# Group Number 1

Members :

- Sunil Jacob
- Sandeep
- Hari Ganapathy
- Sasirekha Sathasivam

### Parkinsons Disease Data Set

### Dataset information:

This dataset is composed of a range of biomedical voice measurements from 31 people, 23 with Parkinson's disease (PD). Each column in the table is a particular voice measure, and each row corresponds one of 195 voice recording from these individuals ("name" column). The main aim of the data is to discriminate healthy people from those with PD, according to "status" column which is set to 0 for healthy and 1 for PD.

The data is in ASCII CSV format. The rows of the CSV file contain an instance corresponding to one voice recording. There are around six recordings per patient, the name of the patient is identified in the first column.

**Attribute Information:**

Matrix column entries (attributes): * name - ASCII subject name and recording number * MDVP:Fo(Hz) - Average vocal fundamental frequency * MDVP:Fhi(Hz) - Maximum vocal fundamental frequency * MDVP:Flo(Hz) - Minimum vocal fundamental frequency * MDVP:Jitter(%),MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP - Several measures of variation in fundamental frequency * MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA - Several measures of variation in amplitude * NHR,HNR - Two measures of ratio of noise to tonal components in the voice * status - Health status of the subject (one) - Parkinson's, (zero) - healthy * RPDE,D2 - Two nonlinear dynamical complexity measures * DFA - Signal fractal scaling exponent * spread1,spread2,PPE - Three nonlinear measures of fundamental frequency variation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB as gnb
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from IPython.display import Image
from sklearn import tree
from os import system
```

```
data = pd.read_csv('parkinsons.txt')
```

```
data.head()
```

| | name | MDVP:Fo(Hz) | MDVP:Fhi(Hz) | MDVP:Flo(Hz) | MDVP:Jitter(%) | MDVP:Jitter(Abs) | MDVP:RAP | MDVP:PPQ | Jitter:DDP | MDVP:Shim |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | phon_R01_S01_1 | 119.992 | 157.302 | 74.997 | 0.00784 | 0.00007 | 0.00370 | 0.00554 | 0.01109 | 0.04374 |
| **1** | phon_R01_S01_2 | 122.400 | 148.650 | 113.819 | 0.00968 | 0.00008 | 0.00465 | 0.00696 | 0.01394 | 0.06134 |
| **2** | phon_R01_S01_3 | 116.682 | 131.111 | 111.555 | 0.01050 | 0.00009 | 0.00544 | 0.00781 | 0.01633 | 0.05233 |
| **3** | phon_R01_S01_4 | 116.676 | 137.871 | 111.366 | 0.00997 | 0.00009 | 0.00502 | 0.00698 | 0.01505 | 0.05492 |
| **4** | phon_R01_S01_5 | 116.014 | 141.781 | 110.655 | 0.01284 | 0.00011 | 0.00655 | 0.00908 | 0.01966 | 0.06425 |

5 rows × 24 columns

```
data.shape
```

```
(195, 24)
```

```
#moving status column to the last
df1=data.pop('status')
data['status'] = df1
```

```
data.describe().transpose()
```

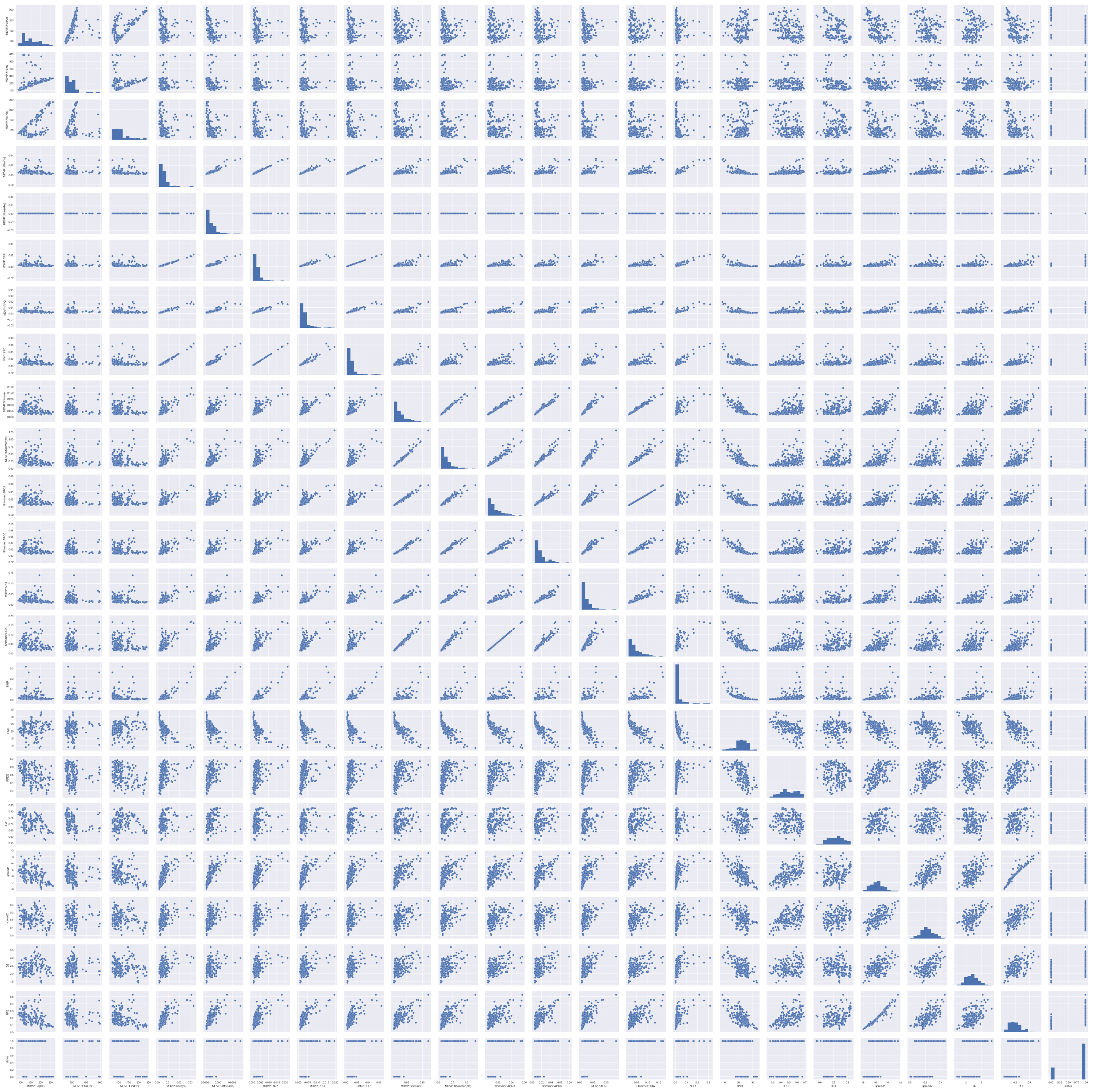
| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **MDVP:Fo(Hz)** | 195.0 | 154.228641 | 41.390065 | 88.333000 | 117.572000 | 148.790000 | 182.769000 | 260.105000 |
| **MDVP:Fhi(Hz)** | 195.0 | 197.104918 | 91.491548 | 102.145000 | 134.862500 | 175.829000 | 224.205500 | 592.030000 |
| **MDVP:Flo(Hz)** | 195.0 | 116.324631 | 43.521413 | 65.476000 | 84.291000 | 104.315000 | 140.018500 | 239.170000 |
| **MDVP:Jitter(%)** | 195.0 | 0.006220 | 0.004848 | 0.001680 | 0.003460 | 0.004940 | 0.007365 | 0.033160 |
| **MDVP:Jitter(Abs)** | 195.0 | 0.000044 | 0.000035 | 0.000007 | 0.000020 | 0.000030 | 0.000060 | 0.000260 |
| **MDVP:RAP** | 195.0 | 0.003306 | 0.002968 | 0.000680 | 0.001660 | 0.002500 | 0.003835 | 0.021440 |
| **MDVP:PPQ** | 195.0 | 0.003446 | 0.002759 | 0.000920 | 0.001860 | 0.002690 | 0.003955 | 0.019580 |
| **Jitter:DDP** | 195.0 | 0.009920 | 0.008903 | 0.002040 | 0.004985 | 0.007490 | 0.011505 | 0.064330 |
| **MDVP:Shimmer** | 195.0 | 0.029709 | 0.018857 | 0.009540 | 0.016505 | 0.022970 | 0.037885 | 0.119080 |
| **MDVP:Shimmer(dB)** | 195.0 | 0.282251 | 0.194877 | 0.085000 | 0.148500 | 0.221000 | 0.350000 | 1.302000 |
| **Shimmer:APQ3** | 195.0 | 0.015664 | 0.010153 | 0.004550 | 0.008245 | 0.012790 | 0.020265 | 0.056470 |
| **Shimmer:APQ5** | 195.0 | 0.017878 | 0.012024 | 0.005700 | 0.009580 | 0.013470 | 0.022380 | 0.079400 |
| **MDVP:APQ** | 195.0 | 0.024081 | 0.016947 | 0.007190 | 0.013080 | 0.018260 | 0.029400 | 0.137780 |
| **Shimmer:DDA** | 195.0 | 0.046993 | 0.030459 | 0.013640 | 0.024735 | 0.038360 | 0.060795 | 0.169420 |
| **NHR** | 195.0 | 0.024847 | 0.040418 | 0.000650 | 0.005925 | 0.011660 | 0.025640 | 0.314820 |
| **HNR** | 195.0 | 21.885974 | 4.425764 | 8.441000 | 19.198000 | 22.085000 | 25.075500 | 33.047000 |
| **RPDE** | 195.0 | 0.498536 | 0.103942 | 0.256570 | 0.421306 | 0.495954 | 0.587562 | 0.685151 |
| **DFA** | 195.0 | 0.718099 | 0.055336 | 0.574282 | 0.674758 | 0.722254 | 0.761881 | 0.825288 |
| **spread1** | 195.0 | -5.684397 | 1.090208 | -7.964984 | -6.450096 | -5.720868 | -5.046192 | -2.434031 |
| **spread2** | 195.0 | 0.226510 | 0.083406 | 0.006274 | 0.174351 | 0.218885 | 0.279234 | 0.450493 |
| **D2** | 195.0 | 2.381826 | 0.382799 | 1.423287 | 2.099125 | 2.361532 | 2.636456 | 3.671155 |
| **PPE** | 195.0 | 0.206552 | 0.090119 | 0.044539 | 0.137451 | 0.194052 | 0.252980 | 0.527367 |
| **status** | 195.0 | 0.753846 | 0.431878 | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |

```
# There are 147 people affected with PD .
data.status.value_counts()
```

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

```
sns.pairplot(data)
```

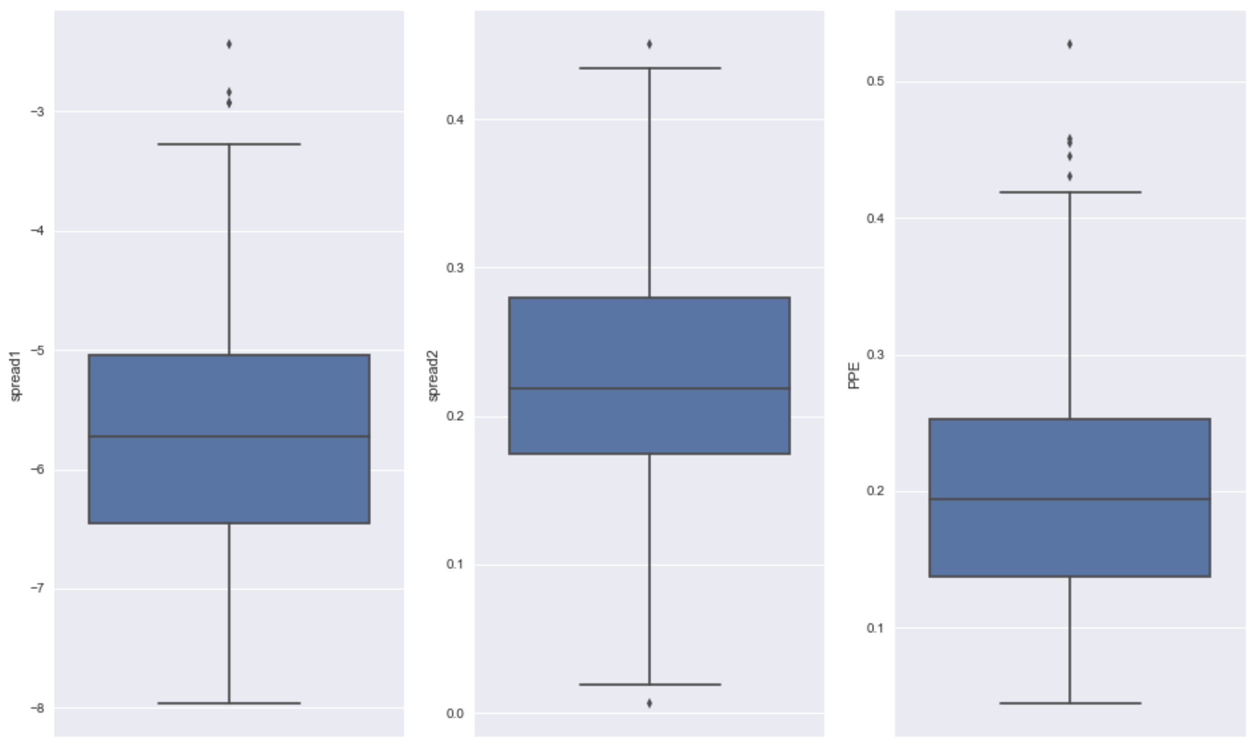
```
<seaborn.axisgrid.PairGrid at 0x117064278>
```



## Univariate Analysis

