G L Bajaj Institute of Technology and Management Greater Noida – 20130

MINOR PROJECT

ODD SEMESTER,2021

SYNOPSIS

TOPIC: Detection Of ParkinsonDisease using ML

Vivek Payla	**1901920100331**
Vivek Sharma	**1901920100332**
Yash Pratap Singh	**1901920100333**

SUPERVISOR: Mr. Pawan Kumar Singh

Acknowledgment

I would like to place on record my deep sense of gratitude to Mr. Pawan Kumar Singh, Assistant Professor, Gl Bajaj Institute Of Technology And Management. India for his generousguidance, help and useful suggestions.

I also wish to extend my thanks to my classmates for their insightful comments and constructive suggestions to improve the quality of this project work.

Declaration

We hereby declare than this submission is our own work and that , to the best our knowledge and beliefs, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma from a university or other institute of higher learning, except where due acknowledgement has been made in the text.

Name	Enrollment No.
Vivek Payla	1901920100331
Vivek Sharma	1901920100332
Yash Pratap Singh	1901920100333

INTRODUCTION:

What is Parkinson's disease?

Parkinson's disease is a nervous system disease that affects your ability to control movement. The disease usually starts out slowly and worsens over time. If you have Parkinson's disease, you may shake, have muscle stiffness, and have trouble walking and maintaining your balance and coordination. As the disease worsens, you may have trouble talking, sleeping, have mental and memory problems, experience behavioral changes and have other symptoms.

Who gets Parkinson's disease?

About 50% more men than women get Parkinson's disease. It is most commonly seen in persons 60 years of age and older. However, up to 10% of patients are diagnosed before age 50.

What are the symptoms of Parkinson's disease?

Symptoms of Parkinson's disease and the rate of decline vary widely from person to person. The most common symptoms include:

- Tremor
- Slowness of Movement
- Speech/Vocal changes: Speech may be quick, become slurred or be soft in tone. You may hesitate before speaking. The pitch of your voice may become unchanged (monotone).
- Handwriting changes : becomes smaller and more difficult to read.
- Depression and Anxiety With Hallucinations/delusions.

We have been working on finding the best solution to detect Parkinson disease and we came up with 2 solutions

1. **Detecting parkinson through patient's handwriting image**
2. **Real time voice analysis of the patient.**

The first step of our project was detecting parkinson through patient's handwriting image by using spiral dataset analysis for which we used Histogram of Oriented Gradient.

History of Oriented Gradient:

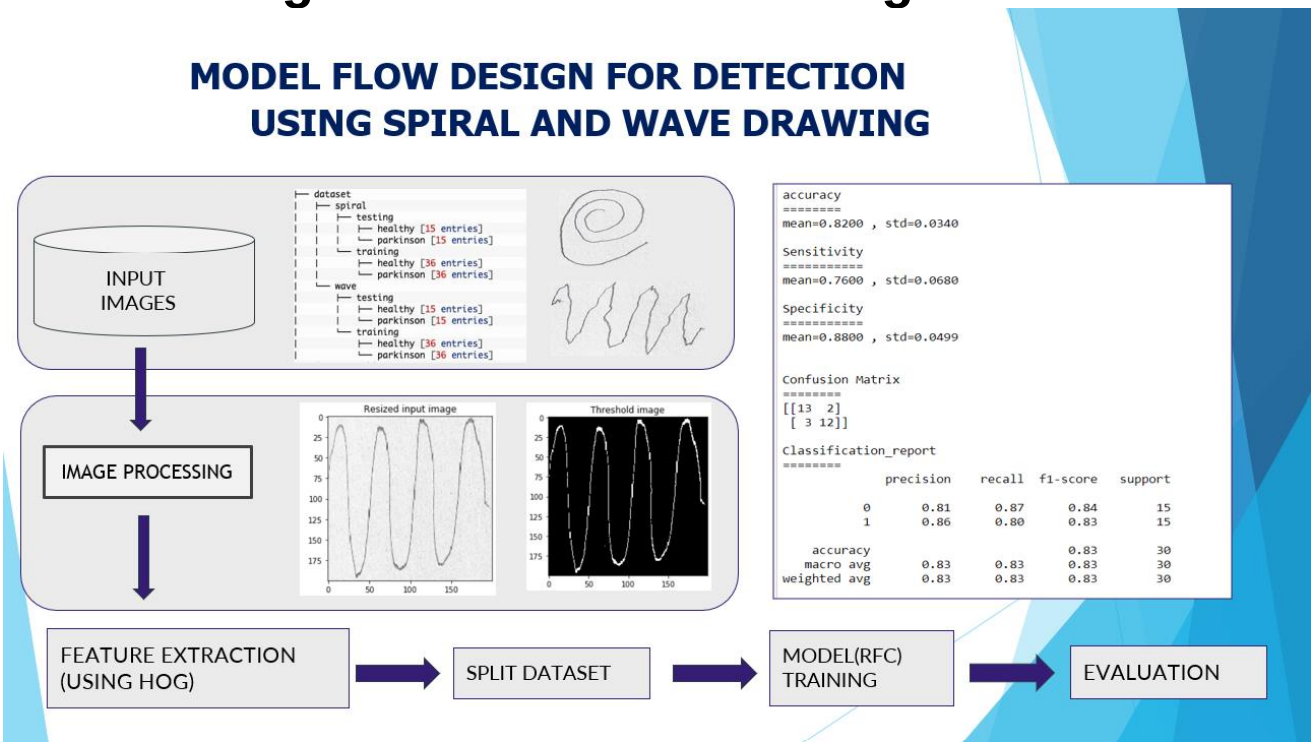
HOG descriptor focuses on the structure or the shape of an object. In the case of edge features, we only identify if the pixel is an edge or not. HOG is able to provide the edge direction as well. This is

done by extracting the gradient and orientation (or magnitude and direction) of the edges.

So basically, when image with handwriting is given it converts it into an array and also has its magnitude and direction, so when we do the same with our testing image it compares both array and gives us the result according to it.

Then we created a model flow design for the above implementation of HOG and how our data will be processed and evaluated.

Following is the model flow design:



After this we, started working on our speech detection part of the project.

Firstly, we defined the features of the speech recording we will be taking in our consideration based on the frequency, amplitude and jitter in the sound wave of our speech data for which we used correlation matrix

Next part of our program is the implementation of best regression model to our program for better accuracy. We compared multiple models and finally found that the best suited regression model for our project is XGBOOST regression model.

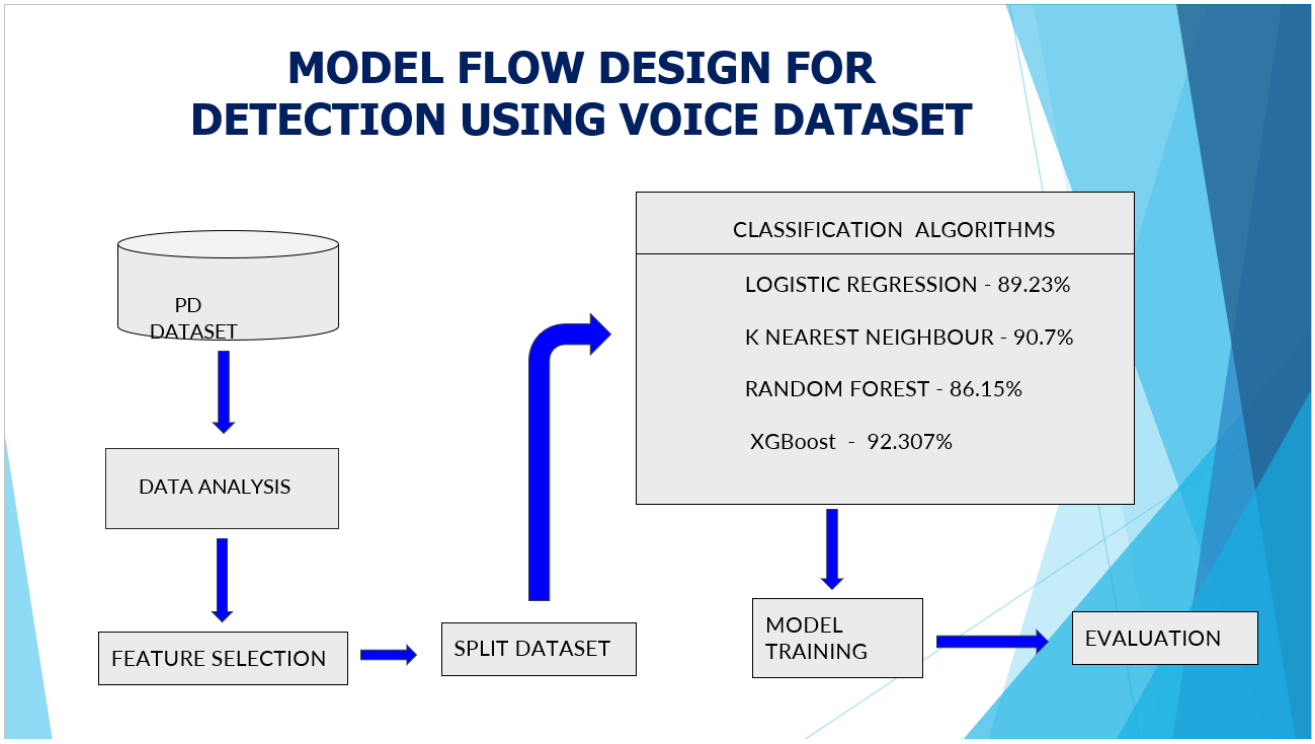
XGBoost

XGBoost expects to have the base learners which are uniformly bad at the remainder so that when all the predictions are combined, bad predictions cancels out and better one sums up to form final good predictions while having better accuracy. It also requires less computation steps for model training.

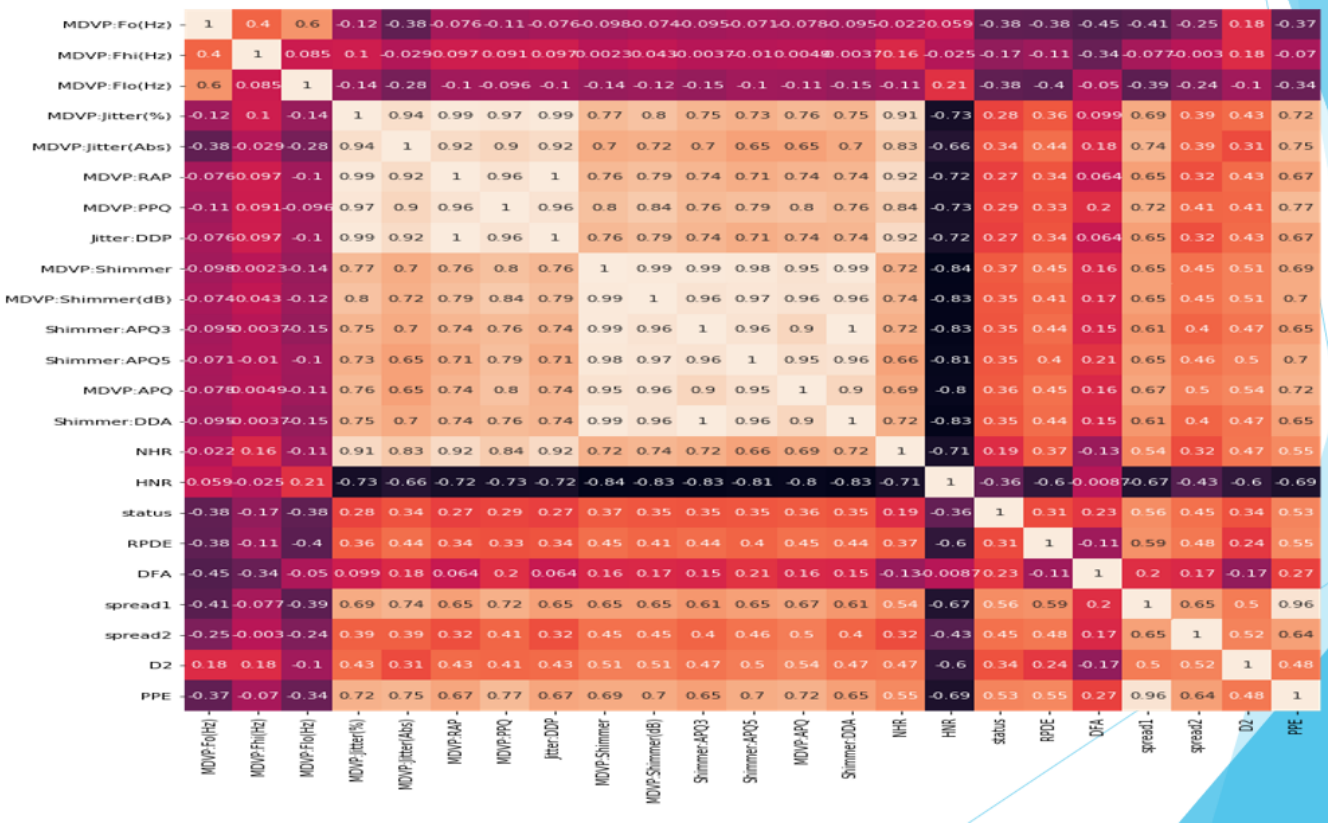
XGBoost utilizes the power of parallel processing and that is why it is much faster than GBM. It uses multiple CPU cores to execute the model.

XGBoost has in-built L1 (Lasso Regression) and L2 (Ridge Regression) regularization which prevents the model from overfitting.

After finalizing the regression model, we created a model flow design for our voice detection part of the program.



CORRELATION MATRIX



```
def load_split(path):
    imagePaths=list(paths.list_images(path))
    data=[]
    labels=[]

    for imagepath in imagepaths:
        label = imagepath.split(os.path.sep)[-2]

        image= cv2.imread(imagepath)
        image= cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        image = cv2.resize(image,(200,200))

        image = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_(

        features = quantify_images(image)

        data.append(features)
        labels.append(label)

    return (np.array(data),np.array(labels))
```

In [1]:

```
def quantify_images(image):
    features = feature.hog(image, orientations=9, pixels_per_cell=(10,10), cells_
    return features
```

In [2]:

In [4]:

```
def predict_new(path):
    img = cv2.imread()
    if img is not None:
        gray = cv2.cvColor(img,cv2,COLOR_BGR2GRAY)
        img = cv2.resize(gray,(200,200))
        out = img.copy()
        img = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)
        feature = quantify_images(img)
        preds = model.predict([feature])
        lab = le.inverse_transform(preds)[0]
        color = (0,122,0) if lab=='healthy' else (0,0,122)
        cv2.putText(out,lab, (3,20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, Color,2)

        return out
```

In [1]:
```
from sklearn.ensemble import RandomForestClassifier
no_of_trails=5

trials= {}

# starting the trials
for i in range(0, no_of_trails):

    #train the model
    print('Training the model {} of {}... '.format(i+1,no_of_trails))

    model = RandomForestClassifier(n_estimators=100)
    model.fit(trainX, trainY)

    #making predictions on the test dataset
    predictions = model.predict(testX)
    metrics={}

    #Computer the confusion matrix to measure accuracy
    cm= confusion_matrix(testY, predictions).flatten()
    (TrueNeg,FalsePos,FalseNeg,TruePos)= cm
    metrics['accuracy']=(TruePos + TrueNeg)/float(cm.sum())
    metrics['Sensitivity']= TruePos / float(TruePos + FalseNeg)
    metrics['Specificity']= TrueNeg / float(TrueNeg + FalsePos)

    for (k,v) in metrics.item():
        l=trials.get(k,[])
        l.append(v)
        trials[k]=1
```

```
Training the model 1 of 5...
Training the model 2 of 5...
Training the model 3 of 5...
Training the model 4 of 5...
Training the model 5 of 5...
```

	accuracy	Sensitivity	Specificity
0	0.866667	0.800000	0.933333
1	0.900000	0.866667	0.933333
2	0.766667	0.666667	0.866667
3	0.800000	0.733333	0.866667
4	0.866667	0.800000	0.933333

In [1]:
```
import pandas as pd
dfo=pd.read_csv('parkinsons.data')
```

In [2]:
```
cls=list(dfo)
print(cls)
```

```
['name', 'MDVP:Fo(Hz)', 'MDVP:Fhi(Hz)', 'MDVP:Flo(Hz)', 'MDVP:Jitter(%)', 'MDV
P:Jitter(Abs)', 'MDVP:RAP', 'MDVP:PPQ', 'Jitter:DDP', 'MDVP:Shimmer', 'MDVP:Shi
mmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5', 'MDVP:APQ', 'Shimmer:DDA', 'NHR', 'H
NR', 'status', 'RPDE', 'DFA', 'spread1', 'spread2', 'D2', 'PPE']
```

In [3]:
```
dfo.head()
```

Out[3]:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	I
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	

5 rows × 24 columns

In [37]:
```
dfo.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
```

In [38]:
```
dfo.describe()
```

Out[38]:

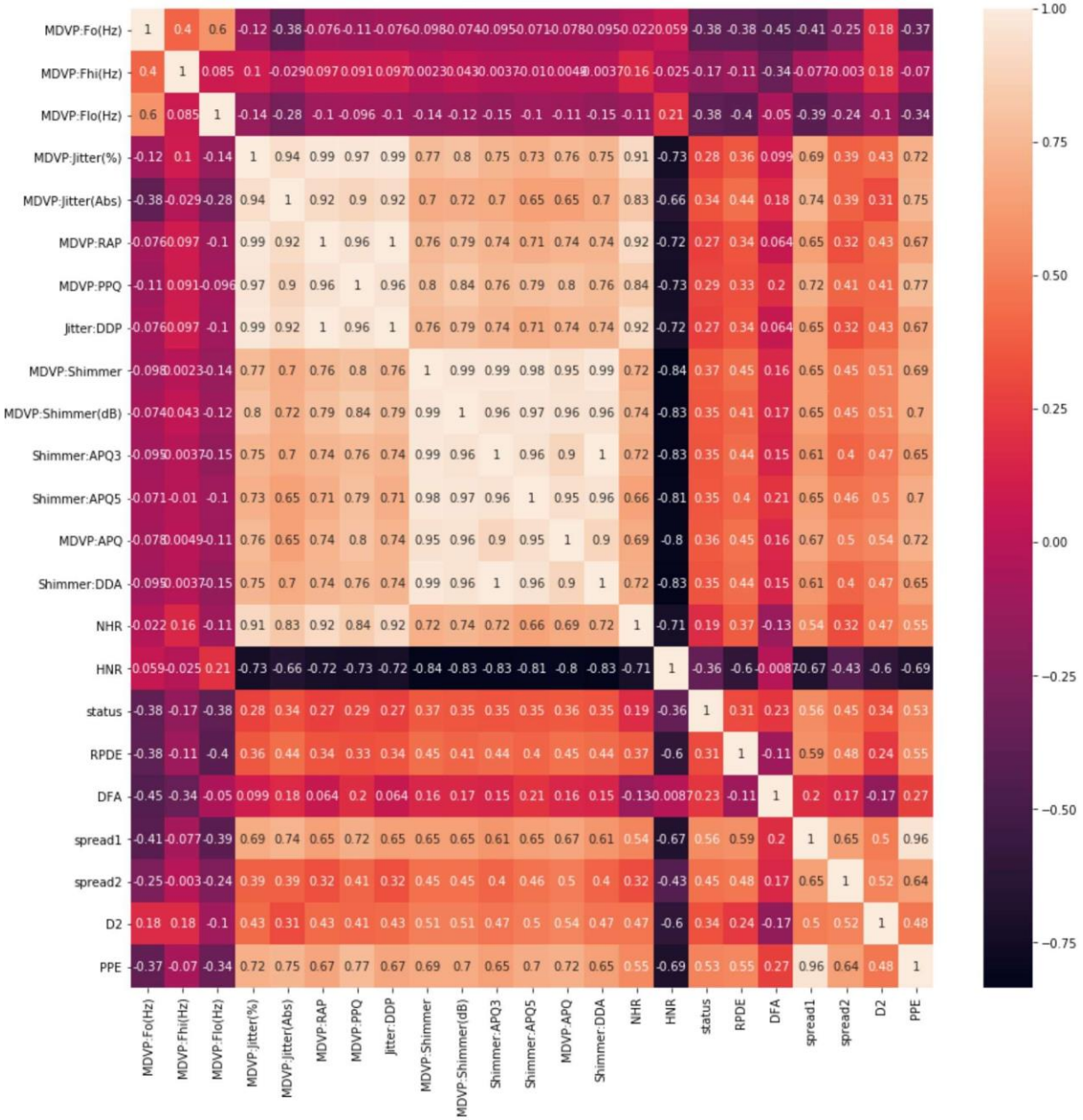
	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	M
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	1
mean	154.228641	197.104918	116.324631	0.006220	0.000044	0.003306	
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	
min	88.333000	102.145000	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

In [39]:
```
dfo.isnull().sum()
```

Out[39]:
```
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 [4]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot heatmap
fig, ax = plt.subplots(figsize=(15,15))
ax = sns.heatmap(dfo.corr(), annot=True);
```

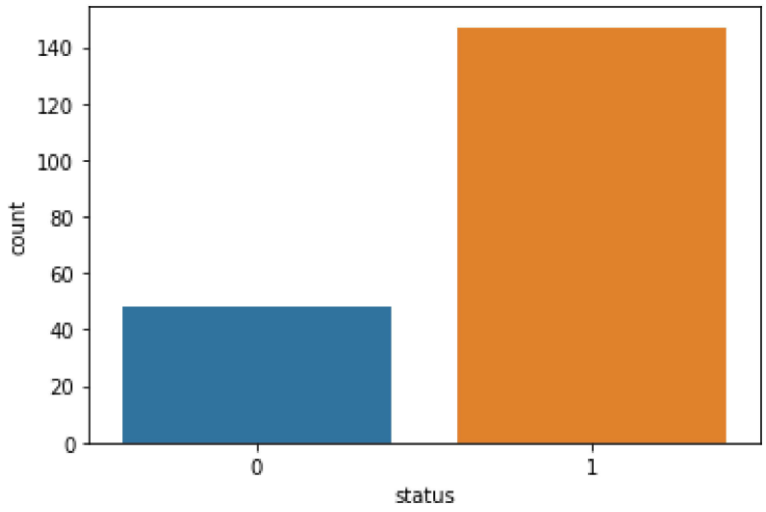


In [6]:
```
dfo['status'].value_counts()
```

Out[6]:
```
1    147
0     48
Name: status, dtype: int64
```

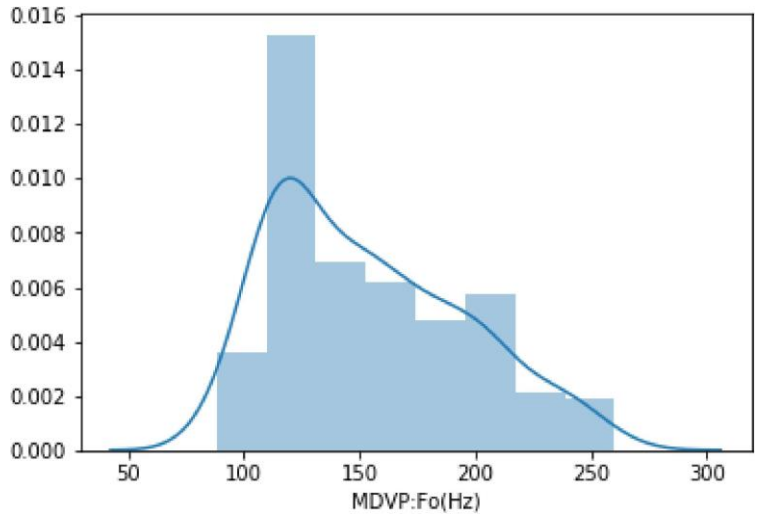
In [8]:
```
sns.countplot(dfo['status'],label='count')
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f537dc0bc8>
```



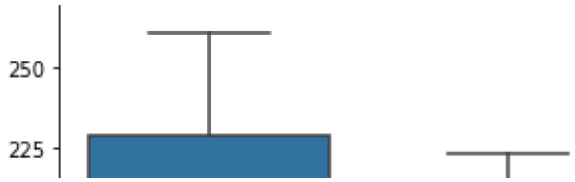
In [9]:
```
sns.distplot(dfo['MDVP:Fo(Hz)'])
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f537538108>
```



In [11]:
```
for i in dfo:
    if i != 'status' and i != 'name':
        sns.catplot(x='status',y=i,kind='box',data=dfo)
```

```
C:\Users\ASUS\anaconda3\lib\site-packages\seaborn\axisgrid.py:324: RuntimeWar
ning: More than 20 figures have been opened. Figures created through the pypl
ot interface (`matplotlib.pyplot.figure`) are retained until explicitly close
d and may consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  fig, axes = plt.subplots(nrow, ncol, **kwargs)
C:\Users\ASUS\anaconda3\lib\site-packages\seaborn\axisgrid.py:324: RuntimeWar
ning: More than 20 figures have been opened. Figures created through the pypl
ot interface (`matplotlib.pyplot.figure`) are retained until explicitly close
d and may consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  fig, axes = plt.subplots(nrow, ncol, **kwargs)
```



In [12]:
```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import accuracy_score,mean_squared_error
```

In [13]:
```
features=dfo.drop(['status','name'],axis=1)
labels=dfo['status']
```

In [14]:
```
scaler=MinMaxScaler((-1,1))
x=scaler.fit_transform(features)
y=labels
```

In [15]:
```
x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.2, random_state=
```

In [36]:
```
print('Training Features Shape:', x_train.shape)
print('Training Labels Shape:', y_train.shape)
print('Testing Features Shape:', x_test.shape)
print('Testing Labels Shape:', y_test.shape)
```

```
Training Features Shape: (156, 22)
Training Labels Shape: (156,)
Testing Features Shape: (39, 22)
Testing Labels Shape: (39,)
```

In [17]:
```
from sklearn.linear_model import LogisticRegression
from xgboost import XGBRFClassifier,XGBClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier,BaggingClassifier,ExtraTreesClass
from sklearn.neighbors import KNeighborsClassifier
```

In [22]:
```
lr=cross_val_score(LogisticRegression(),x_train,y_train)
xgbc=cross_val_score(XGBRFClassifier(),x_train,y_train)
xgb=cross_val_score(XGBClassifier(),x_train,y_train)
svm=cross_val_score(SVC(),x_train,y_train)
#nb=cross_val_score(MultinomialNB(),x_train,y_train)
dtc=cross_val_score(DecisionTreeClassifier(),x_train,y_train)
adb=cross_val_score(AdaBoostClassifier(),x_train,y_train)
bbc=cross_val_score(BaggingClassifier(),x_train,y_train)
etc=cross_val_score(ExtraTreesClassifier(),x_train,y_train)
gbc=cross_val_score(GradientBoostingClassifier(),x_train,y_train)
rfc=cross_val_score(RandomForestClassifier(),x_train,y_train)
#vc=cross_val_score(VotingClassifier(estimators),x_train,y_train)
knc=cross_val_score(KNeighborsClassifier(),x_train,y_train)
```

```
C:\Users\ASUS\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in
a future release. To remove this warning, do the following: 1) Pass option use_
label_encoder=False when constructing XGBClassifier object; and 2) Encode your
labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\ASUS\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in
a future release. To remove this warning, do the following: 1) Pass option use_
label_encoder=False when constructing XGBClassifier object; and 2) Encode your
labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\ASUS\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in
a future release. To remove this warning, do the following: 1) Pass option use_
label_encoder=False when constructing XGBClassifier object; and 2) Encode your
labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\ASUS\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in
a future release. To remove this warning, do the following: 1) Pass option use_
label_encoder=False when constructing XGBClassifier object; and 2) Encode your
labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\ASUS\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in
a future release. To remove this warning, do the following: 1) Pass option use_
label_encoder=False when constructing XGBClassifier object; and 2) Encode your
labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[11:21:50] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the defa
ult evaluation metric used with the objective 'binary:logistic' was changed fro
m 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[11:21:50] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the defa
ult evaluation metric used with the objective 'binary:logistic' was changed fro
m 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[11:21:50] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the defa
ult evaluation metric used with the objective 'binary:logistic' was changed fro
m 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

```
[11:21:50] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the defa
ult evaluation metric used with the objective 'binary:logistic' was changed fro
m 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[11:21:50] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the defa
ult evaluation metric used with the objective 'binary:logistic' was changed fro
m 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.

C:\Users\ASUS\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in
a future release. To remove this warning, do the following: 1) Pass option use_
label_encoder=False when constructing XGBClassifier object; and 2) Encode your
labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\ASUS\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in
a future release. To remove this warning, do the following: 1) Pass option use_
label_encoder=False when constructing XGBClassifier object; and 2) Encode your
labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\ASUS\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in
a future release. To remove this warning, do the following: 1) Pass option use_
label_encoder=False when constructing XGBClassifier object; and 2) Encode your
labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\ASUS\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in
a future release. To remove this warning, do the following: 1) Pass option use_
label_encoder=False when constructing XGBClassifier object; and 2) Encode your
labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\ASUS\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in
a future release. To remove this warning, do the following: 1) Pass option use_
label_encoder=False when constructing XGBClassifier object; and 2) Encode your
labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

```
[11:21:50] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the de
fault evaluation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restor
e the old behavior.
[11:21:51] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the de
fault evaluation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restor
e the old behavior.
[11:21:51] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the de
fault evaluation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restor
e the old behavior.
[11:21:51] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the de
fault evaluation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restor
e the old behavior.
[11:21:51] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the de
fault evaluation metric used with the objective 'binary:logistic' was changed
```

```
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restor
e the old behavior.
```

In [20]:
```
knc.mean()
```

Out[20]:
```
0.8915322580645162
```

In [27]:
```
print('log reg',lr,lr.mean())#
print('knn',knc,knc.mean())#
print('rfc',rfc,rfc.mean())#
print('xgb',xgb,xgb.mean())#
```

```
log reg [0.90625    0.90322581 0.87096774 0.83870968 0.80645161] 0.865120967741
9356
                                                      0.93548387]
rfc             0.93548387 0.87096774 0.93548387 0.87096774] 0.9038306451612904
xgb             0.93548387 0.83870968 0.96774194 0.90322581] 0.9102822580645162
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
knn [0.8125     0.90322581 0.90322581 0.90322581             0.8915322580645162
    [0.90625
    [0.90625
```

Xgboost regressor

In [1]:
```
import pandas as pd
dfo=pd.read_csv('parkinsons.data')
cols = list(dfo)

cols.insert(24, cols.pop(cols.index('status')))
dfo = dfo.loc[:, cols]
```

In [2]:
```
import xgboost as xgb
import pandas as pd
import numpy as np
X,y = dfo.iloc[:,1:-1],dfo.iloc[:,-1]
```

In [3]:
```

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_s
```

In [4]:
```
xg_reg = xgb.XGBRegressor(objective ='binary:hinge', colsample_bytree = 0.3, lear
                max_depth = 5, alpha = 10, n_estimators = 10) #learning_rate = 0.
```

In [5]:
```
xg_reg.fit(X_train,y_train)

preds = xg_reg.predict(X_test)
```

In [6]:
```
X_test.head()
```

Out[6]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MD
56	141.781	181.937	129.157	0.00274	0.00002	0.00132	
141	208.083	253.792	91.802	0.00757	0.00004	0.00428	
170	244.990	272.210	239.170	0.00451	0.00002	0.00279	
65	228.969	239.541	113.201	0.00238	0.00001	0.00136	
66	140.341	159.774	67.021	0.00817	0.00006	0.00430	

5 rows × 22 columns

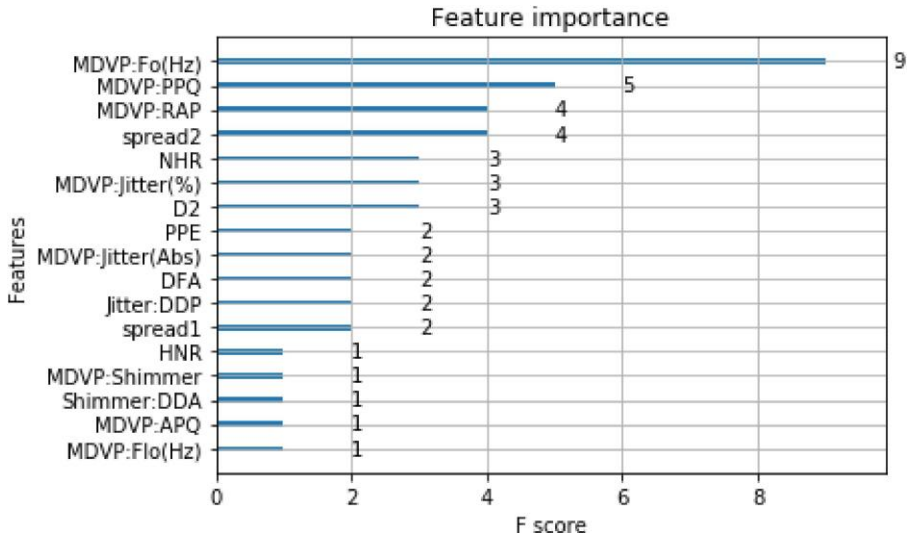
In [7]:
```
print(preds)
```

```
[1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.

 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
```

In [8]:
```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE: %f" % (rmse))

from sklearn.metrics import accuracy_score
print('XGBoost model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, pre
```

```
RMSE: 0.277350
XGBoost model accuracy score: 92.3077
```

In [9]:
```
import matplotlib.pyplot as plt
xgb.plot_importance(xg_reg)
plt.rcParams['figure.figsize'] = [14, 5]
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

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

In [2]:
```
dfo = pd.read_csv("parkinsons.data")
```

In [4]:
```
dfo.shape
```

Out[4]: (195, 24)

In [6]:
```
from sklearn.model_selection import train_test_split
```

In [7]:
```
X = dfo.drop(['name','status'],axis =1)
y = dfo['status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

In [8]:
```
from sklearn.preprocessing import StandardScaler,MinMaxScaler
SS = StandardScaler()
MMS = MinMaxScaler()
```

In [9]:
```
x_train = SS.fit_transform(X_train)
x_test = SS.fit_transform(X_test)
```

USING RANDOM FOREST

In [10]:
```
from sklearn.ensemble import RandomForestClassifier
RFC = RandomForestClassifier(n_estimators=105)
```

In [11]:
```
RFC.fit(x_train,y_train)
```

Out[11]: RandomForestClassifier(n_estimators=105)

In [12]:
```
pred = RFC.predict(x_test)
```

In [13]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [14]:
```
print(confusion_matrix(y_test,pred))
```

```
[[ 8  9]
 [ 0 48]]
```

In [15]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.47      0.64        17
           1       0.84      1.00      0.91        48

    accuracy                           0.86        65
   macro avg       0.92      0.74      0.78        65
weighted avg       0.88      0.86      0.84        65
```

In [16]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, pred)
```

Out[16]:
```
0.8615384615384616
```

USING KNN

In [17]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=11)
```

In [18]:
```
knn.fit(x_train,y_train)
```

Out[18]:
```
KNeighborsClassifier(n_neighbors=11)
```

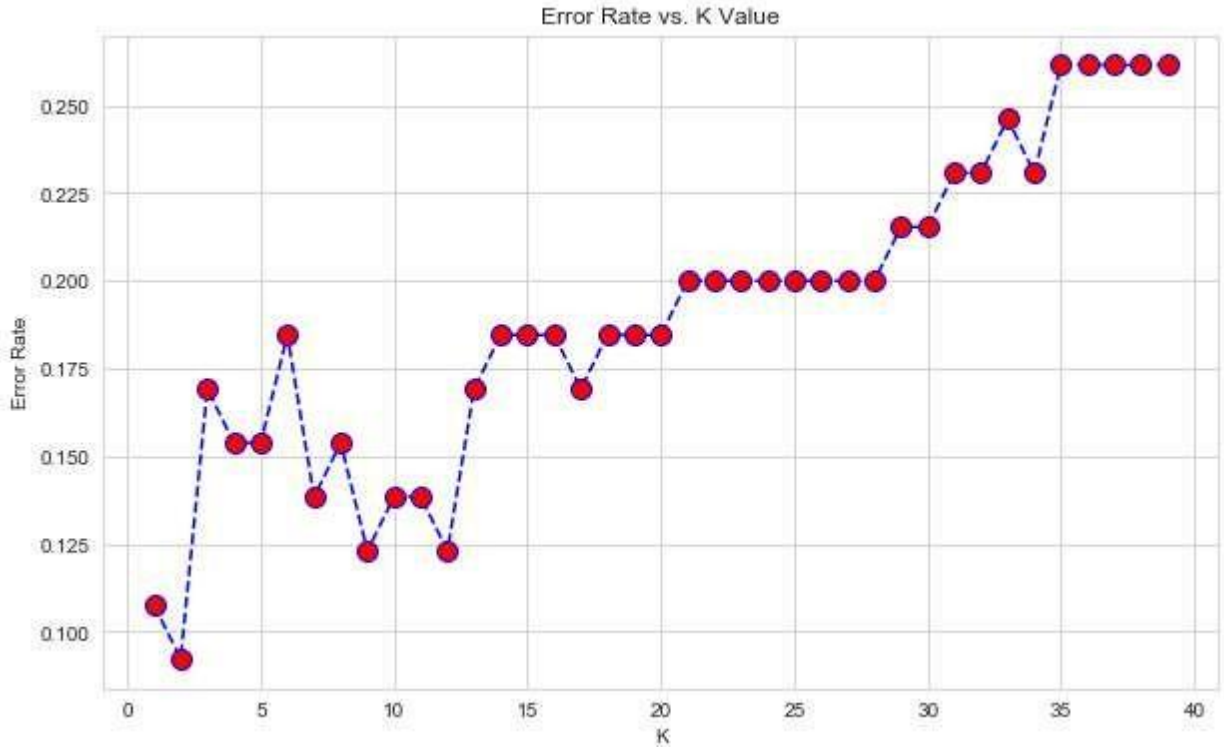
In [19]:
```
#GETTING K VALUE WITH LEAST ERRORS
error_rate = []
K = []

for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    pred_i = knn.predict(x_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [20]:
```
error_rate
```

Out[20]:
```
[0.1076923076923077,
 0.09230769230769231,
 0.16923076923076924,
 0.15384615384615385,
 0.15384615384615385,
 0.18461538461538463,
 0.13846153846153847,
 0.15384615384615385,
 0.12307692307692308,
 0.13846153846153847,
 0.13846153846153847,
 0.12307692307692308,
 0.16923076923076924,
 0.18461538461538463,
 0.18461538461538463,
 0.18461538461538463,
 0.16923076923076924,
 0.18461538461538463,
 0.18461538461538463,
 0.18461538461538463,
 0.2,
 0.2,
 0.2,
 0.2,
 0.2,
 0.2,
 0.2,
 0.2,
 0.2153846153846154,
 0.2153846153846154,
 0.23076923076923078,
 0.23076923076923078,
 0.24615384615384617,
 0.23076923076923078,
 0.26153846153846155,
 0.26153846153846155,
 0.26153846153846155,
 0.26153846153846155,
 0.26153846153846155]
```

In [21]:
```
sns.set_style('whitegrid')
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[21]:
```
Text(0, 0.5, 'Error Rate')
```



In [22]:
```
knn = KNeighborsClassifier(n_neighbors=1)
```

In [23]:
```
knn.fit(x_train,y_train)
```

Out[23]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [24]:
```
predict = knn.predict(x_test)
```

In [25]:
```
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       0.73      0.94      0.82        17
           1       0.98      0.88      0.92        48

    accuracy                           0.89        65
   macro avg       0.85      0.91      0.87        65
weighted avg       0.91      0.89      0.90        65
```

In [26]:
```
knn.score(x_test,y_test)
```

Out[26]:
```
0.8923076923076924
```

In [27]:
```
knn = KNeighborsClassifier(n_neighbors=2)

knn.fit(x_train,y_train)
pred = knn.predict(x_test)
```

In [28]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.74      1.00      0.85        17
           1       1.00      0.88      0.93        48

    accuracy                           0.91        65
   macro avg       0.87      0.94      0.89        65
weighted avg       0.93      0.91      0.91        65
```

In [29]:
```
knn.score(x_test,y_test)
```

Out[29]:
```
0.9076923076923077
```

USING LOGISTIC REGRESSION

In [30]:
```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
```

In [31]:
```
LR.fit(x_train,y_train)
```

Out[31]:
```
LogisticRegression()
```

In [32]:
```
pred1 = LR.predict(x_test)
```

In [33]:
```
pred1
```

Out[33]:
```
array([1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
      dtype=int64)
```

In [34]:
```
x_test
```

Out[34]:
```
array([[ 0.51263844, -0.09795882,  0.92780154, ...,  0.5567087 ,
         0.03147182, -0.39383713],
       [-1.20339893, -0.81139782, -0.61678308, ..., -0.46547203,
        -1.26213644,  0.2073963 ],
       [ 1.3216278 ,  0.2722459 , -0.59351855, ..., -1.69369456,
        -0.12300943, -1.11650354],
       ...,
       [-1.02340152, -0.86225894, -0.45022788, ..., -0.32734624,
        -0.66177624, -0.12772199],
       [ 1.32153699,  0.52352223, -0.81912514, ...,  1.53338771,
         1.19168218,  0.4455287 ],
       [-0.29580604, -0.51049226, -1.04505911, ..., -0.60728783,
        -0.7346092 , -0.39246474]])
```

In [35]:
```
print(classification_report(y_test,pred1))
```

```
              precision    recall  f1-score   support

           0       0.86      0.71      0.77        17
           1       0.90      0.96      0.93        48

    accuracy                           0.89        65
   macro avg       0.88      0.83      0.85        65
weighted avg       0.89      0.89      0.89        65
```

In [36]:
```
LR.score(x_test,y_test)
```

Out[36]:
```
0.8923076923076924
```

In [37]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, pred1)
```

Out[37]:
```
0.8923076923076924
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

RUN TIME PREDICTIONS

In [38]:
```
import glob
import numpy as np
import pandas as pd
import parselmouth
from parselmouth.praat import call
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# This is the function to measure voice pitch
def measurePitch(voiceID, f0min, f0max, unit):
    sound = parselmouth.Sound(voiceID) # read the sound
    pitch = call(sound, "To Pitch", 0.0, f0min, f0max) #create a praat pitch obje
    meanF0 = call(pitch, "Get mean", 0, 0, unit) # get mean pitch
    stdevF0 = call(pitch, "Get standard deviation", 0 ,0, unit) # get standard de
    harmonicity = call(sound, "To Harmonicity (cc)", 0.01, 75, 0.1, 1.0)
    hnr = call(harmonicity, "Get mean", 0, 0)
    pointProcess = call(sound, "To PointProcess (periodic, cc)", f0min, f0max)
    localJitter = call(pointProcess, "Get jitter (local)", 0, 0, 0.0001, 0.02, 1.
    localabsoluteJitter = call(pointProcess, "Get jitter (local, absolute)", 0, 0
    rapJitter = call(pointProcess, "Get jitter (rap)", 0, 0, 0.0001, 0.02, 1.3)
    ppq5Jitter = call(pointProcess, "Get jitter (ppq5)", 0, 0, 0.0001, 0.02, 1.3)
    ddpJitter = call(pointProcess, "Get jitter (ddp)", 0, 0, 0.0001, 0.02, 1.3)
    localShimmer =  call([sound, pointProcess], "Get shimmer (local)", 0, 0, 0.00
    localdbShimmer = call([sound, pointProcess], "Get shimmer (local_dB)", 0, 0,
    apq3Shimmer = call([sound, pointProcess], "Get shimmer (apq3)", 0, 0, 0.0001,
    aqpq5Shimmer = call([sound, pointProcess], "Get shimmer (apq5)", 0, 0, 0.0001
    apq11Shimmer =  call([sound, pointProcess], "Get shimmer (apq11)", 0, 0, 0.00
    ddaShimmer = call([sound, pointProcess], "Get shimmer (dda)", 0, 0, 0.0001, 0


    return meanF0, stdevF0, hnr, localJitter, localabsoluteJitter, rapJitter, ppq
def runPCA(df):
    #Z-score the Jitter and Shimmer measurements
    features = ['localJitter', 'localabsoluteJitter', 'rapJitter', 'ppq5Jitter',
                'localShimmer', 'localdbShimmer', 'apq3Shimmer', 'apq5Shimmer',
    # Separating out the features
    x = df.loc[:, features].values
    # Separating out the target
    #y = df.loc[:,['target']].values
    # Standardizing the features
    x = StandardScaler().fit_transform(x)
    #PCA
    pca = PCA(n_components=2)
    principalComponents = pca.fit_transform(x)
    principalDf = pd.DataFrame(data = principalComponents, columns = ['JitterPCA'
    principalDf
    return principalDf
# create lists to put the results
file_list = []
mean_F0_list = []
sd_F0_list = []
hnr_list = []
localJitter_list = []
localabsoluteJitter_list = []
rapJitter_list = []
ppq5Jitter_list = []
ddpJitter_list = []
```

```
localShimmer_list = []
localdbShimmer_list = []
apq3Shimmer_list = []
aqpq5Shimmer_list = []
apq11Shimmer_list = []
ddaShimmer_list = []

# Go through all the wave files in the folder and measure pitch
for wave_file in glob.glob("audio/*.wav"):
    sound = parselmouth.Sound(wave_file)
    (meanF0, stdevF0, hnr, localJitter, localabsoluteJitter, rapJitter, ppq5Jitte
    file_list.append(wave_file) # make an ID list
    mean_F0_list.append(meanF0) # make a mean F0 list
    sd_F0_list.append(stdevF0) # make a sd F0 list
    hnr_list.append(hnr)
    localJitter_list.append(localJitter)
    localabsoluteJitter_list.append(localabsoluteJitter)
    rapJitter_list.append(rapJitter)
    ppq5Jitter_list.append(ppq5Jitter)
    ddpJitter_list.append(ddpJitter)
    localShimmer_list.append(localShimmer)
    localdbShimmer_list.append(localdbShimmer)
    apq3Shimmer_list.append(apq3Shimmer)
    aqpq5Shimmer_list.append(aqpq5Shimmer)
    apq11Shimmer_list.append(apq11Shimmer)
    ddaShimmer_list.append(ddaShimmer)
df = pd.DataFrame(np.column_stack([file_list, mean_F0_list, sd_F0_list, hnr_list,
                                   columns=['voiceID', 'meanF0Hz', 'stdevF0Hz', 'HNR
                                            'ppq5Jitter', 'ddpJitter', 'localShimmer
                                            'apq11Shimmer', 'ddaShimmer'])  #add thes
pcaData = runPCA(df)

df = pd.concat([df, pcaData], axis=1)

# Write out the updated dataframe
df.to_csv("processed_results.csv", index=False)
```

In [39]:
```
data_p={'MDVP:Fo(Hz)':list(df['meanF0Hz']),'MDVP:Fhi(Hz)':[dfo['MDVP:Fhi(Hz)'].me
'MDVP:Jitter(Abs)':[dfo['MDVP:Jitter(Abs)'].mean()]*len(df),'MDVP:RAP':list(df['r
'MDVP:Shimmer':list(df['localShimmer']),'MDVP:Shimmer(dB)':list(df['localdbShimme
'MDVP:APQ':list(df['apq11Shimmer']),'Shimmer:DDA':list(df['ddaShimmer']),'NHR':[d
'HNR':list(df['HNR']),'RPDE':[dfo['RPDE'].mean()]*len(df),'DFA':[dfo['DFA'].mean(
'spread2':[dfo['spread2'].mean()]*len(df),'D2':[dfo['D2'].mean()]*len(df),'PPE':[
dfsample = pd.DataFrame.from_dict(data_p)
dfsample.head()
```

Out[39]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)
0	113.77680569854286	197.104918	116.324631	0.0014213010729582554	0.000044
1	201.9450715201333	197.104918	116.324631	0.004846807759666541	0.000044
2	111.15389977177607	197.104918	116.324631	0.0019680369258500167	0.000044
3	212.09857905298142	197.104918	116.324631	0.0031189223175170765	0.000044
4	232.57400286508445	197.104918	116.324631	0.0020022482298386506	0.000044

5 rows × 22 columns

In [40]:
```
dfsample.dtypes
```

Out[40]:
```
MDVP:Fo(Hz)          object
MDVP:Fhi(Hz)        float64
MDVP:Flo(Hz)        float64
MDVP:Jitter(%)       object
MDVP:Jitter(Abs)    float64
MDVP:RAP             object
MDVP:PPQ             object
Jitter:DDP           object
MDVP:Shimmer         object
MDVP:Shimmer(dB)     object
Shimmer:APQ3         object
Shimmer:APQ5         object
MDVP:APQ             object
Shimmer:DDA          object
NHR                 float64
HNR                  object
RPDE                float64
DFA                 float64
spread1             float64
spread2             float64
D2                  float64
PPE                 float64
dtype: object
```

In [41]:
```
pred1 = LR.predict(dfsample)
```

In [42]:
```
pred1
```

Out[42]:
```
array([0, 0, 0, 0, 0, 0], dtype=int64)
```

In [43]:
```
for t in pred1:#our output at runtime
    if t==0:
        print(' not parkinson')
    else:
        print('parkinson')
```

```
 not parkinson
 not parkinson
 not parkinson
 not parkinson
 not parkinson
 not parkinson
```

Type *Markdown* and LaTeX: α^2

CONCLUSION

- The proposed model could be the first indication in detection of Parkinson.
- Hassle free detection
- Using Machine Learning we can apply a proactive approach towards detection.

Moreover…

- Real Time Voice Analysis for Parkinson Detection.
- This could be achieved for feature extraction using python library parselmouth and voice analysis software praat.

FURTHER DEVELOPMENT

- Interfacing project with : -

  - Website
  - Application

- Applicability of our project can be increased by taking into account other symptoms.
- We would be starting campaigns using social media, to increase awareness about Parkinson.

Hardware used:

- Personal Computer
- Laptop

Software Used: (language/library)

- For Data Analysis:
1. Pandas
2. NumPy
3. Matplotlib.pyplot

For Machine Learning:
Scikit-Learn

PYTHON LIBRARIES

- **NumPy** : It's a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **SkLearn** : A free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, k-means etc. and is designed to interoperate with NumPy and SciPy.
- **Keras** : Its a open source neural network library written in Python. Capable of running on top of TensorFlow. Designed to enable fast experimentation with deep neural networks, its pros being user-friendly, modular, and extensible.
- **Matplotlib** : A plotting library for the Python programming language and its numerical mathematics extension.

PYTHON LIBRARIES

- **Os:** provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality.
- **OpenCV:** OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc.
- **Pandas:** Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks.
- **Parselmouth:** Audio processing library

REFERENCES

- https://www.nia.nih.gov/health/parkinsons-disease
- https://www.parkinson.org/understanding-parkinsons/what-is-parkinsons
- https://www.kaggle.com/kmader/parkinsons-drawings
- https://archive.ics.uci.edu/ml/datasets/parkinsons
- XGBoost: A Deep Dive into Boosting | by Rohan Harode | SFU Professional Master's Program in Computer Science | Medium
- Saunders-Pullman R, Derby C, Stanley K, Floyd A, Bressman S, Lipton RB, et al. Validity of spiral analysis in early Parkinson's disease. *Mov Disord* (2008) 23(4):531–7. doi:10.1002/mds.21874
- San Luciano M, Wang C, Ortega RA, Yu Q, Boschung S, Soto-Valencia J, et al. Digitized spiral drawing: a possible biomarker for early Parkinson's disease. *PLoS One* (2016) 11(10):e0162799. doi:10.1371/journal.pone.0162799
- Sisti JA, Christophe B, Seville AR, Garton AL, Gupta VP, Bandin AJ, et al. Computerized spiral analysis using the iPad. *J Neurosci Methods* (2017) 275:50–4.

THANK YOU

THANK YOU