CardioRisk: Unraveling Patterns for Early Diagnosis Using Machine Learning

Vani Kancherlapalli, Akansha Malviya, Poojitha Venkat Ram, Shruti Badrinarayanan

Department of Applied Data Science, San Jose State University

DATA 270: Data Analytics Process

Dr. Linsey Pang

November 27, 2023

Abstract

In a fast-paced modern world, health often takes a back seat to daily habits, making early detection and intervention imperative. Cardiovascular diseases (CVD), which claim the lives of approximately 17.9 million people each year, require proactive measures. In the initial phase, we built comprehensive models using methods such as Gradient Boosting (XGBoost), Decision Trees, Support Vector Machines (SVM), and Random Forest powered by a dataset from the University of California, Irvine (UCI) Risk Assessment Repository. We meticulously identified CVD risk using features to assess cardiovascular disease risk, prioritize early intervention, and promote lifestyle changes. The project followed a systematic approach with CRISP-DM phases, ensuring thorough understanding, preparation, modeling, and evaluation. Rigorous model evaluation, including techniques like cross-validation and metrics such as AUC-ROC, Precision, F1 score ensured the utmost accuracy and reliability. Support Vector Machine, Random Forest, XGBoost, and Decision Tree classification models were implemented. The SVM classifier emerged as the top performer among the baseline models, achieving an accuracy of 93.33%, coupled with a validation accuracy of 97.7%. Additionally, the Random Forest model demonstrated good accuracy, reaching 94%. The experimental results underscore the superior performance of the SVM and Random Forest models compared to the other methods, albeit with an associated increase in execution cost. Thus, they holistically contributed to the project's overarching goal of saving lives, reducing healthcare costs, and improving overall health. By providing early detection capabilities, the effort envisions a world where heart health thrives, bringing comfort and vitality to all.

Keywords: decision tree; random forest; gradient boosting; SVM; heart disease; prediction

1. Introduction

1.1. Project Background and Execute Summary

Project Background

Cardiovascular diseases (CVDs) are a primary cause of mortality globally, taking an estimated 17.9 million lives annually, which represents 32% of all global deaths (World Health Organization, 2021). As the prevalence of cardiovascular diseases continues to rise, especially in developing countries, early detection and strengthening of diagnostic methods are essential to reduce its impact (Yusuf et al., 2001). Often important risk factors such as hypertension, hyperlipidemia, heterogeneous demographics constitute a difficult study entity that can be treated with conventional medical approaches.

Need and Importance

Traditional diagnostic methods for cardiovascular disease have many limitations for early detection and personal risk assessment (Arnett et al., 2019). The use of machine learning methods can improve prediction, enable individualized risk profiling, and strengthen intervention strategies in the initial stages. Integrating ML technologies into the diagnosis and treatment of cardiovascular disease can bridge the gap between early diagnosis and effective treatment, thereby reducing the burden of CVD worldwide.

Targeted Problem

The key objective of the project revolves around addressing two aspects in classifying cardiovascular disease. First, we aim to develop robust predictive models for the early detection of CVD. These models will help singularize people with heart disease from people without heart disease, support timely medical intervention and improve patient outcomes. Secondly, we venture to construct a risk assessment model that categorizes individuals into different risk levels—ranging from minimal to moderate to high risk—based on several cardiovascular risk

factors. This project seeks to improve both the diagnosis and prevention of heart disease, ultimately contributing to improved cardiac health in our population.

Motivations and Goals

Driven by the overall goal of improving cardiovascular health worldwide, this project aims to:

- Develop robust machine learning models for early CVD prediction while also leveraging comprehensive risk profiling.
- Provide a steppingstone in enabling preemptive interventions and personalized healthcare strategies for individuals at risk and reduce CVD-related mortalities by amplifying early detection capabilities.

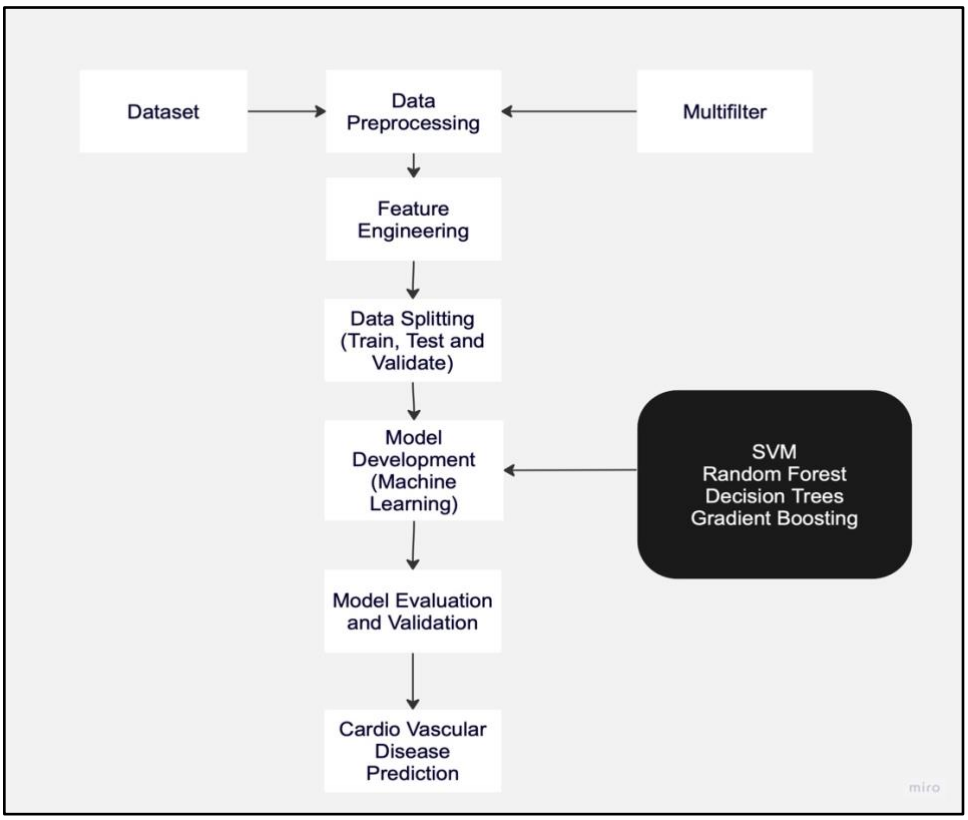
Planned Approaches and Methods

Our study uses the heart disease dataset from the UCI machine learning repository, which comprises of 76 features. Particularly, 14 specific features have been the focus of previous publications and experiments. The Cleveland database of this dataset has so far been of primary interest to machine learning researchers.

We aim to harness the potential of machine learning models such as Gradient Boosting, Support Vector Machines and more to accurately predict cardiovascular disease (CVD) risk using this dataset, which combines data from four different domains and thus covers many cases. This dataset has already been used in several studies, provides a solid basis for our research, and adds both versatility and depth to the input data. Detailed pre-processing steps are carried out, during which data from four different regions are thoroughly cleaned, standardized, and combined, which ensures the consistency and reliability of the data.

Figure 1

Proposed Model Workflow



Feature engineering will be performed to determine that key predictors are aptly represented in the model, aligning with previous research studies (Kurt et al., 2018). The modeling phase uses machine learning algorithms such as Gradient Boosting (specifically XGBoost), decision trees, support vector machines, and a hybrid model of random forest and linear regression to ensure a holistic approach to cardiovascular disease prediction. Leveraging the CRISP-DM model, our workflow will ensure a systematic progression through the phases of understanding, preparing, modeling, and evaluating the predictive model as illustrated in Figure 1. To ensure accuracy and reliability of predictions, each model is trained and evaluated using validation techniques such as cross-validation and analyzed using metrics such as AUC-ROC, mean squared error (MSE) and F1 score.

Guided by the CRISP-DM model, a well-known methodology for conducting data science projects (Wirth & Hipp, 2000), this project will navigate through phases such as:

- Data Understanding and Preparation Phase: Read, clean, and preprocess the UCI Heart Disease dataset.
- Modeling Phase: Build models using algorithms like XGBoost, Decision Tree Classifiers, Support Vector Machines, and a hybrid of Random Forest and Linear Regression using features such as age, resting blood pressure, chest pain type, etc.
- Evaluation Phase: Rigorous model evaluation will be executed using techniques like cross-validation, and metrics like AUC-ROC, Mean Squared Error (MSE), and F1 score, ensuring model reliability.

Expected Project Contributions and Applications

The project expects contributing to:

- Advanced Predictive Diagnostics: Using ML models to accurately predict CVD risk, allowing health professionals to intervene in a timely manner.
- Personalized healthcare: Enabling personalized healthcare and lifestyle reformation approaches tailored to individual risk profiles.
- Global health impact: Potentially lowers global CVD mortality through improved, early risk identification and easier intervention.

In terms of applications, this research could serve as a diagnostic assistant tool for healthcare professionals, power mobile applications for individuals, aiding in self-assessment and awareness regarding cardiovascular health. It could be integrated into healthcare systems to provide robust, data-driven decision-making capabilities concerning CVD management.

1.2. Project Requirements

Functional Requirements

Our project relies on a series of crucial functional requirements. First and foremost is the collection and integration of patient data from our data source. Once collected, the data undergoes meticulous pre-processing and cleaning to address missing values, outliers, and inconsistencies. Feature engineering techniques are employed to extract pertinent features from the data.

The project's core involves the development of machine learning models capable of predicting cardiovascular risk. Various algorithms suitable for classification tasks, such as SVM, Random Forest, Decision Tree, Gradient Boosting, and ensemble methods, are explored. These models are trained and fine-tuned using a dataset split into training, validation, and test sets. Rigorous model validation using appropriate evaluation metrics, including accuracy, precision, recall, F1-score, and AUC-ROC, ensures their effectiveness in early diagnosis.

Upon successful model development, the implementation should be scalable and seamless with the existing healthcare infrastructure. Continuous monitoring and updates are integral to maintaining accuracy. This involves a system for ongoing model performance evaluation and regular improvements.

The final stage involves generating comprehensive reports and visualizing results to provide actionable insights that would enable us to interpret risk scores and predictions for individual patients.

Data Requirements

To ensure the accuracy and effectiveness of cardiovascular risk prediction, specific data requirements must be met. The project necessitates the collection of comprehensive patient data, encompassing demographic information such as age, and sex. Clinical data, including medical

history, medications, family history, and reported symptoms, is vital for a holistic understanding of patient health. Vital signs such as resting blood pressure, resting heart rate, and serum cholesterol levels are also essential components.

Lifestyle and behavioural data provide valuable context. This includes information on smoking habits (including cigarettes per day and years of smoking), physical activity levels, and stress levels. Clinical tests and measurements contribute to risk assessment by including results from blood tests, which encompass cholesterol levels, triglycerides, glucose, and other relevant biomarkers. Resting electrocardiographic results, exercise ECG readings, and related attributes such as exercise-induced angina and ST depression induced by exercise are incorporated if available.

Additionally, genetic data that may influence cardiovascular risk is considered. Environmental and geospatial data, including the geographical location of the patient's region, may be relevant for contextual analysis. Integration with electronic medical records (EMR) facilitates access to historical patient records, including diagnoses, treatments, and outcomes. Tracking outcomes over time is imperative. This encompasses historical data on cardiovascular events, such as heart attacks or strokes, as well as patient outcomes and follow-up visits for long-term health monitoring.

Furthermore, robust data privacy measures are essential to protect sensitive patient information. This comprehensive dataset from UCI serves as the backbone for accurate risk prediction in cardiovascular health, aligning with the project's goals of early diagnosis using machine learning techniques.

1.3. Project Deliverables

Listed below are the project deliverables.

1. Reports: The project utilizes Notion as seen in Figure 2 to track the progress. Everyone is responsible to track the progress of their respective task. Weekly meetings are held as a team to track the overall progress. The rough project timelines are listed below:

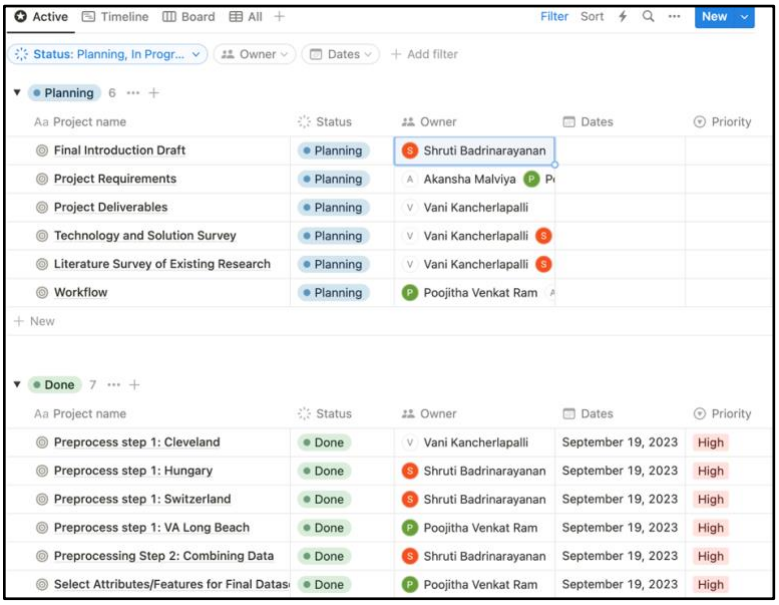
- Data Acquisition (October 21): Have all necessary data, cleaned, and ready for analysis.
- Model Development (October 22): Develop individual robust machine learning models to select and incorporate relevant features.
- Combined Model Integration (November 4): Integrate the machine learning models to generate a comprehensive risk assessment model.
- Conclusion, Results and Visualization (November 18): Document the project's findings through visualizations and analysis.

2. Prototype/Design Document: The project prototype shown below of the initial design flow. The prototype is a high-level diagram of various stages of the project life cycle, and how the components interact with each other. The prototype is working in progress and will be finalized as each stage is completed. Please refer to Figure 1 on the proposed model workflow.

3. Development Applications: The development applications are mainly on Google Colaboratory which is a free cloud-based development environment using python in a Jupyter notebook/lab environment. This environment aids in collaboration during our initial project setup. In addition, each user has their own development environment to work on individual models at a later stage. We are exploring new python libraries in addition to commonly used packages like numpy, matplotlib, etc.

Figure 2

Notion to track the project progress



4. Production Applications: Time permitting to develop a web-based solution, wherein an individual enters a few features like blood pressure, blood sugar, smoking etc with the model predicting the outcome of heart disease or not.

5. Project Documentation: Ongoing process during the CRISP-DM life cycle. This involves comprehensive project documentation and code documentation.

6. Presentations: Ongoing process with project scope. There will be a final project presentation at the end of semester.

1.4. Technology and Solution Survey

Several research studies have made significant contributions for predicting cardiovascular diseases and reducing associated fatalities. Bhatt et al. combined machine learning models like Random Forest, Decision Tree, Multilayer Perceptron, and XGBoost, optimizing classification accuracy through clustering and hyperparameter tuning. The Cleveland and Hungarian databases have been used by many researchers and found to be suitable for developing a mining model,

because of lesser missing values and outliers. The data were cleaned and pre-processed before they were submitted to the proposed algorithm for training and testing.

Shah (2020) focused on feature selection and used multiple algorithms like Decision Trees, Random Forest, K-Nearest Neighbours, and Support Vector Machine. Radhanath Patra & Bonomali Khuntia (2019) compared Weka and Python-based tools, while Palak Khurana et al. (2021) explored various machine learning classifiers and feature selection techniques.

Comparing these approaches, Bhatt et al. & Shah took different paths, with the former emphasizing clustering & hyperparameter tuning & the latter focusing on feature selection & diverse dataset sources. The authors concluded that they achieved higher accuracy by to extracting meaningful features from the available data and averaging the results of a powerful ensemble model. Radhanath Patra & Bonomali Khuntia compared tools, while Palak Khurana et al. evaluated classifiers & feature selection methods. SVM-based approaches, as highlighted by Ahmad & Polat (2023), are popular for predicting cardiovascular diseases. They stress feature engineering, hyperparameter optimization, and performance metrics, showing SVMs' superiority over other algorithms like logistic regression and decision trees.

There is a diverse range of solutions available in existing research literature for addressing each of the aforementioned steps within our problem statement. The project's primary objectives include developing predictive models for early CVD detection and creating comprehensive risk assessment models. Based on our technology survey, our team has decided to use Python packages to perform exploratory data analysis, data cleaning, data pre-processing using various python libraries like (Sklearn, XGboost, DecisionTressClassifier, plot-tree, RandomForestClassifier, matplotlib, SVM).

Based on our technology survey, in our project, machine learning models will be meticulously constructed to enable early diagnosis of cardiovascular risk utilizing the UCI Heart Disease Dataset. The dataset will undergo a thorough preprocessing phase, addressing missing values, outliers, and feature extraction including demographic, clinical, vital signs, lifestyle, and behavioural information. Four distinct algorithms will be employed: Support Vector Machine (SVM) for its capacity to handle complex patterns, Decision Tree for its interpretability, Logistic Regression for its simplicity, and Gradient Boosting for enhanced predictive accuracy along with the Python packages like (Sklearn, XGboost, DecisionTressClassifier, plot-tree, RandomForestClassifier, matplotlib, SVM).

Each model will be trained on a meticulously split dataset, and their performance will be rigorously validated using accuracy, precision, recall, F1-score, and AUC-ROC metrics and Gini Impurity to ensure their effectiveness in early diagnosis. These models will subsequently be deployed for real-time predictions and continuously monitored to maintain their accuracy. An ensemble model will also be constructed, leveraging the collective strengths of individual models, thereby providing a robust framework for early cardiovascular risk assessment.

Figure 3

Evaluation of Model based on different Metrics

Models	Accuracy	Classification error	Precision	F-measure	Sensitivity	Specificity
Naive Bayes	75.8	24.2	90.5	84.5	79.8	60.0
Generalized Linear Model	85.1	14.9	88.8	91.6	94.9	20.0
Logistic Regression	82.9	17.1	89.6	90.2	91.1	25.0
Deep Learning	87.4	12.6	90.7	92.6	95	33.3
Decision Tree	85	15.0	86	91.8	98.8	0.0
Random Forest	86.1	13.9	87.1	92.4	98.8	10.0
Gradient Boosted Trees	78.3	21.7	94.1	86.8	80.7	60.0
Support Vector Machine	86.1	13.9	86.1	92.5	100	0.0
VOTE	87.41	12.59	90.2	84.4	-	-
HRFLM (proposed)	**88.4**	**11.6**	**90.1**	**90**	**92.8**	**82.6**

Figure 3 shows data about different models. This figure is referenced from table 3 of the research paper titled “Effective heart disease prediction using hybrid machine learning techniques” by Senthilkumar Mohan, Chandrasegar Thirumalai & Gautam Srivastava.

Cardiovascular risk prediction relies on a multifaceted classification of features, encompassing diverse aspects of an individual's health profile. Demographic information, such as age forms the foundational context for risk assessment. Clinical data, including medical history, family history, and reported symptoms, provide crucial insights into an individual's health status. Vital signs like resting blood pressure, heart rate, and serum cholesterol levels are vital indicators of cardiovascular health. Lifestyle and behavioural data, such as smoking habits, physical activity levels, and stress, contribute to risk assessment, highlighting the role of modifiable factors. Clinical tests and measurements, like blood lipid levels and biomarkers, offer precise quantitative data for risk evaluation. Resting Electrocardiographic Results, exercise-induced angina, and ST depression readings provide valuable diagnostic insights. Genetic factors influencing cardiovascular risk are considered, and environmental/geospatial data offer contextual information. Electronic Medical Records provide access to historical patient data, enabling a comprehensive overview of diagnoses and treatments. Long-term outcomes, including cardiovascular events and patient outcomes, enable ongoing health monitoring. These diverse features collectively empower healthcare professionals in assessing and mitigating cardiovascular risk effectively.

The applications of this ground-breaking work are diverse and far-reaching. It can serve as a diagnostic assistant tool for healthcare professionals, providing them with invaluable support in their decision-making processes. Additionally, mobile applications can empower individuals to assess and raise awareness about their cardiovascular health. Finally, the integration of project

results into healthcare systems promises to enhance data-driven decision-making for more effective CVD management, ultimately improving the well-being of people worldwide.

1.5. Literature Survey of Existing Research

The research conducted by Bhatt et al. is a significant contribution in the race to formulate a robust mechanism for predicting cardiovascular diseases and to mitigate the fatalities associated with it. The researchers developed various models including Random Forest, Decision Trees, Multilayer Perceptron, and XGBoost, with a focus on optimizing prediction accuracy. GridSearchCV, known for its efficiency in hyperparameter tuning, is employed to refine the models and augment their predictive capabilities. The dataset used has an extensive collection of 70,000 records and is available from Kaggle, served as a base for training and validating the models. They also used binning for age which enhances classification algorithm performance and interpretability. Results showed varying levels of accuracy across different models, with Multilayer Perceptron outshining the others by attaining an accuracy of 87.28% with cross-validation. Moreover, with AUC values ranging from 0.94 to 0.95 across the applied models, the research underscores the potential of these algorithms in effectively predicting CVD and sets a precedent for future research endeavours that aim to integrate machine learning models into healthcare diagnostics and prognostics (pp. 88).

In the research by Shah (2020), the authors pursue an innovative system for predicting heart disease. The framework in this paper primarily worked on a combination of several machine learning algorithms, including Decision Trees, Random Forest, K-Nearest Neighbours, and Support Vector Machine, to build a model that can accurately predict heart disease. This paper uses the UCI Machine Learning Repository's Heart Disease dataset, which has data from four regions, combining them into a holistic data pool, offering a rich, multifaceted dataset.

Feature selection was done, narrowing down from 76 features to 13 principal attributes that are critical for enhancing the model's predictive capabilities. The authors delve into a comparative analysis amongst the enlisted algorithms, scrutinizing their performance through metrics such as accuracy, precision, recall, and F1 score, to understand the best performing algorithm. The paper emphasizes the efficiency of machine learning algorithms, presenting a comprehensive means of predicting heart disease while mitigating the risk of overfitting through feature selection. Ahmad and Rizvi's approach reflects a keen insight into leveraging diverse algorithms to curate a predictive model, offering a robust, validated tool that can potentially streamline the predictive process in a clinical setting, enhancing early detection and intervention in managing heart disease (pp. 1-6).

A research study conducted by Radhanath Patra & Bonomali Khuntia (2019) on the UCI repository's Cleveland Heart dataset having 303 instances and 76 attributes, the authors have used the *Information gain* process to select the best feature. The authors further compared the performance of two machine learning tools: Weka, an open-source data mining and machine learning GUI tool, and a Python-based tool. The reported accuracies were 93.4% for Weka and 87.12% for the Python tool, demonstrating that the Python tool outperformed Weka. This study, published in 2019, has been cited. Using the Python tool, the accuracy results for different models were as follows: Decision Tree (DT) achieved 93.4%, k-Nearest Neighbors (KNN) achieved 76.3%, and Support Vector Machine (SVM) achieved 63.15%. These findings indicated that the Decision Tree algorithm performed the best (pp. 1-4).

A research study/experiment was conducted by Palak Khurana, Shakshi Sharma & Anjali Goyal (2021) on the performance of six machine learning classifiers (Naïve Bayes, Decision Tree, Logistic Regression, Random Forest, Support Vector Machine, k-Nearest Neighbour) and

five feature selection techniques (Chi-Square, Gain Ratio, Information Gain, One-R and RELIEF) on the UCI Machine Learning Repository, Cleveland Heart Disease dataset. Feature selection is an important preprocessing step in machine learning, helps find the optimal subset of features by removing redundant and irrelevant features. The Chi-Square and Information Gain feature selection techniques had 9 features with positive correlation. The One-R feature and RELIEF selection technique had 12 features considered for classification. The ranked list of features obtained using the gain ratio feature had 10 feature selection. The five popular evaluation measures: confusion matrix, precision, recall, F1 Score and accuracy with 4-fold and 10-fold cross validation as test options were used. This was performed using Python Scikit and Weka toolkit for research work. The experiment conducted showed that the machine learning classifiers have a prediction accuracy of 82.81% and the appropriate feature selection technique has further improved the prediction accuracy up to 83.41%. The 0.6% difference on applying the proper feature selection technique can be important in predicting the heart disease patients (pp. 510-515).

Researchers Mythili et al., 2013 have consistently embraced a comprehensive approach to data utilization, drawing upon various patient data sources, including demographic information, clinical measurements (e.g., blood pressure, cholesterol levels), medical history, and lifestyle factors. Central to these endeavours is the employment of feature engineering techniques, emphasizing the extraction of pertinent attributes from the extensive dataset. Notably, feature selection methods have been integral, enhancing model performance by pinpointing the most relevant variables for precise predictions. SVMs shine in handling high-dimensional datasets, with meticulous model training and hyperparameter optimization

processes. Data partitioning into training and testing subsets, accompanied by systematic cross-validation, fortifies model robustness while guarding against overfitting.

The assessment of SVM-based models is underscored by the careful selection of performance metrics, including accuracy, sensitivity, specificity, precision, recall, F1-score, and Receiver Operating Characteristic Area Under the Curve (ROC AUC) (Ahmad & Polat, 2023). Comparative analyses routinely pit SVMs against alternative machine learning algorithms like logistic regression and decision trees, consistently highlighting SVMs' superior predictive accuracy and generalization capabilities. The clinical integration of SVM-based models is of paramount importance, as they seamlessly become part of Clinical Decision Support Systems (CDSS), providing healthcare practitioners with valuable tools for informed CVD management and prevention. Additionally, personalized risk assessment based on individual patient profiles is under exploration, offering tailored recommendations for preventive measures and treatments.

2. Data and Project Management Plan

2.1. Data Management Plan

Data Collection Approaches

Our primary data source is the Heart Disease dataset available from the UCI Machine Learning Repository. This dataset comprises of 76 attributes. Notably, the dataset comprises of information from four distinct locations: Cleveland, Hungary, Switzerland, and the VA Long Beach.

To make a combined dataset, we first changed the original data files (.data) into easier to use CSV files. We added a new feature "location" in the combined dataset to see where each piece of data came from that could help identify patterns in CVD risk based on location. To ensure privacy, any sensitive information about the patients in the dataset was swapped out with fake values by the people who made the UCI Heart Disease dataset.

The Cleveland Heart Disease dataset has been used in numerous research studies using machine learning. Many of these studies have focused on a subset of only fourteen carefully selected features, including age, sex, chest pain type (cp), resting blood pressure (trestbps), serum cholesterol level (chol), fasting blood sugar (fbs), resting electrocardiographic results (restecg), maximum heart rate achieved (thalach), exercise-induced angina (exang), ST depression induced by exercise relative to rest (oldpeak), the slope of the peak exercise ST segment (slope), number of major vessels coloured by fluoroscopy (ca), thalassemia type (thal), and the predicted attribute (num).

The main goal in using this dataset is to build machine learning models for early detection of heart disease. To achieve this, we have primarily focused on distinguishing between

the presence and absence of heart disease which is of clinical importance. By following these steps, we arrive at a comprehensive and privacy-compliant dataset which is suitable for applying machine learning algorithms. The dataset combines information from multiple locations and will be helpful in studying patterns and risk factors associated with CVD.

Data Storage and Management Methods

Efficient data storage and management are important for maintaining a structured and accessible repository of all project-related files and datasets. We have chosen to use a Standard Shared Google Drive as our data management solution due to its collaborative features and ease of use. The merged dataset is at a size of 200 KB (all 76 features). The complete folder structure and naming conventions can be referred in Table 1.

Table 1

Folder Structure and Naming Conventions

Purpose	Folder Name	File Name Convention and Versioning
Main Folder	Heart Disease Datasets	-
UCI Datasets	Raw Data Files	Raw_Data_<location>.DATA
Processed CSVs	Processed CSVs	Processed_Data_<location>.CSV
Merged Data	Merged Data	Heart_Disease_Merged.CSV
Test and Train Data	Split Data	Test_Data_<version_num>.CSV
		Train_Data_<version_num>.CSV
		Validation_Data__<version_num>.CSV

Note. <location> for UCI Datasets and Processed CSVs corresponds to each of the four distinct locations: Cleveland, Hungary, Switzerland, and the VA Long Beach. <version_num> is the version of train, test and validation data starting at “01”.

In our shared drive we have set up a defined folder system where each folder is named thoughtfully to represent its contents. This systematic approach also applies to how we name our

files and by utilizing cloud-based storage we improve data accessibility. This also promotes smooth collaboration among team members. This allows for real time updates and version control. Our data management practices follow industry standards making our project more efficient with an organized repository for all project files. This contributes to the success of the project. We also have weekly backups on our local setups. We regularly update a physical backup, as an extra precautionary measure.

Data Usage Mechanisms

To uphold the integrity and security of our data, we have some data usage protocols in place. Access to the shared drive is only for authorized members, who are required to login using their SJSU (San Jose State University) Email IDs This enables restricting unauthorized access. We have access control policies to control who can view, edit, and delete files/folders. This safeguards sensitive data and ensures data consistency. Regular backups and versioning stand guard against accidental data loss. Our commitment to data privacy and security goes above and beyond by rigorously adhering to relevant data protection laws like GDPR or HIPAA, when they apply. This makes sure that any sensitive data in our project stays private and secure. Our strong rules on how to use data are crucial. They preserve data integrity, promote collaboration, and protect the intellectual property and privacy of our project's data assets.

For the duration of our Master's course and scope of this project, we have decided on a data retention policy for one year. This lets us keep a hold of our past data, which is handy for ongoing and future project tasks, research, and coursework. Our way of storing data involves regular backups and saving old data properly, making sure the data stays reachable and undamaged over time. Beyond serving as a valuable resource for our academic pursuits, this one-

year data storage duration aligns with good data management practices and promotes accountability in data handling and research reproducibility.

2.2. Project Development Methodology

System Development Life Cycle

This project is using CRISP-DM life cycle model. As we need to define a deadline for each stage of the project, we will be following CRISP-DM phases. Each phase of the project is divided into smaller tasks, making it easier to complete. Some of these tasks are repeated, allowing us to go back and forth between the different parts. The responsibility and resource allocation for DMP is detailed in Table 2.

Table 2

Responsibility and Resource Allocation for DMP

DMP Phase	Responsibility	Resource
Data collection, documentation and metadata	Explore different datasets available online Review and finalize the dataset from UCI.	Shruti/Vani
Ethics and legal compliance	Ensure that all the software licenses are acquired and the team complies with the terms and conditions of the agreements.	Poojitha/Akansha
Storage and backup	Version the files appropriately create backup and storage copies of data.	Shruti/Vani
Data Sharing	Data is available publicly on the UCI site, Feature engineering data can be shared and placed on the shared drive.	Poojitha/Akansha
Data Preservation	Decide what data to retain and	Akansha/Shruti

preserve and manage the access,
cost of archival and storage.

Project Development Processes and Activities

Business Understanding. In this phase, the main task is on identify and understand the project objective. We reviewed multiple domain and technology research papers to dive deeper into the project idea and define the project scope. The literature review will help determine the expected output and the resources required. Our project aims to classify if the regular health vital patterns tell us if the patient is more prone to have cardio risk. Classifying patient's vitals are normal or abnormal will help identify individuals at high risk of cardiovascular disease. The limitation of this project is that we are only looking at individuals who have fluctuations in their vitals from the normal range. After finalizing the project scope, we will finalize a project plan by creating a WBS document, Gantt chart, and PERT chart using a project management tools like Notion, Lucidchart and TeamGantt to create a project plan and identify the deliverables for each phase and the resources and tools required at each stage.

Data Understanding. In this phase, we will collect the data required for implementing the machine learning models to identify patient's vital patterns to diagnose cardiovascular disease. We have explored the datasets from different sources and picked the best dataset from UCI repositories. Currently, the collective size of data collected is approximately 191KB and consists of columns – age, sex, smokes etc. Out of these columns, we will be using couple of columns, which would be the result of feature engineering. The combined data has an additional column name "Location" used for testing and training is approximately 200KB. Once we have all these (Cleveland, Hungary, Switzerland, and the VA Long Beach) we will perform feature engineering, we will explore the data to identify which columns are helpful and which should be

discarded. We continue to look for data quality issues and identify solutions to resolve them, use Python packages to perform exploratory data analysis and pre-processing.

Data Preparation. The dataset is collected from UCI repository. The dataset has 76 features, we will perform feature engineering on each country dataset. During data cleaning, we remove the unnecessary columns, standardize all datasets to have and merge the datasets into one. The pre- processed data is then used for feature extraction. Finally, the data is divided into training and testing in the ratio of 80:20. The dataset combines information from multiple locations and will be helpful in studying patterns and risk factors associated with CVD.

Modeling. In this phase, we will build machine learning models using Python libraries like Scikit Learn, matplotlib, decision tree classifier, XGBoost, SVM, Random Forest classifier, NumPy, and Pandas etc. The algorithms we are using for creating our model are Support vector machine which is good for solving complex pattern of data. Decision tree for its interpretability,

Random Forest for overfitting issues of Decision tree. Logistic regression for its simplicity. Gradient boosting for enhanced predictive accuracy. All these algorithms used to design the model which uses python packages such as Sklearn, XGboost, decision tree classifier, plot tree, random forest classifier, matplotlib and SVM.

To classify, categorize, and calculate cardio risk score, we will work on multiple algorithms like SVM, which works well on smaller datasets and has higher accuracy, Random Forest, and Logistics Regression which works well when the number of input parameters is small. Individual ML models are implemented and tested using a test plan created during the beginning of this phase. To test the models, we will use the test dataset generated and verify that the results are synchronous with the expected output defined in the business understanding phase, i.e., classifying if the patient is prone to cardio risk or not, we will assess the results

generated by the different models and interpret them. Once each team member implements the models, we will move to the next step, in which we will collate and compare all the models and their implementations. An ensemble model will be created by combining the individual models and applying bagging, boosting, averaging, and voting techniques. Finally, the best ensemble model will be selected and analysed, and the results will be compared with that of individual models.

Evaluation. In this phase, we will evaluate all the individual models and then ensemble model using evaluation methods like ROC AUC, MSE and Accuracy. The results will be analysed and compared to select the best fit model for deployment.

Deployment. The model we build can be deployed in one of the four ways. As this is an academic project, the team is looking into various options to deploy the model like GCP, Google Colab, Binder, or Github.

2.3. Project Organization Plan

The project follows the CRISP-DM life cycle. There are 6 phases to the CRISP-DM methodology. Each stage is broken down into several steps, which aid in a successful delivery of the project. Some of these steps are iterative, which helps in going back and forth between the steps/stages. The Figure below shows the various stages and process of the data mining life cycle.

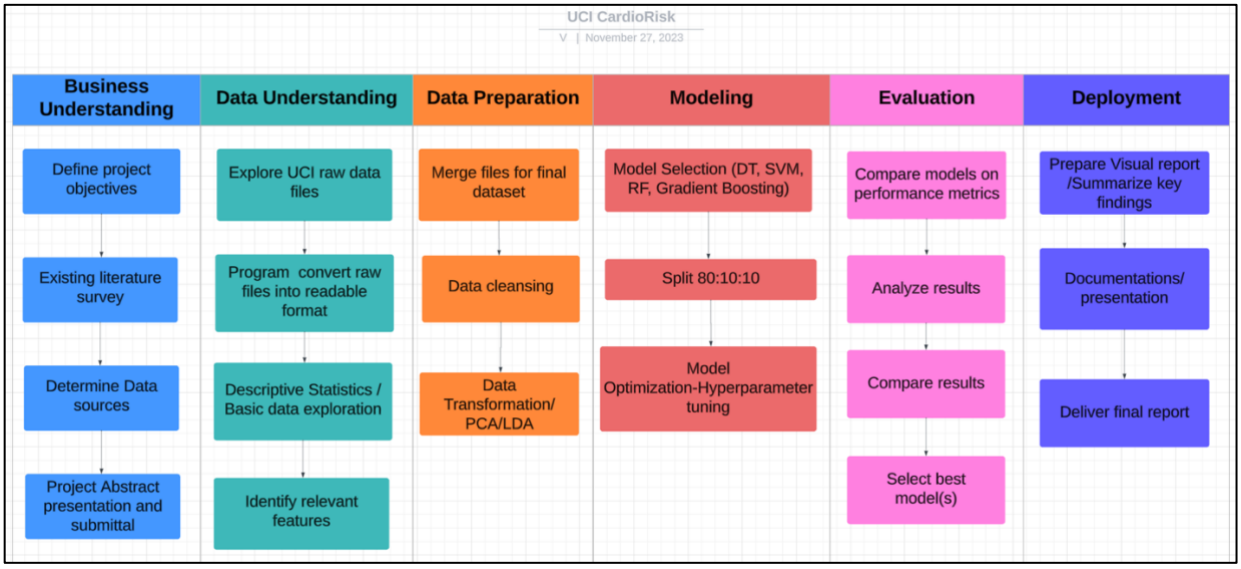
The first stage consists of the project objectives and goals. The team researched literature survey papers of problem statements and summarized them. The team later presented a group project presentation of the project abstract for the first stage.

The second stage is the data understanding phase. The data consists of 76 features involving 4 data files for the Cleveland, Hungary, Switzerland and VA-Long-Beach regions.

This data is stored in raw form on the UCI website. A program was developed to read this raw data and convert it into a readable csv format. The 4 files were then combined into a single csv file. Basic exploratory data analysis tasks were performed on the dataset to better understand the data. Data cleansing and transformation tasks were performed on the combined file. The team will perform data visualizations, looking more closely into some useful insights. This is shown in Figure 4.

Figure 4

Work breakdown structure of the "Cardio Risk" assessment project



The combined file will be used by each member to build and train the dataset with different models. We will be applying hyperparameter techniques to achieve the best performing model. Based on the model being evaluated, the hyperparameter criteria changes.

Lastly, we will evaluate the best model based on the performance and accuracy.

2.4. Project Resource Requirements and Plan

Hardware Requirements

The project is implemented using an appropriate server and a cloud infrastructure that is in tandem with sufficient computing resources such as (CPU, RAM, and GPU). We utilize

cloud-based data storage solutions such as Google Drive which gives us up to 15 GB and is currently sufficient for us in case of any scaling requirements, OneDrive in conjunction with tools like Jupyter Lab for creating Python Notebooks for data pre-processing and for machine learning development, these services are essential for seamless data management, collaboration, and model development which would empower us to drive advancements in our Project efficiently. Table 3 lists all the project's hardware requirements, configuration, and purpose.

Table 3

Hardware Requirements

Resource	Purpose	Configuration	Location	Storage
Google Cloud Storage (GCS)	Data and Source Files Storage	Standard Storage Class	Iowa (us-central1) / South Carolina (us-east1)	3 GB (Currently) 20 GB (Scale-out)
Local Machine	Running local versions of Jupyter Notebook, Python, and other software.	MacOS	Installed and stored directly on the local machine.	8 GB RAM M1 Processor, 1 GB graphics card.
OneDrive Client Apps	Synchronization, Offline Access, and a Backup solution for important files	User Account	Microsoft Azure Cloud	1 TB

Software Requirements

Table 4 provides a comprehensive overview of the essential software tools utilized throughout the project. All the tools utilized are available at no cost, open-source, or provide free access for students. Tools such as Zoom helps facilitate video conferencing and collaboration, while Jupyter Notebook and Colab play a central role in tasks involving data preprocessing,

coding, and analysis. For machine learning endeavours, open-source libraries such as Scikit Learn, NumPy, and XGBoost enhance capabilities. Microsoft 365 tools, including Word and PowerPoint, is instrumental in document creation and presentations. Notion helps in managing timelines, status updates, and promoting collaborative teamwork.

Canva is a platform that helps with the visual presentation design for class presentations, while Google Docs enables cloud-based interactive document creation and editing. GitHub serves as the repository platform for deploying project code at the project's conclusion. Lastly, TeamGantt and Lucidchart, tailored for flowcharts and diagrams, provides essential tools for creating flowcharts, as well as charts such as Gantt and PERT Charts, essential for CRISP-DM data flow, project planning, and visualization.

Table 4

Software Requirements

Software	Purpose	License
Jupyter Notebook	Data preprocessing, coding, and analysis	Free
Python Version 3.9	Model Development	Open Source
Libraries in Python for Machine Learning	Scikit Learn, Pandas, NumPy, XGBoost, etc.	
Zoom	Video conferencing and collaboration	Student License
Microsoft 365	Use various Microsoft tools such Microsoft Word, Excel and PowerPoint Presentation and visual design	Student License
Canva	Collaborative Project Scheduling and Management Tool	Free Version

Google Colab	Allows multiple users to collaborate and integrate in real time.	Free
GitHub	Hosting Service to store code repositories towards the end of the Project.	Free Plan
Lucidchart/TeamGantt	Create flowcharts for CRISP-DM data flow, project management	Free version

2.5. Project Schedule

Gantt Chart

A Gantt chart provides a visual representation of the entire timeline of "CardioRisk" project, encapsulating tasks, sub-tasks, durations, milestones, and resources. The Gantt chart allows stakeholders to perceive how tasks are interlinked to form the larger structure of the project. To streamline our approach, we've aligned our Gantt chart with key stages of a typical machine learning project lifecycle, structured in weekly sprints that target specific deliverables. It's imperative to consider the following:

- Weekends are excluded from estimated efforts.
- The project will acknowledge a break on November 10 and from November 23, 2023, to November 25, 2023.
- Individual subtasks are assigned based on skill and availability, with teamwork being encouraged for complex tasks.
- An upper limit of two days is set for each subtask to ensure timely delivery.

We are following the CRISP-DM methodology, guaranteeing a structured and systematic approach to our project. Each phase of CRISP-DM is reliant on successful completion of the preceding phase and certain tasks within these phases have dependencies on tasks from prior phases.

For example, the Data Preparation phase in CRISP-DM is dependent on the insights learnt and outcomes from the Data Understanding phase. The following sections will investigate the task allocation and effort distribution for each CRISP-DM phase. A comprehensive Gantt chart depicting the entire CRISP-DM process timeline and task breakdown can be accessed in Appendix A.

Business Understanding. Figure 5 illustrates that the business understanding phase kicks off on September 11, 2023, and concludes on October 6, 2023. This phase involves six essential tasks: determining project objectives, conducting literature survey of existing research, determine data sources, surveying current solutions and technology in the market, drafting project proposal, and drafting the introduction chapter.

Tasks such as the literature survey, project proposal drafting, and the creation of the project introduction span a week each, with their individual sub-tasks planned to not exceed a two-day duration. The remaining tasks are planned for a shorter duration. Table 5 presents a detailed overview of the tasks, their start and end dates, and their dependent tasks within the business understanding phase.

Table 5

Simplified Task List for Business Understanding Phase

Name	Task	Start Date	End Date	Dependent On
A	Determine project objective	09/11/2023	09/12/2023	-
B	Literature Survey	09/13/2023	09/17/2023	-
C	Determine Data Sources	09/18/2023	09/22/2023	-
D	Technology Survey	09/23/2023	09/27/2023	-
E	Create Project Proposal	09/28/2023	10/02/2023	A,B,C
F	Create Project Introduction Draft	10/03/2023	10/06/2023	D,E

Data Understanding. The data understanding phase commenced on September 20, 2023, and concluded on October 15, 2023, as seen in Figure 5. This phase encompassed eight tasks: project management planning, selection of data sources, data collection, creation of a data quality strategy, merging datasets, crafting a data management strategy, and establishing a data collection blueprint. Each of these tasks was designated to a weekly sprint.

Project management planning entailed the creation of the Work Breakdown Structure (WBS), Gantt chart, and PERT chart. No specific resources were allocated to the project management planning task, as it needed collective devising among all team members. After identifying and choosing the data sources, we proceeded with our approaches for data collection, organization, and utilization.

We identified our Dataset from the UCI Repository. Subsequently, each team member was assigned to execute tasks related to data collection, exploration, and developing a data quality plan for one of the selected data sources. Following this, the datasets were re consolidated.

The data collection and management plans constituted essential deliverables within the data exploration phase. Table 6 shows an overview of the tasks, their start and end dates, as well as their dependencies within the data understanding phase.

Table 6

Simplified Task List for Data Understanding Phase

Name	Task	Start Date	End Date	Dependent On
A	Project Management Plan	09/20/2023	09/27/2023	-
B	Select Data Source	09/28/2023	09/29/2023	C (Business Understanding)
C	Collect Data	09/02/2023	10/03/2023	B
D	Data Exploration	10/04/2023	10/05/2023	C

E	Create Data Quality Plan	10/06/2023	10/09/2023	D
F	Merge Datasets	10/10/2023	10/11/2023	C
G	Create data management plan	10/12/2023	10/13/2023	B
H	Create Data Collection Plan	10/13/2023	10/15/2023	B

Data Preparation. Figure 5 shows that the data preparation phase begins on October 11, 2023, and ends on October 23, 2023. The phase has six tasks as emphasized in Table 7: standardize data, clean data, merge data, feature selection, split data into train and test, and prepare data engineering report. The datasets belonging to different locations are merged in data understanding phase. The merged dataset is used in the feature selection phase.

Figure 5

Gantt Chart of Business Understanding, Data Understanding and, Data Preparation Phase

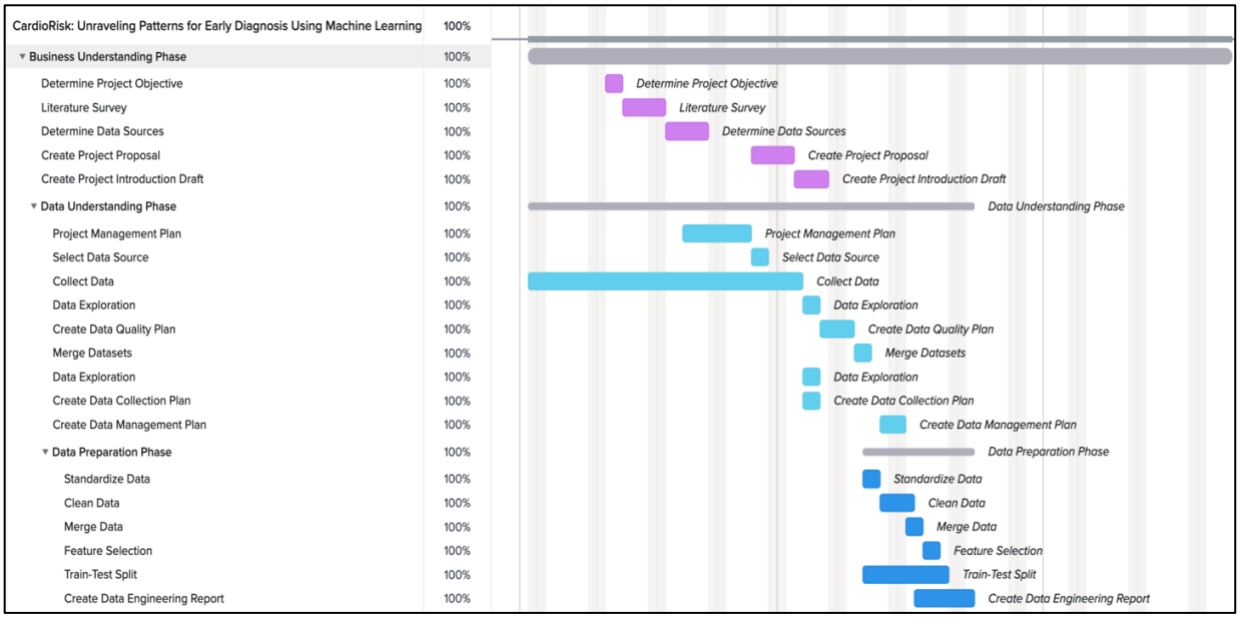


Table 7

Simplified Task List for Data Preparation Phase

Name	Task	Start Date	End Date	Dependent On
A	Standardize Data	10/11/2023	10/12/2023	F (Data

				Understanding)
B	Clean Data	10/13/2023	10/15/2023	A
C	Merge Data	10/16/2023	10/17/2023	B
D	Feature Selection	10/18/2023	10/19/2023	C
E	Train-Test Split	10/11/2023	10/20/2023	D
F	Create Data Engineering Report	10/17/2023	10/23/2023	D

Modeling. The modeling phase, as shown in Figure 6, spans from September 20, 2023, to November 1, 2023. This phase has four tasks such as select machine learning models, create test plan, train individual models, and save each model.

Each team member dedicates a day to select their optimal model for implementation, making sure that all pertinent details about them are gathered. Later, each individual drafts a test plan for the chosen models. The models are trained and saved. Tasks, start and end dates, and dependent tasks for modeling phase can be referred in Table 8.

Table 8

Simplified Task List for Modeling Phase

Name	Task	Start Date	End Date	Dependent On
A	Select ML models	09/20/2023	10/23/2023	-
B	Create test plan	10/24/2023	10/25/2023	-
C	Build, train, and test individual ML models	10/26/2023	10/30/2023	A, B
D	Save Models	10/31/2023	11/01/2023	C

Evaluation. The evaluation phase started on November 2, 2023, and concluded on November 22, 2023 as seen in Figure 6. This phase includes tasks like evaluating individual models using test data, conducting result analysis, performing result comparisons, selecting best-fit model, drafting project abstract, creating project presentation, and creation of project report.

To manage these tasks effectively, they are distributed across five weekly sprints. Tasks like result analysis, result comparison, and model selection are consolidated into a single sprint for improved coordination and effectiveness. Refer to Table 9 for an overview of the tasks, start and end dates, and dependencies.

Figure 6

Gantt Chart of Modeling Phase, Evaluation Phase

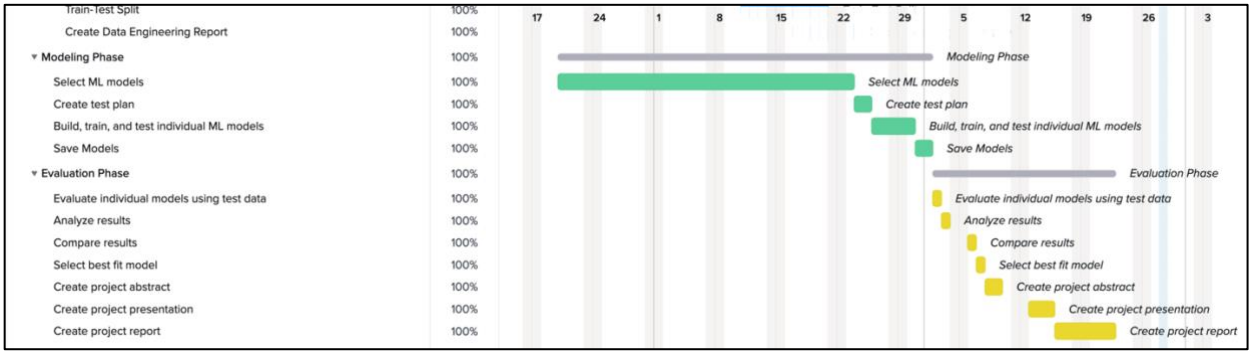


Table 9

Simplified Task List for Evaluation Phase

Name	Task	Start Date	End Date	Dependent On
A	Evaluate individual models using test data	11/02/2023	11/02/2023	C (Modeling)
B	Analyze results	11/03/2023	11/03/2023	D (Modeling)
C	Compare results	11/06/2023	11/06/2023	A
D	Select best fit model	11/07/2023	11/07/2023	B
E	Create project abstract	11/08/2023	11/09/2023	D
F	Create project presentation	11/13/2023	11/15/2023	D
G	Create project report	11/16/2023	11/22/2023	E

PERT Chart

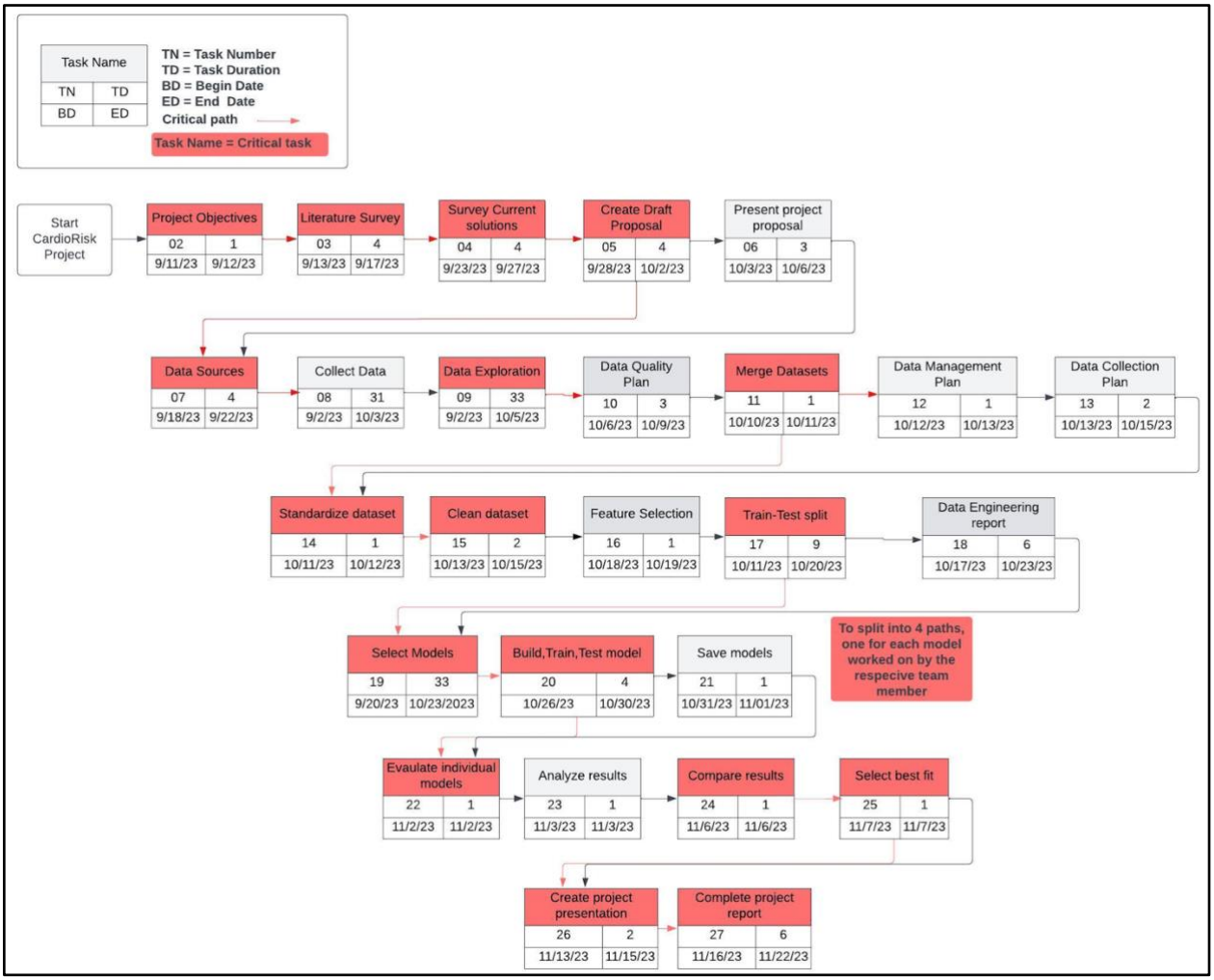
PERT or Program Evaluation and Review Technique chart is another project management tool like Gantt chart. It is used to schedule, organize, and coordinate tasks within a project.

PERT uses a graphical representation showing the project tasks, durations, and the criticality of the task. Each task shows the task name, task number, duration in days, a beginning date and end date for the task. Any critical task is shown with the task name colored in red, along with critical path being followed. The project owners get a comprehensive overview of the project status. This gives the ability to change the project status at any time, to remove any tasks, increase or decrease the duration on a particular task. Any changes are made to the PERT chart and the latest copy is distributed to the project stakeholders, so everyone is on the current schedule. At task number 17, the process is forked into 4 separate steps wherein each team member will work on their selected model for development separately. At task number 22 the results are compared by the 4 different models. Some tasks can run in parallel like task number 13, wherein we can continue with our data management plan and start to normalize/standardize the dataset parallelly.

The graphical representation of the the PERT chart shown in Figure 7 is very helpful to track the current project status by the various team members and stakeholders.

Figure 7

PERT chart for "CardioRisk" Assessment



3. Data Engineering

3.1. Data Process

The data used was obtained from the UCI machine learning repository which is quite well known and has a good reputation. The number of reputed sources for heart disease related data is less in number, and UCI was the best option available for this project. Some datasets looked good in terms of the number of records and information it contained but didn't not have any sources linked to it and hence deemed unfeasible. Since this project relates to patient health, we chose to go with the Heart Disease Dataset which has been utilised and mentioned in numerous studies. It has data from four different locations which are available in different .data files.

Before using the Heart Disease Dataset, we converted the datasets to CSV as it a more suitable format to work with, merged the four datasets, performed data cleaning and pre-processing on the raw data. We identified and removed null values and outliers, and changed the data types and column names to a standard format that makes it easier for the models applied to understand the data.

In the next phase, data transformation is done. There are seventy-six features in our dataset implying that the dimension is very high. Most studies used fourteen particular features from this dataset. We performed feature engineering and selected the most relevant features according to feature importance and also checked if the commonly used features appeared as the ones with higher feature importance. We apply dimensionality reduction methods such as PCA and LDA. A baseline model was selected to conduct the dimensionality reduction methods. These results are evaluated based on accuracy and other relevant metrics.

The data is split into train, test, and validation datasets in the ratio of 80:10:10. The resulting datasets are saved to be utilised in the Model Development stage.

3.2. Data Collection

Table 10 describes different aspects of data collection of the Heart Disease dataset from the UCI Machine Learning Repository.

Table 10

Description of Data Collection: UCI Heart Disease Dataset

Aspect	Description
Why are we collecting data?	The UCI Heart Disease dataset is collected to facilitate the development of predictive models for heart disease. It includes a range of clinical variables critical for identifying heart disease risk factors.
How will the data help?	By applying machine learning techniques to the dataset, we aim to uncover patterns that can predict the presence or absence of heart disease, potentially aiding in early diagnosis and personalized treatment strategies.
What should we do after collecting data?	The data should be preprocessed which includes normalization, handling missing values, and categorical data encoding. The dataset must be divided into a training set for model development, a validation set for validating model performance and a testing set for model evaluation.

Table 11 goes into detail about the metadata (key variables) of eight features and duration of the data collection process along with responsible team members.

Table 11

Key Variables and Data Collection Timeline

Variable title	1 (age)	2 (sex)	3 (cp)	4 (trestbps)	5 (chol)	6 (fbs)	7 (restecg)	8 (num)
Input (X) or output (Y) variable?	X	X	X	X	X	X	X	Y
Unit of measurement	Years	Binary	Chest Pain Type	mm Hg	mg/dl	Binary	Result from ECG	Diagnosis
Data type	Integer	Categorical	Categorical	Integer	Integer	Categorical	Categorical	Categorical
Collection method	**Data is downloaded from the UCI Machine Learning Repository, then processed and converted from .DATA format to .CSV.**							
Historical data exists?	**No. First documentation dates back to 1988 according to files downloaded by the team. Few updates to the database have been made since then.**							
Operational definition exists?	**Operational definition exists for the Heart Disease Dataset to ensure that data collected is consistent. Complete attribute information has been provided which are mapped towards end of data collection process. Each location's data is available in a separate file to ensure separation.**							
Data collectors	Shruti Badrinarayanan				Vani Kancherapalli			
Start date	19-Sept-2023				19-Sept-2023			
Due date	20-Sept-2023				20-Sept-2023			
Duration (in days)	1				1			

Dataset Samples of Raw Data Resources

The collection of data by the authors was done giving preference to more features that contribute to heart disease in the real world. For example, smoking cigarettes and including fatty foods in diet mean that there is more possibility for the presence of cardiovascular disease. The raw data was available in .DATA file formats. A sample is given in Figure 12.

Figure 12

Sample Record from Raw Dataset (Cleveland)

```
1   1 0 63 1 -9 -9 -9
2   -9 1 145 1 233 -9 50 20
3   1 -9 1 2 2 3 81 0
4   0 0 0 0 1 10.5 6 13
5   150 60 190 90 145 85 0 0
6   2.3 3 -9 172 0 -9 -9 -9
7   -9 -9 -9 6 -9 -9 -9 2
8   16 81 0 1 1 1 -9 1
9   -9 1 -9 1 1 1 1 1
10  1 1 -9 -9 name
```

In Figure 13, the actual structure of the dataset can be seen after performing steps to view the data in a Pandas Data Frame. The resulting Data Frame is saved for each location with 76 features each.

Figure 13

Sample Records from Dataset saved in a DataFrame (Hungary)

	0	1	2	3	4	5	6	7	8	9	...	66	67	68	69	70	71	72	73	74	75
0	1254	0	40	1	1	0	0	-9	2	140	...	-9	-9	1	1	1	1	1	-9.	-9.	name
1	1255	0	49	0	1	0	0	-9	3	160	...	-9	-9	1	1	1	1	1	-9.	-9.	name
2	1256	0	37	1	1	0	0	-9	2	130	...	-9	-9	1	1	1	1	1	-9.	-9.	name
3	1257	0	48	0	1	1	1	-9	4	138	...	2	-9	1	1	1	1	1	-9.	-9.	name
4	1258	0	54	1	1	0	1	-9	3	150	...	1	-9	1	1	1	1	1	-9.	-9.	name

5 rows × 76 columns

The four saved CSV files are merged into a comprehensive dataset with an extra Location label to inform where the record originates from. The Location feature can be seen in Figure 14, where there are a total of 77 features instead of the original 76.

Figure 14

Sample Records from Dataset saved as a CSV (Switzerland)

	Location	0	1	2	3	4	5	6	7	8	...	66	67	68	69	70	71	72	73	74	75
0	Switzerland	3001	0	65	1	1	1	1	-9	4	...	1	1	1	1	1	1	1	75.0	-9.0	name
1	Switzerland	3002	0	32	1	0	0	0	-9	1	...	1	1	1	1	1	5	1	63.0	-9.0	name
2	Switzerland	3003	0	61	1	1	1	1	-9	4	...	2	1	1	1	1	1	1	67.0	-9.0	name
3	Switzerland	3004	0	50	1	1	1	1	-9	4	...	1	1	1	1	1	5	4	36.0	-9.0	name
4	Switzerland	3005	0	57	1	1	1	1	-9	4	...	2	1	1	1	1	1	1	60.0	-9.0	name

5 rows × 77 columns

For merging the four individual CSVs for each location, the files are concatenated and saved in a Pandas DataFrame. Once this is done, the column names (‘0’ up to ‘76’) have to be mapped to the corresponding attribute names. It is to be noted that the column labels are not available directly from raw data. The attributes and their descriptions are given in the documentation section of the files from UCI and the mappings are mentioned till the last column, with none missing in the README file. The merged, feature names handled DataFrame is further extracted as a CSV which can be referred to from Figure 15.

Figure 15

Merged Dataset

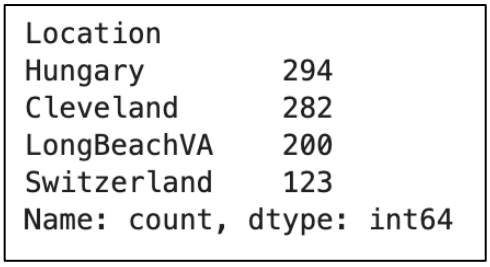
	Location	id	ccf	age	sex	painloc	painexer	relrest	pncaden	cp	...	rcaprox	rcadist	lvx1	lvx2	lvx3	lvx4	lvf	cathef	junk	name
0	Cleveland	1	0	63	1	-9	-9	-9	-9	1	...	1	1	1	1	1	1	1	-9.0	-9.0	name
1	Cleveland	2	0	67	1	-9	-9	-9	-9	4	...	1	1	1	1	1	1	1	-9.0	-9.0	name
2	Cleveland	3	0	67	1	-9	-9	-9	-9	4	...	2	2	1	1	1	7	3	-9.0	-9.0	name
3	Cleveland	4	0	37	1	-9	-9	-9	-9	3	...	1	1	1	1	1	1	1	-9.0	-9.0	name
4	Cleveland	6	0	41	0	-9	-9	-9	-9	2	...	1	1	1	1	1	1	1	-9.0	-9.0	name

5 rows × 77 columns

The total number of instances/records from all four locations put together is 899 as seen from Figure 16.

Figure 16

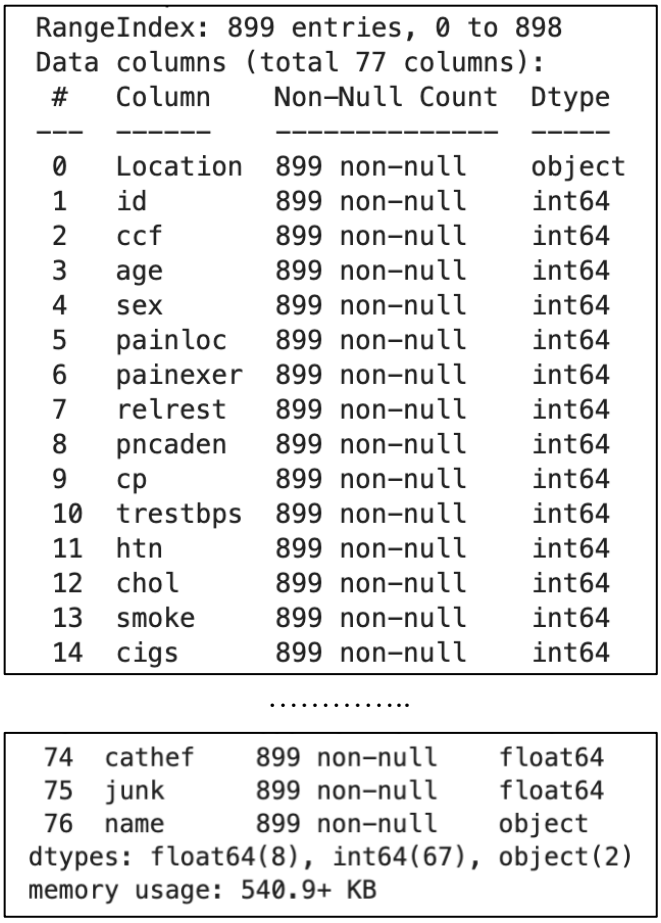
Instance Counts for Each Location



```
Location
Hungary        294
Cleveland      282
LongBeachVA    200
Switzerland    123
Name: count, dtype: int64
```

Figure 17

Summary of Collected Dataset using Pandas Data Frame



```
RangeIndex: 899 entries, 0 to 898
Data columns (total 77 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Location  899 non-null    object
 1   id        899 non-null    int64
 2   ccf       899 non-null    int64
 3   age       899 non-null    int64
 4   sex       899 non-null    int64
 5   painloc   899 non-null    int64
 6   painexer  899 non-null    int64
 7   relrest   899 non-null    int64
 8   pncaden   899 non-null    int64
 9   cp        899 non-null    int64
 10  trestbps  899 non-null    int64
 11  htn       899 non-null    int64
 12  chol      899 non-null    int64
 13  smoke     899 non-null    int64
 14  cigs      899 non-null    int64
```

.....


```
 74  cathef    899 non-null    float64
 75  junk      899 non-null    float64
 76  name      899 non-null    object
dtypes: float64(8), int64(67), object(2)
memory usage: 540.9+ KB
```

Figure 17 presents an overview of the DataFrame, providing valuable insights into its composition. The dataset contains 899 entries with 76 columns. Notably, there are no missing values in any column. The data types include 2 columns with the 'object' type, 8 with 'float' type, and 67 with 'integer' type. The memory usage for the DataFrame is approximately 541 KB. The description for few of the features is listed. The complete description of each feature is documented as part of the python notebook. The ‘Location’ feature summary is shown in Figure 16. The ‘id’ is the unique identifier for each row. The ‘ccf’ is the SSN of the patient which removed and is set to 0. The ‘age’ feature is represented in years captures the age of the patient. The ‘sex’ is represented with female as 0 and male as 1. The ‘painloc’ is the chest pain location having 1 as substernal and 0 as otherwise. The ‘cp’ is chest pain type, with value 1 is typical angina, value 2 is atypical angina, value 3 is non-anginal pain and value 4 is asymptomatic.

3.3. Data Pre-processing

Data pre-processing is an important step in the CRISP-DM data life cycle. This is the step after the data preparation stage and before the modeling stage. There can be multiple iterations between the data pre-processing and modeling stages.

This stage involves cleaning and transforming raw data into suitable format for modeling or performing any other analysis. For example, handling missing values, duplicate values, and encoding are some of the tasks. The UCI Heart Disease dataset has four raw data files with 76 features each. These files are converted into a readable csv format and later merged into a single csv file.

Figure 18 examines the Data Frame for any duplicated entries. No duplicate values are identified in the output.

Figure 18

Check for duplicated data

```
No duplicate records in dataset
```

In Figure 19 the code iterates over the entire DataFrame to show the column name and the number of unique column values. This process helped us understand the cardinality of the DataFrame and decide if any feature required further exploration. The target feature was shown as 5 unique values i.e. 0, 1, 2, 3, 4. We decided to keep the original target feature for performing further analysis. Further we created a new encoded feature with values 0 and 1 only for modeling purposes.

Figure 19

Sample to count the features unique values

```
age 50
sex 2
painloc 3
painexer 3
relrest 3
pncaden 1
cp 4
trestbps 61
htn 3
chol 214
smoke 3
cigs 26
years 43
fbs 3
dm 3
famhist 3
restecg 4
ekgmo 13
ekgday 32
ekgyr 8
dig 3
prop 4
nitr 3
pro 3
diuretic 3
proto 15
thaldur 87
thaltime 65
met 35
thalach 120
```

Figure 20 shows there are no missing values in the DataFrame.

Figure 20

Check for missing values in dataset

```
Empty DataFrame
Columns: [Feature, Missing Percentage]
Index: []
```

In the data preparation stage, we noticed a use of special code of -9 in the DataFrame. Features such as ‘chol’ (cholesterol showed 30 count), ‘trestbps’ (resting blood pressure showed 59), ‘smoke’ (smoke showed 669 count), fbs (fasting blood sugar showed 90 count), dm (diabetics showed 804 count), famhist (family history showed 422 count), exang (exercise induced angina showed 55 count) consistently showed the special code. We performed more analysis on the special code and decided to clean the special code. This is seen below in Figure 21.

Figure 21

Sample data on more analysis on special code (–9) (BEFORE)

	Location	id	ccf	age	sex	painloc	painexer	relrest	pncaden	cp	...	rcaprox	rcadist	lvx1	lvx2	lvx3	lvx4	lvf	cathef	junk	name
0	Cleveland	1	0	63	1	-9	-9	-9	-9	1	...	1	1	1	1	1	1	1	-9.0	-9.0	name
1	Cleveland	2	0	67	1	-9	-9	-9	-9	4	...	1	1	1	1	1	1	1	-9.0	-9.0	name
2	Cleveland	3	0	67	1	-9	-9	-9	-9	4	...	2	2	1	1	1	7	3	-9.0	-9.0	name

We first show the feature name and percentage missing of the special code with the DataFrame as seen in Figure 22. This helped give us a better understanding.

Figure 22

Sample data on more analysis on special code (–9) (BEFORE)

```
Feature: trestbps, Missing Percentage: 6.56%, Category: 1-10%
Feature: htn, Missing Percentage: 3.78%, Category: 1-10%
Feature: chol, Missing Percentage: 3.34%, Category: 1-10%
Feature: ekgmo, Missing Percentage: 5.90%, Category: 1-10%
Feature: ekgday, Missing Percentage: 6.01%, Category: 1-10%
Feature: ekgyr, Missing Percentage: 5.90%, Category: 1-10%
Feature: dig, Missing Percentage: 7.56%, Category: 1-10%
Feature: prop, Missing Percentage: 7.34%, Category: 1-10%
Feature: nitr, Missing Percentage: 7.23%, Category: 1-10%
Feature: pro, Missing Percentage: 7.01%, Category: 1-10%
Feature: diuretic, Missing Percentage: 9.12%, Category: 1-10%
Feature: thaldur, Missing Percentage: 6.23%, Category: 1-10%
Feature: thalach, Missing Percentage: 6.12%, Category: 1-10%
Feature: thalrest, Missing Percentage: 6.23%, Category: 1-10%
Feature: tpeakbps, Missing Percentage: 7.01%, Category: 1-10%
```

In Figure 23, the special code -9 was replaced with NaNs (Not a Number) which helped identify the special code data.

Figure 23

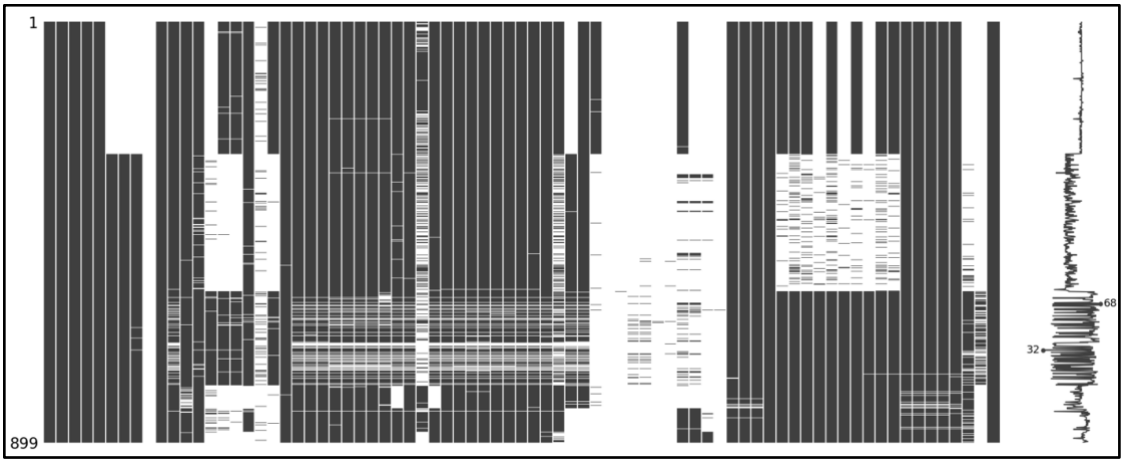
Replacing special code (–9) with Nan (AFTER)

	Location	id	ccf	age	sex	painloc	painexer	relrest	pncaden	cp	...	rcaprox	rcadist	lvx1	lvx2	lvx3	lvx4	lvf	cathef	junk	name
0	Cleveland	1	0	63	1	NaN	NaN	NaN	NaN	1	...	1.0	1.0	1.0	1.0	1.0	1.0	1.0	NaN	NaN	name
1	Cleveland	2	0	67	1	NaN	NaN	NaN	NaN	4	...	1.0	1.0	1.0	1.0	1.0	1.0	1.0	NaN	NaN	name
2	Cleveland	3	0	67	1	NaN	NaN	NaN	NaN	4	...	2.0	2.0	1.0	1.0	1.0	7.0	3.0	NaN	NaN	name

Figure 24 uses the missing no library and provides a matrix visualization which highlights the presence of missing values in a Data Frame. This visualization can help you quickly identify patterns of missing data in your dataset. In this matrix plot, each row corresponds to a row in Data Frame, and each column represents a variable. The white lines in the matrix indicate missing values. The whiter space in a column, the more missing values it has.

Figure 24

missingno matrix to visualize missing data (BEFORE CLEANING)



The missing values are handled by the imputation method. Figure 25 shows sample features with missing values.

Figure 25

Handle missing values by imputation method (BEFORE)

```
     prop  nitr   chol
0     0.0   0.0  233.0
1     1.0   0.0  286.0
2     1.0   0.0  229.0
3     1.0   0.0  250.0
4     0.0   0.0  204.0
..    ...   ...    ...
894   0.0   0.0    0.0
895   0.0   0.0    0.0
896   0.0   0.0    0.0
897   0.0   0.0    0.0
898   0.0   0.0    0.0

[899 rows x 3 columns]
```

The imputation by mode is for the features having a low cardinality as shown in Figure 26. The imputation by mean is for numerical features as shown in figure 27.

Figure 26

Handle missing values by mode method (AFTER)

```
Updated 684 values in column 'prop' with mode value 0.0
Updated 677 values in column 'nitr' with mode value 0.0
Updated 755 values in column 'pro' with mode value 0.0
Updated 807 values in column 'diuretic' with mode value 0.0
Updated 569 values in column 'exang' with mode value 0.0
Updated 877 values in column 'xhypo' with mode value 0.0
Updated 108 values in column 'cmo' with mode value 3.0
```

Figure 27

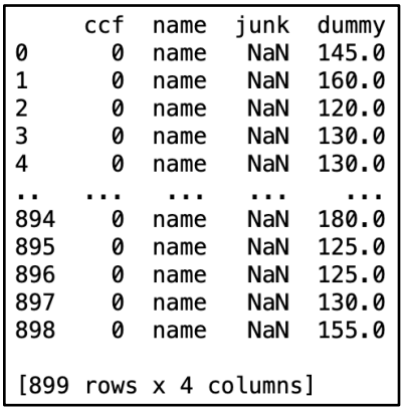
Handle missing values by mean method (AFTER)

```
Updated 899 values in column 'chol' with mean value 198.76
Updated 899 values in column 'ekgmo' with mean value 5.97
Updated 899 values in column 'ekgday' with mean value 15.49
Updated 899 values in column 'ekgyr' with mean value 84.06
Updated 899 values in column 'thaldur' with mean value 8.66
Updated 899 values in column 'thalach' with mean value 137.30
Updated 899 values in column 'thalrest' with mean value 75.49
```

Figure 28 drops the unnecessary feature like ‘name’ which is the name of the patient, which is stored as ‘name’ for a value as well. This feature will be of no use. The ‘ccf’ is the SSN of the patient which has a value 0. Figure 29 shows that the unnecessary features have been dropped.

Figure 28

Removing unnecessary features (BEFORE)

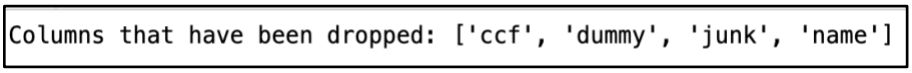


	ccf	name	junk	dummy
0	0	name	NaN	145.0
1	0	name	NaN	160.0
2	0	name	NaN	120.0
3	0	name	NaN	130.0
4	0	name	NaN	130.0
..	...	...	...	...
894	0	name	NaN	180.0
895	0	name	NaN	125.0
896	0	name	NaN	125.0
897	0	name	NaN	130.0
898	0	name	NaN	155.0

[899 rows x 4 columns]

Figure 29

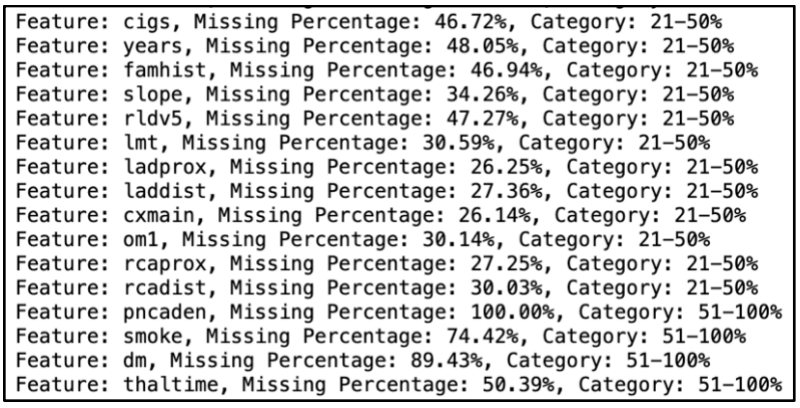
Removing unnecessary features (AFTER)



```
Columns that have been dropped: ['ccf', 'dummy', 'junk', 'name']
```

Figure 30

Remove features with missing values more than 20% with exception list (BEFORE)



```
Feature: cigs, Missing Percentage: 46.72%, Category: 21-50%
Feature: years, Missing Percentage: 48.05%, Category: 21-50%
Feature: famhist, Missing Percentage: 46.94%, Category: 21-50%
Feature: slope, Missing Percentage: 34.26%, Category: 21-50%
Feature: rldv5, Missing Percentage: 47.27%, Category: 21-50%
Feature: lmt, Missing Percentage: 30.59%, Category: 21-50%
Feature: ladprox, Missing Percentage: 26.25%, Category: 21-50%
Feature: laddist, Missing Percentage: 27.36%, Category: 21-50%
Feature: cxmain, Missing Percentage: 26.14%, Category: 21-50%
Feature: om1, Missing Percentage: 30.14%, Category: 21-50%
Feature: rcaprox, Missing Percentage: 27.25%, Category: 21-50%
Feature: rcadist, Missing Percentage: 30.03%, Category: 21-50%
Feature: pncaden, Missing Percentage: 100.00%, Category: 51-100%
Feature: smoke, Missing Percentage: 74.42%, Category: 51-100%
Feature: dm, Missing Percentage: 89.43%, Category: 51-100%
Feature: thaltime, Missing Percentage: 50.39%, Category: 51-100%
```

Figure 30 shows the features with missing than 20% of the value are dropped. There is an exception list of features created to not drop ‘'slope', 'ca', and 'thal'. This decision is grounded in the UCI Heart Disease dataset, where the primary analysis involves the 14 features which includes the 3 features ‘'slope', 'ca', and 'thal'. By preserving these features, we ensure their

availability for any subsequent analysis, study, or comparison. Figure 31 shows the dropped features missing more than 20% of values.

Figure 31

Remove features with missing values more than 20% with exception list (AFTER)

```
Location     0.000000
id           0.000000
age          0.000000
sex          0.000000
painloc      0.000000
              ...
lvx2         0.000000
lvx3         0.000000
lvx4         0.000000
lvf          1.779755
cathef      65.406007
Length: 73, dtype: float64
20
['pncaden', 'smoke', 'cigs', 'years', 'dm', 'famhist', 'thaltime', 'rldv5', 'restckm', 'exerckm', 'restef', 'restw
m', 'exeref', 'exerwm', 'thalsev', 'thalpul', 'earlobe', 'diag', 'ramus', 'om2', 'cathef']
```

Figure 32 shows the matrix visualization after the data cleaning process. We do not see any missing values for the features.

Figure 32

missingno Matrix to visualize missing data (AFTER CLEANING)

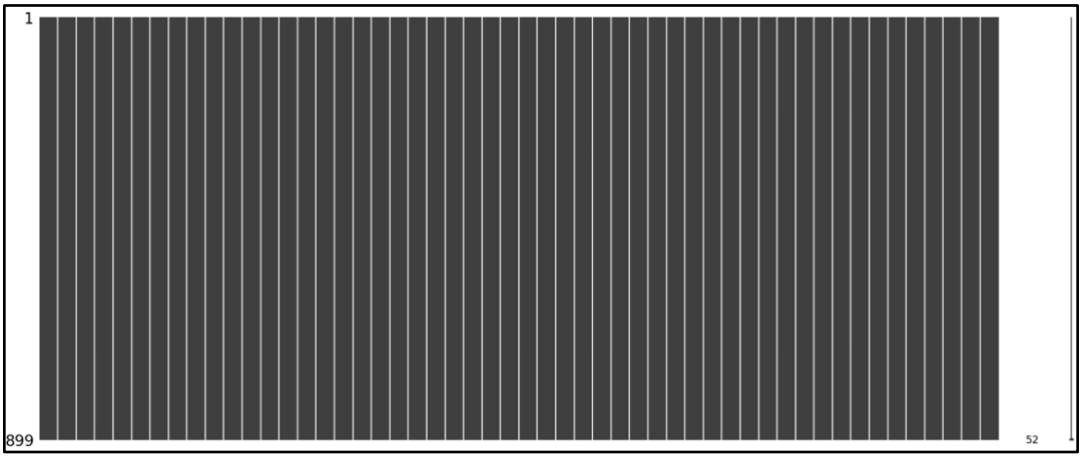


Figure 33 shows the code for identifying outliers in the DataFrame. This code specifically examines continuous features, excluding those with a cardinality of less than 4. The models used by the team is robust to outliers. We have decided to retain the outliers.

Figure 33

Check records for outliers, with sample records shown

```
For the feature trestbps, No of Outliers is 28
For the feature chol, No of Outliers is 185
For the feature proto, No of Outliers is 2
For the feature thaldur, No of Outliers is 40
For the feature met, No of Outliers is 73
For the feature thalach, No of Outliers is 3
For the feature thalrest, No of Outliers is 13
For the feature tpeakbps, No of Outliers is 11
For the feature tpeakbpd, No of Outliers is 10
For the feature trestbpd, No of Outliers is 30
For the feature oldpeak, No of Outliers is 14
For the feature rldv5e, No of Outliers is 17
For the feature cyr, No of Outliers is 3
For the feature lvx1, No of Outliers is 25
For the feature lvx2, No of Outliers is 28
For the feature lvx3, No of Outliers is 56
For the feature lvx4, No of Outliers is 130
```

Figure 34

Boxplot showing outliers for 'chol' feature

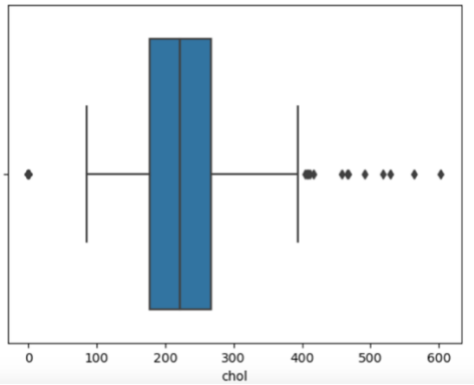
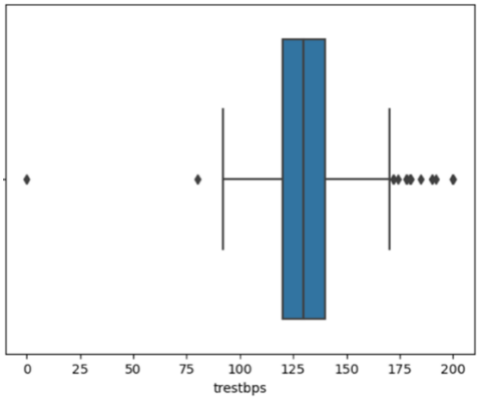


Figure 35

Boxplot showing outliers for 'trestbps' feature



Figures 34 and 35 display boxplot visualizations for the 'chol' (cholestrol) and 'trestbps' (resting blood pressure) features, highlighting the presence of outliers in these continuous variables.

3.4. Data Transformation

Encoding Target Variable

Figure 36 shows using the get_dummies function from the pandas library to perform one-hot encoding on the 'Location' column of the DataFrame. One-hot encoding is a technique used to convert categorical variables into a binary matrix format. Figure 37 shows the encoded ‘Location’ feature. Once the ‘Location’ feature is encoded, the original feature is dropped from the DataFrame.

Figure 36

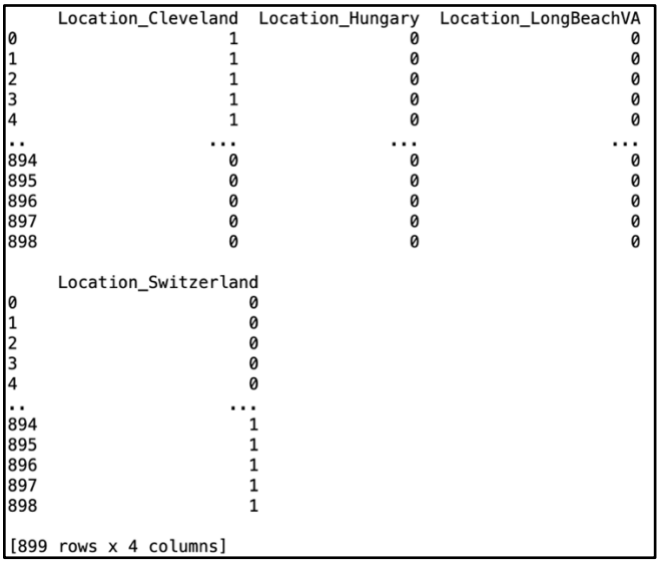
Encoding categorical feature, ‘Location’ (BEFORE)

Location	id	age	sex	painloc	painexer	relrest	cp	trestbps
Cleveland	16	52	1	1.0	1.0	1.0	3	172.0
Cleveland	89	68	1	1.0	1.0	1.0	3	180.0
Cleveland	136	56	0	1.0	1.0	1.0	4	200.0
Cleveland	188	59	0	1.0	1.0	1.0	4	174.0
Cleveland	200	59	1	1.0	1.0	1.0	1	178.0
Cleveland	205	54	1	1.0	1.0	1.0	2	192.0
Cleveland	218	64	0	1.0	1.0	1.0	4	180.0
Cleveland	230	66	0	1.0	1.0	1.0	4	178.0
Cleveland	248	55	0	1.0	1.0	1.0	4	180.0
Hungary	1235	39	1	1.0	0.0	0.0	2	190.0
Hungary	1249	58	0	1.0	0.0	0.0	2	180.0
Hungary	1064	53	1	1.0	1.0	1.0	4	180.0
Hungary	1065	46	1	1.0	1.0	1.0	4	180.0
Hungary	1005	54	1	1.0	1.0	1.0	4	200.0
Hungary	1038	45	0	0.0	0.0	0.0	2	180.0
Hungary	1039	59	1	1.0	0.0	1.0	3	180.0

The original target variable is illustrated in Figure 38. The newly encoded target feature is shown in Figure 39. The original and the newly encoded target feature exist within the DataFrame, for any additional analyses.

Figure 37

Encoding categorical feature, 'Location' (AFTER)



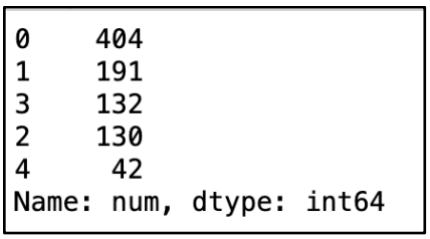
```
     Location_Cleveland  Location_Hungary  Location_LongBeachVA
0                     1                 0                     0
1                     1                 0                     0
2                     1                 0                     0
3                     1                 0                     0
4                     1                 0                     0
..                  ...               ...                   ...
894                   0                 0                     0
895                   0                 0                     0
896                   0                 0                     0
897                   0                 0                     0
898                   0                 0                     0

     Location_Switzerland
0                       0
1                       0
2                       0
3                       0
4                       0
..                    ...
894                     1
895                     1
896                     1
897                     1
898                     1

[899 rows x 4 columns]
```

Figure 38

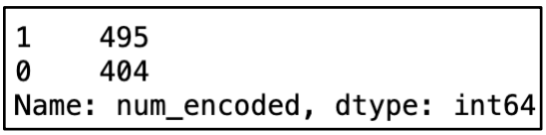
DataFrame with original target feature



```
0    404
1    191
3    132
2    130
4     42
Name: num, dtype: int64
```

Figure 39

DataFrame with newly encoded target feature



```
1    495
0    404
Name: num_encoded, dtype: int64
```

In Figure 40 the code is using the IterativeImputer from scikit-learn to perform imputation (filling in missing values) on the DataFrame. The imputation is based on Bayesian Ridge Regression by default.

Figure 40

Additional cleansing for 3 features using IterativeImputer (BEFORE)

```
restecg      2
ca         608
lvf         16
dtype: int64
```

The IterativeImputer from scikit-learn is used when dealing with datasets that contain missing values and when you want to impute those missing values based on the observed values in other columns. The 'ca' feature is missing 67% of the data.

The UCI Heart Disease dataset where the primary analysis involves the 14 features and one of the features is 'ca'. We decided to include this feature as part of our modeling and analysis. Figure 41 shows the outcome of using the IterativeImputer.

Figure 41

Additional cleansing for 3 features using IterativeImputer (AFTER)

	ca	restecg	lvf
count	899.000000	899.000000	899.000000
mean	1.132168	0.602574	1.183335
std	0.821611	0.802888	0.509799
min	-0.285430	0.000000	0.000000
25%	0.501552	0.000000	1.000000
50%	1.142465	0.000000	1.000000
75%	1.632400	1.000000	1.000000
max	9.000000	2.000000	5.000000

Dimensionality Reduction

Dimensionality reduction is done to reduce the number of dimensions or columns present in a dataset. It transforms the dataset to have fewer dimensions while still containing most information. The main issue with the dataset post all previous pre-processing steps is that there

are 50 features left to study. There were a few considerations and the team decided to go with dimensionality reduction techniques to handle it.

The reason is that we cannot build baseline models using a technique such as an ablation study which basically leaves one feature out while building the model in multiple iterations as it is a computationally expensive process and therefore Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) were chosen to select the most significant features that contribute most to the target variable.

Dimensionality reduction reduces computational costs and helps mitigate the curse of dimensionality, potentially improving model performance by removing noise and redundant features. It also aids in visualizing high-dimensional data by projecting it onto lower-dimensional spaces for better interpretation and insights.

Principal Component Analysis (PCA). PCA turns related variables into principal components, a new set of unrelated variables. The first component takes up most of the data's spread, and each one after that captures less. Ordered by how much of the data's spread they capture, shown by their eigenvalues, these components reduce the data's complexity by keeping only the most informative parts.

Eigen value indicates the amount of variance in the direction of its corresponding eigenvector and the Eigen vector is the direction of maximum variance in the data. Normalizing or standardising the data before applying PCA is recommended to prevent a few components from dominating due to their scale. The standardised training data is shown in Figure 42. Similarly, the test data is also standardised before applying PCA.

Figure 42

Training Data after Standardisation

	age	sex	painloc	painexer	relrest	cp	trestbps	htn	chol	fbs	...	cxmain	om1	rcaprox	rcadist	lvx1	lvx2	lvx3	lvx4	lvf	target
0	2.150388	0.511276	-4.420730	-1.627804	-1.876703	-1.391787	0.676765	1.067652	-1.812612	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	0.030945	0.000538	0.055681	0.003910	1.687619	1.0
1	0.892606	0.511276	0.226207	-1.627804	-1.876703	-1.391787	0.409648	-0.936635	0.647353	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	0.0
2	1.102236	0.511276	0.226207	0.614325	0.532849	0.802720	-0.124587	1.067652	0.529347	2.315779	...	-0.547888	-0.383054	1.724058	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	1.0
3	-0.365177	0.511276	0.226207	0.614325	0.532849	0.802720	0.409648	1.067652	-0.641633	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	0.0
4	0.997421	0.511276	0.226207	0.614325	0.532849	0.802720	-0.124587	-0.936635	1.182917	2.315779	...	1.825192	2.610597	1.724058	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
714	0.578160	0.511276	0.226207	0.614325	0.532849	0.802720	0.409648	1.067652	-0.205919	-0.431820	...	1.825192	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	1.0
715	0.787791	0.511276	0.226207	0.614325	0.532849	0.802720	0.409648	-0.936635	0.066401	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	1.0
716	1.207051	0.511276	0.226207	0.614325	0.532849	0.802720	1.478117	1.067652	-1.812612	2.315779	...	-0.547888	-0.383054	1.724058	-0.375859	-0.072970	-0.075728	-0.171137	3.167493	1.687619	1.0
717	0.158899	0.511276	0.226207	0.614325	0.532849	0.802720	-0.658822	-0.936635	0.638275	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	0.0
718	0.368530	-1.955889	0.226207	0.614325	0.532849	0.802720	-0.231434	1.067652	0.937828	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	0.0

719 rows × 51 columns

Figure 43

Training Data after PCA with 40 Components

	1	2	3	4	5	6	7	8	9	10	...	32	33	34	35	36	37	38	39	40	target
0	-0.766935	-2.353474	0.666855	1.790006	2.471166	2.477473	0.962876	-2.439171	0.404615	-1.986088	...	1.710136	0.313500	-0.651644	-0.216032	-0.820337	-0.844128	-0.591588	0.538382	0.012462	1.0
1	-3.732850	-2.177280	-1.935404	1.336955	-0.652893	0.839545	0.218358	-0.386887	-0.144375	-0.500492	...	-0.443013	0.311716	-0.672536	-1.066886	-0.055299	0.003194	-0.539007	0.788195	-0.187139	0.0
2	1.843476	-0.389318	-0.046146	1.063272	0.402922	-0.240648	0.047470	0.841115	0.135274	-2.172294	...	-0.168109	-0.205389	-0.774231	-0.651174	0.174469	-0.121232	0.297091	-1.092843	-0.510920	1.0
3	-2.048637	-0.149163	0.158482	-1.459777	-1.914820	-0.667933	-0.366468	1.192296	1.081936	-0.271876	...	0.575804	-0.208756	0.483045	0.270546	0.331686	-0.144162	0.006243	0.144280	-0.086948	0.0
4	4.328346	1.455783	-1.130231	0.692913	-0.472766	1.465060	-0.779669	-2.449843	-1.778051	-0.790932	...	-0.673324	0.403204	-0.510762	1.919140	-0.062170	-0.202954	-0.479562	0.876458	0.032876	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
714	1.762246	1.532235	0.036873	0.364361	-1.307962	0.481195	0.663398	0.225311	-0.153818	-0.526096	...	-0.810439	0.109616	-1.039656	-0.139942	1.744363	0.886578	0.243788	-0.111357	-0.340558	1.0
715	1.287805	1.787164	-0.403692	0.910878	-1.444662	0.448444	-0.452303	-1.021009	-0.918742	0.508371	...	-0.725539	-0.616482	-1.587135	0.294064	-0.221532	-0.403016	0.156068	-0.448974	-0.259074	1.0
716	4.576414	-1.701957	0.571544	2.505701	1.047498	2.503107	0.723274	2.139162	1.321981	-4.264630	...	0.634040	-1.640712	-0.407667	-1.701723	-1.475635	-0.548300	1.359511	0.597624	0.928807	1.0
717	-2.055850	-0.752798	0.085860	-2.090918	-0.781065	-1.193076	2.593513	0.790864	-0.904424	-0.437684	...	0.464635	0.168204	0.303898	-0.379718	0.230896	0.161814	0.237173	0.280401	-0.086077	0.0
718	-1.154553	3.183363	-0.157419	0.639956	-0.572280	0.132195	-0.450627	0.626799	0.271956	0.168394	...	-0.071319	0.088282	-0.856900	-0.189416	0.260927	-0.030486	-0.175321	0.424356	-0.103499	0.0

719 rows × 41 columns

The team experimented with different values for “n_components” which is the number of components. This is fed into the PCA function of the “sklearn.decomposition” library. The results are discussed in detail under the “Baseline Modeling using PCA and LDA” section.

The dataset sample after applying Principal Component Analysis with 23 components is shown in Figure 43. After PCA, the shape of the training dataset is 719 rows and 24 columns (the target variable is dropped during baseline modelling and is not part of the feature set).

Linear Discriminant Analysis (LDA). Linear Discriminant Analysis (LDA) is a supervised technique that uses known class labels to find the axes which best separate different classes. It calculates within-class scatter (Sw) and between-class scatter (Sb) to maximize the Sb/Sw ratio, aiming to spread out different classes as far as possible while keeping data points of

the same class close together. This method is useful for both simplifying data for classification and for visualizing complex datasets with clear class distinctions.

We used Linear Discriminant Analysis (LDA) from the “sklearn.discriminant_analysis” library to separate classes in our dataset clearly. LDA works by finding the best directions to spread out different groups while keeping the same group data points close. Our "Baseline Modeling Using PCA and LDA" section talks about how LDA reduced our dataset's dimensions and made class boundaries clearer for better classification.

Baseline Modeling Using PCA and LDA. Thorough experiments were conducted to gain a deeper understanding of the outcomes after using PCA and LDA respectively after building a Baseline Model using a Random Forest Classifier from the sklearn library. This baseline model was further evaluated based on the accuracy and run-time documented.

The data was split into train and test sets in the ratio of 80:20. Figure 44 shows the training data sample containing 719 rows and 24 rows. Similarly, the Figure 45 shows the test data sample including 180 rows and 24 columns. It is to be noted that the target variable has been included as a reference and is not included in the feature set for the modeling.

Figure 44

Training Data Sample

	age	sex	painloc	painexer	relrest	cp	trestbps	htn	chol	fbs	...	cxmain	om1	rcaprox	rcadist	lvx1	lvx2	lvx3	lvx4	lvf	target
0	2.150388	0.511276	-4.420730	-1.627804	-1.876703	-1.391787	0.676765	1.067652	-1.812612	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	0.030945	0.000538	0.055681	0.003910	1.687619	1.0
1	0.892606	0.511276	0.226207	-1.627804	-1.876703	-1.391787	0.409648	-0.936635	0.647353	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	0.0
2	1.102236	0.511276	0.226207	0.614325	0.532849	0.802720	-0.124587	1.067652	0.529347	2.315779	...	-0.547888	-0.383054	1.724058	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	1.0
3	-0.365177	0.511276	0.226207	0.614325	0.532849	0.802720	0.409648	1.067652	-0.641633	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	0.0
4	0.997421	0.511276	0.226207	0.614325	0.532849	0.802720	-0.124587	-0.936635	1.182917	2.315779	...	1.825192	2.610597	1.724058	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
714	0.578160	0.511276	0.226207	0.614325	0.532849	0.802720	0.409648	1.067652	-0.205919	-0.431820	...	1.825192	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	1.0
715	0.787791	0.511276	0.226207	0.614325	0.532849	0.802720	0.409648	-0.936635	0.066401	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	1.0
716	1.207051	0.511276	0.226207	0.614325	0.532849	0.802720	1.478117	1.067652	-1.812612	2.315779	...	-0.547888	-0.383054	1.724058	-0.375859	-0.072970	-0.075728	-0.171137	3.167493	1.687619	1.0
717	0.158899	0.511276	0.226207	0.614325	0.532849	0.802720	-0.658822	-0.936635	0.638275	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	0.0
718	0.368530	-1.955889	0.226207	0.614325	0.532849	0.802720	-0.231434	1.067652	0.937828	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.072970	-0.075728	-0.171137	-0.355012	-0.348856	0.0

719 rows × 51 columns

Figure 45

Testing Data Sample

	age	sex	painloc	painexer	relrest	cp	trestbps	htn	chol	fbs	...	cxmain	om1	rcaprox	rcadist	lvx1	lvx2	lvx3	lvx4	lvf	target
0	-0.365177	0.511276	0.226207	0.614325	0.532849	0.802720	-0.124587	-0.936635	0.302413	-0.431820	...	-0.547888	-0.383054	1.724058	-0.375859	-0.07297	-0.075728	-0.171137	3.754578	1.687619	1.0
1	-0.155546	0.511276	0.226207	0.614325	0.532849	0.802720	-1.994409	-0.936635	-1.812612	-0.431820	...	1.825192	-0.383054	1.724058	-0.375859	-0.07297	-0.075728	-0.171137	-0.355012	0.509724	1.0
2	1.102236	0.511276	0.226207	0.614325	0.532849	0.802720	-0.012334	1.067652	0.402264	2.315779	...	1.825192	-0.383054	1.724058	-0.375859	-0.07297	-0.075728	-0.171137	3.167493	-0.348856	1.0
3	0.682975	-1.955889	0.226207	0.614325	0.532849	-0.294534	-0.658822	-0.936635	-0.196842	2.315779	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.07297	-0.075728	-0.171137	-0.355012	-0.348856	0.0
4	0.787791	0.511276	0.226207	0.614325	0.532849	-0.294534	0.943882	1.067652	0.393186	2.315779	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.07297	-0.075728	-0.171137	-0.355012	-0.348856	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
175	0.787791	0.511276	0.226207	0.614325	0.532849	0.802720	-1.460174	-0.936635	-1.812612	-0.431820	...	-0.547888	-0.383054	1.724058	-0.375859	-0.07297	-0.075728	-0.171137	-0.355012	-0.348856	1.0
176	-0.365177	-1.955889	0.226207	0.614325	0.532849	-0.294534	-0.658822	-0.936635	0.175330	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.07297	-0.075728	-0.171137	-0.355012	-0.348856	0.0
177	-0.155546	0.511276	0.226207	0.614325	0.532849	-1.391787	-0.658822	1.067652	1.137530	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.07297	-0.075728	-0.171137	-0.355012	-0.348856	0.0
178	0.368530	0.511276	0.226207	0.614325	0.532849	0.802720	0.409648	-0.936635	-0.069759	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.07297	-0.075728	-0.171137	-0.355012	-0.348856	0.0
179	-0.679622	-1.955889	0.226207	0.614325	0.532849	0.802720	-0.658822	-0.936635	0.048247	-0.431820	...	-0.547888	-0.383054	-0.580027	-0.375859	-0.07297	-0.075728	-0.171137	-0.355012	-0.348856	1.0

180 rows × 51 columns

Next, PCA is applied with varying numbers of components, and results are saved to a CSV for documentation and analysis. After preliminary feature selection, the dataset comprises 899 records and 50 features. PCA is beneficial as it simplifies the dataset by reducing the number of features, focusing on those that capture the most variance. This reduction is crucial in preventing overfitting, especially when the dataset isn't very large such as the UCI Heart Disease Dataset. By highlighting key patterns in the heart disease data, PCA can improve the performance of machine learning models designed to predict heart disease. PCA was applied with various component sizes ranging from 5 to 50 as seen in Table 12. With the component size as 10, 15, an optimal accuracy of 96% was achieved. The resulting accuracies and run-time in seconds for both PCA and LDA can be referred in Table. PCA on average has an accuracy of 95% for this data.

LDA was applied subsequently. LDA was initially chosen as it is specifically designed to enhance class separation, which is key in a diagnostic setting. In other words, LDA looks for patterns that make the classes stand out from each other. This would lead to better results when creating models to predict heart disease. The Random Forest model achieved an accuracy of 94.4% after applying LDA dimensionality technique. The results are documented in Table 12.

Table 12

Accuracy and Runtime of PCA and LDA

Type	Number of Components	Accuracy	Run-Time in Seconds
PCA	5	0.905	0.168
PCA	10	0.911	0.208
PCA	15	0.894	0.210
PCA	20	0.922	0.232
PCA	25	0.922	0.266
PCA	30	0.905	0.270
PCA	35	0.927	0.310
PCA	40	0.933	0.307
PCA	45	0.911	0.309
PCA	50	0.933	0.368
LDA	-	0.944	0.12

After thoroughly analysing the outcomes of all the conducted experiments, Principal Component Analysis (PCA) stood out as the most effective dimensionality reduction method and the team has chosen to employ this technique.

In selecting the optimal number of PCA components for our model, we considered both the accuracy and the computational cost measured by the run-time. The analysis revealed that models with 40 and 50 PCA components achieved the highest test accuracy, at 93.3%. However, the model with 40 components was faster, with a run-time of approximately 0.35 seconds, compared to 0.39 seconds for the model with 50 components. Therefore, we decided to proceed with the model using 40 PCA components, as it provides a balance between high accuracy and computational efficiency. Though LDA achieves a commendable accuracy of 94.4% with a

notably fast run-time of 0.12 seconds, PCA is chosen as the technique for Dimensionality Reduction.

3.5. Data Preparation

After completing data pre-processing and transformation, the next step is to ready the dataset for modeling. Data preparation for heart disease prediction involves the standardization and readying the dataset to establish a uniform structure for modelling. The dataset is then structured into two essential columns: 'features,' containing pertinent information, and 'target,' indicating the absence (0) or presence (1) of heart disease. Subsequently, a strategic 80-10-10 split is performed on the training dataset, with 80% allocated for model training and 10% each for testing and validation.

The dataset sizes are as follows: 719 samples for training, 90 for validation, and 90 for testing. This meticulous split ensures a consistent data distribution across the three sets, fostering unbiased and reliable data for model training and evaluation. To ensure the separation of features from the target variable, the latter is dropped from the feature set before the modeling phase.

Thus, establishing a foundation for developing an accurate and reliable machine learning model for heart disease prediction.Visual representations of the sample data for each set and its structure, are illustrated in Figure 46 – Figure 48, further enhancing interpretability.

Figure 46

Sample from the Training Dataset

	1	2	3	4	5	6	7	8	9	10	...	32	33	34	35	36	37	38	39	40	target
0	-0.766935	-2.353474	0.666855	1.790006	2.471166	2.477473	0.962876	-2.439171	0.404615	-1.986088	...	1.710136	0.313500	-0.651644	-0.216032	-0.820337	-0.844128	-0.591588	0.538382	0.012462	1.0
1	-3.732850	-2.177280	-1.935404	1.336955	-0.652893	0.839545	0.218358	-0.386887	-0.144375	-0.500492	...	-0.443013	0.311716	-0.672536	-1.066886	-0.055299	0.003194	-0.539007	0.788195	-0.187139	0.0
2	1.843476	-0.389318	-0.046146	1.063272	0.402922	-0.240648	0.047470	0.841115	0.135274	-2.172294	...	-0.168109	-0.205389	-0.774231	-0.651174	0.174469	-0.121232	0.297091	-1.092843	-0.510920	1.0
3	-2.048637	-0.149163	0.158482	-1.459777	-1.914820	-0.667933	-0.366468	1.192296	1.081936	-0.271876	...	0.575804	-0.208756	0.483045	0.270546	0.331686	-0.144162	0.006243	0.144280	-0.086948	0.0
4	4.328346	1.455783	-1.130231	0.692913	-0.472766	1.465060	-0.779669	-2.449843	-1.778051	-0.790932	...	-0.673324	0.403204	-0.510762	1.919140	-0.062170	-0.202954	-0.479562	0.876458	0.032876	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
714	1.762246	1.532235	0.036873	0.364361	-1.307962	0.481195	0.663398	0.225311	-0.153818	-0.526096	...	-0.810439	0.109616	-1.039656	-0.139942	1.744363	0.886578	0.243788	-0.111357	-0.340558	1.0
715	1.287805	1.787164	-0.403692	0.910878	-1.444662	0.448444	-0.452303	-1.021009	-0.918742	0.508371	...	-0.725539	-0.616482	-1.587135	0.294064	-0.221532	-0.403016	0.156068	-0.448974	-0.259074	1.0
716	4.576414	-1.701957	0.571544	2.505701	1.047498	2.503107	0.723274	2.139162	1.321981	-4.264630	...	0.634040	-1.640712	-0.407667	-1.701723	-1.475635	-0.548300	1.359511	0.597624	0.928807	1.0
717	-2.055850	-0.752798	0.085860	-2.090918	-0.781065	-1.193076	2.593513	0.790864	-0.904424	-0.437684	...	0.464635	0.168204	0.303898	-0.379718	0.230896	0.161814	0.237173	0.280401	-0.086077	0.0
718	-1.154553	3.183363	-0.157419	0.639956	-0.572280	0.132195	-0.450627	0.626799	0.271956	0.168394	...	-0.071319	0.088282	-0.856900	-0.189416	0.260927	-0.030486	-0.175321	0.424356	-0.103499	0.0

719 rows × 41 columns

Figure 47

Sample from the Validation Dataset

	1	2	3	4	5	6	7	8	9	10	...	32	33	34	35	36	37	38	39	40	target
0	4.027235	-1.208855	2.382666	-1.490612	1.471426	1.946134	-0.463684	0.575949	0.687090	2.246827	...	1.010056	-3.212606	0.060761	-4.535548	-0.708055	-0.212160	-0.110899	0.471038	1.612919	1.0
1	-0.299588	2.740717	-0.883797	-1.051960	0.612017	-0.261867	0.589333	0.830389	2.029004	-1.118576	...	0.403736	-0.411271	1.175541	0.199788	-0.709673	-0.057885	0.273499	0.290625	0.671585	0.0
2	4.742652	-2.150197	2.022480	2.053729	-1.942904	0.421636	0.950476	-0.530450	0.746796	-0.592331	...	-0.116253	-1.028769	-0.284396	-0.248165	-0.200482	0.092726	-0.055671	-0.440314	-0.387434	1.0
3	3.642856	-1.335302	2.745068	0.348092	-4.206645	3.516037	-0.433059	-2.082450	0.098109	-0.741439	...	-0.686204	-0.684754	-1.319149	0.625995	-1.768072	1.458167	-0.330908	0.066761	1.491952	1.0
4	9.068448	-4.371941	3.567319	-4.876332	4.186393	10.287135	-2.649089	7.947423	-3.699511	4.207660	...	-1.030756	-3.080694	-0.421101	-1.159176	-0.667355	1.178084	-1.750864	1.015000	-0.588737	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	2.612977	-2.045695	0.276530	-0.570358	2.233862	0.482197	-0.250445	3.691157	-3.380202	-1.572645	...	-0.600315	-0.744156	-1.818869	-0.265212	-1.565192	0.437648	-2.591140	1.840803	-1.519305	1.0
86	0.763549	0.830619	1.625899	-2.102867	0.570731	-2.224988	1.020102	-0.245991	-0.562334	0.660858	...	-0.810355	1.144991	0.500716	0.500708	-0.109506	-0.509059	-0.514900	-0.339596	-0.361709	1.0
87	-3.682426	-1.213354	1.599450	-0.401168	1.170228	-0.412823	0.760563	-0.779524	0.075499	0.455980	...	-0.424761	0.367584	-1.181349	-0.390730	0.085740	0.066691	0.023064	0.208902	0.031795	0.0
88	1.129205	1.025513	2.163184	-0.073920	-0.998707	-1.316303	1.415864	0.662023	1.577756	-0.930364	...	-1.043207	0.606110	-1.399164	-0.334714	0.632390	-1.374277	-0.452751	0.847350	0.081500	0.0
89	0.927789	0.148099	-0.666059	-0.070758	-0.168624	-0.323310	-0.121596	-0.577819	-0.382220	-0.795173	...	0.790104	-0.704514	-0.172941	0.930994	0.709396	0.395248	-0.552216	0.065953	0.950717	1.0

90 rows × 41 columns

Figure 48

Sample from the Test Dataset

	1	2	3	4	5	6	7	8	9	10	...	32	33	34	35	36	37	38	39	40	target
0	1.963897	3.312018	-0.492742	0.716432	0.500319	-0.667362	1.842020	-0.149831	0.149125	0.247344	...	-0.376934	-0.739054	-1.299408	-0.078289	0.736444	0.574571	0.391968	-1.113413	0.192045	1.0
1	2.605402	-0.010679	-0.465319	1.484976	-0.161494	0.168188	1.773480	1.050517	2.742299	-2.438317	...	0.199789	-0.341554	-0.202215	0.495685	1.251404	-0.912478	-0.012528	-0.887978	-0.594072	1.0
2	1.465118	1.901726	0.924220	-2.026664	-1.002603	0.007550	-0.973233	-1.583329	-1.028633	-0.515163	...	0.362198	-0.453499	-0.029237	-0.556419	0.769007	-0.404074	-0.062946	-1.099522	-0.334097	1.0
3	-3.942047	-0.208087	1.525755	-1.393688	1.369831	-0.149219	0.091647	-0.487857	-0.063387	0.208683	...	-0.405234	-0.392000	0.846692	0.212451	0.023352	0.284554	0.005036	-0.413273	0.018656	0.0
4	2.444208	0.407474	-3.205163	1.137219	-1.037876	1.046500	-1.041223	-0.712243	-0.040577	1.055758	...	-0.769215	-0.176596	-0.803753	-0.195633	-0.112768	-0.041968	-0.675655	-0.541130	0.583435	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	-0.860501	1.944474	-2.174360	1.097662	0.281725	1.619351	-0.912577	0.078777	2.175257	-0.870730	...	-0.079752	0.060890	0.473737	0.813514	-0.424143	1.077358	0.353909	-0.179338	0.277537	0.0
86	-4.096910	-2.866549	-1.681153	1.511321	-0.726947	0.848420	0.152375	-0.126661	-0.408773	-0.500199	...	-0.591350	-0.127535	-0.464372	-0.456002	-0.401066	0.232518	-0.145193	-0.404850	0.315576	0.0
87	-1.871070	2.930804	-0.306529	-0.233074	-0.223521	0.677403	-0.300307	0.009680	-0.049187	-0.220039	...	-0.191732	-0.293560	0.662368	0.031777	-0.124635	0.452857	0.426775	-0.477175	-0.022281	0.0
88	-1.682366	2.864284	-0.729686	2.321626	0.458405	0.292697	0.186423	0.556544	-1.724291	-0.261264	...	-0.168151	0.604138	0.169979	-0.849006	-1.523453	-0.442773	-0.593973	-0.345359	0.978618	0.0
89	2.732532	1.710480	-1.079449	-0.030737	-0.210170	1.094822	-0.126636	-1.412591	-0.977046	-0.461310	...	-0.056780	0.730190	-0.477042	0.105322	-1.206439	-0.396060	-0.587205	0.208510	0.332619	1.0

90 rows × 41 columns

3.6. Data Statistics

Table 13 shows how the datasets change through various processes followed during phases of our project. The raw dataset collected from UCI repository consists of 899 rows and 76 columns after combining Cleveland, Switzerland, Hungary and Long Beach VA.

Table 13

Summary of Dataset Sizes After Different Processes

Stage	**Process**	**After process [Row X Col]**
Raw	Cleveland	282 X 75
	Hungary	294 X 75
	Switzerland	123 X 75
	Long Beach VA	200 X 75

Pre-processing	Cleveland	282 X 52
	Hungary	294 X 52
	Switzerland	123 X 52
	Long Beach VA	200 X 52
	Merge	899 X 52
Transformation	Dimensionality Reduction using PCA	41
Preparation	Training Data	719 X 41
	Testing Data	90 X 41
	Validation Data	90 X 41

After the pre-processing step, unnecessary columns, rows with missing values are removed. In the dataset collected from UCI for four countries, we dropped unnecessary columns, deleted rows containing null value, selected rows labeled as 0 absence and 1 presence of heart disease, we have a dataset with 899 rows and 52 columns.

During data transformation, the dataset resulting from feature extraction using PCA has 719 rows and 41 features, i.e., 40 descriptive features and one target feature. To reduce the high dimensionality, we apply PCA and reduce the number of features to 41. At the same time, the row count remains the same at 719, which helps in reducing the overall size of the data and improves the processing speed. The target feature is added back to the reduced dataset, and we end up with a dataset containing 719 rows and 41 columns.

Finally, in the data preparation step, we split the dataset by the ratio of 80:10:10, resulting in 719 rows and 41 features, 90 rows and 41 features, 90 rows and 41 features for training, testing, and validation, respectively. Statistical analysis is performed to check the distribution of data in train, test, and validation datasets to ensure quality and check for any bias that might have been introduced during the processing, transformation, or preparation steps. Figures 49, 50, and

51 show frequency distribution in the train, test, and validation datasets respectively. The graphs show that the distribution ratio of heart disease present (1) and heart disease absence (0) is the same as within the distribution of labels in the actual dataset.

Figure 49

Distribution Frequency of Class Labels in the Training Dataset

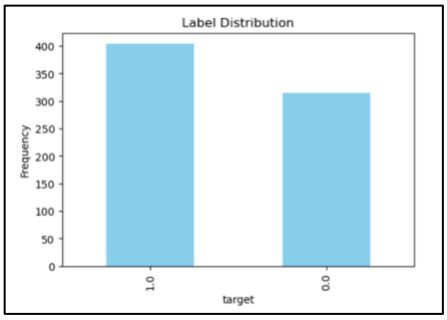


Figure 50

Distribution Frequency of Label in the Testing Dataset

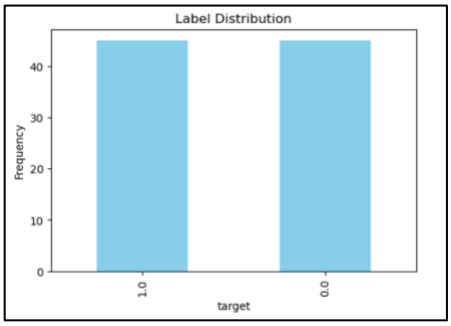
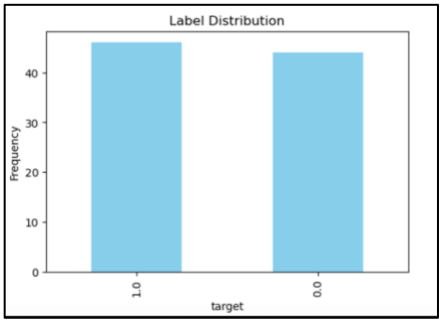


Figure 51

Distribution Frequency of Label in the Validation Dataset



Data Analytics Results

To visualize and explore high-dimensional data, we can use Principal component analysis (PCA), which helps plot data samples to show the data distribution in lower dimension spaces like two and three dimensions. When we plot high-dimensional data in low dimension, there is no meaning associated with the axes; however, the relative distance between these low-dimension points reveals the relevant information showing that neighboring points in high-dimensional data will tend to follow the same trend in the low-dimensional data structure.

Figure 52

T*he PCA Plot in 2D*

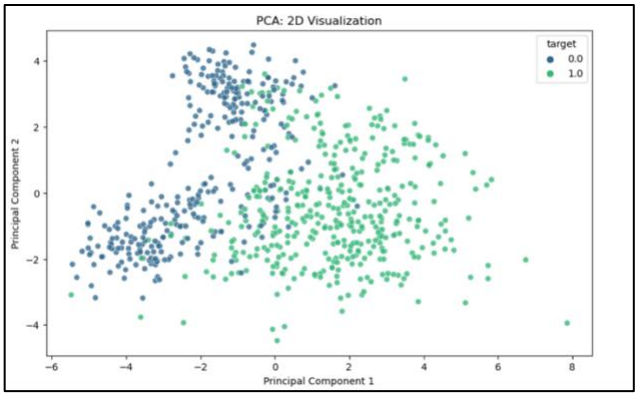
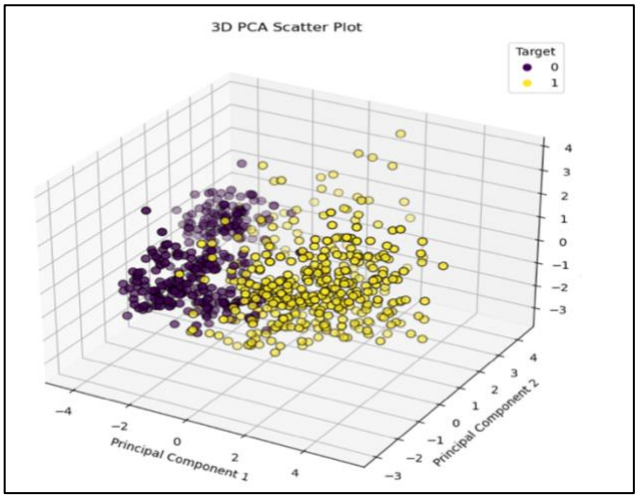


Figure 53

The PCA Plot in 3D



In Figure 52, green datapoints show the presence of cardio risk disease in the patient whereas blue data point shows the absence of Cardio risk disease. So, if points are close to each other in the low-dimensional plot, it suggests that they exhibit similar characteristics or trends in the original high dimensional space.

Figure 52 shows the data plotted after PCA in two dimensions. The data is closely knitted, and there is an equal distribution of data between heart disease presence and absence. Figure 53 shows the data plotted after PCA in three dimensions.

Figure 54

Discriminative performance of Classes after LDA

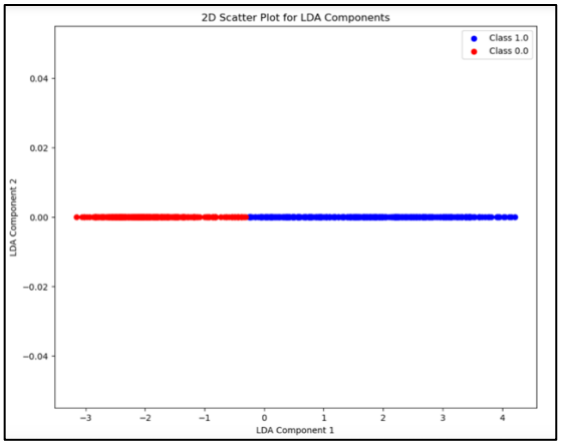


Figure 55

Distribution in bars plotted for types of Chest Pain

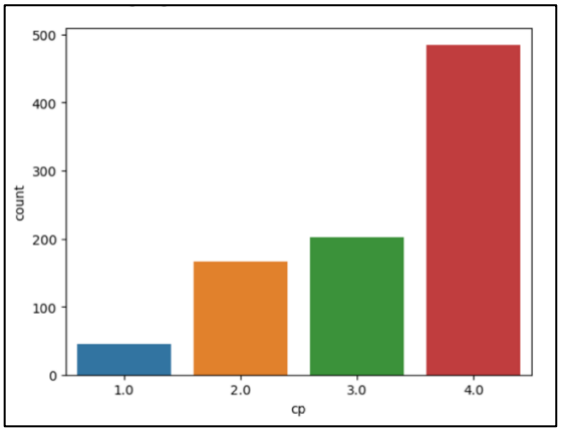


Figure 54 shows the data plotted after LDA in two dimensions. The data is closely knitted, and the distribution shows data between heart disease presence and absence, where class 0 represents absence and class 1 represents presence of heart disease. This visualization succinctly captures the class separation achieved by LDA in the reduced space, providing a quick overview of its discriminative performance.

Figure 55 shows the type of chest pain. The data shows 4 types of heart pain with values as 1 (typical angina), 2 (atypical angina), 3 (non-anginal pain) and 4 (asymptomatic).

Figure 56

Distribution of Patients in terms of Age and Sex

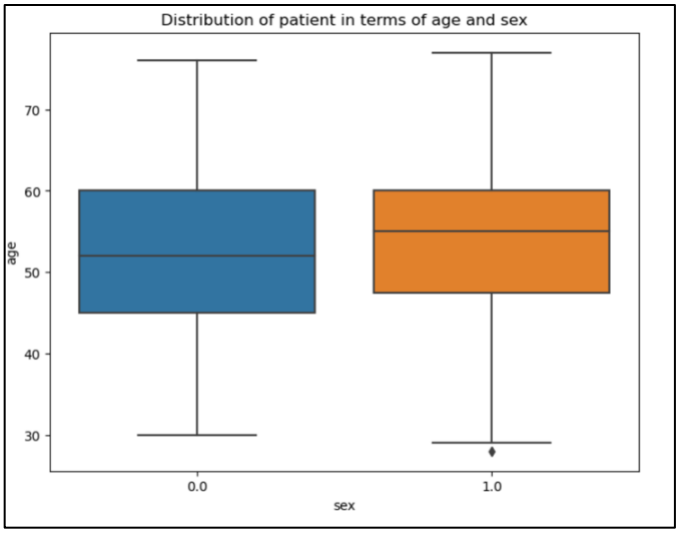


Figure 56 shows a box plot for sex and age of the patients more prone to have cardio disease. Where sex is represented by 0 and 1, 0 is male and 1 is female. It is evident from the box plot females between the ages of 48 to 60 are more prone to have heart disease.

4. Model Development

4.1. Model Proposals

This project aims to construct a supervised classification model for predicting heart disease sourced from the UCI repository. The heart disease prediction dataset, comprising various features extracted from patient vital records. Considering the binary nature of the target features, classification models are employed to categorize the output as '1' for the presence of heart disease risk and '0' for the absence of risk.

Given the likelihood of repeated combination records representing a class with heart disease risk, a probabilistic classifier, specifically Random Forest, is chosen for its effectiveness in handling such datasets. Notably, a study by Surjeet Dalal (2023) utilizes Pearson's correlation in Random Forest, achieving an accuracy of 91.5%. In another research endeavor by Shah et al. (2020), feature selection plays a pivotal role, with 14 selected features used in Decision trees. Despite the limitation of a minimal dataset, this approach yields a modest accuracy score of 83% in its test model.

Researchers, such as Mythili et al. (2013), consistently advocate for a comprehensive approach to data utilization in cardiovascular disease research. This involves leveraging various patient data sources, including demographic information, clinical measurements (e.g., blood pressure, cholesterol levels), medical history, and lifestyle factors. Feature engineering techniques, especially feature selection methods, play a crucial role in enhancing model performance by identifying the most relevant variables for precise predictions. In research by Bhatt, C. M., Patel, P., Ghetia, T. & Mazzeo, P. L. (2023), The XGBoost is a version of gradient boosted decision trees. This algorithm involves creating decision trees in a sequential manner. All the independent variables are allocated weights, which are subsequently used to produce

predictions by the decision tree. If the tree makes a wrong prediction, the importance of the relevant variables is increased and used in the next decision tree. The output of each of these classifiers/predictors is then merged to produce a more robust and accurate model.The XGBoost model achieved 87 % accuracy with the parameters ‘learning_rate’: 0.1, ‘max_depth’: 4, ‘n_estimators’: 100, ‘cross-validation’: 10 folds including 49,000 training and 21,000 testing data instances on 70,000 CVD dataset.

The proposed classification models in this project—SVM, Random Forest, Decision Tree, and XGBoost—are based on the success of these models in different research endeavors, underscoring their potential efficacy in predicting heart disease from patient vital records.

XGBoost

eXtreme Gradient Boosting (XGBoost) is excellent for its crucial classification tasks such as predicting and diagnosing heart disease effectively. It starts with a straightforward model and refines it through cycles of assessment. The model learns by addressing previous errors, each time adding new decision trees that correct them.

Figure 57 goes into detail about the generic pseudocode for the XGBoost algorithm. This pseudocode is high level, obtained from studying the XGBoost algorithm for this problem from various sources such as the XGBoost library from sklearn, YouTube, and Wikipedia. In practice, XGBoost uses more sophisticated techniques for handling various aspects of the training process, such as regularization and handling of missing values.

A crucial element of XGBoost is the learning rate, which dictates how fast the model learns. Too fast, and the model might be overfit to the training data, failing to generalize to new data. Too slow, and the model might not learn enough to be effective. The learning continues until the model no longer shows substantial improvement or meets the iteration limit. The final

model is an aggregation of all the iterative changes, combining the knowledge gained for the most accurate predictions.

Figure 57

Pseudocode for XGBoost Algorithm

Algorithm 1 XGBoost Algorithm

Require: Training set $\{(x_i, y_i)\}_{i=1}^{N}$, a differentiable loss function $L(y, F(x))$, a number of weak learners M and a learning rate α.

Initialize model with a constant value:

$\hat{f}_0(x) = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \theta)$.

for $m = 1$ to M **do**

Compute the 'gradients' and 'hessians':

$\hat{g}_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}_{(m-1)}(x)}$.

$\hat{h}_m(x_i) = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=\hat{f}_{(m-1)}(x)}$.

Fit a base learner (or weak learner, e.g., tree) using the training set $\{x_i, \frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\}_{i=1}^{N}$ by solving the optimization problem below:

$\phi_m = \underset{\phi \in \Phi}{\operatorname{argmin}} \sum_{i=1}^{N} \frac{1}{2} \left[\phi(x_i) - \frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\right]^2$.

$\hat{f}_m(x) = \alpha \phi_m(x)$.

Update the model:

$\hat{f}_m(x) = \hat{f}_{(m-1)}(x) + \hat{f}_m(x)$.

end for

Output $\hat{f}(x) = \hat{f}_M(x) = \sum_{m=0}^{M} \hat{f}_m(x)$.

Decision Tree

Decision Tree is one of the supervised machine learning algorithms designed to solve classification or regression problems. A decision tree is like a flowchart tree structure. The algorithm can work with both categorical and continuous variables. The algorithm operates by creating a tree-like structure, where each node specifies a decision. The final decision of the tree is made by following the path from root node to the leaf node. The UCI "Heart Disease" prediction is a classification problem.

The main dataset is split into smaller subsets based on the main decision point, which is set at the top of the tree called the root node. The outcome of the node branches into internal

nodes which represents a test on the feature. The outcome is called a leaf, which is the classification. Figure 58 shows the workings of splitting the decision tree visually.

Figure 58

Visual representation of Decision Tree algorithm

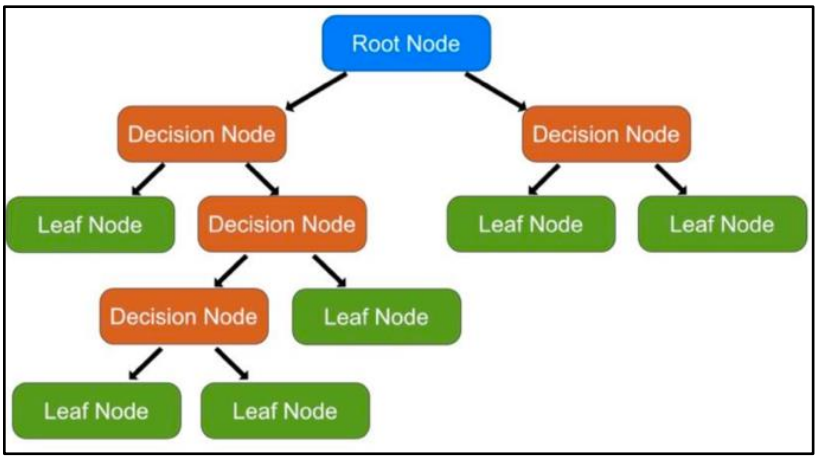


Figure 59 shows the skeleton algorithm of the steps involved in decision tree modeling.

Figure 59

Skeleton algorithm for Decision Tree

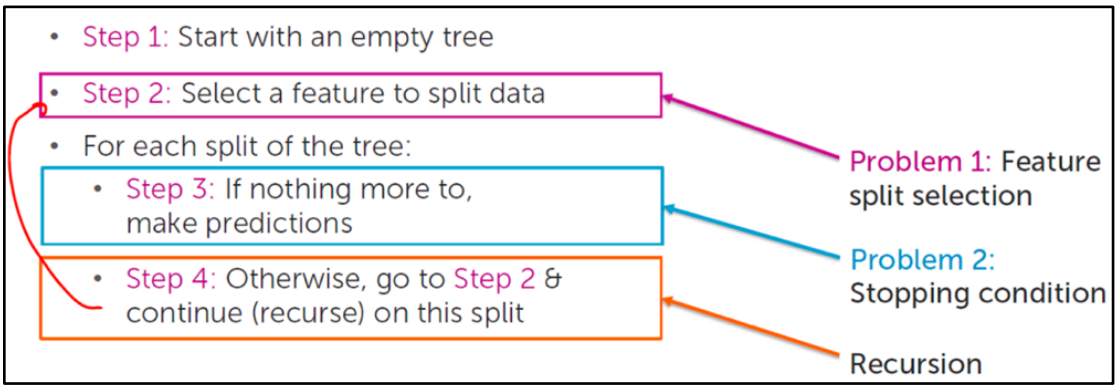
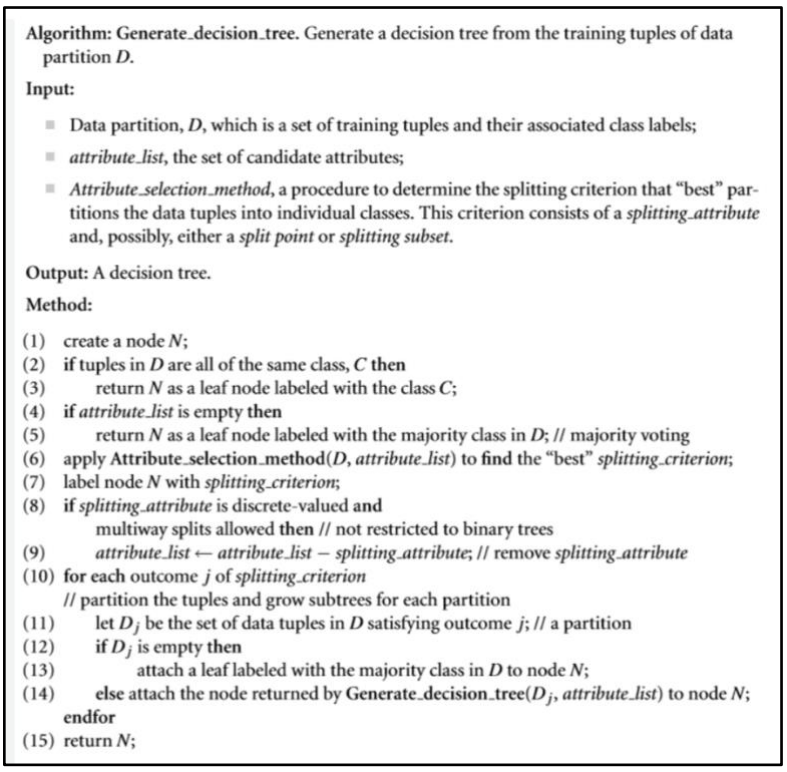


Figure 60 visually presents the pseudocode detailing the decision tree algorithm. The pseudocode lists step-by-step instructions and logical flow that the decision tree follows during its construction and decision-making processes. The algorithm shows insights into how nodes are split, criteria for branching, and the ultimate decision-making logic.

Figure 60

Pseudocode for Decision Tree Algorithm



Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition *D*.

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split point* or *splitting subset*.

Output: A decision tree.

Method:

```
(1)  create a node N;
(2)  if tuples in D are all of the same class, C then
(3)      return N as a leaf node labeled with the class C;
(4)  if attribute_list is empty then
(5)      return N as a leaf node labeled with the majority class in D; // majority voting
(6)  apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
(7)  label node N with splitting_criterion;
(8)  if splitting_attribute is discrete-valued and
         multiway splits allowed then // not restricted to binary trees
(9)      attribute_list ← attribute_list − splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)     let Dj be the set of data tuples in D satisfying outcome j; // a partition
(12)     if Dj is empty then
(13)         attach a leaf labeled with the majority class in D to node N;
(14)     else attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N;
     endfor
(15) return N;
```

The hyperparameters for the decision tree are but not limited to criterion, max depth, min samples split, min samples leaf, max features, splitter, max leaf nodes and min impurity decrease. The current model uses criterion, max depth, min samples split, and min samples leaf. Setting too many hyperparameters may cause the model to overfit. The 'criterion' is used to measure the quality of the split and has options of 'gini' and 'entropy'. The 'max depth' is the maximum depth of the three, controlling the depth to prevent overfitting. The 'min samples split' is the minimum number of samples required to split an internal node, this helps prevent small splits and captures noise. The 'min sample leaf' is the minimum number of samples required to be at a leaf node, this helps prevent creating leaves with very few samples.

Decision Trees uses impurity methods or splitting criteria for the data to make decisions on how to split the data at each node during the tree building process. The 'Entropy' and Gini index are commonly used criteria.

Gini Index or Gini impurity measures the probability that a randomly chosen element is incorrectly classified. The range of the Gini Index is [0,1], where 0 indicates maximum purity. The Gini Index is a linear measure. Lower Gini Index values are preferred. Figure 61 shows the formula for Gini Index.

Figure 61

'Gini index' formula to measure impurity

The Gini index for a set of classes is calculated using the formula:

$$Gini(D) = 1 - \sum_{i=1}^{k} (p_i \cdot \log_2(p_i))$$

where:

- D is the dataset containing instances of k different classes.
- p_i is the proportion of instances of class i in the dataset D.
- $\log_2$ represents the base-2 logarithm.

Entropy is a measure of impurity in the distribution of class labels in a node. The range of entropy is [0, log(c)] where c is the number of classes. Entropy is a logarithmic measure. Entropy uses a logarithmic base 2, since it is easy to scale to a more manageable range. Lower entropy value is preferred.

Figure 62

'Entropy' formula

$$H(X) = -\sum_{i=1}^{n} P(x_i) \cdot \log_2(P(x_i))$$

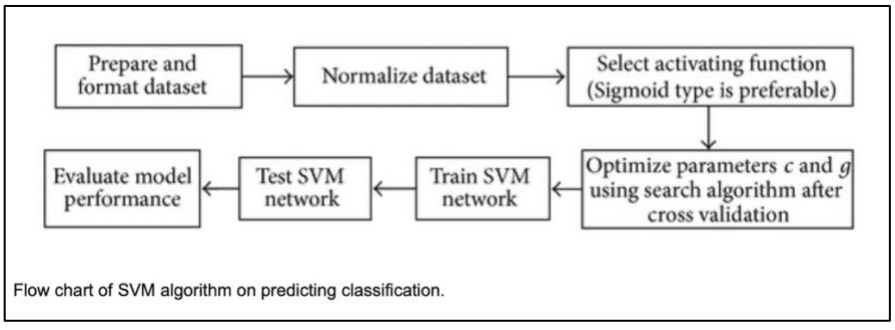
In Figure 62, Entropy(D) is the entropy of the dataset D; p(i) is the probability of a randomly chosen data point in the class; K is the sum runs over all k classes and Log2 is the base 2 logarithmic.

Support Vector Machine (SVM)

Support Vector Machines (SVM) is a supervised learning algorithm designed for classification tasks. The Support Vector Machine (SVM) algorithm involves a series of steps for classification tasks. First, input a labeled dataset with feature vectors and corresponding class labels. Choose an appropriate kernel function, such as a linear or radial basis function, to map the data into a higher-dimensional space. During training, initialize weights and bias and optimize hyperplane parameters to maximize the margin between classes. This typically involves solving a convex optimization problem and adjusting parameters iteratively, often using gradient descent. For prediction, compute the decision function for new data points based on the learned hyperplane parameters and classify them according to the sign of the result. Evaluate the SVM's performance using metrics like accuracy or precision on a separate dataset. Fine-tune hyperparameters, including the regularization parameter and kernel parameters, to optimize the algorithm's effectiveness. These steps collectively outline the process of training and utilizing an SVM for accurate and robust classification.

Figure 63

Support Vector Machine Algorithm Flowchart



Flow chart of SVM algorithm on predicting classification.

SVM is versatile and capable of handling both linear and non-linear boundaries, with the choice of kernel and tuning parameters critical for optimal performance. The algorithm's ability to handle high-dimensional spaces and outliers has found applications in various domains. Thus, it is a formidable tool in classification tasks, particularly in predicting and diagnosing heart disease.

Figure 64

Pseudocode for SVM Algorithm

Training Model for SVM

```
Input: D=[X,Y]; X(array of input with m features), Y(array of class labels)
Y=array(C) // Class label
Output: Find the performance of the system
function train_svm(X,Y, number_of_runs)
initialize:learning_rate=Math.random();
for learning_ rate innumber_of_runs
error=0;
for i in X
if (Y[i] *(X[i]*w))<1 then
update : w=w + learning_rate * ((X[i]*Y[i])*(-2*(1/number_of_runs)*w)
else
update: w=w+learing_rate *(-2*(1/number_of_runs)*w)
end if
end
end
```

The above pseudocode as shown in Figure 64, outlines a basic implementation of the training phase for a Support Vector Machine (SVM) algorithm. The algorithm takes input data D, consisting of feature vectors X and corresponding class labels Y. It initializes the learning rate randomly and performs multiple runs to update the weight vector w.

The inner loop iterates through the data points in X, and for each point, it checks whether the current classification aligns with the true class label. If not, it updates the weight vector using a gradient descent-like approach, aiming to minimize errors and adjust the decision boundary. The learning rate controls the step size of these updates. The algorithm repeats this process for the specified number of runs, aiming to optimize the decision boundary for accurate classification.

Random Forest

Random Forest (RF) is a supervised learning model that combines several decision trees trained with a random subset of features. The prediction is made using majority voting from all the individual trees. According to Kelleher et al. (2015), the decision trees use the Iterative Dichotomizer 3 (ID3) algorithm for induction. The algorithm chooses the best descriptive feature by calculating the Information Gain (IG) of the features in the training dataset and partitions the data based on the target feature. First, the Gini index is calculated for the features. The decrease in Gini impurity after splitting the dataset on an attribute will give the IG for the descriptive feature.

Where *D* is a dataset with a target feature *t, levels (t)* are the set of levels in the domain of the target feature, and *P (t = l)* is the probability of an instance of *D* having the target level *l.* Next, the IG is calculated using Equation 1.

$$IG\ (d, D) = Gini\ (d) - Gini\ (d, D) \quad (1)$$

Where *d* is the target variable, *D* represents the feature to be split on, and *Gini (d, D)* is the Gini impurity index calculated after splitting the dataset on feature *D.* Once the IG is calculated, the ID3 algorithm creates a partition containing the training instances that return the results for each possible target value. A branch is grown from the node for each partition. This process is iterated for each branch till all instances at a partition have the same target level and ends by creating a leaf node labeled with that level. The pseudocode for the ID3 algorithm, as described in the book by Kelleher et al. (2015), is shown in Figure (p. 135 - 149).

Figure 65

ID3 Algorithm

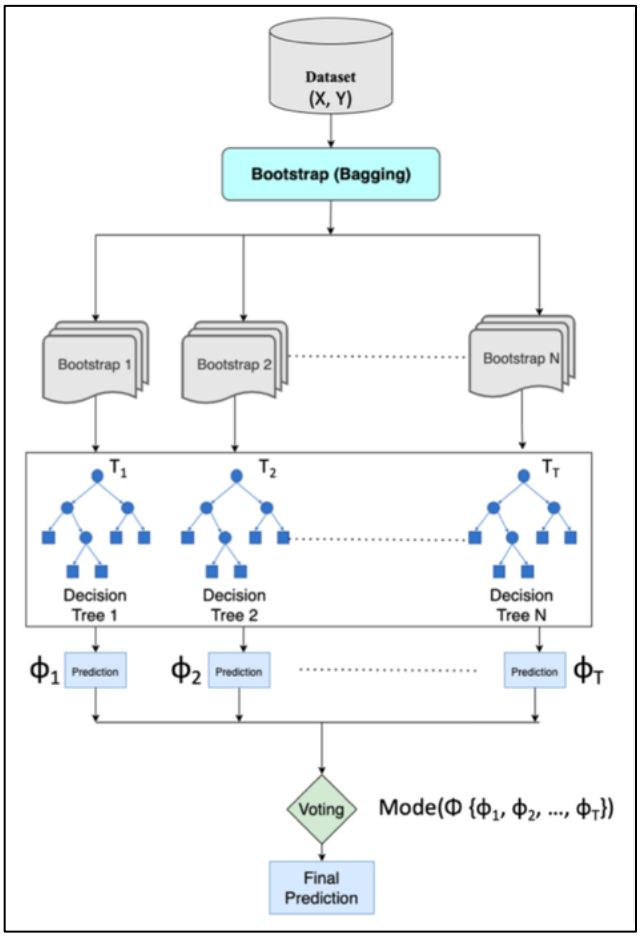
Pseudocode Description of the ID3 Algorithm

Require: set of descriptive features d
Require: set of training instances $\mathcal{D}$
1: if all the instances in $\mathcal{D}$ have the same target level C then
2: return a decision tree consisting of a leaf node with label C
3: else if d is empty then
4: return a decision tree consisting of a leaf node with the label of the majority target level in $\mathcal{D}$
5: else if $\mathcal{D}$ is empty then
6: return a decision tree consisting of a leaf node with the label of the majority target level of the dataset of the immediate parent node
7: else
8: d [*best*] ← arg max $IG(d, \mathcal{D})$ $d \in d$
9: make a new node, $Node_{d[best]}$, and label it with d [*best*]
10: partition $\mathcal{D}$ using d [*best*]
11: remove d [*best*] from d
12: for each partition $\mathcal{D}_i$ of $\mathcal{D}$ do
13: grow a branch from $Node_{d[best]}$ to the decision tree created by rerunning *ID3* with $\mathcal{D} = \mathcal{D}_i$

Furthermore, Kelleher et al. (2015) say RF, when used with bootstrap aggregation (bagging) as shown in Figure 66, trains each model in the ensemble with a random sample of the dataset created using sampling with replacement. Each random sample is the same size as the dataset. Using bootstrap reduces the training time for each tree and enhances the diversity of trees in the ensemble (pp. 165, 166). The predictions from all the individual trees are collected, and the mode of the classes is calculated to find the most predicted class to project as the final prediction. Given $T = \{T_1, T_2, ..., T_T\}$ where T is a set of trees in RF and $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_T\}$ are the predictions of these decision trees, and the RF model is trained from (X, Y) where $X = \{X_1, X_2, ..., X_k\}$ denotes the document corpus, and $Y = \{1, 2, ..., K\}$ denotes the possible classes in documents X. The goal is to calculate the Mode(Φ) to find the final prediction of the random forest classifier. The ensemble technique creates a robust learning model that can improve predictions and reduce the bias and overfitting of data.

Figure 66

Random Forest Ensemble Model with Bootstrap Aggregation



4.2. Model Supports

Environment, Platform, and Tools

The models were trained with optimal resource allocation using high-performance processing. Data is stored in CSV files, and Python serves as the primary platform, excelling across various domains. Python's extensive library ecosystem for machine learning, combined with the versatility of the Jupyter Notebook IDE with Python 3.9, was employed for implementing, training, evaluating, testing, and visualizing metrics for the various models. All the libraries used in this implementation are detailed in Table 14.

Table 14

Libraries Used for Model Development

Purpose	**Libraries Used**	**Description**
Data Manipulation	Pandas, Numpy, DataFrame	They offer efficient data manipulation tools for analysis and data manipulation.
Data Visualization	matplotlib, seaborn	Plotting various visualizations
Data Preprocessing	scikit-learn	To effectively prepare and explore datasets before model training.
Model Building and Training	train_test_split	Split the training data for training and validation
Model Evaluation	from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, roc_auc_score, mean_squared_error,log_loss, precision_score, recall_score, f1_score	A comprehensive set of tools for assessing the performance of models.
Hyperparameter Tuning	scikit-learn, GridSearchCV, RandomizedSearchCV	For hyperparameter tuning, enabling systematic exploration and optimization of model parameters to enhance model performance.
Model Interpretation	SHAP, Lime	Provides insights into the nature of the models by explaining individual predictions and feature importance.

Model Architecture and Dataflow

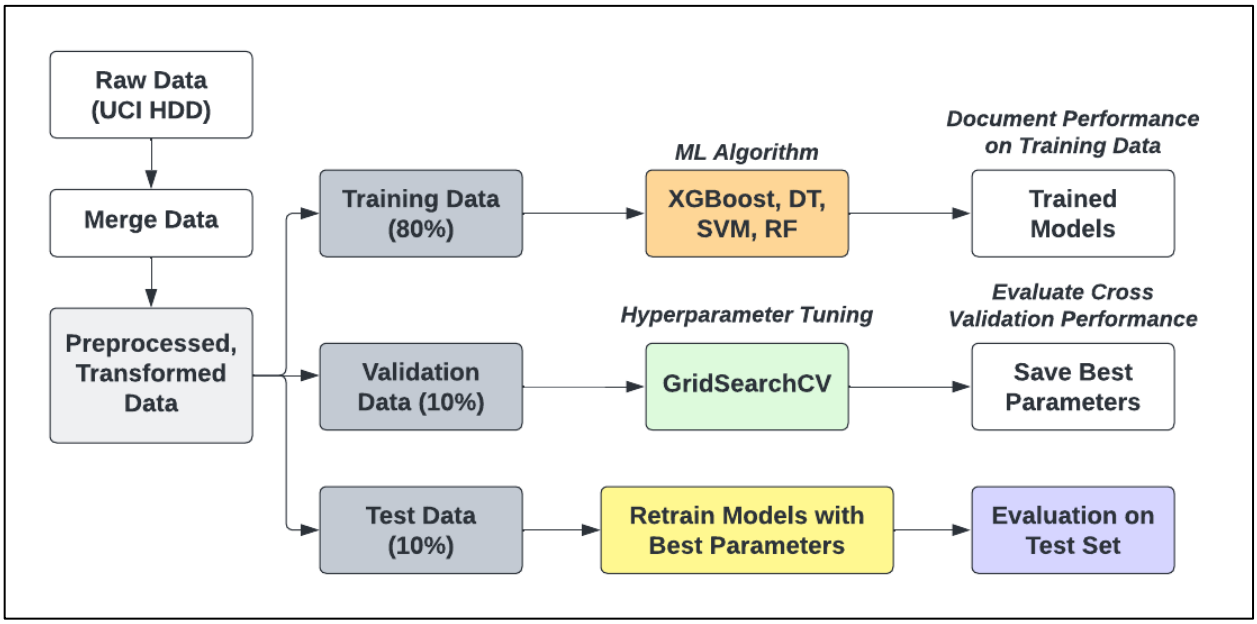
Figure 67 demonstrates a general data flow diagram for the implementation and evaluation of the model. The raw datasets undergo a split into test and train datasets with an 80:20 ratio, facilitated by sklearn.model_selection. To facilitate this split, the data is separately saved as CSV files. Individual models are implemented using sklearn library's sklearn.ensemble.RandomForestClassifier, SVM, XGBoost, and Decision Trees.

Hyperparameter tuning is employed to enhance performance and execution time for machine learning models. Various hyperparameters, including rbf kernel and c, are utilized in the tuning process. Methods like GridSearchSV and RandomizedSearchCV are employed to search for optimal parameters by exploring different instances of each parameter, estimating results, and returning the parameter set with the best outcomes. RandomizedSearchCV is favored for its efficiency, as it randomly selects parameter combinations, unlike GridSearchSV, which considers all possible combinations. Consequently, RandomizedSearchCV is employed for cross-validation. Once the best hyperparameters are determined, the models are optimized using these values.

Dimensionality reduction is accomplished using PCA, and the models undergo evaluation using various metrics from sklearn.metrics, such as roc_curve, accuracy_score, f1-score, etc. Subsequently, the obtained metrics are plotted using SHAP values to enhance model interpretability. General-purpose tasks are facilitated through the utilization of Pandas and NumPy libraries. The scikit-learn library is utilized for the implementation of machine learning models. Specific modules such as sklearn.model_selection for dataset splitting and sklearn.ensemble for RandomForestClassifier, as well as sklearn.svm for SVM, xgboost for XGBoost, and sklearn.tree for Decision Trees are employed.

Figure 67

System Architecture and Model Support Diagram



4.3. Model Comparison and Justification

The Table 15 outlines different models along with their respective strengths and weakness. Decision Trees offer simplicity and easy visualizations, but they may become overly complex, challenging generalization and risking overfitting. Random Forest (RF) and XGBoost are both high predictors of accuracy.

RF, Support Vector Machines (SVM), and XGBoost demonstrate resilience to missing values and outliers. Additionally, these models exhibit robustness against overfitting. However, it's important to note that Random Forest, SVM, and XGBoost can be computationally and/or memory intensive.

Table 15

Table listing various models' comparisons

Model	Advantages	Disadvantages
Decision Tree	• Simple to understand and interpret. The trees can be visualized. • No assumptions about data, the data distribution. • Handles non-linear relationship.	• Decision trees learners can create complex trees which are hard to generalize, this is called as overfitting. • Instability implying small changes in data can lead to different trees. • Decision trees may not capture complex relationships as well as other algorithms.
Random Forests	• Ensemble learning reduces overfitting. • High accuracy for classification and regression problems. • The ensemble nature of Random Forests makes them robust to outliers in the data	• Less interpretable compared to a single decision tree. • Computational complexity during training, leading to longer training times. • Imbalanced classes in classification problems, Random Forests may be biased toward the dominant class.
SVM	• High Performance: Excels in processing complex data, crucial for medical diagnostics. • Automatic Feature Handling: Manages missing values and diverse data types effectively. • Regularization: Includes features to combat overfitting in small datasets. • Customization: Supports specific clinical objectives for targeted results.	• Sensitivity to Parameter Tuning: SVMs' performance is susceptible to the choice of hyperparameters. • Resource Intensive: Tuning numerous hyperparameters requires significant computing resources. • Small Data Challenges: Risk Overfitting. • Limited Generalization: Success on specific

	• Model Interpretability: Offers insights into feature contributions using SHAP values.	datasets might not extend to different datasets.
XGBoost	• Excels in processing complex data, crucial for medical diagnostics • Manages missing values and diverse data types effectively • Regularization to combat overfitting in small datasets • Model Interpretability: Offers insights into feature contributions using SHAP values.	• It can overfit if regularization isn't managed properly • Tuning numerous hyperparameters requires significant compute power • Limited size of dataset raises overfitting risks. • Interpretability-Accuracy Balance: While XGBoost can be made interpretable, the most accurate models may use complex ensembles of trees that are harder to interpret than simpler models

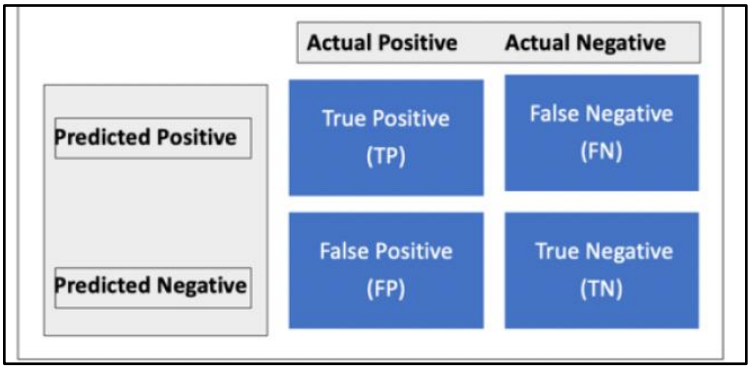
4.4. Model Evaluation Methods

Confusion Matrix

A confusion matrix is a foundational tool for evaluating the performance of a classification model. It provides a comprehensive breakdown of predicted and actual class memberships, with four key components: True Positive (correctly predicted positive instances), True Negative (correctly predicted negative instances), False Positive (incorrectly predicted as positive), and False Negative (incorrectly predicted as negative). From the confusion matrix, several other metrics can be derived, such as accuracy, precision, recall, specificity, and the F1-Score. Figure 68 shows a confusion matrix for a binary classification problem. These metrics offer nuanced insights into different aspects of model performance, making the confusion matrix a versatile and widely used evaluation tool.

Figure 68

Confusion Matrix for Classification



Accuracy Score

Accuracy is a fundamental metric that gauges the overall correctness of predictions made by a model. It is calculated by dividing the sum of correctly predicted instances (both true positives and true negatives) by the total number of instances as shown in Figure 69. While accuracy provides a quick and easy-to-understand measure of performance, it may not be suitable for imbalanced datasets where one class significantly outnumbers the other. In such cases, accuracy can be misleading, and it's crucial to consider additional metrics, such as precision, recall, and the F1-Score, for a more comprehensive evaluation.

Figure 69

Formula for Accuracy Score

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

Precision

Precision is a metric that emphasizes the accuracy of positive predictions made by a model. It assesses the ratio of true positive predictions to the total number of predicted positive instances

as shown in Figure 70. Precision is particularly relevant in scenarios where the cost of false positives is high.

For example, in medical diagnoses, a high precision indicates a low rate of false positives, meaning that when the model predicts a positive result, it is likely to be correct. However, precision should be considered alongside other metrics, as optimizing for high precision might lead to lower recall, and striking the right balance is crucial for a well-rounded evaluation.

Figure 70

Formula for Precision in terms of 'True positives' and 'False Positives'

$$Precision = \frac{True\ positive}{True\ Positive + False\ Positive}$$

Recall (Sensitivity or True Positive Rate)

Recall, also known as sensitivity or the true positive rate, measures a model's ability to correctly identify all relevant instances of a particular class. It is calculated by dividing the number of true positive predictions by the sum of true positives and false negatives as shown in Figure 71. Recall is crucial in scenarios where missing positive instances is more critical than erroneously identifying negative instances. For instance, in spam email detection, high recall ensures that the model captures most of the spam emails, even if it means tolerating some false positives.

Figure 71

Formula for Recall

$$Recall = \frac{True\ Positive}{True\ Positive + True\ Negative}$$

F1-Score

The F1-Score is a harmonic mean of precision and recall as shown in Figure 72, offering a balanced assessment of a model's performance. It considers both false positives and false negatives, providing a single metric that reflects the trade-off between precision and recall. The F1-Score is particularly useful when there is an uneven class distribution or when both false positives and false negatives need to be minimized simultaneously. It becomes especially relevant in situations where precision and recall alone may not provide a clear picture of the model's effectiveness, making it a valuable metric for comprehensive model evaluation.

Figure 72

Formula for F1-Score

$$F = 2 * \frac{Precision * Recall}{Precison + Recall}$$

Receiver Operating Curve (ROC) and Area Under the Curve (AUC)

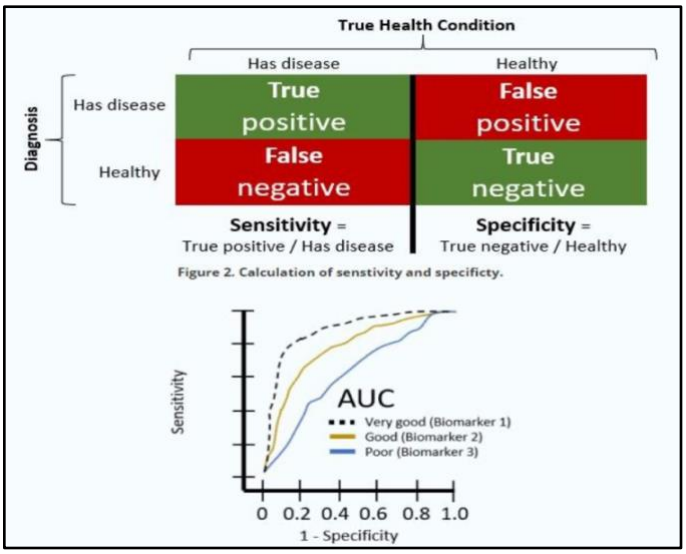
The Receiver Operating Characteristic (ROC) curve is a graphical representation of a model's ability to distinguish between classes at various classification thresholds, plotting the true positive rate against the false positive rate. The Area Under the Curve (AUC) is a numerical measure derived from the ROC curve, offering a comprehensive assessment of a model's discriminatory performance. A perfect model has an AUC of 1, indicating optimal discrimination, while random guessing corresponds to an AUC of 0.5.

The ROC AUC is particularly valuable in binary classification tasks, providing a model with the ability to correctly identify. It is instrumental in scenarios with imbalanced class distributions or when the consequences of false positives and false negatives differ. The ROC AUC complements other evaluation metrics, offering a holistic view of a model's performance in

distinguishing between classes across varying decision thresholds. Figure 73 shows the specifics of the AUC-ROC curve.

Figure 73

AUC-ROC Curve



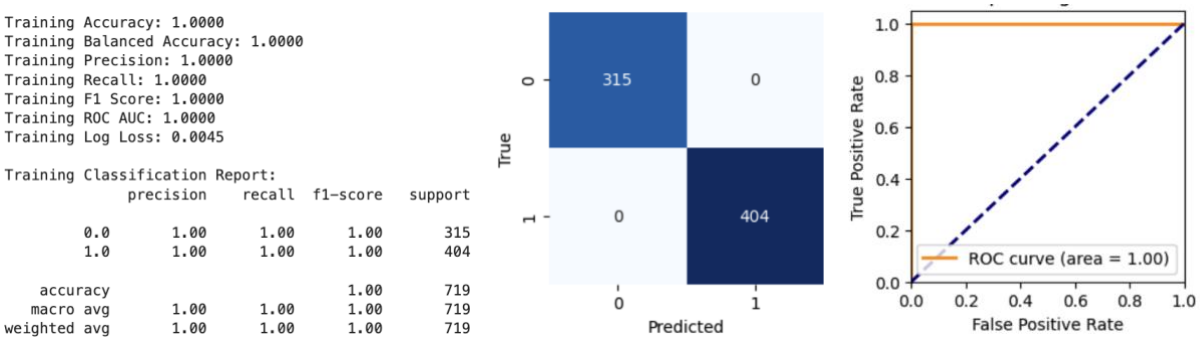
4.5. Model Validation and Evaluation

XGBoost

Baseline Model. The training performance metrics of the Baseline XGBoost model, as depicted in Figure 74 indicate a perfect score, which is highly unusual in real world scenarios. Specifically, the model has achieved a training accuracy, balanced accuracy, precision, recall, and F1 score of 1.00. Furthermore, the training log loss is exceptionally low at 0.0045, and the ROC AUC score is at the maximum possible value of 1.00. The confusion matrix confirms that the model has predicted all 315 instances of one class and 404 instances of the other class with 100% accuracy.

Figure 74

Metrics for XGBoost Baseline Model



Such results suggest that the model has fit the training data to an extent that it has likely memorized it, evidenced by the absence of any misclassifications. While this may seem ideal, it raises significant concerns about the model's ability to generalize to new, unseen data. In the next step, we validate these results against a separate validation dataset to truly gauge the model's predictive power. Without this validation, there is a strong possibility that the model may not perform as well when faced with real-world data, which could lead to misleading conclusions about its effectiveness in diagnosing heart disease.

Hyperparameter Tuned Model. The hyperparameter tuning phase for the model was conducted using GridSearchCV. The search spanned a range of hyperparameters, including max_depth values of 3, 5, and 7, learning_rate options of 0.01, 0.1, and 0.2, n_estimators set to 100, 200, and 300, and subsample ratios of 0.8 and 1. Upon executing the grid search, the best parameter combination was identified as a learning_rate of 0.1, max_depth of 5, n_estimators of 100, and a subsample of 1. This configuration achieved a cross-validation accuracy of 0.9000 on the validation data as shown in Figure 75.

Figure 75

Best Parameters, Validation Accuracy of XGBoost Hyperparameter Tuned Model

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100, 'subsample': 1}
Best Cross Validation Accuracy: 0.9000
```

The results suggest that these hyperparameters provide a strong balance between model complexity and generalization capability. With a relatively modest depth and number of estimators, the model is complex enough to capture underlying patterns in the data without overfitting, as evidenced by the high cross-validation accuracy. Following this, the optimal parameters will be applied to the final model, which is expected to enhance its predictive performance while maintaining robustness against overfitting. This step is crucial in ensuring that the model can generalize well to unseen data, which is critical for diagnosing heart disease.

Test Data. The test results for the XGBoost model, seen in Figure 76, show a promising accuracy of 86.67%. Precision is high at 92.31%, indicating a low rate of false positives, and recall is at 80%, reflecting the model's ability to identify most positive cases. The F1 score, at 85.71%, suggests a balanced model regarding precision and recall.

Figure 76

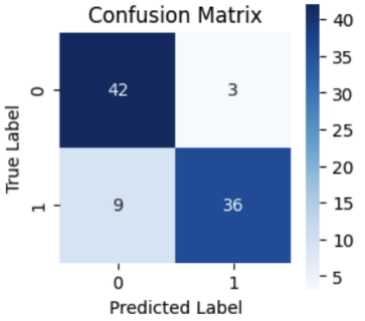
Metrics for XGBoost Hyperparameter Tuned with Test Data

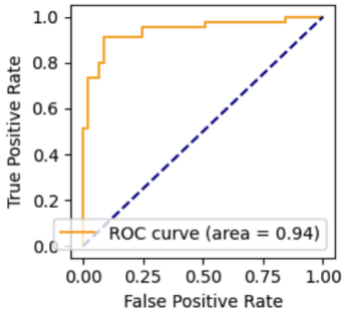
```
Test Accuracy: 0.8667
Test Balanced Accuracy: 0.8667
Test Precision: 0.9231
Test Recall: 0.8000
Test F1 Score: 0.8571
Test ROC AUC: 0.9398
Test Log Loss: 0.3353

Test Classification Report:
              precision    recall  f1-score   support

         0.0       0.82      0.93      0.87        45
         1.0       0.92      0.80      0.86        45

    accuracy                           0.87        90
   macro avg       0.87      0.87      0.87        90
weighted avg       0.87      0.87      0.87        90
```





The ROC AUC score of 93.98% demonstrates the model's strong discriminatory ability. A look at the confusion matrix in Figure 76, confirms the model's effectiveness, with a majority of both classes being correctly classified.

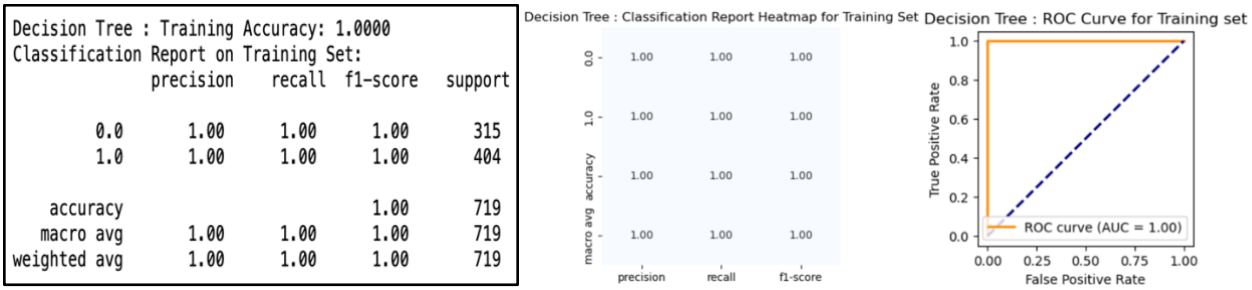
Decision Tree

Baseline Model. The Decision Tree Baseline model has learned the training data perfectly, achieving 100% accuracy as shown in Figure 77. It has correctly classified all instances in both classes, resulting in precision, recall, and F1-score values of 1.00 for both class 0 and class 1. For class 0, there are 315 instances, and for class 1, there are 404 instances. The ROC AUC score is at the maximum possible value of 1.00.

While a perfect training accuracy is a positive sign, it's important to evaluate the model on an independent validation and test set to assess its generalization performance. Overfitting to the training set may occur, and a model that performs well on the training set might not generalize well to new, unseen data.

Figure 77

Metrics for Decision Tree Baseline Model



Hyperparameter Tuned Model. Hyperparameter tuning is the process of finding the best set of hyperparameters for a machine learning model to achieve optimal performance. The model uses Grid search, which is a hyperparameter tuning technique where a predefined set of hyperparameter values is specified, and the model is trained and evaluated for each combination

of these values. The combination that results in the best performance on a validation set is then chosen. For the hyperparameter-tuned model, as shown in Figure 78 the decision tree with the following set of hyperparameters through grid search {'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 4, 'min_samples_split': 2}.

The model achieved a validation accuracy of 77.78% during the hyperparameter tuning process, and the final validation accuracy is 87.78. Precision for class 0.0 is 84%, and recall is 93%, with an F1-score of 88%. Precision for class 1.0 is 93%, and recall is 83%, with an F1-score of 87%. The overall accuracy on the validation set is 88%. In summary, the Decision Tree model with the specified hyperparameters performs well on the validation set, demonstrating good precision, recall, and accuracy for both classes.

Figure 78

Best Parameters, Validation Accuracy of Decision Tree Hyperparameter Tuned Model

```
Decision Tree Best Parameters: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 4, 'min_samples_split':
2}
Decision Tree Best Validation Accuracy: 0.7778
Decision Tree Validation Accuracy: 0.8778

Decision Tree Validation Classification Report:
              precision    recall  f1-score   support

         0.0       0.84      0.93      0.88        44
         1.0       0.93      0.83      0.87        46

    accuracy                           0.88        90
   macro avg       0.88      0.88      0.88        90
weighted avg       0.88      0.88      0.88        90
```

Test data. The Decision Tree model's accuracy is 83.33% of the test data, lower than the baseline accuracy as shown in Figure 79. Precision is at 89%, indicating a low rate of false positives, and recall is at 76%. It appears that the model is reasonably good, with high precision and recall for class 0.0, but slightly lower recall for class 1.0. The F1 score, at 85 and 82% for class 0 and 1, is indicative of a well-performing classifier with a good balance of precision and recall. The weighted average of precision, recall, and F1-score, where each class's score is weighted by its support. The weighted average precision, recall, and F1-score are all around 84%

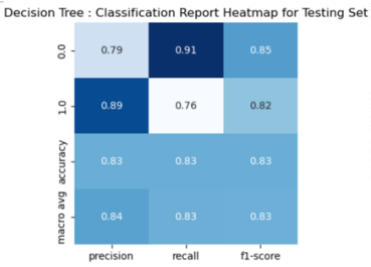
as well. The ROC AUC score of 83% indicates a strong performance of the classification model, suggesting its high ability to discriminate between positive and negative instances.

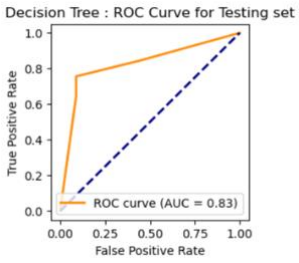
Figure 79

Metrics for Decision Tree with Test data

Decision Tree : Test Accuracy: 0.8333
Decision Tree : Classification Report on Testing Set:

	precision	recall	f1-score	support
0.0	0.79	0.91	0.85	45
1.0	0.89	0.76	0.82	45
accuracy			0.83	90
macro avg	0.84	0.83	0.83	90
weighted avg	0.84	0.83	0.83	90





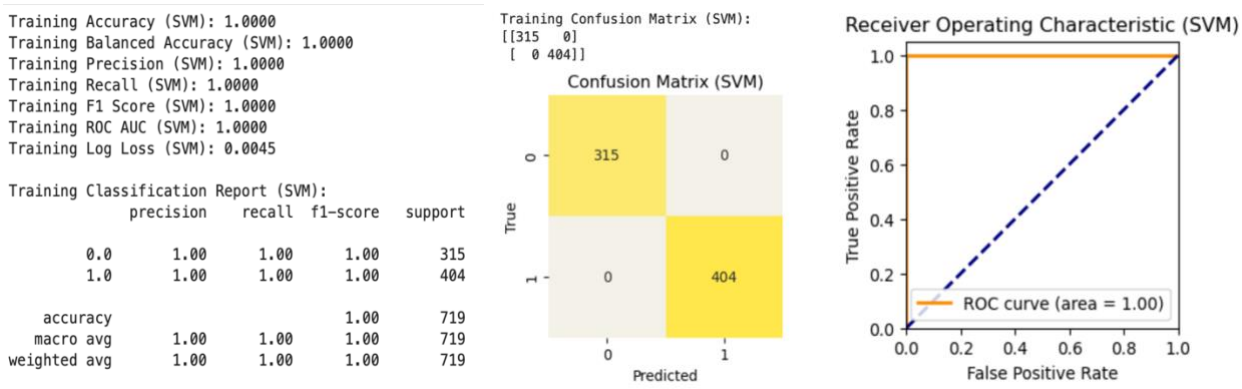
Support Vector Machine (SVM)

Baseline Model. For the Baseline Model, the training evaluation metrics as shown in Figure 80 for the Support Vector Machine (SVM) model present an outstanding performance, indicating that the model has learned the training data with exceptional accuracy. The accuracy, balanced accuracy, precision, recall, F1 score, ROC AUC, and log loss metrics all reveal perfect scores, each registering at 1.0000. The classification report provides further insight, showing flawless precision, recall, and F1 scores for both classes (0 and 1), with a macro and weighted average of 1.00.

The confusion matrix as shown in Figure 80 also solidifies these findings, indicating that all instances in the training set were correctly classified, with no occurrences of false positives or false negatives. In summary, these results strongly suggest that the SVM model has successfully captured intricate patterns within the training data, achieving a level of performance that approaches ideal accuracy across a diverse set of evaluation criteria.

Figure 80

Metrics for SVM Baseline Model



Hyperparameter Tuned Model. For the hyperparameter-tuned model, as shown in figure 81 SVM achieved optimal performance with the following set of hyperparameters obtained through grid search: {'C': 10, 'class_weight': None, 'gamma': 'scale', 'kernel': 'rbf'}. The best cross-validation accuracy during the grid search process was found to be 0.9889. Subsequently, when evaluating the model on a validation set, it demonstrated a high accuracy of 0.9778. These results suggest that the SVM model, with the identified hyperparameters, generalizes well to new, unseen data and maintains a high level of accuracy in classification tasks. The grid search process is crucial for identifying the most suitable hyperparameters, and the obtained values indicate effective tuning for enhanced model performance.

Figure 81

Best Parameters, Validation Accuracy of the SVM Hyperparameter Tuned Model

```
Best Parameters (Grid Search - SVM): {'C': 10, 'class_weight': None, 'gamma': 'scale', 'kernel': 'rbf'}
Best Cross Validation Accuracy (Grid Search - SVM): 0.9889
Validation Accuracy (SVM): 0.9778
```

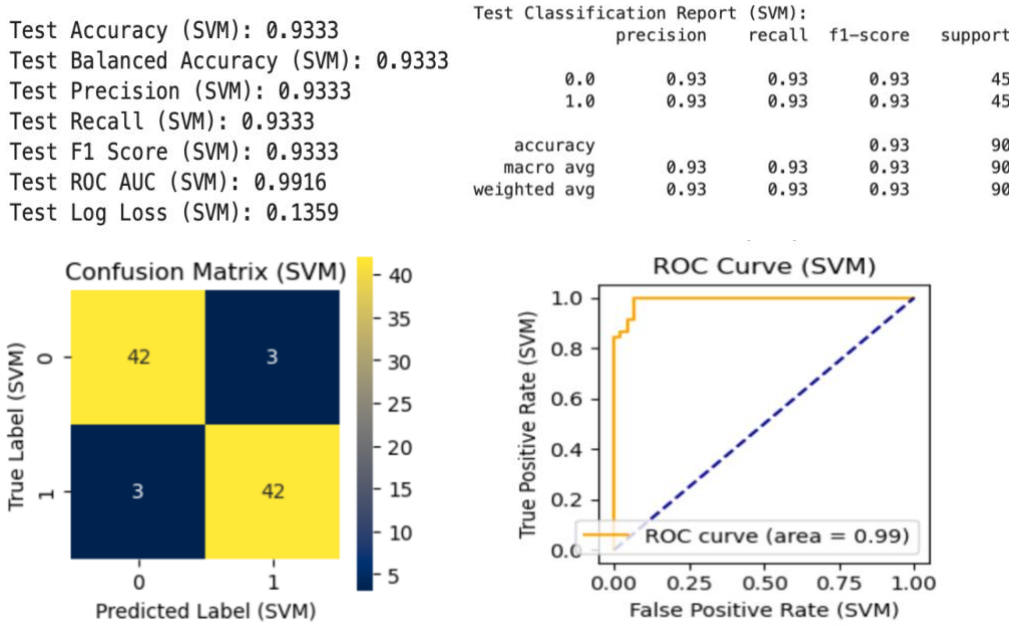
Test Data. The evaluation of the Support Vector Machine (SVM) model on the test dataset as shown in Figure 82 demonstrates a remarkably high level of performance. The overall accuracy, balanced accuracy, precision, recall, and F1 score consistently register at 0.9333, indicative of the

model's proficiency in correctly classifying instances with minimal error. The impressive ROC AUC of 0.9916 underscores the SVM's ability to effectively distinguish between classes. The low log loss of 0.1359 reflects the model's competence in providing precise probabilistic predictions.

The confusion matrix as shown in Figure 82 denoting 42 true negatives, 3 false positives, 3 false negatives, and 42 true positives, offer detailed insights into the model's performance in binary classification. The ROC curve, with an area under the curve of 0.99, supports the SVM model's strong discriminatory power. In summary, these comprehensive evaluation metrics collectively affirm the SVM model's excellence, showcasing its accuracy, effectiveness, and discrimination capability on the given test dataset.

Figure 82

Metrics for SVM Model with Test Data



Random Forest

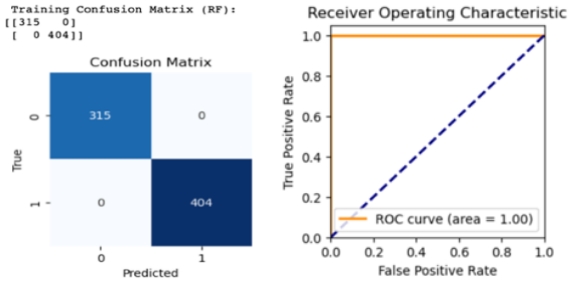
Baseline Model. The training performance metrics of the Baseline Random forest model, as depicted in Figure 83 indicate a perfect score, which is highly unusual in real world scenarios.

Specifically, the model has achieved a training accuracy, balanced accuracy, precision, recall, and F1 score of 1.00. Furthermore, the training entropy is exceptionally low at 0.9889, Gini Impurity is 0.492 and the ROC AUC score is at the maximum possible value of 1.00. The confusion matrix confirms that the model has predicted all 315 instances of one class and 404 instances of the other class with 100% accuracy.

Figure 83

Metrics for Random Forest Baseline Model

```
Train Entropy: 0.9889189412129864
Train MSE: 0.0
Train Gini Impurity: 0.49233888049582064
Train Log Loss (Cross-Entropy): 0.08043613252447657
Training Accuracy (RF): 1.0000
Training Balanced Accuracy (Rf): 1.0000
Training Precision (RF): 1.0000
Training Recall (RF): 1.0000
Training F1 Score (RF): 1.0000
Training ROC AUC (RF): 1.0000
Training Log Loss (RF): 0.0804
```



Hyperparameter Tuned Model. The model uses Grid search, which is a hyperparameter tuning technique where a predefined set of hyperparameter values is specified, and the model is trained and evaluated for each combination of these values. The combination that results in the best performance on a validation set is then chosen. For the hyperparameter-tuned model, as shown in the Figure 84 the random forest with the following set of hyperparameters through grid search {'bootstrap': True, 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}

The model achieved a validation accuracy of 94.33% during the hyperparameter tuning process, and the final validation accuracy is 93.33. Precision for class 0.0 is 97%, and recall is 89%, with an F1-score of 88%. Precision for class 1.0 is 93%, and recall is 83%, with an F1-score of 93%. The overall accuracy on the validation set is 93%. In summary, the Random Forest

model with the specified hyperparameters performs well on the validation set, demonstrating good precision, recall, and accuracy for both classes.

Figure 84

Best Parameters, Validation Accuracy of the Random Forest Hyperparameter Tuned Model

```
Best Parameters:  {'bootstrap': True, 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimator
s': 100}
Best Accuracy: 0.9430
Validation Accuracy (RF): 0.9333

Validation Classification Report (RF):
              precision    recall  f1-score   support

         0.0       0.97      0.89      0.93        44
         1.0       0.90      0.98      0.94        46

    accuracy                           0.93        90
   macro avg       0.94      0.93      0.93        90
weighted avg       0.94      0.93      0.93        90
```

Test Data. The evaluation of the Random Forest (RF) model on the test dataset as shown in Figure 84 demonstrates a remarkably high level of performance. The overall accuracy, balanced accuracy, precision, recall, and F1 score consistently register at 0.94, indicative of the model's proficiency in correctly classifying instances with minimal error. The impressive ROC AUC of 0.97 underscores the RF's ability to effectively distinguish between classes. Furthermore, the training entropy is exceptionally low at 0.9889, Gini Impurity is 0.5 and the ROC AUC score is at the maximum possible value of 0.97. The confusion matrix as shown in Figure 85, denoting 41 true negatives, 3 false positives, 1 false negative, and 44 true positives, offer detailed insights into the model's performance in binary classification. The ROC curve, with an area under the curve of 0.97, supports the RF model's strong discriminatory power. In summary, these comprehensive evaluation metrics collectively affirm the Random Forest model's excellence, showcasing its accuracy, effectiveness, and discrimination capability on the given test dataset.

Figure 85

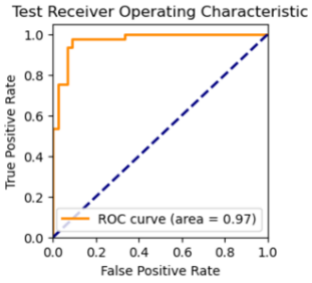
Metrics for Random Forest Test Model

```
Test Entropy: 1.0
Test MSE: 0.05555555555555555
Test Gini Impurity: 0.5
Test Log Loss (Cross-Entropy): 0.1303664399834323
Test Accuracy (RF): 0.9444
Test Balanced Accuracy (RF): 0.9444
Test Precision (RF): 0.9167
Test Recall (RF): 0.9778
Test F1 Score (RF): 0.9462
Test ROC AUC (RF): 0.9719
Test Log Loss (RF): 0.2601

 Test Classification Report (RF):
              precision    recall  f1-score   support

         0.0       0.98      0.91      0.94        45
         1.0       0.92      0.98      0.95        45

    accuracy                           0.94        90
```



Comparison of Model Evaluation Results

The model evaluation results reveal distinct performance characteristics across the four machine learning algorithms that we implemented. Table 16 shows the comparison of all the models in terms of Training, Validation, and Test Accuracy along with their F1-score and ROC-AUC curve.

Table 16

Performance Comparison of all Models

Model	Train Accuracy	Validation Accuracy	Test Accuracy	F1 score	ROC Curve
XGBoost	1.00	0.90	0.8667	0.85	0.94
SVM	1.00	0.9778	0.9333	0.93	0.99
Decision Trees	1.00	0.778	0.83	0.85	0.83
Random Forest	1.00	0.94	0.94	0.94	0.97

XGBoost, while achieving perfect training accuracy, shows a slight decrease in validation and test accuracies, suggesting potential overfitting (Train: 1.00, Validation: 0.90, Test: 0.8667; F1: 0.85). SVM consistently maintains high accuracy across everything (Train: 1.00, Validation: 0.9778, Test: 0.9333; F1: 0.93). Decision Trees exhibit perfect training accuracy but with a drop in validation accuracy, indicating susceptibility to overfitting (Train: 1.00, Validation: 0.778; F1: 0.85). Random Forest, with perfect training accuracy, generalizes well in validation and test sets

(Train: 1.00, Validation: 0.94, Test: 0.94; F1: 0.94) and is supported by a robust ROC curve (0.97).

Conclusion

In our evaluation of machine learning models, Support Vector Machine (SVM) demonstrated excellence in accuracy and classification, demonstrating its effectiveness in handling complex dataset. Similarly, Random Forest showed robust predictive capabilities and resilience to overfitting. XGBoost showed efficient gradient boosting, leveraging its boosting capabilities. Additionally, Decision Trees were chosen for their simplicity and ease of interoperability. Taking a holistic approach, the combined strengths of SVM, Random Forest, XGBoost, and Decision Trees synergistically improved our modeling strategy, ultimately leading to more accurate and reliable diagnostic outcomes for the medical industry.

Future Scope

To enhance the robustness of our models, we aim to collect a more diverse dataset, ensuring a broader representation of scenarios and cases from the medical industry. Increasing the size of our dataset to beyond 899 instances will contribute to better training and generalization. Exploring additional dimensionality reduction techniques will provide an enhanced understanding of the data, while more intensive hyperparameter tuning aims to further optimize the model's performance. To validate the accuracy and reliability of our models, we plan to consult with medical practitioners who can provide valuable insights and validation for the diagnoses generated by our models. This comprehensive approach ensures a well-rounded and validated framework for our machine learning models in the medical domain.

References

A. S. Abdullah & R. R. Rajalaxmi, "*A data mining model for predicting the coronary heart disease using random forest classifier*", *Proc. Int. Conf. Recent Trends Comput. Methods Commun. Controls*, Apr. 2012. https://ieeexplore.ieee.org/abstract/document/8740989

Ahmad, A. A., & Polat, H. (2023). "*Prediction of Heart Disease Based on Machine Learning Using Jellyfish Optimization Algorithm*. Diagnostics". https://doi.org/10.3390/diagnostics13142392

Alizadehsani, R., Roshanzamir, M., Abdar, M., Beykikhoshk, A., Khosravi, A., Panahiazar, M. & Sarrafzadegan, N. (2019). *A database for using machine learning and data mining techniques for coronary artery disease diagnosis*. Scientific data, 6(1), 227. https://www.nature.com/articles/s41597-019-0206-3

Ambale-Venkatesh, B., Yang, X., Wu, C. O., Liu, K., Hundley, W. G., McClelland, R., ... & Lima, J. A. (2017). *Cardiovascular event prediction by machine learning: the multi-ethnic study of atherosclerosis.* Circulation research, 121(9), 1092-1101. https://www.ahajournals.org/doi/full/10.1161/CIRCRESAHA.117.311312

Arnett, D. K., Blumenthal, R. S., Albert, M. A., Buroker, A. B., Goldberger, Z. D., Hahn, E. J., ... & Michos, E. D. (2019). *2019 ACC/AHA guideline on the primary prevention of cardiovascular disease: a report of the American College of Cardiology/American Heart Association Task Force on Clinical Practice Guidelines.* Circulation, 140(11), e596-e646. https://www.ahajournals.org/doi/full/10.1161/CIR.0000000000000677

Bhatt, C. M., Patel, P., Ghetia, T. & Mazzeo, P. L. (2023). *Effective heart disease prediction using machine learning techniques.* Algorithms, *16*(2), 88. https://www.mdpi.com/1999-4893/16/2/88

H. A. Esfahani & M. Ghazanfari, "*Cardiovascular disease detection using a new ensemble classifier", Proc. IEEE 4th Int. Conf. Knowl.-Based Eng. Innov. (KBEI)* Dec. 2017. https://ieeexplore.ieee.org/abstract/document/8324946?casa_token=KQCC2qdfzsUAAAAA:jER1EON2cMzLkop9XOf1G0cJRHKrXTQVSr8q47USG3csJw2DOyuz6-2OSXa3W5Z7hoQoGbi_axez

Khurana, P., Sharma, S., & Goyal, A. (2021, August). *Heart disease diagnosis: Performance evaluation of supervised machine learning and feature selection techniques.* In 2021 8th International Conference on Signal Processing and Integrated Networks (SPIN) (pp. 510-515). IEEE. https://ieeexplore.ieee.org/abstract/document/9565963

Kurt, I., Ture, M., & Kurum, A. T. (2018). *Comparing performances of logistic regression, classification and regression tree, and neural networks for predicting coronary artery disease.* Expert Systems with Applications, 34(1), 366-374. https://doi.org/10.1016/j.eswa.2006.09.004

Mohan, S., Thirumalai, C., & Srivastava, G. (2019). *Effective heart disease prediction using hybrid machine learning techniques.* IEEE access, 7, 81542-81554. https://ieeexplore.ieee.org/abstract/document/8740989/references#references

Mythili, T., Mukherji, D., Padalia, N., & Naidu, A. (2013). *A heart disease prediction model using SVM-decision trees-logistic regression (SDL)*. International Journal of Computer Applications, 68(16).

https://www.academia.edu/56949651/A_Heart_Disease_Prediction_Model_using_SVM_Decision_Trees_Logistic_Regression_SDL_

Patra, R., & Khuntia, B. (2019, February). Predictive analysis of rapid spread of heart disease with data mining. In 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT) (pp. 1-4). IEEE. https://ieeexplore.ieee.org/abstract/document/8869194/

Shah, D., Patel, S., & Bharti, S. K. (2020). Heart disease prediction using machine learning techniques. SN Computer Science, 1, 1-6. https://link.springer.com/article/10.1007/s42979-020-00365-y

Surjeet Dalal, Pallavi Goel, Edeh Michael Onyema, Adnan Alharbi, Amena Mahmoud, Majed A. Algarni & Halifa Awal https://www.hindawi.com/journals/cin/2023/9418666/

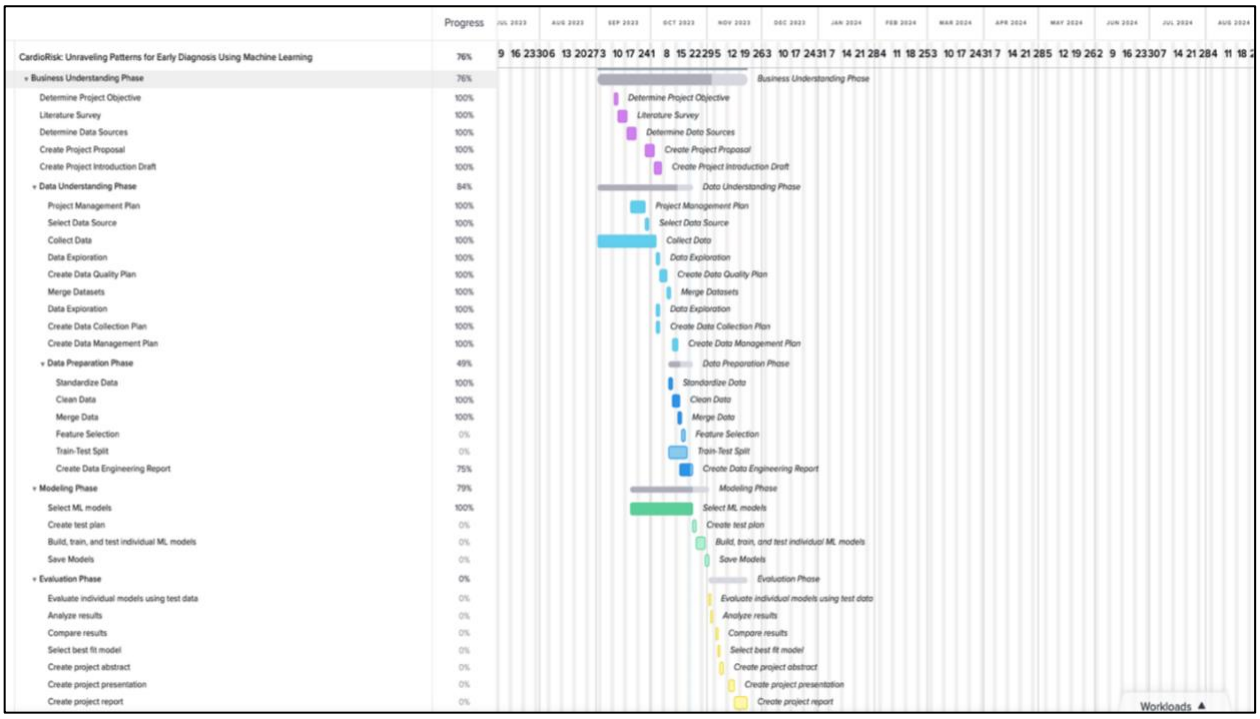
Weka Toolkit https://www.cs.waikato.ac.nz/ml/weka/

Wirth, R., & Hipp, J. (2000). *CRISP-DM: Towards a standard process model for data mining.* In Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining (pp. 29-39). Springer-Verlag. http://www.cs.unibo.it/~danilo.montesi/CBD/Beatriz/10.1.1.198.5133.pdf

World Health Organization. (2021). Cardiovascular diseases (CVDs). https://www.who.int/health-topics/cardiovascular-diseases#tab=tab_1

Yusuf, S., Reddy, S., Ôunpuu, S., & Anand, S. (2001). *Global burden of cardiovascular diseases: Part II: variations in cardiovascular disease by specific ethnic groups and geographic regions and prevention strategies.* Circulation, 104(23), 2855-2864. https://www.ahajournals.org/doi/full/10.1161/hc4701.099488

Appendix A



Appendix A. *Complete Gantt Chart: CardioRisk*

Appendix B

Code for Data Preprocessing and Data Transformation

This appendix consists of some key snapshots of sample code from the preprocessing stage of the project. Figure B1 shows the code to check for unique values. Figure B2 shows the code for handling missing values with mean/mode imputation methods. Figure B3 is a function to find all the special code (-9). Figure B4 is code for checking missing values. Figure B5 is code for dimensionality reduction.

```
for i in dataCVD.columns:
    print(i, len(dataCVD[i].unique()))
```

Figure B1. Check for unique values within DataFrame

```
def replace_missing_with_mode(df, lower_threshold, columns_to_update_mode):
    """
    The function takes in DataFrame i.e. dataCVD
    Replaces based on threshold values which can be controlled
    List of features/columns to update as mode
    """
    for column in columns_to_update_mode:
        missing_percentage = df[column].isnull().sum() / len(df) * 100
        # Make sure to update only those records which fall in the specified condition
        if lower_threshold < missing_percentage:
            # Get the mode value
            mode_value = df[column].mode().values[0]
            df[column].fillna(mode_value, inplace=True)

def replace_missing_with_mean(df, lower_threshold, columns_to_update_mean):
    """
    The function takes in DataFrame i.e. dataCVD
    Replaces based on threshold values which can be controlled
    List of features/columns to update as mean
    """
    for column in columns_to_update_mean:
        missing_percentage = df[column].isnull().sum() / len(df) * 100
        # Make sure to update only those records which fall in the specified condition
        if lower_threshold < missing_percentage:
            # Get the mean value
            mean_value = df[column].mean()
            df[column].fillna(mean_value, inplace=True)
```

Figure B2. Code for Handling Missing Values with Mean/Mode Imputation Methods

```python
def categorize_missing_values(dataframe, missing_value):
    """
    The function takes in the DataFrame i.e. dataCVD
    missing_value is -9
    """
    feature_missing_data = []

    for column in dataframe.columns:
        count = (dataframe[column] == missing_value).sum()
        total_values = dataframe[column].size
        missing_percentage = (count / total_values) * 100

        if missing_percentage >= 1 and missing_percentage <= 10:
            category = "1-10%"
        elif missing_percentage > 10 and missing_percentage <= 20:
            category = "11-20%"
        elif missing_percentage > 20 and missing_percentage <= 50:
            category = "21-50%"
        elif missing_percentage > 50:
            category = "51-100%"
        else:
            category = "No Missing Values"

        feature_missing_data.append((column, missing_percentage, category))

    # Sort the data by category
    sorted_data = sorted(feature_missing_data, key=lambda x: x[2])

    return sorted_data
```

Figure B3. Function to find all the special code (-9)

```python
# Calculate the total number of missing values for each feature
missing_values = dataCVD.isnull().sum()

# Calculate the total number of values for each feature
total_values = dataCVD.shape[0]

# Calculate the percentage of missing values for each feature
percentage_missing = (missing_values / total_values) * 100

# Create a DataFrame to display the results
missing_data = pd.DataFrame({'Feature': missing_values.index, 'Missing Percentage': percentage_missing})
missing_data_values = missing_data[missing_data['Missing Percentage'] > 0]

print(missing_data_values)
```

Figure B4. Code for Checking Missing Values

```python
# Normalize the features for each dataset separately
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Apply PCA with 40 components to the standardized training data
pca = PCA(n_components=40)
X_train_pca = pca.fit_transform(X_train_scaled)

# Transform the validation and testing sets with the same PCA model
X_val_pca = pca.transform(X_val_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Convert the PCA-transformed data back to DataFrames
X_train_pca_df = pd.DataFrame(X_train_pca, columns=[i+1 for i in range(40)])
X_val_pca_df = pd.DataFrame(X_val_pca, columns=[i+1 for i in range(40)])
X_test_pca_df = pd.DataFrame(X_test_pca, columns=[i+1 for i in range(40)])
```

Figure B5. Code for Dimensionality Reduction

Appendix C

Code for Implementing XGBoost

This appendix has sample code from the modeling stage of the XGBoost Algorithm for cardiovascular disease prediction. The code for Model Validation is shown in Figure C1 and Figure C2 has the code for Model Evaluation with best hyperparameters on test data.

```python
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300],
    'subsample': [0.8, 1]
}

# Initialize GridSearchCV with the XGBoost classifier
xgb_grid_search = GridSearchCV(
    estimator=xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss'),
    param_grid=param_grid,
    scoring='accuracy',
    cv=10
)

# Perform the grid search on the validation data
xgb_grid_search.fit(X_val, y_val)

# Output the best parameters and the highest cross validation accuracy
best_params = xgb_grid_search.best_params_
best_score = xgb_grid_search.best_score_
print(f"Best Parameters: {best_params}")
print(f"Best Cross Validation Accuracy: {best_score:.4f}")
```

Figure C1. Code for XGBoost Model Validation

```python
# Use the best estimator from the grid search to make predictions on the test data
y_test_pred = xgb_grid_search.best_estimator_.predict(X_test)
y_test_proba = xgb_grid_search.best_estimator_.predict_proba(X_test)[:, 1]

# Compute various evaluation metrics for the test data
test_accuracy = accuracy_score(y_test, y_test_pred)
test_balanced_accuracy = balanced_accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
test_roc_auc = roc_auc_score(y_test, y_test_proba)
test_log_loss = log_loss(y_test, y_test_proba)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)
test_classification_report = classification_report(y_test, y_test_pred)

# Display the evaluation metrics for the test set
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Balanced Accuracy: {test_balanced_accuracy:.4f}")
print(f"Test Precision: {test_precision:.4f}")
print(f"Test Recall: {test_recall:.4f}")
print(f"Test F1 Score: {test_f1:.4f}")
print(f"Test ROC AUC: {test_roc_auc:.4f}")
print(f"Test Log Loss: {test_log_loss:.4f}")
print("\nTest Classification Report:")
print(test_classification_report)

# Plot the confusion matrix
print("\nTest Confusion Matrix:")
plt.figure(figsize=(3, 3))
sns.heatmap(test_conf_matrix, annot=True, fmt='d', cmap='Blues', square=True)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Figure C2. Code for XGBoost Model Evaluation

Appendix D

Code for Implementing Decision Tree

This appendix has sample code from the modeling stage of the Decision Tree Algorithm for cardiovascular disease prediction. The code for Model Validation is shown in Figure D1 and Figure D2 has the code for model evaluation with best hyperparameters on test data.

```
# Define the parameter grid for grid search
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 3, 5, 7],
    'min_samples_split': [2, 4, 6, 8],
    'min_samples_leaf': [1, 2, 4]
}

# Create the GridSearchCV object
grid_search = GridSearchCV(decision_tree_model, param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the data (using training set)
grid_search.fit(X_val, y_val)

# Print the best parameters and corresponding accuracy
print("Decision Tree Best Parameters:", grid_search.best_params_)
print(f"Decision Tree Best Validation Accuracy: {grid_search.best_score_:.4f}")

# Get the best model
best_dt_model = grid_search.best_estimator_

# Now you can use this best model for predictions on your validation or test set
y_val_pred_dt = best_dt_model.predict(X_val)

# You can also calculate metrics on the validation set using the same approach as you did for the training set
val_accuracy_dt = accuracy_score(y_val, y_val_pred_dt)
val_classification_report_dt = classification_report(y_val, y_val_pred_dt)

# Print validation metrics
print(f"Decision Tree Validation Accuracy: {val_accuracy_dt:.4f}")
print("\nDecision Tree Validation Classification Report:")
print(val_classification_report_dt)
```

Figure D1. Code for Decision Tree Model Validation

```
# Using the best estimator from the grid search to make predictions on the test data
y_test_pred = grid_search.best_estimator_.predict(X_test)
y_test_proba = grid_search.best_estimator_.predict_proba(X_test)[:, 1]

# Calculate metrics
test_entropy = calculate_entropy(y_test)
test_mse = calculate_mse(y_test, y_test_pred)
test_gini_impurity = calculate_gini_impurity(y_test)


# Print the metrics
print("Test Entropy:", test_entropy)
print("Test MSE:", test_mse)
print("Test Gini Impurity:", test_gini_impurity)

# Evaluate the model on the test set
accuracy_test = accuracy_score(y_test, y_test_pred)
print(f"Decision Tree : Test Accuracy: {accuracy_test:.4f}")

# Display classification report on the test set
print("Decision Tree : Classification Report on Testing Set:\n", classification_report(y_test, y_test_pred))
```

Figure D2. Code for Decision Tree Model Evaluation

Appendix E

Code for Implementing Support Vector Machines

This appendix consists of some snapshots for implementing SVM. Figure E1 to Figure E2 shows the code for GridSearchCV and Hyperparameter Tuning for SVM.

```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score  # Import accuracy_score

# Scaling the features for SVM
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Defining an extended hyperparameter grid for SVM
param_grid_svm = {
    'C': [0.1, 1, 10],  # Include larger values for C
    'kernel': ['rbf'],
    'gamma': [0.1, 1, 10, 'scale', 'auto'],
    'class_weight': [None, 'balanced']  # Considering class weights for imbalanced datasets
}

# Initializing the GridSearchCV with the SVM classifier
svm_grid_search = GridSearchCV(
    estimator=SVC(probability=True),
    param_grid=param_grid_svm,
    scoring='accuracy',
    cv=10
)

# Performing the grid search on the scaled training data
svm_grid_search.fit(X_train_scaled, y_train)

# Displaying the the best parameters and the highest cross-validation accuracy
best_params_grid_svm = svm_grid_search.best_params_
best_score_grid_svm = svm_grid_search.best_score_
print(f"Best Parameters (Grid Search - SVM): {best_params_grid_svm}")
print(f"Best Cross Validation Accuracy (Grid Search - SVM): {best_score_grid_svm:.4f}")
```

Figure E1. Code for Model Training with GridSearchCV

```python
# Initializing the GridSearchCV with the SVM classifier
svm_grid_search = GridSearchCV(
    estimator=SVC(probability=True),
    param_grid=param_grid_svm,
    scoring='accuracy',
    cv=10
)

# Performing the grid search on the scaled training data
svm_grid_search.fit(X_train_scaled, y_train)

# Displaying the the best parameters and the highest cross-validation accuracy
best_params_grid_svm = svm_grid_search.best_params_
best_score_grid_svm = svm_grid_search.best_score_
print(f"Best Parameters (Grid Search - SVM): {best_params_grid_svm}")
print(f"Best Cross Validation Accuracy (Grid Search - SVM): {best_score_grid_svm:.4f}")

# Evaluating the model on the validation set
y_val_pred_svm = svm_grid_search.predict(X_val_scaled)
val_accuracy_svm = accuracy_score(y_val, y_val_pred_svm)
print(f"Validation Accuracy (SVM): {val_accuracy_svm:.4f}")
```

Figure E2. Code for Model Validation with Hyperparameter Tuned Model

Appendix F

Code for Implementing Random Forest

This appendix consists of key snapshots of sample code from training the random forest classifier of the project. Figure F1 shows the code for the baseline model. Figure F2 shows the code for model validation and hyperparameter tuning of model using GridSearchCV and Figure F3 has the code for evaluation of the model with the best hyperparameters on test data.

```
# Entropy calculation function
def calculate_entropy(y):
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy

# Mean Squared Error (MSE) calculation function
def calculate_mse(y_true, y_pred):
    return mean_squared_error(y_true, y_pred)

# Gini impurity calculation function
def calculate_gini_impurity(y):
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    gini_impurity = 1 - np.sum(probabilities ** 2)
    return gini_impurity

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model on the training data
rf_classifier.fit(X_train, y_train)

# Prediction on training data
y_train_pred_rf = rf_classifier.predict(X_train)
y_train_proba_rf = rf_classifier.predict_proba(X_train)[:, 1]

# Calculate metrics
train_entropy_rf = calculate_entropy(y_train)
train_mse_rf = calculate_mse(y_train, y_train_pred_rf)
train_gini_impurity_rf = calculate_gini_impurity(y_train)

# Additional metric for binary classification - log loss (cross-entropy)
train_log_loss_rf = log_loss(y_train, rf_classifier.predict_proba(X_train)[:, 1])

# Print the metrics
print("Train Entropy:", train_entropy_rf)
print("Train MSE:", train_mse_rf)
print("Train Gini Impurity:", train_gini_impurity_rf)
print("Train Log Loss (Cross-Entropy):", train_log_loss_rf)

# Calculate other metrics
train_accuracy_rf = accuracy_score(y_train, y_train_pred_rf)

train_balanced_accuracy_rf = balanced_accuracy_score(y_train, y_train_pred_rf)

train_precision_rf = precision_score(y_train, y_train_pred_rf)

train_recall_rf = recall_score(y_train, y_train_pred_rf)

train_f1_rf = f1_score(y_train, y_train_pred_rf)

train_roc_auc_rf = roc_auc_score(y_train, y_train_proba_rf)

train_log_loss_rf = log_loss(y_train, y_train_proba_rf)

train_conf_matrix_rf = confusion_matrix(y_train, y_train_pred_rf)

train_classification_report_rf = classification_report(y_train, y_train_pred_rf)

# Printing all the metrics
print(f"Training Accuracy (RF): {train_accuracy_rf:.4f}")
print(f"Training Balanced Accuracy (Rf): {train_balanced_accuracy_rf:.4f}")
print(f"Training Precision (RF): {train_precision_rf:.4f}")
print(f"Training Recall (RF): {train_recall_rf:.4f}")
print(f"Training F1 Score (RF): {train_f1_rf:.4f}")
print(f"Training ROC AUC (RF): {train_roc_auc_rf:.4f}")
print(f"Training Log Loss (RF): {train_log_loss_rf:.4f}")
print("\n Training Classification Report (RF):")
print(train_classification_report_rf)
```

Figure F1. Code for Random Forest Baseline Model

```python
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid,
                           scoring='accuracy', cv=3, n_jobs=-1, verbose=2)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and the corresponding accuracy
print("Best Parameters: ", grid_search.best_params_)
print("Best Accuracy: {:.4f}".format(grid_search.best_score_))

# Get the best model
best_rf_model = grid_search.best_estimator_

# Now you can use this best model for predictions on your validation or test set
y_val_pred_rf = best_rf_model.predict(X_val)

# You can also calculate metrics on the validation set using the same approach as you did for the training set
val_accuracy_rf = accuracy_score(y_val, y_val_pred_rf)
val_classification_report_rf = classification_report(y_val, y_val_pred_rf)

# Print validation metrics
print(f"Validation Accuracy (RF): {val_accuracy_rf:.4f}")
print("\nValidation Classification Report (RF):")
print(val_classification_report_rf)
```

Figure F2. Code for Random Forest Model Validation

```python
# Prediction on test data
y_test_pred_rf = best_rf_model.predict(X_test)
y_test_proba_rf = best_rf_model.predict_proba(X_test)[:, 1]

# Entropy calculation function
def calculate_entropy(y):
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy

# Mean Squared Error (MSE) calculation function
def calculate_mse(y_true, y_pred):
    return mean_squared_error(y_true, y_pred)

# Gini impurity calculation function
def calculate_gini_impurity(y):
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    gini_impurity = 1 - np.sum(probabilities ** 2)
    return gini_impurity

# Calculate metrics
test_entropy_rf = calculate_entropy(y_test)
test_mse_rf = calculate_mse(y_test, y_test_pred_rf)
test_gini_impurity_rf = calculate_gini_impurity(y_test)

# Additional metric for binary classification - log loss (cross-entropy)
test_log_loss_rf = log_loss(y_test, y_test_proba_rf)

# Print the metrics
print("Test Entropy:", test_entropy_rf)
print("Test MSE:", test_mse_rf)
print("Test Gini Impurity:", test_gini_impurity_rf)
print("Test Log Loss (Cross-Entropy):", test_log_loss_rf)

# Calculate other metrics for test data
test_accuracy_rf = accuracy_score(y_test, y_test_pred_rf)

test_balanced_accuracy_rf = balanced_accuracy_score(y_test, y_test_pred_rf)

test_precision_rf = precision_score(y_test, y_test_pred_rf)

test_recall_rf = recall_score(y_test, y_test_pred_rf)

test_f1_rf = f1_score(y_test, y_test_pred_rf)

test_roc_auc_rf = roc_auc_score(y_test, y_test_proba_rf)

test_log_loss_rf = log_loss(y_test, y_test_proba_rf)

test_conf_matrix_rf = confusion_matrix(y_test, y_test_pred_rf)

test_classification_report_rf = classification_report(y_test, y_test_pred_rf)
```

Figure F3. Code for Random Forest Model Evaluation on Test Data