NAME:N.BHARATH

COLLEGE:MADANAPALLE INSTITUTE OF TECHNOLOGY AND SCIENCE

YEAR:2nd YEAR

ACADEMIC YEAR:2021-2023

MAJOR PROJECT-1

In

In [2]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import os
import warnings
```

In [3]:

```
os.getcwd()
```

Out[3]:

```
'C:\\Users\\BHARATH'
```

In [4]:

```
os.chdir ('C:\\Users\\BHARATH\\OneDrive\\Desktop')
os.getcwd()
```

Out[4]:

```
'C:\\Users\\BHARATH\\OneDrive\\Desktop'
```

[5]:

```
parkinsons_data=pd.read_csv('parkinsons.data')
display(parkinsons_data)
```

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(A
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00
...	...	...	...	...	...	
190	phon_R01_S50_2	174.188	230.978	94.261	0.00459	0.00
191	phon_R01_S50_3	209.516	253.017	89.488	0.00564	0.00
192	phon_R01_S50_4	174.688	240.005	74.287	0.01360	0.00
193	phon_R01_S50_5	198.764	396.961	74.904	0.00740	0.00

In

194	phon_R01_S50_6	214.289	260.277	77.973	0.00567	0.00

195 rows × 24 columns

In [6]:

```
import pandas_profiling as pf
display(pf.ProfileReport(parkinsons_data))
```

```
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] In
```

[7]:

```
parkinsons_data.head()
```

Out[7]:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.0000
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.0000
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.0000
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.0000
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.0001

5 rows × 24 columns

[8]:

```
parkinsons_data.shape
```

Out[8]:

```
(195, 24)
```

In [9]:

```
parkinsons_data.columns
```

Out[9]:

```
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 [10]:
```

In

```
parkinsons_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
```

[11]:

```
# checking for missing values in each column
parkinsons_data.isnull().sum()
```

Out[11]:

```
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
```

```
In
0 status
0 RPDE
0 DFA
0 spread1
0 spread2
0 D2
0 PPE
0 dtype: int64 In
[12]:
```

```
# getting some statistical measures about the data
parkinsons_data.describe()
```

```
Out[12]:
```

	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

```
[13]:
```

```
# distribution of target variable
parkinsons_data['status'].value_counts()
```

```
Out[13]:

1    147
0     48
Name: status, dtype: int64
```

1--> Parkinson's Positive

0--> Parkinson's Negative (Healthy)

```
In [14]:
```

```
# grouping the data based on the target variable
parkinsons_data.groupby('status').mean()
```

```
Out[14]:
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP
status						

In

0	181.937771	223.636750	145.207292	0.003866	0.000023	0.001925
1	145.180762	188.441463	106.893558	0.006989	0.000051	0.003757

2 rows × 22 columns

Data Pre-Processing

Separating the features & Target

[15]:

```
X = parkinsons_data.drop(columns=['name','status'], axis=1)
Y = parkinsons_data['status']
print(X)
```

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

     MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  \
0             0.00007   0.00370   0.00554     0.01109       0.04374
1             0.00008   0.00465   0.00696     0.01394       0.06134
2             0.00009   0.00544   0.00781     0.01633       0.05233
3             0.00009   0.00502   0.00698     0.01505       0.05492
4             0.00011   0.00655   0.00908     0.01966       0.06425    ..
              ...       ...       ...         ...           ...
190           0.00003   0.00263   0.00259     0.00790       0.04087
191           0.00003   0.00331   0.00292     0.00994       0.02751
192           0.00008   0.00624   0.00564     0.01873       0.02308
193           0.00004   0.00370   0.00390     0.01109       0.02296
194           0.00003   0.00295   0.00317     0.00885       0.01884

     MDVP:Shimmer(dB)  ...  MDVP:APQ  Shimmer:DDA      NHR     HNR      RPDE \
0               0.426  ...   0.02971      0.06545  0.02211  21.033  0.414783
1               0.626  ...   0.04368      0.09403  0.01929  19.085  0.458359
2               0.482  ...   0.03590      0.08270  0.01309  20.651  0.429895
3               0.517  ...   0.03772      0.08771  0.01353  20.644  0.434969
4               0.584  ...   0.04465      0.10470  0.01767  19.649  0.417356
                ..                ...  ...       ...          ...      ...
                ...       ...
190             0.405  ...   0.02745      0.07008  0.02764  19.517  0.448439
191             0.263  ...   0.01879      0.04812  0.01810  19.147  0.431674
192             0.256  ...   0.01667      0.03804  0.10715  17.883  0.407567
193             0.241  ...   0.01588      0.03794  0.07223  19.020  0.451221
194             0.190  ...   0.01373      0.03078  0.04398  21.209  0.462803
```

```
In
          DFA   spread1   spread2        D2       PPE
0    0.815285 -4.813031  0.266482  2.301442  0.284654
1    0.819521 -4.075192  0.335590  2.486855  0.368674
2    0.825288 -4.443179  0.311173  2.342259  0.332634
3    0.819235 -4.117501  0.334147  2.405554  0.368975
4    0.823484 -3.747787  0.234513  2.332180  0.410335
..        ...       ...       ...       ...       ...
190  0.657899 -6.538586  0.121952  2.657476  0.133050
191  0.683244 -6.195325  0.129303  2.784312  0.168895
192  0.655683 -6.787197  0.158453  2.679772  0.131728
193  0.643956 -6.744577  0.207454  2.138608  0.123306
194  0.664357 -5.724056  0.190667  2.555477  0.148569

[195 rows x 22 columns]
```

In [16]:

```
print(Y)
```

```
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
```

Splitting the data to training data & Test data

In [17]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=2) print(X.shape, X_train.shape, X_test.shape)
```

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

Data Standardization

In [18]:

```
scaler = StandardScaler()
scaler.fit(X_train)
```

Out[18]:

```
StandardScaler()
```

In [19]:

```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
print(X_train)
```

```
[[ 0.63239631 -0.02731081 -0.87985049 ... -0.97586547 -0.55160318
 0.07769494]
 [-1.05512719 -0.83337041 -0.9284778  ...  0.3981808  -0.61014073
 0.39291782]
 [ 0.02996187 -0.29531068 -1.12211107 ... -0.43937044 -0.62849605  -
 0.50948408]
 ...
 [-0.9096785  -0.6637302  -0.160638   ...  1.22001022 -0.47404629  -
 0.2159482 ]
 [-0.35977689  0.19731822 -0.79063679 ... -0.17896029 -0.47272835
 0.28181221]
```

In

```
 [ 1.01957066  0.19922317 -0.61914972 ... -0.716232    1.23632066  -
 0.05829386]]
```

Model Training

Support Vector Machine Model

In [20]:

```
model = svm.SVC(kernel='linear')
#training the SVM model with training data
model.fit(X_train, Y_train)
```

Out[20]:

```
SVC(kernel='linear')
```

Model Evaluation

Accuracy Score

In [21]:

```
# accuracy score on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
print('Acuuracy score of training data :', training_data_accuracy)
```

```
Acuuracy score of training data : 0.8846153846153846
```

In [22]:

```
# accuracy score on training data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
print('Acuuracy score of test data :', test_data_accuracy)
```

```
Acuuracy score of test data : 0.8717948717948718
```

Predictive System

[23]:

```
input_data = (197.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.01098

# changing input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the data
std_data = scaler.transform(input_data_reshaped)
prediction = model.predict(std_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 [24]:

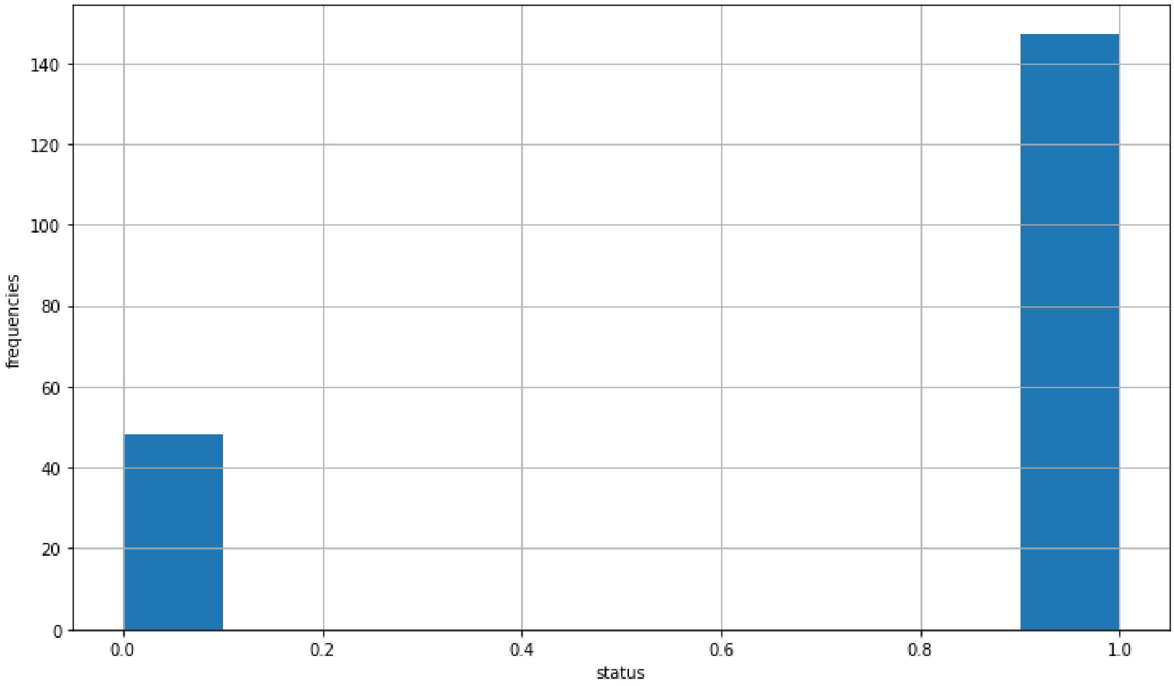
```
input_data = (95.05600,120.10300,91.22600,0.00532,0.00006,0.00268,0.00332,0.00803,0.02838,0

# changing input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the data
std_data = scaler.transform(input_data_reshaped)
prediction = model.predict(std_data)
print(prediction)
if (prediction[0] == 0):
    print("The Person does not have Parkinsons Disease")
else:
    print("The Person has Parkinsons")
```

```
[1]
The Person has Parkinsons
```

Representing the Dataset in Bar plots

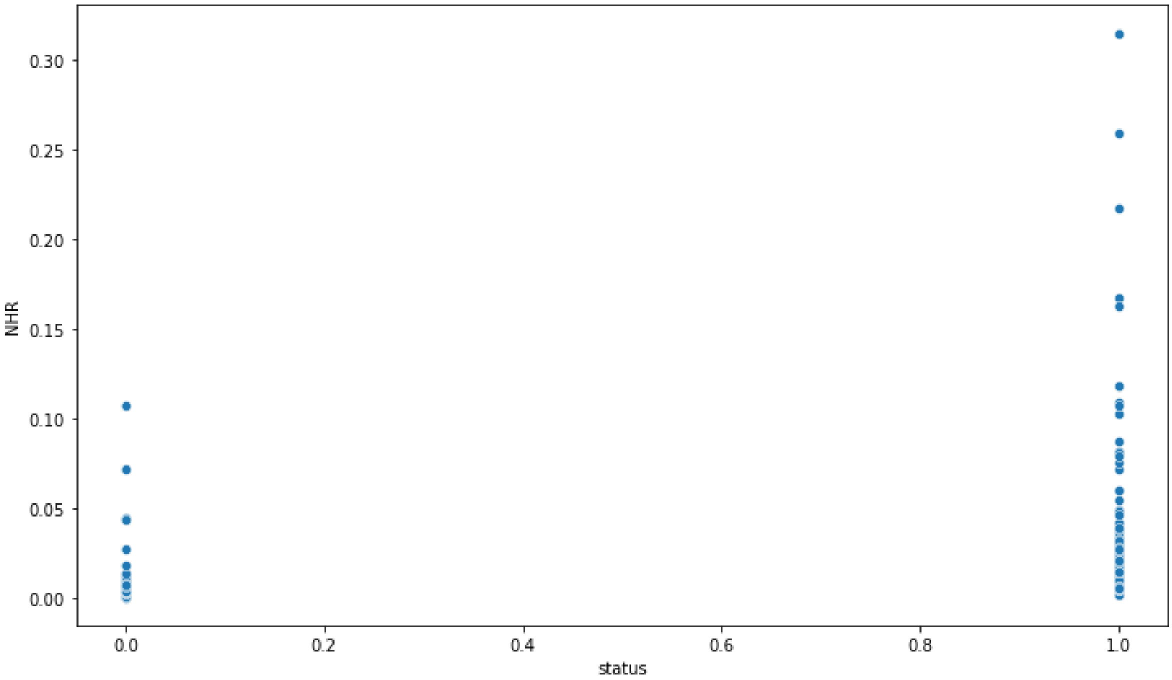
In [36]:

```
plt.figure(figsize=(12,7))
parkinsons_data.status.hist()
plt.xlabel('status')
plt.ylabel('frequencies')
plt.plot()
plt.show()
```



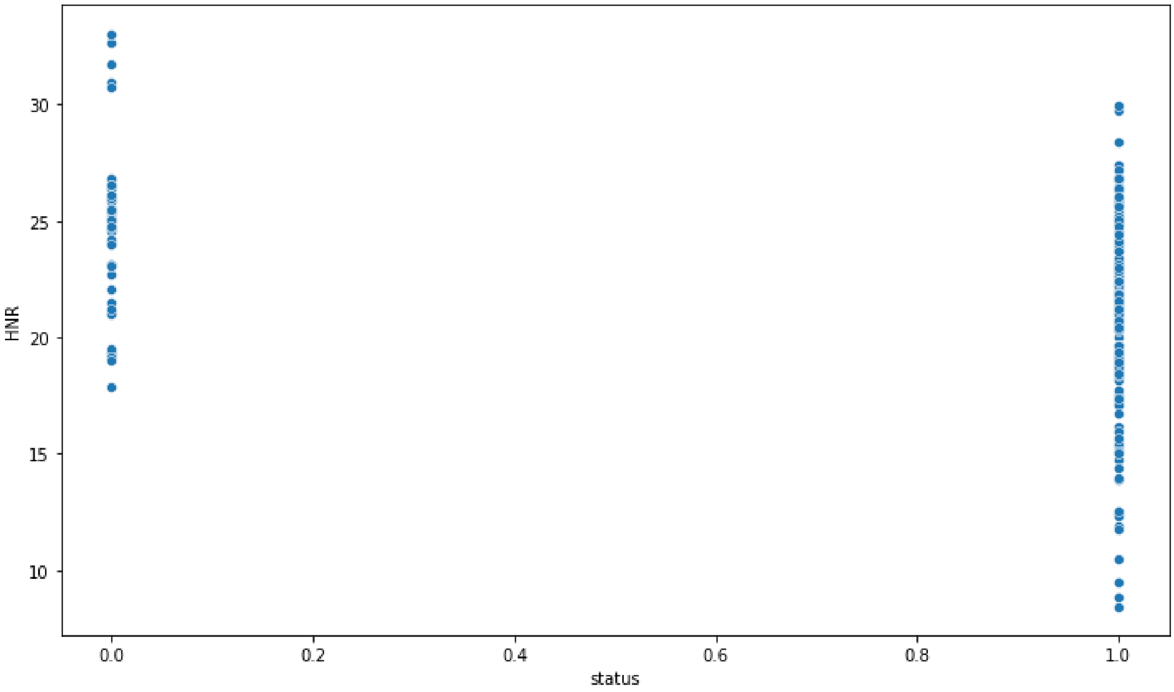
In [26]:

```
plt.figure(figsize=(12,7))
sns.scatterplot(x="status",y="NHR",data=parkinsons_data);
```



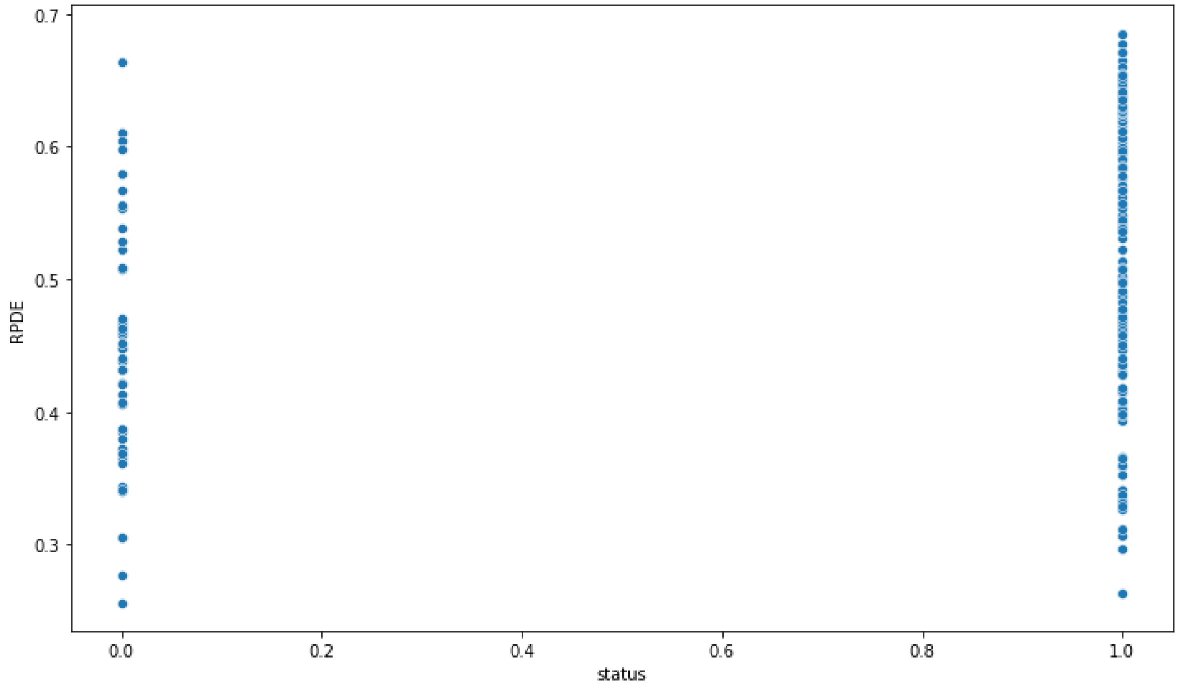
In [27]:

```
plt.figure(figsize=(12,7))
sns.scatterplot(x="status",y="HNR",data=parkinsons_data);
```



In [28]:

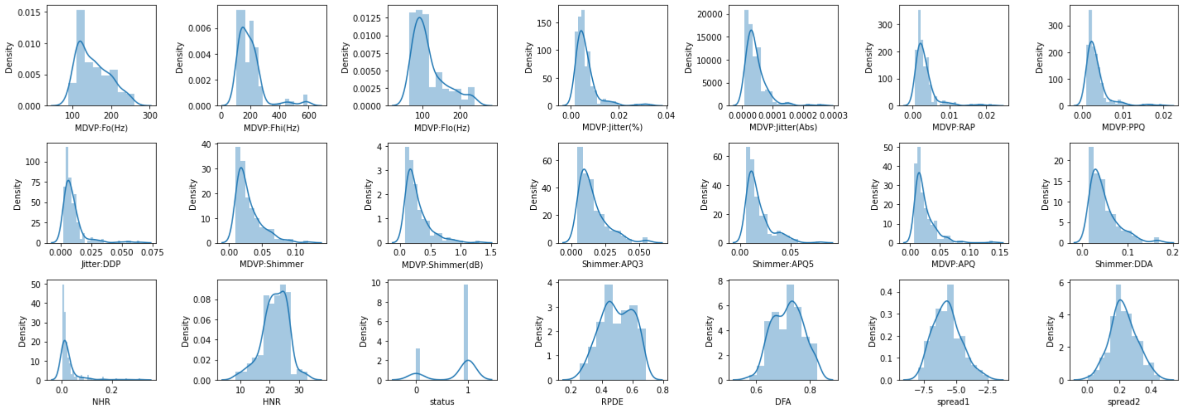
```
plt.figure(figsize=(12,7))
sns.scatterplot(x="status",y="RPDE",data=parkinsons_data);
plt.show()
```



Subplot

In [34]:

```
warnings.filterwarnings('ignore')
rows = 3
cols = 7
fig,ax = plt.subplots(nrows=rows,ncols=cols,figsize=(20,7))
col = parkinsons_data.columns
index = 1
for i in range(rows):
    for j in range(cols):
        sns.distplot(parkinsons_data[col[index]],ax=ax[i][j])
        index=index+1
plt.tight_layout()
```



In [ ]:

```

```

Correlation

[54]:

```
corr = parkinsons_data.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	-0.382027
MDVP:Fhi(Hz)	0.400985	1.000000	0.084951	0.102086	-0.029198
MDVP:Flo(Hz)	0.596546	0.084951	1.000000	-0.139919	-0.277815
MDVP:Jitter(%)	-0.118003	0.102086	-0.139919	1.000000	0.935714
MDVP:Jitter(Abs)	-0.382027	-0.029198	-0.277815	0.935714	1.000000
MDVP:RAP	-0.076194	0.097177	-0.100519	0.990276	0.92291
MDVP:PPQ	-0.112165	0.091126	-0.095828	0.974256	0.897778
Jitter:DDP	-0.076213	0.097150	-0.100488	0.990276	0.922913
MDVP:Shimmer	-0.098374	0.002281	-0.144543	0.769063	0.703322
MDVP:Shimmer(dB)	-0.073742	0.043465	-0.119089	0.804289	0.71660
Shimmer:APQ3	-0.094717	-0.003743	-0.150747	0.746625	0.697153
Shimmer:APQ5	-0.070682	-0.009997	-0.101095	0.725561	0.64896

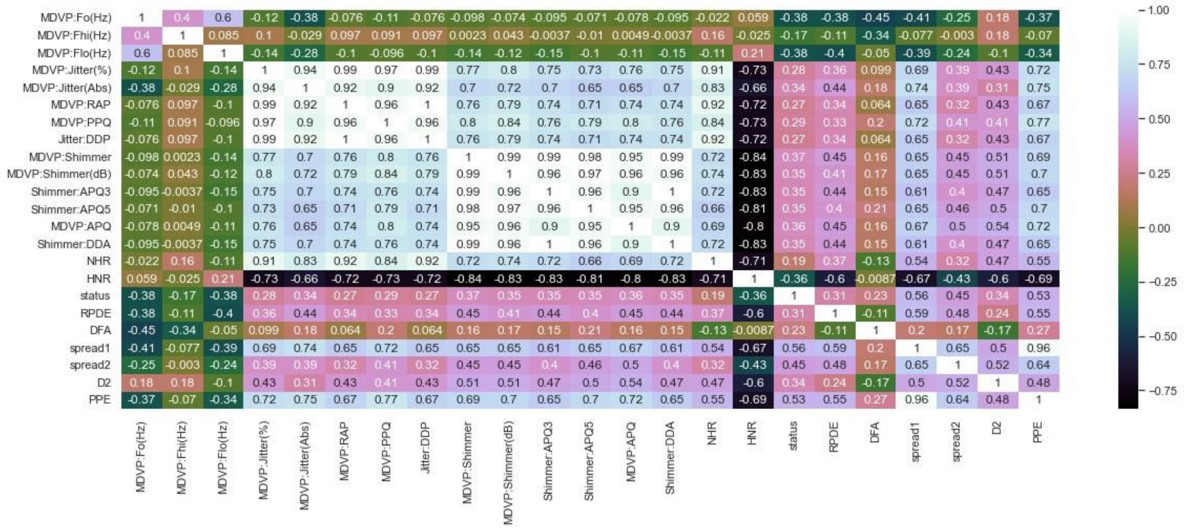
In

```
    MDVP:APQ  -0.077774   0.004937  -0.107293   0.758255   0.648793
Shimmer:DDA  -0.094732  -0.003733  -0.150737   0.746635   0.697170
        NHR  -0.021981   0.163766  -0.108670   0.906959   0.834972
        HNR   0.059144  -0.024893   0.210851  -0.728165  -0.656810
     status  -0.383535  -0.166136  -0.380200   0.278220   0.338653
       RPDE  -0.383894  -0.112404  -0.400143   0.360673   0.441839
        DFA  -0.446013  -0.343097  -0.050406   0.098572   0.175036
    spread1  -0.413738  -0.076658  -0.394857   0.693577   0.735779
    spread2  -0.249450  -0.002954  -0.243829   0.385123   0.388543
         D2   0.177980   0.176323  -0.100629   0.433434   0.310694
        PPE  -0.372356  -0.069543  -0.340071   0.721543   0.748162
23 rows × 23 columns
```

HeatMap

[56]:

```
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 20,7
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,cmap='cubehelix',annot=T
plt.show()
```



In [ ]:

In

```
A
```

[68]:

```
log_reg = LogisticRegression().fit(X_train,Y_train)

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

#predict on test
test_preds = log_reg.predict(X_test)

#Confusion matrix
print("confusion_matrix train is:\n",confusion_matrix(Y_train, train_preds))
print("confusion_matrix test is:\n",confusion_matrix(Y_test, test_preds))
print('\nClassification Report Train is')
print(classification_report(Y_train, train_preds))
print('\nClassification Report Test is')
print(classification_report(Y_test, test_preds))
```

```
confusion_matrix train is:
 [[ 24  16]
 [  4 112]] confusion_matrix
test is:
 [[ 5  3]
 [ 3 28]]

Classification Report Train is              precision
 recall  f1-score   support

           0       0.86      0.60      0.71        40
       1       0.88      0.97      0.92       116

    accuracy                           0.87       156
macro avg       0.87       0.78       0.81        156
weighted avg       0.87      0.87      0.86       156


Classification Report Test is              precision
 recall  f1-score   support

           0       0.62      0.62      0.62         8
       1       0.90      0.90      0.90        31

    accuracy                           0.85        39
macro avg       0.76       0.76       0.76         39
weighted avg       0.85      0.85      0.85        39
```

[69]:

```
RF = RandomForestClassifier().fit(X_train,Y_train)

#predict on train
train_preds2 = RF.predict(X_train)

#predict on test
test_preds2 = RF.predict(X_test)

#Confusion matrix
```

```
In
print("confusion_matrix train is:\n",confusion_matrix(Y_train, train_preds2))
print("confusion_matrix test is:\n",confusion_matrix(Y_test, test_preds2))
print('\nClassification Report Train is')
print(classification_report(Y_train,train_preds2)) print('\nClassification
Report Test is') print(classification_report(Y_test,test_preds2))
print((Y_test!=test_preds2).sum(),'/',((Y_test==test_preds2).sum()+(Y_test!=test_preds2).s
u
```

```
confusion_matrix train is:
 [[ 40   0]
 [  0 116]] confusion_matrix
test is:
 [[ 6  2]
 [ 1 30]]

Classification Report Train is              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


Classification Report Test is              precision
 recall  f1-score   support

           0       0.86      0.75      0.80         8
 1       0.94      0.97      0.95        31

    accuracy                           0.92        39
macro avg       0.90      0.86      0.88        39 weighted
avg       0.92      0.92      0.92        39

3 / 39
```

[61]:

```
parkinsons_data.drop(['name'],axis=1,inplace=True)
X = parkinsons_data.drop(labels=['status'],axis=1)
Y = parkinsons_data['status']
X.head()
```

Out[61]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MD
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

In [63]:

```
display(Y.head())
```

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

In [64]:

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2,random_state=40)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

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

In [67]:

```
print('KappaScore is:',metrics.cohen_kappa_score(Y_test,test_preds2))
```

```
KappaScore is: 0.587737843551797
```

[71]:

```
dparkinsons_data = pd.DataFrame(data=[test_preds2,Y_test])
display(dparkinsons_data)
```

	0	1	2	3	4	5	6	7	8	9	...	29	30	31	32	33	34	35	36	37	38
0	1	1	1	0	1	1	1	1	1	1	...	1	0	0	0	1	1	1	1	1	1
1	1	1 1	1 0	0 0	1 0	0 1	1 1	1 1	1 1	0 1	... 1										

2 rows × 39 columns

In [72]:

```
display(dparkinsons_data.T)
```

	0	1
0	1	1
1	1	1
2	1	1
3	0	0
4	1	1
5	1	0
6	1	1
7	1	1
8	1	1
9	1	0
10	1	1
11	1	1
12	0	0
13	1	1
14	1	1
15	1	1
16	1	1
17	1	1
18	1	1
19	1	1
20	1	1
21	0	1
22	1	1
23	1	1
24	0	0
25	1	1
26	1	1
27	1	1
28	1	1
29	1	1
30	0	0
31	0	0
32	0	0
33	1	1
34	1	1

In

```
35   1   1
```

	0	1
36	1	1
37	1	1
38	1	1

In [77]:

```python
from sklearn.tree import DecisionTreeClassifier

#fit 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)

#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))
print('wrong predictions out of total')
print('-'*50)
print('\nClassification Report Train is')
print(classification_report(Y_train,train_preds3))
print('\nClassification Report Test is')
print(classification_report(Y_test,test_preds3))
```

```
model accuracy on train is: 1.0
model accuracy on test is: 1.0
--------------------------------------------------
confusion_matrix train is:
  [[ 40   0]
 [  0 116]]
confusion_matrix test is: 
 [[ 8  0]
 [ 0 31]]
wrong predictions out of total
--------------------------------------------------

Classification Report Train is
              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


Classification Report Test is
              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
```

In [78]:

```
# Wrong Predictions made.
print((Y_test!=test_preds3).sum(),'/',((Y_test==test_preds3).sum()+(Y_test!=test_preds3).su
print('-'*50)

# Kappa Score
print('KappaScore is:',metrics.cohen_kappa_score(Y_test,test_preds3))
```

```
0 / 39
-------------------------------------------------- KappaScore
is: 1.0
```

In [79]:

```
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)

#Confusion matrix
print("confusion_matrix train is:\n",confusion_matrix(Y_train, train_preds4))
print("confusion_matrix test is:\n",confusion_matrix(Y_test, test_preds4))
print('wrong predictions out of total')
print('-'*50)
print('\nClassification Report Train is')
print(classification_report(Y_train, train_preds4))
print('\nClassification Report Test is')
print(classification_report(Y_test, test_preds4))
```

```
model accuracy on train is: 0.7307692307692307
model accuracy on test is: 0.6923076923076923
--------------------------------------------------
confusion_matrix train is:
 [[38  2]
 [40 76]]
confusion_matrix test is:
 [[ 8  0]
 [12 19]]
wrong predictions out of total
--------------------------------------------------

Classification Report Train is
              precision    recall  f1-score   support

           0       0.49      0.95      0.64        40
           1       0.97      0.66      0.78       116

    accuracy                           0.73       156
   macro avg       0.73      0.80      0.71       156
weighted avg       0.85      0.73      0.75       156


Classification Report Test is
              precision    recall  f1-score   support

           0       0.40      1.00      0.57         8
           1       1.00      0.61      0.76        31

    accuracy                           0.69        39
   macro avg       0.70      0.81      0.67        39
weighted avg       0.88      0.69      0.72        39 In
```

[80]:

```
# Wrong Predictions made.
print((Y_test!=test_preds4).sum(),'/',((Y_test==test_preds4).sum()+(Y_test!=test_preds4).su
print('-'*50)

# Kappa Score
print('KappaScore is:',metrics.cohen_kappa_score(Y_test,test_preds4))
```

```
12 / 39
-------------------------------------------------- KappaScore
is: 0.3937823834196892
```

In [83]:

```
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)

#Confusion matrix
print("confusion_matrix train is:\n",confusion_matrix(Y_train, train_preds5))
print("confusion_matrix test is:\n",confusion_matrix(Y_test, test_preds5))
print('wrong predictions out of total')
print('-'*50)
print('\nClassification Report Train is')
print(classification_report(Y_train,train_preds5))
print('\nClassification Report Test is')
print(classification_report(Y_test,test_preds5))
```

```
model accuracy on train is: 0.9102564102564102
model accuracy on test is: 0.8461538461538461
--------------------------------------------------
confusion_matrix train is:
 [[ 30  10]
 [  4 112]]
confusion_matrix test is:
 [[ 4  4]
 [ 2 29]]
wrong predictions out of total
--------------------------------------------------

Classification Report Train is
              precision    recall  f1-score   support

           0       0.88      0.75      0.81        40
           1       0.92      0.97      0.94       116

    accuracy                           0.91       156
   macro avg       0.90      0.86      0.88       156
weighted avg       0.91      0.91      0.91       156


Classification Report Test is
              precision    recall  f1-score   support

           0       0.67      0.50      0.57         8
           1       0.88      0.94      0.91        31

    accuracy                           0.85        39
   macro avg       0.77      0.72      0.74        39
```

```
weighted avg       0.84      0.85      0.84        39
```

In [84]:

```
# Wrong Predictions made.
print((Y_test!=test_preds5).sum(),'/',((Y_test==test_preds5).sum()+(Y_test!=test_preds5).su
print('-'*50)

# Kappa Score
print('KappaScore is:',metrics.cohen_kappa_score(Y_test,test_preds5))
```

```
6 / 39
-------------------------------------------------- KappaScore
is: 0.48
```

In [86]:

```
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_preds6))
print('-'*50)

#Confusion matrix
print("confusion_matrix train is:\n",confusion_matrix(Y_train, train_preds6))
print("confusion_matrix test is:\n",confusion_matrix(Y_test, test_preds6))
print('wrong predictions out of total')
print('-'*50)
print("recall",metrics.recall_score(Y_test, test_preds6))
print('-'*50)
print('\nClassification Report Train is')
print(classification_report(Y_train,train_preds6))
print('\nClassification Report Test is')
print(classification_report(Y_test,test_preds6))
```

```
model accuracy on train is: 0.8782051282051282
model accuracy on test is: 0.8974358974358975
--------------------------------------------------
confusion_matrix train is:
 [[ 23  17]
 [  2 114]]
confusion_matrix test is:
 [[ 5  3]
 [ 1 30]]
wrong predictions out of total
--------------------------------------------------
recall 0.967741935483871
--------------------------------------------------

Classification Report Train is
              precision    recall  f1-score   support

           0       0.92      0.57      0.71        40
           1       0.87      0.98      0.92       116

    accuracy                           0.88       156
   macro avg       0.90      0.78      0.82       156
weighted avg       0.88      0.88      0.87       156


Classification Report Test is
              precision    recall  f1-score   support
           0       0.83      0.62      0.71         8
 1       0.91      0.97      0.94        31

    accuracy                           0.90        39

macro avg       0.87      0.80      0.83        39

weighted avg       0.89      0.90      0.89        39
```

In [87]:

```
# Wrong Predictions made.
print((Y_test!=test_preds6).sum(),'/',((Y_test==test_preds6).sum()+(Y_test!=test_preds6).su
print('-'*50)

# Kappa Score
print('KappaScore is:',metrics.cohen_kappa_score(Y_test,test_preds6))
```

```
4 / 39
--------------------------------------------------
KappaScore is: 0.6533333333333333
```

In [88]:

```
import pickle

# Saving model to disk
pickle.dump(SVM,open('deploy_SVM.pkl','wb'))

# Open the Pickle File
model = pickle.load(open('deploy_SVM.pkl','rb'))

# Prediction
model.predict(X_train)
```

Out[88]:

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

In [ ]: