Introduction to R & data

Quick intro: who are you?

Jonathan de Bruin
research data engineer @ ITS
j.debruin1@uu.nl

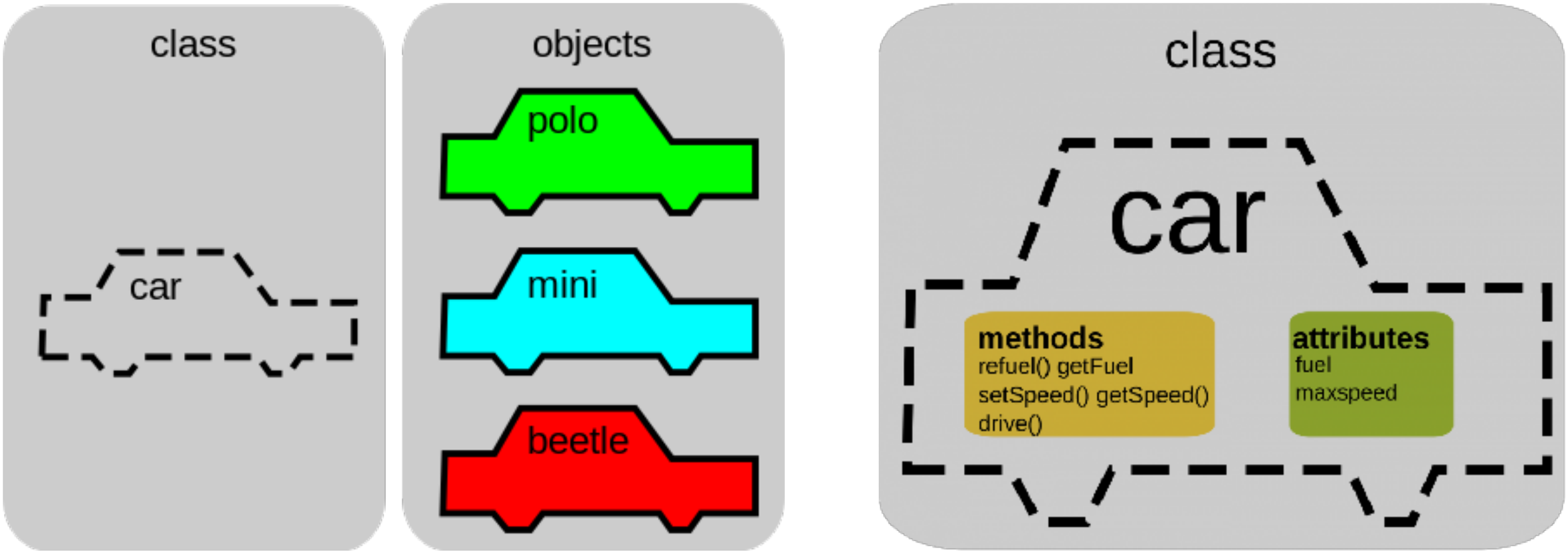
Barbara Vreede
subject specialist Science @ UU Library
b.m.i.vreede@uu.nl

Introduction to R and RStudio

What is R?

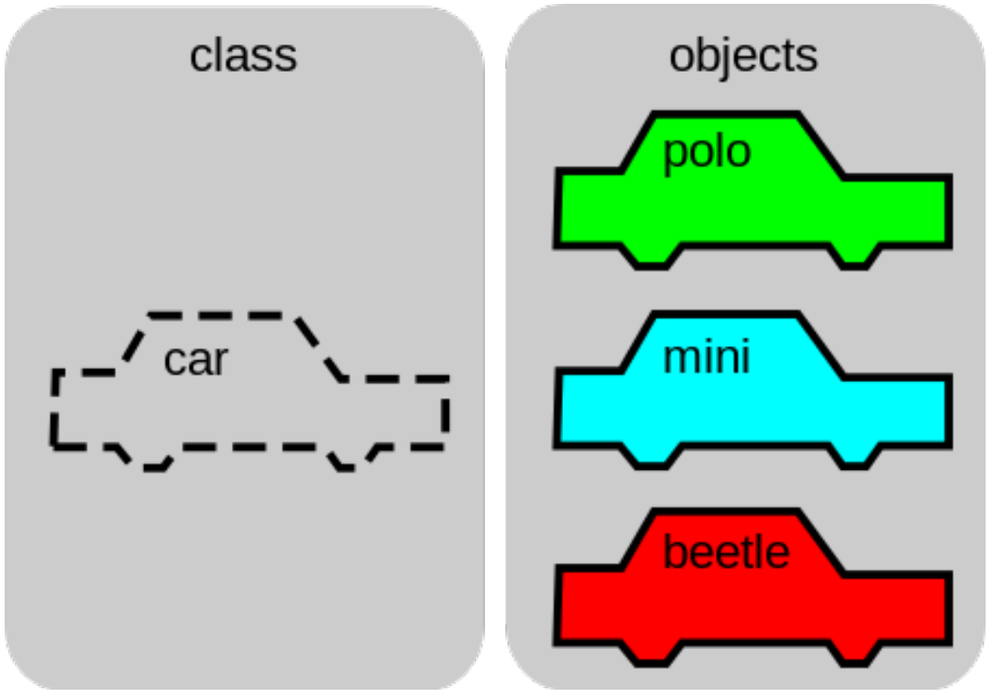
- Widely used programming language for data analysis
- Based on statistical programming language **S** (1976)
- Developed by **R**oss Ihaka & **R**obert Gentleman (1995)
- Very active community, with many (often subject-specific) packages
- Interpreted language (direct communication, no compiling!)
- Object-oriented

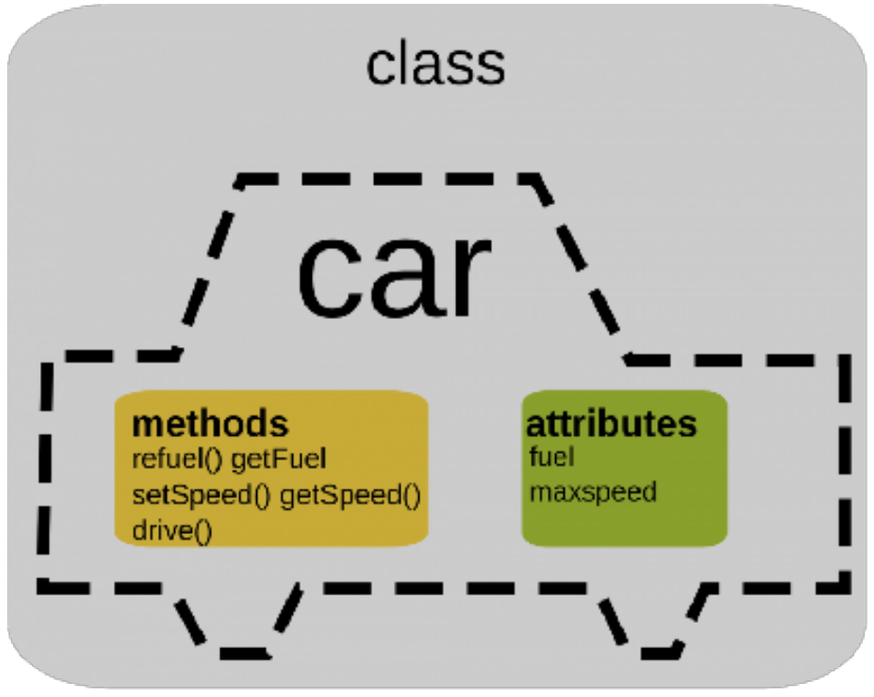
Object oriented programming



Today you will meet…

- **Classes**: logicals, numerics, characters, vectors, lists, data frames…
- **Objects** in these classes, and their **attributes**
- **Methods**: both *generic* and *class-specific* functions to process and question the objects





We will work in RStudio

- Integrated Development Environment for R
- Founded by JJ Allaire, available since 2010
- Bloody useful! Let’s take a look: please open RStudio!

R syntax & the RStudio console

Declaring variables: many ways to skin a cat!

```
> x <- 1
```

```
> 1 -> x
```

```
> x = 1
```

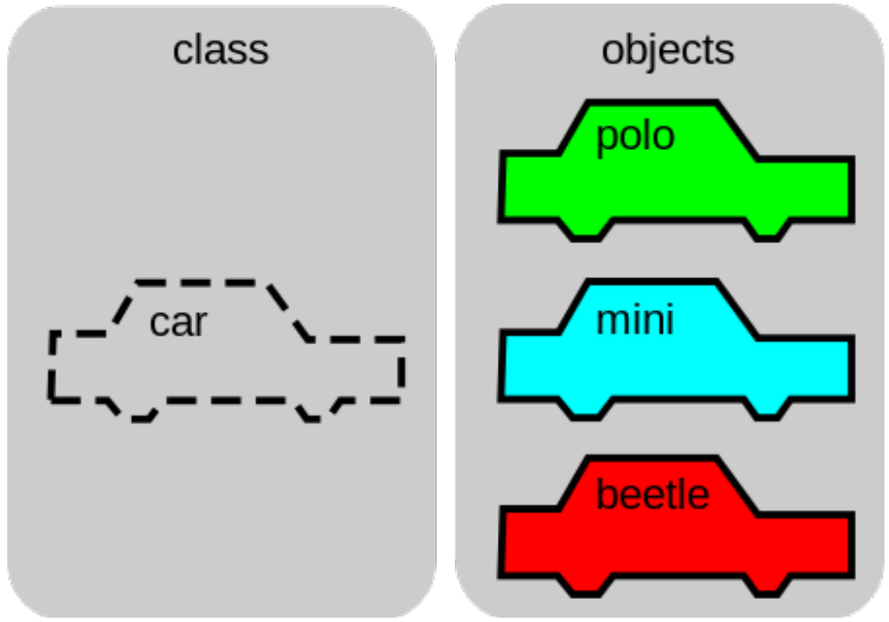
```
> 1 = x
```

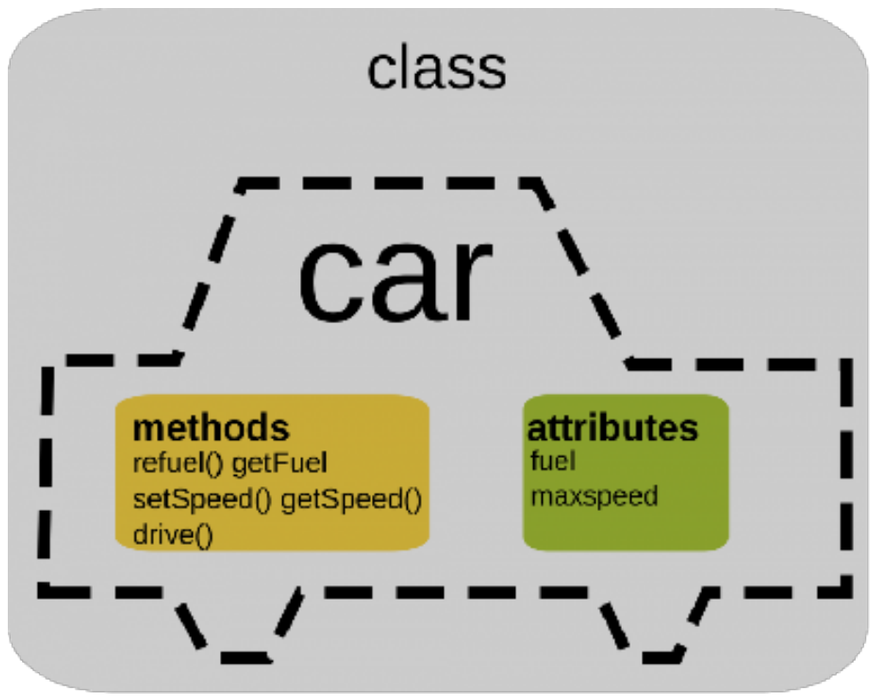
Declaring variables

```
> x <- 1

> y <- 2

> x + y
[1] 3

> z <- x + y

> class(z)
[1] "numeric"
```

your first objects

your first method





Vectors and operations on them

```
> p <- 1:5
> p
[1] 1 2 3 4 5
> p * 2
[1] 2 4 6 8 10
> q <- 6:10
> p * q
[1] 6 14 24 36 50
> r <- 1:3
> p * r
[1]  1  4  9  4 10
Warning message:
In p * r : longer object length is not a multiple of shorter
object length
```

Vectors: again, many ways to skin a cat…

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(50,100,by=3)
 [1] 50 53 56 59 62 65 68 71 74 77 80 83 86 89 92 95 98
> seq(18,2)
 [1] 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2
> rep(4,6)
 [1] 4 4 4 4 4 4
> c(4,2,5,7,2,6,3)
 [1] 4 2 5 7 2 6 3
> c(1:4,rep(3,7),seq(6))
 [1] 1 2 3 4 3 3 3 3 3 3 3 1 2 3 4 5 6
```

Exercise

Write a line of code that produces the following vector:

```
[1]  1  2  3  4  5  9  9  9 11 13 15
```

(Note! There are multiple possible answers!)

Exercise

Write a line of code that produces the following vector:

```
> c(1:5,rep(9,3),seq(11,15,by=2))
[1]  1  2  3  4  5  9  9  9 11 13 15
```

(Note! There are multiple possible answers!)

Exercise - extra credit!

Make a vector with a range of numbers from 1:10

Test the class of this vector

Now divide the vector by two, and test the class again

What happened?

Exercise - extra credit!

Make a vector with a range of numbers from 1:10

```
> myvector <- 1:10
```

Test the class of this vector

```
> class(myvector)
[1] "integer"
```

Now divide the vector by two, and test the class again

```
> myvector <- myvector/2
> class(myvector)
[1] “numeric”
```

What happened?

Characters and combinations

```
> myname <- “Barbara"
> myname <- Barbara
Error: object 'Barbara' not found
> mycv1 <- c("This","is","a","character","vector")
> mycv2 <- c("This", 1, "is", 4, "both")
> c(mycv1,mycv2)
[1] "This"      "is"        "a"         "character" "vector"
"This"      "1"         "is"        "4"         "both"
> mylist <- list(mycv1,mycv2)
> mylist
[[1]]
[1] "This"      "is"        "a"         "character" "vector"

[[2]]
[1] "This" "1"    “is"   "4"    "both"
```

Combine to make a data frame

```
> names <- c("Ann","Bob","Chloe","Dan")
> ages <- c(35,22,21,47)
> mydb <- data.frame(names,ages)
> mydb
  names ages
1   Ann   35
2   Bob   22
3 Chloe   21
4   Dan   47
> mydb$sex <- c("F","M","F","M")
> mydb
  names ages sex
1   Ann   35   F
2   Bob   22   M
3 Chloe   21   F
4   Dan   47   M
```

Exercise

Add a column to the data frame containing the participants' eye color.

Check your data frame to see if the new information has been added.

Exercise

Add a column to the data frame containing the participants' eye color.

Check your data frame to see if the new information has been added.

```
> mydb$eye_color <- c("brown","hazel","brown","grey")
> mydb
  names ages sex eye_color
1   Ann   35   F     brown
2   Bob   22   M     hazel
3 Chloe   21   F     brown
4   Dan   47   M      grey
```

Another data class: logicals!

```
> T
[1] TRUE
> FALSE
[1] FALSE
> tvar <- TRUE
> tvar
[1] TRUE
> fvar <- F
> fvar
[1] FALSE
> 1==1
[1] TRUE
> 3>=5
[1] FALSE
> tvar!=F
[1] TRUE
```

Question

Note that the equal signs (=) in a test are double.

What happens if we do `tvar = F` instead of `tvar == F` ?

Back to the data frame…

```
> mydb$payment_status <- c(T,F,F,F)
> mydb
  names ages sex eye_color payment_status
1   Ann   35   F     brown           TRUE
2   Bob   22   M     hazel          FALSE
3 Chloe   21   F     brown          FALSE
4   Dan   47   M      grey          FALSE
```

```
> summary(mydb)
  names       ages           sex              eye_color          payment_status
Ann  :1   Min.   :21.00   Length:4           Length:4           Mode :logical
Bob  :1   1st Qu.:21.75   Class :character   Class :character   FALSE:3
Chloe:1   Median :28.50   Mode  :character   Mode  :character   TRUE :1
Dan  :1   Mean   :31.25
          3rd Qu.:38.00
          Max.   :47.00
```

Factors and characters

```
> class(mydb$names)
[1] "factor"
> attributes(mydb$names)
$levels
[1] "Ann"   "Bob"   "Chloe" "Dan"

$class
[1] "factor"
```

Question: are these classes appropriate?

```
> summary(mydb)
```

```
  names       ages          sex            eye_color          payment_status
Ann  :1   Min.   :21.00   Length:4           Length:4           Mode :logical
Bob  :1   1st Qu.:21.75   Class :character   Class :character   FALSE:3
Chloe:1   Median :28.50   Mode  :character   Mode  :character   TRUE :1
Dan  :1   Mean   :31.25
          3rd Qu.:38.00
          Max.   :47.00
```

What would you change?

Exercise: change the data types in the data frame

```
> mydb$names <- as.character(mydb$names)
> mydb$sex <- as.[your code]
> mydb$eye_color <- as.[your code]
```

Hint: you have not seen this function before. But you can probably deduce it!

Note that you are overwriting the existing columns. What would happen if you would simply do:

```
> as.character(mydb$names)
```

Extra credit exercise!

Change **`mydb$ages`** into a factor, then back to numeric.

```
> mydb$ages <- as.[your code]
> mydb$ages <- as.[your code]
```

What happens here?

(You do not need to fix it yet. We will get to that!)

But what if there *is no data?* Introducing: NA

```
> mydb$pet <- c(“Dog","None","Cat",NA)
> mydb$pet <- as.factor(mydb$pet)
> mydb$pet
[1] Dog  None Cat  <NA>
Levels: Cat Dog None
```

<NA> = **N**ot **A**vailable:

We **know** that Bob has **no pets**.
We **do not know if** Dan has pets.

NA NA NA NA NA NA…

```
> a <- NA
> b <- NA
```

Exercise: we are going to test if a equals b.

What do you predict the answer will be?

```
> a == b
[1] NA
```

Exercise: play with NA! For example…

```
> a + 3
[1] NA
> a > 1
[1] NA
...
```

NA NA NA NA NA NA NULL!

```
> is.na(a)
[1] TRUE

> c <- NULL
> is.null(c)
[1] TRUE
```

`NA` Information is **Not Available**

`NULL` Information **does not exist**

`“None”` Data entry specifying **“None”**

Exercise: try to predict the results!

```
> is.na(NA)
[1] TRUE
> is.null(NULL)
[1] TRUE
```

Predict the results. Does the (real) answer make sense to you?

```
> is.null(NA)

> is.na(NULL)
```

Exercise: try to predict the results!

```
> is.na(NA)
[1] TRUE
> is.null(NULL)
[1] TRUE
```

Predict the results. Does the (real) answer make sense to you?

```
> is.null(NA)
[1] FALSE
> is.na(NULL)
[1] logical(0)
Warning message:
In is.na(NULL) : is.na() applied to non-(list or vector) of
type 'NULL'
```

Let's breathe and recap!

What data types have you encountered so far?

`logical`
`numeric`
`integer`
`character`
`factor`

And what data collections have you encountered?

`vector` (one dimension)
`data frame` (two dimensions)
`list` (++ dimensions)

Functions

What functions have you encountered so far?

