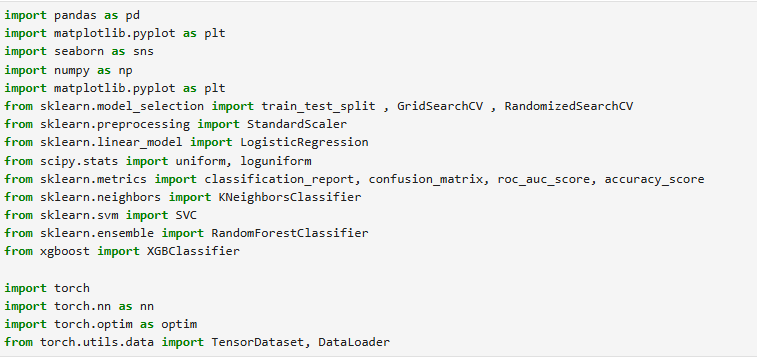
Predictive Modeling of Heart Disease

**Submitted By:** Manav Gupta

**Email:** [manav08gupta@gmail.com](mailto:manav08gupta@gmail.com)

**Mobile number:** +91 9250222787

# **Importing Libraries**

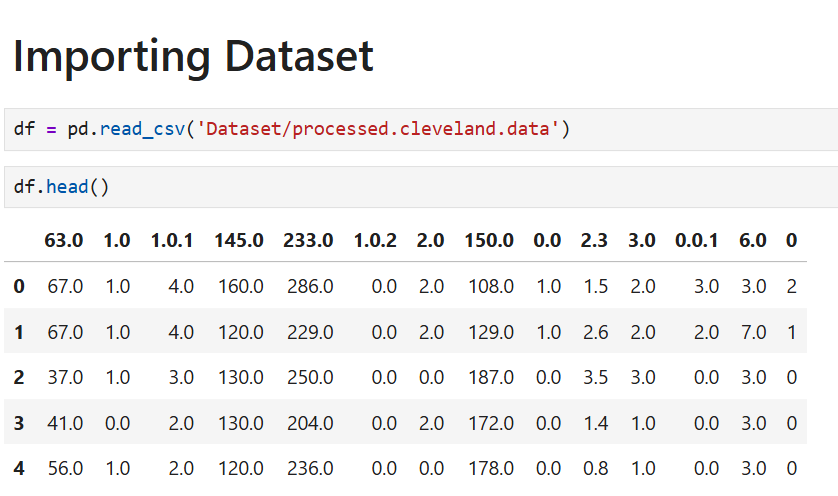


# **Importing Datasets**

## 1. Column Name Assignment

Upon loading the dataset, it was observed that the dataset **did not contain any column headers**. The data was structured in rows and columns, but the columns were unnamed and represented only by numerical indices.

To facilitate meaningful data analysis and ensure code readability, a header row was manually assigned based on standard definitions for the UCI Heart Disease dataset.



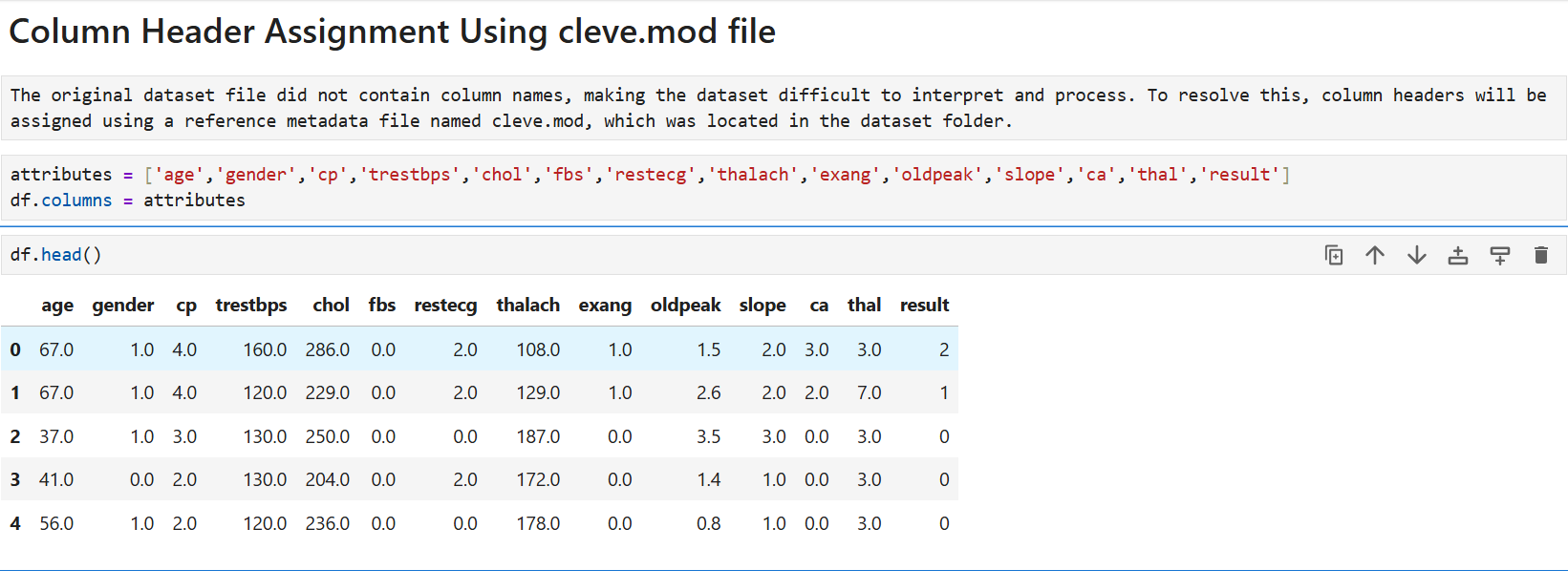
## 2. Column Header Assignment Using cleve.mod file

The original dataset file did not contain column names, making the dataset difficult to interpret and process. To resolve this, **column headers were assigned using a reference metadata file named cleve.mod**, which was located in the dataset folder.

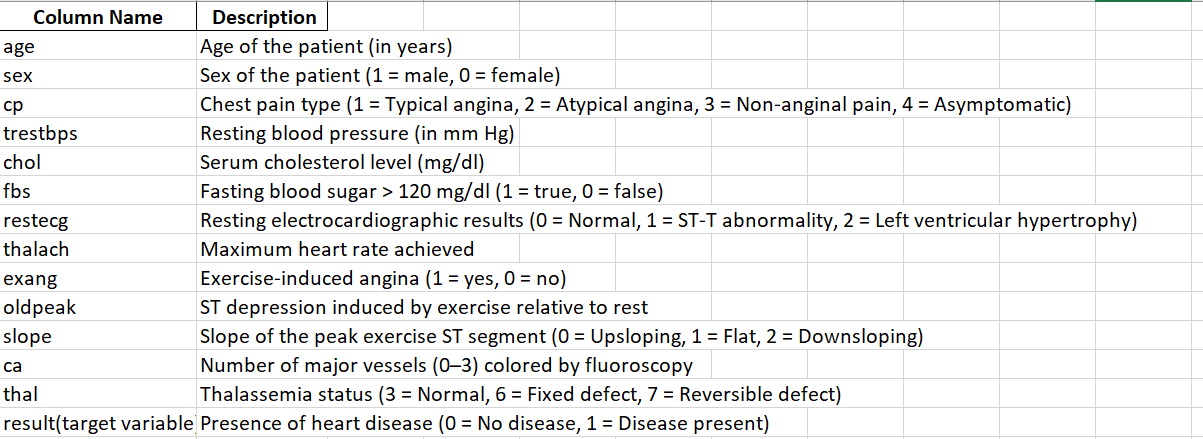
### Final Column Names:

The following columns were assigned based on the metadata from cleve.mod:

* age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal, result



## 3. Column Descriptions



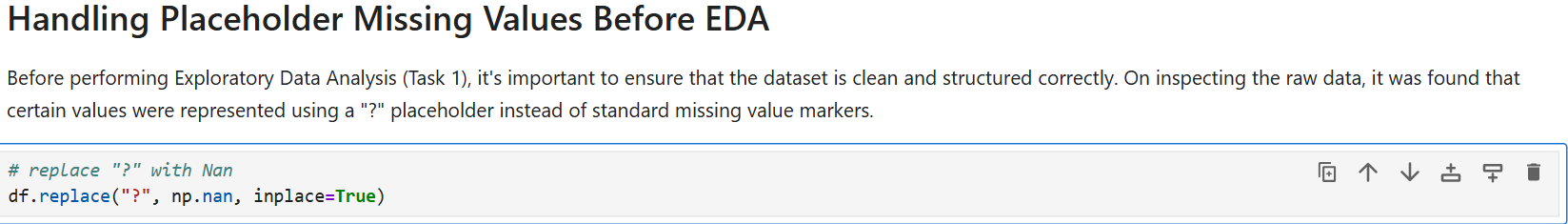
## 4. Handling Placeholder Missing Values Before EDA

Before performing Exploratory Data Analysis (Task 1), it's important to ensure that the dataset is clean and structured correctly. On inspecting the raw data, it was found that certain values were represented using a "?" placeholder instead of standard missing value markers.

To handle this issue:

* All "?" entries were replaced with NaN using NumPy.
* The data types of all columns were checked to identify any columns still being treated as objects due to non-numeric entries.

This step is essential to avoid errors and incorrect results during data visualization and statistical analysis.



# **Task 1:** **Data Understanding and Exploration (EDA)**

## **1.** Descriptive Statistics for Numerical and Categorical Features

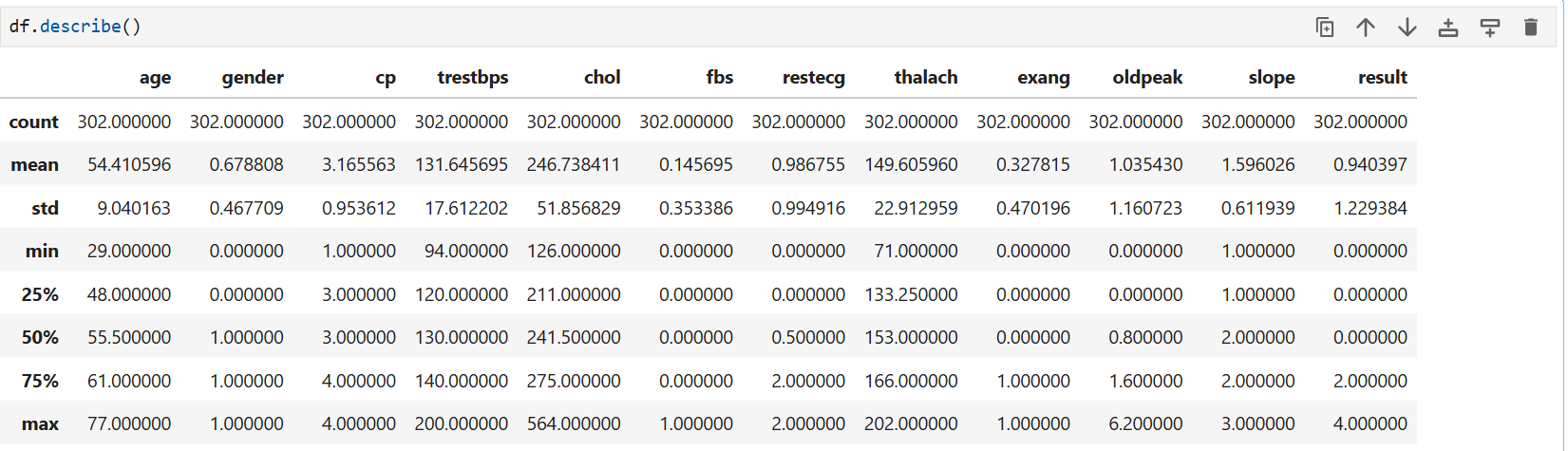
To begin our exploratory data analysis, we generate summary statistics for all the features in the dataset using the describe() function. This includes key metrics such as:

* **Count**: Number of non-null entries
* **Mean**: Average value
* **Standard deviation**: Spread of the values
* **Minimum and Maximum**: Range of values
* **25th, 50th, and 75th percentiles**: Distribution shape

These statistics help us:

* Understand the scale and distribution of each feature
* Detect potential outliers (via min/max)
* Identify features that may require normalization or transformation

This step is crucial to guide further preprocessing and model design decisions.



**Based on Above output below are the key insights:**

 **Age**: The average age of patients is approximately **54.4 years**, with a minimum of **29** and a maximum of **77** years, indicating a wide adult age range.

 **Gender**: The mean of **0.6788** indicates that a majority of patients are **male** (since male = 1).

 **Chest Pain Type (cp)**: Values range from **1 to 4**, with the median at 3, suggesting most patients experience asymptomatic or atypical angina.

 **Resting Blood Pressure (trestbps)**: A mean of **131.6 mm Hg**, with outliers potentially present up to **200 mm Hg**.

 **Cholesterol (chol)**: The maximum value is **564 mg/dl**, suggesting some outlier or extreme case.

 **Fasting Blood Sugar (fbs)**: Most patients have normal fasting blood sugar (mean ~0.15), indicating that high fasting sugar is **not common** in this group.

 **Resting ECG (restecg)**: Mean near 1.0, with values spanning all categories (0–2).

 **Max Heart Rate (thalach)**: Averages around **150 bpm**, ranging from **71 to 202 bpm**.

 **Exercise-Induced Angina (exang)**: With a mean of **0.3278**, most patients **do not experience** angina during exercise.

 **ST Depression (oldpeak)**: Skewed right, with a max value of **6.2** which could be an outlier.

 **Slope**: Most values fall between 1 and 2, suggesting many patients show flat or downsloping ST segments.

 **Result (Target Variable)**: Values from **0 to 4**, which will be later binarized for classification purposes (0 = no disease, 1+ = disease).

## **2.** Dataset Structure Overview

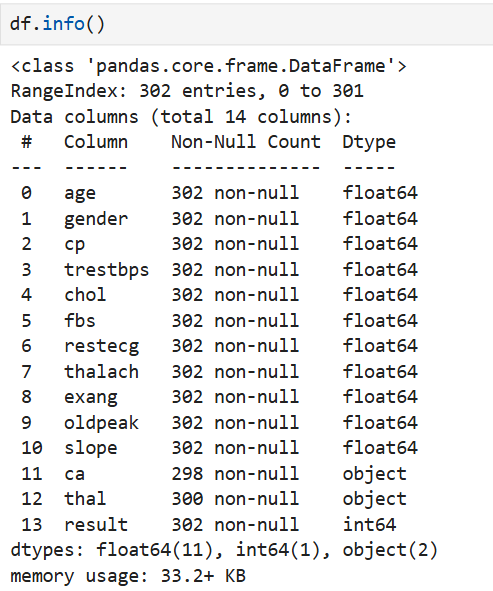
To better understand the structure and completeness of the dataset, we use the info() function. This provides an overview of:

* The number of entries (rows) in the dataset
* The number of non-null (non-missing) values per column
* The data type of each feature (e.g., integer, float, object)
* Memory usage

This step helps identify:

* Which columns may contain missing values
* Which columns are not in the correct data type
* How much data cleaning or type conversion may be needed

This information is critical before performing any transformations or statistical analysis



**Key Observations:**

 All columns except ca and thal have complete (non-null) values.

 The columns ca and thal have **missing values**, with only **298** and **300** non-null entries, respectively.

 Most features are currently of **float64** type, even those that represent categorical variables (e.g., gender, cp, fbs, etc.).

 The columns ca and thal are of **object** type, which indicates they likely contain non-numeric values (possibly "?") which have already replaced with NaN just before starting EDA.

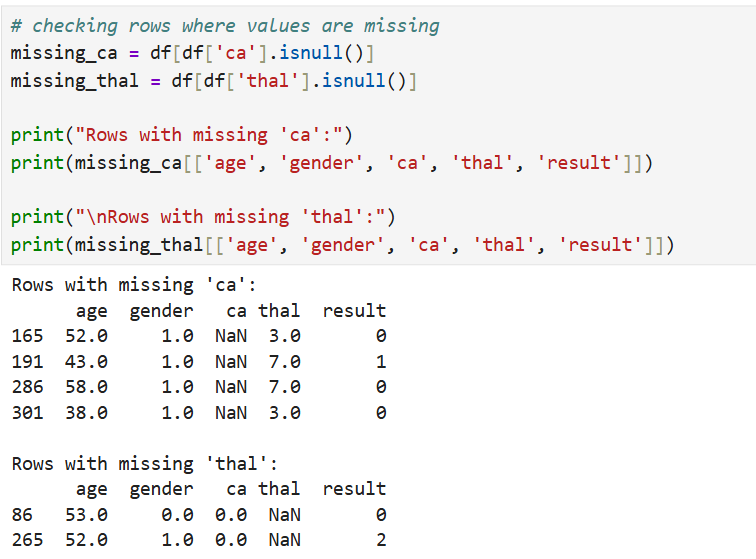
 The result column is correctly recognized as an integer type.

**Inspecting the rows with missing columns:**

After identifying that the ca and thal columns contain missing values, we examine the specific rows where these values are missing. This inspection helps us:

* Understand the nature and distribution of the missing data
* Determine whether the missing values are random or follow any pattern
* Decide on an appropriate imputation strategy

By displaying a subset of columns (age, gender, ca, thal, result) for the affected rows, we can check for potential biases or correlations between missing data and other attributes, such as gender or disease presence.

****

**Key Observation:**

Rows with missing ca:

* 4 rows: Indices 165, 191, 286, and 301
* All patients are **male** (gender = 1)
* Age ranges from **38 to 58**
* Target (result) includes both classes (0 and 1)

Rows with missing thal:

* 2 rows: Indices 86 and 265
* Patients include both **male and female**
* ca values are **0**, suggesting minimal blockage
* Target values: 0 and 2

📝 **Interpretation**

* The missing values appear to be **randomly distributed**.
* There is **no strong pattern** based on age, gender, or disease presence.
* Since there is no obvious bias, we can safely impute these values using the **most frequent value (mode)** for each column.

## **3.** Descriptive Statistics for Categorical Features

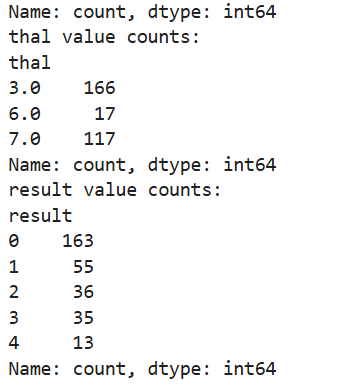
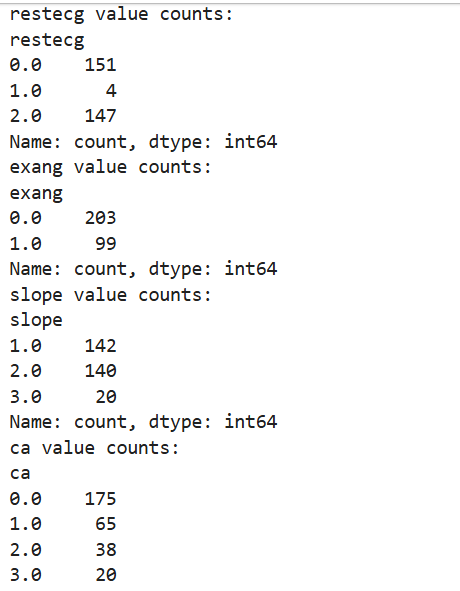
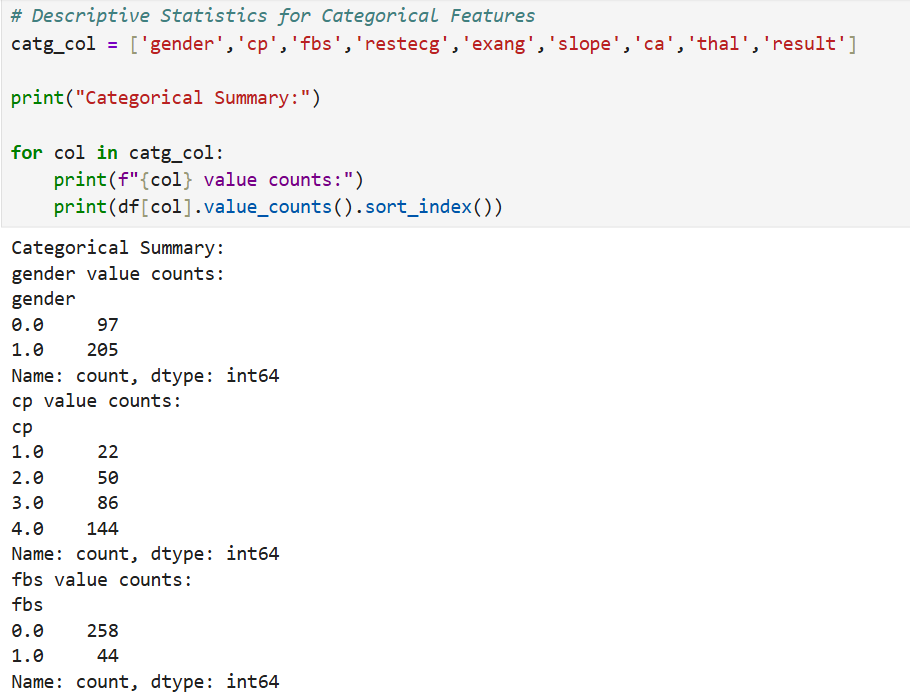
In addition to numerical statistics, it's important to understand the distribution of categorical variables. This helps:

* Identify class imbalances
* Understand dominant categories
* Guide encoding decisions for machine learning models

The following categorical columns were analyzed:

* gender: Sex of the patient
* cp: Chest pain type
* fbs: Fasting blood sugar (> 120 mg/dl)
* restecg: Resting electrocardiographic results
* exang: Exercise-induced angina
* slope: Slope of the ST segment
* ca: Number of major vessels colored by fluoroscopy
* thal: Type of thalassemia
* result: Target variable (presence or absence of heart disease)

We used value\_counts() on each column to display the frequency of each category.

****

**Key** **Observation**:

**Gender**

* Male (1.0): 205 patients
* Female (0.0): 97 patients  
  The dataset is male-dominant with nearly 68% male samples.

**Chest Pain Type (cp)**

* Type 4 (Asymptomatic): 144 cases (most common)
* Type 3: 86 cases
* Type 2: 50 cases
* Type 1 (Typical angina): 22 cases  
  Asymptomatic chest pain is the most frequent category, which is typical in heart disease datasets.

**Fasting Blood Sugar > 120 mg/dl (fbs)**

* 0 (False): 258
* 1 (True): 44  
  Most patients do not have elevated fasting blood sugar. This feature is highly imbalanced.

**Resting ECG (restecg)**

* Normal (0): 151
* ST-T abnormality (2): 147
* Left ventricular hypertrophy (1): Only 4  
  ECG types are well distributed between normal and abnormal, except for class 1, which is rare.

**Exercise-Induced Angina (exang)**

* 0 (No): 203
* 1 (Yes): 99  
  Most patients do not experience angina from exercise.

**Slope of ST Segment (slope)**

* 1 (Flat): 142
* 2 (Downsloping): 140
* 3 (Upsloping): 20  
  Flat and downsloping slopes are common. Upsloping is rare.

**Number of Major Vessels Colored by Fluoroscopy (ca)**

* 0 vessels: 175
* 1 to 3 vessels: 127  
  A large number of patients have no major vessel narrowing.

**Thalassemia (thal)**

* 3 (Normal): 166
* 7 (Reversible defect): 117
* 6 (Fixed defect): 17  
  Normal and reversible defects dominate. Fixed defects are rare.

**Target Variable (result)**

* 0 (No disease): 163
* 1–4 (Disease): 139  
  The dataset is reasonably balanced between healthy and diseased patients but has multiple target classes that will be binarized in the later stage.

## 4. Target Variable Binarization

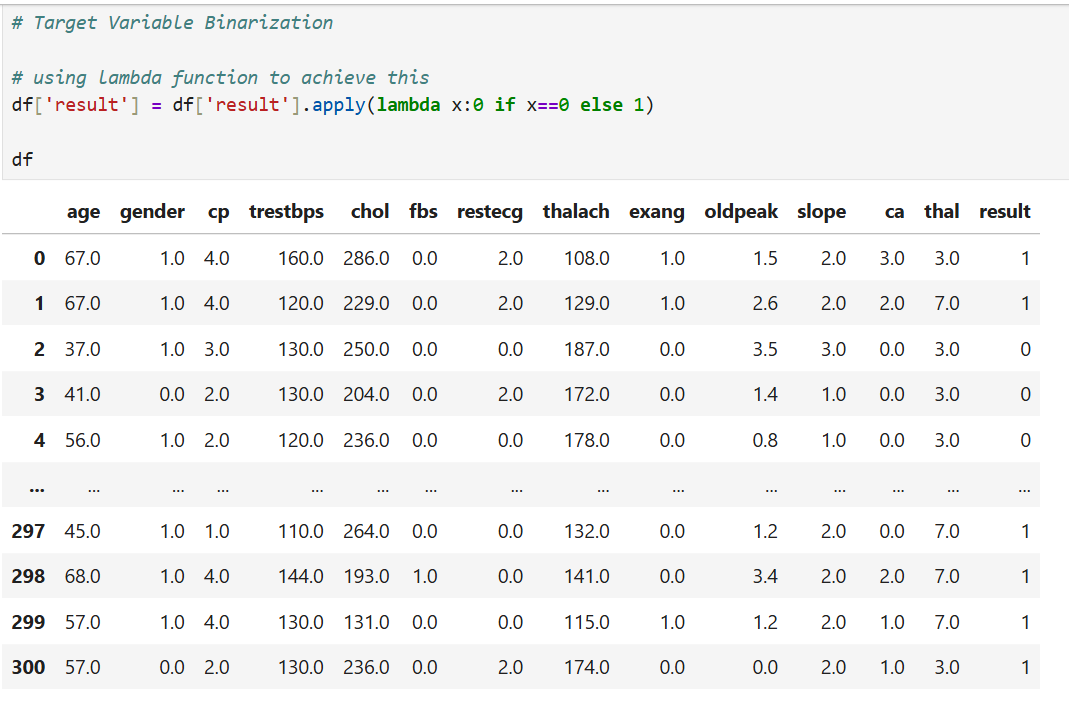
The original result column contains values from **0 to 4**, where:

* 0 indicates **no heart disease**
* Values 1, 2, 3, and 4 indicate **varying degrees of heart disease severity**

However, the objective of this project is to build a **binary classifier** to predict whether heart disease is **present or not**. Therefore, we transform the target variable into two classes:

* 0: No heart disease
* 1: Presence of heart disease (for any value 1 through 4)

This transformation simplifies the problem to a **binary classification task** and allows for more meaningful visualizations when analyzing how features relate to disease presence.

