

**Do you have a github account?
Give us a star! :)**

Welcome!

Please download the course material:

tinyurl.com/introRData > [Clone or download ▾](#) > [Download ZIP](#)

and store it in a single, accessible folder on your computer.

Introduction to R & data

Outline

- Part I: Basics of R
- Lunch (~12:15)
- Part II: Modern R with tidyverse
- Final remarks (~17:00)

move the samples from the data
* {r}
select the samples to keep
keepsamples <- row.names(pheno)
apply sample selection to counts
counts.sub <- counts.sub[,keepsamples]
pheno.sub <- pheno[keepsamples]
mt.assay <- mt.assay[,keepsamples]
rld.sub <- rld[,keepsamples]

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 & data

Part 1: Basics of R



What is R?

- Widely used programming language for data analysis
- Based on statistical programming language **S** (1976)
- Developed by **Ross Ihaka & Robert Gentleman** (1995)
- Very active community, with many (often subject-specific) packages



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!

Have you downloaded the course material?

tinyurl.com/introRData > [Clone or download](#) > [Download ZIP](#)

Please store it in a single, accessible folder on your computer.

R syntax & the RStudio console

Variable assignment

>

Variable assignment

```
> x <- 1
```

Variable assignment

```
> x <- 1
```

```
> 1 -> x
```

```
> x = 1
```

```
> 1 = x
```

```
Error in 1 = x : invalid (do_set) left-hand side to assignment
```

Base R

Cheat Sheet

Variable assignment

```
> x <- 6  
> hi <- "hello world"  
> x * 3  
[1] 18  
> y <- x + 2  
> log2(y)  
[1] 3
```

Maths Functions

<code>log(x)</code>	Natural log.	<code>sum(x)</code>	Sum.
<code>exp(x)</code>	Exponential.	<code>mean(x)</code>	Mean.
<code>max(x)</code>	Largest element.	<code>median(x)</code>	Median.
<code>min(x)</code>	Smallest element.	<code>quantile(x)</code>	Percentage quantiles.
<code>round(x, n)</code>	Round to n decimal places.	<code>rank(x)</code>	Rank of elements.
<code>signif(x, n)</code>	Round to n significant figures.	<code>var(x)</code>	The variance.
<code>cor(x, y)</code>	Correlation.	<code>sd(x)</code>	The standard deviation.

Variable Assignment

```
> a <- 'apple'  
> a  
[1] 'apple'
```

A quick note on notes

- The console is for execution, not for storage
- Everything we do is on the slides!
- BUT: you can store, if you want, by copy-pasting to a text document
- Do you want to write notes in between?

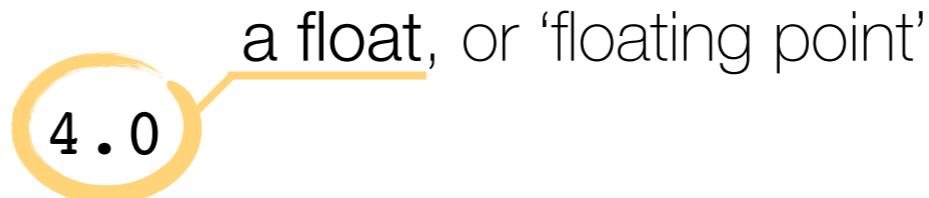
```
> # write your notes in the console like this  
> # using a #hash sign.  
> # pressing enter will do nothing.  
> # go ahead and try!
```

Creating vectors

```
> 1:5  
[1] 1 2 3 4 5  
> seq(5)  
[1] 1 2 3 4 5  
> seq(2,4,by=0.5)  
[1] 2.0 2.5 3.0 3.5  
> rep(4,6)  
[1] 4 4 4 4 4 4  
> c(1,61,101)  
[1] 1 61 101  
> c(2:4,rep(8,3))  
[1] 2 3 4 8 8 8
```



vs.



Vector functions

```
> p <- 1:5  
> mean(p)  
[1] 3  
> p * 2  
[1] 2 4 6 8 10  
> q <- 5:1  
> p * q  
[1] 5 8 9 8 5  
> table(p*q)  
5 8 9  
2 2 1
```

p * 2		
p	2	
1	2	2
2	2	4
3	2	6
4	2	8
5	2	10

p * q		
p	q	
1	5	5
2	4	8
3	3	9
4	2	8
5	1	5

Exercise

1. Write a line of code that produces the following vector:

(Note: there are multiple possible answers!)

[1] 4 4 4 10 15 20 10 7 4

2. Divide the vector by 3. Round up to 1 decimal.

(Hint: check the cheat sheet element ‘Maths Functions’.)

Exercise

1. Write a line of code that produces the following vector:

(Note: there are multiple possible answers!)

```
> c(rep(4,3),seq(10,20,by=5),seq(10,4,by=-3))  
[1] 4 4 4 10 15 20 10 7 4
```

2. Divide the vector by 3. Round up to 1 decimal.

(Hint: check the cheat sheet element ‘Maths Functions’.)

```
> v <- c(rep(4,3),seq(10,20,by=5),seq(10,4,by=-3))  
> round(v/3,1)  
[1] 1.3 1.3 1.3 3.3 5.0 6.7 3.3 2.3 1.3
```

Another data type: logical

```
> T  
[1] TRUE  
> FALSE  
[1] FALSE  
> 1==1  
[1] TRUE  
> 3>=5  
[1] FALSE  
> 2!=4  
[1] TRUE
```

== is equal to
!= is not
>= larger than or equal to
< smaller than

Vectors and factors

```
> fc <- c("a", "b", "a", "c")
> fc
[1] "a" "b" "a" "c"
> fc <- factor(fc)
> fc
[1] a b a c
Levels: a b c
> fn <- 2:5
> fn
[1] 2 3 4 5
> fn <- factor(fn)
> fn
[1] 2 3 4 5
Levels: 2 3 4 5
> as.numeric(fn)
[1] 1 2 3 4
```

Note that vectors may consist of characters!

A factor is defined by its **levels**. Turning a factor into numeric specifies the order of the levels, but loses their content.

Exercise

1. Generate a vector that contains numeric, logical, and character elements.
2. Turn it into a factor.
3. See what happens to the content when you convert this factor into numeric, logical, and character vectors.

(Hint: check the cheat sheet element ‘Types’ for the functions.)

Exercise

1. Generate a vector that contains numeric, logical, and character elements.

```
> ft <- c(T,T,2,4,"apple","orange")
```

2. Turn it into a factor.

```
> ft <- factor(ft)
```

3. See what happens to the content when you convert this factor into numeric, logical, and character vectors.

(*Hint: check the cheat sheet element ‘Types’ for the functions.*)

```
> as.numeric(ft)
```

```
[1] 5 5 1 2 3 4
```

```
> as.logical(ft)
```

```
[1] TRUE TRUE NA NA NA NA
```

```
> as.character(ft)
```

```
[1] "TRUE" "TRUE" "2" "4" "apple" "orange"
```

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

```
> ft <- c(T,T,2,4,"apple","orange")  
> ft <- as.logical(ft)  
> ft  
[1] TRUE TRUE     NA     NA     NA     NA  
> factor(ft)  
[1] TRUE TRUE <NA> <NA> <NA> <NA>  
Levels: TRUE
```

NA = Not Available

Name	Age	Pet
Ann	33	Cat
Bob	25	None
Chloe	21	Iguana
Dan	45	NA

In other words:

We **know** that Bob has **no pets**.

We **do not know** if Dan has pets.

Exercise: predict the answer

```
> NA == NA
```

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

Exercise: predict the answer

```
> NA == NA
```

```
[1] NA
```

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

NA != NULL

```
> is.na(NA)
```

```
[1] TRUE
```

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

NA Information is **Not Available**

NULL Information **does not exist**

"None" Data entry specifying **"None"**

Exercise: predict the answer

```
> 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'

Selecting vector elements by position

```
> m <- 8:4  
> m  
[1] 8 7 6 5 4  
> m[2]  
[1] 7  
> m[1:3]  
[1] 8 7 6  
> m[-(2:4)]  
[1] 8 4
```

Selecting vector elements by value

```
> m <- 8:4  
  
> m  
[1] 8 7 6 5 4  
  
> m[m>5]  
[1] 8 7 6  
  
> m>5  
[1] TRUE TRUE TRUE FALSE FALSE  
  
> n <- 5:10  
  
> m[m %in% n]  
[1] 8 7 6 5  
  
> m[NA]  
[1] NA NA NA NA NA
```

m	m>5	result
8	TRUE	8
7	TRUE	7
6	TRUE	6
5	FALSE	
4	FALSE	

Which bracket does what?

- [] **Indexing** vectors, lists, dataframes...
 - () Passing **arguments** to functions
 - { } **Defining content** of loops, functions, etc.
- ```
> myvector[length(myvector)]
```

# Vectors, lists, and data frames

```
> v1 <- 4:6
> v2 <- rep(3,3)
> v <- c(v1,v2)
> v
[1] 4 5 6 3 3 3
> mylist <- list(v1,v2)
> mylist
[[1]]
[1] 4 5 6

[[2]]
[1] 3 3 3
> df <- data.frame(v1,v2)
> df

 v1 v2
1 4 3
2 5 3
3 6 3
```

|            | how many dimensions? | function     |
|------------|----------------------|--------------|
| vector     | 1                    | c()          |
| list       | any number           | list()       |
| data frame | 2                    | data.frame() |

# Indexing lists

---

```
> v1 <- 4:6
> v2 <- rep(3,3)
> myList <- list(v1,v2)
> myList[1]
[[1]]
[1] 4 5 6
> myList[[1]]
[1] 4 5 6
```

# Exercise

---

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

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

# Exercise

---

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

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

[ [ 1 ] ]

NULL

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

[ 1 ] 5

# Indexing a data frame

---

```
> name <- c("Ann", "Bob", "Chloe", "Dan")
> age <- c(33, 25, 21, 45)
> df <- data.frame(name, age)
> df
```

|   | name  | age |
|---|-------|-----|
| 1 | Ann   | 33  |
| 2 | Bob   | 25  |
| 3 | Chloe | 21  |
| 4 | Dan   | 45  |

# Indexing a data frame

```
> df$age
[1] 33 25 21 45

> df[,2]
[1] 33 25 21 45

> df[2,]
 name age
2 Bob 25

> df[2,2]
[1] 25
```

## Matrix subsetting

```
df[, 2]
```



```
df[2,]
```



The diagram shows a 2x2 matrix with four colored cells: top-left is blue, top-right is blue, bottom-left is grey, and bottom-right is yellow. Above the matrix, the word "row" is positioned above the first column with a yellow curved arrow pointing down to the first row. The word "column" is positioned to the right of the second column with a yellow curved arrow pointing down to the second row. Below the matrix, the text "df[2, 2]" is displayed, indicating the element at the second row and second column.



# Exercise

---

Return the names of everyone in your data frame under 30.

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

# Exercise

---

Return the names of everyone in your data frame under 30.

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

```
> df[df$age<30, "name"]
```

OR

```
> df[df$age<30, 1]
```

```
[1] Bob Chloe
```

```
Levels: Ann Bob Chloe Dan
```

Question: what is the data type of your result?

# 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()
> seq()
> round()
> sort()
> table()
...
```

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

```
> ?table()
> help.search("standard deviation")
```

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

# Take a break!

---



# Ready?

---

- Writing your first script
- Programming with loops and functions
- Handling and processing a dataset

move the samples from the data  
\* {r}  
select the samples to keep  
keepsamples <- row.names(pheno)  
apply sample selection to counts  
counts.sub <- counts.sub[,keepsamples]  
pheno.sub <- pheno[keepsamples]  
mt.assay <- mt.assay[,keepsamples]  
rld.sub <- rld[,keepsamples]

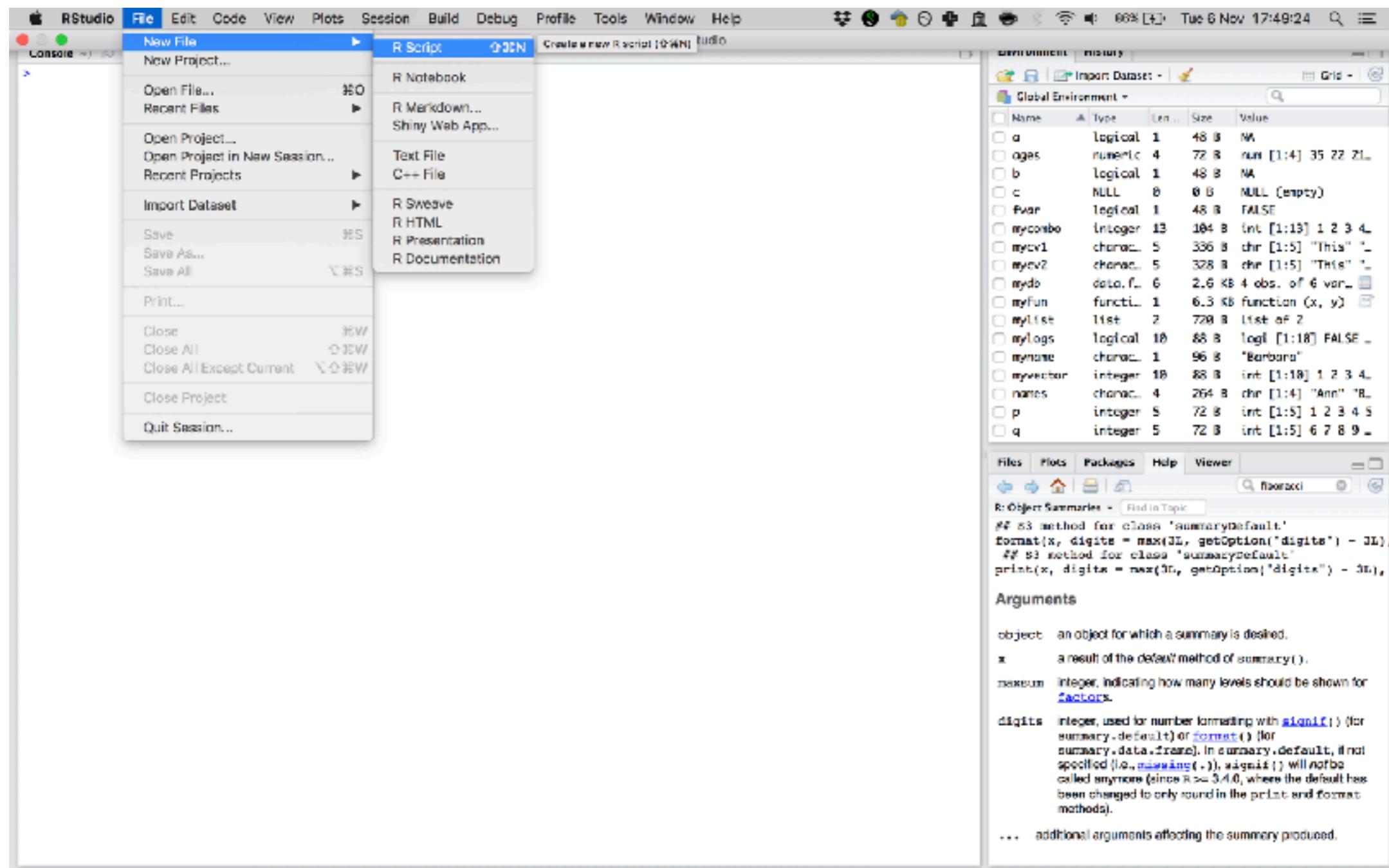
# Console and Scripts

---

|           |                  |                                         |
|-----------|------------------|-----------------------------------------|
| Console   | Code execution   | -                                       |
| Script    | Code             | Extension: .R<br>(example_script.R)     |
| Rmarkdown | Code + Narrative | Extension: .Rmd<br>(example_report.Rmd) |

# Ready to script?

RStudio > File > New File > R Script



# Console vs script on our slides

---

writing in the console

```
> x <- 1
```

```
> x
```

```
[1] 1
```

console output

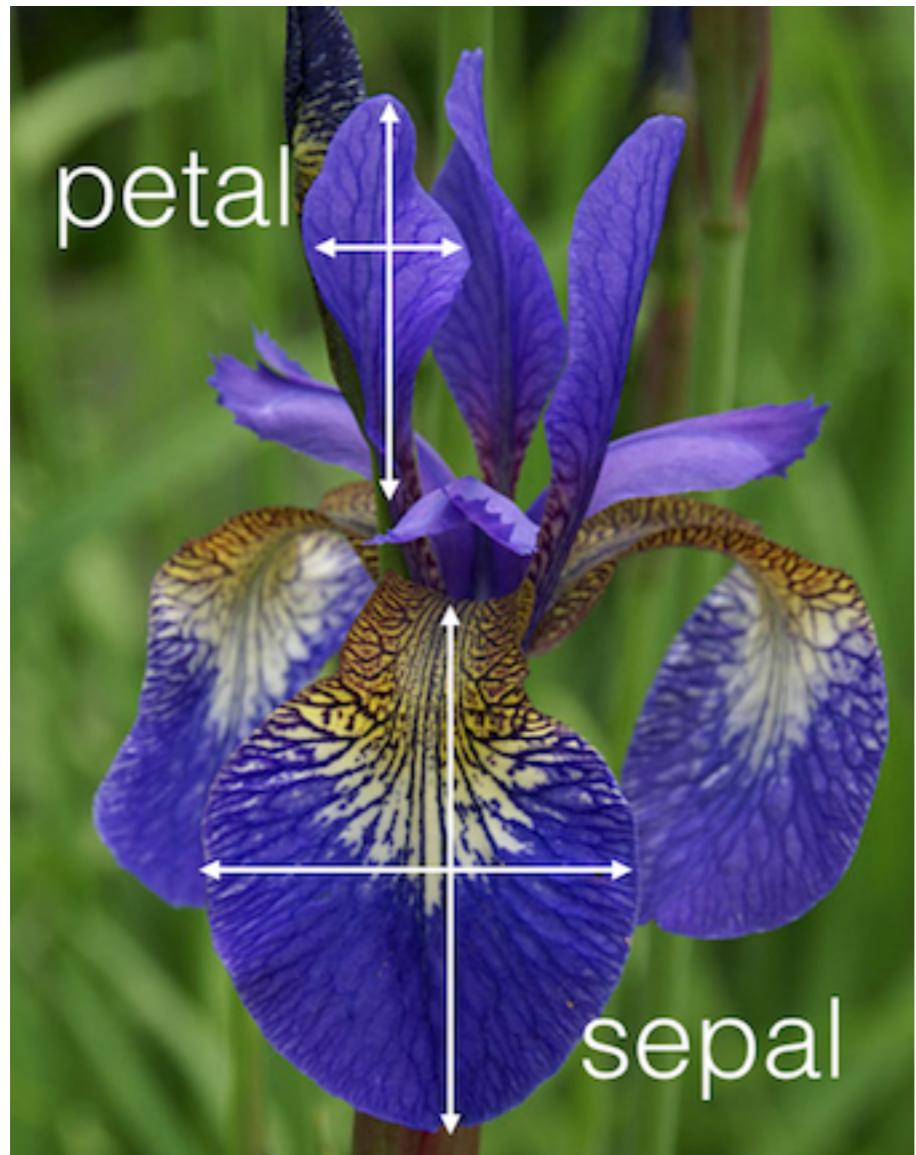
# indicates comment  
computer stops reading  
(but you should not!)

```
Assigning the value 1 to x
x <- 1
```

# 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   |               |

# Programming: for loop

---

```
Iterate over 10 values
for(i in 1:10){
 print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

# Calculating averages: preparation

---

```
what measurements exist in the iris db?
mmt <- colnames(iris)

> mmt
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"

remove "Species" from the list of names
mmt <- mmt[mmt!="Species"]

make a df with a single column: measurements
iris_res <- data.frame(mmt)

initiate a vector for averages
avgs <- NULL
```

# Calculating averages in a for loop

---

```
iterate over each measurement with a for-loop
for(i in mmt){

}
```

# Calculating averages in a for loop

---

```
iterate over each measurement with a for-loop
for(i in mmt){
 # select the appropriate column

 # calculate the average for this column

 # put the average in the averages vector

}
```

# Calculating averages in a for loop

---

```
iterate over each measurement with a for-loop
for(i in mmt){
 # select the appropriate column
 col_i <- iris[,i]
 # calculate the average for this column

 # put the average in the averages vector

}
```

# Calculating averages in a for loop

---

```
iterate over each measurement with a for-loop
for(i in mmt){
 # select the appropriate column
 col_i <- iris[,i]
 # calculate the average for this column
 avg_i <- mean(col_i)
 # put the average in the averages vector

}
```

# Calculating averages in a for loop

---

```
iterate over each measurement with a for-loop
for(i in mmt){
 # select the appropriate column
 col_i <- iris[,i]
 # calculate the average for this column
 avg_i <- mean(col_i)
 # put the average in the averages vector
 avgs <- c(avgs,avg_i)
}

> avgs
[1] 5.843333 3.057333 3.758000 1.199333
```

# Calculating averages in a for loop

---

```
add the averages as a column to the data frame
iris_res$average <- avgs

> iris_res
 mmt average
1 Sepal.Length 5.843333
2 Sepal.Width 3.057333
3 Petal.Length 3.758000
4 Petal.Width 1.199333
```

# Programming: functions

---

```
myFun <- function(x){
 return(x)
}
```

```
> myFun(1)
[1] 1
```

# Programming: functions

---

```
myFun <- function(x){
 x <- x*20
 return(x)
}
```

```
> myFun(1)
[1] 20
```

# Programming: functions

---

```
myFun <- function(x,y){
 z <- x*y
 return(z)
}
```

```
> myFun(2,4)
[1] 8
```

# Exercise: combine functions and for loops

---

- Open the script **programming\_exercise.R**
- You will see the for-loop (including comments) as we have just written
- You will then see the start of a function:

```
average_df <- function(df){
 mmt <- names(df)
 mmt <- mmt[mmt != "Species"]
 avgs <- NULL
 ### ENTER YOUR OWN CODE ###
 return(avgs)
}
```

- **Exercise:** place the for-loop in the function, and edit as necessary, so that the function now generates a list of averages.
- **TIP:** within the function, `iris` is renamed `df`!

# Exercise: combine functions and for loops

---

```
average_df <- function(df){
 # determine measurement names and remove species
 mmt <- names(df)
 mmt <- mmt[mmt!="Species"]
 # initialize the vector
 avgs <- NULL
 # iterate over each measurement with a for-loop
 for(i in mmt){
 # select the appropriate column
 col_i <- df[,i]
 # calculate the average
 avg_i <- mean(col_i)
 # put the average in the averages vector
 avgs <- c(avgs,avg_i)
 }
 # return the averages as a column
 return(avgs)
}
apply the function: add the averages as a column to the data frame
iris_res$average <- average_df(iris)
```

# Programming: if statement

---

```
if(4>5){
 print("math has changed!")
} else{
 print("all is well.")
}
```

# Programming: if statement

---

```
if(4>5){
 print("math has changed!")
} else if(5>6){
 print("this is not good either!")
} else{
 print("all is well.")
}
```

# Exercise: add an if-statement to your function

---

- We want to build on our `average_df` function so it can do more calculations!
- To do this, we add a second argument, and include an if statement.
- First, copy-paste your function, and give it a new name. Consider changing some variable names, too (and make sure to change them all!).

```
stats_df <- function(df){
 mmt <- names(df)
 mmt <- mmt[mmt!="Species"]
 res <- NULL
 for(i in mmt){
 col_i <- df[,i]
 res_i <- mean(col_i)
 res <- c(res,res_i)
 }
 return(res)
}
```

# Exercise: add an if-statement to your function

---

- Our function can calculate averages. We might want to add standard deviations (`sd()`), maximums (`max()`), and minimums (`min()`).
- Which line in the function do we need to expand on, to allow this?
- What additional information do we need?

```
stats_df <- function(df){
 mmt <- names(df)
 mmt <- mmt[mmt!="Species"]
 res <- NULL
 for(i in mmt){
 col_i <- df[,i]
 res_i <- mean(col_i)
 res <- c(res,res_i)
 }
 return(res)
}
```

# Exercise: add an if-statement to your function

---

```
stats_df <- function(df,fnx){
 mmt <- names(df)
 mmt <- mmt[mmt!="Species"]
 res <- NULL
 for(i in mmt){
 col_i <- df[,i]
 # perform different functions based on the input
 if(fnx=="mean"){
 res_i <- mean(col_i)
 } else{
 stop("no valid input received")
 }
 res <- c(res,res_i)
 }
 return(res)
}
```

# Exercise: add an if-statement to your function

---

1. Add the if-statement below to your function! (Do you understand how it works?)

```
if(fnx=="mean"){
 res_i <- mean(col_i)
} else{
 stop("no valid input received")
}
```

2. Elaborate on the statement using `if else`, adding alternative functions. Consider standard deviation (`sd()`), maximum (`max()`), and minimum (`min()`).
3. Finally, call your new function to add columns to your `iris_res` dataframe!

# Exercise: add an if-statement to your function

---

```
stats_df <- function(df,fnx){
 mmt <- names(df)
 mmt <- mmt[mmt!="Species"]
 res <- NULL
 for(i in mmt){
 col_i <- df[,i]
 if(fnx=="mean"){
 res_i <- mean(col_i)
 } else if(fnx=="sd"){
 res_i <- sd(col_i)
 } else{
 stop("no valid input received")
 }
 res <- c(res,res_i)
 }
 return(res)
}

iris_res$standarddev <- stats_df(fnx="sd",iris)
```

# Your result: a summary data frame

---

```
iris_res$standarddev <- stats_df(fnx="sd",iris)
iris_res$largest <- stats_df(fnx="max",iris)
iris_res$smallest <- stats_df(fnx="min",iris)
```

```
> iris_res
```

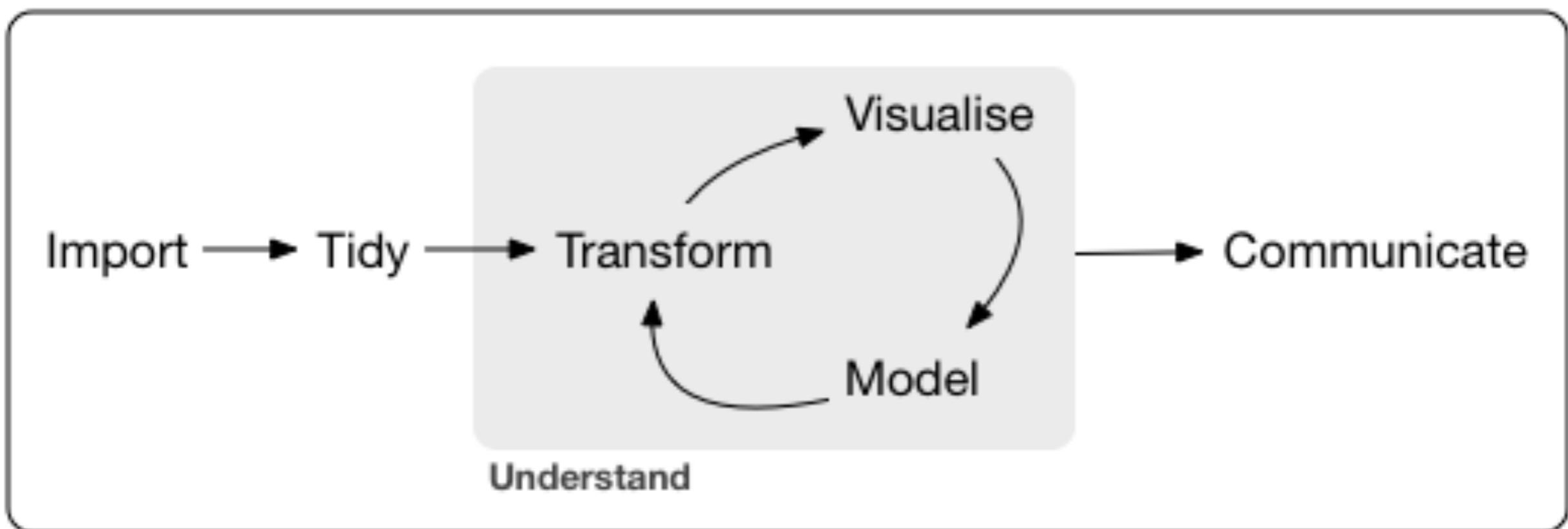
|                | mmt      | average   | standarddev | largest | smallest |
|----------------|----------|-----------|-------------|---------|----------|
| 1 Sepal.Length | 5.843333 | 0.8280661 | 7.9         | 4.3     |          |
| 2 Sepal.Width  | 3.057333 | 0.4358663 | 4.4         | 2.0     |          |
| 3 Petal.Length | 3.758000 | 1.7652982 | 6.9         | 1.0     |          |
| 4 Petal.Width  | 1.199333 | 0.7622377 | 2.5         | 0.1     |          |

```
> summary(iris)
```

| Sepal.Length  | Sepal.Width   | Petal.Length  | Petal.Width   | Species    |
|---------------|---------------|---------------|---------------|------------|
| Min. :4.300   | Min. :2.000   | Min. :1.000   | Min. :0.100   | setosa     |
| 1st Qu.:5.100 | 1st Qu.:2.800 | 1st Qu.:1.600 | 1st Qu.:0.300 | versicolor |
| Median :5.800 | Median :3.000 | Median :4.350 | Median :1.300 | virginica  |
| 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   |            |

# 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.



Program

*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

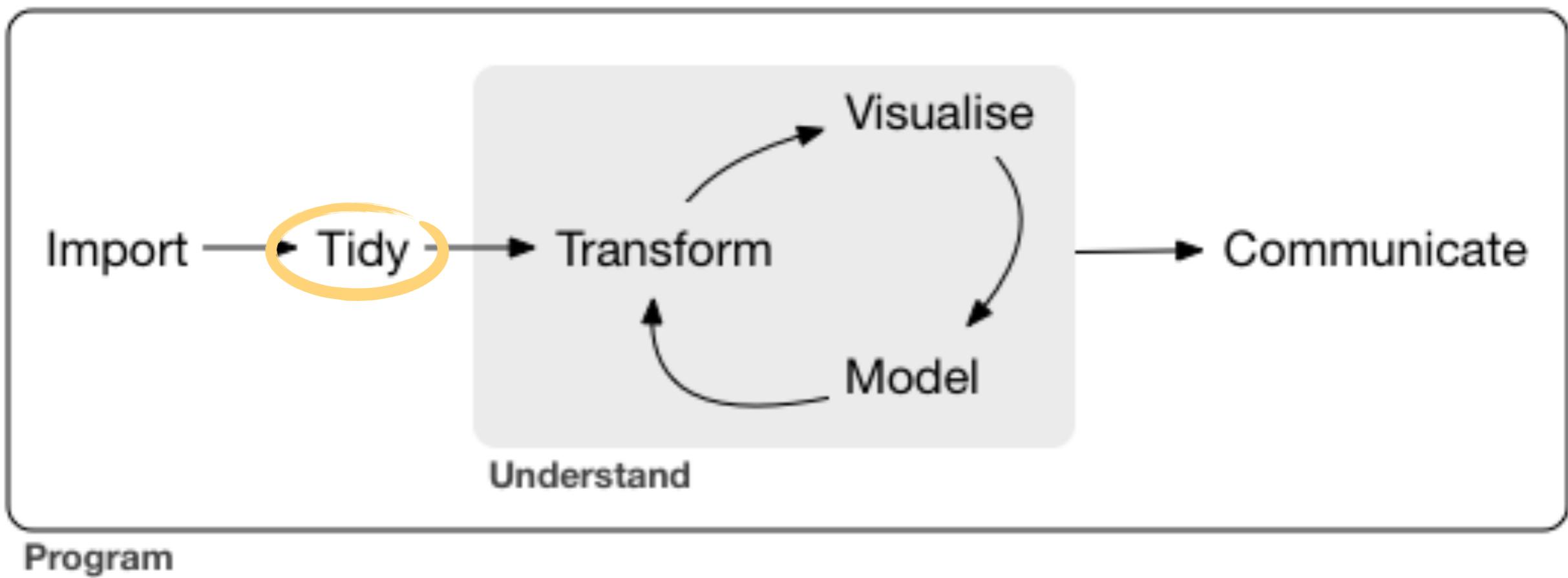
```
Date: 7 November 2018
Author: Barbara Vreede
This script was written as part of the R course
"Introduction to R & Data", at Utrecht University
```

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

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

# Data science workflow

---



Tidy data ensures that further processing can be done efficiently, and reproducibly.

Tidy data is easy to manipulate, model, and visualize.

# Tidy data

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

values in column names

| Patient | BP     | Med_A | BP_after_A | Med_B | BP_after_B |
|---------|--------|-------|------------|-------|------------|
| 122030  | 120_82 | 300   | 119_85     | NA    | NA         |
| 122021  | 131_91 | 200   | 132_85     | 85    | 125_90     |
| 124500  | 118_86 | 300   | 119_70     | NA    | NA         |
| 126098  | 99_67  |       |            | 100   |            |

multiple data points  
in a single cell

multiple  
observations

# Tidy data

**ID no longer unique per row**  
(but can still be used to collect  
all info on single patient)

| Patient | Sys | Dia | Treatment |
|---------|-----|-----|-----------|
| 122030  | 120 | 82  | None      |
| 122030  | 119 | 85  | A         |
| 122021  | 131 | 91  | None      |
| 122021  | 132 | 85  | A         |
| 122021  | 119 | 70  | B         |
| 124500  | 118 | 86  | None      |
| 124500  | 119 | 70  | A         |
| 126098  | 99  | 67  | None      |
| 126098  | 110 | 71  | B         |

**Time series (order) lost**  
(so make sure this is  
explicit)

# To check before lunch: please 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
✓ tidyrr 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()
```

Enjoy your lunch!

---



# Introduction to R & data

---

## Part II: Modern R with tidyverse

## Outline part II

---

- Introduction to **tidyverse**, **rmarkdown**, and **tibble**.
- Load and save data with **readr**
- [break]
- Data visualisation with **ggplot**
- [break]
- Data transformation with **tidyr** and **dplyr**

```
row.names(pheno[pheno$sampleID %in% keepsamples,], na.rm = TRUE)
density(glnorm[, sname], na.rm = TRUE)
move the samples from the data
 {r}
select the samples to keep
keepsamples <- row.names(pheno[pheno$sampleID %in% keepsamples,], na.rm = TRUE)
apply sample selection to counts
counts.sub <- counts.sub[, keepsamples]
pheno.sub <- pheno[keepsamples,]
assay <- nt.assay[, keepsamples]
rld.sub <- rld[, keepsamples]
```



Tidyverse

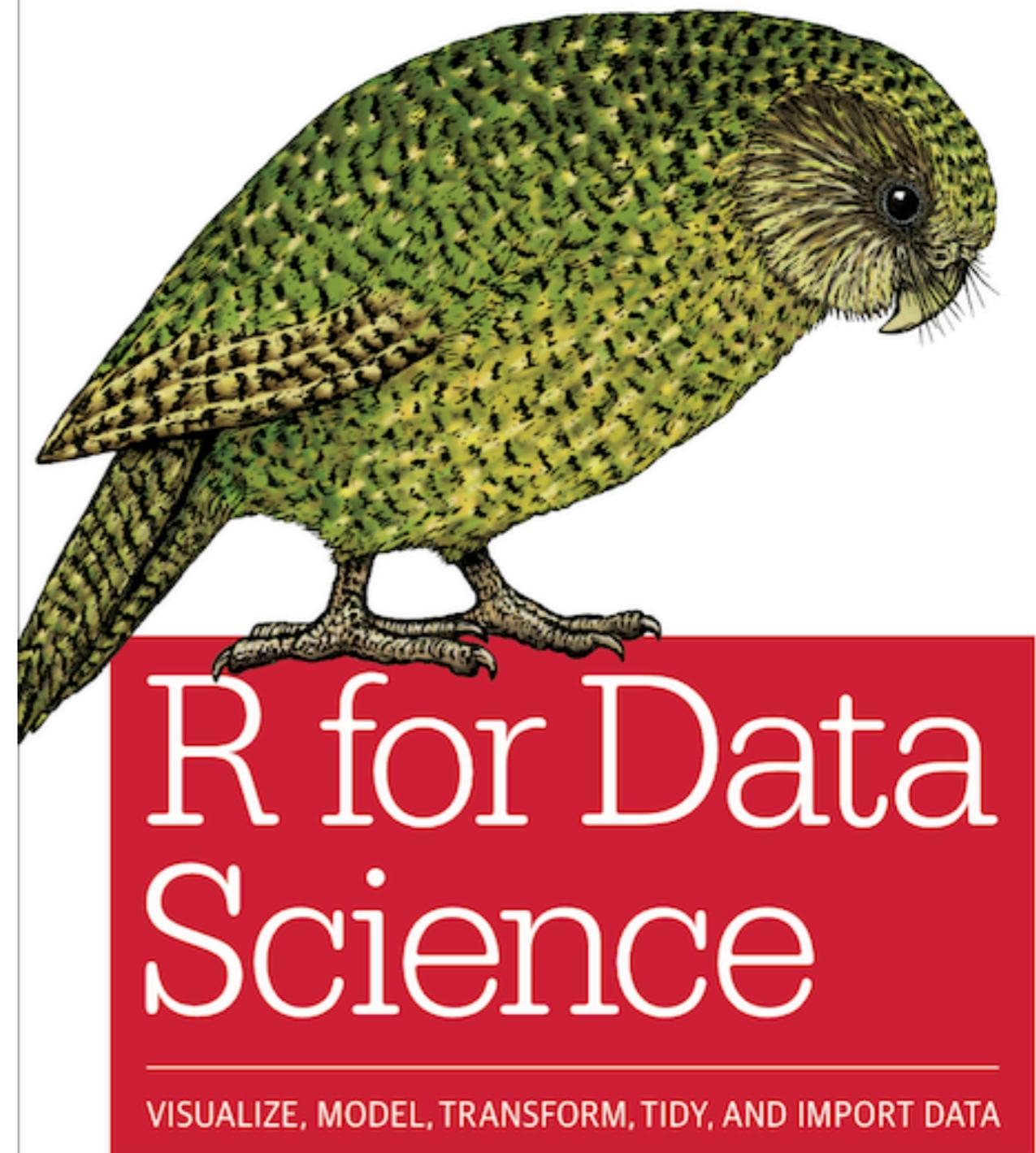
Modern R for data science

"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)**

# Learn tidyverse

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



Hadley Wickham &  
Garrett Grolemund

# Learn tidyverse

- R for Data Science (book)  
Freely available on:  
[r4ds.had.co.nz/](http://r4ds.had.co.nz/)
- ggplot2 (book)

Hadley Wickham

**ggplot2**

Elegant Graphics for Data Analysis



# Your afternoon roadmap

---

- A short explanation on screen (~15 minutes)
- Exercises in a document on your computer (~45 minutes)
  - 2-3 basic exercises
  - optional exercises, for further practice
  - reading exercises, for some extra insight into Tidyverse
- We will work with a new data set
- The exercises are presented in Rmarkdown format. We'll explain this later!

From code to  
document

---

Turn your analyses into high quality  
documents with Rmarkdown



# R Markdown

---

- **Combine code with narrative (R with markdown)**
- Turn your analyses into high quality documents, reports, presentations and dashboards.
- ... or your note book.
- Rmarkdown is different from a R script
- Rmarkdown files have file extension **.Rmd** (R scripts have extension **.R**)

# Console, Scripts and Rmarkdown

---

|           |                  |                                                |
|-----------|------------------|------------------------------------------------|
| Console   | Code execution   | -                                              |
| Script    | Code             | Extension: <b>.R</b><br>(example_script.R)     |
| Rmarkdown | Code + Narrative | Extension: <b>.Rmd</b><br>(example_report.Rmd) |

# Markdown

---

```
title
```

```
Some text about your project
```

```
Section
```

```
More text about your project
```

```
```{r}
max(iris$Petal.Length)
min(iris$Petal.Length)
```
```

# Rmarkdown: demo

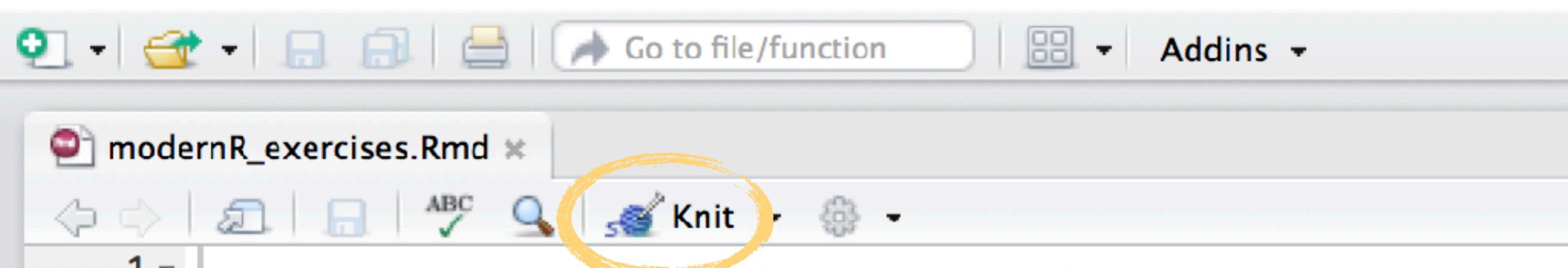
---



# Exercise: open the exercise Rmarkdown

1. RStudio > File > Open Project > introduction-to-R-and-data-github.Rproj
2. From the ‘files’ menu (bottom right), select modernR\_exercises.Rmd
3. Update the header information with your name!

# All done? Time to knit!



The screenshot shows the RStudio interface with the 'modernR\_exercises.Rmd' file open. The 'Knit' button in the toolbar is highlighted with a yellow circle. The code in the document includes YAML front matter and a note about the workshop.

```
1 ---
2 title: "Modern R with tidyverse"
3 author: "[Insert your name]"
4 output:
5 html_document:
6 toc: true
7 ---
8
9 *This document is part of the workshop **Introduction to R & Data
10
11 # Introduction
12
13 In this document, we explore Crane migration, through the GPS dat
data was kindly provided for this course by Sasha Pekarsky at the
```

# package: Tibble

---

Improved data frames



# 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
✓ tidyverse 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()
```

"Tibbles are data frames, but they tweak some older behaviours to make life a little easier."

– Hadley Wickham (creator of Tidyverse)

# 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`



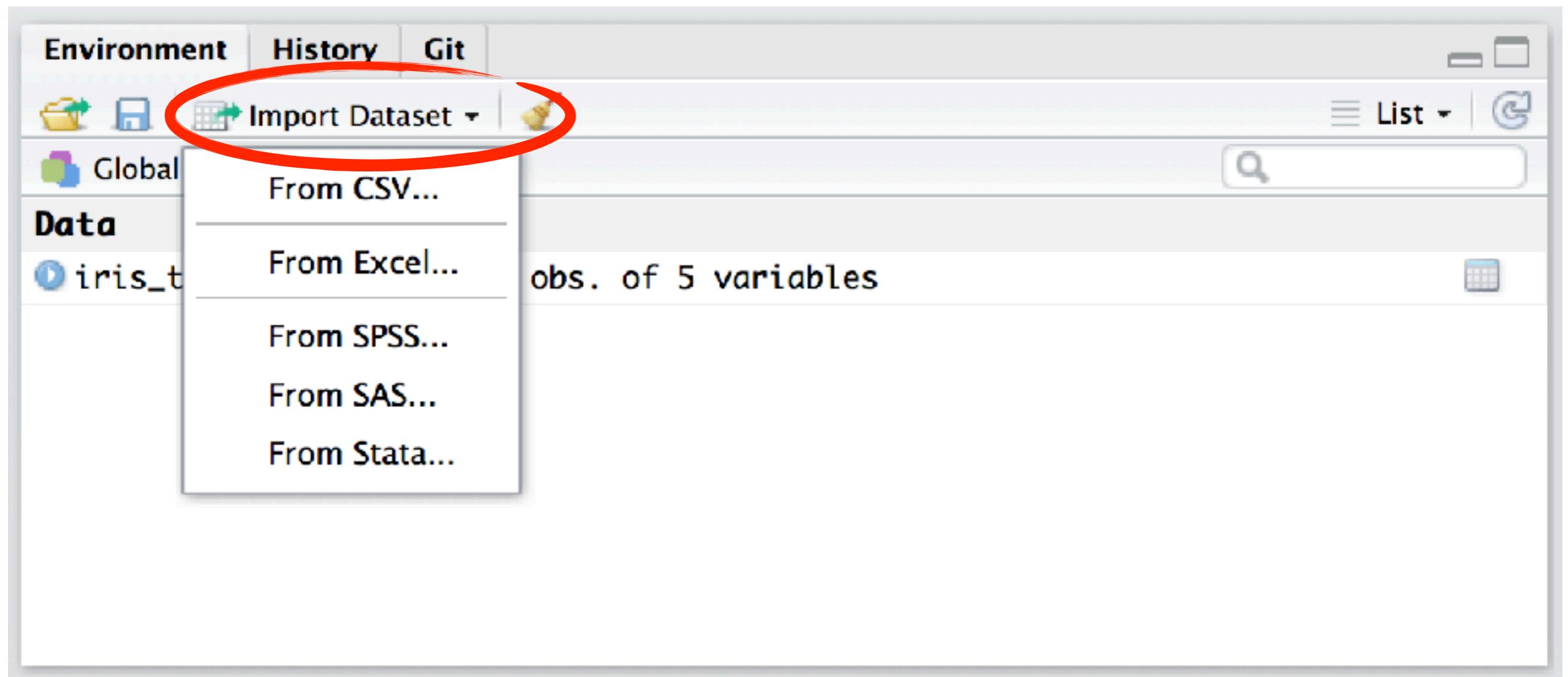
## Flat data files

---

- Most common format: Comma-separated values (CSV)
- Delimiter-separated values (also stored as CSV)
- Common delimiters:
  - comma ,
  - semicolon ;
  - tab \t

