Sure! Let's break down the heart disease prediction code step by step:

**Step 1: Import Libraries**

python

Copy code

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, accuracy\_score

* **pandas**: For data manipulation and analysis (especially for handling data in DataFrames).
* **numpy**: For numerical operations (often used for arrays).
* **matplotlib.pyplot**: For plotting and visualizing data.
* **sklearn.model\_selection**: For splitting datasets into training and testing sets.
* **sklearn.tree**: Contains the Decision Tree Classifier.
* **sklearn.ensemble**: Contains the Random Forest Classifier.
* **sklearn.metrics**: For evaluating the performance of the models (like accuracy and classification reports).

**Step 2: Load the Dataset**

python

Copy code

data = pd.read\_csv('heart.csv')

print(data.head())

* **pd.read\_csv**: Reads the dataset from a CSV file.
* **data.head()**: Displays the first few rows of the dataset to understand its structure.

**Step 3: Check for Missing Values**

python

Copy code

print(data.isnull().sum())

* **data.isnull().sum()**: Checks for any missing values in the dataset and sums them up for each column. This helps identify if data cleaning is necessary.

**Step 4: Prepare Features and Target Variable**

python

Copy code

X = data.drop('target', axis=1)

y = data['target']

* **X**: Contains all the features (input variables) by dropping the target variable (assumed to be named 'target').
* **y**: Contains the target variable (output variable), which indicates whether a person has heart disease (1) or not (0).

**Step 5: Split the Data**

python

Copy code

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

* **train\_test\_split**: Splits the dataset into training and testing sets.
  + **X\_train**: Training features.
  + **X\_test**: Testing features.
  + **y\_train**: Training target values.
  + **y\_test**: Testing target values.
  + **test\_size=0.2**: 20% of the data is reserved for testing.
  + **random\_state=42**: Ensures reproducibility of the split.

**Step 6: Train and Evaluate Decision Tree Classifier**

python

Copy code

dt\_classifier = DecisionTreeClassifier(random\_state=42)

dt\_classifier.fit(X\_train, y\_train)

dt\_predictions = dt\_classifier.predict(X\_test)

* **DecisionTreeClassifier**: Initializes a decision tree model.
* **fit**: Trains the classifier using the training data (X\_train, y\_train).
* **predict**: Makes predictions on the test set (X\_test).

python

Copy code

print("Decision Tree Classifier Results:")

print("Accuracy:", accuracy\_score(y\_test, dt\_predictions))

print(classification\_report(y\_test, dt\_predictions))

* **accuracy\_score**: Calculates the accuracy of the model on the test data.
* **classification\_report**: Provides a detailed report showing precision, recall, F1-score, and support for each class.

**Step 7: Train and Evaluate Random Forest Classifier**

python

Copy code

rf\_classifier = RandomForestClassifier(random\_state=42)

rf\_classifier.fit(X\_train, y\_train)

rf\_predictions = rf\_classifier.predict(X\_test)

* **RandomForestClassifier**: Initializes a random forest model.
* The rest of the process (fitting and predicting) is similar to the Decision Tree.

python

Copy code

print("Random Forest Classifier Results:")

print("Accuracy:", accuracy\_score(y\_test, rf\_predictions))

print(classification\_report(y\_test, rf\_predictions))

* This part evaluates the Random Forest model just like the Decision Tree.

**Step 8: Feature Importances (Random Forest)**

python

Copy code

importances = rf\_classifier.feature\_importances\_

indices = np.argsort(importances)[::-1]

* **feature\_importances\_**: Extracts the importance of each feature from the trained random forest model.
* **np.argsort(importances)[::-1]**: Gets the indices of features sorted by importance in descending order.

**Step 9: Visualize Feature Importances**

python

Copy code

plt.figure()

plt.title("Feature Importances")

plt.bar(range(X.shape[1]), importances[indices], align="center")

plt.xticks(range(X.shape[1]), X.columns[indices], rotation=90)

plt.xlim([-1, X.shape[1]])

plt.show()

* This block visualizes the feature importances:
  + **plt.bar**: Creates a bar chart showing the importance of each feature.
  + **plt.xticks**: Sets the x-axis labels to the feature names, rotating them for better readability.
  + **plt.show()**: Displays the plot.

**Summary**

This code provides a complete workflow for predicting heart disease using Decision Trees and Random Forest classifiers. It involves data loading, preprocessing, model training, evaluation, and visualization of feature importances. You can further enhance this workflow by implementing techniques like hyperparameter tuning or cross-validation. If you have any specific questions about any part, feel free to ask!

4o mini

Top of Form

from sklearn.tree import plot\_tree

# Plotting the Decision Tree

plt.figure(figsize=(15, 10)) # Set the figure size

plot\_tree(dt\_classifier, filled=True, feature\_names=X.columns, class\_names=['No Disease', 'Disease'])

plt.title("Decision Tree Visualization")

plt.show()

**Explanation of the Code:**

* **plt.figure(figsize=(15, 10))**: Sets the size of the figure for better visibility.
* **plot\_tree**: This function plots the tree. The parameters are:
  + **dt\_classifier**: The trained Decision Tree model.
  + **filled=True**: Colors the nodes based on the majority class in that node.
  + **feature\_names**: The names of the features for labeling.
  + **class\_names**: Labels for the target classes (e.g., "No Disease" and "Disease").
* **plt.title**: Adds a title to the plot.
* **plt.show()**: Displays the plot.

Bottom of Form