Heart Disease Prediction: A Machine Learning Approach

1624 Words

Enoshan Devchandra
University of Central Lancashire

Table of Contents

Introduction

Dataset and Objectives

This project uses a consolidated Kaggle dataset of 1,190 patient records from five heart disease databases: Cleveland, Hungarian, Switzerland, Long Beach VA, Statlog, featuring 11 clinical and demographic variables (Karna et al., 2024). Cardiovascular diseases (CVD) remain the leading cause of global mortality, highlighting the importance of predictive analytics to improve patient outcomes, optimize healthcare resources, and reduce costs (Gupta & Gupta, 2024; Mahajan & Kaushik, 2023).

The goal is to enhance a predictive model to achieve at least 85% accuracy through robust preprocessing, exploratory analysis, and experimentation.

Stakeholders

The primary stakeholders for this predictive modeling project are:

- **Clinicians and Healthcare Providers:** Improve diagnostic accuracy and timely patient care.
- **Healthcare Institutions and Policymakers:** Optimize resources, manage costs, and enhance healthcare services.
- **Patients and Public:** Receive personalized, efficient, and improved healthcare outcomes.

Research Questions

This project aims to address the following research questions:

1. *Which clinical and demographic features significantly predict heart disease?*
2. *How do these key features affect model accuracy and effectiveness? (Bandyopadhyay et al., 2023)*
3. *How effectively do preprocessing methods (SMOTE, Z-score normalization) improve fairness and mitigate demographic biases, especially gender imbalance?* (Mahajan & Kaushik, 2023; Rahman & Davis, 2013)
4. *Which algorithm among Logistic Regression, Random Forest, Decision Trees, K-Nearest Neighbors (KNN), or Artificial Neural Networks (ANN) best balances interpretability, computational efficiency, and predictive performance?* (Karna et al., 2024; Mahajan & Kaushik, 2023)

Expected Benefits to the User Group

- **Enhanced Clinical Decision-Making:** Identify high-risk patients early for targeted interventions, improving outcomes and minimizing unnecessary procedures (Gupta & Gupta, 2024).
- **Improved Healthcare Management:** Better resource allocation, operational efficiency, and strategic patient flow management (Karna et al., 2024).
- **Patient-Centric Care:** Personalized treatment approaches, reducing patient anxiety, improving satisfaction, and building trust (Mahajan & Kaushik, 2023).

Importing, Cleansing, Analyzing, and Visualizing Data

The project commenced by importing a consolidated Kaggle dataset into a Jupyter Notebook, utilizing Python's Pandas library. Preprocessing techniques are detailed in **Appendix C, Code Snippet 1.**

Demographic Bias

The dataset shows a significant gender imbalance with 909 male patients compared to 281 female patients, potentially biasing predictions toward male patterns. Addressing this issue through balancing techniques improves prediction equity and mitigates skewed outcomes (Mahajan & Kaushik, 2023).

Appendix A, Figure 1 shows individuals aged 50-60 have the highest prevalence of heart disease. The overlapping distribution between target classes (0 and 1) highlights the importance of preprocessing techniques to address variability and ensure fairness.

Heterogeneity in Features

Nominal Features

Utilizing one-hot encoding prevents ordinal misinterpretations and maintains categorical variable independence, crucial for model accuracy (Remeseiro & Bolon-Canedo, 2019).

Appendix A, Figure 2 shows the distribution of chest pain types across heart disease classes, emphasizing the need to transform this nominal feature into binary columns to avoid ordinal assumptions. *Chest pain type, resting ECG,* and *ST slope* were converted accordingly. **Appendix B, Table 1** shows the original features and their encoded columns.

Numeric Features

Descriptive statistics **(Appendix B, Table 2)** revealed considerable variability in key features like *resting blood pressure, cholesterol, and max heart rate*, necessitating z-score normalization to ensure consistent feature weighting. Although *oldpeak* displayed low variance, it was still standardized for alignment with other features.

Visual analysis further supported these findings. **Appendix A, Figure 3** displayed *cholesterol* variability across age groups, while **Appendix A, Figure 4** revealed significant *cholesterol* outliers, reinforcing the need for normalization. Z-score normalization addressed these disparities by scaling variables such as *age, cholesterol, resting BP, max heart rate,* and *oldpeak* to a mean of ~0 and standard deviation of ~1, as summarized in **Appendix B, Table 3**, aligning with best practices for model consistency (Tadist et al., 2019).

Binary Features

Exercise angina and *fasting blood sugar,* already in binary format, required no additional preprocessing.

Class Imbalance

The dataset exhibited slight class imbalance, with 629 instances of heart disease and 561 of no heart disease. SMOTE (Synthetic Minority Oversampling Technique) was applied, generating synthetic samples for the minority class. This method enhances the model's sensitivity and prediction accuracy by more accurately representing minority class patterns (Rahman & Davis, 2013). The class distribution before and after SMOTE is presented in **Appendix B, Table 4**.

Key Predictors

To determine the most influential predictors of heart disease, we employed p-value analysis and correlation heatmaps. **Appendix B, Table 5** details the p-values, which assess the statistical significance of each feature's relationship with the target variable. These relationships are further visualized in **Appendix A, Figure 5** through a correlation matrix.

Bandyopadhyay et al. supports the effectiveness of correlation-based approaches for ranking and validating predictors. Features like *ST slope, exercise-induced angina, chest pain type,*

maximum heart rate, and *oldpeak* showed both low p-values and strong correlation coefficients, highlighting their importance in prediction.

Predictive Model Development and Evaluation

Algorithm Selection and Experimentation

Several machine learning algorithms were evaluated for their strengths and dataset suitability.

Logistic Regression was chosen for its interpretability, simplicity and effectiveness in modeling crucial clinical variables such as ST slope and exercise-induced angina (Mahajan & Kaushik, 2023). Random Forest and Decision Trees were utilized to handle complex, non-linear interactions among variables like chest pain type and cholesterol levels, also providing valuable feature importance metrics to enhance model transparency (Karna et al., 2024). K-Nearest Neighbors (KNN) was selected for its proficiency in proximity-based classification, ideal for analyzing numerical data like resting blood pressure. Additionally, Artificial Neural Networks (ANN) were incorporated for their ability to model deep, non-linear relationships across various predictors, offering substantial computational power.

All models underwent training and evaluation with an 80:20 train-test split to ensure fair comparisons and assess generalization. This structured approach supported model selection, documented using Python and Scikit-learn. Detailed implementations are available in **Appendix C, Code Snippets 2-8**, ensuring model transparency and replicability.

Statistical Measures and Validation

Key metrics used for evaluating and validating model performance were as follows:

1. **Cross-validation:** Applied 10-fold cross-validation to evaluate consistency across data subsets and reduce biases from individual train-test splits **(Appendix B, Tables 6.1-6.2)**.

2. **Accuracy, Precision, Recall, F1-Score:** These metrics evaluated classification performance, capturing the models' balance of sensitivity and specificity **(Appendix B, Tables 7.1-7.2)**.

3. **ROC-AUC Scores:** Used to measure each model's ability to distinguish between classes, with higher AUC values indicating stronger predictive power **(Appendix A, Figures 6.1-6.5)**.

4. **Confusion Matrices:** Provided detailed insights into classification outcomes, such as true positives and false negatives, illustrating model strengths and weaknesses **(Appendix A, Figures 7.1-7.5; Appendix B, Tables 8.1-8.2)**.

5. **Feature Importance:** Different methods assessed feature significance, including coefficient values in Logistic Regression **(Appendix A, Figure 8.1)**, intrinsic importance scores in tree-based models **(Figures 8.2-8.3)**, and permutation importance in KNN and ANN **(Figures 8.4-8.5)**, identifying the most influential predictors driving model performance.

Overfitting and Underfitting Assessment

Model generalizability was assessed by comparing training and testing performance metrics. **Appendix B, Table 6.1**, shows that Logistic Regression and Multi-Layer Perceptron (MLP) models were initially well-fitted, with aligned training and test accuracies and stable cross-validation scores.

Random Forest and Decision Tree models showed signs of overfitting with perfect training scores (1.000) but lower test accuracies, indicating they memorized the training data. Conversely, the K-Nearest Neighbors (KNN) model underfitted, shown by low training and

test accuracies and comparatively low cross-validation scores, suggesting inadequate learning of data patterns.

To address these issues, hyperparameter tuning and feature evaluation were employed. Random Forest complexity was reduced by limiting tree depth and adjusting split parameters to enhance generalization. Decision Trees were pruned using both pre- and post-pruning techniques to simplify the model. KNN's number of neighbors was optimized through cross-validation, effectively balancing the bias-variance tradeoff.

As reflected in **Appendix B, Table 6.2**, these refinements improved the alignment between training and test performances and enhanced cross-validation stability, resulting in well-fitted models capable of reliably predicting unseen data.

Final Model Justification

The Random Forest algorithm was selected as the final model due to its superior performance, achieving the highest F1-score (0.91), precision (0.92), and recall (0.91), indicating an excellent sensitivity-specificity balance **(Appendix B, Table 7.2)**. It's ROC-AUC score of 0.96 **(Appendix A, Figure 6.2)** underscores its strong ability to differentiate between classes.

The confusion matrix **(Appendix B, Table 8.2)** demonstrates accurate classification across both heart disease and non-disease cases, with minimal false negatives critical in clinical contexts. Random Forest excelled in identifying critical predictors like *ST slope*, *chest pain type*, and *oldpeak* **(Appendix A, Figure 8.2)**, supporting both interpretability and clinical relevance. Moreover, it utilized a broader and balanced set of features compared to other models.

The model went through rigorous cross-validation, with overfitting effectively mitigated via tuned hyperparameters. Random Forest outperformed Logistic Regression, Decision Tree, KNN, and MLP across all key evaluation metrics, confirming its suitability for delivering reliable, interpretable, and actionable predictions, aligning with project objectives and stakeholder expectations.

Ethical Considerations

Machine learning in healthcare presents ethical challenges that must be addressed to ensure responsible and trustworthy model deployment.

Data Privacy and Protection

Although the dataset was anonymized and publicly sourced, clinical use of these models requires compliance with regulations like GDPR and HIPAA. These laws ensure secure handling of sensitive data, emphasizing consent, access, and confidentiality (Margam, 2023). Transparency in data management is essential for maintaining trust and ethical standards.

Fairness and Bias Mitigation

The dataset displayed a significant gender imbalance, with 909 male and 281 female patients, risking biased model outcomes. To address this, SMOTE was applied to balance the class distribution, promoting equity across predictions (Rahman & Davis, 2013). Future datasets should include broader demographic representation to improve fairness across underrepresented groups (Purnamaningsih et al., 2024).

Interpretability and Clinical Trust

Interpretability is crucial for clinical adoption. While Random Forest provided strong performance, its complexity may obscure decision pathways. To improve transparency,

feature importance diagrams highlighted key predictors (Mahajan & Kaushik, 2023). Clinician understanding and trust are essential for ethical deployment. Use of interpretability tools, such as SHAP, would enhance this transparency at the individual patient level.

Minimizing Harm from Prediction Errors

In clinical contexts, false negatives can delay urgent care, while false positives may cause undue anxiety or unnecessary procedures. The model underwent rigorous tuning and validation to reduce these risks (Gupta & Gupta, 2024). Ethical model development requires both accuracy and generalization to avoid diagnostic harm.

Recommendations and Conclusion

Practical Implications

The findings from this study can enhance healthcare delivery and policy-making with the following recommendations:

- **Clinical Integration:** Distribute analyses and reports to healthcare providers to improve the understanding of heart disease predictors, supporting enhanced decision-making and personalized care.
- **Policy Formulation:** Use insights from predictive analytics to guide policymakers and healthcare institutions in resource allocation and strategic planning.
- **Patient Awareness:** Launch educational campaigns to increase awareness on early detection and lifestyle changes to reduce heart disease risks.

Future Enhancements

To further refine our predictive model for heart disease:

- **Data Representativeness:** Enhance the dataset to better represent diverse populations by including a more balanced gender ratio and expanded demographic variables, improving fairness and effectiveness across different groups.
- **Algorithmic Refinement:** Exploring ensemble methods that combine the strengths of various models to enhance accuracy and reliability of predictions.
- **Real-World Testing:** Deploy the model in clinical settings to assess its real-world performance and refine it based on feedback.

Conclusion

This project highlights key heart disease predictors and establishes a foundation for future advancements in medical analytics. With continued refinement and broader demographic representation, the model has strong potential to improve cardiovascular care. By applying machine learning responsibly with transparency, fairness, and ethical integrity this work contributes meaningfully to the future of data-driven healthcare and public health.

References

Bandyopadhyay, S., Sarma, M., & Samanta, D. (2023). Cardiac Risk Factors Identification with Hybrid Statistical Approach and Prediction of Cardiovascular Disease with Machine Learning. *OCIT 2023 - 21st International Conference on Information Technology, Proceedings*, 41–46. https://doi.org/10.1109/OCIT59427.2023.10430934

Gupta, A., & Gupta, S. (2024). Enhanced Classification of Imbalanced Medical Datasets using Hybrid Data-Level, Cost-Sensitive and Ensemble Methods. *International Research Journal of Multidisciplinary Technovation*, *6*(3), 58–76. https://doi.org/10.54392/irjmt2435

Karna, V. V. R., Karna, V. R., Janamala, V., Devana, V. N. K. R., Ch, V. R. S., & Tummala, A. B. (2024). A Comprehensive Review on Heart Disease Risk Prediction using Machine Learning and Deep Learning Algorithms. In *Archives of Computational Methods in Engineering*. Springer Science and Business Media B.V. https://doi.org/10.1007/s11831-024-10194-4

Mahajan, A., & Kaushik, B. (2023). A Review of Machine Learning Algorithms and Feature Selection Techniques for Cardiovascular Disease Prediction: Insights and Implications. *2023 7th International Conference On Computing, Communication, Control And Automation, ICCUBEA 2023*. https://doi.org/10.1109/ICCUBEA58933.2023.10392135

Margam, R. (2023). *Ethics And Data Privacy: The Backbone of Trustworthy Healthcare Practices*. https://doi.org/10.59535/sehati.v1i2.115

Purnamaningsih, S. N. I., Ismono, J., Utami, I. S., Parlindungan, V., & Hanifah, S. N. (2024). The Challenges of Data Privacy Laws in the Age of Big Data: Balancing Security, Privacy, and Innovation. In *Join: Journal of Social Science* (Vol. 1, Issue 6). http://creativecommons.org/licenses/by/4.0/,whichpermitsunrestricteduse,providedtheoriginalauthorandsourcearecredited.https://ejournal.mellbaou.com/index.php/join/index

Rahman, M. M., & Davis, D. N. (2013). Addressing the Class Imbalance Problem in Medical Datasets. *International Journal of Machine Learning and Computing*, 224–228. https://doi.org/10.7763/ijmlc.2013.v3.307

Remeseiro, B., & Bolon-Canedo, V. (2019). A review of feature selection methods in medical applications. In *Computers in Biology and Medicine* (Vol. 112). Elsevier Ltd. https://doi.org/10.1016/j.compbiomed.2019.103375

Tadist, K., Najah, S., Nikolov, N. S., Mrabti, F., & Zahi, A. (2019). Feature selection methods and genomic big data: a systematic review. *Journal of Big Data*, *6*(1). https://doi.org/10.1186/s40537-019-0241-0

Appendix A

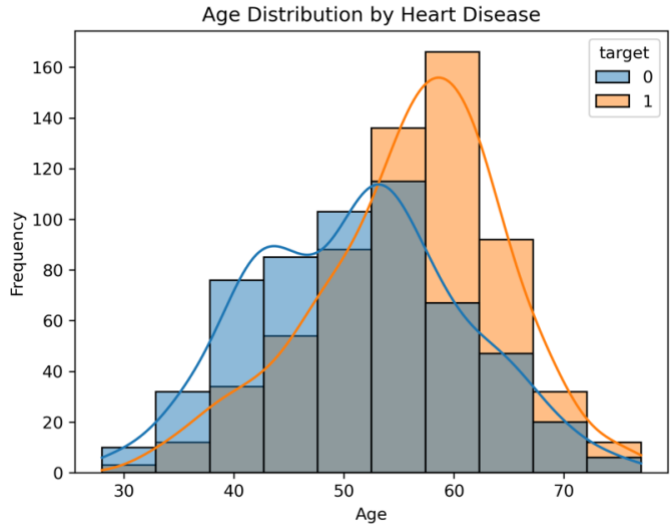


Figure 1 - Age Distribution by Heart Disease

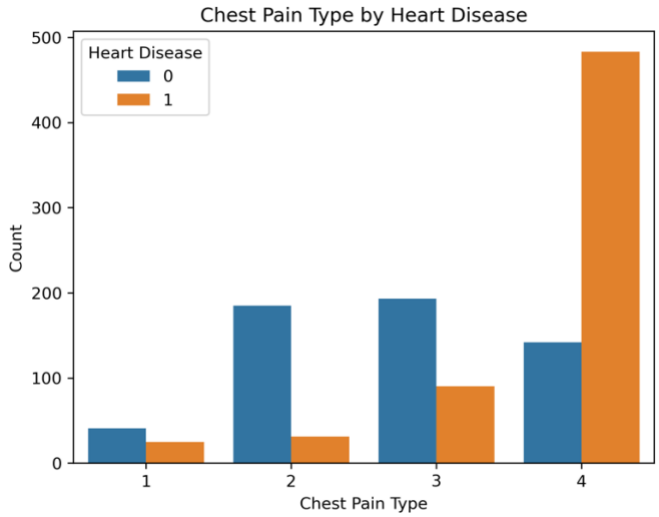


Figure 2 - Chest Pain Type by Heart Disease Bar Chart

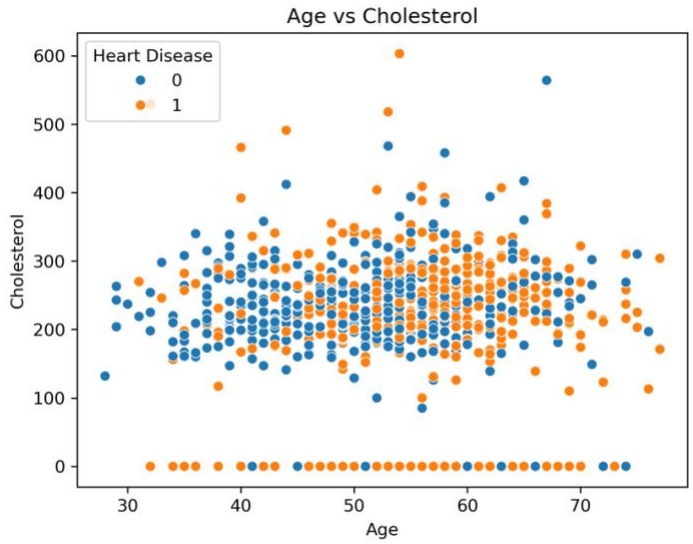


Figure 3 - Age vs Cholesterol Scatterplot

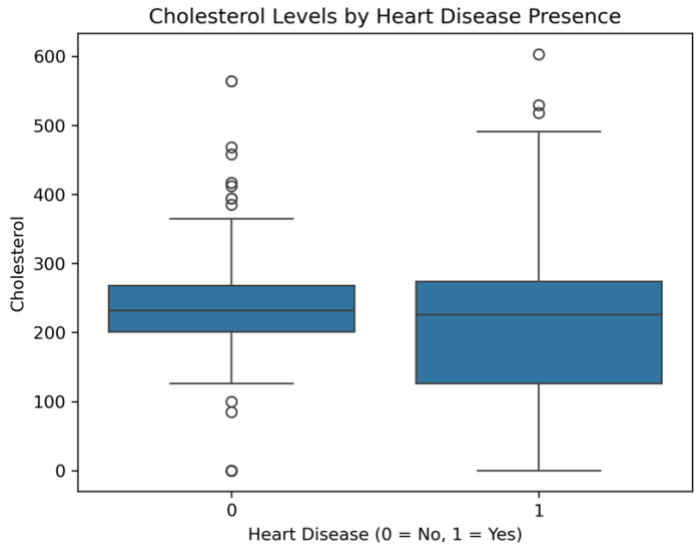


Figure 4 - Cholesterol Levels by Heart Disease Boxplot

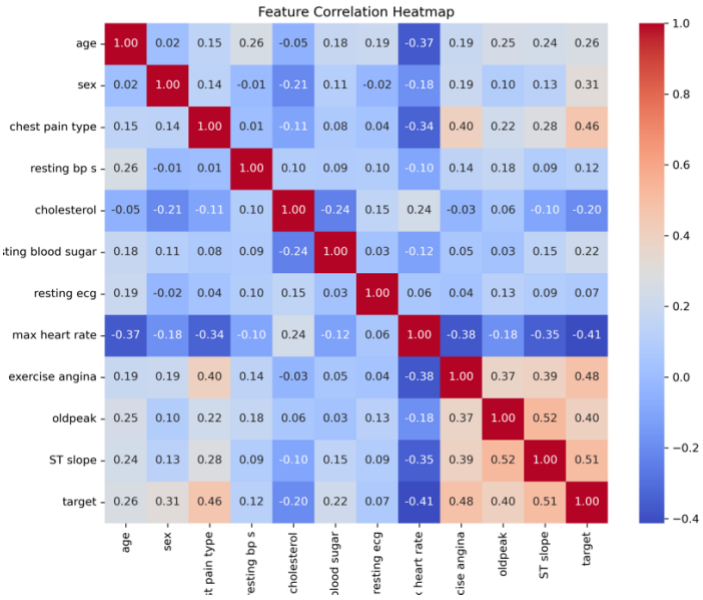


Figure 5 - Feature Correlation Heatmap

Figure 6 – Refined ROC Curves and AUC Scores

Figure 6.1 - ROC Curve and AUC Score for Logistic Regression

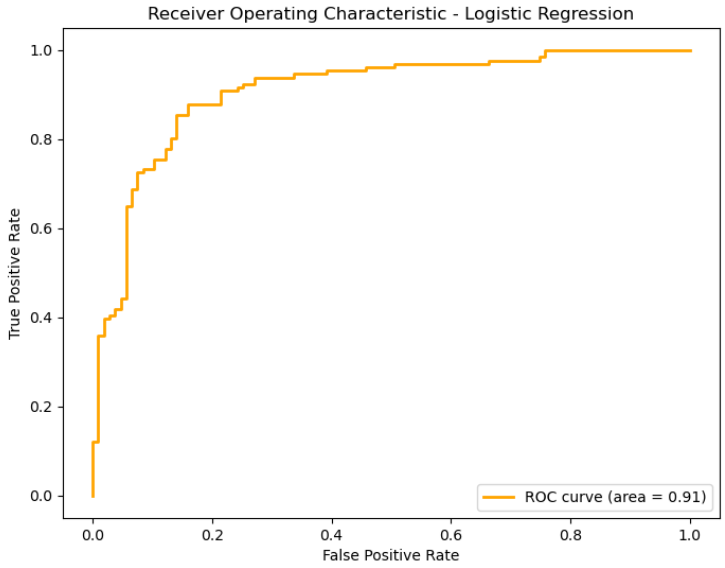


Figure 6.2 - ROC Curve and AUC Score for Random Forest

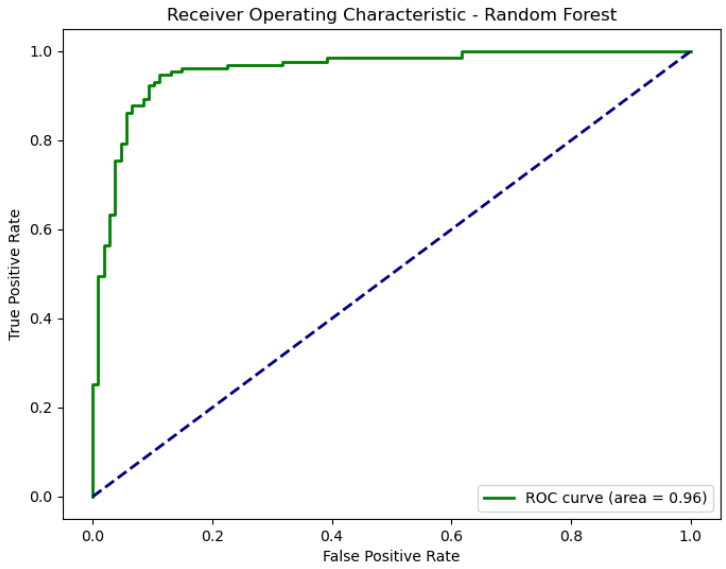


Figure 6.3 - ROC Curve and AUC Score for Decision Tree

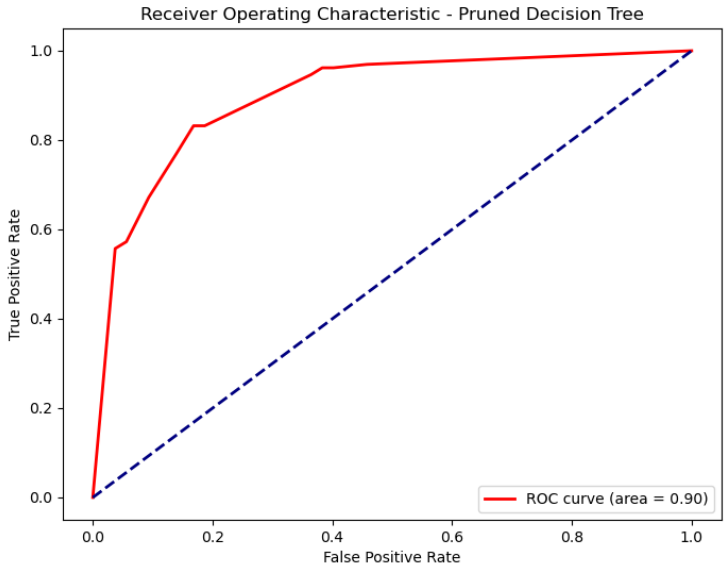


Figure 6.4 - ROC Curve and AUC Score for KNN

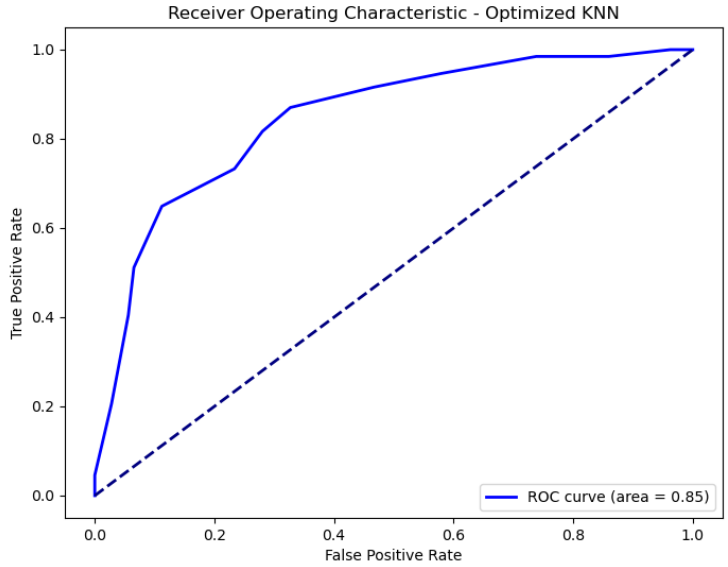


Figure 6.5 - ROC Curve and AUC Score for MLP

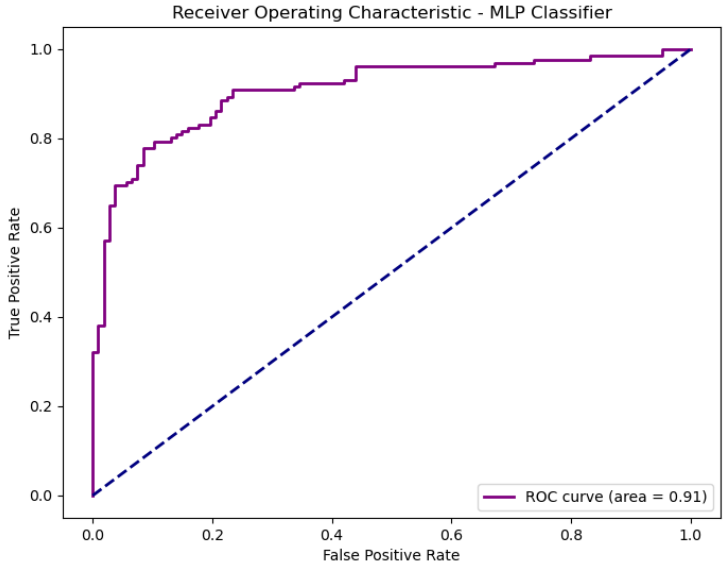


Figure 7 – Refined Confusion Matrices

Figure 7.1 - Confusion Matrix for Logistic Regression

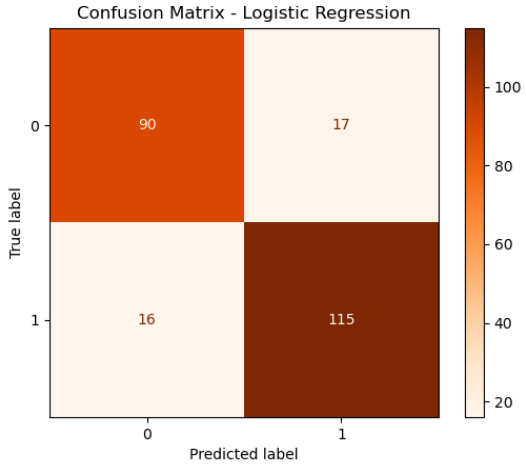


Figure 7.2 - Confusion Matrix for Random Forest

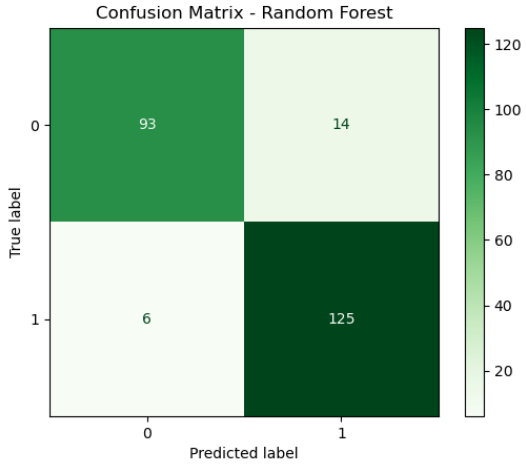


Figure 7.3 - Confusion Matrix for Decision Tree

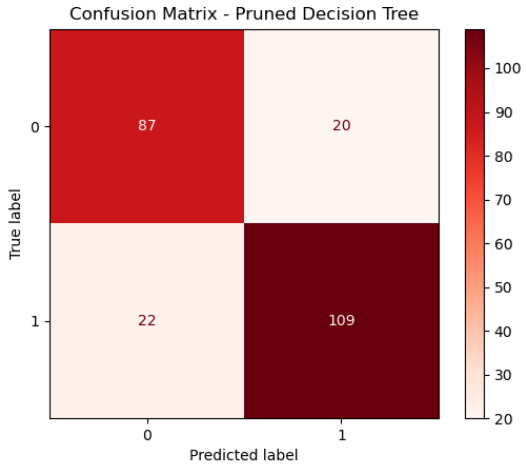


Figure 7.4 - Confusion Matrix for KNN

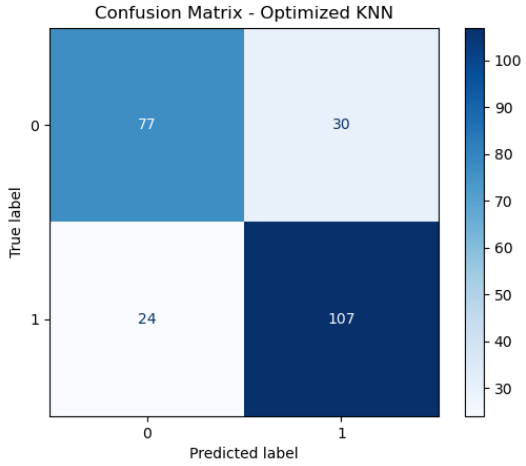


Figure 7.5 - Confusion Matrix for MLP

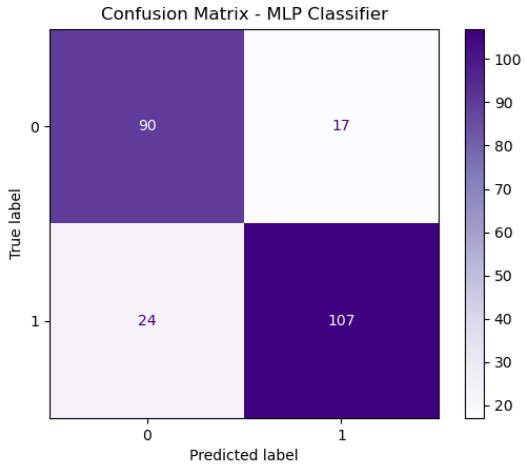


Figure 8 – Refined Feature Importance Diagrams

Figure 8.1 - Feature Importance for Logistic Regression

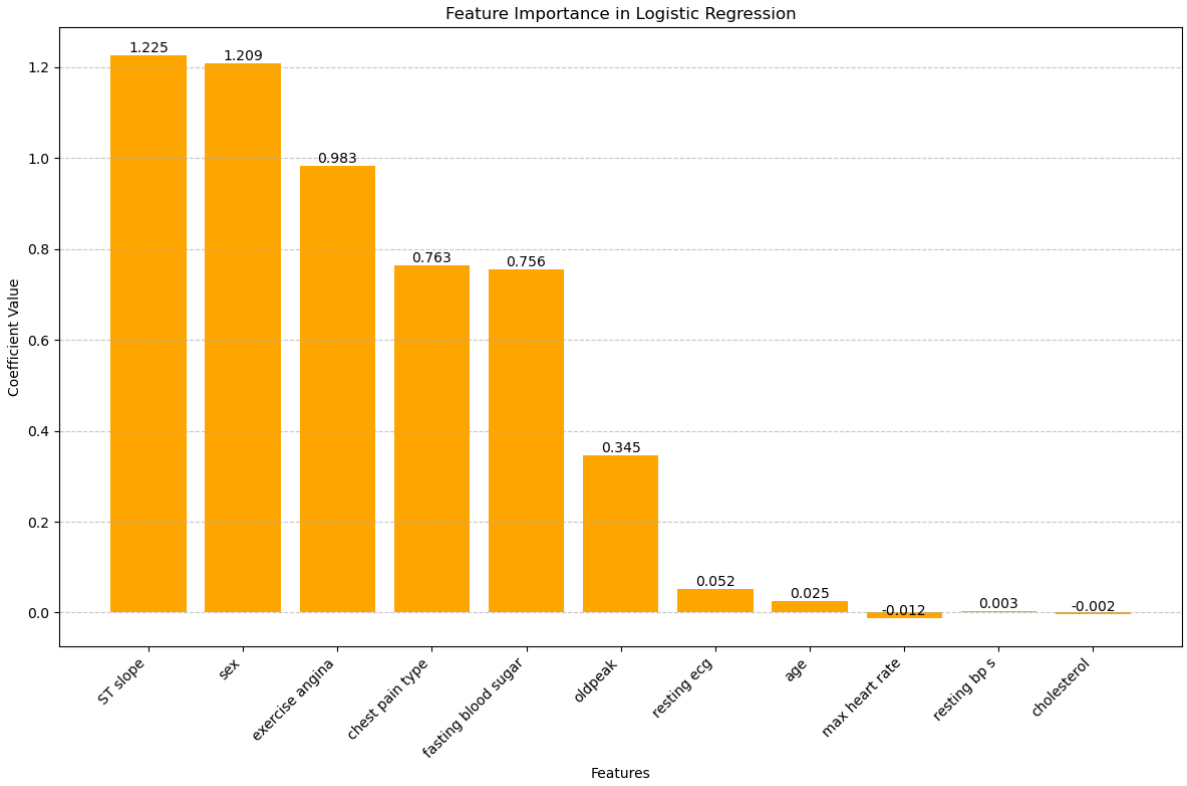


Figure 8.2 - Feature Importance for Random Forest

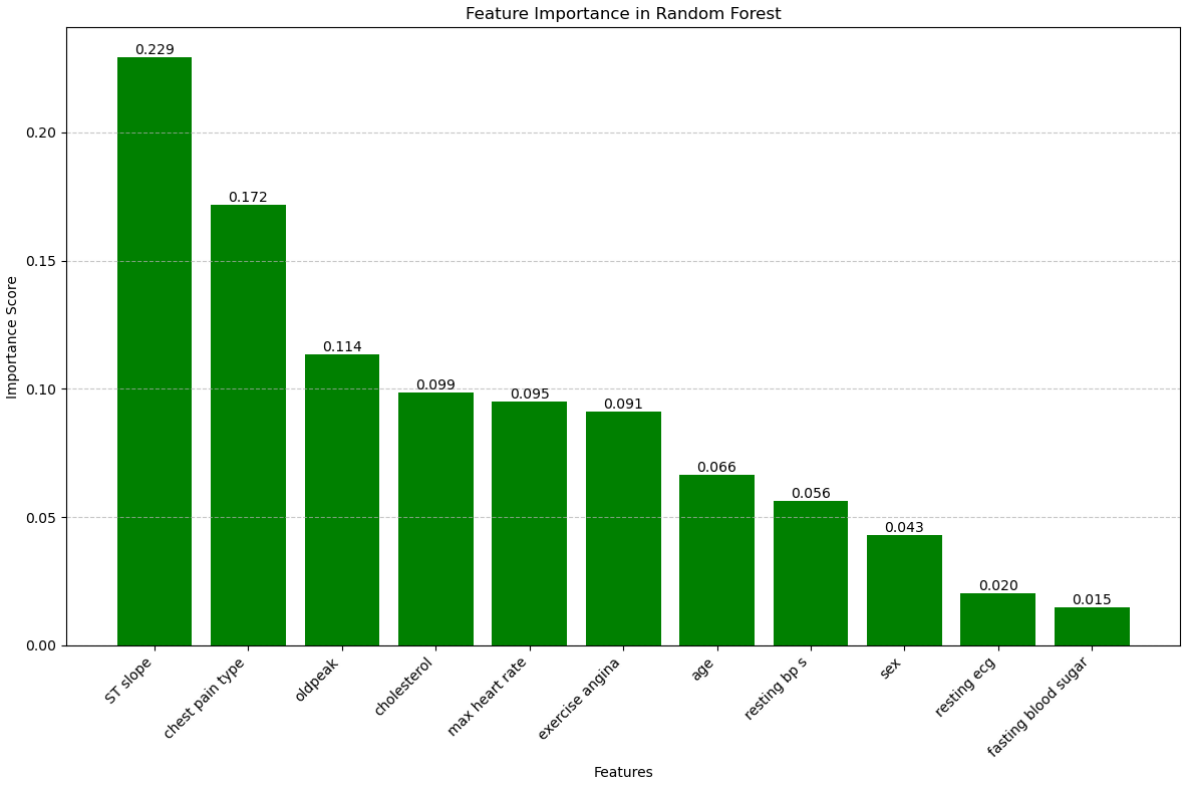


Figure 8.3 - Feature Importance for Decision Tree

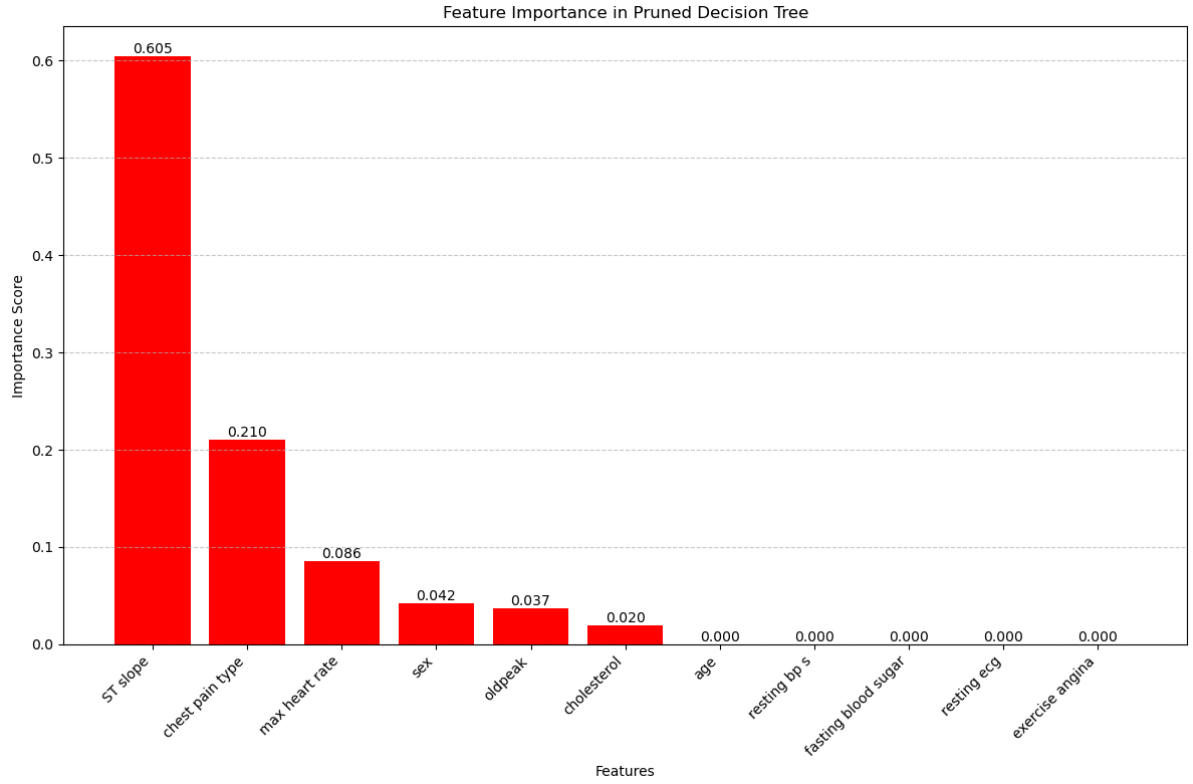


Figure 8.4 - Feature Importance for KNN

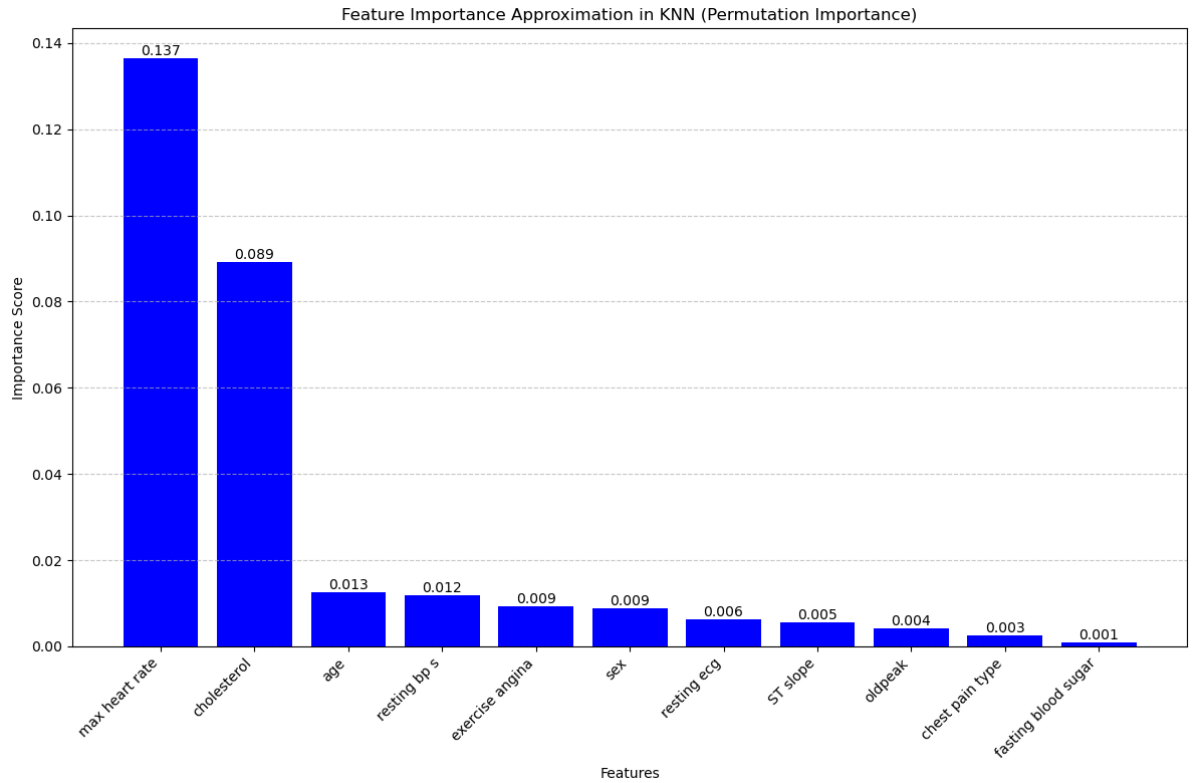
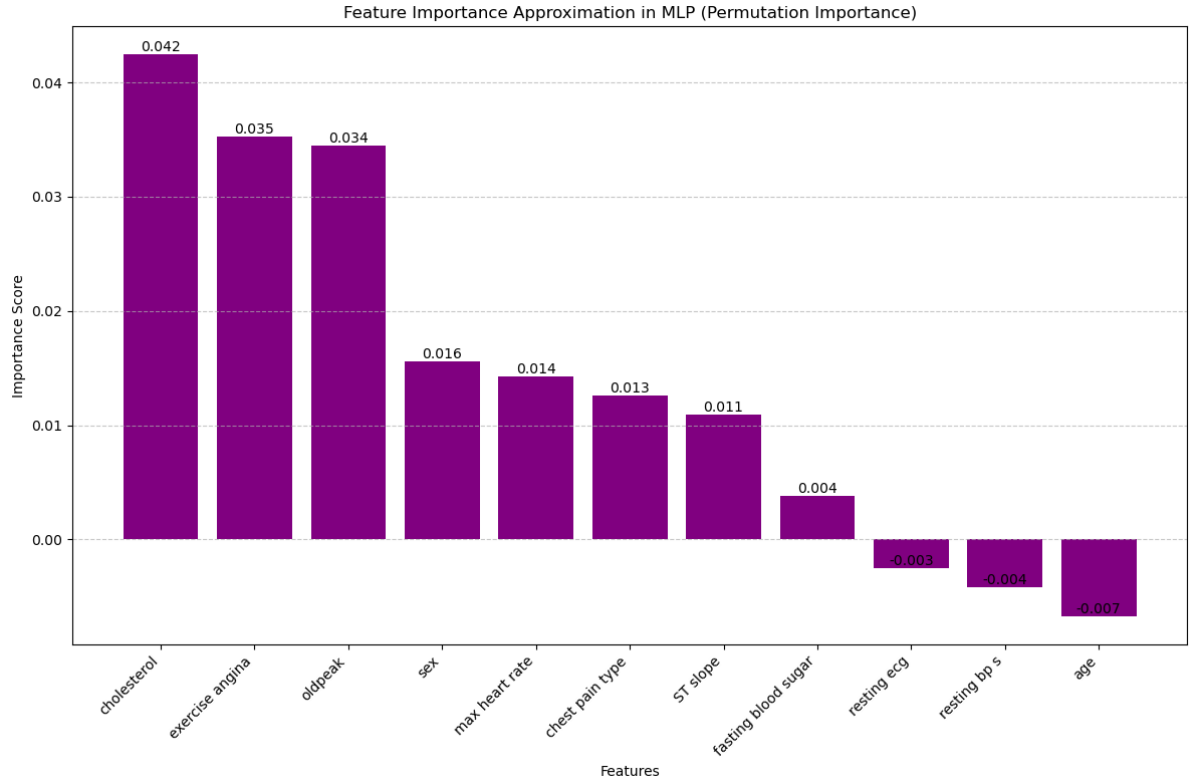


Figure 8.5 - Feature Importance for MLP



Appendix B

Table 1 - Original and Encoded Columns

Original Feature	Encoded Columns	Example Transformation
Chest Pain Type	chest_pain_2, chest_pain_3, chest_pain_4	chest pain type = 2 → chest_pain_2 = 1, chest_pain_3 = 0, chest_pain_4 = 0
Resting ECG	resting_ecg_1, resting_ecg_2	resting ECG = 1 → resting_ecg_1 = 1, resting_ecg_2 = 0
ST Slope	st_slope_1, st_slope_2, st_slope_3	ST slope = 3 → st_slope_3 = 1, st_slope_1 = 0, st_slope_2 = 0

Columns After One-Hot Encoding:

['age', 'sex', 'resting bp s', 'cholesterol', 'fasting blood sugar', 'max heart rate', 'exercise angina', 'oldpeak', 'target', 'chest_pain_2', 'chest_pain_3', 'chest_pain_4', 'resting_ecg_1', 'resting_ecg_2', 'st_slope_1', 'st_slope_2', 'st_slope_3']

Table 2 - Descriptive Statistics of Numeric Features

Feature	Mean	Variance	Standard Deviation
Age	53.72	87.58	9.36
Cholesterol	210.36	10286.12	101.42
Resting Blood Pressure	132.15	337.41	18.37
Max Heart Rate	139.73	651.15	25.52
Oldpeak	0.92	1.18	1.09

Table 3 - Z-Score Normalization Results

Feature	Mean	Standard Deviation	Min	Max
Age	~0	~1	-2.749	2.489
Cholesterol	~0	~1	-2.075	3.873
Resting BP S	~0	~1	-7.197	3.695
Max Heart Rate	~0	~1	-3.126	2.441
Oldpeak	~0	~1	-3.244	4.860

Table 4 - Class Distribution Before and After SMOTE

Class Distribution Before SMOTE	Class Distribution After SMOTE
Class 0 (No Heart Disease): 561	Class 0 (No Heart Disease): 629
Class 1 (Heart Disease): 629	Class 1 (Heart Disease): 629

Table 5 - P-Values and Correlation Coefficients for Predictive Features

Feature	P-Value with Target	Correlation Coefficient	Significance
ST slope	3.15e-78	High (0.51 in heatmap)	Most significant predictor.
Exercise angina	4.46e-70	High (0.48 in heatmap)	Strong predictor of heart disease.
Chest pain type	2.14e-63	High (0.46 in heatmap)	Strong categorical feature for diagnosis.
Max heart rate	2.69e-50	High	Strong indicator of cardiovascular stress.
Oldpeak	1.51e-46	High	Significant marker of ST depression.

Table 6 - Model Performance Evaluation

Table 6.1 - Initial Model Performance Evaluation

Model	10-fold Cross-Validation Scores	Average CV Score	Training Accuracy	Test Accuracy	Model Fit Assessment
Logistic Regression	[0.854, 0.844, 0.863, 0.811, 0.821, 0.821, 0.779, 0.832, 0.737, 0.832]	0.819	0.825	0.861	Well-Fitted
Random Forest	[0.906, 0.917, 0.937, 0.905, 0.916, 0.958, 0.916, 0.895, 0.884, 0.895]	0.913	1.000	0.945	Overfitted
Decision Tree	[0.875, 0.885, 0.842, 0.853, 0.853, 0.905, 0.863, 0.884, 0.821, 0.874]	0.866	1.000	0.899	Overfitted
KNN (k=13)	[0.740, 0.750, 0.705, 0.726, 0.653, 0.705, 0.674, 0.726, 0.695, 0.695]	0.707	0.788	0.718	Underfitted
MLP	[0.865, 0.813, 0.874, 0.779, 0.821, 0.821, 0.747, 0.811, 0.789, 0.779]	0.810	0.832	0.828	Well-Fitted

Table 6.2 - Refined Model Performance Evaluation

Model	10-fold Cross-Validation Scores	Average CV Score	Training Accuracy	Test Accuracy	Model Fit Assessment
Logistic Regression	[0.854, 0.844, 0.863, 0.811, 0.821, 0.821, 0.779, 0.832, 0.737, 0.832]	0.819	0.825	0.861	Well-Fitted
Random Forest	[0.917, 0.917, 0.905, 0.874, 0.874, 0.874, 0.863, 0.842, 0.842, 0.853]	0.876	0.940	0.916	Well-Fitted
Decision Tree	[0.813, 0.792, 0.811, 0.821, 0.747, 0.800, 0.737, 0.821, 0.737, 0.758]	0.784	0.846	0.824	Well-Fitted
KNN (k=13)	[0.781, 0.802, 0.779, 0.768, 0.695, 0.779, 0.705, 0.811, 0.705, 0.695]	0.752	0.771	0.773	Well-Fitted
MLP	[0.865, 0.813, 0.874, 0.779, 0.821, 0.821, 0.747, 0.811, 0.789, 0.779]	0.810	0.832	0.828	Well-Fitted

Table 7 - Classification Metrics Comparison

Table 7.1 - Initial Classification Metrics Comparison

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.86	0.86	0.86	238
Random Forest	0.95	0.94	0.94	238
Decision Tree	0.90	0.90	0.90	238
KNN (k=13)	0.72	0.72	0.72	238
MLP	0.83	0.83	0.83	238

Table 7.2 - Refined Classification Metrics Comparison

Model	Precision	Recall	F1-Score	Support
Logistic Regression	0.86	0.86	0.86	238
Random Forest	0.92	0.91	0.91	238
Decision Tree	0.82	0.82	0.82	238
KNN (k=13)	0.77	0.77	0.77	238
MLP	0.83	0.83	0.83	238

Table 8 - Confusion Matrix Overview

Table 8.1 - Initial Confusion Matrix Overview

Model		Predicted: No Heart Disease	Predicted: Heart Disease
Logistic Regression	**Actual: No Heart Disease**	90	17
	Actual: Heart Disease	16	115
Random Forest	**Actual: No Heart Disease**	98	9
	Actual: Heart Disease	4	127
Decision Tree	**Actual: No Heart Disease**	99	8
	Actual: Heart Disease	16	115
KNN (k=13)	**Actual: No Heart Disease**	76	31
	Actual: Heart Disease	24	107
MLP	**Actual: No Heart Disease**	90	17
	Actual: Heart Disease	24	107

Table 8.2 - Refined Confusion Matrix Overview

Model		Predicted: No Heart Disease	Predicted: Heart Disease
Logistic Regression	**Actual: No Heart Disease**	90	17
	Actual: Heart Disease	16	115
Random Forest	**Actual: No Heart Disease**	93	14
	Actual: Heart Disease	6	125
Decision Tree	**Actual: No Heart Disease**	87	20
	Actual: Heart Disease	22	109
KNN (k=13)	**Actual: No Heart Disease**	77	30
	Actual: Heart Disease	24	107
MLP	**Actual: No Heart Disease**	90	17
	Actual: Heart Disease	24	107

Appendix C

Code Snippet 1 - Data Preprocessing Techniques

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE

# Load the dataset
df = pd.read_csv('heart_statlog_cleveland_hungary_final.csv')

# Step 1: One-Hot Encoding
print("Original Columns:", df.columns.tolist())
df = pd.get_dummies(df, columns=['chest pain type', 'resting ecg', 'ST slope'],
                    prefix=['chest_pain', 'resting_ecg', 'st_slope'], drop_first=True)
print("\nColumns After One-Hot Encoding:", df.columns.tolist())
print("\nDataset After One-Hot Encoding (First 5 Rows):")
print(df.head())

# Step 2: Z-Score Normalization
numeric_features = ['age', 'cholesterol', 'resting bp s', 'max heart rate', 'oldpeak']
scaler = StandardScaler()
df[numeric_features] = scaler.fit_transform(df[numeric_features])

print("\nDataset After Z-Score Normalization (First 5 Rows):")
print(df[numeric_features].head())
print("\nSummary Statistics After Z-Score Normalization:")
print(df[numeric_features].describe())

# Step 3: SMOTE for Class Balancing
X = df.drop('target', axis=1)  # Features
y = df['target']  # Target variable

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Combine resampled features and target for inspection
df_resampled = pd.concat([pd.DataFrame(X_resampled, columns=X.columns), pd.DataFrame(y_resampled, columns=['target'])], axis=1)

print("\nClass Distribution Before SMOTE:")
print(y.value_counts())
print("\nClass Distribution After SMOTE:")
print(y_resampled.value_counts())
print("\nDataset After SMOTE (First 5 Rows):")
print(df_resampled.head())
```

Code Snippet 2 - Model Instantiation

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression(max_iter=5000)

# Random Forest
from sklearn.ensemble import RandomForestClassifier
forest_model = RandomForestClassifier(
    n_estimators=50, max_depth=10, min_samples_split=10,
    min_samples_leaf=5, max_features='sqrt', bootstrap=True, random_state=42)

# Decision Tree (Pruned)
from sklearn.tree import DecisionTreeClassifier
tree_model = DecisionTreeClassifier(
    max_depth=10, min_samples_split=10, min_samples_leaf=5, ccp_alpha=0.005, random_state=42)

# KNN (will be optimized)
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=best_k, weights='uniform', p=1)

# ANN
from sklearn.neural_network import MLPClassifier
ann_model = MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000, random_state=42)
```

Code Snippet 3 - Data Splitting (80:20 Train-Test Split)

```
from sklearn.model_selection import train_test_split

X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Code Snippet 4 - Cross-Validation (10-Fold)

```
# Logistic Regression
from sklearn.model_selection import cross_val_score

cross_val_scores = cross_val_score(logistic_model, X_train, y_train, cv=10)
print("Cross-validation scores:", cross_val_scores)
print("Mean CV score:", cross_val_scores.mean())

# Random Forest
from sklearn.model_selection import cross_val_score

cross_val_scores = cross_val_score(forest_model, X_train, y_train, cv=10)
print("Cross-validation scores:", cross_val_scores)
print("Mean CV score:", cross_val_scores.mean())

# Decision Tree
from sklearn.model_selection import cross_val_score

cross_val_scores = cross_val_score(tree_model, X_train, y_train, cv=10)
print("Cross-validation scores:", cross_val_scores)
print("Mean CV score:", cross_val_scores.mean())

# KNN
from sklearn.model_selection import cross_val_score

cross_val_scores = cross_val_score(knn_model, X_train, y_train, cv=10)
print("Cross-validation scores:", cross_val_scores)
print("Mean CV score:", cross_val_scores.mean())

# ANN
from sklearn.model_selection import cross_val_score

cross_val_scores = cross_val_score(ann_model, X_train, y_train, cv=10)
print("Cross-validation scores:", cross_val_scores)
print("Mean CV score:", cross_val_scores.mean())
```

Code Snippet 5 - Evaluation Metrics (Classification Report & Confusion Matrix)

```
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Logistic Regression
logistic_model.fit(X_train, y_train)
logistic_predictions = logistic_model.predict(X_test)
print("Classification Report - Logistic Regression:\n", classification_report(y_test, logistic_predictions))
print("Confusion Matrix - Logistic Regression:\n", confusion_matrix(y_test, logistic_predictions))
disp = ConfusionMatrixDisplay.from_predictions(y_test, logistic_predictions, cmap=plt.cm.Oranges)
disp.plot()
plt.title("Confusion Matrix - Logistic Regression")
plt.show()

# Random Forest
forest_model.fit(X_train, y_train)
forest_predictions = forest_model.predict(X_test)
print("Classification Report - Random Forest:\n", classification_report(y_test, forest_predictions))
print("Confusion Matrix - Random Forest:\n", confusion_matrix(y_test, forest_predictions))
disp = ConfusionMatrixDisplay.from_predictions(y_test, forest_predictions, cmap=plt.cm.Greens)
disp.plot()
plt.title("Confusion Matrix - Random Forest")
plt.show()

# Decision Tree
tree_model.fit(X_train, y_train)
tree_predictions = tree_model.predict(X_test)
print("Classification Report - Decision Tree:\n", classification_report(y_test, tree_predictions))
print("Confusion Matrix - Decision Tree:\n", confusion_matrix(y_test, tree_predictions))
disp = ConfusionMatrixDisplay.from_predictions(y_test, tree_predictions, cmap=plt.cm.Reds)
disp.plot()
plt.title("Confusion Matrix - Decision Tree")
plt.show()
```

```
# KNN
knn_model.fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)
print("Classification Report - KNN:\n", classification_report(y_test, knn_predictions))
print("Confusion Matrix - KNN:\n", confusion_matrix(y_test, knn_predictions))
disp = ConfusionMatrixDisplay.from_predictions(y_test, knn_predictions, cmap=plt.cm.Blues)
disp.plot()
plt.title("Confusion Matrix - KNN")
plt.show()

# ANN
ann_model.fit(X_train, y_train)
ann_predictions = ann_model.predict(X_test)
print("Classification Report - MLP:\n", classification_report(y_test, ann_predictions))
print("Confusion Matrix - MLP:\n", confusion_matrix(y_test, ann_predictions))
disp = ConfusionMatrixDisplay.from_predictions(y_test, ann_predictions, cmap=plt.cm.Purples)
disp.plot()
plt.title("Confusion Matrix - MLP Classifier")
plt.show()
```

Code Snippet 6 - ROC Curve Plotting

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Logistic Regression
fpr_lr, tpr_lr, _ = roc_curve(y_test, logistic_model.predict_proba(X_test)[:, 1])
roc_auc_lr = auc(fpr_lr, tpr_lr)
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, color='orange', lw=2, label=f'ROC curve (area = {roc_auc_lr:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend(loc="lower right")
plt.show()

# Random Forest
fpr_rf, tpr_rf, _ = roc_curve(y_test, forest_model.predict_proba(X_test)[:, 1])
roc_auc_rf = auc(fpr_rf, tpr_rf)
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='green', lw=2, label=f'ROC curve (area = {roc_auc_rf:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend(loc="lower right")
plt.show()

# Decision Tree
fpr_dt, tpr_dt, _ = roc_curve(y_test, tree_model.predict_proba(X_test)[:, 1])
roc_auc_dt = auc(fpr_dt, tpr_dt)
plt.figure(figsize=(8, 6))
plt.plot(fpr_dt, tpr_dt, color='red', lw=2, label=f'ROC curve (area = {roc_auc_dt:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree')
plt.legend(loc="lower right")
plt.show()
```

```
# KNN
fpr_knn, tpr_knn, _ = roc_curve(y_test, knn_model.predict_proba(X_test)[:, 1])
roc_auc_knn = auc(fpr_knn, tpr_knn)
plt.figure(figsize=(8, 6))
plt.plot(fpr_knn, tpr_knn, color='blue', lw=2, label=f'ROC curve (area = {roc_auc_knn:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - KNN')
plt.legend(loc="lower right")
plt.show()

# ANN
fpr_ann, tpr_ann, _ = roc_curve(y_test, ann_model.predict_proba(X_test)[:, 1])
roc_auc_ann = auc(fpr_ann, tpr_ann)
plt.figure(figsize=(8, 6))
plt.plot(fpr_ann, tpr_ann, color='purple', lw=2, label=f'ROC curve (area = {roc_auc_ann:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - MLP Classifier')
plt.legend(loc="lower right")
plt.show()
```

Code Snippet 7 - Model Tuning (Mitigating Overfitting/Underfitting)

```
# Random Forest
# Initialize with anti-overfitting settings
forest_model = RandomForestClassifier(
    n_estimators=50,            # Reduce number of trees to prevent excessive complexity
    max_depth=10,               # Limit tree depth to prevent overfitting
    min_samples_split=10,       # A node must have at least 10 samples before splitting
    min_samples_leaf=5,         # A leaf must have at least 5 samples
    max_features='sqrt',        # Use sqrt(number of features) for best generalization
    bootstrap=True,             # Use bootstrapping to improve generalization
    random_state=42             # Ensure reproducibility
)

# Decision Tree
# Initialize with Pruning Parameters
tree_model = DecisionTreeClassifier(
    random_state=42,
    max_depth=10,               # Restrict tree depth to prevent overfitting
    min_samples_split=10,       # Require at least 10 samples to split a node
    min_samples_leaf=5,         # Require at least 5 samples per leaf
    ccp_alpha=0.005             # Post-pruning to remove insignificant branches
)


# KNN
# Optimized to find the Best 'k' to Prevent Overfitting & Underfitting
k_values = list(range(7, 14, 2))  # Odd values from 7 to 13 to prevent ties
cv_scores = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k, weights='uniform', p=1)  # Euclidean distance
    scores = cross_val_score(knn, X_train, y_train, cv=10)
    cv_scores.append(scores.mean())

# Find the best k (Avoids k=1 to prevent overfitting)
best_k = k_values[np.argmax(cv_scores)]
print(f"Best k found: {best_k}")

# Initialize KNN with Optimal Hyperparameters
knn_model = KNeighborsClassifier(n_neighbors=best_k, weights='uniform', p=1)  # Uses uniform weighting
```

Code Snippet 8 - Feature Importance Extraction and Plotting

```
# Logistic Regression
# Extract and visualize feature importance from the logistic regression coefficients
feature_names = X_train.columns  # Assumes X_train is a DataFrame
coefficients = logistic_model.coef_[0]
feature_importance = pd.DataFrame(coefficients, index=feature_names, columns=['Coefficient'])
feature_importance['abs_value'] = feature_importance['Coefficient'].abs()
feature_importance = feature_importance.sort_values('abs_value', ascending=False)

# Visualizing Feature Importance with Value Labels
plt.figure(figsize=(12, 8))
bars = plt.bar(feature_importance.index, feature_importance['Coefficient'], color='orange')

# Adding value labels on top of the bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, f'{yval:.3f}', ha='center', va='bottom', fontsize=10)

plt.title('Feature Importance in Logistic Regression')
plt.xlabel('Features')
plt.ylabel('Coefficient Value')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()


# Random Forest
# Extract and visualize feature importance from the Random Forest model
feature_names = X_train.columns  # Assuming X_train is a DataFrame
feature_importance = pd.DataFrame(forest_model.feature_importances_, index=feature_names, columns=['Importance'])
feature_importance = feature_importance.sort_values('Importance', ascending=False)  # Sorting in descending order

# Visualizing Feature Importance with Value Labels
plt.figure(figsize=(12, 8))
bars = plt.bar(feature_importance.index, feature_importance['Importance'], color='green')

# Adding value labels on top of the bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, f'{yval:.3f}', ha='center', va='bottom', fontsize=10)

plt.title('Feature Importance in Random Forest')
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
# Decision Tree
# Extract and Visualize Feature Importance from the Decision Tree Model
feature_names = X_train.columns  # Assuming X_train is a DataFrame
feature_importance = pd.DataFrame(tree_model.feature_importances_, index=feature_names, columns=['Importance'])
feature_importance = feature_importance.sort_values('Importance', ascending=False)  # Sorting in descending order

# Visualizing Feature Importance with Value Labels
plt.figure(figsize=(12, 8))
bars = plt.bar(feature_importance.index, feature_importance['Importance'], color='red')

# Adding value labels on top of the bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, f'{yval:.3f}', ha='center', va='bottom', fontsize=10)

plt.title('Feature Importance in Pruned Decision Tree')
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
# KNN
# Feature Importance (Permutation Importance)
perm_importance = permutation_importance(knn_model, X_test, y_test, scoring='accuracy', n_repeats=10, random_state=42)
feature_importance = pd.DataFrame(perm_importance.importances_mean, index=X_train.columns, columns=['Importance'])
feature_importance = feature_importance.sort_values('Importance', ascending=False)

# Visualizing Feature Importance with Value Labels
plt.figure(figsize=(12, 8))
bars = plt.bar(feature_importance.index, feature_importance['Importance'], color='blue')

# Adding value labels on top of the bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, f'{yval:.3f}', ha='center', va='bottom', fontsize=10)

plt.title('Feature Importance Approximation in KNN (Permutation Importance)')
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()


# ANN
# Approximate Feature Importance for MLP using Permutation Importance
perm_importance = permutation_importance(ann_model, X_test, y_test, scoring='accuracy', n_repeats=10, random_state=42)
feature_importance = pd.DataFrame(perm_importance.importances_mean, index=X_train.columns, columns=['Importance'])
feature_importance = feature_importance.sort_values('Importance', ascending=False)

# Visualizing Feature Importance with Value Labels
plt.figure(figsize=(12, 8))
bars = plt.bar(feature_importance.index, feature_importance['Importance'], color='purple')

# Adding value labels on top of the bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, f'{yval:.3f}', ha='center', va='bottom', fontsize=10)

plt.title('Feature Importance Approximation in MLP (Permutation Importance)')
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```