```
1 mpg;cyl;disp;hp;drat;wt;qsec;v
2 21;6;160;110;3.9;2.62;16.46;0;
3 21;6;160;110;3.9;2.875;17.02;0
4 22.8;4;108;93;3.85;2.32;18.61;
5 21.4;6;258;110;3.08;3.215;19.4
6 18.7;8;360;175;3.15;3.44;17.02
7 18.1;6;225;105;2.76;3.46;20.22
8 14.3;8;360;245;3.21;3.57;15.84
9 24.4;4;146.7;62;3.69;3.19;20;1
10 22.8;4;140.8;95;3.92;3.15;22.9
11 19.2;6;167.6;123;3.92;3.44;18.
12 17.8;6;167.6;123;3.92;3.44;18.
13 16.4;8;275.8;180;3.07;4.07;17.
14 17.3;8;275.8;180;3.07;3.73;17.
15 15.2;8;275.8;180;3.07;3.78;18;
16 10.4;8;472;205;2.93;5.25;17.98
17 10.4;8;460;215;3;5.424;17.82;0
18 14.7;8;440;230;3.23;5.345;17.4
19 32.4;4;78.7;66;4.08;2.2;19.47;
20 30.4;4;75.7;52;4.93;1.615;18.5
21 33.9;4;71.1;65;4.22;1.835;19.9
22 21.5;4;120.1;97;3.7;2.465;20.0
23 17.5;8;312;152;3.75;3.52;16.25
```

# 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

Data Preview:

| Sepal.Length<br>(double) ▾ | Sepal.Width<br>(double) ▾ | Petal.Length<br>(double) ▾ | Petal.Width<br>(double) ▾ | Species<br>(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  First Row as Names Delimiter: Comma ▾ Escape: None ▾  
Skip: 0  Trim Spaces Quotes: Default ▾ Comment: Default ▾  
 Open Data Viewer Locale: Configure... NA: Default ▾

Code Preview:

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

## 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()
)
```

# Exercise dataset

---

- GPS data from 39 tagged individual cranes during fall migration of 2017
- Courtesy of Sasha Pekarsky at the Hebrew University of Jerusalem, Israel
- In the ‘data’ folder of your course materials.



# Exercises

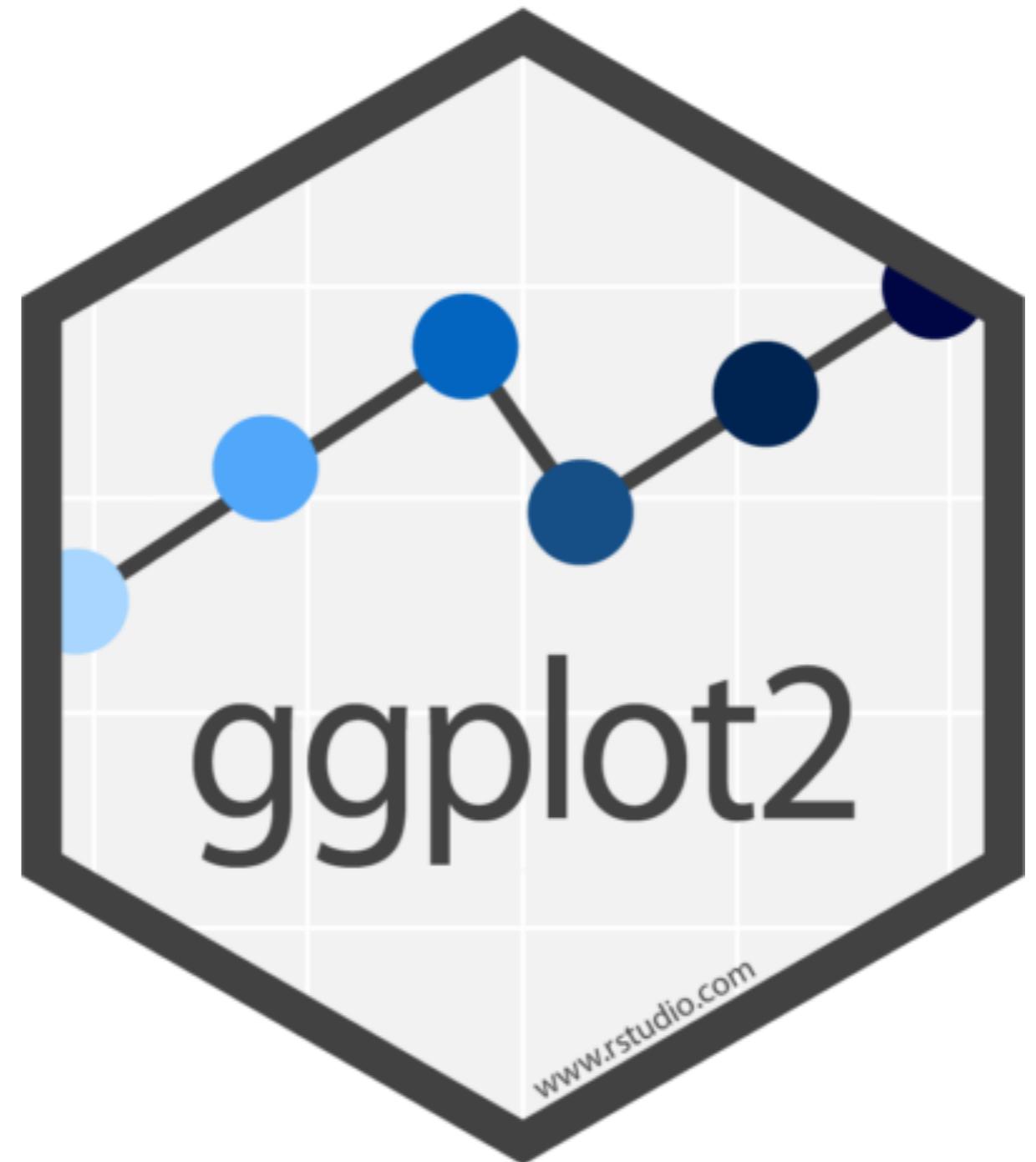
Please do basic exercises 1-I and 1-II.

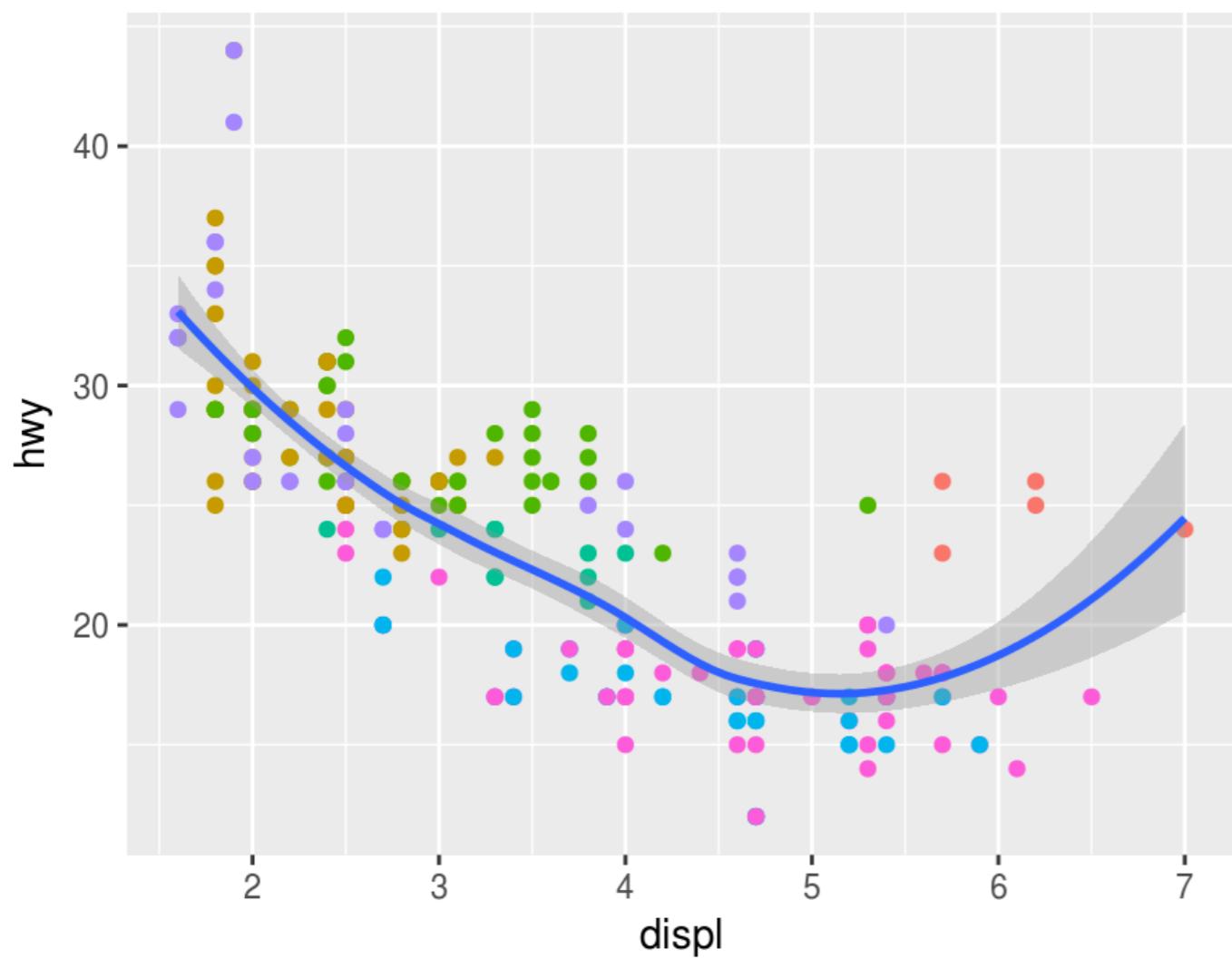
Time left? Opt for a reading exercise or optional exercise!

# Data visualisation

---

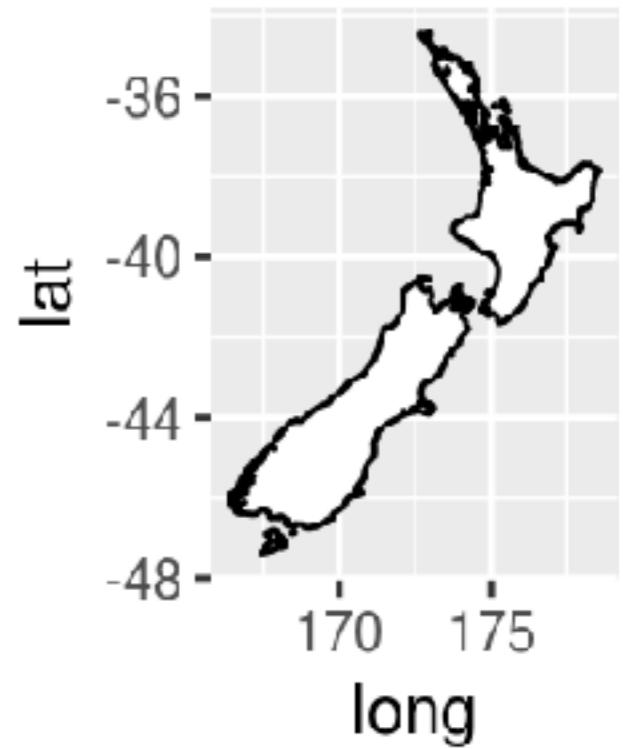
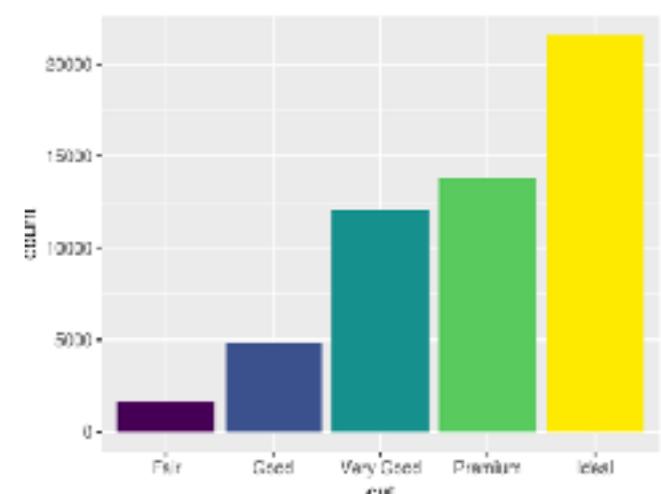
Create eye candy with ggplot2





class

- 2seater
- compact
- midsize
- minivan
- pickup
- subcompact
- suv



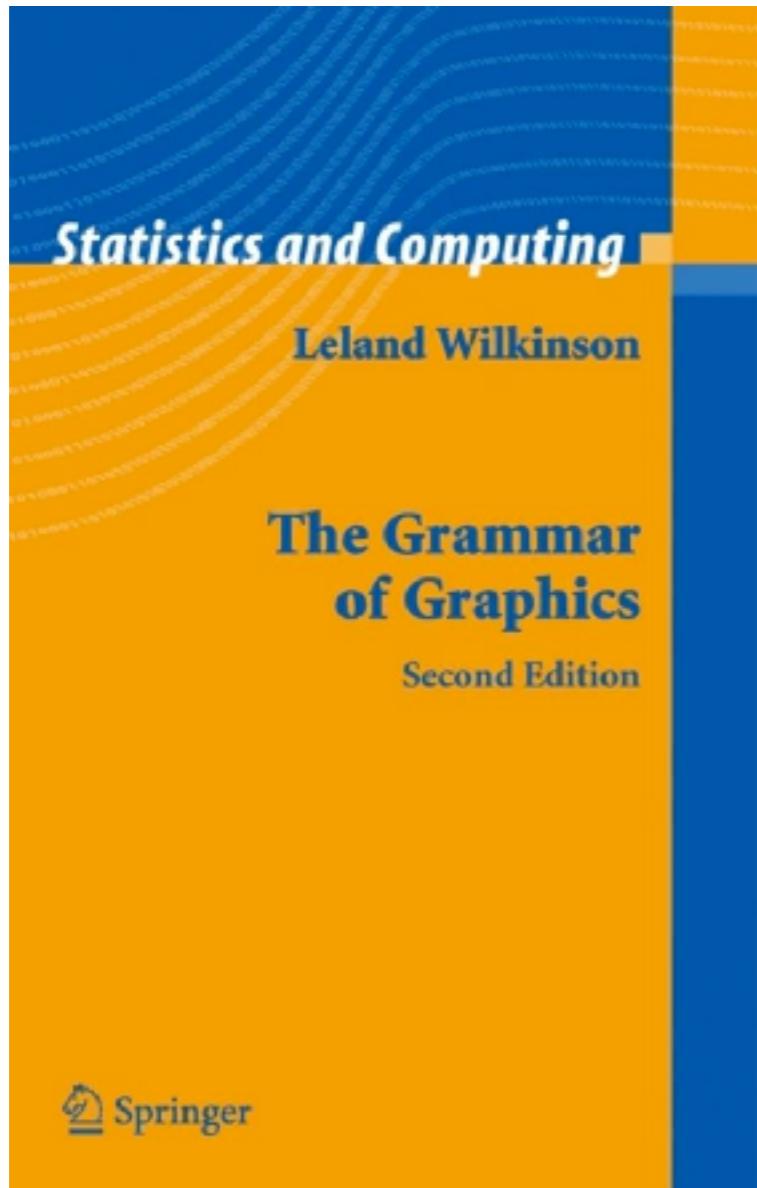
Examples of plots with R (and ggplot2)

# 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



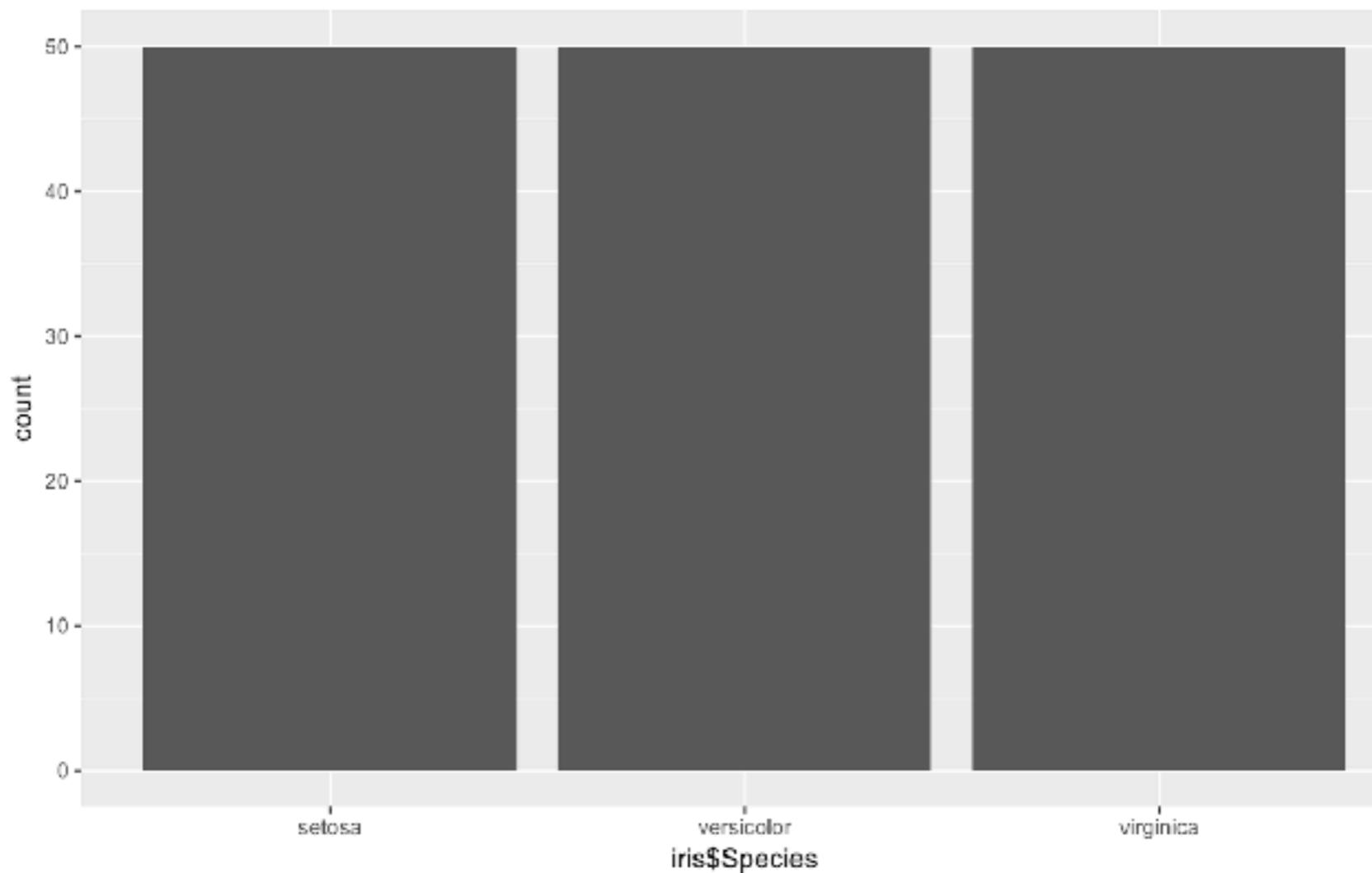
|            |                                           |
|------------|-------------------------------------------|
| Data       | The variables in a tibble or data.frame   |
| Aesthetics | x- and y-axis, colour, size, alpha, shape |
| Geometries | point, line, bar, histogram               |

# Quick plotting

# Quick plots

---

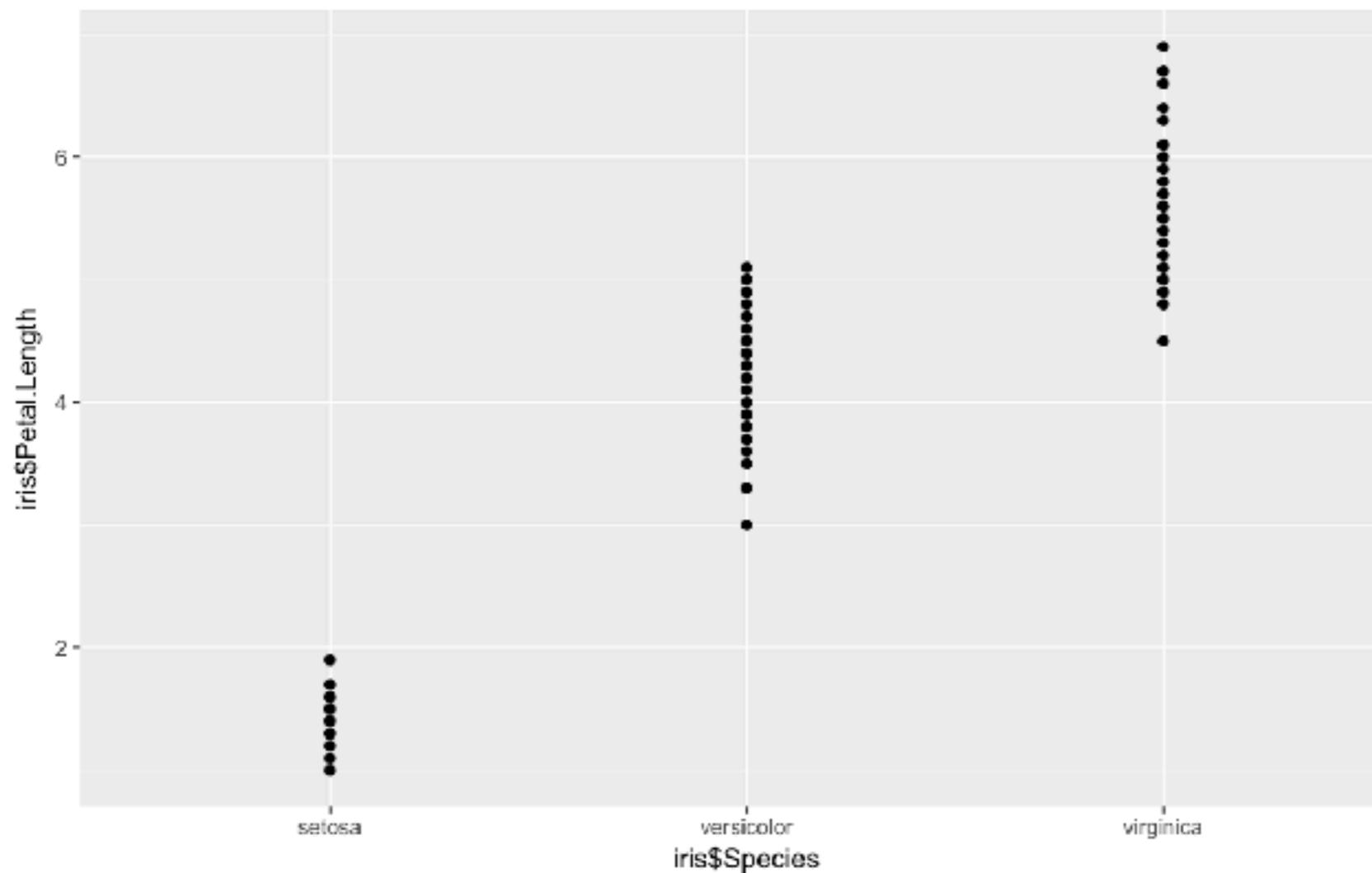
```
> qplot(Species, data = iris)
```



# Quick plots

---

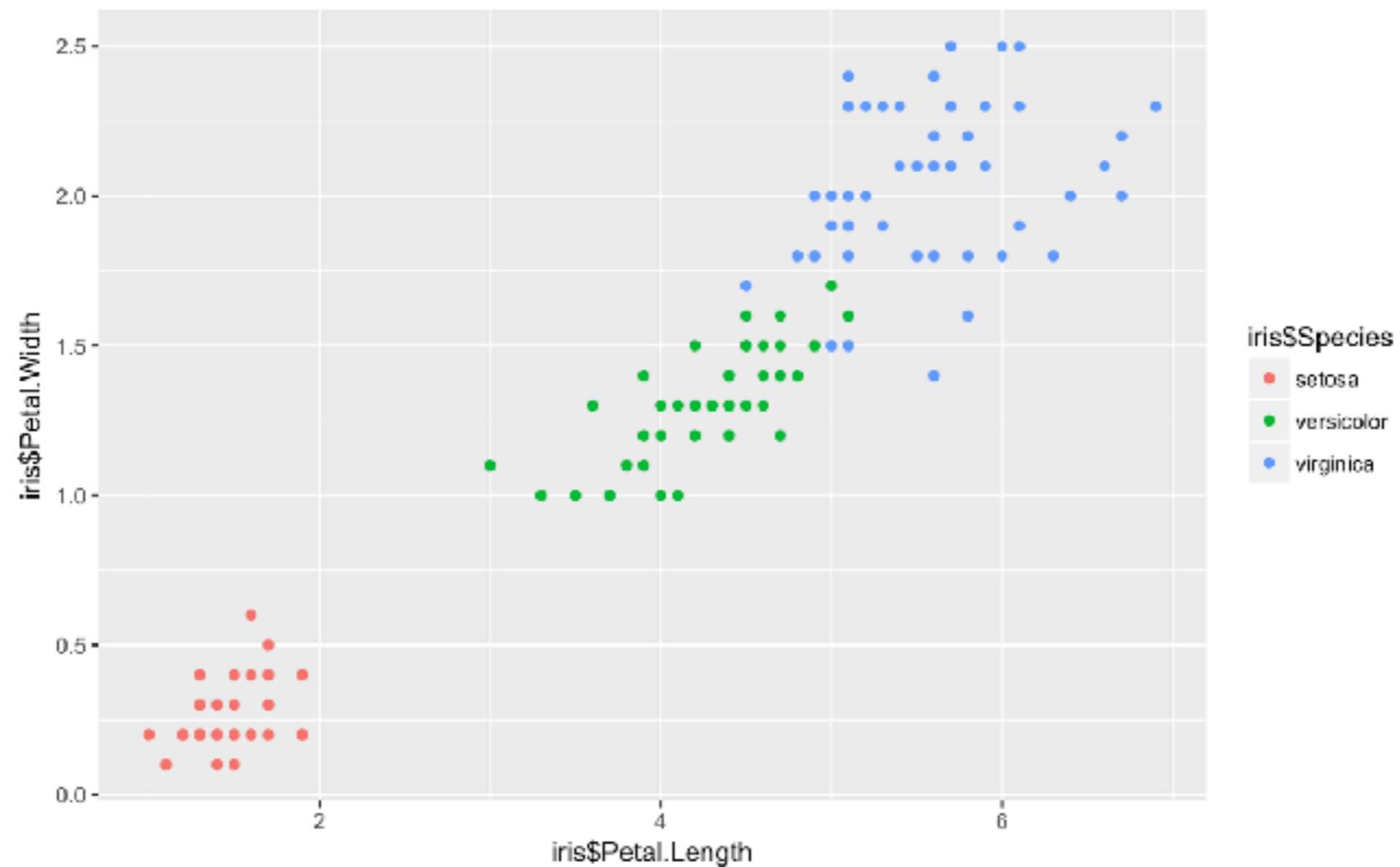
```
> qplot(Species, data = iris)
> qplot(Species, Petal.Length, data = iris)
```



# Quick plots

---

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

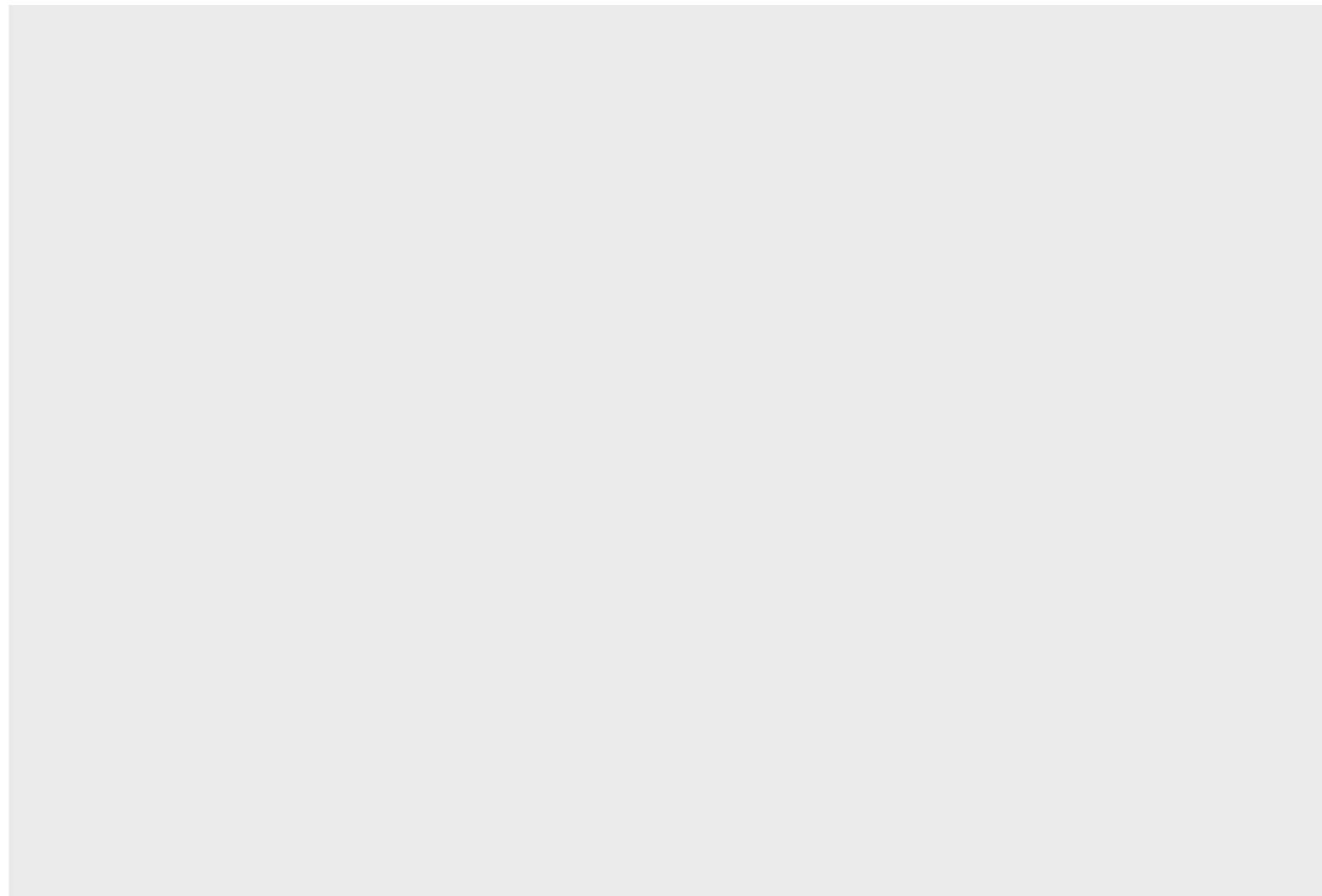


ggplot (grammar of graphics)

# ggplot: Data, aesthetics, geometrics

---

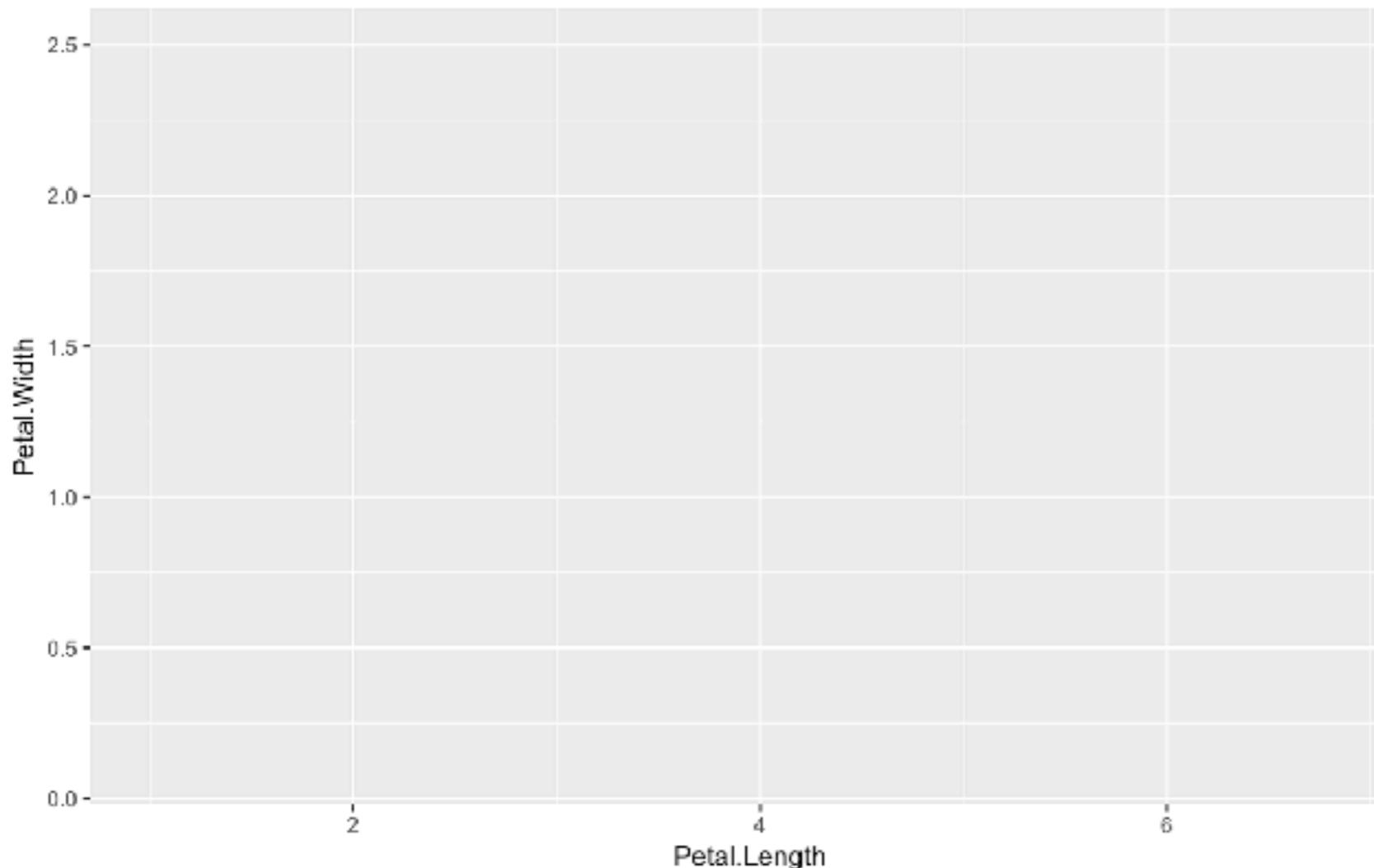
```
> ggplot(iris)
```



# ggplot: Data, aesthetics, geometrics

---

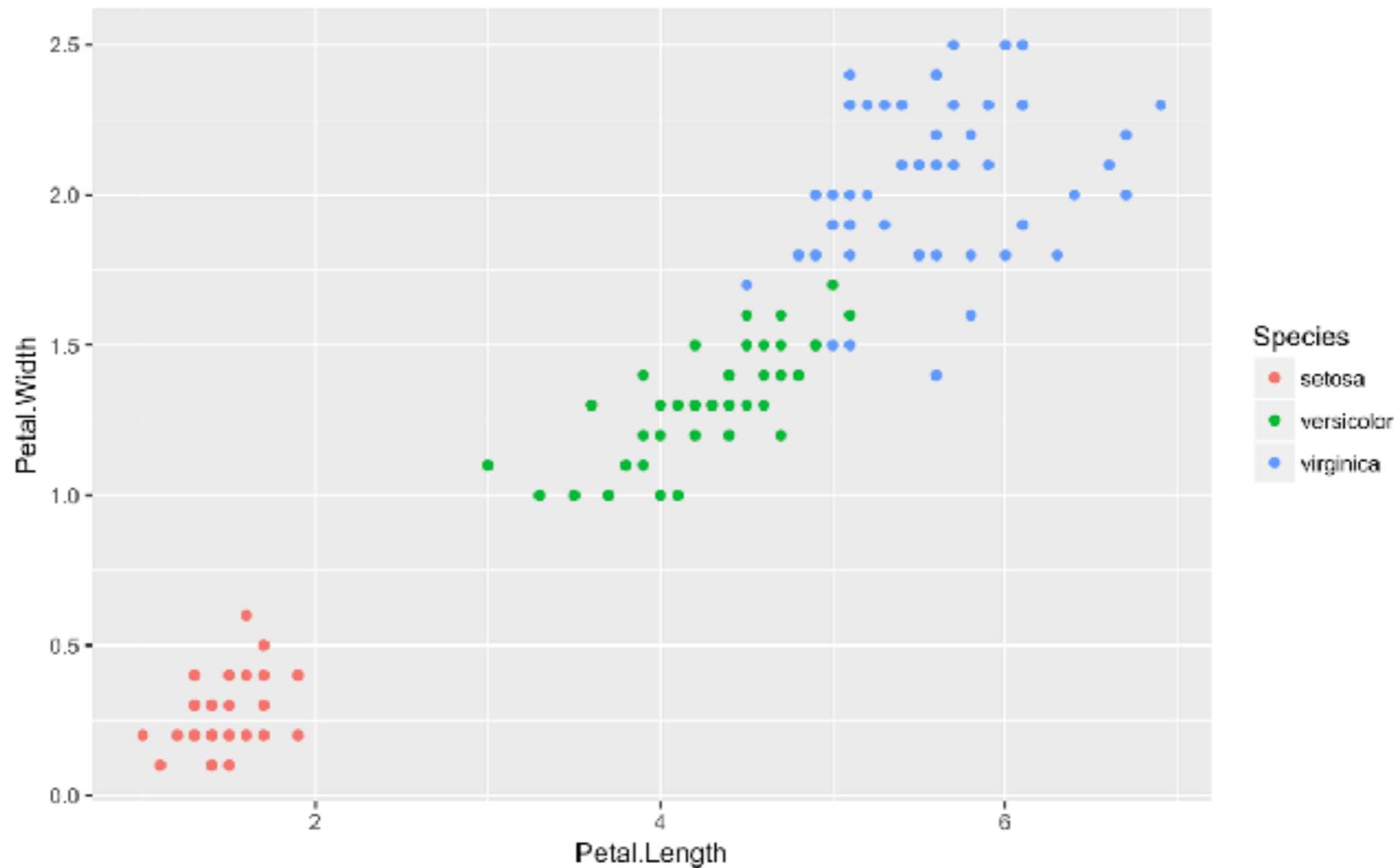
```
> ggplot(iris)
> ggplot(iris, aes(x = Petal.Length, y = Petal.Width, colour=Species))
```



# ggplot: Data, aesthetics, geometrics

---

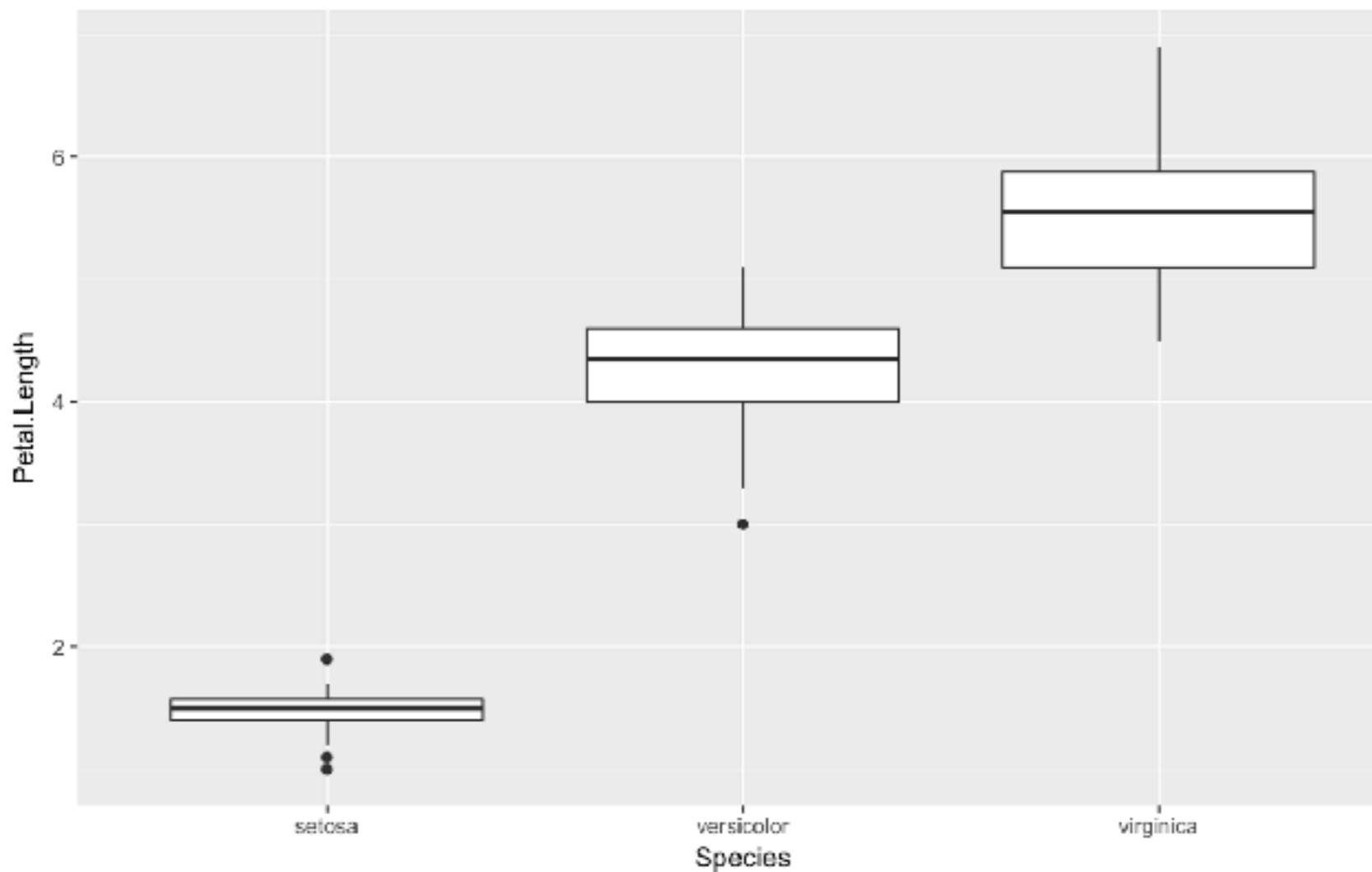
```
> ggplot(iris)
> ggplot(iris, aes(x = Petal.Length, y = Petal.Width, colour=Species))
> ggplot(iris, aes(x = Petal.Length, y = Petal.Width, colour=Species)) +
 geom_point()
```



# ggplot: Multiple geometric layers

---

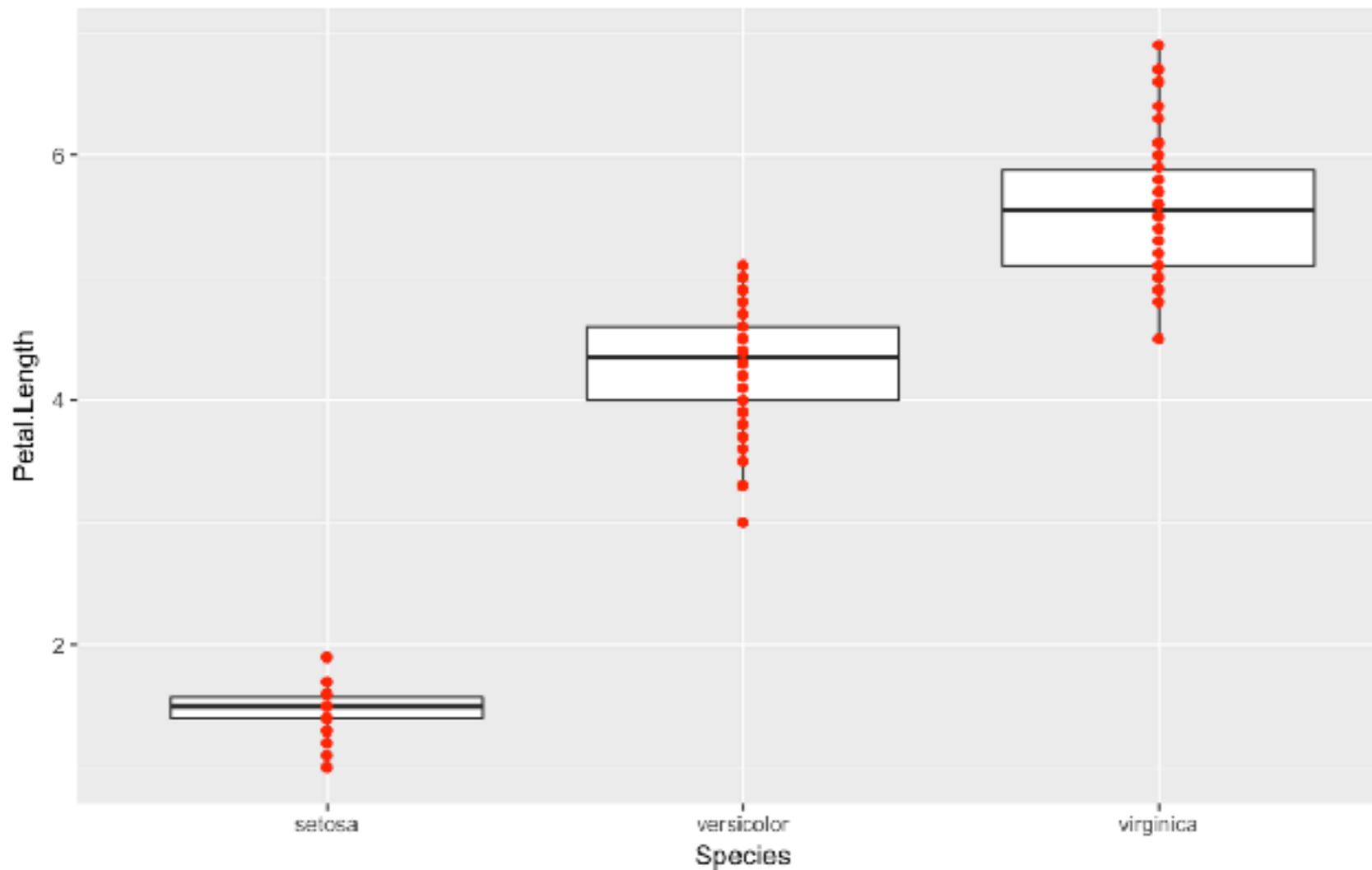
```
> ggplot(iris, aes(x = Species, y = Petal.Length)) +
 geom_boxplot()
```



# ggplot: Multiple geometric layers

---

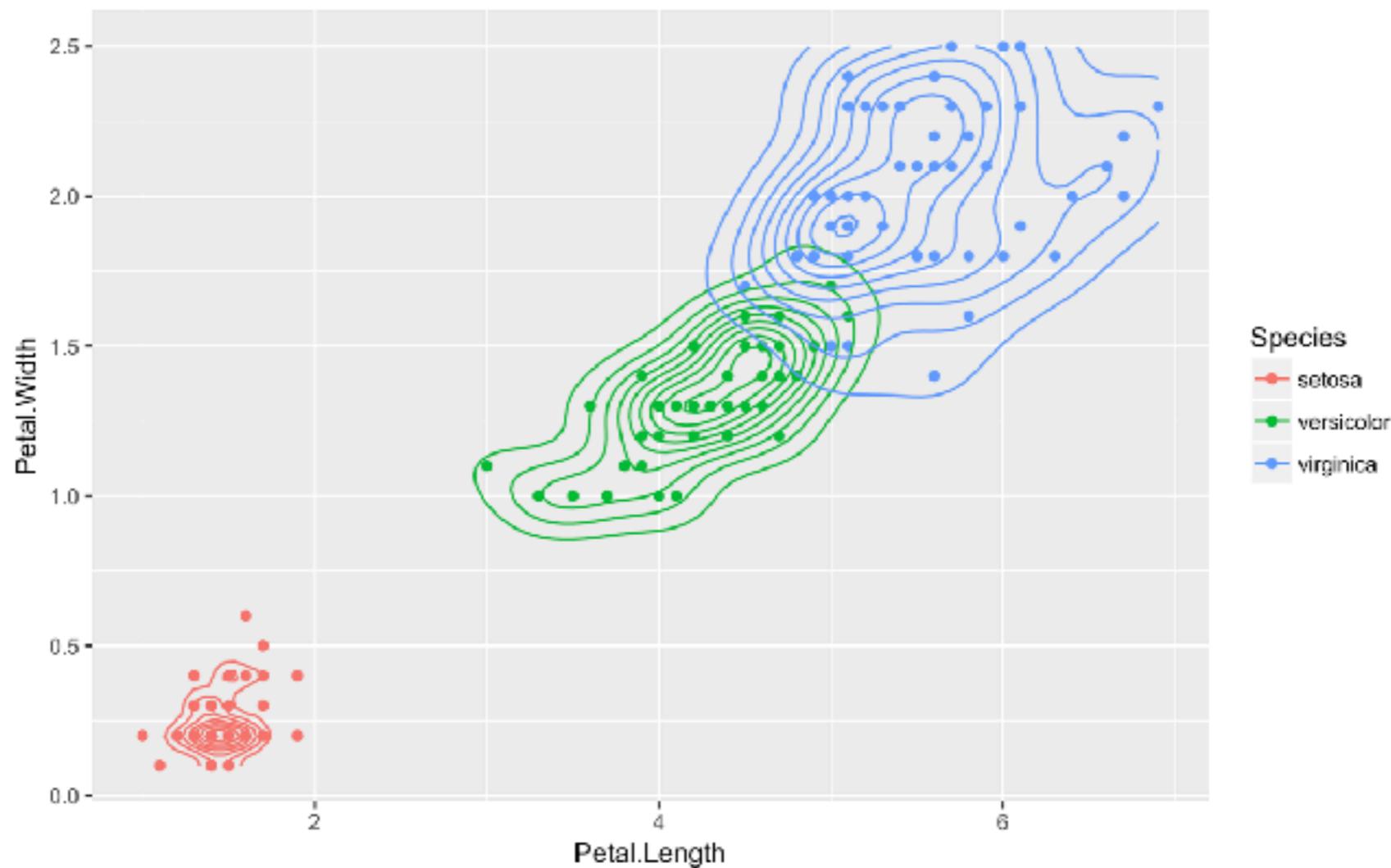
```
> ggplot(iris, aes(x = Species, y = Petal.Length)) +
 geom_boxplot() +
 geom_point(colour='red')
```



# ggplot: Statistical layers

---

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



# Exercises

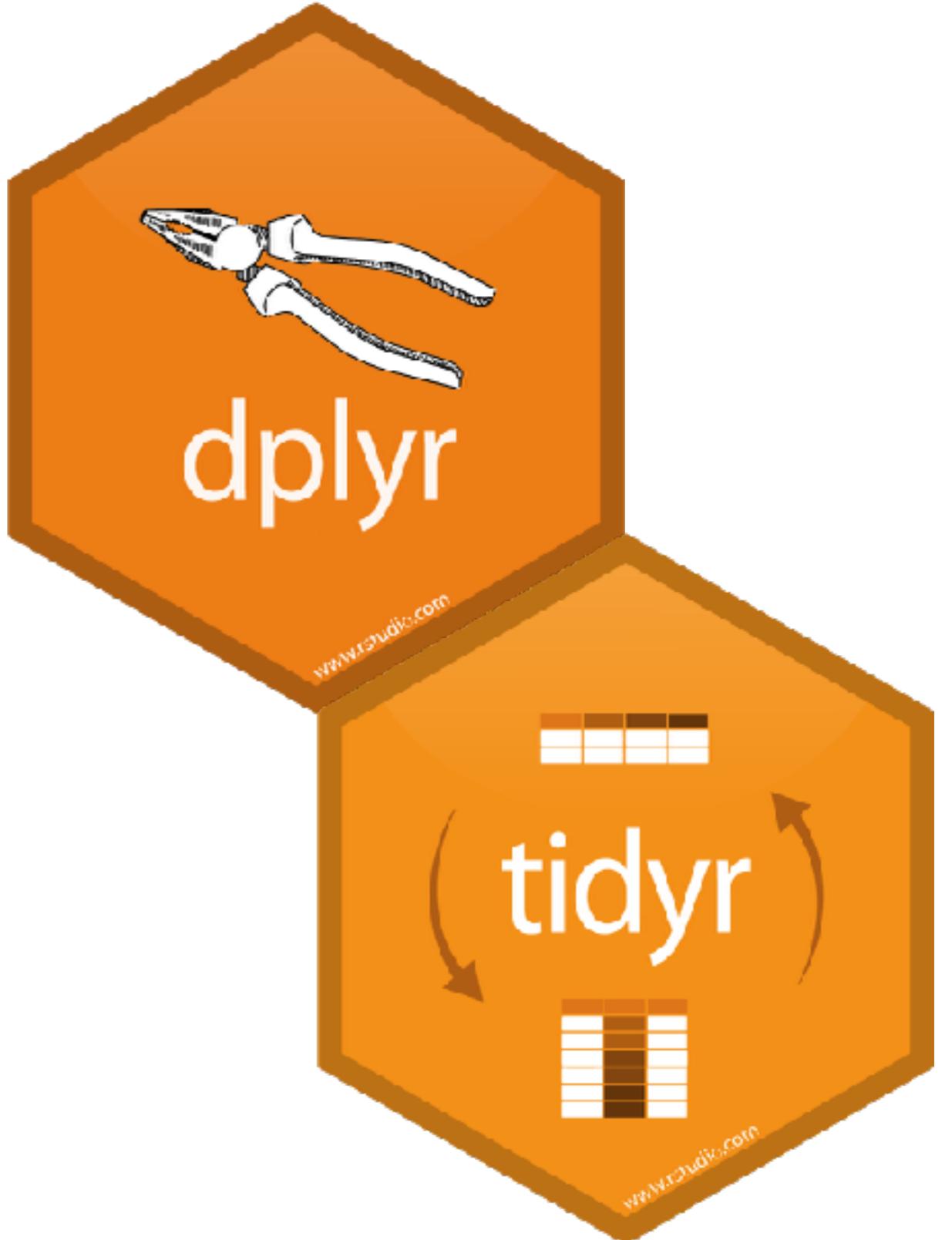
Please do basic exercises 2-I and 2-II.

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



## 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).

| iris > |              |             |              |             |         |
|--------|--------------|-------------|--------------|-------------|---------|
| Filter |              |             |              |             |         |
|        | 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  |
| 7      | 4.6          | 3.4         | 1.4          | 0.3         | setosa  |
| 8      | 5.0          | 3.4         | 1.5          | 0.2         | setosa  |

**dplyr:** `%>%`

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

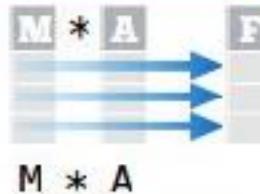


Each **variable** is saved in its own **column**



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.



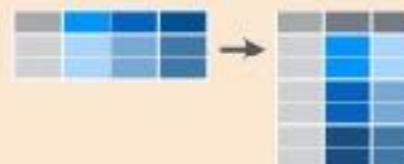
**dplyr:** `data_frame(a = 1:3, b = 4:6)`  
Combine vectors into data frame (optimized).

**dplyr:** `arrange(mtcars, mpg)`  
Order rows by values of a column (low to high).

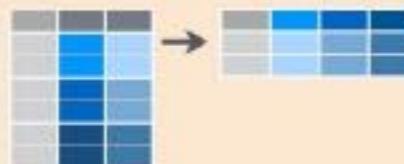
**dplyr:** `arrange(mtcars, desc(mpg))`  
Order rows by values of a column (high to low).

**dplyr:** `rename(tb, y = year)`  
Rename the columns of a data frame.

## 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.

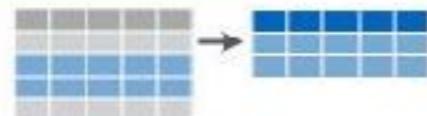


**tidyR:** `separate(storms, date, c("y", "m", "d"))`  
Separate one column into several.



**tidyR:** `unite(data, col, ..., sep)`  
Unite several columns into one.

## 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.

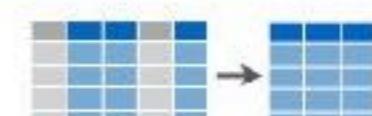
**dplyr:** `slice(iris, 10:15)`

Select rows by position.

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

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

## 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("x:"))`

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.

### 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 |

`filter()`

## Subset Observations (Rows)



## 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()`

## Subset Variables (Columns)



## 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 × 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 × 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()`

## Make New Variables



## 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()

---

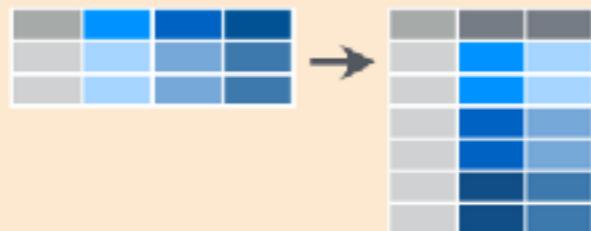
```
> mutate(iris,
 # compute the area of a single petal
 petal_area = pi * Petal.Length * Petal.Width,
 # abbreviate the name of the species
 Species_abbr = substring(Species, 1, 3)
)
```

# 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()



`tidy::gather(cases, "year", "n", 2:4)`  
Gather columns into rows.



`tidy::spread(pollution, size, amount)`  
Spread rows into columns.

# Wide

# VS.

# Long

```
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
```

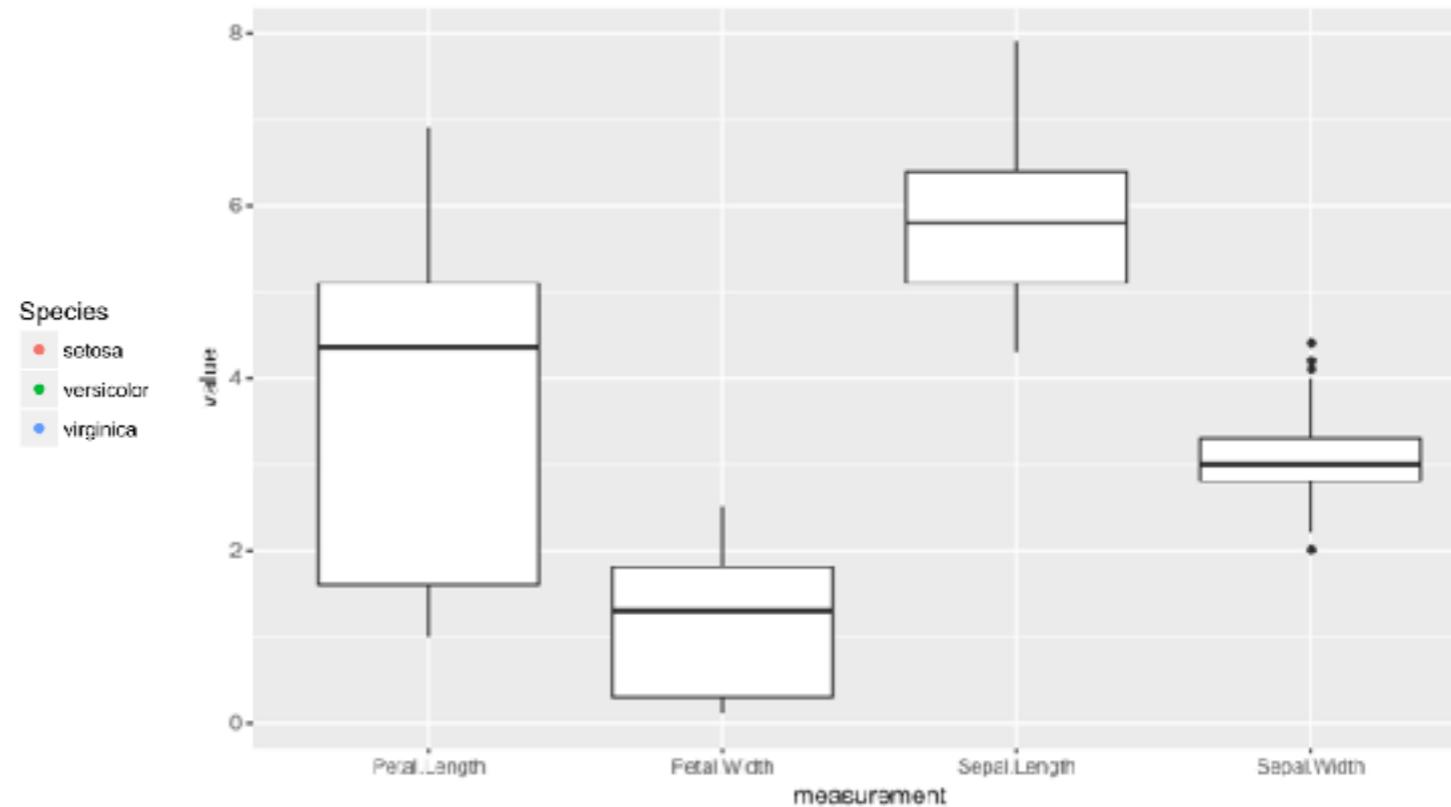
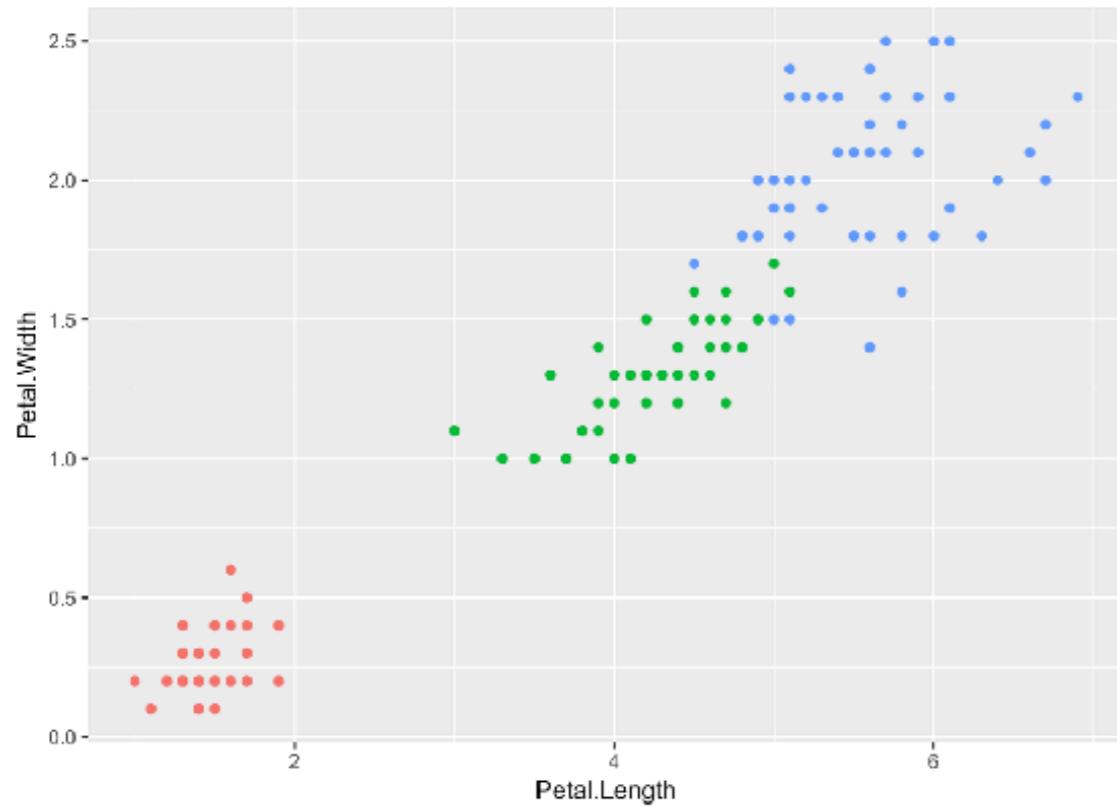
```
A tibble: 600 x 4
 Species observation measurement value
 <chr> <int> <chr> <dbl>
1 setosa 1 Sepal.Length 5.10
2 setosa 1 Sepal.Width 3.50
3 setosa 1 Petal.Length 1.40
4 setosa 1 Petal.Width 0.200
5 setosa 2 Sepal.Length 4.90
6 setosa 2 Sepal.Width 3.00
7 setosa 2 Petal.Length 1.40
8 setosa 2 Petal.Width 0.200
9 setosa 3 Sepal.Length 4.70
10 setosa 3 Sepal.Width 3.20
... with 590 more rows
```

- More information per row
  - Combines all measurements on a single individual
  - Necessary to plot matching measurements
- More information per column
  - No values as column headers (tidy)
  - Single observation in single row (tidy)
  - Necessary to plot large amounts of data in a single plot

# Wide

# VS.

# Long



- More information per row
- Combines all measurements on a single individual
- **Necessary to plot matching measurements**

- More information per column
- No values as column headers (tidy)
- Single observation in single row (tidy)
- **Necessary to plot large amounts of data in a single plot**

## Function: gather()

---

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

# Function: gather()

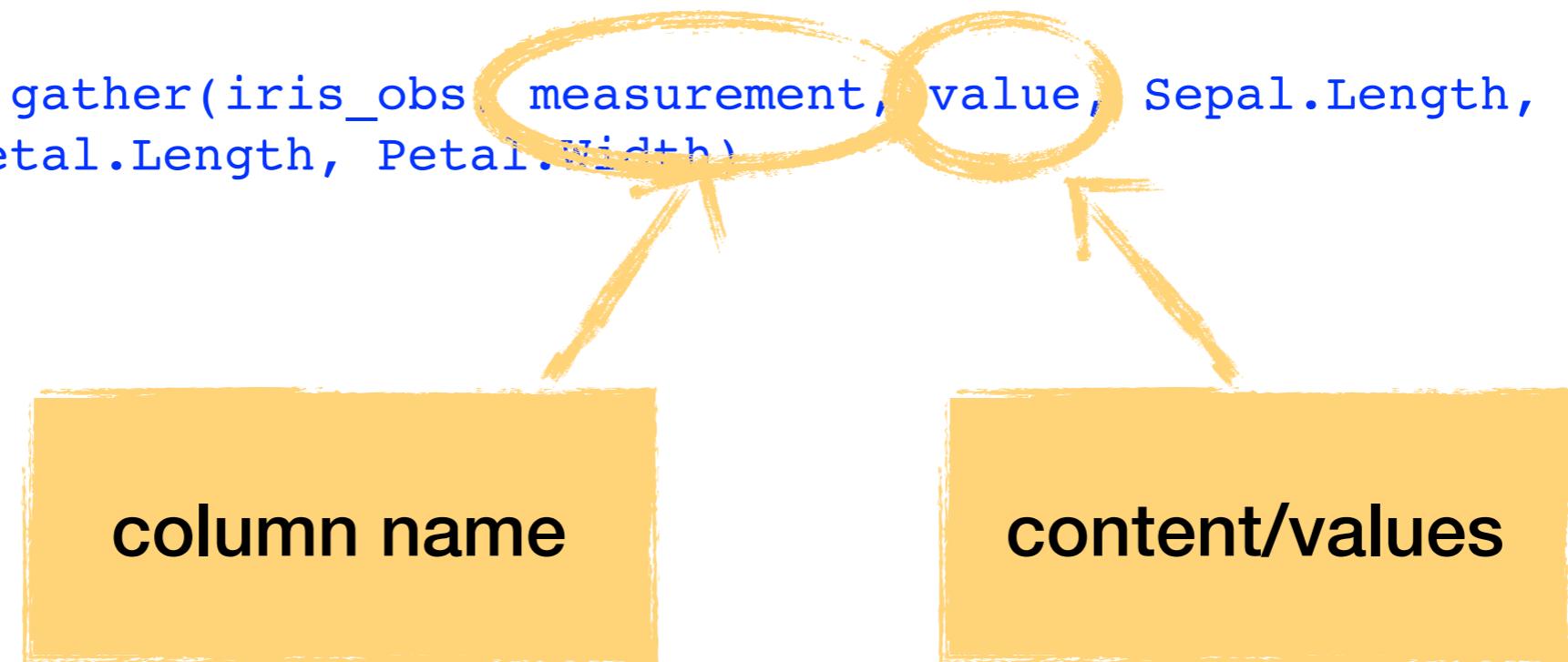
---

```
> iris_obs <- mutate(iris, observation = 1:n())
> iris_obs
A tibble: 150 × 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, Sepal.Length,
Sepal.Width, Petal.Length, Petal.Width)
> iris_long
```



# Function: gather()

---

```
> iris_obs <- mutate(iris, observation = 1:n())
> iris_obs
> iris_long <- gather(iris_obs, measurement, value, Sepal.Length,
Sepal.Width, Petal.Length, Petal.Width)
> 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
```

## Function: spread()

---

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

# Function: spread()

---

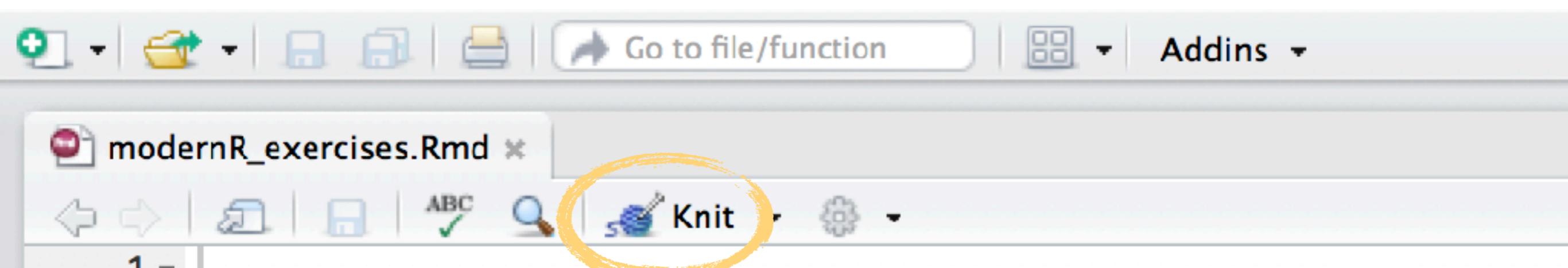
```
> iris_wide <- spread(iris_long, measurement, value)
> iris_wide
A tibble: 150 x 6
 Species observation Petal.Length Petal.Width Sepal.Length Sepal.Width
 <chr> <int> <dbl> <dbl> <dbl> <dbl>
1 setosa 1 1.40 0.200 5.10 3.50
2 setosa 2 1.40 0.200 4.90 3.00
3 setosa 3 1.30 0.200 4.70 3.20
4 setosa 4 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
```

# Exercises

Please do basic exercises 3-I and 3-II.

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

# All done? Time to knit!



The screenshot shows the RStudio interface with the 'modernR\_exercises.Rmd' file open. The 'Knit' button in the toolbar is highlighted with a yellow circle. The code in the document includes YAML front matter and a note about the workshop.

```
1 ---
2 title: "Modern R with tidyverse"
3 author: "[Insert your name]"
4 output:
5 html_document:
6 toc: true
7 ---
8
9 *This document is part of the workshop **Introduction to R & Data
10
11 # Introduction
12
13 In this document, we explore Crane migration, through the GPS dat
data was kindly provided for this course by Sasha Pekarsky at the
```

# Introduction to R & data

---

Before you leave...

# 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](https://github.com/UtrechtUniversity/R-data-cafe)
- Next edition: **Monday 25/2,  
15:00-17:00**



## Feedback is cool.

Please send your feedback, remarks, questions to us:

Barbara: [b.m.i.vreede@uu.nl](mailto:b.m.i.vreede@uu.nl)

Jonathan: [j.debruin1@uu.nl](mailto:j.debruin1@uu.nl)

**Or generate an issue on github!**

[tinyurl.com/introRData](http://tinyurl.com/introRData)

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