Importing Libraries

In [1]:

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

from sklearn.preprocessing import MinMaxScaler
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings('ignore')
```

Load Dataset

In [2]:

```
df=pd.read_csv("parkinsons.data")
df
```

Out[2]:

	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
194	phon_R01_S50_6	214.289	260.277	77.973	0.00567	0.00

195 rows × 24 columns

Renaming columns

In [3]:

```
df.rename(columns={'MDVP:Fo(Hz)':'avg_fre', 'MDVP:Fhi(Hz)':'max_fre', 'MDVP:Flo(Hz)':'min_
       'MDVP:Jitter(Abs)':'var_fre2', 'MDVP:RAP':'var_fre3', 'MDVP:PPQ':'var_fre4', 'Jitter
       'MDVP:Shimmer':'var_amp1', 'MDVP:Shimmer(dB)':'var_amp2', 'Shimmer:APQ3':'var_amp3',
       'MDVP:APQ':'var_amp5', 'Shimmer:DDA':'var_amp6'}),inplace=True)
df
```

Out[3]:

	name	avg_fre	max_fre	min_fre	var_fre1	var_fre2	var_fre3	var_fre4	var_fre5
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.0110
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.0139
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.0163
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.0150
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.0196
...	...	...	...	...	...	...	...	...	...
190	phon_R01_S50_2	174.188	230.978	94.261	0.00459	0.00003	0.00263	0.00259	0.0079
191	phon_R01_S50_3	209.516	253.017	89.488	0.00564	0.00003	0.00331	0.00292	0.0099
192	phon_R01_S50_4	174.688	240.005	74.287	0.01360	0.00008	0.00624	0.00564	0.0187
193	phon_R01_S50_5	198.764	396.961	74.904	0.00740	0.00004	0.00370	0.00390	0.0110
194	phon_R01_S50_6	214.289	260.277	77.973	0.00567	0.00003	0.00295	0.00317	0.0088

195 rows × 24 columns

In [4]:

```
df.columns
```

Out[4]:

```
Index(['name', 'avg_fre', 'max_fre', 'min_fre', 'var_fre1', 'var_fre2',
       'var_fre3', 'var_fre4', 'var_fre5', 'var_amp1', 'var_amp2', 'var_amp
3',
       'var_amp4', 'var_amp5', 'var_amp6', 'NHR', 'HNR', 'status', 'RPDE',
       'DFA', 'spread1', 'spread2', 'D2', 'PPE'],
      dtype='object')
```

Dimensions of Dataset

In [5]:

```
df.shape
```

Out[5]:

```
(195, 24)
```

Peak at the Data

In [6]:

```
df.head()
```

Out[6]:

	name	avg_fre	max_fre	min_fre	var_fre1	var_fre2	var_fre3	var_fre4	var_fre5
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966

5 rows × 24 columns

Statistical Summary

In [7]:

```
df.describe()
```

Out[7]:

	avg_fre	max_fre	min_fre	var_fre1	var_fre2	var_fre3	var_fre4	
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	19
mean	154.228641	197.104918	116.324631	0.006220	0.000044	0.003306	0.003446	
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	0.002759	
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680	0.000920	
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660	0.001860	
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500	0.002690	
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835	0.003955	
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440	0.019580	

8 rows × 23 columns

Information of the dataset

In [8]:

```
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   avg_fre   195 non-null    float64
 2   max_fre   195 non-null    float64
 3   min_fre   195 non-null    float64
 4   var_fre1  195 non-null    float64
 5   var_fre2  195 non-null    float64
 6   var_fre3  195 non-null    float64
 7   var_fre4  195 non-null    float64
 8   var_fre5  195 non-null    float64
 9   var_amp1  195 non-null    float64
 10  var_amp2  195 non-null    float64
 11  var_amp3  195 non-null    float64
 12  var_amp4  195 non-null    float64
 13  var_amp5  195 non-null    float64
 14  var_amp6  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
```

Duplicate Entries

In [9]:

```
df.duplicated().sum()
```

Out[9]:

```
0
```

unwanted columns

In [10]:

```
df.drop(columns="name",axis=1,inplace=True)
df
```

Out[10]:

	avg_fre	max_fre	min_fre	var_fre1	var_fre2	var_fre3	var_fre4	var_fre5	var_amp1	var_
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	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	
4	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	
...	...	...	...	...	...	...	...	...	...	
190	174.188	230.978	94.261	0.00459	0.00003	0.00263	0.00259	0.00790	0.04087	
191	209.516	253.017	89.488	0.00564	0.00003	0.00331	0.00292	0.00994	0.02751	
192	174.688	240.005	74.287	0.01360	0.00008	0.00624	0.00564	0.01873	0.02308	
193	198.764	396.961	74.904	0.00740	0.00004	0.00370	0.00390	0.01109	0.02296	
194	214.289	260.277	77.973	0.00567	0.00003	0.00295	0.00317	0.00885	0.01884	

195 rows × 23 columns

Missing values

In [11]:

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

Out[11]:

```
avg_fre     0
max_fre     0
min_fre     0
var_fre1    0
var_fre2    0
var_fre3    0
var_fre4    0
var_fre5    0
var_amp1    0
var_amp2    0
var_amp3    0
var_amp4    0
var_amp5    0
var_amp6    0
NHR         0
HNR         0
status      0
RPDE        0
DFA         0
spread1     0
spread2     0
D2          0
PPE         0
dtype: int64
```

In [12]:

```
df.notnull()
```

Out[12]:

	avg_fre	max_fre	min_fre	var_fre1	var_fre2	var_fre3	var_fre4	var_fre5	var_amp1	var_
0	True	True	True	True	True	True	True	True	True	
1	True	True	True	True	True	True	True	True	True	
2	True	True	True	True	True	True	True	True	True	
3	True	True	True	True	True	True	True	True	True	
4	True	True	True	True	True	True	True	True	True	
...	...	...	...	...	...	...	...	...	...	
190	True	True	True	True	True	True	True	True	True	
191	True	True	True	True	True	True	True	True	True	
192	True	True	True	True	True	True	True	True	True	
193	True	True	True	True	True	True	True	True	True	
194	True	True	True	True	True	True	True	True	True	

195 rows × 23 columns

Outliers

In [13]:

```
df.skew()
```

Out[13]:

```
avg_fre     0.591737
max_fre     2.542146
min_fre     1.217350
var_fre1    3.084946
var_fre2    2.649071
var_fre3    3.360708
var_fre4    3.073892
var_fre5    3.362058
var_amp1    1.666480
var_amp2    1.999389
var_amp3    1.580576
var_amp4    1.798697
var_amp5    2.618047
var_amp6    1.580618
NHR         4.220709
HNR        -0.514317
status     -1.187727
RPDE       -0.143402
DFA        -0.033214
spread1     0.432139
spread2     0.144430
D2          0.430384
PPE         0.797491
dtype: float64
```

Determining Dependend & Independent Variables

In [14]:

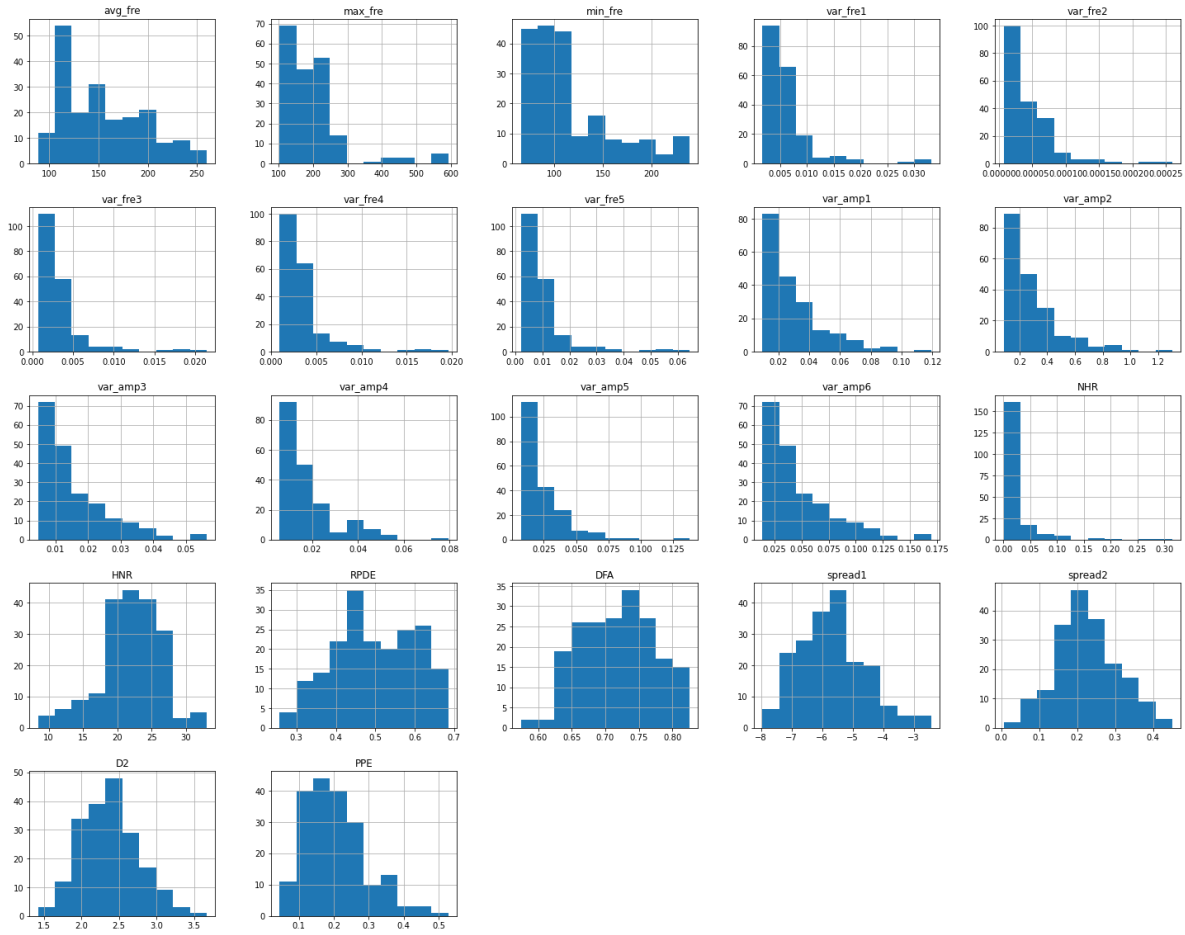
```
# get features and labels

x=df.loc[:,df.columns!='status'].values[:,1:]
x1=df.loc[:,df.columns!='status']
y=df.loc[:,'status'].values
y1=df.loc[:,'status']
```

Analyzing Features

In [15]:

```
x1.hist(figsize=(25,20))
plt.show()
```



In [16]:

```
df=df[df.max_fre<=300]
df=df[df.var_fre1<=0.02]
df=df[df.var_fre2<=0.0001]
df=df[df.var_fre3<=0.01]
df=df[df.var_fre4<=0.01]
df=df[df.var_fre5<=0.02]
df=df[df.var_amp1<=0.10]
df=df[df.var_amp2<=1.0]
df=df[df.var_amp3<=0.04]
df=df[df.var_amp4<=0.050]
df=df[df.var_amp5<=0.075]
df=df[df.var_amp6<=0.125]
df=df[df.NHR<=0.15]
```

In [17]:

```
df.skew()
```

Out[17]:

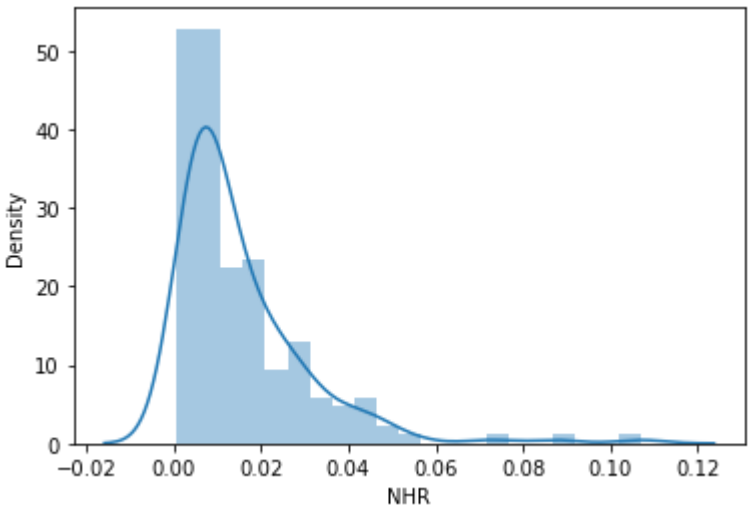
```
avg_fre     0.608391
max_fre     0.290164
min_fre     1.247241
var_fre1    0.843153
var_fre2    0.756592
var_fre3    0.811867
var_fre4    1.142506
var_fre5    0.811544
var_amp1    1.077428
var_amp2    1.138932
var_amp3    1.128533
var_amp4    1.376069
var_amp5    1.096979
var_amp6    1.128416
NHR         2.635106
HNR        -0.035596
status     -1.057890
RPDE       -0.066659
DFA        -0.132660
spread1     0.283933
spread2     0.158902
D2          0.485240
PPE         0.535763
dtype: float64
```

In [18]:

```
sns.distplot(df['NHR'])
```

Out[18]:

```
<AxesSubplot:xlabel='NHR', ylabel='Density'>
```



In [19]:

```
df=df[df.NHR<=0.06]
df.skew()
```

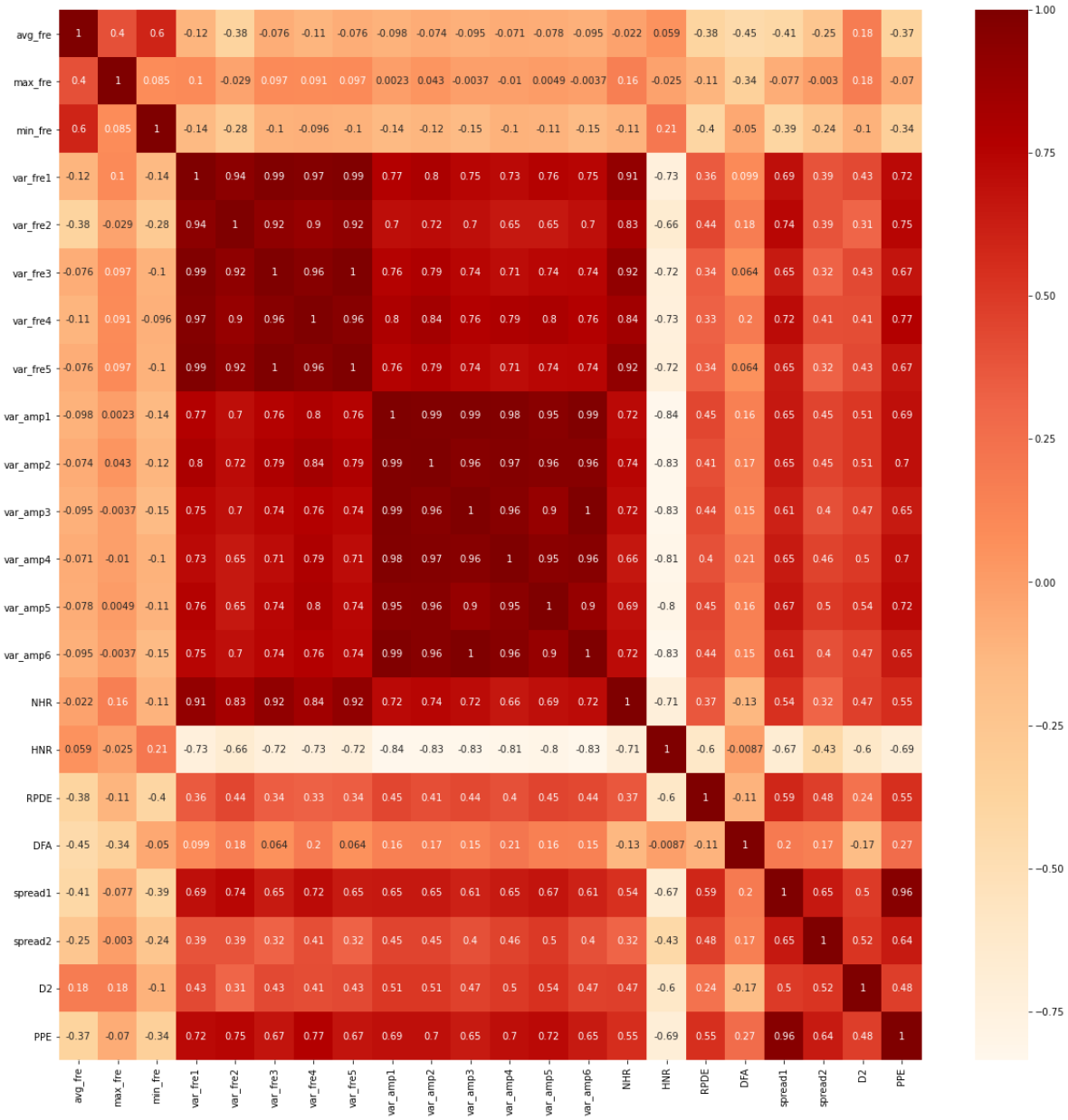
Out[19]:

```
avg_fre     0.629564
max_fre     0.328258
min_fre     1.245583
var_fre1    0.699469
var_fre2    0.769365
var_fre3    0.813203
var_fre4    1.212263
var_fre5    0.812495
var_amp1    1.063387
var_amp2    1.136743
var_amp3    1.116058
var_amp4    1.381370
var_amp5    1.098219
var_amp6    1.115979
NHR         1.327245
HNR         0.174386
status     -1.064996
RPDE       -0.061493
DFA        -0.133070
spread1     0.298066
spread2     0.123992
D2          0.194425
PPE         0.553609
dtype: float64
```

Correlation Matrix

In [20]:

```
correl=x1.corr()
plt.figure(figsize=(20,20))
sns.heatmap(correl,annot=True,cmap='OrRd')
plt.show()
```



In [21]:

```
#Scale the features to between -1 and 1
scaler=MinMaxScaler((-1,1))
x1=scaler.fit_transform(x)
y1=y
```

In [22]:

```
#Split the dataset

xtrain,xtest,ytrain,ytest=train_test_split(x1, y1, test_size=0.2)
```

In [23]:

```
# Train the model

model=XGBClassifier()
model.fit(xtrain,ytrain)
pre=model.predict(xtest)
```

```
[17:12:19] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behav
ior.
```

In [24]:

```
print(accuracy_score(ytest,pre)*100)
```

```
97.43589743589743
```

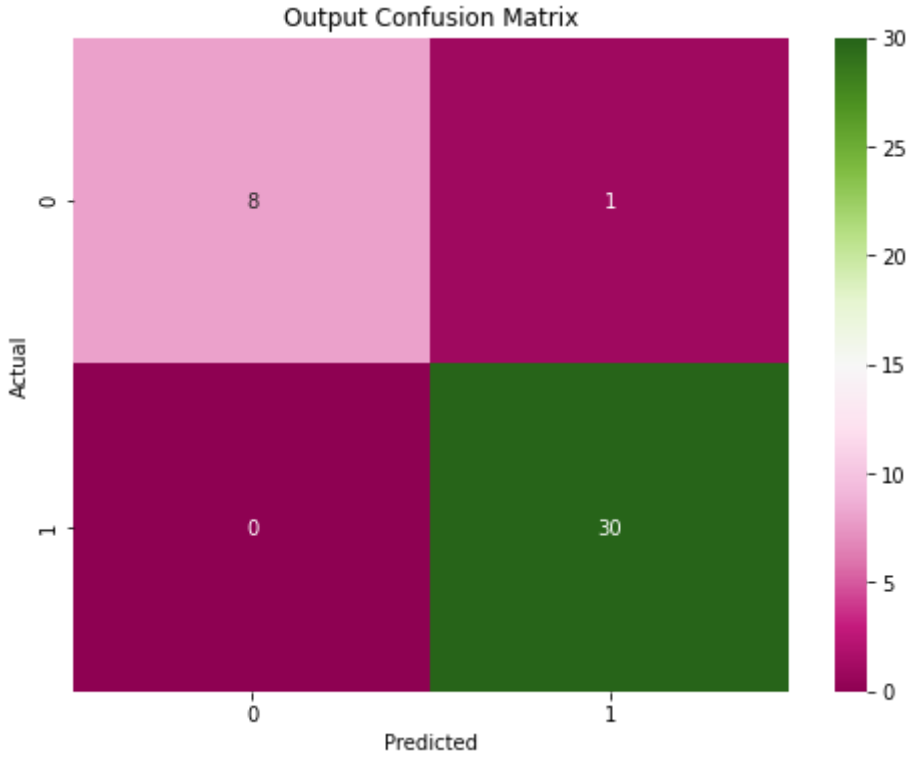
Implementing Confusion Matrix

In [25]:

```
from sklearn.metrics import confusion_matrix
pm=confusion_matrix(ytest,pre)
plt.figure(figsize=(8,6))
fg=sns.heatmap(pm,annot=True,cmap="PiYG")
figure=fg.get_figure()
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title("Output Confusion Matrix")
```

Out[25]:

```
Text(0.5, 1.0, 'Output Confusion Matrix')
```



Output Display

In [26]:

```
pd.DataFrame({'actual':ytest,'predict':pre})
```

Out[26]:

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

	actual	predict
34	1	1
35	1	1
36	0	0
37	1	1
38	0	0

In [ ]: