In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df=pd.read_csv('Downloads/Heart_Disease_Prediction.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST	Numb vesse flu
0	70	1	4	130	322	0	2	109	0	2.4	2	
1	67	0	3	115	564	0	2	160	0	1.6	2	
2	57	1	2	124	261	0	0	141	0	0.3	1	
3	64	1	4	128	263	0	0	105	1	0.2	2	
4	74	0	2	120	269	0	2	121	1	0.2	1	

In [4]:

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

Out[4]:

```
Age                        0
Sex                        0
Chest pain type            0
BP                         0
Cholesterol                0
FBS over 120               0
EKG results                0
Max HR                     0
Exercise angina            0
ST depression              0
Slope of ST                0
Number of vessels fluro    0
Thallium                   0
Heart Disease              0
dtype: int64
```

In [5]:

```
print(df.info())
```

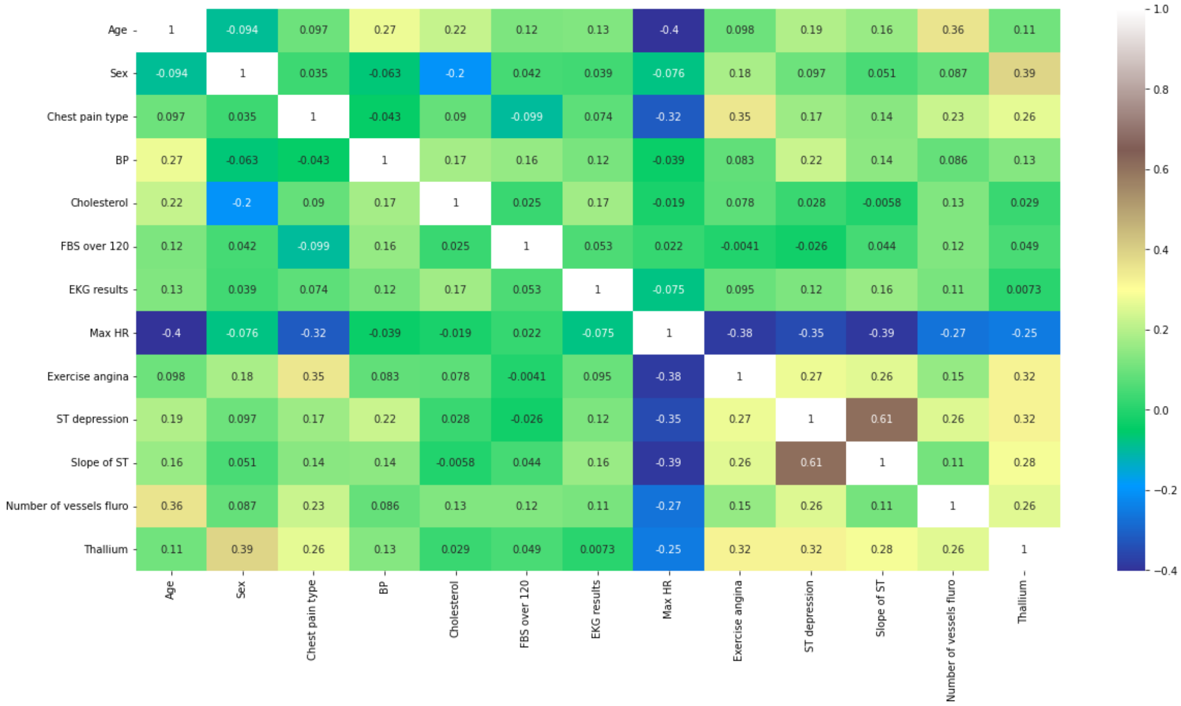
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Age                      270 non-null    int64  
 1   Sex                      270 non-null    int64  
 2   Chest pain type          270 non-null    int64  
 3   BP                       270 non-null    int64  
 4   Cholesterol              270 non-null    int64  
 5   FBS over 120             270 non-null    int64  
 6   EKG results              270 non-null    int64  
 7   Max HR                   270 non-null    int64  
 8   Exercise angina          270 non-null    int64  
 9   ST depression            270 non-null    float64
 10  Slope of ST              270 non-null    int64  
 11  Number of vessels fluro  270 non-null    int64  
 12  Thallium                 270 non-null    int64  
 13  Heart Disease            270 non-null    object 
dtypes: float64(1), int64(12), object(1)
memory usage: 29.7+ KB
None
```

In [6]:

```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), annot=True, cmap='terrain')
```

Out[6]:

