PARKINSON'S DISEASE PREDICTION BY ANALYSING VOICE RECORDING SAMPLE

Submitted in partial fulfilment of the requirement of the degree of Bachelor of Technology

In

Computer Science and Engineering

NAMRATA DE

(ROLL NO: 16500115027)

DEBLINA MITRA

(ROLL NO: 16500115018)

MEGHNA CHATTOPADHYAY

(ROLL NO: 16500115026)

UNDER THE SUPERVISION OF

PROF. MRS. SURANITA SARKAR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CALCUTTA INSTITUTE OF ENGINEERING AND MANAGEMENT

PRASANTA SUR CAMPUS

KOLKATA-700040, INDIA

PARKINSON'S DISEASE PREDICTION BY ANALYSING VOICE RECORDING SAMPLE

Submitted in partial fulfillment of the requirement of the degree of

Bachelor of Technology

In

COMPUTER SCIENCE AND ENGINEERING

NAMRATA DE (16500115027)

DEBLINA MITRA (16500115018)

MEGHNA CHATTOPADHYAY(16500115026)

Under the Supervision of

Prof. SURANITA SARKAR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CALCUTTA INSTITUTE OF ENGINEERING AND MANAGEMENT

PRASANTA SUR CAMPUS
KOLKATA-700040, INDIA

MAY - 2019

APPROVAL SHEET

This project report entitled "**Parkinson's Disease Prediction by Analyzing Voice Recording Sample**" by Namrata De, Deblina Mitra, Meghna Chattopadhyay is approved for the degree of Bachelor of Technology in Computer Science and Engineering.

Supervisor(s)

Board of Examiners

(External)

Head of the Dept.

Debdutta Pal

Principal

Date :

Place :

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

Namrata De

Namrata De (16500115027)

Deblina Mitra (16500115018)

Meghna Chattopadhyay(16500115026)

Date: 20/05/19

CALCUTTA INSTITUTE OF ENGINEERING AND MANAGEMENT
PRASANTA SUR CAMPUS
APPROVED BY A.I.C.T.E. AND AFFILIATED TO M.A.K.A.U.T.
24/1A, CHANDI GHOSH ROAD, TOLLYGUNGE, KOLKATA - 700 040
TEL. : (033) 2421 9951/8998 • TELEFAX : 2481-6767 • WEBSITE : www.ciem.ac.in

CERTIFICATE

is certified that the work contained in the project titled "PARKINSON'S DISEASE PREDICTION SING MACHINE LEARNING" by "Namrata De" has been carried out under my/our supervision and at this work has not been submitted elsewhere for a degree.

Signature of Supervisor(s)

Assistant Professor Suranita Basu

D Department of Computer Science and Engineering
Calcutta Institute of Engineering and Management
May, 2019

CONTENTS

CHAPTER 1

1.1 INTRODUCTION

Parkinson disease (PD) is a progressive neurologic condition that causes motor and non-motor manifestations. Treatment provides symptomatic benefit but no current treatment has been proven to slow disease progression. Research studies of PD require a means of rating the severity of disease by measurement of motor manifestations, assessment of ability to perform daily functional activities, and symptomatic response to medication. The most common rating scales are the Unified Parkinson Disease Rating Scale (UPDRS), Hoehn and Yahr staging, and the Schwab and England rating of activities of daily living. Each of these rating scales are described, including detailed instructions on how to implement these ratings. Although these are the most widely applied rating scales of PD, there are still substantial limitations to these scales that must be considered when using them for research. Finally, some common applications of these scales are described.

Parkinson disease (PD) is a neurodegenerative disease that typically begins about age 60 and causes slowness of movement (bradykinesia), muscular stiffness (rigidity), tremor, poor postural stability, soft voice, shuffling gait, sudden cessation of movement called freezing, and a paucity of spontaneous movements (akinesia). Motor manifestations typically begin on one side of the body, only later affecting the other side as well. Underlying degeneration of dopaminergic nigrostriatal neurons with subsequent striatal dopamine deficiency forms the basis for pharmacotherapy.

1.2 THE SCALES

Unified Parkinson Disease Rating Scale (UPDRS)

Multiple different scales for PD have been developed for quantification of motor manifestations (Webster, Columbia University Rating Scale, and Parkinson's disease Impairment Scale); disability (Schwab and England and Northwestern University Disability Scale); or both (UPDRS and New York University Scale). Of these different scales, the UPDRS has gained the greatest acceptance as a tool for evaluation of interventions and as a clinical tool to follow patients. However, there are important limitations to this scale, and a new UPDRS is undergoing validation testing.

The current UPDRS includes four subscales. Subscale 1 covers mentation, behavior, and mood. Subscale 2 rates activities of daily living. Subscale 3 is a clinician rating of the motor manifestations of PD. Subscale 4 covers complications of therapy. Data for subscales 1, 2, and 4 are elicited from patients and caregivers, whereas data for subscale 3 is examination-based. There are training tapes for the UPDRS subscales, and reviewing these can improve the reliability of the measures. However, reliability of the other subscales depends on patient reporting in addition to examiner skills, but there is a training tape for the activities of daily living component subscale. The total UPDRS score and the UPDRS subscale scores are not interval scales, which means that there are not quantified, equal distances between values on these scales. For example, a score of 4 is greater than 2 but does not necessarily indicate twice the degree of severity. Each part of the rating is a rank order measure rather than a precise interval change. This must be considered when using these data for statistical analyses.

1.3 SYMPTOMS OF PARKINSON'S DISEASE

The symptoms can be categorized in two major types: Motor and Non-motor symptoms

Motor Symptoms

- Tremor
- Muscle Rigidity
- Slowness of Movement
- Forward-bent posture

Non-Motor Symptoms

- Pain
- Anxiety
- Depression
- Reduced of sense of smell

1.4 OBJECTIVE OF THE PROJECT

Research suggests that speech production can be modelled as a nonlinear dynamical system, wherein small perturbations in the interaction of its parts give rise to chaotic yet deterministic behaviour. Parkinson's-related impairments (e.g. tremors, etc.) to the vocal organs, muscles and nerves can affect dynamics of the entire system, suggesting that nonlinear measures may benefit the prediction of disease stage from voice recordings.

UPDRS or Unified Parkinson's Disease Rating Scale helps us differentiate between healthy (non-affected) people and people with Parkinson's.

- Our analysis focuses on the development of an objective, automated method to extract clinically useful information from in the context of Parkinson's disease (PD).
- We aim to predict total UPDRS score of a person, analyzing his or her voice.

1.5 SCOPE OF THE PROJECT

- As Unified Parkinson's Disease Rating Scale (UPDRS) metric provides a clinical impression of PD symptom severity,the model will help enforce the findings in the actual physical interview.

- Knowing UPDRS Score clinically is costly (in terms of time, money, energy and resources), so automation of it will help us to a great extent.

CHAPTER 2

2.1 MACHINE LEARNING

Arthur Samuel, a pioneer in the field of artificial intelligence and computer gaming, coined the term "Machine Learning". He defined machine learning as – "Field of study that gives computers the capability to learn without being explicitly programmed".
In a very layman manner, Machine Learning(ML) can be explained as automating and improving the learning process of computers based on their experiences without being actually programmed i.e. without any human assistance. The process starts with feeding good quality data and then training our machines(computers) by building machine learning models using the data and different algorithms. The choice of algorithms depends on what type of data do we have and what kind of task we are trying to automate.

2.2 TYPES OF MACHINE LEARNING

Machine learning implementations are classified into three major categories, depending on the nature of the learning "signal" or "response" available to a learning system which are as follows:-

2.2.1 Supervised learning : When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of Supervised learning. This approach is indeed similar to human learning under the supervision of a teacher. The teacher provides good examples for the student to memorize, and the student then derives general rules from these specific examples.

2.2.2 Unsupervised learning : Whereas when an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of un-correlated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

As a kind of learning, it resembles the methods humans use to figure out that certain objects or events are from the same class, such as by observing the degree of similarity between objects. Some recommendation systems that we find on the web in the form of marketing automation are based on this type of learning.

2.2.3 Reinforcement learning: When we present the algorithm with examples that lack labels, as in unsupervised learning. However, we can accompany an example with positive or negative feedback according to the solution the algorithm proposes comes under the category of Reinforcement learning, which is connected to applications for which the algorithm must make decisions (so the product is prescriptive, not just descriptive, as in unsupervised learning), and the decisions bear consequences. In the human world, it is just like learning by trial and error.

Errors help we learn because they have a penalty added (cost, loss of time, regret, pain, and so on), teaching we that a certain course of action is less likely to succeed than others. An interesting example of reinforcement learning occurs when computers learn to play video games by themselves.

In this case, an application presents the algorithm with examples of specific situations, such as having the gamer stuck in a maze while avoiding an enemy. The application lets the algorithm know the outcome of actions it takes, and learning occurs while trying to

avoid what it discovers to be dangerous and to pursue survival. One can have a look at how the company Google DeepMind has created a reinforcement learning program that plays old Atari's videogames. When watching the video, notice how the program is initially clumsy and unskilled but steadily improves with training until it becomes a champion.

2.2.4 Semi-supervised learning : Where an incomplete training signal is given: a training set with some (often many) of the target outputs missing. There is a special case of this principle known as Transduction where the entire set of problem instances is known at learning time, except that part of the targets are missing.

2.3 CATEGORIZING ON THE BASIS OF REQUIRED OUTPUT

Another categorization of machine learning tasks arises when one considers the desired output of a machine-learned system:

2.3.1 Classification : When inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".

2.3.2 Regression: Which is also a supervised problem, A case when the outputs are continuous rather than discrete.

2.3.3 Clustering: When a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.

2.4 SUPERVISED AND UNSUPERVISED LEARNING

2.4.1 Supervised learning

Supervised learning as the name indicates the presence of a supervisor as a teacher. Basically supervised learning is a learning in which we teach or train the machine using data which is well

labelled that means some data is already tagged with the correct answer. After that, the machine is provided with a new set of examples(data) so that supervised learning algorithm analyses the training data(set of training examples) and produces a correct outcome from labelled data.

The majority of practical machine learning uses supervised learning.
Supervised learning is where we have input variables (x) and an output variable (Y) and we use an algorithm to learn the mapping function from the input to the output.
$Y = f(X)$
The goal is to approximate the mapping function so well that when we have new input data (x) that we can predict the output variables (Y) for that data.
It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

Supervised learning problems can be further grouped into regression and classification problems.

- Classification: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
- Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Some common types of problems built on top of classification and regression include recommendation and time series prediction respectively.
Some popular examples of supervised machine learning algorithms are:

- Linear regression for regression problems.
- Random forest for classification and regression problems.
- Support vector machines for classification problems.

2.4.2 Unsupervised learning

Unsupervised learning is the training of machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.

Unsupervised learning is where we only have input data (X) and no corresponding output variables.
The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.
These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data.
Unsupervised learning problems can be further grouped into clustering and association problems.

- Clustering: A clustering problem is where we want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- Association: An association rule learning problem is where we want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Some popular examples of unsupervised learning algorithms are:

- k-means for clustering problems.
- Apriori algorithm for association rule learning problems.

2.4.3 Semi-supervised learning

As the name suggests, its working lies between Supervised and Unsupervised techniques. We use these techniques when we are dealing with a data which is a little bit labelled and rest large portion of it is unlabelled. We can use unsupervised technique to predict labels and then feed these labels to supervised techniques. This technique is mostly applicable in case of image data-sets where usually all images are not labelled.

2.4.4 Reinforcement Learning

In this technique, model keeps on increasing its performance using a Reward Feedback to learn the behavior or pattern. These algorithms are specific to a particular problem e.g. Google Self Driving car, AlphaGo where a bot competes with human and even itself to getting better and better performer of Go Game. Each time we feed in data, they learn and add the data to its knowledge that is training data. So, more it learns the better it get trained and hence experienced.

- Agents observe input.
- Agent performs an action by making some decisions.
- After its performance, agent receives reward and accordingly reinforce and the model stores in state-action pair of information.

2.5 STEPS OF MACHINE LEARNING

Machine learning is a field of computer science that gives computers the ability to learn without being programmed explicitly. The power of machine learning is that we can determine how to differentiate using models, rather than using human judgment. The basic steps that lead to machine learning and will teach we how it works are described below in a big picture:

1. Gathering data
2. Preparing that data
3. Choosing a model
4. Training
5. Evaluation
6. Hyper parameter tuning
7. Prediction.

2.5.1 Gathering Data:

Once we know exactly what we want and the equipments are in hand, it takes we to the first real step of machine learning- Gathering Data. This step is very crucial as the quality and quantity of data gathered will directly determine how good the predictive model will turn out to be. The data collected is then tabulated and called as Training Data.

2.5.2 Data Preparation:

After the training data is gathered, we move on to the next step of machine learning: Data preparation, where the data is loaded into a suitable place and then prepared for use in machine learning training. Here, the data is first put all together and then the order is randomized as the order of data should not affect what is learned.
This is also a good enough time to do any visualizations of the data, as that will help we see if there are any relevant relationships between the different variables, how we can take their

advantage and as well as show we if there are any data imbalances present. Also, the data now has to be split into two parts. The first part that is used in training our model, will be the majority of the dataset and the second will be used for the evaluation of the trained model's performance. The other forms of adjusting and manipulation like normalization, error correction, and more take place at this step.

2.5.3 Choosing a model:

The next step that follows in the workflow is choosing a model among the many that researchers and data scientists have created over the years. Make the choice of the right one that should get the job done.

2.5.4 Training:

After the before steps are completed, we then move onto what is often considered the bulk of machine learning called training where the data is used to incrementally improve the model's ability to predict.
The training process involves initializing some random values for say A and B of our model, predict the output with those values, then compare it with the model's prediction and then adjust the values so that they match the predictions that were made previously.
This process then repeats and each cycle of updating is called one training step.

2.5.5 Evaluation:

Once training is complete, we now check if it is good enough using this step. This is where that dataset we set aside earlier comes into play. Evaluation allows the testing of the model against data that has never been seen and used for training and is meant to be representative of how the model might perform when in the real world.

2.5.6 Parameter Tuning:

Once the evaluation is over, any further improvement in your training can be possible by tuning the parameters. There were a few parameters that were implicitly assumed when the training was done. Another parameter included is the learning rate that defines how far the line is shifted during each step, based on the information from the previous training step. These values all play a role in the accuracy of the training model, and how long the training will take.For models that are more complex, initial conditions play a significant role in the determination of the outcome of training. Differences can be seen depending on whether a model starts off training with values initialized to zeros versus some distribution of values,

which then leads to the question of which distribution is to be used. Since there are many considerations at this phase of training, it's important that we define what makes a model good. These parameters are referred to as Hyper parameters. The adjustment or tuning of these parameters depends on the dataset, model, and the training process. Once we are done with these parameters and are satisfied we can move on to the last step.

2.5.7 Prediction of Target Feature:

Machine learning is basically using data to answer questions. So this is the final step where we get to answer few questions. This is the point where the value of machine learning is realized. Here we can Finally use your model to predict the outcome of what we want.The above-mentioned steps take we from where we create a model to where we Predict its output and thus acts as a learning path.

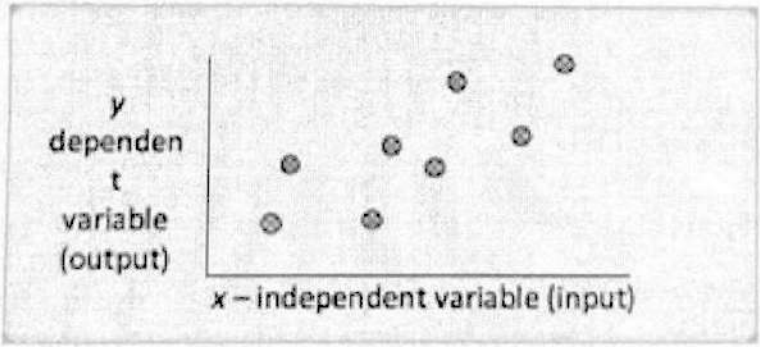
CHAPTER 3

3.1 Supervised Machine Learning

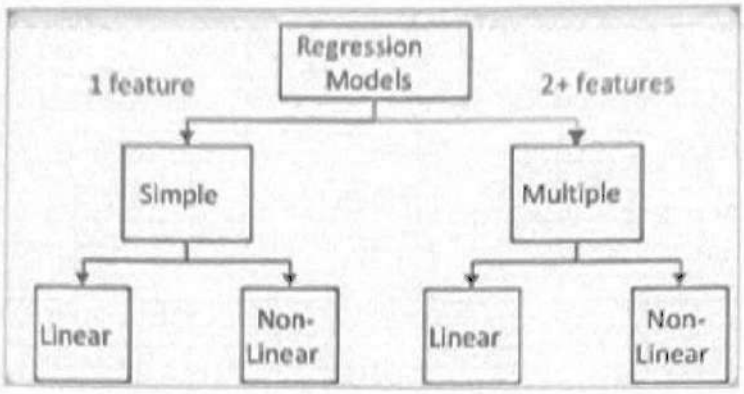
The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output Y = f(X) . The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data. Techniques of Supervised Machine Learning algorithms include linear and logistic regression, multi-class classification, Decision Trees and support vector machines. Supervised learning requires that the data used to train the algorithm is already labeled with correct answers.
For example, a classification algorithm will learn to identify animals after being trained on a dataset of images that are properly labeled with the species of the animal and some identifying characteristics. Supervised learning problems can be further grouped into Regression and Classification problems. Both problems have as goal the construction of a succinct model that can predict the value of the dependent attribute from the attribute variables. The difference between the two tasks is the fact that the dependent attribute is numerical for regression and categorical for classification.

3.1.1 REGRESSION

A regression problem is when the output variable is a real or continuous value, such as "salary" or "weight". Many different models can be used, the simplest is the linear regression. It tries to fit data with the best hyper-plane which goes through the points.



Types of Regression Models:



Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

3.1.1.1 LINEAR REGRESSION

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is − **y = ax + b**

Following is the description of the parameters used −

- y is the response variable.
- x is the predictor variable.
- a and b are constants which are called the coefficients.

Steps to Establish a Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.
The steps to create the relationship is −

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the lm() functions in R.
- Find the coefficients from the model created and create the mathematical equation using these

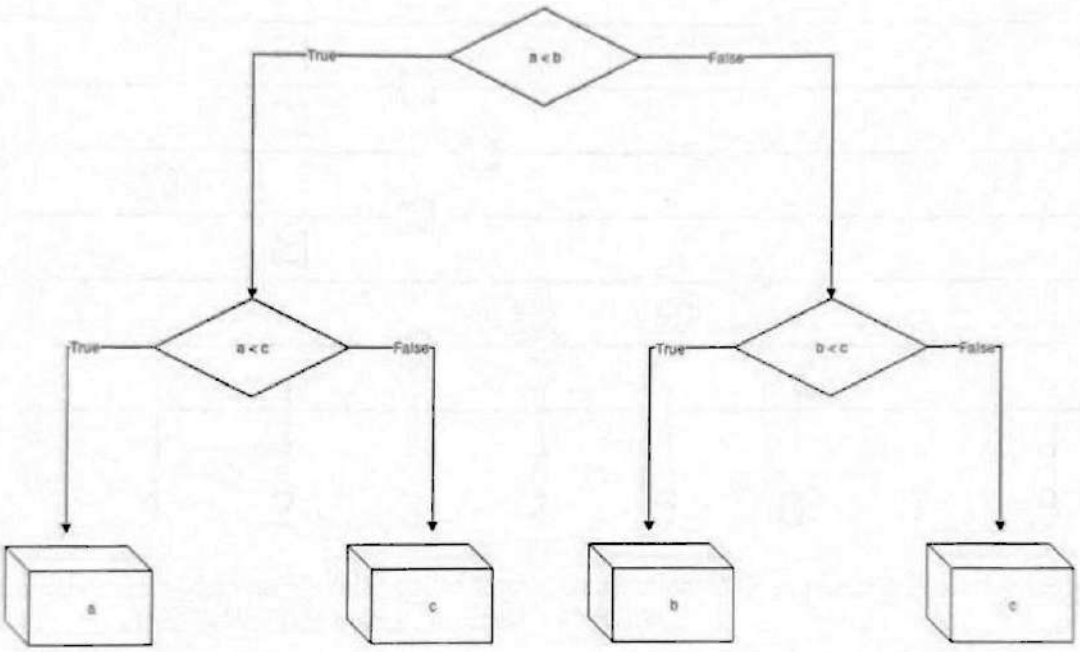
- Get a summary of the relationship model to know the average error in prediction. Also called residuals.
- To predict the weight of new persons, use the predict() function in R.

3.1.1.2 Decision Tree is a decision-making tool that uses a flowchart-like tree structure or is a model of decisions and all of their possible results, including outcomes, input costs and utility.
Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

The branches/edges represent the result of the node and the nodes have either:

1. Conditions [Decision Nodes]
2. Result [End Nodes]

The branches/edges represent the truth/falsity of the statement and takes makes a decision based on that in the example below which shows a decision tree that evaluates the smallest of three numbers:



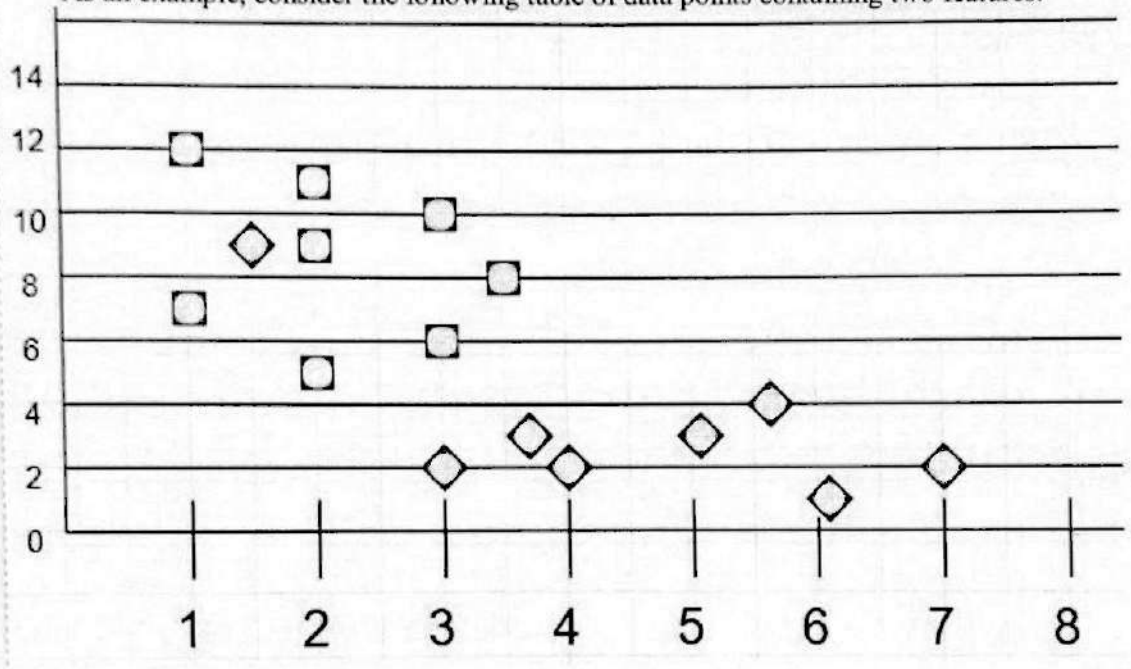
Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

3.1.1.3 K-Nearest Neighbours

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.

It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data).
We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

As an example, consider the following table of data points containing two features:



Now, given another set of data points (also called testing data), allocate these points a group by analyzing the training set. Note that the unclassified points are marked as 'White'.

1. It indicates the significant relationships between dependent variable and independent variable.

2. It indicates the strength of impact of multiple independent variables on a dependent variable.

Regression analysis also allows us to compare the effects of variables measured on different scales, such as the effect of price changes and the number of promotional activities. These benefits help market researchers / data analysts / data scientists to eliminate and evaluate the best set of variables to be used for building predictive models.

3.1.2 CLASSIFICATION

A classification problem is when the output variable is a category, such as "red" or "blue" or "disease" and "no disease". A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes.

