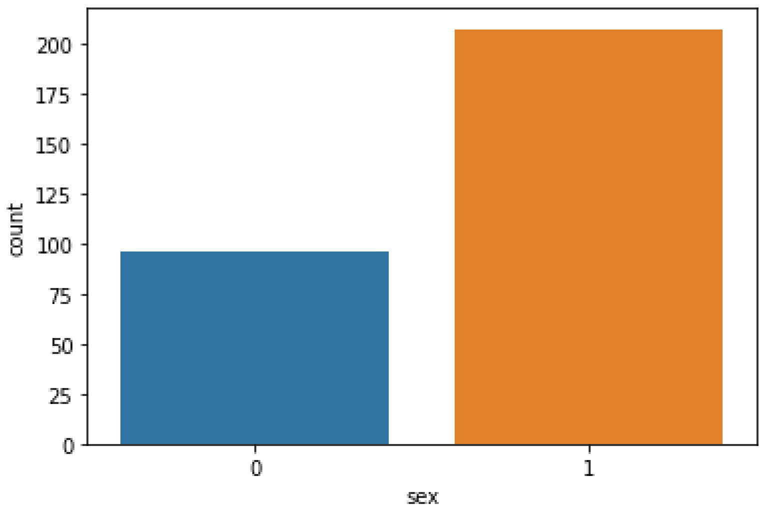

```
# TODO :- Sex (Gender) Analysis
```

```
print(data.sex.value_counts())
```

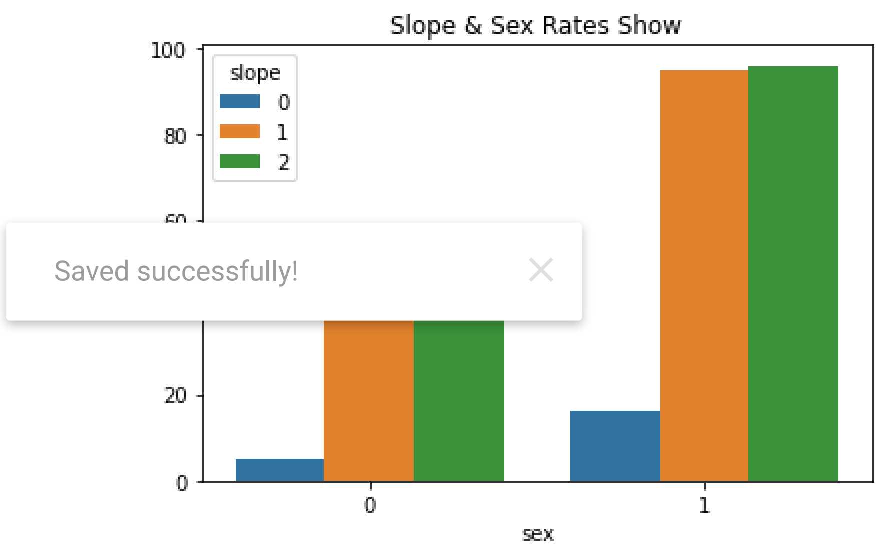
```
1    207
0     96
```

```
Name: sex, dtype: int64
```

```
#Sex (1 = male; 0 = female)
sns.countplot(data.sex)
plt.show()
```




```
sns.countplot(data.sex,hue=data.slope)
plt.title('Slope & Sex Rates Show')
plt.show()
```



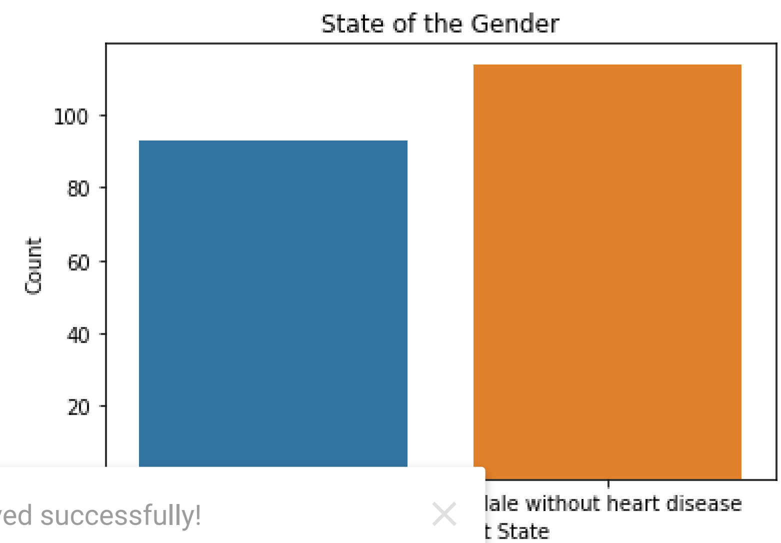

```
total_genders_count=len(data.sex)
male_count=len(data[data['sex']==1])
female_count=len(data[data['sex']==0])
print('Total Genders :',total_genders_count)
print('Male Count    :',male_count)
print('Female Count  :',female_count)
```

```
Total Genders : 303
Male Count    : 207
Female Count  : 96
```

```
#Percentage ratios
print("Male State: {:.2f}%".format((male_count / (total_genders_count)*100)))
print("Female State: {:.2f}%".format((female_count / (total_genders_count)*100)))
```

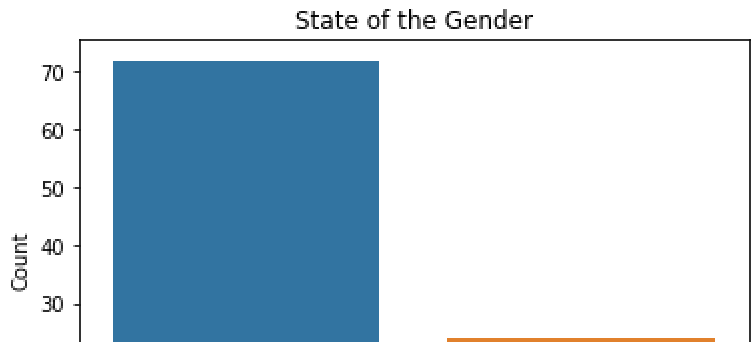
```
Male State: 68.32%
Female State: 31.68%
```

```
# Male State & target 1 & 0
male_andtarget_on=len(data[(data.sex==1)&(data['target']==1)])
male_andtarget_off=len(data[(data.sex==1)&(data['target']==0)])
####
sns.barplot(x=['Male with heart disease','Male without heart disease'],y=[male_andtarget_on,m
plt.xlabel('Male and Target State')
plt.ylabel('Count')
plt.title('State of the Gender')
plt.show()
```



Saved successfully!

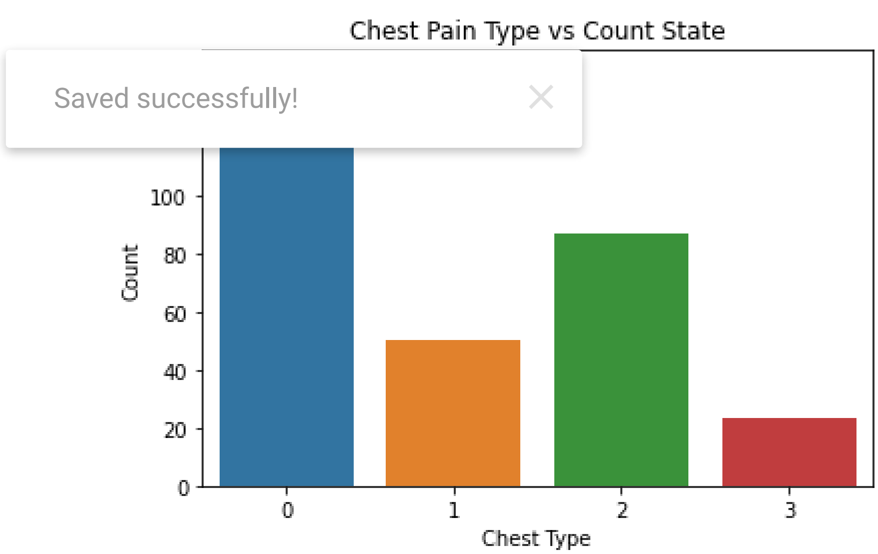
```
#Female State & target 1 & 0
female_andtarget_on=len(data[(data.sex==0)&(data['target']==1)])
female_andtarget_off=len(data[(data.sex==0)&(data['target']==0)])
####
sns.barplot(x=['Female with heart disease','Female without heart disease'],y=[female_andtarge
plt.xlabel('Female and Target State')
plt.ylabel('Count')
plt.title('State of the Gender')
plt.show()
```



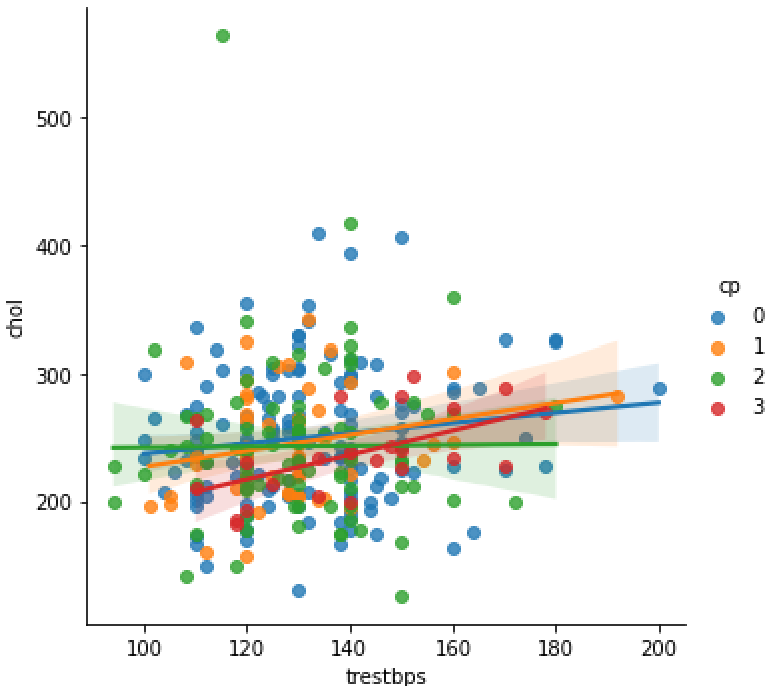

```
# TODO:- Chest Pain Type Analysis

print(data.cp.value_counts())
```

```
0    143
2     87
1     50
3     23
Name: cp, dtype: int64
```

```
sns.countplot(data.cp)
plt.xlabel('Chest Type')
plt.ylabel('Count')
plt.title('Chest Pain Type vs Count State')
plt.show()
#0 status at least
#1 condition slightly distressed
#2 condition medium problem
#3 condition too bad
```




```
# Show the results of a linear regression within each dataset
sns.lmplot(x="trestbps", y="chol",data=data,hue="cp")
plt.show()
```




```
data.describe(include =[np.number])
```

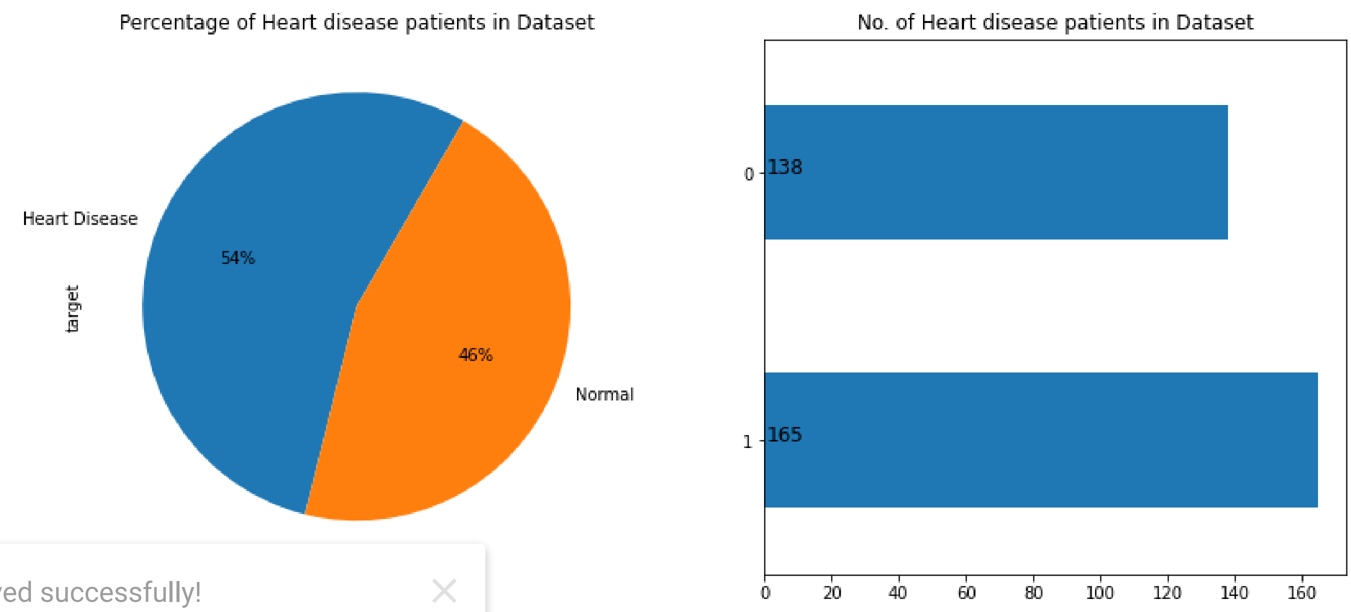
	age	sex	cp	trestbps	chol	fbs	restecg
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000
[illegible]	[illegible]	[illegible]	1.000000	130.000000	240.000000	0.000000	1.000000
[illegible]	[illegible]	[illegible]	2.000000	140.000000	274.500000	0.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000


```
dt= data.copy()
dt.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2

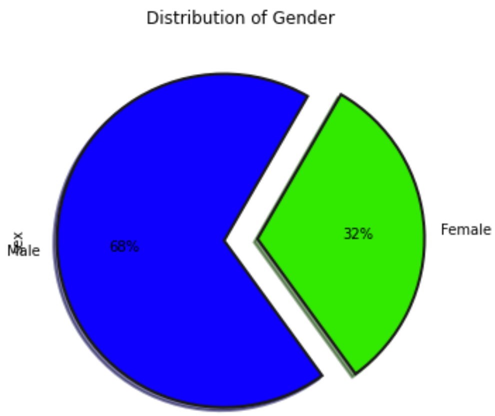
```
dt=dt.rename(columns={'cp':'chest_pain_type','restecg':'rest_ecg','slope':'st_slope'})
```

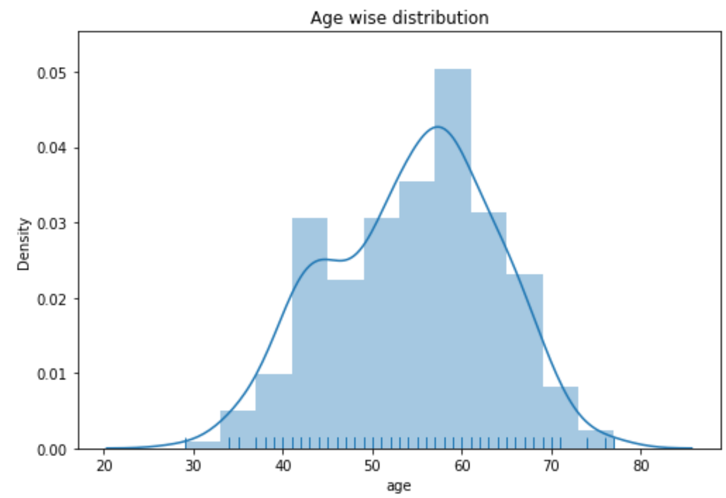
```
# Plotting attrition of employees
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, sharey=False, figsize=(14,6))

ax1 = dt['target'].value_counts().plot.pie( x="Heart disease" ,y ='no.of patients',
                   autopct = "%1.0f%%",labels=["Heart Disease","Normal"], startangle = 60,ax=
ax1.set(title = 'Percentage of Heart disease patients in Dataset')

ax2 = dt["target"].value_counts().plot(kind="barh" ,ax =ax2)
for i,j in enumerate(dt["target"].value_counts().values):
    ax2.text(.5,i,j,fontsize=12)
ax2.set(title = 'No. of Heart disease patients in Dataset')
plt.show()
```



Saved successfully!

```
plt.figure(figsize=(18,12))
plt.subplot(221)
dt["sex"].value_counts().plot.pie(autopct = "%1.0f%%",colors = sns.color_palette("prism",5),s
wedgeprops={"linewidth":2,"edgecolor":"k"},explode=[.1,.1],shadow =True)
plt.title("Distribution of Gender")
plt.subplot(222)
ax= sns.distplot(dt['age'], rug=True)
plt.title("Age wise distribution")
plt.show()
```

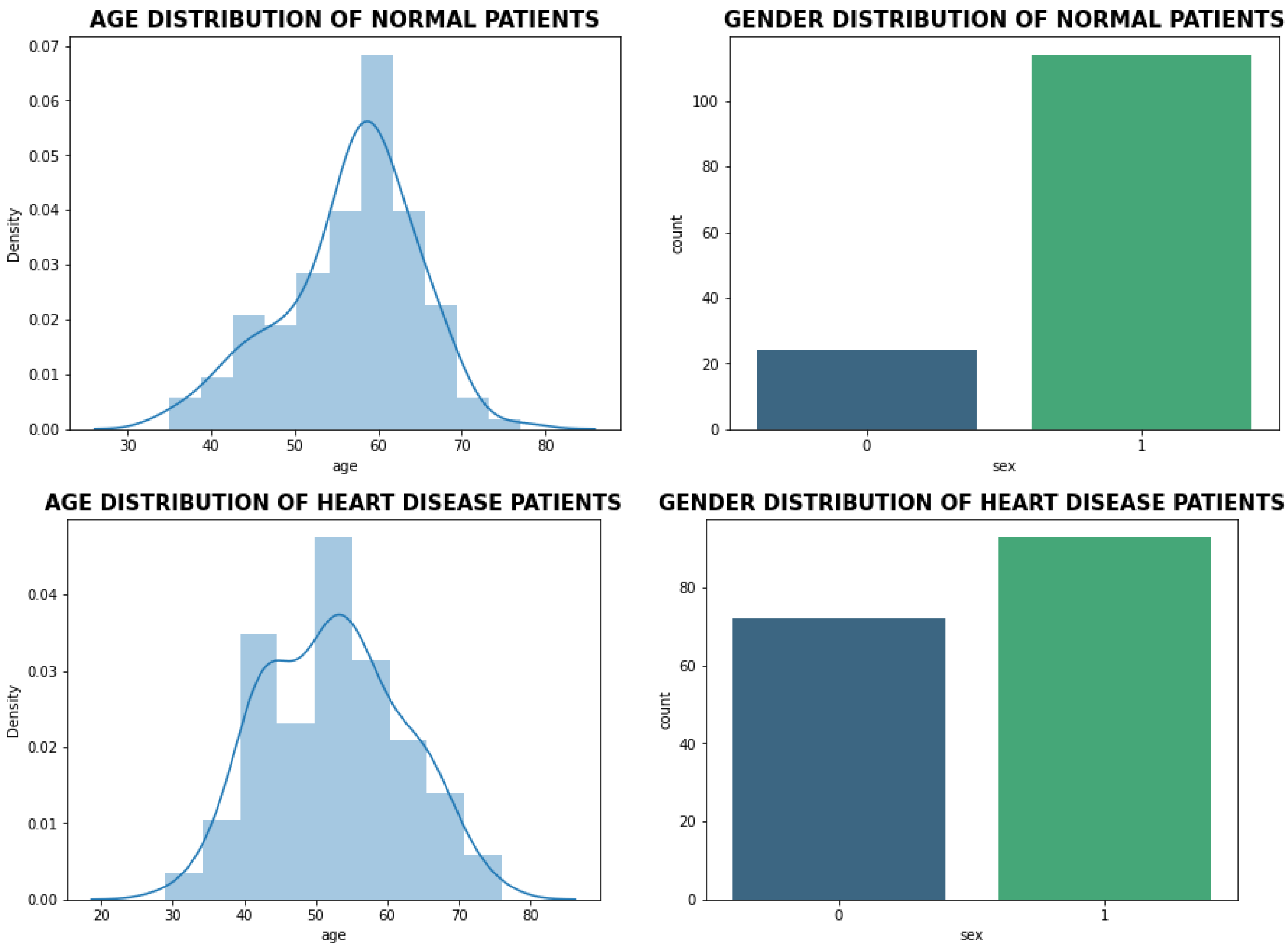





```
# creating separate df for normal and heart patients

attr_1=dt[dt['target']==1]

attr_0=dt[dt['target']==0]

# plotting normal patients
fig = plt.figure(figsize=(15,5))
ax1 = plt.subplot2grid((1,2),(0,0))
sns.distplot(attr_0['age'])
plt.title('AGE DISTRIBUTION OF NORMAL PATIENTS', fontsize=15, weight='bold')

ax1 = plt.subplot2grid((1,2),(0,1))
='viridis')
RMAL PATIENTS', fontsize=15, weight='bold' )
```

Saved successfully!

```
#plotting heart patients

fig = plt.figure(figsize=(15,5))
ax1 = plt.subplot2grid((1,2),(0,0))
sns.distplot(attr_1['age'])
plt.title('AGE DISTRIBUTION OF HEART DISEASE PATIENTS', fontsize=15, weight='bold')

ax1 = plt.subplot2grid((1,2),(0,1))
sns.countplot(attr_1['sex'], palette='viridis')
plt.title('GENDER DISTRIBUTION OF HEART DISEASE PATIENTS', fontsize=15, weight='bold' )
plt.show()
```




```
dt.chest_pain_type.unique()
```

```
array([3, 2, 1, 0])
```

```
print(dt.chest_pain_type.unique())
print(dt.rest_ecg.unique())
print(dt.st_slope.unique())
```

```
[3 2 1 0]
[0 1 2]
[0 2 1]
```

```
# converting features to categorical features

dt['chest_pain_type'][dt['chest_pain_type'] == 0] = 'typical angina'
dt['chest_pain_type'][dt['chest_pain_type'] == 1] = 'atypical angina'
dt['chest_pain_type'][dt['chest_pain_type'] == 2] = 'non-anginal pain'
```

```
dt['chest_pain_type'][dt['chest_pain_type'] == 3] = 'asymptomatic'



dt['rest_ecg'][dt['rest_ecg'] == 0] = 'normal'
dt['rest_ecg'][dt['rest_ecg'] == 1] = 'ST-T wave abnormality'
dt['rest_ecg'][dt['rest_ecg'] == 2] = 'left ventricular hypertrophy'



dt['st_slope'][dt['st_slope'] == 0] = 'upsloping'
dt['st_slope'][dt['st_slope'] == 1] = 'flat'
dt['st_slope'][dt['st_slope'] == 2] = 'downsloping'

dt["sex"] = dt.sex.apply(lambda  x:'male' if x==1 else 'female')
```

```
dt.chest_pain_type.unique()
```

```
array(['asymptomatic', 'non-anginal pain', 'atypical angina',
       'typical angina'], dtype=object)
```

```
dt.head()
```

	age	sex	chest_pain_type	trestbps	chol	fbs	rest_ecg	thalach	exang	oldpeak
0	63	male	asymptomatic	145	233	1	normal	150	0	2.3
1	37	male	non-anginal pain	130	250	0	ST-T wave abnormality	187	0	3.5
2	41	female	atypical angina	130	204	0	normal	172	0	1.4
[illegible]	[illegible]	[illegible]	[illegible]	120	236	0	ST-T wave abnormality	178	0	0.8

Saved successfully!

```
dt['rest_ecg'].value_counts()
```

```
ST-T wave abnormality           152
normal                          147
left ventricular hypertrophy      4
Name: rest_ecg, dtype: int64
```

```
dt['chest_pain_type'].value_counts()
```

```
typical angina      143
non-anginal pain     87
atypical angina      50
asymptomatic         23
Name: chest_pain_type, dtype: int64
```

```
dt['st_slope'].value_counts()
```

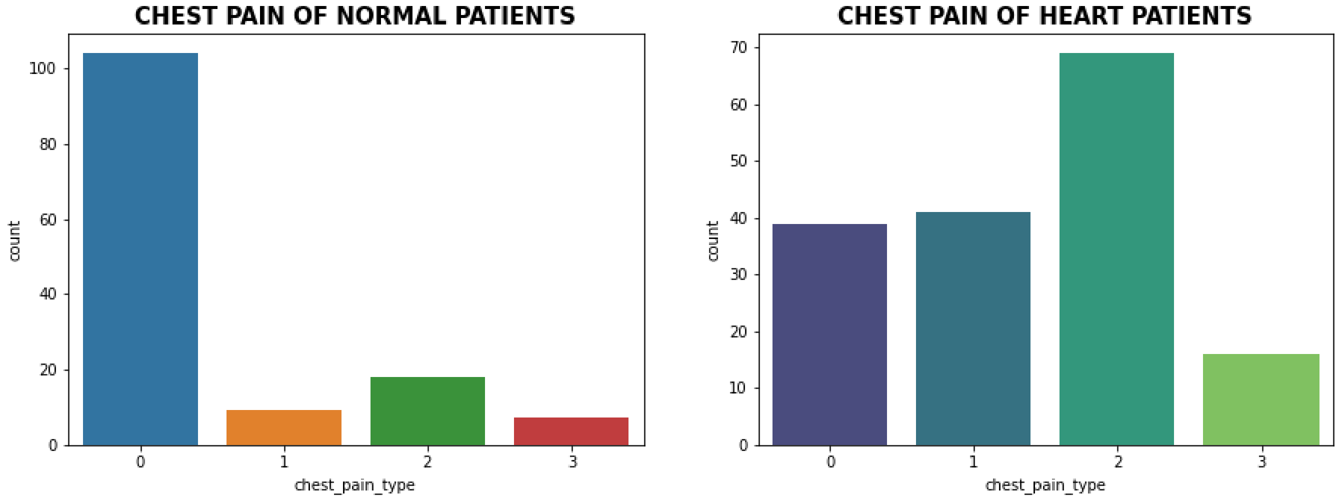
```
downsloping    142
flat           140
upsloping       21
Name: st_slope, dtype: int64
```

```
dt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              303 non-null    int64
 1   sex              303 non-null    object
 2   chest_pain_type  303 non-null    object
 3   trestbps         303 non-null    int64
 4   chol             303 non-null    int64
 5   fbs              303 non-null    int64
 6   rest_ecg         303 non-null    object
 7   thalach          303 non-null    int64
 8   exang            303 non-null    int64
 9   oldpeak          303 non-null    float64
 10  st_slope         303 non-null    object
                            ull    int64
                            ull    int64
                            ull    int64
 14  ageRange         303 non-null    int64
dtypes: float64(1), int64(10), object(4)
memory usage: 35.6+ KB
```

Saved successfully!

```
# plotting normal patients
fig = plt.figure(figsize=(15,5))
ax1 = plt.subplot2grid((1,2),(0,0))
sns.countplot(attr_0['chest_pain_type'])
plt.title('CHEST PAIN OF NORMAL PATIENTS', fontsize=15, weight='bold')

#plotting heart patients
ax1 = plt.subplot2grid((1,2),(0,1))
sns.countplot(attr_1['chest_pain_type'], palette='viridis')
plt.title('CHEST PAIN OF HEART PATIENTS', fontsize=15, weight='bold' )
plt.show()
```

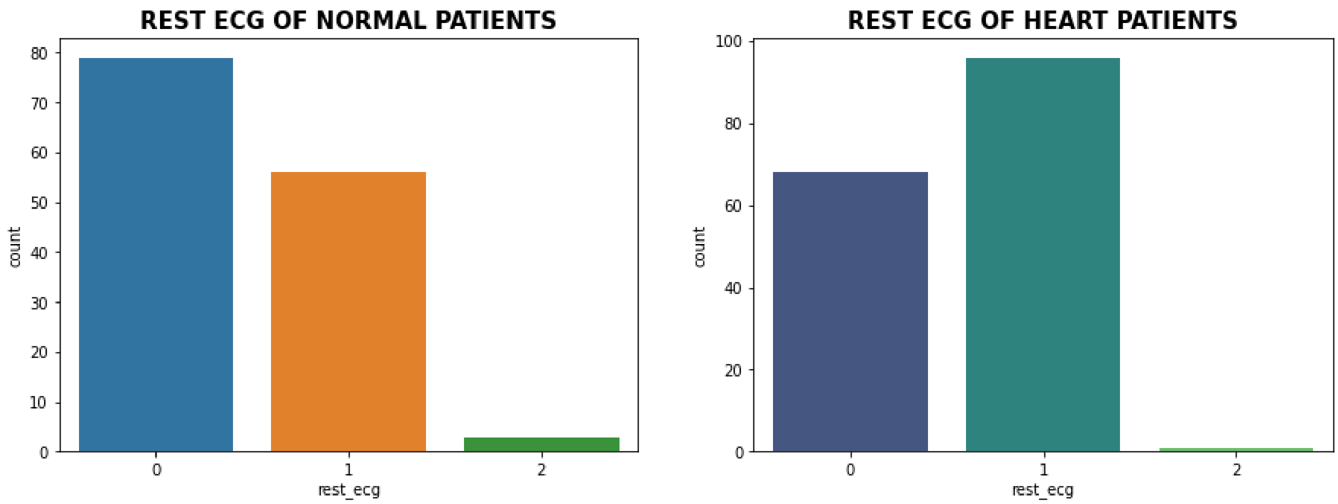



```
#Exploring the Heart Disease patients based on Chest Pain Type
plot_criteria= ['chest_pain_type', 'target']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(dt[plot_criteria[0]], dt[plot_criteria[1]], normalize='columns') * 100,2))
```

target	0	1
chest_pain_type		
asymptomatic	5.070000	9.700000
atypical angina	6.520000	24.850000
[illegible]	[illegible]	[illegible].820000
typical angina	[illegible]	[illegible].640000

Saved successfully!

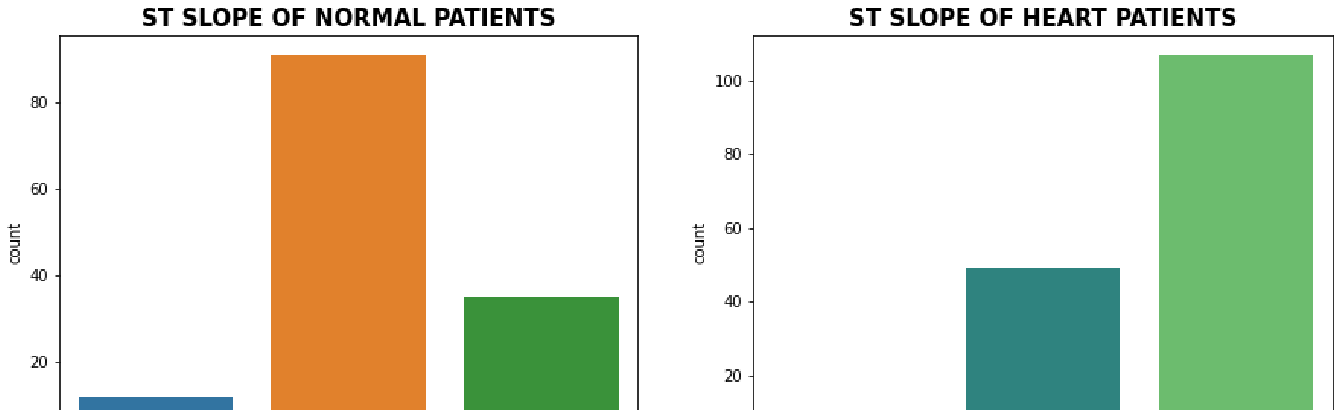
```
# plotting normal patients
fig = plt.figure(figsize=(15,5))
ax1 = plt.subplot2grid((1,2),(0,0))
sns.countplot(attr_0['rest_ecg'])
plt.title('REST ECG OF NORMAL PATIENTS', fontsize=15, weight='bold')

#plotting heart patients
ax1 = plt.subplot2grid((1,2),(0,1))
sns.countplot(attr_1['rest_ecg'], palette='viridis')
plt.title('REST ECG OF HEART PATIENTS', fontsize=15, weight='bold' )
plt.show()
```




```
#Exploring the Heart Disease patients based on REST ECG
plot_criteria= ['rest_ecg', 'target']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(dt[plot_criteria[0]], dt[plot_criteria[1]], normalize='columns') * 100,2))
```

target	0	1
rest_ecg		
ST-T wave abnormality	40.580000	58.180000
left ventricular hypertrophy	2.170000	0.610000
normal	57.250000	41.210000


```
ax1 = plt.subplot2grid((1,2),(0,0))
sns.countplot(attr_0['st_slope'])
plt.title('ST SLOPE OF NORMAL PATIENTS', fontsize=15, weight='bold')

#plotting heart patients
ax1 = plt.subplot2grid((1,2),(0,1))
sns.countplot(attr_1['st_slope'], palette='viridis')
plt.title('ST SLOPE OF HEART PATIENTS', fontsize=15, weight='bold' )
plt.show()
```




```
# summary statistics of categorical columns
dt.describe(include =[np.object])
```

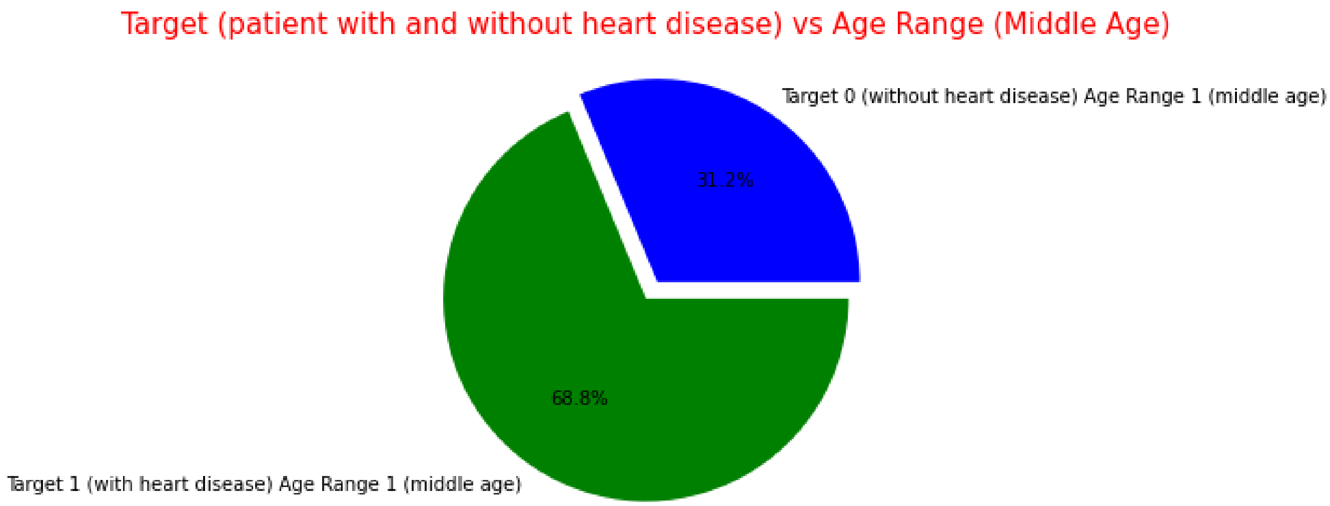
	sex	chest_pain_type	rest_ecg	st_slope
count	303	303	303	303
unique	2	4	3	3
top	male	typical angina	ST-T wave abnormality	downsloping
freq	207	143	152	142

```
# TODO:- Age Range Analysis

target_0_agerang_0=len(data[(data.target==0)&(data.ageRange==0)])
target_1_agerang_0=len(data[(data.target==1)&(data.ageRange==0)])

colors = ['blue','green']
```



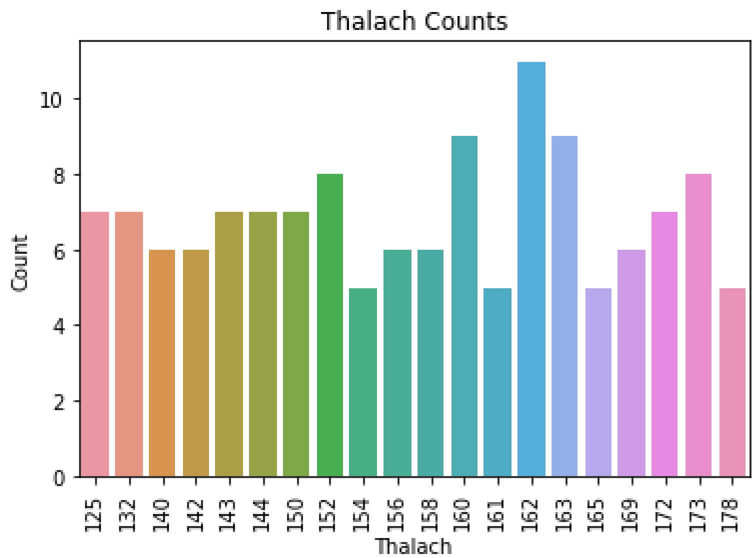
```
plt.pie([target_0_agerang_0,target_1_agerang_0], explode=explode, labels=['Target 0 (without
plt.title('Target(patient with and without heart disease) vs Age Range (Young Age) ',color =
plt.show()
```


```
target_0_agerang_1=len(data[(data.target==0)&(data.ageRange==1)])
target_1_agerang_1=len(data[(data.target==1)&(data.ageRange==1)])
colors = ['blue','green']
explode = [0.1,0]
plt.figure(figsize = (5,5))
plt.pie([target_0_agerang_1,target_1_agerang_1], explode=explode, labels=['Target 0 (without
plt.title('Target (patient with and without heart disease) vs Age Range (Middle Age)',color =
plt.show()
```



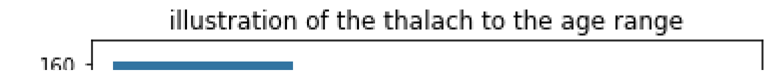

```
target_0_agerang_2=len(data[(data.target==0)&(data.ageRange==2)])
[illegible]rget==1)&(data.ageRange==2)])
[illegible]
explode = [0,0.1]
plt.figure(figsize = (5,5))
plt.pie([target_0_agerang_2,target_1_agerang_2], explode=explode, labels=['Target 0 (without
plt.title('Target (patient with and without heart disease) vs Age Range (Elderly Age) ',color
plt.show()
```


```
sns.barplot(x=data.thalach.value_counts()[:20].index,y=data.thalach.value_counts()[:20].value
plt.xlabel('Thalach')
plt.ylabel('Count')
plt.title('Thalach Counts')
plt.xticks(rotation=90)
plt.show()
```

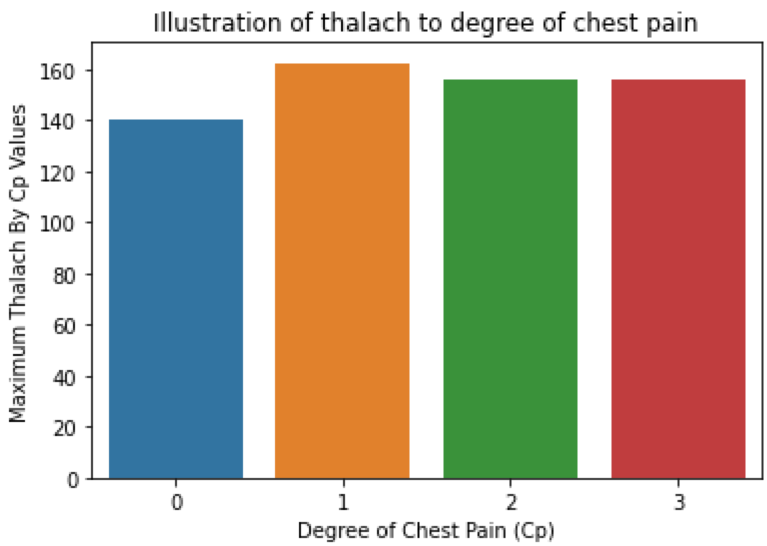



```
age_range_thalach=data.groupby('ageRange')['thalach'].mean()
sns.barplot(x=age_range_thalach.index,y=age_range_thalach.values)
ange')
ch to the age range')
plt.show()
#As shown in this graph, this rate decreases as the heart rate
#is faster and in old age areas.
```

Saved successfully!




```
cp_thalach=data.groupby('cp')['thalach'].mean()
sns.barplot(x=cp_thalach.index,y=cp_thalach.values)
plt.xlabel('Degree of Chest Pain (Cp)')
plt.ylabel('Maximum Thalach By Cp Values')
plt.title('Illustration of thalach to degree of chest pain')
plt.show()
#As seen in this graph, it is seen that the heart rate is less
#when the chest pain is low. But in cases where chest pain is
#1, it is observed that the area is more. 2 and 3 were found to
#be of the same degree.
```




```
# TODO :- Thal Analysis
```

Saved successfully!

```
2    166
3    117
1     18
0      2
Name: thal, dtype: int64
```

```
#Target 1
a=len(data[(data['target']==1)&(data['thal']==0)])
b=len(data[(data['target']==1)&(data['thal']==1)])
c=len(data[(data['target']==1)&(data['thal']==2)])
d=len(data[(data['target']==1)&(data['thal']==3)])
print('Target 1 Thal 0: ',a)
print('Target 1 Thal 1: ',b)
print('Target 1 Thal 2: ',c)
print('Target 1 Thal 3: ',d)

#so,Apparently, there is a rate at Thal 2.Now, draw graph
```

```
print('*'*50)
#Target 0
e=len(data[(data['target']==0)&(data['thal']==0)])
f=len(data[(data['target']==0)&(data['thal']==1)])
g=len(data[(data['target']==0)&(data['thal']==2)])
h=len(data[(data['target']==0)&(data['thal']==3)])
print('Target 0 Thal 0: ',e)
print('Target 0 Thal 1: ',f)
print('Target 0 Thal 2: ',g)
print('Target 0 Thal 3: ',h)
```

```
Target 1 Thal 0:  1
Target 1 Thal 1:  6
Target 1 Thal 2:  130
Target 1 Thal 3:  28
**************************************************
Target 0 Thal 0:  1
Target 0 Thal 1:  12
Target 0 Thal 2:  36
Target 0 Thal 3:  89
```

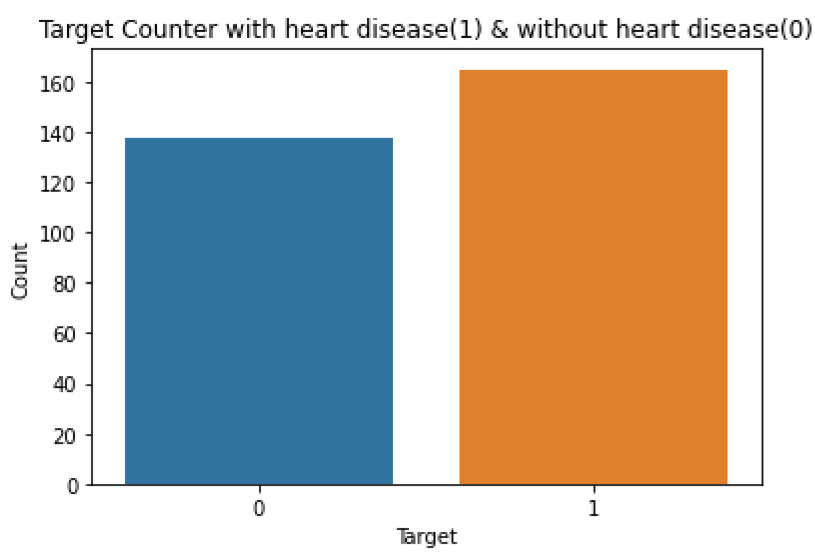
```
# TODO :- Target Analysis

print(data.target.unique())
#only two values are shown.
#A value of 1 is the value of patient 0
```

```
[1 0]
```

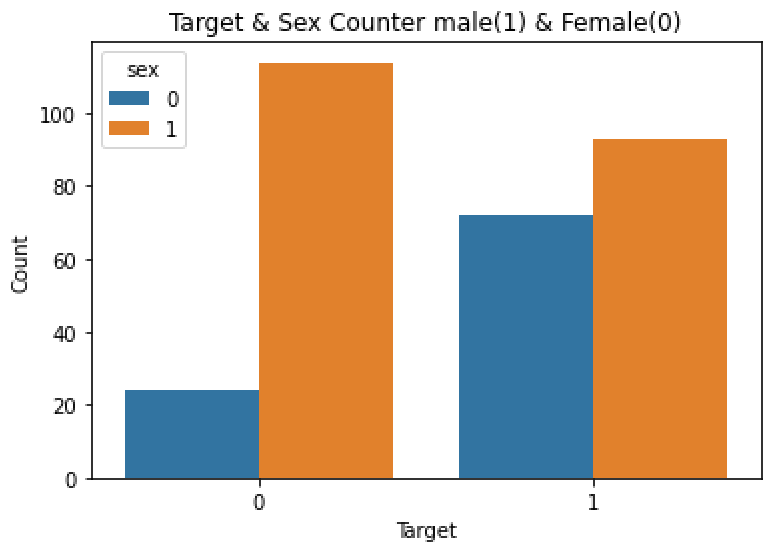
```
sns.countplot(data.target)
plt.xlabel('Target')
```



```
disease(1) & without heart disease(0)')
plt.show()
```



```
sns.countplot(data.target,hue=data.sex)
plt.xlabel('Target')
plt.ylabel('Count')
plt.title('Target & Sex Counter male(1) & Female(0)')
plt.show()
```




```
#determine the age ranges of patients with and without sickness and make analyzes about them
age_counter_target_1=[]
age_counter_target_0=[]
for age in data.age.unique():
    age_counter_target_1.append(len(data[(data['age']==age)&(data.target==1)]))
    age_counter_target_0.append(len(data[(data['age']==age)&(data.target==0)]))

#now, draw show on graph

e_counter_target_1,color='blue',label='Patient with heart
plt.scatter(x=data.age.unique(),y=age_counter_target_0,color='red',label='Patient without hea
plt.legend(loc='upper right',frameon=True)
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Target 0 (Patient without heart disease) & Target 1 (Patient with heart disease) S
plt.show()
```


```
df=data.copy()
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2

```
from tensorflow.keras.utils import to_categorical
# one hot encode
encoded1 = pd.DataFrame(to_categorical(df.restecg),columns=['restecg0','restecg1','restecg2']
encoded2 = pd.DataFrame(to_categorical(df.cp),columns=['cp0','cp1','cp2','cp3'])
encoded3 = pd.DataFrame(to_categorical(df.thal),columns=['thal0','thal1','thal2','thal3'])
encoded4 = pd.DataFrame(to_categorical(df.ca),columns=['ca0','ca1','ca2','ca3','ca4'])
encoded5 = pd.DataFrame(to_categorical(df.slope),columns=['slope0','slope1','slope2'])

df = pd.concat([df,encoded1,encoded2,encoded3,encoded4,encoded5],axis=1).drop(['restecg','cp'
df.head()
```

Saved successfully!

					bs	thalach	exang	oldpeak	target	ageRange	restecg0	r
					1	150	0	2.3	1	2	1.0	
1	37	1	130	250	0	187	0	3.5	1	0	0.0	
2	41	0	130	204	0	172	0	1.4	1	1	1.0	
3	56	1	120	236	0	178	0	0.8	1	2	0.0	
4	57	0	120	354	0	163	1	0.6	1	2	0.0	

```
X_train, X_test, y_train, y_test=train_test_split(
    df.drop(['target'], axis=1),
    df[['target']],
    test_size=0.3,
    random_state=41)
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(212, 28)
(91, 28)
```

```
for column in X_train.columns:

    df_train1 = X_train[(y_train.target==0) & (X_train[column]<np.mean(X_train.loc[y_train.ta
    df_test1 = X_test[(y_test.target==0) & (X_test[column]<np.mean(X_train.loc[y_train.target

    label_train1 = y_train[(y_train.target==0) & (X_train[column]<np.mean(X_train.loc[y_train
    label_test1 = y_test[(y_test.target==0) & (X_test[column]<np.mean(X_train.loc[y_train.tar

    df_train2 = X_train[(y_train.target==1) & (X_train[column]<np.mean(X_train.loc[y_train.ta
    df_test2 = X_test[(y_test.target==1) & (X_test[column]<np.mean(X_train.loc[y_train.target

    label_train2 = y_train[(y_train.target==1) & (X_train[column]<np.mean(X_train.loc[y_train
    label_test2 = y_test[(y_test.target==1) & (X_test[column]<np.mean(X_train.loc[y_train.tar
```

```
X_train=pd.concat([df_train1,df_train2])
y_train=pd.concat([label_train1,label_train2])

X_test=pd.concat([df_test1,df_test2])
y_test=pd.concat([label_test1,label_test2])
```

```
from sklearn.feature_selection import VarianceThreshold
from sklearn.metrics import classification_report
eClassifier
from sklearn import tree
from sklearn import metrics
constant_filter = VarianceThreshold(threshold=0.0)
constant_filter.fit(X_train)
X_train = constant_filter.transform(X_train)
X_test = constant_filter.transform(X_test)

X_train.shape, X_test.shape
```



```
((212, 28), (91, 28))
```

```
columns=df.columns
columns_new=[]
for i in columns:
    columns_new.append(any(df[i].isnull()|df[i].isnull()))
df=df.drop(columns[columns_new],axis=1)
```

```
mm_scaler = preprocessing.StandardScaler()
X_train = pd.DataFrame(mm_scaler.fit_transform(X_train))
X_test=pd.DataFrame(mm_scaler.transform(X_test))
```

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(max_iter=50)
logreg.fit(X_train, y_train.values.ravel())
y_pred_logreg=logreg.predict(X_test)
acc = metrics.accuracy_score(y_pred_logreg,y_test.values.ravel())*100

print("Test Accuracy of Logistic Regression Algorithm: {:.2f}%".format(acc))
```

```
    Test Accuracy of Logistic Regression Algorithm: 84.62%
```

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, y_train.values.ravel())

y_pred_nb=nb.predict(X_test)
acc = metrics.accuracy_score(y_pred_nb,y_test.values.ravel())*100

print("Accuracy of Naive Bayes: {:.2f}%".format(acc))
```

```
    Accuracy of Naive Bayes: 80.22%
```

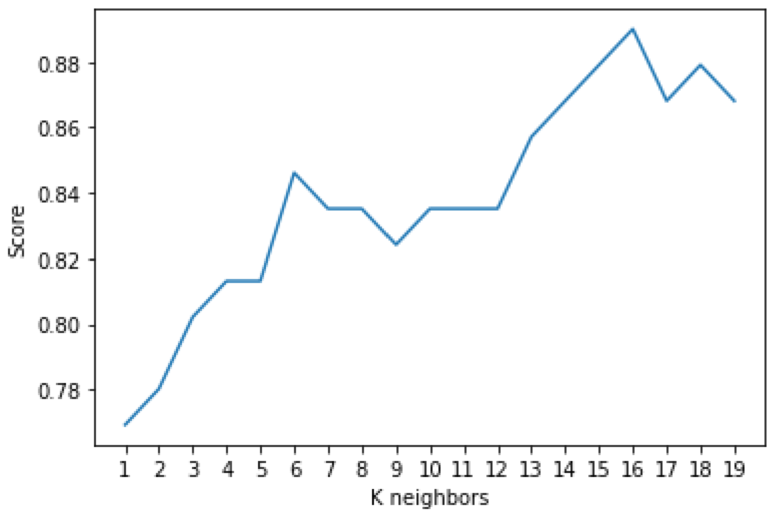
```
from sklearn.svm import SVC
from sklearn import svm

                              bility=True)
                              l())

y_pred_svm=svm.predict(X_test)
acc = metrics.accuracy_score(y_pred_svm,y_test.values.ravel())*100

print("Test Accuracy of SVM Algorithm: {:.2f}%".format(acc))
```

Saved successfully!

```
    Test Accuracy of SVM Algorithm: 76.92%
```

```
# KNN Model
from sklearn.neighbors import KNeighborsClassifier

# try ro find best k value
score = []

for i in range(1,20):
    knn = KNeighborsClassifier(n_neighbors = i)  # n_neighbors means k
    knn.fit(X_train, y_train.values.ravel())
```

```
    score.append(knn.score(X_test, y_test.values.ravel()))

plt.plot(range(1,20), score)
plt.xticks(np.arange(1,20,1))
plt.xlabel("K neighbors")
plt.ylabel("Score")
plt.show()

acc = max(score)*100
print("Maximum KNN Score is {:.2f}%".format(acc))
```




```
Maximum KNN Score is 89.01%
```

```
knn = KNeighborsClassifier(n_neighbors = 7)  # n_neighbors means k
knn.fit(X_train, y_train.values.ravel())
```

```
KNeighborsClassifier(n_neighbors=7)
```

Saved successfully!

```
acc = metrics.accuracy_score(y_pred_knn,y_test.values.ravel())*100
print("KNN Accuracy Score : {:.2f}%".format(acc))
```

```
KNN Accuracy Score : 83.52%
```

```
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(class_weight="balanced",n_estimators=200,random_state = 1)
rf.fit(X_train, y_train.values.ravel())
y_pred_rf=rf.predict(X_test)
acc = metrics.accuracy_score(y_pred_rf,y_test.values.ravel())*100
print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))
```

```
Random Forest Algorithm Accuracy Score : 85.71%
```

```
from keras.models import Sequential
from keras.layers import Dense
```

```
from keras.layers import Dropout

# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(8, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X_train, y_train, epochs=40, batch_size=32)
# evaluate the keras model
_, accuracy = model.evaluate(X_test, y_test)
```

```
Epoch 1/40
7/7 [==============================] - 1s 3ms/step - loss: 0.8126 - accuracy: 0.4387
Epoch 2/40
7/7 [==============================] - 0s 2ms/step - loss: 0.8098 - accuracy: 0.4481
Epoch 3/40
7/7 [==============================] - 0s 3ms/step - loss: 0.7532 - accuracy: 0.4953
Epoch 4/40
7/7 [==============================] - 0s 3ms/step - loss: 0.7479 - accuracy: 0.4670
Epoch 5/40
7/7 [==============================] - 0s 5ms/step - loss: 0.7442 - accuracy: 0.4906
Epoch 6/40
7/7 [==============================] - 0s 2ms/step - loss: 0.7226 - accuracy: 0.5519
Epoch 7/40
7/7 [==============================] - 0s 2ms/step - loss: 0.7028 - accuracy: 0.5377
Epoch 8/40
7/7 [==============================] - 0s 3ms/step - loss: 0.6868 - accuracy: 0.5660
Epoch 9/40
====] - 0s 3ms/step - loss: 0.6965 - accuracy: 0.5849
7/7 [
====] - 0s 3ms/step - loss: 0.6828 - accuracy: 0.5708
Epoch 11/40
7/7 [==============================] - 0s 2ms/step - loss: 0.6582 - accuracy: 0.6321
Epoch 12/40
7/7 [==============================] - 0s 2ms/step - loss: 0.6762 - accuracy: 0.6179
Epoch 13/40
7/7 [==============================] - 0s 4ms/step - loss: 0.6528 - accuracy: 0.6368
Epoch 14/40
7/7 [==============================] - 0s 2ms/step - loss: 0.6356 - accuracy: 0.6887
Epoch 15/40
7/7 [==============================] - 0s 2ms/step - loss: 0.6149 - accuracy: 0.6887
Epoch 16/40
7/7 [==============================] - 0s 2ms/step - loss: 0.6007 - accuracy: 0.7547
Epoch 17/40
7/7 [==============================] - 0s 2ms/step - loss: 0.6115 - accuracy: 0.7123
Epoch 18/40
7/7 [==============================] - 0s 3ms/step - loss: 0.5923 - accuracy: 0.6981
Epoch 19/40
7/7 [==============================] - 0s 3ms/step - loss: 0.5883 - accuracy: 0.7547
```

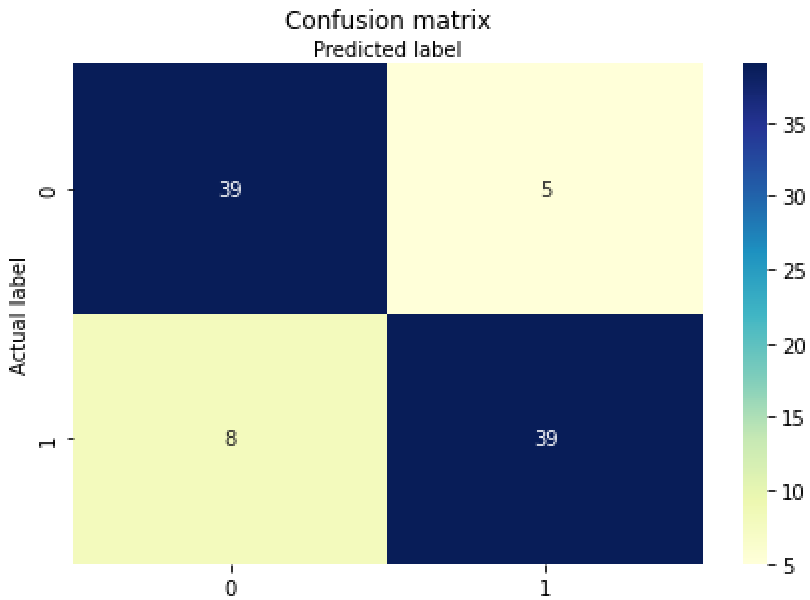
Saved successfully!

```
Epoch 20/40
7/7 [==============================] - 0s 2ms/step - loss: 0.5880 - accuracy: 0.7264
Epoch 21/40
7/7 [==============================] - 0s 2ms/step - loss: 0.5746 - accuracy: 0.7170
Epoch 22/40
7/7 [==============================] - 0s 2ms/step - loss: 0.5831 - accuracy: 0.7123
Epoch 23/40
7/7 [==============================] - 0s 2ms/step - loss: 0.5704 - accuracy: 0.7642
Epoch 24/40
7/7 [==============================] - 0s 2ms/step - loss: 0.5744 - accuracy: 0.7500
Epoch 25/40
7/7 [==============================] - 0s 2ms/step - loss: 0.5457 - accuracy: 0.7547
Epoch 26/40
7/7 [==============================] - 0s 3ms/step - loss: 0.5360 - accuracy: 0.7783
Epoch 27/40
7/7 [==============================] - 0s 2ms/step - loss: 0.5289 - accuracy: 0.7547
Epoch 28/40
7/7 [==============================] - 0s 3ms/step - loss: 0.5524 - accuracy: 0.7547
Epoch 29/40
```

```
#Evaluating models
def conf_matrix(matrix,pred):
    class_names= [0,1]# name  of classes
    fig, ax = plt.subplots()
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks, class_names)
    plt.yticks(tick_marks, class_names)
    # create heatmap
    sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    ax.xaxis.set_label_position("top")
    plt.tight_layout()
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
```

Saved successfully!

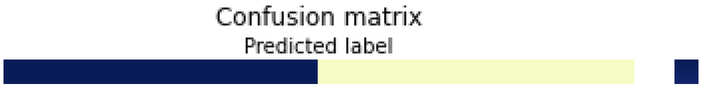
```
#Random Forest
# make class predictions with the model
y_pred = rf.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_pred,y_test)
conf_matrix(cnf_matrix,y_test)
# calculate prediction
report = classification_report(y_pred,y_test)
print(report)
```



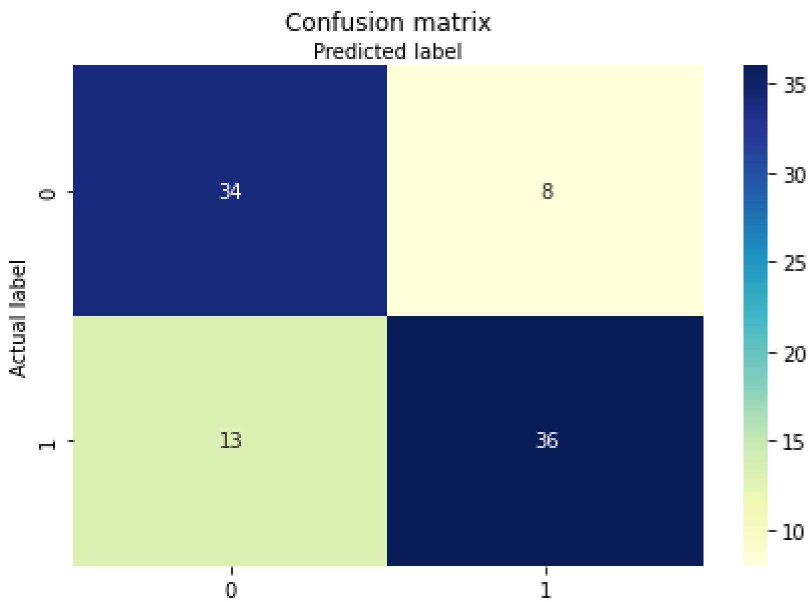

```
              precision    recall  f1-score   support
```

```
#Naive bayes
# make class predictions with the model
y_pred = nb.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_pred,y_test)
conf_matrix(cnf_matrix,y_test)
# calculate prediction
report = classification_report(y_pred,y_test)
print(report)
```

Saved successfully!



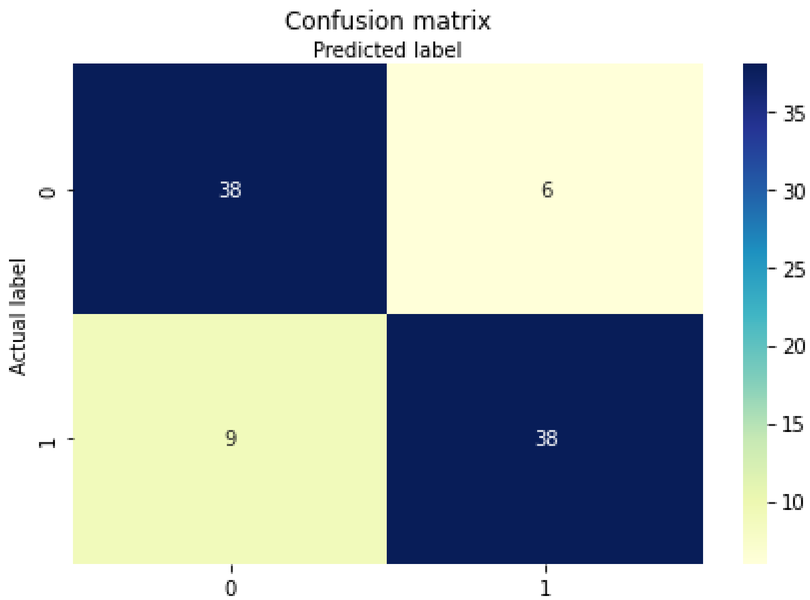

```
#svm
# make class predictions with the model
y_pred = svm.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_pred,y_test)
conf_matrix(cnf_matrix,y_test)
# calculate prediction
report = classification_report(y_pred,y_test)
print(report)
```




```
              precision    recall  f1-score   support

           0       0.72      0.81      0.76        42
                               73      0.77        49

                                       0.77        91
   macro avg       0.77      0.77      0.77        91
weighted avg       0.77      0.77      0.77        91
```



```
#K-Neighbours
# make class predictions with the model
y_pred = knn.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_pred,y_test)
conf_matrix(cnf_matrix,y_test)
# calculate prediction
report = classification_report(y_pred,y_test)
print(report)
```




```
              precision    recall  f1-score   support

           0       0.81      0.86      0.84        44
           1       0.86      0.81      0.84        47
```

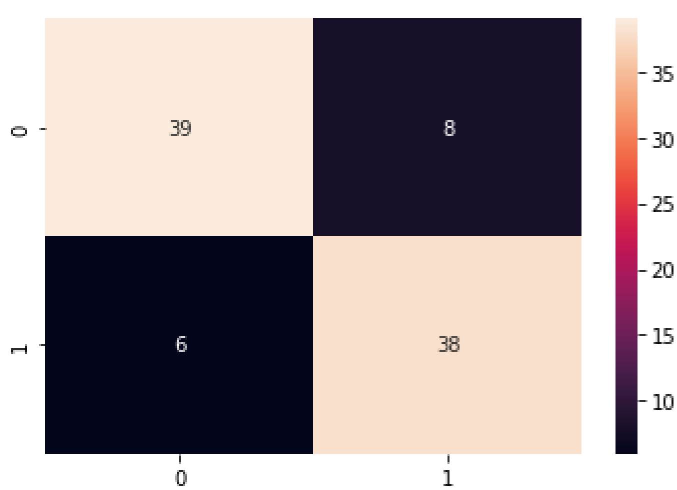
```
#Neural network
# make class predictions with the model
y_predict = np.argmax(model.predict(X_test), axis=-1)
cnf_matrix = metrics.confusion_matrix(y_pred,y_test)
conf_matrix(cnf_matrix,y_test)
report = classification_report(y_pred,y_test)
print(report)
```

Saved successfully! ×


```
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score,fbeta_scor
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score


from sklearn.metrics import f1_score
CM=confusion_matrix(y_test,y_pred_logreg)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
acc= accuracy_score(y_test, y_pred_logreg)
roc=roc_auc_score(y_test, y_pred_logreg)
prec = precision_score(y_test, y_pred_logreg)
rec = recall_score(y_test, y_pred_logreg)
f1 = f1_score(y_test, y_pred_logreg)


model_results =pd.DataFrame([['Logistic Regression',acc, prec,rec,specificity, f1,roc]],
               columns = ['Model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 S

model_results
```

Saved successfully!

			Precision	Sensitivity	Specificity	F1 Score	ROC
0	Logistic Regression	[illegible]	0.826087	0.863636	0.829787	0.844444	0.846712




```
from sklearn.metrics import log_loss,roc_auc_score,precision_score,f1_score,recall_score,roc_
data = {        'logreg': y_pred_logreg,
```

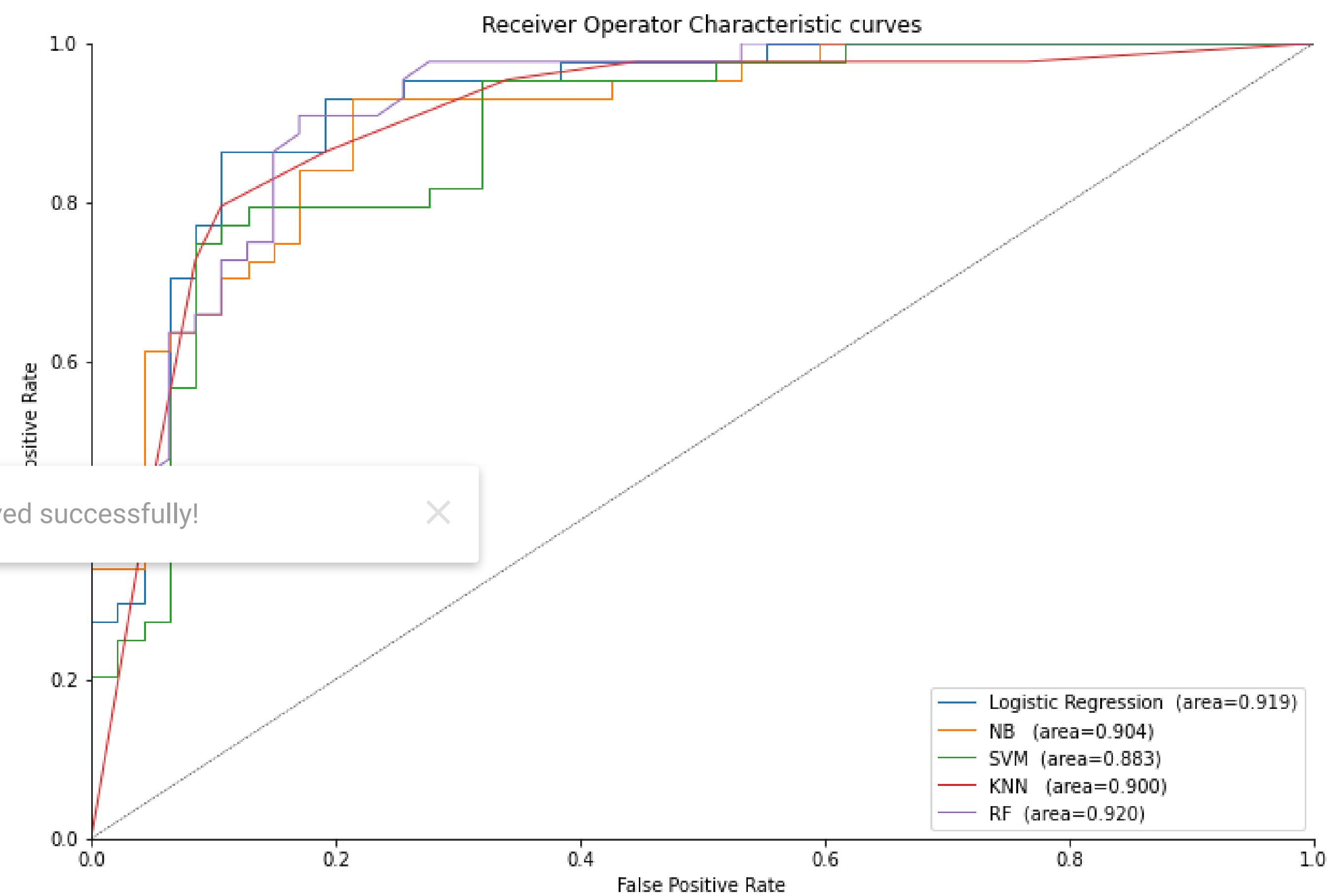
```
                'nb': y_pred_nb,
                'svm': y_pred_svm,
                'knn': y_pred_knn,
                'rf': y_pred_rf,

            }

models = pd.DataFrame(data)

for column in models:
    CM=confusion_matrix(y_test,models[column])

    TN = CM[0][0]
    FN = CM[1][0]
    TP = CM[1][1]
    FP = CM[0][1]
    specificity = TN/(TN+FP)

    acc= accuracy_score(y_test, models[column])
    roc=roc_auc_score(y_test, models[column])
    prec = precision_score(y_test, models[column])
    rec = recall_score(y_test, models[column])
    f1 = f1_score(y_test, models[column])


    results =pd.DataFrame([[column,acc, prec,rec,specificity, f1,roc]],
               columns = ['Model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 S
    model_results = model_results.append(results, ignore_index = True)

model_results
```

Saved successfully!

			Precision	Sensitivity	Specificity	F1 Score	ROC
			0.826087	0.863636	0.829787	0.844444	0.846712
1	logreg	0.846154	0.826087	0.863636	0.829787	0.844444	0.846712
2	nb	0.802198	0.809524	0.772727	0.829787	0.790698	0.801257
3	svm	0.769231	0.734694	0.818182	0.723404	0.774194	0.770793
4	knn	0.835165	0.808511	0.863636	0.808511	0.835165	0.836074
5	rf	0.857143	0.829787	0.886364	0.829787	0.857143	0.858075

```
def roc_auc_plot(y_true, y_proba, label=' ', l='-', lw=1.0):
    from sklearn.metrics import roc_curve, roc_auc_score
    fpr, tpr, _ = roc_curve(y_true, y_proba[:,1])
    ax.plot(fpr, tpr, linestyle=l, linewidth=lw,
            label="%s (area=%.3f)"%(label,roc_auc_score(y_true, y_proba[:,1])))
```

```
f, ax = plt.subplots(figsize=(12,8))


roc_auc_plot(y_test,logreg.predict_proba(X_test),label='Logistic Regression ',l='-')
roc_auc_plot(y_test,nb.predict_proba(X_test),label='NB  ',l='-')
roc_auc_plot(y_test,svm.predict_proba(X_test),label='SVM ',l='-')
roc_auc_plot(y_test,knn.predict_proba(X_test),label='KNN  ',l='-')
roc_auc_plot(y_test,rf.predict_proba(X_test),label='RF ',l='-')

ax.plot([0,1], [0,1], color='k', linewidth=0.5, linestyle='--',
        )
ax.legend(loc="lower right")
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.set_xlim([0, 1])
ax.set_ylim([0, 1])
ax.set_title('Receiver Operator Characteristic curves')
sns.despine()
```



Saved successfully!

```
# TODO:- MODEL, TRAINING and TESTING
import pandas as pd
data = pd.read_csv("/content/heart.csv")
print(data.corr())
data.corr()
```

```
               age       sex        cp  ...        ca      thal    target
age       1.000000 -0.098447 -0.068653  ...  0.276326  0.068001 -0.225439
sex      -0.098447  1.000000 -0.049353  ...  0.118261  0.210041 -0.280937
cp       -0.068653 -0.049353  1.000000  ... -0.181053 -0.161736  0.433798
trestbps  0.279351 -0.056769  0.047608  ...  0.101389  0.062210 -0.144931
chol      0.213678 -0.197912 -0.076904  ...  0.070511  0.098803 -0.085239
fbs       0.121308  0.045032  0.094444  ...  0.137979 -0.032019 -0.028046
restecg  -0.116211 -0.058196  0.044421  ... -0.072042 -0.011981  0.137230
thalach  -0.398522 -0.044020  0.295762  ... -0.213177 -0.096439  0.421741
exang     0.096801  0.141664 -0.394280  ...  0.115739  0.206754 -0.436757
oldpeak   0.210013  0.096093 -0.149230  ...  0.222682  0.210244 -0.430696
slope    -0.168814 -0.030711  0.119717  ... -0.080155 -0.104764  0.345877
ca        0.276326  0.118261 -0.181053  ...  1.000000  0.151832 -0.391724
thal      0.068001  0.210041 -0.161736  ...  0.151832  1.000000 -0.344029
target   -0.225439 -0.280937  0.433798  ... -0.391724 -0.344029  1.000000

[14 rows x 14 columns]
```

	age	sex	cp	trestbps	chol	fbs	restecg	thala
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.3985
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.0440
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.2957
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.0466
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.0099
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.0085
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.0441
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.0000
[illegible]	[illegible]	[illegible]	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.3788
[illegible]	[illegible]	[illegible]	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.3441
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.3867
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.2131
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.0964
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.4217

Saved successfully!

```
data=data.rename(columns={'age':'Age','sex':'Sex','cp':'Cp','trestbps':'Trestbps','chol':'Cho
```

```
#New show columns
print(data.columns)
```

```
Index(['Age', 'Sex', 'Cp', 'Trestbps', 'Chol', 'Fbs', 'Restecg', 'Thalach',
       'Exang', 'Oldpeak', 'Slope', 'Ca', 'Thal', 'Target'],
      dtype='object')
```

```
dataX=data.drop('Target',axis=1)
dataY=data['Target']
X_train,X_test,y_train,y_test=train_test_split(dataX,dataY,test_size=0.3,random_state=42)
print('X_train',X_train.shape)
print('X_test',X_test.shape)
print('y_train',y_train.shape)
print('y_test',y_test.shape)
```

```
X_train (212, 13)
X_test (91, 13)
y_train (212,)
y_test (91,)
```

```
#Normalization as the first process
# Normalize
X_train=(X_train-np.min(X_train))/(np.max(X_train)-np.min(X_train)).values
X_test=(X_test-np.min(X_test))/(np.max(X_test)-np.min(X_test)).values

from sklearn.decomposition import PCA
pca=PCA().fit(X_train)
print(pca.explained_variance_ratio_)
print()
print(X_train.columns.values.tolist())
print(pca.components_)
```

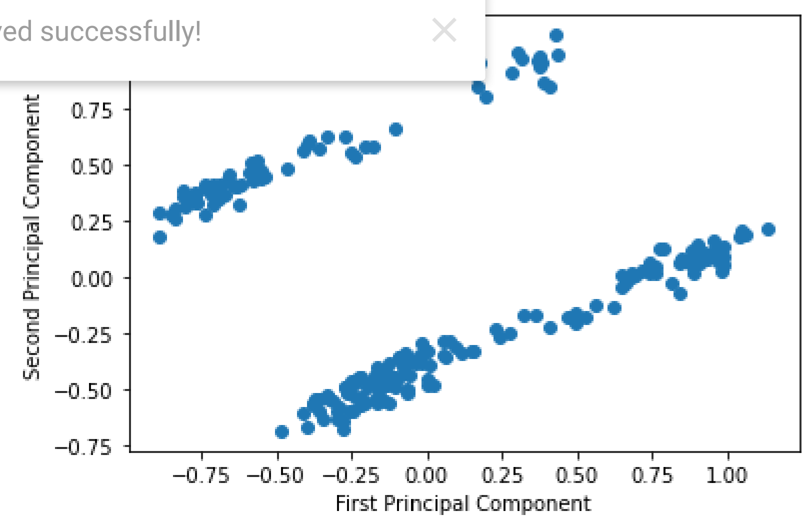


```
[0.27983251 0.19244371 0.1190256  0.08185051 0.07970383 0.06329404
 44 0.02481422 0.01802431 0.01653643

['Age', 'Sex', 'Cp', 'Trestbps', 'Chol', 'Fbs', 'Restecg', 'Thalach', 'Exang', 'Oldpeak
[[ 0.06748528  0.46514006 -0.33713815  0.00523708 -0.00675834 -0.02008128
  -0.0181853  -0.17291147  0.72233753  0.17007432 -0.23143612  0.11507873
   0.13122623]
 [ 0.08899783 -0.86322002 -0.24103172  0.05347111  0.06543148 -0.09444788
   0.05938857 -0.11409076  0.37485988  0.03054307 -0.11196353 -0.03895024
  -0.02690692]
 [ 0.14654406 -0.08815103  0.38377419  0.15650123  0.03611168  0.80618106
  -0.16386029 -0.06959297  0.09691224  0.10496249 -0.29659932  0.080185
  -0.03526663]
 [ 0.16278572 -0.00556925 -0.29501656 -0.01030338  0.02152029 -0.15195454
  -0.06535169 -0.1782624  -0.51825582  0.29299049 -0.65122965  0.21392761
   0.04929124]
 [-0.19141487  0.00820035  0.60018286 -0.08955885 -0.07324012 -0.35074511
   0.26120455  0.06601115  0.17659696  0.14948834 -0.47074568 -0.34463381
  -0.01319521]
 [-0.18145061  0.03180462 -0.29522609 -0.02976676 -0.06633548  0.37037866
   0.83843847  0.03970393 -0.10675794 -0.00683364 -0.04770591 -0.11953167
```

```
  0.06225589]
 [ 0.31042588 -0.02880005  0.36713044  0.09113953  0.02969298 -0.18959162
  0.40349644 -0.20138476  0.03627626  0.14717541  0.23055634  0.66083732
  0.08175907]
 [-0.13661882  0.03241732 -0.02414453 -0.3308474  -0.17679525  0.02178087
  0.03093033 -0.19029236  0.0632074  -0.20104409 -0.10414964  0.24216847
 -0.82911761]
 [ 0.5272751   0.14295977 -0.00819303  0.41238968  0.01682368 -0.08211305
  0.12443058 -0.41118173 -0.06874053 -0.0907869   0.07947623 -0.48748334
 -0.28793772]
 [ 0.07309209 -0.04401741  0.09203714 -0.62824629 -0.11726315  0.09247649
 -0.04833253 -0.60408946 -0.05520955 -0.15508501  0.08138407 -0.16070915
  0.37475708]
 [-0.25411673 -0.01547649 -0.01385981 -0.08318313  0.18258015  0.05907082
 -0.05099075 -0.21262928 -0.04363894  0.81460093  0.33386583 -0.15727544
 -0.20434411]
 [ 0.54908487  0.0415315  -0.02051409 -0.51363394  0.44670922  0.03592389
  0.08996002  0.4317689   0.04378352  0.08804434 -0.00757107 -0.11116942
 -0.12287669]
 [-0.32484899  0.05827329  0.04502966  0.10170275  0.83969328 -0.03101672
  0.03919764 -0.26857643 -0.01542372 -0.29187053 -0.09739615  0.06380959
 -0.00238118]]
```

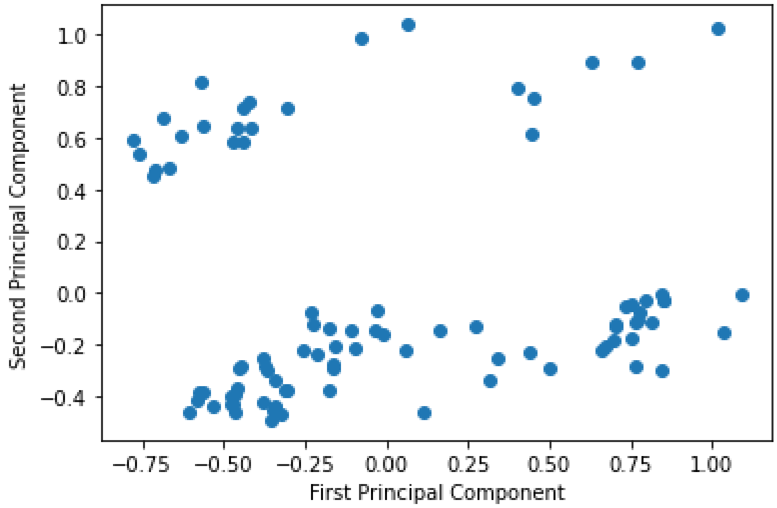
```
pca = PCA(n_components=8)
pca.fit(X_train)
reduced_data_train = pca.transform(X_train)
#inverse_data = pca.inverse_transform(reduced_data)
plt.scatter(reduced_data_train[:, 0], reduced_data_train[:, 1], label='reduced')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()
```

Saved successfully!




```
pca = PCA(n_components=8)
pca.fit(X_test)
reduced_data_test = pca.transform(X_test)
#inverse_data = pca.inverse_transform(reduced_data)
```

```
plt.scatter(reduced_data_test[:, 0], reduced_data_test[:, 1], label='reduced')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()
```




```
reduced_data_train = pd.DataFrame(reduced_data_train, columns=['Dim1', 'Dim2','Dim3','Dim4','
reduced_data_test = pd.DataFrame(reduced_data_test, columns=['Dim1', 'Dim2','Dim3','Dim4','Di
X_train=reduced_data_train
X_test=reduced_data_test
```

```
def plot_roc_(false_positive_rate,true_positive_rate,roc_auc):
    plt.figure(figsize=(5,5))
    plt.title('Receiver Operating Characteristic')
    plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.2f' % roc_
    plt.legend(loc = 'lower right')
```

Saved successfully!

```
e='--')
```

```
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

```
def plot_feature_importances(gbm):
    n_features = X_train.shape[1]
    plt.barh(range(n_features), gbm.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), X_train.columns)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)
combine_features_list=[
    ('Dim1','Dim2','Dim3'),
    ('Dim4','Dim5','Dim5','Dim6'),
    ('Dim7','Dim8','Dim1'),
    ('Dim4','Dim8','Dim5')
]
```

```
# TODO :------Logistic Regression

parameters=[
{
    'penalty':['l1','l2'],
    'C':[0.1,0.4,0.5],
    'random_state':[0]
    },
]

for features in combine_features_list:
    print(features)
    print("*"*50)

    X_train_set=X_train.loc[:,features]
    X_test_set=X_test.loc[:,features]

    gslog=GridSearchCV(LogisticRegression(),parameters,scoring='accuracy')
    gslog.fit(X_train_set,y_train)
    print('Best parameters set:')
    print(gslog.best_params_)
    print()
    predictions=[
    (gslog.predict(X_train_set),y_train,'Train'),
    (gslog.predict(X_test_set),y_test,'Test'),
    ]

    for pred in predictions:
        print(pred[2] + ' Classification Report:')
        print("*"*50)
                                          pred[1],pred[0]))

        print(pred[2] + ' Confusion Matrix:')
        print(confusion_matrix(pred[1], pred[0]))
        print("*"*50)

    print("*"*50)
    basari=cross_val_score(estimator=LogisticRegression(),X=X_train,y=y_train,cv=12)
    print(basari.mean())
    print(basari.std())
    print("*"*50)
```



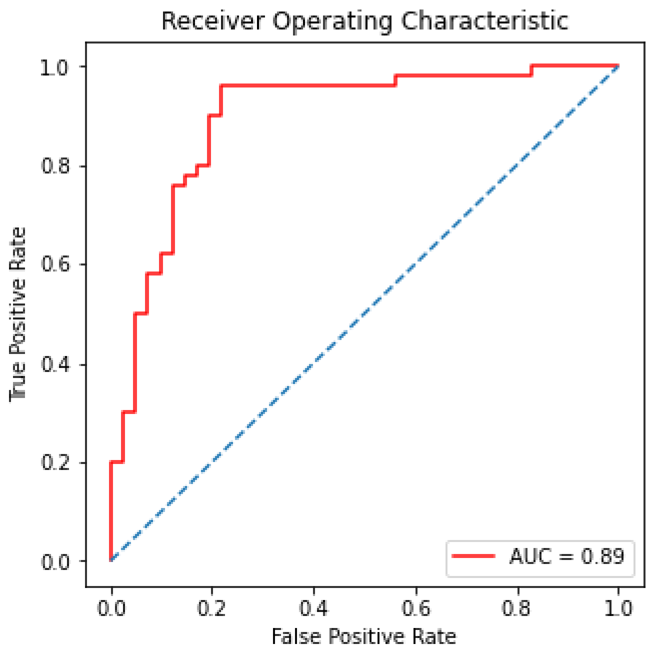
```
('Dim1', 'Dim2', 'Dim3')
**************************************************
Best parameters set:
{'C': 0.5, 'penalty': 'l2', 'random_state': 0}

Train Classification Report:
**************************************************
```

```
              precision    recall  f1-score   support

           0       0.83      0.64      0.72        97
           1       0.74      0.89      0.81       115

    accuracy                           0.77       212
   macro avg       0.79      0.76      0.77       212
weighted avg       0.78      0.77      0.77       212

**************************************************
Train Confusion Matrix:
[[ 62  35]
 [ 13 102]]
**************************************************
Test Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.74      0.61      0.67        41
           1       0.72      0.82      0.77        50

    accuracy                           0.73        91
   macro avg       0.73      0.71      0.72        91
weighted avg       0.73      0.73      0.72        91

**************************************************
Test Confusion Matrix:
[[25 16]
 [ 9 41]]
**************************************************
**************************************************
0.8259803921568629
0.06685549585135142
**************************************************
[illegible])
[illegible]******************

{'C': 0.4, 'penalty': 'l2', 'random_state': 0}

Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.66      0.47      0.55        97
           1       0.64      0.79      0.71       115

    accuracy                           0.65       212
   macro avg       0.65      0.63      0.63       212
weighted avg       0.65      0.65      0.64       212

**************************************************
```

Saved successfully!

```
from sklearn.linear_model import LogisticRegression

lr=LogisticRegression()
lr.fit(X_train,y_train)
```

```
y_pred=lr.predict(X_test)

y_proba=lr.predict_proba(X_test)

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,y_proba[:,1])
roc_auc = auc(false_positive_rate, true_positive_rate)
plot_roc_(false_positive_rate,true_positive_rate,roc_auc)
```



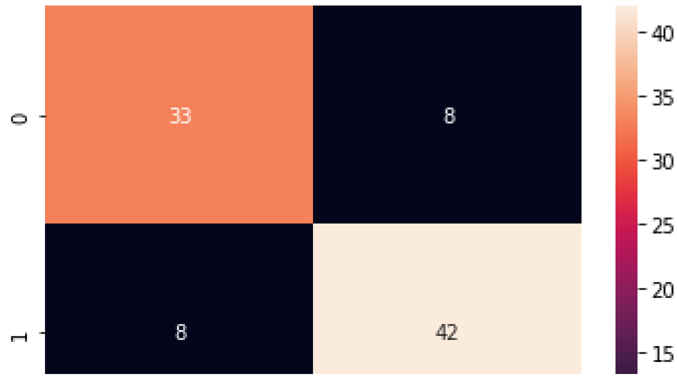

```
from sklearn.metrics import r2_score,accuracy_score

print('Error rate :',r2_score(y_test,y_pred))
ore(y_test, y_pred))
ormat(lr.score(X_train, y_train)))
print("Logistic TEST score with ",format(lr.score(X_test, y_test)))
print()
```

Saved successfully!

```
Error rate : 0.28975609756097553
Accurancy rate : 0.8241758241758241
Logistic TRAIN score with  0.8537735849056604
Logistic TEST score with  0.8241758241758241
```

```
cm=confusion_matrix(y_test,y_pred)
print(cm)
sns.heatmap(cm,annot=True)
plt.show()
```

```
[[33  8]
 [ 8 42]]
```




```
# TODO:- K-Nearest Neighbors


parameters=[
{
    'n_neighbors':np.arange(2,33),
    'n_jobs':[2,6]
    },
]
print("*"*50)
for features in combine_features_list:
    print("*"*50)

    X_train_set=X_train.loc[:,features]
    X_test_set=X_test.loc[:,features]

    gsknn=GridSearchCV(KNeighborsClassifier(),parameters,scoring='accuracy')
    gsknn.fit(X_train_set,y_train)
    print('Best parameters set:')
```

Saved successfully!

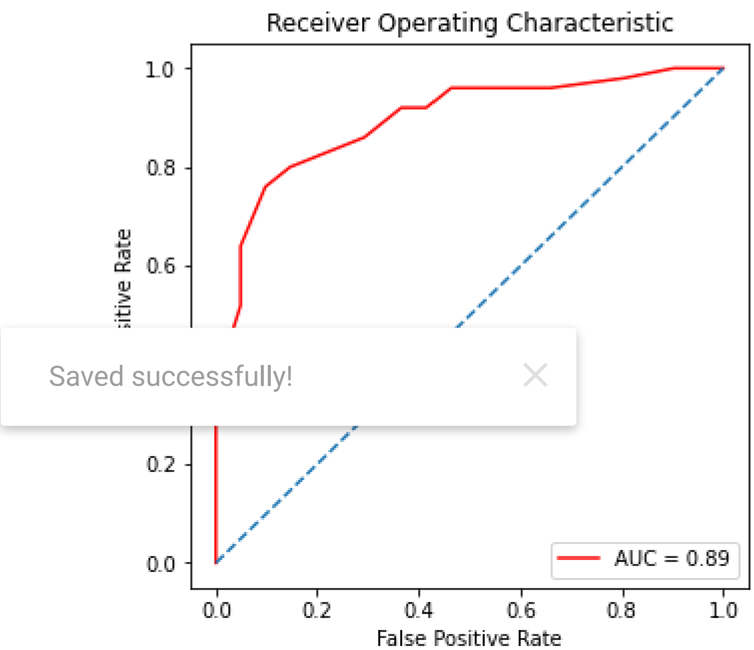
```
    predictions = [
    (gsknn.predict(X_train_set), y_train, 'Train'),
    (gsknn.predict(X_test_set), y_test, 'Test1')
    ]
    for pred in predictions:
        print(pred[2] + ' Classification Report:')
        print("*"*50)
        print(classification_report(pred[1], pred[0]))
        print("*"*50)
        print(pred[2] + ' Confusion Matrix:')
        print(confusion_matrix(pred[1], pred[0]))
        print("*"*50)

    print("*"*50)
    basari=cross_val_score(estimator=KNeighborsClassifier(),X=X_train,y=y_train,cv=12)
    print(basari.mean())
```

```
print(basari.std())
print("*"*50)
```

```
**************************************************
**************************************************
Best parameters set:
{'n_jobs': 2, 'n_neighbors': 14}
**************************************************
Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.75      0.87      0.80        97
           1       0.87      0.76      0.81       115

    accuracy                           0.81       212
   macro avg       0.81      0.81      0.81       212
weighted avg       0.82      0.81      0.81       212

**************************************************
Train Confusion Matrix:
[[84 13]
 [28 87]]
**************************************************
Test1 Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.74      0.68      0.71        41
           1       0.75      0.80      0.78        50

    accuracy                           0.75        91
   macro avg       0.75      0.74      0.74        91
weighted avg       0.75      0.75      0.75        91

Saved successfully!                          *******************

[[28 13]
 [10 40]]
**************************************************
**************************************************
0.7794117647058824
0.12371270306248845
**************************************************
**************************************************
Best parameters set:
{'n_jobs': 2, 'n_neighbors': 4}
**************************************************
Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.70      0.89      0.78        97
           1       0.88      0.68      0.76       115

    accuracy                           0.77       212
```

```
   macro avg       0.79      0.78      0.77       212
weighted avg       0.80      0.77      0.77       212

**************************************************
Train Confusion Matrix:
```

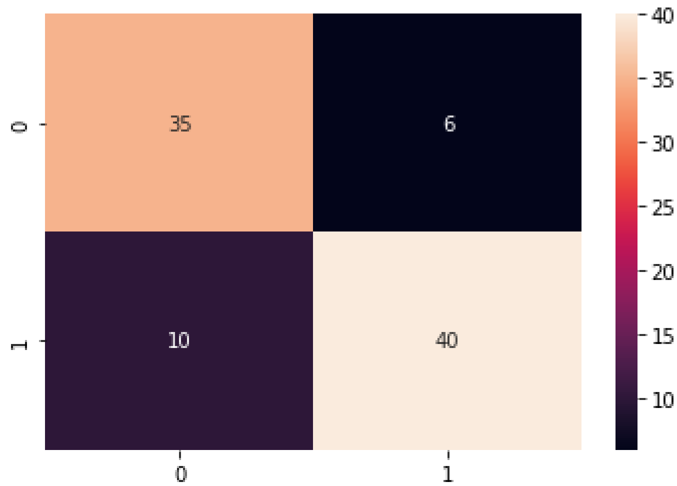
```
knn=KNeighborsClassifier(n_jobs=2, n_neighbors=22)
knn.fit(X_train,y_train)

y_pred=knn.predict(X_test)

y_proba=knn.predict_proba(X_test)
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,y_proba[:,1])
roc_auc = auc(false_positive_rate, true_positive_rate)
plot_roc_(false_positive_rate,true_positive_rate,roc_auc)

from sklearn.metrics import r2_score,accuracy_score

print('Accurancy rate :',accuracy_score(y_test, y_pred))
print("KNN TRAIN score with ",format(knn.score(X_train, y_train)))
print("KNN TEST score with ",format(knn.score(X_test, y_test)))
print()
```



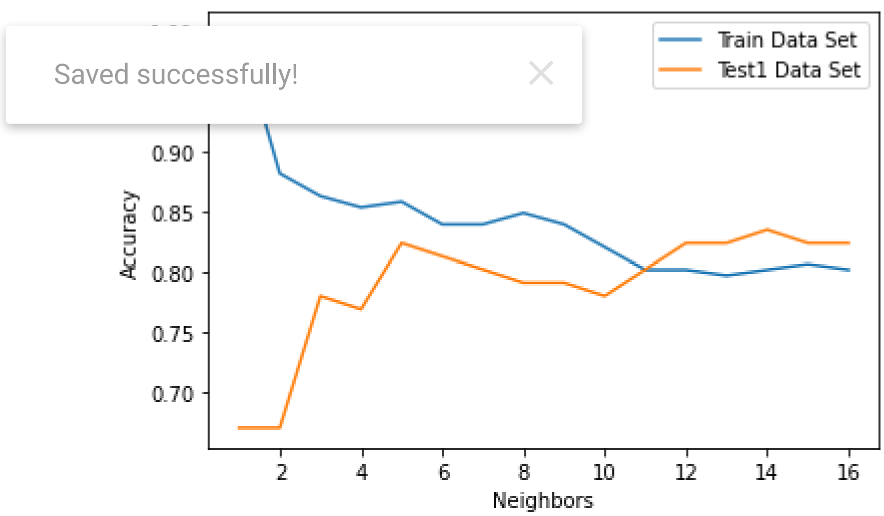

```
Accurancy rate : 0.8241758241758241
KNN TRAIN score with  0.8018867924528302
KNN TEST score with  0.8241758241758241
```

```
cm=confusion_matrix(y_test,y_pred)
print(cm)
sns.heatmap(cm,annot=True)
plt.show()
```

```
[[35  6]
 [10 40]]
```




```
n_neighbors = range(1, 17)
train_data_accuracy = []
test1_data_accuracy = []
for n_neigh in n_neighbors:
    knn = KNeighborsClassifier(n_neighbors=n_neigh,n_jobs=5)
    knn.fit(X_train, y_train)
    train_data_accuracy.append(knn.score(X_train, y_train))
    test1_data_accuracy.append(knn.score(X_test, y_test))
plt.plot(n_neighbors, train_data_accuracy, label="Train Data Set")
plt.plot(n_neighbors, test1_data_accuracy, label="Test1 Data Set")
plt.ylabel("Accuracy")
plt.xlabel("Neighbors")
plt.legend()
plt.show()
```

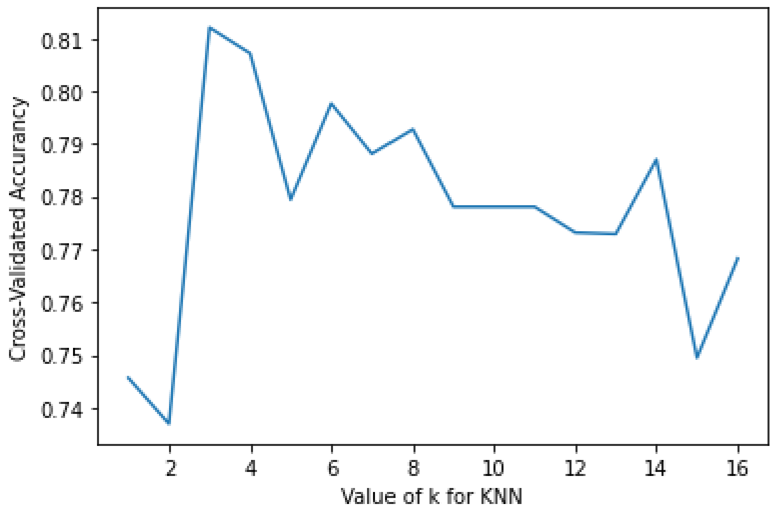



```
n_neighbors = range(1, 17)
k_scores=[]
for n_neigh in n_neighbors:
    knn = KNeighborsClassifier(n_neighbors=n_neigh,n_jobs=5)
    scores=cross_val_score(estimator=knn,X=X_train,y=y_train,cv=12)
```

```
    k_scores.append(scores.mean())
print(k_scores)
```

```
[0.7456427015250545, 0.7369281045751634, 0.812091503267974, 0.8071895424836603, 0.77941
```

```
plt.plot(n_neighbors,k_scores)
plt.xlabel('Value of k for KNN')
plt.ylabel("Cross-Validated Accurancy")
plt.show()
```




```
print('Leaf Size :',knn.leaf_size)
print('Metric :',knn.metric_params)
print('Radius :',knn.radius)
print('Weights :',knn.weights)
```

Saved successfully!

```
Metric : None
Radius : None
Weights : uniform
Algorithms : auto
```

```
# DOTO :0 Naive Baes
parameters = [
    {
        'kernel': ['linear'],
        'random_state': [2]
    },
    {
        'kernel': ['rbf'],
        'gamma':[0.9,0.06,0.3],
        'random_state': [0],
        'C':[1,2,3,4,5,6],
        'degree':[2],
```

```
        'probability':[True]
    },
]

for features in combine_features_list:
    print("*"*50)
    X_train_set=X_train.loc[:,features]
    X_test_set=X_test.loc[:,features]

    svc = GridSearchCV(SVC(), parameters,
    scoring='accuracy')
    svc.fit(X_train_set, y_train)
    print('Best parameters set:')
    print(svc.best_params_)
    print("*"*50)
    predictions = [
    (svc.predict(X_train_set), y_train, 'Train'),
    (svc.predict(X_test_set), y_test, 'Test1')
    ]
    for pred in predictions:
        print(pred[2] + ' Classification Report:')
        print("*"*50)
        print(classification_report(pred[1], pred[0]))
        print("*"*50)
        print(pred[2] + ' Confusion Matrix:')
        print(confusion_matrix(pred[1], pred[0]))
        print("*"*50)

    print("*"*50)
    basari=cross_val_score(estimator=SVC(),X=X_train,y=y_train,cv=4)
    print(basari.mean())
```

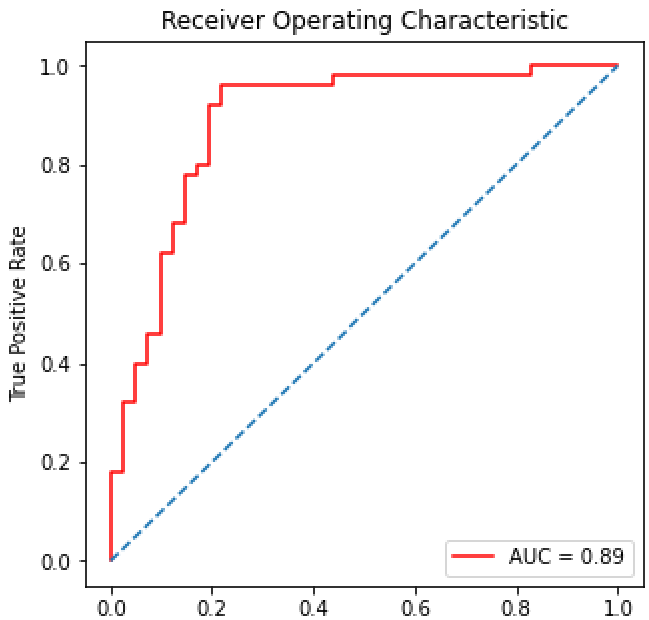
Saved successfully!

```
**************************************************
Best parameters set:
{'C': 6, 'degree': 2, 'gamma': 0.9, 'kernel': 'rbf', 'probability': True, 'random_sta
**************************************************
Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.77      0.73      0.75        97
           1       0.78      0.82      0.80       115

    accuracy                           0.78       212
   macro avg       0.78      0.77      0.78       212
weighted avg       0.78      0.78      0.78       212

**************************************************
Train Confusion Matrix:
[[71 26]
```

```
 [21 94]]
**************************************************
Test1 Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.71      0.61      0.66        41
           1       0.71      0.80      0.75        50

    accuracy                           0.71        91
   macro avg       0.71      0.70      0.71        91
weighted avg       0.71      0.71      0.71        91

**************************************************
Test1 Confusion Matrix:
[[25 16]
 [10 40]]
**************************************************
**************************************************
0.7877358490566038
0.05398831670877171
**************************************************
**************************************************
Best parameters set:
{'C': 2, 'degree': 2, 'gamma': 0.9, 'kernel': 'rbf', 'probability': True, 'random_sta
**************************************************
Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.64      0.51      0.57        97
           1       0.65      0.77      0.70       115

    accuracy                           0.65       212
                                64      0.63       212
                                65      0.64       212

**************************************************
Train Confusion Matrix:
```

Saved successfully!

```
from sklearn import svm

svc=svm.SVC(C=5,degree=2,gamma=0.06,kernel='rbf',probability=True,random_state=0)
svc.fit(X_train,y_train)

y_pred=svc.predict(X_test)

y_proba=svc.predict_proba(X_test)
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,y_proba[:,1])
roc_auc = auc(false_positive_rate, true_positive_rate)
plot_roc_(false_positive_rate,true_positive_rate,roc_auc)
```



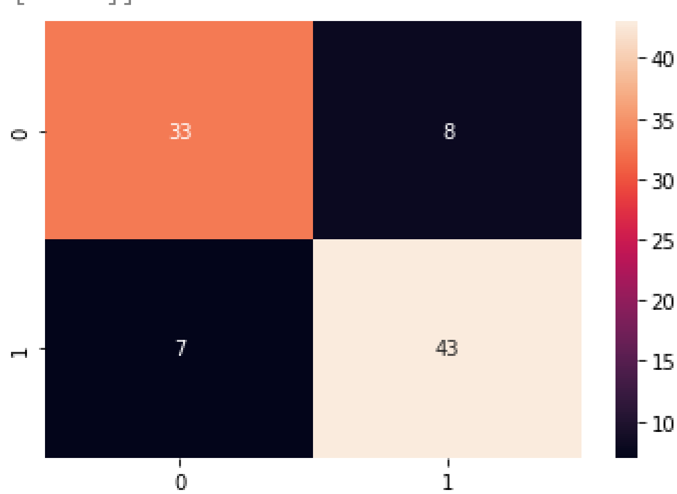

```
from sklearn.metrics import r2_score,accuracy_score

print('Accurancy rate :',accuracy_score(y_test, y_pred))
print("SVC TRAIN score with ",format(svc.score(X_train, y_train)))
print("SVC TEST score with ",format(svc.score(X_test, y_test)))
print()
```

```
Accurancy rate : 0.8351648351648352
SVC TRAIN score with  0.8301886792452831
SVC TEST score with  0.8351648351648352
```

```
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

Saved successfully!

```
[[33  8]
 [ 7 43]]
```



```
# Random Forest
parameters = [
    {
        'max_depth': np.arange(1, 10),
        'min_samples_split': np.arange(2, 5),
        'random_state': [3],
        'n_estimators': np.arange(10, 20)
    },
]

for features in combine_features_list:
    print("*"*50)

    X_train_set=X_train.loc[:,features]
    X_test1_set=X_test.loc[:,features]

    tree=GridSearchCV(RandomForestClassifier(),parameters,scoring='accuracy')
    tree.fit(X_train_set, y_train)

    print('Best parameters set:')
    print(tree.best_params_)
    print("*"*50)
    predictions = [
        (tree.predict(X_train_set), y_train, 'Train'),
        (tree.predict(X_test1_set), y_test, 'Test1')
    ]

    for pred in predictions:

        print(pred[2] + ' Classification Report:')
        print("*"*50)
                                  (pred[1], pred[0]))

                                  Matrix:')
        print(confusion_matrix(pred[1], pred[0]))
        print("*"*50)

    print("*"*50)
    basari=cross_val_score(estimator=RandomForestClassifier(),X=X_train,y=y_train,cv=4)
    print(basari.mean())
    print(basari.std())
    print("*"*50)
```

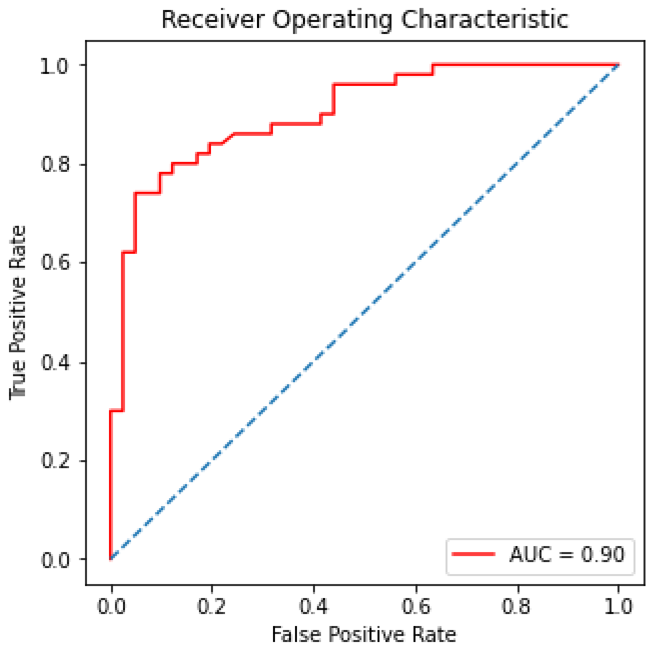
Saved successfully!

```
**************************************************
Best parameters set:
{'max_depth': 6, 'min_samples_split': 3, 'n_estimators': 18, 'random_state': 3}
**************************************************
Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        97
```

```
           1       0.93      0.93      0.93       115

    accuracy                           0.92       212
   macro avg       0.92      0.92      0.92       212
weighted avg       0.92      0.92      0.92       212

**************************************************
Train Confusion Matrix:
[[ 89   8]
 [  8 107]]
**************************************************
Test1 Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.74      0.68      0.71        41
           1       0.75      0.80      0.78        50

    accuracy                           0.75        91
   macro avg       0.75      0.74      0.74        91
weighted avg       0.75      0.75      0.75        91

**************************************************
Test1 Confusion Matrix:
[[28 13]
 [10 40]]
**************************************************
**************************************************
0.7830188679245284
0.021094980919809332
**************************************************
**************************************************
Best parameters set:
{'max_depth': 9, 'min_samples_split': 3, 'n_estimators': 13, 'random_state': 3}
**************************************************
Saved successfully!
                              ******************
              precision    recall  f1-score   support

           0       0.94      0.99      0.96        97
           1       0.99      0.95      0.97       115

    accuracy                           0.97       212
   macro avg       0.97      0.97      0.97       212
weighted avg       0.97      0.97      0.97       212

**************************************************
Train Confusion Matrix:
[[ 96   1]
```

```
rfc=RandomForestClassifier(max_depth=7,min_samples_split=4,n_estimators=19,random_state=3)
rfc.fit(X_train,y_train)

y_pred=rfc.predict(X_test)
```

```
y_proba=rfc.predict_proba(X_test)
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,y_proba[:,1])
roc_auc = auc(false_positive_rate, true_positive_rate)
plot_roc_(false_positive_rate,true_positive_rate,roc_auc)
```




```
from sklearn.metrics import r2_score,accuracy_score
print('Accurancy rate :',accuracy_score(y_test, y_pred))
print("RandomForestClassifier TRAIN score with ",format(rfc.score(X_train, y_train)))
print("RandomForestClassifier TEST score with ",format(rfc.score(X_test, y_test)))
print()
```

```
Accurancy rate : 0.8241758241758241
RandomForestClassifier TRAIN score with  0.9764150943396226
RandomForestClassifier TEST score with  0.8241758241758241
```

Saved successfully!

```
cm=confusion_matrix(y_test,y_pred)
print(cm)
sns.heatmap(cm,annot=True)
plt.show()
```

```
[[33  8]
 [ 8 42]]
```


```
for i in range(1,10):
    rf = RandomForestClassifier(n_estimators=i, random_state = 3, max_depth=7)
    rf.fit(X_train, y_train)
    print("TEST set score w/ " +str(i)+" estimators: {:.5}".format(rf.score(X_test, y_test)))
```

```
TEST set score w/ 1 estimators: 0.74725
TEST set score w/ 2 estimators: 0.7033
TEST set score w/ 3 estimators: 0.75824
TEST set score w/ 4 estimators: 0.78022
TEST set score w/ 5 estimators: 0.78022
TEST set score w/ 6 estimators: 0.83516
TEST set score w/ 7 estimators: 0.85714
TEST set score w/ 8 estimators: 0.83516
TEST set score w/ 9 estimators: 0.85714
```

```
# TODO:- SVM

parameters = [
{
    'random_state': [42],
    },
]
for features in combine_features_list:
    print("*"*50)
    X_train_set=X_train.loc[:,features]
    X_test1_set=X_test.loc[:,features]

                                          ters, scoring='accuracy')

    dtr.fit(X_train_set, y_train)
    print('Best parameters set:')
    print(dtr.best_params_)
    print("*"*50)
    predictions = [
    (dtr.predict(X_train_set), y_train, 'Train'),
    (dtr.predict(X_test1_set), y_test, 'Test1')
    ]
    for pred in predictions:
        print(pred[2] + ' Classification Report:')
        print("*"*50)
        print(classification_report(pred[1], pred[0]))
        print("*"*50)
        print(pred[2] + ' Confusion Matrix:')
        print(confusion_matrix(pred[1], pred[0]))
        print("*"*50)

    print("*"*50)
```

Saved successfully!

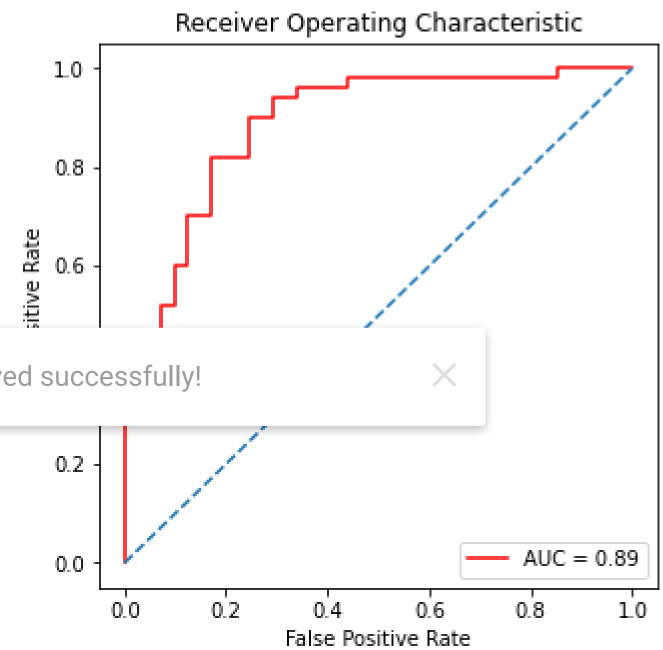
```
basari=cross_val_score(estimator=SVC(),X=X_train,y=y_train,cv=4)
print(basari.mean())
print(basari.std())
print("*"*50)
```

```
**************************************************
Best parameters set:
{'random_state': 42}
**************************************************
Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.78      0.69      0.73        97
           1       0.76      0.83      0.80       115

    accuracy                           0.77       212
   macro avg       0.77      0.76      0.76       212
weighted avg       0.77      0.77      0.77       212

**************************************************
Train Confusion Matrix:
[[67 30]
 [19 96]]
**************************************************
Test1 Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.74      0.61      0.67        41
           1       0.72      0.82      0.77        50

    accuracy                           0.73        91
                                 71      0.72        91
                                 73      0.72        91

**************************************************
Test1 Confusion Matrix:
[[25 16]
 [ 9 41]]
**************************************************
**************************************************
0.7877358490566038
0.05398831670877171
**************************************************
**************************************************
Best parameters set:
{'random_state': 42}
**************************************************
Train Classification Report:
**************************************************
              precision    recall  f1-score   support

           0       0.68      0.55      0.61        97
```

Saved successfully!

```
           1       0.67      0.78      0.72       115

    accuracy                           0.67       212
   macro avg       0.68      0.66      0.66       212
weighted avg       0.68      0.67      0.67       212

**************************************************
Train Confusion Matrix:
```

```
from sklearn import svm

SV=svm.SVC(random_state = 1,probability=True)
SV.fit(X_train,y_train)

y_pred=SV.predict(X_test)

y_proba=SV.predict_proba(X_test)

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,y_proba[:,1])
roc_auc = auc(false_positive_rate, true_positive_rate)
plot_roc_(false_positive_rate,true_positive_rate,roc_auc)
```



Saved successfully!

```
from sklearn.neural_network import MLPClassifier
MLP = MLPClassifier(hidden_layer_sizes=(32), learning_rate_init=0.001, max_iter=150)
model = MLP.fit(X_train, y_train)
MLP_predict = MLP.predict(X_test)
MLP_conf_matrix = confusion_matrix(y_test, MLP_predict)
MLP_acc_score = accuracy_score(y_test, MLP_predict)


#Printing the confussion matrix and accuracy scoresprint("confussion matrix")
```

```
print(MLP_conf_matrix)
print("\n")
print(classification_report(y_test,MLP_predict))
print("Accuracy of Multilayer Perceptron classifier: {:.3f}".format(MLP_acc_score*100),'%\n')
```

```
[[34  7]
 [ 8 42]]


              precision    recall  f1-score   support

           0       0.81      0.83      0.82        41
           1       0.86      0.84      0.85        50

    accuracy                           0.84        91
   macro avg       0.83      0.83      0.83        91
weighted avg       0.84      0.84      0.84        91

Accuracy of Multilayer Perceptron classifier: 83.516 %
```

Saved successfully! ×