Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import metrics
```

Importing Dataset

In [2]:
```
df=pd.read_csv("G:\\Detection of Parkinsons Disease\\parkinsons.csv")
display (df)
```

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Al
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.000
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.000
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.000
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.000
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.000
...	...	...	...	...	...	
190	phon_R01_S50_2	174.188	230.978	94.261	0.00459	0.000
191	phon_R01_S50_3	209.516	253.017	89.488	0.00564	0.000
192	phon_R01_S50_4	174.688	240.005	74.287	0.01360	0.000
193	phon_R01_S50_5	198.764	396.961	74.904	0.00740	0.000
194	phon_R01_S50_6	214.289	260.277	77.973	0.00567	0.000

195 rows × 24 columns

Displaying the Data Set Profile Report

In [3]:
```
import ydata_profiling as pf
display(pf.ProfileReport(df))
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	24
Number of observations	195
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	36.7 KiB
Average record size in memory	192.7 B

Variable types

Categorical	2
Numeric	22

Alerts

`name` has a high cardinality: 195 distinct values	**High cardinality**
`MDVP:Fo(Hz)` is highly overall correlated with `MDVP:Fhi(Hz)` and 2 other fields (MDVP:Fhi(Hz), MDVP:Jitter(Abs), status)	**High correlation**

In [4]:
```
display (df.shape)
```

```
(195, 24)
```

In [5]:
```
print (len(df))
```

```
195
```

Displaying the Dataset and Data Type

In [6]:
```
display (df.dtypes )
```

```
name                 object
MDVP:Fo(Hz)         float64
MDVP:Fhi(Hz)        float64
MDVP:Flo(Hz)        float64
MDVP:Jitter(%)      float64
MDVP:Jitter(Abs)    float64
MDVP:RAP            float64
MDVP:PPQ            float64
Jitter:DDP          float64
MDVP:Shimmer        float64
MDVP:Shimmer(dB)    float64
Shimmer:APQ3        float64
Shimmer:APQ5        float64
MDVP:APQ            float64
Shimmer:DDA         float64
NHR                 float64
HNR                 float64
status                int64
RPDE                float64
DFA                 float64
spread1             float64
spread2             float64
D2                  float64
PPE                 float64
dtype: object
```

displaying complete Information and Discription of Dataset

In [7]:
```
print (df.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
None
```

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

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP
count	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
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440

8 rows × 23 columns

Looking for Mising Values

And Performing EDA Before spliting the Data into x and y

In [9]:
```
display (df.isna().sum() )
```

```
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
```

Checking for columns that we have in our dataset

In [10]:
```
print (df.columns)
```

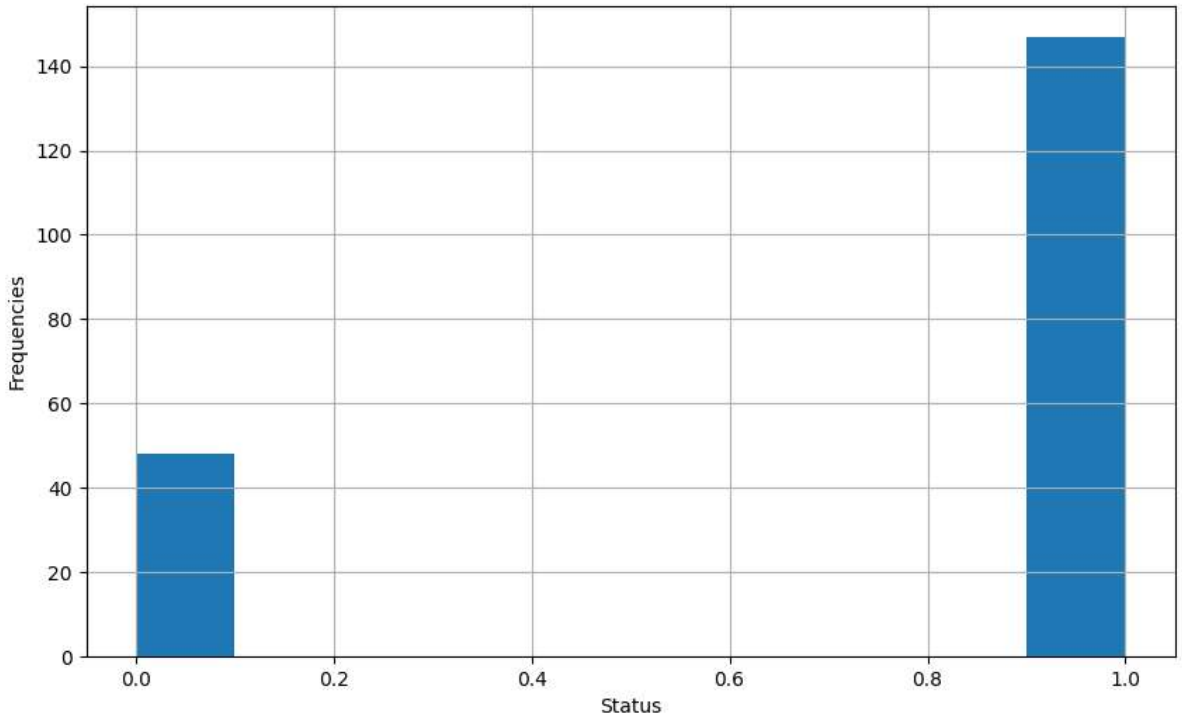
```
Index(['name', 'MDVP:Fo(Hz)', 'MDVP:Fhi(Hz)', 'MDVP:Flo(Hz)', 'MDVP:Jitter(%)',
       'MDVP:Jitter(Abs)', 'MDVP:RAP', 'MDVP:PPQ', 'Jitter:DDP',
       'MDVP:Shimmer', 'MDVP:Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5',
       'MDVP:APQ', 'Shimmer:DDA', 'NHR', 'HNR', 'status', 'RPDE', 'DFA',
       'spread1', 'spread2', 'D2', 'PPE'],
      dtype='object')
```

Checking the status data and plotting status with NHR, HNR and RPDE.

