**Title Page**

**Heart Disease Prediction Using Machine Learning**

**Project Title:** Heart Disease Prediction Using Machine Learning

**Prepared By:** D.Tejaswini– Dhanekula Institute Of Engineering And Technology (DIET)

**Submitted To:** Atharva Pandey – IIDT Blackbucks

**Date:** 27-07-2024

### **Abstract**

Heart disease is one of the leading causes of mortality worldwide, necessitating early and accurate diagnosis for effective treatment and management. This project aims to predict the presence of heart disease using machine learning techniques. Utilizing a dataset from Kaggle, which contains various medical attributes relevant to heart disease, we employ a Random Forest algorithm to build a predictive model. The dataset comprises 303 instances with 14 features including age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise-induced angina, old peak, slope of the peak exercise ST segment, number of major vessels colored by fluoroscopy, thalassemia, and the target variable indicating the presence or absence of heart disease.

The process begins with data collection and preprocessing, involving loading the data, handling missing values, and encoding categorical features. This is followed by exploratory data analysis (EDA) to understand the distribution and relationships of the attributes. The data is then split into training and testing sets, maintaining the distribution of the target variable using stratified sampling.

A Random Forest classifier is trained on the training set, and its performance is evaluated using accuracy, precision, recall, and F1-score metrics. The trained model is then used to build a predictive system capable of determining the likelihood of heart disease based on new patient data.

The Random Forest algorithm was chosen for its high accuracy, robustness to overfitting, ability to handle missing values, and capability to capture non-linear relationships in the data. The model achieved an accuracy of 85.1% on the training data and 81.9% on the test data, demonstrating its effectiveness in predicting heart disease.

This project highlights the potential of machine learning in healthcare, particularly in the early diagnosis of heart disease. Future work can involve enhancing the model's performance by incorporating more sophisticated algorithms, tuning hyperparameters, and using larger, more diverse datasets. The ultimate goal is to integrate such predictive models into clinical settings to assist healthcare professionals in making timely and informed decisions, thereby improving patient outcomes.

**Table of Contents**

**1. Introduction**

* Overview of the Project
* Importance of Heart Disease Prediction
* Machine Learning in Healthcare

**2. Data Collection and Processing**

* Data Source
* Data Description
* Data Loading
* Data Exploration

**3. Data Preprocessing**

* Handling Missing Values
* Encoding Categorical Features
* Feature Scaling

**4. Exploratory Data Analysis (EDA)**

* Statistical Summary
* Data Visualization
  + Age Distribution
  + Gender Distribution
  + Chest Pain Type Distribution
  + Target Variable Distribution

**5. Splitting the Data**

* Features and Target Separation
* Train-Test Split

**6. Model Selection and Training**

* Overview of Algorithms
* Why Random Forest?
* Training the Random Forest Model

**7. Model Evaluation**

* Accuracy on Training Data
* Accuracy on Test Data
* Confusion Matrix
* Precision, Recall, F1 Score

**8. Building a Predictive System**

* Input Data Preparation
* Making Predictions
* Interpreting Results

**9. Conclusion**

* Summary of Findings
* Importance of the Model
* Future Work

**10. References**

### 1. Introduction

**Overview of the Project**

Heart disease is a leading cause of death worldwide. Early diagnosis and treatment are crucial for managing heart disease and improving patient outcomes. Machine learning provides powerful tools for analyzing large datasets and making predictions that can aid in early diagnosis.

**Importance of Heart Disease Prediction**

Predicting heart disease at an early stage can save lives by allowing timely intervention and treatment. Accurate prediction models can help healthcare professionals make informed decisions and provide better care.

**Machine Learning in Healthcare**

Machine learning algorithms can process complex datasets to identify patterns and make predictions. In healthcare, these models can be used for various applications, including disease prediction, patient monitoring, and personalized treatment planning.

**2. Data Collection and Processing**

**Data Source**

The dataset used for this project is the Heart Disease dataset available on Kaggle.

**Data Description**

The dataset contains 14 attributes, including age, gender, chest pain type, resting blood pressure, cholesterol levels, fasting blood sugar, and the target variable indicating the presence or absence of heart disease.

**Data Loading**

**Code:**

import pandas as pd

heart\_data = pd.read\_csv('/content/heart\_disease\_data.csv')

**Data Exploration**

**Code:**

# Display the first 5 rows

print(heart\_data.head())

# Display the last 5 rows

print(heart\_data.tail())

# Retrieve the shape of the dataset

print(heart\_data.shape)

# Get basic information about the dataset

print(heart\_data.info())

# Check for any missing values

print(heart\_data.isnull().sum())

# Generate statistical measures

print(heart\_data.describe())

# Check the distribution of the target variable

print(heart\_data['target'].value\_counts())

**3. Data Preprocessing**

**Handling Missing Values**

The dataset does not contain any missing values.

**Encoding Categorical Features**

Categorical features such as gender and chest pain type need to be encoded into numerical values.

**Code:**

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

heart\_data['sex'] = encoder.fit\_transform(heart\_data['sex'])

heart\_data['cp'] = encoder.fit\_transform(heart\_data['cp'])

heart\_data['restecg'] = encoder.fit\_transform(heart\_data['restecg'])

heart\_data['exang'] = encoder.fit\_transform(heart\_data['exang'])

heart\_data['slope'] = encoder.fit\_transform(heart\_data['slope'])

heart\_data['thal'] = encoder.fit\_transform(heart\_data['thal'])

**Feature Scaling**

Feature scaling is important to ensure that all features contribute equally to the model.

**Code:**

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled\_features = scaler.fit\_transform(heart\_data.drop(columns='target'))

**4. Exploratory Data Analysis (EDA)**

**Statistical Summary**

A detailed statistical summary of the dataset.

1. **Importing Dependencies**:
   * Import the necessary libraries for data manipulation, model building, and evaluation.

**Code:**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

1. **Data Collection and Processing**:
   * Load the heart disease dataset from a CSV file into a Pandas DataFrame.

**Code:**

heart\_data = pd.read\_csv('/content/heart\_disease\_data.csv')

* + Display the first and last 5 rows of the dataset to understand its structure.

**Code:**

heart\_data.head()

heart\_data.tail()

* + Retrieve the shape of the dataset to know the number of rows and columns.

**Code:**

heart\_data.shape

* + Get basic information about the dataset, including data types and non-null counts.

**Code:**

heart\_data.info()

* + Check for any missing values in the dataset.

**Code:**

heart\_data.isnull().sum()

* + Generate statistical measures (mean, standard deviation, etc.) of the data.

**Code:**

heart\_data.describe()

* + Check the distribution of the target variable (presence or absence of heart disease).

**Code:**

heart\_data['target'].value\_counts()

1. **Splitting Features and Target**:
   * Separate the features (X) and the target variable (Y) from the dataset.

**Code:**

X = heart\_data.drop(columns='target', axis=1)

Y = heart\_data['target']

1. **Splitting Data into Training and Test Sets**:
   * Split the data into training and testing sets using an 80-20 split. Use stratified sampling to ensure that the distribution of the target variable is consistent across .

**Code:**

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, stratify=Y, random\_state=2)

print(X.shape, X\_train.shape, X\_test.shape)

#### Detailed Statistical Analysis:

* + **Data Distribution**: It is crucial to maintain the distribution of the target variable in both the training and test sets. Stratified sampling ensures that the proportion of positive (heart disease) and negative (no heart disease) cases in the dataset is preserved in both sets.

**Code**:

Y\_train.value\_counts(normalize=True)

Y\_test.value\_counts(normalize=True)

This code will output the proportion of each class in the training and test sets, confirming that the stratified split maintains the class distribution:

1 0.544

0 0.456

Name: target, dtype: float64

* + **Class Balance**: Checking for class balance in the training data is essential to ensure that the model is not biased towards the majority class. In this case, the classes are relatively balanced, which is advantageous for model training.

Y\_train.value\_counts().plot(kind='bar', title='Class Distribution in Training Set')

* + **Feature Analysis**: Analyzing the features to understand their distribution and potential correlations with the target variable. For example, visualizing the distribution of age across the two classes can provide insights into its predictive power.

