MULTIPLE ORGAN FAILURE DETECTION USING MACHINE LEARNING

Project report submitted for the partial fulfilment of the requirement for the degree of Bachelor of Technology

In

Electronics and Communication Engineering

By

Shashwat (161038)

Under the supervision of Mr. Pardeep Garg

Department of Electronics and Communication Engineering

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

CANDIDATE'S DECLARATION

I herewith affirm that the work offered in this report entitled **"MULTIPLE ORGAN FAILURE DETECTION USING MACHINE LEARNING"** in partial fulfilment of the necessities for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering submitted to the department of Electronics and Communication Engineering, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a time period of September 2019 to March 2020 under the supervision of Mr. Pardeep Garg (Assistant Professor(Grade-II)), Electronics and Communication Engineering.

The matter encased in this report has not been submitted for the reward of any other degree or diploma.

[illegible]

Shashwat (161038)

This is to certify that the above declaration made by the applicant is true to the best of my knowledge.

[illegible]

Mr. Pardeep Garg

Assistant Professor (Grade-II), Department of Electronics and Communication Engineering

ACKNOWLEDGEMENT

I would like to express my thankfulness to our teacher and mentor Mr. Pardeep Garg who gave me the opportunity to organize this project on the topic **MULTIPLE ORGAN FAILURE DETECTION USING MACHINE LEARNING**, which has helped me in a lot of research and I learnt about so many new perspectives. I am truly thankful to him.

Secondly, I would also like to acknowledge my lab assistants who helped me a lot in completing this project in less than the given time frame.

CONTENTS

S/No.	Title	Page.
1.	Abstract	3
2.	Aim and Objectives	4
3.	Chapter 1 - **INTRODUCTION**	5-8
	1.1 Introduction to The Project	
	1.2 Motivation	
	1.3 Organization of the Report	
4.	Chapter 2 – **LITERATURE REVIEW**	9-22
	2.1 How Machine Learning Works	
	2.2 Types of Machine Learning	
	2.3 Machine Learning Algorithms	
	2.4 Interpretation of Performance Measures	
	2.5 Deep Learning	
	2.6 Perceptrons	
	2.7 Loss Function	
	2.8 Problems in Supervised Learning	
5.	Chapter 3 – **SYSTEM DESIGN**	23-24
	3.1 Why Python and no other languages?	
	3.2 Why Jupyter?	
	3.3 SCIKIT-LEARN	
	3.4 PANDAS	
	3.5 KERAS	
	3.6 TENSORFLOW	
6.	Chapter 4 – **IMPLEMENTATION (Kidney Failure)**	25-38
7.	Chapter 5 – **IMPLEMENTATION (Heart Failure)**	39-47
8.	Chapter 6 – **RESULTS AND ANALYSIS**	48-50
9.	Chapter 7 – **CONCLUSION AND FUTURE SCOPE**	51-52
10.	**REFERENCES**	53
11.	**DATASET INFORMATION**	54
12.	**ATTRIBUTE INFORMATION**	55

LIST OF FIGURES

Figure No.	Description	Page No.
1.1	Steps for feeding a ML model	9
2.1	Introduction to ML	10
2.2	ML algorithms	12
2.3	ML Types	13
2.4	Formulae used for calculating nearest distance	14
2.5	Graphical depiction of Linear Regression	15
2.6	Pictorial representation of confusion matrix	15
2.7	Deep Learning	17
2.8	Data of 400 patients with 24 attributes	19
4.1	Loading of modules and helper functions	25
4.2	Loading the .csv file	26
4.3	Cleaning and pre-processing of the data	26
4.4	Generating the colormap graph	27
4.5	Splitting data into training and testing models	28
4.6	Feature Importance selection	28
4.7	Making predictions and obtaining results	29
4.8	Training the CDNN	30
4.9	Training the CDNN	31
4.10	Heatmap of all 400 values	31
4.11	Results of comparison of various NN	32
4.12(a-j)	Results of comparison of various NN	33
4.13	Trained CDNN after data feeding	33
4.14	Blank heatmap (all values processed)	34
4.15	Using MinMaxScaler and LabelEncoder	35
4.16	Result of Epoch	35
4.17	Accuracy Results	35
4.18(a-b)	Graph of Accuracy and Loss	36
4.19	Important Libraries	36

4.20	Building the ANN Model	37
4.21	Training the model	37
4.22	Results of Epoch with accuracy 99.56	37
4.23	Model Accuracy and Loss	38
4.24	Final results of ANN	38
5.1	Dataset of 303 heart patients with 14 selective parameters	39
5.2	Description of the variables with the ‘target’	40
5.3	Data comparison 1	40
5.4	Data comparison 2	41
5.5	Data comparison 3	41
5.6	Data comparison 4	42
5.7	70% training and 30% testing split	42
5.8	Logistic Regression	43
5.9	Naïve Bayes	43
5.10	Support Vector Machines	44
5.11	K-Nearest Neighbours	44
5.12	Decision Tree	45
5.13	Random Forest Classifier	46
5.14	XGBoost	46
5.15	The Neural Networks working	47
5.16	Neural Networks	47
6.1	Final Results for the various algorithms	49

LIST OF TABLES

Table No.	Description	Page No.
2.1	Difference between various ML algorithms	16
2.2	Confusion Matrix	16
6.1	Results of comparative analysis	48
6.2	Final results for the various algorithms	50

ABSTRACT

In this task, I was approached to explore different avenues regarding genuine world datasets, and to perceive how AI calculations can be utilized to discover the examples in information. I was relied upon to pick up experience utilizing normal information mining and AI libraries, and was required to present a report about the dataset and the calculations utilized with their outcomes. Subsequent to playing out the necessary undertakings on the gave datasets, in this lies my report.

AIM

The project's aim is to plan a systematically enhanced machine learning model which will be helping us in the early detection of various organ failure instances and thus in turn, will help in the early detection and prevention of the aforementioned cases, saving lives, time and money.

OBJECTIVES

- Study about various methods of detection of different types of organ failures.
- Download multiple specific organ sample dataset to train the model.
- Use of different AI calculations to the made dataset utilizing Python and looking at the calculations based on parameters like exactness, accuracy, time utilization and so on.
- Obtaining the result with the best parameter judgement and converting it into a real-world application.

Chapter-1

INTRODUCTION

1.1 Introduction

ML is a subset of computer sciences which developed from example acknowledgment in information, and furthermore from the hypothetical learning in user reasoning. It is the top of the line calling to most fascinating vocations related to information investigation today[4]. As information sources multiply alongside the registering capacity to process them, approaching directly to the information is one of the clearest ways to rapidly pick up experiences and make predictions[6].

AI is the investigation of a rundown of sub-issues, like: dynamic, bunching, arrangement, determining, profound learning, inductive rationale programming, SVM, support learning, similitude and metric learning, hereditary calculations, meagre word reference learning, and so forth. Supervised learning, or arrangement is the undertaking of deriving a quantity from a named data[6]. In this, we use sets namely preparation and test. The preparation and test set comprises of a lot of models comprising of info and yield vectors, and the objective of the administered learning calculation is to gather a capacity that maps the information vector to the yield vector with negligible blunder. In an ideal situation, a model prepared on a lot of models will characterize an inconspicuous model in a right manner, requiring summing up from the preparation set in a sensible manner. In layman's terms, managed learning can be named as the procedure of idea realizing, where a mind is presented to a lot of data sources and result vectors and the cerebrum learns the idea that relates said contributions to yields. A wide cluster of regulated AI calculations is accessible to the ML aficionado, for instance Decision Trees, Neural Networks, SVM, Naïve Bayes Classifier, Random Forest, Bayes Net and so on, and they each have their own benefits and negative marks with respect to their distinguished purpose[4].

1.2 Motivation

Chronic Kidney Disease (CKD) progressively advances and typically after months or years the kidney loses its usefulness. By and large, it may not be distinguished before it loses 25% of its usefulness[1]. The beginning of renal failure may not be perceived by the patients since renal failure may not give any manifestations at first. Renal failure treatment focuses to control the causes and slow down the development of the renal failure. In the event that medications are insufficient, patient will be at long last phase of failure and the final option is transplant or dialysis. At present, 4 of each 1000 man in the United Kingdom are experiencing renal failure and in excess of 300,000 American patients at long last phase of kidney malady make due with dialysis.[3]

Also, as indicated by the National Health Service, kidney failure is progressive in South Asia, Africa, then in other different nations. Because the identification of the constant kidney disease isn't plausible until the failure has totally taken over; subsequently, understanding the kidney failure in the principal stage is critical. Through early analysis, the demonstration of every kidney can be taken care of, which prompts diminishing the danger of irreversible outcomes. Thus, standard checkups and early analysis are vital to the patients, for they can forestall fundamental dangers of renal stoppage and related maladies. One of the means to identify failure is blood test. Hence, it very well may be recognized by estimating components, and doctors can choose treatment forms, reducing the pace of progression.

We work on finding dataset patterns through a sample of the data from various kidney disease diagnosis (ECG and other methods), containing attributes such as blood pressure, age, RBC count, haemoglobin etc, and use various algorithms on this data, observing the results. The best results will be the ones which consume the least time with most accurate outputs.

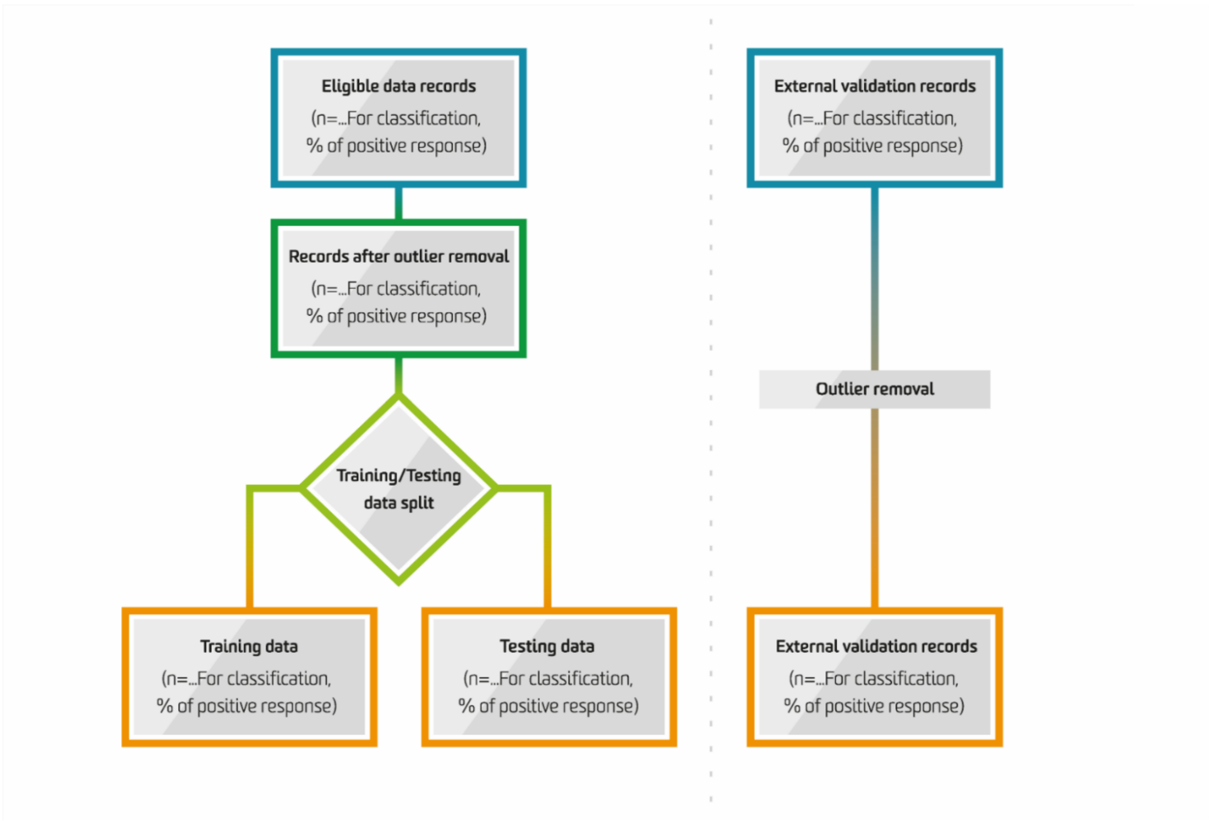

Eligible data records
(n=...For classification,
% of positive response)
Records after outlier removal
(n=...For classification,
% of positive response)
Training/Testing data split
Training data
(n=...For classification,
% of positive response)
Testing data
(n=...For classification,
% of positive response)
External validation records
(n=...For classification,
% of positive response)
Outlier removal
External validation records
(n=...For classification,
% of positive response)

Figure 1.1: Steps for training and testing a Machine Learning Model

1.3 Organization of The Report

Chapter-1: I introduced the scope of our project and why is it of concern in today's date. I have also further shared our motivation behind the establishment of the idea behind this project and how I came across its formulation and working.

Chapter-2: I've talked about the writing survey of the project, which comprises of the terms, significance and the working of different sorts of algorithms utilized. I have depicted different favourable circumstances and weaknesses to distinguish the most appropriate algorithm for our task.

Chapter-3: I've designed and mentioned all those framework necessities to run the calculations and the runtime environment where the calculations are tried and tested.

Chapter-4: I've talked insight regarding the calculations utilized and the arithmetic or the plan behind these calculations for better understanding and tried out these calculations on the dataset to acquire results.

Chapter-5: I've talked about the outcomes and done testing based on different parameters to acquire the most appropriate calculation for our dataset.

Chapter-6: I've finished up the report and learning of the undertaking. Likewise, I have talked about the future and the progressions that can be done to the project.

CHAPTER-2

LITERATURE REVIEW

2.1 Terminologies

Machine Learning (ML) - ML is a sort of calculation that is a part of computerized reasoning and that makes the framework or the product to be sufficiently intelligent to have the choice of being right without being unequivocally customized and can anticipate the results. The primary thought behind these sorts of calculations is that it gets input information as content or pictures and the framework of the model is prepared with the measurable inputs to distinguish or foresee the yield, refreshing the outputs as new information is at avail. It requires the calculation to look through the dataset and search for examples or similitudes and controlling or changing the framework as needs be.

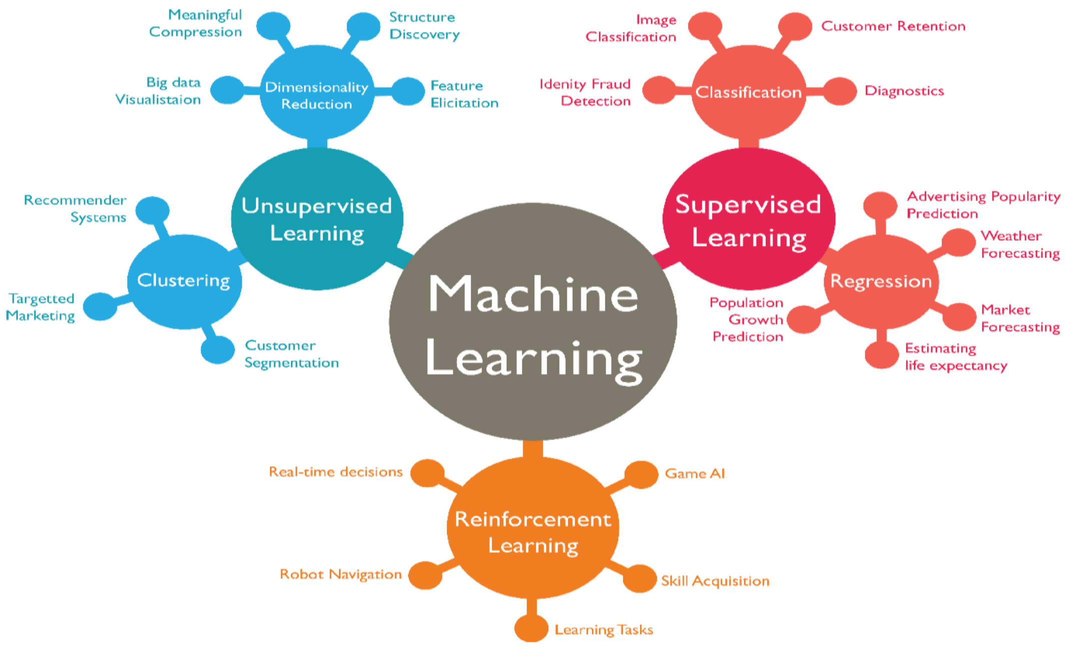


Figure 2.1: Introduction to ML

2.1 How Machine Learning Works

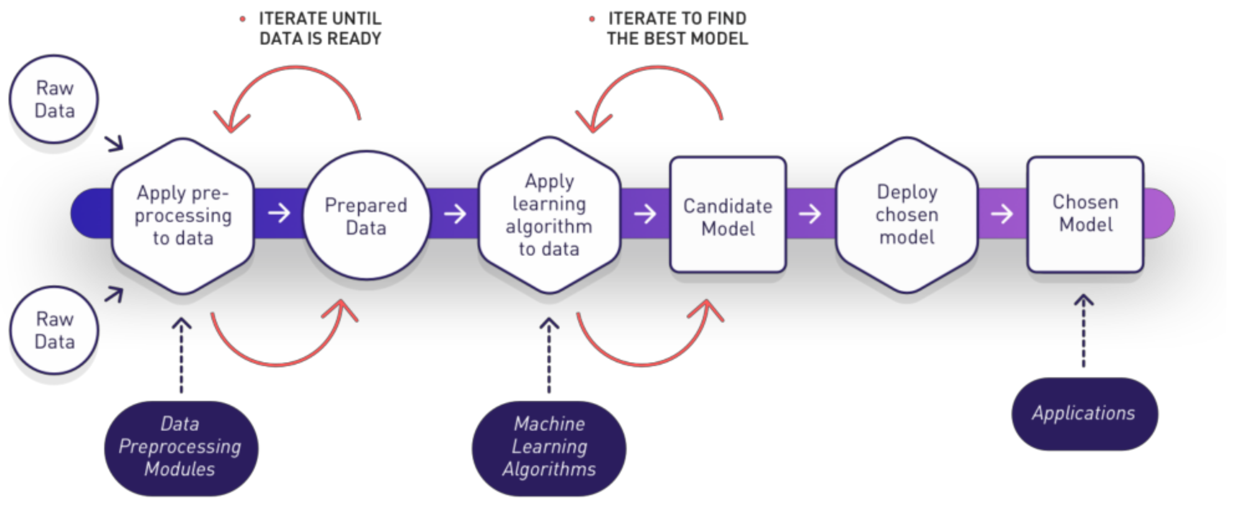

ITERATE UNTIL DATA IS READY
ITERATE TO FIND THE BEST MODEL
Raw Data
Raw Data
Apply pre-processing to data
Prepared Data
Apply learning algorithm to data
Candidate Model
Deploy chosen model
Chosen Model
Data Preprocessing Modules
Machine Learning Algorithms
Applications

Figure 2.2: ML algorithm workflow

The procedure of learning begins with the assortment of information or perceptions as the inputs which can be as pictures, content, tables and so forth. Further, numerous predefined AI calculations are applied to the inputs which either classify the information into collections or recognizes patterns amongst the dataset to calculate and give proper results for the output obtained. AI calculations are approximately arranged into unsupervised and supervised algorithms.

2.2 Types of Machine Learning

Unsupervised Machine Learning - This kind is not quite the same as supervised ML as the calculations are utilized when the model isn't prepared before neither is it trained nor is it marked. These calculations make the framework to tangle a shrouded pattern in the unnamed dataset and foresee potential outcomes with utilization of such examples while expelling the exceptions.

Supervised Machine learning - Supervised learning - This kind is used by data which is now prepared from the past outputs from the results of past utilizing marked information to anticipate the result of the new information. For this case this dataset is investigated, the algorithm at that point delivers a gathered function that can help in expectation of the yield estimations of new information. It can likewise break down the input and the result and contrast with the recently stored information to get errors and as ready to change and train the model as needs be.

Semi-Supervised Machine Learning – This is the calculation which uses benefits of both types of ML for preparing the dataset and subsequently delivers lots of useful and rewarding classifiers. The model uses both named and unnamed data for the preparation and it for the most part requires a limited quantity of named data and a generally huge number of unnamed data which are utilized all the while to prepare the model. This is utilized for improving the precision and the forecast capacities of the model and consequently frequently utilized on account of information requiring both gain and loss from the useful information.

Reinforcement Machine Learning- The model trains on a reward-based activity of finding mistakes and correctness. The attributes to this type of learning are the rewards and mistakes. For this situation learning from past missteps or mistakes for the model is made to work with the learning ability to determine the result and the perfect conduct for maximum performance.

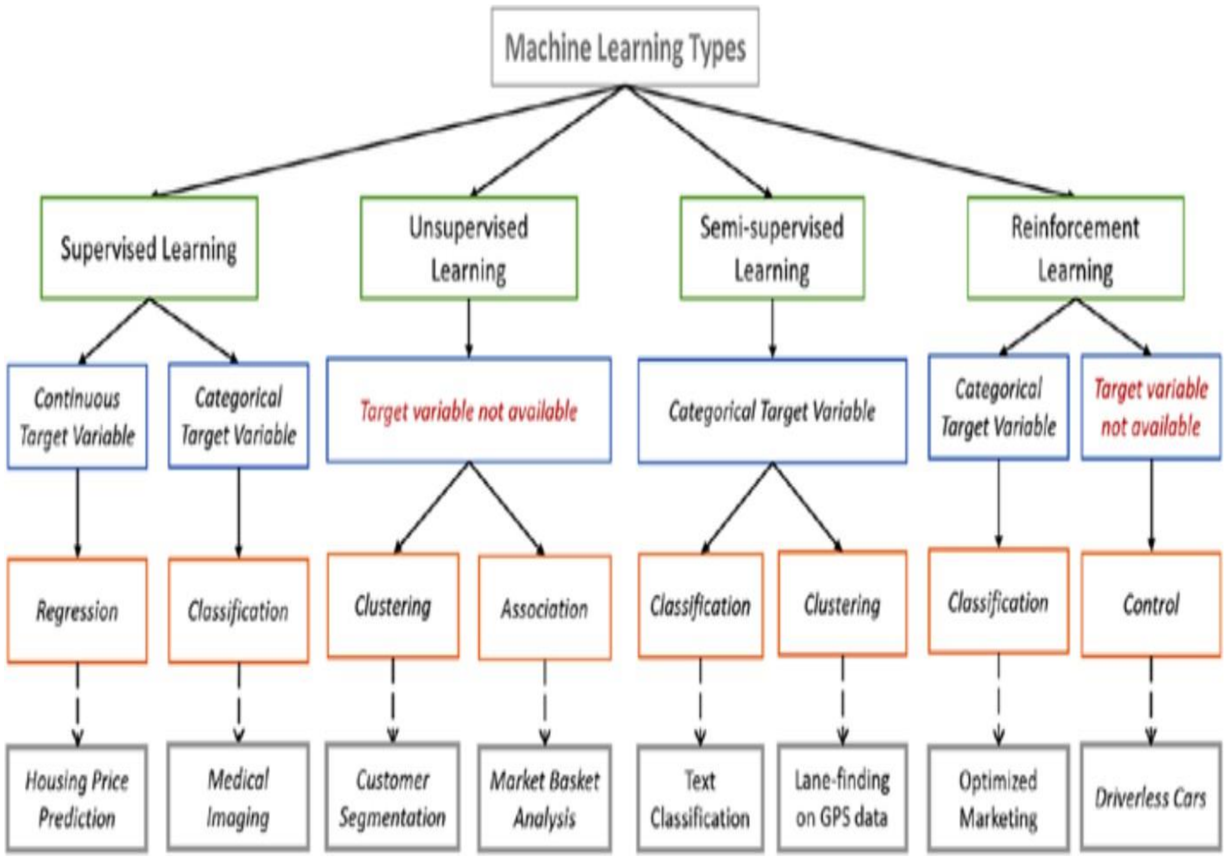


Figure 2.3: Types of machine learning

2.3 Algorithms

This segment comprises of several useful algorithms which are applied in this project and commented on:

Supervised learning

K-Nearest Neighbours- Nearest neighbors is utilized both for classification and regression yet is generally utilized for classification issues. This calculation is simple and consumes lesser time. The K represents the no. of neighbors assigned by user. The calculation utilizes the Euclidean separation to gauge KNN to the information point and anticipate the output as indicated by its neighbours[8].

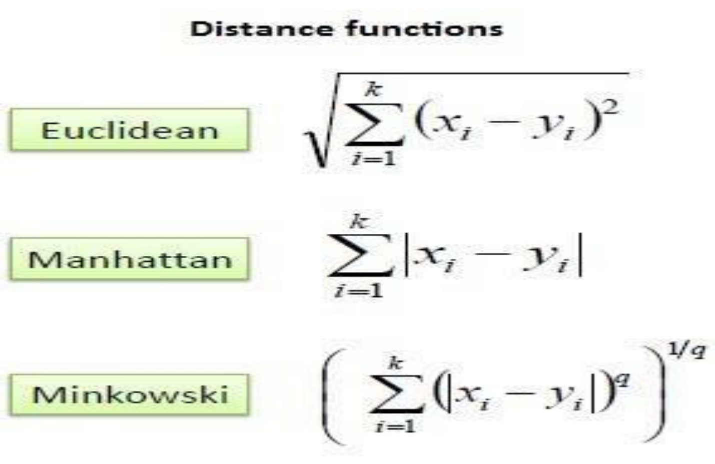

Distance functions
Euclidean
$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$
Manhattan
$\sum_{i=1}^{k}|x_i - y_i|$
Minkowski
$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

Figure 2.4: Formula for calculating minimum separation.

Naïve Bayes- Bayes hypothesis is a kind of classifier algorithms utilized for different types of classification not regression problems. This hypothesis is known as so in light of the fact that it has foundations in the Bayes theorem. N.Bayes frequently is spoken to by probabilities. The information is put as probabilities in this type on model.

Formula - P(x,y) = (P(y|x)*P(x))/P(y)

Decision Trees- A supervised learning algorithm where an information structure is utilized to take care of an issue. For this situation a node or centre is mentioned to as a class name and the attributes are mentioned in the insider nodes. They can take care of the issues of both regression and classification. At first, we take the entire dataset as a root and an unmitigated element value are used and the continuous inputs are first made discrete qualities before utilizing them to assemble the model needed. At that point factual strategies are utilized for requesting the traits as inward node or root.

Formula-

$$Gain(S, A) = Entropy(S) - \sum_{v\epsilon Values(A)} \frac{|S_v|}{|S|}.Entropy(S_v)$$

Where:

- S - represents the training set
- A - represents the attribute that we are comparing
- $|S|$ - represents the number of elements in the training set
- $|S_v|$ - represents the number of elements where the attribute has a part

$$Entropy(S) = \sum_i -p_i log_2 p_i$$

Regression- This algorithm utilizes the factual ideas and establishes a connect between the inputs and number outputs. The model is characterized by a linear equation which consolidates the input estimations of the set chosen and can predict the output based on past learning experiences of the model.

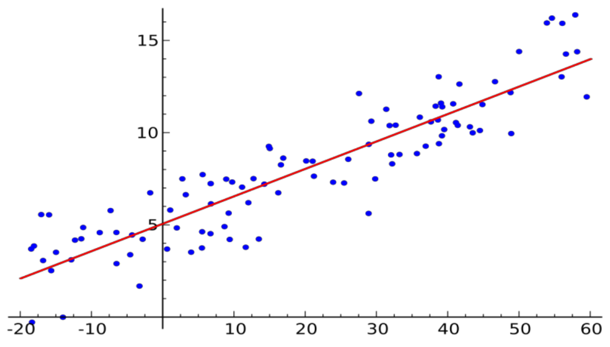


Figure 2.5: Graphical depiction of regression.

SVM- It is a type of classification algorithm used for both the former and regression problems. This is generally utilized where every item of data is plotted in a n-dimensional space and n characterizes the highlights in there and the estimation of each and every element is the estimation of each organize. Furthermore, different hyperplanes are made to separate those 2 modules.

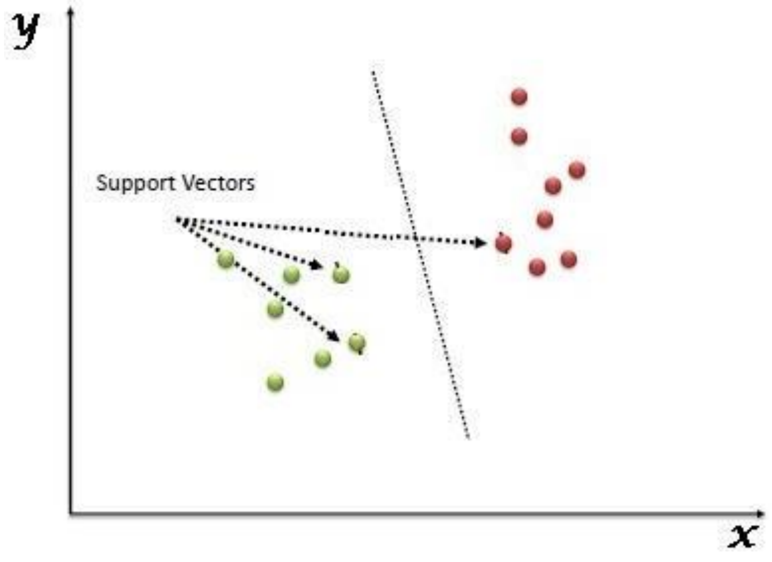
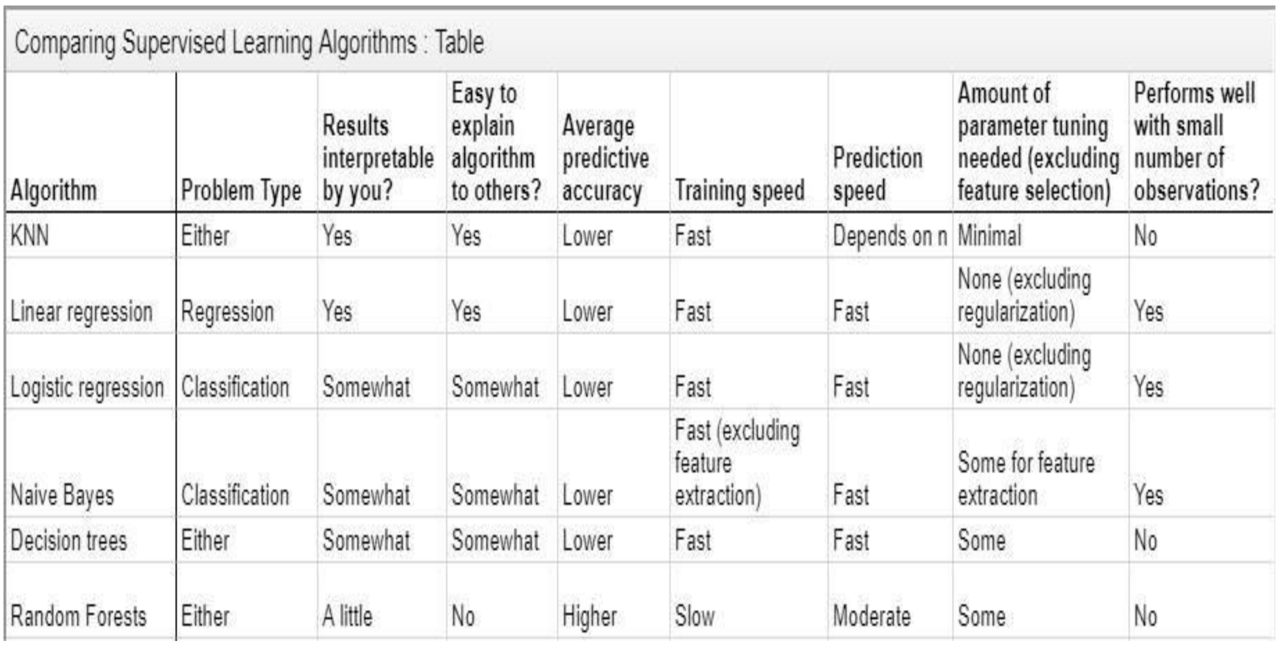


Figure 2.6: Graphical depiction of SVM.



Comparing Supervised Learning Algorithms : Table

Algorithm	Problem Type	Results interpretable by you?	Easy to explain algorithm to others?	Average predictive accuracy	Training speed	Prediction speed	Amount of parameter tuning needed (excluding feature selection)	Performs well with small number of observations?
KNN	Either	Yes	Yes	Lower	Fast	Depends on n	Minimal	No
Linear regression	Regression	Yes	Yes	Lower	Fast	Fast	None (excluding regularization)	Yes
Logistic regression	Classification	Somewhat	Somewhat	Lower	Fast	Fast	None (excluding regularization)	Yes
Naive Bayes	Classification	Somewhat	Somewhat	Lower	Fast (excluding feature extraction)	Fast	Some for feature extraction	Yes
Decision trees	Either	Somewhat	Somewhat	Lower	Fast	Fast	Some	No
Random Forests	Either	A little	No	Higher	Slow	Moderate	Some	No

Table 2.1: Different Machine Learning algorithms

2.4 Understanding of Performance Factors

There are different strategies to assess the use of various algorithms. One of the ways is to decide the curve are or the curve ROC and other different parameters otherwise called as Confusion Values. To assess the present proportion of the grouping model that gives the genuine qualities are known, the matrix table is used[9].

		Predicted	
		Yes	**No**
Actual	**Yes**	**True Positive**	**False Negative**
	No	**False Positive**	**True Negative**

Table 2.2: The Matrix (Confusion)

The table presented comprises of confusion values and has 4 segments. The two area in the green are the True values and these are the perceptions which are correctly anticipated. The two areas in red in light of the fact that these qualities are wrongly anticipated and, in this way, should be limited. These segments are False and happen when we experience an inconsistency between genuine class with the anticipated class[10]].

Genuine Positives (TP) – These qualities are effectively anticipated and are sure quantities which can be portrayed as the correct estimation of real class and correct estimation of anticipated class. It is denoted by TP.

Genuine Negatives (TN) - These quantities are accurately anticipated yet untrue qualities which alludes the invalidation of real class and disproof of anticipated class. This is signified by TN.

Incorrect Positives (FP) – These quantities that are incorrect yet are valid in genuine.

Incorrect Negative (FN) – These qualities are incorrectly anticipated and are negative in real class.

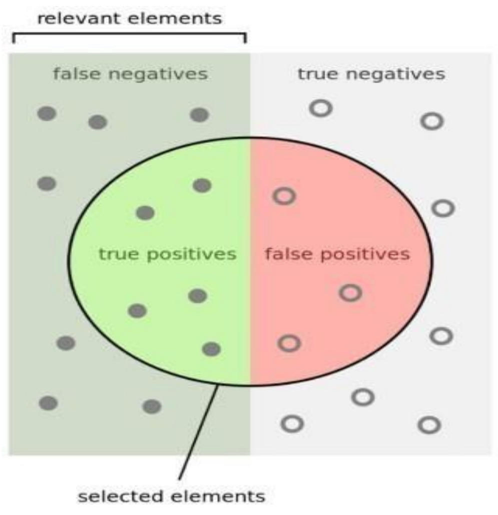


Figure 2.7: Graphic depiction of misperception matrix

Further, I studied upon more limitations of performance.

Accuracy – This is a normal measure and is basically an extent of adequately foreseen perception by the total observations. Envisioning that, in case of having high precision, the model turns out to be perfect. Really, accuracy is considered an unparalleled measure yet exactly when having symmetric datasets where estimations of non-true positive and non-true negatives are generally same. As such, we have to look at changed parameters for evaluation of the execution of the model.

Accuracy = (TN+TP)/(FN+TP+TN+FP)

Precision – This is the extent of precisely foreseen correct output observations to the total foreseen positive perceptions. Large precision values can relate to the low false positive rate, which is a good thing.

Precision = TP/FP+TP

Recall – This is described as the extent of viably foreseen positive judgements to the all observations in a real defined class.

Recall = TP/FN+TP

F1 Score – This is the average of Precision and Recall, weighted one. Thus, F1 thinks about both false positives and negatives. It isn't accuracy, yet this score is typically considered more important than precision, especially on possibility that we might have a distributed class movement. Exactness gives best results if false negatives positives have practically identical weight. In case the cost of false negatives and positives are through and various, it's a better practice to look at both Recall and Precision.

F1 = 2*(Precision*Recall) / (Precision + Recall)

2.5 Deep Learning (DL)

Versatile Learning is an increasingly intricate and clever sub-classification of AI having all the calculations roused by the working and biological composition of the brain known as Artificial neural network. Furthermore, it likewise alludes to the strategies that are utilized for learning networks with various layers. Artificial Neural Network is the kind of model which is made out of motivation and takes a shot at the essential thought of sensory systems and the handling of data in brain to gain from information. Here, the learning components can be either administered, semi-managed or unaided. Profound learning can be demonstrated as fruitful in different fields and brought about increasingly practical machine learning stages. It can be used in the field of medication structure till the traffic forecast and furthermore for the article acknowledgment. A deep neural network is unique in relation to a neural network due to the quantity of layers. The usage of profound learning whilst I did for this, I experienced 2 primary issues, for example, 1) the computational force required for the way toward preparing the model was not lesser than the framework accessible and, in a way, requires more opportunity for calculation. 2) A more difficult problem experienced during the usage was the inclination disappearing issue, that was, in a neural system that has enactment capacities, for example, the hyperbolic digression and the slope is (-1,1) or (0,1), the chain rule is generally used to figure out the back-propagation. It comprises of duplicating 'k' to some little numbers from output layer through a k-layer networks, which implies that the angle diminished exponentially with k. Results are that the above layers of the model trains gradually than the rest of the layers present in the formulation of the deep learning model skeleton.

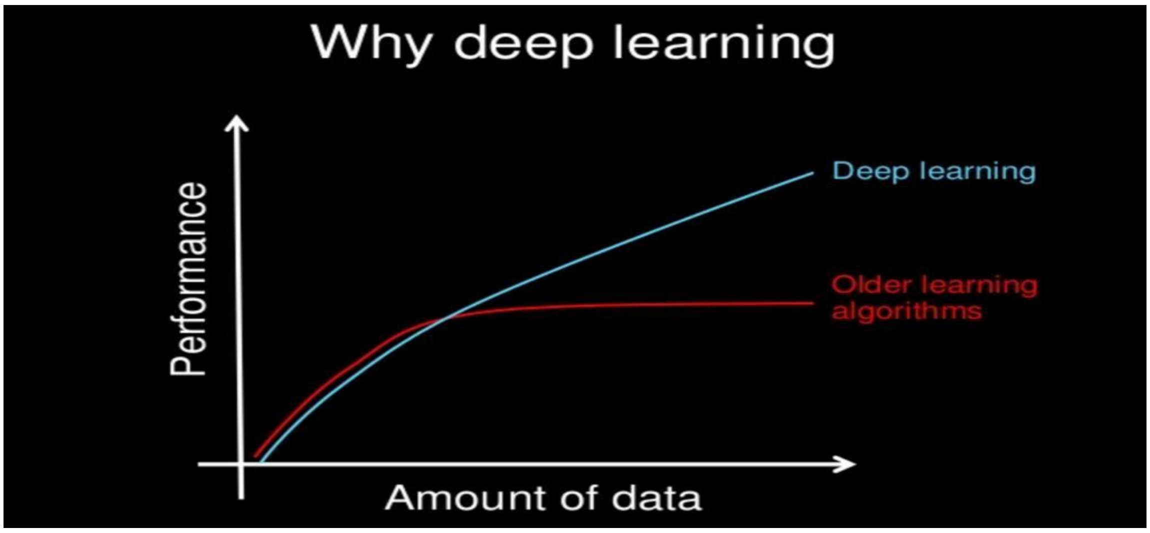


Figure 2.8: Graphical depiction of Deep Learning vs additional learning procedures

2.6 Perceptrons

One of the most useful learning calculations is the perceptron one and it is a minuscule block of the Artificial Neural Network. The working of the perceptron is by taking different information sources (a1, a2… … aj) and creation of a solitary output (y). Furthermore, weighted sources of ideas were additionally helping decide the significance of separate contributions to the output. The subsequent output is either 1 or 0 and it is possibly controlled by cross-checking the weighted entirety being more prominent than 0 or under 0.

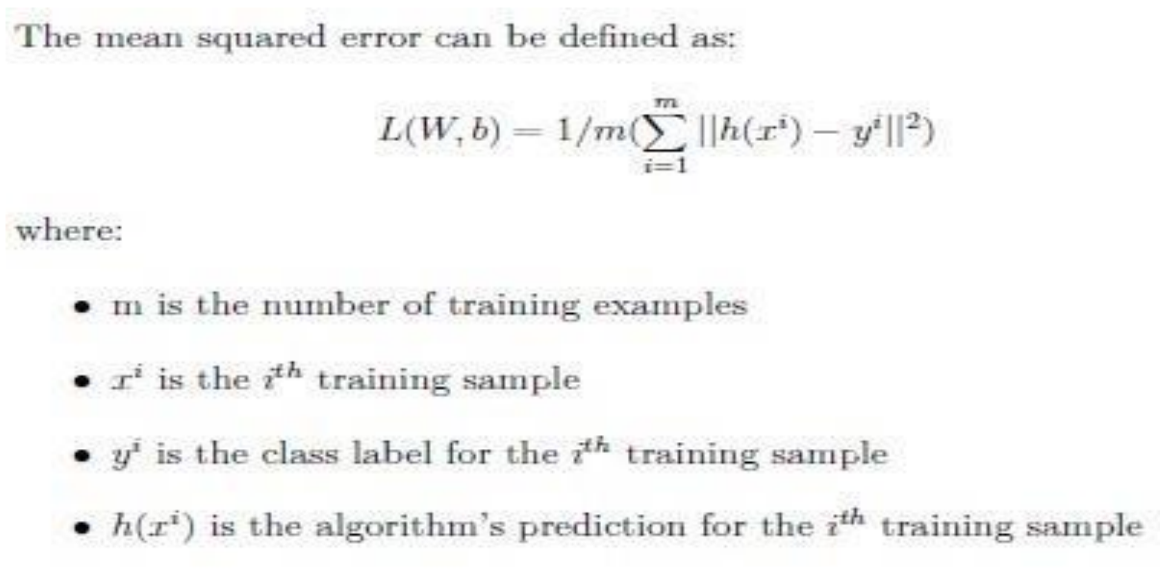
Weighted Summation- $\sum_i w_i * a_i + b$

$$\text{Output} = \left\{ \begin{array}{ll} 1: & \text{if } w * a + b > 0 \\ 0: & \text{if } w * a + b <= 0 \end{array} \right\}$$

Be that as it may, as perceptron algorithm just gives the yield as 1 or 0, making it troublesome or about difficult to expand the functionalities of the model to have the option to take a shot at characterization errands having various classes. Moreover, this issue might be settled by considering numerous perceptrons in a layer ensuring that every layer having its respective perceptron acquires a similar info and the perceptrons are answerable for the output work. The ANN is just perceptrons with multiple layers, though the perceptron simply an ANN with only one layer, which is regularly the yield layer having just a single neuron.

2.7 Loss/Cost Function

The usefulness of a NN can be estimated by a capacity also known as the work function / loss work function which helps in estimating the error in the expectation by the calculations and the right label if the forecast or the aggregate arrangement of forecast can be mentioned alongside the mark or a lot of names. Among these different cost work accessible the least complex and the generally utilized in NN systems is the MSE[10].



The mean squared error can be defined as:

$$L(W, b) = 1/m(\sum_{i=1}^{m} ||h(x^i) - y^i||^2)$$

where:

- m is the number of training examples
- x^i is the i^{th} training sample
- y^i is the class label for the i^{th} training sample
- $h(x^i)$ is the algorithm's prediction for the i^{th} training sample

The final output coming from training these networks is to reduce the work / cost function and determine the individual parameters that help in doing so. For this technique, I implemented the gradient descend algorithm.

2.8 Problems in Learning

We should be informed on as to choose a suitable training algorithm for a specified dataset. To diligently use up an algorithm for a supervised learning job, we must take into account these factors:

1. Non-Homogeneity of Data:

Most neural systems and SVM have the vectors used to be non-heterogeneous and standardized. The calculations that utilize separation measurements are exceptionally touchy to this, and thus if the information is non-homogeneous, these strategies ought to be the idea in retrospect. The Decision Trees way of working can deal with heterogeneous information quite easily[10].

2. Redundancy of Data:

On the off chance that the information contains repetitive data, for example contain profoundly related qualities, at that point it's futile to utilize separation-based techniques as a result of numerical unsteadiness. For this situation, a regularization can be utilized to the information to forestall this situation[10].

Features: If there is some reliance between the component vectors, at that point calculations that screen complex collaborations like Neural Networks and Decision Trees toll better than other algorithms[10].

3. Bias-Variance Trade-off:

Calculation is one-sided for a specific information 'x' prepared on every one of these informational collections, it is deliberately mistaken while anticipating the right output for x, though a learning calculation difference for a specific information x on the off prediction chance of diverse values when prepared on a variety of sets is high. The expectation blunder of a trained classifier can be identified with the aggregate of bias and change of the calculation, and neither be higher as the forecast mistake would be higher. An important element of AI calculations is that they can tune the harmony among inclination and change naturally, or by manual tuning utilizing predisposition values, and utilizing these performed calculations will settle this condition[10].

4. Dimension Problems:

In the event that the issue has an input that has an enormous number of measurements, and the issue just relies upon the information space with little measurements, the AI algorithm will confound by the tremendous number of measurements and consequently the difference of the calculation can be great. Practically speaking, when the information researcher can physically expel unimportant highlights from the information, it probably improves the precision of the trained capacity. Moreover, there are a variety of calculations for inclusion of choice that try to distinguish the significant highlights and discard the useless ones, for example Principle Component Analysis. This lessens the dimensionality of the dataset[10].

5. Overfitting:

Sometimes the output we have from all the processing work can be comprised of noise or disturbance in the furnished data due to user errors. For this situation, these calculations must not endeavour to derive the capacity that precisely coordinates all the information. Being excessively cautious in accommodating of all the information causes overfitting, after which the model would answer consummately for all preparation models. However, it will have a high error rate for inconspicuous examples. A pragmatic method of forestalling this is halting the learning procedure rashly, just as applying channels to the information in the pre-learning stage to evacuate commotions. Simply in the wake of considering every one of these variables would we be able to use a directed calculation with the dataset we are chipping away at. For instance, on the off chance that we were using on this dataset comprising of non-homogeneous information, at that point choice trees will be a lot better than others for the case being[10].

CHAPTER-3

SYSTEM DESIGN

3.1 System Requirements

We used following system configuration for this project as prerequisites:

Windows Operating System version 10

Jupyter Notebook

Python 3.7

8 GB memory

An i7-8700HQ CPU

NVIDIA® GeForce™ GTX 1050Ti

3.2 Why Python and no other languages?

Python is a highly popular language and can be very much intelligible. Moreover, it provides the user with and assortment of libraries which help in making scariest calculations or activities less complex. Python contains blocks for pretty much every usable document for example: graphical usage, content creation or with sound documents. In any event, when shifting to another system, python is truly flexible. Python has a great connection with others so that it simple to look up help in time of need.

3.3 Why Jupyter?

Jupyter notebook is used widely as it avails the pre-installed libraries making the programmer free from the installation of all these libraries. This contains about 200 packages used on a daily basis by various programmers.

3.4 Python -SCIKITLEARN

Scikitlearn is widely used in python generally for ML and is able to feature different types of algorithms.

3.5 Python -PANDAS

A python library that can provide with well managed performance metrics. This is laidback due to its ease of usage and analysis tools. It is being excessively used in all commerce and manufacturing fields.

3.6 Python -KERAS

Its general use is in ANN. It is designed for swift trialling of many intricate algorithms.It focuses on providing easy of usage, access to many fields and being modular.

3.7 TENSORFLOW

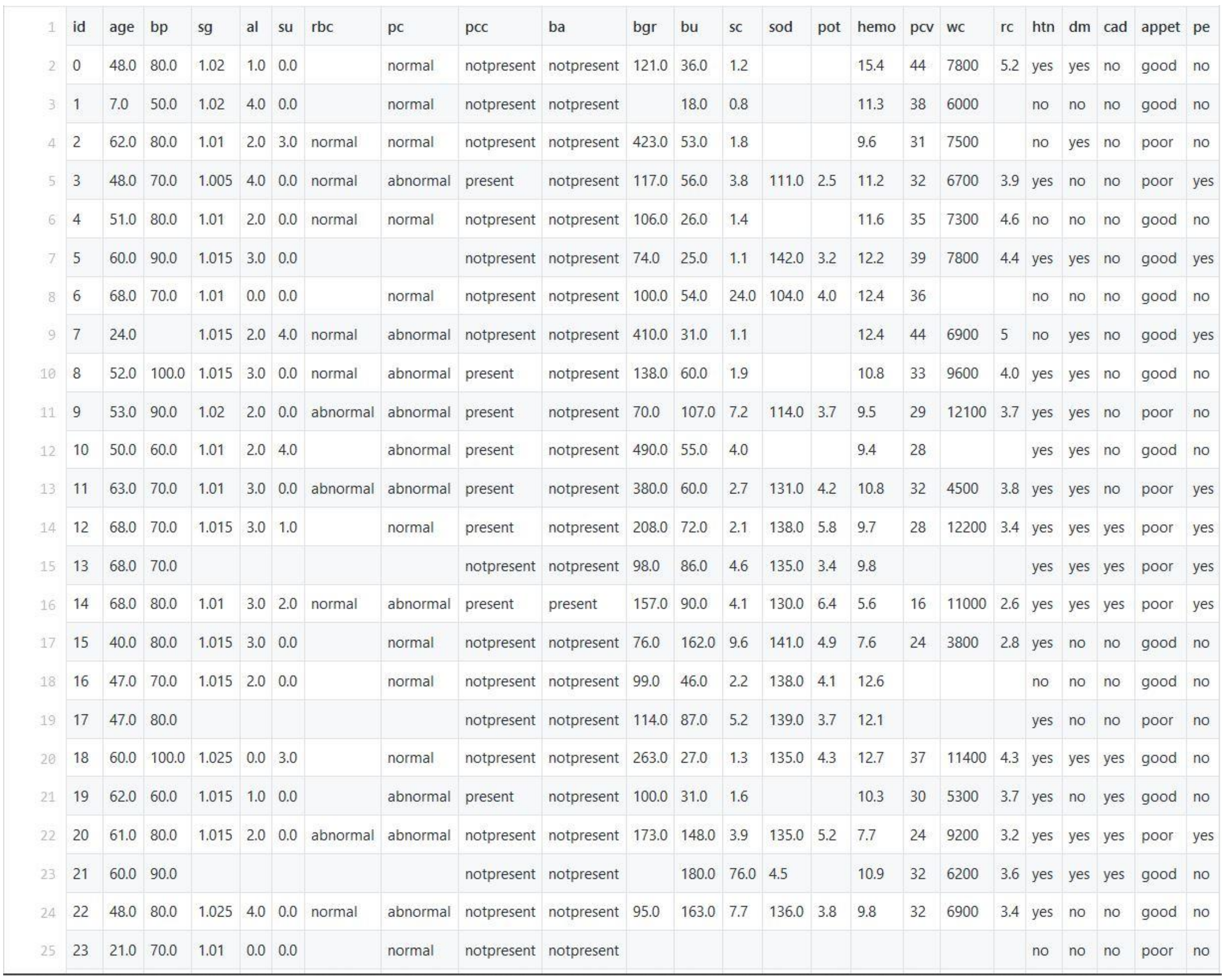
Used to perform a variety of tasks like programming and dataflow. TensorFlow can be well-defined as a representative math which used in neural networks like fields of machine learning universe.

CHAPTER-4

IMPLEMENTATION (Kidney Failure)

Here I would be discussing about the testing and research phases of the project.

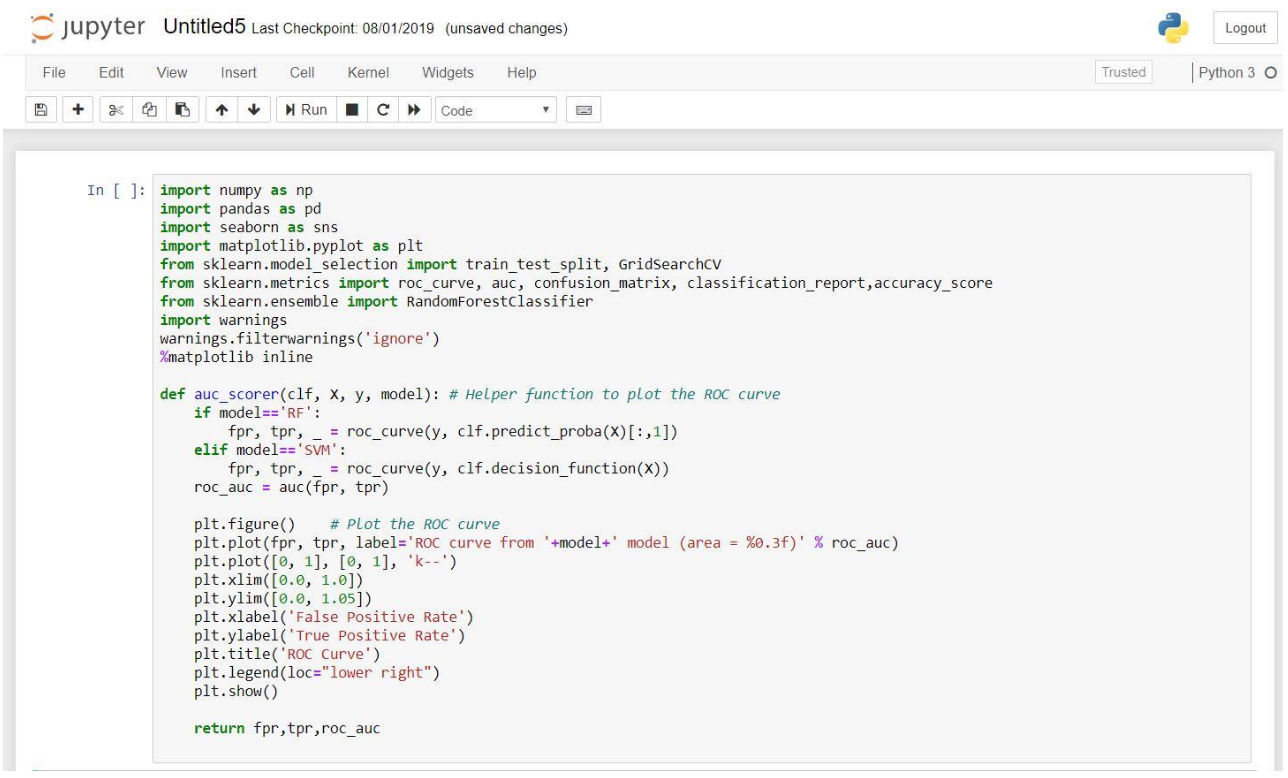
I studied about the various stages and factors that might be affecting the patient’s kidneys, declassifying it from normal. Given 24 health related attributes taken in a set time period for four hundred patients, by means of the data of one hundred and fifty-eight patients through comprehensive archives to get the result(i.e. whether someone has CKD) of the lasting two hundred and forty two patients(with unidentified parameters present in the data).



id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	htn	dm	cad	appet	pe
0	48.0	80.0	1.02	1.0	0.0		normal	notpresent	notpresent	121.0	36.0	1.2			15.4	44	7800	5.2	yes	yes	no	good	no
1	7.0	50.0	1.02	4.0	0.0		normal	notpresent	notpresent		18.0	0.8			11.3	38	6000		no	no	no	good	no
2	62.0	80.0	1.01	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8			9.6	31	7500		no	yes	no	poor	no
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	111.0	2.5	11.2	32	6700	3.9	yes	no	no	poor	yes
4	51.0	80.0	1.01	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4			11.6	35	7300	4.6	no	no	no	good	no
5	60.0	90.0	1.015	3.0	0.0			notpresent	notpresent	74.0	25.0	1.1	142.0	3.2	12.2	39	7800	4.4	yes	yes	no	good	yes
6	68.0	70.0	1.01	0.0	0.0		normal	notpresent	notpresent	100.0	54.0	24.0	104.0	4.0	12.4	36			no	no	no	good	no
7	24.0		1.015	2.0	4.0	normal	abnormal	notpresent	notpresent	410.0	31.0	1.1			12.4	44	6900	5	no	yes	no	good	yes
8	52.0	100.0	1.015	3.0	0.0	normal	abnormal	present	notpresent	138.0	60.0	1.9			10.8	33	9600	4.0	yes	yes	no	good	no
9	53.0	90.0	1.02	2.0	0.0	abnormal	abnormal	present	notpresent	70.0	107.0	7.2	114.0	3.7	9.5	29	12100	3.7	yes	yes	no	poor	no
10	50.0	60.0	1.01	2.0	4.0		abnormal	present	notpresent	490.0	55.0	4.0			9.4	28			yes	yes	no	good	no
11	63.0	70.0	1.01	3.0	0.0	abnormal	abnormal	present	notpresent	380.0	60.0	2.7	131.0	4.2	10.8	32	4500	3.8	yes	yes	no	poor	yes
12	68.0	70.0	1.015	3.0	1.0		normal	present	notpresent	208.0	72.0	2.1	138.0	5.8	9.7	28	12200	3.4	yes	yes	yes	poor	yes
13	68.0	70.0						notpresent	notpresent	98.0	86.0	4.6	135.0	3.4	9.8				yes	yes	yes	poor	yes
14	68.0	80.0	1.01	3.0	2.0	normal	abnormal	present	present	157.0	90.0	4.1	130.0	6.4	5.6	16	11000	2.6	yes	yes	yes	poor	yes
15	40.0	80.0	1.015	3.0	0.0		normal	notpresent	notpresent	76.0	162.0	9.6	141.0	4.9	7.6	24	3800	2.8	yes	no	no	good	no
16	47.0	70.0	1.015	2.0	0.0		normal	notpresent	notpresent	99.0	46.0	2.2	138.0	4.1	12.6				no	no	no	good	no
17	47.0	80.0						notpresent	notpresent	114.0	87.0	5.2	139.0	3.7	12.1				yes	no	no	poor	no
18	60.0	100.0	1.025	0.0	3.0		normal	notpresent	notpresent	263.0	27.0	1.3	135.0	4.3	12.7	37	11400	4.3	yes	yes	yes	good	no
19	62.0	60.0	1.015	1.0	0.0		abnormal	present	notpresent	100.0	31.0	1.6			10.3	30	5300	3.7	yes	no	yes	good	no
20	61.0	80.0	1.015	2.0	0.0	abnormal	abnormal	notpresent	notpresent	173.0	148.0	3.9	135.0	5.2	7.7	24	9200	3.2	yes	yes	yes	poor	yes
21	60.0	90.0						notpresent	notpresent		180.0	76.0	4.5		10.9	32	6200	3.6	yes	yes	yes	good	no
22	48.0	80.0	1.025	4.0	0.0	normal	abnormal	notpresent	notpresent	95.0	163.0	7.7	136.0	3.8	9.8	32	6900	3.4	yes	no	no	good	no
23	21.0	70.0	1.01	0.0	0.0		normal	notpresent	notpresent										no	no	no	poor	no

Figure 4.1: Data of 400 patients (23 shown) with 24 health related attributes

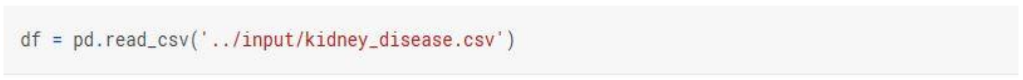
I picked up an existing machine learning algorithm related to the **Random Forest Classifier**, to setup a training model and feed the previous data of 400 patients into it. I did it by the help of Python (3.7) programming language running on Anaconda3 GUI through Jupyter Notebook. I implemented several python3 libraries such as Matplotlib, Pandas, NumPy, SkLearn etc. These are predefined functions and figures that help the computer compile, load and train our dataset according to set parameters of our needs. Our first model was getting ready for a test run for measuring the initial accuracy and obtaining a graph regarding the same.



```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import roc_curve, auc, confusion_matrix, classification_report,accuracy_score
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

def auc_scorer(clf, X, y, model): # Helper function to plot the ROC curve
    if model=='RF':
        fpr, tpr, _ = roc_curve(y, clf.predict_proba(X)[:,1])
    elif model=='SVM':
        fpr, tpr, _ = roc_curve(y, clf.decision_function(X))
    roc_auc = auc(fpr, tpr)

    plt.figure()    # Plot the ROC curve
    plt.plot(fpr, tpr, label='ROC curve from '+model+' model (area = %0.3f)' % roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc="lower right")
    plt.show()

    return fpr,tpr,roc_auc
```

Figure 4.2: Loading of modules and helper functions

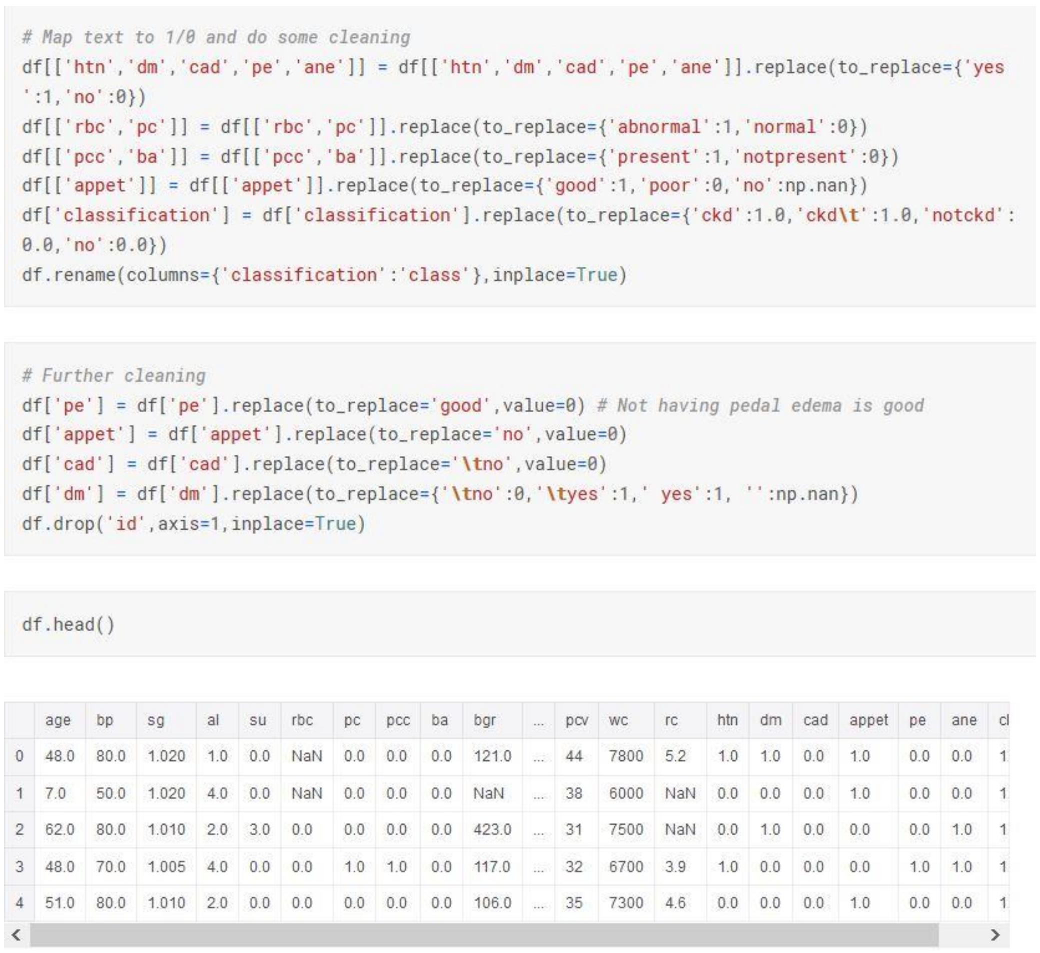
To load the dataset (saved as a .csv file) from our repository, I used the **pd.read_csv** command from the Pandas library.



```
df = pd.read_csv('../input/kidney_disease.csv')
```

Figure 4.3: Loading the csv file into the training model for the classifier

The data given is cleaned with NaN values removed and remaining data sorted out from the unwanted one. This will be providing us with more accuracy in the coming stages of the model training.



```
# Map text to 1/0 and do some cleaning
df[['htn','dm','cad','pe','ane']] = df[['htn','dm','cad','pe','ane']].replace(to_replace={'yes
':1,'no':0})
df[['rbc','pc']] = df[['rbc','pc']].replace(to_replace={'abnormal':1,'normal':0})
df[['pcc','ba']] = df[['pcc','ba']].replace(to_replace={'present':1,'notpresent':0})
df[['appet']] = df[['appet']].replace(to_replace={'good':1,'poor':0,'no':np.nan})
df['classification'] = df['classification'].replace(to_replace={'ckd':1.0,'ckd\t':1.0,'notckd':
0.0,'no':0.0})
df.rename(columns={'classification':'class'},inplace=True)
```

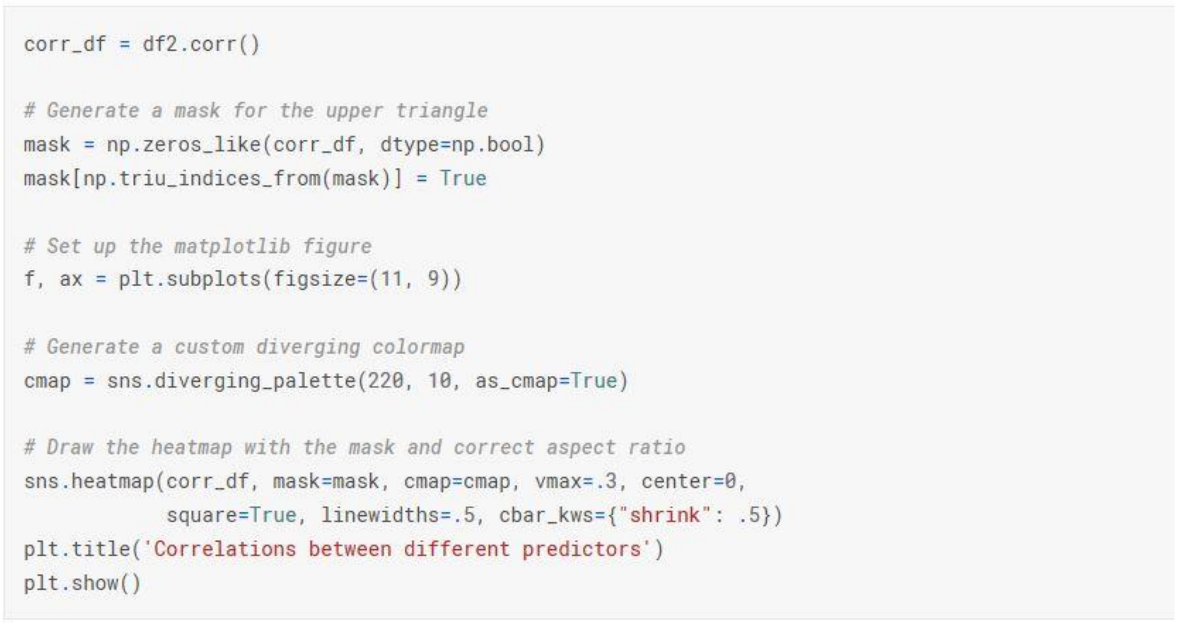
```
# Further cleaning
df['pe'] = df['pe'].replace(to_replace='good',value=0) # Not having pedal edema is good
df['appet'] = df['appet'].replace(to_replace='no',value=0)
df['cad'] = df['cad'].replace(to_replace='\tno',value=0)
df['dm'] = df['dm'].replace(to_replace={'\tno':0,'\tyes':1,' yes':1, '':np.nan})
df.drop('id',axis=1,inplace=True)
```

```
df.head()
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	cl
0	48.0	80.0	1.020	1.0	0.0	NaN	0.0	0.0	0.0	121.0	...	44	7800	5.2	1.0	1.0	0.0	1.0	0.0	0.0	1
1	7.0	50.0	1.020	4.0	0.0	NaN	0.0	0.0	0.0	NaN	...	38	6000	NaN	0.0	0.0	0.0	1.0	0.0	0.0	1
2	62.0	80.0	1.010	2.0	3.0	0.0	0.0	0.0	0.0	423.0	...	31	7500	NaN	0.0	1.0	0.0	0.0	0.0	1.0	1
3	48.0	70.0	1.005	4.0	0.0	0.0	1.0	1.0	0.0	117.0	...	32	6700	3.9	1.0	0.0	0.0	0.0	1.0	1.0	1
4	51.0	80.0	1.010	2.0	0.0	0.0	0.0	0.0	0.0	106.0	...	35	7300	4.6	0.0	0.0	0.0	1.0	0.0	0.0	1

Figure 4.4: Cleaning and pre-processing of data for the classifiers

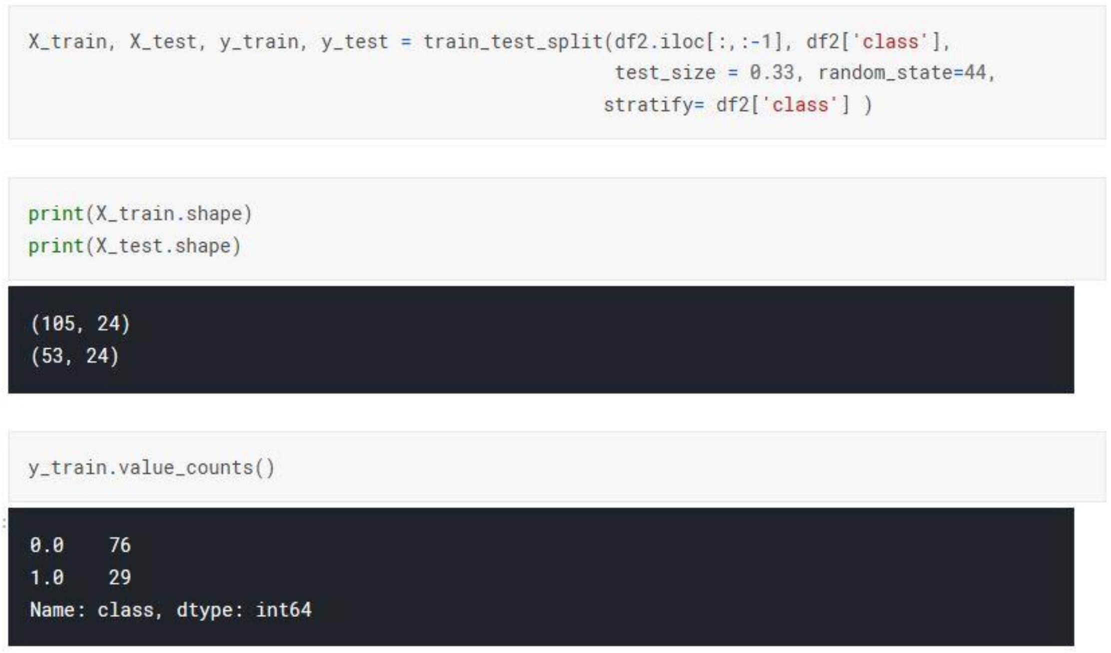
The cleansing of data is done for now and I will now be loading it up in the classifier and will be examining the correlations between different features to plot up a heatmap-based colormap graph.



```
corr_df = df2.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr_df, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_df, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.title('Correlations between different predictors')
plt.show()
```

Figure 4.5: Generating the Colormap Graph

After obtaining the results, I now split the set for training models into sub training and testing sets. I will then be choosing parameters with GridSearchCV with 10-fold cross validations.



```
X_train, X_test, y_train, y_test = train_test_split(df2.iloc[:,:-1], df2['class'],
                                                    test_size = 0.33, random_state=44,
                                                   stratify= df2['class'] )
```

```
print(X_train.shape)
print(X_test.shape)
```

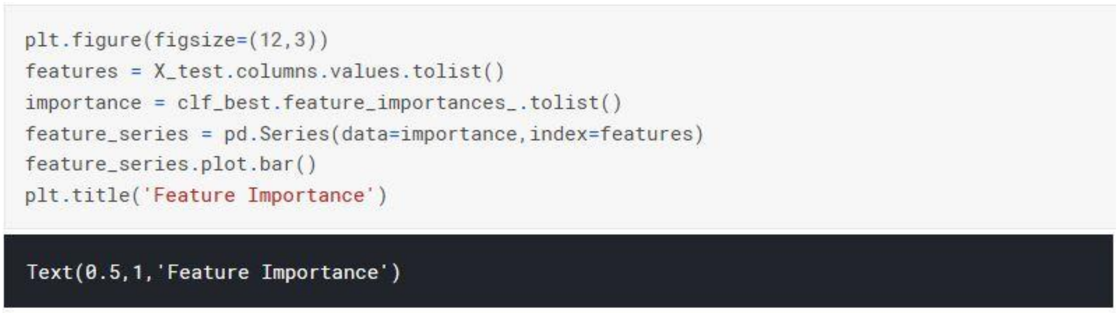
```
(105, 24)
(53, 24)
```

```
y_train.value_counts()
```

```
0.0    76
1.0    29
Name: class, dtype: int64
```

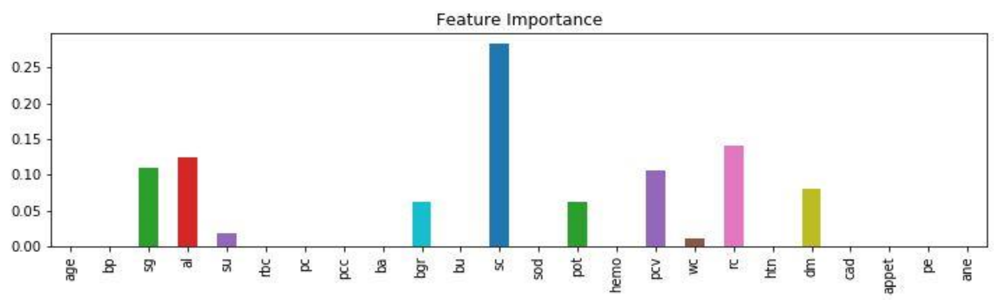
Figure 4.6: Splitting Data into Training and Testing Models

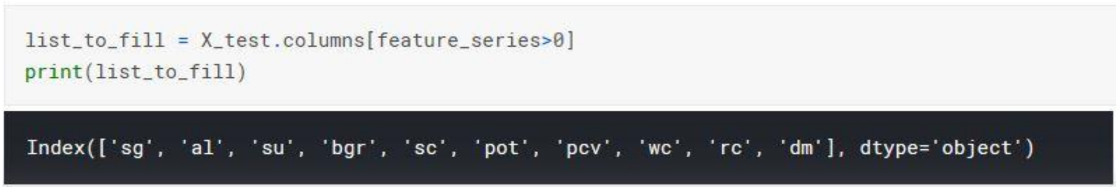
Next, I would be examining the feature importance to decrease the number of trees in the forest and also pruning it would be beneficial to narrow down the results for better accuracy, as all features are not used.



```
plt.figure(figsize=(12,3))
features = X_test.columns.values.tolist()
importance = clf_best.feature_importances_.tolist()
feature_series = pd.Series(data=importance,index=features)
feature_series.plot.bar()
plt.title('Feature Importance')
```

```
Text(0.5,1,'Feature Importance')
```





```
list_to_fill = X_test.columns[feature_series>0]
print(list_to_fill)
```

```
Index(['sg', 'al', 'su', 'bgr', 'sc', 'pot', 'pcv', 'wc', 'rc', 'dm'], dtype='object')
```

Figure 4.7: Feature Importance Selection

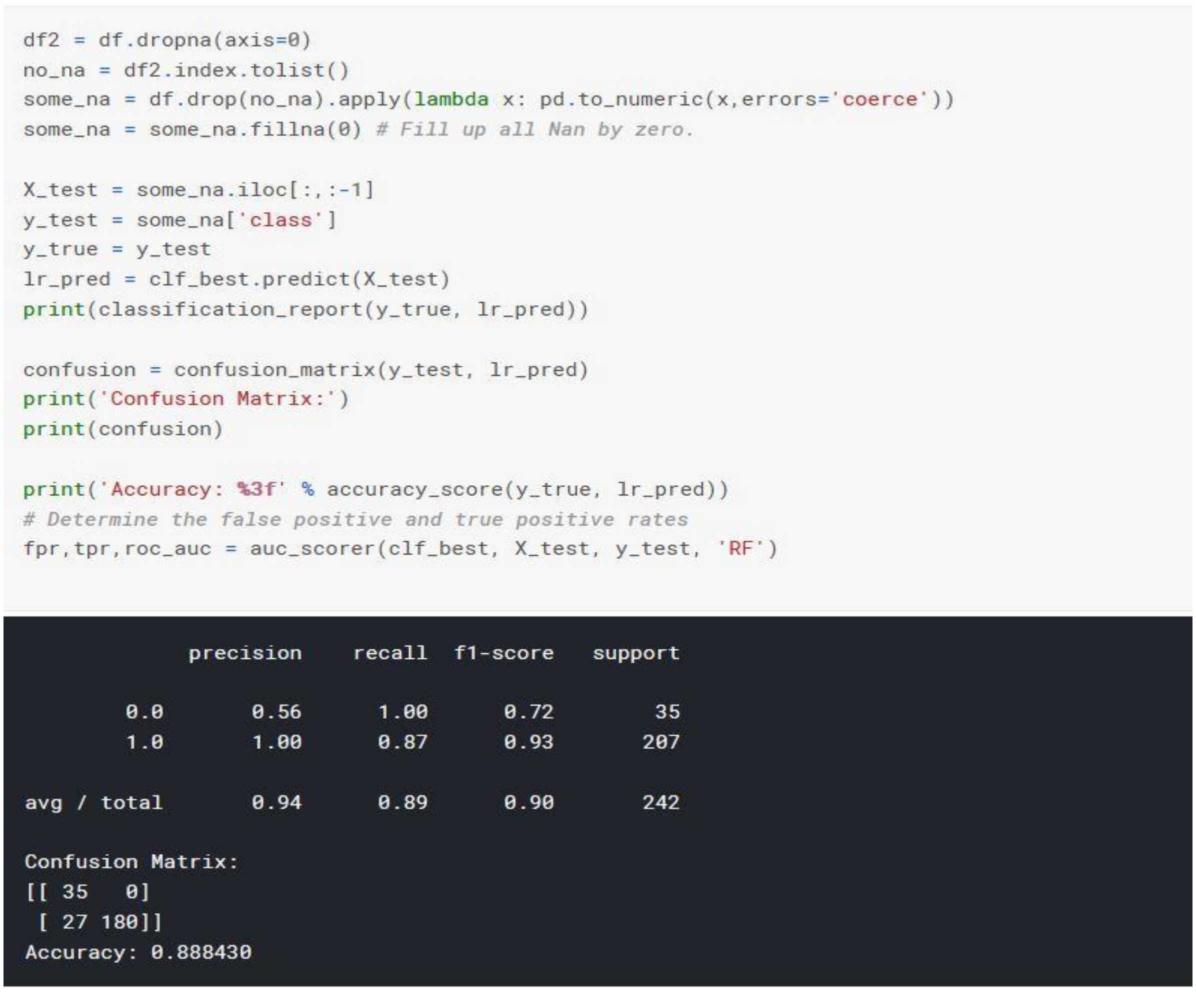
Lastly, I made predictions by obtaining the results from the selected model and formulating the confusion matrix. I filled in all NaN points with 0 and pass it to the trained classifier. The results were as follows:

Confusion Matrix:

[35 0]

[27 180]

Accuracy: 0.888430



```python
df2 = df.dropna(axis=0)
no_na = df2.index.tolist()
some_na = df.drop(no_na).apply(lambda x: pd.to_numeric(x,errors='coerce'))
some_na = some_na.fillna(0) # Fill up all Nan by zero.

X_test = some_na.iloc[:,:-1]
y_test = some_na['class']
y_true = y_test
lr_pred = clf_best.predict(X_test)
print(classification_report(y_true, lr_pred))

confusion = confusion_matrix(y_test, lr_pred)
print('Confusion Matrix:')
print(confusion)

print('Accuracy: %3f' % accuracy_score(y_true, lr_pred))
# Determine the false positive and true positive rates
fpr,tpr,roc_auc = auc_scorer(clf_best, X_test, y_test, 'RF')
```

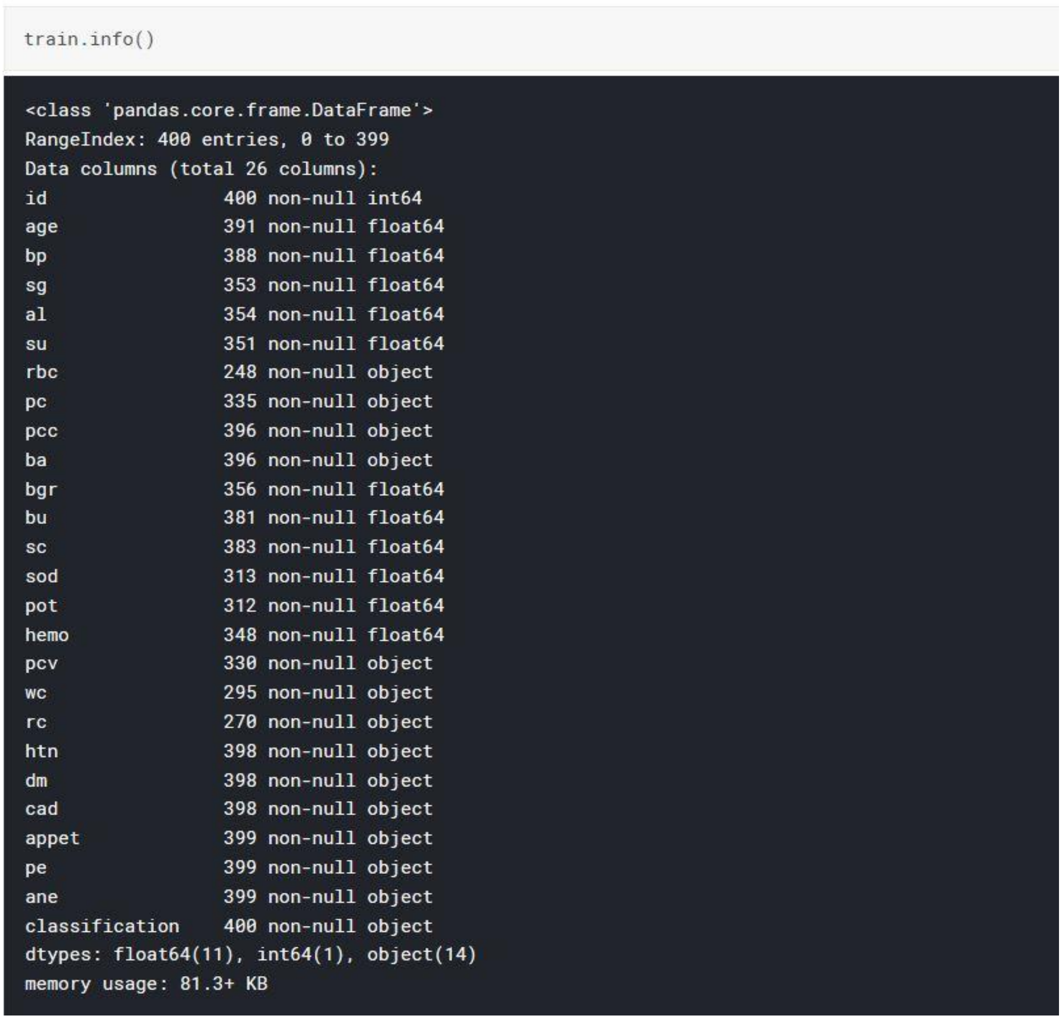
```
             precision    recall  f1-score   support

        0.0       0.56      1.00      0.72        35
        1.0       1.00      0.87      0.93       207

avg / total       0.94      0.89      0.90       242

Confusion Matrix:
[[ 35   0]
 [ 27 180]]
Accuracy: 0.888430
```

Figure 4.8: Making Predictions and Obtaining the Results

I repeated the same steps above for

Convolutional Deep Learning Neural Network (CDNN)

I trained the CDNN model with 80% training and 20% testing data.



```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
id                400 non-null int64
age               391 non-null float64
bp                388 non-null float64
sg                353 non-null float64
al                354 non-null float64
su                351 non-null float64
rbc               248 non-null object
pc                335 non-null object
pcc               396 non-null object
ba                396 non-null object
bgr               356 non-null float64
bu                381 non-null float64
sc                383 non-null float64
sod               313 non-null float64
pot               312 non-null float64
hemo              348 non-null float64
pcv               330 non-null object
wc                295 non-null object
rc                270 non-null object
htn               398 non-null object
dm                398 non-null object
cad               398 non-null object
appet             399 non-null object
pe                399 non-null object
ane               399 non-null object
classification    400 non-null object
dtypes: float64(11), int64(1), object(14)
memory usage: 81.3+ KB
```

Figure 4.10: Training the Convolutional Deep Learning Neural Network (CDNN)

Before the initial learning, I took out the heatmap of the fed training data to compare it later on when the training of the model is complete.

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
plt.grid()
plt.title("Number of Missing Values")
plt.savefig('missing.png')
```

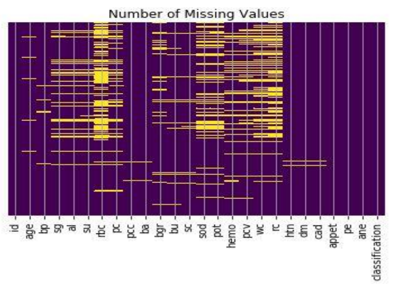
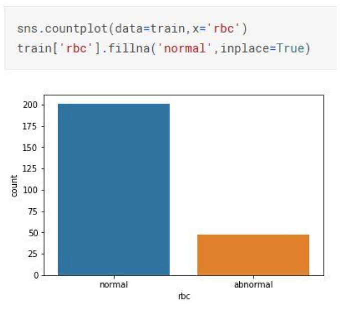


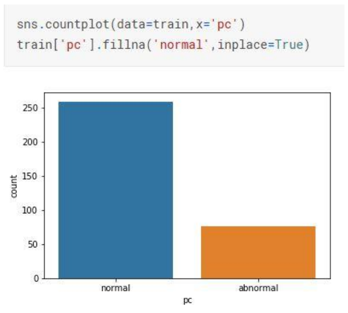
Figure 4.11: The heatmap showing all the 400 values as yellow lines

I now selected the 10 most important feature vectors out of the 25 in the list to proceed with the model training tasks and hence achieved 10 different graphs regarding the accuracy scores of the model on each feature versus the other.



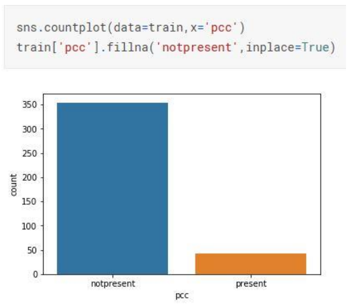
```
sns.countplot(data=train,x='rbc')
train['rbc'].fillna('normal',inplace=True)
```

Fig 4.12(a)



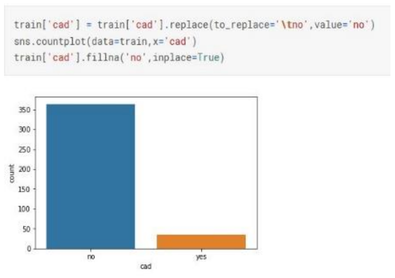
```
sns.countplot(data=train,x='pc')
train['pc'].fillna('normal',inplace=True)
```

Fig 4.12(b)



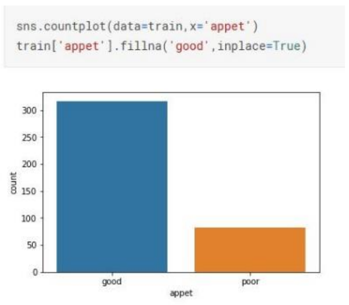
```
sns.countplot(data=train,x='pcc')
train['pcc'].fillna('notpresent',inplace=True)
```

Fig 4.12(c)



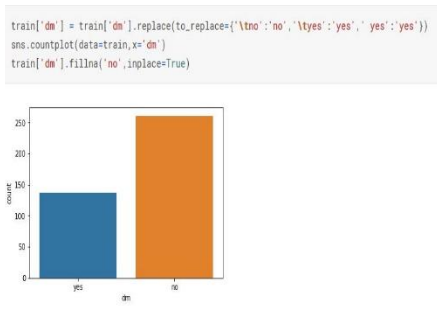
```
train['cad'] = train['cad'].replace(to_replace='\tno',value='no')
sns.countplot(data=train,x='cad')
train['cad'].fillna('no',inplace=True)
```

Fig 4.12(d)



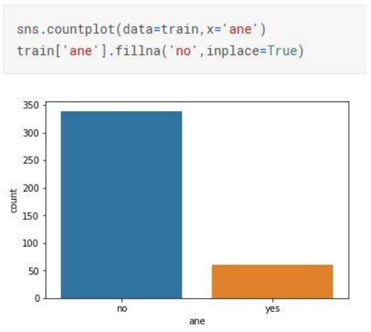
```
sns.countplot(data=train,x='appet')
train['appet'].fillna('good',inplace=True)
```

Fig 4.12(e)



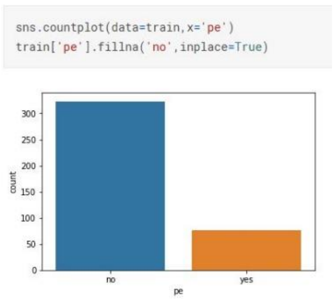
```
train['dm'] = train['dm'].replace(to_replace={'\tno':'no','\tyes':'yes',' yes':'yes'})
sns.countplot(data=train,x='dm')
train['dm'].fillna('no',inplace=True)
```

Fig 4.12(f)



```
sns.countplot(data=train,x='ane')
train['ane'].fillna('no',inplace=True)
```

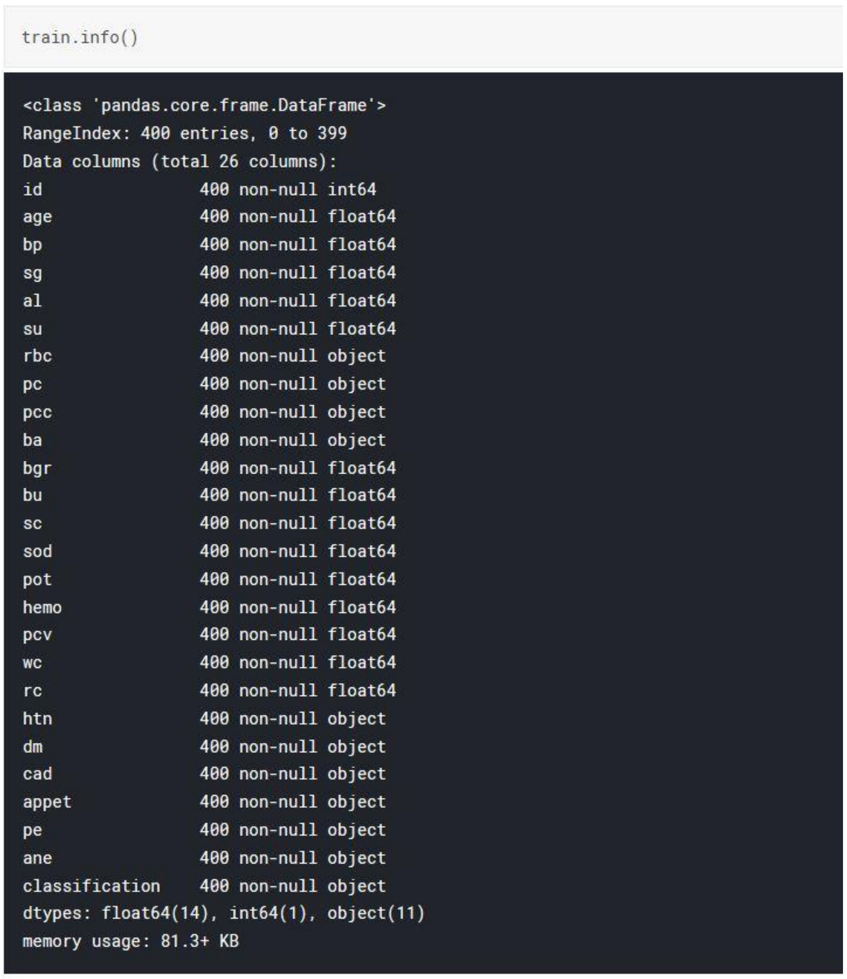
Fig 4.12(g)



```
sns.countplot(data=train,x='pe')
train['pe'].fillna('no',inplace=True)
```

Fig 4.12(h)

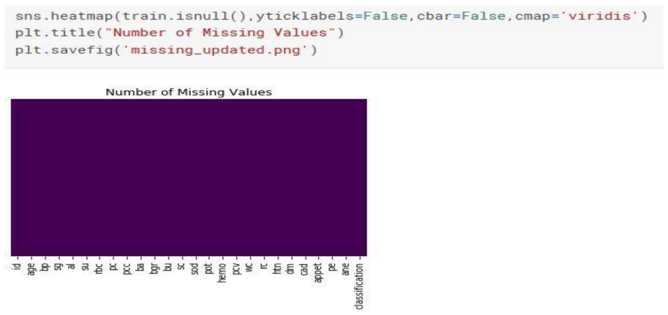
Training the model now results in a much cleaner and expected data formation in which there is no missing / left out data entry that can cause accuracy issues in the training model. The resulting heatmap is also blank which means no data value was stale or left behind in the pre-processing.



```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
id                400 non-null int64
age               400 non-null float64
bp                400 non-null float64
sg                400 non-null float64
al                400 non-null float64
su                400 non-null float64
rbc               400 non-null object
pc                400 non-null object
pcc               400 non-null object
ba                400 non-null object
bgr               400 non-null float64
bu                400 non-null float64
sc                400 non-null float64
sod               400 non-null float64
pot               400 non-null float64
hemo              400 non-null float64
pcv               400 non-null float64
wc                400 non-null float64
rc                400 non-null float64
htn               400 non-null object
dm                400 non-null object
cad               400 non-null object
appet             400 non-null object
pe                400 non-null object
ane               400 non-null object
classification    400 non-null object
dtypes: float64(14), int64(1), object(11)
memory usage: 81.3+ KB
```

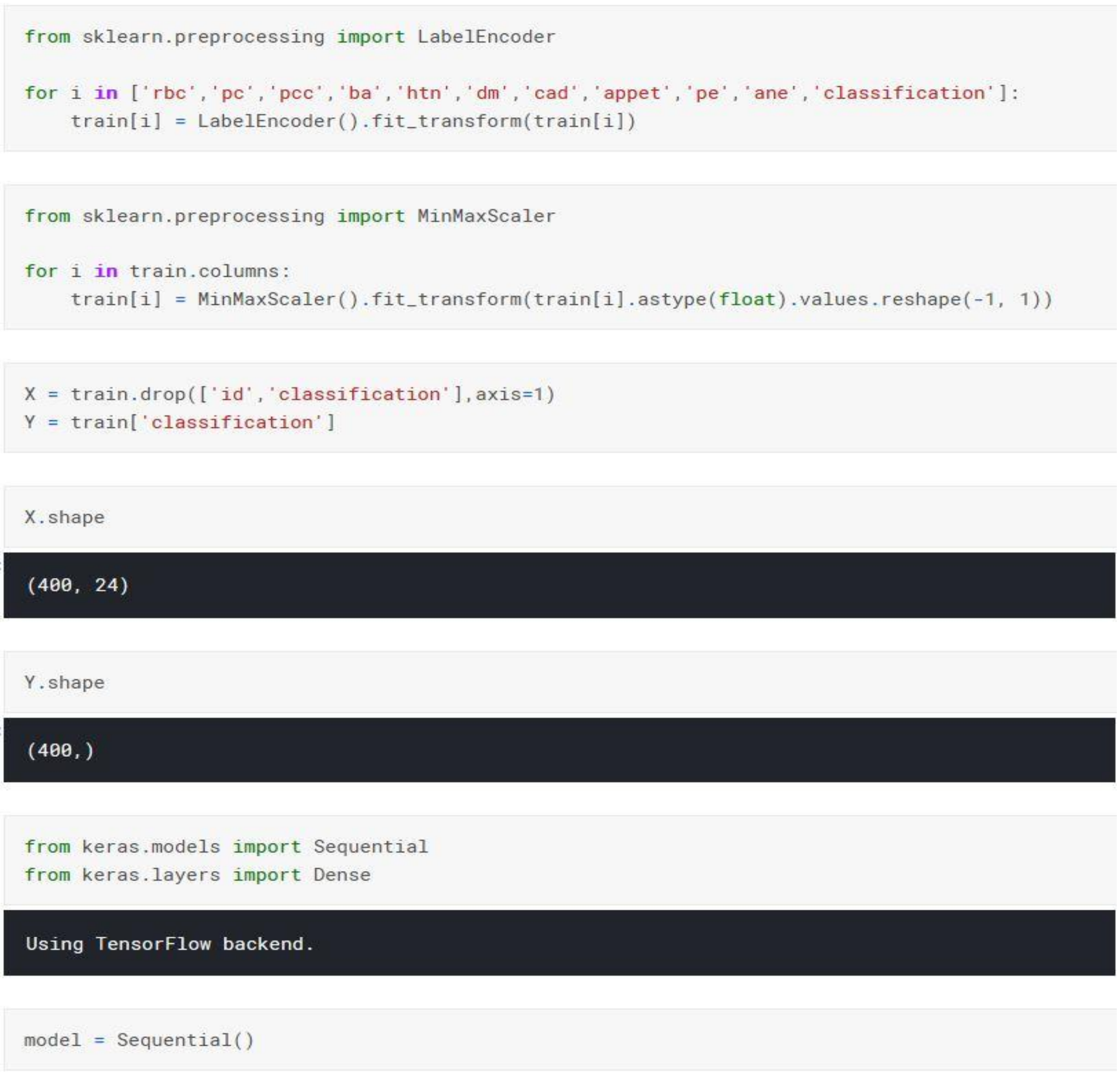
Figure 4.13: Trained CDNN after feeding data values



```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
plt.title("Number of Missing Values")
plt.savefig('missing_updated.png')
```

Figure 4.14: Blank heatmap (no yellow lines) means no unfiltered data was left behind

Finally, I use the **MinMaxScaler** and **LabelEncoder** from sklearn.preprocessing library from the sci-kit learn toolkit and collectively find out the results as 50 Epochs per run, training 320 samples with 80 for testing in a Sequential analysis TensorFlow environment.



```
from sklearn.preprocessing import LabelEncoder

for i in ['rbc','pc','pcc','ba','htn','dm','cad','appet','pe','ane','classification']:
    train[i] = LabelEncoder().fit_transform(train[i])
```

```
from sklearn.preprocessing import MinMaxScaler

for i in train.columns:
    train[i] = MinMaxScaler().fit_transform(train[i].astype(float).values.reshape(-1, 1))
```

```
X = train.drop(['id','classification'],axis=1)
Y = train['classification']
```

```
X.shape
```

```
(400, 24)
```

```
Y.shape
```

```
(400,)
```

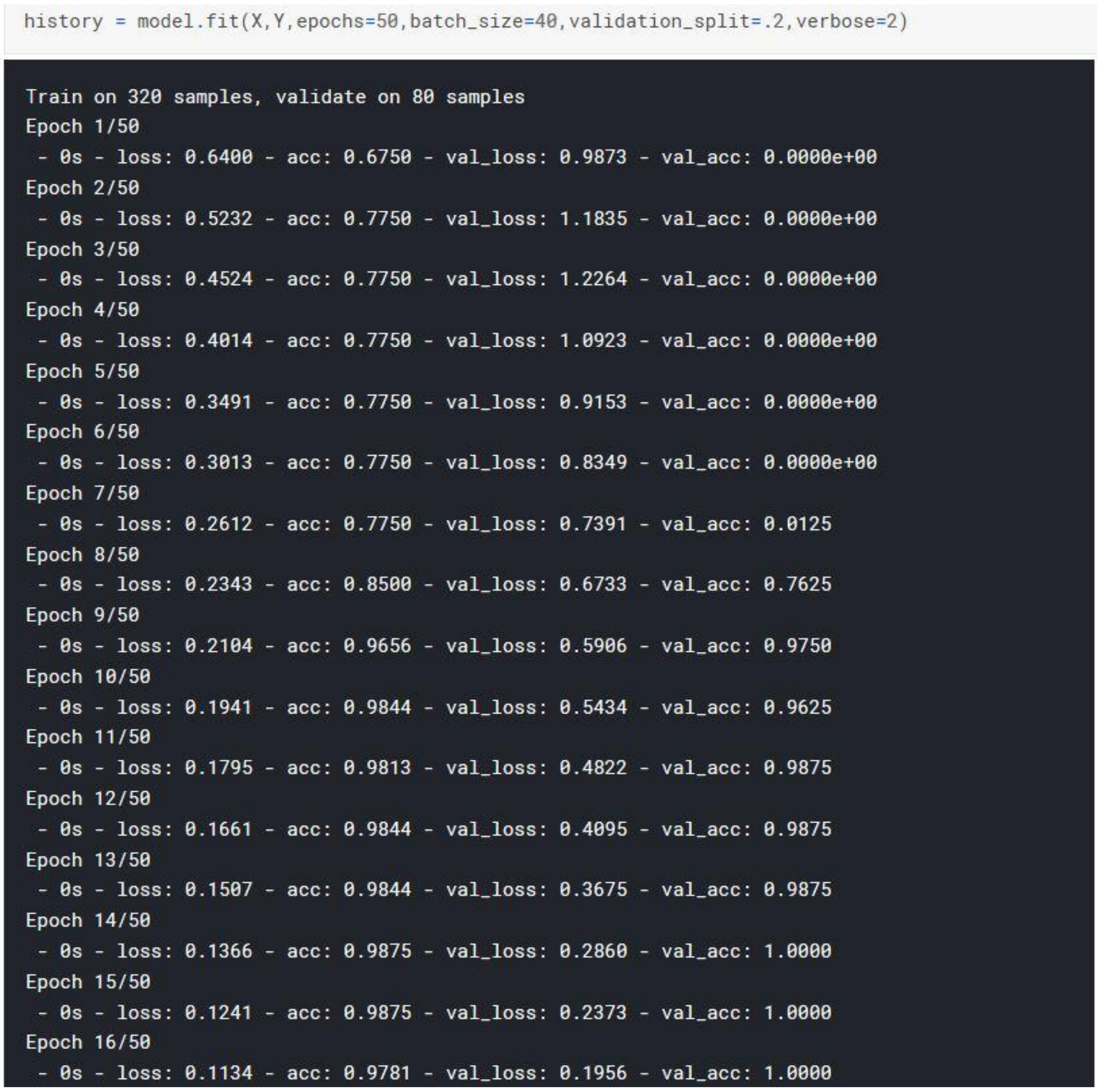
```
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

```
model = Sequential()
```

Figure 4.15: Using the MinMaxScaler and LabelEncoder

It was time for the results, and the accuracy was surprisingly high. I do notice that even after scaling, the results are still around same after the 16/50 Epoch.

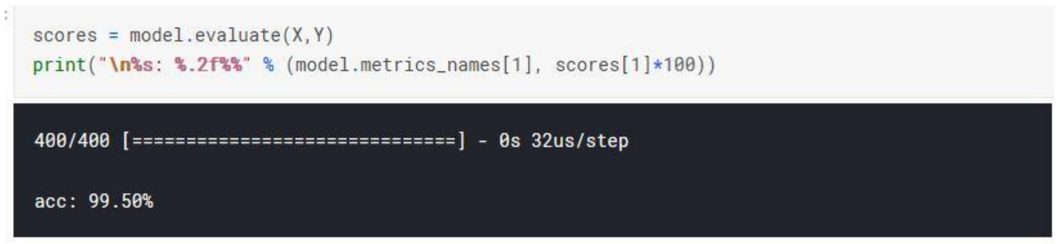


```
history = model.fit(X,Y,epochs=50,batch_size=40,validation_split=.2,verbose=2)
```

```
Train on 320 samples, validate on 80 samples
Epoch 1/50
 - 0s - loss: 0.6400 - acc: 0.6750 - val_loss: 0.9873 - val_acc: 0.0000e+00
Epoch 2/50
 - 0s - loss: 0.5232 - acc: 0.7750 - val_loss: 1.1835 - val_acc: 0.0000e+00
Epoch 3/50
 - 0s - loss: 0.4524 - acc: 0.7750 - val_loss: 1.2264 - val_acc: 0.0000e+00
Epoch 4/50
 - 0s - loss: 0.4014 - acc: 0.7750 - val_loss: 1.0923 - val_acc: 0.0000e+00
Epoch 5/50
 - 0s - loss: 0.3491 - acc: 0.7750 - val_loss: 0.9153 - val_acc: 0.0000e+00
Epoch 6/50
 - 0s - loss: 0.3013 - acc: 0.7750 - val_loss: 0.8349 - val_acc: 0.0000e+00
Epoch 7/50
 - 0s - loss: 0.2612 - acc: 0.7750 - val_loss: 0.7391 - val_acc: 0.0125
Epoch 8/50
 - 0s - loss: 0.2343 - acc: 0.8500 - val_loss: 0.6733 - val_acc: 0.7625
Epoch 9/50
 - 0s - loss: 0.2104 - acc: 0.9656 - val_loss: 0.5906 - val_acc: 0.9750
Epoch 10/50
 - 0s - loss: 0.1941 - acc: 0.9844 - val_loss: 0.5434 - val_acc: 0.9625
Epoch 11/50
 - 0s - loss: 0.1795 - acc: 0.9813 - val_loss: 0.4822 - val_acc: 0.9875
Epoch 12/50
 - 0s - loss: 0.1661 - acc: 0.9844 - val_loss: 0.4095 - val_acc: 0.9875
Epoch 13/50
 - 0s - loss: 0.1507 - acc: 0.9844 - val_loss: 0.3675 - val_acc: 0.9875
Epoch 14/50
 - 0s - loss: 0.1366 - acc: 0.9875 - val_loss: 0.2860 - val_acc: 1.0000
Epoch 15/50
 - 0s - loss: 0.1241 - acc: 0.9875 - val_loss: 0.2373 - val_acc: 1.0000
Epoch 16/50
 - 0s - loss: 0.1134 - acc: 0.9781 - val_loss: 0.1956 - val_acc: 1.0000
```

Figure 4.16: The results of 16/50 Epoch runs resulting in **1.0000** accuracy

I then got the final total accuracy for the neural network and analysed it with the resultant graphs.



```
scores = model.evaluate(X,Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
400/400 [==============================] - 0s 32us/step

acc: 99.50%
```

Figure 4.17: Final Accuracy Resulting to **99.50%**

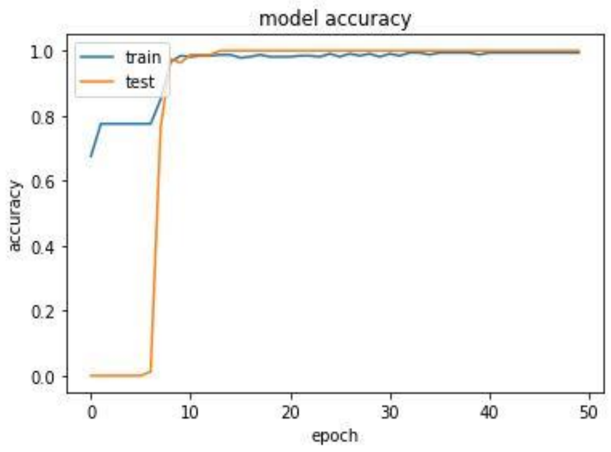


Figure 4.18(a): Accuracy Graph

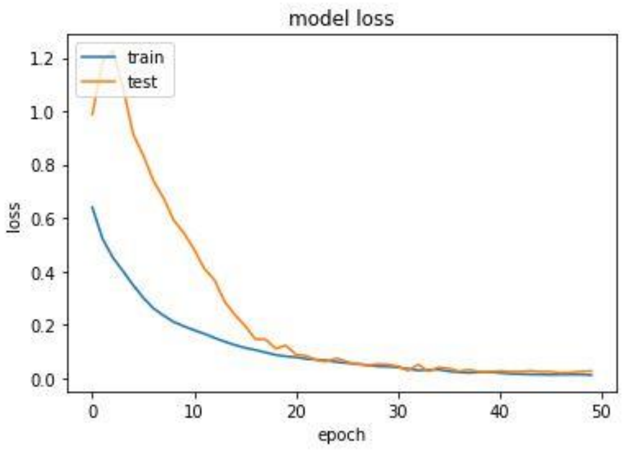


Figure 4.18(b): Loss Graph

The final algorithm I will be using will be:

Artificial Neural Network (ANN)

I use the same steps as the CDNN but the only difference would be in the selection criteria of vectors and their analysis. This is displayed by the set of images below.

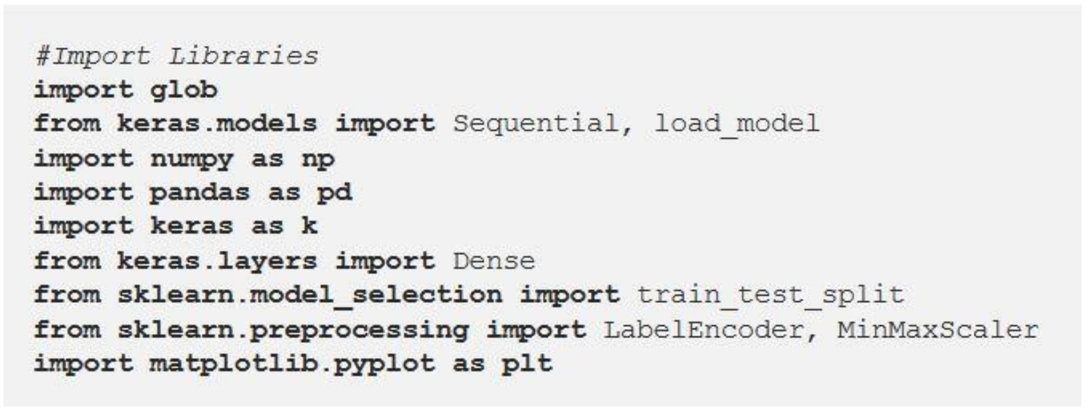
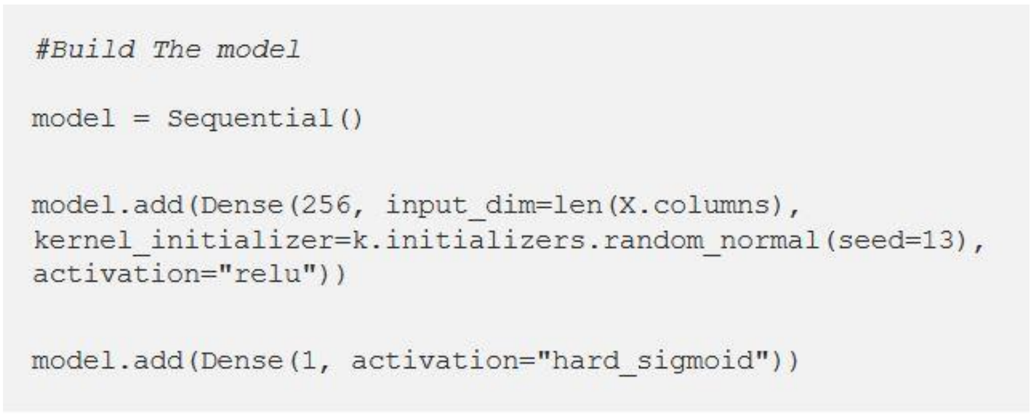
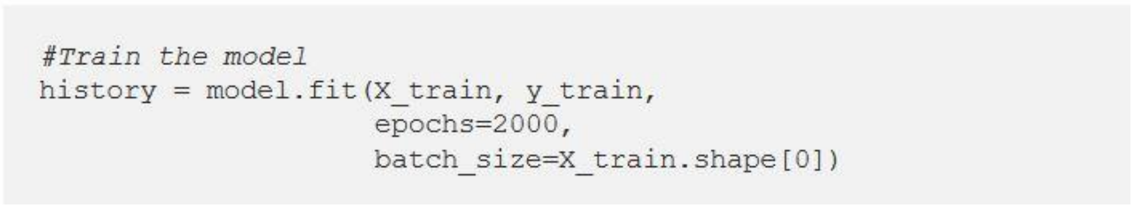

```
#Import Libraries
import glob
from keras.models import Sequential, load_model
import numpy as np
import pandas as pd
import keras as k
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import matplotlib.pyplot as plt
```

Figure 4.19: Importing Libraries



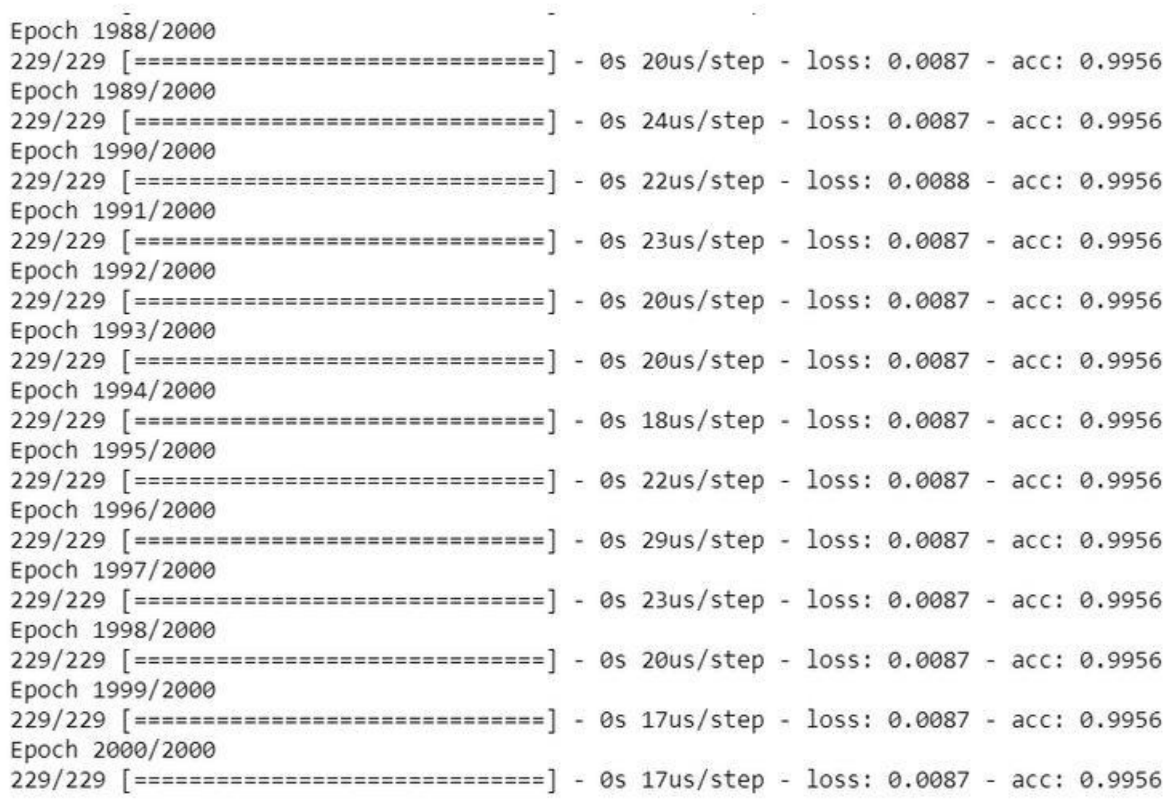
```
#Build The model

model = Sequential()


model.add(Dense(256, input_dim=len(X.columns),
kernel_initializer=k.initializers.random_normal(seed=13),
activation="relu"))


model.add(Dense(1, activation="hard_sigmoid"))
```

Figure 4.20: Building the ANN Model



```
#Train the model
history = model.fit(X_train, y_train,
                    epochs=2000,
                    batch_size=X_train.shape[0])
```

Figure 4.21: Training the Model



```
Epoch 1988/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1989/2000
229/229 [==============================] - 0s 24us/step - loss: 0.0087 - acc: 0.9956
Epoch 1990/2000
229/229 [==============================] - 0s 22us/step - loss: 0.0088 - acc: 0.9956
Epoch 1991/2000
229/229 [==============================] - 0s 23us/step - loss: 0.0087 - acc: 0.9956
Epoch 1992/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1993/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1994/2000
229/229 [==============================] - 0s 18us/step - loss: 0.0087 - acc: 0.9956
Epoch 1995/2000
229/229 [==============================] - 0s 22us/step - loss: 0.0087 - acc: 0.9956
Epoch 1996/2000
229/229 [==============================] - 0s 29us/step - loss: 0.0087 - acc: 0.9956
Epoch 1997/2000
229/229 [==============================] - 0s 23us/step - loss: 0.0087 - acc: 0.9956
Epoch 1998/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1999/2000
229/229 [==============================] - 0s 17us/step - loss: 0.0087 - acc: 0.9956
Epoch 2000/2000
229/229 [==============================] - 0s 17us/step - loss: 0.0087 - acc: 0.9956
```

Figure 4.22: Results of 2000 Epoch with accuracy **99.56%**

```
#Visualize the models accuracy and loss
plt.plot(history.history["acc"])
plt.plot(history.history["loss"])
plt.title("model accuracy & loss")
plt.ylabel("accuracy and loss")
plt.xlabel("epoch")
plt.legend(['acc', 'loss'], loc='lower right')
plt.show()
```

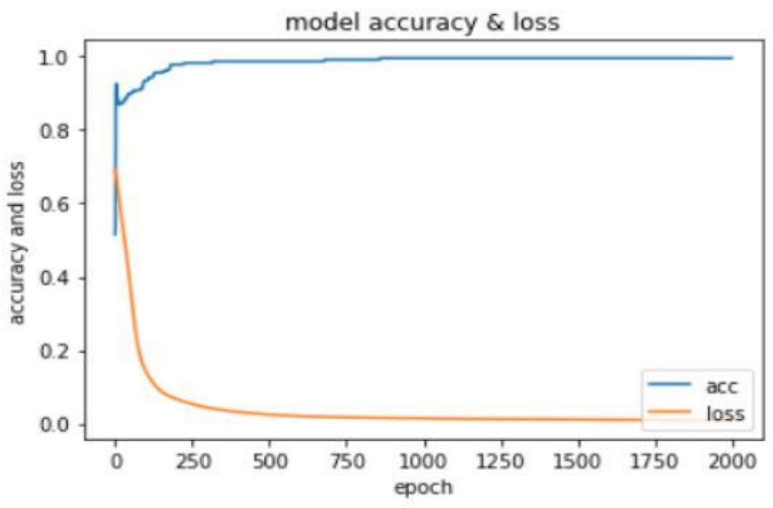
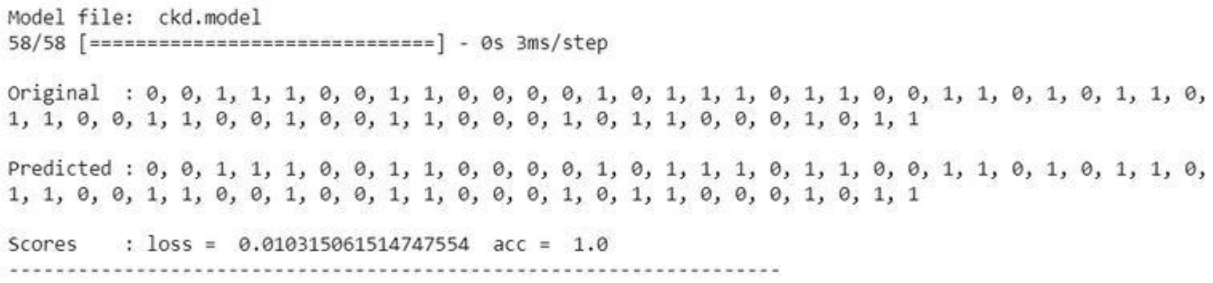


Fig 4: The models loss (orange) & accuracy (blue)



```
Model file:  ckd.model
58/58 [==============================] - 0s 3ms/step

Original  : 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1

Predicted : 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1

Scores    : loss =  0.010315061514747554  acc =  1.0
-------------------------------------------------------------------
```

Printing the model(s) output.

Figure 4.23: Results of the ANN model

As we see from the results of the ANN model, it gives an accuracy score of 99.56 going upwards to 99.997% (~1.00), which is by far the best result out of the three models I trained.

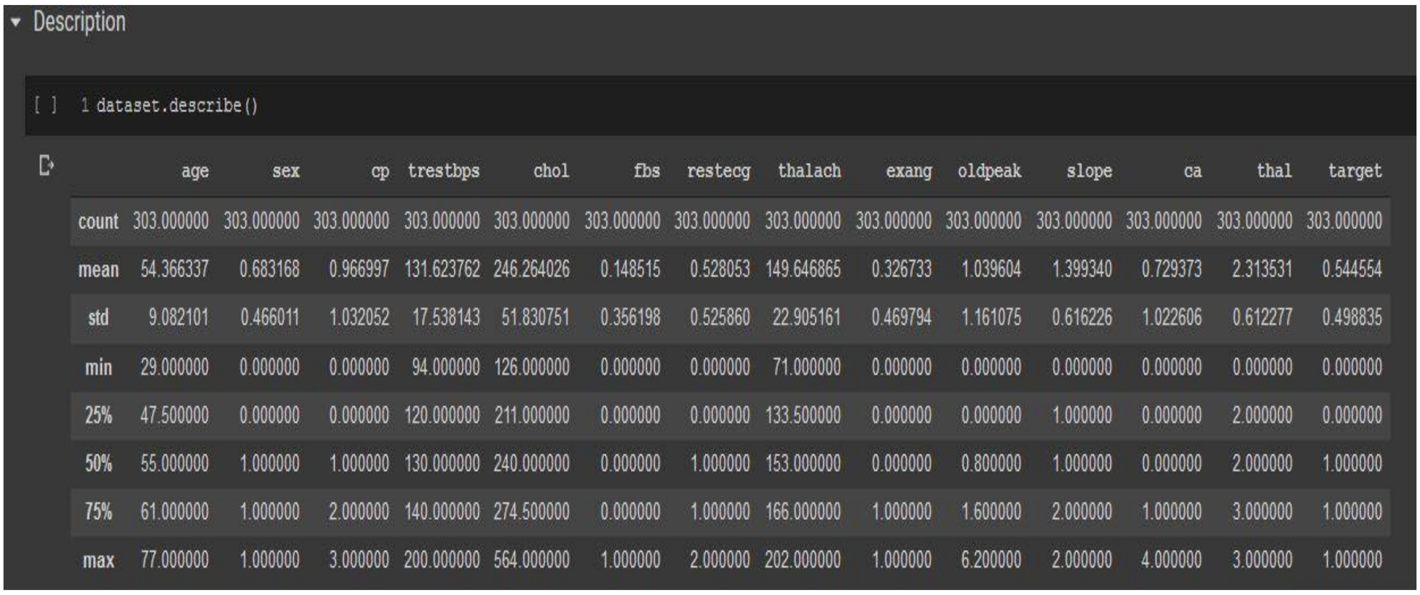
CHAPTER-5

IMPLEMENTATION (Heart Failure)

I moved onto my next organ being the most vital organs of all, the heart. This project involves analysis of the heart disease patient dataset with proper data processing. After studying about the various factors that are taken into consideration for the measurement of severity of the disease, we would be making a judgement based upon the given 14 heart related attributes of 303 patients to foresee the consequence (i.e. does one have CHD).

Decent data-driven systems aimed at foreseeing heart diseases can expand the entire study and preclusion procedure, making certain that more individuals can live in a good physical shape. Machine learning helps in predicting heart diseases, and these predictions made are quite accurate. I've used a variety of machine learning algorithms like KNN, Decision Tree, Random Forest, SVM, Logistic Regression etc. implemented in python, to predict the presence of heart disease in a patient.

This time onwards, I worked in a more streamlined manner so as to keep things clean and sorted in an orderly fashion.

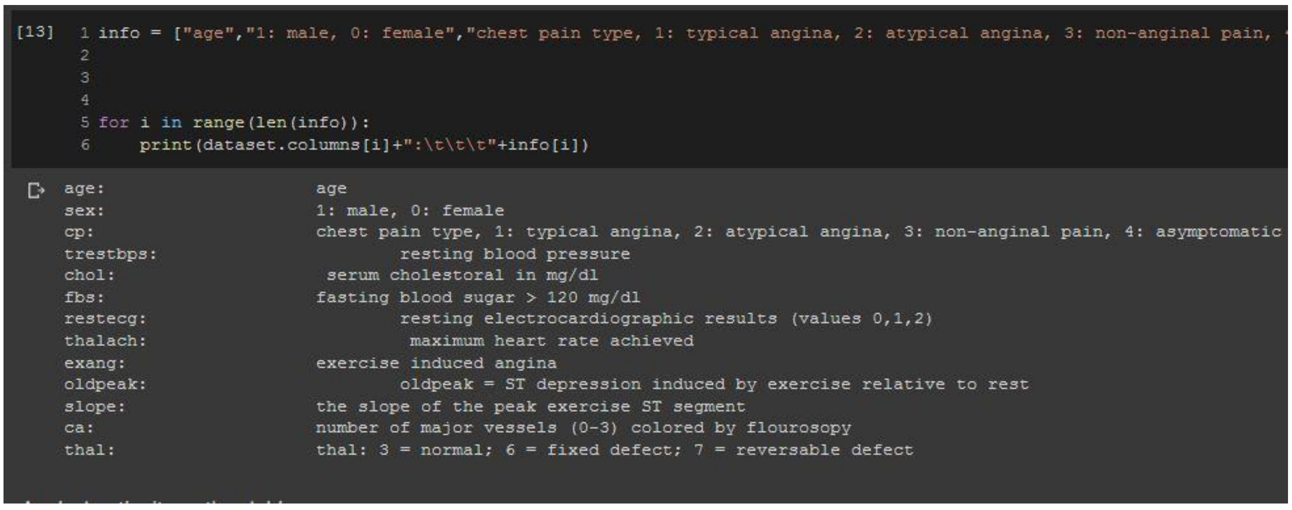

Description

```
dataset.describe()
```

```
              age         sex          cp    trestbps        chol         fbs     restecg     thalach       exang     oldpeak       slope          ca        thal      target
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean    54.366337    0.683168    0.966997  131.623762  246.264026    0.148515    0.528053  149.646865    0.326733    1.039604    1.399340    0.729373    2.313531    0.544554
std      9.082101    0.466011    1.032052   17.538143   51.830751    0.356198    0.525860   22.905161    0.469794    1.161075    0.616226    1.022606    0.612277    0.498835
min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000000    0.000000   71.000000    0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%     47.500000    0.000000    0.000000  120.000000  211.000000    0.000000    0.000000  133.500000    0.000000    0.000000    1.000000    0.000000    2.000000    0.000000
50%     55.000000    1.000000    1.000000  130.000000  240.000000    0.000000    1.000000  153.000000    0.000000    0.800000    1.000000    0.000000    2.000000    1.000000
75%     61.000000    1.000000    2.000000  140.000000  274.500000    0.000000    1.000000  166.000000    1.000000    1.600000    2.000000    1.000000    3.000000    1.000000
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000000    2.000000  202.000000    1.000000    6.200000    2.000000    4.000000    3.000000    1.000000
```

Figure 5.1: Dataset of 303 heart patients with 14 selective parameters

After uploading the .csv on my Google colab notebook, I did some basic operations on the dataset for a better understanding of it. From the operations I found out that I was working upon 303 entities with 14 vectors having one as a "target" variable which can take up a value of 0 or 1 accordingly.



```
[13]  info = ["age","1: male, 0: female","chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain,



      for i in range(len(info)):
          print(dataset.columns[i]+":\t\t\t"+info[i])
```

```
age:                    age
sex:                    1: male, 0: female
cp:                     chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic
trestbps:                       resting blood pressure
chol:                    serum cholestoral in mg/dl
fbs:                    fasting blood sugar > 120 mg/dl
restecg:                        resting electrocardiographic results (values 0,1,2)
thalach:                         maximum heart rate achieved
exang:                  exercise induced angina
oldpeak:                        oldpeak = ST depression induced by exercise relative to rest
slope:                  the slope of the peak exercise ST segment
ca:                     number of major vessels (0-3) colored by flourosopy
thal:                   thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
```

Figure 5.2: Description of the variables with the 'target'

The next few graphs will be portraying some of the above variables as a target and would be further judging the possibility of a heart disease being present, through graphs.

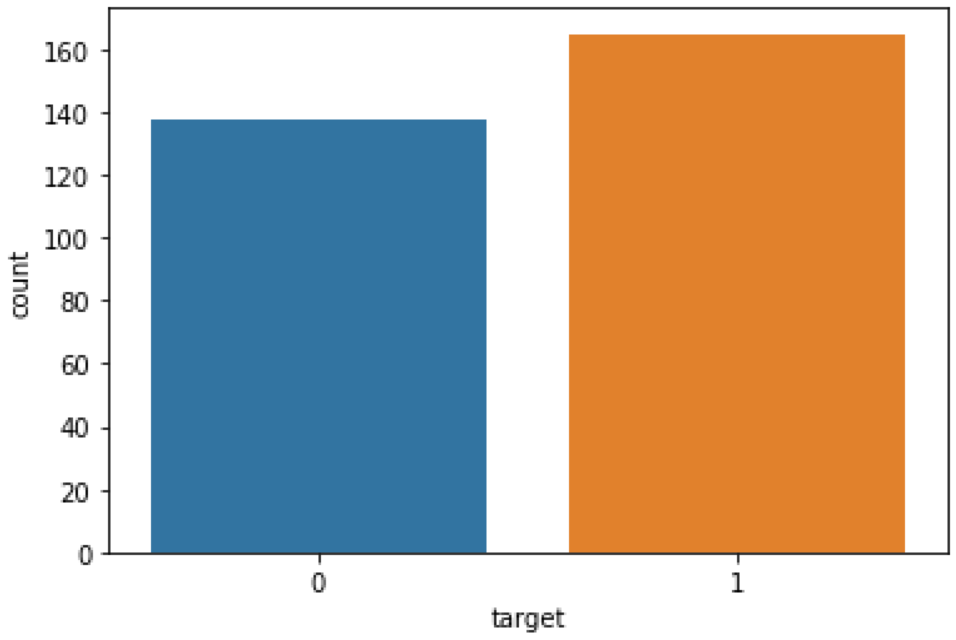


Figure 5.3: We notice that females are more likely to have heart disease than males

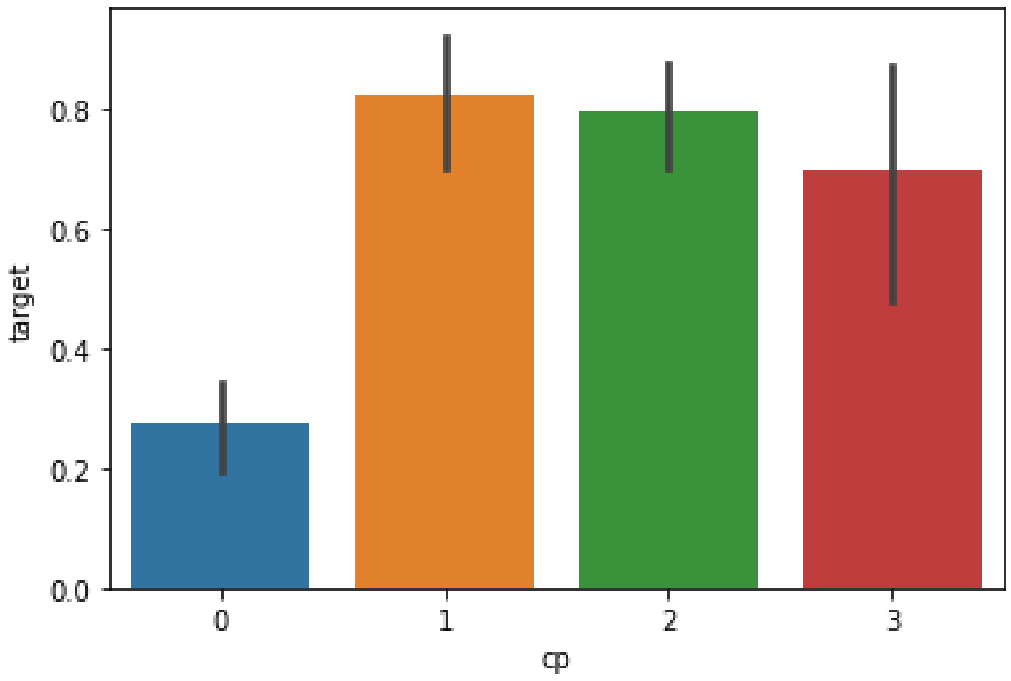


Figure 5.4: The chest pain '0' are the ones with typical angina who are less likely to have heart problems

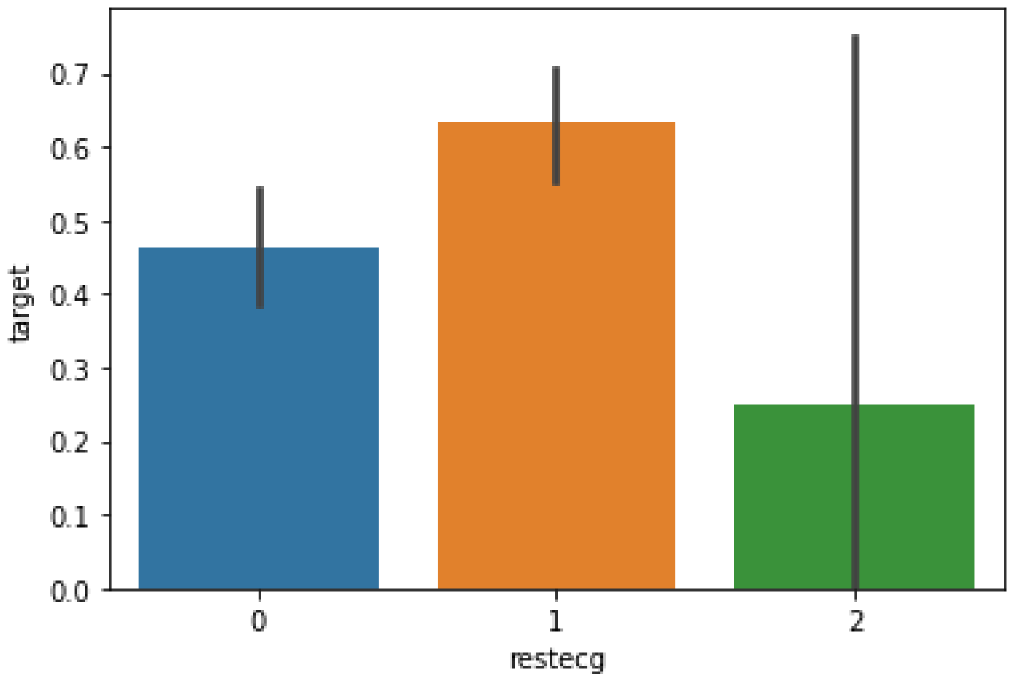


Figure 5.5: We realise that people with restecg '1' and '0' are much more likely to have a heart disease

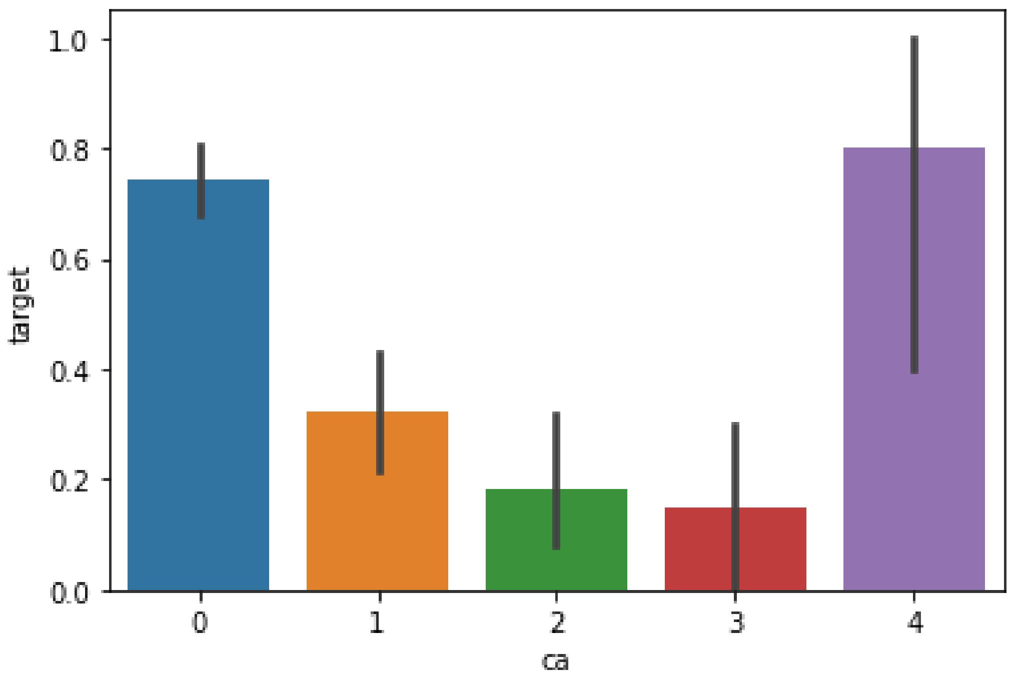
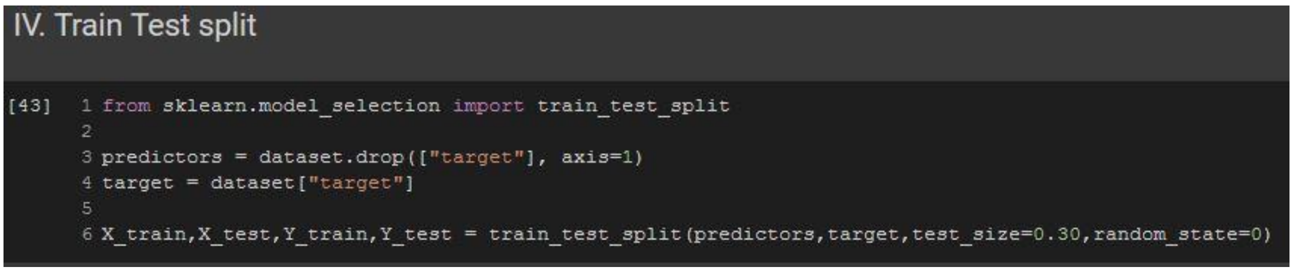


Figure 5.6: ca=4 has astonishingly large number of heart patients

As always, we split the data into test and training, keeping the latter in majority, to train various upcoming model algorithms with the data we have and sorted up. Always keeping a 70/30 or 60/40 train-test ratio helps.



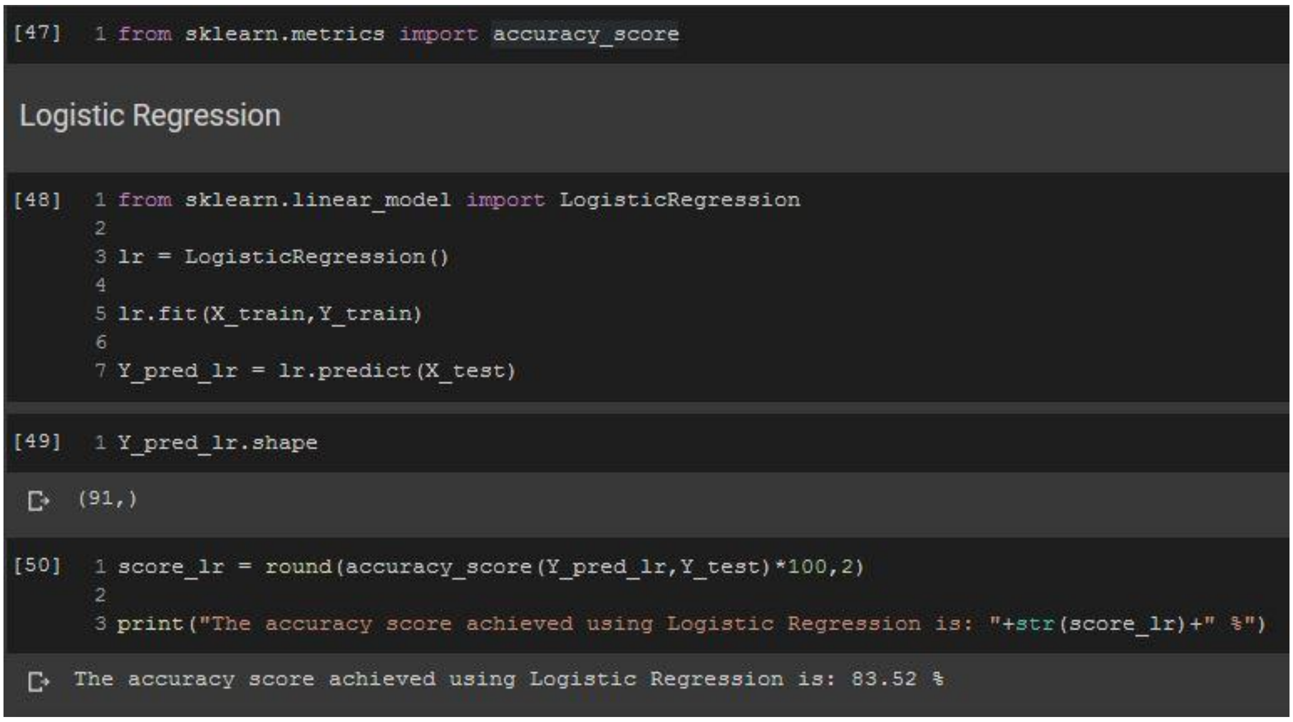
IV. Train Test split

```
[43]
from sklearn.model_selection import train_test_split

predictors = dataset.drop(["target"], axis=1)
target = dataset["target"]

X_train,X_test,Y_train,Y_test = train_test_split(predictors,target,test_size=0.30,random_state=0)
```

Figure 5.7: 70% training and 30% testing split

Moving on to the testing phase of our model, we implement several modern machine learning model algorithms, and thoroughly train the former with our training data, so as to see which one of them would be the most effective (highest accuracy in the least amount of average time).

Logistic Regression


```
[47] from sklearn.metrics import accuracy_score
```

Logistic Regression

```
[48] from sklearn.linear_model import LogisticRegression

     lr = LogisticRegression()

     lr.fit(X_train,Y_train)

     Y_pred_lr = lr.predict(X_test)
```

```
[49] Y_pred_lr.shape
```

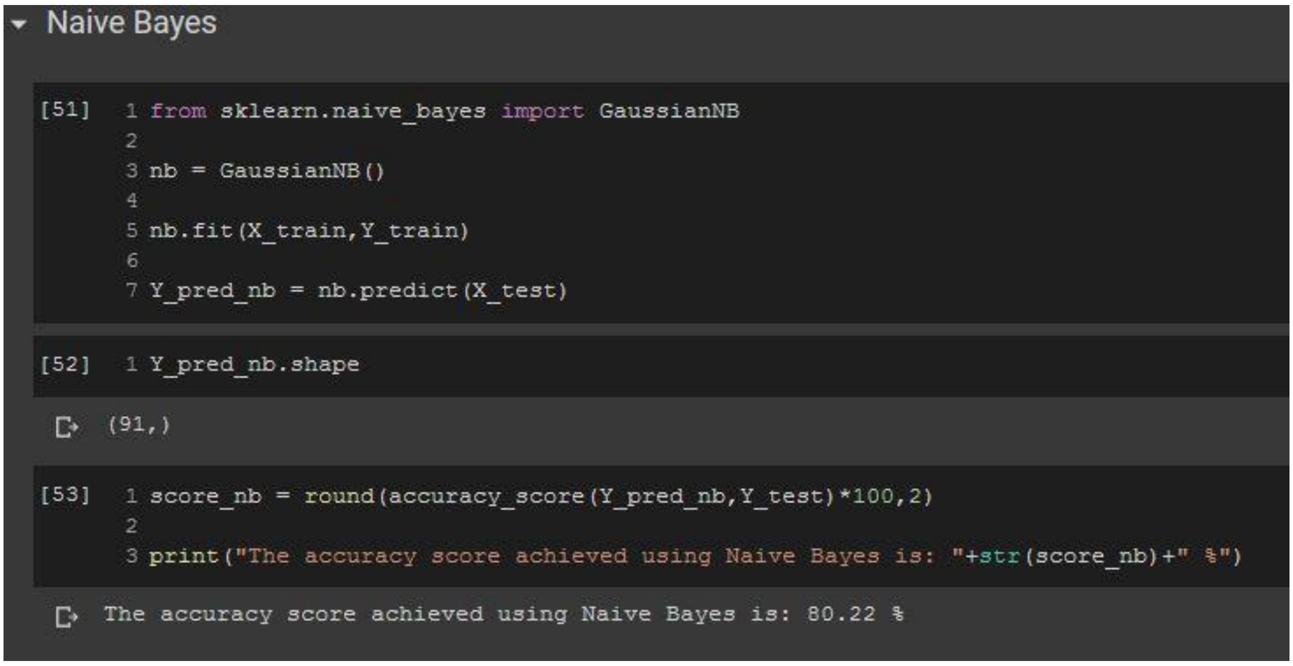
```
(91,)
```

```
[50] score_lr = round(accuracy_score(Y_pred_lr,Y_test)*100,2)

     print("The accuracy score achieved using Logistic Regression is: "+str(score_lr)+" %")
```

```
The accuracy score achieved using Logistic Regression is: 83.52 %
```

Figure 5.8: **Logistic Regression** gave us an accuracy score of **83.52%**

Naïve Bayes


Naive Bayes

```
[51] from sklearn.naive_bayes import GaussianNB

     nb = GaussianNB()

     nb.fit(X_train,Y_train)

     Y_pred_nb = nb.predict(X_test)
```

```
[52] Y_pred_nb.shape
```

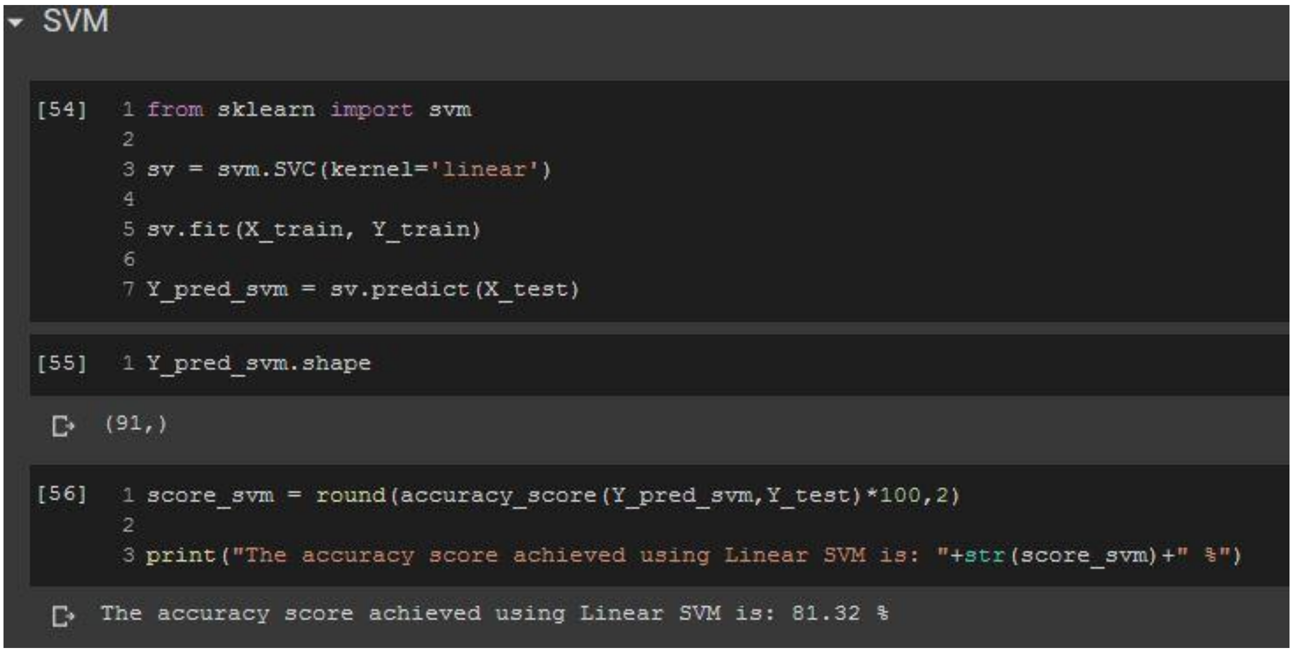
```
(91,)
```

```
[53] score_nb = round(accuracy_score(Y_pred_nb,Y_test)*100,2)

     print("The accuracy score achieved using Naive Bayes is: "+str(score_nb)+" %")
```

```
The accuracy score achieved using Naive Bayes is: 80.22 %
```

Figure 5.9: **Naïve Bayes** gave us an accuracy score of **80.22%**

Support Vector Machine (SVM)



SVM

```
[54]  from sklearn import svm

      sv = svm.SVC(kernel='linear')

      sv.fit(X_train, Y_train)

      Y_pred_svm = sv.predict(X_test)
```

```
[55]  Y_pred_svm.shape
```

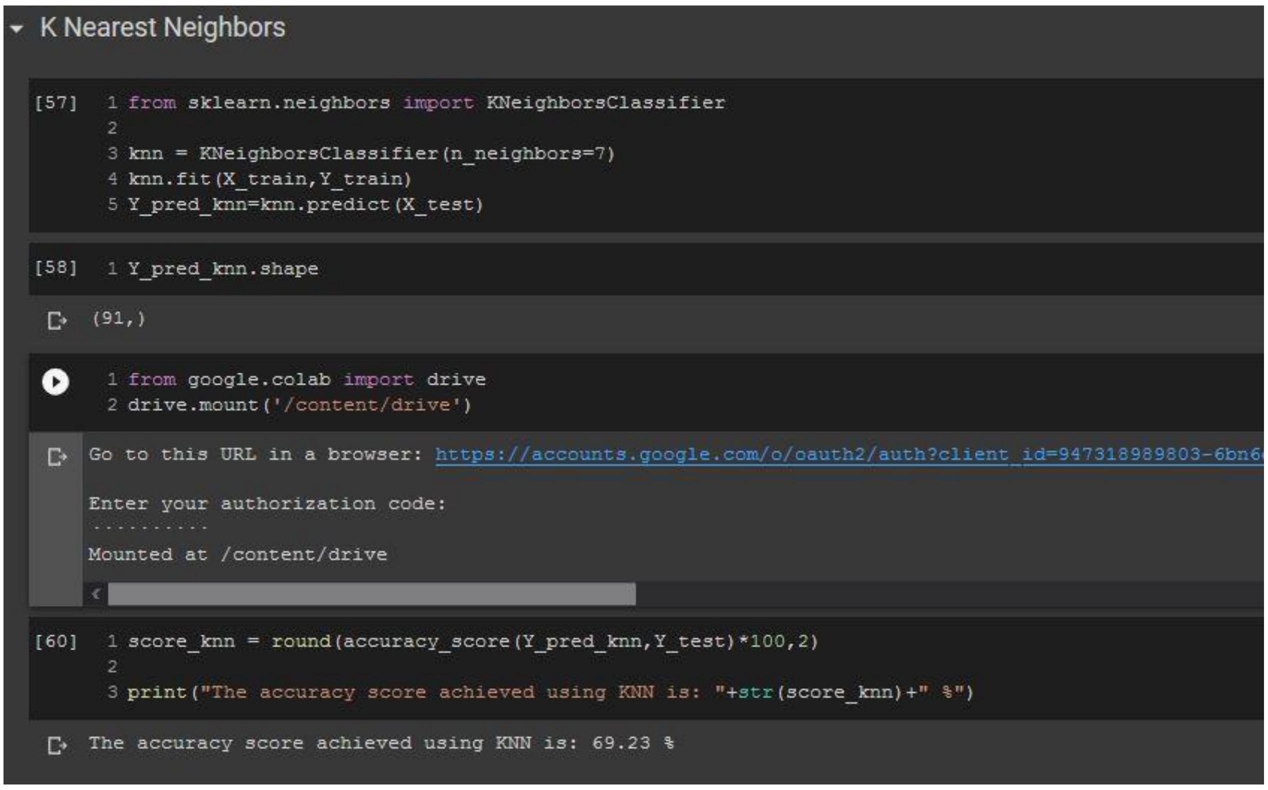
```
(91,)
```

```
[56]  score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)

      print("The accuracy score achieved using Linear SVM is: "+str(score_svm)+" %")
```

```
The accuracy score achieved using Linear SVM is: 81.32 %
```

Figure 5.10: **SVM** gave us an accuracy score of **81.32%**

K Nearest Neighbours (K-NN)



K Nearest Neighbors

```
[57]  from sklearn.neighbors import KNeighborsClassifier

      knn = KNeighborsClassifier(n_neighbors=7)
      knn.fit(X_train,Y_train)
      Y_pred_knn=knn.predict(X_test)
```

```
[58]  Y_pred_knn.shape
```

```
(91,)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

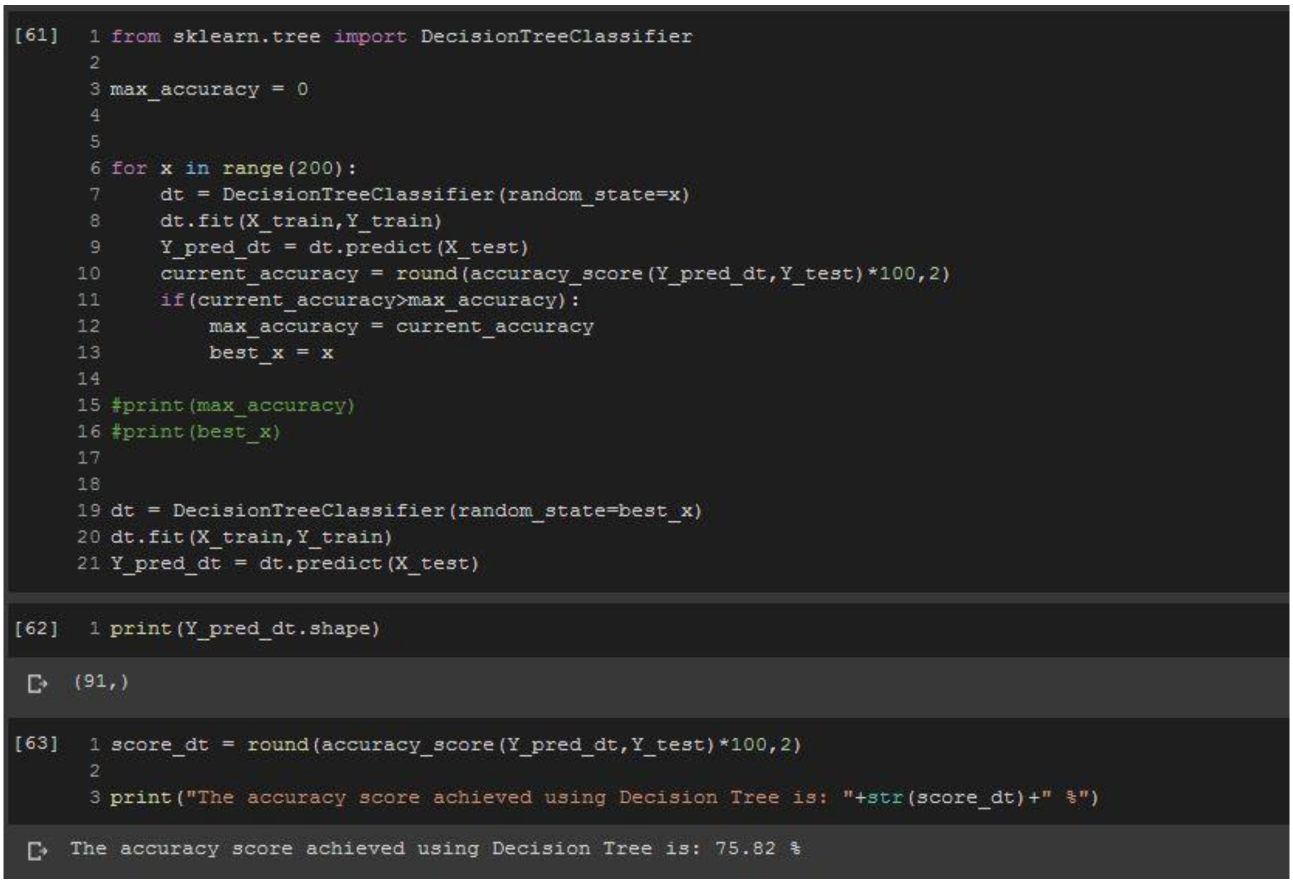
```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6

Enter your authorization code:
··········
Mounted at /content/drive
```

```
[60]  score_knn = round(accuracy_score(Y_pred_knn,Y_test)*100,2)

      print("The accuracy score achieved using KNN is: "+str(score_knn)+" %")
```

```
The accuracy score achieved using KNN is: 69.23 %
```

Figure 5.11: **K-NN** gave us an accuracy score of **69.23%**

Decision Tree



```
[61]
from sklearn.tree import DecisionTreeClassifier

max_accuracy = 0


for x in range(200):
    dt = DecisionTreeClassifier(random_state=x)
    dt.fit(X_train,Y_train)
    Y_pred_dt = dt.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

#print(max_accuracy)
#print(best_x)


dt = DecisionTreeClassifier(random_state=best_x)
dt.fit(X_train,Y_train)
Y_pred_dt = dt.predict(X_test)
```

```
[62]
print(Y_pred_dt.shape)
```

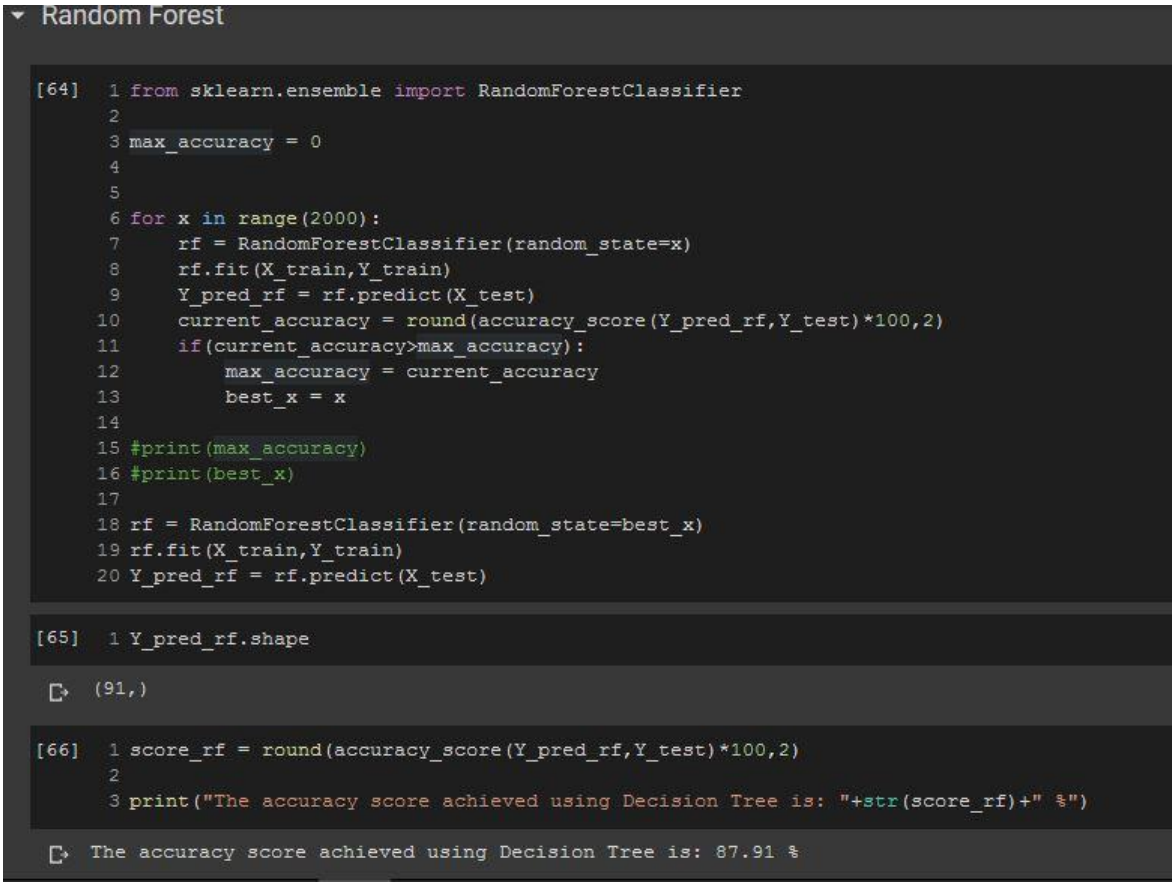
```
(91,)
```

```
[63]
score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)

print("The accuracy score achieved using Decision Tree is: "+str(score_dt)+" %")
```

```
The accuracy score achieved using Decision Tree is: 75.82 %
```

Figure 5.12: **Decision Tree** gave us an accuracy score of **75.82%**

Random Forest Classifier



```
[64]
from sklearn.ensemble import RandomForestClassifier

max_accuracy = 0


for x in range(2000):
    rf = RandomForestClassifier(random_state=x)
    rf.fit(X_train,Y_train)
    Y_pred_rf = rf.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

#print(max_accuracy)
#print(best_x)

rf = RandomForestClassifier(random_state=best_x)
rf.fit(X_train,Y_train)
Y_pred_rf = rf.predict(X_test)
```

```
[65]
Y_pred_rf.shape
```

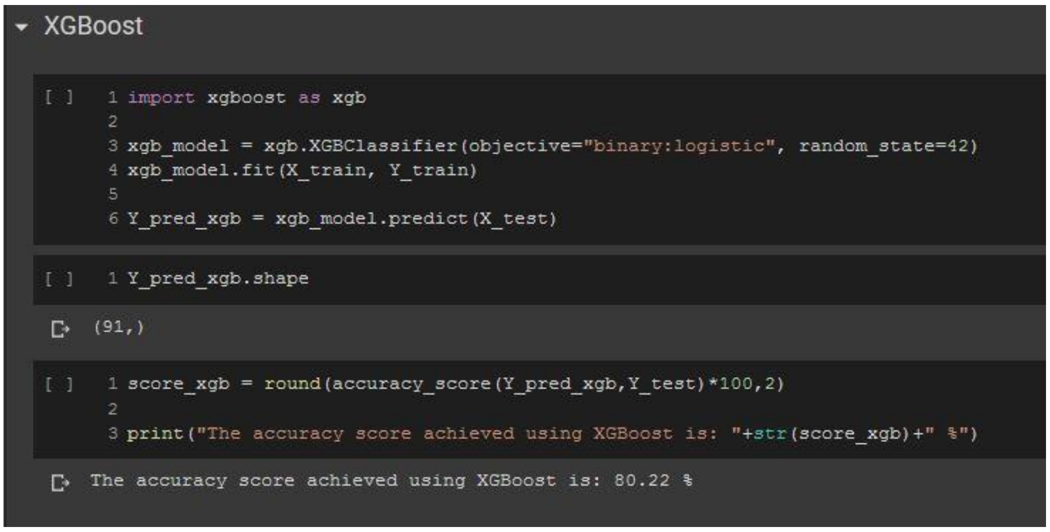
```
(91,)
```

```
[66]
score_rf = round(accuracy_score(Y_pred_rf,Y_test)*100,2)

print("The accuracy score achieved using Decision Tree is: "+str(score_rf)+" %")
```

```
The accuracy score achieved using Decision Tree is: 87.91 %
```

Figure 5.13: **Random Forest Classifier** gave us an accuracy score of **87.91%**

XGBoost



```
[ ]
import xgboost as xgb

xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(X_train, Y_train)

Y_pred_xgb = xgb_model.predict(X_test)
```

```
[ ]
Y_pred_xgb.shape
```

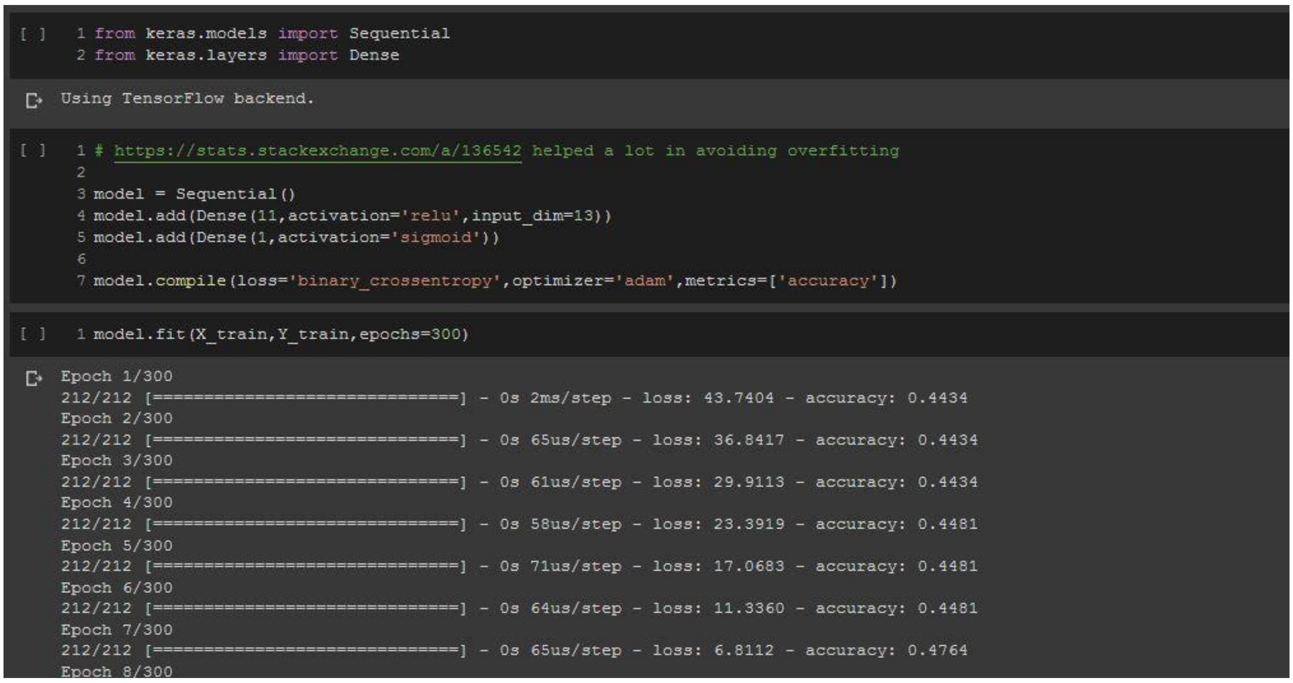
```
(91,)
```

```
[ ]
score_xgb = round(accuracy_score(Y_pred_xgb,Y_test)*100,2)

print("The accuracy score achieved using XGBoost is: "+str(score_xgb)+" %")
```

```
The accuracy score achieved using XGBoost is: 80.22 %
```

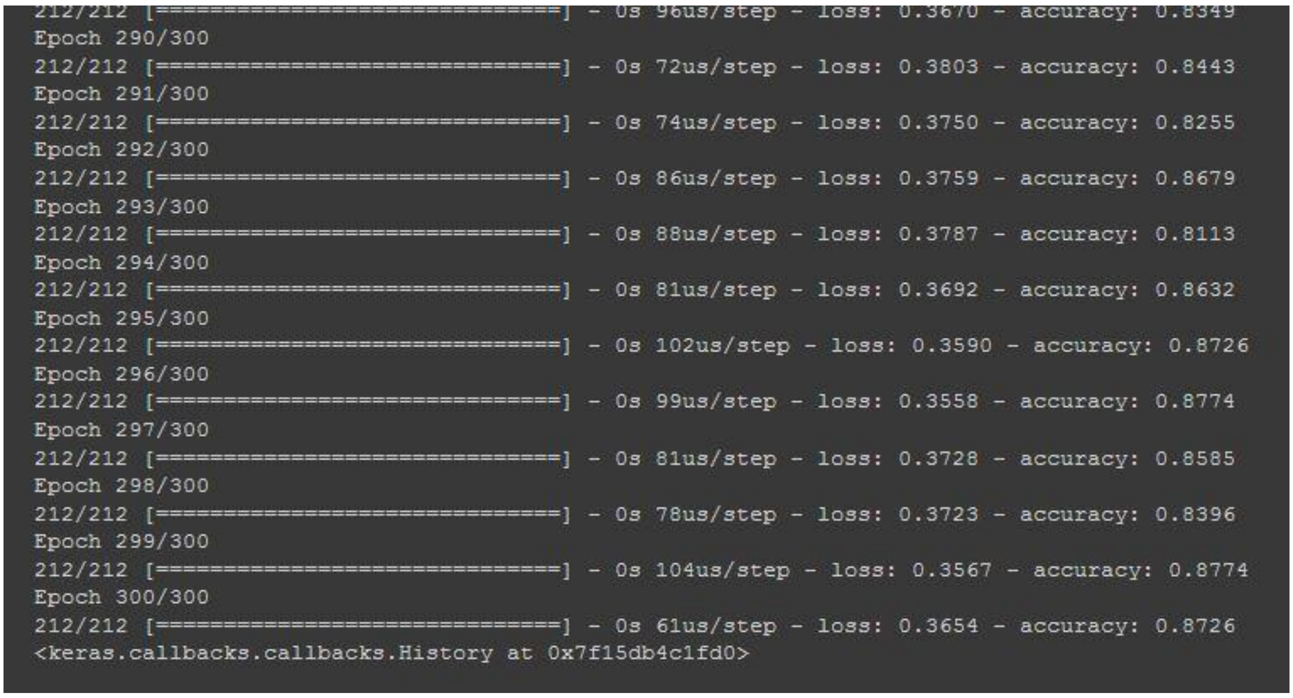
Figure 5.14: XGBoost gave us an accuracy score of **80.22%**

Neural Network



```
[ ]  from keras.models import Sequential
     from keras.layers import Dense

Using TensorFlow backend.

[ ]  # https://stats.stackexchange.com/a/136542 helped a lot in avoiding overfitting

     model = Sequential()
     model.add(Dense(11,activation='relu',input_dim=13))
     model.add(Dense(1,activation='sigmoid'))

     model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

[ ]  model.fit(X_train,Y_train,epochs=300)

Epoch 1/300
212/212 [==============================] - 0s 2ms/step - loss: 43.7404 - accuracy: 0.4434
Epoch 2/300
212/212 [==============================] - 0s 65us/step - loss: 36.8417 - accuracy: 0.4434
Epoch 3/300
212/212 [==============================] - 0s 61us/step - loss: 29.9113 - accuracy: 0.4434
Epoch 4/300
212/212 [==============================] - 0s 58us/step - loss: 23.3919 - accuracy: 0.4481
Epoch 5/300
212/212 [==============================] - 0s 71us/step - loss: 17.0683 - accuracy: 0.4481
Epoch 6/300
212/212 [==============================] - 0s 64us/step - loss: 11.3360 - accuracy: 0.4481
Epoch 7/300
212/212 [==============================] - 0s 65us/step - loss: 6.8112 - accuracy: 0.4764
Epoch 8/300
```

Figure 5.15: The Neural Network was running for 300 epochs through Keras and TensorFlow backend



```
212/212 [==============================] - 0s 96us/step - loss: 0.3670 - accuracy: 0.8349
Epoch 290/300
212/212 [==============================] - 0s 72us/step - loss: 0.3803 - accuracy: 0.8443
Epoch 291/300
212/212 [==============================] - 0s 74us/step - loss: 0.3750 - accuracy: 0.8255
Epoch 292/300
212/212 [==============================] - 0s 86us/step - loss: 0.3759 - accuracy: 0.8679
Epoch 293/300
212/212 [==============================] - 0s 88us/step - loss: 0.3787 - accuracy: 0.8113
Epoch 294/300
212/212 [==============================] - 0s 81us/step - loss: 0.3692 - accuracy: 0.8632
Epoch 295/300
212/212 [==============================] - 0s 102us/step - loss: 0.3590 - accuracy: 0.8726
Epoch 296/300
212/212 [==============================] - 0s 99us/step - loss: 0.3558 - accuracy: 0.8774
Epoch 297/300
212/212 [==============================] - 0s 81us/step - loss: 0.3728 - accuracy: 0.8585
Epoch 298/300
212/212 [==============================] - 0s 78us/step - loss: 0.3723 - accuracy: 0.8396
Epoch 299/300
212/212 [==============================] - 0s 104us/step - loss: 0.3567 - accuracy: 0.8774
Epoch 300/300
212/212 [==============================] - 0s 61us/step - loss: 0.3654 - accuracy: 0.8726
<keras.callbacks.callbacks.History at 0x7f15db4c1fd0>
```

Figure 5.16: Neural Network gave us an accuracy score of **81.32%**

CHAPTER-6

RESULTS AND ANALYSIS

I can now say that I successfully implemented and trained the three mentioned models for our chronic kidney disease dataset, namely:

Random Forest Classifier, CDNN and ANN

and for our chronic heart disease dataset, namely:

Logistic Regression, Naïve Bayes, SVM, K-NN, Decision Tree, Random Forest, XGBoost and Neural Network

The aforementioned models are tested on several constraints and topographies such as TN, TP, FN and FP and the measures of Exactness and Loss were considered.

For the kidney models, the results were as follows:

Classifier	**Accuracy**	**Loss**
Random Forest Classifier	88.843%	11.157%
CDNN	99.501%	0.497%
ANN	99.993%	0.007%

Table 6.1: Results of comparative analysis

After looking at these results, I settled on the argument that from all the algorithms that were tried upon our kidney disease dataset, **ANN (Artificial Neural Network)** gives out an accuracy of 99.993%

In consideration of the heart disease models, the results are as follows:

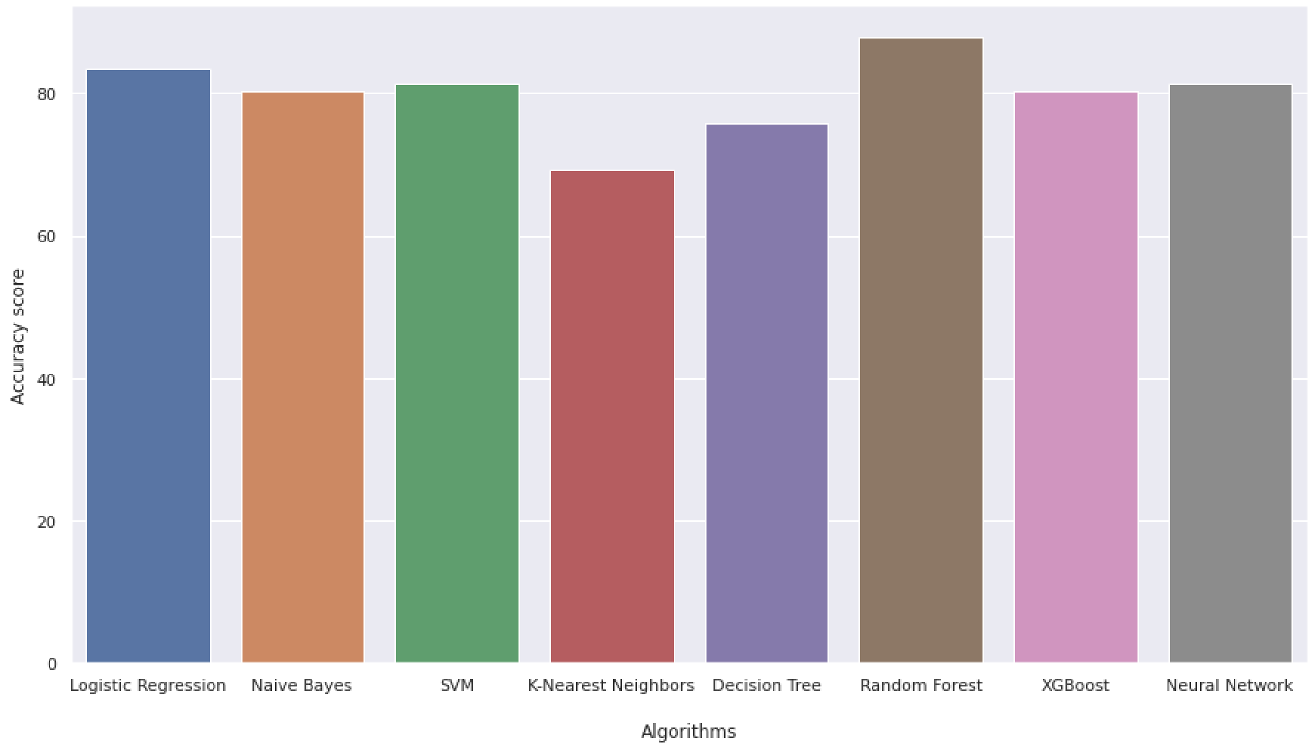


Figure 6.1: Final Results for the various algorithms

According to the results, the **Random Forest Classifier** comes at the top with an accuracy score of **87.91%**, with the **Logistic Regression** obtaining a 2nd place with an accuracy score of **83.52%**, with the **Neural Networks** and **SVM** at a tied score of **81.32%**, followed by a tie of the **Naïve Bayes** and **XGBoost** at **80.22%**, ending with the **Decision Trees** at **75.82%** and **K-NN** score the lowest at a 69.23%, proving to be least used in this case.

Below is a list of all the accuracy scores of the various algorithms used.

Classifier	Accuracy
Random Forest	**87.91%**
Logistic Regression	**83.52%**
Neural Networks	**81.32%**
SVM	**81.32%**
Naïve Bayes	**80.22%**
XGBoost	**80.22%**
Decision Trees	**75.82%**
K-NN	**69.23%**

Table 6.2: Final Results for the various algorithms

CHAPTER-7

CONCLUSION

In this project, I have actualized different AI algorithms particularly for discovery of Chronic Kidney Disorder (CKD) as well as the Chronic Heart Disease (CHD). For the classification, I used named datasets which were made using 25 of the selected attribute vectors contributing to various features of the kidney dataset, and 13 of the selected attribute vectors for the various features of the heart disease dataset. I was able to use learning algorithms upon these datasets by splitting into testing and training sets. Using the training portions, I considered the standards of different constraints and compared those features of all the different algorithms, concluding on these algorithms, provided the highest accuracy value is the **Artificial Neural Network (ANN)** for the kidney and the **Random Forest Classifier** for the heart. Thus, with an accuracy value of **99.993% (kidney)** and **87.91% (heart)**, these are the most appropriate algorithms for our present datasets. After getting the results, it is superlative to state that I obtained most of the samples of those diagnosed of CKD and CHD. Concluding, ML algorithms, particularly neural networks work best for the kidney and heart disease datasets.

FURTHER SCOPE OF THE PROJECT

After learning about the numerous ML algorithms and their subsequent application on the used datasets, I was highly motivated to work upon gathering data samples of various other organs of the human body so as to create a one-for-all machine model that can help in fast discovery and treatment of diseases (if any).

Due to the current worldwide pandemic of the Coronavirus COVID-19 and with millions of people suffering from it, I will be working upon the Lung Disease database mainly based upon the covid-19 results next, using these and various other algorithms to obtain highly accurate results just like our kidney failure and heart disease detection engines. I would be glad to work upon this project as a part of a larger industry on a bigger and more sophisticated scale, so as to help many in saving their lives and improving the industry standards.

REFERENCES

[1] Tahsim M. Rahman, S. Siddiqua, Siam -E- Rabby, “Early Detection of Kidney Diseases Using ECG Signals Through Machine Learning Based Modelling”. In the proceedings of the International Conference on Robotics, Electrical and Signal Processing Techniques. **(ICREST- 2019)**

[2] A. Abdelaziz, A. M. Riad, A. S. Salama, A. N. Mahmoud, “A Machine Learning Model for Predicting of Chronic Kidney Disease Based Internet of Things and Cloud Computing in Smart Cities” **(2019)**

[3] H. Mohamadlou, A. Lynn-Palevsky, C. Barton, U. Chettipally, L. Shieh, J. Calvert, N. R. Saber, R. Das, “Prediction of Acute Kidney Injury with a Machine Learning Algorithm Using Electronic Health Record Data” **(2018)**

[4] "Intro to Machine Learning | Udacity." Intro to Machine Learning | Udacity. [https://www.udacity.com/course/intro-to-machine-learning--ud120].

[5] "Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd Edition. Datasets: Coronary Heart Disease Dataset." Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd Edition. [http://statweb.stanford.edu/~tibs/ElemStatLearn/].

[6] Manju M N, Mayur G K, Shachi D S, Chandan J, “Student Performance Evaluation Using Predictive Analysis”. International Research Journal of Engineering and Technology (IRJET) **2020**

[7] A. Rairikar, V. Kulkarni, V. Sabale, “Heart Disease Prediction Using Data Mining Techniques”. International Conference on Intelligent Computing and Control (I2C2) **2017**

[8] "No Free Lunch Theorems." No Free Lunch Theorems. [http://www.no-free-lunch.org/].

[9] "https://www.sciencedirect.com/topics/engineering/confusion-matrix"

[10] "https://www.researchgate.net/publication/303326261_Machine_Learning_Project"

[11] "https://link.springer.com/book/10.1007%2F978-981-15-1216-2"

[12] "https://archive.ics.uci.edu/ml/datasets/Heart+Disease"

[13] "https://towardsdatascience.com/heart-disease-prediction-73468d630cfc"

[14] "https://medium.com/@dskswu/machine-learning-with-a-heart-predicting-heart-disease-b2e9f24fee84"

[15] "https://www.engpaper.com/medical/heart-disease-prediction.html"

DATASET INFORMATION (LEGACY)

I use the following representations to collect the dataset:

1. age - age
2. bp - blood pressure
3. sg - specific gravity
4. al - albumin
5. su - sugar
6. rbc - red blood cells
7. pc - pus cell
8. pcc - pus cell clumps
9. ba - bacteria
10. bgr - blood glucose random
11. bu - blood urea
12. sc - serum creatinine
13. sod - sodium
14. pot - potassium
15. hemo - haemoglobin
16. pcv - packed cell volume
17. wc - white blood cell count
18. rc - red blood cell count
19. htn - hypertension
20. dm - diabetes mellitus
21. cad - coronary artery disease
22. appet - appetite
23. pe - pedal edema
24. ane - anaemia
25. class – class

ATTRIBUTE INFORMATION

I use 24 + class = 25 (11 numeric ,14 nominal)

1.Age(numerical) - age in years

2.Blood Pressure(numerical) - bp in mm/Hg

3.Specific Gravity(nominal) - sg - (1.005,1.010,1.015,1.020,1.025)

4.Albumin(nominal) - al - (0,1,2,3,4,5)

5.Sugar(nominal) - su - (0,1,2,3,4,5)

6.Red Blood Cells(nominal) - rbc - (normal, abnormal)

7.Pus Cell (nominal) - pc - (normal, abnormal)

8.Pus Cell clumps(nominal) - pcc - (present, not present)

9.Bacteria(nominal) - ba - (present, not present)

10.Blood Glucose Random(numerical) - bgr in mgs/dl

11.Blood Urea(numerical) - bu in mgs/dl

12.Serum Creatinine(numerical) - sc in mgs/dl

13.Sodium(numerical) - sod in mEq/L

14.Potassium(numerical) - pot in mEq/L

15.Haemoglobin(numerical) - hemo in gms

16.Packed Cell Volume(numerical) – cc in cells/cmm

17.White Blood Cell Count(numerical) - wc in cells/cmm

18.Red Blood Cell Count(numerical) - rc in millions/cmm

19.Hypertension(nominal) - htn - (yes, no)

20.Diabetes Mellitus(nominal) - dm - (yes, no)

21.Coronary Artery Disease(nominal) - cad - (yes, no)

22.Appetite(nominal) - appet - (good, poor)

23.Pedal Edema(nominal) - pe - (yes, no)

24.Anemia(nominal) - ane - (yes, no)

25.Class (nominal) - class - (ckd, not ckd

Proj_Report

by Shashwat Kumar

MULTIPLE ORGAN FAILURE DETECTION USING MACHINE LEARNING

Project report submitted for the partial fulfilment of the requirement for the degree of Bachelor of Technology

In

Electronics and Communication Engineering

By

Shashwat (161038)

Under the supervision of Mr. Pardeep Garg

Department of Electronics and Communication Engineering

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

CANDIDATE'S DECLARATION

I herewith affirm that the work offered in this report entitled **"MULTIPLE ORGAN FAILURE DETECTION USING MACHINE LEARNING"** in partial fulfilment of the necessities for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering submitted to the department of Electronics and Communication Engineering, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a time period of September 2019 to March 2020 under the supervision of Mr. Pardeep Garg (Assistant Professor(Grade-II)), Electronics and Communication Engineering.

The matter encased in this report has not been submitted for the reward of any other degree or diploma.

[illegible]

Shashwat (161038)

This is to certify that the above declaration made by the applicant is true to the best of my knowledge.

[illegible]

Mr. Pardeep Garg

Assistant Professor (Grade-II), Department of Electronics and Communication Engineering

ACKNOWLEDGEMENT

I would like to express my thankfulness to our teacher and mentor Mr. Pardeep Garg who gave me the opportunity to organize this project on the topic **MULTIPLE ORGAN FAILURE DETECTION USING MACHINE LEARNING**, which has helped me in a lot of research and I learnt about so many new perspectives. I am truly thankful to him.

Secondly, I would also like to acknowledge my lab assistants who helped me a lot in completing this project in less than the given time frame.

CONTENTS

S/No.	Title	Page.
1.	Abstract	3
2.	Aim and Objectives	4
3.	Chapter 1 - **INTRODUCTION**	5-8
	1.1 Introduction to The Project	
	1.2 Motivation	
	1.3 Organization of the Report	
4.	Chapter 2 – **LITERATURE REVIEW**	9-22
	2.1 How Machine Learning Works	
	2.2 Types of Machine Learning	
	2.3 Machine Learning Algorithms	
	2.4 Interpretation of Performance Measures	
	2.5 Deep Learning	
	2.6 Perceptrons	
	2.7 Loss Function	
	2.8 Problems in Supervised Learning	
5.	Chapter 3 – **SYSTEM DESIGN**	23-24
	3.1 Why Python and no other languages?	
	3.2 Why Jupyter?	
	3.3 SCIKIT-LEARN	
	3.4 PANDAS	
	3.5 KERAS	
	3.6 TENSORFLOW	
6.	Chapter 4 – **IMPLEMENTATION (Kidney Failure)**	25-38
7.	Chapter 5 – **IMPLEMENTATION (Heart Failure)**	39-47
8.	Chapter 6 – **RESULTS AND ANALYSIS**	48-50
9.	Chapter 7 – **CONCLUSION AND FUTURE SCOPE**	51-52
10.	**REFERENCES**	53
11.	**DATASET INFORMATION**	54
12.	**ATTRIBUTE INFORMATION**	55

8

LIST OF FIGURES

Figure No.	Description	Page No.
1.1	Steps for feeding a ML model	9
2.1	Introduction to ML	10
2.2	ML algorithms	12
2.3	ML Types	13
2.4	Formulae used for calculating nearest distance	14
2.5	Graphical depiction of Linear Regression	15
2.6	Pictorial representation of confusion matrix	15
2.7	Deep Learning	17
2.8	Data of 400 patients with 24 attributes	19
4.1	Loading of modules and helper functions	25
4.2	Loading the .csv file	26
4.3	Cleaning and pre-processing of the data	26
4.4	Generating the colormap graph	27
4.5	Splitting data into training and testing models	28
4.6	Feature Importance selection	28
4.7	Making predictions and obtaining results	29
4.8	Training the CDNN	30
4.9	Training the CDNN	31
4.10	Heatmap of all 400 values	31
4.11	Results of comparison of various NN	32
4.12(a-j)	Results of comparison of various NN	33
4.13	Trained CDNN after data feeding	33
4.14	Blank heatmap (all values processed)	34
4.15	Using MinMaxScaler and LabelEncoder	35
4.16	Result of Epoch	35
4.17	Accuracy Results	35
4.18(a-b)	Graph of Accuracy and Loss	36
4.19	Important Libraries	36

4.20	Building the ANN Model	37
4.21	Training the model	37
4.22	Results of Epoch with accuracy 99.56	37
4.23	Model Accuracy and Loss	38
4.24	Final results of ANN	38
5.1	Dataset of 303 heart patients with 14 selective parameters	39
5.2	Description of the variables with the ‘target’	40
5.3	Data comparison 1	40
5.4	Data comparison 2	41
5.5	Data comparison 3	41
5.6	Data comparison 4	42
5.7	70% training and 30% testing split	42
5.8	Logistic Regression	43
5.9	Naïve Bayes	43
5.10	Support Vector Machines	44
5.11	K-Nearest Neighbours	44
5.12	Decision Tree	45
5.13	Random Forest Classifier	46
5.14	XGBoost	46
5.15	The Neural Networks working	47
5.16	Neural Networks	47
6.1	Final Results for the various algorithms	49

LIST OF TABLES

Table No.	Description	Page No.
2.1	Difference between various ML algorithms	16
2.2	Confusion Matrix	16
6.1	Results of comparative analysis	48
6.2	Final results for the various algorithms	50

ABSTRACT

In this task, I was approached to explore different avenues regarding genuine world datasets, and to perceive how AI calculations can be utilized to discover the examples in information. I was relied upon to pick up experience utilizing normal information mining and AI libraries, and was required to present a report about the dataset and the calculations utilized with their outcomes. Subsequent to playing out the necessary undertakings on the gave datasets, in this lies my report.

AIM

The project's aim is to plan a systematically enhanced machine learning model which will be helping us in the early detection of various organ failure instances and thus in turn, will help in the early detection and prevention of the aforementioned cases, saving lives, time and money.

OBJECTIVES

- Study about various methods of detection of different types of organ failures.
- Download multiple specific organ sample dataset to train the model.
- Use of different AI calculations to the made dataset utilizing Python and looking at the calculations based on parameters like exactness, accuracy, time utilization and so on.
- Obtaining the result with the best parameter judgement and converting it into a real-world application.

Chapter-1

INTRODUCTION

1.1 Introduction

ML is a subset of computer sciences which developed from example acknowledgment in information, and furthermore from the hypothetical learning in user reasoning. It is the top of the line calling to most fascinating vocations related to information investigation today[4]. As information sources multiply alongside the registering capacity to process them, approaching directly to the information is one of the clearest ways to rapidly pick up experiences and make predictions[6].

AI is the investigation of a rundown of sub-issues, like: dynamic, bunching, arrangement, determining, profound learning, inductive rationale programming, SVM, support learning, similitude and metric learning, hereditary calculations, meagre word reference learning, and so forth. Supervised learning, or arrangement is the undertaking of deriving a quantity from a named data[6]. In this, we use sets namely preparation and test. The preparation and test set comprises of a lot of models comprising of info and yield vectors, and the objective of the administered learning calculation is to gather a capacity that maps the information vector to the yield vector with negligible blunder. In an ideal situation, a model prepared on a lot of models will characterize an inconspicuous model in a right manner, requiring summing up from the preparation set in a sensible manner. In layman's terms, managed learning can be named as the procedure of idea realizing, where a mind is presented to a lot of data sources and result vectors and the cerebrum learns the idea that relates said contributions to yields. A wide cluster of regulated AI calculations is accessible to the ML aficionado, for instance Decision Trees, Neural Networks, SVM, Naïve Bayes Classifier, Random Forest, Bayes Net and so on, and they each have their own benefits and negative marks with respect to their distinguished purpose[4].

1.2 Motivation

Chronic Kidney Disease (CKD) progressively advances and typically after months or years the kidney loses its usefulness. By and large, it may not be distinguished before it loses 25% of its usefulness[1]. The beginning of renal failure may not be perceived by the patients since renal failure may not give any manifestations at first. Renal failure treatment focuses to control the causes and slow down the development of the renal failure. In the event that medications are insufficient, patient will be at long last phase of failure and the final option is transplant or dialysis. At present, 4 of each 1000 man in the United Kingdom are experiencing renal failure and in excess of 300,000 American patients at long last phase of kidney malady make due with dialysis.[3]

Also, as indicated by the National Health Service, kidney failure is progressive in South Asia, Africa, then in other different nations. Because the identification of the constant kidney disease isn't plausible until the failure has totally taken over; subsequently, understanding the kidney failure in the principal stage is critical. Through early analysis, the demonstration of every kidney can be taken care of, which prompts diminishing the danger of irreversible outcomes. Thus, standard checkups and early analysis are vital to the patients, for they can forestall fundamental dangers of renal stoppage and related maladies. One of the means to identify failure is blood test. Hence, it very well may be recognized by estimating components, and doctors can choose treatment forms, reducing the pace of progression.

We work on finding dataset patterns through a sample of the data from various kidney disease diagnosis (ECG and other methods), containing attributes such as blood pressure, age, RBC count, haemoglobin etc, and use various algorithms on this data, observing the results. The best results will be the ones which consume the least time with most accurate outputs.

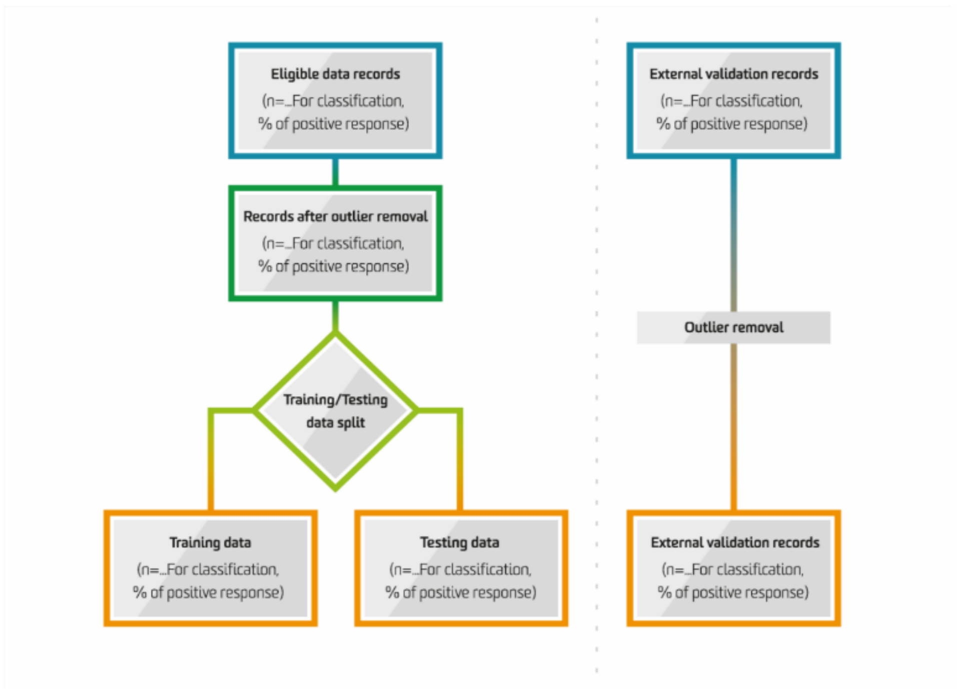


Figure 1.1: Steps for training and testing a Machine Learning Model

1.3 Organization of The Report

Chapter-1: I introduced the scope of our project and why is it of concern in today's date. I have also further shared our motivation behind the establishment of the idea behind this project and how I came across its formulation and working.

Chapter-2: I've talked about the writing survey of the project, which comprises of the terms, significance and the working of different sorts of algorithms utilized. I have depicted different favourable circumstances and weaknesses to distinguish the most appropriate algorithm for our task.

Chapter-3: I've designed and mentioned all those framework necessities to run the calculations and the runtime environment where the calculations are tried and tested.

Chapter-4: I've talked insight regarding the calculations utilized and the arithmetic or the plan behind these calculations for better understanding and tried out these calculations on the dataset to acquire results.

Chapter-5: I've talked about the outcomes and done testing based on different parameters to acquire the most appropriate calculation for our dataset.

Chapter-6: I've finished up the report and learning of the undertaking. Likewise, I have talked about the future and the progressions that can be done to the project.

CHAPTER-2

LITERATURE REVIEW

2.1 Terminologies

Machine Learning (ML) - ML is a sort of calculation that is a part of computerized reasoning and that makes the framework or the product to be sufficiently intelligent to have the choice of being right without being unequivocally customized and can anticipate the results. The primary thought behind these sorts of calculations is that it gets input information as content or pictures and the framework of the model is prepared with the measurable inputs to distinguish or foresee the yield, refreshing the outputs as new information is at avail. It requires the calculation to look through the dataset and search for examples or similitudes and controlling or changing the framework as needs be.

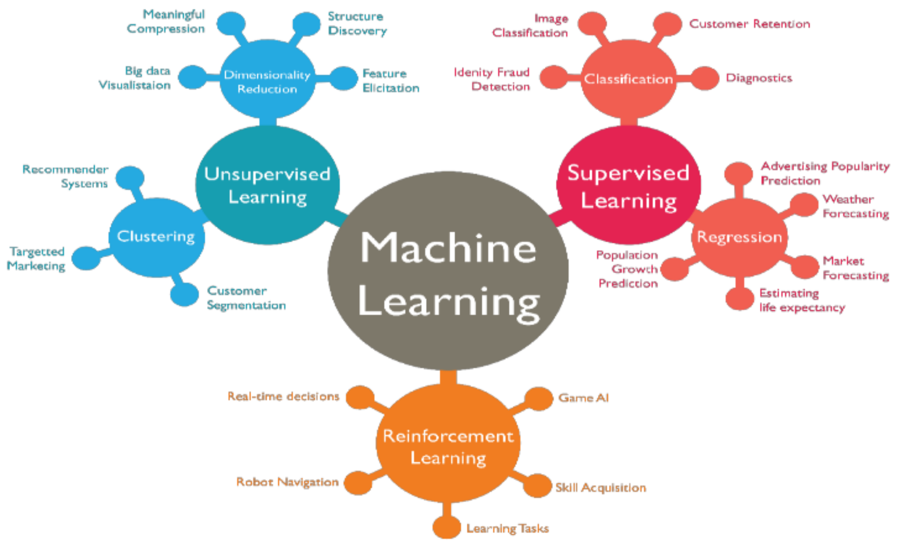


Figure 2.1: Introduction to ML

2.1 How Machine Learning Works

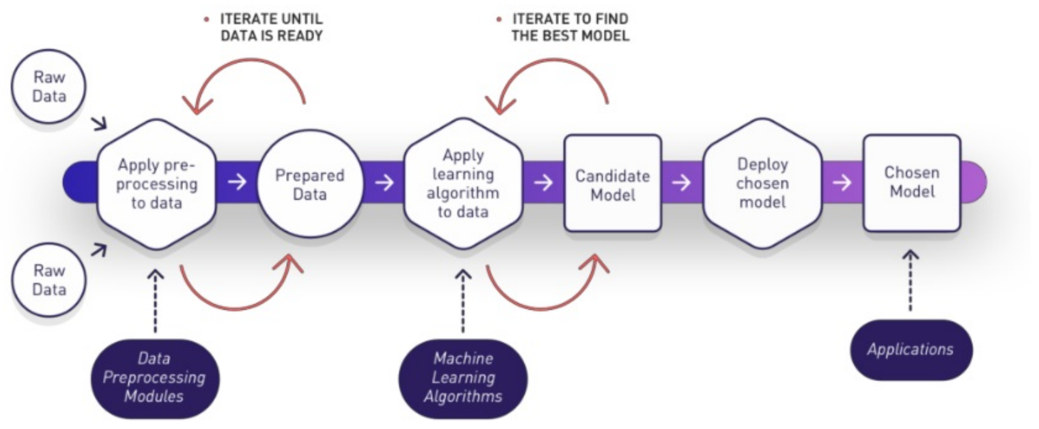

ITERATE UNTIL DATA IS READY
ITERATE TO FIND THE BEST MODEL
Raw Data
Raw Data
Apply pre-processing to data
Prepared Data
Apply learning algorithm to data
Candidate Model
Deploy chosen model
Chosen Model
Data Preprocessing Modules
Machine Learning Algorithms
Applications

Figure 2.2: ML algorithm workflow

The procedure of learning begins with the assortment of information or perceptions as the inputs which can be as pictures, content, tables and so forth. Further, numerous predefined AI calculations are applied to the inputs which either classify the information into collections or recognizes patterns amongst the dataset to calculate and give proper results for the output obtained. AI calculations are approximately arranged into unsupervised and supervised algorithms.

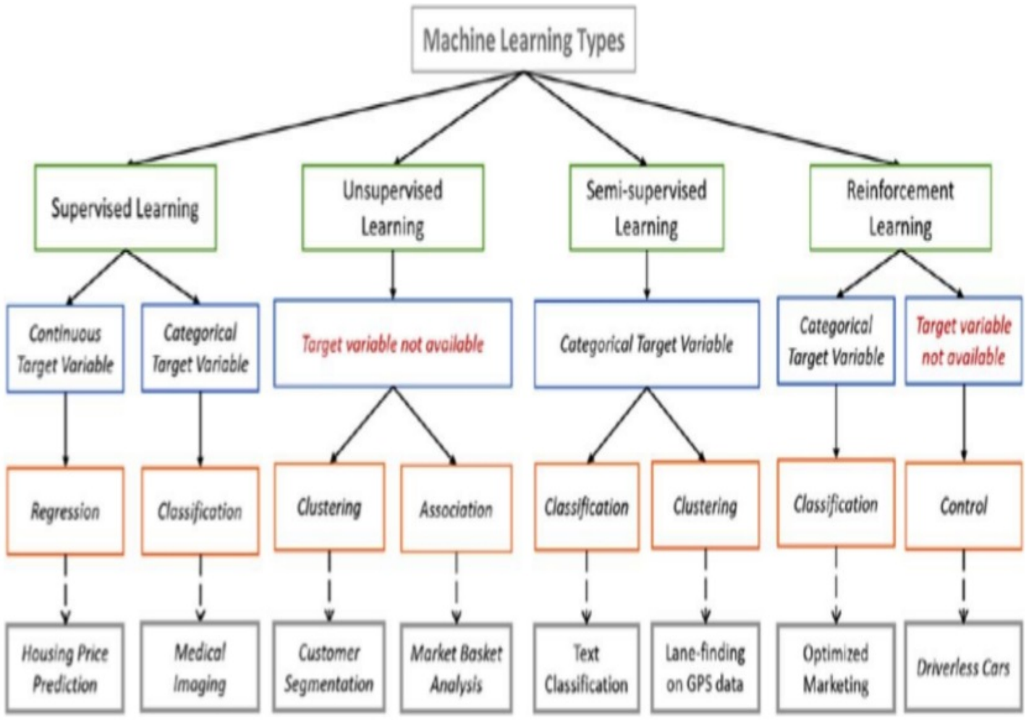
2.2 Types of Machine Learning

Unsupervised Machine Learning - This kind is not quite the same as supervised ML as the calculations are utilized when the model isn't prepared before neither is it trained nor is it marked. These calculations make the framework to tangle a shrouded pattern in the unnamed dataset and foresee potential outcomes with utilization of such examples while expelling the exceptions.

Supervised Machine learning - Supervised learning - This kind is used by data which is now prepared from the past outputs from the results of past utilizing marked information to anticipate the result of the new information. For this case this dataset is investigated, the algorithm at that point delivers a gathered function that can help in expectation of the yield estimations of new information. It can likewise break down the input and the result and contrast with the recently stored information to get errors and as ready to change and train the model as needs be.

Semi-Supervised Machine Learning – This is the calculation which uses benefits of both types of ML for preparing the dataset and subsequently delivers lots of useful and rewarding classifiers. The model uses both named and unnamed data for the preparation and it for the most part requires a limited quantity of named data and a generally huge number of unnamed data which are utilized all the while to prepare the model. This is utilized for improving the precision and the forecast capacities of the model and consequently frequently utilized on account of information requiring both gain and loss from the useful information.

Reinforcement Machine Learning- The model trains on a reward-based activity of finding mistakes and correctness. The attributes to this type of learning are the rewards and mistakes. For this situation learning from past missteps or mistakes for the model is made to work with the learning ability to determine the result and the perfect conduct for maximum performance.



5

Figure 2.3: Types of machine learning

2.3 Algorithms

This segment comprises of several useful algorithms which are applied in this project and commented on:

Supervised learning

K-Nearest Neighbours- Nearest neighbors is utilized both for classification and regression yet is generally utilized for classification issues. This calculation is simple and consumes lesser time. The K represents the no. of neighbors assigned by user. The calculation utilizes the Euclidean separation to gauge KNN to the information point and anticipate the output as indicated by its neighbours[8].

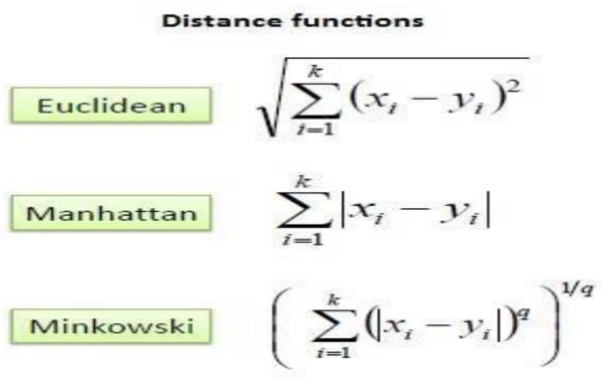

Distance functions
Euclidean
$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$
Manhattan
$\sum_{i=1}^{k}|x_i - y_i|$
Minkowski
$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

Figure 2.4: Formula for calculating minimum separation.

Naïve Bayes- Bayes hypothesis is a kind of classifier algorithms utilized for different types of classification not regression problems. This hypothesis is known as so in light of the fact that it has foundations in the Bayes theorem. N.Bayes frequently is spoken to by probabilities. The information is put as probabilities in this type on model.

Formula - P(x,y) = (P(y|x)*P(x))/P(y)

Decision Trees- A supervised learning algorithm where an information structure is utilized to take care of an issue. For this situation a node or centre is mentioned to as a class name and the attributes are mentioned in the insider nodes. They can take care of the issues of both regression and classification. At first, we take the entire dataset as a root and an unmitigated element value are used and the continuous inputs are first made discrete qualities before utilizing them to assemble the model needed. At that point factual strategies are utilized for requesting the traits as inward node or root.

Formula-

$$Gain(S, A) = Entropy(S) - \sum_{v \epsilon Values(A)} \frac{|S_v|}{|S|} . Entropy(S_v)$$

Where:

- S - represents the training set
- A - represents the attribute that we are comparing
- $|S|$ - represents the number of elements in the training set
- $|S_v|$ - represents the number of elements where the attribute has a part

$$Entropy(S) = \sum_i -p_i log_2 p_i$$

Regression- This algorithm utilizes the factual ideas and establishes a connect between the inputs and number outputs. The model is characterized by a linear equation which consolidates the input estimations of the set chosen and can predict the output based on past learning experiences of the model.

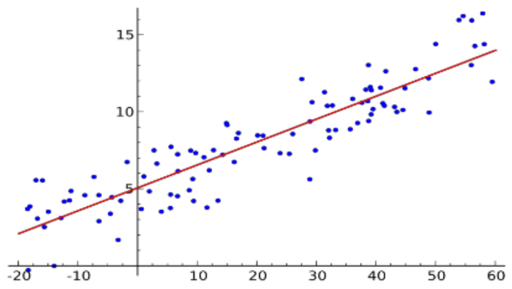


Figure 2.5: Graphical depiction of regression.

SVM- It is a type of classification algorithm used for both the former and regression problems. This is generally utilized where every item of data is plotted in a n-dimensional space and n characterizes the highlights in there and the estimation of each and every element is the estimation of each organize. Furthermore, different hyperplanes are made to separate those 2 modules.

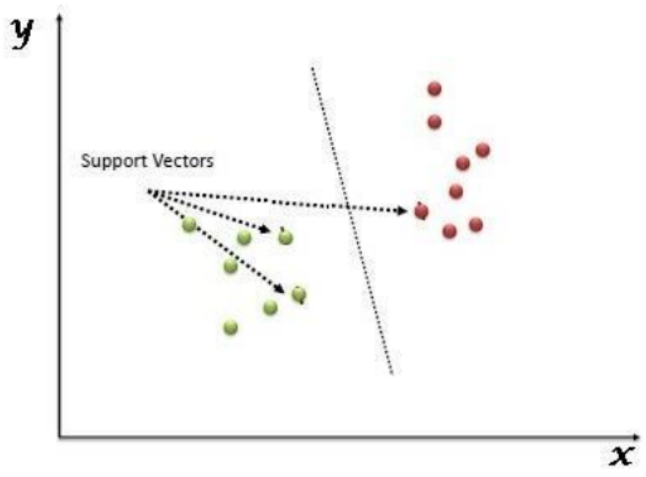
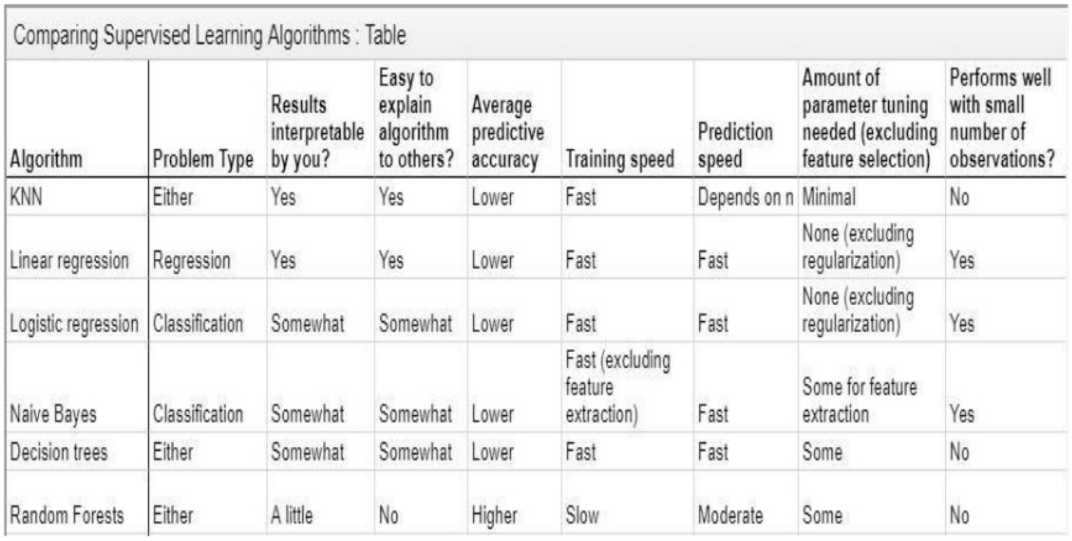


Figure 2.6: Graphical depiction of SVM.



Comparing Supervised Learning Algorithms : Table								
Algorithm	Problem Type	Results interpretable by you?	Easy to explain algorithm to others?	Average predictive accuracy	Training speed	Prediction speed	Amount of parameter tuning needed (excluding feature selection)	Performs well with small number of observations?
KNN	Either	Yes	Yes	Lower	Fast	Depends on n	Minimal	No
Linear regression	Regression	Yes	Yes	Lower	Fast	Fast	None (excluding regularization)	Yes
Logistic regression	Classification	Somewhat	Somewhat	Lower	Fast	Fast	None (excluding regularization)	Yes
Naive Bayes	Classification	Somewhat	Somewhat	Lower	Fast (excluding feature extraction)	Fast	Some for feature extraction	Yes
Decision trees	Either	Somewhat	Somewhat	Lower	Fast	Fast	Some	No
Random Forests	Either	A little	No	Higher	Slow	Moderate	Some	No

Table 2.1: Different Machine Learning algorithms

2.4 Understanding of Performance Factors

There are different strategies to assess the use of various algorithms. One of the ways is to decide the curve are or the curve ROC and other different parameters otherwise called as Confusion Values. To assess the present proportion of the grouping model that gives the genuine qualities are known, the matrix table is used[9].

		Predicted	
Actual		**Yes**	**No**
	Yes	**True Positive**	**False Negative**
	No	**False Positive**	**True Negative**

Table 2.2: The Matrix (Confusion)

The table presented comprises of confusion values and has 4 segments. The two area in the green are the True values and these are the perceptions which are correctly anticipated. The two areas in red in light of the fact that these qualities are wrongly anticipated and, in this way, should be limited. These segments are False and happen when we experience an inconsistency between genuine class with the anticipated class[10]].

Genuine Positives (TP) – These qualities are effectively anticipated and are sure quantities which can be portrayed as the correct estimation of real class and correct estimation of anticipated class. It is denoted by TP.

Genuine Negatives (TN) - These quantities are accurately anticipated yet untrue qualities which alludes the invalidation of real class and disproof of anticipated class. This is signified by TN.

Incorrect Positives (FP) – These quantities that are incorrect yet are valid in genuine.

Incorrect Negative (FN) – These qualities are incorrectly anticipated and are negative in real class.

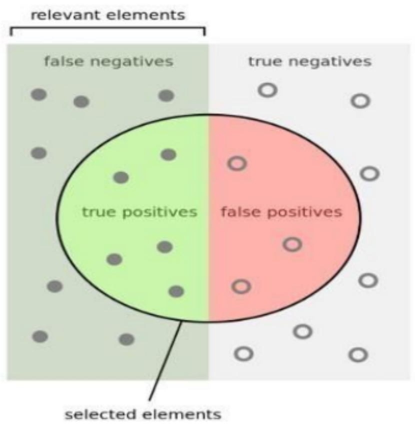


Figure 2.7: Graphic depiction of misperception matrix

Further, I studied upon more limitations of performance.

Accuracy – This is a normal measure and is basically an extent of adequately foreseen perception by the total observations. Envisioning that, in case of having high precision, the model turns out to be perfect. Really, accuracy is considered an unparalleled measure yet exactly when having symmetric datasets where estimations of non-true positive and non-true negatives are generally same. As such, we have to look at changed parameters for evaluation of the execution of the model.

Accuracy = (TN+TP)/(FN+TP+TN+FP)

Precision – This is the extent of precisely foreseen correct output observations to the total foreseen positive perceptions. Large precision values can relate to the low false positive rate, which is a good thing.

Precision **=** TP/FP+TP

Recall – This is described as the extent of viably foreseen positive judgements to the all observations in a real defined class.

Recall = TP/FN+TP

F1 Score – This is the average of Precision and Recall, weighted one. Thus, F1 thinks about both false positives and negatives. It isn't accuracy, yet this score is typically considered more important than precision, especially on possibility that we might have a distributed class movement. Exactness gives best results if false negatives positives have practically identical weight. In case the cost of false negatives and positives are through and various, it's a better practice to look at both Recall and Precision.

F1 = 2*(Precision*Recall) / (Precision + Recall)

2.5 Deep Learning (DL)

Versatile Learning is an increasingly intricate and clever sub-classification of AI having all the calculations roused by the working and biological composition of the brain known as Artificial neural network. Furthermore, it likewise alludes to the strategies that are utilized for learning networks with various layers. Artificial Neural Network is the kind of model which is made out of motivation and takes a shot at the essential thought of sensory systems and the handling of data in brain to gain from information. Here, the learning components can be either administered, semi-managed or unaided. Profound learning can be demonstrated as fruitful in different fields and brought about increasingly practical machine learning stages. It can be used in the field of medication structure till the traffic forecast and furthermore for the article acknowledgment. A deep neural network is unique in relation to a neural network due to the quantity of layers. The usage of profound learning whilst I did for this, I experienced 2 primary issues, for example, 1) the computational force required for the way toward preparing the model was not lesser than the framework accessible and, in a way, requires more opportunity for calculation. 2) A more difficult problem experienced during the usage was the inclination disappearing issue, that was, in a neural system that has enactment capacities, for example, the hyperbolic digression and the slope is (-1,1) or (0,1), the chain rule is generally used to figure out the back-propagation. It comprises of duplicating 'k' to some little numbers from output layer through a k-layer networks, which implies that the angle diminished exponentially with k. Results are that the above layers of the model trains gradually than the rest of the layers present in the formulation of the deep learning model skeleton.

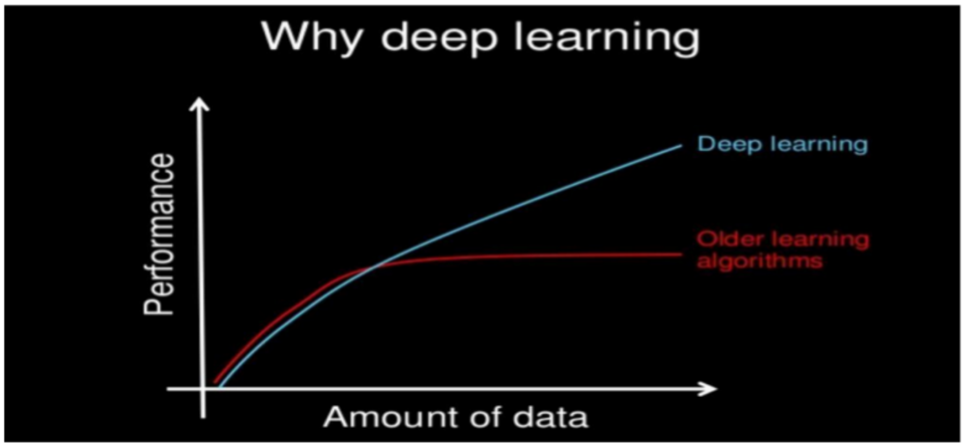


Figure 2.8: Graphical depiction of Deep Learning vs additional learning procedures

2.6 Perceptrons

One of the most useful learning calculations is the perceptron one and it is a minuscule block of the Artificial Neural Network. The working of the perceptron is by taking different information sources (a1, a2… … aj) and creation of a solitary output (y). Furthermore, weighted sources of ideas were additionally helping decide the significance of separate contributions to the output. The subsequent output is either 1 or 0 and it is possibly controlled by cross-checking the weighted entirety being more prominent than 0 or under 0.

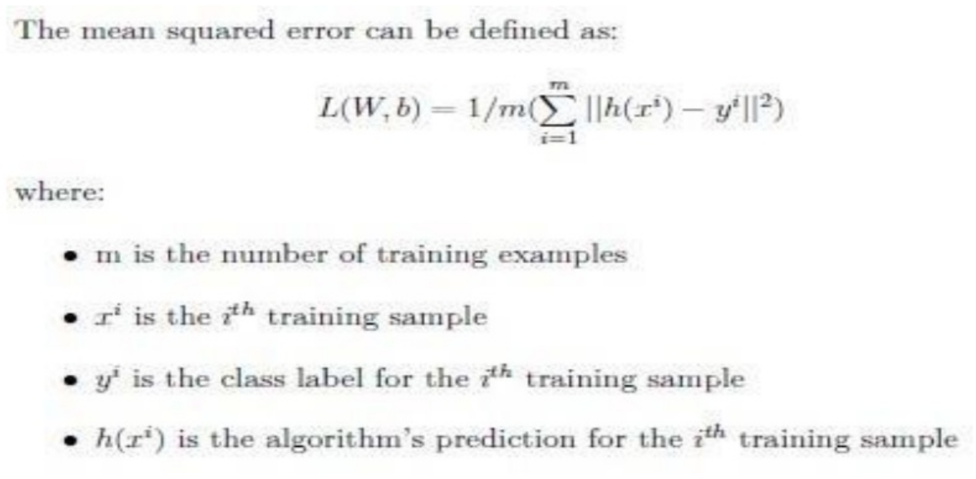
Weighted Summation- $\sum_i w_i * a_i + b$

$$\text{Output} = \begin{cases} 1: & \text{if } w * a + b > 0 \\ 0: & \text{if } w * a + b <= 0 \end{cases}$$

Be that as it may, as perceptron algorithm just gives the yield as 1 or 0, making it troublesome or about difficult to expand the functionalities of the model to have the option to take a shot at characterization errands having various classes. Moreover, this issue might be settled by considering numerous perceptrons in a layer ensuring that every layer having its respective perceptron acquires a similar info and the perceptrons are answerable for the output work. The ANN is just perceptrons with multiple layers, though the perceptron simply an ANN with only one layer, which is regularly the yield layer having just a single neuron.

2.7 Loss/Cost Function

The usefulness of a NN can be estimated by a capacity also known as the work function / loss work function which helps in estimating the error in the expectation by the calculations and the right label if the forecast or the aggregate arrangement of forecast can be mentioned alongside the mark or a lot of names. Among these different cost work accessible the least complex and the generally utilized in NN systems is the MSE[10].


The mean squared error can be defined as:

$$L(W, b) = 1/m(\sum_{i=1}^{m} ||h(x^i) - y^i||^2)$$

where:

- m is the number of training examples
- x^i is the i^{th} training sample
- y^i is the class label for the i^{th} training sample
- $h(x^i)$ is the algorithm's prediction for the i^{th} training sample

The final output coming from training these networks is to reduce the work / cost function and determine the individual parameters that help in doing so. For this technique, I implemented the gradient descend algorithm.

2.8 Problems in Learning

We should be informed on as to choose a suitable training algorithm for a specified dataset. To diligently use up an algorithm for a supervised learning job, we must take into account these factors:

1. Non-Homogeneity of Data:

Most neural systems and SVM have the vectors used to be non-heterogeneous and standardized. The calculations that utilize separation measurements are exceptionally touchy to this, and thus if the information is non-homogeneous, these strategies ought to be the idea in retrospect. The Decision Trees way of working can deal with heterogeneous information quite easily[10].

2. Redundancy of Data:

On the off chance that the information contains repetitive data, for example contain profoundly related qualities, at that point it's futile to utilize separation-based techniques as a result of numerical unsteadiness. For this situation, a regularization can be utilized to the information to forestall this situation[10].

Features: If there is some reliance between the component vectors, at that point calculations that screen complex collaborations like Neural Networks and Decision Trees toll better than other algorithms[10].

3. Bias-Variance Trade-off:

Calculation is one-sided for a specific information 'x' prepared on every one of these informational collections, it is deliberately mistaken while anticipating the right output for x, though a learning calculation difference for a specific information x on the off prediction chance of diverse values when prepared on a variety of sets is high. The expectation blunder of a trained classifier can be identified with the aggregate of bias and change of the calculation, and neither be higher as the forecast mistake would be higher. An important element of AI calculations is that they can tune the harmony among inclination and change naturally, or by manual tuning utilizing predisposition values, and utilizing these performed calculations will settle this condition[10].

4. Dimension Problems:

In the event that the issue has an input that has an enormous number of measurements, and the issue just relies upon the information space with little measurements, the AI algorithm will confound by the tremendous number of measurements and consequently the difference of the calculation can be great. Practically speaking, when the information researcher can physically expel unimportant highlights from the information, it probably improves the precision of the trained capacity. Moreover, there are a variety of calculations for inclusion of choice that try to distinguish the significant highlights and discard the useless ones, for example Principle Component Analysis. This lessens the dimensionality of the dataset[10].

5. Overfitting:

Sometimes the output we have from all the processing work can be comprised of noise or disturbance in the furnished data due to user errors. For this situation, these calculations must not endeavour to derive the capacity that precisely coordinates all the information. Being excessively cautious in accommodating of all the information causes overfitting, after which the model would answer consummately for all preparation models. However, it will have a high error rate for inconspicuous examples. A pragmatic method of forestalling this is halting the learning procedure rashly, just as applying channels to the information in the pre-learning stage to evacuate commotions. Simply in the wake of considering every one of these variables would we be able to use a directed calculation with the dataset we are chipping away at. For instance, on the off chance that we were using on this dataset comprising of non-homogeneous information, at that point choice trees will be a lot better than others for the case being[10].

CHAPTER-3

SYSTEM DESIGN

3.1 System Requirements

We used following system configuration for this project as prerequisites:

Windows Operating System version 10

Jupyter Notebook

Python 3.7

8 GB memory

An i7-8700HQ CPU

NVIDIA® GeForce™ GTX 1050Ti

3.2 Why Python and no other languages?

Python is a highly popular language and can be very much intelligible. Moreover, it provides the user with and assortment of libraries which help in making scariest calculations or activities less complex. Python contains blocks for pretty much every usable document for example: graphical usage, content creation or with sound documents. In any event, when shifting to another system, python is truly flexible. Python has a great connection with others so that it simple to look up help in time of need.

3.3 Why Jupyter?

Jupyter notebook is used widely as it avails the pre-installed libraries making the programmer free from the installation of all these libraries. This contains about 200 packages used on a daily basis by various programmers.

3.4 Python -SCIKITLEARN

Scikitlearn is widely used in python generally for ML and is able to feature different types of algorithms.

3.5 Python -PANDAS

A python library that can provide with well managed performance metrics. This is laidback due to its ease of usage and analysis tools. It is being excessively used in all commerce and manufacturing fields.

3.6 Python -KERAS

Its general use is in ANN. It is designed for swift trialling of many intricate algorithms.It focuses on providing easy of usage, access to many fields and being modular.

3.7 TENSORFLOW

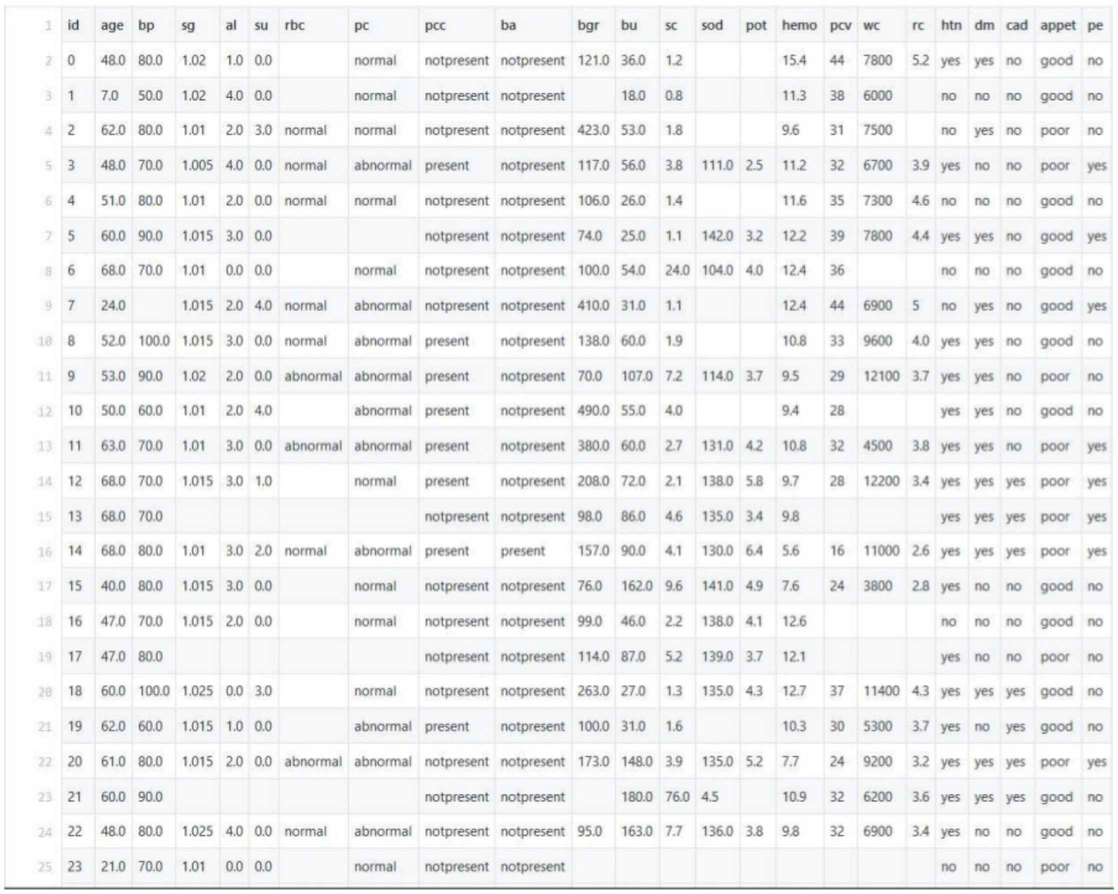
Used to perform a variety of tasks like programming and dataflow. TensorFlow can be well-defined as a representative math which used in neural networks like fields of machine learning universe.

CHAPTER-4

IMPLEMENTATION (Kidney Failure)

Here I would be discussing about the testing and research phases of the project.

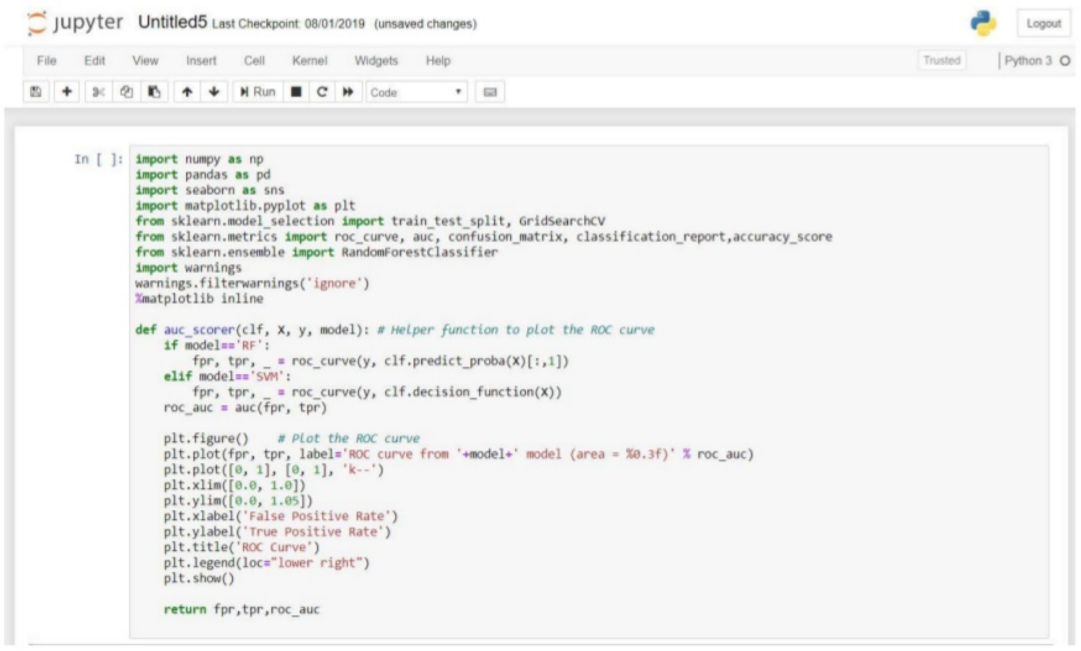
I studied about the various stages and factors that might be affecting the patient's kidneys, declassifying it from normal. Given 24 health related attributes taken in a set time period for four hundred patients, by means of the data of one hundred and fifty-eight patients through comprehensive archives to get the result(i.e. whether someone has CKD) of the lasting two hundred and forty two patients(with unidentified parameters present in the data).



id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	htn	dm	cad	appet	pe
0	48.0	80.0	1.02	1.0	0.0		normal	notpresent	notpresent	121.0	36.0	1.2			15.4	44	7800	5.2	yes	yes	no	good	no
1	7.0	50.0	1.02	4.0	0.0		normal	notpresent	notpresent		18.0	0.8			11.3	38	6000		no	no	no	good	no
2	62.0	80.0	1.01	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8			9.6	31	7500		no	yes	no	poor	no
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	111.0	2.5	11.2	32	6700	3.9	yes	no	no	poor	yes
4	51.0	80.0	1.01	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4			11.6	35	7300	4.6	no	no	no	good	no
5	60.0	90.0	1.015	3.0	0.0			notpresent	notpresent	74.0	25.0	1.1	142.0	3.2	12.2	39	7800	4.4	yes	yes	no	good	yes
6	68.0	70.0	1.01	0.0	0.0		normal	notpresent	notpresent	100.0	54.0	24.0	104.0	4.0	12.4	36			no	no	no	good	no
7	24.0		1.015	2.0	4.0	normal	abnormal	notpresent	notpresent	410.0	31.0	1.1			12.4	44	6900	5	no	yes	no	good	yes
8	52.0	100.0	1.015	3.0	0.0	normal	abnormal	present	notpresent	138.0	60.0	1.9			10.8	33	9600	4.0	yes	yes	no	good	no
9	53.0	90.0	1.02	2.0	0.0	abnormal	abnormal	present	notpresent	70.0	107.0	7.2	114.0	3.7	9.5	29	12100	3.7	yes	yes	no	poor	no
10	50.0	60.0	1.01	2.0	4.0		abnormal	present	notpresent	490.0	55.0	4.0			9.4	28			yes	yes	no	good	no
11	63.0	70.0	1.01	3.0	0.0	abnormal	abnormal	present	notpresent	380.0	60.0	2.7	131.0	4.2	10.8	32	4500	3.8	yes	yes	no	poor	yes
12	68.0	70.0	1.015	3.0	1.0		normal	present	notpresent	208.0	72.0	2.1	138.0	5.8	9.7	28	12200	3.4	yes	yes	yes	poor	yes
13	68.0	70.0						notpresent	notpresent	98.0	86.0	4.6	135.0	3.4	9.8				yes	yes	yes	poor	yes
14	68.0	80.0	1.01	3.0	2.0	normal	abnormal	present	present	157.0	90.0	4.1	130.0	6.4	5.6	16	11000	2.6	yes	yes	yes	poor	yes
15	40.0	80.0	1.015	3.0	0.0		normal	notpresent	notpresent	76.0	162.0	9.6	141.0	4.9	7.6	24	3800	2.8	yes	no	no	good	no
16	47.0	70.0	1.015	2.0	0.0		normal	notpresent	notpresent	99.0	46.0	2.2	138.0	4.1	12.6				no	no	no	good	no
17	47.0	80.0						notpresent	notpresent	114.0	87.0	5.2	139.0	3.7	12.1				yes	no	no	poor	no
18	60.0	100.0	1.025	0.0	3.0		normal	notpresent	notpresent	263.0	27.0	1.3	135.0	4.3	12.7	37	11400	4.3	yes	yes	yes	good	no
19	62.0	60.0	1.015	1.0	0.0		abnormal	present	notpresent	100.0	31.0	1.6			10.3	30	5300	3.7	yes	no	yes	good	no
20	61.0	80.0	1.015	2.0	0.0	abnormal	abnormal	notpresent	notpresent	173.0	148.0	3.9	135.0	5.2	7.7	24	9200	3.2	yes	yes	yes	poor	yes
21	60.0	90.0						notpresent	notpresent		180.0	76.0	4.5		10.9	32	6200	3.6	yes	yes	yes	good	no
22	48.0	80.0	1.025	4.0	0.0	normal	abnormal	notpresent	notpresent	95.0	163.0	7.7	136.0	3.8	9.8	32	6900	3.4	yes	no	no	good	no
23	21.0	70.0	1.01	0.0	0.0		normal	notpresent	notpresent										no	no	no	poor	no

Figure 4.1: Data of 400 patients (23 shown) with 24 health related attributes

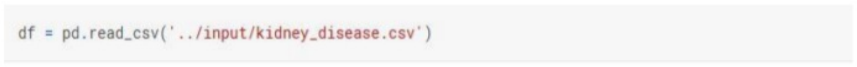
I picked up an existing machine learning algorithm related to the **Random Forest Classifier**, to setup a training model and feed the previous data of 400 patients into it. I did it by the help of Python (3.7) programming language running on Anaconda3 GUI through Jupyter Notebook. I implemented several python3 libraries such as Matplotlib, Pandas, NumPy, SkLearn etc. These are predefined functions and figures that help the computer compile, load and train our dataset according to set parameters of our needs. Our first model was getting ready for a test run for measuring the initial accuracy and obtaining a graph regarding the same.


jupyter
Untitled5 Last Checkpoint: 08/01/2019 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [ ]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split, GridSearchCV
        from sklearn.metrics import roc_curve, auc, confusion_matrix, classification_report,accuracy_score
        from sklearn.ensemble import RandomForestClassifier
        import warnings
        warnings.filterwarnings('ignore')
        %matplotlib inline

        def auc_scorer(clf, X, y, model): # Helper function to plot the ROC curve
            if model=='RF':
                fpr, tpr, _ = roc_curve(y, clf.predict_proba(X)[:,1])
            elif model=='SVM':
                fpr, tpr, _ = roc_curve(y, clf.decision_function(X))
            roc_auc = auc(fpr, tpr)

            plt.figure()    # Plot the ROC curve
            plt.plot(fpr, tpr, label='ROC curve from '+model+' model (area = %0.3f)' % roc_auc)
            plt.plot([0, 1], [0, 1], 'k--')
            plt.xlim([0.0, 1.0])
            plt.ylim([0.0, 1.05])
            plt.xlabel('False Positive Rate')
            plt.ylabel('True Positive Rate')
            plt.title('ROC Curve')
            plt.legend(loc="lower right")
            plt.show()

            return fpr,tpr,roc_auc
```

Figure 4.2: Loading of modules and helper functions

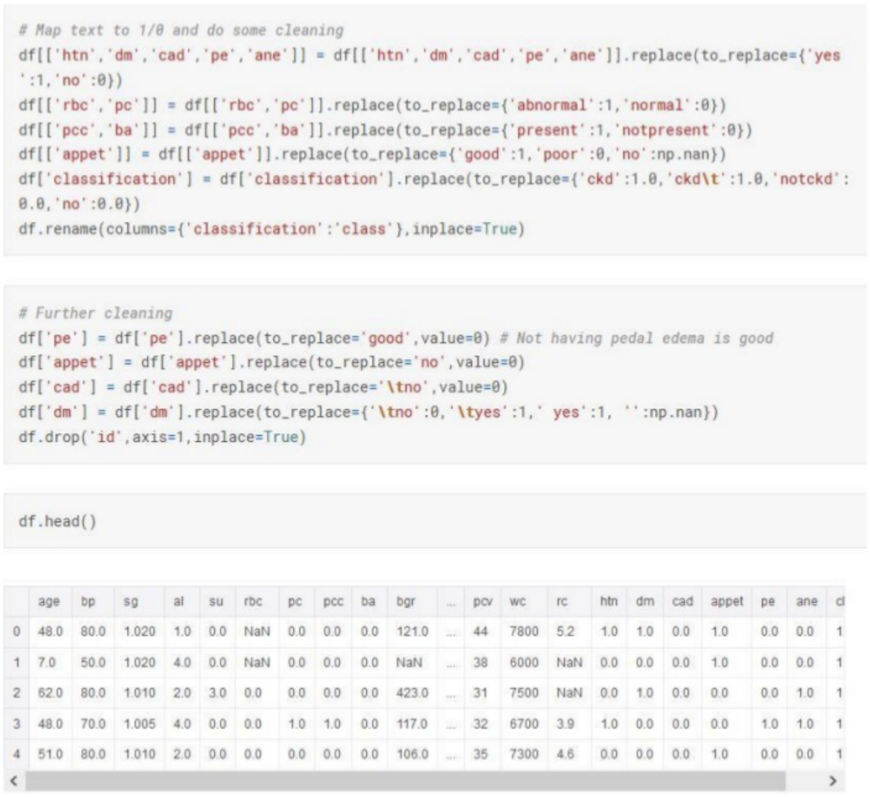
To load the dataset (saved as a .csv file) from our repository, I used the **pd.read_csv** command from the Pandas library.



```
df = pd.read_csv('../input/kidney_disease.csv')
```

Figure 4.3: Loading the csv file into the training model for the classifier

The data given is cleaned with NaN values removed and remaining data sorted out from the unwanted one. This will be providing us with more accuracy in the coming stages of the model training.



```
# Map text to 1/0 and do some cleaning
df[['htn','dm','cad','pe','ane']] = df[['htn','dm','cad','pe','ane']].replace(to_replace={'yes
':1,'no':0})
df[['rbc','pc']] = df[['rbc','pc']].replace(to_replace={'abnormal':1,'normal':0})
df[['pcc','ba']] = df[['pcc','ba']].replace(to_replace={'present':1,'notpresent':0})
df[['appet']] = df[['appet']].replace(to_replace={'good':1,'poor':0,'no':np.nan})
df['classification'] = df['classification'].replace(to_replace={'ckd':1.0,'ckd\t':1.0,'notckd':
0.0,'no':0.0})
df.rename(columns={'classification':'class'},inplace=True)
```

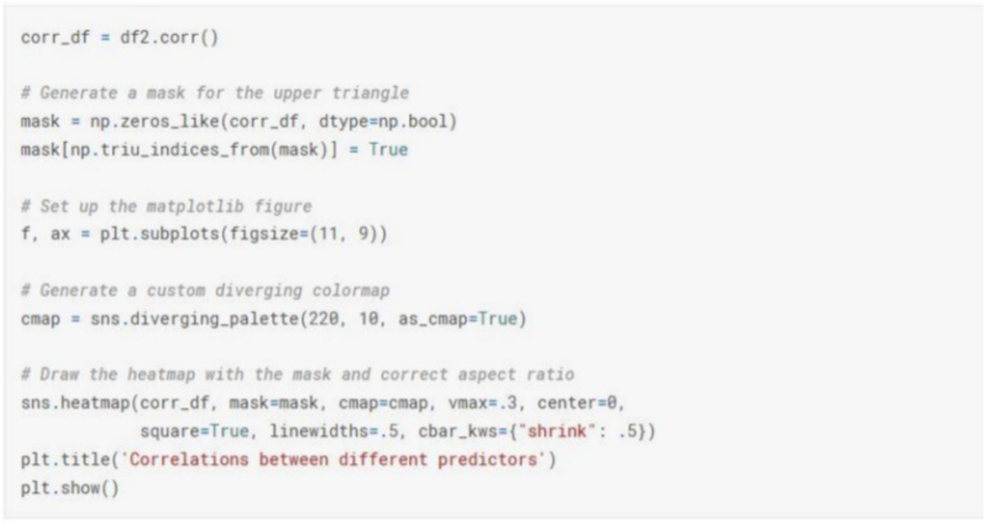
```
# Further cleaning
df['pe'] = df['pe'].replace(to_replace='good',value=0) # Not having pedal edema is good
df['appet'] = df['appet'].replace(to_replace='no',value=0)
df['cad'] = df['cad'].replace(to_replace='\tno',value=0)
df['dm'] = df['dm'].replace(to_replace={'\tno':0,'\tyes':1,' yes':1, '':np.nan})
df.drop('id',axis=1,inplace=True)
```

```
df.head()
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	cl
0	48.0	80.0	1.020	1.0	0.0	NaN	0.0	0.0	0.0	121.0	...	44	7800	5.2	1.0	1.0	0.0	1.0	0.0	0.0	1
1	7.0	50.0	1.020	4.0	0.0	NaN	0.0	0.0	0.0	NaN	...	38	6000	NaN	0.0	0.0	0.0	1.0	0.0	0.0	1
2	62.0	80.0	1.010	2.0	3.0	0.0	0.0	0.0	0.0	423.0	...	31	7500	NaN	0.0	1.0	0.0	0.0	0.0	1.0	1
3	48.0	70.0	1.005	4.0	0.0	0.0	1.0	1.0	0.0	117.0	...	32	6700	3.9	1.0	0.0	0.0	0.0	1.0	1.0	1
4	51.0	80.0	1.010	2.0	0.0	0.0	0.0	0.0	0.0	106.0	...	35	7300	4.6	0.0	0.0	0.0	1.0	0.0	0.0	1

Figure 4.4: Cleaning and pre-processing of data for the classifiers

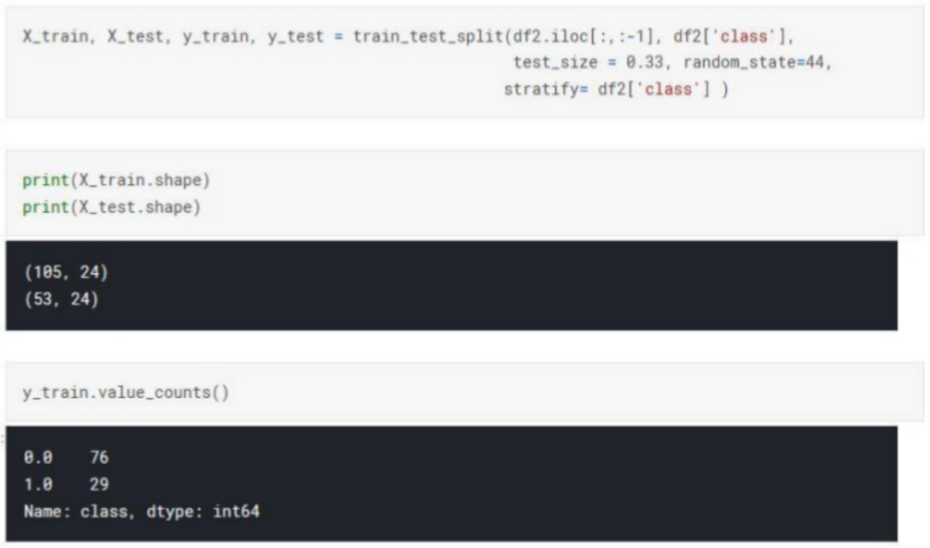
The cleansing of data is done for now and I will now be loading it up in the classifier and will be examining the correlations between different features to plot up a heatmap-based colormap graph.



```
corr_df = df2.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr_df, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_df, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.title('Correlations between different predictors')
plt.show()
```

Figure 4.5: Generating the Colormap Graph

After obtaining the results, I now split the set for training models into sub training and testing sets. I will then be choosing parameters with GridSearchCV with 10-fold cross validations.



```
X_train, X_test, y_train, y_test = train_test_split(df2.iloc[:,:-1], df2['class'],
                                                    test_size = 0.33, random_state=44,
                                                   stratify= df2['class'] )
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(105, 24)
(53, 24)
```

```
y_train.value_counts()
```

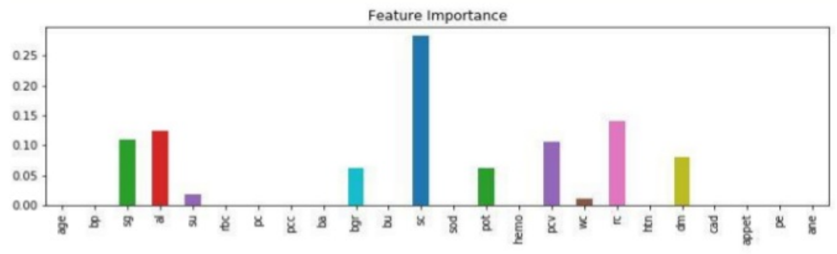
```
0.0    76
1.0    29
Name: class, dtype: int64
```

Figure 4.6: Splitting Data into Training and Testing Models

Next, I would be examining the feature importance to decrease the number of trees in the forest and also pruning it would be beneficial to narrow down the results for better accuracy, as all features are not used.

```
plt.figure(figsize=(12,3))
features = X_test.columns.values.tolist()
importance = clf_best.feature_importances_.tolist()
feature_series = pd.Series(data=importance,index=features)
feature_series.plot.bar()
plt.title('Feature Importance')
```

```
Text(0.5,1,'Feature Importance')
```



```
list_to_fill = X_test.columns[feature_series>0]
print(list_to_fill)
```

```
Index(['sg', 'al', 'su', 'bgr', 'sc', 'pot', 'pcv', 'wc', 'rc', 'dm'], dtype='object')
```

Figure 4.7: Feature Importance Selection

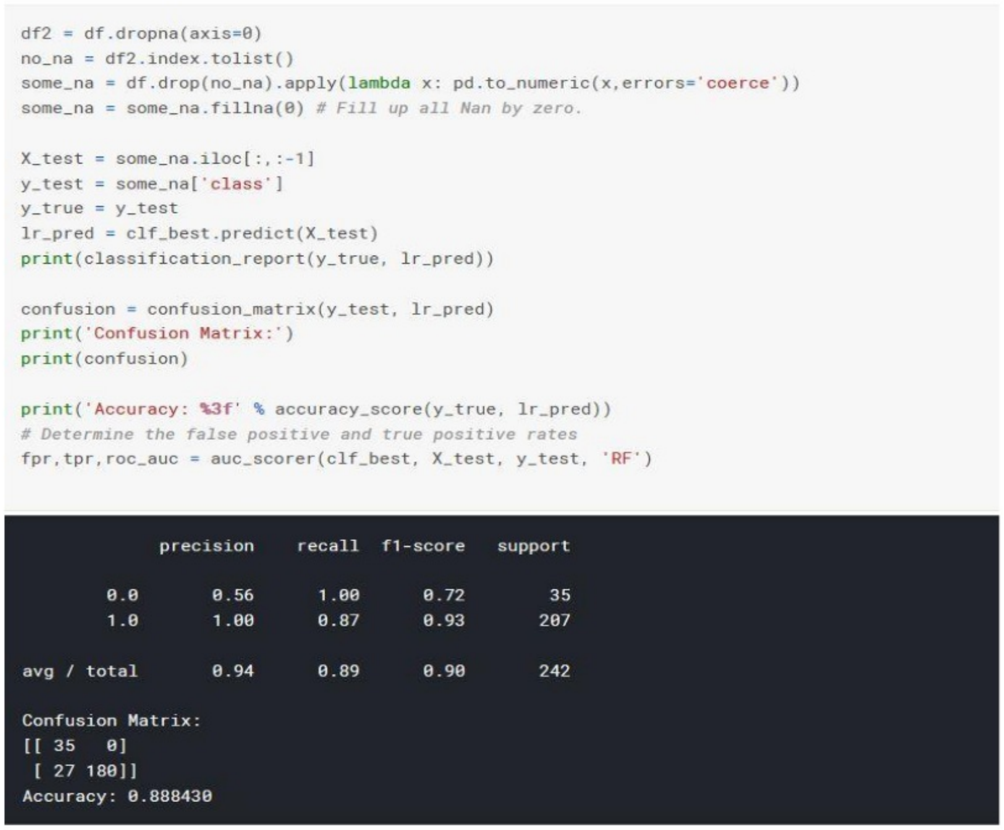
Lastly, I made predictions by obtaining the results from the selected model and formulating the confusion matrix. I filled in all NaN points with 0 and pass it to the trained classifier. The results were as follows:

Confusion Matrix:

[35 0]

[27 180]

Accuracy: 0.888430



```
df2 = df.dropna(axis=0)
no_na = df2.index.tolist()
some_na = df.drop(no_na).apply(lambda x: pd.to_numeric(x,errors='coerce'))
some_na = some_na.fillna(0) # Fill up all Nan by zero.

X_test = some_na.iloc[:,:-1]
y_test = some_na['class']
y_true = y_test
lr_pred = clf_best.predict(X_test)
print(classification_report(y_true, lr_pred))

confusion = confusion_matrix(y_test, lr_pred)
print('Confusion Matrix:')
print(confusion)

print('Accuracy: %3f' % accuracy_score(y_true, lr_pred))
# Determine the false positive and true positive rates
fpr,tpr,roc_auc = auc_scorer(clf_best, X_test, y_test, 'RF')
```

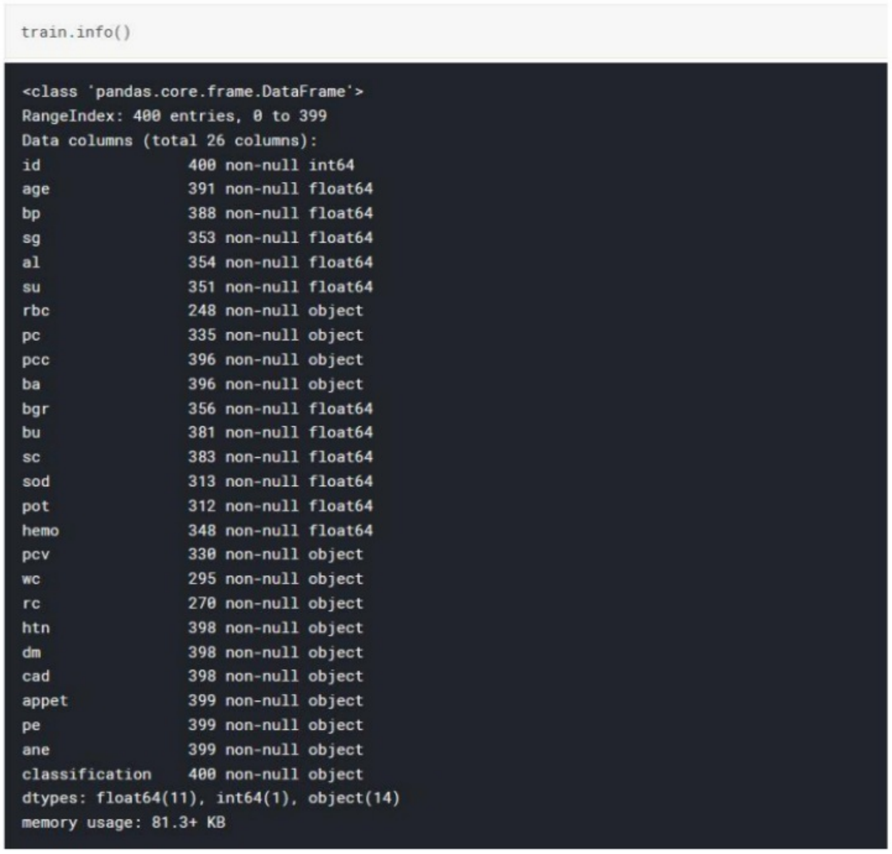
```
             precision    recall  f1-score   support

        0.0       0.56      1.00      0.72        35
        1.0       1.00      0.87      0.93       207

avg / total       0.94      0.89      0.90       242

Confusion Matrix:
[[ 35   0]
 [ 27 180]]
Accuracy: 0.888430
```

Figure 4.8: Making Predictions and Obtaining the Results

I repeated the same steps above for

Convolutional Deep Learning Neural Network (CDNN)

I trained the CDNN model with 80% training and 20% testing data.

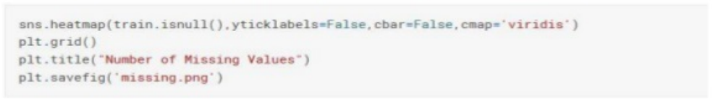


```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
id                400 non-null int64
age               391 non-null float64
bp                388 non-null float64
sg                353 non-null float64
al                354 non-null float64
su                351 non-null float64
rbc               248 non-null object
pc                335 non-null object
pcc               396 non-null object
ba                396 non-null object
bgr               356 non-null float64
bu                381 non-null float64
sc                383 non-null float64
sod               313 non-null float64
pot               312 non-null float64
hemo              348 non-null float64
pcv               330 non-null object
wc                295 non-null object
rc                270 non-null object
htn               398 non-null object
dm                398 non-null object
cad               398 non-null object
appet             399 non-null object
pe                399 non-null object
ane               399 non-null object
classification    400 non-null object
dtypes: float64(11), int64(1), object(14)
memory usage: 81.3+ KB
```

Figure 4.10: Training the Convolutional Deep Learning Neural Network (CDNN)

Before the initial learning, I took out the heatmap of the fed training data to compare it later on when the training of the model is complete.



```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
plt.grid()
plt.title("Number of Missing Values")
plt.savefig('missing.png')
```

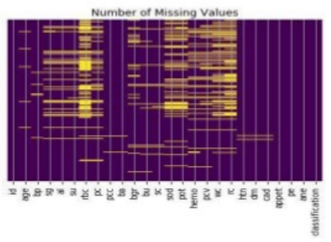
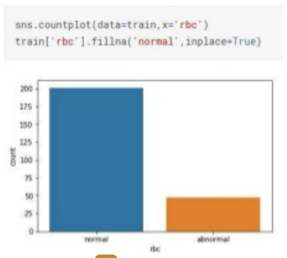


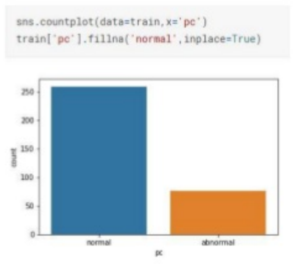
Figure 4.11: The heatmap showing all the 400 values as yellow lines

I now selected the 10 most important feature vectors out of the 25 in the list to proceed with the model training tasks and hence achieved 10 different graphs regarding the accuracy scores of the model on each feature versus the other.



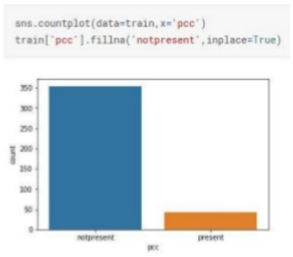
```
sns.countplot(data=train,x='rbc')
train['rbc'].fillna('normal',inplace=True)
```

Fig 4.12(a)



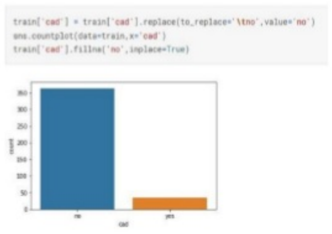
```
sns.countplot(data=train,x='pc')
train['pc'].fillna('normal',inplace=True)
```

Fig 4.12(b)



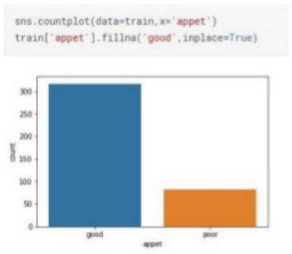
```
sns.countplot(data=train,x='pcc')
train['pcc'].fillna('notpresent',inplace=True)
```

Fig 4.12(c)



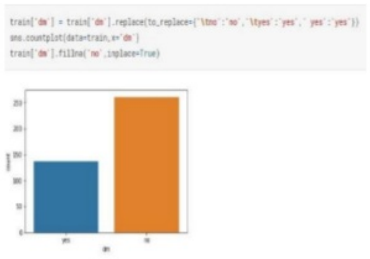
```
train['cad'] = train['cad'].replace(to_replace='\tno',value='no')
sns.countplot(data=train,x='cad')
train['cad'].fillna('no',inplace=True)
```

Fig 4.12(d)



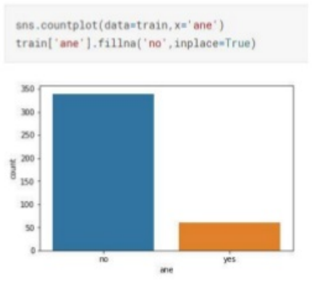
```
sns.countplot(data=train,x='appet')
train['appet'].fillna('good',inplace=True)
```

Fig 4.12(e)



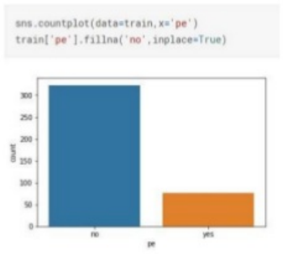
```
train['dm'] = train['dm'].replace(to_replace={'\tno':'no','\tyes':'yes',' yes':'yes'})
sns.countplot(data=train,x='dm')
train['dm'].fillna('no',inplace=True)
```

Fig 4.12(f)



```
sns.countplot(data=train,x='ane')
train['ane'].fillna('no',inplace=True)
```

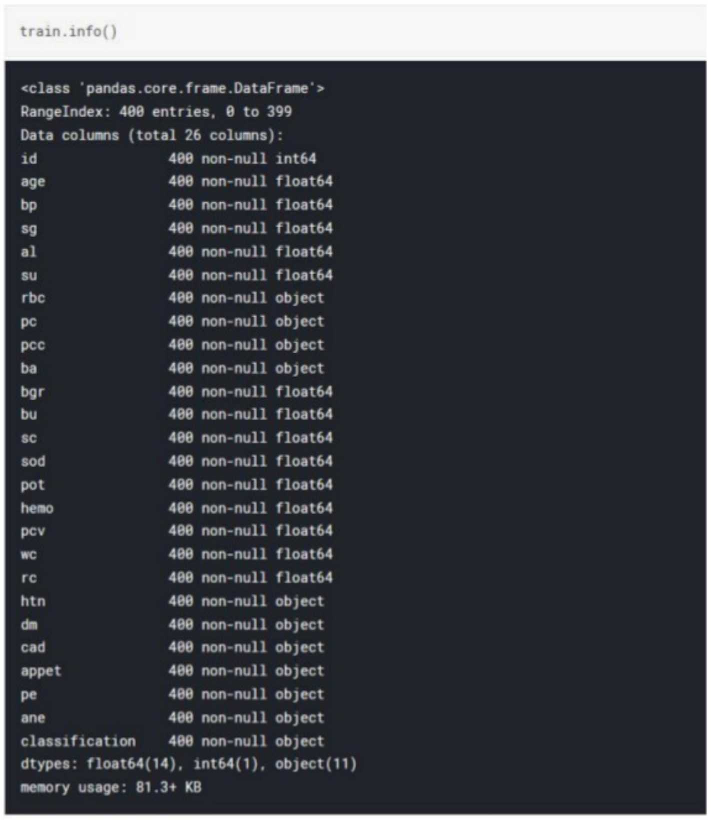
Fig 4.12(g)



```
sns.countplot(data=train,x='pe')
train['pe'].fillna('no',inplace=True)
```

Fig 4.12(h)

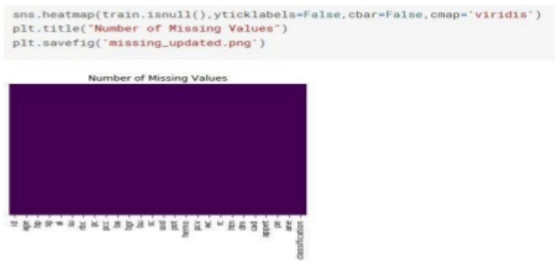
Training the model now results in a much cleaner and expected data formation in which there is no missing / left out data entry that can cause accuracy issues in the training model. The resulting heatmap is also blank which means no data value was stale or left behind in the pre-processing.



```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
id                400 non-null int64
age               400 non-null float64
bp                400 non-null float64
sg                400 non-null float64
al                400 non-null float64
su                400 non-null float64
rbc               400 non-null object
pc                400 non-null object
pcc               400 non-null object
ba                400 non-null object
bgr               400 non-null float64
bu                400 non-null float64
sc                400 non-null float64
sod               400 non-null float64
pot               400 non-null float64
hemo              400 non-null float64
pcv               400 non-null float64
wc                400 non-null float64
rc                400 non-null float64
htn               400 non-null object
dm                400 non-null object
cad               400 non-null object
appet             400 non-null object
pe                400 non-null object
ane               400 non-null object
classification    400 non-null object
dtypes: float64(14), int64(1), object(11)
memory usage: 81.3+ KB
```

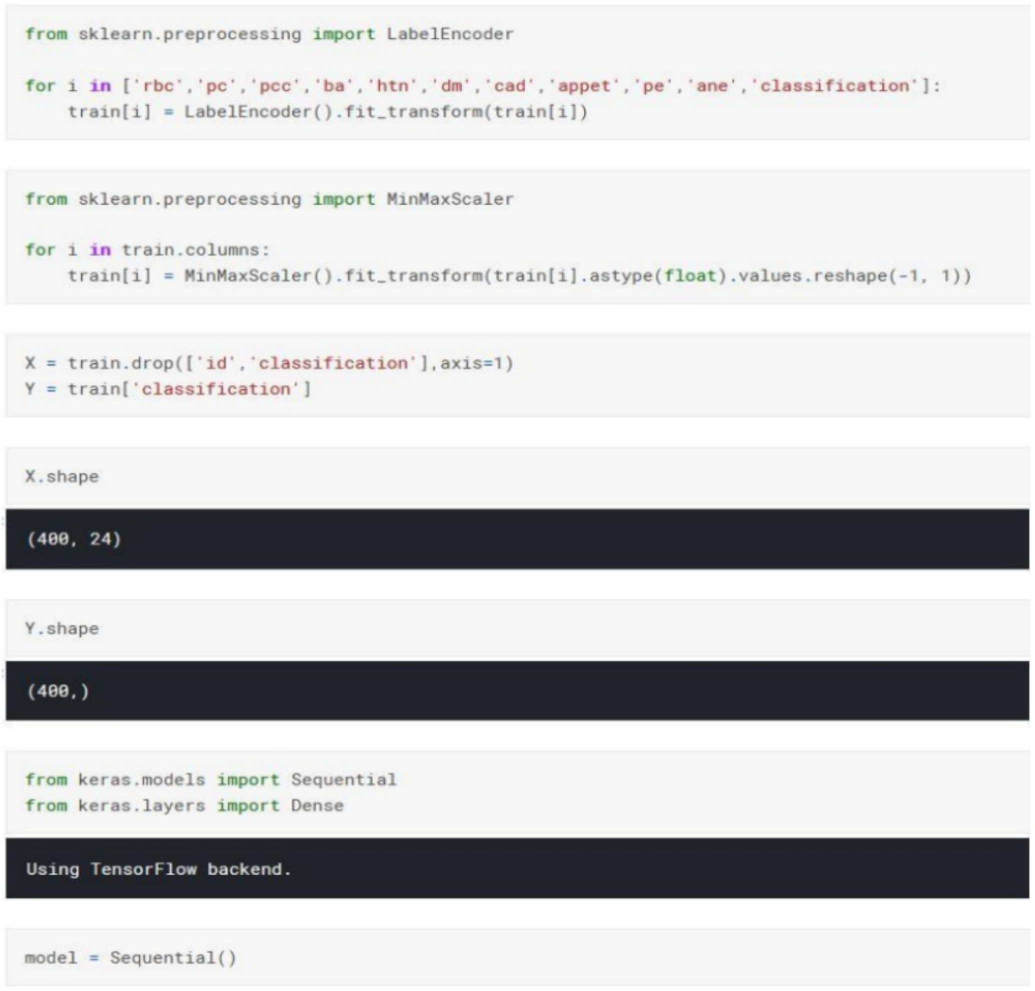
Figure 4.13: Trained CDNN after feeding data values



```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
plt.title("Number of Missing Values")
plt.savefig('missing_updated.png')
```

Figure 4.14: Blank heatmap (no yellow lines) means no unfiltered data was left behind

Finally, I use the **MinMaxScaler** and **LabelEncoder** from sklearn.preprocessing library from the sci-kit learn toolkit and collectively find out the results as 50 Epochs per run, training 320 samples with 80 for testing in a Sequential analysis TensorFlow environment.



```
from sklearn.preprocessing import LabelEncoder

for i in ['rbc','pc','pcc','ba','htn','dm','cad','appet','pe','ane','classification']:
    train[i] = LabelEncoder().fit_transform(train[i])
```

```
from sklearn.preprocessing import MinMaxScaler

for i in train.columns:
    train[i] = MinMaxScaler().fit_transform(train[i].astype(float).values.reshape(-1, 1))
```

```
X = train.drop(['id','classification'],axis=1)
Y = train['classification']
```

```
X.shape
```

```
(400, 24)
```

```
Y.shape
```

```
(400,)
```

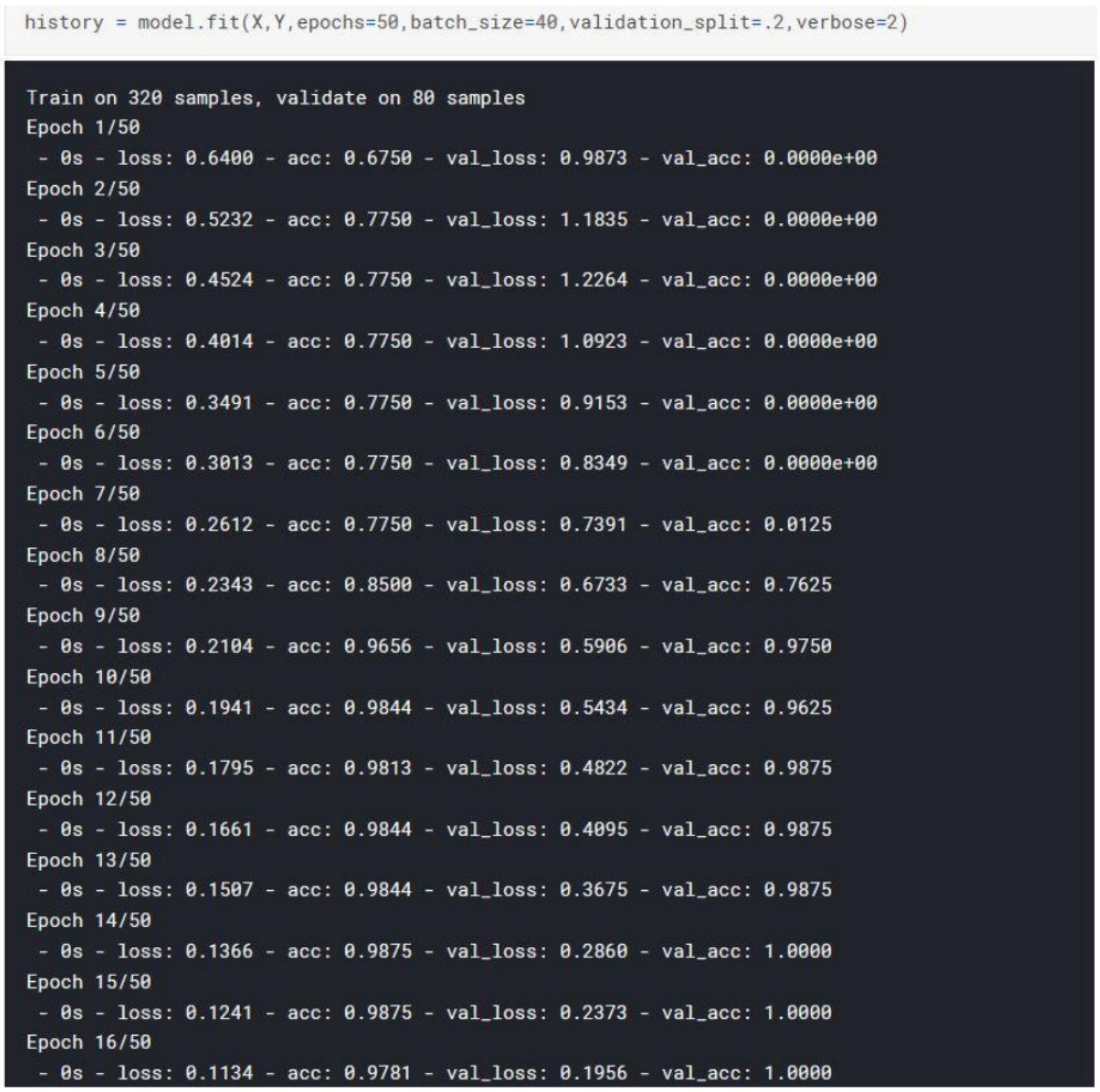
```
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

```
model = Sequential()
```

Figure 4.15: Using the MinMaxScaler and LabelEncoder

It was time for the results, and the accuracy was surprisingly high. I do notice that even after scaling, the results are still around same after the 16/50 Epoch.

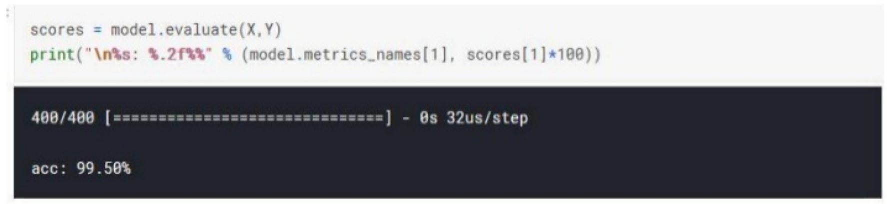


```
history = model.fit(X,Y,epochs=50,batch_size=40,validation_split=.2,verbose=2)
```

```
Train on 320 samples, validate on 80 samples
Epoch 1/50
 - 0s - loss: 0.6400 - acc: 0.6750 - val_loss: 0.9873 - val_acc: 0.0000e+00
Epoch 2/50
 - 0s - loss: 0.5232 - acc: 0.7750 - val_loss: 1.1835 - val_acc: 0.0000e+00
Epoch 3/50
 - 0s - loss: 0.4524 - acc: 0.7750 - val_loss: 1.2264 - val_acc: 0.0000e+00
Epoch 4/50
 - 0s - loss: 0.4014 - acc: 0.7750 - val_loss: 1.0923 - val_acc: 0.0000e+00
Epoch 5/50
 - 0s - loss: 0.3491 - acc: 0.7750 - val_loss: 0.9153 - val_acc: 0.0000e+00
Epoch 6/50
 - 0s - loss: 0.3013 - acc: 0.7750 - val_loss: 0.8349 - val_acc: 0.0000e+00
Epoch 7/50
 - 0s - loss: 0.2612 - acc: 0.7750 - val_loss: 0.7391 - val_acc: 0.0125
Epoch 8/50
 - 0s - loss: 0.2343 - acc: 0.8500 - val_loss: 0.6733 - val_acc: 0.7625
Epoch 9/50
 - 0s - loss: 0.2104 - acc: 0.9656 - val_loss: 0.5906 - val_acc: 0.9750
Epoch 10/50
 - 0s - loss: 0.1941 - acc: 0.9844 - val_loss: 0.5434 - val_acc: 0.9625
Epoch 11/50
 - 0s - loss: 0.1795 - acc: 0.9813 - val_loss: 0.4822 - val_acc: 0.9875
Epoch 12/50
 - 0s - loss: 0.1661 - acc: 0.9844 - val_loss: 0.4095 - val_acc: 0.9875
Epoch 13/50
 - 0s - loss: 0.1507 - acc: 0.9844 - val_loss: 0.3675 - val_acc: 0.9875
Epoch 14/50
 - 0s - loss: 0.1366 - acc: 0.9875 - val_loss: 0.2860 - val_acc: 1.0000
Epoch 15/50
 - 0s - loss: 0.1241 - acc: 0.9875 - val_loss: 0.2373 - val_acc: 1.0000
Epoch 16/50
 - 0s - loss: 0.1134 - acc: 0.9781 - val_loss: 0.1956 - val_acc: 1.0000
```

Figure 4.16: The results of 16/50 Epoch runs resulting in **1.0000** accuracy

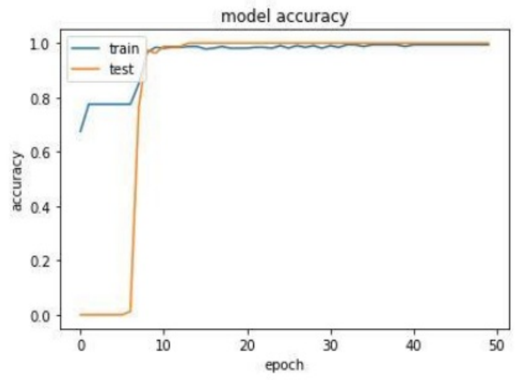
I then got the final total accuracy for the neural network and analysed it with the resultant graphs.



```
scores = model.evaluate(X,Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
400/400 [==============================] - 0s 32us/step

acc: 99.50%
```

Figure 4.17: Final Accuracy Resulting to **99.50%**



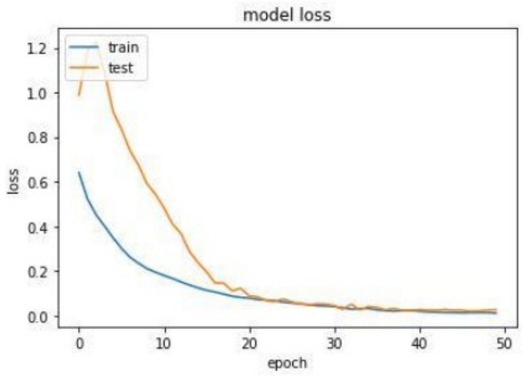


Figure 4.18(a): Accuracy Graph **Figure 4.18(b):** Loss Graph

The final algorithm I will be using will be:

Artificial Neural Network (ANN)

I use the same steps as the CDNN but the only difference would be in the selection criteria of vectors and their analysis. This is displayed by the set of images below.

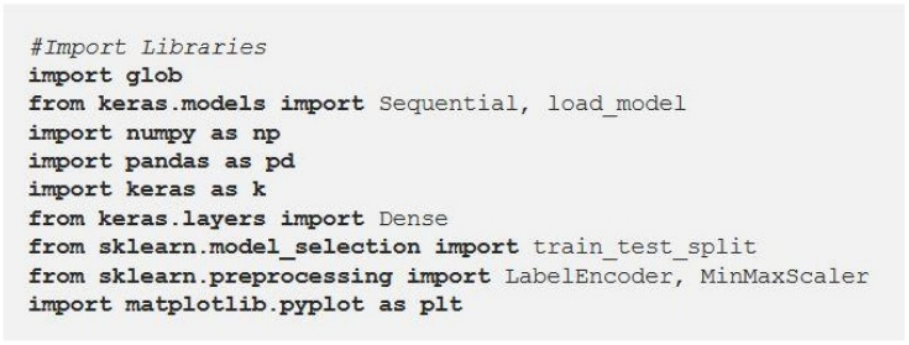
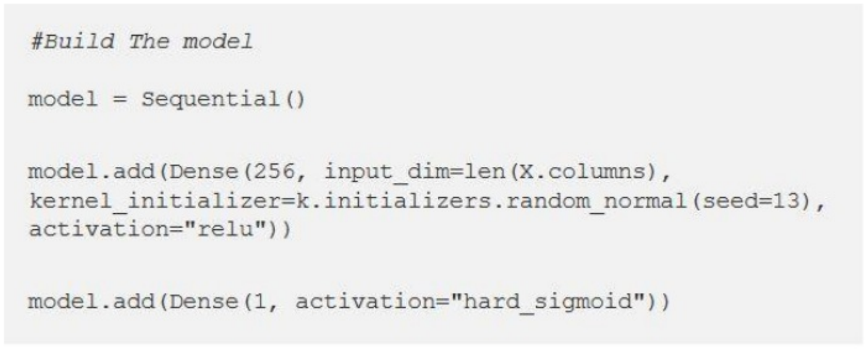
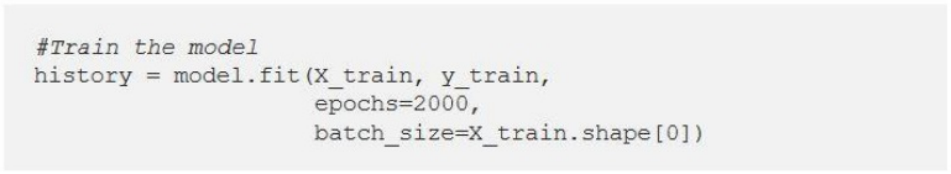

```
#Import Libraries
import glob
from keras.models import Sequential, load_model
import numpy as np
import pandas as pd
import keras as k
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import matplotlib.pyplot as plt
```

Figure 4.19: Importing Libraries



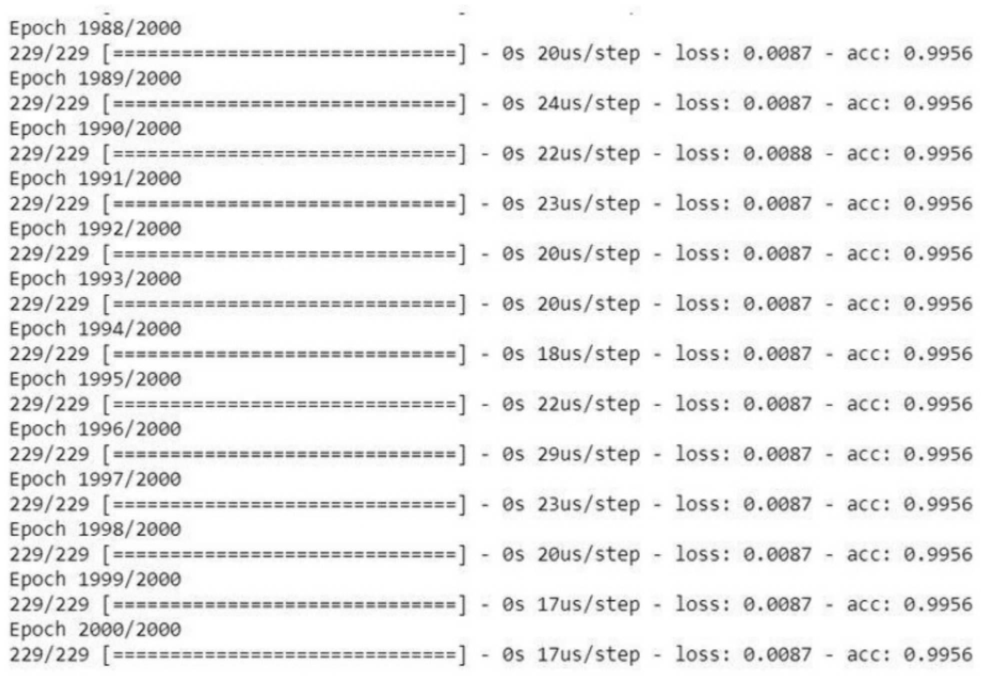
```
#Build The model

model = Sequential()

model.add(Dense(256, input_dim=len(X.columns),
kernel_initializer=k.initializers.random_normal(seed=13),
activation="relu"))

model.add(Dense(1, activation="hard_sigmoid"))
```

Figure 4.20: Building the ANN Model



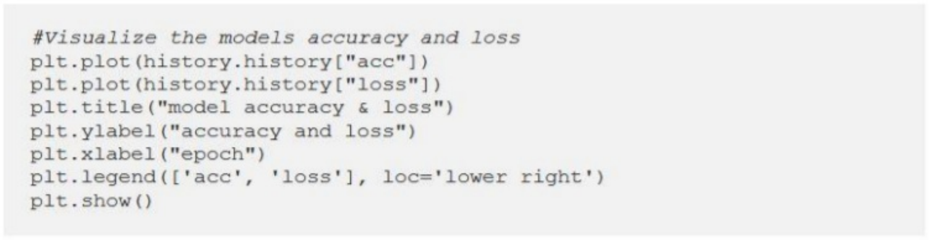
```
#Train the model
history = model.fit(X_train, y_train,
                    epochs=2000,
                    batch_size=X_train.shape[0])
```

Figure 4.21: Training the Model



```
Epoch 1988/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1989/2000
229/229 [==============================] - 0s 24us/step - loss: 0.0087 - acc: 0.9956
Epoch 1990/2000
229/229 [==============================] - 0s 22us/step - loss: 0.0088 - acc: 0.9956
Epoch 1991/2000
229/229 [==============================] - 0s 23us/step - loss: 0.0087 - acc: 0.9956
Epoch 1992/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1993/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1994/2000
229/229 [==============================] - 0s 18us/step - loss: 0.0087 - acc: 0.9956
Epoch 1995/2000
229/229 [==============================] - 0s 22us/step - loss: 0.0087 - acc: 0.9956
Epoch 1996/2000
229/229 [==============================] - 0s 29us/step - loss: 0.0087 - acc: 0.9956
Epoch 1997/2000
229/229 [==============================] - 0s 23us/step - loss: 0.0087 - acc: 0.9956
Epoch 1998/2000
229/229 [==============================] - 0s 20us/step - loss: 0.0087 - acc: 0.9956
Epoch 1999/2000
229/229 [==============================] - 0s 17us/step - loss: 0.0087 - acc: 0.9956
Epoch 2000/2000
229/229 [==============================] - 0s 17us/step - loss: 0.0087 - acc: 0.9956
```

Figure 4.22: Results of 2000 Epoch with accuracy **99.56%**



```
#Visualize the models accuracy and loss
plt.plot(history.history["acc"])
plt.plot(history.history["loss"])
plt.title("model accuracy & loss")
plt.ylabel("accuracy and loss")
plt.xlabel("epoch")
plt.legend(['acc', 'loss'], loc='lower right')
plt.show()
```

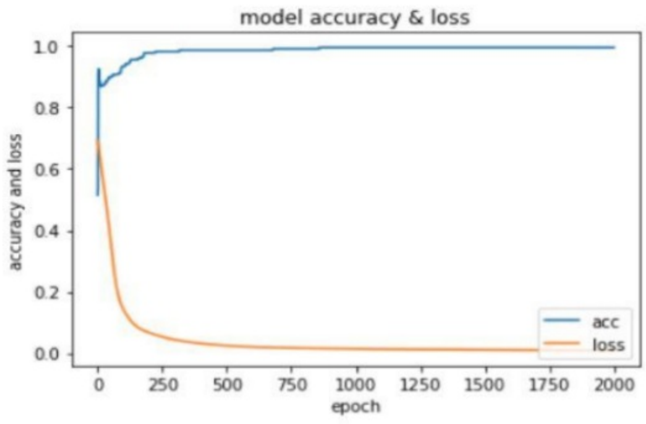
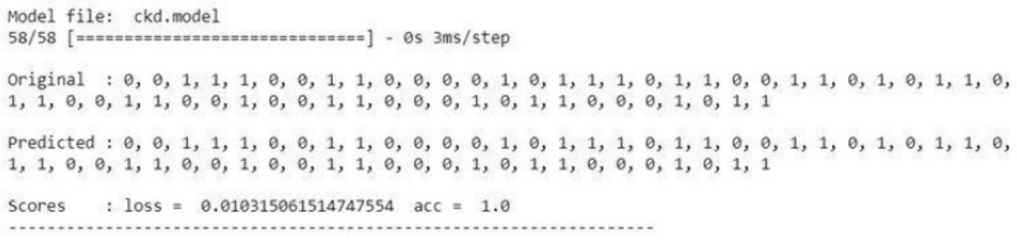


Fig 4: The models loss (orange) & accuracy (blue)



```
Model file:  ckd.model
58/58 [==============================] - 0s 3ms/step

Original  : 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1

Predicted : 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1

Scores    : loss =  0.010315061514747554  acc =  1.0
-----------------------------------------------------------------
```

Printing the model(s) output.

Figure 4.23: Results of the ANN model

As we see from the results of the ANN model, it gives an accuracy score of 99.56 going upwards to 99.997% (~1.00), which is by far the best result out of the three models I trained.

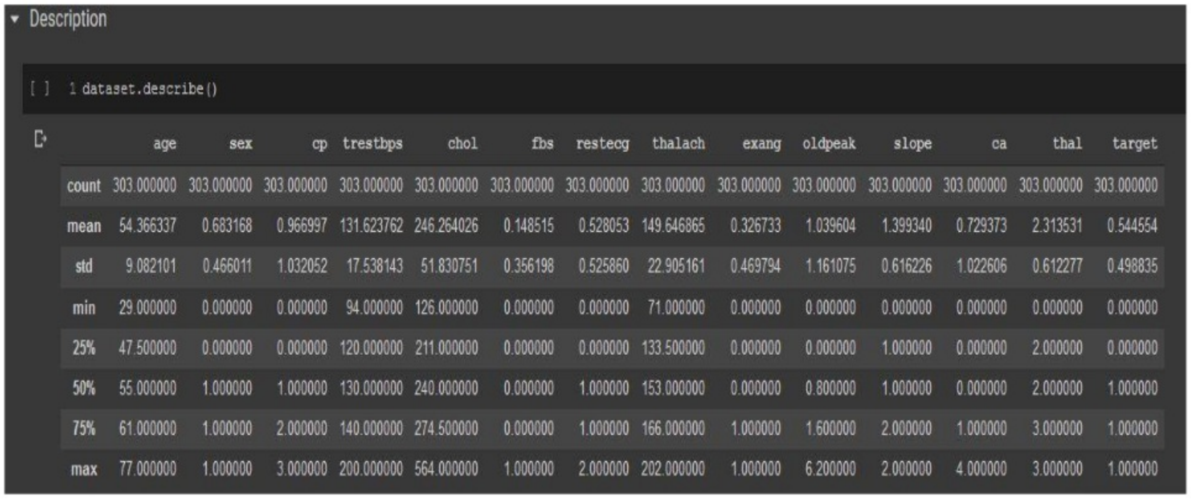
CHAPTER-5

IMPLEMENTATION (Heart Failure)

I moved onto my next organ being the most vital organs of all, the heart. This project involves analysis of the heart disease patient dataset with proper data processing. After studying about the various factors that are taken into consideration for the measurement of severity of the disease, we would be making a judgement based upon the given 14 heart related attributes of 303 patients to foresee the consequence (i.e. does one have CHD).

Decent data-driven systems aimed at foreseeing heart diseases can expand the entire study and preclusion procedure, making certain that more individuals can live in a good physical shape. Machine learning helps in predicting heart diseases, and these predictions made are quite accurate. I've used a variety of machine learning algorithms like KNN, Decision Tree, Random Forest, SVM, Logistic Regression etc. implemented in python, to predict the presence of heart disease in a patient.

This time onwards, I worked in a more streamlined manner so as to keep things clean and sorted in an orderly fashion.



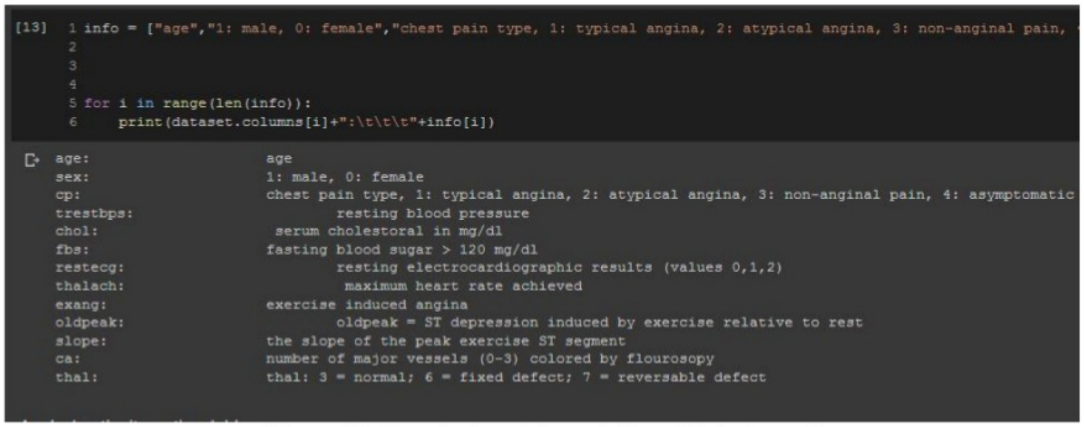
Description

```
dataset.describe()
```

```
              age         sex          cp    trestbps        chol         fbs     restecg     thalach       exang     oldpeak       slope          ca        thal      target
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean    54.366337    0.683168    0.966997  131.623762  246.264026    0.148515    0.528053  149.646865    0.326733    1.039604    1.399340    0.729373    2.313531    0.544554
std      9.082101    0.466011    1.032052   17.538143   51.830751    0.356198    0.525860   22.905161    0.469794    1.161075    0.616226    1.022606    0.612277    0.498835
min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000000    0.000000   71.000000    0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%     47.500000    0.000000    0.000000  120.000000  211.000000    0.000000    0.000000  133.500000    0.000000    0.000000    1.000000    0.000000    2.000000    0.000000
50%     55.000000    1.000000    1.000000  130.000000  240.000000    0.000000    1.000000  153.000000    0.000000    0.800000    1.000000    0.000000    2.000000    1.000000
75%     61.000000    1.000000    2.000000  140.000000  274.500000    0.000000    1.000000  166.000000    1.000000    1.600000    2.000000    1.000000    3.000000    1.000000
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000000    2.000000  202.000000    1.000000    6.200000    2.000000    4.000000    3.000000    1.000000
```

Figure 5.1: Dataset of 303 heart patients with 14 selective parameters

After uploading the .csv on my Google colab notebook, I did some basic operations on the dataset for a better understanding of it. From the operations I found out that I was working upon 303 entities with 14 vectors having one as a “target” variable which can take up a value of 0 or 1 accordingly.



```
[13] info = ["age","1: male, 0: female","chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain,



     for i in range(len(info)):
         print(dataset.columns[i]+":\t\t\t"+info[i])
```

```
age:                       age
sex:                       1: male, 0: female
cp:                        chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic
trestbps:                          resting blood pressure
chol:                       serum cholestoral in mg/dl
fbs:                       fasting blood sugar > 120 mg/dl
restecg:                           resting electrocardiographic results (values 0,1,2)
thalach:                            maximum heart rate achieved
exang:                     exercise induced angina
oldpeak:                           oldpeak = ST depression induced by exercise relative to rest
slope:                     the slope of the peak exercise ST segment
ca:                        number of major vessels (0-3) colored by flourosopy
thal:                      thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
```

Figure 5.2: Description of the variables with the ‘target’

The next few graphs will be portraying some of the above variables as a target and would be further judging the possibility of a heart disease being present, through graphs.

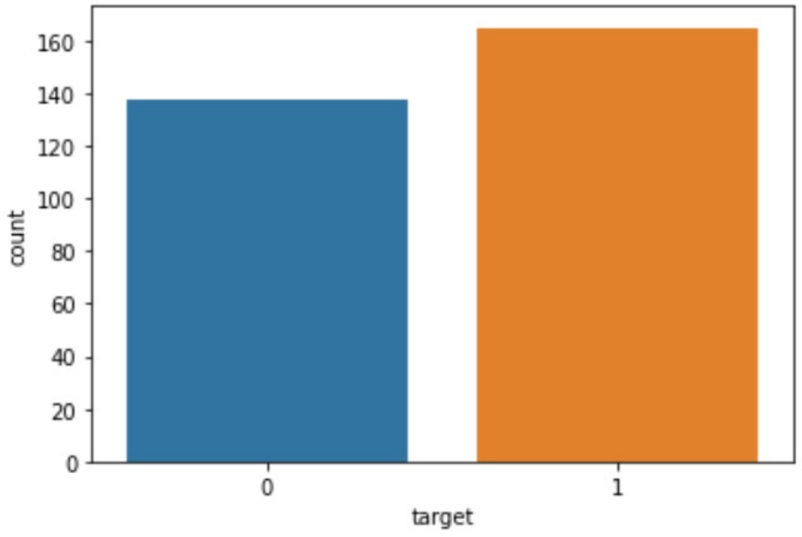


Figure 5.3: We notice that females are more likely to have heart disease than males

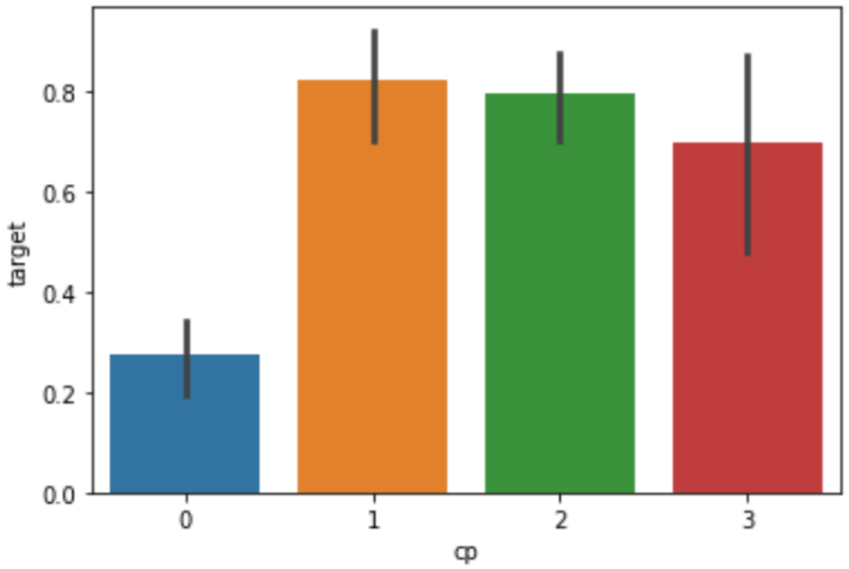


Figure 5.4: The chest pain '0' are the ones with typical angina who are less likely to have heart problems

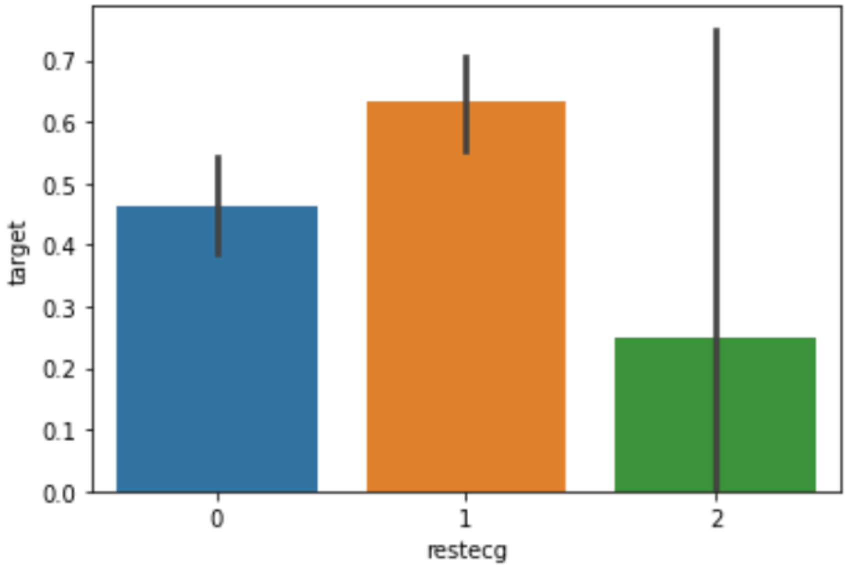


Figure 5.5: We realise that people with restecg '1' and '0' are much more likely to have a heart disease

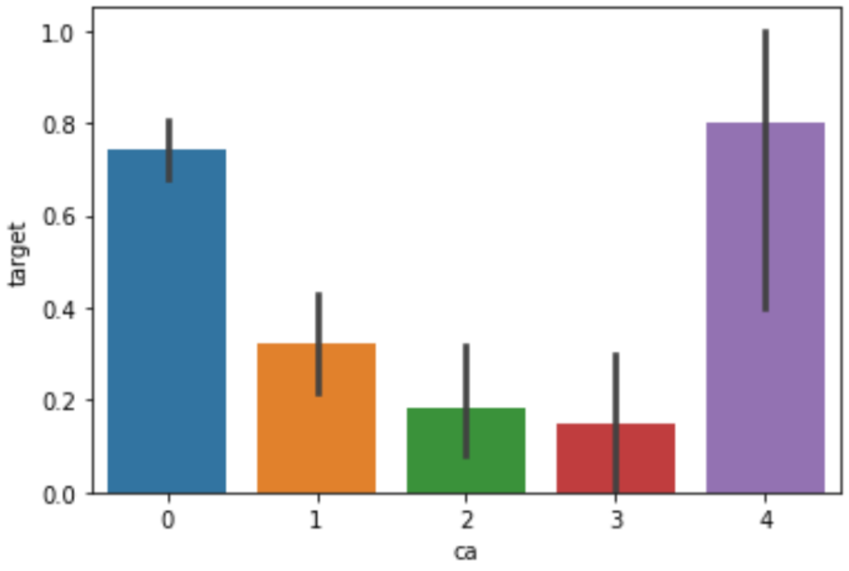
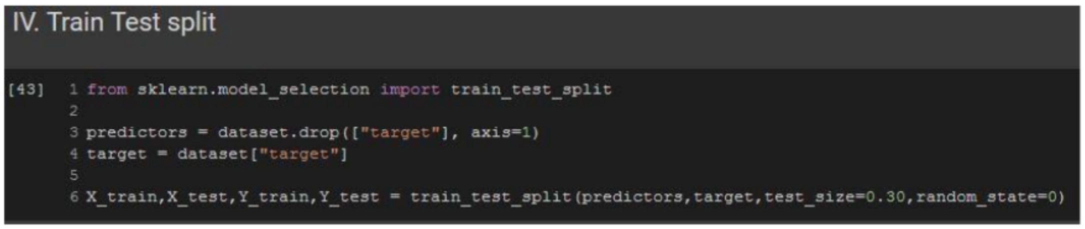


Figure 5.6: ca=4 has astonishingly large number of heart patients

As always, we split the data into test and training, keeping the latter in majority, to train various upcoming model algorithms with the data we have and sorted up. Always keeping a 70/30 or 60/40 train-test ratio helps.



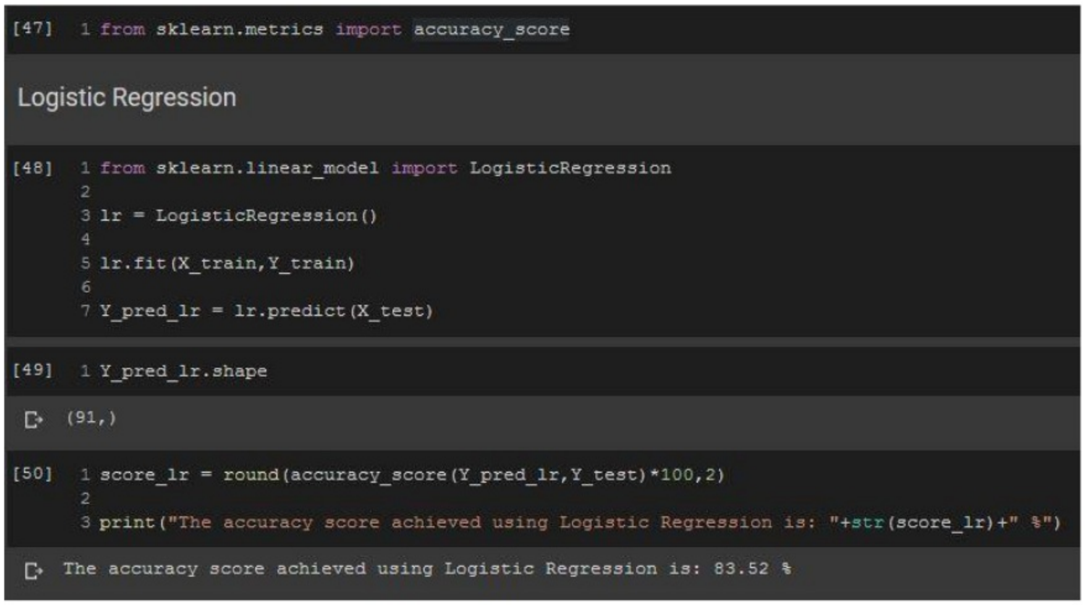
IV. Train Test split

```
[43]  from sklearn.model_selection import train_test_split

      predictors = dataset.drop(["target"], axis=1)
      target = dataset["target"]

      X_train,X_test,Y_train,Y_test = train_test_split(predictors,target,test_size=0.30,random_state=0)
```

Figure 5.7: 70% training and 30% testing split

Moving on to the testing phase of our model, we implement several modern machine learning model algorithms, and thoroughly train the former with our training data, so as to see which one of them would be the most effective (highest accuracy in the least amount of average time).

Logistic Regression



```
[47] from sklearn.metrics import accuracy_score
```

Logistic Regression

```
[48] from sklearn.linear_model import LogisticRegression

     lr = LogisticRegression()

     lr.fit(X_train,Y_train)

     Y_pred_lr = lr.predict(X_test)
```

```
[49] Y_pred_lr.shape
```

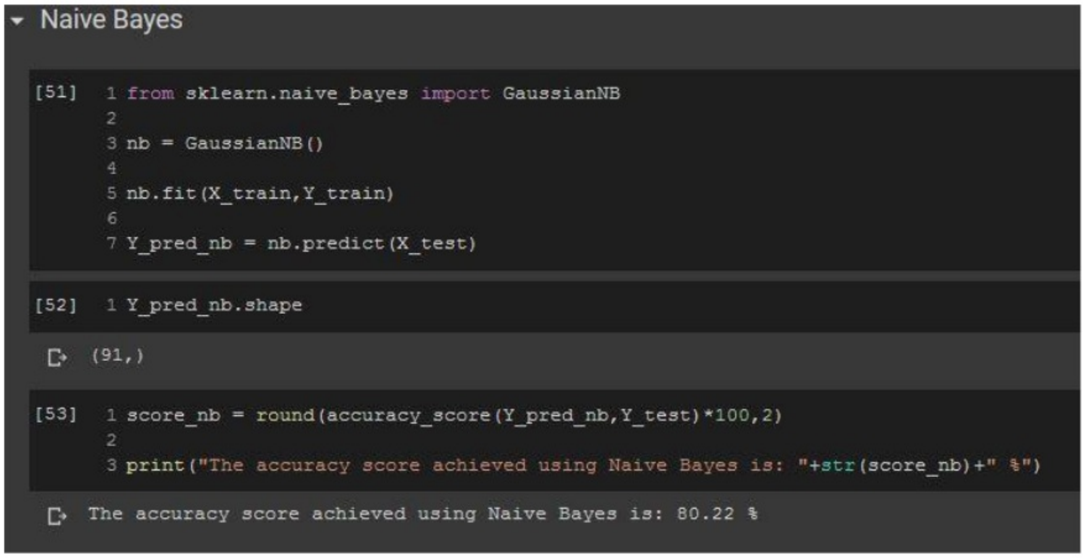
```
(91,)
```

```
[50] score_lr = round(accuracy_score(Y_pred_lr,Y_test)*100,2)

     print("The accuracy score achieved using Logistic Regression is: "+str(score_lr)+" %")
```

```
The accuracy score achieved using Logistic Regression is: 83.52 %
```

Figure 5.8: **Logistic Regression** gave us an accuracy score of **83.52%**

Naïve Bayes



Naive Bayes

```
[51] from sklearn.naive_bayes import GaussianNB

     nb = GaussianNB()

     nb.fit(X_train,Y_train)

     Y_pred_nb = nb.predict(X_test)
```

```
[52] Y_pred_nb.shape
```

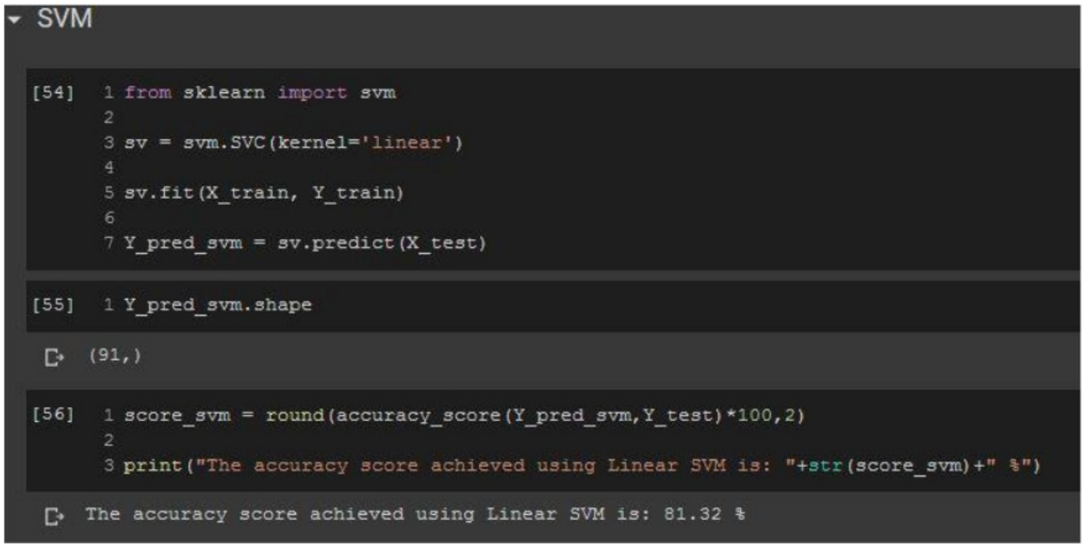
```
(91,)
```

```
[53] score_nb = round(accuracy_score(Y_pred_nb,Y_test)*100,2)

     print("The accuracy score achieved using Naive Bayes is: "+str(score_nb)+" %")
```

```
The accuracy score achieved using Naive Bayes is: 80.22 %
```

Figure 5.9: **Naïve Bayes** gave us an accuracy score of **80.22%**

Support Vector Machine (SVM)


SVM

```
[54] from sklearn import svm

     sv = svm.SVC(kernel='linear')

     sv.fit(X_train, Y_train)

     Y_pred_svm = sv.predict(X_test)
```

```
[55] Y_pred_svm.shape
```

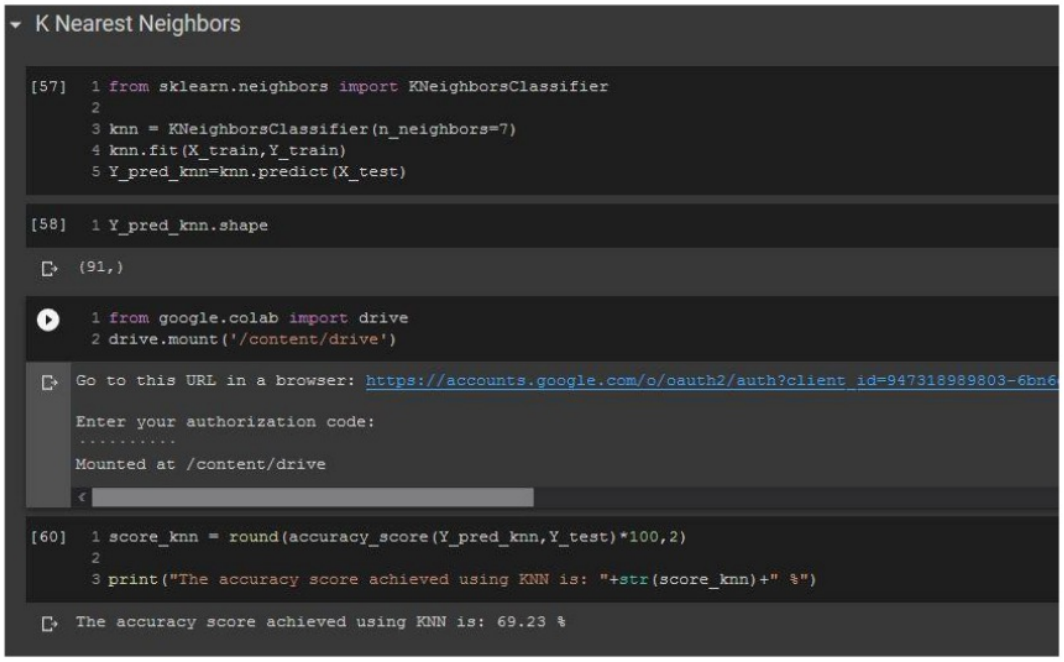
```
(91,)
```

```
[56] score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)

     print("The accuracy score achieved using Linear SVM is: "+str(score_svm)+" %")
```

```
The accuracy score achieved using Linear SVM is: 81.32 %
```

Figure 5.10: **SVM** gave us an accuracy score of **81.32%**

K Nearest Neighbours (K-NN)


K Nearest Neighbors

```
[57] from sklearn.neighbors import KNeighborsClassifier

     knn = KNeighborsClassifier(n_neighbors=7)
     knn.fit(X_train,Y_train)
     Y_pred_knn=knn.predict(X_test)
```

```
[58] Y_pred_knn.shape
```

```
(91,)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

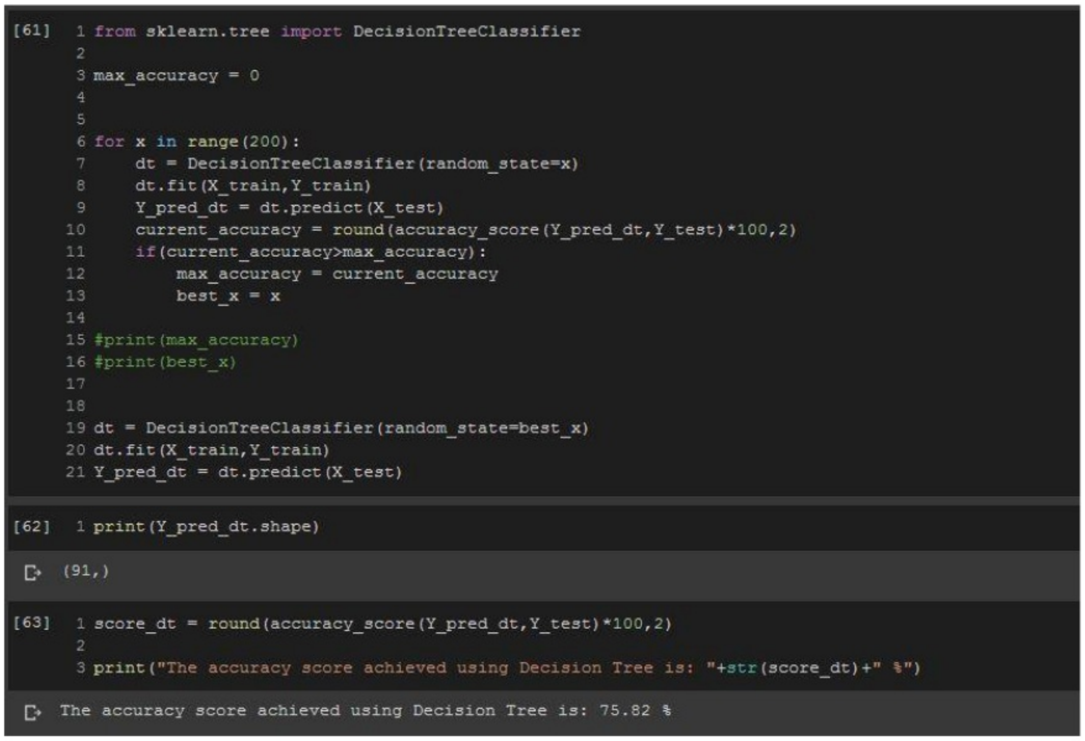
```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6

Enter your authorization code:
··········
Mounted at /content/drive
```

```
[60] score_knn = round(accuracy_score(Y_pred_knn,Y_test)*100,2)

     print("The accuracy score achieved using KNN is: "+str(score_knn)+" %")
```

```
The accuracy score achieved using KNN is: 69.23 %
```

Figure 5.11: **K-NN** gave us an accuracy score of **69.23%**

Decision Tree



```
[61]
from sklearn.tree import DecisionTreeClassifier

max_accuracy = 0


for x in range(200):
    dt = DecisionTreeClassifier(random_state=x)
    dt.fit(X_train,Y_train)
    Y_pred_dt = dt.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

#print(max_accuracy)
#print(best_x)


dt = DecisionTreeClassifier(random_state=best_x)
dt.fit(X_train,Y_train)
Y_pred_dt = dt.predict(X_test)
```

```
[62]
print(Y_pred_dt.shape)
```

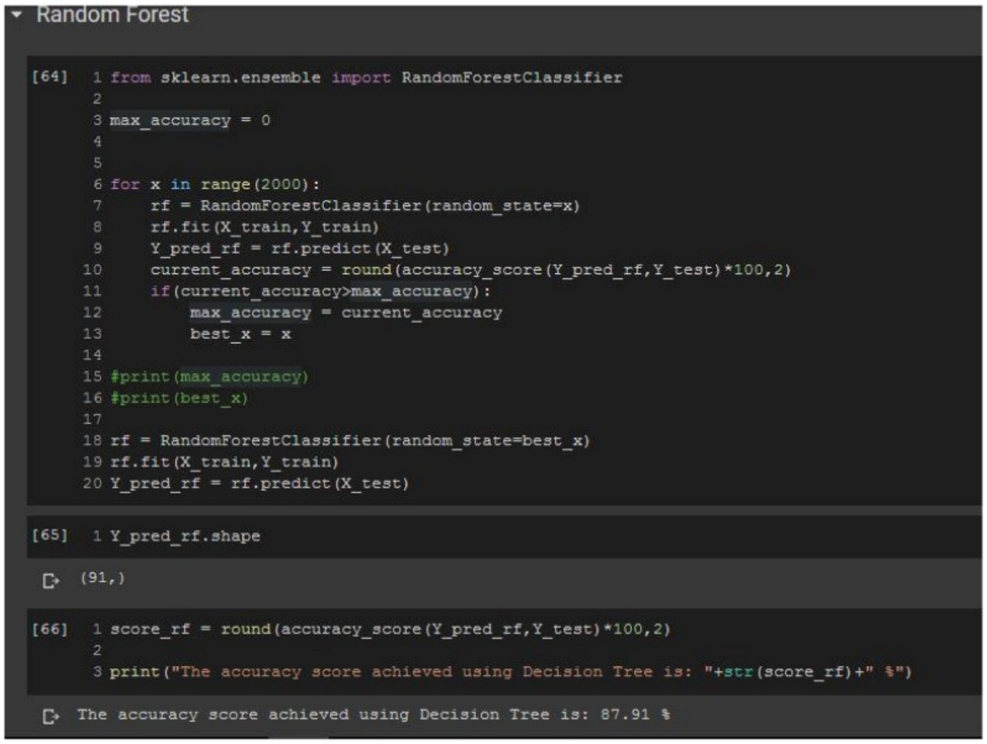
```
(91,)
```

```
[63]
score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)

print("The accuracy score achieved using Decision Tree is: "+str(score_dt)+" %")
```

```
The accuracy score achieved using Decision Tree is: 75.82 %
```

Figure 5.12: **Decision Tree** gave us an accuracy score of **75.82%**

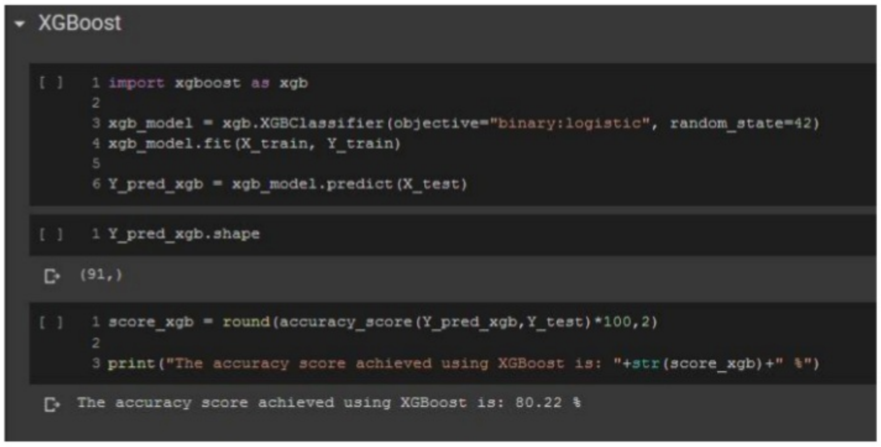
Random Forest Classifier



```
[64] from sklearn.ensemble import RandomForestClassifier

     max_accuracy = 0


     for x in range(2000):
         rf = RandomForestClassifier(random_state=x)
         rf.fit(X_train,Y_train)
         Y_pred_rf = rf.predict(X_test)
         current_accuracy = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
         if(current_accuracy>max_accuracy):
             max_accuracy = current_accuracy
             best_x = x

     #print(max_accuracy)
     #print(best_x)

     rf = RandomForestClassifier(random_state=best_x)
     rf.fit(X_train,Y_train)
     Y_pred_rf = rf.predict(X_test)

[65] Y_pred_rf.shape

     (91,)

[66] score_rf = round(accuracy_score(Y_pred_rf,Y_test)*100,2)

     print("The accuracy score achieved using Decision Tree is: "+str(score_rf)+" %")

     The accuracy score achieved using Decision Tree is: 87.91 %
```

Figure 5.13: **Random Forest Classifier** gave us an accuracy score of **87.91%**

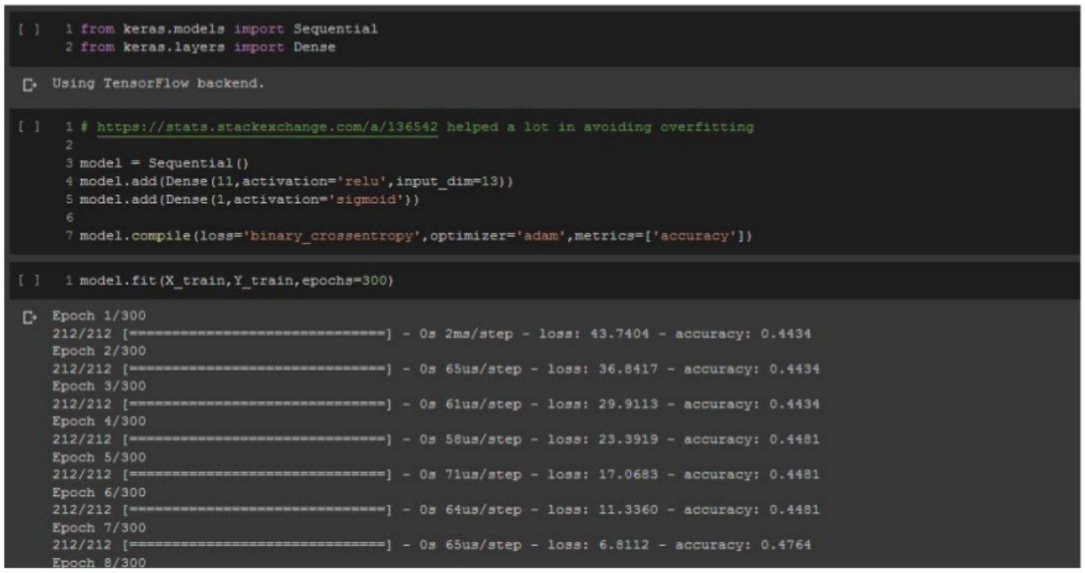
XGBoost



```
[ ] import xgboost as xgb

    xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
    xgb_model.fit(X_train, Y_train)

    Y_pred_xgb = xgb_model.predict(X_test)

[ ] Y_pred_xgb.shape

    (91,)

[ ] score_xgb = round(accuracy_score(Y_pred_xgb,Y_test)*100,2)

    print("The accuracy score achieved using XGBoost is: "+str(score_xgb)+" %")

    The accuracy score achieved using XGBoost is: 80.22 %
```

Figure 5.14: XGBoost gave us an accuracy score of **80.22%**

Neural Network



```
from keras.models import Sequential
from keras.layers import Dense
```

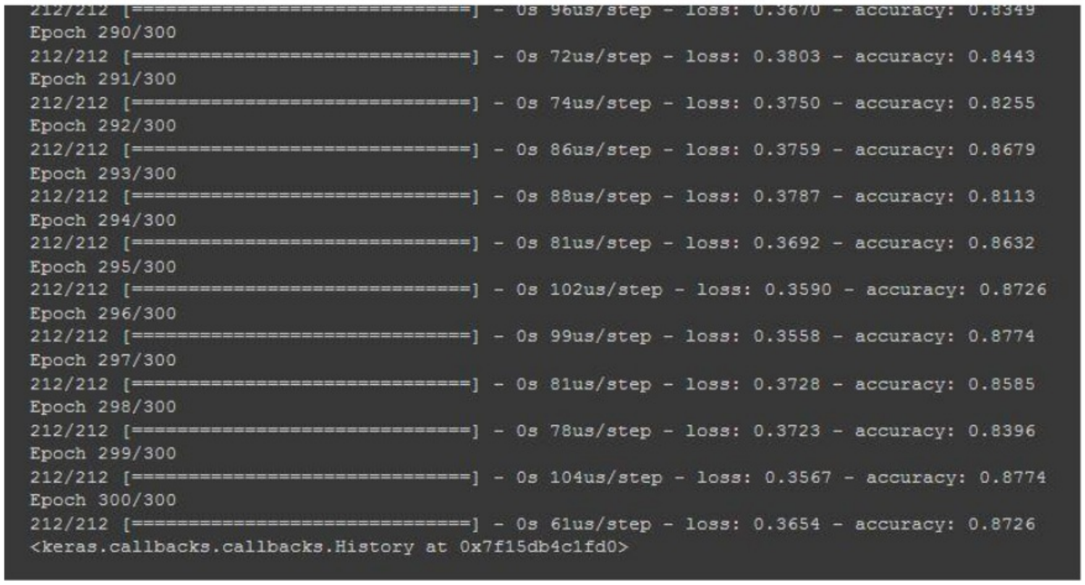
```
Using TensorFlow backend.
```

```
# https://stats.stackexchange.com/a/136542 helped a lot in avoiding overfitting

model = Sequential()
model.add(Dense(11,activation='relu',input_dim=13))
model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
model.fit(X_train,Y_train,epochs=300)
```

```
Epoch 1/300
212/212 [==============================] - 0s 2ms/step - loss: 43.7404 - accuracy: 0.4434
Epoch 2/300
212/212 [==============================] - 0s 65us/step - loss: 36.8417 - accuracy: 0.4434
Epoch 3/300
212/212 [==============================] - 0s 61us/step - loss: 29.9113 - accuracy: 0.4434
Epoch 4/300
212/212 [==============================] - 0s 58us/step - loss: 23.3919 - accuracy: 0.4481
Epoch 5/300
212/212 [==============================] - 0s 71us/step - loss: 17.0683 - accuracy: 0.4481
Epoch 6/300
212/212 [==============================] - 0s 64us/step - loss: 11.3360 - accuracy: 0.4481
Epoch 7/300
212/212 [==============================] - 0s 65us/step - loss: 6.8112 - accuracy: 0.4764
Epoch 8/300
```

Figure 5.15: The Neural Network was running for 300 epochs through Keras and TensorFlow backend



```
212/212 [==============================] - 0s 96us/step - loss: 0.3670 - accuracy: 0.8349
Epoch 290/300
212/212 [==============================] - 0s 72us/step - loss: 0.3803 - accuracy: 0.8443
Epoch 291/300
212/212 [==============================] - 0s 74us/step - loss: 0.3750 - accuracy: 0.8255
Epoch 292/300
212/212 [==============================] - 0s 86us/step - loss: 0.3759 - accuracy: 0.8679
Epoch 293/300
212/212 [==============================] - 0s 88us/step - loss: 0.3787 - accuracy: 0.8113
Epoch 294/300
212/212 [==============================] - 0s 81us/step - loss: 0.3692 - accuracy: 0.8632
Epoch 295/300
212/212 [==============================] - 0s 102us/step - loss: 0.3590 - accuracy: 0.8726
Epoch 296/300
212/212 [==============================] - 0s 99us/step - loss: 0.3558 - accuracy: 0.8774
Epoch 297/300
212/212 [==============================] - 0s 81us/step - loss: 0.3728 - accuracy: 0.8585
Epoch 298/300
212/212 [==============================] - 0s 78us/step - loss: 0.3723 - accuracy: 0.8396
Epoch 299/300
212/212 [==============================] - 0s 104us/step - loss: 0.3567 - accuracy: 0.8774
Epoch 300/300
212/212 [==============================] - 0s 61us/step - loss: 0.3654 - accuracy: 0.8726
<keras.callbacks.callbacks.History at 0x7f15db4c1fd0>
```

Figure 5.16: Neural Network gave us an accuracy score of **81.32%**

CHAPTER-6

RESULTS AND ANALYSIS

I can now say that I successfully implemented and trained the three mentioned models for our chronic kidney disease dataset, namely:

Random Forest Classifier, CDNN and ANN

and for our chronic heart disease dataset, namely:

Logistic Regression, Naïve Bayes, SVM, K-NN, Decision Tree, Random Forest, XGBoost and Neural Network

The aforementioned models are tested on several constraints and topographies such as TN, TP, FN and FP and the measures of Exactness and Loss were considered.

For the kidney models, the results were as follows:

Classifier	**Accuracy**	**Loss**
Random Forest Classifier	88.843%	11.157%
CDNN	99.501%	0.497%
ANN	99.993%	0.007%

Table 6.1: Results of comparative analysis

After looking at these results, I settled on the argument that from all the algorithms that were tried upon our kidney disease dataset, **ANN (Artificial Neural Network)** gives out an accuracy of 99.993%

In consideration of the heart disease models, the results are as follows:

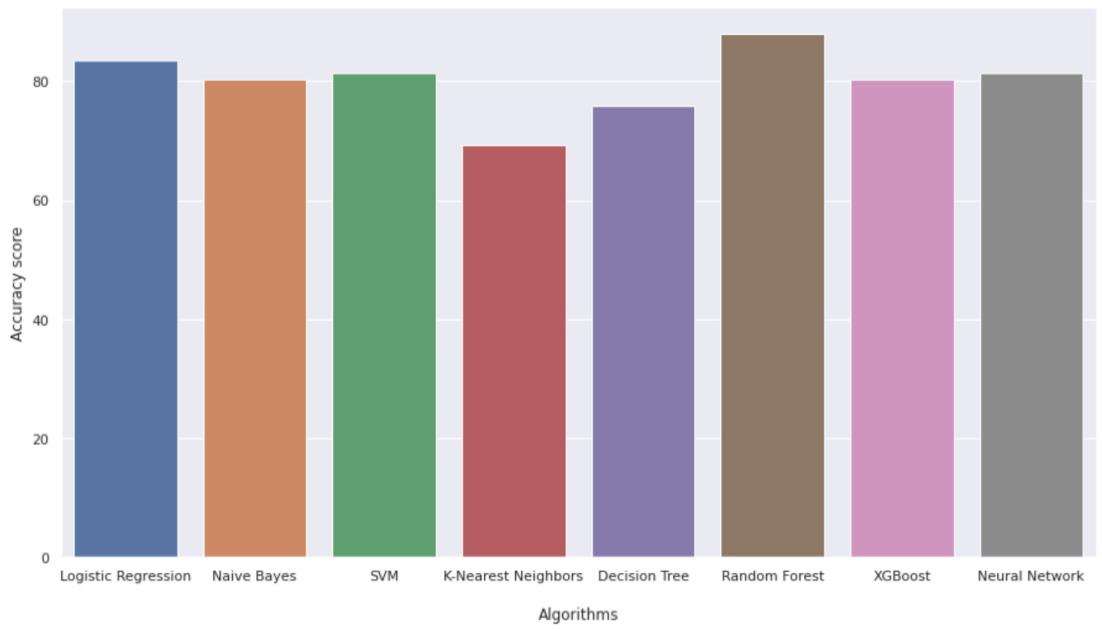


Figure 6.1: Final Results for the various algorithms

According to the results, the **Random Forest Classifier** comes at the top with an accuracy score of **87.91%**, with the **Logistic Regression** obtaining a 2nd place with an accuracy score of **83.52%**, with the **Neural Networks** and **SVM** at a tied score of **81.32%**, followed by a tie of the **Naïve Bayes** and **XGBoost** at **80.22%**, ending with the **Decision Trees** at **75.82%** and **K-NN** score the lowest at a 69.23%, proving to be least used in this case.

Below is a list of all the accuracy scores of the various algorithms used.

Classifier	Accuracy
Random Forest	**87.91%**
Logistic Regression	**83.52%**
Neural Networks	**81.32%**
SVM	**81.32%**
Naïve Bayes	**80.22%**
XGBoost	**80.22%**
Decision Trees	**75.82%**
K-NN	**69.23%**

Table 6.2: Final Results for the various algorithms

CHAPTER-7

CONCLUSION

In this project, I have actualized different AI algorithms particularly for discovery of Chronic Kidney Disorder (CKD) as well as the Chronic Heart Disease (CHD). For the classification, I used named datasets which were made using 25 of the selected attribute vectors contributing to various features of the kidney dataset, and 13 of the selected attribute vectors for the various features of the heart disease dataset. I was able to use learning algorithms upon these datasets by splitting into testing and training sets. Using the training portions, I considered the standards of different constraints and compared those features of all the different algorithms, concluding on these algorithms, provided the highest accuracy value is the **Artificial Neural Network (ANN)** for the kidney and the **Random Forest Classifier** for the heart. Thus, with an accuracy value of **99.993% (kidney)** and **87.91% (heart)**, these are the most appropriate algorithms for our present datasets. After getting the results, it is superlative to state that I obtained most of the samples of those diagnosed of CKD and CHD. Concluding, ML algorithms, particularly neural networks work best for the kidney and heart disease datasets.

FURTHER SCOPE OF THE PROJECT

After learning about the numerous ML algorithms and their subsequent application on the used datasets, I was highly motivated to work upon gathering data samples of various other organs of the human body so as to create a one-for-all machine model that can help in fast discovery and treatment of diseases (if any).

Due to the current worldwide pandemic of the Coronavirus COVID-19 and with millions of people suffering from it, I will be working upon the Lung Disease database mainly based upon the covid-19 results next, using these and various other algorithms to obtain highly accurate results just like our kidney failure and heart disease detection engines. I would be glad to work upon this project as a part of a larger industry on a bigger and more sophisticated scale, so as to help many in saving their lives and improving the industry standards.

REFERENCES

[1] Tahsim M. Rahman, S. Siddiqua, Siam -E- Rabby, “Early Detection of Kidney Diseases Using ECG Signals Through Machine Learning Based Modelling”. In the proceedings of the International Conference on Robotics, Electrical and Signal Processing Techniques. **(ICREST- 2019)**

[2] A. Abdelaziz, A. M. Riad, A. S. Salama, A. N. Mahmoud, “A Machine Learning Model for Predicting of Chronic Kidney Disease Based Internet of Things and Cloud Computing in Smart Cities” **(2019)**

[3] H. Mohamadlou, A. Lynn-Palevsky, C. Barton, U. Chettipally, L. Shieh, J. Calvert, N. R. Saber, R. Das, “Prediction of Acute Kidney Injury with a Machine Learning Algorithm Using Electronic Health Record Data” **(2018)**

[4] "Intro to Machine Learning | Udacity." Intro to Machine Learning | Udacity. [https://www.udacity.com/course/intro-to-machine-learning--ud120].

[5] "Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd Edition. Datasets: Coronary Heart Disease Dataset." Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd Edition. [http://statweb.stanford.edu/~tibs/ElemStatLearn/].

[6] Manju M N, Mayur G K, Shachi D S, Chandan J, “Student Performance Evaluation Using Predictive Analysis”. International Research Journal of Engineering and Technology (IRJET) **2020**

[7] A. Rairikar, V. Kulkarni, V. Sabale, “Heart Disease Prediction Using Data Mining Techniques”. International Conference on Intelligent Computing and Control (I2C2) **2017**

[8] "No Free Lunch Theorems." No Free Lunch Theorems. [http://www.no-free-lunch.org/].

[9] "https://www.sciencedirect.com/topics/engineering/confusion-matrix"

[10] "https://www.researchgate.net/publication/303326261_Machine_Learning_Project"

[11] "https://link.springer.com/book/10.1007%2F978-981-15-1216-2"

[12] "https://archive.ics.uci.edu/ml/datasets/Heart+Disease"

[13] "https://towardsdatascience.com/heart-disease-prediction-73468d630cfc"

[14] "https://medium.com/@dskswu/machine-learning-with-a-heart-predicting-heart-disease-b2e9f24fee84"

[15] "https://www.engpaper.com/medical/heart-disease-prediction.html"

DATASET INFORMATION (LEGACY)

I use the following representations to collect the dataset:

1. age - age
2. bp - blood pressure
3. sg - specific gravity
4. al - albumin
5. su - sugar
6. rbc - red blood cells
7. pc - pus cell
8. pcc - pus cell clumps
9. ba - bacteria
10. bgr - blood glucose random
11. bu - blood urea
12. sc - serum creatinine
13. sod - sodium
14. pot - potassium
15. hemo - haemoglobin
16. pcv - packed cell volume
17. wc - white blood cell count
18. rc - red blood cell count
19. htn - hypertension
20. dm - diabetes mellitus
21. cad - coronary artery disease
22. appet - appetite
23. pe - pedal edema
24. ane - anaemia
25. class – class

ATTRIBUTE INFORMATION

I use 24 + class = 25 (11 numeric ,14 nominal)

1.Age(numerical) - age in years

2.Blood Pressure(numerical) - bp in mm/Hg

3.Specific Gravity(nominal) - sg - (1.005,1.010,1.015,1.020,1.025)

4.Albumin(nominal) - al - (0,1,2,3,4,5)

5.Sugar(nominal) - su - (0,1,2,3,4,5)

6.Red Blood Cells(nominal) - rbc - (normal, abnormal)

7.Pus Cell (nominal) - pc - (normal, abnormal)

8.Pus Cell clumps(nominal) - pcc - (present, not present)

9.Bacteria(nominal) - ba - (present, not present)

10.Blood Glucose Random(numerical) - bgr in mgs/dl

11.Blood Urea(numerical) - bu in mgs/dl

12.Serum Creatinine(numerical) - sc in mgs/dl

13.Sodium(numerical) - sod in mEq/L

14.Potassium(numerical) - pot in mEq/L

15.Haemoglobin(numerical) - hemo in gms

16.Packed Cell Volume(numerical) – cc in cells/cmm

17.White Blood Cell Count(numerical) - wc in cells/cmm

18.Red Blood Cell Count(numerical) - rc in millions/cmm

19.Hypertension(nominal) - htn - (yes, no)

20.Diabetes Mellitus(nominal) - dm - (yes, no)

21.Coronary Artery Disease(nominal) - cad - (yes, no)

22.Appetite(nominal) - appet - (good, poor)

23.Pedal Edema(nominal) - pe - (yes, no)

24.Anemia(nominal) - ane - (yes, no)

25.Class (nominal) - class - (ckd, not ckd

Proj_Report

ORIGINALITY REPORT

16%	5%	4%	16%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Jaypee University of Information Technology Student Paper	8%
2	archive.ics.uci.edu Internet Source	3%
3	Submitted to University of Bradford Student Paper	2%
4	Submitted to Indian Institute of Information Technology, Allahabad Student Paper	1%
5	Submitted to Higher Education Commission Pakistan Student Paper	<1%
6	Submitted to Victoria University Student Paper	<1%
7	Submitted to Middlesex University Student Paper	<1%
8	gyan.iitg.ernet.in Internet Source	<1%

9	freedomrunners.org Internet Source	<1%

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		