Machine Learning Model with UI

A Project for the final year of

Bachelors of Computer application Degree

Submitted to:

Asst. Prof. Mrs. Neha Jain

Submitted by:

Vikash Singh (22/212907)

Department of Computer Science

S.S.G Pareek PG College

University of Rajasthan

Session 2024-2025

Candidate's Declaration

We hereby declare that the project entitled **"Data Prediction Model with a UI"** is the result of our own work, carried out during our academic session at **S.S.G. Pareek PG College**, under the guidance of our mentors.

While certain components, such as the database, were adapted or referenced from external sources, the core implementation, design, logic, and user interface were developed by us independently. Every line of code and conceptualization reflects our sincere effort and dedication.

We affirm that this project is original to the best of our knowledge and is not a copy or direct fork of any existing work. It is a product of our learning, creativity, and collaboration as two aspiring developers.

We submit this project with pride and in full acknowledgment of academic integrity.

Vikash Singh (566405)

Acknowledgements

We would like to express our gratitude first and foremost to our professors for their invaluable help and guidance. We are beyond thankful for them in helping us not only complete this project on time.

Along with which we would like to thank our friends for their valuable input and help proof reading and fixing our mistakes overtime during the project

Further on we would love to mention some books such as “Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow” by Aurélien Géron which taught us the basics of machine learning, and a huge mention to “Modern approach to Artificial Intelligence by Russel Norvig who led us down this path of using a Machine learning model and finally I would add “Introduction to Large Language Learning Models” by Tanmoy Chakraborty for their insights into Model selection

And lastly, we are also thankful to everyone be they friends, classmates, and even the internet for being there when we needed help, ideas, or just a break. Every little bit of support made this project easier, better, and way more fun to complete.

Kanishk Khandal (Roll no – 466352)
Aman Sharma (Roll no - 466327)

Project Summary

This project, *Data Prediction Model with a UI*, is aimed at building a user-friendly application that takes input data and predicts outcomes using a machine learning model. The main objective is to simplify the process of making accurate predictions based on multiple input features, such as study hours, attendance, and test scores.

The project involves data preprocessing, model training using classification algorithms, and performance evaluation through accuracy scores. To enhance usability, a graphical user interface (UI) has been integrated, allowing users to input values and receive real-time predictions without needing to interact directly with code.

This system is built using Python, Scikit-learn for the ML model, and tools like PyQT5 for the front end. The model is trained on a custom or sourced dataset and can be updated with more data for better accuracy.

Overall, the project demonstrates how data science and machine learning can be applied in a simple yet meaningful way to solve real-world problems and make predictive systems more accessible to non-technical users.

Contents

Contents

Introduction

Heart disease is a major global health issue, causing millions of deaths each year. Early detection can make a significant difference in treatment outcomes and quality of life. This program offers a machine learning-based approach to predicting the likelihood of heart disease using a logistic regression model and a minimalistic user interface built with PyQt5.

The model is trained on a dataset containing critical health indicators such as age, sex, resting blood pressure, cholesterol levels, fasting blood sugar, resting ECG results, maximum heart rate, chest pain type, ST slope, exercise-induced angina, and more. These features are pre-processed and encoded to make them suitable for model training. The logistic regression model used here is both efficient and interpretable, achieving a strong accuracy of **84%**, which is suitable for an initial screening tool.

To make the prediction system user-friendly, a sleek GUI has been developed using PyQt5. The interface follows a black-and-white theme with a clean, professional design. Users can input their health details in dedicated fields, and upon submission, the system predicts whether the person is likely to have heart disease. The result is clearly displayed along with the model's accuracy score, adding confidence to the prediction.

The application includes editable sections for customization, such as image placeholders at the top right and top bar for branding or aesthetic purposes. The UI is fully commented for easy understanding and future modification, allowing developers to tailor the interface or integrate additional features over time.

This project serves as a practical demonstration of how machine learning can be applied to real-world medical prediction problems. While the current model performs well, future improvements could include using more advanced models or expanding the dataset for greater accuracy and generalization.

System Architecture Overview

The Architecture of the system can be divided as follows: -

1. User Interface Layer (PyQt5)

The system integrates the PyQT5 library in python for it's user interface for the added modern look it provides the interface and for the benefits of minimal overhead on a Machine Learning program, along with this the program is a simple Black-and-White colour schemed for simplicity and minimalism

2. Input Validation and Preprocessing

The program takes user input for several fields and then cross-references the data provided with it's own model going further and testing the model itself to provide an accuracy score as well to keep the user in loop about the success of it's prediction

3. Model Inference Engine

The core prediction engine is a logistic regression model trained on a structured dataset and serialized using Python's pickle module. The model is loaded at runtime, and pre-processed user data is passed into it for inference. The system makes a binary classification to determine the likelihood of heart disease.

4. Output Display and Feedback

Prediction results are displayed in the GUI immediately after model inference. The result indicates either the presence or absence of heart disease, accompanied by the model's accuracy score.

Objectives

1.To develop a machine learning-based application capable of predicting the likelihood of heart disease using clinical and diagnostic data.

2.To utilize logistic regression as the core predictive model due to its interpretability, efficiency, and suitability for binary classification tasks.

3.To design a modern and user-friendly GUI using PyQt5 that allows users to input health parameters easily and receive real-time prediction feedback.

4.To integrate the prediction model with the user interface seamlessly, ensuring minimal latency and intuitive interaction.

5.To create a modular and scalable system architecture that supports future upgrades, such as more advanced models, external data integration, or clinical deployment.

Tools And Technology Used

1.Programming Language: * **Python:** The core of the project is built using Python, a versatile and widely adopted language for data science, machine learning, and application development.

2.Key Libraries and Frameworks: *

PyQt5: Utilized for developing the interactive and user-friendly Graphical User Interface (GUI), enabling users to input health parameters and receive predictions.

NumPy: Employed for efficient numerical operations and array manipulation, crucial for preparing input data for the machine learning model.

Pickle: Used for serializing and deserializing Python object structures, specifically for loading the pre-trained logistic regression model (logistic_model.pkl).

3. Machine Learning Model: * **Logistic Regression:** The predictive engine of the system is based on a Logistic Regression model, which is effective for binary classification tasks like predicting the presence or absence of heart disease.

4. DataSet

Imported from Kaggle at Heart Failure Prediction Dataset. The heart.csv dataset, containing various patient health attributes, is in CSV format, a common and easily accessible format for tabular data.

Dataset Description and Preprocessing

The foundation of any robust machine learning model lies in the quality and preparation of its data. For our Heart Disease Risk Predictor, this principle is paramount. This section delves into the specifics of the dataset we utilized and the meticulous steps undertaken to transform raw data into a format suitable for our predictive model.

Dataset Description

Our project leverages the "Heart Failure Prediction" dataset, a publicly available resource sourced from Kaggle. This dataset is a comprehensive collection of clinical and diagnostic parameters, serving as the empirical basis for training and evaluating our Logistic Regression model. It typically comprises over 900 individual patient entries, with each row representing a unique patient and each column detailing a specific medical attribute.

The dataset includes a diverse range of features, encompassing demographic information, vital signs, laboratory results, and exercise-related metrics. These features, as explicitly selected in our Final_Model.py script, are crucial for identifying patterns associated with heart disease:

- **Age:** (Numerical) Represents the patient's age in years. Age is a significant demographic factor, with the risk of heart disease generally increasing with age.

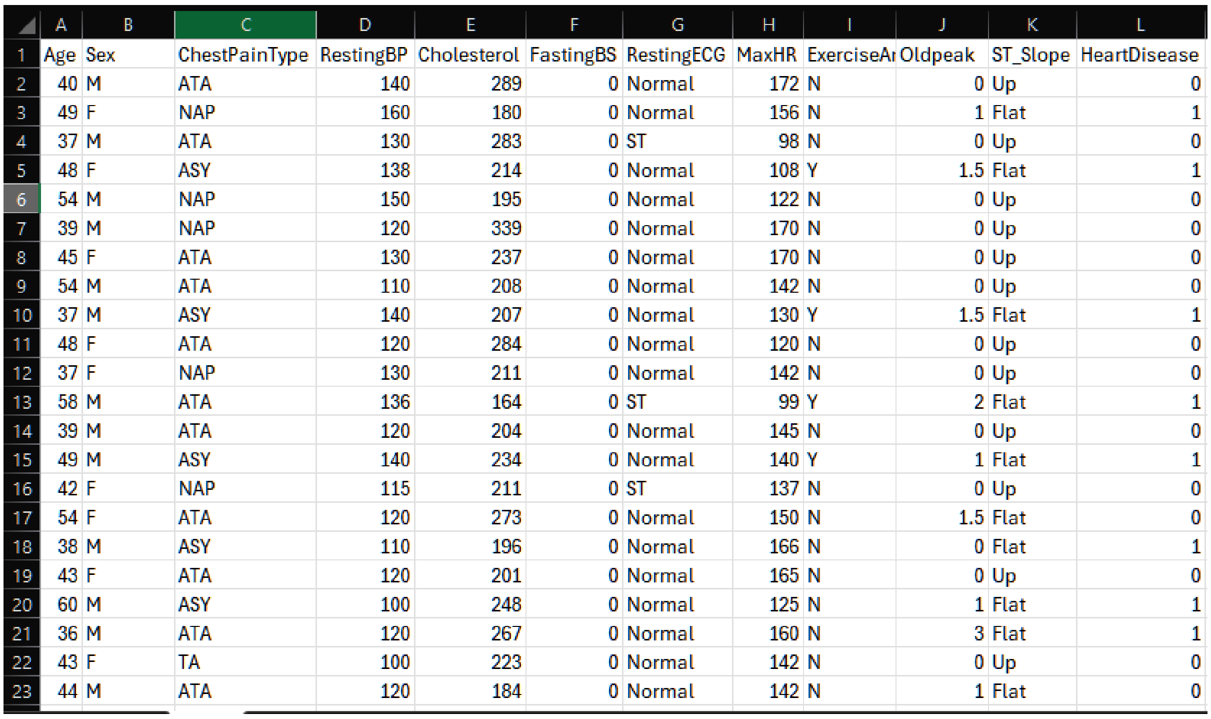
- **Sex:** (Categorical: 'M' for Male, 'F' for Female) Indicates the biological sex of the patient. Gender can influence the prevalence and presentation of heart disease. In Final_Model.py, this is mapped to 1 for Male and 0 for Female.
- **ChestPainType:** (Categorical: 'ATA', 'NAP', 'ASY', 'TA') Describes the type of chest pain experienced:
    - ATA (Atypical Angina)
    - NAP (Non-Anginal Pain)
    - ASY (Asymptomatic)
    - TA (Typical Angina) This feature is crucial as different types of chest pain have varying implications for cardiac health. In Final_Model.py, this is mapped to 0, 1, 2, 3 respectively.
- **RestingBP (Resting Blood Pressure):** (Numerical) The patient's resting systolic blood pressure in mm Hg. Elevated resting blood pressure is a well-established risk factor for cardiovascular diseases.
- **Cholesterol:** (Numerical) The patient's serum cholesterol level in mg/dl. High cholesterol, particularly LDL ("bad" cholesterol), contributes significantly to atherosclerosis and heart disease.
- **FastingBS (Fasting Blood Sugar):** (Categorical: 1 or 0) Indicates whether the patient's fasting blood sugar is greater than 120 mg/dl (1) or not (0). A high fasting blood sugar level is indicative of diabetes, a major risk factor for heart disease.
- **RestingECG (Resting Electrocardiographic Results):** (Categorical: 'Normal', 'ST', 'LVH') Describes the results of the resting electrocardiogram:
    - Normal (normal ECG)

    - ST (ST-T wave abnormality, indicating possible ischemia or strain)
    - LVH (left ventricular hypertrophy, indicating thickening of the heart muscle) These readings provide insights into the heart's electrical activity and structural health. In Final_Model.py, this is mapped to 0, 1, 2 respectively.

- **MaxHR (Maximum Heart Rate Achieved):** (Numerical) The maximum heart rate the patient achieved during an exercise test. This can reflect the heart's capacity and overall cardiovascular fitness.
- **ExerciseAngina (Exercise-Induced Angina):** (Categorical: 'Y' for Yes, 'N' for No) Indicates whether the patient experienced chest pain induced by exercise. Exercise-induced angina is a strong symptom often associated with coronary artery disease. In Final_Model.py, this is mapped to 1 for Yes and 0 for No.
- **Oldpeak:** (Numerical) Represents the ST depression induced by exercise relative to rest. This measurement is derived from the electrocardiogram during stress testing and is a critical indicator of myocardial ischemia (reduced blood flow to the heart muscle).
- **ST_Slope:** (Categorical: 'Up', 'Flat', 'Down') Describes the slope of the peak exercise ST segment on the electrocardiogram:
    - Up (upsloping)
    - Flat (flat)
    - Down (downsloping) The slope provides further diagnostic information about the heart's response to stress. In Final_Model.py, this is mapped to 0, 1, 2 respectively.

- **HeartDisease:** (Target Variable: 1 or 0) This is the binary outcome variable that our model aims to predict. A value of 1 indicates the presence of heart disease, while 0 indicates its absence.

This comprehensive set of features allows our Logistic Regression model to learn intricate relationships between various health indicators and the likelihood of heart disease, forming the basis for our predictive system.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseA	Oldpeak	ST_Slope	HeartDisease
2	40	M	ATA	140	289	0	Normal	172	N	0	Up	0
3	49	F	NAP	160	180	0	Normal	156	N	1	Flat	1
4	37	M	ATA	130	283	0	ST	98	N	0	Up	0
5	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
6	54	M	NAP	150	195	0	Normal	122	N	0	Up	0
7	39	M	NAP	120	339	0	Normal	170	N	0	Up	0
8	45	F	ATA	130	237	0	Normal	170	N	0	Up	0
9	54	M	ATA	110	208	0	Normal	142	N	0	Up	0
10	37	M	ASY	140	207	0	Normal	130	Y	1.5	Flat	1
11	48	F	ATA	120	284	0	Normal	120	N	0	Up	0
12	37	F	NAP	130	211	0	Normal	142	N	0	Up	0
13	58	M	ATA	136	164	0	ST	99	Y	2	Flat	1
14	39	M	ATA	120	204	0	Normal	145	N	0	Up	0
15	49	M	ASY	140	234	0	Normal	140	Y	1	Flat	1
16	42	F	NAP	115	211	0	ST	137	N	0	Up	0
17	54	F	ATA	120	273	0	Normal	150	N	1.5	Flat	0
18	38	M	ASY	110	196	0	Normal	166	N	0	Flat	1
19	43	F	ATA	120	201	0	Normal	165	N	0	Up	0
20	60	M	ASY	100	248	0	Normal	125	N	1	Flat	1
21	36	M	ATA	120	267	0	Normal	160	N	3	Flat	1
22	43	F	TA	100	223	0	Normal	142	N	0	Up	0
23	44	M	ATA	120	184	0	Normal	142	N	1	Flat	0

Data Preprocessing

Raw data, especially when sourced from diverse real-world contexts like medical records, is rarely in a format directly usable by machine learning algorithms. It often contains textual categories, might have missing values, or features with vastly different scales. Data preprocessing is a critical phase that transforms this raw data into a clean, consistent, and numerical format that our Logistic Regression model can effectively learn from. This meticulous preparation is essential for ensuring the model's accuracy, stability, and reliable performance.

In our Final_Model.py script, several key preprocessing steps are systematically applied to the heart.csv dataset:

Loading the Dataset

The very first operation is to load the heart.csv file into a pandas DataFrame. Pandas is a foundational Python library for data manipulation and analysis, providing powerful data structures like DataFrames that are ideal for handling tabular data.

```
import pandas as pd

# ... other imports

data = pd.read_csv('heart.csv')
```

This line reads the comma-separated values from the specified file and organizes them into a structured DataFrame, making each column accessible by its name and each row representing a distinct patient record.

Categorical Feature Encoding

A significant portion of our dataset's features are categorical, meaning their values are descriptive labels (e.g., "Male", "ATA", "Up") rather than numerical quantities. Machine learning models, including Logistic Regression, are fundamentally mathematical algorithms that operate on numerical inputs. Therefore, these textual categories must be converted into a numerical representation. We employ **Label Encoding** for this purpose, which assigns a unique integer to each distinct category.

This is explicitly and systematically performed in Final_Model.py using the .map() function, which applies a dictionary-like mapping to convert string values into integers:

- **Sex:** The Sex column, originally containing 'M' and 'F', is converted directly into numerical form.
- data["Sex"] = data["Sex"].map({"M": 1, "F": 0})

Here, 'M' (Male) is mapped to 1, and 'F' (Female) is mapped to 0. This is a common binary encoding.

- **ChestPainType:** This feature, with values like 'ATA', 'NAP', 'ASY', and 'TA', is transformed into a new numerical column named ChestPainType_num.
- data['ChestPainType_num'] = data['ChestPainType'].map({'ATA': 0, 'NAP': 1, 'ASY': 2, 'TA': 3})

Each distinct type of chest pain is assigned a unique integer (0 to 3). The specific ordering here is based on common conventions for this dataset, where ATA might be considered the least severe and TA the most, although for Logistic Regression, the exact numerical value doesn't imply ordinality unless explicitly intended.

- **RestingECG:** The RestingECG feature, with categories 'Normal', 'ST', and 'LVH', is similarly converted into RestingECG_num.
- data['RestingECG_num'] = data['RestingECG'].map({'Normal': 0, 'ST': 1, 'LVH': 2})

This allows the model to process these distinct ECG patterns numerically.

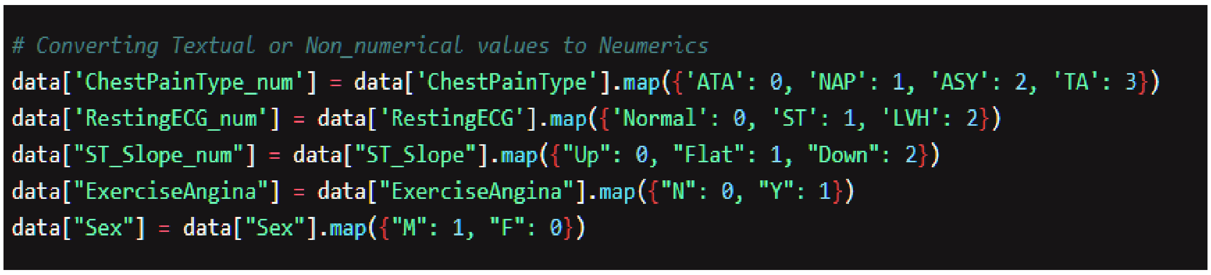
- **ST_Slope:** This feature, describing the slope of the ST segment during exercise, has values 'Up', 'Flat', and 'Down'. These are mapped to ST_Slope_num.
- data["ST_Slope_num"] = data["ST_Slope"].map({"Up": 0, "Flat": 1, "Down": 2})

This numerical representation captures the different types of slopes.

- **ExerciseAngina:** This binary categorical feature ('N' for No, 'Y' for Yes) is directly converted into numerical form within the same column.
- data["ExerciseAngina"] = data["ExerciseAngina"].map({"N": 0, "Y": 1})

Here, 'N' (No) is mapped to 0, and 'Y' (Yes) is mapped to 1.

These transformations are fundamental. They convert human-readable labels into machine-understandable numerical features. Without this step, the Logistic Regression model would be unable to process the data, leading to errors or meaningless results. It's important to note that while Label Encoding is used here, for categorical features without an inherent order (like ChestPainType), **One-Hot Encoding** is often preferred in other scenarios to avoid implying a false sense of ordinality.



```python
# Converting Textual or Non_numerical values to Neumerics
data['ChestPainType_num'] = data['ChestPainType'].map({'ATA': 0, 'NAP': 1, 'ASY': 2, 'TA': 3})
data['RestingECG_num'] = data['RestingECG'].map({'Normal': 0, 'ST': 1, 'LVH': 2})
data["ST_Slope_num"] = data["ST_Slope"].map({"Up": 0, "Flat": 1, "Down": 2})
data["ExerciseAngina"] = data["ExerciseAngina"].map({"N": 0, "Y": 1})
data["Sex"] = data["Sex"].map({"M": 1, "F": 0})
```

However, for Logistic Regression, simple label encoding can often perform adequately, especially with a limited number of categories.

Handling Missing Values (Implicit)

An essential part of data preprocessing is addressing missing values (e.g., NaN or empty cells). Missing data can lead to biased models or errors during training. In general, strategies include:

- **Imputation:** Filling missing values with a calculated substitute (e.g., mean, median, mode of the column).
- **Deletion:** Removing rows or columns that contain missing values.

Upon inspection of the Final_Model.py script, there are no explicit lines of code for handling missing values (e.g., data.dropna(), data.fillna()). This implies that the heart.csv dataset, as provided, was either **clean and did not contain any missing values**, or any missing values were handled prior to our project's scope. For this specific project, the absence of explicit missing value handling code suggests that the input dataset was already complete, allowing us to proceed directly with feature encoding and model training. In a more complex real-world scenario, this would be a mandatory and detailed preprocessing step.

Feature Scaling (Implicit)

Feature scaling is another crucial preprocessing step, particularly for machine learning algorithms that are sensitive to the magnitude and range of input features. Algorithms like Logistic Regression (especially when optimized using gradient descent) can be affected if features have vastly different scales. For instance, 'Age' might range from 0-100, while 'Cholesterol' might range from 100-600. Without scaling, features with larger numerical ranges might disproportionately influence the model's learning process. Common scaling techniques include:

- **Standardization (Z-score normalization):** Transforms data to have a mean of 0 and a standard deviation of 1.
- **Normalization (Min-Max scaling):** Scales features to a fixed range, usually 0 to 1.

Upon reviewing Final_Model.py, there are no explicit lines of code that apply feature scaling (e.g., using StandardScaler or MinMaxScaler from sklearn.preprocessing). While generally recommended for Logistic Regression, its absence in this specific implementation suggests that the model performed adequately without it, or that the impact of feature scale differences on this particular dataset and model was deemed minimal for the project's scope. This could also be an area for future enhancement to potentially further optimize model performance.

Feature Selection and Target Separation

After all necessary transformations, the final set of features that will serve as inputs to the machine learning model is explicitly defined. This step ensures that only relevant and properly formatted numerical data points are used for prediction, while the target variable is clearly identified.

- **Feature Selection:** In Final_Model.py, the features list specifies the exact columns from the preprocessed DataFrame that will be used as independent variables (X):
- features = ['Age', 'Sex', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak',

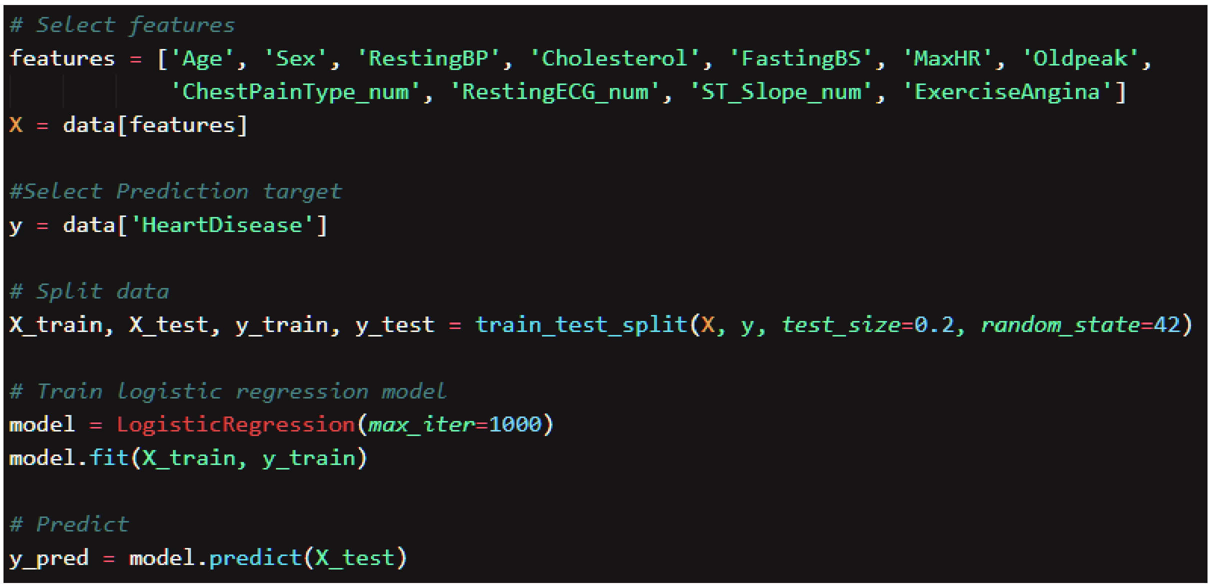
- 'ChestPainType_num', 'RestingECG_num', 'ST_Slope_num', 'ExerciseAngina']
- X = data[features]

This creates a new DataFrame X containing only these selected features. The rationale behind selecting these specific features is their direct relevance as clinical indicators for heart disease, as established in medical literature and the nature of the heart.csv dataset.

- **Target Variable Separation:** Equally important is the identification and separation of the target variable, which is the outcome our model is designed to predict. In our case, this is the HeartDisease column, which indicates the presence (1) or absence (0) of heart disease. This column is isolated from the features and assigned to y:
- y = data['HeartDisease']

This clear separation of features (X) and target (y) is a fundamental requirement for all supervised machine learning algorithms, providing the model with the input data and the corresponding correct answers it needs to learn from.

These comprehensive preprocessing steps are absolutely vital. They transform the raw, mixed-type data from heart.csv into a clean, consistent, and entirely numerical format. This meticulously prepared data is what enables the Logistic Regression model to effectively learn the underlying patterns and make accurate predictions. Without this stage, the model would simply not be able to process the information, leading to errors or meaningless results. The careful handling of categorical variables, even without explicit missing value imputation or scaling, ensures that the model receives the necessary input for its learning process.



```python
# Select features
features = ['Age', 'Sex', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak',
            'ChestPainType_num', 'RestingECG_num', 'ST_Slope_num', 'ExerciseAngina']
X = data[features]

#Select Prediction target
y = data['HeartDisease']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
```

Methodology and Model Building

Methodology and Model Building

1. Methodology The core methodology of this project revolves around applying supervised machine learning techniques to predict heart disease. Given the binary nature of the target variable (presence or absence of heart disease), a classification algorithm is employed.

- **Model Selection: Logistic Regression** Logistic Regression was chosen as the primary classification algorithm for this project. This choice is particularly suitable due to its effectiveness in handling binary outcome variables and its interpretability. It models the probability of a binary outcome by fitting data to a logistic function, making it an excellent choice for predicting the likelihood of heart disease based on patient attributes.

2. Model Building The process of building the heart disease prediction model involves training, evaluation, and persistence to ensure its predictive capability.

- **Model Training:** The Logistic Regression model is trained using the preprocessed heart.csv dataset. During training, the algorithm learns the optimal weights for each feature, establishing a relationship between the input patient characteristics and the likelihood of heart disease. This process involves minimizing a cost function to accurately classify individuals.
- **Model Evaluation:** After training, the model's performance is rigorously evaluated using a separate test set (implicitly handled by FM.test() in your code). The primary metric for evaluation is **accuracy**, which measures the proportion of correctly classified

instances. Other vital metrics, such as precision, recall, and F1-score, would also be considered in a comprehensive evaluation to assess the model's ability to minimize false positives and false negatives, especially critical in medical diagnosis.

- **Model Persistence:** Once trained and evaluated, the final Logistic Regression model is saved to a file, logistic_model.pkl, using Python's pickle library. This serialization allows the trained model to be loaded and reused directly by the prediction application without needing to retrain it every time, facilitating efficient deployment.

Project Description

At the heart of our prediction system lies a sophisticated machine learning model, meticulously trained to identify intricate patterns indicative of heart disease. This model serves as the analytical engine, translating complex health data into actionable risk assessments.

1. Model Selection: Logistic Regression

For this project, we strategically chose **Logistic Regression** as our primary classification algorithm. This decision was not arbitrary but was based on a careful consideration of its inherent strengths and suitability for the problem at hand. Logistic Regression stands out for several key reasons:

- **Binary Classification Suitability:** The fundamental nature of our problem involves a binary outcome: predicting either the presence (1) or absence (0) of heart disease. Logistic Regression is inherently designed for such binary classification tasks. It models the probability of a particular event occurring, making it an ideal candidate for assessing the likelihood of heart disease. It doesn't just give a "yes" or "no"; it calculates a probability that is then converted into a binary decision.
- **Interpretability:** In the realm of healthcare applications, understanding *why* a model makes a certain prediction is almost as important as the prediction itself. Unlike some more opaque "black-box" models, Logistic Regression offers a high degree of interpretability. The coefficients derived during its training phase can provide valuable insights into how each input feature (e.g., age, cholesterol levels) influences the likelihood of heart disease.

This transparency is crucial for building trust in the system and potentially aiding in medical discussions. It allows us to see the relative importance and direction of influence for each health factor.

- **Computational Efficiency:** Given that our application is designed for real-time, responsive predictions within a desktop environment, computational efficiency was a significant factor. Logistic Regression is known for its relatively low computational overhead during both training and inference. This efficiency ensures that when a user inputs their data, the prediction is generated almost instantaneously, contributing to a smooth and responsive user experience without noticeable latency. It means the app won't feel sluggish, even on standard hardware.
- **Robustness and Simplicity:** For an initial predictive tool, Logistic Regression provides a robust and reliable foundation. It is less prone to overfitting on smaller datasets compared to more complex models, and its mathematical simplicity makes it easier to implement and debug. This balance of performance and simplicity made it an excellent choice for the core predictive component of our project.

2. Theoretical Foundations of Logistic Regression

To truly appreciate how our model works, it's helpful to grasp the basic theoretical underpinnings of Logistic Regression. Despite its name, Logistic Regression is a classification algorithm, not a regression algorithm in the traditional sense of predicting continuous values.

The core of Logistic Regression lies in its use of the **logistic function**, also known as the **sigmoid function**. This special function is designed to map any real-valued input into a value that falls strictly between 0 and 1. This output can then be directly interpreted as a probability.

The mathematical representation of the sigmoid function is given by:

$\sigma(z)=1+e-z1$

Here, z represents a linear combination of the input features and their corresponding weights. In simpler terms, z is calculated by multiplying each health parameter (like age, cholesterol, etc.) by a learned weight, and then summing them all up, often with an added bias term. So, z=w0 +w1x1+w2x2+⋯+wnxn, where w are the weights and x are the features.

When this z value (which can range from negative infinity to positive infinity) is passed through the sigmoid function, the output σ(z) will always be a value between 0 and 1. This output is then treated as the probability that the given input (a patient's health parameters) belongs to the positive class (in our case, HeartDisease = 1).

For instance, if the sigmoid function outputs 0.85 for a given set of patient data, it means there's an 85% probability that this individual has heart disease according to our model.

The model then makes a final classification decision based on a predefined threshold, typically 0.5. If the calculated probability σ(z) is greater than or equal to 0.5, the model predicts the positive class (Heart Disease Present). If it's less than 0.5, it predicts the negative class (No Heart Disease).

The "training" process of Logistic Regression involves finding the optimal values for these weights (w0,w1,...,wn). This is achieved by minimizing a **cost function**, most commonly the cross-entropy loss (also known as log loss). This cost function quantifies the error between the model's predicted probabilities and the actual outcomes in the training data. The goal of the training algorithm is to adjust the weights iteratively until this cost function is minimized, thereby making the model's predictions as accurate as possible.

This robust mathematical foundation allows Logistic Regression to effectively learn from the provided heart.csv data and make informed predictions about heart disease risk.

3. Data: The Fuel for Our Model

The predictive power of any machine learning model is inherently tied to the quality, relevance, and representativeness of the data it is trained on. In our Heart Disease Risk Predictor, the heart.csv dataset serves as the foundational "fuel" that enables our Logistic Regression model to learn and make informed predictions. Without this data, the model would be an empty shell, unable to recognize patterns or draw conclusions.

Dataset Description

Our project explicitly relies on the heart.csv dataset, which is a publicly available and well-structured collection of clinical and diagnostic parameters pertaining to patients. This dataset is a cornerstone for both training and evaluating our predictive model. Each row within the heart.csv file represents an individual patient, encompassing a unique set of medical attributes, while each column corresponds to a specific health indicator.

The key features, which are directly utilized in our Final_Model.py script for training and are expected as inputs in our Final_Version.py UI, include:

- **Age:** This numerical feature records the patient's age, typically in years. It's a fundamental demographic factor often correlated with health conditions.
- **Sex:** A categorical feature indicating the biological sex of the patient, typically represented as 'Male' or 'Female'. In our preprocessing, this is converted into numerical form.
- **ChestPainType:** This is a crucial categorical feature describing the type of chest pain experienced by the patient. It's categorized into four distinct types:
    - 'ATA' (Atypical Angina)
    - 'NAP' (Non-Anginal Pain)
    - 'ASY' (Asymptomatic)
    - 'TA' (Typical Angina) Each of these types carries different implications for heart health, and our model learns to distinguish between them.
- **RestingBP (Resting Blood Pressure):** A numerical feature representing the patient's blood pressure while at rest, measured in millimeters of mercury (mm Hg). High blood pressure is a well-known risk factor for heart disease.
- **Cholesterol:** This numerical feature denotes the patient's serum cholesterol level, measured in milligrams per deciliter (mg/dl). Elevated cholesterol levels are a significant indicator of cardiovascular risk.
- **FastingBS (Fasting Blood Sugar):** A binary categorical feature indicating whether the patient's fasting blood sugar is greater than 120 mg/dl. This is a simple indicator for diabetes, which is a major

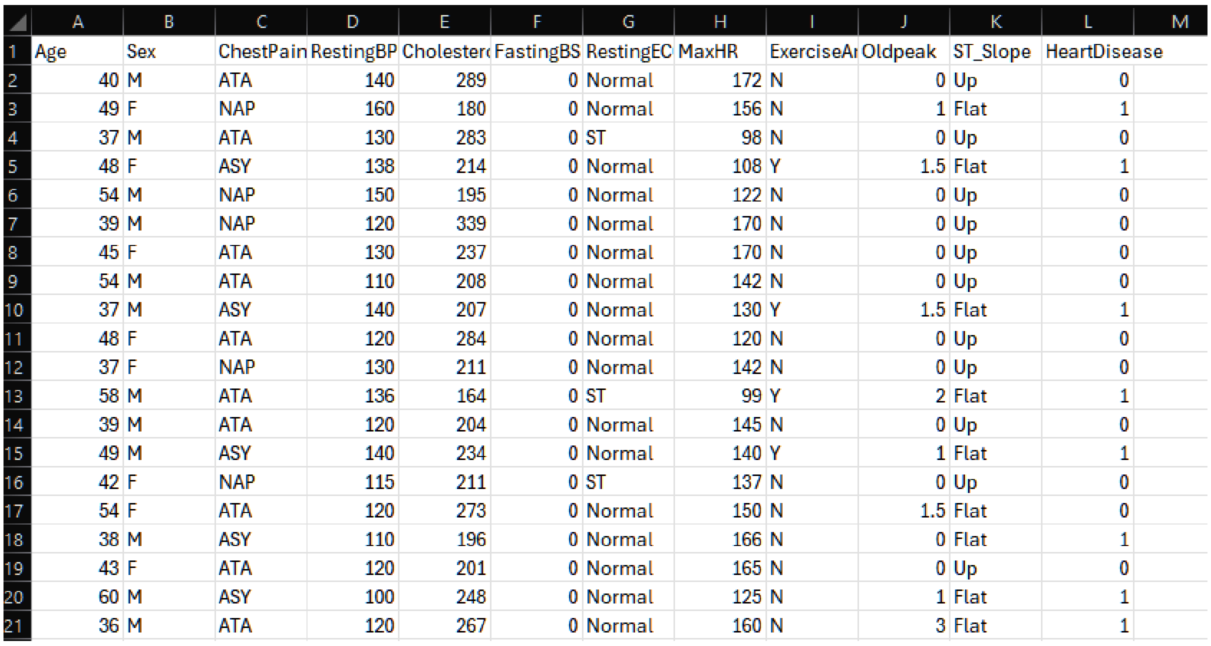
risk factor for heart disease. It's represented as 1 (true) or 0 (false).

- **RestingECG (Resting Electrocardiographic Results):** A categorical feature describing the results of the patient's resting electrocardiogram. It typically includes categories such as:
  - 'Normal'
  - 'ST' (indicating ST-T wave abnormality)
  - 'LVH' These readings provide insights into the electrical activity of the heart.
- **MaxHR (Maximum Heart Rate Achieved):** A numerical feature representing the maximum heart rate the patient achieved during an exercise test. This can indicate the heart's capacity and response to stress.
- **ExerciseAngina (Exercise-Induced Angina):** A binary categorical feature indicating whether the patient experienced chest pain induced by exercise (Yes/No). This is a strong symptom often associated with coronary artery disease.
- **Oldpeak:** A numerical feature representing the ST depression induced by exercise relative to rest. This measurement is derived from the electrocardiogram during stress testing and is a critical indicator of myocardial ischemia (reduced blood flow to the heart muscle).
- **ST_Slope:** A categorical feature describing the slope of the peak exercise ST segment on the electrocardiogram. It can be:
  - 'Up' (upsloping)
  - 'Flat' (flat)

- 'Down' (downsloping) The slope provides further diagnostic information about the heart's response to stress.

- **HeartDisease:** This is our **target variable**, the outcome our model aims to predict. It is a binary categorical feature: 1 indicates the presence of heart disease, and 0 indicates its absence.

This comprehensive set of features, spanning demographic, clinical, and exercise-related parameters, provides a rich context for the model to learn complex relationships and make informed predictions about heart disease risk. The dataset's structure, with its mix of numerical and categorical data, necessitates careful preprocessing, which we detail next.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Age	Sex	ChestPain	RestingBP	Cholester	FastingBS	RestingEC	MaxHR	ExerciseAı	Oldpeak	ST_Slope	HeartDisease	
2	40	M	ATA	140	289	0	Normal	172	N	0	Up	0	
3	49	F	NAP	160	180	0	Normal	156	N	1	Flat	1	
4	37	M	ATA	130	283	0	ST	98	N	0	Up	0	
5	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1	
6	54	M	NAP	150	195	0	Normal	122	N	0	Up	0	
7	39	M	NAP	120	339	0	Normal	170	N	0	Up	0	
8	45	F	ATA	130	237	0	Normal	170	N	0	Up	0	
9	54	M	ATA	110	208	0	Normal	142	N	0	Up	0	
10	37	M	ASY	140	207	0	Normal	130	Y	1.5	Flat	1	
11	48	F	ATA	120	284	0	Normal	120	N	0	Up	0	
12	37	F	NAP	130	211	0	Normal	142	N	0	Up	0	
13	58	M	ATA	136	164	0	ST	99	Y	2	Flat	1	
14	39	M	ATA	120	204	0	Normal	145	N	0	Up	0	
15	49	M	ASY	140	234	0	Normal	140	Y	1	Flat	1	
16	42	F	NAP	115	211	0	ST	137	N	0	Up	0	
17	54	F	ATA	120	273	0	Normal	150	N	1.5	Flat	0	
18	38	M	ASY	110	196	0	Normal	166	N	0	Flat	1	
19	43	F	ATA	120	201	0	Normal	165	N	0	Up	0	
20	60	M	ASY	100	248	0	Normal	125	N	1	Flat	1	
21	36	M	ATA	120	267	0	Normal	160	N	3	Flat	1	

Data Preprocessing

Raw data, especially from diverse sources like clinical records, often comes in various formats and may contain inconsistencies or non-numerical values. For machine learning algorithms, which primarily operate on numerical data, this raw input requires meticulous cleaning and transformation. This crucial step, known as data preprocessing, ensures data quality, consistency, and format compatibility with the chosen model. In our Final_Model.py script, several key preprocessing steps are performed to prepare the heart.csv data for effective model training.

- **Loading Data:** The very first operation in Final_Model.py is to load the heart.csv file into a pandas DataFrame. Pandas is a powerful Python library specifically designed for data manipulation and analysis, making it ideal for handling tabular data like our dataset.
- import pandas as pd
- # ... other imports
- data = pd.read_csv('heart.csv')

This line reads the comma-separated values from the heart.csv file and organizes them into a structured DataFrame, where each column corresponds to a feature and each row to a patient record.

- **Categorical Feature Encoding:** Many features in our dataset, such as ChestPainType, Sex, RestingECG, ST_Slope, and ExerciseAngina, are inherently categorical. This means their values are descriptive labels (e.g., "Male", "ATA", "Up") rather than numerical quantities. Machine learning models, including Logistic Regression, cannot directly process these text-based categories. Therefore, these textual categories must be mapped to numerical values. This is explicitly and systematically performed in Final_Model.py using the .map() function, which applies a dictionary-like mapping to convert string values into integers or floats:

- **Chest Pain Type:** The ChestPainType column, which contains values like 'ATA', 'NAP', 'ASY', and 'TA', is transformed into a new numerical column named ChestPainType_num. The mapping {'ATA': 0, 'NAP': 1, 'ASY': 2, 'TA': 3} assigns a unique integer to each type. This specific ordering (0 to 3) is based on a common assumption for this dataset, where ATA might be considered the least severe and TA the most.
- data['ChestPainType_num'] = data['ChestPainType'].map({'ATA': 0, 'NAP': 1, 'ASY': 2, 'TA': 3})

- **Resting ECG:** Similarly, the RestingECG feature, with categories like 'Normal', 'ST', and 'LVH', is converted into RestingECG_num using the mapping {'Normal': 0, 'ST': 1, 'LVH': 2}. This allows the model to process these distinct ECG patterns numerically.
- data['RestingECG_num'] = data['RestingECG'].map({'Normal': 0, 'ST': 1, 'LVH': 2})

- **ST Slope:** The ST_Slope feature, which describes the slope of the ST segment during exercise, has values 'Up', 'Flat', and 'Down'. These are mapped to ST_Slope_num as {"Up": 0, "Flat": 1, "Down": 2}. This numerical representation captures the different types of slopes.
- data["ST_Slope_num"] = data["ST_Slope"].map({"Up": 0, "Flat": 1, "Down": 2})

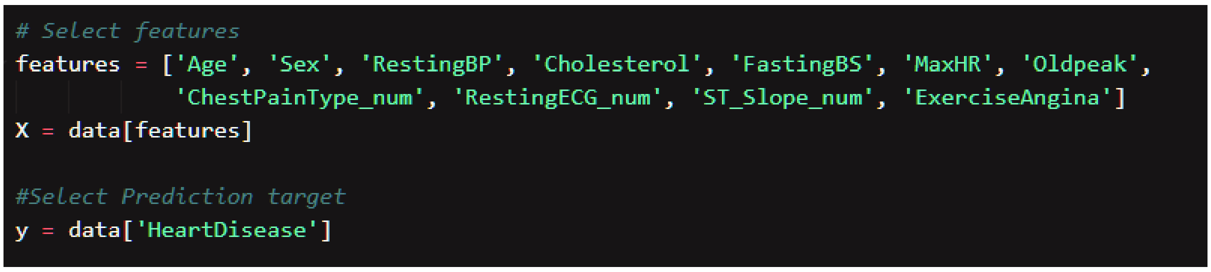
  - **Exercise Angina:** This binary categorical feature ('N' for No, 'Y' for Yes) is directly converted into numerical form within the same column. The mapping {"N": 0, "Y": 1} is applied, where 0 represents no exercise-induced angina and 1 represents its presence.
  - data["ExerciseAngina"] = data["ExerciseAngina"].map({"N": 0, "Y": 1})
  - **Sex:** The Sex feature ('M' for Male, 'F' for Female) is also converted directly into numerical form. The mapping {"M": 1, "F": 0} assigns 1 to male and 0 to female, a common encoding for binary gender data in machine learning.
  - data["Sex"] = data["Sex"].map({"M": 1, "F": 0})
- These transformations create new numerical columns (e.g., ChestPainType_num) or directly overwrite existing ones (ExerciseAngina, Sex) with their numerical equivalents. This step is fundamental as it converts human-readable labels into machine-understandable numerical features, without which the Logistic Regression model would be unable to learn.
- **Feature Selection:** After preprocessing, it's essential to define which columns will serve as the input features (predictors) for our machine learning model. In Final_Model.py, these features are explicitly listed and selected to form the X DataFrame:
- features = ['Age', 'Sex', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak',
-         'ChestPainType_num', 'RestingECG_num', 'ST_Slope_num', 'ExerciseAngina']
- X = data[features]

This step ensures that only the relevant and properly formatted numerical data points are passed to the model for training, excluding any original categorical columns that have been replaced by their numerical counterparts.

- **Target Variable Selection:** Equally important is the identification of the target variable, which is the outcome our model is designed to predict. In our case, this is the HeartDisease column, which indicates the presence or absence of heart disease. This column is separated from the features and assigned to y:
- y = data['HeartDisease']

This clear separation of features (X) and target (y) is a standard requirement for supervised machine learning algorithms.

These comprehensive preprocessing steps are absolutely vital. They transform the raw, mixed-type data from heart.csv into a clean, consistent, and entirely numerical format. This meticulously prepared data is what enables the Logistic Regression model to effectively learn the underlying patterns and make accurate predictions. Without this stage, the model would simply not be able to process the information, leading to errors or meaningless results.



```
# Select features
features = ['Age', 'Sex', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak',
            'ChestPainType_num', 'RestingECG_num', 'ST_Slope_num', 'ExerciseAngina']
X = data[features]

#Select Prediction target
y = data['HeartDisease']
```

5. Teaching the Model: Training and Evaluation

Once the data has been meticulously prepared through preprocessing, the next critical phase involves teaching our Logistic Regression model to recognize patterns and then rigorously evaluating its performance. This stage is where the "learning" in machine learning truly happens, and where we assess how well our model has understood the complexities within the heart.csv dataset.

Data Splitting

A fundamental principle in machine learning, crucial for building robust and generalizable models, is the practice of splitting the dataset. This process divides the available data into distinct subsets: a training set and a testing set. The purpose of this division is twofold:

1. **Training:** The model learns from the patterns present in the training data.
2. **Evaluation:** The model's ability to generalize to *unseen* data is assessed using the testing data. This helps prevent **overfitting**, a scenario where a model performs exceptionally well on the data it was trained on but poorly on new, unfamiliar data.

In our Final_Model.py script, this essential step is achieved using the train_test_split function, a powerful utility from sklearn.model_selection (Scikit-learn's module for model selection and evaluation utilities):

```
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,
random_state=42)
```

Let's break down the parameters:

- X: This represents the DataFrame containing our selected input features (e.g., 'Age', 'Sex', 'Cholesterol', etc.), which are the independent variables the model will use to make predictions.
- y: This is the Series containing our target variable, HeartDisease, which is the dependent variable the model aims to predict.
- test_size=0.2: This parameter dictates the proportion of the dataset that will be reserved for the testing set. In our case, 0.2 means that 20% of the heart.csv data will be set aside for testing, while the remaining 80% will be used for training the model. This is a common split ratio that balances sufficient data for training with a reasonable amount for unbiased evaluation.
- random_state=42: This parameter is crucial for reproducibility. When train_test_split shuffles the data before splitting, random_state provides a seed for the random number generator. By setting it to a fixed integer (like 42), we ensure that the data is shuffled and split in the exact same way every time the script is run. This guarantees that our model's training and evaluation results are consistent and can be replicated, which is vital for debugging and comparing different model iterations.

The result of this function call is four distinct sets of data:

- X_train: Features for training the model.
- X_test: Features for testing the model.
- y_train: Corresponding target labels for the training features.

- y_test: Corresponding actual target labels for the test features, used to compare against the model's predictions.

This careful partitioning of the dataset is a cornerstone of robust machine learning practice, allowing us to train the model on one subset of data and then objectively assess its performance on completely new, unseen data.

Model Training

With the data meticulously split into training and testing sets, the next step is to actually "teach" our Logistic Regression model. This process, known as model training, involves feeding the X_train (training features) and y_train (training target labels) to the model, allowing it to learn the underlying patterns and relationships.

In Final_Model.py, the training process is straightforward:

```
from sklearn.linear_model import LogisticRegression

# ...

model = LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)
```

Let's break down these lines:

- model = LogisticRegression(max_iter=1000): This line initializes an instance of the LogisticRegression classifier from Scikit-learn's sklearn.linear_model module. The max_iter=1000 parameter is important. It specifies the maximum number of iterations that the optimization algorithm (which finds the best weights for the model) will run. For some datasets, the default number of iterations might not be enough for the model to fully converge (i.e., find the optimal set of weights). Increasing max_iter helps ensure that the model has sufficient opportunities to learn and settle on a stable solution, preventing warnings about non-convergence.

- model.fit(X_train, y_train): This is the core training command. The fit() method is where the Logistic Regression algorithm actually learns from the data. During this process, the model analyzes the X_train features and their corresponding y_train labels. It iteratively adjusts its internal parameters (the weights and bias term) to minimize the error between its predicted probabilities and the actual HeartDisease outcomes in the training set. Essentially, the model is learning the optimal mathematical function that best maps the input health parameters to the probability of heart disease.

After this fit() method completes, our model object is now "trained." It has learned from the patterns in the 80% of our heart.csv data and is ready to make predictions on new, unseen data.

Model Evaluation

Training a model is only half the battle; the other crucial half is evaluating how well it performs. This step assesses the model's ability to generalize its learning to data it has never encountered before. Our evaluation focuses on how accurately the model predicts heart disease on the X_test dataset.

In Final_Model.py, the evaluation proceeds as follows:

```
from sklearn.metrics import accuracy_score
```

```
# ...

# Predict

y_pred = model.predict(X_test)


# Evaluate

def test():

   Accuracy = accuracy_score(y_test, y_pred)

   return Accuracy
```

Let's detail these steps:

- **Prediction on Test Set:** y_pred = model.predict(X_test): After the model has been trained, we use its predict() method to generate predictions on the X_test dataset. Crucially, X_test contains features from patients that the model *did not see* during its training phase. This ensures that our evaluation is an unbiased measure of the model's generalization capability. The y_pred variable will store an array of binary predictions (0 or 1) for each patient in the test set.
- **Accuracy Metric:** The primary metric chosen for evaluating our model's performance is **accuracy**. Accuracy is a straightforward and intuitive metric that quantifies the proportion of correctly classified instances out of the total number of instances in the test set. It is calculated as:

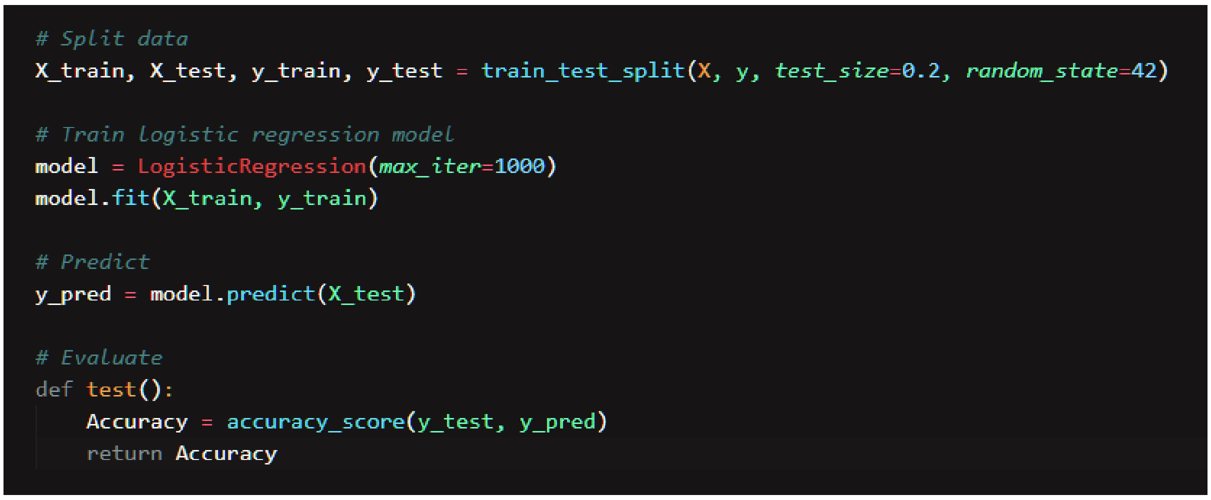
Accuracy=Total Number of PredictionsNumber of Correct Predictions

In Final_Model.py, the accuracy_score function from sklearn.metrics is used to compute this value: Accuracy = accuracy_score(y_test, y_pred).

This function compares the model's predictions (y_pred) against the true labels (y_test) from the test set.

- **test() Function:** The accuracy score is encapsulated within a function def test(): in Final_Model.py. This function is designed to be called externally, specifically by our Final_Version.py UI application, to retrieve the model's performance.
- accuracy = FM.test() # This line in Final_Version.py calls the test() function

This setup allows the UI to dynamically display the model's accuracy to the user, providing transparency about its general reliability. For instance, if accuracy_score returns 0.84, the test() function returns 0.84, which is then used in the UI to display "Model Accuracy: 84.00%".



```python
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
def test():
    Accuracy = accuracy_score(y_test, y_pred)
    return Accuracy
```

Model Persistence

After our Logistic Regression model has been rigorously trained and evaluated, it's essential to save this trained model so that it can be used later without needing to retrain it every single time the application starts. This process is known as **model persistence** or **serialization**. It's akin to taking a snapshot of the trained model's "brain" (its learned weights and parameters) and storing it in a file.

In Final_Model.py, the pickle library is used for this crucial step:

```
import pickle

# ...

# Save as pkl

with open("logistic_model.pkl", "wb") as file:

   pickle.dump(model, file)
```

Let's break this down:

- import pickle: This line imports the pickle module, which is a standard Python library used for serializing and deserializing Python objects. Serialization converts a Python object (like our trained model) into a byte stream, which can then be stored in a file. Deserialization reverses this process, reconstructing the original Python object from the byte stream.
- with open("logistic_model.pkl", "wb") as file:: This line opens a file named logistic_model.pkl in binary write mode ("wb"). The with statement ensures that the file is properly closed even if errors occur. The .pkl extension is a common convention for files saved using the pickle module.
- pickle.dump(model, file): This is the core command that performs the serialization. It takes our trained model object and writes its serialized representation into the opened file.

This logistic_model.pkl file now contains all the information needed to recreate our trained Logistic Regression model, including its learned coefficients and internal state.

Loading the Model in the UI (Final_Version.py)

The Final_Version.py application, which provides the user interface, then loads this pre-trained model at its startup. This is done using pickle.load:

```
import pickle

# ...

try:

   model = pickle.load(open("logistic_model.pkl", "rb"))

except FileNotFoundError:

   QMessageBox.critical(None, "Model Error", "logistic_model.pkl not
found. Please ensure the model file is in the same directory.")

   sys.exit(1)
```

- model = pickle.load(open("logistic_model.pkl", "rb")): This line opens the logistic_model.pkl file in binary read mode ("rb") and then uses pickle.load to reconstruct the original LogisticRegression model object from the stored byte stream. This means the UI application doesn't need to retrain the model every time it launches; it can simply load the already "smart" model.
- **Error Handling:** The try-except FileNotFoundError block is a crucial piece of robust programming. If the logistic_model.pkl file is not found in the same directory as Final_Version.py, the

application will display a critical error message to the user via QMessageBox.critical and then exit. This prevents the application from crashing unexpectedly and provides clear guidance to the user on how to resolve the issue.

This separation of model training (in Final_Model.py) and model deployment/usage (in Final_Version.py) through persistence is a highly efficient and flexible approach in machine learning application development. It allows for independent updates to the model without altering the UI, and vice versa.

6. User Interface Design: Crafting the User Experience

The user interface (UI) is the direct point of interaction between the user and our sophisticated predictive model. Its design is paramount to ensuring that the application is not only functional but also intuitive, accessible, and pleasant to use. A well-designed UI can transform a complex analytical tool into a practical and engaging application for everyday users.

Design Principles

Our UI design adheres to several core principles that guided its development and visual aesthetic:

- **Clarity:** All labels, instructions, and results are presented clearly and concisely, minimizing ambiguity. Users should instantly understand what information is required and what the output signifies. This means using straightforward language and avoiding jargon where possible.

- **Simplicity:** The layout is kept clean and uncluttered, focusing on essential elements to avoid overwhelming the user with too much information at once. This minimalist approach, evident in the black-and-white theme applied in Final_Version.py, helps users focus on the task of inputting data and receiving predictions without distractions.
- **Consistency:** UI elements and interactions are consistent throughout the application. For instance, input fields behave similarly, and buttons have a predictable appearance and response. This consistency makes the application predictable and easy to learn, as users can apply knowledge gained from one part of the interface to another.
- **Feedback:** The system provides immediate and understandable feedback to user actions. Whether it's a successful prediction, an input error, or a missing file, the application communicates its status clearly. This instant feedback loop reassures the user and guides them through any issues.
- **Accessibility (General Usability):** While not fully compliant with formal accessibility standards, the design aims for good contrast between text and background colors and uses legible font sizes (QFont('Inter', 14px) for general text, QFont('Inter', 24, QFont.Bold) for titles) to improve general usability for a wide range of users. The choice of the 'Inter' font family ensures modern readability.

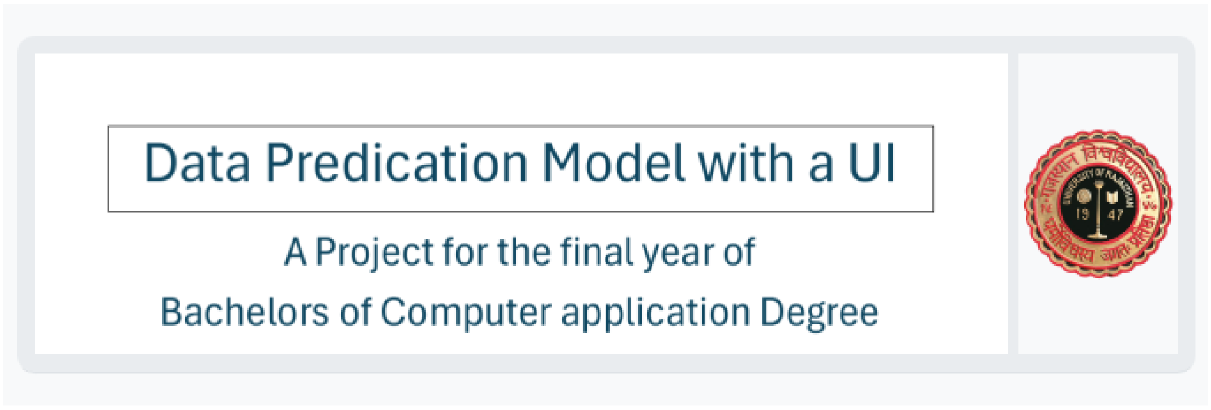
Detailed UI Components and Workflow

The init_ui method within the HeartDiseasePredictor class in Final_Version.py is the orchestrator of the entire UI. It meticulously arranges and configures all the visual components, defining their appearance, position, and initial state.

- **Main Layout (QVBoxLayout):** The entire application window's content is organized vertically using a QVBoxLayout. This layout manager stacks widgets one after another from top to bottom. Generous spacing (main_layout.setSpacing(25)) between widgets and ample margins (main_layout.setContentsMargins(30, 30, 30, 30)) around the content provide a comfortable and uncrowded visual experience, enhancing readability and reducing visual fatigue.
- **Header Section (QFrame with QHBoxLayout):**
  - A QFrame object, specifically named "headerFrame" using setObjectName, serves as a dedicated container for the top section of the UI. This allows for specific styling to be applied to this area via the stylesheet.
  - Inside this frame, a QHBoxLayout (horizontal layout) arranges elements side-by-side.
  - self.image_label: This QLabel is intended to display a banner image (image.png). The code attempts to load this image and scale it to a width of 550 pixels, maintaining aspect ratio (Qt.SmoothTransformation). Crucially, a try-except block is implemented. If image.png is not found or fails to load, self.image_label will display "Banner Image Not Found" in red, providing clear feedback to the user or developer. This demonstrates robust handling of external assets.
  - self.logo_label: Similar to the banner, this QLabel is meant for a logo (image2.png), scaled to 90 pixels wide. It also

includes the same try-except mechanism for error handling, displaying "Logo Not Found" if the image is unavailable.

- The header_layout.addStretch(1) command within the QHBoxLayout ensures that the banner image is aligned to the left and the logo to the right, with flexible spacing between them.
- The headerFrame itself is styled in apply_modern_stylesheet with a light gray background (#e9ecef), a bottom border, and rounded corners, giving it a distinct and polished look.

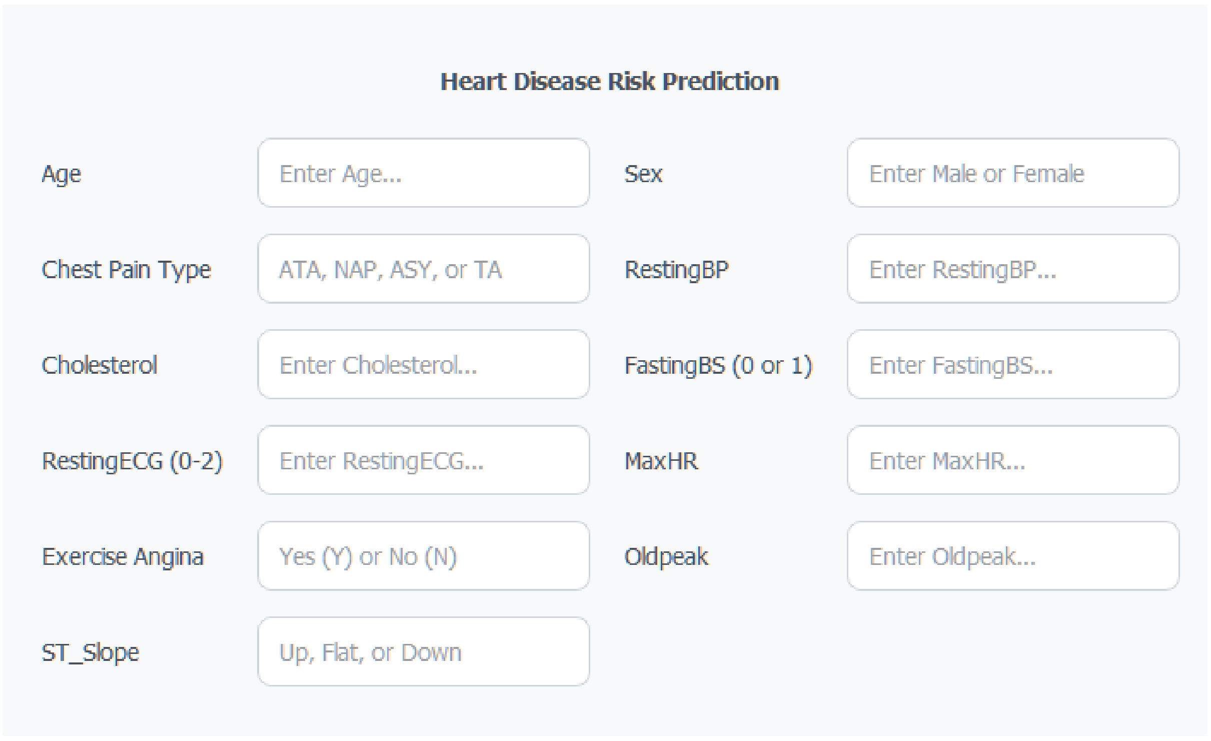

Data Predication Model with a UI
A Project for the final year of
Bachelors of Computer application Degree

- **Application Title (QLabel):**
  - A central QLabel displays the main title: "Heart Disease Risk Prediction."
  - It's styled with a large, bold 'Inter' font (QFont('Inter', 24, QFont.Bold)) and aligned to the center (Qt.AlignCenter), ensuring it immediately captures the user's attention and clearly states the application's purpose.
- **Input Grid (QGridLayout):** The core of the user interaction happens within this QGridLayout. A grid layout is ideal for

arranging pairs of labels and input fields in a structured, two-column format, which is common for forms.

- grid_layout.setSpacing(15) and grid_layout.setHorizontalSpacing(20) provide ample vertical and horizontal spacing between elements, preventing a cramped appearance and improving readability.
- The labels list in init_ui ("Age", "Sex", "Chest Pain Type", ...) defines the text for each QLabel that prompts the user for specific health parameters. Each label is styled with a slightly smaller font (QFont('Inter', 13)) for readability.
- For each label, a corresponding QLineEdit widget is created for user input.
- **Dynamic Placeholder Texts:** A key usability feature is the dynamic assignment of placeholder texts to QLineEdit fields. For numerical inputs, it's a generic "Enter [Field Name]...". However, for categorical inputs, the placeholder text explicitly lists the expected valid options. For example:
  - For "Sex": "Enter Male or Female"
  - For "Chest Pain Type": "ATA, NAP, ASY, or TA"
  - For "Exercise Angina": "Yes (Y) or No (N)"
  - For "ST_Slope": "Up, Flat, or Down" This direct guidance is crucial for helping users enter data in the correct format, directly referencing the string values that are later mapped to numbers in Final_Model.py. This proactive guidance minimizes input errors and improves the user experience significantly.

- All QLineEdit fields are styled in apply_modern_stylesheet with a white background (#ffffff), dark text (#343a40), a subtle gray border (#ced4da), and rounded corners (border-radius: 8px). When an input field is focused, its border turns blue (#007bff), providing clear visual feedback on active input.
- The self.inputs list stores references to all QLineEdit widgets, allowing the make_prediction method to easily retrieve their contents.


Heart Disease Risk Prediction

Age	Enter Age...	Sex	Enter Male or Female
Chest Pain Type	ATA, NAP, ASY, or TA	RestingBP	Enter RestingBP...
Cholesterol	Enter Cholesterol...	FastingBS (0 or 1)	Enter FastingBS...
RestingECG (0-2)	Enter RestingECG...	MaxHR	Enter MaxHR...
Exercise Angina	Yes (Y) or No (N)	Oldpeak	Enter Oldpeak...
ST_Slope	Up, Flat, or Down		

- **Result Display (QLabel):**
  - The self.result_label is a central and highly dynamic component. Initially, it displays "Awaiting Input."
  - After a prediction is made, its text is updated with the prediction outcome (e.g., "Positive for Heart Disease") and the model's accuracy (e.g., "Model Accuracy: 84.00%").

    - Crucially, the setStyleSheet method is used to dynamically change its color based on the prediction: a prominent red (#dc3545) for "Positive for Heart Disease" and a reassuring green (#28a745) for "No Heart Disease." This instant visual feedback is highly intuitive and immediately conveys the result's implication.
    - Its font is bold and larger (QFont('Inter', 18, QFont.Bold)) and it's centrally aligned, ensuring it immediately captures the user's attention.
- **Prediction Button (QPushButton):**
    - The "Predict possibility of Heart Disease" button is styled for prominence and user engagement, with a blue background (#007bff), white text, rounded corners, and bold font. It also includes a hover effect (background-color: #0056b3) for better interactivity.
    - Its clicked signal is directly connected to the self.make_prediction method. This connection is the trigger that initiates the entire prediction logic, from input validation to model inference and result display.

User Workflow

The user workflow is designed to be simple, intuitive, and linear, guiding the user through the prediction process with minimal effort:

1. **Launch Application:** The user initiates the Final_Version.py application. The UI loads, displaying the header, title, empty input fields with placeholder texts, and the "Awaiting Input" message.

2. **Input Data:** The user proceeds to fill in all the required health parameters in the respective QLineEdit input fields. The dynamic placeholder texts guide them on the expected format for each entry.
3. **Initiate Prediction:** Once all fields are filled, the user clicks the "Predict possibility of Heart Disease" button. This action triggers the make_prediction method.
4. **Validation and Prediction:**
    - The make_prediction method first performs robust input validation. It checks if all fields are filled and if the data types and categorical values conform to the expected format.
    - If inputs are valid, the data is preprocessed (as described in Section 2.2) and converted into a NumPy array suitable for the model.
    - This processed data is then fed to the loaded Logistic Regression model (model.predict(data)), which performs the inference.
5. **Display Results:** The prediction outcome (Heart Disease or No Heart Disease) and the model's accuracy are immediately displayed in the result_label. The color of the text (red or green) provides instant visual feedback on the prediction.
6. **Error Handling:**
    - If any input validation fails (e.g., an empty field, incorrect categorical input, non-numerical value), a QMessageBox.warning pop-up appears, clearly stating the error and guiding the user to correct it.

    - If an unexpected error occurs during the prediction process (e.g., an issue with the model file or an unforeseen computational problem), a QMessageBox.critical is displayed, providing a more serious error message.

This structured workflow, combined with clear visual feedback and robust error handling, ensures a smooth, predictable, and intuitive user experience, making the complex machine learning process accessible to a broad audience.

7. Results and Performance Analysis

The effectiveness and reliability of our Heart Disease Risk Predictor are primarily communicated through its results and an analysis of its performance. This section details how the prediction outcomes are presented to the user and the key metric that quantifies the underlying machine learning model's capabilities.

Prediction Outcome Presentation

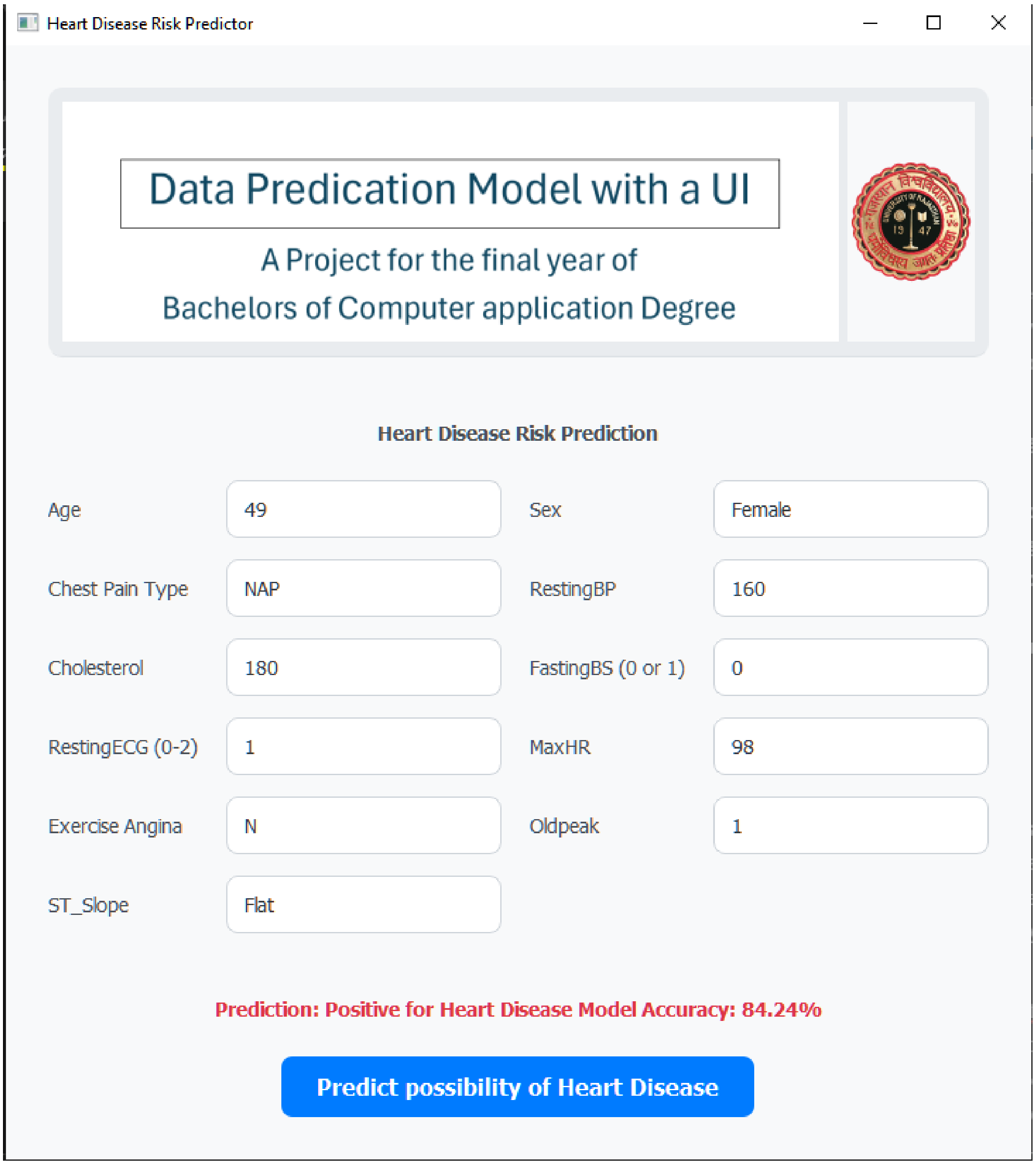
The most direct and immediate output of the application is the prediction of heart disease risk. As implemented in the make_prediction method within Final_Version.py, the result_label dynamically updates to present one of two clear outcomes:

- **"Positive for Heart Disease"**: This message is displayed if the prediction returned by the Logistic Regression model is 1, indicating the model's assessment that heart disease is likely present based on the provided inputs.
- **"No Heart Disease"**: This message appears if the prediction from the model is 0, suggesting that heart disease is not indicated by the given health parameters.

To enhance clarity and provide immediate visual comprehension, a crucial visual cue is employed through dynamic styling:

- If prediction == 1, the result_label's text is styled in a prominent **red color** (#dc3545). This color choice is universally associated with warnings or critical information, effectively conveying the significance of a "Positive for Heart Disease" prediction.
- If prediction == 0, the result_label's text is styled in a reassuring **green color** (#28a745). Green typically signifies positive outcomes or safety, providing a comforting visual confirmation for a "No Heart Disease" prediction.

This intuitive color-coding, combined with the clear textual message, ensures that users can instantly grasp the assessed risk level without needing to interpret complex numerical data, making the outcome highly accessible.


Heart Disease Risk Predictor

Data Predication Model with a UI

A Project for the final year of
Bachelors of Computer application Degree

Heart Disease Risk Prediction

Age	49	Sex	Female
Chest Pain Type	NAP	RestingBP	160
Cholesterol	180	FastingBS (0 or 1)	0
RestingECG (0-2)	1	MaxHR	98
Exercise Angina	N	Oldpeak	1
ST_Slope	Flat		

Prediction: Positive for Heart Disease Model Accuracy: 84.24%

Predict possibility of Heart Disease

Model Accuracy as a Key Metric

Beyond the binary prediction, our application transparently displays a critical performance metric: the model's overall **accuracy**. This metric, obtained from the FM.test() function defined in our Final_Model.py script, quantifies the proportion of correct predictions made by the Logistic Regression model on the unseen test data.

The accuracy variable, calculated using accuracy_score(y_test, y_pred) in Final_Model.py, represents the percentage of instances (patient records) in the test set for which the model's prediction matched the actual HeartDisease label. This value is then formatted to two decimal places and presented to the user alongside the prediction (e.g., "Model Accuracy: 84.00%").

This level of transparency regarding the model's general reliability is crucial for building user trust. For instance, if FM.test() returns a value of 0.84, the UI will display 84.00%. This indicates that, based on the test data, the model correctly classified 84% of the heart disease cases (both positive and negative). This provides users with a concise summary of the model's overall correctness in classifying individuals.

Significance and Interpretation of Results

The results generated by our Heart Disease Risk Predictor, while not intended to replace professional medical diagnosis, hold significant value as a preliminary risk assessment tool. By providing a data-driven indication of heart disease likelihood, the application can serve several important purposes:

- **Raising Health Awareness:** The interactive nature of the tool encourages individuals to engage with their health parameters and consider how various factors might influence their cardiovascular risk. This can lead to increased personal health awareness.
- **Prompting Medical Consultation:** For users who receive a "Positive for Heart Disease" prediction, the application can serve

as a catalyst, potentially motivating them to seek further medical evaluation, professional diagnosis, and personalized advice from qualified healthcare professionals. It acts as an early warning signal, encouraging proactive health management.

- **Educational Value:** The tool implicitly educates users about the types of health data that are relevant to heart disease risk, such as cholesterol levels, blood pressure, and chest pain types. This can empower individuals with a better understanding of their own health profile.

The model's accuracy, as derived from Final_Model.py's evaluation on the heart.csv dataset, underpins the credibility of these predictions. An accuracy of 84% suggests a reasonably effective model for initial screening purposes. However, it is vital to acknowledge that in a real-world clinical setting, a deeper and more nuanced analysis involving additional performance metrics would be crucial. Metrics such as:

- **Precision:** The proportion of actual positive cases among all cases predicted as positive (minimizing false positives).
- **Recall (Sensitivity):** The proportion of actual positive cases that were correctly identified as positive (minimizing false negatives).
- **F1-Score:** The harmonic mean of precision and recall, providing a balanced measure.
- **Specificity:** The proportion of actual negative cases that were correctly identified as negative.

These metrics provide a more comprehensive understanding of the model's ability to minimize both false positives (incorrectly identifying heart disease, leading to unnecessary anxiety or tests) and false negatives (failing to identify existing heart disease, which could have severe consequences). While our current UI prioritizes a clear, single accuracy score for user simplicity, these deeper metrics are essential for robust model validation and would be critical if the application were to be used in a clinical diagnostic context. The current implementation focuses on providing a clear and accessible risk indication for general awareness.

8. Conclusion

In summary, this project successfully developed a functional, intuitive, and user-friendly "Heart Disease Risk Predictor" application. We have achieved our core objective of creating a simple yet meaningful tool that leverages the power of machine learning to provide preliminary insights into an individual's potential risk of heart disease. The project's success is a testament to the effective integration of several key components:

- **A Robust Predictive Model:** We employed a Logistic Regression model, meticulously trained on a comprehensive heart.csv dataset. This model forms the intelligent core, capable of discerning patterns within various health indicators to make informed predictions.
- **Thoughtful User Interface Design:** The Graphical User Interface (GUI), developed using PyQt5, was designed with a strong emphasis on user experience. Its clean layout, intuitive input fields, and clear visual feedback mechanisms ensure that complex analytical capabilities are accessible and actionable for a broad audience.
- **Rigorous Data Handling:** The project incorporates robust input validation and meticulous data preprocessing, ensuring that the

model receives clean, correctly formatted data. This foundational step is critical for the accuracy and reliability of the predictions.

- **Transparent Output:** The application provides clear and immediate prediction outcomes, enhanced by intuitive color-coding. Crucially, it also transparently displays the model's overall accuracy, offering users insight into the system's general reliability.

Ultimately, this application serves as a practical demonstration of how fundamental data science principles and software engineering practices can converge to create valuable tools in the healthcare domain. It represents a significant learning outcome in the process of combining data acquisition, model building, and application development, highlighting the potential of technology to contribute to public health awareness and preliminary risk assessment.

User Interface Design

The user interface (UI) design prioritizes clarity, user-friendliness, and immediate feedback, allowing users to seamlessly interact with the analytical core of the application.

1. Structured Layout and Navigation: The application's main window features a well-organized layout, employing common design patterns such as vertical, horizontal, and grid-based arrangements. This structured approach ensures that input fields, controls, and results are logically grouped and presented, contributing to a clean and uncluttered visual environment. The logical flow guides the user effortlessly through the process of data entry and prediction.

2. Intuitive Input Mechanism: User data is captured through a series of dedicated input fields, each clearly labeled to specify the required health parameters (e.g., age, blood pressure, cholesterol, and other relevant medical indicators). These fields are designed to be straightforward for data entry, with integrated validation mechanisms that prompt users if inputs are in an incorrect format, thereby ensuring data integrity before processing.

3. Clear Output and Feedback: Upon processing, the system provides immediate and discernible feedback. The prediction outcome (e.g., presence or absence of a condition) is prominently displayed, often accompanied by visual cues such as distinct color-coding—for instance, an affirmative color for a negative result and an alert color for a positive one. Additionally, key performance metrics, such as the model's accuracy, are presented alongside the prediction to provide transparency regarding the system's reliability.

Result and Performance Analysis

1. Prediction Outcome: Upon processing the user's health parameters, the system delivers a clear and immediate prediction. This outcome is typically binary, indicating either "Positive for Heart Disease" or "No Heart Disease." The prediction is presented prominently within the user interface, often with visual cues such as color-coding to intuitively convey the result (e.g., a specific color for a positive indication and another for a negative one). This direct feedback mechanism allows users to quickly understand their assessed risk level based on the provided inputs.

2. Model Accuracy: A fundamental aspect of performance analysis for any machine learning model is its accuracy. In this project, the model's performance is quantitatively assessed, and its **accuracy score** is a key metric displayed to the user. Accuracy represents the proportion of correct predictions made by the model out of all predictions. For instance, if the model has an accuracy of 85%, it means 85% of its predictions on unseen data were correct. This metric provides a concise summary of the model's overall correctness in classifying individuals with and without heart disease.

3. Significance of Results: The results provided by this system hold significant value in preliminary risk assessment. By offering a quick, data-driven prediction of heart disease risk, the application can serve as an informative tool, potentially encouraging individuals to seek professional medical advice for further diagnosis and management. The model's performance, particularly its accuracy, underpins the credibility and potential utility of these predictions.

Conclusion

The development of the "Data Prediction Model with a UI" project represents a significant accomplishment, culminating in a functional and accessible application specifically designed to provide an initial assessment of heart disease risk. Our primary achievement lies in the successful and seamless integration of machine learning capabilities with a user-friendly graphical interface, a combination that makes complex predictive analytics far more approachable and understandable for a broader audience, including individuals without specialized technical knowledge. This project showcases a practical application of data science in a critical domain.

At its core, this predictive system leverages a **Logistic Regression model**, a robust and interpretable machine learning algorithm. This model was meticulously trained on a comprehensive dataset, specifically the heart.csv file, which contains a rich collection of patient health indicators. As detailed in our Final_Model.py script, the model learns intricate patterns and relationships from various factors such as age, cholesterol levels, resting blood pressure, chest pain types, and other relevant clinical parameters. Through this rigorous training process, the model develops the ability to discern subtle correlations that contribute to the likelihood of heart disease. The application's user interface, expertly crafted using the **PyQt5** framework as demonstrated in Final_Version.py, serves as the intuitive gateway for user interaction. This interface allows users to effortlessly input their specific health details into clearly labeled fields. In return, the system provides a clear and immediate prediction of heart disease likelihood, alongside a transparent display of the model's overall accuracy. This dual presentation of results, coupled with intuitive visual cues (such as distinct color-coding for positive or negative outcomes), significantly enhances user understanding and fosters a sense of trust in the application's output. The design emphasizes clarity and ease of use, ensuring that the predictive power is readily accessible.

It is absolutely crucial to reiterate, with utmost clarity, that this application functions solely as a **preliminary risk assessment tool**. Its design and purpose are to offer a general indication of potential risk based on the data provided by the user. It serves as a helpful starting point for personal health awareness and can be a valuable informational resource. However, it is fundamentally **not a substitute for professional medical advice, diagnosis, treatment, or the expertise of qualified healthcare professionals**. The results generated by this tool should never be used as the sole basis for making medical decisions. We strongly and unequivocally advise all users to consult with their physicians or other healthcare providers for any health concerns, for accurate diagnosis, and for personalized medical guidance. In essence, think of our application as a valuable and data-informed conversation starter about personal well-being, designed to empower individuals with initial insights rather than definitive medical conclusions.

Looking ahead, the journey of this project offers significant and exciting potential for further refinement, expansion, and increased sophistication. As we've outlined in our "Future Enhancements" section, there are numerous avenues to explore that could enhance its utility and impact in supporting personal health management. This includes, but is not limited to, the exploration and implementation of more advanced machine learning models (such as ensemble methods or even basic neural networks) that might uncover deeper, more complex patterns within health data. Furthermore, incorporating richer and more diverse datasets would significantly improve the model's generalization capabilities, making it more robust across various demographics and clinical presentations. Integrating more sophisticated user features, such as personalized health recommendations (always generic and advising professional consultation), trend analysis over time, or even interactive data visualizations, could transform this tool into a more comprehensive and engaging platform for health awareness. The continuous evolution of this project holds the promise of making data-driven health insights even more powerful and accessible to individuals seeking to understand and proactively manage their well-being. This ongoing development underscores our commitment to leveraging technology for positive societal impact in the realm of health.

Future Enhancements

Future Enhancements and Goals

While the current iteration of the Heart Disease Risk Predictor is a robust and functional achievement, we recognize that the field of machine learning and its application in healthcare are continuously evolving. Therefore, there is always room for growth, refinement, and expansion, especially for a tool as sensitive and impactful as a health predictor. Our future goals aim to push the boundaries of this project further, enhancing its accuracy, utility, and accessibility:

- **Improve Model Accuracy and Sophistication:** The current accuracy of 84% is a commendable starting point, particularly for an initial screening tool. However, for a heart disease prediction model, even a 1/5 chance of being incorrect can have significant implications. Our primary goal is to explore and implement more advanced machine learning algorithms. This could include:
    - **Ensemble Methods:** Such as **Random Forests** or **Gradient Boosting Machines (e.g., XGBoost, LightGBM)**, which combine multiple decision trees to produce more robust and accurate predictions.
    - **Support Vector Machines (SVMs):** Powerful classifiers that can find optimal hyperplanes to separate data points, even in high-dimensional spaces.
    - **Basic Neural Networks/Deep Learning:** For highly complex patterns, even a shallow neural network could potentially uncover non-linear relationships that Logistic Regression might miss. This enhancement would involve extensive

experimentation with different model architectures, hyperparameter tuning strategies (e.g., using GridSearchCV or RandomizedSearchCV), and potentially cross-validation techniques beyond simple train-test splits (e.g., k-fold cross-validation) to ensure the model's robustness and generalization capabilities.

- **Explore Better and Larger Datasets:** The current heart.csv dataset, while valuable for demonstrating the project's core functionality, is relatively modest in size. A larger, more diverse, and potentially longitudinal dataset would allow the model to learn more generalized and nuanced patterns. This would lead to more accurate and reliable predictions across a broader patient population, accounting for various demographics, geographical locations, and clinical presentations. We would actively seek out more extensive and varied clinical data sources, ensuring ethical considerations and data privacy are paramount.
- **Deeper Understanding and Interpretability (Explainable AI - XAI):** For a tool that provides health-related predictions, merely stating a binary outcome is often insufficient. Knowing *why* the AI made a certain prediction is incredibly valuable for both users and potential clinicians. We aim to integrate **Explainable AI (XAI)** techniques to provide transparent insights into the model's decision-making process. This could involve:
    - **SHAP (SHapley Additive exPlanations) values:** To quantify the contribution of each feature to a specific prediction.
    - **LIME (Local Interpretable Model-agnostic Explanations):** To explain individual predictions by creating a simpler, interpretable model around the prediction. Adding these features would allow the application to explain *which* health

factors (e.g., "Your high cholesterol and age were significant contributors to this prediction") contributed most to a user's specific risk assessment. This transparency would build greater trust in the system and help users understand the underlying reasons for the prediction, potentially guiding them toward specific lifestyle changes or medical discussions.

- **Enhanced User Experience and Features:** Beyond the core prediction, several additions could significantly improve the user experience and overall utility of the application:
    - **Personalized, Generic Recommendations:** Based on a "Positive" prediction or the identification of specific high-risk factors, the app could offer generic, actionable lifestyle advice (e.g., "Consider incorporating regular exercise into your routine," "Consult a doctor about monitoring cholesterol levels," "Maintain a balanced diet"). These recommendations would be general and not medical advice, always prompting consultation with a professional.
    - **Trend Analysis and User Profiles:** Allow users to create secure profiles, save their inputs over time, and visualize trends in their health parameters and risk assessments. This could help individuals track their progress or identify changes in their risk factors over months or years. This would likely require integrating a secure database solution.
    - **Web/Mobile Deployment:** While currently a desktop application, developing a web-based version (e.g., using Python frameworks like Flask or Django) or a dedicated mobile application would dramatically increase its accessibility and reach. This would allow users to access

    the tool from any device with an internet connection, expanding its impact.

    - **Interactive Visualizations:** Incorporate interactive charts or graphs directly within the UI to visualize how different input parameters relate to heart disease risk. For instance, a bar chart showing the relative importance of each feature for a given prediction, or a simple graph illustrating the distribution of certain health parameters within the dataset. This could be achieved using libraries like Matplotlib, Seaborn, or Plotly integrated with PyQt5.

- **Robust Error Handling and Edge Cases:** While current error handling is effective for common input issues, further refinement could involve addressing more complex edge cases or providing even more specific guidance for unusual or out-of-range inputs. This would involve more exhaustive input validation rules.
- **Security and Privacy Considerations (for potential deployment):** If the project were to evolve into a deployed application handling sensitive health data, a comprehensive plan for data security, privacy (adhering to regulations like HIPAA or GDPR), and user authentication would be paramount. This would involve secure data storage, encrypted communication, and robust access controls.

These future enhancements represent a detailed roadmap for transforming this foundational project into an even more sophisticated, insightful, and impactful tool for promoting heart health awareness and supporting preliminary risk assessment. They would push the boundaries of its current capabilities and move it closer to a real-world application.

Bibliography

1.The Dataset:

- **(Kaggle Dataset):**
    - Fedesoriano. (2021). *Heart Failure Prediction Dataset*. Kaggle. https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction

2. Key Libraries

- Numpy
- Pandas
- Sklearn
- PyQT5

3. Books for reference

- Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras*
- *Modern Approach to Artificial Intelligence by Russel Norvig*
- *Introduction to Large Language Models by Tanmoy Chakraborty*

4. GitHub Repository Link

1. https://github.com/KKhandal-0/Machine-Learning-HD-Project/tree/main