**SP24: DATA-245 Sec 11 – Machine Learning Tech**

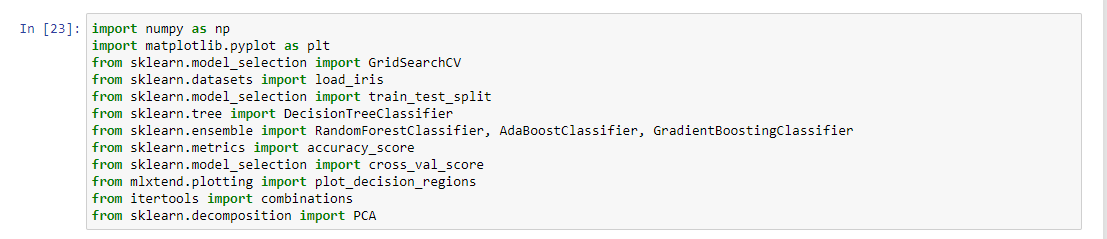
**Homework - - 2**

**Name :- Prayag Nikul Purani**

**SJSU Id :- 017416737**

**Question 5:-**

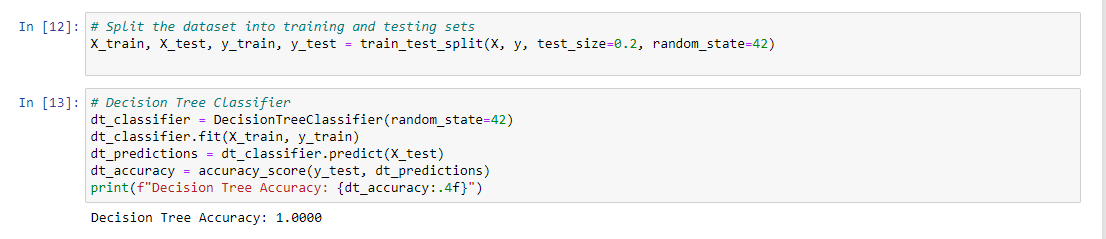
1. **Import Libraries and Load Data**

****

Here, we import necessary libraries, including scikit-learn for machine learning tasks, numpy for numerical operations, matplotlib and seaborn for plotting, and load the Iris dataset using load\_iris(). Import necessary libraries including accuracy\_score, precision\_score, recall\_score, and f1\_score from scikit-learn.

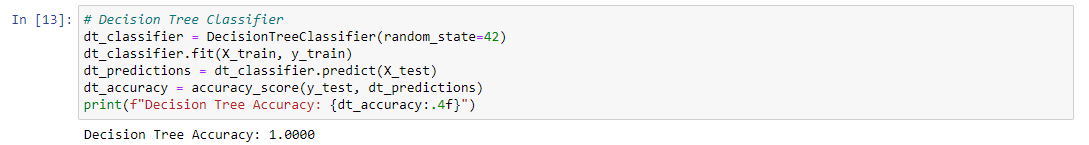
1. **Split the Data**

****

****

We load the Iris dataset and split it into training and testing sets using train\_test\_split. Create a function evaluate\_classifier\_performance that takes the true labels (y\_true) and predicted labels (y\_pred) along with the classifier name. Inside the function, calculate accuracy, precision, recall, and F1-score using appropriate scikit-learn functions. Print the calculated metrics.

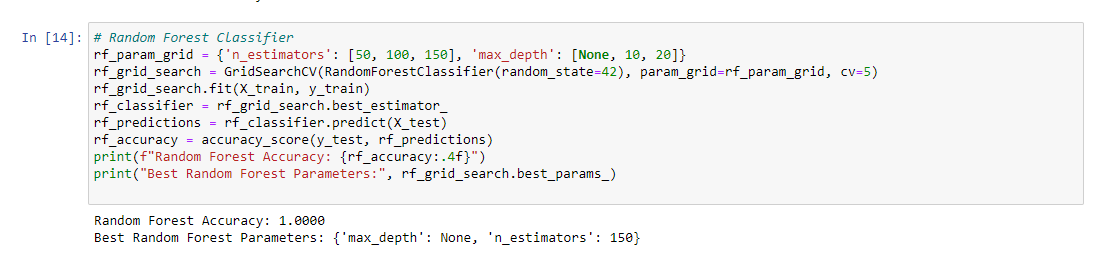
1. **Train and Evaluate Decision Tree Classifier**

****

Train a Decision Tree classifier (dt\_classifier) using the training data (X\_train, y\_train).

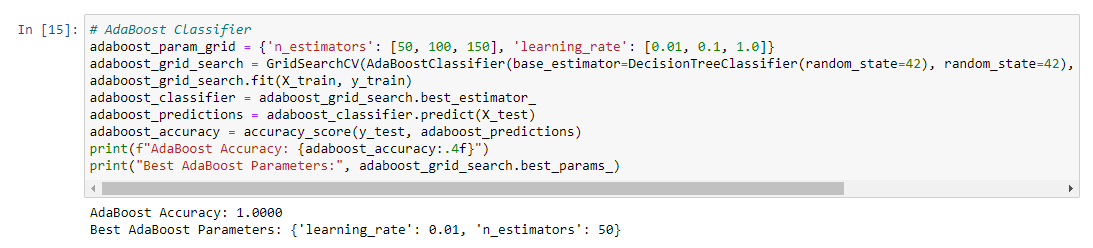
Make predictions on the test data (X\_test) and store them in dt\_predictions. Call evaluate\_classifier\_performance function to print the metrics for the Decision Tree classifier.

1. **Train and Evaluate Random Forest Classifier**

****

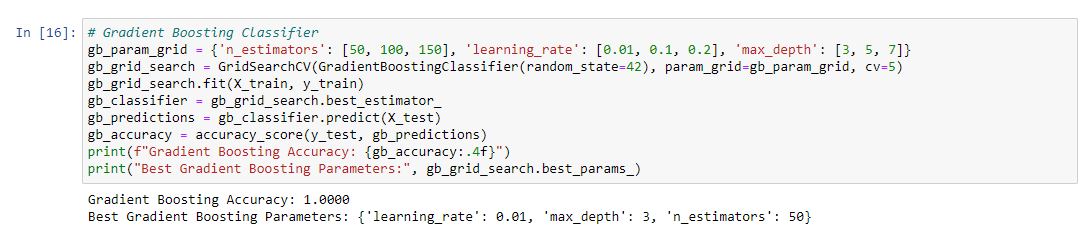
Perform a grid search (GridSearchCV) to find optimal hyperparameters for the Random Forest classifier (rf\_classifier) using the training data. Train the Random Forest classifier with the best hyperparameters. Make predictions on the test data and store them in rf\_predictions. Call evaluate\_classifier\_performance function to print the metrics for the Random Forest classifier.

1. **Train and Evaluate AdaBoost Classifier**

****

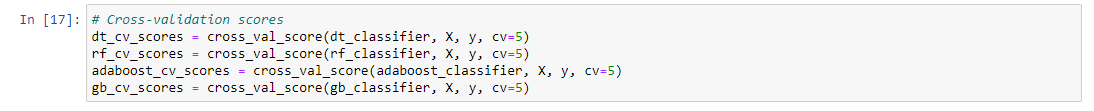
Perform a grid search to find optimal hyperparameters for the AdaBoost classifier (adaboost\_classifier) using the training data. Train the AdaBoost classifier with the best hyperparameters. Make predictions on the test data and store them in adaboost\_predictions. Call evaluate\_classifier\_performance function to print the metrics for the AdaBoost classifier.

1. **Train and Evaluate Gradient Boosting Classifier**

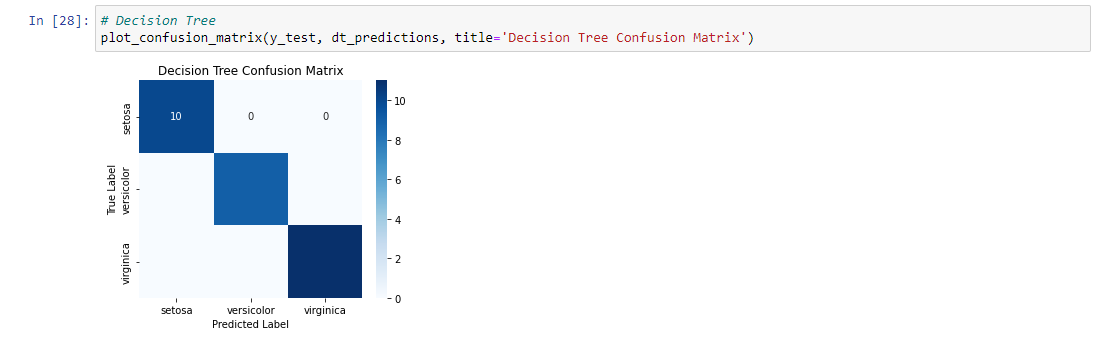
****

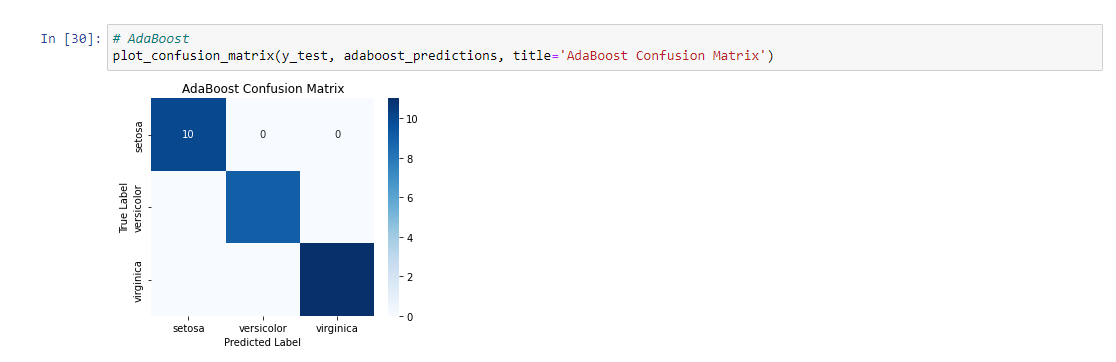
Perform a grid search to find optimal hyperparameters for the Gradient Boosting classifier (gb\_classifier) using the training data. Train the Gradient Boosting classifier with the best hyperparameters. Make predictions on the test data and store them in gb\_predictions. Call evaluate\_classifier\_performance function to print the metrics for the Gradient Boosting classifier.

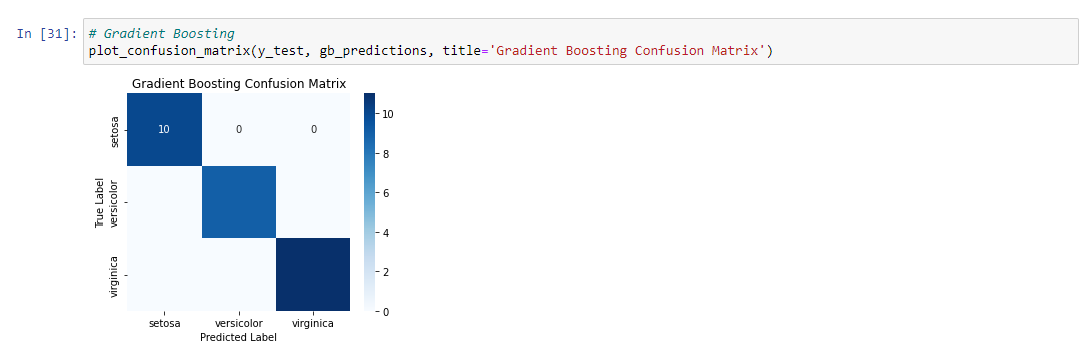
1. **Additional Checks and Debugging**

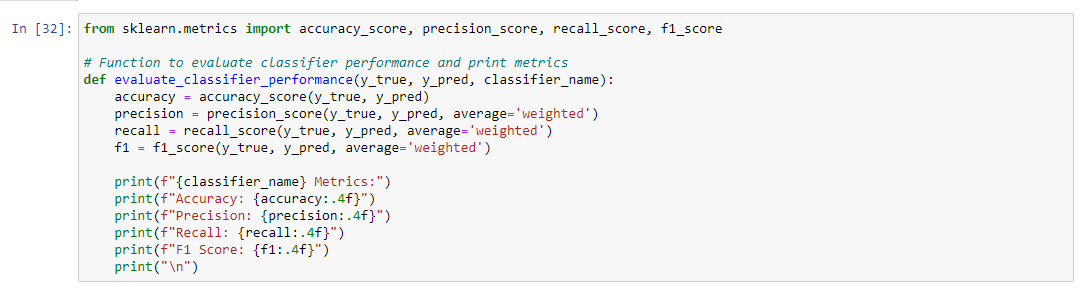
****

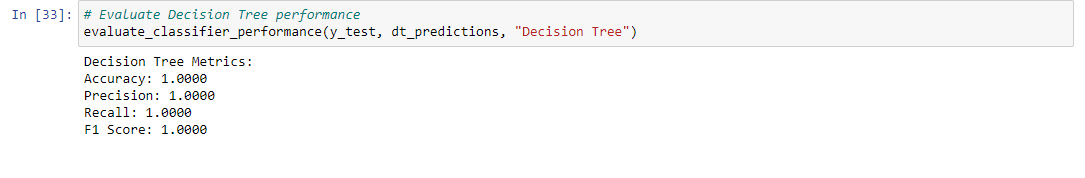
****

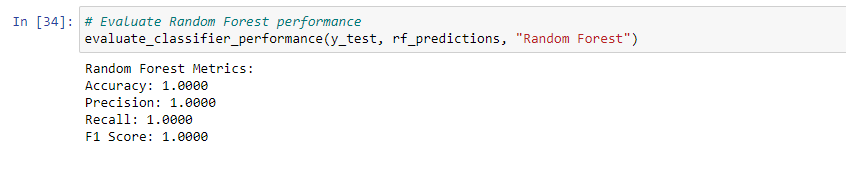
****

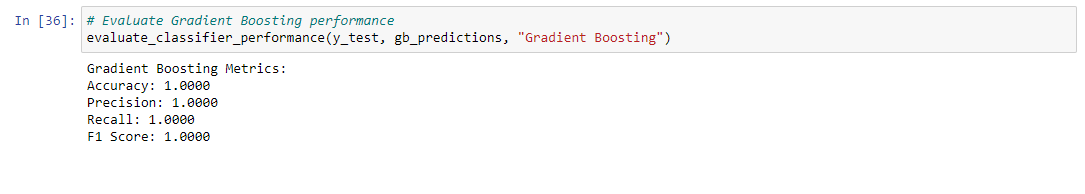
****

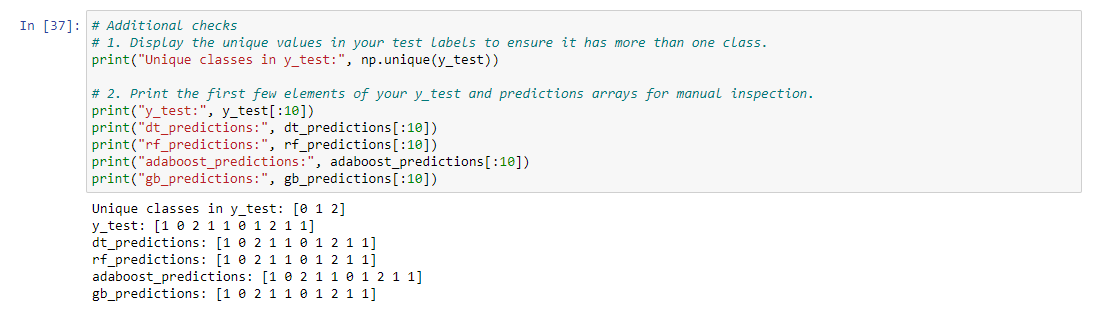
****

****

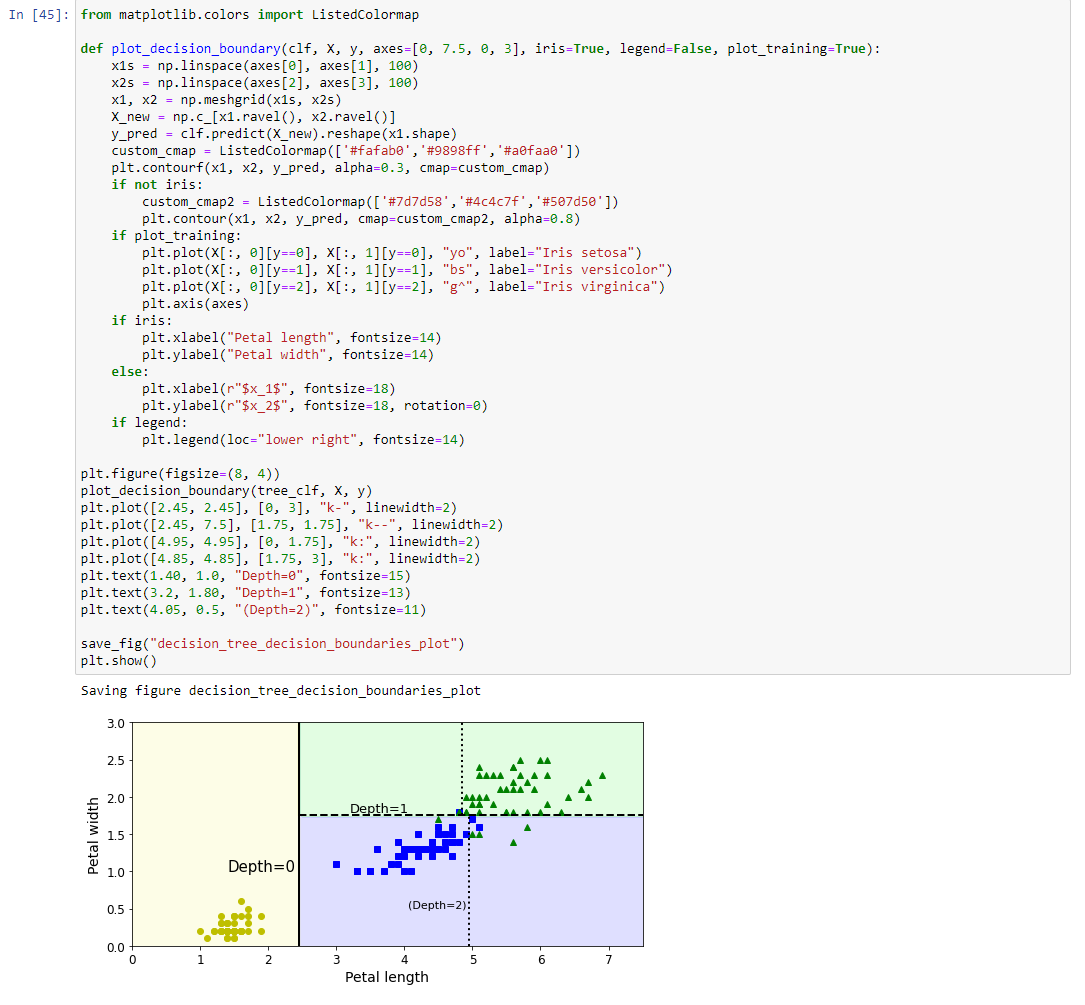
****

****

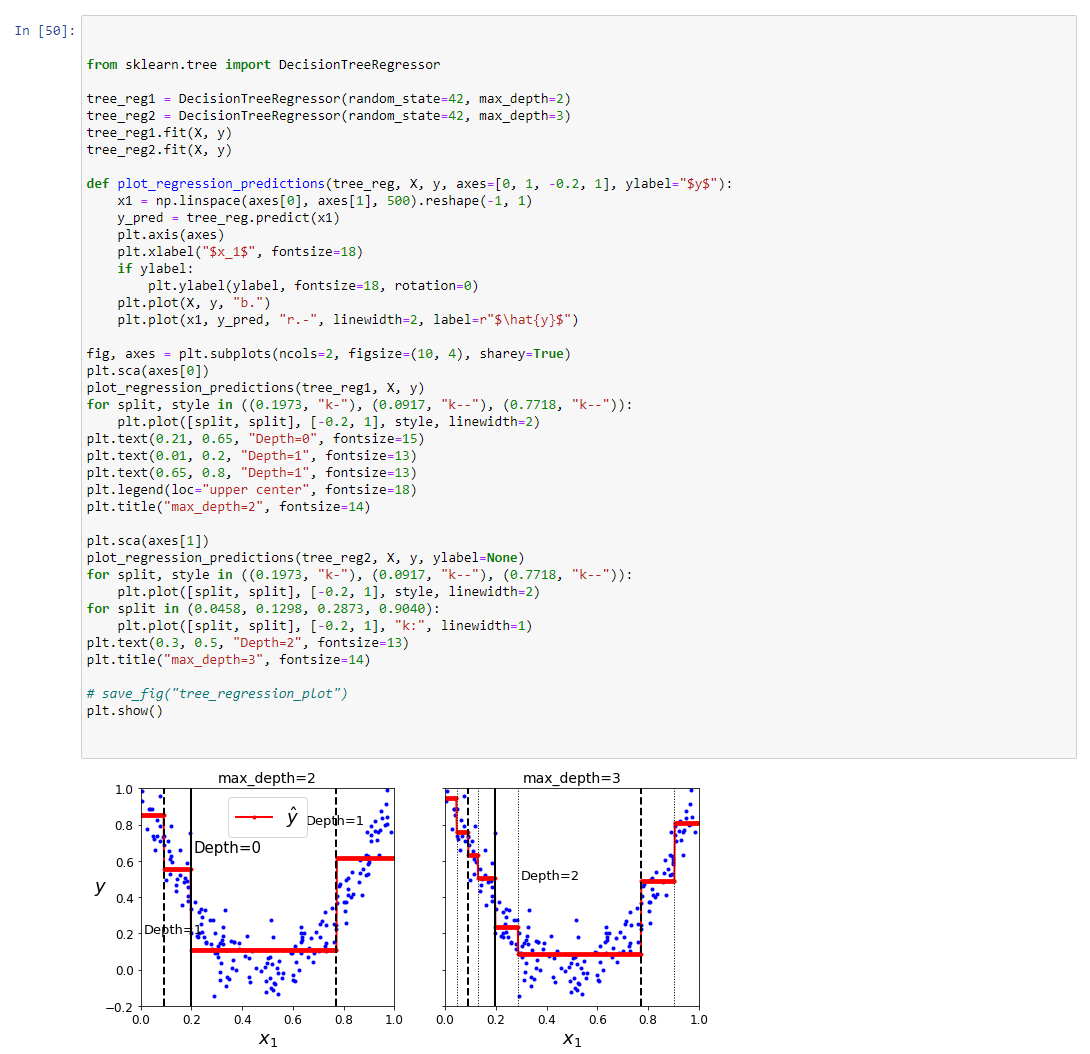
****

****

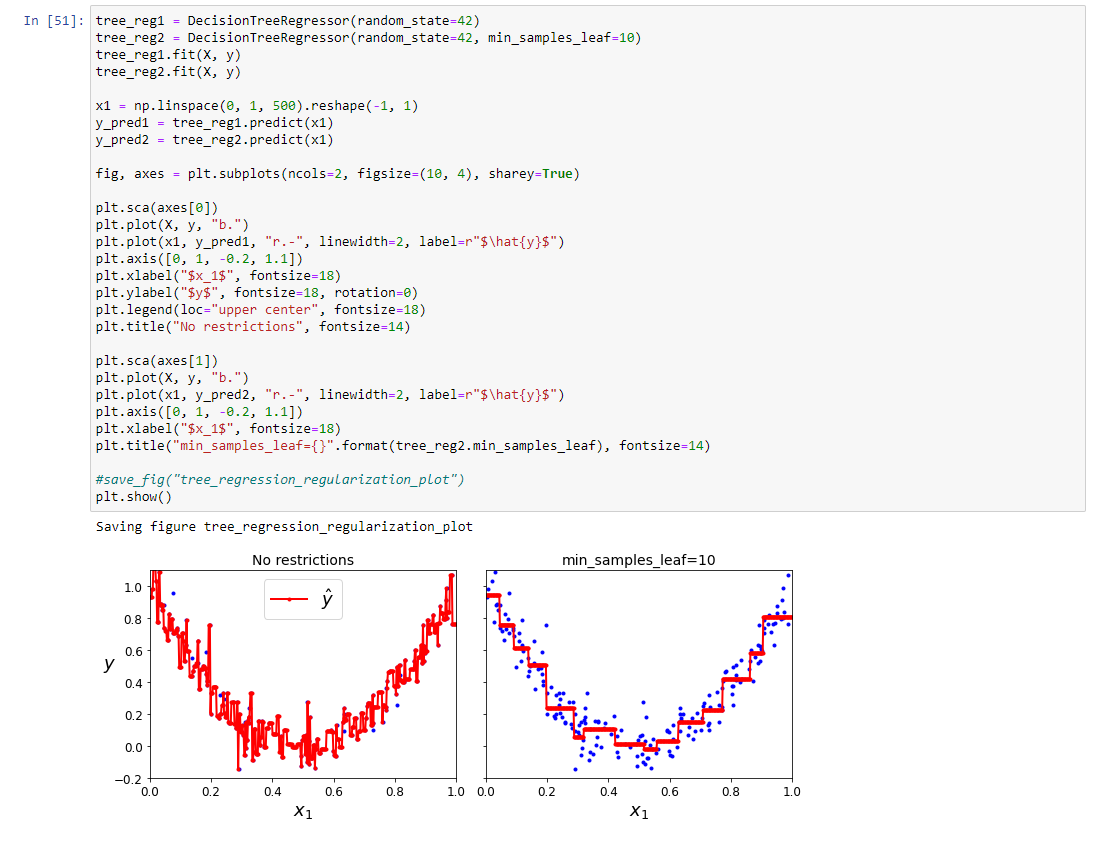
This code defines a function, plot\_decision\_boundary, to visualize decision boundaries of a machine learning classifier (clf) on a 2D dataset (X and y). It uses NumPy and Matplotlib to create a meshgrid of points, predict the class labels for each point, and plot decision boundaries. The function supports customization for different datasets and classifier types. The main script then applies this function to visualize decision boundaries of a decision tree (tree\_clf) on the Iris dataset. Additional annotations and styling are added to enhance the plot, demonstrating decision boundaries at different depths of the decision tree.

1

The code creates and visualizes two decision tree regressors (`tree\_reg1` and `tree\_reg2`) with different maximum depths using the `DecisionTreeRegressor` from scikit-learn. The `plot\_regression\_predictions` function generates plots showing the regression predictions of the trees on a 1D dataset (`X` and `y`). Each subplot represents a tree with its associated decision boundaries and depths. The visualizations include data points (`X`, `y`), regression predictions (`$\hat{y}$`), and vertical lines indicating the splits at different depths. The code demonstrates how varying the maximum depth affects the complexity and fit of the decision tree regressors.



The code compares two decision tree regressors (`tree\_reg1` and `tree\_reg2`) using different regularization settings. `tree\_reg1` has no specific restrictions, while `tree\_reg2` has a minimum number of samples per leaf set to 10 (`min\_samples\_leaf=10`). Both trees are trained on a 1D dataset (`X` and `y`). The code then plots the regression predictions of each tree alongside the original data, illustrating the impact of regularization on the model's flexibility. The second subplot demonstrates how setting a minimum number of samples per leaf can lead to a smoother and less complex regression model, helping prevent overfitting.



The code defines a function `plot\_decision\_boundary` to visualize the decision boundaries of a classifier (`clf`) on a 2D dataset (`X` and `y`). It uses Matplotlib and NumPy to create a meshgrid of points, predict the class labels for each point, and plot decision regions with color-filled contours. The function allows customization of the plot, including specifying the plot boundaries (`axes`), transparency (`alpha`), and whether to display decision contours. The dataset points are also plotted with different colors and transparency based on their class labels. Overall, the function facilitates the visualization of classifier decision boundaries in a 2D feature space.