```
> c()
> rep()
> data.frame()
> summary()
> class()
> as.character()
...
```

And do you still know what they mean? And how to use them? **No?**

```
> ?summary()
```

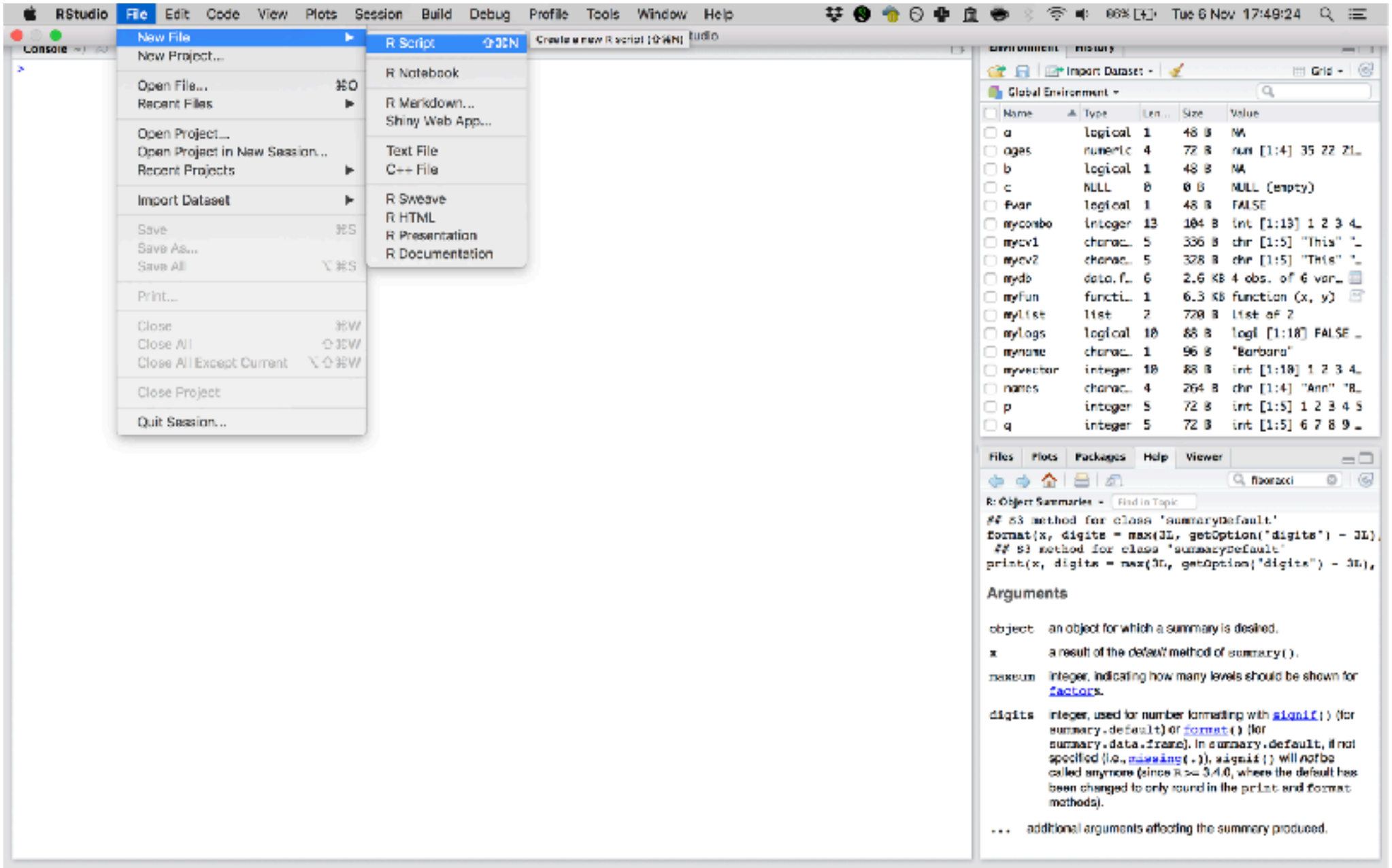
(or use the Help window to the right of your console)

Did you lose track of the data frame?

```
> names <- c("Ann","Bob","Chloe","Dan")
> ages <- c(35,22,21,47)
> mydb <- data.frame(names,ages)

> mydb$sex <- c("F","M","F","M")
> mydb$eye_color <- c(“brown”,"hazel","brown","grey")
> mydb$payment_status <- c(T,F,F,F)

> mydb$names <- as.character(mydb$names)
> mydb$sex <- as.factor(mydb$sex)
> mydb$eye_color <- as.factor(mydb$eye_color)
```

Ready to script?

RStudio > File > New File > R Script



Introducing: comments

```
# Let's generate a vector.
myvector <- c(1:5,seq(10,6))
```

 # indicates comment
computer stops reading
(but you should not!)

Indexing and selecting

```
# Let's generate a vector. (Feel free to generate your own!)
> myvector <- c(1:5,seq(10,6))

# How long is your vector?
> length(myvector)
[1] 10
```

```
# Let's say I am only interested in the 7th element
# We can select it using square brackets:
> myvector[7]
[1] 9
```

Indexing and selecting

```
# Note that R starts counting at 1.
# Not all programming languages do this.
# This means the final element in the vector is:
> myvector[length(myvector)]
[1] 6
```

EXERCISE: try selecting other elements.

What happens if you select an element that is not indexed (i.e. it is out of range)?

Indexing and selecting

```
# Another way to get the final element is
> tail(myvector,n=1)
[1] 6

# What does the following code do?
> head(myvector,n=3)
[1] 1 2 3
```

Which bracket does what?

[ ] **Indexing** vectors, lists, dataframes…

( ) Passing **arguments** to functions

{ } **Defining content** of loops, functions, etc.

```
> myvector[length(myvector)]
```

Indexing and selecting

```
# Let's select multiple elements. Using a vector, you can
select in your vector!
> myvector[c(2,4)]
[1] 2 4
> myvector[2:4]
[1] 2 3 4
```

```
# We can also use double brackets, which functions much the
same:
> myvector[[5]]
[1] 5
```

Double brackets and lists

```
# Why would you use double brackets?
# Remember our list
> mylist
[[1]]
[1] "This"      "is"        "a"         "character" "vector"

[[2]]
[1] "This" "1"    "is"   "4"    "both"

# Note that it consists of two different elements.
# Let's select the first one
> mylist[1]
[[1]]
[1] "This"      "is"        "a"         "character" "vector"

# Note that this element is returned to us _as part of_ the list.
# If we want to select the element by itself we use two brackets
> mylist[[1]]
[1] "This"      "is"        "a"         "character" "vector"
```

Exercise: why is that important?

How would you select the second vector element of the first list-element?

A. `> mylist[1][2]`

B. `> mylist[[1]][2]`

Exercise: why is that important?

How would you select the second vector element of the first list-element?

A.
```
> mylist[1][2]
[[1]]
NULL
```

B.
```
> mylist[[1]][2]
[1] "is"
```

Indexing a data frame

```
> mydb
  names ages sex eye_color payment_status  pet
1   Ann   35   F     brown           TRUE  Dog
2   Bob   22   M     hazel          FALSE None
3 Chloe   21   F     brown          FALSE  Cat
4   Dan   47   M      grey          FALSE <NA>
```

```
# If we want to select Ann's pet:
# Our selection has two elements:
# - first: rows (which row is Ann? -> 1)
# - then: columns (which column is pets? -> 6)
> mydb[1,6]
[1] Dog
Levels: Cat Dog None
```

Exercises

1. Select **Ann, Bob, and Chloe's** payment status with a single line of code.

Make sure to include their names!

Hint: make good use of vectors as part of your index!

2. Return all the pets in the database.

Exercises

1. Select **Ann, Bob, and Chloe's** payment status with a single line of code.

Make sure to include their names!

Hint: make good use of vectors as part of your index!

```
> mydb[c(1:3),c(1,5)]
  names payment_status
1   Ann            TRUE
2   Bob           FALSE
3 Chloe           FALSE
```

2. Return all the pets in the database.

```
> mydb[,6]
[1] Dog  None Cat  <NA>
Levels: Cat Dog None
```

Indexing with logicals

```
# Let's make a vector with numbers 1:10
> myvector <- seq(10)

# And a vector with logicals, of the same length
> mylogs <- logical(10)

# What does this vector look like?
> mylogs
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

# All false! Let's turn a few into TRUE
```

Exercise

Replace 3 of the 10 **`FALSE`** values in **`mylogs`** with **`TRUE`**.

Exercise

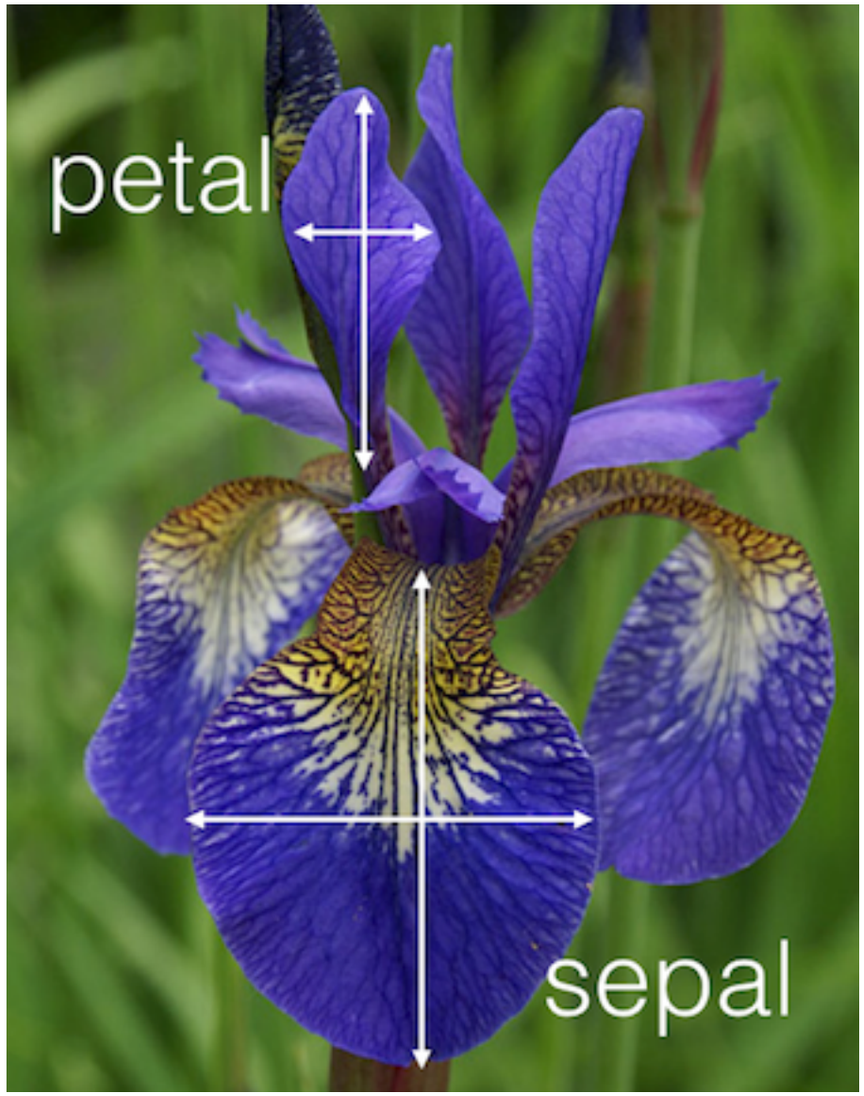
Replace 3 of the 10 **`FALSE`** values in **`mylogs`** with **`TRUE`.**

```
> mylogs[c(2,4,7)] <- TRUE

> mylogs
[1] FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE

# now, select the logicals from the numeric vector
> myvector[mylogs]
[1] 2 4 7
```

Question: what happened here?

Introducing a sample dataset

- Dataset: ‘iris’
- Standard dataset in R, measurements on 3 species of iris flowers



Introducing a sample dataset

- Dataset: ‘iris’

- Standard dataset in R, measurements on 3 species of iris flowers

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa     :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Selecting data based on a factor

```
# We want to select ‘setosa’ measurements only.
# To start, let’s look at the species column.
> iris$Species
  [1] setosa     setosa     setosa     setosa     setosa
(...)
[141] virginica  virginica  virginica  virginica  virginica
virginica  virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica

# Which species are setosa?
> iris$Species == “setosa”
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
(...)
[145] FALSE FALSE FALSE FALSE FALSE FALSE
```

Selecting data based on a factor

```
# Let’s select from the data frame only those rows
# which are ‘setosa’ species:
> iris.setosa <- iris[iris$Species==“setosa”,]

# How many are there?
# Let’s check the dimensions of the original data set
> dim(iris)
[1] 150   5

# And the new data set
> dim(iris.setosa)
[1] 50   5
```

Selecting data: exercise

Exercise: make a new data set that contains **only petals that are longer** than the **average versicolor** petal length.

Hint: work stepwise!

Selecting data: exercise

Exercise: make a new data set that contains **only petals that are longer** than the **average versicolor** petal length.

Hint: work stepwise!

```
# What is the average versicolor petal length?
# First, select only versicolor petals


# Then calculate the mean


# Select from the main dataset
```

Selecting data: exercise

Exercise: make a new data set that contains **only petals that are longer** than the **average versicolor** petal length.

Hint: work stepwise!

```
# What is the average versicolor petal length?
# First, select only versicolor petals
> versicolor.petal.length <-
                    iris[iris$Species==“versicolor”,”Petal.Length”]
# Then calculate the mean
> mean.vpl <- mean(versicolor.petal.length)

# Select from the main dataset
> iris.largepetals <- iris[iris$Petal.Length>mean.vpl,]
```

Selecting data: selecting columns

Exercise: select from the iris data only the columns with the word ‘Petal’.

Hint: use the function grep(), and the function names().

```
# Which columns have the word “Petal”?
```

```
# Select these columns from the dataset
```

```
# Can you do this in a single line of code?
```

Selecting data: selecting columns

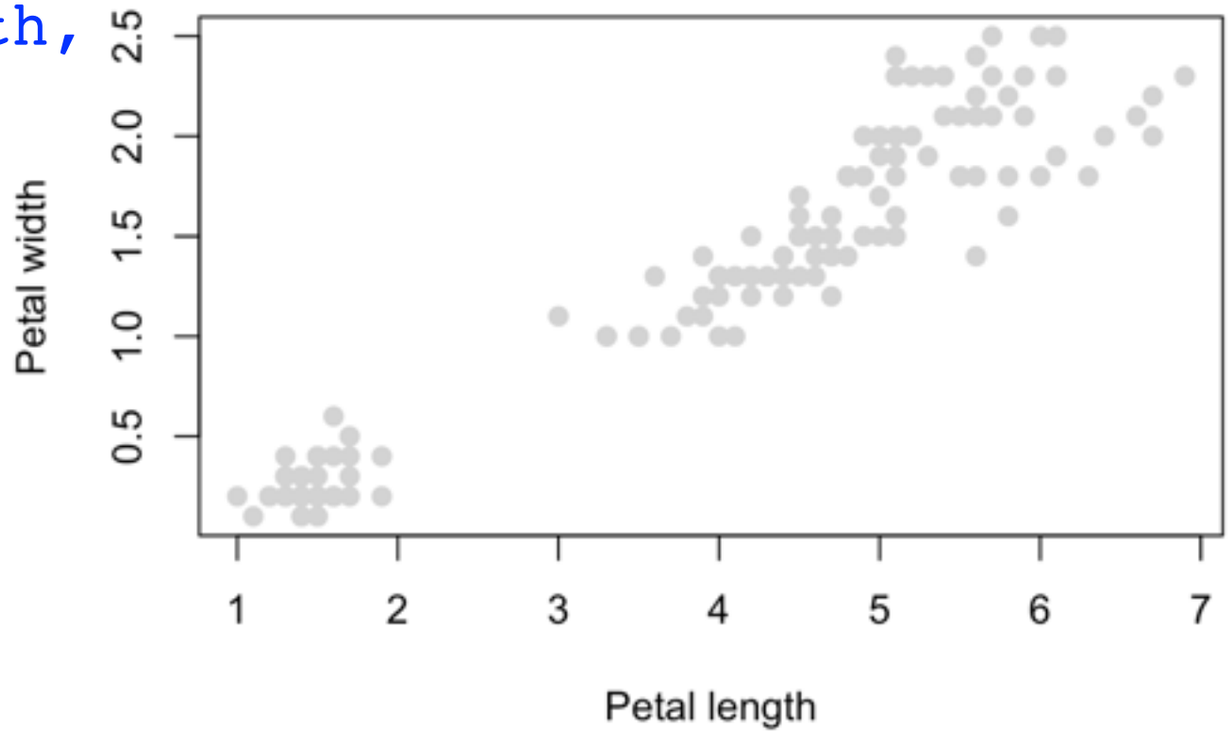
Exercise: select from the iris data only the columns with the word ‘Petal’.

Hint: use the function grep(), and the function names().

```
# Which columns have the word “Petal”?
> petalcols <- grep(“Petal",names(iris))

# Select these columns from the dataset
> iris.petalcols <- iris[,petalcols]

# Can you do this in a single line of code?
> iris.petalcols <- iris[, grep(“Petal",names(iris))]
```

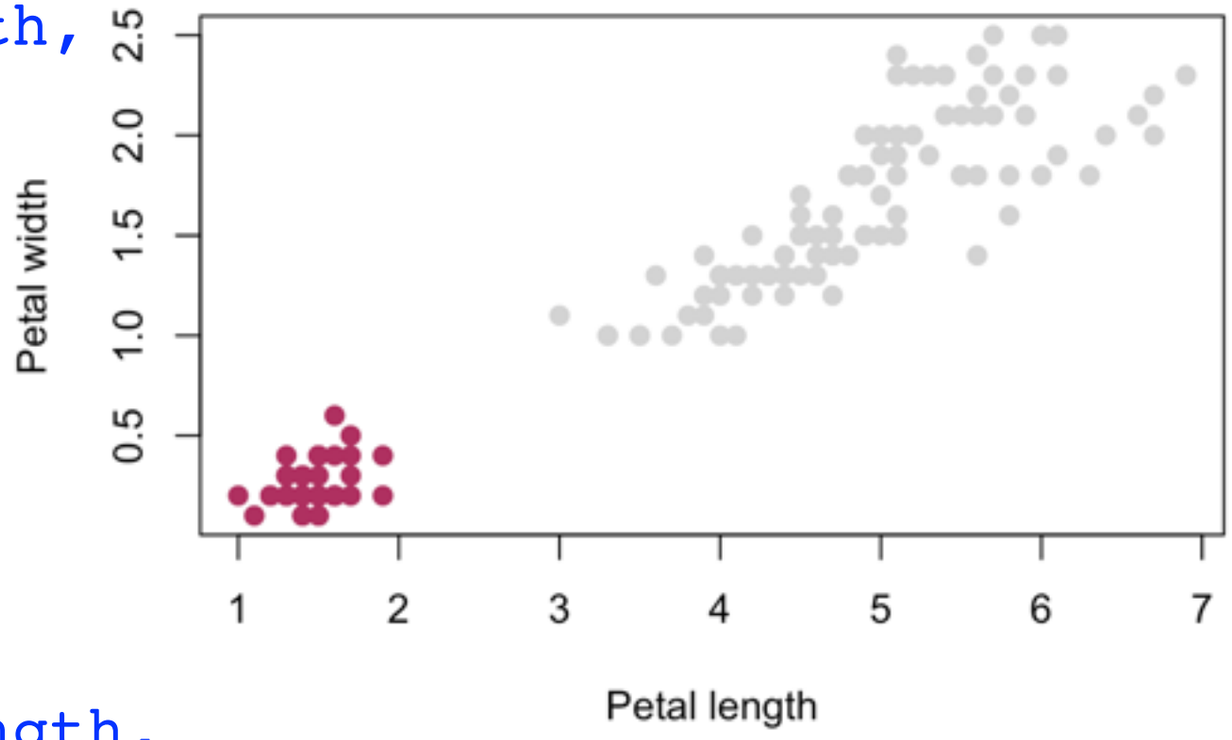
Visualizing data: don’t try this at home!

```
# Let’s plot petal width and petal length
> plot(Petal.Width~Petal.Length,
  data=iris,
  col=“lightgrey",
  pch=19,
  xlab="Petal length”,
  ylab="Petal width”)
```



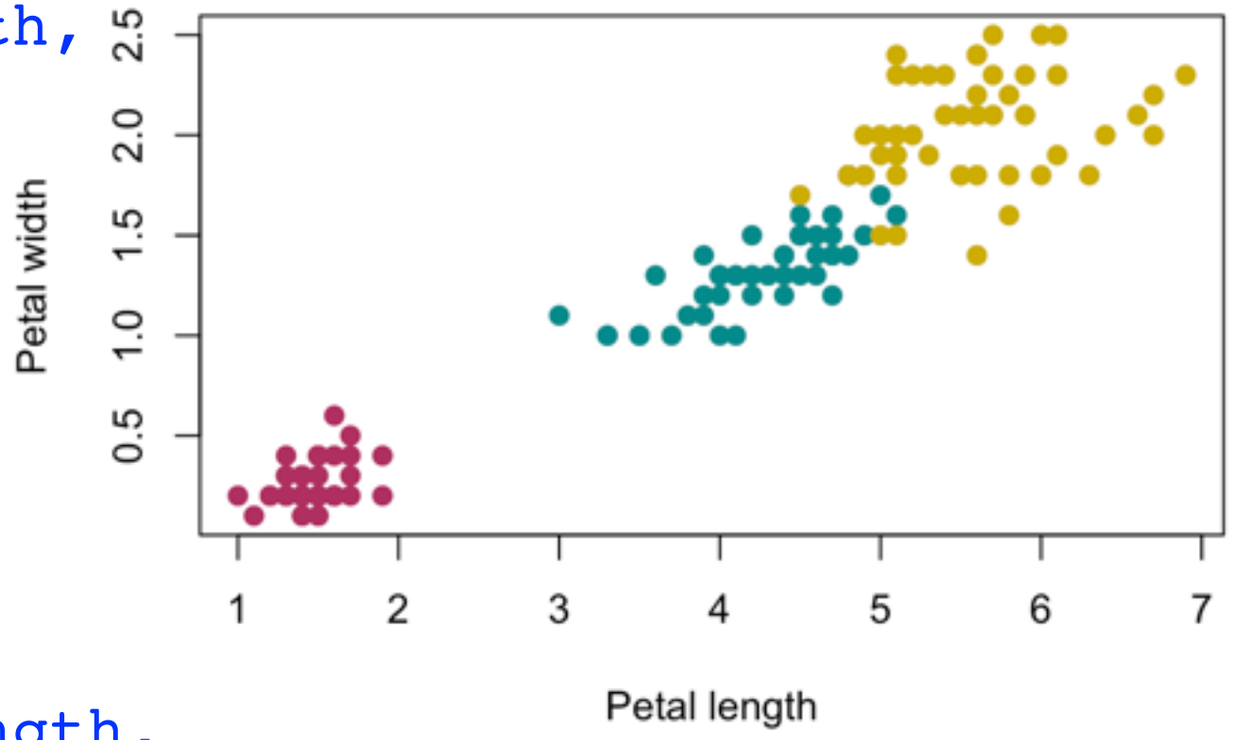
Visualizing data: don't try this at home!

```
# Let's plot petal width and petal length
> plot(Petal.Width~Petal.Length,
  data=iris,
  col="lightgrey",
  pch=19,
  xlab="Petal length",
  ylab="Petal width")

# Label individual species
> points(Petal.Width~Petal.Length,
  data=iris[iris$Species=="setosa",],
  col="maroon",
  pch=19)
```



Visualizing data: don't try this at home!

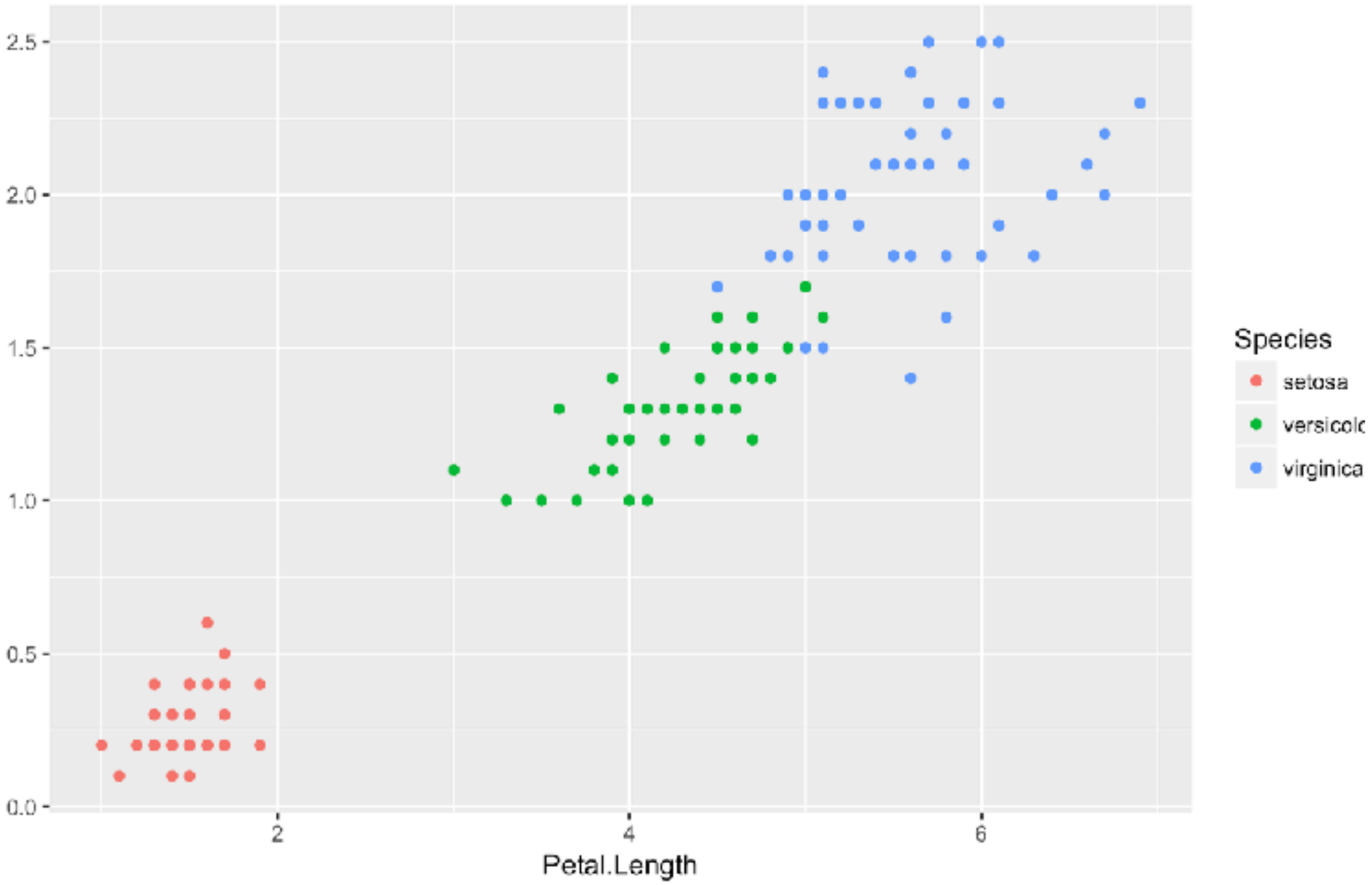
```
# Let's plot petal width and petal length
> plot(Petal.Width~Petal.Length,
  data=iris,
  col="lightgrey",
  pch=19,
  xlab="Petal length",
  ylab="Petal width")
```



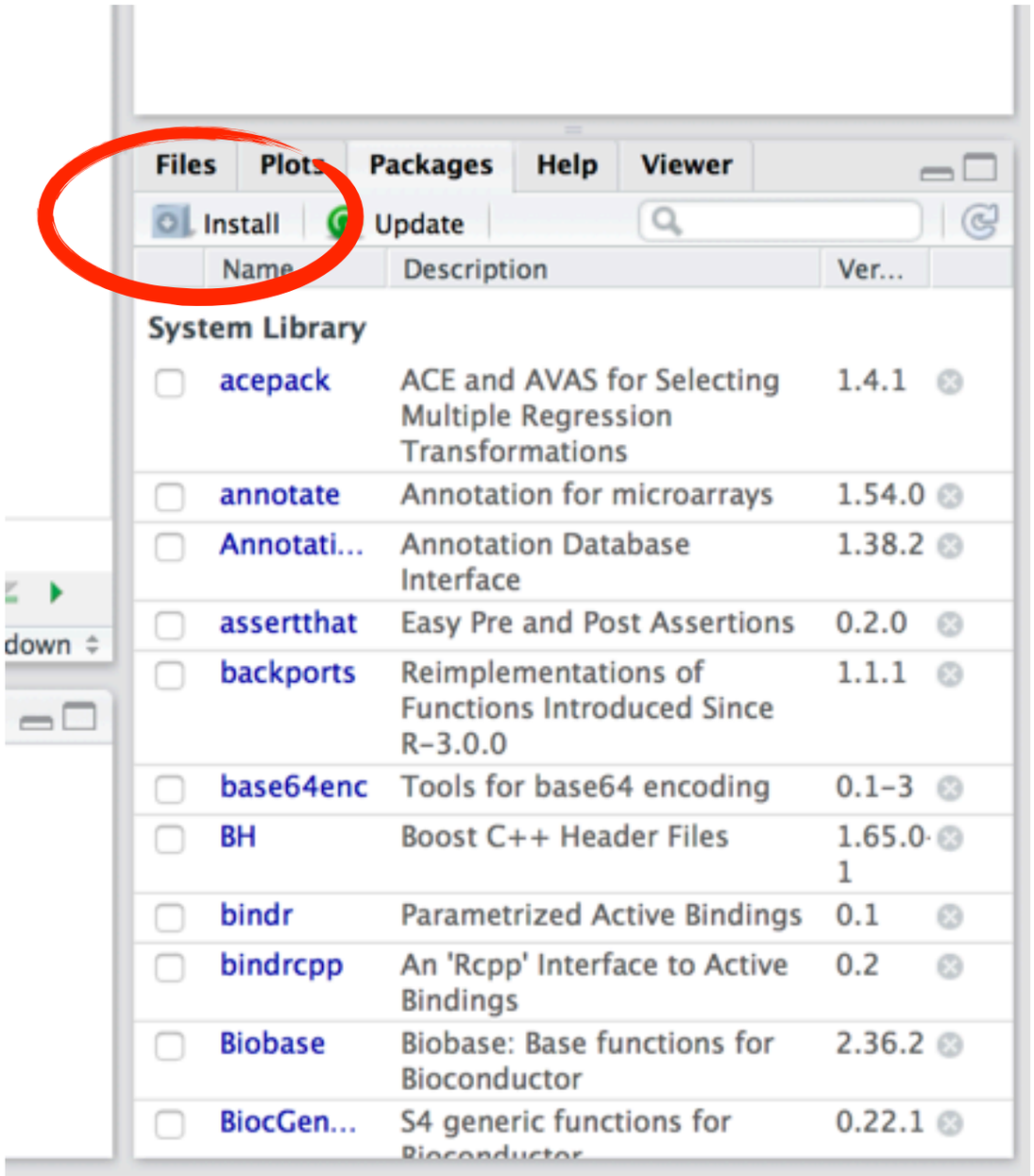

```
# Label individual species
> points(Petal.Width~Petal.Length,
  data=iris[iris$Species=="setosa",],col="maroon",pch=19)
> points(Petal.Width~Petal.Length,
  data=iris[iris$Species=="versicolor",],col="darkcyan",pch=19)
> points(Petal.Width~Petal.Length,
  data=iris[iris$Species=="virginica",],col="gold3",pch=19)
```

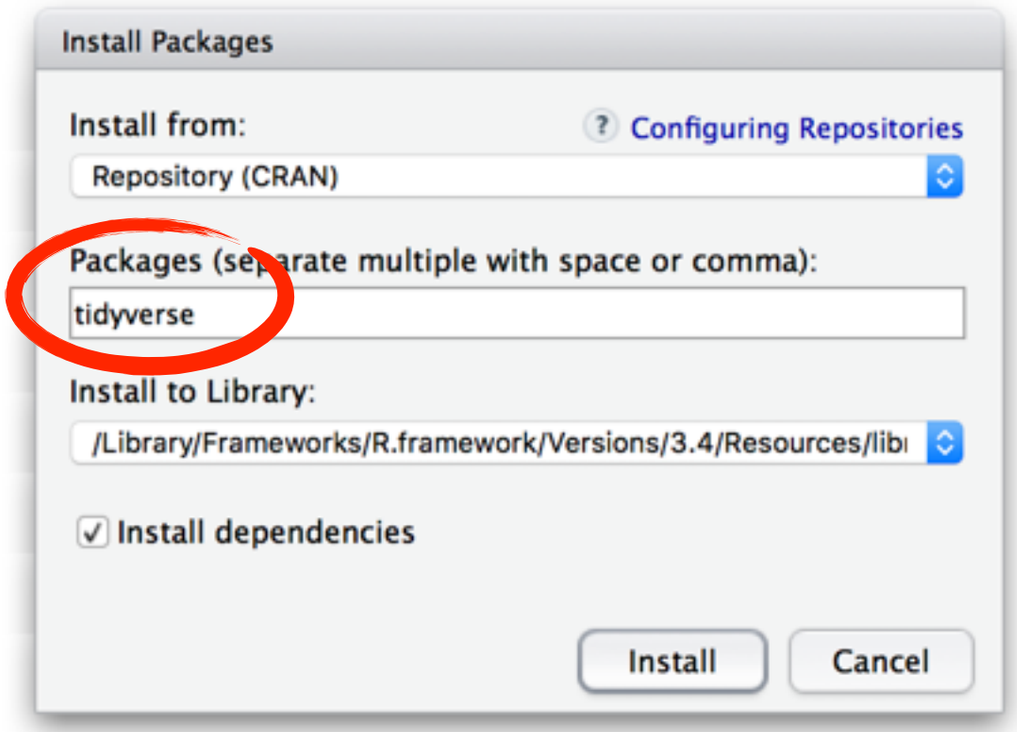
A sneak peek to a better world!

```
> ggplot(iris, aes(x = Petal.Length, y = Petal.Width, colour=Species)) +
    geom_point()
```



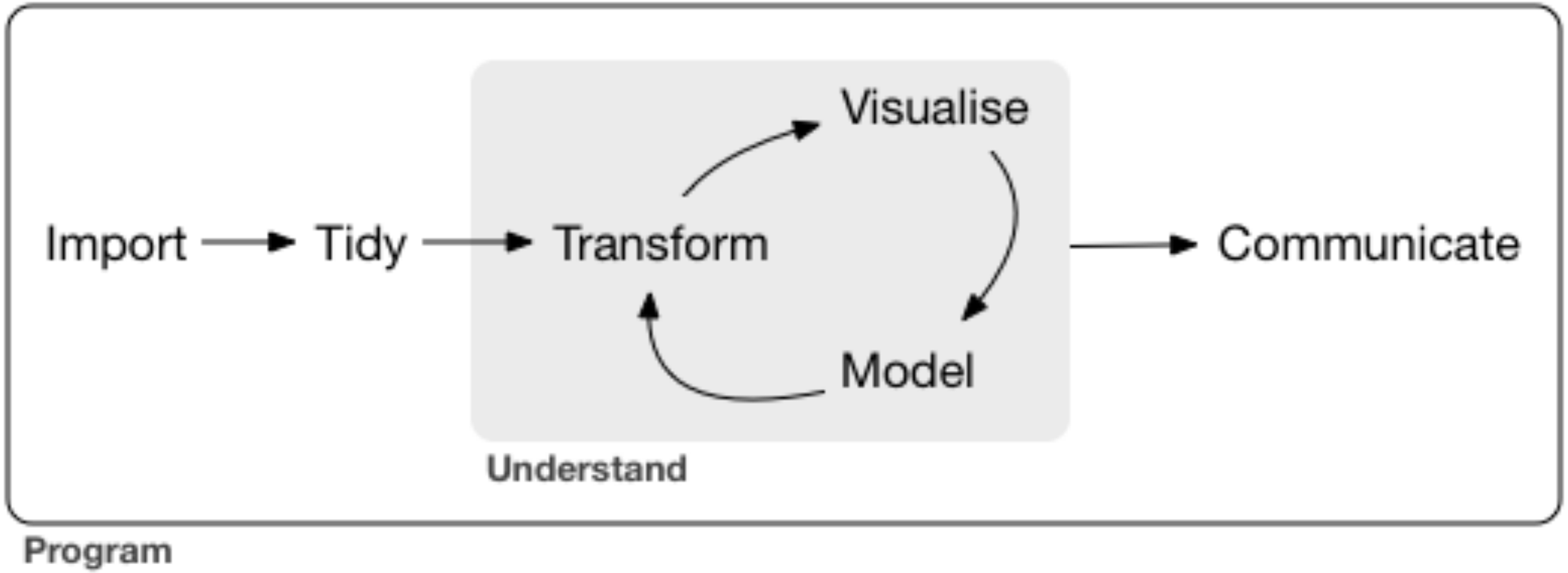
Afternoon prep: please install Tidyverse tools





Data science workflow: scripting is crucial

- Scripting combines commands to a comprehensive set of instructions.
- A script is code that can be **saved, reused, shared, published**!
- In short: a crucial step towards reproducible data analysis.



a single script for a single purpose!

Starting the script: write a header

```
## Date: 7 November 2018
## Author: Barbara Vreede
## This script was written as part of the R course
## "Introduction to R & Data", at Utrecht University
```

Load packages and dependencies

```
## Date: 7 November 2018
## Author: Barbara Vreede
## This script was written as part of the R course
## “Introduction to R & Data”, at Utrecht University

# Load required packages
library(dplyr)
library(tidyr)
library(ggplot2)
```

Custom functions

```
## Date: 7 November 2018
## Author: Barbara Vreede
## This script was written as part of the R course
## "Introduction to R & Data", at Utrecht University

# Load required packages
library(dplyr)
library(tidyr)
library(ggplot2)

# Functions
myFun <- function(var){
  var <- var*2*pi
  return(var)
}
```

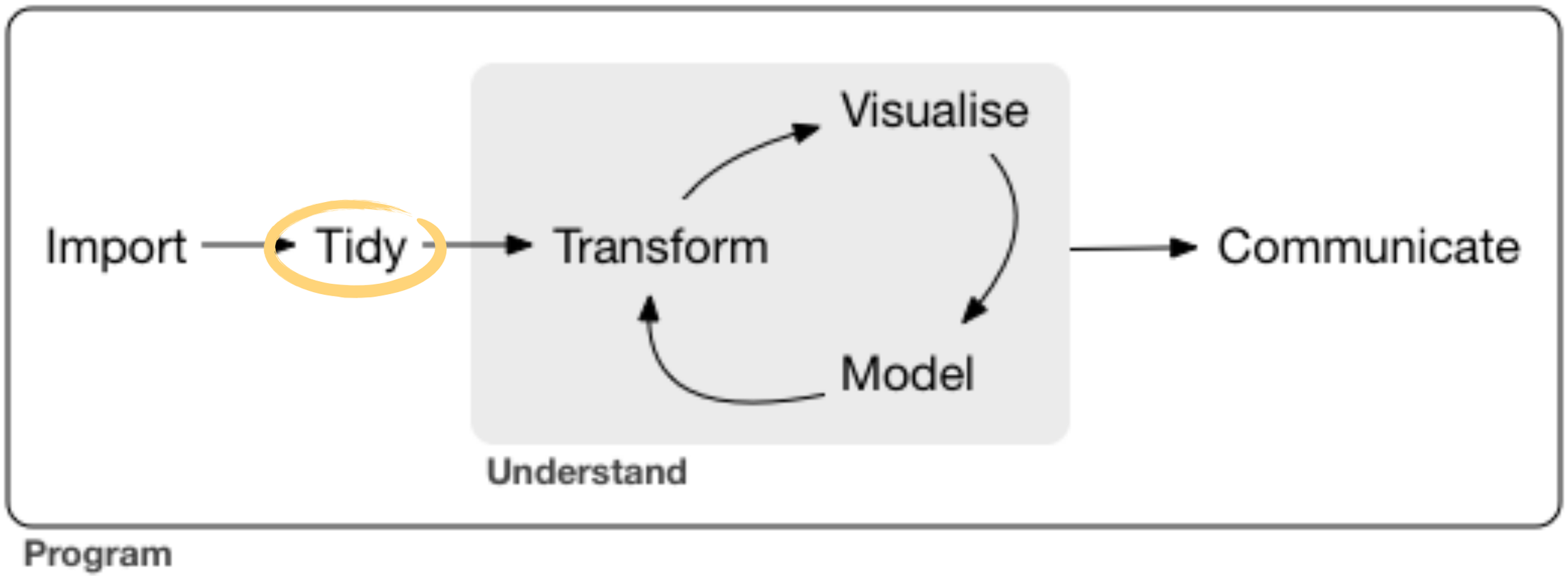
Starting your script: header, packages, functions

```
## Date: 7 November 2018
## Author: Barbara Vreede
## This script was written as part of the R course
## "Introduction to R & Data", at Utrecht University
```

```
# Load required packages
library(dplyr)
library(tidyr)
library(ggplot2)
```

```
# Functions
myFun <- function(var){
  var <- var*2*pi
  return(var)
}
```

Data science workflow



Tidy data

- Each *variable* is a column and contains *values*
- Each *observation* is a row
- Each type of *observational unit* forms a table

Messy:

	Treatment A	Treatment B
John Smith	-	2
Jane Doe	16	11
Mary Johnson	3	1

Tidy:

Name	Treatment	Result
John Smith	a	-
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

(Examples from: ***http://www.jeannicholashould.com/tidy-data-in-python.html****)*

“All tidy data is alike; each messy dataset is messy in its own way.”

–Leo Tolstoy (or possibly not)

Rmarkdown

- Combine code with narrative
- Tutorials, books, journal papers
- … or your note book.
- Let's take a look!

Introduction to R and Data

Barbara Vreede

11/7/2018

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authori
on using R Markdown see http://rmarkdown.rstudio.com.

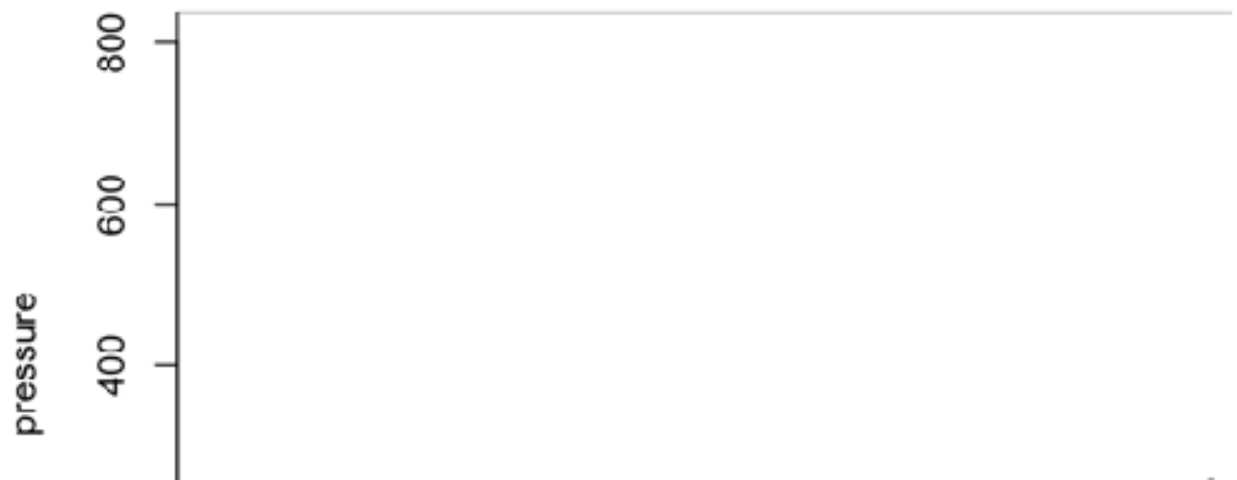
When you click the **Knit** button a document will be generated that includes both conte
within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

Including Plots

You can also embed plots, for example:



Markup in markdown: headers

```
56
57 # H1
58 ## H2
59 ### H3
60 #### H4
61 ##### etc.
62
```

H1

H2

H3

H4

etc.

Markup in markdown: emphasis

```
## Emphasis

You can add emphasis to text with *asterisks* or _underscores_.

Bold text is generated with **double asterisks** or __two underscores__.

And of course **they can be _combined_ like _this_**.

Show that you changed your mind with tildes to ~~scratch text~~ show improvement.
```

Emphasis

You can add emphasis to text with *asterisks* or *underscores*.

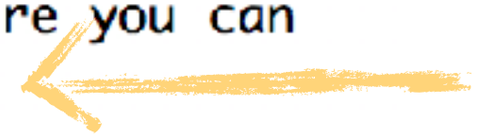
Bold text is generated with **double asterisks** or **two underscores**.

And of course **they can be *combined* like *this***.

Show that you changed your mind with tildes to ~~scratch text~~ show improvement.

Markup in markdown: lists

```
73 ## Make a list
74
75 Here you can
76
77 - sum
78 - up
79 - your
80 - ideas!
```



Make a list

Here you can

- sum
- up
- your
- ideas!

Markup in markdown: embedding results

```
79
80 ## Including code
81
82 You write your code in code chunks, of course.
83
84 But sometimes you need to refer to data in the text. For instance:
85
86 The mean petal length of iris flowers is `r mean(iris$Petal.Length)`.
87
```

Including code

You write your code in code chunks, of course.

But sometimes you need to refer to data in the text. For instance:

The mean petal length of iris flowers is 3.758.

Please download the course material:

---

tinyurl.com/introRData > Clone or download > Download ZIP

…and enjoy your lunch!

Introduction to R & data

Part II: Modern R with tidyverse

Tidyverse

Modern R for data science

"The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures."

– tidyverse.org (2018)

Tidyverse's author

- **Hadley Wickham**
- Productive PhD student in statistics
- Author of many R packages (50+)
- Chief scientist at Rstudio
- Tidyverse is sometimes named **hadleyverse**

Load tidyverse

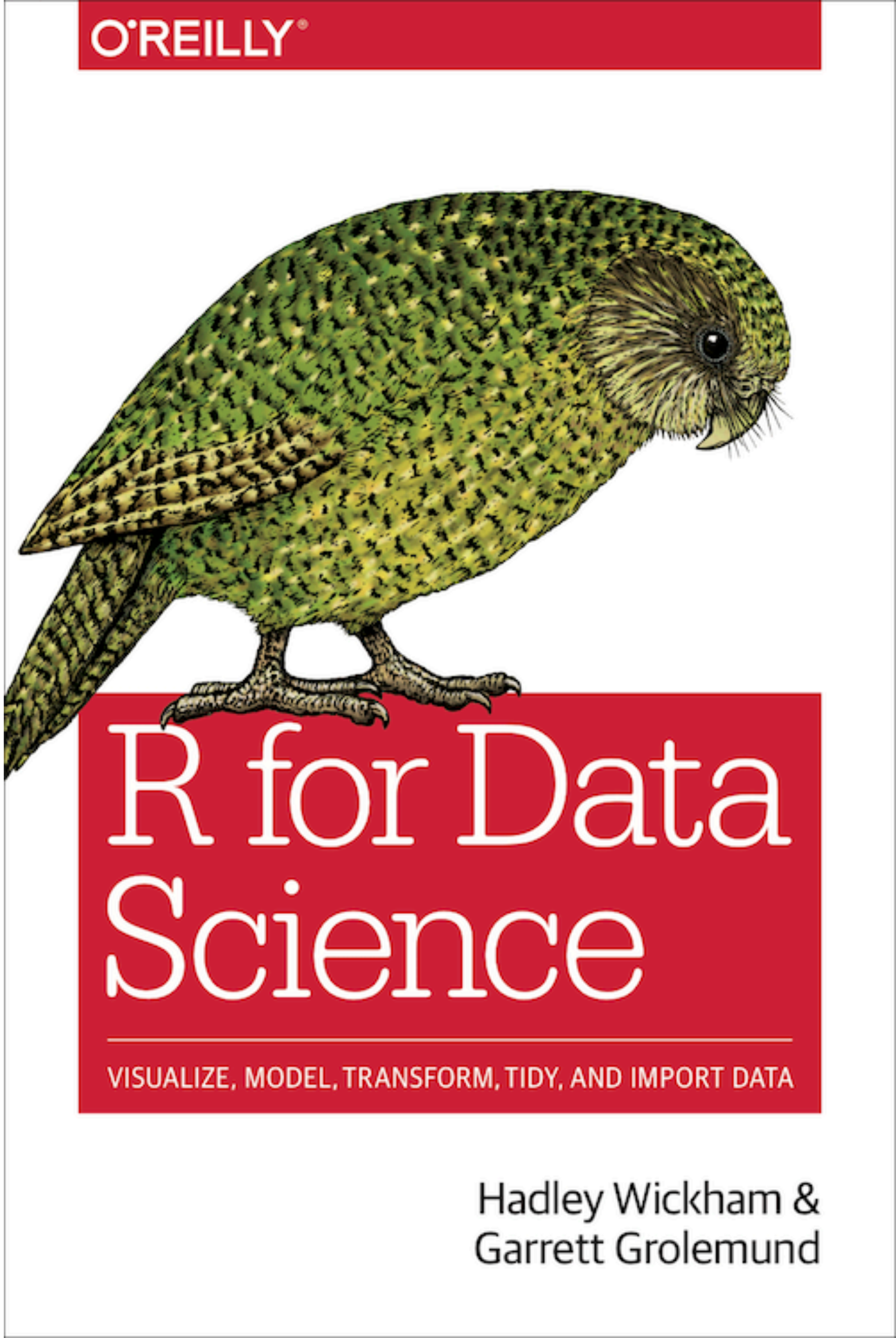
```
> library(tidyverse)
```

Load tidyverse

```
> library(tidyverse)
── Attaching packages ─────────────────────────────────────
tidyverse 1.2.1 ──
✔ ggplot2 2.2.1     ✔ purrr   0.2.4
✔ tibble  1.4.2     ✔ dplyr   0.7.4
✔ tidyr   0.8.0     ✔ stringr 1.3.1
✔ readr   1.1.1     ✔ forcats 0.3.0
── Conflicts ────────────────────────────────────────────────
tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

Learn tidyverse

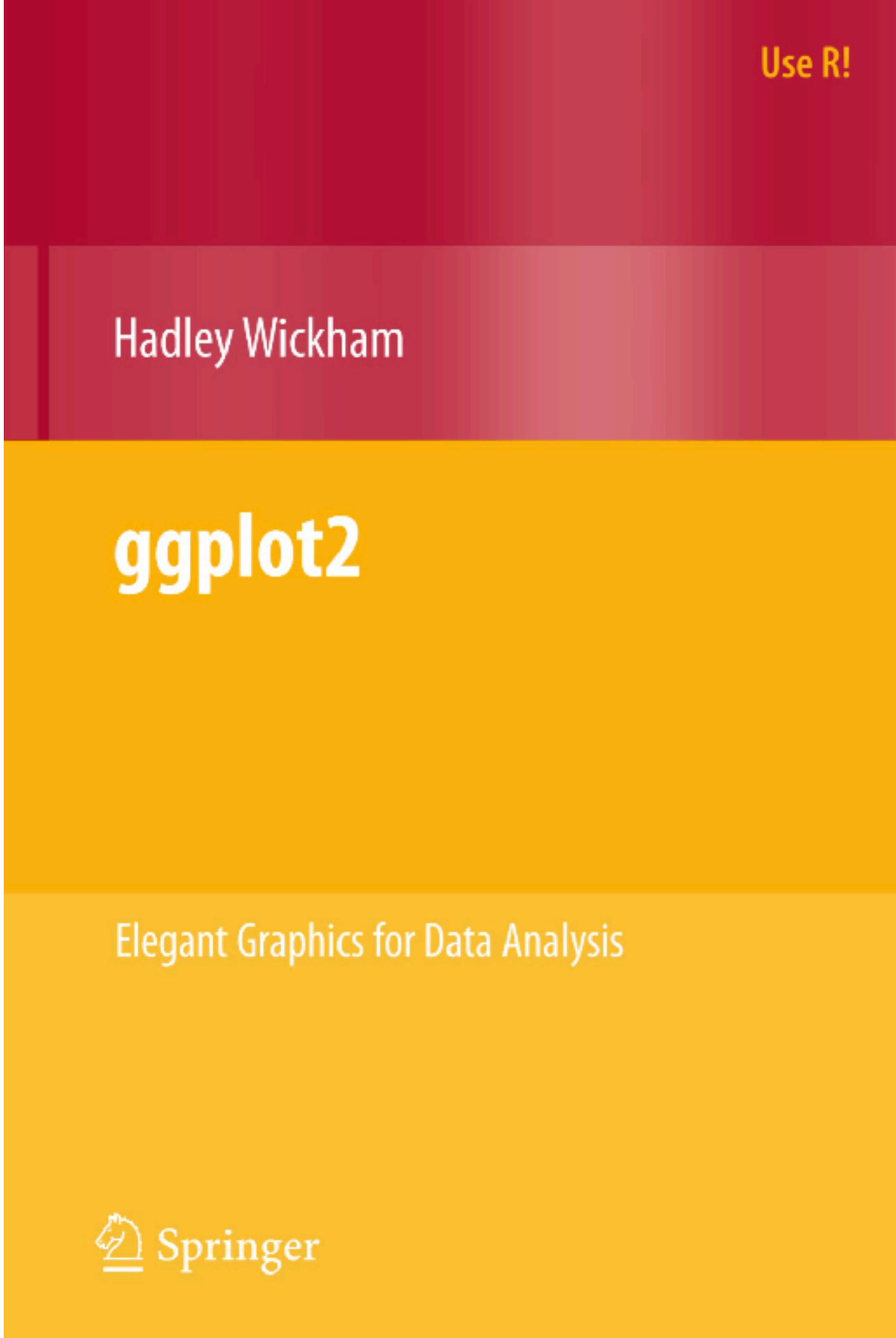
- R for Data Science (book)
  Freely available on:
  [r4ds.had.co.nz/](r4ds.had.co.nz/)



Learn tidyverse

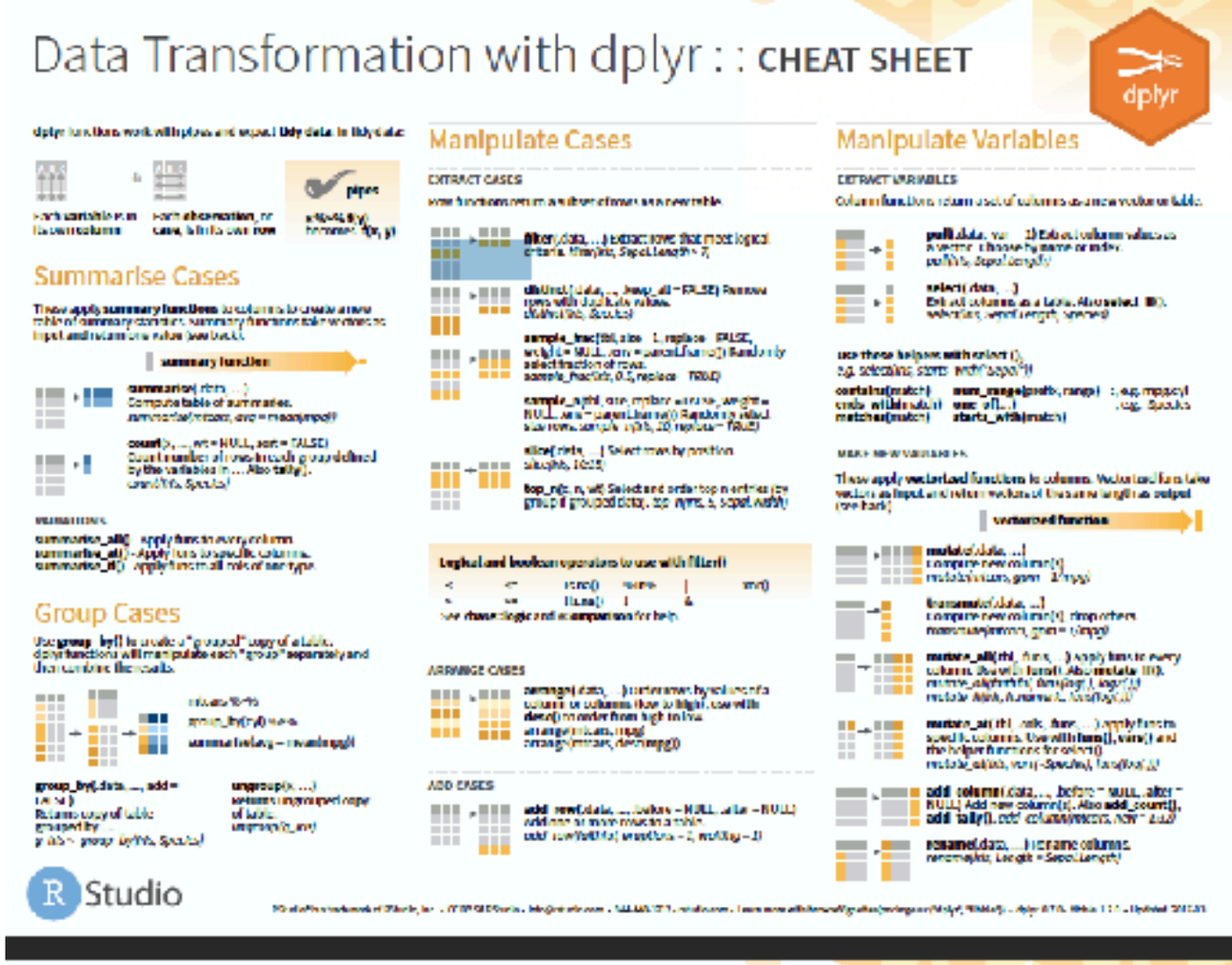
- R for Data Science (book)
  Freely available on:
  [r4ds.had.co.nz/](r4ds.had.co.nz/)

- ggplot2 (book)



Learn tidyverse

- R for Data Science (book)
  Freely available on:
  [r4ds.had.co.nz/](r4ds.had.co.nz/)
- ggplot2 (book)
- cheatsheets
  www.rstudio.com/resources/cheatsheets/



package: Tibble

Improved data frames

“Tibbles are data frames, but they tweak some older behaviours to make life a little easier. R is an old language, and some things that were useful 10 or 20 years ago now get in your way. It’s difficult to change base R without breaking existing code, so most innovation occurs in packages.”

– Hadley Wickham

Tibbles

```
> tibble(a = 1:26, b = letters)
# A tibble: 26 x 2
       a b
   <int> <chr>
 1     1 a
 2     2 b
 3     3 c
 4     4 d
 5     5 e
 6     6 f
 7     7 g
 8     8 h
 9     9 i
10    10 j
# ... with 16 more rows
```

Tibbles: from data.frame to tibble

```
> as_tibble(iris)
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.10        3.50         1.40       0.200 setosa
 2          4.90        3.00         1.40       0.200 setosa
 3          4.70        3.20         1.30       0.200 setosa
 4          4.60        3.10         1.50       0.200 setosa
 5          5.00        3.60         1.40       0.200 setosa
 6          5.40        3.90         1.70       0.400 setosa
 7          4.60        3.40         1.40       0.300 setosa
 8          5.00        3.40         1.50       0.200 setosa
 9          4.40        2.90         1.40       0.200 setosa
10          4.90        3.10         1.50       0.100 setosa
# ... with 140 more rows
```

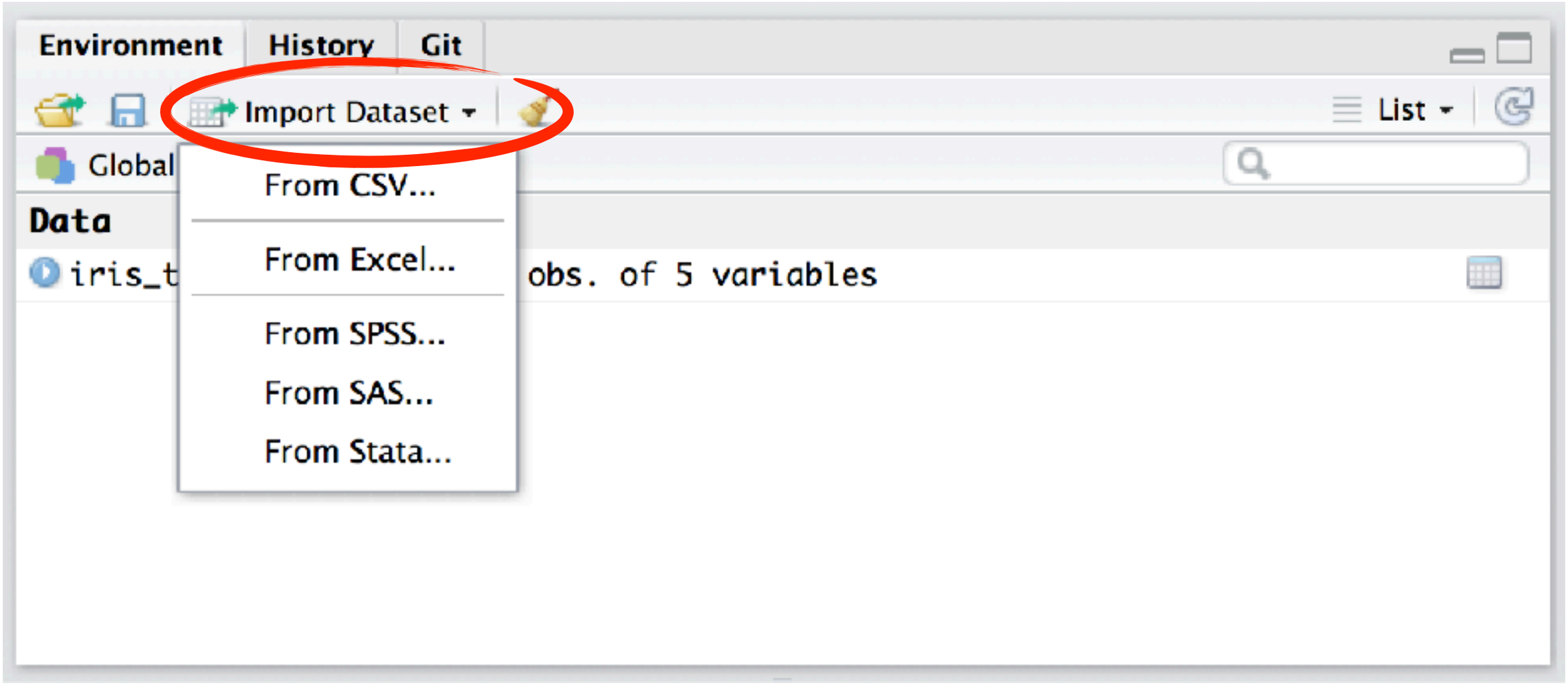
Tibbles: from tibble to data.frame

```
> iris_tibble <- as_tibble(iris)
> as.data.frame(iris_tibble)
    Sepal.Length Sepal.Width Petal.Length Petal.Width
Species
1            5.1         3.5          1.4         0.2
setosa
2            4.9         3.0          1.4         0.2
setosa
3            4.7         3.2          1.3         0.2
setosa
4            4.6         3.1          1.5         0.2
setosa
5            5.0         3.6          1.4         0.2
setosa
6            5.4         3.9          1.7         0.4
setosa
```

Load and save data

The basics of readr

Load data by using buttons



Load data by using buttons

Import Text Data

File/Url:

~/surfdrive/Projects/presentations/workshops/introduction-to-R-and-data-github/data/iris.csv

Browse...

Data Preview:

Sepal.Length (double)	Sepal.Width (double)	Petal.Length (double)	Petal.Width (double)	Species (character)
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa

Previewing first 50 entries.

Import Options:

Name: iris

Skip: 0

First Row as Names

Trim Spaces

Open Data Viewer

Delimiter: Comma

Quotes: Default

Locale: Configure...

Escape: None

Comment: Default

NA: Default

Code Preview:

```
library(readr)
iris <- read_csv("~/surfdrive/Projects/presentations/workshops/introduction-to-R-and-data-github/data/iris.csv")
View(iris)
```

Import

Cancel

Read flat files

```
> library(readr)
> data_iris <- read_delim("data/iris.csv", delim=',')
```

Read flat files

```
> library(readr)
> data_iris <- read_delim("data/iris.csv", delim=',')
Parsed with column specification:
cols(
  Sepal.Length = col_double(),
  Sepal.Width = col_double(),
  Petal.Length = col_double(),
  Petal.Width = col_double(),
  Species = col_character()
)
```

Read flat files

```
> library(readr)
> data_iris <- read_csv("data/iris.csv")
Parsed with column specification:
cols(
  Sepal.Length = col_double(),
  Sepal.Width = col_double(),
  Petal.Length = col_double(),
  Petal.Width = col_double(),
  Species = col_character()
)
```

Exercises

Please do Basic exercises 1.1 and 1.2

Time left? Opt for a reading exercise or optional exercise!

Exercise dataset

- Pizza Restaurants and the Pizza They Sell

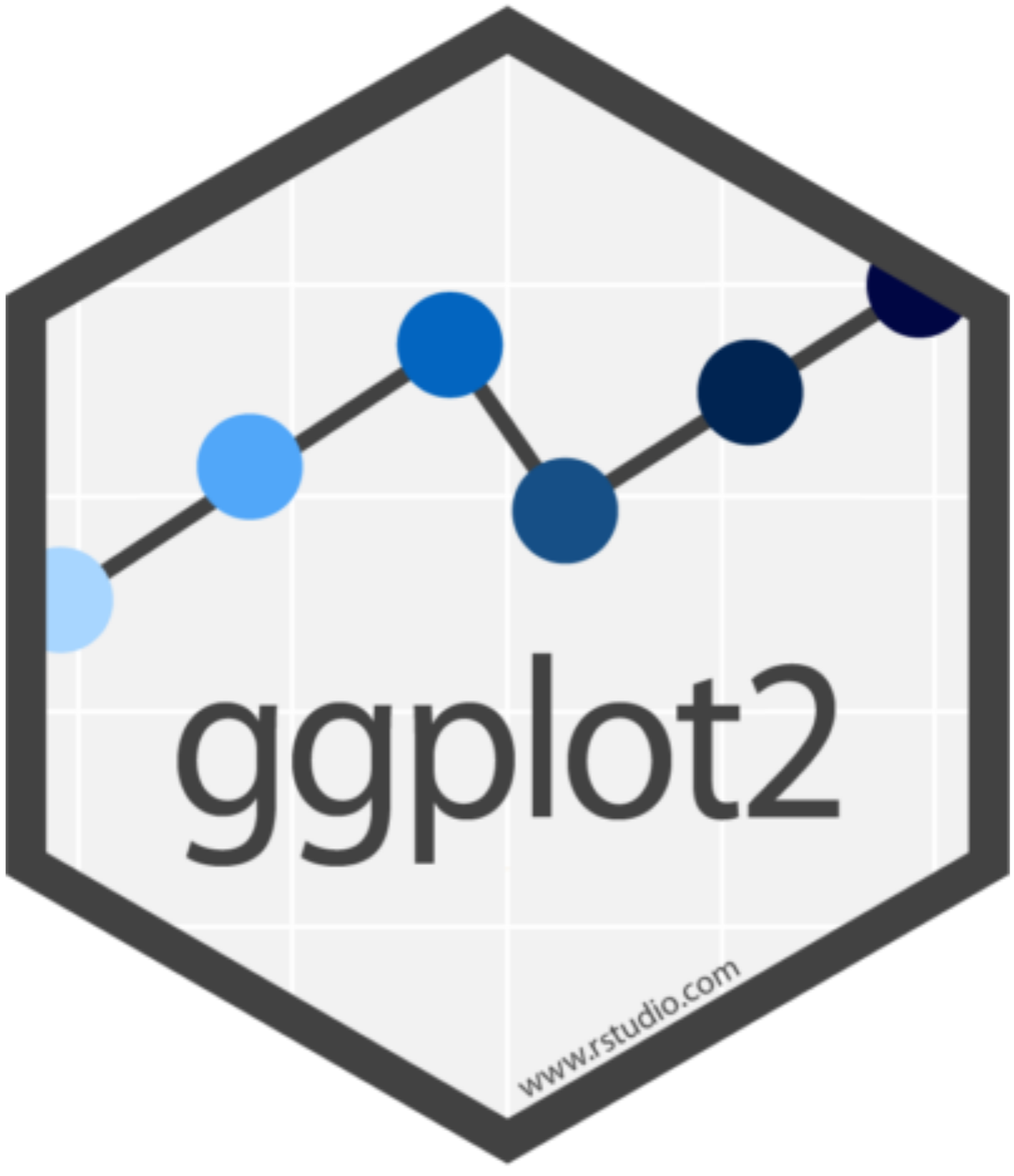
- A list of pizza restaurants, 3,500 pizzas, and their menu prices.

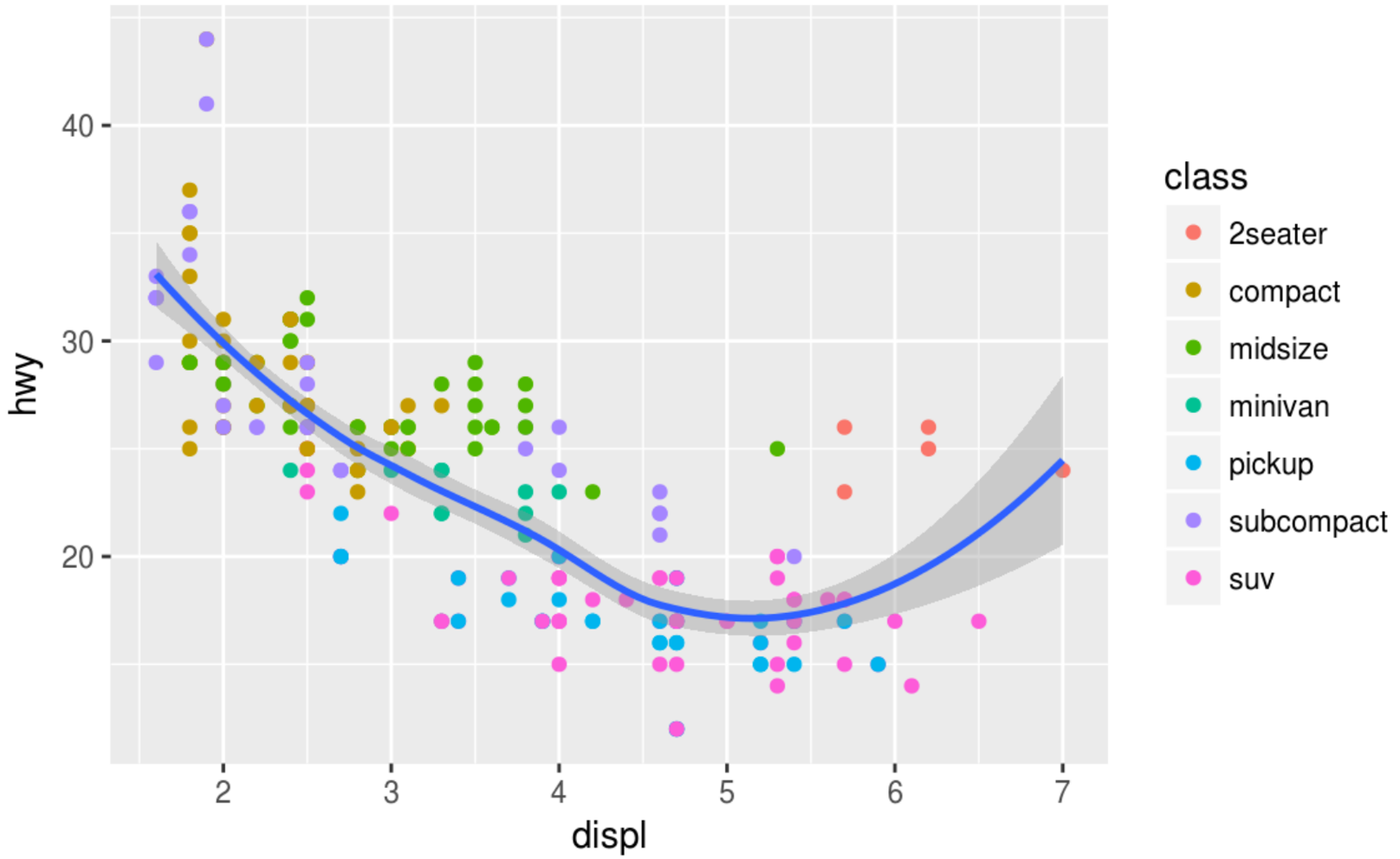
- Download ‘restaurants.csv’ and ‘menus.csv’ from github.com/UtrechtUniversity/workshop-introduction-to-R-and-data

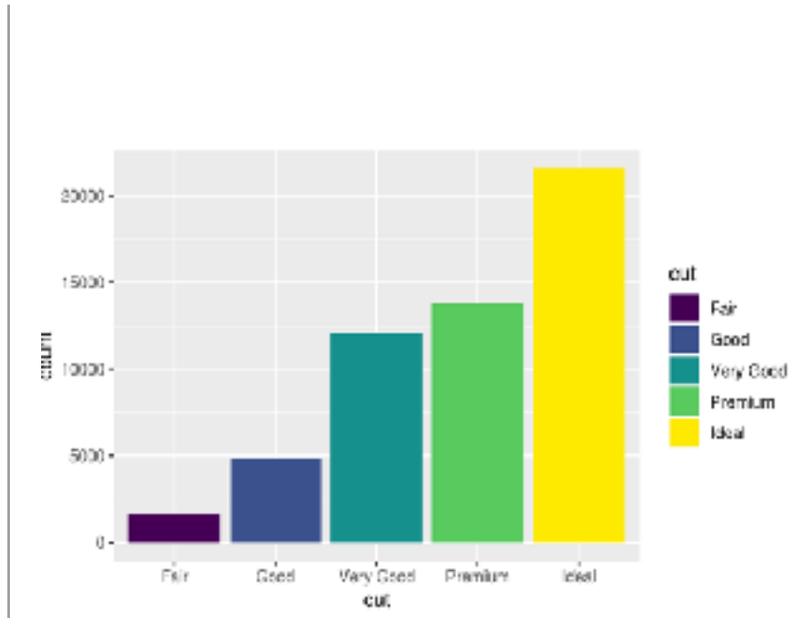
- Create a folder ‘data’ in your project and add the datasets.

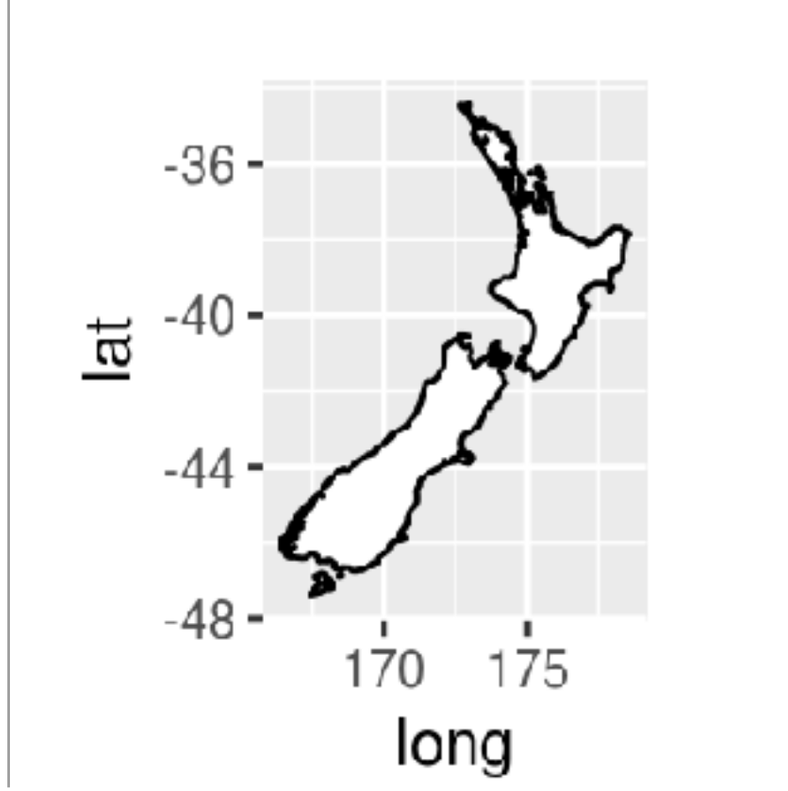
Data visualisation

Create eye candy with ggplot2







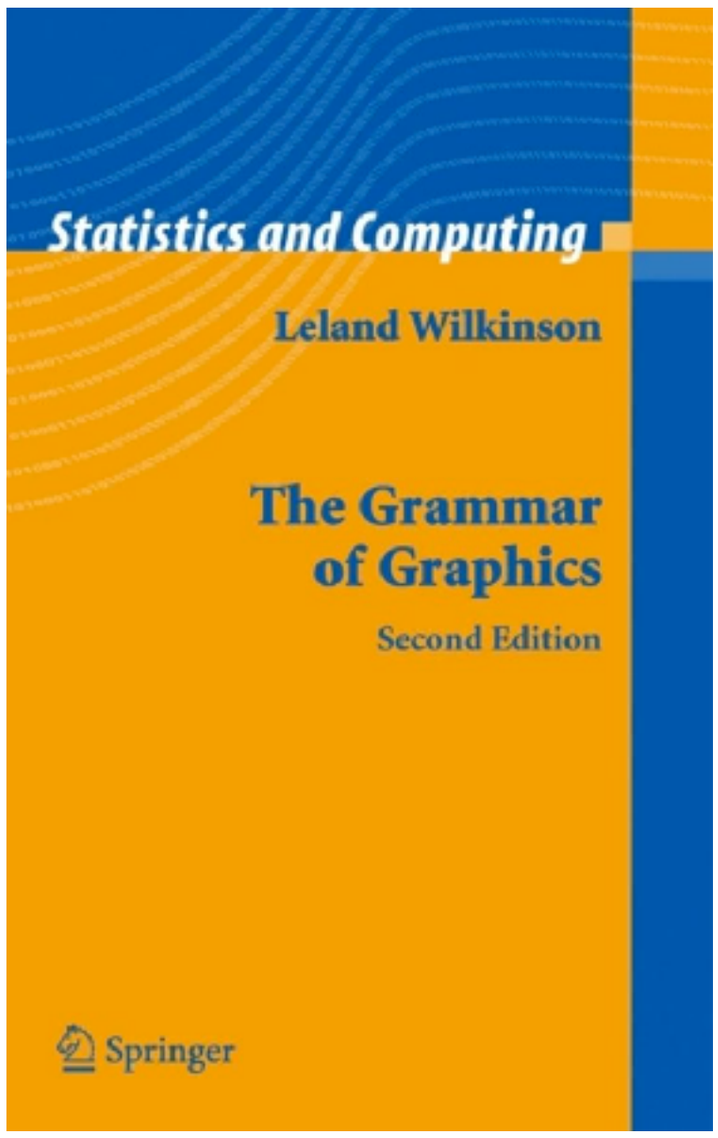


Examples of plots with R (and ggplot2)

Data visualisation with ggplot2

- **ggplot2** is an extremely popular data visualisation package for R
- Simple syntax, easy to learn, nice plots
- Developed and maintained by Hadley Wickham
- Based on the book: The Grammar of Graphics (Wilkinson, 2005)

The grammar of graphics



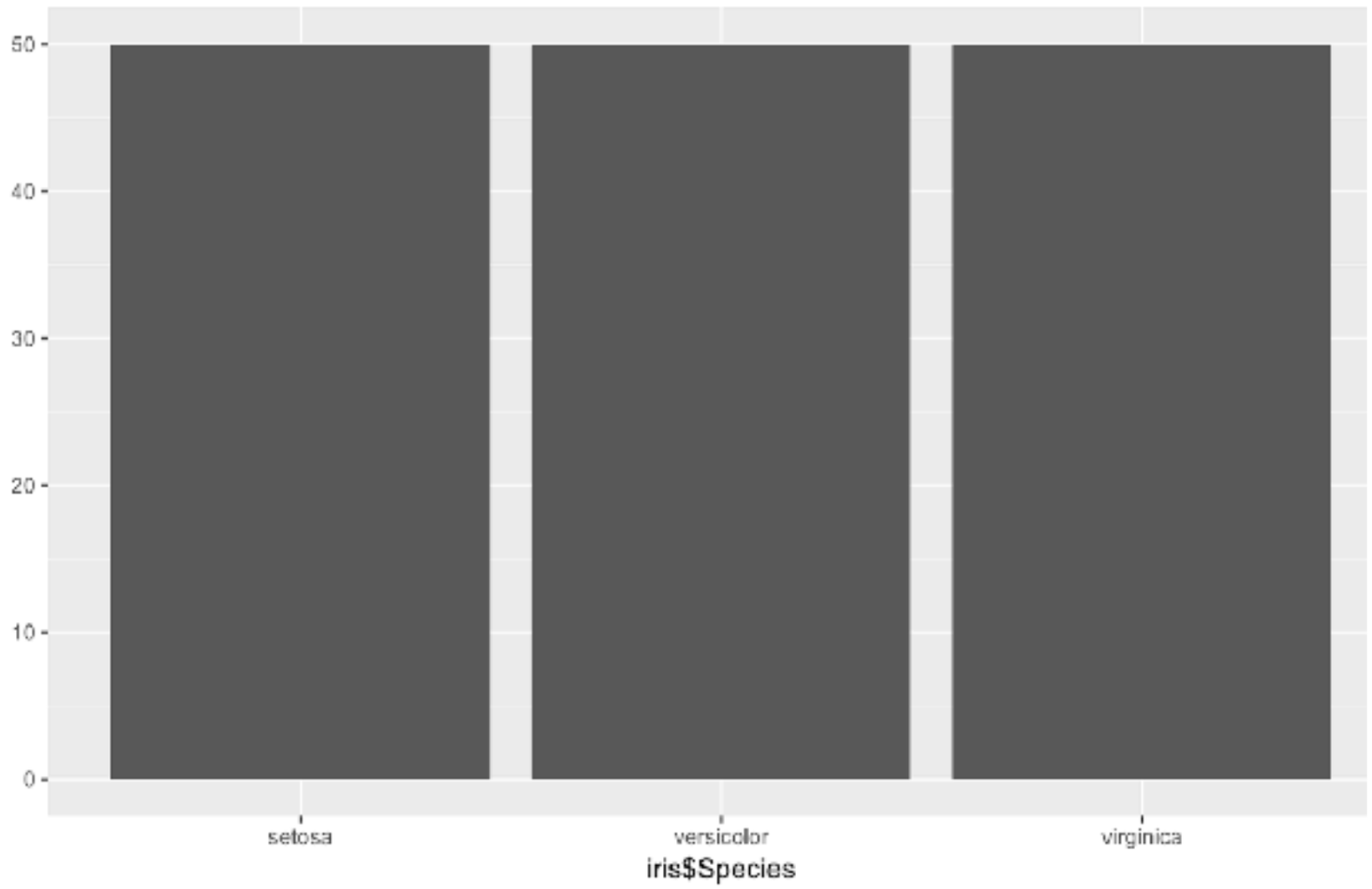
Data	The variables in a tibble or data.frame
Aesthetics	x- and y-axis, colour, size, alpha, shape
Geometries	point, line, bar, histogram

Source: https://www.youtube.com/watch?v=uiTc55clwuA

Quick plotting

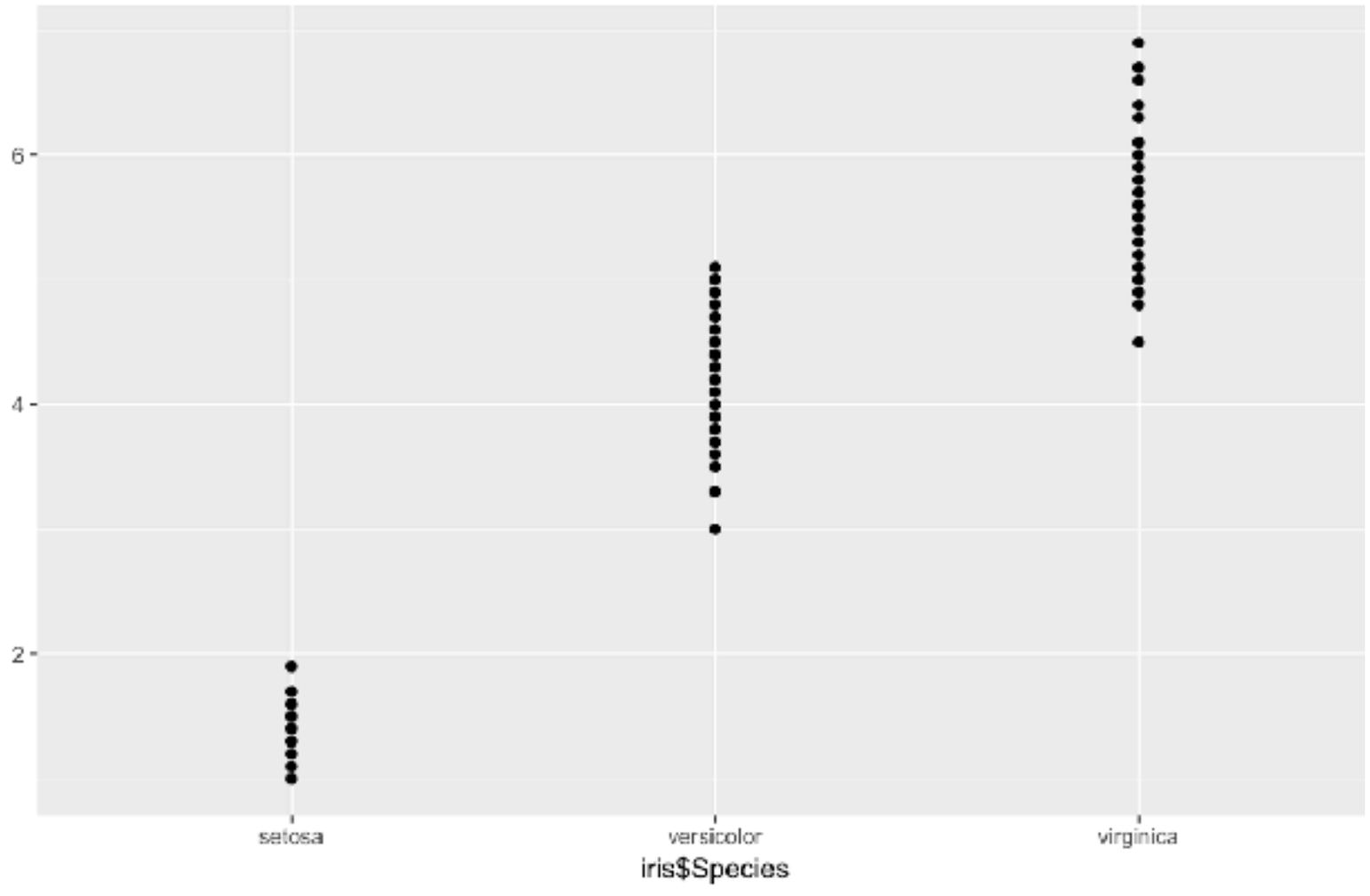
Quick plots

```
> qplot(iris$Species)
```



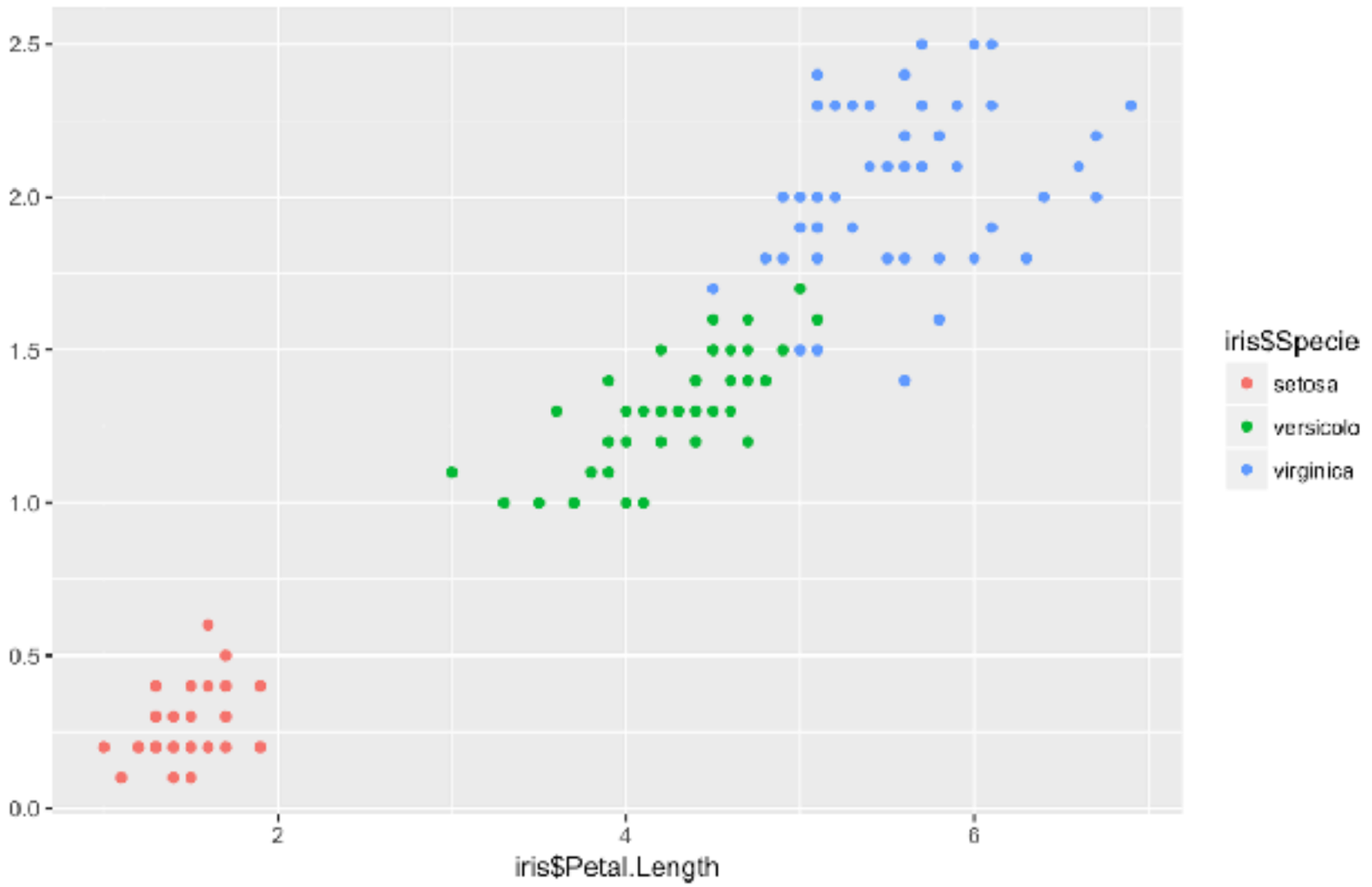
Quick plots

```
> qplot(iris$Species)
> qplot(iris$Species, iris$Petal.Length)
```



Quick plots

```
> qplot(iris$Species)
> qplot(iris$Species, iris$Petal.Length)
> qplot(iris$Petal.Length, iris$Petal.Width, colour=iris$Species)
```



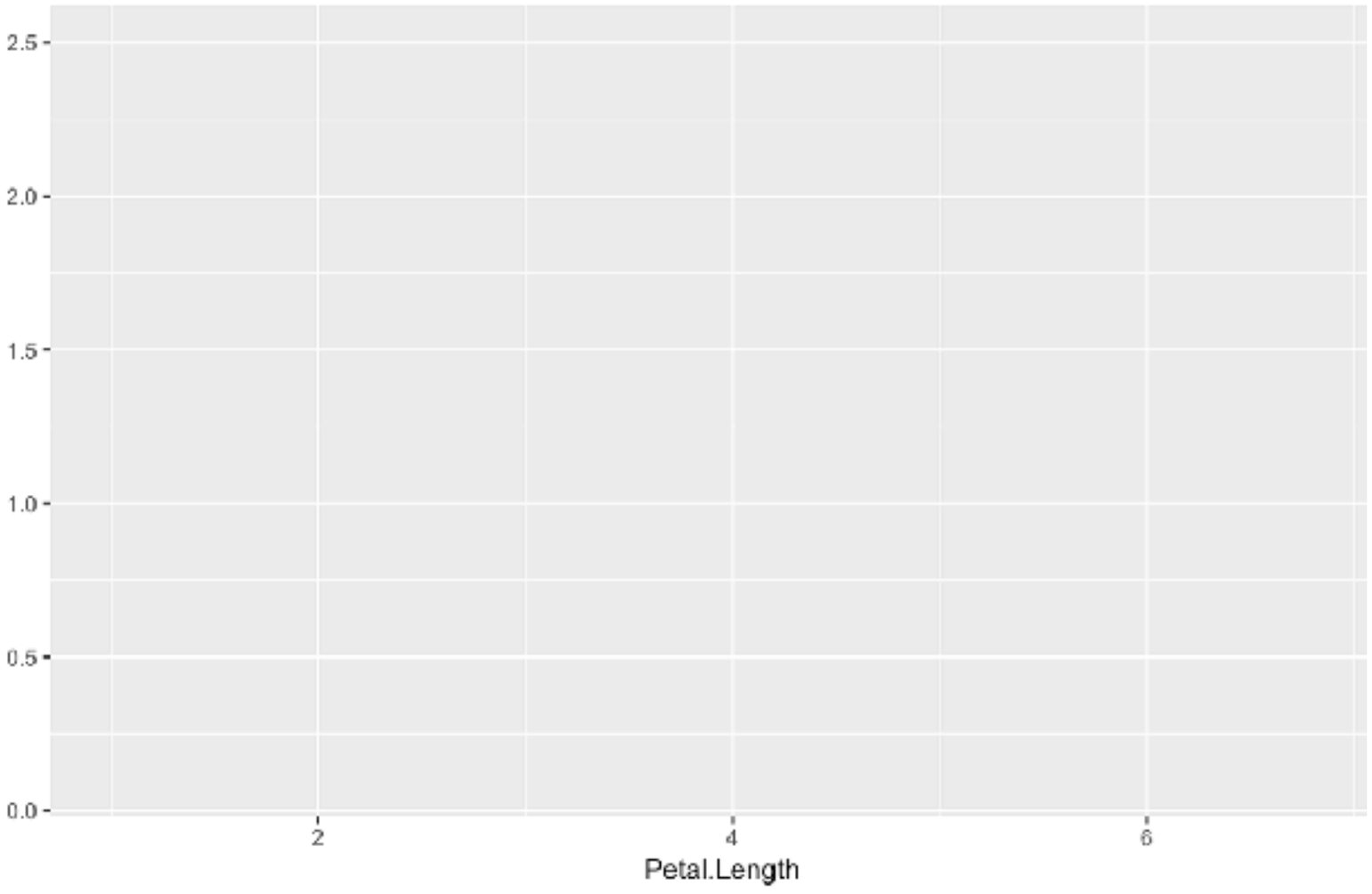
ggplot (grammar of graphics)

ggplot: Data, aesthetics, geometrics

```
> ggplot(iris)
```

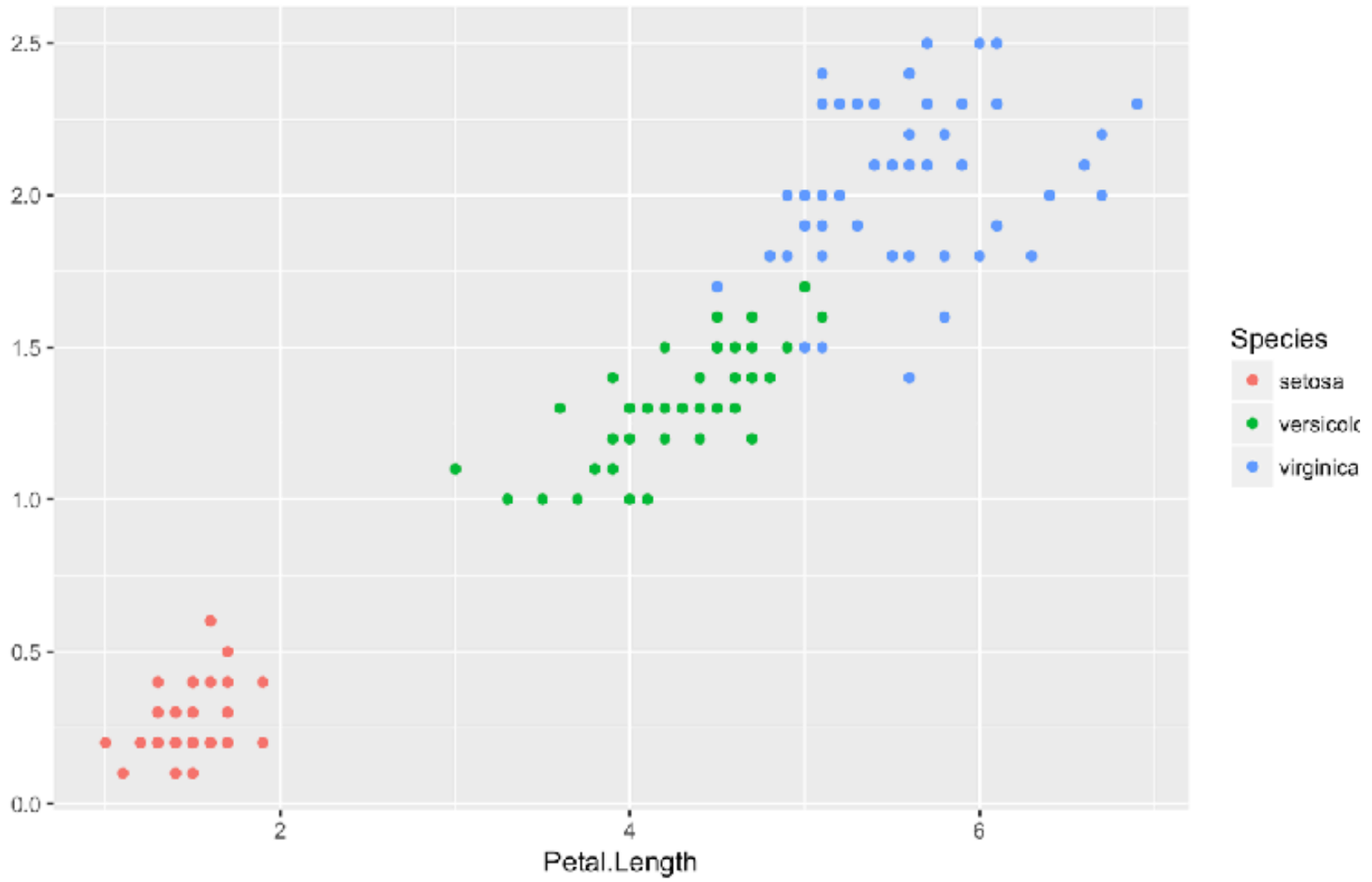
ggplot: Data, aesthetics, geometrics

```
> ggplot(iris)
> ggplot(iris, aes(x = Petal.Length, y = Petal.Width, colour=Species))
```



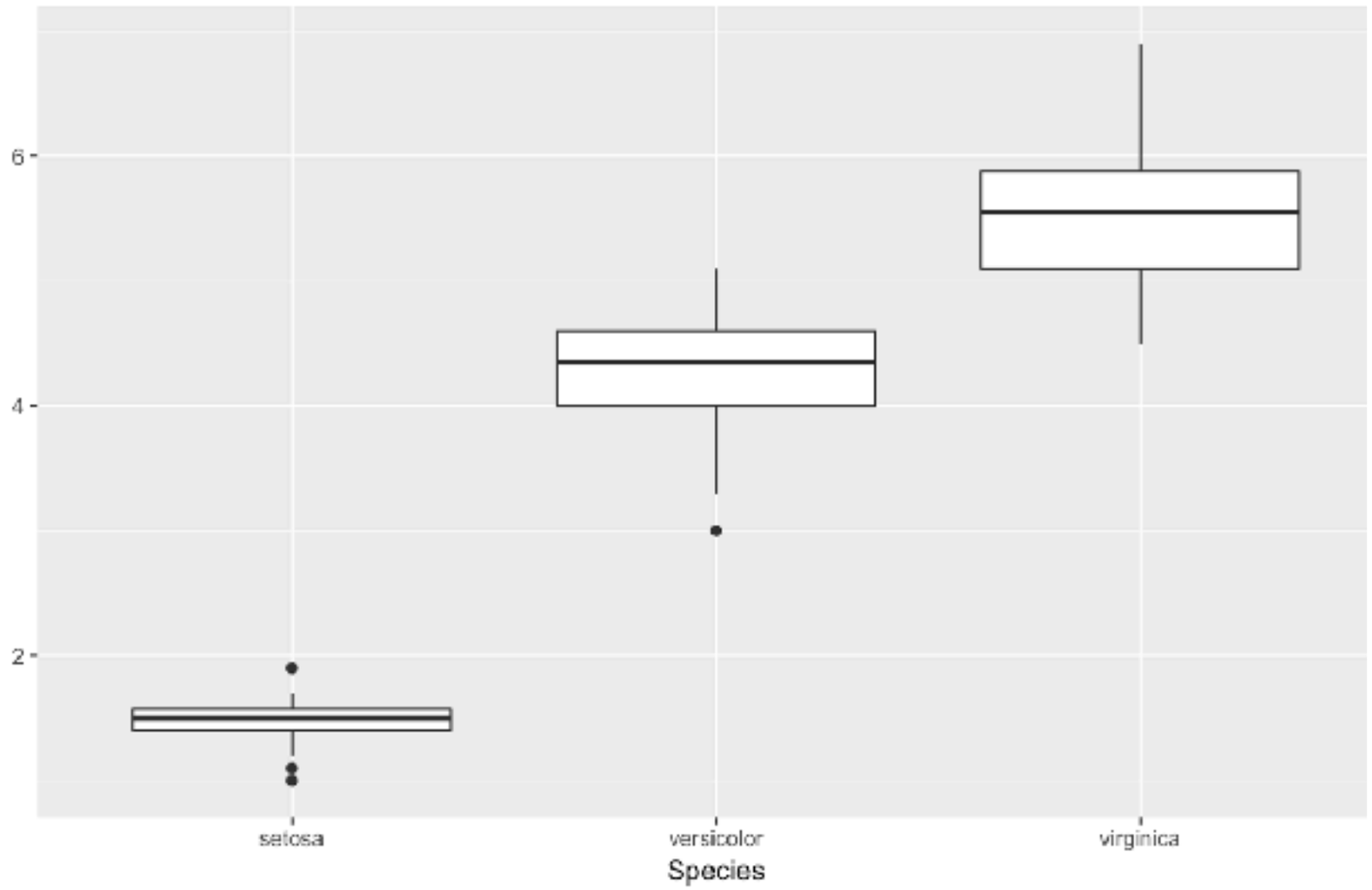
ggplot: Data, aesthetics, geometrics

```
> ggplot(iris)
> ggplot(iris, aes(x = Petal.Length, y = Petal.Width, colour=Species))
> ggplot(iris, aes(x = Petal.Length, y = Petal.Width, colour=Species)) +
    geom_point()
```



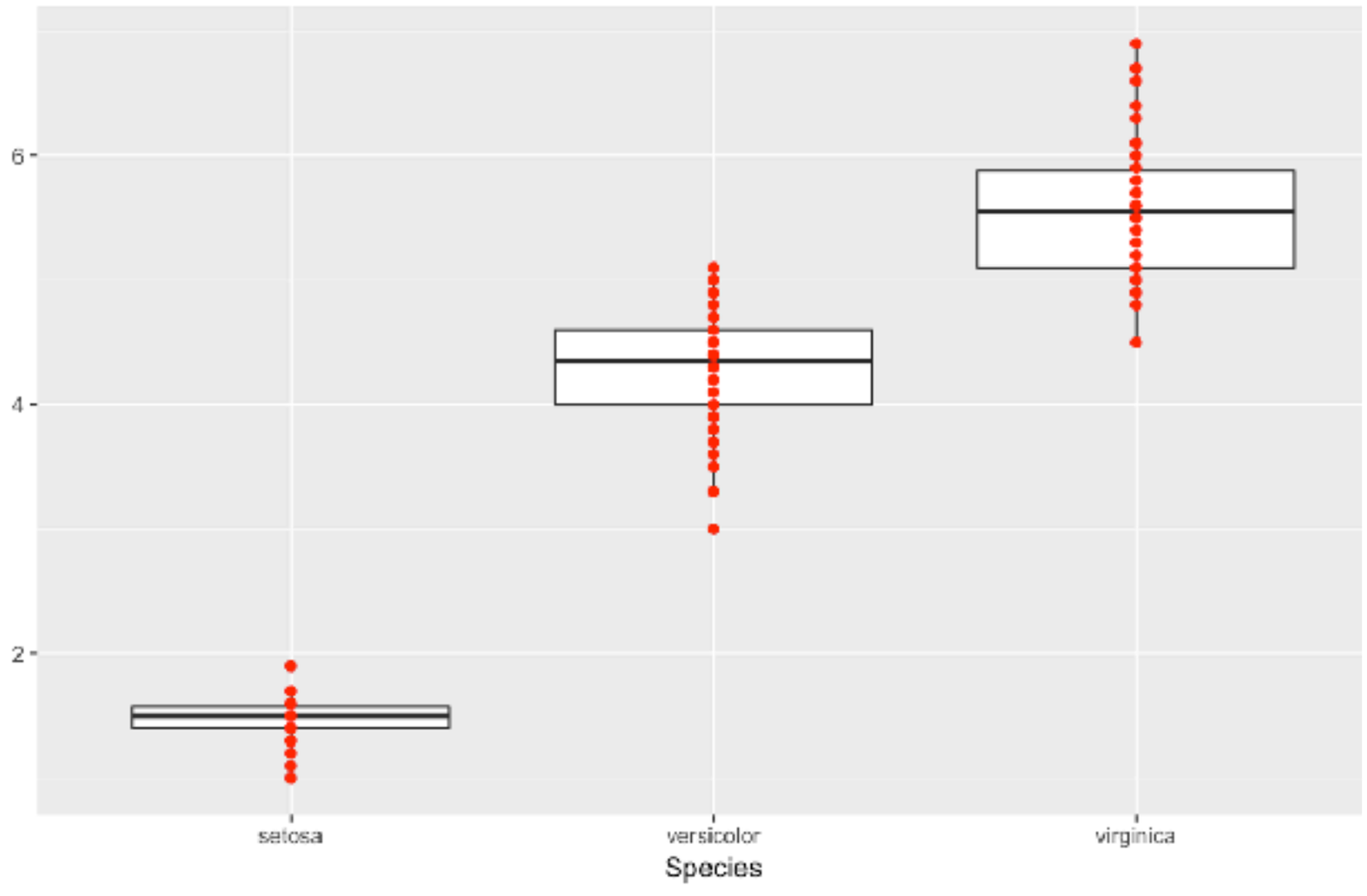
ggplot: Multiple geometric layers

```
> ggplot(iris, aes(x = Species, y = Petal.Length)) +
      geom_boxplot()
```



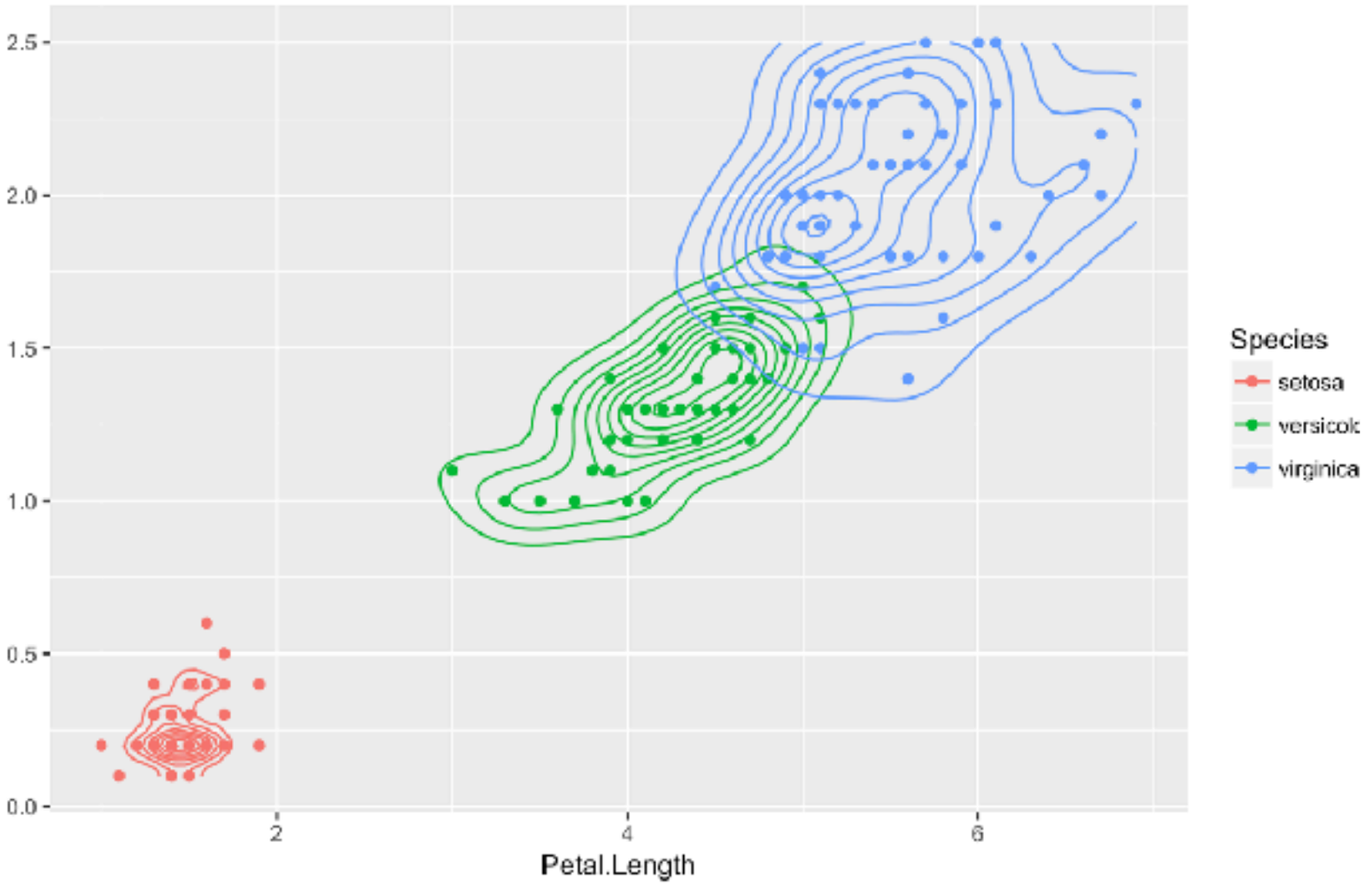
ggplot: Multiple geometric layers

```
> ggplot(iris, aes(x = Species, y = Petal.Length)) +
      geom_boxplot() +
      geom_point(colour='red')
```



ggplot: Statistical layers

```
> ggplot(iris, aes(x = Petal.Length, y = Petal.Width, colour=Species)) +
    geom_point() +
    stat_density_2d()
```



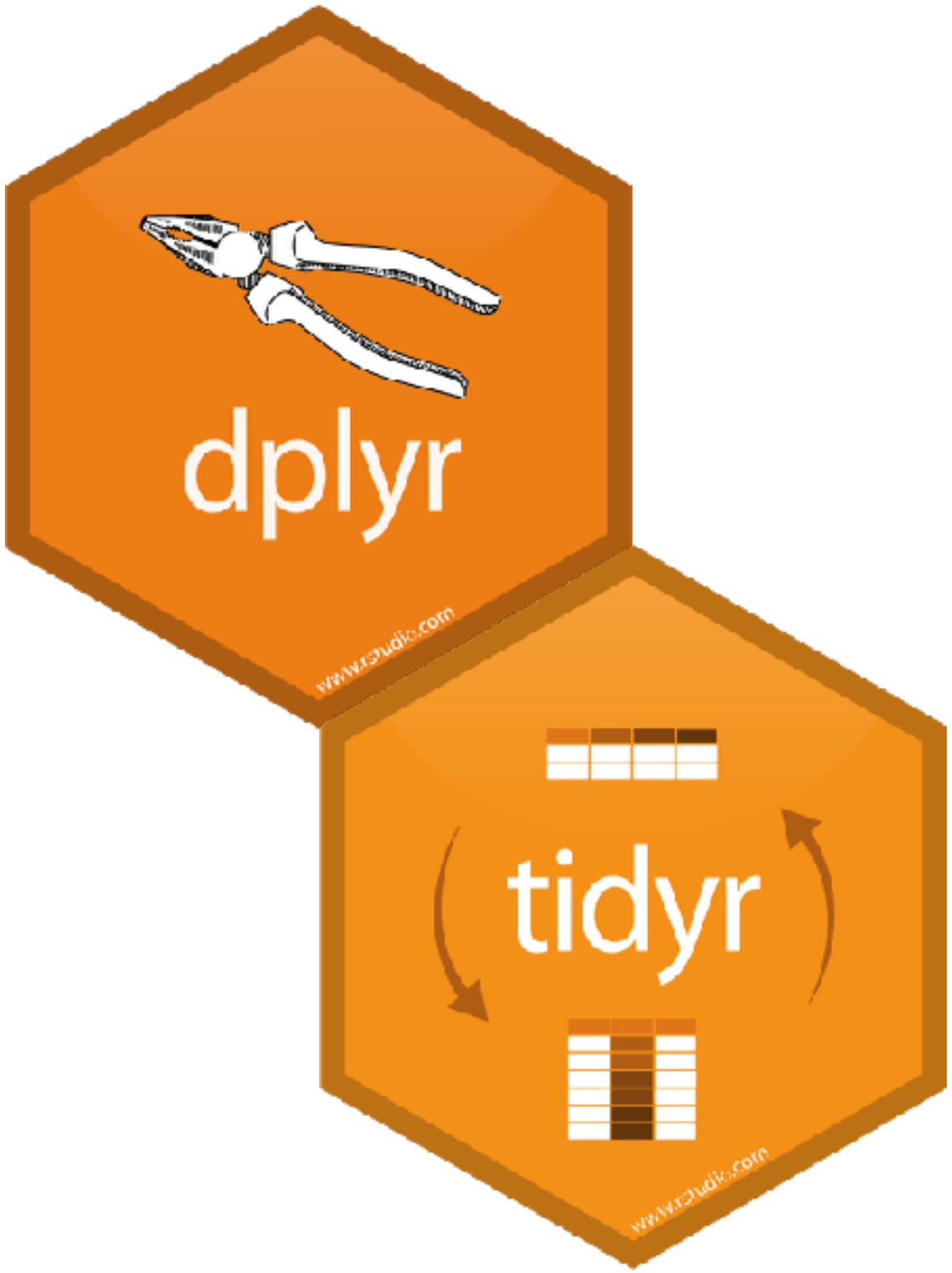
Exercises

Please do Basic exercises 2.1 and 2.2

Time left? Opt for a reading exercise or optional exercise!

Data transformation

Explore the power of dplyr and tidyr



Data Wrangling
with dplyr and tidyr
Cheat Sheet

RStudio

Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

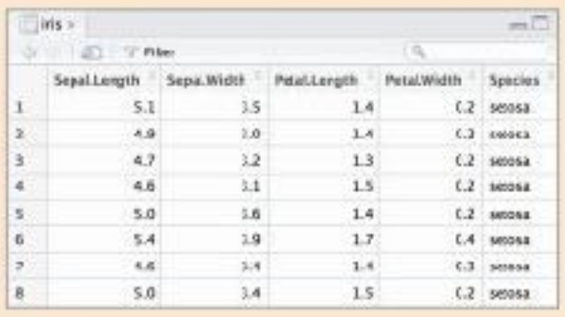
```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

View data set in spreadsheet-like display (note capital V).



dplyr::**%>%**

Passes object on left hand side as first argument (or . argument) of function on righthand side.

`x %>% f(y)` *is the same as* `f(x, y)`

`y %>% f(x, ., z)` *is the same as* `f(x, y, z )`

"Piping" with %>% makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Tidy Data - A foundation for wrangling in R

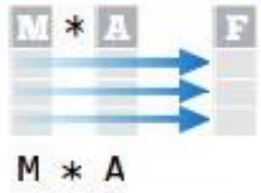
In a tidy data set:

Each **variable** is saved in its own **column**

&

Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



Reshaping Data - Change the layout of a data set

tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.

tidyr::**spread(pollution, size, amount)**

Spread rows into columns.

tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.

tidyr::**unite(data, col, ..., sep)**

Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**

Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**

Order rows by values of a column (low to high).

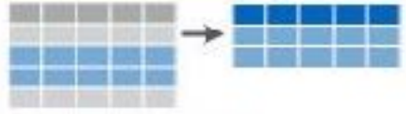
dplyr::**arrange(mtcars, desc(mpg))**

Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**

Rename the columns of a data frame.

Subset Observations (Rows)



dplyr::**filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**

Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**

Randomly select n rows.

dplyr::**slice(iris, 10:15)**

Select rows by position.

dplyr::**top_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, |, !, xor, any, all	Boolean operators

Subset Variables (Columns)

dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, **contains(".")**)

Select columns whose name contains a character string.

select(iris, **ends_with("Length")**)

Select columns whose name ends with a character string.

select(iris, **everything()**)

Select every column.

select(iris, **matches("..")**)

Select columns whose name matches a regular expression.

select(iris, **num_range("x", 1:5)**)

Select columns named x1, x2, x3, x4, x5.

select(iris, **one_of(c("Species", "Genus"))**)

Select columns whose names are in a group of names.

select(iris, **starts_with("Sepal")**)

Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)

Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)

Select all columns except Species.

filter()

Function: filter()

```
> filter(iris, Species=="virginica")
```

Function: filter()

```
> filter(iris, Species=="virginica")
# A tibble: 50 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          6.30        3.30         6.00        2.50 virginica
 2          5.80        2.70         5.10        1.90 virginica
 3          7.10        3.00         5.90        2.10 virginica
 4          6.30        2.90         5.60        1.80 virginica
 5          6.50        3.00         5.80        2.20 virginica
 6          7.60        3.00         6.60        2.10 virginica
 7          4.90        2.50         4.50        1.70 virginica
 8          7.30        2.90         6.30        1.80 virginica
 9          6.70        2.50         5.80        1.80 virginica
10          7.20        3.60         6.10        2.50 virginica
# ... with 40 more rows
```

Function: filter()

```
> filter(iris, Species=="virginica", Sepal.Length>= 7.5)
```

Function: filter()

```
> filter(iris, Species=="virginica", Sepal.Length>= 7.5)
# A tibble: 6 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <fct>
1          7.60        3.00         6.60        2.10 virginica
2          7.70        3.80         6.70        2.20 virginica
3          7.70        2.60         6.90        2.30 virginica
4          7.70        2.80         6.70        2.00 virginica
5          7.90        3.80         6.40        2.00 virginica
6          7.70        3.00         6.10        2.30 virginica
```

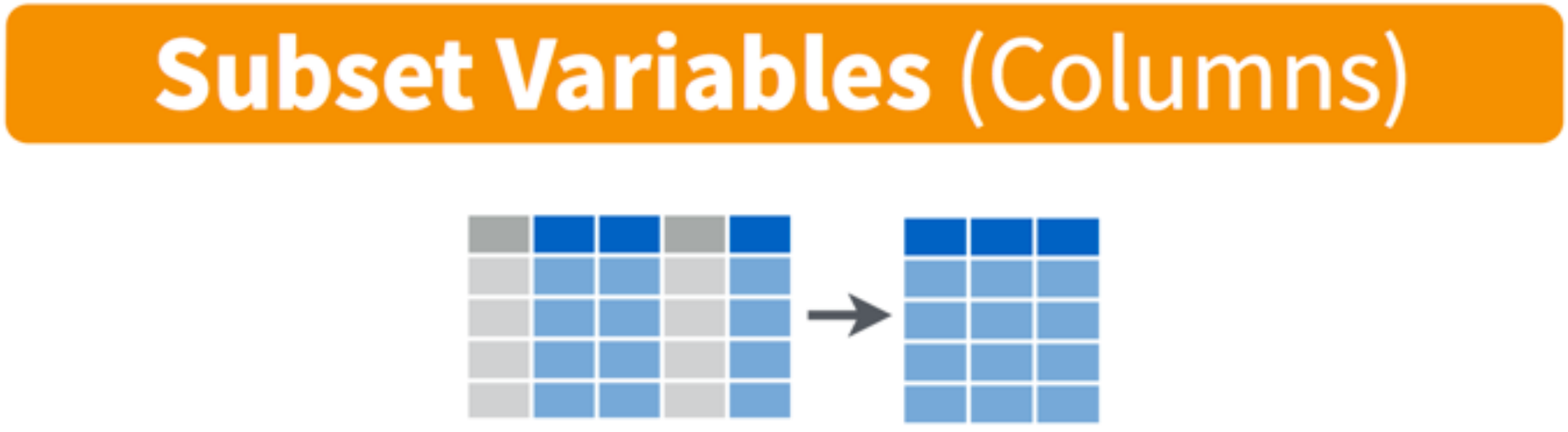
Function: filter()

```
> filter(iris, Species=="virginica", Sepal.Length>= 7.5)
# A tibble: 6 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <fct>
1          7.60        3.00         6.60        2.10 virginica
2          7.70        3.80         6.70        2.20 virginica
3          7.70        2.60         6.90        2.30 virginica
4          7.70        2.80         6.70        2.00 virginica
5          7.90        3.80         6.40        2.00 virginica
6          7.70        3.00         6.10        2.30 virginica
```

Same as:

```
> filter(iris, Species==“virginica" & Sepal.Length>= 7.5)
```

select()



Function: select()

```
> select(iris, Sepal.Length, Sepal.Width, Species)
```

Function: select()

```
> select(iris, Sepal.Length, Sepal.Width, Species)
# A tibble: 150 x 3
   Sepal.Length Sepal.Width Species
          <dbl>       <dbl> <fct>
 1          5.10        3.50 setosa
 2          4.90        3.00 setosa
 3          4.70        3.20 setosa
 4          4.60        3.10 setosa
 5          5.00        3.60 setosa
 6          5.40        3.90 setosa
 7          4.60        3.40 setosa
 8          5.00        3.40 setosa
 9          4.40        2.90 setosa
10          4.90        3.10 setosa
# ... with 140 more rows
```

Function: select()

```
> select(iris, Sepal.Length, Sepal.Width, Species)
> select(iris, starts_with("Sepal"), Species)
```

Function: select()

```
> select(iris, Sepal.Length, Sepal.Width, Species)
> select(iris, starts_with("Sepal"), Species)
# A tibble: 150 x 3
   Sepal.Length Sepal.Width Species
          <dbl>       <dbl> <fct>
 1          5.10        3.50 setosa
 2          4.90        3.00 setosa
 3          4.70        3.20 setosa
 4          4.60        3.10 setosa
 5          5.00        3.60 setosa
 6          5.40        3.90 setosa
 7          4.60        3.40 setosa
 8          5.00        3.40 setosa
 9          4.40        2.90 setosa
10          4.90        3.10 setosa
# ... with 140 more rows
```

Function: select()

```
> select(iris, Sepal.Length, Sepal.Width, Species)
> select(iris, starts_with("Sepal"), Species)
> select(iris, -starts_with("Petal"))
```

Function: select()

```
> select(iris, Sepal.Length, Sepal.Width, Species)
> select(iris, starts_with("Sepal"), Species)
> select(iris, -starts_with("Petal"))
# A tibble: 150 x 3
   Sepal.Length Sepal.Width Species
          <dbl>       <dbl> <fct>
 1          5.10        3.50 setosa
 2          4.90        3.00 setosa
 3          4.70        3.20 setosa
 4          4.60        3.10 setosa
 5          5.00        3.60 setosa
 6          5.40        3.90 setosa
 7          4.60        3.40 setosa
 8          5.00        3.40 setosa
 9          4.40        2.90 setosa
10          4.90        3.10 setosa
# ... with 140 more rows
```

mutate()

Function: mutate()

```
> mutate(iris,
         petal_area = pi * Petal.Length * Petal.Width)
```

Function: mutate()

```
> mutate(iris,
         petal_area = pi * Petal.Length * Petal.Width)
# A tibble: 150 x 6
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species petal_area
          <dbl>       <dbl>        <dbl>       <dbl> <fct>        <dbl>
 1          5.10        3.50         1.40       0.200 setosa       0.880
 2          4.90        3.00         1.40       0.200 setosa       0.880
 3          4.70        3.20         1.30       0.200 setosa       0.817
 4          4.60        3.10         1.50       0.200 setosa       0.942
 5          5.00        3.60         1.40       0.200 setosa       0.880
 6          5.40        3.90         1.70       0.400 setosa       2.14
 7          4.60        3.40         1.40       0.300 setosa       1.32
 8          5.00        3.40         1.50       0.200 setosa       0.942
 9          4.40        2.90         1.40       0.200 setosa       0.880
10          4.90        3.10         1.50       0.100 setosa       0.471
# ... with 140 more rows
```

Function: mutate()

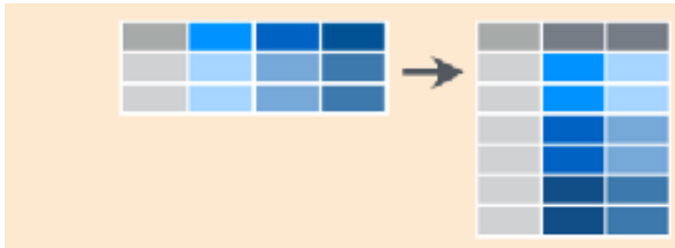
```
> mutate(iris,
         # compute the area of a single petal
         petal_area = pi * Petal.Length * Petal.Width,
         # abbreviate the name of the species
         Species_abbr = substring(Species, 1, 3)
)
```

Function: mutate()

```
> mutate(iris,
         # compute the area of a single petal
         petal_area = pi * Petal.Length * Petal.Width,
         # abbreviate the name of the species
         Species_abbr = substring(Species, 1, 3)
)
```

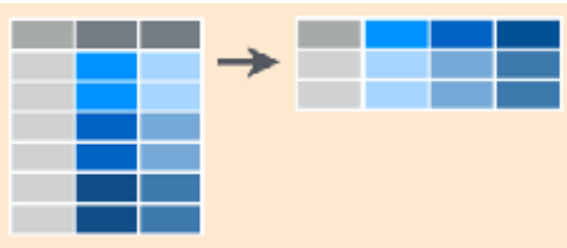
```
# A tibble: 6 x 7
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species petal_area Species_abbr
         <dbl>       <dbl>        <dbl>       <dbl> <fct>        <dbl> <chr>
1          5.10        3.50         1.40       0.200 setosa       0.880 set
2          4.90        3.00         1.40       0.200 setosa       0.880 set
3          4.70        3.20         1.30       0.200 setosa       0.817 set
4          4.60        3.10         1.50       0.200 setosa       0.942 set
5          5.00        3.60         1.40       0.200 setosa       0.880 set
6          5.40        3.90         1.70       0.400 setosa       2.14  set
```

gather() and spread()



tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.



tidyr::**spread(pollution, size, amount)**

Spread rows into columns.

Function: gather()

```
> iris_obs <- mutate(iris, observation = 1:n())
> iris_obs
```

Function: gather()

```
> iris_obs <- mutate(iris, observation = 1:n())
> iris_obs
# A tibble: 150 x 6
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species observation
          <dbl>       <dbl>        <dbl>       <dbl> <chr>          <int>
 1          5.10        3.50         1.40       0.200 setosa             1
 2          4.90        3.00         1.40       0.200 setosa             2
 3          4.70        3.20         1.30       0.200 setosa             3
 4          4.60        3.10         1.50       0.200 setosa             4
 5          5.00        3.60         1.40       0.200 setosa             5
 6          5.40        3.90         1.70       0.400 setosa             6
 7          4.60        3.40         1.40       0.300 setosa             7
 8          5.00        3.40         1.50       0.200 setosa             8
 9          4.40        2.90         1.40       0.200 setosa             9
10          4.90        3.10         1.50       0.100 setosa            10
# ... with 140 more rows
```

Function: gather()

```
> iris_obs <- mutate(iris, observation = 1:n())
> iris_obs
> iris_long <- gather(iris_obs, measurement, value, -Species, -observation)
> iris_long
```

Function: gather()

```
> iris_obs <- mutate(iris, observation = 1:n())
> iris_obs
> iris_long <- gather(iris_obs, measurement, value, -Species, -observation)
> iris_long
# A tibble: 600 x 4
   Species observation measurement  value
   <chr>         <int> <chr>        <dbl>
 1 setosa            1 Sepal.Length 5.10
 2 setosa            1 Sepal.Width  3.50
 3 setosa            1 Petal.Length 1.40
 4 setosa            1 Petal.Width  0.200
 5 setosa            2 Sepal.Length 4.90
 6 setosa            2 Sepal.Width  3.00
 7 setosa            2 Petal.Length 1.40
 8 setosa            2 Petal.Width  0.200
 9 setosa            3 Sepal.Length 4.70
10 setosa            3 Sepal.Width  3.20
```

Function: spread()

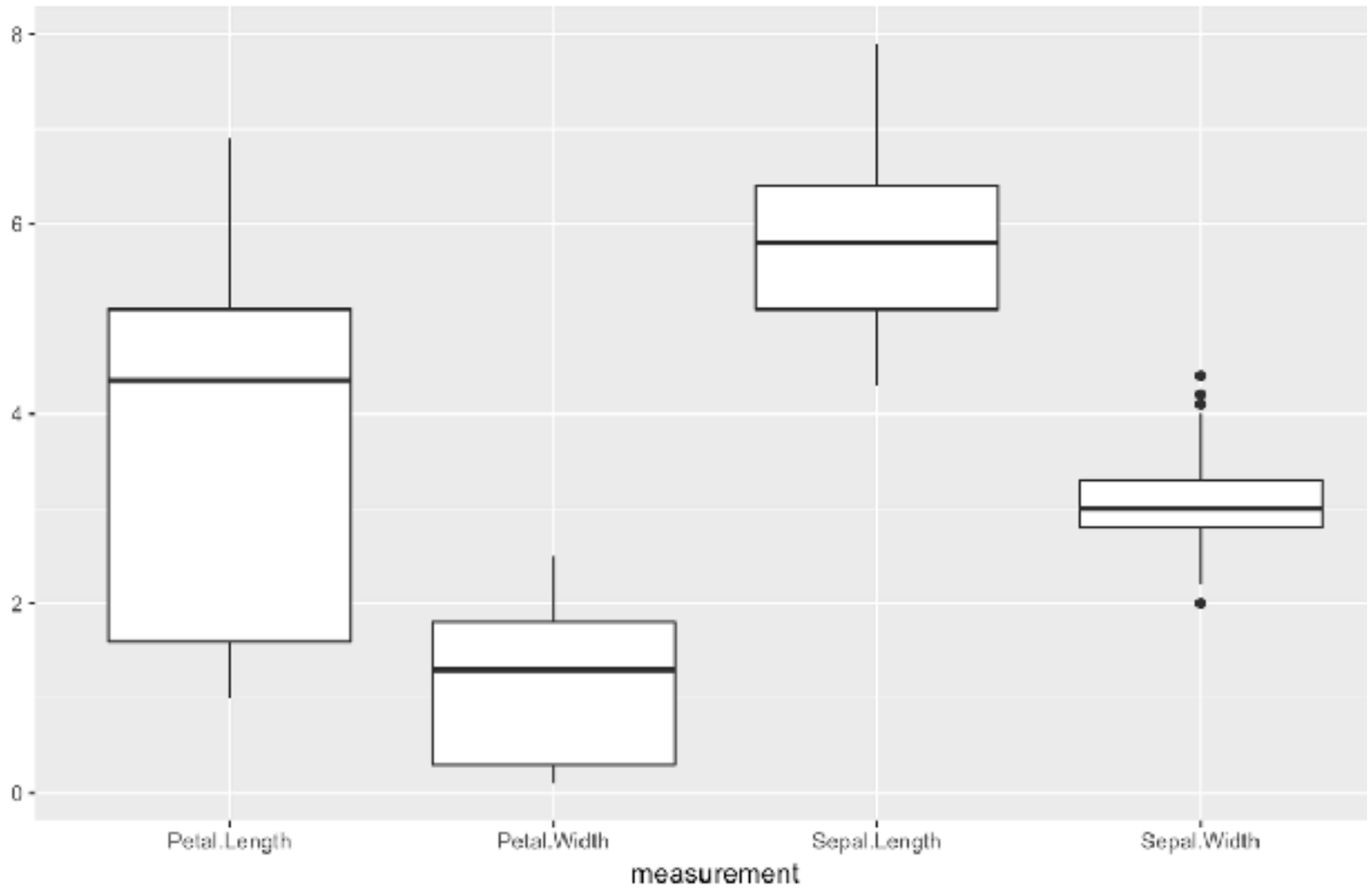
```
> iris_wide <- spread(iris_long, measurement, value)
> iris_wide
```

Function: spread()

```
> iris_wide <- spread(iris_long, measurement, value)
> iris_wide
# A tibble: 150 x 6
   Species observation Petal.Length Petal.Width Sepal.Length Sepal.Width
   <chr>         <int>        <dbl>       <dbl>        <dbl>       <dbl>
 1 setosa            1         1.40       0.200         5.10        3.50
 2 setosa            2         1.40       0.200         4.90        3.00
 3 setosa            3         1.30       0.200         4.70        3.20
 4 setosa            4         1.50       0.200         4.60        3.10
 5 setosa            5         1.40       0.200         5.00        3.60
 6 setosa            6         1.70       0.400         5.40        3.90
 7 setosa            7         1.40       0.300         4.60        3.40
 8 setosa            8         1.50       0.200         5.00        3.40
 9 setosa            9         1.40       0.200         4.40        2.90
10 setosa           10         1.50       0.100         4.90        3.10
# ... with 140 more rows
```

Quick plots

```
> ggplot(iris_long, aes(measurement, value)) + geom_boxplot()
```



Exercises

Please do basic exercises 3.1 and 3.2.

Time left? Opt for a reading exercise or optional exercise!

Other RDM workshops

- https://www.uu.nl/en/research/research-data-management/tools-services/training-and-workshops

- Learn to write your Data Management Plan (online course)
- Research Data Management basics
- Introduction to Python & Data (coming soon)
- Introduction to computational reproducibility (coming soon)

R Cafe

- Monthly event for R programmers
- Join the R community
- Work on your project with the ability to ask questions

- Subscribe to the newsletter or follow RDM support website
- Check https://github.com/UtrechtUniversity/R-data-cafe
- Next edition: **Monday 12/11!**

Feedback is cool.

Please send your feedback, remarks, questions to info.rdm@uu.nl.

Or to us!

Barbara: b.m.i.vreede@uu.nl
Jonathan: j.debruin1@uu.nl

```
useR <- function(){
    print("Good luck and see you!")
}
useR()
```