Report on

Predictive Modelling of Heart Disease

By

Jitendra Singh Kushwah

Student of ITM Gwalior

Branch : CSE-DS

Batch : 2022-26 (expected)

Email: jeetu.singh4824@gmail.com

Contents

1.Introduction

Heart disease is one of the leading causes of death globally, with millions affected each year. Early and accurate diagnosis plays a crucial role in preventing fatal outcomes. With the growing availability of clinical data and the advancement of machine learning techniques, predictive models offer significant potential to support medical professionals.

In this project, we use a dataset containing demographic and clinical information such as age, sex, blood pressure, cholesterol levels, and ECG results. We aim to build and compare various machine learning models to classify patients as either having heart disease or not.

The dataset is preprocessed, visualized, and then fed into several classifiers including logistic regression, support vector machines, decision trees, ensemble models like XGBoost and LightGBM, and a neural network (MLP). The performance of each model is evaluated using metrics such as accuracy, precision, recall, F1-score, and AUC-ROC.

2.Dataset Description

The dataset used in this project is the **processed Cleveland Heart Disease dataset**, one of the most commonly referenced datasets for heart disease prediction, originally from the UCI Machine Learning Repository. It consists of **303 samples** and **14 attributes**, including both **numerical** and **categorical** clinical variables.

- **Numerical features :**
  age, trestbps (resting blood pressure), chol (cholesterol), thalach (max heart rate), oldpeak, ca (major vessels)
- **Categorical features :**
  sex, cp (chest pain type), fbs, restecg, exang, slope, thal
- The **target variable num** originally ranged from 0 to 4, indicating increasing severity of heart disease. For this study, it was **binarized**:
  - 0 → No heart disease
  - 1 → Presence of heart disease (values 1–4 grouped together)

This cleaned version of the dataset was used for all data exploration, preprocessing, and model training.

3.Data Cleaning and Preprocessing

The dataset initially included non-numeric placeholder values (such as '?') in some columns, most notably in 'ca' and 'thal'. These were not suitable for numerical analysis therefore to handle this:-

- All '?' values were replaced with NaN
- the columns 'ca' and 'thal' were explicitly converted to numeric using pd.to_numeric() with errors='coerce', which automatically set any unconvertible values to NaN.

3.1 Raw data sample

Top five data rows

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num	
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	1
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0

Bottom five data rows

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num	
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	2.0	0.0	7.0	1
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	2.0	2.0	7.0	1
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	2.0	1.0	7.0	1
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	2.0	1.0	3.0	1
302	38.0	1.0	3.0	138.0	175.0	0.0	0.0	173.0	0.0	0.0	1.0	NaN	3.0	0

4.Exploratory Data Analysis (EDA)

To gain insights into the dataset, exploratory data analysis was conducted on both **numerical** and **categorical** features. This step helps understand data distributions, detect anomalies, and identify patterns that may influence model performance.

4.1 Numerical Features Summary

The following numerical features were analyzed:
age, trestbps, chol, thalach, oldpeak, num, ca

Using the .describe() method, we generated basic statistical summaries such as **mean**, **standard deviation**, **minimum**, and **maximum** for each feature.

Numerical features description:

	age	trestbps	chol	thalach	oldpeak	num	ca
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	299.000000
mean	54.438944	131.689769	246.693069	149.607261	1.039604	0.458746	0.672241
std	9.038662	17.599748	51.776918	22.875003	1.161075	0.499120	0.937438
min	29.000000	94.000000	126.000000	71.000000	0.000000	0.000000	0.000000
25%	48.000000	120.000000	211.000000	133.500000	0.000000	0.000000	0.000000
50%	56.000000	130.000000	241.000000	153.000000	0.800000	0.000000	0.000000
75%	61.000000	140.000000	275.000000	166.000000	1.600000	1.000000	1.000000
max	77.000000	200.000000	564.000000	202.000000	6.200000	1.000000	3.000000

4.2 Categorical Features Summary

The dataset includes several categorical variables:
sex, cp, fbs, restecg, exang, slope, and thal

To understand the distribution of values in each categorical feature, the .describe() method was applied:

Categorical features description:

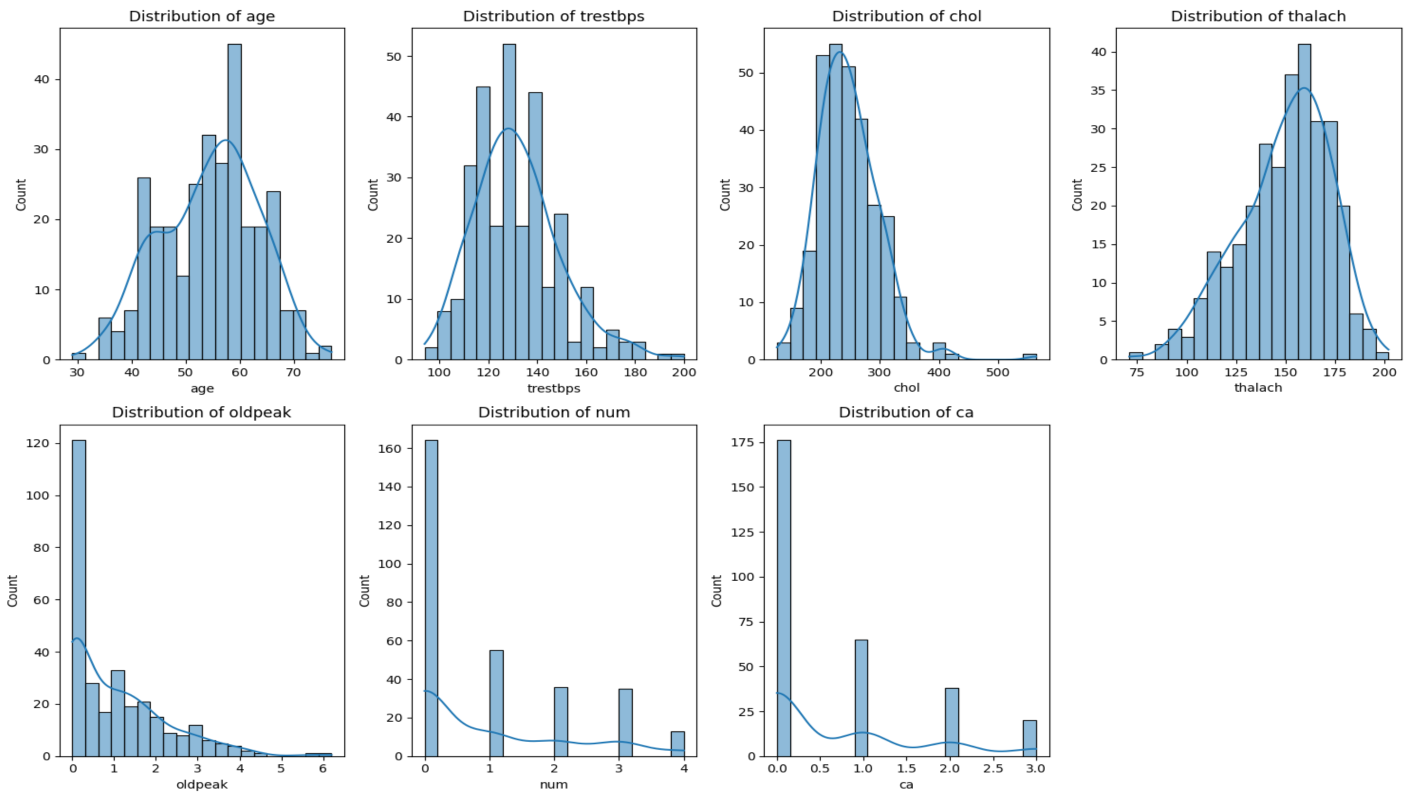
	sex	cp	fbs	restecg	exang	slope	thal
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	301.000000
mean	0.679868	3.158416	0.148515	0.990099	0.326733	1.600660	4.734219
std	0.467299	0.960126	0.356198	0.994971	0.469794	0.616226	1.939706
min	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	3.000000
25%	0.000000	3.000000	0.000000	0.000000	0.000000	1.000000	3.000000
50%	1.000000	3.000000	0.000000	1.000000	0.000000	2.000000	3.000000
75%	1.000000	4.000000	0.000000	2.000000	1.000000	2.000000	7.000000
max	1.000000	4.000000	1.000000	2.000000	1.000000	3.000000	7.000000

4.3 Numerical Feature Distributions

To better understand how the values in each numerical column are distributed, **histograms** were plotted using seaborn.histplot() with kernel density estimates (KDE). This visualizes not just the frequency of values, but also the smooth distribution shape.

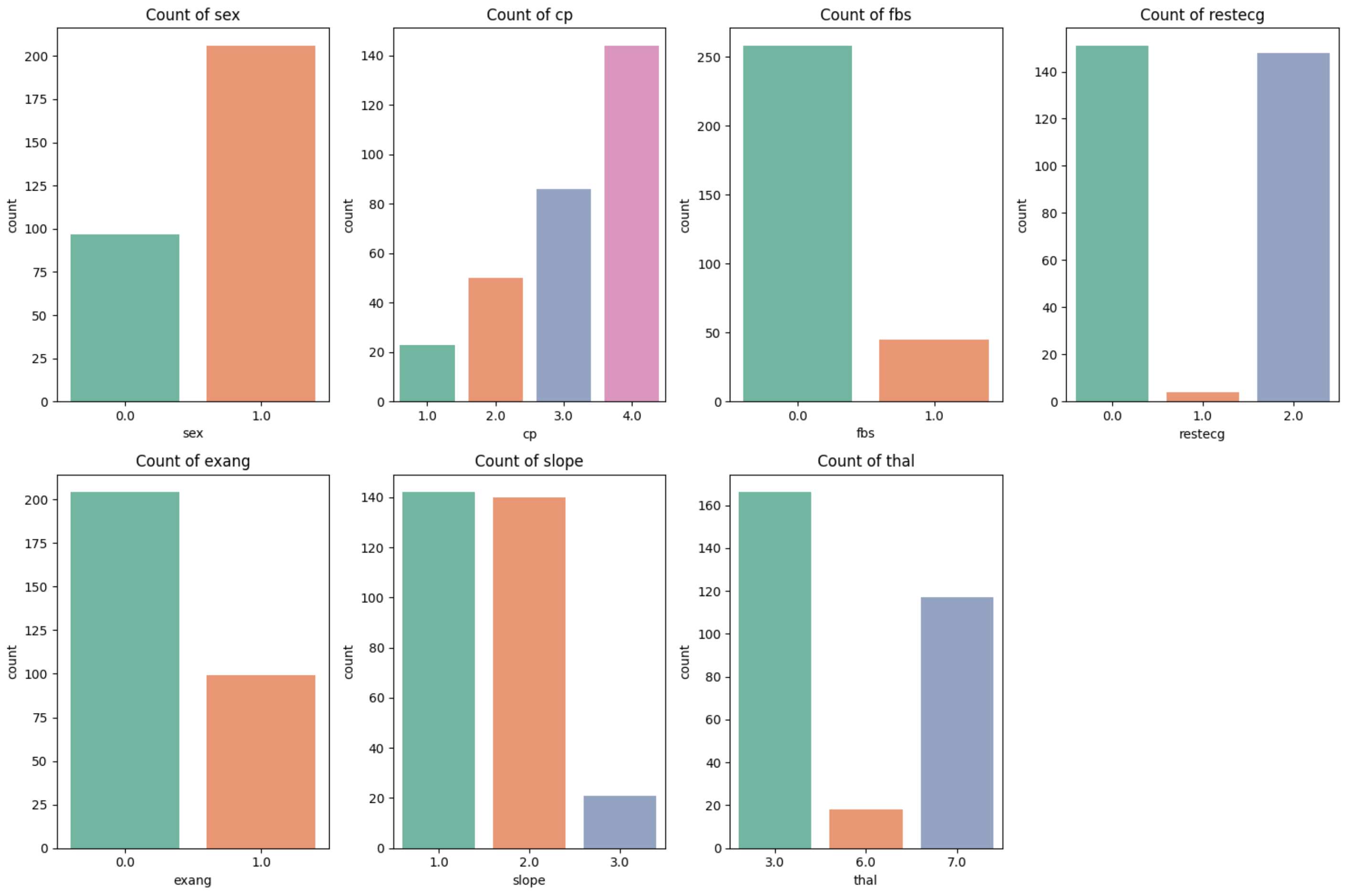
These plots help assess:

- The **normality** or skewness of variables like age, chol, thalach
- The presence of **extreme values** (e.g., oldpeak, ca)
- Whether **scaling** is needed before modeling



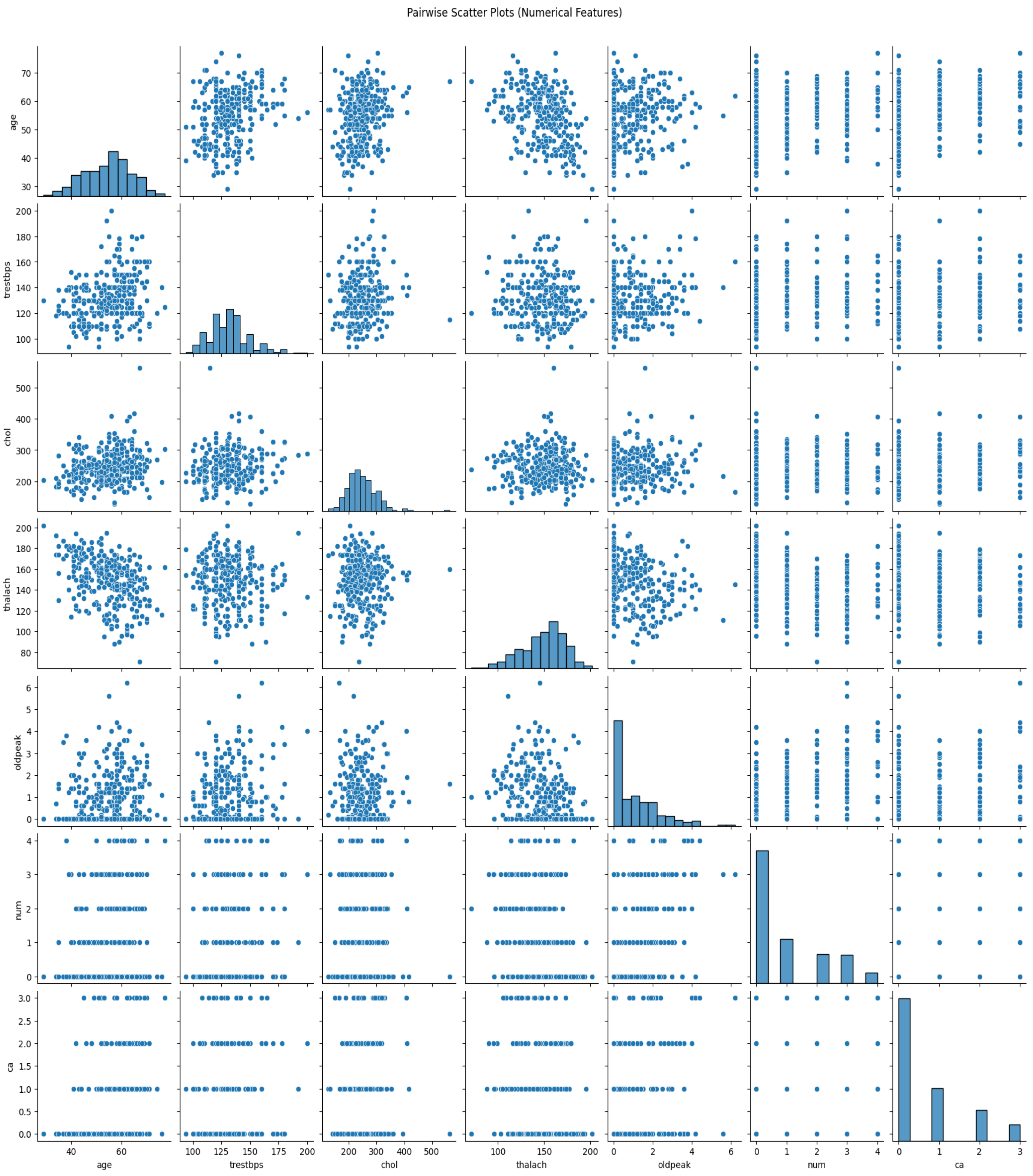
4.4 Categorical Feature Distributions

To understand the distribution of each **categorical feature**, **count plots** were created using Seaborn. These plots show the frequency of each class within a feature.



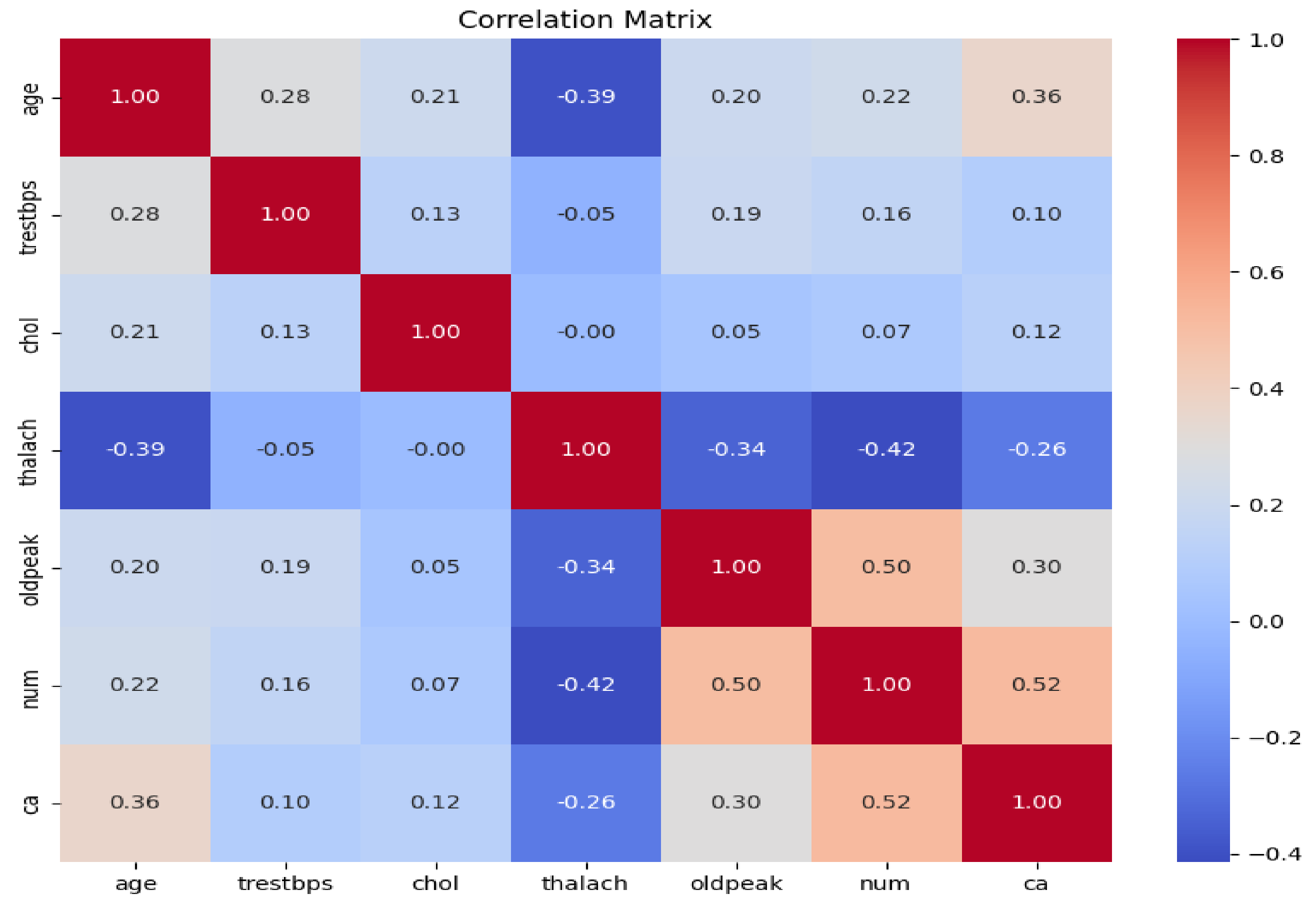
4.5 Pairwise Relationships of Numerical Features

To explore the interactions and potential relationships between numerical features, a **pairplot** was generated using Seaborn. This plot creates scatter plots between every pair of selected numerical features and diagonal histograms for individual features.



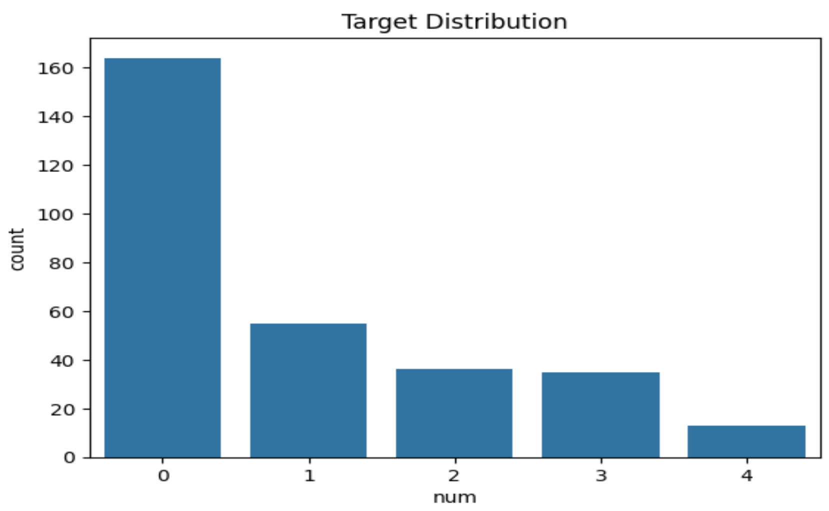
4.6 Correlation Matrix

To examine the **linear relationships** between numerical variables, a **correlation matrix** was created and visualized using a heatmap. This step helps identify which features are positively or negatively correlated, which can be important for both feature selection and model interpretation.



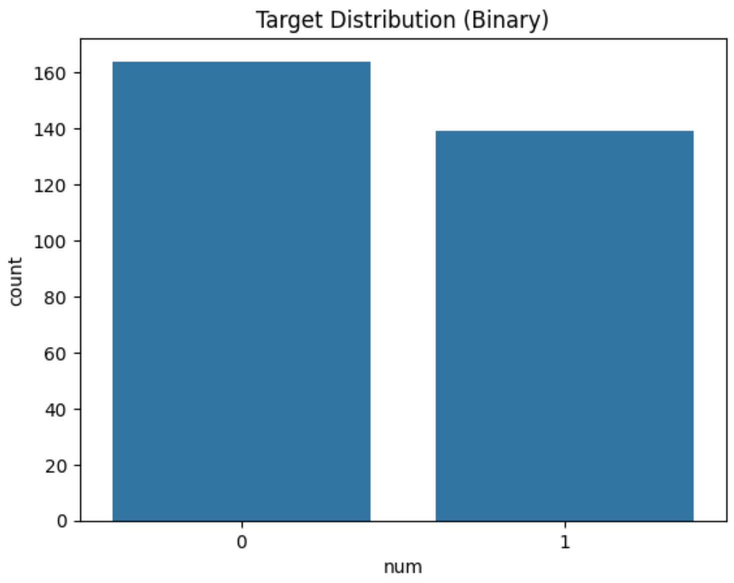
4.7 Target Variable Distribution (Before Binarization)

Before converting the target variable into a binary format, the distribution of the original num values was analyzed. This variable originally represented different levels of heart disease severity, ranging from 0 (no disease) to 4 (most severe).



4.8 Target Variable Distribution (After Binarization)

To simplify the classification task and avoid issues with severe class imbalance, the original multiclass num variable (ranging from 0 to 4) was **binarized**:



5. Preparing Features and Target

5.1 Splitting Features and Target

To prepare the data for modeling, the dataset was divided into two parts:

- **X** → Contains all the input (independent) features
- **y** → Contains the target (dependent) variable, num, which indicates the presence (1) or absence (0) of heart disease

5.2 Identifying Feature Types

Based on domain knowledge and prior exploration, features were manually divided into **numerical** and **categorical** groups. This ensured that each feature was treated appropriately during preprocessing.

- **numerical_cols**: These include features that are inherently numeric, such as age, cholesterol levels, or resting blood pressure.
- **categorical_cols**: These are features that represent categories, such as sex, chest pain type, and thalassemia. These may be stored as strings or already encoded as categories.

These columns were explicitly defined to ensure that:

- Numerical columns could be scaled and imputed appropriately.
- Categorical columns could be encoded using One-Hot Encoding.

5.3 Building the Preprocessing Pipeline

With the numerical and categorical features identified, a **modular preprocessing pipeline** was built using scikit-learn's Pipeline and ColumnTransformer. This pipeline automates the transformation of each feature group appropriately.

Numerical Feature Processing:

- Missing values were imputed using the **mean** of each feature.
- Features were **standardized** using StandardScaler to normalize them around zero with unit variance.

Categorical Feature Processing:

- Missing values were filled using the **most frequent** category for each column.
- Features were encoded using **One-Hot Encoding** to represent each category as a separate binary column without introducing ordinal relationships.

5.4 Splitting the Data into Training and Testing Sets

- After preparing and preprocessing the features, the dataset was divided into **training** and **testing** sets using an 80-20 split. Stratified sampling was used to ensure that the distribution of the target variable (num) remained consistent in both subsets.

• **Training set (80%)**: Used to fit and tune the machine learning models.

• **Test set (20%)**: Reserved exclusively for evaluating model generalization on unseen data.

• **Stratification**: Ensured that both 0 (no disease) and 1 (disease) classes are proportionally represented in both sets

5.5 Applying the Preprocessing Pipeline

With the preprocessor pipeline defined and the data split into training and test sets, the transformations were applied .

- fit_transform() was used on the **training set** to learn scaling parameters and encoding structure.
- transform() was used on the **test set** to apply those same learned transformations — ensuring the model doesn't "see" the test data during training.

This results in:

- A **numerical matrix** ready for use in model training
- All features properly imputed, scaled, and encoded

6. Model Training and Evaluation

To identify the most effective algorithm for heart disease prediction, five machine learning models were trained, tuned, and evaluated using stratified 5-fold cross-validation and grid search.

6.1 Models Trained

The following models were included in the comparison:

- **Logistic Regression**
- **K-Nearest Neighbors (KNN)**
- **Support Vector Machine (SVM)**
- **XGBoost**
- **LightGBM**

Each model was tuned using **GridSearchCV**, which tested multiple hyperparameter combinations to identify the best configuration based on cross-validated accuracy.

6.2 Training & Tuning Code

Each model was evaluated on the test set using the following metrics:

- **Accuracy**
- **Precision**
- **Recall**
- **F1 Score**
- **AUC-ROC**

Additionally, **confusion matrices** were generated to visualize the classification performance for each model.

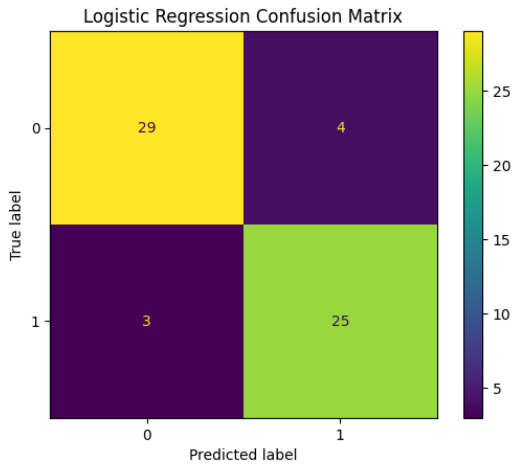
Logistic Regression:

```
              precision    recall  f1-score   support

           0       0.93      0.82      0.87        33
           1       0.81      0.93      0.87        28

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.88      0.87      0.87        61
```

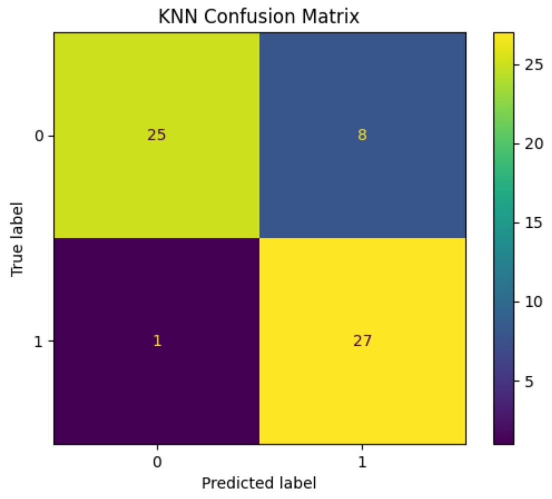
AUC-ROC: 0.9610



KNN:

	precision	recall	f1-score	support
0	0.93	0.85	0.89	33
1	0.84	0.93	0.88	28
accuracy			0.89	61
macro avg	0.89	0.89	0.89	61
weighted avg	0.89	0.89	0.89	61

AUC-ROC: 0.9481

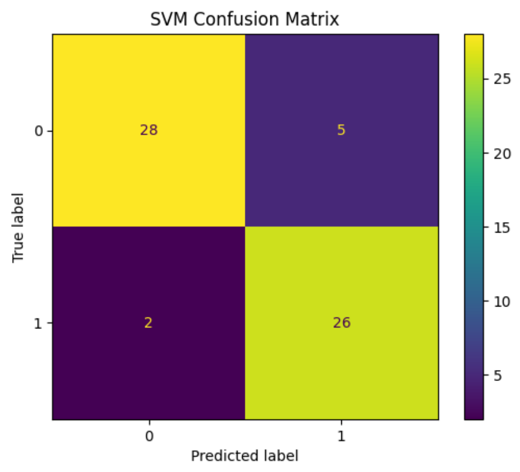


SVM:

	precision	recall	f1-score	support
0	0.93	0.85	0.89	33
1	0.84	0.93	0.88	28
accuracy			0.89	61

macro avg	0.89	0.89	0.89	61
weighted avg	0.89	0.89	0.89	61

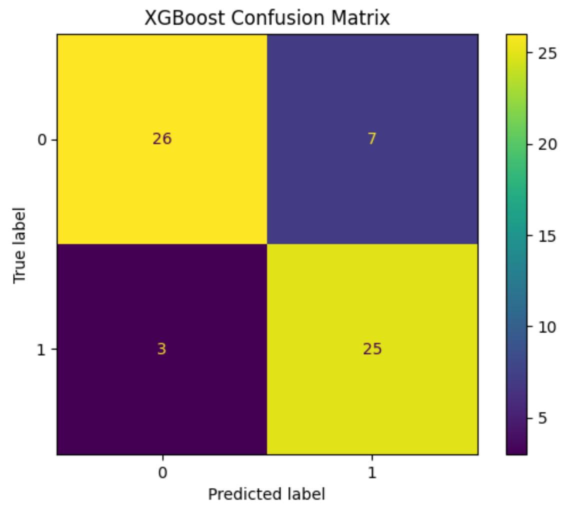
AUC-ROC: 0.9535



XGBoost:

	precision	**recall**	**f1-score**	**support**
0	0.90	0.79	0.84	33
1	0.78	0.89	0.83	28
accuracy			0.84	61
macro avg	0.84	0.84	0.84	61
weighted avg	0.84	0.84	0.84	61

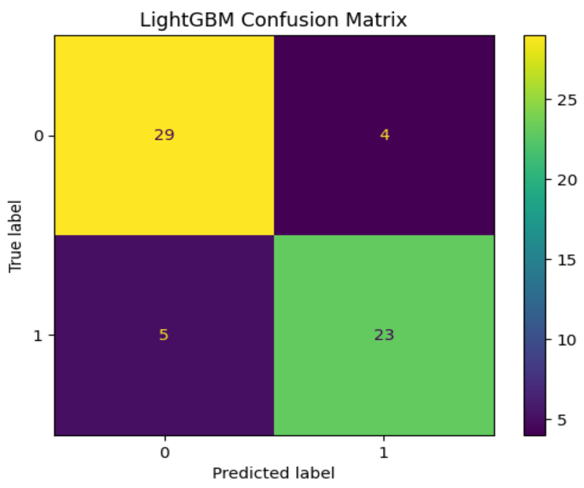
AUC-ROC: 0.8939



LightGBM:

	precision	**recall**	**f1-score**	**support**
0	0.85	0.88	0.87	33
1	0.85	0.82	0.84	28
accuracy			0.85	61
macro avg	0.85	0.85	0.85	61
weighted avg	0.85	0.85	0.85	61

AUC-ROC: 0.9329



6.3 Training a Neural Network (MLP) Using PyTorch

In addition to traditional machine learning models, a **Multi-Layer Perceptron (MLP)** was implemented using **PyTorch** to evaluate the performance of a deep learning approach on the same dataset.

Model Architecture

The MLP consisted of the following layers:

- Input layer (size equal to number of features after preprocessing)
- Hidden Layer 1: 64 neurons with ReLU activation
- Hidden Layer 2: 32 neurons with ReLU activation
- Output Layer: 1 neuron with Sigmoid activation (for binary classification)

Training Setup

- **Loss Function**: Binary Cross Entropy (nn.BCELoss)
- **Optimizer**: Adam with learning rate 0.001
- **Epochs**: 20
- **Batch Size**: 32

6.4 Neural Network (MLP) Evaluation

After training, the MLP model was evaluated on the test set. Predictions were made using a **threshold of 0.5**, and performance metrics were calculated using scikit-learn.

MLP Test Accuracy: 0.8689

```
              precision recall f1-score support

     0.0          0.93     0.82     0.87     33

     1.0          0.81     0.93     0.87     28

 accuracy                           0.87     61

 macro avg        0.87     0.87     0.87     61

 weighted avg     0.88     0.87     0.87     61
```

7. Final Model Comparison and Results Summary

After training and evaluating all machine learning models, the results were consolidated into a single comparison table.

Model Evaluation Summary:

```
                        Accuracy  Precision  Recall    F1 Score  AUC-ROC

    KNN                 0.885246  0.838710  0.928571  0.881356  0.948052

    SVM                 0.885246  0.838710  0.928571  0.881356  0.953463

    Logistic Regression 0.868852  0.812500  0.928571  0.866667  0.961039

    MLP (Neural Net)    0.868852  0.812500  0.928571  0.866667  0.948052

    LightGBM            0.852459  0.851852  0.821429  0.836364  0.932900

    XGBoost             0.836066  0.781250  0.892857  0.833333  0.893939
```

8. Best Model Discussion and Final Analysis

The final comparison of all trained models was based on five key evaluation metrics: **Accuracy**, **Precision**, **Recall**, **F1 Score**, and **AUC-ROC**. These metrics provide a balanced perspective on overall correctness, sensitivity to disease cases, and the ability to avoid false alarms.

Key Takeaways:

- **KNN** and **SVM** achieved the **highest accuracy** (88.52%) and **F1 scores**, indicating strong performance in both precision and recall.
- **Logistic Regression** had slightly lower accuracy (86.89%) but the **highest AUC-ROC (0.9610)**, suggesting it's the most reliable in distinguishing between disease and non-disease cases across thresholds.
- The **MLP (Neural Network)** performed closely behind, showing deep learning methods can be effective even with a relatively small dataset.
- **LightGBM** had the **highest precision (0.8519)**, which makes it useful in scenarios where **false positives must be minimized**, though its recall was slightly lower.
- **XGBoost**, while a strong baseline, slightly underperformed compared to others in this particular dataset.

Recommended Model: SVM

Given the strong balance between:

- High **accuracy** (0.8852),
- High **F1 Score** (0.8814),
- Excellent **AUC-ROC** (0.9535),

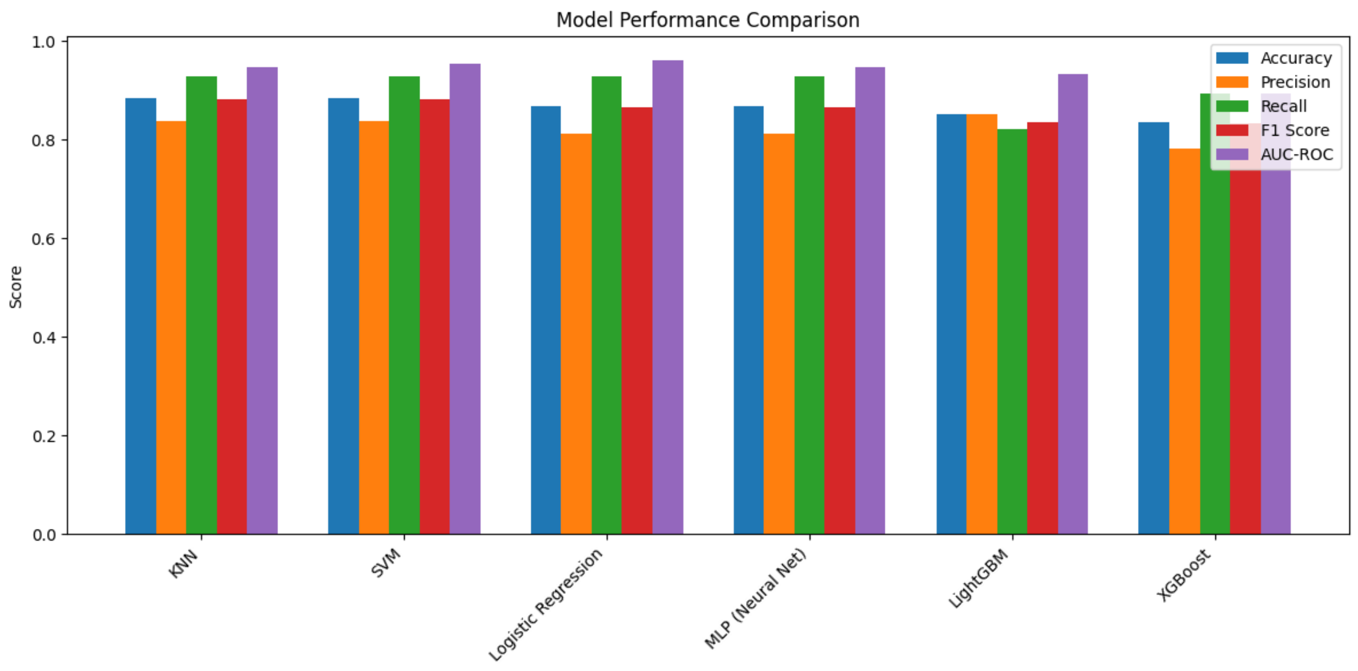
SVM stands out as the most balanced and effective model overall. It demonstrates excellent classification ability while remaining relatively simple to interpret and deploy.

However, **Logistic Regression** may be preferred if **model interpretability** is a key requirement, especially in clinical settings.

Final Notes:

- All models show high recall (~92.86%), which is critical in medical applications where **missing a true case of heart disease can be life-threatening**.
- Trade-offs between complexity, accuracy, and interpretability should guide model selection depending on the deployment scenario.

To summarize and visually compare the performance of all models, a grouped bar chart was created using matplotlib. This chart shows each model's performance across five key metrics:



1.

9.Appendix A – Colab Notebook

The complete code used for this project, including data preprocessing, model training, evaluation, and visualizations, is documented in a Colab notebook.

Notebook Access:

Heart_disease_notebook

This notebook includes:

- Cleaned and structured code blocks
- Inline outputs and evaluation metrics
- Charts and plots used in analysis
- Model training loop with hyperparameter tuning

10.Appendix B – Model Results CSV

A CSV file summarizing the evaluation metrics for each model has been generated and included in this submission.

File name : Result