For example, when filtering emails "spam" or "not spam", when looking at transaction data, "fraudulent", or "authorized". In short Classification either predicts categorical class labels or classifies data (construct a model) based on the training set and the values (class labels) in classifying attributes and uses it in classifying new data. There are a number of classification models. Classification models include **logistic regression, decision tree, random forest, gradient boosted tree, multilayer perceptron, one-vs-rest, and Naive Bayes**.

For example : Among of the following,

- Predicting the gender of a person by his/her handwriting style
- Predicting house price based on area
- Predicting whether monsoon will be normal next year
- Predict the number of copies a music album will be sold next month Predicting the gender of a person and Predicting whether monsoon will be normal next year are classification problems. The other two are regression.

Here we have the types of classification algorithms in Machine Learning:

- Logistic Regression
- Naive Bayes' Classifier
- Support Vector Machines
- Decision Trees
- Boosted Trees
- Random Forest
- Neural Networks

- Nearest Neighbour

3.1.2.1 Naive Bayes Classifier (Generative Learning Model): It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

3.1.2.2 Logistic Regression (Predictive Learning Model) : It is a statistical method for analysing a data set in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables.

3.1.2.3 Decision Trees: Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches and a leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

3.1.2.4 Random Forest: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.

3.1.2.5 K Nearest Neighbour: The k-nearest-neighbours algorithm is a classification algorithm, and it is supervised: it takes a bunch of labelled points and uses them to learn how to label other points. To label a new point, it looks at the labelled points closest to that new point (those are its nearest neighbours), and has those neighbours vote, so whichever label the most of the neighbours have is the label for the new point (the "k" is the number of neighbours it checks).

3.2 Measurement of Different Performance Metrices:

A. For Classificaton:

There are essentially four states for any particular piece of data we evaluate. On one axis we choose whether this piece of data correctly belongs to a class or whether it does not correctly belong to a class, and thus this axis we describe as truth. Therefore, there are four possibilities that occur – true positive (tp), true negative (tn), false positive (fp) and false negative (fn).

Definition of the Terms:

- Positive (P) : Observation is positive (for example: is an apple).
- Negative (N) : Observation is not positive (for example: is not an apple).
- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

For example, if the truth of the data instance is correct and we select it, then it is a true positive. Another possibility is that our classifier does not say it it correct, then it is a false negative. On the other hand, it is possible that the instance is not correct in which case there are two possibilities – our classifier mistakenly classifies the instance to the "correct" category, which is called false positive, or our classifier correctly places the instance into the "not correct" category, which is called true negative.

a.Accuracy

Accuracy is the first reasonable measure to look at. Accuracy equals the true positives, plus the true negatives over all four classes (true positives, true negatives, false positives and false negatives). In many applications, accuracy is a useful measure for the Naïve Bayes classifier. But there is a particular scenario when dealing with things that are uncommon, in which the accuracy is not a useful measure. For example, 99.99% of the data are from category A, while only 0.01% is from the counterpart category B. So, we will have 99.99% of accuracy even when our classifier assigned all data into category B, which is apparently undesired. For this situation, precision and recall are used as the measurement of our classifier.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

b. Recall

Recall is the percentage of correct items that are selected. By applying recall as a measurement, the babove-mentioned situation for accuracy can be resolved. Simply assigning all items to category A will give zero recall, which turns out to be an undesired classifier. In various applications, such as for things like legal applications, where you want to find all of the appropriate evidence, such as in discovery procedures. What you really want to do is having a classifier that has high recall, that finds as much of the relevant items as possible. In other words, you do not want relevant evidences left unselected.

$$\textit{Recall} = \frac{TP}{TP + FN}$$

c. Precision

Precision is the percentage of selected items that are correct. Precision can also be a resolution for the condition when dealing with uncommon things. In some contexts, you 12 might be more interested in precision over recall. For example, you want to show customers some merchandises that are good (correct), and do not care that only 1/10 or 1/20 of the things do satisfy their query.

$$\textit{Precision} = \frac{TP}{TP + FP}$$

d. Confusion Matrix:

A confusion matrix is a summary of prediction results on a classification problem.

The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.

The confusion matrix shows the ways in which your classification model is confused when it makes predictions.

It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

Here,

- Class 1 : Positive
- Class 2 : Negative

B. For Regression

a. Mean Absolute Error (MAE)

The Mean Absolute Error measures the average of the absolute difference between each ground truth and the predictions. Whether the predictions is 10 or 6 while the ground truth was 8, the absolute difference is 2.

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

The y^(hat) is the prediction but again, the order doesn't matter since we are calculating the absolute contrast.

b.Root Mean Squared Error (RMSE)

The Root Mean Squared Error measures the square root of the average of the squared difference between the predictions and the ground truth.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$$

c. R-Squared Value

$$\hat{R}^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2} = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

R-squared is a statistical measure of how close the data are to the fitted **regression** line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple **regression**. 0% indicates that the model explains none of the variability of the response data around its mean.

d. Explained Variance Score

In statistics, explained variation measures the proportion to which a mathematical model accounts for the variation (dispersion) of a given data set. Often, variation is quantified as variance; then, the more specific term explained variance can be used.

The complementary part of the total variation is called unexplained or residual variation.

CHAPTER 4

4.1 RANDOM FOREST REGRESSION

INTRODUCTION

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

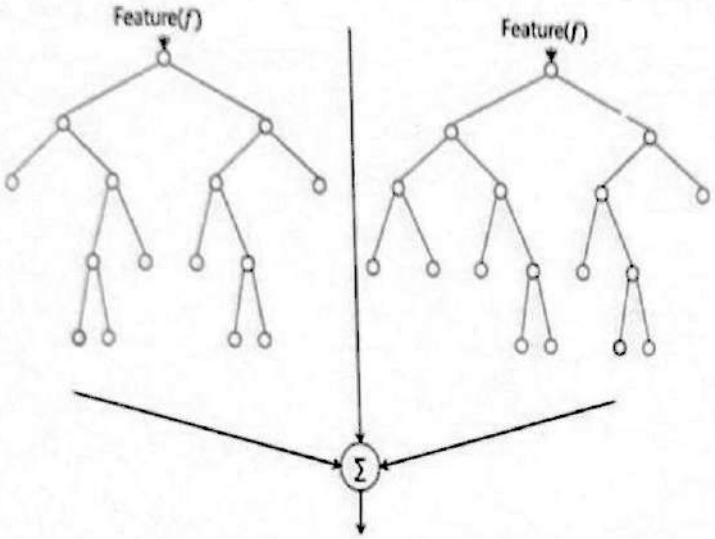
The first algorithm for random decision forests was created by Tin Kam Ho using the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

An extension of the algorithm was developed by Leo Breiman and Adele Cutler, who registered "Random Forests" as a trademark(as of 2019, owned by Minitab, Inc.). The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho and later independently by Amit and Geman in order to construct a collection of decision trees with controlled variance.

The Random Forest is one of the most effective machine learning models for predictive analytics, making it an industrial workhorse for machine learning.The **random forest** model is a type of additive model that makes predictions by combining decisions from a sequence of base models. More formally we can write this class of models as: where the final model is the sum of simple base models . Here, each base classifier is a simple decision tree. This broad technique of using multiple models to obtain better predictive performance is called **model ensembling**. In random forests, all the base models are constructed independently using a **different subsample** of the data.

HOW DO RANDOM FOREST WORKS?

Random Forest is a supervised learning algorithm. Like you can already see from it's name, it creates a forest and makes it somehow random. The „forest" it builds, is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.
To say it in simple words: "Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction."



One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. I will talk about random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with two trees:

Random Forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, you don't have to combine a decision tree with a bagging classifier and can just easily use the classifier-class of Random Forest. Like I already said, with Random Forest, you can also deal with Regression tasks by using the Random Forest regressor.
Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

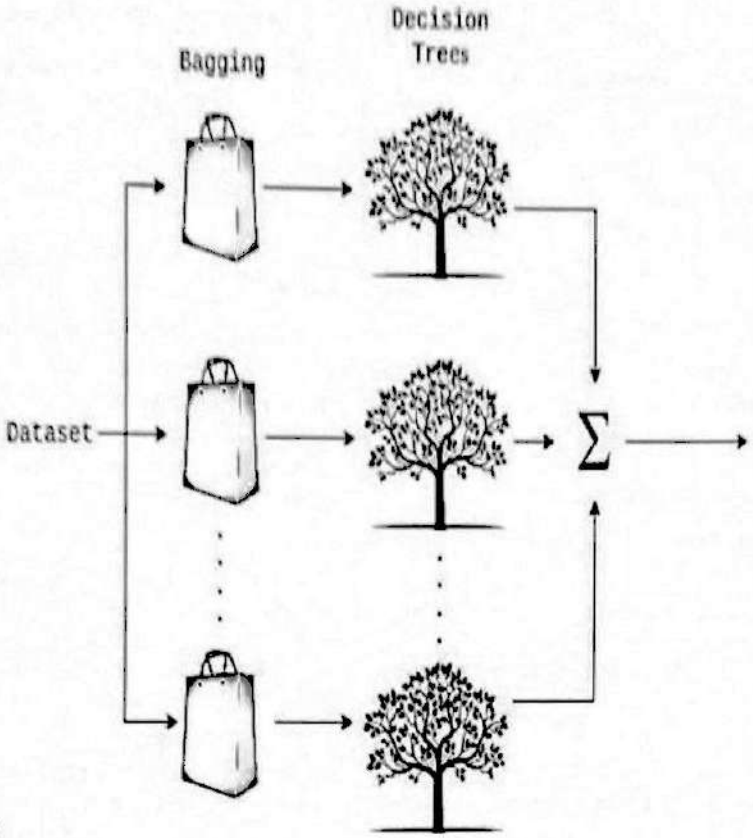
Real Life Analogy:

Imagine a guy named Andrew, that want's to decide, to which places he should travel during a one-year vacation trip. He asks people who know him for advice. First, he goes to a friend, tha asks Andrew where he traveled to in the past and if he liked it or not. Based on the answers, he will give Andrew some advice.This is a typical decision tree algorithm approach. Andrews friend created rules to guide his decision about what he should recommend, by using the answers of Andrew.Afterwards, Andrew starts asking more and more of his friends to advise him and they again ask him different questions, where they can derive some recommendations from. Then he chooses the places that where recommend the most to him, which is the typical Random Forest algorithm approach.

Feature Importance:

Another great quality of the random forest algorithm is that it is very easy to measure the relative importance of each feature on the prediction. Sklearn provides a great tool for this, that measures a features importance by looking at how much the tree nodes, which use that feature, reduce impurity across all trees in the forest. It computes this score automatically for each feature after training and scales the results, so that the sum of all importance is equal to 1.If you don't know how a decision tree works and if you don't know what a leaf or node is, here is a good description from Wikipedia: In a decision tree each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). A node that has no children is a leaf.Through looking at the feature importance, you can decide which features you may want to drop, because they don't contribute enough or nothing to the prediction process. This is important, because a general rule in machine learning is that the more features you have, the more likely your model will suffer from overfitting and vice versa.

RANDOM FOREST ALGORITHM:



BAGGING:

The ensemble method we will be using today is called bagging, which is short for bootstrap aggregating.Bagging builds multiple base models with resampled training data with replacement. We train k base classifiers on k different samples of training data. Using random subsets of the data to train base models promotes more differences between the base models.

We can use the BaggingRegressor class to form an ensemble of regressors. One such Bagging algorithms are random forest regressor. A random forest regressor is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default). Random Forest Regressors uses some kind of splitting criterion to measure the quality of a split. Supported criteria are "MSE" for the mean squared error, which is equal to variance reduction as feature selection criterion, and "Mean Absolute Error" for the mean absolute error.

Several estimators are built independently on subsets of the data and their predictions are averaged. Typically, the combined estimator is usually better than any of the single base estimator.Bagging can reduce variance with little to no effect on bias.

ex: Random Forests

Random Forest Algorithm Advantages:

- Random Forest can be used to solve both kinds of problems: regression and classification.
- Is capable of handling high dimensional data sets.
- Can be used to extract out relevant features.
- Handles missing data effectively internally.

Random Forest Algorithm Disadvantages:

- Difficult to interpret because of various trees involved internally
- It tends to return erratic predictions for observations out of range of training data. For example, the training data contains two variable x and y. The range of x variable is 30 to 70. If the test data has x = 200, the random forest would give an unreliable prediction.
- It can take a much longer time than expected to grow a large number of trees.

Important Parameters

The Parameters in random forest are either used to increase the predictive power of the model or to make the model faster.

n_estimators : integer, optional (default=10)
The number of trees in the forest. The default value of n_estimators will change from 10 in version 0.20 to 100 in version 0.22.

criterion : string, optional (default="mse")
The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to variance reduction as feature selection criterion, and "mae" for the mean absolute error.

max_depth : integer or None, optional (default=None)
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : int, float, optional (default=2)
The minimum number of samples required to split an internal node:

- If int, then consider min_samples_split as the minimum number.
- If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

min_samples_leaf : int, float, optional (default=1)

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

- If int, then consider min_samples_leaf as the minimum number.
- If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node..

min_weight_fraction_leaf : float, optional (default=0.)
The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_features : int, float, string or None, optional (default="auto")
The number of features to consider when looking for the best split:

- If int, then consider max_features features at each split.
- If float, then max_features is a fraction and int(max_features * n_features) features are considered at each split.
- If "auto", then max_features=n_features.
- If "sqrt", then max_features=sqrt(n_features).
- If "log2", then max_features=log2(n_features).
- If None, then max_features=n_features.

Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features features.

max_leaf_nodes : int or None, optional (default=None)

Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

min_impurity_decrease : float, optional (default=0.)

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

The weighted impurity decrease equation is the following:

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

where N is the total number of samples, N_t is the number of samples at the current node, N_t_L is the number of samples in the left child, and N_t_R is the number of samples in the right child.

N, N_t, N_t_R and N_t_L all refer to the weighted sum, if sample_weight is passed.

min_impurity_split : float, (default=1e-7)

Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.

bootstrap : boolean, optional (default=True)

Whether bootstrap samples are used when building trees. If False, the whole datset is used to build each tree.

oob_score : bool, optional (default=False)

whether to use out-of-bag samples to estimate the R^2 on unseen data.

n_jobs : int or None, optional (default=None)

The number of jobs to run in parallel for both **fit** and **predict**. None' means 1 unless in a **joblib.parallel_backend** context. -1 means using all processors. See Glossary for more details.

random_state : int, RandomState instance or None, optional (default=None)

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.

verbose : int, optional (default=0)
Controls the verbosity when fitting and predicting.

warm_start : bool, optional (default=False)
When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest.

Summary

Random Forest is a great algorithm to train early in the model development process, to see how it performs and it's hard to build a "bad" Random Forest, because of its simplicity. This algorithm is also a great choice, if you need to develop a model in a short period of time. On top of that, it provides a pretty good indicator of the importance it assigns to your features.

Random Forests are also very hard to beat in terms of performance. Of course you can probably always find a model that can perform better, like a neural network, but these usually take much more time in the development. And on top of that, they can handle a lot of different feature types, like binary, categorical and numerical.

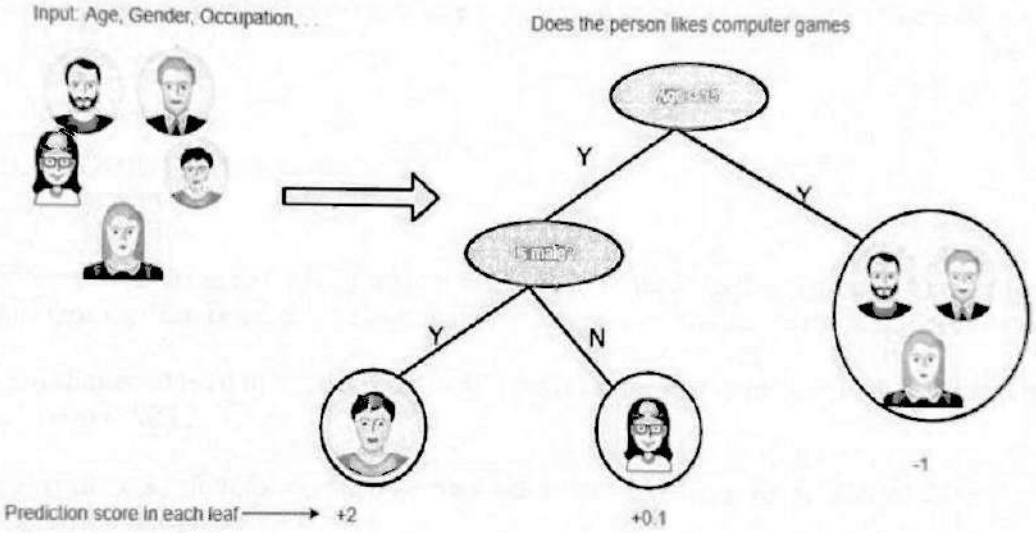
Overall, Random Forest is a (mostly) fast, simple and flexible tool, although it has its limitations.

4.2 DECISION TREE REGRESSION

INTRODUCTION

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

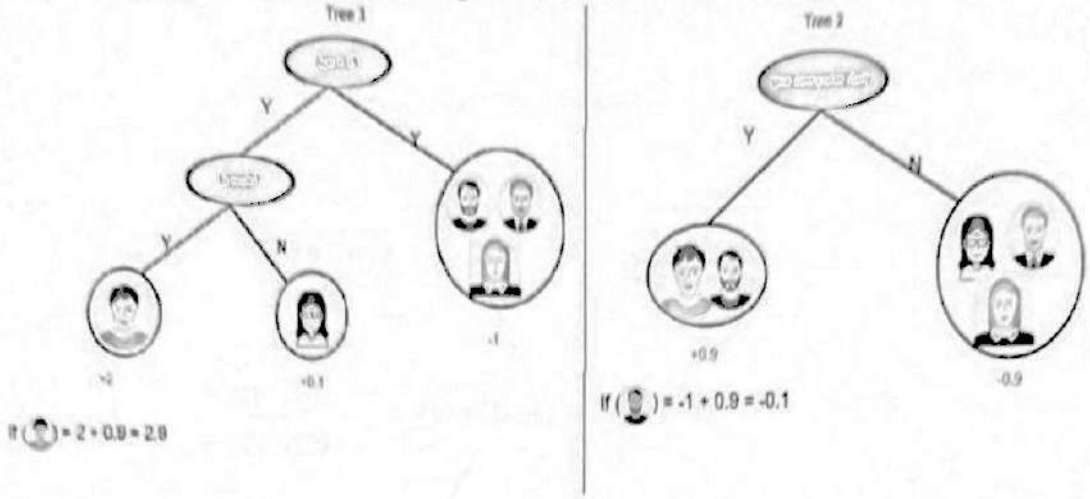
- Decision tree algorithm falls under the category of the supervised learning. They can be used to solve both regression and classification problems.
- Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.
- We can represent any boolean function on discrete attributes using the decision tree.



Below are some assumptions that we made while using decision tree:

- At the beginning, we consider the whole training set as the root.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- On the basis of attribute values records are distributed recursively.

- We use statistical methods for ordering attributes as root or the internal node.



As you can see from the above image that Decision Tree works on the Sum of Product form which is also known as *Disjunctive Normal Form*. In the above image we are predicting the use of computer in daily life of the people.

DECISION TREE ALGORITHM

Classification and Regression Trees or CART for short is a term introduced by Leo Breiman to refer to Decision Tree algorithms that can be used for classification or regression predictive modeling problems.

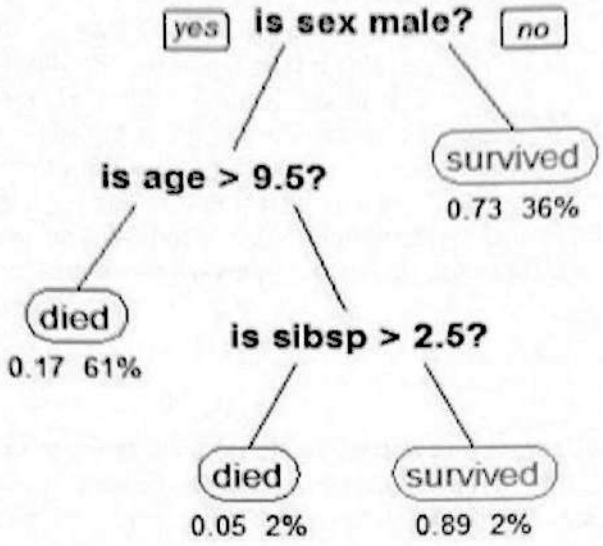
Classically, this algorithm is referred to as "decision trees", but on some platforms like R they are referred to by the more modern term CART.

The CART algorithm provides a foundation for important algorithms like bagged decision trees, random forest and boosted decision trees.

For this let's consider a very basic example that uses titanic data set for predicting whether a passenger will survive or not. Below model uses 3 features/attributes/columns from the data set, namely sex, age and sibsp (number of spouses or children along).

A decision tree is drawn upside down with its root at the top. In the image on the left, the bold text in black represents a condition/internal node, based on which the tree splits into branches/ edges. The end of the branch that doesn't split anymore is the decision/leaf, in this case, whether the passenger died or survived,

represented as red and green text respectively.



Although, a real dataset will have a lot more features and this will just be a branch in a much bigger tree, but you can't ignore the simplicity of this algorithm. The feature importance is clear and relations can be viewed easily. This methodology is more commonly known as learning decision tree from data and above tree is called Classification tree as the target is to classify passenger as survived or died. Regression trees are represented in the same manner, just they predict continuous values like price of a house. In general, Decision Tree algorithms are referred to as CART or Classification and Regression Trees.

So, what is actually going on in the background? Growing a tree involves deciding on which features to choose and what conditions to use for splitting, along with knowing when to stop. As a tree generally grows arbitrarily, you will need to trim it down for it to look beautiful. Let's start with a common technique used for splitting.

Recursive Binary Splitting

In this procedure all the features are considered and different split points are tried and tested using

Advantages of Decision Tree Regression

- Simple to understand, interpret, visualize.
- Decision trees implicitly perform variable screening or feature selection.
- Can handle both numerical and categorical data. Can also handle multi-output problems.
- Decision trees require relatively little effort from users for data preparation.
- Nonlinear relationships between parameters do not affect tree performance.

Disadvantages of Decision Tree Regression

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging and boosting.
- Greedy algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.

CHAPTER 5

5.1 TOOLS AND SOFTWARE

5.1.1 Anaconda

Anaconda is a an open source distribution of the Python and R programming languages and it is used in data science, machine learning, deep learning-related applications aiming at simplifying package management and deployment.Anaconda Distribution is used by over 7 million users, and it includes more than 300 data science packages suitable for Windows, Linux, and MacOS.

Anaconda® is a package manager, an environment manager, a Python/R data science distribution, and a collection of over 1,500+ open source packages. Anaconda is free and easy to install, and it offers free community support.

Packages available in Anaconda

- Over 200 packages are automatically installed with Anaconda.
- Over 2000 additional open source packages (including R) can be individually installed from the Anaconda repository with the condainstall command.
- Thousands of other packages are available from Anaconda Cloud.
- One can download other packages using the pip install command that is installed with Anaconda. Pip packages provide many of the features of conda packages and in some cases they can work together. However, the preference should be to install the conda package if it is available.

- One can also make his or her own custom packages using the conda build command, and he or she can share them with others by uploading them to Anaconda Cloud, PyPi or other repositories.

5.1.2 Praat

Praat is a free computer software package for the scientific analysis of speech in phonetics. It was designed, and continues to be developed, by Paul Boersma and David Weenink of the University of Amsterdam. It can run on a wide range of operating systems, including various versions of Unix, Linux, Mac and Microsoft Windows (2000, XP, Vista, 7, 8, 10). The program supports speech synthesis, including articulatory synthesis.

5.1.3 Sublime Text

Sublime Text is a proprietary cross-platform source code editor with a Python application programming interface (API). It natively supports many programming languages and markup languages, and functions can be added by users with plugins, typically community-built and maintained under free-software licenses.

CHAPTER 6

6.1 DATA DESCRIPTION

6.1.1 Source:

The dataset was created by Athanasios Tsanas (tsanasthanasis@gmail.com) and Max Little (littlem@physics.ox.ac.uk) of the University of Oxford, in collaboration with 10 medical centers in the US and Intel Corporation who developed the telemonitoring device to record the speech signals. The original study used a range of linear and nonlinear regression methods to predict the clinician's Parkinson's disease symptom score on the UPDRS scale.

6.1.2 Data Set Information:

This dataset is composed of a range of biomedical voice measurements from 42 people with early-stage Parkinson's disease recruited to a six-month trial of a telemonitoring device for remote symptom progression monitoring. The recordings were automatically captured in the patient's homes.

Columns in the table contain subject number, subject age, time interval from baseline recruitment date, motor UPDRS, total UPDRS, and 16 biomedical voice measures. Each row corresponds to one of 5,875 voice recording from these individuals. The main aim of the data is to predict the motor and total UPDRS scores ('motor_UPDRS' and 'total_UPDRS') from the 16 voice measures.

The data is in ASCII CSV format. The rows of the CSV file contain an instance corresponding to one voice recording. There are around 200 recordings per patient, the subject number of the patient is identified in the first column.

Data Set Characteristics:	Multivariate
Number of Instances:	5875
Attribute Characteristics:	Integer, Real
Associated Tasks:	Regression
Number of Attributes:	26
Missing Values?	N/A

Table 1: Dataset information

6.1.3 Feature Description:

Subject number: Integer that uniquely identifies each subject
Subject Age: Integer that uniquely identifies each subject age
Test time - Time since recruitment into the trial. The integer part is the number of days since recruitment.
UPDRS: This is a clinician's scale for recording symptoms related to Parkinson's disease. The UPDRS metric consists of 44 sections, where each section addresses different symptoms in different parts of the body. Summing up these 44 sections gives rise to the total-UPDRS score, which spans the range 0-176, with 0 representing perfectly healthy individual and 176 total disability.
Motor UPDRS - Clinician's motor UPDRS score, linearly interpolated - this forms sections 18-44 from the UPDRS sections
Total_UPDRS - Clinician's total UPDRS score, linearly interpolated - this includes all 44 sections
Jitter Percentage - measure of variation in fundamental frequency
Jitter (Absolute) - measure of variation in fundamental frequency
Jitter (RAP) - measure of variation in fundamental frequency
Jitter (PPQ5) - measure of variation in fundamental frequency
Jitter (DDP) - measure of variation in fundamental frequency
Shimmer - measures of variation in amplitude
Shimmer(dB)- measure of variation in amplitude
Shimmer:APQ3- measure of variation in amplitude
Shimmer:APQ5- measure of variation in amplitude
Shimmer:APQ11- measure of variation in amplitude
Shimmer:DDA- measure of variation in amplitude
NHR: measures of ratio of noise to tonal components in the voice
HNR: measures of ratio of noise to tonal components in the voice
RPDE (recurrence period density entropy)- this entropy measures the periodicity of the system.
When the signal deviates from its trajectory of recurring to the same point in the phase space, this may indicate a voice disorder. Many voice disorders impair the patient's ability to sustain vocal fold vibration, which can be measured as in terms of aperiodicity.

DFA(detrended fluctuation analysis) - extent of stochastic self-similarity of noise in the speech signal.

Air blowing over vocal folds is a major cause of noise in speech, the pattern of which may be disrupted in some voice disorders. This noise can be characterized by a scaling exponent, which is higher in those with vocal disorders.

PPE(pitch period entropy) - this entropy provides another measure of pitch variation (compare to jitter)

- Because pitch is produced and perceived on a logarithmic scale, PPE is calculated first by converting a pitch sequence to the logarithmic semitone scale. A filter then removes natural pitch variations (such as those due to gender and individual differences), and a probability distribution of voice variations is constructed. Finally, entropy is calculated, characterizing the extent of variation beyond natural fluctuations in pitch. Increased PPE may suggest speech variations beyond those seen in healthy speech production.

We will aim to use UDPRS as our response variable.
The aim is twofold: (a) this attribute helps us differentiate between healthy (non-affected) people and people with Parkinson's. (b) replicating the Unified Parkinson's Disease Rating Scale (UPDRS) metric which provides a clinical impression of PD symptom severity will help enforce the findings in the actual physical interview.
This metric (UPDRS) spans the range 0 to 176 , where 0 denotes a healthy person and 176 total disability. Currently, UPDRS assessment requires the physical presence of the subject in the clinic, is subjective relying on the clinical rater's expertise, and costly (in terms of time, money, energy and resources). Hence, the practical frequency of symptom tracking is typically confined to once every several months, hindering recruitment for large-scale clinical trials and under-representing the true time scale of PD fluctuations
We will aim to build and refine a model that will enable us to accurately estimate a measure for UPDRS (total).

6.2 Data Pre-Processing

Exploratory Data Analysis:
Now we will proceed to represent graphically the distribution of the different variables in our dataset performing histograms of each of them:

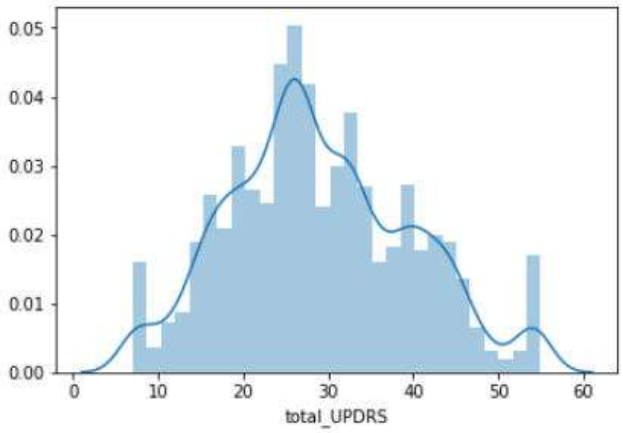
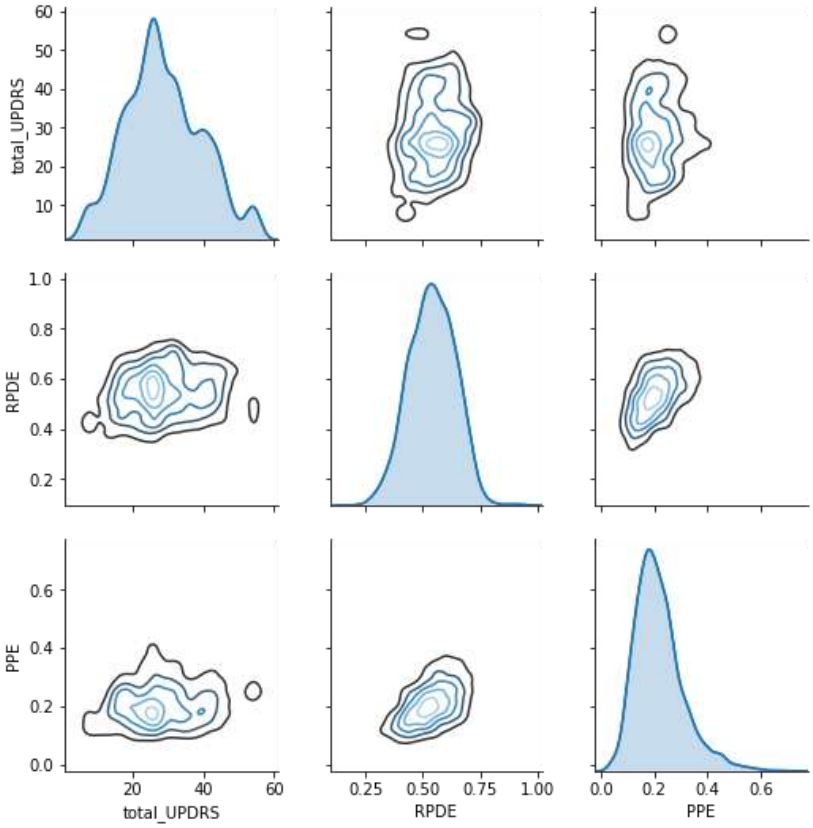
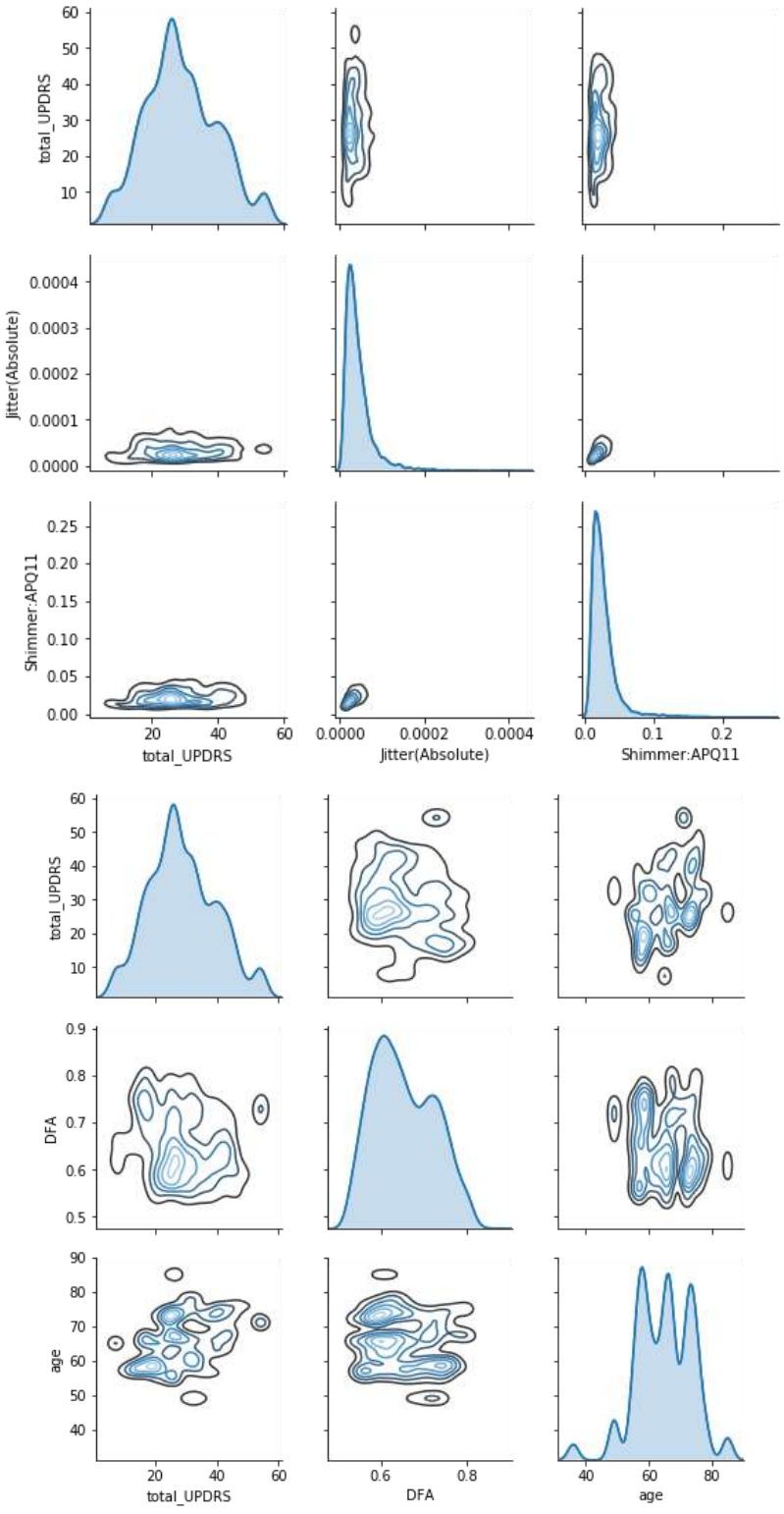


Fig.1 Distribution of total_UPDRS




total_UPDRS
Jitter(Absolute)
Shimmer:APQ11
DFA
age

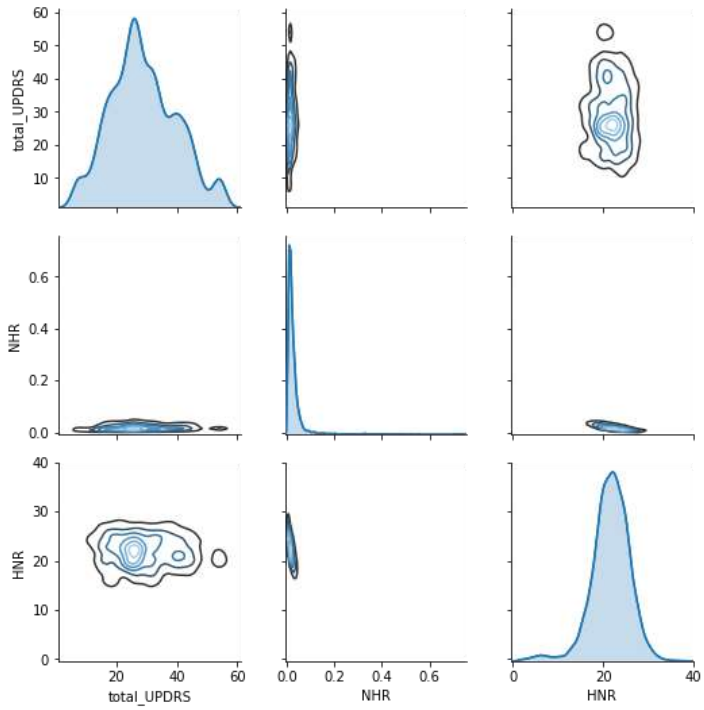


Fig 2. Distribution of multiple features

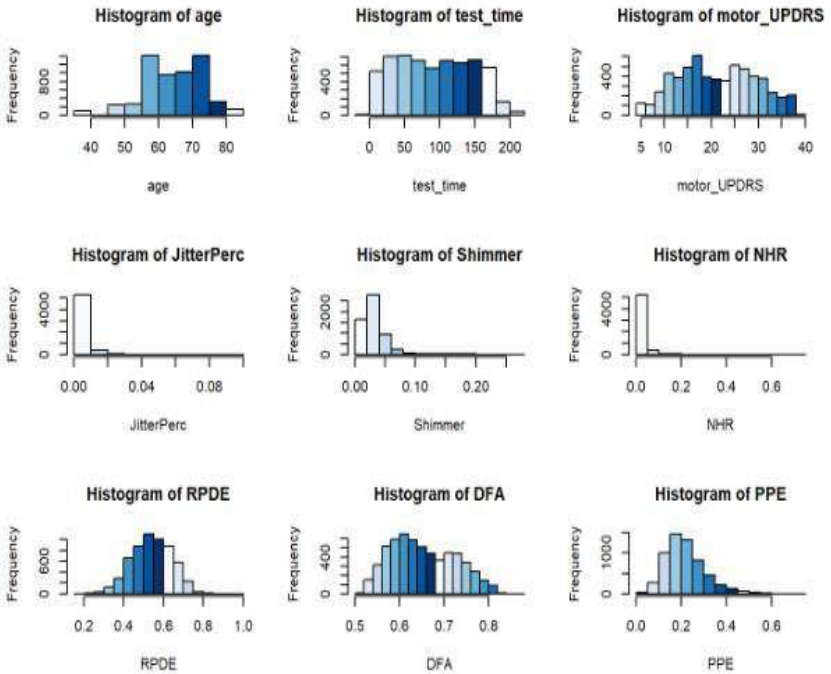


Fig 3: Histogram of multiple features

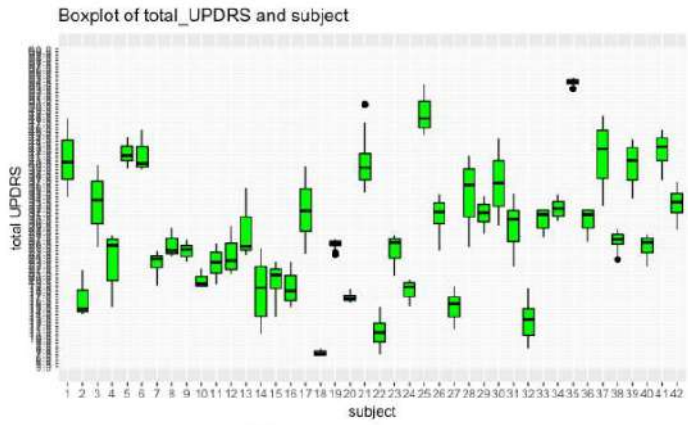


Fig 4.Boxplot of total_UPDRS and subject

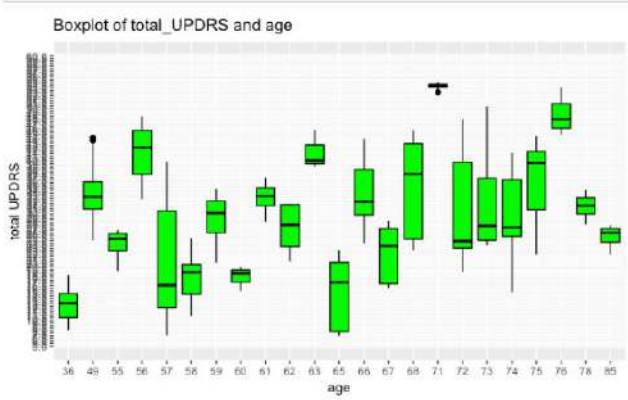


Fig 5.Boxplot of total_UPDRS and age

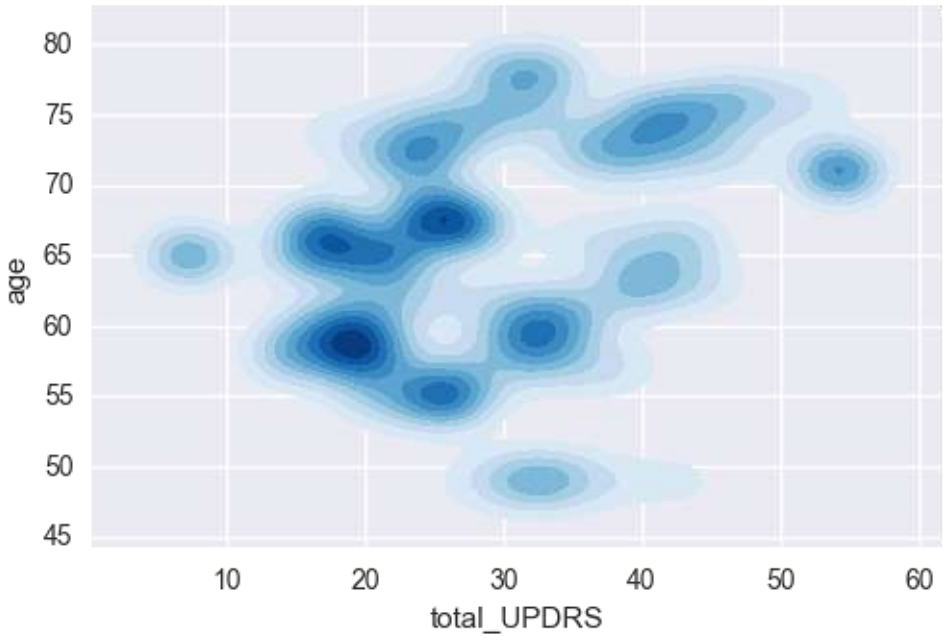


Fig 6. Probability density estimates by age

CHAPTER 7

7.1 APPROACH TO SOLVE THE PROBLEM

[Part 1: Training The Model To Predict total_UPDRS Score]

Step 1: Loading the data in Jupyter Notebook and dropping rows with negative test time (as time can not be negative)

```
In [2]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import statsmodels.api as sm
        import sklearn as s
        from sklearn import datasets,linear_model
        from sklearn.linear_model import lasso_path
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        #from matplotlib import pyplot as plt

        df=pd.read_csv('E:\Project_Data/Data.csv',sep=',',header=0)
        df = df[df.test_time >= 0] #removing data with negative testtime
```

Step 2: Observing 5 Number Statistics of the dataset

```
df.describe()
```

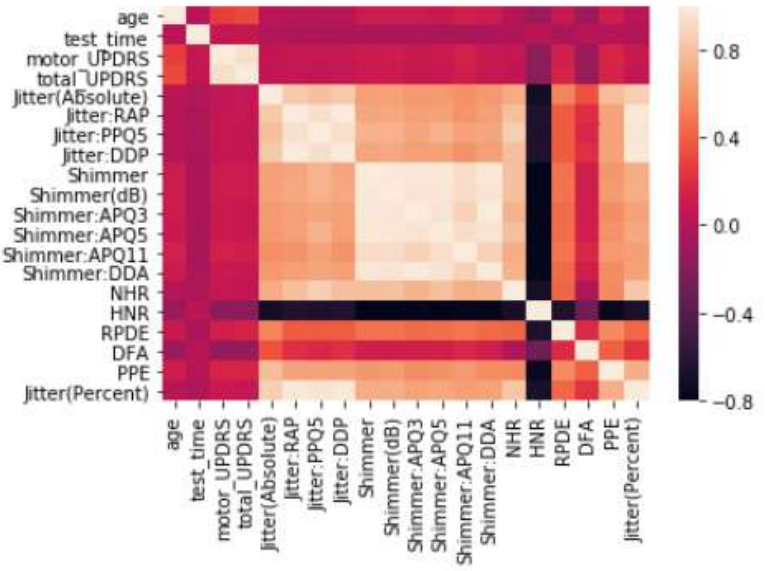
Out[2]:

	age	test_time	motor_UPDRS	total_UPDRS	Jitter(Absolute)	Jitter:RAP	Jitter:PPQ5	Jitter:DDP	Shimmer	Shimmer(dB)	Shimmer:AF
count	5863.000000	5863.000000	5863.000000	5863.000000	5863.000000	5863.000000	5863.000000	5863.000000	5863.000000	5863.000000	5863.000
mean	64.814771	93.061538	21.289655	29.014090	0.000044	0.002988	0.003278	0.008965	0.034038	0.311012	0.017
std	8.827750	53.320896	8.133419	10.709966	0.000036	0.003126	0.003734	0.009378	0.025836	0.230310	0.013
min	36.000000	0.395830	5.037700	7.000000	0.000002	0.000330	0.000430	0.000980	0.003060	0.026000	0.001
25%	58.000000	47.335500	15.000000	21.362000	0.000022	0.001580	0.001825	0.004730	0.019110	0.175000	0.009
50%	65.000000	91.754000	20.871000	27.522000	0.000035	0.002250	0.002490	0.006750	0.027530	0.253000	0.013
75%	72.000000	138.460000	27.594000	36.400500	0.000053	0.003290	0.003470	0.009880	0.039775	0.365000	0.020
max	85.000000	215.490000	39.511000	54.992000	0.000446	0.057540	0.069560	0.172630	0.268630	2.107000	0.162

Step 3. Checking correlation of the features

```
In [21]: sns.heatmap(df.corr())
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x235b6c49908>
```



Step 4. Drop highly correlated features and build Label Set (named X)

```
In [14]: from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import MinMaxScaler
         X = df.drop(["motor_UPDRS","total_UPDRS", "Jitter(Percent)", "Jitter:RAP", "Jitter:PPQ5", "Jitter:DDP", "Shimmer", "Shimmer(dB)",
         X.head()
```

Out[14]:

	age	test_time	Jitter(Absolute)	Shimmer:APQ11	NHR	HNR	RPDE	DFA	PPE
0	72	5.6431	0.000034	0.01662	0.014290	21.640	0.41888	0.54842	0.16006
1	72	12.6660	0.000017	0.01689	0.011112	27.183	0.43493	0.56477	0.10810
2	72	19.6810	0.000025	0.01458	0.020220	23.047	0.46222	0.54405	0.21014
3	72	25.6470	0.000027	0.01963	0.027837	24.445	0.48730	0.57794	0.33277
4	72	33.6420	0.000020	0.01819	0.011625	26.126	0.47188	0.56122	0.19361

Step 5. Take total_UPDRS as target feature (Y)

```
In [15]: #Select target
         Y = df["total_UPDRS"]
         Y.head()
         type(Y)
```

```
Out[15]: pandas.core.series.Series
```

Step 6. Converts target and features to numpy arrays for sklearn API

```
In [16]: # Converts target and features to numpy arrays for sklearn API
         target_y = Y.as_matrix()
         features_x = X.as_matrix()
         type(target_y)
```

```
C:\Users\Namrata\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: Method .as_matrix will be removed in a fut
ure version. Use .values instead.

C:\Users\Namrata\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: FutureWarning: Method .as_matrix will be removed in a fut
ure version. Use .values instead.
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
Out[16]: numpy.ndarray
```

Step 7: Split features_X and target_Y arrays into test and train sets

```
In [22]: # Split into train and test sets
         X_train, X_test, Y_train, Y_test = train_test_split(features_x, target_y, test_size=0.18, random_state=0)

         # Scale features
         min_max_scaler = MinMaxScaler()
         X_train_minmax = min_max_scaler.fit_transform(X_train)
         X_test_minmax = min_max_scaler.fit_transform(X_test)
```

Step 8. Apply Linear Regression Model

```
In [23]: lm = linear_model.LinearRegression() #LinearRegression apply
         model = lm.fit(X_train, Y_train) #actual "Learning" happens here
         predictions = lm.predict(X_test)
         predictions[0:5]

Out[23]: array([30.02862046, 36.64752998, 18.19006808, 26.05842221, 23.71446015])
```

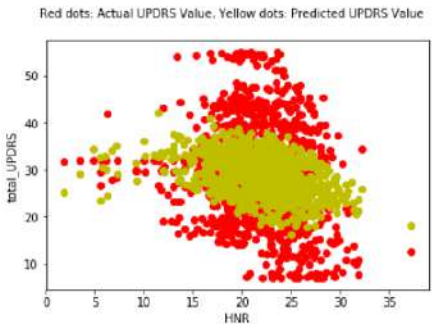
Step 9. Get cross validation R2 Score, RMSE, Mean Absolute Error, R2 Score, Expected Variance Score for Linear Regression Model:

```
In [24]: from sklearn import model_selection
         from sklearn.model_selection import train_test_split,cross_val_score
         lm1=linear_model.LinearRegression()  #LinearRegression
         s_lm=model_selection.cross_val_score(lm1,X_train,Y_train,scoring="r2",cv=5)
         s_lm.mean()

Out[24]: 0.15220514184292485

In [25]: from sklearn.metrics import mean_squared_error
         from math import sqrt #LinearRegressionRMSE

         rms1 = sqrt(mean_squared_error(Y_test, predictions))
         print(rms1)

         9.757927832519483

In [15]: from sklearn import metrics #LinearRegressionMAE
         from sklearn.metrics import mean_absolute_error
         mean_absolute_error(Y_test, predictions)

Out[15]: 8.104121972878216
```

```
In [16]: from sklearn.metrics import r2_score #0 or 1: No fit or Perfect Fit
         r2_score(Y_test, predictions) #Linear RegressionR2score

Out[16]: 0.1579038563445887

In [17]: from sklearn.metrics import explained_variance_score #The best possible score is 1.0, lower values are worse.
         explained_variance_score(Y_test, predictions) #LinearRegressionEVS

Out[17]: 0.15792879767656265
```

Step 10. Plot actual UPDRS values vs predicted UPDRS values

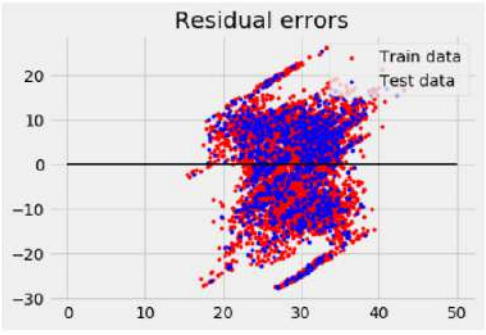
```
In [20]: fig = plt.figure()

plt.plot(test_df_linear['HNR'], test_df_linear['total_UPDRS'],'ro' , label='Train Data')
plt.plot(test_df_linear['HNR'], test_df_linear['total_UPDRS_predicted'],'yo', label='Test Data' )
plt.xlabel('HNR')
plt.ylabel('total_UPDRS')
fig.suptitle('Red dots: Actual UPDRS Value, Yellow dots: Predicted UPDRS Value', fontsize=10)

Out[20]: Text(0.5, 0.98, 'Red dots: Actual UPDRS Value, Yellow dots: Predicted UPDRS Value')
```



Step 11. Plot Residual Errors

```
print('Coefficients: \n', lm.coef_)

# variance score: 1 means perfect prediction
print('Variance score: {}'.format(lm.score(X_test, Y_test)))

# plot for residual error

## setting plot style
plt.style.use('fivethirtyeight')

## plotting residual errors in training data
plt.scatter(lm.predict(X_train), lm.predict(X_train) - Y_train, color = "red", s = 10, label = 'Train data')

## plotting residual errors in test data
plt.scatter(lm.predict(X_test), lm.predict(X_test) - Y_test, color = "blue", s = 10, label = 'Test data')

## plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

## plotting legend
plt.legend(loc = 'upper right')

## plot title
plt.title("Residual errors")

## function to show plot
plt.show()
```

```
Coefficients:
 [ 3.02626081e-01  1.68354285e-02 -2.22543616e+04 -3.50741323e+00
 -2.11411530e+01 -3.58430471e-01  8.35227081e+00 -2.95494146e+01
  2.11731109e+01]
Variance score: 0.157903856344588
```



Step 12. Import Decision Tree Regressor Model

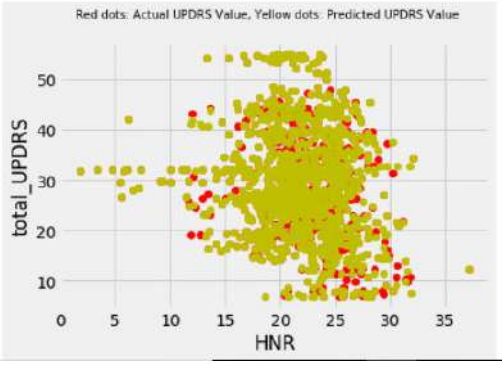
```
In [22]: # import the regressor
         from sklearn.tree import DecisionTreeRegressor #DecisionTreeRegression

         # create a regressor object
         regressor = DecisionTreeRegressor(random_state = 0)

         # fit the regressor with X and Y data
         regressor.fit(X_train, Y_train)
         predictions1 = regressor.predict(X_test)
         predictions1[0:5]

Out[22]: array([42.481, 33.503, 11.496, 19.974, 26.49 ])
```

Step 13. Get cross validation R2 Score, RMSE, Mean Absolute Error, R2 Score, Expected Variance Score for Decision Tree Regression Model:

```
In [23]: from sklearn import tree #DecisionTreeRegression
         tr1=tree.DecisionTreeRegressor()
         s_tr=model_selection.cross_val_score(tr1,X_train,Y_train,scoring="r2",cv=5)
         s_tr.mean()

Out[23]: 0.8746543811360172

In [24]: from sklearn.metrics import mean_squared_error #DecisionTreeRMSE
         from math import sqrt

         rms2 = sqrt(mean_squared_error(Y_test, predictions1))
         print(rms2)

         3.1939128346437964

In [25]: from sklearn import metrics #DecisionTreeRegressionMAE
         from sklearn.metrics import mean_absolute_error
         mean_absolute_error(Y_test, predictions1)

Out[25]: 0.9334752209595963

In [26]: from sklearn.metrics import r2_score #DecisionTreeR2
         r2_score(Y_test, predictions1)

Out[26]: 0.9097821248834737

In [27]: from sklearn.metrics import explained_variance_score #The best possible score is 1.0, lower values are worse.
         explained_variance_score(Y_test, predictions1) #DecisionTreeEVS

Out[27]: 0.9099661158007657
```

Step 15. Plot actual UPDRS values vs predicted UPDRS values

```
In [29]: fig = plt.figure()
         plt.plot(test_df_decisiontree['HNR'], test_df_decisiontree['total_UPDRS'],'ro' )
         plt.plot(test_df_decisiontree['HNR'], test_df_decisiontree['total_UPDRS_predicted'],'yo' )
         plt.xlabel('HNR')
         plt.ylabel('total_UPDRS')
         fig.suptitle('Red dots: Actual UPDRS Value, Yellow dots: Predicted UPDRS Value', fontsize=10)

Out[29]: Text(0.5, 0.98, 'Red dots: Actual UPDRS Value, Yellow dots: Predicted UPDRS Value')
```



Step 16. Plot Residual Errors.

```
In [30]: #print('Coefficients: \n', regressor.coef_)
print('Variance score: {}'.format(regressor.score(X_test, Y_test)))
plt.style.use('fivethirtyeight')
## plotting residual errors in training data
plt.scatter(regressor.predict(X_train), regressor.predict(X_train) - Y_train, color = "red", s = 10, label = 'Train data')

## plotting residual errors in test data
plt.scatter(regressor.predict(X_test), regressor.predict(X_test) - Y_test, color = "yellow", s = 10, label = 'Test data')

## plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

## plotting legend
plt.legend(loc = 'upper right')

## plot title
plt.title("Residual errors")

## function to show plot
plt.show()
```

```
Variance score: 0.9097821248834737
```

Step 17. Import Random Forest Regressor Model

```
In [37]: from sklearn.ensemble import RandomForestRegressor #RandomForestRegression

regressor1 = RandomForestRegressor(n_estimators=800, random_state=1)
regressor1.fit(X_train, Y_train)
predictions2= regressor1.predict(X_test)
predictions2[0:5]
```

```
Out[37]: array([41.38646125, 29.95463875, 12.05914524, 19.91422528, 26.55201975])
```

Step 18. Get cross validation R2 Score, RMSE, Mean Absolute Error, R2 Score, Expected Variance Score for Random Forest Regression Model:

```
In [38]: from sklearn import ensemble
rnf1=ensemble.RandomForestRegressor()
s_rnf=model_selection.cross_val_score(rnf1,X_train,Y_train,scoring="r2",cv=5)
s_rnf.mean()
```

```
Out[38]: 0.9336343049072576
```

```
In [39]: from sklearn.metrics import mean_squared_error
from math import sqrt #RandomForestRMSE

rms3 = sqrt(mean_squared_error(Y_test, predictions2))
print(rms3)
```

```
2.26632628543212
```

```
In [40]: from sklearn import metrics #RandomForestRegressionMAE
from sklearn.metrics import mean_absolute_error
mean_absolute_error(Y_test, predictions2)
```

```
Out[40]: 1.0357832539271992
```

```
In [41]: from sklearn.metrics import r2_score #RandomForestR2Score
r2_score(Y_test, predictions2)
```

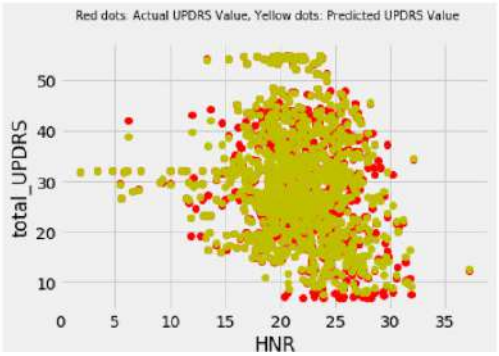
```
Out[41]: 0.9545753754309079
```

```
In [42]: from sklearn.metrics import explained_variance_score #The best possible score is 1.0, lower values are worse.
explained_variance_score(Y_test, predictions2) #Random Forest EVS
```

```
Out[42]: 0.9546079943057417
```

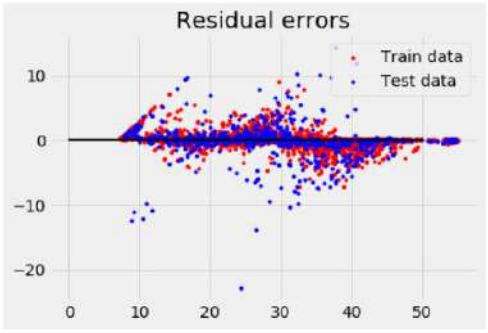
Step 19. Plot actual UPDRS values vs predicted UPDRS values

```
In [38]: fig = plt.figure()
plt.plot(test_df_randomforest['HNR'], test_df_randomforest['total_UPDRS'],'ro' )
plt.plot(test_df_randomforest['HNR'], test_df_randomforest['total_UPDRS_predicted'],'yo' )
plt.xlabel('HNR')
plt.ylabel('total_UPDRS')
fig.suptitle('Red dots: Actual UPDRS Value, Yellow dots: Predicted UPDRS Value', fontsize=10)

Out[38]: Text(0.5, 0.98, 'Red dots: Actual UPDRS Value, Yellow dots: Predicted UPDRS Value')
```



Step 20. Plot Residual Errors

```
In [39]: #print('Coefficients: \n', regressor.coef_)
print('Variance score: {}'.format(regressor1.score(X_test, Y_test)))
plt.style.use('fivethirtyeight')
## plotting residual errors in training data
plt.scatter(regressor1.predict(X_train), regressor1.predict(X_train) - Y_train, color = "red", s = 10, label = 'Train data')

## plotting residual errors in test data
plt.scatter(regressor1.predict(X_test), regressor1.predict(X_test) - Y_test, color = "blue", s = 10, label = 'Test data')

## plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

## plotting legend
plt.legend(loc = 'upper right')

## plot title
plt.title("Residual errors")

## function to show plot
plt.show()

Variance score: 0.9538757887652384
```



Step 21. Import Polynomial Regression

```
In [40]: from sklearn.preprocessing import PolynomialFeatures #PolynomialRegression
         from sklearn.linear_model import LinearRegression
         # PolynomialFeatures (prepreprocessing)
         poly = PolynomialFeatures(degree=2)
         X_ = poly.fit_transform(X_train)
         X_test_ = poly.fit_transform(X_test)
         # Instantiate
         lg = LinearRegression()

         # Fit
         lg.fit(X_, Y_train)

         # Obtain coefficients
         lg.coef_
         # Predict
         predictions3=lg.predict(X_test_)
         predictions3[0:5]

Out[40]: array([31.32175827, 43.45185089, 21.92698288, 25.39307404, 25.03594971])
```

Step 22. Get RMSE, Mean Absolute Error, R2 Score, Expected Variance Score for Polynomial Regression Model:

```
In [41]: from sklearn.metrics import mean_squared_error #PolynomialRMSE
         from math import sqrt

         rms4 = sqrt(mean_squared_error(Y_test, predictions3))
         print(rms4)

         9.131731982907882

In [42]: from sklearn import metrics #PolynomialRegressionMAE
         from sklearn.metrics import mean_absolute_error
         mean_absolute_error(Y_test, predictions3)

Out[42]: 7.537255593155369

In [43]: from sklearn.metrics import r2_score #PolynomialRegressionR2Score
         r2_score(Y_test, predictions3)

Out[43]: 0.26251569855341317

In [44]: from sklearn.metrics import explained_variance_score #The best possible score is 1.0, lower values are worse.
         explained_variance_score(Y_test, predictions3) #Polynomial EVS

Out[44]: 0.26251616748935125
```

Step 23. Import KNN Regressor

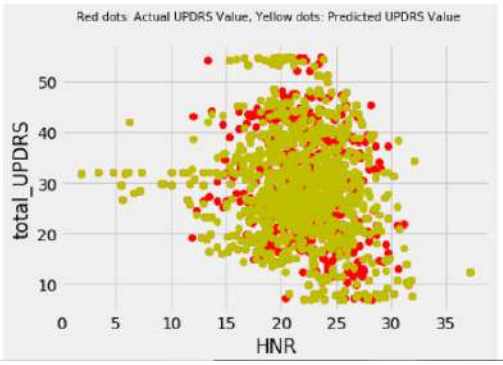
```
In [45]: from sklearn.neighbors import KNeighborsRegressor #KNeighboursRegression
         neigh = KNeighborsRegressor(n_neighbors=2)
         neigh.fit(X_train, Y_train)
         KNeighborsRegressor(...)
         predictions4=(neigh.predict(X_test))
         predictions4[0:5]

Out[45]: array([33.389 , 26.353 , 11.496 , 19.974 , 28.4735])
```

Step 24. Get cross validation R2 Score, RMSE, Mean Absolute Error, R2 Score, Expected Variance Score for KNN Regression Model:

```
In [46]: from sklearn import neighbors #KNeighborsRegression
         knn1=neighbors.KNeighborsRegressor()
         s_knn=model_selection.cross_val_score(knn1,X_train,Y_train,scoring="r2",cv=5)
         s_knn.mean()

Out[46]: 0.4167642613927594

In [47]: from sklearn.metrics import mean_squared_error
         from math import sqrt #KNNRMSE

         rms5 = sqrt(mean_squared_error(Y_test, predictions4))
         print(rms5)

         6.30702370106705

In [48]: from sklearn import metrics #KNNMAE
         from sklearn.metrics import mean_absolute_error
         mean_absolute_error(Y_test, predictions4)

Out[48]: 3.1918665246212123

In [49]: from sklearn.metrics import r2_score #KNNR2Score
         r2_score(Y_test, predictions4)

Out[49]: 0.6482003517480912

In [50]: from sklearn.metrics import explained_variance_score #The best possible score is 1.0, lower values are worse.
         explained_variance_score(Y_test, predictions4) #KNN EVS

Out[50]: 0.6482052498250046
```

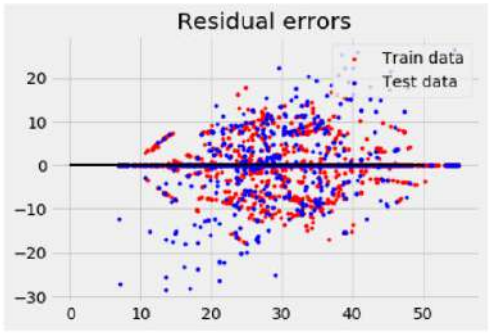
Step 25. Plot actual UPDRS values vs predicted UPDRS values

```
In [52]: fig = plt.figure()
         plt.plot(test_df_neigh['HNR'], test_df_neigh['total_UPDRS'],'ro' )
         plt.plot(test_df_neigh['HNR'], test_df_neigh['total_UPDRS_predicted'],'yo' )
         plt.xlabel('HNR')
         plt.ylabel('total_UPDRS')
         fig.suptitle('Red dots: Actual UPDRS Value, Yellow dots: Predicted UPDRS Value', fontsize=10)

Out[52]: Text(0.5, 0.98, 'Red dots: Actual UPDRS Value, Yellow dots: Predicted UPDRS Value')
```



Step 26. Plot Residual Errors

```
In [53]: #print('Coefficients: \n', regressor.coef_)
         print('Variance score: {}'.format(neigh.score(X_test, Y_test)))
         plt.style.use('fivethirtyeight')
         ## plotting residual errors in training data
         plt.scatter(neigh.predict(X_train), neigh.predict(X_train) - Y_train, color = "red", s = 10, label = 'Train data')

         ## plotting residual errors in test data
         plt.scatter(neigh.predict(X_test), neigh.predict(X_test) - Y_test, color = "blue", s = 10, label = 'Test data')

         ## plotting line for zero residual error
         plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

         ## plotting legend
         plt.legend(loc = 'upper right')

         ## plot title
         plt.title("Residual errors")

         ## function to show plot
         plt.show()

         Variance score: 0.6482003517480912
```



Step 27. Import Support Vector Regressor

```
In [54]: from sklearn.svm import SVR
         from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
         from scipy.stats import randint, expon

         # Hyperparameter optimization:
         # Use randomized search for drastically faster param tuning.
         # TODO: Compare with kernel ridge regression and other algos

         params = {'C': expon(scale=100), 'gamma': expon(scale=.1),
           'kernel': ['linear', 'rbf']}

         svr = SVR()

         n_iter_search = 200
         clf = RandomizedSearchCV(svr, param_distributions=params,
                                  n_iter=n_iter_search)

         clf.fit(X_train_minmax, Y_train)

         clf.best_estimator_

         y_pred = clf.predict(X_test_minmax)
```

Step 28. Get RMSE, Mean Absolute Error, R2 Score, Expected Variance Score for Support Vector Regression Model:

```
In [55]: from sklearn.metrics import mean_squared_error #SVMMAE
         from math import sqrt

         rms6 = sqrt(mean_squared_error(Y_test, y_pred))
         print(rms6)

         9.250723625694544
```

Step 29. Compare different performance evaluation metrics for used models:

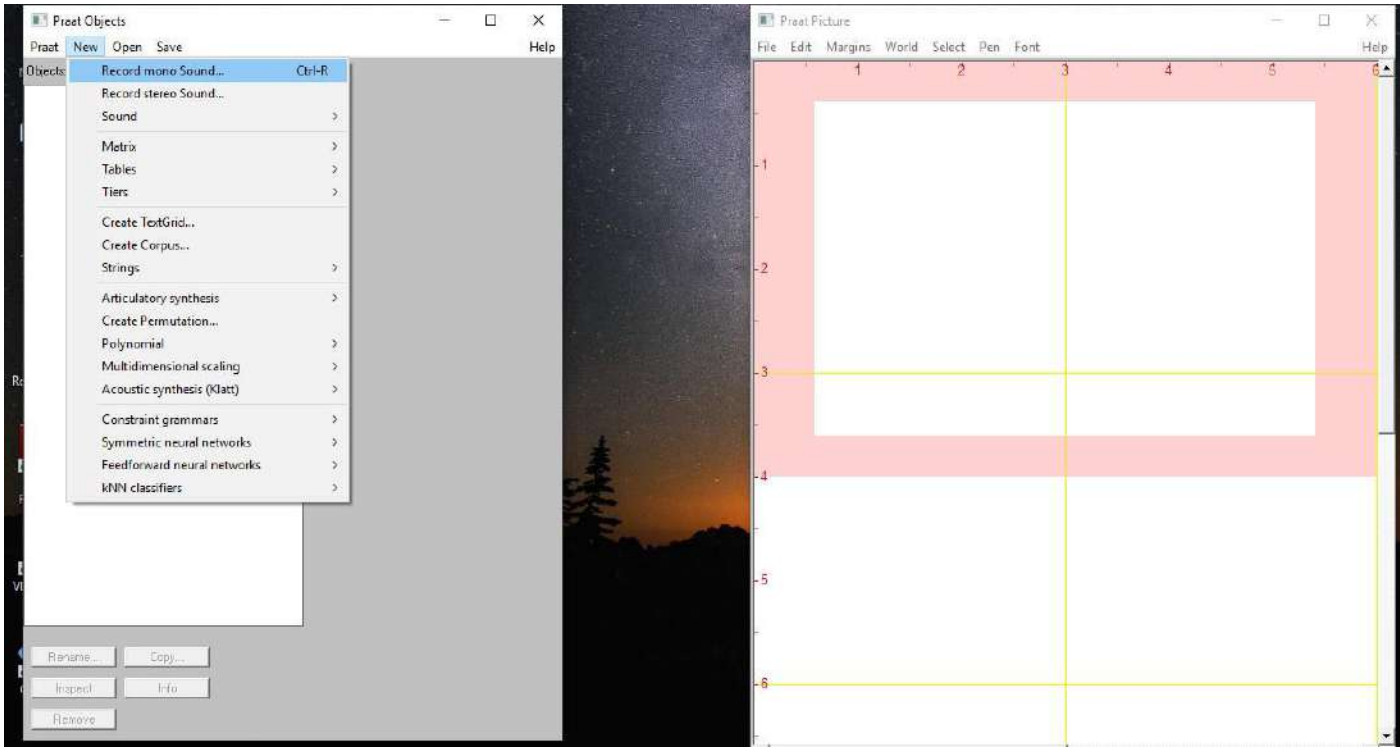
	Ideal Score	Linear Regression	Decision Tree Regression	Random Forest Regression	Polynomial Regression	KNN Regression	Suppo Vecto Regres
Cross-Val r2	1	0.152	0.874	0.933	N/A	0.416	N/A
RMSE	0	9.757	3.193	2.266	9.132	6.307	9.41
Mean Abs Error	0	8.104	0.933	1.035	7.537	3.191	N/A
Expected Var Score	1	0.157	0.909	0.954	0.262	0.648	0.2

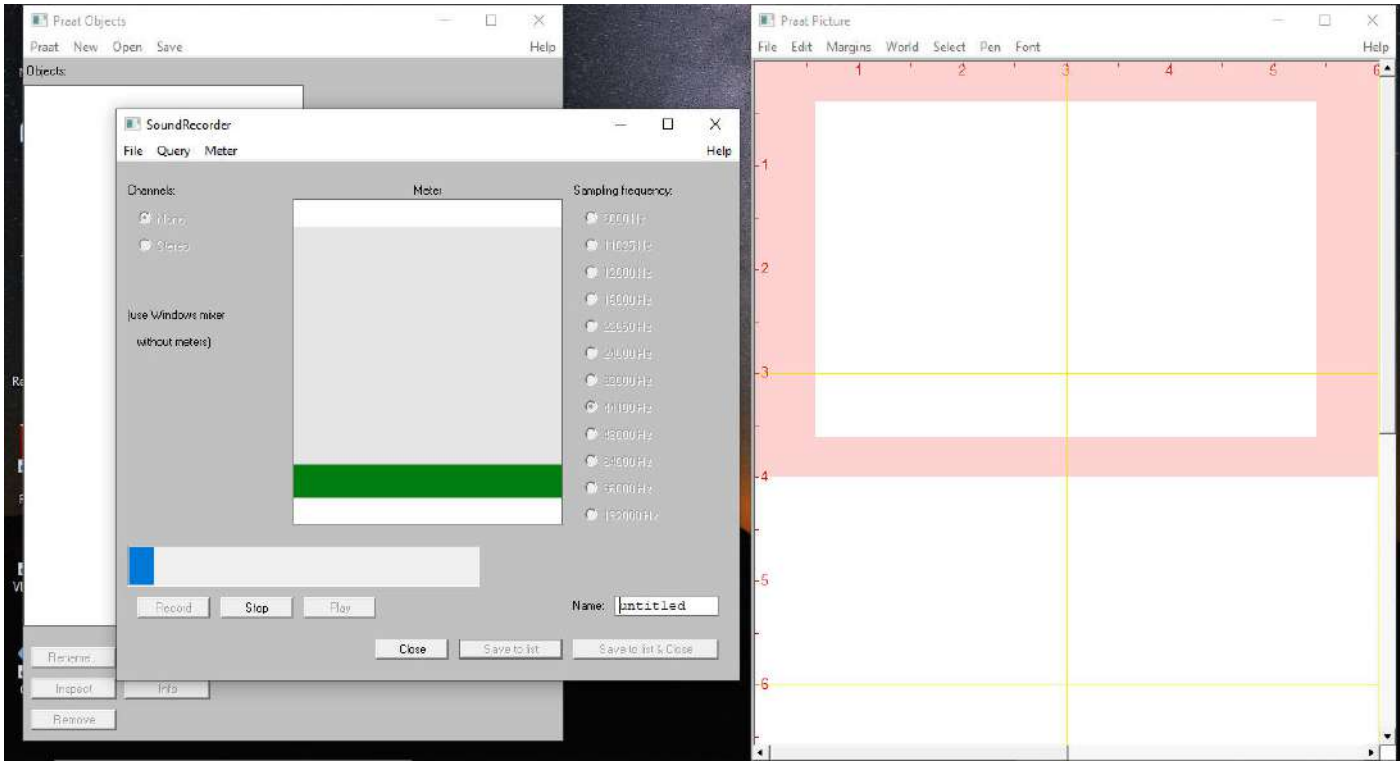
R2 Score	1	0.158	0.909	0.954	0.262	0.648	0.21

Step 30. Considering above table, we conclude Random Forest Regressor is the best suited model to train our dataset.

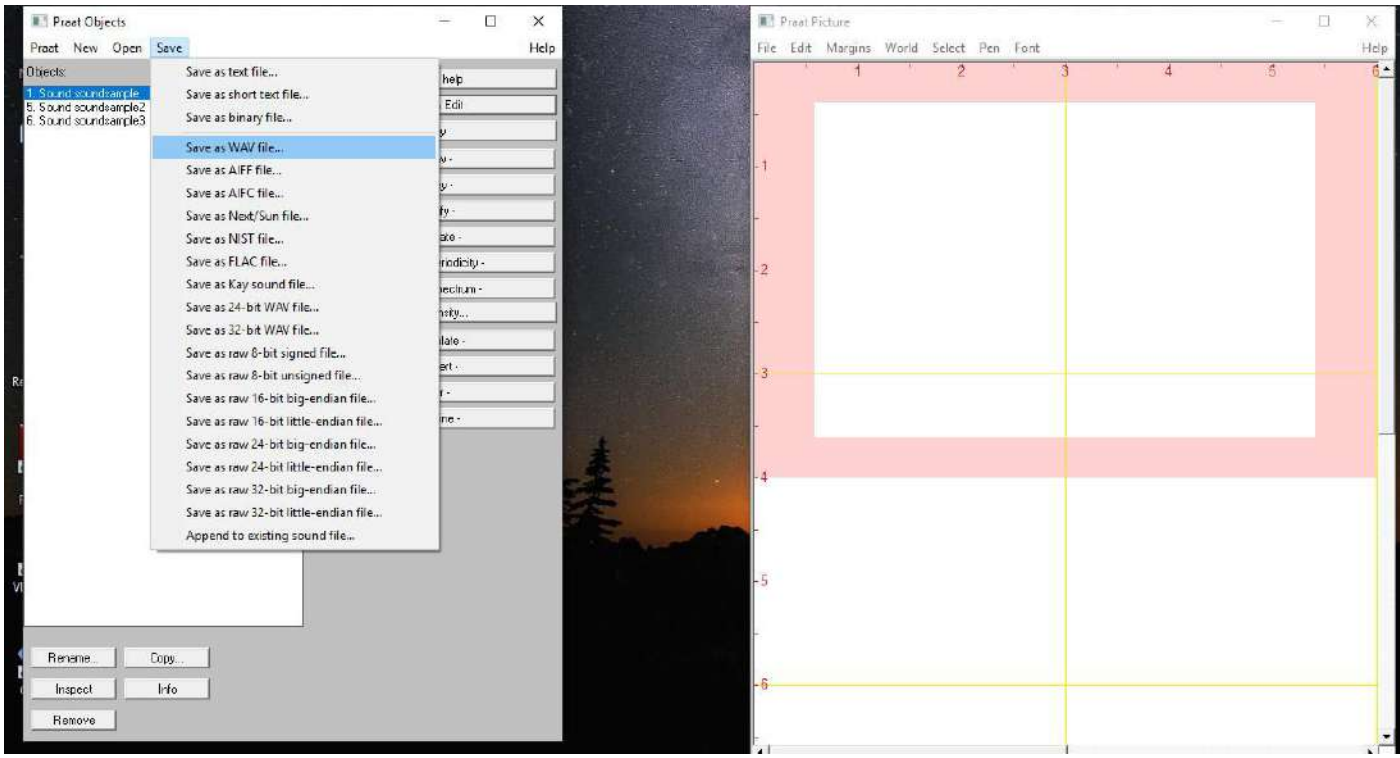
[Part 2: Record And Analyze Audio Sample And Extract Features From Them]

Step 31. Open Praat Software and record audio samples (Minimum Three).





Step 31: Save the recordings as WAV Files.



Step 32. Analyze the recordings and extract features from them using Parselmouth.

In [1]:
```
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
```

In [2]:
```
# 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 object
    meanF0 = call(pitch, "Get mean", 0, 0, unit) # get mean pitch
    stdevF0 = call(pitch, "Get standard deviation", 0 ,0, unit) # get standard deviation
    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, 1.3)
    localabsoluteJitter = call(pointProcess, "Get jitter (local, absolute)", 0, 0, 0.0001, 0.02, 1.3)
    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, 0.0001, 0.02, 1.3, 1.6)
    localdbShimmer = call([sound, pointProcess], "Get shimmer (local_dB)", 0, 0, 0.0001, 0.02, 1.3, 1.6)
    apq3Shimmer = call([sound, pointProcess], "Get shimmer (apq3)", 0, 0, 0.0001, 0.02, 1.3, 1.6)
    aqpq5Shimmer = call([sound, pointProcess], "Get shimmer (apq5)", 0, 0, 0.0001, 0.02, 1.3, 1.6)
    apq11Shimmer =  call([sound, pointProcess], "Get shimmer (apq11)", 0, 0, 0.0001, 0.02, 1.3, 1.6)
    ddaShimmer = call([sound, pointProcess], "Get shimmer (dda)", 0, 0, 0.0001, 0.02, 1.3, 1.6)


    return meanF0, stdevF0, hnr, localJitter, localabsoluteJitter, rapJitter, ppq5Jitter, ddpJitter, localShimmer, localdbShimmer
```

In [3]:
```
def runPCA(df):
    #Z-score the Jitter and Shimmer measurements
    features = ['localJitter', 'localabsoluteJitter', 'rapJitter', 'ppq5Jitter', 'ddpJitter',
                'localShimmer', 'localdbShimmer', 'apq3Shimmer', 'apq5Shimmer', 'apq11Shimmer', 'ddaShimmer']
    # 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(copy=True, iterated_power='auto', n_components=2, random_state=None,svd_solver='full', tol=0.0, whiten=False)
    principalComponents = pca.fit_transform(x)
    #principalComponents - sklearn_pca.fit_transform(x)
    principalDf = pd.DataFrame(data = principalComponents, columns = ['JitterPCA', 'ShimmerPCA'])
    principalDf

    #vals_std - StandardScaler().fit_transform(vals)

    #sklearn_pca = PCA(n_components = 'however many you want')

   # vals_pca = sklearn_pca.fit_transform(vals_std)
    return principalDf
```

In [4]:
```
# 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("*.wav"):
    print(wave_file)
    sound = parselmouth.Sound(wave_file)
    (meanF0, stdevF0, hnr, localJitter, localabsoluteJitter, rapJitter, ppq5Jitter, ddpJitter, localShimmer, localdbShimmer, apq
    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, localJitter_list, localabsoluteJitter_list, ra
                                  columns=['voiceID', 'meanF0Hz', 'stdevF0Hz', 'HNR', 'localJitter', 'localabsoluteJitter', 'rapJit
                                           'ppq5Jitter', 'ddpJitter', 'localShimmer', 'localdbShimmer', 'apq3Shimmer', 'apq5Shimmer
                                           'apq11Shimmer', 'ddaShimmer'])  #add these lists to pandas in the right order
pcaData = runPCA(df)
```

```
In [5]: df
```

Out[5]:

	voiceID	meanF0Hz	stdevF0Hz	HNR	localJitter	localabsoluteJitter	rapJitter	
0	sample1_wav.wav	310.1161252911025	68.96243181258946	13.004998258928454	0.008083761937912146	2.598102578130824e-05	0.0033241090787032197	0.0
1	sample2_wav.wav	335.85990845311613	80.61284107275875	13.433162955442677	0.013214419277260886	3.9687703000667514e-05	0.005863079138177156	0
2	sample3_wav.wav	319.21408637894024	107.40005008143514	9.74627807077706	0.032323396797505746	0.00010200919600344213	0.017727880178727177	0.

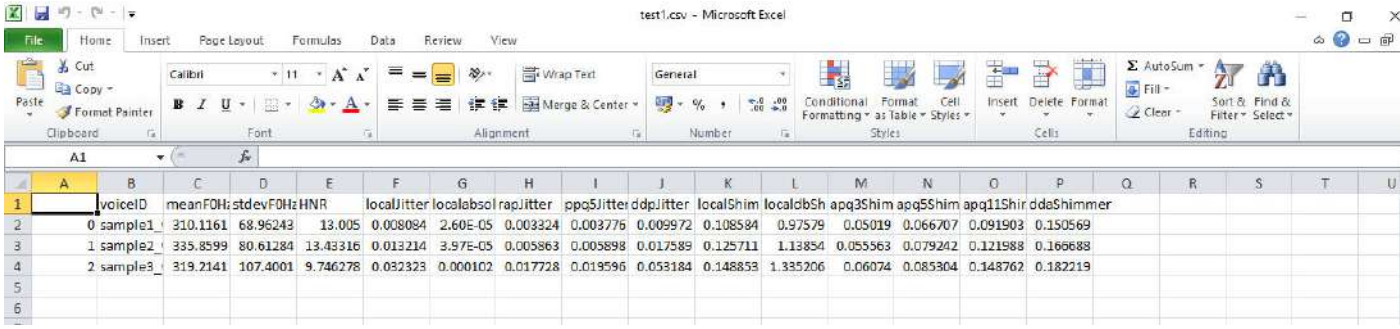
Here, the extracted features are: 'meanF0Hz', 'stdevF0Hz', 'HNR', 'localJitter', 'localabsoluteJitter', 'rapJitter', 'ppq5Jitter', 'ddpJitter', 'localShimmer', 'localdbShimmer', 'apq3Shimmer', 'apq5Shimmer', 'apq11Shimmer', 'ddaShimmer'.

Step 33. Write the results to a CSV file further use

```
In [8]: df.to_csv("test1.csv")
        #read it back
        pd.read_csv("test1.csv").head()
```

Out[8]:

	Unnamed: 0	voiceID	meanF0Hz	stdevF0Hz	HNR	localJitter	localabsoluteJitter	rapJitter	ppq5Jitter	ddpJitter	localShimmer	localdbShimme
0	0	sample1_wav.wav	310.116125	68.962432	13.004998	0.008084	0.000026	0.003324	0.003776	0.009972	0.108584	0.97579
1	1	sample2_wav.wav	335.859908	80.612841	13.433163	0.013214	0.000040	0.005863	0.005898	0.017589	0.125711	1.13854
2	2	sample3_wav.wav	319.214086	107.400050	9.746278	0.032323	0.000102	0.017728	0.019596	0.053184	0.148853	1.33520



[Part 3: Predict NHR, RPDE, DFA, PPE From Previously Extracted Features]

Step 34: In our Random Forest Regression Model to predict total_UPDRS, among various voice acoustic measures, we have used Jitter(Absolute), ShimmerAPQ11, HNR, NHR, RPDE, DFA, PPE. Among these. From Part 2, we have extracted the values of Jitter(Absolute) ('localabsoluteJitter'), ShimmerAPQ11 ('apq11Shimmer'), HNR ('HNR'). To predict NHR, RPDE, DFA, PPE, we shall use Machine Learning Models again.

Step 35: Predict the value of NHR:

a. Label: age, test_time, Jitter(Abs), ShimmerAPQ11, HNR
Target: NHR

b. Split the original dataset into test and train data.

c. Import different Regression Models.

d. After comparing different performance metrics for these Regression Models, we conclude Decision Tree Regressor is best-suited to predict NHR

```
In [23]: # import the regressor
         from sklearn.tree import DecisionTreeRegressor #DecisionTreeRegression

         # create a regressor object
         regressor = DecisionTreeRegressor(random_state = 0)

         # fit the regressor with X and Y data
         regressor.fit(X_train, Y_train)
         predictions1 = regressor.predict(X_test)
         predictions1[0:5]

Out[23]: array([0.0156445, 0.02084  , 0.017767 , 0.019494 , 0.005883 ])

In [24]: from sklearn import tree #DecisionTreeRegression
         tr1=tree.DecisionTreeRegressor()
         s_tr=model_selection.cross_val_score(tr1,X_train,Y_train,scoring="r2",cv=5)
         s_tr.mean()

Out[24]: 0.862575721856372

In [25]: from sklearn.metrics import r2_score #DecisionTreeR2
         r2_score(Y_test, predictions1)

Out[25]: 0.8436806701481143

In [31]: from sklearn.metrics import mean_squared_error #DecisionTreeRMSE
         from math import sqrt

         rms2 = sqrt(mean_squared_error(Y_test, predictions1))
         print(rms2)

         0.02087179943941631

In [32]: from sklearn import metrics #DecisionTreeRegressionMAE
         from sklearn.metrics import mean_absolute_error
         mean_absolute_error(Y_test, predictions1)

Out[32]: 0.01115069700689935

In [33]: from sklearn.metrics import explained_variance_score #The best possible score is 1.0, lower values are worse.
         explained_variance_score(Y_test, predictions1) #DecisionTreeEVS

Out[33]: 0.8436835951666449
```

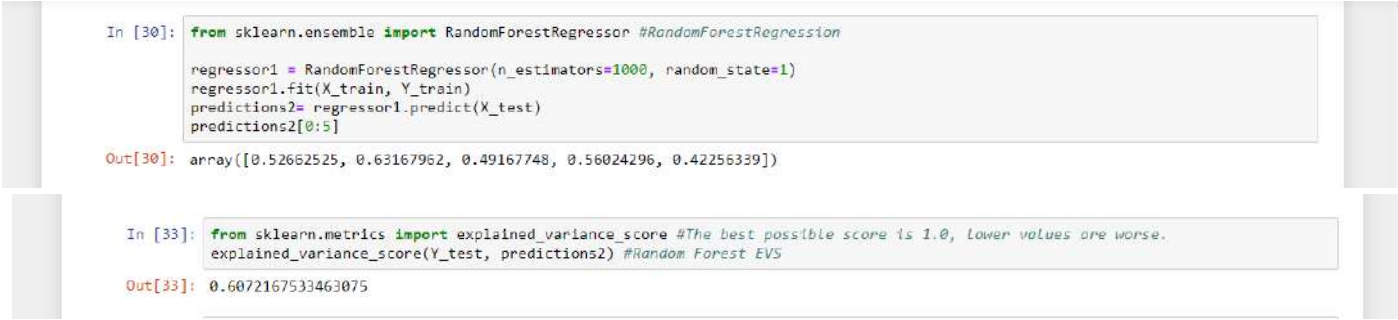
Step 36: Predict the value of RPDE:

a. Label: age, test_time, Jitter(Abs), ShimmerAPQ11, HNR
Target: RPDE

b. Split the original dataset into test and train data.

c. Import different Regression Models.

d. After comparing different performance metrics for these Regression Models, we conclude Random Forest Regressor is best-suited to predict RPDE.



```
In [30]: from sklearn.ensemble import RandomForestRegressor #RandomForestRegression

         regressor1 = RandomForestRegressor(n_estimators=1000, random_state=1)
         regressor1.fit(X_train, Y_train)
         predictions2= regressor1.predict(X_test)
         predictions2[0:5]

Out[30]: array([0.52662525, 0.63167962, 0.49167748, 0.56024296, 0.42256339])
```



```
In [33]: from sklearn.metrics import explained_variance_score #The best possible score is 1.0, lower values are worse.
         explained_variance_score(Y_test, predictions2) #Random Forest EVS

Out[33]: 0.6072167533463075
```

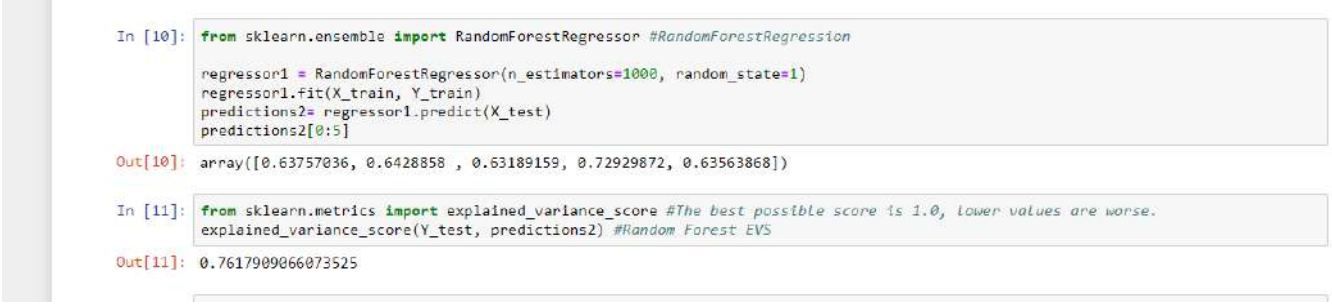
Step 37: Predict the value of DFA:

a. Label: age, test_time, Jitter(Abs), ShimmerAPQ11, HNR
Target: DFA

b. Split the original dataset into test and train data.

c. Import different Regression Models.

d. After comparing different performance metrics for these Regression Models, we conclude Random Forest Regressor is best-suited to predict DFA.



```
In [10]: from sklearn.ensemble import RandomForestRegressor #RandomForestRegression

         regressor1 = RandomForestRegressor(n_estimators=1000, random_state=1)
         regressor1.fit(X_train, Y_train)
         predictions2= regressor1.predict(X_test)
         predictions2[0:5]

Out[10]: array([0.63757036, 0.6428858 , 0.63189159, 0.72929872, 0.63563868])
```



```
In [11]: from sklearn.metrics import explained_variance_score #The best possible score is 1.0, lower values are worse.
         explained_variance_score(Y_test, predictions2) #Random Forest EVS

Out[11]: 0.7617909066073525
```

Step 38: Predict the value of PPE:

a. Label: age, test_time, Jitter(Abs), ShimmerAPQ11, HNR
Target: PPE

b. Split the original dataset into test and train data.

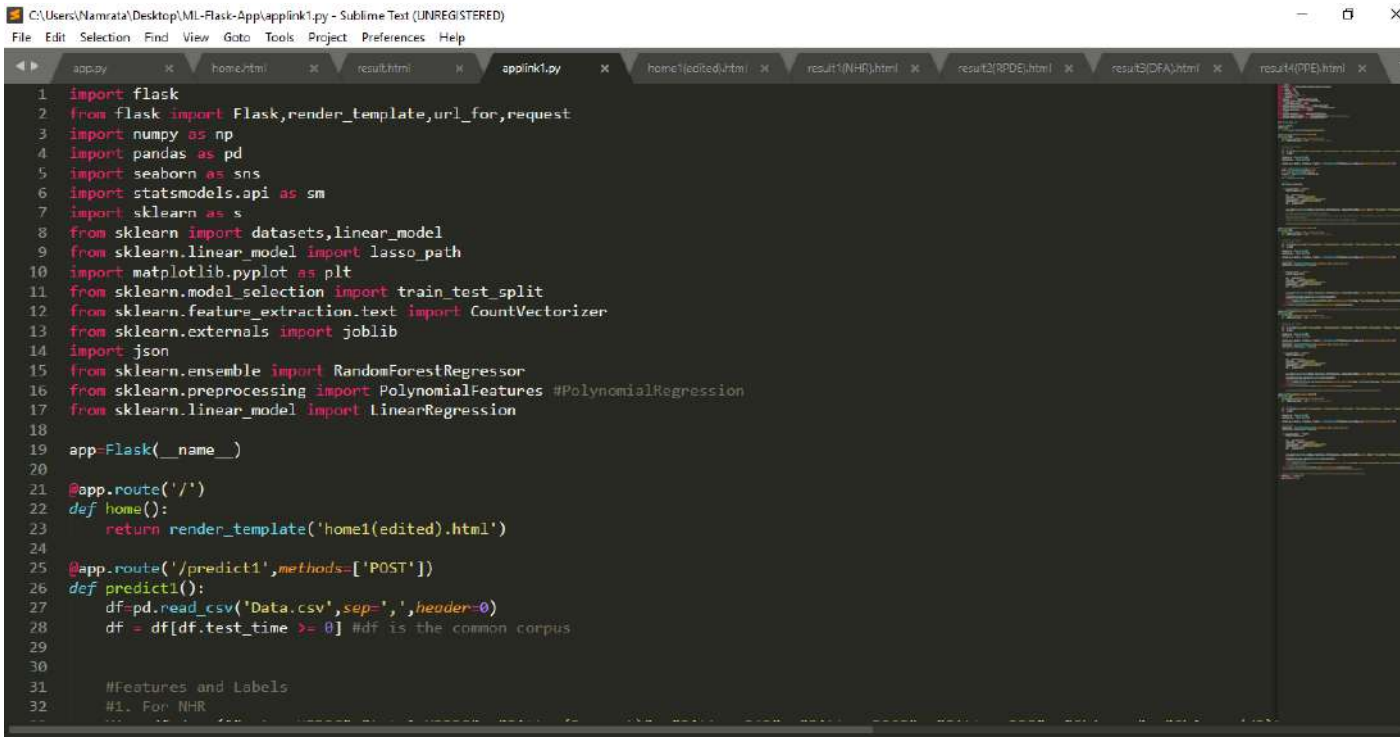
c. Import different Regression Models.

d. After comparing different performance metrics for these Regression Models, we conclude Random Forest Regressor is best-suited to predict PPE.

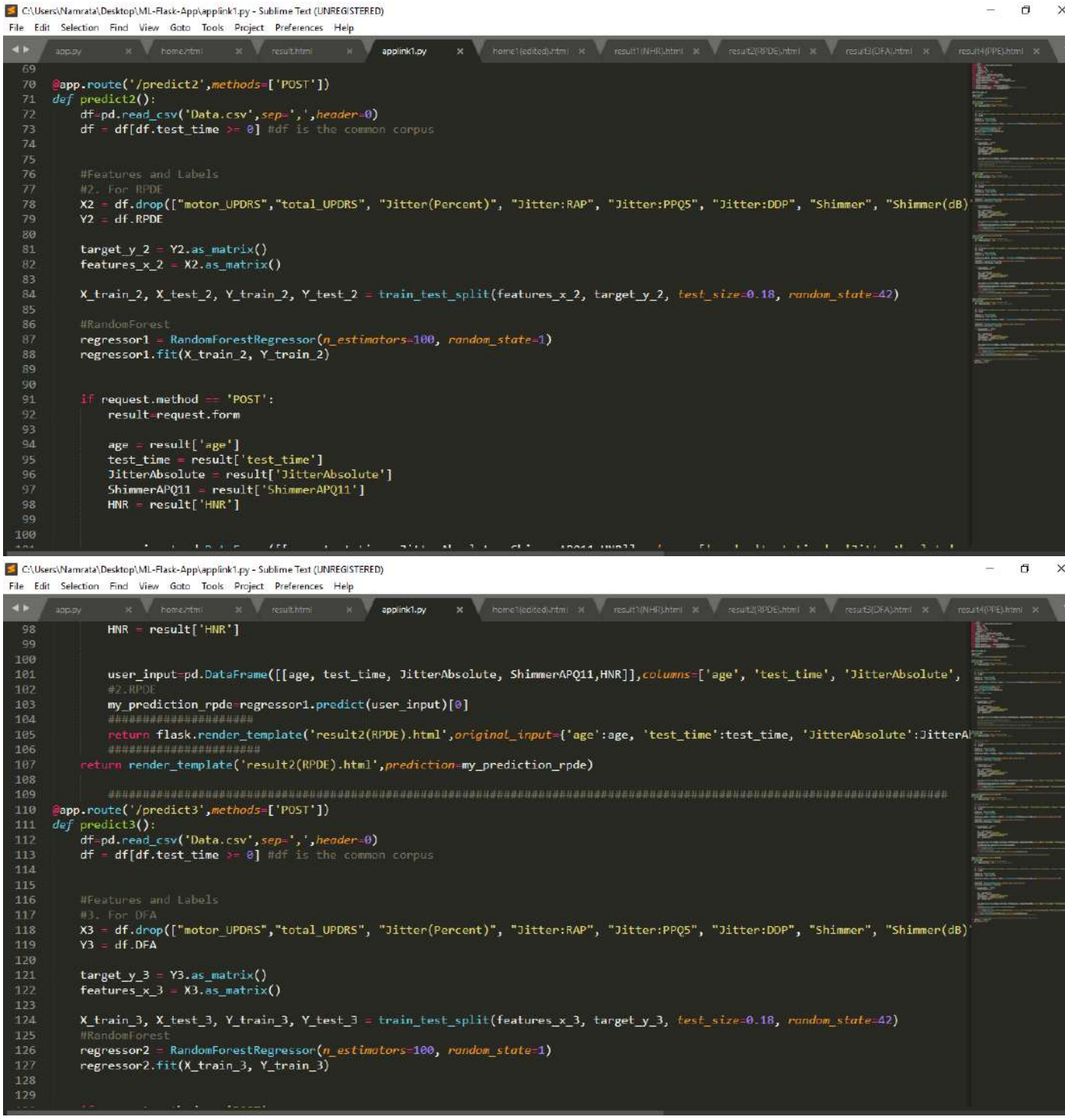
```
In [10]: from sklearn.ensemble import RandomForestRegressor #RandomForestRegression

         regressor1 = RandomForestRegressor(n_estimators=1000, random_state=1)
         regressor1.fit(X_train, Y_train)
         predictions2= regressor1.predict(X_test)
         predictions2[0:5]

Out[10]: array([0.2032783 , 0.18235955, 0.17454123, 0.21193419, 0.11750948])

In [11]: from sklearn.metrics import explained_variance_score #The best possible score is 1.0, lower values are worse.
         explained_variance_score(Y_test, predictions2) #Random Forest EVS

Out[11]: 0.7706663073824689
```

[Part Four: Deploying Model To Predict Features Via Flask]

Step 39. Create a python flask and use micro framework known as Flask to deploy the model in local host.



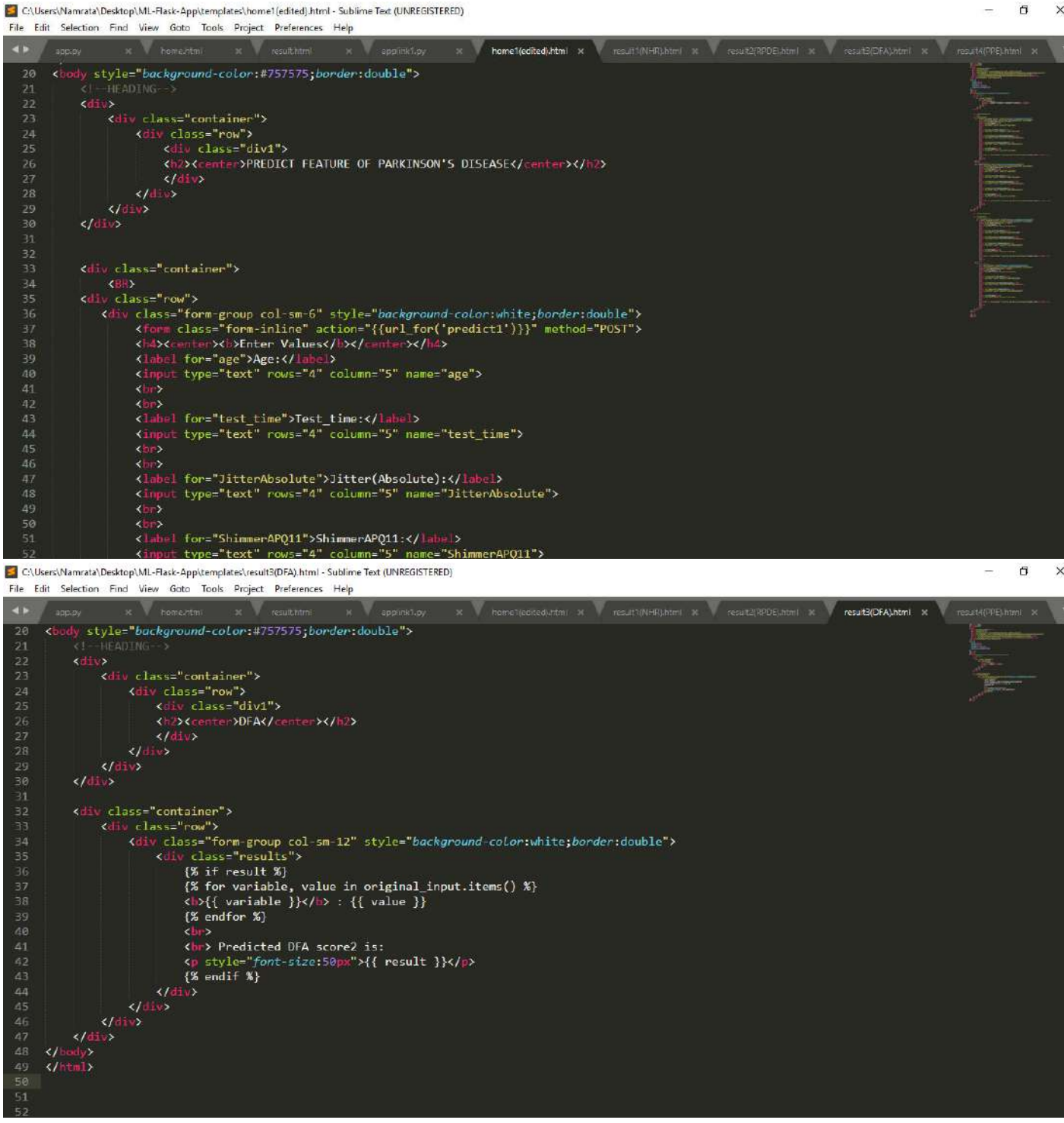
Step 40. Use Previously decided Regression models to predict values of NHR(Decision Tree), RPDE, DFA and PPE (Random Forest Regression).



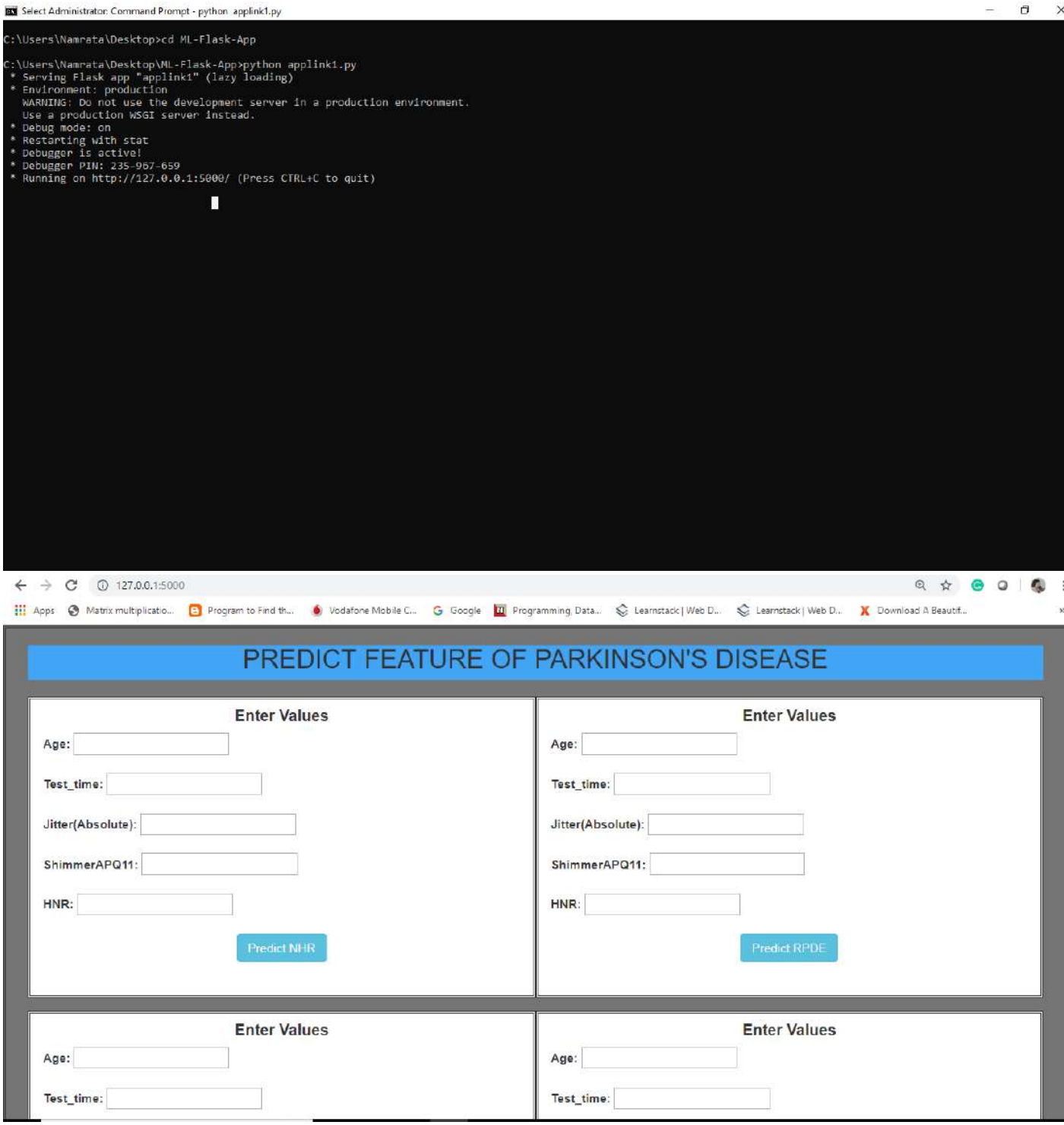
```
@app.route('/predict2',methods=['POST'])
def predict2():
    df=pd.read_csv('Data.csv',sep=',',header=0)
    df = df[df.test_time >= 0] #df is the common corpus


    #Features and Labels
    #2. For RPDE
    X2 = df.drop(["motor_UPDRS","total_UPDRS", "Jitter(Percent)", "Jitter:RAP", "Jitter:PPQ5", "Jitter:DDP", "Shimmer", "Shimmer(dB)"
    Y2 = df.RPDE

    target_y_2 = Y2.as_matrix()
    features_x_2 = X2.as_matrix()

    X_train_2, X_test_2, Y_train_2, Y_test_2 = train_test_split(features_x_2, target_y_2, test_size=0.18, random_state=42)

    #RandomForest
    regressor1 = RandomForestRegressor(n_estimators=100, random_state=1)
    regressor1.fit(X_train_2, Y_train_2)


    if request.method == 'POST':
        result=request.form

        age = result['age']
        test_time = result['test_time']
        JitterAbsolute = result['JitterAbsolute']
        ShimmerAPQ11 = result['ShimmerAPQ11']
        HNR = result['HNR']
```

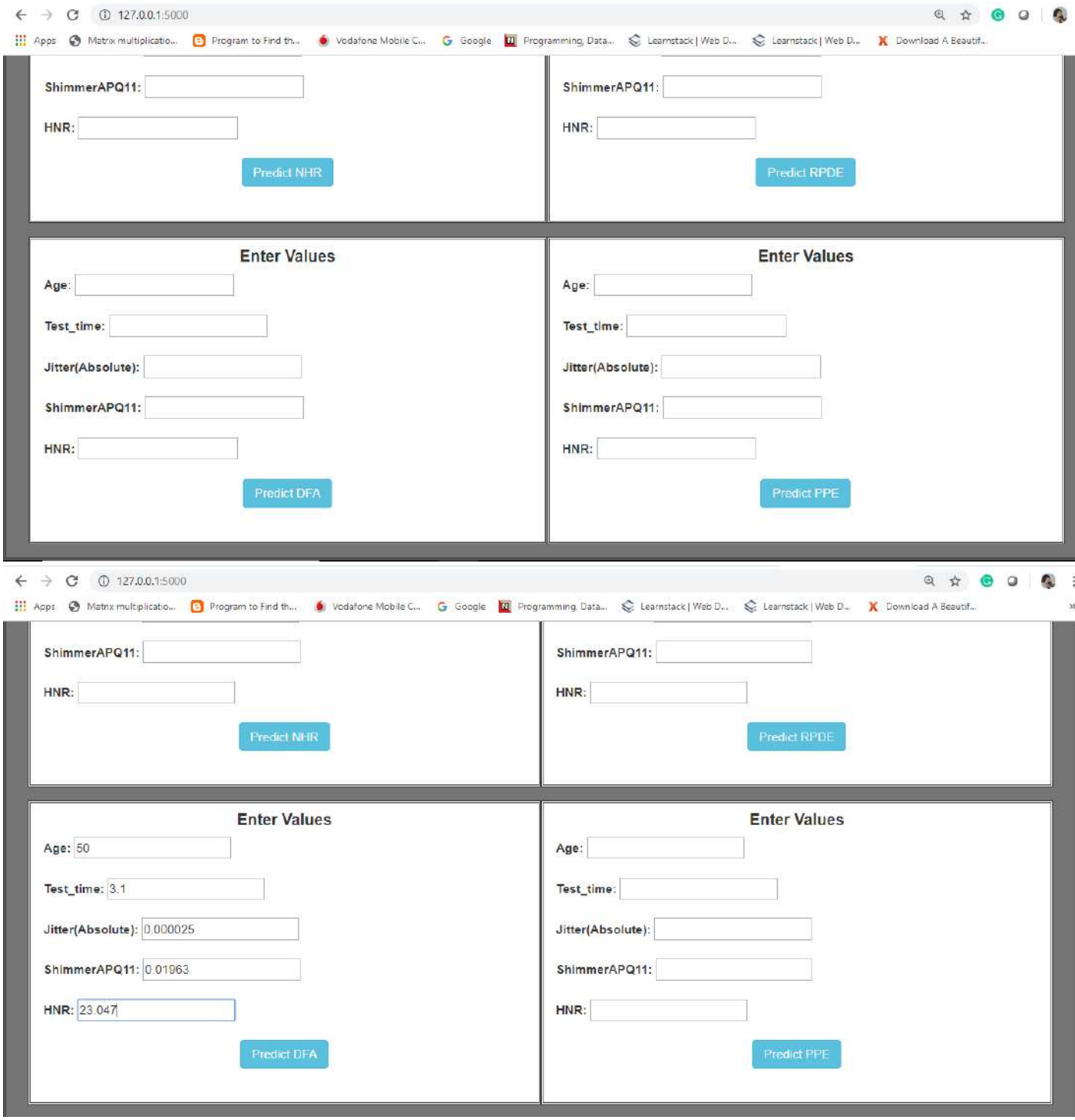
```
        HNR = result['HNR']


        user_input=pd.DataFrame([[age, test_time, JitterAbsolute, ShimmerAPQ11,HNR]],columns=['age', 'test_time', 'JitterAbsolute',
        #2.RPDE
        my_prediction_rpde=regressor1.predict(user_input)[0]
        ######################
        return flask.render_template('result2(RPDE).html',original_input={'age':age, 'test_time':test_time, 'JitterAbsolute':JitterA
        ######################
    return render_template('result2(RPDE).html',prediction=my_prediction_rpde)

        ##########################################################################################################################
@app.route('/predict3',methods=['POST'])
def predict3():
    df=pd.read_csv('Data.csv',sep=',',header=0)
    df = df[df.test_time >= 0] #df is the common corpus


    #Features and Labels
    #3. For DFA
    X3 = df.drop(["motor_UPDRS","total_UPDRS", "Jitter(Percent)", "Jitter:RAP", "Jitter:PPQ5", "Jitter:DDP", "Shimmer", "Shimmer(dB)"
    Y3 = df.DFA

    target_y_3 = Y3.as_matrix()
    features_x_3 = X3.as_matrix()

    X_train_3, X_test_3, Y_train_3, Y_test_3 = train_test_split(features_x_3, target_y_3, test_size=0.18, random_state=42)
    #RandomForest
    regressor2 = RandomForestRegressor(n_estimators=100, random_state=1)
    regressor2.fit(X_train_3, Y_train_3)
```

Step 41. Link a user input page and a result page using HTML, CSS, Bootstrap in the python app. Create forms in user input page where a person can enter values and by clicking on predict button, see desired output on result page. Result page also uses jinja2.

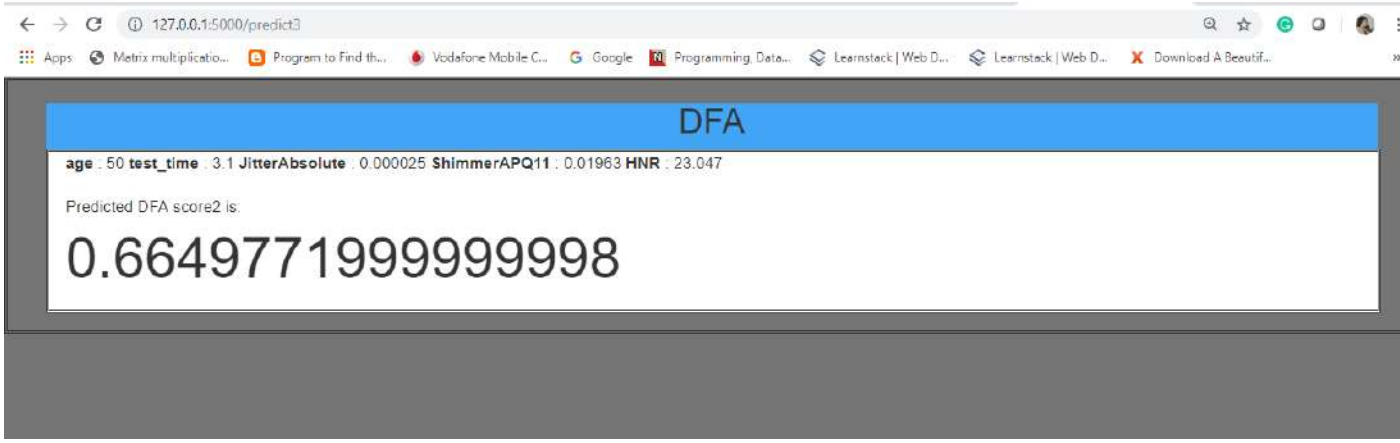


Step 42: Go to CMD and run the file on a localhost by using “python appname.py” command. See the user input on Localhost (http://127.0.0.1:5000/).


Select Administrator: Command Prompt - python applink1.py
C:\Users\Namrata\Desktop>cd ML-Flask-App
C:\Users\Namrata\Desktop\ML-Flask-App>python applink1.py
* Serving Flask app "applink1" (lazy loading)
* Environment: production
WARNING: Do not use the development server in a production environment.
Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 235-967-659
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1:5000
Apps
Matrix multiplicatio...
Program to Find th...
Vodafone Mobile C...
Google
Programming Data...
Learnstack | Web D...
Learnstack | Web D...
Download A Beautif...
PREDICT FEATURE OF PARKINSON'S DISEASE
Enter Values
Age:
Test_time:
Jitter(Absolute):
ShimmerAPQ11:
HNR:
Predict NHR
Enter Values
Age:
Test_time:
Jitter(Absolute):
ShimmerAPQ11:
HNR:
Predict RPDE
Enter Values
Age:
Test_time:
Enter Values
Age:
Test_time:

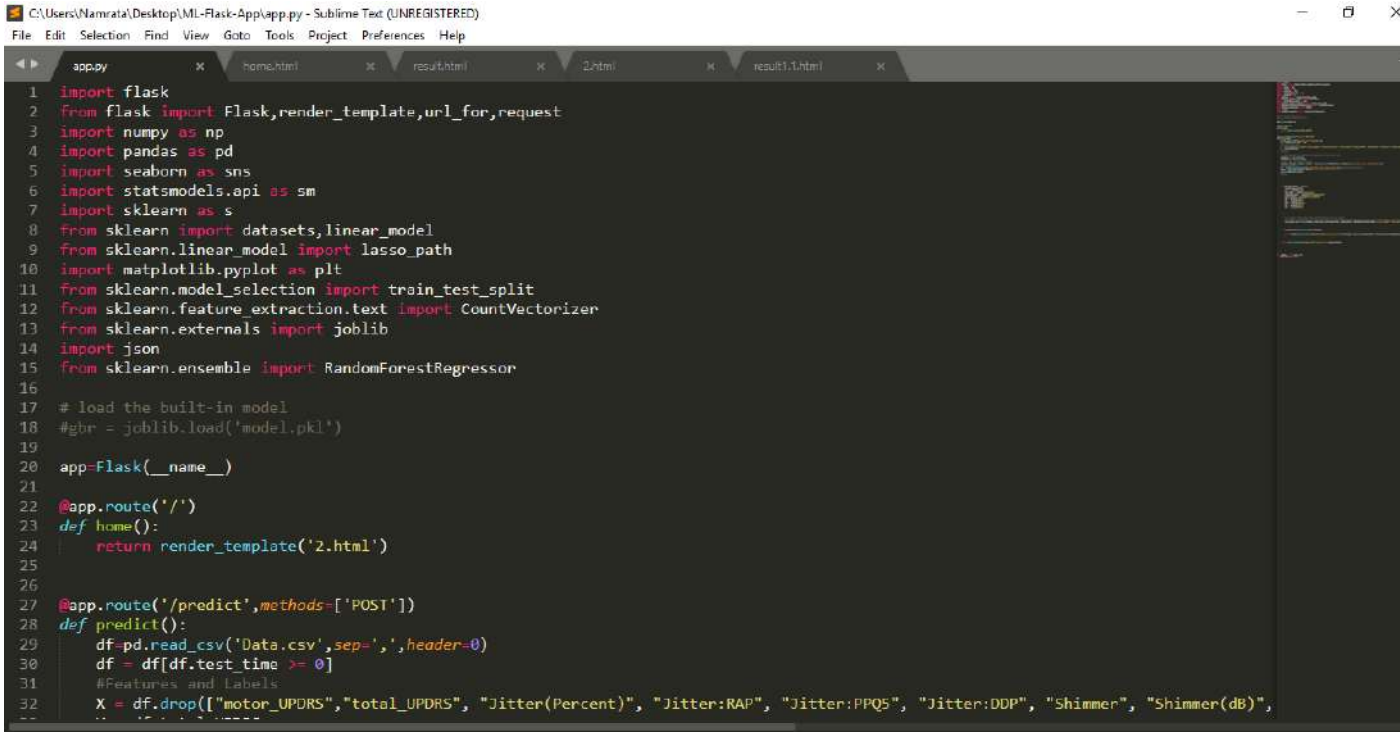


After analysing voice samples, calculate value of Jitter(Absolute), ShimmerAPQ11 and HNR as explained in Step 32 and 33. Enter those values as input parameters and get NHR/RPDE/DFA/PPE scores.

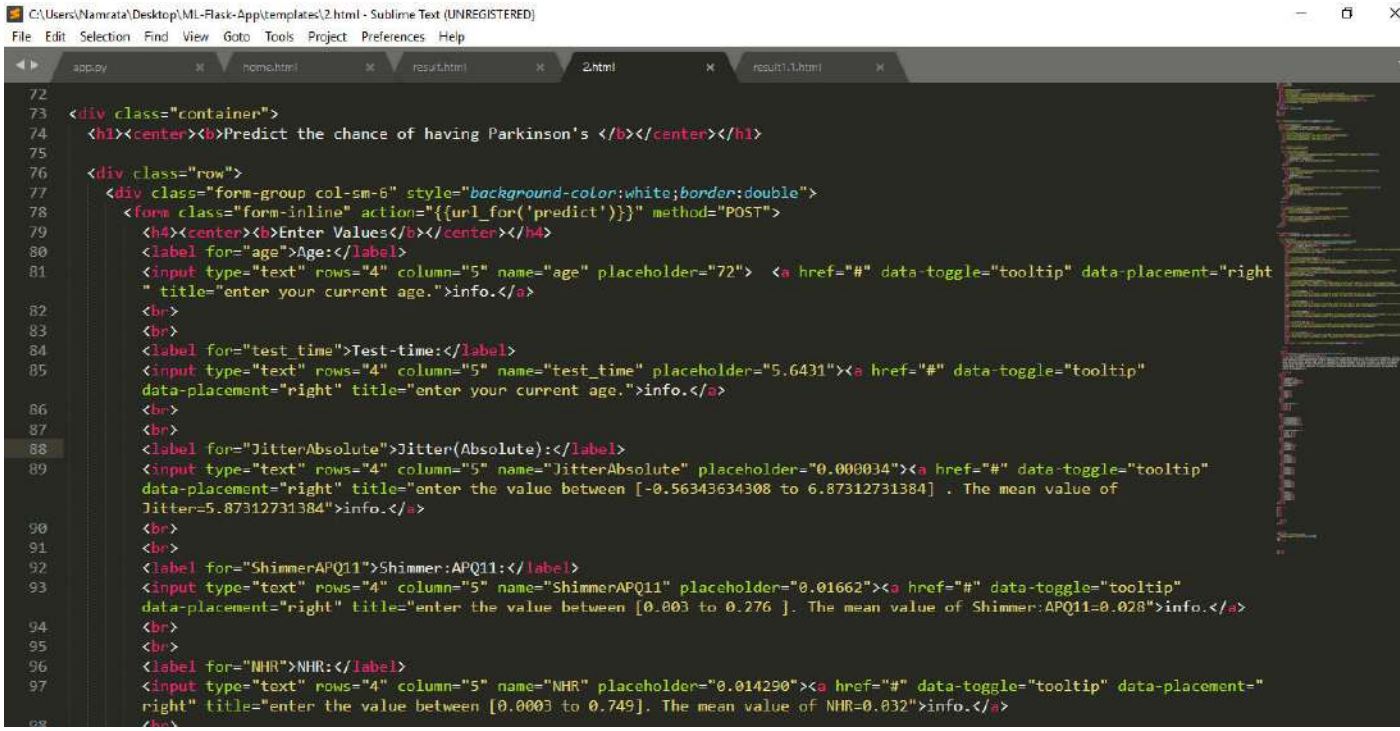


[Part Five: Note The Feature Values And Give Them As Inputs In A New Form And Get total_UPDRS Score (Target Score)]

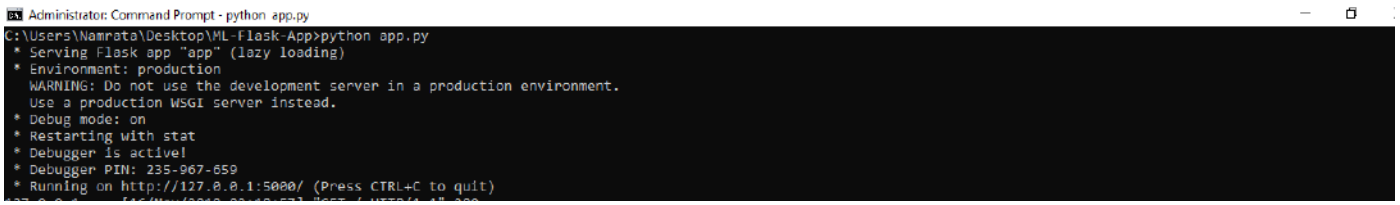
Step 43: Create a python flask and use micro framework known as Flask to deploy the model in local host.



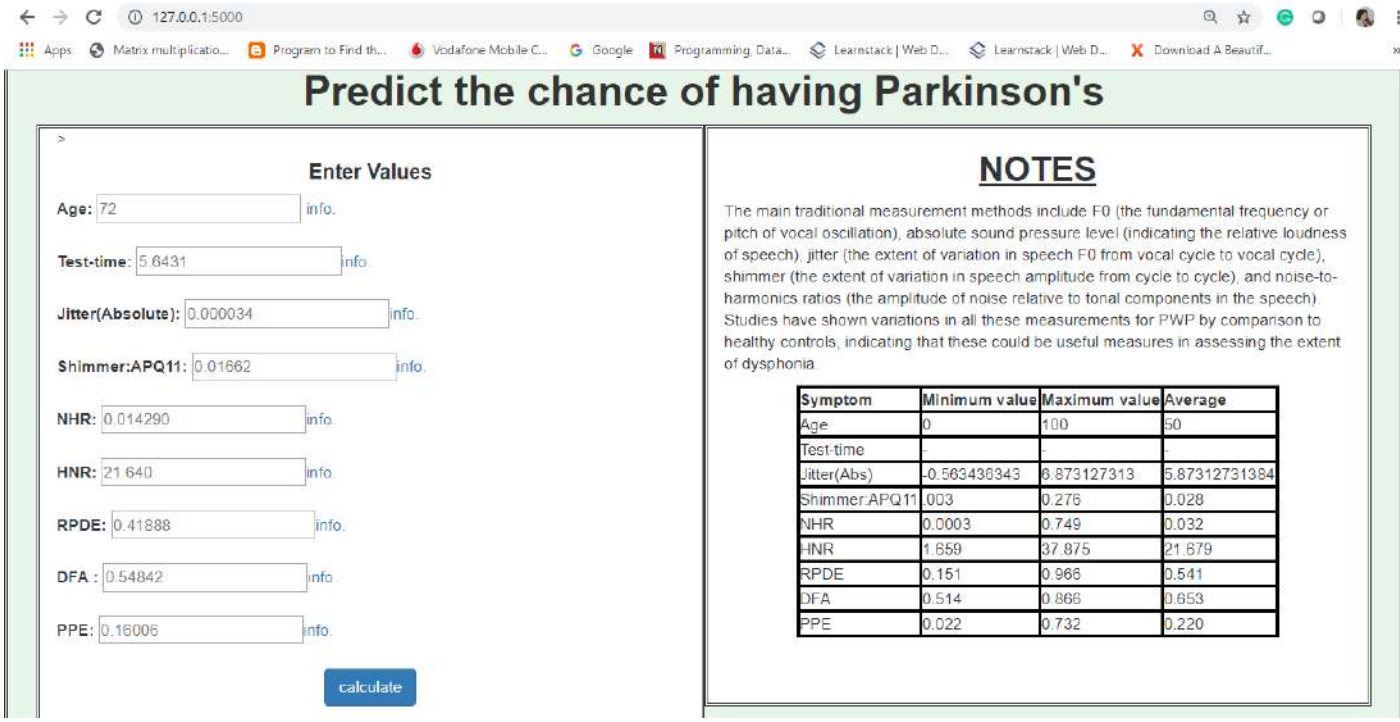
Step 44: Create Home page using where user gives inputs, and create a result page, both using html and link them with the python file mentioned in previous step. Result page also uses jinja2.

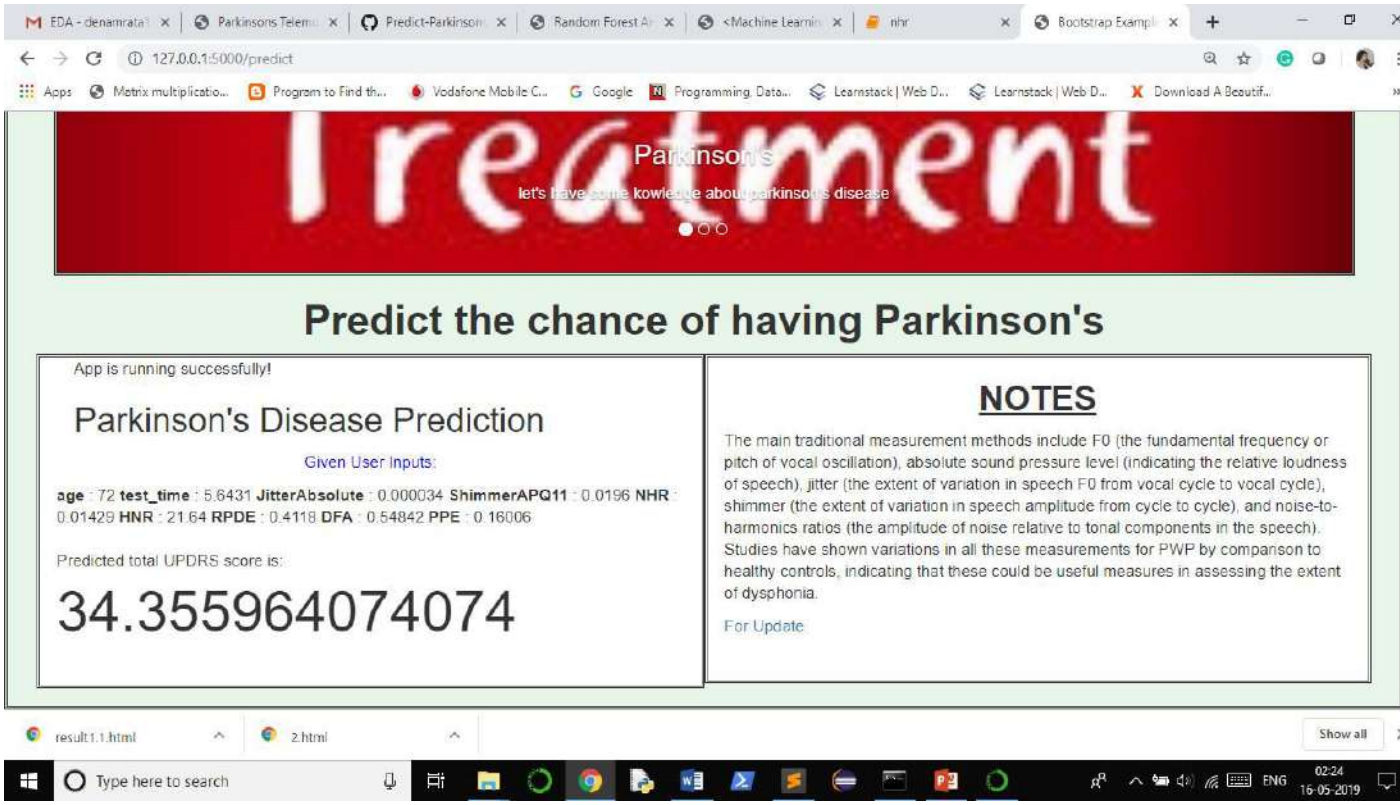


Step 45: Go to CMD and run the file on a localhost by using “python appname.py” command. See the user input on Localhost (http://127.0.0.1:5000/).



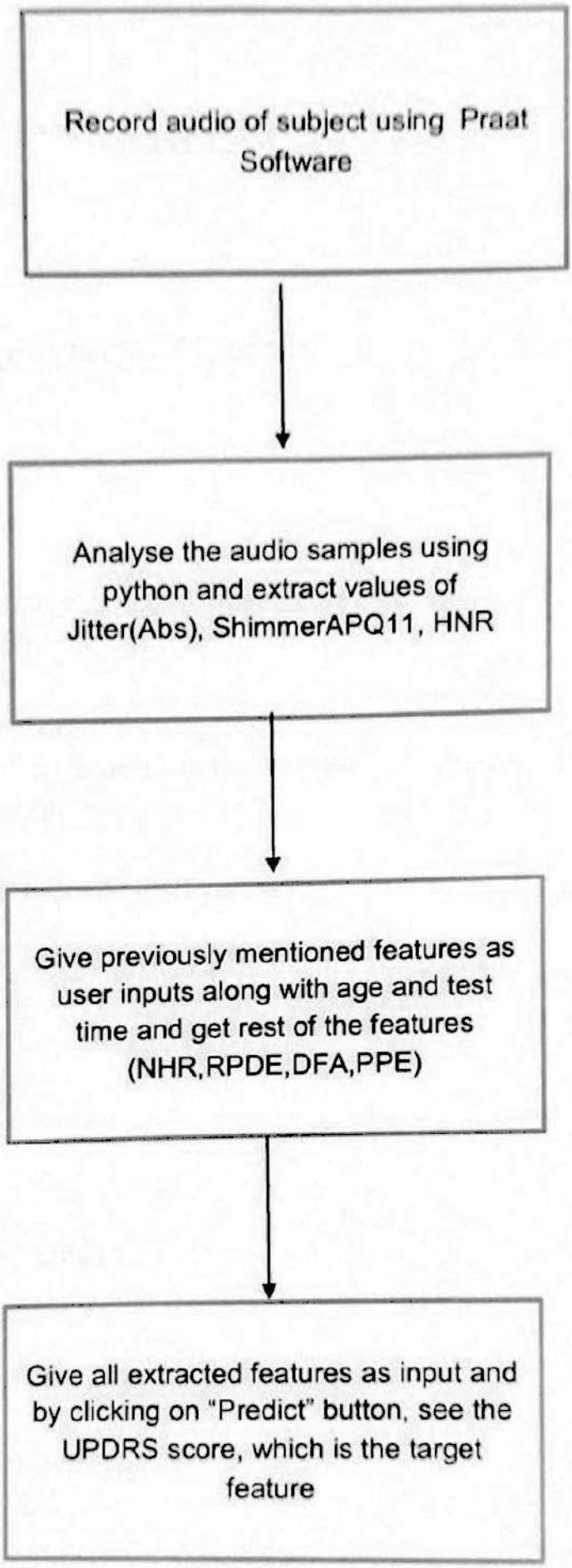
Step 46: Give inputs in the form and click the button to see the results.





THIS IS OUR FINAL RESULT!

7.2 WORKFLOW


Record audio of subject using Praat Software
Analyse the audio samples using python and extract values of Jitter(Abs), ShimmerAPQ11, HNR
Give previously mentioned features as user inputs along with age and test time and get rest of the features (NHR,RPDE,DFA,PPE)
Give all extracted features as input and by clicking on "Predict" button, see the UPDRS score, which is the target feature

CHAPTER 8

8.1. CODE

app.py

```
import flask
from flask import Flask,render_template,url_for,request
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import sklearn as s
from sklearn import datasets,linear_model
from sklearn.linear_model import lasso_path
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.externals import joblib
import json
from sklearn.ensemble import RandomForestRegressor

# load the built-in model
#gbr = joblib.load('model.pkl')

app=Flask(__name__)

@app.route('/')
def home():
        return render_template('2.html')


@app.route('/predict',methods=['POST'])
def predict():
        df=pd.read_csv('Data.csv',sep=',',header=0)
        df = df[df.test_time >= 0]
        #Features and Labels
        X = df.drop(["motor_UPDRS","total_UPDRS", "Jitter(Percent)", "Jitter:RAP", "Jitter:PPQ5",
"Jitter:DDP", "Shimmer", "Shimmer(dB)", "Shimmer:APQ3", "Shimmer:APQ5", "Shimmer:DDA"], axis=1)
        Y = df.total_UPDRS
        ########
```

```
        # Converts target and features to numpy arrays for sklearn API
        target_y = Y.as_matrix()
        features_x = X.as_matrix()
        # Split into train and test sets
        X_train, X_test, Y_train, Y_test = train_test_split(features_x, target_y, test_size=0.18,
random_state=42)
        #LinearRegression
        lm = RandomForestRegressor(n_estimators=300, random_state=20) #RandomForestRegression
        model = lm.fit(X_train, Y_train) #actual "learning" happens here
        lm.score(X_test,Y_test)
        #######




        if request.method == 'POST':
                result=request.form
                age = result['age']
                test_time = result['test_time']
                JitterAbsolute = result['JitterAbsolute']
                ShimmerAPQ11 = result['ShimmerAPQ11']
                NHR = result['NHR']
                HNR = result['HNR']
                RPDE = result['RPDE']
                DFA = result['DFA']
                PPE = result['PPE']



                # we create a json object that will hold data from user inputs
                #user_input = {'age':age, 'test_time':test_time, 'JitterAbsolute':JitterAbsolute,
'ShimmerAPQ11':ShimmerAPQ11, 'NHR':NHR,'HNR':HNR,'RPDE':RPDE,'DFA':DFA,'PPE':PPE}
                user_input=pd.DataFrame([[age, test_time, JitterAbsolute, ShimmerAPQ11,
NHR,HNR,RPDE,DFA,PPE]],columns=['age', 'test_time', 'JitterAbsolute', 'ShimmerAPQ11',
'NHR','HNR','RPDE','DFA','PPE'],dtype=float)
```

```
            my_prediction=lm.predict(user_input)[0]

            return flask.render_template('result1.1.html',original_input={'age':age, 'test_time':test_time,
'JitterAbsolute':JitterAbsolute, 'ShimmerAPQ11':ShimmerAPQ11,
'NHR':NHR,'HNR':HNR,'RPDE':RPDE,'DFA':DFA,'PPE':PPE},result=my_prediction)



        return render_template('result1.1.html',prediction=my_prediction)




if __name__ == '__main__':
  app.run(debug=True)
```

2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
<link rel="stylesheet" href="styles.css">
<style>
table, th, td {
 border: 3px solid black;
}
</style>
</head>


<body style="background-color:#e8f5e9;white;border:double">

 <div class="container">
```

```
<h2><center>PARKINSON'S DISEASE PREDICTOR</center></h2>
<div id="myCarousel" class="carousel slide" data-ride="carousel">
 <!-- Indicators -->
 <ol class="carousel-indicators">
  <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
  <li data-target="#myCarousel" data-slide-to="1"></li>
  <li data-target="#myCarousel" data-slide-to="2"></li>
 </ol>

 <!-- Wrapper for slides -->
 <div class="carousel-inner">

  <div class="item active">
   <img src="./static/photos/parkinsons disease4.png" alt="Parkinson's Disease1" style="width:100%;">
   <div class="carousel-caption">
    <h3>Parkinson's Disease</h3>
    <p>Let's have some khowledge about parkinson's</p>
   </div>
  </div>

  <div class="item">
   <img src="./static/photos/parkinson'sdisease6.jpg" alt="Parkinson's Disease2" style="width:100%;">
   <div class="carousel-caption">
    <h3>Chicago</h3>
    <p>What is Parkinson's??</p>
   </div>
  </div>

  <div class="item">
   <img src="./static/photos/parkinson's disease 5.jpg" alt="Parkinson's Disease3" style="width:100%;">
   <div class="carousel-caption">
    <h3>New York</h3>
    <p>What are the main symptoms of Parkinson's Disease??</p>
   </div>
  </div>

</div>

<!-- Left and right controls -->
<a class="left carousel-control" href="#myCarousel" data-slide="prev">
```

```
    <span class="glyphicon glyphicon-chevron-left"></span>
    <span class="sr-only">Previous</span>
   </a>
   <a class="right carousel-control" href="#myCarousel" data-slide="next">
    <span class="glyphicon glyphicon-chevron-right"></span>
    <span class="sr-only">Next</span>
   </a>
  </div>
 </div>


 <div class="container">
  <h1><center><b>Predict the chance of having Parkinson's </b></center></h1>

  <div class="row">
   <div class="form-group col-sm-6" style="background-color:white;border:double">
    <form class="form-inline" action="{{url_for('predict')}}" method="POST">
     <h4><center><b>Enter Values</b></center></h4>
     <label for="age">Age:</label>
     <input type="text" rows="4" column="5" name="age" placeholder="72"> <a href="#" data-
toggle="tooltip" data-placement="right" title="enter your current age.">info.</a>
     <br>
     <br>
     <label for="test_time">Test-time:</label>
     <input type="text" rows="4" column="5" name="test_time" placeholder="5.6431"><a href="#" data-
toggle="tooltip" data-placement="right" title="enter your current age.">info.</a>
     <br>
     <br>
     <label for="JitterAbsolute">Jitter(Absolute):</label>
     <input type="text" rows="4" column="5" name="JitterAbsolute" placeholder="0.000034"><a href="#"
data-toggle="tooltip" data-placement="right" title="enter the value between [-0.56343634308 to
6.87312731384] . The mean value of Jitter=5.87312731384">info.</a>
     <br>
     <br>
     <label for="ShimmerAPQ11">Shimmer:APQ11:</label>
     <input type="text" rows="4" column="5" name="ShimmerAPQ11" placeholder="0.01662"><a href="#"
data-toggle="tooltip" data-placement="right" title="enter the value between [0.003 to 0.276 ]. The mean value
of Shimmer:APQ11=0.028">info.</a>
     <br>
     <br>
```

```
    <label for="NHR">NHR:</label>
    <input type="text" rows="4" column="5" name="NHR" placeholder="0.014290"><a href="#" data-
toggle="tooltip" data-placement="right" title="enter the value between [0.0003 to 0.749]. The mean value of
NHR=0.032">info.</a>
    <br>
    <br>
    <label for="HNR">HNR:</label>
    <input type="text" rows="4" column="5" name="HNR" placeholder="21.640"><a href="#" data-
toggle="tooltip" data-placement="right" title="enter the value between [1.659 to 37.875]. The mean value o
HNR= 21.679">info.</a>
    <br>
    <br>
    <label for="RPDE">RPDE:</label>
    <input type="text" rows="4" column="5" name="RPDE" placeholder="0.41888"><a href="#" data-
toggle="tooltip" data-placement="right" title="enter  the value between [0.151 to 0.966].The mean value of
RPDE= 0.541">info.</a>
    <br>
    <br>
    <label for="DFA"> DFA :</label>
    <input type="text" rows="4" column="5" name="DFA" placeholder="0.54842"><a href="#" data-
toggle="tooltip" data-placement="right" title="enter the value between[0.514 to 0.866].The mean value of
DFA= 0.653">info.</a>
    <br>
    <br>
    <label for="PPE">PPE:</label>
    <input type="text" rows="4" column="5" name="PPE" placeholder="0.16006"><a href="#" data-
toggle="tooltip" data-placement="right" title="enter the value between [0.514  0.866].The mean value of
PPE=0.653">info.</a>
    <br>
    <br>
    <center><button type="submit" class="btn btn-primary">calculate</button></center>

   <br>

  </div>
  <div class="col-sm-6" style="background-color:white;border:double">
   <center><h2><b><u>NOTES</u></b></h2></center>
   <p>The main traditional measurement methods include F0 (the fundamental frequency or pitch of vocal
oscillation), absolute sound pressure level (indicating the relative loudness of speech), jitter (the extent of
variation in speech F0 from vocal cycle to vocal cycle), shimmer (the extent of variation in speech amplitude
```

from cycle to cycle), and noise-to-harmonics ratios (the amplitude of noise relative to tonal components in the speech). Studies have shown variations in all these measurements for PWP by comparison to healthy controls, indicating that these could be useful measures in assessing the extent of dysphonia.</p>

<table>
<tr>
<th>Symptom</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Average</th>
</tr>
<tr>
<td>Age</td>
<td>0</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>Test-time</td>
<td>-</td>
<td>-</td>
<td>-</td>

</tr>
<tr>
<td>Jitter(Abs)</td>
<td>-0.563436343</td>
<td>6.873127313</td>
<td>5.87312731384</td>
</tr>
<tr>
<td>Shimmer:APQ11</td>
<td>.003 </td>
<td>0.276 </td>
<td>0.028</td>
</tr>
<tr>
<td>NHR</td>
<td>0.0003</td>

```
    <td>0.749</td>
    <td>0.032</td>
   </tr>
   <tr>
    <td>HNR</td>
    <td>1.659</td>
    <td>37.875</td>
    <td>21.679</td>
   </tr>
   <tr>
    <td>RPDE</td>
    <td>0.151</td>
    <td>0.966</td>
    <td>0.541</td>
   </tr>
   <tr>
    <td>DFA</td>
    <td>0.514</td>
    <td>0.866</td>
    <td>0.653</td>
   </tr>
   <tr>
    <td>PPE</td>
    <td>0.022</td>
    <td>0.732</td>
    <td>0.220</td>
   </tr>
  </table>

 </center>
 <br>
 <br>
 <br>
 <br>

    </div>
   </div>
  </div>
```

```
<script>
$(document).ready(function(){
 $('[data-toggle="tooltip"]').tooltip();
});
</script>




</body>
</html>
```

result1.1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
 <link rel="stylesheet" href="styles.css">
<style>
table, th, td {
 border: 3px solid black;
}
</style>
</head>


<body style="background-color:#e8f5e9;white;border:double">

 <div class="container">
 <h2><center><B>PARKINSON'S DISEASE PREDICTOR</B></center></h2>
 <div id="myCarousel" class="carousel slide" data-ride="carousel" >
  <!-- Indicators -->
  <ol class="carousel-indicators">
```

```
  <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
  <li data-target="#myCarousel" data-slide-to="1"></li>
  <li data-target="#myCarousel" data-slide-to="2"></li>
 </ol>

 <!-- Wrapper for slides -->
 <div class="carousel-inner" style="border:double">

  <div class="item active">
   <img src="./static/photos/cure.jpg" alt="parkinson's disease" style="width:100%;">
   <div class="carousel-caption">
    <h3>Parkinson's </h3>
    <p>let's have some kowledge about parkinson's disease</p>
   </div>
  </div>

  <div class="item">
   <img src="./static/photos/cure4.jpg" alt="what is PD" style="width:100%;">
   <div class="carousel-caption">
    <h3>What is Parkinson's??</h3>
    <p>Parkinson's disease is a progressive nervous system disorder that affects movement. Symptoms start gradually, sometimes starting with a barely noticeable tremor in just one hand. Tremors are common, but the disorder also commonly causes stiffness or slowing of movement.</p>
   </div>
  </div>

  <div class="item">
   <img src="./static/photos/cure5.jpg" alt="Basic symptom" style="width:100%;">
   <div class="carousel-caption">
    <h3>Symptoms of Parkinson's</h3>
    <p>Parkinson's disease signs and symptoms can be different for everyone. Early signs may be mild and go unnoticed. Symptoms often begin on one side of your body and usually remain worse on that side, even after symptoms begin to affect both sides.</p>
   </div>
  </div>

 </div>

 <!-- Left and right controls -->
 <a class="left carousel-control" href="#myCarousel" data-slide="prev">
```

```
    <span class="glyphicon glyphicon-chevron-left"></span>
    <span class="sr-only">Previous</span>
   </a>
   <a class="right carousel-control" href="#myCarousel" data-slide="next">
    <span class="glyphicon glyphicon-chevron-right"></span>
    <span class="sr-only">Next</span>
   </a>
  </div>
 </div>

 <div class="container">
  <h1><center><b>Predict the chance of having Parkinson's </b></center></h1>

  <div class="row">
   <div class="form-group col-sm-6" style="background-color:white;border:double">
    <form class="form-inline">

     <header>
    <div class="container">
     <div id="name">
      App is running successfully!
     </div>
     <h2>Parkinson's Disease Prediction</h2>

    </div>
  </header>
  <p style="color: blue;font-size: 20;text-align: center;">Given User Inputs:</p>
  <div class="results">
   {% if result %}
      {% for variable, value in original_input.items() %}
        <b>{{ variable }}</b> : {{ value }}
      {% endfor %}
      <br>
      <br> Predicted total UPDRS score is:
        <p style="font-size:50px">{{ result }}</p>
   {% endif %}
  </div>
  <br>
 </form>
 </div>
```

```
<div class="col-md-6" style="background-color:white;border:double">
<center><h2><b><u>NOTES</u></b></h2></center>
<p>The main traditional measurement methods include F0 (the fundamental frequency or pitch of vocal oscillation), absolute sound pressure level (indicating the relative loudness of speech), jitter (the extent of variation in speech F0 from vocal cycle to vocal cycle), shimmer (the extent of variation in speech amplitude from cycle to cycle), and noise-to-harmonics ratios (the amplitude of noise relative to tonal components in the speech). Studies have shown variations in all these measurements for PWP by comparison to healthy controls, indicating that these could be useful measures in assessing the extent of dysphonia.</p>

<a href="https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2897716/">For Update</a>
<br><br>
<br>

</body>
```

applink1.py

```
import flask
from flask import Flask,render_template,url_for,request
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import sklearn as s
from sklearn import datasets,linear_model
from sklearn.linear_model import lasso_path
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.externals import joblib
import json
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures #PolynomialRegression
from sklearn.linear_model import LinearRegression

app=Flask(__name__)
```

```
@app.route('/')
def home():
    return render_template('home1(edited).html')

@app.route('/predict1',methods=['POST'])
def predict1():
    df=pd.read_csv('Data.csv',sep=',',header=0)
    df = df[df.test_time >= 0] #df is the common corpus


    #Features and Labels
    #1. For NHR
    X1 = df.drop(["motor_UPDRS","total_UPDRS", "Jitter(Percent)", "Jitter:RAP", "Jitter:PPQ5",
"Jitter:DDP", "Shimmer", "Shimmer(dB)", "Shimmer:APQ3", "Shimmer:APQ5",
"Shimmer:DDA","NHR","RPDE","DFA","PPE"], axis=1)
    Y1 = df.NHR

    target_y_1 = Y1.as_matrix()
    features_x_1 = X1.as_matrix()

    X_train_1, X_test_1, Y_train_1, Y_test_1 = train_test_split(features_x_1, target_y_1, test_size=0.18,
random_state=42)

    # PolynomialFeatures (prepreprocessing)
    poly = PolynomialFeatures(degree=2)
    X_ = poly.fit_transform(X_train_1)
    X_test_ = poly.fit_transform(X_test_1)
    # Instantiate
    lg = LinearRegression()

    # Fit
    lg.fit(X_, Y_train_1)

    if request.method == 'POST':
        result=request.form

        age = result['age']
        test_time = result['test_time']
        JitterAbsolute = result['JitterAbsolute']
        ShimmerAPQ11 = result['ShimmerAPQ11']
```

```
        HNR = result['HNR']


        user_input=pd.DataFrame([[age, test_time, JitterAbsolute,
ShimmerAPQ11,HNR]],columns=['age', 'test_time', 'JitterAbsolute', 'ShimmerAPQ11','HNR'],dtype=float)
        #1. NHR
        #my_prediction_nhr=lg.predict(user_input)[0]
        #return flask.render_template('result1(NHR).html',original_input={'age':age2,
'test_time':test_time2, 'JitterAbsolute':JitterAbsolute2,
'ShimmerAPQ11':ShimmerAPQ112,'HNR':HNR2},result=my_prediction_rpde)
        ######################
        #return render_template('result1(NHR).html',prediction=my_prediction_nhr)


    ##########################################################################################
#####################################

@app.route('/predict2',methods=['POST'])
def predict2():
    df=pd.read_csv('Data.csv',sep=',',header=0)
    df = df[df.test_time >= 0] #df is the common corpus


    #Features and Labels
    #2. For RPDE
    X2 = df.drop(["motor_UPDRS","total_UPDRS", "Jitter(Percent)", "Jitter:RAP", "Jitter:PPQ5",
"Jitter:DDP", "Shimmer", "Shimmer(dB)", "Shimmer:APQ3", "Shimmer:APQ5",
"Shimmer:DDA","NHR","RPDE","DFA","PPE"], axis=1)
    Y2 = df.RPDE

    target_y_2 = Y2.as_matrix()
    features_x_2 = X2.as_matrix()

    X_train_2, X_test_2, Y_train_2, Y_test_2 = train_test_split(features_x_2, target_y_2, test_size=0.18,
random_state=42)

    #RandomForest
    regressor1 = RandomForestRegressor(n_estimators=100, random_state=1)
    regressor1.fit(X_train_2, Y_train_2)
```

```
    if request.method == 'POST':
        result=request.form

        age = result['age']
        test_time = result['test_time']
        JitterAbsolute = result['JitterAbsolute']
        ShimmerAPQ11 = result['ShimmerAPQ11']
        HNR = result['HNR']


        user_input=pd.DataFrame([[age, test_time, JitterAbsolute,
ShimmerAPQ11,HNR]],columns=['age', 'test_time', 'JitterAbsolute', 'ShimmerAPQ11','HNR'],dtype=float)
        #2.RPDE
        my_prediction_rpde=regressor1.predict(user_input)[0]
        ######################
        return flask.render_template('result2(RPDE).html',original_input={'age':age,
'test_time':test_time, 'JitterAbsolute':JitterAbsolute,
'ShimmerAPQ11':ShimmerAPQ11,'HNR':HNR},result=my_prediction_rpde)
        ######################
    return render_template('result2(RPDE).html',prediction=my_prediction_rpde)


    ##########################################################################################
####################################
@app.route('/predict3',methods=['POST'])
def predict3():
    df=pd.read_csv('Data.csv',sep=',',header=0)
    df = df[df.test_time >= 0] #df is the common corpus


    #Features and Labels
    #3. For DFA
    X3 = df.drop(["motor_UPDRS","total_UPDRS", "Jitter(Percent)", "Jitter:RAP", "Jitter:PPQ5",
"Jitter:DDP", "Shimmer", "Shimmer(dB)", "Shimmer:APQ3", "Shimmer:APQ5",
"Shimmer:DDA","NHR","RPDE","DFA","PPE"], axis=1)
    Y3 = df.DFA

    target_y_3 = Y3.as_matrix()
    features_x_3 = X3.as_matrix()
```

```
        X_train_3, X_test_3, Y_train_3, Y_test_3 = train_test_split(features_x_3, target_y_3, test_size=0.18,
random_state=42)
        #RandomForest
        regressor2 = RandomForestRegressor(n_estimators=100, random_state=1)
        regressor2.fit(X_train_3, Y_train_3)


        if request.method == 'POST':
                result=request.form

                age = result['age']
                test_time = result['test_time']
                JitterAbsolute = result['JitterAbsolute']
                ShimmerAPQ11 = result['ShimmerAPQ11']
                HNR = result['HNR']


                user_input=pd.DataFrame([[age, test_time, JitterAbsolute,
ShimmerAPQ11,HNR]],columns=['age', 'test_time', 'JitterAbsolute', 'ShimmerAPQ11','HNR'],dtype=float)
                #3.DFA
                my_prediction_dfa=regressor2.predict(user_input)[0]
                ######################
                return flask.render_template('result3(DFA).html',original_input={'age':age,
'test_time':test_time, 'JitterAbsolute':JitterAbsolute,
'ShimmerAPQ11':ShimmerAPQ11,'HNR':HNR},result=my_prediction_dfa)
                ######################
        return render_template('result3(DFA).html',prediction=my_prediction_dfa)


        ##########################################################################################
#################

@app.route('/predict4',methods=['POST'])
def predict4():
        df=pd.read_csv('Data.csv',sep=',',header=0)
        df = df[df.test_time >= 0] #df is the common corpus


        #4. For PPE
```

```
    X4 = df.drop(["motor_UPDRS","total_UPDRS", "Jitter(Percent)", "Jitter:RAP", "Jitter:PPQ5",
"Jitter:DDP", "Shimmer", "Shimmer(dB)", "Shimmer:APQ3", "Shimmer:APQ5",
"Shimmer:DDA","NHR","RPDE","DFA","PPE"], axis=1)
    Y4 = df.PPE

    target_y_4 = Y4.as_matrix()
    features_x_4 = X4.as_matrix()

    X_train_4, X_test_4, Y_train_4, Y_test_4 = train_test_split(features_x_4, target_y_4, test_size=0.18,
random_state=42)

    #RandomForest
    regressor3 = RandomForestRegressor(n_estimators=100, random_state=1)
    regressor3.fit(X_train_4, Y_train_4)

    if request.method == 'POST':
        result=request.form

        age = result['age']
        test_time = result['test_time']
        JitterAbsolute = result['JitterAbsolute']
        ShimmerAPQ11 = result['ShimmerAPQ11']
        HNR = result['HNR']


        user_input=pd.DataFrame([[age, test_time, JitterAbsolute,
ShimmerAPQ11,HNR]],columns=['age', 'test_time', 'JitterAbsolute', 'ShimmerAPQ11','HNR'],dtype=float)
        #3.PPE
        my_prediction_ppe=regressor3.predict(user_input)[0]
        ######################
        return flask.render_template('result4(PPE).html',original_input={'age':age,
'test_time':test_time, 'JitterAbsolute':JitterAbsolute,
'ShimmerAPQ11':ShimmerAPQ11,'HNR':HNR},result=my_prediction_ppe)
        ######################
    return render_template('result4(PPE).html',prediction=my_prediction_ppe)
    ##########################################################################################
##

    ##########################################################################################
####
```

```
if __name__ == '__main__':
  app.run(debug=True)
```

home1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
 <link rel="stylesheet" href="styles.css">
<style>
.div1 {
 width: 1170px;
 height: 40px;
 border: 5px #42a5f5;
 background-color:#42a5f5;
}
</style>
</head>
<body style="background-color:#757575;border:double">
        <!--HEADING-->
        <div>
                <div class="container">
                        <div class="row">
                                <div class="div1">
                                <h2><center>PREDICT FEATURE OF PARKINSON'S
DISEASE</center></h2>
                                </div>
                        </div>
                </div>
        </div>


        <div class="container">
                <BR>
        <div class="row">
```

```
<div class="form-group col-sm-6" style="background-color:white;border:double">
                <form class="form-inline" action="{{url_for('predict1')}}" method="POST">
                <h4><center><b>Enter Values</b></center></h4>
                <label for="age">Age:</label>
                <input type="text" rows="4" column="5" name="age">
                <br>
                <br>
                <label for="test_time">Test_time:</label>
                <input type="text" rows="4" column="5" name="test_time">
                <br>
                <br>
                <label for="JitterAbsolute">Jitter(Absolute):</label>
                <input type="text" rows="4" column="5" name="JitterAbsolute">
                <br>
                <br>
                <label for="ShimmerAPQ11">ShimmerAPQ11:</label>
                <input type="text" rows="4" column="5" name="ShimmerAPQ11">
                <br>
                <br>
                <label for="HNR">HNR:</label>
                <input type="text" rows="4" column="5" name="HNR">
                <br>
                <br>
                <center><button type="submit" class="btn btn-info"value="predict1">Predict
NHR</button></center>
                <br>
                </form>
                <br>
        </div>
        <!--FORM FOR RPDE-->
        <div class="col-sm-6" style="background-color:white;border:double">
                <form class="form-inline" action="{{url_for('predict2')}}" method="POST">
                <h4><center><b>Enter Values</b></center></h4>
                <label for="age">Age:</label>
                <input type="text" rows="4" column="5" name="age">
                <br>
                <br>
                <label for="test_time">Test_time:</label>
                <input type="text" rows="4" column="5" name="test_time">
                <br>
```

```
                    <br>
                    <label for="JitterAbsolute">Jitter(Absolute):</label>
                    <input type="text" rows="4" column="5" name="JitterAbsolute">
                    <br>
                    <br>
                    <label for="ShimmerAPQ11">ShimmerAPQ11:</label>
                    <input type="text" rows="4" column="5" name="ShimmerAPQ11">
                    <br>
                    <br>
                    <label for="HNR">HNR:</label>
                    <input type="text" rows="4" column="5" name="HNR">
                    <br>
                    <br>
                    <center><button type="submit" class="btn btn-info"value="predict1">Predict
RPDE</button></center>
                    <br>
                    </form>
                    <br>
                </div>
            </div>
</div>

        <div class="container">

        <div class="row">
                <!--FORM FOR DFA-->
                <div class="form-group col-sm-6" style="background-color:white;border:double">
                    <form class="form-inline" action="{{url_for('predict3')}}" method="POST">
                    <h4><center><b>Enter Values</b></center></h4>
                    <label for="age">Age:</label>
                    <input type="text" rows="4" column="5" name="age">
                    <br>
                    <br>
                    <label for="test_time">Test_time:</label>
                    <input type="text" rows="4" column="5" name="test_time">
                    <br>
                    <br>
                    <label for="JitterAbsolute">Jitter(Absolute):</label>
                    <input type="text" rows="4" column="5" name="JitterAbsolute">
                    <br>
```

```
                    <br>
                    <label for="ShimmerAPQ11">ShimmerAPQ11:</label>
                    <input type="text" rows="4" column="5" name="ShimmerAPQ11">
                    <br>
                    <br>
                    <label for="HNR">HNR:</label>
                    <input type="text" rows="4" column="5" name="HNR">
                    <br>
                    <br>
                    <center><button type="submit" class="btn btn-info"value="predict1">Predict
PFA</button></center>
                    <br>
                    </form>
                    <br>
        </div>
            <!--FORM FOR PPE-->
                    <div class="col-sm-6" style="background-color:white;border:double">
                    <form class="form-inline" action="{{url_for('predict4')}}" method="POST">
                    <h4><center><b>Enter Values</b></center></h4>
                    <label for="age">Age:</label>
                    <input type="text" rows="4" column="5" name="age">
                    <br>
                    <br>
                    <label for="test_time">Test_time:</label>
                    <input type="text" rows="4" column="5" name="test_time">
                    <br>
                    <br>
                    <label for="JitterAbsolute">Jitter(Absolute):</label>
                    <input type="text" rows="4" column="5" name="JitterAbsolute">
                    <br>
                    <br>
                    <label for="ShimmerAPQ11">ShimmerAPQ11:</label>
                    <input type="text" rows="4" column="5" name="ShimmerAPQ11">
                    <br>
                    <br>
                    <label for="HNR">HNR:</label>
                    <input type="text" rows="4" column="5" name="HNR">
                    <br>
                    <br>
```

```
                    <center><button type="submit" class="btn btn-info"value="predict1">Predict
PE</button></center>
                    <br>
                    </form>
                    <br>
                </div>
            </div>
        </div>

    </body>
    </html>
```

resultnhr.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
 <link rel="stylesheet" href="styles.css">
<style>
.div1 {
 width: 1168px;
 height: 40px;
 border: 5px #42a5f5;
 background-color:#42a5f5;
}
</style>
</head>
<body style="background-color:#757575;border:double">
    <!--HEADING-->
    <div>
        <div class="container">
            <div class="row">
                <div class="div1">
                <h2><center>NHR</center></h2>
                </div>
```

```
                </div>
            </div>
        </div>

        <div class="container">
            <div class="row">
                <div class="form-group col-sm-12" style="background-color:white;border:double">
                    <div class="results">
                        {% if result %}
                        {% for variable, value in original_input.items() %}
                        <b>{{ variable }}</b> : {{ value }}
                        {% endfor %}
                        <br>
                        <br> Predicted NHR score2 is:
                        <p style="font-size:50px">{{ result }}</p>
                        {% endif %}
                    </div>
                </div>
            </div>
        </div>
</body>
</html>
```

resultrpde.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
 <link rel="stylesheet" href="styles.css">
<style>
.div1 {
 width: 1168px;
 height: 40px;
 border: 5px #42a5f5;
 background-color:#42a5f5;
```

```
}
</style>
</head>
<body style="background-color:#757575;border:double">
        <!--HEADING-->
        <div>
                <div class="container">
                        <div class="row">
                                <div class="div1">
                                <h2><center>RPDE</center></h2>
                                </div>
                        </div>
                </div>
        </div>

        <div class="container">
                <div class="row">
                        <div class="form-group col-sm-12" style="background-color:white;border:double">
                                <div class="results">
                                {% if result %}
                                {% for variable, value in original_input.items() %}
                                <b>{{ variable }}</b> : {{ value }}
                                {% endfor %}
                                <br>
                                <br> Predicted RPDE is:
                                <p style="font-size:50px">{{ result }}</p>
                                {% endif %}
                                </div>
                        </div>
                </div>
        </div>


</body>
</html>

resultdfa.html
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<title>Bootstrap Example</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
<link rel="stylesheet" href="styles.css">
<style>
.div1 {
width: 1168px;
height: 40px;
border: 5px #42a5f5;
background-color:#42a5f5;
}
</style>
</head>
<body style="background-color:#757575;border:double">
        <!--HEADING-->
        <div>
                <div class="container">
                        <div class="row">
                                <div class="div1">
                                <h2><center>DFA</center></h2>
                                </div>
                        </div>
                </div>
        </div>

        <div class="container">
                <div class="row">
                        <div class="form-group col-sm-12" style="background-color:white;border:double">
                                <div class="results">
                                        {% if result %}
                                        {% for variable, value in original_input.items() %}
                                        <b>{{ variable }}</b> : {{ value }}
                                        {% endfor %}
                                        <br>
                                        <br> Predicted DFA score2 is:
                                        <p style="font-size:50px">{{ result }}</p>
                                        {% endif %}
```

```
                    </div>
                </div>
            </div>
        </div>
</body>
</html>
```

resultppe.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
<link rel="stylesheet" href="styles.css">
<style>
.div1 {
 width: 1168px;
 height: 40px;
 border: 5px #42a5f5;
 background-color:#42a5f5;
}
</style>
</head>
<body style="background-color:#757575;border:double">
        <!--HEADING-->
        <div>
                <div class="container">
                        <div class="row">
                                <div class="div1">
                                <h2><center>NHR</center></h2>
                                </div>
                        </div>
                </div>
        </div>
```

```
    <div class="container">
        <div class="row">
            <div class="form-group col-sm-12" style="background-color:white;border:double">
                <div class="results">
                    {% if result %}
                    {% for variable, value in original_input.items() %}
                    <b>{{ variable }}</b> : {{ value }}
                    {% endfor %}
                    <br>
                    <br> Predicted PPE is:
                    <p style="font-size:50px">{{ result }}</p>
                    {% endif %}
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

CHAPTER 9

9.1 FUTURE SCOPE OF THIS PROJECT

There are some more future scope to make this app more useful.

1.We can link up all the web pages of both the Flask App to each other as it can work progressively from taking the voice sample to find UPDRS score

2.By further EDA, r2,rmse,eva and other evaluation score can be improved.

3.By tweaking attributes of Random Forest Regression the scores can be improved.

4."Gender" feature is not present in our data set. In our next model we will try to include this feature too as sex of the subject seems to be an important feature.

5.In our data set we have about 6000 data which is not adequate to train a model properly. Further we will try to assemble more data to train the model to be more proper and accurate .

6.In future we can add a database with this model to make the app for public use . As well as we can host this app too.

Acknowledgement

We would like to express our special thanks to our mentor **Mrs. SURANITA SARKAR** as well as our **HOD Mrs. DEBDUTTA PAL** who gave us the opportunity to do the project on the topic of "**Parkinson's Disease Prediction by Analysing Voice Recording Sample**", which also helped us to do a research on this topic .We are really thankful to them.
Secondly we would also like to thank our friends who helped us for finishing this project successfully.

BIBLIOGRAPHY

- https://praat.en.softonic.com/
- https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring
- https://www.analyticsvidhya.com/blog/2017/09/machine-learning-models-as-apis-using-flask/
- http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3051371/#R16
- https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring