

Name: Mira Moheb Attia

ID: 211002019

the dataset related to heart disease. The target field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease. These features cover a range of demographic, physiological, and diagnostic indicators that are commonly associated with heart health. Here's a brief description of each feature:

1. Age: The age of the patient.
2. Sex: The gender of the patient (0 = female, 1 = male).
3. Chest Pain Type: Describes the type of chest pain experienced by the patient. It's categorized into four values.
4. Resting Blood Pressure: The blood pressure of the patient while at rest.
5. Serum Cholesterol: Serum cholesterol level in mg/dl.
6. Fasting Blood Sugar: Indicates whether the patient's fasting blood sugar level is above 120 mg/dl (1 = yes, 0 = no).
7. Resting Electrocardiographic Results: Results of resting electrocardiogram, categorized into three values (0, 1, 2).
8. Maximum Heart Rate Achieved: The maximum heart rate achieved during exercise.
9. Exercise Induced Angina: Indicates whether exercise induced angina is present (1 = yes, 0 = no).
10. Oldpeak: ST depression induced by exercise relative to rest.
11. Slope of the Peak Exercise ST Segment: Describes the slope of the peak exercise ST segment.
12. Number of Major Vessels Colored by Fluoroscopy: The number of major vessels (0-3) colored by fluoroscopy.
13. Thal: Thalassemia, categorized into three values (0 = normal, 1 = fixed defect, 2 = reversible defect).

Link: [Heart Disease Dataset \(kaggle.com\)](https://www.kaggle.com/datasets/uciml/heart-disease-dataset)

- Checking Missing Values:
 - `missing_values = df.isnull().sum()` calculates the total number of missing values in each column of the DataFrame `df`.
 - `print(missing_values)` displays the count of missing values for each column.

```
[ ]: #Checkin for missing values
```

```
[6]: missing_values = df.isnull().sum()  
      print(missing_values)
```

```
age      0  
sex      0  
cp       0  
trestbps 0  
chol     0  
fbs      0  
restecg  0  
thalach  0  
exang    0  
oldpeak  0  
slope    0  
ca       0  
thal     0  
target   0  
dtype: int64
```

- Handling Duplicate Rows:
 - `df.duplicated()` identifies duplicate rows in the DataFrame.
 - `df.duplicated().sum()` calculates the total number of duplicate rows.
 - `data = df.drop_duplicates()` removes duplicate rows from the DataFrame and stores the cleaned data in `data`.
 - `data.duplicated().sum() = 0` verifies that there are no duplicate rows in the cleaned data.

```
[28]: # Duplicate values
```

```
[7]: df.duplicated()
```

```
[7]: 0      False
      1      False
      2      False
      3      False
      4      False
      ...
      1020    True
      1021    True
      1022    True
      1023    True
      1024    True
      Length: 1025, dtype: bool
```

```
[8]: df.duplicated().sum()
```

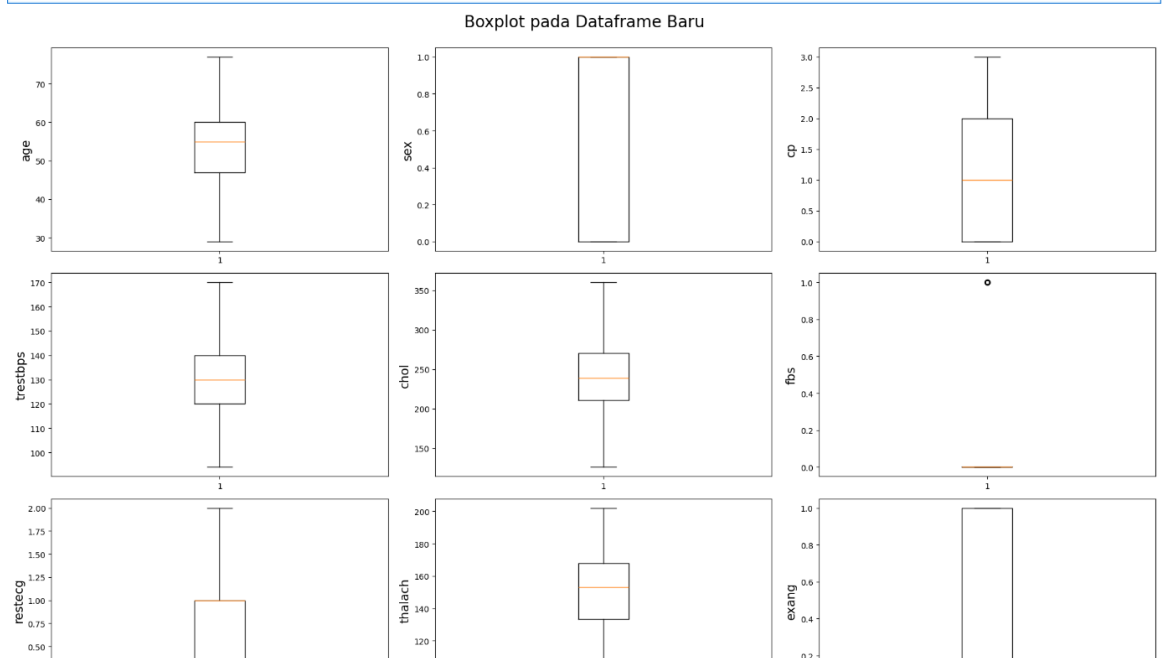
```
[8]: 723
```

```
[9]: data = df.drop_duplicates()
```

```
[10]: data.duplicated().sum()
```

```
[10]: 0
```

- Detecting Outliers:
 - Quartiles and IQR are calculated for the numerical columns ("trestbps", "chol", "thalach", "oldpeak") using quantile ().
 - Lower and upper bounds for outliers are determined based on the calculated quartiles and IQR.
 - Outliers are identified using conditions based on the calculated bounds.
 - Outliers are handled by removing them from the DataFrame and storing the cleaned data in "data2".
 - Boxplots are created for each column in the cleaned DataFrame "data2" to visualize the distribution and identify outliers.
 - The loop iterates through each column, creating subplots for visualization.
 - Output handle outliers.



- Checking Data Consistency and Distribution:
 - Categorical variables (categorical_vars) and numerical variables (numerical_vars) are defined.
 - The loop checks for inconsistencies in categorical variables by printing unique values for each column.
 - Summary statistics are printed for numerical variables using describe ().
 - Boxplots are created to visually inspect the distribution of numerical variables and identify potential outliers.
 - Count plots are generated to visualize the distribution of categorical variables.

```
[18]: categorical_vars = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal', 'target']
      numerical_vars = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
      # Check for inconsistencies in categorical variables
      for column in categorical_vars:
          print(f"Unique values in {column}: {data2[column].unique()}")

      # Check for inconsistencies in numerical variables
      for column in numerical_vars:
          print(f"Summary statistics for {column}:")
          print(data2[column].describe())

      # Check for unexpected values or outliers in numerical variables using box plots
      plt.figure(figsize=(12, 8))
      for i, column in enumerate(numerical_vars, 1):
          plt.subplot(3, 2, i) # Adjusting layout to 3 rows and 2 columns
          plt.boxplot(data2[column])
          plt.title(column)
      plt.tight_layout()
      plt.show()

      # Check for distribution of categorical variables using count plots
      plt.figure(figsize=(12, 8))
      for column in categorical_vars:
          plt.subplot(3, 3, categorical_vars.index(column) + 1)
          sns.countplot(data2[column])
          plt.title(column)
      plt.tight_layout()
      plt.show()
```

```
Unique values in sex: [1 0]
Unique values in cp: [0 1 2 3]
Unique values in fbs: [0 1]
Unique values in restecg: [1 0 2]
Unique values in exang: [0 1]
```

- Feature Engineering:

1. Age Group:

- Categorizes individuals into different age groups based on their age.
- Creates a new column called `age_group` using the `pd.cut()` function, binning the 'age' column into specified ranges.
- Labels for age groups: '<40', '40-59', '60-79', '80+'.

2. Total Cholesterol:

- Combines `chol` (cholesterol) and `fbs` (fasting blood sugar) to calculate total cholesterol.
- Creates a new column `total_chol` by summing 'chol' and 'fbs' for everyone.

3. Exercise Induced Angina & Max Heart Rate Interaction:

- Captures interaction between `exang` (exercise-induced angina) and `thalach` (maximum heart rate achieved).
- Creates `exang_thalach_interaction` by multiplying `exang` and `thalach`.

4. Age and Cholesterol Interaction:

- Represents interaction between age and cholesterol levels.
- Creates `age_chol_interaction` by multiplying `age` and `chol`.

5. Max Heart Rate and Exercise Induced Angina Interaction:

- Captures interaction between maximum heart rate achieved and exercise-induced angina.
- Creates `thalach_exang_interaction` by multiplying `thalach` and `exang`.

- Evaluation Results:

Using the evaluation metrics that were derived from the models:

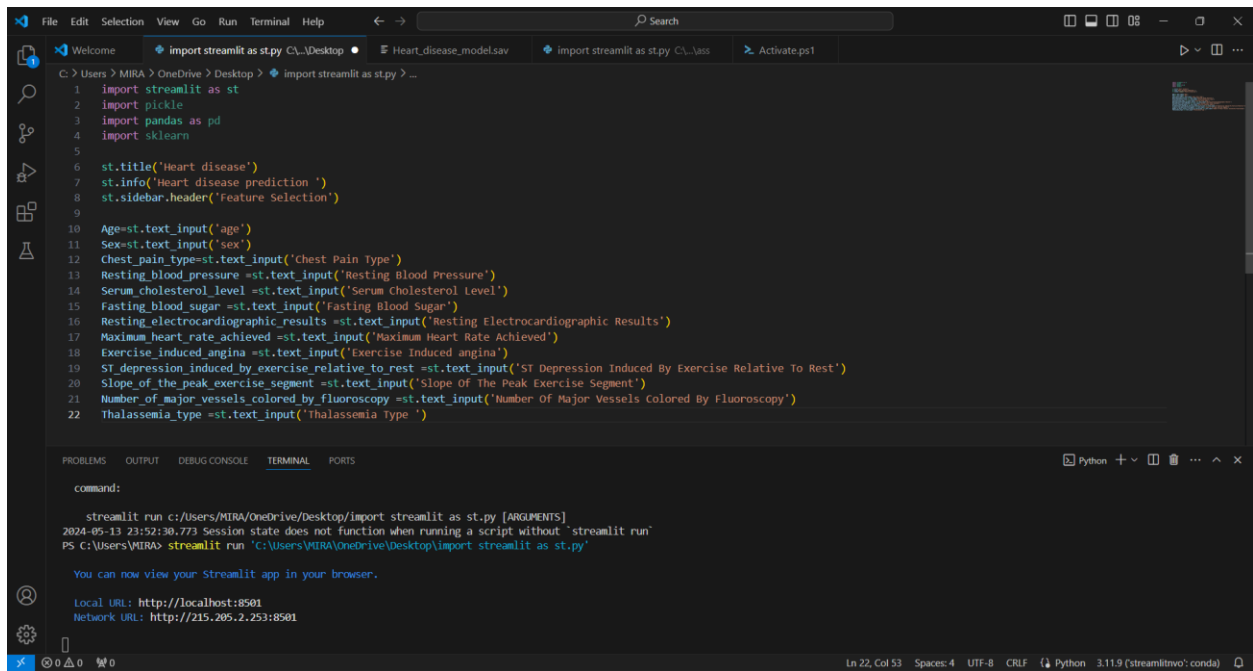
- Logistic Regression: 84.21% accuracy, 82.35% precision, 90.32% recall, 86.15% F1 score, and 83.62% ROC-AUC were attained.
- With Random Forest, 82.46% accuracy, 80.00% precision, 90.32% recall, 84.85% F1 score, and 81.70% ROC-AUC were attained.
- With the support vector machine, 61.40% accuracy, 58.82% precision, 96.77% recall, 73.17% F1 score, and 58.00% ROC-AUC were attained.
- Using gradient boosting, the following results were obtained: 77.19% accuracy, 76.47% precision, 83.87% recall, 80.00% F1 score, and 76.55% ROC-AUC.

	Accuracy	Precision	Recall	F1 Score	ROC-AUC
Logistic Regression	0.842105	0.823529	0.903226	0.861538	0.836228
Random Forest	0.824561	0.800000	0.903226	0.848485	0.816998
Support Vector Machine	0.614035	0.588235	0.967742	0.731707	0.580025
Gradient Boosting	0.771930	0.764706	0.838710	0.800000	0.765509

- Conclusion:

Among the models tested, the Logistic Regression model appears to have the best accuracy, precision, recall, F1 Score, and ROC-AUC. Although it does not perform as well as Logistic Regression, Random Forest still does well on all these metrics. In comparison to the other models, the Support Vector Machine performs worse, as evidenced by its lowest accuracy and ROC-AUC scores. In terms of performance metrics, gradient boosting is situated between random forest and logistic regression. Based on the given evaluation metrics, Logistic Regression appears to be the model that performs the best overall for this task.

GUI using streamlit



```
1 import streamlit as st
2 import pickle
3 import pandas as pd
4 import sklearn
5
6 st.title("Heart disease")
7 st.info("Heart disease prediction ")
8 st.sidebar.header("Feature Selection")
9
10 Age=st.text_input('age')
11 Sex=st.text_input('sex')
12 Chest_pain_type=st.text_input('Chest Pain Type')
13 Resting_blood_pressure =st.text_input('Resting Blood Pressure')
14 Serum_cholesterol_level =st.text_input('Serum Cholesterol Level')
15 Fasting_blood_sugar =st.text_input('Fasting Blood Sugar')
16 Resting_electrocardiographic_results =st.text_input('Resting Electrocardiographic Results')
17 Maximum_heart_rate_achieved =st.text_input('Maximum Heart Rate Achieved')
18 Exercise_induced_angina =st.text_input('Exercise Induced angina')
19 ST_depression_induced_by_exercise_relative_to_rest =st.text_input('ST Depression Induced By Exercise Relative To Rest')
20 Slope_of_the_peak_exercise_segment =st.text_input('Slope Of The Peak Exercise Segment')
21 Number_of_major_vessels_colored_by_fluoroscopy =st.text_input('Number Of Major Vessels Colored By Fluoroscopy')
22 Thalassemia_type =st.text_input('Thalassemia Type ')
```

command:

```
streamlit run c:/Users/MIRA/OneDrive/Desktop/import streamlit as st.py [ARGUMENTS]
2024-05-13 23:52:30.773 Session state does not function when running a script without 'streamlit run'
PS C:\Users\MIRA> streamlit run 'C:\Users\MIRA\OneDrive\Desktop\import streamlit as st.py'

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://215.205.2.253:8501
```