**Code**:

import seaborn as sns

import matplotlib.pyplot as plt

sns.boxplot(x='target', y='age', data=heart\_data)

plt.title('Age Distribution by Heart Disease Status')

plt.show()

This boxplot will show the distribution of ages for patients with and without heart disease, highlighting any significant differences.

* + **Correlation Matrix**: A correlation matrix helps identify the relationships between features and the target variable. Features highly correlated with the target are potentially more useful for prediction.

**Code**:

correlation\_matrix = heart\_data.corr()

plt.figure(figsize=(12, 10))

sns.heatmap(correlation\_matrix, annot=True, fmt='.2f', cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()

This heatmap will visualize the correlation between features, allowing us to identify highly correlated pairs, which can inform feature selection and engineering.

1. **Model Training**:
   * Initialize a Random Forest classifier.

**Code:**

model = RandomForestClassifier(random\_state=2)

Train the Random Forest classifier using the training data (X\_train and Y\_train).

model.fit(X\_train, Y\_train)

1. **Model Evaluation**:
   * Predict the target variable for the training data and calculate the accuracy score.

**Code:**

X\_train\_prediction = model.predict(X\_train)

training\_data\_accuracy = accuracy\_score(X\_train\_prediction, Y\_train)

print('Accuracy on Training data : ', training\_data\_accuracy)

* + Predict the target variable for the testing data and calculate the accuracy score.

**Code:**

X\_test\_prediction = model.predict(X\_test)

test\_data\_accuracy = accuracy\_score(X\_test\_prediction, Y\_test)

print('Accuracy on Test data : ', test\_data\_accuracy)

* + Calculate additional evaluation metrics such as precision, recall, and F1-score for a comprehensive assessment of the model's performance.

**Code:**

precision = precision\_score(Y\_test, X\_test\_prediction)

recall = recall\_score(Y\_test, X\_test\_prediction)

f1 = f1\_score(Y\_test, X\_test\_prediction)

print(f'Precision: {precision}')

print(f'Recall: {recall}')

print(f'F1 Score: {f1}')

1. **Building a Predictive System**:
   * Create a sample input data tuple representing a single instance of a patient's health metrics.

**Code:**

input\_data = (54, 1, 0, 120, 188, 0, 1, 113, 0, 1.4, 1, 1, 3)

* + Convert the input data to a NumPy array.

input\_data\_as\_numpy\_array = np.asarray(input\_data)

* + Reshape the NumPy array to match the model's expected input format.

**Code:**

input\_data\_reshaped = input\_data\_as\_numpy\_array.reshape(1, -1)

* + Use the trained model to predict whether the patient has heart disease based on the input data.

**Code:**

prediction = model.predict(input\_data\_reshaped)

print(prediction)

* + Print the prediction result and an appropriate message indicating whether the patient has heart disease or not.

**Code:**

if (prediction[0] == 0):

print('The Person does not have a Heart Disease')

else:

print('The Person has Heart Disease')

**Data Visualization**

Visualizations help in understanding the distribution and relationships between different attributes.

**Age Distribution**

**Code:**

import matplotlib.pyplot as plt

plt.hist(heart\_data['age'], bins=20, color='blue', edgecolor='black')

plt.title('Age Distribution')

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.show()

**Gender Distribution**

**Code:**

plt.bar(heart\_data['sex'].value\_counts().index, heart\_data['sex'].value\_counts().values, color=['blue', 'pink'])

plt.title('Gender Distribution')

plt.xlabel('Gender')

plt.ylabel('Frequency')

plt.xticks([0, 1], ['Male', 'Female'])

plt.show()

**Chest Pain Type Distribution**

**Code:**

plt.bar(heart\_data['cp'].value\_counts().index, heart\_data['cp'].value\_counts().values, color='orange')

plt.title('Chest Pain Type Distribution')

plt.xlabel('Chest Pain Type')

plt.ylabel('Frequency')

plt.show()

**Target Variable Distribution**

**Code:**

plt.bar(heart\_data['target'].value\_counts().index, heart\_data['target'].value\_counts().values, color=['green', 'red'])

plt.title('Target Variable Distribution')

plt.xlabel('Target')

plt.ylabel('Frequency')

plt.xticks([0, 1], ['No Heart Disease', 'Heart Disease'])

plt.show()

**5. Splitting the Data**

Splitting the dataset is a fundamental step in building a machine learning model. It involves dividing the dataset into separate parts to train and evaluate the model effectively. Here are the key reasons and theoretical foundations for splitting the data:

#### 1. **Purpose of Data Splitting**

* **Training Set**: This subset is used to train the model. The model learns from this data by adjusting its parameters to minimize the error.
* **Testing Set**: This subset is used to evaluate the model's performance. It helps in assessing how well the model generalizes to unseen data.

#### 2. **Why Split the Data?**

* **Avoid Overfitting**: Overfitting occurs when a model learns the training data too well, capturing noise and outliers, which leads to poor performance on new, unseen data. Splitting the data helps in identifying and mitigating overfitting.
* **Evaluate Generalization**: The primary goal of a machine learning model is to perform well on new data. By using a separate testing set, we can evaluate the model's ability to generalize beyond the training data.
* **Parameter Tuning**: Data splitting allows for tuning hyperparameters using techniques like cross-validation on the training set, while still having a separate testing set to evaluate the final model performance.

#### 3. **Common Splitting Techniques**

* **Holdout Method**: This is the simplest method, where the dataset is divided into two parts: a training set and a testing set. A common split ratio is 80/20 or 70/30.
* **K-Fold Cross-Validation**: This method involves splitting the dataset into K subsets (folds). The model is trained on K-1 folds and tested on the remaining fold. This process is repeated K times, with each fold serving as the test set once. The results are averaged to provide a more robust evaluation.
* **Stratified Sampling**: This method is used when the target variable has imbalanced classes. Stratified sampling ensures that the proportion of each class in the training and testing sets is similar to that in the original dataset.

#### 4. **Implementation in Code**

In the context of our Heart Disease Prediction project, we use the holdout method with stratified sampling to ensure the class distribution is maintained in both the training and testing sets.

**Features and Target Separation**

**Code:**

X = heart\_data.drop(columns='target', axis=1)

Y = heart\_data['target']

**Train-Test Split**

**Code:**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, stratify=Y, random\_state=2)

print(X.shape, X\_train.shape, X\_test.shape)

#### 5. **Explanation of the Code**

* **train\_test\_split**: This function from the sklearn.model\_selection module splits the dataset into training and testing sets. The test\_size parameter specifies the proportion of the dataset to include in the test split (20% in this case).
* **stratify=Y**: This parameter ensures that the split is stratified based on the target variable Y, maintaining the same distribution of classes in both training and testing sets as in the original dataset.
* **random\_state=2**: This parameter ensures that the results are reproducible by setting a seed for the random number generator.

#### 6. **Benefits of Proper Data Splitting**

* **Reliable Performance Metrics**: Proper data splitting ensures that the performance metrics obtained during model evaluation are reliable and indicative of the model's real-world performance.
* **Better Generalization**: By evaluating the model on a separate testing set, we can be more confident in its ability to generalize to new data, making it more robust and useful in practical applications.
* **Prevention of Data Leakage**: Data leakage occurs when information from outside the training dataset is used to create the model. Proper splitting helps prevent such issues, ensuring that the model is evaluated on truly unseen data.

#### 7. **Common Pitfalls to Avoid**

* **Not Shuffling Data**: If the data is not shuffled before splitting, especially if it is ordered, the training and testing sets might not be representative, leading to biased evaluation results.
* **Ignoring Class Imbalance**: If the target variable has imbalanced classes, a simple random split might not preserve the class distribution, leading to a biased model. Stratified sampling addresses this issue.
* **Overlapping Data**: Ensuring that the training and testing sets are mutually exclusive is crucial to avoid data leakage and ensure proper model evaluation.

**6. Model Selection and Training**

**Overview of Algorithms**

Various algorithms can be used for heart disease prediction. In this project, we chose the Random Forest algorithm due to its high accuracy and robustness.

**Why Random Forest?**

* **Accuracy and Robustness**: Random Forests generally offer high accuracy and robustness to overfitting compared to individual decision trees.
* **Handling Missing Values**: Random Forest can handle missing values and maintain accuracy when a large proportion of the data is missing.
* **Feature Importance**: It provides an estimate of the importance of different features, which can be useful for understanding the data.
* **Versatility**: Random Forests can be used for both classification and regression tasks, making them versatile.
* **Non-linear Relationships**: It can capture non-linear relationships in the data, which is beneficial for complex datasets like medical records.

**Training the Random Forest Model**

**Code:**

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(X\_train, Y\_train)

**7.Model Evaluation**

Model evaluation is a crucial step in the machine learning pipeline. It involves assessing how well the trained model performs on unseen data. This step ensures that the model is not only accurate but also generalizes well to new data. Here are key concepts and metrics used in model evaluation:

#### **Purpose of Model Evaluation**

* **Assessing Performance**: To determine how well the model has learned from the training data and how accurately it can predict outcomes on new data.
* **Comparing Models**: To compare different models or algorithms to choose the best one for the specific problem.
* **Tuning Parameters**: To fine-tune the hyperparameters of the model to achieve optimal performance.
* **Ensuring Generalization**: To ensure that the model performs well not only on the training data but also on unseen data.

#### **Evaluation Metrics**

Different types of metrics are used based on the problem type (classification, regression, etc.).

#### **Cross-Validation**

Cross-validation is a technique for assessing how the results of a statistical analysis generalize to an independent dataset. It is primarily used in settings where the goal is prediction and one wants to estimate how accurately a predictive model will perform in practice.

#### **Model Evaluation in Practice**

In practice, multiple evaluation metrics should be used to get a comprehensive understanding of the model's performance. The choice of metrics depends on the specific problem, the cost of false positives and false negatives, and the business requirements.

#### **Implementation in Code**

For the Heart Disease Prediction project, we use accuracy as the primary evaluation metric for the Logistic Regression model.

**Accuracy on Training Data**

**Code:**

from sklearn.metrics import accuracy\_score

X\_train\_prediction = model.predict(X\_train)

training\_data\_accuracy = accuracy\_score(X\_train\_prediction, Y\_train)

print('Accuracy on Training data : ', training\_data\_accuracy)

**Accuracy on Test Data**

**Code:**

X\_test\_prediction = model.predict(X\_test)

test\_data\_accuracy = accuracy\_score(X\_test\_prediction, Y\_test)

print('Accuracy on Test data : ', test\_data\_accuracy)

**Confusion Matrix**

**Code:**

from sklearn.metrics import confusion\_matrix

confusion\_mat = confusion\_matrix(Y\_test, X\_test\_prediction)

print(confusion\_mat)

**Precision, Recall, F1 Score**

**Code:**

from sklearn.metrics import classification\_report

report = classification\_report(Y\_test, X\_test\_prediction)

print(report)

**8.Building a Predictive System**

Building a predictive system involves creating a framework that can make predictions on new, unseen data based on the model trained during the previous steps. This system takes raw input data, processes it, and utilizes the trained model to output predictions. Here, we describe the key steps involved in building a predictive system for heart disease prediction.

#### 1. **Input Data Preparation**

The first step in the predictive system is preparing the input data. The model expects data in a specific format that matches the format of the training data.

* **Define Input Data**: The input data should be a tuple or list containing values for all the features that the model was trained on. In this case, the features might include age, sex, chest pain type, resting blood pressure, cholesterol levels, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise-induced angina, ST depression, slope of the peak exercise ST segment, number of major vessels colored by fluoroscopy, and thalassemia.

**Code:**

input\_data = (54, 1, 0, 120, 188, 0, 1, 113, 0, 1.4, 1, 1, 3)

* **Convert to NumPy Array**: For compatibility with the model, the input data is converted to a NumPy array. This step ensures that the data is in a format that the model can process.

**Code:**

input\_data\_as\_numpy\_array = np.asarray(input\_data)

* **Reshape Data**: The model expects the input to be in the form of a 2D array (even if predicting for a single instance). Reshaping the array ensures that it matches the expected input shape.

**Code:**

input\_data\_reshaped = input\_data\_as\_numpy\_array.reshape(1, -1)

#### 2. **Making Predictions**

Once the input data is prepared, the next step is to use the trained model to make predictions.

* **Predict Using Model**: The predict method of the trained model is called with the reshaped input data. The model outputs a prediction based on the input features.

**Code:**

prediction = model.predict(input\_data\_reshaped)

print(prediction)

In the case of a classification problem like heart disease prediction, the output is typically a class label indicating whether the condition is present or not (e.g., 0 for no heart disease, 1 for heart disease).

#### 3. **Interpreting Results**

Interpreting the prediction results involves translating the model's output into a meaningful statement that can be understood by users.

* **Conditional Check**: The predicted class label is checked using a conditional statement to determine whether the model predicts the presence of heart disease.

**Code:**

if (prediction[0] == 0):

print('The Person does not have a Heart Disease')

else:

print('The Person has Heart Disease')

* + **Prediction == 0**: If the model predicts a class label of 0, the output indicates that the person does not have heart disease.
  + **Prediction == 1**: If the model predicts a class label of 1, the output indicates that the person has heart disease.

This predictive system allows for quick and accurate predictions based on new input data, making it a powerful tool for early detection and intervention in heart disease.

### Summary

Building a predictive system is the final step in the machine learning pipeline, transforming a trained model into a practical tool that can be used for real-world decision-making. The process includes preparing the input data, making predictions with the trained model, and interpreting the results to provide actionable insights. This system can be integrated into larger applications, such as web services or mobile apps, to provide users with immediate health assessments based on their input data.

### 9. Conclusion

#### Summary of Findings

In this project, we developed a machine learning model to predict the presence of heart disease using a dataset obtained from Kaggle. The primary steps included data collection and processing, feature-target separation, data splitting, model training, evaluation, and building a predictive system. The logistic regression model demonstrated an accuracy of approximately 85.12% on the training data and 81.97% on the test data, indicating its effectiveness in predicting heart disease based on the given features.

#### Importance of the Model

Heart disease remains one of the leading causes of death globally. Early detection and diagnosis are crucial in preventing serious outcomes and managing the disease effectively. The machine learning model developed in this project can serve as a valuable tool for healthcare professionals, providing quick and accurate predictions that can assist in early diagnosis and intervention. The model's ability to analyze multiple factors simultaneously and provide a prediction can aid in better decision-making and personalized treatment plans.

#### Future Work

While the logistic regression model provided satisfactory results, there are several areas for potential improvement and future research:

* **Exploring Other Algorithms**: Investigating other machine learning algorithms, such as Random Forest, Support Vector Machines, or Neural Networks, could enhance predictive accuracy and robustness.
* **Feature Engineering**: Further refining the features, incorporating domain knowledge, and creating new features could improve the model's performance.
* **Data Augmentation**: Using larger and more diverse datasets could help in building a more generalized model.
* **Deployment**: Integrating the model into a user-friendly application or platform could make it accessible to a broader audience, including healthcare providers and patients.
* **Validation**: Conducting cross-validation and extensive testing with external datasets to ensure the model's reliability and validity in different scenarios.

### 10. References

**1.Dataset**

* + Kaggle: Heart Disease UCI Dataset

<https://drive.google.com/file/d/1nDFWGGVPx-8zcqbF_oheWXn0Le5zG18l/view?usp=sharing>

**2.Libraries and Tools**

* + NumPy: [<https://numpy.org/>]
  + Pandas: [https://pandas.pydata.org/]
  + Scikit-Learn: [<https://scikit-learn.org/>]

**3.Research Papers and Articles**

* + "Heart Disease Prediction using Machine Learning" by various authors, accessible through academic databases.
  + Various online resources and documentation on logistic regression and model evaluation techniques.

**4.Software and Platform**

* + Jupyter Notebook: [<https://jupyter.org/>]
  + Google Colab: [https://colab.research.google.com/]