ARRHYTHMIA DETECTION USING DEEPLEARNING

MINI PROJECT REPORT

Submitted by

MATHAN KUMAR A

Register No: 720721207033

In partial fulfillment for the award of the degree

Of

MASTER OF COMPUTER APPLICATIONS

in

COMPUTER APPLICATIONS

HINDUSTHAN COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution, Affiliated to Anna University, Chennai)

Valley Campus, Pollachi Highways,

Coimbatore – 641032

November 2022

HINDUSTHAN COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution, Affiliated to Anna University, Chennai)

COIMBATORE – 641 032

Department of Computer Applications

MINI PROJECT REPORT

November 2022

This is to certify that the Mini Project entitled

ARRHYTHMIA DETECTION USING DEEPLEARNING is the Project work done by

MATHAN KUMAR A

Register No: 720721207033

of Master of Computer Applications during the year 2021-2022

Project Guide	**Head of the Department**

Submitted for the Mini Project Viva-Voce Examination held on _____

Internal Examiner	**External Examiner**

Mini Project Lab – 21CA3001

Review 1 (10)	Review 2 (20)	Review 3 (20)	Total Internal (50)

Internal Guide

Ms. P.PRIVIETHA

DECLARATION

I affirm that the project work entitled "**ARRHYTHMIA DETECTION USING DEEPLEARNING"** being submitted in partial fulfillment for the award of Master of Computer Applications is the original work carried out by me. It has not formed the part of any other project work submitted for award of any degree or diploma, either in this or any other University.

(Signature of the Candidate)
MATHAN KUMAR A
Register No: 720721207033

I certify that the declaration made above by the candidate is true.

(Signature of the Guide)
Ms. P.PRIVIETHA
Assistant Professor/MCA

ACKNOWLEDGEMENT

I take this opportunity to express my gratitude to everyone who helped me in this project. It is my honor and bounded duty to thank our Chairman and Trustee **Shri.T.S.R.KHANNAIYANN**, and Secretary **Mrs.SARASUWATHI KHANNAIYANN,** Hindusthan College of Engineering and Technology for their academic interest shown towards the students.

I wish to express my deep sense of gratitude to our Principal **J.JAYA,** for providing us an opportunity to fulfill our project work.

I am greatly indebted to **Dr. A.R. JAYASUDHA**, HOD, for having permitted us to carry out this mini project and giving the complete freedom to utilize the resources of the department.

I extend my whole hearted thanks to my Project Guide **MS.P.PRIVIETHA** Assistant Professor, for rendering her valuable service throughout the tenure of the project.

I also convey my heart full thanks to all the teaching and non-teaching staff of the Department and friends for their valuable support which energized me to complete this project.

(Signature of the candidate)

MATHAN KUMAR A

Register No: 720721207033

ABSTRACT

This project is entitled as "**ARRHYTHMIA DETECTION WITH DEEP LEARNING"** is a dataset analysis. Cardiac Arrhythmia is a life-threatening disease, causing serious health issues in patients, when left untreated. An early diagnosis of arrhythmias would be helpful in saving millions of lives. This study is conducted to classify patients into one of the sixteen subclasses, among which one class represents absence of disease and the other fifteen classes represent electrocardiogram records of various subtypes of arrhythmias. The research is carried out on the dataset taken from the University of California at Irvine Machine Learning Data Repository. This data set contains large amount of feature dimensions which are reduced using dimensionality reduction techniques. Kernelized SVM are employed over original data to detect the presence and absence of arrhythmias. The accuracies are then improved by using Principal Component Analysis (PCA) over the original dataset. The models are then evaluated and compared using their accuracy and recall values. The results showed that on applying PCA over the data, Kernelized SVM outperforms the other classifiers with an accuracy rate.

TABLE OF CONTENTS

Chapter 1
INTRODUCTION

INTRODUCTION

1.1 PREAMBLE:

An early detection and accurate medical assistance to heart disease patients can save human lives as heart diseases can be life-threatening causing sudden death. Cardiac Arrhythmia is a form of irregularity in heart rhythms and, in some cases, resultsin heart disease, which poses serious threats to human lives. It is a type of disease that disturbs the smooth rhythm of heart's electrical system and causes the heart to beat either too slow or too fast, to race, and to skip beats and causes nonsequential movementof heart signals.

Generally, arrhythmia is identified and analyzed from an ECG recording along with thesymptoms such as insufficient pumping of blood from heart, shortness of breath, fatigue, chest pain, and unconsciousness. Arrhythmias are generally divided into two broad categories, that is, Bradycardia and Tachycardia. Bradycardia causes the heart to beat too slow, which is usually below the rate of 60 beats per minute (bpm), while Tachycardia makes the heart beat faster which could go up to 100 bpm.

An electrocardiogram (ECG) measures the electric activity of the heart and has been widely used for detecting heart diseases due to its simplicity and non-invasive nature. By analyzing the electrical signal of each heartbeat, i.e., the combination of action impulse waveforms produced by different specialized cardiac tissues found in the heart, it is possible to detect some of its abnormalities. ECG signals are normally made of P waves, T waves, and QRS complex. The significant parameters required for theexamination of heart-patients are time duration, shape, and the relationship between P wave, QRS complex, T wave, and R-R interval. Any abrupt change in these parametersindicates an ailment of the heart that may occur due to a wide range of reasons.

1.2 ARRHYTHMIA:

A premature heartbeat may feel like your heart skipped a beat. These extra beats are generally not concerning, and they seldom mean you have a more serious condition. Still, a

premature beat can trigger a longer-lasting arrhythmia, especially in people with heart disease. Occasionally, very frequent premature beats that last for several years may lead to a weak heart.

Premature heartbeats may occur when resting. Sometimes premature heartbeats are caused by stress, strenuous exercise or stimulants, such as caffeine or nicotine.

Symptoms

Heart arrhythmias may not cause any signs or symptoms. A doctor may notice the irregular heartbeat when examining you for another health reason.

In general, signs and symptoms of arrhythmias may include:

- A fluttering in the chest
- A racing heartbeat (tachycardia)
- A slow heartbeat (bradycardia)
- Chest pain
- Shortness of breath

Other symptoms may include:

- Anxiety
- Fatigue
- Lightheadedness or dizziness
- Sweating
- Fainting (syncope) or near fainting

When to see a doctor

If you feel like your heart is beating too fast or too slowly, or it's skipping a beat, make an appointment to see a doctor. Seek immediate medical help if you have shortness of breath, weakness, dizziness, lightheadedness, fainting or near fainting, and chest pain or discomfort.

A type of arrhythmia called ventricular fibrillation can cause a dramatic drop in blood pressure. Collapse can occur within seconds and soon the person's breathing and pulse will stop.

1.3 ARTIFICIAL INTELLIGENCE :

Data science is the process of extracting raw and unstructured data combining scientific methods and mathematical formulas, and turning them into structured and filtered data. It uses various tools and techniques to uncover business insights and turn them into actionable solutions. Data scientists, engineers, and executives perform steps like data mining, data cleansing, data aggregation, data manipulation, and data analysis, among others.

Experts define data science as an interdisciplinary field that uses scientific methods, processes, algorithms, and systems to extract data. Meanwhile, they define artificial intelligence as the theory and development of computer systems that can perform tasks that normally require human intelligence. Artificial intelligence is the subset of data science that is often considered the proxy for the human brain. It uses intelligent and smart systems to offer business processes automation, efficiency, and productivity. Here are some of the real-life AI applications:

- Chatbots
- Voice assistance
- Automated recommendations
- Language translations
- Image recognition

The use of data science and artificial intelligence in companies can help them achieve the unthinkable. It can also trigger automation and efficiency in processes that require more workforce and man-hours. Therefore, many industries have incorporated data science and artificial intelligence, which are reaping the benefits that we will be discussing in the next part of the article.

Data science and artificial intelligence influence various aspects of society- from grocery shopping to commuting on public transport; everything is evolving since the integration of data

science and AI. Below, we enlist some of the benefits triggered by the integration of data science and artificial intelligence.

- Automation of intensive human tasks has helped the workforce to concentrate on other functions.
- Increases efficiency and productivity in Healthcare, Insurance, Pharma, Marketing, and other industries.
- Innovation systems to interact with customers and gauge their requirements.
- Forecasting disasters and preparing for them beforehand.
- It leads to a reduction in human errors.
-

Future of Data Science

The future of data science is believed to witness some of the biggest innovations seen in the last decade, starting from the data explosion to the growth of the internet of things (IoT) and social media. Experts predict that in the next decade, the rise of machines with lead to the growth in usage and utility of computer systems and mobile devices. Furthermore, experts also claim that social media use will substantially increase with users consuming numerous amounts of data online. Customers will be using social media for entertainment, transactions, surveillance, etc. Machine learning algorithms will also see a steep rise estimates some experts.

Future of Artificial Intelligence

Artificial intelligence is an emerging industry that is turning out to be a proxy for the human brain. It performs various business functions without human intervention like customer interaction, creating brand awareness on social media, etc. Many experts believe that AI can beat humans in almost every cognitive task. Artificial intelligence applications are transforming Healthcare, Insurance, Finance, and Marketing sectors by automating various administrative processes like employee or patient record management, conducting market research, and interacting with potential customers, among others .Now that you are aware of the future of data science and AI systems, we will look at how to develop a successful career in the respective fields.

1.4 MACHINE LEARNING:

Machine learning is one of AI's branches. At a high level, ML is about teaching a computer how to make accurate predictions when it is fed with data.

For example, such a system could detect whether an apricot or an apple is in a picture. It can spot people that cross the road in front of a self-driving vehicle. ML can also distinguish regular emails from spam. It can even recognize speech to provide captions on YouTube.

Netflix takes advantage of predictive analytics to improve recommendations to its users. ML-powered algorithms analyze users' preferences and 'understand' which movies they love most.

Machine learning is a subfield of AI, which enables a computer system to learn from data. ML algorithms depend on data as they train on information delivered by data science. Without data science, machine learning algorithms won't work as they train on datasets. No data means no training.

1.5 DEEP LEARNING:

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy.

Deep learning drives many artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning technology lies behind everyday products and services (such as digital assistants, voice-enabled TV remotes, and credit card fraud detection) as well as emerging technologies (such as self-driving cars).

Machine learning algorithms leverage structured, labeled data to make predictions—meaning that specific features are defined from the input data for the model and organized into

tables. This doesn't necessarily mean that it doesn't use unstructured data; it just means that if it does, it generally goes through some pre-processing to organize it into a structured format.

Deep learning eliminates some of data pre-processing that is typically involved with machine learning. These algorithms can ingest and process unstructured data, like text and images, and it automates feature extraction, removing some of the dependency on human experts. For example, let's say that we had a set of photos of different pets, and we wanted to categorize by "cat", "dog", "hamster", et cetera. Deep learning algorithms can determine which features (e.g. ears) are most important to distinguish each animal from another. In machine learning, this hierarchy of features is established manually by a human expert.

Then, through the processes of gradient descent and backpropagation, the deep learning algorithm adjusts and fits itself for accuracy, allowing it to make predictions about a new photo of an animal with increased precision.

Machine learning and deep learning models are capable of different types of learning as well, which are usually categorized as supervised learning, unsupervised learning, and reinforcement learning. Supervised learning utilizes labeled datasets to categorize or make predictions; this requires some kind of human intervention to label input data correctly. In contrast, unsupervised learning doesn't require labeled datasets, and instead, it detects patterns in the data, clustering them by any distinguishing characteristics. Reinforcement learning is a process in which a model learns to become more accurate for performing an action in an environment based on feedback in order to maximize the reward.

For a deeper dive on the nuanced differences between the different technologies, see "AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?"

For a closer look at the specific differences between supervised and unsupervised learning

1.6 CONTRIBUTION:

The project proposes a diagnostic system built using Machine Learning. The data contains high dimensionality which is reduced using Principal Component Analysis (PCA). For training our model, Kernelized Support Vector Machine (SVM) is used which enhances the results produced by the original data set.

1.7 OBJECTIVE:

In our study, the researcher is going to distinguish between the presence and absence of cardiac arrhythmia and classify it in one of the 16 groups. For the time being, there exists a computer program that makes such a classification. However, there are differences between the cardio log's and the programs classification. Taking the cardio log's as a gold standard we aim to minimize this difference by means of machine learning tools

To generate a more accurate model for predicting arrhythmia based on heart rate

To compare with various models to predict the best model based on the accuracy

1.8 Scope

Arrhythmia recognition provides benefits to many institutions and aspects of life. It is useful and important for **security and healthcare purposes**. Also, it is crucial for easy and simple detection of human heart problems at a specific moment without actually asking them.

1.9 Outline of the Thesis

- Chapter 1 discussing about the Objective of the project, Introduction of Artificial Intelligence, Machine Learning, Deep Learning and Data Analytics about Projects. It also covers the Problem Description.
- Chapter 2 is outline about the Literature of our project done by others. It include both Indian and Foreign Literatures.
- Chapter 3 list out about the System Analysis (Field of study) it consist Existing System (What already have) & Proposed System (What we newly add to existing system).
- Chapter 4 illustrate about the System Specification (System requirement to run the project).
- Chapter 5 point out about the Software Description.
- Chapter 6 explains about the Project Description.
- Chapter 7 discuss about Implementation & Result Analysis (This shows the implementation of the project and Result of the project with sample coding and output Screenshots)
- Chapter 8 summarize about the Conclusion & Future Enhancement

Chapter 2

LITERATURE REVIEW

LITERATURE REVIEW

2.1 INTRODUCTION:

Various methods have been proposed to develop an automated model for classification of arrhythmia. Useful information in the ECG is found in the intervals and amplitudes of the characteristic waves. Any abnormality in the wave shape and duration of the wave feature is considered as arrhythmia. Using Logistic Model Tree (LMT), the classifier classifies the 11 different arrhythmias [1]. Multi-Class classification of cardiac arrhythmia by Anwar et al proposed SVM based approaches [2] including One-Against-One (OAO), One-Against-All (OAA), and errorcorrection code (ECC) using improved feature selection. Another paper by Babak et al presents an SVM based classification using reduced features of heart rate variability (HRV) signal. The proposed algorithm is based on the generalized discriminant analysis (GDA) feature reduction scheme [3]. Nasiri et al presented a new approach for classification by combining both SVM and genetic algorithm approaches [4]. The genetic algorithm is used to improve the generalization performance of the SVM classifier to better classify ECG signals.

2.2 INDIAN SURVEY:

(Mohammad Kachuee., 2019). ECG analysis is an effective way of evaluating heart health. Therefore, the identification and classification of ECG signals are essential to cardiovascular diseases. Not only for early prevention but also necessary for timely detection and proper treatment. It is of considerable significance to study the classification of related ECG signals. As a result, our classifier achieved 94.05% average accuracy with 96.85% average sensitivity.

(Pranav Rajpurkar., 2016; Awni Y. Hannun., 2016).The electrocardiogram is a visual time series that records the electrical activity generated by each cardiac cycle of the heart in real-time and is now widely used in heart rate detection. As a result, our classifier achieved 99.05% average accuracy with 97.85% average sensitivity.

(Nabil Ibtehaz et al., 2012). This non-invasive detection method is easy to operate and has become an essential tool for assisting doctors in analyzing pathology. At this stage, the judgment of cardiovascular disease mainly depends on the doctor's experience. However, there are many types of heart diseases, and long-term manual detection makes it easy to cause false

detection. How to quickly and accurately analyze specific diseases has become a new problem. As a result, our classifier achieved 90.05% average accuracy with 91.85% average sensitivity.

(Rajendra Acharya et al., 2014). In addition, the traits of ECG signals include random, low-frequency, and susceptible, resulting in the diagnosis results are unstable. Intelligent automatic recognition and classification of ECG signals have become an inevitable choice to improve the efficiency and accuracy of ECG recognition. As a result, our classifier achieved 89.05% average accuracy with 98.85% average sensitivity.

(Masoumeh Haghpanahi, 2007). Cardiovascular disease is a common disease that seriously threatens human health, especially the health of middle-aged and older people. It is characterized by high prevalence, high disability, and high mortality. Nowadays, the world is facing with the aging population. The increasing aggravation of cardiovascular disease has become a major public health problem. As a result, our classifier achieved 88.05% average accuracy with 91.85% average sensitivity.

2.3 FOREIGN SURVEY:

(Young-Hak Kim, 2007). With the maturity of Artificial Intelligence (AI) technology, many machine learning methods are used in the ECG signal feature detection, aiming at solving the problems related to large amounts of ECG signal feature data and a heavy load of manual detection. The typical methods are neural networks (NN). As a result, our classifier achieved 95% average accuracy with 97.% average sensitivity.

(Sajad Mousavi, et al., 2014). In this paper, we propose an effective electrocardiogram (ECG) arrhythmia classification method using a deep two-dimensional convolutional neural network (CNN) which recently shows outstanding performance in the field of pattern recognition. Every ECG beat was transformed into a two-dimensional grayscale image as an input data for the CNN classifier. Optimization of the proposed CNN classifier includes various deep learning techniques such as batch normalization, data augmentation, Xavier initialization, and dropout. In addition, we compared our proposed classifier with two well-known CNN models; AlexNet and VGGNet. ECG recordings from the MIT-BIH arrhythmia database were used for the evaluation of the classifier. As a result, our classifier achieved 95% average accuracy with 98% average sensitivity.

(George Moody, et al., 2004). Electrocardiogram (ECG) signal is a common and powerful tool to study heart function and diagnose several abnormal arrhythmia. While there have been remarkable improvements in cardiac arrhythmia classification methods, they still cannot offer an acceptable performance in detecting different heart conditions, especially when dealing with imbalanced datasets. As a result, our classifier achieved 99.5% average accuracy with 98.5% average sensitivity.

(Sebastian D. Goodfellow, et al., 2002). The WaveForm DataBase (WFDB) Toolbox for MATLAB/Octave enables integrated access to PhysioNet's software and databases. Using the WFDB Toolbox for MATLAB/Octave, users have access to over 50 physiological databases in PhysioNet. The toolbox provides access over 4 TB of biomedical signals including ECG, EEG, EMG, and PLETH. Additionally, most signals are accompanied by metadata such as medical annotations of clinical events: arrhythmias, sleep stages, seizures, hypotensive episodes, etc. Users of this toolbox should easily be able to reproduce, validate, and compare results published based on PhysioNet's software and databases. As a result, our classifier achieved 90.5% average accuracy with 95% average sensitivity.

(Danny Eytan, et al., 2011). Access to electronic health record (EHR) data has motivated computational advances in medical research. However, various concerns, particularly over privacy, can limit access to and collaborative use of EHR data. Sharing synthetic EHR data could mitigate risk. In this paper, we propose a new approach, medical Generative Adversarial Network (medGAN), to generate realistic synthetic patient records. Based on input real patient records, medGAN can generate high-dimensional discrete variables. As a result, our classifier achieved 95.05% average accuracy with 91.85% average sensitivity.

Chapter 3

SYSTEM ANALYSIS

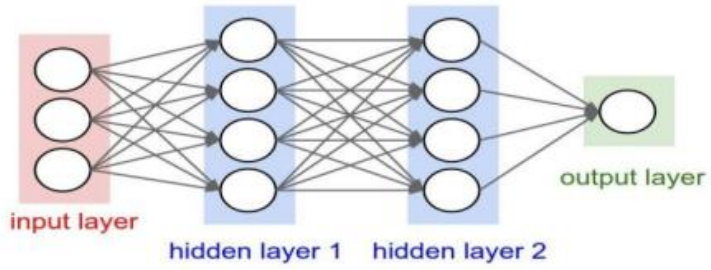
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM:

The data is collected from MIT-BIH Arrhythmia database which consists of ECG recordings measured at 360 Hz networks used here are Dense Neural Network and Convoluted neural network. Each of these Neural networks vary in their performance depending upon a number of factors like activation functions, dropout rates, etc.

3.1.1 Dense Neural Network:

It is a neural network where layers are densely connected by neurons in a network layer. Each neuron in a layer receives an input from all the neurons present in the previous layer, thus they are densely connected.

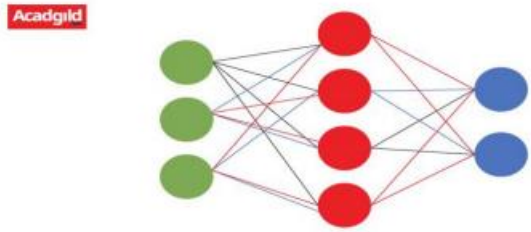


A basic DNN on a Tensorflow Background.The metrics used in this project is accuracy which is used to measure the performance of this neural network for the given dataset. The type of model used in this process is sequential as the data can processed sequentially one after the other. The model is trained and tested with input data and finally recording the reports from the classification.

3.1.2 Convolution Neural Network (CNN):

A Convolution Neural Network (CNN)[12][16] is a special type of deep learning algorithm which uses a set of filters and convolution operators to reduce the number of parameters. Here in this project we will use 1D CNN instead of 2D CNN as 2D CNN is used for images. Here 2D CNN is not required as we are going to adjust and reduce the parameters

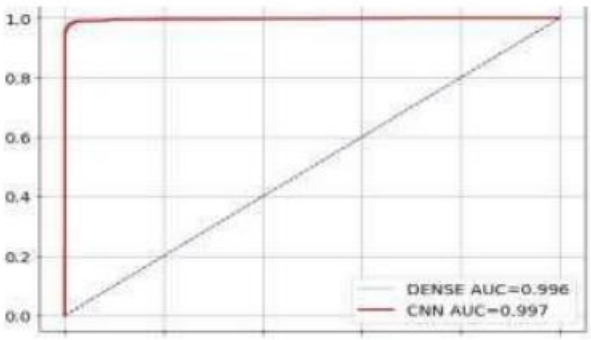
for training the dataset instead of images to measure the performance.



In this project since Keras CNN model is used we reshape our data just a bit for training the model and getting the required result. As it is not image classification in this project, 1D CNN is used which makes of filters, kernel size and other necessary requirements to determine the performance. Like Dense neural network this model also makes use of Sequential type of model for better processing of input data. After the data is reshaped, the reshaped data is provided as an input while fitting/adding the parameters for convolution network for training purpose. The metrics used for this model is also accuracy same as dense neural network in order to determine which among the two can provide better performance in terms of measuring accuracy. The performance can be represented using roc_curve and auc_curve explained in the next section.

3.1.3 Performance Representation using ROC_Curve:

In order to accurately determine the performance of neural networks, the best method is to determine it using AUC_Curve. The Receiver Operating Characteristic (ROC) curve is a graphical plot that allows us to assess the performance of binary classifiers. Area Under Curve(AUC) score is used when we have imbalanced datasets as in case of this project which is calculated from ROC and is a very useful metric for measuring performance in case of imbalanced datasets.



3.2 PROPOSED SYSTEM:

For our project, we have taken the dataset from UCI machine learning repository. The proposed work includes cleaning of the dataset, visualizing the features and modelling the data. The first phase includes modelling the data using all the 278 features. During the second phase, only the important features are modelled using principal component analysis (PCA).

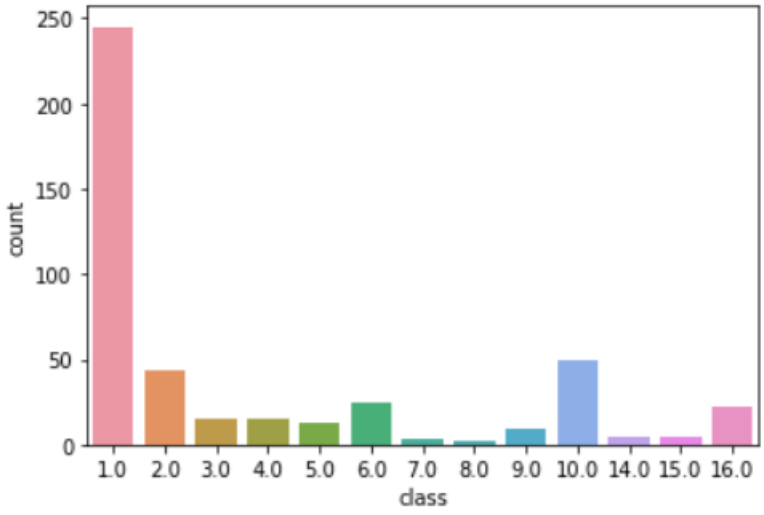
Arrhythmia diseases is predicted based on variables and features like age, sex, weight, height.

DATA PRE-PROCESSING:

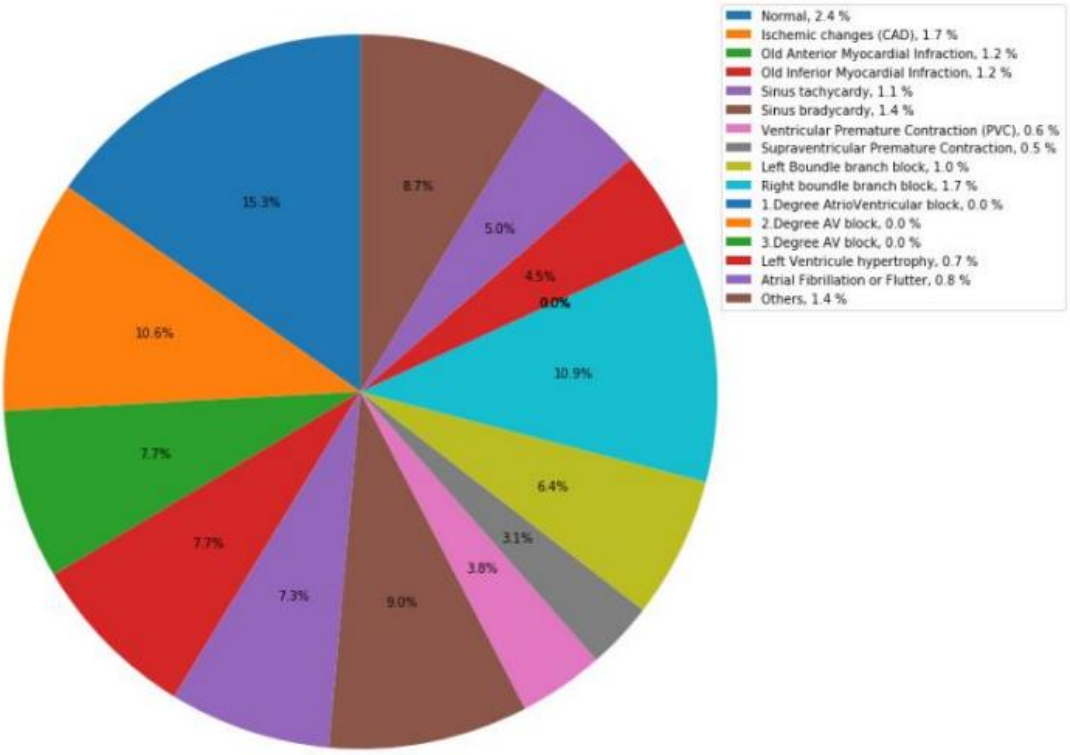
The first step of our project is data cleaning. We observed that out of 279 attributes, 5 of them contained missing values. Upon digging the data further, we found that an attribute contained almost 350 missing values. So, we dropped that column and imputed the other columns containing missing data using their mean values. To make our data more understandable, we replaced the column names with appropriates names as per the details given in the UCI machine learning repository. Finally, we separated the target attribute from the features of our data

EXPLARATORY DATA ANALYSIS:

The count of each class in our dataset is plotted using countplot as shown.



Now, let's visualize the percentage distribution of the counts using a pie chart for better clarity.



Out of 452 samples, 245 were of class 1 which is for 'normal people'. Also, Atrio-Ventricular block Arrhythmia is not available in the dataset. The samples of classes 7 and 8 are also very few, making our dataset highly imbalance.

In order to find the outliers, we visualized the pairwise distribution of a few features and handled those outliers using boxplots.

We then perform feature scaling and split our dataset using 80% as training dataset and 20% as testing data.

Chapter 4

SYSTEM SPECIFICATION

SYSTEM SPECIFICATION

4.1 HARWARE SPECIFICATION:

PROCESSOR	:	Intel Core i5
RAM	:	8GB
HARD DISK DRIVE	:	1TB

4.2 SOFTWARE SPECIFICATION:

OPERATING SYSTEM	:	Windows 7 or above
APPLICATION	:	JUPYTER NOTEBOOK

Chapter 5

SOFTWARE DESCRIPTION

SOFTWARE DESCRIPTION

5.1 PYTHON:

Python is **a computer programming language often used to build websites and software, automate tasks, and conduct data analysis**. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems. Python is used for server-side web development, software development, mathematics, and system scripting, and is popular for Rapid Application Development and as a scripting or glue language to tie existing components because of its high-level, built-in data structures, dynamic typing, and dynamic binding.

5.2 JUPYTER NOTEBOOK:

JUPYTER NOTEBOOK is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design invites extensions to expand and enrich functionality.

Chapter 6

PROJECT DESCRIPTION

PROJECT DESCRIPTION

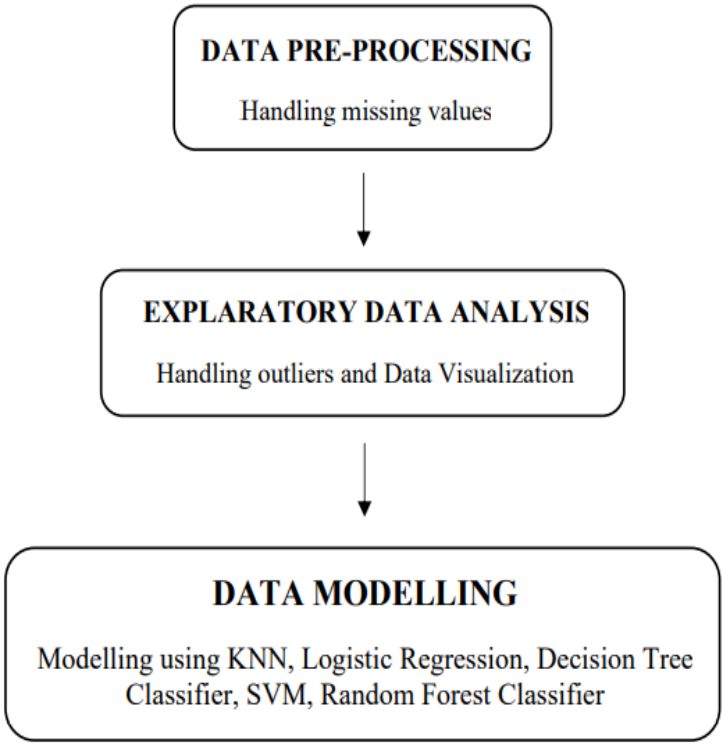
6.1 DATA SET:

This database contains 279 attributes, 206 of which are linear valued and the rest are nominal.

Concerning the study of H. Altay Guvenir: "The aim is to distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups. Class 01 refers to 'normal' ECG classes 02 to 15 refers to different classes of arrhythmia and class 16 refers to the rest of unclassified ones. For the time being, there exists a computer program that makes such a classification. However there are differences between the cardiolog's and the programs classification. Taking the cardiolog's as a gold standard we aim to minimise this difference by means of machine learning tools."The names and id numbers of the patients were recently removed from the database.

6.1.1 DATA MODELLING:

The data is modelled using various algorithms. Let us see each of them, one by one.



6.1.2 K-NEAREST NEIGHBOURS:

K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. The K parameter decides how many of the nearest neighbors have to be considered to determine the property of our unknown point. KNN uses a similarity metric to determine the nearest neighbors.

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

This similarity metric is more often the Euclidean distance between our unknown point and the other points in the dataset. The general formula for Euclidean distance is:

6.1.3 LOGISTIC REGRESSION:

If Y takes on more than two values like in our case, say k of them, we can still use logistic regression. Instead of having one set of parameters β0, β, each class c in 0: (k −1) will have its own offset β(c) 0 and vector β(c), and the predicted conditional probabilities will be

$$\Pr\left(Y = c|\vec{X} = x\right) = \frac{e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}{\sum_c e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}$$

6.1.4 DECISION TREE:

A decision tree is a tree where each node represents a feature (attribute), each link (branch) represents a decision (rule) and each leaf represents an outcome (categorical or continuous value).

Decision trees learn how to best split the dataset into smaller and smaller subsets to predict the target value. The splitting process continues until no further gain can be madeor a preset rule is met, e.g. the maximum depth of the tree is reached.

6.1.5 RANDOM FOREST:

The Random Forest Algorithm is composed of different decision trees, each with the same nodes, but using different data that leads to different leaves. It merges the decisionsof multiple decision trees in order to find an answer, which represents the average of allthese decision trees. Gini-index is often used to how branching is done by nodes in a decision tree.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

This formula uses the class and probability to determine the Gini of each branch on a node, determining which of the branches is more likely to occur. Here, *pi* represents the relative frequency of the class we are observing in the dataset and *c* represents the number of classes.

6.1.6 SUPPORT VECTOR MACHINE:

The main objective of SVM is to find the optimal hyperplane which linearly separates the data points in two components by maximizing the margin. The point above or on the hyperplane will be classified as class +1, and the point below the hyperplane will be classified as class -1. Computing the (soft-margin) SVM classifier amounts to minimizing an expression of the for

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(w \cdot x_i - b)\right)\right] + \lambda\|w\|^2.$$

We focus on the soft-margin classifier since choosing a sufficiently small value for lambda yields the hard-margin classifier for linearly-classifiable input data.

The kernel methods map the data into higher dimensional spaces in the hope that in thishigher-dimensional space the data could become more easily separated or better

structured. There are also no constraints on the form of this mapping, which could even lead to infinite-dimensional spaces. The Sigmoid Kernel (Hyperbolic Tangent) comes from the Neural Networks field, where the bipolar sigmoid function is often used asan activation function for artificial neurons.

$$k(x, y) = \tanh(\alpha x^T y + c)$$

6.1.7 PRINCIPAL COMPONENT ANALYSIS:

In order to reduce the dimensionality of dataset consisting of large inter-related variables,while retaining as much as possible of the variation present in the dataset, we use PCA. The features are transformed into new set of variables called principal components, which are uncorrelated. Covariance is a metric measured between two variables. It givesa measure of how changes in one-dimension affect changes in the other.

$$cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n} (Xi - \bar{x})(Yi - \bar{y})$$

An eigenvector or characteristic vector of a linear transformation, is a nonzero vector that changes at most by a scalar factor when that linear transformation is appliedto it. The corresponding eigenvalue is the factor by which the eigenvector is scaled.

6.2 LIBRARY AND PACKAGES DETAILS:

6.2.1 Numpy

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. It is an open source project and you can use it freely.

NumPy arrays are faster and more compact than Python lists. An array consumes less memory and is convenient to use. NumPy uses much less memory to store data and it provides a mechanism of specifying the data types.

6.2.2 Pandas

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

Pandas is powerful and flexible quantitative analysis tool, pandas has grown into one of the most popular Python libraries. It has an extremely active community of contributors.

6.2.3 Seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data.

6.2.4 Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible. Create publication quality plots. Make interactive figures that can zoom, pan, update.

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. applications.

6.2.5 SciPy

SciPy stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing. Like NumPy, SciPy is open source so we can use it freely. SciPy was created by NumPy's creator Travis Olliphant.

6.2.6 Twilio

The **Twilio Python** Helper Library makes it easy to interact with the Twilio API from your Python application.

Chapter 7

IMPLEMENTATION AND RESULT ANALYSIS

IMPLEMENTATION & RESULT ANALYSIS

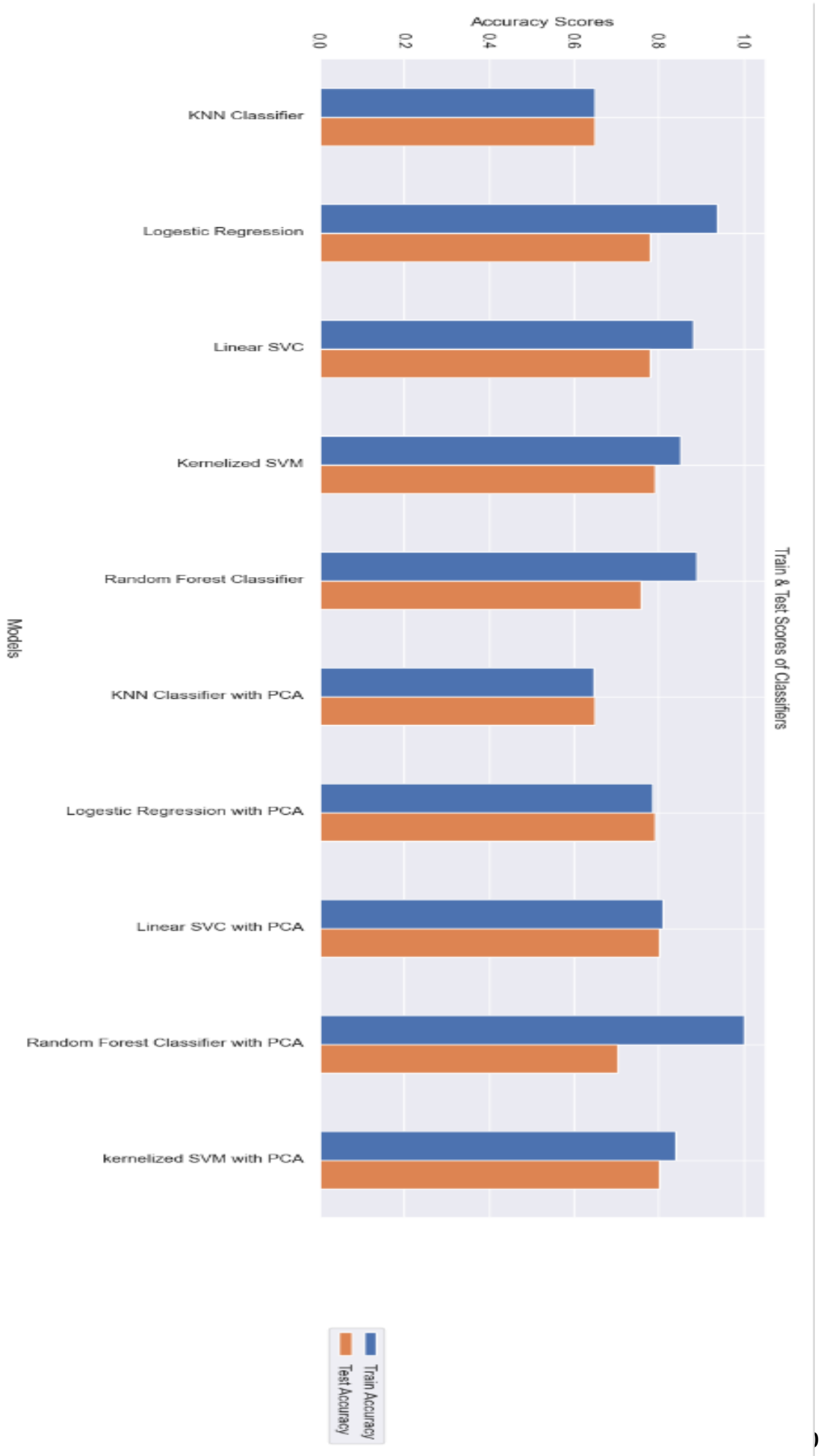
When trained over original data, Kernelized SVM proved to be the best among other classifiers in terms of recall value, with an accuracy percent of 79.12%. Also, LogisticRegression showed better training accuracy.

	Model	Train Accuracy	Test Accuracy
0	KNN Classifier	0.648199	0.648352
1	Logestic Regression	0.939058	0.780220
2	Linear SVC	0.880886	0.780220
3	Kernelized SVM	0.850416	0.791209
4	Random Forest Classifier	0.889197	0.758242

After performing Principal Component Analysis, when data is trained our models, Let found no improvement in KNN results, also, Random Forest too did not yield better results. However, the obtained improvised results from Kernelized SVM model with an accuracy of **80.21%**.

	Model	Train Accuracy	Test Accuracy
0	KNN Classifier	0.648199	0.648352
1	Logestic Regression	0.939058	0.780220
2	Linear SVC	0.880886	0.780220
3	Kernelized SVM	0.850416	0.791209
4	Random Forest Classifier	0.889197	0.758242
5	KNN Classifier with PCA	0.645429	0.648352
6	Logestic Regression with PCA	0.783934	0.791209
7	Linear SVC with PCA	0.808864	0.802198
8	Random Forest Classifier with PCA	1.000000	0.703297
9	kernelized SVM with PCA	0.839335	0.802198

Let visualize our final results.


Accuracy Scores
0.0
0.2
0.4
0.6
0.8
1.0
KNN Classifier
Logestic Regression
Linear SVC
Kernelized SVM
Random Forest Classifier
KNN Classifier with PCA
Logestic Regression with PCA
Linear SVC with PCA
Random Forest Classifier with PCA
kernelized SVM with PCA
Models
Train & Test Scores of Classifiers
Train Accuracy
Test Accuracy

Chapter 8

CONCLUSION

AND

FUTURE ENHANCEMENTS

8.1 CONCLUSION:

This study proposes a method for classification of arrhythmia using ECG data by implementing various machine learning techniques. After cleaning and pre-processing, the data is modelled using multiple machine learning algorithms like K-Nearest Neighbors, Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Linear SVM, and Kernelized SVM. To improve the accuracy of the model, Principal Component Analysis (PCA) is performed over the data to reduce its dimensions and then the data is modelled. The results show that Kernelized SVM outperformed other classifiers when data with reduced features is trained, as PCA reduced the complexity of the original data. The model predicts the absence or presence of cardiac arrhythmia and classifies it into one of the 16 classes with an accuracy of 80.21%. Our results suggest that Kernelized SVM model can be used to diagnose cardio-vascular diseases like arrhythmia in hospitals

8.2 FUTURE ENHANCEMENTS:

In future a plan is there to create an application to sense a heartbeat through smartwatch and regularly monitor Arrhythmia Disease

If Arrhythmia exist message will be send to the emergeny contacts along with the display in mobile and smart watch.

Data set can be replaced with the recent updates of parameters for better accuracy.

Planned to create a self designed CNN model for Arrhythmia detection

Chapter 9

APPENDICES

9.1 SOURCE CODE:

Arrhythmia Predection with Deep-Learning

The Dataset used in this project is available on the UCI machine learning Repository.

- It can be found at: https://archive.ics.uci.edu/ml/datasets/Arrhythmia.
- It consists of 452 different examples spread over 16 classes. Of the 452 examples,
  - 245 are of "normal" people. We also have 12 different types of arrhythmias.
  - Among all these types of arrhythmias, the most representative are the "coronary artery disease" and "Rjgbt boundle branch block".
- We have 279 features, which include age, sex, weight, height of patients and other related information. We explicitly observe that the number of features is relatively high compared to the number of examples we are available.
- Our goal is to predict if a person is suffering from arrhythmia or not, and if **yes**, classify it in to one of 12 available groups.

Importing Essential Libraries

```
import pandas as pd
import numpy as np
import scipy as sp
import math as mt
import seaborn as sns
import conf, json,time
import os
from twilio.rest import Client
from boltiot import Sms,Bolt

import matplotlib.pyplot as plt
#to avoid writing plot.show
```

```
%matplotlib inline
from sklearn.impute import SimpleImputer
df=pd.read_csv("D:/heartbeat/testing/Project-Arrhythmia-master/Project-Arrhythmia-master/Data/arrhythmia.csv",header=None)
df.head()
df.tail()
#Dimension of dataset.

df.shape
#concise summary of the dataframe.

df.info()
#descriptive statistics of dataframe.

df.describe()
```

1. Data preprocessing

Handling Missing Values

When we went through the dataset we observed that out of 279 attributes, 5 Attributes have missing value in the form of '?'. Our approach is

- first replacing '?' with numpy.NAN.
- Imputing the mean using Simple Imputer.

Checking for null values in dataset

```
#Counting total Number of null values

pd.isnull(df).sum().sum()
#Replacing ? with np.nan value-
```

```
df = df.replace('?', np.NaN)
#final counting total number of null values in dataset

nv=pd.isnull(df).sum().sum()
nv
```

Visualizing the distribution of our missing data:

```
pd.isnull(df).sum().plot()
plt.xlabel('Column numbers')
plt.ylabel('Total number of null value in each column')
#Zooming in

pd.isnull(df).sum()[7:17].plot(kind="line")
plt.xlabel('Columns')
plt.ylabel('Total number of null value in each column')
#Dropping the column 13

df.drop(columns = 13, inplace=True)
```

Using the mean strategy for imputation

```
# make copy to avoid changing original data (when Imputing)


new_df = df.copy()
 # make new columns indicating what will be imputed


 cols_with_missing = (col for col in new_df.columns if new_df[col].isnull().any())
 for col in cols_with_missing:
   new_df[col] = new_df[col].isnull()
 # Imputation
```

```
 # my_imputer = SimpleImputer(missing_values=np.nan, strategy='mean')


 my_imputer = SimpleImputer()
 new_df = pd.DataFrame(my_imputer.fit_transform(new_df))
new_df.columns = df.columns
 # imputed dataframe


new_df.head()
 # DataSet with Zero null Values.


pd.isnull(new_df).sum().sum()
```

Generating final dataset

```
 #Creating column names


 final_df_columns=["Age","Sex","Height","Weight","QRS_Dur",
 "P-R_Int","Q-T_Int","T_Int","P_Int","QRS","T","P","J","Heart_Rate",
 "Q_Wave","R_Wave","S_Wave","R'_Wave","S'_Wave","Int_Def","Rag_R_Nom",
 "Diph_R_Nom","Rag_P_Nom","Diph_P_Nom","Rag_T_Nom","Diph_T_Nom",
 "DII00", "DII01","DII02", "DII03",
 "DII04","DII05","DII06","DII07","DII08","DII09","DII10","DII11",
 "DIII00","DIII01","DIII02", "DIII03",
 "DIII04","DIII05","DIII06","DIII07","DIII08","DIII09","DIII10","DIII11",
 "AVR00","AVR01","AVR02","AVR03","AVR04","AVR05","AVR06","AVR07","AVR08
 ","AVR09","AVR10","AVR11",
 "AVL00","AVL01","AVL02","AVL03","AVL04","AVL05","AVL06","AVL07","AVL08",
 "AVL09","AVL10","AVL11",
 "AVF00","AVF01","AVF02","AVF03","AVF04","AVF05","AVF06","AVF07","AVF08","
```

AVF09","AVF10","AVF11",

"V100","V101","V102","V103","V104","V105","V106","V107","V108","V109","V110","V111",

"V200","V201","V202","V203","V204","V205","V206","V207","V208","V209","V210","V211",

"V300","V301","V302","V303","V304","V305","V306","V307","V308","V309","V310","V311",

"V400","V401","V402","V403","V404","V405","V406","V407","V408","V409","V410","V411",

"V500","V501","V502","V503","V504","V505","V506","V507","V508","V509","V510","V511",

"V600","V601","V602","V603","V604","V605","V606","V607","V608","V609","V610","V611",

"JJ_Wave","Amp_Q_Wave","Amp_R_Wave","Amp_S_Wave","R_Prime_Wave","S_Prime_Wave","P_Wave","T_Wave",

"QRSA","QRSTA","DII170","DII171","DII172","DII173","DII174","DII175","DII176","DII177","DII178","DII179",

"DIII180","DIII181","DIII182","DIII183","DIII184","DIII185","DIII186","DIII187","DIII188","DIII189",

"AVR190","AVR191","AVR192","AVR193","AVR194","AVR195","AVR196","AVR197","AVR198","AVR199",

"AVL200","AVL201","AVL202","AVL203","AVL204","AVL205","AVL206","AVL207","AVL208","AVL209",

"AVF210","AVF211","AVF212","AVF213","AVF214","AVF215","AVF216","AVF217","AVF218","AVF219",

"V1220","V1221","V1222","V1223","V1224","V1225","V1226","V1227","V1228","V1229",

"V2230","V2231","V2232","V2233","V2234","V2235","V2236","V2237","V2238","V2239",

"V3240","V3241","V3242","V3243","V3244","V3245","V3246","V3247","V3248","V3249",

"V4250","V4251","V4252","V4253","V4254","V4255","V4256","V4257","V4258","V4259",
"V5260","V5261","V5262","V5263","V5264","V5265","V5266","V5267","V5268","V5269",
"V6270","V6271","V6272","V6273","V6274","V6275","V6276","V6277","V6278","V6279","class"]

#Adding Column names to dataset

```
new_df.columns=final_df_columns
new_df.to_csv("new data with target class.csv")
new_df.head()
```

As our dataframe is completely cleaned and preprocessed. we will remove the target attribute and store our final dataframe.

```
target=new_df["class"]

final_df = new_df.drop(columns ="class")
final_df.shape
```

2. Exploratory Data Analysis (EDA)

Analyzing data sets to summarize their main characteristics.

Making List of all the type of Arrythmia corresponsing to their class label

#List with class names

```
class_names = ["Normal",
"Ischemic changes (CAD)",
"Old Anterior Myocardial Infraction",
```

```
"Old Inferior Myocardial Infraction",
"Sinus tachycardy",
"Sinus bradycardy",
"Ventricular Premature Contraction (PVC)",
"Supraventricular Premature Contraction",
"Left Boundle branch block",
"Right boundle branch block",
"1.Degree AtrioVentricular block",
"2.Degree AV block",
"3.Degree AV block",
"Left Ventricule hypertrophy",
"Atrial Fibrillation or Flutter",
"Others"]
```

Analyzing the dataset and check how many examples we have for each class:

we need to sort our dataset with respect to class attributes to count the number of instances available for each class

```
t=new_df.sort_values(by=["class"])
 # Counting the number of instances for each class

 la = t["class"].value_counts(sort=False).tolist()
la
 sns.countplot(x ='class',data =new_df)
plt.show()
```

Lets Count the total number of instances we have for each class.

```
 values = la[0:10]
```

```
 values.extend([0,0,0])
 values.extend(la[10:13])
print(values)
 labels = class_names
 Log_Norm = []
 for i in values:
    Log_Norm.append(mt.log10(i+1))
 fig1, ax1 = plt.subplots(figsize=(16,9))
 patches = plt.pie(Log_Norm, autopct='%1.1f%%', startangle=90)

 leg = plt.legend( loc = 'best', labels=['%s, %1.1f %%' % (l, s) for l, s in zip(labels, Log_Norm)])
 plt.axis('equal')

 for text in leg.get_texts():
    plt.setp(text, color = 'Black')
 plt.tight_layout()
 plt.show()
```

We found that Of the 452 examples, 245 are of class A which refers to "normal" people. We also have 12 different types of arrhythmias and 3 other type of arrthmias are not present in our dataset.

Handling Outliers & Data Visualization

```
#looking for pairwise relationships and outliers

g = sns.PairGrid(final_df, vars=['Age', 'Sex', 'Height', 'Weight'],hue='Sex', palette='BrBG')
g.map(plt.scatter, alpha=0.8)
g.add_legend();
```

According to scatter plots, there are few outliers in 'height' and 'weight' attributes.check the maximums of heights and weights

```
sorted(final_df['Height'], reverse=True)[:10]
```

The tallest person ever lived in the world was **272** cm (1940). His followers were **267** cm(1905)

and **263.5** cm(1969). Replacing **780** and **608** with **180** and **108** cm respectively

```
final_df['Height']=final_df['Height'].replace(608,108)
final_df['Height']=final_df['Height'].replace(780,180)
```

```
sorted(final_df['Weight'], reverse=True)[:10]
```

176 kgs is a possible weight. so we'll keep them in the dataframe.

```
sns.boxplot(data=final_df[["QRS_Dur","P-R_Int","Q-T_Int","T_Int","P_Int"]]);
```

PR interval is the period, measured in milliseconds, that extends from the beginning of the P wave until the beginning of the QRS complex; it is normally between 120 and 200ms in duration.

```
 final_df['P-R_Int'].value_counts().sort_index().head().plot(kind='bar')
 plt.xlabel('P-R Interval Values')
plt.ylabel('Count');
 final_df['P-R_Int'].value_counts().sort_index().tail().plot(kind='bar')
 plt.xlabel('P-R Interval Values')
plt.ylabel('Count');
```

PR Interval data is including outliers 0(x18). we'll keep them

```
sns.boxplot(data=final_df[["QRS","T","P","J","Heart_Rate"]]);
sns.boxplot(data=final_df[["R'_Wave","S'_Wave","Int_Def","Rag_R_Nom"]]);
```

S'Wave has 0's which is not a NaN. So, we can't assume it as including outliers.

```
final_df["S'_Wave"].value_counts().sort_index(ascending=False)
final_df["V101"].value_counts().sort_index(ascending=False)
```

V101 has an outlier, but when we look at other sets (V201, V301, V501) we can see that there's an outlier similarly. Since our data is heavily biased, I can't say these outliers should be dropped.

For example, when we look at our data, we can see that class # 8 (Supraventricular Premature Contraction) **has only 2 instances**. Or # 3 (Ventricular Premature Contraction (PVC)) has only 3. The outliers appearing with our plots might belong to these instances and needs to be kept.

```
final_df["V201"].value_counts().sort_index(ascending=False)
final_df["V301"].value_counts().sort_index(ascending=False)
final_df["V501"].value_counts().sort_index(ascending=False)
```

Now we can see outlier within the last two attributes of each series(DIII188, DIII189, AVR198, AVR199, AVL208, AVL209, AVF218, AVF219, V2238, V2239, V3248, V3249,V4258, V4259,V5268, V5269, V6278, V6279). Similiarly assuming that these outliers might belong to the classes with few instances.

```
 sns.set(rc={'figure.figsize':(11.7,5.27)})
sns.boxplot(data=final_df[["AVR190","AVR191","AVR192","AVR193","AVR194","AVR195","AVR196","AVR197","AVR198","AVR199"]]);
 sns.set(rc={'figure.figsize':(11.7,5.27)})
sns.boxplot(data=final_df[["AVL200","AVL201","AVL202","AVL203","AVL204","AVL205","AVL206","AVL207","AVL208","AVL209"]]);
 sns.set(rc={'figure.figsize':(11.7,5.27)})
sns.boxplot(data=final_df[["AVF210","AVF211","AVF212","AVF213","AVF214","AVF215","AVF216","AVF217","AVF218","AVF219"]]);
 #finding correlation with target feature using pearson correlation


        target=new_df["class"]
        pearsoncorr = final_df.corrwith(other = target,method='pearson')
    pearsoncorr.values
```

Feature Scaling and Splitting dataset

We will be using 80% of our dataset for training purpose and 20% for testing purpose.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(final_df, target ,test_size=0.2, random_state=1)
from sklearn.preprocessing import StandardScaler scaler = StandardScaler() scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
import warnings
warnings.filterwarnings('ignore')
```

Evaluation strategy

As the dependent variable is a categorical variable we will be using classification models. The best evaluation strategy for classification models is comparing the precision and recall. Thinking about the classification evaluation metrics, the importance of our models' precitions (we can't accept a result having the probability of saying to a healthy person that you have Cardiac Arrhythmia (FN)).

We definitely will focus on Sensitivity (the percentage of sick people who are correctly identified as having the condition) not Specificity (percentage of healthy people who are correctly identified as not having the condition).

```
# importing evaluation metrices.From sklearn.metrics import r2_score,mean_squared_error,accuracy_score,recall_score,precision_score,confusion_matrix,classification_report from scikitplot.metrics import plot_confusion_matrix,plot_roc
```

3. Modeling

```
# will store result of each model.

result = pd.DataFrame(columns=['Model','Train Accuracy','Test Accuracy'])
```

KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
knnclassifier = KNeighborsClassifier()
knnclassifier.fit(X_train, y_train)
y_pred = knnclassifier.predict(X_test)
```

```
    knn_train_accuracy = accuracy_score(y_train, knnclassifier.predict(X_train))
    knn_test_accuracy = accuracy_score(y_test, knnclassifier.predict(X_test))
    knn_train_recall = recall_score(y_train,
    knnclassifier.predict(X_train),average="weighted")
    knn_test_recall = recall_score(y_test,
    knnclassifier.predict(X_test),average="weighted")
    print('Train Recall score: {}'
        .format(knn_train_recall))
    print('Test Recall score: {}'
        .format(knn_test_recall))
    confusion_matrix(y_test, y_pred)
    result = result.append(pd.Series({'Model':'KNN Classifier','Train
    Accuracy':knn_train_accuracy,'Test
    Accuracy':knn_test_accuracy}),ignore_index=True)
result
```

Logistic regression

```
    from sklearn.linear_model import LogisticRegression
    lgclassifier = LogisticRegression(solver = 'saga',random_state = 0)
    lgclassifier.fit(X_train, y_train)
    y_pred = lgclassifier.predict(X_test)

    lg_train_recall = recall_score(y_train,
    lgclassifier.predict(X_train),average='weighted')
    lg_test_recall = recall_score(y_test, lgclassifier.predict(X_test),average='weighted')
```

```
lg_train_accuracy = accuracy_score(y_train, lgclassifier.predict(X_train))
lg_test_accuracy = accuracy_score(y_test, lgclassifier.predict(X_test))
print('Train Recall score: {}'
    .format(lg_train_recall))
print('Test Recall score: {}'
    .format(lg_test_recall))
confusion_matrix(y_test, y_pred)
result = result.append(pd.Series({'Model':'Logestic Regression','Train
Accuracy':lg_train_accuracy,'Test
Accuracy':lg_test_accuracy}),ignore_index=True )
result
```

Linear SVM

```
from sklearn.svm import LinearSVC
lsvclassifier = LinearSVC(C=0.01)
lsvclassifier.fit(X_train, y_train)
y_pred_test = lsvclassifier.predict(X_test)
y_pred_train = lsvclassifier.predict(X_train)
lsvc_train_accuracy = accuracy_score(y_train, y_pred_train)
lsvc_test_accuracy = accuracy_score(y_test, y_pred_test)
lsvc_train_recall = recall_score(y_train,y_pred_train,average="weighted" )
lsvc_test_recall = recall_score(y_test, y_pred_test,average="weighted")
print('Train Recall score: {}'
    .format(lsvc_train_recall))
print('Test Recall score: {}'
```

```
    .format(lsvc_test_recall))
confusion_matrix(y_test, y_pred_test)
result = result.append(pd.Series({'Model':'Linear SVC','Train
Accuracy':lsvc_train_accuracy,'Test
Accuracy':lsvc_test_accuracy}),ignore_index=True )
result
```

Kernelized SVM

```
from sklearn import svm
KSVM_clf = svm.SVC(kernel='sigmoid',C=10,gamma=0.001)
KSVM_clf.fit(X_train, y_train)
y_pred_train = KSVM_clf.predict(X_train)
y_pred_test = KSVM_clf.predict(X_test)
ksvm_train_recall = recall_score(y_train,y_pred_train,average="weighted" )
ksvm_test_recall = recall_score(y_test, y_pred_test,average="weighted")

ksvm_train_accuracy = accuracy_score(y_train, y_pred_train)
ksvm_test_accuracy = accuracy_score(y_test, y_pred_test)
print('Train Recall score: {}'
    .format(ksvm_train_recall))
print('Test Recall score: {}'
    .format(ksvm_test_recall))
confusion_matrix(y_test, y_pred_test)
result = result.append(pd.Series({'Model':'Kernelized SVM','Train
Accuracy':ksvm_train_accuracy,'Test
Accuracy':ksvm_test_accuracy}),ignore_index=True )
```

result

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators=300,
criterion='gini',max_features=100,max_depth=10,max_leaf_nodes=30)
rf_clf.fit(X_train, y_train)
y_pred_train = rf_clf.predict(X_train)
y_pred_test = rf_clf.predict(X_test)

rf_train_accuracy = accuracy_score(y_train, y_pred_train)
rf_test_accuracy = accuracy_score(y_test, y_pred_test)
rf_train_recall = recall_score(y_train,y_pred_train,average="weighted" )
rf_test_recall = recall_score(y_test, y_pred_test,average="weighted")
print('Train Recall score: {}'
    .format(rf_train_recall))
print('Test Recall score: {}'
    .format(rf_test_recall))
confusion_matrix(y_test, y_pred_test)
result = result.append(pd.Series({'Model':'Random Forest Classifier','Train
Accuracy':rf_train_accuracy,'Test Accuracy':rf_test_accuracy}),ignore_index=True
)
result
```

We found that the best model in term of recall Score is kernelized SVM with accuracy percentage of 79.12 over other models. We also found that Logestic

Regression has better accuracy score

.

PCA

We will be using PCA(Principal Component Analysis) to reduce the dimension of our sampled dataset to get best feature to find better accuracy.

```
from sklearn.decomposition import PCA
pca = PCA(.98)
pca.fit(X_train)
pca.n_components_
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
from sklearn.model_selection import StratifiedKFold
kFold = StratifiedKFold(n_splits=5)
from sklearn.model_selection import GridSearchCV
```

KNN with PCA

```
from sklearn.neighbors import KNeighborsClassifier
knnp_clf = KNeighborsClassifier(n_neighbors=5)
knnp_clf.fit(X_train_pca, y_train)
y_pred_train = knnp_clf.predict(X_train_pca)
```

```
y_pred_test = knnp_clf.predict(X_test_pca)
knnp_train_recall = recall_score(y_train, y_pred_train, average='weighted')
knnp_test_recall = recall_score(y_test, y_pred_test, average='weighted')
knnp_train_accuracy = accuracy_score(y_train, y_pred_train)
knnp_test_accuracy = accuracy_score(y_test, y_pred_test)
a=accuracy_score(y_test, y_pred_test)
print('Train Recall score: {}'
    .format(knnp_train_recall))
print('Test Recall score: {}'
    .format(knnp_test_recall))

confusion_matrix(y_test, y_pred_test)
print(a)
result = result.append(pd.Series({'Model':'KNN Classifier with PCA','Train
Accuracy':knnp_train_accuracy,'Test
Accuracy':knnp_test_accuracy}),ignore_index=True)
result
```

we didn't find any improvement on PCA data with knn classifier model.

Logestic with PCA

```
from sklearn.linear_model import LogisticRegression
lgp_clf = LogisticRegression(solver='saga',C=0.01)
lgp_clf.fit(X_train_pca, y_train)
y_pred_train = lgp_clf.predict(X_train_pca)
```

```
y_pred_test = lgp_clf.predict(X_test_pca)
lgp_train_recall = recall_score(y_train, y_pred_train, average='weighted')
lgp_test_recall = recall_score(y_test, y_pred_test, average='weighted')
lgp_train_accuracy = accuracy_score(y_train, y_pred_train)
lgp_test_accuracy = accuracy_score(y_test, y_pred_test)
b=accuracy_score(y_test, y_pred_test)
print('Train Recall score: {}'
    .format(lgp_train_recall))
print('Test Recall score: {}'
    .format(lgp_test_recall))

confusion_matrix(y_test, y_pred_test)
print(b)
result = result.append(pd.Series({'Model':'Logestic Regression with PCA','Train
Accuracy':lgp_train_accuracy,'Test
Accuracy':lgp_test_accuracy}),ignore_index=True )
result
```

Linear svm with PCA

```
from sklearn.svm import LinearSVC
LSVC_clf = LinearSVC(C=0.001)
LSVC_clf.fit(X_train_pca, y_train)
y_pred_train = LSVC_clf.predict(X_train_pca)
y_pred_test = LSVC_clf.predict(X_test_pca)
lsvcp_train_recall = recall_score(y_train, y_pred_train, average='weighted')
```

```
lsvcp_test_recall = recall_score(y_test, y_pred_test, average='weighted')

lsvcp_train_accuracy = accuracy_score(y_train, y_pred_train)

lsvcp_test_accuracy = accuracy_score(y_test, y_pred_test)

c=accuracy_score(y_test, y_pred_test)

print('Train Recall score: {}'

    .format(lsvcp_train_recall))

print('Test Recall score: {}'

    .format(lsvcp_test_recall))


confusion_matrix(y_test, y_pred_test)

print(c)

result = result.append(pd.Series({'Model':'Linear SVC with PCA','Train
Accuracy':lsvcp_train_accuracy,'Test
Accuracy':lsvcp_test_accuracy}),ignore_index=True )

result
```

Random Forest Classifier With PCA

```
from sklearn.ensemble import RandomForestClassifier

rfp_clf = RandomForestClassifier()

rfp_clf.fit(X_train_pca, y_train)

y_pred_train = rfp_clf.predict(X_train_pca)

y_pred_test = rfp_clf.predict(X_test_pca)

rfp_train_recall = recall_score(y_train, y_pred_train, average='weighted')

rfp_test_recall = recall_score(y_test, y_pred_test, average='weighted')
```

```
rfp_train_accuracy = accuracy_score(y_train, y_pred_train)

rfp_test_accuracy = accuracy_score(y_test, y_pred_test)

d=accuracy_score(y_test, y_pred_test)

print('Train Recall score: {}'

    .format(rfp_train_recall))

print('Test Recall score: {}'

    .format(rfp_test_recall))


confusion_matrix(y_test, y_pred_test)

print(d)

result = result.append(pd.Series({'Model':'Random Forest Classifier with
PCA','Train Accuracy':rfp_train_accuracy,'Test
Accuracy':rfp_test_accuracy}),ignore_index=True )

result
```

Random Forest Classifier is overfitting the model and is not yielding good result

Kernal svm with PCA

```
from sklearn import svm

KSVM_clf = svm.SVC(kernel='sigmoid',C=10,gamma=0.001)


KSVM_clf.fit(X_train_pca, y_train)

y_pred_train = KSVM_clf.predict(X_train_pca)

y_pred_test = KSVM_clf.predict(X_test_pca)

ksvmp_train_recall = recall_score(y_train, y_pred_train, average='weighted')

ksvmp_test_recall = recall_score(y_test, y_pred_test, average='weighted')
```

```
ksvmp_train_accuracy = accuracy_score(y_train, y_pred_train,)

ksvmp_test_accuracy = accuracy_score(y_test, y_pred_test)

e=accuracy_score(y_test, y_pred_test)

print('Train Recall score: {}'

    .format(ksvmp_train_recall))

print('Test Recall score: {}'

    .format(ksvmp_test_recall))


confusion_matrix(y_test, y_pred_test)

print(e)

result = result.append(pd.Series({'Model':'kernelized SVM with PCA','Train
Accuracy':ksvmp_train_accuracy,'Test
Accuracy':ksvmp_test_accuracy}),ignore_index=True )

result

result

print(a,b,c,d,e)
```

Result

```
ax=result.plot(kind="bar",figsize=(20,5))

ax.set_xticks(result.index)

ax.set_xticklabels(result.Model,rotation=90)

plt.title('Train & Test Scores of Classifiers')

plt.xlabel('Models')

plt.ylabel('Accuracy Scores')

plt.legend(loc=4 , bbox_to_anchor=(1.2, 0))
```

```
plt.show();
```

Conclusion

The models started performing better after we applied PCA on the resampled data. The reason behind this is, PCA reduces the complexity of the data. It creates components based on giving importance to variables with large variance and also the components which it creates are non collinear in nature which means it takes care of collinearity in large data set. PCA also improves the overall execution time and quality of the models and it is very beneficial when we are working with huge amount of variables.

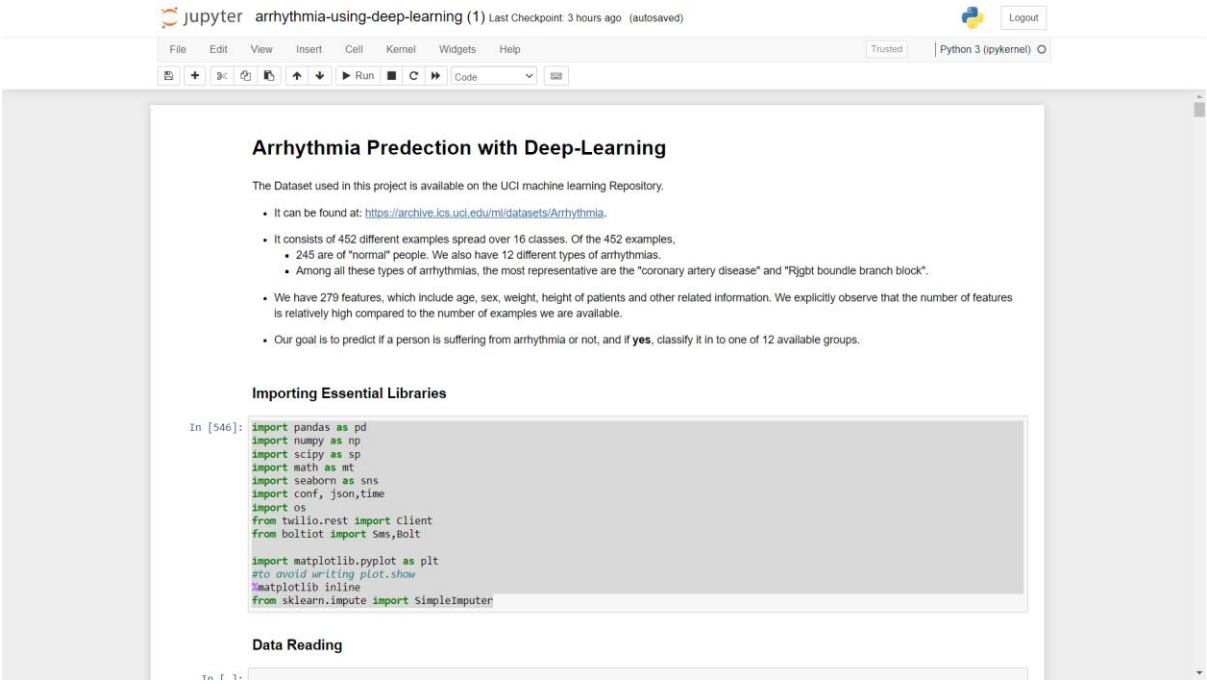
The Best model in term of recall score is **Kernalized SVM with PCA** having accuracy of **80.21%.**

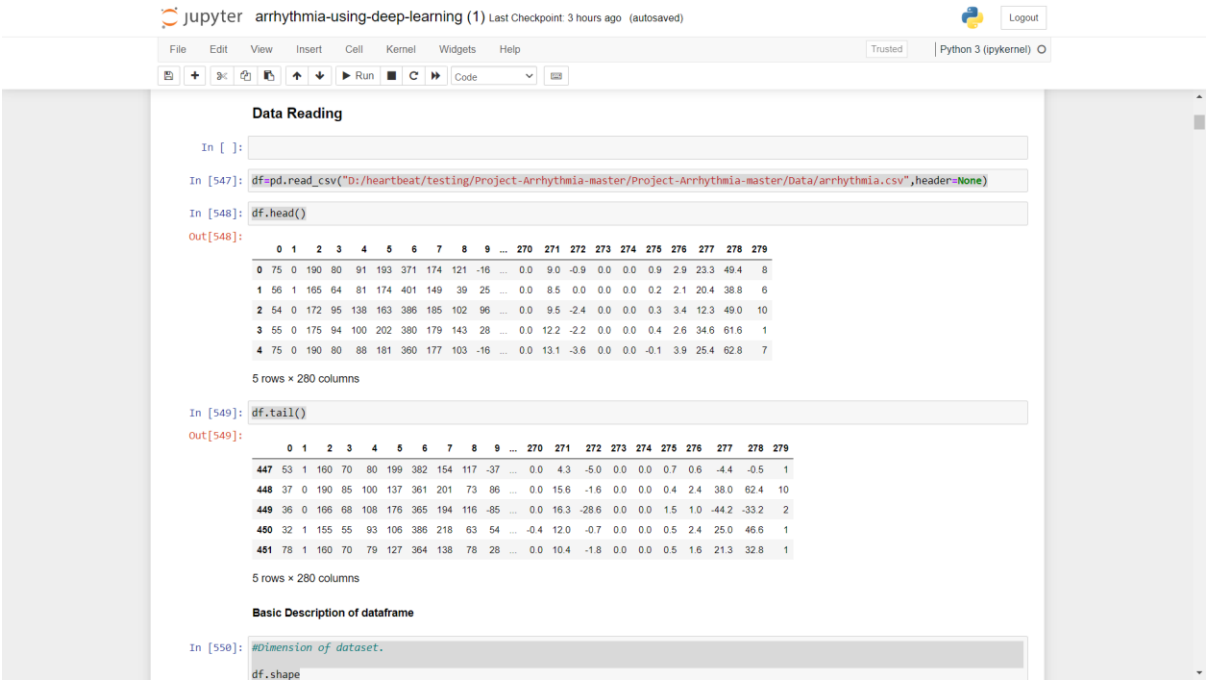
SENDING AN SMS TO MOBILE PHONE

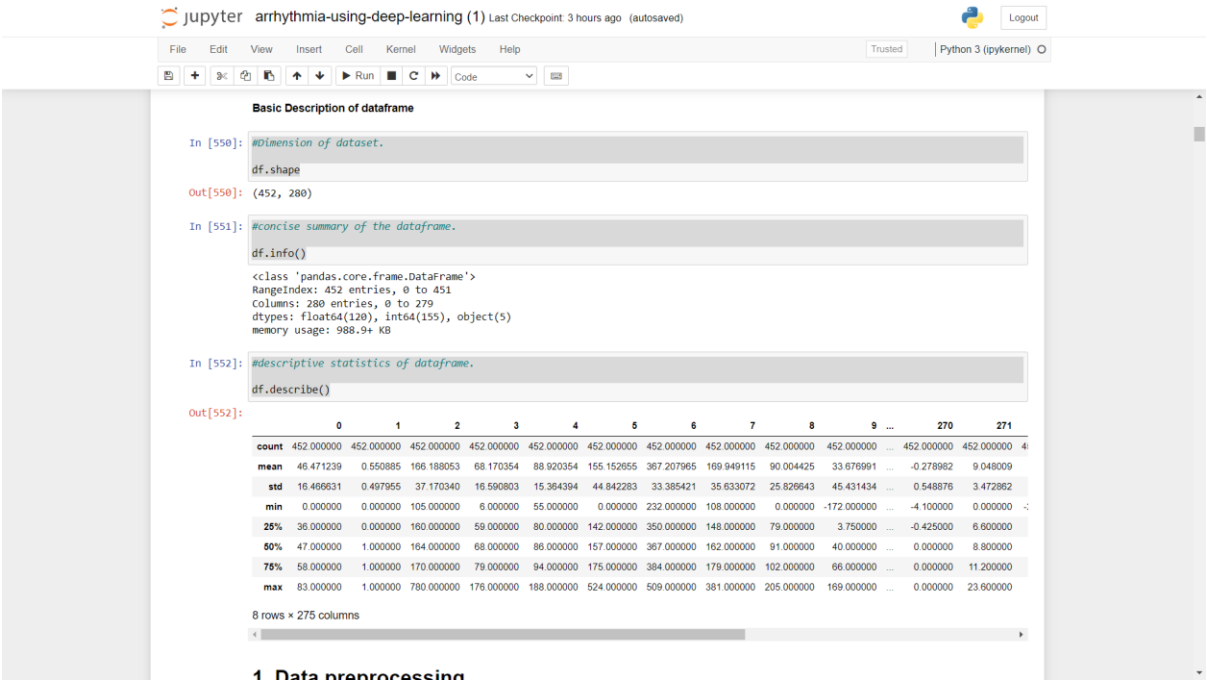
```
sms = Sms(conf.SID, conf.AUTH_TOKEN, conf.TO_NUMBER,
conf.FROM_NUMBER)

p=a

q=b

r=c

s=d

t=e

z=(max(p,q,r,s,t))

if z>0.75 :
```

```
            response = sms.send_sms("HEY Chikku your current Heart Rate is
ubnormal " )

        elif z<0.65:

            response = sms.send_sms("HEY Chikku The Current Heart Rate is low " )

        else:

            response = sms.send_sms("HEY Chikku The Current Heart Rate is normal "
)



        if Sms(conf.SID, conf.AUTH_TOKEN, conf.TO_NUMBER,
conf.FROM_NUMBER):

            print('SMS message successfully sent!')

        else:

            print('Could not send SMS message.')

        print(z)



        time.sleep(10)
```

9.2 SCREENSHOTS:







jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

Basic Description of dataframe

In [550]:
```
#Dimension of dataset.

df.shape
```

Out[550]: (452, 280)

In [551]:
```
#concise summary of the dataframe.

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 452 entries, 0 to 451
Columns: 280 entries, 0 to 279
dtypes: float64(120), int64(155), object(5)
memory usage: 988.9+ KB
```

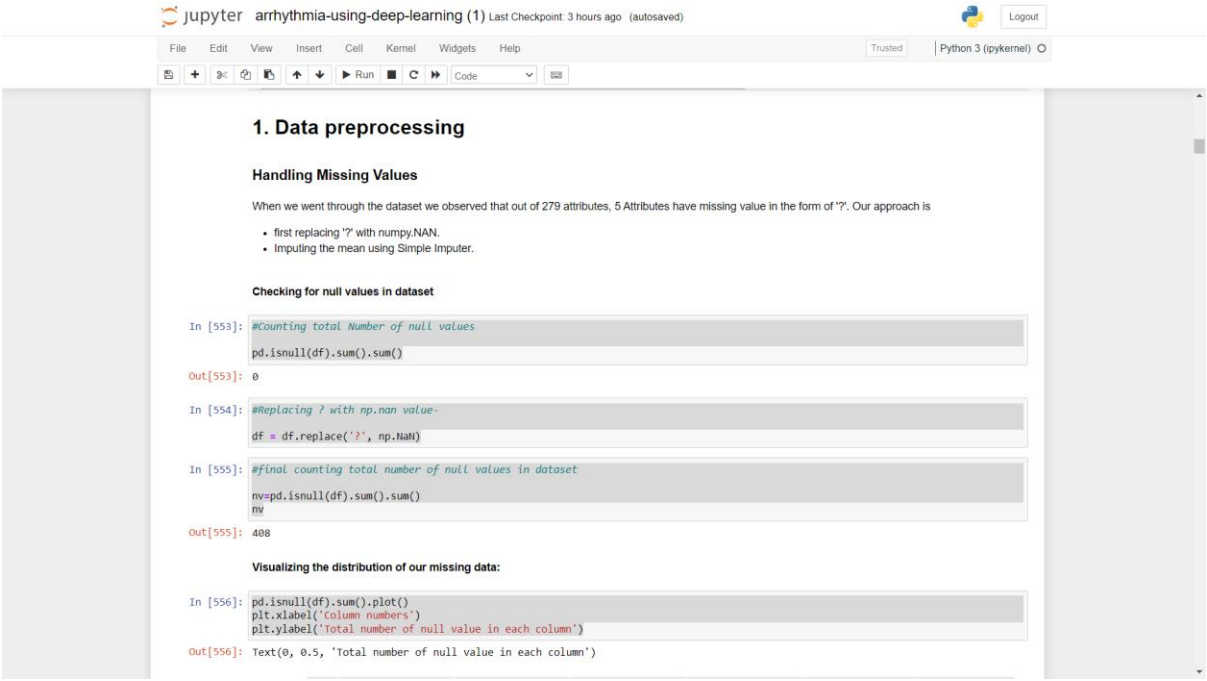
In [552]:
```
#descriptive statistics of dataframe.

df.describe()
```

Out[552]:

	0	1	2	3	4	5	6	7	8	9	...	270	271
count	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	...	452.000000	452.000000
mean	46.471239	0.550885	166.188053	68.170354	88.920354	155.152655	367.207965	169.949115	90.004425	33.676991	...	-0.278982	9.048009
std	16.466631	0.497955	37.170340	16.590803	15.364394	44.842283	33.385421	35.633072	25.826643	45.431434	...	0.548876	3.472862
min	0.000000	0.000000	105.000000	6.000000	55.000000	0.000000	232.000000	108.000000	0.000000	-172.000000	...	-4.100000	0.000000
25%	36.000000	0.000000	160.000000	59.000000	80.000000	142.000000	350.000000	148.000000	79.000000	3.750000	...	-0.425000	6.600000
50%	47.000000	1.000000	164.000000	68.000000	86.000000	157.000000	367.000000	162.000000	91.000000	40.000000	...	0.000000	8.800000
75%	58.000000	1.000000	170.000000	79.000000	94.000000	175.000000	384.000000	179.000000	102.000000	66.000000	...	0.000000	11.200000
max	83.000000	1.000000	780.000000	176.000000	188.000000	524.000000	509.000000	381.000000	205.000000	169.000000	...	0.000000	23.600000

8 rows × 275 columns

1. Data preprocessing



jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

1. Data preprocessing

Handling Missing Values

When we went through the dataset we observed that out of 279 attributes, 5 Attributes have missing value in the form of '?'. Our approach is

- first replacing '?' with numpy.NAN.
- Imputing the mean using Simple Imputer.

Checking for null values in dataset

In [553]:
```
#Counting total Number of null values

pd.isnull(df).sum().sum()
```

Out[553]: 0

In [554]:
```
#Replacing ? with np.nan value-

df = df.replace('?', np.NaN)
```

In [555]:
```
#final counting total number of null values in dataset

nv=pd.isnull(df).sum().sum()
nv
```

Out[555]: 408

Visualizing the distribution of our missing data:

In [556]:
```
pd.isnull(df).sum().plot()
plt.xlabel('Column numbers')
plt.ylabel('Total number of null value in each column')
```

Out[556]: Text(0, 0.5, 'Total number of null value in each column')

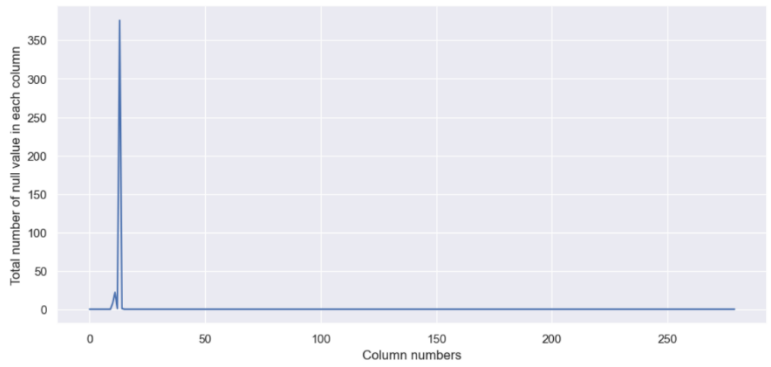
jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

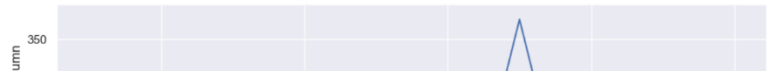
Run Code

Out[556]: Text(0, 0.5, 'Total number of null value in each column')



In [557]:
```
#Zooming in

pd.isnull(df).sum()[7:17].plot(kind="line")
plt.xlabel('Columns')
plt.ylabel('Total number of null value in each column')
```

Out[557]: Text(0, 0.5, 'Total number of null value in each column')



jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

Column 13 contains more than 350 missing values out of total 452 instances. so we will drop column 13. other attributes have comparatively less null values. So instead of droping, we will replace the null value of other attributes with their mean values.

In [558]:
```
#Dropping the column 13

df.drop(columns = 13, inplace=True)
```

Using the mean strategy for imputation

In [559]:
```
# make copy to avoid changing original data (when Imputing)

new_df = df.copy()
```

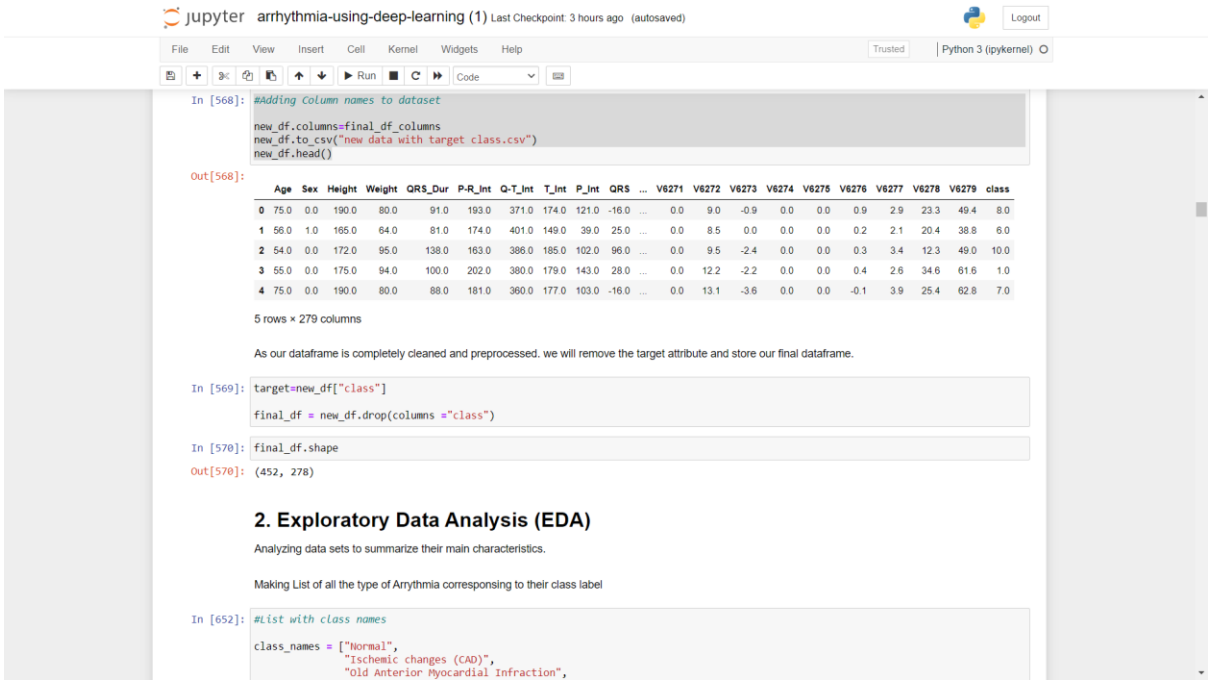
In [560]:
```
# make new columns indicating what will be imputed

cols_with_missing = (col for col in new_df.columns if new_df[col].isnull().any())
for col in cols_with_missing:
    new_df[col] = new_df[col].isnull()
```

In [561]:
```
# Imputation
# my_imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

my_imputer = SimpleImputer()
new_df = pd.DataFrame(my_imputer.fit_transform(new_df))
new_df.columns = df.columns
```

In [562]:
```
# imputed dataframe

new_df.head()
```

Out[562]:

	0	1	2	3	4	5	6	7	8	9	...	270	271	272	273	274	275	276	277	278	279
0	75.0	0.0	190.0	80.0	91.0	193.0	371.0	174.0	121.0	-16.0	...	0.0	9.0	-0.9	0.0	0.0	0.9	2.9	23.3	49.4	8.0
1	56.0	1.0	165.0	64.0	81.0	174.0	401.0	149.0	39.0	25.0	...	0.0	8.5	0.0	0.0	0.0	0.2	2.1	20.4	38.8	6.0
2	54.0	0.0	172.0	95.0	138.0	163.0	386.0	185.0	102.0	96.0	...	0.0	9.5	-2.4	0.0	0.0	0.3	3.4	12.3	49.0	10.0
3	55.0	0.0	175.0	94.0	100.0	202.0	380.0	179.0	143.0	28.0	...	0.0	12.2	-2.2	0.0	0.0	0.4	2.6	34.6	61.6	1.0
4	75.0	0.0	190.0	80.0	88.0	181.0	360.0	177.0	103.0	-16.0	...	0.0	13.1	-3.6	0.0	0.0	-0.1	3.9	25.4	62.8	7.0



jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [568]:
```
#Adding Column names to dataset

new_df.columns=final_df_columns
new_df.to_csv("new data with target class.csv")
new_df.head()
```

Out[568]:

	Age	Sex	Height	Weight	QRS_Dur	P-R_Int	Q-T_Int	T_Int	P_Int	QRS	...	V6271	V6272	V6273	V6274	V6275	V6276	V6277	V6278	V6279	class
0	75.0	0.0	190.0	80.0	91.0	193.0	371.0	174.0	121.0	-16.0	...	0.0	9.0	-0.9	0.0	0.0	0.9	2.9	23.3	49.4	8.0
1	56.0	1.0	165.0	64.0	81.0	174.0	401.0	149.0	39.0	25.0	...	0.0	8.5	0.0	0.0	0.0	0.2	2.1	20.4	38.8	6.0
2	54.0	0.0	172.0	95.0	138.0	163.0	386.0	185.0	102.0	96.0	...	0.0	9.5	-2.4	0.0	0.0	0.3	3.4	12.3	49.0	10.0
3	55.0	0.0	175.0	94.0	100.0	202.0	380.0	179.0	143.0	28.0	...	0.0	12.2	-2.2	0.0	0.0	0.4	2.6	34.6	61.6	1.0
4	75.0	0.0	190.0	80.0	88.0	181.0	360.0	177.0	103.0	-16.0	...	0.0	13.1	-3.6	0.0	0.0	-0.1	3.9	25.4	62.8	7.0

5 rows × 279 columns

As our dataframe is completely cleaned and preprocessed. we will remove the target attribute and store our final dataframe.

In [569]:
```
target=new_df["class"]

final_df = new_df.drop(columns ="class")
```

In [570]:
```
final_df.shape
```

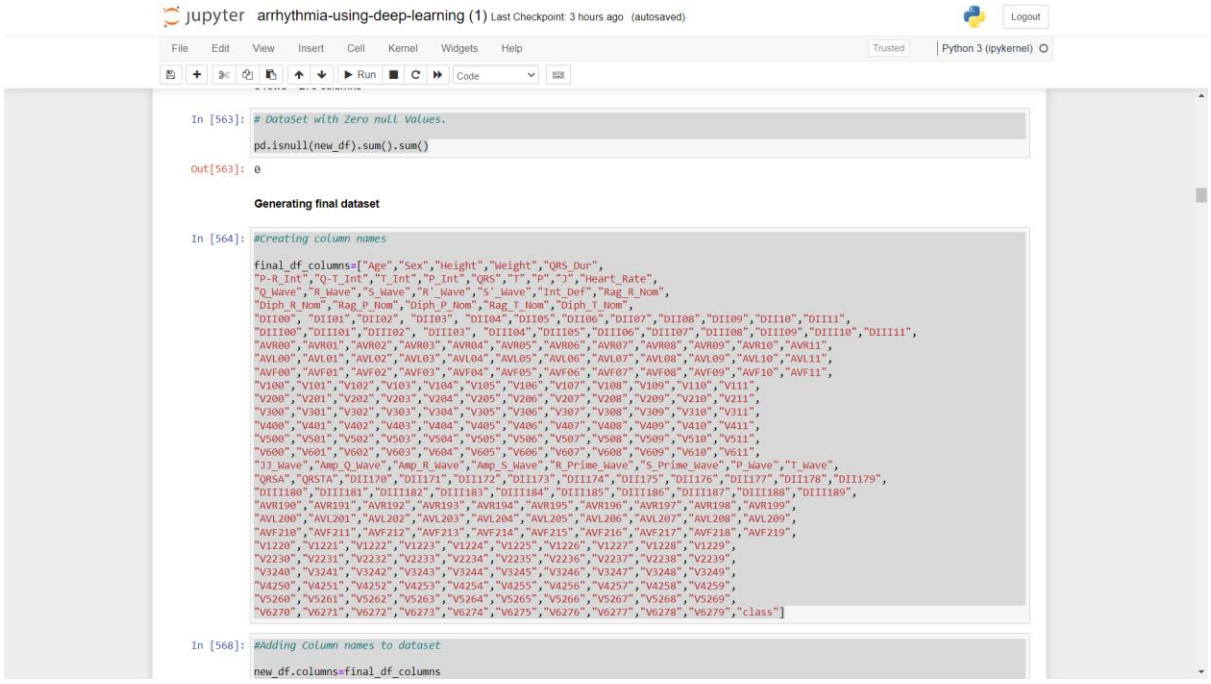
Out[570]: (452, 278)

2. Exploratory Data Analysis (EDA)

Analyzing data sets to summarize their main characteristics.

Making List of all the type of Arrythmia corresponsing to their class label

In [652]:
```
#List with class names

class_names = ["Normal",
               "Ischemic changes (CAD)",
               "Old Anterior Myocardial Infraction",
```



jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

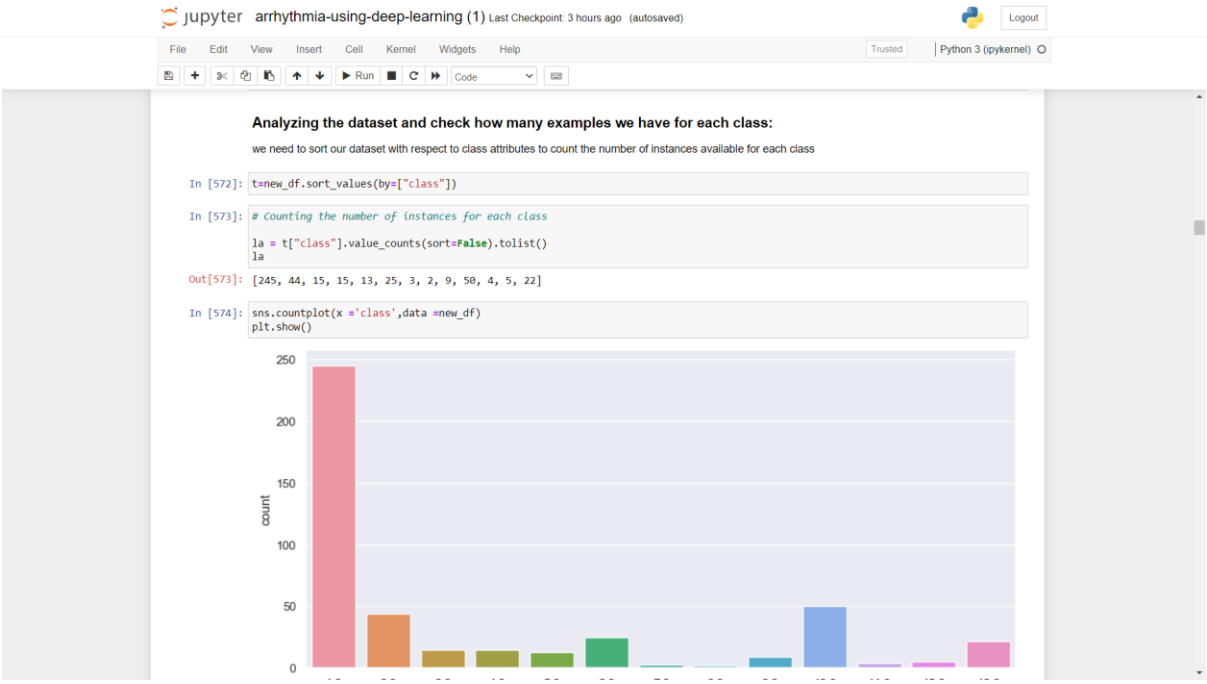
In [563]:
```
# DataSet with Zero null Values.

pd.isnull(new_df).sum().sum()
```

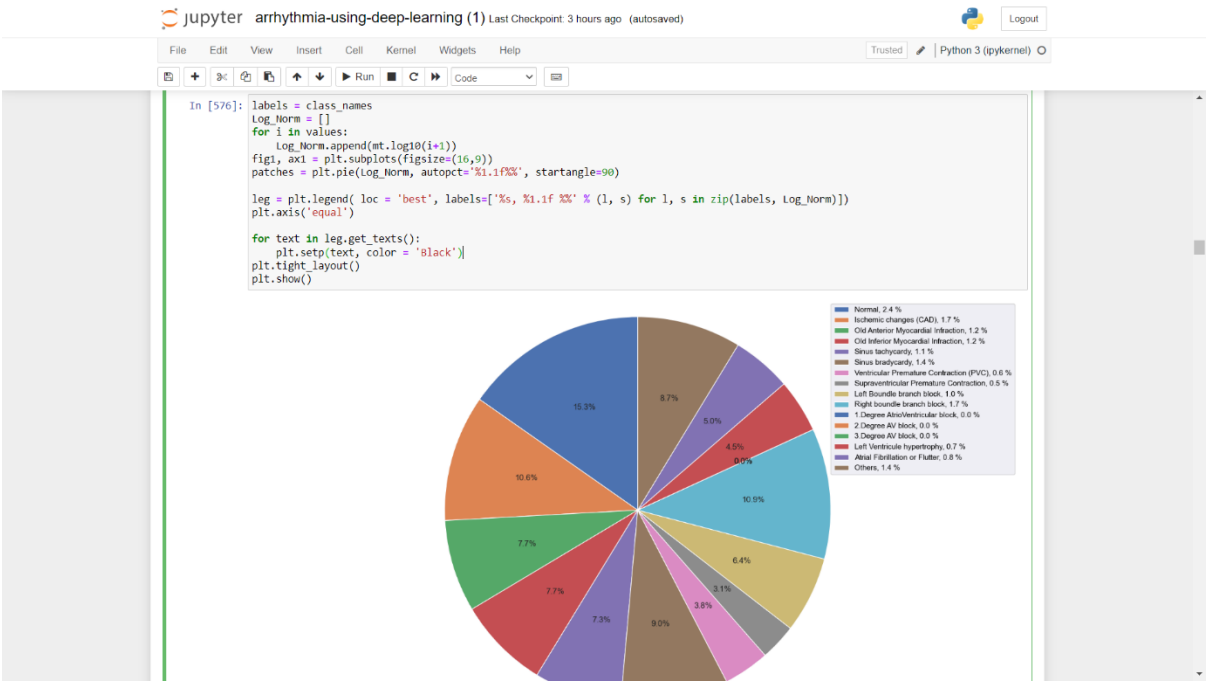
Out[563]: 0

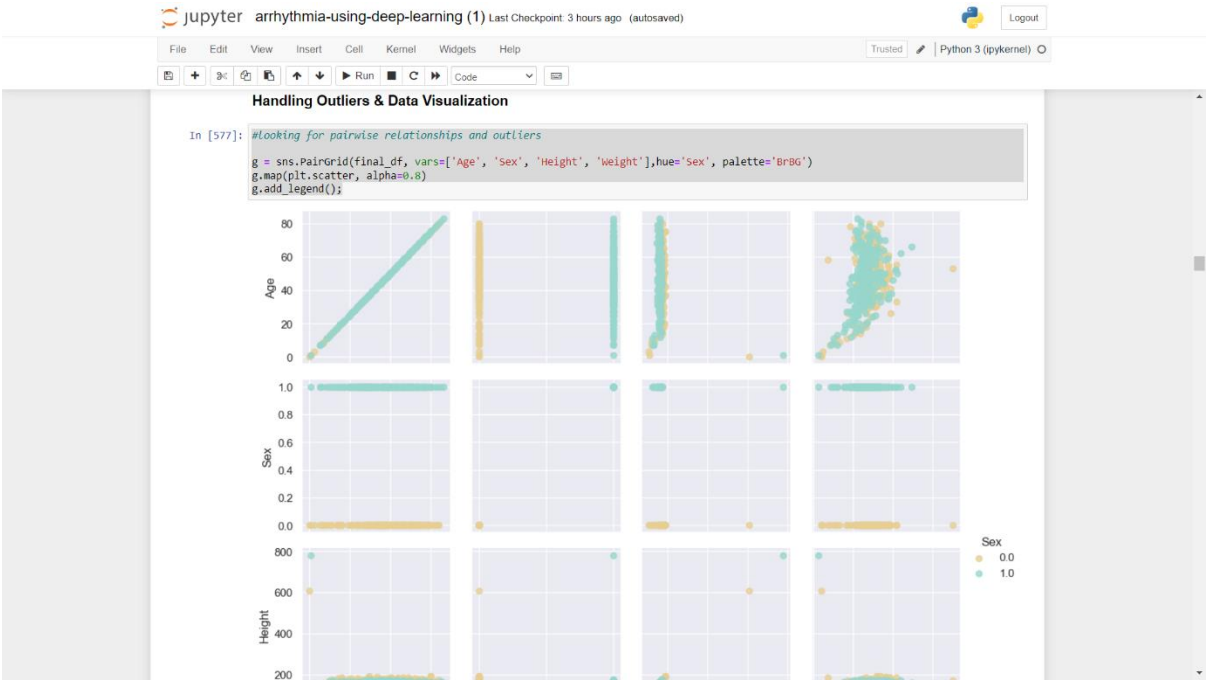
Generating final dataset

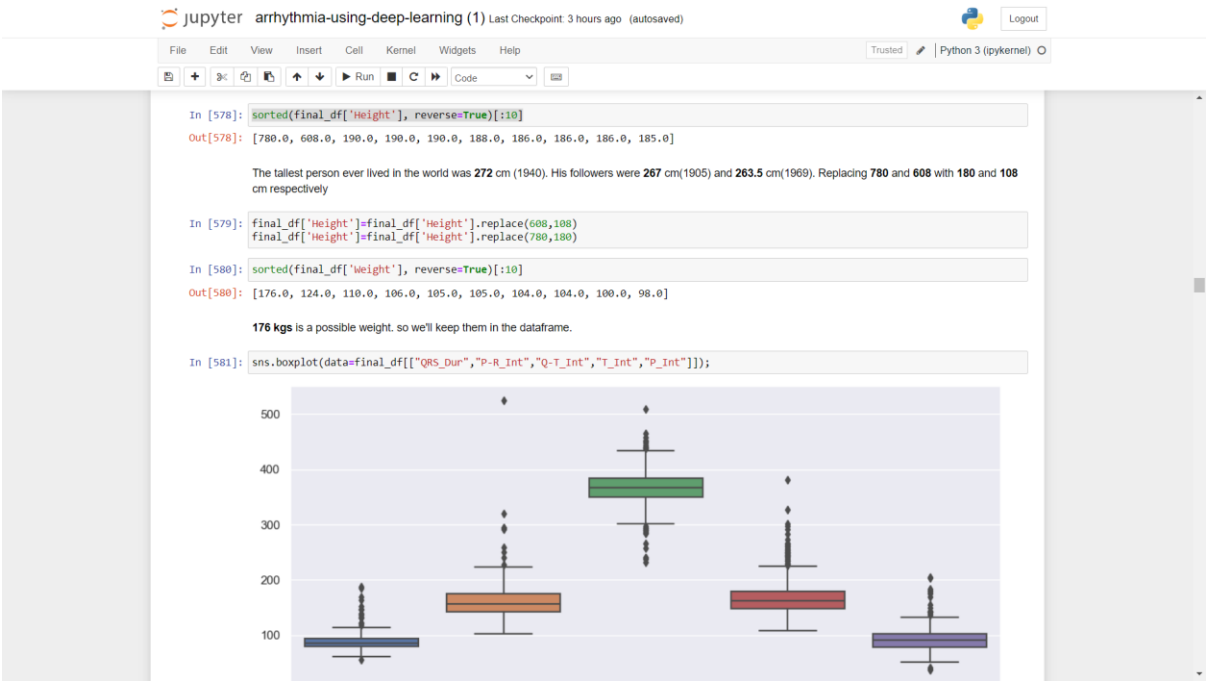
In [564]:
```
#Creating column names

final_df_columns=["Age","Sex","Height","Weight","QRS_Dur",
"P-R_Int","Q-T_Int","T_Int","P_Int","QRS","T","P","J","Heart_Rate",
"Q_Wave","R_Wave","S_Wave","R'_Wave","s'_Wave","Int_Def","Rag_R_Nom",
"Diph_R_Nom","Rag_P_Nom","Diph_P_Nom","Rag_T_Nom","Diph_T_Nom",
"DII00", "DII01","DII02", "DII03", "DII04","DII05","DII06","DII07","DII08","DII09","DII10","DII11",
"DIII00","DIII01","DIII02", "DIII03", "DIII04","DIII05","DIII06","DIII07","DIII08","DIII09","DIII10","DIII11",
"AVR00","AVR01","AVR02","AVR03","AVR04","AVR05","AVR06","AVR07","AVR08","AVR09","AVR10","AVR11",
"AVL00","AVL01","AVL02","AVL03","AVL04","AVL05","AVL06","AVL07","AVL08","AVL09","AVL10","AVL11",
"AVF00","AVF01","AVF02","AVF03","AVF04","AVF05","AVF06","AVF07","AVF08","AVF09","AVF10","AVF11",
"V100","V101","V102","V103","V104","V105","V106","V107","V108","V109","V110","V111",
"V200","V201","V202","V203","V204","V205","V206","V207","V208","V209","V210","V211",
"V300","V301","V302","V303","V304","V305","V306","V307","V308","V309","V310","V311",
"V400","V401","V402","V403","V404","V405","V406","V407","V408","V409","V410","V411",
"V500","V501","V502","V503","V504","V505","V506","V507","V508","V509","V510","V511",
"V600","V601","V602","V603","V604","V605","V606","V607","V608","V609","V610","V611",
"JJ_Wave","Amp_Q_Wave","Amp_R_Wave","Amp_S_Wave","R_Prime_Wave","S_Prime_Wave","P_Wave","T_Wave",
"QRSA","QRSTA","DII170","DII171","DII172","DII173","DII174","DII175","DII176","DII177","DII178","DII179",
"DIII180","DIII181","DIII182","DIII183","DIII184","DIII185","DIII186","DIII187","DIII188","DIII189",
"AVR190","AVR191","AVR192","AVR193","AVR194","AVR195","AVR196","AVR197","AVR198","AVR199",
"AVL200","AVL201","AVL202","AVL203","AVL204","AVL205","AVL206","AVL207","AVL208","AVL209",
"AVF210","AVF211","AVF212","AVF213","AVF214","AVF215","AVF216","AVF217","AVF218","AVF219",
"V1220","V1221","V1222","V1223","V1224","V1225","V1226","V1227","V1228","V1229",
"V2230","V2231","V2232","V2233","V2234","V2235","V2236","V2237","V2238","V2239",
"V3240","V3241","V3242","V3243","V3244","V3245","V3246","V3247","V3248","V3249",
"V4250","V4251","V4252","V4253","V4254","V4255","V4256","V4257","V4258","V4259",
"V5260","V5261","V5262","V5263","V5264","V5265","V5266","V5267","V5268","V5269",
"V6270","V6271","V6272","V6273","V6274","V6275","V6276","V6277","V6278","V6279","class"]
```

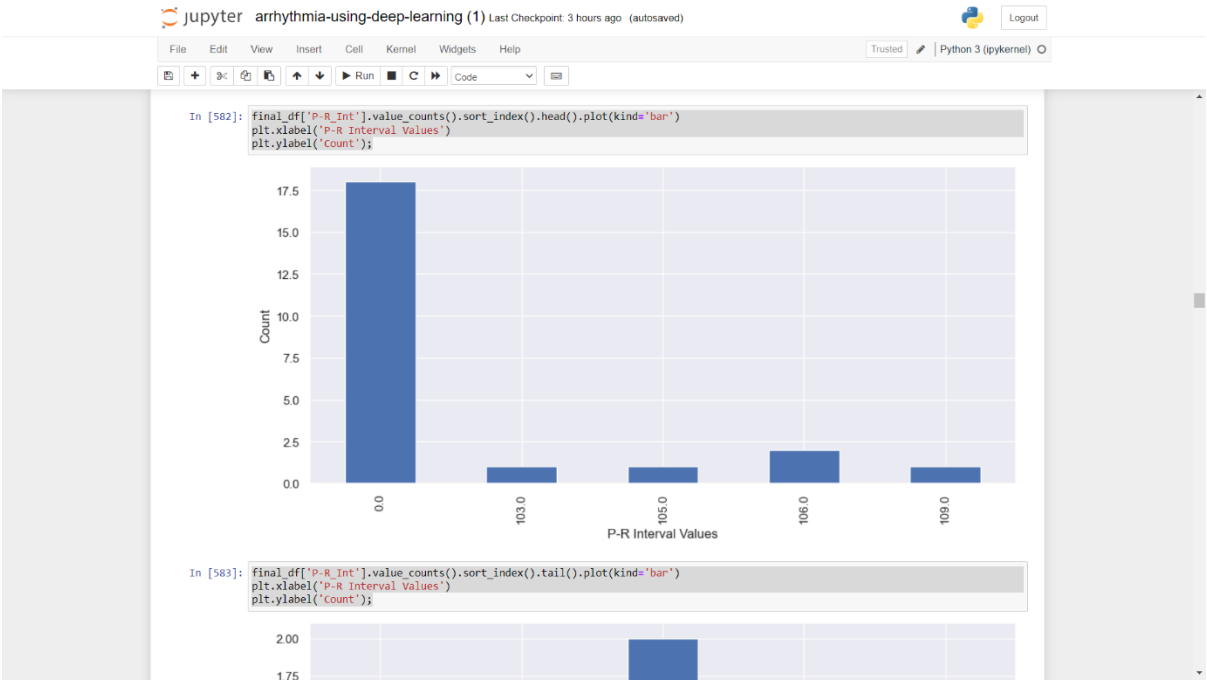
In [568]:
```
#Adding Column names to dataset

new_df.columns=final_df_columns
```

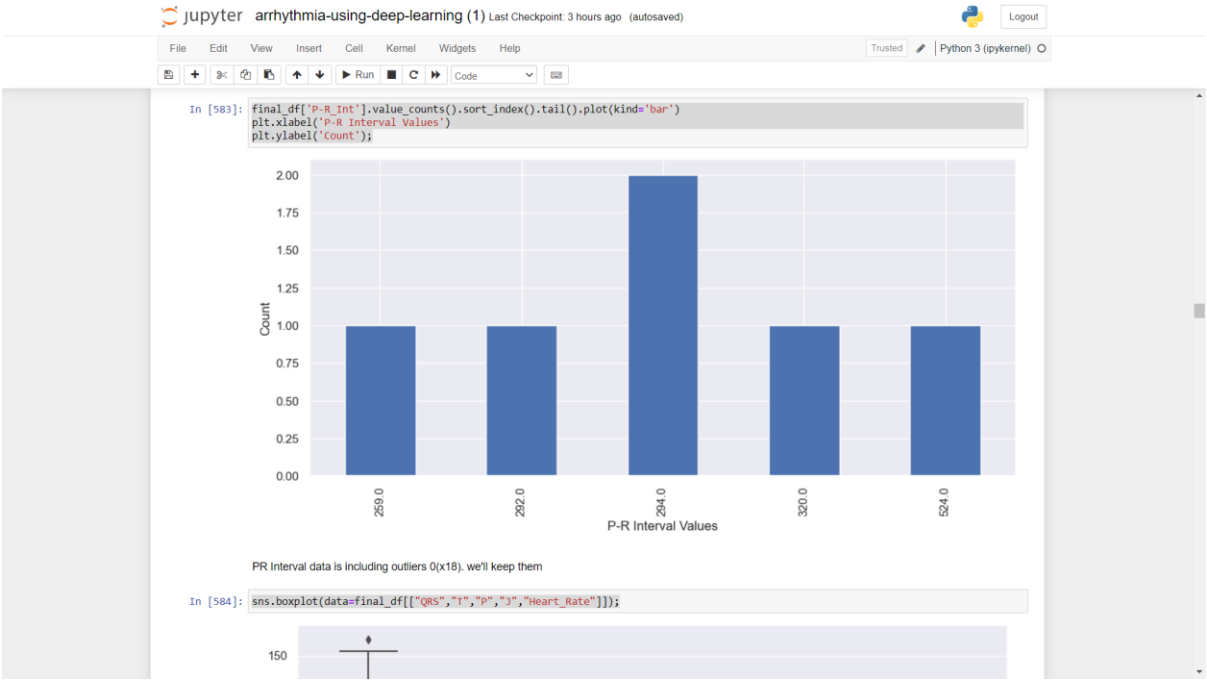

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Analyzing the dataset and check how many examples we have for each class:
we need to sort our dataset with respect to class attributes to count the number of instances available for each class
In [572]: t=new_df.sort_values(by=["class"])
In [573]: # Counting the number of instances for each class
la = t["class"].value_counts(sort=False).tolist()
la
Out[573]: [245, 44, 15, 15, 13, 25, 3, 2, 9, 50, 4, 5, 22]
In [574]: sns.countplot(x ='class',data =new_df)
plt.show()
count
250
200
150
100
50
0

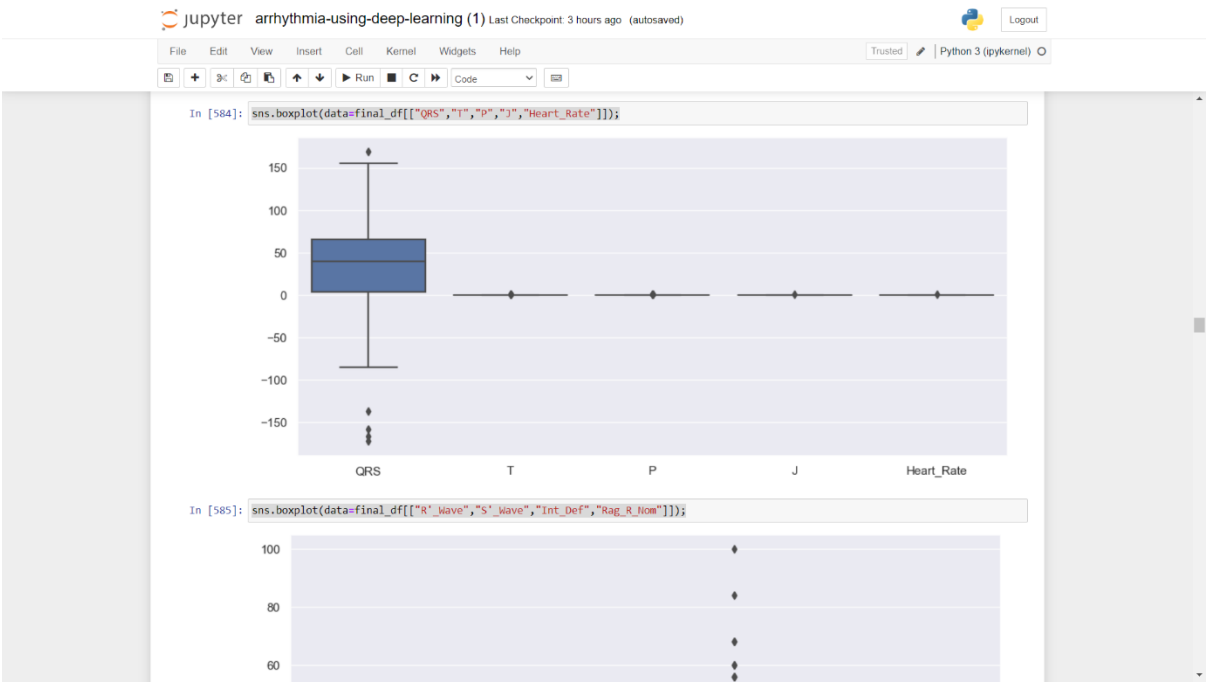

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [576]: labels = class_names
Log_Norm = []
for i in values:
Log_Norm.append(mt.log10(i+1))
fig1, ax1 = plt.subplots(figsize=(16,9))
patches = plt.pie(Log_Norm, autopct='%1.1f%%', startangle=90)
leg = plt.legend(loc = 'best', labels=['%s, %1.1f %%' % (l, s) for l, s in zip(labels, Log_Norm)])
plt.axis('equal')
for text in leg.get_texts():
plt.setp(text, color = 'Black')
plt.tight_layout()
plt.show()
Normal, 2.4 %
Ischemic changes (CAD), 1.7 %
Old Anterior Myocardial Infraction, 1.2 %
Old Inferior Myocardial Infraction, 1.2 %
Sinus tachycardy, 1.1 %
Sinus bradycardy, 1.4 %
Ventricular Premature Contraction (PVC), 0.6 %
Supraventricular Premature Contraction, 0.5 %
Left Boundle branch block, 1.0 %
Right boundle branch block, 1.7 %
1.Degree Atrio/Ventricular block, 0.0 %
2.Degree AV block, 0.0 %
3.Degree AV block, 0.0 %
Left Ventricule hypertrophy, 0.7 %
Atrial Fibrillation or Flutter, 0.8 %
Others, 1.4 %
15.3%
8.7%
5.0%
4.5%
10.6%
10.9%
7.7%
6.4%
7.7%
3.1%
7.3%
3.8%
9.0%

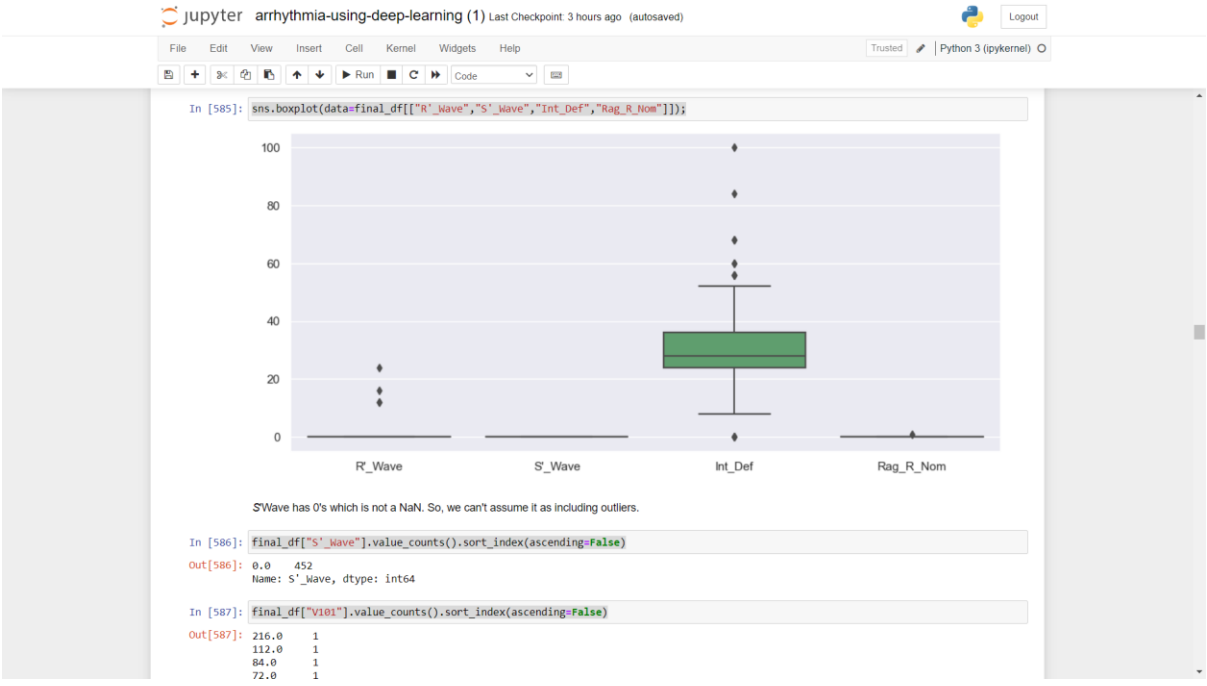

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Handling Outliers & Data Visualization
In [577]:
#looking for pairwise relationships and outliers
g = sns.PairGrid(final_df, vars=['Age', 'Sex', 'Height', 'Weight'],hue='Sex', palette='BrBG')
g.map(plt.scatter, alpha=0.8)
g.add_legend();
Age
Sex
Height
Sex
0.0
1.0

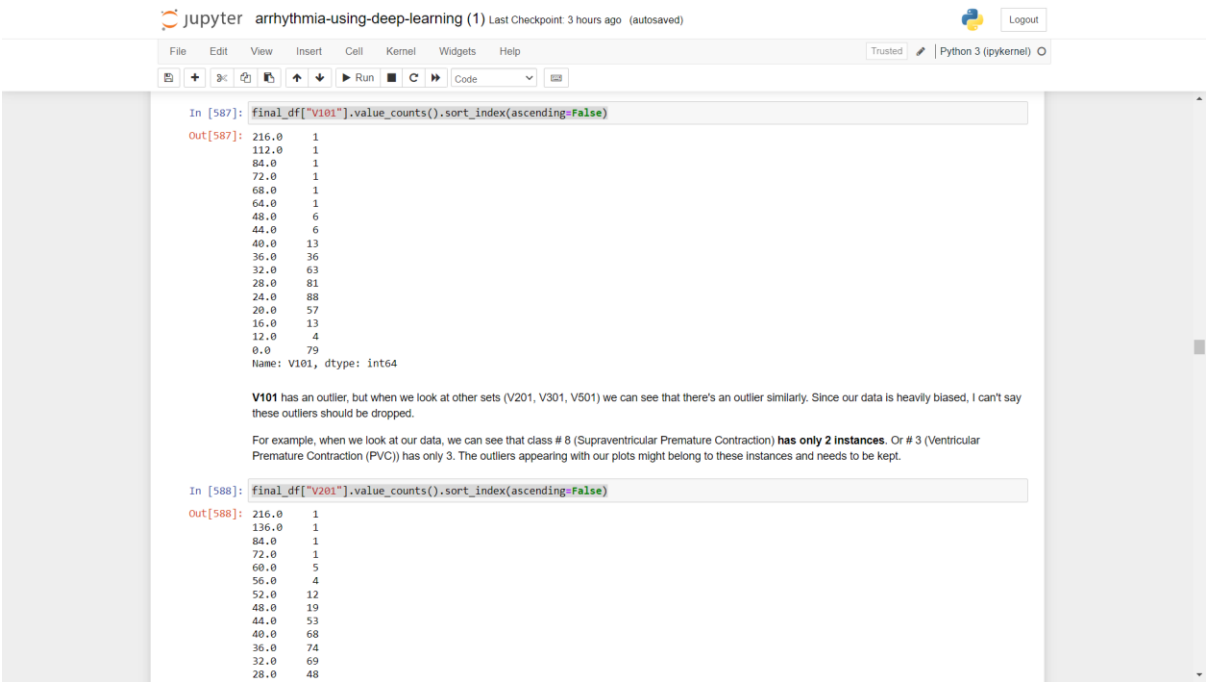

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [578]:
sorted(final_df['Height'], reverse=True)[:10]
Out[578]:
[780.0, 608.0, 190.0, 190.0, 190.0, 188.0, 186.0, 186.0, 186.0, 185.0]
The tallest person ever lived in the world was 272 cm (1940). His followers were 267 cm(1905) and 263.5 cm(1969). Replacing 780 and 608 with 180 and 108 cm respectively
In [579]:
final_df['Height']=final_df['Height'].replace(608,108)
final_df['Height']=final_df['Height'].replace(780,180)
In [580]:
sorted(final_df['Weight'], reverse=True)[:10]
Out[580]:
[176.0, 124.0, 110.0, 106.0, 105.0, 105.0, 104.0, 104.0, 100.0, 98.0]
176 kgs is a possible weight. so we'll keep them in the dataframe.
In [581]:
sns.boxplot(data=final_df[["QRS_Dur","P-R_Int","Q-T_Int","T_Int","P_Int"]]);

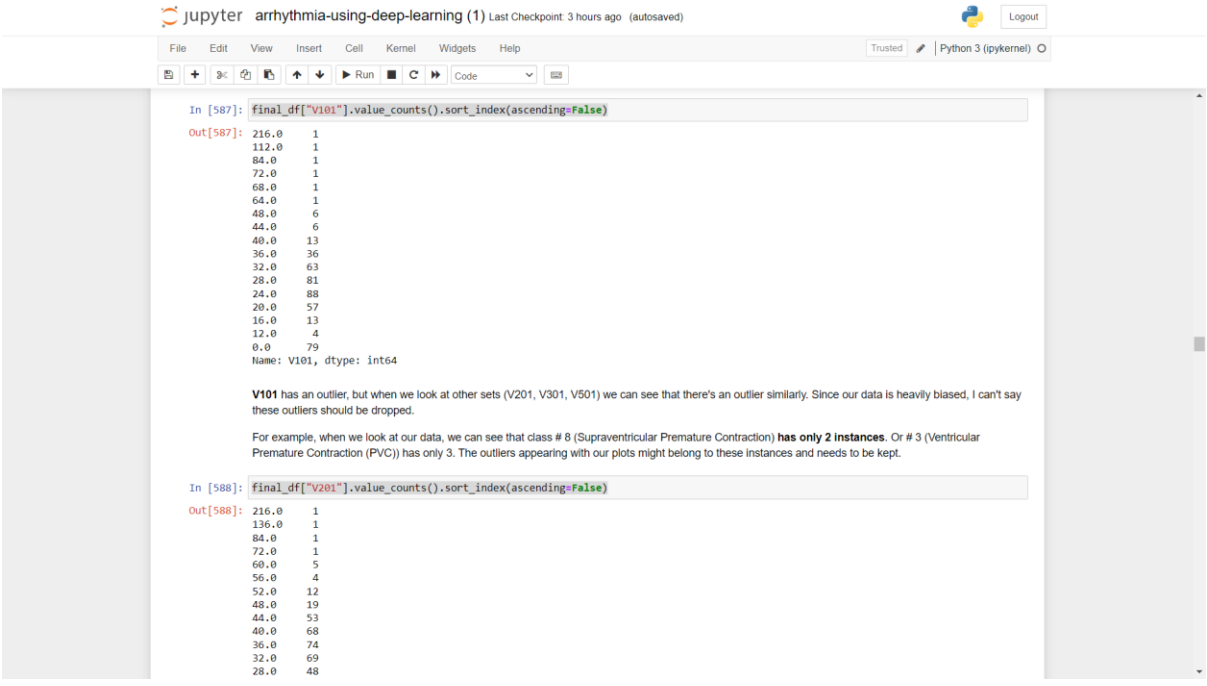

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
In [582]:
final_df['P-R_Int'].value_counts().sort_index().head().plot(kind='bar')
plt.xlabel('P-R Interval Values')
plt.ylabel('Count');
Count
P-R Interval Values
0.0
103.0
105.0
106.0
109.0
In [583]:
final_df['P-R_Int'].value_counts().sort_index().tail().plot(kind='bar')
plt.xlabel('P-R Interval Values')
plt.ylabel('Count');


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
In [583]:
final_df['P-R_Int'].value_counts().sort_index().tail().plot(kind='bar')
plt.xlabel('P-R Interval Values')
plt.ylabel('Count');
Count
P-R Interval Values
259.0
292.0
294.0
320.0
524.0
PR Interval data is including outliers 0(x18). we'll keep them
In [584]:
sns.boxplot(data=final_df[["QRS","T","P","J","Heart_Rate"]]);
150


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
In [584]: sns.boxplot(data=final_df[["QRS","T","P","J","Heart_Rate"]]);
150
100
50
0
−50
−100
−150
QRS
T
P
J
Heart_Rate
In [585]: sns.boxplot(data=final_df[["R'_Wave","S'_Wave","Int_Def","Rag_R_Nom"]]);
100
80
60


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
In [585]: sns.boxplot(data=final_df[["R'_Wave","S'_Wave","Int_Def","Rag_R_Nom"]]);
100
80
60
40
20
0
R'_Wave
S'_Wave
Int_Def
Rag_R_Nom
S'Wave has 0's which is not a NaN. So, we can't assume it as including outliers.
In [586]: final_df["S'_Wave"].value_counts().sort_index(ascending=False)
Out[586]: 0.0 452
Name: S'_Wave, dtype: int64
In [587]: final_df["V101"].value_counts().sort_index(ascending=False)
Out[587]: 216.0 1
112.0 1
84.0 1
72.0 1


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [587]: final_df["V101"].value_counts().sort_index(ascending=False)
Out[587]:
216.0 1
112.0 1
84.0 1
72.0 1
68.0 1
64.0 1
48.0 6
44.0 6
40.0 13
36.0 36
32.0 63
28.0 81
24.0 88
20.0 57
16.0 13
12.0 4
0.0 79
Name: V101, dtype: int64
V101 has an outlier, but when we look at other sets (V201, V301, V501) we can see that there's an outlier similarly. Since our data is heavily biased, I can't say these outliers should be dropped.
For example, when we look at our data, we can see that class # 8 (Supraventricular Premature Contraction) has only 2 instances. Or # 3 (Ventricular Premature Contraction (PVC)) has only 3. The outliers appearing with our plots might belong to these instances and needs to be kept.
In [588]: final_df["V201"].value_counts().sort_index(ascending=False)
Out[588]:
216.0 1
136.0 1
84.0 1
72.0 1
60.0 5
56.0 4
52.0 12
48.0 19
44.0 53
40.0 68
36.0 74
32.0 69
28.0 48



jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [589]:
```
final_df["V301"].value_counts().sort_index(ascending=False)
```

Out[589]:
```
132.0     1
88.0      2
76.0      2
72.0      1
68.0      2
64.0      3
60.0     16
56.0     26
52.0     57
48.0     68
44.0     76
40.0     74
36.0     49
32.0     32
28.0     13
24.0      4
20.0      6
16.0      1
0.0      19
Name: V301, dtype: int64
```

In [590]:
```
final_df["V501"].value_counts().sort_index(ascending=False)
```

Out[590]:
```
136.0      1
120.0      2
116.0      1
108.0      1
96.0       1
92.0       1
84.0       1
76.0       2
72.0       3
68.0       4
64.0       5
60.0       9
56.0      22
52.0      67
48.0     105
44.0     104
40.0      73
```

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

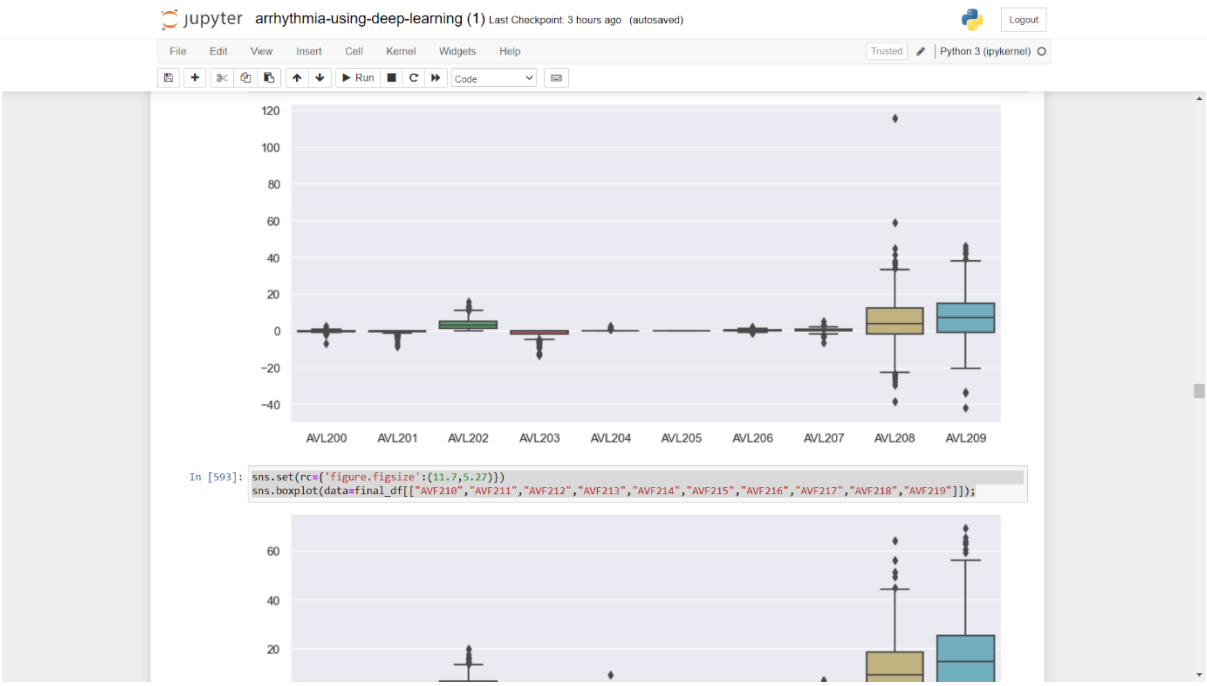
File Edit View Insert Cell Kernel Widgets Help

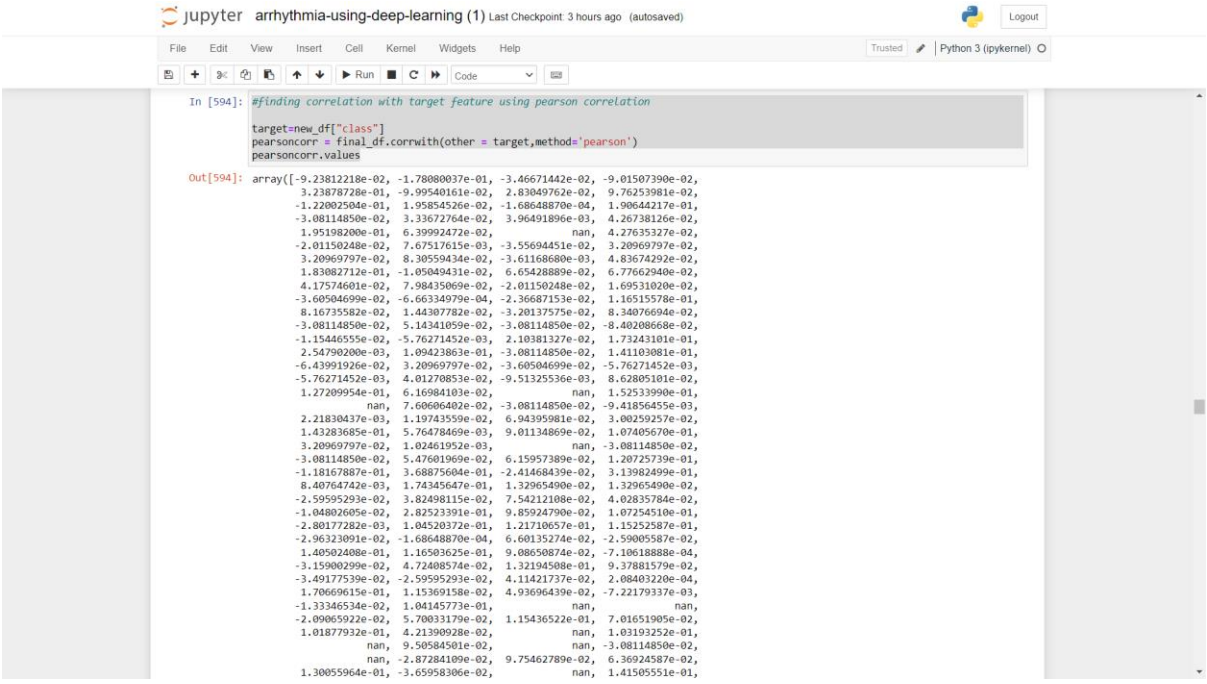
Trusted | Python 3 (ipykernel)

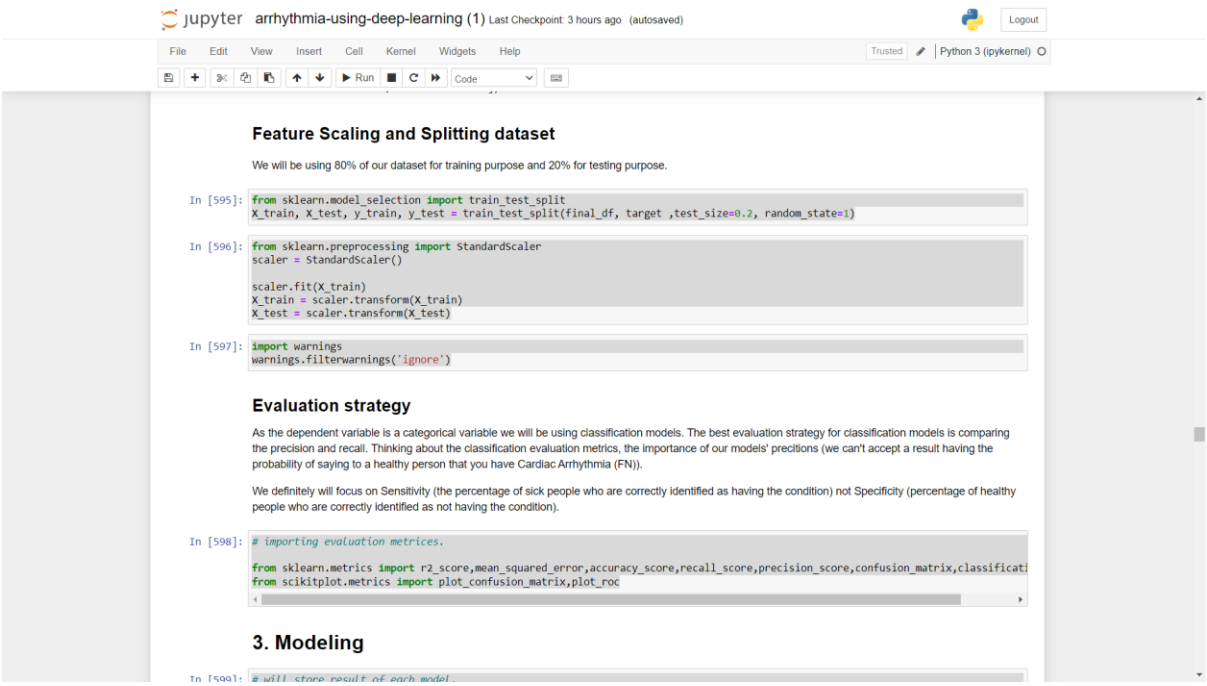
Run Code

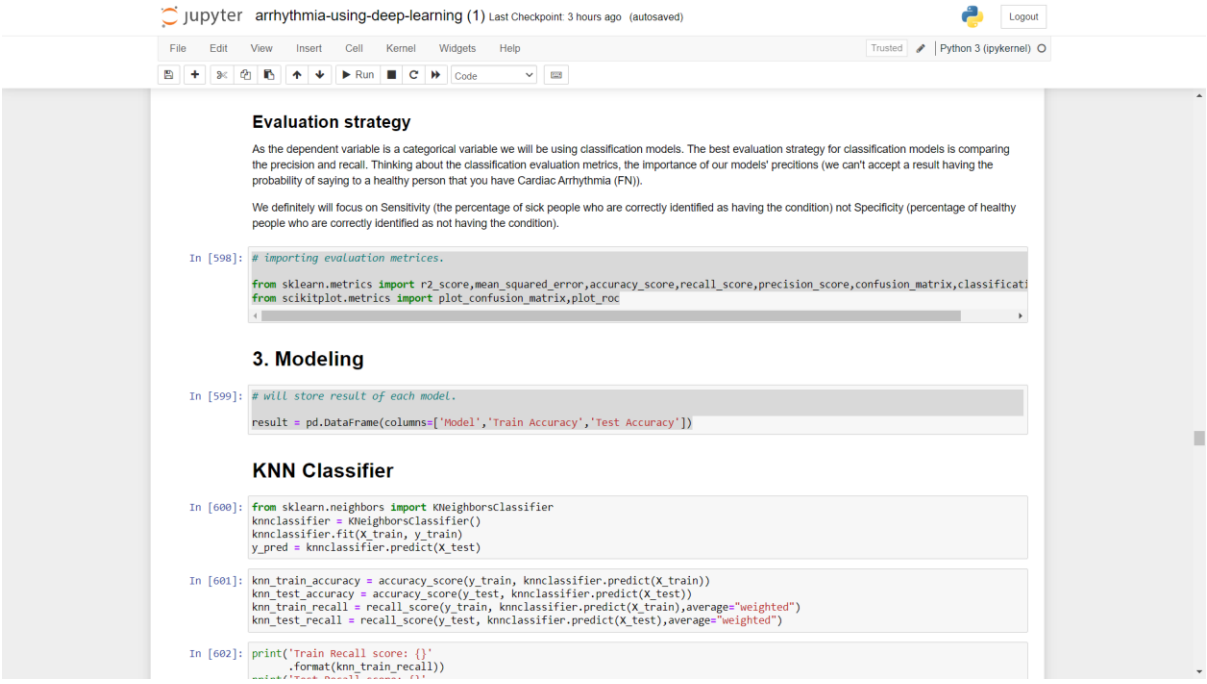
In [591]:
```
sns.set(rc={'figure.figsize':(11.7,5.27)})
sns.boxplot(data=final_df[["AVR190","AVR191","AVR192","AVR193","AVR194","AVR195","AVR196","AVR197","AVR198","AVR199"]]);
```

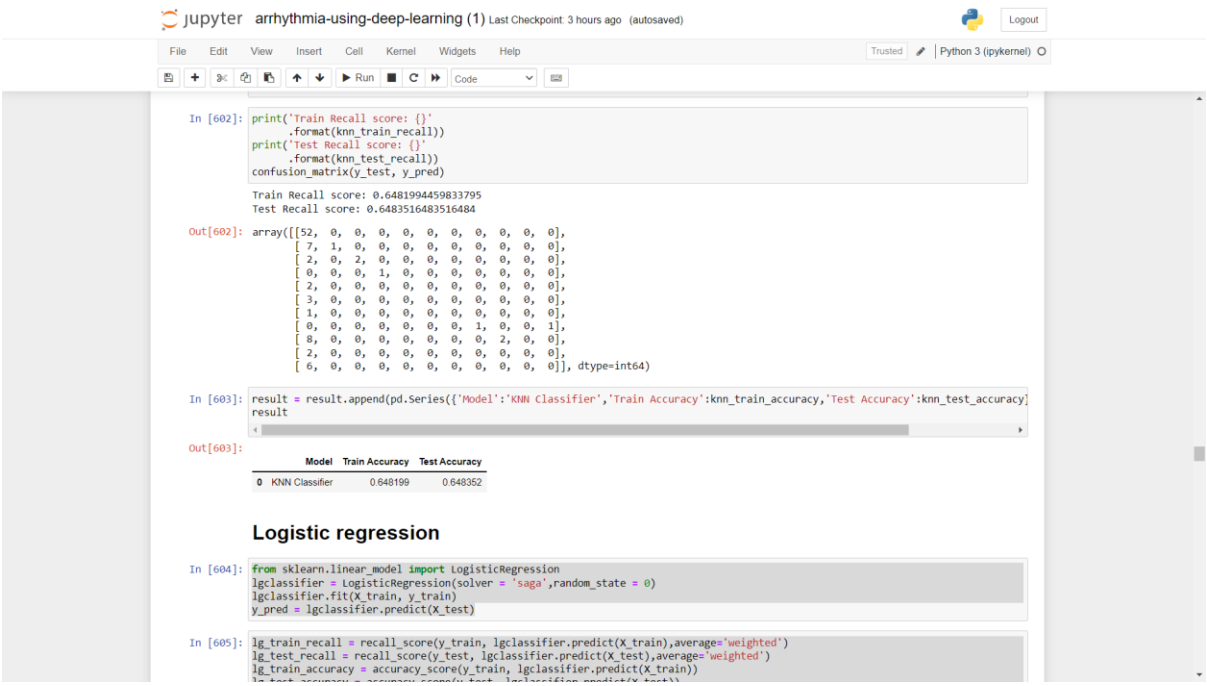
In [592]:
```
sns.set(rc={'figure.figsize':(11.7,5.27)})
sns.boxplot(data=final_df[["AVL200","AVL201","AVL202","AVL203","AVL204","AVL205","AVL206","AVL207","AVL208","AVL209"]]);
```

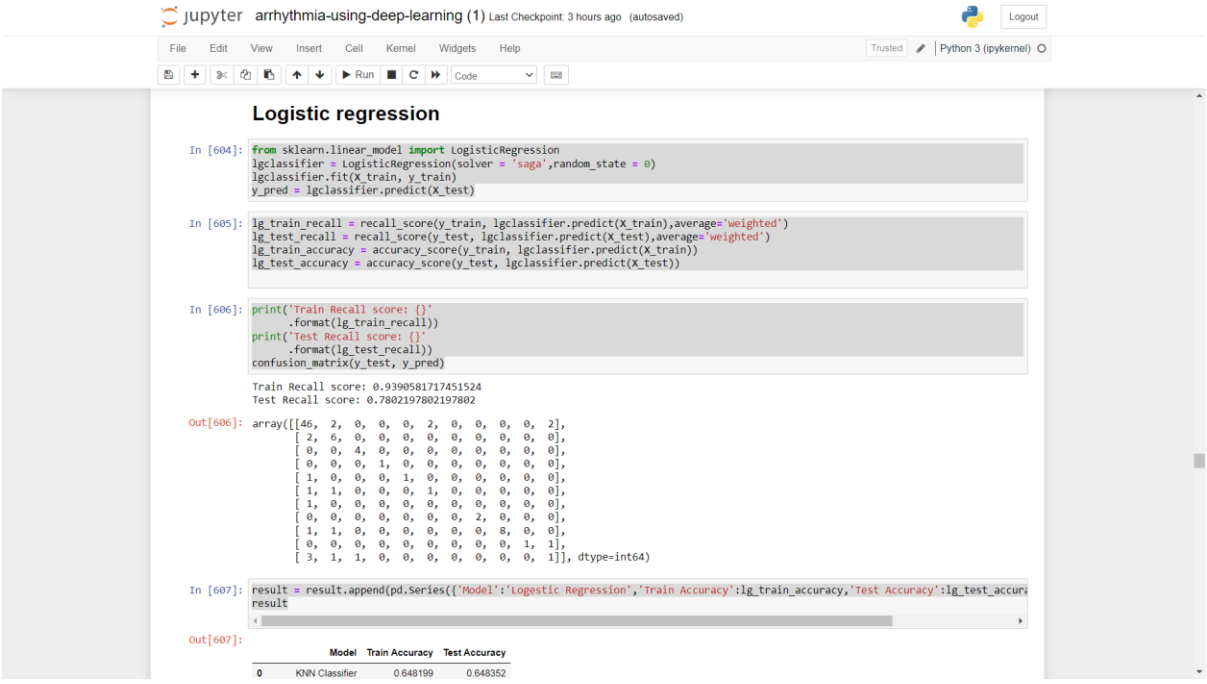

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
120
100
80
60
40
20
0
−20
−40
AVL200
AVL201
AVL202
AVL203
AVL204
AVL205
AVL206
AVL207
AVL208
AVL209
In [593]:
sns.set(rc={'figure.figsize':(11.7,5.27)})
sns.boxplot(data=final_df[["AVF210","AVF211","AVF212","AVF213","AVF214","AVF215","AVF216","AVF217","AVF218","AVF219"]]);
60
40
20

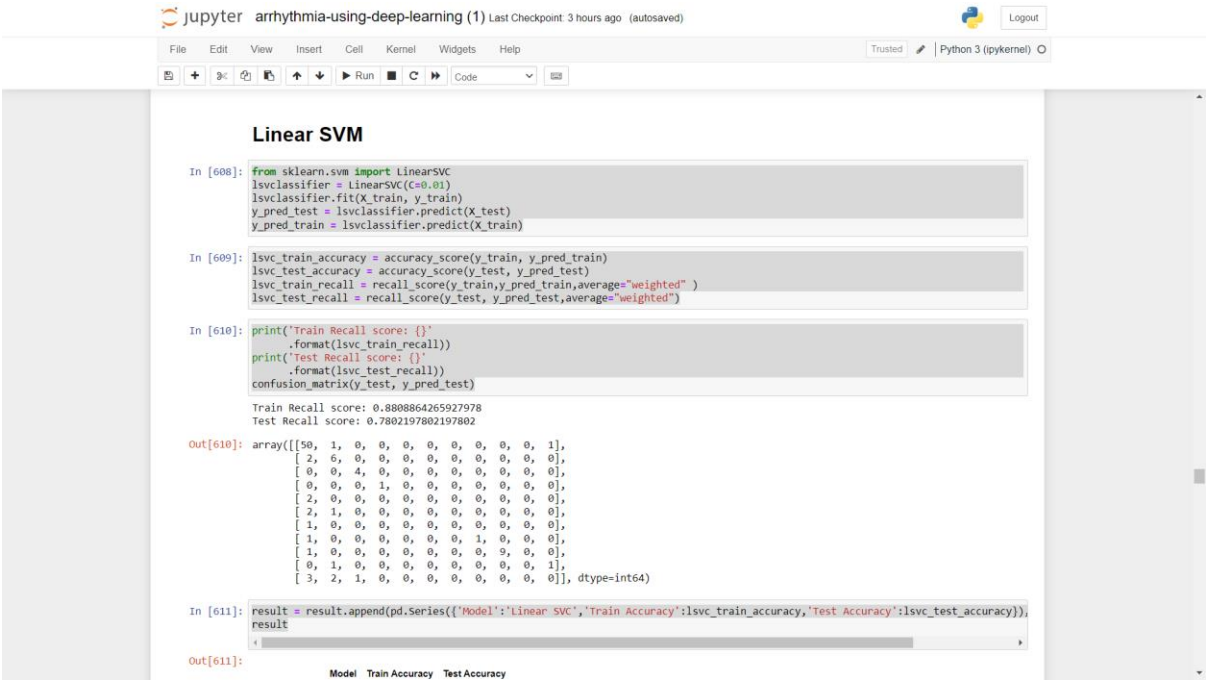

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [594]:
#finding correlation with target feature using pearson correlation
target=new_df["class"]
pearsoncorr = final_df.corrwith(other = target,method='pearson')
pearsoncorr.values
Out[594]:
array([-9.23812218e-02, -1.78080037e-01, -3.46671442e-02, -9.01507390e-02,
3.23878728e-01, -9.99540161e-02, 2.83049762e-02, 9.76253981e-02,
-1.22002504e-01, 1.95854526e-02, -1.68648870e-04, 1.90644217e-01,
-3.08114850e-02, 3.33672764e-02, 3.96491896e-03, 4.26738126e-02,
1.95198200e-01, 6.39992472e-02, nan, 4.27635327e-02,
-2.01150248e-02, 7.67517615e-03, -3.55694451e-02, 3.20969797e-02,
3.20969797e-02, 8.30559434e-02, -3.61168680e-03, 4.83674292e-02,
1.83082712e-01, -1.05049431e-02, 6.65428889e-02, 6.77662940e-02,
4.17574601e-02, 7.98435069e-02, -2.01150248e-02, 1.69531020e-02,
-3.60504699e-02, -6.66334979e-04, -2.36687153e-02, 1.16515578e-01,
8.16735582e-02, 1.44307782e-02, -3.20137575e-02, 8.34076694e-02,
-3.08114850e-02, 5.14341059e-02, -3.08114850e-02, -8.40208668e-02,
-1.15446555e-02, -5.76271452e-03, 2.10381327e-02, 1.73243101e-01,
2.54790200e-03, 1.09423863e-01, -3.08114850e-02, 1.41103081e-01,
-6.43991926e-02, 3.20969797e-02, -3.60504699e-02, -5.76271452e-03,
-5.76271452e-03, 4.01270853e-02, -9.51325536e-03, 8.62805101e-02,
1.27209954e-01, 6.16984103e-02, nan, 1.52533990e-01,
nan, 7.60606402e-02, -3.08114850e-02, -9.41856455e-03,
2.21830437e-03, 1.19743559e-02, 6.94395981e-02, 3.00259257e-02,
1.43283685e-01, 5.76478469e-03, 9.01134869e-02, 1.07405670e-01,
3.20969797e-02, 1.02461952e-03, nan, -3.08114850e-02,
-3.08114850e-02, 5.47601969e-02, 6.15957389e-02, 1.20725739e-01,
-1.18167887e-01, 3.68875604e-01, -2.41468439e-02, 3.13982499e-01,
8.40764742e-03, 1.74345647e-01, 1.32965490e-02, 1.32965490e-02,
-2.59595293e-02, 3.82498115e-02, 7.54212108e-02, 4.02835784e-02,
-1.04802605e-02, 2.82523391e-01, 9.85924790e-02, 1.07254510e-01,
-2.80177282e-03, 1.04520372e-01, 1.21710657e-01, 1.15252587e-01,
-2.96323091e-02, -1.68648870e-04, 6.60135274e-02, -2.59005587e-02,
1.40502408e-01, 1.16503625e-01, 9.08650874e-02, -7.10618888e-04,
-3.15900299e-02, 4.72408574e-02, 1.32194508e-01, 9.37881579e-02,
-3.49177539e-02, -2.59595293e-02, 4.11421737e-02, 2.08403220e-04,
1.70669615e-01, 1.15369158e-02, 4.93696439e-02, -7.22179337e-03,
-1.33346534e-02, 1.04145773e-01, nan, nan,
-2.09065922e-02, 5.70033179e-02, 1.15436522e-01, 7.01651905e-02,
1.01877932e-01, 4.21390928e-02, nan, 1.03193252e-01,
nan, 9.50584501e-02, nan, -3.08114850e-02,
nan, -2.87284109e-02, 9.75462789e-02, 6.36924587e-02,
1.30055964e-01, -3.65958306e-02, nan, 1.41505551e-01,


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
Feature Scaling and Splitting dataset
We will be using 80% of our dataset for training purpose and 20% for testing purpose.
In [595]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(final_df, target ,test_size=0.2, random_state=1)
In [596]:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
In [597]:
import warnings
warnings.filterwarnings('ignore')
Evaluation strategy
As the dependent variable is a categorical variable we will be using classification models. The best evaluation strategy for classification models is comparing the precision and recall. Thinking about the classification evaluation metrics, the importance of our models' precitions (we can't accept a result having the probability of saying to a healthy person that you have Cardiac Arrhythmia (FN)).
We definitely will focus on Sensitivity (the percentage of sick people who are correctly identified as having the condition) not Specificity (percentage of healthy people who are correctly identified as not having the condition).
In [598]:
# importing evaluation metrices.
from sklearn.metrics import r2_score,mean_squared_error,accuracy_score,recall_score,precision_score,confusion_matrix,classificati
from scikitplot.metrics import plot_confusion_matrix,plot_roc
3. Modeling
In [599]:
will store result of each model.


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
Evaluation strategy
As the dependent variable is a categorical variable we will be using classification models. The best evaluation strategy for classification models is comparing the precision and recall. Thinking about the classification evaluation metrics, the importance of our models' precitions (we can't accept a result having the probability of saying to a healthy person that you have Cardiac Arrhythmia (FN)).
We definitely will focus on Sensitivity (the percentage of sick people who are correctly identified as having the condition) not Specificity (percentage of healthy people who are correctly identified as not having the condition).
In [598]:
# importing evaluation metrices.
from sklearn.metrics import r2_score,mean_squared_error,accuracy_score,recall_score,precision_score,confusion_matrix,classificati
from scikitplot.metrics import plot_confusion_matrix,plot_roc
3. Modeling
In [599]:
# will store result of each model.
result = pd.DataFrame(columns=['Model','Train Accuracy','Test Accuracy'])
KNN Classifier
In [600]:
from sklearn.neighbors import KNeighborsClassifier
knnclassifier = KNeighborsClassifier()
knnclassifier.fit(X_train, y_train)
y_pred = knnclassifier.predict(X_test)
In [601]:
knn_train_accuracy = accuracy_score(y_train, knnclassifier.predict(X_train))
knn_test_accuracy = accuracy_score(y_test, knnclassifier.predict(X_test))
knn_train_recall = recall_score(y_train, knnclassifier.predict(X_train),average="weighted")
knn_test_recall = recall_score(y_test, knnclassifier.predict(X_test),average="weighted")
In [602]:
print('Train Recall score: {}'
.format(knn_train_recall))


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
In [602]:
print('Train Recall score: {}'
.format(knn_train_recall))
print('Test Recall score: {}'
.format(knn_test_recall))
confusion_matrix(y_test, y_pred)
Train Recall score: 0.6481994459833795
Test Recall score: 0.6483516483516484
Out[602]:
array([[52, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[7, 1, 0, 0, 0, 0, 0, 0, 0, 0],
[2, 0, 2, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[2, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[3, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 1],
[8, 0, 0, 0, 0, 0, 0, 0, 2, 0],
[2, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[6, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64)
In [603]:
result = result.append(pd.Series({'Model':'KNN Classifier','Train Accuracy':knn_train_accuracy,'Test Accuracy':knn_test_accuracy}
result
Out[603]:
Model Train Accuracy Test Accuracy
0 KNN Classifier 0.648199 0.648352
Logistic regression
In [604]:
from sklearn.linear_model import LogisticRegression
lgclassifier = LogisticRegression(solver = 'saga',random_state = 0)
lgclassifier.fit(X_train, y_train)
y_pred = lgclassifier.predict(X_test)
In [605]:
lg_train_recall = recall_score(y_train, lgclassifier.predict(X_train),average='weighted')
lg_test_recall = recall_score(y_test, lgclassifier.predict(X_test),average='weighted')
lg_train_accuracy = accuracy_score(y_train, lgclassifier.predict(X_train))


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
Logistic regression
In [604]:
from sklearn.linear_model import LogisticRegression
lgclassifier = LogisticRegression(solver = 'saga',random_state = 0)
lgclassifier.fit(X_train, y_train)
y_pred = lgclassifier.predict(X_test)
In [605]:
lg_train_recall = recall_score(y_train, lgclassifier.predict(X_train),average='weighted')
lg_test_recall = recall_score(y_test, lgclassifier.predict(X_test),average='weighted')
lg_train_accuracy = accuracy_score(y_train, lgclassifier.predict(X_train))
lg_test_accuracy = accuracy_score(y_test, lgclassifier.predict(X_test))
In [606]:
print('Train Recall score: {}'
.format(lg_train_recall))
print('Test Recall score: {}'
.format(lg_test_recall))
confusion_matrix(y_test, y_pred)
Train Recall score: 0.9390581717451524
Test Recall score: 0.7802197802197802
Out[606]:
array([[46, 2, 0, 0, 0, 2, 0, 0, 0, 0, 2],
[2, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0],
[1, 1, 0, 0, 0, 0, 0, 0, 8, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
[3, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1]], dtype=int64)
In [607]:
result = result.append(pd.Series({'Model':'Logestic Regression','Train Accuracy':lg_train_accuracy,'Test Accuracy':lg_test_accura
result
Out[607]:
Model Train Accuracy Test Accuracy
0 KNN Classifier 0.648199 0.648352



jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

Linear SVM

In [608]:
```
from sklearn.svm import LinearSVC
lsvclassifier = LinearSVC(C=0.01)
lsvclassifier.fit(X_train, y_train)
y_pred_test = lsvclassifier.predict(X_test)
y_pred_train = lsvclassifier.predict(X_train)
```

In [609]:
```
lsvc_train_accuracy = accuracy_score(y_train, y_pred_train)
lsvc_test_accuracy = accuracy_score(y_test, y_pred_test)
lsvc_train_recall = recall_score(y_train,y_pred_train,average="weighted" )
lsvc_test_recall = recall_score(y_test, y_pred_test,average="weighted")
```

In [610]:
```
print('Train Recall score: {}'
      .format(lsvc_train_recall))
print('Test Recall score: {}'
      .format(lsvc_test_recall))
confusion_matrix(y_test, y_pred_test)
```

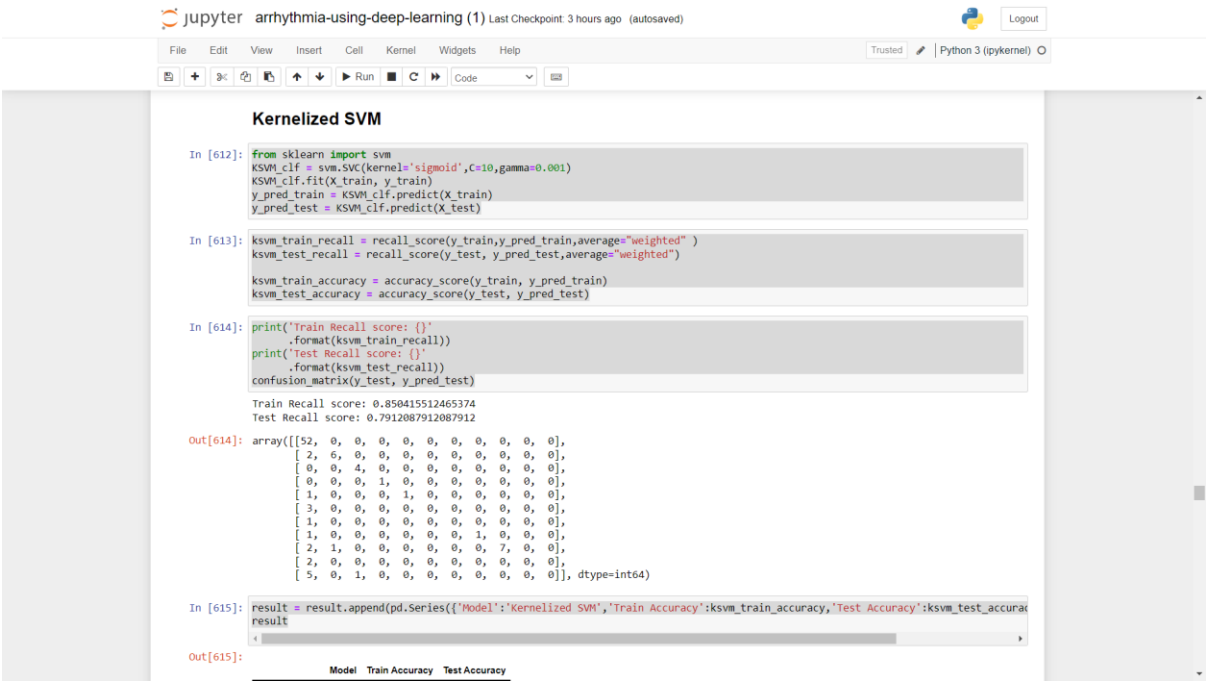
```
Train Recall score: 0.8808864265927978
Test Recall score: 0.7802197802197802
```

Out[610]:
```
array([[50,  1,  0,  0,  0,  0,  0,  0,  0,  1],
       [ 2,  6,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  4,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  1,  0,  0,  0,  0,  0,  0],
       [ 2,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 2,  1,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0,  1,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0,  0,  9,  0],
       [ 0,  1,  0,  0,  0,  0,  0,  0,  0,  1],
       [ 3,  2,  1,  0,  0,  0,  0,  0,  0,  0]], dtype=int64)
```

In [611]:
```
result = result.append(pd.Series({'Model':'Linear SVC','Train Accuracy':lsvc_train_accuracy,'Test Accuracy':lsvc_test_accuracy}),
result
```

Out[611]:

	Model	Train Accuracy	Test Accuracy



jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

Kernelized SVM

In [612]:
```
from sklearn import svm
KSVM_clf = svm.SVC(kernel='sigmoid',C=10,gamma=0.001)
KSVM_clf.fit(X_train, y_train)
y_pred_train = KSVM_clf.predict(X_train)
y_pred_test = KSVM_clf.predict(X_test)
```

In [613]:
```
ksvm_train_recall = recall_score(y_train,y_pred_train,average="weighted" )
ksvm_test_recall = recall_score(y_test, y_pred_test,average="weighted")

ksvm_train_accuracy = accuracy_score(y_train, y_pred_train)
ksvm_test_accuracy = accuracy_score(y_test, y_pred_test)
```

In [614]:
```
print('Train Recall score: {}'
      .format(ksvm_train_recall))
print('Test Recall score: {}'
      .format(ksvm_test_recall))
confusion_matrix(y_test, y_pred_test)
```

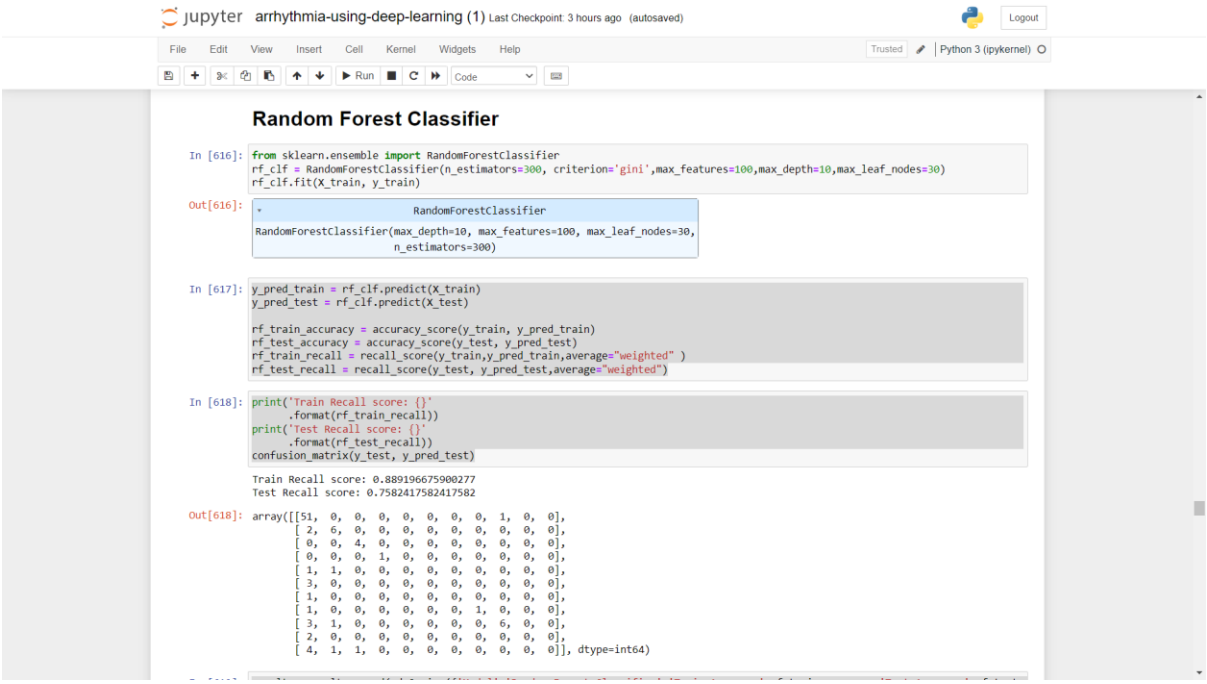
```
Train Recall score: 0.850415512465374
Test Recall score: 0.7912087912087912
```

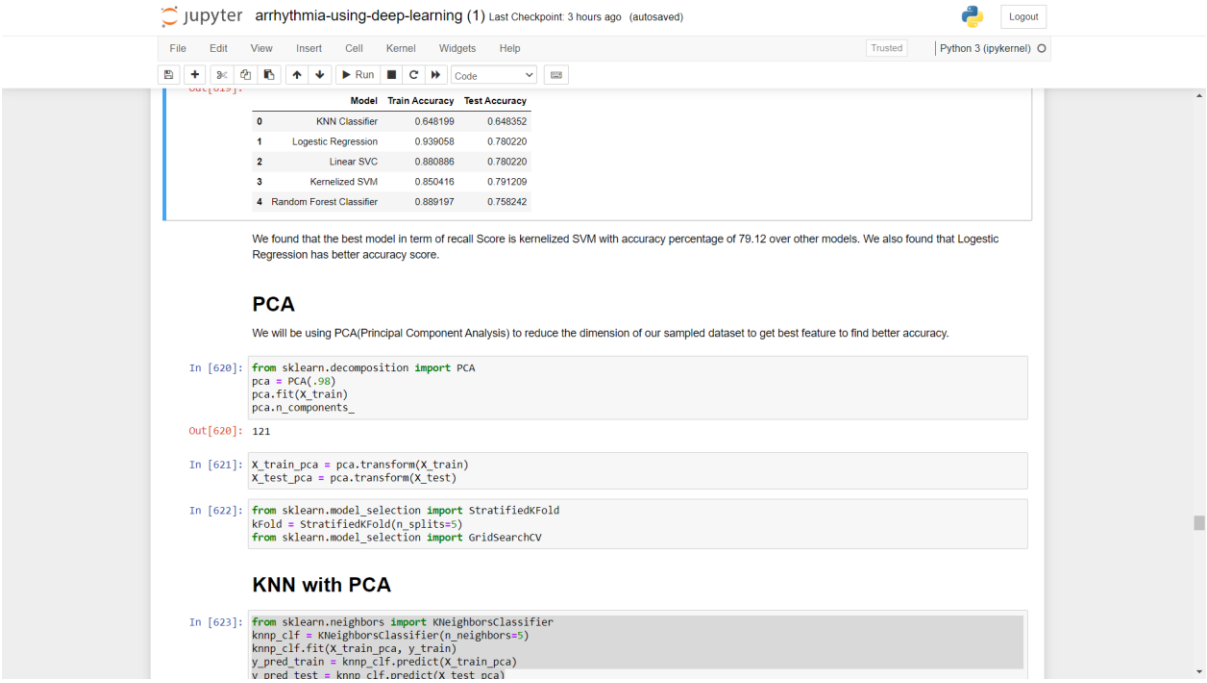
Out[614]:
```
array([[52,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 2,  6,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  4,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  1,  0,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 3,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0,  1,  0,  0],
       [ 2,  1,  0,  0,  0,  0,  0,  7,  0,  0],
       [ 2,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 5,  0,  1,  0,  0,  0,  0,  0,  0,  0]], dtype=int64)
```

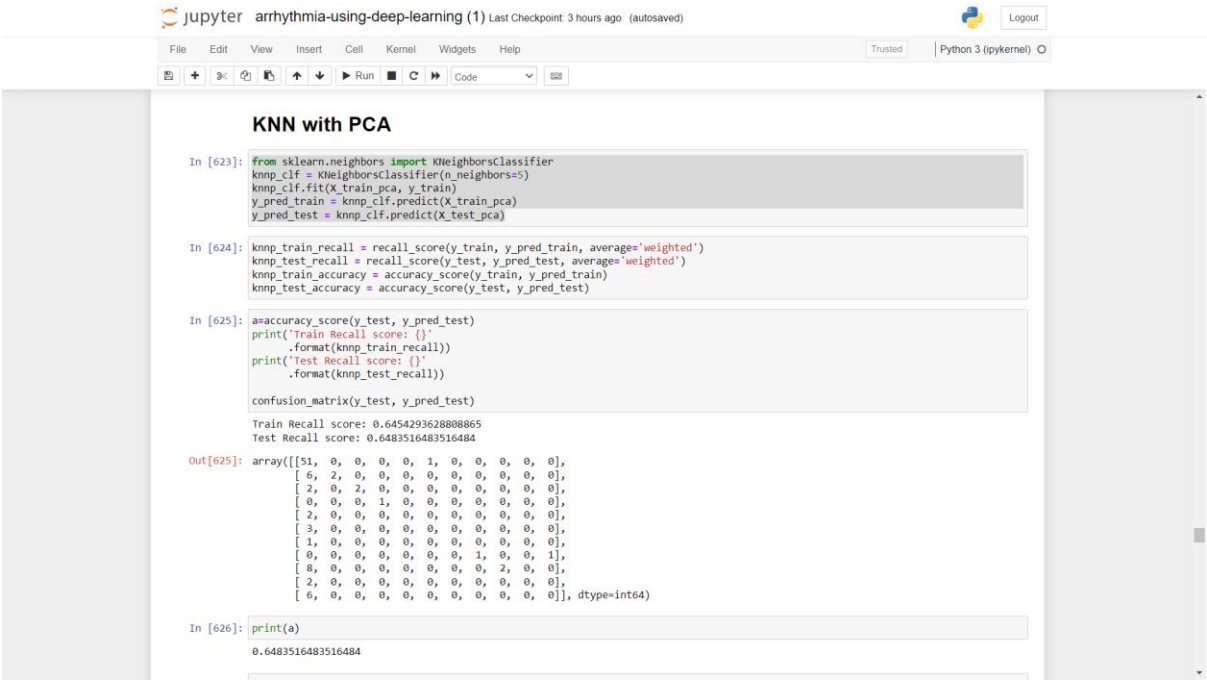
In [615]:
```
result = result.append(pd.Series({'Model':'Kernelized SVM','Train Accuracy':ksvm_train_accuracy,'Test Accuracy':ksvm_test_accurac
result
```

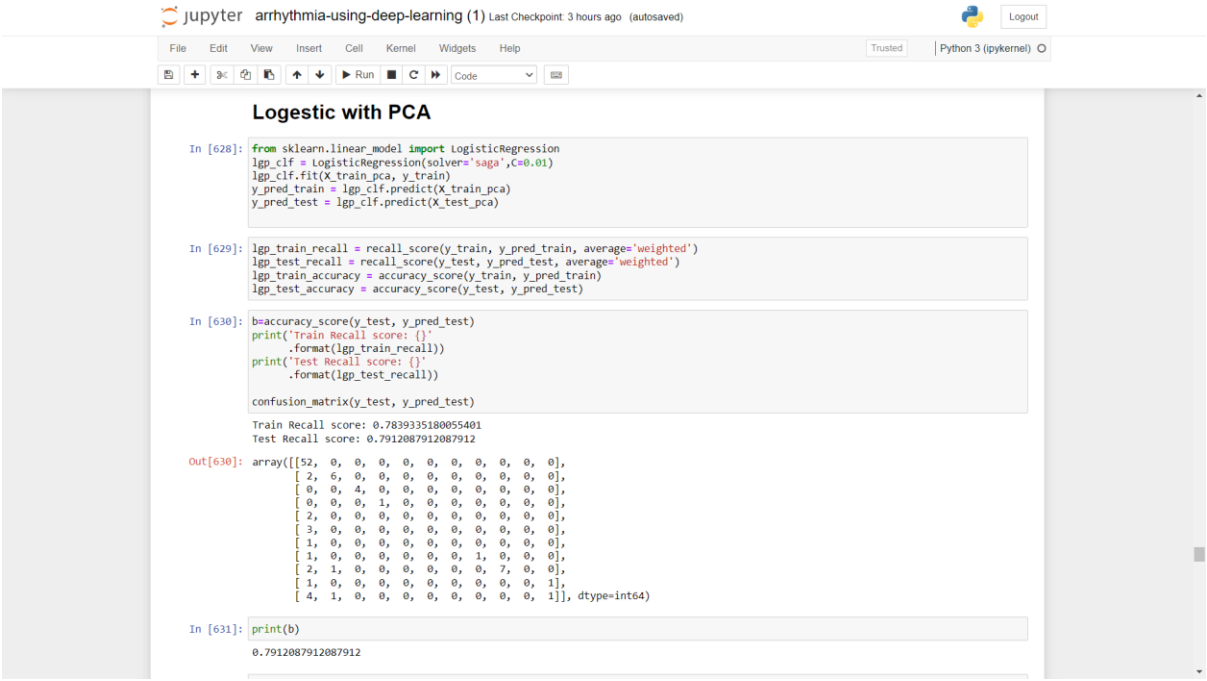
Out[615]:

	Model	Train Accuracy	Test Accuracy


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Random Forest Classifier
In [616]:
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators=300, criterion='gini',max_features=100,max_depth=10,max_leaf_nodes=30)
rf_clf.fit(X_train, y_train)
Out[616]:
RandomForestClassifier
RandomForestClassifier(max_depth=10, max_features=100, max_leaf_nodes=30,
n_estimators=300)
In [617]:
y_pred_train = rf_clf.predict(X_train)
y_pred_test = rf_clf.predict(X_test)
rf_train_accuracy = accuracy_score(y_train, y_pred_train)
rf_test_accuracy = accuracy_score(y_test, y_pred_test)
rf_train_recall = recall_score(y_train,y_pred_train,average="weighted")
rf_test_recall = recall_score(y_test, y_pred_test,average="weighted")
In [618]:
print('Train Recall score: {}'
.format(rf_train_recall))
print('Test Recall score: {}'
.format(rf_test_recall))
confusion_matrix(y_test, y_pred_test)
Train Recall score: 0.889196675900277
Test Recall score: 0.7582417582417582
Out[618]:
array([[51, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[2, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[3, 1, 0, 0, 0, 0, 0, 0, 6, 0, 0],
[2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[4, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64)


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Model Train Accuracy Test Accuracy
0 KNN Classifier 0.648199 0.648352
1 Logestic Regression 0.939058 0.780220
2 Linear SVC 0.880886 0.780220
3 Kernelized SVM 0.850416 0.791209
4 Random Forest Classifier 0.889197 0.758242
We found that the best model in term of recall Score is kernelized SVM with accuracy percentage of 79.12 over other models. We also found that Logestic Regression has better accuracy score.
PCA
We will be using PCA(Principal Component Analysis) to reduce the dimension of our sampled dataset to get best feature to find better accuracy.
In [620]:
from sklearn.decomposition import PCA
pca = PCA(.98)
pca.fit(X_train)
pca.n_components_
Out[620]:
121
In [621]:
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
In [622]:
from sklearn.model_selection import StratifiedKFold
kFold = StratifiedKFold(n_splits=5)
from sklearn.model_selection import GridSearchCV
KNN with PCA
In [623]:
from sklearn.neighbors import KNeighborsClassifier
knnp_clf = KNeighborsClassifier(n_neighbors=5)
knnp_clf.fit(X_train_pca, y_train)
y_pred_train = knnp_clf.predict(X_train_pca)


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
KNN with PCA
In [623]:
from sklearn.neighbors import KNeighborsClassifier
knnp_clf = KNeighborsClassifier(n_neighbors=5)
knnp_clf.fit(X_train_pca, y_train)
y_pred_train = knnp_clf.predict(X_train_pca)
y_pred_test = knnp_clf.predict(X_test_pca)
In [624]:
knnp_train_recall = recall_score(y_train, y_pred_train, average='weighted')
knnp_test_recall = recall_score(y_test, y_pred_test, average='weighted')
knnp_train_accuracy = accuracy_score(y_train, y_pred_train)
knnp_test_accuracy = accuracy_score(y_test, y_pred_test)
In [625]:
a=accuracy_score(y_test, y_pred_test)
print('Train Recall score: {}'
.format(knnp_train_recall))
print('Test Recall score: {}'
.format(knnp_test_recall))
confusion_matrix(y_test, y_pred_test)
Train Recall score: 0.6454293628808865
Test Recall score: 0.6483516483516484
Out[625]:
array([[51, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[6, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1],
[8, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0],
[2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64)
In [626]:
print(a)
0.6483516483516484


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Logestic with PCA
In [628]:
from sklearn.linear_model import LogisticRegression
lgp_clf = LogisticRegression(solver='saga',C=0.01)
lgp_clf.fit(X_train_pca, y_train)
y_pred_train = lgp_clf.predict(X_train_pca)
y_pred_test = lgp_clf.predict(X_test_pca)
In [629]:
lgp_train_recall = recall_score(y_train, y_pred_train, average='weighted')
lgp_test_recall = recall_score(y_test, y_pred_test, average='weighted')
lgp_train_accuracy = accuracy_score(y_train, y_pred_train)
lgp_test_accuracy = accuracy_score(y_test, y_pred_test)
In [630]:
b=accuracy_score(y_test, y_pred_test)
print('Train Recall score: {}'
.format(lgp_train_recall))
print('Test Recall score: {}'
.format(lgp_test_recall))
confusion_matrix(y_test, y_pred_test)
Train Recall score: 0.7839335180055401
Test Recall score: 0.7912087912087912
Out[630]:
array([[52, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[2, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[2, 1, 0, 0, 0, 0, 0, 0, 7, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
[4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1]], dtype=int64)
In [631]:
print(b)
0.7912087912087912

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [636]:
```
print(c)
```

```
0.8021978021978022
```

In [637]:
```
result = result.append(pd.Series({'Model':'Linear SVC with PCA','Train Accuracy':lsvcp_train_accuracy,'Test Accuracy':lsvcp_test_
result
```

Out[637]:

	Model	Train Accuracy	Test Accuracy
0	KNN Classifier	0.648199	0.648352
1	Logestic Regression	0.939058	0.780220
2	Linear SVC	0.880886	0.780220
3	Kernelized SVM	0.850416	0.791209
4	Random Forest Classifier	0.889197	0.758242
5	KNN Classifier with PCA	0.645429	0.648352
6	Logestic Regression with PCA	0.783934	0.791209
7	Linear SVC with PCA	0.808864	0.802198

Random Forest Classifier With PCA

In [638]:
```
from sklearn.ensemble import RandomForestClassifier
rfp_clf = RandomForestClassifier()
rfp_clf.fit(X_train_pca, y_train)
y_pred_train = rfp_clf.predict(X_train_pca)
y_pred_test = rfp_clf.predict(X_test_pca)
```

In [639]:
```
rfp_train_recall = recall_score(y_train, y_pred_train, average='weighted')
rfp_test_recall = recall_score(y_test, y_pred_test, average='weighted')
rfp_train_accuracy = accuracy_score(y_train, y_pred_train)
rfp_test_accuracy = accuracy_score(y_test, y_pred_test)
```

In [640]:
```
d=accuracy_score(y_test, y_pred_test)
print('Train Recall score: {}'
      .format(rfp_train_recall))
print('Test Recall score: {}'
```

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

Random Forest Classifier With PCA

In [638]:
```
from sklearn.ensemble import RandomForestClassifier
rfp_clf = RandomForestClassifier()
rfp_clf.fit(X_train_pca, y_train)
y_pred_train = rfp_clf.predict(X_train_pca)
y_pred_test = rfp_clf.predict(X_test_pca)
```

In [639]:
```
rfp_train_recall = recall_score(y_train, y_pred_train, average='weighted')
rfp_test_recall = recall_score(y_test, y_pred_test, average='weighted')
rfp_train_accuracy = accuracy_score(y_train, y_pred_train)
rfp_test_accuracy = accuracy_score(y_test, y_pred_test)
```

In [640]:
```
d=accuracy_score(y_test, y_pred_test)
print('Train Recall score: {}'
      .format(rfp_train_recall))
print('Test Recall score: {}'
      .format(rfp_test_recall))

confusion_matrix(y_test, y_pred_test)
```

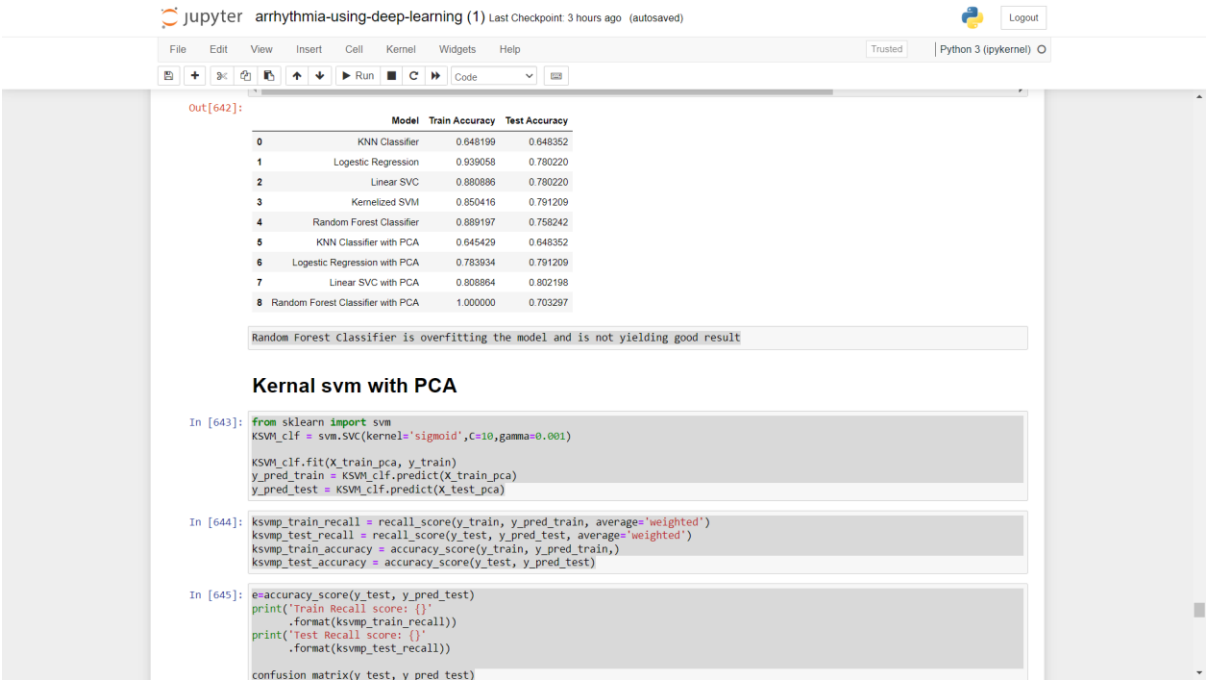
```
Train Recall score: 1.0
Test Recall score: 0.7032967032967034
```

Out[640]:
```
array([[52,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 3,  5,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 3,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 3,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  1,  0,  0,  0,  0,  0,  0,  1,  0,  0],
       [ 4,  0,  0,  0,  0,  0,  0,  0,  6,  0,  0],
       [ 1,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 6,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0]], dtype=int64)
```

In [641]:
```
print(d)
```

```
0.7032967032967034
```

In [642]:
```
result = result.append(pd.Series({'Model':'Random Forest Classifier with PCA','Train Accuracy':rfp_train_accuracy,'Test Accuracy'
```



Out[642]:

	Model	Train Accuracy	Test Accuracy
0	KNN Classifier	0.648199	0.648352
1	Logestic Regression	0.939058	0.780220
2	Linear SVC	0.880886	0.780220
3	Kernelized SVM	0.850416	0.791209
4	Random Forest Classifier	0.889197	0.758242
5	KNN Classifier with PCA	0.645429	0.648352
6	Logestic Regression with PCA	0.783934	0.791209
7	Linear SVC with PCA	0.808864	0.802198
8	Random Forest Classifier with PCA	1.000000	0.703297

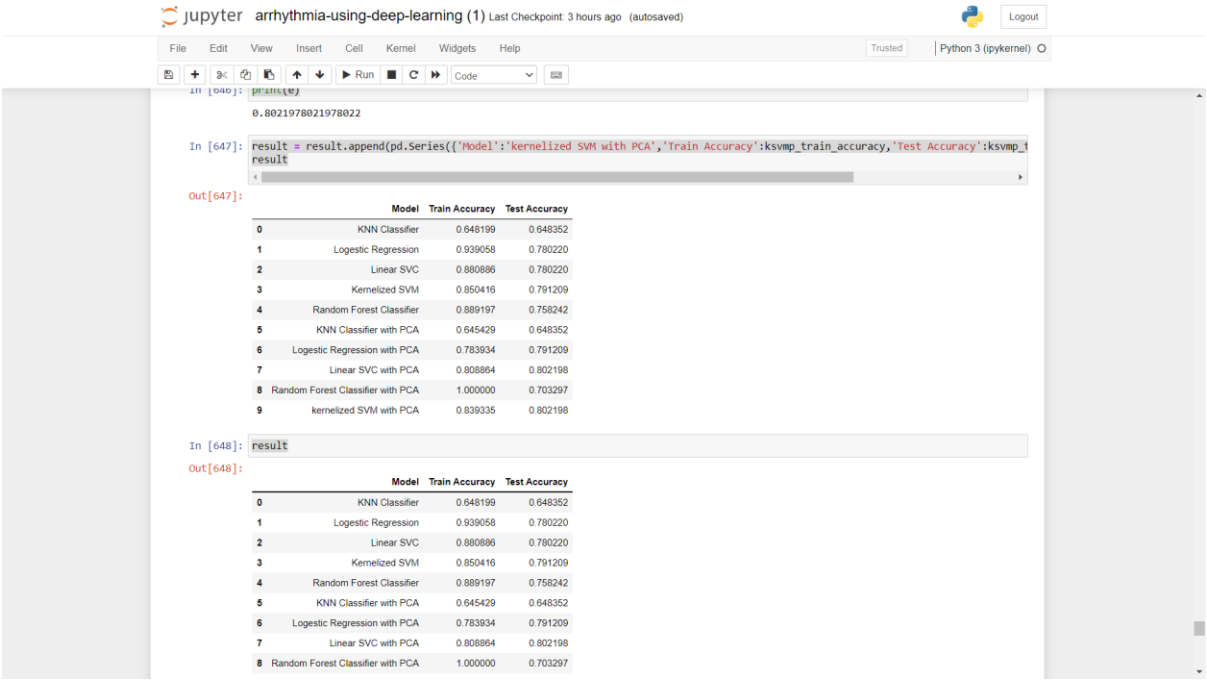
```
Random Forest Classifier is overfitting the model and is not yielding good result
```

Kernal svm with PCA

In [643]:
```
from sklearn import svm
KSVM_clf = svm.SVC(kernel='sigmoid',C=10,gamma=0.001)

KSVM_clf.fit(X_train_pca, y_train)
y_pred_train = KSVM_clf.predict(X_train_pca)
y_pred_test = KSVM_clf.predict(X_test_pca)
```

In [644]:
```
ksvmp_train_recall = recall_score(y_train, y_pred_train, average='weighted')
ksvmp_test_recall = recall_score(y_test, y_pred_test, average='weighted')
ksvmp_train_accuracy = accuracy_score(y_train, y_pred_train,)
ksvmp_test_accuracy = accuracy_score(y_test, y_pred_test)
```

In [645]:
```
e=accuracy_score(y_test, y_pred_test)
print('Train Recall score: {}'
      .format(ksvmp_train_recall))
print('Test Recall score: {}'
      .format(ksvmp_test_recall))

confusion_matrix(y_test, y_pred_test)
```



In [646]:
```
print(e)
```

```
0.8021978021978022
```

In [647]:
```
result = result.append(pd.Series({'Model':'kernelized SVM with PCA','Train Accuracy':ksvmp_train_accuracy,'Test Accuracy':ksvmp_t
result
```

Out[647]:

	Model	Train Accuracy	Test Accuracy
0	KNN Classifier	0.648199	0.648352
1	Logestic Regression	0.939058	0.780220
2	Linear SVC	0.880886	0.780220
3	Kernelized SVM	0.850416	0.791209
4	Random Forest Classifier	0.889197	0.758242
5	KNN Classifier with PCA	0.645429	0.648352
6	Logestic Regression with PCA	0.783934	0.791209
7	Linear SVC with PCA	0.808864	0.802198
8	Random Forest Classifier with PCA	1.000000	0.703297
9	kernelized SVM with PCA	0.839335	0.802198

In [648]:
```
result
```

Out[648]:

	Model	Train Accuracy	Test Accuracy
0	KNN Classifier	0.648199	0.648352
1	Logestic Regression	0.939058	0.780220
2	Linear SVC	0.880886	0.780220
3	Kernelized SVM	0.850416	0.791209
4	Random Forest Classifier	0.889197	0.758242
5	KNN Classifier with PCA	0.645429	0.648352
6	Logestic Regression with PCA	0.783934	0.791209
7	Linear SVC with PCA	0.808864	0.802198
8	Random Forest Classifier with PCA	1.000000	0.703297

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [648]:
```
result
```

Out[648]:

	Model	Train Accuracy	Test Accuracy
0	KNN Classifier	0.648199	0.648352
1	Logestic Regression	0.939058	0.780220
2	Linear SVC	0.880886	0.780220
3	Kernelized SVM	0.850416	0.791209
4	Random Forest Classifier	0.889197	0.758242
5	KNN Classifier with PCA	0.645429	0.648352
6	Logestic Regression with PCA	0.783934	0.791209
7	Linear SVC with PCA	0.808864	0.802198
8	Random Forest Classifier with PCA	1.000000	0.703297
9	kernelized SVM with PCA	0.839335	0.802198

In [649]:
```
print(a,b,c,d,e)
```

```
0.6483516483516484 0.7912087912087912 0.8021978021978022 0.7032967032967034 0.8021978021978022
```

Result

In [650]:
```
ax=result.plot(kind="bar",figsize=(20,5))
ax.set_xticks(result.index)
ax.set_xticklabels(result.Model,rotation=90)
plt.title('Train & Test Scores of Classifiers')
plt.xlabel('Models')
plt.ylabel('Accuracy Scores')
plt.legend(loc=4 , bbox_to_anchor=(1.2, 0))
plt.show();
```

jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)

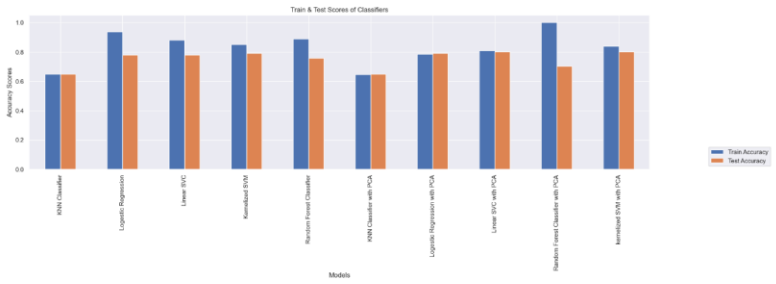
Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

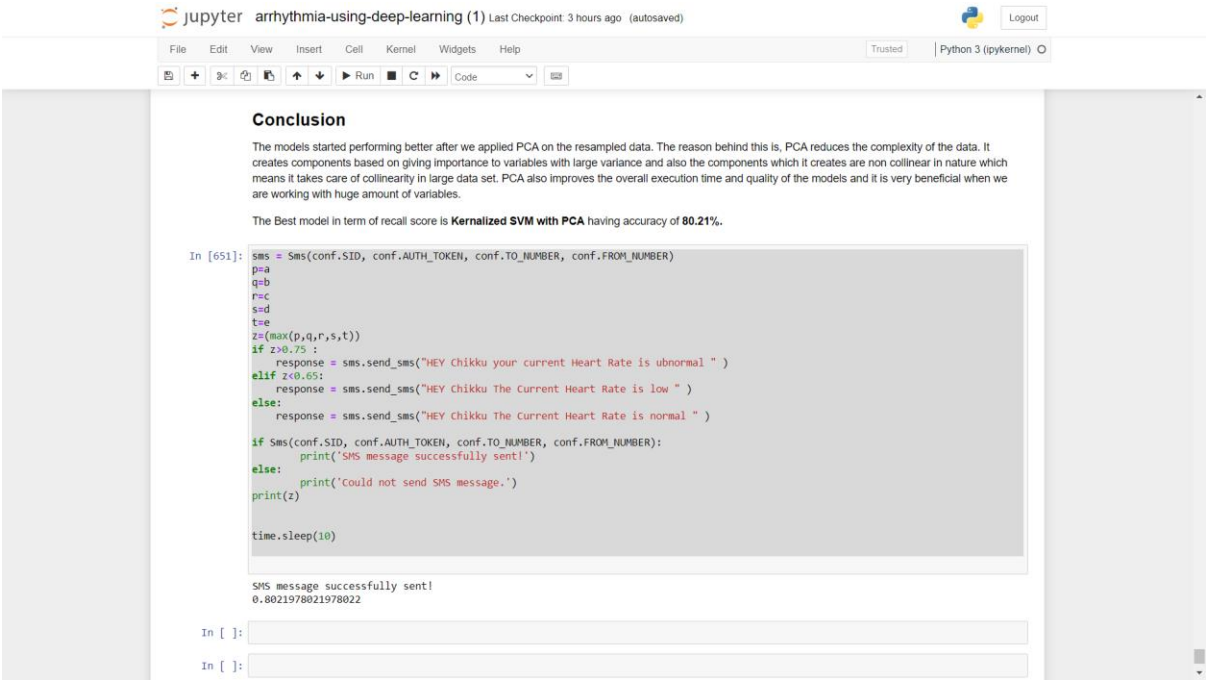
Result

In [650]:
```
ax=result.plot(kind="bar",figsize=(20,5))
ax.set_xticks(result.index)
ax.set_xticklabels(result.Model,rotation=90)
plt.title('Train & Test Scores of Classifiers')
plt.xlabel('Models')
plt.ylabel('Accuracy Scores')
plt.legend(loc=4 , bbox_to_anchor=(1.2, 0))
plt.show();
```

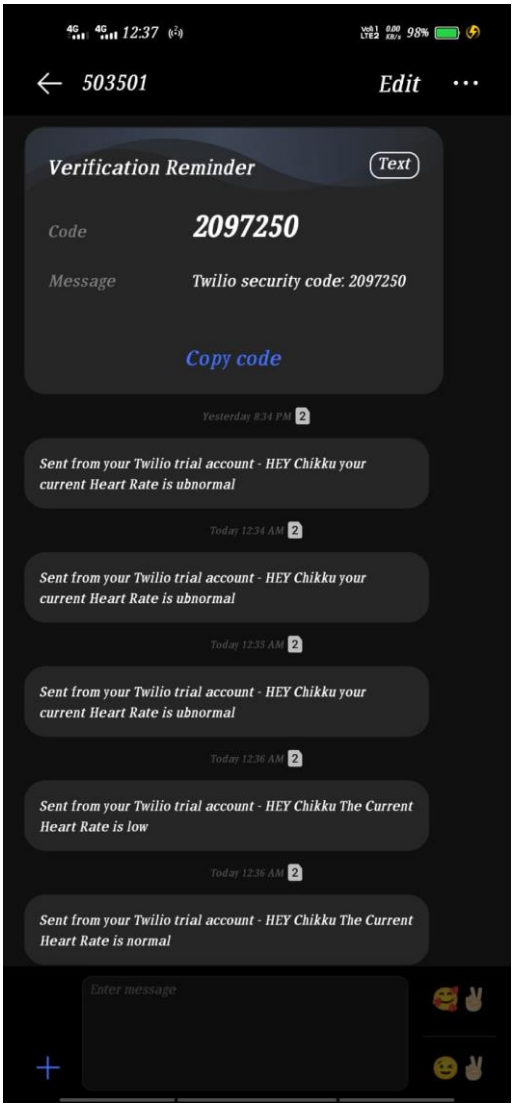


Conclusion

The models started performing better after we applied PCA on the resampled data. The reason behind this is, PCA reduces the complexity of the data. It creates components based on giving importance to variables with large variance and also the components which it creates are non collinear in nature which means it takes care of collinearity in large data set. PCA also improves the overall execution time and quality of the models and it is very beneficial when we are working with huge amount of variables.


jupyter arrhythmia-using-deep-learning (1) Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Conclusion
The models started performing better after we applied PCA on the resampled data. The reason behind this is, PCA reduces the complexity of the data. It creates components based on giving importance to variables with large variance and also the components which it creates are non collinear in nature which means it takes care of collinearity in large data set. PCA also improves the overall execution time and quality of the models and it is very beneficial when we are working with huge amount of variables.
The Best model in term of recall score is Kernalized SVM with PCA having accuracy of 80.21%.
In [651]:
sms = Sms(conf.SID, conf.AUTH_TOKEN, conf.TO_NUMBER, conf.FROM_NUMBER)
p=a
q=b
r=c
s=d
t=e
z=(max(p,q,r,s,t))
if z>0.75 :
response = sms.send_sms("HEY Chikku your current Heart Rate is ubnormal ")
elif z<0.65:
response = sms.send_sms("HEY Chikku The Current Heart Rate is low ")
else:
response = sms.send_sms("HEY Chikku The Current Heart Rate is normal ")
if Sms(conf.SID, conf.AUTH_TOKEN, conf.TO_NUMBER, conf.FROM_NUMBER):
print('SMS message successfully sent!')
else:
print('Could not send SMS message.')
print(z)
time.sleep(10)
SMS message successfully sent!
0.802197802197802
In [ ]:
In [ ]:

OUTPUT DISPLAYED IN MOBILE:



Chapter 10

REFERENCES

REFERENCES

[1] V. Mahesh, A. Kandaswamy, C. Vimal, and B. Sathish (2009) ECG Arrhythmia Classification Based on Logistic Model Tree

[2] Anam Mustaqeem, Syed Muhammad Anwar, and Muahammad Majid (2018) Multi-Class Classification of Cardiac Arrhythmia Using Feature Selection andSVM Invariants

[3] Babak Mohammadzadeh, Seyed Kamaledin Setarehdan, & Maryam Mohebbi(2008) Support Vector Machine – based arrhythmia classification using reduced features of heart rate variability signal

[4] Jalal A. Nasiri, Mahmoud Naghibzadeh, H. Sadoghi Yazdi, and Bahram Naghibzadeh (2009) ECG Arrhythmia Classification with Support Vector Machines and Genetic Algorithm