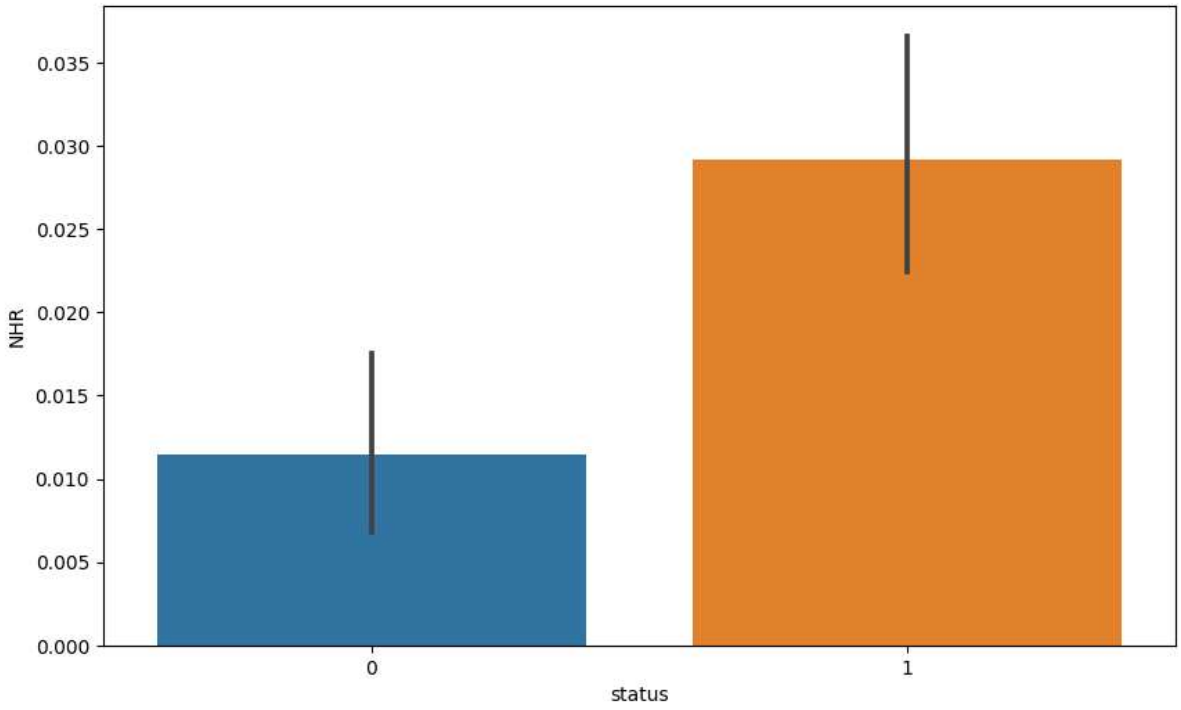
In [11]:
```
print (df['status'])
```

```
0      1
1      1
2      1
3      1
4      1
      ..
190    0
191    0
192    0
193    0
194    0
Name: status, Length: 195, dtype: int64
```

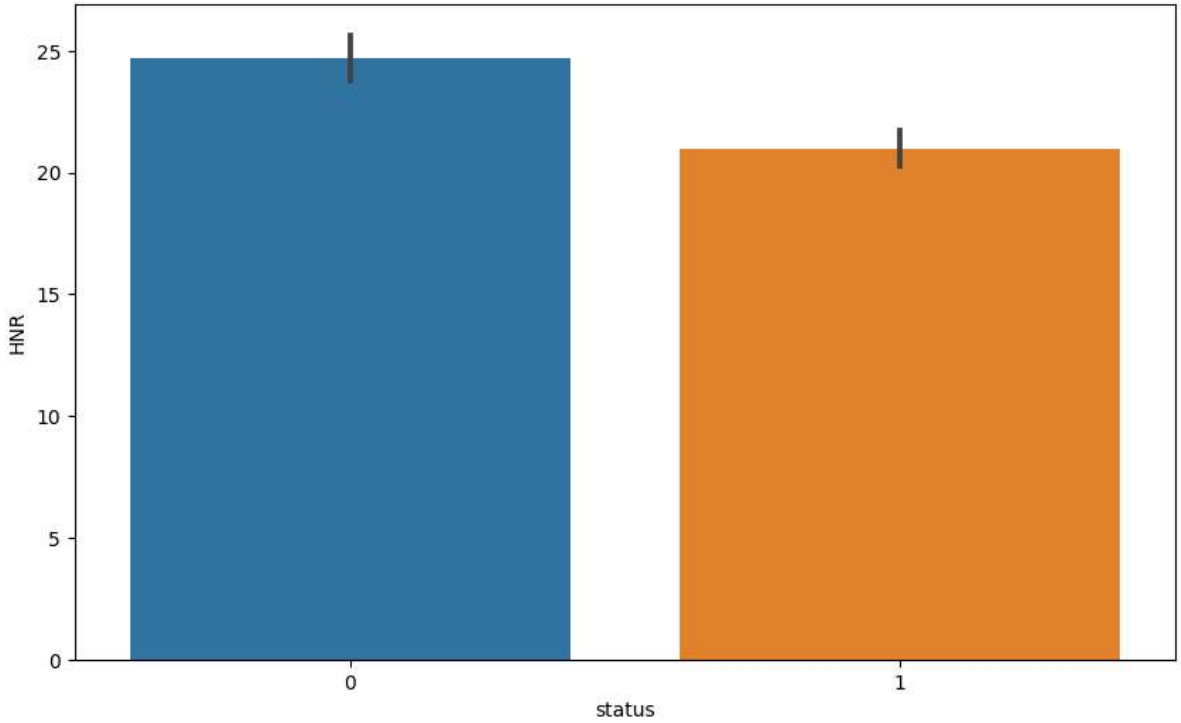
In [12]:
```
plt.figure(figsize=(10, 6))
df.status.hist()
plt.xlabel('Status')
plt.ylabel('Frequencies')
plt.plot()
plt.show()
```



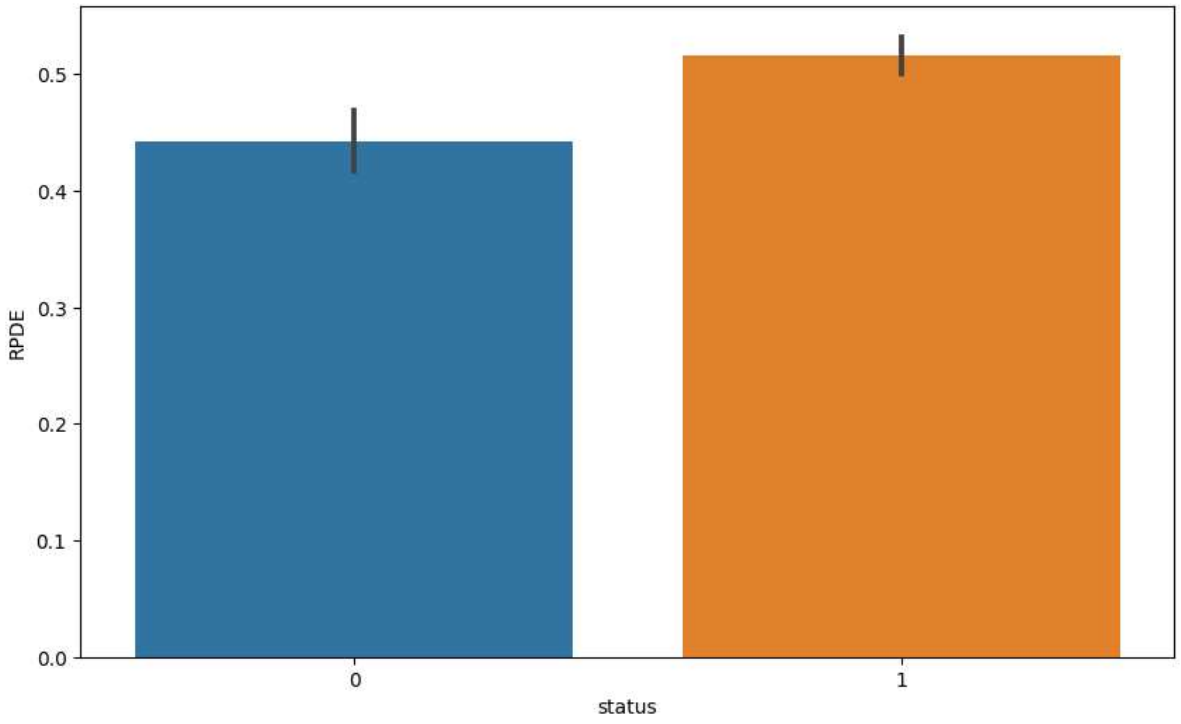
In [13]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x="status",y="NHR",data=df);
plt.show()
```



In [14]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x="status",y="HNR",data=df);
plt.show()
```



In [15]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x="status",y="RPDE",data=df);
plt.show()
```

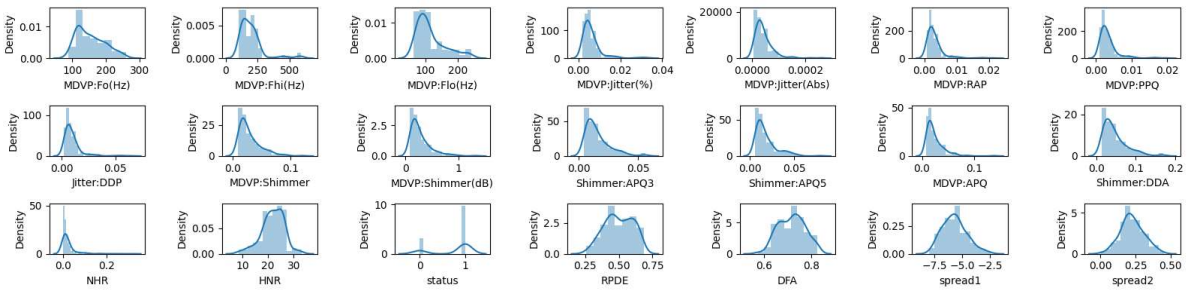


In [16]:
```
print (df.columns)
```

```
Index(['name', 'MDVP:Fo(Hz)', 'MDVP:Fhi(Hz)', 'MDVP:Flo(Hz)', 'MDVP:Jitter(%)',
       'MDVP:Jitter(Abs)', 'MDVP:RAP', 'MDVP:PPQ', 'Jitter:DDP',
       'MDVP:Shimmer', 'MDVP:Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5',
       'MDVP:APQ', 'Shimmer:DDA', 'NHR', 'HNR', 'status', 'RPDE', 'DFA',
       'spread1', 'spread2', 'D2', 'PPE'],
      dtype='object')
```

In [17]:
```
import warnings
warnings.filterwarnings('ignore')
rows=3
cols=7
fig, ax=plt.subplots(nrows=rows,ncols=cols,figsize=(16,4))
col=df.columns
index=1
for i in range(rows):
    for j in range(cols):
        sns.distplot(df[col[index]],ax=ax[i][j])
        index=index+1

plt.tight_layout()
plt.show()
```



Displaying the Cor-relation and Plotting the Heatmap

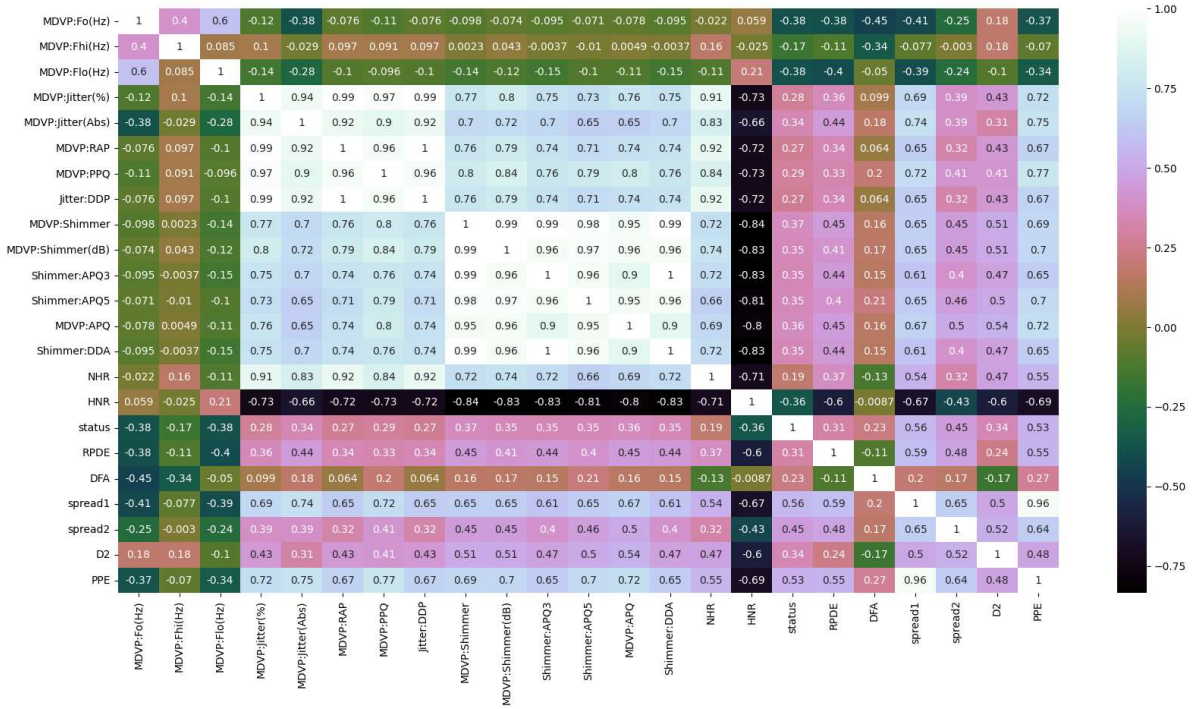
In [18]:
```
corr = df.corr()
display (corr)
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs
MDVP:Fo(Hz)	1.000000	0.400985	0.596546	-0.118003	-0.38202
MDVP:Fhi(Hz)	0.400985	1.000000	0.084951	0.102086	-0.02919
MDVP:Flo(Hz)	0.596546	0.084951	1.000000	-0.139919	-0.27781
MDVP:Jitter(%)	-0.118003	0.102086	-0.139919	1.000000	0.93571
MDVP:Jitter(Abs)	-0.382027	-0.029198	-0.277815	0.935714	1.00000
MDVP:RAP	-0.076194	0.097177	-0.100519	0.990276	0.92291
MDVP:PPQ	-0.112165	0.091126	-0.095828	0.974256	0.89777
Jitter:DDP	-0.076213	0.097150	-0.100488	0.990276	0.92291
MDVP:Shimmer	-0.098374	0.002281	-0.144543	0.769063	0.70332
MDVP:Shimmer(dB)	-0.073742	0.043465	-0.119089	0.804289	0.71660
Shimmer:APQ3	-0.094717	-0.003743	-0.150747	0.746625	0.69715
Shimmer:APQ5	-0.070682	-0.009997	-0.101095	0.725561	0.64896
MDVP:APQ	-0.077774	0.004937	-0.107293	0.758255	0.64879
Shimmer:DDA	-0.094732	-0.003733	-0.150737	0.746635	0.69717
NHR	-0.021981	0.163766	-0.108670	0.906959	0.83497
HNR	0.059144	-0.024893	0.210851	-0.728165	-0.65681
status	-0.383535	-0.166136	-0.380200	0.278220	0.33865
RPDE	-0.383894	-0.112404	-0.400143	0.360673	0.44183
DFA	-0.446013	-0.343097	-0.050406	0.098572	0.17503
spread1	-0.413738	-0.076658	-0.394857	0.693577	0.73577
spread2	-0.249450	-0.002954	-0.243829	0.385123	0.38854
D2	0.177980	0.176323	-0.100629	0.433434	0.31069
PPE	-0.372356	-0.069543	-0.340071	0.721543	0.74816