```
fig, ax = plt.subplots(1,3,figsize=(16,10))
sns.boxplot(x='spread1',data=data, ax=ax[0],orient='v')
sns.boxplot(x='spread2',data=data, ax=ax[1],orient='v')
sns.boxplot(x='PPE',data=data, ax=ax[2],orient='v')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x119816f60>
```



The above figure shows the box plot of the frequency variation. All the three variations have outliers.

Generally speaking, decision trees are able to handle outliers. It is very unlikely that decision tree will create a leaf to isolate them

```
# dropping name column as this column is not much significant
data = data.drop('name',axis=1)
```

```
type(data)
```

```
pandas.core.frame.DataFrame
```

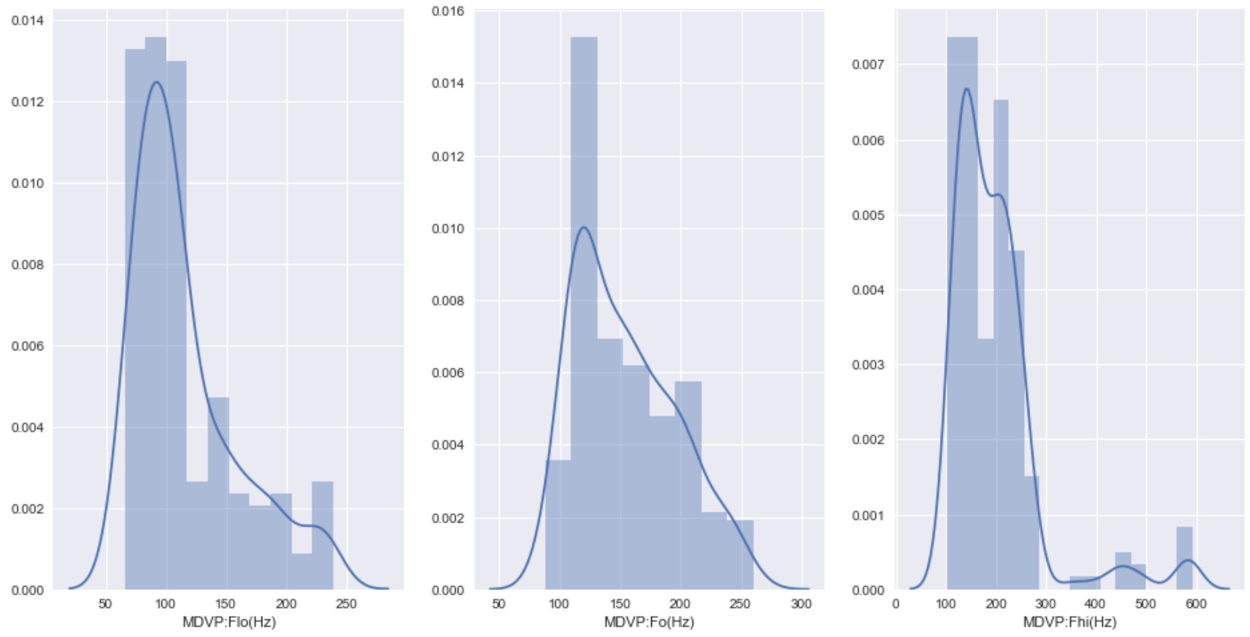
```
# Using z-score function in scipy to detect outliers
# from scipy import stats
# z = np.abs(stats.zscore(data))
# print(z)
```

```
# Defining a threshold to detect an outlier as we cannot infer any meaningful info from the above dataset
# threshold = 3
# print(np.where(z > 3))
```

```
# print(z[73][1])
# data = data[(z < 3).all(axis=1)]
```

```
fig, ax = plt.subplots(1,3,figsize=(16,8))
sns.distplot(data['MDVP:Flo(Hz)'],ax=ax[0])
sns.distplot(data['MDVP:Fo(Hz)'],ax=ax[1])
sns.distplot(data['MDVP:Fhi(Hz)'],ax=ax[2])
```

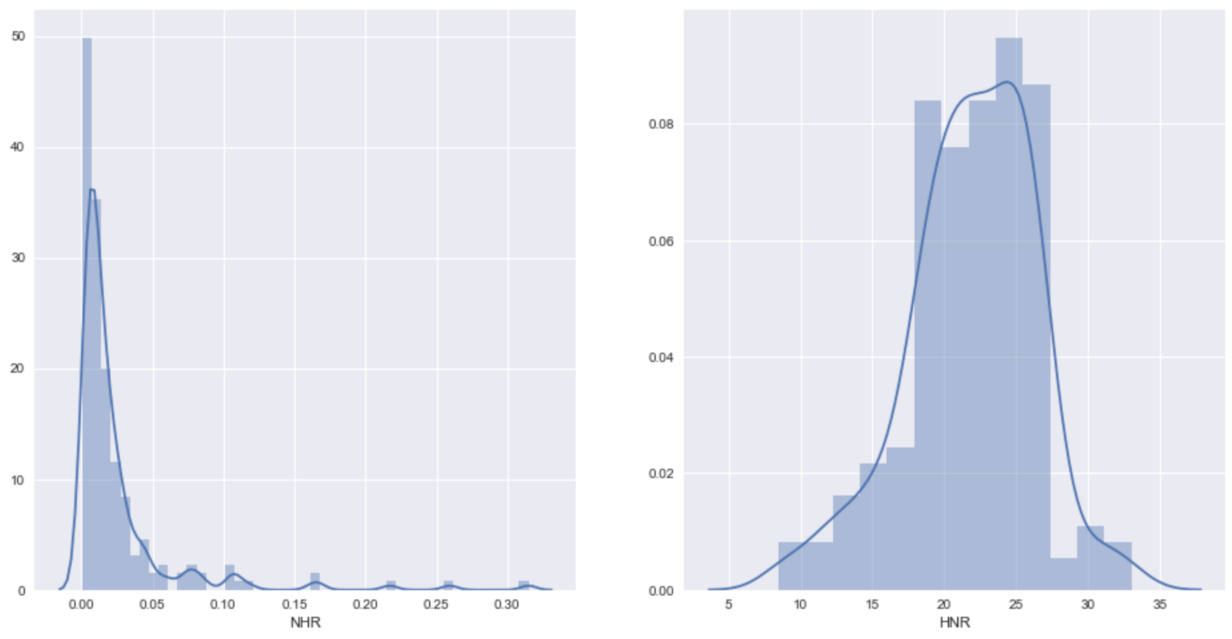
```
<matplotlib.axes._subplots.AxesSubplot at 0x119d37dd8>
```



The measures of vocal fundamental frequency are shown above. There is a positive skewness for `minimum vocal fundemental frequency` with more high values between 75Hz and 125Hhz. The `average vocal frequency` is almost normally distributed with more values ranging 115Hz and 130Hz. The `high vocal frequency` does not have any skewness, but some range of values are at the right most tail

```
fig, ax = plt.subplots(1,2,figsize=(16,8))
sns.distplot(data['NHR'],ax=ax[0])
sns.distplot(data['HNR'],ax=ax[1])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11a3fe908>
```

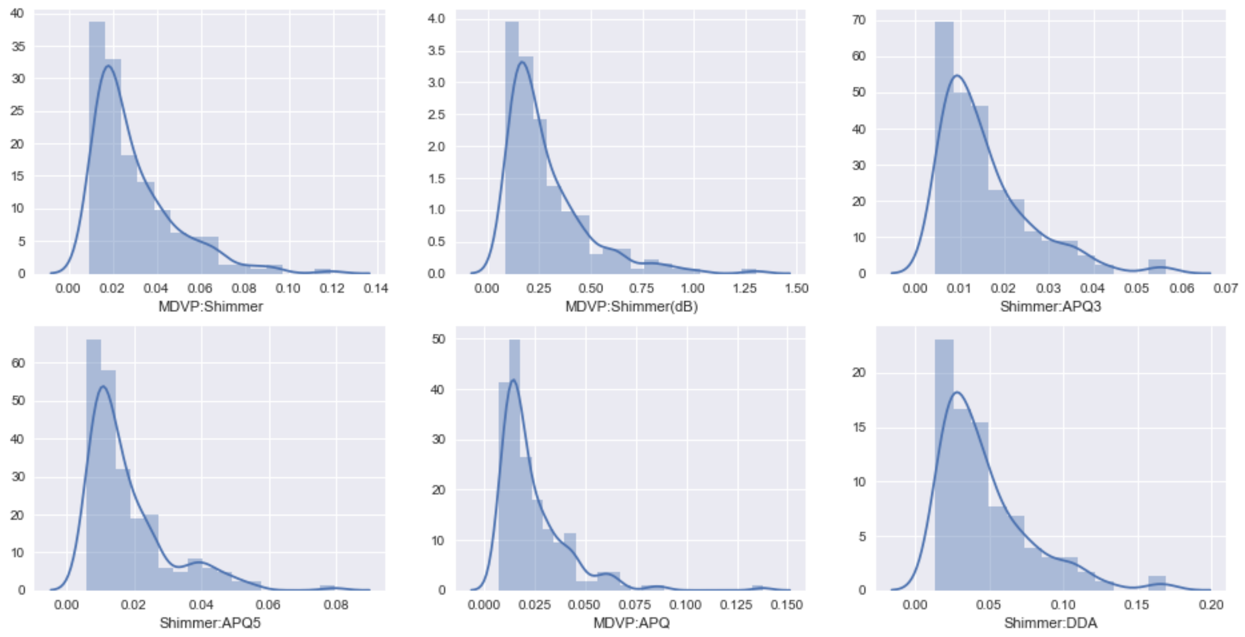


The measure of tonal component of frequency is shown above. The value `NHR` is right skewed for there are so many observations in the area, but they seem to be with very minimal values. The maximum number of observations is between 0 and 0.04.

The value `HNR` looks like normally distributed, but in a first look there seems to be a slight negative skewness

```
fig, ax = plt.subplots(2,3,figsize=(16,8))
sns.distplot(data['MDVP:Shimmer'],ax=ax[0,0])
sns.distplot(data['MDVP:Shimmer(dB)'],ax=ax[0,1])
sns.distplot(data['Shimmer:APQ3'],ax=ax[0,2])
sns.distplot(data['Shimmer:APQ5'],ax=ax[1,0])
sns.distplot(data['MDVP:APQ'],ax=ax[1,1])
sns.distplot(data['Shimmer:DDA'],ax=ax[1,2])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11a9a4be0>
```



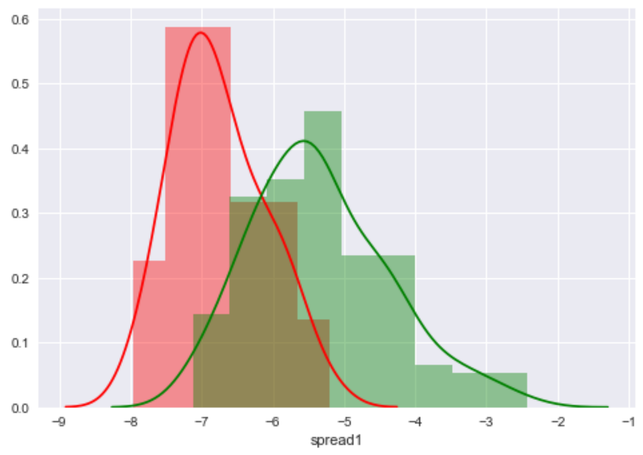
For all of the above graphs, we can observe that the measure of variation in amplitude is positively skewed

## Bi -Variate Analysis

Relation between target variable and independent variables

```
sns.distplot( data[data.status == 0]['spread1'], color = 'r')
sns.distplot( data[data.status == 1]['spread1'], color = 'g')
```

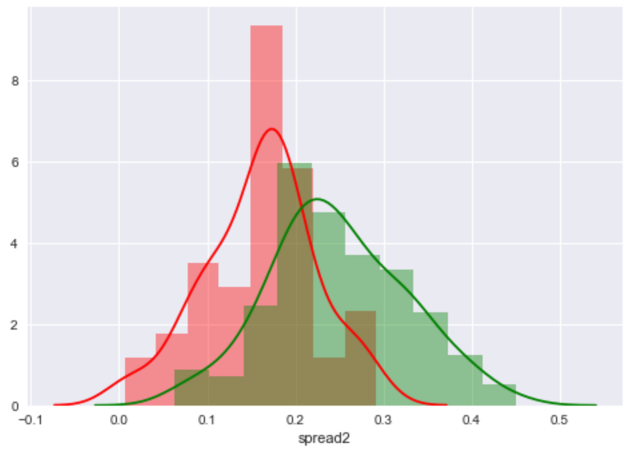
```
<matplotlib.axes._subplots.AxesSubplot at 0x11a4f07f0>
```



**spread1** is normally distributed between person who have PD and who is normal. People who have **spread1** between - 8.5 and -7.5 are more and they are normal. People whose **spread1** is between -6.5 and -5 are having PD

```
sns.distplot( data[data.status == 0]['spread2'], color = 'r')
sns.distplot( data[data.status == 1]['spread2'], color = 'g')
```

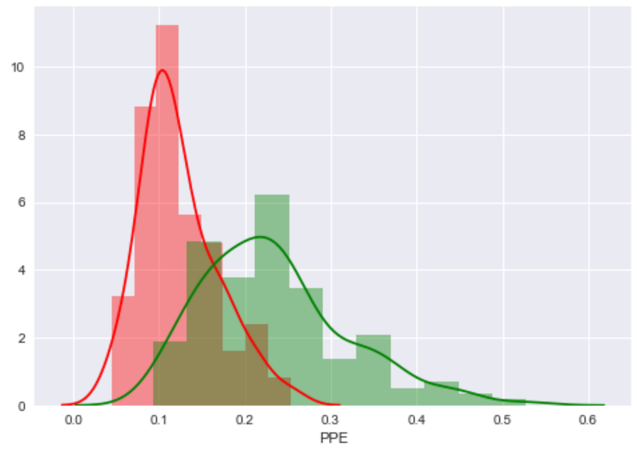
```
<matplotlib.axes._subplots.AxesSubplot at 0x11af8fe48>
```



**spread1** and **spread2** is normally distributed. between person who have PD and who is normal. People who have **spread1** between 0.15 and 0.175 are more and they are normal. People whose **spread1** is between 0.175 and 0.2 are having PD

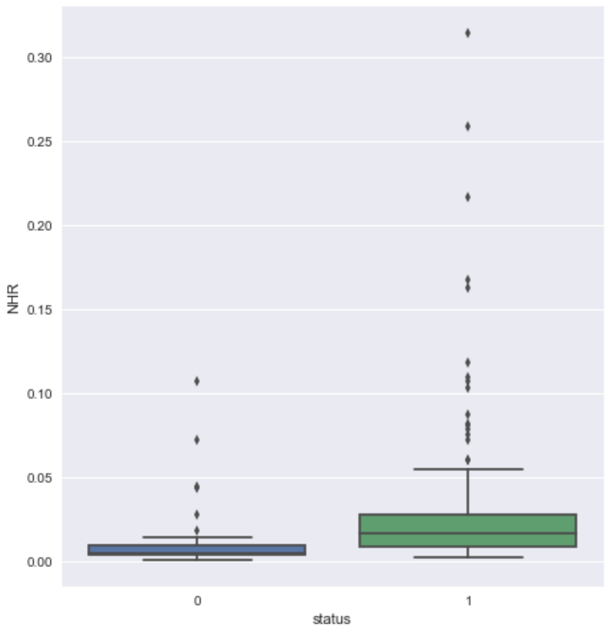
```
sns.distplot( data[data.status == 0]['PPE'], color = 'r')
sns.distplot( data[data.status == 1]['PPE'], color = 'g')
```

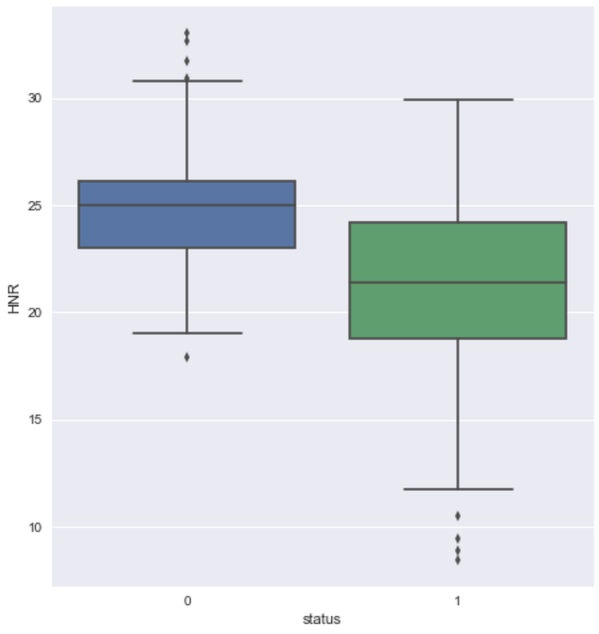
```
<matplotlib.axes._subplots.AxesSubplot at 0x11b0d7e80>
```



```
fig, ax = plt.subplots(1,2,figsize=(16,8))
sns.boxplot(x='status',y='NHR',data=data,ax=ax[0])
sns.boxplot(x='status',y='HNR',data=data,ax=ax[1])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11b403390>
```

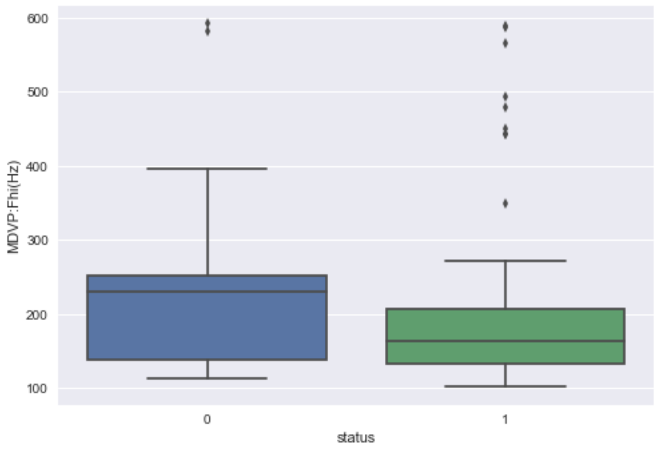




People who have PD(status equal to one) have higher levels of `Noise to Harmonic` ratio. Also, looking into the `HNR` ratio people who have PD have lower levels in the same.

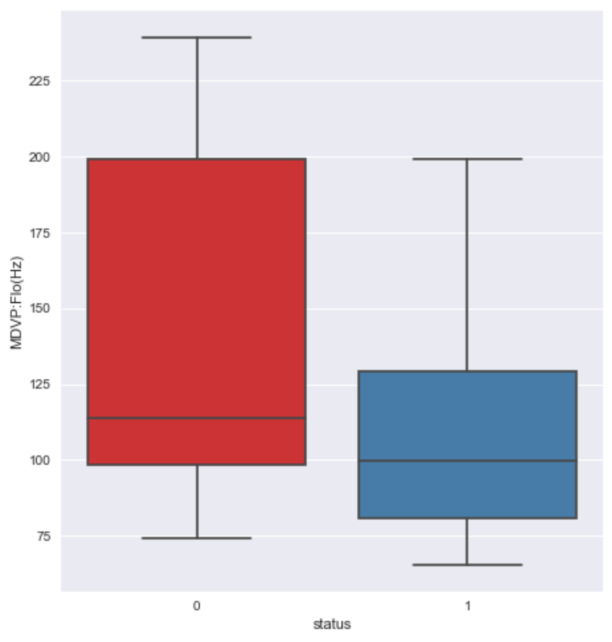
```
sns.boxplot(x='status',y='MDVP:Fhi(Hz)',data=data)
```

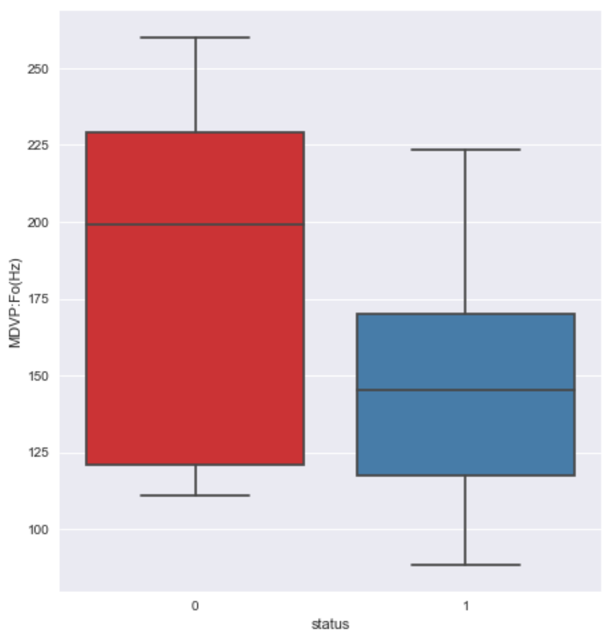
```
<matplotlib.axes._subplots.AxesSubplot at 0x11b5115c0>
```



```
fig, ax = plt.subplots(1,2,figsize=(16,8))
sns.boxplot(x='status',y='MDVP:Flo(Hz)',data=data,palette="Set1",ax=ax[0])
sns.boxplot(x='status',y='MDVP:Fo(Hz)',data=data,palette="Set1",ax=ax[1])
```

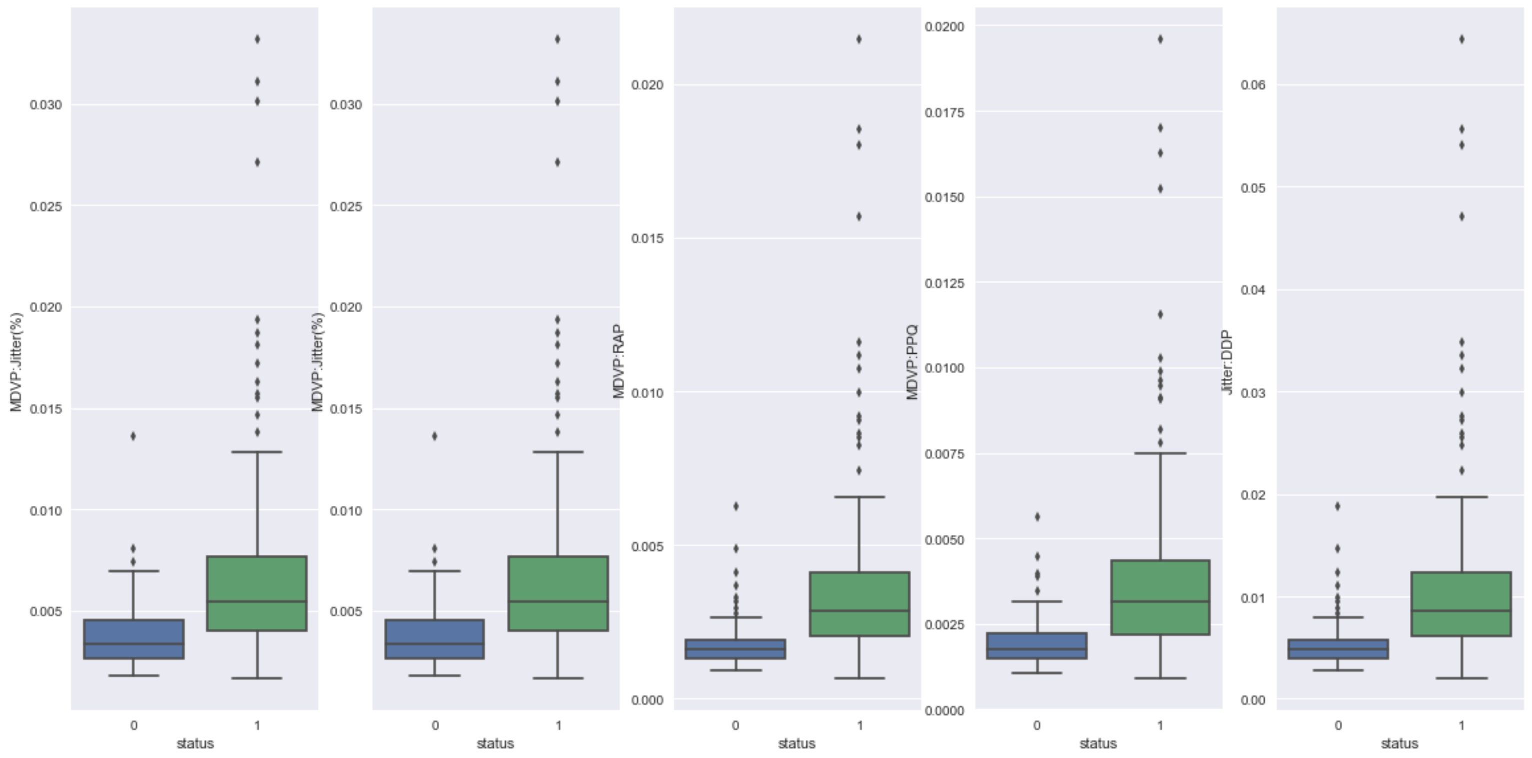
```
<matplotlib.axes._subplots.AxesSubplot at 0x11b88b0b8>
```





When we look the relationship between status and MDVP:Fo(Hz) we can see the median value is around 199 Hz for people who are normal. For people who are affected with Parkinsons the median value comes around 145 Hz

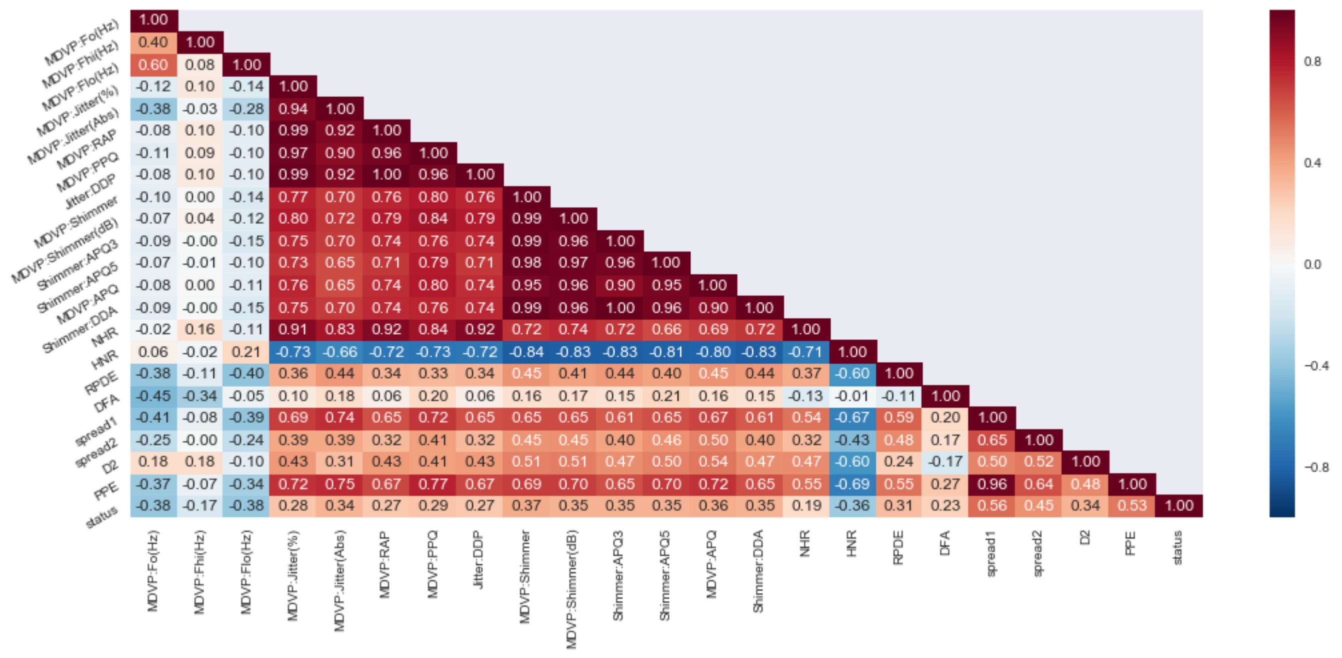
```
# For categorical predictors
cols = ["MDVP:Jitter(%)","MDVP:Jitter(%)","MDVP:RAP","MDVP:PPQ","Jitter:DDP"]
fig, axs = plt.subplots(ncols = 5,figsize=(16,8))
fig.tight_layout()
for i in range(0,len(cols)):
    sns.boxplot(x='status',y=cols[i],data=data, ax = axs[i])
```



People who are suffering for PD tend to have higher `jitter %`. It seems if the values goes above 0.15 we can confirm the patient is having PD. The variation of fundamental frequency is in a low range for people who is normal.

## Correlation comparision with heat map

```
import matplotlib.pyplot as plt
import seaborn as sns
corr = data.corr()
sns.set_context("notebook", font_scale=1.0, rc={"lines.linewidth": 3.5})
plt.figure(figsize=(18,7))
# create a mask so we only see the correlation values once
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask, 1)] = True
a = sns.heatmap(corr,mask=mask, annot=True, fmt='.2f')
rotx = a.set_xticklabels(a.get_xticklabels(), rotation=90)
roty = a.set_yticklabels(a.get_yticklabels(), rotation=30)
```



- **MDVP:Jitter(%)** has a very high correlation with **MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP**
- **MDVP:Shimmer** has a very correlation with **MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA** this may be because they are related to each other. This may be because `multi-dimensinal voice programs analysis` is closely related with these variables
- The target variable **status** has a weak positive corelation with **spread1**

## Applying models

### Decision Tree

Decision trees can be used to predict both continuous and discrete values i.e. they work well for both regression and classification tasks.

```
from sklearn import metrics
```

```
X = data.drop("status",axis=1)
Y = data["status"]
```

```
# Splitting Data into 70% Training data and 30% Testing Data:
X_train, X_test, y_train,  y_test = train_test_split(X, Y,train_size=0.7, test_size=0.3, random_state=42)
print(len(X_train)),print(len(X_test))
```

```
136
59







(None, None)
```

```
# Applying decision tree model
dt_model = DecisionTreeClassifier(criterion='entropy',max_depth=6,random_state=100,min_samples_leaf=5)
```

```
dt_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=6,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=5,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=100, splitter='best')
```

```
dt_model.score(X_test , y_test)
```

```
0.89830508474576276
```

```
y_pred = dt_model.predict(X_test)
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[11,  4],
       [ 2, 42]])
```

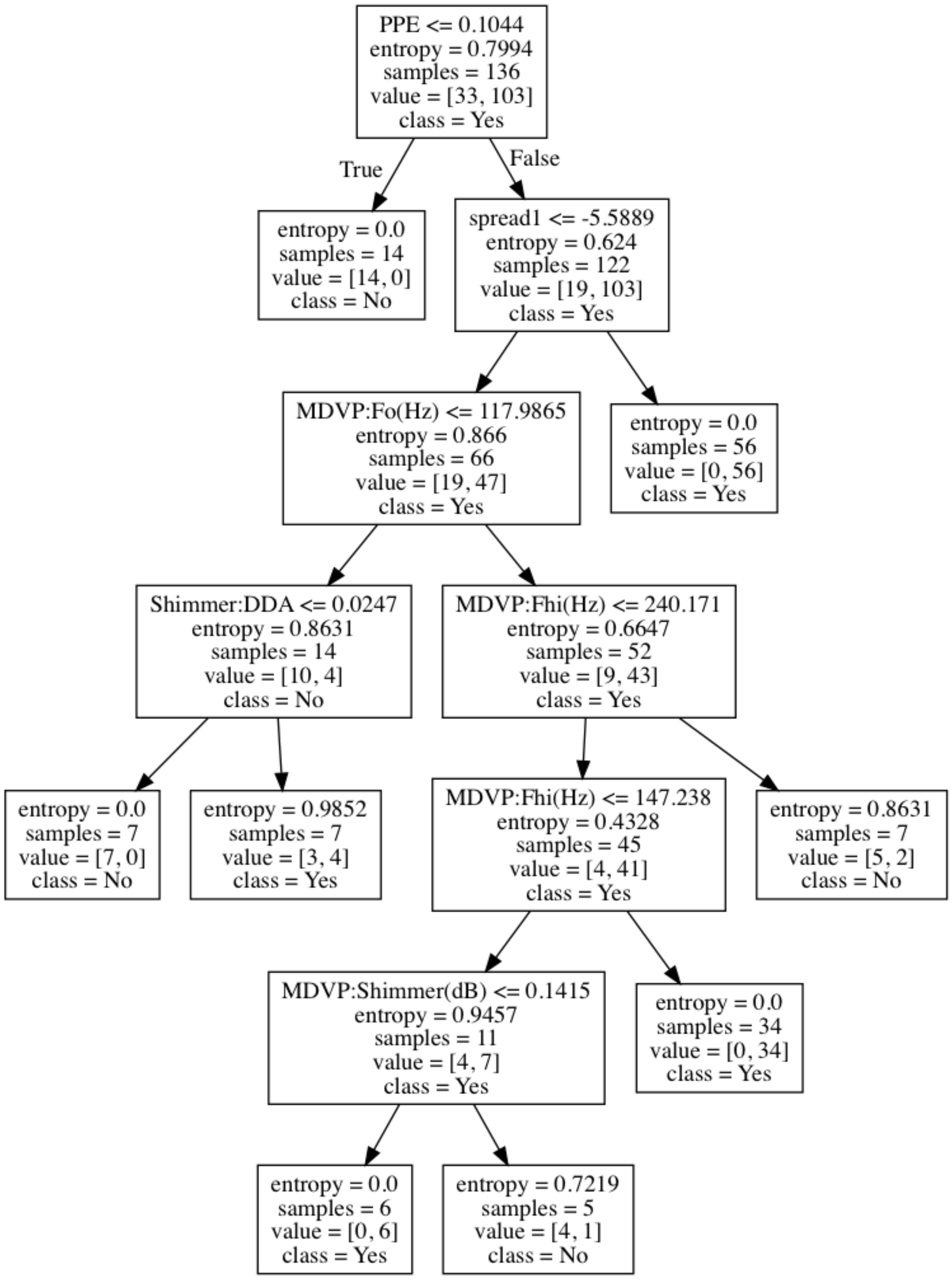
```
#Count mis-classified one
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
```

```
Misclassified samples: 6
```

```
from IPython.display import Image
from sklearn import tree
from os import system

train_char_label = ['No', 'Yes']
pd_tree_regularized = open('pd_tree_regularized.dot','w')
dot_data = tree.export_graphviz(dt_model, out_file= pd_tree_regularized , feature_names = list(X_train), class_names = list(train_char_label))

pd_tree_regularized.close()

print (pd.DataFrame(dt_model.feature_importances_, columns = ["Imp"], index = X_train.columns))
```

```
                       Imp
MDVP:Fo(Hz)       0.113981
MDVP:Fhi(Hz)      0.196589
MDVP:Flo(Hz)      0.000000
MDVP:Jitter(%)    0.000000
MDVP:Jitter(Abs)  0.000000
MDVP:RAP          0.000000
MDVP:PPQ          0.000000
Jitter:DDP        0.000000
MDVP:Shimmer      0.000000
MDVP:Shimmer(dB)  0.073695
Shimmer:APQ3      0.000000
Shimmer:APQ5      0.000000
MDVP:APQ          0.000000
Shimmer:DDA       0.056276
NHR               0.000000
HNR               0.000000
RPDE              0.000000
DFA               0.000000
spread1           0.205876
spread2           0.000000
D2                0.000000
PPE               0.353584
```

```
# You can also copy the script in the .dot file and paste it at http://webgraphviz.com/ to get tree view
# or create a .png as below
system("dot -Tpng pd_tree_regularized.dot -o pd_tree_regularized.png")
Image("pd_tree_regularized.png")
```



### K Nearest Neighbour

```
k_model = KNeighborsClassifier(n_neighbors=5)
k_model.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

```
k_model.score(X_test,y_test)
```

```
0.830508474576271 19
```

```
y_pred = k_model.predict(X_test)
```

```
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples in KNN: {}'.format(count_misclassified))
```

```
Misclassified samples in KNN: 10
```

### Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rfcl = RandomForestClassifier(n_estimators = 50)
rfcl = rfcl.fit(X_train, y_train)
```

```
y_pred = rfcl.predict(X_test)
rfcl.score(X_test , y_test)
```

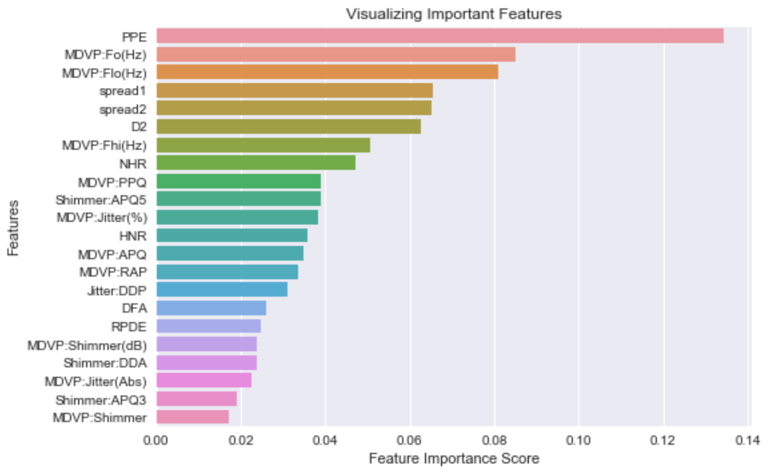
```
0.9491525423728 8138
```

```
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples in Random Forest: {}'.format(count_misclassified))
```

```
Misclassified samples in Random Forest: 3
```

```
feature_imp = pd.Series(rfcl.feature_importances_,index=X.columns).sort_values(ascending=False)
feature_imp
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```

```
/Users/suniljacob/anaconda/lib/python3.6/site-packages/matplotlib/axes/_axes.py:545: UserWarning: No labelled objects found. Use label='...' kwa
rg on individual plots.
  warnings.warn("No labelled objects found. "
```



### Bagging

```
from sklearn.ensemble import BaggingClassifier
bgcl = BaggingClassifier(base_estimator=dt_model, n_estimators=50, max_samples=.7)
bgcl = bgcl.fit(X_train, y_train)
y_pred = bgcl.predict(X_test)
bgcl.score(X_test , y_test)
```

```
0.89830508474576276
```

```
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples in Bagging: {}'.format(count_misclassified))
```

```
Misclassified samples in Bagging: 6
```

### AdaBoosting

```
from sklearn.ensemble import AdaBoostClassifier
abcl = AdaBoostClassifier( n_estimators= 50)
abcl = abcl.fit(X_train,y_train)
y_pred = abcl.predict(X_test)
abcl.score(X_test , y_test)
```

```
0.89830508474576276
```

```
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples in Ada Boosting: {}'.format(count_misclassified))
```

```
Misclassified samples in Ada Boosting: 6
```

### Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
gbcl = GradientBoostingClassifier(n_estimators = 50, learning_rate = 0.05)
gbcl = gbcl.fit(X_train,y_train)
y_pred = gbcl.predict(X_test)
gbcl.score(X_test , y_test)
```

```
0.89830508474576276
```

```
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples in Gradient Boosting: {}'.format(count_misclassified))
```

```
Misclassified samples in Gradient Boosting: 6
```

Of all the above ones Random Forest algorithm gave the maximum accuracy.