PROJECT 2: DETECTING PARKINSON’S DISEASE

1. Introduction:

Parkinson’s is a disease of the nervous system disorder that affects movement. The symptoms gradually starts with barely noticeable tremors which are then progressed as showing little or no expression, loss of motor functions, rigid muscles, slow movements, and changes in handwriting or slurred speech, impaired posture and balance and more. The symptoms worsen as the condition progresses overtime.

Using Parkinson’s data, I have tried to build a model which uses the dataset provided and splits it into two, training and test set, in the ratio of 75:25 respectively, they are both used to determine the accuracy the model that is being generated and the data, this data is been acquired from UCI’s dataset, The following is the link of the dataset.

Dataset: https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/

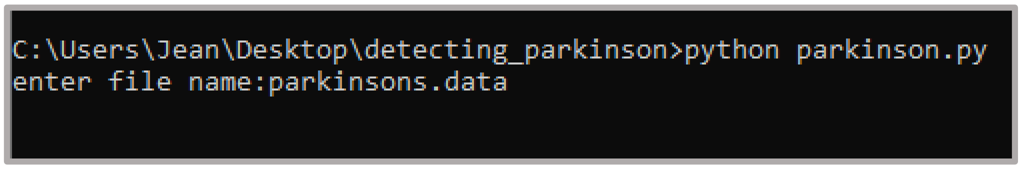
2. Technology Used:

To build this model I have used XGBoost which along with other libraries is been used to build this model.

XGBoost provides a wrapper class to allow models to be treated like classifiers or regressors in the scikit-learn framework.

This means we can use the full scikit-learn library with XGBoost models.

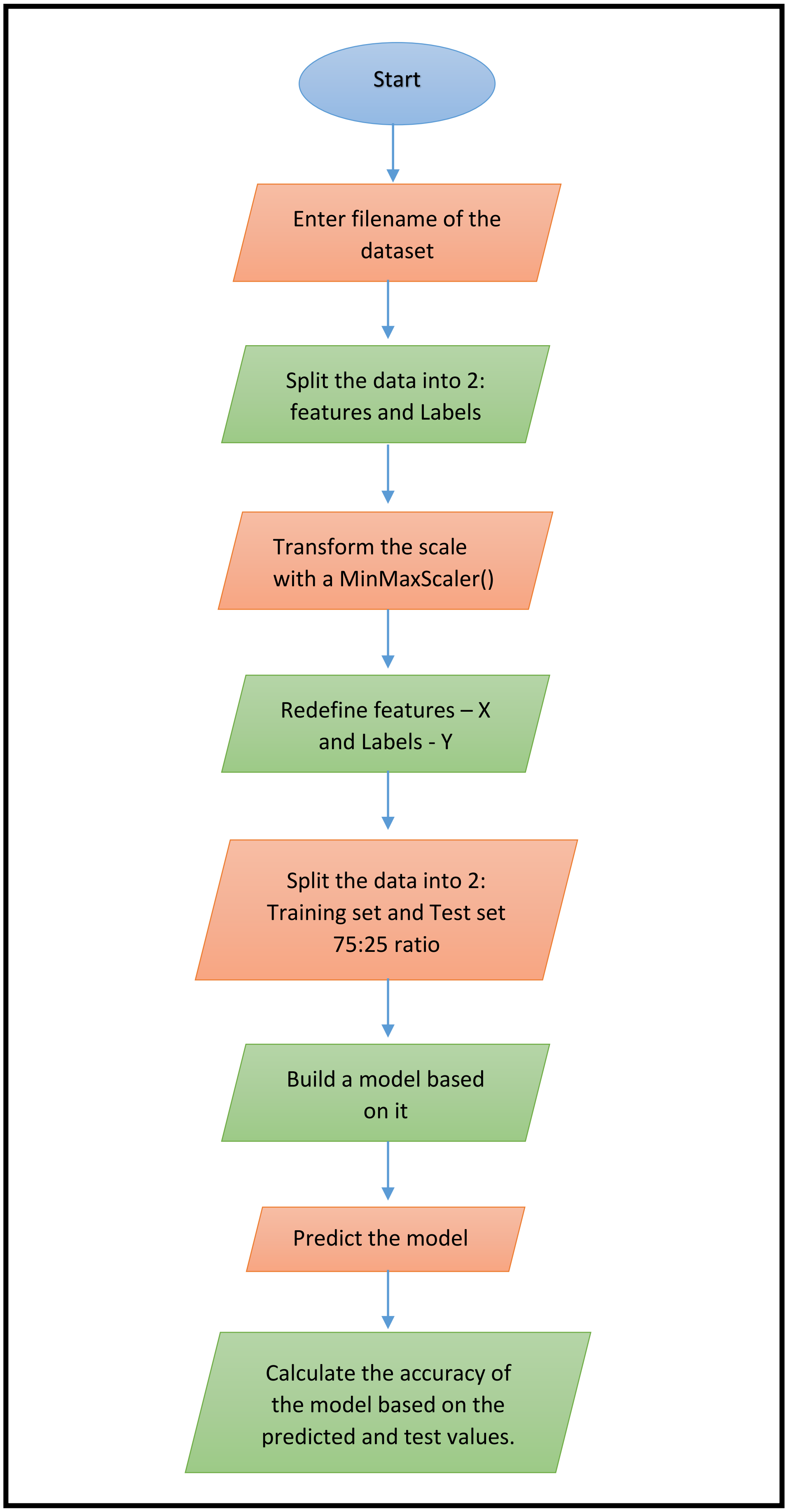
The XGBoost model for classification is called XGBClassifier. We can create and and fit it to our training dataset. Models are fit using the scikit-learn API and the model.fit() function.



```
C:\Users\Jean\Desktop\detecting_parkinson>python parkinson.py
enter file name:parkinsons.data
```