```
<AxesSubplot:>
```

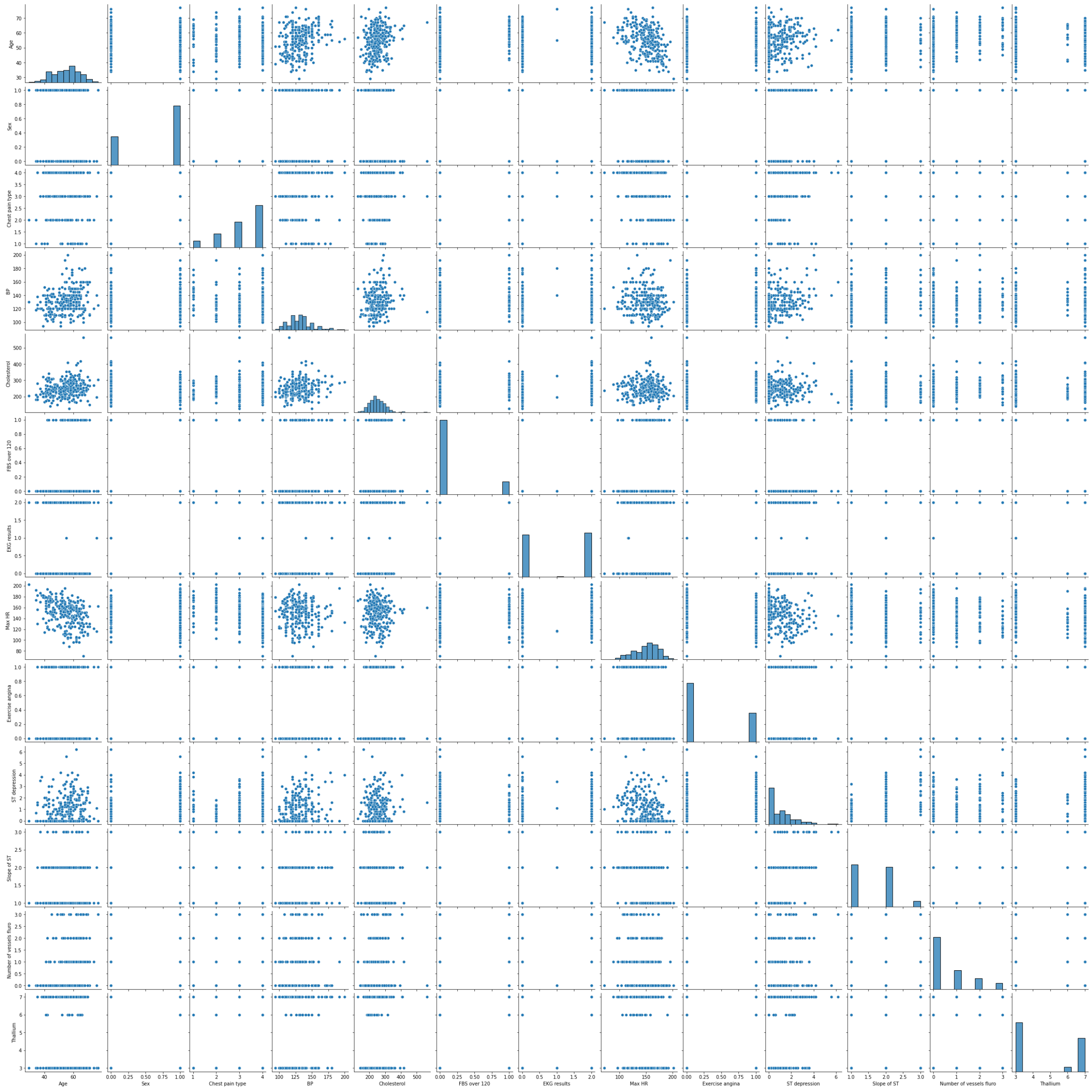


In [7]:

```
sns.pairplot(data=df)
```

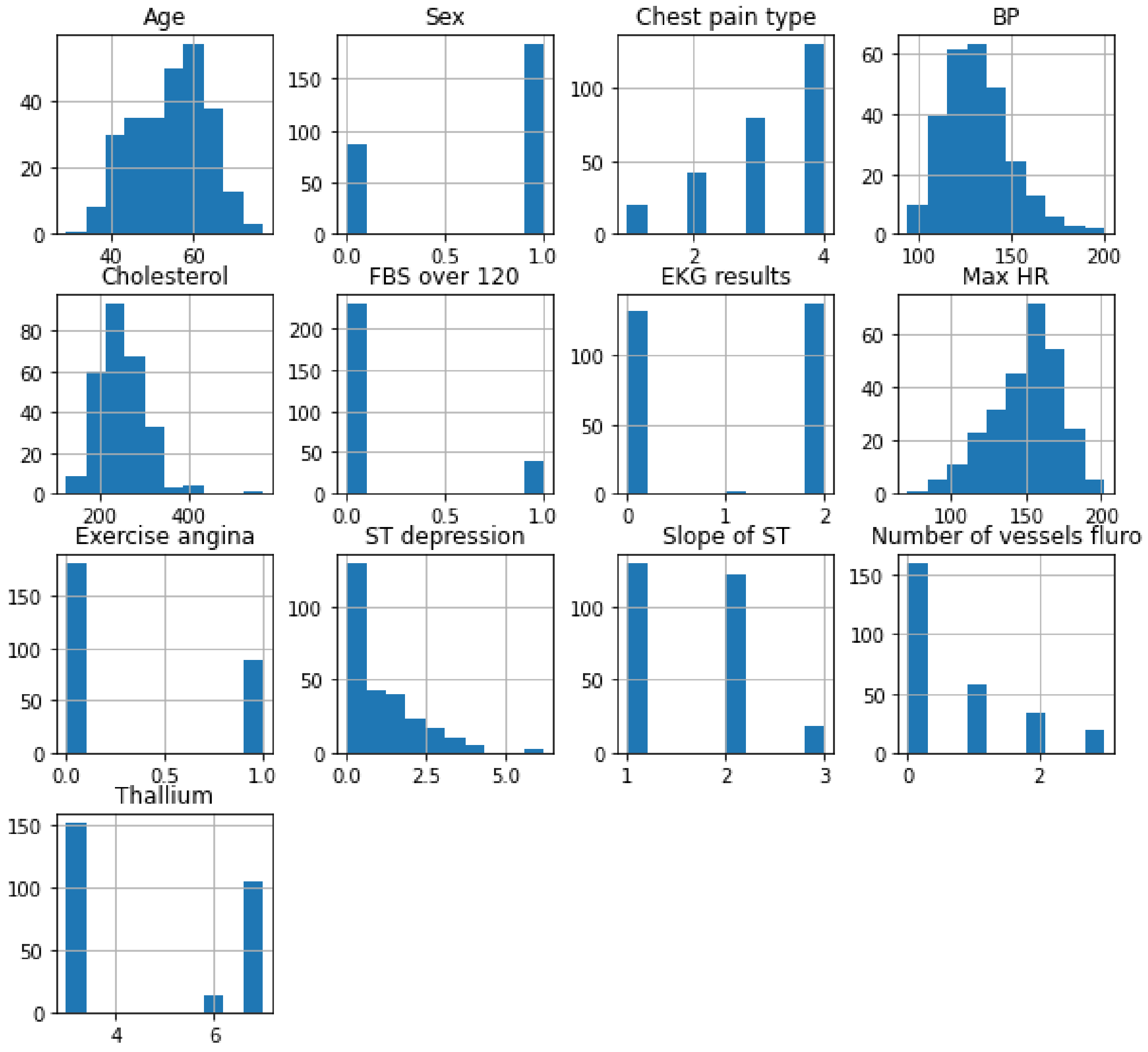
Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x1bb678b6308>
```