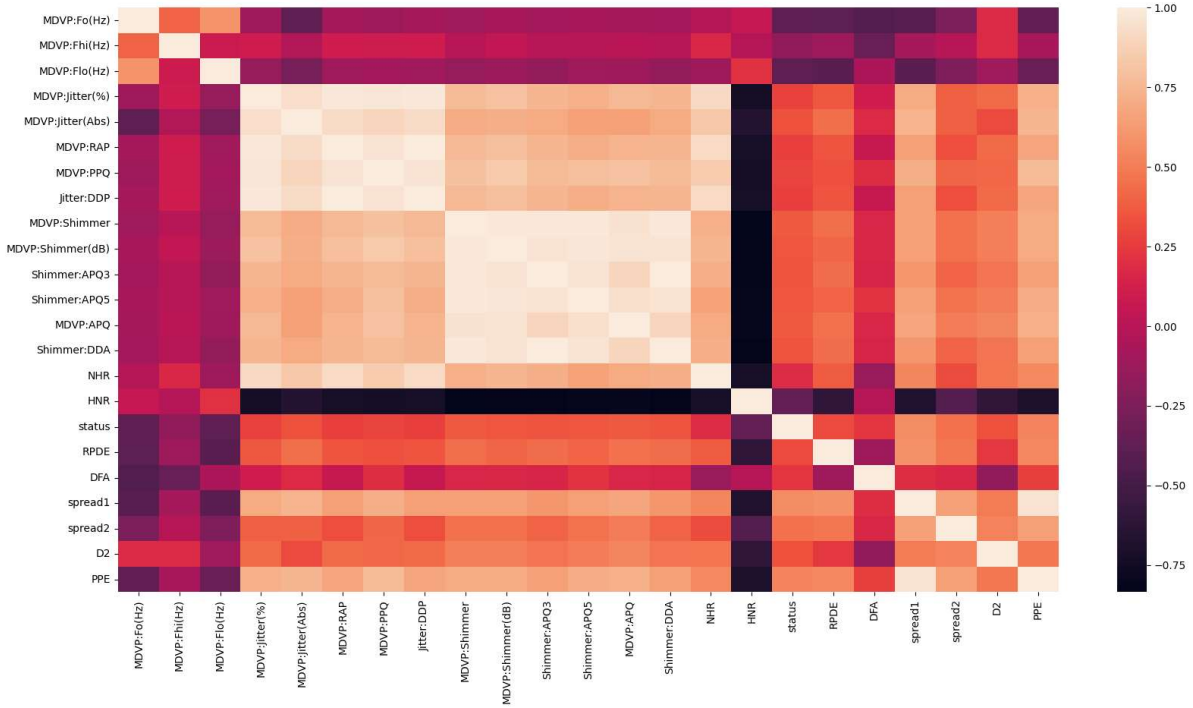
23 rows × 23 columns

In [19]:
```
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 20,10

sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, cmap='cubehe
plt.show()
```



In [20]:
```
rcParams['figure.figsize'] = 20,10
sns.heatmap(corr)
plt.show()
```



Checking our Dataset

In [21]:
```
df
```

Out[21]:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Al
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.000
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.000
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.000
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.000
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.000
...	...	...	...	...	...	
190	phon_R01_S50_2	174.188	230.978	94.261	0.00459	0.000
191	phon_R01_S50_3	209.516	253.017	89.488	0.00564	0.000
192	phon_R01_S50_4	174.688	240.005	74.287	0.01360	0.000
193	phon_R01_S50_5	198.764	396.961	74.904	0.00740	0.000
194	phon_R01_S50_6	214.289	260.277	77.973	0.00567	0.000

195 rows × 24 columns

Now Droping the Data which isn't needed in our Prediction

In [22]:
```
df.drop(['name'],axis=1,inplace=True)
display (df)
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP
0	119.992	157.302	74.997	0.00784	0.00007	0.00370
1	122.400	148.650	113.819	0.00968	0.00008	0.00465
2	116.682	131.111	111.555	0.01050	0.00009	0.00544
3	116.676	137.871	111.366	0.00997	0.00009	0.00502
4	116.014	141.781	110.655	0.01284	0.00011	0.00655
...	...	...	...	...	...	...
190	174.188	230.978	94.261	0.00459	0.00003	0.00263
191	209.516	253.017	89.488	0.00564	0.00003	0.00331
192	174.688	240.005	74.287	0.01360	0.00008	0.00624
193	198.764	396.961	74.904	0.00740	0.00004	0.00370
194	214.289	260.277	77.973	0.00567	0.00003	0.00295

195 rows × 23 columns

After Droping name from Dataset we will now split the Data in x and y

In [23]:
```
X=df.drop(labels=['status'],axis=1)
display (X.head())
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MI
0	119.992	157.302	74.997	0.00784	0.00007	0.00370	
1	122.400	148.650	113.819	0.00968	0.00008	0.00465	
2	116.682	131.111	111.555	0.01050	0.00009	0.00544	
3	116.676	137.871	111.366	0.00997	0.00009	0.00502	
4	116.014	141.781	110.655	0.01284	0.00011	0.00655	

5 rows × 22 columns

In [24]:
```
Y=df['status']
display (Y.head())
```

```
0    1
1    1
2    1
3    1
4    1
Name: status, dtype: int64
```

Now we will split x and y in Train Test Split from Sklearn Library

In [25]:
```
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=40)
print (X.shape,Y.shape)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

```
(195, 22) (195,)
(156, 22) (39, 22) (156,) (39,)
```

Now Creating Prediction Model for our Train Test Data

And Checking the Accuracy

In [26]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
log_reg = LogisticRegression().fit(X_train, Y_train)
#predict on train
```

```
train_preds = log_reg.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds))

#predict on test
test_preds = log_reg.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds))
print('-'*50)
```

```
Model accuracy on train is:  0.8717948717948718
Model accuracy on test is:  0.8461538461538461
--------------------------------------------------
```

In [27]:

```
from sklearn.ensemble import RandomForestClassifier
RF=RandomForestClassifier().fit(X_train,Y_train)
#predict on train
train_preds2 = RF.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds2))

#predict on test
test_preds2 = RF.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds2))
```

```
Model accuracy on train is:  1.0
Model accuracy on test is:  0.8717948717948718
```

In [28]:

```
from sklearn.tree import DecisionTreeClassifier
#fit the model on train data
DT = DecisionTreeClassifier().fit(X,Y)

#predict on train
train_preds3 = DT.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds3))

#predict on test
test_preds3 = DT.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds3))
print('-'*50)
```

```
Model accuracy on train is:  1.0
Model accuracy on test is:  1.0
--------------------------------------------------
```

In [29]:

```
from sklearn.naive_bayes import GaussianNB
#fit the model on train data
NB=GaussianNB()
NB.fit(X_train,Y_train)
#predict on train
train_preds4 = NB.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds4))

#predict on test
test_preds4 = NB.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds4))
print('-'*50)
```

```
Model accuracy on train is:  0.7307692307692307
Model accuracy on test is:  0.6923076923076923
------------------------------------------------
```

In [30]:
```python
from sklearn.neighbors import KNeighborsClassifier
#fit the model on train data
KNN = KNeighborsClassifier().fit(X_train,Y_train)
#predict on train
train_preds5 = KNN.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds5))

#predict on test
test_preds5 = KNN.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds5))
print('-'*50)
```

```
Model accuracy on train is:  0.9102564102564102
Model accuracy on test is:  0.8461538461538461
------------------------------------------------
```