Fig.1: Initial display of program, asking for input dataset filename.

3. Flowchart:


Start
Enter filename of the dataset
Split the data into 2: features and Labels
Transform the scale with a MinMaxScaler()
Redefine features – X and Labels - Y
Split the data into 2: Training set and Test set 75:25 ratio
Build a model based on it
Predict the model
Calculate the accuracy of the model based on the predicted and test values.

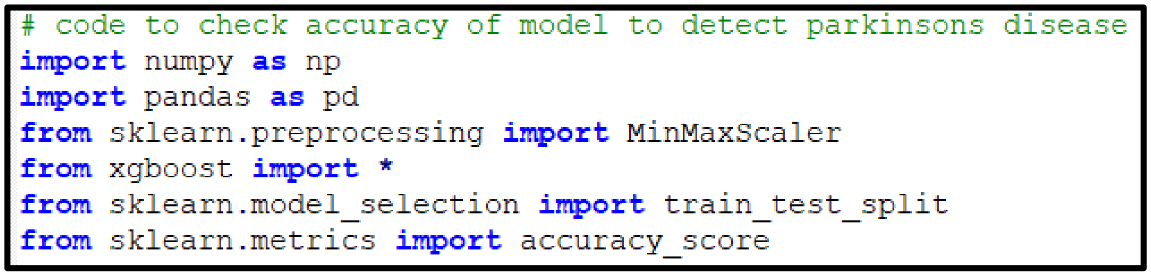
4. Code for the project:

https://github.com/mokshadashah/detecting_parkinsons

5. Explanation of the code:

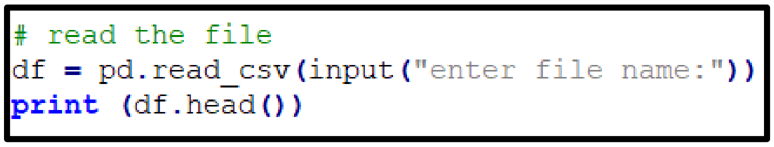
a. Importing Libraries:

Before running any program all the necessary libraries and modules should be imported into the program.

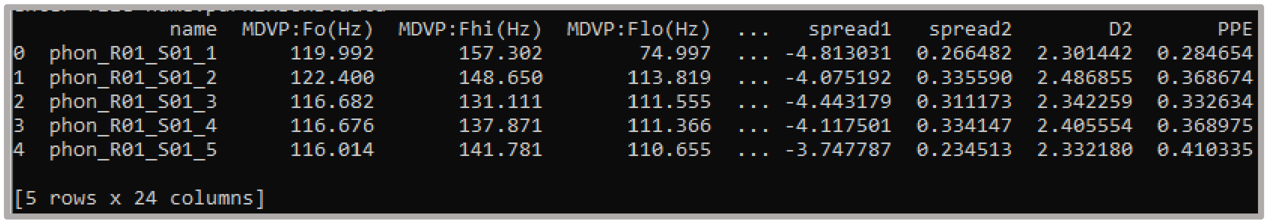


```python
# code to check accuracy of model to detect parkinsons disease
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from xgboost import *
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

b. Reading Parkinson's Data file:



```python
# read the file
df = pd.read_csv(input("enter file name:"))
print (df.head())
```



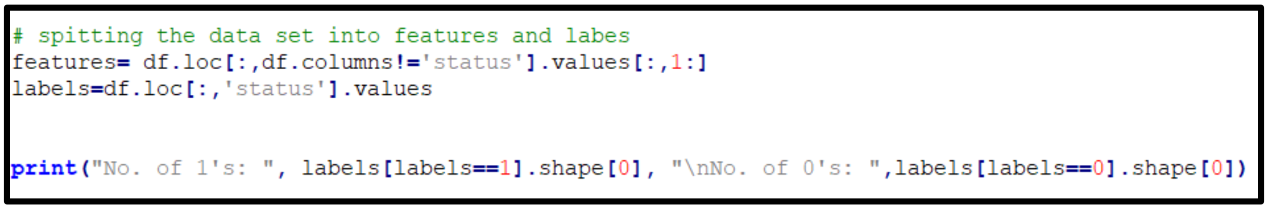
```
             name  MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  ...   spread1   spread2        D2       PPE
0  phon_R01_S01_1      119.992       157.302        74.997  ... -4.813031  0.266482  2.301442  0.284654
1  phon_R01_S01_2      122.400       148.650       113.819  ... -4.075192  0.335590  2.486855  0.368674
2  phon_R01_S01_3      116.682       131.111       111.555  ... -4.443179  0.311173  2.342259  0.332634
3  phon_R01_S01_4      116.676       137.871       111.366  ... -4.117501  0.334147  2.405554  0.368975
4  phon_R01_S01_5      116.014       141.781       110.655  ... -3.747787  0.234513  2.332180  0.410335

[5 rows x 24 columns]
```

Fig.2: Output of the head of the data file showing top 5 records

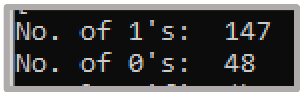
c. **Splitting the data into features and labels:**

Features includes all the data of the input data patterns and labels includes data of the output data patterns. The column status includes numbers 1 and 0 which indicates Parkinson's or not. Hence that should be the output data pattern.



```
# spitting the data set into features and labes
features= df.loc[:,df.columns!='status'].values[:,1:]
labels=df.loc[:,'status'].values


print("No. of 1's: ", labels[labels==1].shape[0], "\nNo. of 0's: ",labels[labels==0].shape[0])
```

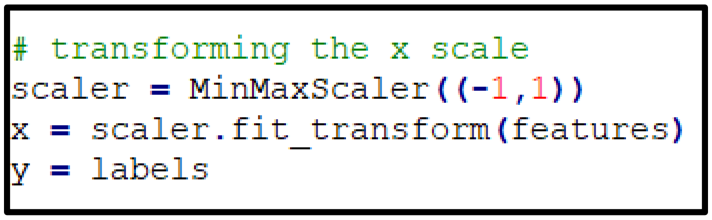


```
No. of 1's:  147
No. of 0's:  48
```

Fig.3: Record of 1's and 0's in labels variable

d. **Transform the X scale:**

We transform the x scale with features and reassign labels to variable y

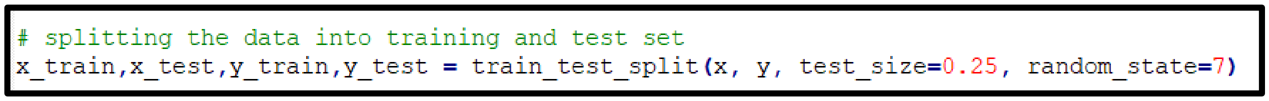


```
# transforming the x scale
scaler = MinMaxScaler((-1,1))
x = scaler.fit_transform(features)
y = labels
```

e. **Splitting data into training and test set:**

We split the X and Y data into a training and test dataset in the ration 75:25 respectively. The training set will be used to prepare the XGBoost model and the test set will be used to make new predictions, from which we can evaluate the performance of the model.

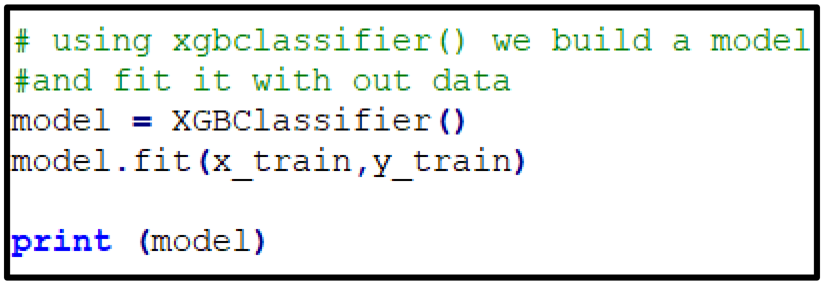
For this we will use the train_test_split() function from the scikit-learn library. We also specify a random state for the random number generator so that we always get the same split of data each time this example is executed.



```
# splitting the data into training and test set
x_train,x_test,y_train,y_test = train_test_split(x, y, test_size=0.25, random_state=7)
```

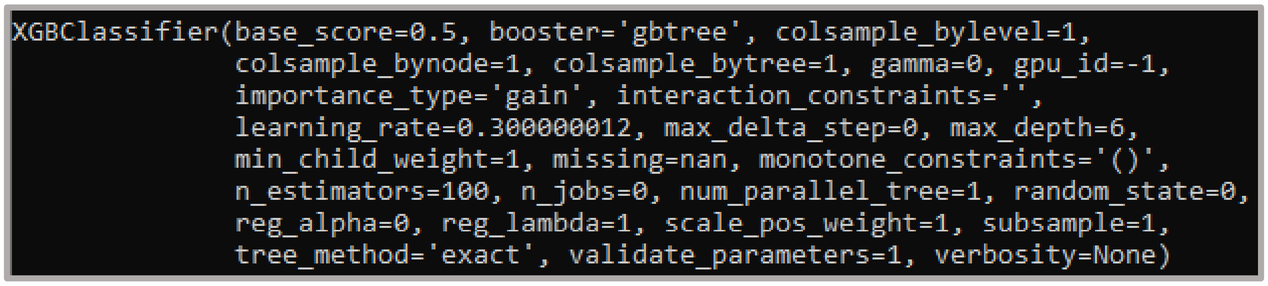
f. Build a model and train it with XGBClassifier():

The XGBoost model for classification is called XGBClassifier. We can create and and fit it to our training dataset. Models are fit using the scikit-learn API and the model.fit() function. Then we print the model to check the parameters, they can be changed later accordingly.



```
# using xgbclassifier() we build a model
#and fit it with out data
model = XGBClassifier()
model.fit(x_train,y_train)

print (model)
```

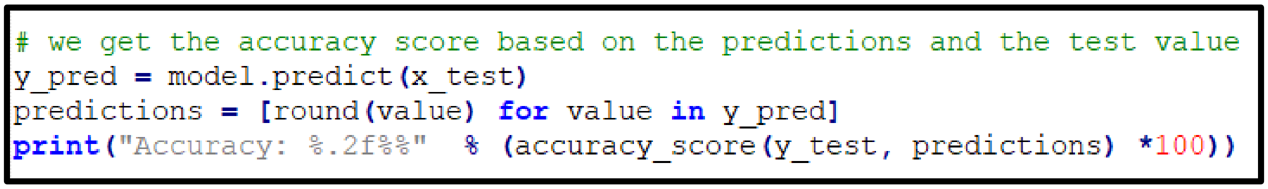


```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

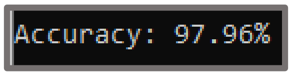
Fig.4: Output of the model with its default parameters

g. Make Predictions with XGBoost Model:

Using model.predict() we predict the Y values (output values) based on the X test values. Then we round the values and calculate the accuracy of the model based on the predicted value and the test values.



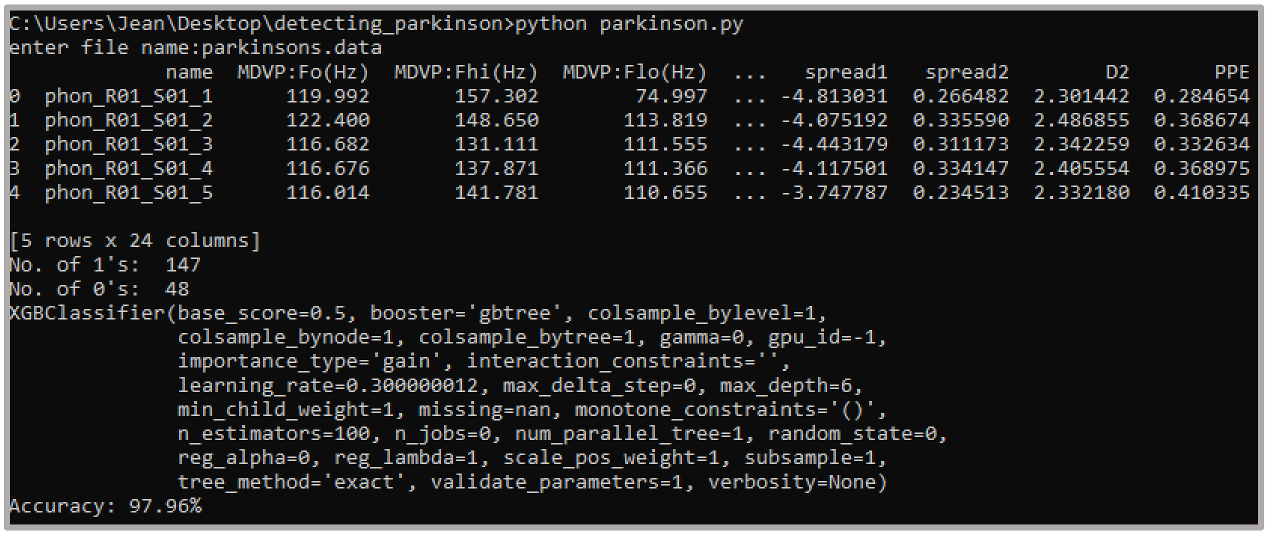
```
# we get the accuracy score based on the predictions and the test value
y_pred = model.predict(x_test)
predictions = [round(value) for value in y_pred]
print("Accuracy: %.2f%%"  % (accuracy_score(y_test, predictions) *100))
```



```
Accuracy: 97.96%
```

Fig.5: Output of the accuracy of the model

Final Output:



```
C:\Users\Jean\Desktop\detecting_parkinson>python parkinson.py
enter file name:parkinsons.data
             name  MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  ...   spread1   spread2        D2       PPE
0  phon_R01_S01_1      119.992       157.302        74.997  ... -4.813031  0.266482  2.301442  0.284654
1  phon_R01_S01_2      122.400       148.650       113.819  ... -4.075192  0.335590  2.486855  0.368674
2  phon_R01_S01_3      116.682       131.111       111.555  ... -4.443179  0.311173  2.342259  0.332634
3  phon_R01_S01_4      116.676       137.871       111.366  ... -4.117501  0.334147  2.405554  0.368975
4  phon_R01_S01_5      116.014       141.781       110.655  ... -3.747787  0.234513  2.332180  0.410335

[5 rows x 24 columns]
No. of 1's:  147
No. of 0's:  48
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
Accuracy: 97.96%
```