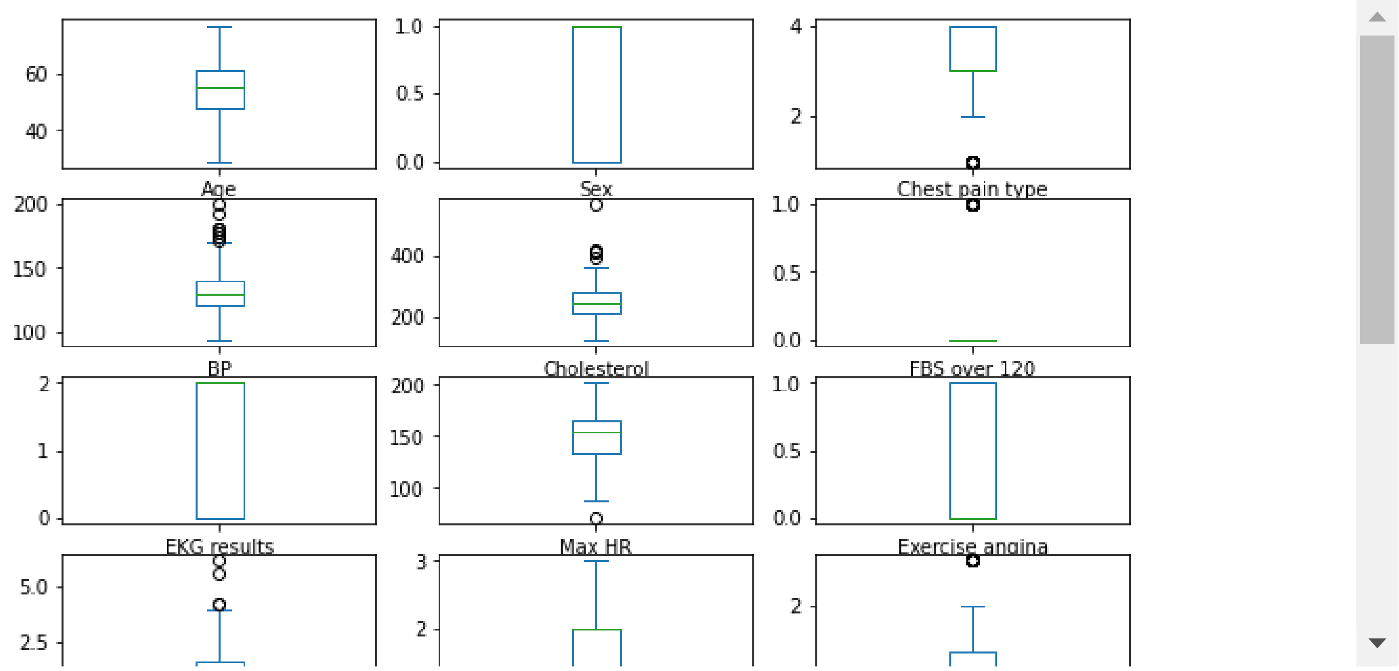
In [8]:

```
df.hist(figsize=(10,12), layout=(5,4));
```



In [9]:

```
df.plot(kind='box', subplots=True, layout=(6,3), figsize=(10,10))
plt.show()
```

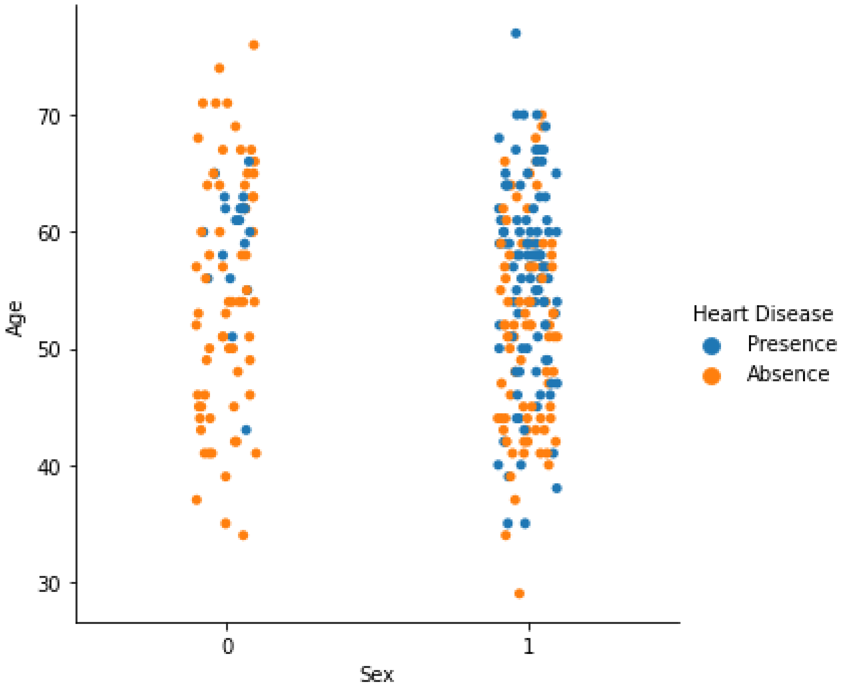


In [10]:

```
sns.catplot(data=df, x='Sex', y='Age', hue='Heart Disease', palette='tab10')
```

Out[10]:

```
<seaborn.axisgrid.FacetGrid at 0x1bb71a93fc8>
```

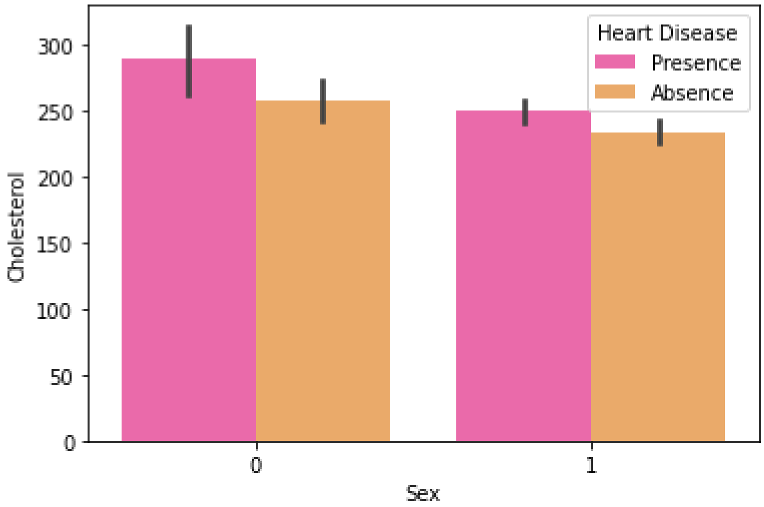


In [11]:

```
sns.barplot(data=df, x='Sex', y='Cholesterol', hue='Heart Disease', palette='spring')
```

Out[11]:

```
<AxesSubplot:xlabel='Sex', ylabel='Cholesterol'>
```



In [12]:

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

Out[12]:

```
1    183
0     87
Name: Sex, dtype: int64
```

In [13]:

```
df['Chest pain type'].value_counts()
```

Out[13]:

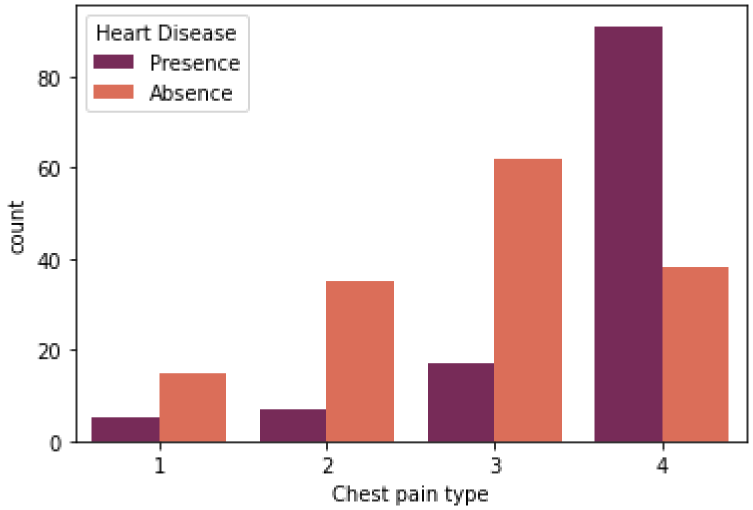
```
4    129
3     79
2     42
1     20
Name: Chest pain type, dtype: int64
```

In [14]:

```
sns.countplot(x='Chest pain type', hue='Heart Disease' , data=df, palette='rocket')
```

Out[14]:

```
<AxesSubplot:xlabel='Chest pain type', ylabel='count'>
```



In [15]:

```
gen = pd.crosstab(df['Sex'], df['Heart Disease'])
print(gen)
```

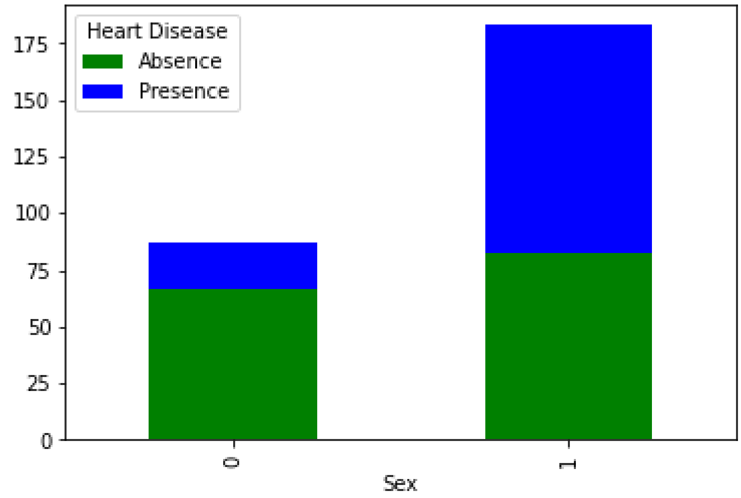
```
Heart Disease  Absence  Presence
Sex
0                   67        20
1                   83       100
```

In [16]:

```
gen.plot(kind='bar', stacked='True', color=['green','blue'],grid=False)
```

Out[16]:

```
<AxesSubplot:xlabel='Sex'>
```



In [17]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
StandardScaler = StandardScaler()
columns_to_scale=['Age', 'EKG results', 'Cholesterol', 'Thallium', 'Number of vessels fluro
df[columns_to_scale] = StandardScaler.fit_transform(df[columns_to_scale])
```

In [18]:

```
df.head()
```

Out[18]:

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST
0	1.712094	1	4	130	1.402212	0	0.981664	109	0	2.4	2
1	1.382140	0	3	115	6.093004	0	0.981664	160	0	1.6	2
2	0.282294	1	2	124	0.219823	0	-1.026285	141	0	0.3	1
3	1.052186	1	4	128	0.258589	0	-1.026285	105	1	0.2	2
4	2.152032	0	2	120	0.374890	0	0.981664	121	1	0.2	1

In [19]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
StandardScaler = StandardScaler()
columns_to_scale=['Age', 'EKG results', 'Cholesterol', 'Thallium', 'Number of vessels fluro
df[columns_to_scale] = StandardScaler.fit_transform(df[columns_to_scale])
```

In [20]:

```
df.head()
```

Out[20]:

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST
0	1.712094	1	4	130	1.402212	0	0.981664	109	0	2.4	2
1	1.382140	0	3	115	6.093004	0	0.981664	160	0	1.6	2
2	0.282294	1	2	124	0.219823	0	-1.026285	141	0	0.3	1
3	1.052186	1	4	128	0.258589	0	-1.026285	105	1	0.2	2
4	2.152032	0	2	120	0.374890	0	0.981664	121	1	0.2	1

In [21]:

```
x=df.drop(['Heart Disease'], axis=1)
y=df['Heart Disease']
```

In [22]:

```
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.3, random_state=40)
```

In [23]:

```
print('x_train-', x_train.size)
print('x_test-', x_test.size)
print('y_train-', y_train.size)
print('x_test-', x_test.size)
```

```
x_train- 2457
x_test- 1053
y_train- 189
x_test- 1053
```

In [24]:

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
model1=lr.fit(x_train,y_train)
prediction1=model1.predict(x_test)
```

In [25]:

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,prediction1)
cm
```

Out[25]:

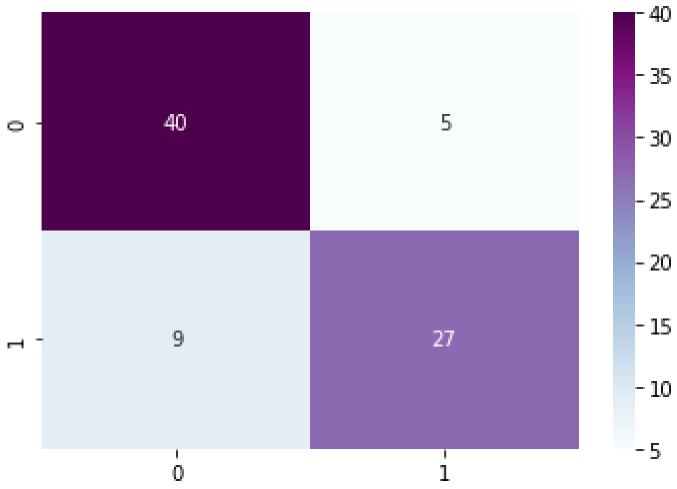
```
array([[40,  5],
       [ 9, 27]], dtype=int64)
```

In [26]:

```
sns.heatmap(cm, annot=True,cmap='BuPu')
```

Out[26]:

```
<AxesSubplot:>
```



In [27]:

```
TP=cm[0][0]
TN=cm[1][1]
FN=cm[1][0]
FP=cm[0][1]
print('Testing Accuracy:', (TP+TN+FN)/(TP+TN+FN+FP))
```

```
Testing Accuracy: 0.9382716049382716
```

In [28]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,prediction1)
l=accuracy_score(y_test,prediction1)
```

In [29]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, prediction1))
```

```
              precision    recall  f1-score   support

     Absence       0.82      0.89      0.85        45
    Presence       0.84      0.75      0.79        36

    accuracy                           0.83        81
   macro avg       0.83      0.82      0.82        81
weighted avg       0.83      0.83      0.83        81
```

In [30]:

```


import pandas as pd
from sklearn import neighbors,metrics
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import pickle
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import seaborn as sns
import matplotlib.pyplot as plt
```

In [31]:

```
from sklearn.metrics import accuracy_score
```

In [32]:

```
dataset = pd.read_csv("Downloads/Heart_Disease_Prediction.csv")
```

In [33]:

```
KX = dataset[['Age','Sex','Chest pain type','BP','Cholesterol','FBS over 120','EKG results'
```

In [34]:

```
KY = dataset[['Heart Disease']].values
```

In [35]:

```
KX
```

Out[35]:

```
array([[70.,  1.,  4., ...,  2.,  3.,  3.],
       [67.,  0.,  3., ...,  2.,  0.,  7.],
       [57.,  1.,  2., ...,  1.,  0.,  7.],
       ...,
       [56.,  0.,  2., ...,  2.,  0.,  3.],
       [57.,  1.,  4., ...,  2.,  0.,  6.],
       [67.,  1.,  4., ...,  2.,  3.,  3.]])
```

In [36]:

```
KY = KY.flatten()
print(KY)
```

```
['Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'
 'Presence' 'Presence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'
 'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence'
 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence'
 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'
 'Presence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence'
 'Presence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence'
 'Presence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence' 'Presence'
 'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'
 'Presence' 'Presence' 'Presence' 'Presence' 'Presence' 'Absence'
 'Presence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence'
 'Presence' 'Presence' 'Absence' 'Presence' 'Presence' 'Absence'
 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'
 'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Presence'
 'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence'
 'Presence' 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence'
 'Presence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence'
 'Presence' 'Presence' 'Presence' 'Presence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence'
 'Absence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence' 'Presence'
 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'
 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'
 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence'
 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence'
 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence'
 'Presence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence'
 'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence'
 'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'
 'Absence' 'Presence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence'
 'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence'
 'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Presence'
 'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Absence' 'Presence']
```

In [37]:

```
KX_train , KX_test , KY_train , KY_test = train_test_split(KX,KY,test_size=0.2,random_state
```

In [38]:

```
knn = KNeighborsClassifier(n_neighbors = 20)
knn.fit(KX_train, KY_train)
print(knn.score(KX_test, KY_test))
```

```
0.6111111111111112
```

In [39]:

```
pickle.dump(knn,open('heart_knn_model.sav','wb'))
```

In [40]:

```
predict_knn = knn.predict(KX_test)
accuracy_knn  = metrics.accuracy_score(KY_test,predict_knn)
```

In [41]:

```
predict_knn
```

Out[41]:

```
array(['Absence', 'Absence', 'Absence', 'Presence', 'Presence', 'Absence',
       'Presence', 'Absence', 'Absence', 'Absence', 'Presence', 'Absence',
       'Absence', 'Presence', 'Presence', 'Absence', 'Absence', 'Absence',
       'Presence', 'Presence', 'Presence', 'Presence', 'Absence',
       'Absence', 'Absence', 'Presence', 'Presence', 'Absence', 'Absence',
       'Presence', 'Absence', 'Presence', 'Presence', 'Absence',
       'Absence', 'Absence', 'Absence', 'Absence', 'Absence', 'Presence',
       'Absence', 'Presence', 'Absence', 'Absence', 'Absence', 'Absence',
       'Presence', 'Absence', 'Absence', 'Presence', 'Absence', 'Absence',
       'Absence', 'Absence'], dtype=object)
```

In [42]:

```
accuracy_knn
```

Out[42]:

```
0.6111111111111112
```

In [43]:

```
k=accuracy_knn
```

In [45]:

```
import csv
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix, f1_score, roc_curve, auc
import matplotlib.pyplot as plt
from itertools import cycle
from scipy import interp
```

In [46]:

```
df = pd.read_csv('Downloads/Heart_Disease_Prediction.csv', header = None)
```

In [47]:

```
training_x=df.iloc[1:df.shape[0],0:13]
```

In [48]:

```
training_y=df.iloc[1:df.shape[0],13:14]
```

In [49]:

```
nx=np.array(training_x)
ny=np.array(training_y)
```

In [52]:

```
for z in range(5):
    print("\nTest Train Split no. ",z+1,"\n")
    nx_train,nx_test,ny_train,ny_test = train_test_split(nx,ny,test_size=0.25,random_state=
    # Gaussian function of sklearn
    gnb = GaussianNB()
    gnb.fit(nx_train, ny_train.ravel())
    ny_pred = gnb.predict(nx_test)
```

```

Test Train Split no.  1


Test Train Split no.  2


Test Train Split no.  3


Test Train Split no.  4


Test Train Split no.  5

```

In [61]:

```
print("\n Naive Bayes model accuracy(in %):", metrics.accuracy_score(ny_test, ny_pred))
```

```

 Naive Bayes model accuracy(in %): 0.7794117647058824
```

In [62]:

```
n=metrics.accuracy_score(ny_test, ny_pred)
```

In [64]:

```
import pandas as pd
from sklearn import neighbors,metrics
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import pickle
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import seaborn as sns
import matplotlib.pyplot as plt
```

In [65]:

```
from sklearn.metrics import accuracy_score
```

In [67]:

```
dataset = pd.read_csv("Downloads/Heart_Disease_Prediction.csv")
```

In [69]:

```
DX = dataset[['Age','Sex','Chest pain type','BP','Cholesterol','FBS over 120','EKG results'
```

In [70]:

```
dy = dataset[['Heart Disease']].values
```

In [71]:

```
DX
```

Out[71]:

```
array([[70.,  1.,  4., ...,  2.,  3.,  3.],
       [67.,  0.,  3., ...,  2.,  0.,  7.],
       [57.,  1.,  2., ...,  1.,  0.,  7.],
       ...,
       [56.,  0.,  2., ...,  2.,  0.,  3.],
       [57.,  1.,  4., ...,  2.,  0.,  6.],
       [67.,  1.,  4., ...,  2.,  3.,  3.]])
```

In [72]:

```
dy = dy.flatten()
print(dy)
```

```
['Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'
 'Presence' 'Presence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'
 'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence'
 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence'
 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'
 'Presence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence'
 'Presence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence'
 'Presence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence' 'Presence'
 'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'
 'Presence' 'Presence' 'Presence' 'Presence' 'Presence' 'Absence'
 'Presence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence'
 'Presence' 'Presence' 'Absence' 'Presence' 'Presence' 'Absence'
 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'
 'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Presence'
 'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence'
 'Presence' 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence'
 'Presence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence'
 'Presence' 'Presence' 'Presence' 'Presence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence'
 'Absence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence' 'Presence'
 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'
 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'
 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence'
 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence'
 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence'
 'Presence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence'
 'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence'
 'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'
 'Absence' 'Presence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence'
 'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence'
 'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Presence'
 'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'
 'Absence' 'Absence' 'Absence' 'Presence']
```

In [73]:

```
DX_train , DX_test , dy_train , dy_test = train_test_split(DX,dy,test_size=0.2,random_state
```

In [74]:

```
from sklearn.tree import DecisionTreeClassifier

max_accuracy = 0
```

In [75]:

```
for x in range(200):
    dt = DecisionTreeClassifier(random_state=x)
    dt.fit(DX_train,dy_train)
    dy_pred_dt = dt.predict(DX_test)
    current_accuracy = round(accuracy_score(dy_pred_dt,dy_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x
```

In [85]:

```
dt = DecisionTreeClassifier(random_state=best_x)
dt.fit(DX_train,dy_train)
dy_pred_dt = dt.predict(DX_test)
```

In [88]:

```
score_dt = (accuracy_score(dy_pred_dt,dy_test))
```

In [89]:

```
print("The accuracy score achieved using Decision Tree is: "+str(score_dt))
```

```
The accuracy score achieved using Decision Tree is: 0.7962962962962963
```

In [90]:

```
d=(accuracy_score(dy_pred_dt,dy_test))
```

In [91]:

```
print('Logistic Regression :',l)
print('KNN :',k)
print('Naive Bayes :',n)
print('Decision Tree :' ,d)
```

```
Logistic Regression : 0.8271604938271605
KNN : 0.6111111111111112
Naive Bayes : 0.7794117647058824
Decision Tree : 0.7962962962962963
```

In [93]:

```
print('Logistic Regression :',l*100,'%')
print('KNN :',k*100,'%')
print('Naive Bayes :',n*100,'%')
print('Decision Tree :' ,d*100,'%')
```

```
Logistic Regression : 82.71604938271605 %
KNN : 61.111111111111114 %
Naive Bayes : 77.94117647058823 %
Decision Tree : 79.62962962962963 %
```

In [ ]: