PROJECT REPORT

On

"Detecting Parkinson's Disease"

Submitted in partial fulfilment of the requirements for the award of

Bachelor of Technology (B.Tech)

In the departments of

Computer Science & Engineering,

Electronics & Communication Engineering and

Electrical Engineering

Submitted by:

Arindom Sharma (ET17BTHCS010)

Amrit Kr. Baruah (ET17BTHCS007)

Aditya Chakraborty (ET17BTHCS004)

Popee Borah (ET18BTHEC011L)

Jili Tali (ET18BTHEE017L)

Under the Guidance of

Internal Mentor – **Mrs. Mousoomi Borah**
(H.O.D CSE Department, KU)

External Mentor-
Mr. Viqaruddin Surki
(IBM)

External Mentor-
Mr. Amit Bhat
(IBM)

School of Engineering & Technology
The Assam Kaziranga University, Jorhat, Assam
Dec- 2020

CERTIFICATE

This is to certify that the project report entitled **"Detecting Parkinson's Disease"**, submitted to the School of Engineering & Technology (SET), **KAZIRANGA UNIVERSITY, JORHAT** in partial fulfilment for the completion of **Semester – 7th** of the degree of **Bachelor of Technology** in the department of **Computer Science & Engineering**, **Electrical Engineering** and **Electronics & Communication Engineering** is a record of bonafide work carried out by **Mr. Arindom Sharma**, **Roll No**. **ET17BTHCS010, Ms. Popee Borah, Roll No. ET18BTHEC011L, Mr. Amrit Kr. Baruah, Roll No. ET17BTHCS007, Mr. Aditya Chakraborty, Roll No. ET17BTHCS004, Mr. Jili Tali, Roll No. ET18BTHEE017L**, under my guidance.

All help received by us from various sources have been duly acknowledged.
No part of this report has been submitted elsewhere for award of any other degree.

Internal Mentor - Mrs. Mousoomi Borah

(H.O.D CSE Department)

External Mentor –
Mr. Viqaruddin Surki
(IBM)

External Mentor –
Mr. Amit Bhatt
(IBM)

ACKNOWLEDGEMENT

The Project opportunity we had with **IBM** at **Kaziranga University, Jorhat** was a great chance for learning and gaining practical knowledge. Therefore, we consider ourselves very lucky as we were provided with an opportunity to be a part of it.

Our indebtedness and gratitude is towards those individuals who have helped us shape this report. We must record our immense gratitude to those who have helped us to learn.

We would like to thank **Mrs. Mousoomi Borah (H.O.D CSE Department, Internal Mentor), Mr. Amit Bhatt** and **Mr. Viqaruddin Surki (External Mentor),** under whose guidance we have completed our project. All these people were of immense importance regarding the knowledge and support for the well-furnished equipment. We greatly acknowledge the help and the mental strength provided by entire **KU SET** and **IBM** family for encouraging and providing knowledge & guidance related with our project.

Declaration

We, the undersigned, declare that the project entitled ‘Detecting Parkinson', being submitted in partial fulfillment for the award of Bachelor of Engineering Degree in Computer Science & Engineering, Electronics and Communication Engineering and Electrical Engineering, affiliated to Kaziranga University, is the work carried out by us.

Arindom Sharma
(ET17BTHCS010)

Amrit Kr. Baruah
(ET17BTHCS007)

Aditya Chakraborty
(ET17BTHCS004)

Popee Borah
(ET18BTHEC011L)

Jili Tali
(ET18BTHEE017L)

ABSTRACT

Biomarkers derived from human voice can offer insight into neurological disorders, such as Parkinson's disease (PD), because of their underlying cognitive and neuromuscular function. PD is a progressive neurodegenerative disorder that affects about one million people in India, with approximately sixty thousand new clinical diagnoses made each year. Historically, PD has been difficult to quantify and doctors have tended to focus on some symptoms while ignoring others, relying primarily on subjective rating scales. Due to the decrease in motor control that is the hallmark of the disease, voice can be used as a means to detect and diagnose PD. With advancements in technology and the prevalence of audio collecting devices in daily lives, reliable models that can translate this audio data into a diagnostic tool for healthcare professionals would potentially provide diagnoses that are cheaper and more accurate. We provide evidence to validate this concept here using a voice dataset collected from people with and without PD. This paper explores the effectiveness of using supervised classification algorithms, such as Xtreme Gradient Boosting (xgboost),which is a new algorithm for Machine Learning developed with speed and efficiency in mind to accurately diagnose individuals with the disease. Our peak accuracy of 92.3077% provided by the machine learning model with RMSE of 0.277350 which is quite good. Also on computing the k-fold cross validation the model gives Average Test RMSE of 0.281145.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Background

Parkinson's disease is a progressive disorder of the central nervous system, characterized by the progressive degeneration of the structure and function of the nervous system. They are incurable and debilitating conditions that cause problems with mental functioning of an individual.

Parkinson's disease affect millions of people worldwide. It affects more than 1 million people in India per year. An estimated 930,000 people in the United States could be living with Parkinson's disease by 2020.

Parkinson's disease (PD) is a neuropathological disorder which deteriorates the motor functions of the human body. It is the second most common neurological disease seen after Alzheimer's disease and it is estimated that more than one million people are suffering from PD in North America alone. In 1817, PD was termed as shaking palsy by Dr. James Parkinson. Various studies have shown that this number will rise in an ageing population as it is commonly seen in the people whose age is over 60.

Parkinson's disease is characterized by the degeneration of certain brain cell clusters that are responsible for producing the neurotransmitters that include dopamine, serotonin and acetylcholine. The loss of dopamine's result in the symptoms like anxiety, depression, weight loss and visual problems. The other symptoms that can be seen in the people with Parkinson's disease are poor balance, voice impairment and tremor. Various research studies have shown that 90% of people who suffer from PD have speech and vocal problems which include dysphonia, monotone and hypophonia. Thus, the degradation of voice is considered to be as the initial symptom of Parkinson's disease.

The cause and cure of PD are yet unknown but the availability of various drug therapies offers the significant mitigation of symptoms especially at its earlier stages, thus improving the life quality of patients and also reduces the estimated cost of the Pathology. The analysis of voice measurement is simple and non-invasive. Thus, to track the progression of PD the measurement of voice can be used. For assessing the progression of PD, various vocal tests have been devised which include sustained phonations and running speech texts. The telemonitoring and telediagnosis systems have been widely used as these systems are based on speech signals which are economical and easy to use. Hence, in this paper, there is an attempt to explore a better machine learning based model for an early detection of PD from the voice samples of the subject.

1.2 Purpose of the Project

Early detection of a Parkinson's disease could be useful for the identification of people who can participate in trials of its agents, or ultimately to try and halt disease progression once effective disease-modifying interventions have been identified.

1.1 Problem Statement

The goal of this project is to build a model to accurately predict the presence of Parkinson's disease in an individual.

1.2 Objective

This paper aims to build a model using an XGBClassifier that accurately predicts the presence of Parkinson's disease in an individual using the mentioned dataset. We will use the python libraries; scikit-learn, numpy, pandas, and xgboost. We'll load the data, explore the data, get the features and labels, then split the dataset, build an XGBClassifier, and then calculate the accuracy and the RMSE of our model. And perform k-fold cross validation to make our model more robust. At last we will calculate the feature importance on our predictive modeling problem.

1.3 Structure of thesis

The outline of this thesis is shown as follows -

In Chapter 2, Literature Reviews of some of the previous related studies are provided.
In Chapter 3, Information on Machine Learning Algorithm used.
In Chapter 4, Information on Dataset used
In Chapter 5, Software requirements will be provided.
In Chapter 6, Methodology will be discussed.
In Chapter 6, Results will be provided.
In Chapter 7, The conclusion and recommendation will be provided.

CHAPTER 2
LITERATURE REVIEW

2.1 Literature Reviews of some of the previous related studies

- Richa Mathur et al [23] suggested a method for predicting the PD. They used a weka tool for implementing the algorithms to perform preprocessing of data, classification and the result analysis on the given dataset. They used k-NN along with Adaboost.M1, bagging, and MLP. It was observed that k-NN + Adaboost.M1 yielded the best classification accuracy of 91.28%.

- A.Yasar et al [24] used artificial neural networksfor the detection of Parkinson's disease. The dataset was taken from UCI machine learning repository. Using the MATLAB tool, 45 properties were chosen as input values and one output for the classification. Their proposed model was able to distinguish the healthy subjects from the PD subjects with an accuracy of 94.93%.

- Max A. little et al [15] suggested a novel technique for the classification of subjects into Parkinson diseased and control subjects by detecting dysphonia. In their work, pitch period entropy (PPE) a new robust measure of dysphonia was introduced. The data was collected from 31 people (23 were PD patients and 8 were healthy subjects) which comprised of 195 sustained vowel phonations. Their methodology consisted of three stages; feature calculation, preprocessing and selection of features and finally the classification. For the classification purpose, they used linear kernel support vector machine (SVM). Their proposed model achieved an accuracy of 91.4%.

- To separate the healthy subjects from PD subjects, Ipsita Bhattacharya et al [20] used a tool for data mining known as weka. They used SVM, a supervised machine learning algorithm for the classification purpose. Prior to classification, the data preprocessing was done on the dataset. Different kernel values were used to get the best possible accuracy by applying libSVM. The linear kernel SVM produced the best accuracy of 65.2174%, whereas the RBF kernel and polykernel SVM achieved the accuracy of 60.8696%

CHAPTER 3
ML ALGORITHM

3.1. What is Machine Learning?

Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people.

Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs.

In machine learning, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed.

Two of the most widely adopted machine learning methods are: -

• **Supervised learning** which trains algorithms based on example input and output data that is labeled by humans.

• **Unsupervised learning** provides the algorithm with no labeled data in order to allow it to find structure within its input data.

3.2. Algorithm used- XGBoost?

XGBoost is a new Machine Learning algorithm designed with speed and performance in mind. XGBoost stands for eXtreme Gradient Boosting and is based on decision trees.

3.2.1. How XGBoost Algorithm works?

- XGBoost builds really short and simple decision trees iteratively.
- XGBoost starts by creating a first simple tree which has poor performance by itself.
- It then builds another tree which is trained to predict what the first tree was not able to, and is itself a weak learner too.
- The algorithm goes on by sequentially building more weak learners, each one correcting and reduce the errors of the previous tree until a stopping condition is reached.

Here's a popular graphic from the XGBoost website as an example (Fig. 3.1):

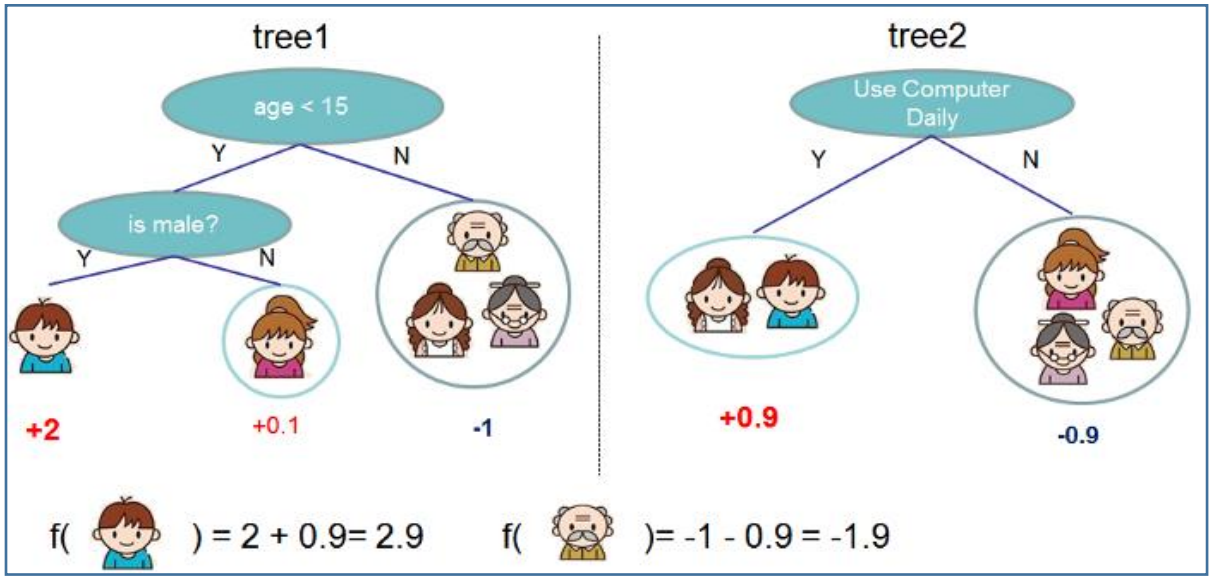

Fig. 3.1: XGBoost example

3.2.2. Advantages of XGBoost:

- **Better Speed and performance**: XGBoost is comparatively faster and it has shown better performance over other algorithms on a variety of machine learning benchmark datasets.

- **Regularization**: Standard GBM implementation has no regularization like XGBoost, therefore it also helps to reduce overfitting.

- **Parallel Processing:** 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.

- **Handling Missing Values:** XGBoost has an in-built capability to handle missing values

CHAPTER 4
DATASET

4.1. About the Dataset:

The dataset was created by Max Little of the University of Oxford, in collaboration with the National Centre for Voice and Speech, Denver, Colorado, who recorded the speech signals. The original study published the feature extraction methods for general voice disorders.

From existing metadata, we get some important information, such as:

- Title: Parkinsons Disease Data Set
- Abstract: Oxford Parkinson's Disease Detection Dataset
- Data Set Characteristics: Multivariate
- Number of Instances: 197
- Number of Attributes: 23

4.2. 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.

4.3. 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

This dataset had been downloaded from UCI ML repository.

CHAPTER 5
SOFTWARE REQUIREMENTS

5.1. Software Used:

1. Anaconda Navigator

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system *conda*. The Anaconda distribution is used by over 15 million users and includes more than 1500 popular data-science packages suitable for Windows, Linux, and macOS.

We can download this from the official website of Anaconda Navigator. We will need Jupiter Notebook, which we can directly use through Anaconda Navigator

2. Python 3.7 (64-bit)

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It is high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

1. Open a browser window and navigate to the Download page for Windows at python.org.

2. Underneath the heading at the top that says Python Releases for Windows, click on the link for the Latest Python 3 Release - Python 3.x.x. (As of this writing, the latest in Python 3.7.x)

3. Scroll to the bottom and select Windows x86-64 executable installer for 64-bit. We strongly recommend 64-bit because TensorFlow doesn’t support 32-bit.

4. Once you have chosen and downloaded an installer, simply run it by double-clicking on the downloaded file.

5. Then just click Install Now. That should be all there is to it. A few minutes later you should have a working Python 3 installation on your system.

5.2. Libraries Used:

1. NumPy:

To install this package with conda run the following:

conda install -c conda-forge numpy

To install this package with python, run the following:

pip install numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

2. Sklearn:

To install this package with conda run one of the following:

conda install -c anaconda scikit-learn

To install this package with python, run the following:

pip install scikit-learn

Scikit-learn (formerly scikits.learn and also known as sklearn) is 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, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

3. Pandas:

To install this package with conda run one of the following:

conda install -c anaconda pandas

To install this package with python, run the following:

pip install pandas

Pandas offer data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

4. Matplotlib:

To install this package with conda run one of the following:

conda install -c conda-forge matplotlib
conda install -c conda-forge/label/testing matplotlib
conda install -c conda-forge/label/testing/gcc7 matplotlib
conda install -c conda-forge/label/gcc7 matplotlib
conda install -c conda-forge/label/broken matplotlib
conda install -c conda-forge/label/rc matplotlib
conda install -c conda-forge/label/cf201901 matplotlib

To install this package with python, run the following:

pip install matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

5. Seaborn:

Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

To install this package with conda run one of the following:

conda install seaborn

To install this package with python, run the following:

pip install seaborn

\

CHAPTER 6
METHODOLOGY

6.1. Workflow Diagram:

The methodology for building a model to detect the Parkinson's disease at its early stage using the machine learning algorithms is presented in figure5.3. It consists of the following steps:

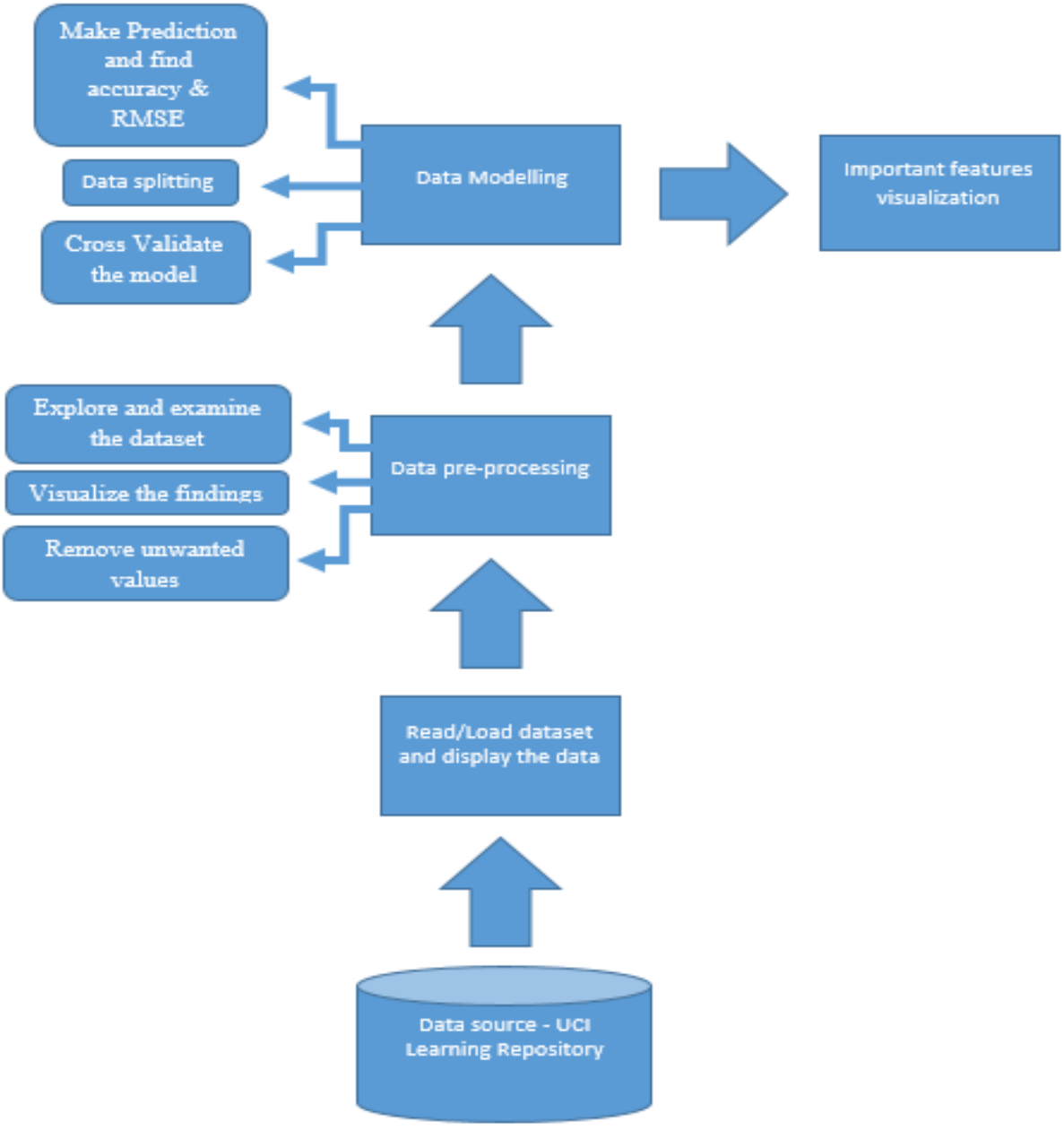

Fig. 5.3: Workflow Diagram

Explanation of the different stages depicted in the workflow diagram (Fig. 2) are as follows:

1. **Acquiring the dataset**

- We have download the voice sample dataset from UCI Learning Repository and stored in our PC.

2. **Read/Load the dataset**

Now we will load the dataset from our PC and display it in our code using Pandas data-frame.

3. **Data Pre-processing**

Here we will explore and examine our dataset (using .info, .corr, .describe methods), remove unwanted values and then visualize our findings using heat-map and bar-chart.

4. **Data Modelling**

Within this step we will split our data into training and testing part using sklearn library and we will do our prediction using the XGBoost algorithm and find out the Accuracy and Root Mean Square Error of the model's prediction after that we will again use k-fold cross validation in order to make our model more robust and again find out the mean of the Accuracy and mean Root Mean Square Error of the result of validation.

5. **Important feature visualization using XGboost**

Finally, we will visualize and find out the feature that has the highest importance among all the features

6.2 Procedure:

1. **Import the dataframe using Pandas. After that Load/Read the Dataset and print the first 5 rows.**

```
In [46]: import pandas as pd

         df = pd.read_csv(r"C:\Users\Arindom\Downloads\parkinsons (7).data")
         df.head()
```

Out[46]:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...	Shi
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

Fig 6.1

DATA-PREPROCESSING

2. **Check for its shape by using the ".shape" attribute, which will return the size of the dataset i.e the number of rows and columns.**

```
In [47]: #return the size of the dataset
         df.shape

Out[47]: (195, 24)
```

Fig 6.2

As you can see it returned (195, 24), that means there are 195 rows of data with 24 columns.

3. **Using ".info()" method to examine each column data type and possible missing data.**

```
In [48]: #examine each column
         df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
name              195 non-null object
MDVP:Fo(Hz)       195 non-null float64
MDVP:Fhi(Hz)      195 non-null float64
MDVP:Flo(Hz)      195 non-null float64
MDVP:Jitter(%)    195 non-null float64
MDVP:Jitter(Abs)  195 non-null float64
MDVP:RAP          195 non-null float64
MDVP:PPQ          195 non-null float64
Jitter:DDP        195 non-null float64
MDVP:Shimmer      195 non-null float64
MDVP:Shimmer(dB)  195 non-null float64
Shimmer:APQ3      195 non-null float64
Shimmer:APQ5      195 non-null float64
MDVP:APQ          195 non-null float64
Shimmer:DDA       195 non-null float64
NHR               195 non-null float64
HNR               195 non-null float64
status            195 non-null int64
RPDE              195 non-null float64
DFA               195 non-null float64
spread1           195 non-null float64
spread2           195 non-null float64
D2                195 non-null float64
PPE               195 non-null float64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.6+ KB
```

Fig 6.3

4. **Using .describe() method to see data summary statictic, such as min, median, max, and so on.**

```
In [49]: #data summary statictic
         df.describe()
```

Out[49]:

	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)
count	195.000000	195.000000	195.000000	195.000000	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	0.003446	0.009920	0.029709	0.282251
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	0.002759	0.008903	0.018857	0.194877
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680	0.000920	0.002040	0.009540	0.085000
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660	0.001860	0.004985	0.016505	0.148500
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500	0.002690	0.007490	0.022970	0.221000
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835	0.003955	0.011505	0.037885	0.350000
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440	0.019580	0.064330	0.119080	1.302000

8 rows × 23 columns

Fig 6.4

We use describe to disable non-truncated(not comparable) dataframe. Hence, here 'name' column is gone as it is of datatype object and is not comparable with other dataframe of int and float.

5. **Use ".corr()" to see if there is a correlation between a pair of feature.**

```
In [50]: #correlation between a pair of feature
         df.corr()
```

Out[50]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:S
MDVP:Fo(Hz)	1.000000	0.400985	0.596546	-0.118003	-0.382027	-0.076194	-0.112165	-0.076213	-0.098374	
MDVP:Fhi(Hz)	0.400985	1.000000	0.084951	0.102086	-0.029198	0.097177	0.091126	0.097150	0.002281	
MDVP:Flo(Hz)	0.596546	0.084951	1.000000	-0.139919	-0.277815	-0.100519	-0.095828	-0.100488	-0.144543	
MDVP:Jitter(%)	-0.118003	0.102086	-0.139919	1.000000	0.935714	0.990276	0.974256	0.990276	0.769063	
MDVP:Jitter(Abs)	-0.382027	-0.029198	-0.277815	0.935714	1.000000	0.922911	0.897778	0.922913	0.703322	
MDVP:RAP	-0.076194	0.097177	-0.100519	0.990276	0.922911	1.000000	0.957317	1.000000	0.759581	
MDVP:PPQ	-0.112165	0.091126	-0.095828	0.974256	0.897778	0.957317	1.000000	0.957319	0.797826	
Jitter:DDP	-0.076213	0.097150	-0.100488	0.990276	0.922913	1.000000	0.957319	1.000000	0.759555	
MDVP:Shimmer	-0.098374	0.002281	-0.144543	0.769063	0.703322	0.759581	0.797826	0.759555	1.000000	
MDVP:Shimmer(dB)	-0.073742	0.043465	-0.119089	0.804289	0.716601	0.790652	0.839239	0.790621	0.987258	
Shimmer:APQ3	-0.094717	-0.003743	-0.150747	0.746625	0.697153	0.744912	0.783580	0.744894	0.987625	
Shimmer:APQ5	-0.070682	-0.009997	-0.101095	0.725561	0.648961	0.709927	0.786780	0.709907	0.982835	
MDVP:APQ	-0.077774	0.004937	-0.107293	0.758255	0.648793	0.737455	0.804139	0.737439	0.950083	
Shimmer:DDA	-0.094732	-0.003733	-0.150737	0.746635	0.697170	0.744919	0.783592	0.744901	0.987626	
NHR	-0.021981	0.163766	-0.108670	0.906959	0.834972	0.919521	0.844604	0.919548	0.722194	
HNR	0.059144	-0.024893	0.210851	-0.728165	-0.656810	-0.721543	-0.731510	-0.721494	-0.835271	
status	-0.383535	-0.166136	-0.380200	0.278220	0.338653	0.266668	0.288698	0.266646	0.367430	
RPDE	-0.383894	-0.112404	-0.400143	0.360673	0.441839	0.342140	0.333274	0.342079	0.447424	
DFA	-0.446013	-0.343097	-0.050406	0.098572	0.175036	0.064083	0.196301	0.064026	0.159954	
spread1	-0.413738	-0.076658	-0.394857	0.693577	0.735779	0.648328	0.716489	0.648328	0.654734	
spread2	-0.249450	-0.002954	-0.243829	0.385123	0.388543	0.324407	0.407605	0.324377	0.452025	
D2	0.177980	0.176323	-0.100629	0.433434	0.310694	0.426605	0.412524	0.426556	0.507088	
PPE	-0.372356	-0.086543	-0.340071	0.721543	0.748162	0.670999	0.769647	0.671005	0.693771	

23 rows × 23 columns

Fig 6.5

6. **Then we draw the heatmap to easily examine the result of correlation in the previous**

```
In [51]: import matplotlib.pyplot as plt
         import seaborn as sns

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

Fig 6.6

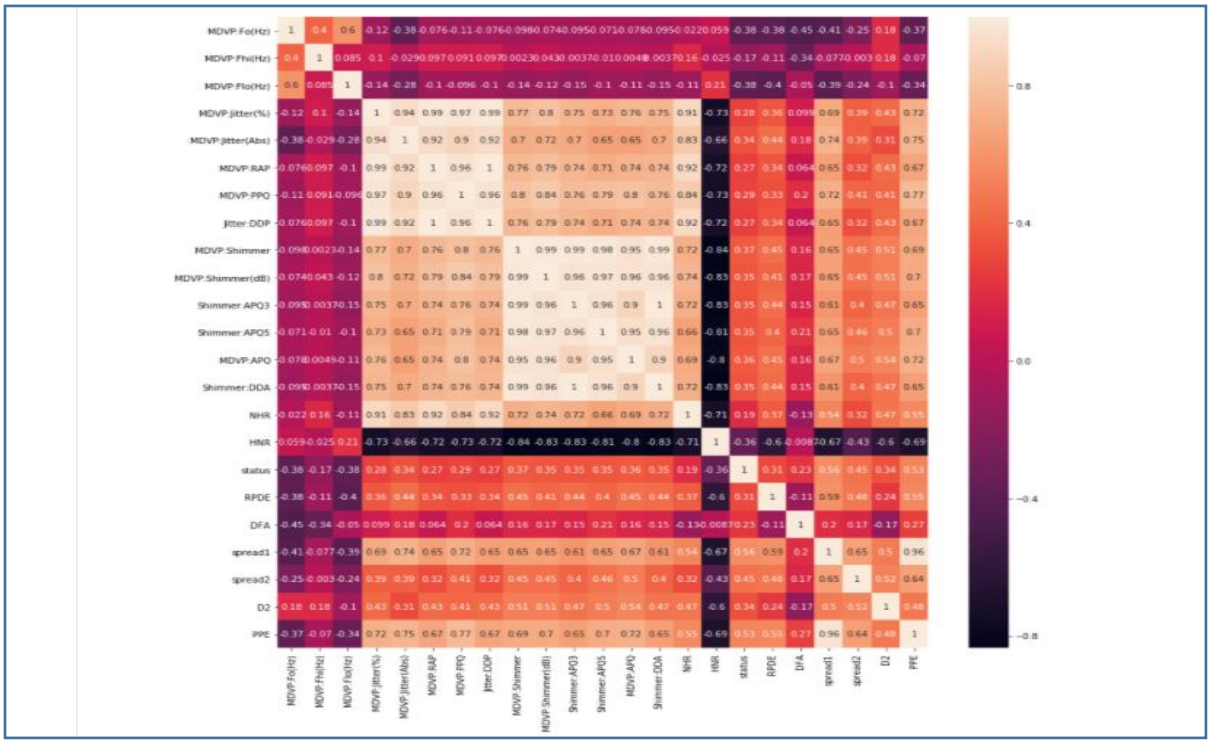

Fig 6.7

7. Count number of active PD patients (1) and inactive patients (0)

```
In [52]: df['status'].value_counts()

Out[52]: 1    147
         0     48
         Name: status, dtype: int64
```

Fig 6.8

8. Plot status column using bar chart.

```
In [53]: import seaborn as sns

         sns.countplot(df['status'],label="Count")

Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x7c79c7ada0>
```

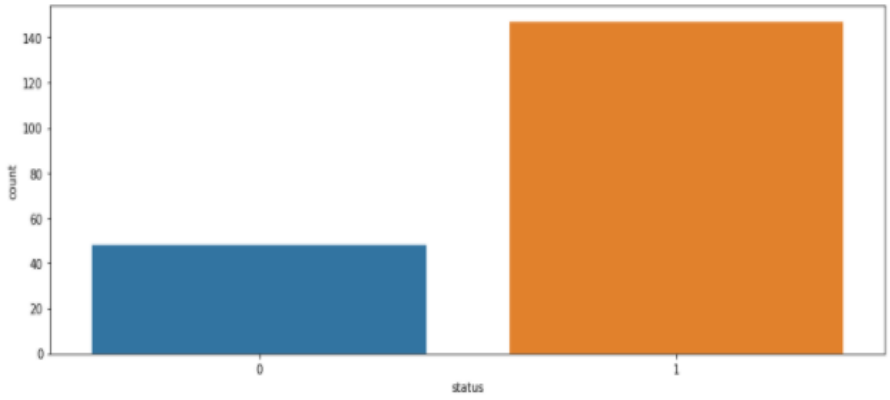

Fig 6.9

9. Since the status column is located in the middle of dataset, we need to move it to the far right, so we can easily slice the dataset.

```
In [54]: cols = list(df)
         cols.insert(24, cols.pop(cols.index('status')))

         cols

Out[54]: ['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',
          'RPDE',
          'DFA',
          'spread1',
          'spread2',
          'D2',
          'PPE',
          'status']
```

Fig 6.10

10. Display all rows and columns after moving 'status' column to extreme right.

```
In [55]: #Display all rows and columns after moving status to extreme right
         df = df.loc[:, cols]
         df

Out[55]:
```

s)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...	Shimmer:DDA	NHR	HNR	RPDE	DFA	spread1	spread2	D2	PPE	status
07	0.00370	0.00554	0.01109	0.04374	...	0.06545	0.02211	21.033	0.414783	0.815285	-4.813031	0.266482	2.301442	0.284654	1
08	0.00465	0.00696	0.01394	0.06134	...	0.09403	0.01929	19.085	0.458359	0.819521	-4.075192	0.335590	2.486855	0.368674	1
09	0.00544	0.00781	0.01633	0.05233	...	0.08270	0.01309	20.651	0.429895	0.825288	-4.443179	0.311173	2.342259	0.332634	1
09	0.00502	0.00698	0.01505	0.05492	...	0.08771	0.01353	20.644	0.434969	0.819235	-4.117501	0.334147	2.405554	0.368975	1
11	0.00655	0.00908	0.01966	0.06425	...	0.10470	0.01767	19.649	0.417356	0.823484	-3.747787	0.234513	2.332180	0.410335	1
08	0.00463	0.00750	0.01388	0.04701	...	0.06985	0.01222	21.378	0.415564	0.825069	-4.242867	0.299111	2.187560	0.357775	1
03	0.00155	0.00202	0.00466	0.01608	...	0.02337	0.00607	24.886	0.596040	0.764112	-5.634322	0.257682	1.854785	0.211756	1
03	0.00144	0.00182	0.00431	0.01567	...	0.02487	0.00344	26.892	0.637420	0.763262	-6.167603	0.183721	2.064693	0.163755	1
06	0.00293	0.00332	0.00880	0.02093	...	0.03218	0.01070	21.812	0.615551	0.773587	-5.498678	0.327769	2.322511	0.231571	1
06	0.00268	0.00332	0.00803	0.02838	...	0.04324	0.01022	21.862	0.547037	0.798463	-5.011879	0.325996	2.432792	0.271362	1
06	0.00254	0.00330	0.00763	0.02143	...	0.03237	0.01166	21.118	0.611137	0.776156	-5.249770	0.391002	2.407313	0.249740	1
06	0.00281	0.00336	0.00844	0.02752	...	0.04272	0.01141	21.414	0.583390	0.792520	-4.960234	0.363566	2.642476	0.275931	1
02	0.00118	0.00153	0.00355	0.01259	...	0.01968	0.00581	25.703	0.460600	0.646846	-6.547148	0.152813	2.041277	0.138512	1
03	0.00165	0.00208	0.00496	0.01642	...	0.02184	0.01041	24.889	0.430166	0.665833	-5.660217	0.254989	2.519422	0.199889	1
02	0.00121	0.00149	0.00364	0.01828	...	0.03191	0.00609	24.922	0.474791	0.654027	-6.105098	0.203653	2.125618	0.170100	1
03	0.00157	0.00203	0.00471	0.01503	...	0.02316	0.00839	25.175	0.565924	0.658245	-5.340115	0.210185	2.205546	0.234589	1
04	0.00211	0.00292	0.00632	0.02047	...	0.02908	0.01859	22.333	0.567380	0.644692	-5.440040	0.239764	2.264501	0.218164	1
04	0.00284	0.00387	0.00853	0.03327	...	0.04322	0.02919	20.376	0.631099	0.605417	-2.931070	0.434326	3.007463	0.430788	1
05	0.00364	0.00432	0.01092	0.05517	...	0.07413	0.03160	17.280	0.665318	0.719467	-3.949079	0.357870	3.109010	0.377429	1
05	0.00372	0.00399	0.01116	0.03995	...	0.05164	0.03365	17.153	0.649554	0.686080	-4.554466	0.340176	2.856676	0.322111	1
05	0.00428	0.00450	0.01285	0.03810	...	0.05000	0.03871	17.536	0.660125	0.704087	-4.095442	0.262564	2.739710	0.365391	1
03	0.00232	0.00287	0.00696	0.04137	...	0.06062	0.01849	19.493	0.629017	0.698951	-5.186960	0.237622	2.557536	0.259765	1
03	0.00220	0.00247	0.00661	0.04351	...	0.06685	0.01280	22.468	0.619060	0.679834	-4.330956	0.262384	2.916777	0.285695	1
03	0.00221	0.00258	0.00663	0.04192	...	0.06562	0.01840	20.422	0.537264	0.686894	-5.248776	0.210279	2.547508	0.253556	1
05	0.00380	0.00390	0.01140	0.01659	...	0.02214	0.01778	23.831	0.397937	0.732479	-5.557447	0.220890	2.692176	0.215961	1
06	0.00316	0.00375	0.00948	0.03767	...	0.05197	0.02887	22.066	0.522746	0.737948	-5.571843	0.236853	2.846389	0.219514	1

Fig 6.11

DATA-MODELLING

11. Importing necessary libraries

```
In [56]: #importing necessary libraries
         import xgboost as xgb
         import pandas as pd
         import numpy as np
```

Fig 6.12

12. Separate the target variable and rest of the variables using .iloc to subset the data.

```
In [57]: X,y = df.iloc[:,1:-1],df.iloc[:,-1]
```

Fig 6.13

13. **Now, we will convert the dataset into an optimized data structure called Dmatrix that XGBoost supports and gives it acclaimed performance and efficiency gains.**

```
In [60]: data_dmatrix = xgb.DMatrix(data=X,label=y)
```

Fig 6.14

14. **We then proceed to split our data set into a training and testing set using the train_test_split functionality from the model_selection module. We take a test size of 20% and set the random state to 42 to ensure we're getting the same results.**

```
In [61]: #Splitting the dataset
         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_state=43)
```

Fig 6.15

15. **Printing out the rows and columns that are allocated to training features, training labels, testing features and testing labels respectively.**

```
In [62]: 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,)
```

Fig 6.16

16. **The next step is to instantiate an XGBoost regressor object by calling the XGBRegressor() class from the XGBoost library with the hyper-parameters passed as arguments.**

```
In [63]: # Instantiate the XGBRegressor: xg_reg
         xg_reg = xgb.XGBRegressor(objective ='binary:hinge', colsample_bytree = 0.3, learning_rate = 0.3,
                         max_depth = 5, alpha = 10, n_estimators = 10) #Learning_rate = 0.1
```

Fig 6.17

17. **Fit the regressor to the training set and make predictions on the test set using the familiar .fit() and .predict() methods.**

```
In [64]: xg_reg.fit(X_train,y_train)

         # Predict the labels of the test set: preds
         preds = xg_reg.predict(X_test)
```

Fig 6.18

18. **Compute the rmse by invoking the mean_sqaured_error function from sklearn's metrics module. And compute the accuracy by invoking the accuracy_score function from sklearn's metrics module.**

```
In [65]: # compute and print RMSE
         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, preds)*100))

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

Fig 6.19

19. **In order to build more robust models, it is common to do a k-fold cross validation where all the entries in the original training dataset are used for both training as well as validation. Also, each entry is used for validation just once. XGBoost supports k-fold cross validation via the cv() method. All you have to do is specify the nfolds parameter, which is the number of cross validation sets you want to build.**

    This time you will create a hyper-parameter dictionary params which holds all the hyper-parameters and their values as key-value pairs but will exclude the n_estimators from the hyper-parameter dictionary because you will use num_boost_rounds instead.

    You will use these parameters to build a 3-fold cross validation model by invoking XGBoost's cv() method and store the results in a cv_results DataFrame. Note that here you are using the Dmatrix object you created before.

    cv_results contains train and test RMSE metrics for each boosting round.

```
In [66]: # k-fold Cross Validation using XGBoost

         # Create the parameter dictionary: params
         params = {"objective":"binary:hinge",'colsample_bytree': 0.3,'learning_rate': 0.3,
                         'max_depth': 5, 'alpha': 10}

         # Perform cross-validation: cv_results
         cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=30,
                             num_boost_round=50,early_stopping_rounds=10,metrics="rmse", as_pandas=True, seed=43)

         # Print cv_results
         cv_results.head()
```

Out[66]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	0.496113	0.005460	0.460401	0.195016
1	0.496113	0.005460	0.460401	0.195016
2	0.496113	0.005460	0.460401	0.195016
3	0.427551	0.038619	0.413115	0.193059
4	0.386898	0.034045	0.358827	0.221406

Fig 6.20

20. Extract and print the final boosting round metric.

```
In [67]: #Final boosting round metric.
         c=(cv_results["test-rmse-mean"]).tail(1)
         print(c)

         10    0.281145
         Name: test-rmse-mean, dtype: float64
```

Fig 6.21

IMPORTANT FEATURE VISUALIZATION USING XGBOOST

21. By examining the importance of each feature column in the original dataset within the model. We can visualize XGBoost model. One simple way of doing this involves counting the number of times each feature is split on across all boosting rounds (trees) in the model, and then visualizing the result as a bar graph, with the features ordered according to how many times they appear. XGBoost has a plot_importance() function that allows you to do exactly this.

```
In [68]: xgb.plot_importance(xg_reg)
         plt.rcParams['figure.figsize'] = [14, 5]
         plt.show()
```

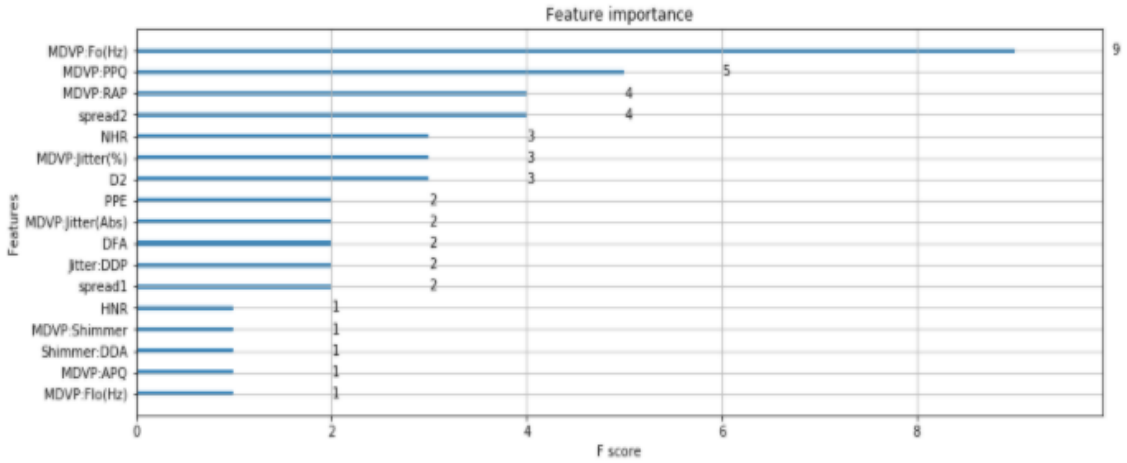

Fig 6.22

CHAPTER 7
RESULT ANALYSIS

1. **An accuracy of 92.3077% was provided by the machine learning model with Root Mean Square Error of 0.277350.**

```
In [65]: # compute and print RMSE
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, preds)*100))

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

Fig 7.1

2. **On computing the k-fold cross validation of the model. The model gives Average Test RMSE of 0.281145, which is very close to the RMSE given by our model initially.**

```
In [67]: #Final boosting round metric.
c=(cv_results["test-rmse-mean"]).tail(1)
print(c)

10    0.281145
Name: test-rmse-mean, dtype: float64
```

Fig 7.2

3. **Feature MDVP.Fo(Hz) (i.e. Average vocal fundamental frequency) has been given the highest importance score among all the features.**

```
In [68]: xgb.plot_importance(xg_reg)
plt.rcParams['figure.figsize'] = [14, 5]
plt.show()
```

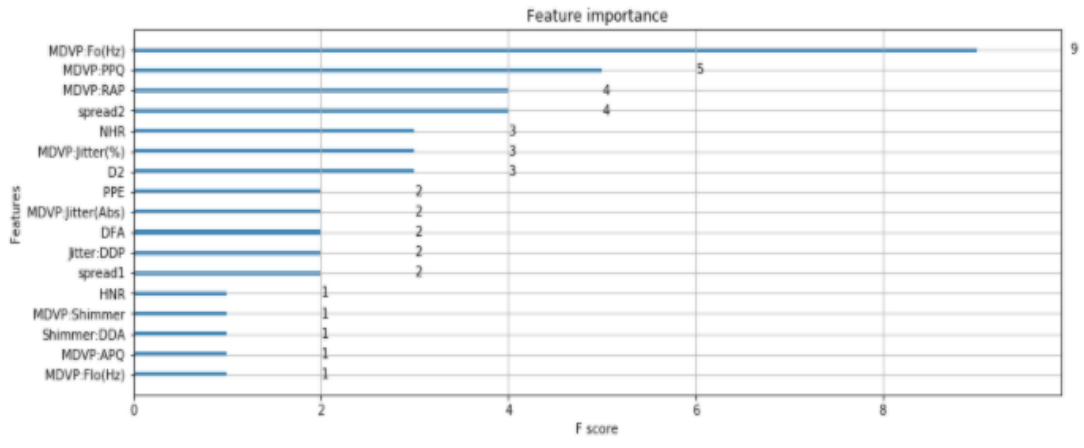

Fig 7.3

CONCLUSION

Currently, the Parkinson's disease research area is of much significance and its detection at the early stage can make the patient's life better. The recent developments in the methodologies through speech analysis have produced significant results. In our work, the problem of identification of Parkinson's disease is coped through a machine learning approach. The main aim of this work is to show the PD diagnosis by analysing the voice signals. From many years, speech processing has an incredible potential in the detection of PD as voice measurements are non-invasive. In our project we have used XGBoost machine learning algorithm. An accuracy of 92.3077% was provided by the machine learning model with RMSE of 0.277350 which is quite good. Also on computing the k-fold cross validation to test the model, the model gives Mean Test RMSE of 0.281145 which is very close to the RMSE given by our model initially. Finally using the **XGBoost's plot_importance()** function we have found out that the feature **MDVP.Fo(Hz) (i.e. Average vocal fundamental frequency)** has the highest importance score among all the features. Thus the proposed model is a reliable model to detect Parkinson's disease due to its efficient accuracy rates.

Though the model works efficiently, this is limited by the richness of the dataset with which it is being trained. The selected dataset, has only 197 instances, hence in future if we use a dataset with more no of samples it would help the model generalize even better.

REFERENCES

1. https://www.kaggle.com/prashant111/xgboost-k-fold-cv-feature-importance
2. https://randerson112358.medium.com/ai-in-health-2e9f84906bed
3. https://www.datacamp.com/community/tutorials/xgboost-in-python
4. https://codeburst.io/using-python-to-detect-early-onset-parkinsons-disease-b89651b0ed3
5. Sakar, C. O., & Kursun, O. (2010). Telediagnosis ofParkinson's disease using measurements of dysphonia.Journal of medical systems, 34(4), 591-599.
6. Rahn III, D. A., Chou, M., Jiang, J. J., & Zhang, Y. (2007).Phonatory impairment in Parkinson's disease: evidence
7. https://www.datainsightonline.com/post/xgboost_predicting-parkinson-diseases
8. D.J. Gelb, E. Oliver, and S. Gilman, "Diagnostic criteria for
9. Parkinson disease." Archives of neurology, vol. 56, no.1, pp. 3999.
10. R. Das, "A Comparison of multiple classification methods for diagnosis of Parkinson disease." Expert Systems with Applications, vol. 37, no. 2, pp. 1568-1572, 2010.