# Big Data Analytics

## Session 6

## Decision Trees

# Where were we last week

- To assess the model accuracy, the measure of fit is
  - Test MSE for regression
  - Test error rate for classification

- Cross Validation
  - Estimate the test MSE/error rate in the absence of the designated test data
  - Compare different models and select the best one
  - Validation set approach, LOOCV and k-fold CV

- Bias and Variance tradeoff
  - The more flexible a method is the less bias it will generally have.
  - Generally, the more flexible a method is the more variance it has.

# Outline

- The Basics of Decision Trees
  - Regression Trees
  - Classification Trees

  - Pruning Trees

  - Trees vs. Linear Models
  - Advantages and Disadvantages of Trees

# An Example

## Grade distinctions at postgraduate level

At postgraduate taught level (PGCert, PGDip and Master's degrees), you may be awarded one of the following:

- **Distinction**: You will be awarded a Distinction if you achieve an average result of **70% or above** in modules at Level 7(M) as well as a distinction mark in the dissertation.
- **Merit**: You will be awarded a Merit if you achieve an average result of between 60% and 69% in modules at credit level 7(M).
- **Pass**: You will be awarded a Pass if you achieve an average result of between 50% and 59% in modules at credit level 7(M).
- **Fail**: You will be considered to have failed if you achieve an average result of below 50% in modules at credit level 7(M).

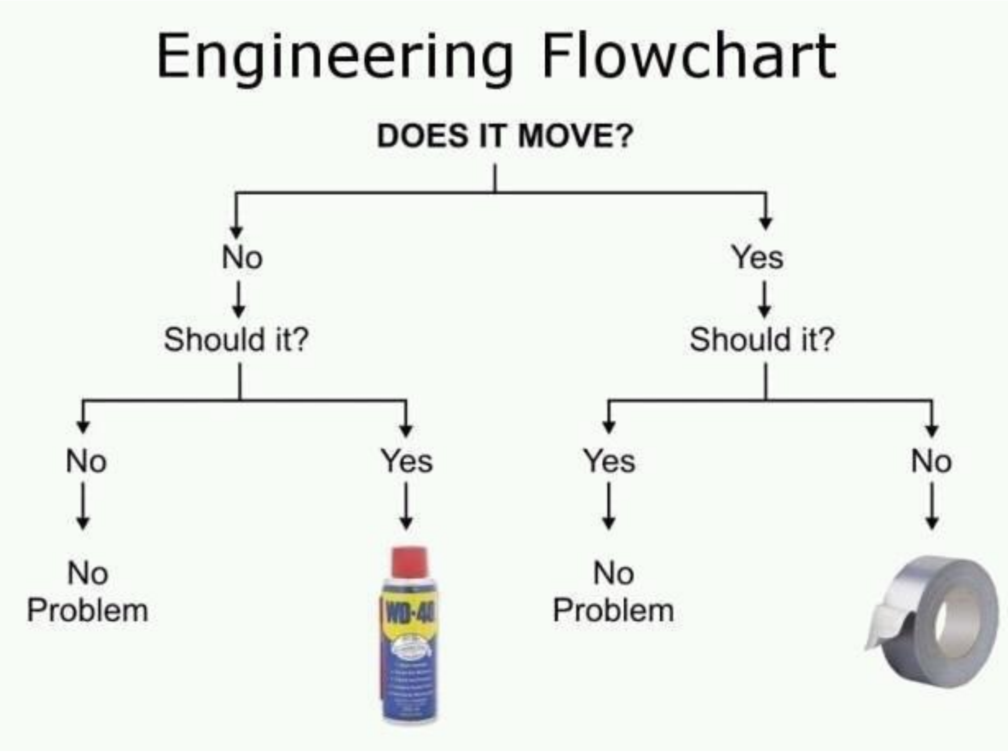
# Partitioning Up the Predictor Space

- In the example:
  - X: Average marks in Level 7 modules
    - ranging from 0 to 100 (predictor space)
  - Y: Grades
    - ranging from Fail, Pass, Merit, Distinction

  - Divide [0,100] into four regions
    - $R_1$: [0,50) – Fail
    - $R_2$: [50,60) – Pass
    - $R_3$: [60,70) – Merit
    - $R_4$: [70,100] – Distinction

# Decision Tree

- Decision tree
  - A flow-chart-like tree structure
  - Internal node denotes a test on an attribute
  - Branch represents an outcome of the test
  - Leaf nodes represent class labels or class distribution



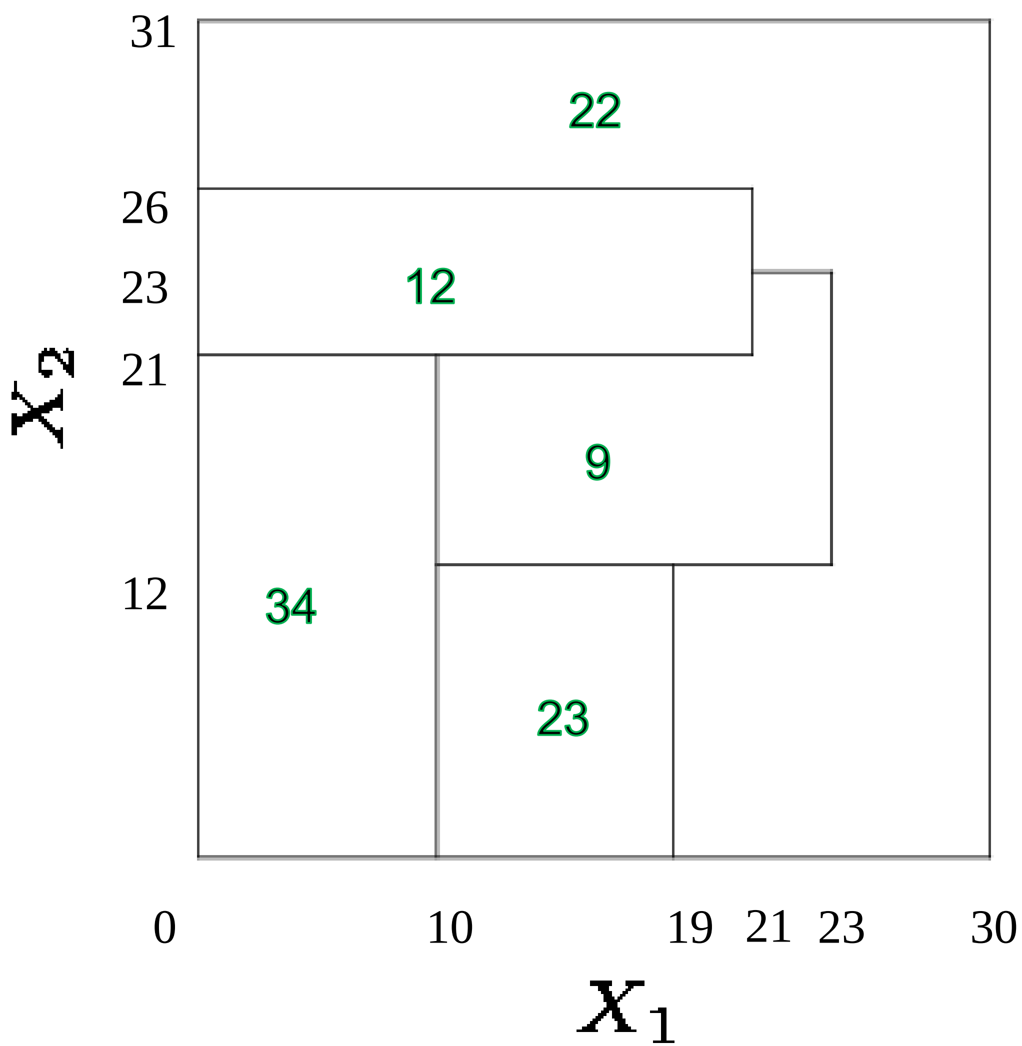
# Regression Trees

Predicting a quantitative response

e.g., predicting baseball players' salary

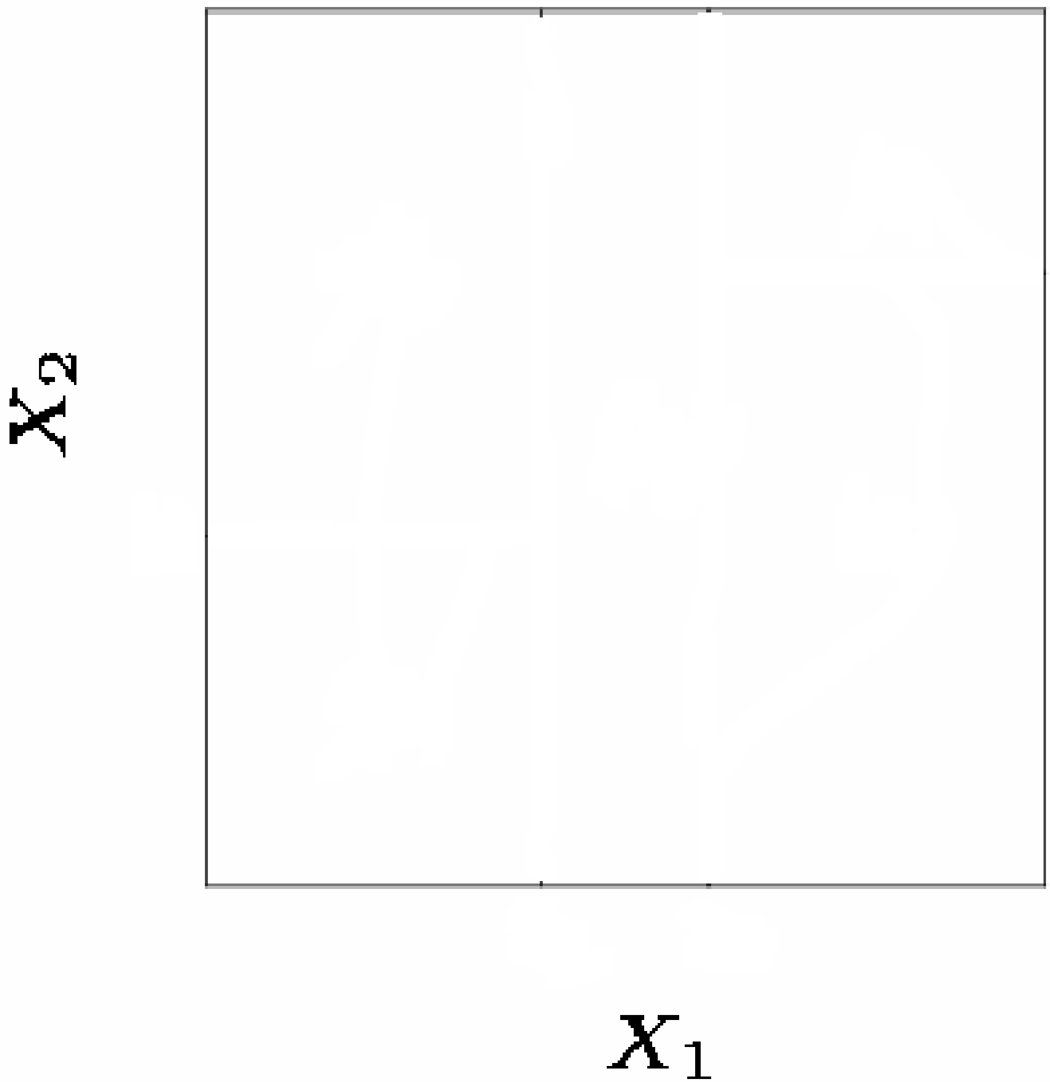
# The General View

- Here we have two predictors and five distinct regions
- Depending on which region our new X comes from we would make one of five possible predictions for Y
- Predict Y based on
  - $X_1=15, X_2=15$
  - $X_1=20, X_2=24$
  - $X_1=5, X_2=29$
  - $X_1=22, X_2=25$



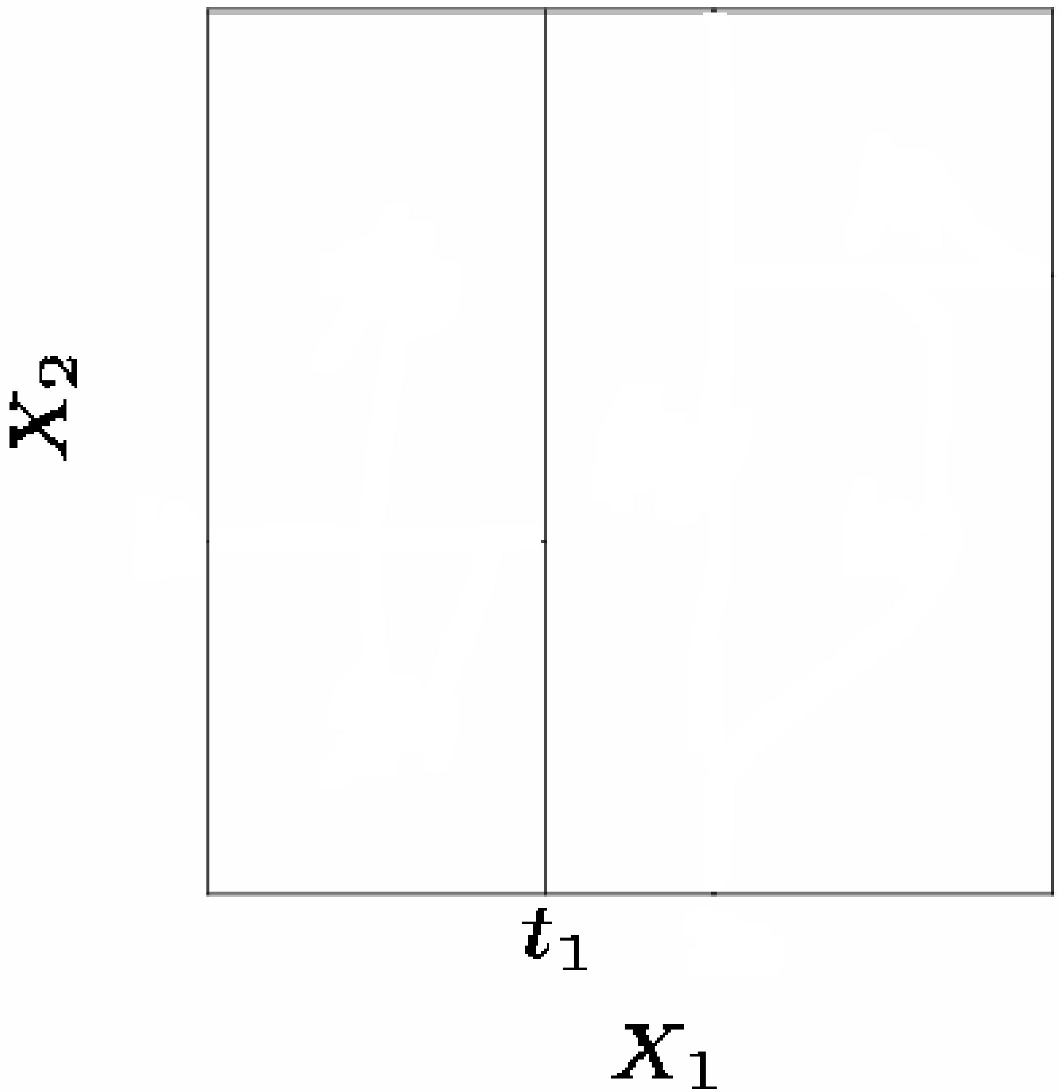
# Splitting the X Variables

- Generally we create the partitions by iteratively splitting one of the X variables into two regions



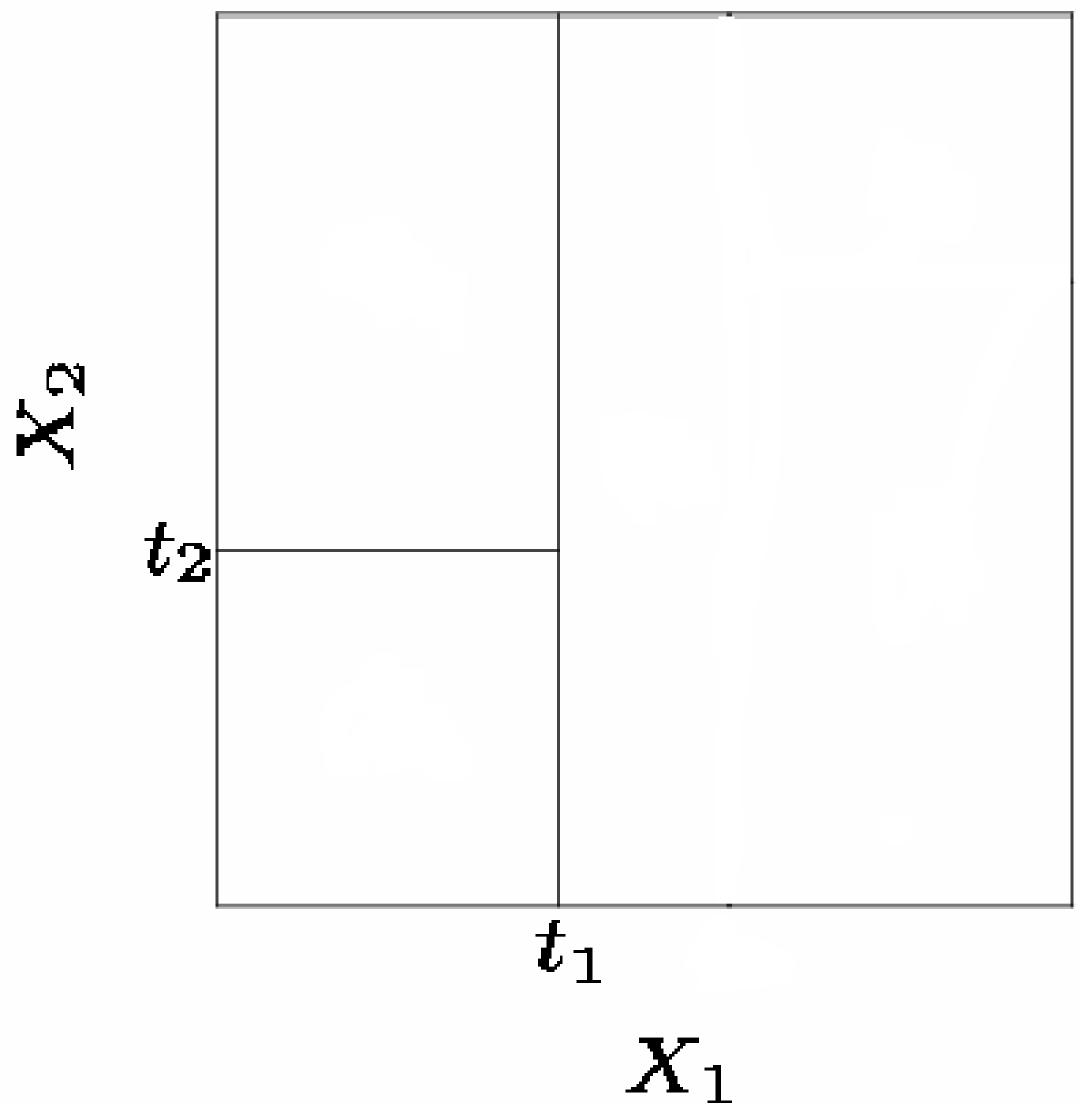
# Splitting the X Variable

1. First split on $X_1=t_1$



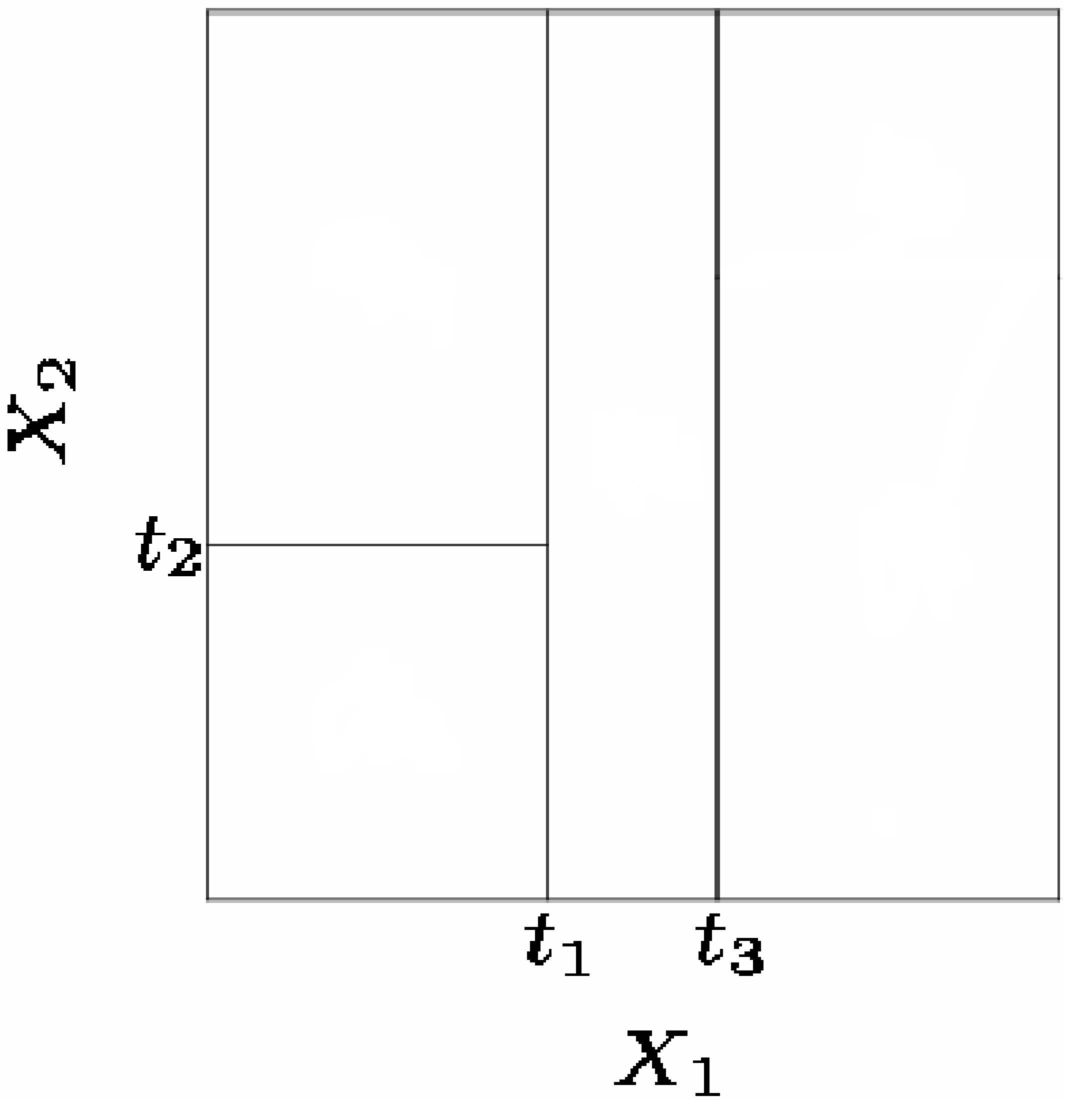
# Splitting the X Variable

1. First split on $X_1=t_1$
2. If $X_1<t_1$, split on $X_2=t_2$



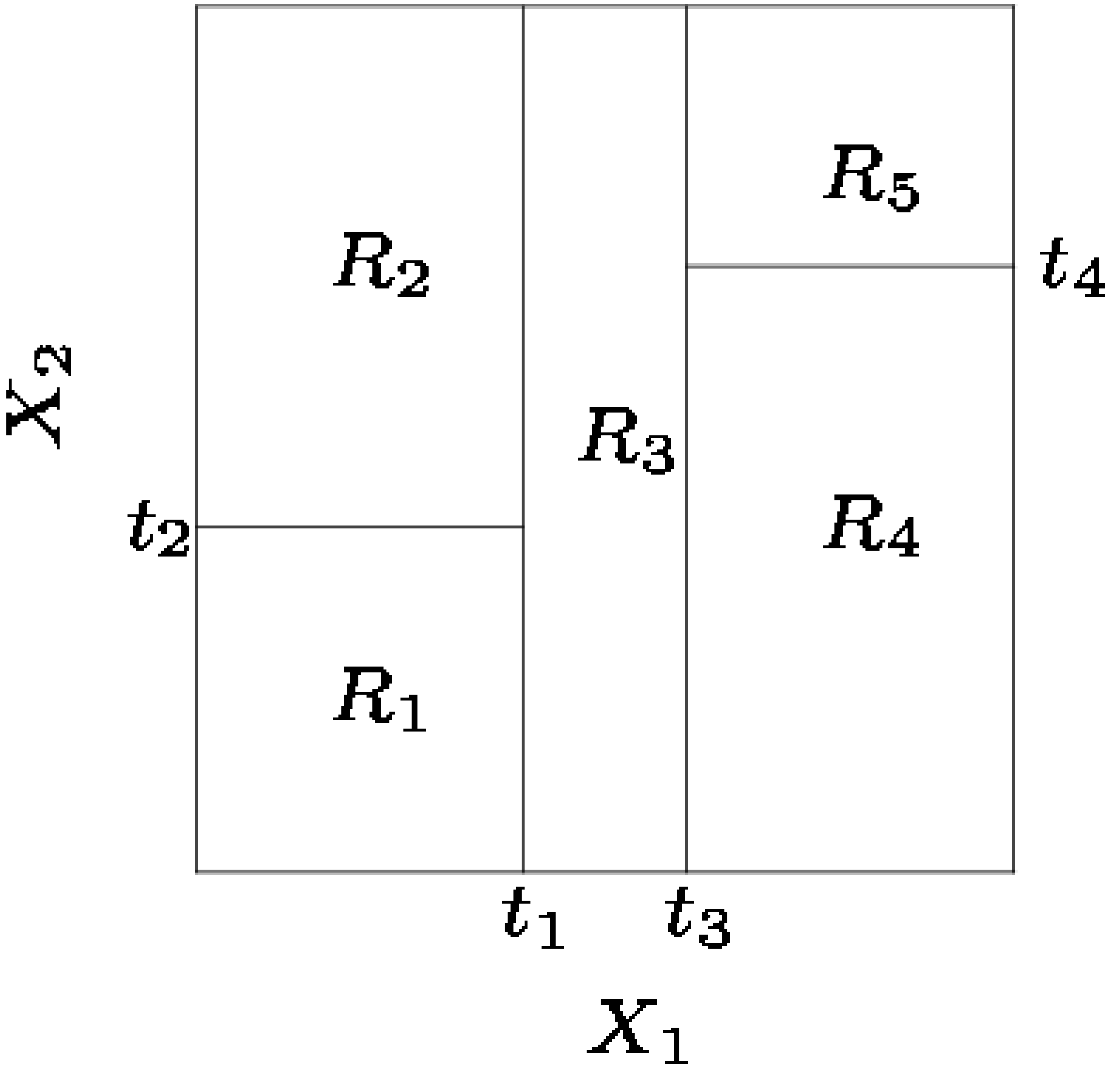
# Splitting the X Variable

1. First split on $X_1=t_1$
2. If $X_1<t_1$, split on $X_2=t_2$
3. If $X_1>t_1$, split on $X_1=t_3$

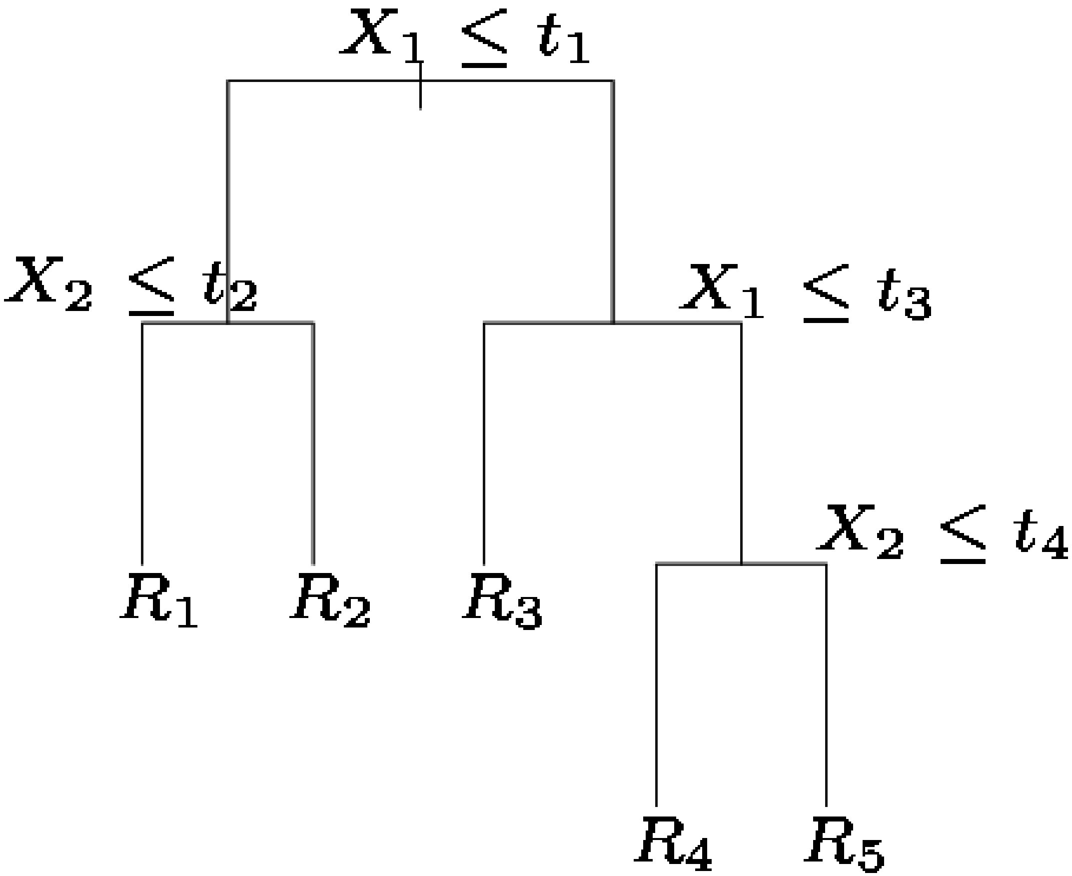


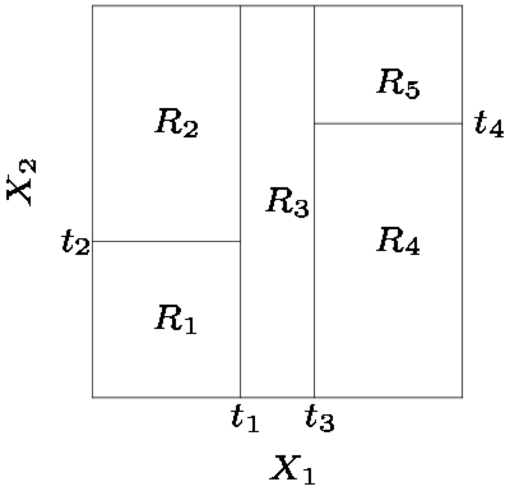
# Splitting the X Variable

1. First split on $X_1=t_1$
2. If $X_1<t_1$, split on $X_2=t_2$
3. If $X_1>t_1$, split on $X_1=t_3$
4. If $X_1>t_3$, split on $X_2=t_4$



# Splitting the X Variable




- When we create partitions this way we can always represent them using a tree structure.
- This provides a very simple way to explain the model to a non-expert i.e. your boss!

# Example: Baseball Players’ Salaries

- To predict baseball player’s salaries by regression tree based on
  - Years: the number of years that he has played in the major league
  - Hits: the number of hits that he made in the previous year
- Note that Salary is measured in 1000s, and log-transformed
  - Hitters$Salary=log(Hitters$Salary)
- The predicted salary for a player who played in the league for more than 4.5 years and had less than 117.5 hits last year is $\$1000 \times e^{6.00} = \$402,834$


- Can you 1) build a regression tree using Years and Hits and 2) make the prediction for a player with “Years”=5 and “Hits”=100 using R?

## Example: Baseball Players' Salaries

- Step 1: Building the tree:

```
> library(tree)
> library(ISLR) #You need to install package if necessary
> nrow(Hitters)
[1] 322

> Hitters=na.omit(Hitters)   #remove rows with missing observations
> nrow(Hitters)
[1] 263

> tree.hitters=tree(log(Salary)~Years+Hits, Hitters)
# type in tree.hitters or summary(tree.hitters) to see more details

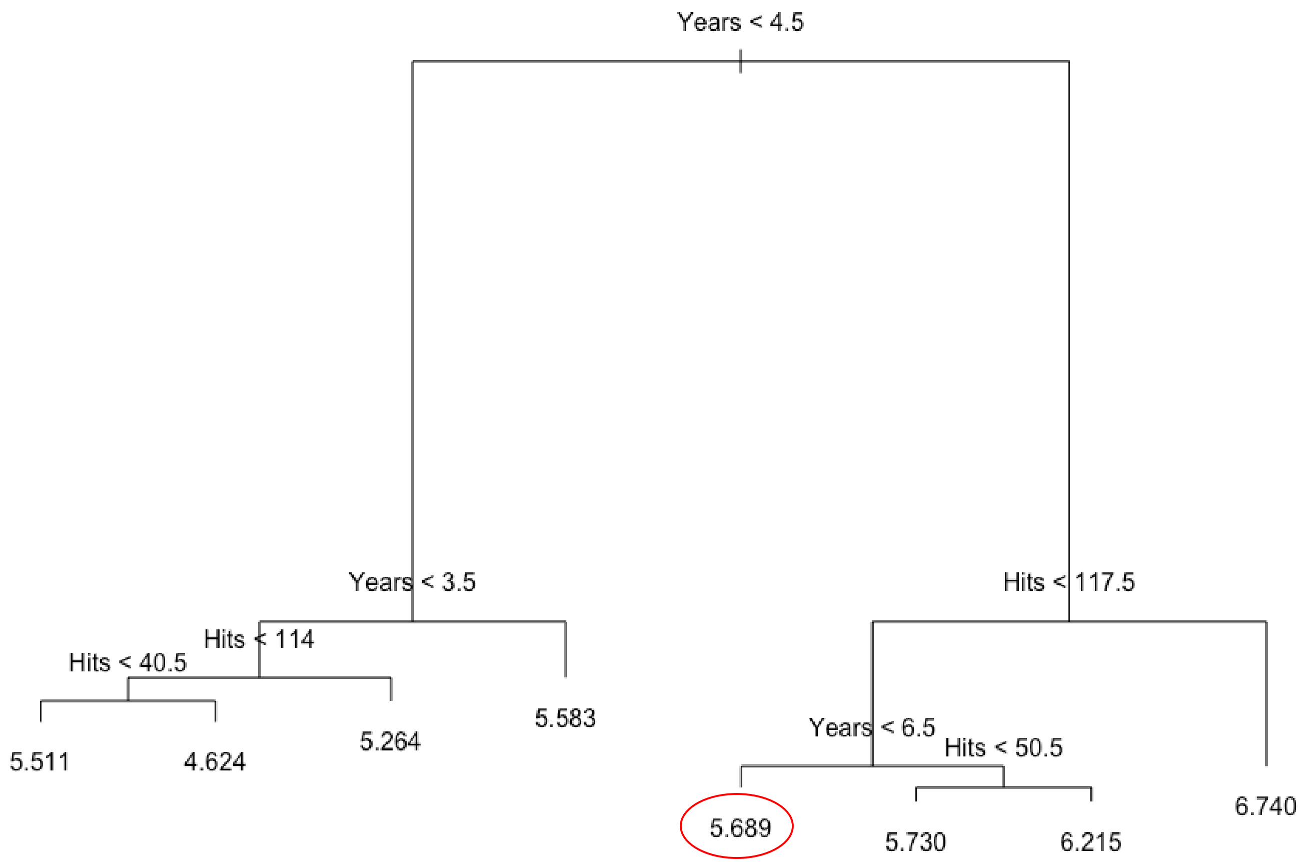
> plot(tree.hitters)                    #Why is this tree different from the one before?
> text(tree.hitters, pretty=0)   #pretty=0  includes the category names for any qualitative
                                 #predictors, rather than simply displaying a letter for each category
```

# Example: Baseball Players’ Salaries

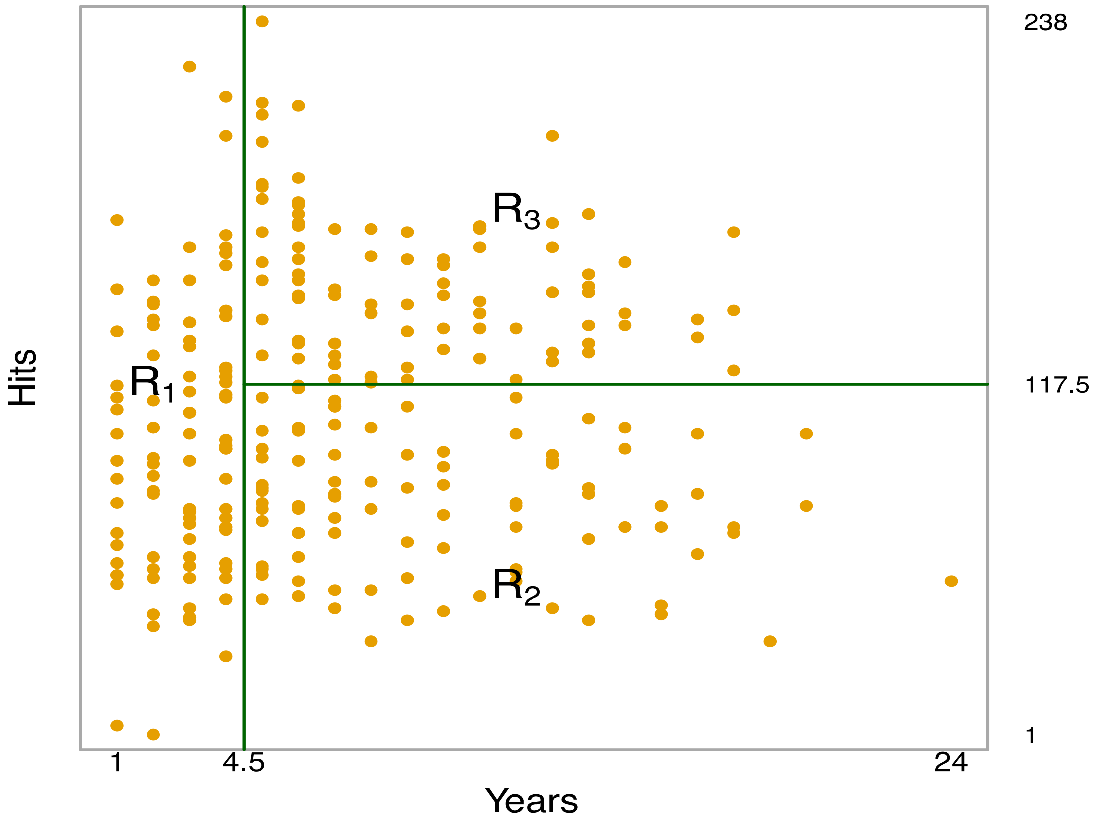
- Step 2: Making predictions

Given “Years”=5 and “Hits”=100, what is the prediction?

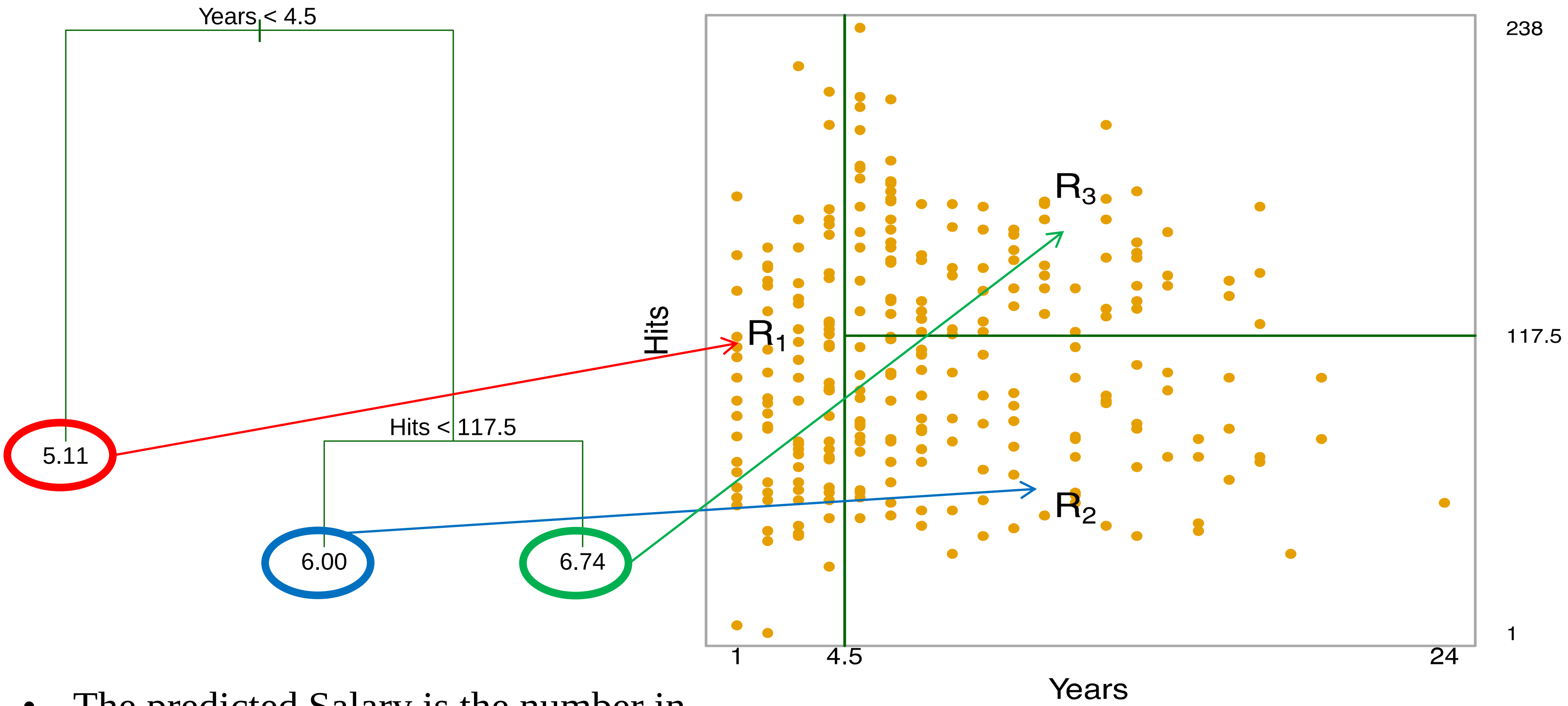
```
> yhat=predict(tree.hitters,newdata=list("Years"=5, "Hits"=100))
> yhat
 1
5.688925
```



# Another way of visualising the decision tree



# Linking two visualisations




- The predicted Salary is the number in each leaf node.
- It is the <u>mean</u> of the response for the observations that fall there

5.11 is the mean salary in region $R_1$
6.00 is the mean salary in region $R_2$
6.74 is the mean salary in region $R_3$

# Terminology of Decision Tree

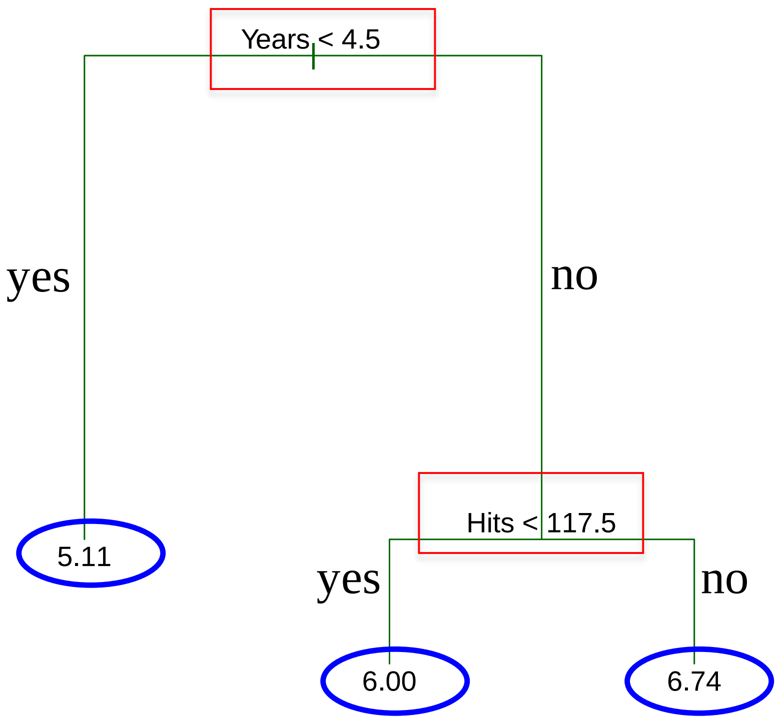
- Terminal nodes (leaves) of the tree
  - Leaf nodes represent class labels or class distribution
- Internal nodes
  - Internal node denotes a test on an attribute
- Branches
  - The segments of the trees that connect the nodes
  - Branch represents an outcome of the test
- Upside down

# Interpreting the Decision Tree

- `Years` is the most important factor in determining `Salary`
  - Players with less experience tend to earn less

- For less experienced players
  - The number of hits plays little role in the salaries

- For experienced players
  - The more hits being made, the higher salary they tend to earn



# Some Natural Questions

Q1. Where to split?

i.e., how do we decide on what regions to use i.e. $R_1, R_2, \ldots, R_k$?

or equivalently, what tree structure should we use?

Q2. What values should we use for $\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_k$ ?

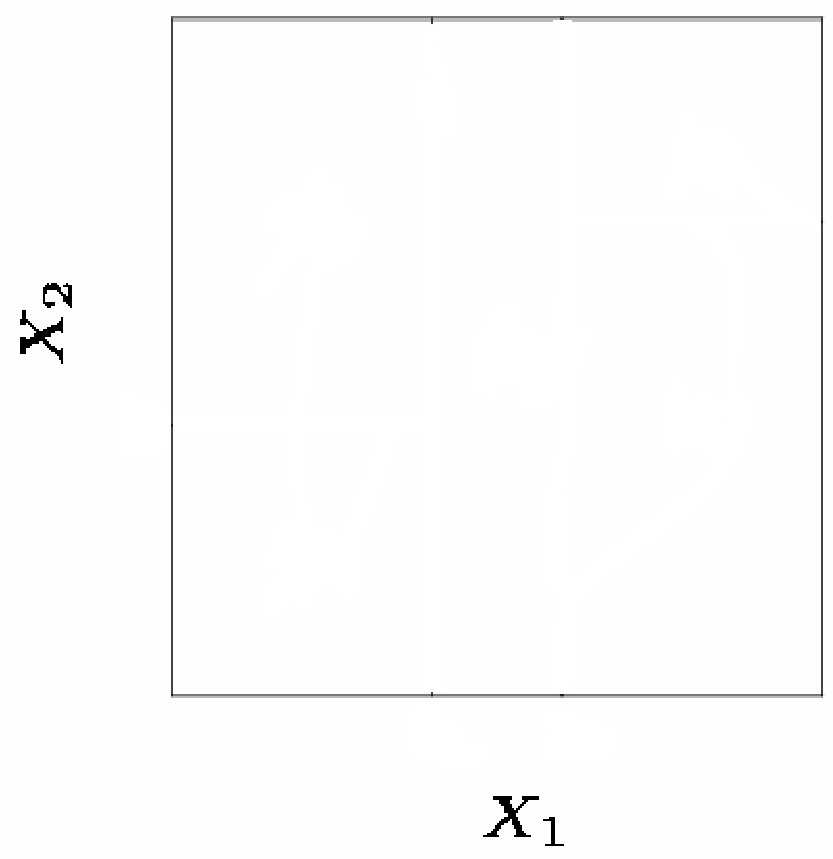
## Q2. What values should we use for $\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_k$ ?

- Simple!

- For region $R_j$, the best prediction is simply the average of all the responses from our training data that fell in region $R_j$.

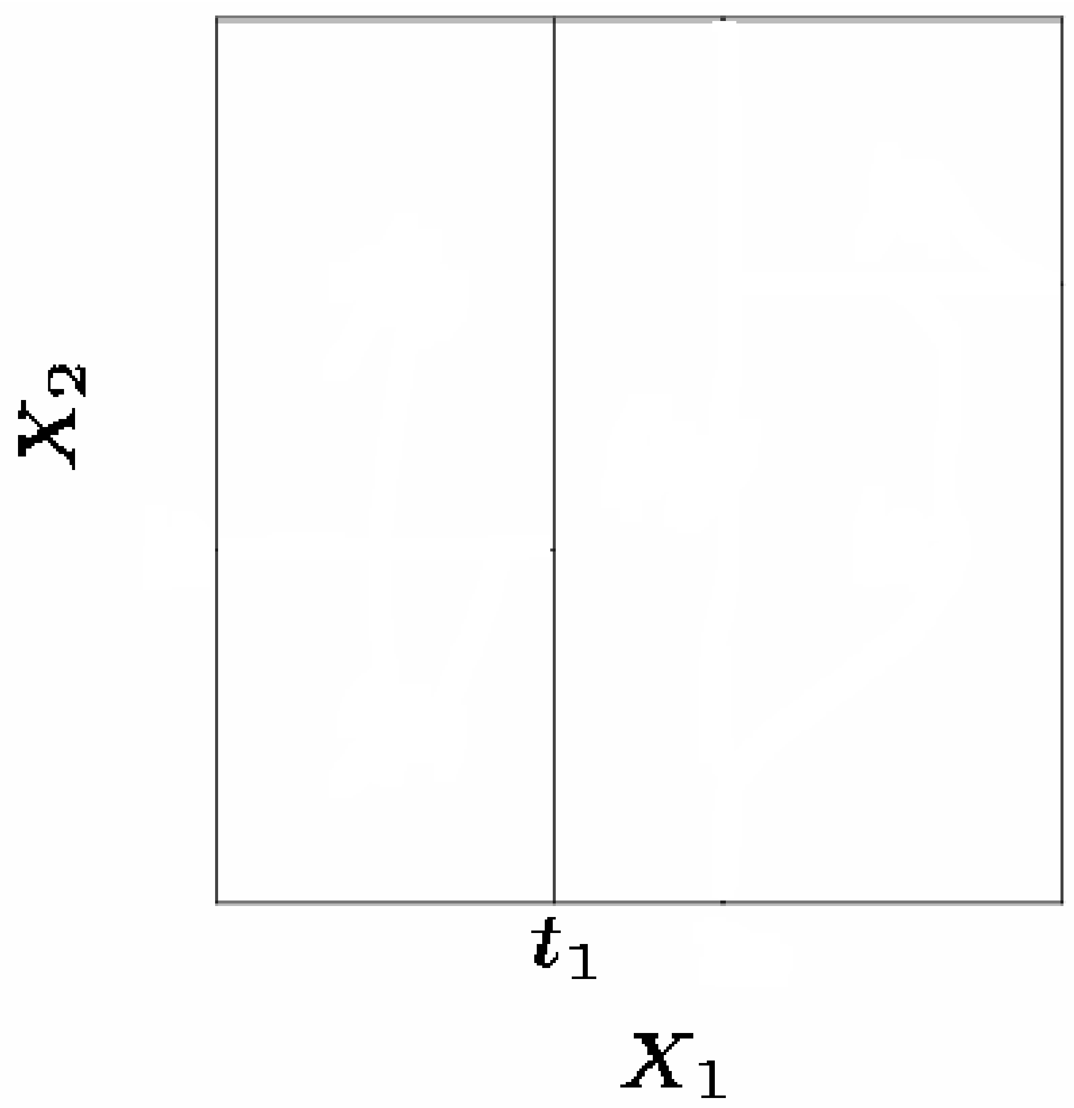
# Q1. Where to Split?

- We consider splitting into two regions, $X_j > s$ and $X_j < s$ for all possible values of s and j.
- We then choose the s and j that results in the lowest MSE on the training data.
- $X_1$: Years
- $X_2$: Hits



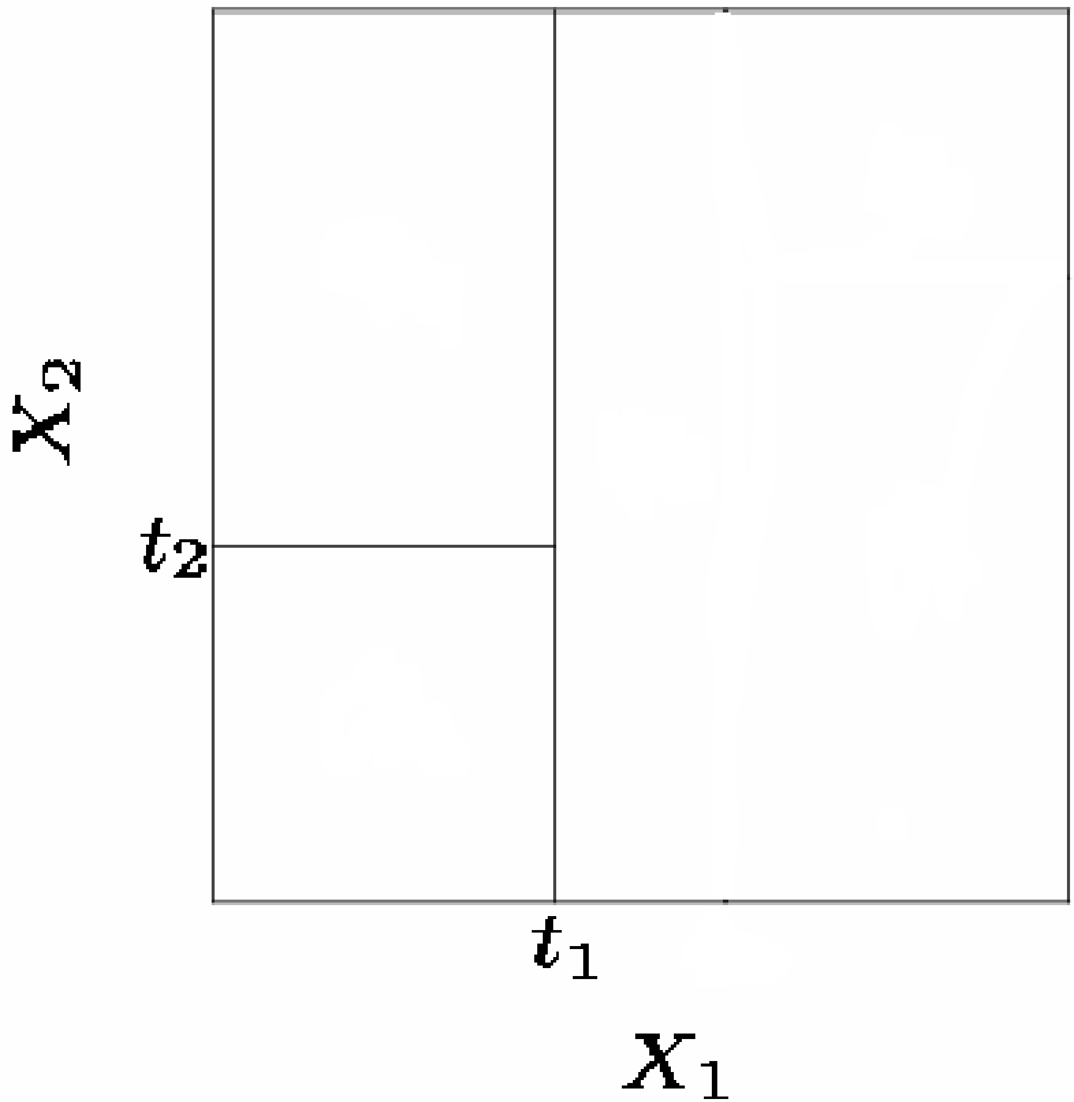
# Q1. Where to Split?

- Here the optimal split was on $X_1$ at point $t_1$.
- Now we repeat the process looking for the next best split except that we must also consider whether to split the first region or the second region up.
- Again the criteria is smallest MSE.



# Where to Split?

- Here the optimal split was the left region on $X_2$ at point $t_2$.
- [Stopping criteria] This process continues until our regions have too few observations to continue e.g. all regions have 5 or fewer points.



# Tree Pruning

# Improving Tree Accuracy

- A large tree (i.e. one with many terminal nodes) may tend to over fit the training data.
  - Large tree: lower bias, higher variance, worse interpretation
  - Small tree: higher bias, lower variance, better interpretation

- Generally, we can improve accuracy by "pruning" the tree, i.e. cutting off some of the terminal nodes.

- How do we know how far back to prune the tree?
  - We use **cross validation** to see which tree has the lowest error rate.

# Outline of the Decision Tree Approach

- Split the dataset `DS` into two subsets:
  - `DS.train` and `DS.test`

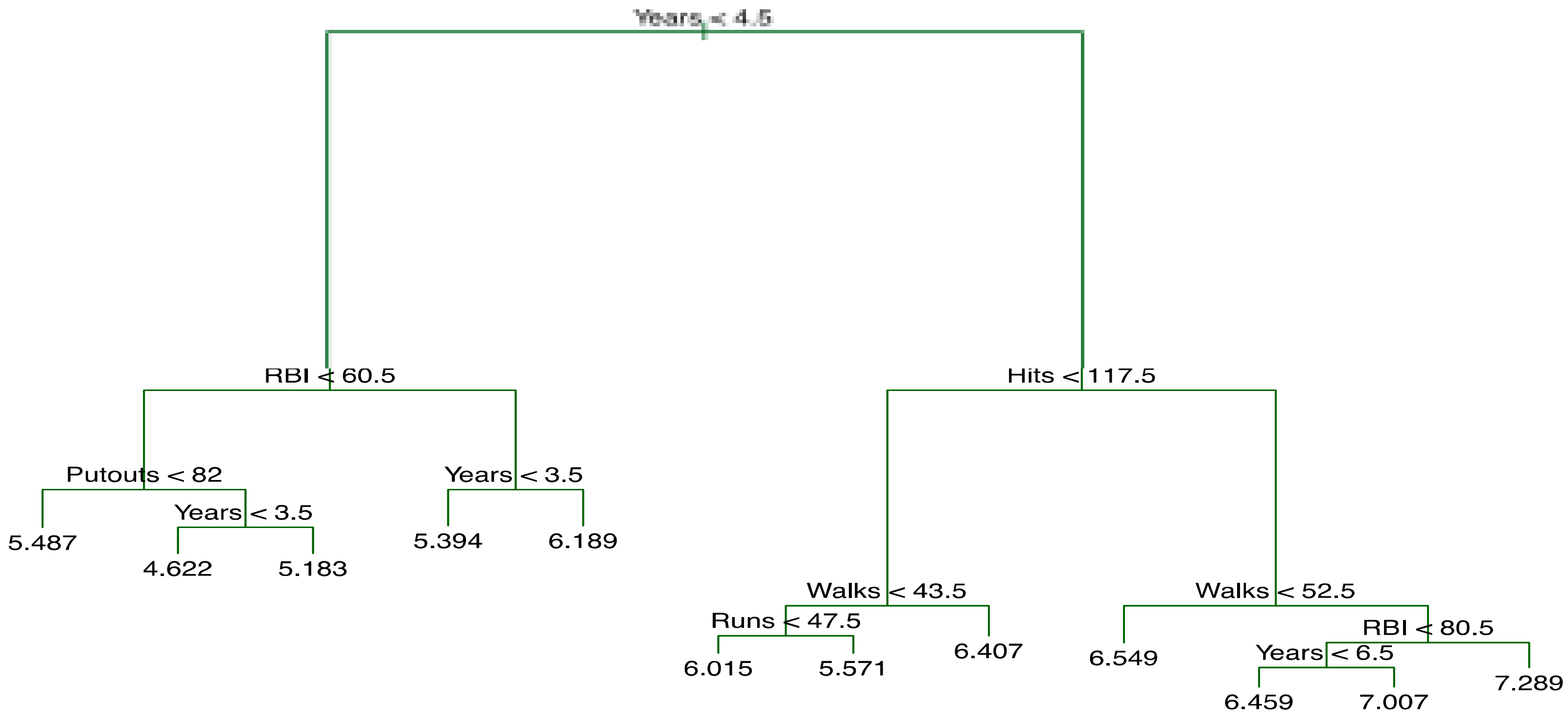
- Use `DS.train` to build a decision tree `tree.train`

- Use cross validation (`cv.tree()`) to see
  - whether pruning the tree will improve performance
  - If yes, how many leaves (`w`) the best tree will have

- Use `prune.tree(tree.train,best=w)` to prune the tree to have `w` leaves if necessary

- Make predictions on the test set `DS.test` and evaluate how well the model performs
  - Calculate the test MSE or test error rate

# Example: Baseball Players' Salaries




Can anyone get the same tree as this one when building a regression tree of Salary on 9 predictors?

Y: Salary

X: 9 predictors

Hits+Runs+RBI+Walks+Years+PutOuts+AtBat+Assists+Errors

# Example: Baseball Players' Salaries

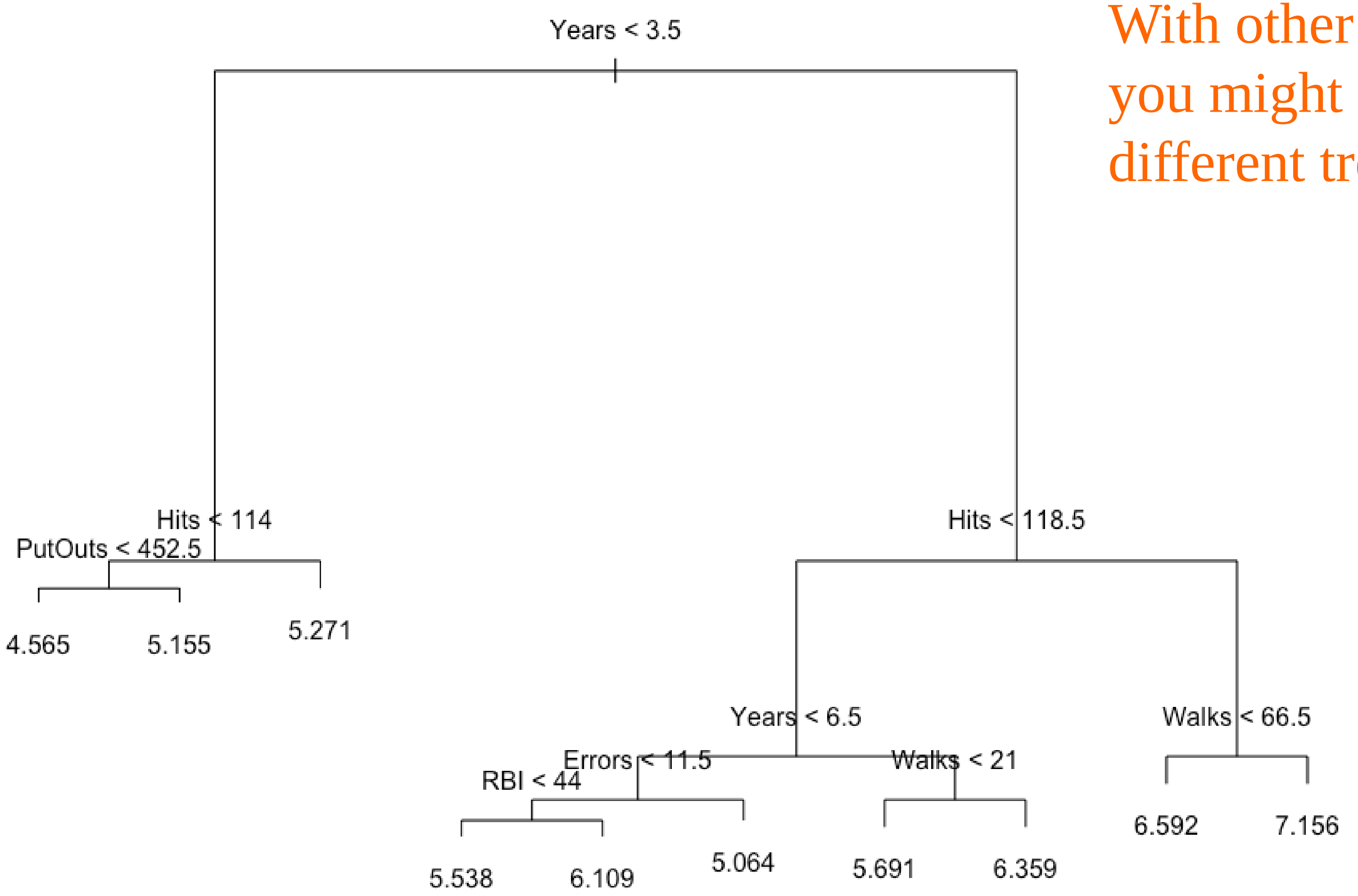
```
> set.seed(2)     #choosing a different seed → a different tree
> train=sample(1:nrow(Hitters),132)

>
tree.hitters.train=tree(log(Salary)~Hits+Runs+RBI+Walks+Years+PutOuts+AtBat+Assists+Errors,Hitters,subset=train)
>
> plot(tree.hitters.train)
> text(tree.hitters.train,pretty=0)
```

Use `summary(tree.hitters.train)` to see how many predictors actually contributed to the tree

```
Variables actually used in tree construction:
[1] "Years"   "Hits"    "PutOuts" "Errors"  "RBI"     "Walks"
```

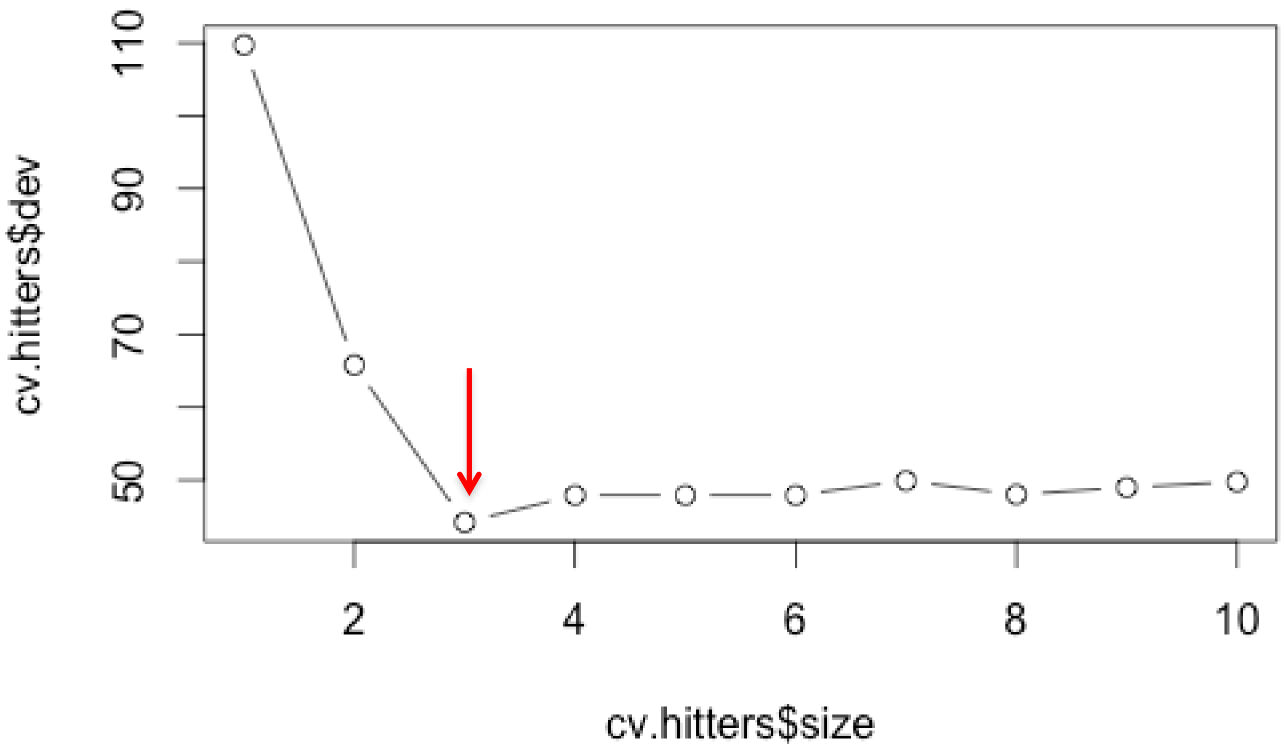
# Example: Baseball Players' Salaries




With other seeds, you might get a different tree

# Example: Baseball Players' Salaries

Now we use cv.tree() function to see whether pruning the tree will improve performance.

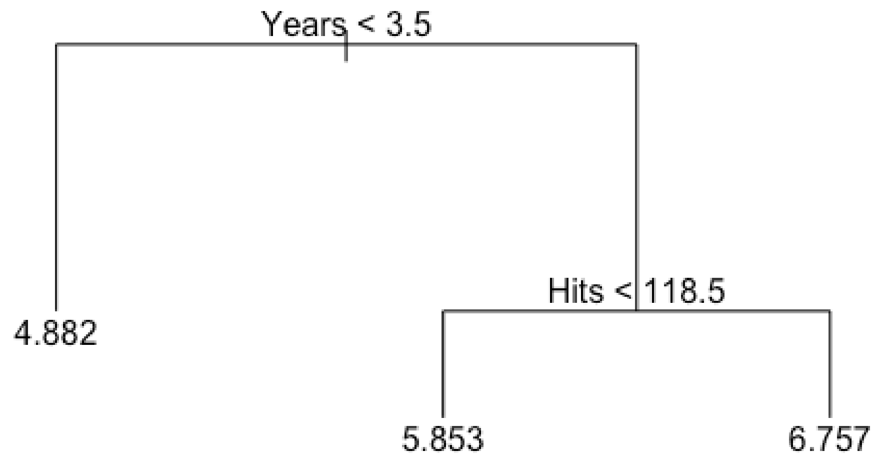



```
> cv.hitters=cv.tree(tree.hitters.train)
> plot(cv.hitters$size,cv.hitters$dev,type='b')
```

deviance: node RSS, summed over all nodes

# Example: Baseball Players' Salaries

- Cross Validation indicated that the minimum MSE is when the tree size is three (i.e. the number of leaf nodes is 3)



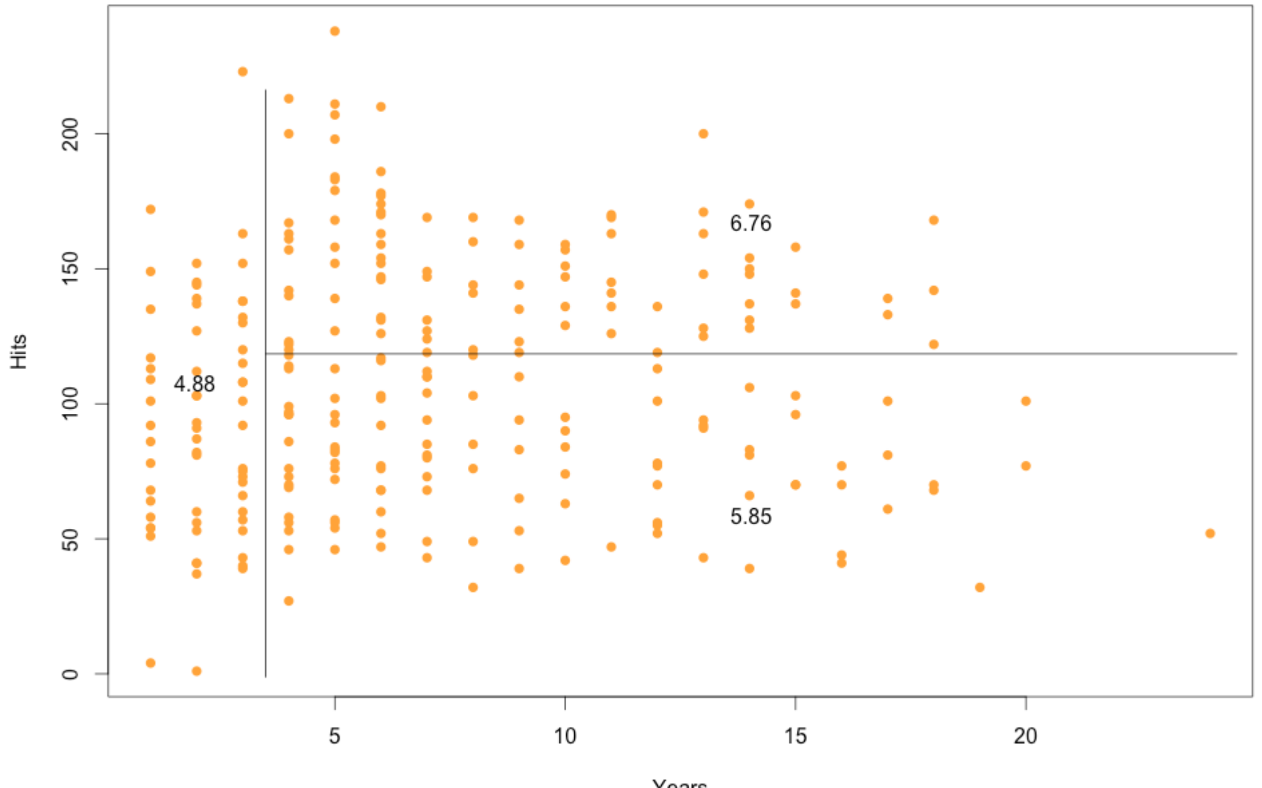

- Now, we prune the tree to be of size 3:

```
> prune.hitters=prune.tree(tree.hitters.train,best=3)
> plot(prune.hitters)
> text(prune.hitters,pretty=0)
```

# Another Way of Visualisation

This visualisation only works for one or two predictors.

```
> plot(Hitters$Years,Hitters$Hits, col="orange",pch=16,
xlab="Years",ylab="Hits")
>
partition.tree(prune.hitters,ordvars=c("Years","Hits"),add=TRUE)
```



# Making Predictions

Use the pruned tree to make predictions on the test set

```
> yhat=predict(prune.hitters,newdata=Hitters[-train,])
> hitters.test=log(Hitters[-train, "Salary"])

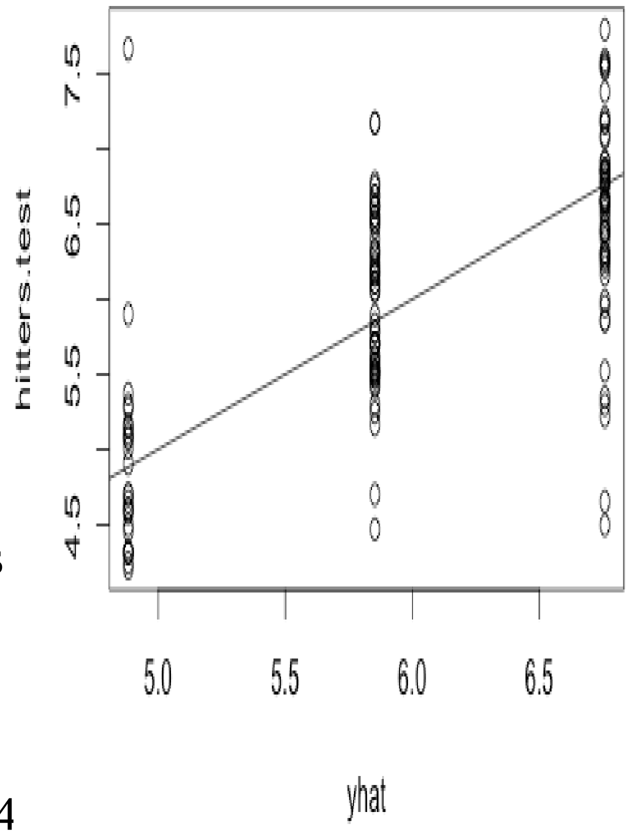
> plot(yhat,hitters.test)
> abline(0,1)

> mean((yhat-hitters.test)^2)
```

`[1] 0.4445136` 🡐 The test MSE for the regression tree

```
> sqrt(0.4445136)
```

`[1] 0.6667185` 🡐 The square root of the MSE, which means this model leads to test predictions that are within around 1000*e^0.6667185=1947.8 of the true Salary of hitters



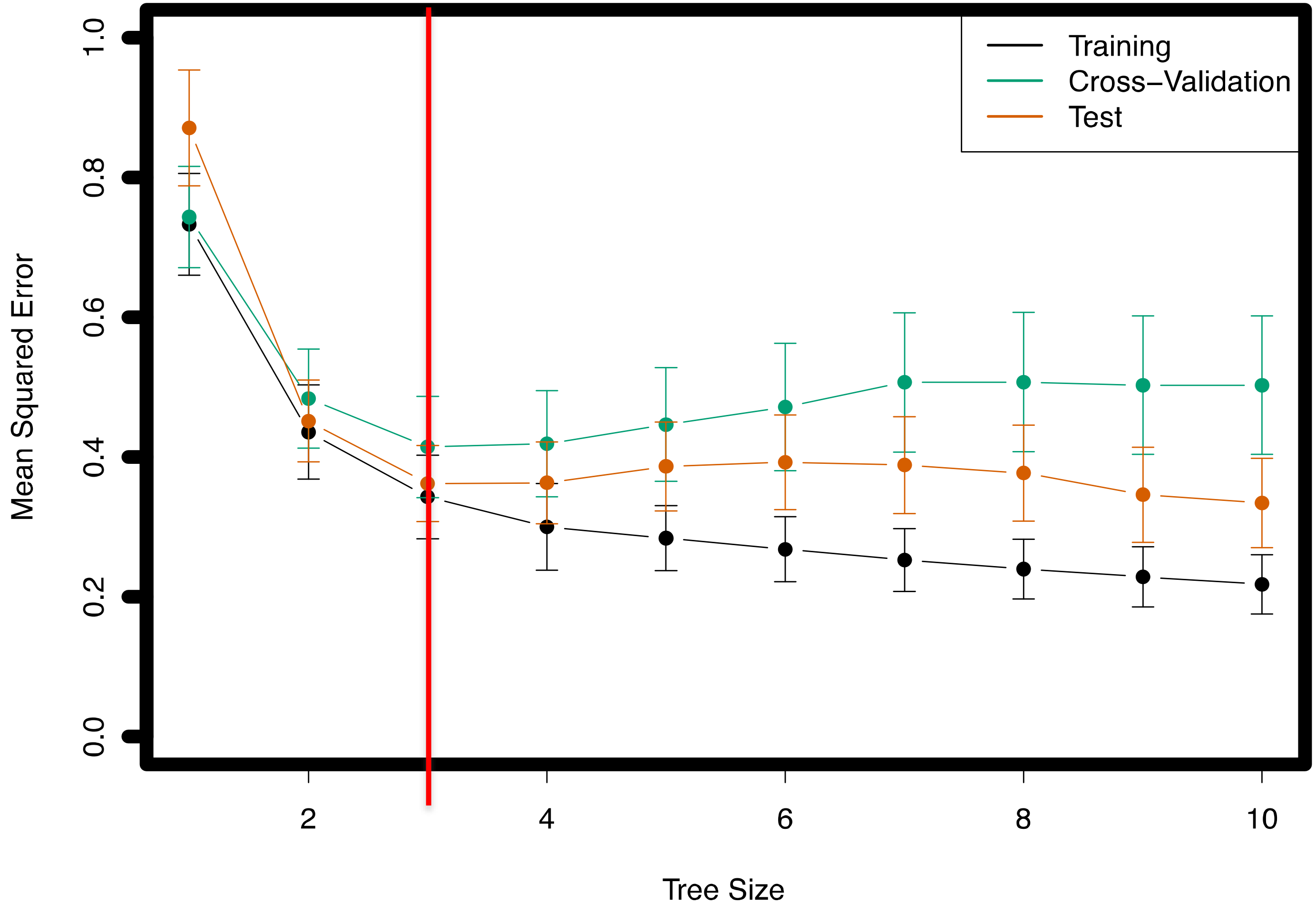

If using the unpruned tree to do the same, the MSE is 0.374624

```
>yhat.unpruned=predict(tree.hitters.train,newdata=Hitters
[-train,])
> mean((yhat.unpruned-hitters.test)^2)
```

Why the unpruned tree has a smaller MSE?

# Example: Baseball Players' Salaries

- **In the book**, with an unspecified seed to random split the data set, the minimum cross validation error occurs at a tree size of 3



# Classification Trees

Predicting a qualitative response

e.g., Predicting whether a customer will default,
whether an email is a spam, etc

# Growing a Classification Tree

- A classification tree is very similar to a regression tree except that we try to make a prediction for a categorical rather than continuous Y.
- For each region (or node) we predict the most common category among the training data within that region.
- The tree is grown (i.e. the splits are chosen) in exactly the same way as with a regression tree except that minimising MSE no longer makes sense.
- There are several possible different criteria to use such as the "gini index" and "cross-entropy" but the easiest one to think about is to minimise the error rate.

# Evaluation of Classification Models

- Recall: Counts of test records that are correctly (or incorrectly) predicted by the classification model

- **Confusion matrix**

| Actual Class \ Predicted Class | Class = 1 | Class = 0 |
|---|---|---|
| Class = 1 | $f_{11}$ | $f_{10}$ |
| Class = 0 | $f_{01}$ | $f_{00}$ |

$$\text{Accuracy} = \frac{\#\,\text{correct predictions}}{\text{total}\#\,\text{of predictions}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

$$\text{Error rate} = \frac{\#\,\text{wrong predictions}}{\text{total}\,\#\,\text{of predictions}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

# Example: Carseats

- Goal: to analyse the `Carseats` data set

```
> library(ISLR)
```

Sales, CompPrice, Income, Advertising, Population, Price, ShelveLoc, Age, Education, Urban, US

- `Sales` is a continuous variable, discretise it using `ifelse()`

```
> High=ifelse(Carseats$Sales<=8,"No","Yes")
```

- Merge `High` with the rest of the `Carseats` data

`> Carseats=data.frame(Carseats,High)` # add one more column `High` to Carseats dataset

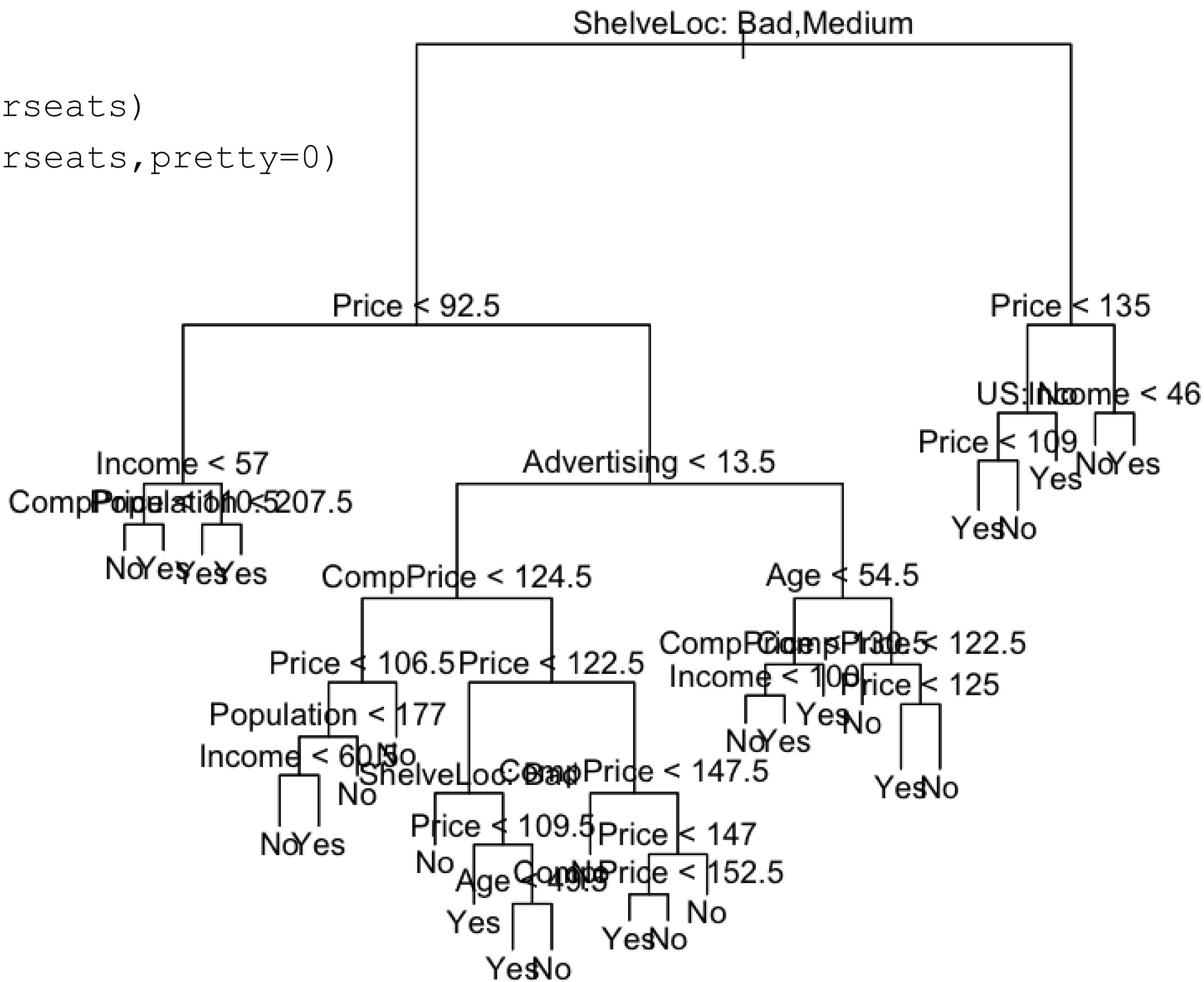
- Fitting a classification tree

```
> tree.carseats=tree(High~.-Sales,Carseats)
```

`> summary(tree.carseats)` #How many predictors are used?

# Plotting the tree

```
> plot(tree.carseats)
> text(tree.carseats,pretty=0)
```



# Test Error Rate Estimation

- Estimate the test error rather than computing the training error
  - Split the observations into a training set and a test set
  - Build the tree using the training set
  - Evaluate its performance on the test data
    - By `predict()`, where `type="class"` returns the actual class prediction

```
> set.seed(2)
> train=sample(1:nrow(Carseats),nrow(Carseats)/2)
> Carseats.test=Carseats[-train,]
> High.test=High[-train]
> tree.carseats.train=tree(High~.-Sales,Carseats,subset=train)
> tree.pred.test=predict(tree.carseats.train,Carseats.test,type="class")
```

```
> table(tree.pred.test,High.test)
              High.test
tree.pred.test No Yes
           No  86  27
           Yes 30  57
```

Test Error Rate is 28.5%

```
> (27+30)/200
[1] 0.285
```

# Calculate the Train Error Rate

- What is training error rate?

```
> High.train=High[train]

> tree.pred.train=predict(tree.carseats.train,Carseats[train,],type="class")

> table(tree.pred.train,High.train)
               High.train
tree.pred.train  No Yes
            No  114  12
            Yes   6  68
```

(12+6)/200=0.09

# Pruning a Tree

- Consider whether pruning the tree might lead to improved results

Step 1: Use `cv.tree()` to determine the optimal level of tree complexity

```
> set.seed(3)
> cv.carseats=cv.tree(tree.carseats.train,FUN=prune.misclass)
> cv.carseats
$size
[1] 19 17 14 13  9  7  3  2  1
$dev
[1] 55 55 53 52 50 56 69 65 80
```

$dev ← this is the cv error rate

Means we want the classification error rate to guide the CV and pruning process

Step 2: Use `prune.misclass()` to prune the tree

```
> prune.carseats=prune.misclass(tree.carseats.train,best=9)  #plot the tree here
```

Step 3: Performance evaluation

```
> tree.pred=predict(prune.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
         High.test
tree.pred No Yes
      No  94  24
      Yes 22  60
```

```
> (24+22)/200   test error rate
[1] 0.23
```

← better than that without pruning 28.5% (see 2 slides before)

Pruning improved interpretability and classification accuracy

# Pruning a Tree

- If we increase the value of best, we obtain a larger pruned tree with lower classification accuracy

```
> prune.carseats=prune.misclass(tree.carseats.train,best=15) #plot the tree
here

> tree.pred=predict(prune.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
         High.test
tree.pred No Yes
      No  86  22
      Yes 30  62
> (30+22)/200
[1] 0.26 > 0.23 (see previous slide)
```

# Summary: Decision Tree Induction

- Decision tree generation consists of two phases:
  - Tree construction
    - At start, all the training examples are at the root
    - Partition examples recursively based on selected attributes
  - Tree pruning
    - Identify and remove branches that reflect noise or outliers
- Use of decision tree:
  - Regressing or classifying an unknown sample
    - Test the attribute values of the sample against the decision tree

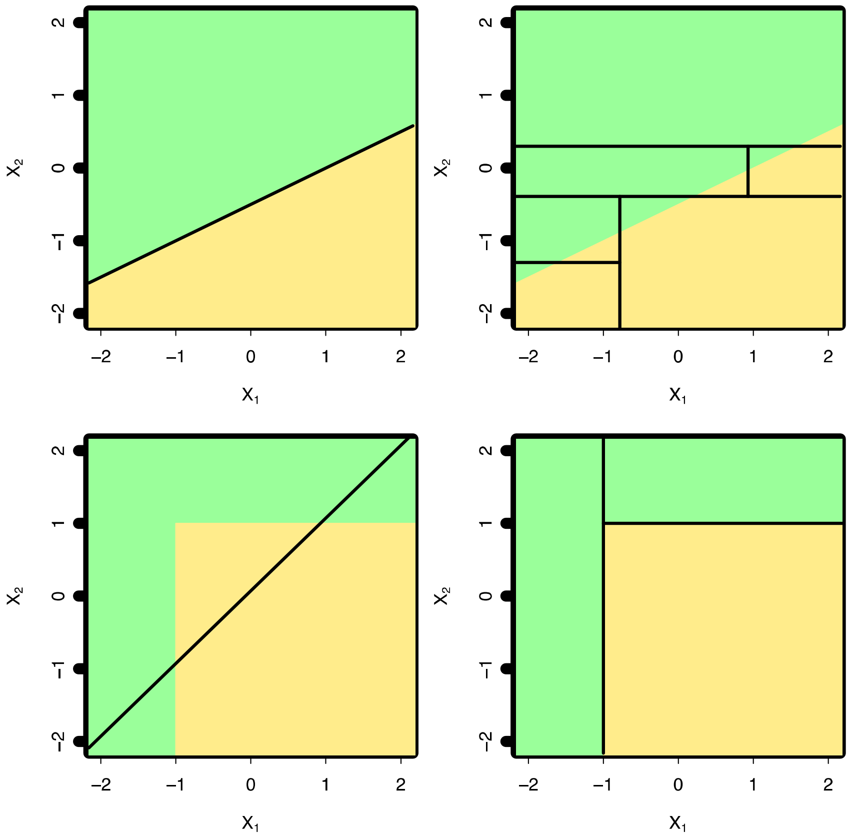
# Trees vs. Linear models

# Trees vs. Linear Models

- Which model is better?
  - If the relationship between the predictors and response is linear, then classical linear models such as linear regression would outperform regression trees
  - On the other hand, if the relationship between the predictors is non-linear, then decision trees would outperform classical approaches

# Trees vs. Linear Model: Classification Example

- Top row: the true decision boundary is linear
  - Left: linear model (good)
  - Right: decision tree
- Bottom row: the true decision boundary is non-linear
  - Left: linear model
  - Right: decision tree (good)



# Advantages and disadvantages of trees

# Pros and Cons of Decision Trees

- Pros:
  - Trees are very easy to explain to people (probably even easier than linear regression)
  - Trees can be plotted graphically, and are easily interpreted even by non-expert
  - They work fine on both classification and regression problems

- Cons:
  - Trees don’t have the same prediction accuracy as some of the more complicated approaches that we examine in this course

# LAB

# Exercises

- Fit a regression tree to the Boston house price data set
  - Create a training set and fit the tree to the training set
  - Plot the tree

  - Using `cv.tree()` function to see whether pruning the tree will improve performance
  - Plot the result of cv.tree() with sizes and deviances

  - Use the unpruned tree to make predictions on the test set
  - Calculate the test MSE and interpret the square root of the test MSE

  - Use the pruned tree instead and repeat the above procedure
  - Compare the two MSEs

# Fitting Classification Trees

- The `tree` library is used to construct classification and regression trees
  - Install this library if necessary (How?)

- Several key points:
  - Fitting a tree
  - Plotting a tree
  - Pruning a tree
  - Estimating test error of the fitting

# Fitting a Tree

- Goal: to analyse the `Carseats` data set

```
> library(ISLR)
> attach(Carseats)
```

- `Sales` is a continuous variable, discretise it using `ifelse()`

```
> High=ifelse(Sales<=8,"No","Yes")
```

- Merge `High` with the rest of the `Carseats` data

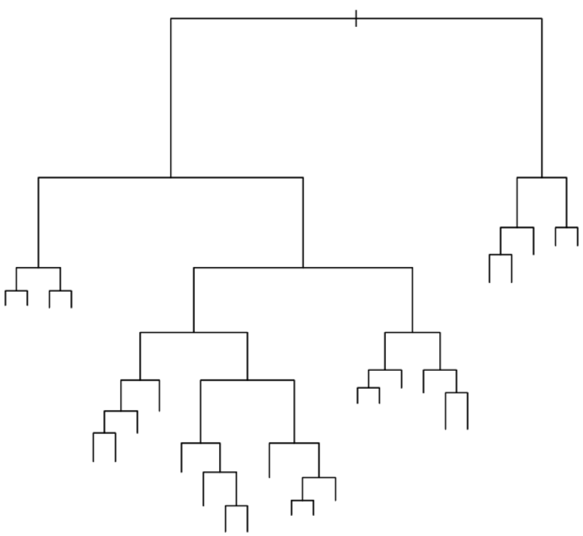
```
> Carseats=data.frame(Carseats,High)
```

- Fitting a classification tree

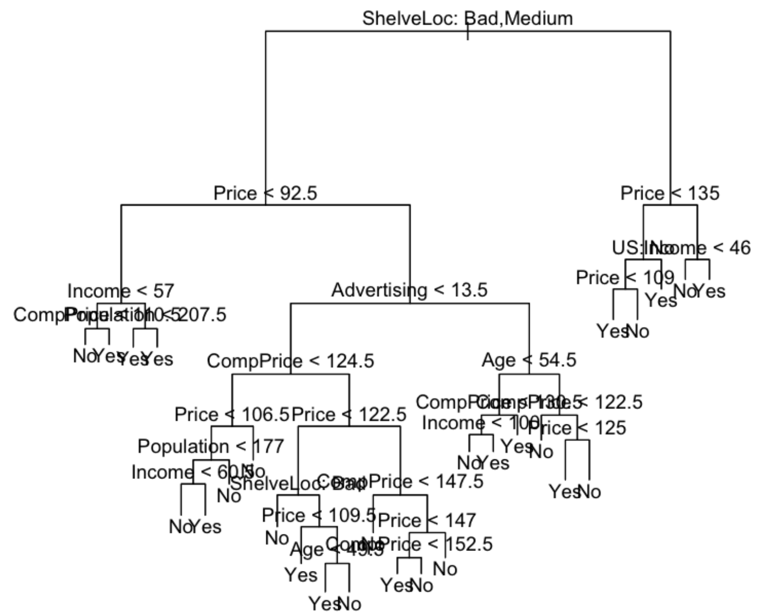
```
> tree.carseats=tree(High~.-Sales,Carseats)
> summary(tree.carseats)
```

# Plotting a Tree

- Trees can be naturally graphically displayed
  - `plot()` to display structure
  - `text()` to display the node labels
    - `pretty=0` includes the category names for any qualitative predictors, rather than simply displaying a letter for each category



`> plot(tree.carseats)`




`> text(tree.carseats,pretty=0)`

# Showing More Info

```
> tree.carseats
node), split, n, deviance, yval, (yprob)      * denotes terminal node

  1) root 400 541.500 No ( 0.59000 0.41000 )
    2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
      4) Price < 92.5 46  56.530 Yes ( 0.30435 0.69565 )
        8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
         16) CompPrice < 110.5 5   0.000 No ( 1.00000 0.00000 ) *
         17) CompPrice > 110.5 5   6.730 Yes ( 0.40000 0.60000 ) *
        9) Income > 57 36  35.470 Yes ( 0.19444 0.80556 )
         18) Population < 207.5 16  21.170 Yes ( 0.37500 0.62500 ) *
         19) Population > 207.5 20   7.941 Yes ( 0.05000 0.95000 ) *
      5) Price > 92.5 269 299.800 No ( 0.75465 0.24535 )
       10) Advertising < 13.5 224 213.200 No ( 0.81696 0.18304 )
         20) CompPrice < 124.5 96  44.890 No ( 0.93750 0.06250 )
           40) Price < 106.5 38  33.150 No ( 0.84211 0.15789 )
             80) Population < 177 12  16.300 No ( 0.58333 0.41667 )
              160) Income < 60.5 6   0.000 No ( 1.00000 0.00000 ) *
              161) Income > 60.5 6   5.407 Yes ( 0.16667 0.83333 ) *
             81) Population > 177 26   8.477 No ( 0.96154 0.03846 ) *
           41) Price > 106.5 58   0.000 No ( 1.00000 0.00000 ) *
         21) CompPrice > 124.5 128 150.200 No ( 0.72656 0.27344 )
```

# Test Error Rate Estimation

- Estimate the test error rather than computing the training error
  - Split the observations into a training set and a test set
  - Build the tree using the training set
  - Evaluate its performance on the test data
    - By `predict()`, where `type="class"` returns the actual class prediction

```
> set.seed(2)
> train=sample(1:nrow(Carseats),200)
> Carseats.test=Carseats[-train,]
> High.test=High[-train]
> tree.carseats=tree(High~.-Sales,Carseats,subset=train)
> tree.pred=predict(tree.carseats,Carseats.test,type="class")

> table(tree.pred,High.test)
         High.test
tree.pred No Yes
      No  86  27
      Yes 30  57
> (86+57)/200
[1] 0.715
```

# Pruning a Tree

- Consider whether pruning the tree might lead to improved results

Step 1: Use `cv.tree()` to determine the optimal level of tree complexity

```
> set.seed(3)
> cv.carseats=cv.tree(tree.carseats,FUN=prune.misclass)
> cv.carseats
$size
[1] 19 17 14 13  9  7  3  2  1
$dev    ← this is the cv error rate
[1] 55 55 53 52 50 56 69 65 80
```

Step 2: Use `prune.misclass()` to prune the tree

```
> prune.carseats=prune.misclass(tree.carseats,best=9)
```

Step 3: Performance evaluation

```
> tree.pred=predict(prune.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
         High.test
tree.pred No Yes
      No  94  24
      Yes 22  60
```

```
> (94+60)/200
[1] 0.77   ← better than that without pruning
```

# Fitting Regression Trees

- Fit a regression tree to the Boston data
  - Create a training set and fit the tree to the training set

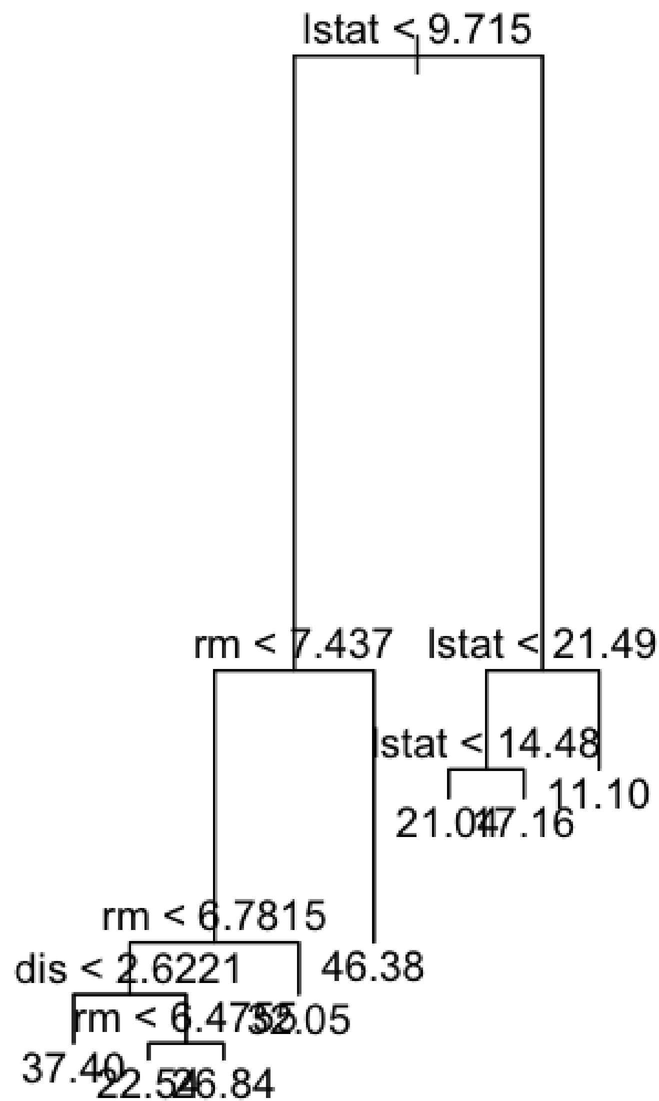
```
> library(MASS)
> set.seed(1)
> train=sample(1:nrow(Boston),nrow(Boston)/2)
> tree.boston=tree(medv~.,Boston,subset=train)

> summary(tree.boston)
Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "lstat" "rm"    "dis"  ← only 3 variables have been used
Number of terminal nodes:  8
Residual mean deviance:  12.65 = 3099 / 245
Distribution of residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-14.10000  -2.04200  -0.05357   0.00000   1.96000  12.60000
```

# Fitting Regression Trees

- Plot the tree

```
> plot(tree.boston)
> text(tree.boston,pretty=0)
```



# Fitting Regression Trees

– Using `cv.tree()` function to see whether pruning the tree will improve performance

```
> cv.boston=cv.tree(tree.boston)
> cv.boston
$size
[1] 8 7 6 5 4 3 2 1

$dev   min
[1]  5226.322  5228.360  6462.626  6692.615  6397.438  7529.846 11958.691 21118.139

$k
[1]      -Inf  255.6581  451.9272  768.5087  818.8885 1559.1264 4276.5803 9665.3582

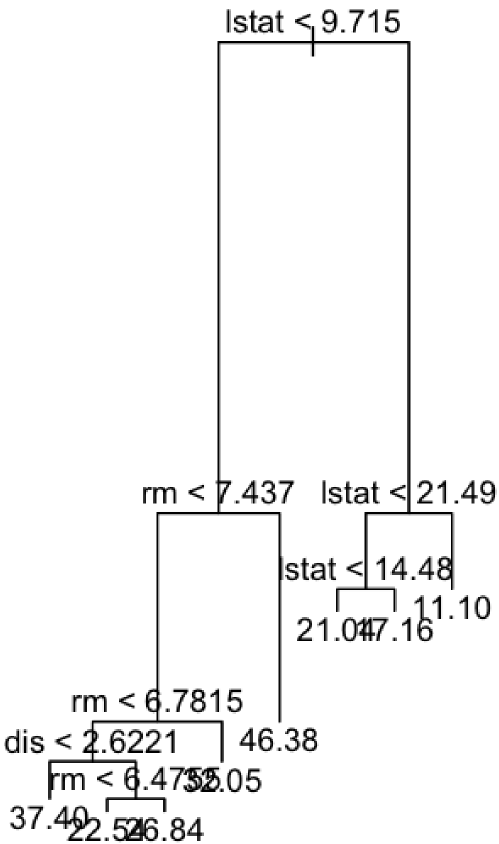
$method
[1] "deviance"

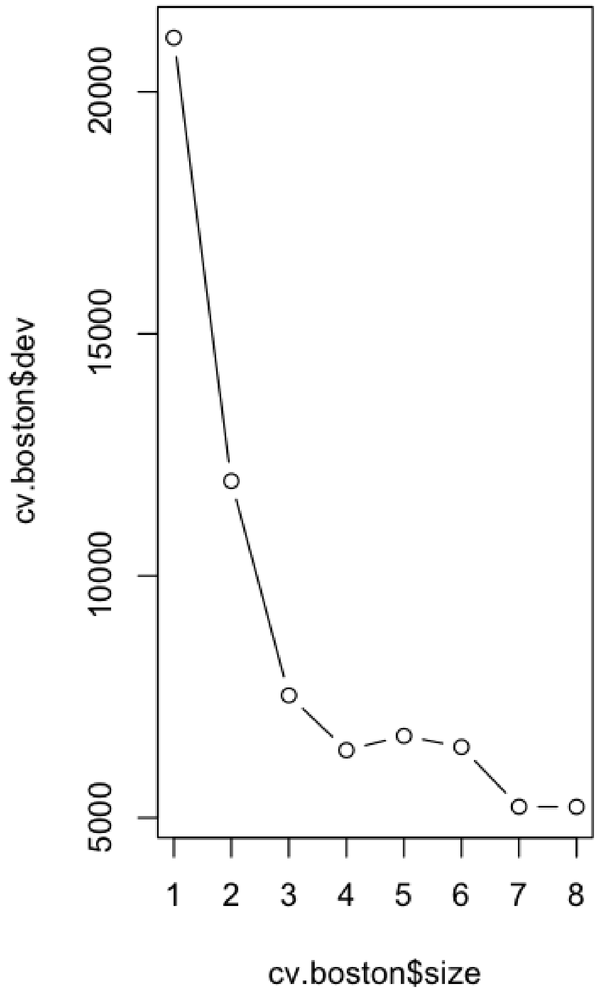
attr(,"class")
[1] "prune"         "tree.sequence”
```

The result shows that the best tree is the one with 8 terminals. ➔ No need to prune.

# Fitting Regression Trees

- ```
  > plot(cv.boston$size,cv.boston$dev,type='b')
  ```
- The graph is the same as plot(tree.boston)





  ```

# Fitting Regression Trees

Use the unpruned tree to make predictions on the test set

```
> yhat=predict(tree.boston,newdata=Boston[-train,])
> boston.test=Boston[-train,"medv"]

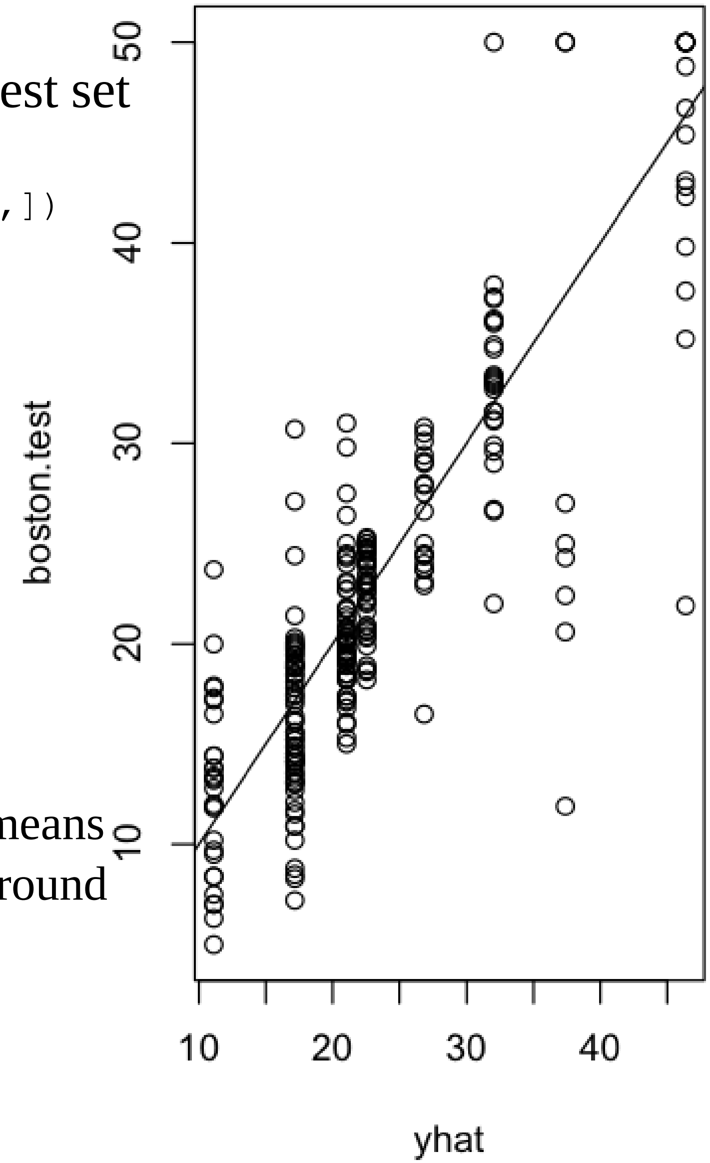
> plot(yhat,boston.test)
> abline(0,1)

> mean((yhat-boston.test)^2)
```

`[1] 25.04559` ← The test set MSE associated with the regression tree

```
> sqrt(25.04559)
```

`[1] 5.004557` ← The square root of the MSE, which means that this model leads to test predictions that are within around $5,005 of the true median home value for the suburb



## Example: Baseball Players' Salaries

- Cross Validation indicated that the minimum MSE is when the tree size is three (i.e. the number of leaf nodes is 3)

- Now, we prune the tree to be of size 3:

```
> prune.hitters.3=prune.tree(tree.hitters.train,best=3)
> plot(prune.hitters.3)
> text(prune.hitters.3,pretty=0)
```

## How to plot this?

```
> plot(Hitters$Years,Hitters$RBI,col="orange",pch=16,xlab="Years",ylab="RBI")
> partition.tree(prune.hitters,ordvars=c("Years","RBI"),add=TRUE)
```