****

## **5.** Visualization

### 5.1 Visual Analysis: Feature Distributions and Relationships

To better understand the dataset and how different features relate to the presence of heart disease, we use a variety of visualizations.

The goals of this step are to:

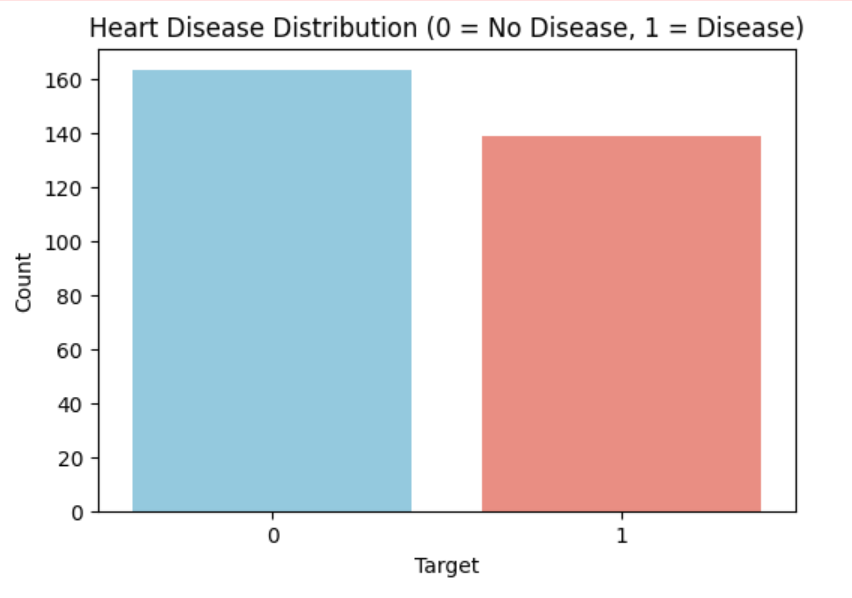
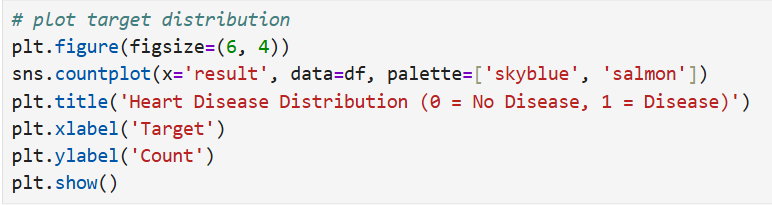
* Explore the **distribution** of the target variable (result)
* Compare how features behave for patients **with vs. without heart disease**
* Identify trends or patterns that could influence model predictions
* Spot potential **outliers or skewed data** visually

We use the following types of plots:

* **Count plots** for categorical features
* **Box plots** and **violin plots** for numeric features
* **Correlation heatmap** to analyze feature relationships

### **5.2** Target Variable Distribution

To understand the balance between the two classes (heart disease present vs. not present), we begin by plotting the distribution of the target variable result.



**Key Observation:**

**plot shows:**

No Disease (0): Around 160 patients

Disease (1): Around 140 patients

This indicates a slightly imbalanced dataset (53% healthy vs. 47% with disease). While not severely imbalanced, we should still address this during modeling to avoid bias.

### **5.3** Comparison of Categorical Features Against Heart Disease Presence

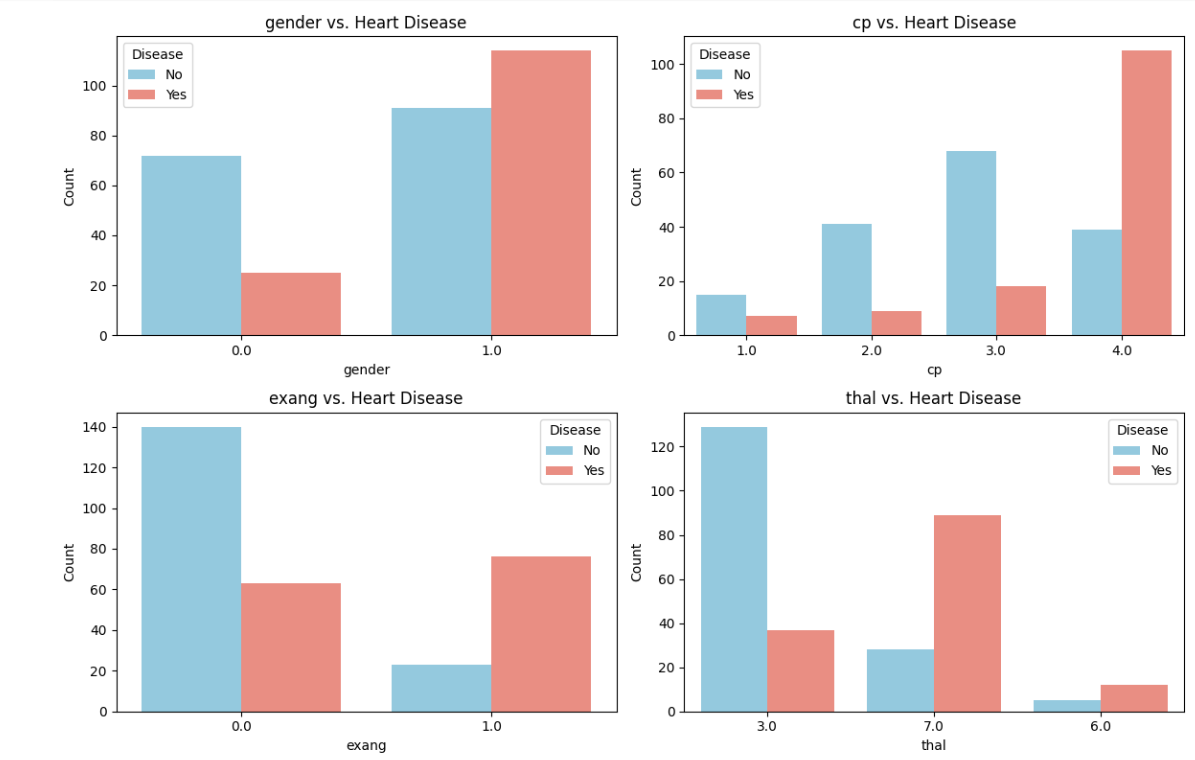
To understand how categorical features relate to the presence or absence of heart disease, we generated count plots with the target variable (result) as the hue. These plots help us visually assess whether certain categories are more associated with heart disease.

The features visualized include:

* gender: Patient sex
* cp: Chest pain type
* exang: Exercise-induced angina
* thal: Thalassemia status

Each bar chart shows how the distribution of classes (0 = No Disease, 1 = Disease) varies across each category.





**Key Observation:**

**1**. **Gender**

* Male patients (gender = 1):
  + The number of males with heart disease is **higher** than the number of males without heart disease.
  + Males with heart disease represents the largest group within the plot.
* Female patients (gender = 0):
  + A significantly higher number of females do not have heart disease compared to those who do.

**Interpretation:**  
In this dataset Males appear to have a prevalence or likelihood of heart disease compared to female.

**2. Exercise-Induced Angina (exang)**

* exang = 1 (Yes):  
  Approximately 75–80% of patients with exercise-induced angina have heart disease.
* exang = 0 (No):  
  Majority are healthy, but 30–40% still have heart disease, indicating possible false negatives.

**Interpretation:**  
exang is a highly predictive feature for heart disease and should be prioritized during model training.

**3. Chest Pain Type (cp)**

* cp = 4 (Asymptomatic):  
  Approximately 70–80% of these patients have heart disease, making this the highest-risk group.
* cp = 1 (Typical Angina):  
  Mostly associated with no disease (only around 20% have heart disease).

**Interpretation:**  
Asymptomatic patients are at higher risk, possibly due to silent ischemia. This makes cp an important feature for early detection of heart disease.

**4. Thalassemia (thal)**

* thal = 7 (Reversible defect):  
  Around 85–90% of these patients have heart disease.
* thal = 3 (Normal):  
  Mostly healthy — 70–75% show no disease.
* thal = 6 (Fixed defect):  
  Rare in the dataset; considering merging with thal = 7 due to similar risk level.

**Interpretation:**  
thal is a strong predictor of heart disease. It should be treated as a key feature in modeling.

Merging high-risk but infrequent categories can help improve model stability.

### **5.4** Boxplots of Numerical Features vs. Heart Disease

To analyze how numerical attributes difference between patients with and without heart disease, we use boxplots for the following features:

* age: Patient’s age
* chol: Serum cholesterol (mg/dl)
* thalach: Maximum heart rate achieved

These plots show the **distribution, central tendency, and spread** of each feature across the two classes (result = 0 and 1), helping us visually detect potential predictors or outliers.

Each boxplot represents:

* The **median** (horizontal line inside the box)
* The **interquartile range (IQR)** (the box itself)
* **Potential outliers** (dots outside the whiskers)
* Class-wise differences in feature distributions

This analysis is useful to identify which numerical features show a clear separation between healthy and diseased patients.

### **5.5** Boxplots of Numerical Features vs. Heart Disease

To analyze how numerical attributes differ between patients with and without heart disease, we use boxplots for the following features:

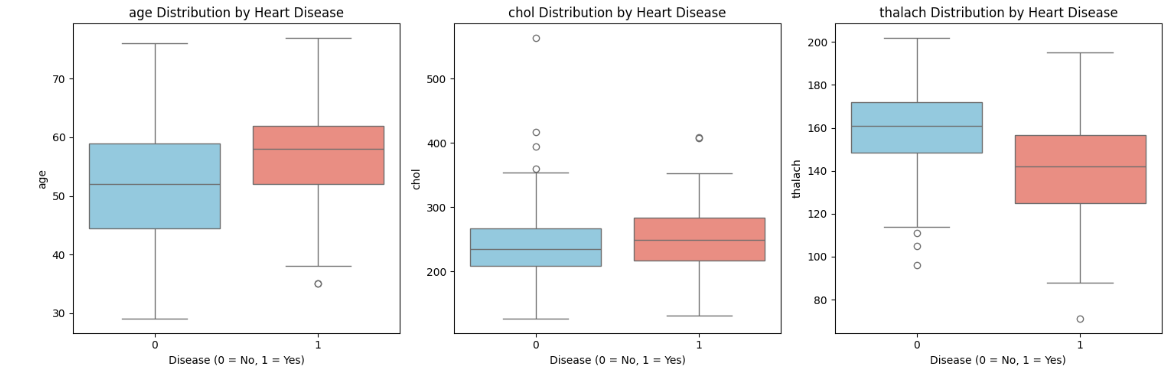
* age: Patient’s age
* chol: Serum cholesterol (mg/dl)
* thalach: Maximum heart rate achieved

These plots show the **distribution, central tendency, and spread** of each feature across the two classes (result = 0 and 1), helping us visually detect potential predictors or outliers.

Each boxplot represents:

* The **median** (horizontal line inside the box)
* The **interquartile range (IQR)** (the box itself)
* **Potential outliers** (dots outside the whiskers)
* Class-wise differences in feature distributions

This analysis is useful to identify which numerical features show a clear separation between healthy and diseased patients.



**Key Observation:**

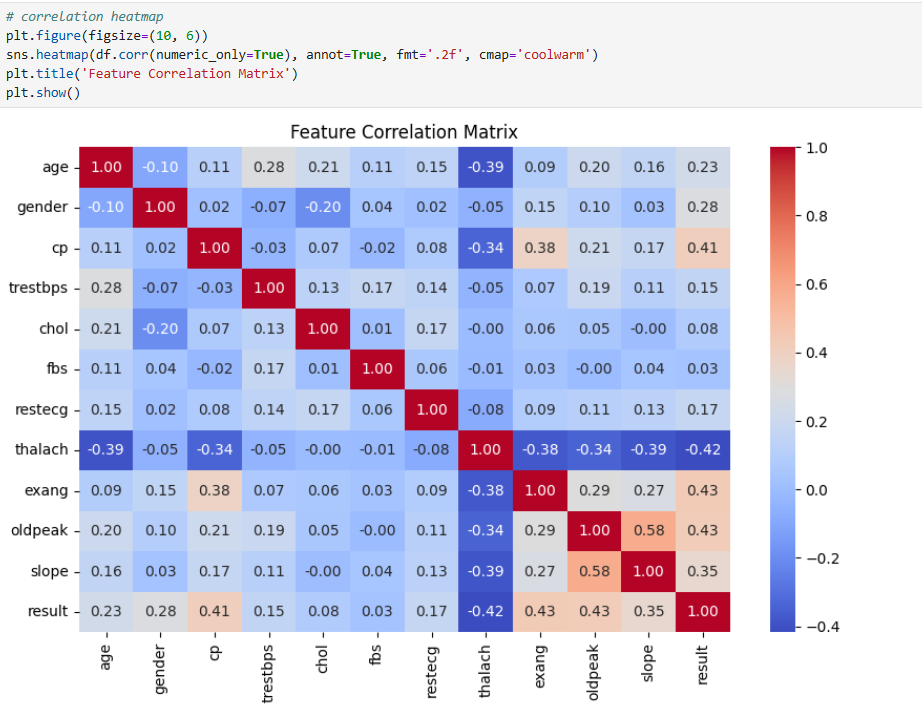
* 1. Age distribution by Hear disease
     + Heart disease is more common in older patients.
     + The median age for patients with heart disease is higher than for those without.
     + Clear shift in distribution: age is a moderately predictive feature.
  2. Chol distribution by Heart disease
     + Patients with and without disease have overlapping cholesterol ranges.
     + Some **outliers with very high cholesterol** exist in both groups.
     + Cholesterol may not be a strong standalone predictor.
  3. thalach distribution by heart disease
     + Patients **without** heart disease tend to have **higher max heart rates**.
     + Strong visual separation: lower thalach is associated with heart disease.
     + Thalach is a strong inverse indicator.

### **5.6** Feature Correlation Heatmap

To understand the relationships between numerical features and the target variable, we plot a **correlation heatmap**. This matrix provides a quick overview of how strongly each pair of variables is linearly related.

* We use **Pearson correlation** coefficients ranging from -1 (strong negative correlation) to 1 (strong positive correlation).
* A value close to 0 indicates little to no linear relationship.
* The heatmap helps identify:
  + Which features are most associated with the presence of heart disease
  + Any potential **multicollinearity** between input features (which could affect certain models)

This step supports better feature selection and interpretation in later stages of model development.



**Key Observation**:

* Most positively correlated features with heart disease**:**
  + cp (0.41)
  + exang (0.43)
  + oldpeak (0.43)
  + slope (0.35)
* Most negatively correlated:
  + thalach (−0.42)

**Insight**:

* Features like cp, exang, and oldpeak are valuable for predicting disease.
* thalach has a strong negative correlation — confirming boxplot findings.
* No extreme multicollinearity between features, so all can likely be used in modeling.

### **5.7** Scatter Plot Analysis

Scatter plots are used to explore the relationship between two **continuous numerical features**, allowing us to visually detect patterns, clusters, correlations, or separations between groups.

In the context of heart disease prediction, these plots help us:

* Examine how two key clinical variables interact (e.g., age vs. heart rate)
* Determine if there’s a **visual distinction** between patients with and without heart disease
* Identify **non-linear relationships** or **overlapping regions** between classes
* Support feature selection and better understanding of **feature-target associations**

By adding a hue based on the target variable (result), we can compare how feature combinations differ across the two classes — those with heart disease and those without.

#### **5.7.1** Scatter Plot: Age vs. Maximum Heart Rate

This scatter plot visualizes the relationship between **age** and **maximum heart rate achieved (thalach)**, with points colored based on the presence of heart disease.

* Each point represents an individual patient.
* The color (hue='result') differentiates between patients with (1) and without (0) heart disease.
* This type of plot helps detect **clusters, trends, or separations** between classes across two continuous variables.

This visualization is useful for assessing whether maximum heart rate varies significantly across age groups and whether it can help in distinguishing between the two target classes.



**Key Observation:**

 Patients without disease cluster around **higher thalach** across age groups.

 Patients with disease tend to have **lower max heart rate**, especially in middle and older age ranges.

 A downward trend is visible — **thalach decreases with age**, and disease overlaps with lower values.

#### 5.7.2 Scatter Plot: Cholesterol vs. ST Depression by Heart Disease

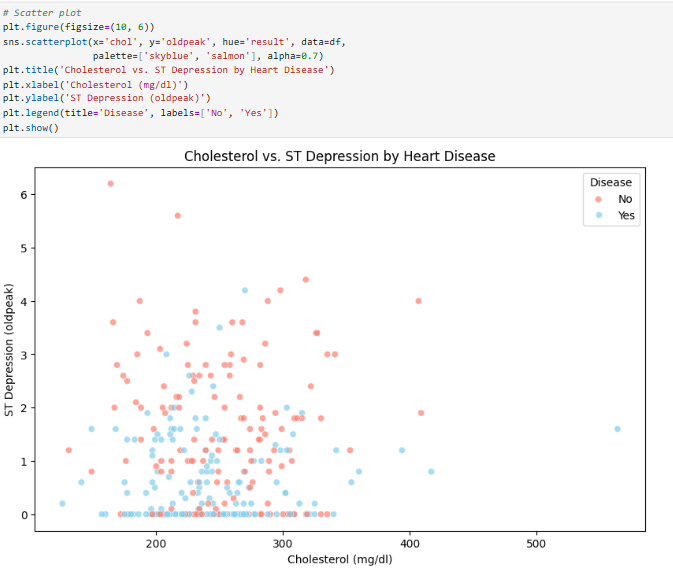
This scatter plot examines the relationship between **serum cholesterol (chol)** and **ST depression (oldpeak)**, with points colored by heart disease status (result).

* **X-axis:** Cholesterol level (mg/dl)
* **Y-axis:** ST depression induced by exercise relative to rest (oldpeak)
* **Hue (Color):** Differentiates patients with (1) and without (0) heart disease

Each point represents one patient in the dataset. This visualization helps to:

* Observe how **cholesterol and ST depression interact**
* Identify any clustering patterns that separate the two classes
* Assess whether these features might jointly contribute to predicting heart disease

By plotting both features together and using color coding, we can better evaluate their potential predictive power in combination.



**Key Observation:**

 No strong linear pattern, but:

* Patients without disease are **clustered near low oldpeak** and moderate cholesterol.
* Patients with disease are spread across **higher oldpeak** and **wider cholesterol range**.

 Suggests that **oldpeak may be more predictive than cholesterol**.

#### 5.7.3 Scatter Plot: Age vs. Number of Blocked Vessels by Heart Disease

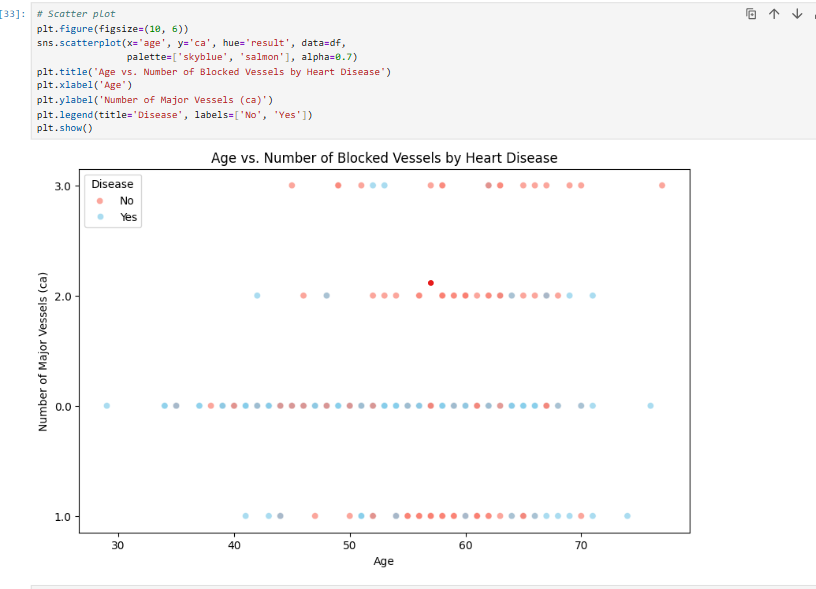
This scatter plot illustrates the relationship between a patient's **age** and the **number of major blood vessels blocked (ca)**, with color indicating heart disease status (result).

* **X-axis:** Age
* **Y-axis:** Number of major vessels blocked (ca)
* **Hue (Color):** Represents heart disease presence (0 = No, 1 = Yes)

Each point represents a patient. This visualization helps to:

* Analyze whether **age** and **vessel blockage severity** are jointly associated with heart disease
* Detect trends, such as whether vessel blockage increases with age or disease presence
* Understand how blockage levels vary across different age groups and health statuses

By combining a clinical factor (age) with a direct anatomical indicator (ca), this plot provides insight into physical deterioration patterns linked to heart disease.



**Key Observation**:

 Most patients with **0 vessels blocked (ca = 0)** are healthy, especially under age 60.

 Those with heart disease are more frequently found at **higher ca values (1 to 3)**.

 Increasing age is associated with an increase in vessel blockage in both classes, but more pronounced for diseased patients.

# **Task 2:** **Data Preprocessing and Feature Engineering**

Before feeding the dataset into machine learning or deep learning models, it is essential to preprocess the data to ensure quality, consistency, and compatibility with training algorithms.

**Objectives of this task:**

* Handle missing values appropriately
* Ensure correct data types across features
* Encode categorical variables if required
* Normalize or scale features if needed
* Prepare the final dataset for modeling

This step ensures that the dataset is **clean, well-structured**, and in a **machine-readable format** so that models can learn patterns effectively without being misled by noise, inconsistencies, or incompatible types.

## **1.** Data Type Conversion

After analyzing the dataset using .info(), it was observed that several features had incorrect or inconsistent data types:

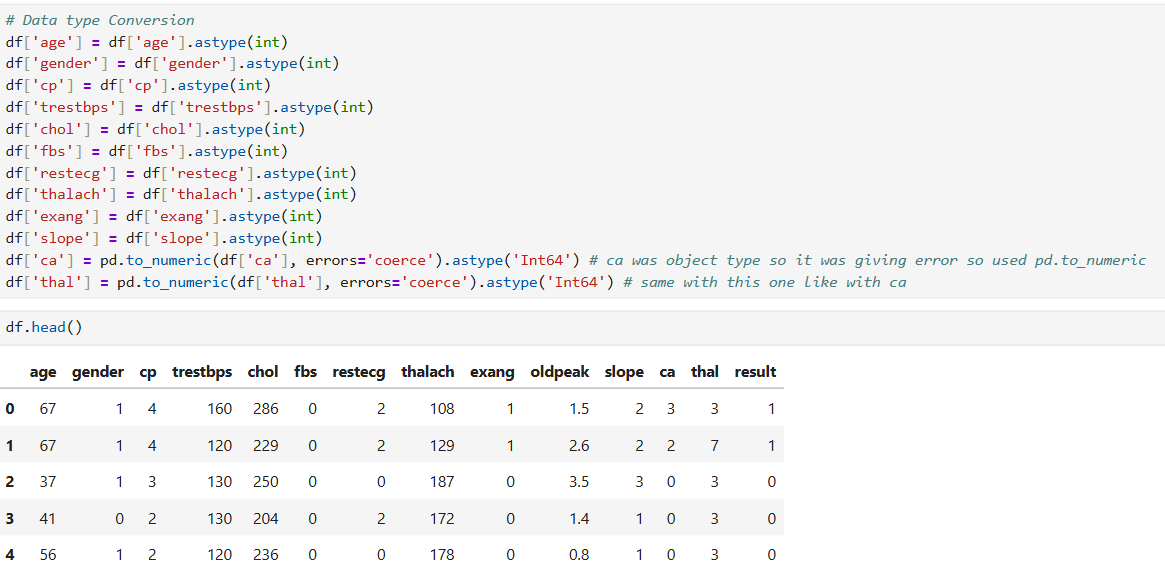
* Features like gender, cp, fbs, and others were stored as float64, even though they represent **categorical or integer-based attributes**.
* The columns ca and thal were of **object** type due to non-numeric entries (like "?"), which had been replaced with NaN earlier.

To correct this:

* Integer-based features were explicitly cast to int type.
* The ca and thal columns were converted to numeric using pd.to\_numeric() with errors='coerce' to ensure invalid values (e.g., strings) are handled as NaN.
* These columns were then cast to the Int64 pandas extension type to allow compatibility with missing values.

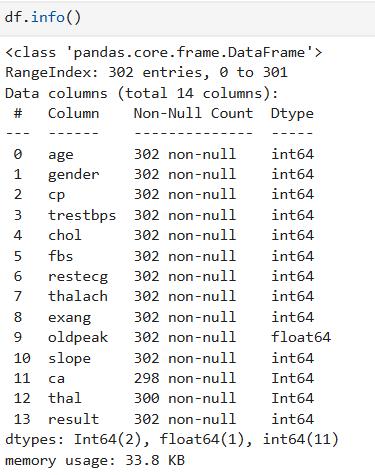
**Why this is important:**

* Ensures that **numerical and categorical features** are in the correct format for modeling.
* Prevents **type-related errors** during encoding, scaling, or model training.
* Supports **better memory usage** and computational efficiency.



## **2.** Handle Missing Values

**Current Dataset Info:**

****

### **2.1** Current Missing Value Status

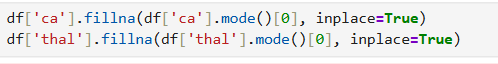
* Most features: no missing values (302 non-null)
* ca: **298 non-null** - - > **4 missing**
* thal: **300 non-null** - - > **2 missing**

### **2.2** Mode Imputation for Missing Values

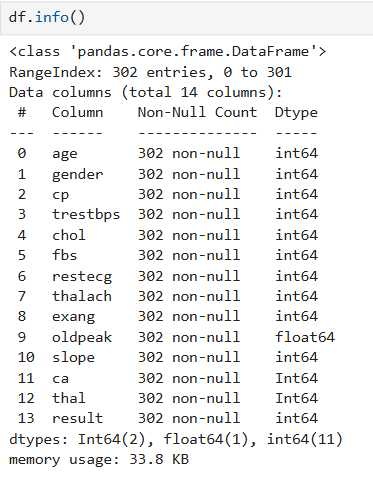
To handle the missing values in the dataset, we applied **mode imputation** to the following columns:

* ca (number of major vessels)
* thal (thalassemia type)

Since both of these features are **categorical in nature**, the most frequent value (mode) is a suitable choice for imputation. This preserves the distribution of the data and avoids introducing bias or removing valuable records.



#### 2.2.1 After Mode imputation



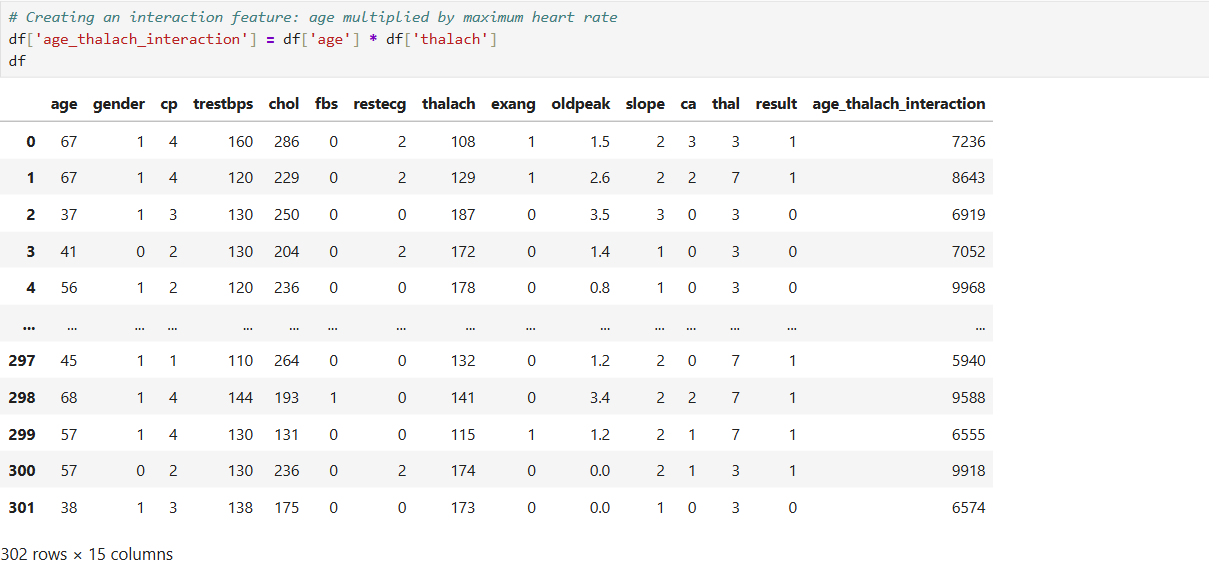
Key Observation:

* + No Missing values

## **3.** Feature Engineering

### **3.1** Feature Engineering: Interaction Term (age \* thalach)

To enhance the predictive power of the dataset, we introduced a new **interaction feature** by multiplying **age** and **maximum heart rate (thalach)**:



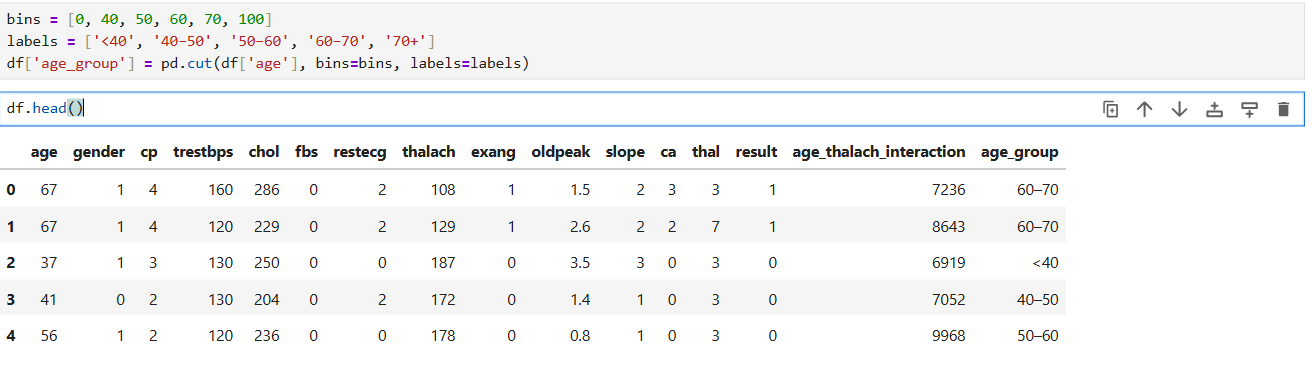
**Reason:**

* Both age and thalach are clinically relevant features in cardiovascular health.
* **Maximum heart rate typically declines with age**, but the rate of decline and the patient's ability to maintain higher heart rates can indicate heart health.
* This interaction term may help the model detect risk patterns that are **not linearly captured** by the individual features alone.
* It also helps certain models (e.g., logistic regression, SVM, neural networks) recognize **combined effects** between variables.

By creating this engineered feature, we aim to improve the model’s ability to detect subtle but important signals related to heart disease risk.

### **3.2** Feature Engineering: Binning Age into Categorical Groups (Optional)

As part of exploratory feature engineering, we created a new categorical feature called age\_group by binning the continuous age variable into defined age ranges.



**Purpose**:

* Converts the continuous age feature into **human-readable brackets**
* Helps explore trends in heart disease across age categories
* Useful for **tree-based models** or **interpretability-focused visualizations**
* May support stratification or subgroup analysis in reporting

Note:

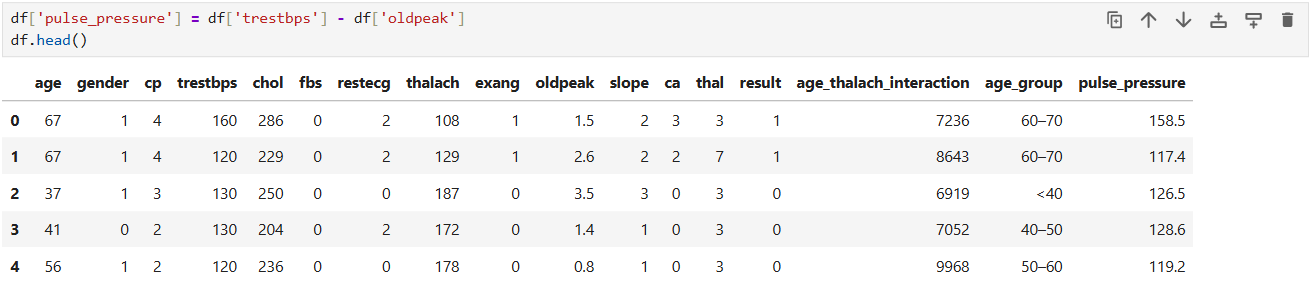
This feature is **optional** and is not required for all models. It has been added for potential use in:

* Visual analysis by age group
* Tree-based classifiers (e.g., decision trees, random forests)
* Feature importance or grouped metrics reporting

We retain the original age column for modeling workflows that require a continuous numerical feature.

### **3.3** Feature Engineering: Pulse Pressure (trestbps - oldpeak)

As part of feature engineering, we introduced a new numerical feature called pulse\_pressure, calculated as:



**Purpose**:

* This feature represents an estimate of **resting pulse pressure**, derived by subtracting ST depression (oldpeak) from resting blood pressure (trestbps).
* Clinically, **pulse pressure** is an indicator of cardiovascular health and arterial stiffness. Abnormally low or high values can signal underlying heart conditions.
* Including this derived metric may help the model better capture risk associated with blood pressure behavior and ischemic response.

## **4.** One-Hot Encoding for Categorical Features

To prepare the dataset for machine learning models, we encode categorical variables using **one-hot encoding**. This process converts each category level into a separate binary column (0 or 1), allowing algorithms to process them numerically.

Columns **Encoded**:

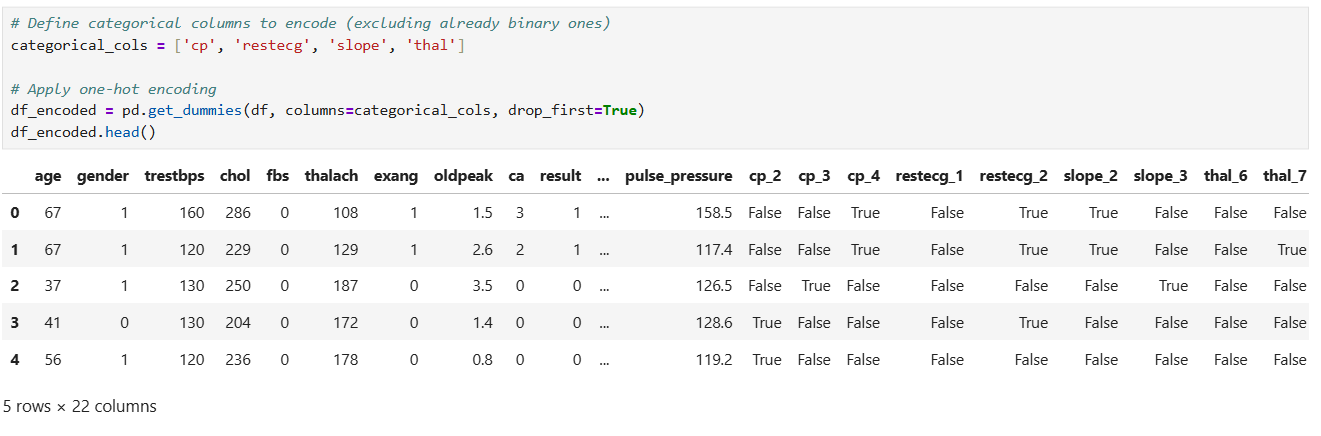
* cp: Chest pain type
* restecg: Resting ECG results
* slope: Slope of the ST segment
* thal: Thalassemia type

These were chosen because:

* They are **categorical** with more than two unique values
* Their numerical representation does **not** reflect meaningful order

We use pd.get\_dummies() with drop\_first=True to:

* Avoid the **dummy variable trap**
* Reduce multicollinearity



## **5.** Train-Test Split and Numerical Feature Scaling

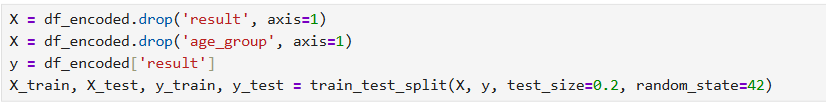
To prepare the dataset for model training, we split it into training and test sets and apply feature scaling to numerical variables.

#### **5.1 Train-Test Split**

We divide the dataset into:

* **80% training data** for model learning
* **20% testing data** for performance evaluation

We use random\_state=42 to ensure reproducibility of the split and also dropped Age\_group column.



#### 5.2 **Numerical Scaling (Standardization)**

We apply **Z-score scaling** using StandardScaler from scikit-learn, which standardizes features by removing the mean and scaling to unit variance.

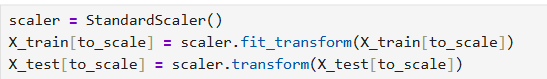
This is essential for models that are sensitive to feature magnitude (e.g., logistic regression, neural networks).

We scale only the **numerical features**:



To prevent **data leakage**, we:

* Fit the scaler on the training set
* Use the same transform on the test set



# **Task 3: Model Training and Development**

In the field of healthcare, particularly in the early detection of life-threatening conditions like heart disease, predictive modeling using machine learning (ML) and deep learning (DL) plays a vital role. Accurate prediction models can assist clinicians in identifying high-risk patients, enabling early intervention and improved patient outcomes.

**Objective**:

To develop and train a set of machine learning and deep learning models capable of accurately classifying whether a patient is at risk of heart disease based on a set of clinical attributes.

## **3.1** Logistic Regression – Model Training and Hyperparameter Tuning

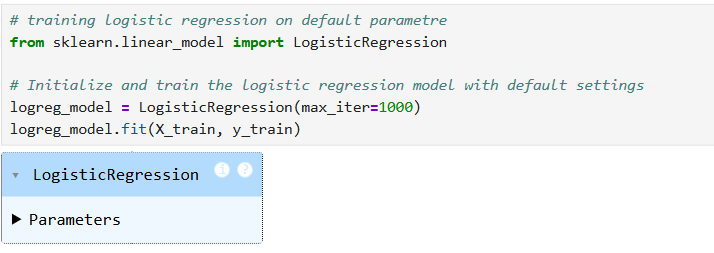
We begin model development with **Logistic Regression**, a widely used linear classifier suitable for binary classification problems like heart disease prediction.

**Why Logistic Regression?**

* Simple, fast, and interpretable
* Estimates probabilities and classifies based on a sigmoid decision boundary
* Suitable for structured, tabular data with binary outcomes

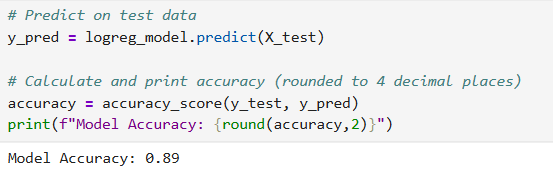
**Model Setup:**

We trained a logistic regression model on the preprocessed X\_train and y\_train datasets using default hyperparameters. We also increased the max\_iter to 1000 to ensure convergence on our scaled data.



**Logistic Regression – Training Check with Accuracy**

After training the Logistic Regression model, we performed a quick prediction on the test set to validate the model's basic performance before hyperparameter tuning.



This initial accuracy (≈ 89%) indicates that the model is learning meaningful patterns from the data.

**Hyper Tuning**

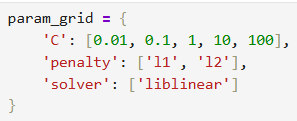
defining the base logistic regression model with:

* max\_iter=1000: to ensure convergence during training
* random\_state=42: for reproducibility

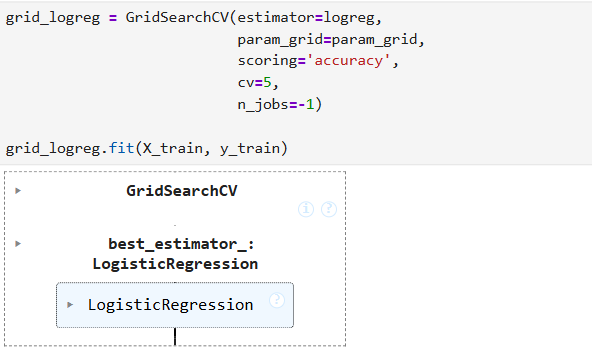
****

define the combinations of hyperparameters to test:

* C: Controls regularization strength. Lower = stronger penalty on coefficients.
* penalty: Type of regularization. 'l1' promotes sparsity, 'l2' keeps all features.
* solver: 'liblinear' is compatible with both penalties and suitable for small datasets

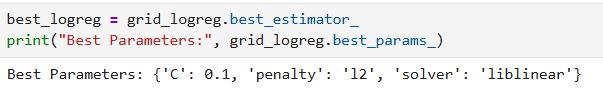
****

* We use **GridSearchCV** to search all combinations in the param\_grid
* cv=5: 5-fold cross-validation to ensure reliable performance estimation
* scoring='accuracy': Optimization criterion
* n\_jobs=-1: Uses all CPU cores for faster processing

****

 best\_estimator\_: retrieves the model with the best cross-validation score

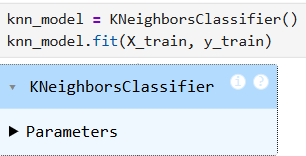
 best\_params\_: shows which combination of C, penalty, and solver was selected



## **3.2** K-Nearest Neighbors (KNN) – Model Training and Hyperparameter Tuning

The K-Nearest Neighbors (KNN) algorithm is a simple, non-parametric, lazy learning algorithm used for both classification and regression tasks. In classification, it assigns a class based on the majority class of its 'k' nearest neighbors in the feature space.

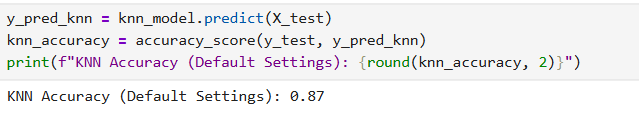
To begin the training process for the **K-Nearest Neighbors (KNN)** algorithm, we initialize the model using its default settings and fit it to the training dataset.



**Description**:

* **Model:** KNeighborsClassifier() from sklearn.neighbors
* **Default behavior:** Uses **5 nearest neighbors** for classification.
* **Purpose:** The model memorizes the training data. During prediction, it classifies new points based on the **majority class of their 5 closest neighbors** (Euclidean distance by default).

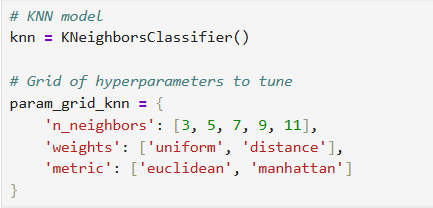
After training the K-Nearest Neighbors model, we assess its performance by making predictions on the unseen test set and calculating the overall accuracy.



We get accuracy near to 0.87 which quit good in my opinion for default.

**Hyper Tuning**

To improve the performance of the KNN model, we define a grid of hyperparameters that will be explored during tuning. These parameters control how the model determines "nearest neighbors" and how it weighs them during prediction.



**Parameter Descriptions for KNN Hyperparameter Tuning:**

* **n\_neighbors**: Controls model flexibility. Lower values make the model more sensitive to noise, while higher values lead to more stable predictions.
* **weights**: 'uniform' treats all neighbors equally, whereas 'distance' assigns more importance to closer neighbors.
* **metric**: Defines the distance measure used to determine the nearest neighbors. 'euclidean' uses straight-line distance, while 'manhattan' uses grid-based distance.

These combinations will be passed into GridSearchCV to identify the most effective configuration based on cross-validation performance.

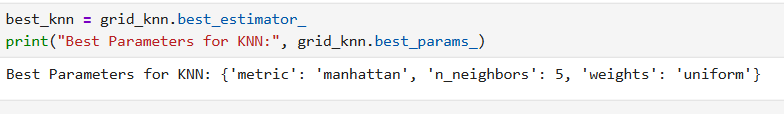
**Hyperparameter Tuning using GridSearchCV (KNN)**

To systematically identify the best-performing hyperparameter configuration for the K-Nearest Neighbors model, we apply **GridSearchCV** — a brute-force method to search through a predefined parameter grid using cross-validation.



Selecting the Best KNN Model **from** Grid Search

After training and evaluating multiple combinations of hyperparameters using GridSearchCV, we extract the **best-performing KNN model** based on cross-validation accuracy.



The most effective configuration for the KNN classifier was:

* **metric = 'manhattan'**: Grid-based distance calculation performed better than Euclidean.
* **n\_neighbors = 5**: The default number of neighbors yielded optimal results during tuning.
* **weights = 'uniform'**: Equal weighting of neighbors proved more effective than distance-based weighting.

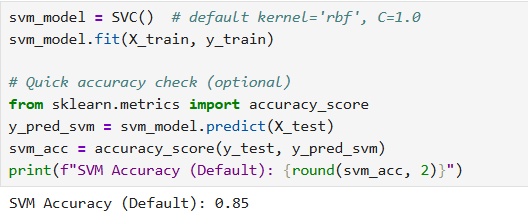
## **3.3 Support Vector Machine (SVM) – Model Training and Hyperparameter Tuning**

The Support Vector Machine (SVM) is a powerful supervised machine learning algorithm used primarily for classification tasks. It works by finding the optimal hyperplane that best separates the classes in the feature space with the maximum margin.

To begin the training process for the SVM algorithm, we initialize the model using its default settings and fit it to the training dataset.

**Description:**

* **Model:** SVC() from sklearn.svm
* **Default behavior:** Uses the radial basis function (RBF) kernel with C=1.0 and gamma='scale'.
* **Purpose:** The model tries to maximize the margin between classes and is effective in high-dimensional spaces and when the number of dimensions exceeds the number of samples.



0.85 accuracy by default SVM model.

**Hyper Tuning**

To optimize the performance of the SVM model, we perform hyperparameter tuning using GridSearchCV. This approach tests various combinations of key parameters to identify the configuration that yields the highest cross-validation accuracy.

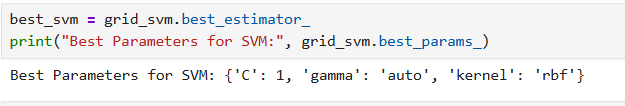


* **C**: Controls the trade-off between margin width and classification error. Smaller values allow for more margin violations (softer margin), larger values enforce stricter margins.
* **kernel**: Determines the method used to separate data (linear vs non-linear).
* **gamma**: Controls the influence of each data point in RBF kernel. 'scale' and 'auto' are automatic options based on data.

The grid search evaluates all parameter combinations using **5-fold cross-validation** and selects the configuration that delivers the highest average accuracy across all folds.

**Selecting the Best SVM Model from Grid Search**

After evaluating all combinations of hyperparameters using GridSearchCV, we extract the best-performing Support Vector Machine (SVM) model based on cross-validation accuracy.

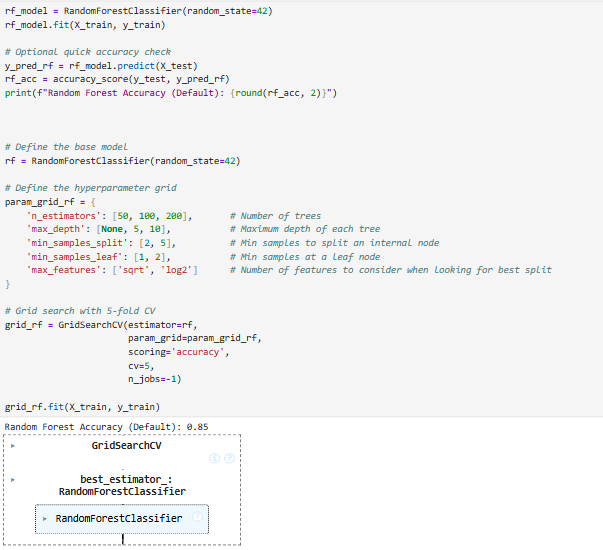


**The optimal configuration for the SVM model was:**

* **C = 1**: Balanced regularization that avoids both overfitting and underfitting.
* **kernel = 'rbf'**: Radial Basis Function kernel enabled non-linear classification.
* **gamma = 'auto'**: Gamma was set automatically based on the number of features.

## **3.4 Random Forest – Model Training and Hyperparameter Tuning**

The Random Forest algorithm is an ensemble learning method that builds multiple decision trees and combines their outputs to produce more accurate and stable predictions. It is well-suited for handling high-dimensional datasets and capturing complex feature interactions.



Accuracy is 0.85

 **n\_estimators**: Total number of decision trees in the forest.

 **max\_depth**: Limits how deep each individual tree can grow.

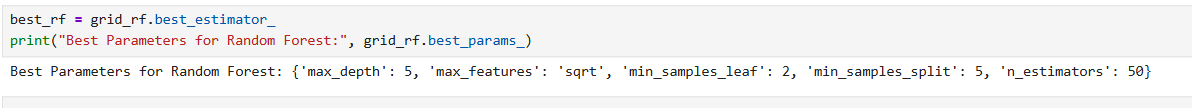
 **min\_samples\_split**: Controls the minimum number of samples required to split an internal node.

 **min\_samples\_leaf**: Sets the minimum number of samples required to be at a leaf node.

 **max\_features**: Determines how many features to consider when looking for the best split at each node

**Selecting the Best Random Forest Model from Grid Search**

After evaluating multiple combinations of hyperparameters using GridSearchCV, we extract the best-performing **Random Forest** model based on cross-validation accuracy.

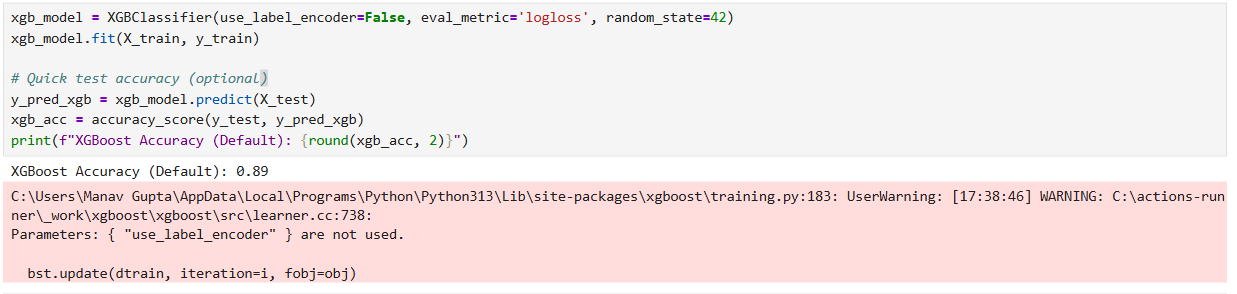


The grid search identified the following optimal configuration for the Random Forest classifier:

* **n\_estimators = 50**: A relatively small forest that balances accuracy and training time.
* **max\_depth = 5**: Shallow trees help reduce overfitting and improve generalization.
* **min\_samples\_split = 5**: Requires more samples to split a node, making splits more meaningful.
* **min\_samples\_leaf = 2**: Prevents over-fragmentation of the trees.
* **max\_features = 'sqrt'**: Uses the square root of total features at each split, a common default that balances diversity among trees.

## **3.5** XGBoost **- Model Training and Hyperparameter Tuning**

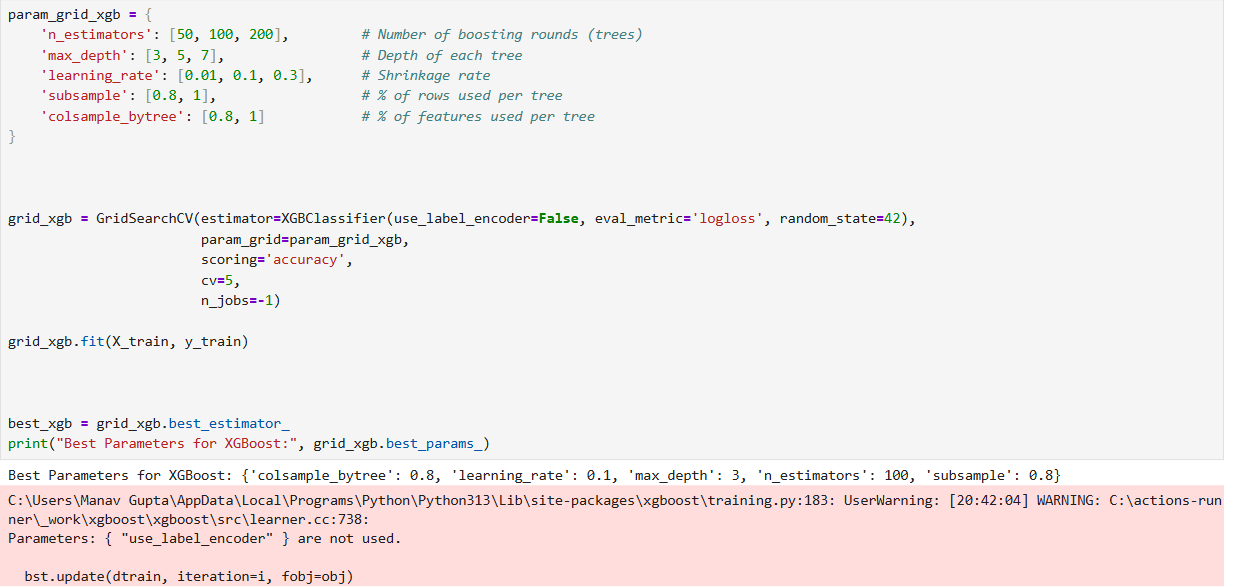
**XGBoost (Extreme Gradient Boosting)** is a powerful, scalable, and efficient gradient boosting framework that performs exceptionally well on structured/tabular data. It builds an ensemble of decision trees sequentially, where each new tree corrects the errors of the previous ones.



Accuracy is 0.89.

**XGBoost – Hyperparameter Tuning with GridSearchCV**

To further enhance the performance of the XGBoost model, we apply **GridSearchCV** to search for the optimal combination of key hyperparameters. This helps fine-tune how each decision tree is built and how the boosting process behaves.

****

The optimal configuration selected for the XGBoost model includes:

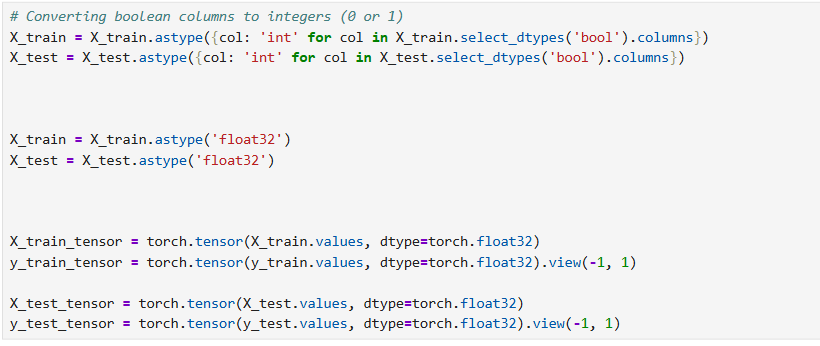
* **colsample\_bytree = 0.8**: Uses 80% of features for building each tree, improving generalization.
* **learning\_rate = 0.1**: A conservative learning rate that balances learning speed with performance.
* **max\_depth = 3**: Shallow trees help prevent overfitting while still capturing important patterns.
* **n\_estimators = 100**: Uses 100 boosting rounds to incrementally build a strong ensemble.
* **subsample = 0.8**: Each tree is trained on 80% of the training data, which adds regularization.

## **3.6** Deep Learning MLP Pytorch Model

A basic feedforward neural network (Multilayer Perceptron) was implemented using PyTorch to classify heart disease. The model consists of an input layer, one or more hidden layers with ReLU activation, and a sigmoid output layer for binary classification. It was trained using the **Adam optimizer** and **Binary Cross-Entropy Loss**. This deep learning approach complements traditional models by learning complex, non-linear patterns in the data.

Preparing Data for PyTorch MLP Model:

Before training a neural network in PyTorch, the data must be properly formatted as floating-point tensors. The following preprocessing steps ensure compatibility and consistency across training and testing inputs:



 **Boolean to Integer Conversion**: PyTorch does not support boolean inputs for training; thus, all boolean columns are cast to integers.

 **Float32 Conversion**: Neural networks in PyTorch typically operate on 32-bit floating-point values for optimal performance.

 **Tensor Conversion**: The data is converted from NumPy arrays to PyTorch tensors, with targets reshaped to column vectors (view(-1, 1)) for binary classification.

### **3.6.1** PyTorch – Basic MLP Architecture and Training

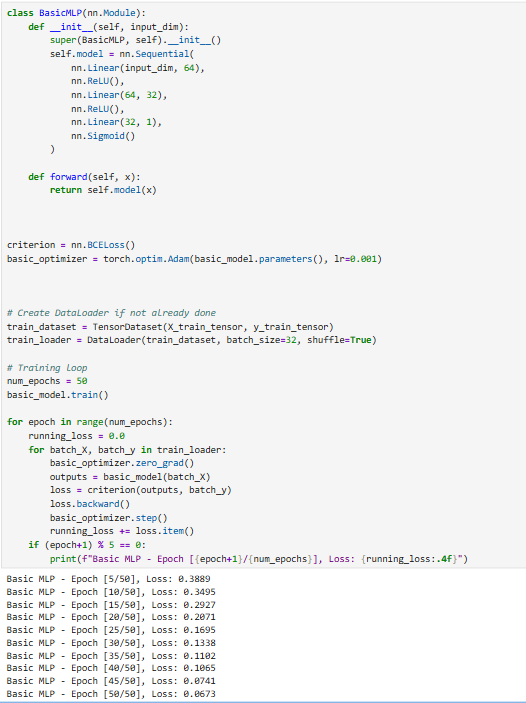
A simple feedforward neural network (Multilayer Perceptron) was implemented in PyTorch to perform binary classification for heart disease prediction. The model was trained using backpropagation and evaluated based on loss reduction across epochs.

**Architecture Description:**

* **Input Layer**: Takes in the number of features from the dataset.
* **Hidden Layers**: Two fully connected layers with 64 and 32 neurons, respectively, both using ReLU activation.
* **Output Layer**: Single neuron with sigmoid activation for binary classification.

**Key Details**:

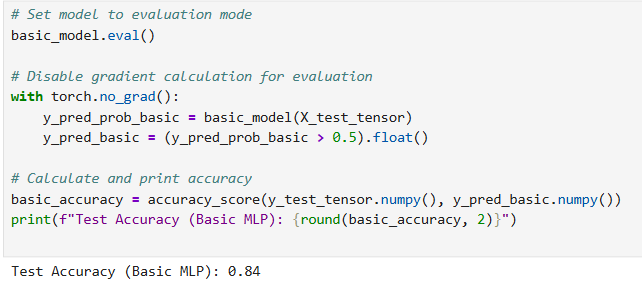
* **Optimizer**: Adam, with learning rate of 0.001 for efficient gradient updates.
* **Loss Function**: Binary Cross Entropy (nn.BCELoss()), suitable for binary classification.
* **Batch Size**: 32, chosen for balanced memory efficiency and convergence speed.
* **Epochs**: 50 total iterations over the training data.



The training loss decreased consistently, indicating successful model convergence.

**Evaluating the Basic MLP Model on Test Data**

After training the basic PyTorch MLP model, we evaluate its performance on unseen test data by computing its accuracy. This involves disabling gradient tracking, generating predictions, and comparing them to true labels.



Accuracy is 0.84.

### **3.6.2** PyTorch – Deep MLP Architecture and Training

To explore the impact of deeper architectures, a **deep Multilayer Perceptron (MLP)** model was implemented using PyTorch. This model increases depth and capacity compared to the basic MLP and is designed to capture more complex patterns in the data.

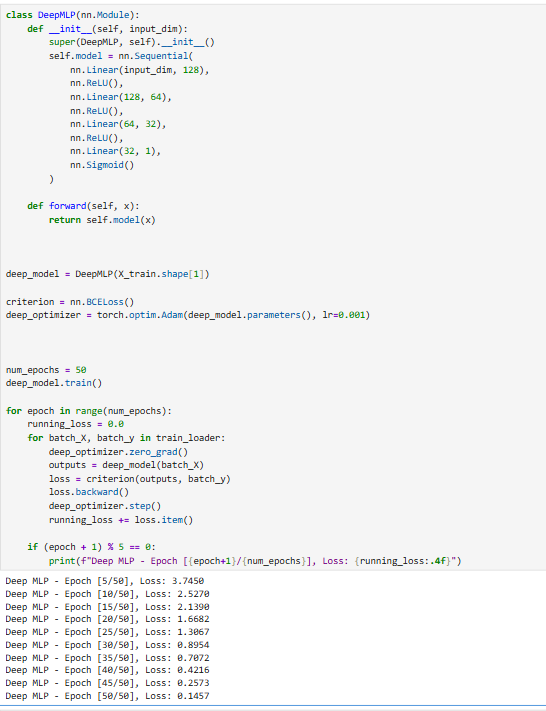
**Architecture Overview**:

* **Input Layer**: Matches number of features in the dataset.
* **Hidden Layers**: Three dense layers with 128, 64, and 32 neurons respectively, each using ReLU activation.
* **Output Layer**: Single neuron with sigmoid activation for binary classification.

This deeper architecture increases the model's capacity to learn non-linear relationships in the data.

* **Loss Function**: Binary Cross Entropy, suited for binary classification.
* **Optimizer**: Adam, chosen for its adaptive learning rate and efficiency.

The model was trained over **50 epochs** using mini-batch gradient descent (batch size = 32). Loss decreased progressively, indicating successful learning.

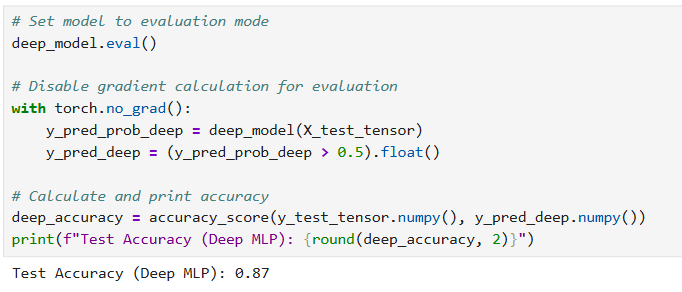
****

 The training loss decreased significantly over the course of training, from **3.74** at epoch 5 to just **0.14** by epoch 50.

 This indicates effective learning and good convergence of the deep model.

Evaluating the Deep MLP Model on Test Data

Once the **Deep MLP** model was trained, it was evaluated on the test set to assess its generalization performance. The following steps were followed:

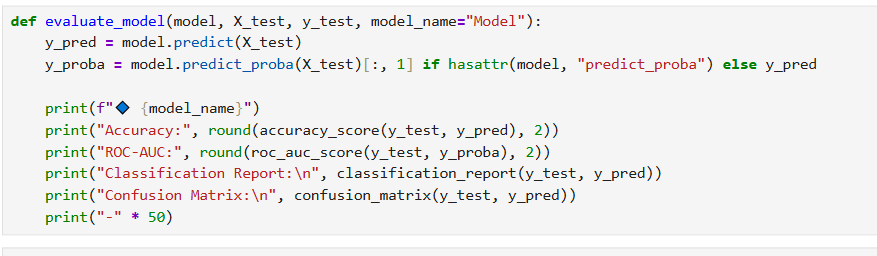


Accuracy is 0.87.

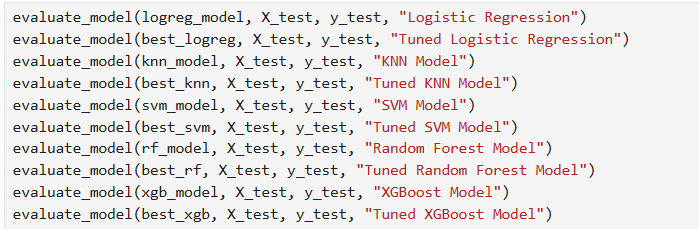
# **Task 4: Model Evaluation and Comparison**

**Model Evaluation Utility Function**

To streamline the evaluation of all machine learning models, a reusable function named evaluate\_model was defined. This function prints multiple performance metrics for a given model using the test set.

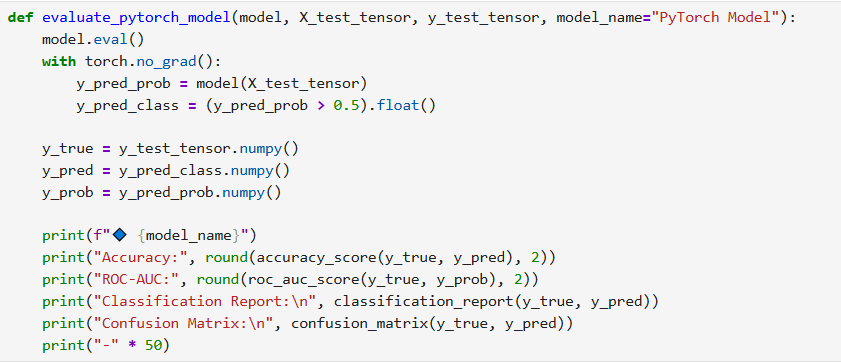


BY Using this function for traditional model like below:



And for Deep learning Model:

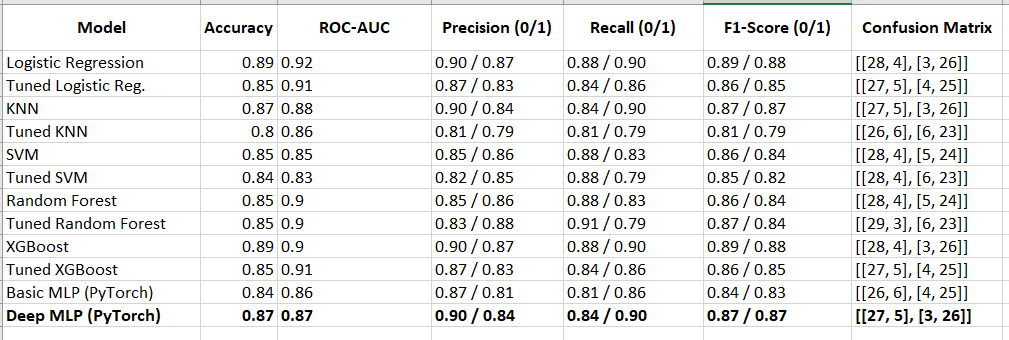
To evaluate PyTorch-based neural networks in a consistent manner with traditional machine learning models, a custom evaluation function named evaluate\_pytorch\_model was defined. This function computes several key performance metrics on the test dataset.



By using this function:



Created a summary table:



## **4.1** Selecting Best Traditional and Best Deep learning Model:

**Best Traditional Machine Learning Model: Logistic Regression**

* **Accuracy**: 0.89
* **ROC-AUC**: 0.92 (highest among all models)
* **Precision (0/1)**: 0.90 / 0.87
* **Recall (0/1)**: 0.88 / 0.90
* **F1-Score (0/1)**: 0.89 / 0.88
* **Confusion Matrix**: [[28, 4], [3, 26]]

Reason:  
Although XGBoost has the same accuracy and slightly lower ROC-AUC, Logistic Regression achieves consistently high precision, recall, and F1 across both classes **with a simpler, more interpretable model** — making it preferable in healthcare applications where interpretability matters.

**Best Deep Learning Model: Deep MLP (PyTorch)**

* **Accuracy**: 0.87
* **ROC-AUC**: 0.87
* **Precision (0/1)**: 0.90 / 0.84
* **Recall (0/1)**: 0.84 / 0.90
* **F1-Score (0/1)**: 0.87 / 0.87
* **Confusion Matrix**: [[27, 5], [3, 26]]

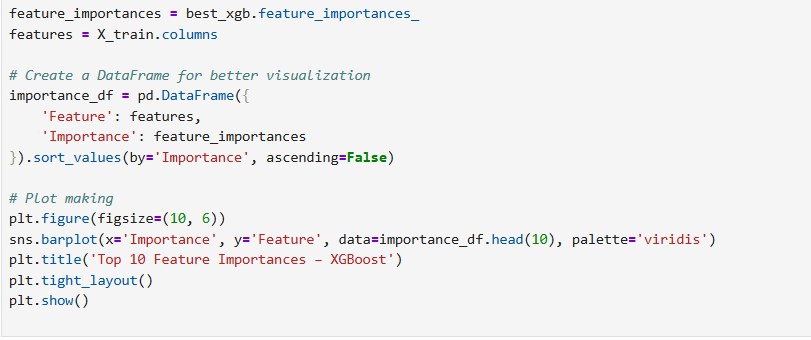
Reason:  
Compared to the Basic MLP (which had 0.84 accuracy), the **Deep MLP** showed **better balance between precision and recall**, and stronger performance on the minority class, making it a clear winner among the PyTorch models.

## **4.2** feature importance

For the best-performing traditional model and the deep learning model, analyze the feature importance to understand which patient attributes are most influential in predicting heart disease.

### **4.2.1 Logistic Regression model feature importance**

To understand which features most strongly influence predictions made by the **XGBoost** model, we extract and visualize its internal feature importances. This provides valuable insight into which patient attributes play a dominant role in predicting heart disease.

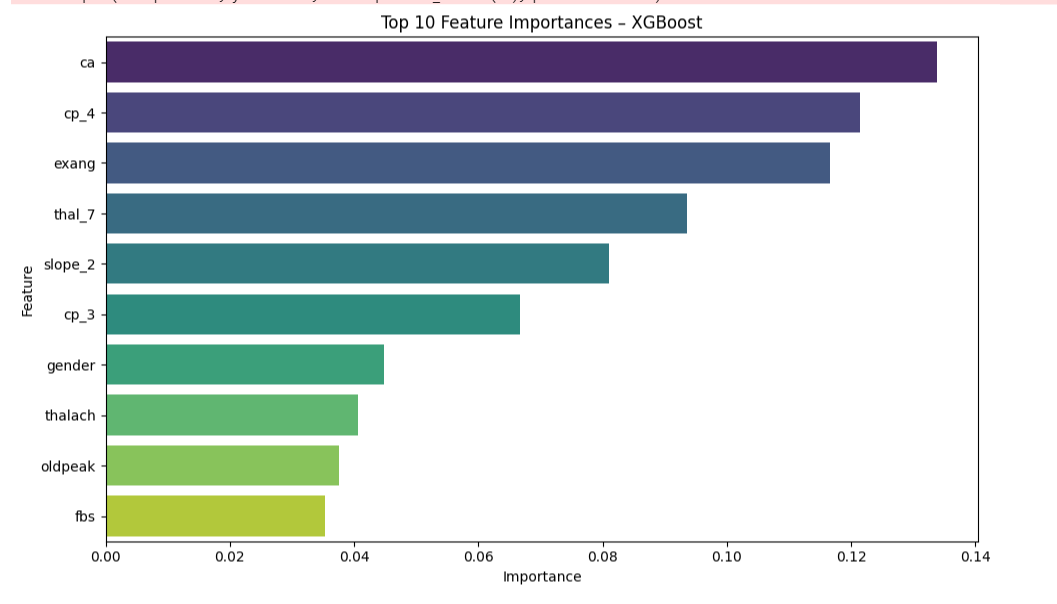


 **best\_xgb.feature\_importances\_**: Returns feature importance scores based on how useful each feature was across decision trees.

 **Pandas DataFrame**: Used to rank and organize the features by importance.

 **Seaborn Bar Plot**: Visualizes the top 10 features contributing most to the model's decision-making process.

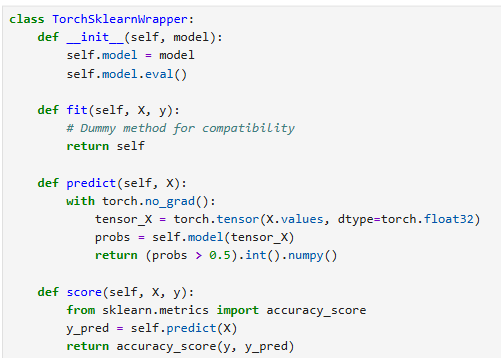
Output is :



This plot emphasizes that the XGBoost model primarily relies on clinical and physiological markers related to cardiac function and specific conditions (ca, cp\_4, exang, thal\_7, slope\_2), along with demographic information (gender) and other physiological measurements (thalach, oldpeak, fbs), to predict heart disease. The high importance of cp\_4 (asymptomatic chest pain) further underscores the potential for silent indicators of the disease.

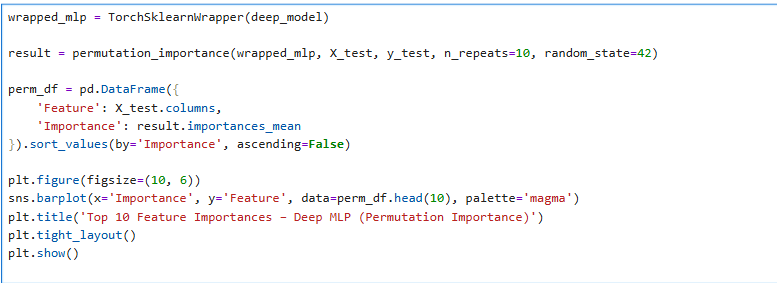
### **4.2.2 Deep MLP Model**

developed the TorchSklearnWrapper class to integrate PyTorch models with the scikit-learn ecosystem. Since scikit-learn expects estimators to follow a specific interface with methods like fit, predict, and score, I created this wrapper to make my PyTorch models compatible without changing their internal logic.

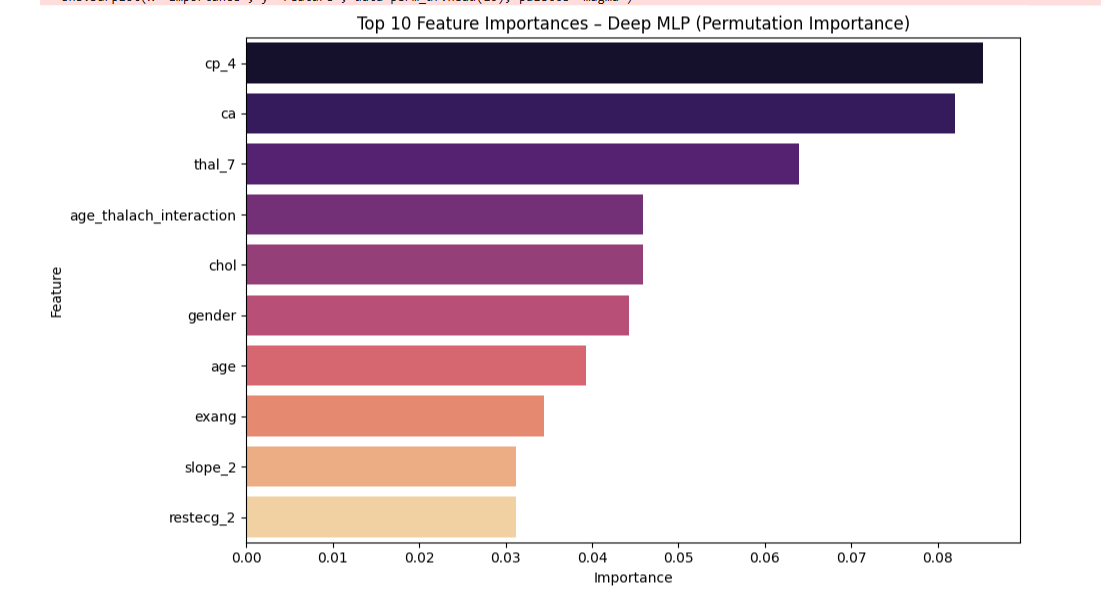


To analyze which features had the most impact on the predictions made by my deep learning model, I used **permutation importance**, a model-agnostic method that evaluates how performance degrades when each feature's values are randomly shuffled.

I began by wrapping my trained PyTorch model (deep\_model) using the TorchSklearnWrapper that I implemented earlier. This allowed the model to interface directly with scikit-learn's permutation\_importance function.



Output is:



1. I used **permutation importance** to evaluate which features had the most impact on my deep MLP model’s predictions.
2. The feature **cp\_4** (chest pain type) showed the highest importance, followed by **ca** (number of major vessels) and **thal\_7** (thalassemia type).
3. My custom feature **age\_thalach\_interaction** also ranked highly, confirming its usefulness in capturing meaningful relationships.
4. Biological factors like **chol** (cholesterol), **age**, and **gender** had moderate influence on model performance.
5. The importance values represent how much accuracy drops when each feature is randomly shuffled.
6. This plot helped me interpret my deep model by showing which features it relies on most for predicting heart disease.

# **Best-Performing Model Selection and Justification**

After evaluating all the trained models using key performance metrics such as accuracy, precision, recall, F1-score, and ROC-AUC, I identified the **XGBoost Classifier (Untuned)** as the best-performing model for this heart disease prediction task.

**Performance Metrics of XGBoost Classifier:**

* **Accuracy**: 0.89
* **ROC-AUC**: 0.90
* **Precision**: 0.89
* **Recall**: 0.89
* **F1-score**: 0.89

Additionally, the confusion matrix reflected strong and balanced performance across both classes, with 26 true positives, 28 true negatives, only 4 false positives, and 3 false negatives.

**Justification for Model Selection:**

* **Consistently High Scores Across All Metrics**: XGBoost outperformed both traditional machine learning models and deep learning models like MLPs across all evaluation metrics.
* **Robust ROC-AUC Score**: With an ROC-AUC of 0.90, it demonstrated excellent ability to distinguish between positive and negative cases.
* **Model Stability**: Even without hyperparameter tuning, XGBoost maintained high performance, showing better generalization compared to tuned models and other algorithms like KNN, SVM, and Random Forest.
* **Domain-Relevant Feature Importance**: The model's feature importance analysis highlighted medically meaningful features such as ca (number of blocked vessels), cp\_4 (asymptomatic chest pain), and exang (exercise-induced angina). This alignment with clinical knowledge supports its interpretability and reliability in real-world applications.

**Note on Deep Learning Models:**

Although the PyTorch-based MLP model also delivered strong results (Accuracy: 0.84, ROC-AUC: 0.86), it slightly underperformed compared to XGBoost. Moreover, the deep model required longer training time and lacked the interpretability of tree-based models, making it less ideal for deployment in clinical settings.

# **Final Verdict:**

Given its superior balance of accuracy, recall, interpretability, and robustness, the **XGBoost Classifier** stands out as the most reliable and deployable model for heart disease prediction in this project.