In [31]:
```python
from sklearn.svm import SVC
#fit the model on train data
SVM = SVC(kernel='linear')
SVM.fit(X_train, Y_train)

#predict on train
train_preds6 = SVM.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds6))

#predict on test
test_preds6 = SVM.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds5))
```

```
Model accuracy on train is:  0.8782051282051282
Model accuracy on test is:  0.8461538461538461
```

Cross Validation

In [32]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
SVM = SVC(kernel='linear')
SVM.fit(X_train, Y_train)
KNN = KNeighborsClassifier().fit(X_train,Y_train)
log_reg = LogisticRegression().fit(X_train, Y_train)
NB=GaussianNB()
NB.fit(X_train,Y_train)
DT = DecisionTreeClassifier().fit(X,Y)
RF=RandomForestClassifier().fit(X_train,Y_train)

#predict on train
train_preds = log_reg.predict(X_train)
```

```
#accuracy on train
print("Logistic Regression Model accuracy on train is: ", accuracy_score(Y_train, t
#predict on test
test_preds = log_reg.predict(X_test)
#accuracy on test
print("Logistic Regression Model accuracy on test is: ", accuracy_score(Y_test, tes
print('-'*50)

#predict on train
train_preds5 = KNN.predict(X_train)
#accuracy on train
print("KNN Model accuracy on train is: ", accuracy_score(Y_train, train_preds5))
#predict on test
test_preds5 = KNN.predict(X_test)
#accuracy on test
print("KNN Model accuracy on test is: ", accuracy_score(Y_test, test_preds5))
print('-'*50)

#predict on train
train_preds4 = NB.predict(X_train)
#accuracy on train
print("Navie Bayce Model accuracy on train is: ", accuracy_score(Y_train, train_pre
#predict on test
test_preds4 = NB.predict(X_test)
#accuracy on test
print("Navie Bayce Model accuracy on test is: ", accuracy_score(Y_test, test_preds4
print('-'*50)


#predict on train
train_preds3 = DT.predict(X_train)
#accuracy on train
print("Decision tree Model accuracy on train is: ", accuracy_score(Y_train, train_p
#predict on test
test_preds3 = DT.predict(X_test)
#accuracy on test
print("Decision tree Model accuracy on test is: ", accuracy_score(Y_test, test_prec
print('-'*50)

#predict on train
train_preds2 = RF.predict(X_train)
#accuracy on train
print("Random Forest Model accuracy on train is: ", accuracy_score(Y_train, train_p
#predict on test
test_preds2 = RF.predict(X_test)
#accuracy on test
print("Random Forest Model accuracy on test is: ", accuracy_score(Y_test, test_prec
print('-'*50)

#predict on train
train_preds6 = SVM.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds6))

#predict on test
test_preds6 = SVM.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds6))
print('-'*50)
```

```
Logistic Regression Model accuracy on train is:  0.8717948717948718
Logistic Regression Model accuracy on test is:  0.8461538461538461
--------------------------------------------------
KNN Model accuracy on train is:  0.9102564102564102
KNN Model accuracy on test is:  0.8461538461538461
--------------------------------------------------
Navie Bayce Model accuracy on train is:  0.7307692307692307
Navie Bayce Model accuracy on test is:  0.6923076923076923
--------------------------------------------------
Decision tree Model accuracy on train is:  1.0
Decision tree Model accuracy on test is:  1.0
--------------------------------------------------
Random Forest Model accuracy on train is:  1.0
Random Forest Model accuracy on test is:  0.8974358974358975
--------------------------------------------------
Model accuracy on train is:  0.8782051282051282
Model accuracy on test is:  0.8974358974358975
--------------------------------------------------
```

As we can see in the above Cross Validation Code we have the highest Accuracy with Decision Tree Model. Therefore we will proceed with Decision Tree Model and Display the Confusion Matrix, Classification Report And Wrong Prediction Made

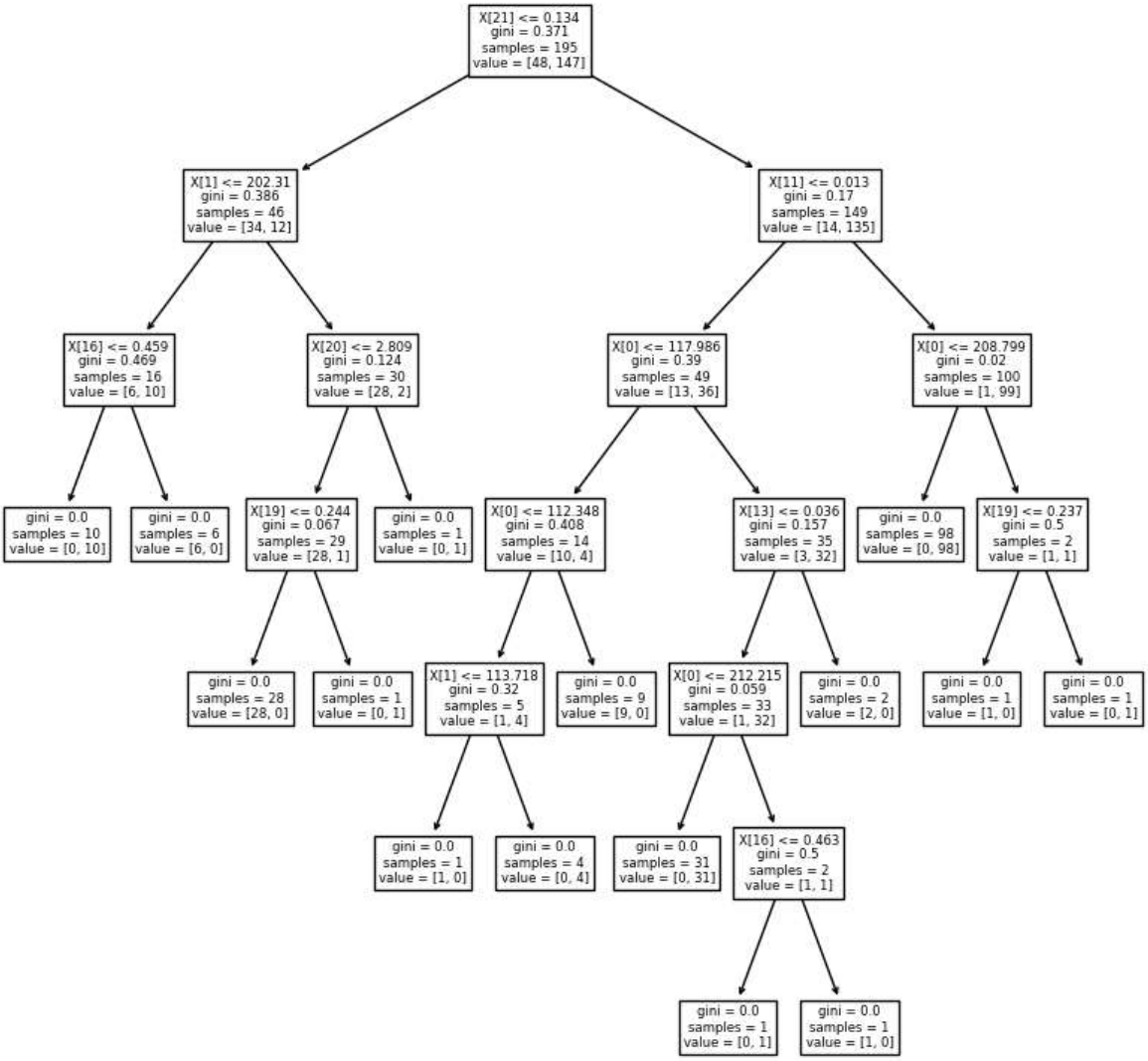
Creating Decision Tree Model and Making Prediction.

In [33]:
```
from sklearn.tree import DecisionTreeClassifier
DT = DecisionTreeClassifier()
DT.fit(X,Y)
```

Out[33]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

Plotting the Decision made by Decision Tree Model

In [34]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(10,10))
plot_tree(DT)
plt.show()
```



