**School of Computer Science Engineering & Information Science**

**Report on**

**ANALYSIS OF HEART DISEASES USING MACHINE LEARNING**

**Date : Time:**

**Course: Data Analysis And Visualization**

**Course Code**: **CSE2015**

**Activity Conducted By**.

|  |  |
| --- | --- |
| **NAME** | **ROLL NUMBER** |
| M.ADHARSH | 20211ISD0018 |
| K.VISHNU VARDHAN | 20211ISD0021 |
| JATHIN.B.S | 20211ISD0031 |
| ABHAY R ACHARYA | 20211ISD0023 |

**Topic :**

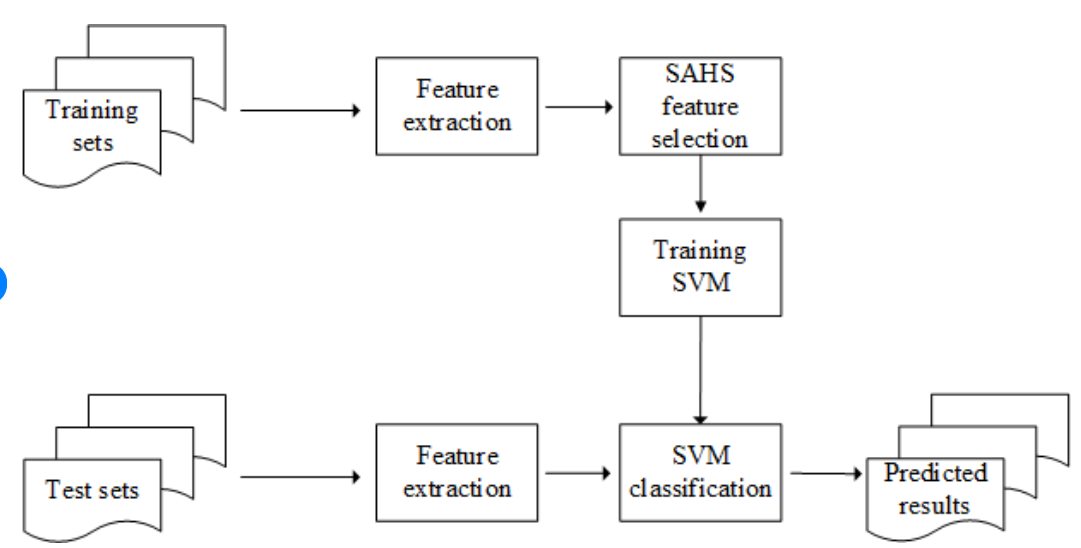
**ANALYSIS OF HEART DISEASES USING MACHINE LEARNING**

**Abstract :**

The disease may be an explicit status that negatively affects human health. Cardiopathy is one of the common deadly diseases that is attributed to unhealthy human habits compared to alternative diseases. With the help of machine learning (ML) algorithms, heart disease can be noticed in a short time as well as at a low cost. This study adopted four machine learning models, such as random forest (RF), decision tree (DT), AdaBoost (AB), and K-nearest neighbor (KNN), to detect heart disease. A generalized algorithm was constructed to analyze the strength of the relevant factors that contribute to heart disease prediction. The models were evaluated using the datasets Cleveland, Hungary, Switzerland, and Long Beach (CHSLB), and all were collected from Kaggle. Based on the CHSLB dataset, RF, DT, AB, and KNN models predicted an accuracy of 99.03%, 96.10%, 100%, and 100%, respectively. In the case of a single (Cleveland) dataset, only two models, namely RF and KNN, show good accuracy of 93.437% and 97.83%, respectively.

**Introduction:**

Heart disease is a broad term encompassing various conditions that affect the heart's structure and function. It's a leading cause of morbidity and mortality worldwide. The most common type is coronary artery disease, which leads to heart attacks. Other forms include heart failure, arrhythmias, and congenital heart defects. Risk factors for heart disease include high blood pressure, high cholesterol, smoking, diabetes, obesity, and a sedentary lifestyle. Symptoms can range from chest pain and shortness of breath to fatigue and dizziness. Prevention strategies involve lifestyle modifications such as regular exercise, a balanced diet, and avoiding tobacco. Early detection through screenings and prompt treatment are crucial for managing heart disease effectively.

**Architecture:**

For analyzing and visualizing Heart disease data across the globe, we can consider the following model of architecture:

1. Data Collection: From the *Kaggle* database, the heart disease data were extracted from the Cleveland dataset [[**63**](https://www.mdpi.com/2227-9032/10/6/1137#B63-healthcare-10-01137)]. Males and females are represented in patients’ datasets. The samples were split into 13 characteristics, with the class distribution being the 14th. In the collected dataset, 138 persons do not have heart disease, while 165 persons do. There are no missing data in this dataset.

2. Data Preprocessing: The dataset’s pattern determines the success of classification challenges. Falling values seldom hamper the result. Therefore, in the beginning, we examined the dataset to see whether it had any lost values or not. The mislaid values can be verified in various ways, including totally ignoring them, replacing them with any numeric value, replacing them with the maximum time resembling that property, or restoring the value with the mean value for that property.

3. Data Integration: Data integration refers to the process of combining data from multiple sources into a unified, coherent view. This unified view enables organizations to analyze and derive insights from their data more effectively.

4. Data Analysis: Data analysis is the process of inspecting, cleaning, transforming, and modeling data with the goal of discovering useful information, informing conclusions, and supporting decision-making..

5. Data Visualization: Create visualizations such as charts, graphs, maps, and dashboards to present the findings of the analysis. Use interactive visualizations to allow users to explore the data dynamically.

**6.Data Exploration**: After cleaning the data, analysts explore the dataset to understand its structure, patterns, and relationships. This may involve summarizing the data with descriptive statistics, visualizing the data with charts or graphs, and identifying trends or outliers.

**7. Statistical Analysis:** Statistical analysis techniques are used to explore relationships between variables, test hypotheses, and make inferences about the population based on sample data. This may involve conducting correlation analysis, regression analysis, hypothesis testing, and analysis of variance (ANOVA).

**8.Validation and Sensitivity Analysis:** It's important to validate the results of data analysis to ensure their accuracy and reliability. This may involve cross-validation techniques, sensitivity analysis, or comparing model performance metrics.

**9.Reporting and Documentation:** Finally, the findings of the data analysis are typically documented in reports, presentations, or dashboards for stakeholders. Clear and concise communication of the results is essential for informing decision-making and driving action.

By following this architecture, data analysis is a iterative process that involves multiple steps, from data collection and cleaning to analysis and interpretation. By applying various analytical techniques, organizations can derive valuable insights from their data to inform strategic decisions, improve processes, and drive innovation.

#### **Classification Modeling**

#### **1. Random Forest**

Random forests organize decision trees on randomly selected information units, prepare a forecast per tree, and opt for the fittest answer through voting. It additionally offers a fairly smart pointer of the feature’s importance. This composite classifier produces varied decision trees and incorporates them to urge the foremost effective result. For tree learning, principally implements bootstrap aggregating or bagging.

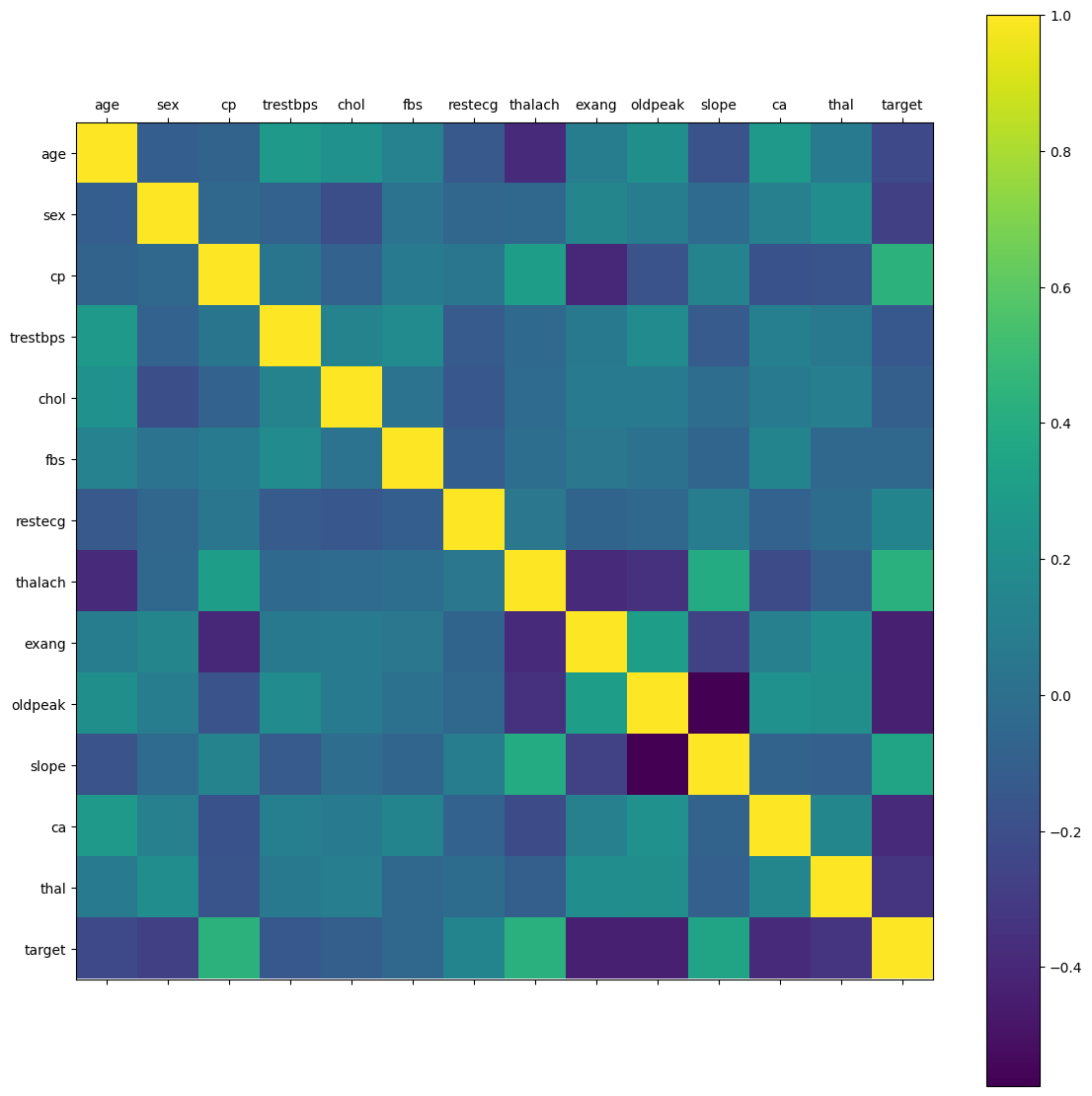
#### **2. Decision Tree**

The decision Tree formula applies to the family of supervised learning algorithms. In distinction to different supervised learning algorithms, the selection tree algorithms are used for locating regression and classification problems. The aim of using a choice tree is to vogue a training model, which can predict the class or advantage of the victim variable by learning easy decision rules induced from training data.

Top of Form

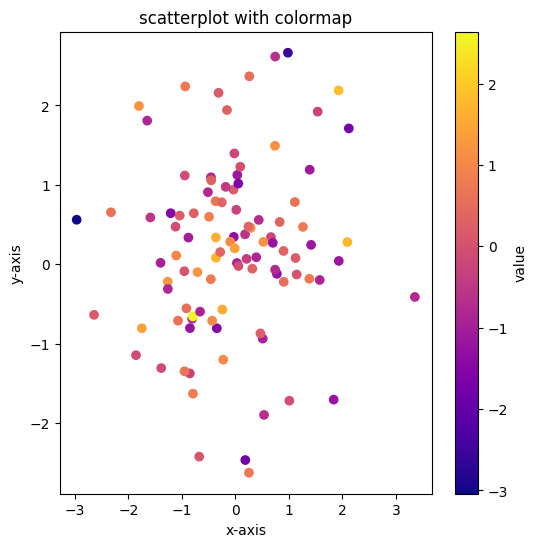
**Outcomes:**

|  |
| --- |
| **Output:** predict heart disease present or not present |
| 1. If (the model has not been trained), then |
| 2. Dataset load; |
| 3. Correlation of data; |
| 4. Check outliers; |
| 5. Remove outliers; |
| 6. Split x and y; |
| 7. Train (80%), test (20%); |
| 8. Load pre-trained model; |
| 9. Educate the model; |
| 10. Save the model that has been trained. |
| 11. Loads trained model if everything else fails; |
| 12. Validate the model using the test data set; |
| 13. Confusion metrics and plot graphs. |

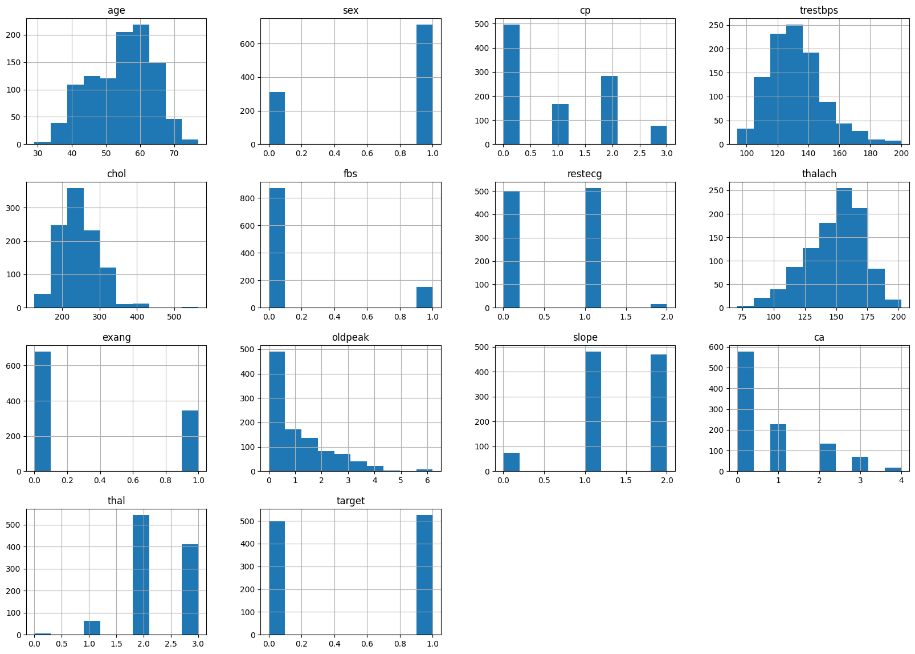


This is used for visualizing the correlation matrix of a DataFrame using the **matplotlib** library in Python. Here's what each line does:

1. **rcParams['figure.figsize'] = 20, 14**: This line sets the size of the figure that will contain the heatmap. The width is set to 20 units and the height to 14 units.
2. **plt.matshow(df.corr())**: This line generates a heatmap of the correlation matrix of the DataFrame **df**. **df.corr()** calculates the pairwise correlation of columns in the DataFrame, and **plt.matshow()** displays this correlation matrix as a heatmap.
3. **plt.xticks(np.arange(df.shape[1]), df.columns)**: This line sets the tick marks on the x-axis. It places ticks at positions corresponding to the columns of the DataFrame **df**, with labels taken from the column names.
4. **plt.yticks(np.arange(df.shape[1]), df.columns)**: This line sets the tick marks on the y-axis. It's similar to the previous line but applied to the y-axis.
5. **plt.colorbar()**: This line adds a colorbar to the plot, which serves as a legend for the colors in the heatmap. It shows the mapping between colors and the correlation values they represent.



1. This creates a DataFrame named **data** with three columns: "x", "y", and "value". The "x" and "y" columns contain 100 random numbers drawn from a standard normal distribution (**np.random.randn(100)**), and the "value" column contains 100 random numbers from the same distribution.
2. The code sets the colormap (**cmap**) to "plasma" and the transparency (**alpha**) of the markers to 1 (fully opaque).
3. This line creates a new figure with a size of 6x6 inches
4. It uses the "x" and "y" columns of the **data** DataFrame for the x and y coordinates of the markers, respectively. The color of each marker is determined by the corresponding value in the "value" column. The **cmap** parameter specifies the colormap to be used, and **alpha** specifies the transparency of the markers.
5. This line adds a colorbar to the plot, which serves as a legend for the colormap. The label "value" is displayed on the colorbar to indicate the variable being represented by the colormap.

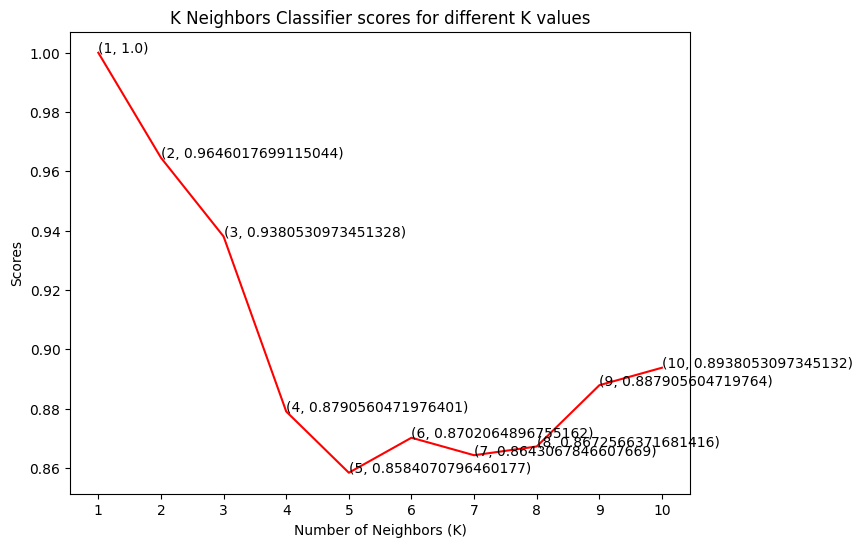


1. When you call **df.hist()**, matplotlib (which pandas uses internally for plotting) creates a separate histogram for each numerical column in the DataFrame. These histograms are typically displayed as a grid of plots, with each plot showing the distribution of values for a single column.
2. The **hist()** method accepts several optional parameters to customize the appearance of the histograms, such as **bins**, **range**, **density**, **cumulative**, etc. These parameters can be used to adjust the number of bins, the range of values to consider, whether to plot a density rather than counts, and whether to create cumulative histograms



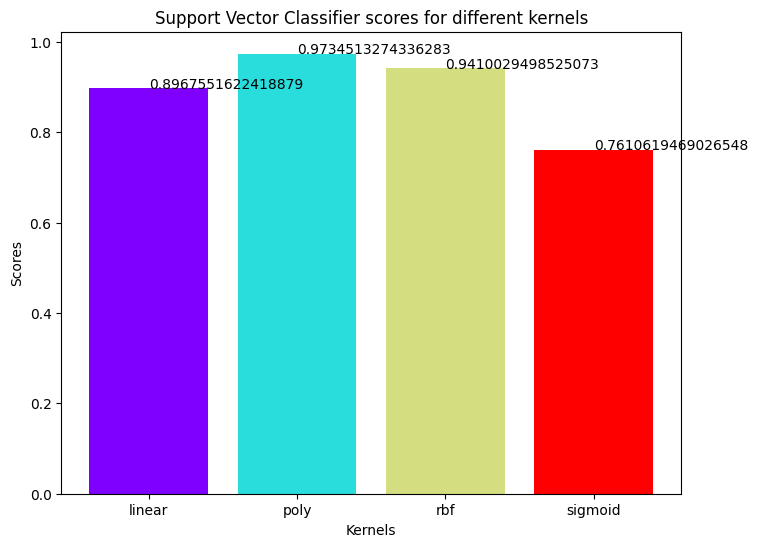
This creates a bar plot displaying the count of each unique value in the 'target' column of the DataFrame **df**. Let's break down the code:

1. **rcParams['figure.figsize'] = 8,6**: This line sets the size of the figure that will contain the bar plot. The width is set to 8 units and the height to 6 units.
2. **plt.bar(df['target'].unique(), df['target'].value\_counts(), color=['red', 'green'])**: This line creates the bar plot. It uses the unique values of the 'target' column as the x-values and the count of each unique value as the y-values. The **color** parameter specifies the colors of the bars for each unique value. The colors are set as red and green in this case.
3. **plt.xticks([0, 1])**: This line sets the tick locations on the x-axis. Since there are only two unique values in the 'target' column, **[0, 1]** are used to specify the positions of the ticks.
4. **plt.xlabel('Target Classes')**: This line sets the label for the x-axis as 'Target Classes'.
5. **plt.ylabel('Count')**: This line sets the label for the y-axis as 'Count'.
6. **plt.title('Count of each Target Class')**: This line sets the title of the plot as 'Count of each Target Class'.



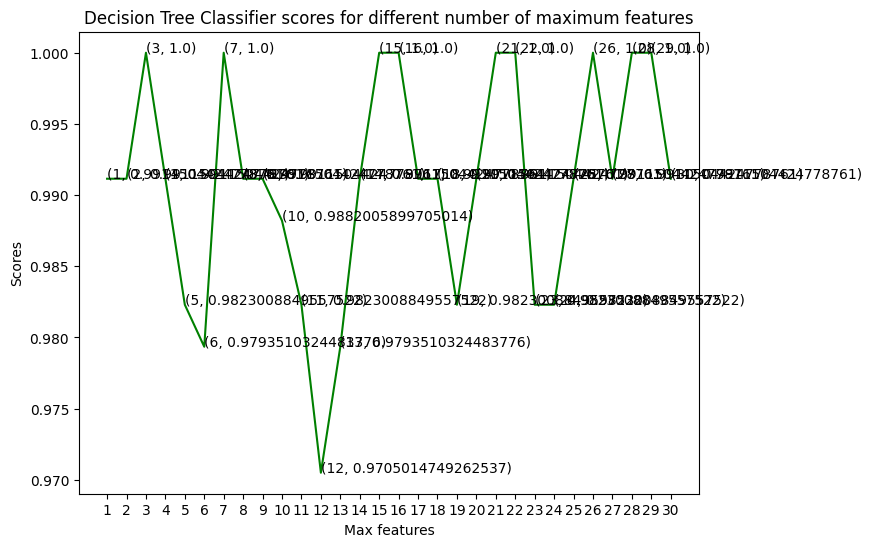
This snippet plots the scores obtained from a k-nearest neighbors (KNN) classifier for different values of k. Here's a breakdown of the code:

1. **plt.plot([k for k in range(1, 11)], knn\_scores, color='red')**: This line plots the scores obtained from the KNN classifier. It uses a list comprehension to generate x-values ranging from 1 to 10 (inclusive) and plots the corresponding scores stored in the **knn\_scores** list. The line is plotted in red.
2. **for i in range(1,11):**: This line starts a for loop that iterates over the numbers from 1 to 10.
3. **plt.text(i, knn\_scores[i-1], (i, knn\_scores[i-1]))**: This line adds text annotations to the plot. For each value of k, it adds text at the coordinates (i, knn\_scores[i-1]) representing the value of k and its corresponding score.
4. **plt.xticks([i for i in range(1, 11)])**: This line sets the tick marks on the x-axis to range from 1 to 10.
5. **plt.xlabel('Number of Neighbors (K)')**: This line sets the label for the x-axis as 'Number of Neighbors (K)'.
6. **plt.ylabel('Scores')**: This line sets the label for the y-axis as 'Scores'.
7. **plt.title('K Neighbors Classifier scores for different K values')**: This line sets the title of the plot as 'K Neighbors Classifier scores for different K values'.



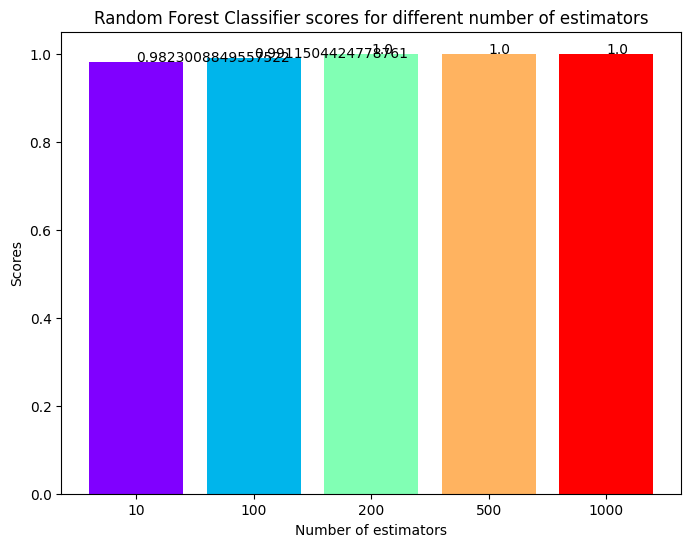
This snippet creates a bar plot displaying the scores obtained from a Support Vector Classifier (SVC) for different kernel types. Here's a breakdown:

1. **colors = rainbow(np.linspace(0, 1, len(kernels)))**: This line generates a list of colors using the **rainbow** colormap from matplotlib. The number of colors generated is equal to the length of the **kernels** list.
2. **plt.bar(kernels, svc\_scores, color=colors)**: This line creates the bar plot. It plots the scores obtained from the SVC classifier on the y-axis (**svc\_scores**) against the kernel types on the x-axis (**kernels**). Each bar is colored according to the colors generated in the previous step.
3. **for i in range(len(kernels)):**: This line starts a loop iterating over the indices of the **kernels** list.
4. **plt.text(i, svc\_scores[i], svc\_scores[i])**: Inside the loop, this line adds text annotations to each bar of the plot. It places the text at the coordinates (i, svc\_scores[i]), where **i** is the index of the current kernel, and **svc\_scores[i]** is the corresponding score.
5. **plt.xlabel('Kernels')**: This line sets the label for the x-axis as 'Kernels'.
6. **plt.ylabel('Scores')**: This line sets the label for the y-axis as 'Scores'.
7. **plt.title('Support Vector Classifier scores for different kernels')**: This line sets the title of the plot as 'Support Vector Classifier scores for different kernels'.



This creates a line plot showing the scores obtained from a Decision Tree Classifier (DTC) for different numbers of maximum features. Let's break down the code:

1. **plt.plot([i for i in range(1, len(X.columns) + 1)], dt\_scores, color='green')**: This line plots the scores obtained from the DTC. It generates x-values ranging from 1 to the number of columns in the feature matrix **X**, and plots the corresponding scores stored in the **dt\_scores** list. The line is plotted in green.
2. **for i in range(1, len(X.columns) + 1):**: This line starts a for loop that iterates over the numbers from 1 to the number of columns in **X**.
3. **plt.text(i, dt\_scores[i-1], (i, dt\_scores[i-1]))**: This line adds text annotations to the plot. For each value of **i**, it adds text at the coordinates **(i, dt\_scores[i-1])**, representing the value of **i** and its corresponding score.
4. **plt.xticks([i for i in range(1, len(X.columns) + 1)])**: This line sets the tick marks on the x-axis to range from 1 to the number of columns in **X**.
5. **plt.xlabel('Max features')**: This line sets the label for the x-axis as 'Max features'.
6. **plt.ylabel('Scores')**: This line sets the label for the y-axis as 'Scores'.
7. **plt.title('Decision Tree Classifier scores for different number of maximum features')**: This line sets the title of the plot as 'Decision Tree Classifier scores for different number of maximum features'.



This snippet creates a bar plot illustrating the scores obtained from a Random Forest Classifier (RFC) for different numbers of estimators. Here's a breakdown of the code:

1. **colors = rainbow(np.linspace(0, 1, len(estimators)))**: This line generates a list of colors using the **rainbow** colormap from matplotlib. The number of colors generated is equal to the length of the **estimators** list.
2. **plt.bar([i for i in range(len(estimators))], rf\_scores, color=colors, width=0.8)**: This line creates the bar plot. It plots the scores obtained from the RFC classifier on the y-axis (**rf\_scores**) against the indices of the **estimators** list on the x-axis. Each bar is colored according to the colors generated in the previous step, and the bars have a width of 0.8.
3. **for i in range(len(estimators)):**: This line starts a loop iterating over the indices of the **estimators** list.
4. **plt.text(i, rf\_scores[i], rf\_scores[i])**: Inside the loop, this line adds text annotations to each bar of the plot. It places the text at the coordinates (i, rf\_scores[i]), where **i** is the index of the current estimator, and **rf\_scores[i]** is the corresponding score.
5. **plt.xticks(ticks=[i for i in range(len(estimators))], labels=[str(estimator) for estimator in estimators])**: This line sets the tick marks and labels on the x-axis. The **ticks** parameter specifies the positions of the ticks, and the **labels** parameter specifies the labels for the ticks. Here, the labels are created by converting each estimator in the **estimators** list to a string.
6. **plt.xlabel('Number of estimators')**: This line sets the label for the x-axis as 'Number of estimators'.
7. **plt.ylabel('Scores')**: This line sets the label for the y-axis as 'Scores'.
8. **plt.title('Random Forest Classifier scores for different number of estimators')**: This line sets the title of the plot as 'Random Forest Classifier scores for different number of estimators'.

**Results:**

In four machine learning algorithms, such as RF, DT, and KNN, were used for both Cleveland, Hungary, Switzerland, and Long Beach (CHSLB) and heart disease datasets. A total of 1025 samples were extracted from the Heart disease database. There are two sorts of diagnoses: normal and patients at risk of heart disease. Among the 1025 samples, 499 showed no evidence of heart illness, and 526 showed evidence of heart disease. Among 303 data in the heart disease dataset, 138 show the absence of heart disease, and 165 identify the presence of heart disease. The confusion metrics for evaluating the heart disease detection system of test data using dataset and Cleveland in our study .

**Conclusion:**

Heart disease is challenging, and it kills thousands of people each year. If the initial signs of heart disease are neglected, the patient may have substantial repercussions in a concise period. This study employed four machine learning models (RF, DT, and KNN) to predict coronary heart disease using heart disease datasets. The data were preprocessed using some appropriate methods and techniques in order to improve the detection accuracy of the used ML models. Among the studied models, the KNN shows a better accuracy of 100% and 97.82% with the disease and Heart disease datasets, respectively. In the case of the Heart disease dataset, RF, AB, and DT models show relatively better accuracy of 99.025%, 96.103%, and 100%, respectively. This type of process intelligence approach is critical in medical diagnosis. Following the improved detection accuracy of the used ML algorithms, a computer-aided smart system together with the freely accessible internet-based cloud hosting platform was developed. It is expected that the developed system will assist in the diagnosis of cardiac problems in a very convenient manner, i.e., making the doctor’s job simpler. Above all, the study has made a significant addition to the computation of strength ratings that are strong predictors of heart disease prognosis.

The applied process can be improved by adding more data, doing k-fold cross-validation, checking for overfitting issues, and testing with more critical or statistically generated data such as numeric data augmentation. The authors consider this to be an upgradable future work.

