LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CU6051NI Artificial Intelligence

Choose an item.**Individual Coursework**

Submission: Final Submission

Academic Semester: Autumn Semester 2025

Credit: 15 credit semester long module

Student Name: Abhinav Shakya

London Met ID: 23050332

College ID: Np01cp4a230199

Assignment Due Date: 21/01/2026.

Assignment Submission Date: 21/01/2026

Submitted To: Binod Bhattrai

GitHub Link	*https://github.com/ShakyaAbi/AbhinavShakya-courcework*

Table of Contents

Table of Figures

1 Introduction

Machine learning and the integration of AI in healthcare is revolutionizing how patients diagnosed and cared for(Foresee medical, n.d.) As one of the major health problem Heart and cardiovascular disease are one of the major causes of deaths worldwide, with a estimated 19.8 million lives lost to CVD yearly accounting for almost 32% of all global deaths (World Health Organization, 2025). Detecting and diagnosing heart conditions early on are vital for carrying out effective treatments and preventing fatal events from occurring. In recent years, machine learning has become a potent part of healthcare, assisting doctors in making data-centric forecasts and predictions of patients who are more likely to have heart conditions and other diseases as well. Supervised classification techniques, for instance, can be applied to patient data to predict the presence or absence of many diseases if sufficient data is provided

Supervised learning is a machine learning method where the algorithm is trained using labeled data. In other words, the training dataset has the right output (label) for every input example. While training the model, it learns the correspondence of the input features and those labels so that it can give correct predictions on new, unseen data. It is extensively used in classification and regression problems where the main objective is to forecast a certain outcome or category. Some common examples are spam filtering, price forecasting, and credit scoring. The essential factor is that the model receives “ground truth” labels during the training.

On the other hand, unsupervised learning is an approach that operates on unlabeled data. Thus, it is the model's task is to find patterns, similarities, clusters, or find structures of the data on its own. It is commonly applied to clustering, dimensionality reduction, segmentation, and anomaly detection. Rather than predicting an outcome, unsupervised learning reveals the structure or relationships in your data when you have no idea what the answers are. Customer segmentation and dataset grouping by similar behaviors are among the examples.

1.1 Explanation of the problem area

Heart disease includes many conditions that affect the heart and blood vessels, such as coronary artery disease, arrhythmias, and heart failure. In clinical practice, we use risk scores (like the Framingham Risk Score) and diagnostic tests (e.g., ECG, stress tests, angiography) to assess patients to find out if they have heart disease. These methods have limitations in accuracy, and they often require specialized procedures. Using artificial intelligence and machine learning offers a new, data-driven way to augment these traditional methods. By training algorithms on historical patient data, we can help computers learn to detect subtle patterns or combinations of risk factors that correlate with the presence of heart disease. Previous advancements in our computational power and the availability of large datasets have spurred research into the use of machine learning for cardiac risk prediction.

In this context, the UCI Heart Disease dataset serves as a valuable resource it contains patient records with various clinical attributes, enabling the development of predictive models. The background of this study lies in bridging the gap between clinical cardiology and data science: leveraging historical data to create models that can predict whether a patient is likely to have heart disease. This not only could improve diagnostic accuracy but also help in preventive cardiology by flagging high-risk individuals early on making it easier for treatment. The rationale for our project is built on this intersection of healthcare and ML, aiming to contribute to more efficient and accurate heart disease screening tools.

2 Background

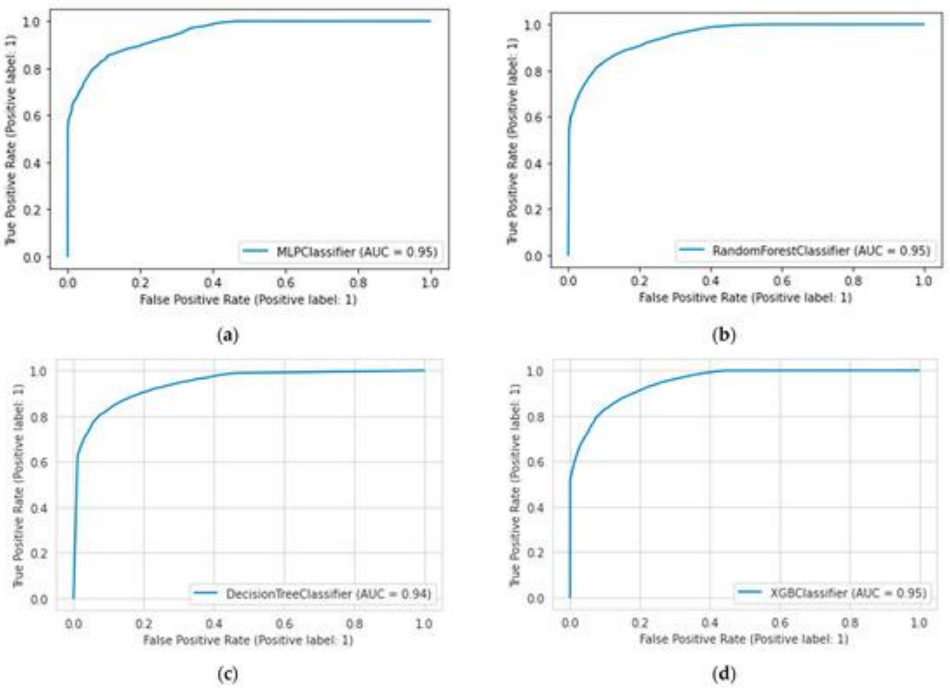
2.1 Review on Existing Systems on the Problem Domain

In the heart disease prediction problem domain, most "existing systems" follow a similar pipeline: clinical dataset collection (often UCI-style heart disease datasets), preprocessing and feature handling, training multiple classifiers, and evaluating performance using accuracy and related metrics. A concise review of closely related studies is presented below.

2.1.1 Effective Heart Disease Prediction Using Machine Learning Techniques

Authors: Chintan M. Bhatt, Parth Patel, Tarang Ghetia, Pier Luigi Mazzeo

In this paper, the use of supervised machine learning for the prediction of heart diseases is reviewed along with the common processing steps like data pretreatment, feature selection, and classifier comparison. One of the most important findings of their review is that the instance-based methods, like KNN, can yield very good results on the Cleveland heart disease dataset (303 records), suggesting that KNN can be used as a standard reference point for structured clinical tabular data in experiments conducted by them. Their experiments found that the K-Nearest Neighbors (KNN) algorithm achieved the highest accuracy at 90.8%, illustrating the effectiveness of instance-based learning for this prediction task. (Bhatt, 2023)



2.1.2 Implementation of Machine Learning Model to Predict Heart Failure Disease

Author: Fahd Saleh Alotaibi

Paper: Implementation of Machine Learning Model to Predict Heart Failure Disease (IJACSA)

This research paper implemented various ML algorithms like Decision Tree, Logistic Regression, Random Forest, Naïve Bayes and Support Vector Machine to predict heart disease using Cleveland Clinic Foundation data. Using 10-fold cross-validation, the study found that a Decision Tree classifier performed best with 93.19% accuracy, closely followed by SVM at 92.30%. This result underlined the strong performance of tree-based models for structured medical data. (Alotaibi. , 2019)

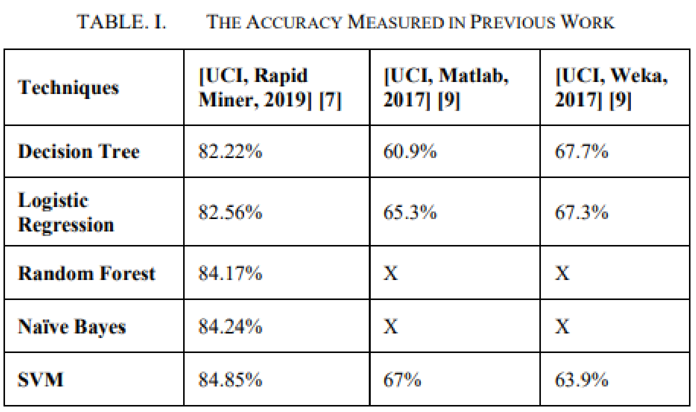


TABLE. I. THE ACCURACY MEASURED IN PREVIOUS WORK

Techniques	[UCI, Rapid Miner, 2019] [7]	[UCI, Matlab, 2017] [9]	[UCI, Weka, 2017] [9]
Decision Tree	82.22%	60.9%	67.7%
Logistic Regression	82.56%	65.3%	67.3%
Random Forest	84.17%	X	X
Naïve Bayes	84.24%	X	X
SVM	84.85%	67%	63.9%

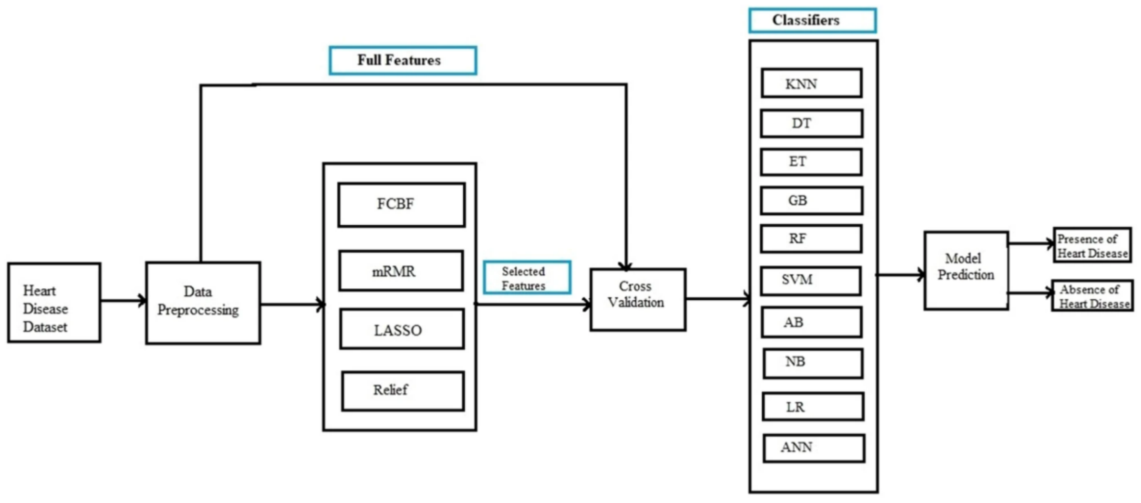
Figure 1: Findings of the paper

2.1.3 Early and Accurate Detection and Diagnosis of Heart Disease Using Intelligent Computational Model

Authors: Yar Muhammad, Muhammad Tahir, Maqsood Hayat, Kil To Chong

Journal: Scientific Reports

The article proposed an intelligent computational model for early heart disease detection. Their approach applied a combination of feature selection and ensemble learning on a clinical dataset, achieving an accuracy of 92.42% in predicting heart diseases. The high accuracy demonstrates the potential of ensemble and hybrid techniques to enhance prediction in healthcare applications. (Yar Muhammad, 2020)



An Intelligent Hybrid Framework for the prediction of heart disease.

2.1.4 Improving the Heart Disease Detection and Patients' Survival Using Supervised Infinite Feature Selection and Improved Weighted Random Forest

Authors: Abdellatif et al.

Journal: IEEE Access

This article published on IEEE site introduced a novel method that combines a supervised infinite feature selection technique with an improved weighted Random Forest classifier to detect heart disease. This approach, focused on enhancing feature relevance, yielded an accuracy of 92.14%. The study suggests that carefully selecting the most informative patient attributes and using ensemble methods can significantly improve prediction performance. (Abdellatif, et al., 2022)

3 A Novel Approach for the Effective Prediction of Cardiovascular Disease Using Applied Artificial Intelligence Techniques

Authors: Azka Mir et al.

Developed an effective cardiovascular disease prediction framework using an ensemble of Random Forest with AdaBoost (boosting) and 10-fold cross-validation across multiple heart disease datasets. In the case of the UCI Heart Disease data, their model attained a high accuracy of 93.90%. This work highlights the benefit of boosting techniques to strengthen classifier performance, as well as the importance of handling data issues (like class imbalance and missing values) to achieve robust results. (Mir, 2024)

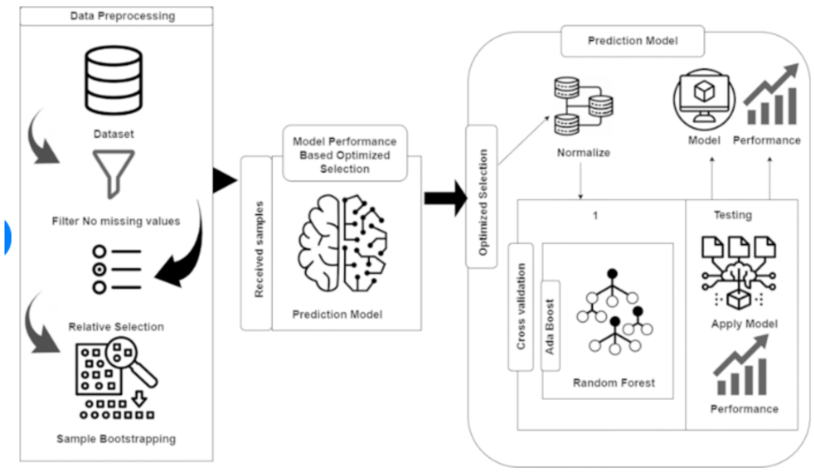


Figure 2: Aza Mir article figure

These research articles demonstrate the various ML approaches that used in the past for heart disease prediction, from single classifiers to ensembles and hybrid feature techniques. Some achieve more than 90% higher accuracy in heart disease prediction. Building on the insights from these studies, this project will compare several algorithms and attempt to maximize prediction accuracy while identifying which feature contributes most to the model decision.

3.1 Comparision of related works

A compact comparative summary (aligned with your earlier “existing systems” table style) is below.

Paper	**Bhatt et al. (2023)**	**Alotaibi (2019)**	**Muhammadetal. (2020)**	**Abdellatif et al. (2022)**	**Mir et al. (2024)**
Core goal	Compare/tune ML models for CVD prediction	Benchmark classic ML models for prediction	Intelligent ML-based diagnostic support	Improve prediction via feature selection + weighted RF	Strong prediction using applied AI, includes boosting/ensembles
Approach	Model comparison + hyperparameter tuning	Multiple classifiers + cross-validation	ML decision support framing + evaluation	Feature selection + improve	Multi-dataset evaluation, highlights

				d RF ensembl e	ensemble benefits
Typical models used	RF, DT, XGBoost, MLP	DT, LR, RF, NB, SVM	ML classification for diagnosis	Weighte d RF + feature selectio n	Ensemble / boosting-style methods

3.2 Problem Statement

Problem: Heart disease remains difficult to diagnose early due to the subtlety of risk factors and symptom variability among patients. Doctors often rely on a combination of clinical exams, patient history, and standard risk scores, which may not fully capture complex interactions between variables. This project addresses the challenge of creating a machine learning model that can accurately predict the presence of heart disease in patients using their clinical data. The dataset is relatively small handling missing data and performing proper cleaning, selecting the most relevant features, and choosing appropriate algorithms to avoid overfitting while delivering strong predictive performance is key here. Ultimately, the goal is to develop a model that could assist healthcare professionals by serving as a decision-support tool for early diagnosis of heart disease.

3.3 Research Questions / Objectives

Research Questions:

- Which machine learning algorithm can most accurately predict heart disease on our dataset?
- What are the most influential features in the dataset for predicting the presence of heart disease, and how do they correlate with the outcome?
- How can data visualization and preprocessing techniques improve the model's performance in heart disease prediction?

Objectives:

- To evaluate and compare multiple ML algorithms on performance on dataset and identify which model yields the highest predictive accuracy.
- To perform exploratory data analysis and feature important analysis to determine which patient attributes are most strongly corelated with heart disease outcomes.
- To apply appropriate data preprocessing steps and assess their impact on model performance.

- To develop a validated predictive model that can classify patients as having heart disease or not, and to report its performance using standard evaluation metrics.
- To document the findings and provide recommendations on how such a machine learning model could be applied in real life.

4 Solution

4.1 Proposed ML Algorithms

4.1.1 K-Nearest Neighbors (KNN):

K-Nearest Neighbors (KNN) is a method of supervised, non-parametric learning that can be applied to both classification and regression tasks. KNN finds the k closest examples in the training dataset, according to a distance metric, and decides the label or value for the new observation based on these examples, (IBM, n.d).

Being a non-parametric model, KNN does not learn any parameters during training and is often referred to as a “lazy learner.” It simply stores the entire dataset and computes at the time of prediction. In case of classification, the predicted class is usually assigned to the majority of the votes from the k nearest neighbors, whereas in regression the neighbors’ target values (scikit-learn, n.d.) are averaged (or distance-weighted average) and taken as the output.

The early theoretical foundation of nearest neighbor classification is largely tied to Cover and Hart’s classic result regarding the effectiveness of the nearest-neighbor rule, which in turn, led to the establishment of nearest neighbors as a primary method in pattern classification, (Cover and Hart, 1967).

One of the main advantages of KNN is that it does not require any assumptions regarding the data distribution and can therefore freely represent complicated, non-linear decision boundaries. Yet, the performance of KNN is greatly influenced by the scaling of features, the selection of a distance metric, and the parameter k. Typically, k is selected from a range of values by validation or cross-validation method in order to achieve the right level of model complexity (not too small k – overfitting; not too large k – underfitting), which is a common practice as discussed in classical references of statistical learning (Hastie, Tibshirani, and Friedman, 2009).

KNN can be considered a three-phase process, where each phase is based on some fundamental concepts:

Distance metric: A distance function defines what “close” means. A common choice is Euclidean distance for numeric features:

$$d(x, x_i) = \sqrt{\sum_{j=1}^{p} (x_j - x_{ij})^2}$$

k (number of neighbors): **K** is the parameter that governs the locality. A small value of k can lead to fitting the noise and creating irregular boundaries, on the contrary, a larger k smooths the boundary thus can aid better generalization (Hastie, Tibshirani, and Friedman, 2009).

Voting and weighting: In most cases, the prediction for classification is done through majority voting, which may or may not be weighted depending on the distance; thus, the closer neighbors will have a greater impact on the prediction. On the other hand, for regression, the prediction is usually made by taking the average (or weighted average) of the outputs of the neighbors (scikit-learn, n.d.).

Neighbor search strategy: The efficiency of the implementation is an important factor when it comes to larger datasets. Libraries for practical usage may opt for brute force search or tree-based structures such as KD-trees or Ball Trees, depending on the kind of data and the particular settings (scikit-learn, n.d.).

Pseudocode:

```
START
IMPORT libraries
LOAD dataset
ENCODE categorical features
SCALE numerical features
SPLIT into train and test sets
DEFINE range of k values to test
```

```
FOR each k:
    TRAIN KNN model
    PREDICT on test set
    EVALUATE performance metrics
SELECT the best k
END
```

Flowchart for KNN:

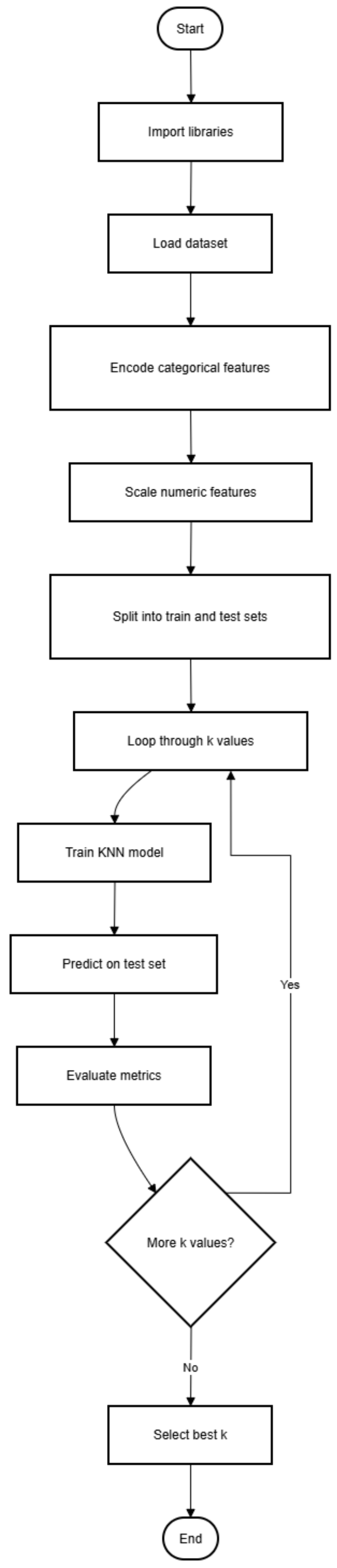
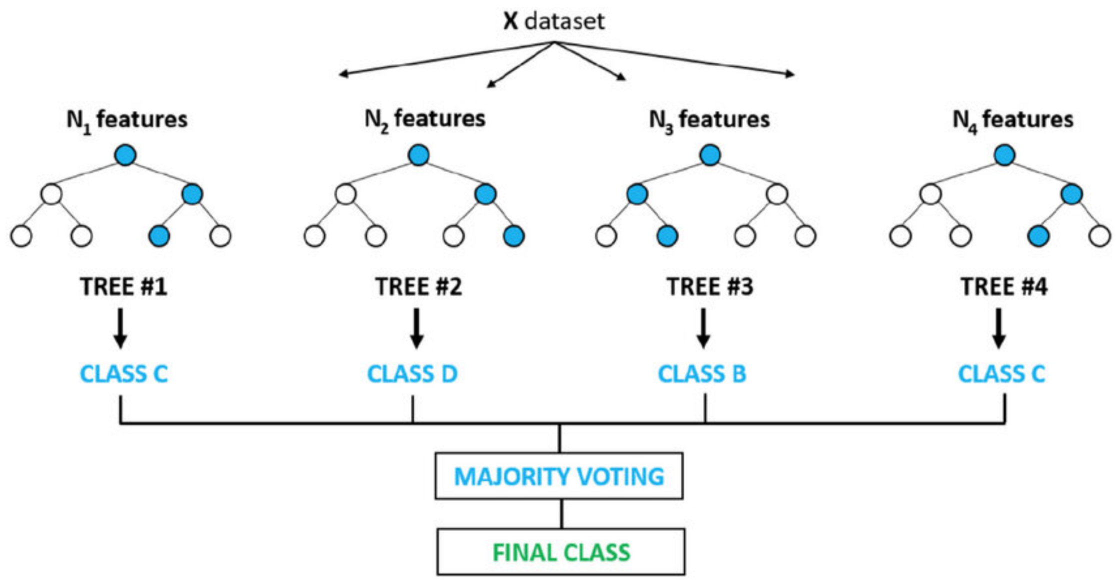


Figure 3: Knn flow chart

4.2 Random Forest:

Random Forest is an ensemble learning approach that generates numerous decision trees and then merges their outputs to deliver a single prediction that is more trustworthy. In classification, the forest will usually forecast the class selected by the majority votes from trees. For regression, it normally predicts through the average of the trees' numeric outputs. This "many weak models together" configuration not only boosts but also facilitates controlling the predictive accuracy and overfitting respectively in comparison with a single decision tree.

The technique was put forward by Leo Breiman who characterized random forests as a mixture of tree predictors with each tree trained using randomness and the final prediction being the aggregation of the results across the whole forest. He further pointed out that performance is a function of both the strength of individual trees and the degree of correlation between the trees which is precisely what randomization is trying to mitigate.



Random Forests introduce randomness primarily in two ways. Firstly, every tree is trained on a bootstrap sample which is a sample drawn with replacement from the training data. Secondly, the algorithm at each split considers only a random subset of features rather than all features, thereby reducing similarity between trees and enhancing the ensemble effect.

Pseudocode

```
START
IMPORT libraries
LOAD dataset
ENCODE categorical features
SPLIT into train and test sets
DEFINE range of n_estimators and max_depth
FOR each hyperparameter combination:
    TRAIN Random Forest model
    PREDICT on test set
    EVALUATE performance
SELECT best model
OUTPUT feature importance
END
```

Flow chart for random forest:

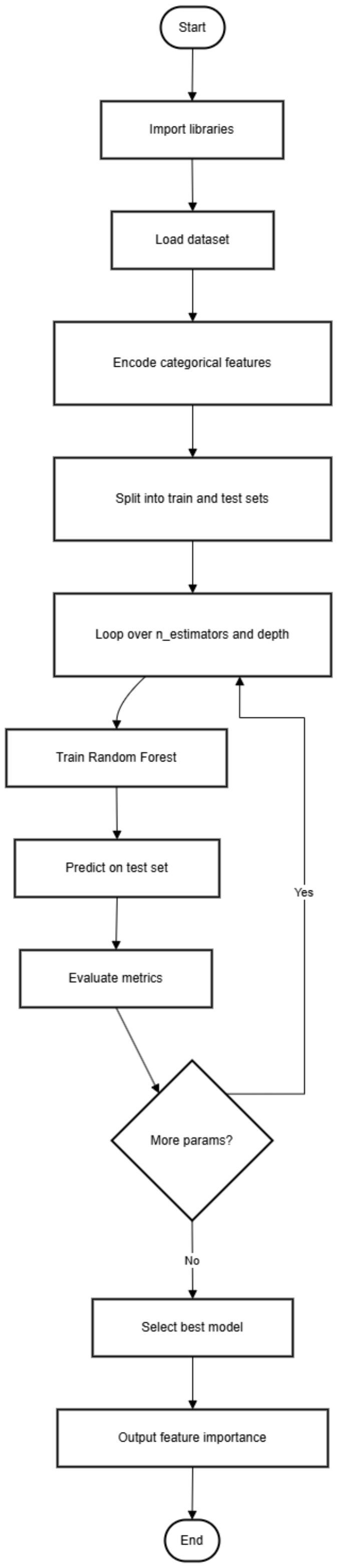


Figure 4: Random forest flow chart

4.3 Support Vector Machine (SVM):

SVM is a powerful margin- based classifier that finds the optimal decision boundary while separating the two classes that maximizes the margin, meaning the gap between the boundary of a class and closest training point for each class. SVMs can model nonlinear boundaries using kernel functions as well. They are effective in high-dimensional feature spaces and are robust to outliers (using the soft-margin parameter C). Kernel parameters need to be fine-tuned and regularization strength to get the best performance from SVM. (Scikitlearn, n.d.)

SVMs have been utilized in numerous fields such as text categorization, image classification, bioinformatics, and fraud detection, which demand dependable classification borders. The selection of a suitable kernel function coupled with the tuning of regularization parameters that control the balance between margin maximization and classification error mainly determines their capacity to represent complicated decision surfaces. (Kavlakoglu, n.d.)

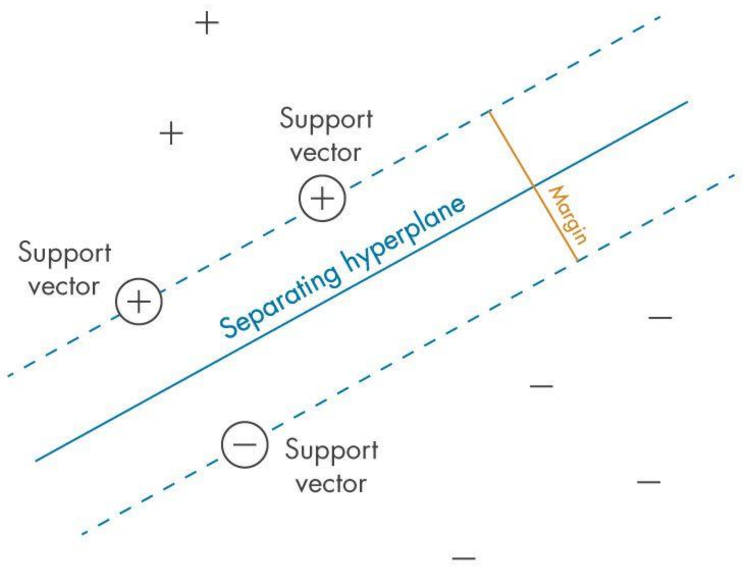


Figure 5: SVM image

An SVM model is composed of several core elements which dictate its separation of classes and its generalization to new data points:

Hyperplane: For SVMs, a hyperplane is the decision boundary that delineates different classes in the feature space. A hyperplane in case of a d-dimensional feature space can be expressed by the following equation:

$$w \cdot x + b = 0$$

Whereas, in this expression w indicates the vector of weights that is perpendicular to the hyperplane, x refers to the feature vector, and b denotes the bias term that moves the hyperplane concerning the origin. The SVM training process aims at discovering the w and b values that create the largest separation between the classes.

Support Vectors: Support vectors represent the dataset points that are nearest to the hyperplane. These are the most important points for defining the location and angle of the hyperplane since they are directly responsible for the margin. In effect, the SVM model relies upon these support vectors as opposed to all the training samples.

Margin: Margin refers to the shortest length of a line that is perpendicular to the hyperplane and touches the support vectors. The SVM algorithm's objective of maximizing the margin is because usually wider margins mean better generalization on previously unseen data.

Pseudocode

```
START
IMPORT libraries
LOAD dataset
ENCODE categorical features
SCALE numeric features
SPLIT into train and test sets
DEFINE grid of (kernel, C, gamma)
FOR each hyperparameter combination:
    TRAIN SVM model
    PREDICT on test set
    EVALUATE metrics
SELECT best model
END
```

Flowchart for SVM:

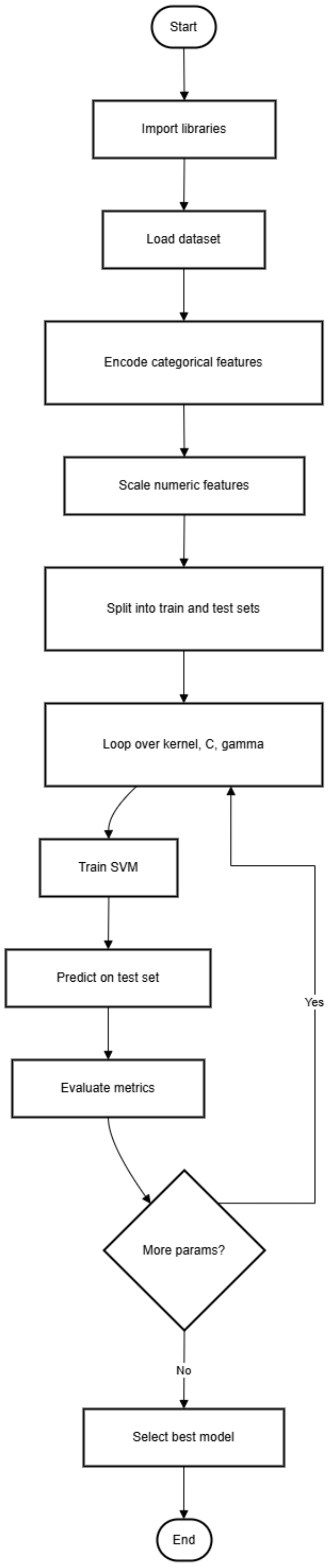
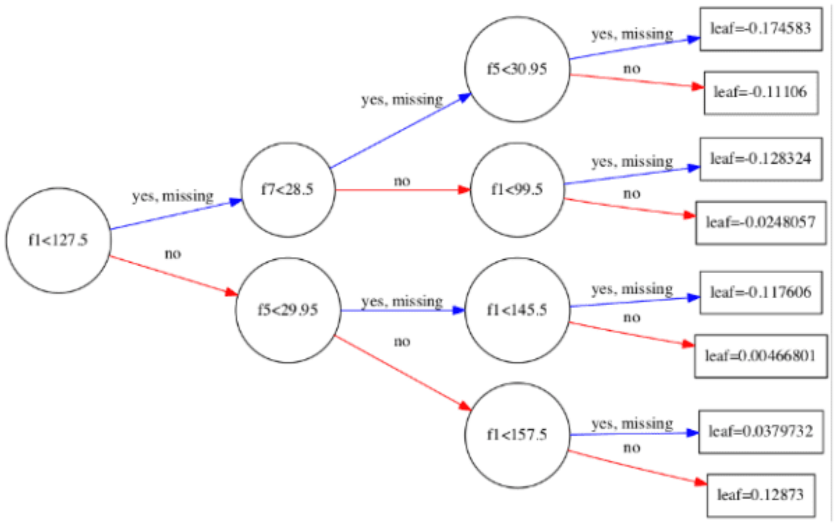


Figure 6:SVM Flow chart

4.4 XGBoost (Extreme Gradient Boosting):



XGBoost (Extreme Gradient Boosting) is a machine learning algorithm that operates under the supervised mechanism and applies the gradient boosting method to create decision trees as an ensemble. Unlike the native method which constructs one large tree, XGBoost builds up an ensemble with multiple small trees that are trained in turn. A tree is added each time that targets to eliminate the mistakes made by the current ensemble, thus, the model gets better bit by bit and not all at once. This setting was formalized by Chen and Guestrin as a scalable tree boosting system that has become widely acknowledged as the top performer in predictive accuracy across a variety of machine learning tasks.

The distinguishing factor from simpler gradient boosting is that it merges the latter with the controlling for complexity and overfitting explicitly. The adopting method to optimize the objective function comprises a loss term (wrong predictions) and a regularization term (complex trees incurring a penalty). Such an approach bolsters the model thus making it able to withstand even the noisiest or the smallest amount of data.

Pseudocode:

```
START
IMPORT libraries
LOAD dataset
EXPLORE dataset (shape, types, missing values)
DEFINE X = all features, y = binary target
ENCODE categorical features (one-hot)
SPLIT into train and test sets (stratified)
DEFINE grid of (n_estimators, max_depth, learning_rate, subsample, colsample_bytree, reg_alpha, reg_lambda)
FOR each hyperparameter combination:
TRAIN XGBoost classifier (objective = binary:logistic)
PREDICT on test set (labels and probabilities)
EVALUATE metrics (Accuracy, Precision, Recall, F1, ROC-AUC)
SELECT best model
REPORT feature importance (gain or split)
END
```

Flowchart for XGBoost:

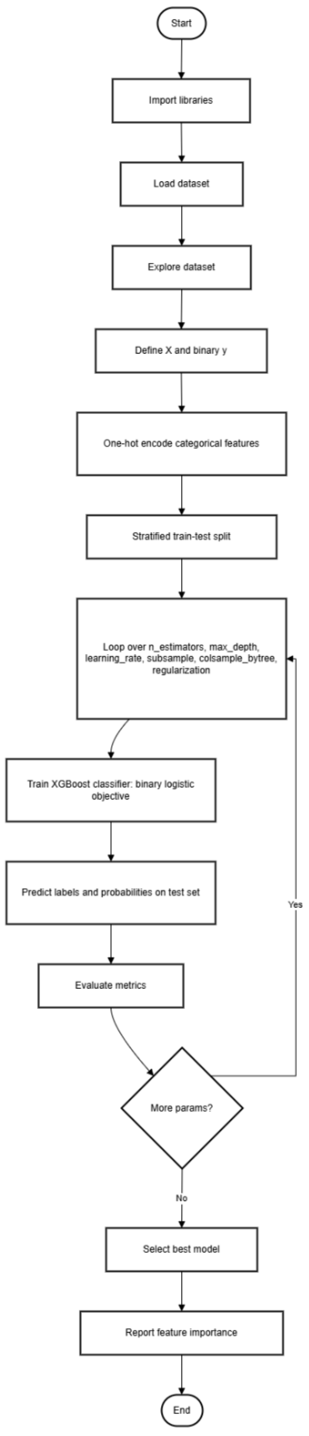

Start
Import libraries
Load dataset
Explore dataset
Define X and binary y
One-hot encode categorical features
Stratified train-test split
Loop over n_estimators, max_depth, learning_rate, subsample, colsample_bytree, regularization
Train XGBoost classifier: binary logistic objective
Predict labels and probabilities on test set
Evaluate metrics
More params?
Yes
No
Select best model
Report feature importance
End

Figure 7: XGboost flow chart

4.5 Dataset Description

We will use the heart disease dataset from the UC Irvine Machine Learning Repository. The repository describes four related databases from Cleveland, Hungary, Switzerland, and the VA Long Beach system. The full dataset contains 76 attributes, and most published experiments use a standard subset of 14 attributes.

The input features include clinical and test-related variables such as:

- **Age:** Age of the patient in years.
- **Sex**: Sex of the patient coded as 1 and 0.
- **CP (Chest Pain Type):** Category of chest pain.
- **Trestbps (Resting Blood Pressure):** Resting blood pressure in mm Hg.
- **Chol (Serum Cholesterol):** Serum cholesterol in mg/dl.
- **FBS (Fasting Blood Sugar):** Indicator for fasting blood sugar above 120 mg/dl.
- **Restecg (Resting ECG Results):** Encoded resting ECG results.
- **Thalach (Max Heart Rate Achieved):** Maximum heart rate achieved during exercise.
- **Exang (Exercise Induced Angina):** 1 for yes, 0 for no.
- **Oldpeak (ST Depression):** ST depression induced by exercise relative to rest.
- **Slope:** Slope of the peak exercise ST segment.
- **Ca:** Number of major vessels colored by fluoroscopy.
- **Thal:** Thallium stress test result code.
- **Num:** Target

4.6 Methodology

This project builds a heart disease prediction system using multiple supervised classification models Logistic Regression, Support Vector Machine, and Random Forest and XGboost. All the models use the same dataset split and the same preprocessing rules apply to every model till available so we can compare results.

Step 1. Data loading

We first load the heart disease dataset from a CSV file. Record the number of rows, columns names and data types, then prepare it for further preprocessing.

Step 2. Target definition

Define the prediction label as binary heart disease status. Use one rule across the project and state it once in the report so the reader knows the exact meaning of class 0 and class 1.

Step 3. Data screening and quality control

Scan every column and row for missing values and duplicate rows. Perform tests to find impossible values and obvious entry errors. Apply a documented method to fix the issues in the data set

Step 4. Feature typing and encoding plan

Then group variables into numeric and categorical sets treating coded fields such as cp, restecg, slope, thal, and ca as categorical which represent categories even when changed to integers.

Step 5. Train test split

Then split the data into training and test sets using stratification. This keeps the class ratio stable in both sets for machine learning training for all models. Hold the test set for final evaluation only.

Step 6. Preprocessing pipeline

Build a preprocessing pipeline that can be reused for every model. Standardize numeric variables and encode categorical variables so scale does not distort distance or margin calculations.

Step 7. Run Baseline Machine Models

Then train each model with default settings as a baseline for the model primarily using the training set only. Record baseline metrics so we can make improvements from fine tuning later.

Step 8. Hyperparameter tuning with cross-validation

Tune each ML model using cross-validation on the training set. Use one primary tuning score, F1-score for the positive class keeping the tuning strategy consistent across models.

Step 9. Final training with best settings

Perform fitting for each model on the full training set using the selected hyperparameters. Save the final configurations and fit it in to the models.

Step 10. Test set evaluation

Evaluate every final model on the held-out test set. Perform model evaluation Accuracy, Precision, Recall, F1-score, and ROC-AUC. Present the confusion matrix for each model.

Step 11. Error analysis and threshold adjustment

Inspect false negatives and false positives. Treat false negatives as high-risk errors for clinical screening context. Test threshold adjustments using predicted probabilities, then report the metric trade-offs in a small table.

Step 12. Model interpretation

Interpret what all models individually to evaluate actual final performance.

Step 13. Model comparison and selection

Then compare the models using a single results table on the same test set. Select the best model based on the project goal, prioritizing recall and F1-score for the positive class.

General flow chart:

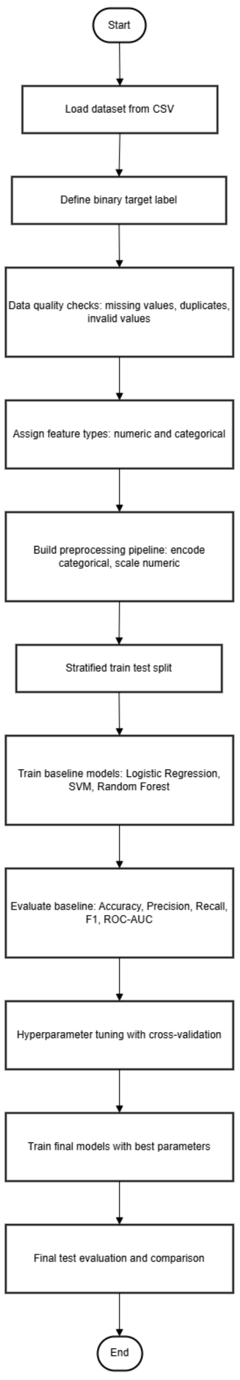

Start
Load dataset from CSV
Define binary target label
Data quality checks: missing values, duplicates, invalid values
Assign feature types: numeric and categorical
Build preprocessing pipeline: encode categorical, scale numeric
Stratified train test split
Train baseline models: Logistic Regression, SVM, Random Forest
Evaluate baseline: Accuracy, Precision, Recall, F1, ROC-AUC
Hyperparameter tuning with cross-validation
Train final models with best parameters
Final test evaluation and comparison
End

Figure 8: General flow chart

4.7 Evaluation Metrics

To assess the actual performance of the machine learning models, several evaluation metrics common in classification tasks will be used, such as (Neptune.ai, 2025):

- **Accuracy**: Accuracy in classification models simply put is the number of correct predictions divided by the total number of predictions in percentage. Accuracy can be used as a quick overall indicator for the model for interpretation alongside class focused metrics.
- **Precision** (Positive Predictive Value): The ratio of predicted positive cases that are truly positive. Precision focuses on Type-I errors to check when a model predicts presence of heart disease, how often the prediction is correct.

  P=TP/TP+FP

- **Recall or Sensitivity:** The share of true positive cases the model identifies correctly. Recall measures how well the model finds heart disease cases in the dataset.

  R= TP / TP + FN

- **F1-score**: It is the harmonic mean of precision and recall. F1-score summarizes the trade-off between false positives and false negatives in one number, which supports model comparison when precision and recall when thy are moving indifferent directions.

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

- **Confusion matrix**: It is the tabular version of true positives, true negatives, false positives, and false negatives. This provides a full error breakdown by class with assumption our null hypothesis Ho is the “patient doesn’t have heart condition”. Specificity, the true negative rate, is derived from the confusion matrix and reports how well the model identifies non positive cases.

- **ROC-AUC**: The ROC curve plots true positive rate against the false positive rate across decision thresholds. AUC summarizes this curve into one threshold independent score supporting the model comparison when probability outputs are available and helps assess ranking quality.
- **Cross-validation scores**: Average performance across multiple training validation splits. Cross-validation estimates generalization and helps detect overfitting when training performance and validation performance differ.

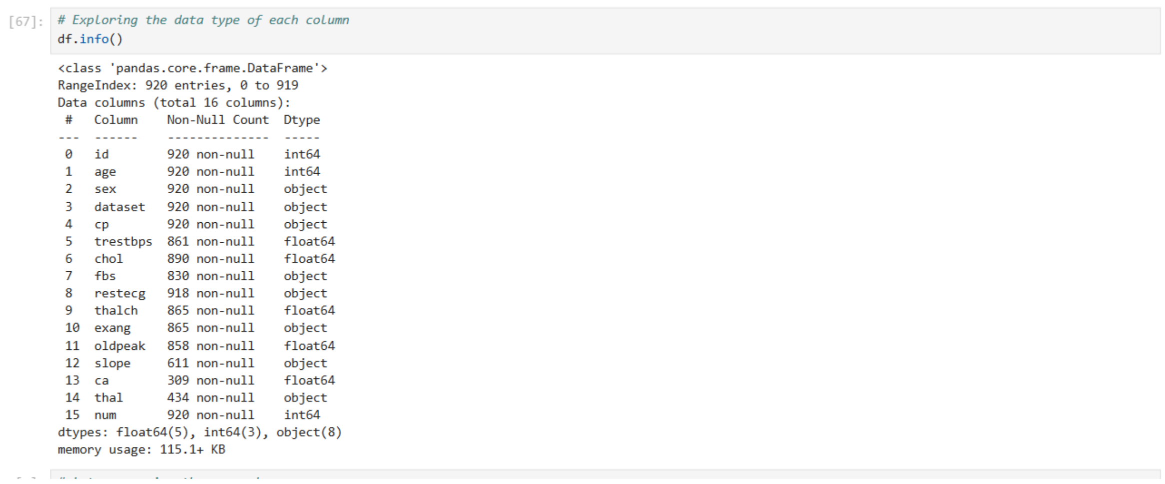
4.8 Tools and Technologies

- **Language programming**: Python for data preparation, visualization, and model training.
- **Development environment:** Jupyter Notebook for running code, showing outputs, and keeping analysis reproducible in one file. (Jupyter, n.d.)
- **Data handling:** pandas for loading the CSV and preparing clean feature tables for modeling.
- **Visualization:** Matplotlib and Seaborn for EDA plots such as distributions, boxplots, and correlation heatmaps.
- **Machine learning:** scikit-learn for preprocessing, train test splitting, cross validation, hyperparameter tuning, and evaluation metrics.
- **Version control:** Git for tracking changes, with GitHub for storing code and sharing notebook.

4.9 Achieved Result

4.9.1 Exploring the data type of each column

Inspects column types and non-null counts to understand the dataset.

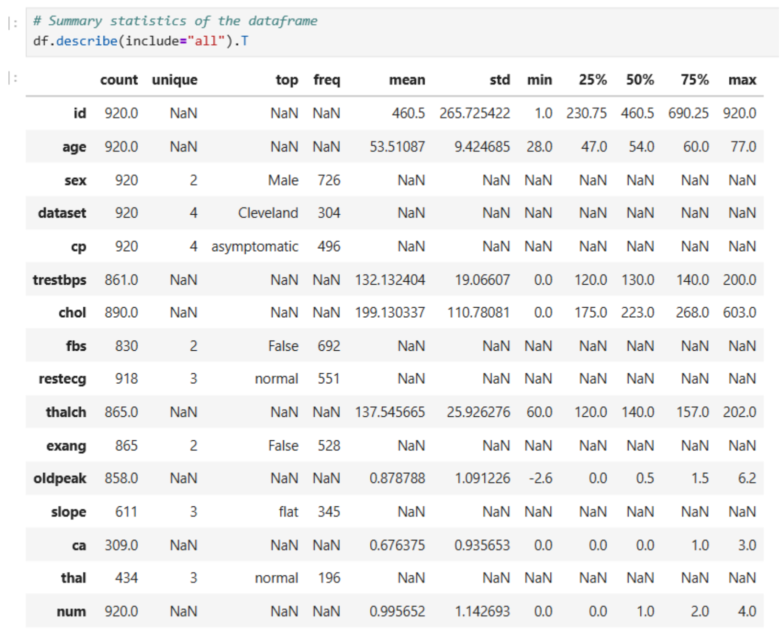


```
[67]: # Exploring the data type of each column
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   id        920 non-null    int64  
 1   age       920 non-null    int64  
 2   sex       920 non-null    object 
 3   dataset   920 non-null    object 
 4   cp        920 non-null    object 
 5   trestbps  861 non-null    float64
 6   chol      890 non-null    float64
 7   fbs       830 non-null    object 
 8   restecg   918 non-null    object 
 9   thalch    865 non-null    float64
 10  exang     865 non-null    object 
 11  oldpeak   858 non-null    float64
 12  slope     611 non-null    object 
 13  ca        309 non-null    float64
 14  thal      434 non-null    object 
 15  num       920 non-null    int64  
dtypes: float64(5), int64(3), object(8)
memory usage: 115.1+ KB
```

Figure 3 Exploring the data type of each column

4.9.2 Summary statistics of the dataframe

This cell executes: Summary statistics of the dataframe



```
# Summary statistics of the dataframe
df.describe(include="all").T
```

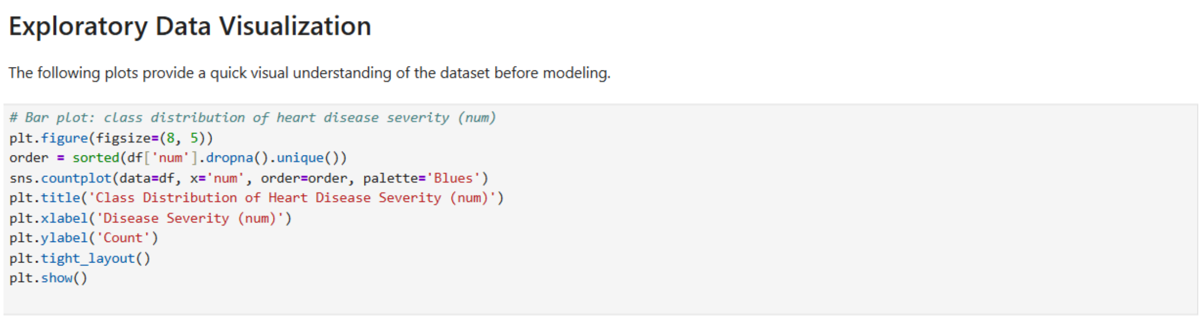
	count	unique	top	freq	mean	std	min	25%	50%	75%	max
id	920.0	NaN	NaN	NaN	460.5	265.725422	1.0	230.75	460.5	690.25	920.0
age	920.0	NaN	NaN	NaN	53.51087	9.424685	28.0	47.0	54.0	60.0	77.0
sex	920	2	Male	726	NaN	NaN	NaN	NaN	NaN	NaN	NaN
dataset	920	4	Cleveland	304	NaN	NaN	NaN	NaN	NaN	NaN	NaN
cp	920	4	asymptomatic	496	NaN	NaN	NaN	NaN	NaN	NaN	NaN
trestbps	861.0	NaN	NaN	NaN	132.132404	19.06607	0.0	120.0	130.0	140.0	200.0
chol	890.0	NaN	NaN	NaN	199.130337	110.78081	0.0	175.0	223.0	268.0	603.0
fbs	830	2	False	692	NaN	NaN	NaN	NaN	NaN	NaN	NaN
restecg	918	3	normal	551	NaN	NaN	NaN	NaN	NaN	NaN	NaN
thalch	865.0	NaN	NaN	NaN	137.545665	25.926276	60.0	120.0	140.0	157.0	202.0
exang	865	2	False	528	NaN	NaN	NaN	NaN	NaN	NaN	NaN
oldpeak	858.0	NaN	NaN	NaN	0.878788	1.091226	-2.6	0.0	0.5	1.5	6.2
slope	611	3	flat	345	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ca	309.0	NaN	NaN	NaN	0.676375	0.935653	0.0	0.0	0.0	1.0	3.0
thal	434	3	normal	196	NaN	NaN	NaN	NaN	NaN	NaN	NaN
num	920.0	NaN	NaN	NaN	0.995652	1.142693	0.0	0.0	1.0	2.0	4.0

Figure 9: Summary statistics of the dataframe

4.9.3 Section: Exploratory Data Visualization

A descriptive summary of the age column was generated to understand its distribution and range. Following such, the code is executed as following:

A bar-graph displaying the distribution of the target variable (num) for class. The code is provided as follows:



Exploratory Data Visualization

The following plots provide a quick visual understanding of the dataset before modeling.

```
# Bar plot: class distribution of heart disease severity (num)
plt.figure(figsize=(8, 5))
order = sorted(df['num'].dropna().unique())
sns.countplot(data=df, x='num', order=order, palette='Blues')
plt.title('Class Distribution of Heart Disease Severity (num)')
plt.xlabel('Disease Severity (num)')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

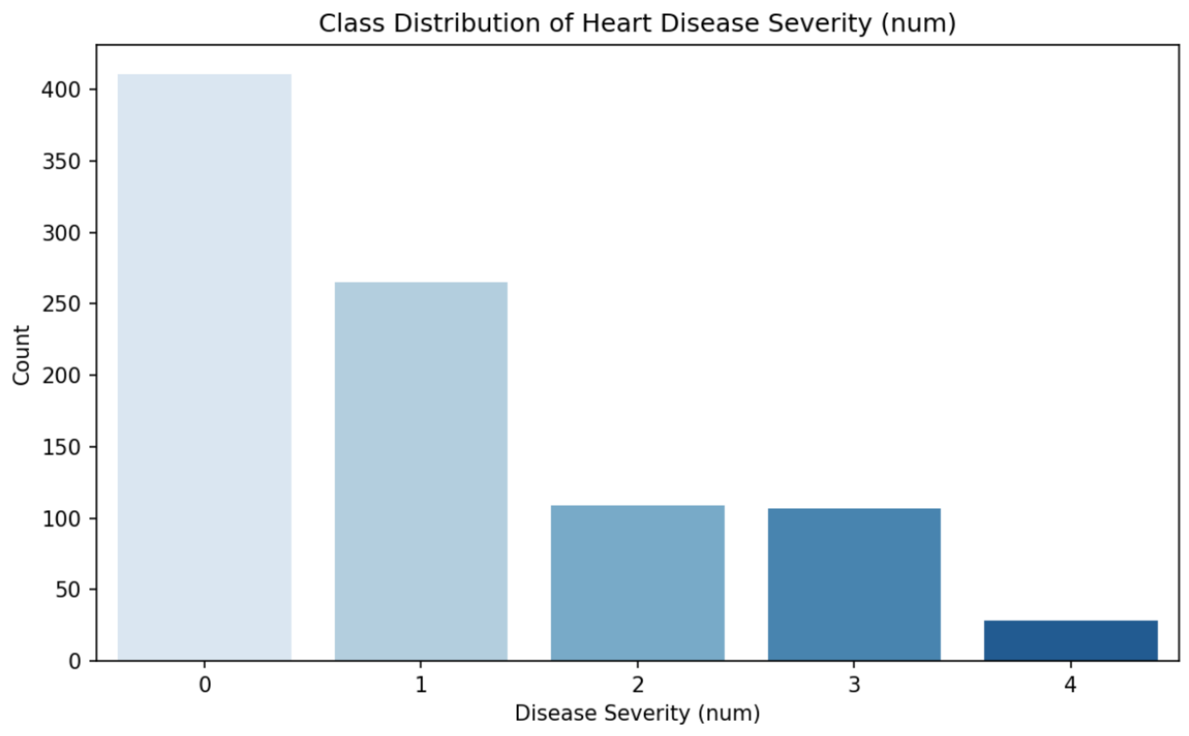


Figure 10: Class distribution of heart disease severity

The class distribution is not equal: class 0 with least heart disease cases is the most frequent, on the other hand, class 4 which indicates the most severe case is hardly ever seen. So, to enhance the reliability and clinical readability of the task, I suggest combining the target categories into two: heart disease not present (class 0) and heart disease present (classes 1–4). In this manner, the learning process will be more stable and less influenced by the sparsity of the higher severity classes, and you can administer imbalance-aware evaluation (like precision, recall, F1-score, ROC-AUC, and PR-AUC) along with stratified splits to guarantee that no bias exists toward the majority class.

4.9.4 Bar Plot showing Binary conversion

The plot indicates that the initial 0–4 target was extremely imbalanced (with a very small number of severe cases), therefore it was transformed into a binary label to develop a more stable and clinically useful "disease vs no disease" which is more suitable for the small size of the data set.

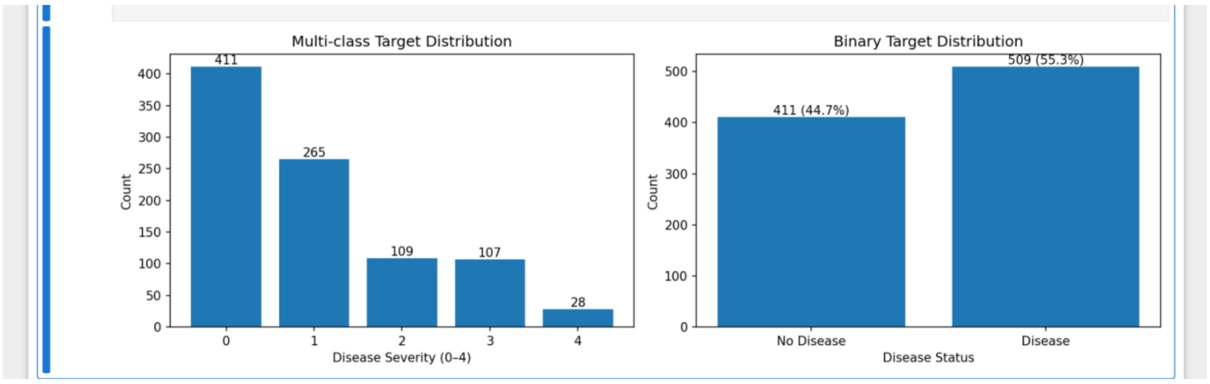
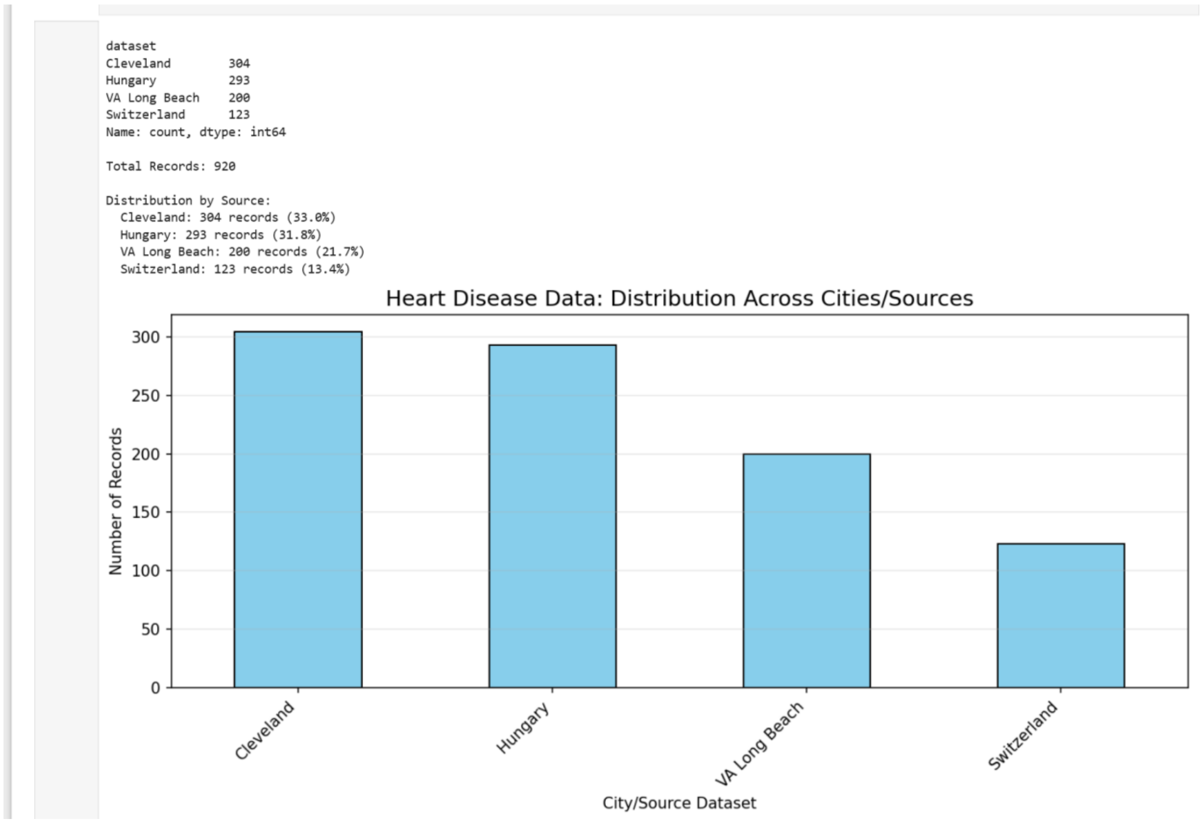


Figure 11:4.9.4 Bar Plot showing Binary conversion

4.9.5 Bar chart for distribution of data set according to city

The dataset has 920 records, mostly from Cleveland (304) and Hungary (293), with fewer from VA Long Beach (200) and Switzerland (123).



```
dataset
Cleveland        304
Hungary          293
VA Long Beach    200
Switzerland      123
Name: count, dtype: int64

Total Records: 920

Distribution by Source:
  Cleveland: 304 records (33.0%)
  Hungary: 293 records (31.8%)
  VA Long Beach: 200 records (21.7%)
  Switzerland: 123 records (13.4%)
```

4.9.6 Histograms for numerical features

Histograms for numeric features were plotted to examine distribution shapes and potential skewness

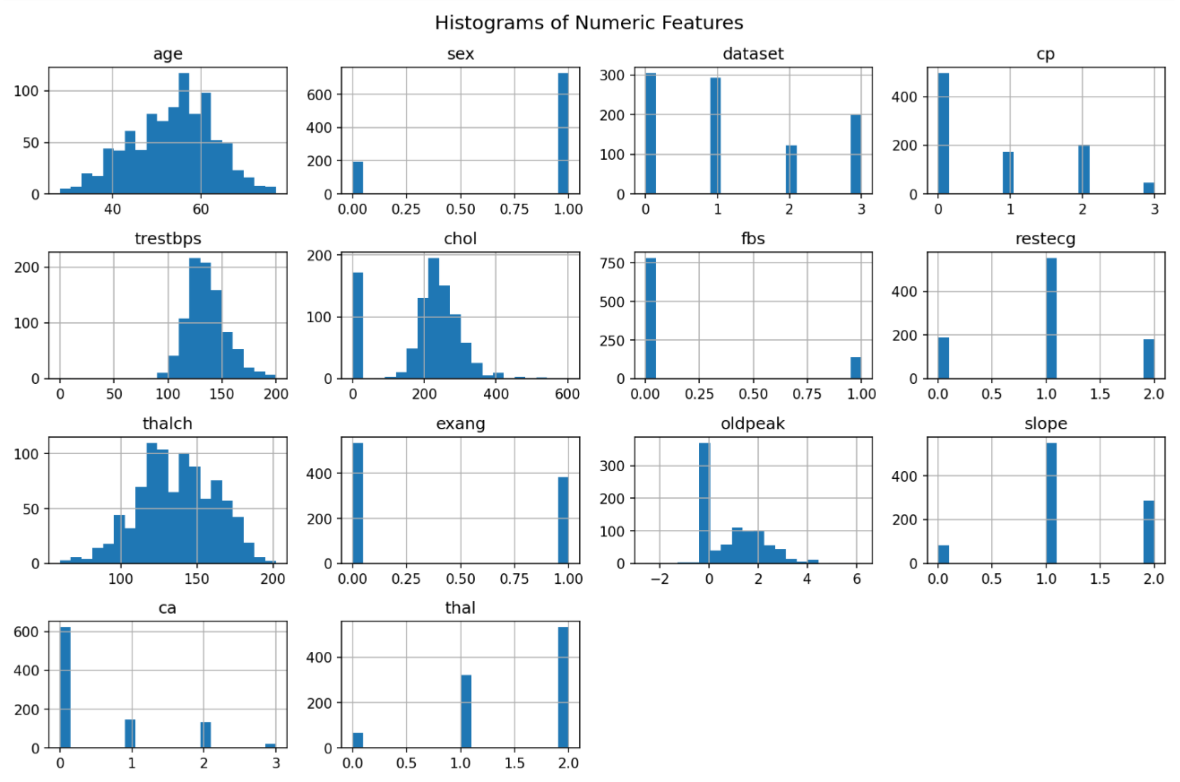
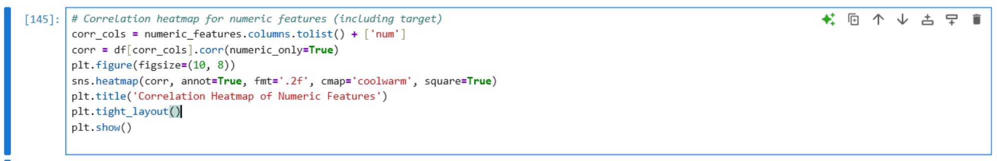


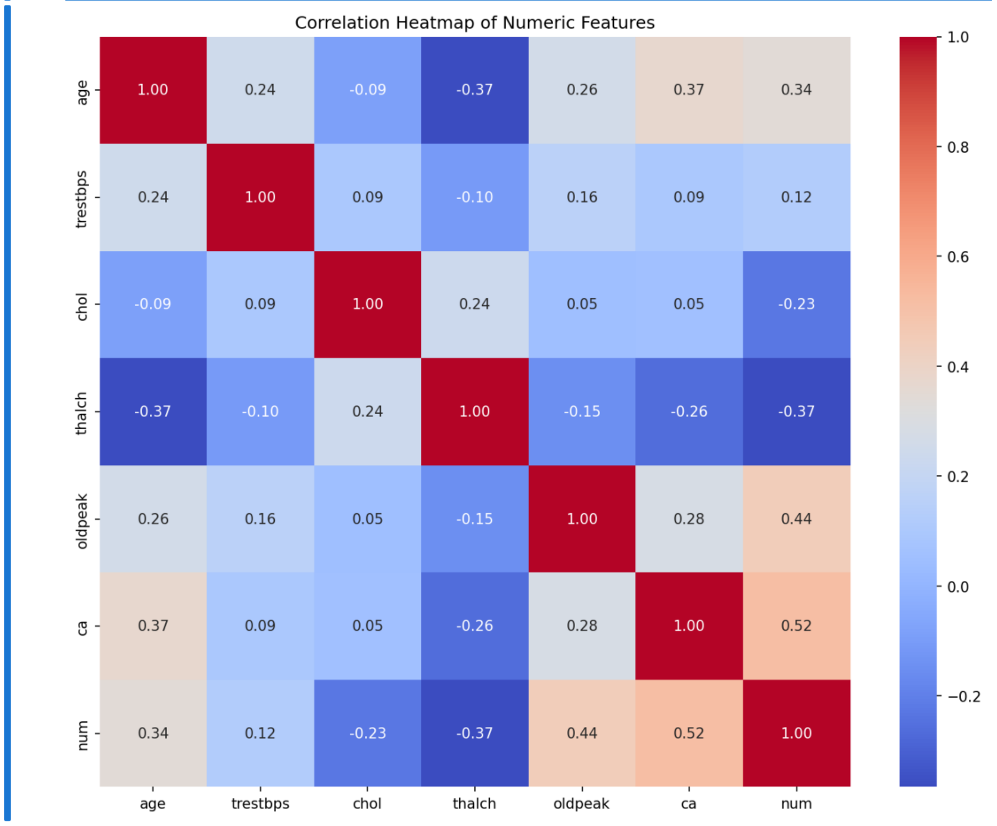
Figure 12 Histograms of numeric features

4.9.7 Correlation heatmap for numeric features (including target)

This correlation heatmap was generated to identify relationships between numeric features and the target. According to your graph, the colors have warm hues for indicating heavy positive correlations and cool hues for heavy negative correlations. This technique facilitates rapid identification of the features that correlate and the ones that are anticorrelated. This is helpful in exploratory data analysis to ascertain possible predictors, discover feature overlaps, and direct feature selection or engineering prior to model training.



```
[145]: # Correlation heatmap for numeric features (including target)
corr_cols = numeric_features.columns.tolist() + ['num']
corr = df[corr_cols].corr(numeric_only=True)
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt='.2f', cmap='coolwarm', square=True)
plt.title('Correlation Heatmap of Numeric Features')
plt.tight_layout()
plt.show()
```

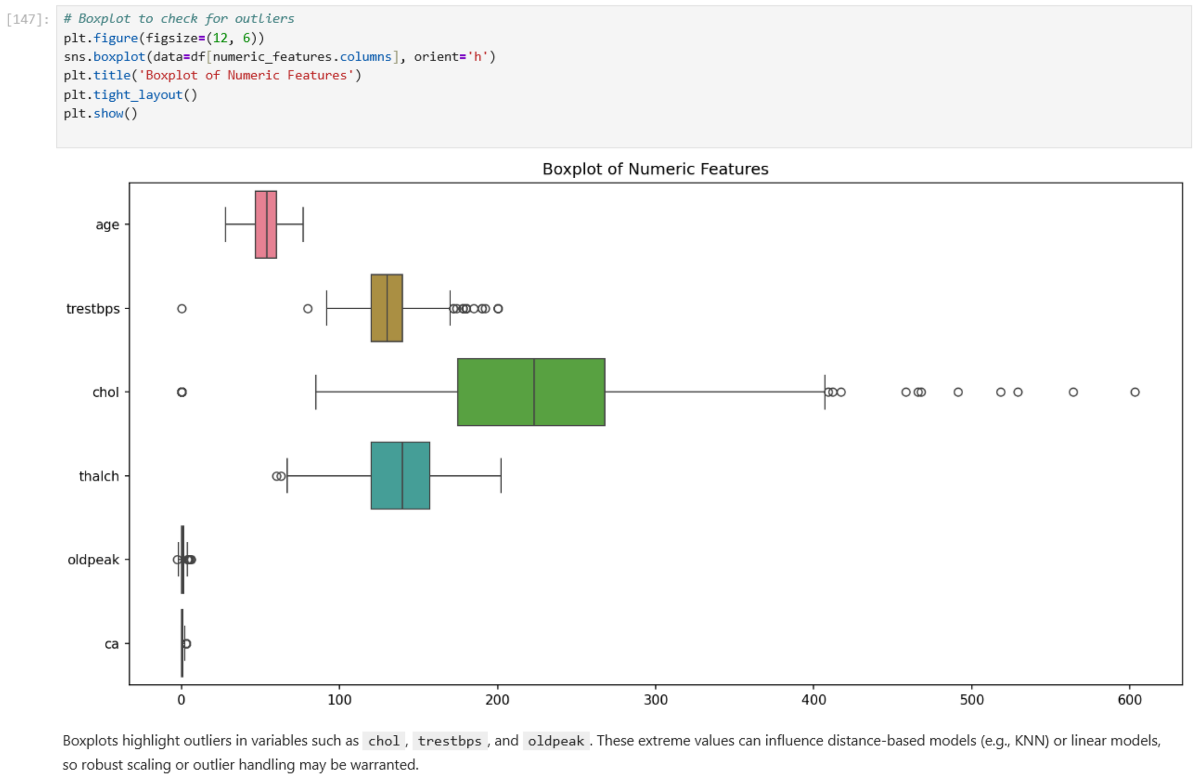


The heatmap shows moderate relationships between the target (`num`) and features like `ca` and `oldpeak`, while `thalch` is negatively correlated with `num`. Most feature-feature correlations are not extreme, suggesting limited multicollinearity among numeric predictors.

Figure 13: Correlation heatmap of numeric features (including target)

4.9.8 Boxplot to check for outliers

Uses boxplots to surface outliers and spread for numeric features



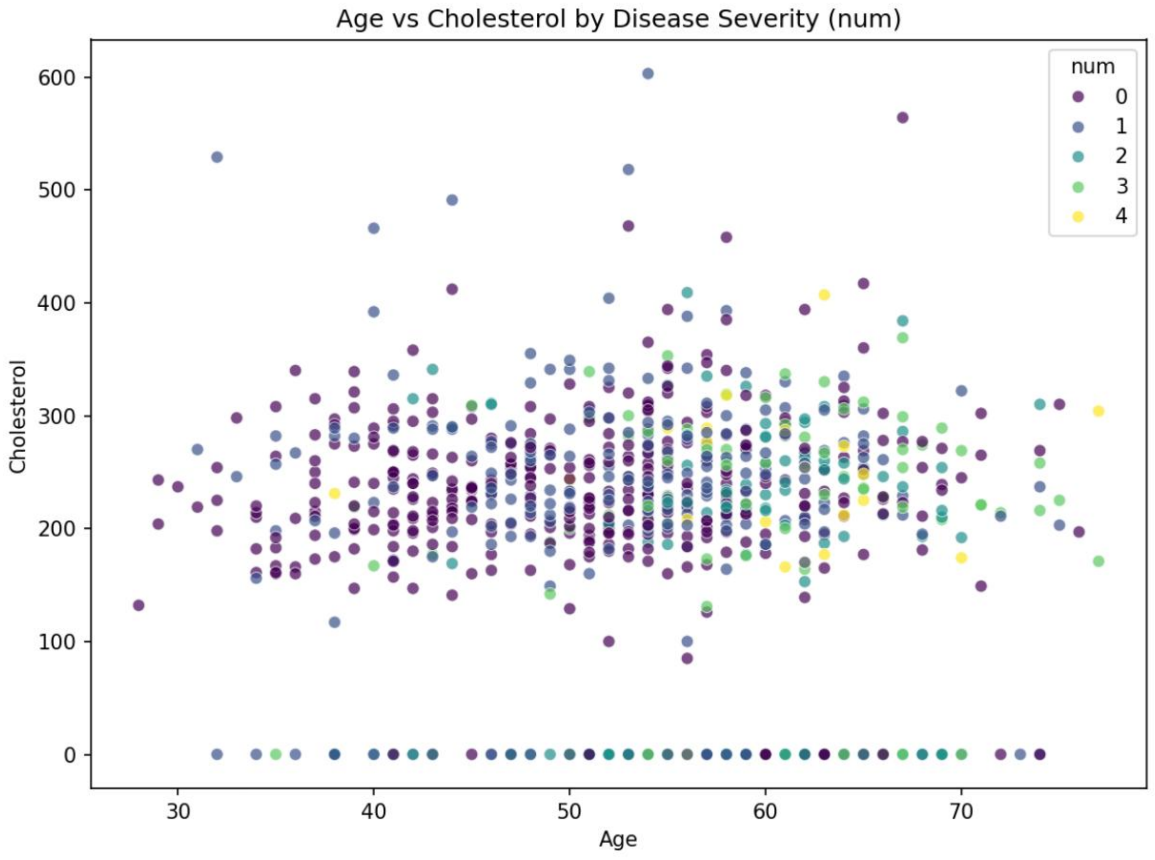
```
[147]: # Boxplot to check for outliers
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[numeric_features.columns], orient='h')
plt.title('Boxplot of Numeric Features')
plt.tight_layout()
plt.show()
```

Boxplots highlight outliers in variables such as `chol`, `trestbps`, and `oldpeak`. These extreme values can influence distance-based models (e.g., KNN) or linear models, so robust scaling or outlier handling may be warranted.

Figure 8 Boxplot of numeric features

4.9.9 Scatter plot to inspect relationships (age vs cholesterol)

```
9]: # Scatter plot to inspect relationships (age vs cholesterol)
    plt.figure(figsize=(8, 6))
    sns.scatterplot(data=df, x='age', y='chol', hue='num', palette='viridis', alpha=0.7)
    plt.title('Age vs Cholesterol by Disease Severity (num)')
    plt.xlabel('Age')
    plt.ylabel('Cholesterol')
    plt.tight_layout()
    plt.show()
```



The scatter plot shows substantial overlap between severity classes across both age and cholesterol, which suggests that no simple linear boundary separates the classes.
This supports trying non-linear models (e.g., Random Forest, SVM with non-linear kernels) in addition to linear baselines.

Figure 14: Age vs Cholesterol scatter plot by disease severity.

The relationship between age and cholesterol is shown in this scatter plot, where the points are colored according to the severity of the disease (target class). Scatter plots are often the first choice to visualize the relationship between two continuous variables and to get a rough idea of the data's overall patterns or possible outliers occurring.

The overlapping nature of the severity groups in the dataset across almost all ages and cholesterol levels indicates that age and cholesterol alone cannot distinctly separate classes, and thus, a model would need to rely on multiple features and non-linear relationships to differentiate outcomes.

4.9.10 Violin plot for key features

The distributions of the most important numeric characteristics for the different disease classes are shown in these violin plots that indicate the tendency of thalach being lower and oldpeak higher in the disease groups, while age, trestbps, and cholesterol feature more overlap and less clear separation.

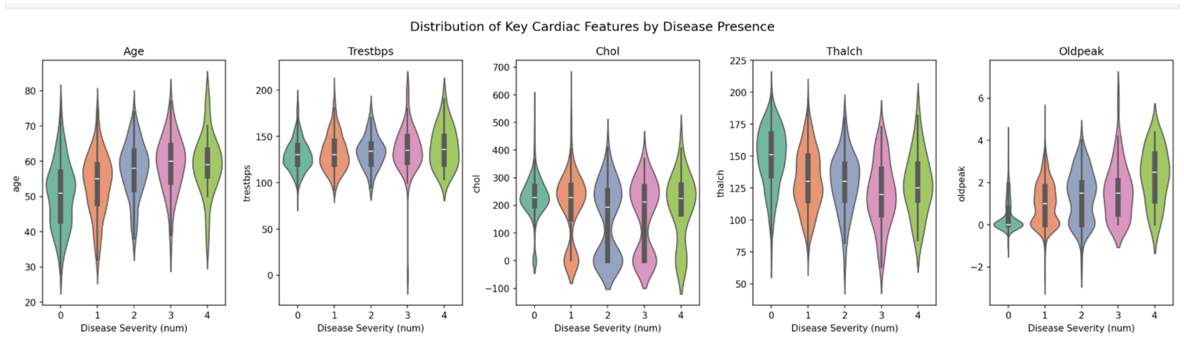


Figure 15: Violin plot for key features

4.9.11 Pairwise Feature Relationships by Disease Status

This pair plot compares age, cholesterol, resting blood pressure, and maximum heart rate by disease status, showing noticeable class overlap overall but with clearer separation in **thalach** (disease cases tend to have lower maximum heart rate) and a slight shift toward higher age in the disease group.

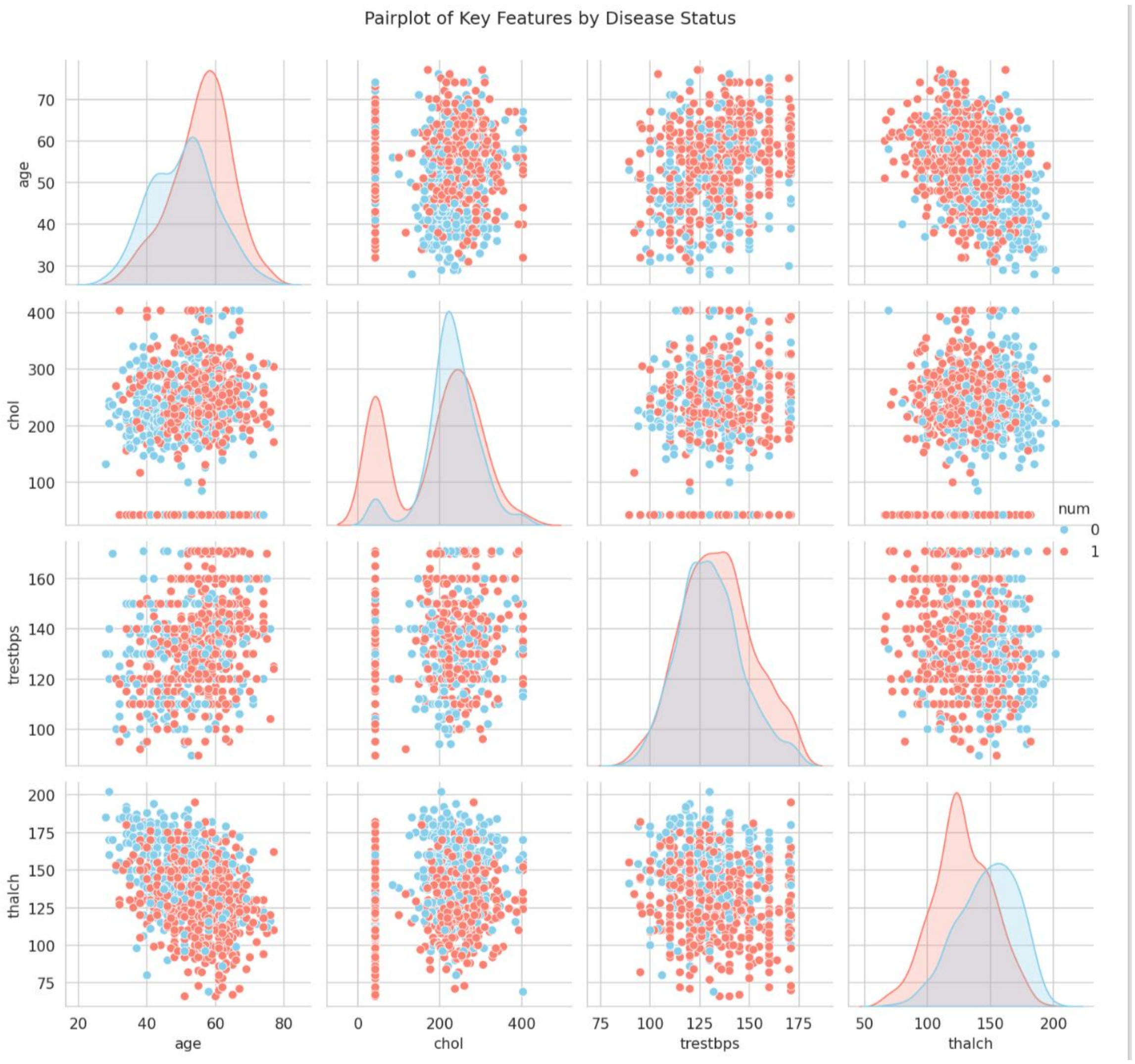
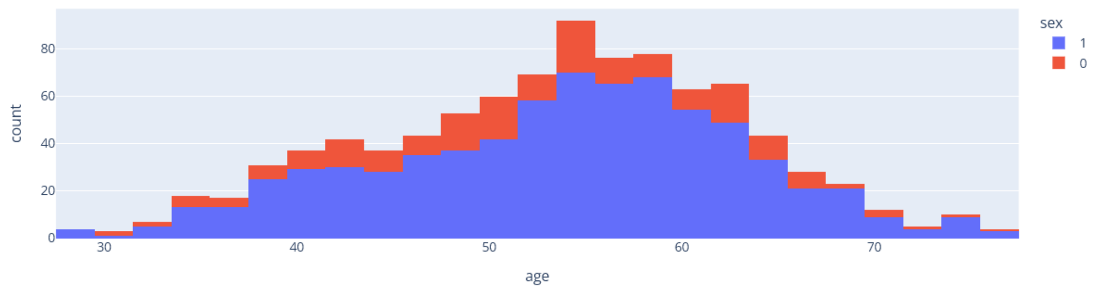


Figure 16: Pairplot of key features

4.9.12 Histogram of age column

A Plotly histogram was used to compare age distributions by sex for a more interactive view.

```
[199]: fig = px.histogram(data_frame=df, x='age', color= 'sex')
       fig.show()
```



4.9.13 Categorical Feature Distributions by Heart Disease Status

The counts of disease vs no disease for the different categorical features (sex, chest pain type, fasting blood sugar, exercise induced angina, and resting ECG) are clearly distributed differently, as illustrated by these grouped bar charts. This is especially true for the case of cp and exang, which seem to be more strongly related to disease status than fbs and restecg, where the classes are not so clearly defined.

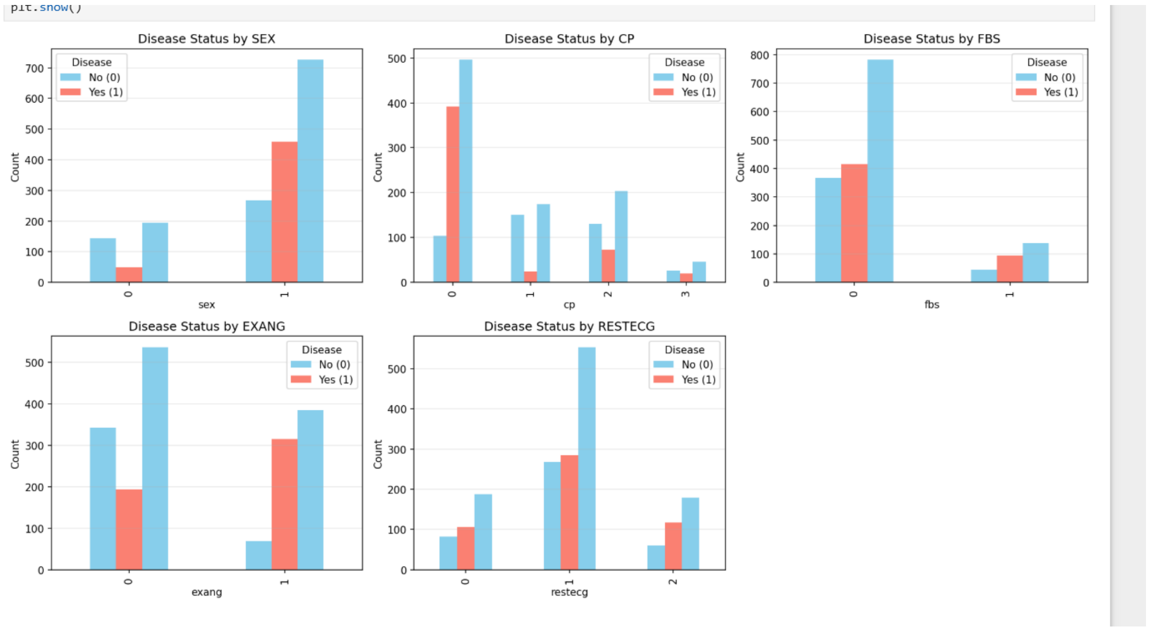
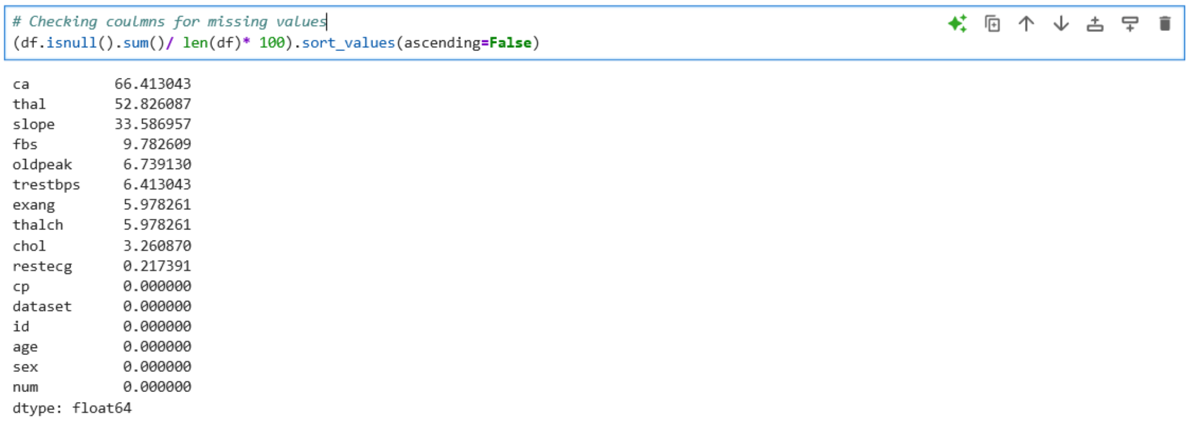


Figure 17: Grouped bar charts showing disease (1) vs no disease (0) counts across categorical variables

4.9.14 Data Cleaning and Preprocessing

4.9.15 Checking null values

Calculating the missing value percentages to identify columns requiring imputation.

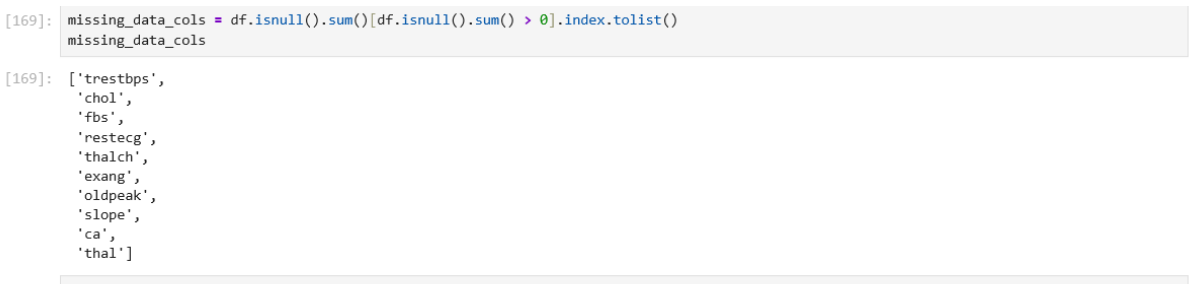


```
# Checking coulmns for missing values
(df.isnull().sum()/ len(df)* 100).sort_values(ascending=False)
```

```
ca          66.413043
thal        52.826087
slope       33.586957
fbs          9.782609
oldpeak      6.739130
trestbps     6.413043
exang        5.978261
thalch       5.978261
chol         3.260870
restecg      0.217391
cp           0.000000
dataset      0.000000
id           0.000000
age          0.000000
sex          0.000000
num          0.000000
dtype: float64
```

Figure 18: Checking missing values

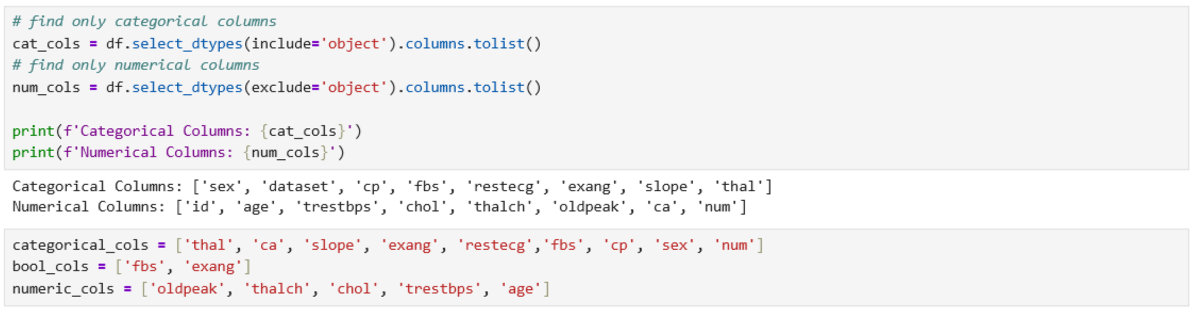
4.9.16 Computing columns with missing values



```
[169]: missing_data_cols = df.isnull().sum()[df.isnull().sum() > 0].index.tolist()
       missing_data_cols
```

```
[169]: ['trestbps',
        'chol',
        'fbs',
        'restecg',
        'thalch',
        'exang',
        'oldpeak',
        'slope',
        'ca',
        'thal']
```

4.9.17 Separating categorical and numeric columns and placing them in lists

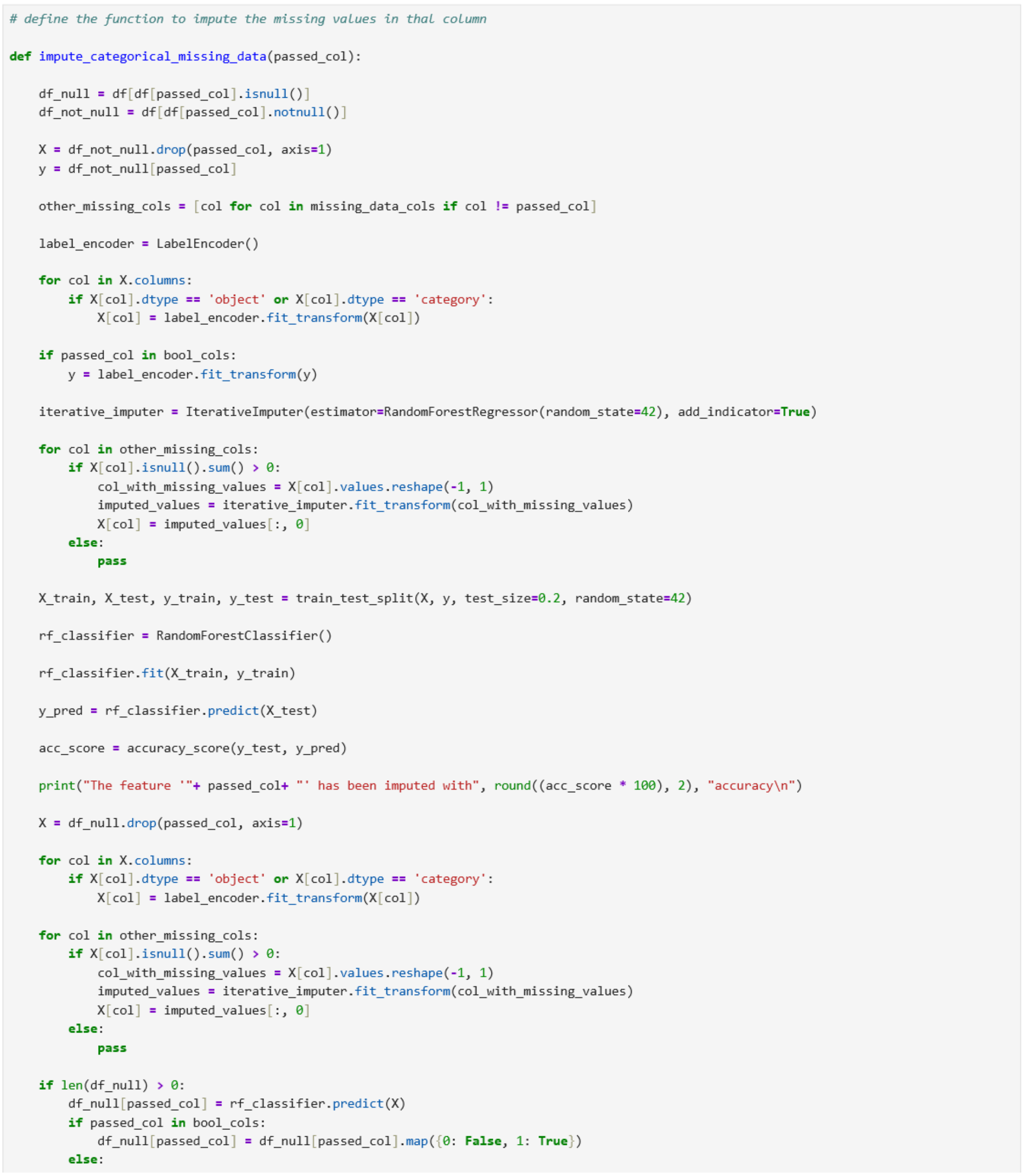


```
# find only categorical columns
cat_cols = df.select_dtypes(include='object').columns.tolist()
# find only numerical columns
num_cols = df.select_dtypes(exclude='object').columns.tolist()

print(f'Categorical Columns: {cat_cols}')
print(f'Numerical Columns: {num_cols}')
```

```
Categorical Columns: ['sex', 'dataset', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'thal']
Numerical Columns: ['id', 'age', 'trestbps', 'chol', 'thalch', 'oldpeak', 'ca', 'num']
```

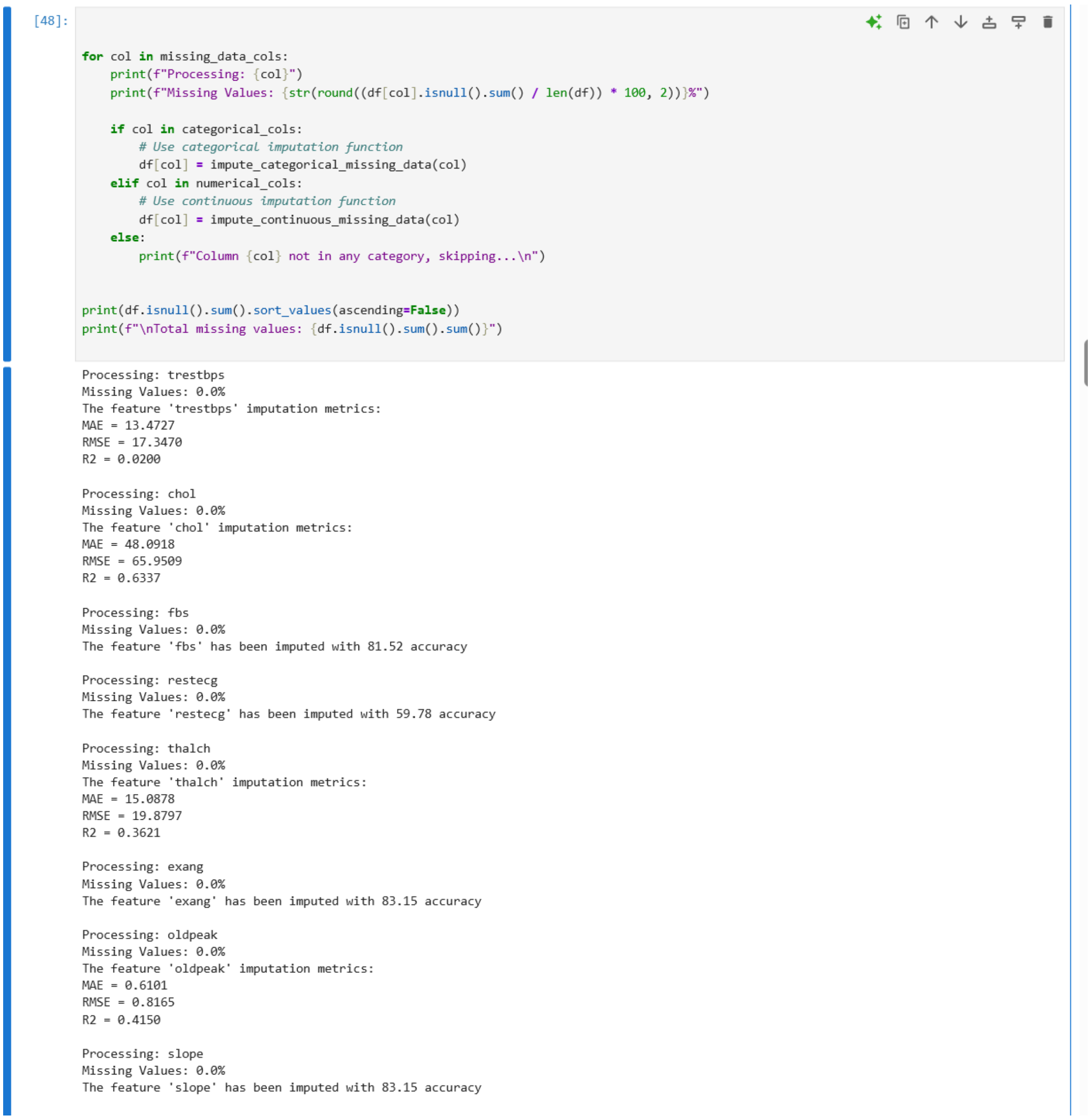
```
categorical_cols = ['thal', 'ca', 'slope', 'exang', 'restecg','fbs', 'cp', 'sex', 'num']
bool_cols = ['fbs', 'exang']
numeric_cols = ['oldpeak', 'thalch', 'chol', 'trestbps', 'age']
```

4.9.18 Define function to impute missing values



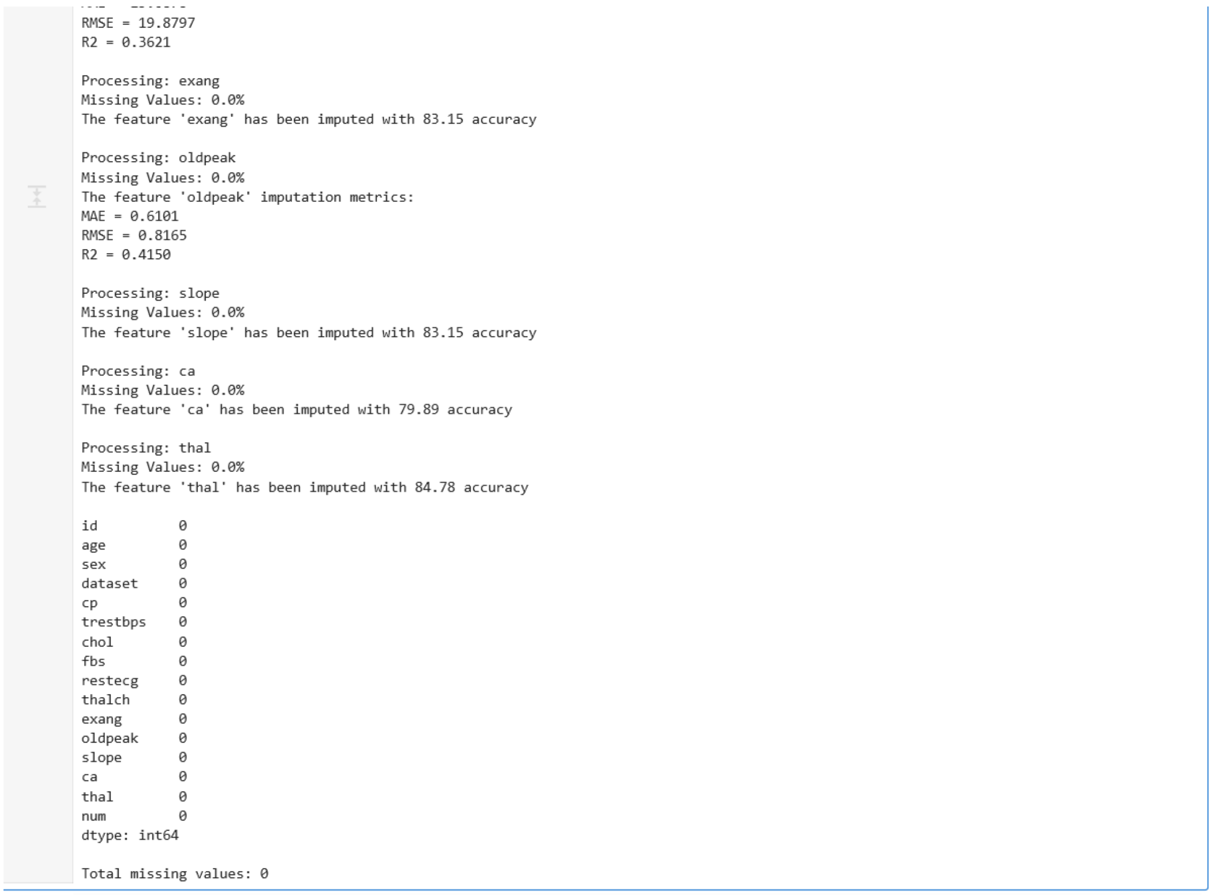
```
# define the function to impute the missing values in that column

def impute_categorical_missing_data(passed_col):

    df_null = df[df[passed_col].isnull()]
    df_not_null = df[df[passed_col].notnull()]

    X = df_not_null.drop(passed_col, axis=1)
    y = df_not_null[passed_col]

    other_missing_cols = [col for col in missing_data_cols if col != passed_col]

    label_encoder = LabelEncoder()

    for col in X.columns:
        if X[col].dtype == 'object' or X[col].dtype == 'category':
            X[col] = label_encoder.fit_transform(X[col])

    if passed_col in bool_cols:
        y = label_encoder.fit_transform(y)

    iterative_imputer = IterativeImputer(estimator=RandomForestRegressor(random_state=42), add_indicator=True)

    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            col_with_missing_values = X[col].values.reshape(-1, 1)
            imputed_values = iterative_imputer.fit_transform(col_with_missing_values)
            X[col] = imputed_values[:, 0]
        else:
            pass

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    rf_classifier = RandomForestClassifier()

    rf_classifier.fit(X_train, y_train)

    y_pred = rf_classifier.predict(X_test)

    acc_score = accuracy_score(y_test, y_pred)

    print("The feature '"+ passed_col+ "' has been imputed with", round((acc_score * 100), 2), "accuracy\n")

    X = df_null.drop(passed_col, axis=1)

    for col in X.columns:
        if X[col].dtype == 'object' or X[col].dtype == 'category':
            X[col] = label_encoder.fit_transform(X[col])

    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            col_with_missing_values = X[col].values.reshape(-1, 1)
            imputed_values = iterative_imputer.fit_transform(col_with_missing_values)
            X[col] = imputed_values[:, 0]
        else:
            pass

    if len(df_null) > 0:
        df_null[passed_col] = rf_classifier.predict(X)
        if passed_col in bool_cols:
            df_null[passed_col] = df_null[passed_col].map({0: False, 1: True})
        else:
```

4.9.19 Runing imputation of missing values



```
[48]:
for col in missing_data_cols:
    print(f"Processing: {col}")
    print(f"Missing Values: {str(round((df[col].isnull().sum() / len(df)) * 100, 2))}%")

    if col in categorical_cols:
        # Use categorical imputation function
        df[col] = impute_categorical_missing_data(col)
    elif col in numerical_cols:
        # Use continuous imputation function
        df[col] = impute_continuous_missing_data(col)
    else:
        print(f"Column {col} not in any category, skipping...\n")


print(df.isnull().sum().sort_values(ascending=False))
print(f"\nTotal missing values: {df.isnull().sum().sum()}")
```

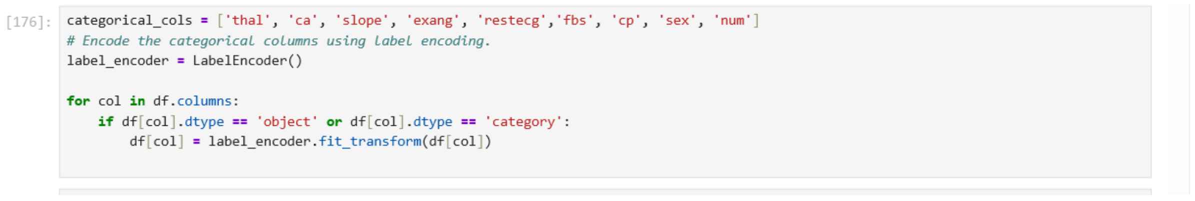
```
Processing: trestbps
Missing Values: 0.0%
The feature 'trestbps' imputation metrics:
MAE = 13.4727
RMSE = 17.3470
R2 = 0.0200

Processing: chol
Missing Values: 0.0%
The feature 'chol' imputation metrics:
MAE = 48.0918
RMSE = 65.9509
R2 = 0.6337

Processing: fbs
Missing Values: 0.0%
The feature 'fbs' has been imputed with 81.52 accuracy

Processing: restecg
Missing Values: 0.0%
The feature 'restecg' has been imputed with 59.78 accuracy

Processing: thalch
Missing Values: 0.0%
The feature 'thalch' imputation metrics:
MAE = 15.0878
RMSE = 19.8797
R2 = 0.3621

Processing: exang
Missing Values: 0.0%
The feature 'exang' has been imputed with 83.15 accuracy

Processing: oldpeak
Missing Values: 0.0%
The feature 'oldpeak' imputation metrics:
MAE = 0.6101
RMSE = 0.8165
R2 = 0.4150

Processing: slope
Missing Values: 0.0%
The feature 'slope' has been imputed with 83.15 accuracy
```



```
RMSE = 19.8797
R2 = 0.3621

Processing: exang
Missing Values: 0.0%
The feature 'exang' has been imputed with 83.15 accuracy

Processing: oldpeak
Missing Values: 0.0%
The feature 'oldpeak' imputation metrics:
MAE = 0.6101
RMSE = 0.8165
R2 = 0.4150

Processing: slope
Missing Values: 0.0%
The feature 'slope' has been imputed with 83.15 accuracy

Processing: ca
Missing Values: 0.0%
The feature 'ca' has been imputed with 79.89 accuracy

Processing: thal
Missing Values: 0.0%
The feature 'thal' has been imputed with 84.78 accuracy

id          0
age         0
sex         0
dataset     0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalch      0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
num         0
dtype: int64

Total missing values: 0
```

4.9.20 Encode the categorical columns using label encoding.

Encodes categorical variables to numeric labels for modeling readiness.

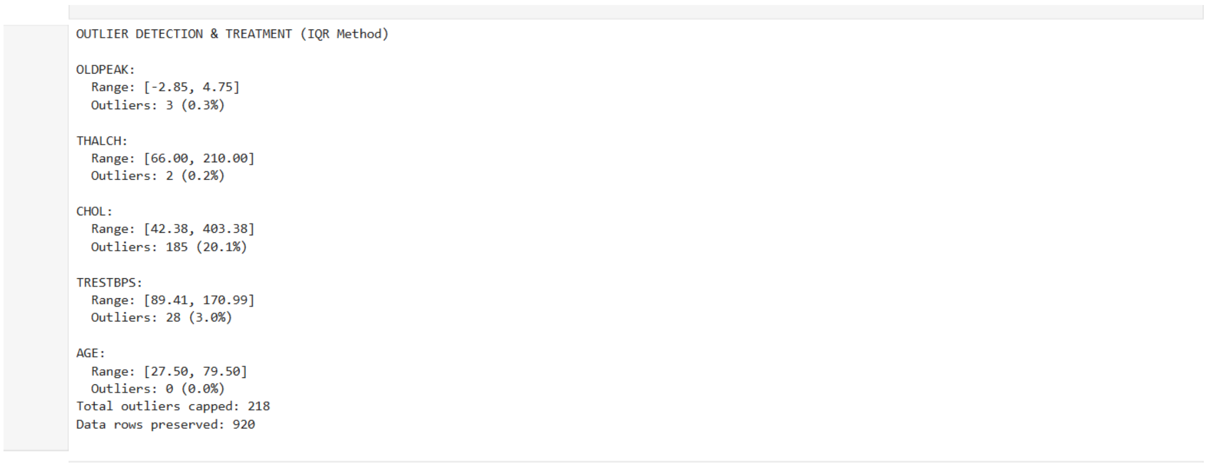


```
[176]: categorical_cols = ['thal', 'ca', 'slope', 'exang', 'restecg','fbs', 'cp', 'sex', 'num']
       # Encode the categorical columns using label encoding.
       label_encoder = LabelEncoder()

       for col in df.columns:
           if df[col].dtype == 'object' or df[col].dtype == 'category':
               df[col] = label_encoder.fit_transform(df[col])
```

Figure 24 Encode the categorical columns using label encoding.

4.9.21 IQR-Based Outlier Check

Computes the first (Q1) and third (Q3) quartiles and the interquartile range, then counts observations outside the 1.5×IQR bounds for each numeric column. The resulting outliers_count_specified shows how many potential outliers each numeric feature has.

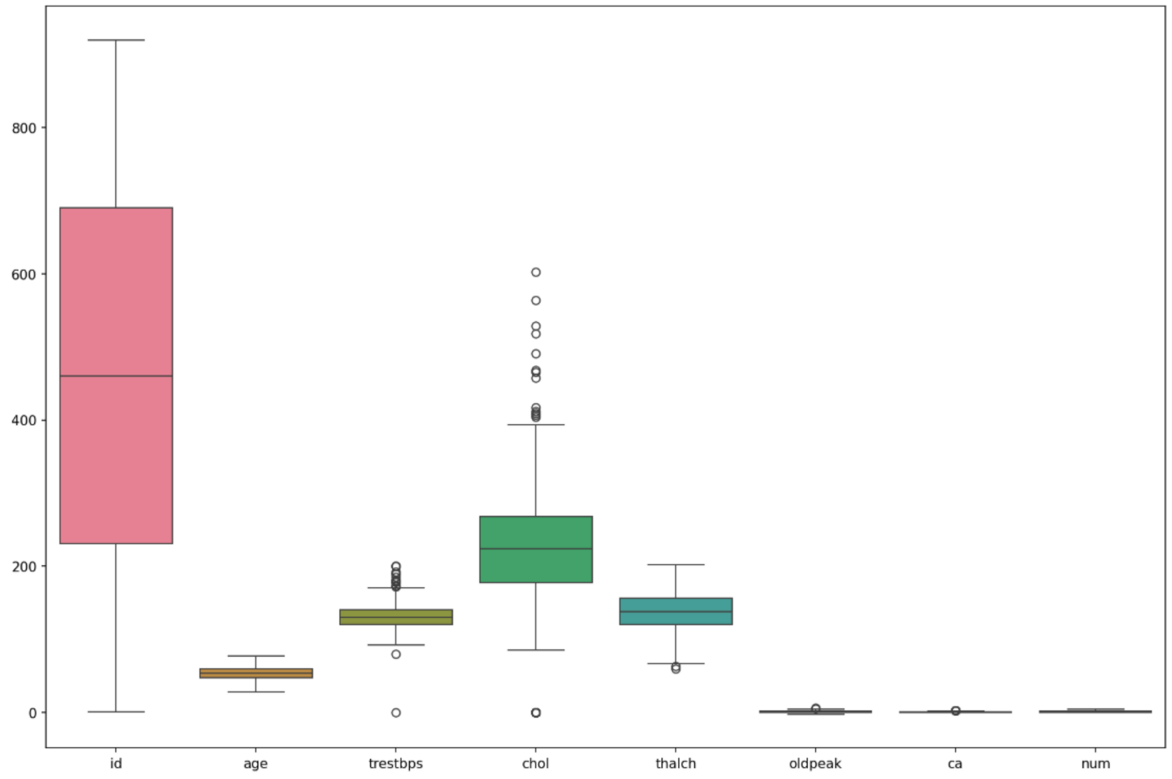


```
OUTLIER DETECTION & TREATMENT (IQR Method)

OLDPEAK:
  Range: [-2.85, 4.75]
  Outliers: 3 (0.3%)

THALCH:
  Range: [66.00, 210.00]
  Outliers: 2 (0.2%)

CHOL:
  Range: [42.38, 403.38]
  Outliers: 185 (20.1%)

TRESTBPS:
  Range: [89.41, 170.99]
  Outliers: 28 (3.0%)

AGE:
  Range: [27.50, 79.50]
  Outliers: 0 (0.0%)
Total outliers capped: 218
Data rows preserved: 920
```

Figure 19: Outlier check

4.9.22 Outlier boxplot

```
# PLot the boxplot to check the outliers
plt.figure(figsize=(15,10))
sns.boxplot(data=df[num_cols])
plt.show()
plt.suptitle('Boxplot of Continuous Features', fontsize=22)
```

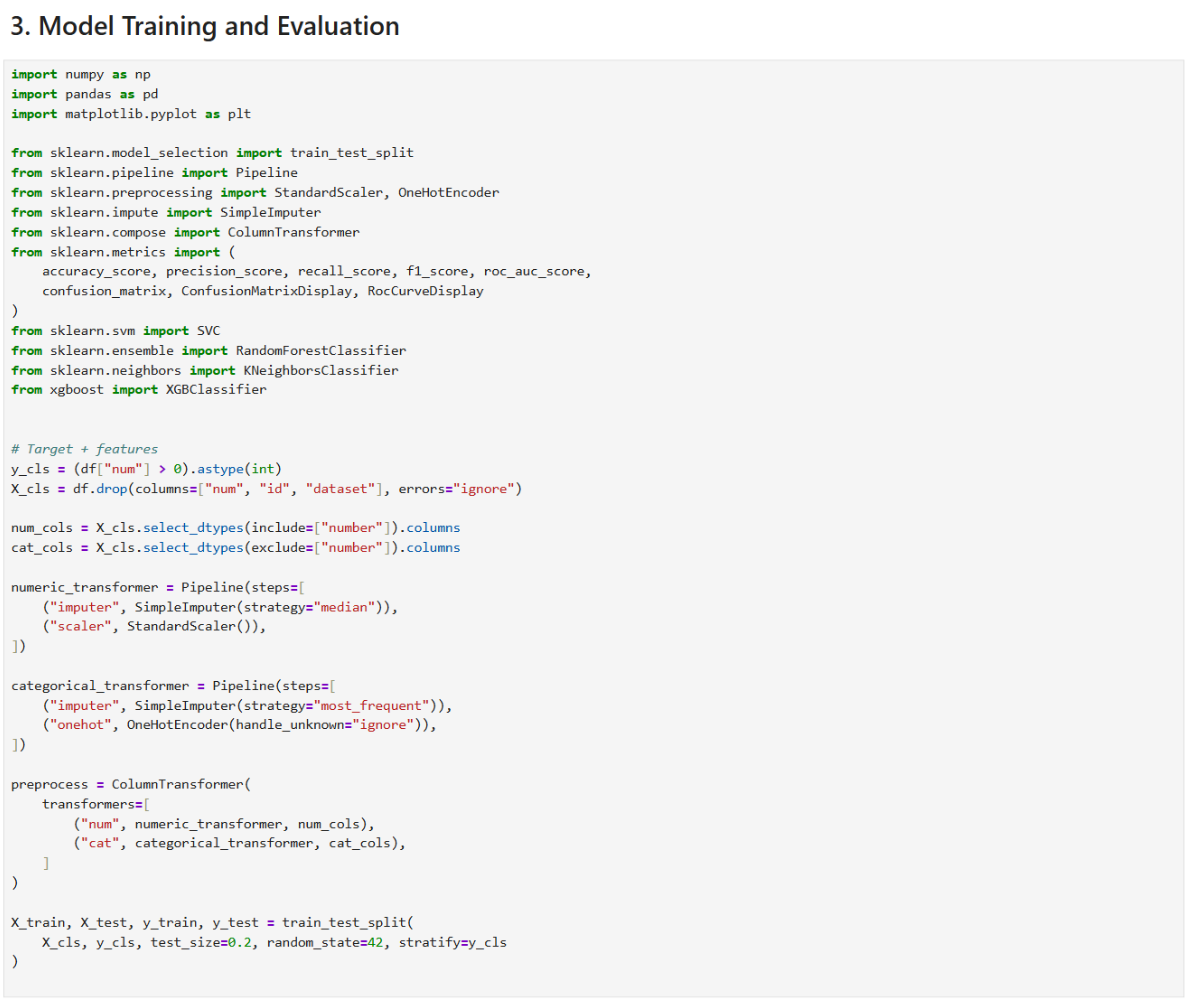


```
Text(0.5, 0.98, 'Boxplot of Continuous Features')
<Figure size 960x720 with 0 Axes>
```

4.9.23 Model Training and train test split

4.9.24 Preperation

This cell sets up the full machine learning workflow by creating the binary target (disease vs no disease), separating features into numeric and categorical columns, building preprocessing pipelines to impute missing values (median for numeric, most frequent for categorical), scale numeric features, one hot encode categorical features, and finally splitting the dataset into stratified train and test sets to keep the class balance consistent for fair model evaluation.



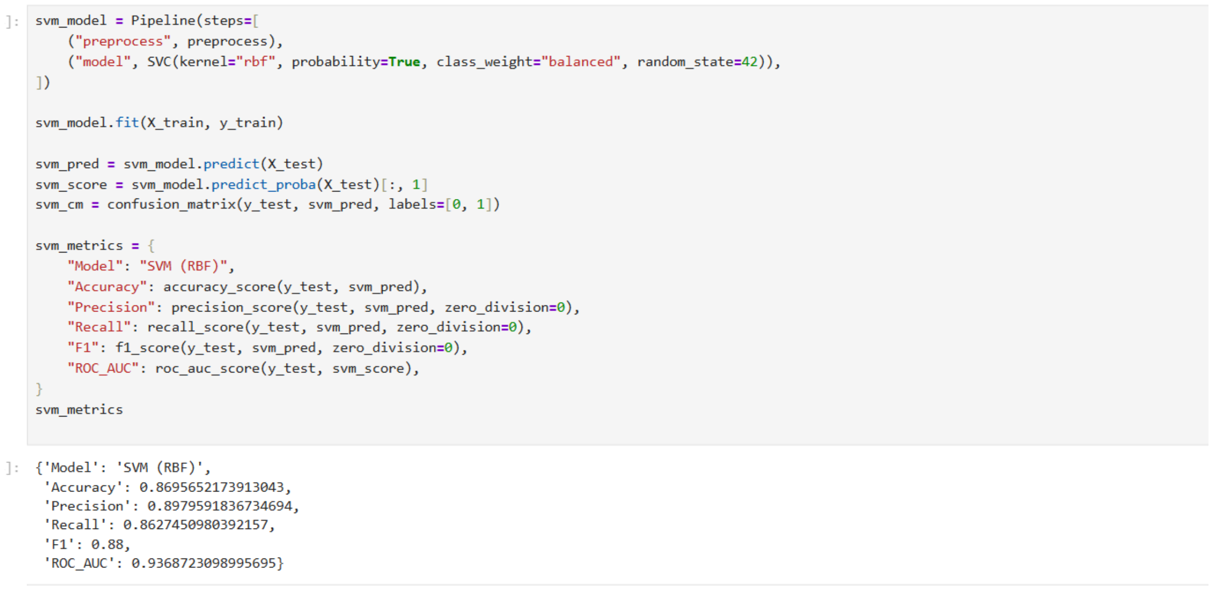
3. Model Training and Evaluation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
    confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay
)
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier


# Target + features
y_cls = (df["num"] > 0).astype(int)
X_cls = df.drop(columns=["num", "id", "dataset"], errors="ignore")

num_cols = X_cls.select_dtypes(include=["number"]).columns
cat_cols = X_cls.select_dtypes(exclude=["number"]).columns

numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler()),
])

categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore")),
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, num_cols),
        ("cat", categorical_transformer, cat_cols),
    ]
)

X_train, X_test, y_train, y_test = train_test_split(
    X_cls, y_cls, test_size=0.2, random_state=42, stratify=y_cls
)
```

Figure 20:pipe line setup

4.9.25 SVM Classification (SVC)

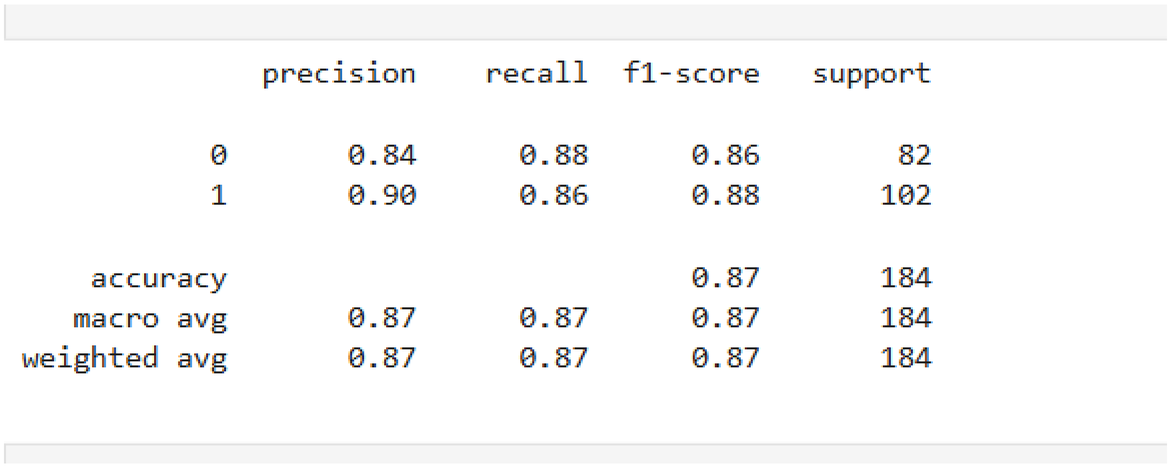
Splits features and target into training and testing sets for model evaluation. Runs grid search cross-validation to tune hyperparameters and select the best model. Trains an SVM classifier pipeline and evaluates predictive performance.


```
]: svm_model = Pipeline(steps=[
       ("preprocess", preprocess),
       ("model", SVC(kernel="rbf", probability=True, class_weight="balanced", random_state=42)),
   ])

   svm_model.fit(X_train, y_train)

   svm_pred = svm_model.predict(X_test)
   svm_score = svm_model.predict_proba(X_test)[:, 1]
   svm_cm = confusion_matrix(y_test, svm_pred, labels=[0, 1])

   svm_metrics = {
       "Model": "SVM (RBF)",
       "Accuracy": accuracy_score(y_test, svm_pred),
       "Precision": precision_score(y_test, svm_pred, zero_division=0),
       "Recall": recall_score(y_test, svm_pred, zero_division=0),
       "F1": f1_score(y_test, svm_pred, zero_division=0),
       "ROC_AUC": roc_auc_score(y_test, svm_score),
   }
   svm_metrics
```

```
]: {'Model': 'SVM (RBF)',
    'Accuracy': 0.8695652173913043,
    'Precision': 0.8979591836734694,
    'Recall': 0.8627450980392157,
    'F1': 0.88,
    'ROC_AUC': 0.9368723098995695}
```

Figure 21: SVM

4.9.26 Classification report



```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86        82
           1       0.90      0.86      0.88       102

    accuracy                           0.87       184
   macro avg       0.87      0.87      0.87       184
weighted avg       0.87      0.87      0.87       184
```

Figure 22: clasification report

4.9.27 Confusion Matrix

Confusion Matrix Numbers are 2×2 representation that summarize the SVM model's performance on the test patients as it compares the actual labels (rows) with the predicted labels (columns).

	Predicted: No Disease	**Predicted: Disease**
Actual: No Disease	**66 (TN)**	**16 (FP)**
Actual: Disease	**10 (FN)**	**92 (TP)**

True Negatives (TN) = 66: These are patients who did not have heart disease, and the model accurately predicted them as "No Disease."

False Positives (FP) = 16: This refers to the cases where the patient did not have heart disease but the model wrongly predicted them as "Disease." Such situations can cause unnecessary worry and lead to extra testing.

False Negatives (FN) = 10: The patients with heart disease who the model classified as "No Disease" are the ones suffering from the most serious error in medical screening because it is a case of missing a real one.

True Positives (TP) = 92: They are the patients with heart disease who were accurately classified by the model as "Disease," which is to say the model has succeeded in detecting most cases of the disease..

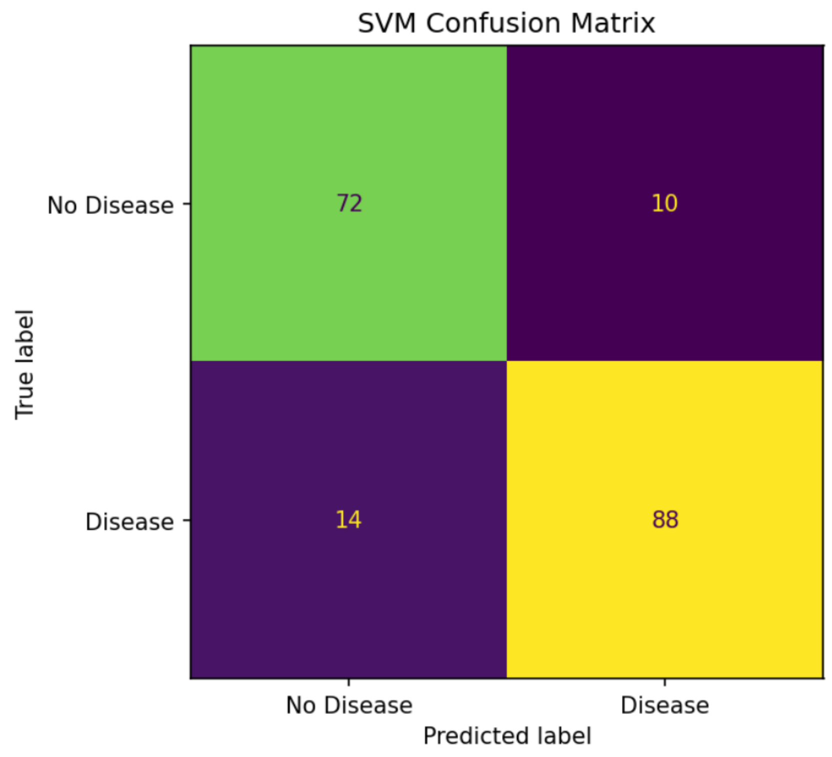
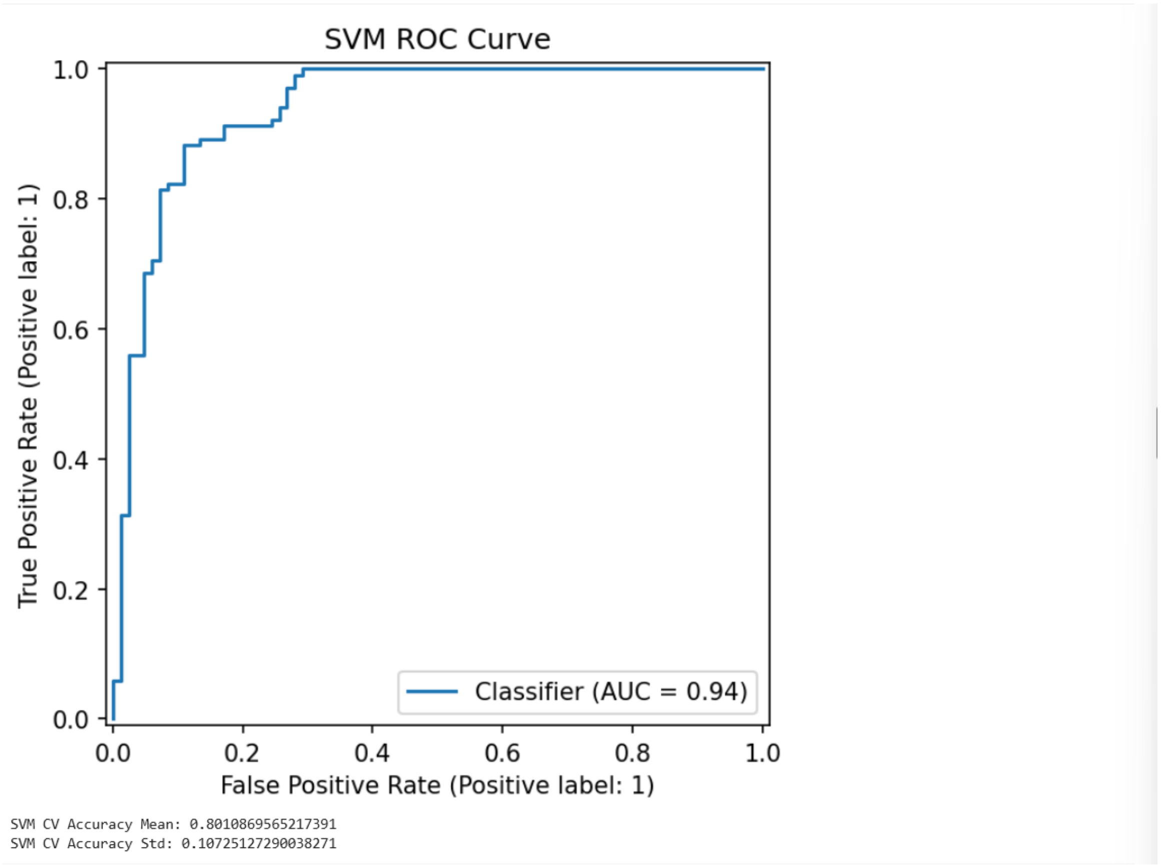


Figure 23: Confusion matrix

4.9.28 ROC curve

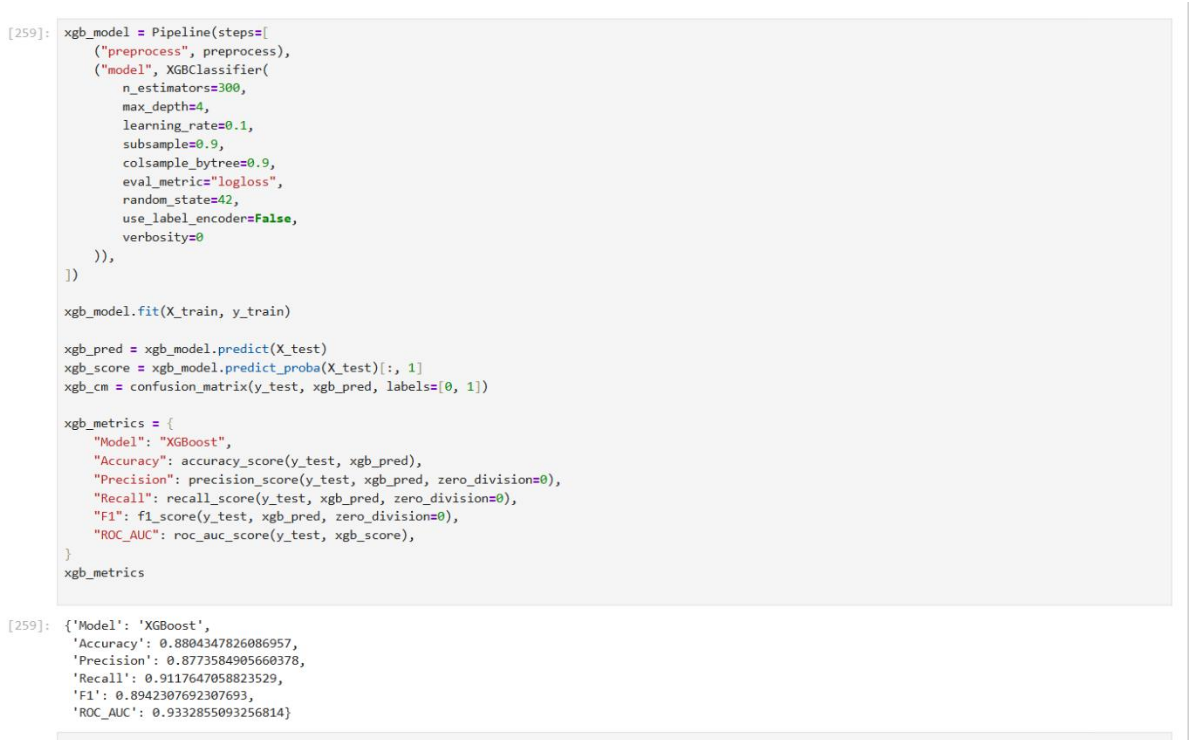


```
SVM CV Accuracy Mean: 0.8010869565217391
SVM CV Accuracy Std: 0.10725127290038271
```

Figure 24: Roc curve

4.9.29 XGBoost

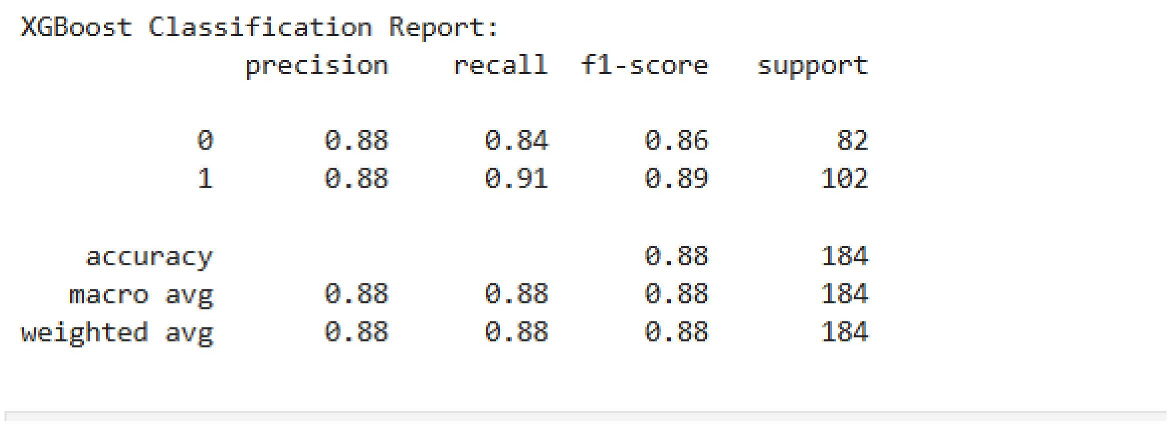
Splits features and target into training and testing sets for model evaluation. Fits an XGBoost classifier with tuned parameters and reports performance.



```
[259]: xgb_model = Pipeline(steps=[
           ("preprocess", preprocess),
           ("model", XGBClassifier(
               n_estimators=300,
               max_depth=4,
               learning_rate=0.1,
               subsample=0.9,
               colsample_bytree=0.9,
               eval_metric="logloss",
               random_state=42,
               use_label_encoder=False,
               verbosity=0
           )),
       ])

       xgb_model.fit(X_train, y_train)

       xgb_pred = xgb_model.predict(X_test)
       xgb_score = xgb_model.predict_proba(X_test)[:, 1]
       xgb_cm = confusion_matrix(y_test, xgb_pred, labels=[0, 1])

       xgb_metrics = {
           "Model": "XGBoost",
           "Accuracy": accuracy_score(y_test, xgb_pred),
           "Precision": precision_score(y_test, xgb_pred, zero_division=0),
           "Recall": recall_score(y_test, xgb_pred, zero_division=0),
           "F1": f1_score(y_test, xgb_pred, zero_division=0),
           "ROC_AUC": roc_auc_score(y_test, xgb_score),
       }
       xgb_metrics

[259]: {'Model': 'XGBoost',
        'Accuracy': 0.8804347826086957,
        'Precision': 0.8773584905660378,
        'Recall': 0.9117647058823529,
        'F1': 0.8942307692307693,
        'ROC_AUC': 0.9332855093256814}
```

Figure 25: XGBoost

4.9.30 Classification report



```
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.84      0.86        82
           1       0.88      0.91      0.89       102

    accuracy                           0.88       184
   macro avg       0.88      0.88      0.88       184
weighted avg       0.88      0.88      0.88       184
```

Figure 26: classification report

4.9.31 Confusion matrix

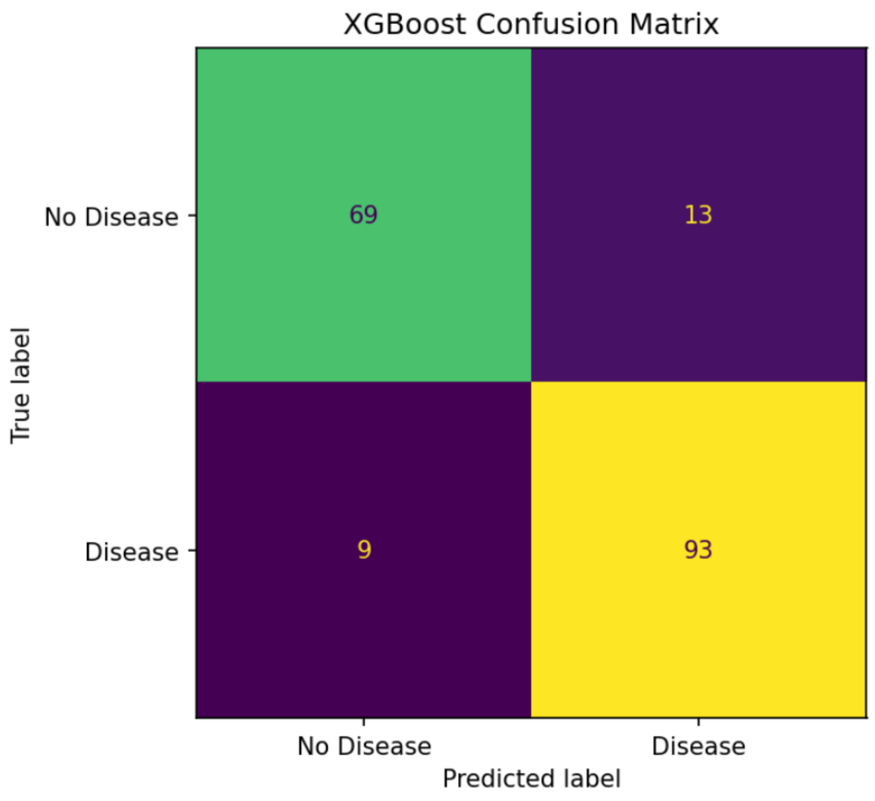
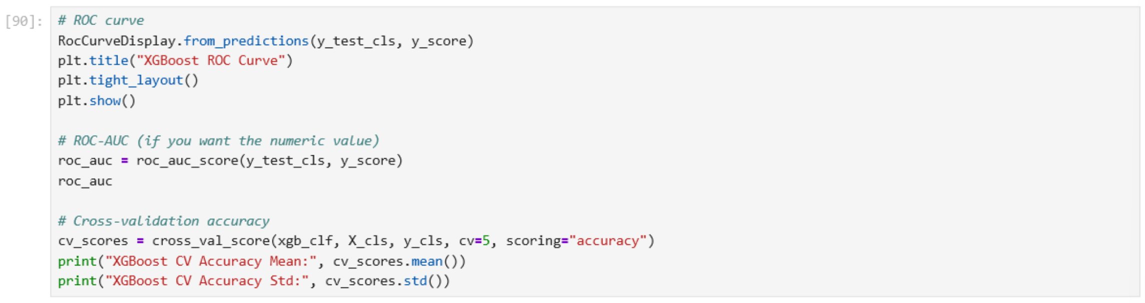


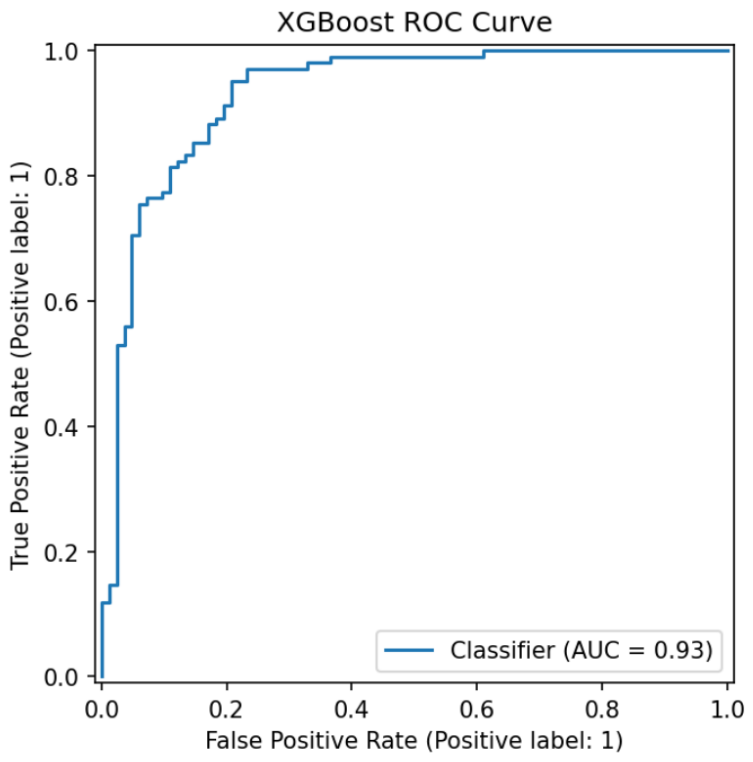
Figure 27: Confusion matrix

4.9.32 ROC curve

Plots ROC curves and computes AUC for threshold-free performance.



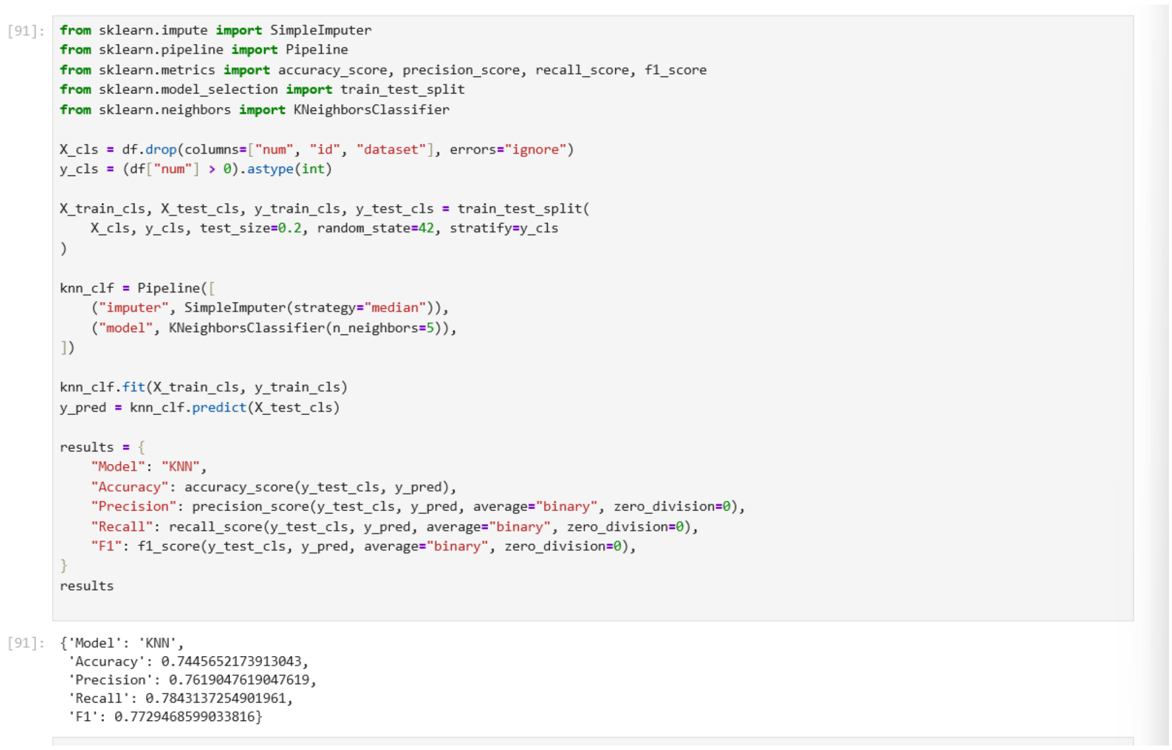
```
[90]: # ROC curve
      RocCurveDisplay.from_predictions(y_test_cls, y_score)
      plt.title("XGBoost ROC Curve")
      plt.tight_layout()
      plt.show()

      # ROC-AUC (if you want the numeric value)
      roc_auc = roc_auc_score(y_test_cls, y_score)
      roc_auc

      # Cross-validation accuracy
      cv_scores = cross_val_score(xgb_clf, X_cls, y_cls, cv=5, scoring="accuracy")
      print("XGBoost CV Accuracy Mean:", cv_scores.mean())
      print("XGBoost CV Accuracy Std:", cv_scores.std())
```



```
XGBoost CV Accuracy Mean: 0.7684782608695652
XGBoost CV Accuracy Std: 0.08367447501703361
```

4.9.33 KNN

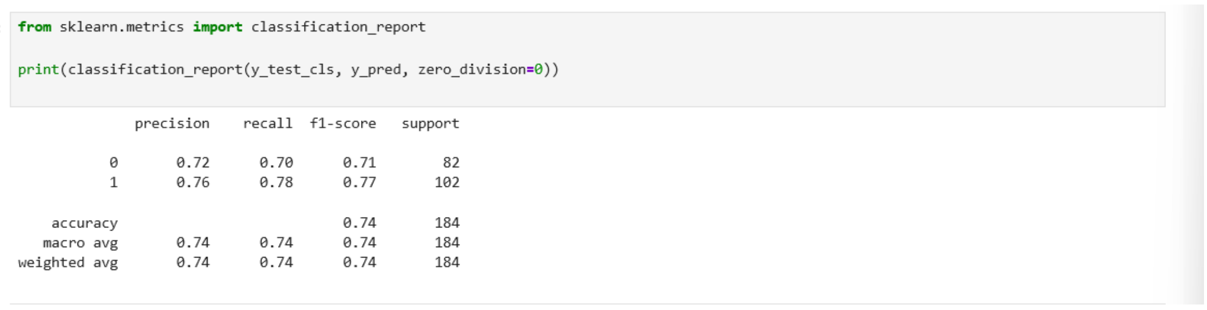
Splits features and target into training and testing sets for model evaluation. Fits a KNN classifier and measures accuracy, precision, recall, and F1.



```
[91]: from sklearn.impute import SimpleImputer
      from sklearn.pipeline import Pipeline
      from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
      from sklearn.model_selection import train_test_split
      from sklearn.neighbors import KNeighborsClassifier

      X_cls = df.drop(columns=["num", "id", "dataset"], errors="ignore")
      y_cls = (df["num"] > 0).astype(int)

      X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(
          X_cls, y_cls, test_size=0.2, random_state=42, stratify=y_cls
      )

      knn_clf = Pipeline([
          ("imputer", SimpleImputer(strategy="median")),
          ("model", KNeighborsClassifier(n_neighbors=5)),
      ])

      knn_clf.fit(X_train_cls, y_train_cls)
      y_pred = knn_clf.predict(X_test_cls)

      results = {
          "Model": "KNN",
          "Accuracy": accuracy_score(y_test_cls, y_pred),
          "Precision": precision_score(y_test_cls, y_pred, average="binary", zero_division=0),
          "Recall": recall_score(y_test_cls, y_pred, average="binary", zero_division=0),
          "F1": f1_score(y_test_cls, y_pred, average="binary", zero_division=0),
      }
      results
```

```
[91]: {'Model': 'KNN',
       'Accuracy': 0.7445652173913043,
       'Precision': 0.7619047619047619,
       'Recall': 0.7843137254901961,
       'F1': 0.7729468599033816}
```

4.9.34 Classification report



```
from sklearn.metrics import classification_report

print(classification_report(y_test_cls, y_pred, zero_division=0))
```

```
              precision    recall  f1-score   support

           0       0.72      0.70      0.71        82
           1       0.76      0.78      0.77       102

    accuracy                           0.74       184
   macro avg       0.74      0.74      0.74       184
weighted avg       0.74      0.74      0.74       184
```

4.9.35 Confusion matrix

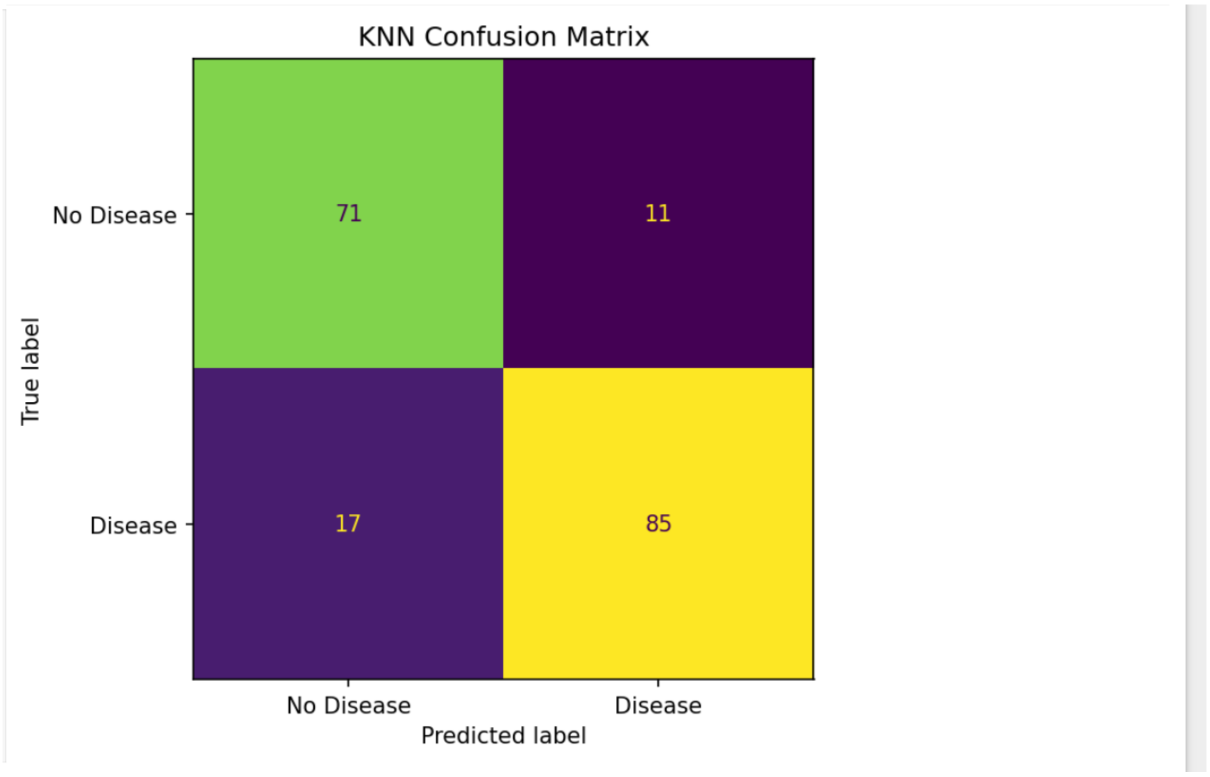


Figure 28: Confusion matrix

4.9.36 ROC Curve

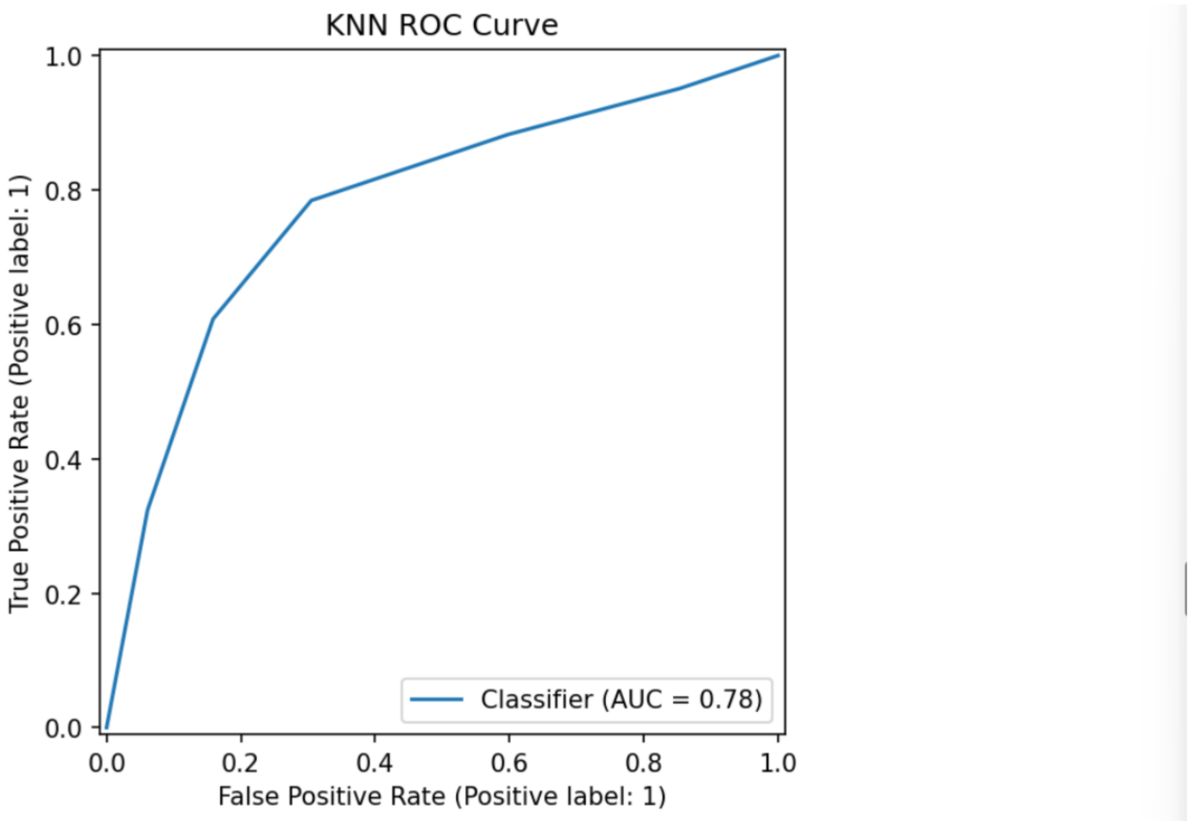
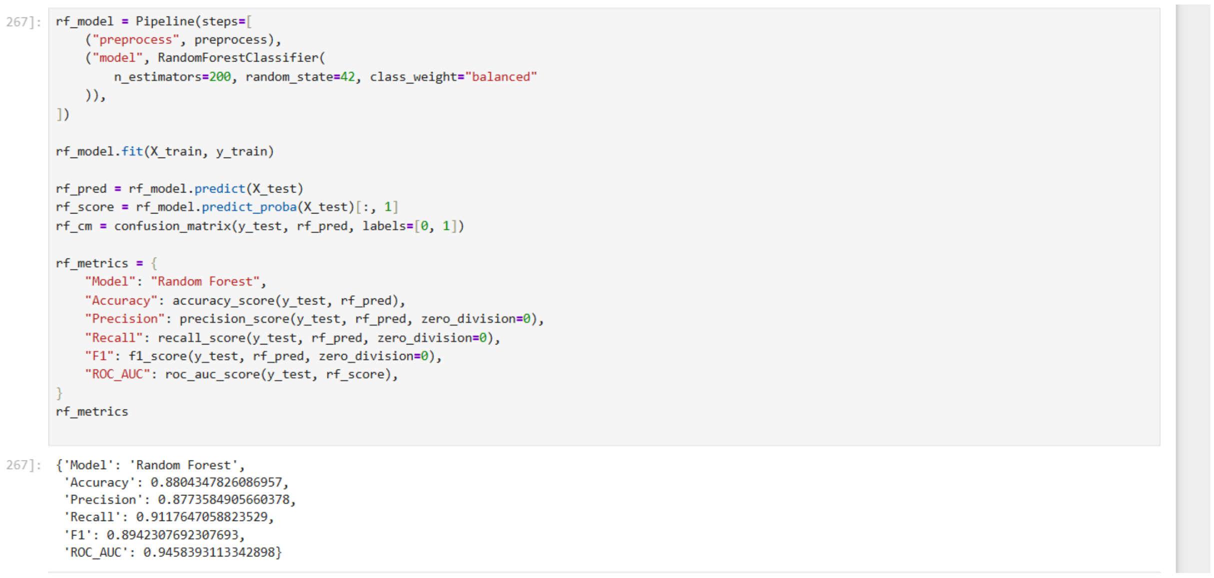


Figure 29 roc curve

4.9.37 Model: Random forest

Splits features and target into training and testing sets for model evaluation. Trains a class-weighted Random Forest classifier and evaluates metrics.

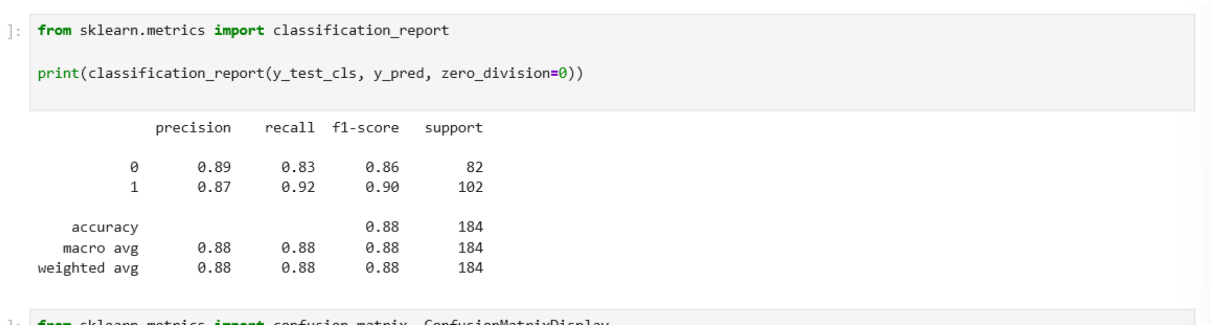


```
267]: rf_model = Pipeline(steps=[
          ("preprocess", preprocess),
          ("model", RandomForestClassifier(
              n_estimators=200, random_state=42, class_weight="balanced"
          )),
      ])

      rf_model.fit(X_train, y_train)

      rf_pred = rf_model.predict(X_test)
      rf_score = rf_model.predict_proba(X_test)[:, 1]
      rf_cm = confusion_matrix(y_test, rf_pred, labels=[0, 1])

      rf_metrics = {
          "Model": "Random Forest",
          "Accuracy": accuracy_score(y_test, rf_pred),
          "Precision": precision_score(y_test, rf_pred, zero_division=0),
          "Recall": recall_score(y_test, rf_pred, zero_division=0),
          "F1": f1_score(y_test, rf_pred, zero_division=0),
          "ROC_AUC": roc_auc_score(y_test, rf_score),
      }
      rf_metrics
```

```
267]: {'Model': 'Random Forest',
       'Accuracy': 0.8804347826086957,
       'Precision': 0.8773584905660378,
       'Recall': 0.9117647058823529,
       'F1': 0.8942307692307693,
       'ROC_AUC': 0.9458393113342898}
```

Figure 30 random forest

4.9.38 Classification report

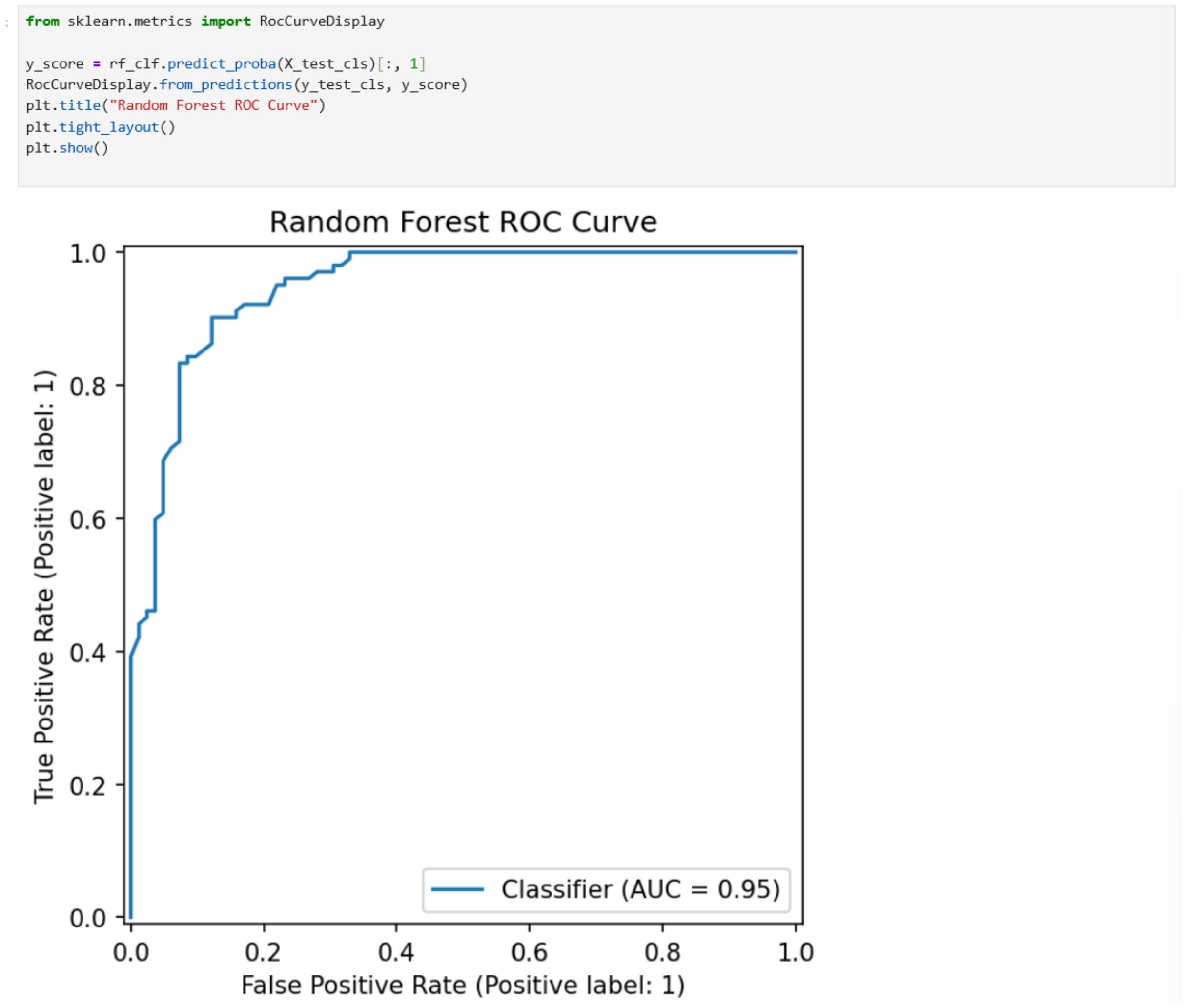


```
]: from sklearn.metrics import classification_report

   print(classification_report(y_test_cls, y_pred, zero_division=0))
```

```
              precision    recall  f1-score   support

           0       0.89      0.83      0.86        82
           1       0.87      0.92      0.90       102

    accuracy                           0.88       184
   macro avg       0.88      0.88      0.88       184
weighted avg       0.88      0.88      0.88       184
```

Figure 31: Classification report

4.9.39 ROC curve



```
from sklearn.metrics import RocCurveDisplay

y_score = rf_clf.predict_proba(X_test_cls)[:, 1]
RocCurveDisplay.from_predictions(y_test_cls, y_score)
plt.title("Random Forest ROC Curve")
plt.tight_layout()
plt.show()
```

Figure 32 ROC curve

4.9.40 Confusion matrix

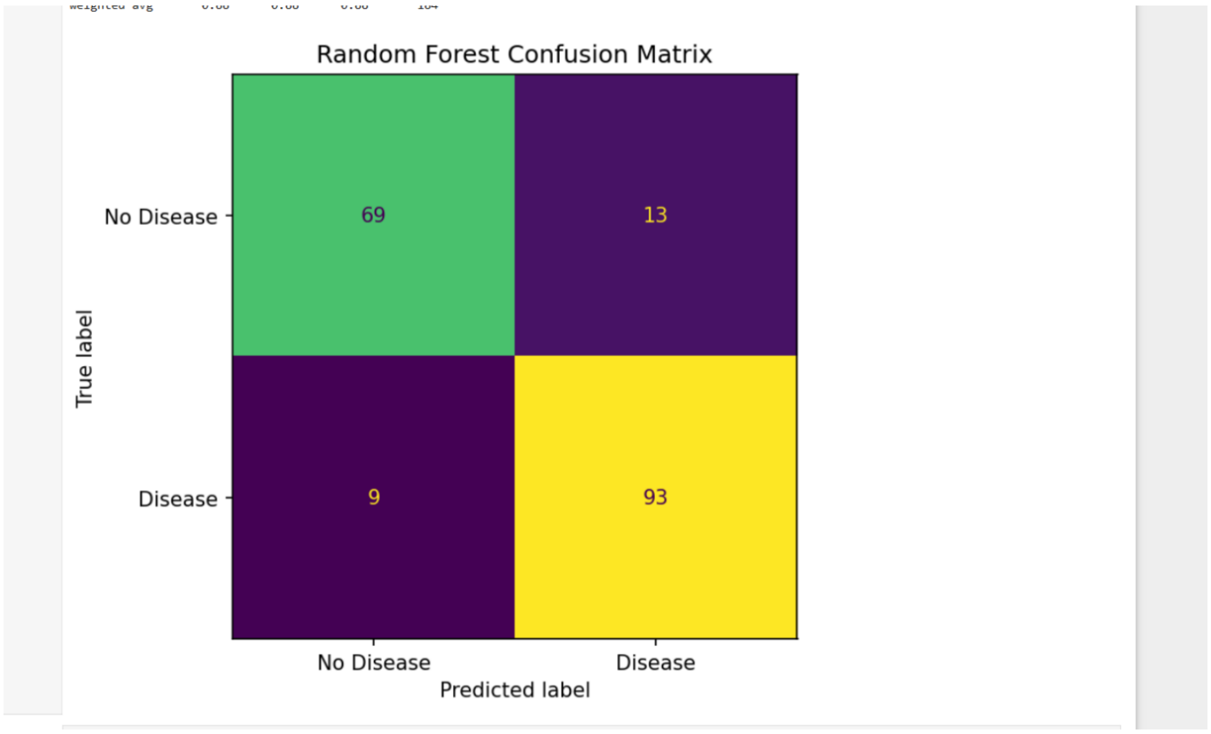
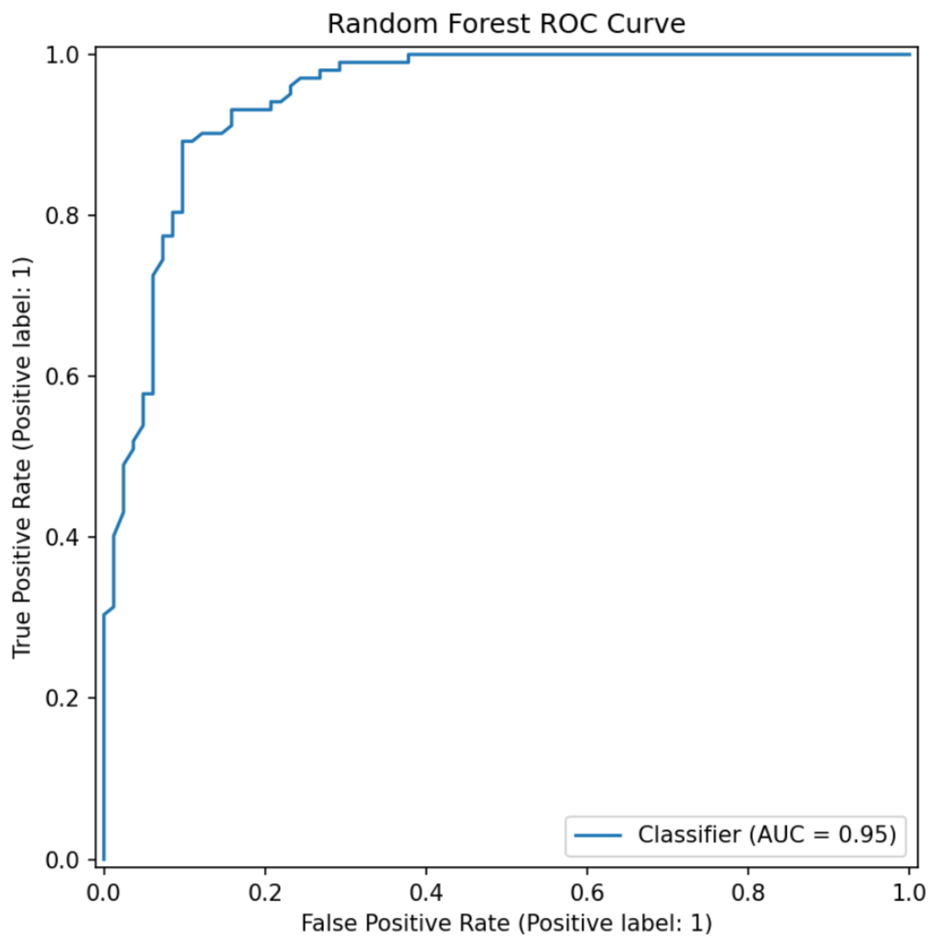


Figure 33 Confusion matrix

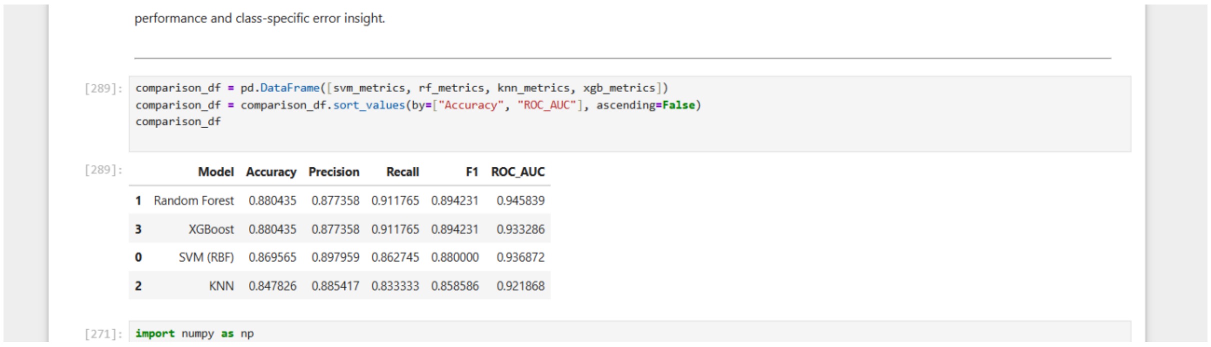
4.9.41 Random Forest ROC Curve



```
Random Forest CV Accuracy Mean: 0.8076
Random Forest CV Accuracy Std: 0.1005
```

Figure 34 Random Forest ROC Curve

4.9.42 Model comparison



performance and class-specific error insight.

```
[289]: comparison_df = pd.DataFrame([svm_metrics, rf_metrics, knn_metrics, xgb_metrics])
       comparison_df = comparison_df.sort_values(by=["Accuracy", "ROC_AUC"], ascending=False)
       comparison_df
```

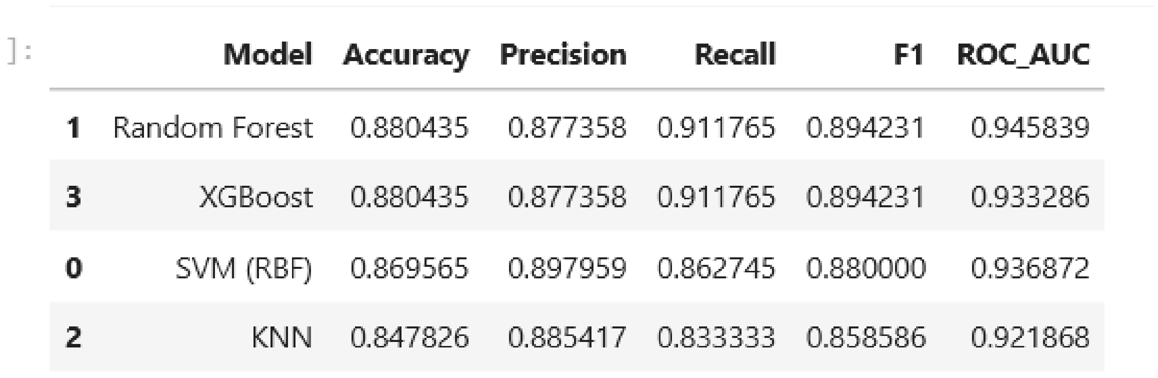
[289]:

	Model	Accuracy	Precision	Recall	F1	ROC_AUC
1	Random Forest	0.880435	0.877358	0.911765	0.894231	0.945839
3	XGBoost	0.880435	0.877358	0.911765	0.894231	0.933286
0	SVM (RBF)	0.869565	0.897959	0.862745	0.880000	0.936872
2	KNN	0.847826	0.885417	0.833333	0.858586	0.921868

```
[271]: import numpy as np
```

Figure 35 Model comparision

Splits features and target into training and testing sets for model evaluation. Trains an SVM classifier pipeline and evaluates predictive performance. Fits a KNN classifier and measures accuracy, precision, recall, and F1. Trains a class-weighted Random Forest classifier and evaluates metrics. Fits an XGBoost classifier with tuned parameters and reports performance. Generates confusion matrices to inspect class-wise errors.



]:

	Model	Accuracy	Precision	Recall	F1	ROC_AUC
1	Random Forest	0.880435	0.877358	0.911765	0.894231	0.945839
3	XGBoost	0.880435	0.877358	0.911765	0.894231	0.933286
0	SVM (RBF)	0.869565	0.897959	0.862745	0.880000	0.936872
2	KNN	0.847826	0.885417	0.833333	0.858586	0.921868

Figure 36: Model comparision

4.9.43 Bar chart comparison of model performance

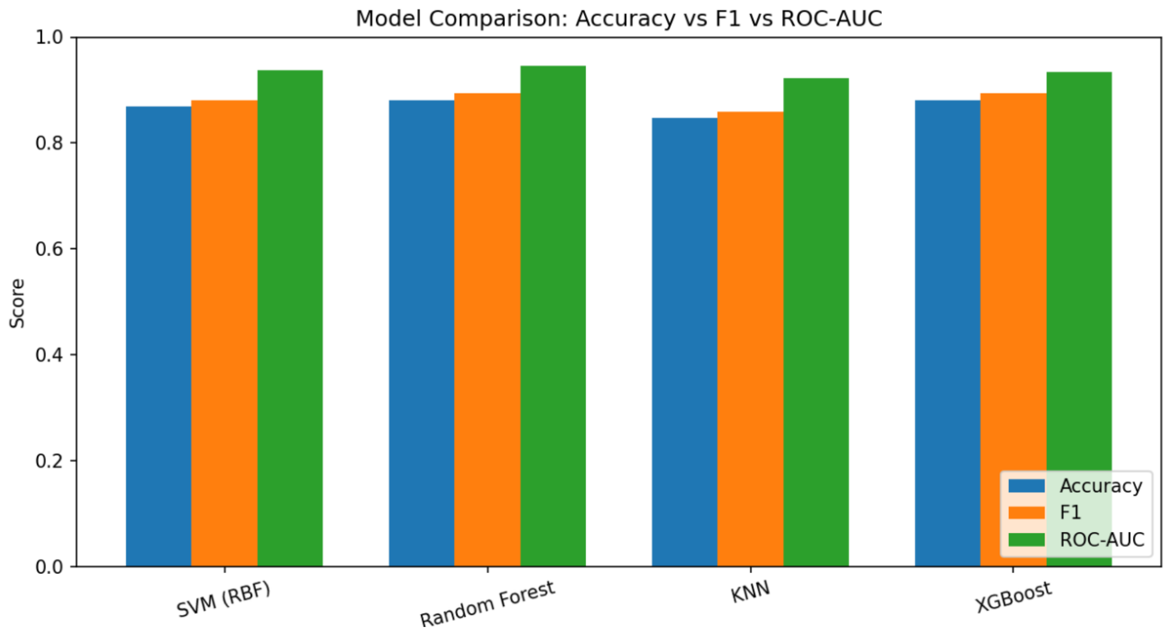


Figure 37: Visual comparison of model performance

4.9.44 Roc Curve comparison

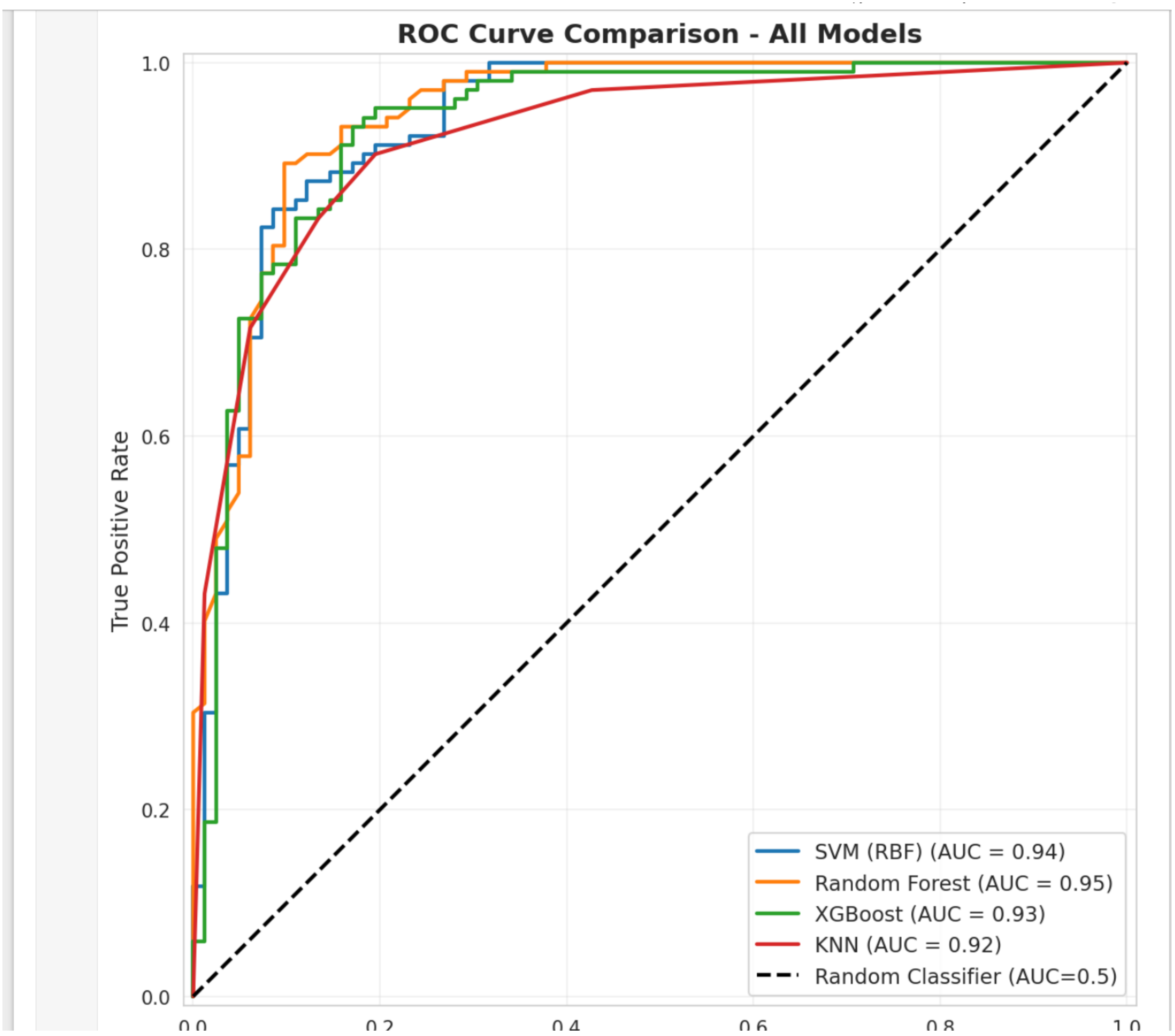


Figure 38 Roc Curve comparison

5 Conclusions

5.1 Analysis of the work done

This research piece showed the potential of machine learning for the early detection of heart disease via structured clinical information. With the worldwide distribution of heart disease and the necessity of fast diagnosis, the study considered the prediction of heart disease as a problem of supervised classification, scattering the model or rather its building upon labelled patient records and ultimately predicting the occurrence or non-occurrence of heart disease in new cases. The UCI Heart Disease dataset was chosen as it is practical, consisting of common clinical attributes that are associated with cardiovascular risk and thus permitting controlled research model development and comparison.

The whole development scheme followed the typical of machine learning: get the dataset, unit exploration data analysis, deal with missing values and other data quality problems, select features, teach models, and then finally, assess their performance with the help of several metrics. Many algorithms were executed and analyzed in order to know the most effective one for this medical prediction task. Support Vector Machine with an RBF kernel, Random Forest, K-Nearest Neighbors, and XGBoost formed the final model set as they are the most talked-about in the literature for heart disease prediction and also represent a wide range of instance-based learning, margin-based learning, ensemble bagging, and gradient boosting.

Moreover, the evaluation process stressed the use of accuracy metrics besides those applied in model training. In healthcare prediction tasks, relying on accuracy only can give the wrong impression since the actual clinical situations do not have the same impact of false negatives and false positives.

Model	**Accuracy**	**Precision**	**Recall**	**F1**	**ROC-AUC**	**Model**
Random Forest	0.8804	0.8774	0.9118	0.8942	0.9458	Random Forest
XGBoost	0.8804	0.8774	0.9118	0.8942	0.9333	XGBoost
SVM	0.8696	0.8980	0.8627	0.8800	0.9369	SVM (RBF)
KNN	0.8478	0.8854	0.8333	0.8586	0.9219	KNN

By these outcomes, the Random Forest method is by far the best choice for this project. It has the highest value of the ROC-AUC metric (0.9458), highest accuracy (0.8804) and an excellent trade-off between precision (0.8774) and recall (0.9118), thus leading to the highest F1-score (0.8942) as well. This is such an important factor for the healthcare sector as a model with a high recall will always put the patient who has the sickness at risk of being labeled healthy, even though this situation might sometimes result in more false positives. The large ROC-AUC also shows that Random Forest distinguishes very well between the disease and no-disease classes at different decision thresholds, which is a plus if the model ends up for risk scoring rather than strict yes or no classification.

XGBoost is a very close rival and it ties Random Forest in the accuracy (0.8804), precision (0.8774), recall (0.9118), and F1-score (0.8942) metrics, which indicates that boosting-based ensemble learning can be really productive with this dataset. Nonetheless, its ROC-AUC (0.9333) is below that of Random Forest, which points to somewhat lesser overall separability across thresholds and possibly further tuning may lead it to its optimum performance as previous studies frequently underline the advantages of optimization in boosting models (Mir, 2024).

The SVM (RBF) model performance is almost equal to that of the ensemble models, yet it is slightly ahead of them on the precision scale (0.8980). This indicates that it is the choice for applications where the greatest reduction of false alarms is needed. Its recall (0.8627), however, does not meet that of the leading models leading to more true disease cases being missed, and this trade-off is very important considering the main goal is early detection. KNN, even though it is still performing adequately, shows the lowest overall results in this case (accuracy 0.8478, recall 0.8333, F1 0.8586), which implies that it is more influenced by the feature space structure and may require more tuning or feature engineering to be able to compete with the other methods.

In Conclusion, the entire project through this research proves that the usage of machine learning can greatly assist predicting heart disease by just using common clinical variables, where the performance of ensemble models has the strongest hold while SVM has a competitive edge in precision.

6 References

Abdellatif, A. et al., 2022. *Improving the Heart Disease Detection and Patients' Survival Using Supervised Infinite Feature Selection and Improved Weighted Random Forest.* [Online]
Available at: https://ieeexplore.ieee.org/document/9802107

Alotaibi. , F. S., 2019. *Implementation of Machine Learning Model to Predict Heart Failure Disease.* [Online]
Available at: https://www.proquest.com/openview/1d045d9c46ce2917e06efe7f3c92d5e5/1?pq-origsite=gscholar&cbl=5444811

Bhatt, C. M., 2023. *Effective Heart Disease Prediction Using Machine Learning Techniques.* [Online]
Available at: https://www.mdpi.com/1999-4893/16/2/88

Foresee medical, n.d. *AI in Healthcare.* [Online]
Available at: https://www.foreseemed.com/artificial-intelligence-in-healthcare

Jupyter, n.d. *Jupyter Notebook Documentation.* [Online]
Available at: https://jupyter-notebook.readthedocs.io/en/stable/

Mir, A., 2024. *A novel approach for the effective prediction of cardiovascular disease using applied artificial intelligence techniques.* [Online]
Available at: https://www.researchgate.net/publication/382180707_A_novel_approach_for_the_effective_prediction_of_cardiovascular_disease_using_applied_artificial_intelligence_techniques#:~:text=learning%20classifiers%20were%20applied%20out,cardiovascular%20disease%20at

Neptune.ai, 2025. *Performance Metrics in Machine Learning [Complete Guide].* [Online]
Available at: https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide

Scikitlearn, n.d. *1.4. Support Vector Machines.* [Online]
Available at: https://scikit-learn.org/stable/modules/svm.html

World Health Organization, 2025. *Cardiovascular diseases (CVDs).* [Online]
Available at: https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)

Yar Muhammad, M. T., 2020. *Early and accurate detection and diagnosis of heart disease using intelligent computational model.* [Online]
Available at: https://www.nature.com/articles/s41598-020-76635-9