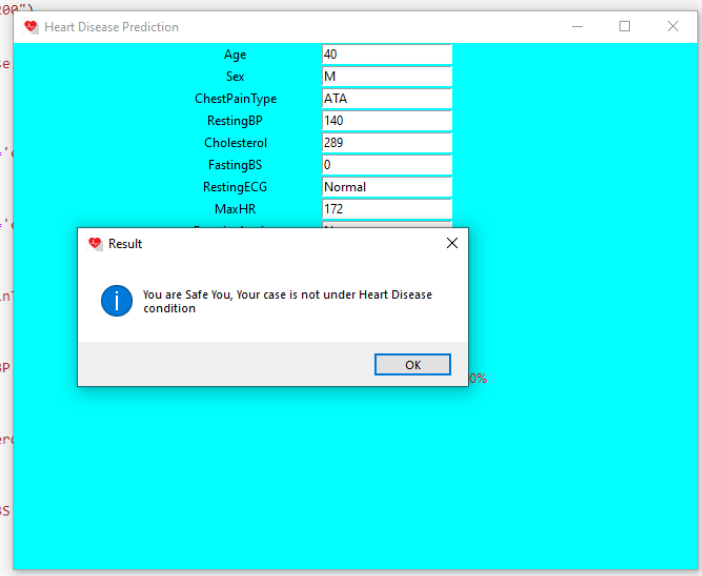
Working

In [*]:

```
import tkinter as tk
win=tk.Tk()
win.title('Heart Disease Prediction')

win.geometry("650x500+500+200")
win.attributes('-alpha',1)

win.iconbitmap('heartdisease
win.configure(bg='cyan')


tk.Label(win,text='
tk.Label(win,text='Age',bg='
e1=tk.Entry(win)
e1.grid(row=0,column=2)

tk.Label(win,text='Sex',bg='
e2=tk.Entry(win)
e2.grid(row=1,column=2)

tk.Label(win,text='ChestPain
e3=tk.Entry(win)
e3.grid(row=2,column=2)

tk.Label(win,text='RestingBP
e4=tk.Entry(win)
e4.grid(row=3,column=2)

tk.Label(win,text='Cholester
e5=tk.Entry(win)
e5.grid(row=4,column=2)

tk.Label(win,text='FastingBS
e6=tk.Entry(win)
e6.grid(row=5,column=2)

tk.Label(win,text='RestingECG',bg='cyan').grid(row=6,column=1)
e7=tk.Entry(win)
e7.grid(row=6,column=2)

tk.Label(win,text='MaxHR',bg='cyan').grid(row=7,column=1)
e8=tk.Entry(win)
e8.grid(row=7,column=2)

tk.Label(win,text='ExerciseAngina',bg='cyan').grid(row=8,column=1)
e9=tk.Entry(win)
e9.grid(row=8,column=2)

tk.Label(win,text='Oldpeak',bg='cyan').grid(row=9,column=1)
e10=tk.Entry(win)
e10.grid(row=9,column=2)
```



Heart Disease Prediction

TABLE OF CONTENTS

1. Import Necessary Library

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
from sklearn.preprocessing import OneHotEncoder,StandardScaler,LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
```

```
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests is a
n internal NumPy module and should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

2. Data

In [2]:
```
df=pd.read_csv('heart.csv')
df.head()
```

Out[2]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

```
Data Dictionary:
1 Age: Age of the patient [years]
2 Sex: Sex of the patient [M: Male, F: Female]
3 ChestPainType: [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
4 RestingBP: Resting blood pressure [mm Hg]
5 Cholesterol: Serum cholesterol [mm/dl]
6 FastingBS: Fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
7 RestingECG: Resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST
elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]

8 MaxHR: Maximum heart rate achieved [Numeric value between 60 and 202] 9 ExerciseAngina: Exercise-induced angina [Y: Yes, N:
No]

10 Oldpeak: ST [Numeric value measured in depression]

11 ST_Slope: The slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]

12 HeartDisease: Output class [1: heart disease, 0: Normal]
```

In [3]:
```
print('Shape of Data Frame: ',df.shape)
```

```
Shape of Data Frame:  (918, 12)
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

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

Out[5]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.553377
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.497414
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.000000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.000000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.000000

3.Preprocessing

In [6]:
```
df.isna().sum()
```

Out[6]:
```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

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

Out[7]:
```
0
```

4. Exploratory Data Analysis

In [8]:
```
df.describe()
```

Out[8]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.553377
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.497414
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.000000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.000000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.000000

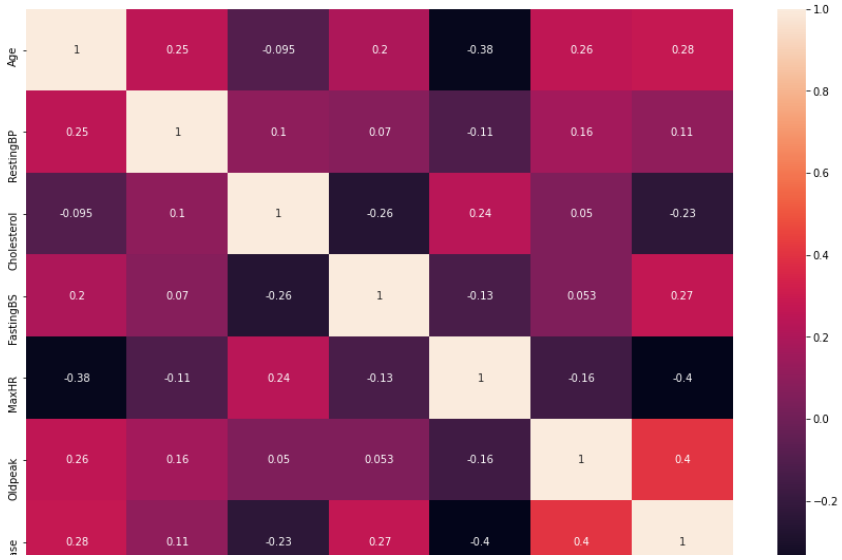
In [9]:
```
df.corr()
```

Out[9]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
Age	1.000000	0.254399	-0.095282	0.198039	-0.382045	0.258612	0.282039
RestingBP	0.254399	1.000000	0.100893	0.070193	-0.112135	0.164803	0.107589
Cholesterol	-0.095282	0.100893	1.000000	-0.260974	0.235792	0.050148	-0.232741
FastingBS	0.198039	0.070193	-0.260974	1.000000	-0.131438	0.052698	0.267291
MaxHR	-0.382045	-0.112135	0.235792	-0.131438	1.000000	-0.160691	-0.400421
Oldpeak	0.258612	0.164803	0.050148	0.052698	-0.160691	1.000000	0.403951
HeartDisease	0.282039	0.107589	-0.232741	0.267291	-0.400421	0.403951	1.000000

In [10]:
```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),annot=True)
```

Out[10]: <AxesSubplot:>

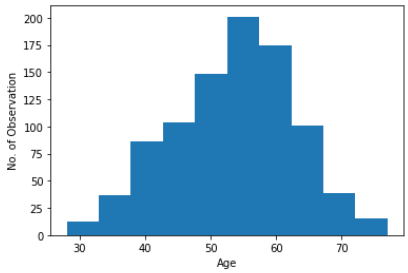


In [11]:
```
df.dtypes
```

Out[11]:
```
Age                int64
Sex               object
ChestPainType     object
RestingBP          int64
Cholesterol        int64
FastingBS          int64
RestingECG        object
MaxHR              int64
ExerciseAngina    object
Oldpeak          float64
ST_Slope          object
HeartDisease       int64
dtype: object
```

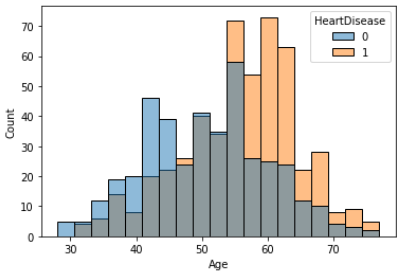
In [12]:
```
numerical_column=['Age','RestingBP','Cholesterol','FastingBS','MaxHR','Oldpeak']
categorical_column=['Sex','ChestPainType','RestingECG','ExerciseAngina','ST_Slope']
```

In [13]:
```
for i in numerical_column:
    plt.xlabel(i)
    plt.ylabel('No. of Observation')
    plt.hist(df[i])
    plt.show()
```



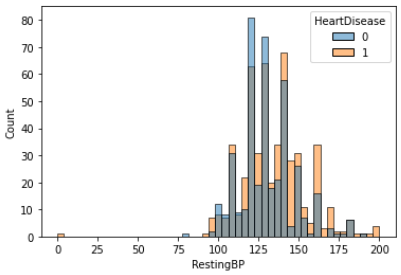
In [14]:
```
sns.histplot(data=df,x='Age',hue='HeartDisease')
```

Out[14]:
```
<AxesSubplot:xlabel='Age', ylabel='Count'>
```



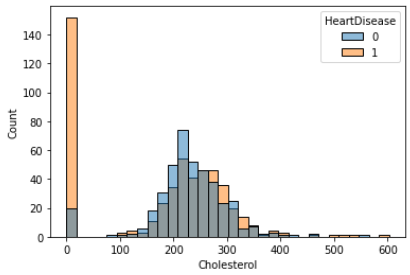
In [15]:
```
sns.histplot(data=df,x='RestingBP',hue='HeartDisease')
```

Out[15]:
```
<AxesSubplot:xlabel='RestingBP', ylabel='Count'>
```



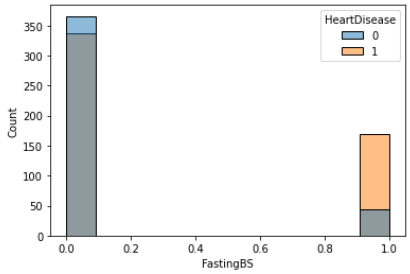
In [16]:
```
sns.histplot(data=df,x='Cholesterol',hue='HeartDisease')
```

Out[16]: <AxesSubplot:xlabel='Cholesterol', ylabel='Count'>



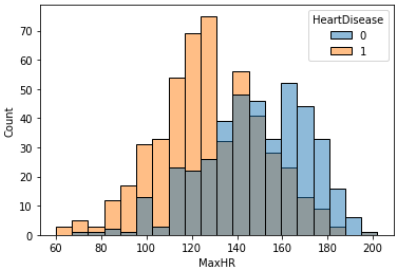
In [17]:
```
sns.histplot(data=df,x='FastingBS',hue='HeartDisease')
```

Out[17]: <AxesSubplot:xlabel='FastingBS', ylabel='Count'>



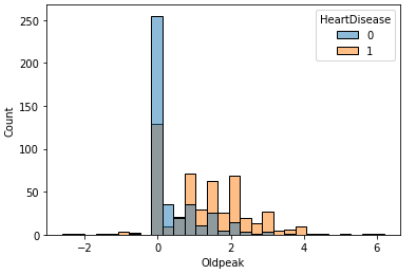
In [18]:
```
sns.histplot(data=df,x='MaxHR',hue='HeartDisease')
```

Out[18]: <AxesSubplot:xlabel='MaxHR', ylabel='Count'>



In [19]:
```
sns.histplot(data=df,x='Oldpeak',hue='HeartDisease')
```

Out[19]: <AxesSubplot:xlabel='Oldpeak', ylabel='Count'>



In [20]:
```
HaveHeartDisease=df[df['HeartDisease']==1].shape[0]
```

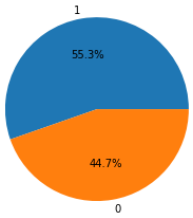
In [21]:
```
NotHaveHeartDisease=df[df['HeartDisease']==0].shape[0]
```

In [22]:
```
print("No. of Patient Having Heart Disease: ",HaveHeartDisease)
print("No. of Patient don't Have Heart Disease: ",NotHaveHeartDisease)
```

```
No. of Patient Having Heart Disease:  508
No. of Patient don't Have Heart Disease:  410
```

In [23]:
```
plt.pie([HaveHeartDisease,NotHaveHeartDisease],autopct='%1.1f%%',labels=['1','0'])
```

Out[23]:
```
([<matplotlib.patches.Wedge at 0x11c666a5b00>,
  <matplotlib.patches.Wedge at 0x11c666b3208>],
 [Text(-0.1835941114214546, 1.0845705151124876, '1'),
  Text(0.1835941114214547, -1.0845705151124876, '0')],
 [Text(-0.10014224259352068, 0.5915839173340841, '55.3%'),
  Text(0.10014224259352074, -0.5915839173340841, '44.7%')])
```

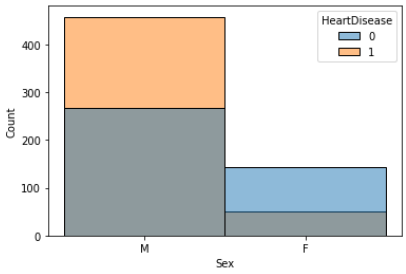


Working with Categorical Data

In [24]:
```
categorical_column=['Sex','ChestPainType','RestingECG','ExerciseAngina','ST_Slope']
```

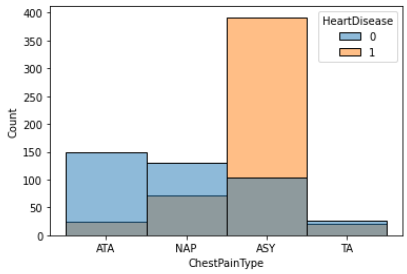
In [25]:
```
sns.histplot(data=df,x='Sex',hue='HeartDisease')
```

Out[25]:
```
<AxesSubplot:xlabel='Sex', ylabel='Count'>
```



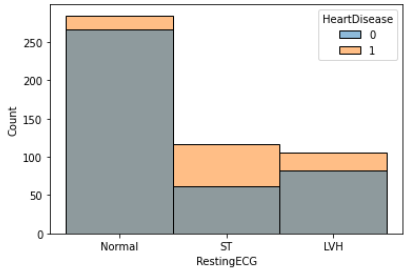
In [26]:
```
sns.histplot(data=df,x='ChestPainType',hue='HeartDisease')
```

Out[26]:
```
<AxesSubplot:xlabel='ChestPainType', ylabel='Count'>
```



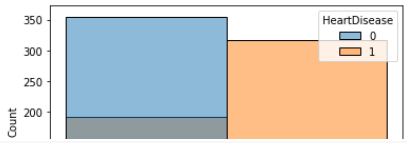
In [27]:
```
sns.histplot(data=df,x='RestingECG',hue='HeartDisease')
```

Out[27]:
```
<AxesSubplot:xlabel='RestingECG', ylabel='Count'>
```



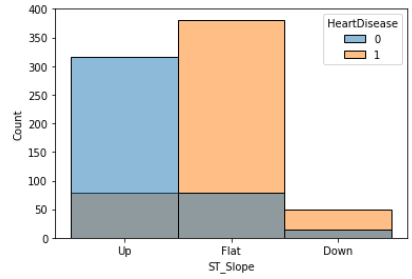
In [28]:
```
sns.histplot(data=df,x='ExerciseAngina',hue='HeartDisease')
```

Out[28]:
```
<AxesSubplot:xlabel='ExerciseAngina', ylabel='Count'>
```



In [29]:
```
sns.histplot(data=df,x='ST_Slope',hue='HeartDisease')
```

Out[29]:
```
<AxesSubplot:xlabel='ST_Slope', ylabel='Count'>
```



5. Feature Engineering

In [30]:
```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

6. Training of Model ¶

In [31]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3)
```

In [32]:
```
xtrain.head()
```

Out[32]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
789	34	M	TA	118	182	0	LVH	174	N	0.0	Up
621	56	M	NAP	130	256	1	LVH	142	Y	0.6	Flat
444	56	M	ASY	120	100	0	Normal	120	Y	1.5	Flat
340	43	M	ASY	100	0	1	Normal	122	N	1.5	Down
693	42	F	NAP	120	209	0	Normal	173	N	0.0	Flat

In [33]:
```
xtest.head()
```

Out[33]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
710	47	M	ASY	110	275	0	LVH	118	Y	1.0	Flat
426	56	M	ATA	126	166	0	ST	140	N	0.0	Up
198	53	M	ASY	120	246	0	Normal	116	Y	0.0	Flat
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat
241	54	M	ASY	200	198	0	Normal	142	Y	2.0	Flat

In [34]:
```
#xtrain=pd.get_dummies(xtrain,columns=categorical_column,drop_first=True)
#xtest=pd.get_dummies(xtest,columns=categorical_column,drop_first=True)
#LE=LabelEncoder()
#xtrain['Sex']=LE.fit_transform(xtrain['Sex'])
#xtest['Sex']=LE.transform(xtest['Sex'])
obj={}

#categorical_column=['Sex','ChestPainType','RestingECG','ExerciseAngina','ST_Slope']
for i in categorical_column:
    le=LabelEncoder()
    obj[i]=le
    xtrain[i]=obj[i].fit_transform(xtrain[i])
    xtest[i]=obj[i].transform(xtest[i])
```

```
C:\Users\hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
C:\Users\hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  del sys.path[0]
```

In [35]:
```
obj
```

Out[35]:
```
{'Sex': LabelEncoder(),
 'ChestPainType': LabelEncoder(),
 'RestingECG': LabelEncoder(),
 'ExerciseAngina': LabelEncoder(),
 'ST_Slope': LabelEncoder()}
```

In [36]:
```
print(xtrain.shape,xtest.shape)
```

```
(642, 11) (276, 11)
```

In [37]:
```
xtrain.head()
```

Out[37]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
789	34	1	3	118	182	0	0	174	0	0.0	2
621	56	1	2	130	256	1	0	142	1	0.6	1
444	56	1	0	120	100	0	1	120	1	1.5	1
340	43	1	0	100	0	1	1	122	0	1.5	0
693	42	0	2	120	209	0	1	173	0	0.0	1

In [38]:
```
xtest.head()
```

Out[38]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
710	47	1	0	110	275	0	0	118	1	1.0	1
426	56	1	1	126	166	0	2	140	0	0.0	2
198	53	1	0	120	246	0	1	116	1	0.0	1
913	45	1	3	110	264	0	1	132	0	1.2	1
241	54	1	0	200	198	0	1	142	1	2.0	1

In [39]:
```
sc=StandardScaler()
x_train_scaled=sc.fit_transform(xtrain)
x_test_scaled=sc.fit_transform(xtest)
```

In [40]:
```
x_train_scaled
```

Out[40]:
```
array([[-2.07480056,  0.52568236,  2.32217382, ..., -0.79859571,
        -0.82213815,  1.02178617],
       [ 0.27500965,  0.52568236,  1.27450299, ...,  1.25219807,
         -0.26212468, -0.64739123],
       [ 0.27500965,  0.52568236, -0.82083867, ...,  1.25219807,
          0.57789552, -0.64739123],
       ...,
       [ 0.06139054,  0.52568236,  0.22683216, ..., -0.79859571,
         -0.82213815,  1.02178617],
       [ 2.19758163,  0.52568236,  1.27450299, ..., -0.79859571,
         -0.635467  ,  1.02178617],
       [-0.15222857,  0.52568236, -0.82083867, ..., -0.79859571,
         -0.72880258,  1.02178617]])
```

In [41]:
```
xtrain=pd.DataFrame(x_train_scaled,columns=xtrain.columns[:])
xtest=pd.DataFrame(x_test_scaled,columns=xtest.columns[:])
xtrain.head()
```

Out[41]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
0	-2.074801	0.525682	2.322174	-0.795393	-0.196754	-0.542545	-1.611883	1.476244	-0.798596	-0.822138	1.021786
1	0.275010	0.525682	1.274503	-0.135222	0.480103	1.843166	-1.611883	0.195827	1.252198	-0.262125	-0.647391
2	0.275010	0.525682	-0.820839	-0.685365	-0.946786	-0.542545	-0.012457	-0.684460	1.252198	0.577896	-0.647391
3	-1.113515	0.525682	-0.820839	-1.785650	-1.861458	1.843166	-0.012457	-0.604434	-0.798596	0.577896	-2.316569
4	-1.220324	-1.902289	1.274503	-0.685365	0.050207	-0.542545	-0.012457	1.436231	-0.798596	-0.822138	-0.647391

In [42]:
```
m1=LogisticRegression()
m2=RidgeClassifier(alpha=0.25)
m3=DecisionTreeClassifier(criterion='gini',max_depth=5)
m4=RandomForestClassifier(criterion='gini',n_estimators=50)
m5=AdaBoostClassifier(algorithm='SAMME',learning_rate=0.1,n_estimators=150)
m6=GradientBoostingClassifier(criterion='friedman_mse',learning_rate=0.05,n_estimators=100)
m7=XGBClassifier(learning_rate=0.1,n_estimators=50)
m8=KNeighborsClassifier(n_neighbors=7)
```

In [43]:
```
dictionary={'Logistic Regression':m1,'Ridge':m2,'Decision Tree':m3,'Random Forest':m4,'Ada Booster':m5,'Gradient Boost':m6,'XGB b
```

In [44]:
```
best=0
bestName='Logistic Regression'
ScoreName={}
for i in dictionary.keys():
    model=dictionary[i]
    model.fit(xtrain,ytrain)
    pred=model.predict(xtest)
    score=model.score(xtest,ytest)
    print("Model Name: ",i)
    print("Accuracy: ",accuracy_score(ytest,pred))
    print()
    print(classification_report(ytest,pred))
    print('Confusion Matrix: \n',confusion_matrix(ytest,pred))
    print('################################################################')
    if(score>best):
        best=score
        bestName=i
    ScoreName[i]=score
```

```
Model Name:  Logistic Regression
Accuracy:  0.8115942028985508

              precision    recall  f1-score   support

           0       0.76      0.80      0.78       117
           1       0.85      0.82      0.83       159

avg / total       0.81      0.81      0.81       276

Confusion Matrix:
 [[ 94  23]
 [ 29 130]]
##################################################################
Model Name:  Ridge
Accuracy:  0.8043478260869565

              precision    recall  f1-score   support
```

In [45]:
```
print("Maximum Score: ",best)
print("Name: ",bestName)
```

```
Maximum Score:  0.8623188405797102
Name:  Gradient Boost
```

In [46]:
```
ScoreName
```

Out[46]:
```
{'Logistic Regression': 0.8115942028985508,
 'Ridge': 0.8043478260869565,
 'Decision Tree': 0.822463768115942,
 'Random Forest': 0.8333333333333334,
 'Ada Booster': 0.8333333333333334,
 'Gradient Boost': 0.8623188405797102,
 'XGB boost': 0.8478260869565217,
 'KNN': 0.8260869565217391}
```

Building Model with Best algorithm

In [47]:
```
model=dictionary[bestName]
model.fit(xtrain,ytrain)
```

Out[47]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.05, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=100,
```

In [48]:
```
def Prediction(feature):
    #categorical_column=['Sex','ChestPainType','RestingECG','ExerciseAngina','ST_Slope']
    # obj dictionary contins objects of all column of Label Encoder(Categorical type)
    for i in categorical_column:
        feature[i]=obj[i].transform(feature[i])  # LabelEncoder/ Changing Categorical data to numerical form

    feature=sc.transform(feature)    #Standard Scalling
    predctValue=model.predict(feature)
    print(predctValue)

    return predctValue
```

In [49]:
```
def CheckEntry():
    from tkinter import messagebox
    if(e1.get() == "" or e2.get() == "" or e3.get()=="" or e4.get()=="" or e5.get()=="" or e6.get()=="" or e7.get()=="" or e8.get

        messagebox.showinfo('Alert','Please provide all information')


    else:
        '''
            Age               int64
            Sex              object
            ChestPainType    object
            RestingBP         int64
            Cholesterol       int64
            FastingBS         int64
            RestingECG       object
            MaxHR             int64
            ExerciseAngina   object
            Oldpeak         float64
            ST_Slope         object
            HeartDisease      int64

        '''
        d={'Age':int(e1.get()),'Sex':e2.get(),'ChestPainType':e3.get(),'RestingBP':int(e4.get()),'Cholesterol':int(e5.get()),'Fas
        feature=pd.DataFrame(data=d,index=np.array([0]))

        predctValue=Prediction(feature)
        if predctValue==0:
            messagebox.showinfo('Result','You are Safe You, Your case is not under Heart Disease condition ')
        else:
            messagebox.showinfo('Result','You need treatmeant, Your case is under Heart Disease condition, Take care')
```

In [ ]:

In [*]:

```
import tkinter as tk
win=tk.Tk()
win.title('Heart Disease Prediction')

win.geometry("650x500+500+200")
win.attributes('-alpha',1)

win.iconbitmap('heartdisease.ico')
win.configure(bg='cyan')


tk.Label(win,text='                                                                    ',bg='cyan',fg='cyan').grid(column=0)
tk.Label(win,text='Age',bg='cyan').grid(row=0,column=1)
e1=tk.Entry(win)
e1.grid(row=0,column=2)

tk.Label(win,text='Sex',bg='cyan').grid(row=1,column=1)
e2=tk.Entry(win)
e2.grid(row=1,column=2)

tk.Label(win,text='ChestPainType',bg='cyan').grid(row=2,column=1)
e3=tk.Entry(win)
e3.grid(row=2,column=2)

tk.Label(win,text='RestingBP',bg='cyan').grid(row=3,column=1)
e4=tk.Entry(win)
e4.grid(row=3,column=2)

tk.Label(win,text='Cholesterol',bg='cyan').grid(row=4,column=1)
e5=tk.Entry(win)
e5.grid(row=4,column=2)

tk.Label(win,text='FastingBS',bg='cyan').grid(row=5,column=1)
e6=tk.Entry(win)
e6.grid(row=5,column=2)

tk.Label(win,text='RestingECG',bg='cyan').grid(row=6,column=1)
e7=tk.Entry(win)
e7.grid(row=6,column=2)

tk.Label(win,text='MaxHR',bg='cyan').grid(row=7,column=1)
e8=tk.Entry(win)
e8.grid(row=7,column=2)

tk.Label(win,text='ExerciseAngina',bg='cyan').grid(row=8,column=1)
e9=tk.Entry(win)
e9.grid(row=8,column=2)

tk.Label(win,text='Oldpeak',bg='cyan').grid(row=9,column=1)
e10=tk.Entry(win)
e10.grid(row=9,column=2)
```


```
e6.grid(row=5,column=2)

tk.Label(win,text='RestingECG',bg='cyan').grid(row=6,column=1)
e7=tk.Entry(win)
e7.grid(row=6,column=2)

tk.Label(win,text='MaxHR',bg='cyan').grid(row=7,column=1)
e8=tk.Entry(win)
e8.grid(row=7,column=2)

tk.Label(win,text='ExerciseAngina',bg='cyan').grid(row=8,column=1)
e9=tk.Entry(win)
e9.grid(row=8,column=2)

tk.Label(win,text='Oldpeak',bg='cyan').grid(row=9,column=1)
e10=tk.Entry(win)
e10.grid(row=9,column=2)
```


```
tk.Label(win,text='ST_Slope',bg='cyan').grid(row=10,column=1)
e11=tk.Entry(win)
e11.grid(row=10,column=2)

def HalloCall():
    print("Hello")


tk.Label(win,text='            ',bg='cyan').grid(row=11)

tk.Label(win,text='            ',bg='cyan').grid(row=12)

B=tk.Button(win,text="Check",command=CheckEntry,fg='red',activebackground='yellow',activeforeground='pink',bd='5')
B.grid(row=15,column=2)

tk.Label(win,text="We cannot ensure accuracy of 100%",fg='red',bd=5,bg='cyan').grid(row=17,column=2)

win.mainloop()
```

```
[0]
```