Machine Learning Assignment-1

Project Report

Group-number: 27

Pranav Mehrotra (20CS10085) and Saransh Sharma (20CS30065)

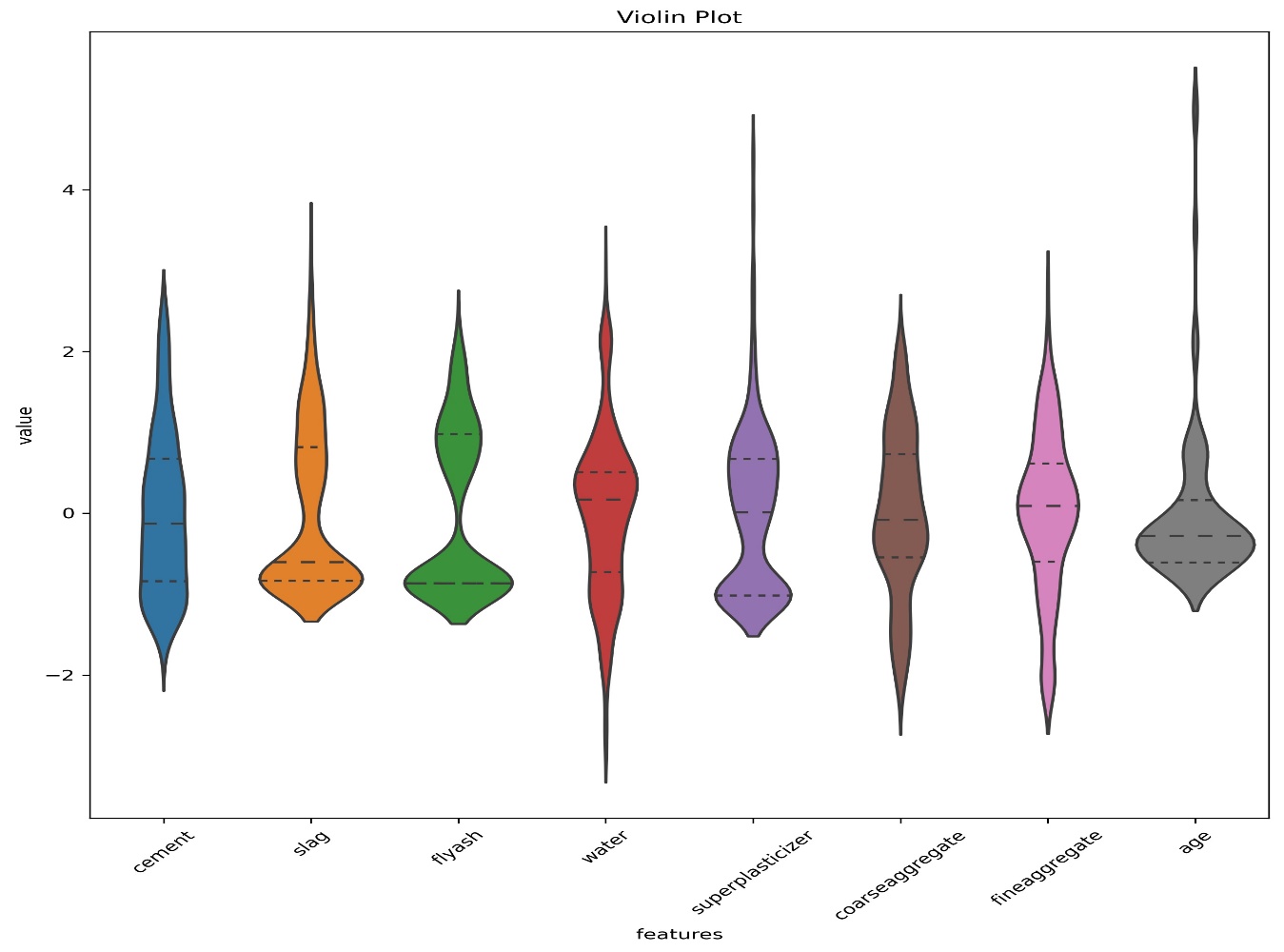
**Question 1) Regression Decision Tree:**

Source File: **Q1.py**

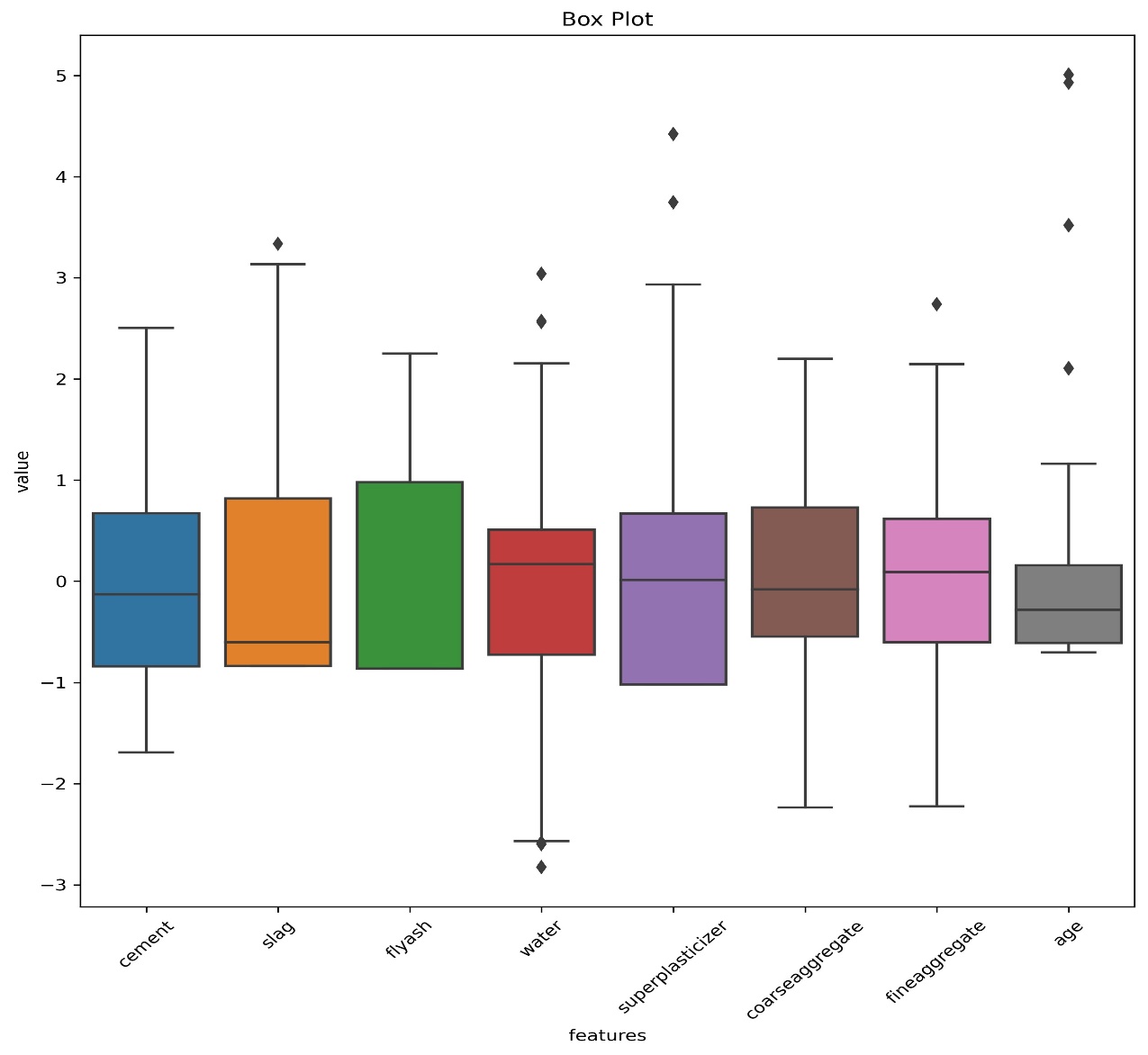
**Exploratory Data Analysis and Data visualization:**

This step is performed to gain insights from data. Data visualizations help us to understand data better and make conclusions about the distribution of data.

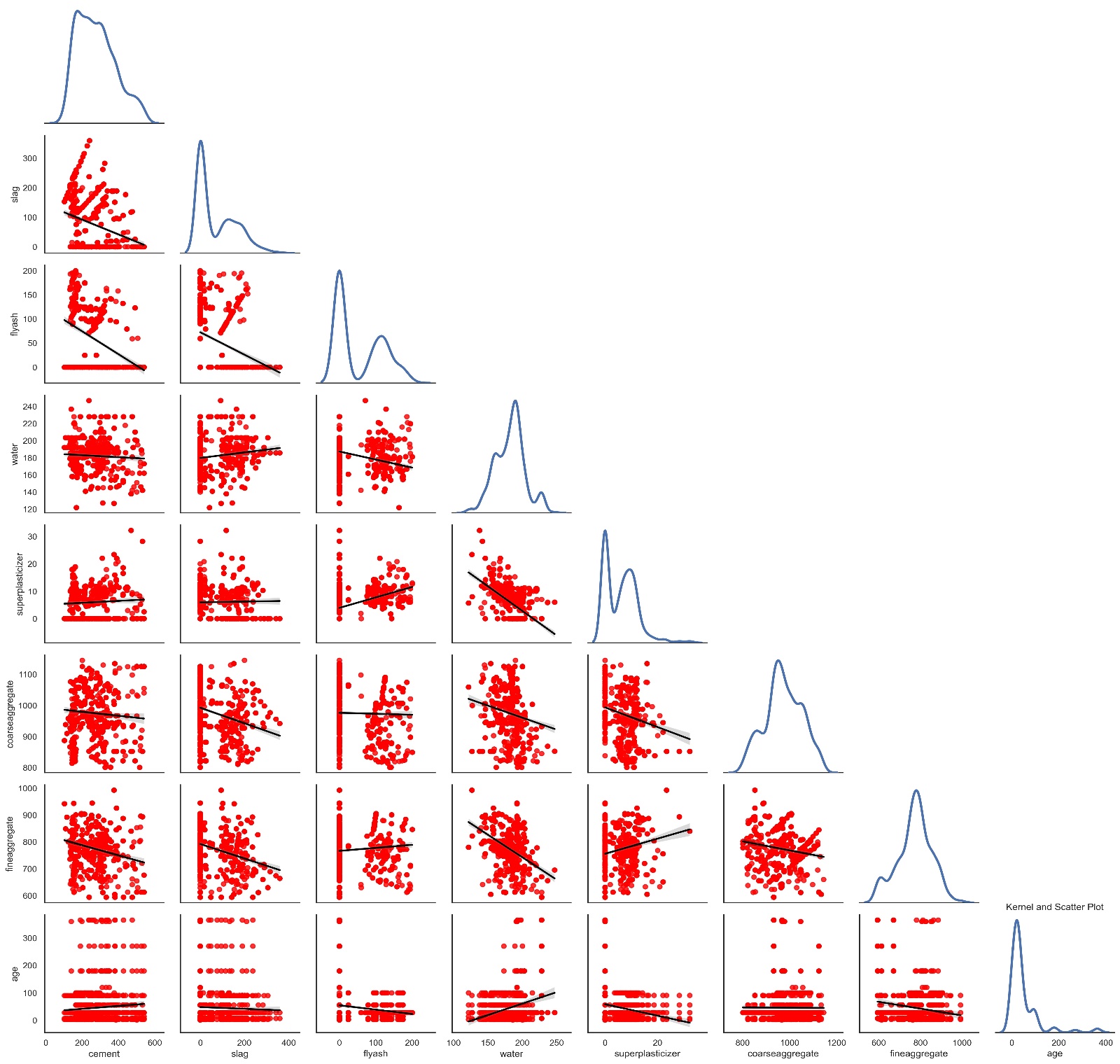
1. Violin Plot: depicts the distribution of different columns of the data along with 25%, 50%(median) and 75% percentile of the data.



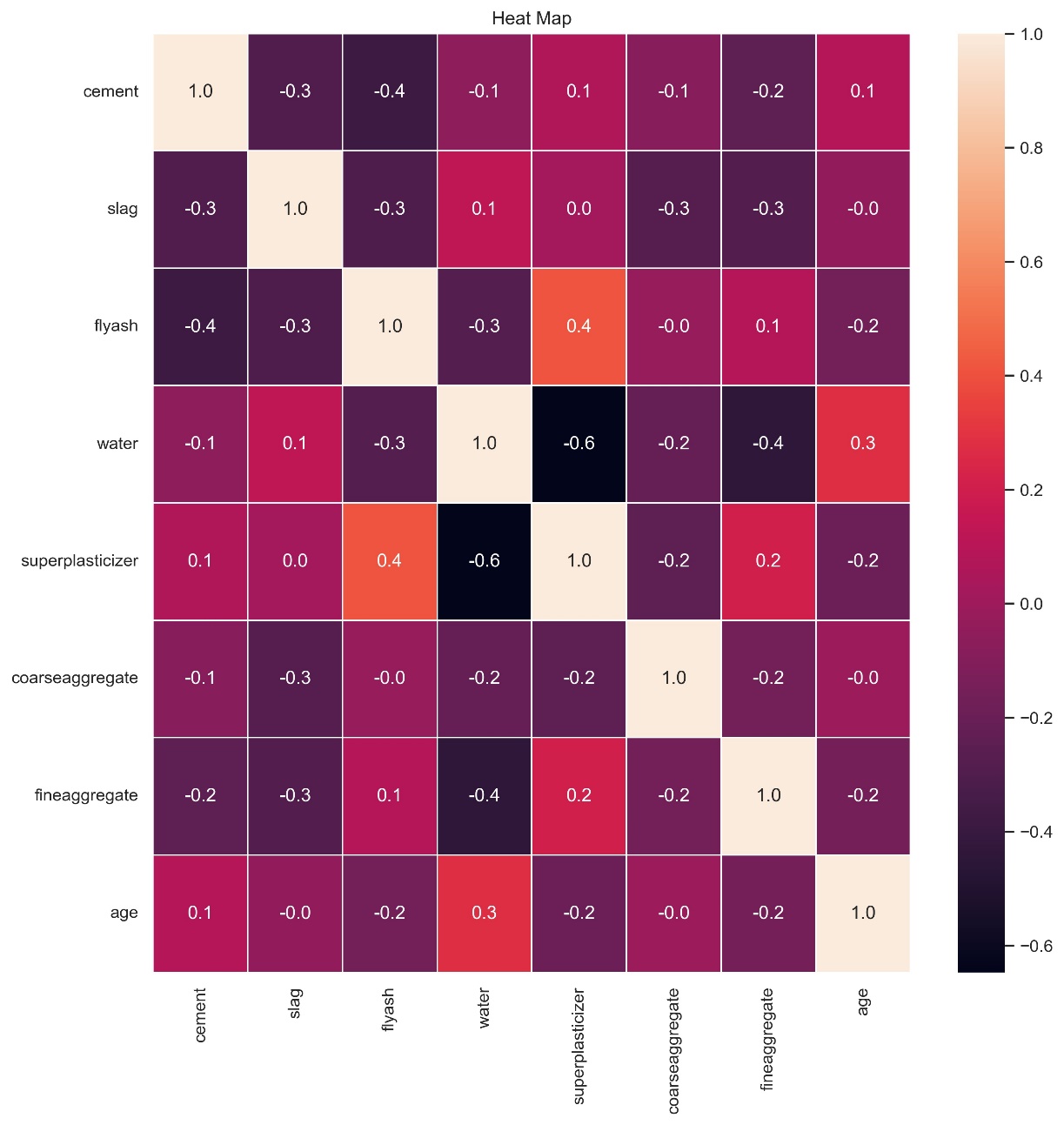
1. Box plot: depicts the spread of each column of data.



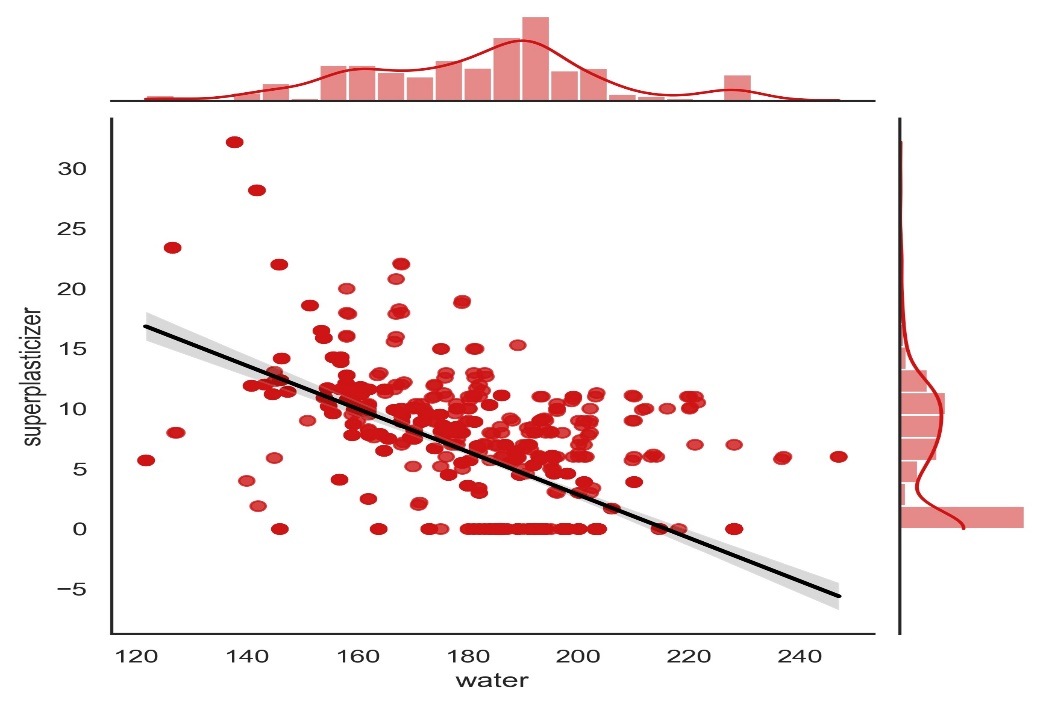
1. Kernel\_Scatter\_plot: depicts individual kernel density of features and scatter plots between every pair of features. Scatter plots help us to explore inter feature relationships.



1. Heatmap: depicts the correlation i.e. extent of linear dependability between different features of the data.



1. Jointplot: explicitly analyse the pair which has the highest correlation to check if these features are highly dependable.



**Class Definations:**

1. **Node**:

Data members:

1. **attribute**: To store the attribute index with respect to whom a node is split.

2. **threshold**: to store the threshold of the attribute used to split the data.

3. **child\_left**: to store the left child of a node.

4. **child\_right**: to store the right child of a node.

5. **variance\_red**: To store the variance reduction caused by splitting the dataset at node with respect to node.attribute and node.threshold.

6. **level**: To store the level of the Node, the root being at level 0.

7. **leaf\_value**: To store the leaf\_value of the node.

The same definition of node will be used to represent two types of nodes, i.e. decision node and leaf node. Decision node will help us in splitting data while leaf nodes will help us to make predictions.

Decision\_node would have leaf\_value = None. This None will help us in distinguishing a between leaf\_node and decision\_node.

Leaf\_node: all other attributes of except leaf\_value is None, while leaf\_value will store the value that will help us to make predictions.

Member functions:

**\_\_init\_\_** : constructor to construct an instance node taking in attribute, threshold, child\_left, child\_right, level, variance\_reduction and leaf\_value.

1. **RegressionTree**:

Data members:

1. **root**: To store the root of the tree.
2. **minimum\_samples**: to keep track of how many minimum samples you should have before you decide to split the data.
3. **maximum height**: to be used when the tree height is prevented from growing larger than a threshold.

Member functions:

1)**\_\_init\_\_**: constructor to create an instance of tree taking in as parameters root, minimum\_samples and max\_height.

2) **variance\_reduction**: takes in parent dataset, left\_branch dataset and right branch dataset and calculates the reduction in the variance of parent dataset when it is split into left and right branches. The variance reduction is calculates as variance of parent – weighted sum of variance of children.

3) **split\_left\_right**: takes in as parameter dataset to be splitted, index of attribute and threshold that will determine how the dataset is to be splitted. It returns two arrays : left\_dataset ( all datapoints whose value of attribute is less than or equal to the threshold) and right\_dataset(all datapoints whose value of attribute is more than the threshold).

4) **cal\_leaf\_node**: takes in as parameter dataset at a node and returns the leaf\_value of the leaf\_node i.e. the mean of output labels .

5) **get\_best\_feature**: takes in as input dataset, number of datapoints and number of features. The function iterates over all features. For every feature, it extracts the unique values corresponding to that feature. It then sorts these values, calculate mean of adjacent values which would act as possible thresholds for data splitting. The data is then split wrt the threshold and variance reduction are calculated. The feature and threshold that yields maximum variance reduction is returned in the form of dictionary.

6) **construct\_tree**: takes in as input dataset and current depth. The function first divides the data into feature matrix and output vector. Call get\_best\_feature for the present dataset to get the splitting attribute for the root at depth=0. Then recursively call the function to grow the left subtree and right tree and assign them to root. The function recursively goes down the tree, calculates the best splitting for every node and at any point if the variance reduction is non-positive or we are left with 1 sample, the node is made a leaf\_node.

7)**construct\_tree\_depth**: same as construct tree function, but keeps a track of current depth. If the current depth is equal to the max\_depth, the tree construction process is terminated at that point and the last node is made a leaf\_node.

8) **fit\_model**: takes in as input feature matrix X and output vector y. It then concatenates them to form dataset and call construct tree function.

9) **fit\_model\_depth**: takes in as input X and y and returns a tree of height atmost max\_height.

10) **predict**: takes in as input datapoint x and tree. It then traverses the tree based upon splitting attributes and splitting thresholds until it reaches a leaf node, and returns the value of leaf\_node it encounters as output for this datapoint.

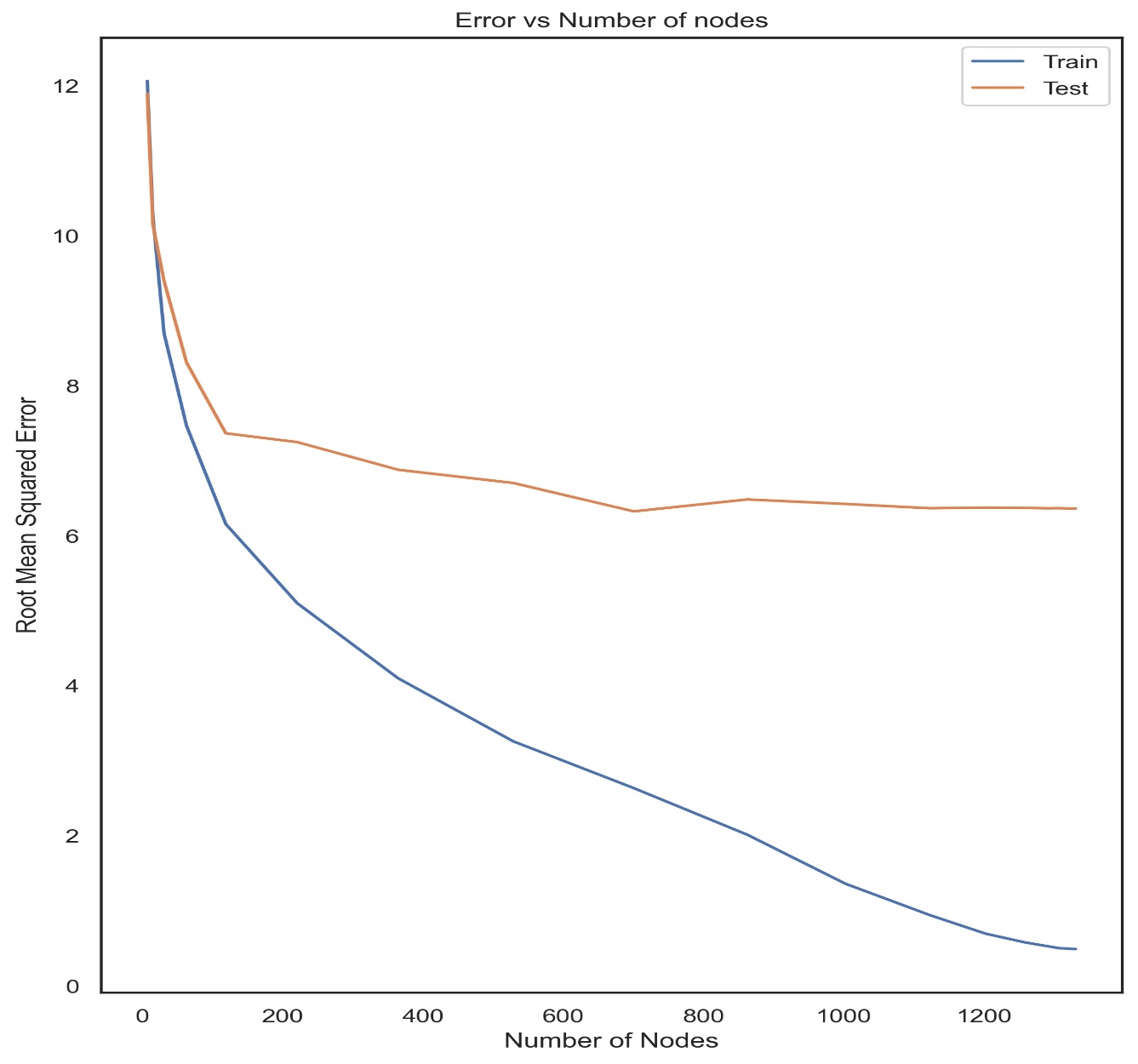
11) **post\_pruning**: takes in as input root, decision\_tree pointing to a node, error list, depth list and original dataset. For every node, the algorithm tries to convert this node to a leaf node and check if error on actual dataset has improved or not. If the error has improved, the minimum error is updated. The number of nodes at the tree after making the node leaf and error at this instance is then appended to depth list to help us plot how number of nodes and error varies during the pruning process. If converting the node doesn’t help, the same process is called for left\_child and right\_child of the node. This way recursively from top to down decision nodes are being converted to leaf nodes and length of each branch is reduced.

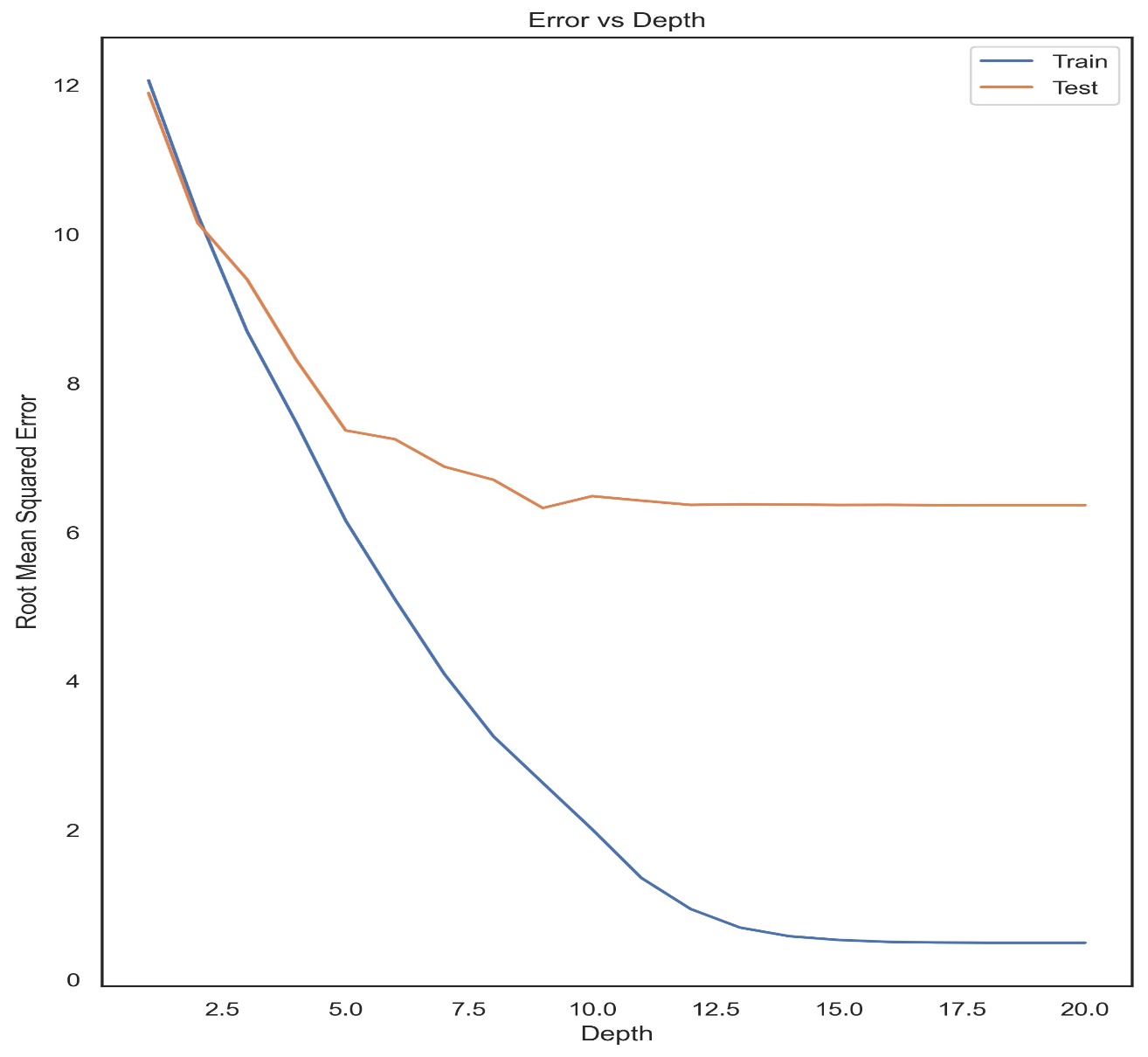
**Helper functions:**

1. **rec\_height**: recursively travels every node and appends the level of the node into the height list.
2. **find\_height**: calls rec\_height function and returns the maximum level of the list.
3. **changenode**: takes in input root, node whose value needs to be changed and new value of data members of this node. The algorithm recursively travels the tree until it finds the node and then updates the values of data members of this node. This function performs a inplace update and returns the root of the updated tree.
4. **num\_nodes**: recursively calculates the number of nodes in left and right subtrees rooted at root and adds 1 to return total number of nodes in the tree.
5. **printtree**: prints the nodes at different heights from root to bottommost node.
6. **printlevel**: to print the tree such that all nodes at the same level are printed in the same line.
7. **mean\_error**: to calculate root mean squared error on predicted output calculated wrt original output.

**Model Training:**

1. The data is shuffled randomly and a tree is trained on that data. The shuffled data is split into training and test data **(70-30)** and further subdivided into feature matrix x and output vector y. Regression tree with the training data is constructed and error on train set is recorded. This process is repeated 10 times. The dataset that gives minimum training error would be further used to train the tree.
2. Train the tree for different heights using fit\_model\_depth and calculate num of nodes and error on test set at each iteration. Plot error vs depth and error vs number of nodes plot.

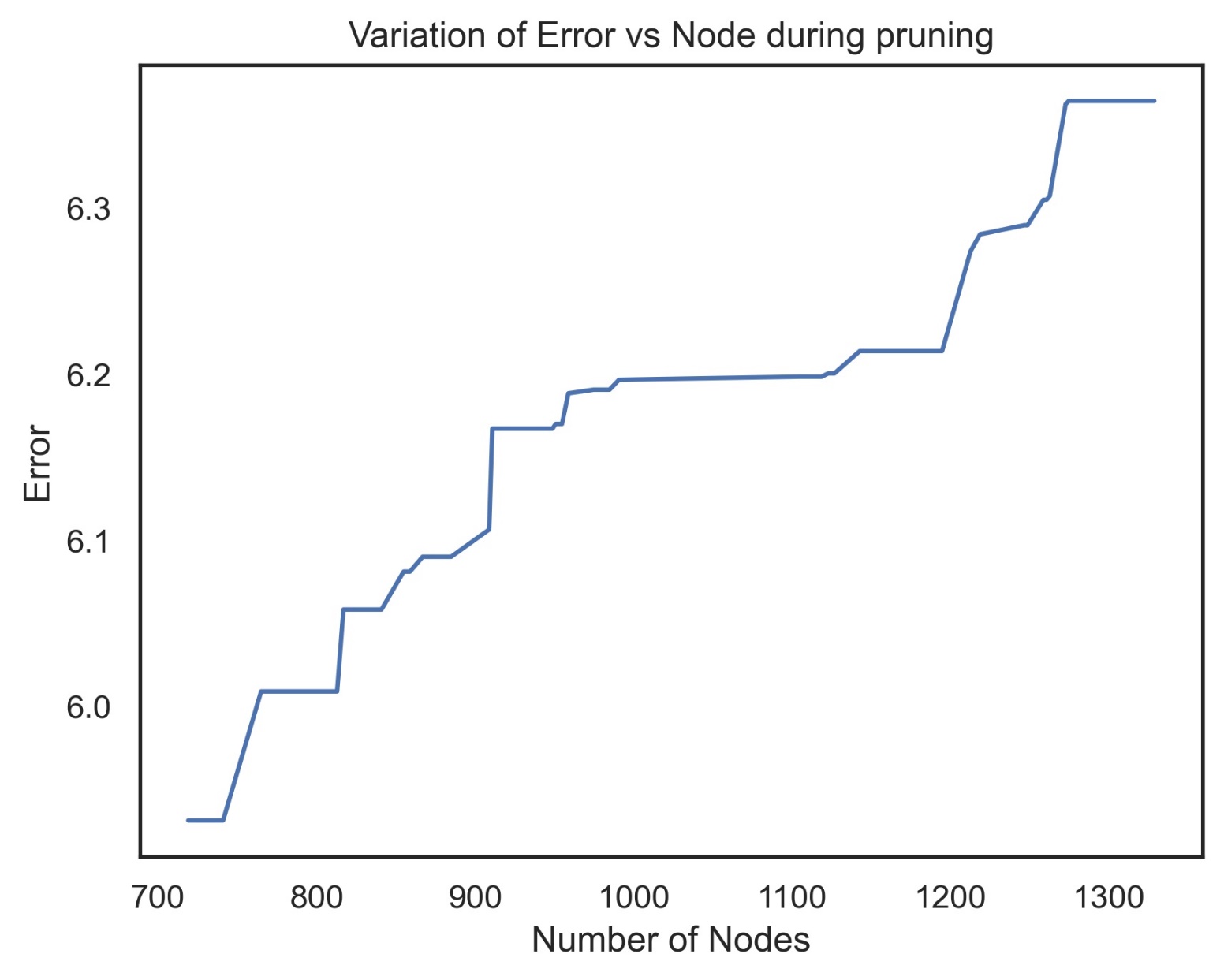




1. The plot suggests that optimal number of nodes are **701**, while optimal height is around 9. The tree starts to exhibit overfitting after height of 12 and **maximum overfitting occurs at 20**.
2. Then we construct the tree without any depth constraints. The tree grows up to **height of 20** with **1331 nodes** indicating overfitting. The error on test set is **6.36435807**. We would take **number of nodes as an indication of height or size** of tree to make further analysis into variation of error with size during pruning.
3. This tree is then printed on a file named **decision\_tree.txt**
4. We then perform post-pruning of this tree and store the number of nodes and error at every instance of pruning.
5. The number of nodes after pruning is **719** (very close to optimal number of nodes) while th test error has reduced to **5.93129378**.
6. The pruned tree is then printed in a file named as **pruned.txt.**

Summary of results:

1. After 10 random splits, the best training accuracy obtained is: **0.49198919** and at height: **20**
2. The tree overfits at height: **20**
3. The optimal number of nodes i.e. number of nodes corresponding to minimum test error is: **701**
4. Number of nodes before pruning: **1331**
5. Test error before pruning: **6.36435807**
6. Number of nodes after pruning: **719**
7. Test error after pruning: **5.93129378**



Before pruning

After pruning