

SUNWAY
UNIVERSITY



Capstone Project Report - Final Report

Heart Disease Prediction Using Machine Learning

by
Saurabh Varughese M. Koor
(20017604)

Bachelor of Computer Science (Hons)

Supervisor: Dr. Richard Wong Teck Ken
Date: 18/11/2022

Abstract

The primary focus of this paper is to present the comparative study that has been performed using the different machine learning (ML) techniques or classifiers and their performance in predicting, classifying, and detecting heart or cardiovascular disease and based on that, the designed prediction algorithm for classifying and forecasting the occurrence and level of such disease in patients based on their input data. This designed prediction algorithm is then deployed in a production web application format which was developed using the Django web framework and deployed to the AWS cloud hosting environment, to ease and coordinate the end-user in interacting with the algorithm, by entering their risk factor values, so the prediction algorithm can relay an accurately computed output.

Therefore, this paper will start by studying, analysing and comparing existing systems and documentation in this problem domain in regard to machine learning and heart disease screening/classification. Based on the summary of the literature review that reflects the current status quo and limitations of ML-based heart disease prediction and classification systems, the parameter selections for our study will be determined, including the specific ML algorithms to design and develop, dataset preparation techniques, features utilised, data split ratios, hyperparameter tuning methods to employ as well as evaluation metrics, and model selection method to prioritise. It is found from the model comparisons and from testing the designed and implemented ML models that alterations in these departments result in a change in the mentioned evaluation metrics, and so the optimal model, producing the highest evaluation metric performances, is sought after. Based on this, the implementation of our own machine learning models to develop a predictive algorithm was performed and will be explored in more depth. Conclusively, from the ML implementation process and model comparison performed, the StackingClassifier model with the Random Forest model, with the best prediction accuracy scores (86.9%), minimal error rates (0.311 mean square error), and acceptable testing times (0.0522s) for web application deployment. After that, the development of the web app component is discussed, which is used to house the chosen best ML model, making it easily accessible for the end-user through their web browser, allowing them to input their physiological information that are potential risk factors and be used for predicting and classifying their risk of developing heart or cardiovascular disease. From the unit and integration testing performed, the web application was implemented successfully and fulfils the requirements and system functionalities of the requirements plan. Additionally, the hosted web application is accessible through its domain name, <https://heartassist.net/>, which was registered and configured to point to it, allowing the platform to be accessible on any device through the use of a web browser.

Therefore, the project consists of 2 main sections, which include the ML model construction and the web app development for this project of developing an ML-based heart disease prediction and classification system. The project source code is accessible through this onedrive link (https://imailsunwayedu-my.sharepoint.com/:f/g/personal/20017604_imail_sunway_edu_my/Er_6WNkzLWBDt7RuLh9Rz5sBGXeO4fi5o4uf-M0szDS8Kg?e=CinA3w) or in this Github repository (<https://github.com/saurabhkovoovr/heartassist>). The instructions to execute the source code are available in the README.md file.

Keywords: Heart Disease, Cardiovascular Disease (CVD), Coronary Artery Disease (CAD), Artificial Intelligence, Machine Learning, Classification, Feature Selection

Table of Contents

1. Introduction	1
1.1 Motivation of the Project	1
1.2 Problem Statement	2
1.3 Project Goal	4
1.4 Project Objectives	4
1.5 Project Deliverables	4
1.6 Project Scope	5
2. Literature Review	7
2.1 Machine Learning, Prediction, Classification and Data Mining	7
2.1.1 Simple Logistic Regression	8
2.1.2 Decision Tree	9
2.1.3 Naive Bayes	9
2.1.4 Random Forest	10
2.1.5 Support-Vector Machine	10
2.1.6 K-Nearest Neighbour (KNN)	11
2.1.7 Artificial Neural Network (ANN)	12
2.1.8 Recurrent Neural Network (RNN)	12
2.1.9 K-Means Clustering	13
2.2 Cardiovascular Diseases (CVDs) and Heart Diseases (HD)	14
2.3 Electrocardiography (ECG)	15
2.4 Existing Systems and Literature Analysis	17
2.4.1 Heart Disease Detection by Using Machine Learning Algorithms and a Real-Time Cardiovascular Health Monitoring System [9]	18
2.4.2 Developing a Hyperparameter Tuning Based Machine Learning Approach of Heart Disease Prediction [35]	18
2.4.3 Classification and Feature Selection Approaches by Machine Learning Techniques: Heart Disease Prediction [36]	19
2.4.4 Machine Learning-Based Classification Algorithms for the Prediction of Coronary Heart Diseases [37]	20
2.4.5 Prediction of Heart Disease by Classifying With Feature Selection and Machine Learning Methods [7]	21
2.4.6 Heart Disease Diagnosis Based On Deep Learning Network [38]	23
2.4.7 A Deep Learning Method for Prediction of Cardiovascular Disease Using Convolutional Neural Network [39]	23
2.4.8 Cardiovascular disease risk prediction using automated machine learning: A prospective study of 423,604 UK Biobank participants [40]	24
2.4.9 Summary of Existing System and Literature Analysis	27

3. Methodology	33
3.1 Phases of Implementation	33
3.2 Graphical Flow/Flowchart of Components for the Methodology/Implementation	34
3.3 Software Process Model	35
3.4 Machine Learning Implementation	39
3.4.1 Data Acquisition and Dataset Description	39
3.4.2 Data Understanding and Visualisation	45
3.4.3 Data Cleaning	54
3.4.4 Dataset Splitting	58
3.4.5 Model Creation (Hyperparameter Tuning and Model Development and Training)	58
3.4.6 Model Validation/Evaluation Metrics	80
3.4.7 Model Selection	85
3.5 Requirements Plan for Web Application	87
3.5.1 Requirements Elicitation	87
3.5.2 System Functionality or Software Requirement Specification (SRS)	89
3.6 Technologies and Tools Required	91
3.7 Web App Prototype Design	96
3.8 UML Diagrams	110
3.9 Web App Implementation	117
4. Results and Discussion	127
5. Conclusion	139
6. References	142
7. Appendices	146
Appendix A: Results of ML model validation and evaluation metrics	146
Appendix B: Unit Test Cases	160
Appendix C: Integration Test Cases	190

List of Figures

Figure 1: Main components of a cardiac cycle recorded in an ECG [33]	16
Figure 2: Flowchart of main phases of the project and its implementation	34
Figure 3: Flowchart of main phases of the ML implementation process	35
Figure 4: Waterfall development model flowchart	36
Figure 5: Shape of the dataset and general info about dataset	46
Figure 6: Descriptive statistics of the dataset's numerical attributes	46
Figure 7: Univariate plots of density graphs of the dataset's numerical attributes	48
Figure 8: Univariate plots of bar charts of the dataset's categorical attributes	49
Figure 9: Univariate plots of box plots of the dataset's numerical attributes	50
Figure 10: Multivariate plots in heatmap form of numerical attributes with the target "num"	51
Figure 11: Multivariate plots in bar chart form of numerical attributes with the target	51
Figure 12: Multivariate plots in bar chart form of individual categorical attributes	52
Figure 13: The mathematical formulation for the min-max scaling	57
Figure 14: Support Vector Machine hyperplane identification	62
Figure 15: K-Nearest Neighbour algorithm operation	64
Figure 16: Results of testing different MLPClassifier activation function and solver	74
Figure 17: Accuracy score for MLPClassifier model with 1 hidden layer	75
Figure 18: Accuracy score for MLPClassifier model with 2 hidden layer	75
Figure 19: Accuracy score for MLPClassifier model with 3 hidden layer	76
Figure 20: Results of testing different MLPClassifier hidden layer combinations	76
Figure 21: Accuracy score for MLPClassifier model with maximum iterations from 1-1000	78
Figure 22: Results of testing different MLPClassifier maximum iterations	79
Figure 23: Confusion Matrix	81
Figure 24: Prototype design of the Home/Calculator Page	97
Figure 25: Prototype design of the Home/Calculator Page With Results	99
Figure 26: Prototype design of the Patient Registration Form	101
Figure 27: Prototype design of the Doctor Registration Form	101
Figure 28: Prototype design of the Admin Registration Form	102
Figure 29: Prototype design of the Login Form	103
Figure 30: Prototype design of the Patient Account Page	104
Figure 31: Prototype design of the Doctor Account Page	106
Figure 32: Prototype design of the Admin Account Page	108
Figure 33: Use case diagram	111
Figure 34: Sequence diagram	113
Figure 35: Activity diagram	115
Figure 36: Class diagram	117
Figure 37: Output of feature importance score of dataset attributes in the terminal	134

Figure 38: Visualisation of feature importance score of the dataset attributes in a bar chart 134

List of Tables

Table 1: Summary of Existing System and Literature Analysis	27
Table 2: Dataset description	40
Table 3: Results of ML model validation and evaluation metrics for 80:20 split ratio	128
Table 4: Summary of Unit Testing	136
Table 5: Summary of Integration Testing	138

1. Introduction

1.1 Motivation of the Project

Cardiovascular diseases (CVDs) or heart diseases (HDs) are the leading cause of deaths globally with a mortality rate of 17.9 million each year according to the World Health Organisation, WHO [1]. In fact, CVDs are especially prevalent in our country, Malaysia, regardless of race, as Ischaemic heart diseases remain the principal cause of death, where in 2019 alone, 16,374 medical-certified deaths were recorded [2]. However, at its early stages, CVDs are indeed preventable as patients should report to healthcare professionals to begin medicinal assistance and counselling to prevent their condition from worsening. Besides that, CVDs can be prevented through mitigating behavioural risk factors, including tobacco usage, unbalanced and unhealthy dietary patterns, obesity, physical inactivity, and excessive intake of alcohol [3]. Overall, the odds of surviving are favourable if the issue is discovered and diagnosed earlier on.

Despite that, according to the World Heart Federation, CVDs and HDs are especially prevalent in low and middle-income countries, where over 75% of CVD-related deaths occur [4]. One of the factors resulting in these monstrous numbers is that such cases are typically detected and treated late. This is mainly attributed to the majority of the population in these areas not having the financial capacity to be able to visit a healthcare professional to undergo these extensive tests. Not only do they have to travel to these healthcare institutions multiple times, but they will also have to spend a significant amount of time awaiting a clear analysis of their results from a limited personnel of cardiologists and at times these tests can come at a burdening cost, especially for patients in an area where healthcare is not subsidised or covered. On top of that, even if one has the financial capacity, they will typically only undergo an annual checkup on their heart health. Therefore, this expenditure for healthcare can be significant not just for the patient, but for national and corporate budgets, providing treatment for these asymptomatic diseases [5].

That being said, the traditional method of heart health screening involves the patient having to visit a health institution, to undergo an electrocardiogram (ECG) test and meet with a heart specialist or cardiologist. Plus, analysing the ECG data following the traditional method can be laborious and time-consuming as a cardiologist would have to manually inspect and analyse the ECG trace or the electrocardiogram for any abnormalities, before being able to provide a diagnosis [6].

Even then, although it is possible for a patient to be saved by a doctor after a minor or major attack, however, in many cases, a heart attack can still be fatal owing to the minimal amount of time during an attack period rendering the patient incapable of informing anyone about this emergency condition. Hence, these sudden attacks may lead to fatality before the patient is able to reach a doctor for consulting [3]. Therefore, precautionary steps before the occurrence of heart attacks should be bolstered.

Therefore, many believe in the future of the healthcare sector, the goal should be to reduce the number of cases to treat not only once heart diseases have developed but instead to prevent its occurrence earlier on through data and algorithms. For this, biometric devices can be utilised that are capable of generating a multitude of data points by monitoring bodily signals or real-time biomarkers, such as heart rate variability, nutrition analytes, temperature, etc. For instance,

wearable sensory technology that is becoming ever more easily accessible can be utilised, such as a smartwatch, sleep ring, heart monitor, etc. Thus, in the future, such technologies can be capable of continuous and constant tracking of the heart patterns and make more accurate and representative health predictions. From there, classification systems that utilise machine learning models can be applied to effectively process these large amounts of data to provide more accurate output or diagnoses that are backed by the data.

Therefore, as prevention is better than cure, this places high importance on developing prediction systems that can detect the onset of cardiovascular risk. Hence, the general purpose of this project is to develop a prediction algorithm for heart disease based on machine learning techniques, as seen in section 3.4.5. This project will help us identify key features or factors that are statistically significant to build these predictive models as well as to understand which machine learning algorithm/model is best suited for this application domain in providing the best performance or highest accuracy.

All in all, this project aims to contribute to the existing development of heart disease prediction systems by incorporating advanced machine learning techniques such as neural networks as well as testing along with a well-curated dataset and introducing hybrid data mining models to further combine other patient medical information that might be statistically significant and provide improved diagnosis and treatment. These benchmark algorithms are independently verified for their performance based on the set of the predetermined evaluation metrics, as seen in section 3.4.6. From there, we can determine the best-performing model that relates with this problem domain. Also, by making the technology more easily accessible, it can ease the technology's deployment for general public usage, so people can get access to information and diagnosis regarding their cardiovascular health quicker.

1.2 Problem Statement

By now, it is clear that there exists limitations in the traditional procedure of heart disease screening that we aim to alleviate by providing an improved solution with the development of this prediction system. Hence, this section highlights some problems that we can identify, which will help us pinpoint the specific goals and objectives the system should accomplish:

Firstly, as explored earlier, the traditional protocol of diagnosing heart disease can be rather extensive and time-consuming, which can especially be taxing on the limited personnel of cardiologists or heart specialists having to supervise or monitor the process. The extensive data in the ECG report would have to be manually inspected and analysed by a heart specialist in order to make valid inferences [6]. Along with that, multiple checkups and tests have to be scheduled to make a full diagnosis, which can not only be time-intensive but financially and resource-dependent as well.

Additionally, typically, at the first stage of the heart disease screening process, the diagnosis of the disease is not conducted with the help of computer-aided systems and the final decision or verdict is made through the analysis of the doctors. Thus, not only does this slow the process, this can introduce the risk or potential of human error to the diagnosis of the patients. These results or diagnoses can be affected by the conditions, education taken, working conditions, and the number of patients per physician along with a number of other factors [7]. Plus, these

traditional methods are mostly reactive and not precautionary, in that they examine a patient's medical history, symptoms and physical reports and only provides support after identifying the occurrence of heart disease than utilising the large amount of data points generated of patient data to make accurate predictions.

Plus, generally after the event of such attacks, there needs to be 24 hours of monitoring the patient's health at home, to diagnose any further fatal physiological conditions or symptoms [9]. A hospital environment is equipped with facilities that allow for this constant monitoring of critical condition patients, but after they are discharged, they are no longer part of the direct supervision. So, patients require this supervision to avoid the risks of undesired complications.

Looking at the current status quo, another reason ML is not widely or fully incorporated into medical screening applications is due to the lack of data mining and the utilisation of digital patient records and AI and computer systems in the healthcare or biomedical department not being so common, accessible or refined. In a traditional medical institutional environment medical data is often stored in the form of handwritten paper-based records or perhaps if they are more advanced, in computerised systems. However, even then, they are not typically utilised for real-world analysis [10]. Nevertheless, this data can be harnessed for predicting and analysing various diseases, including cancer, diabetes, and in this case, HDs. Despite that, the healthcare industry generates a large number of datapoints about patients' medical and physiological conditions daily, but most of it is not analysed thoroughly, raising the need for tools to extract these critical knowledge from the dataset for the clinical classification and prediction of disease [11]. So, with the inception of Electronic Medical Records (EMR), this has further assisted medical professionals and institutions transition to an electronic/digital management of patient records. Having this vast digital storage of data, assists with simplifying the acquisition, analysis, processing of these medical data to provide more accurate and valid inferences, predictions and classifications [11]. By having a centralised and easily accessible data source, this benefits the research division, where they can discover previously unknown patterns in the dataset to further refine the healthcare service provided. This data availability can assist our study and similar studies in the future, where these crucial data can be useful for further training and testing of the ML models to be even more accurate as well as development of novel computational techniques and various domain-specific applications, especially considering that the amount of local datasets and even ones that are open-source, is limited in numbers. Thus, by having a dataset that is representative of the local population, this can be effective to determine if there are location or social determinants or predictors for the occurrence of heart disease in patients.

It is clear that the medical field involves extensive manual and repetitive tasks to read and analyse the large amount of data points even for each patient let alone for all heart patients in a hospital. Hence, the diagnosis of disease is truly an intricate part of the medical field [12]. This further increases the demand for ML-based prediction and classification systems that can analyse the large amount of data and support the medical professional's decision making process. This can be achieved through the combination of data mining or sourcing the large amount of digital patient data that is available in the medical field and the development of ML algorithms to make intelligent decision-making programs. These big data available in health institution databases when correctly processed, can give rise to valid inferences and assist practitioners in making predictions and diagnoses.

Additionally, the technology was not advanced enough to make safe, accurate and decisive verdicts or diagnoses that can concern human health. So, even at its early stages of deployment, a heart specialist would be required to monitor and ensure the diagnoses are just and acceptable [9]. Nevertheless, as mentioned before, such technologies can certainly supplement the doctor's decision-making process, by processing large amounts of patient data to give a more simple and understandable output. Hence, the human factor will not be eliminated in the decision-making process, but instead, the human factor will become even more effective and efficient as the entire process is accelerated and accurate decisions are made based on data, hence minimising human errors in the process. On that note, continuous development and research into this field should be conducted to further improve the state of the technology in terms of its performance (metrics) and easy acceptance by the target user base.

1.3 Project Goal

The primary aim or goal of this project is to design and develop a heart disease prediction algorithm using a machine learning model that produces the best performance based on our comparative study of different ML algorithms.

1.4 Project Objectives

In an effort to overcome the problems stated above, based on this generalised goal statement, we can further divide this into more specific objectives that we aim to accomplish by the end of this project. Thus, this section will give the first look at the system's capabilities:

1. To perform a comparative study and determine the ML algorithm (2.1.1-2.1.9) that provides the best performance in terms of the chosen evaluation metrics as stated in 3.4.6, for predicting the onset of heart disease based on ECG data input.
2. To study the effect of parameters changes on the model's performance, such as the dataset utilised, features/attributes used, data preparation and cleaning tasks, data split ratios, hyperparameter tuning, and the evaluation metric and model selection method prioritised.
3. To design and develop the ML model for effectively and accurately predicting and diagnosing heart disease based on patients' ECG data input. The proposed system will eliminate the need to conduct various testing and analysis of heart disease as in the traditional mode, and instead support and supplement the heart specialist's decision-making process in their diagnosis, simultaneously mitigating the extensive time for checkups. The system shall accept a singleton query and return/display a clear output of the HD presence and risk level.
4. To develop a production-ready web app using the Django python framework, along with other web tools and technologies further explained in section 3.6, that integrates the designed optimal ML model with a user interface (UI) that is easily accessible through a web browser by the end-user.

1.5 Project Deliverables

This project will be delivered or deployed in the form of a production-ready web application that is integrated with the heart disease prediction algorithm that is developed using the chosen best-performing machine learning model. By deploying it as a lightweight web application, the prediction algorithm will be more easily accessible and easy to manoeuvre by the general public

through a web browser that even the most basic computers or smartphones can load, which is becoming ever more accessible for communities around the world.

1.6 Project Scope

The scope of this project involves: 1) Determining the dataset to utilise and data acquisition 2) Preparing, understanding and preprocessing the dataset 3) Employing feature engineering and selection methods to select relevant features that strongly correlate with the target attribute and improve prediction accuracy 4) Developing and implementing hyperparameter tuning methods 5) Designing and implementing the ML models that are to be compared in terms of the predetermined evaluation metrics 6) Measuring and comparing the performance of the ML models in detecting the onset, presence, and level of heart disease using the testing dataset 7) Model selection to determine the best performing model based on various factors 8) Developing a web application to deploy the chosen best ML algorithm for this problem domain. Therefore, any device (mobile device, desktop) with a web browser along with an Internet connection is required to access the application and its functionalities.

Thus, the development of this project can be seen to be in two parts, which include the implementation of ML to determine the best-performing ML model for this problem domain and then the development of a web app component that will integrate the chosen ML model into an easily accessible website format which users can access using a web browser.

Hence, the bulk of the project is in the implementation of the ML component to be utilised. The first part of the ML implementation involves acquiring the data from the chosen dataset.

Then, understanding the dataset and its distribution to aid us in our implementation of the ML models. We can also determine whether the dataset has sufficient records and data to train our model, else more can be incorporated in the previous data acquisition stage. For data understanding, a description of the dataset is generated, where information such as mean, median, mode, etc. are studied to further understand the distribution and to determine the necessary preprocessing and cleaning steps to perform. Hence, the following step is to perform data preprocessing where the dataset is cleaned using the appropriate methods determined from understanding the dataset and its faults. This can ensure that no noise, outliers, or other faults in the dataset will affect the building of the ML models.

For the next stage, we perform feature engineering or selection where only relevant features are considered for building the ML models. We can determine the appropriate features to utilise from viewing the features' correlations with each other and the target data using the univariate and multivariate plots from the data understanding stage.

Subsequently, hyperparameter optimisation methods are explored and developed to further improve the predictive performance of the prospective ML algorithms.

From there, we can partition the data into training and validation datasets. For this, different split ratios will be studied to determine the performance of the model at different split ratios (e.g., 50:50, 60:40, etc.) and to find the most optimal combination.

The next part involves the development of the machine learning model algorithms as well as designing hybrid models to test whether a combination of ML methods produce better performance for this problem domain. The ML algorithms that will be considered are explained in sections 2.1.1 to 2.1.9 and section 3.4.5.

The following part involves the experimental testing and results phase, where we are evaluating and comparing the performance of the ML models that were developed in the previous part. The evaluation metrics that were considered are explained in section 3.4.6. This section is crucial to determine the best-performing ML model that is to be integrated in the final system.

After that, based on the obtained performance and evaluation results from the previous stage, model selection is performed to determine the best-performing model. Simultaneously, at this stage, there are other considerations that are maintained when choosing the ideal model, for example, based on the principle of parsimony, for multiple models with the same performance metrics, the model with the least complexity is chosen.

The final part involves the development of the web app where the ML model will be integrated and deployed in this environment where it can be easily accessible for the end-user, along with the appropriate system testing methods to ensure the final system is functional and fulfils initial requirements and expectations.

Some out-of-scope aspects of this project include, the usability and speed of the web application will not be studied or evaluated in this project due to time and resource constraints and as it is only a secondary functionality in this project. There is also a personnel limitation in terms of the number of testers to evaluate the usability of the system. This is also because the application will not be published or released for real-world usage as mentioned in the earlier limitation. Thus, usability tests, such as user acceptance testing (UAT), will not be performed.

Additionally, it is worth noting and emphasising that the final developed system and project is only for educational and research purposes to understand the application, strengths and weaknesses of ML models and the features that are significant for this problem domain of heart disease prediction/classification. So, it will certainly need clearances and approvals from the right departments before it can be utilised for real-world commercial scenarios or making final definitive verdicts, especially considering this project is concerned with the critical aspect of human health. The role of a doctor, cardiologist or healthcare professional is still prominent and necessary to make the final verdict on a patient's heart health, although we believe that our system aims to supplement their decision-making process by analysing large amounts of data to identify patterns and supplement the diagnosis. So, this project is also intended to help us better learn the differences, performance, and application of different ML models and feature selection techniques.

2. Literature Review

2.1 Machine Learning, Prediction, Classification and Data Mining

The core topic of this project is classification and prediction which utilises machine learning techniques. Machine learning (ML) is considered a subset model or a subdivision of the more general artificial intelligence (AI) network that utilises complex algorithms and deep learning neural networks [13]. ML enables a program to iteratively and automatically learn or “train” through the input of data and improves the results based on the previous experiences and knowledge (heuristics) [14]. So, this training is based on the derived knowledge from the database, which allows the model to make predictions and classifications. Plus, as mentioned earlier in the problem statement, with the blooming big data sector in the medical field, there is a larger amount of data that can be used to train the models and ameliorate the performance and accuracy of the analytical tools used for earlier detection of these CVDs.

In accordance with that, prediction and classification is an area where machine learning is employed. Classification is a method in which ML algorithms are trained to assign class labels to cases for a particular problem set. Hence, it is a predictive modelling of any problem domain with labelled classes to be predicted based on a dataset [14]. Thus, it distributes data between predefined classes according to the specified rules. Therefore, it is commonly associated with prediction tasks, as the aim is typically to forecast or classify the target value from the possible outcomes, presence of heart disease, based on given input features.

Along with that, data mining is utilised with ML, as a discovery method for sourcing and analysing big data or usable information from raw datasets and finding important information from this enormous collection of data. Data mining is considered a non-trivial extraction of implicit and potentially helpful information about data that was formerly unrevealed [15]. It couples an effective data collection or warehousing method along with data processing or the analysis of patterns in the large batches of raw data. Hence, this can be influential for exploratory and illustration of patterns to make intelligent business-related decisions. In this case, as the medical field is one which generates large amounts of complex data points daily concerning disease diagnosis and patient physiological and medical information, data mining examines large amount of data and sets a certain outcome and provides techniques to discover unseen patterns or similarities in the data distribution and draw vital conclusions to make valid inferences and improve clinical decision support.

In the field of ML, it is generally understood that no single model is more accurate to its counterparts and it depends on the problem domain that is being studied. So, ML provides various classification algorithms to compute the probability of a patient possessing HD. Apart from applying singular ML algorithms on a dataset, a hybrid model can also be developed which is a incorporation or ensemble of several classification and feature selection techniques in a unified single model, in hopes of increasing accuracy and providing better prediction performance [14]. Thus, if the proper combination of ML algorithms is assembled, the hybrid model can provide favourable evaluation results, such as enhanced accuracy. Therefore, we aim to study the different ML models that can be utilised to make accurate predictions. Through the use of ML models, analytical models can be created aiding us to process large amounts of data and identifying earlier unknown patterns and trends in the dataset and utilising the attained information.

As the dataset we are working with possesses clear known labels where it has the corresponding correct outputs, this learning is “supervised”, as opposed to “unsupervised” learning where instances are unlabeled. This form of ML is highly robust, applying for various robusts and extremely effective for certain problem domains, where the correct outputs are labelled.

The following is a general overview of each machine learning model that will be implemented or explored in our study. This is to help us understand the workings of each model and assist us in its implementation. Each model will be first trained or fitted with a portion of the dataset known as the “training set”, then validated with the “validation set”, which is separated and set aside currently as “unseen data” to evaluate the algorithm performance.

2.1.1 Simple Logistic Regression

Logistic regression is a simple parametric supervised ML classification algorithm from the study of statistics that is utilised to predict the chance for a target variable to occur and performs binary classification where values are assigned to either of two classes. Similar to linear regression, the aim is to compute the coefficient values for the input variables, however, logistic regression transforms the variables into a dichotomous target value in that it can possess one of two possible classes, either 0 for failure or 1 for success [16]. It is a linear model used for finding solutions to binary and linear classification problem domains by predicting the likelihood of a categorical-dependent variable based on a given set of determinants or independent variables where the response variable can be of a binary nature or having continuous explanatory variables.

In an ideal situation for logistic regression, the input features are unrelated to the output variable and the input features correlating to each other are eliminated

Additionally unlike linear regression, it utilises the logistic or sigmoid function for computing the results between a dichotomous response variable between 0 and 1, which can be defined as:

$$\text{logistic}(n) = \frac{1}{1 + e^{-n}}$$

Here, n denotes the linear model’s output trained with the logistic regression model to produce a value between 0 and 1. Also, as the output of the above equation is between 0 and 1, a threshold needs to be set to ensure the output is assigned to either of 2 classes, 0 or 1. Hence, for our study, we set the threshold value to 0.5 so, if the output of the logistic function is more than 0.5 the responding variable will be transformed to 1 (presence of HD) and for output less than 0.5 the responding variable will be 0 (absence of HD).

Aside from a binary target value, logistic regression can also produce a dependent variable in the form of a multinomial or ordinal value. A multinomial value is one where the responding variable can be of three or more unordered classes, such as “Type 1”, “Type 2”, and “Type 3”. As for ordinal, the responding variable can possess three or more possible ordered classes that have quantitative significance, such as “bad”, “moderate”, and “good” along with their score values such as “0”, “1” and “2”.

2.1.2 Decision Tree

A decision tree is a non-parametric supervised ML model or classifier that possesses a tree and flowchart-like graph structure with nodes containing rules for classifying instances/records to its categorical target class (0 for absence of HD or 1 for presence of HD) based on its attribute values. Hence, at each node is a variable or a feature in an instance used for classifying and subsequently the branches represent the values that the node can possess or the outcome of the test. Thus, the rule can split a tree into two or more analogous sets based on the most important indicators [13].

So, the classification begins at the root node and then sorted based on its attribute values. The attribute that best distinguishes a dataset is designated towards the root node. The internal nodes are other decision-making components of the decision tree that represent the defined characteristics of a dataset which collectively make up a decision depending on various algorithms and to traverse the following nodes. The traversal stops or the split process halts when a leaf node is reached where the predetermined criteria is met and the node does not contain any more branches or child nodes. The path spanning from the root to the leaf contains classification rules.

Looking at it more simply, a decision tree asks a question at every node, at which the tree is split into subtrees based on the response or values available (Yes, No). This data can be either categorical (e.g. Yes, No) or numerical (e.g., >50 and <50). Within the topic of decision trees, there are different types that can be employed, which include CART, C4.5, CHAID, J48, ID3 among others.

Overall, decision trees can be more accurate compared to other models as it studies a dataset in a tree-like pattern. Nevertheless, the downside is, there is a possibility of the dataset being overclassified and only one feature being evaluated at a time for decision-making.

2.1.3 Naive Bayes

The naive bayes model is a supervised ML algorithm and a simple, probabilistic, and statistical classifier based on Bayes' theorem. This model is considered "naive" as it assumes that values of a certain predictor are independent of the values of another and there exists no correlation, provided the class variable. So, changes in one determinant will not affect another, thus making it suitable for large datasets. Hence, every determinant is required to independently contribute to the probability value to maximise it [17]. This theorem is a mathematical concept that is utilised to acquire probability values and is applied for many real-world applications.

It is associated with two types of probabilities, that can be computed based on the training dataset directly:

- The probability of all classes
- The conditional probability for all classes, given each x value

The following is Bayes' theorem equation:

$$P(X|Y) = P(Y|X) * P(X)/P(Y)$$
$$P(X|Y)=P(X)*P(Y/X)/P(Y) , \text{ where } P(Y|X)=P(X \cap Y)/P(X)$$

Here, x represents the data tuple while C is the class, given that $P(X)$ is constant for every class. Based on the above formula, the Bayesian classifier computes the conditional probability of a record belonging to each class and the instance is classified as the class possessing the largest conditional probability. When the probability values are computed, the probabilistic model is applied to make predictions with the novel data utilising the Naive Bayes theorem. If the data possesses a real value, it likely assumes a Gaussian or normal distribution that mimics that of a bell-shaped curve. This allows a simple estimation of the probability values.

The benefit of this model is that even though it assumes an unrealistic condition where attribute values are conditionally independent, it functions effectively for vast datasets where the condition is assumed and applied. So, it typically has a minimal error rate, however, this may not be true in all scenarios. Inaccuracies may arise due to assumptions owing to class conditional independence and the absence of available probability data.

2.1.4 Random Forest

Random forest is a supervised ML model, and it is considered an ensemble model providing the function of ensemble learning (sometimes termed nearest neighbour predictor). Structurally, it consists of multiple regression trees constructed during the training time, thus giving its name “forest” and it trains each one with a slightly different randomly selected set of training dataset records where it splits the nodes at every tree, taking into consideration only a small number of features [18]. From the significant number of decision trees, the tree with the most upvoted class prediction or the largest mode value becomes the predictor for the model. Alternatively, the last prediction of the model is computed by averaging the predictions of every tree to find a natural balance between the two extremes, thus enhancing the overall accuracy for the particular hidden data and reducing high variance and bias [19].

The advantage of a random forest classifier is that it provides high accuracy while requiring relatively lesser training time. Additionally, it decreases the likelihood of overfitting the ML model owing to its utilisation of multiple decision trees. It performs well for overcoming missing values, however, can be slow for obtaining predictions as it necessitates large data sets and a significant number of trees, and the results are unaccountable.

2.1.5 Support-Vector Machine

SVM is a supervised ML model that classifies records through finding an optimal hyperplane distinguishing the classes on multidimensional spaces. This hyperplane is iteratively created by the SVM model to reduce error. So, the data points are plotted in the n -dimensional space with each feature’s values being the coordinate value and the classification is conducted based on the hyperplane that distinguishes the two data classes [20]. Subsequently, the features of new instances are used to predict the class in which subsequent instances should be classified.

Overall, the aim of SVM is to separate the dataset or input points into classes and search for the maximum marginal hyperplane (MMH) or the distance between the hyperplane and the two nearest adjacent data points from the particular classes. The line that has the largest marginal difference to distinguish the two classes is considered the optimal hyperplane as it can distinguish the two classes better. This ensures to minimise the risk of misclassification. The

points that are used are called support vectors, thus giving the name support vector machine, as they support or define the hyperplane. For the implementation, an optimisation algorithm is applied to compute the parameter values that maximise this margin.

There are different forms of SVM, where it constructs the knowledge-based model with different kernels, either utilising a linear kernel, radial basis function (RBF) kernel, sigmoid and polynomial kernel. Thus, this allows for hyperplanes to be constructed for non-linear data points as well.

Different tools can be used to implement SVM, such as the scikit-learn library, MATLAB and LIBSVM.

2.1.6 K-Nearest Neighbour (KNN)

The K-Nearest Neighbour (KNN) model is a simple non-parametric algorithm that does not have assumptions about the underlying data. It is based on the principle of Euclidean distance where the records in a dataset are in close proximity to each other and possess homogenous properties. Hence, if the records are assigned a classification label, the unclassified record's label value can be obtained by observing the closest adjacent node/neighbours class. It stores available cases and classifies new records based on the closest majority neighbours, for instance, based on the euclidean distance function. So, it assumes that identical classes exist in close proximity to each other as these entities are similar and must exist together [21]. In the name KNN, the K lettering represents neighbourhood cardinality or the number of nearest neighbours close to a new datapoint for assigning the class to the new point. So, for when K is equivalent to 1, 2, or 3, it will either choose one, two, or three of the closest neighbouring data points to determine the class to assign the record.

It has assumptions such as the data possessing noise, is labelled, and should include relevant attributes. Nonetheless, before implementing KNN on a dataset, the data should be preprocessed to clean noise and outliers as well as to normalise the variables to prevent high value variables causing the model to be biased. Thus, it can be seen as a grouping method that considers the distance between a data point and the coordinates, and its neighbours [21]. The distance is calculated using the following euclidean distance function and the neighbours are determined from the datapoint and located in the area that is closest to the neighbouring points:

$$p(x, x') = |x - x'| = \sqrt{\sum_{k=0}^n (x_i - x'_i)^2}$$

Here x_i and x'_i represent two vectors from an initial node, whereas n represents the size for the n -dimensional space.

The downside to KNN is that it can take a noticeable amount of time to be trained with large datasets and a dataset with noise or irrelevant attributes can influence its accuracy and cause bias. It is considered a “lazy” learner in that it is an instance-based learner that depends on the distance that does not learn any classification rule. Nonetheless, it is simple to understand as it has a non-complex structure and is versatile enough to be applied to various practical problem

domains, including for classification, regression and search problems. It has been applied in statistical estimation and pattern recognition applications.

2.1.7 Artificial Neural Network (ANN)

Artificial neural networks are also known as multilayer perceptrons, and they are considered biologically inspired in that they aim to replicate a human learning and decision-making mechanism, and it is able to model extremely complex non-linear functions and statistical data. Data that is inputted into these networks affects the structure of the ANN as a neural network constantly adapts to changes in the data and learns in a sense-based manner based on the input and output, for the current and subsequent stages [13]. Therefore, ANNs are effective tools for machine learning and discovering complex and ambiguous patterns to train itself to recognise them. Structurally, ANNs have interconnected layers and the networks themselves are fairly simply mathematical models that can improve existing data analysis technologies.

It is the first form of neural network (NN) models that will be studied and implemented. Generally, neural networks are a series of algorithms that aims to discover or match underlying hidden relations in a dataset through the process of replicating the thought pattern of the human brain [22]. Hence, as the name “neural” suggests, these are brain-oriented systems that are intended to replicate the way in which humans learn and make decisions.

Neural networks typically comprise three layers, the input, output and hidden layer and consist of interconnected nodes within the layers. Most of the time, this hidden layer consists of units that are capable of transforming the input data into a pattern in which the output layer can alter. In neural networks there are neurons which are essentially mathematical functions that accumulate and classify information based on the specific architecture. NNs can be of single or multiple layers where single layer NNs possess nodes which are perceptrons, similar to a multiple linear regression model [22]. These perceptrons relay the signals from the multiple linear regression to an activation function that is likely non-linear. In a multi-layered perceptron, these perceptrons are ordered in interconnected layers, where the input layer collects the input patterns and the output layer possesses classifications or output signals to which input patterns may map. NNs are akin to statistical methods such as curve fitting and regression analysis [23]. They have especially become a crucial aspect of AI owing to the inception of the new methodology, known as backpropagation where networks can adjust their hidden layers of neurons in scenarios where the output does not correspond with the developer’s expectation.

2.1.8 Recurrent Neural Network (RNN)

Recurrent neural networks (RNN) are a robust and powerful form of ANN that utilises sequential and time-series data and is considered a type of deep learning algorithm. Thus, it is commonly utilised for ordinal or temporal problem domains, namely for language translation, natural language processing (NLP), etc. In RNNs the connections between the nodes form a directed or undirected graph along a temporal sequence, thus displaying temporal dynamic behaviour. So, these sequential data are ordered data where related nodes follow one another [24]. Hence, it is akin to a feed-forward NN, where information moves straight through a network without traversing a node twice.

As with all NNs, it requires a training dataset to learn and improve its model. However, the way it differs from ANN or any other form of NN is that it possesses internal memory based on prior input data that affects the current input and output. Traditional deep NN assumes that these input/output values are independent of each other, however, for RNN the output is dependent on the initial elements in a sequence [25].

A noteworthy issue of RNNs is with its gradients. A gradient in this case is a partial derivative in regards to its inputs which measures the change in the output of a function when a change is applied to the input [24]. So, envisioning the gradient as a slope of a function, at high gradients the slope is steeper and this represents a faster time in which the model learns and conversely, if the slope is zero, the model's learning halts. This gradient value measures the change in every weight point when there is a change in the error rate. So, in the exploding gradient problem, the model assigns an unjust high importance to the weight value. Whereas for the vanishing gradient problem, the gradient values are too low, where the model learning halts or requires an extensive computation time [24].

The benefits of employing an RNN model include its ability to process sequence data, input of varying length and that it stores and utilises prior information in its internal memory to determine a proper output. Nonetheless, its disadvantages include the computation time being slow, the network not being able to consider future inputs in decision-making as well as the risk of the vanishing and exploding gradient problem.

Therefore, after careful consideration and reevaluating the model, its purpose, structure and operation, it was found that it is not suitable to be studied for this problem domain of supervised classification, as it is more suited towards applications which perform reinforcement learning or unsupervised learning where the target attribute is absent. Thus, this model was eventually left out of the ML implementation due to these reasons.

2.1.9 K-Means Clustering

The k-means clustering model is an unsupervised ML model that groups or aggregates unlabeled data points into groups of varying clusters because of certain similarities and there is not a target variable and the goal is to highlight any patterns in the data, making them more evident [26]. The K lettering represents the number of clusters that are to be formed through the process, so if K is defined to be 5 there will be five groups or clusters. Besides designating the number of clusters, this model can automatically learn or find a suitable number of clusters on its own without the information being declared or inputted [14]. So, this makes the model semi-supervised.

Therefore, the model identifies the number of clusters then allocates the data to the nearest cluster/centroid, while aiming to keep the clusters as minute as possible. Then the model averages the data to find the centroid, as stated in its name, "means". Iteratively, the model calculates this optimal position of the centroid, until the centroids are stabilised or the values remain constant after clustering is successful or a predefined number of iterations is performed [26].

The benefit of this model is that it is effective for datasets with many records. However, its performance is not as competitive compared to other models or clustering techniques since variations in the input data can cause high variance in the output.

2.2 Cardiovascular Diseases (CVDs) and Heart Diseases (HD)

The heart is among the most vital organs in the entire human anatomy, serving the crucial function as a central part of the circulatory and cardiovascular system of rhythmically pumping blood all over the body to transport oxygen, nutrients, and hormones to body cells and for eliminating waste products [27]. Thus, it is clear to see that when it fails to pump blood correctly and effectively, this can lead to the human body encountering fatal conditions.

Therefore, cardiovascular or heart diseases can be classified as any disorder or abnormalities of the heart. So, heart disease is an umbrella term consisting of a range of conditions that can affect the human heart whether it is concerned with vessel diseases (coronary artery disease, and arrhythmias) and heart defects one is born with with congenital heart defects, as well as others. [27]. The term heart disease is also used interchangeably with cardiovascular disease since the heart is the central component of the cardiovascular system. It describes any pathologies that change or negatively impact the function or structure of the circulatory system [28]. Nevertheless, CVD is more concerned with blood vessel conditions where they are blocked or narrowed leading to heart attacks, strokes or angina (chest pain) [27]. Generally, some common determinants for heart attacks that are used as measurable indicators/metrics include blood pressure, cholesterol and pulse rate [29]. Heart attacks are the main form of heart disease where it occurs when one or more of the coronary arteries become blocked.

Hence, the following are some of the reasons for heart disease to develop and occur:

- All or part of the heart is experiencing damage
- Blood vessels from and to the heart are damaged or affected
- Hardening of the arteries (atherosclerosis)
- Weakening of the heart muscles (cardiomyopathy)
- Heart defects present from birth (congenital heart defects)
- There is insufficient supply of oxygen and nutrients reaching the heart
- Bacterial, viral or parasitic infections of the heart.
- Damage to the heart valves (rheumatic heart disease)
- Rhythmic problems with the heart (arrhythmia)

Factors of Heart Disease

The underlying factors may vary depending on the disease. Nonetheless, it is estimated that dietary risk factors are associated with 53% of CVD deaths [3]. Other strong physiological factors and habitual predictors include high blood pressure, smoking, diabetes mellitus, lack of exercise, obesity, hypertension, stress, high blood cholesterol, pre-existing heart problems, poor diet, excessive alcohol and caffeine consumption, and poor sleep, among other things [30]. High blood pressure is estimated to account for approximately 13% of CVD deaths, while tobacco accounts for 9%, diabetes 6%, lack of exercise 6%, and obesity 5% [1]. Genetics can also play a role in one's likelihood of developing heart problems [28].

Preventative Measures

However, it is estimated that up to 90% of CVD may be preventable if the established risk factors are avoided [3]. So typically, cardiovascular disease is treatable with initial treatment primarily focused on diet and lifestyle interventions and involves improving risk factors through

healthy eating, exercise, avoidance of tobacco smoke and limiting alcohol intake. Treating risk factors, such as high blood pressure, blood lipids and diabetes is also beneficial. There are also surgical or procedural interventions that can save someone's life or prolong it. For heart valve problems, a person could have surgery to replace the valve [28]. For arrhythmias, a pacemaker can be put in place to help reduce abnormal heart rhythms and for a heart attack, there are multiple options two of these are a coronary angioplasty and a coronary artery bypass surgery [27]. Medication is also provided to treat, cure and control a patient's heart disease, for example, this includes the prescription of Anticoagulants, Angiotensin-converting enzyme inhibitors, Angiotensin II receptor blockers, and Beta-blockers, amongst others [30]. A cardiologist will work with the individual to find a suitable option.

On that note, this poses a crucial need to monitor high-risk patients' vitals. For this, more sophisticated technology, especially sensory technology can be employed along with machine learning models to frequently and efficiently collect and process patient's medical information, to make more well-informed and quicker decisions to ensure one's heart health is maintained or for cardiac healing and rehabilitation purposes.

For this purpose, it is found that the application of ML and AI for heart disease classification and prediction has been consistently more accurate and displaying better performances than with manual classification [7]. There is additional evidence to suggest that simply providing people with a cardiovascular disease risk score may reduce cardiovascular disease risk factors by a small amount compared to usual care [9]. Errors in the clinical results otherwise can lead to misdiagnosis that can lead to fatalities. Thus, computer-based support systems can support the decision-making process of healthcare professions to achieve the correct and cost-effective measures.

Overall it is clear that the efficient, accurate and early discovery and medical diagnosis of the heart problem earlier on, can allow for medical intervention to take place and preventive procedures to be conducted, to avoid complications, increase the patient's likelihood of recovering from the CVDs and reduce the fatality rate.

2.3 Electrocardiography (ECG)

This section describes some key information about electrocardiography and the output data that it produces in that it can be useful to our study as it is a commonly used determinant or measurable indicator for determining a patient's likelihood of developing heart problems. It can be useful to our study and implementation of the heart disease prediction system, for understanding a key determinant that can possibly be utilised as an input feature in our model and prospective system:

By definition, electrocardiography (ECG) is the process in which an electrocardiogram (ECG or EKG) is created, which represents the electrical activity of the human heart in an amplified recording format [31]. This recording of the heart's electrical activity is obtained through placing electrodes on the patient's skin, and is subsequently plotted in a voltage-time graph. The purpose of these electrodes is to perceive minute electric changes as a result of the depolarisation and polarisation of the heart muscles at every cardiac cycle or heartbeat [32]. Hence, alterations to the normal ECG pattern can be pathological and of clinical significance, indicating that the

patient might be experiencing any cardiac abnormalities, rhythm disturbances, insufficient coronary artery blood flow, or electrolyte disturbances.

These electrodes detect the small electrical changes that are a consequence of cardiac muscle depolarization followed by repolarization during each cardiac cycle (heartbeat). Changes in the normal ECG pattern occur in numerous cardiac abnormalities, including cardiac rhythm disturbances (e.g., atrial fibrillation), inadequate coronary artery blood flow (e.g., myocardial ischemia and myocardial infarction), and electrolyte disturbances (e.g., hypokalemia and hyperkalemia) [31]. Traditionally, a 12-lead ECG is performed on patients, by attaching ten electrodes to their limbs as well as on the surface of the chest. The overall magnitude and the direction of the heart's electrical depolarisation is recorded for every heartbeat [33]. Nevertheless, in the modern day, this technology is becoming even more accessible as its deployed in more consumer sensory gadgets such as certain smart watch models (e.g., Apple Watch Series 4, 5, 6, 7).

From viewing the ECG we can see a few of the main components in each heartbeat wave or cardiac cycle and the following describe what each segment represents:

- P wave: The depolarisation of the atria (upper two compartments of the heart)
- QRS complex: The depolarisation of the ventricles (bottom two compartments of the heart)
- T wave: The repolarisation of the ventricles
- U wave: The repolarisation of the papillary muscles, generally not observed and ignored

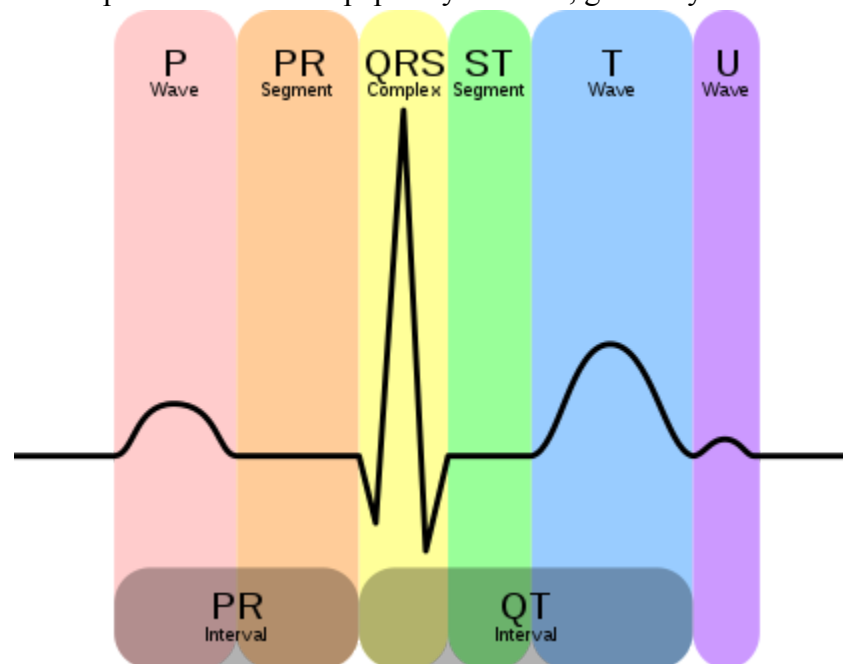


Figure 1: Main components of a cardiac cycle recorded in an ECG [33]

Overall, to a heart specialist or a trained ECG clinician, these raw data from the ECG can be analysed to relay vast information about a patient's heart electrical activity [32]. Hence, this analysis is essentially a form of pattern matching or recognition where one with the understanding of theory heart functionality can recognise what the ECG data is displaying and

from there they can make valid inferences. Thus, these data can be analysed to measure the heartbeat rate, rhythm, size and positioning of the heart compartments/chambers, the existence of disorders or damage in the cardiac muscles or conduction system, the effect of heart drugs as well as the functioning of pacemakers and sinoatrial nodes. For instance, irregularities in the QRS complex without presence of P waves indicate the possibility of atrial fibrillation [31]. Furthermore it is crucial to consider the context of the patient and not rely solely on directly and independently analysing the ECG data to ensure applicability and a correct diagnosis [33]. Otherwise it is possible for an ECG to falsely imply the presence of an issue that can lead to misdiagnosis, recommendation of invasive measures and even overtreatment that can also be detrimental and fatal. Thus, it is still crucial for the context of the patient to be considered and for the ECG data to be correctly read as well as minimising the risk of human error that can cause misdiagnosis.

Generally, an ECG is only performed after a is suspected to be experiencing HD or CVD, so this is reactive in nature. It may also be in conjunction with a patient's routine clinical check up or physical examination, typically applicable for middle-aged and older patients. This allows to obtain a control ECG report which can be used as the basis of comparison for when later ECG reports are conducted to test for HD [31]. It can be performed continuously or to produce short intermittent tracings. The former is typically performed for critical patients, ones that are given anaesthetics, or ones who experience infrequent cardiac arrhythmia.

There are three forms of ECG that can be recorded at different conditions:

- Resting ECG: recorded while the patient is resting, lying down or in a comfortable situation
- Stress or Exercise ECG: recorded while the patient is exerting pressure or exercising
- Ambulatory ECG: Monitors and records the patient at home for one or more days using electrodes attached to a mini portable machine

That being said, with the incorporation of intelligent support systems that utilise ML and sensory technology becoming more accessible, these comparisons and tests can be performed more easily and cost-effectively so more members of society can have a clean bill of health. With its deployment in consumer gadgets, they can even perform these tests while engaging in regular daily activities, with the use of continuous ambulatory ECG that checks for short or sudden abnormalities in the heart rhythms and insufficient blood flow to the cardiac muscles [32]. Apart from ECG recordings, anatomical heart imaging modality is also utilised for measuring the mechanical functioning of patients' hearts.

2.4 Existing Systems and Literature Analysis

The following section provides a summary and comparison of 8 different literature sources or journal articles that are related to this topic of heart disease prediction and classification using ML algorithms. By analysing these similar projects and literature, can help us identify the characteristics, methods/ML models employed, and gaps present in current heart disease prediction and classification systems to aid in generating requirements and designing our proposed system.

2.4.1 Heart Disease Detection by Using Machine Learning Algorithms and a Real-Time Cardiovascular Health Monitoring System [9]

This paper by Nashif et al. [9] presents their proposed design of a cloud-based heart disease prediction system that detects the onset of heart disease using ML techniques built using the java-based open access data mining platform, WEKA. Additionally, to provide continuous monitoring of the patient's condition in accordance with the concept of Internet of Things (IoT), the author proposes a real-time patient monitoring system using Arduino that uses physiological sensor technology to monitor real-time parameters, including body temperature, blood pressure, humidity, and heartbeat. The system also allows for the prescribed doctors to be notified in the event the real-time parameter exceeds the threshold through GSM technology.

They start by selecting and preparing different heart disease dataset to train the ML algorithms. They opted for a combination of the Cleveland Heart Disease dataset consisting of 303 records and the Statlog Heart Disease dataset with 270 records, both with 13 features. This merging of datasets is believed to make the ML model more robust.

Moving on, the models that the authors proposed to use for this problem domain include Naive Bayes, Artificial Neural Networks, Support Vector Machine, Random Forest, and Simple Logistic Regression. The evaluation metrics they prioritised were accuracy, precision, recall, fscore, sensitivity, and specificity.

From their comparative study, they found that SVM, Random Forest and Simple Logistic models showed higher accuracy rates of more than 95 percent which make them considerable models for biomedical applications of disease detection and prediction. For the other evaluation metrics, they found that the SVM model performed better than the other models. Therefore, in their study, they found it is decisive that SVM is the most efficient algorithm to be implemented on the heart disease prediction system as found in our study when the 13 features were considered. From their experimental study and literature review, no algorithm attained an accuracy level of more than 90 percent in heart disease prediction using the same number and types features as used in this study.

Conclusively, through reviewing this study, a better understanding of the possible classification algorithms and the evaluation metrics that can be used to compare them for this problem domain. Thus, as it is the first literature source reviewed, it provides a good basis for understanding the problem domain and the necessary steps to be taken to find a solution, from dataset preparation methods, appropriate classifiers, and evaluation metrics.

2.4.2 Developing a Hyperparameter Tuning Based Machine Learning Approach of Heart Disease Prediction [35]

This paper by Hashi and Zaman [35], is aimed at presenting their proposed system for heart disease prediction using ML techniques along with comparing its performance with traditional systems. For this, they implemented Logistic regression, K-nearest neighbour (KNN), Support vector machine (SVM), Decision tree (DT), and Random Forest (RF) classification models to study the best performing model for this problem domain. Along with that, another specialised aspect of their study was they analysed the effects of tuning the hyperparameters using a grid

search approach for the five aforementioned classification models to further enhance the performance of the prediction models.

For their study, they utilised the Cleveland Heart Disease dataset from the University of California, Irvine (UCI) machine learning repository for both training and testing their ML models. Although the dataset contains 75 attributes, they focused on using 14 numerical valued attributes. Then, for preprocessing the dataset, they identified and cleaned missing values, processed noise, incomplete or inconsistent values and removed redundancies with the attributes. Afterwards, the preprocessing involved separation, feature scaling and normalisation to find the standard format of data. During this step they generated a description of the dataset to understand its distribution by generating the maximum, minimum, mean and standard deviation values of each feature set. With the preprocessing completed, they partitioned the dataset into the 80:20 ratios, 80% for training and 20% for testing.

Then, they developed and generated their ML classification models with the five algorithms mentioned earlier. They performed this twice, the first one including the five algorithms without hyperparameter tuning and the latter using this technique where grid search and cross validation are employed to optimise the hyperparameters. Essentially, grid search entails an exhaustive search technique which computes the optimal hyperparameter values. It builds a model that generates all the parameter combinations and stores combinations.

The performances of the conventional and proposed methods were evaluated and compared, based on their chosen evaluation metrics, consisting of accuracy, precision, recall, and F1 score using the True Positive, False Negative, False Positive, and True Negative values. They found that the traditional method produces accuracies between the 81.97% and 90.16% range whereas the proposed approach coupled with hyperparameter tuning produced improved accuracies and performance between 82.25% and 91.80%. Thus, showing that their proposed approach is capable of improving what is currently available through the acquisition of feasible performance.

Conclusively, their study proves and illustrates the importance of using or incorporating hyperparameter tuning in the ML model in improving predictive performance of heart disease classification. Besides that, their study showed the strength of the KNN model for this problem domain, as their results displayed that this model possessed the best performance, especially in terms of accuracy (91.8%), followed by SVM, logistic regression, decision tree, and random forest.

2.4.3 Classification and Feature Selection Approaches by Machine Learning Techniques: Heart Disease Prediction [36]

This paper by Reddy et al. [36] aims to present their study to predict the classification model and identify the selected features that are influential to the classification and prediction of HDs. For this study, the authors used five heart disease datasets (Cleveland, Switzerland, Hungarian, V.A. Medical and Statlog project heart disease, all of which accumulated from the University of California Irvine (UCI) machine learning repository website and were then combined into one. They also performed data preprocessing as some records had missing values which were replaced with the modal or most frequently occurring value. Besides that, another preprocessing technique that they employed was normalisation where they eliminated high values of respective

attributes or outliers in the data distribution. Then the data was partitioned with three different ratios, 60–40%, 70–30% and 80–20%.

For the development they utilised the R programming language along with the CARET package for data preprocessing, splitting, and the ML models. The ML models they opted to explore and experiment include K-Nearest Neighbour, Support Vector Machine, Random Forest, Naïve Bayes and Neural Network. Along with that, they studied the effects of using different feature selection methods on the performance or accuracy of the proposed system in classifying or predicting heart disease. For this, the feature selection methods that were experimented include, correlation matrix, recursive feature elimination with random forest algorithm, variable importance estimations, rank feature by importance with learning vector quantization model. The feature selection method is conducted on the merged dataset to choose significant or relevant features for the model construction in order to enhance the prediction accuracy of the target attribute. For this, this project conducted a correlation study to identify the highly correlated attributes. Based on the order of the features, the common 8 and 6, in total 14 selected features are taken into consideration to build a model.

The evaluation metrics to study the performance of a model on the test dataset is calculated by accuracy, sensitivity/recall, and specificity using R. For their study, the sensitivity and specificity metrics are especially significant as it evaluates the true positives (risk class) and the true negatives (normal class) respectively. Hence, reflecting the predictive capabilities of the ML models or classifiers.

Based on their results, they found that the random forest classifier with the use of 8 and 6 selected features displayed the highest accuracy using a 70:30 or 80:40 split ratio. The two split ratios show an insignificant increase or change in average accuracy. Overall, their study shows that the performed random forest algorithm exhibits the best performance with the existing model accuracies. The accuracy of using 8 selected features is more substantial than using the combined dataset, indicating the significance of the feature selection method employed. The use of 6 selected features and less displayed decrease in average accuracies compared to using the combined dataset, hence, this could be due to underfitting of the attributes on the ML models and the performance of the classifier under default conditions. This number acts as the minimum number required to build a strong model.

All in all, this study was a clear example of selecting the correct minimum and prominent attributes that improves the performance when compared to using whole features from the dataset. It also showed that the random forest algorithm produced the most accurate predictions and is a good classification model for this problem domain. It also illustrated that the incorporation of non-modifiable risk factors (genetic risk factors) and modifiable risk factors, including smoking, physical exercise and alcohol consumption improves the predictive performance of the models.

2.4.4 Machine Learning-Based Classification Algorithms for the Prediction of Coronary Heart Diseases [37]

This paper by Kwakye and Dadzie [37] presents their comparative study and implementation of different classification algorithms of coronary heart disease datasets with ML models that they

developed and evaluated. For their development, they utilised the Framingham dataset which was collected from Framingham, Massachusetts residents to train and test their designed model. Their primary goal for the designed algorithm is to classify whether a particular patient will develop coronary heart disease in the following decade. They used all 4000 records and 16 attributes of the dataset, believing that all the characteristics are potential risk factors. They split the characteristics into 3 main categories, namely, demographic, behavioural, and medical risk factors.

Then, for the next stage, they preprocessed their dataset, by examining for potential missing values and the range of the features to avoid outliers. Here, categorical features were programmed, missing values and outliers were pinpointed and removed/replaced from the data. Next, they performed feature engineering of the data that is to be used for the model development and training. Feature engineering involves data transformation where they are selecting relevant features, attributes and comprehensively tuning the data points.

In the next stage, they developed their classification models based on ML algorithms, which include k-nearest neighbour (KNN), support vector machines (SVM), decision tree (CART), logistic regression (LR), naive bayes (NB), and random forest (RF). For their experimental testing, they utilised the cross-validation test based on ROC-AUC for unbalanced data algorithm models which revealed that there were low mean accuracy scores for the SVM model. They also performed a prediction test based on ROC-AUC for both balanced and unbalanced-data algorithm models.

Based on their study they found that having an unbalanced target feature can cause the performance of the various algorithms to be low. Therefore a synthetic minority oversampling technique (SMOTE) is incorporated to transform the initial data and balance the classes. After this, the best performing model or classifier was found to be the random forest model, an ensemble model, that displays an accuracy of 0.946337. Based on their study, they found that using hyperparameter optimisation, in this case the grid search method, did not change the performance of the model more than when default settings were utilised.

Conclusively, their study shows the importance of a well prepared, cleaned, preprocessed, and standardised dataset in building the ML models that exhibit optimal predictive performance.

2.4.5 Prediction of Heart Disease by Classifying With Feature Selection and Machine Learning Methods [7]

This study by Gazeloğlu [7] presents their comparative study of several different ML algorithms and various feature selection methods that they developed using WEKA, Python and MATLAB as well as their performances in classifying cardiovascular disease among patients in a dataset. Some of their requirements include the system possessing low execution time and possessing high accuracy in order to make correct decisions and diagnoses and minimising the risk of human error. Based on their literature review, they found that the limitations that exist with other current systems include their lack of incorporating different feature selection methods to improve the prediction accuracy as well as the current computer-aided systems are too slow in its execution. Additionally, they find that the traditional process of CVD screening can be slow-moving, especially considering the final decision is made by the heart specialists and there

is also the risk of human errors taking place in the decision making as the results can be affected by conditions such as the practitioner's education, working conditions, number of patients, or high rate of misclassification or misdiagnosis.

They declared four hypotheses that they aim to study through their experiments so that they can determine whether at the end of the study these hypotheses will still remain true and valid. These hypothesis generally include:

1. Different ML algorithms result in different accuracy rates and performances.
2. The number of parameters used affects the model's accuracy rate.
3. Different feature selection methods used result in different accuracy rates and performances.
4. We cannot evaluate the performance of an ML model solely on the percentage of success.

The author then prepares their dataset which they source from the Heart Disease Dataset from the UCI machine learning repository containing data of 303 patients and with 14 variables. According to the journal article, they did not perform any further preprocessing measures to the dataset.

In the next stage, they reviewed 18 ML techniques and used them to develop classification systems and applied them to the dataset. These ML algorithms include decision tree (J48), ADTree, KNN, roughSet, logistic regression, random forest, NBTree, RBFNetwork, fuzzy rough NN, fuzzy NN, NN, multilayer perceptron (MLP), naïve bayes, SVM (polykernel, normalised polykernel, puk and RBF kernel), and genetic programming. Along with that, they tested three types of feature selection algorithms, which include correlation-based feature selection (CFS), fuzzy rough set and chi-square algorithms. Hence, they tested every pair combination of one of each of the 18 ML algorithms with one of each of three feature selection methods.

The evaluation metrics they used to test their models include the classification rate, the area under curve calculated by receiver operating characteristic (AUC-ROC) analysis, the sensitivity (true positive, TP, and false positive, FP, rates) and the kappa coefficient. The AUC-ROC metric is utilised for comparing different testing techniques where the highest values are ideal. In this respect, the best performing model was the naive bayes without feature selection model. The TP rate represents classification of a true condition as true, and for this, the best model was found to be the SVM (polykernel) model with CFS or no feature selection. The FP rate is the classification of a sick patient with "0 or presence of HD" or also known as the alpha error in statistical sciences. So, for this metric a lower value is more ideal. In that sense, it is found that the ADTree with CFS feature selection performs the best. The kappa coefficient is a robust statistic that measures the inter-rater agreement with categorical elements and having a value of between -1 and +1 where +1 indicates a perfect agreement whereas -1 represents a perfect disagreement. In this respect, the best performing model is the naive bayes classification with CFS feature selection method.

Overall, it is found that with CFS feature selection, the naive bayes algorithm is the best performing with an accuracy rate of 85%. Additionally, the author also found that all their earlier specified hypotheses were found to be true. Also, through this study, they found that their model

can produce a high accuracy rate despite using less variables, thus being more time and cost effective. The limitation of their study includes they could not take the patient's race into consideration if that could be a possible feature correlating with the target value as that feature was absent from their dataset. For future works, they plan to implement the ML model in the form of a mobile application which can allow users to input their parameter values.

2.4.6 Heart Disease Diagnosis Based On Deep Learning Network [38]

This paper by Alhussainy and Jasim [38] presents their implementation of deep neural networks to be used as classifiers for developing a heart disease prediction system. Thus, their focus is studying the theory behind deep learning neural networks and its applicability to this problem domain. Thus, in their literature review, they reviewed other papers that were concerned with heart disease prediction using NN techniques such as recurrent neural networks, deep belief networks, and artificial neural networks optimised by particle swarm optimisation (PSO) and ant colony optimisation (ASO).

They began by collecting their dataset from the Heart Disease Dataset from the UCI machine learning repository. Then, they preprocessed it by replacing missing values in the dataset and treating continuous values and making them discrete. After that the data is partitioned or isolated into a training and testing set. The next step involves feature extraction to increase the record attributes by adding more important details or exploratory information such as mean, median, standard deviation, sum, average, min, max, etc. After that data augmentation is performed, where the derived features from the previous step are added to the original data set.

The deep learning neural network model consists of 5 layers and a different number of neurons. The first layer is the dense layer, representing the dot product of input with weight with addition to bias, then the dropout layer, used to reduce the feature size, the rectifier layer, used as activation function reducing the training time without influencing the performance, the sigmoid layer, which is used as an activation function for the later stage to sum the output to represent the probability of the class for every output, and finally the classification output layer, which represents the number of the required classification.

The model performance was evaluated using three metrics which include accuracy, specificity and sensitivity. Their proposed model was compared with several other ML algorithms that were applied on the same dataset, namely, decision tree, naive bayes, KNN (K=7), and SVM. From their results, it is found that their proposed system surpassed and improved on these existing models where it achieved an accuracy of 84.67%, sensitivity of 80% and specificity 90.72%. Overall, this paper shed more light on deep neural networks in terms of its implementation and structure as well as the optimal parameter values that the author found when using this model and the applicability of this form of algorithm (deep learning) for this problem domain.

2.4.7 A Deep Learning Method for Prediction of Cardiovascular Disease Using Convolutional Neural Network [39]

In this paper Sajja and Kalluri [39] explain their proposal of a deep learning approach for prediction of CVDs from an earlier stage. So, in this paper they aim to compare and contrast the performance of their proposed method that uses convolution neural network (CNN) with traditional approaches, namely, logistic regression, k-nearest neighbours (KNN), naïve bayes

(NB), support vector machine (SVM), and neural networks (NN). Thus, this is the second paper that we have reviewed which has the primary focus of using NN in hopes of improving the prediction accuracy for this problem domain. They performed literature review as well as performed background research on the aforementioned traditional ML algorithms before explaining the implementation of their proposed method.

They started by preparing the dataset which is the open source Cleveland Heart Disease Dataset sourced from the UCI ML repository. It contains 14 features along with 303 instances. Then, they preprocessed or cleaned the dataset to remove missing values, using the pandas package. They visualised the data distribution using counter plot and histogram. The dataset was partitioned into 80% training set and 20% testing set. This same dataset was used for their proposed CNN model as well as the traditional models that they are comparing with.

For their implementation of a deep learning neural network algorithm, they proposed a convolutional architecture. Structurally, it first contains the input layer, subsequently the convolutional layer containing 16 kernels together with activation function, ReLU, and in the subsequent layer 25% of the nodes are removed by the dropout layer. After that, the convolutional layer was conducted again but with eight kernels reusing the previous parameters and also utilised the dropout layer with 25%. Finally, there is a softmax layer.

The evaluation metric that they utilised was accuracy and the results were also plotted on ROC curves. It is found that the proposed method surpassed the traditional methods by achieving the high accuracy values, which were 95.04% for the training accuracy and 94.78% for the testing accuracy. Hence, the authors find that their model supports the medicos for prediction and another advantage it possesses over the traditional methods is that since it employs deep learning it can be effective for large volume of data and the preprocessing, feature extraction and prediction are handled by the model itself whereas the traditional algorithms would require different methods for each of those tasks. Overall, this literature explains the key structural elements of a deep learning model to consider when developing our own model as well as its applicability for this problem domain of heart disease prediction.

2.4.8 Cardiovascular disease risk prediction using automated machine learning: A prospective study of 423,604 UK Biobank participants [40]

This paper by Alaa et al. [40] presents their study regarding cardiovascular disease and the clinical application of machine learning models to predict the onset of such conditions. Along with that, they explain the development of their automated approach (AutoPrognosis model) to assess the risk value of CVD in patients.

For their project, they utilised and analysed a dataset of 423,604 participants who do not have CVDs at baseline from UK Biobank. For this, they extracted the records of participants who were 40 years of age and older and had no known history of CVD at baseline from the UK Biobank and this formed the 423,604 participants. Then, they cleaned and preprocessed the data as it contained missing values for certain attributes, and their method to do so was by excluding those attributes, so eventually a total of 473 attributes were considered during analysis. Additionally, they performed data imputation on their models using the MissForest algorithm. They grouped the attributes into 9 categories, namely, health and medical history, lifestyle and

environment, blood assays, physical activity, family history, physical measures, psychosocial factors, dietary and nutritional information, and sociodemographics.

For their analysis, they compared their proposed approach (AutoPrognosis model) with some existing models, which are the Framingham Risk Score, Cox Proportional Hazard Model, and 5 benchmark ML models, including linear support vector machines (SVM), random forest, neural networks, AdaBoost and gradient boosting machines. The benchmark ML models were implemented using the Scikit-learn library in the Python programming language. The model's hyperparameters were found using the grid search method.

For the implementation, their ML-based model for the CVD risk prediction, they utilised the AutoPrognosis which is an algorithmic framework that automates the design of ML-based clinical prognostic models. Hence, when provided the participants' attribute values and CVD outcomes (supervised model), this trains the model using 200 iterations of an advanced bayesian optimisation method, that automatically designs a prognostic model from the weighted ensemble of ML pipelines and then explores these new pipelines at every iteration to evaluate its performance by using cross-validation. Each of these pipelines consists of design options for performing data imputation, feature processing, classification and calibration algorithms of hyperparameters. Overall, the design consists of 5460 possible ML pipelines, with 7 imputation, 9 feature processing, 20 classification and 3 hyperparameter calibration algorithms, the most accurate being the model using the MissForest data imputation technique, no feature processing, and an XGBoost ensemble classifier containing 200 estimators and using sigmoid regression for the calibration. As part of the implementation, the author ranked the contribution of the different variables in the prediction model to find the relative importance of the 473 variables. This variable ranking is attained by fitting a random forest algorithm with the participant variables as inputs and the model predictions as the outputs. Then the variable importance scores are labelled at the different variables using a standard permutation method which they obtained from their literature review, which assesses the mean decrease in the classification accuracy after permuting each variable over every tree. So, the resulting scores reflect how the variables affect the prediction by the AutoPrognosis model.

For the evaluation, the metric they used was the area under the receiver operating characteristic curve (AUC-ROC) obtained through cross-validation, in an effort to prevent overfitting of the model. Based on their results, they found that the benchmark algorithms except for SVM performed better or are statistically more significant than the baseline Framingham score. Compared to the Cox PH model, neural networks, AdaBoost, gradient boosting, and AutoPrognosis performed better in terms of AUC-ROC score. Furthermore, the proposed AutoPrognosis model (AUC-ROC: 0.752, 95% CI: 0.747-0.757, $p < 0.001$) is statistically significant and has better performance over the Frammingham score as well as the benchmark algorithms. The authors also evaluated the models in terms of sensitivity and positive predictive value (PPV) and found that their proposed model attained a sensitivity of 69.9% and a PPV or 2.6% which is admirable. From this, we can infer that more CVD patients would receive correct preventative treatment in a timely manner through the utilisation of this proposed model. From analysing the variable importance, it is seen that variables related with physical activity (usual walking page) and blood measurement is typically more significant for this AutoPrognosis model than traditional risk factors such as in existing systems.

Conclusively, through this study the authors explained their implementation of their proposed approach to CVD prediction using the AutoPrognosis model and through their evaluation and comparison have proven its statistical significance and improved prediction accuracy over existing systems such as the Frammingham score, the benchmark algorithms and the Cox PH model. Additionally through the variable ranking they were able to find new determinants in the prediction of CVDs as well as identify the relative importance of the features. They also revealed complex interactions between characteristics or features in individuals, identifying risk predictors specific to specific sub-populations. For instance, the feature of the existence of long-standing illness is a strong determinant for female patients but less predictive for males. Additionally, features from ECG records were stronger determinants for males than females. The limitation to this study is the absence of cholesterol biomarkers or blood-based biomarkers, which are effective determinants, so they were not able to study its effect on CVD risk prediction. Additionally, the patients in the dataset are ethnically homogenous where the majority being of white ethnicity, so ethnicity-specific risk determinants were not able to be studied.

Overall, this paper has been influential to showcase the importance of understanding the features in a dataset and the complex relations that exist among these features and their relative importance in affecting prediction accuracy. Along with that, the structure of the author's proposed model can be taken into consideration when designing our model.

2.4.9 Summary of Existing System and Literature Analysis

Table I
Summary of Existing System and Literature Analysis

Study	Dataset	Preprocessing	Tool	No. of Attributes Used	Hyperparameter Tuning	Classification/ Algorithms Used	Evaluation metrics	Best Algorithm and Result
Nashif et al. [9] (2.4.1)	UCI Cleveland Heart Disease dataset and Statlog Heart Disease dataset	Feature selection on UCI Cleveland Heart Disease dataset, selecting 13 features	WEKA	13 features	None	- Naive Bayes - ANN - SVM - Random Forest - Simple Logistic Regression	- accuracy - precision - recall - fscore - sensitivity - specificity	SVM - accuracy: 97.53% - precision: 95.95 - recall: 97.50% - f-score: 96.72% - sensitivity: 97.50% - specificity: 94.94%
Hashi and Zaman [35] (2.4.2)	UCI Cleveland Heart Disease dataset	- Missing value identification and removal - Removed outliers or inconsistent values - Removed redundancies with the attributes - Separation - Feature scaling - Normalisation	Not mentioned	14 numerical-valued attributes	Grid search	- Logistic Regression - KNN - SVM - Decision Tree - Random Forest	- accuracy - precision - recall - fscore	KNN with hyperparameter tuning - accuracy: 91.80% - precision: 93.55% - recall: 90.62% - f-score: 92.06%

Reddy et al. [36] (2.4.3)	UCI Heart Disease dataset (Cleveland, Switzerland, Hungary, V.A. Medical) and Statlog Heart Disease dataset	<ul style="list-style-type: none"> - Feature selection methods, such as wrapper methods, filter, recursive feature elimination, variable importance estimations, learning vector quantization model, embedded and ensemble and hybrid methods - Normalisation (eliminate outliers or high values in the attributes) 	R programming language and the CARET package for data preprocessing, splitting and ML model implementation	14 attributes	None	<ul style="list-style-type: none"> - KNN - SVM - Random Forest - Naive Bayes - NN 	<ul style="list-style-type: none"> - accuracy - sensitivity - specificity 	Random Forest with 8 features <ul style="list-style-type: none"> - accuracy: 94.96% - sensitivity: 0.914 - specificity: 0.977
Kwakye and Dadzie [37] (2.4.4)	Framingham Heart Study Dataset	<ul style="list-style-type: none"> - Identified missing values, range of attributes to avoid outliers - Data transformation -feature selection 	Not mentioned	16 attributes (all)	Grid search	<ul style="list-style-type: none"> - KNN - SVM - Decision Tree (CART) - Logistic Regression - Random Forest - Naive Bayes 	<ul style="list-style-type: none"> -cross-validation accuracy (CV) - Hold-out prediction (HOP) testing using 	Logistic regression <ul style="list-style-type: none"> - CV: 0.728592 (original), 0.729461 (smote) - HOP testing: 0.5127 (original), 0.6745 (smote)

							ROC-AUC	
Gazeloğlu [7] (2.4.5)	UCI Heart Disease dataset (Cleveland, Switzerland, Hungary, V.A. Medical)	Feature selection (Correlation-based Feature Selection (CFS), Fuzzy Rough Set and Chi-Square algorithms)	WEKA, Python and MATLAB	14 attributes	None	<ul style="list-style-type: none"> - Decision Tree (J48) - ADTree - KNN - RoughtSet - Logistic Regression - Random Forest - NBTree - RBFNetwork - Fuzzy Rough NN - Fuzzy NN - NN - Multilayer Perceptron (MLP) - Naïve Bayes - SVM (polykernel, normalised polykernel, puk and RBF kernel) - Genetic Programming 	<ul style="list-style-type: none"> - Classification Rate - TP Rate (sensitivity) - FP Rate (sensitivity) - ROC-AUC - Kappa Statistic/Coefficient 	<ul style="list-style-type: none"> Naive-Bayes Algorithm with CFS feature selection - Accuracy Rate: 85% - TP Rate: 0.897 - FP Rate: 0.210 - ROC-AUC: 0.905 - Kappa Statistic/Coefficient: 0.691
Alhussainy and Jasim [38] (2.4.6)	UCI Heart Disease dataset (Cleveland, Switzerland, Hungary)	<ul style="list-style-type: none"> - Replacing missing values - Transforming continuous values to discrete 	Not mentioned	14 attributes	None	<ul style="list-style-type: none"> - Recurrent Neural Networks (RNN) - Deep Belief Networks - Artificial Neural Networks (ANN) Optimised by Particle Swarm Optimisation (PSO) 	<ul style="list-style-type: none"> - accuracy - sensitivity - specificity 	<ul style="list-style-type: none"> Proposed Neural Network Model - accuracy: 84.67% - sensitivity: 80% - specificity: 90.72%

	n, V.A. Medical)					and Ant Colony Optimisation (ASO)		
Sajja and Kalluri [39] (2.4.7)	UCI Cleveland Heart Disease dataset	- Preprocessing using Pandas package (no further explanation)	Not mentioned	14 attributes	None	- Convolution Neural Network (CNN) (Proposed) - Logistic Regression - KNN - Naïve Bayes - SVM - NN	- accuracy	Proposed Convolution Neural Network (CNN) - accuracy: 94.78%
Alaa et al. [40] (2.4.8)	UK Biobank dataset	- Missing values imputation or removal of attribute with missing values	Python Programming Language and the Scikit-Learn Library	473 attributes	Grid Search	- AutoPrognosis model (proposed) - Framingham Risk Score, - Cox Proportional Hazard Model - SVM - Random Forest - NN -AdaBoost -Gradient Boosting Machines	- AUC-ROC - sensitivity	- AUC-ROC: 0.752 - sensitivity: 69.9%

Conclusion and Gaps in Existing Studies/Literature

To sum up, the above are summaries (2.4.1-2.4.8) of some or 8 of the chosen literature sources that were reviewed and highlighted during our research of journal articles in this field of heart disease classification/prediction with ML due to their contribution in further developing our understanding regarding this problem domain and in implementing the solution using ML. Hence, from reviewing these sources, a better understanding of the general steps required to be performed, such as specific preprocessing techniques, potential ML algorithms, feature selection techniques, hyperparameter tuning methods and evaluation metric were learned. Thus, from this preliminary review it is evident that the prediction accuracy of the ML model can change depending on the quality of the data source, attributes considered, preprocessing performed, type of ML algorithms, hyperparameter tuning methods, and the data partitioning ratio (train:test) employed. Designed and proposed hybrid models tend to be performant as well such as Alaa et al. 's implementation of their AutoPrognosis model.

The main points that were standard across the journal articles and similar projects done were summarised in the table 1 above to help with further condensing the outcome of the literature review and to help identify the specific information to be incorporated in our implementation as well as for quicker future revisions.

At the same time, from analysing these literature sources, gaps or limitations that exist with the current heart disease prediction and classification projects and systems were identified. The following are some of the limitations that were identified:

Feature Selection

Firstly, there is a lack of use of feature selection methods in current literature and proposed systems. Utilising more input attributes to train and test a model does not necessarily indicate a proposed ML model that has more prediction accuracy. Instead, this could be the reason for the low accuracy, as less correlated attributes are also considered for the model building process. So, first the correlation of the attributes in the dataset are studied using a multivariate plot such as a correlation matrix is generated to identify the degree of correlation between the features present in the dataset and use this to find hidden patterns. From there, feature selection can be performed which is essentially the choosing or selection of only relevant and appropriate input features to be considered for the ML model construction. In the Machine Learning Implementation section, this process is explained in more detail as well as how it will be implemented in practice.

Hyperparameter Optimisation

Secondly, another methodology that is not made use of when developing these systems is hyperparameter tuning or optimisation. This process is performed in an attempt to improve the prediction accuracy by finding and utilising the most optimal combination of parameters also known as the hyperparameters that is able to train a model to maximise its performance. One way the hyperparameter combination is found is through the grid search method among others. This method is an exhaustive way of finding hyperparameters where every combination of hyperparameters is trialled using a grid, before returning the best-performing combination. This process is explained more thoroughly and how it is implemented in practice in the Machine Learning Implementation section for Hyperparameter Optimisation (section 3.4.5). Overall, this

is one aspect that should be considered in the ML model development process to increase and maximise prediction accuracy.

Data Preprocessing

Additionally, another limitation to present studies and systems is the lack of preprocessing methods used to prepare the dataset before it is fitted with an ML model. This can be seen in sources [9] and [7], where the role of data preprocessing is overlooked or not included in the model development process. In reality, a correctly cleaned and pruned dataset makes it more suitable to be used for training and testing ML models as it can even improve the performance and evaluation metric results obtained. Therefore rather than studying and understanding the dataset, its distribution and faults, as well as the data cleaning and pruning techniques available, the implementation may tend to rush into determining the ML algorithm or designing model. Overall, the combination of a well-prepared dataset with the correct classification algorithms can ensure a prediction system with improved performance and accuracy. In the data preparation section of the ML Implementation section (section 3.4.3), the tentatively explored data preparation techniques are explained in terms of theory and practical application/implementation.

Therefore, these limitations were kept in consideration when designing our ML models and for our implementation of the heart disease prediction and classification system, to provide novelty features and functionality that can improve the program's prediction accuracy as well as the state of the field.

3. Methodology

This section will detail the steps behind the methodology for the development of this heart disease prediction and classification system.

3.1 Phases of Implementation

The following describes a brief overview of the general stages that our project will involve. Hence, the methodology for this project can be divided into five phases as follows:

Phase 1: First, we start with the system planning stage, in which the most fundamental aspects of our project are clearly defined, including the problem statements, aim, objectives, deliverables scopes, and milestones to define our motivation, ensure the development is on track and the final product fulfils or satisfies predetermined aspirations. During this stage literature review is conducted to understand the current status quo of heart disease classification systems and the gaps that they possess and our project can improve on. Additionally, this preliminary study aids us in determining the various ML algorithms that can be applicable and implemented for our problem domain of heart disease classification and prediction. Through this, we can understand the capabilities, characteristics and considerations for these models. Literature review is also conducted to understand the problem domain of CVDs, HDs, and ECGs, to gain a better understanding of the context the model will be applied on to improve its effectiveness.

Phase 2: The next stage touches more on the development of the ML models. This stage starts with the selection of the dataset and the acquisition or collection of the data from the dataset. If it has insufficient records, data from other sources are merged as well. Subsequently, involves the preparation of the dataset that is to be used to train and test our ML models as data understanding is performed where the dataset contents are visualised through univariate and multivariate plots and through exploratory data analysis (EDA) is performed where a description of the main characteristics is generated to be visualised in a graphical format and to understand its distribution and the range of each attribute. Preprocessing is conducted to clean the dataset of noise and eliminate any missing values and outliers in the distribution. The exact preprocessing methods necessary will only be determined during the actual development of the ML stage after data understanding is completed. Feature selection or engineering is performed where we select the relevant features from the group of attributes and the extraction of hidden features influential to our study. It is performed based on the multivariate plots and from understanding the correlation between features between each other and the target variable. The dataset is then partitioned into specified ratios of training and testing data. We will test the effect of different data partitioning ratios on the performance of the ML model to find the best combination with the least underfitting and overfitting. For instance, we will test 50:50, 60:40, 70:30, and 80:20. Then, begins the development of the classification algorithms based on the predetermined ML models that were chosen. The ML models are trained with the training dataset and then subsequently validated with the testing dataset. In the experimental testing and results phase, we will utilise appropriate evaluation metrics to compare the performance of the ML models, namely accuracy, precision, f-score, etc. So, the performances of the different ML models are computed, and compared hyperparameters are incorporated with the designed ML models to test their effect on the model performance. From there, using the model selection method employed, we can observe the best-performing ML model for classifying and predicting heart disease among patients and it will be chosen to be integrated with our web-based system.

Phase 3: During this stage, requirements for the prospective web app system are elicited through methods such as reviewing previous works and literature sources as well as ethnography or reviewing similar heart disease prediction and classification systems. By having a clear list of system requirements, we can then define the desired system functionalities. The technologies and tools required for the implementation of the ML model, as well as the web app system, are defined.

Phase 4: A prototype design is made, and diagrams are constructed to visualise the proposed system's UI as well as its workflow. This will provide a clear and visual plan which will aid in development. Appropriate UML diagrams are designed to illustrate the information and logic flow of the final system.

Phase 5: This stage involves the development of a web app in which the chosen ML model will be integrated with and deployed, along with deploying tasks to deploy the web application to the hosting environment (AWS). This will allow the end user to easily input their credentials and details that are required for the ML model to provide a verdict. Appropriate testing methods will be performed, namely, unit testing of the web app components and integration testing of the integration between the ML component and the web app system with the database and backend, so that it is free from errors that may affect the user experience to ensure the system is functional and fulfils initial expectations.

3.2 Graphical Flow/Flowchart of Components for the Methodology/Implementation

This section illustrates the sequence of the above main methodology steps or phases for the development cycle for this project. Hence, figure 2 shows the flowchart of the general phases involved in the overall project development.

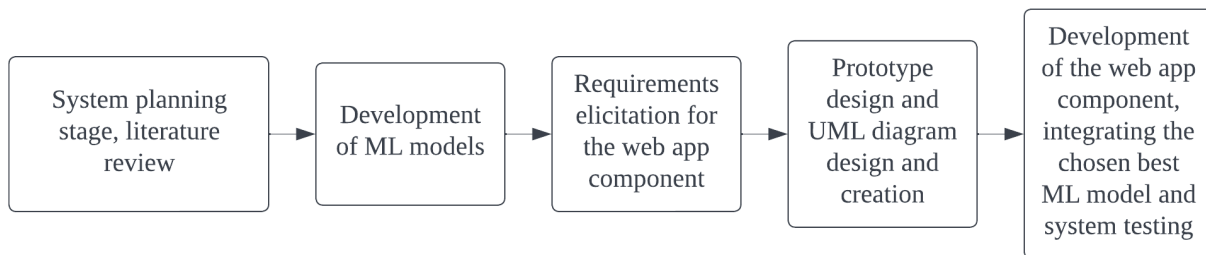


Figure 2: Flowchart of main phases of the project and its implementation. An abstract view of the project development cycle.

Following that, figure 3 displays the steps for implementing and developing the ML models as described in phase 2 of the phases of implementation, to determine the best performing model that will be chosen and integrated to the web platform.

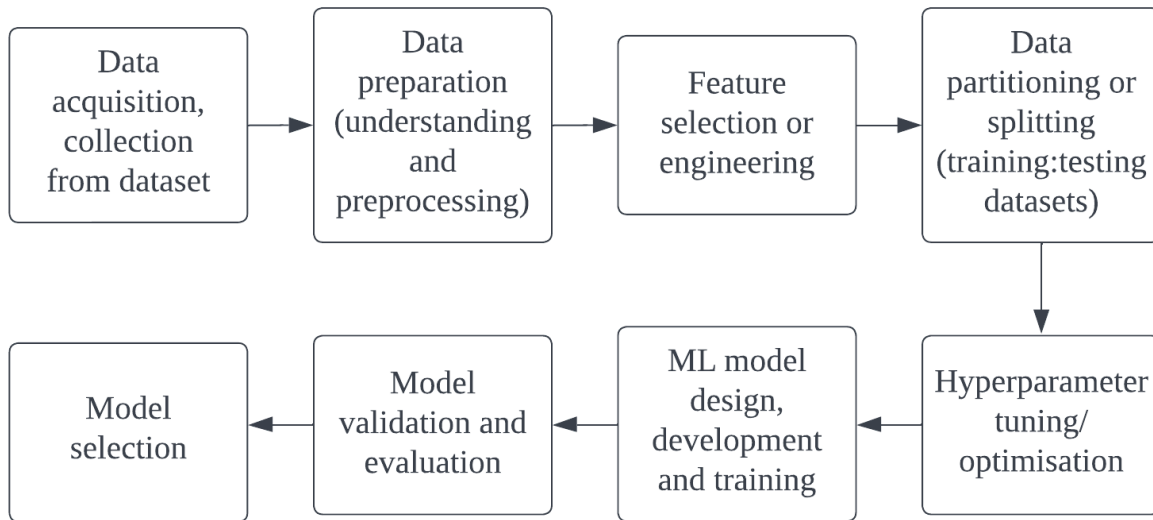


Figure 3: Flowchart of main phases of the ML implementation process.

3.3 Software Process Model

In order to develop this heart disease prediction/classification system, the waterfall development model was opted for the entire software process model or Software Development Life Cycle (SDLC). A waterfall model is considered a plan-driven model with several distinct phases of specification and development that are performed sequentially from start to finish, typically in one cycle. Thus, it is seen to possess a linear-sequential and continuous design approach. This process model ensures that the final system can be produced in one cycle with minimal backtracking to previous phases for changes and minimal iterations. So, it ensures a quick development process and is suitable for smaller scope or scale projects with a limited time constraint. Additionally, applying this model necessitates clearly and thoroughly defining the system requirements beforehand, as it does not easily enable backtracking to the planning and specification stage and so changes to the project scope can be minimised. Nonetheless, by fixing the requirements, this ensures the project has been understood thoroughly, and the development progresses smoothly from start to finish sequentially without requiring backtracking.

The following diagram displays the flowchart of the general phases in a typical waterfall software process model:

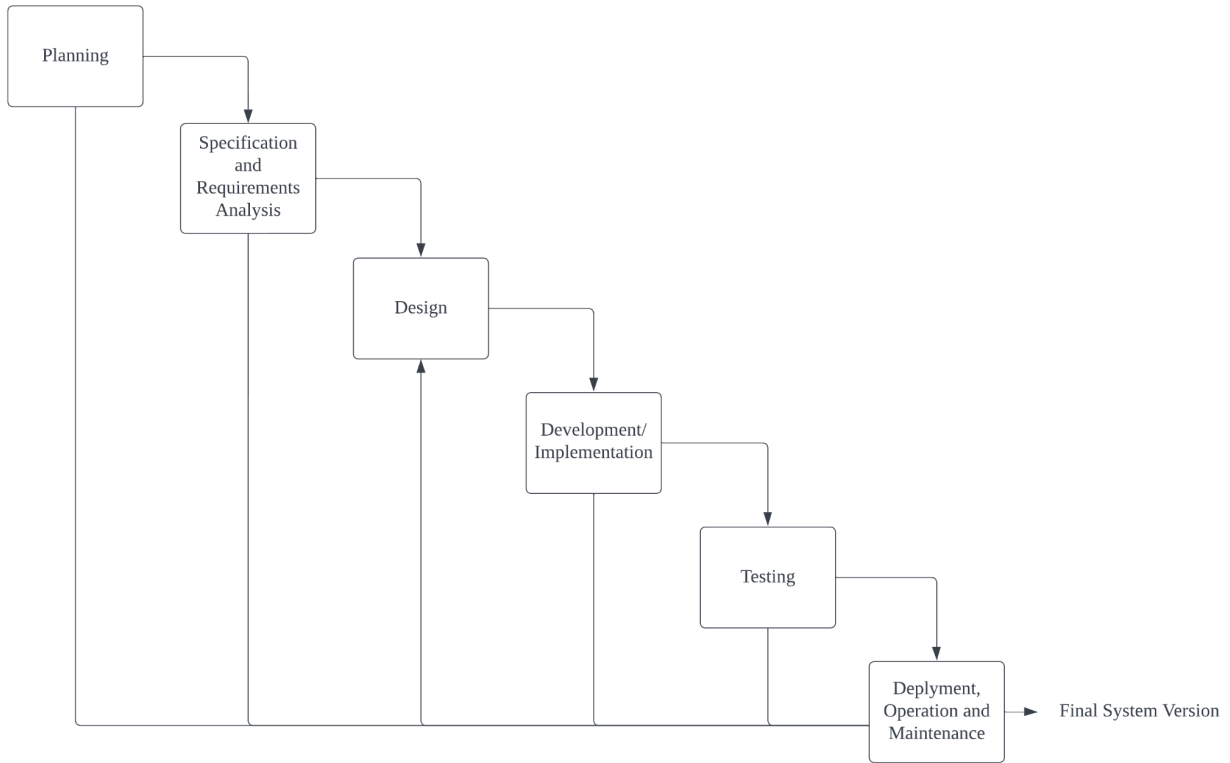


Figure 4: Waterfall development model flowchart.

In contrast to the other available software process models that are available, such as the incremental model or the agile development method, which allows for progressive builds to be released after every cycle or increment, it is found that the waterfall model is more suitable for implementing this project and the development phases that are involved. Typically, an incremental model would be useful in the event an earlier build of the system should be deployed and released for testing and usage, thus decreasing the waiting time to begin the usage of the system as well as being more flexible to requirement changes and reducing the risk of system failures as testing is performed at every increment and backtracking to previous phases is easier. However, our justification for choosing the waterfall model is that for our development, from looking at the phases to undergo, it involves some major dependencies between the ML implementation (Phase 4) and the web development phases (Phase 5). This means that we intend to ensure that all the ML algorithms we have set out to implement have been developed, tested and compared, before the best model is chosen as well as deployed and integrated in a website format that is usable by the end-user. This ensures that there is only one deliverable which is the final system utilising an ML model that fulfils evaluation metrics and possesses admirable prediction and classification accuracy, especially considering the problem domain at hand, which necessitates that the system provides a correct and accurate diagnosis.

Additionally, a waterfall model is generally more suitable for smaller scope or scale projects and systems where the requirements are defined and assessed clearly and comprehensively, ensuring the development time is quicker as there is little to no backtracking to any previous phases. This holds true for our project which only consists of mainly five phases. Moreover, for the

implementation of the ML component, which is seen as the main component of our system, this component/phase involves dataset preparation, preprocessing, partitioning, ML algorithm development and application, and finally testing. Thus seeing as these steps are sequential and each step is dependent on the previous step to occur, a waterfall method will ensure that every step is performed sequentially, ensuring there is less complexity in the development process and the process is completed quicker. Along with that, having a well-defined work plan with predefined deadlines for every stage ensures the final system can be ready for use within the specified timeline.

The following sections provide a brief analysis and explanation for each phase of the SDLC for this project development:

Planning Phase

This is the first phase of this waterfall development model and during this stage, the developers define the project's fundamental aspects and are part of the initial project plan, including the problem statement, project goals, objectives, deliverables and scope. This provides the developers a clear list of goals the platform should fulfil before it is released officially for use resulting in lesser backtracks or changes to the project goals. For our project these key information have been highlighted in the introduction section and its subheadings (1.1-1.6). As per the work plan in the activity log, a total of 7 weeks are allotted for this planning phase to ensure it is completed and checked thoroughly before progressing to the subsequent stage.

Specification Phase

This specification and requirements analysis phase is the second stage of our waterfall development model. In this stage, the requirements of our prospective system are elicited, gathered from various sources and assessed, which are then organised to generate the Software Requirement Specification (SRS) document. These requirements serve as a checklist to be considered when designing and developing the system in the following development phase. Typically these requirements can be divided into two categories in the SRS, namely, functional and non-functional requirements. In our case, for the system requirements, that would be listing the system functionality that our heart disease prediction system should fulfil, such as allowing users to sign up for a patient account or to input physiological data and obtain a diagnosis based on the ML model's processing of the input data. As for the non-functional requirements, other aspects of the system should meet expectation, such as the main requirement being accuracy where the system should possess high prediction accuracy to ensure users obtain the correct diagnosis based on their entered data. As per the work plan in the activity log, a total of 6 weeks are allotted for this specification and requirements analysis phase to ensure it is completed and checked thoroughly before progressing to the subsequent stage.

Design Phase

The third stage is the system and software design phase, where the design of the web app of the system and its user interface (UI) is prototyped. This was done using Adobe XD where the pages and main components of the prospective system's web component is designed to visualise how it will be represented and implemented. The design and layout elements of the prospective system can be determined. Additionally, for this, we can make use of Unified Modelling Language (UML) diagrams, such as an activity, sequence or use case diagram. This can help the developers

to visualise the overall system architecture as well as to serve as a plan or blueprint for them to convert into code which they can develop. These diagrams also showcase the operations that can be performed on the web system and the flow between actions. These designs can be found in section 3.7 and 3.8 of this planning document. As per the work plan in the activity log, a total of 3 weeks are allotted for this designing phase to create the required designs to plan out the envisioned web app system and its operations and logic flow in a graphical or visual format before progressing to the subsequent stage.

Development Phase

In the development or implementation stage, as the name suggests, the system functionality and requirements mentioned before are realised and implemented. Using the tools mentioned in this planning document, the system is developed according to the specifications defined before and following the system design in the previous stage, which acts as reference points. The system functionalities from before are split into smaller units and iteratively developed and evaluated to ensure their fulfilling requirements and expectations and are functional. Hence, it is evident that this stage requires the longest time and forms the bulk effort of this project's focus. For this project, the different ML algorithms will be developed first using the Python programming language and the scikit-learn library and then the web application that will be housing the chosen best performing ML algorithm will be implemented using the Django web framework. Hence, as per the work plan in the activity log, a total of 8 weeks are allotted for this development phase to ensure the individual units are developed correctly following the requirements and the initial expectations. Simultaneously, testing is performed to ensure the developed components are functional and to resolve any errors before progressing with any of the future stages of the development and the project.

Testing Phase

Towards the end of the software development lifecycle, the developed system is evaluated in terms of some predefined evaluation metrics or test cases by internal and external testers. For our project, we have two main components to test, which are the ML models as well as the web app that houses the chosen best model. For evaluating the ML models, we use evaluation metrics such as accuracy, f-score, sensitivity and specificity which is explained further in the Testing section. From there, we can determine the best performing model that is then chosen for the next stage to be integrated with the web application. As for the web application, we test its system functionalities to ensure they are working as intended and the non-functional requirements are fulfilled. For this user acceptance testing can be performed to test the system from the end-users perspective and to ensure the interactions are seamless, error-free and as envisioned when preparing the SRS and system design. This stage is performed once all the system functionalities have been developed and implemented. Hence, any issues or bugs with the system are identified and resolved at this stage of the SDLC and if the program is not working to expectation backtracking to the development phase may be required to ensure the system is functioning as intended. This stage is crucial to ensure that our initial expectations or requirements of the system are correctly implemented without any errors and so the end-user does not experience bugs when interacting with our system. Once all the tests have been passed, all components of the system will then be combined into a unified system, forming the final product. For this there are three forms of testing that are proposed and will be prioritised as seen in the System Testing Plan (3.10), which include unit testing, integration testing and user acceptance testing (UAT)

with beta testing and how we aim to perform the testing of the system components. As per the work plan in the activity log, tentatively 1 week is allotted for this testing phase, however, there are testing steps interleaved with the development steps. Therefore, after the development of the main components and the secondary components, there will be a round of unit testing conducted to ensure the developed components are functional and fulfil the initial expectations and requirements. This 1 week of testing phase, includes among the final rounds of testing of the system to ensure the system is functional before being deployed to the Firebase server platform to test the integration between the web app and ML components and a user acceptance testing and beta testing to validate the system from the end-users' perspective.

Deployment Phase

This final phase involves the deployment, continuous operation and maintenance of the developed system. Therefore, once the developed system's components fulfil our predetermined system functional and non-functional requirements and have been implemented as intended, the components have to be unified or integrated into one system. For our system, that would be integrating or deploying the chosen best ML model with the developed web application to ensure that our heart disease prediction/classification system is easily accessible for the end-user, simply by accessing it through their web browser. Additionally, the final integrated system will be deployed to the Firebase Realtime Database platform. The platform will be developed with correct configurations and the final system is deployed to the designed and developed server. This stage will ensure the system is accessible to the general public through the search engine and web browser and ensure the performance of the system is consistent and stable. Simultaneously, the web system should be tested and evaluated to ensure that it remains functional and working as intended, else appropriate debugging measures should be performed. As mentioned earlier in the project limitations, as this study and project is only for research purposes, at its current iteration the system will not be deployed commercially or for real-world usage to make definitive diagnosis or medical decisions. In conjunction with that, maintenance is performed to ensure the website remains functional for the end-user to resolve any bugs or issues with the platform as well as to incorporate updates to introduce new features or improvements to the system functionality. As per the work plan in the activity log, a total of 4 weeks are allotted for this deployment, operation and maintenance phase to ensure it is completed and checked thoroughly before it can be cleared for end-user usage or in this case, for the submission of the capstone 2 project.

3.4 Machine Learning Implementation

3.4.1 Data Acquisition and Dataset Description

The dataset chosen for this comparative analysis of ML algorithms is a combination of four heart disease datasets from the University of California Irvine (UCI) Heart Disease dataset [41], which was able to be merged, as there were similarities in the attributes of each dataset. Therefore, this UCI heart disease dataset consists of four separate datasets, with data collected from four locations and created by:

- Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
- University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
- University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
- V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

This combination of UCI datasets were found ideal and chosen, after downloading and viewing each of the previously mentioned four datasets, in the planning stage. From my study of the datasets, to summarise, the MLDataR dataset was specific to the R language, and would require additional processing before it could be suitable to our comparative analysis, which is primarily using the Python programming language. Whereas the Frammingham dataset, although is ideal with an admirable number of records and attributes, the attributes were found to be limited or basic, concerning demographic information such as alcohol intake and physical health, whereas the UCI dataset, had those demographic information along with key medical data from sensory equipment, including ECG data. Thus, when studying the correlation of the attributes in this dataset, it was found that there is not strong correlation between the attributes and the target, as seen with the UCI dataset, and will be explored and explained in more detail. Nonetheless, this dataset would be kept as a secondary dataset for our future works to further improve the algorithm by considering more features. Finally the Z-Alizadeh Sani dataset, although had a good number of attributes, had lesser number of records that could be used to sufficiently train and test our ML algorithms, as compared to the UCI dataset.

Therefore, the four UCI datasets were merged. Then, duplicate rows were identified and removed, producing 920 usable rows. Originally, this dataset consists of 76 attributes, but similar published studies relating to the use of this dataset only consider a subset of around 14 attributes, as discovered during our literature review. This was further reinstated by the creators of the dataset through a note at the UCI data repository. Additionally, a simple preliminary correlation study was also performed on the attributes of this raw dataset on its target, to confirm this assumption. Rightly so, it was found that there were 16 attributes we found that had moderate to strong correlation with the target attribute that could be further studied or utilised to train the ML model. This is the first form of feature selection that was performed, where based on the correlation score, the attributes that have an impact on the output or target are found and these features would be considered for the development of the ML models. Hence, the following table shows the 16 attributes that are part of the final dataset.csv file with the removed duplicate rows and the optimal selected features that was loaded into the Python program and further ML processing was performed on, along with the reason for choosing them apart from having a strong correlation score with the target attribute:

Table II
Dataset description

Attribute name	Feature Values	Description	Data Type	Reason For Choosing This Feature
id	0 - 919	Unique id for each patient. 920 records in the final dataset.	Numerical, int	For identifying each record. Removed before conducting further processing or ML algorithm as it doesn't correlate with the target and is simply to uniquely identify each row.
age	28 - 77	Age of the patient in	Numerical,	With the risk of developing

		years	int	cardiovascular or heart diseases roughly tripling with each decade of life, age is a significant risk factor. In adolescence, coronary fatty streaks can start to develop. According to estimates, 65 and older people make up 82 percent of coronary heart disease fatalities. The risk of stroke also doubles every ten years after age 55 [27].
sex	0: Female, 1: Male	Sex of the patient	Categorical, object	In general, men have a higher risk of developing heart disease than do women before menopause. Although more recent data from the WHO and UN refute this, it has been argued that a woman's risk after menopause is comparable to a man's. Compared to a male with diabetes, a female has a higher risk of developing heart disease [29].
dataset	Cleveland, Hungary, Switzerland, VA Long Beach	Place of study	Categorical, object	For identifying source of each record. Removed before conducting further processing or ML algorithm as it doesn't correlate with the target and is simply to identify the source of each row.
cp	0: typical angina, 1: atypical angina, 2: non-anginal, 3: asymptomatic	Chest pain type Typical angina: chest pain related to reduction of blood supply to the heart Atypical angina: non-heart related chest pain Non-anginal pain: generally esophageal spasms (non-heart related) Asymptomatic: chest pain that generally does not display signs or	Categorical, object	Angina is a type of chest pain or discomfort brought on by a lack of oxygen-rich blood to the heart muscle [30]. The sensation in your chest might be one of pressure or squeezing. Your neck, jaw, shoulders, arms, or back may also feel uncomfortable. Even the pain from angina can resemble indigestion.

		symptoms of disease		
trestbps	80 - 200	Resting blood pressure, (in mm Hg). Values exceeding above 130-140 are generally cause for concern.	Numerical, int	The arteries that supply your heart can become damaged over time by high blood pressure [27]. Your risk is even higher if you have high blood pressure along with another condition, such as diabetes, high cholesterol, or obesity.
chol	85 - 603	Serum cholestrol (in mg/dl) serum = LDL + HDL + .2 * triglycerides Values exceeding above 200 is generally cause for concern	Numerical, int	The most likely cause of artery narrowing is a high level of low-density lipoprotein (LDL) cholesterol, also known as "bad" cholesterol. Your risk of a heart attack is also increased by having high blood levels of triglycerides, a type of blood fat connected to your diet. Nonetheless, high-density lipoprotein (HDL) cholesterol, or the "good" cholesterol, reduces your risk of having a heart attack [32].
fbs	0 (FALSE): > 120 mg/dl, 1 (TRUE): < 120 mg/dl	Fasting blood sugar. Compares an individual's fasting blood sugar value of 120mg/dL. Values exceeding 126 mg/dL generally signals diabetes.	Categorical, object	Your body's blood sugar levels rise as a result of insufficient pancreatic hormone production or improper insulin response, which raises your risk of having a heart attack [30].
restecg	0: normal, 1: ST-T wave abnormality, 2: ventricular hypertrophy	Resting electrocardiographic (ECG) results 0: Nothing to note 1: ST-T Wave abnormality, can vary from minor symptoms to serious issues. Indicates an abnormal heartbeat. (T wave inversions	Categorical, object	The United States Preventive Services Taskforce (USPSTF) comes to the conclusion that the potential risks of screening with a resting or exercise ECG are equal to or greater than the potential benefits for people at low risk of cardiovascular disease [33]. There is currently insufficient evidence to determine the balance between screening's advantages and disadvantages for those at intermediate to high risk.

		and/or ST elevation or depression of > 0.05 mV) 2: Left ventricular hypertrophy, either probable or certain by Estes' criteria. Expanded main pumping chamber of the heart.		
thalch	60 - 202	Maximum heart rate achieved. (beats per minute)	Numerical, int	The increase in cardiovascular risk, associated with the acceleration of heart rate, was comparable to the increase in risk observed with high blood pressure. It has been shown that an increase in heart rate by 10 beats per minute was associated with an increase in the risk of cardiac death by at least 20%, and this increase in the risk is similar to the one observed with an increase in systolic blood pressure by 10 mm Hg [30].
exang	0: FALSE, 1: TRUE	Exercise induced angina.	Categorical, object	Angina can cause mild to severe pain or discomfort that typically feels tight, gripping, or squeezing [30]. Angina is typically felt in the middle of the chest, but it can also affect one or both shoulders, as well as your back, neck, jaw, arm or hands. Types of Angina i. Stable Angina / Angina Pectoris ii. Unstable Angina iii. Variant (Prinzmetal) Angina iv. Microvascular Angina.
oldpeak	-2.6 - 6.02	ST depression induced by exercise relative to rest.	Numerical, float	When exercising, a heart that is unhealthy will be under more stress. Thus, this is a key indicator of

				prevalent heart disease in a patient.
slope	0: upsloping, 1: flat, 2: downsloping	Slope of the peak exercise ST segment. Upsloping: improved heart rate with exercise (uncommon) Flatsloping: less alteration (typical healthy heart) Downsloping: Indications of an unhealthy heart.	Categorical, object	When the ST-segment depression is flat or downward-sloping and is greater than 1 mm 60–80 ms after the J point, the treadmill ECG stress test is deemed abnormal. Exercise ECGs that exhibit up-sloping ST-segment depressions are frequently classified as "equivocal" tests. In general, a worse prognosis and a greater likelihood of multi-vessel disease are indicated by the presence of horizontal or down-sloping ST-segment depression at a lower workload (calculated in METs) or heart rate [33]. The length of the ST-segment depression is also crucial because a prolonged period of recovery following a stressful event is consistent with a successful treadmill ECG stress test. The presence of ST-segment elevation > 1 mm, which frequently indicates transmural ischemia, is another finding that is highly suggestive of significant CAD; these patients are frequently urgently referred for coronary angiography [31].
ca	0-3	Number of major vessels (0-3) colored by fluoroscopy.	Categorical, object	A colored vessel indicates that the physician can see the blood flowing through it. The more blood movement, the better, indicating absence of clots and better heart health [41].
thal	0: normal, 1: fixed defect, 2: reversible defect	Thalium stress test. Shows the thalassemia level. 0: Normal 1: Previously defective but now	Categorical, object	Thalassemia is an inherited blood disorder that causes your body to have less hemoglobin than usual. Thus, it has correlations with the likelihood for a patient to develop heart disease [30].

		okay 2: Improper blood flow during exercise target		
num (Target)	0-4	The predicted target attribute. The level or degree of heart disease in the patient. 0 = absence of heart disease 1, 2, 3 = presence of heart disease	Categorical, object	This is a multivalued target attribute, that is crucial at is the output that is aimed to be predicted based on the input of the rest of the attributes.

After the dataset has been prepared, it is loaded into the Python program using the `read_csv` method of the Pandas library, allowing to read and import the UCI-published heart disease dataset in *.csv (comma-separated value) file format and stored in a DataFrame object. The main Pandas data structure is the DataFrame object, which is a two-dimensional table with labelled axes running along the rows and columns. From there, the Pandas dataframe can be used to perform a variety of data manipulation operations along rows and columns.

3.4.2 Data Understanding and Visualisation

Data understanding is where exploratory data analysis (EDA) is performed on the collected data to understand the distribution of the attributes of the data, identify its hidden patterns, discover initial insight, as well as generating the data description (descriptive statistics) or key information regarding the data's attributes, such as the skewness, min, max, percentile values, measures of central tendency (mean, median, mode) and measures of variability (range, interquartile range, standard deviation, variance). Thus, enabling us to familiarise with the data.

This is the first part of the data preparation step that was performed, as EDA and data understanding is a key step to the ML process in order to get meaningful results. It helps to identify necessary data preprocessing steps that need to be performed in the following stage. Additionally, through this step, we gain a better understanding of the dataset we are dealing with, to better understand the problem domain, the data distribution (with the `.info` method), and the patterns in the data (correlation).

First, the size of the dataset, we're dealing with is identified, using the `df.shape()` method, that returns the dataset's number of rows (920 records) and columns (16 attributes). Following that, the Pandas dataframe `info()` method returns preliminary general information about the dataframe's number of columns and row entries, the number of non-null values, and the datatype of the column's values, as well as the memory size usage of the dataframe.

```

Size of UCI dataset: (920, 16)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   id           920 non-null    int64
1   age          920 non-null    int64
2   sex          920 non-null    object
3   dataset      920 non-null    object
4   cp           920 non-null    object
5   trestbps     861 non-null    float64
6   chol         890 non-null    float64
7   fbs          830 non-null    object
8   restecg      918 non-null    object
9   thalch       865 non-null    float64
10  exang        865 non-null    object
11  oldpeak      858 non-null    float64
12  slope        611 non-null    object
13  ca           309 non-null    float64
14  thal         434 non-null    object
15  num          920 non-null    int64
dtypes: float64(5), int64(3), object(8)
memory usage: 115.1+ KB
None

```

Figure 5: Shape of the dataset and general info about dataset

Following that, descriptive statistics of the numeric columns in the dataset are produced using the Pandas dataframe describe() method, which highlights the central tendency (mean score), dispersion (standard deviation), and shape of a dataset's distribution (min, max, quartile, median, third-quartile) while excluding NaN values.

```

describe 1:

```

	count	mean	std	min	25%	50%	75%	max
id	920.0	460.500000	265.725422	1.0	230.75	460.5	690.25	920.0
age	920.0	53.510870	9.424685	28.0	47.00	54.0	60.00	77.0
trestbps	861.0	132.132404	19.066070	0.0	120.00	130.0	140.00	200.0
chol	890.0	199.130337	110.780810	0.0	175.00	223.0	268.00	603.0
thalch	865.0	137.545665	25.926276	60.0	120.00	140.0	157.00	202.0
oldpeak	858.0	0.878788	1.091226	-2.6	0.00	0.5	1.50	6.2
ca	309.0	0.676375	0.935653	0.0	0.00	0.0	1.00	3.0
num	920.0	0.995652	1.142693	0.0	0.00	1.0	2.00	4.0

Figure 6: Descriptive statistics of the dataset's numerical attributes

Following that, the Pandas dataframe isna().sum() method returns the number of null values in each attribute of the dataset. Thus, from here, we can see that owing to the presence of missing values for some of the attributes, the missing values need to be treated to ensure that all records have valid values to train and test the ML models, in this case, the chosen method is imputation.

After that, we identify whether there exists duplicate records in the dataset using the df.duplicate() function. Here, it is seen that there is no duplicate records, so the removal of duplicate rows was skipped.

Next, univariate analysis, bivariate analysis, data visualization, and data analysis were performed on the target variable as well as the other input attributes of the dataset. Exploratory Data Analysis (EDA) uses visualization techniques to display or identify trends and patterns, hence, the data are represented graphically. So, the main characteristics of the dataset are summarized and patterns are sought using exploratory data analysis (EDA).

Univariate Plots

We start with univariate plots of the dataset's attributes, to understand each one's distribution and identify whether preprocessing on individual attributes is necessary, for instance outlier removal or missing value imputation.

Firstly, we view the distribution of the target attribute, num, which is the level of heart disease a patient possesses, using the `df["num"].value_counts()` statement. This will let us know whether the target attribute values are balanced and suitable for further processing with the ML models. Rightly so, based on the results, it shows that the values are balanced between the 5 classes, according to the real-life context. This can be further verified by grouping the classes of the attribute, where 0 denotes the absence of heart disease and 1,2,3,4 denote the presence of heart disease. So, with this a separate column is created in the dataset, "hd", to observe the distribution of the presence and absence of heart disease amongst the patients in the dataset, and this information can be used together with the other attributes in the dataset, to see their correlation with this "hd" attribute.

Moving on, a density plot (`df.plot(kind= "density")`, Pandas dataframe method) is used to visualise the distribution of the numerical attributes of the dataset, namely, age, resting blood pressure, cholesterol, maximum heart rate, ST depression, and number of fluoroscopy-colored major blood vessels. From here, the plots show an almost normal bell-shaped curve, showing that the numeric attributes are normally distributed, except for a slight deviation with the oldpeak and cholesterol attributes, but doesn't necessitate removal of outlier values, as they are valid values looking at the context, where cholesterol and oldpeak values at that rate is possible. For instance there is a potential outlier with cholesterol value greater than 500, which is checked, and then observed to be usual.

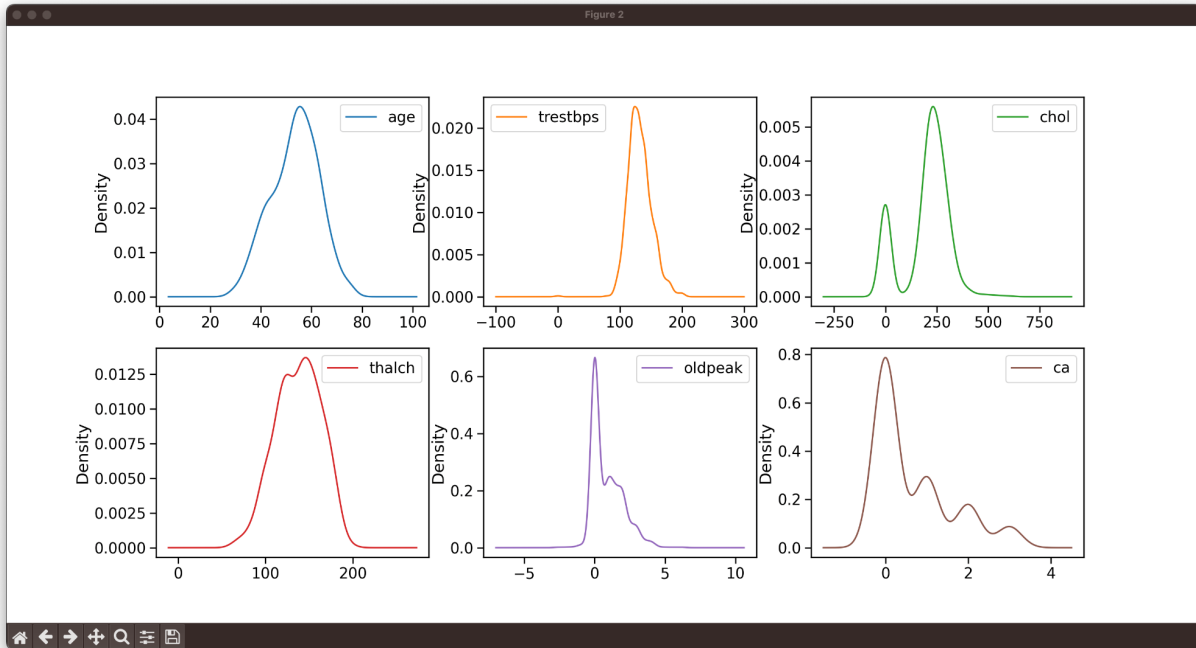


Figure 7: Univariate plots of density graphs of the dataset's numerical attributes

Meanwhile for the categorical attributes of the dataset, a bar chart is constructed of the frequency of the feature values for that particular attribute using the Pandas dataframe method, `df.plot(kind="box")`. These categorical attributes include the patient's gender, location or source of the dataset, chess pain type, fasting blood sugar level, rest ECG rate, exercised induced angina level, slope of peak exercise ST segment, and thalium stress test. By looking at these distributions, we can judge whether the data is balanced between the classes of the attributes and whether any data balancing is in order. From looking at the distribution, there is not any large distinction in the frequencies, that is abnormal with its real-life context, thus no data balancing is required to be performed on these categorical data.

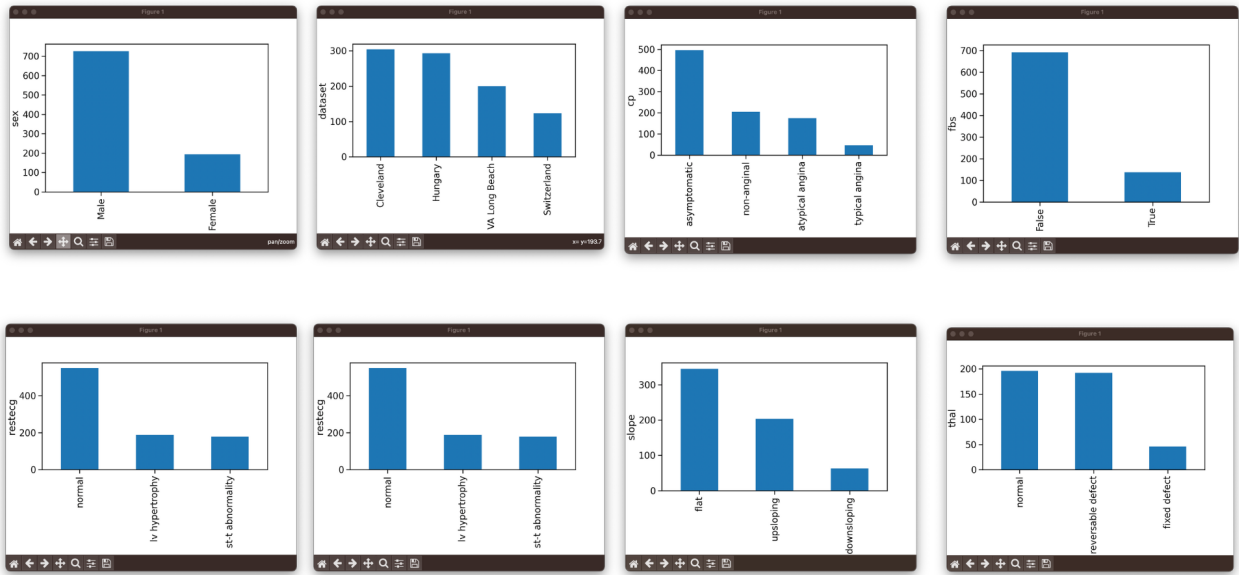


Figure 8: Univariate plots of bar charts of the dataset's categorical attributes

Based on these univariate plots, some findings can be derived from analysing the visual diagrams. Some of the categorical variables (Sex, Chest Pain Type) are represented in textual form, which would be better to be represented in numeric form for the algorithm to understand the inputs better and perform more accurate processing. Thus, one hot encoding is ideal for these categorical data, as there is less number of classes, and so the number of dummy variables that will be generated will not be too overwhelming on the program memory space consumption. Also, there are no significant outliers looking at the individual distribution bar charts and density graphs.

Furthermore, boxplots were used to visualise the distribution or scale of the numerical attributes of the dataset, especially when placed next to or comparing with the other attributes. This can also help us to identify outliers present in the data. From looking at the boxplots, it is clear that scaling is required to ensure the data is distributed consistently across the numerical attributes, to ensure more accurate processing of this data by the machine learning models. Currently, the data appears to be skewed for attributes such as oldpeak, chol, trestbps, and thalch, hence requiring scaling, preferably min-max scaling, which scales the data based on the range of the attributes.

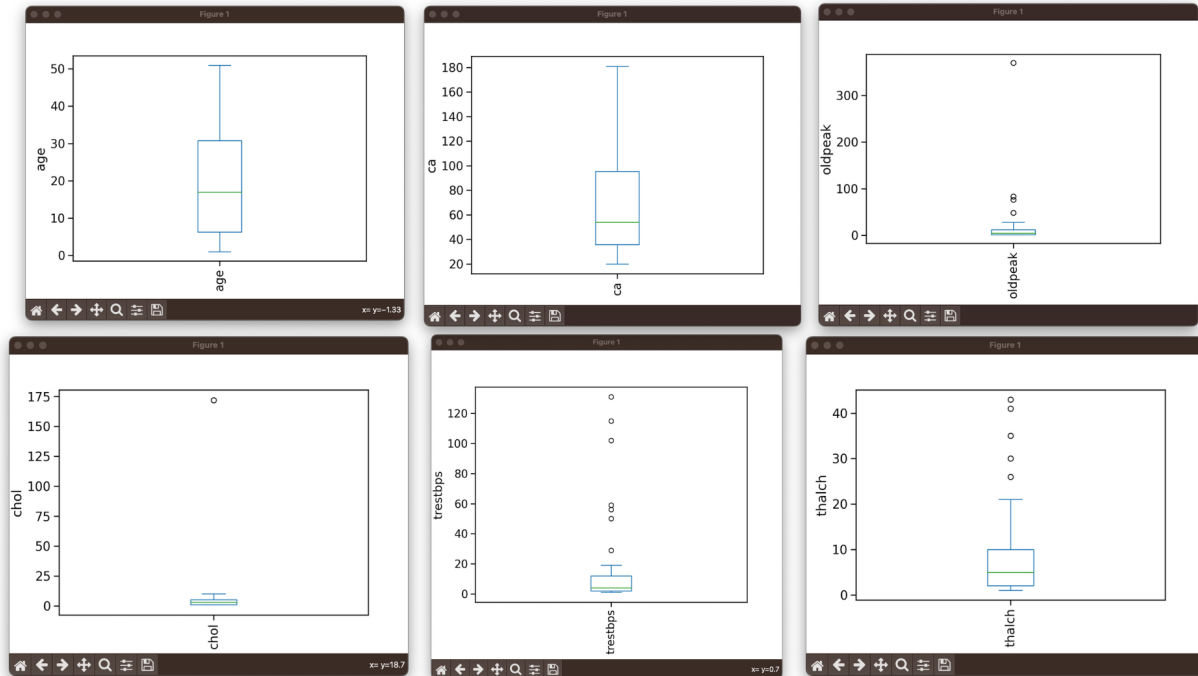


Figure 9: Univariate plots of box plots of the dataset’s numerical attributes

Multivariate Plots

Then for the multivariate plots, heatmaps are used, as it is a great way of representing correlation between the target variable “num” and the numerical input attributes, visually. Visualizing the data features to find the correlation between them which will infer the important features. Therefore, for this classification problem, a correlation matrix can be developed which illustrates the relations or degree of association between the attributes or features to one another and the target value, which according to the chosen dataset, is the “num” (attribute 58) attribute. From there we can determine what features we can exclude from the subsequent processing phases and what crucial features should be maintained. There are other tests that can also be performed such as univariate testing to find the p-value of the features to determine whether they are significant (for $p < 0.05$). However, for this project, this was not conducted to reduce the complexity of the program and the correlation matrix relayed sufficient information regarding the feature importance of the dataset.

The color-coded plot for the 2-D matrix data is provided by the Seaborn heatmap API, whereas pairwise correlation (correlation of the two variables in a matrix) of the dataframe's columns is provided by the Pandas dataframe `corr()` method. This method excludes NA or null values. Hence, using this method, we can identify both strong and weak correlations between the various columns and the target variable, as well as positive and negative correlations between the individual numerical attributes. This can help us in feature selection, as features that are weakly correlated can be disregarded, whereas strong features can be prioritised. Model predictions can be described using positive and negative correlations, where positive correlation suggests that when one variable's value increases, the other variable's value should increase as well. Conversely, if there is a negative correlation, it means that as the value of one variable decreases, so does the value of the other variable. There is no linear relationship between the variables if

there is zero correlation. The attributes used in the Seaborn heatmap method include the annot parameter set to True, so each cell is labelled with the corresponding correlation value, and the linewidth specifies the width of the line separating each cell of the matrix.



Figure 10: Multivariate plots in heatmap form of numerical attributes with the target “num”

Additionally, due to the high amount of attributes in the dataframe, it may look slightly cluttered. Hence, a barchart with the attribute’s correlations with the target attribute, num, has been added and sorted accordingly.

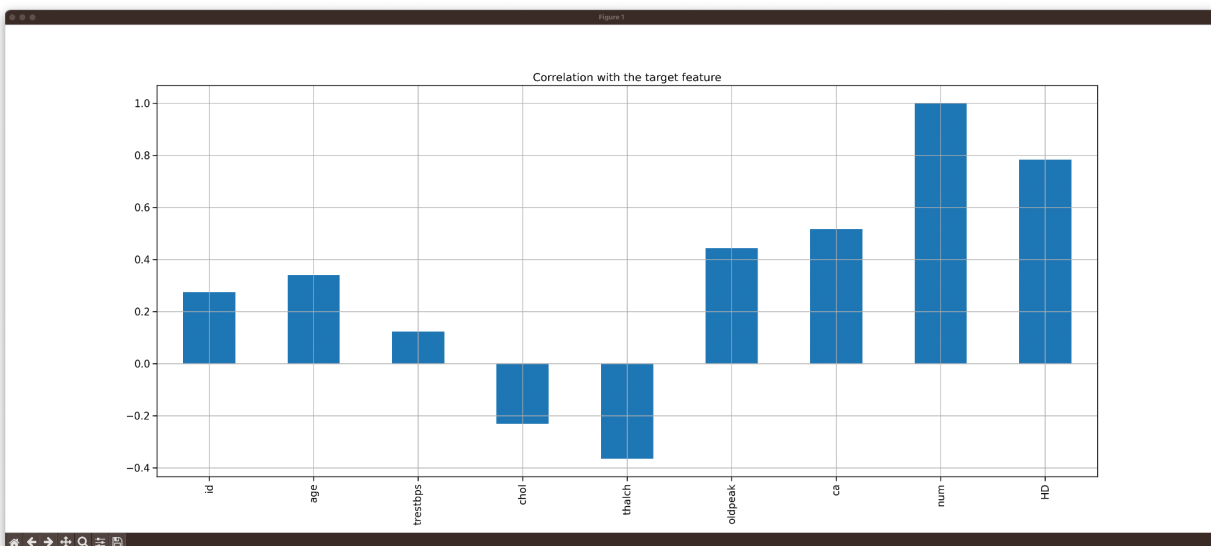


Figure 11: Multivariate plots in bar chart form of numerical attributes with the target

For the other categorical attributes, bar charts can be plotted to compare the individual feature values and its correlation with the heart disease level or the target, which can be obtained using the Pandas dataframe, crosstab() method. Thus, this can provide a clear comparison of the dataset attributes classes and their relation with the target attribute (num), which is the level of heart disease a person possesses.

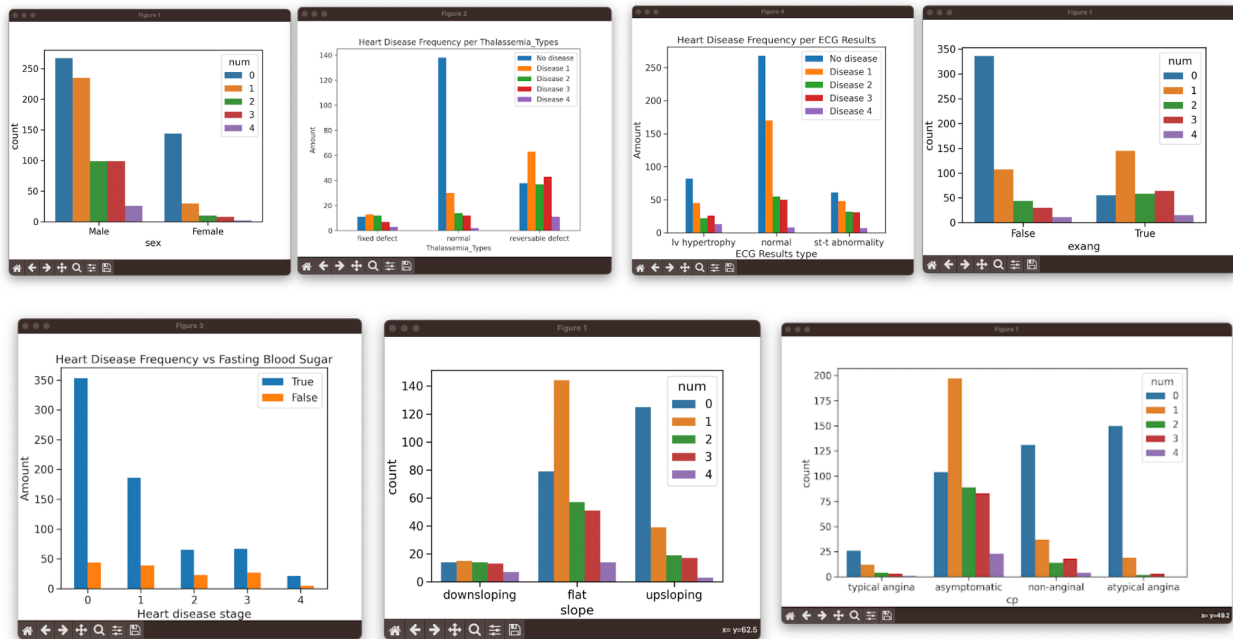


Figure 12: Multivariate plots in bar chart form of individual categorical attributes with the target “num”

Based on the multivariate plots, there are a few inferences that can be made. The dataset and id attributes were removed as they were found to be no strong correlation with the target attribute. Additionally, the “hd” attribute that was previously created, is not used in the further ML processing steps as we only require one target attribute to be studied, which is the “num” attribute that convey more information, which is the presence of heart disease as well as its degree or level in a particular patient.

Only thalch (maximum heart rate) and cholesterol has a negative correlation with HeartDisease, while the majority of features have a moderate to strong positive correlation (more than 0.1) with HeartDisease. According to correlation studies, there is a strong positive correlation between heart disease and the type of chest pain, the maximum heart rate, and the slope_peak_exercise_ST. However, there is a significant inverse relationship between heart disease and the number of major vessels, ST depression, and exercise-induced angina. Maximum heart rate, type of chest pain, and slope-peak exercise ST are strongly correlated with heart disease in men, whereas number of major vessels, ST depression, and exercise-induced angina are strongly inversely correlated. For women, however, there is a strong inverse relationship between heart disease and the number of major vessels, ST depression, and

exercise-induced angina. Nevertheless, overall, from the dataset, it is seen that males are more susceptible to heart disease than females.

Looking at the frequency plot of ECG results type against different heart disease levels, st-t abnormality and lv-hypertrophy, which are conditions denoting non-normal heartbeat, ranging from mild symptoms to severe issues, have less patients with no disease, indicating a high likelihood of the presence of HD.

Additionally, the plot for thallium stress test against different heart disease levels, indicates patients with a normal thallium level, are more likely to not have heart diseases, signified by the blue column spike in that group. Whereas patients with a thal value equivalent to fixed defect, which means there used to be a defect in this department, but is currently normal, are also more likely to have heart disease.

Another attribute to watch is the chest pain type which has a strong correlation with the presence of heart disease and the heart disease level from seeing its bar chart plot. The majority of patients with heart disease are discovered to have asymptomatic chest pain. These individuals may exhibit unusual symptoms like indigestion, the flu, or a pulled chest muscle. As with any heart attack, an asymptomatic attack involves a blockage of the blood supply to your heart and potential heart muscle damage. The same risk factors apply to heart attacks without symptoms as they do to heart attacks with symptoms. An asymptomatic heart attack increases your risk of another, potentially fatal heart attack. Your risk of complications, such as heart failure, increases if you experience another heart attack. There is no test that can predict whether you will experience an asymptomatic heart attack. An electrocardiogram or echocardiogram is the only way to determine whether you had an asymptomatic attack which can reveal alterations that denote a heart attack.

For the exang attribute, exercise-induced angina, patients with the value of false, significantly have less chance of developing heart disease than patients with a value of true for exercise-induced angina, signified by the spike of the blue bar in the “false” group.

Looking at the slope attribute or the slope of the peak exercise ST segment, patients with a downsloping, which are typical signs of an unhealthy heart, are more likely to have heart disease than people with slope value equal to upsloping, indicating better heart rate with exercise, or flatsloping which indicates minimal change or a typical healthy heart. This can be seen with the very low blue bar height with the downsloping class, indicating a higher percentage or likelihood of heart disease presence. Similarly, ST depression or the oldpeak attribute is also a good factor contributing to heart health, where patients with a lower ST depression, results in more likelihood to develop heart disease, hence a strong positive correlation can be seen in the heatmap and correlation bar graph.

For the ca attribute or the number of major vessels coloured by fluoroscopy, as there are more blood vessels and more blood movement to the heart, indicates a patient with a better heart health. Hence, patients with a ca value equivalent to 0 have more probability to develop or possess heart disease.

For the resting blood pressure attribute, values exceeding 130 and 140 mm Hg are a typical cause for concern for heart disease development. Similarly, serum cholesterol levels exceeding 200 mg/dl are also typically strong factors for heart disease.

Heart disease is very prevalent in seniors, defined as those who are 60 years of age and older, as well as among adults who fall within the age range of 41 to 60. However, it's uncommon among people aged 19 to 40 and extremely uncommon among those aged 0 to 18, as there are no records of such patients with heart disease in the dataset.

Overall, these are strong features that can be prioritised to be used in the following ML processing. We must modify the features in the following data preprocessing step, once we have obtained the insights from the data in order for the model-building phase to proceed.

3.4.3 Data Cleaning

As for data preparation, this involves preparing the initial raw data sourced directly from the data source into a final data set that can be utilised for the subsequent phases of processing. Here, the collected datasets may contain faults, missing/incomplete information, obsolete/redundancies, noise/outliers, values in a form not suitable for processing or not consistent with the context or problem domain, and other problems that can affect the training and testing process of the ML model. Noisy data is essentially, random errors present in the dataset, whether it is data that is duplicated, incorrectly entered, incorrectly processed, or outliers. The latter entails extreme values that can be difficult for the program to handle, affecting the central tendency of the data distribution in an attribute.

Dropping Unnecessary Attributes in the Dataset

The first step performed for this section, dropping the attributes that are unnecessary to be used for the further ML processing. Thus, from the previous section, it is determined that the id, dataset, and HD attributes are no longer required, as they do not significantly impact the ML processing as they have a weak correlation with the target “num”. Although HD has a strong positive correlation with HD, it is removed because HD is another target attribute, but conveys less information than the num attribute, as mentioned previously, num conveys the presence and level of heart disease, whereas HD only describes the former.

Partitioning the Dataset

Following that, the data is partitioned into 2 sections, the target attribute, y which consists of the num attribute and the input attribute, xtr, which consists of the other attributes in the dataset.

For this, the integer-location-based indexing ‘iloc’ pandas DataFrame method is utilised on the dataset where the first 13 columns are separated as the x or xtr subset, and the last 14th column is utilised as the target variable. This will be further split later on, into a training and testing dataset used to build and validate our ML models.

Converting Categorical Attributes into Numerical

After that, it is found that some attributes in the dataset have values in a string format, which may be difficult to be interpreted by the program and the ML models. Thus, these object/categorical datatype values need to be converted into an integer and float datatype. For

this two methods were tested namely the scikit-learn preprocessing method, LabelEncoder, and the pandas DataFrame method, get_dummies.

The former was applied using a for loop, so that it only applies to a column with categorical or object datatype values, and it essentially performs integer encoding, by converting each value in that column into a numerical value, based on the classes within that attribute, hence converted to 1 attribute, with the classes being labelled by 0 to the n_class value. Utilizing numbers has the drawback of introducing comparison and relation between them, and that it can be misinterpreted by the ML algorithms as having a hierarchy or order between the classes. Additionally, it is only a preferable option when there are numerous classes within a particular categorical attribute, which is not the case as previously analysed with the current dataset. In fact, even in the official documentation for the LabelEncoder method in scikit-learn, this method is only recommended to be applied to the target attribute.

The latter method, df.get_dummies(), is a one-hot encoding method, that aims to alleviate the downside of the previous LabelEncoder method, by converting the categorical attributes into a number of new columns, based on the number of classes, and assigned a binary value of 1 (True) or 0 (False) to the new column. Therefore, no ordinal relationship exists in between the newly formed attributes or classes within a parent attribute which typically results in the ML models performing better as they can understand and interpret the input attributes more correctly and accurately. The only downside to this method is that it creates too many new columns after the one-hot encoding process. However, after analysing our dataset, this is found to be not a significant issue, as there are only a few classes (2-5), for the categorical attributes in the dataset. Additionally, to support this assumption, both the different preprocessing methods were implemented, applied to the dataset, and ran through the ML models to understand the differences in the final model validation results, and whether they had an impact or effect on the results. As previously assumed, the df.get_dummies() method was seen to be returning generally better final evaluation results of the ML models, as compared to when the LabelEncoder method was implemented, so, the former was implemented and used for further processing.

Replacing Missing Values in the Dataset, Imputation

From the previous, of understanding the dataset and viewing its missing values, we found that certain attributes contain missing values. So for this, the chosen method is imputation, a common way this is implemented is to impute or replace the missing values, rather than removing those records. For this, the scikit-learn impute method, KNNImputer was used which replaces missing values with a value that is found using the k-Nearest Neighbours method. Thus, the mean value from the n_neighbors nearest neighbours found in the training set is used to impute the missing values for each sample. Therefore, this method is more preferable compared to replacing the missing value with the mode or mean of the dataset, as the missing values are replaced with the mean/average value from the n records that are nearest to it.

The parameters tested and found to be optimal for this KNNImputer function is the n_neighbors parameter at its default value, 5. Thus, it was implemented and applied to the dataset, and afterwards, the null values in the dataset are identified again, using the previous df.isnull() method, and it is found that the dataset no longer contains any missing values, so the imputation was done successfully.

Using SMOTE as Resampling Technique to Handle Imbalanced Data

This following preprocessing step was found to have a positive impact on the final evaluation results from validating the ML models, thus it was kept in the final program. Essentially it was found that some of the attributes in the dataset contained slight imbalances, thus to treat this, resampling techniques can be employed to handle imbalanced data.

Class imbalance occurs when there is an unequal distribution of classes in a dataset, meaning that there are significantly more data points in the negative class (majority class) than in the positive class (minority class). The classifier model's performance will suffer if the skewed data is not corrected beforehand.

One of the most frequently chosen methods for dealing with an unbalanced dataset is resampling the data. For this, there are primarily two types of methods: undersampling and oversampling. Oversampling methods are generally preferred to undersampling ones. The reason is that when we undersample, we frequently leave out instances from the data that could contain crucial information. Therefore, for our implementation the Synthetic Minority Oversampling Technique (SMOTE) method of the imblearn oversampling library is utilised

carries out data augmentation by constructing artificial data points based on the original data points. SMOTE can be viewed as an improved form of oversampling or as a particular data augmentation algorithm. With SMOTE, you avoid producing duplicate data points and instead produce synthetic data points that are marginally different from the original data points. It is an oversampling technique where synthetic samples are created for the minority classes, by performing oversampling in a structured manner. This overcomes the overfitting problem caused by otherwise random oversampling methods.

The operation of the SMOTE algorithm can be summarised as balancing the dataset by slightly moving the data point closer to its neighbour, following the k-nearest neighbour method. By doing this, it ensures that the synthetic data points do not exactly duplicate an existing data point, and that it also does not deviate significantly from the known observations in your minority class.

Therefore, the SMOTE method is first instantiated with the sampling strategy being the default “not majority”, which resamples all classes but the majority class, then fitten onto the previously partitioned, x and y datasets. After that the distribution of the target attribute is viewed, and it is found that the data is more balanced after the action of SMOTE.

Data Scaling Using Min-Max Normalisation

From previously analysing the numerical attributes of the dataset, it is found that different numerical attributes have a different range, and this needs to be standardised or normalised before being used before fitting with the ML models. For this, the min-max normalisation was found to be producing the ML models with the best evaluation metric results. Through this method, the features will be transformed into a range between 0 and 1, where the minimum and maximum value for a particular feature being 0 and 1, respectively. The data in the dataset is scaled to a specific range, based on the attribute's minimum and maximum value. This is

opposed to standard scaling where data are scaled based on standard normal distribution (mean = 0 and standard deviation = 1). Thus, keeping to the context of the dataset.

So, this scaled value proportionate to the range of the feature, can be calculated using the following formula:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Figure 13: The mathematical formulation for the min-max scaling

The importance of this normalisation/standardisation method is seen as the variables prior to scaling is measured to be at different scales, and so individually they do not contribute equally to the model fitting, leading to bias where one attribute might influence the ML model's decision making more than another. Additionally, scaling the dataset in this manner can aid in enhancing the speed effectiveness and efficiency of algorithm execution. As a result of the data's already-reduced size, complicated calculations, which are primarily needed to improve algorithms, take less time.

When input features are scaled (or generally scaled) instead of using the original, unscaled data, some ML models, such as the Multi-layer Perceptrons (MLP) or ANN, may benefit greatly from this. In these cases, the back-propagation may be more stable and even faster. While scaling is typically not a factor in tree-based models, it is frequently a major factor in non-tree models like SVM, LDA, etc.

Therefore, to handle the issue of feature-wise normalisation, the MinMaxScaler method of the scikit-learn preprocessing library is utilised, by first instantiating it and then fitting it with the input variables of the dataset, which computes the minimum and maximum to be used for further scaling and transformation of the dataset.

After that, this fitted min-max scaler is pickled and stored using the joblib or pickle library method, `joblib.dump()`, which is used to serialise or save the internal state of the scaler locally so that it can be easily called when required later on, primarily in the web app implementation. This is because otherwise, it would be time-consuming to retrain a new min-max scaler which can delay the operation of the overall ML model computing, especially considering this ML model being deployed in a web app format to return real-time results from input variables. Thus, it is crucial to eliminate such time delay, in this case, by preserving the scaler to be reused.

Following that, the previously fitted scaler is used to transform the dataset's input variables where it scales features of the x or input attribute dataset according to the previously computed feature range.

To visualise the effects of the preprocessing methods performed thus far, a statistical description of the dataset is generated using the `df.describe()` method. From here, we can see that the attributes of the dataset follow a consistent scale or range and so it is suitable to fit with the ML models and be used for further processing.

3.4.4 Dataset Splitting

After the dataset has been prepared, it needs to split into two parts, the training set which builds the predictive models and the validation/testing set which is used to evaluate or test the ability of the developed model to be generalised or its ability to classify inputs correctly. For instance if the chosen data split ratio is 80:20, 80% of the data is used to train the ML models, whereas 20% is used to evaluate the ML models.

It is important to ensure a split proportion where underfitting and overfitting does not occur or the accuracy values or results for both sets are similar. Underfitting occurs when the performance and results of the training set and test set are low typically because there are insufficient records to train the model. Overfitting occurs when the training set results are significantly superior to the testing set, as the ML model is too tailored or suited to the training set and performs poorly on new data so it is not generalised enough to be used in real-world cases.

Therefore, the effect of different data partitioning ratios on the performance of the ML model was tested to find the best combination with the least underfitting and overfitting. The tested split proportions include 50:50, 60:40, 70:30, 80:20, and 90:10.

This data partitioning method was implemented using the `train_test_split()` function of the Scikit-learn library, which will return a randomly assigned training and testing subsets for the x and y datasets to be inserted into the ML model or for validating the models (total 4 returned datasets), based on the entered transformed x and y datasets, according to the testing dataset size. So, by entering 0.2 as the `test_size`, the split proportion of training and testing dataset will be 80:20.

3.4.5 Model Creation (Hyperparameter Tuning and Model Development and Training)

At this stage, we design and build our ML models, using the aforementioned algorithms in the literature review (2.1.1-2.1.9) along with the XGBoost, ExtraTrees and StackingClassifier algorithm, to be fit and tested with the prepared dataset and evaluate their performance. At the initial stage, it is difficult to pinpoint which algorithm will perform the best for this problem domain as when reviewed independently, no single model is superior to another, and it depends on the problem domain and the dataset being fitted.

The list of tested and developed algorithms displays a good blend of simple linear (e.g., logistic regression, XGBoost) and nonlinear algorithms (e.g., K-Means Clustering, Decision Trees, Random Forest, ExtraTrees, KNN, Naive Bayes, SVM) as well as deep learning methods (e.g., ANN, RNN). Apart from applying a singular ML algorithm to the dataset, hybrid models are designed, developed using the StackingClassifier method of the scikit-learn library, fitted and tested with the dataset, which is essentially a combination or ensemble of several classification algorithms as well as feature selection techniques in order to increase the prediction accuracy and performance. Therefore, each of these linear and non-linear models will be developed using the corresponding Scikit-Learn method, except for the XGBoost classifier, which is from the XGBoost library.

First, the `compare_models` array is initialised, which will be used for the later stage of comparing the models that have been developed and their validation results or evaluation metrics to

determine the best-performing model. In total, there are over 17 machine-learning models (11 different ML algorithms) that are finally constructed, tested and compared.

Structure of the Model Creation and Validation Process

The model creation step for each of these models follows a similar structure or workflow, to ensure consistency and the final results are comparable.

I. Hyperparameter Optimisation

This pipeline consists of first, printing the name of the model that is being developed. Then, begins the hyperparameter optimisation or tuning step to improve the model's prediction accuracy and other evaluation metric results. This is done through finding the ideal combination of parameter values that optimise or results in a model that has the best performance.

A hyperparameter is a parameter which possesses a value that is utilised to control the training process of the ML model. Hence, hyperparameter optimisation is the process of selecting optimal hyperparameters or the right combination of hyperparameters for model training that maximises the model performance [40]. Therefore, the value of a model's hyperparameters has a significant impact on its performance. Additionally, there is no way to know in advance the best values for hyperparameters; therefore, in theory, we should try every possible value before settling on the best ones.

One such way of finding the optimal hyperparameter combination is through the grid search method, where a grid of possible hyperparameter values are created, and at every iteration, a combination of hyperparameters are trialled in a specific orientation [37]. It fits the model at all the possible hyperparameter combinations while keeping track of the performance, and finally, the best model or best performant hyperparameter combination is returned. Hence, we use the grid search method to automate the tuning of hyperparameters because doing so manually could require a significant amount of time and resources. Additionally, this method performs the hyperparameter testing of the models in a systematic manner, returning the best performing parameter values after the round of testing, and to be used in the estimator and ML model.

Thus, in this program, the grid search method of hyperparameter optimisation is implemented using the Scikit-Learn library's GridSearchCV function that executes the exhaustive search for every parameter set/combination in the grid for a particular estimator or ML model. This was chosen over the RandomizedSearchCV. This is because with GridSearchCV, it attempts every combination from the list of current hyper-parameter values and chooses the best combination based on the cross-validation score. So, the downside is the fitting process requires a lot of time because all possible combinations are tested, however, GridSearchCV is guaranteed to give us the best hyper-parameters.

In contrast, RandomizedSearchCV tries a wide range of random value combinations, and requires the quantity of iterations to be declared. The downside is it cannot guarantee to return the best parameter combination because not all parameter values are tested, thus it is the suggested choice for large datasets which has a high number of parameters to tune, and certain combinations can be skipped. It excels at evaluating a broad range of values and typically finds a

very good combination quickly. However, looking at this dataset and problem domain, the dataset is manageable, in that it doesn't have too many records or attributes, and the number of parameters to be tested for the estimators have been defined and are not overwhelming to be tested as well with the currently possessed hardware and resources. Overall, this proves why GridSearchCV is the chosen mode of testing hyperparameters in this implementation.

Therefore, returning to the structure of the model creation, the pipeline, then initialises the parameters dictionary, which consists of the parameters available for the particular model or estimator that is being studied or built, and their possible values. For instance, for the SVC or support vector machine method of the scikit-learn library, we chose to test the different possible values for the kernel, C or regularisation strength, and the gamma parameters. In the source code, you will see a second dictionary of parameters for every ML model that was designed, with parameter fields having 1 value each. These parameters contain the most optimal values that have been identified after the exhaustive operation of the GridSearchCV object. So, they are hardcoded here simply for testing and validation purposes of the designed ML models, to speed up its operation and execution of the GridSearchCV object, by narrowing down the values to the most optimal ones that were previously identified, so the GridSearchCV does not have to begin searching for the best parameter values all over again, as it is a rather time-consuming process, if one simply only wants to run the program and see the performance metric values. If one wishes to view the operation of hyperparameter tuning and to find the optimal hyperparameter values again for a particular ML model, they can simply comment or remove the second parameter list.

The estimator to be used with the GridSearchCV object is initialised. This depends on which estimator is currently being constructed and tested. No parameters are initialised for this estimator object, as it is like a control subject, where the different parameters will be tested on it, to determine the best combination. Then, the GridSearchCV object is initialised with the inputs being the instantiated estimator, then the parameters dictionary, and then the verbose parameter with value 2, which controls the verbosity or the amount of messages returned at each test run. The GridSearchCV is then fitted with the training dataset, and this fit action will be repeated for every set of the different parameter value combinations.

Once all the parameter combinations have been tested, the best-performing parameters are extracted, identified and displayed, using the `clf.best_params` statement.

II. Model Initialisation, Fitting and Predicting

The next step of the pipeline or building the ML model is initialising the ML estimator again using the previously found best-performing hyperparameters, fitting it with the training dataset, using it to make predictions, and evaluating the performance of the models. Hence, first, similar to before, using the corresponding scikit-learn method or object, the current ML estimator is initialised with the best-performing parameters identified before, as the inputs. Then, the `fit()` function is used to perform the model training, and the `predict()` function will be used together with the testing set to observe the performance of the models, and to test for overfitting or underfitting.

Since we already know whether each patient has heart disease, we can use the data that is currently available to train our prediction model. This procedure is also referred to as learning and supervision. Then, the trained model is applied to predict whether users have heart disease.

During this, the duration or time taken to train the model and use the model for testing is tracked, using the `time.time()` method.

III. Model Testing

Towards the end of the model creation structure, the `modelValidation()` function is called, which is a user-defined function that takes in the value of the testing subset of the `y` or target attribute of the dataset, the predicted `y` or target attribute (result of `model.predict()`), the tracked time, and the name of the current estimator being studied. This function, contains the various ML model evaluation tests from the scikit-learn metrics library, including `classification_report`, `accuracy_score`, `precision_score`, `recall_score`, `f1_score`, `mean_squared_error`, `mean_absolute_error`, `matthews_corrcoef`, and the user-defined methods to calculate the root mean square error, misclassification rate, the training, testing and total time. Hence, all these values are computed and then together with the model name, are appended to the previously defined `compare_models` array, which will then be converted into a Pandas DataFrame, allowing for easier visualisation and comparison of the different models and their evaluation results. It also makes it easier to export the Pandas DataFrame into an excel file (.xlsx) for easier reading and analysing.

Machine Learning Models That Were Built and Tested

This section provides a brief description of the structure, workflow and implementation for each of the machine learning models that were developed and tested with the dataset. Additionally, the different hyperparameters that were tested for each of the ML models are explained.

I. Support Vector Machine, SVC

For the Support Vector Machine (SVC) model, the SVC object of the scikit-learn svm library. For more information, the operation and a general background of SVCs are explored in the literature review section (2.1.5).

Briefly put, how SVCs of the scikit-learn library functions is by creating a hyperplane to divide the various classes in a multidimensional space. Then, error minimization is accomplished by SVM, which iteratively generates the optimal hyperplane. Overall, the main goal of SVM is to identify the maximum marginal hyperplane (MMH) that best classifies the dataset.

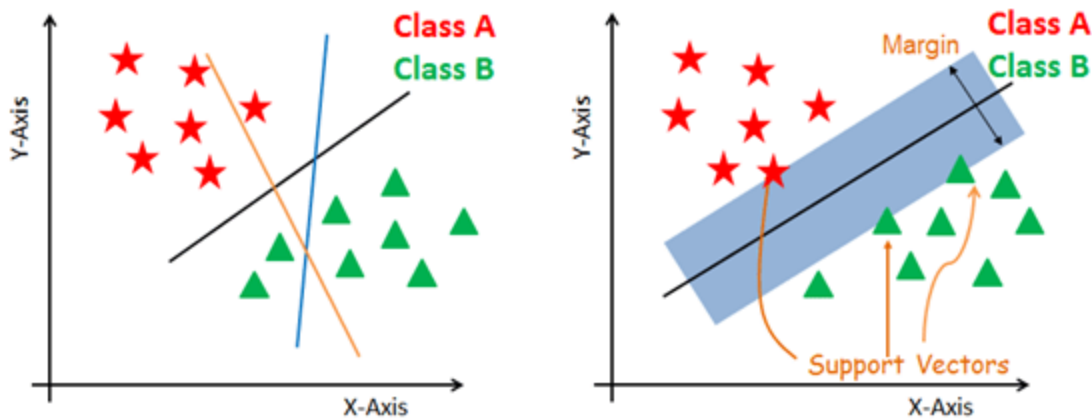


Figure 14: Support Vector Machine hyperplane identification

There are three components to this. Firstly the support vectors, data points which are closest to the hyperplane. By using margin calculations, these points will more clearly define the dividing line. A decision plane called a hyperplane divides up a collection of objects into different classes. Thirdly a margin is the distance between the two lines on the closest class points. This is calculated as the angle between the line and the nearest points or support vectors. A larger gap between the classes is regarded as a good gap; a smaller gap is regarded as a bad gap.

Overall, the goal is to choose a hyperplane in the given dataset that has the largest margin between support vectors.

There are three parameters of the SVC object that can be manipulated, and so are tested to find the optimal values that result in the highest evaluation results, the kernel, C or regularisation and gamma parameters.

Kernel

The kernel's primary job is to change the input data from the given dataset into the necessary form. Hence, there are four functions that are tested in our implementation, including radial basis function (RBF), polynomial, linear, and sigmoid. A non-linear hyperplane such as the one in our problem domain, can benefit from polynomial and RBF. A separation line in a higher dimension is preferably calculated using polynomial and RBF kernels. Additionally, it is advisable to use a more complex kernel in some applications to separate the curvy or nonlinear classes. Through this, improved classifiers may result from this transformation.

Regularization

For scikit-learn objects, the regularization parameter is represented by C, which is used to maintain regularization. Here, C stands for the penalty parameter, which stands for the misclassification or error term. The possible values that are tested for this include 0.1, 0.5, 1, 2, 5, 10, 20. The SVM optimization is informed by the misclassification or error term regarding the acceptable level of error. With this, you can manage the trade-off between the decision boundary and the misclassification term. Hence, larger values of C result in larger-margin hyperplanes, while smaller values of C result in smaller-margin hyperplanes.

Gamma

A lower gamma value results in a loose or generalised fit to the training dataset, whereas a higher gamma value results in an exact fit, leading to over-fitting. In other words, you could say that a low gamma value only takes into account nearby points when determining the separation line, whereas a high gamma value takes into account all of the data points. The tested gamma values were 0.001, 0.01, 0.1, 0.25, 0.5, 0.75, and 1.

Hence, based on the GridSearchCV operation, it was found that the best performing hyperparameters was the rbf kernel, 1 value for gamma, and 20 value for the regularization.

II. K-Nearest Neighbours, KNN

For the K-Nearest Neighbours, (KNN) model, the KNeighborsClassifier object of the scikit-learn neighbors library was utilised for its implementation. For more information, the operation and a general background of KNNs are explored in the literature review section (2.1.6).

The KNeighborsClassifier object functions by finding the K closest points to a particular record based on the euclidean distance formula, and then classifying it based on the majority in the K-nearest neighbouring points. Therefore, the number of neighbors or K, is a key parameter in this model for determining the performance of the model. This K value is typically an odd number to allow for a majority to be found within the class labels to be assigned to the current point to be classified.

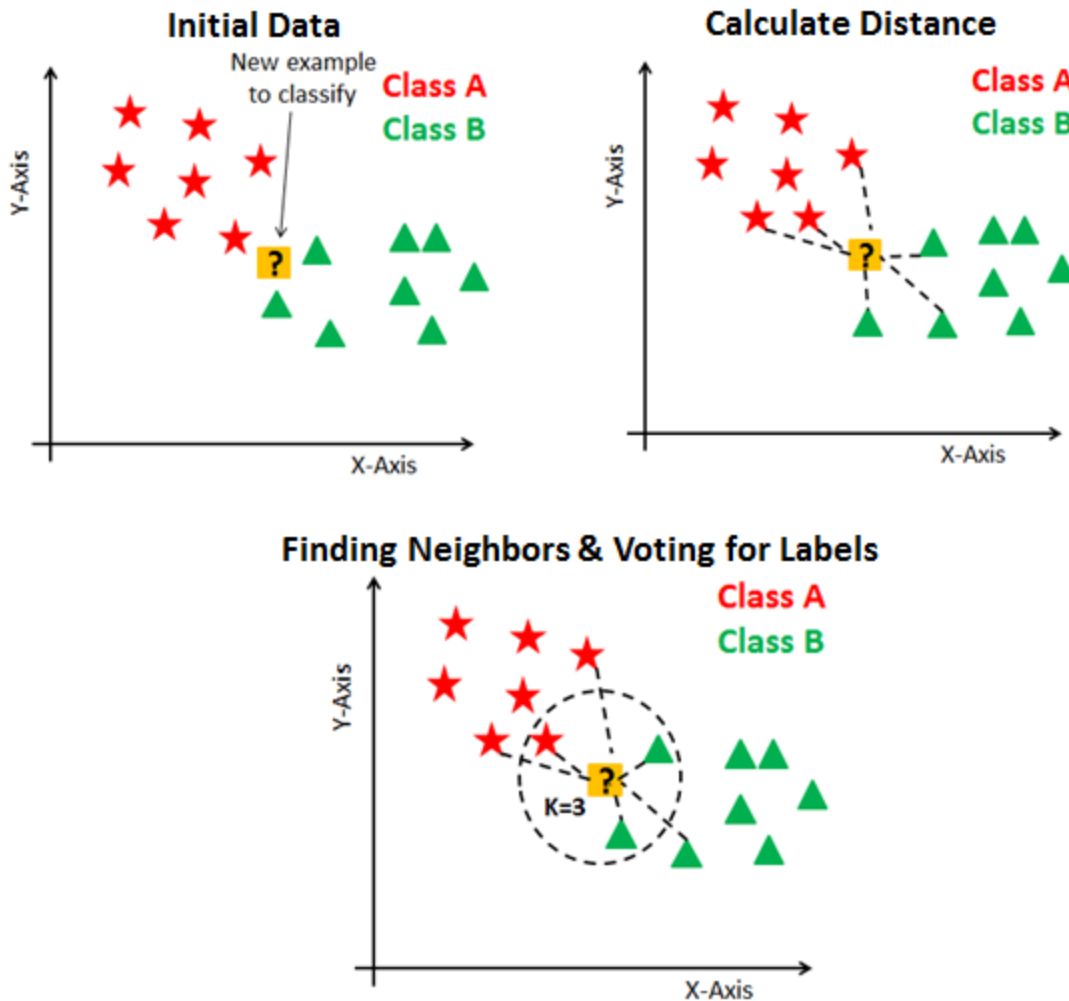


Figure 15: K-Nearest Neighbour algorithm operation

Therefore, there are two parameters of the `KNeighborsClassifier` object that can be manipulated, the kernel, `n_neighbors` or the number of neighbours (K) and the weights.

Number of Neighbours, K

As mentioned earlier the `n_neighbors` value denotes the number of neighbouring records to consider, when determining the label of the current record. So, this value is typically an odd number to allow for the majority to be from the same class and the number of points from two class being split evenly. So, for this, the values that were tested include 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, and 29.

Weights

The weight parameter specifies the weight function used in the prediction, whether there is any distinction between any points, respective to weight or importance. Thus, this can be set to “uniform” and so all points will possess equal weights or importance. Otherwise, it can be set to “distance” where the weights are inversely proportional to the points’ distance to that neighbour. So, the closer a neighbour is to a particular selected point, this will have a greater influence than

neighbours that are further away. So, for this parameter, those were the 2 values that were tested, uniform and distance.

Hence, based on the GridSearchCV operation, it was found that the best performing hyperparameters were the 3 neighbours and weights value equivalent to the distance.

III. Naive Bayes, NB

For the Naive Bayes, (NB) model, the GaussianNB object of the scikit-learn naive_bayes library was utilised for its implementation. For more information, the operation and a general background of NBs are explored in the literature review section (2.1.3).

The GaussianNB object functions by calculating the probability of an event using the Bayes Theorem. The formula for this is explored in more depth in the literature review section. So, it starts by calculating the prior probability for a given number of class labels. Then, it finds the probability for each attribute for every class. It enters these values into the Bayes formula to calculate the posterior probability, and finally places the input point into the higher probability class.

The GaussianNB object uses Gaussian Naive Bayes, which is useful when dealing with continuous attribute values where the probabilities are modelled with a Gaussian distribution, such as with the following function:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The conditional probabilities $p(x)$ are gaussian distributed so the parameters variance and mean are estimated using the maximum likelihood approach.

For GaussianNB, there is only one parameter of to be manipulated and tested, which is the 'var_smoothing' attribute. This denotes the portion of the largest variance for all features that are summed with the variances for calculation stability. Therefore, this attribute is used for stability calculation either to widen or smooth the curve, to allow for more samples that are farther distributed from the distribution's mean. So, for this the np.logspace statement is used to return a sequence of numbers that are spaced evenly on the logarithmic scale, beginning from 0 till -9, generating a total of 100 samples, np.logspace(0,-9, num=100).

Hence, based on the GridSearchCV operation, it was found that the best performing hyperparameter was the var_smoothing value equivalent to 0.5336699.

IV. Decision Trees, DT

For the Decision Tree, (DT) model, the DecisionTreeClassifier object of the scikit-learn tree library was utilised for its implementation. For more information, the operation and a general background of DTs are explored in the literature review section (2.1.2).

The DecisionTreeClassifier operates by recursively selecting the best feature or attribute that separates the records at every stage or tree depth, designating that feature as the decision node, which further splits the dataset into subsets. It performs this until one of the following termination conditions are achieved, either all tuples belong to the same attribute value, there are no remaining attributes to utilise, there are no more instances, or the maximum depth has been reached. The decision tree uses Attribute Selection Measures (ASM), a heuristic to determine the splitting criterion or the best attribute to split the dataset in the best possible manner. ASM ranks the features in a dataset, and the best score attribute will be selected as the splitting first.

Therefore, this is the first hyperparameter of the scikit-learn DecisionTreeClassifier object, the criterion parameter, which is the function to measure the quality of the split. Based on this, the best score feature will be selected as the splitting criteria or attribute. The possible values for this include “gini” for measure based on Gini impurity and “log_loss” and “entropy” for measures based on the Shannon information gain. Firstly the Gini index or impurity is for the decision trees algorithm CART (Classification and Regression Tree), where the Gini index considers a binary split for each attribute after computing the weighted sum of impurity of each partition. Finally, the attribute with the least Gini index is chosen as the splitting attribute first.

Whereas for the “log_loss” and “entropy” parameters, the Shannon information gain is based on the concept of entropy that measures the randomness or impurity in an input set. So, where there is more information gain there is a decrease in entropy. Essentially information gain calculates the entropy value before and after a split of a dataset with an attribute value. Thus, this is useful for ID3 decision tree algorithms. How log loss differentiates from this is it considers the log probability of the point being in a particular class to the computation.

The second parameter being tested is the max_depth parameter which specifies the maximum depth the tree can be expanded to. If the value is chosen to be None, the nodes of the tree will be expanded until all the leaves are pure or possess less than the min_samples_split value. Otherwise, any number specified here, will be the max depth to which the tree will reach in depth. For this the values that are tested include None, 4, 5, 6, 7, 8, and 9.

Hence, based on the GridSearchCV operation, it was found that the best-performing hyperparameters were the None max depth and entropy criterion.

V. eXtreme Gradient Boosting, XGB

For the eXtreme Gradient Boosting, (XGB) model, the XGBClassifier object of the xgboost library was utilised for its implementation. The gradient-boosted decision tree (GBDT) machine learning library XGBoost, which stands for "Extreme Gradient Boosting," is scalable and distributed. It offers parallel tree boosting and is the top machine-learning library for problems involving regression, classification, and ranking.

A Gradient Boosting Decision Trees (GBDT) is a decision tree ensemble learning algorithm for classification and regression that is similar to random forest as it combines multiple machine learning algorithms, in this a combination of decision trees to attain a more accurate predictive model. What differentiates an extreme gradient boosting algorithm from an ensemble model such

as random forests is it performs “boosting” or improves a single weak model by combining it with other weak models to attain a stronger predictive model or to reduce or correct errors made by existing models. Hence, models are added sequentially, until it cannot be improved any further. The process of generating weak models is typically using the gradient descent algorithm over an objective function.

The main goal of the development of XGBoost was to boost the performance and computational speed of machine learning models. It is a scalable and highly accurate gradient boosting implementation that pushes the limits of computing power for boosted tree algorithms. Trees are constructed using XGBoost in parallel. Utilizing a level-wise approach, it assesses the quality of splits at each potential split in the training set by scanning across gradient values and using these partial sums.

On that note, there are 3 hyperparameters of the XGBClassifier object that can be manipulated and studied to identify the optimal combination with the GridSearchCV.

Firstly, the “n_estimators” or number of estimators parameter, which specifies the number of trees or rounds in the XGBoost model. We need to find the cut off point for this parameter as increasing the number of trees beyond a certain limit, generally does not enhance the model performance any further. For this, the values that were tested include 100, 150 and 200.

Secondly is the “max_depth” parameter, where similar to the previous models, this specifies the maximum depth of the trees or estimators part of the ensemble, and choosing None will expand the tree until any of the aforementioned termination conditions is reached. The values chosen for this parameter include None, 6, and 7.

Moving on, the “subsample” parameter specifies the subsample ratio of the training where the model implements bagging, a ML ensemble algorithm to improve stability and accuracy of the algorithm by performing model averaging, through subsampling at every boosting iteration. The values chosen for this parameter are calculated using `np.arange(0.05, 1.01, 0.05)`.

Hence, based on the GridSearchCV operation, it was found that the best performing hyperparameters were no. of estimators equivalent to 100, None max depth and 0.75 subsample.

VI. Random Forest, RF

For the Random Forest, (DT) model, the RandomForestClassifier object of the scikit-learn ensemble library was utilised for its implementation. For more information, the operation and a general background of RFs are explored in the literature review section (2.1.4).

A random forest is an ensemble method consisting of decision trees, generated by randomly split dataset. The individual units are decision trees, such as the ones mentioned previously, where they are constructed using attribute selection indicators, namely information gain, gain ratio and Gini index for each attribute. The scikit-learn RandomForestClassifier object functions by selecting random samples from a dataset, then building a decision tree for each sample based on the chosen criterion or attribute selection indicator and obtaining the prediction result of each

decision tree. After that, a vote is done for each of the obtained predicted result and based on the prediction result, the highest score or the one with the most votes is utilised for the final prediction or estimator. The scikit-learn RandomForestClassifier model can also be seen as a meta estimator that uses averaging to increase predictive accuracy and reduce overfitting by fitting a number of decision tree classifiers to different subsamples of the dataset.

On that note, there are 4 hyperparameters of the random forest object that can be manipulated and studied to identify the optimal combination with the GridSearchCV. Firstly, as mentioned previously for the XGBoost model, is the number of estimators or “n_estimators” parameter which specifies the maximum number of decision trees that are developed in a random forest structure. For this, the values that are tested are similar to before, which are 100, 150, and 200. Secondly, is the max_depth parameter or the maximum depth that a decision tree can reach. For this the tested values are None, 4, 5, 6, and 7. Moving on, is the criterion attribute where as previously explained for the decision trees model specifies the function used to measure the quality of a split in a decision tree, and for this the tested values are gini, entropy and log_loss. Finally, is the max_features parameter which specifies functions to calculate the number of features to consider when finding the best split for the decision trees. For this the values that are chosen include auto, which is the same as the second which is sqrt, that use the square root of the total number of features. The third being log 2 which finds the log2 of the total number of features and finally is None, which uses all the features.

Hence, based on the GridSearchCV operation, it was found that the best performing hyperparameters were no. of estimators equivalent to 200, “None” max depth, “auto” max features, and the “gini” criterion.

VII. ExtraTrees, EXT

For the Extremely Randomized Trees Classifier, (EXT) model, or ExtraTrees model the ExtraTreesClassifier object of the scikit-learn ensemble library was utilised for its implementation.

Similar to random forest, the Extremely Randomized Trees Classifier model is an ensemble learning technique, which combines the results of multiple de-correlated individual units or decision trees to be collected in a “forest” and to output a classification result. Its differentiating factor from random forest is in the manner in which the decision trees of the forest are built, where each decision tree is constructed from the original training sample than subsamples. From there, at each test node, the trees are given random samples of k features from the list of features, where the decision trees must choose the best features to split the dataset, based on the previously mentioned attribute selection indicators gini, entropy, or log_loss. Therefore, there are numerous de-correlated decision trees produced as a result of this random sample of features.

The scikit-learn ExtraTreesClassifier model can also be seen as a meta estimator that uses averaging to increase predictive accuracy and reduce overfitting. The meta estimator fits a number of randomized decision trees (also known as extra-trees) on different sub-samples of the dataset.

Similar to the random forest model, there are the same 4 hyperparameters for the ExtraTreesClassifier object that can be manipulated and studied to identify the optimal combination with the GridSearchCV. This includes the number of estimators, maximum features, maximum depth, and criterion. The parameters perform the same function as previously mentioned for the other ML models. For this implementation the number of estimators tested were 10, 20, 50, 100, and 200, whereas max_features tested were 'auto', 'sqrt', and 'log2', while max_depth tested were None, 4,5,6, and 7, and finally, the criterion tested were 'gini', 'entropy', and 'log_loss'.

Hence, based on the GridSearchCV operation, it was found that the best performing hyperparameters were no. of estimators equivalent to 200, “None” max depth, “log2” max features, and the “gini” criterion.

VIII. K-Means Clustering, KMC

For the K-Means Clustering, (KMC) model, the KMeans object of the scikit-learn cluster library was utilised for its implementation. For more information, the operation and a general background of KMCs are explored in the literature review section (2.1.9).

In short, the K-Means Clustering model is an unsupervised machine-learning technique that assigns the class for a particular point based on the points near it or the cluster it is in. Thus because of certain similarities it has with other data points, it will be categorised into a class, and so there is no target attribute that is utilised for training the model, instead, the model finds patterns in the data and assigns classes based on those similarities.

So, it starts by choosing k centroids at random, where k is the number of clusters you want to use. The centre of a cluster is represented by centroids, which are data points. The algorithm's main component operates using a two-step procedure known as expectation-maximization. Each data point is assigned to the closest centroid during the expectation step. The next step, known as maximization, establishes the new centroid by computing the mean of all the points for each cluster. After the centroids converge or follow the assignment from the previous iteration, the sum of the squared errors (SSE) is calculated to test the quality of the cluster assignments. The SSE is calculated as the sum of the squared Euclidean distances between each point and its nearest centroid. Given that this is a measure of error, k-means seeks to reduce this value. Finally, repeating the expectation-maximization step with a different set of clusters and initial centroids until the centroid positions converge and remain unchanged.

On that note, there are 3 hyperparameters of the KMeans object that can be manipulated and studied to identify the optimal combination with the GridSearchCV. Firstly is the number of clusters which denote how many centroids must be produced and how many clusters must emerge. The values tested for this are 2,3,5,10, 20, and 50. Secondly is the max_iter parameter, which specifies the maximum iteration that the k-means algorithm can execute for a single run, and the values tested for this include 50, 100, and 150. Finally, is the algorithm parameter which specifies the type of k-means algorithm to use for the operation of the model.

“lloyd” is the traditional EM-style algorithm. Utilizing the triangle inequality, the “elkan” variation can be more effective on some datasets with clearly defined clusters. Whereas the “auto” and “full” values are aliases of the “lloyd” parameter value. So, the values tested for this include “lloyd”, “elkan”, “auto” and “full”.

Hence, based on the GridSearchCV operation, it was found that the best performing hyperparameters were no. of clusters equivalent to 200, 150 maximum iterations, and the “lloyd” algorithm.

Nevertheless, based on understanding its structure and workflow, it is clear that the KMC model is found to not be optimal for this problem domain, as it is an unsupervised ML classifier. This is further supported by its evaluation metric scores which are very underwhelming and far lower than the other classifiers. This is because it does not utilise the additional information that is available in the form of the target values as with supervised models. So, it only performs classification based on patterns it finds from the dataset.

IX. Logistic Regression, LR

For the Logistic Regression, (LR) model, the LogisticRegression object of the scikit-learn linear_model library was utilised for its implementation. For more information, the operation and a general background of LRs are explored in the literature review section (2.1.1).

The scikit-learn logistic regression object is a simple classification algorithm that classifies rows into two or more classes depending on the number of classes the target values adhere to. Therefore, there are two types of logistic regression binary classification when there is only 2 possible outcomes and multi-class classification when there are more than 2 possible outcomes. Therefore, looking at our dataset, the latter explains it more accurately as the dataset has a multivalued target attribute.

The LogisticRegression object estimates the likelihood of an outcome, and the result will range from 0 to 1 (true or false). Probabilities that are less than 0.5 are classified as false in binary class classification, while probabilities that are more than 0.5 are classified as true. The natural log and sigmoid functions are thought to keep the output values at 0 or 1, respectively, by the algorithm. Hence, fitting the data to an S-shaped line is the goal of logistic regression.

Additionally, in logistic regression, the dependent variables or target attributes adhere to the Bernoulli distribution where just two possible results in the discrete probability distribution known as the Bernoulli distribution are possible.

On that note, there are 2 hyperparameters of the KMeans object that can be manipulated and studied to identify the optimal combination with the GridSearchCV. Firstly, is the regularisation value, C, which as previously described in the SVC section, stands for the penalty parameter, which stands for the misclassification or error term that denote the acceptable error rate for a particular estimator. The values tested for this are 1.0, 10.0, 100.0, and 1000.0. Additionally, the solver parameter is specifies the function or algorithm used for the optimisation of the logistic

regression model. For this, the values tested are “newton-cg”, “lbfgs”, “liblinear”, “sag”, and “saga”.

X. Artificial Neural Network, ANN or Multilayer Perceptron, MLP

For the Artificial Neural Network (ANN) or Multilayer Perceptron (MLP) model, the MLPClassifier object of the scikit-learn neural_network library was utilised for its implementation. For more information, the operation and a general background of ANNs are explored in the literature review section (2.1.7).

The MLPClassifier object of scikit-learn follows a feedforward neural network structure consisting of three layers, the input, output and hidden layer and consists of interconnected nodes within the layers. So an MLP joins multiple layers in a directed graph, where there is only one possible signal path through each node. Except for the input nodes, every node has a nonlinear activation function.

For starters, the input layer receives data, which is then subjected to abstraction in the hidden layer. Finally, the output layer distributes the predictions, classifications or outputs. Similar to a typical unit of perceptron, all input layers are subjected to an activation function and initial weights in a weighted sum. These weights are akin to the coefficients used in a regression equation. An activation function or transfer function is a straightforward mapping of the neuron's output to its summed weighted input. Because it controls the threshold at which the neuron is activated and the intensity of the output signal, it is known as an activation function.

However, an MLP sets itself apart by using the supervised learning method of backpropagation, as the supervised learning technique for training the network. Backpropagation allows the neural network to iteratively modify its network weights to reduce its cost function. Owing to their being multiple layers in an MLP, it is considered a deep learning method. The hierarchical or multi-layered structure of neural networks is what gives them their predictive ability.

The MLPClassifier object of the scikit-learn library allows for the easy manipulation of key parameters or hyperparameters to see its effect on the resultant model's performance and evaluation metric results. Therefore, the following are the parameters of the MLPClassifier object that are tested along with the corresponding values that are tested:

1. Activation function: identity('identity'), logistic sigmoid ('logistic'), hyperbolic tan function ('tanh'), and rectified linear unit ('relu'). This parameter specifies the activation function at every neuron of the hidden layer. For this, the “identity” value is a no-op activation technique which is effective for implementing linear bottle-neck by returning a linear function based on $f(x) = x$. Whereas the “logistic” value utilises a logistic sigmoid function, based on a function that returns $f(x) = 1/(1+\exp(-x))$. Moving on, the “tanh” value is a hyperbolic tan function that, as the name suggests, returns based on the function $f(x) = \tanh(x)$. Finally, the “relu” value specifies a rectified linear unit function that returns based on the function $f(x) = \max(0, x)$.
2. Solver: Limited-memory BFGS ('lbfgs'), stochastic gradient descent ('sgd'), and stochastic gradient-based optimizer ('adam'). As in the previous model, this solver parameter specifies the weight optimisation solver across the noted, where “adam” works

effectively in terms of training time and validation score, especially on relatively large datasets. Whereas for smaller datasets, “lbfgs” performs slightly better as it converges more rapidly.

3. Hidden Layer Size (`hidden_layer_sizes`): 1 to 30 neurons per layer until 3 layers. So, (1) to (30,30,30). Meaning that at every run a new neuron is added to the current layer up until 30 neurons in that layer, and then the next layer will be created starting from 1 until 30 and so on, until 3 layers are filled with 30 neurons each. This parameter essentially describes the number of layers and neurons in that layer for the `MLPClassifier` object. It is a tuple datatype where every element of the tuple specifies the number of neurons in that layer and the index of the element and the length of the tuple denote the number of hidden layers in the `MLPClassifier`.
4. Maximum Iterations (`max_iteration`). 1 to 1000 maximum training iterations are tested. This parameter denotes the number of epochs or the maximum number of training iterations to perform and the data points will be used till the final NN model is obtained and evaluated. Therefore, typically the solver iterates until convergence is reached or there is no more increase in the performance or evaluation metrics, otherwise it will stop at this specified number of iterations. This value is important to be discovered to identify the optimal number of iterations, where performing more iterations will not improve the model’s performance and will only increase its complexity or time taken to train the model. So, an optimal value will maximise the model’s performance in an acceptable training timeframe and minimal model complexity.

There are more outside available values for the activation function and weight optimisation solver in the real-world that can optimise the NN models, however, for simplicity of testing, only the available values for these parameters or ones provided by default by the `MLPClassifier` object are studied and tested.

Additionally, apart from using the scikit-learn library’s object, `MLPClassifier`, the Keras library was also utilised to construct a multi-layer perceptron. For this, the `Sequential` model is created, acting as a stack of layers where hidden layers will be added to it, before compiling it with the Adam compiler. This compile is utilised to compile the model as it is able to perform stochastic gradient descent, an optimisation technique used in ML and DL, minimising the cost function or the error between the actual output and predicted output, resulting in better predicted output that is closer to the value of actual output. After that, the designed NN model is then trained with the training data, using the `fit()` function and to test its prediction accuracy with the testing data, using the `predict()` function. However, this method of developing an MLP classifier was abandoned in preference to scikit-learn’s `MLPClassifier` object. The reason for this was, from our experience it was found that the `Sequential` model of the Keras library required more time to be trained or fitted with the dataset as compared to scikit-learn’s `MLPClassifier`. Thus, this can accumulate, as we want to test the effect of different hyperparameters, especially the different activation functions, solvers, hidden layer structures and sizes and the maximum epochs. Additionally, as mentioned earlier, the `MLPClassifier` allows for easy manipulation of these parameter values. Thus, it was chosen as the final mode for building and testing ANNs or MLPs.

For testing the hyperparameters of this `MLPClassifier` object a different approach is taken compared to the other machine learning models, as `GridSearchCV` is not used. Instead, three

different tests are designed using for loops to determine the optimal values for the aforementioned hyperparameters that result in a model that performs the best. This has a similar effect as the grid search method of hyperparameter optimisation, as these for-loop based tests exhaustively assess every combination of the parameter values to determine the best-performing hyperparameters. Then, finally, after the hyperparameter values have been identified, they will be used as the parameter values when constructing the final MLPClassifier object, which will be fitted with the dataset, used to make predictions and tested to be evaluated and compared with the other ML models that have been tested thus far.

That being said, the tests include the first test is for finding the combination of activation function and solver, whereas the second is for finding the optimal number of layers and neurons, and the third is for finding the maximum iterations.

Test 1: Activation Function and Solver

For the first test, firstly, the parameters list for the activation function and solvers are initialised to specify the values for these parameters. Then, the default values for the maximum iterations and hidden layer size are initialised to 1000 and (32, 16) respectively. Therefore, as there are 4 activations and 3 solvers, there will be a total of 12 combinations to test with the MLP having 2 layers with 32 neurons in the first layer, 16 in the second, and it will be trained for a maximum iteration of 1000. This ensures that every combination reaches a convergence to a solution. During every test run, the `nn_pipeline` user-defined method is called, which essentially is a quick way to initiate the construction of the MLPClassifier object along with the parameter values for that particular test run, and to call the aforementioned `modelValidation` function to evaluate it. After every test run, some key information regarding the test run is stored in an array, namely the activation and solver used, and their accuracy, precision and f1 scores. This array is sorted in descending order of those evaluation metric scores, and then stored in a Pandas DataFrame to visualise the parameters and their corresponding performance and evaluation metric results easily, which is then printed.

The following is the visualisation of the Pandas DataFrame to identify the best performing number of hidden layers and neurons combination and their corresponding performance and evaluation metric results.

	Activation	Solver	Acc	F1-Score	Prec
0	tanh	adam	0.790754	0.790754	0.790754
1	relu	adam	0.781022	0.781022	0.781022
2	relu	lbfgs	0.766423	0.766423	0.766423
3	tanh	lbfgs	0.763990	0.763990	0.763990
4	logistic	lbfgs	0.717762	0.717762	0.717762
5	logistic	adam	0.673966	0.673966	0.673966
6	relu	sgd	0.588808	0.588808	0.588808
7	identity	adam	0.557178	0.557178	0.557178
8	tanh	sgd	0.557178	0.557178	0.557178
9	identity	lbfgs	0.549878	0.549878	0.549878
10	identity	sgd	0.532847	0.532847	0.532847
11	logistic	sgd	0.177616	0.177616	0.177616

Figure 16: Visualisation of the evaluation metric results from the testing of different activation function and solver combination with the MLPClassifier.

From here, it is found that the “tanh” activation function and “adam” solver combination performed the best, returning an accuracy result of 0.790754, an f1-score of 0.790754, and precision score of 0.790754. Thus it is the combination chosen for the subsequent tests and the final MLPClassifier object that is created, as it is the most stable and best-performing activation function and solver combination in the parameter list, outperforming every other combination.

Test 2: Number of Hidden Layers and Neurons

For the second test, first, the default values for the activation function and solver are initiated, using the best-performing hyperparameter values found from the previous test, which are the “tanh” activation function and “adam” solver. After that, the maximum iteration is initialised to 1000, as mentioned before to allow for the models to converge to a solution. Then the maximum number of hidden layers and neurons are instantiated to 3 and 30 respectively so that MLPClassifier objects with a hidden layer size of (1) to (30, 30, 30), will be created and tested. This maximum number of neurons per layer was kept the same to provide consistency and to simplify the test as well as for lower computational time, as testing a different number of neurons for every layer would lead to more complexity in the test sequence.

Hence, with this method, in total there will be 90 total test runs. So, a nested for loop is used for the first looping through the max number of hidden layers and the child for loop for looping through the maximum number of neurons to achieve this sequence. Similar to the previous test, the nn_pipeline method is called at every test run to make it easier to initialise the MLPClassifier with the parameters based on what is passed in that test run. The parameters along with their evaluation metric scores are stored in an array and sorted in descending order of their evaluation metric scores (accuracy, f1-score and precision). After that, the array is stored in a Pandas DataFrame to visualise the parameters and their corresponding performance and evaluation metric results easily, which is then printed. Along with that, at every layer iteration, the

evaluation metrics for the number of neurons for that number of hidden layers is stored to be visualised using the Matplotlib method plot. This is to visualise the effect of increasing the number of neurons at every hidden layer stage on the performance of the model. So, the following are the line graph plots at the end of each hidden layer for loop.

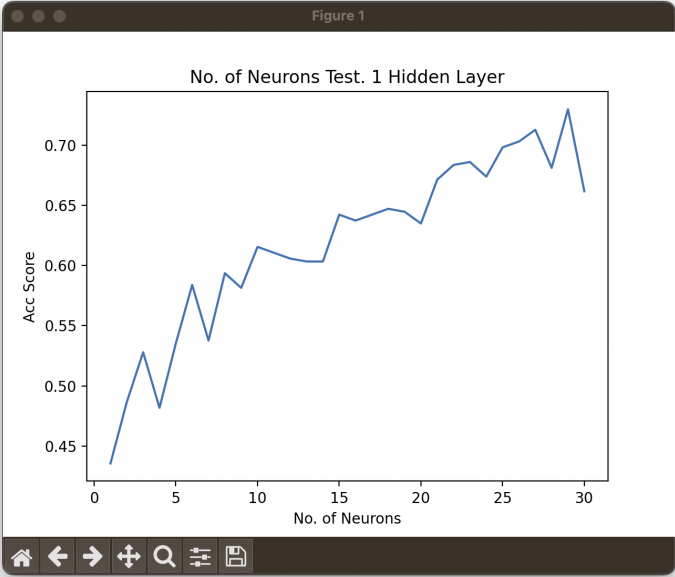


Figure 17: Accuracy score for MLPClassifier model with 1 hidden layer and different number of neurons.

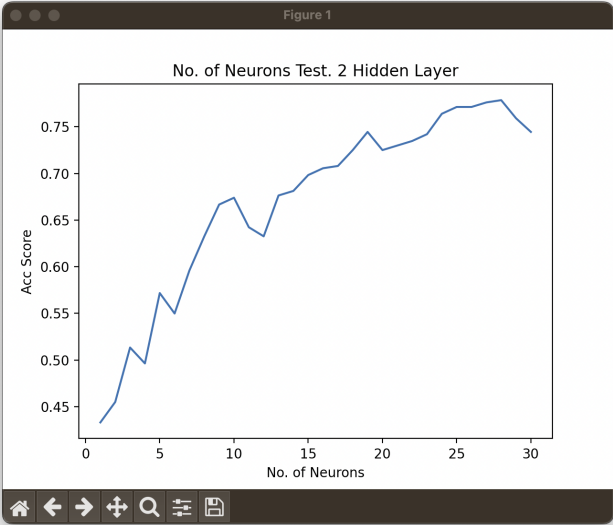


Figure 18: Accuracy score for MLPClassifier model with 2 hidden layers and different number of neurons.

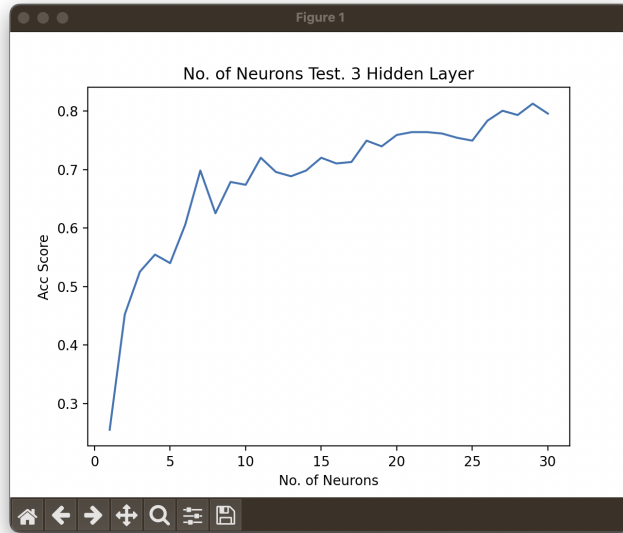


Figure 19: Accuracy score for MLPClassifier model with 3 hidden layers and different number of neurons.

From the diagrams above it is observed that at every hidden layer, increasing the number of neurons improves the accuracy of the model until it stabilises as it approaches values exceeding 25. This demonstrates that an increase in the number of neurons enhances the MLP's performance. When the number of hidden layers increases, the accuracy values stabilises at an earlier number of neurons. The two hidden layers, 1, and 2, exhibit the greatest fluctuations and potentially volatile behavior, thus 3 hidden layers are preferable for the final chosen hidden layer combination.

The following is the visualisation of the Pandas DataFrame to identify the best performing number of hidden layers and neurons combination and their corresponding performance and evaluation metric results.

	No. of Hidden Layers	No. of Neurons	Hidden Layers	Acc	F1-Score	Prec	Training Time (s)	Testing Time (s)	Total Time (s)
0	3	27	(27, 27, 27)	0.812652	0.812652	0.812652	2.421825	0.001957	2.423782
1	3	26	(26, 26, 26)	0.795620	0.795620	0.795620	2.754163	0.008283	2.762446
2	3	22	(22, 22, 22)	0.793187	0.793187	0.793187	2.485826	0.000324	2.486150
3	3	30	(30, 30, 30)	0.785888	0.785888	0.785888	3.087715	0.003930	3.091645
4	3	18	(18, 18, 18)	0.778589	0.778589	0.778589	2.149912	0.000285	2.150197
5	3	24	(24, 24, 24)	0.778589	0.778589	0.778589	2.533557	0.000319	2.533876
6	3	23	(23, 23, 23)	0.776156	0.776156	0.776156	2.620560	0.000317	2.620877
7	2	25	(25, 25)	0.773723	0.773723	0.773723	1.965197	0.000266	1.965463
8	3	29	(29, 29, 29)	0.773723	0.773723	0.773723	2.990791	0.014269	3.005060
9	2	29	(29, 29)	0.771290	0.771290	0.771290	2.183536	0.000622	2.184158

Figure 20: Visualisation of the evaluation metric results from the testing of different hidden layer combinations with the MLPClassifier

From here, it is found that the (27, 27, 27) hidden layer combination performed the best, returning an accuracy result of 0.812652, an f1-score of 0.812652, and precision score of 0.812652, training time of 2.421825s, testing time of 0.001957s and a total time of 2.423782s. Thus it is the combination chosen for the subsequent tests and the final MLPClassifier object that is created, as it is the most stable and best-performing hidden layer combination in the parameter list, outperforming every other combination.

Test 3: Maximum Number of Iterations (Epochs)

For the third test, first, the default values for the activation function, solver, and hidden layer sizes are initiated, using the best-performing hyperparameter values found from the previous test, which are the “tanh” activation function, “adam” solver, and (27, 27, 27) hidden layer size. After that, the maximum number of iterations is instantiated to 1000 to test the maximum epochs or training runs from 1 to 1000. Hence, with this method, in total there will be 1000 total test runs.

So, a for loop is used for looping through the max number of iterations so at every loop the `nn_pipeline` is called and used to create the `MLPClassifier` object with the current `max_iteration` parameter.

The parameters along with their evaluation metric scores are stored in an array and sorted in descending order of their evaluation metric scores (accuracy, f1-score and precision). After that, the array is stored in a Pandas DataFrame to visualise the parameters and their corresponding performance and evaluation metric results easily, which is then printed.

Additionally, a graph is plotted showcasing the effect of increasing the maximum iteration on the accuracy of the `MLPClassifier` model, as seen below.

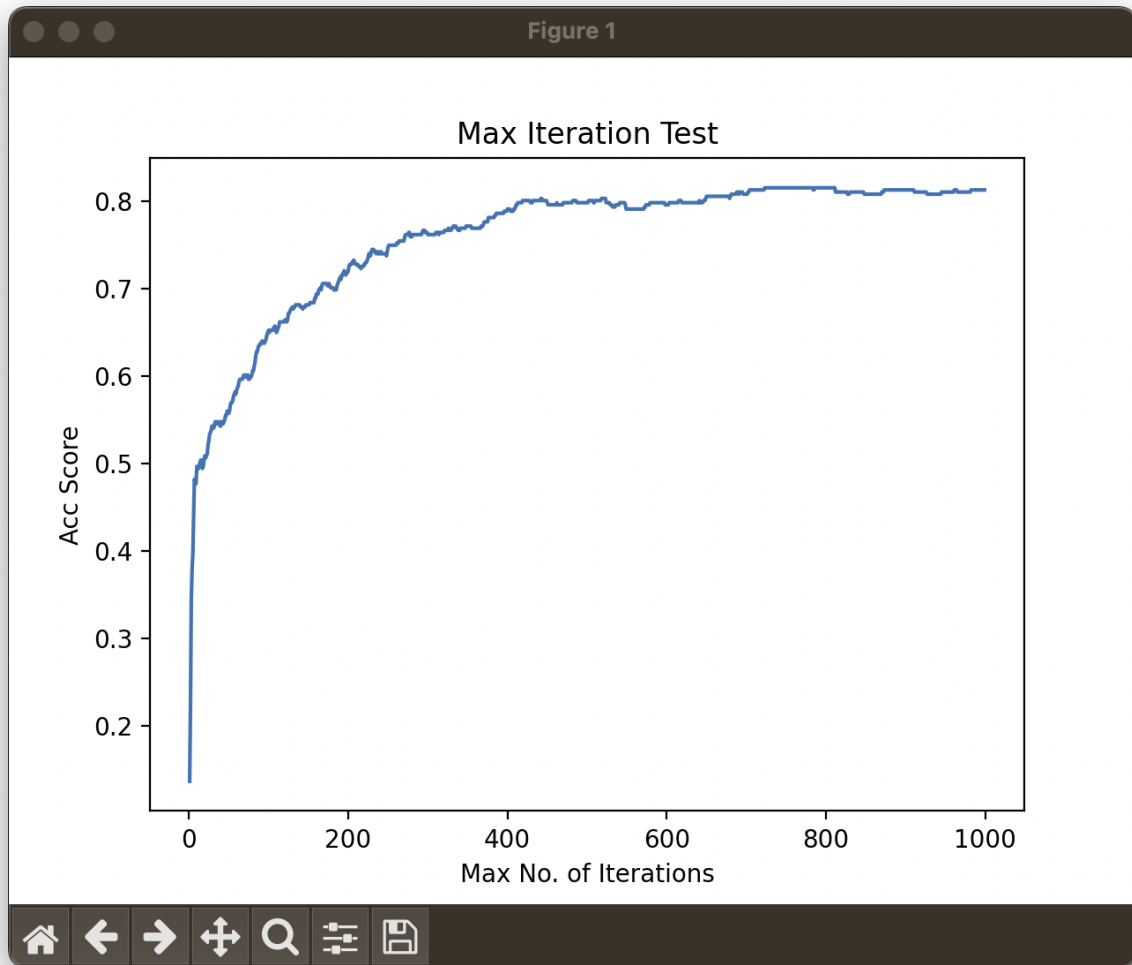


Figure 21: Accuracy score for MLPClassifier model with maximum iterations from 1-1000.

From the figure above, it can be seen that increasing the maximum number of iterations increases the accuracy of the NN model, up until a certain value to which the accuracy score flattens or stabilises. This is because the NN model is converging and does not increase significantly past an accuracy value of 0.815, as it has obtained sufficient time to train completely. Therefore, increasing the number of maximum iterations will only increase the complexity of the NN model as more time is taken to train the model and so the smallest value that maximises the accuracy score is obtained.

For this, the following is the visualisation of the Pandas DataFrame to identify the best-performing number of maximum number of iterations and their corresponding performance and evaluation metric results.

	Maximum No. of Iterations	Acc	F1-Score	Prec	Training Time (s)	Testing Time (s)	Total Time (s)
0	723	0.815085	0.815085	0.815085	2.204982	0.012781	2.217763
1	724	0.815085	0.815085	0.815085	2.200824	0.004882	2.205706
2	725	0.815085	0.815085	0.815085	2.201397	0.002738	2.204135
3	726	0.815085	0.815085	0.815085	2.259706	0.015781	2.275487
4	727	0.815085	0.815085	0.815085	2.213277	0.029241	2.242518
5	728	0.815085	0.815085	0.815085	2.247004	0.000624	2.247628
6	729	0.815085	0.815085	0.815085	2.200015	0.002450	2.202465
7	730	0.815085	0.815085	0.815085	2.228675	0.000595	2.229270
8	731	0.815085	0.815085	0.815085	2.221932	0.044730	2.266662
9	732	0.815085	0.815085	0.815085	2.266050	0.008651	2.274701

Figure 22: Visualisation of the evaluation metric results from the testing of different number of maximum iterations with the MLPClassifier.

From here, it is found that the 723 maximum number of iterations performed the best, returning an accuracy result of 0.815085, an f1-score of 0.815085, and precision score of 0.815085, training time of 2.204982s, testing time of 0.0012781s and total time of 2.217763s. Thus it is the value chosen for the final MLPClassifier object that is created, as it is the most stable and best-performing maximum iteration value in the parameter list, outperforming every other parameter value.

Final MLPClassifier() Object

Conclusively, after obtaining all the optimal values for the aforementioned hyperparameters, it is utilised to build and configure the scikit-learn MLPClassifier() object that will be used to compare with the other ML models that have been designed and developed so far. Therefore, as per the results obtained the applied hyperparameters are the following:

1. Activation function: tanh
2. Solver: adam
3. Hidden Layer Size (hidden_layer_sizes): (27, 27, 27)
4. Maximum Iterations (max_iteration): 723

XI. StackingClassifier

For the hybrid model, the StackingClassifier object of the scikit-learn neural_network library was utilised for its implementation.

The scikit-learn StackingClassifier object is an ensemble model, seen as a stack of estimators with a final classifier or estimator that is designated. It employs a meta-learning algorithm to discover the most effective way to combine the predictions from two or more base machine-learning algorithms. So, it uses the ML technique of stacking, by combining the output of individual base estimators and the final classifier for computing the final prediction. Thus, combining individual models into a singular powerful mode. Hence, through stacking it leverages the strength of each of the individual classifiers that make up the hybrid model, and uses their collective outputs as the input for the final specified estimator. The final specified estimator is trained through cross-validation and will combine these base estimators.

Using a cross-validation technique, such as k-fold cross-validation, the individual models are trained on various subsets of the data, and the predictions from each model are then combined to produce the final prediction. A meta-classifier is used to combine the predictions made by the various models after they have each been trained on different subsets of the data. Since the various models can learn complementary information, this method frequently results in improved performance. Overfitting can be minimized through stacking as you are training each classifier

on a different subset of the data. Due to its ability to lower prediction variance, stacking is also helpful for handling unbalanced datasets.

However, for a base model, considering it has a lower level of complexity and is easier to be described, trained, and maintained than a stacking ensemble, if it is found to outperform the stacking ensemble, the base model should be used in preference.

In this model, the only parameter that was manipulated was the `final_estimator` parameter. So, for the implementation, first the list of the estimators were initialised with the previously trained/fitted model and their name. For this, only the estimators that have acceptable evaluation metric results were considered, this mainly consists of models that had an accuracy rate of more than 0.7. So, the models that were used to construct the `StackingClassifier` were SVC, KNN, DT, XGB, RF, EXT, and ANN.

Then, using a for loop, the models in this list are iterated over, and at each iteration, the `StackingClassifier` object is instantiated with the estimators list and a different final estimator is chosen at every iteration of the for loop. After that, the `StackingClassifier` model is evaluated and its results are appended to the `compare_models` array to be compared with the rest of the ML models that have been designed and developed.

3.4.6 Model Validation/Evaluation Metrics

The next stage of the ML implementation is to evaluate the performance of the designed and built models to allow an efficient comparison between the ML models to determine the best-performing model. Thus, a thorough and versatile evaluation is necessary to develop a model that is robust to different kinds of records. By using different metrics we can see limitations in the model, and attempt adjustments to improve the model's overall predictive power before the model is utilised for unseen data the end-user might input.

Thus, evaluation metrics have to be defined and set to be used to measure the performance models. Considering this problem domain is a classification which is a supervised learning model where the target attribute is a nominal binary attribute, the evaluation metrics are chosen accordingly.

In the program, this stage is performed after every model has been fitted with the dataset and is used for prediction to validate the predictions and assess the performance of the models. So, the `modelValidation()` function, a user-defined function is called, which contains all the commands or statements to call the appropriate metrics calculation methods or libraries.

Accuracy

In the previous stage, from fitting the ML models to the dataset and utilising the testing and training set to make predictions, we will obtain the accuracy estimation values of each of the models. Accuracy measures how often the classification algorithm correctly predicts or the proportion of total dataset records that were correctly classified out of the total instances. Hence, it represents the percentage of accurate results.

$$\text{Accuracy} = \text{Number of Correct Predictions} / \text{Number of All Predictions}$$

Following the confusion matrix, accuracy can be calculated using the following equation, where TP is True Positive, FP is False Positive, TN is True Negative, and FN is False Negative.

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

It is implemented or computed in the program using the Scikit-Learn library's metrics package's `accuracy_score` function (`sklearn.metrics.accuracy_score`) found in the `modelValidation` function().

Confusion Matrix

The confusion matrix is a classification report in the form of a 2x2 matrix generated by various classifiers to relay information regarding the classifications and errors found to give insight and help in determining the effectiveness of the proposed methodology. Hence, the confusion matrix displays the number of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) in the classification. The following diagram displays a general confusion matrix, the information it represents and how these values (TP, FP, FN, TN) are determined:

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Figure 23: Confusion Matrix

There are a number of data that is utilised in the confusion matrix:

- Condition Negative (N): The number of negative cases in the data, or in this case when the target value is 0 (absence of HD)
- Condition Positive (P): The number of positive cases in the data, or in this case when the target value is 1 (presence of HD)
- True Positive (TP): The number of correct positive condition classifications/predictions
- True Negative (TN): The number of correct negative condition classifications/predictions
- False Positive (FP): The number of incorrect positive condition classifications/predictions (Type I Error)
- False Negative (FN): The number of incorrect negative condition classifications/predictions (Type II Error)

The confusion matrix was implemented in the programme using the Scikit-Learn library's metrics package's `confusion_matrix` function (`sklearn.metrics.confusion_matrix`). Based on the confusion matrix values, we can deduce the precision, recall, and f1-score or f-measure values of the classification. These evaluation metrics can also be generated using the Scikit-Learn library's metrics package's `classification_report` function (`sklearn.metrics.classification_report`). The latter was the chosen method in the program as seen in the `modelValidation` function.

Precision

Precision represents the proportion of positively predicted records when the sum of records that are predicted is true or turned out to be true for the true positive rate whereas for the true negative rate is the proportion of negatively predicted records when the sum of records that are predicted is negative or turned out to be false. Hence, It is typically useful for ensuring the program is precise and gives correct results, classifications or predictions. It can be calculated using the following equations:

$$\begin{aligned}\text{True Positive Rate (Precision Positive)} &= \text{TP}/(\text{TP}+\text{FP}) \\ \text{True Negative Rate (Precision Negative)} &= \text{TN}/(\text{TN}+\text{FN})\end{aligned}$$

It is implemented or computed in the program using the Scikit-Learn library's metrics package's `precision_score` function (`sklearn.metrics.precision_score`) found in the `modelValidation` function().

Recall

Recall represents the proportion of the positively predicted records when the instance is actually positive for sensitivity, conversely, for specificity it represents the proportion of negatively predicted records when the instance is actually negative. So, it evaluates the returned relevant values. It is typically useful for measuring the occurrence of when the false negative is of higher concern than the false positive. It can be calculated using the following equations:

$$\begin{aligned}\text{Sensitivity (Recall Negative)} &= \text{TP}/(\text{TN}+\text{FN}) \\ \text{Specificity (Recall Positive)} &= \text{TN}/(\text{TN}+\text{FP})\end{aligned}$$

It is implemented or computed in the program using the Scikit-Learn library's metrics package's `recall_score` function (`sklearn.metrics.recall_score`) found in the `modelValidation` function().

Misclassification Rate

This is the direct opposite of the accuracy metric, which measures the errors of the classification model. In other words, it measures how often the classification algorithm incorrectly predicts or the proportion of total dataset records that were incorrectly classified out of the total instances. Hence, it represents the percentage of inaccurate results:

$$\text{Misclassification Rate} = (\text{FP} + \text{FN})/(\text{TP}+\text{TN}+\text{FP}+\text{FN})$$

It is implemented or computed in the program using the Scikit-Learn library's metrics package's `accuracy_score` function (`sklearn.metrics.accuracy_score`) found in the `modelValidation` function() and subtracting it from 1.

F-Measure/F1-Score

The f1-score or f-measure utilises both the precision and recall values to evaluate the harmonic mean of value of precision and recall, also known as the weighted average of reciprocals. It is at its maximum when the precision and recall values are equivalent. Essentially, it is a useful metric when the FP and FN are equally significant, the addition of data does not change the model's output, or the TN value is substantial. Hence it can be evaluated using the following equation:

$$F1 = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

It is implemented or computed in the program using the Scikit-Learn library's metrics package's `f1_score` function (`sklearn.metrics.f1_score`) found in the `modelValidation` function().

Mean Square Error (MSE) and Root Mean Square Error (RMSE)

The mean squared error calculates the sum of all squared errors. This translates to a return of the average of the square sums of each difference between the estimated value and the true value. A larger MSE is a sign that the linear regression model did not successfully predict the model, and vice versa. It serves as a reliable gauge of how well a model fits your data. The following is the formula in which MSE is calculated:

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

The MSE's sensitivity to outliers is a crucial point to keep in mind. This is due to the fact that it computes the average error of each data point. As a result, an increased error on outliers will increase the MSE.

The square root of a value obtained using the Mean Square Error function is the root mean square error. It is a quadratic scoring formula that also calculates the average error's magnitude and is proportional to the MSE, so it also reflects the amount of error in the predictions. It is calculated by taking the square root of the averaging squared observations' differences from predictions.

It is implemented or computed in the program using the Scikit-Learn library's metrics package's `mean_squared_error` function (`sklearn.metrics.mean_squared_error`) found in the `modelValidation` function().

Mean Absolute Error (MAE)

The average differences between expected and observed values are measured by the mean absolute error. It calculates the exact difference between the predicted value and the observed value and then calculates the average by adding up all of these values.

The mean absolute error (MAE), as opposed to the mean squared error (MSE), calculates the error on the same scale as the data. It is therefore simpler to understand and the MAE is less prone to outliers and does not square the differences.

The following is the formula in which MSE is calculated, where n is the number of errors, Σ the summation symbol, and $|x_i - x|$ denotes the absolute errors:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |x_i - x|$$

It is implemented or computed in the program using the Scikit-Learn library's metrics package's `mean_absolute_error` function (`sklearn.metrics.mean_absolute_error`) found in the `modelValidation` function().

Matthews Correlation Coefficient (MCC)

In machine learning, the Matthews correlation coefficient is used to evaluate the accuracy of binary and multiclass classifications. It considers both true and false positives and negatives, is generally regarded as a balanced measure that can be applied even when the classes have very different sizes, and incorporates both. In its most basic form, the MCC represents a correlation coefficient value between -1 and +1. Inverse predictions are represented by a coefficient of -1, while perfect predictions are represented by a coefficient of +1. This measure or statistic is also called the phi coefficient.

The following is the formula in which MCC is calculated:

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}$$

It is implemented or computed in the program using the Scikit-Learn library's metrics package's `matthews_corrcoef` function (`sklearn.metrics.matthews_corrcoef`) found in the `modelValidation` function().

Training and Testing Time

The testing time is another evaluation metric to determine the time complexity of the ML model. If the model requires a significant amount of time to process the inputs to return an output or provide a classification, this might indicate the program has high complexity and is not suitable for real-world and real-time usage.

If the model performance is not ideal, adjustments can be made to improve the model performance or complexity and to reduce bias. This can be performed through applying the different but appropriate data preparation techniques, whether it is detecting highly correlated features in the dataset, transformation for attributes with high data sparsity or outliers, or imputation of missing values.

That being said, in this department, it is crucial to optimise on the testing time to allow for the form or model inputs to be processed and for a suitable result, prediction or classification to be returned. The training time is also measured, which indicates the complexity of the model, where a longer time to train the model or fit with a dataset indicates that it is of higher complexity. This training operation is not done repeatedly as the ML model is only trained and fitted once. In fact, when the model is deployed into the web app, the chosen best ML model is already fitted with the dataset and then is serialised and pickled, so it is easily loaded and can be used to process the inputs to return a classification rapidly. Similarly, the total time is computed and returned, in the same way only to convey information regarding the model's complexity.

Thus, the testing time is the only crucial metric that has an impact on the real-world usage of the ML model. Even this is improved on, thanks to the operation of pickling the ML model.

It is implemented or computed in the program using the time packages time function (time.time) found in the modelValidation function(). Thus, this is applied in the code for an ML model, before and after the fit() function call to measure the training time, and then before and after the predict() function call to measure the testing time. The sum of the training and testing time returns the total time score.

Conclusively, the actual evaluation metric results of the individual ML models will be presented and explored in more depth in the Results and Discussion section.

3.4.7 Model Selection

After the results have been obtained, model selection is performed, which is the process of selecting one among the possible candidate models for the predictive modelling problem, in this case, for the HD prediction/classification model. The existence of competing concerns aside from the model performance, including complexity, maintainability and available resources, necessitates that a standard model selection method should be adopted.

For this implementation, we will follow the concept or principle of parsimony or Occam's razor where the most simple model is chosen. Therefore, each model is fitted with the dataset, and we select the best model at every level when comparing the accuracy or evaluation metric results with the chosen models. Then, in the event, there are two or more models having the similar best accuracy or performance, the model that is more simple and possesses less complexity will be selected to be integrated with our web app system [15].

So, the performance of the prospective model will be the most important criteria, but the complexity of the program will be kept in consideration, to ensure that the training time is not too extensive and to ensure that it is practical for real-world scenarios. As mentioned previously, in order for the ML model to be applicable or practical, the testing time for the ML model should be optimised or minimised so that the form or model inputs are processed more rapidly and an accurate output is returned quickly. Also, this will ensure the maintainability of the program is optimal for future maintenance and deployment tasks as well as requiring lesser hardware resources to run smoothly.

In the program implementation, this step is done manually, with the help of the `compare_models` array that stores all the tested ML models and their evaluation metric results. This array is converted into a Pandas DataFrame, and then displayed in the terminal and outputted in an excel file (.xlsx) to allow easy analysis of the results and models.

Therefore, from this step, the best-performing model was found to be the StackingClassifier model with the Random Forest model as the final estimator. The evaluation metric results of this model and why it was chosen as the ideal model will be explored in more depth in the Results and Discussion section. Therefore the following are steps to perform after the best-performing model had been chosen which includes saving, pickling or serialising it to be used with the web app and viewing the feature importance of the model. This can be seen in the program under the StackingClassifier model creation section, as a separate if statement, so that it only executes for the StackingClassifier object that utilises the Random Forest as the final estimator.

Saving the ML Model With Pickle

After selecting the model, in order to load this ML model into Django and the web app, the model is “pickled”.

Pickle is a helpful Python tool that lets you share, commit, and re-load pre-trained machine learning models. It also helps you save your ML models and reduce time-consuming retraining. The majority of machine learning (ML) data scientists will use Pickle or Joblib to store their ML models for later use. Pickle is a universal object serialization module that supports both object serialization and object deserialization. Thus, in this case, it is used to serialize the ML objects, to be reused in the web app.

For this, the `dump()` function of the joblib package is utilised to serialize the object and convert it into a “byte stream” which is in turn saved as a file with the name “heart-strat.pkl”.

Viewing the Feature Importance of the ML Model

Following that, the feature importance of the chosen best-performing ML model is identified and visualised. The feature (variable) importance identifies the contribution of each feature to the model prediction. In essence, it establishes the extent to which a particular variable is useful for a given model and prediction. We use a score to quantify the significance of a feature; the higher the score, the more significant the feature. Possessing a feature importance score has numerous advantages.

a) Model Improvement

For example, the relationship between independent variables (features) and dependent variables (targets) can be established. We could identify and eliminate irrelevant features by looking at variable importance scores. The model may run faster or even perform better if the number of meaningless variables is decreased.

b) Model Understanding and Interpretability

Additionally, feature importance is frequently used as a tool to improve the interpretability of ML models. It is possible to deduce from the scores why the ML model makes particular predictions as well as how its predictions can be altered by changing the features. Similar to a

correlation matrix, feature importance makes it possible to comprehend the connection between the features and the desired outcome. Additionally, it enables you to comprehend which features are unnecessary for the model. Hence, other stakeholders can understand and understand your model better if you use feature importance. Moreover, you can figure out which features contribute most to the accuracy of your model's predictions by assigning scores to each feature.

Implementation of Feature Importance

For the implementation of this, first, the column headers are initialised, this is following the features that have been selected of the dataset and the Pandas method `get_dummies` has been applied to it, to obtain the hot-encoded counterparts for the categorical/object attributes. After that, using the `model.feature_importances_` attribute, we can identify these individual importance scores to be matched with the aforementioned headers to be converted into a Pandas DataFrame, and then sorted in descending order to view the most influential features. This DataFrame was then plotted to visualise the feature importance better. The results of these feature importance will be explained in more detail in the Results and Discussion section.

3.5 Requirements Plan for Web Application

Introduction

The process of requirements engineering or the requirements planning stage involves several stages to ensure that the web app component of the system is designed and implemented to fulfil its required functionalities. Typically, these generic processes are similar for most software development projects even for this heart disease prediction/classification system, which includes, requirements elicitation, analysis, validation, and management. However, although similar, the implementation of these requirements stages differ depending on the application domain or objectives, the stakeholders involved and the developers in charge of fulfilling the requirements.

Therefore, the following requirements plan, goes over how these stages were conducted and the outcome of each stage for the development of this heart disease prediction/classification system. With thorough planning and research, it ensures the final software requirements specification (SRS) or system functionality document comprehensively highlights all the necessary functional and non-functional requirements the software should fulfil. Simultaneously, reducing the chances of neglecting any critical system functionalities and ensuring the implementation stage proceeds smoothly with minimal backtracking.

3.5.1 Requirements Elicitation

This first stage is intended to identify the methods in which we discover or elicit requirements that the web component of the heart disease prediction/classification system should fulfil.

Stakeholders

The initial part of the requirements elicitation stage is determining the stakeholders of the project and system. The stakeholders refer to everyone who's involved or interacts with the heart disease prediction/classification system, in this case, stakeholders include end-users (patients and doctors), system administrators, engineers/developers performing maintenance. Therefore, it's crucial to ensure that the proposed requirements and functionalities cater to these stakeholders, since they'll be using the final system. So, they serve as a source of requirements as well.

Based on this we can identify some common characteristics of the stakeholders and end-users. For instance, the age of the audience or site visitors will be of an adult's age which regularly has access or uses computer or web-based systems (approximately 35-75). So, this will help us determine the design choices for the web platform, in terms of colour scheme, where a professional and minimal option will be chosen that is not too striking or alarming and for font size and style a legible and clear option will be ideal. We should refrain from adding too much cluttering of texts on the pages of the website and ensure that all the fields of the heart disease predictor form also provide legible and helpful information to assist the site visitor in entering details.

Methods of Requirement Discovery, Elicitation or Collection

At this stage we identify the sources or specific methods of obtaining the requirements of the web app for the heart disease prediction/classification system. The following section will explain how these requirements discovery methods were implemented or conducted.

Therefore, altogether there are two main methods employed to identify these requirements, which include reviewing similar heart disease risk calculator websites and from reviewing journal articles related to heart disease risk predictions:

1. Ethnography or Reviewing Similar Heart Disease Risk Calculator Web Platforms

Ethnography in requirements engineering is the qualitative study or analysis of how users interact with a system. In this case, we can review similar heart disease risk calculator web systems from a users' point of view. This allows us to gain a deep understanding of how the overall system domain and its functionalities should work. Therefore, this method of requirements elicitation, is beneficial for understanding existing processes in other heart disease prediction/classification systems.

These alternative heart disease risk calculator websites have a similar target demographic and stakeholders as mine. So, we can gain inspiration from these platforms to generate a list of requirements our desired web platform should also fulfil. This makes the site reputable as it fits into the market of heart disease prediction websites and can perform similar functionalities. Plus, as it has common components, such as the heart disease prediction form, labels and input fields, results page, this improves the platform's user-friendliness since they're already familiar with the other platforms. Thus, the learning curve is lower.

Therefore, I can identify the implementation of different functionalities and features that are integral to their website and for end user interaction. These key functionalities I analysed were:

- Website and page's design and layout
- Form layout and guiding users to entering field inputs

I made a list of these websites analysed and gathered their key info to a note-taking app, Google Keep. This makes referring back far easier. Thus, the heart disease risk calculator websites I analysed includes:

- UpToDate.com's Calculator: Cardiovascular risk assessment [45]
- MayoClinicHealthSystem.org's Heart Disease Risk Calculator [46]
- Reynolds Risk Score [47]

- Medical College of Wisconsin's Coronary Heart Disease Risk Calculator [48]

Results and Analysis

From analysing these website, the following are some key information worth noting when developing our prospective system:

- All of the sites house the HD prediction risk calculator form within the home page of the HD Risk Calculator section or they do not have to be redirected to the appropriate page. So, this can improve the usability and intuitivity of the website where the main component, the risk calculator form where users can enter their data, is ready to be accessed from the front page.
- The layout of the forms are similar across all the websites, where it begins with providing a brief introduction, instructions, disclaimers to using the system, before showing the field names and the input fields where users can enter their data (general and physiological).
- Some websites such as the Reynolds Risk Score, provide an FAQ section and an on-hover description to explain the question and assist or guide users in entering their data in the form.

2. Reviewing Similar Heart Disease Prediction and Classification-Related Literature

As seen from the literature review section (2.0), several journal articles were reviewed in depth. Therefore, through this review, we developed a better understanding of the problem domain (heart disease prediction/classification), the goals/aims of the authors' projects and study, the datasets used, and the features correlating with HDs and CVDs occurrence. We performed a brief review to understand what are CVDs or HDs, the medical implications it entails, types of problems, contributing factors, preventative measures and electrocardiography (a key measure of the likelihood of patients developing HD). This would aid us in better understanding the problem domain, and to ensure the ML models developed will take the context into consideration to ensure effective and optimal results and diagnosis.

Along with that, from these journal articles, we determined how the authors conducted their ML implementation for finding the optimal solution that provides the best prediction accuracy for this problem domain, including the method chosen to preprocess the data, partition the data, feature selection, hyperparameter tuning, ML classification algorithms used to develop their model (KNN, logistic regression, etc.), evaluation methods and metrics used. We also performed an in-depth review into the prospective individual ML algorithms to understand their functioning, the required parameter values and how they can be applied for our problem domain.

Overall, through reviewing these journal articles, we have a better understanding of the general steps involved and key considerations for the ML implementation of the system. Section 2.0 of this planning document provides a more in-depth look at the results of the literature review conducted and section 2.4 summarises the literature review section.

3.5.2 System Functionality or Software Requirement Specification (SRS)

After gathering and analysing the data from the requirements elicitation stage, we categorised and organised the requirements collected in order of priority. After that, we arrive at the requirements specification stage, which is essentially the process of writing down the functional

and non-functional requirements of the heart disease prediction system in a comprehensive requirements document, known as the Software Requirement Specification (SRS).

Functional Requirements

These are system requirements for the prospective web app component of the heart disease prediction/classification system. So, it entails the functionalities and services that the system provides. This involves high-level description of functionalities of the system.

With that said, the following are functional requirements identified that this system should fulfil:

1. Users shall be able to enter their information (general information and physiological data) into the heart disease predictor/classifier form.
2. The system shall return and display a result or the level of risk of the patient developing HD based on the ML model's processing of the user's input.
3. The system shall indicate the features or the entered parameters that contribute to a user's risk of heart disease and display appropriate tips to mediate their condition.
4. The user shall be able to print the results page containing the presence of HD and risk report, along with the entered parameter inputs.
5. The system shall allow users to sign up for a patient or doctor admin account on the system. By signing up, the users entered data shall be saved in their list of trials on their account page. For signing up for a doctor's account, the user shall have to enter more information to verify themselves as doctors, such as their name, university, place of practice and registration/TPC numbers. These are the key information required to verify doctors.
6. The user shall be able to log in to their patient, doctor, or admin account.
7. The patient account shall be able to see the list of their trials of using the HD predictor form on their account page, which will display the results report along with the entered parameter inputs.
8. The patient account shall be able to request for assistance from doctors on the platform.
9. After the connection is approved by the patient account, the doctor or patient can message each other to discuss the results and arrange for further appointment the patient's and doctor's contact information that are unveiled and they can email to set an appointment.

Non-Functional Requirements

As opposed to functional requirements, these form of requirements entail the behaviour and properties the system should possess as well as constraints the system should comply with. So, often these non-functional requirements have a more crucial need to be fulfilled as they dictate how the entire system functions instead of singular components as seen with functional requirements. Therefore, if not fulfilled it could render the system unusable.

For this heart disease prediction system, the following are the primary non-functional requirements that have been identified and need to be fulfilled:

Accuracy

This is the most important non-functional requirement of the heart disease prediction system, where the system should utilise the chosen best ML model, that should be accurate in processing

the user inputs to output a correct and valid diagnosis or result. In other words, the system should possess an admirable performance in classifying inputs to output correct results. Although it is difficult to obtain a perfect system that is able to attain 100% accuracy in classifying heart disease among patient records, the system should strive to increase its performance in terms of the evaluation metrics. Hence, there are several ways the ML model's performance is measured and evaluated, as described further in the evaluation metrics or model validation section of the ML implementation (3.4.6).

Fast Processing Speed

While ensuring the heart disease prediction system returns an accurate result, the system and the chosen ML model should also be sufficiently quick or require low execution time to process the user inputs and return the appropriate output. For this project, having a moderate to long training time of the model is fine, since that section is not always performed in a real-world scenario. However, the execution time of the system's operations (ML model testing time) should be low, which includes retrieving user inputs, processing them using the trained ML model, returning the output of the ML model, and displaying the output in a clearly formatted results page. This is because the users utilise the system in real-time and by providing a fast response the users would not have to wait an extended period of time to obtain their results, which could even lead to a high bounce rate. Thus, we can test the processing time or testing time of the system, in terms of seconds, during the implementation (3.4.6).

Reliable

The system should be online, live, functional and accessible at all times and experience little to no downtime. This is another form of measurable non-functional requirement where we can test in terms of time or seconds the platform is inaccessible. Thus in the event, downtime does occur it should not exceed 5 seconds in any one day, to ensure it is not noticeable to the end user and they can resume with their activities or interactions with the system almost instantly. All in all, the system should be able to handle multiple concurrent users and activities/interactions, and simultaneously be less likely to experience crashes and backend errors.

This is to preserve the reputation and reliability of the system. Besides that, it's to ensure that the user's activities or tasks aren't halted midway. Overall, by keeping platform crashes and downtime to a minimum, this can ensure the user experience is conserved.

3.6 Technologies and Tools Required

This section details the various tools, technologies, software, libraries, programming and markup languages utilised throughout the methodology, namely for the system design, development and design processes of our heart disease prediction/classification system.

Python Programming Language

The Python programming language is to be used throughout the development process, to define the functions and logic necessary, namely for the implementation of the ML models and for the development of the web application where the chosen ML model will be deployed. This programming language was chosen, in order to utilise some important external Python libraries that are crucial to our development and implementation process, which include scikit-learn for developing and implementing the ML models and Django web framework necessary for the

backend or server-side scripting of the web application. Besides that, Python provides a simplistic, readable and intuitive syntax that mimics natural language along with the numerous learning resources and support available for the language as well as its libraries (Scikit-learn, Django, etc.). All this ensures that more focus can be placed on designing the ML models and web system and reducing the overall development and debugging time requirements, especially considering the short time constraint.

Scikit-Learn

Scikit-Learn is a free-to-use machine learning library for the Python programming language. It contains various classification, regression and clustering algorithms that we can utilise to develop ML algorithms and design hybrid models to be tested with our dataset for finding the best-performing model for our problem domain of classifying and predicting the onset or presence of heart disease amongst patients. It can also be used for data mining and analysis applications. Therefore, in our implementation, methods provided by the Scikit-Learn library will be utilised for data partitioning, model development, model training, and model evaluation or testing.

Django

Django is a free and open-source Python-based web framework utilised for the server-side or backend scripting part of the website development, making it more efficient and rapid to create safe, secure, and maintainable websites and web applications using the Python programming language. It promotes the reusability of components and possesses a range of ready-to-use features, thus ensuring that the web app development process or portion of the development is not too extensive and completed more easily and rapidly, thus more focus can be placed on implementing the ML model. It will especially be useful for collecting form inputs from the CVD prediction or classification form to be stored in the MongoDB server. In our implementation Django serves as the main backend framework, from which the web app is built, defining the URL routing, HTML page templating, the user models for creating the different users accessing the platform (patient, doctor, admin), and for defining the other backend functions as seen in the MyAPI/views.py file, including handling form submissions, the Django REST framework for handling forms through API calls, the registration, login/authentication requests, and loading the various web app pages.

Pandas

Pandas is an external library for the Python programming language that is utilised for data manipulation and analysis, providing data structures and methods for manipulating numerical size mutable tabular structures (DataFrame) and time-series. It is typically useful for the data extraction and preparation tasks of the project, especially owing to its high-level data structures and tools for data analysis as well as its built-in methods for grouping, combining and filtering data. Hence, it can be useful for interacting with the dataset, by storing attribute values in a dataframe that can be utilised for further processing with the ML models.

Numpy

Numpy is another external library for the Python programming language which is a dependency of pandas, so after installing pandas, we will be able to import the numpy library into our Python programme. It is one of the most powerful Python libraries, providing mathematical operation

and is mainly used for scientific computing and working with multidimensional arrays, such as for matrix processing. Thus, it is important for our implementation of the ML models for interacting with the dataset which can be treated in the form of a multidimensional array.

Matplotlib

Matplotlib is an external library for the Python programming language which provides plotting functions, which is useful for data visualisation that are static, animated, and interactive. For this project, the Matplotlib functions will be necessary for visualising the dataset and the distribution of its features to help with identifying appropriate preprocessing techniques and patterns in the dataset. It can also be useful for displaying the graphical output of an ML model.

Keras

Keras is an external library for the Python programming language that provides machine learning capabilities, such as allowing developers to design and develop deep learning models or neural networks, including ANN, convolutional neural network (CNN), and RNN. Along with that, built-in methods for grouping, combining and filtering data are provided. For our implementation, this library will be used to design and develop NN models to be applied and tested with the dataset.

Seaborn

Python-based Seaborn is a Matplotlib-based tool for statistical data visualization. With the help of the library, we can make descriptive and impressive Python visualizations. In our ML model comparison program, it is utilised to generate multivariate plots such as the correlation heatmap matrix plot and bar chart for the numerical attributes of the dataset and the individual multivariate plots in the form of a bar chart for the categorical attributes with the target attribute.

HTML5, CSS

HyperText Markup Language (HTML) is the markup language used to construct the structure of the web pages of our web applications that will house the ML models. It will be used together with Cascading Style Sheets (CSS) which is utilised for defining the presentation or style of the HTML elements to format the pages of the heart disease prediction system in a more well-designed and clear manner.

MongoDB

MongoDB is the chosen database for the web application implementation as it is used to store the user models created, their login information, form submissions and data, as well as any other data that is stored by Django backend framework. The database MongoDB is based on documents with collections, which are composed of documents, that are the building blocks of a database. In an RDBMS, documents are essentially the same as records. MongoDB is easy to develop for and scale, in part because there is no schema required when creating documents and documents have a field value pair setup akin to JSON. Strings, numbers, arrays, and even other documents can all be included in values. Especially for this application where no strict structure is needed for storing the data as seen from its requirements and the kind of data that need to be stored.

Hence, a NoSQL database is seen to be the more preferable option, as it is optimised for speed as well, ensuring records are written and read at a rapid rate to the database. Another advantage of MongoDB is how it hashes sensitive information, mainly user passwords after accounts have been authenticated or created. Besides that, the MongoDB database is hosted in their shared cloud database service, MongoDB Cloud Atlas, in the Amazon Web Services Cloud.

TailwindCSS and DaisyUI

Tailwind CSS is an open-source CSS framework where its primary distinguishing characteristic is that, in contrast to other CSS frameworks (e.g.m Bootstrap), it does not offer a list of predefined classes for elements like buttons or tables, instead, generates "utility" CSS classes to style each element. Thus, a HTML component can be easily styled by just specifying specific CSS classes for it, as per its documentation. On top of that, daisyUI is a customizable Tailwind CSS component library that extends TailwindCSS to include more design augmentations and classes to be specified that can improve the design, layout, structure and interactability of the site.

Joblib

Joblib offers tools for pipelining Python jobs and is a component of the SciPy ecosystem. In our program Joblib is utilised in order to pickle the chosen best-performing ML model to be deployed into the web app, to process form inputs as model inputs and return a classification or prediction based on the processing of the ML model. So, Joblib or Pickle is a helpful Python tool that lets you share, commit, and re-load pre-trained machine learning models. It also helps you save your ML models and reduce time-consuming retraining of the ML models with the dataset.

Amazon Web Services, AWS

AWS provides on-demand cloud computing services and resources. Thus, it is utilised as the hosting environment for the Django web application of this project through its robust, scalable and flexible Amazon Elastic Compute Cloud (Amazon EC2) service which provides scalable computing capacity in the AWS cloud. Thus, the Django web application component of the project is deployed and hosted on the AWS platform, to improve its accessibility, by simply accessing the platform through entering the IP address or domain name (explained later as a domain name is pointed to this IP address) of the web application and further establishing it as a production-ready application.

Docker

Docker is a free OS-level virtualisation tool that delivers software in the form of packages known as containers. So, it provides a self-contained system isolated from the host OS, removing any drawbacks that may bring, by delivering and running the application in a virtualised environment. This can be seen in the DjangoAPI/Dockerfile. Overall, Docker makes the process of deploying the Django web app to the hosting environment, AWS, far simpler and in a structured and systematic manner.

Nginx and Gunicorn

Gunicorn and Nginx are both HTTP servers for Python. Gunicorn, is a production-ready Web Server Gateway Interface (WSGI) web server for Django and Python applications, as it is robust and can handle production levels of traffic. It is a significant step up over the default

development server that ships with Django, which was only used for testing purposes on my local machine. Gunicorn extends from this, and is even able to handle all the dynamic files. The call to the Gunicorn method to build and run the Django application can be seen in the Dockerfile. Nginx is utilised as a reverse proxy for Gunicorn owing to its high performance connection handling mechanisms to serve static and media files, as seen in the DjangoAPI/nginx/default.conf file.

Jenkins

Jenkins is an open-source automation server written in Java to automate certain aspects of software development and production websites, including for tasks such as building, testing, and deploying. Therefore, in this implementation, it is utilised for facilitating and implementing continuous integration and continuous deliver (CI/CD) workflows or pipelines. In short it allows commits made to the web app's connected Github repository to be automatically built, integrated and deployed with the instance hosted on AWS EC2. The implementation of this Jenkins pipeline is explained in more depth in the web app implementation section.

Cloudflare

Cloudflare is essentially a free content deliver network (CDN) and Distributed Denial of Service (DDoS) mitigation service that serves a few purposes for the web app implementation using Django to make it more production-ready. Cloudflare is the chosen domain name registrar, from which the domain name heartassist.net was purchased and registered securely with. Along with this, Cloudflare provides domain name system (DNS) management services to manage the DNS records of the domain name and the web application and proxy the traffic to the domain name. Additionally, other Cloudflare configurations that were made to improve the web application and its production-ready environment is its SSL/TLS certification service feature, caching with its CDN, and securing against DDoS attacks.

IDE - Visual Studio Code

Visual Studio Code (VSC) is the chosen integrated development environment (IDE) for the development of this project, both for implementing the ML models as well as for developing the web application. It is chosen owing to its simplistic design and its useful functionalities in aiding debugging, syntax highlighting and the execution of programs, especially since plugins can be easily installed to extend its capabilities.

Adobe XD

Adobe XD is the user interface (UI) design tool used for designing and creating the initial mockup and prototype of the web application that will integrate with the ML model, It will be used to design the layout of the web pages that are to be implemented that will serve as a blueprint or reference point for the development or implementation process. It was chosen as the ideal UI design tool owing to its intuitive controls ensuring the designing tasks are completed efficiently with minimal errors. Additionally, it provides default frames with the same sizing or aspect ratio as a website view (1920 x 1080px), thus ensuring the elements can be placed easily and accurately to match with the actual implementation or web app. Also, designs on Adobe XD possess prototyping capabilities, which allow for simple animations to be implemented and

visualised in the prototype, for example button clicks redirecting to the next page or drop-down menus in the form.

Visual Paradigm

Visual Paradigm is used to design the UML diagrams to visualise the logic of the web app that will integrate the chosen ML model. It has intuitive drag-and-drop controls to allow for the appropriate diagrams to be created efficiently.

Software Testing - Web Browsers

To test the web application that is developed, web browsers will be used, namely, Google Chrome and Firefox Developer Edition. By testing and observing the system on different web browsers, we can ensure that the system appears correctly and consistently no matter which web browser the end-user utilises to access the system. Web browsers will be used as part of the User Acceptance Testing to check the system from the end-users perspective, ensuring that the form is submitting, and the links and buttons are working according to requirements.

3.7 Web App Prototype Design

This section displays the prototype design, illustrating the pages and main components of the web app component of the prospective heart disease prediction system. For this, the online prototyping tool, Adobe XD, was used to design the system and its layout. The following is a link to the designs at Adobe XD which can be accessed:

<https://xd.adobe.com/view/906f59c0-dd3e-4485-bd9e-d910847abedd-c53c/?fullscreen>

Besides that, the following sections contain the attached images of the designs made, to support the brief narrations. For a clearer and zoomed in image and viewpoint, you may view the designs through the above link on Adobe XD and using a PC.

Heart Disease Risk Calculator

Powered by the best performing machine learning model designed and developed after performing extensive comparative tests, this calculator predicts the likelihood of you developing cardiovascular diseases or heart diseases over the next 10 years. It requires some physiological data inputs which are potential risk factors to calculator your risk. So, be sure to enter your most recent values.

For more information about this calculator, visit the About page, by clicking [here](#).

Notes:

- You can consult with a doctor to assist in understanding your personal risk and interpreting these results.
- The calculator may overestimate or underestimate your risk, hence, it cannot classify for certain 100% whether you will have a cardiovascular event.
- You can save your records to the system for future viewing or comparisons, by signing up or logging in to your patient account.

Enter the patient's information below:

 Male	 Female
---	---

Age: Years

Height: m

Weight: kg

? Systolic Blood Pressure: mmHg

? Diastolic Blood Pressure: mmHg

SMOKING STATUS (Smoked in the last year)

Yes No

? Total Cholesterol: mmol/L

? HDL Cholesterol: mmol/L

? Diabetes present: ▾

? Taking blood pressure medication: ▾


Figure 24: Prototype design of the Home/Calculator Page

Explanation of Home/Calculator Page

This home page is the landing page of the system, so when users land on the page, the calculator or predictor form, which is considered the most important main component of the web app system, will be present on this main front page. This ensures to improve the ease-of-use and intuitivity where the site visitor can utilise this form from the first page without having to find the appropriate button or link on the page to be redirected to a separate calculator page. This page starts with the sticky header which maintains on the viewport as the user scrolls down. This header section is consistent in terms of functionality and design across every page in the website. Another consistent element is the footer which is the same on every page of the website. At this header section, there's the navigation bar with the centred menu consisting of 4 buttons namely, the calculator page button (linking to this home page), the about page button (linking to a simple about page, which describes the system to the end user and addressing some frequently asked questions), the log in button (brings up the login form) and sign in (brings up the registration form).

After that, this front page starts with some text giving a general overview about the system and its purpose. Then, in the notes section some key information or disclaimers are presented to the site visitor. Following that is the main component of this page which is the calculator or predictor form, which accepts user inputs for certain possible risk factors. Hence, the risk factors or input features to be considered will be used as the labels for form, so when users enter their information into the fields, the data can be extracted and entered into the chosen best ML model to process the information and output a result. For now, the labels used are general indicators or risk factors for heart disease screening, and after the ML implementation process and the data understanding and feature selection stages, the exact input fields and labels to include for this form will be more evident. Error handling will be performed by this form, using Javascript, to check the user input and to ensure that data inputted is in the right data type and range to ensure it can be inputted into the ML model and the correct and appropriate output is generated. Next to some of the fields which may be difficult for the user to comprehend, there is a button, where on-hover, it will display overlay information about the particular field and question and the data it is expecting. This can help users identify and know what information to input into a particular field, hence, further improving the navigability and usability of the system. After the user has filled all the necessary fields, they may submit the form, by clicking the Calculate button, reset the form to its blank state using the Reset button, or switch to the US/Imperial or metric units for certain input fields, where the standard of measurement may vary for different regions.

Home/Calculator Page With Results

 [Calculator](#) [About](#) [Log In](#) | [Sign Up](#)

Heart Disease Risk Calculator

Powered by the best performing machine learning model designed and developed after performing extensive comparative tests, this calculator predicts the likelihood of you developing cardiovascular diseases or heart diseases over the next 10 years. It requires some physiological data inputs which are potential risk factors to calculate your risk. So, be sure to enter your most recent values.

For more information about this calculator, visit the About page, by clicking [here](#).

Notes:

- You can consult with a doctor to assist in understanding your personal risk and interpreting these results.
- The Calculator may overestimate or underestimate your risk, hence, it cannot classify for certain 100% whether you will have a cardiovascular event.
- You can save your records to the system for future viewing or comparisons, by signing up or logging in to your patient account.

Results, Your Heart Disease Risk Score is:

12%

Meaning you are at relatively low risk of developing cardiovascular disease in the next 10 years.

Show Additional Info

General statement describing user's risk, and factors contributing to risk

Factor: risk percentage
Explanation of why factor contributes to the risk. Ways to mitigate the risk

- **Habitual Smoking** 4%
Cigarettes contain harmful substances that affect the respiratory and circulatory system, impacting the heart.

To mitigate, cut down or quit habitual smoking.

[Show All](#)

[Print Results](#) [Retry](#)

What next?

You can save your results to the system and contact a doctor, by signing up for a patient account.

Click [here](#) to register for a patient account.

Inputs

Male Female

Age: Years

Height: m

Weight: kg

1 mmHg

2 [Show All](#)

SMOKING STATUS (Smoked in the last year)


Yes No

3 mmol/L

4 mmol/L

5

6

 [Calculator](#) [Sign Up](#) [Privacy Policy](#)
[About HeartAssist](#) [Log In](#) [Terms and Conditions](#)

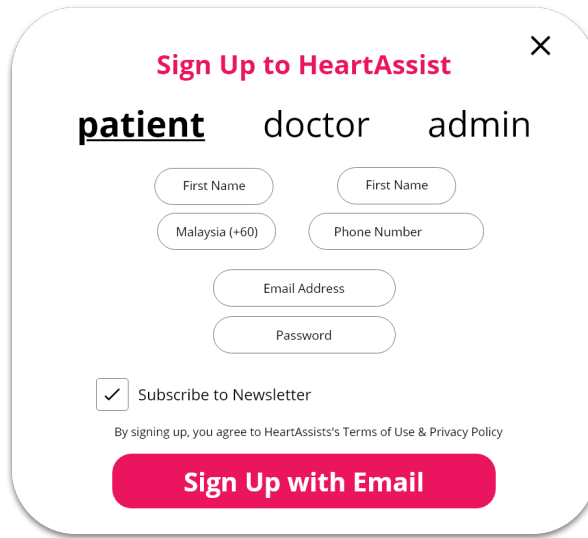
© 2022 HeartAssist. All rights reserved.

Figure 25: Prototype design of the Home/Calculator Page With Results

Explanation of Home/Calculator Page With Results

After the user has submitted the form from earlier with their input data passing the error handling, within the same calculator/home page, the output of the processing done by the ML model using the input data will be displayed. So, tentatively this is the envisioned design of the results presentation or output. It will start by displaying the percentage of risk associated with the particular user in developing CVD or HD. Then, there will be a list of additional or more in-depth analysis on the results. Here, a description of the user's risk and the contributing factors based on the input data will be presented. Hence, based on the input data, the list here will show the specific factors contributing to the risk percentage, along with some background information justifying the reason or explaining why the factor is considered, and the ways to mitigate this risk factor. The user's inputs are also displayed, to allow for easy referral to compare with the result. The user has the option to either print the result by clicking the Print button or to attempt at submitting another form by clicking the Retry button. If the user wishes to save their results to the system for future review or for consulting with a doctor, as well as to connect and reach out to a doctor on the platform they may sign in or register for a patient account.

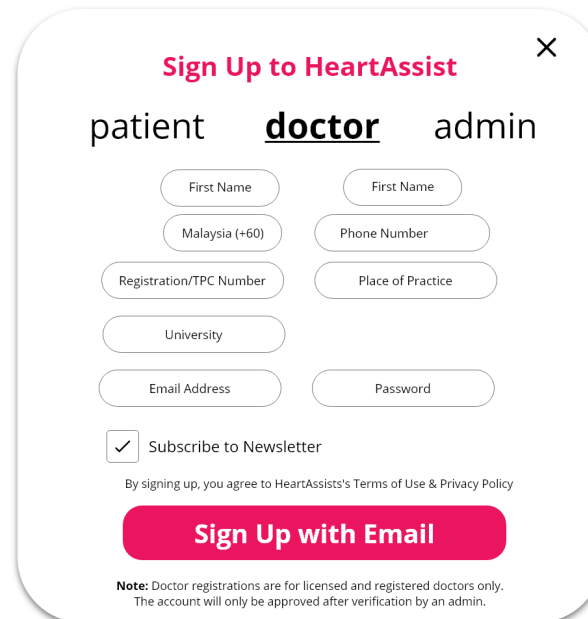
Registration Form For Patient



The form is titled "Sign Up to HeartAssist" and has a close button (X) in the top right corner. It features three tabs: "patient" (which is underlined), "doctor", and "admin". The form contains the following fields: two "First Name" fields, a "Malaysia (+60)" field, a "Phone Number" field, an "Email Address" field, and a "Password" field. There is a checked checkbox for "Subscribe to Newsletter" and a line of text stating "By signing up, you agree to HeartAssist's Terms of Use & Privacy Policy". At the bottom, there is a prominent red button labeled "Sign Up with Email".

Figure 26: Prototype design of the Patient Registration Form

For Doctor



The form is titled "Sign Up to HeartAssist" and has a close button (X) in the top right corner. It features three tabs: "patient", "doctor" (which is underlined), and "admin". The form contains the following fields: two "First Name" fields, a "Malaysia (+60)" field, a "Phone Number" field, a "Registration/TPC Number" field, a "Place of Practice" field, a "University" field, an "Email Address" field, and a "Password" field. There is a checked checkbox for "Subscribe to Newsletter" and a line of text stating "By signing up, you agree to HeartAssist's Terms of Use & Privacy Policy". At the bottom, there is a prominent red button labeled "Sign Up with Email". A note at the very bottom states: "Note: Doctor registrations are for licensed and registered doctors only. The account will only be approved after verification by an admin."

Figure 27: Prototype design of the Doctor Registration Form

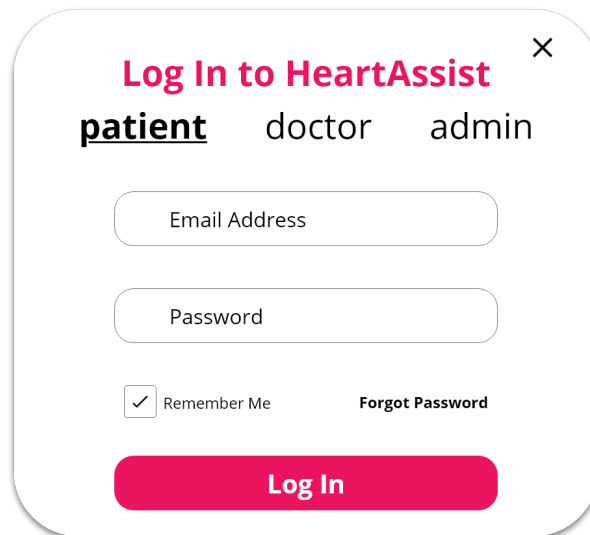
For Admin

Figure 28: Prototype design of the Admin Registration Form

Explanation of Registration Form

There are 3 types of registration forms, for patients, doctors and admins. The registration form for the patient and admin accounts appear alike with the same general input fields required for typical account verification and creation. However, for the doctor registration form, tentatively there are three extra input fields (Registration/TPC number, Place of practice, and University) to verify that the user is a licensed practising doctor to ensure that the users on the system connect and contact with verified and true medical practitioners. Hence, after the input fields have been filled, there is a box to tick to subscribe to the website newsletter, followed by a privacy policy statement, and a button to confirm the registration. For the patient accounts their account will be authenticated and created after the submission of the registration form. Whereas for admin and doctor accounts, will require the existing admin to check and verify the details before approving the account creation, and allowing them to log in to their accounts.

Login Form




The image shows a prototype design of a login form titled "Log In to HeartAssist". The form is contained within a rounded rectangular box with a close button (X) in the top right corner. Below the title, there are three user role options: "patient" (which is underlined), "doctor", and "admin". The form includes two input fields: "Email Address" and "Password". Below these fields, there is a "Remember Me" checkbox (which is checked) and a "Forgot Password" link. At the bottom of the form is a prominent pink "Log In" button.


Figure 29: Prototype design of the Login Form

Explanation of Login Form

Similar to the registration form, there are 3 separate login forms as well, to make login form handling more efficient. At the top, users can select whether they require the patient, doctor or admin form. Then, there are the standard login credential input fields, email and password. A Remember Me checkbox to make logging in again easier and a Forgot Password button to help users who can't access their account. Finally, the Log In button submits the login form along with their credentials to authenticate the user and authorise their login request.

Patient Account Page

Calculator About Account Name (Patient) Log Out



Patient

Display Name

[Edit Profile](#)

[Settings](#)

Profile

Profile Details	Contact Details
Name	Detail 2
John Edwards	+ Detail 2
	Email Address
	example@mail.com
	Phone Number
	+60140901259

Heart Risk Calculator Trials

Trial 1 - 13/4/2022 13:50

Risk Value: 5%

Inputs

Systolic Blood Pressure: 120
Diastolic Blood Pressure: 90
...

Results info:

...

Done on 13 April 1:50pm

[View Full Info](#) [Print](#)

Trial 1 - 13/4/2022 13:50

Risk Value: 5%

Inputs

Systolic Blood Pressure: 120
Diastolic Blood Pressure: 90
...

Results info:

...

Done on 13 April 1:50pm

[View Full Info](#) [Print](#)

Trial 1 - 13/4/2022 13:50

Risk Value: 5%

Inputs

Systolic Blood Pressure: 120
Diastolic Blood Pressure: 90
...

Results info:

...

Done on 13 April 1:50pm

[View Full Info](#) [Print](#)

[Show All Heart Disease Risk Calculator Trials](#)

Connected Doctor

Dr. x

Registration/TPC No.:

Place of Practice:

University:

Connected on 13 April 1:50pm

[View Full Info](#) [Message](#)

Connect with a Doctor

Dr. x

Registration/TPC No.:

Place of Practice:

University:

Joined on 13 April 1:50pm

[View Full Info](#) [Connect](#)

Dr. x

Registration/TPC No.:

Place of Practice:

University:

Joined on 13 April 1:50pm

[View Full Info](#) [Connect](#)

Dr. x

Registration/TPC No.:


Place of Practice:

University:

Joined on 13 April 1:50pm

[View Full Info](#) [Connect](#)

[Show More](#)

Calculator Sign Up Privacy Policy
About HeartAssist Log In Terms and Conditions


© 2022 HeartAssist. All rights reserved.


Figure 30: Prototype design of the Patient Account Page

Explanation of Patient Account Page

This is the account page of a patient account on the system, typically, what they will see after logging in. First, it displays the Display Name of the account, other account details, how long they have been a member for. Then, is a list of their trials of using the heart disease risk calculator system, in case they would like to compare their results and inputs for reviewing and discussions with their doctors. They can view more information for the individual reports. After that, is the details of their connected doctor, in case they would like to view more information regarding their doctor (View Full Info) or they can also message them (Message). For the current envisioned system, we are planning to only allow the user to connect with one doctor at a time, to prevent the doctors on the platform from being overloaded or spam messaged by many different patient user accounts. After that, is a list of doctors the patient account can connect to, so they can see some general information about the doctor before requesting to connect with them. As another method to avoid the doctor account from being spam messaged by patient user accounts we made sure that a patient user shall request to connect with a doctor, and only after the connection request is approved by the doctor will they be able to message amongst each other to further discuss the results.

Doctor Account Page

 [Calculator](#) [About](#) Account Name (Doctor) [Log Out](#)



Doctor
Display Name
[Edit Profile](#)
[Settings](#)

Profile

Profile Details

Name	Detail 2	Email Address	Phone Number
John Edwards	+ Detail 2	example@mail.com	+60140901259

Contact Details

Registration/TPC Number	Place of Practice	University
X	X	X

Other Details

Connected Patients

Name
Risk Value: 5%

Inputs
Systolic Blood Pressure: 120
Diastolic Blood Pressure: 90
...

Results info:
...

Done on 13 April 1:50pm

[View Full Info](#) [View other trial results](#) [Message](#)

Name
Risk Value: 5%

Inputs
Systolic Blood Pressure: 120
Diastolic Blood Pressure: 90
...

Results info:
...

Done on 13 April 1:50pm

[View Full Info](#) [View other trial results](#) [Message](#)

Name
Risk Value: 5%

Inputs
Systolic Blood Pressure: 120
Diastolic Blood Pressure: 90
...

Results info:
...

Done on 13 April 1:50pm

[View Full Info](#) [View other trial results](#) [Message](#)

Connect with a Patient

Name
Risk Value: 5%

Inputs
Systolic Blood Pressure: 120
Diastolic Blood Pressure: 90
...

Results info:
...

Done on 13 April 1:50pm

[View Full Info](#) [View Full Results](#) [Approve Connection](#)
[Decline Connection](#)

Name
Risk Value: 5%

Inputs
Systolic Blood Pressure: 120
Diastolic Blood Pressure: 90
...

Results info:
...

Done on 13 April 1:50pm

[View Full Info](#) [View Full Results](#) [Approve Connection](#)
[Decline Connection](#)

Name
Risk Value: 5%


Inputs
Systolic Blood Pressure: 120
Diastolic Blood Pressure: 90
...

Results info:
...

Done on 13 April 1:50pm

[View Full Info](#) [View Full Results](#) [Approve Connection](#)
[Decline Connection](#)

[Show More](#)

 [Calculator](#) [Sign Up](#) [Privacy Policy](#)
[About HeartAssist](#) [Log In](#) [Terms and Conditions](#)


© 2022 HeartAssist. All rights reserved.


Figure 31: Prototype design of the Doctor Account Page

Explanation of Doctor Account Page

Similar to the patient account page, this page is what the user sees after logging into their doctor account on the web app system. It starts off with displaying information that the user has entered previously, as well as new information they can add to improve their profile on the system. After that, is a list of their connected patients, which they can choose to see the particular patient's info (View Full Info), view their results (View other trial results) or to message them (Message). Following that, is their list of patients that have requested to connect with them. So, the doctor user can check their info (View Full Info), their recent results from using the system (View Full Results), and either choose to accept or reject the connection request. As mentioned earlier, only after approving the connection request will the messaging be able to take place between the particular doctor and patient account.

Admin Account Page

Calculator About Account Name (Admin) Log Out



Admin

Display Name

[Edit Profile](#)

[Settings](#)

Profile

Profile Details

Name: John Edwards

Detail 2: + Detail 2

Contact Details

Email Address: example@mail.com

Phone Number: +60140901259

Pending Doctor Accounts


No.	Registration ID	Name	Phone	Email	Submission Date	TPC No.	Place of Practice	University	Actions
1	32323333	John Edwards	0182339090	jedw@gmail.com	19-01-2021, 09:31	x	x	x	Approve Reject Email
2	32323334	Jacob Smith	0186738989	jsmith@gmail.com	23-01-2021, 10:23	x	x	x	Approve Reject Email

[View All](#)

Pending Admin Accounts

No.	Registration ID	Name	Phone	Email	Submission Date	Actions
1	32323333	John Edwards	0182339090	jedw@gmail.com	19-01-2021, 09:31	Approve Reject Email
2	32323334	Jacob Smith	0186738989	jsmith@gmail.com	23-01-2021, 10:23	Approve Reject Email

[View All](#)

Calculator Sign Up Privacy Policy
About HeartAssist Log In Terms and Conditions

© 2022 HeartAssist. All rights reserved.

Figure 32: Prototype design of the Admin Account Page

Explanation of Admin Account Page

Similar to the two other account pages, the admin account page starts off by displaying the general information that the admin user has previously entered. Then, there is a table consisting of a list of pending doctor accounts, where the doctor registration forms were sent by the particular user but are pending approval from the system administrators. Hence, each record in the table is a registration request, where the attribute values are the credentials the user used to sign up. Thus, this tabular representation of information makes it easier for the admin to view and verify the entered information to determine if the user is a licensed, valid and practising doctor. They can choose to email the user for further clarification or verification, and then they can either approve or reject the account authentication request. There is a similar table after that with records of users trying to register for a system admin account on the platform. Hence, the existing system admins can check these information entered to determine if the user is a valid system administrator before permitting them to access the system by logging in with their credentials.

3.8 UML Diagrams

The following consists of Unified Modelling Diagrams (UML) Diagrams representing the operations and information and logic flow in the web component of the envisioned ML-based heart disease prediction system, which were constructed with Visual Paradigm. Screenshots or exports of the diagrams are attached below:

Use Case Diagram

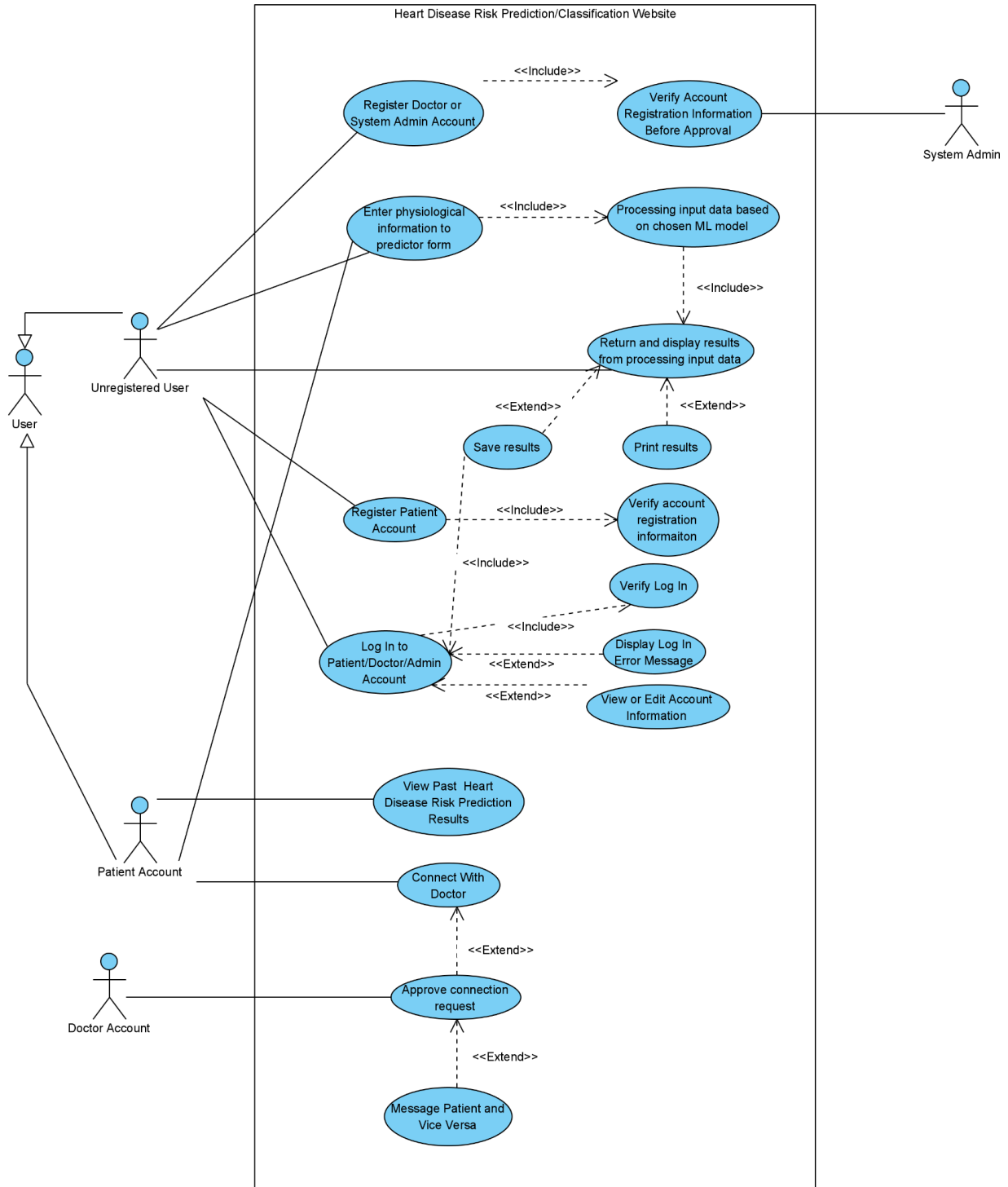


Figure 33: Use case diagram

Explanation of Use Case Diagram

The use case diagram displays the possible main functionalities that are planned for the web app of the ML-based heart disease prediction system to fulfil, as well as illustrating the methods in

which different actors, end-users or stakeholders can interact and utilise the system. So, in a sense it visualises all the possible interactions the user can perform with the web app system. All in all, it provides an overall impression of the system.

Sequence Diagram

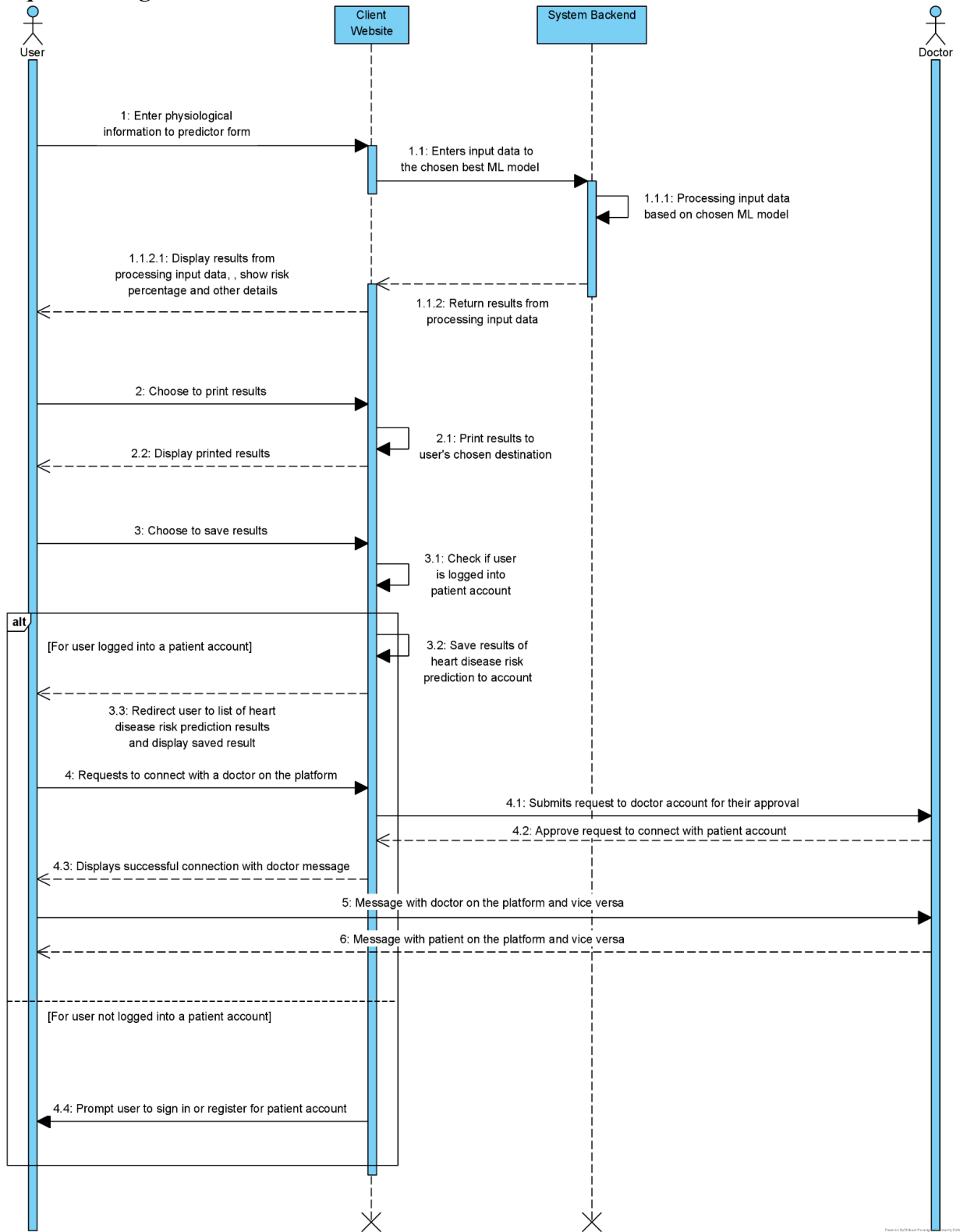


Figure 34: Sequence diagram

Explanation of Sequence Diagram

This sequence diagram is intended to showcase the flow of the processes that the end-user can perform with the system, such as using the heart disease risk calculator form, displaying its results, printing its results, saving the output to their account, and connecting with doctors on the platform.

Activity Diagram

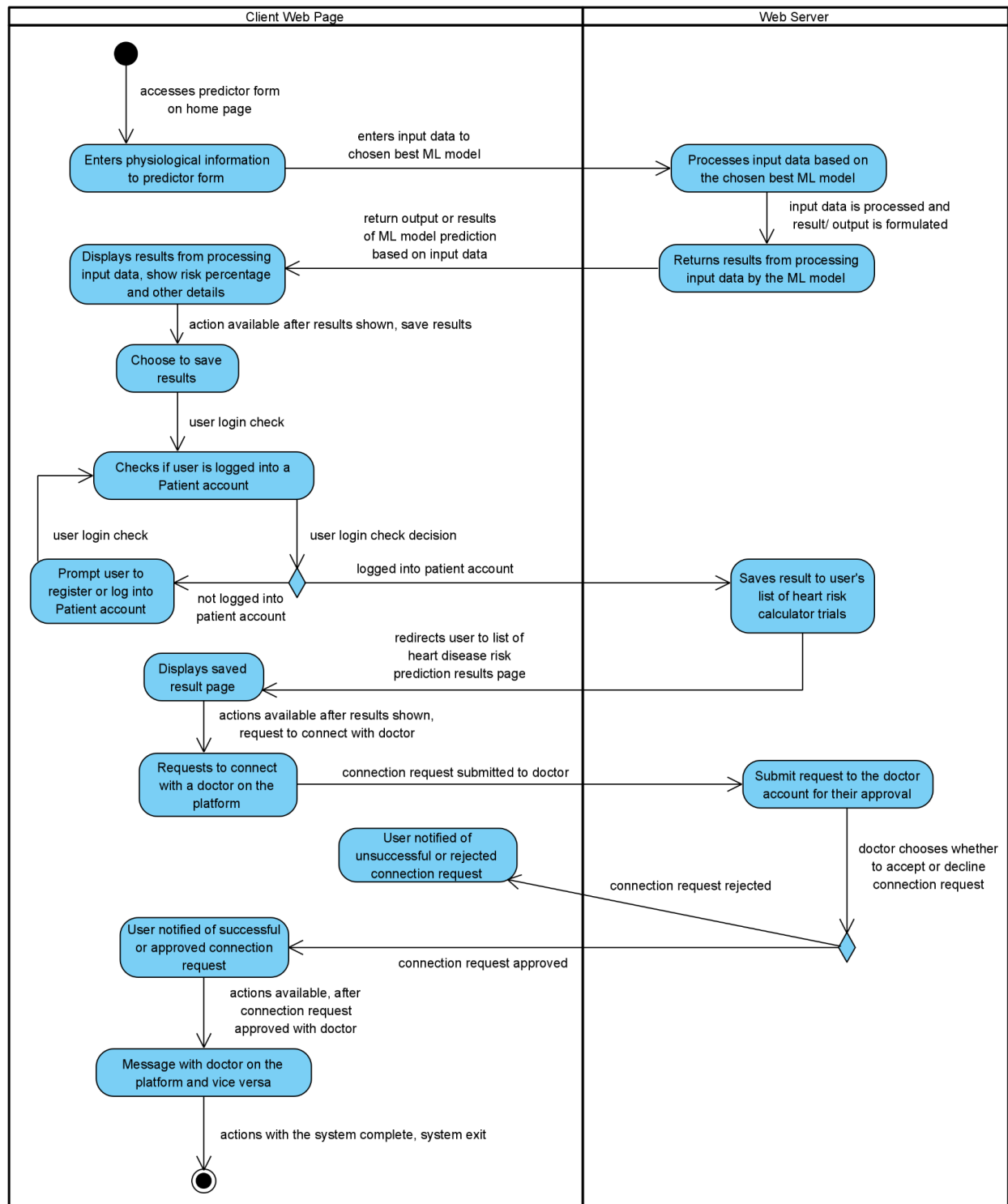


Figure 35: Activity diagram

Explanation of Activity Diagram

This activity diagram illustrates the flow of processes and activities of the possible interactions and operations that the end-user can perform with the web app system. Hence, the diagram

showcases when the user is using or entering data in the heart disease risk calculator form, displaying its results, saving the output to their account, and connecting with doctors on the platform.

Class Diagram

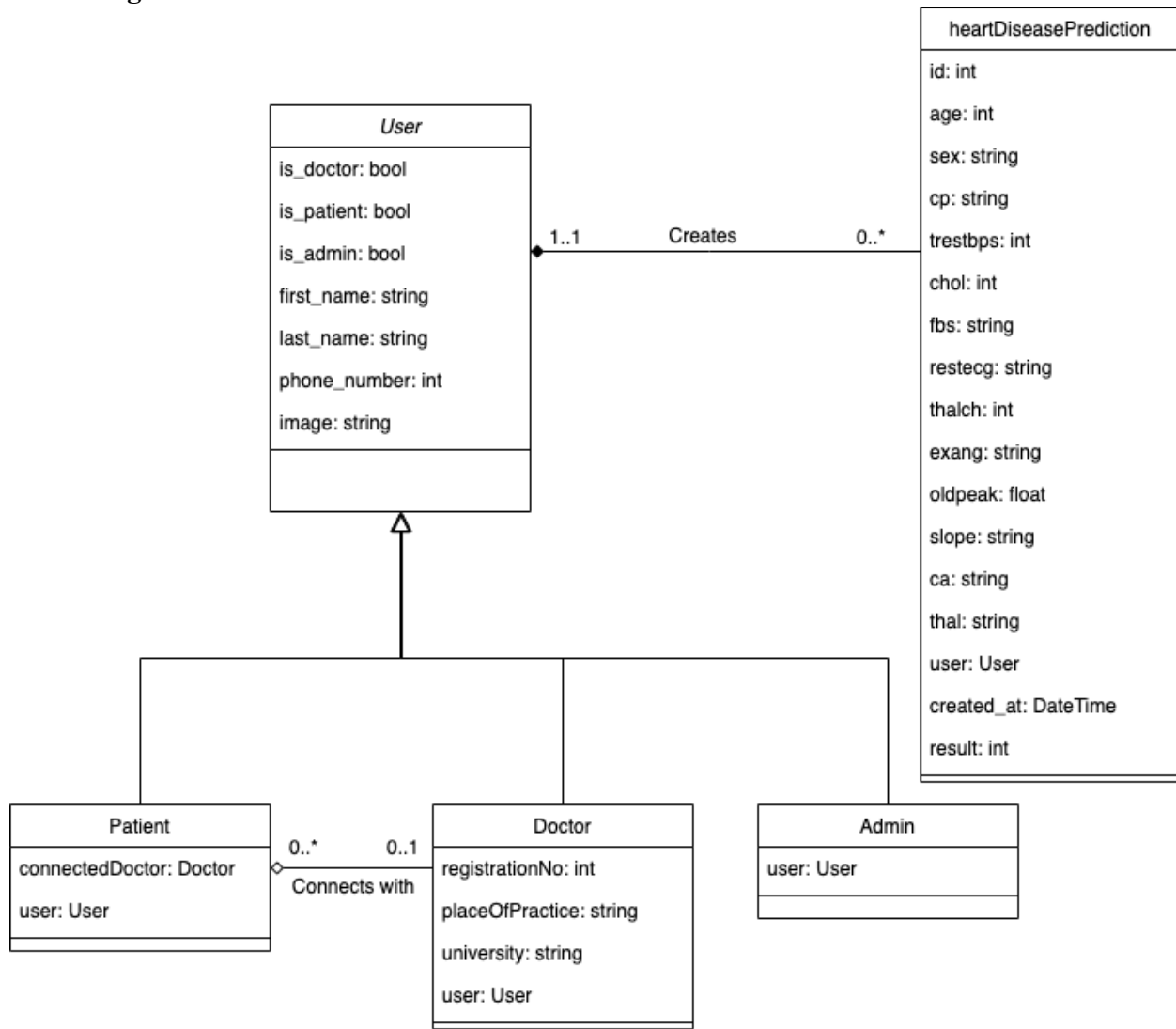


Figure 36: Class diagram

Explanation of Class Diagram

This class diagram illustrates the structure of the database of the system, showing the different classes in the system and how they interact with one another. It helps us to identify the components in the system and their relationship between one another. Also, it enables us to identify the attributes of each class, and so it is easier to implement the models in the `models.py` file of the web application implementation. These classes and what this class diagram represents are explained in more detail in the “Creating User Models With Django” section of the Django Web App Implementation section.

3.9 Web App Implementation

After the ML implementation stage and the model was been developed and chosen, it was integrated into a web application system to allow users to easily access and interact with it through entering the HeartAssist.net Uniform Resource Locator (URL) to their web browser. This will allow them to enter their details into a form on the website which will then be inputted

into the ML model, which will process these inputs and predict the likelihood of the user developing HD or CVD.

Therefore, a simple web application was designed and developed, with functionalities including form saving, submission, user login and registration, user models (patient, doctor) for rendering different types of pages based on the logged in account, and the ability for users to connect with doctors and patients on the platform. Hence, this section describes the main stages to developing and deploying this web application:

1. Django Web App Implementation

Firstly, the Django app was created with the name DjangoAPI, with the main app folder name being DjangoAPI and the secondary folder housing all the custom application files to build the application in the MyAPI folder. This DjangoAPI folder contains the settings file describing the default system configurations, such as the allowed hosts for connection, installed apps on top of Django that were utilised, middleware configurations, and default static and media root files.

The urls.py file for this, points to the urls file in the MyAPI, as it is the directory housing all the custom app files used to customise and build the app pages, to keep things separate. The urls.py file specifies all the available URLs that can be visited or the URLs that are utilised for operations by the web app (e.g., for connection with doctor/patient operation, with the /connect/ url).

a) Creating User Models With Django, MyAPI/models.py

Django Models is the built-in feature of Django that creates tables, their fields, and constraints in the chosen database (in this case, the MongoDB database) based on the defined models in this models.py file (MyAPI/models.py) that need to be created for the web app. So, after defining the classes of objects, the Django application will handle the SQL (Structured Query Language) or in this case, the database commands to store those model data. Thus, this simplifies the tasks of defining the user models that are present on the website, and that users can sign in to (Patient, Doctor, Admin) as well as to create a class for heart disease prediction form submissions (heartDiseasePrediction).

The MongoDB database credentials to store/save these model information have been configured in the DjangoAPI/settings.py file. For this, the Django engine is utilised, which acts as a database connector between the Django application and MongoDB. It is an extension to this Django object relation model (ORM) framework to interface with the MongoDB database. Thus, it essentially makes it easy for this Django application to connect with the MongoDB database, that has been created. Therefore, the database host and the database name information are configured, after setting up and hosting the MongoDB at the Atlas platform. The MongoDB database is hosted in their shared cloud database service, MongoDB Cloud Atlas, in the Amazon Web Services Cloud. The region chosen for this is kept the same as the AWS cloud for hosting the web application, which is ap-southeast-1, Asia Pacific (Singapore). This ensures both the components are hosted in the same region, alleviating latency due to location differences as their both in the same region.

So, the heartDiseasePrediction class which is for creating and storing form submission data has the model inputs as its defined attributes or fields, along with fields for holding the result of the ML model, the user information (User object), and the date and time the form was submitted.

For the user models, there is one abstract User class from which there are the other child classes, Doctor, Patient, and Admin inherit from. Each of these are types of users that the user can log in to, they serve different purposes as mentioned in the requirements section, where patients can connect with doctors to consult with regarding their form submissions. Also, they have different attributes, since they serve different purposes for the particular user type.

Overall, the class diagram in figure 36 describes the relationship between the defined user models on the platform.

The admin user model is the superuser of the system who has administrator privileges to remove users on the platform. This account can be created through the terminal using the command “python manage.py createsuperuser” and then entering the required credentials, whereas for the other two user models, accounts can be created through a simple registration page that has been designed and developed, which will be explored and explained further in the next section. So, the admins of the platform have a separate login page to access the admin actions on the system at <https://heartassist.net/admin>. A user profile on the platform can create and save a heartDiseasePrediction object, which is an object of that class, that stores the form inputs, results and information of which user created it. Thus, it is represented by the composition relationship, when the user model is deleted, its stored form submission are also removed from the database. Non-logged in users can only create form submissions and obtain a result, but the form is not saved in this manner to the database.

b) Defining Website Functionality, MyAPI/views.py

The views.py file (MyAPI/views.py) contains Python functions that receive HTTP requests and then returns a response, based on the purpose of those functions. Thus, it is utilised for rendering various pages on the website and their various functions. So, the following are the custom functions that have been defined and the purpose they play for the website:

- **heartDiseasePredictionAPI**

This user-defined function receives API requests (GET, POST, PUT, DELETE, and PURGE), and performs their corresponding action concerning the heart disease prediction form submission. For instance the GET API request to the website URL, <https://heartassist.net/heartDiseasePrediction>, will obtain all the form submissions stored in the database, the POST API request, creates and saves a form submission (this should be sent along with a JSON dictionary of the form inputs), the PUT request edits or updates previously submitted form submissions, the DELETE request removes a single submitted form submission, and the PURGE request removes all the form submissions stored in the website’s database. Therefore, these API submissions are secured with the csrf token, preventing requests from being made outside the website. The original intention was to make these API calls the mode for creating and deleting form submissions, however, it was found that it was easier and faster to use function calls from within this views.py file for the handling form submissions, especially considering no other frontend framework is used with this web app. Therefore, this API function

was simply used to test whether form submissions that were made on the site were successfully stored in the database, along with Postman, an API platform for building and testing API requests on web applications.

- **heartForm()**

This is the main function for processing form submissions made on the website. The main URL or the home page (<https://heartassist.net/>) as specified in the MyAPI/urls.py file, calls this function so that the heart disease prediction form is rendered and displayed on the home page, awaiting a POST API request after submitting the form. Otherwise, only a blank form, with no inputs is displayed on the home page. The HTML forms on this website, including this heart disease prediction form, the login, patient/doctor registration, and profile edit forms, are handled by Django in the forms.py file. So, as mentioned previously, as there is a model created for the heart disease prediction form submissions in the models.py file to store its fields and inputs, a ModelForm can be created which renders the HTML form automatically for these fields. Thus, the model used and the fields receiving input is specified. Additionally, the widgets field can be used to hide certain fields that are not taking inputs, such as the form result and user, which will be returned after the form has been submitted (for result) and the user is found to be authenticated and logged in (for user). The labels field is used to change or customise the fields for input fields, so they reflect more information regarding a particular form field, guiding the user to know what is the type of information they need to enter there. Additionally in this modelForm class, the other form validation methods (clean_attributeName) are implemented to ensure users enter correct and valid values for the input fields, otherwise, a ValidationError is raised which will display an appropriate error message, guiding the users to know what was the error made in the filled form and how to rectify it.

Therefore, after a user has filled in the form fields and submitted the form, a POST request is made due to the form submission, and the request consists of a JSON dictionary with the field-value pairs. The values that have been entered will be checked to ensure they are clean and not blank otherwise a validation error will be raised as seen in this heartForm() function along with checking with the other form validation functions defined in the forms.py file.

After that, the add array is initialised, which is used to store information regarding the user's form submission and then return additional information to indicate to the user, the fields they filled up that might contribute to the heart disease diagnosis. Thus, indicating the features or the entered parameters that contribute to a user's risk of heart disease and display appropriate tips to mediate their condition.

Following that, the request or form submission values is run through the ohevalue() function, which converts the categorical attributes of the form inputs to dummy attributes and their corresponding values, to match the input fields of the ML model. Thus the form input values can be directly entered into the ML model as inputs. After that, this processed form input is entered into the heartResult function which is the ML model function. Hence, it starts by loading the chosen best ML model from the ML implementation section that was pickled, using the Joblib method load(). Then, the scaler (MinMaxScaler) that was previously fitted with the dataset is also loaded and used to transform the form inputs based on the same configurations as in the ML implementation, to ensure consistency and the values are scaled accordingly. After that, the

transformed form inputs are inputted into the loaded ML model and used to make predictions (y_{pred}). This computed y_{pred} is stored in a Pandas DataFrame and returned to the initial function call, to display the results of the ML processing of the form inputs to the end-user.

The context dictionary is used to specify the Python variables that have been instantiated and the identifiers to use to call and render them in the Django HTML template.

After that, the result of the form input processing is added to the result field of the form submission request. If the user is authenticated and logged in, the User object of the logged in user is added to the user field of the form submission request and the form is saved to the database under their user profile. Thus, it can be shown on their account page.

- **register()**

This function is for rendering the register.html file and called when the <https://heartassist.net/register/> URL is requested. From here, users can select whether they want to register as a patient or doctor, and the corresponding registration will be rendered, by calling the appropriate function, either `patient_register()` or `doctor_register()`. These 2 functions perform similarly by creating and rendering the corresponding form class, `DoctorSignUpForm` or `PatientSignUpForm` as specified in the forms.py file. This form is created with the use of the `UserCreationForm` method of the `django.contrib.auth.forms` library. Thus, this form possesses the logic to handle the authentication of new users based on the custom fields that are specified. So, after the user profile has been authenticated and created, the profile will be saved to the database, as seen in those 2 classes in the forms.py file and they will be automatically logged in and redirected to the home page.

- **login_request() and logout_view()**

The `login_request()` function is for rendering the login form to the user. For this the `AuthenticationForm` method of the `django.contrib.auth.forms` library is displayed which possesses the logic to handle the authorisation of login requests from users and handle the user sessions, so that the user stays logged in even if they close the website and revisit it later on, until they have requested to log out or have cleared their cookies/sessions on their browser. Similarly the `logout_view` function as the name suggests logs out a user from their account by calling the `logout` method of the `django.contrib.auth` library, and redirects them to the home page.

- **account() and change_connection()**

The `account()` function renders a different account page HTML file, for the <https://heartassist.net/account/> URL with different information that is presented based on the type of user that is logged in. For instance, if they are a patient account, the “`account_patient.html`” is rendered, and the information that is rendered include the patients information (name, profile details, contact details), heart disease form submissions, and list of connected doctor or doctors on the platform to connect with. Otherwise, for a doctor account, the “`account_doctor.html`” is rendered and displayed, starting off similarly by showing the information of the doctor (profile details, contact details, and other medical professional details), heart disease form submissions, and list of connected patients. Else, for an admin account, the <https://heartassist.net/account/> URL will redirect them to the admin page where they can perform their admin actions including managing users and form submissions on the platform as well as changing their profile details

such as account password. For this, page the default Django administrator page was utilised as it provided all the functions that were needed for an admin user of this system as described in the requirements, and implementing it was more efficient and more attention can be put to the other more crucial aspects such as developing and improving the ML model.

Some special functionalities that are implemented in this account page include, for patients, they can choose to connect with a doctor, and the doctor will be added as their ConnectedDoctor. This change will be reflected in the particular doctor account as well, where the patient will be added to their list of connected patients. From there, patients can choose to contact the doctor using the contact information that is presented. They can also remove the doctor and choose a different doctor account by selecting the Remove button and likewise, the change will be reflected in the doctor's list of connected patients. Doctors also have similar controls where they can remove a patient from their list of connections. These connection operations were handled by the `change_connection()` function in the `views.py` file, where after selecting the connect or remove button it will visit a URL where the operation selected (either add or remove) and the user's primary key will be added to this URL. Calling this URL will initiate the `change_connection()` function, and the operation and the user's primary key will be passed in as inputs to perform the corresponding action on the particular user. Therefore, the remove operation will remove a doctor from a patient's list of connected doctors (change will be reflected on the corresponding doctor's side as well), the remove-from-doctor operation will remove a patient from a doctor's list of connected patients (change will be reflected on corresponding patient's side as well), the add operation will add the particular doctor to the patient's list of connected doctors (change will be reflected on corresponding doctor's side as well). For doctors, they have an additional button to view their patient's list of heart disease prediction form submissions, this will call this `change_connection()` function and call the `view-trials` operation to return and rerender the account page with that particular user's form submission information displayed.

- **`edit_account()` and `changepassword()`**

The `edit_account()` function is called by the <https://heartassist.net/account/edit/> URL and renders the `edit_account.html` page. It renders the `EditProfileForm` and `ProfileUpdateForm` from the `forms.py` file, the first to display fields of the Django default user model and the second to render additional fields of the custom User model in the `models.py` file. Therefore, this page has a combination of the fields to allow users to change their profile details, and they can then submit the form after form validation to make and apply the changes to their profile in the database. Through here, they can also add an image to their user profile.

Similarly, the `change_password()` function is called by the <https://heartassist.net/change-password/> URL and renders the `PasswordChangeForm` from the `django.contrib.auth.forms` library, which also provides the backend logic to verify the old password entered and implement and save the new password to the Django user model.

2. Developing the Pages of the Web App Using Django

For most of the above mentioned functions in the `views.py` file, it renders a corresponding HTML file. Django templating is utilised where the HTML files contain Django markup, such as variables surrounded by `{{` and `}}` for variables as well `{%` and `%}` for tags for arbitrary logic in the rendering process, primarily for “if” statements and “for” loops. This makes it easier to make

the HTML pages dynamic, so that results can be presented after a form has been submitted or to present account information in the user's account page.

Therefore, the HTML files or the templates created for this web application are stored in the MyAPI/templates folder. Here, the headerPage is the main skeleton consisting of the header or navigation bar and the footer, which is consistently the same on every page. The navbar consists of the site logo (pointing to the home page), the currently logged-in user and log out (if authenticated) or the login and registration buttons (if not authenticated). Then, the content of this template, will house the other HTML files that will be rendered, depending on the URL that is entered and subsequently the views.py function that is called.

To make the frontend editing and designing quicker with better design elements (e.g., tooltips, button animations, table layout, etc.), the HTML pages were designed using TailwindCSS and DaisyUI classes. So, to style a HTML element a predefined class following both library's documentation was assigned to it to achieve the desired design. Therefore, through referring to the documentation and through rounds of trial and error the ideal design for a particular HTML element was found and implemented. Overall these two technologies work hand in hand to make designing the frontend of the web application effortless and more rapid, so that more attention and time can be placed to the more important aspects of the project, especially designing the backend and integrating the ML implementation.

3. Deploying the Django Web Application

Finally, after all the requirements and system functionalities of the web application were implemented, it was deployed to the AWS EC2 instance, which is currently hosting it. The goal here is to make the web application easily accessible by the end-user, simply by entering a URL into their search engine. Hence, making the application production-ready as it can be accessible from anywhere, at all times, at an acceptable loading speed, these are part of the aforementioned non-functional requirements that the system was desired to achieve.

a. Setting up Gunicorn and Nginx

The first part of the deployment process was setting up Gunicorn and Nginx, which as mentioned before are HTTP web servers for this Python Django web application. Therefore Gunicorn was utilised as the Web Server Gateway Interface (WSGI) web server for the application by first installing its package and then the Gunicorn server process can be invoked, initiated or used to build the application by calling its WSGI application object in the DjangoAPI/wsgi.py file, and binded to the localhost's port 8000 using the command "gunicorn DjangoAPI.wsgi:application --bind 0.0.0.0:8000". This is used in preference to the default development server that ships with Django, as it can handle more requests as in a production environment and can handle all the applications' dynamic files. Along with that, Nginx was set up and configured as a reverse proxy for Gunicorn to serve the application's static and media files. So, the Nginx configurations can be seen in the nginx directory in the default.conf file. The Dockerfile in that directory consists of the steps to set up the Nginx configurations and will be initiated when the docker image is built in the following step.

b. Dockerising the Django Web App

So, the next step of the deployment was to “dockerise” or wrap the Django web application in a docker container. To implement this, the Dockerfile was created (found in the DjangoAPI directory) which provides instructions to build the web application. So, it starts by importing the Python 3.8-slim-buster image or environment as the web application is built with Django which is Python-based. This will serve as the OS that is delivered with the web app to serve as the OS or environment to run the app. Then, the contents of the requirements.txt file are copied and installed, which contains the libraries and dependencies required to run the application. After that, the aforementioned Gunicorn command was utilised to build the Django web application. With this, the Django web application can be built on any server or hosting platform after the system files have been added there by executing the command “docker build -t heartpred .” and then “docker run -p 8000:8000 heartpred” to bind the application to the localhost’s port 8000. Additionally, the docker compose tool was used to help define and share the container of the web application. This will make it more easier to deploy and rebuild the Django web application on the AWS EC2 instance, simply by calling the “docker-compose up --build” command. It was used to create the YAML file, docker-compose.yaml, which specifies the instructions to set up the Nginx configurations pointing to the application’s static and media resources and the previous Dockerfile to build the Django web app.

c. Uploading the Source Code Base to a Github Repository for Version Control

Following that, these source code of the program was added to a Github repository used for version control of the web application. Sensitive information, such as the security key were added as an env variable and ignored from being pushed to the Github repository (added to .gitignore) so random unauthorised visitors cannot access it from the Github repository. By adding the program source code to Github, this allows to control the version of the program, reverting back to previous builds or versions, in the event there is a broken feature, errors or bugs in the platform. Thus, making development safer and more structured as changes or commits to the code base will only be pushed to the main branch, once it is found to be functional in the local testing environment. Additionally, by having the codebase in a Github repository, it can be easily cloned anywhere, in this case, more importantly at the hosting environment in the AWS EC2 instance.

d. Configuring the AWS EC2 Hosting Environment

Therefore, the next step is to setup and configure the hosting environment at AWS with an EC2 instance. For this, after the EC2 instance had been created, using the terminal the EC2 instance was accessed remotely using Secure Shell Protocol (SSH). First, the Github repo was cloned in this new environment. After that, using the aforementioned configured docker-compose command, the Django web application was built and tied to the EC2 instance’s public IP address’s 8000 port. Then, after the application was found to be accessible and functional using the public IP address, an elastic IP address was configured, which is a static IPv4 address designed for the EC2 instance. This allows the IP to remain static and constant, even if the server is stopped temporarily, preventing issues down the line when the server is halted for some reason.

The server was migrated to the same region as the MongoDB database hosted in the Amazon Web Services Cloud at the ap-southeast-1 region in Asia Pacific (Singapore). For this, an image of the server was created, and then in the new region, a new EC2 instance was created or

replicated based on the previously created image. After that, the docker-compose commands were initiated to rebuild the web application. After performing this migration, the speed of the site page loading and changes or updates to the database saw a significant increase.

e. Setting Up Cloudflare for Linking Domain Name and DNS Management

The next step was to register the domain name, heartassist.net, with Cloudflare as the domain name registrar. By registering with Cloudflare, it was used as the domain name system (DNS) management service to manage DNS records of the domain name and the web application.

Therefore, this allows for the heartassist.net domain name to be attached to the AWS EC2 instance public elastic IP address, by creating an A record for the heartassist.net domain name with that IP address as its content. This allows users to visit and access the web application simply by entering heartassist.net into their search engine, instead of having to enter the unintuitive IP address and port number. The traffic to this domain name is also proxied to improve the security and performance of the web application.

Besides that, other settings were configured in the Cloudflare settings or dashboard. For instance, some features of Cloudflare that were utilised includes the Secure Sockets Layer/Transport Layer Security, SSL/TLS certification service, which improves the security of the website by encrypting HTTP traffic to and from the webservers with SSL, which also has search engine optimisation (SEO) benefits, owing to the site's improved security. Cloudflare handles the SSL certificate lifecycle to extend security to the site visitors. Besides that, Cloudflare is used as a CDN to deliver the website and its assets in the form of cache over its distributed network of web servers, thus improving the website's performance and load times, wherever in the world it is loaded up at. Along with that, Cloudflare improves the security of the website by protecting the site against DDoS attacks, bots, targeting APIs, and excessive requests to the web server. These optimisations were seen to make a significant improvement to the website's page loading speeds and navigation between the pages when tested with GTMetrix and Google PageSpeed Insights, with time to functional being less than a second.

f. Configuring Jenkins as the CI/CD component

Finally, Jenkins was set up and configured on the AWS EC2 instance to allow Git commits to be deployed with this hosted application. So, for this, the Jenkinsfile was created in the DjangoAPI directory, which contains the pipeline with instructions to build the Django Web Application using the aforementioned docker-compose function.

This pipeline consists of a sequence of required tasks to build and run the Django web application. Thus, more specifically, the Jenkins CI/CD pipeline connects the Django web application hosted in AWS to its Github code repository, so when a new commit, update or change is developed and made to the main branch of the repository, this CI/CD pipeline facilitates continuous delivery or the testing of this new commit, building and packaging code, production staging and environment (continuous integration), and once the tests are passed it automates the final step which is to deploy the new updates by building the new software with the application or deploying the updated code (continuous deployment).

Overall, Jenkins pipelines quickly find bugs in a code base, creates the software, automates testing of builds, get the code base ready for deployment (delivery), and finally deploy the code to the cloud server, in this case, the AWS EC2 instance. The Jenkins dashboard can be accessed by visiting the public ipv4 addresses 8080 port at <http://54.179.245.143:8080>.

4. Results and Discussion

This section presents the results of the testing performed on the ML models and the web application, along with providing an explanation or discussion for these results, while relating back to the project's initial objectives and goals to determine if predetermined requirements have been fulfilled. Thus this section is divided into those two sections, catering to the ML algorithm study of the project and validating the web app implementation.

4.1 ML Implementation Results and Discussion

The evaluation of the ML algorithms was performed using a laptop equipped with an M1 Pro processor and 8.82GB usable main memory. The evaluation metrics used to evaluate the ML models include accuracy, precision, recall, f1-score, MSE, RMSE, MAE, misclassification rate, training time, testing time and total time. Each of these metrics are explained in more depth in the methodology section under model validation, explaining what the individual evaluation metrics describe, denote, convey, and how they are measured or implemented in the program (using the user-defined `modelValidation()` function).

The full table of the ML model evaluation metric results are available in Appendix A for reference. Throughout the ML implementation stage there are a number of manipulated variables that were studied, to see their effect on the model performance. So, they were manipulated to find the optimal values that maximise the model's prediction accuracy while minimising its error rate. Therefore, these manipulated variables can be narrowed down to the dataset features (final features/attributes that were utilised can be found in 3.4.1 under the Methodology section), the data preprocessing techniques used (the different data preparation methods were used and the specific final chosen technique can be found in 3.4.3 under the Methodology section), split ratio (tested Train:Test ratios include, 50:50, 60:40, 70:30, 80:20, and 90:10), the model hyperparameters (the tested and finally chosen hyperparameters can be found in 3.4.5 under the Methodology section). The table in Appendix A, presents the evaluation metric results of the different ML models when the different split ratios are used, when the features used, preprocessing performed, and hyperparameter values are the aforementioned in the Methodology section.

To narrow it down, and for easier reference, the following table presents the evaluation metrics of the ML models when the split ratio is 80:20, which is found to generally produce the best-performing models. This table contains the best performing model, which is the "StackingClassifier model with random forest" as its final estimator, since it has the highest accuracy, precision, recall, f1-score and MCC scores, while minimising on MSE, RMSE, MAE, misclassification rate, training and testing time, although its not the lowest in terms of these metrics. Therefore, the following sections explore the underlying information from the comparison of the different split ratios, prediction accuracy results (Accuracy, Precision, Recall, F1-Score, MCC), error rates (MSE, RMSE, MAE, and Misclassification Rate), training time and testing time of the different models. This will help solidify the reasoning behind choosing the stacking classifier with the random forest as the final estimator model as the chosen model to be deployed.

Table III
Results of ML model validation and evaluation metrics for 80:20 Train:Test split ratio

Algorithm	Parameters	Accuracy	Precision	Recall	F1-Score	MSE	RMSE	MAE	Misclassification Rate	MCC	Training Time (s)	Testing Time (s)	Total Time (s)
SVC	kernel: 'rbf' C: 20 gamma: 1	0.841849	0.841849	0.841849	0.841849	0.40146	0.633609	0.226277	0.158151	0.802512	0.10457	0.026688	0.131258
KNN	n_neighbors: 3 weights: 'distance'	0.783455	0.783455	0.783455	0.783455	0.605839	0.778357	0.323601	0.216545	0.729999	0.000291	0.008914	0.009205
NB	var_smoothing: 0.43287612810830584	0.476886	0.476886	0.476886	0.476886	1.766423	1.329069	0.86618	0.523114	0.349082	0.000575	0.00027	0.000845
DT	max_depth: None criterion: 'gini'	0.683698	0.683698	0.683698	0.683698	0.990268	0.995122	0.498783	0.316302	0.604506	0.013766	0.000143	0.013909
XGB	n_estimators: 100 max_depth: None subsample: 0.75	0.844282	0.844282	0.844282	0.844282	0.40146	0.633609	0.226277	0.155718	0.806121	1.885593	0.001801	1.887394
RF	n_estimators: 200 max_features: 'auto' max_depth: None	0.83455	0.83455	0.83455	0.83455	0.532847	0.729964	0.270073	0.16545	0.793823	0.372334	0.013986	0.38632

	critierion: 'gini'												
EXT	n_estimators: 200 max_features: log2 max_depth: None critierion: 'gini'	0.836983	0.836983	0.836983	0.836983	0.476886	0.690569	0.253041	0.163017	0.796523	0.23796	0.017439	0.255399
KMC	n_clusters: 200 max_iter: 150 algorithm: 'lloyd'	0	0	0	0	10320.1	101.5879	84.25791	1	-0.00849	0.702787	0.000468	0.703255
LR	C: 1000.0 solver: 'lbfgs'	0.527981	0.527981	0.527981	0.527981	1.347932	1.161005	0.715328	0.472019	0.409429	0.026302	0.0000913	0.026393
ANN	activation: 'tanh' solver: 'adam' hidden_layer: (27, 27, 27) max_iteration: 723	0.785888	0.785888	0.785888	0.785888	0.518248	0.719895	0.309002	0.214112	0.732452	2.056033	0.001495	2.057528
StackingClassifier with SVC Final Estimator	final_estimator: svc_model	0.798054	0.798054	0.798054	0.798054	0.476886	0.690569	0.277372	0.201946	0.747648	25.07528	0.100786	25.17607
StackingClassifier with KNN Final Estimator	final_estimator: knn_model	0.827251	0.827251	0.827251	0.827251	0.396594	0.629757	0.23601	0.172749	0.783898	25.00379	0.069261	25.07305

StackingClassifier with DT Final Estimator	final_estimator : dt_model	0.793187	0.793187	0.793187	0.793187	0.420925	0.648787	0.270073	0.206813	0.741486	25.22474	0.064495	25.28924
StackingClassifier with XGB Final Estimator	final_estimator : xgb_model	0.86618	0.86618	0.86618	0.86618	0.282238	0.531261	0.180049	0.13382	0.832577	26.86886	0.070001	26.93886
StackingClassifier with RF Final Estimator	final_estimator : rf_model	0.868613	0.868613	0.868613	0.868613	0.311436	0.558064	0.184915	0.131387	0.835799	26.60936	0.081212	26.69057
StackingClassifier with EXT Final Estimator	final_estimator : ext_model	0.863747	0.863747	0.863747	0.863747	0.309002	0.55588	0.187348	0.136253	0.82951	29.84438	0.093992	29.93837
StackingClassifier with MLP Final estimator	final_estimator : mlp_model	0.817518	0.817518	0.817518	0.817518	0.396594	0.629757	0.245742	0.182482	0.772953	31.47315	0.112023	31.58517

Comparing Split Ratios

Comparing the evaluation metric values and performances at different split ratios, it is seen that the best tested split ratio is 80:20, where it contains the top 5 best performing ML models, namely the StackingClassifier with RF, XGB, EXT, and normal XGB, and SVC models. From viewing the performance metric values, we can rank the split ratios in this descending order, 80:20, 90:10, 70:30, 60:40, and 50:50. Thus, a split ratio of 80% training data and 20% testing data is found to be the right balance to minimise overfitting and underfitting, so that the ML models are not overly fitted to the dataset, to the point they cannot generalise to actual real-world data (overfitting) and they are well trained enough to possess acceptable prediction accuracy (not underfitting).

Comparing Accuracy, Precision, Recall, F1-Score and MCC Scores

With the split ratio fixed to 80:20, we compare the ML models performance metrics within that subset of the table. Here, we can see that the best performing models in terms of accuracy, precision, recall, f1-score and MCC scores, in descending order are the StackingClassifier with RF, XGB, EXT as the final estimator, and the normal XGB, and SVC models. Therefore, the StackingClassifier with Random Forest as the final estimator being the best-performing model in this respect, earned its place as the best-performing model with the best prediction accuracy. Additionally, we can see that the stacking classifier objects are significantly better in terms of these metrics as expected and explained when viewing the structure and operation of the StackingClassifier object in the Methodology section, where it is an ensemble model that combines the outputs of multiple strong ML models and discovers the most effective orientation using its meta-learning algorithm to deliver a model that is capable of achieving even higher accuracies. Thus, acting as a hybrid model, combining the strengths of different ML models to form one strong model. Moreover, it is believed that the improved prediction accuracy is achieved by placing RF as the final estimator as it is a simple ensemble model (as previously discussed during the Methodology and Literature Review sections) that complements the structure of the stacking classifier. Other notable models include the XGB, SVC, EXT, RF, and ANN, KNN models, which performed very well and had really good prediction accuracy (almost all being > 0.8). Here, we can see that after performing hyperparameter tuning to obtain the optimal model parameter values, these scores were significantly enhanced. The effect of choosing the optimal preprocessing techniques, for instance, one-hot-encoding over numerical encoding, KNN Imputer over other simple imputation techniques (other tested include replacement with mean, mode, median values), implementation of oversampling with SMOTE, MinMaxScaler over other scaling techniques (other tested include StandardScaler, MaxAbsScaler, RobustScaler), was seen to significantly improve the final prediction accuracies and other evaluation metric scores as well. Models that did not perform well in this respect are the LR, NB and KMC models, with the latter achieving significantly low prediction accuracies. This is because of the nature of the KMC model being an unsupervised model that does not take advantage of the information of target values that are available in the dataset to train its model, instead finding patterns in the dataset with the input attributes to make predictions or classify records. For the NB and DT models, they are believed to have comparatively lower scores because they are simple machine learning algorithms either based on computing the probability based on Bayes principle or by performing simple regression for classification purposes.

Comparing MSE, RMSE, MAE, and Misclassification Rate

In terms of the lowest MSE, RMSE, MAE, misclassification rate, in ascending order, are the StackingClassifier with XGB, EXT, RF, KNN, and MLP models. Here, the reasoning behind these scores are the same as for the prediction accuracies, where StackingClassifier objects are hybrid ensemble models combining the strength of its base models with a designated final estimator. Although the StackingClassifier with RF final estimator is not the best model in this respect, the difference is very minimal (± 0.03) with the StackingClassifier with XGB and EXT models. The StackingClassifier with RF final estimator is still the chosen model, as its prediction accuracy is significantly better (0.868) (± 0.04 difference) and its testing time is significantly low (being 0.02s faster/lesser time taken), which is significantly important considering the model will be deployed into the production environment and its crucial for it to process these inputs in a low amount of time and return accurate predictions. Additionally, the MLP model is also seen to have low errors, owing to its fundamental structure of iteratively improving and reducing the model errors at every epoch or training run by optimising the weights and thresholds at the neurons or perceptrons. Similar to before, the worst-performing models in this respect are again the LR, NB and KMC, with MSE and RMSE more than 1 and significantly high MAE and misclassification rate. This is because, as mentioned before LR and NB are less complex models compared to the other more performant models, and KMC does not fit this problem domain which requires more supervised learning models.

Comparing Training Time

In terms of training time, the fastest models or the models requiring the least time in ascending order are the normal KNN, NB, DT, LR, and SVC models. Similarly, the models requiring the most training time are the StackingClassifier models in descending order are StackingClassifier with MLP, EXT, XGB, RF, and DT. As mentioned earlier, the StackingClassifiers are hybrid ensemble models containing multiple base models/estimators, thus as each of the underlying classifiers need to be fitted and trained as well as the meta-learning algorithm to determine the best orientation or combination, this can extend the time taken to train the model. However, this training is only performed once as the model is fitted with the training dataset, thus, it will likely not be an issue in a production or real-world environment that relies more on the testing time. In the web application, a fitted model with the computed weights and parameter values will be used to inputs will be entered and passed through the model to compute the output, thus only relying on the testing time or speed at which the model makes predictions. Therefore, in this respect, it is seen that the KNN, NV, DT, and LR models require the least training time as they are simple, non-complex non-ensemble supervised learning methods. Therefore, models such as the EXT, RF, and XGB require more time for training in comparison, as they are ensemble models or a combination of these simple models, primarily the decision trees. The ANN model also requires significant training time, considering its structure of requiring numerous epochs or training runs to train the model and optimise its weights and threshold parameter values, and this value increases with the structure of the neural network, where if it contains more hidden layers and neurons in those layers, this increases the complexity and the time taken to fit all the neurons and the overall model. Thus, this value was optimised, and (27,27,27) was found to be a perfect balance, giving an improved and stable performance or accuracies while reducing the complexity or time taken for training the model.

Comparing Testing Time

In terms of testing time, the fastest models or the models requiring least time in ascending order are the normal LR (0.0000913s), DT (0.000143s), NB (0.000270s), K-Means (0.000468s), ANN (0.00149s). This is as expected and, as mentioned previously, where these models are typically of low complexity. As for the ANN model, once it is fitted and the optimal weights and threshold values have been computed for its neurons, the numerical values are available, and the process of computing or making predictions and classifications is akin to involving simple multiplication calculations to compute the output. Similarly, the models requiring the most testing time are the StackingClassifier models in descending order are StackingClassifier with MLP (0.112s), SVC (0.100s), EXT (0.094s), XGB (0.070s), and RF (0.052s) owing to its complex structure and as inputs need to be passed through each of the individual base estimators to produce a final highly-accurate classification or prediction. Nevertheless, these testing times are still around and less than 0.1s, so in the web application, real-world usage or production environment, this time is highly unnoticeable to the end-user, and as it is a value close to regular page rendering times which will mask it.

Thus, the StackingClassifier with RF as final estimator is still kept as the chosen model, because it has a significantly low testing time among the StackingClassifier models (0.052s), while taking advantage of its significant prediction accuracy and minimal error rate.

Feature Importance

After choosing StackingClassifier with RF as the final estimator as the best performing model, its feature importance or the contribution score of each feature to the model prediction is identified and visualised. Thus, the following figure and graph show the score computed of the feature importance and the visualisation of these scores in a bar chart to observe the differences better.

```
#####\Feature Importance
oldpeak          0.087283
age              0.087054
thalch          0.085115
ca              0.082517
chol            0.080429
trestbps        0.072603
cp_asymptomatic 0.056347
exang_False     0.045068
slope_flat      0.041124
restecg_lv hypertrophy 0.038881
exang_True      0.035051
restecg_normal  0.034193
thal_reversible defect 0.033795
fbs_False       0.028939
slope_upsloping 0.022799
restecg_st-t abnormality 0.021184
fbs_True        0.019957
slope_downsloping 0.019613
cp_non-anginal  0.018533
thal_normal     0.018499
thal_fixed defect 0.017829
cp_atypical angina 0.016274
sex_Male        0.016201
sex_Female      0.014909
cp_typical angina 0.005805
dtype: float64
```

Figure 37: Output of feature importance score of dataset attributes in the terminal

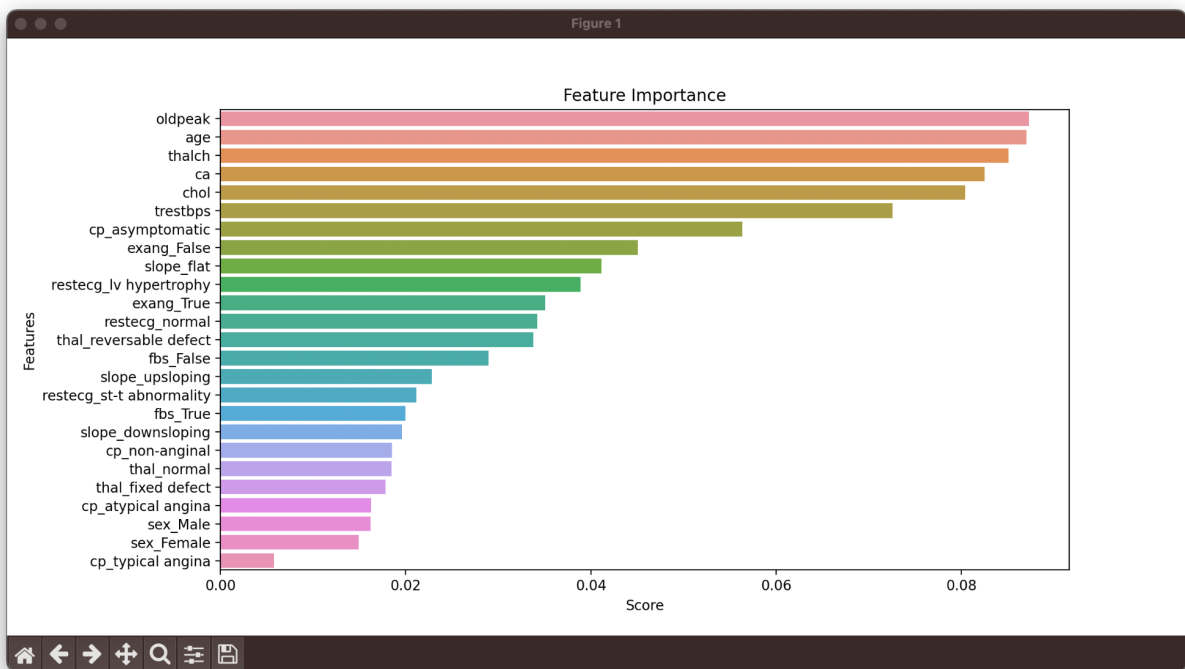


Figure 38: Visualisation of feature importance score of the dataset attributes in a bar chart

From the above results and plot, it can be seen that the features that have the most effect on the model are the oldpeak (ST depression induced), age, thalch (Maximum heart rate achieved), ca (Number of major vessels), cholesterol, and trestbps (Resting blood pressure). These are numerical attributes and so they are not split into multiple sub-attributes like with the categorical attributes as per the action of the one-hot-encoding during the data cleaning stage. Thus, this is why these attributes have particularly high feature importance. Rightly so, we can see that the oldpeak, age and heart rate of an individual are key factors for heart disease presence in an individual, especially oldpeak as it directly corresponds to the ECG results, which are a key measure when dealing with heart disease detection.

Furthermore, for the categorical attributes, chest pain type, exercise-induced angina, slope of ST segment, and rest ECG are key contributors, especially considering chest pain type and angina are common symptoms that are monitored, and the latter are part of the ECG test data, a common measure for heart disease presence. Gender is seen to be among the lowest contributors, which is fairly true, as previous literature review has shown that one's gender does not strongly impact or polarise their level of heart disease, where other parameters would need to be placed more priority to be observed.

Comparing With Literature Sources

From comparing the performance metric results obtained with the ones presented by the other literature sources (as seen in the Literature Review section), there are a few things that can be noted. Firstly, our best-performing models are able to outperform the best-performing models developed by [37], [7], [38], and [40]. This is owing to the development or usage of advanced ML models or techniques such as the StackingClassifier, ExtraTrees, and XGBoost models that are not employed by the other sources. Additionally, implementing hyperparameter optimisation to find the optimal parameter values to maximise the prediction accuracy of the ML models significantly improved our ML models, and is not practised in a large majority of the other literature sources.

Nonetheless, sources such as [9], [36] and [39] were seen to obtain better performance metric scores. For sources [9] and [36], the best models are their SVC (97.35%) and RF models (94.96%) respectively, which had one of the highest accuracies in our implementation of those models as well. However, their significant improvement in those models is possibly due to them using a subset of the UCI dataset, where our dataset used was a combination of 4 datasets. Thus, their dataset had lesser records to train, and so in conjunction, lesser records to validate or test with, potentially giving way to lesser errors in the dataset to rectify and a higher percentage of accurate classifications. Additionally, they each used a different set of numerical only attributes. Therefore, this could have played a role as the ML algorithms can learn better from the dataset to improve its fitting and, in conjunction, its prediction. So, in our future works or implementation of our ML models, we would have to test training and testing the model with only numerical attribute values. For source [39], their improvement came from implementing the CNN model, which gave them a prediction accuracy of 94.78%. This algorithm was left out of our comparative study to focus on the existing 11 different algorithms, to understand their structure

and optimise those models better. However, it is surely a model that will be placed as part of future works to be designed and implemented with our obtained datasets.

4.2 Web Application Results and Discussion

This section describes the results of testing that was performed on the web app after its development had been completed. Overall, it is important to ensure that the web system with the integrated ML component is thoroughly and systematically tested to ensure the initially mentioned goals and requirements have been fulfilled and the system is functional with little to no errors in its operations or functionalities, before the system is cleared to be completely deployed and utilised by the end-users. This is to prevent users from encountering errors with the website and to ensure the user experience of the website is conserved. From the testing performed, if any errors are found in the developed program, revert back to the development stage to proceed with identifying and pinpointing the error and its source, and subsequently, efforts were made to resolve the issue. Hence, this step was interleaved or conducted along with the development phase, so if any defects, faults, or errors are identified with the developed system, reversion to the implementation stage is possible to debug and resolve the errors.

Following the project's implementation, two forms of testing were performed on the web app, namely, unit testing and integration testing were carried out. These tests were performed on the locally-hosted web application, before the deployment tasks and were hosted on AWS. After that, these testings were performed again on the hosted and deployed web application (at <https://heartassist.net/>), and the results in this report and in the Appendix are from this final round of testing and validating the hosted web application as it is the system that the end-user will be utilising and interacting with.

Unit Testing

Unit testing is performed to ensure that every unit of the system, this includes components of a web page, will function and are rendered or displayed correctly as expected according to requirements. It is a specific form of testing where individual components of the system are tested separately to verify they are working correctly and fulfilling the initial expectations and design.

For this unit testing that was performed, there are 7 unit test modules that were used with a combined total of 52 unit test cases used to evaluate the web app system components. From performing the unit testing, it was found that all 52 test cases were passed successfully. The following table summarises the results of the unit testing that was performed. For more detailed information regarding the test cases, including how the testing was performed (scenario), the specific steps, prerequisites, test data used, expected result, and status or outcome of testing, the unit test cases are available in Appendix B.

Table IV
Summary of Unit Testing

Unit Test Module ID	Unit Test Module Name	Number of unit test cases in the module	Number of passed unit test cases in the module

UT-TC-001	Home/Heart Disease Risk Calculator Form Page	7	7
UT-TC-002	Registration Pages	6	6
UT-TC-003	Login Page	11	11
UT-TC-004	Patient Account Page	9	9
UT-TC-005	Doctor Account Page	8	8
UT-TC-006	Admin Account Page	5	5
UT-TC-007	Edit Profile/Change Password Page	6	6
Total		52	52

For consistency these unit tests were also performed on different web browsers, namely Google Chrome (chromium), Mozilla Firefox (firefox), and Safari (webkit). In all these major browsers, the results of the testing were the same, where all 52 test cases were passed successfully. Additionally, to ensure responsiveness, and to validate the web application components are functional and rendered correctly on different viewports, the web application was tested on desktop, tablet and mobile viewports (360×640, 1366×768, 1920×1080), this was achieved with the help of Google Chrome DevTools. Similarly, the test cases were passed on these different viewports.

Overall, these unit test results show that all the aforementioned functionalities and features as per the requirements plan were implemented correctly and are functional. Additionally, the design, although does not follow the exact designs as in the original system prototype designs (3.7) created using Adobe XD, the website components are still rendered correctly, clearly, as desired. For instance, the web application theme was changed to follow a light blue colour scheme, owing to personal preference as it was found to be a more calming and professional colour scheme.

Integration Testing

Integration testing validates the communication or connection between the frontend of the web app system to its backend, including the database (MongoDB), by ensuring that the database data is modified and updated correctly and the changes are reflected in the web app system when the system sends a request. Thus, this was verified by using the MongoDB Compass tool to observe whether changes

For this integration testing that was performed, there are 5 integration test modules that were used with a combined total of 15 integration test cases used to evaluate the web app system components and functionalities. From performing the integration testing, it was found that all 15

integration test cases were passed successfully. The following table summarises the results of the integration testing that was performed. For more detailed information regarding the test cases, including how the testing was performed (scenario), the specific steps, prerequisites, test data used, expected result, and status or outcome of testing, the integration test cases are available in Appendix C.

Table V
Summary of Integration Testing

Integration Test Module ID	Integration Test Module Name	Number of integration test cases in the module	Number of passed integration test cases in the module
IT-TC-001	Registration Pages and System	2	2
IT-TC-002	Login Page	3	3
IT-TC-003	Home and Account Page Functionalities	4	4
IT-TC-004	Admin Dashboard Functionalities	4	4
IT-TC-005	Edit Profile/Change Password Page	2	2
Total		15	15

Similar to before, to ensure consistency in functionality these unit tests were also performed on different web browsers, namely Google Chrome (chromium), Mozilla Firefox (firefox), and Safari (webkit). In all these major browsers the results of the testing were the same, where all 15 test cases were passed successfully. Additionally, to ensure responsiveness, to validate the web application components are functional on different viewports, the web application was tested on desktop, tablet and mobile viewports (360×640, 1366×768, 1920×1080), this was achieved with the help of Google Chrome DevTools. Similarly, the test cases were passed on these different viewports. Similarly, the test cases were passed on these different viewports.

These integration tests show that these web app functionalities were implemented correctly to be able to initiate the commands to communicate with the backend and the MongoDB database to make read and write requests quickly and correctly following requirements and according to the designed UML class diagram. Therefore, in the database, the right tables have been created according to the UML class diagram plans (User, Patient, Doctor, Admin, HeartDiseasePrediction), and the functions in the views.py file perform correctly to update these tables correctly according to what the user wants to achieve.

5. Conclusion

This section will summarise or draw an ending to this project, to summarise the findings, the significance of performing this project, its limitations and challenges and future works to improve the project.

Summary of Project

One of the biggest issues facing modern society is heart disease and cardiovascular diseases, taking millions of lives each year, according to the World Health Organisation, WHO [1]. Even in our country, Malaysia, it is especially prevalent. However, at an early stage, it is a disease that can be prevented from worsening through the right medicinal assistance. However, the challenge remains where manually calculating the likelihood of developing heart disease based on risk factors is challenging. Nevertheless, with the aid of machine learning, we can quickly determine whether or not the individual has heart disease. The quick and precise classification of heart disease will help doctors treat patients correctly and potentially save their lives. Prior to the development of machine learning technology, healthcare institutions followed a protocol using rule-based techniques that involved asking a series of questions about the variables that together indicate the presence or absence of HD. This can especially be personnel-intensive, having to sift through a large amount of data available in the ECG reports and other medically-significant data.

Overall, this project was performed to contribute to the existing development of heart disease prediction systems through the incorporation of advanced machine learning techniques such as neural networks as well as testing along with a well-curated dataset and introducing hybrid data mining models to further combine other patient medical information that might be statistically significant and provide improved diagnosis and treatment. Therefore, this system's primary focus was to be designed as a tool to help doctors and medical professionals' decision-making process when dealing with the numerous data available when diagnosing and treating heart disease cases.

The objectives of this project included, designing, implementing and validating different ML algorithms (a full list of implemented algorithms can be seen in section 3.4.5) performance on the UCI heart disease dataset that was chosen. Parameters including dataset features, data preprocessing techniques, split ratio, and model hyperparameter values, were also tuned to find the optimal configurations that produce the best-performing models. The way in which a model can be determined as the best-performing model, was further explored in section 3.4.6, using prediction accuracy scores, error rates, and complexity indications (e.g., training and testing time). This facilitated the model selection process where the chosen ML model to be integrated with the web application was done following the principle of parsimony, where the best-performing model with the least complexity was chosen. This was found to be the StackingClassifier model with the Random Forest model, with the best prediction accuracy scores (86.9%), minimal error rates (0.311 mean square error), and acceptable testing times (0.0522s) for web application deployment. Overall, from viewing the results of the ML implementation process, it can be said that this objective was successfully accomplished, as we have validated our initial assumptions of how these said parameters have an impact on the ML model performance, especially the role of hyperparameter optimisation in finding the optimal parameter values to maximise performance. Additionally, we were able to identify, design, and implement multiple ML models that were performant in heart disease classification (9 models

with more than 80% accuracy), but were able to pinpoint the said StackingClassifier model to be the best performing in terms of accuracy, error rate and complexity.

The next objective was to develop a production-ready web application that would house this chosen ML model, allowing for users to submit their physiological data relating to heart disease classification in a form, that will be inputted into the ML model, that will process these inputs and output a classification (the level of heart disease). Thus, this web application was designed using the Django Python Framework, owing to the ML model being implemented in the Python Programming Language and providing a smooth integration between the frontend and backend components. How the ML algorithm was deployed in this web application format (using pickling) is further explained in the web application implementation section (3.9). Overall, in terms of the web application implementation, all the aforementioned requirements and system functionalities part of the requirements plan, were successfully implemented following the results of the unit and integration testing that were performed. Additionally, the web application can be easily accessible using a web browser, as it has been deployed and hosted on the cloud platform AWS with a domain name, <https://heartassist.net/>, pointing to it.

Limitations and Future Work

Nonetheless, for now, at its first iteration, there exists limitations during the implementation of this project, or out of scope items, several boundaries or functionalities that are not included or covered in this entire project. Therefore, corresponding future works are also proposed along with it. So, these limitations and future work include:

- The algorithm only predicts the presence and level of heart disease, but is not able to identify the exact type of heart disease that a patient is experiencing. For this, image processing and computer vision can be utilised to classify and determine the exact type of heart disease a patient possesses.
- The dataset used for this project is limited (920 records) and not representative of patients in Malaysia at its current iteration as we are utilising the limited resources that are available open-source as well as to avoid an excessively large dataset which will necessitate significant storage, computation and time for training requirements. The low number of attributes selected, 16 attributes, is to reduce the complexity of the designed algorithm and as they were the features that best correlated with the target attribute. Nonetheless, as part of future work, a dataset with more features would be ideal for studying the effect of different features' importance or contribution to heart disease classification and prediction. Also, the dataset we are currently using is a combination of 4 datasets based on records of participants from Budapest, Hungary, Zurich and Basel, Switzerland, and Cleveland, USA, so if we are to deploy it for use in our native country, Malaysia, the location factor in heart disease prediction is not able to be studied. Overall, this may further reduce the system's applicability for commercial or real-world usage. Thus, in future works, we aim to obtain records or datasets from local participants, whether it is through surveys or collaborations with relevant medical and educational institutions.
- Additionally, the final system and overall project are solely intended to be used for educational and research reasons to comprehend the applicability, advantages and disadvantages of ML models, and the attributes that are crucial for this problem domain of heart disease prediction/classification. Therefore, it will undoubtedly need clearances

and approvals from the appropriate authorities before it can be used to make final, conclusive decisions or for real-world commercial scenarios, especially given that this research is focused on the crucial topic of human health. Although the primary goal of this system is to supplement the decision-making process of a doctor, cardiologist, or healthcare professional by analysing vast amounts of data to identify patterns and support the diagnosis. Therefore, the role of a doctor, cardiologist, or healthcare professional is still prominent and essential to reaching a final conclusion regarding a patient's heart health. As a result, this study aims to improve our understanding of the similarities, shortcomings, and potential uses of various ML models.

- For the ANN models implemented, there is a limitation in that not all the topologies or possible structures or the hidden layer sizes have been tested. The hidden layers are capped at 30 neurons in each layer up to 3 layers. So, only when the first layer has reached 30 neurons, will the next layer be filled up with neurons. Therefore hidden layer combinations or topologies, such as (25, 24, 23), have not been tested. This is to simplify the Test 2 (Number of Hidden Layers and Neurons Test) as well as for lower computational time, as testing a different number of neurons for every layer would lead to more complexity in the test sequence. Therefore, to test more topologies, an evolutionary neural network can be designed and developed. It is a hybrid model combining the strengths of neural networks and evolutionary algorithm (genetic algorithm, GA). So, it uses GA to find the optimal values for the weights and topology of the neural network model. For this, the neural network weights and topology are encoded as a chromosome for the GA, the fitness function is defined as the performance of the neural network, minimising the sum of squared errors (SSA). And so, it generates multiple ANNs with different weights and topologies and GA will optimise these chromosomes, by means of crossover and mutation for a number of epochs or training runs that iteratively improve the weights and topologies that maximise the model performance as computed by the fitness function. This evolutionary neural network hybrid model is especially applicable to improve the existing ANN models, as the current models are feedforward in structure, and the direct encoding to chromosomes performed can only be applied on feedforward networks. Overall, the exploration of this hybrid model could pose performance improvements over the existing designed ANN model that could lead to the improved StackingClassifier model in conjunction. Thus, improving the model performances in general.

6. References

- [1] World Health Organisation, "Cardiovascular diseases (CVDs)," who.int.
[https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-\(cvds\)](https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)) (accessed: 20 April, 2022).
- [2] Department of Statistics Malaysia, "Statistics on Causes of Death, Malaysia, 2018,"
dosm.gov.my.
https://www.dosm.gov.my/v1/index.php?r=column/cthemByCat&cat=401&bul_id=aWg2VjJkZHhYcDdEM3JQSGloeTVlZz09&menu_id=L0pheU43NWJwRWVSZklWdzQ4TlhUUT09
(accessed: 20 April, 2022).
- [3] J. Kundu and S. Kundu, "Cardiovascular disease (CVD) and its associated risk factors among older adults in India: Evidence from LASI Wave 1," *Clinical Epidemiology and Global Health*, vol. 13, 2022.
- [4] World Heart Federation, "Cardiovascular Disease," world-heart-federation.org.
<https://world-heart-federation.org/what-is-cvd/> (accessed: 20 April, 2022).
- [5] T. Nishadi, "Predicting Heart Diseases In Logistic Regression Of Machine Learning Algorithms By Python Jupyterlab," *International Journal of Advanced Research and Publications*, vol. 3, no. 8, pp. 69-74, 2019.
- [6] A. Akella and V. Kaushik, "Machine Learning Algorithms for Predicting Coronary Artery Disease: Efforts Toward an Open Source Solution," *Future Sci OA*, vol. 7, no. 6, pp. 31-41, 2021.
- [7] C. Gazeloglu, "Prediction of heart disease by classifying with feature selection and machine learning methods," *Progress in Nutrition*, vol. 22, no. 2, pp. 660-670, 2022.
- [8] A. Hazra, S. K. Mandal, A. Gupta, A. Mukherjee and A. Mukherjee, "Heart Disease Diagnosis and Prediction Using Machine Learning and Data Mining Techniques: A Review," *Advances in Computational Sciences and Technology*, vol. 10, no. 7, pp. 2137-2159, 2017.
- [9] S. Nashif, M. R. Raihan, M. R. Islam and M. H. Imam, "Heart Disease Detection by Using Machine Learning Algorithms and a Real-Time Cardiovascular Health Monitoring System," *World Journal of Engineering and Technology*, vol. 6, no. 4, pp. 854-873, 2018.
- [10] J. Ahamed, A. M. Koli, K. Ahmad, M. A. Jamal and B. B. Gupta, "CDPS-IoT: Cardiovascular Disease Prediction System Based on IoT Using Machine Learning," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 7, no. 4, pp. 78-86, 2021.
- [11] K. Ng, S. R. Steinhubl, C. deFilippi, S. Dey and W. F. Stewart, "Early Detection of Heart Failure Using Electronic Health Records," *Circulation: Cardiovascular Quality and Outcomes*, vol. 9, no. 6, pp. 649-658, 2016.
- [12] G. Malavika, N. Rajathi, V. Vanitha and P. Parameswari, "Heart Disease Prediction Using Machine Learning Algorithms," *Bioscience Biotechnology Research Communications*, vol. 13, no. 11, pp. 24-27, 2020.

- [13] Lucci, S. & Kopec, D., 2016. Artificial Intelligence in the 21st Century. 2nd ed. Mercury Learning and Information
- [14] Negnivitsky, M., 2005. Artificial Intelligence: A Guide to Intelligent Systems 2nd ed. Addison Wesley.
- [15] Han, J., Kamber, M., Pei, J., 2012. Data Mining: Concepts and Techniques 2nd ed. Elsevier.
- [16] TutorialsPoint, "Machine Learning - Logistic Regression," tutorialspoint.com. https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_logistic_regression.htm (accessed: 13 June, 2022).
- [17] JavaTPoint, "Naïve Bayes Classifier Algorithm," javatpoint.com. <https://www.javatpoint.com/machine-learning-naive-bayes-classifier> (accessed: 13 June, 2022).
- [18] L. Breiman, "Random Forests," *Machine Learning*, vol. 4, no. 5, pp. 5-32, 2001.
- [19] IBM Cloud Learn Hub, "Random Forest," ibm.com. <https://www.ibm.com/cloud/learn/random-forest> (accessed: 13 June, 2022).
- [20] R. Gholami and N. Fakhari, "Chapter 27 - Support Vector Machine: Principles, Parameters, and Applications," *Handbook of Neural Computation*, vol. 5, no. 3, pp. 515-535, 2017.
- [21] S. Uddin, I. Haque, H. Lu, M. A. Moni and E. Gide, "Comparative performance analysis of K-nearest neighbour (KNN) algorithm and its different variants for disease prediction," *Scientific Reports*, vol. 10, no. 12, pp. 43-59, 2022.
- [22] TutorialsPoint, "Artificial Intelligence - Neural Networks," tutorialspoint.com. https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm (accessed: 15 June, 2022).
- [23] MathWorks, "What Is a Neural Network?," mathworks.com. <https://www.mathworks.com/discovery/neural-network.html> (accessed: 16 June, 2022).
- [24] N. Donges, "A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks," builtin.com. <https://builtin.com/data-science/recurrent-neural-networks-and-lstm> (accessed: 27 June, 2022).
- [25] S. Grossberg, "Recurrent neural networks," scholarpedia.org. http://www.scholarpedia.org/article/Recurrent_neural_networks (accessed: 18 June, 2022).
- [26] JavaTPoint, "K-Means Clustering Algorithm," javatpoint.com. <https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning> (accessed: 18 June, 2022).
- [27] Mayo Clinic, "Heart disease," mayoclinic.org. <https://www.mayoclinic.org/diseases-conditions/heart-disease/symptoms-causes/syc-20353118> (accessed: 27 June, 2022).

- [28] J. Beckerman, "Heart Disease: Types, Causes, and Symptoms," webmd.com. <https://www.webmd.com/heart-disease/heart-disease-types-causes-symptoms> (accessed: 27 June, 2022).
- [29] HealthLine, "Heart Disease: Risk Factors, Prevention, and More," healthline.com. <https://www.healthline.com/health/heart-disease> (accessed: 27 June, 2022).
- [30] Centers for Disease Control and Prevention, "About Heart Disease," cdc.gov. <https://www.cdc.gov/heartdisease/about.htm> (accessed: 27 June, 2022).
- [31] NHS, "Electrocardiogram (ECG)," nhs.uk. <https://www.nhs.uk/conditions/electrocardiogram/> (accessed: 27 June, 2022).
- [32] Mayo Clinic, "Electrocardiogram (ECG or EKG)," mayoclinic.org. <https://www.mayoclinic.org/tests-procedures/ekg/about/pac-20384983> (accessed: 27 June, 2022).
- [33] MSD Manual, "Electrocardiography," msdmanuals.com. <https://www.msdmanuals.com/home/heart-and-blood-vessel-disorders/diagnosis-of-heart-and-blood-vessel-disorders/electrocardiography> (accessed: 27 June, 2022).
- [34] S. Nashif, M. R. Raihan, M. R. Islam and M. H. Imam, "Heart Disease Detection by Using Machine Learning Algorithms and a Real-Time Cardiovascular Health Monitoring System," *World Journal of Engineering and Technology*, vol. 6, no. 4, pp. 854-873, 2018.
- [35] E. K. Hashi and M. S. U. Zaman, "Developing a Hyperparameter Tuning Based Machine Learning Approach of Heart Disease Prediction," *Journal of Applied Science & Process Engineering*, vol. 7, no. 2, pp. 631-647, 2020.
- [36] N. S. C. Reddy, S. S. Nee, L. Z. Min and C. XinYing, "Classification and Feature Selection Approaches by Machine Learning Techniques: Heart Disease Prediction," *International Journal of Innovative Computing*, vol. 9, no. 1, pp. 39-46, 2018.
- [37] K. Kwakye and E. Dadzie, "Machine Learning-Based Classification Algorithms for the Prediction of Coronary Heart Diseases," *Journal of Computer Science Technical State University*, vol. 12, no. 3, pp. 52-64, 2021.
- [38] A. M. H. Alhussainy and A. D. Jasim, "Heart disease diagnosis based on deep learning network," *Open Journal of Science & Technology*, vol. 4, no. 4, pp. 52-61, 2021.
- [39] T. K. Sajja and H. K. Kalluri, "A Deep Learning Method for Prediction of Cardiovascular Disease Using Convolutional Neural Network," *Revue d'Intelligence Artificielle*, vol. 34, no. 5, pp. 601-606, 2020.
- [40] A. M. Alaa, T. Bolton, E. D. Angelantonio, J. H. F. Rudd and M. v. d. Schaar, "Cardiovascular disease risk prediction using automated machine learning: A prospective study of 423,604 UK Biobank participants," *PLoS ONE*, vol. 14, no. 5, pp. 1-17, 2019.

- [41] UCI Center for Machine Learning and Intelligent Systems, “Heart Disease Data Set,” archive.ics.uci.edu. <http://archive.ics.uci.edu/ml/datasets/Heart+Disease> (accessed: 27 June, 2022).
- [42] MLDataR, “heartdisease: Heart disease dataset,” rdrv.io. <https://rdrv.io/cran/MLDataR/man/heartdisease.html> (accessed: 27 June, 2022).
- [43] National Heart, Lung, and Blood Institute, “Framingham Heart Study-Cohort (FHS-Cohort),” biolincc.nhlbi.nih.gov. <https://biolincc.nhlbi.nih.gov/studies/framcohort/> (accessed: 27 June, 2022).
- [44] UCI Center for Machine Learning and Intelligent Systems, “Z-Alizadeh Sani Data Set,” archive.ics.uci.edu. <https://archive.ics.uci.edu/ml/datasets/Z-Alizadeh+Sani> (accessed: 3 July, 2022).
- [45] UpToDate, “Calculator: Cardiovascular risk assessment,” uptodate.com. <https://www.uptodate.com/contents/calculator-cardiovascular-risk-assessment-10-year-men-patient-education> (accessed: 3 July, 2022).
- [46] Mayo Clinic Health System, “Heart Disease Risk Calculator,” mayoclinichealthsystem.org. <https://www.mayoclinichealthsystem.org/locations/cannon-falls/services-and-treatments/cardiology/heart-disease-risk-calculator> (accessed: 3 July, 2022).
- [47] Reynolds Risk Score, “Reynolds Risk Score,” reynoldsriskscore.org. <http://www.reynoldsriskscore.org/default.aspx> (accessed: 3 July, 2022).
- [48] Medical College of Wisconsin, “Coronary Heart Disease Risk Calculator,” mcw.edu. <https://www.mcw.edu/calculators/coronary-heart-disease-risk> (accessed: 3 July, 2022).

7. Appendices

Appendix A: Results of ML model validation and evaluation metrics

Split Ratio (Train: Test)	Algorithm	Parameters	Accuracy	Precision	Recall	F1-Score	MSE	RMSE	MAE	Misclassification Rate	MCC	Training Time (s)	Testing Time (s)	Total Time (s)
50:50	SVC	kernel: 'rbf' C: 20 gamma: 1	0.763619	0.763619	0.763619	0.763619	0.54572	0.738729	0.32393	0.236381	0.705883	0.054361	0.045641	0.100002
	KNN	n_neighbors: 3 weights: 'distance'	0.713035	0.713035	0.713035	0.713035	0.893969	0.945499	0.446498	0.286965	0.645292	0.000959	0.736005	0.736964
	NB	var_smoothin g: 0.4328761281 0830584	0.45428	0.45428	0.45428	0.45428	1.903696	1.379745	0.923152	0.54572	0.332964	0.000971	0.000666	0.001637
	DT	max_depth: None criterion: 'gini'	0.61965	0.61965	0.61965	0.61965	1.155642	1.075008	0.601167	0.38035	0.525886	0.006155	0.000217	0.006372
	XGB	n_estimators: 100 max_depth: None subsample: 0.75	0.765564	0.765564	0.765564	0.765564	0.672179	0.819865	0.357004	0.234436	0.708809	3.157911	0.003232	3.161143
	RF	n_estimators: 200	0.765564	0.765564	0.765564	0.765564	0.622568	0.78903	0.342412	0.234436	0.708805	0.250775	0.023087	0.273862

	max_features: 'auto' max_depth: None criterion: 'gini'												
EXT	n_estimators: 200 max_features: log2 max_depth: None criterion: 'gini'	0.76361 9	0.7636 19	0.7636 19	0.7636 19	0.6118 68	0.7822 2	0.3414 4	0.2363 81	0.7073 55	0.1664 11	0.026182	0.192593
KMC	n_clusters: 200 max_iter: 150 algorithm: 'lloyd'	0.00291 8	0.0029 18	0.0029 18	0.0029 18	8997.8 73	94.857 12	77.100 19	0.9970 82	-0.011 19	0.2049 25	0.001062	0.205987
LR	C: 1000.0 solver: 'lbfgs'	0.5	0.5	0.5	0.5	1.3638 13	1.1678 24	0.7412 45	0.5	0.3766 45	0.0148 04	0.000107	0.014911
ANN	activation: 'tanh' solver: 'adam' hidden_layer: (27, 27, 27) max_iteration: 723	0.72665 4	0.7266 54	0.7266 54	0.7266 54	0.7762 65	0.8810 59	0.4143 97	0.2733 46	0.6595 28	1.3251 05	0.001062	1.326167
StackingClassifier with SVC Final Estimator	final_estimator: svc_model	0.72081 7	0.7208 17	0.7208 17	0.7208 17	0.7276 26	0.8530 1	0.4105 06	0.2791 83	0.6534 83	19.619 96	0.166609	19.78657

	StackingClassifier with KNN Final Estimator	final_estimator: knn_model	0.72859	0.728599	0.728599	0.728599	0.691634	0.831646	0.393969	0.271401	0.663155	25.69568	0.13461	25.83029
	StackingClassifier with DT Final Estimator	final_estimator: dt_model	0.719844	0.719844	0.719844	0.719844	0.685798	0.828129	0.395914	0.280156	0.650326	22.18681	0.110635	22.29744
	StackingClassifier with XGB Final Estimator	final_estimator: xgb_model	0.792802	0.792802	0.792802	0.792802	0.501946	0.708481	0.293774	0.207198	0.742278	23.82319	0.122879	23.94607
	StackingClassifier with RF Final Estimator	final_estimator: rf_model	0.798638	0.798638	0.798638	0.798638	0.460117	0.678319	0.277237	0.201362	0.749015	22.41306	0.130151	22.54321
	StackingClassifier with EXT Final Estimator	final_estimator: ext_model	0.784047	0.784047	0.784047	0.784047	0.488327	0.698804	0.297665	0.215953	0.73088	22.36948	0.131988	22.50147
	StackingClassifier with MLP Final estimator	final_estimator: mlp_model	0.735409	0.735409	0.735409	0.735409	0.663424	0.814509	0.381323	0.264591	0.67082	22.91071	0.189039	23.09975
60:40	SVC	kernel: 'rbf' C: 20 gamma: 1	0.791971	0.791971	0.791971	0.791971	0.50365	0.709683	0.289538	0.208029	0.741364	0.068263	0.041172	0.109435
	KNN	n_neighbors: 3	0.762774	0.762774	0.762774	0.762774	0.686131	0.82833	0.357664	0.237226	0.7065	0.000156	0.003658	0.003814

	weights: 'distance'												
NB	var_smoothin g: 0.4328761281 0830584	0.46958 6	0.4695 86	0.4695 86	0.4695 86	1.8965 94	1.3771 69	0.9087 59	0.5304 14	0.3555 36	0.0005 89	0.000346	0.000935
DT	max_depth: None criterion: 'gini'	0.64111 9	0.6411 19	0.6411 19	0.6411 19	1.0109 49	1.0054 6	0.5437 96	0.3588 81	0.5513 06	0.0068 77	0.000236	0.007113
XGB	n_estimators: 100 max_depth: None subsample: 0.75	0.79927	0.7992 7	0.7992 7	0.7992 7	0.5036 5	0.7096 83	0.2871 05	0.2007 3	0.7515 17	1.4979 02	0.003198	1.5011
RF	n_estimators: 200 max_features: 'auto' max_depth: None criterion: 'gini'	0.78223 8	0.7822 38	0.7822 38	0.7822 38	0.5717 76	0.7561 59	0.3163 02	0.2177 62	0.7305 07	0.2859 68	0.020161	0.306129
EXT	n_estimators: 200 max_features: log2 max_depth: None criterion: 'gini'	0.79805 4	0.7980 54	0.7980 54	0.7980 54	0.4975 67	0.7053 84	0.2834 55	0.2019 46	0.7490 12	0.1886 12	0.023047	0.211659

KMC	n_clusters: 200 max_iter: 150 algorithm: 'lloyd'	0.00365	0.00365	0.00365	0.00365	10067.01	100.3345	82.18735	0.99635	-0.00719	0.229458	0.001041	0.230499
LR	C: 1000.0 solver: 'lbfgs'	0.534063	0.534063	0.534063	0.534063	1.339416	1.157331	0.706813	0.465937	0.420476	0.022563	0.000103	0.022666
ANN	activation: 'tanh' solver: 'adam' hidden_layer: (27, 27, 27) max_iteration: 723	0.754258	0.754258	0.754258	0.754258	0.548662	0.740717	0.33455	0.245742	0.693839	1.574862	0.00145	1.576312
StackingClassifier with SVC Final Estimator	final_estimator: svc_model	0.783455	0.783455	0.783455	0.783455	0.540146	0.734946	0.306569	0.216545	0.729755	24.97785	0.160725	25.13858
StackingClassifier with KNN Final Estimator	final_estimator: knn_model	0.79562	0.79562	0.79562	0.79562	0.474453	0.688805	0.279805	0.20438	0.745159	24.58221	0.117437	24.69965
StackingClassifier with DT Final Estimator	final_estimator: dt_model	0.759124	0.759124	0.759124	0.759124	0.536496	0.732459	0.324818	0.240876	0.698833	24.92413	0.115252	25.03938
StackingClassifier with XGB Final Estimator	final_estimator: xgb_model	0.824818	0.824818	0.824818	0.824818	0.43309	0.658096	0.248175	0.175182	0.781174	31.12186	0.109678	31.23154

	StackingClassifier with RF Final Estimator	final_estimator: rf_model	0.836983	0.836983	0.836983	0.836983	0.357664	0.59805	0.218978	0.163017	0.796406	26.60964	0.163843	26.77349
	StackingClassifier with EXT Final Estimator	final_estimator: ext_model	0.833333	0.833333	0.833333	0.833333	0.360097	0.600081	0.221411	0.166667	0.792983	24.4792	0.126261	24.60546
	StackingClassifier with MLP Final estimator	final_estimator: mlp_model	0.788321	0.788321	0.788321	0.788321	0.558394	0.747258	0.307786	0.211679	0.735501	30.26761	0.11429	30.3819
70:30	SVC	kernel: 'rbf' C: 20 gamma: 1	0.794165	0.794165	0.794165	0.794165	0.523501	0.723534	0.296596	0.205835	0.743796	0.087871	0.034395	0.122266
	KNN	n_neighbors: 3 weights: 'distance'	0.743922	0.743922	0.743922	0.743922	0.784441	0.885687	0.405186	0.256078	0.681607	0.000318	0.012193	0.012511
	NB	var_smoothin g: 0.4328761281 0830584	0.474878	0.474878	0.474878	0.474878	1.71637	1.310103	0.854133	0.525122	0.353951	0.000602	0.000316	0.000918
	DT	max_depth: None criterion: 'gini'	0.669368	0.669368	0.669368	0.669368	0.896272	0.946717	0.487844	0.330632	0.586984	0.008479	0.000199	0.008678
	XGB	n_estimators: 100	0.790924	0.790924	0.790924	0.790924	0.573744	0.757459	0.314425	0.209076	0.74014	2.996262	0.003676	2.999938

	max_depth: None subsample: 0.75												
RF	n_estimators: 200 max_features: 'auto' max_depth: None criterion: 'gini'	0.78282	0.7828 2	0.7828 2	0.7828 2	0.6061 59	0.7785 62	0.3273 91	0.2171 8	0.7312 34	0.3322 96	0.017522	0.349818
EXT	n_estimators: 200 max_features: log2 max_depth: None criterion: 'gini'	0.80389	0.8038 9	0.8038 9	0.8038 9	0.5170 18	0.7190 4	0.2901 13	0.1961 1	0.7557 1	0.2134 33	0.0206	0.234033
KMC	n_clusters: 200 max_iter: 150 algorithm: 'lloyd'	0.03241 5	0.0324 15	0.0324 15	0.0324 15	8076.4 78	89.869 23	71.440 84	0.9675 85	0.0268 17	0.2670 47	0.000514	0.267561
LR	C: 1000.0 solver: 'lbfgs'	0.57698 5	0.5769 85	0.5769 85	0.5769 85	1.2171 8	1.1032 59	0.6434 36	0.4230 15	0.4731 08	0.0218 33	9.01E-05	0.021923
ANN	activation: 'tanh' solver: 'adam' hidden_layer: (27, 27, 27)	0.76175	0.7617 5	0.7617 5	0.7617 5	0.7098 87	0.8425 48	0.3695 3	0.2382 5	0.7024 37	1.8077 81	0.041328	1.849109

		max_iteration: 723											
StackingClassifier with SVC Final Estimator	final_estimator: svc_model	0.764992	0.76492	0.76492	0.76492	0.6094	0.780641	0.34684	0.23508	0.707402	25.07398	0.123224	25.1972
StackingClassifier with KNN Final Estimator	final_estimator: knn_model	0.797407	0.797407	0.797407	0.797407	0.508914	0.713382	0.288493	0.202593	0.74744	27.82919	0.09378	27.92297
StackingClassifier with DT Final Estimator	final_estimator: dt_model	0.732577	0.732577	0.732577	0.732577	0.701783	0.837725	0.387358	0.267423	0.66582	22.19044	0.085077	22.27552
StackingClassifier with XGB Final Estimator	final_estimator: xgb_model	0.808752	0.808752	0.808752	0.808752	0.410049	0.64035	0.257699	0.191248	0.761756	23.26932	0.10682	23.37614
StackingClassifier with RF Final Estimator	final_estimator: rf_model	0.82658	0.82658	0.82658	0.82658	0.400324	0.632712	0.241491	0.17342	0.783659	21.56434	0.104018	21.66835
StackingClassifier with EXT Final Estimator	final_estimator: ext_model	0.833063	0.833063	0.833063	0.833063	0.410049	0.64035	0.241491	0.166937	0.792139	21.5571	0.152033	21.70913
StackingClassifier with mlp_model	final_estimator: mlp_model	0.756888	0.756888	0.756888	0.756888	0.679092	0.824071	0.371151	0.243112	0.697172	23.4914	0.087115	23.57852

	MLP Final estimator													
80:20	SVC	kernel: 'rbf' C: 20 gamma: 1	0.841849	0.841849	0.841849	0.841849	0.40146	0.633609	0.226277	0.158151	0.802512	0.10457	0.026688	0.131258
	KNN	n_neighbors: 3 weights: 'distance'	0.783455	0.783455	0.783455	0.783455	0.605839	0.778357	0.323601	0.216545	0.729999	0.000291	0.008914	0.009205
	NB	var_smoothing: 0.43287612810830584	0.476886	0.476886	0.476886	0.476886	1.766423	1.329069	0.86618	0.523114	0.349082	0.000575	0.00027	0.000845
	DT	max_depth: 8 None criterion: 'gini'	0.683698	0.683698	0.683698	0.683698	0.990268	0.995122	0.498783	0.316302	0.604506	0.013766	0.000143	0.013909
	XGB	n_estimators: 100 max_depth: None subsample: 0.75	0.844282	0.844282	0.844282	0.844282	0.40146	0.633609	0.226277	0.155718	0.806121	1.885593	0.001801	1.887394
	RF	n_estimators: 200 max_features: 'auto' max_depth: None criterion: 'gini'	0.834555	0.834555	0.834555	0.834555	0.532847	0.729964	0.270073	0.16545	0.793823	0.372334	0.013986	0.38632

EXT	n_estimators: 200 max_features: log2 max_depth: None criterion: 'gini'	0.83698 3	0.8369 83	0.8369 83	0.8369 83	0.4768 86	0.6905 69	0.2530 41	0.1630 17	0.7965 23	0.2379 6	0.017439	0.255399
KMC	n_clusters: 200 max_iter: 150 algorithm: 'lloyd'	0	0	0	0	10320. 1	101.58 79	84.257 91	1	-0.008 49	0.7027 87	0.000468	0.703255
LR	C: 1000.0 solver: 'lbfgs'	0.52798 1	0.5279 81	0.5279 81	0.5279 81	1.3479 32	1.1610 05	0.7153 28	0.4720 19	0.4094 29	0.0263 02	0.0000913	0.026393
ANN	activation: 'tanh' solver: 'adam' hidden_layer: (27, 27, 27) max_iteration: 723	0.78588 8	0.7858 88	0.7858 88	0.7858 88	0.5182 48	0.7198 95	0.3090 02	0.2141 12	0.7324 52	2.0560 33	0.001495	2.057528
StackingClassifier with SVC Final Estimator	final_estimator: svc_model	0.79805 4	0.7980 54	0.7980 54	0.7980 54	0.4768 86	0.6905 69	0.2773 72	0.2019 46	0.7476 48	25.075 28	0.100786	25.17607
StackingClassifier with KNN Final Estimator	final_estimator: knn_model	0.82725 1	0.8272 51	0.8272 51	0.8272 51	0.3965 94	0.6297 57	0.2360 1	0.1727 49	0.7838 98	25.003 79	0.069261	25.07305

	StackingClassifier with DT Final Estimator	final_estimator: dt_model	0.793187	0.793187	0.793187	0.793187	0.420925	0.648787	0.270073	0.206813	0.741486	25.22474	0.064495	25.28924
	StackingClassifier with XGB Final Estimator	final_estimator: xgb_model	0.866188	0.866188	0.866188	0.866188	0.282238	0.531261	0.180049	0.13382	0.832577	26.86886	0.070001	26.93886
	StackingClassifier with RF Final Estimator	final_estimator: rf_model	0.868613	0.868613	0.868613	0.868613	0.311436	0.558064	0.184915	0.131387	0.835799	26.60936	0.052212	26.69057
	StackingClassifier with EXT Final Estimator	final_estimator: r'' ext_model	0.863747	0.863747	0.863747	0.863747	0.309002	0.55588	0.187348	0.136253	0.82951	29.84438	0.093992	29.93837
	StackingClassifier with MLP Final estimator	final_estimator: mlp_model	0.817518	0.817518	0.817518	0.817518	0.396594	0.629757	0.245742	0.182482	0.772953	31.47315	0.112023	31.58517
90:10	SVC	kernel: 'rbf' C: 20 gamma: 1	0.839223	0.839223	0.839223	0.839223	0.407767	0.638566	0.213592	0.140777	0.824106	0.130873	0.01549	0.146363
	KNN	n_neighbors: 3 weights: 'distance'	0.76699	0.76699	0.76699	0.76699	0.665049	0.815505	0.354369	0.23301	0.709975	0.000247	0.00805	0.008297
	NB	var_smoothin g:	0.475728	0.475728	0.475728	0.475728	1.757282	1.325625	0.854369	0.524272	0.342348	0.000646	0.000202	0.000848

		0.4328761281 0830584											
DT	max_depth: None criterion: 'gini'	0.71844 7	0.7184 47	0.7184 47	0.7184 47	0.8300 97	0.9110 97	0.4320 39	0.2815 53	0.6477 11	0.0111 12	0.000153	0.011265
XGB	n_estimators: 100 max_depth: None subsample: 0.75	0.84893 2	0.8489 32	0.8489 32	0.8489 32	0.3592 23	0.5993 52	0.1941 75	0.1310 68	0.8356 03	2.9572 49	0.009753	2.967002
RF	n_estimators: 200 max_features: 'auto' max_depth: None criterion: 'gini'	0.82466	0.8246 6	0.8246 6	0.8246 6	0.4271 84	0.6535 94	0.2330 1	0.1553 4	0.8056 99	0.4126 2	0.010519	0.423139
EXT	n_estimators: 200 max_features: log2 max_depth: None criterion: 'gini'	0.82466	0.8246 6	0.8246 6	0.8246 6	0.4660 19	0.6826 56	0.2427 18	0.1553 4	0.8058 74	0.2570 78	0.013125	0.270203
KMC	n_clusters: 200 max_iter: 150 algorithm: 'lloyd'	0.00485 4	0.0048 54	0.0048 54	0.0048 54	8522.3 79	92.316 73	73.737 86	0.9951 46	-0.005 69	0.7407 37	0.00071	0.741447

	LR	C: 1000.0 solver: 'lbfgs'	0.563107	0.563107	0.563107	0.563107	1.140777	1.068071	0.645631	0.436893	0.452035	0.024586	8.3E-05	0.024669
	ANN	activation: 'tanh' solver: 'adam' hidden_layer: (27, 27, 27) max_iteration: 723	0.820388	0.820388	0.820388	0.820388	0.466019	0.682656	0.262136	0.179612	0.775403	2.309323	0.00026	2.309583
	StackingClassifier with SVC Final Estimator	final_estimator: svc_model	0.814951	0.814951	0.814951	0.814951	0.485437	0.696733	0.252427	0.165049	0.793306	27.1125	0.061795	27.17429
	StackingClassifier with KNN Final Estimator	final_estimator: knn_model	0.815534	0.815534	0.815534	0.815534	0.480583	0.693241	0.266909	0.184466	0.768979	27.29363	0.049391	27.34302
	StackingClassifier with DT Final Estimator	final_estimator: dt_model	0.796117	0.796117	0.796117	0.796117	0.529126	0.727411	0.296117	0.203883	0.745081	25.30777	0.043911	25.35168
	StackingClassifier with XGB Final Estimator	final_estimator: xgb_model	0.853786	0.853786	0.853786	0.853786	0.305825	0.553015	0.179612	0.126214	0.841924	27.00791	0.047935	27.05584
	StackingClassifier with RF Final Estimator	final_estimator: rf_model	0.853786	0.853786	0.853786	0.853786	0.320388	0.566029	0.184466	0.126214	0.841923	26.35669	0.050248	26.40694

	StackingClassifier with EXT Final Estimator	final_estimator: ext_model	0.858641	0.858641	0.858641	0.858641	0.286408	0.535171	0.169903	0.121359	0.847996	25.88951	0.051342	25.94085
	StackingClassifier with MLP Final estimator	final_estimator: mlp_model	0.805243	0.805243	0.805243	0.805243	0.359223	0.599352	0.23301	0.174757	0.782063	28.8087	0.045385	28.85409

Appendix B: Unit Test Cases

Module: Home/Heart Disease Risk Calculator Form Page			Test Module ID: UT-TC-001			
Created by: Saurabh Varughese M. Kovoov			Tested by: Saurabh Varughese M. Kovoov			
Description						
To test the home page with the heart disease risk calculator form and the results page after the form validation has passed and inputs have been processed by the ML model.						
Test case no.	Test scenario	Steps	Prerequisite	Test data	Expected result	Status (Pass/Fail)
1	Load home page and render heart disease prediction form.	1. Enter URL https://heartassistent.net/ in the search engine.	N/A	N/A	Home page with heart disease prediction form is loaded and rendered correctly.	Pass
2	Rendering text on page, and their formatting including the welcome text, notes text, and footer content.	1. Enter URL https://heartassistent.net/ in the search engine. 2. View the welcome text, notes text, and footer content, their positioning,	N/A	N/A	Welcome text, footer text should be centered with the viewport while the notes section should be left-aligned. They should follow	Pass

		sizing and clarity. 3. Test at different viewport sizes (360×640, 1366×768, 1920×1080), using Google Chrome DevTools.			consistent sizing as per the Tailwind CSS format. Ensure these sizing and formatting sizes remain consistent for both mobile (360×640), tablet (1366×768) and desktop viewports (1920×1080)	
3	Rendering form components correctly (fields, labels, tooltips)	1. Enter URL https://heartassisst.net/ in the search engine. 2. View the form labels, enter values into fields, hover over tooltips.	N/A	N/A	Form label texts, texts in fields, and tooltips texts should be visible and correctly rendered.	Pass
4	Rendering correct and corresponding error messages for form validation	1. Enter form inputs, preferably incorrect values or outside range to trigger form validation error	N/A	N/A	Correct and corresponding error message should be displayed under field with incorrect input	Pass

		message. 2. Test submitting form leaving out certain fields.			entered. Error message should be rendered correctly and clearly.	
5.	Testing navigation bar buttons (when not logged in), main logo, register button, log in button	1. Test clicking buttons on the navigation bar (i.e., main logo, register button, log in button).	N/A	N/A	Buttons should link to corresponding page, main logo to homepage (https://heartassist.net/), register to register page (https://heartassist.net/register/), and log in to login page (https://heartassist.net/login/)	Pass
6.	Testing form buttons, submit and reset buttons. Testing buttons after form submission, including the print, retry, register and log in buttons.	1. Test clicking buttons on the form (i.e., submit, reset, print, retry, register and log in buttons).	N/A	N/A	Buttons should perform corresponding function submit should submit the form for form validation and processing inputs, reset should clear the form inputs, print should	Pass

					initiate browser function to request print of web page, retry should clear the page and await for inputs, register should point to registration page and log in should point to login page.	
7.	Rendering results of form submission and input processing correctly. Main results are rendered and displayed first correctly, center aligned, with write font sizing and color. Second division should be a collapsable container, when clicked should expand to contain additional information or potential factors that contributed to the final result	1. Enter form inputs, ensure the values are correct and pass the form validation. Preferably form inputs should be following the conditions in the line 119-142 of the views.py file to trigger the potential factors messages to test the corresponding messages being rendered.	N/A	N/A	First results section, should display the level of heart disease based on the form inputs. Results should be center aligned, with the first text bolded, second text bolded and large primary colour text, with subtitle text beneath. Second division should contain collapsable	Pass

		2. View the results section below the form inputs.			container, when clicked should expand and when clicked again should hide the contents. Contents should consist of corresponding values that are triggered by the form inputs following the conditions in the line 119-142 of the views.py file.	
--	--	--	--	--	---	--

Module: Registration Pages				Test Module ID: UT-TC-002		
Created by: Saurabh Varughese M. Koor				Tested by: Saurabh Varughese M. Koor		
Description						
To test the registration page and the individual patient register and doctor register pages and their forms.						
Test case no.	Test scenario	Steps	Prerequisite	Test data	Expected result	Status (Pass/Fail)

8	Load Register page (https://heartassist.net/register/)	1. Enter URL https://heartassist.net/register/ in the search engine. OR 1. Press the Register button in the navigation bar.	N/A	N/A	Register page is loaded, shown and rendered.	Pass
9	Load Patient Register page (https://heartassist.net/patient_register/)	1. Enter URL https://heartassist.net/patient_register/ in the search engine. OR 1. Press the I AM A PATIENT button.	N/A	N/A	Patient Register page is loaded, along with the corresponding registration form shown and rendered.	Pass
10	Load Patient Register page (https://heartassist.net/doctor_register/)	1. Enter URL https://heartassist.net/doctor_register/ in the search engine. OR	N/A	N/A	Doctor Register page is loaded, along with the corresponding registration form shown and rendered.	Pass

		1. Press the I AM A DOCTOR button.				
11	Rendering correct and corresponding error messages for form validation (repeated for both registration forms)	1. Enter form inputs, preferably incorrect values or outside range to trigger form validation error message. For instance incorrect characters in username, first name, last name phone number, and email fields. Also, leaving out the “@” symbol for the email field and not following a typical email address structure. Additionally password and correct password not matching. 2. Test	N/A	Incorrect account credentials for sign up form for corresponding account.	Correct and corresponding error message should be displayed under field with incorrect input entered. Error message should be rendered correctly and clearly.	Pass

		submitting form leaving out certain fields.				
12	Submitting form without ticking terms and conditions checkbox (repeated for both registration forms)	1. Enter form inputs correctly, passing form validation 2. Submit form without ticking terms and conditions checkbox	N/A	Correct account credentials for sign up form for corresponding account.	Correct and corresponding error message should be displayed under field with incorrect input entered. Error message should be rendered correctly and clearly.	Pass
13	Testing submission of registration forms and correct redirection (repeated for both registration forms)	1. Enter form inputs correctly, passing form validation 2. Submit form by clicking CREATE button	N/A	Correct account credentials for sign up form for corresponding account.	After entering correct registration form inputs and passing form validation, user should be authenticated and automatically logged in, redirected to home page.	Pass

Module: Login Page			Test Module ID: UT-TC-003			
Created by: Saurabh Varughese M. Koor			Tested by: Saurabh Varughese M. Koor			
Description						
To test the log in page and the login system for all users of the system (patient, doctor, and admin).						
Test case no.	Test scenario	Steps	Prerequisite	Test data	Expected result	Status (Pass/Fail)
14	Load Login page (https://heartassist.net/login/)	1. Enter URL https://heartassist.net/login/ in the search engine. OR 1. Press the Log In button in the navigation bar.	N/A	Any user account credentials (username and password) that have been previously registered	Login page is loaded, shown and rendered	Pass
15	Rendering correct and corresponding error messages for form validation. For instance, incorrect username and password combination or leaving out field entries to trigger error messages.	1. Enter and submit incorrect form inputs, preferably incorrect values or outside range or leaving fields blank to trigger form validation error message.	N/A	Incorrect account credentials for login form.	Correct and corresponding error message should be displayed under field with incorrect input entered. Error message should be rendered	Pass

					correctly and clearly. For example, error “Invalid username or password” should be displayed	
16	Patient account logging into account	1. Enter correct username and password 2. Click “Log In” button	A registered patient account should exist in the database and be previously registered/ authenticated.	Test patient account username and password	Patient account will be authorised and logged into the system and redirected to the home page.	Pass
17	Doctor account logging into account	1. Enter correct username and password 2. Click “Log In” button	A registered doctor account should exist in the database and be previously registered/ authenticated.	Test doctor account username and password	Doctor account will be authorised and logged into the system and redirected to the home page.	Pass
18	Admin account logging into account	1. Enter correct username and password 2. Click “Log In” button	A registered admin account should exist in the database and be previously registered/ authenticated.	Test admin account username and password	Admin account will be authorised and logged into the system and redirected to the home page.	Pass

19	Account profile image and name appearing on navigation bar after logging in.	1. Log into account 2. View the account name and profile image section of the navigation bar	A registered patient account should exist in the database and be previously registered/ authenticated.	Test patient/ doctor/ admin account username and password	User's profile picture and username will appear at the navigation bar.	Pass
20	Doctor and patient account try to access admin page	1. Visit https://heartassis.t.net/admin/ url after logging into patient or doctor account.	Logged into patient or doctor account	N/A	Error message will be shown on admin page that current authenticated user is not authorised to access the admin dashboard and they'll have to log in to a different admin account and redirected to admin login page.	Pass
21	Admin account try to access admin page	1. Visit https://heartassis.t.net/admin/ url after logging into admin account.	Logged into admin account	N/A	Admin page and dashboard is loaded, shown and rendered	Pass

22	Patient account logging out of account	1. Press the Log Out button in the navigation bar.	N/A	N/A	Patient account will be logged out from the system and redirected to the home page.	Pass
23	Doctor account logging out of account	1. Press the Log Out button in the navigation bar.	N/A	N/A	Doctor account will be logged out from the system and redirected to the home page.	Pass
24	Admin account logging out of account	1. Press the Log Out button in the navigation bar.	N/A	N/A	Admin account will be logged out from the system and redirected to the home page.	Pass

Module: Patient Account Page			Test Module ID: UT-TC-004			
Created by: Saurabh Varughese M. Koor			Tested by: Saurabh Varughese M. Koor			
Description						
To test the account page for a patient account on the platform.						
Test case no.	Test scenario	Steps	Prerequisite	Test data	Expected result	Status (Pass/Fail)

25	Load Patient Account page (https://heartassist.net/account/)	1. Log in to a patient account. 2. Press the account name in the menu bar or enter https://heartassist.net/account/ in the search engine	User is logged into a Patient account.	N/A	Patient account page is loaded, shown and rendered	Pass
26	Rendering text on page, and their formatting including the patient account details (first and last name and account type), profile picture, profile details, contact details, and edit profile and change password buttons.	1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into patient account. 2. View the aforementioned text on page, and their formatting, their positioning, sizing and clarity. 3. Test at different viewport sizes (360×640,	User is logged into a Patient account.	N/A	Text and components of the page should be centered with the viewport. They should follow consistent sizing as per the Tailwind CSS format. Ensure these sizing and formatting sizes remain consistent for both mobile (360×640), tablet (1366×768) and desktop viewports (1920×1080)	Pass

		1366×768, 1920×1080), using Google Chrome DevTools.				
27	Rendering table of heart disease risk trials.	<p>1. Visit the home page and submit the heart disease prediction form correctly and obtaining a result.</p> <p>2. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into patient account.</p> <p>3. Check the heart disease risk trials table for new entry. Check every column of the row to ensure all cells are</p>	User is logged into an account (Patient or Doctor).	Heart disease prediction form inputs.	Heart disease risk trials table is rendered and displayed with new row. Columns of the row has values, so all cells are populated with same, consistent and valid values.	Pass

		populated and values are consistent and valid.				
28	Rendering table of Connect With a Doctor.	1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into patient account. Preferably a newly registered account with no connected doctors. 2. View the Connect with a Doctor table.	User is logged into a Patient account	N/A	Connect With a Doctor table is rendered and displayed with all the doctor accounts that have been registered with the platform. Columns of the row has values, so all cells are populated with same, consistent and valid values corresponding to that particular doctors details.	Pass
29	Connecting with a Doctor function	1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar	User is logged into a Patient account	N/A	The particular doctor account is added as a connected doctor for this current account. Account page is	Pass

		<p>after logging into patient account. Preferably a newly registered account with no connected doctors.</p> <p>2. View the Connect with a Doctor table.</p> <p>3. Press Connect button for any doctor account or row.</p>			<p>reloaded, with the Connect With a Doctor table no longer visible and only the Connected Doctor table is visible.</p>	
30	Rendering table of Connected Doctor.	<p>1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into patient account.</p> <p>2. View the Connected Doctor table.</p>	User is logged into a Patient account. Current user has previously connected with a doctor on the platform.	N/A	<p>Connected Doctor table is rendered and displayed with the doctor account that have been registered with the platform and added as the patient's connected doctor. Columns of the row has values, so all cells are populated with</p>	Pass

					same, consistent and valid values corresponding to that particular doctors details.	
31	Removing a Connected Doctor function	<ol style="list-style-type: none"> 1. Visit the account page (https://heartassisst.net/account/) by clicking username button in navigation bar after logging into patient account. 2. View the Connected Doctor table. 3. Press Remove button for the connected doctor account. 	User is logged into a Patient account. Current user has previously connected with a doctor on the platform.	N/A	The particular doctor account is removed from the current user account as a connected doctor. Account page is reloaded, with the Connected Doctor table no longer visible and only the Connected With a Doctor table is visible.	Pass
32	Clicking the Edit Profile Details button.	<ol style="list-style-type: none"> 1. Visit the account page (https://heartassisst.net/account/) by clicking username button in navigation bar after logging into patient 	User is logged into a Patient account	N/A	Change Profile Details page is loaded, shown and rendered along with form	Pass

		account. 2. Click the Edit Profile Details button.				
33	Clicking the Change Password button.	1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into patient account. 2. Click the Change Password button.	User is logged into a Patient account	N/A	Change Password page is loaded, shown and rendered along with form	Pass

Module: Doctor Account Page			Test Module ID: UT-TC-005			
Created by: Saurabh Varughese M. Koor			Tested by: Saurabh Varughese M. Koor			
Description						
To test the account page for a doctor account on the platform.						
Test case no.	Test scenario	Steps	Prerequisite	Test data	Expected result	Status (Pass/Fail)

34	Load Doctor Account page (https://heartassist.net/account/)	1. Log in to a doctor account. 2. Press the account name in the menu bar or enter https://heartassist.net/account/ in the search engine	User is logged into a Doctor account.	N/A	Doctor account page is loaded, shown and rendered	Pass
35	Rendering text on page, and their formatting including the doctor account details (first and last name and account type), profile picture, profile details, contact details, other details and edit profile and change password buttons.	1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into doctor account. 2. View the aforementioned text on page, and their formatting, their positioning, sizing and clarity. 3. Test at different viewport sizes (360×640,	User is logged into a Doctor account.	N/A	Text and components of the page should be centered with the viewport. They should follow consistent sizing as per the Tailwind CSS format. Ensure these sizing and formatting sizes remain consistent for both mobile (360×640), tablet (1366×768) and desktop viewports (1920×1080)	Pass

		1366×768, 1920×1080), using Google Chrome DevTools.				
36	Rendering table of heart disease risk trials.	<p>1. Visit the home page and submit the heart disease prediction form correctly and obtaining a result.</p> <p>2. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into doctor account.</p> <p>3. Check the heart disease risk trials table for new entry. Check every column of the row to ensure all cells are populated and</p>	User is logged into an account (Doctor).	Heart disease prediction form inputs.	Heart disease risk trials table is rendered and displayed with new row. Columns of the row has values, so all cells are populated with same, consistent and valid values.	Pass

		values are consistent and valid.				
37	Rendering table of Connected Patients.	1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into doctor account. Preferably a previously registered account with connected patients. 2. View the Connected Patients table.	User is logged into an account (Doctor). User has connected patients.	N/A	Connected Doctor table is rendered and displayed with the doctor account that have been registered with the platform and added as the patient's connected doctor. Columns of the row has values, so all cells are populated with same, consistent and valid values corresponding to that particular doctors details.	Pass
38	Removing a Connected Patient function	1. Visit the account page (https://heartassist.net/account/) by clicking username button	User is logged into an account (Doctor). User has connected patients.	N/A	The particular patient account is removed from the current user account as a connected	Pass

		<p>in navigation bar after logging into doctor account.</p> <p>2. View the Connected Patients table.</p> <p>3. Press Remove button for a particular connected patient account.</p>			<p>patient. Account page is reloaded, with the patient removed from the Connected Patients table.</p>	
39	Viewing a Connected Patients Trial Results Function	<p>1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into doctor account.</p> <p>2. View the Connected Patients table.</p> <p>3. Press VIEW TRIAL RESULTS button for a particular connected patient account.</p>	User is logged into an account (Doctor). User has connected patients.	N/A	Account page is reloaded, with the particular patient's list of heart disease risk trials in a table.	Pass

40	Clicking the Edit Profile Details button.	1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into doctor account. 2. Click the Edit Profile Details button.	User is logged into a Doctor account	N/A	Change Profile Details page is loaded, shown and rendered along with form	Pass
41	Clicking the Change Password button.	1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into doctor account. 2. Click the Change Password button.	User is logged into a Doctor account	N/A	Change Password page is loaded, shown and rendered along with form	Pass

Module: Admin Account Page	Test Module ID: UT-TC-006
----------------------------	---------------------------

Created by: Saurabh Varughese M. Kovoor			Tested by: Saurabh Varughese M. Kovoor			
Description						
To test the account page for a admin account on the platform.						
Test case no.	Test scenario	Steps	Prerequisite	Test data	Expected result	Status (Pass/Fail)
42	Load Admin Account page (https://heartassist.net/admin/)	1. Log in to a admin account. 2. Press the account name in the menu bar or enter https://heartassist.net/admin/ in the search engine	User is logged into a Admin account.	N/A	Admin account page and dashboard is loaded, shown and rendered	Pass
43	Edit users' data and remove user accounts	1. Log in to a admin account at the admin dashboard https://heartassist.net/admin/ 2. Use the controls available to edit a particular user accounts information or	User is logged into a Admin account.	N/A	The change to the user account will be reflected after the admin dashboard has reloaded, whether it is with a new piece of information to a particular attribute or if it is a removal/	Pass

		delete the account			deletion of the user account on the platform.	
44	Edit patients' data and remove patient accounts	1. Log in to a admin account at the admin dashboard https://heartassis.t.net/admin/ 2. Use the controls available to edit a particular patient account's information or delete the account	User is logged into a Admin account.	N/A	The change to the Patient account will be reflected after the admin dashboard has reloaded, whether it is with a new piece of information to a particular attribute or if it is a removal/ deletion of the patient account on the platform.	Pass
45	Edit doctor's data and remove doctor accounts	1. Log in to a admin account at the admin dashboard https://heartassis.t.net/admin/ 2. Use the controls available to edit a particular doctor account's information or	User is logged into a Admin account.	N/A	The change to the Doctor account will be reflected after the admin dashboard has reloaded, whether it is with a new piece of information to a particular attribute or if it	Pass

		delete the account			is a removal/deletion of the doctor account on the platform.	
46	Edit heart disease prediction object's data and remove them	1. Log in to a admin account at the admin dashboard https://heartassis.t.net/admin/ 2. Use the controls available to edit a particular heart disease prediction object's information or delete the object	User is logged into a Admin account.	N/A	The change to the heart disease prediction object will be reflected after the admin dashboard has reloaded, whether it is with a new piece of information to a particular attribute or if it is a removal/deletion of the heart disease prediction object on the platform.	Pass

Module: Edit Profile/Change Password Page			Test Module ID: UT-TC-007			
Created by: Saurabh Varughese M. Kovoor			Tested by: Saurabh Varughese M. Kovoor			
Description						
To test the edit profile details page and the change password page and their forms.						
Test	Test scenario	Steps	Prerequisite	Test data	Expected result	Status

case no.						(Pass/Fail)
47	Load Edit Profile page (https://heartassist.net/account/edit/)	1. Enter URL https://heartassist.net/account/edit/ in the search engine. OR 1. Press the Edit Profile Details button in the account page.	User is logged into a Patient or Doctor account.	N/A	Edit Profile page is loaded, shown and rendered along with the form its labels and fields	Pass
48	Load Change Password page (https://heartassist.net/change-password/)	1. Enter URL https://heartassist.net/change-password/ in the search engine. OR 1. Press the Change Password button in the account page.	User is logged into a Patient or Doctor account	N/A	Change Password page is loaded, shown and rendered along with the form its labels and fields	Pass
49	Rendering correct and corresponding error messages for form	1. Load Edit Profile page 2. Enter form	User is logged into a Patient or Doctor account	Incorrect account credentials for	Correct and corresponding error message	Pass

	validation for edit profile page. For instance, username that already exists, incorrect characters in username, first name, last name phone number, and email fields, leaving out the “@” symbol for the email field and not following a typical email address structure, or leaving out field entries to trigger error messages.	inputs, preferably incorrect values or outside range to trigger form validation error message. For instance incorrect characters in username, first name, last name phone number, and email fields. Also, leaving out the “@” symbol for the email field and not following a typical email address structure. 3. Test submitting form leaving out certain fields.		profile details form.	should be displayed under field with incorrect input entered. Error message should be rendered correctly and clearly.	
50	Rendering correct and corresponding error messages for form validation for change password page. For instance, password not matching	1. Load Change Password page 2. Enter form inputs, preferably incorrect values	User is logged into a Patient or Doctor account	Incorrect inputs for change password form.	Correct and corresponding error message should be displayed under field with	Pass

	confirm password field, or leaving out field entries to trigger error messages.	or outside range to trigger form validation error message. For instance password not matching confirm password field. 3. Test submitting form leaving out certain fields.			incorrect input entered. Error message should be rendered correctly and clearly.	
51	Testing submission of edit profile details form and correct redirection	1. Enter form inputs correctly, passing form validation 2. Submit form by clicking Make Changes button	User is logged into a Patient or Doctor account	Correct account credentials for profile details form for corresponding account.	After entering correct edit profile details form inputs and passing form validation, user information should be updated and redirected to account page showing new updated information.	Pass
52	Testing submission of change password form and correct redirection	1. Enter form inputs correctly, passing form validation	User is logged into a Patient or Doctor account	Correct password entered for change	After entering correct change password form inputs and	Pass

		2. Submit form by clicking Change Password button		password form for corresponding account.	passing form validation, user password should be updated and redirected to account page.	
--	--	---	--	--	--	--

Appendix C: Integration Test Cases

Module: Registration Pages and System			Test Module ID: IT-TC-001			
Created by: Saurabh Varughese M. Kovoov			Tested by: Saurabh Varughese M. Kovoov			
Description						
To test the registration page and the individual patient register and doctor register pages and their forms and whether the data entered and users created are reflected in the database.						
Test case no.	Test scenario	Steps	Prerequisite	Test data	Expected result	Status (Pass/Fail)
1	Testing submission of patient registration forms	1. Load Patient Register page 2. Enter form inputs correctly, passing form validation 3. Submit form by clicking CREATE button	N/A	Correct account credentials for sign up form for corresponding account.	After entering correct registration form inputs and passing form validation, user should be authenticated and automatically logged in. New User account and new Patient Account tied to that User created in the MongoDB database. Can be viewed using	Pass

					MongoDB Compass.	
2	Testing submission of doctor registration forms	1. Load Doctor Register page 2. Enter form inputs correctly, passing form validation 3. Submit form by clicking CREATE button	N/A	Correct account credentials for sign up form for corresponding account.	After entering correct registration form inputs and passing form validation, user should be authenticated and automatically logged in. New User account and new Doctor Account tied to that User created in the MongoDB database. Can be viewed using MongoDB Compass.	Pass

Module: Login Page	Test Module ID: IT-TC-002
Created by: Saurabh Varughese M. Koor	Tested by: Saurabh Varughese M. Koor
Description	
To test the log in page and the login system for all users of the system (patient, doctor, and admin) and whether there is correct	

communication with the database to authorise users' login requests.						
Test case no.	Test scenario	Steps	Prerequisite	Test data	Expected result	Status (Pass/Fail)
3	Patient account logging into account	1. Load login page 2. Enter correct username and password 3. Click "Log In" button	A registered patient account should exist in the database and be previously registered/ authenticated.	Test patient account username and password	Patient account will be authorised and logged into the system and redirected to the home page. Showing that connection to database to authorise user is successful.	Pass
4	Doctor account logging into account	1. Load login page 2. Enter correct username and password 3. Click "Log In" button	A registered doctor account should exist in the database and be previously registered/ authenticated.	Test doctor account username and password	Doctor account will be authorised and logged into the system and redirected to the home page. Showing that connection to database to authorise user is successful.	Pass
5	Admin account logging into	1. Load login	A registered	Test admin	Admin account	Pass

	account	page 2. Enter correct username and password 3. Click “Log In” button	admin account should exist in the database and be previously registered/ authenticated.	account username and password	will be authorised and logged into the system and redirected to the home page. Showing that connection to database to authorise user is successful.	
--	---------	--	---	-------------------------------	---	--

Module: Home and Account Page Functionalities			Test Module ID: IT-TC-003			
Created by: Saurabh Varughese M. Koor			Tested by: Saurabh Varughese M. Koor			
Description						
To test the home page with the heart disease risk calculator form and the results page after the form validation has passed and inputs have been processed by the ML model and whether the data updated to the database. Also to test the account page functionalities’ connection with the database, including the Connecting With a Doctor feature.						
Test case no.	Test scenario	Steps	Prerequisite	Test data	Expected result	Status (Pass/Fail)
6	Saving new Heart Disease Risk Trial to profile	1. Visit the home page and submit the heart disease prediction form correctly and	User is logged into an account (Patient or Doctor).	Heart disease prediction form inputs.	Heart disease risk trials table is rendered and displayed with new row. Columns of the	Pass

		<p>obtaining a result.</p> <p>2. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into account.</p> <p>3. Check the heart disease risk trials table for new entry. Check every column of the row to ensure all cells are populated and values are consistent and valid.</p>			<p>row has values, so all cells are populated with same, consistent and valid values. New Heart Disease Prediction Form object tied to that User created in the MongoDB database. Can be viewed using MongoDB Compass.</p>	
7	Connecting with a Doctor function	<p>1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into patient</p>	User is logged into a Patient account	N/A	<p>The particular doctor account is added as a connected doctor for this current account. Account page is reloaded, with the Connect</p>	Pass

		<p>account. Preferably a newly registered account with no connected doctors. 2. View the Connect with a Doctor table. 3. Press Connect button for any doctor account or row.</p>			<p>With a Doctor table no longer visible and only the Connected Doctor table is visible. Patient's connectedDoctor attribute is updated in the database as seen using MongoDB Compass.</p>	
8	Removing a Connected Doctor function	<p>1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into patient account. 2. View the Connected Doctor table. 3. Press Remove button for the connected doctor account.</p>	<p>User is logged into a Patient account. Current user has previously connected with a doctor on the platform.</p>	N/A	<p>The particular doctor account is removed from the current user account as a connected doctor. Account page is reloaded, with the Connected Doctor table no longer visible and only the Connected With a Doctor table is visible. Patient's connectedDoctor attribute is updated in the</p>	Pass

					database as seen using MongoDB Compass.	
9	Removing a Connected Patient function	<ol style="list-style-type: none"> 1. Visit the account page (https://heartassist.net/account/) by clicking username button in navigation bar after logging into doctor account. 2. View the Connected Doctor table. 3. Press Remove button for the connected doctor account 	User is logged into an account (Doctor). User has connected patients.	N/A	The particular patient account is removed from the current user account as a connected patient. Account page is reloaded, with the patient removed from the Connected Patients table. Corresponding Patient's connectedDoctor attribute is updated in the database as seen using MongoDB Compass.	Pass

Module: Admin Dashboard Functionalities	Test Module ID: IT-TC-004
Created by: Saurabh Varughese M. Koor	Tested by: Saurabh Varughese M. Koor
Description	

To test the account page for a admin account on the platform and its functionalities and whether it is able to connect with the database to perform updates.

Test case no.	Test scenario	Steps	Prerequisite	Test data	Expected result	Status (Pass/Fail)
10	Edit users' data and remove user accounts	<ol style="list-style-type: none"> 1. Log in to a admin account at the admin dashboard https://heartassis.t.net/admin/ 2. Use the controls available to edit a particular user accounts information or delete the account 	User is logged into a Admin account.	N/A	The change to the user account will be reflected after the admin dashboard has reloaded, whether it is with a new piece of information to a particular attribute or if it is a removal/ deletion of the user account on the platform. Change of information or deletion of User object applied in database as seen using MongoDB Compass.	Pass
11	Edit patients' data and remove patient accounts	<ol style="list-style-type: none"> 1. Log in to a admin account at the admin 	User is logged into a Admin account.	N/A	The change to the Patient account will be	Pass

		<p>dashboard https://heartassis.t.net/admin/ 2. Use the controls available to edit a particular patient account's information or delete the account</p>			<p>reflected after the admin dashboard has reloaded, whether it is with a new piece of information to a particular attribute or if it is a removal/deletion of the patient account on the platform. Change of information or deletion of Patient object applied in database as seen using MongoDB Compass.</p>	
12	Edit doctor's data and remove doctor accounts	<p>1. Log in to a admin account at the admin dashboard https://heartassis.t.net/admin/ 2. Use the controls available to edit a particular doctor account's</p>	User is logged into a Admin account.	N/A	<p>The change to the Doctor account will be reflected after the admin dashboard has reloaded, whether it is with a new piece of information to a particular</p>	Pass

		information or delete the account			attribute or if it is a removal/ deletion of the doctor account on the platform. Change of information or deletion of Doctor object applied in database as seen using MongoDB Compass.	
13	Edit heart disease prediction object's data and remove them	<p>1. Log in to a admin account at the admin dashboard https://heartassis.t.net/admin/</p> <p>2. Use the controls available to edit a particular heart disease prediction object's information or delete the object</p>	User is logged into a Admin account.	N/A	The change to the heart disease prediction object will be reflected after the admin dashboard has reloaded, whether it is with a new piece of information to a particular attribute or if it is a removal/ deletion of the heart disease prediction object on the platform. Change of information or	Pass

					deletion of heart disease prediction object applied in database as seen using MongoDB Compass.	
--	--	--	--	--	--	--

Module: Edit Profile/Change Password Page			Test Module ID: IT-TC-005			
Created by: Saurabh Varughese M. Koor			Tested by: Saurabh Varughese M. Koor			
Description						
To test the edit profile details page and the change password page and their forms and whether the changes are implemented to the database.						
Test case no.	Test scenario	Steps	Prerequisite	Test data	Expected result	Status (Pass/Fail)
14	Testing submission of edit profile details form and correct redirection	1. Enter form inputs correctly, passing form validation 2. Submit form by clicking Make Changes button	User is logged into a Patient or Doctor account	Correct account credentials for profile details form for corresponding account.	After entering correct edit profile details form inputs and passing form validation, user information should be updated and redirected to account page	Pass

					showing new updated information. Change of information of User object applied in database as seen using MongoDB Compass.	
15	Testing submission of change password form and correct redirection	User is logged into a Patient or Doctor account	User is logged into a Patient or Doctor account	Correct password entered for change password form for corresponding account.	After entering correct change password form inputs and passing form validation, user password should be updated and redirected to account page. Change of information of User object applied in database as seen using MongoDB Compass.	Pass