```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df=pd.read_csv('/content/heart_disease.csv')
```

```
df.head()
```

	age	sex	chest	resting_blood_pressure	serum_cholestoral	fasting_blood_sugar	resting_electrocardiographic_results	maximum_heart_
0	70	1	4	130	322	0	2	
1	67	0	3	115	564	0	2	
2	57	1	2	124	261	0	0	
3	64	1	4	128	263	0	0	
4	74	0	2	120	269	0	2	

```
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                                 270 non-null    int64
 3   resting_blood_pressure                270 non-null    int64
 4   serum_cholestoral                     270 non-null    int64
 5   fasting_blood_sugar                   270 non-null    int64
 6   resting_electrocardiographic_results  270 non-null    int64
 7   maximum_heart_rate_achieved           270 non-null    int64
 8   exercise_induced_angina               270 non-null    int64
 9   oldpeak                               270 non-null    float64
 10  slope                                 270 non-null    int64
 11  number_of_major_vessels               270 non-null    int64
 12  thal                                  270 non-null    int64
 13  result                                270 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 29.7 KB
```

```
df.describe()
```

	age	sex	chest	resting_blood_pressure	serum_cholestoral	fasting_blood_sugar	resting_electrocardiographic_r
count	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.(
mean	54.433333	0.677778	3.174074	131.344444	249.659259	0.148148	1.(
std	9.109067	0.468195	0.950090	17.861608	51.686237	0.355906	0.!
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	0.(
25%	48.000000	0.000000	3.000000	120.000000	213.000000	0.000000	0.(
50%	55.000000	1.000000	3.000000	130.000000	245.000000	0.000000	2.(
75%	61.000000	1.000000	4.000000	140.000000	280.000000	0.000000	2.(
max	77.000000	1.000000	4.000000	200.000000	564.000000	1.000000	2.(

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()#creating
```

```
cool_names=df.columns
scaled_df=scaler.fit_transform(df)#applying scaling
scaled_df=pd.DataFrame(scaled_df,columns=cool_names)
scaled_df.head()
```

	age	sex	chest	resting_blood_pressure	serum_cholestoral	fasting_blood_sugar	resting_electrocardiographic_results	m
0	1.712094	0.689500	0.870928	-0.075410	1.402212	-0.417029	0.981664	
1	1.382140	-1.450327	-0.183559	-0.916759	6.093004	-0.417029	0.981664	
2	0.282294	0.689500	-1.238045	-0.411950	0.219823	-0.417029	-1.026285	
3	1.052186	0.689500	0.870928	-0.187590	0.258589	-0.417029	-1.026285	
4	2.152032	-1.450327	-1.238045	-0.636310	0.374890	-0.417029	0.981664	

```
#separating dependent and independent columns
x=scaled_df.drop(columns='result')
y=scaled_df['result']
y=y.astype('int')
```

```
#using naive bays algorithm

from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.model_selection import train_test_split
```

```
#Data splitting between train and test sets
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=4)
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((216, 13), (54, 13), (216,), (54,))
```

```
#creating object of NB algorithm
nb=GaussianNB()
```

```
#training model
nb.fit(x_train,y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
#prediction on y (target column)
y_pred=nb.predict(x_test) #prediction on test data
y_pred_train=nb.predict(x_train) #prediction on train data
```

```
y_pred
```

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

```
y_pred_train
```

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

```
#checking performance of algorithm on training data
print(classification_report(y_train,y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90       124
           1       0.87      0.86      0.86        92
```

```
    accuracy                           0.88       216
   macro avg       0.88      0.88      0.88       216
weighted avg       0.88      0.88      0.88       216
```

```
#checking performance of algorithm on training data
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.69      0.85      0.76        26
           1       0.82      0.64      0.72        28

    accuracy                           0.74        54
   macro avg       0.75      0.74      0.74        54
weighted avg       0.76      0.74      0.74        54
```

```
#training accuracy =88%
#testing accuracy=75%
#as traianing accuracy is greater than testing accuracy , by reasonable amount we can consider this algorithm is overfitting
```

```
training_accuracy=accuracy_score(y_train,y_pred_train)
testing_accuracy=accuracy_score(y_test,y_pred)
print(f'Training accuracy is {training_accuracy}')
print(f'Testing accuracy is {testing_accuracy}')
```

```
Training accuracy is 0.8842592592592593
Testing accuracy is 0.7407407407407407
```

```
#confusion matrix
confusion_matrix(y_test,y_pred)
```

```
array([[22,  4],
       [10, 18]])
```

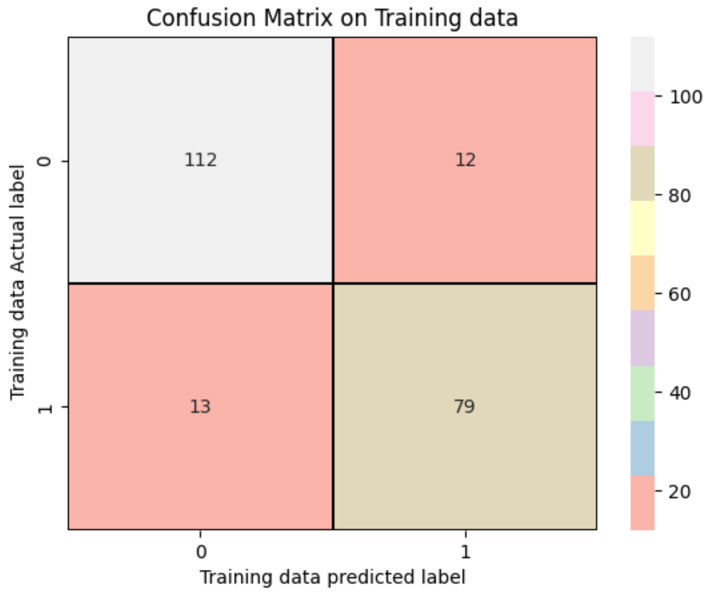
```
#confusion matrix
confusion_matrix(y_train,y_pred_train)
```

```
array([[112,  12],
       [ 13,  79]])
```

```
cm2=confusion_matrix(y_test,y_pred)
```

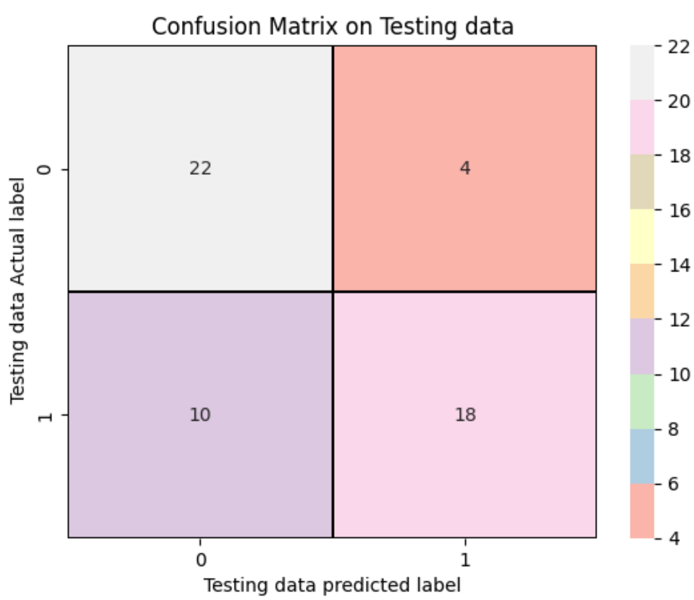
```
cm1=confusion_matrix(y_train,y_pred_train)
```

```
#plotting confusion matrix
sns.heatmap(cm1,annot=True,cmap='Pastel1',linewidths=0.3,linecolor='black',fmt='d')
plt.xlabel('Training data predicted label')
plt.ylabel(' Training data Actual label')
plt.title('Confusion Matrix on Training data')
plt.show()
```




```
#0 indicates no heart disease
#1 indicates heart
```

```
#plotting confusion matrix
sns.heatmap(cm2,annot=True,cmap='Pastel1',linewidths=0.3,linecolor='black',fmt='d')
plt.xlabel('Testing data predicted label')
plt.ylabel(' Testing data Actual label')
plt.title('Confusion Matrix on Testing data')
plt.show()
```




```
#false nagative = 1 0 = 10
#false positive = 0 1 =4
#true nagative = 0 0 =22
#true positive = 1 1 =18 ;;; first take y axis then x axis
```

```
#conclusion
#for confusion matrix 2
#false nagative = 1 0 = 10
#false positive = 0 1 =4
#true nagative = 0 0 =22
#true positive = 1 1 =18 ;

#for confusion matrix 2
#false nagative = 1 0 = 12 (TP)
#false positive = 0 1 =13  (FP)
```

```
#false positive = 0 1 =13  (FP)
#true nagative = 0 0 =112   (TN)
#true positive = 1 1 =79;   (TP)
```

Start coding or generate with AI