Machine Learning Assignment-1

CS60050

Project Report

Group-number: 27

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

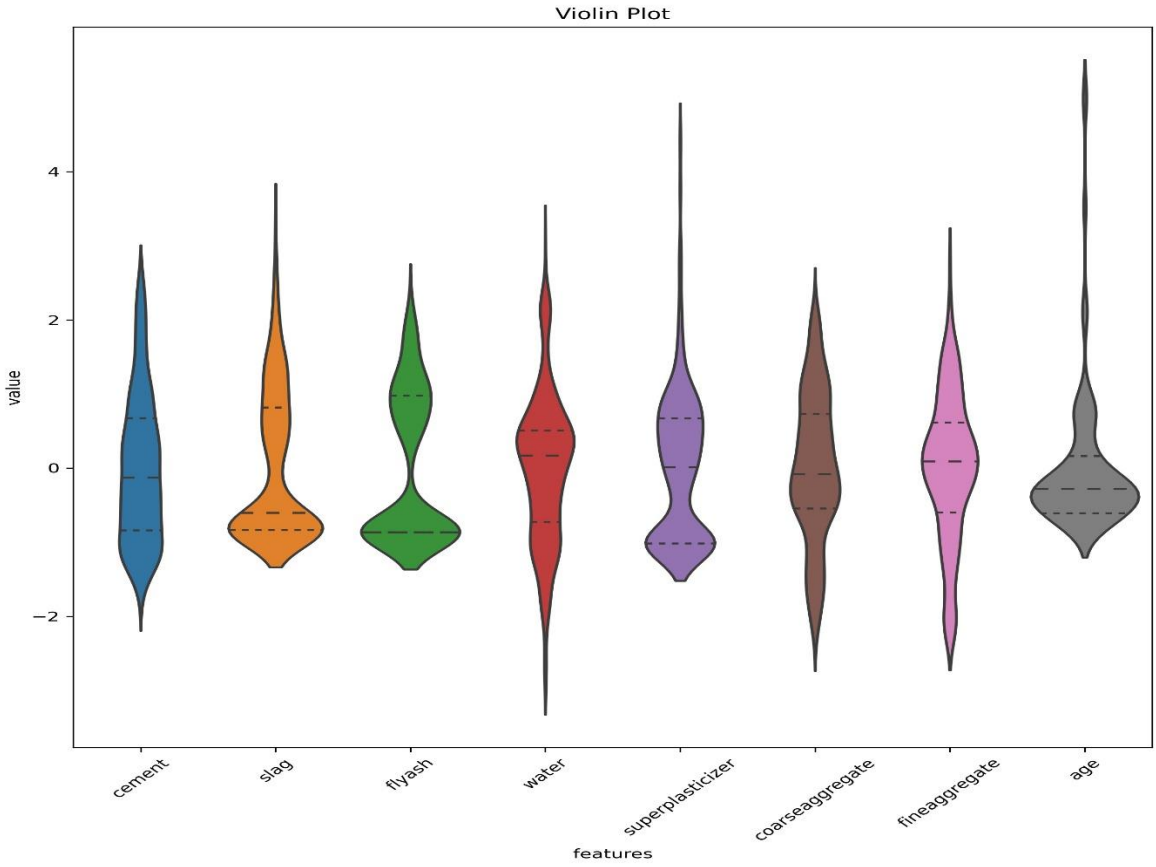
Question 1) Regression Decision Tree:

Source File: **q1.py**

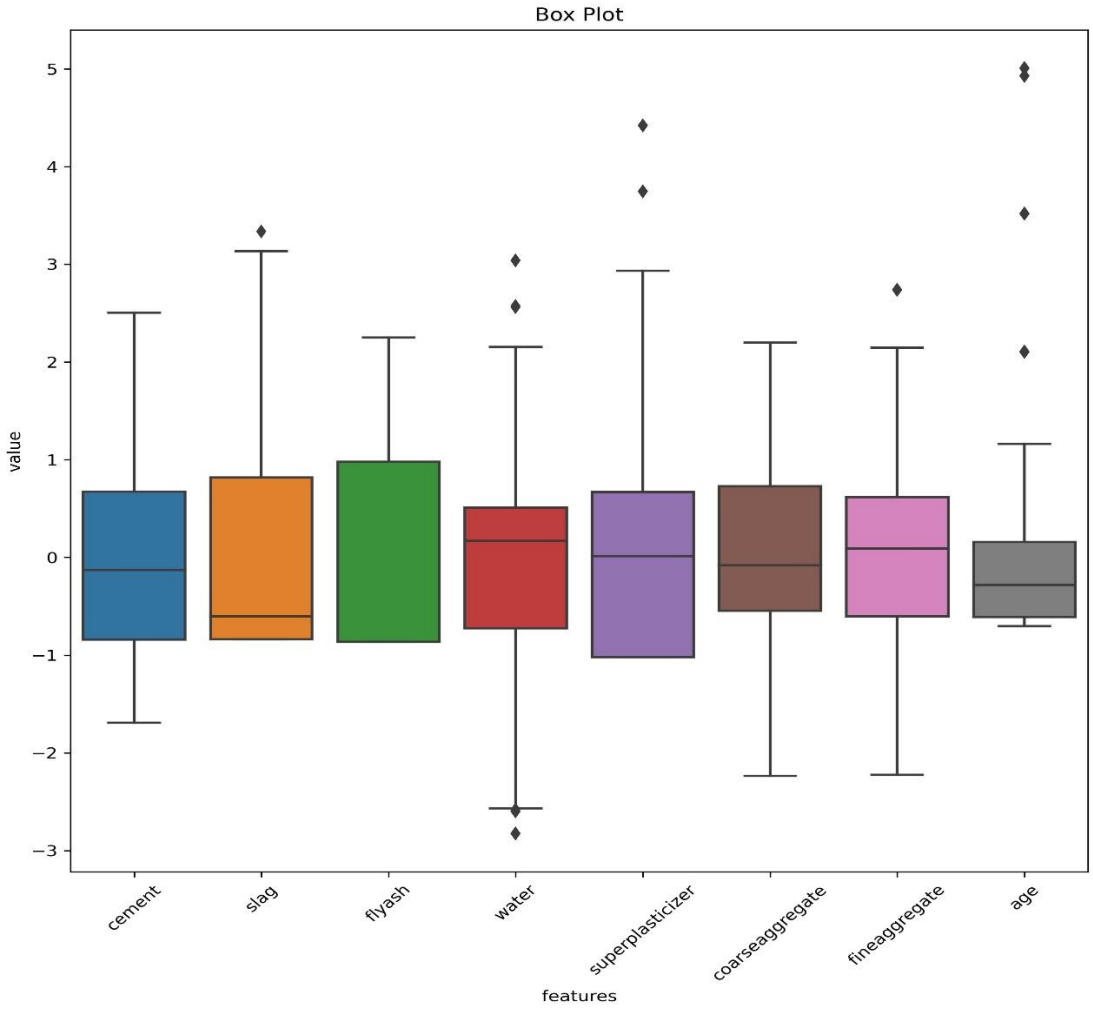
Exploratory Data Analysis and Data visualization:

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.

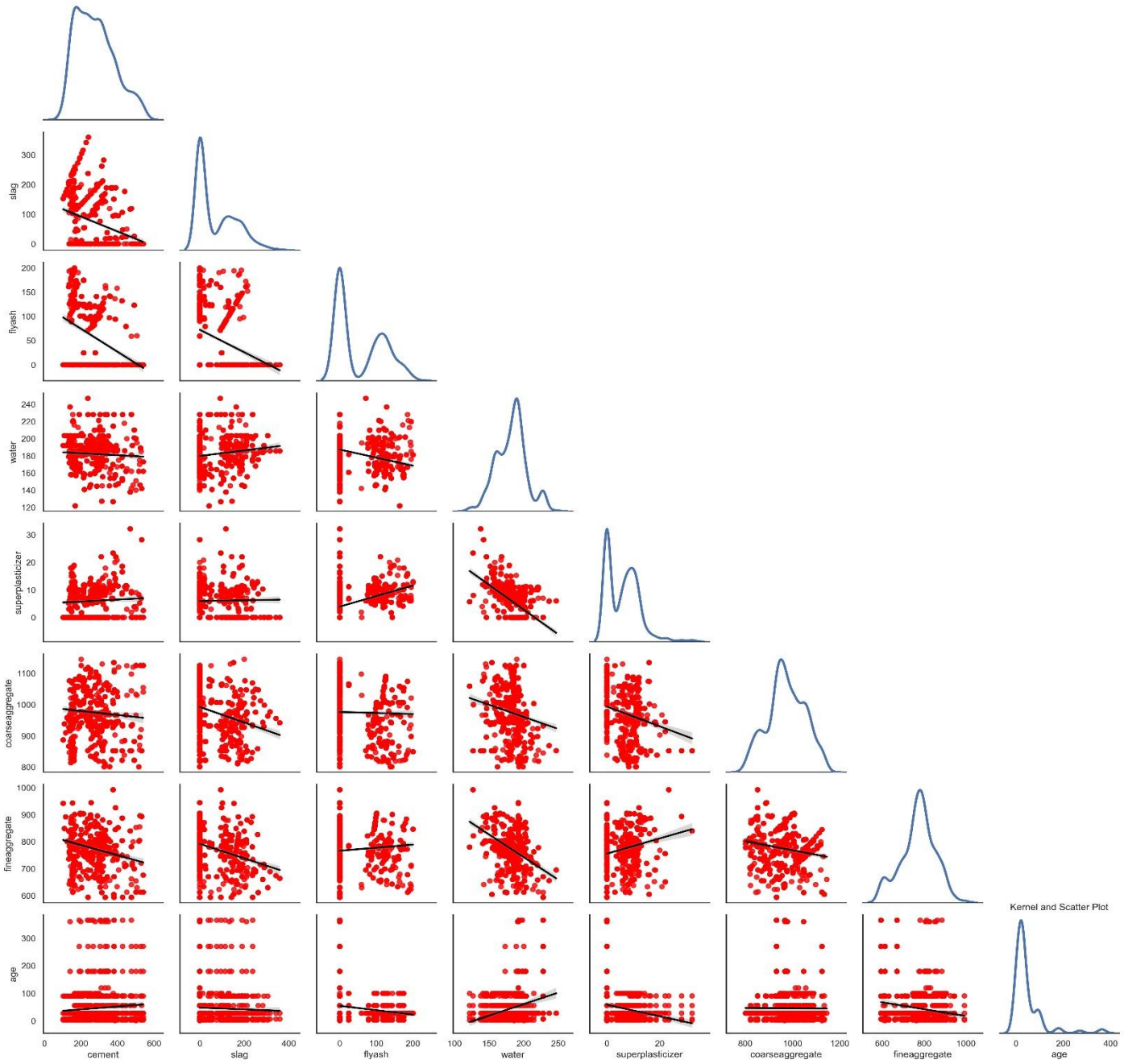
1. Violin Plot: depicts the distribution of different columns of the data along with 25%, 50%(median) and 75% percentile of the data.



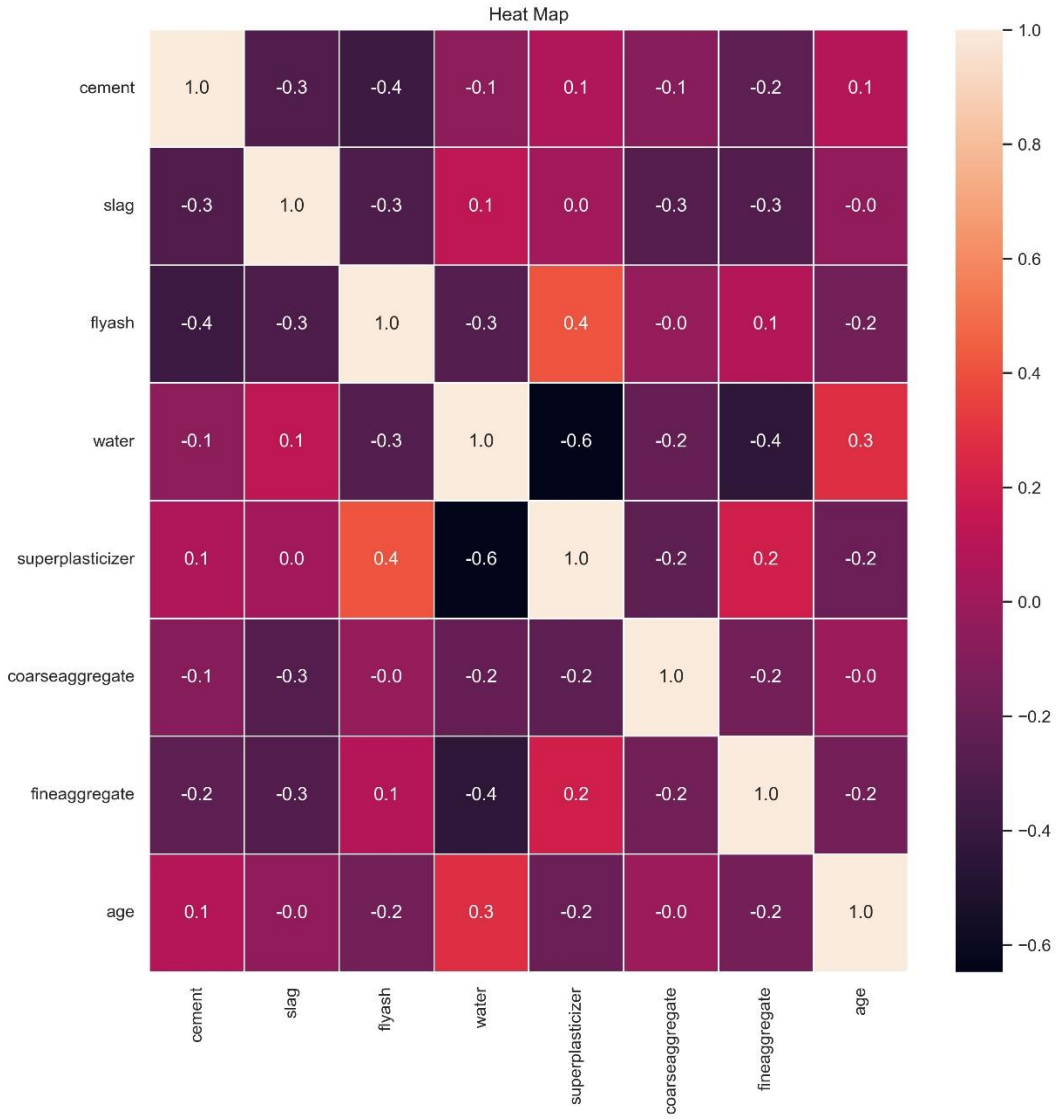
2. Box plot: depicts the spread of each column of data.



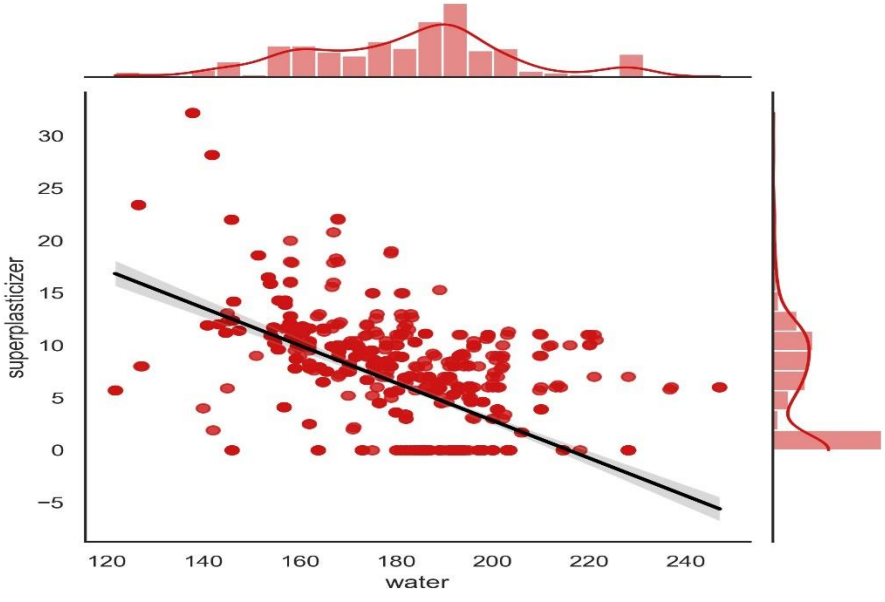
3. Kernel_Scatter_plot: depicts individual kernel density of features and scatter plots between every pair of features. Scatter plots help us to explore inter feature relationships.


Kernel and Scatter Plot
cement
slag
flyash
water
superplasticizer
coarseaggregate
fineaggregate
age

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



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



Class Defination:

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.

2. **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.

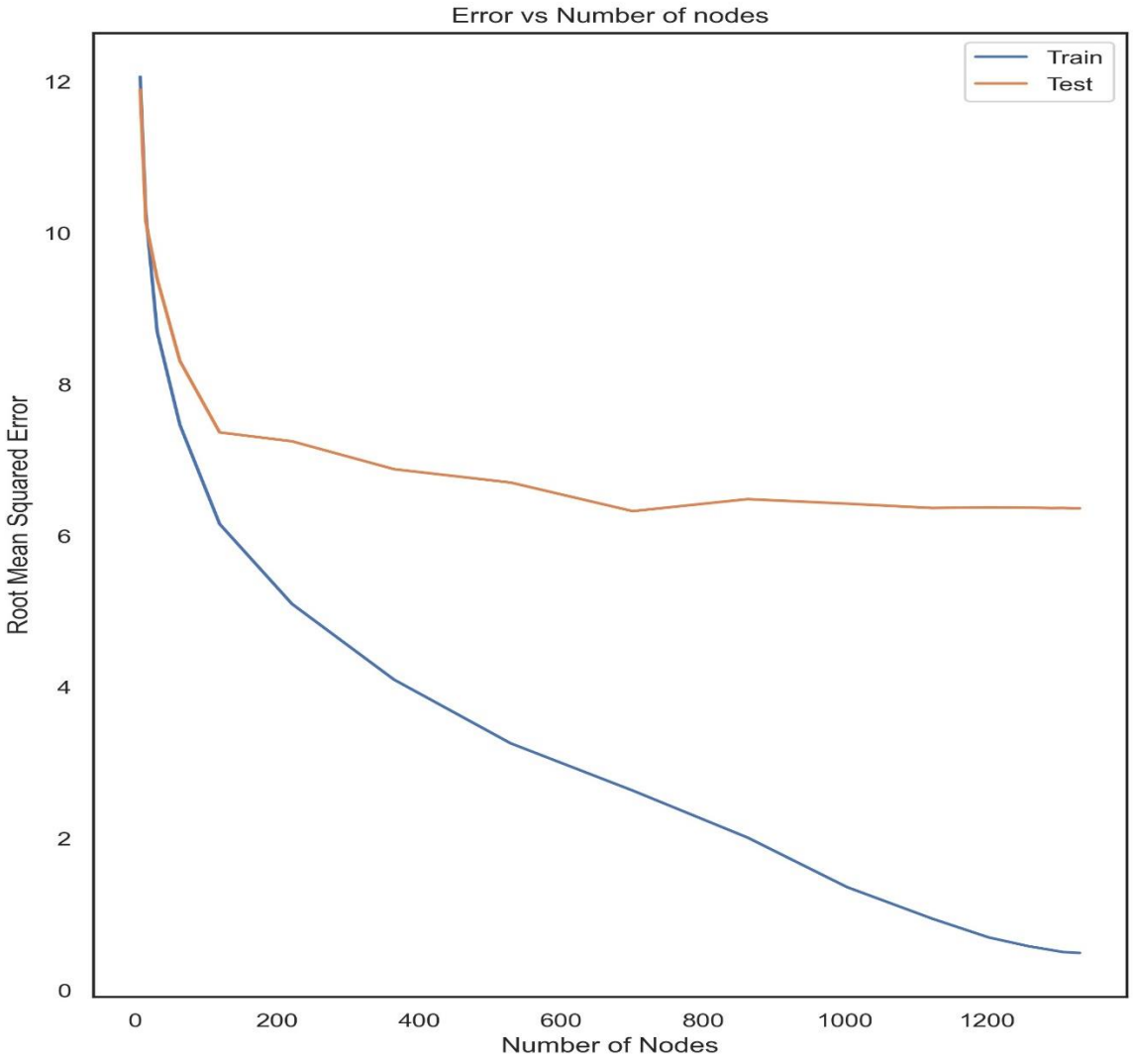
Helper functions:

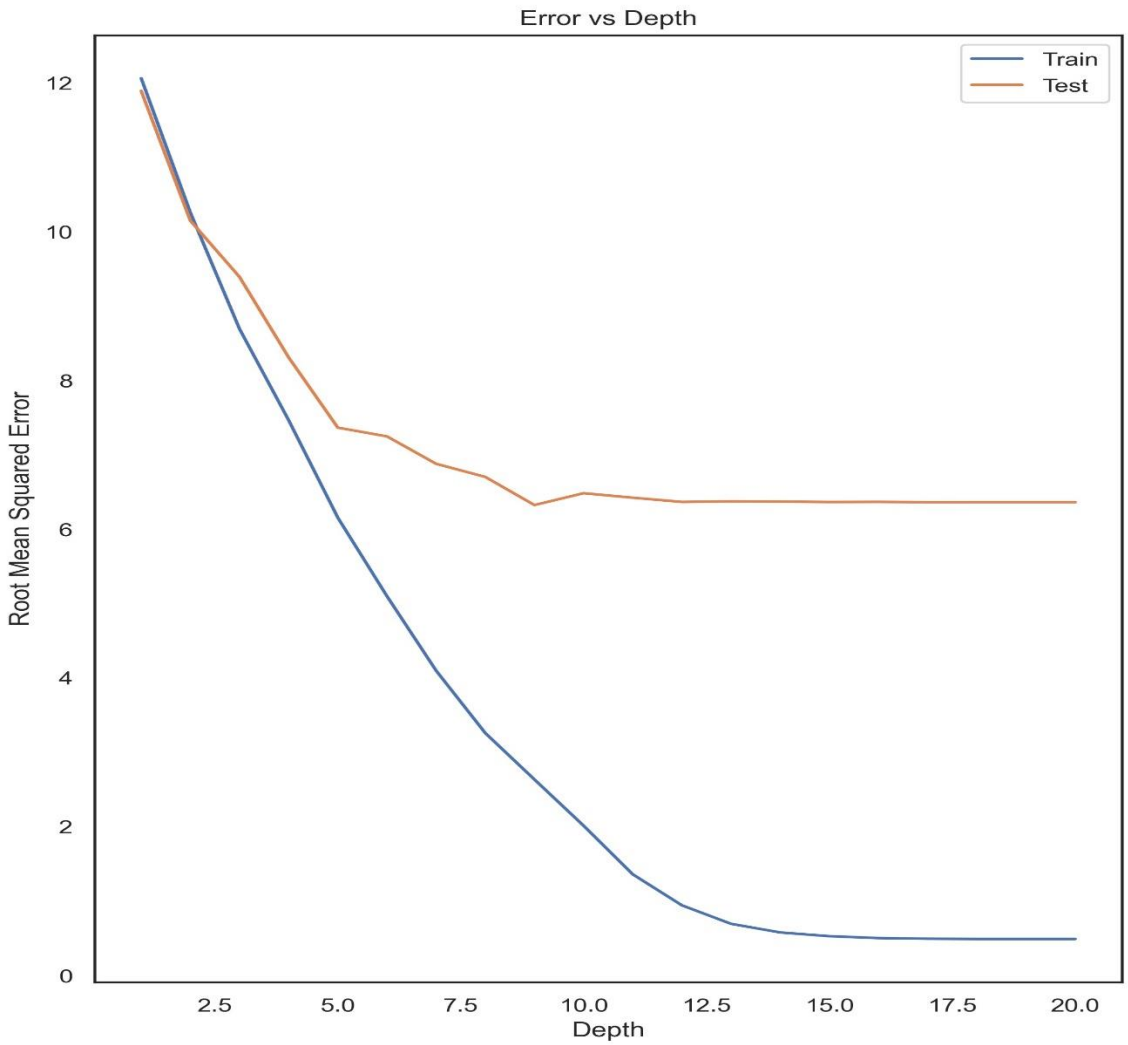
1) **rec_height**: recursively travels every node and appends the level of the node into the height list.
2) **find_height**: calls rec_height function and returns the maximum level of the list.
3) **changenode**: takes in input root, node whose value needs to be changed and new value of data members 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.

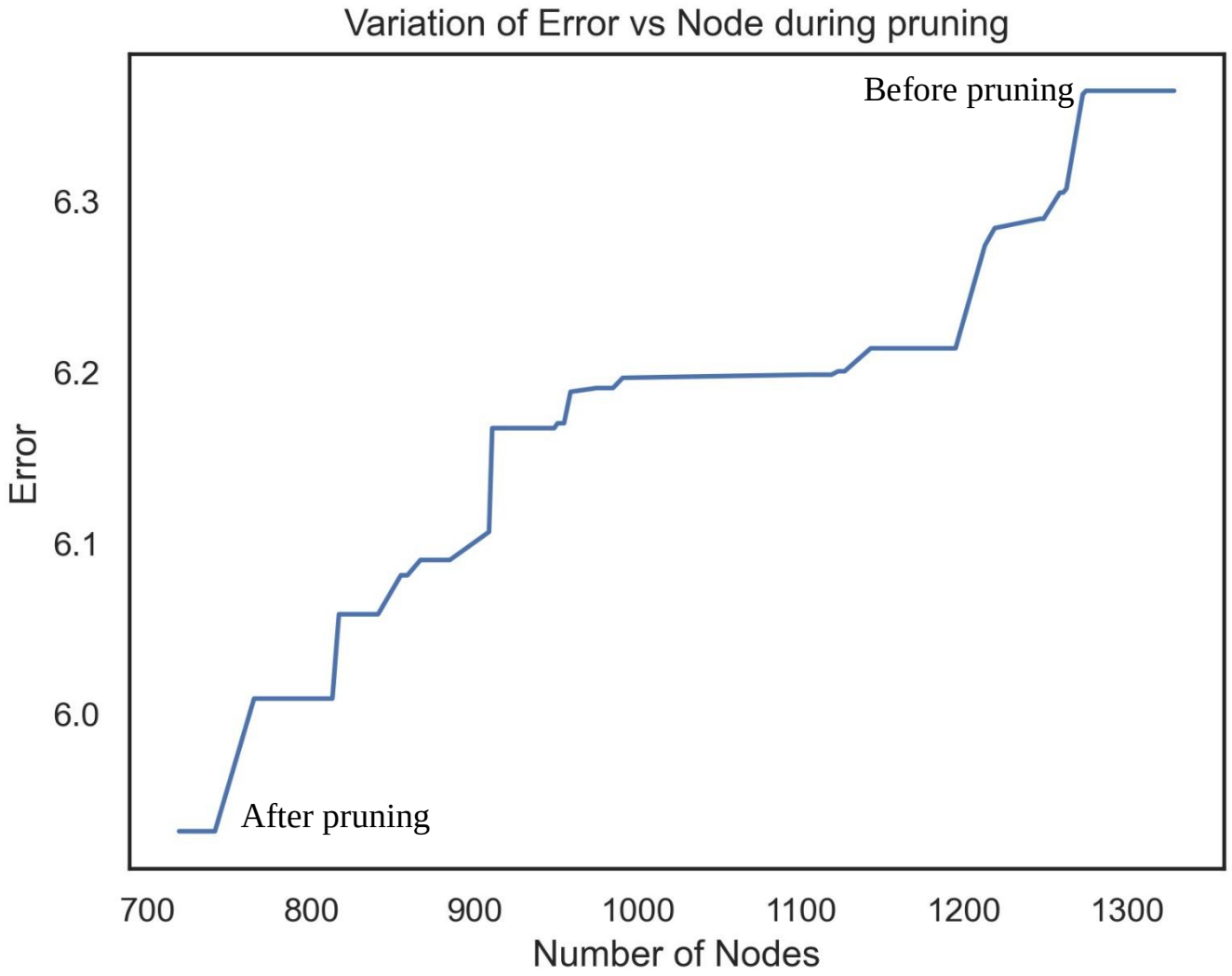
Model Training:

1. The data is shuffled randomly and the **random state is preserved** and a tree is trained on that data. The shuffled data is split into training and test data 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.





3. 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**.
4. 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.
5. This tree is then printed on a file named **decision_tree.txt**
6. We then perform post-pruning of this tree and store the number of nodes and error at every instance of pruning.
7. The number of nodes after pruning is **719** (very close to optimal number of nodes) while th test error has reduced to **5.93129378**.
8. The pruned tree is then printed in a file named as **pruned.txt.**


Variation of Error vs Node during pruning
Before pruning
After pruning
Error
6.3
6.2
6.1
6.0
700
800
900
1000
1100
1200
1300
Number of Nodes

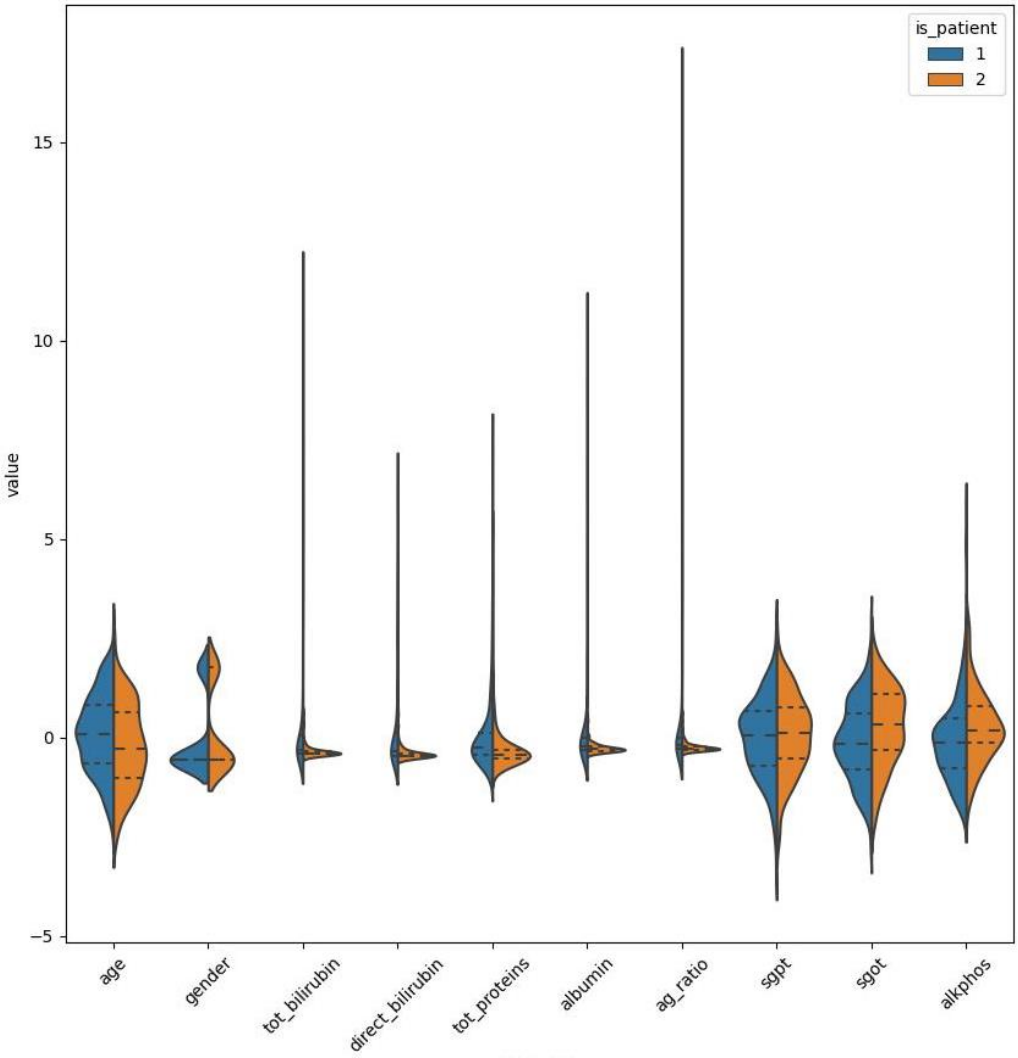
Question 2) Naive Bayesian Classifier

The files of the code: "**q2.py**"

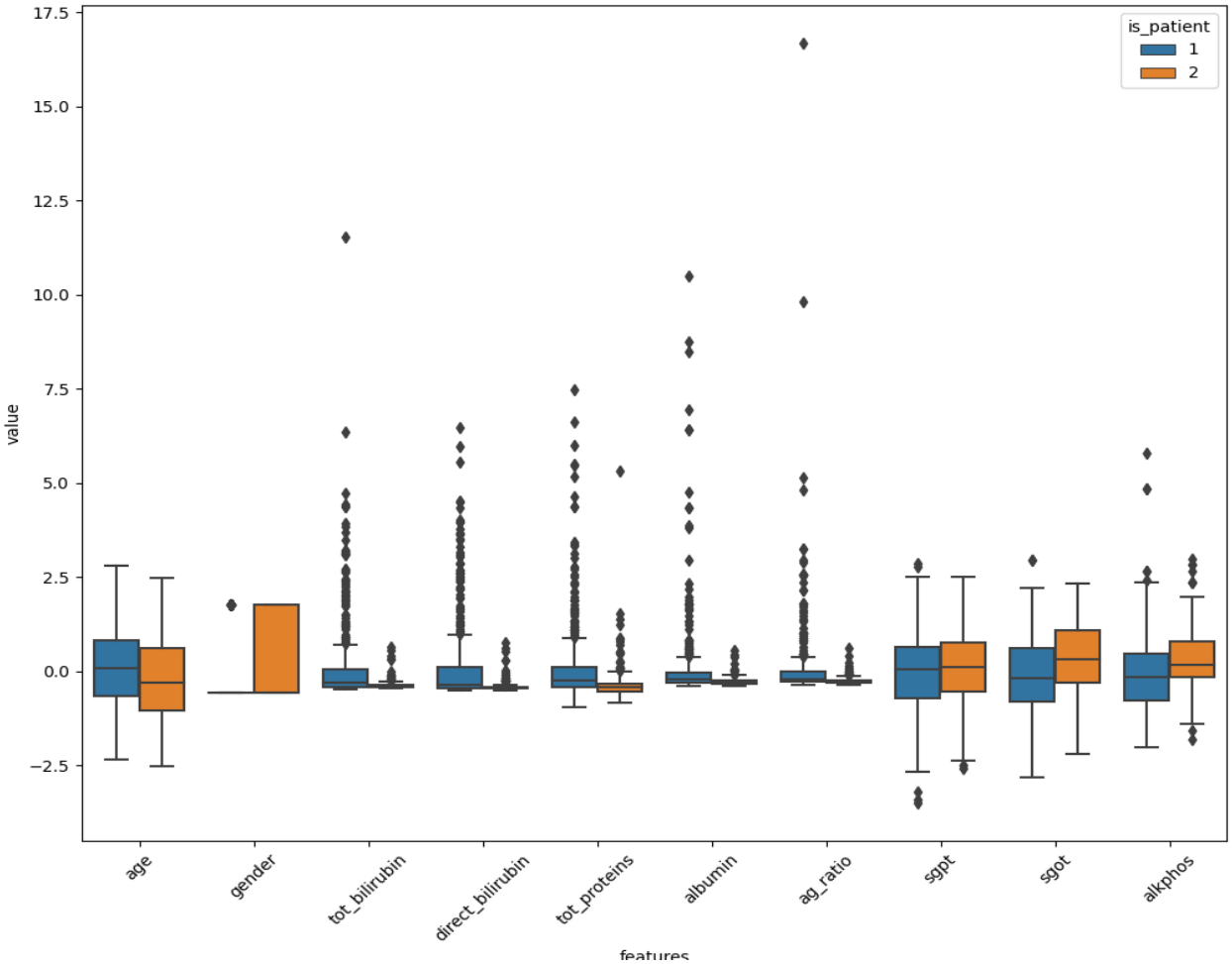
This file contains the whole python code of the model and data evaluation. It mainly has mainly following functions:

- Exploratory Data Analysis and Data visualization (EDA):
  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.

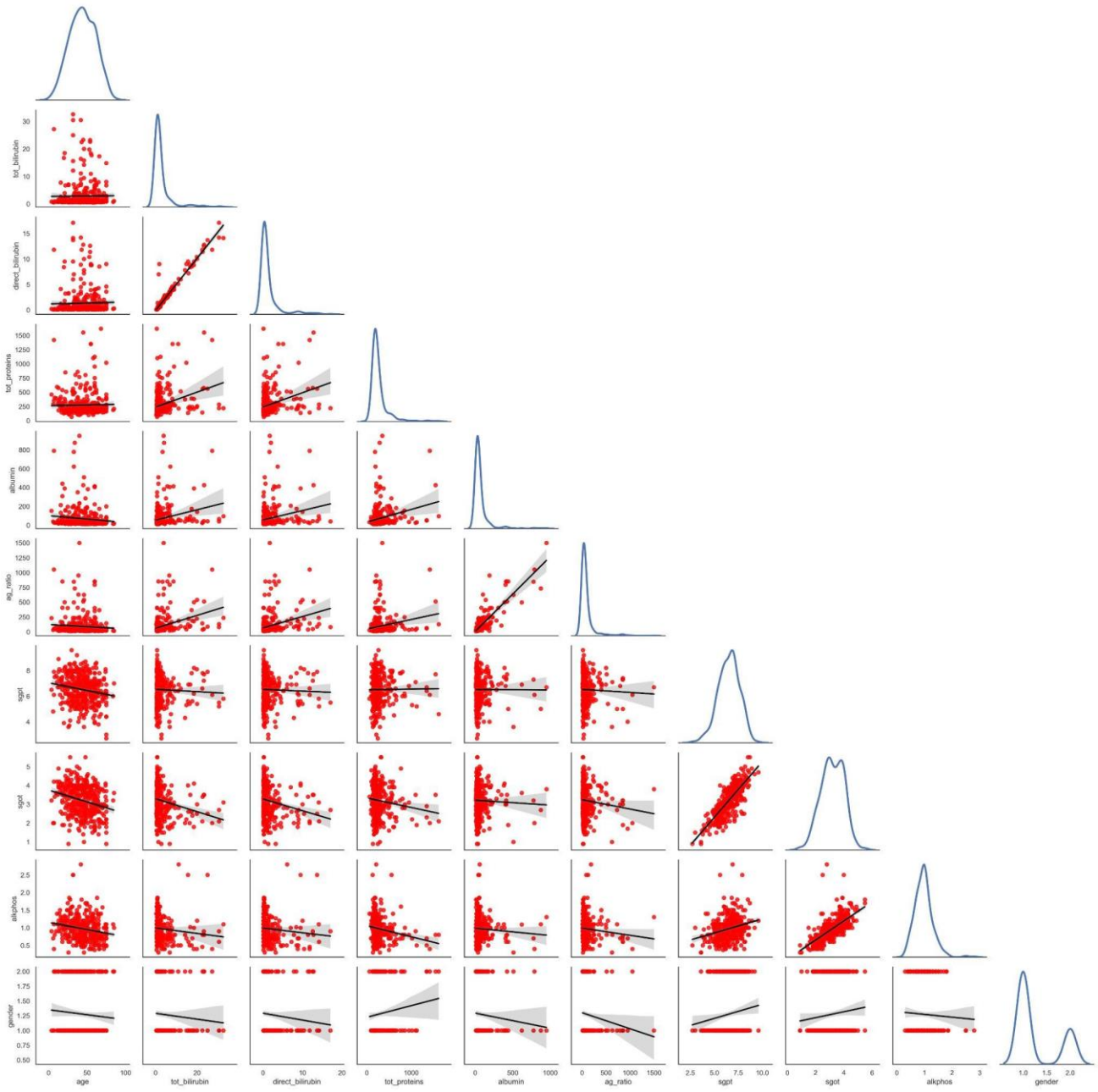
1. **Violin Plot**: depicts the distribution of different columns of the data along with 25%, 50%(median) and 75% percentile of the data.



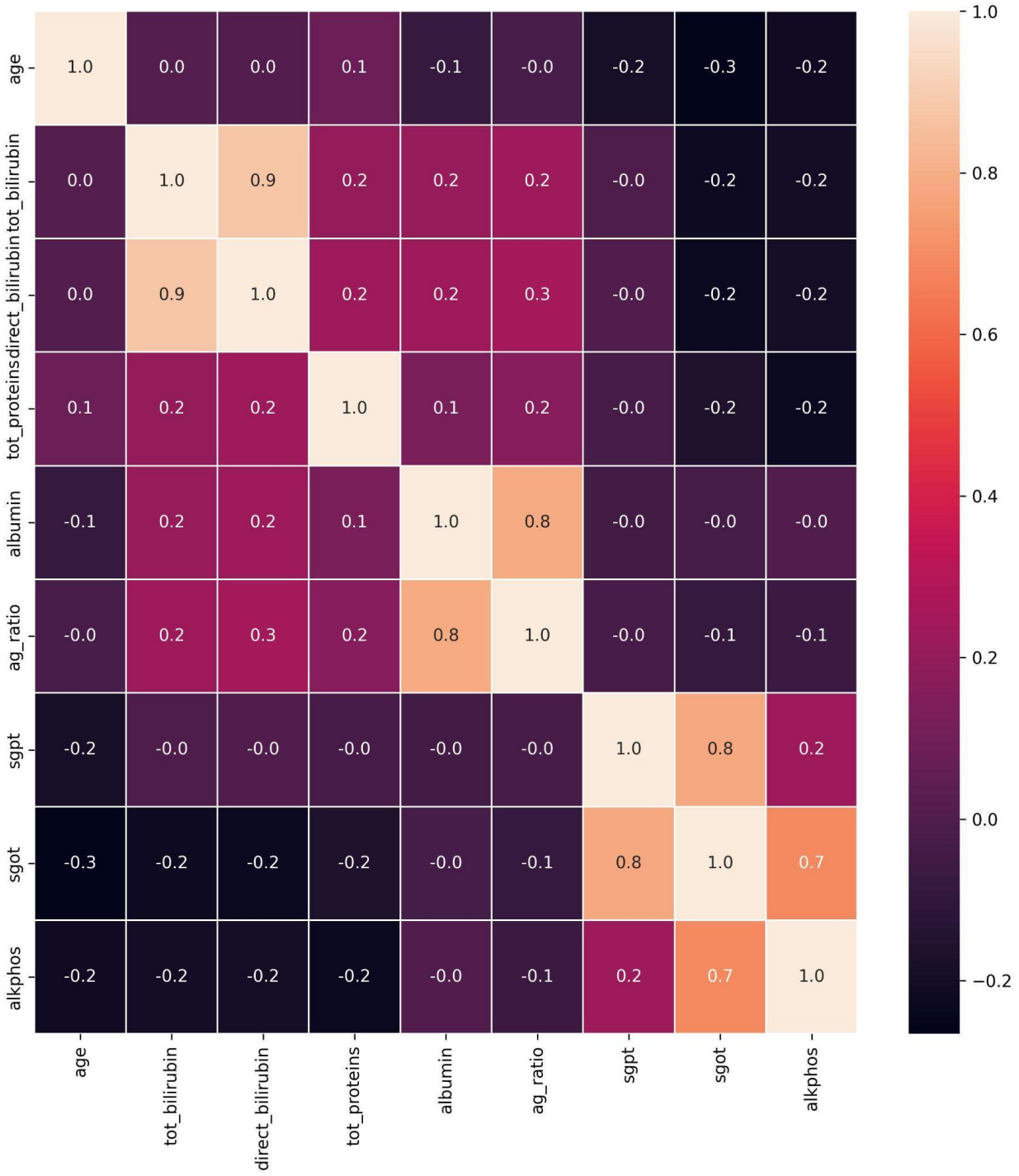
2. **Box plot**: depicts the spread of each column of data.



3. 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.



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



- category_encoding():
  This function encodes the categorical features to numerical values for easy processing of the data.

- remove_outliers():
  This function removes the outliers found in the data, by the formula given in the assignment.

$$(2 \times \mu + 5 \times \sigma)$$

The given data does not contain many outliers(10-15 samples out of 400+), and that too with more than one outlier features in the same sample is very rare(only 3-4 such samples). So, just removing 3-4 samples won't help much in terms of accuracy, so I have also removed the samples with only one outlier feature, so as to help in increasing accuracy.

- normal_distribution():
  This function calculates the probability, given the values of x, Mean and standard deviation, assuming 'x' follows Normal distribution. Using the following formula of PDF. It ignores the constant 1/sqrt(2*pi), because it will be the same for all, so it will be of no use while comparing the relative probabilities of different categories in the Naive Bayesian Classifier.

```
def normal_distribution(x , mean , sd):
    prob_density = (np.pi*sd) * np.exp(-0.5*(((x-mean)/sd)**2))
    return prob_density
```

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

- calc_proba():
  This function calculates the probability of the given data point, assuming that all the features are independent. It uses the following formula.

Naive Bayes classifier:

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i|v_j)$$

- initialise():
  It trains the model on the given training data and also does Laplace correction if 'laplace_factor' is non-zero. It calculates the probability of categorical features and the final category classifications. It also distributes the data into different categories and finds the means and standard deviations of respective features, so that the calc_proba() function can calculate the normal distribution probability of a given data point.
- classify():
  This function calls the calc_proba() function to calculate the relative probabilities of different categories and based on that it decides, the most likely category of the given sample.
- find_accuracy():
  This function takes as input the training and the testing data and laplace_factor. It calls the initialise() function to train the Naive Bayesian Classifier model on the training data and then calls the classify() function to classify the different samples of test data and calculates the accuracy of the model.
- five_fold_cross_validation():
  This function executes Five-fold cross-validation on the training set. It divides the training data into five equal parts and iterates on different parts, considering one part as a test set and the rest four parts as a training set. Then it computes the mean accuracy on all five folds and prints it.

Flow of Model Execution:

1. The data is shuffled randomly, and the random state is preserved. The shuffled data is split into training and test data (70-30).
2. Then the categorical variables are encoded into numbers for ease of processing. After this, the outliers are detected and removed from the data, by the formula given in the assignment.
3. Model executes the five-fold cross-validation on the training set and finds the mean accuracy of the five folds.
4. Then, the model is applied to the test (30%) data, and the final accuracy is found.
5. At last, the model executes Laplace correction, by passing the correction factor as 1, and then finds the accuracy on the test data.

Summary of Results:

1. We can see that the model finds 8-9 samples with outlier features, and it successfully removes those outliers.
2. The average accuracy of five-fold cross-validation is **0.701265823.**

```
Executing 5-fold cross validation on the training data...
Accuracy on fold 1(as test data) is: 0.6835443037974683
Accuracy on fold 2(as test data) is: 0.6962025316455697
Accuracy on fold 3(as test data) is: 0.7848101265822784
Accuracy on fold 4(as test data) is: 0.5949367088607594
Accuracy on fold 5(as test data) is: 0.7468354430379747
Five Fold cross validation mean accuracy =  0.7012658227848101
```

3. The test(30%) data accuracy is **0.73142857.**

```
Test data accuracy =  0.7314285714285714
Test data accuracy(with Laplace Correction) =  0.7314285714285714
```

4. After Laplace correction, the accuracy is **0.73142857**. We can easily see that the accuracy does not change; this is because there are no zero probability features in the dataset. If there would have been categorical features with zero probability, then the Laplace correction would have changed the accuracy.