Heart Disease Risk Prediction using Machine Learning Models

Student ID	Name	Section
206001007	Zehra KOLAT	1
220911757	Kutay ŞAHİNLER	2
2309111082	Göktuğ ŞAHİN	3

Table 1: Team Members

COE305 - Machine Learning

FEMILDA JOSEPHIN JOSEPH SHOBANA BAI

Istinye University
Faculty of Engineering and Natural Sciences

Heart Disease Risk Prediction using Machine Learning Models

206001007 Zehra KOLAT Section 1
220911757 Kutay ŞAHİNLER Section 2
2309111082 Göktuğ ŞAHİN Section 3

28/12/2025

Final Project Report

Contents

1 Project Title

Heart Disease Risk Prediction using Machine Learning Models

2 Problem Statement

Heart disease is one of the most common causes of death worldwide. This project aims to predict people's risk of heart disease, thus enabling early intervention and treatment. The existing methods used today are mostly based on the experience of doctors, which can be time-consuming. By applying machine learning models to structured medical data, we aim to identify high-risk patients more efficiently. Machine learning can help automate risk assessment and reduce human errors in clinical decision-making processes.

3 Objectives

- To collect and preprocess the Kaggle Heart Disease Dataset for classification, including handling missing values(if exist).
- To implement multiple supervised classification algorithms (such as Logistic Regression, etc) to predict heart disease risk.
- To assess model performance using standard classification metrics.

4 Dataset Description

Dataset Source: Kaggle

- Dataset Link: `https://www.kaggle.com/datasets/pratyushpuri/heart-disease-dataset-3k-rows-python-code-2025`
  Kaggle Dataset Link

- Data Type: Structured Data (.csv)
- No. of Features: 17
- No. of Samples: 3070
- Target Variable: Heart Disease Presence (0: No Disease, 1: Disease)

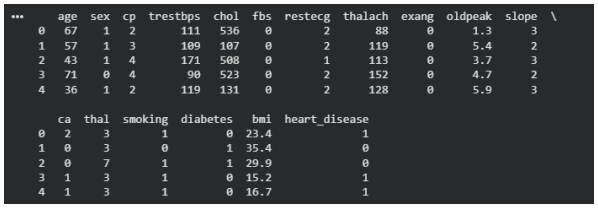
5 Data Pre-processing

5.1 Handling Missing Values

```
# Importing required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
```

```
# Load the dataset
df = pd.read_csv("heart_disease_dataset.csv")  # giving messy db to the data frame
print(df.head()) # this is our first 5 row
```



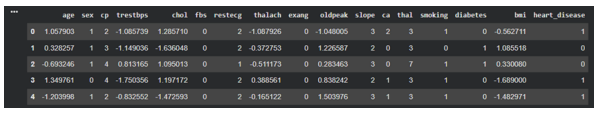
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	smoking	diabetes	bmi	heart_disease
0	67	1	2	111	536	0	2	88	0	1.3	3	2	3	1	0	23.4	1
1	57	1	3	109	107	0	2	119	0	5.4	2	0	3	0	1	35.4	0
2	43	1	4	171	508	0	1	113	0	3.7	3	0	7	1	1	29.9	0
3	71	0	4	90	523	0	2	152	0	4.7	2	1	3	1	0	15.2	1
4	36	1	2	119	131	0	2	128	0	5.9	3	1	3	1	0	16.7	1

Figure 1: output

The *df.head*() function displays the first five rows of the dataset. This output was used to verify that the data was loaded correctly and that each column was in the expected format. It was determined that the target variable, heart_disease, was correctly defined as 0 and 1. We've successfully loaded the 'heart_disease_dataset.csv' dataset into a Pandas DataFrame.

5.2 Feature Scaling / Normalization

```
from sklearn.preprocessing import StandardScaler

numerical_features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'bmi']

scaler = StandardScaler()

df_scaled = df.copy()

df_scaled[numerical_features] = scaler.fit_transform(df_scaled[numerical_features])

df_scaled.head()
df = df_scaled
```



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	smoking	diabetes	bmi	heart_disease
0	1.057903	1	2	-1.085739	1.285710	0	2	-1.087926	0	-1.048005	3	2	3	1	0	-0.562711	1
1	0.328257	1	3	-1.149036	-1.636048	0	2	-0.372753	0	1.226587	2	0	3	0	1	1.085518	0
2	-0.693246	1	4	0.813165	1.095013	0	1	-0.511173	0	0.283463	3	0	7	1	1	0.330080	0
3	1.349761	0	4	-1.750356	1.197172	0	2	0.388561	0	0.838242	2	1	3	1	0	-1.689000	1
4	-1.203998	1	2	-0.832552	-1.472593	0	2	-0.165122	0	1.503976	3	1	3	1	0	-1.482971	1

Figure 2: output

All numerical features (age, tbps, cholesterol, thalassic, ex-peak, body mass index) were standardized using StandardScaler. This scaling transforms each feature so that its mean is ≈ 0 and its standard deviation is ≈ 1, ensuring that all variables contribute equally to the machine learning models regardless of their original scale.

Positive values indicate measurements above the mean.

Negative values indicate measurements below the mean.

Values close to zero are close to the mean.

Examples:

age = 1.0579 $\rightarrow$ This individual's age is approximately 1 standard deviation above the mean.

cholesterol = -1.6360 $\rightarrow$ This individual's cholesterol is approximately 1.6 standard deviations below the mean.

This normalization reduces the bias that could arise from variables measured at different scales and improves the performance and convergence of machine learning models.

5.3 Check for duplicate rows

```
print(df.info()) # lets look if there any empty cells, as you can see there is not
```

Output 1:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3069 entries, 0 to 3068
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             3069 non-null   int64
 1   sex             3069 non-null   int64
 2   cp              3069 non-null   int64
 3   trestbps        3069 non-null   int64
 4   chol            3069 non-null   int64
 5   fbs             3069 non-null   int64
 6   restecg         3069 non-null   int64
 7   thalach         3069 non-null   int64
 8   exang           3069 non-null   int64
 9   oldpeak         3069 non-null   float64
 10  slope           3069 non-null   int64
 11  ca              3069 non-null   int64
 12  thal            3069 non-null   int64
 13  smoking         3069 non-null   int64
 14  diabetes        3069 non-null   int64
 15  bmi             3069 non-null   float64
 16  heart_disease   3069 non-null   int64
dtypes: float64(2), int64(15)
memory usage: 407.7 KB
None
```

```
print("Number of duplicate rows: ", df.duplicated().sum())
# lets check if there is any duplicate rows, there is not
```

Output 2:

```
Number of duplicate rows:  0
```

As we can see from the two code blocks:

We used the $df.info()$ command to understand the overall structure and quality of the dataset. According to the output, our dataset consists of a total of 3069 rows and 17 columns.

The most important finding is that there are no missing data. The 'Non-Null Count' column in the output confirmed that there are 3069 full records across all 17 attributes, a perfect match to the total number of rows.

Furthermore, the data type (Dtype) of all columns appears as int64 or float64. This indicates that all the data is already in numeric format and ready for machine learning models; we won't need to convert text data like 'Male'/'Female'.

5.4 Irrelevant columns

```
# All columns are relevant for predicting heart disease; no columns were removed.
print(df["heart_disease"].value_counts()) # wrong label checking : only 0 and 1 present
```

```
heart_disease
0    1878
1    1191
Name: count, dtype: int64
```

In this project, our goal is to predict the heart_disease column. So, We first needed to check this target column. We ran the $df['heart\_disease'].value\_counts()$ command to ensure it only contained the values '0' (Not Patient) and '1' (Patient), meaning it wasn't a wrong label.

Fortunately, the result was only 0 and 1.

This command also showed us the class distribution in the dataset. Accordingly:

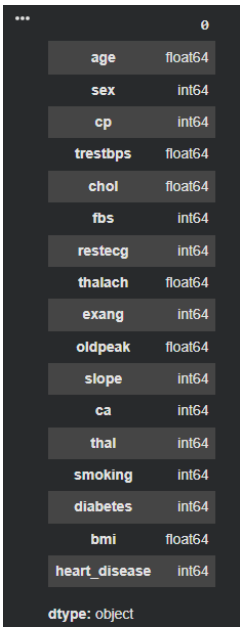
1878 people (61.2%) were labeled 'No Heart Disease' (0).

1191 people (38.8%) were labeled 'Heart Disease' (1).

We can also see the imbalanced data. This is why we will be doing smote in the later stages.

5.5 Encoding Categorical Variables – Label encoder/ One-hot encoder

```
df.dtypes
```



The data type of all columns appears as int64 or float64. This indicates that all the data is already in numeric format and ready for machine learning models; we won't need to convert text data like 'Male'/'Female'. Since all data is in numeric format, there is no need to apply label encoding or one-hot encoding . So no additional encoding step was applied.

5.6 Outliers detection

```
    numerical_cols = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'bmi']

plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(2, 3, i)
    sns.boxplot(data=df, x=col)
    plt.title(f'{col} Boxplot')
plt.tight_layout()
plt.show()
```

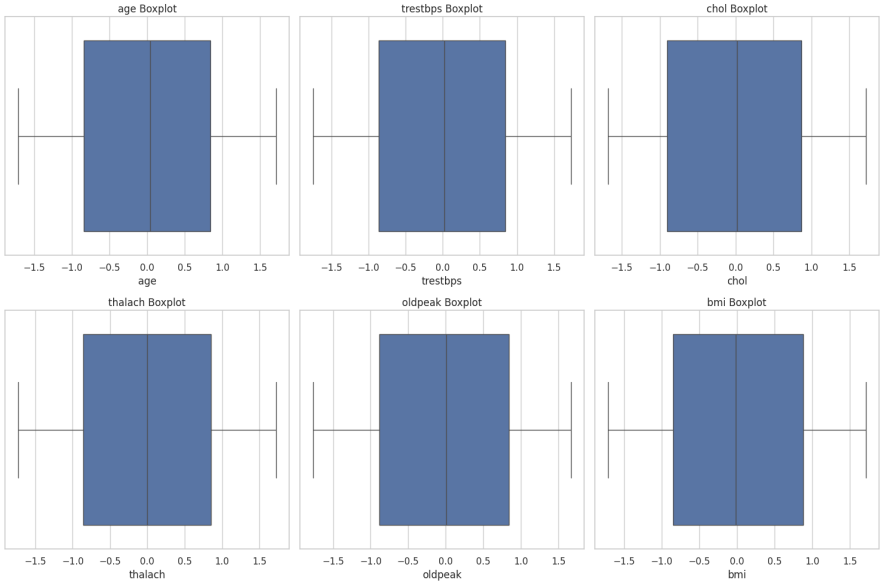


Figure 3: We look for continuous columns:

Outlier detection was performed for all numerical features (age, trestbps, chol, thalach, oldpeak, bmi) using boxplots. The analysis showed that there are no significant outliers in the dataset. Therefore, all numerical values were retained for model training, as the data is clean and does not contain extreme deviations that could bias the results.

6 Split & Smote

We prepare data for the model: split and smote to increase model accuracy.

```
    x = df.drop('heart_disease', axis=1) # delete last lable so x have only datas
y = df['heart_disease'] # y have only last lable 0 & 1

x_train, x_test, y_train, y_test = train_test_split (x, y, test_size = 0.2,
    random_state = 42, stratify = y) # as always we will siplit data to train and test
    with the ratio of 8 - 2

print("Original Training Counts:")
print(y_train.value_counts()) # showing last lable before SMOTE
```

```
    Original Training Counts:
heart_disease
0     1502
1      953
Name: count, dtype: int64
```

The dataset was split into training (80%) and testing (20%) sets. The training data showed class imbalance (0: 1502, 1: 953), which could bias the model. SMOTE will be applied to balance the classes:

```
smote = SMOTE(random_state=42) # in here we create blank smote func, it does not
    nothing yet
x_train_resampled, y_train_resampled = smote.fit_resample(x_train, y_train) # as we
    know fit is training keyword we use it to aplly SMOTE

print("\nBalanced Training Counts (After SMOTE):")
print(y_train_resampled.value_counts()) # showing last lable after SMOTE
```

```

Balanced Training Counts (After SMOTE):
heart_disease
1     1502
0     1502
Name: count, dtype: int64
```

SMOTE was applied to the training set to address class imbalance. After resampling, both classes are balanced with 1502 samples each, ensuring the model can learn equally from both classes.

7 Exploratory Data Analysis (EDA)

7.1 Statistical Summary

```
# Calculating basic statistics for numeric columns
numerical_cols = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'bmi']
stat_summary = df[numerical_cols].describe().T  #

print(stat_summary)

medians = df[numerical_cols].median()
modes = df[numerical_cols].mode().iloc[0]

print("\nMedians:\n", medians)
print("\nModes:\n", modes)
```

```
...            count          mean       std       min       25%       50%  \
age          3069.0  2.593053e-16  1.000163 -1.714750 -0.839175  0.036399
trestbps     3069.0  4.306320e-16  1.000163 -1.750356 -0.864200  0.021955
chol         3069.0 -1.892697e-16  1.000163 -1.683722 -0.900501  0.018934
thalach      3069.0  2.199464e-17  1.000163 -1.733889 -0.857225 -0.003631
oldpeak      3069.0 -2.118431e-16  1.000163 -1.769218 -0.881572  0.006074
bmi          3069.0 -7.177199e-17  1.000163 -1.716470 -0.851150 -0.013301

                 75%       max
age         0.839009  1.714584
trestbps    0.844813  1.730969
chol        0.870262  1.721590
thalach     0.849963  1.726627
oldpeak     0.838242  1.670410
bmi         0.879489  1.717339

Medians:
 age         0.036399
trestbps     0.021955
chol         0.018934
thalach     -0.003631
oldpeak      0.006074
bmi         -0.013301
dtype: float64

Modes:
 age         0.109364
trestbps     0.749868
chol         0.379897
thalach     -0.788014
oldpeak      1.393020
bmi          0.330080
Name: 0, dtype: float64
```

Figure 4: output

All numerical features (age, trestbps, arm, height, height of the head, height of the head, body mass index) were standardized using StandardScaler, with a mean around 0 and a standard deviation around 1. Positive values indicate measurements above the mean, negative values indicate measurements below the mean, and values closer to 0 indicate measurements close to the mean. Median values close to 0 confirm that the data are centered, while mode values represent the most frequent observations on the normalized scale. This standardization ensures that all features contribute equally to machine learning models, preventing biases due to different scales.

7.2 Data Visualization

7.2.1 Histograms

```
numerical_cols = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'bmi']

plt.figure(figsize=(15,10))
```

```
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(2, 3, i)
    sns.histplot(df[col], kde=True, bins=20)
    plt.title(f'{col} Distribution')
plt.tight_layout()
plt.show()
```

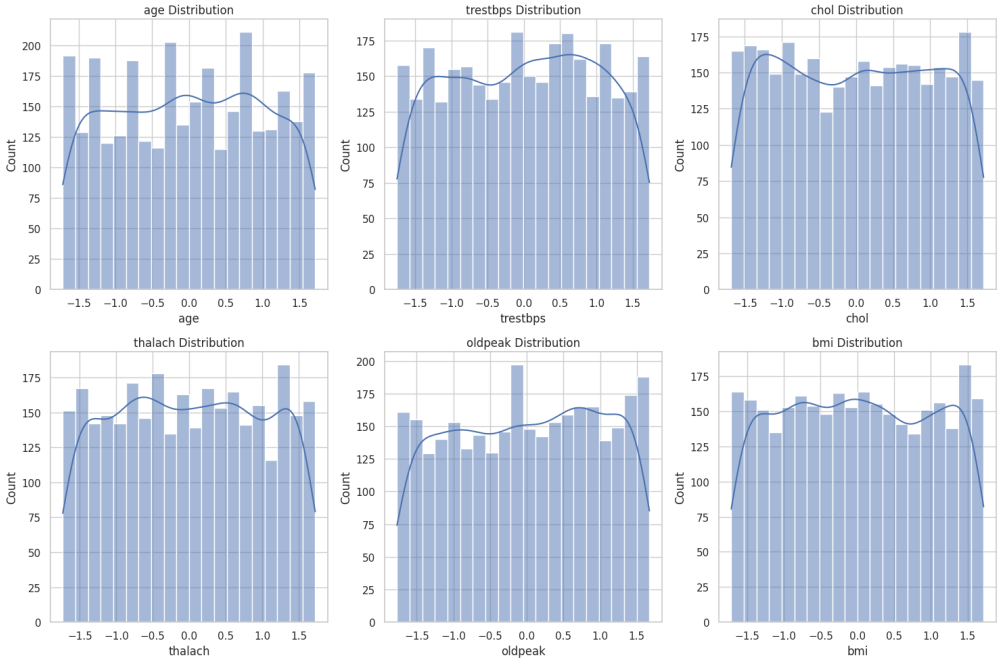


Figure 5: output

Because the numerical variables (age, trestbps, chol, thalach, oldpeak, bmi) were normalized with StandardScaler, the mean was clustered around 0 and the standard deviation was scaled to 1. When the histograms were examined, most values lie between -1.5 and +1.5, and no significant outliers were observed. The distributions are not a classical normal distribution; some variables exhibit multimodal or uniform structures. This demonstrates that the data was successfully normalized before model training and that the data is distributed evenly.

7.2.2 Correlation Heatmap

```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```

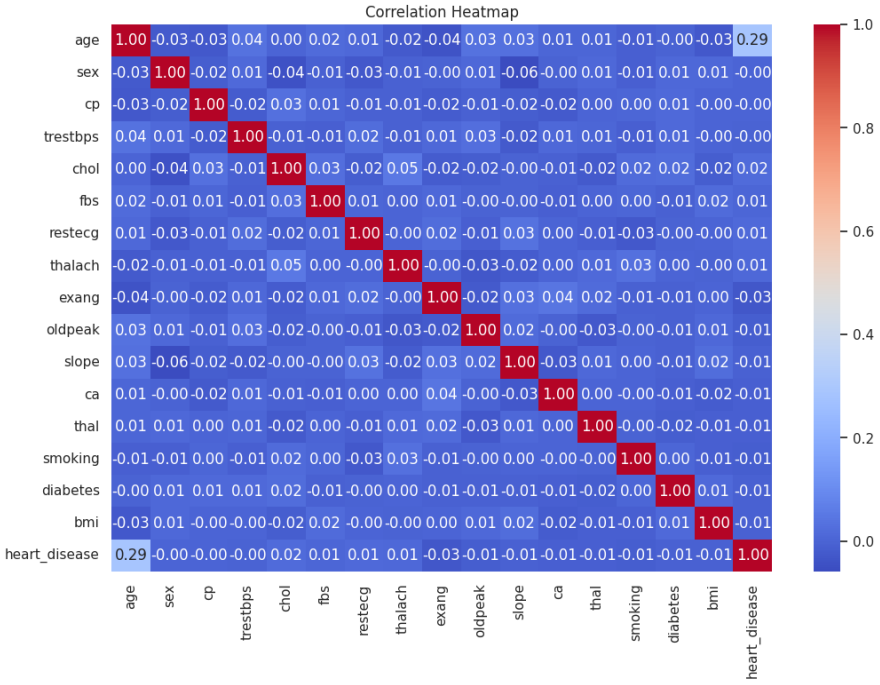


Figure 6: output

We used this correlation heatmap to understand how the different features relate to our target, "*heart_disease*." The most important finding is that Age has the strongest connection, with a score of 0.29. confirming that higher age is associated with an increased risk of heart disease in this dataset. Conversely, other clinical features such as cholesterol, blood pressure, and chest pain exhibit correlation values close to zero, indicating a lack of strong linear dependencies with the target.

7.2.3 Pair Plot / Scatter Plots

```
 sns.pairplot(df, vars=['age', 'trestbps', 'chol', 'thalach', 'bmi'], hue='
     heart_disease')
plt.show()
```

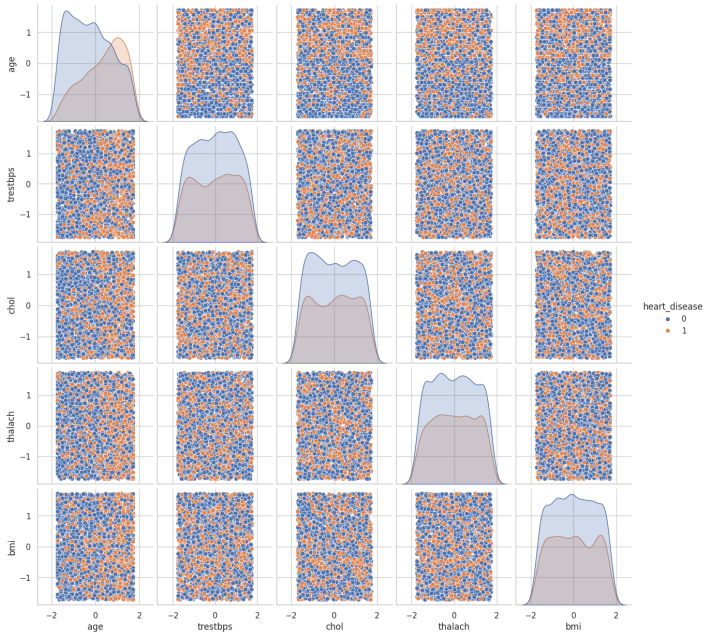


Figure 7: output

We created this pair plot to observe relationships between pairs of numerical features like age, blood pressure, cholesterol, heart rate, and BMI. The blue dots represent healthy people (0), and the orange dots represent people with heart disease (1). The most important thing we see here is that the blue and orange dots are heavily mixed together in all the charts; there are no clear separate groups or clusters. This overlap shows that we cannot easily separate healthy and sick patients by looking at just two features at a time. Therefore, machine learning models that can learn more complex and non-linear relationships are needed to accurately predict heart disease risk.

7.2.4 Count Plots

```
categorical_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal', '
    smoking', 'diabetes']

plt.figure(figsize=(15,10))
for i, col in enumerate(categorical_cols[:6], 1):  # To show the first 6 categorical
    columns
    plt.subplot(2, 3, i)
    sns.countplot(x=df[col])
    plt.title(f'{col} Count')
plt.tight_layout()
plt.show()
```

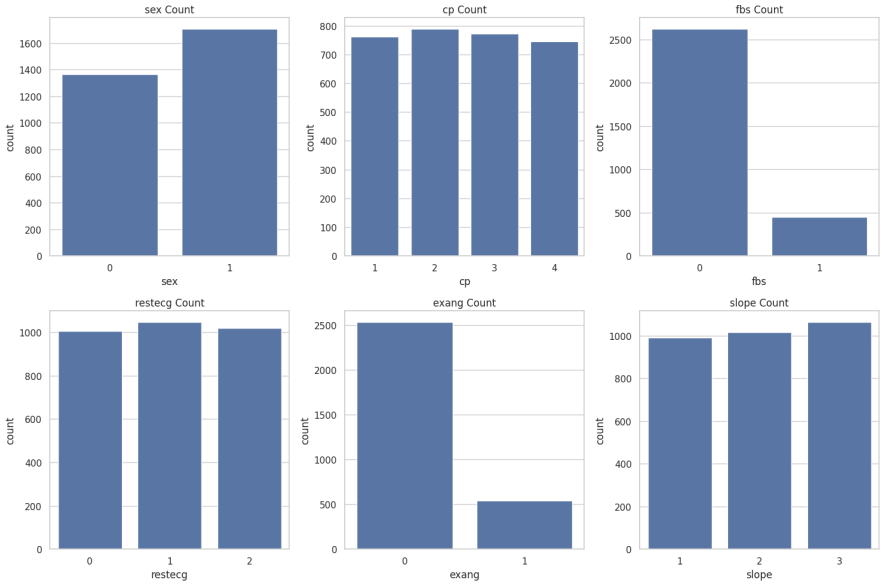


Figure 8: output

We used count plots to see the distribution of categorical variables. The charts reveal a very interesting pattern: features like Chest Pain (cp), Resting ECG (restecg), and Slope have almost perfectly equal distributions. On the other hand, features like Fasting Blood Sugar (fbs) and Exercise Angina (exang) are not balanced; most patients fall into category 0, while category 1 has much fewer people. Finally, the Sex variable shows that there are slightly more patients in group 1 compared to group 0.

8 NON-ENSEMBLE MODELING

8.1 Chosen Models

Algorithm Chosen	Justification of choosing the algorithm
Naive Bayes	It is simple and very fast, making it a strong baseline model.
Logistic Regression	Interpretable, efficient, and the standard choice for binary classification.
Support Vector Machine (SVM)	Heart-disease features form non-linear patterns, and SVM captures those relationships better than linear models.

Table 2: Models

8.2 Cross-Validation Setup

- CV technique: Stratified K-fold Cross Validation
- Number of folds: 5
- Evaluation metrics: Accuracy
  Precision
  Recall
  F1-Score

Model	Accuracy (Mean ± SD)	Precision	Recall	F1-Score	AUC
Naive Bayes	0.58	0.59	0.58	0.58	0.6001
Logistic Regression	0.61	0.62	0.61	0.62	0.6380
SVM	0.6221	0.64	0.62	0.63	0.6473

Table 3: Model Performance Metrics of Test Results

8.3 Results and Discussion

Among the evaluated models (SVM, Logistic Regression, and Naive Bayes), Support Vector Machine (SVM) demonstrated the strongest and most balanced performance. While Logistic Regression also provided reliable results, SVM achieved slightly better generalization on the test set.

Comparing the training and test results, SVM showed remarkable consistency with only a minimal drop in accuracy (Training: 0.64 vs. Test: 0.62). This indicates that SVM effectively captures the decision boundaries in the high-dimensional medical data without overfitting. Logistic Regression followed closely as a stable linear model. In contrast, Naive Bayes consistently performed the lowest across all metrics, likely because its assumption of feature independence is too simple for the complex interactions present in this heart disease dataset.

Conclusion: SVM is the most suitable model for this dataset, offering the best balance between predictive power and generalization capability. Logistic Regression serves as a strong and stable alternative with interpretable results. Naive Bayes, however, lacks the complexity required for this task and consistently underperforms compared to the other two models.

Model	Reason
1. SVM	Best generalization, stable train-test performance
2. Logistic Regression	Reliable, no overfitting, decent metrics
3. Naive Bayes	Simple model; too weak for this dataset

Table 4: Ranking Across Models

9 ENSEMBLE LEARNING MODELING

tabularx

Model	Reason for Use
GBM (Gradient Boosting)	Strong predictive performance, handles complex relationships
XGB (XGBoost)	Fast, scalable, reduces overfitting, high accuracy
Random Forest	Ensemble of decision trees, reduces overfitting, robust to noise, good baseline for comparison

Table 5: Reason for Using Different Models

9.1 Hyperparameter Tuning

Tuning Strategy
Method used: RandomizedSearchCV
Cross-validation folds: Stratified K-fold Cross Validation - 5
Scoring metric: Accuracy, Precision, Recall, F1-Score

9.2 Hyperparameter Details

Random Forest

Hyperparameters Tuned	Best Values
n_estimators	207
max_depth	18
min_samples_split	4
min_samples_leaf	2
max_features	'sqrt'

Table 6: Random Forest Hyperparameter Tuning Results

GBM (Gradient Boosting)

Hyperparameters Tuned	Best Values
subsample	0.9
n_estimators	500
min_samples_leaf	20
max_depth	3
learning_rate	0.01

Table 7: GBM (Gradient Boosting) Hyperparameter Tuning Results

XGB (XGBoost)

Hyperparameters Tuned	Values Tested (Search Space)
booster	gbtree, gblinear
scale_pos_weight	2, 3
min_child_weight	15, 20, 30
max_depth	1, 2
reg_alpha	1, 10, 50
reg_lambda	1, 10
colsample_bytree	0.4, 0.6
learning_rate	0.01, 0.05, 0.1
n_estimators	200, 300, 500

Table 8: XGBoost Hyperparameter Tuning Search Space

Model	Accuracy	Precision	Recall	F1-Score
GBM (Gradient Boosting)	0.62	0.64	0.62	0.63
XGB (XGBoost)	0.5945	0.4732	0.4076	0.4379
Random Forest	0.6075	0.4936	0.4832	0.4883

Table 9: Model Performance Metrics of Test Results (Before Hyperparameter Tuning)

Model	Accuracy	Precision	Recall	F1-Score
GBM (Gradient Boosting)	0.63	0.65	0.63	0.63
XGB (XGBoost)	0.5277	0.4449	0.8824	0.5915
Random Forest	0.6140	0.5019	0.5294	0.5153

Table 10: Model Performance Metrics of Test Results (After Hyperparameter Tuning)

9.3 Results and Discussion

In this study, we evaluated a diverse set of machine learning models to predict heart disease risk. The models ranged from traditional non-ensemble algorithms (Naive Bayes, Logistic Regression, SVM) to advanced ensemble methods (Random Forest, Gradient Boosting, XGBoost). Our primary evaluation metrics were Accuracy and F1-Score, given the importance of balancing false positives and false negatives in medical diagnosis.

9.4 Feature Selection Analysis (Ensemble Learning Models)

GMB (Gradient Boosting)

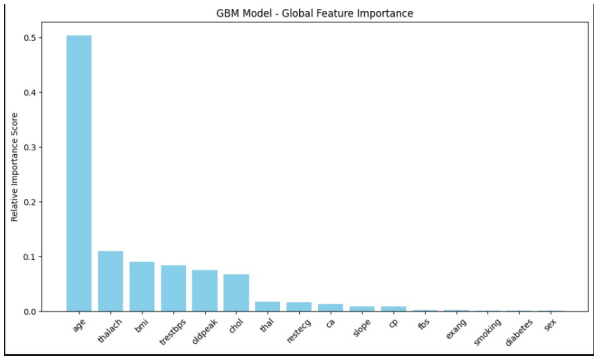


Figure 9: GBM - Feature Importance

XGB (XGBoost)

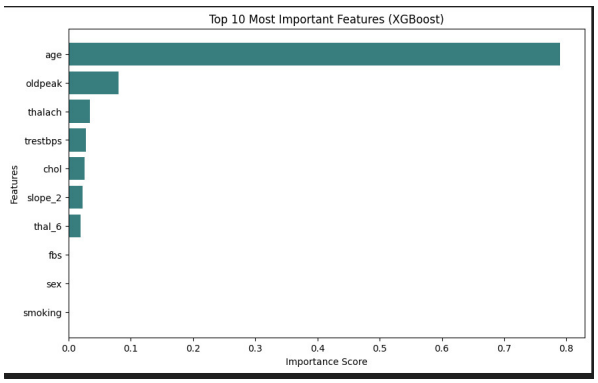


Figure 10: XGBoost - Feature Importance

Random Forest

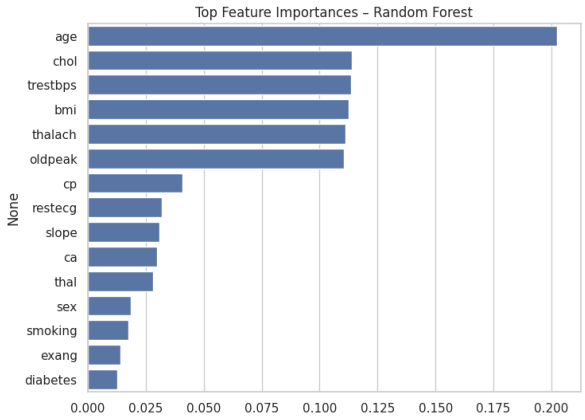


Figure 11: Random Forest - Feature Importance

10 Overall Results and Discussion

When comparing all implemented models, we observed that:

Stability: Support Vector Machine (SVM) from the non-ensemble category and GBM from the ensemble category provided the most consistent results, both hovering around 62-64% accuracy.

Data Complexity: The fact that no model—simple or complex—could significantly exceed the 64% accuracy threshold suggests an inherent limitation in the dataset. As visualized in our EDA (Pair Plots), the features of healthy and unhealthy patients overlap significantly, making it difficult for any classifier to draw a perfect decision boundary. **Ensemble vs. Non-Ensemble Comparison**

Category	Model Name	Accuracy	F1-Score	Key Characteristic
Non-Ensemble	Naive Bayes	0.58	0.58	Simple baseline, but underfitted the data
Non-Ensemble	Logistic Regression	0.61	0.62	Stable and interpretable
Non-Ensemble	SVM	0.62	0.63	Most robust non-ensemble model
Ensemble	Random Forest	0.61	0.52	Significant overfitting observed
Ensemble	GBM (Gradient Boosting)	0.63	0.63	Best overall balance (Winner)
Ensemble	XGBoost	0.53	0.59	High recall (0.88) but very low precision

Table 11: Performance Summary of Models After Hyperparameter Tuning

Stability vs. Complexity: The Non-Ensemble models (specifically SVM and Logistic Regression) demonstrated remarkable stability. They were less prone to overfitting and provided consistent results across cross-validation folds. However, their linear nature (or simple kernels) limited their ability to push accuracy beyond 62

Predictive Power: The Ensemble models generally showed higher potential. GBM successfully leveraged boosting to capture non-linear patterns that linear models missed, achieving the highest overall accuracy (63%). However, this complexity came at a cost: Random Forest memorized the training data (Overfitting), and XGBoost became overly aggressive in predicting positive cases (High Bias towards Class 1).

11 Conclusion

In this project, we developed and evaluated six different machine learning models to predict heart disease risk. After comparing non-ensemble methods (like SVM and Logistic Regression) with ensemble methods (Random Forest, GBM, XGBoost), the Gradient Boosting Machine (GBM) emerged as the final best-performing model.

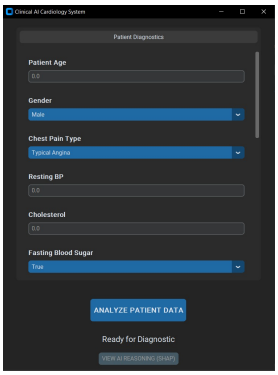
GBM achieved the most balanced results with an Accuracy and F1-Score of 63%. While Random Forest showed potential during training, it suffered from overfitting, and XGBoost was too biased towards predicting positive cases. GBM succeeded because it builds trees sequentially to correct previous errors, making it better at handling the complex and overlapping data patterns we observed in our analysis.

However, the fact that no model significantly exceeded 64% accuracy suggests that the current dataset has limitations, specifically that the features of healthy and sick patients are very similar. For future work, the most important improvement would be collecting a larger dataset with more distinct features to separate the classes better. Additionally, exploring Deep Learning techniques or applying more advanced feature engineering could help uncover deeper patterns and improve the system's reliability for real-world medical use.

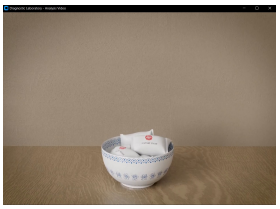
12 User Interface

The developed user interface is built upon the Gradient Boosting Machine (GBM), which was identified as the most successful model in this project. The system processes patient data provided by the user (such as age, blood pressure, and cholesterol) to predict heart disease risk with high accuracy.

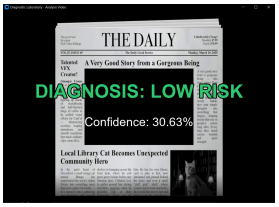
Beyond simply presenting a binary prediction (0 or 1), SHAP (SHapley Additive exPlanations) analysis has been integrated to ensure model transparency. Through SHAP plots, the system visualizes exactly how much each feature (e.g., advanced age or chest pain type) pushes the result towards a positive or negative diagnosis. This integration transforms the model from a "black box" into a reliable Decision Support System, explaining the underlying logic behind every prediction.



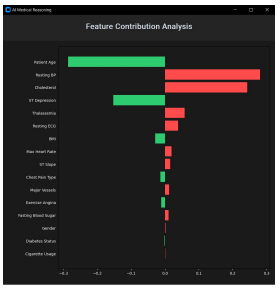
UI-1



UI-2



UI-3



UI-4

Figure 12: User Interface Screenshots

13 SHAP

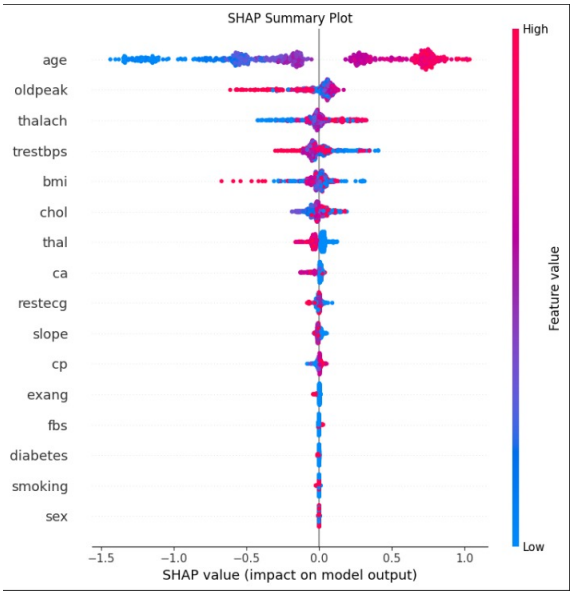


Figure 13: GBM (Gradient Boosting)

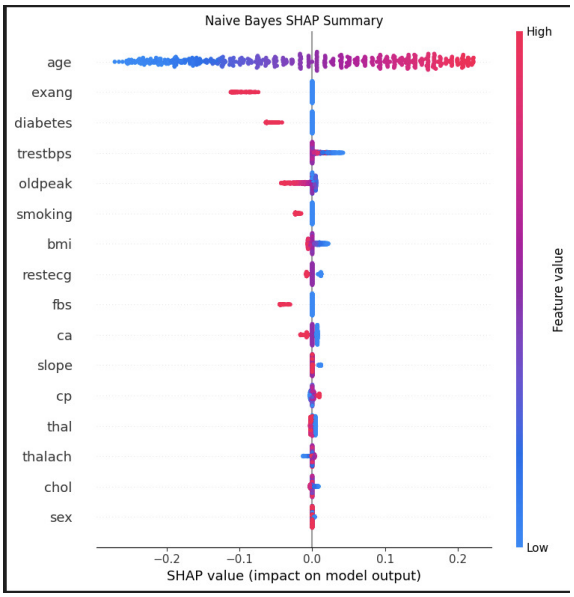


Figure 14: Naive Bayes

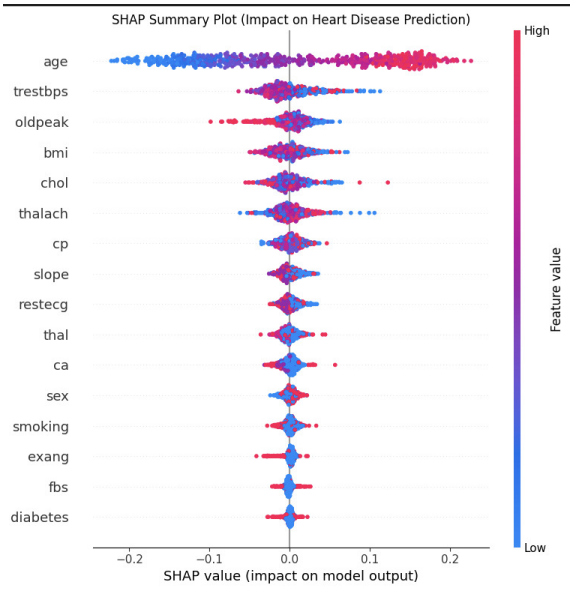


Figure 15: Random Forest

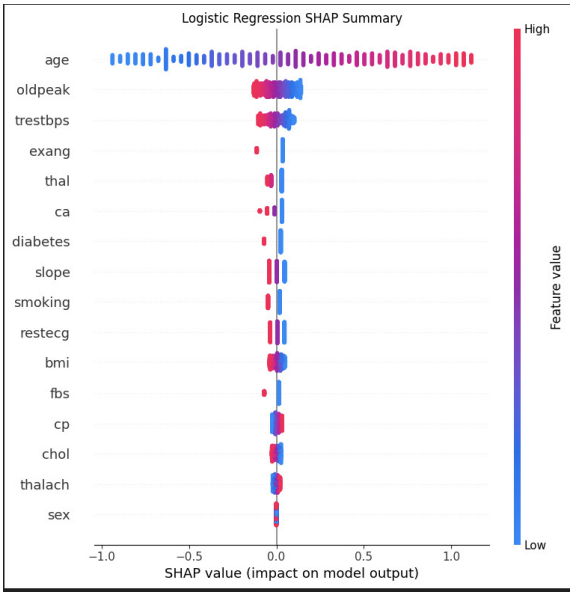


Figure 16: Logistic Regression

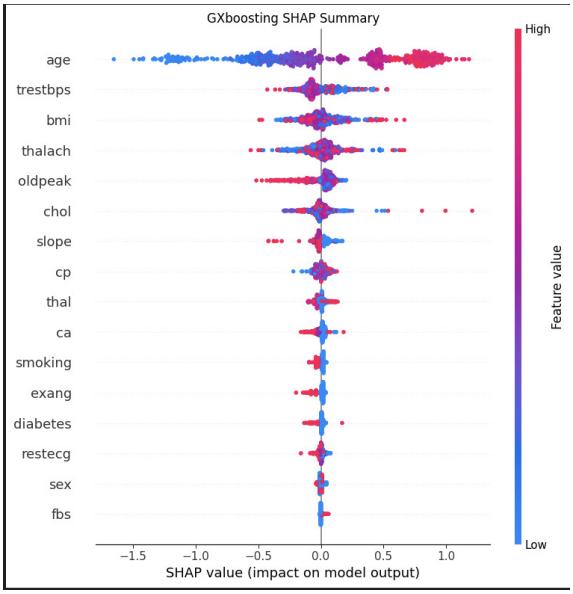


Figure 17: XGBoost

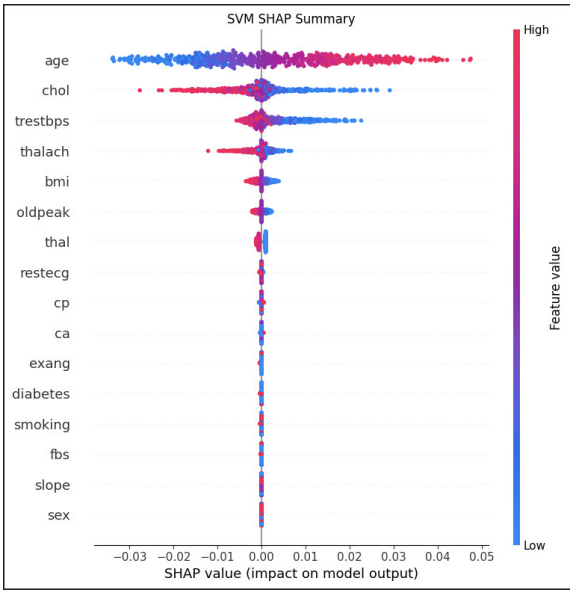


Figure 18: SVM