Regression Trees

Prof Wells

STA 295: Stat Learning

April 16th, 2024

Outline

- Introduction to Decision Trees
- Discuss Theory and Algorithm for Decision Trees
- Describe the Pruning Algorithm as means of improving RMSE
- Implement Decision Trees in R

Section 1

Decision Trees

My favorite animal is . . .

My favorite animal is . . .

Let's start with a game.

My favorite animal is . . .

Let's start with a game.

As a class, you may ask me up to **six** yes-or-no questions which I will truthfully answer, in order to learn more about my favorite animal.

After 6 questions, you may submit your guess for my favorite animal on a slip of paper.

- Don't look ahead on the slides!

My favorite animal is . . .

Let's start with a game.

As a class, you may ask me up to **six** yes-or-no questions which I will truthfully answer, in order to learn more about my favorite animal.

After 6 questions, you may submit your guess for my favorite animal on a slip of paper.

- Don't look ahead on the slides!

This slide intentionally blank

Don't look ahead at the next slide

My favorite animal is . . . the Pacific Tree Frog

Guessing Algorithm

- The guessing algorithm determines a (partial) decision tree for classification

Guessing Algorithm

- The guessing algorithm determines a (partial) decision tree for classification
  - Each Yes/No questions represent a branching point or node
  - The final guess represents the prediction
  - The 6 question limit restricts the depth of the tree
  - I am the test observation
  - What is the training data?

Guessing Algorithm

- The guessing algorithm determines a (partial) decision tree for classification
  - Each Yes/No questions represent a branching point or node
  - The final guess represents the prediction
  - The 6 question limit restricts the depth of the tree
  - I am the test observation
  - What is the training data?
- What makes an effective question?

Guessing Algorithm

- The guessing algorithm determines a (partial) decision tree for classification
  - Each Yes/No questions represent a branching point or node
  - The final guess represents the prediction
  - The 6 question limit restricts the depth of the tree
  - I am the test observation
  - What is the training data?
- What makes an effective question?
  - Separates data into roughly equal sizes
  - Data in each group are relatively similar
  - Later questions should be based on answers to earlier questions.
  - Early questions are general, later questions are specific.

Section 2

Regression Trees

Regression Trees

Basic regression trees partition data into smaller groups that are more homogeneous with respect to predictors.

Regression Trees

Basic regression trees partition data into smaller groups that are more homogeneous with respect to predictors.

- They then make predictions based on average value of response in each group

Regression Trees

Basic regression trees partition data into smaller groups that are more homogeneous with respect to predictors.

- They then make predictions based on average value of response in each group

The most common technique is the Classification and Regression Tree (CART) method, which uses **Recursive Binary Splitting**

Regression Trees

Basic regression trees partition data into smaller groups that are more homogeneous with respect to predictors.

- They then make predictions based on average value of response in each group

The most common technique is the Classification and Regression Tree (CART) method, which uses **Recursive Binary Splitting**

1. The method begins with the entire data set S and searches every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squared error:

$$\text{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

Regression Trees

Basic regression trees partition data into smaller groups that are more homogeneous with respect to predictors.

- They then make predictions based on average value of response in each group

The most common technique is the Classification and Regression Tree (CART) method, which uses **Recursive Binary Splitting**

1. The method begins with the entire data set S and searches every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squared error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. The method then repeats step 1 for each of the two groups S_1 and S_2.

Regression Trees

Basic regression trees partition data into smaller groups that are more homogeneous with respect to predictors.

- They then make predictions based on average value of response in each group

The most common technique is the Classification and Regression Tree (CART) method, which uses **Recursive Binary Splitting**

1. The method begins with the entire data set S and searches every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squared error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. The method then repeats step 1 for each of the two groups S_1 and S_2.
3. The method continues splitting groups until each subdivision has few observation

Regression Trees

Basic regression trees partition data into smaller groups that are more homogeneous with respect to predictors.

- They then make predictions based on average value of response in each group

The most common technique is the Classification and Regression Tree (CART) method, which uses **Recursive Binary Splitting**

1. The method begins with the entire data set S and searches every value of every predictor to cut S into two groups S_1 and S_2 that minimizes sum of squared error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. The method then repeats step 1 for each of the two groups S_1 and S_2.
3. The method continues splitting groups until each subdivision has few observation

- This is a *greedy* algorithm similar to forward selection, making the best choice at each stage. But its not necessary that the algorithm creates model with best RSS
  - Its possible a suboptimal choice early could lead to extremely beneficial choice later, reducing the overall RSS

Trees

Portland, OR is known for its coffee, its politics, its 117 degree summers, but also its . . .

Trees

Portland, OR is known for its coffee, its politics, its 117 degree summers, but also its . . .

Trees!

pdxTrees

We'll study regression trees using a subset of the pdxTrees data set from the pdxTrees GitHub repo (Maintained by K. McConville, I. Caldwell, and N. Horton)

pdxTrees

We'll study regression trees using a subset of the `pdxTrees` data set from the `pdxTrees` GitHub repo (Maintained by K. McConville, I. Caldwell, and N. Horton)

- The data was collected by the Portland Parks and Rec's *Urban Forestry Tree Inventory Project*.
- The Tree Inventory Project has gathered data on Portland trees since 2010, collecting this data in the summer months with a team of over 1,300 volunteers and city employees.

pdxTrees

We'll study regression trees using a subset of the `pdxTrees` data set from the `pdxTrees` GitHub repo (Maintained by K. McConville, I. Caldwell, and N. Horton)

- The data was collected by the Portland Parks and Rec's *Urban Forestry Tree Inventory Project*.
- The Tree Inventory Project has gathered data on Portland trees since 2010, collecting this data in the summer months with a team of over 1,300 volunteers and city employees.
- The `pdxTrees` dataset is too large to install alongside the package. Instead, the package provides helper loading functions:
  - `get_pdxTrees_parks()` pulls data on 25,534 trees from 174 Portland parks
  - `get_pdxTrees_streets()` pulls data on 218,602 trees along Portland streets

pdxTrees Data

- To keep things manageable, we'll limit our study to trees in just a few parks:

```
library(pdxTrees)
my_pdxTrees <- get_pdxTrees_parks(park = c("Kenilworth Park", "Westmoreland Park",
                                          "Woodstock Park","Berkeley Park", "Powell Park"))
```

pdxTrees Data

- To keep things manageable, we'll limit our study to trees in just a few parks:

```
library(pdxTrees)
my_pdxTrees <- get_pdxTrees_parks(park = c("Kenilworth Park", "Westmoreland Park",
                                           "Woodstock Park","Berkeley Park", "Powell Park"))
```

- How many observations?

```
dim(my_pdxTrees)
```

```
## [1] 1039   35
```

pdxTrees Data

- To keep things manageable, we'll limit our study to trees in just a few parks:

```
library(pdxTrees)
my_pdxTrees <- get_pdxTrees_parks(park = c("Kenilworth Park", "Westmoreland Park",
                                          "Woodstock Park","Berkeley Park", "Powell Park"))
```

- How many observations?

```
dim(my_pdxTrees)
```

```
## [1] 1039   35
```

- What variables are present?

```
names(my_pdxTrees)
```

pdxTrees Data

- To keep things manageable, we'll limit our study to trees in just a few parks:

```
library(pdxTrees)
my_pdxTrees <- get_pdxTrees_parks(park = c("Kenilworth Park", "Westmoreland Park",
                                          "Woodstock Park","Berkeley Park", "Powell Park"))
```

- How many observations?

```
dim(my_pdxTrees)
```

```
## [1] 1039   35
```

- What variables are present?

```
names(my_pdxTrees)
```

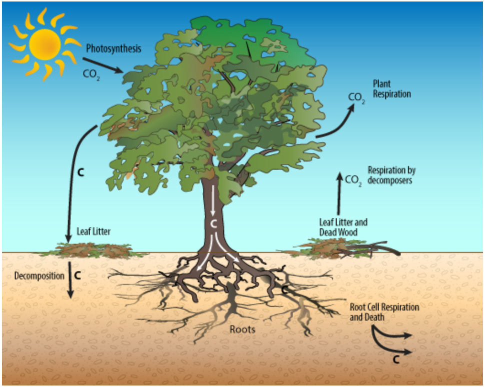
```
##  [1] "Longitude"                   "Latitude"
##  [3] "UserID"                      "Genus"
##  [5] "Family"                      "DBH"
##  [7] "Inventory_Date"              "Species"
##  [9] "Common_Name"                 "Condition"
## [11] "Tree_Height"                 "Crown_Width_NS"
## [13] "Crown_Width_EW"              "Crown_Base_Height"
## [15] "Collected_By"                "Park"
## [17] "Scientific_Name"             "Functional_Type"
## [19] "Mature_Size"                 "Native"
## [21] "Edible"                      "Nuisance"
## [23] "Structural_Value"            "Carbon_Storage_lb"
## [25] "Carbon_Storage_value"        "Carbon_Sequestration_lb"
## [27] "Carbon_Sequestration_value" "Stormwater_ft"
## [29] "Stormwater_value"            "Pollution_Removal_value"
## [31] "Pollution_Removal_oz"        "Total_Annual_Services"
## [33] "Origin"                      "Species_Factoid"
## [35] "Crown_Width"
```

Carbon Sequestration

- Forestry carbon sequestration is the process by which trees capture and store carbon dioxide from the atmosphere

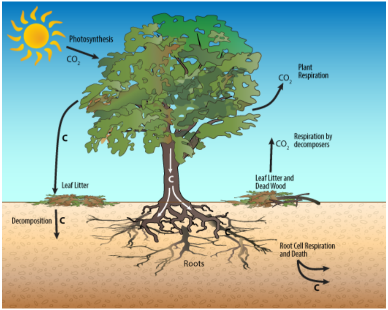
Carbon Sequestration

- Forestry carbon sequestration is the process by which trees capture and store carbon dioxide from the atmosphere



Carbon Sequestration

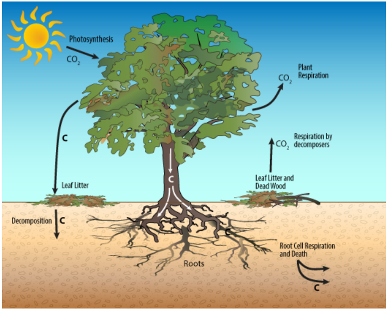
- Forestry carbon sequestration is the process by which trees capture and store carbon dioxide from the atmosphere



- Annual carbon sequestration of tree depends on several factors:

Carbon Sequestration

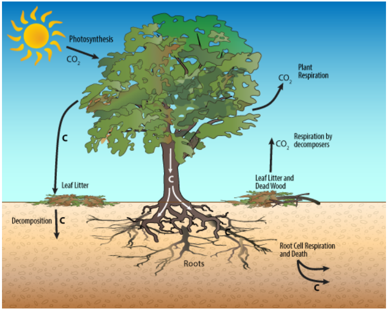
- Forestry carbon sequestration is the process by which trees capture and store carbon dioxide from the atmosphere



- Annual carbon sequestration of tree depends on several factors:
  - Species, Size, Age, Location, Weather, etc.

Carbon Sequestration

- Forestry carbon sequestration is the process by which trees capture and store carbon dioxide from the atmosphere



- Annual carbon sequestration of tree depends on several factors:
  - Species, Size, Age, Location, Weather, etc.
- Who might be interested in estimating the carbon sequestration of a tree?
  - Why?

Predicting Carbon Sequestration

- Can we predict carbon sequestration based on other tree features?
  - In `pdxTrees`, annual carbon sequestration is encoded as `Carbon_Sequestration_lb`.

Predicting Carbon Sequestration

- Can we predict carbon sequestration based on other tree features?
  - In `pdxTrees`, annual carbon sequestration is encoded as `Carbon_Sequestration_lb`.
- Let's focus on size-related variables:
  - `Tree_height`, height from ground to the live top of tree (in ft)
  - `Crown_Width`, the average of North-South and East-West canopy width (in ft)

Predicting Carbon Sequestration

- Can we predict carbon sequestration based on other tree features?
  - In `pdxTrees`, annual carbon sequestration is encoded as `Carbon_Sequestration_lb`.
- Let's focus on size-related variables:
  - `Tree_height`, height from ground to the live top of tree (in ft)
  - `Crown_Width`, the average of North-South and East-West canopy width (in ft)
- We are looking at a model of the form

```
Carbon_Sequestration ~ Tree_Height + Crown_Width
```

Predicting Carbon Sequestration

- Can we predict carbon sequestration based on other tree features?
    - In `pdxTrees`, annual carbon sequestration is encoded as `Carbon_Sequestration_lb`.
- Let's focus on size-related variables:
    - `Tree_height`, height from ground to the live top of tree (in ft)
    - `Crown_Width`, the average of North-South and East-West canopy width (in ft)
- We are looking at a model of the form

```
Carbon_Sequestration ~ Tree_Height + Crown_Width
```

- What are correlations among these 3 variables?

Predicting Carbon Sequestration

- Can we predict carbon sequestration based on other tree features?
  - In `pdxTrees`, annual carbon sequestration is encoded as `Carbon_Sequestration_lb`.
- Let's focus on size-related variables:
  - `Tree_height`, height from ground to the live top of tree (in ft)
  - `Crown_Width`, the average of North-South and East-West canopy width (in ft)
- We are looking at a model of the form

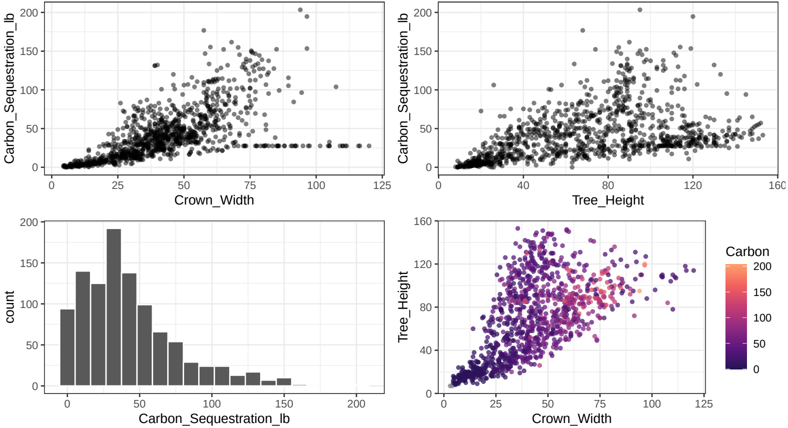
```
Carbon_Sequestration ~ Tree_Height + Crown_Width
```

- What are correlations among these 3 variables?

```
##                          Carbon_Sequestration_lb Crown_Width Tree_Height
## Carbon_Sequestration_lb                1.0000000   0.6126951   0.4362020
## Crown_Width                            0.6126951   1.0000000   0.5980118
## Tree_Height                            0.4362020   0.5980118   1.0000000
```

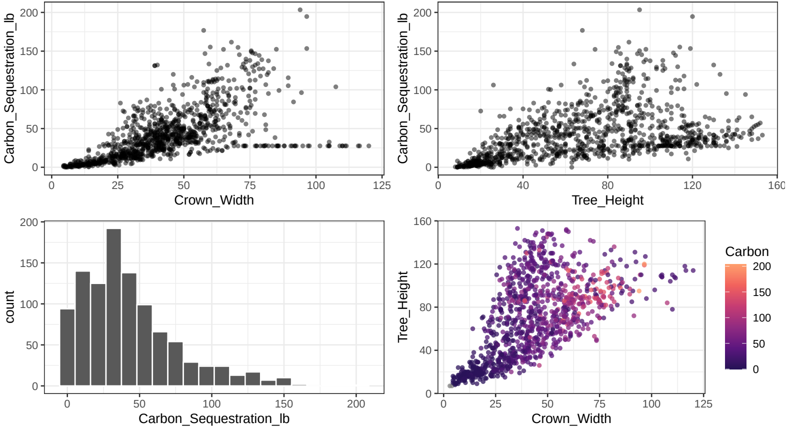
Predicting Carbon Sequestration

- Can we predict carbon sequestration based on other tree features?



Predicting Carbon Sequestration

- Can we predict carbon sequestration based on other tree features?



- Observations?

An Old Friend

This seems like a good time to implement linear regression:

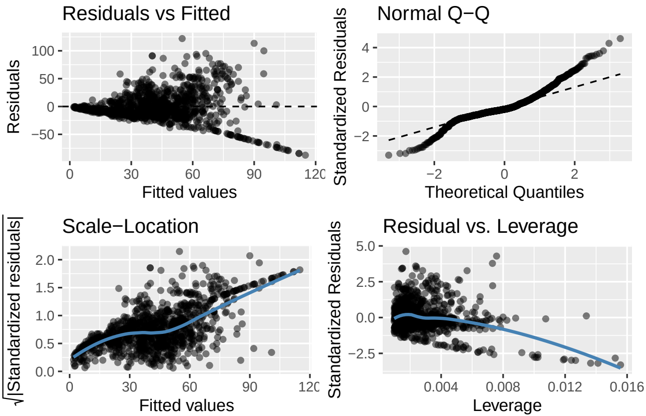
An Old Friend

This seems like a good time to implement linear regression:

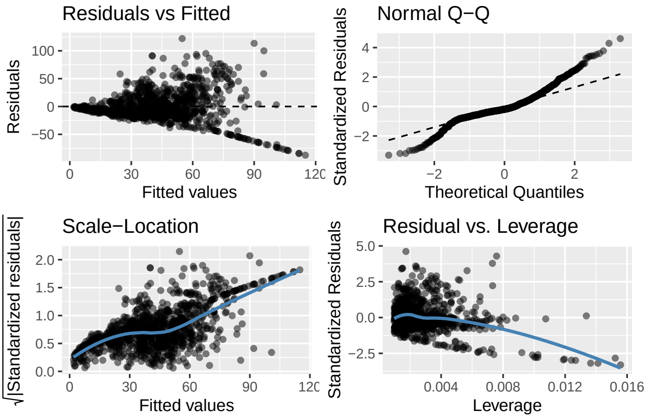
```r
tree_lm<-lm(Carbon_Sequestration_lb ~Crown_Width + Tree_Height, data=my_pdxTrees)
summary(tree_lm)
```

```
##
## Call:
## lm(formula = Carbon_Sequestration_lb ~ Crown_Width + Tree_Height,
##     data = my_pdxTrees)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -87.395 -13.283  -4.912  10.982 121.950
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.08819    2.03721  -1.516 0.129853
## Crown_Width  0.88769    0.04947  17.944  < 2e-16 ***
## Tree_Height  0.10140    0.02848   3.560 0.000388 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 26.46 on 1031 degrees of freedom
##   (5 observations deleted due to missingness)
## Multiple R-squared:  0.383,  Adjusted R-squared:  0.3818
## F-statistic:   320 on 2 and 1031 DF,  p-value: < 2.2e-16
```

Diagnostic Plots



Diagnostic Plots



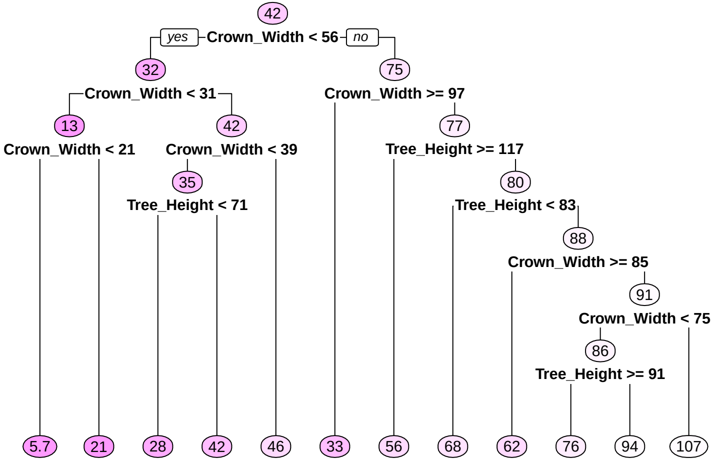
- Concerns?

Regression Tree

- Instead, let's build a regression tree:

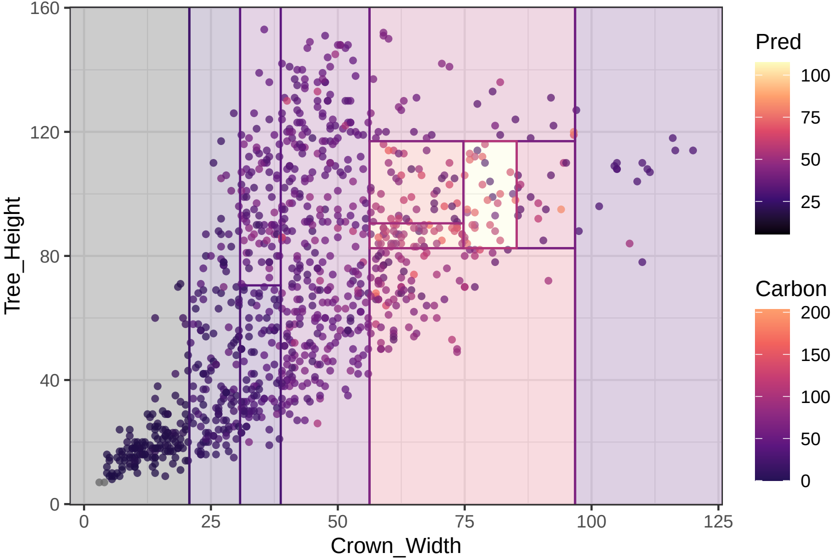
Regression Tree

- Instead, let's build a regression tree:



- Leaves at the bottom of the tree provide predictions

Another Visualization



Interpretation

- `Crown_Width` is the most important predictor of `Carbon_Sequestration_lb`
- After accounting for width, `Tree_Height` has some impact on `Carbon_Sequestration_lb`
- Very narrow and very wide trees tend to have low `Carbon_Sequestration_lb`
- Trees of moderate width and height have largest `Carbon_Sequestration_lb`

Tree Accuracy

- Let's create a test set consisting of two other parks:

```
my_pdxTrees_test <- get_pdxTrees_parks(park = c("Mt Scott Park", "Glenwood Park"))
```

Tree Accuracy

- Let's create a test set consisting of two other parks:

```
my_pdxTrees_test <- get_pdxTrees_parks(park = c("Mt Scott Park", "Glenwood Park"))
```

- We'll measure the accuracy of our model using root Mean Square Error:

$$\mathrm{rMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

```
##   Tree_rMSE
## 1  15.37258
```

Tree Accuracy

- Let's create a test set consisting of two other parks:

```
my_pdxTrees_test <- get_pdxTrees_parks(park = c("Mt Scott Park", "Glenwood Park"))
```

- We'll measure the accuracy of our model using root Mean Square Error:

$$\mathrm{rMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

```
##   Tree_rMSE
## 1  15.37258
```

- And compared to the linear model:

```
##    lm_rMSE
## 1 16.87355
```

Tree Accuracy

- Let's create a test set consisting of two other parks:

```r
my_pdxTrees_test <- get_pdxTrees_parks(park = c("Mt Scott Park", "Glenwood Park"))
```

- We'll measure the accuracy of our model using root Mean Square Error:

$$\text{rMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

```
##   Tree_rMSE
## 1  15.37258
```

- And compared to the linear model:

```
##    lm_rMSE
## 1 16.87355
```

- Why did the tree model outperform the linear model?

Tree Accuracy

- Let's create a test set consisting of two other parks:

```r
my_pdxTrees_test <- get_pdxTrees_parks(park = c("Mt Scott Park", "Glenwood Park"))
```

- We'll measure the accuracy of our model using root Mean Square Error:

$$\text{rMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

```
##   Tree_rMSE
## 1  15.37258
```

- And compared to the linear model:

```
##    lm_rMSE
## 1 16.87355
```

- Why did the tree model outperform the linear model?
  - Nevertheless, what are some downsides to the tree model?

Section 3

Pruning

Model Accuracy and Tree Algorithm

Recall the CART algorithm:

Model Accuracy and Tree Algorithm

Recall the CART algorithm:

1. Cut the data set S into two groups S_1 and S_2 that minimizes sum of squared error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. Repeats step 1 for each of the two groups S_1 and S_2.
3. Continue splitting groups until each subdivision has few observation

Model Accuracy and Tree Algorithm

Recall the CART algorithm:

1. Cut the data set S into two groups S_1 and S_2 that minimizes sum of squared error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. Repeats step 1 for each of the two groups S_1 and S_2.
3. Continue splitting groups until each subdivision has few observation

- This model will tend to produce accurate estimates on the *training set* (why?)

Model Accuracy and Tree Algorithm

Recall the CART algorithm:

1. Cut the data set S into two groups S_1 and S_2 that minimizes sum of squared error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. Repeats step 1 for each of the two groups S_1 and S_2.
3. Continue splitting groups until each subdivision has few observation

- This model will tend to produce accurate estimates on the *training set* (why?)
- However, it is extremely liable to overfit, and produce inaccurate estimates on *test set*

Model Accuracy and Tree Algorithm

Recall the CART algorithm:

1. Cut the data set S into two groups S_1 and S_2 that minimizes sum of squared error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. Repeats step 1 for each of the two groups S_1 and S_2.
3. Continue splitting groups until each subdivision has few observation

- This model will tend to produce accurate estimates on the *training set* (why?)
- However, it is extremely liable to overfit, and produce inaccurate estimates on *test set*
- To improve test RMSE, we need to reduce variance in model estimates (at the potential cost of some increased bias)

Model Accuracy and Tree Algorithm

Recall the CART algorithm:

1. Cut the data set S into two groups S_1 and S_2 that minimizes sum of squared error:
$$\text{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$
2. Repeats step 1 for each of the two groups S_1 and S_2.
3. Continue splitting groups until each subdivision has few observation

- This model will tend to produce accurate estimates on the *training set* (why?)
- However, it is extremely liable to overfit, and produce inaccurate estimates on *test set*
- To improve test RMSE, we need to reduce variance in model estimates (at the potential cost of some increased bias)
- To accomplish this, we can force trees to be less thorough in creating homogeneous groups (i.e. have fewer nodes)

Model Accuracy and Tree Algorithm

Recall the CART algorithm:

1. Cut the data set S into two groups S_1 and S_2 that minimizes sum of squared error:

$$\mathrm{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

2. Repeats step 1 for each of the two groups S_1 and S_2.
3. Continue splitting groups until each subdivision has few observation

- This model will tend to produce accurate estimates on the *training set* (why?)
- However, it is extremely liable to overfit, and produce inaccurate estimates on *test set*
- To improve test RMSE, we need to reduce variance in model estimates (at the potential cost of some increased bias)
- To accomplish this, we can force trees to be less thorough in creating homogeneous groups (i.e. have fewer nodes)
  - Option 1: Grow "younger" trees that are shorter
  - Option 2: Grow "mature" trees that are longer, and prune them back

Subtrees

Recall that when we grow trees, we use the greedy recursive binary splitting algorithm

Subtrees

Recall that when we grow trees, we use the greedy recursive binary splitting algorithm

- This means any splits that are made early will persist throughout the rest of the model

Subtrees

Recall that when we grow trees, we use the greedy recursive binary splitting algorithm

- This means any splits that are made early will persist throughout the rest of the model
- For this reason, growing a “young” tree (Method 1) tends to be ineffective, since the model may be burdened with a poor early choice

Subtrees

Recall that when we grow trees, we use the greedy recursive binary splitting algorithm

- This means any splits that are made early will persist throughout the rest of the model
- For this reason, growing a “young” tree (Method 1) tends to be ineffective, since the model may be burdened with a poor early choice
- The better strategy is to grow a full tree, and then remove some of the less helpful terminal leaves (Method 2)

Subtrees

Recall that when we grow trees, we use the greedy recursive binary splitting algorithm

- This means any splits that are made early will persist throughout the rest of the model
- For this reason, growing a “young” tree (Method 1) tends to be ineffective, since the model may be burdened with a poor early choice
- The better strategy is to grow a full tree, and then remove some of the less helpful terminal leaves (Method 2)

A **subtree** is a regression tree obtained by removing some of the branches and nodes from the full regression tree.

Subtrees

Recall that when we grow trees, we use the greedy recursive binary splitting algorithm

- This means any splits that are made early will persist throughout the rest of the model
- For this reason, growing a "young" tree (Method 1) tends to be ineffective, since the model may be burdened with a poor early choice
- The better strategy is to grow a full tree, and then remove some of the less helpful terminal leaves (Method 2)

A **subtree** is a regression tree obtained by removing some of the branches and nodes from the full regression tree.

- A subtree will tend to have higher *training* RSS than the full tree. But can often have lower *test* RSS than the full tree.

Pruning Algorithm

- Once a tree is fully grown, we *prune* it using *cost-complexity tuning*

Pruning Algorithm

- Once a tree is fully grown, we *prune* it using *cost-complexity tuning*
  - The goal is to find a tree of optimal size with the smallest error rate. We consider a sequence of trees indexed by a tuning parameter α.

Pruning Algorithm

- Once a tree is fully grown, we *prune* it using *cost-complexity tuning*
  - The goal is to find a tree of optimal size with the smallest error rate. We consider a sequence of trees indexed by a tuning parameter α.
- For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

$$\mathrm{RSS} + \alpha|T|$$

where $|T|$ is the number of terminal nodes of the tree T.

Pruning Algorithm

- Once a tree is fully grown, we *prune* it using *cost-complexity tuning*
  - The goal is to find a tree of optimal size with the smallest error rate. We consider a sequence of trees indexed by a tuning parameter α.
- For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

$$\mathrm{RSS} + \alpha|T|$$

  where $|T|$ is the number of terminal nodes of the tree T.
  - That is, α penalizes a tree based on its number of terminal nodes.

Pruning Algorithm

- Once a tree is fully grown, we *prune* it using *cost-complexity tuning*
  - The goal is to find a tree of optimal size with the smallest error rate. We consider a sequence of trees indexed by a tuning parameter α.
- For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

$$\mathrm{RSS} + \alpha|T|$$

where $|T|$ is the number of terminal nodes of the tree T.

  - That is, α penalizes a tree based on its number of terminal nodes.
  - This is analogous to the penalty parameter in Penalized Regression (LASSO / Ridge Regression) as well as in Cp/AIC/BIC for Best Subset

Pruning Algorithm

- Once a tree is fully grown, we *prune* it using *cost-complexity tuning*
  - The goal is to find a tree of optimal size with the smallest error rate. We consider a sequence of trees indexed by a tuning parameter α.
- For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes
$$\mathrm{RSS} + \alpha|T|$$
where $|T|$ is the number of terminal nodes of the tree T.
  - That is, α penalizes a tree based on its number of terminal nodes.
  - This is analogous to the penalty parameter in Penalized Regression (LASSO / Ridge Regression) as well as in Cp/AIC/BIC for Best Subset
- As α increases from 0 (i.e. the full tree), branches get pruned in a predictable way, making for relatively quick computation.

Pruning Algorithm

- Once a tree is fully grown, we *prune* it using *cost-complexity tuning*
  - The goal is to find a tree of optimal size with the smallest error rate. We consider a sequence of trees indexed by a tuning parameter α.
- For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

  $$\mathrm{RSS} + \alpha|T|$$

  where $|T|$ is the number of terminal nodes of the tree T.
  - That is, α penalizes a tree based on its number of terminal nodes.
  - This is analogous to the penalty parameter in Penalized Regression (LASSO / Ridge Regression) as well as in Cp/AIC/BIC for Best Subset
- As α increases from 0 (i.e. the full tree), branches get pruned in a predictable way, making for relatively quick computation.
- We can find the optimal value of α by further splitting training data into a training and test set (or using k-fold cross-validation). We can choose the **best** subtree by:

Pruning Algorithm

- Once a tree is fully grown, we *prune* it using *cost-complexity tuning*
  - The goal is to find a tree of optimal size with the smallest error rate. We consider a sequence of trees indexed by a tuning parameter α.
- For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

$$\mathrm{RSS} + \alpha|T|$$

  where $|T|$ is the number of terminal nodes of the tree T.
  - That is, α penalizes a tree based on its number of terminal nodes.
  - This is analogous to the penalty parameter in Penalized Regression (LASSO / Ridge Regression) as well as in Cp/AIC/BIC for Best Subset
- As α increases from 0 (i.e. the full tree), branches get pruned in a predictable way, making for relatively quick computation.
- We can find the optimal value of α by further splitting training data into a training and test set (or using k-fold cross-validation). We can choose the **best** subtree by:

1. Choosing the tree with smallest rMSE.

Pruning Algorithm

- Once a tree is fully grown, we *prune* it using *cost-complexity tuning*
  - The goal is to find a tree of optimal size with the smallest error rate. We consider a sequence of trees indexed by a tuning parameter α.
- For each value of α, there exists a unique subtree T of the full tree T_0 that minimizes

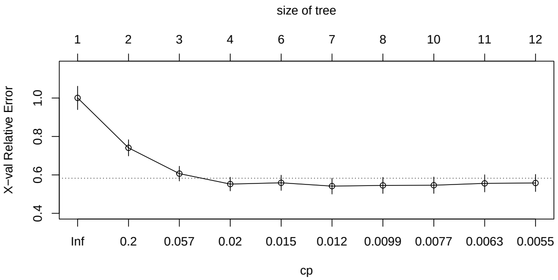
$$\mathrm{RSS} + \alpha|T|$$

  where $|T|$ is the number of terminal nodes of the tree T.
  - That is, α penalizes a tree based on its number of terminal nodes.
  - This is analogous to the penalty parameter in Penalized Regression (LASSO / Ridge Regression) as well as in Cp/AIC/BIC for Best Subset
- As α increases from 0 (i.e. the full tree), branches get pruned in a predictable way, making for relatively quick computation.
- We can find the optimal value of α by further splitting training data into a training and test set (or using k-fold cross-validation). We can choose the **best** subtree by:

1. Choosing the tree with smallest rMSE.
2. Choosing the *smallest* tree with rMSE within 1 standard deviation of lowest rMSE

Pruning Example

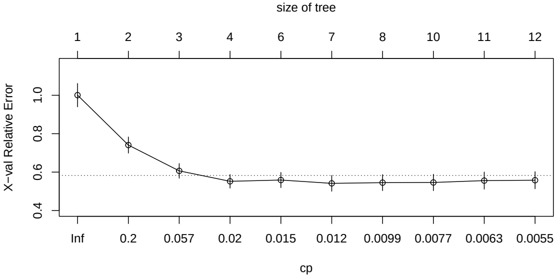
- How does rMSE vary as tree size changes?



- Horizontal axis gives values of complexity parameter (cp)
- Upper scale indicates number of terminal nodes for given tree
- Vertical axis gives the cross-validated relative root mean squared error
- Dotted horizontal line has height equal to 1 standard error above smallest rMSE

Pruning Example

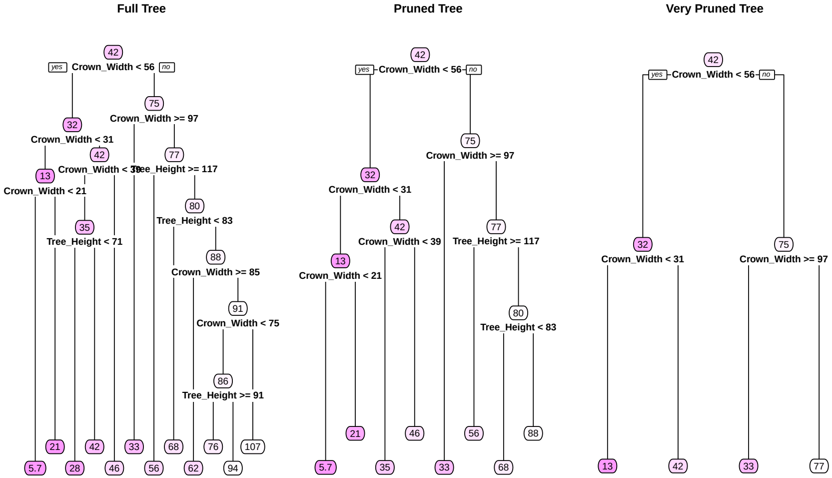
- How does rMSE vary as tree size changes?



- What are the test MSEs for the full tree and the subtrees with 4 and 8 leaves?

```
## # A tibble: 4 x 4
##   model       .metric .estimator .estimate
##   <chr>       <chr>   <chr>          <dbl>
## 1 pruned      rmse    standard        14.3
## 2 full        rmse    standard        15.4
## 3 linear      rmse    standard        16.9
## 4 very pruned rmse    standard        16.9
```

Comparison



Full Tree
42
yes
Crown_Width < 56
no
75
Crown_Width >= 97
32
Crown_Width < 31
42
77
13
Crown_Width < 39
Tree_Height >= 117
Crown_Width < 21
80
35
Tree_Height < 83
Tree_Height < 71
88
Crown_Width >= 85
91
Crown_Width < 75
86
Tree_Height >= 91
21
42
33
68
76
107
5.7
28
46
56
62
94
Pruned Tree
42
yes
Crown_Width < 56
no
75
Crown_Width >= 97
32
Crown_Width < 31
42
77
Crown_Width < 39
Tree_Height >= 117
13
Crown_Width < 21
80
Tree_Height < 83
21
46
56
88
5.7
35
33
68
Very Pruned Tree
42
yes
Crown_Width < 56
no
32
Crown_Width < 31
75
Crown_Width >= 97
13
42
33
77

Section 4

Trees in R

Creating Tree Models in R

There are two common packages for creating regression trees in R: `tree` and `rpart`.

Creating Tree Models in R

There are two common packages for creating regression trees in R: `tree` and `rpart`.

- The `tree` package is one of the oldest packages on CRAN. It is a (tiny) bit easier to use. But allows far less customization. (Traditional)

Creating Tree Models in R

There are two common packages for creating regression trees in R: `tree` and `rpart`.

- The `tree` package is one of the oldest packages on CRAN. It is a (tiny) bit easier to use. But allows far less customization. (Traditional)
- The `rpart` package is newer, computationally faster, and has more options. It also can be combined with other packages for **much** nicer plots. (Recommended)

Trees using ‘rpart“

- To fit a tree using variables `Tree_Height` and `Crown_Width`:

```
set.seed(1)
library(rpart)
tree_model1 <- rpart(Carbon_Sequestration_lb ~
                       Tree_Height + Crown_Width,
                     data = my_pdxTrees)
```

Trees using ‘rpart“

- To fit a tree using variables `Tree_Height` and `Crown_Width`:

```
set.seed(1)
library(rpart)
tree_model1 <- rpart(Carbon_Sequestration_lb ~
                       Tree_Height + Crown_Width,
                     data = my_pdxTrees)
```

- We can change several features of the tree by adding a `control` argument:

```
set.seed(1)
tree_model2 <- rpart(Carbon_Sequestration_lb ~
                       Tree_Height + Crown_Width,
                     control = rpart.control(
                       minsplit = 20, xval = 10, maxdepth = 10, cp = 0.005),
                     data = my_pdxTrees)
```

Trees using ‘rpart“

- To fit a tree using variables `Tree_Height` and `Crown_Width`:

```
set.seed(1)
library(rpart)
tree_model1 <- rpart(Carbon_Sequestration_lb ~
                       Tree_Height + Crown_Width,
                     data = my_pdxTrees)
```

- We can change several features of the tree by adding a `control` argument:

```
set.seed(1)
tree_model2 <- rpart(Carbon_Sequestration_lb ~
                       Tree_Height + Crown_Width,
                     control = rpart.control(
                       minsplit = 20, xval = 10, maxdepth = 10, cp = 0.005),
                     data = my_pdxTrees)
```

- `minsplit` is the minimum number of observations in a node
- `xval` is the number of cross-validation folds used
- `maxdepth` is the maximum depth of any node in the final tree
- `cp` is the minimum reduction in RSS needed in order to attempt a split

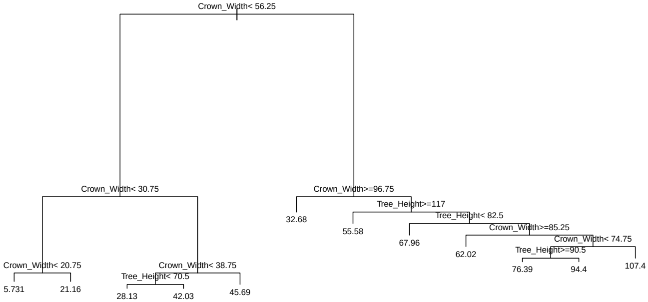
Plots using `plot`

- There are several options for visualizing trees with varying ease-of-use and aesthetics.
  - The base R `plot` function quickly generates plots, but...

Plots using plot

- There are several options for visualizing trees with varying ease-of-use and aesthetics.
  - The base R plot function quickly generates plots, but. . .

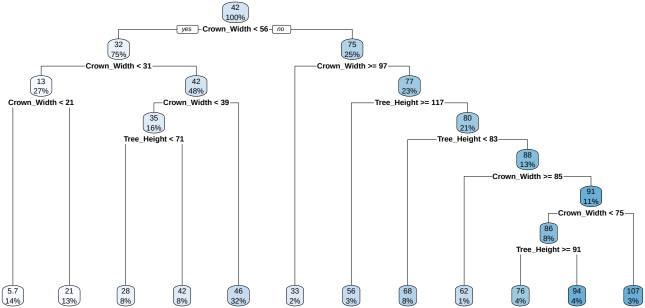
```
plot(tree_model2)
text(tree_model2, pretty = 0, cex = .5)
```



Plots using rpart.plot

- An alternative to plot is the rpart.plot function from the package of the same name:

```
library(rpart.plot)
rpart.plot(tree_model2)
```



- Some further customization available (see ?rpart.plot)

Trees in R via `rpart` cont'd

- The `rpart` function automatically performs k-fold CV when choosing among potential splits.

Trees in R via `rpart` cont'd

- The `rpart` function automatically performs k-fold CV when choosing among potential splits.
- To access results, append `$cptable` to the `rpart` model object:

```
tree_model2$cptable
```

```
##             CP nsplit rel error    xerror       xstd
## 1  0.304594426      0 1.0000000 1.0015819 0.06083997
## 2  0.127260632      1 0.6954056 0.7296616 0.04066145
## 3  0.025587347      2 0.5681449 0.6138089 0.03803769
## 4  0.015177861      3 0.5425576 0.5753989 0.03914593
## 5  0.014222123      5 0.5122019 0.5705610 0.03960569
## 6  0.010849075      6 0.4979797 0.5548873 0.03897808
## 7  0.009024312      7 0.4871307 0.5342982 0.03851474
## 8  0.006608748      9 0.4690820 0.5269734 0.04016718
## 9  0.006064592     10 0.4624733 0.5365762 0.04102266
## 10 0.005000000     11 0.4564087 0.5360229 0.04159291
```

Trees in R via `rpart` cont'd

- The `rpart` function automatically performs k-fold CV when choosing among potential splits.
- To access results, append `$cptable` to the `rpart` model object:

```
tree_model2$cptable
```

```
##             CP nsplit rel error    xerror       xstd
## 1  0.304594426      0 1.0000000 1.0015819 0.06083997
## 2  0.127260632      1 0.6954056 0.7296616 0.04066145
## 3  0.025587347      2 0.5681449 0.6138089 0.03803769
## 4  0.015177861      3 0.5425576 0.5753989 0.03914593
## 5  0.014222123      5 0.5122019 0.5705610 0.03960569
## 6  0.010849075      6 0.4979797 0.5548873 0.03897808
## 7  0.009024312      7 0.4871307 0.5342982 0.03851474
## 8  0.006608748      9 0.4690820 0.5269734 0.04016718
## 9  0.006064592     10 0.4624733 0.5365762 0.04102266
## 10 0.005000000     11 0.4564087 0.5360229 0.04159291
```

- `CP` is the value of the complexity parameter
- `nsplit` is number of splits
- `rel error` is $(1 - R^2)$, using $R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$
- `xerror` is cross-validated estimate of relative error
- `xstd` is the standard deviation in `xerror` based on CV

Analyze Results

- The `printcp` function displays key model information

```
printcp(tree_model2)
```

```
##
## Regression tree:
## rpart(formula = Carbon_Sequestration_lb ~ Tree_Height + Crown_Width,
##     data = my_pdxTrees, control = rpart.control(minsplit = 20,
##         xval = 10, maxdepth = 10, cp = 0.005))
##
## Variables actually used in tree construction:
## [1] Crown_Width Tree_Height
##
## Root node error: 1175664/1037 = 1133.7
##
## n=1037 (2 observations deleted due to missingness)
##
##           CP nsplit rel error  xerror     xstd
## 1  0.3045944      0   1.00000 1.00158 0.060840
## 2  0.1272606      1   0.69541 0.72966 0.040661
## 3  0.0255873      2   0.56814 0.61381 0.038038
## 4  0.0151779      3   0.54256 0.57540 0.039146
## 5  0.0142221      5   0.51220 0.57056 0.039606
## 6  0.0108491      6   0.49798 0.55489 0.038978
## 7  0.0090243      7   0.48713 0.53430 0.038515
## 8  0.0066087      9   0.46908 0.52697 0.040167
## 9  0.0060646     10   0.46247 0.53658 0.041023
## 10 0.0050000     11   0.45641 0.53602 0.041593
```

Analyze Results cont'd

- Detailed listing of model parts can be accessed via `summary`:

Analyze Results cont'd

- Detailed listing of model parts can be accessed via `summary`:

```
summary(tree_model2)
```

```
## Call:
## rpart(formula = Carbon_Sequestration_lb ~ Tree_Height + Crown_Width,
##     data = my_pdxTrees, control = rpart.control(minsplit = 20,
##         xval = 10, maxdepth = 10, cp = 0.005))
##   n=1037 (2 observations deleted due to missingness)
##
##             CP nsplit rel error    xerror       xstd
## 1  0.304594426      0 1.0000000 1.0015819 0.06083997
## 2  0.127260632      1 0.6954056 0.7296616 0.04066145
## 3  0.025587347      2 0.5681449 0.6138089 0.03803769
## 4  0.015177861      3 0.5425576 0.5753989 0.03914593
## 5  0.014222123      5 0.5122019 0.5705610 0.03960569
## 6  0.010849075      6 0.4979797 0.5548873 0.03897808
## 7  0.009024312      7 0.4871307 0.5342982 0.03851474
## 8  0.006608748      9 0.4690820 0.5269734 0.04016718
## 9  0.006064592     10 0.4624733 0.5365762 0.04102266
## 10 0.005000000     11 0.4564087 0.5360229 0.04159291
##
## Variable importance
## Crown_Width Tree_Height
##          80          20
##
## Node number 1: 1037 observations,    complexity param=0.3045944
##   mean=42.47387, MSE=1133.717
##   left son=2 (778 obs) right son=3 (259 obs)
##   Primary splits:
##       Crown_Width < 56.25 to the left,  improve=0.3025602, (3 missing)
##       Tree_Height < 42.5  to the left,  improve=0.2366680, (0 missing)
##   Surrogate splits:
##       Tree_Height < 149.5 to the left,  agree=0.75, adj=0.004, (3 split)
##
```

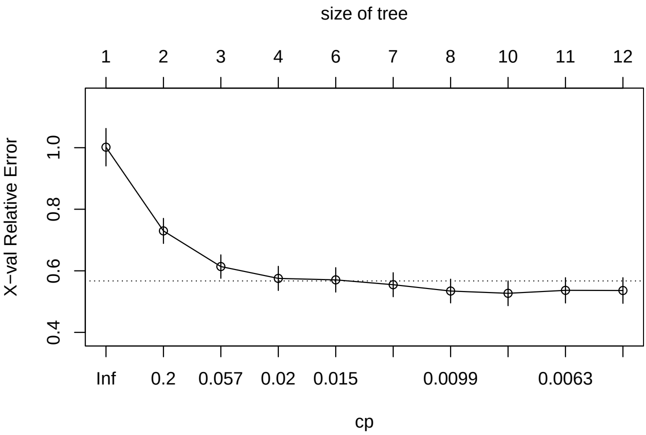
CV Plots

- We can plot the results of cross-validation using `plotcp`:

CV Plots

- We can plot the results of cross-validation using `plotcp`:

```
plotcp(tree_model2)
```



Pruning

- Based on the CV plot, 10 leaves with $CP = 0.0077$ gives the lowest error
  - While 7 leaves with $CP = 0.012$ gives smallest tree within 1 SE of best.

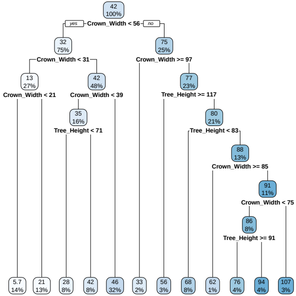
Pruning

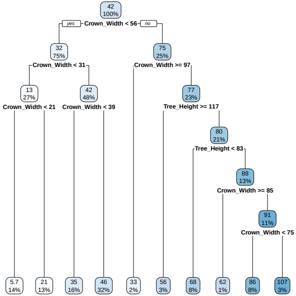
- Based on the CV plot, 10 leaves with $CP = 0.0077$ gives the lowest error
    - While 7 leaves with $CP = 0.012$ gives smallest tree within 1 SE of best.
- We can prune our tree using the `prune` function with a given value of `cp`

Pruning

- Based on the CV plot, 10 leaves with $CP = 0.0077$ gives the lowest error
  - While 7 leaves with $CP = 0.012$ gives smallest tree within 1 SE of best.
- We can prune our tree using the `prune` function with a given value of `cp`

```
pruned_tree <- prune(tree_model2, cp = 0.0077)
```





Test Error Rates

- How well do models do on the test data?

Test Error Rates

- How well do models do on the test data?
    - Let's build a results data frame:

```
results <- data.frame(model = "full",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(tree_model2, my_pdxTrees_test))
results <- rbind(results,
                 data.frame(model = "pruned",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(pruned_tree, my_pdxTrees_test)))
```

Test Error Rates

- How well do models do on the test data?
    - Let's build a results data frame:

```
results <- data.frame(model = "full",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(tree_model2, my_pdxTrees_test))
results <- rbind(results,
                 data.frame(model = "pruned",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(pruned_tree, my_pdxTrees_test)))
```

- And use `rmse` from `yardstick` to assess:

Test Error Rates

- How well do models do on the test data?
  - Let's build a results data frame:

```
results <- data.frame(model = "full",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(tree_model2, my_pdxTrees_test))
results <- rbind(results,
                 data.frame(model = "pruned",
                      obs = my_pdxTrees_test$Carbon_Sequestration_lb,
                      preds = predict(pruned_tree, my_pdxTrees_test)))
```

- And use rmse from yardstick to assess:

```
library(yardstick)
results %>% group_by(model) %>%
  rmse(truth = obs, estimate = preds) %>% arrange(.estimate)
```

```
## # A tibble: 2 x 4
##   model  .metric .estimator .estimate
##   <chr>  <chr>   <chr>          <dbl>
## 1 pruned rmse    standard        14.2
## 2 full   rmse    standard        15.4
```