Evaluating the Decision Tree Model

In [35]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Making Prediction on Train Data and Displaying the Accuracy

In [36]:
```
#predict on train
train_preds3 = DT.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds3))
```

```
Model accuracy on train is:  1.0
```

Making Prediction on Test Data and Displaying the Accuracy

In [37]:
```
test_preds3 = DT.predict(X_test)
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds3))
```

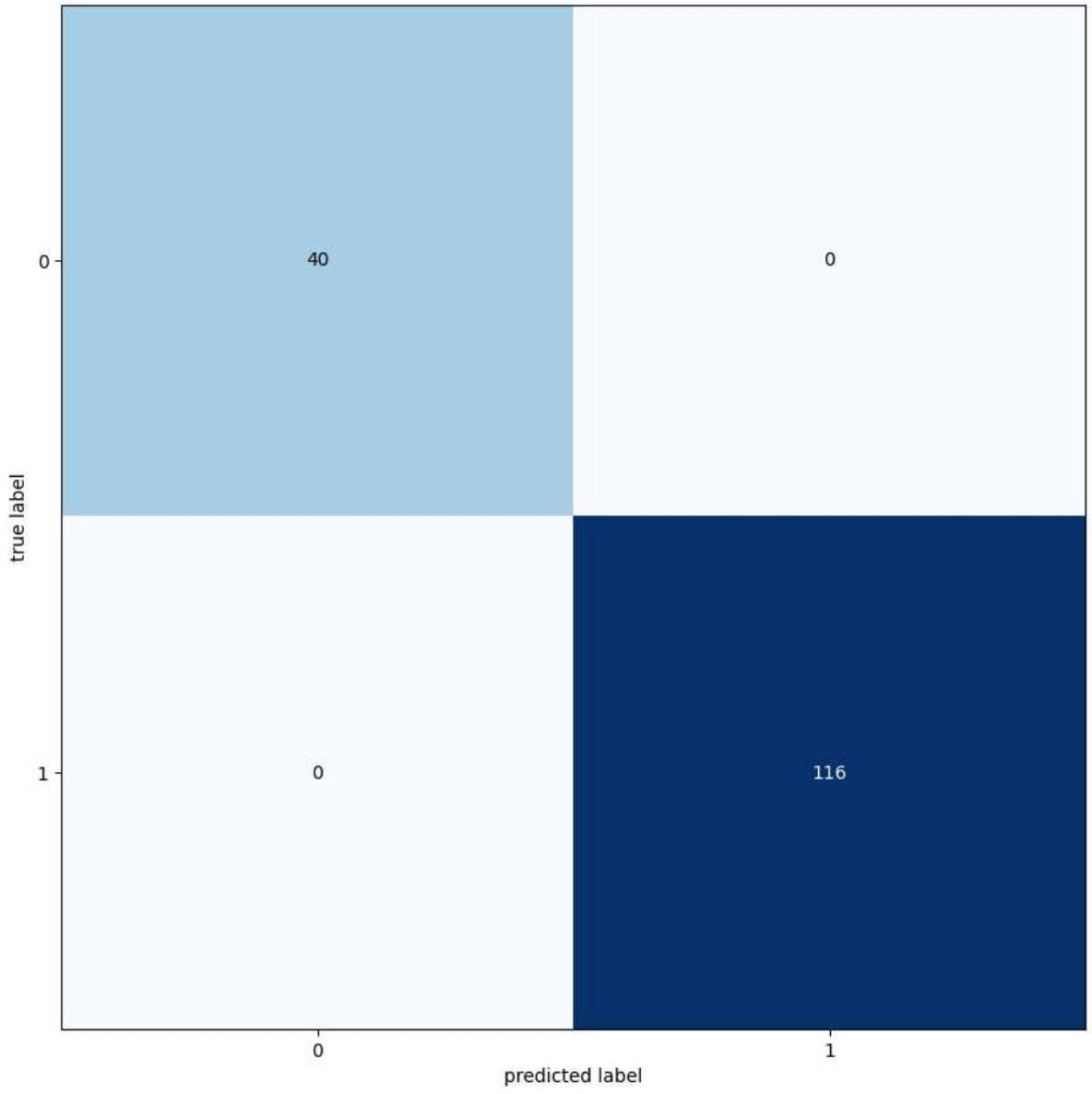
```
Model accuracy on test is:  1.0
```

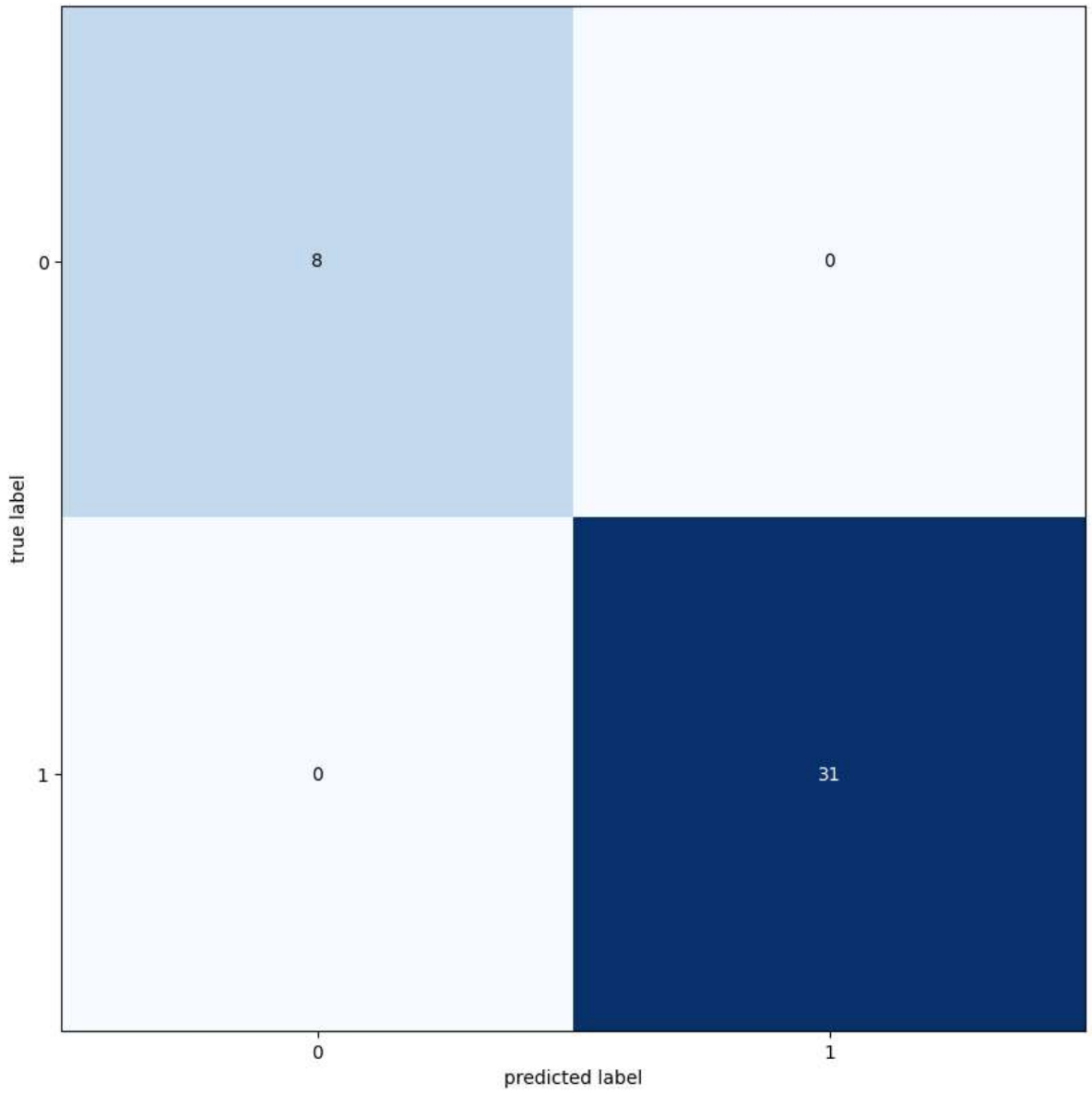
Plotting Confusion Matrics

In [38]:
```
from mlxtend.plotting import plot_confusion_matrix
print("confusion_matrix train is: \n", confusion_matrix(Y_train, train_preds3))
print("confusion_matrix test is:\n ", confusion_matrix(Y_test, test_preds3))
```

```
confusion_matrix train is:
 [[ 40   0]
 [  0 116]]
confusion_matrix test is:
  [[ 8  0]
 [ 0 31]]
```

In [39]:
```
Cm = confusion_matrix(Y_train, train_preds3)
CM= confusion_matrix(Y_test, test_preds3)
plot_confusion_matrix(Cm)
plot_confusion_matrix(CM)
plt.show()
```





Classification Report on Train data

In [40]:
```
print(classification_report (Y_train, train_preds3))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      1.00      1.00       116

    accuracy                           1.00       156
   macro avg       1.00      1.00      1.00       156
weighted avg       1.00      1.00      1.00       156
```

Calssification Report on Test Data

In [41]:
```
print(classification_report (Y_test, test_preds3))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	8
1	1.00	1.00	1.00	31
accuracy			1.00	39
macro avg	1.00	1.00	1.00	39
weighted avg	1.00	1.00	1.00	39

Wrong Predictions made on Test Data

In [42]:
```
print((Y_test !=test_preds3).sum(),'/',((Y_test == test_preds3).sum()+(Y_test != te
```

```
0 / 39
```

Wrong Predictions made on Train Data

In [43]:
```
print((Y_train !=train_preds3).sum(),'/',((Y_train == train_preds3).sum()+(Y_train
```

```
0 / 156
```