

# Data Science

## CS301

### Decision Trees

### In Code

Week 12

Fall 2024

Oliver BONHAM-CARTER

**Are you here today?!**

**ATTENDANCE**

<https://forms.gle/iaY7zBmxj8KvsDMa8>



# Decision Tree Code

```
#Decision Trees :: kyphorsis data

rm(list = ls()) # clear out unused variables from memory
graphics.off() # clear out all plots from previous work.
cat("\014") # clear the console

if(!require('rpart')) {
  install.packages('rpart')
  library('rpart')
}
if(!require('rpart.plot')) {
  install.packages('rpart.plot')
  library('rpart.plot')
}
```



# Trees: Two Variables

```
# Create three models using the variable Start to test
fit_start1 <- rpart(Kyphosis ~ Start, data = kyphosis)
fit_start2 <- rpart(Kyphosis ~ Start, data = kyphosis,
                    parms = list(prior = c(.65,.35), split = "information"))
fit_start3 <- rpart(Kyphosis ~ Start, data = kyphosis,
                    control = rpart.control(cp = 0.05))

# par() is for formatting text to prevent cut-offs, two columns
par(mfrow = c(1,3), xpd = NA)

plot(fit_start1)
text(fit_start1, use.n = TRUE)

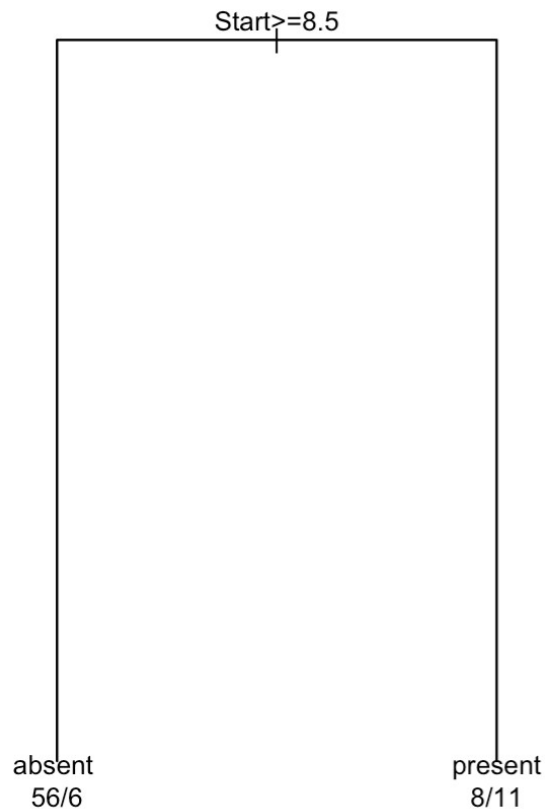
plot(fit_start2)
text(fit_start2, use.n = TRUE)

plot(fit_start3)
text(fit_start3, use.n = TRUE)
```

