Heart Disease Classification

Project Report

Leen Omar Abdallah Al Oyoun 202420244

Data Mining and Machine Learning A0373405
Dr. Ahmad Abu-Shareha
June 11, 2025

Abstract

Heart disease is one of the most serious health challenges worldwide, which makes early detection extremely important and lifesaving. In this project, I developed a machine learning model to predict whether a patient has heart disease based on clinical features using the Heart Disease UCI dataset. The dataset was visualized and preprocessed, with missing values imputed using a LightGBM Regressor and categorical variables encoded effectively. I explored two different feature selection techniques (SelectKBest and Recursive Feature Elimination (RFE)) followed by balancing the data using SMOTE. Four classifiers were trained and evaluated: Logistic Regression, SVM, XGBoost, and a Multi-layer Perceptron (MLP). After comparing performance metrics across models, the MLP classifier trained on RFE-selected features achieved the best results, with the highest accuracy (88%), F1-score (89.68%), and the lowest number of false negatives. These findings are encouraging in a medical context, where minimizing false negatives is critical to avoid misdiagnosing patients.

1. Introduction

Cardiovascular diseases are the leading cause of death globally, taking an estimated 17.9 million lives each year. More than four out of five cardiovascular diseases deaths are due to heart attacks and strokes, and one third of these deaths occur prematurely in people under 70 years of age. It is important to detect cardiovascular disease as early as possible so that management with counselling and medicines can begin.
The objective of this project is to develop a machine learning model that can accurately classify whether a patient has a heart disease or not based on a set of clinical features.

The dataset used for this project is the Heart Disease UCI dataset which was acquired from Kaggle repository. It consists of 918 patient records and 11 clinical features.
The target variable to be predicted is HeartDisease, where a value of '1' indicates the presence of the disease and '0' indicates its absence.

2. Data Exploration and Preprocessing

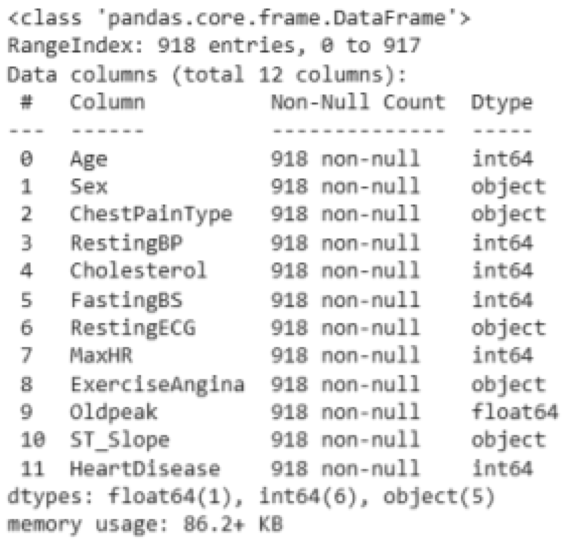
In every dataset, preprocessing should be applied to ensure the data we are working with is clean and of good quality, because data quality directly impacts the performance and accuracy of the models.

- Libraries used:
    - **Pandas**: an open source library, easy-to-use data structures and data analysis tools for the Python programming language.
    - **NumPy**: a Python library that provides a multidimensional array object, an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, and much more.
    - **Scikit-learn**: an open source library, which includes Simple and efficient tools for data mining and data analysis.
    - **LightGBM**: a gradient boosting framework that uses tree based learning algorithms.
    - **Matplotlib**: a comprehensive library for creating static, animated, and interactive visualizations.
    - **Seaborn**: a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

2.1. Data Exploration

First, we need to understand the data and its structure to know what data preprocessing steps to follow and in what order.

The dataset was loaded into pandas, `.head()` and `.shape()` were used to have a first look at the data. It was revealed that we have categorical and numerical values, and that the dataset contains 918 records and 12 features.
To get a closer look at our data, `.info()` was used to give us a summary of the columns and their data types.

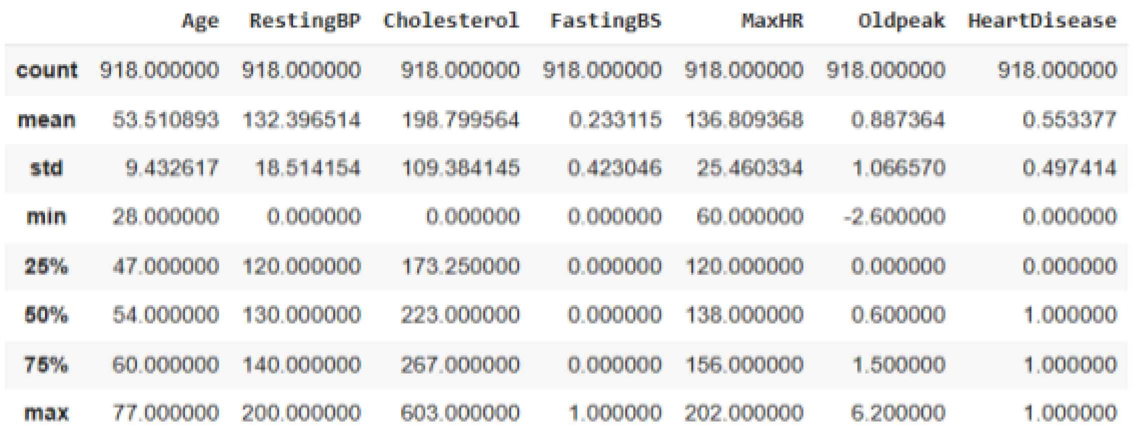


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

Figure 1

As shown in Figure 1, we have 7 numerical features; one column of type float64, 6 columns of type int64 and 5 categorical features; 5 columns of type object.
All columns reported 918 non-null values, indicating no immediately missing data at this stage.

Then to provide an overview of the count, mean, standard deviation, and quartile values for each feature, a descriptive statistical summary of the numerical columns was generated using the `.describe()` method. This statistical analysis highlighted significant data quality issues.



	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.553377
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.497414
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.000000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.000000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.000000

Figure 2

As shown in Figure 2 , the minimum value for both the RestingBP (Resting Blood Pressure) and Cholesterol columns is 0.0. These values are physiologically impossible for a patient and it is clear that the zero was used as a placeholder for missing data. This issue will be addressed in next steps.

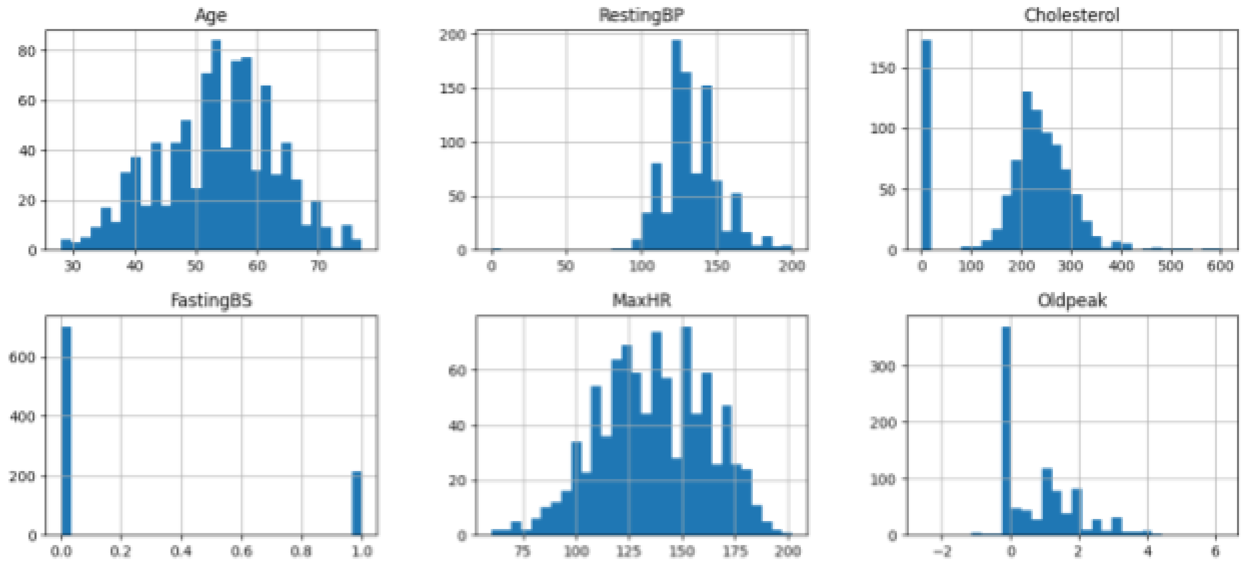


Figure 3

To better understand the characteristics of the numerical features, their distributions were visualized using histograms, as shown in Figure 3. Visualizations are important for identifying data patterns, seeing how data is distributed and if skewness is present in the distribution.
We see clear evidence of the Cholesterol data quality issue identified in the statistical summary, the zero values are completely detached from the distribution of other cholesterol values, which is also the case for RestingBP.

One last important step is to check if our target feature classes are balanced or not. This was done using seaborn's `histplot()` and using the `.count()` method to reveal the number of each distinct class in our target variable.

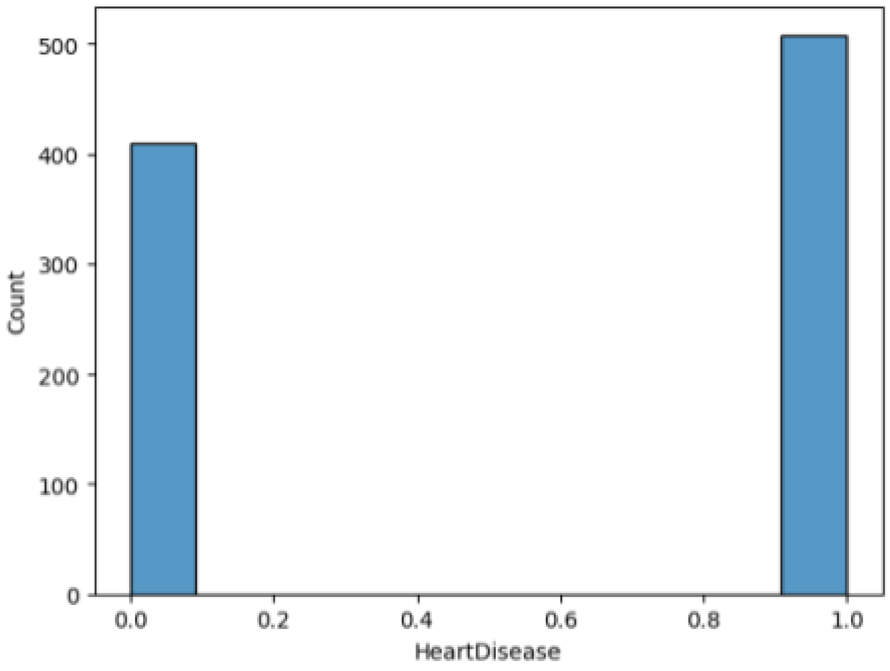
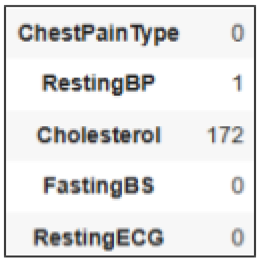


Figure 4

As shown in Figure 4 and from the `.count()` method, it was revealed that we have 508 counts for class 1 (55% of patients having heart disease) and 410 for class 0 (45% of patients not having a heart disease). While this imbalance is not severe, it still makes the dataset imbalanced and therefore the dataset should be balanced to avoid any bias towards the majority class.

2.2. Data Preprocessing:

2.2.1. Checking for Missing Values and Duplicates



ChestPainType	0
RestingBP	1
Cholesterol	172
FastingBS	0
RestingECG	0

Figure 5

The data was checked for the existence of duplicates and it was revealed that there were no duplicate records in the dataset.

It was addressed in data exploration that we have zeros as placeholders for nan values. That is why the 0 values in Cholesterol and RestingBP were converted to nan values, making them direct missing values.

Now when checking for null values using `data.isnull().sum()` as shown in Figure 5, we see that there is one missing RestingBP value (which will be dropped since its one single value) and 172 missing Cholesterol values (which will be imputed using LGBMRegressor).

2.2.2. Encoding Categorical Features

We mentioned earlier that we have 5 categorical features which are:

- **Sex**: represents the patient's gender [M: Male, F: Female].
- **ExerciseAngina**: exercise-induced angina [Y: Yes, N: No].
- **ChestPainType**: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic].
- **RestingECG**: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
- **ST_Slope**: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]

The encoding of categorical features was performed while taking into account the type of categorical attributes (nominal or ordinal) and whether features were binary or multi-valued.

Sex and **ExerciseAngina** were encoded using (0,1) mapping as they are both binary nominal features. **ST_Slope**, a multi-valued ordinal feature, was encoded using mapping (Down, Flat, Up) to (0,1,2) respectively. **ChestPainType** and **RestingECG**, both multi-valued nominal features, were encoded using one-hot encoding.

2.2.3. Handling Negative OldPeak Values

The Oldpeak feature represents ST-segment deviation during stress. Some values were observed to be negative. To simplify the data and avoid potential misinterpretation of medical indicators or incorrect assumptions, the feature was transformed to its absolute value. This allows the model to focus on the magnitude of deviation, regardless of direction.

2.2.4. Imputation of Cholesterol Missing Values Using LGBMRegressor

To handle the missing values in the Cholesterol column, regression imputation was applied using the LightGBMRegressor, a powerful gradient boosting model known for its efficiency and accuracy. The goal was to predict the missing cholesterol values based on the features from our dataset.

As shown in the code below, rows with missing cholesterol values were identified and labeled using a new binary column `is_nan`, where the missing values of cholesterol were indicated by 1 and the rest by 0. This helped separate known and missing cholesterol values.

```
import lightgbm as lgb
from lightgbm import LGBMRegressor
data_copy = data.copy()
nan_chol_ind = np.where(data_copy['Cholesterol'].isna())[0]
nan_chol = data_copy.index[nan_chol_ind]
data['is_nan']= 0
data.loc[nan_chol, 'is_nan'] = 1
```

Then the data was split into train and test as shown below, as the `train` included records where Cholesterol values are known and `test` included records where Cholesterol values were missing.

```
train = data[data['is_nan']==0]
test = data[data['is_nan']==1]
```

As shown in the code below, we dropped the target column `Cholesterol` and the column `is_nan` from the x_train_impute set to form the feature set, because we want to keep all training features except Cholesterol and is_nan.
The same was done for the x_test_impute set. For y_train_impute, the actual cholesterol values were stored (which are the values the model will learn to predict), and for the y_test_impute, nan values will be stored in it.
a LightGBM regression model was then trained on the training features and their corresponding cholesterol values. Once trained, the model was used to predict the missing cholesterol values in the test set.

```
x_train_impute = train.drop(['Cholesterol','is_nan'],axis=1)
y_train_impute = train['Cholesterol']
```

```
x_test_impute = test.drop(['Cholesterol','is_nan'],axis=1)
y_test_impute = test['Cholesterol']
model = LGBMRegressor()
model.fit(x_train_impute,y_train_impute)
y_pred = model.predict(x_test_impute)
```

The predicted values were then inserted back into the original dataset to fill in the missing entries. To validate the success of the imputation, it was verified that the number of missing values in the Cholesterol column dropped to zero.

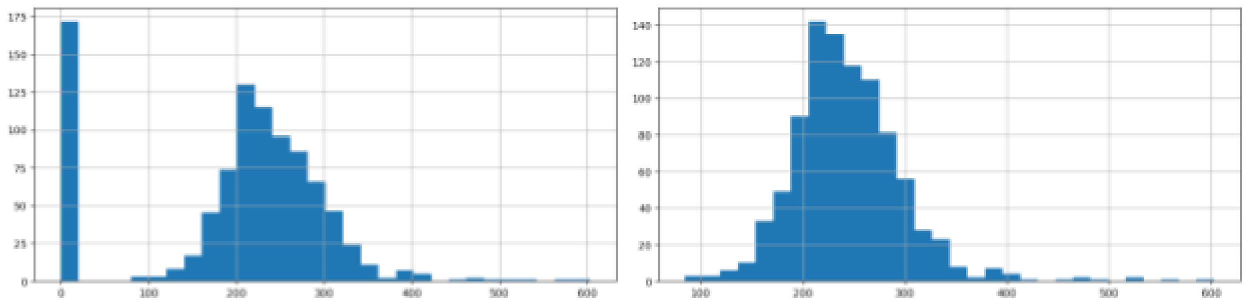


Figure 6

To visualize the effect of the imputation, histograms were used to show the distribution of cholesterol values before (to the left) and after (to the right) imputation as shown in Figure 6. This provided a clear picture of how the model filled in the missing values in a way that preserved the overall data distribution.

2.2.5. Data Splitting and Normalization Using MinMaxScaler

As shown in the code below, the dataset was split into training and testing sets using an 80/20 split. This ensures that the model is evaluated on unseen data, providing a more realistic estimate of its performance.
After splitting the data, Min-Max normalization was applied to scale the feature values to a range between 0 and 1. Importantly, the scaler was fitted only on the training data and then applied to both the training and testing sets. This approach and order was followed to prevent data leakage,because we dont want information from the test set to influence the training process and lead to overly optimistic and overfitted results.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(features, y, test_size=0.2,
random_state=42)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

```
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

3. Feature Selection, Oversampling and Model Training

The order I followed after splitting and normalizing the data, is feature selection, separate oversampling for each feature selection method, and then training the models.
Two feature selection techniques were used, Univariate filter-based feature selection: SelectKBest, and Wrapper feature selection: Recursive Feature Elimination (RFE).

3.1. Univariate FS: SelectKBest

To reduce the dimensionality of the dataset, and improve model performance, SelectKbest was used to select the top 10 features with the best statistical measurements. The statistic used is Chi-Square (chi2). Chi2 measures the association between each feature and the class label, HeartDisease in our case.

To ensure the integrity of the final model evaluation and prevent data leakage, the SelectKBest selector was fitted on the preprocessed training data. This trained selector was then used to transform both the training and testing sets, creating new datasets containing only the 10 chosen features, which are: Sex, ASY, ATA, NAP, ST, FastingBS, MaxHR, ExerciseAngina, Oldpeak, and ST_Slope.

```
from sklearn.feature_selection import SelectKBest, chi2
selector1 = SelectKBest(chi2, k=10)
x_train_kBest = selector1.fit_transform(x_train, y_train)
x_test_kBest = selector1.transform(x_test)
output1 = pd.DataFrame(x_train_kBest)
output1.columns = features.columns[selector1.get_support()]
print(output1)
selector1.get_support()
```

3.1.1. Oversampling using SMOTE

After feature selection was done, it was important to solve the imbalance problem that was addressed previously. An analysis of the target variable revealed that the training set contained 338 samples for class 0 (no heart disease) and 395 samples for class 1 (heart disease).
To prevent the models from becoming biased towards the majority class, the Synthetic Minority Over-sampling Technique (SMOTE) was applied.

This technique works by generating new, synthetic samples for the minority class (class 0) based on its existing instances.

After applying SMOTE, the training data was successfully balanced. The new, resampled training set now consists of 395 samples for both classes 0 and 1, creating a final training dataset of 790 samples. It's important to note the oversampling was applied to the training data only to avoid data leakage.

3.1.2. Models Training and Evaluation

Following the feature selection and oversampling, it was time to train the models and evaluate them to see the best performing model and the model that achieves our goal of correctly predicting whether a patient has a heart disease or not.

Four models were selected for training: Logistic Regression, Support Vector Machine (SVM), eXtreme Gradient Boosting (XGBoost), and a Multi-layer Perceptron (MLP) neural network.

After training the four models, all models were assessed using accuracy, precision, recall, f1-score, and the confusion matrix, and the following observations were made:

Model	Accuracy	Precision	Recall	F1-score	TN	FN	FP	TP
Logistic Regression	84.78%	89.62%	84.82%	87.15%	61	17	11	95
SVM	85.32%	90.47%	84.82%	87.55%	62	17	10	95
XGBoost	**85.32%**	**88.99%**	**86.60%**	**87.78%**	**60**	**15**	**12**	**97**
MLP	84.78%	88.88%	85.71%	87.27%	60	16	12	96

As shown in the table above, XGBoost is the highest performing classifier, it achieved the highest accuracy alongside SVM (85.32%), and achieved the highest recall (86.60%), which is important for our data since we are dealing with medical data, as correctly predicted positive instances to the total number of actual positive instances is highly critical to ensure the model is correctly predicting whether patients have a disease or are healthy.

The XGBoost also achieved the best F1-score (87.78%), reflecting a strong balance between sensitivity and precision. And another important measure is that the number of FN instances is the lowest in XGboost, which is highly important in our data, because classifying a patient as healthy when they have a heart disease is a massive problem, therefore XGBoost having the lowest false negative counts (FN = 15) is favorable.

3.2. Wrapper FS: Recursive Feature Elimination (RFE)

To select the most predictive feature subset in a wrapper-based technique, Recursive Feature Elimination (RFE) with a Decision tree estimator was applied to select the top 10 predictors for HeartDisease (target). RFE works by fitting the estimator on the feature set, ranking each feature by its importance to the classifier, eliminating the least important features recursively until the desired number of features is left.
To prevent data leakage, the RFE selector was fitted on the preprocessed training data, then it was used to transform both the training and testing sets. The 10 features maintained by RFE are: Age, Sex, ASY, RestingBP, Cholesterol, FastingBS, LVH, MaxHR, Oldpeak, and ST_Slope.

```
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
selector2 = RFE(DecisionTreeClassifier(random_state=42), n_features_to_select=10)
x_train_rfe = selector2.fit_transform(x_train, y_train)
x_test_rfe = selector2.transform(x_test)
print(selector2.get_support())
mask = selector2.get_support()
selected_features = features.columns[mask]
print("Selected features:",list(selected_features))
```

3.2.1. Oversampling using SMOTE

After using RFE to choose the best features, the training data (only) was balanced using Synthetic Minority Over-sampling Technique (SMOTE). The exact same steps in 3.1.1 (Oversampling using SMOTE), were followed here as well.

3.2.2. Models Training and Evaluation

Utilizing the previously established four models, which initially employed SelectKBest feature selection, we will now apply them with RFE feature selection and SMOTE data balancing.

After evaluating the models performance on the RFE-transformed training data, a final comparison will be done to compare our upcoming best performing RFE-based model and the XGBoost model previously identified as the best performer in the SelectKBest approach.

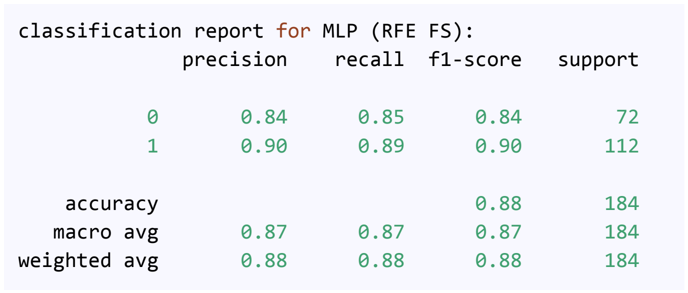
The evaluation results including accuracy, precision, recall, f1-score, and the confusion matrix for the four models trained on the RFE-transformed data are as follows:

Model	Accuracy	Precision	Recall	F1-score	TN	FN	FP	TP
Logistic Regression	85.32%	90.47%	84.82%	87.55%	62	17	10	95
SVM	85.86%	90.56%	85.71%	88.07%	62	16	10	96
XGBoost	86.41%	91.42%	85.71%	88.47%	63	16	9	96
MLP	**87.50%**	**90.09%**	**89.28%**	**89.68%**	**61**	**12**	**11**	**100**

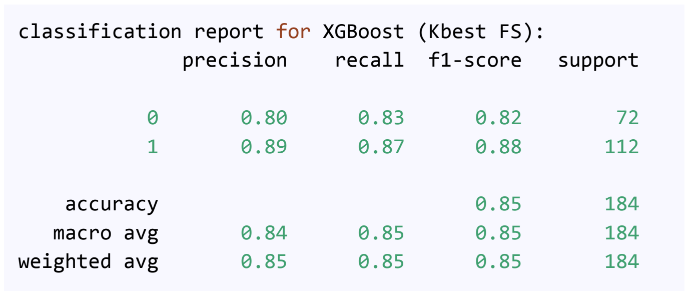
As shown in the table above, MLP showed the best performance, achieving the highest accuracy (87.50%), recall (89.28%) and f1-score (89.68%), It also produced the lowest number of false negatives (12).

4. Final Evaluation and Visualizations

To draw a final conclusion between the two best performing model approaches, XGBoost trained on features selected using SelectKBest, and MLP trained on features selected using RFE, a concluded evaluation was done. Lets take a look at the classification reports to compare:



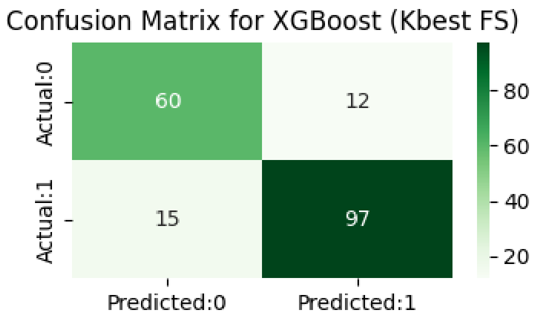
```
classification report for MLP (RFE FS):
              precision    recall  f1-score   support

           0       0.84      0.85      0.84        72
           1       0.90      0.89      0.90       112

    accuracy                           0.88       184
   macro avg       0.87      0.87      0.87       184
weighted avg       0.88      0.88      0.88       184
```

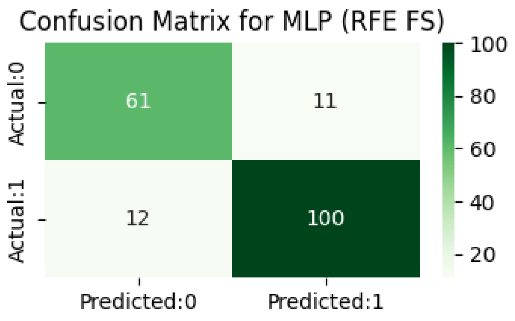


```
classification report for XGBoost (Kbest FS):
              precision    recall  f1-score   support

           0       0.80      0.83      0.82        72
           1       0.89      0.87      0.88       112

    accuracy                           0.85       184
   macro avg       0.84      0.85      0.85       184
weighted avg       0.85      0.85      0.85       184
```

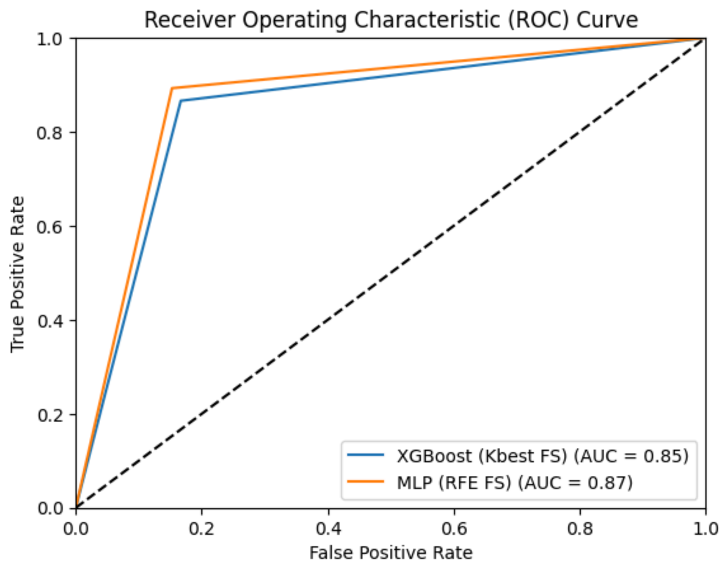
As shown in the classification reports above, MLP achieved a higher overall accuracy (88%) compared to XGBoost's 85%. It also outperformed XGBoost in terms of precision, recall, and f1-score for both classes. Most notably, MLP demonstrated stronger sensitivity in detecting the positive class (heart disease), with a recall of 0.89 versus 0.87 for XGBoost, and fewer false negatives (12 vs. 15), which is especially important in healthcare, because it means that its less likely to miss someone who actually has a heart disease. The confusion matrix heatmaps and ROC curves (shown below) visually reinforce these findings.

● Confusion matrix heatmap





● ROC Curve



The ROC curve illustrates the classification performance of both models. And since both curves are close to the top-left corner, both models perform well in distinguishing between the two classes. But the MLP model achieved an AUC of (0.87) outperforming the XGBoost model, which achieved an AUC of (0.85). Therefore, the MLP model has a higher ability to differentiate between positive and negative cases.

5. Conclusion

This project showed how machine learning can support early heart disease detection. After testing different models and feature selection methods, the MLP classifier with RFE achieved the best performance. If I had more time, I would improve the models by tuning hyperparameters, applying cross-validation, and exploring additional feature selection techniques. This project also made me realize how crucial clean data and careful preprocessing are when building reliable models, especially in sensitive fields like healthcare.

References

GeeksforGeeks. (n.d.). Map true/false to 1/0 in a pandas dataframe. https://www.geeksforgeeks.org/map-true-false-to-1-0-in-a-pandas-dataframe/

Bello, S. M., et al. (2020). A curated knowledgebase of cardiovascular disease models for risk prediction and biomarker discovery. *Database: The Journal of Biological Databases and Curation*. https://academic.oup.com/database/article/doi/10.1093/database/baaa010/5809229

Benjamin, E. J., et al. (2017). Heart Disease and Stroke Statistics. *Circulation*. https://pmc.ncbi.nlm.nih.gov/articles/PMC5203736/

GeeksforGeeks. (n.d.). Working with missing data in pandas. https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/

Tolulade, A. (n.d.). Getting Started with Python for Data Science. https://github.com/Tolulade-A/Getting-Started-with_Python_for_Data-Science/blob/main/Understanding_Python_for_Data_Analysis_Part1.ipynb

Kaggle. (n.d.). Heart Failure Prediction - Discussion. https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction/discussion

Gelman, A. (n.d.). Missing data in regression. https://sites.stat.columbia.edu/gelman/arm/missing.pdf

Cambridge Spark. (n.d.). Introduction to missing data imputation. https://medium.com/@Cambridge_Spark/tutorial-introduction-to-missing-data-imputation-4912b51c34eb

Deep Learning Nerds. (n.d.). Encode ordinal categorical features in pandas. https://www.deeplearningnerds.com/pandas-encode-ordinal-categorical-features/

StackExchange. (n.d.). Imputation before or after splitting into train/test.

https://stats.stackexchange.com/questions/95083/imputation-before-or-after-splitting-into-train-and-test

Géron, A. (n.d.). Hands-On ML with Scikit-Learn.

https://github.com/ageron/handson-ml2/blob/master/02_end_to_end_machine_learning_project.ipynb

GeeksforGeeks. (n.d.). Find duplicate rows in a dataframe.

https://www.geeksforgeeks.org/find-duplicate-rows-in-a-dataframe-based-on-all-or-selected-columns/

GeeksforGeeks. (n.d.). Change the order of pandas dataframe columns.

https://www.geeksforgeeks.org/change-the-order-of-a-pandas-dataframe-columns-in-python/

WebMD. (n.d.). Homozygous familial hypercholesterolemia.

https://www.webmd.com/cholesterol-management/homozygous-familial-hypercholesterolemia

ECGWaves. (n.d.). ST segment depression and elevation.

https://ecgwaves.com/st-segment-normal-abnormal-depression-elevation-causes

UT Southwestern Medical Center. (n.d.). Dr. Zahid Ahmad – Cholesterol.

https://utswmed.org/doctors/zahid-ahmad/cholesterol/

ScienceDirect. (n.d.). ST-segment depression.

https://www.sciencedirect.com/topics/medicine-and-dentistry/st-segment-depression

Stack Overflow. (n.d.). Feature selection before or after train-test split.

https://stackoverflow.com/questions/56308116/should-feature-selection-be-done-before-train-and-test-split-o
r-after

IBM. (n.d.). Data leakage in machine learning. https://www.ibm.com/think/topics/data-leakage-machine-learning

GeeksforGeeks. (n.d.). Drop rows with NaN values in pandas.

https://www.geeksforgeeks.org/how-to-drop-rows-with-nan-values-in-pandas-dataframe/

StatQuest with Josh Starmer. (n.d.). YouTube: Data preprocessing. https://youtu.be/KWrZ59nLLSg

Kaggle. (n.d.). Data preprocessing in machine learning.

https://www.kaggle.com/code/alirezahasannejad/data-preprocessing-in-machine-learning

GeeksforGeeks. (n.d.). ML – Feature scaling. https://www.geeksforgeeks.org/ml-feature-scaling-part-2/

GeeksforGeeks. (n.d.). Introduction to data cleansing. https://www.geeksforgeeks.org/data-cleansing-introduction/

GeeksforGeeks. (n.d.). Get a specific row in pandas dataframe.

https://www.geeksforgeeks.org/get-a-specific-row-in-a-given-pandas-dataframe/

GeeksforGeeks. (n.d.). Get column index from column name.

https://www.geeksforgeeks.org/get-column-index-from-column-name-of-a-given-pandas-dataframe/

Medium. (n.d.). Normalize before or after train-test split.
https://medium.com/@spinjosovsky/normalize-data-before-or-after-split-of-training-and-testing-data-7b8005f81e26

Kaggle Discussions. (n.d.). Feature scaling and imputation order.
https://www.kaggle.com/discussions/questions-and-answers/529881

LightGBM Documentation. (n.d.). https://lightgbm.readthedocs.io/en/stable/

Stack Overflow. (n.d.). Difference between StandardScaler and MinMaxScaler.
https://stackoverflow.com/questions/51237635/difference-between-standard-scaler-and-minmaxscaler

AIMind. (n.d.). Data leakage basics with examples in scikit-learn.
https://pub.aimind.so/data-leakage-basics-with-examples-in-scikit-learn-9c946a6f75b2

DataCamp. (n.d.). Feature selection in Python. https://www.datacamp.com/tutorial/feature-selection-python

DataScience StackExchange. (n.d.). Imputation vs standardization vs splitting.
https://datascience.stackexchange.com/questions/53138/which-comes-first-multiple-imputation-splitting-into-train-test-or-standardiz

GeeksforGeeks. (n.d.). Feature selection with Scikit-learn.
https://www.geeksforgeeks.org/feature-selection-in-python-with-scikit-learn/

Stack Overflow. (n.d.). Get feature names after SelectKBest.
https://stackoverflow.com/questions/39839112/the-easiest-way-for-getting-feature-names-after-running-selectkbest-in-scikit-le

GeeksforGeeks. (n.d.). Classification report and confusion matrix.
https://www.geeksforgeeks.org/compute-classification-report-and-confusion-matrix-in-python/

GeeksforGeeks. (n.d.). AUC-ROC curve. https://www.geeksforgeeks.org/auc-roc-curve/

IBM Developer. (n.d.). Confusion matrix tutorial in Python.
https://developer.ibm.com/tutorials/awb-confusion-matrix-python/

GeeksforGeeks. (n.d.). Confusion matrix in machine learning.
https://www.geeksforgeeks.org/confusion-matrix-machine-learning/

GeeksforGeeks. (n.d.). Heart disease prediction using logistic regression.
https://www.geeksforgeeks.org/ml-heart-disease-prediction-using-logistic-regression/

NCBI. (n.d.). Logistic regression overview. https://www.ncbi.nlm.nih.gov/books/NBK459364/

LinkedIn. (n.d.). Advantages of logistic regression.

https://www.linkedin.com/advice/1/what-advantages-using-logistic-regression-classification

DataCamp. (n.d.). Multilayer Perceptrons tutorial.

https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning

GeeksforGeeks. (n.d.). Plotting ROC curve in Python.

https://www.geeksforgeeks.org/how-to-plot-roc-curve-in-python/

Machine Learning Mastery. (n.d.). Tuning decision trees in XGBoost.

https://machinelearningmastery.com/tune-number-size-decision-trees-xgboost-python/