In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
data=pd.read_csv('E:/parkinsons.csv')
data.head(3)
```

Out[2]:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shi
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.

3 rows × 24 columns

In [3]:
```
data.describe()
```

Out[3]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	
mean	154.228641	197.104918	116.324631	0.006220	0.000044	0.003306	0.003446	0.009920	0.029709	
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	0.002759	0.008903	0.018857	
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680	0.000920	0.002040	0.009540	
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660	0.001860	0.004985	0.016505	
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500	0.002690	0.007490	0.022970	
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835	0.003955	0.011505	0.037885	
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440	0.019580	0.064330	0.119080	

8 rows × 23 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   name              195 non-null    object 
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      195 non-null    float64
 3   MDVP:Flo(Hz)      195 non-null    float64
 4   MDVP:Jitter(%)    195 non-null    float64
 5   MDVP:Jitter(Abs)  195 non-null    float64
 6   MDVP:RAP          195 non-null    float64
 7   MDVP:PPQ          195 non-null    float64
 8   Jitter:DDP        195 non-null    float64
 9   MDVP:Shimmer      195 non-null    float64
 10  MDVP:Shimmer(dB)  195 non-null    float64
 11  Shimmer:APQ3      195 non-null    float64
 12  Shimmer:APQ5      195 non-null    float64
 13  MDVP:APQ          195 non-null    float64
 14  Shimmer:DDA       195 non-null    float64
 15  NHR               195 non-null    float64
 16  HNR               195 non-null    float64
 17  status            195 non-null    int64  
 18  RPDE              195 non-null    float64
 19  DFA               195 non-null    float64
 20  spread1           195 non-null    float64
 21  spread2           195 non-null    float64
 22  D2                195 non-null    float64
 23  PPE               195 non-null    float64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

In [5]:
```
data.isna().sum()
```

Out[5]:
```
name              0
MDVP:Fo(Hz)       0
MDVP:Fhi(Hz)      0
MDVP:Flo(Hz)      0
```

```
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
status              0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
dtype: int64
```

In [6]:
```
data.shape
```

Out[6]:
```
(195, 24)
```

In [7]:
```
data.status.value_counts()
```

Out[7]:
```
1    147
0     48
Name: status, dtype: int64
```

In [8]:
```
data.drop('name',axis=1,inplace=True)
```

In [9]:
```
data.head(3)
```

Out[9]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shin
0	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	
1	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	
2	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	

3 rows × 23 columns

In [10]:
```
# grouping the data bas3ed on the target variable
data.groupby('status').mean()
```

Out[10]:

status	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP
0	181.937771	223.636750	145.207292	0.003866	0.000023	0.001925	0.002056	0.005776	0.017615	
1	145.180762	188.441463	106.893558	0.006989	0.000051	0.003757	0.003900	0.011273	0.033658	

2 rows × 22 columns

MODEL 1: SUPPORT VECTOR MACHINE

In [11]:
```
from sklearn.model_selection import train_test_split
```

In [12]:
```
X=data.drop('status',axis=1)
y=data.status
```

In [13]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

In [14]:
```
len(X_train)
```

Out[14]:
```
156
```

In [15]:
```
from sklearn.preprocessing import StandardScaler
from sklearn import svm
```

Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance.

In [16]:
```
scaler = StandardScaler()
```

In [17]:
```
scaler.fit(X_train)
```

Out[17]:
```
StandardScaler()
```

In [18]:
```
scaler.fit(X_test)
```

Out[18]:
```
StandardScaler()
```

In [19]:
```
X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)
```

In [20]:
```
svm_model=svm.SVC(kernel='linear',C=10) #Incresing Regularization term to increase score
```

In [21]:
```
svm_model.fit(X_train,y_train)
```

Out[21]:
```
SVC(C=10, kernel='linear')
```

In [22]:
```
svm_model.score(X_test,y_test)
```

Out[22]:
```
0.8205128205128205
```

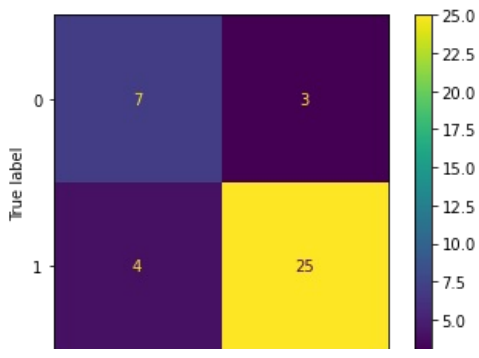
In [23]:
```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
```

In [24]:
```
y_prediction=svm_model.predict(X_test)
```

In [25]:
```
cm=confusion_matrix(y_test,y_prediction)
```

In [26]:
```
cm1=ConfusionMatrixDisplay(cm)
cm1.plot()
```

Out[26]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x208a60de070>
```



START YOUR PREDICTION

In [46]:
```
input_data = ([[200.07600,206.89600,210.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.01098,0.09700,0.00563,0.0

prediction = svm_model.predict(input_data)
print(prediction)

if (prediction[0]== 0):
  print("The Person does not have Parkinsons Disease")

else:
  print("The Person has Parkinsons")
```

```
[0]
The Person does not have Parkinsons Disease
```

MODEL 2- NAVIE BIAS CLASSIFIER

In [28]:
```
from sklearn.naive_bayes import GaussianNB
```

In [29]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [35]:
```
gnb_model=GaussianNB(var_smoothing=2e-9)
```

In [36]:
```
gnb_model.fit(X_train,y_train)
```

Out[36]:
```
GaussianNB(var_smoothing=2e-09)
```

In [37]:
```
gnb_model.score(X_test,y_test)
```

Out[37]:
```
0.6666666666666666
```

In [38]:
```
y_prediction=gnb_model.predict(X_test)
```

In [39]:
```
y_prediction
```

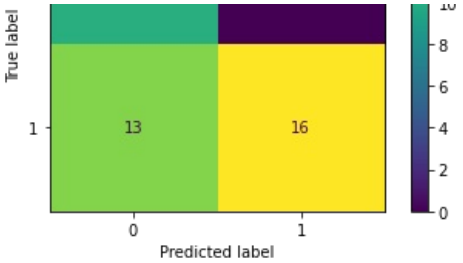
Out[39]:
```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1], dtype=int64)
```

In [40]:
```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
```

In [41]:
```
cm=confusion_matrix(y_test,y_prediction)
```

In [42]:
```
cm1=ConfusionMatrixDisplay(cm)
cm1.plot()
```

Out[42]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x208a8daf040>
```



START YOUR PREDICTION

In [47]:

```
input_data = ([[190.07600,206.89600,180.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.01098,0.09700,0.00563,0.0

prediction = svm_model.predict(input_data)
print(prediction)

if (prediction[0]== 0):
  print("The Person does not have Parkinsons Disease")

else:
  print("The Person has Parkinsons")
```

```
[0]
The Person does not have Parkinsons Disease
```

In [ ]: