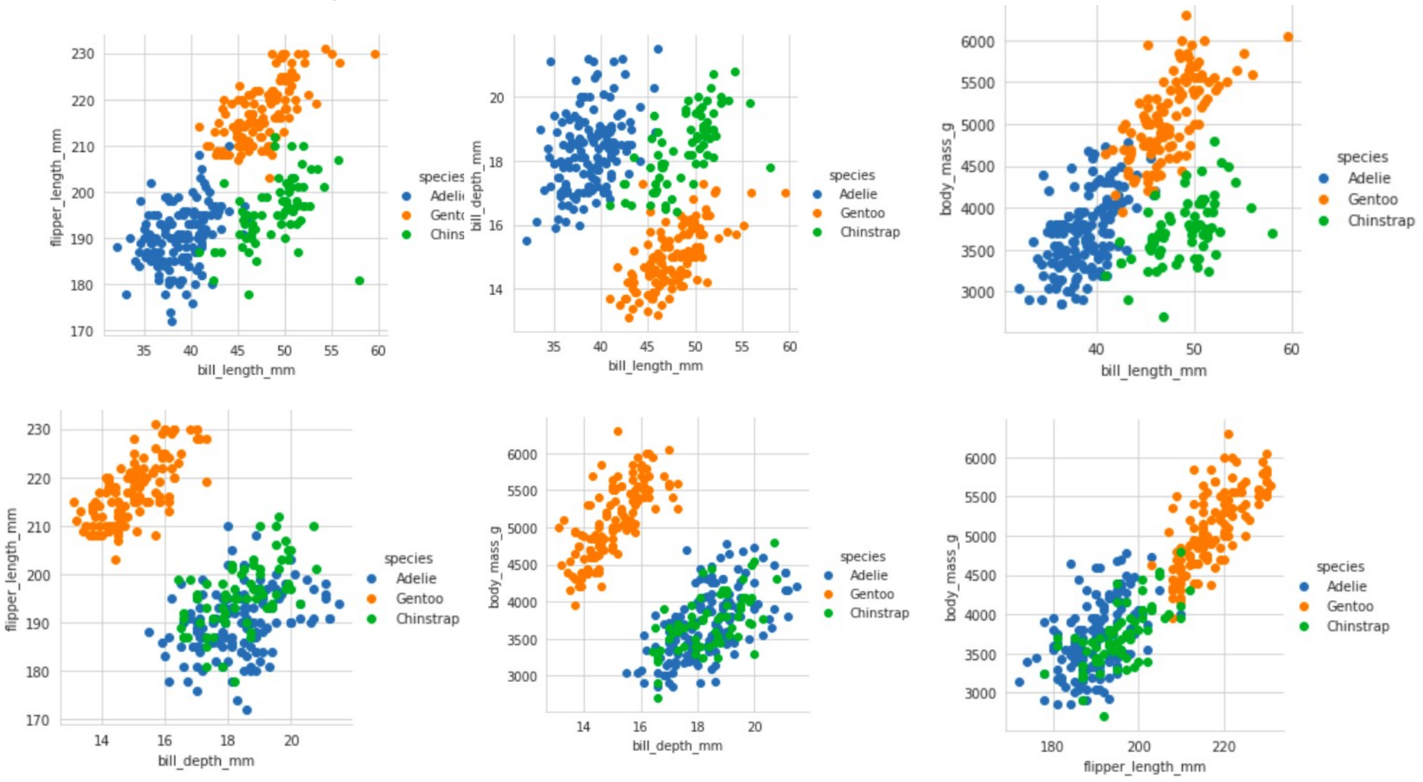
PRML Lab-2

Abu Shahid (B20CS003)- [Link to the File]

Question 1

Subtask 1

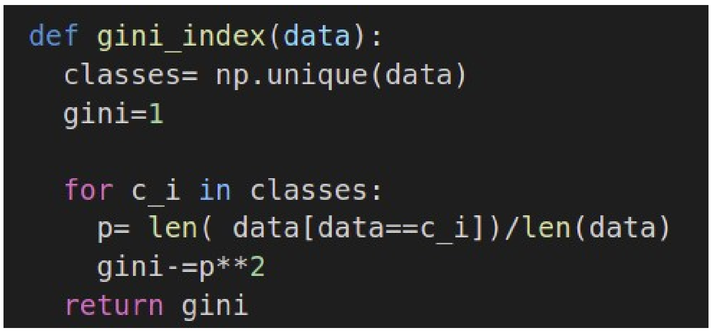
- Values of few features were missing and therefore, corresponding rows were dropped.
- Our data was 7-dimensional and therefore visualization was not straighforward.
- Just to get an insight of the classes, various features were plotted against one another and visualized one by one.



- As seen by the previous plot, even by usuing two features, the classes are quite separable for few select feature pairs and we can expect to have a well defined decision boundary. (Such an exercise overkill here, would have been helpful for feature selection to train our model had the dataset been enormous)
- On inspection of the dataset, year, island and sex were found to be categorical labels and one-hot encoded direclty(skipping the intermediate step of converting them to categorical codes). Sex was also one-hot-encoded. The species being the target variable was left untouched.
- The dataset was then split into 80:20 ratio of train and test set.

Subtask-2

- Implementing gini index calculation as per: gini index= 1- summation(p_i^2). (the function for calculation of information gain was not coded explicitly)

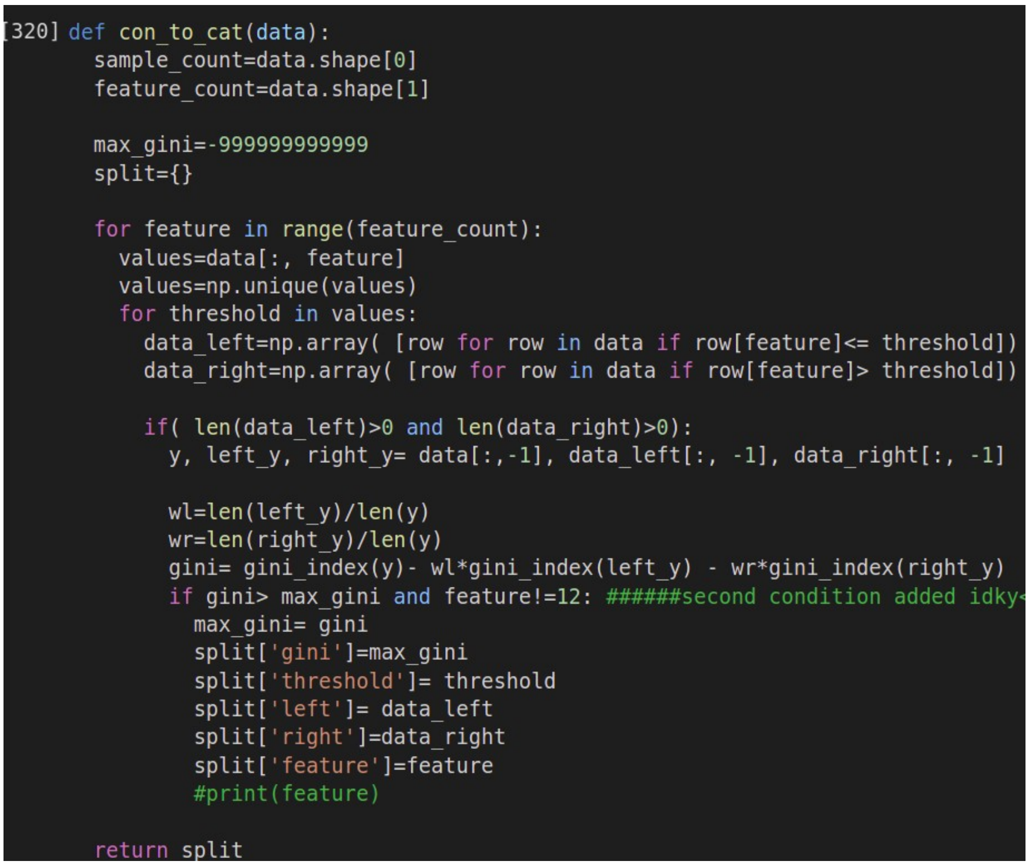


```
def gini_index(data):
  classes= np.unique(data)
  gini=1

  for c_i in classes:
    p= len( data[data==c_i])/len(data)
    gini-=p**2
  return gini
```

Subtask-3

- **con_to_cat()** function was declared and the pseudocode of the same is:

```
> def con_to_cat( data):
> split={} //dictionary -to store the left and right data after the split is made
                        -to store the feature which was analysed for the split
                        -the value of the feature(threshold) used for splitting
                        -the resulting gini index for the split
> a nested iterater to loop through all the features and all the values of the features, calculating all
the possible splits and returninng the case with best split.
```



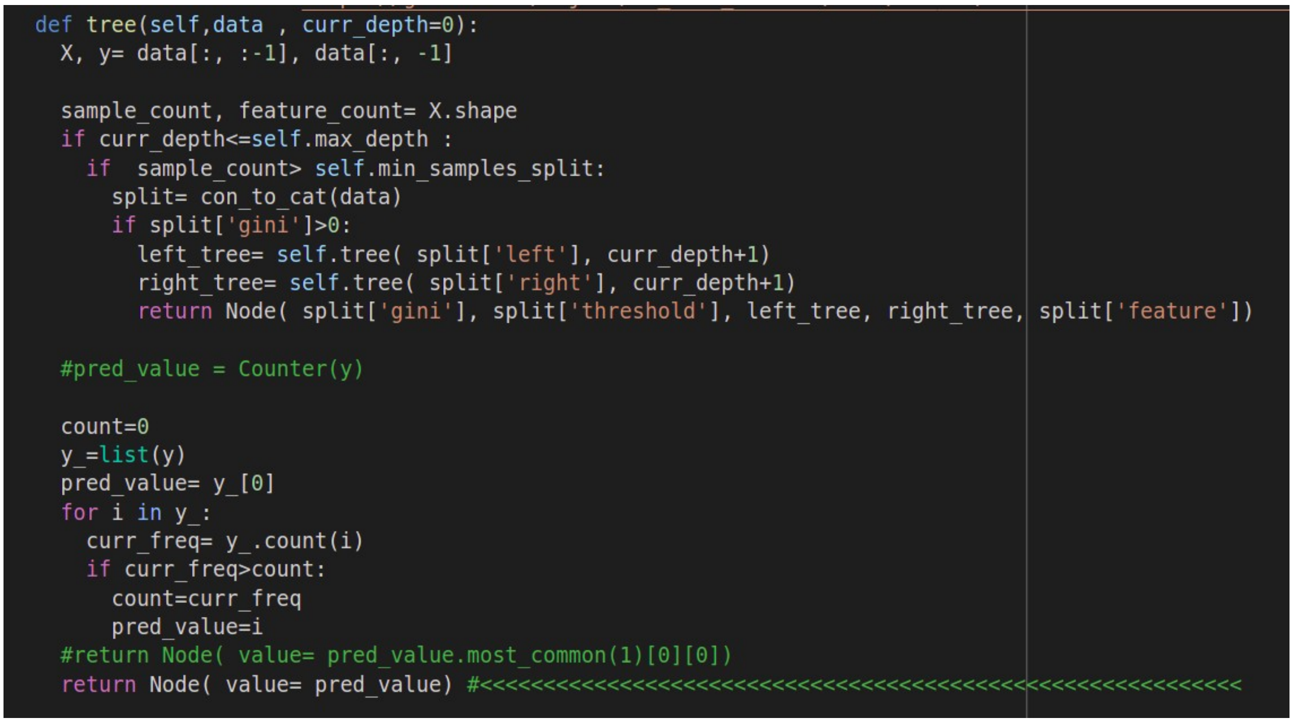
```
[320] def con_to_cat(data):
        sample_count=data.shape[0]
        feature_count=data.shape[1]

        max_gini=-999999999999
        split={}

        for feature in range(feature_count):
          values=data[:, feature]
          values=np.unique(values)
          for threshold in values:
            data_left=np.array( [row for row in data if row[feature]<= threshold])
            data_right=np.array( [row for row in data if row[feature]> threshold])

            if( len(data_left)>0 and len(data_right)>0):
              y, left_y, right_y= data[:,-1], data_left[:, -1], data_right[:, -1]

              wl=len(left_y)/len(y)
              wr=len(right_y)/len(y)
              gini= gini_index(y)- wl*gini_index(left_y) - wr*gini_index(right_y)
              if gini> max_gini and feature!=12: ######second condition added idky
                max_gini= gini
                split['gini']=max_gini
                split['threshold']= threshold
                split['left']= data_left
                split['right']=data_right
                split['feature']=feature
                #print(feature)

        return split
```

Subtask 4 and 5

- Our **con_to_cat()** function itself gets the attribute for the best split and then make the split.
- In our implementation, **con_to_cat()** is continously called recursively called and splits are made till
  - data to be split is > min_samples_leaf
  - depth < max_depth
  - and information gain for the split> 0

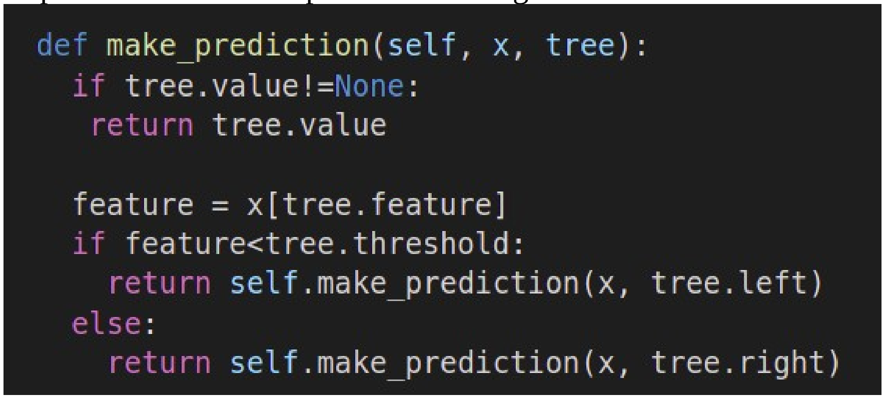
- This was majorly carried out by the **tree()** function which gets automatically called on calling **fit()**.
- The **tree()** implements the **con_to_cat()** and **gini()** function and makes the decision tree as per above conditions.

- Referenced from: https://github.com/Suji04/ML_from_Scratch/blob/master/decision%20tree%20classification.ipynb



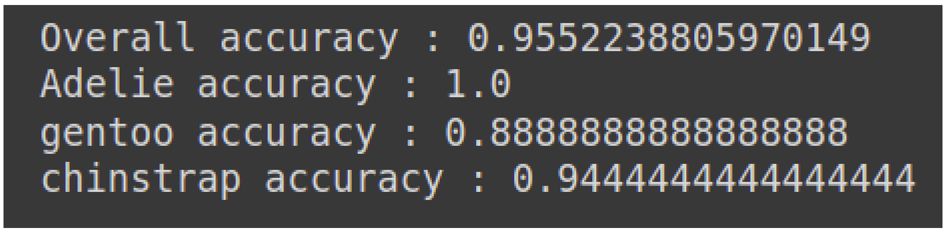
```
def tree(self,data , curr_depth=0):
  X, y= data[:, :-1], data[:, -1]

  sample_count, feature_count= X.shape
  if curr_depth<=self.max_depth :
    if  sample_count> self.min_samples_split:
      split= con_to_cat(data)
      if split['gini']>0:
        left_tree= self.tree( split['left'], curr_depth+1)
        right_tree= self.tree( split['right'], curr_depth+1)
        return Node( split['gini'], split['threshold'], left_tree, right_tree, split['feature'])

  #pred_value = Counter(y)

  count=0
  y_=list(y)
  pred_value= y_[0]
  for i in y_:
    curr_freq= y_.count(i)
    if curr_freq>count:
      count=curr_freq
      pred_value=i
  #return Node( value= pred_value.most_common(1)[0][0])
  return Node( value= pred_value) #<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

- If a split is not possible, value of the leaf is decided using majority voting.
- **Subtask 6:** a fuction was made inside the DTC() class which takes the parameters of X_new and makes predictions based of previous learnings.



```
def make_prediction(self, x, tree):
  if tree.value!=None:
   return tree.value

  feature = x[tree.feature]
  if feature<tree.threshold:
    return self.make_prediction(x, tree.left)
  else:
    return self.make_prediction(x, tree.right)
```

- **Subtask-7:** Finally our model was evaluated using the test_set that was kept aside after the initial train_test_split.
  - Model was fitted using hyperparameters: max_depth=7; min_samples_split=2
  - **Note: no extensive hyperparameters tuning was done as it was not asked in the problem statement and that there is still scope for the model to be tweaked.**



```
Overall accuracy : 0.9552238805970149
Adelie accuracy : 1.0
gentoo accuracy : 0.8888888888888888
chinstrap accuracy : 0.9444444444444444
```

- Overall accuracy sits at 95.52% and can be pushed further using hyperparameter tunng
  - Adelie accuracy: 100%
  - Chinstrap accuracy: 94.44%
  - Gentoo accuracy: 88.88%

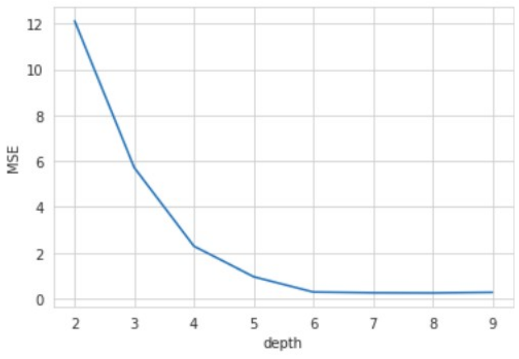
Question 2

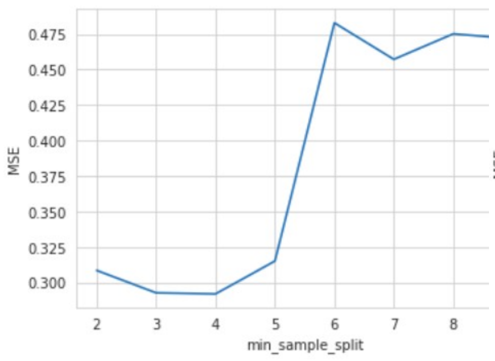
Subtask 1

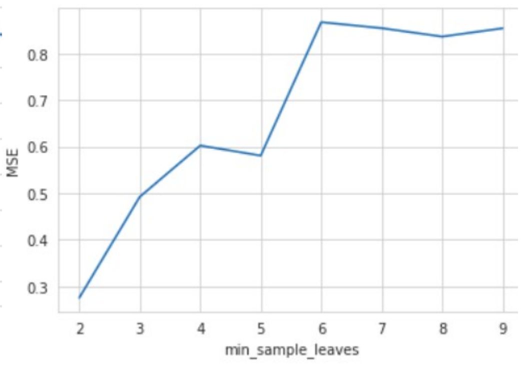
- The data was analysed and it was a complete dataset with no missing values.
- Next we calculated the correlation of different features with our target variable Y1.
    - `X5 0.889431, X1 0.622272, X3 0.455671, X7 0.269841, X8 0.087368, X6 -0.002587, X2 -0.658120, X4 -0.861828`
- Such an exercise, here however unnecessary would help us in feature selection had the dataset been gigantic.
- Dataset source was inspected and it was found that **X6** was a categorical label and therefore was one-hot-encoded. Data was then modified with the one-hot labels appended and original X6 was dropped.
- Finally data was split into train and test set, first in the ration 80:20
- Then it was further split into train and validation set, in the ration 87.5 : 12.5

Subtask 2

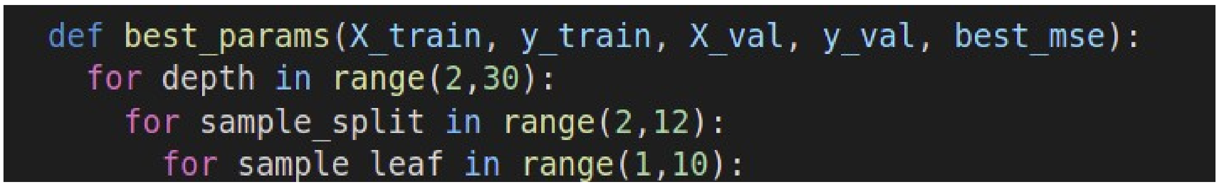
- To get an intuition of how our model behaves on changing hyperparameters; we vary one hyperparameter at a time, training on the training set and calculating MSE on the validation set. The results are plotted.



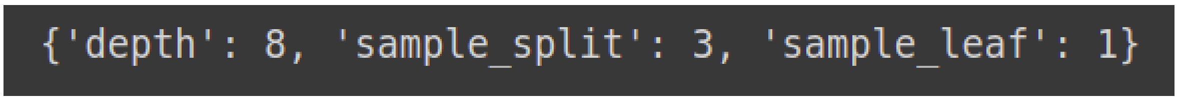




- On a rough note MSE increases with increase in **min_sample_split** and **min_sample_leaves** and deceases with increases in **max_depth.**
- This is so because, as the depth inceases, our model can made diverse threshold conditions to fit more of the data and define complex decision boundaries.
- And as the min_sample_split decreases, our model has more freedom to reduce the entopy of a leaf and increase the information gain by further splitting.
- To find the best combination of parameters, a function is defined with trains our model over varying hyperparameters and returns the combination which gives the best MSE.



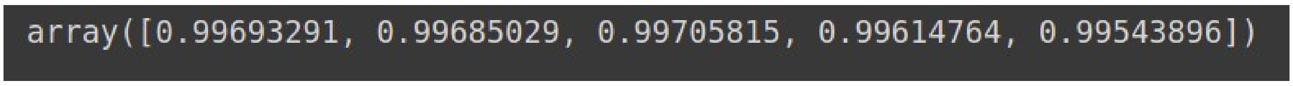
```
def best_params(X_train, y_train, X_val, y_val, best_mse):
  for depth in range(2,30):
    for sample_split in range(2,12):
      for sample_leaf in range(1,10):
```



```
{'depth': 8, 'sample_split': 3, 'sample_leaf': 1}
```

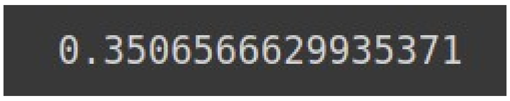
Subtask 3

- Using the parameters we got from the previous exercise, we train a DecisionTreeRegressor using the same parameters and to validate its performance, we perform k-fold cross validation. (k=5). The results are as:



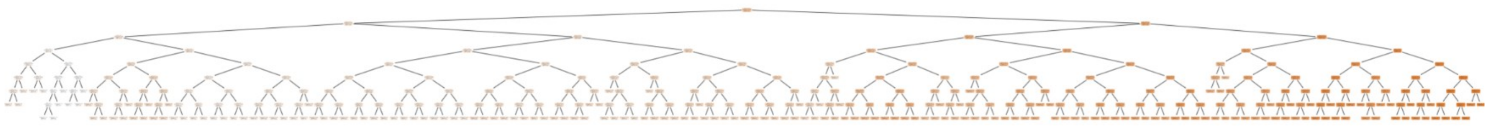
```
array([0.99693291, 0.99685029, 0.99705815, 0.99614764, 0.99543896])
```

- Not only are the validation scores consistent, ensuring that our model is not over-fitting but also they are high: ensuring that our model has indeed learned.
- Finally we calculate the mean squared error between the predicted and the ground-truth values in the test data for our best model.

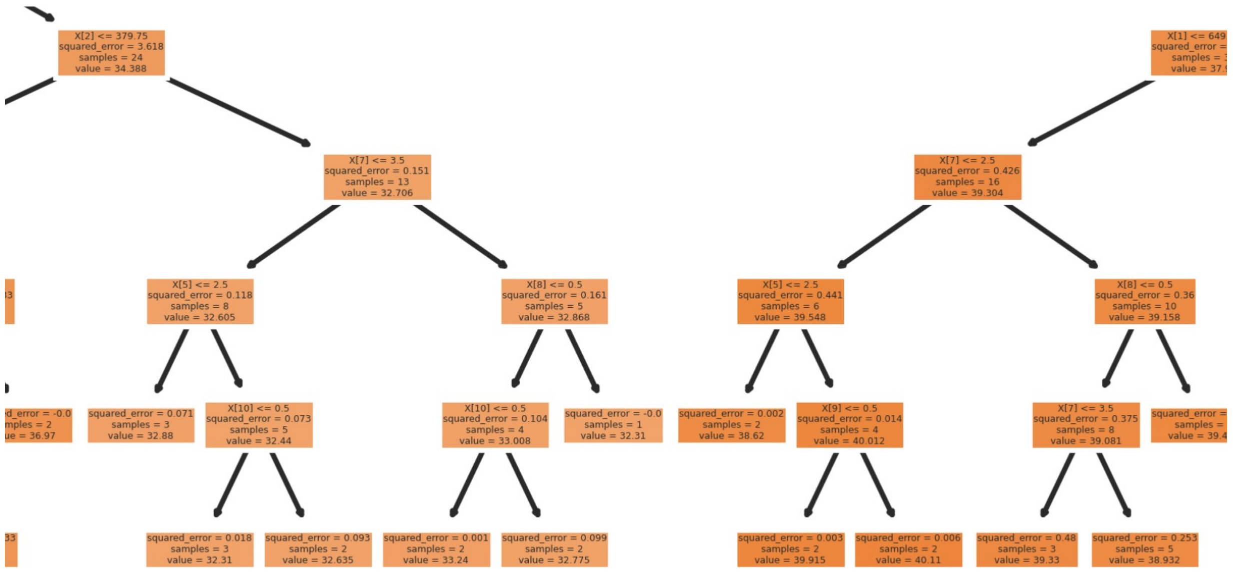


```
0.3506566629935371
```

- At last, tree was plotted:



and some decision nodes for the data that is central in our dataset;



X[2] <= 379.75
squared_error = 3.618
samples = 24
value = 34.388
X[1] <= 649
squared_error =
samples =
value = 37.
X[7] <= 3.5
squared_error = 0.151
samples = 13
value = 32.706
X[7] <= 2.5
squared_error = 0.426
samples = 16
value = 39.304
X[5] <= 2.5
squared_error = 0.118
samples = 8
value = 32.605
X[8] <= 0.5
squared_error = 0.161
samples = 5
value = 32.868
X[5] <= 2.5
squared_error = 0.441
samples = 6
value = 39.548
X[8] <= 0.5
squared_error = 0.36
samples = 10
value = 39.158
d_error = -0.0
mples = 2
ue = 36.97
squared_error = 0.071
samples = 3
value = 32.88
X[10] <= 0.5
squared_error = 0.073
samples = 5
value = 32.44
X[10] <= 0.5
squared_error = 0.104
samples = 4
value = 33.008
squared_error = -0.0
samples = 1
value = 32.31
squared_error = 0.002
samples = 2
value = 38.62
X[9] <= 0.5
squared_error = 0.014
samples = 4
value = 40.012
X[7] <= 3.5
squared_error = 0.375
samples = 8
value = 39.081
squared_error =
samples =
value = 39.4
squared_error = 0.018
samples = 3
value = 32.31
squared_error = 0.093
samples = 2
value = 32.635
squared_error = 0.001
samples = 2
value = 33.24
squared_error = 0.099
samples = 2
value = 32.775
squared_error = 0.003
samples = 2
value = 39.915
squared_error = 0.006
samples = 2
value = 40.11
squared_error = 0.48
samples = 3
value = 39.33
squared_error = 0.253
samples = 5
value = 38.932