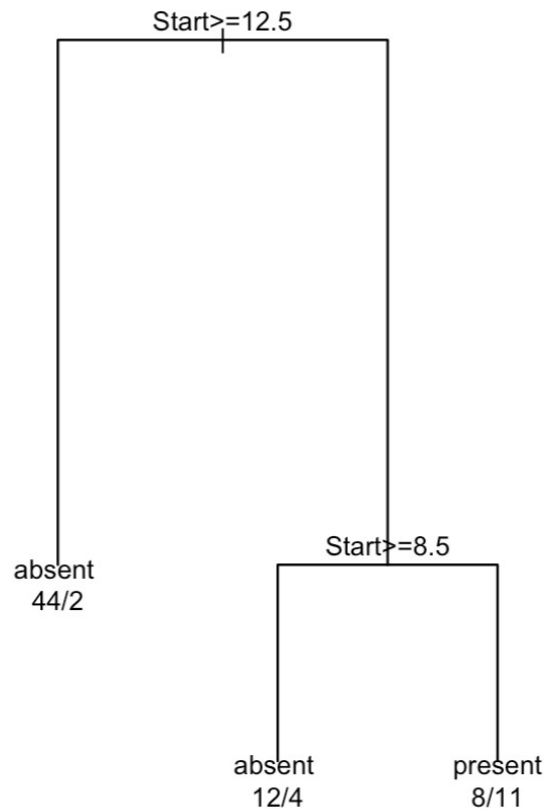


# Comparing Plots: Two Variables

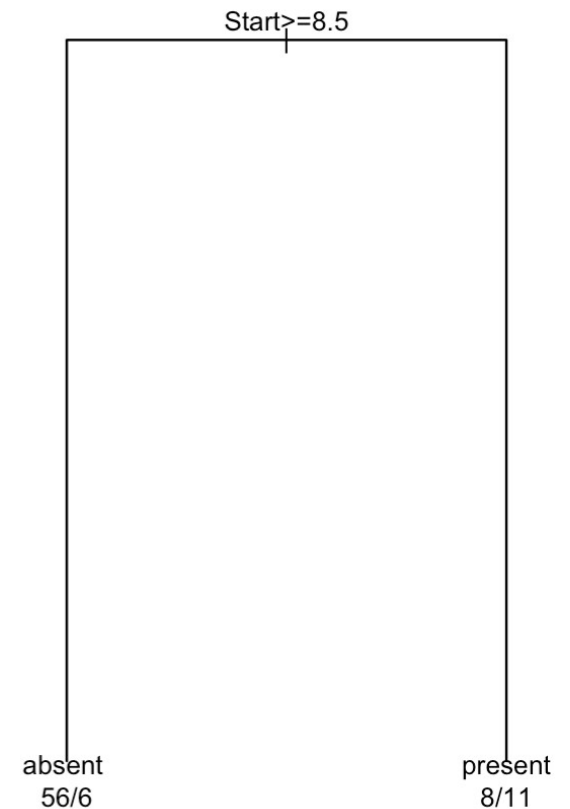
Model:  
fit\_start1



Model:  
fit\_start2



Model:  
fit\_start3





# Trees: All Variables

```
View(kyphosis)
```

```
#Create three models to test
```

```
fit <- rpart(Kyphosis ~ Age + Number + Start, data = kyphosis)
```

```
fit2 <- rpart(Kyphosis ~ Age + Number + Start, data = kyphosis,  
             parms = list(prior = c(.65,.35), split = "information"))
```

```
fit3 <- rpart(Kyphosis ~ Age + Number + Start, data = kyphosis,  
             control = rpart.control(cp = 0.05))
```

```
# par() is for formatting text to prevent cut-offs
```

```
par(mfrow = c(1,2), xpd = NA)
```

```
plot(fit)
```

```
text(fit, use.n = TRUE)
```

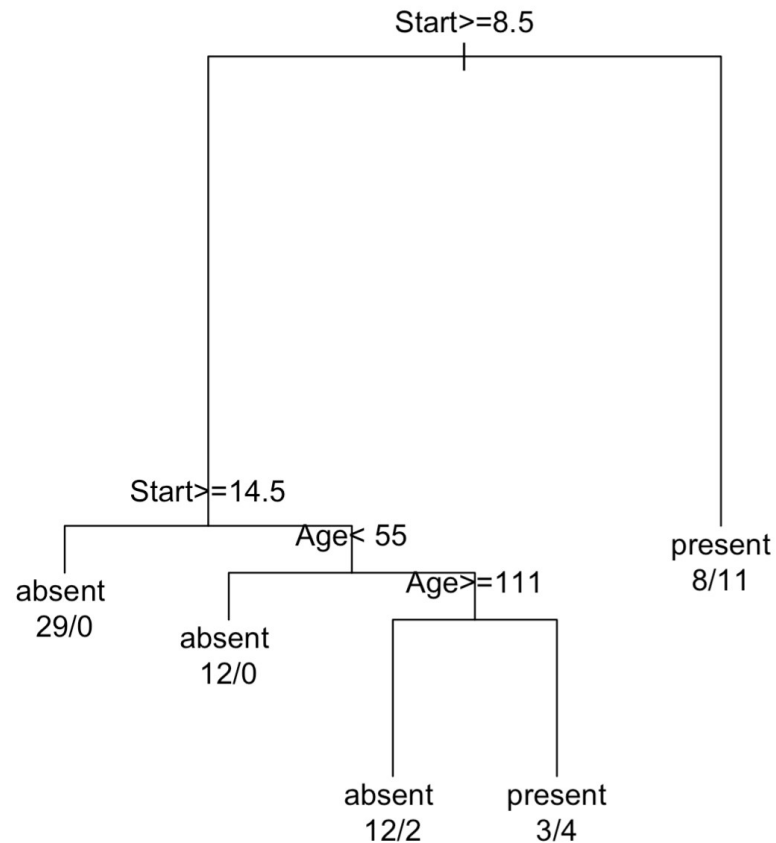
```
plot(fit2)
```

```
text(fit2, use.n = TRUE)
```

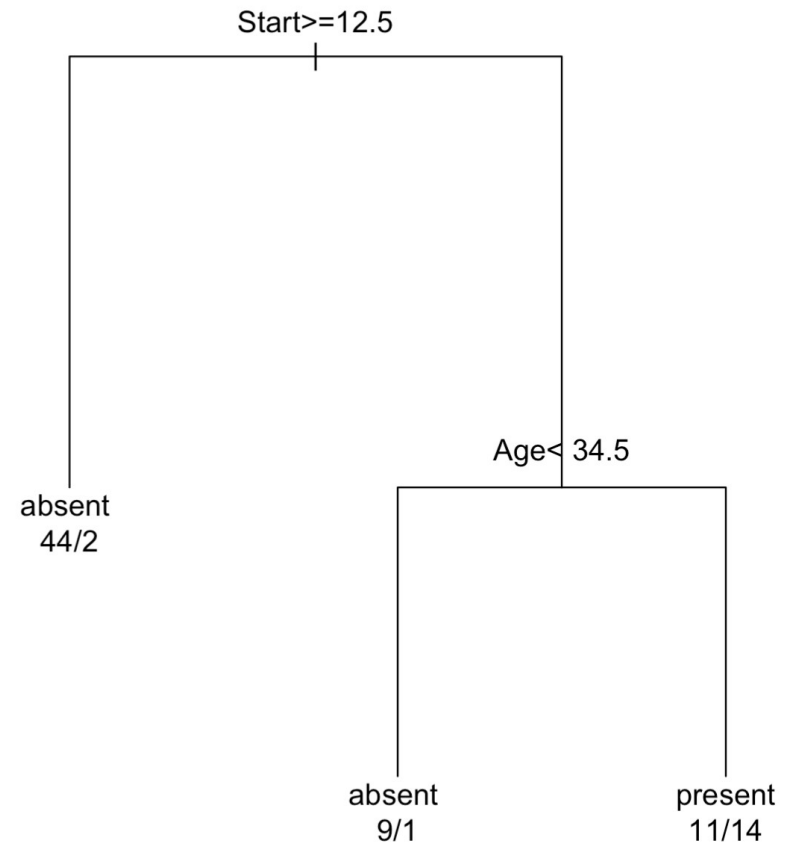


# Tree Plots: All Variables

Model: Fit



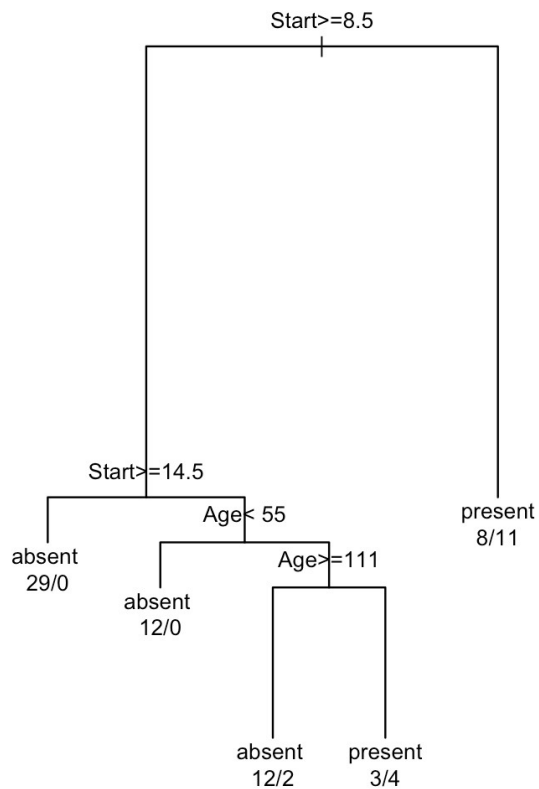
Model: Fit2



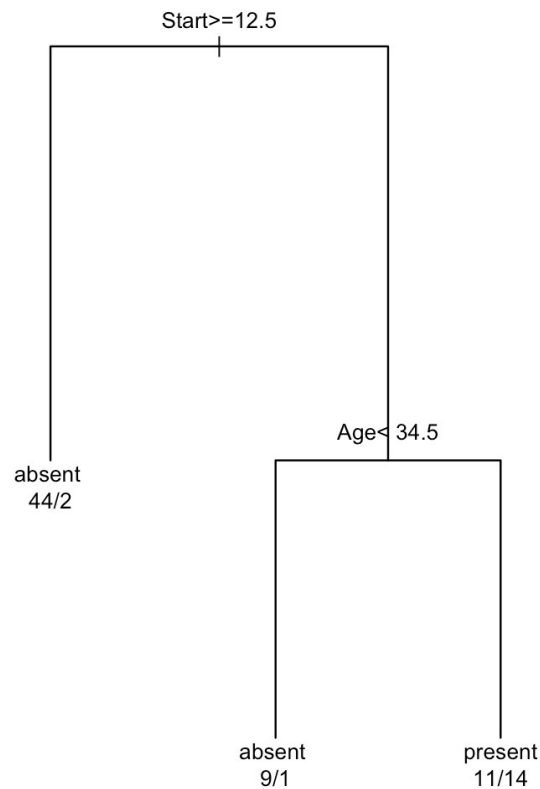


# Comparing Plots: All Variables

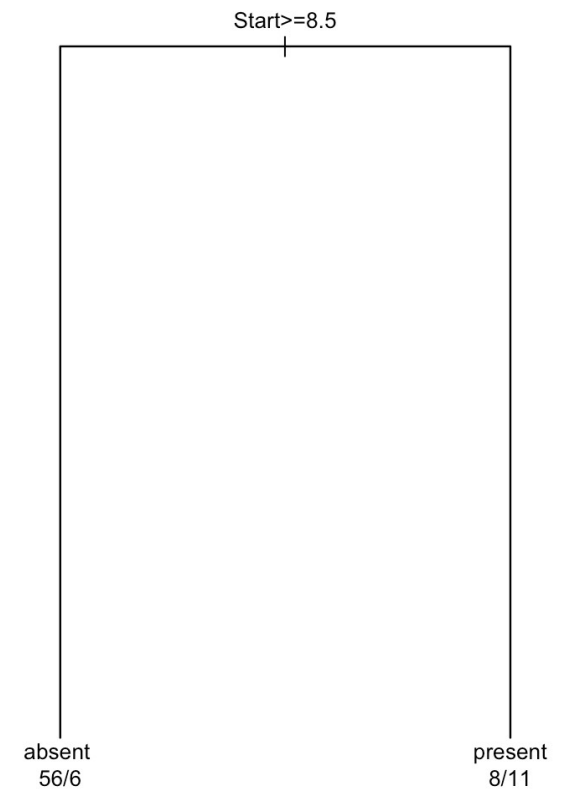
Model: Fit



Model: Fit2



Model: Fit3





# Running this Code (Outside of RStudio!)

It is often very useful to be able to run R code as a script in a unix environment.

## Unix and MacOS:

Rscript filename.R

Example:

Rscript decisionTree\_kyphosis.R

## Windows:

Path/Rscript.exe filename.R

Example:

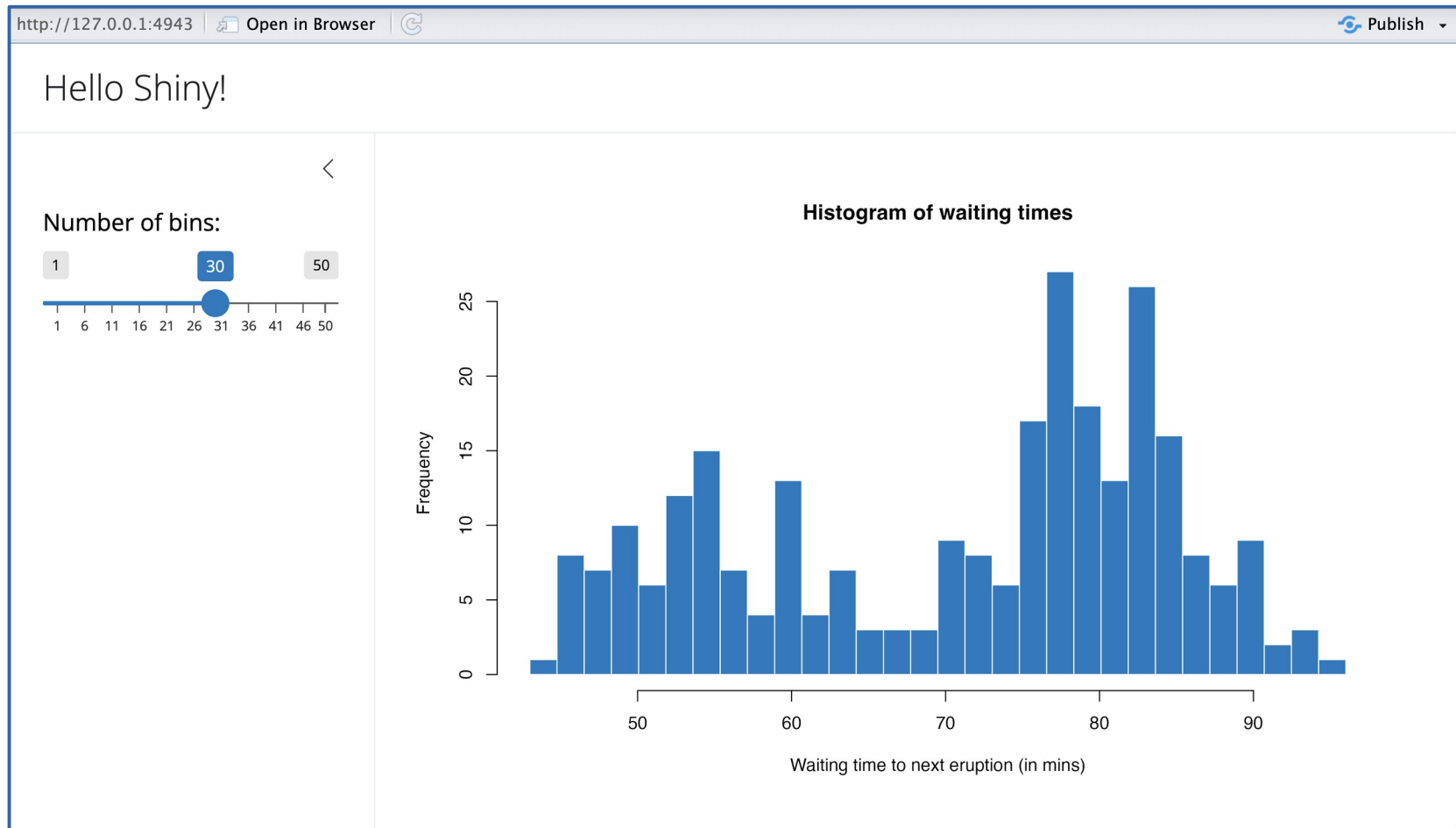
Ex: Path/Rscript.exe decisionTree\_kyphosis.R

Note: Outputs are generated and stored as a PDF file.



# Shiny App

## (A Dashboard for Analysis)



Learn more about making Interactive Shiny apps

<https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/index.html>

# Shiny App (A Dashboard for Analysis)

- Dashboards are interactive pieces of software to allow users to manipulate the data without using code.
- Run quick experiments with built-in datasets
- Demonstrate proof-of-concept for finding signals.

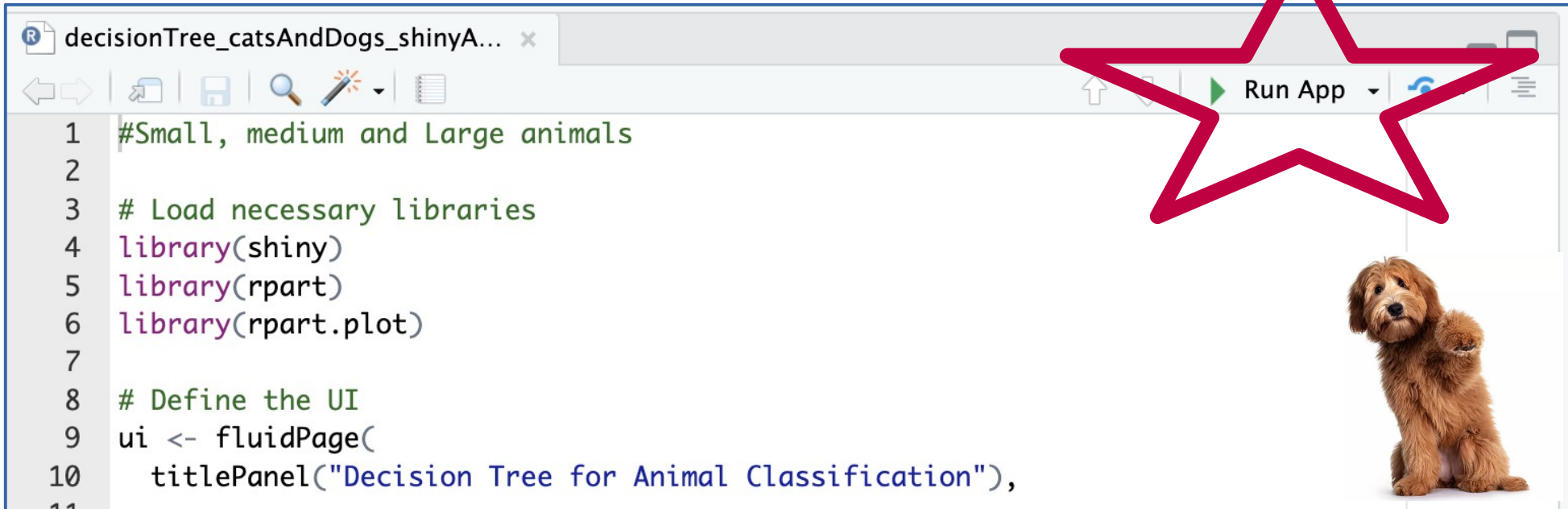


Let's check out a shiny application dashboard to play with decision trees!



# Executing Shiny Apps

## Animal Classification



```
1 #Small, medium and Large animals
2
3 # Load necessary libraries
4 library(shiny)
5 library(rpart)
6 library(rpart.plot)
7
8 # Define the UI
9 ui <- fluidPage(
10   titlePanel("Decision Tree for Animal Classification"),
11
```



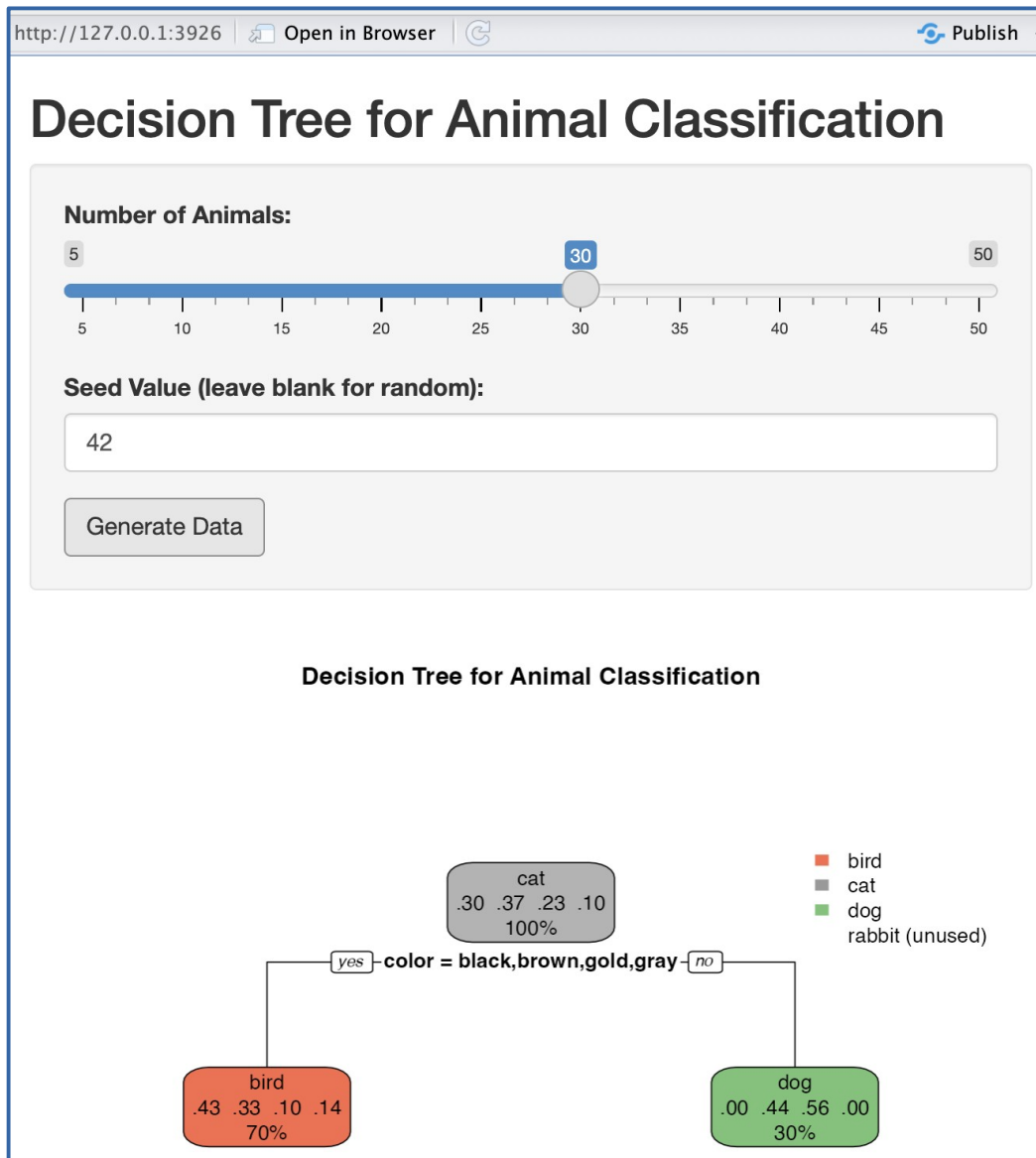
- Load your file in RStudio, click “Run App” to start dashboard.
- Run using Rstudio or in a browser tab
  - Listening on `http://127.0.0.1:3926`

File: decisionTree\_catsAndDogs\_shinyApp.R





# Execute and Output



Manipulate the size of rows  
in dataset

Create synthetic data  
according to random seed

Produce the tree from a  
model

File: decisionTree\_catsAndDogs\_shinyApp.R



# Let's Talk About The Data

Use Categorical Data

```
sizes <- c("small", "medium", "large")  
colors <- c("brown", "black", "white", "gray", "gold", "orange")  
animals <- c("cat", "dog", "rabbit", "bird")
```

Creating a data frame

```
data.frame(  
  size = sample(sizes, input$numAnimals, replace = TRUE),  
  color = sample(colors, input$numAnimals, replace = TRUE),  
  type = sample(animals, input$numAnimals, replace = TRUE)
```



# Qualities of Data (Synthetic)

- **size** can take values like *small*, *medium*, or *large*
- **color** can take values like *brown*, *black*, *white*, etc.
- **type** (the animal type) is still randomly assigned from a list of animal types like *cat*, *dog*, *rabbit*, and *bird*

size	color	type
small	white	cat
small	orange	cat
small	black	rabbit
small	gray	bird
medium	gray	cat
medium	orange	dog
medium	black	cat
small	gold	cat

Subset of Data



# Objectives of the Model

- If tree shows animals with;
  - **size == small** are mostly classified as **cats**
  - **color == black** are classified as **dogs**,
  - then you can interpret that the **size** and **color** features help in classifying the animals into the correct type.
- The tree may show splits like *size == medium* or *color == white*, and these will influence how animals are classified into different types.

<i>size</i>	<i>color</i>	<i>type</i>
small	brown	cat
large	black	dog
medium	white	rabbit
medium	gray	dog
small	orange	rabbit
large	brown	dog





# The Decision Tree Model

<http://127.0.0.1:3926>

[Open in Browser](#)



[Publish](#) ▾

Call:

```
rpart(formula = type ~ size + color, data = data)
n= 30
```

	CP	nsplit	rel error	xerror	xstd
1	0.1578947	0	1.0000000	1.157895	0.1274800
2	0.0100000	1	0.8421053	1.052632	0.1358942

Variable importance

```
color
100
```

Node number 1: 30 observations, complexity param=0.1578947

predicted class=cat expected loss=0.6333333 P(node) =1

class counts: 9 11 7 3

probabilities: 0.300 0.367 0.233 0.100

left son=2 (21 obs) right son=3 (9 obs)

Primary splits:

color splits as LLLLRR, improve=2.6984130, (0 missing)

size splits as LRR, improve=0.3519669, (0 missing)

Node number 2: 21 observations

predicted class=bird expected loss=0.5714286 P(node) =0.7

class counts: 9 7 2 3

probabilities: 0.429 0.333 0.095 0.143

Node number 3: 9 observations

predicted class=dog expected loss=0.4444444 P(node) =0.3

class counts: 0 4 5 0

probabilities: 0.000 0.444 0.556 0.000

The “call” or the creation of the model

Complexity Parameters (CP) Table for Model Evaluation

The nodes and splits of the tree in textual form

# The Decision Tree Model



Call:

```
rpart(formula = type ~ size + color, data = data)
n= 30
```

	CP	nsplit	rel error	xerror	xstd
1	0.1578947	0	1.0000000	1.157895	0.1274800
2	0.0100000	1	0.8421053	1.052632	0.1358942

- The decision tree (`rpart`) is now created with both *size* and *color* as predictors: *type ~ size + color*.
- This means that the decision tree will use both attributes to predict the animal type
- **The decision tree will split on both the size and color features to classify the animals**



# Output Table

## Complexity Parameter (CP)

Call:

```
rpart(formula = type ~ size + color, data = data)  
n= 30
```



	CP	nsplit	rel error	xerror	xstd
1	0.1578947	0	1.0000000	1.157895	0.1274800
2	0.0100000	1	0.8421053	1.052632	0.1358942

- A tree complexity allowance: The CP value is the minimum improvement in the model's performance required to make a split at any given node.
- **Higher CP values imply simpler trees (with fewer splits), but may underfit the data.**
- **Lower CP values imply more complex trees (with more splits) but may overfit the data.**
- The CP values in the table represent the thresholds for pruning the tree. The table shows how the tree changes at different levels of pruning, which helps to control **overfitting** (too fixed on this particular dataset) and **underfitting** (not enough training).

# Output Table

## nsplit

Call:

```
rpart(formula = type ~ size + color, data = data)  
n= 30
```



	CP	nsplit	rel error	xerror	xstd
1	0.1578947	0	1.0000000	1.157895	0.1274800
2	0.0100000	1	0.8421053	1.052632	0.1358942

- The *nsplit* value indicates the number of splits (or internal nodes) in the tree at that pruning level.
- Represents the number of decision points the tree has made up until that stage.
- Note: Prune the tree by increasing the CP means that the number of splits will decrease and results in a simpler model.
- In the given example:
  - **When CP = 0.1578947, there are no splits (nsplit = 0).**
  - **When CP = 0.01, there is one split (nsplit = 1).**

# Output Table

## Relative Error

Call:

```
rpart(formula = type ~ size + color, data = data)  
n= 30
```



	CP	nsplit	rel error	xerror	xstd
1	0.1578947	0	1.0000000	1.157895	0.1274800
2	0.0100000	1	0.8421053	1.052632	0.1358942

- The **Relative Error** is the relative error of a model at a specific pruning level compared to the unpruned tree.
  - Calculated as squared errors at that level divided by total squared errors for the root
- **Lower rel. error indicates a better fit to data, implying a more accurate model (e.g., 1 at no pruning and decreasing with each split)**

# Output Table

## xerror (Cross-Validation Error)

Call:

```
rpart(formula = type ~ size + color, data = data)  
n= 30
```



	CP	nsplit	rel error	xerror	xstd
1	0.1578947	0	1.0000000	1.157895	0.1274800
2	0.0100000	1	0.8421053	1.052632	0.1358942


- xError is a cross-validation estimate used to gauge the model's performance on unseen data, calculated using  $k$ -fold cross-validation (usually 10-fold)
- **Lower xError values suggest better generalization to unseen data (e.g., higher at no splits and decreasing with each split).**

# Output Table

## xstd (Standard Deviation of Cross-Validation Error)

Call:

```
rpart(formula = type ~ size + color, data = data)  
n= 30
```



	CP	nsplit	rel error	xerror	xstd
1	0.1578947	0	1.0000000	1.157895	0.1274800
2	0.0100000	1	0.8421053	1.052632	0.1358942

- xstd is the standard deviation of the cross-validation error estimate, representing the variability of the error across different validation folds.
- **Smaller xstd values indicate consistent model performance across different data subsets.**
- **Lower xstd at no splits and slightly higher after one split in the given table.**



# Overfitting / UnderFitting?

Data

+

Tree with too  
many splits

-->

Tree becomes too  
specific for this dataset,  
not sensitive for  
general data

- If the tree is too complex (too many splits), it may **overfit** the data.
- The tree memorizes the training dataset's *specific* characteristics rather than learning the underlying patterns.
  - Tree may perform very well on the training data
  - Tree performs poorly on unseen data
- If the tree is too simple (too few splits), it may **underfit** the data. This means the tree is too general and may miss important patterns in the data.





# GO PLAY!

## Learn more about Decision Trees!

[https://www.tutorialspoint.com/r/r\\_decision\\_tree.htm](https://www.tutorialspoint.com/r/r_decision_tree.htm)

## Make a Shiny App!

<https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/index.html>



Play with the Decision  
Trees using the Titanic  
dataset!

File: titanic\_shinyApp.R