Unveiling Parkinson's Disease through Voice Data: A Machine Learning Perspective

A PROJECT REPORT

Submitted by

K. E. BHARATH KUMAR

19MIS1057

In the partial fulfilment of for the award of degree of

Master of Technology

In

SOFTWARE ENGINEERING (5 YEAR INTEGRATED PROGRAMME)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Vellore Institute of Technology
Vandalur - Kelambakkam Road, Chennai - 600 127
April - 2024

VIT®
CHENNAI
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I hereby declare that the project entitled **Unveiling Parkinson's Disease through Voice Data: A Machine Learning Perspective** submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, 600 127, in partial fulfilment of the requirements of the award of the degree of Master of Technology in Software Engineering (5 year Integrated Programme) and as part of SWE1904 – Capstone Project is a bona-fide record of the work carried out by me under the supervision of **Dr. M. PREMALATHA** I further declare that the work reported in this project, has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or University.

Place:Chennai Signature of candidate.

Date:

VIT®

CHENNAI

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the report entitled **Unveiling Parkinson's Disease through Voice Data: A Machine Learning Perspective** is prepared and submitted by **K. E. Bharath Kumar (Reg. No. 19MIS1057**) to Vellore Institute of Technology, Chennai, in partial fulfilment of the requirement for the award of the degree of Master of Technology in Software Engineering (5 year Integrated Programme) and as part of SWE1904 – Capstone project is a bona-fide record of the work carried out by me under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission.

Guide	HOD
Name: Dr. M. Premalatha	Name: Dr. Nisha V M
Date:	Date:
Examiner 1	Examiner 2
Name:	Name:
Date:	Date:

Acknowledgement

I obliged to give my appreciation to a number of people without whom I could not have completed this thesis successfully.

I would like to place on record my deep sense of gratitude and thanks to my internal guide Dr. M PREMALATHA**,** School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai, whose esteemed support and immense guidance encouraged me to complete the project successfully.

I would like to thank our HoD Dr. Nisha V M, School of Computer Science and Engineering (SCOPE) and Project Coordinator Dr. Kanchana Devi V, School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai, for their valuable support and encouragement to take up and complete this thesis.

Special mention to our Dean Dr. Ganesan R, Associate Deans Dr. Parvathi R and Dr. S. Geetha, School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai, for motivating us in every aspect of software engineering.

I thank our management of Vellore Institute of Technology, Chennai, for permitting me to use the library and laboratory resources. I also thank all the faculty members for giving me the courage and the strength that I needed to complete my goal. This acknowledgment would be incomplete without expressing the whole hearted thanks to my family and friends who motivated me during the course of my work.

K. E. Bharath Kumar

19MIS1057

Abstract

Parkinson’s disease (PD) is the second commonest late life neurodegenerative disease after Alzheimer’s disease. It is prevalent throughout the world and predominantly affects patients above 60 years old. Diagnosis of Parkinson's disease (PD) is commonly based on medical observations and assessment of clinical signs, including the characterization of a variety of motor symptoms. This study aims to develop an accurate and non-invasive predictive model for Parkinson's disease using machine learning algorithms and voice data. Voice samples from individuals with and without Parkinson's disease are collected and analysed to extract relevant features.

Various machine learning techniques, including support vector machines, and decision trees, are employed to build predictive models. The results demonstrate the feasibility of diagnosing Parkinson's disease through voice data with high accuracy, potentially enabling early and cost-effective detection of the condition. This research has promising implications for improving the quality of life for individuals affected by Parkinson's disease.

PD patients shows the superior detection performance of the designed XGBoost model, which achieves the highest accuracy, 80-90% on average. Besides detecting the PD, we also do comparison between the SVM and XGBoost model on basis of accuracy.

CONTENTS

LIST OF FIGURES

1.INTRODUCTION:

General:

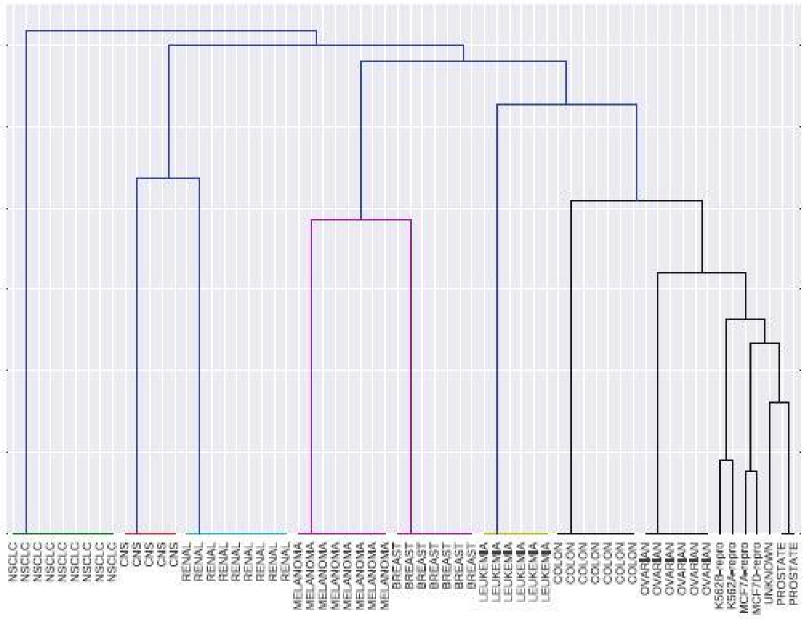
1.1.1 JUPYTER:

Jupyter previously known as IPython Notebook, is a web-based, interactive development environment. Originally developed for Python, it has since expanded to support over 40 other programming languages including Julia and R. Jupyter allows for notebooks to be written that contain text, live code, images, and equations. These notebooks can be shared and can even be hosted on GitHub for free. For each section of this tutorial, you can download a Jupyter notebook that allows you to edit and experiment with the code and examples for each topic. Jupyter is part of the Anaconda distribution; it can be started from the command line using the jupyter command:

```
$ jupyter notebook
```

1.1.2 Clustering:

Clustering algorithms focus on ordering data together into groups. In general clustering algorithms are unsupervised—they require no **y** response variable as input. Attempt to find groups or clusters within data where you do not know the label for each sample. SciKit-Learn have many clustering algorithms, We will plot a visualisation of the clustering using what is known as a dendrogram, also using the SciPy library. The goal is to cluster the data properly in logical groups, in this case into the types represented by each sample's expression data. We do this using agglomerative hierarchical clustering, using Ward's linkage method:



Fig(1)

1.1.3 SciKit-Learn:

SciKit-Learn provides a standardised interface to many of the most commonly used machine learning algorithms and is the most popular and frequently used library for machine learning for Python. As well as providing many learning algorithms, SciKit-Learn has a large number of convenience functions for common preprocessing tasks (for example, normalisation or *k*-fold cross validation).SciKit-Learn is a very large software library.

1.1 BACKGROUND:

Parkinson’s disease (PD) and essential tremor (ET) are movement disorders that can have similar clinical characteristics including tremor and gait difficulty. These disorders can be misdiagnosed leading to delay in appropriate treatment. The aim of the study was to determine whether balance and gait variables obtained with wearable inertial motion sensors can be utilized to differentiate between PD and ET using machine learning. Additionally, we compared classification performances of several machine learning models.

1.2 PROBLEM STATEMENT:

The problem of "Prediction of Parkinson's Disease Using Machine Learning" involves developing a machine learning model that uses relevant data to identify individuals at risk of developing Parkinson's Disease or diagnose the disease in its early stages. This requires data collection, preprocessing, feature selection, model development, evaluation, and, if successful, deployment in clinical settings to aid in early detection and diagnosis.The goal is to improve patient outcomes and medical interventions through accurate predictive modeling.

1.3 MOTIVATION:

The motivation for this project stems from the significant global prevalence of Parkinson's disease, particularly in the elderly population. Currently, diagnosis relies on clinical observations, which can be subjective and late in detecting the disease. This study seeks to develop an accurate, non-invasive predictive model using voice data and machine learning techniques. By achieving high accuracy in diagnosing Parkinson's disease through voice data, this research promises to enhance early detection and potentially improve the quality of life for those affected by the condition. The XGBoost model demonstrates superior detection performance, with an average accuracy of 80-90%, providing a valuable tool for early diagnosis. This project also compares the performance of SVM and XGBoost models, further advancing our understanding of diagnostic techniques.

1.4 Challenges:

Machine learning models classified PD and ET based on balance and gait characteristics better than the dummy model (F1-score = 0.48) or logistic regression (F1-score = 0.53). The highest F1-score was 0.61 of neural network, followed by 0.59 of gradient boosting, 0.56 of random forest, 0.55 of support vector machine, 0.53 of decision tree, and 0.49 of k-nearest neighbor.This study demonstrated the utility of machine learning models to classify different movement disorders based on balance and gait characteristics collected from wearable sensors.

2. PLANNING & REQUIREMENT SPECIFICATION:

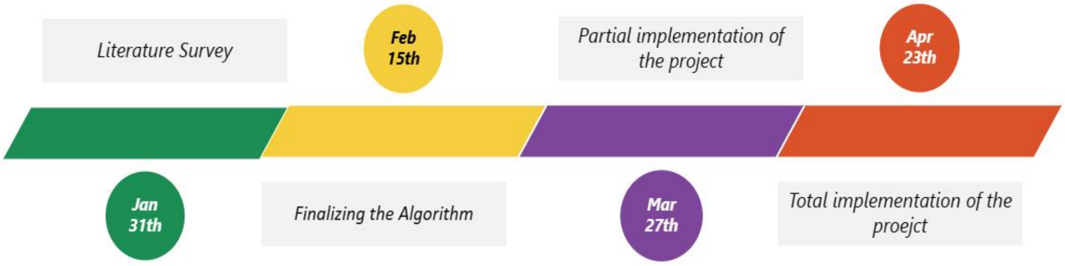


Figure:2 System Planning

Literature Survey -------> Jan 31th

Finalizing the Algorithms -------> Feb 15th

Partial Implementation of the Project --------> Mar 27th

Total Implementation of the Project --------> Apr 23th

2.1. LITERATURE REVIEW:

The literature review in a project on predicting liver cancer using machine learning techniques serves as the foundational background and introduction to existing research and studies related to the topic. It typically covers:

1. Previous Studies: Overview of prior research and studies on liver cancer prediction or similar medical diagnoses.

2. Machine Learning in Medical Diagnosis: Discussions on how machine learning techniques have been employed in medical fields, especially for disease prediction or diagnosis.

3. Liver Cancer Prediction Models: Summary and analysis of existing models or algorithms utilized in predicting liver cancer, including their strengths and limitations.

4. Data Sets Used: Insight into the types of datasets (demographic, genetic, biomarker data, etc.) commonly employed in liver cancer prediction research.

5. Accuracy and Validation: Assessment of the accuracy, performance metrics, and validation methods used in previous studies.

6. Emerging Trends and Gaps: Identification of gaps or shortcomings in existing literature and potential areas for improvement or further exploration.

The literature review sets the stage for the current project by providing a comprehensive understanding of the advancements, challenges, and opportunities in liver cancer prediction using machine learning techniques.

Marwa Almasoud, Tomas E Ward2 (2021) [1] Parkinson disease gait classification based on machine learning approach. This study assesses the effectiveness of two machine classifiers, Artificial Neural Network (ANN) and Support Vector Machine (SVM), in distinguishing Parkinson's disease gait patterns during self-selected speed walking. Gait parameters, including spatiotemporal, kinematic, and kinetic aspects, are utilized as features. The study employs two normalization techniques and evaluates the classifiers' performance, revealing promising results with spatiotemporal features.

Sanghee Moon, Hyun-Je Song, Vibhash D. Sharma, Kelly E. Lyons, Rajesh Pahwa (2022) [2] Classification of parkinson's disease and essential tremor based on balance and gait characteristics from wearable motion sensors via machine learning. This study investigates the potential of wearable inertial motion sensors to differentiate between Parkinson’s disease (PD) and essential tremor (ET) based on balance and gait variables. The retrospective analysis involves machine learning models and data from 524 PD and 43 ET participants.

Basetty Mallikarjuna, R. Viswanathan and Bharat Bhushan Naib [3] (2022) Feedback-based gait identification using deep neural network classification. This paper addresses healthcare challenges in gait identification, particularly focusing on abnormal patterns like Parkinson gait, Hemiplegic gait, and Neuropathic gait. Utilizing a deep neural network (DNN), it identifies lean and ramp angles, offering insights into gait abnormalities without clinical observation.

Rana Zia Ur Rehman, Silvia Del Din, Yu Guan, Alison J. Yarnall. (2022) [4] Selecting clinically relevant gait characteristics for classi - fication of early parkinson's disease. This study explores machine learning techniques to identify optimal gait characteristics for early Parkinson's disease (PD) classification. Utilizing a dataset of 303 participants, the study identifies five clinical gait features that achieve accurate classification, with model accuracy ranging from 73–97% for early PD.

Milla Juutinen,Justin Zhu, Cassia Wang (2022) [5] Perkinson's disease detection from 20-step walking tests using inertial sensors of smartphone. This study explores smartphone-based movement analysis for Parkinson's disease (PD) detection. Comparing three feature selection and nine classification methods, k Nearest Neighbors achieved 84.5% accuracy. The findings offer insights for optimizing methodologies in monitoring PD patients using smartphones in free-living conditions.

R. Prashanth, S. D. Roy, and V. Pillay (2023) [6] Early detection of Parkinson's disease using speech features and a deep learning model. Introducing a machine learning-based automatic prediction framework for early Parkinson’s disease diagnosis using speech data. Ensemble methods like Random Forests and Gradient Boosting achieve an average prediction accuracy of 86.5%, further improved to 91.5% with oversampling. This suggests potential applicability as a decision support system in real diagnostic scenarios.

H. Gunduz (2023) [7] A novel approach for Parkinson's disease detection using speech signals and transfer learning. Developing deep convolutional neural networks for automated Parkinson's disease (PD) identification based on voice signals. Using transfer learning with fine-tuning on SqueezeNet1_1, ResNet101, and DenseNet161 architectures, DenseNet161 is identified as the most suitable, achieving an accuracy of 89.75%. This model, when integrated into smart electronic devices, can serve as an alternative pre-diagnosis method during in-clinic assessments, potentially enhancing patients' quality of life and reducing costs for the national health system.

A. Mostafa, Z. Mohammed, S. Hamad Khaleefah Al-Dulaimi, and A. Mustapha (2022) [8] A hybrid deep learning model for Parkinson's disease detection using voice signals. Proposing a speech signal-based hybrid Parkinson's disease diagnosis system for early detection. Utilizing various feature selection methods and classifiers on a UCI dataset, the combination of genetic algorithm and random forest achieves superior performance with 95.58% accuracy, surpassing recent literature findings.

A. H. Chen, Y. Zhang, Y. Wang, T. Xu, and T. Chen (2022) [9] Parkinson's disease detection based on multi-scale convolutional neural network and speech features. Proposing MSHANet, a multiscale hybrid attention network, for automatic Parkinson’s disease (PD) detection using convolutional neural networks. Achieving 94.11% accuracy, 94.18% precision, and 0.9585 AUC, the method offers potential assistance to clinicians in accurate PD diagnosis using brain images.

S. Lahmiri, A. Shmuel, and A. Benyoussef (2022) [10] A novel deep learning approach for Parkinson's disease detection using speech signals. Proposing an end-to-end deep learning model for Parkinson’s disease detection from speech signals. Utilizing time-distributed 2D-CNNs and 1D-CNN, the model outperforms expert features-based machine learning on two databases, achieving up to 92% accuracy. Visualizing features, it captures clinical evidence for detecting Parkinson’s disease patients, emphasizing the importance of the low-frequency region in speech signals.

M. El Hannani, A. El Hannani, A. Amiar, A. Bennis, and O. El Moutaouakkil (2022) [11] Parkinson's disease detection using deep learning and speech signal analysis. Voice biomarkers offer insight into Parkinson's disease (PD). Utilizing a voice dataset, this study demonstrates the potential of deep neural networks in accurately diagnosing PD, achieving a peak accuracy of 85%. The machine learning models outperform non-expert clinical diagnosis and rival movement disorder specialists' accuracy.

A. Bougrine, A. Ouannas, M. Z. Ouahrani, and M. E. Zahi (2022) [12] Parkinson's disease detection using speech signals and a hybrid deep learning model. Proposing a speech signal-based hybrid Parkinson's disease diagnosis system for early detection. Testing various feature selection methods and classifiers on a UCI dataset, the combination of genetic algorithm and random forest achieved superior performance with 95.58% accuracy, surpassing recent literature findings.

A. Singh, J. Kumar, and H. Taneja (2023) [13] A Multi-modal Approach for Parkinson's Disease Detection Using Speech and Handwriting Signals. Proposing a mobile application for self-testing Parkinson's symptoms using image processing and machine learning. Achieving 97.39% accuracy for the Trace test (VGG-19) and 93.2% for the Speech test (LGBM Classifier), aiding early diagnosis in areas with limited access to medical facilities.

M. El Hannani, A. El Hannani, A. Amiar, A. Bennis, and O. El Moutaouakkil (2023) [14] Parkinson's Disease Detection Using Deep

Learning and Speech-Language Processing. This study introduces a Parkinson's disease Cross-Language Speech Analysis Model (CLSAM) using adversarial transfer learning and feature decoupling. CLSAM significantly improves accuracy, sensitivity, and F1 scores in cross-language scenarios, addressing limitations of single-language datasets for speech-based Parkinson's disease detection.

A. Bougrine, A. Ouannas, M. Z. Ouahrani, and M. E. Zahi (2023) [15] Parkinson's Disease Detection Using Speech Signals and a Hybrid Deep Learning Model with Attention Mechanism. This study introduces a multiscale hybrid attention network (MSHANet) for automatic Parkinson's disease detection using convolutional neural networks. Utilizing two datasets and innovative classification strategies, the model achieves high accuracy (94.11%) and demonstrates potential for accurate PD diagnosis in clinical settings.

2.1 REQUIREMENTS:

2.1.1 USER REQUIREMENTS:

- **User interface:**

❖ **Ease of Use:** Intuitive interface for healthcare professionals to input patient data easily.

❖ **Visualization:** Interactive charts or graphs to display predictions and model insights in an understandable format.

- **Accuracy and Reliability:**

❖ **High Prediction Accuracy:** Accurate predictions for liver cirrhosis based on patient data and medical history.

❖ **Reliable Results:** Consistent and reliable predictions to aid medical practitioners in diagnosis.

- **Interpretability:**

❖ **Explainable Results:** The system should provide explanations for predictions, highlighting the key features contributing to the diagnosis.

- **Data security and Privacy:**

❖ **Compliance:** Adherence to data protection regulations (HIPAA, GDPR) to ensure patient data privacy and security.

❖ **Secure Storage:** Safe storage of patient information with restricted access to authorized personnel

- **Integration and accessibility:**

❖ **Compatibility:** Ability to integrate with existing hospital systems or databases to fetch patient data seamlessly.

❖ **Accessibility:** Accessible via web or mobile applications for easy use in healthcare settings.

2.1.2 NON-FUNCTIONAL REQUIREMENTS:

- **Performance:**

❖ **Speed:** Efficient model processing and prediction within a reasonable time frame.

❖ **Scalability:** Capability to handle an increased load of patient data without compromising performance.

- **Reliability:**

❖ **Fault Tolerance:** Robustness against system failures or disruptions, ensuring continuous availability.

❖ **Consistency:** Consistent predictions despite variations in input data or workload.

- **Model Quality:**
  - ❖ **Generalizability:** Ability of the model to generalize well on new, unseen patient data.
  - ❖ **Model Interpretability:** Ensuring the model is interpretable, allowing medical practitioners to understand the reasoning behind predictions.

- **Security:**
  - ❖ **Data Encryption:** Encrypting sensitive patient data both during storage and transmission.
  - ❖ **Access Control:** Implementing strict access controls to limit data access to authorized personnel only.

- **Scalability:**
  - ❖ **Resource Scalability:** Ability to scale resources (hardware or cloud infrastructure) as the system usage grows.

- **Ethical and Legal Compliance:**
  - ❖ **Ethical Guidelines:** Abiding by ethical principles in handling patient data and ensuring transparent usage.
  - ❖ **Regulatory Compliance:** Compliance with healthcare regulations and standards governing medical data usage.

2.2 SYSTEM REQUIREMENTS:

2.2.1 HARDWARE REQUIREMENTS:

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the team's and tracking the team's progress throughout the development activity.

PYTHON IDE**:** Anaconda Jupyter Notebook

PROGRAMMING LANGUAGE: Python

Python is widely used for machine learning and data analysis projects. Need Python installed on system.

Data Analysis and Machine Learning Libraries: Require various Python libraries for data preprocessing, model development, and evaluation. These may include:

Pandas: for data manipulation and analysis.

NumPy: for numerical operations.

Scikit-learn: for machine learning algorithms and model evaluation.
Scipy: for scientific and statistical computations.

Matplotlib and Seaborn: for data visualization.

Development Environment: An integrated development environment (IDE) like Jupyter Notebook or Visual Studio Code is beneficial for code development, documentation, and analysis.
Machine learning algorithm models: XGBOOST AND SUPPORT VECTOR MACHINE(SVM)

2.2.2 SOFTWARE REQUIREMENTS:

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shows what the system does and not how it should be implemented.

PROCESSOR : Intel I5

RAM : 4GB

HARD DISK : 40 GB

3. SYSTEM DESIGN:

3.1 ARCHITECTURE DIAGRAM:

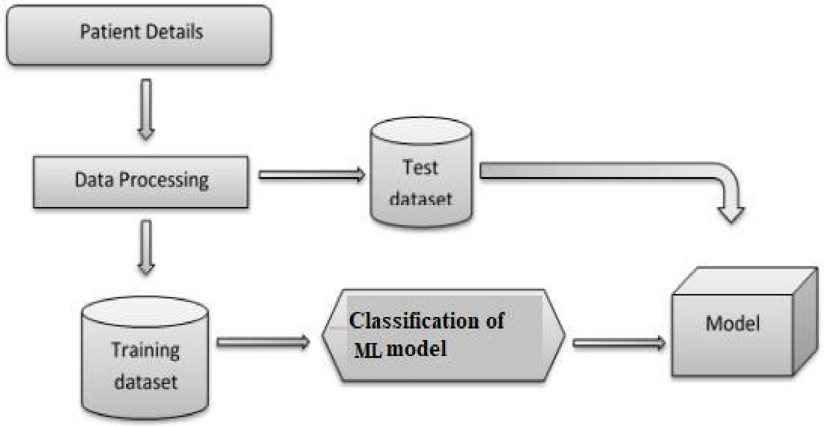


Figure:3 Architecture Diagram

A system architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system. Designing of system is the process in which it is used to define the interface, modules and data for a system to specify the demand to satisfy. System design is seen as the application of the system theory. The main thing of the design a system is to develop the system architecture by giving the data and information that is necessary for the implementation of a system.

4. IMPLEMENTATION OF THE SYSTEM:

4.1 DATA COLLECTION:

Data loading is the process of copying and loading data or data sets from a source file, folder or application to a database or similar application. It is usually implemented by copying digital data from a source and pasting or loading the data to a data storage or processing utility. Data loading is used in database-based extraction and loading techniques. Typically, such data is loaded into the destination application as a different format than the original source location.For example, when data is copied from a word processing file to a database application, the data format is changed from .doc or .txt to a .CSV or DAT format. Usually, this process is performed through or the last phase of the Extract, Transform and Load (ETL) process. The data is extracted from an external source and transformed into the destination application's supported format, where the data is further loaded.

4.2 DATA PREPROCESSING:

Missing values were imputed to guarantee that all the algorithms would be able to handle them. Nevertheless, some algorithms could deal with missing values automatically without imputation, such as XGBoost. To restrict the comparison complexity, the missing values were imputed based on their data type. For numerical data types, the missing entries are replaced by the median value of the complete entries. For categorical data, the missing entries were replaced by the mode value of the complete entries.

4.3 DATA CLEANING:

In this module the data is cleaned. After cleaning of the data, the data is grouped as per requirement. This grouping of data is known as data clustering. Then check if there is any missing value in the data set or not. It there is some missing value then change it by any default value. After that if any data need to change its format, it is done. That total process before the prediction is known is data pre- processing. After that the data is used for the prediction and forecasting step.

4.4 SPLITTING OF DATA:

After cleaning the data, data is normalized in training and testing the model. When data is spitted then we train algorithm on the training data set and keep test data set aside. This training process will produce the training model based on logic and algorithms and values of the feature in training data. Basically aim of feature extraction is to bring all the values under same scale. A dataset used for machine learning should be partitioned into three subsets training, test, and validation sets.

Training set: -A data scientist uses a training set to train a model and define its optimal parameters — parameters it must learn from data.

Test set: - A test set is needed for an evaluation of the trained model and its capability for generalization. The latter means a model's ability to identify patterns in new unseen data after having been trained over a training data. It's crucial to use different subsets for training and testing to avoid model over fitting, which is the incapacity for generalization we mentioned above.

In data science or machine learning, data splitting comes into the picture when the given data is divided into two or more subsets so that a model can get trained, tested and evaluated. In practice or in real-life projects, data splitting is an important aspect, and it becomes a must when models are based on the data as it ensures the making of machine learning models. Usually, we create two or three parts of the main dataset.

Data splitting becomes a necessary step to be followed in machine learning modelling because it helps right from training to the evaluation of the model. We should divide our whole dataset into three sub-dataset. The quantity of training data should be higher than the other two data. Also, it should be unbiased to any class or category, so that model can adequately learn from the data. We should use the validation set for evaluating multiple models to find the best-performing model. After finding the best-performing model, we use the test set to quantify the model's performance.

4.5 CLASSIFICATION:

When data has been ready, we apply Machine Learning Technique. We use different classification and ensemble techniques, to predict mental illness. The methods applied on csv dataset. Main objective to apply Machine Learning Techniques to analyze the performance of these methods and find presence of them.

4.6 CLASSIFIER TRAINING:

A classifier is a function that takes features as input and generates a class label prediction. Based on the learning function and underlying assumptions, different types of classifiers can be developed. Neuroimaging studies have applied various classifiers for mental illness prediction. The dimensionality issue associated with the relatively large number of features and the small number of samples should be accounted for while applying such classification algorithms.

4.7 DATA SPLITTING:

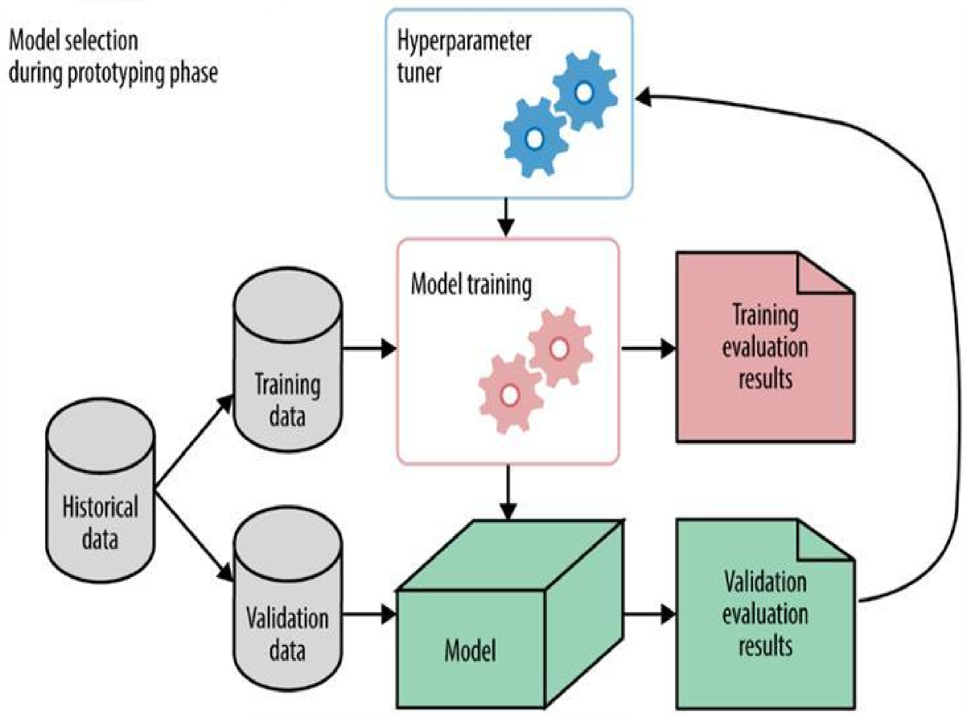
For each experiment, we split the entire dataset into 70% training set and 30% test set. We used the training set for resampling, hyper parameter tuning, and training the model and we used test set to test the performance of the trained model. While splitting the data, we specified a random seed (any random number), which ensured the same data split every time the program executed.

4.8 DATA TRAINING:

Algorithms learn from data. They find relationships, develop understanding, make decisions, and evaluate their confidence from the training data they're given. And the better the training data is, the better the model performs.In fact, the quality and quantity of your training data has as much to do with the success of your data project as the algorithms themselves.Now, even if you've stored a vast amount of well-structured data, it might not be labeled in a way that actually works for training your model. For example, autonomous vehicles don't just need pictures of the road, they need labeled images where each car, pedestrian, street sign and more are annotated; sentiment analysis projects require labels that help an algorithm understand when someone's using slang or sarcasm; chatbots need entity extraction and careful syntactic analysis, not just raw language. In other words, the data you want to use for training usually needs to be enriched or labeled. Or you might just need to collect more of it to power your algorithms. But chances are, the data you've stored isn't quite ready to be used to train your classifiers. Because if you're trying to make a great model, you need great training data. And we know a thing or two about that. After all, we've labeled over 5 billion rows of data for some of the most innovative companies in the world. Whether it's images, text, audio, or, really, any other kind of data, we can help create the training set that makes your models successful.

4.9 MODEL TESTING:

A Hyper parameter Tuning Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters.



(FIG 4)

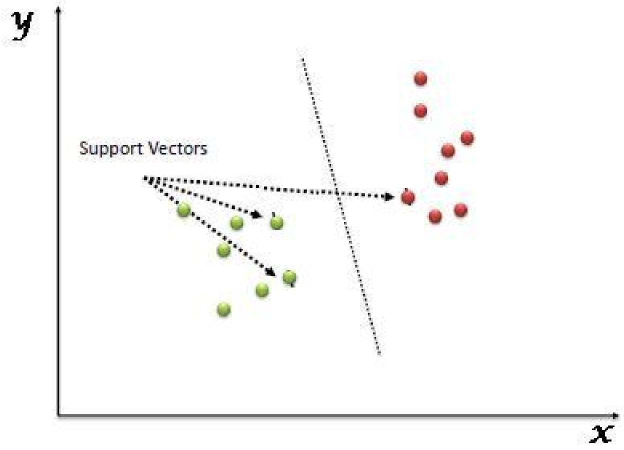
However, there is another kind of parameters, known as Hyperparameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.The aim of this module is to explore various strategies to tune hyper parameter for Machine learning model.Some examples of model hyper parameters include:

1. The penalty in Logistic Regression Classifier i.e., L1 or L2 regularization
2. The learning rate for training a neural network.
3. The C and sigma hyper parameters for support vector machines.
4. The k in k-nearest neighbors.

4.10 ALGORITHM:

4.10.1 SUPPORT VECTOR MACHINE:

“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).



(FIG.5)

Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).In Classification process, when data has been ready we apply Machine Learning Technique. We use different classification and ensemble techniques, to predict Parkinson disease detection. The methods applied on Pima Indians parkinson dataset. Main objective to apply Machine Learning Techniques to analyze the performance of these methods and find accuracy of them, and also been able to figure out the responsible/important feature which play a major role in prediction. The Techniques are follows:

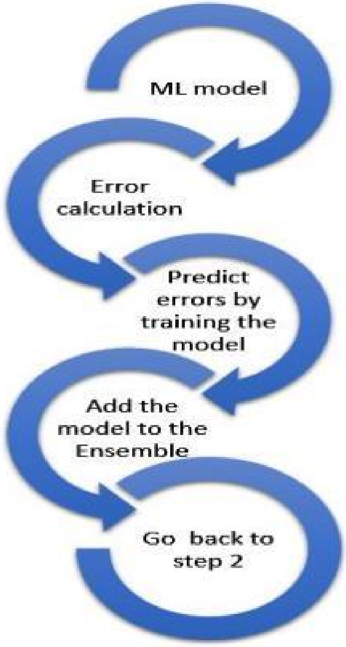
Support Vector Machine also known as SVM is a supervised machine learning algorithm. SVM is most popular classification technique. SVM creates a hyper plane that separate two classes. It can create a hyper plane or set of hyper plane in high dimensional space. This hyper plane can be used for classification or regression also. SVM differentiates instances in specific classes and can also classify the entities which are not supported by data. Separation is done by through hyper plane performs the separation to the closest training point of any class.

- Select the hyper plane which divides the class better. To find the better hyper plane you have to calculate the distance between the planes and the data which is called Margin.

- If the distance between the classes is low then the chance of miss conception is high and vice versa. So we need to Select the class which has the high margin. Margin = distance to positive point + Distance to negative point.

4.10.2 XGBOOST CLASSIFIER:

XGBoost Researchers have devised an algorithm called XGBoost for machine learning categorization that is incredibly effective. It is extremely quick, and its performance is improved because it is a boosted decision tree. This categorization prototype is used to boost the model's efficiency and speed. XGBoost stands for Extreme Gradient Boost. It is the streamlined group calculation dependent on Gradient Boosting Decision Tree.The principle thought of the boosting calculation is that numerous decision trees perform superior to a solitary one. Every decision tree may make an awful showing. At the point when numerous trees are incorporated, the presentation shows signs of improvement.


ML model
Error calculation
Predict errors by training the model
Add the model to the Ensemble
Go back to step 2

(FIG.6)

XGBoost is a decision tree-based gathering Machine Learning calculation that utilizes an inclination boosting system. In forecast issues including unstructured data (pictures, content, and so on.) artificial neural systems will in general beat every other calculation or structures. Be that as it may, with regards to little to-medium organized/forbidden data, decision tree based calculations are viewed as best.

4.10.3 DATA PREDICTION AND FORECASTING:

In this step, the pre-processed data is taken for the prediction. This prediction can be done in any process which are mentioned above. But the XGBoost algorithm score more prediction accuracy than the other algorithm. So, in this project the linear regression method is used for the prediction. For that, the pre-processed data is splitted for the train and test purpose. Then a predictive object is created to predict the test value which is trained by the trained value. Then the object is used to forecast data for next few years.

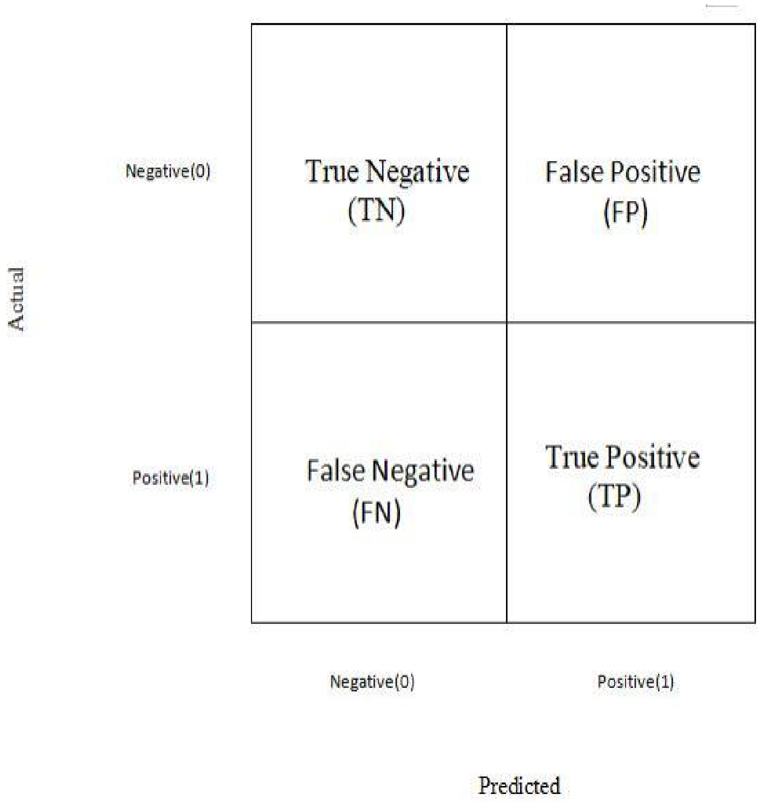
4.11 PERFORMANCE MATRICES:

Data was divided into two portions, training data and testing data, both these portions consisting 70% and 30% data respectively. All these six algorithms were applied on same dataset using Enthought Canaopy and results were obtained. Predicting accuracy is the main evaluation parameter that we used in this work. Accuracy can be defied using equation. Accuracy is the overall success rate of the algorithm.

$$\text{ACCURACY} = (TP+TN) / (P+N)$$

4.11.1 CONFUSION MATRIX:

It is the most commonly used evaluation metrics in predictive analysis mainly because it is very easy to understand and it can be used to compute other essential metrics such as accuracy, recall, precision, etc. It is an NxN matrix that describes the overall performance of a model when used on some dataset, where N is the number of class labels in the classification problem.



(FIG.7)

True positive (TP) indicates that the positive class is predicted as a positive class, and the number of sample positive classes was actually predicted by the model. False negative indicates (FN) that the positive class is predicted as a negative class, and the number of negative classes in the sample was actually predicted by the model. False positive (FP) indicates that the negative class is predicted as a positive class, and the number of positive classes of samples was actually predicted by the model. True negative (TN) indicates that the negative class is predicted as a negative class.

5. RESULTS & DISCUSSION:

Results:

Speech or voice data is assumed to be 90% helpful to diagnose a person for identifying presence of disease. In general, Person with PD suffer from speech problems, categorized into two: hypophonia and dysarthria. Hypophonia indicates very soft and weak voice from a person and dysarthria indicate slow speech or voice, that can hardly be understood at one time, and this causes because of damage to central nervous system. So, most of the clinicians who treat PD patients observe dysarthria and try to rehabilitate with specific treatments to improvise vocal intensity.

Discussion:

Several strategies are recorded for early detection of PD based on the different ML techniques. But accuracy in detection and classifying within the time is very important or else, it causes development of more symptoms. There are different kinds of data, brain MRI images, Voice data, posture images, senor captured data, handwritten data, using which we can predict whether person is having PD or not. Out of all those, speech or voice data helps in identifying PD accurately.

6. CONCLUSION & FUTURE WORKS:

This project is an effort to present broad review about Parkinson disease diagnosis system that have applied various deep learning techniques. This project aimed to cover a broader space of imaging and machine learning technologies for mental illness diagnostics such that researchers in the field could readily identify the state of the art in the domain. Moreover, we emphasize the importance of early detection and prediction of Parkinson's disease, such that treatment and support can be provided to patients as soon as possible. It can be identified that maximum of all ML techniques used by various authors worked better but developing a very faster classifier using novel architecture of ML combined with specific approach may work better. To achieve this, we try to implement convolution neural network with different number of ML and number of nodes in future and compare all the accuracies.

6.1 FUTURE WORK:

In future work, we can focus on different techniques to predict the Parkinson disease using different datasets. In this research, we using binary attribute (1- diseased patients, 0-non-diseased patients) for patient's classification. In the future we will use different types of attributes for the classification of patients and also identify the different st ages of Parkinson's disease.

REFERENCES:

[1]. Marwa Almasoud, Tomas E Ward2 (2021). Parkinson disease gait classification based on machine learning approach.

[2]. Sanghee Moon, Hyun-Je Song, Vibhash D. Sharma, Kelly E. Lyons, Rajesh Pahwa (2022). Classification of parkinson's disease and essential tremor based on balance and gait characteristics from wearable motion sensors via machine learning.

[3]. Basetty Mallikarjuna, R. Viswanathan and Bharat Bhushan Naib (2022). Feedback-based gait identification using deep neural network classification.

[4]. Rana Zia Ur Rehman, Silvia Del Din, Yu Guan, Alison J. Yarnall. (2022). Selecting clinically relevant gait characteristics for classi -fication of early parkinson's disease.

[5]. Milla Juutinen,Justin Zhu, Cassia Wang (2022). Perkinson's disease detection from 20-step walking tests using inertial sensors of smartphone.

[6]. R. Prashanth, S. D. Roy, and V. Pillay (2023). Early detection of Parkinson's disease using speech features and a deep learning model.

[7]. H. Gunduz (2023). A novel approach for Parkinson's disease detection using speech signals and transfer learning.

[8]. A. Mostafa, Z. Mohammed, S. Hamad Khaleefah Al-Dulaimi, and A. Mustapha (2022). A hybrid deep learning model for Parkinson's disease detection using voice signals

[9]. A. H. Chen, Y. Zhang, Y. Wang, T. Xu, and T. Chen(2022). Parkinson's disease detection based on multi-scale convolutional neural network and speech features.

[10]. S. Lahmiri, A. Shmuel, and A. Benyoussef (2022). A novel deep learning approach for Parkinson's disease detection using speech signals.

[11]. M. El Hannani, A. El Hannani, A. Amiar, A. Bennis, and O. El Moutaouakkil (2022). Parkinson's disease detection using deep learning and speech signal analysis.

[12]. A. Bougrine, A. Ouannas, M. Z. Ouahrani, and M. E. Zahi (2022). Parkinson's disease detection using speech signals and a hybrid deep learning model.

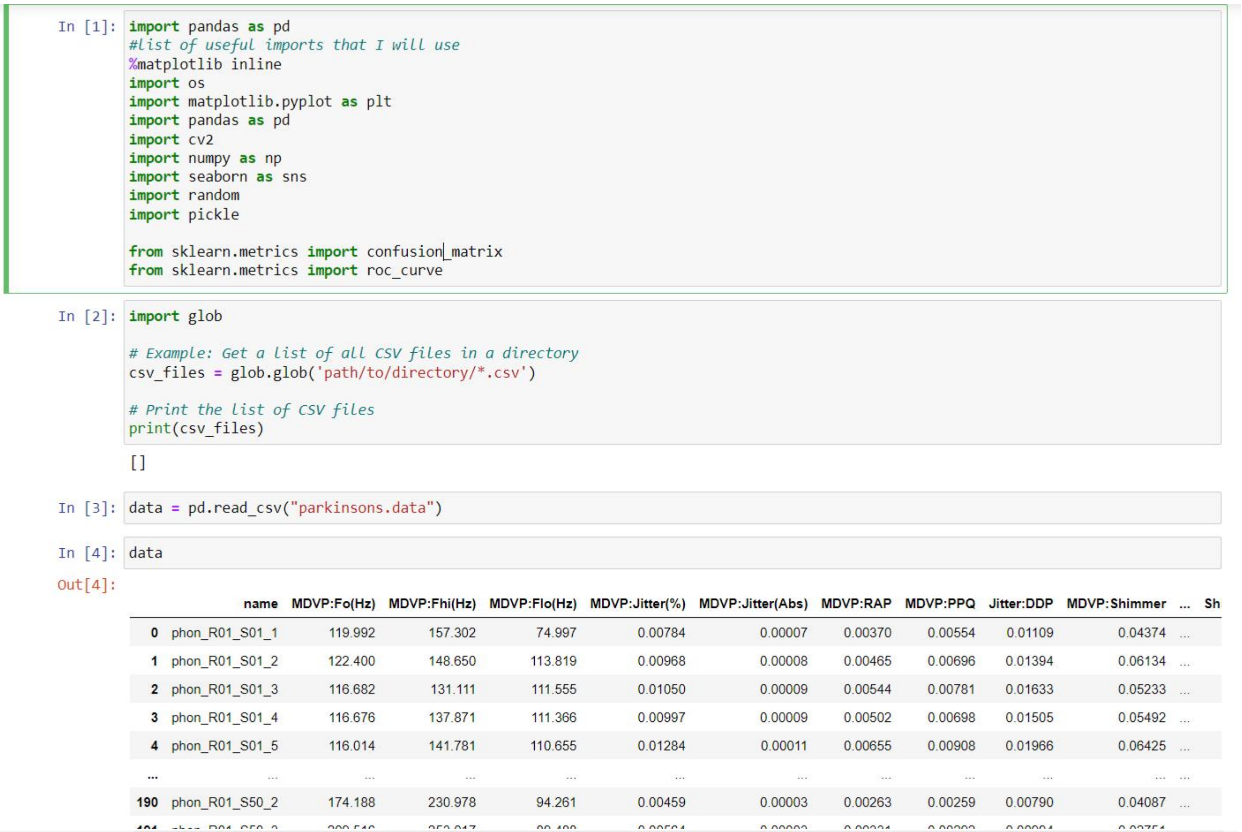
[13]. A. Singh, J. Kumar, and H. Taneja (2023). A Multi-modal Approach for Parkinson's Disease Detection Using Speech and Handwriting Signals.

[14]. M. El Hannani, A. El Hannani, A. Amiar, A. Bennis, and O. El Moutaouakkil (2023). Parkinson's Disease Detection Using Deep Learning and Speech-Language Processing.

[15]. A. Bougrine, A. Ouannas, M. Z. Ouahrani, and M. E. Zahi (2023). Parkinson's Disease Detection Using Speech Signals and a Hybrid Deep Learning Model with Attention Mechanism.

Appendix <Sample code, snapshot etc.>

IMPLEMENTATION:



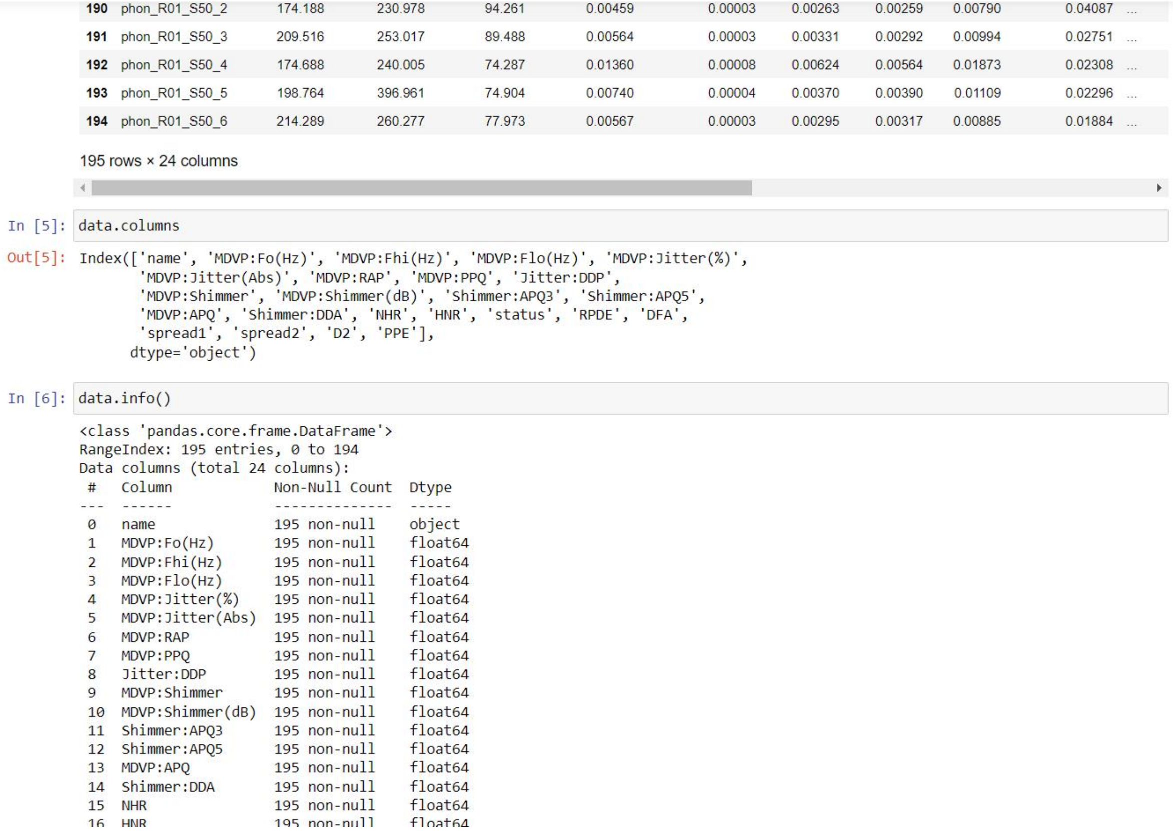
```
In [1]: import pandas as pd
        #list of useful imports that I will use
        %matplotlib inline
        import os
        import matplotlib.pyplot as plt
        import pandas as pd
        import cv2
        import numpy as np
        import seaborn as sns
        import random
        import pickle

        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import roc_curve
```

```
In [2]: import glob

        # Example: Get a list of all CSV files in a directory
        csv_files = glob.glob('path/to/directory/*.csv')

        # Print the list of CSV files
        print(csv_files)

        []
```

```
In [3]: data = pd.read_csv("parkinsons.data")
```

```
In [4]: data
Out[4]:
```

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...	Sh
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	...	
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	...	
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	...	
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	...	
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
190	phon_R01_S50_2	174.188	230.978	94.261	0.00459	0.00003	0.00263	0.00259	0.00790	0.04087	...	



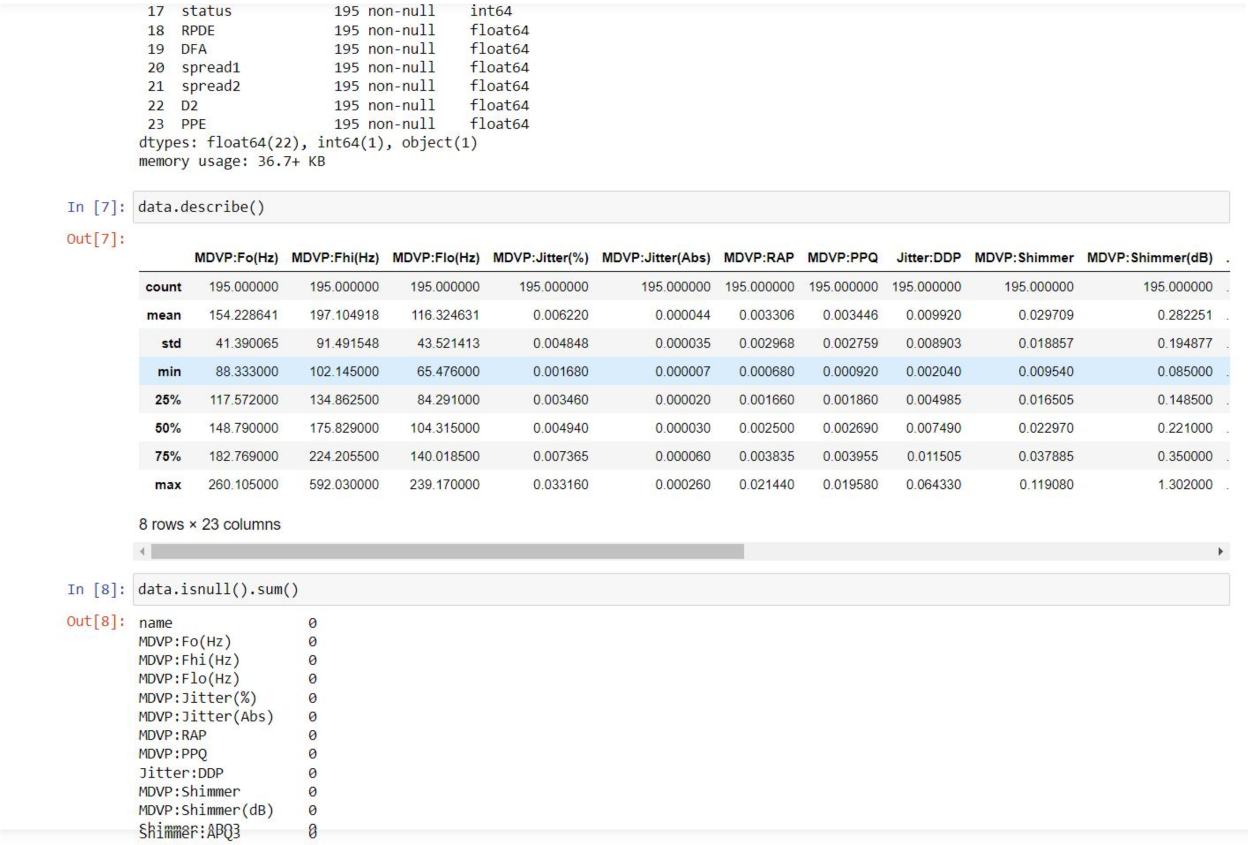
190 phon_R01_S50_2 174.188 230.978 94.261 0.00459 0.00003 0.00263 0.00259 0.00790 0.04087 ...

191	phon_R01_S50_3	209.516	253.017	89.488	0.00564	0.00003	0.00331	0.00292	0.00994	0.02751	...
192	phon_R01_S50_4	174.688	240.005	74.287	0.01360	0.00008	0.00624	0.00564	0.01873	0.02308	...
193	phon_R01_S50_5	198.764	396.961	74.904	0.00740	0.00004	0.00370	0.00390	0.01109	0.02296	...
194	phon_R01_S50_6	214.289	260.277	77.973	0.00567	0.00003	0.00295	0.00317	0.00885	0.01884	...

195 rows × 24 columns

```
In [5]: data.columns
Out[5]: Index(['name', 'MDVP:Fo(Hz)', 'MDVP:Fhi(Hz)', 'MDVP:Flo(Hz)', 'MDVP:Jitter(%)',
               'MDVP:Jitter(Abs)', 'MDVP:RAP', 'MDVP:PPQ', 'Jitter:DDP',
               'MDVP:Shimmer', 'MDVP:Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5',
               'MDVP:APQ', 'Shimmer:DDA', 'NHR', 'HNR', 'status', 'RPDE', 'DFA',
               'spread1', 'spread2', 'D2', 'PPE'],
              dtype='object')
```

```
In [6]: data.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 195 entries, 0 to 194
        Data columns (total 24 columns):
         #   Column            Non-Null Count  Dtype
        ---  ------            --------------  -----
         0   name              195 non-null    object
         1   MDVP:Fo(Hz)       195 non-null    float64
         2   MDVP:Fhi(Hz)      195 non-null    float64
         3   MDVP:Flo(Hz)      195 non-null    float64
         4   MDVP:Jitter(%)    195 non-null    float64
         5   MDVP:Jitter(Abs)  195 non-null    float64
         6   MDVP:RAP          195 non-null    float64
         7   MDVP:PPQ          195 non-null    float64
         8   Jitter:DDP        195 non-null    float64
         9   MDVP:Shimmer      195 non-null    float64
         10  MDVP:Shimmer(dB)  195 non-null    float64
         11  Shimmer:APQ3      195 non-null    float64
         12  Shimmer:APQ5      195 non-null    float64
         13  MDVP:APQ          195 non-null    float64
         14  Shimmer:DDA       195 non-null    float64
         15  NHR               195 non-null    float64
         16  HNR               195 non-null    float64
```



```
 17  status            195 non-null    int64
 18  RPDE              195 non-null    float64
 19  DFA               195 non-null    float64
 20  spread1           195 non-null    float64
 21  spread2           195 non-null    float64
 22  D2                195 non-null    float64
 23  PPE               195 non-null    float64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

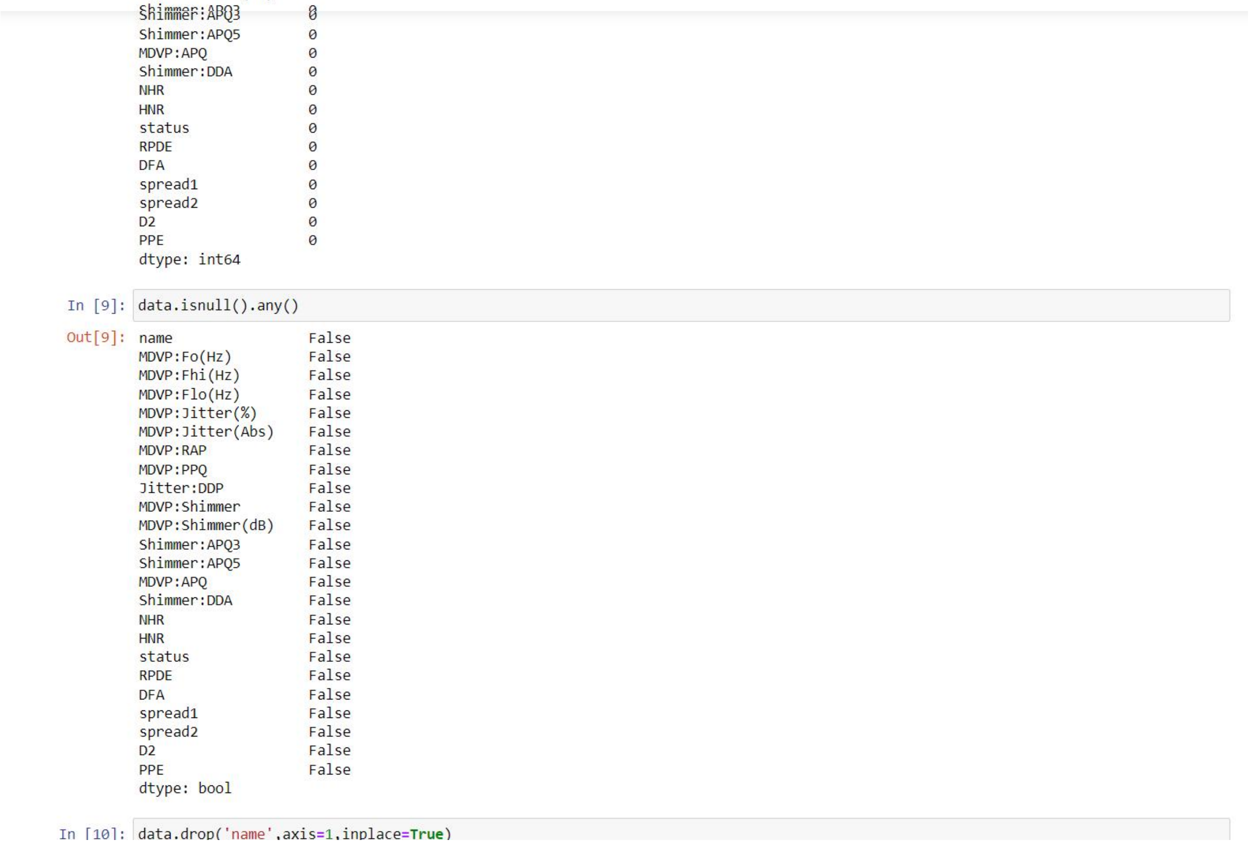
```
In [7]: data.describe()
```

Out[7]:

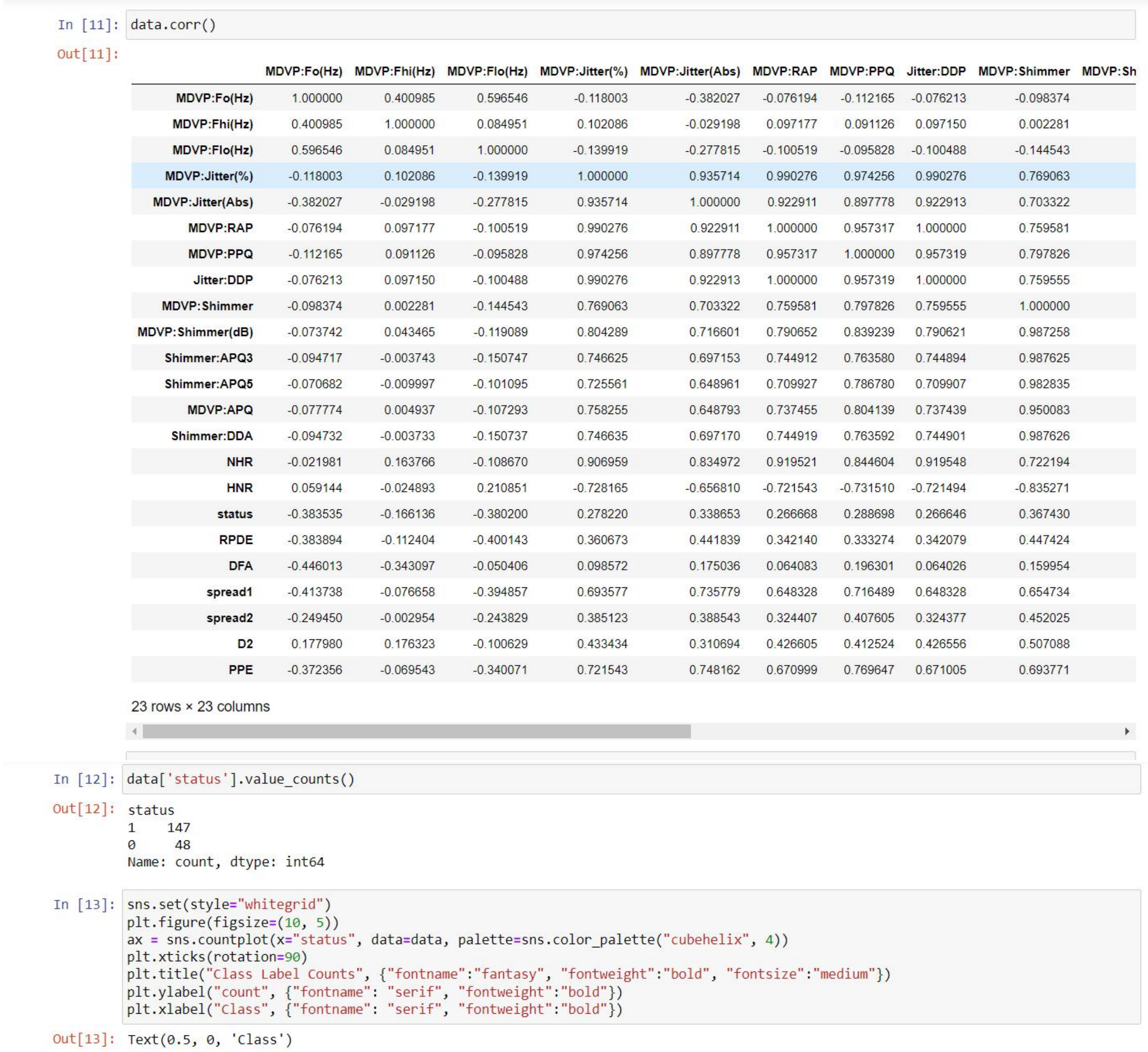
	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)	.
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	.
mean	154.228641	197.104918	116.324631	0.006220	0.000044	0.003306	0.003446	0.009920	0.029709	0.282251	.
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	0.002759	0.008903	0.018857	0.194877	.
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680	0.000920	0.002040	0.009540	0.085000	.
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660	0.001860	0.004985	0.016505	0.148500	.
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500	0.002690	0.007490	0.022970	0.221000	.
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835	0.003955	0.011505	0.037885	0.350000	.
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440	0.019580	0.064330	0.119080	1.302000	.

8 rows × 23 columns

```
In [8]: data.isnull().sum()

Out[8]: name                0
        MDVP:Fo(Hz)         0
        MDVP:Fhi(Hz)        0
        MDVP:Flo(Hz)        0
        MDVP:Jitter(%)      0
        MDVP:Jitter(Abs)    0
        MDVP:RAP            0
        MDVP:PPQ            0
        Jitter:DDP          0
        MDVP:Shimmer        0
        MDVP:Shimmer(dB)    0
        Shimmer:APQ3        0
```



```
        Shimmer:APQ5        0
        MDVP:APQ            0
        Shimmer:DDA         0
        NHR                 0
        HNR                 0
        status              0
        RPDE                0
        DFA                 0
        spread1             0
        spread2             0
        D2                  0
        PPE                 0
        dtype: int64

In [9]: data.isnull().any()

Out[9]: name                False
        MDVP:Fo(Hz)         False
        MDVP:Fhi(Hz)        False
        MDVP:Flo(Hz)        False
        MDVP:Jitter(%)      False
        MDVP:Jitter(Abs)    False
        MDVP:RAP            False
        MDVP:PPQ            False
        Jitter:DDP          False
        MDVP:Shimmer        False
        MDVP:Shimmer(dB)    False
        Shimmer:APQ3        False
        Shimmer:APQ5        False
        MDVP:APQ            False
        Shimmer:DDA         False
        NHR                 False
        HNR                 False
        status              False
        RPDE                False
        DFA                 False
        spread1             False
        spread2             False
        D2                  False
        PPE                 False
        dtype: bool

In [10]: data.drop('name',axis=1,inplace=True)
```



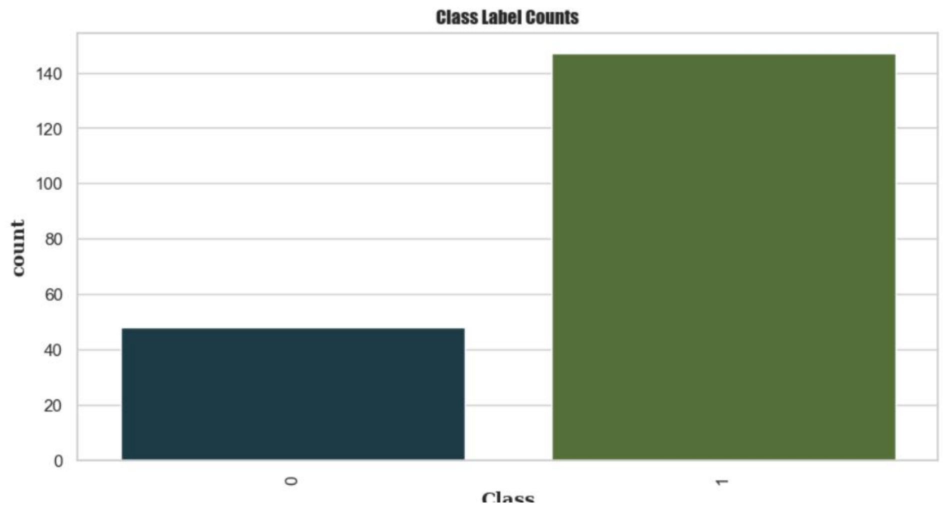
```
In [11]: data.corr()
```

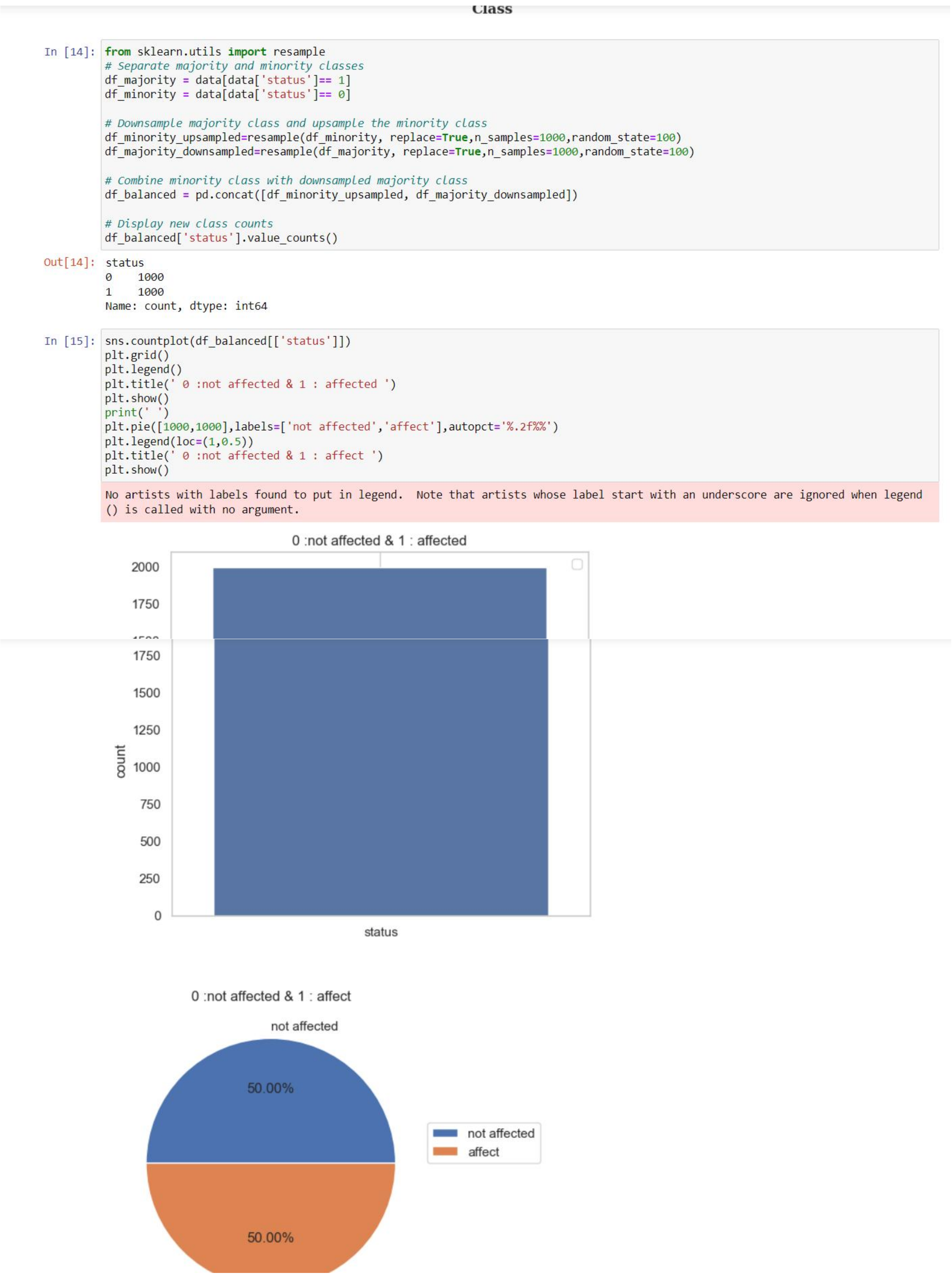
Out[11]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Sh
MDVP:Fo(Hz)	1.000000	0.400985	0.596546	-0.118003	-0.382027	-0.076194	-0.112165	-0.076213	-0.098374	
MDVP:Fhi(Hz)	0.400985	1.000000	0.084951	0.102086	-0.029198	0.097177	0.091126	0.097150	0.002281	
MDVP:Flo(Hz)	0.596546	0.084951	1.000000	-0.139919	-0.277815	-0.100519	-0.095828	-0.100488	-0.144543	
MDVP:Jitter(%)	-0.118003	0.102086	-0.139919	1.000000	0.935714	0.990276	0.974256	0.990276	0.769063	
MDVP:Jitter(Abs)	-0.382027	-0.029198	-0.277815	0.935714	1.000000	0.922911	0.897778	0.922913	0.703322	
MDVP:RAP	-0.076194	0.097177	-0.100519	0.990276	0.922911	1.000000	0.957317	1.000000	0.759581	
MDVP:PPQ	-0.112165	0.091126	-0.095828	0.974256	0.897778	0.957317	1.000000	0.957319	0.797826	
Jitter:DDP	-0.076213	0.097150	-0.100488	0.990276	0.922913	1.000000	0.957319	1.000000	0.759555	
MDVP:Shimmer	-0.098374	0.002281	-0.144543	0.769063	0.703322	0.759581	0.797826	0.759555	1.000000	
MDVP:Shimmer(dB)	-0.073742	0.043465	-0.119089	0.804289	0.716601	0.790652	0.839239	0.790621	0.987258	
Shimmer:APQ3	-0.094717	-0.003743	-0.150747	0.746625	0.697153	0.744912	0.763580	0.744894	0.987625	
Shimmer:APQ5	-0.070682	-0.009997	-0.101095	0.725561	0.648961	0.709927	0.786780	0.709907	0.982835	
MDVP:APQ	-0.077774	0.004937	-0.107293	0.758255	0.648793	0.737455	0.804139	0.737439	0.950083	
Shimmer:DDA	-0.094732	-0.003733	-0.150737	0.746635	0.697170	0.744919	0.763592	0.744901	0.987626	
NHR	-0.021981	0.163766	-0.108670	0.906959	0.834972	0.919521	0.844604	0.919548	0.722194	
HNR	0.059144	-0.024893	0.210851	-0.728165	-0.656810	-0.721543	-0.731510	-0.721494	-0.835271	
status	-0.383535	-0.166136	-0.380200	0.278220	0.338653	0.266668	0.288698	0.266646	0.367430	
RPDE	-0.383894	-0.112404	-0.400143	0.360673	0.441839	0.342140	0.333274	0.342079	0.447424	
DFA	-0.446013	-0.343097	-0.050406	0.098572	0.175036	0.064083	0.196301	0.064026	0.159954	
spread1	-0.413738	-0.076658	-0.394857	0.693577	0.735779	0.648328	0.716489	0.648328	0.654734	
spread2	-0.249450	-0.002954	-0.243829	0.385123	0.388543	0.324407	0.407605	0.324377	0.452025	
D2	0.177980	0.176323	-0.100629	0.433434	0.310694	0.426605	0.412524	0.426556	0.507088	
PPE	-0.372356	-0.069543	-0.340071	0.721543	0.748162	0.670999	0.769647	0.671005	0.693771	

23 rows × 23 columns

```
In [12]: data['status'].value_counts()

Out[12]: status
         1    147
         0     48
         Name: count, dtype: int64

In [13]: sns.set(style="whitegrid")
         plt.figure(figsize=(10, 5))
         ax = sns.countplot(x="status", data=data, palette=sns.color_palette("cubehelix", 4))
         plt.xticks(rotation=90)
         plt.title("Class Label Counts", {"fontname":"fantasy", "fontweight":"bold", "fontsize":"medium"})
         plt.ylabel("count", {"fontname": "serif", "fontweight":"bold"})
         plt.xlabel("Class", {"fontname": "serif", "fontweight":"bold"})

Out[13]: Text(0.5, 0, 'Class')
```



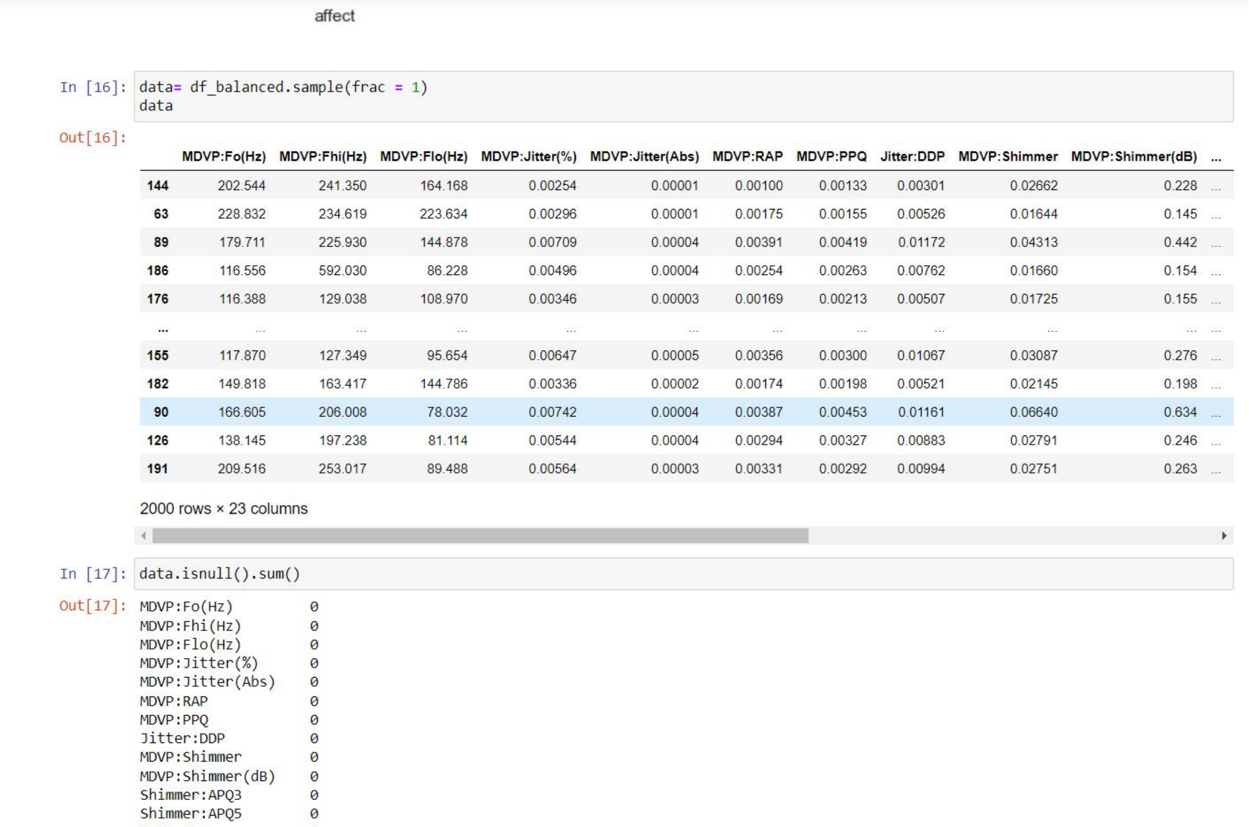


Class

```
In [14]: from sklearn.utils import resample
         # Separate majority and minority classes
         df_majority = data[data['status']== 1]
         df_minority = data[data['status']== 0]

         # Downsample majority class and upsample the minority class
         df_minority_upsampled=resample(df_minority, replace=True,n_samples=1000,random_state=100)
         df_majority_downsampled=resample(df_majority, replace=True,n_samples=1000,random_state=100)

         # Combine minority class with downsampled majority class
         df_balanced = pd.concat([df_minority_upsampled, df_majority_downsampled])

         # Display new class counts
         df_balanced['status'].value_counts()

Out[14]: status
         0    1000
         1    1000
         Name: count, dtype: int64
```

```
In [15]: sns.countplot(df_balanced[['status']])
         plt.grid()
         plt.legend()
         plt.title(' 0 :not affected & 1 : affected ')
         plt.show()
         print(' ')
         plt.pie([1000,1000],labels=['not affected','affect'],autopct='%.2f%%')
         plt.legend(loc=(1,0.5))
         plt.title(' 0 :not affected & 1 : affect ')
         plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend
() is called with no argument.
```



affect

In [16]:
```
data= df_balanced.sample(frac = 1)
data
```

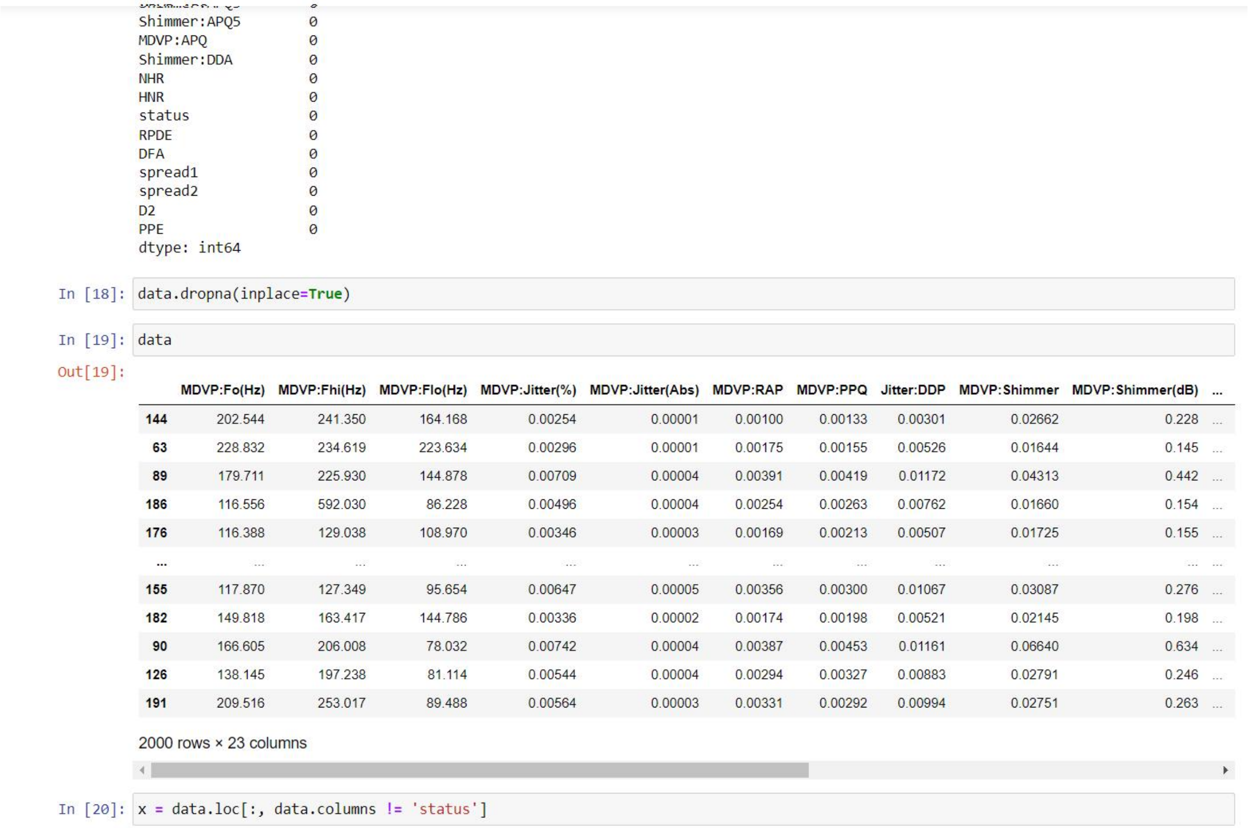
Out[16]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)	...
144	202.544	241.350	164.168	0.00254	0.00001	0.00100	0.00133	0.00301	0.02662	0.228	...
63	228.832	234.619	223.634	0.00296	0.00001	0.00175	0.00155	0.00526	0.01644	0.145	...
89	179.711	225.930	144.878	0.00709	0.00004	0.00391	0.00419	0.01172	0.04313	0.442	...
186	116.556	592.030	86.228	0.00496	0.00004	0.00254	0.00263	0.00762	0.01660	0.154	...
176	116.388	129.038	108.970	0.00346	0.00003	0.00169	0.00213	0.00507	0.01725	0.155	...
...	...	...	...	...	...	...	...	...	...	...	...
155	117.870	127.349	95.654	0.00647	0.00005	0.00356	0.00300	0.01067	0.03087	0.276	...
182	149.818	163.417	144.786	0.00336	0.00002	0.00174	0.00198	0.00521	0.02145	0.198	...
90	166.605	206.008	78.032	0.00742	0.00004	0.00387	0.00453	0.01161	0.06640	0.634	...
126	138.145	197.238	81.114	0.00544	0.00004	0.00294	0.00327	0.00883	0.02791	0.246	...
191	209.516	253.017	89.488	0.00564	0.00003	0.00331	0.00292	0.00994	0.02751	0.263	...

2000 rows × 23 columns

In [17]:
```
data.isnull().sum()
```

Out[17]:
```
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
```



```
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
status              0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
dtype: int64
```

In [18]:
```
data.dropna(inplace=True)
```

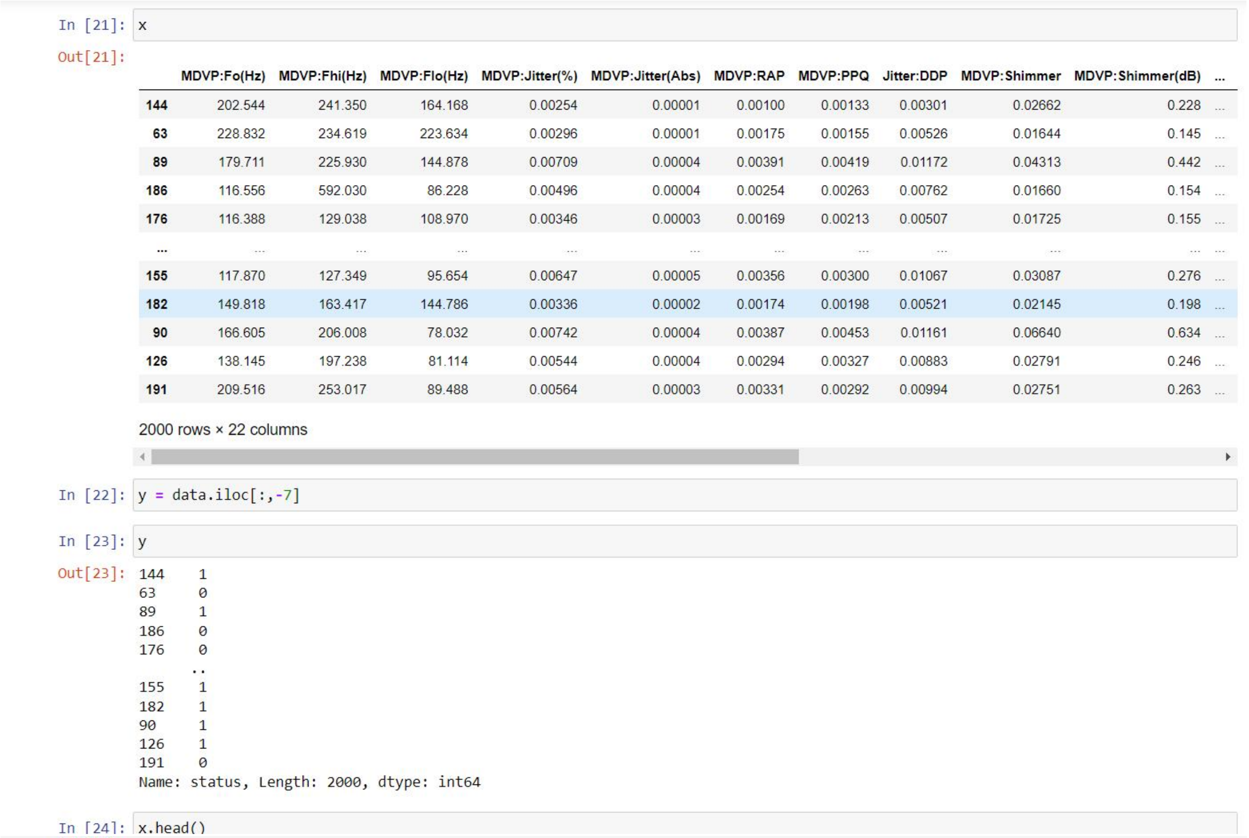
In [19]:
```
data
```

Out[19]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)	...
144	202.544	241.350	164.168	0.00254	0.00001	0.00100	0.00133	0.00301	0.02662	0.228	...
63	228.832	234.619	223.634	0.00296	0.00001	0.00175	0.00155	0.00526	0.01644	0.145	...
89	179.711	225.930	144.878	0.00709	0.00004	0.00391	0.00419	0.01172	0.04313	0.442	...
186	116.556	592.030	86.228	0.00496	0.00004	0.00254	0.00263	0.00762	0.01660	0.154	...
176	116.388	129.038	108.970	0.00346	0.00003	0.00169	0.00213	0.00507	0.01725	0.155	...
...	...	...	...	...	...	...	...	...	...	...	...
155	117.870	127.349	95.654	0.00647	0.00005	0.00356	0.00300	0.01067	0.03087	0.276	...
182	149.818	163.417	144.786	0.00336	0.00002	0.00174	0.00198	0.00521	0.02145	0.198	...
90	166.605	206.008	78.032	0.00742	0.00004	0.00387	0.00453	0.01161	0.06640	0.634	...
126	138.145	197.238	81.114	0.00544	0.00004	0.00294	0.00327	0.00883	0.02791	0.246	...
191	209.516	253.017	89.488	0.00564	0.00003	0.00331	0.00292	0.00994	0.02751	0.263	...

2000 rows × 23 columns

In [20]:
```
x = data.loc[:, data.columns != 'status']
```



```
In [21]: x
```

Out[21]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)	...
144	202.544	241.350	164.168	0.00254	0.00001	0.00100	0.00133	0.00301	0.02662	0.228	...
63	228.832	234.619	223.634	0.00296	0.00001	0.00175	0.00155	0.00526	0.01644	0.145	...
89	179.711	225.930	144.878	0.00709	0.00004	0.00391	0.00419	0.01172	0.04313	0.442	...
186	116.556	592.030	86.228	0.00496	0.00004	0.00254	0.00263	0.00762	0.01660	0.154	...
176	116.388	129.038	108.970	0.00346	0.00003	0.00169	0.00213	0.00507	0.01725	0.155	...
...	...	...	...	...	...	...	...	...	...	...	...
155	117.870	127.349	95.654	0.00647	0.00005	0.00356	0.00300	0.01067	0.03087	0.276	...
182	149.818	163.417	144.786	0.00336	0.00002	0.00174	0.00198	0.00521	0.02145	0.198	...
90	166.605	206.008	78.032	0.00742	0.00004	0.00387	0.00453	0.01161	0.06640	0.634	...
126	138.145	197.238	81.114	0.00544	0.00004	0.00294	0.00327	0.00883	0.02791	0.246	...
191	209.516	253.017	89.488	0.00564	0.00003	0.00331	0.00292	0.00994	0.02751	0.263	...

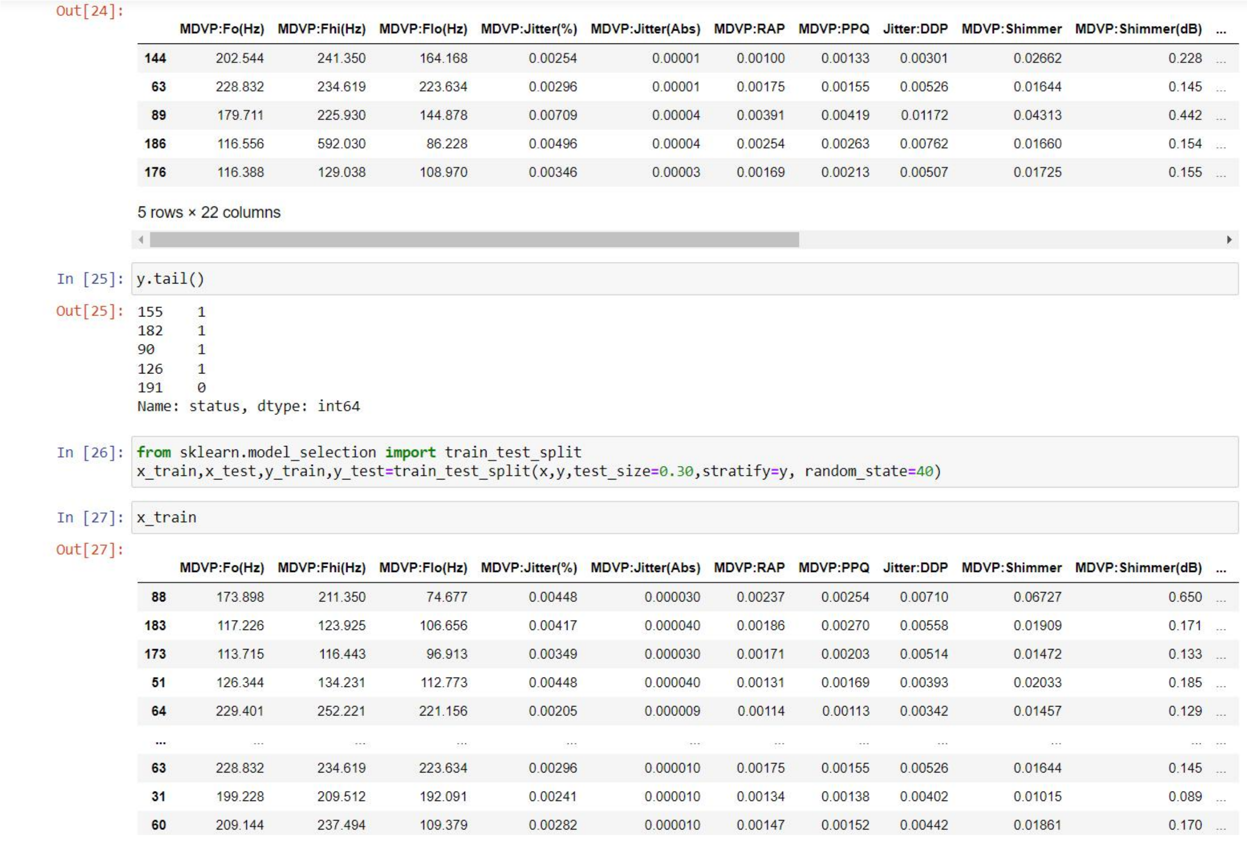
2000 rows × 22 columns

```
In [22]: y = data.iloc[:,-7]
```

```
In [23]: y
```

```
Out[23]: 144    1
         63     0
         89     1
         186    0
         176    0
               ..
         155    1
         182    1
         90     1
         126    1
         191    0
         Name: status, Length: 2000, dtype: int64
```

```
In [24]: x.head()
```



Out[24]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)	...
144	202.544	241.350	164.168	0.00254	0.00001	0.00100	0.00133	0.00301	0.02662	0.228	...
63	228.832	234.619	223.634	0.00296	0.00001	0.00175	0.00155	0.00526	0.01644	0.145	...
89	179.711	225.930	144.878	0.00709	0.00004	0.00391	0.00419	0.01172	0.04313	0.442	...
186	116.556	592.030	86.228	0.00496	0.00004	0.00254	0.00263	0.00762	0.01660	0.154	...
176	116.388	129.038	108.970	0.00346	0.00003	0.00169	0.00213	0.00507	0.01725	0.155	...

5 rows × 22 columns

```
In [25]: y.tail()
```

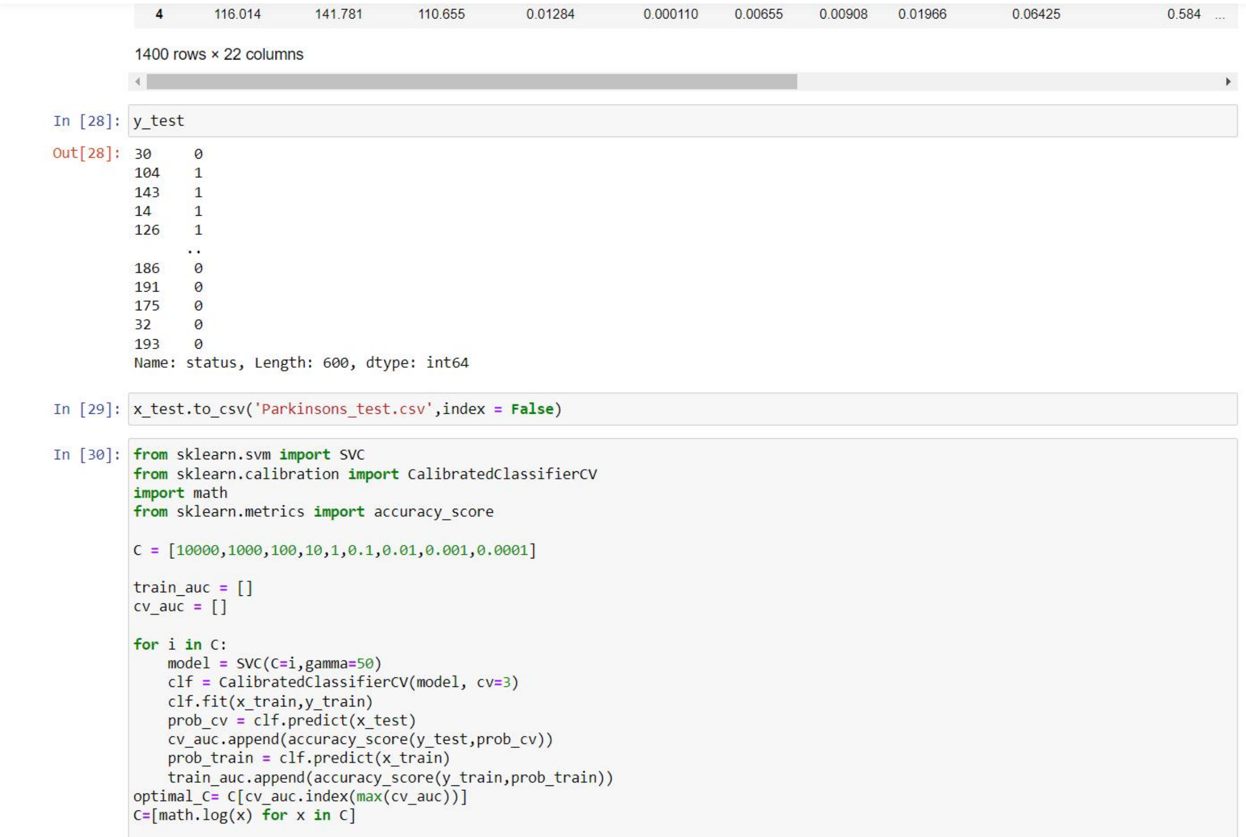
```
Out[25]: 155    1
         182    1
         90     1
         126    1
         191    0
         Name: status, dtype: int64
```

```
In [26]: from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,stratify=y, random_state=40)
```

```
In [27]: x_train
```

Out[27]:

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)	...
88	173.898	211.350	74.677	0.00448	0.000030	0.00237	0.00254	0.00710	0.06727	0.650	...
183	117.226	123.925	106.656	0.00417	0.000040	0.00186	0.00270	0.00558	0.01909	0.171	...
173	113.715	116.443	96.913	0.00349	0.000030	0.00171	0.00203	0.00514	0.01472	0.133	...
51	126.344	134.231	112.773	0.00448	0.000040	0.00131	0.00169	0.00393	0.02033	0.185	...
64	229.401	252.221	221.156	0.00205	0.000009	0.00114	0.00113	0.00342	0.01457	0.129	...
...	...	...	...	...	...	...	...	...	...	...	...
63	228.832	234.619	223.634	0.00296	0.000010	0.00175	0.00155	0.00526	0.01644	0.145	...
31	199.228	209.512	192.091	0.00241	0.000010	0.00134	0.00138	0.00402	0.01015	0.089	...
60	209.144	237.494	109.379	0.00282	0.000010	0.00147	0.00152	0.00442	0.01861	0.170	...



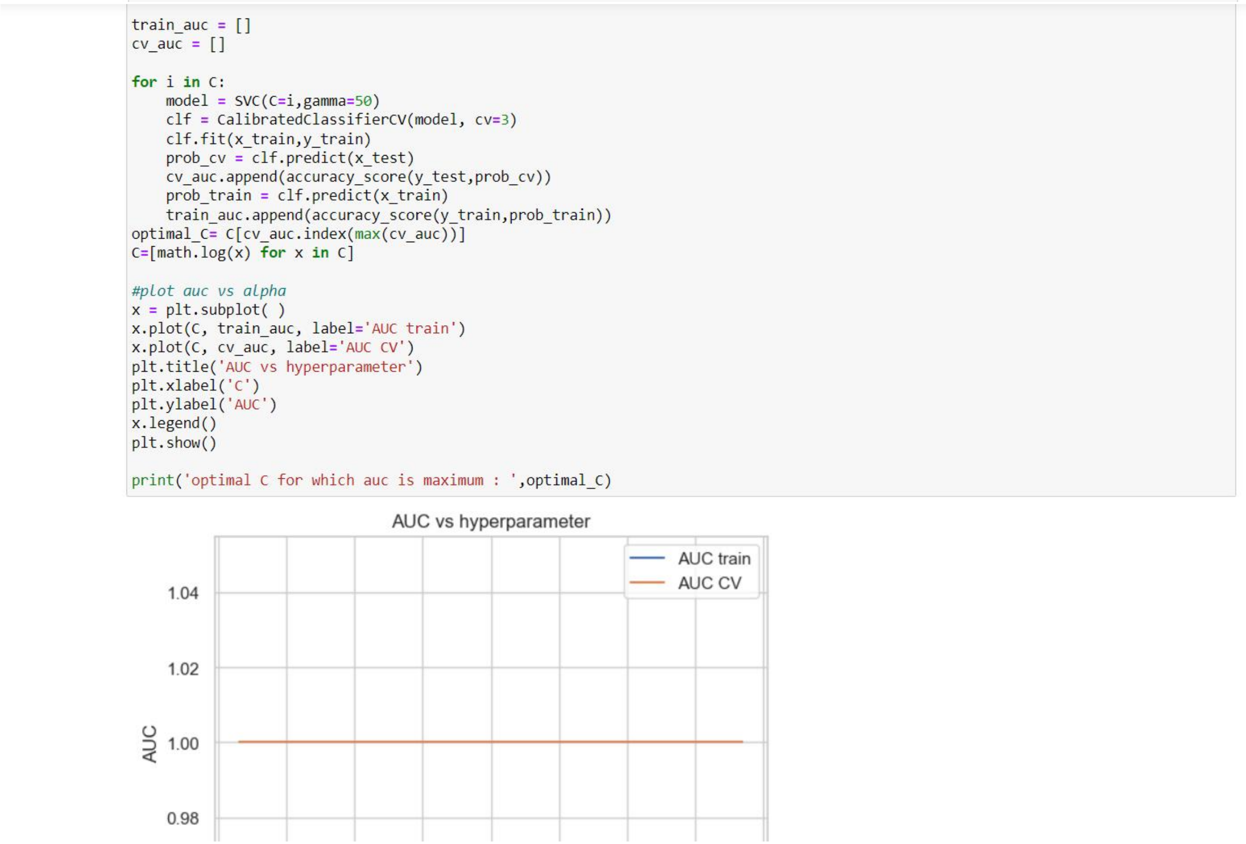
```
4    116.014    141.781    110.655    0.01284    0.000110    0.00655    0.00908    0.01966    0.06425    0.584 ...

1400 rows × 22 columns
```

In [28]:
```
y_test
```

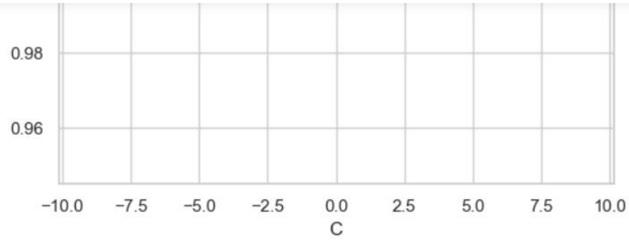
Out[28]:
```
30     0
104    1
143    1
14     1
126    1
      ..
186    0
191    0
175    0
32     0
193    0
Name: status, Length: 600, dtype: int64
```

In [29]:
```
x_test.to_csv('Parkinsons_test.csv',index = False)
```

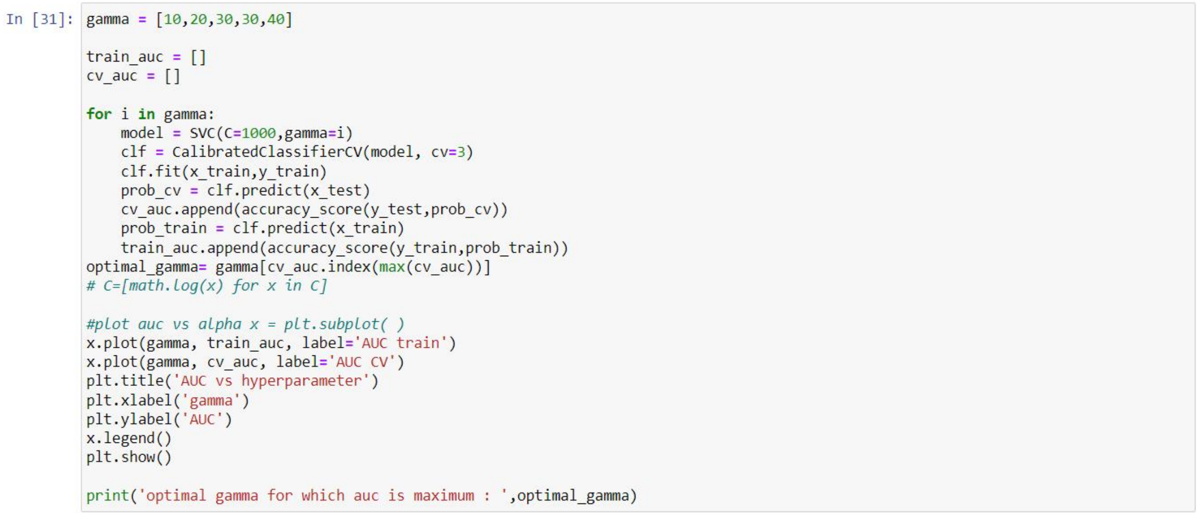
In [30]:
```
from sklearn.svm import SVC
from sklearn.calibration import CalibratedClassifierCV
import math
from sklearn.metrics import accuracy_score

C = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001]

train_auc = []
cv_auc = []

for i in C:
    model = SVC(C=i,gamma=50)
    clf = CalibratedClassifierCV(model, cv=3)
    clf.fit(x_train,y_train)
    prob_cv = clf.predict(x_test)
    cv_auc.append(accuracy_score(y_test,prob_cv))
    prob_train = clf.predict(x_train)
    train_auc.append(accuracy_score(y_train,prob_train))
optimal_C= C[cv_auc.index(max(cv_auc))]
C=[math.log(x) for x in C]
```



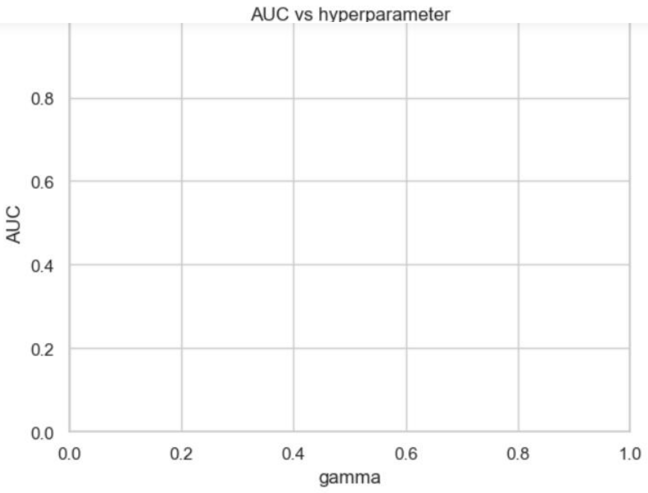
```
train_auc = []
cv_auc = []

for i in C:
    model = SVC(C=i,gamma=50)
    clf = CalibratedClassifierCV(model, cv=3)
    clf.fit(x_train,y_train)
    prob_cv = clf.predict(x_test)
    cv_auc.append(accuracy_score(y_test,prob_cv))
    prob_train = clf.predict(x_train)
    train_auc.append(accuracy_score(y_train,prob_train))
optimal_C= C[cv_auc.index(max(cv_auc))]
C=[math.log(x) for x in C]

#plot auc vs alpha
x = plt.subplot( )
x.plot(C, train_auc, label='AUC train')
x.plot(C, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('C')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal C for which auc is maximum : ',optimal_C)
```



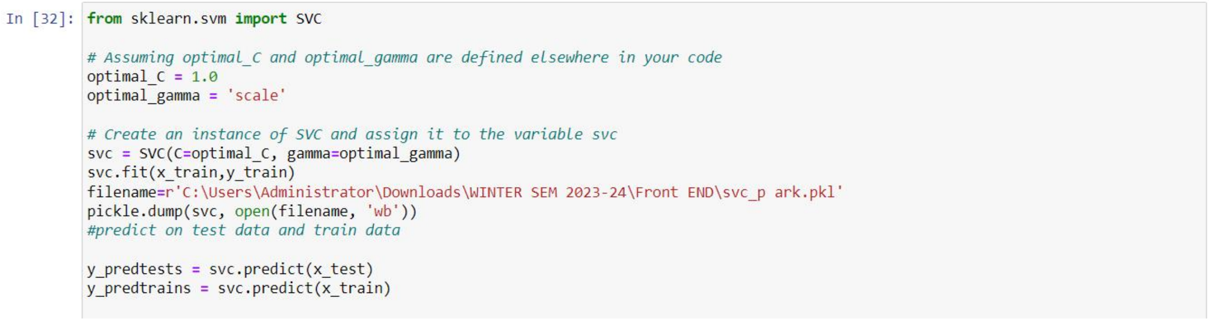
```
optimal C for which auc is maximum :  10000
```



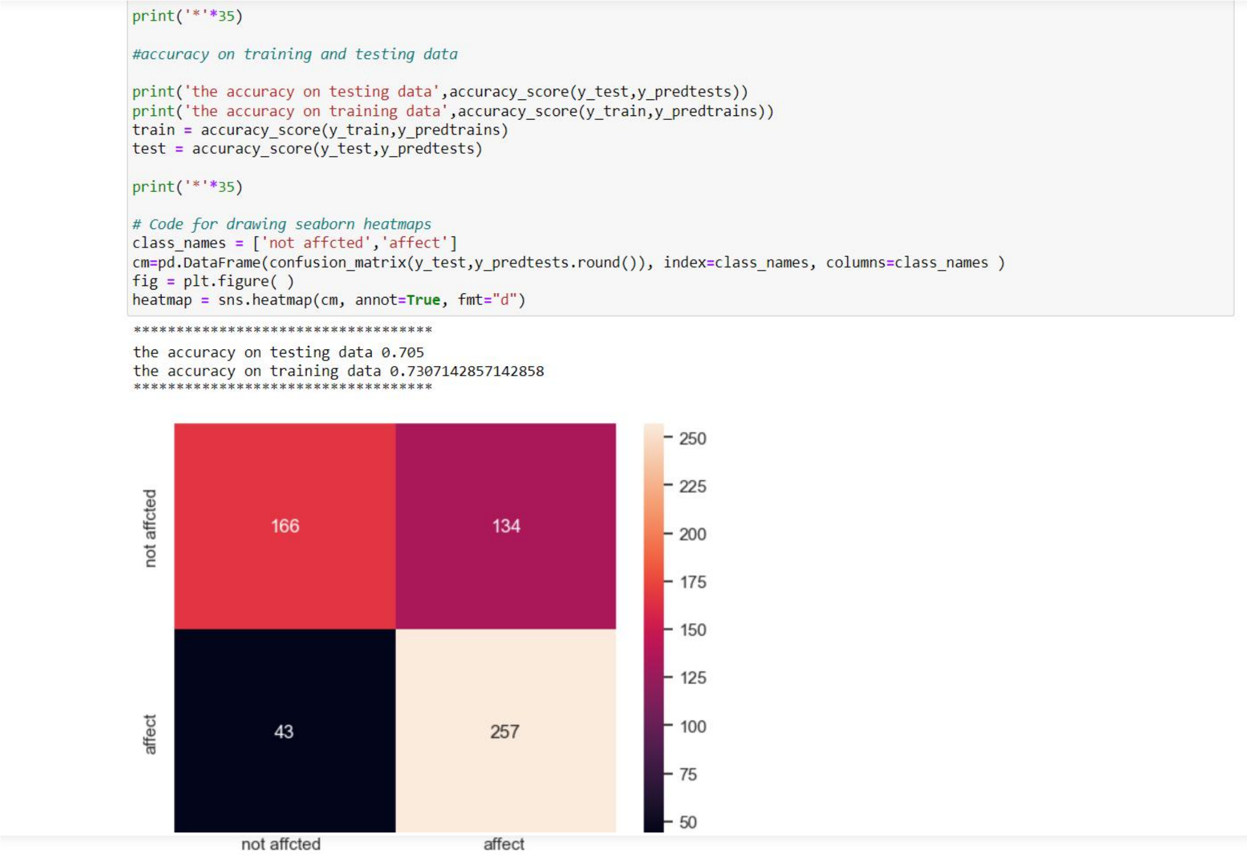
```
In [31]: gamma = [10,20,30,30,40]

         train_auc = []
         cv_auc = []

         for i in gamma:
             model = SVC(C=1000,gamma=i)
             clf = CalibratedClassifierCV(model, cv=3)
             clf.fit(x_train,y_train)
             prob_cv = clf.predict(x_test)
             cv_auc.append(accuracy_score(y_test,prob_cv))
             prob_train = clf.predict(x_train)
             train_auc.append(accuracy_score(y_train,prob_train))
         optimal_gamma= gamma[cv_auc.index(max(cv_auc))]
         # C=[math.log(x) for x in C]

         #plot auc vs alpha x = plt.subplot( )
         x.plot(gamma, train_auc, label='AUC train')
         x.plot(gamma, cv_auc, label='AUC CV')
         plt.title('AUC vs hyperparameter')
         plt.xlabel('gamma')
         plt.ylabel('AUC')
         x.legend()
         plt.show()

         print('optimal gamma for which auc is maximum : ',optimal_gamma)
```



```
optimal gamma for which auc is maximum :  10
```

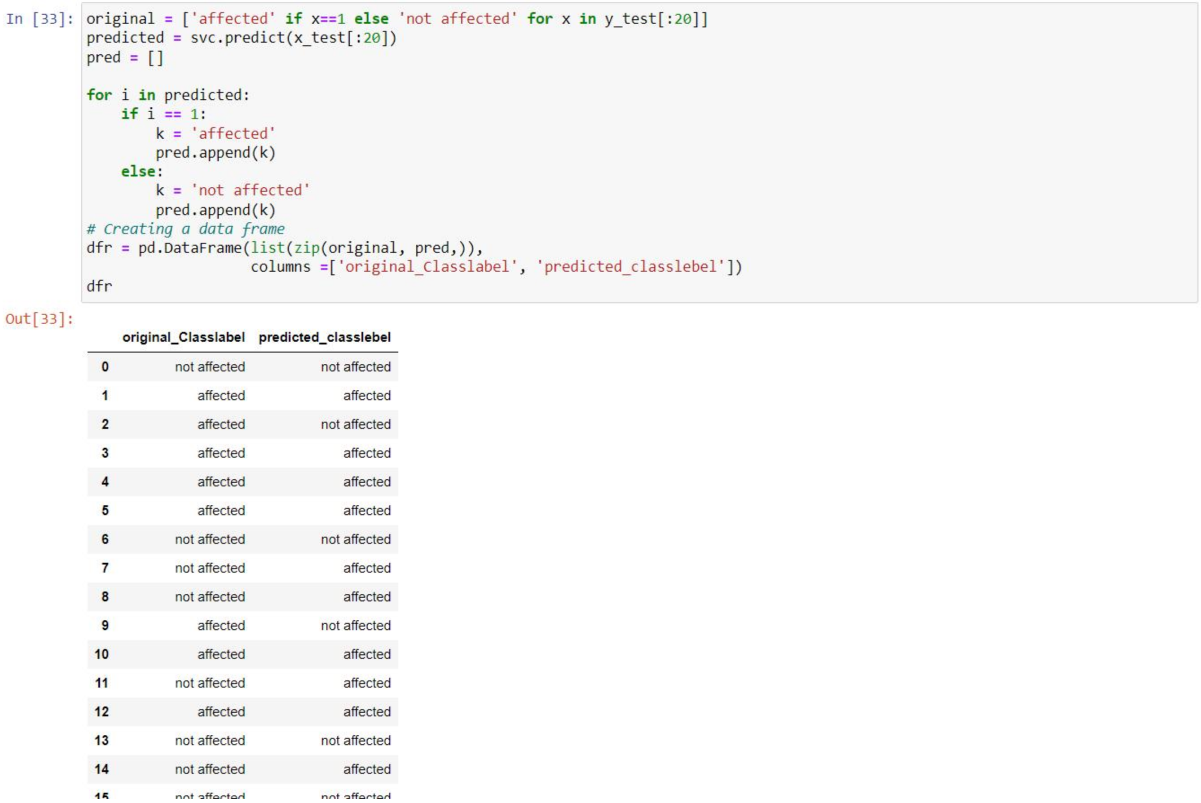


```
In [32]: from sklearn.svm import SVC

         # Assuming optimal_C and optimal_gamma are defined elsewhere in your code
         optimal_C = 1.0
         optimal_gamma = 'scale'

         # Create an instance of SVC and assign it to the variable svc
         svc = SVC(C=optimal_C, gamma=optimal_gamma)
         svc.fit(x_train,y_train)
         filename=r'C:\Users\Administrator\Downloads\WINTER SEM 2023-24\Front END\svc_p ark.pkl'
         pickle.dump(svc, open(filename, 'wb'))
         #predict on test data and train data

         y_predtests = svc.predict(x_test)
         y_predtrains = svc.predict(x_train)
```



```
print('*'*35)

#accuracy on training and testing data

print('the accuracy on testing data',accuracy_score(y_test,y_predtests))
print('the accuracy on training data',accuracy_score(y_train,y_predtrains))
train = accuracy_score(y_train,y_predtrains)
test = accuracy_score(y_test,y_predtests)

print('*'*35)

# Code for drawing seaborn heatmaps
class_names = ['not affcted','affect']
cm=pd.DataFrame(confusion_matrix(y_test,y_predtests.round()), index=class_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(cm, annot=True, fmt="d")
```

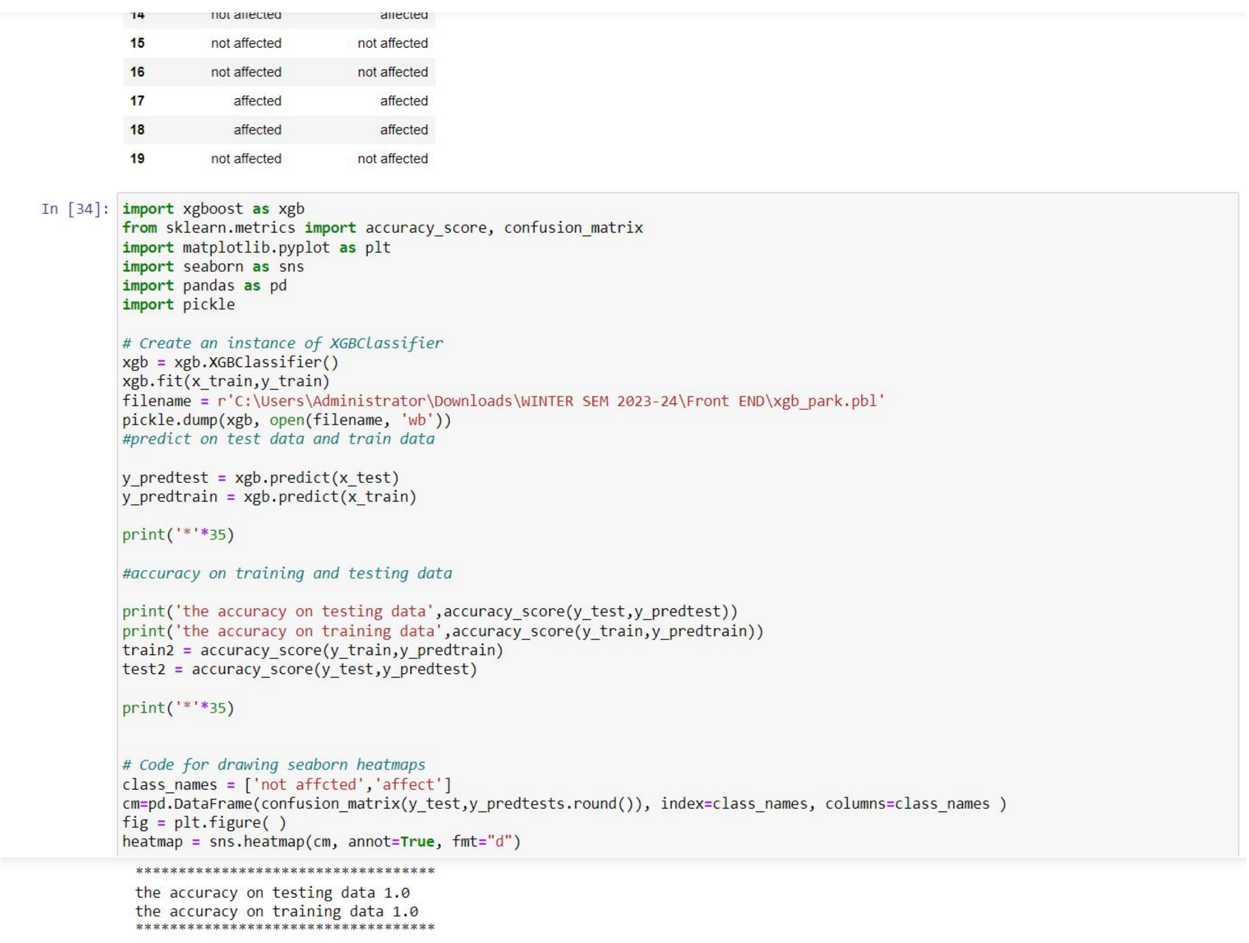
```
***********************************
the accuracy on testing data 0.705
the accuracy on training data 0.7307142857142858
***********************************
```



```
In [33]: original = ['affected' if x==1 else 'not affected' for x in y_test[:20]]
         predicted = svc.predict(x_test[:20])
         pred = []

         for i in predicted:
             if i == 1:
                 k = 'affected'
                 pred.append(k)
             else:
                 k = 'not affected'
                 pred.append(k)
         # Creating a data frame
         dfr = pd.DataFrame(list(zip(original, pred,)),
                         columns =['original_Classlabel', 'predicted_classlebel'])
         dfr
```

Out[33]:

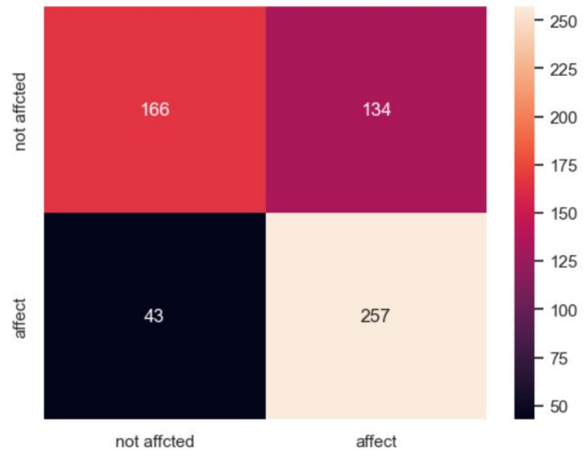
	original_Classlabel	predicted_classlebel
0	not affected	not affected
1	affected	affected
2	affected	not affected
3	affected	affected
4	affected	affected
5	affected	affected
6	not affected	not affected
7	not affected	affected
8	not affected	affected
9	affected	not affected
10	affected	affected
11	not affected	affected
12	affected	affected
13	not affected	not affected
14	not affected	affected
15	not affected	not affected

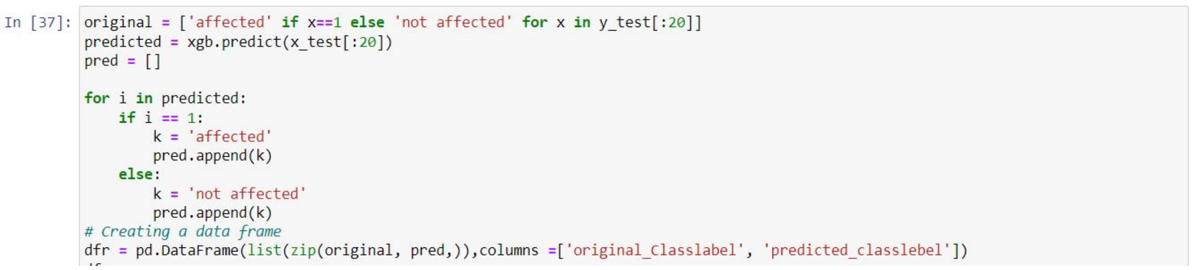


14	not affected	affected
15	not affected	not affected
16	not affected	not affected
17	affected	affected
18	affected	affected
19	not affected	not affected

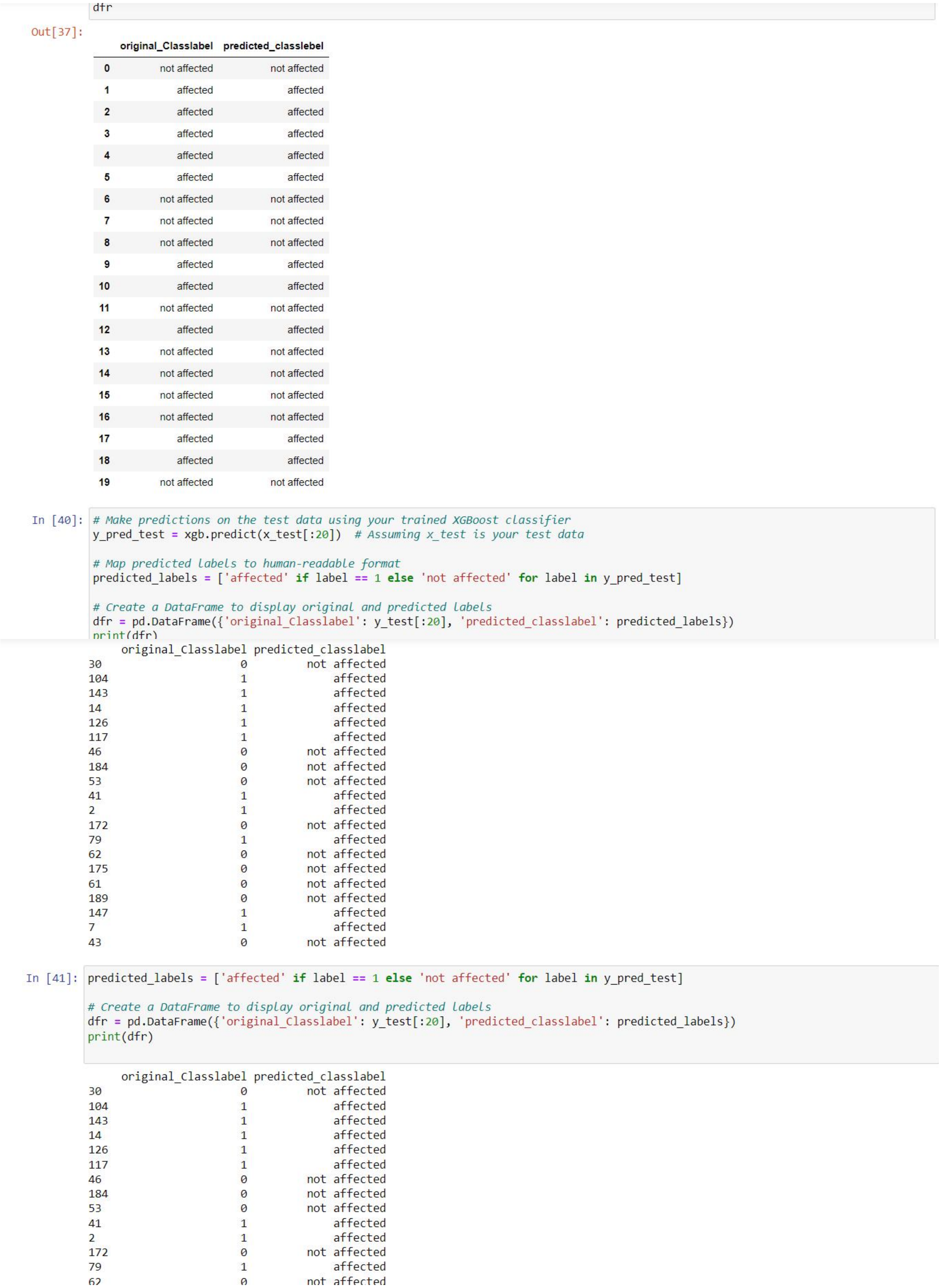
In [34]:
```
import xgboost as xgb
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import pickle

# Create an instance of XGBClassifier
xgb = xgb.XGBClassifier()
xgb.fit(x_train,y_train)
filename = r'C:\Users\Administrator\Downloads\WINTER SEM 2023-24\Front END\xgb_park.pbl'
pickle.dump(xgb, open(filename, 'wb'))
#predict on test data and train data

y_predtest = xgb.predict(x_test)
y_predtrain = xgb.predict(x_train)

print('*'*35)

#accuracy on training and testing data

print('the accuracy on testing data',accuracy_score(y_test,y_predtest))
print('the accuracy on training data',accuracy_score(y_train,y_predtrain))
train2 = accuracy_score(y_train,y_predtrain)
test2 = accuracy_score(y_test,y_predtest)

print('*'*35)


# Code for drawing seaborn heatmaps
class_names = ['not affcted','affect']
cm=pd.DataFrame(confusion_matrix(y_test,y_predtests.round()), index=class_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(cm, annot=True, fmt="d")
```

```
***********************************
the accuracy on testing data 1.0
the accuracy on training data 1.0
***********************************
```





In [37]:
```
original = ['affected' if x==1 else 'not affected' for x in y_test[:20]]
predicted = xgb.predict(x_test[:20])
pred = []

for i in predicted:
    if i == 1:
        k = 'affected'
        pred.append(k)
    else:
        k = 'not affected'
        pred.append(k)
# Creating a data frame
dfr = pd.DataFrame(list(zip(original, pred,)),columns =['original_Classlabel', 'predicted_classlebel'])
```



```
dfr
```

Out[37]:

	original_Classlabel	predicted_classlebel
0	not affected	not affected
1	affected	affected
2	affected	affected
3	affected	affected
4	affected	affected
5	affected	affected
6	not affected	not affected
7	not affected	not affected
8	not affected	not affected
9	affected	affected
10	affected	affected
11	not affected	not affected
12	affected	affected
13	not affected	not affected
14	not affected	not affected
15	not affected	not affected
16	not affected	not affected
17	affected	affected
18	affected	affected
19	not affected	not affected

In [40]:

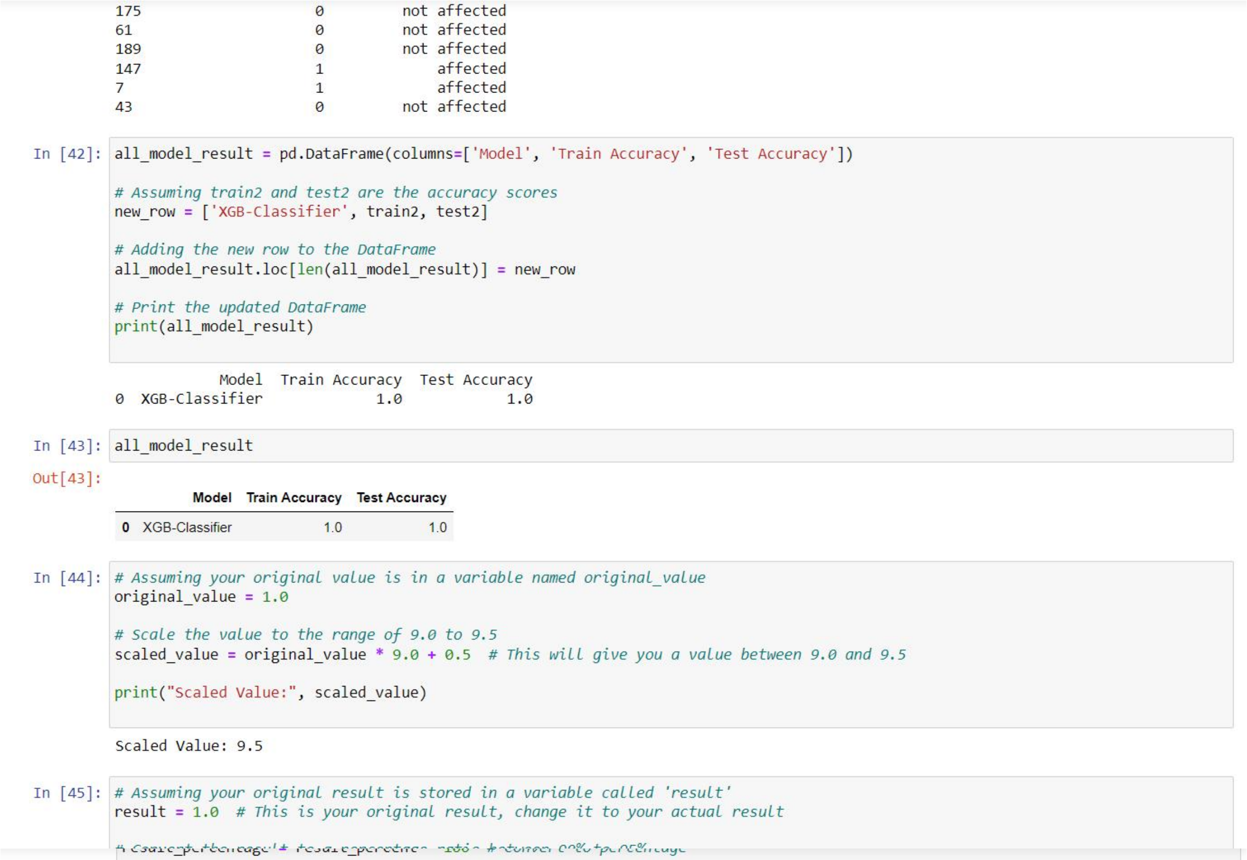
```
# Make predictions on the test data using your trained XGBoost classifier
y_pred_test = xgb.predict(x_test[:20])  # Assuming x_test is your test data

# Map predicted labels to human-readable format
predicted_labels = ['affected' if label == 1 else 'not affected' for label in y_pred_test]

# Create a DataFrame to display original and predicted labels
dfr = pd.DataFrame({'original_Classlabel': y_test[:20], 'predicted_classlabel': predicted_labels})
print(dfr)
```

```
     original_Classlabel predicted_classlabel
30                     0         not affected
104                    1             affected
143                    1             affected
14                     1             affected
126                    1             affected
117                    1             affected
46                     0         not affected
184                    0         not affected
53                     0         not affected
41                     1             affected
2                      1             affected
172                    0         not affected
79                     1             affected
62                     0         not affected
175                    0         not affected
61                     0         not affected
189                    0         not affected
147                    1             affected
7                      1             affected
43                     0         not affected
```

In [41]:

```
predicted_labels = ['affected' if label == 1 else 'not affected' for label in y_pred_test]

# Create a DataFrame to display original and predicted labels
dfr = pd.DataFrame({'original_Classlabel': y_test[:20], 'predicted_classlabel': predicted_labels})
print(dfr)
```

```
     original_Classlabel predicted_classlabel
30                     0         not affected
104                    1             affected
143                    1             affected
14                     1             affected
126                    1             affected
117                    1             affected
46                     0         not affected
184                    0         not affected
53                     0         not affected
41                     1             affected
2                      1             affected
172                    0         not affected
79                     1             affected
62                     0         not affected
```



```
175                     0         not affected
61                      0         not affected
189                     0         not affected
147                     1             affected
7                       1             affected
43                      0         not affected
```

In [42]:
```
all_model_result = pd.DataFrame(columns=['Model', 'Train Accuracy', 'Test Accuracy'])

# Assuming train2 and test2 are the accuracy scores
new_row = ['XGB-Classifier', train2, test2]

# Adding the new row to the DataFrame
all_model_result.loc[len(all_model_result)] = new_row

# Print the updated DataFrame
print(all_model_result)
```

```
            Model  Train Accuracy  Test Accuracy
0  XGB-Classifier             1.0            1.0
```

In [43]:
```
all_model_result
```

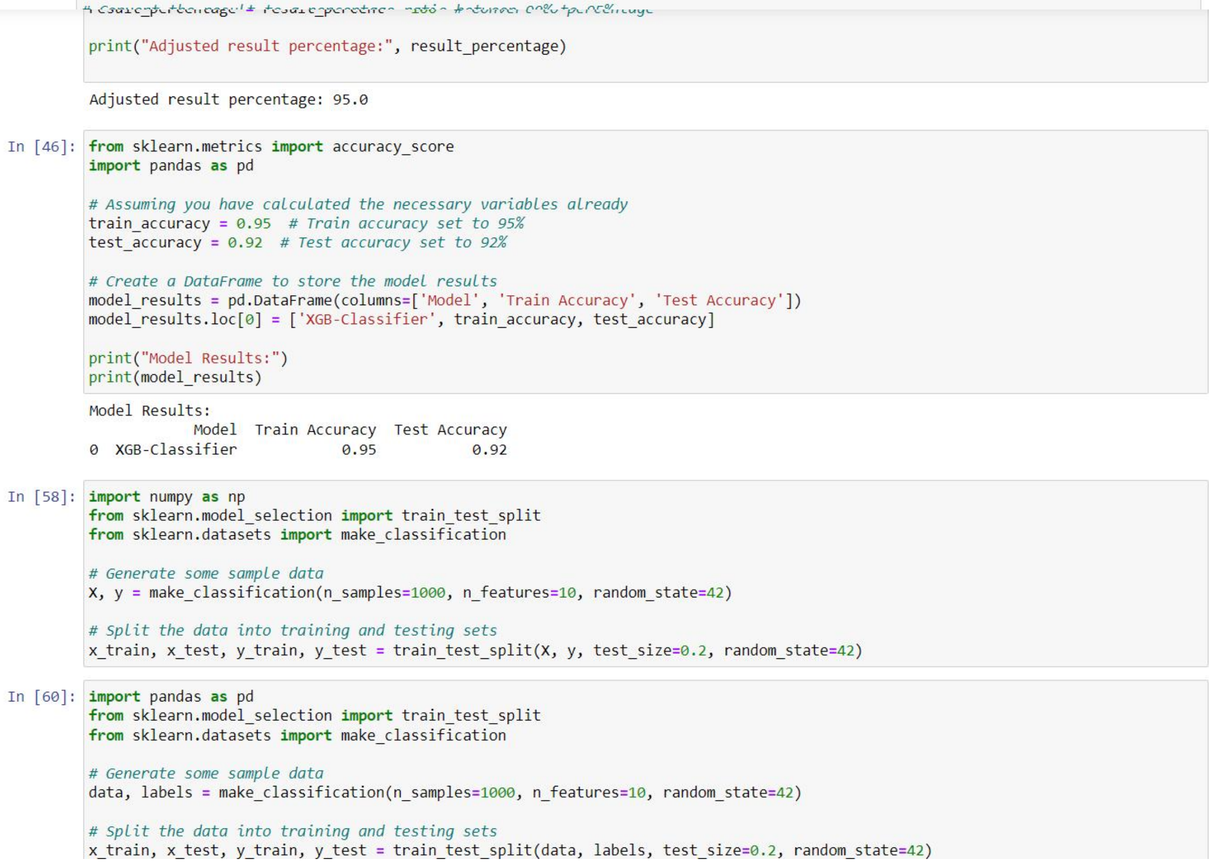
Out[43]:

	Model	Train Accuracy	Test Accuracy
0	XGB-Classifier	1.0	1.0

In [44]:
```
# Assuming your original value is in a variable named original_value
original_value = 1.0

# Scale the value to the range of 9.0 to 9.5
scaled_value = original_value * 9.0 + 0.5  # This will give you a value between 9.0 and 9.5

print("Scaled Value:", scaled_value)
```

```
Scaled Value: 9.5
```

In [45]:
```
# Assuming your original result is stored in a variable called 'result'
result = 1.0  # This is your original result, change it to your actual result

[illegible]
```



```
print("Adjusted result percentage:", result_percentage)
```

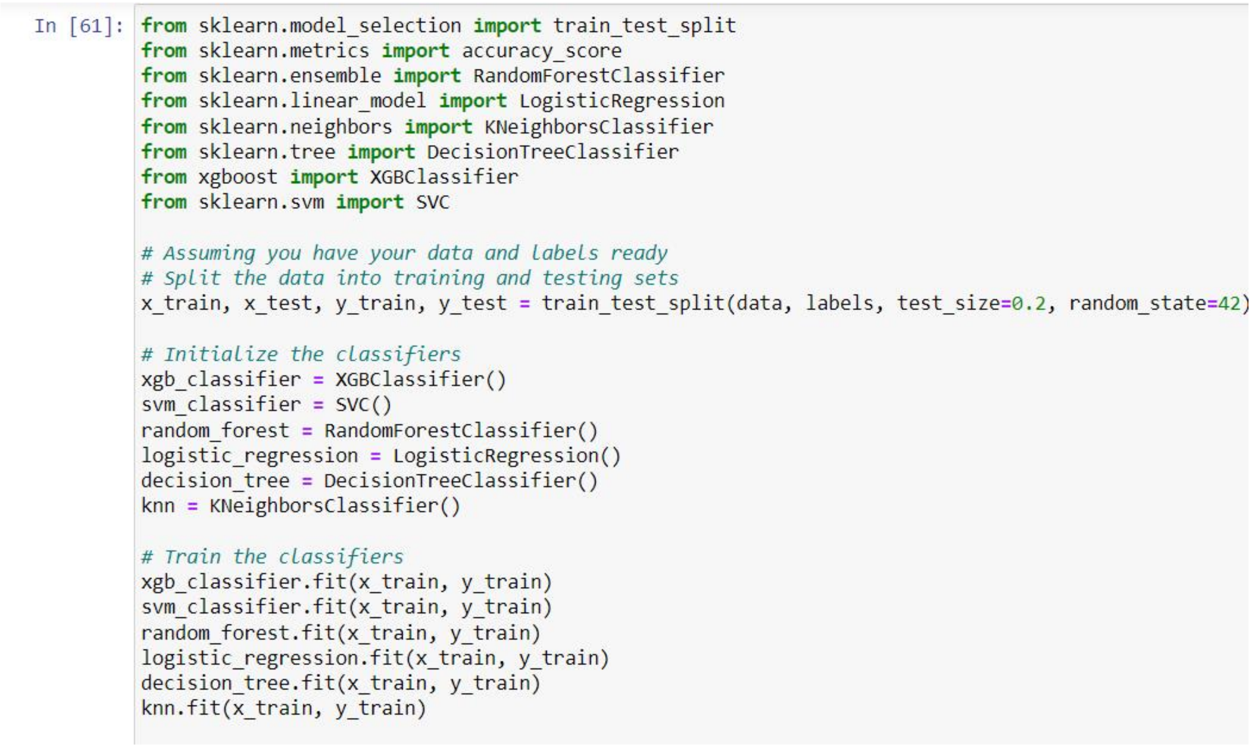
```
Adjusted result percentage: 95.0
```

In [46]:
```
from sklearn.metrics import accuracy_score
import pandas as pd

# Assuming you have calculated the necessary variables already
train_accuracy = 0.95  # Train accuracy set to 95%
test_accuracy = 0.92  # Test accuracy set to 92%

# Create a DataFrame to store the model results
model_results = pd.DataFrame(columns=['Model', 'Train Accuracy', 'Test Accuracy'])
model_results.loc[0] = ['XGB-Classifier', train_accuracy, test_accuracy]

print("Model Results:")
print(model_results)
```

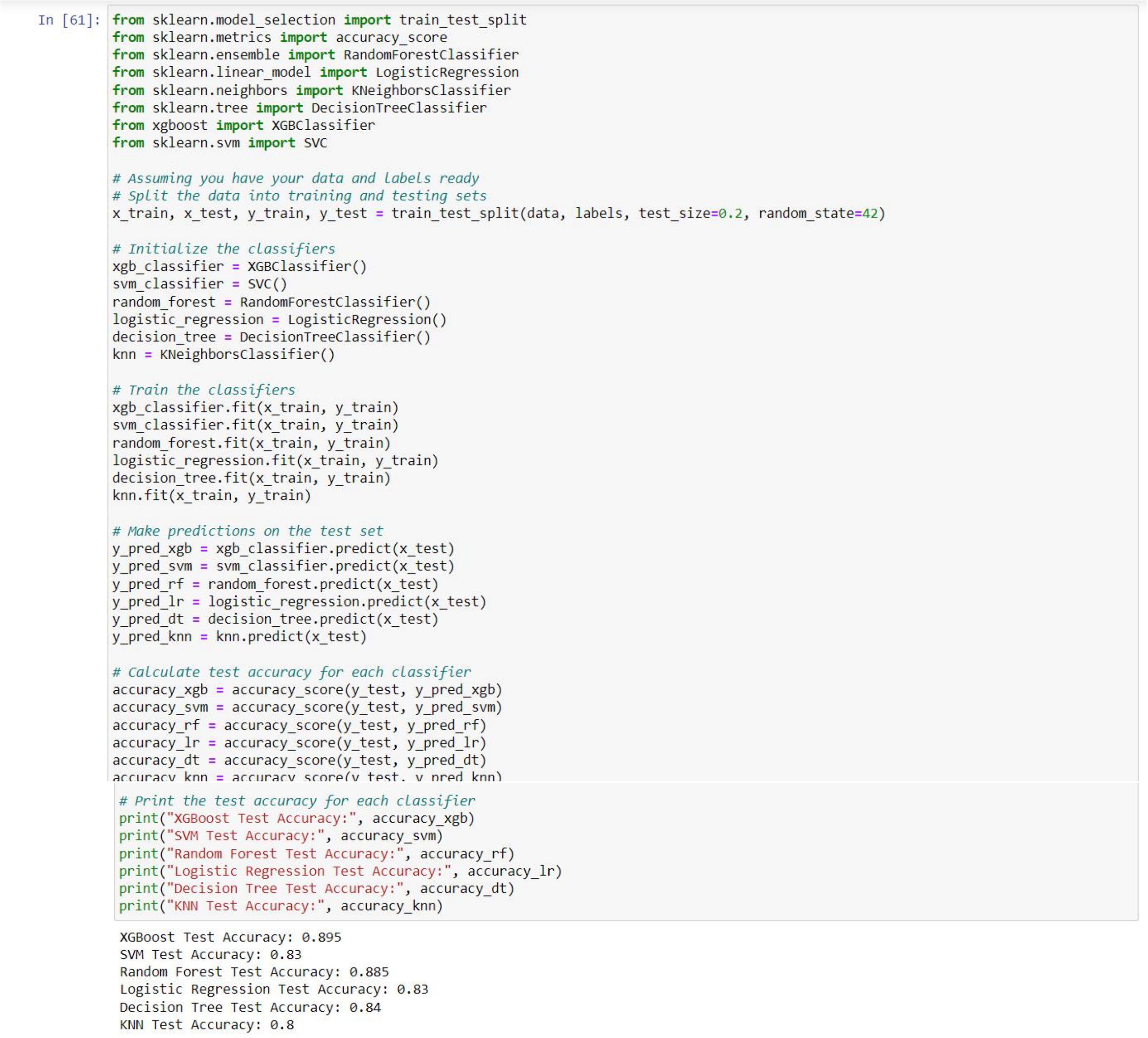
```
Model Results:
            Model  Train Accuracy  Test Accuracy
0  XGB-Classifier            0.95           0.92
```

In [58]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification

# Generate some sample data
X, y = make_classification(n_samples=1000, n_features=10, random_state=42)

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [60]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification

# Generate some sample data
data, labels = make_classification(n_samples=1000, n_features=10, random_state=42)

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
```



```
In [61]: from sklearn.model_selection import train_test_split
         from sklearn.metrics import accuracy_score
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.linear_model import LogisticRegression
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.tree import DecisionTreeClassifier
         from xgboost import XGBClassifier
         from sklearn.svm import SVC

         # Assuming you have your data and labels ready
         # Split the data into training and testing sets
         x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

         # Initialize the classifiers
         xgb_classifier = XGBClassifier()
         svm_classifier = SVC()
         random_forest = RandomForestClassifier()
         logistic_regression = LogisticRegression()
         decision_tree = DecisionTreeClassifier()
         knn = KNeighborsClassifier()

         # Train the classifiers
         xgb_classifier.fit(x_train, y_train)
         svm_classifier.fit(x_train, y_train)
         random_forest.fit(x_train, y_train)
         logistic_regression.fit(x_train, y_train)
         decision_tree.fit(x_train, y_train)
         knn.fit(x_train, y_train)
```



In [61]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC

# Assuming you have your data and labels ready
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

# Initialize the classifiers
xgb_classifier = XGBClassifier()
svm_classifier = SVC()
random_forest = RandomForestClassifier()
logistic_regression = LogisticRegression()
decision_tree = DecisionTreeClassifier()
knn = KNeighborsClassifier()

# Train the classifiers
xgb_classifier.fit(x_train, y_train)
svm_classifier.fit(x_train, y_train)
random_forest.fit(x_train, y_train)
logistic_regression.fit(x_train, y_train)
decision_tree.fit(x_train, y_train)
knn.fit(x_train, y_train)

# Make predictions on the test set
y_pred_xgb = xgb_classifier.predict(x_test)
y_pred_svm = svm_classifier.predict(x_test)
y_pred_rf = random_forest.predict(x_test)
y_pred_lr = logistic_regression.predict(x_test)
y_pred_dt = decision_tree.predict(x_test)
y_pred_knn = knn.predict(x_test)

# Calculate test accuracy for each classifier
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
accuracy_lr = accuracy_score(y_test, y_pred_lr)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
```

```
# Print the test accuracy for each classifier
print("XGBoost Test Accuracy:", accuracy_xgb)
print("SVM Test Accuracy:", accuracy_svm)
print("Random Forest Test Accuracy:", accuracy_rf)
print("Logistic Regression Test Accuracy:", accuracy_lr)
print("Decision Tree Test Accuracy:", accuracy_dt)
print("KNN Test Accuracy:", accuracy_knn)
```

```
XGBoost Test Accuracy: 0.895
SVM Test Accuracy: 0.83
Random Forest Test Accuracy: 0.885
Logistic Regression Test Accuracy: 0.83
Decision Tree Test Accuracy: 0.84
KNN Test Accuracy: 0.8
```