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

Welcome!

Please download **all of** the course material:

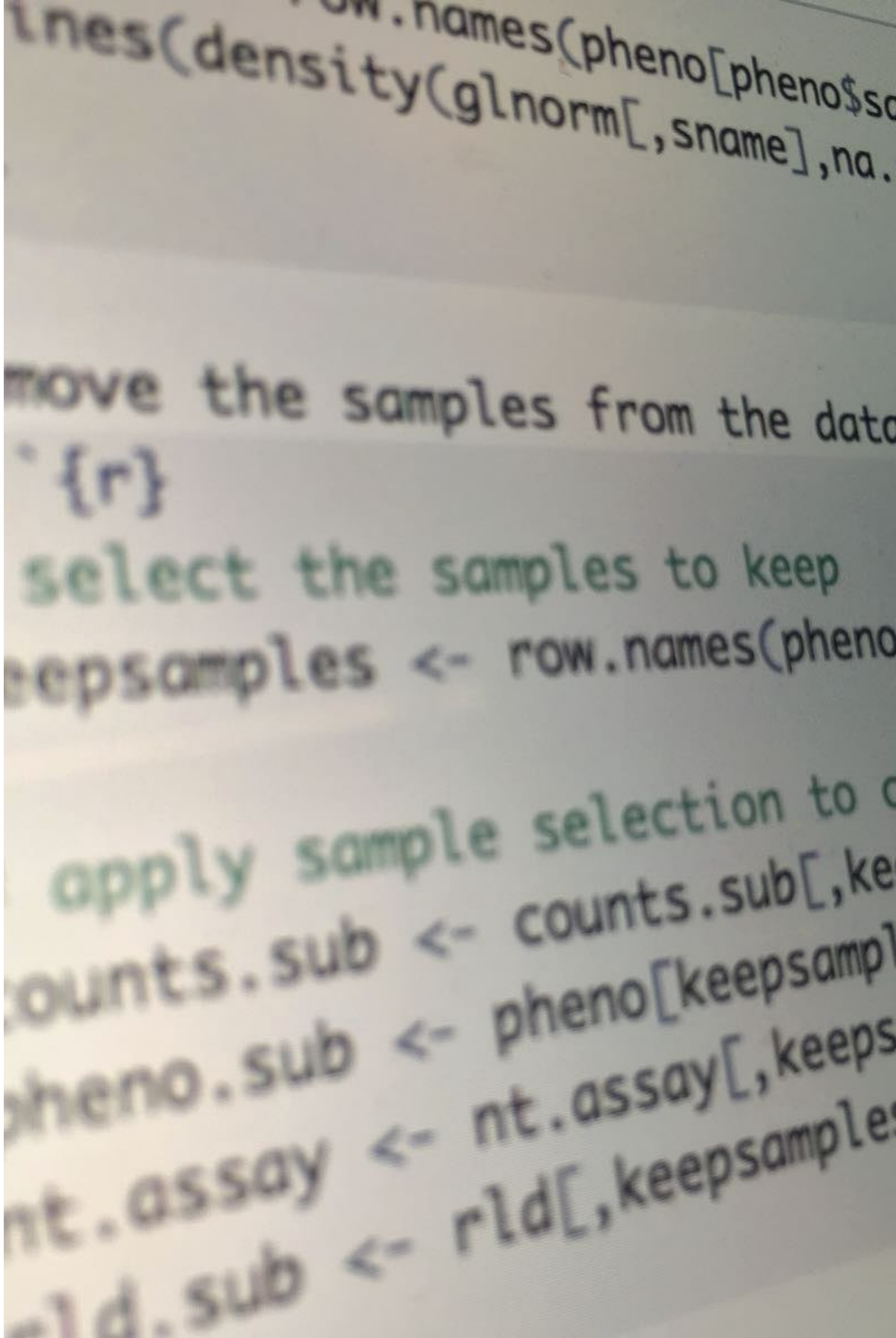
---

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

and store it in a single, accessible folder on your computer. **Don’t forget to unzip!**

Introduction to R & data

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



Quick intro: this is us! And who are you?

Bianca Kramer
subject specialist Biomedical Science
UU Library
b.m.r.kramer@uu.nl

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

What's with those sticky notes?

I need some assistance, please!

I have finished the exercise and am ready to move on!

No sticky note?

Introduction to R & data

Part 1: Basics of R

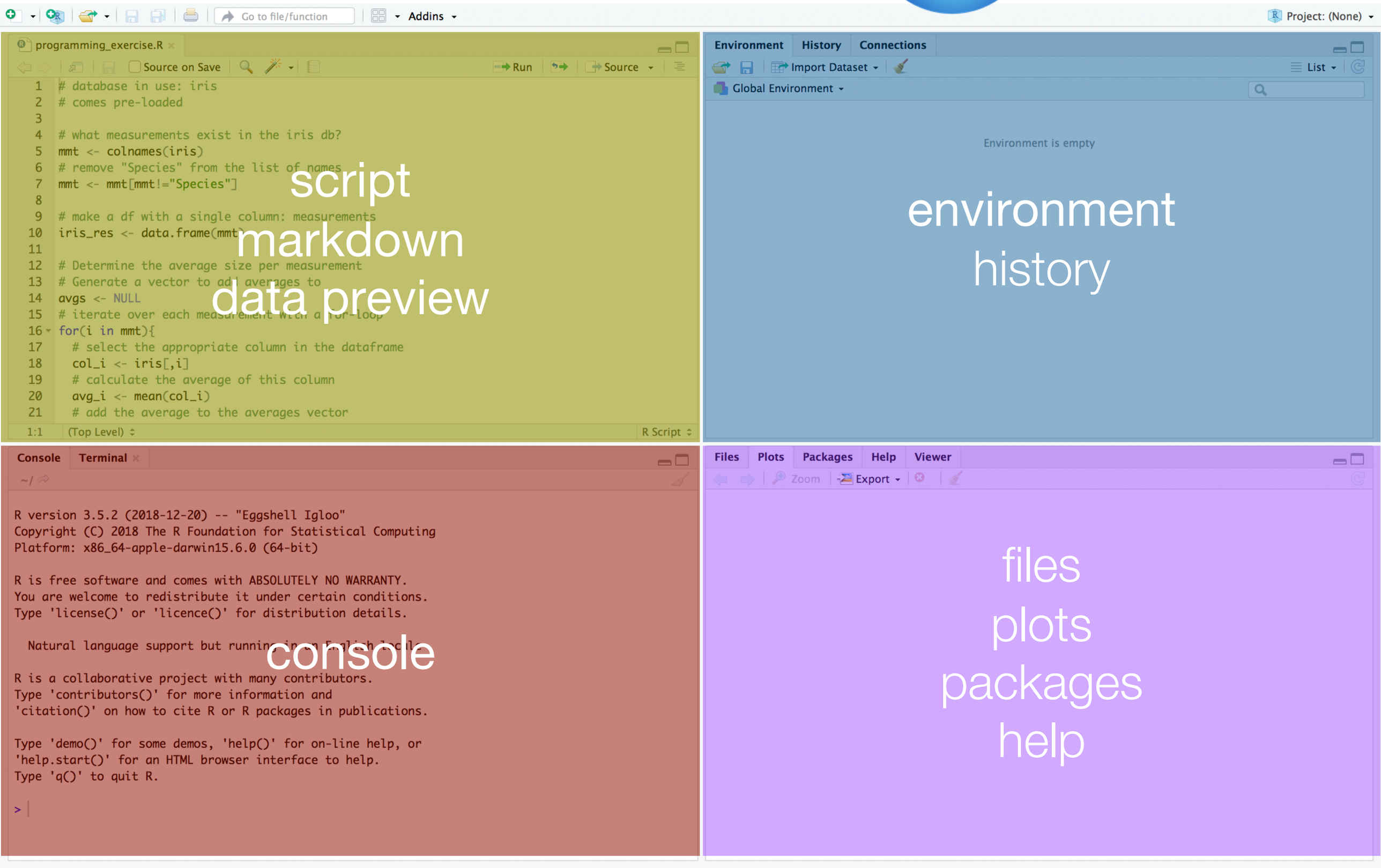
What is R?

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

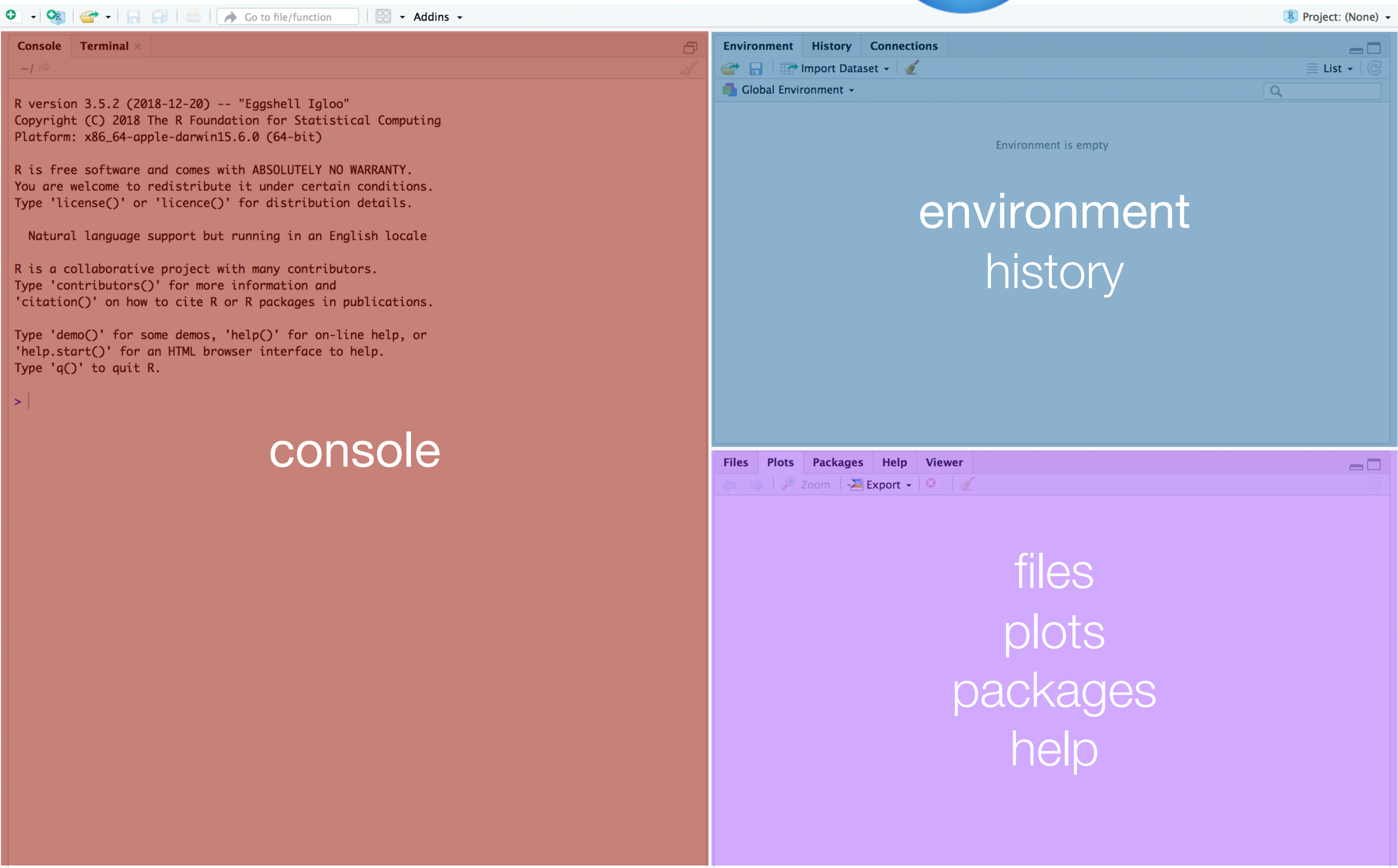
We will work in RStudio

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

The Rstudio interface



The Rstudio interface



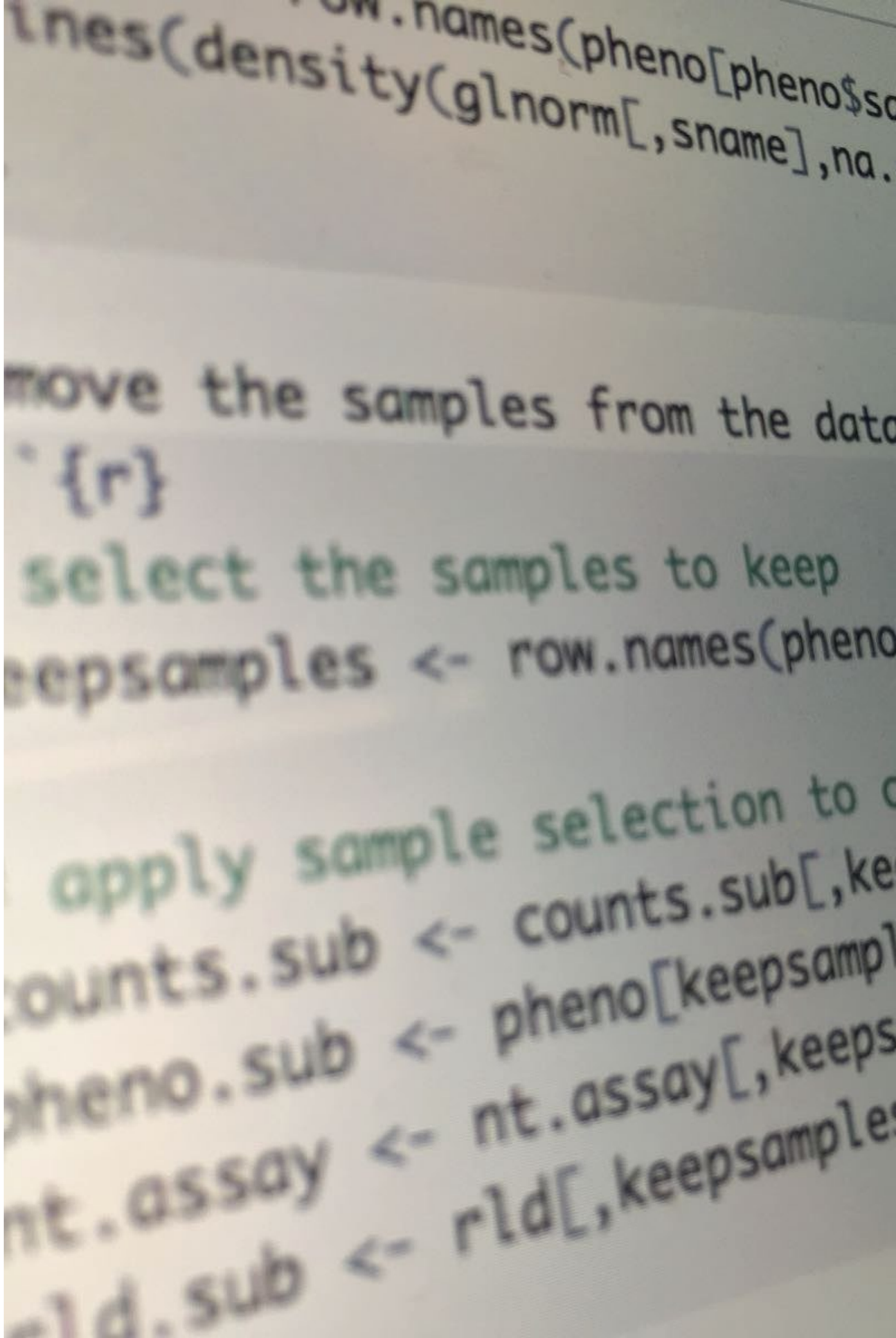
Have you downloaded the course material?

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

Please store it in a single, accessible folder on your computer. **Don’t forget to unzip!**

R syntax & the console

- Data types in R
- Combining data
- First functions



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!
```

Variable assignment

```
>
```

Variable assignment

```
> x <- 6
```

Base R
Cheat Sheet

Variable assignment

```
> x <- 6

> x <- "hi!"

> 6 -> x
```

```
> x = 6

> 6 = x
Error in 6 = x : invalid (do_set) left-hand side to assignment
```

Variable Assignment

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

Maths Functions

```
> x * 3
[1] 18

> y <- x + 2

> log2(y)
[1] 3
```

Maths Functions

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

Exercise

1. Make the following calculation in R:

$$\frac{(1+5)\times(7\ -2)}{9}$$

2. Assign the result of the calculation to a variable.

3. **Bonus:** Round off the result to 1 decimal. *Tip: Use the **Maths Functions** section of your cheat sheet!*

Exercise

1. Make the following calculation in R:

$$\frac{(1+5)\times(7-2)}{9}$$

```
> ((1+5)*(7-2))/9
```

2. Assign the result of the calculation to a variable.

```
> MyCalc <- ((1+5)*(7-2))/9
```

3. **Bonus:** Round off the result to 1 decimal. *Tip: Use the **Maths Functions** section of your cheat sheet!*

```
> round(MyCalc,1)
```

Another data type: logical

T	TRUE
F	FALSE

```
> T
[1] TRUE
```

```
> 1==1
[1] TRUE
```

```
> 2!=4
[1] TRUE
```

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

Combining data: creating vectors

```
> c(1,2,3)
[1] 1  2  3
```

a **numeric** vector

```
> c("a","b","c")
[1] "a" "b" "c"
```

a **character** vector

```
> c(T,TRUE,F)
[1] TRUE  TRUE FALSE
```

a **logical** vector

Vector functions

```
> p <- 1:5
> p
[1] 1 2 3 4 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. Make the following vector using **`seq()`**:

```
[1] 8 10 12 14
```

2. Make another vector using rep():

```
[1] 7 7 7 7
```

3. Combine all three functions **`c()`**, **`rep()`**, and **`seq()`** to generate this vector:

```
> MyVec <- ...
> MyVec
[1] 12 12 12 4 5 6
```

4. **Bonus:** can you sort the values in the last vector from low to high?

Exercise

1. Make the following vector using **`seq()`**:

```
> seq(8,14, by=2)
[1] 8 10 12 14
```

2. Make another vector using rep():

```
> rep(7,4)
[1] 7 7 7 7
```

3. Combine all three functions **`c()`**, **`rep()`**, and **`seq()`** to generate this vector:

```
> MyVec <- c(rep(12,3),seq(4,6))
> MyVec
[1] 12 12 12 4 5 6
```

4. **Bonus:** can you sort the values in the last vector from low to high?

```
> sort(MyVec)
```

Vectors and factors

```
> chars <- c("a","b","a","c")
> chars
[1] "a" "b" "a" "c"

> F_chars <- as.factor(chars)
> F_chars
[1] a b a c
Levels: a b c

> as.character(F_chars)
[1] "a" "b" "a" "c"
```

Vectors and factors

```
> nums <- 10:13
> nums
[1] 10 11 12 13

> F_nums <- as.factor(nums)
> F_nums
[1] 10 11 12 13
Levels: 10 11 12 13

> as.numeric(F_nums)
[1] 1 2 3 4
```

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.

```
> vt <- ...
```

2. Turn it into a 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.)

Exercise

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

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

2. Turn it into a factor.

```
> ft <- as.factor(vt)
```

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 = **N**ot **A**vailable

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?

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

NA != NULL

`NA`	Information is **Not Available**
`NULL`	Information **does not exist**
`"None"` or `0`	Data entry specifying **content of 0**

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: 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)
[1] FALSE
> is.na(NULL)
```

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)
[1] FALSE
> is.na(NULL)
[1] logical(0)
```

Vectors, lists, and data frames

```
> v1 <- 4:6
> v2 <- rep(3,3)

> c(v1,v2)
[1] 4 5 6 3 3 3

> list(v1,v2)
[[1]]
[1] 4 5 6

[[2]]
[1] 3 3 3

> data.frame(v1,v2)
  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()`

Let's breathe and recap!

What data types have you encountered so far?

`logical`

`numeric`

`character`

How can data be absent?

`NA` (not available)

`NULL` (non-existent)

And what data collections have you encountered?

`vector` (one dimension)

`factor` (one dimension, level-based)

`data frame` (two dimensions)

`list` (++ dimensions)

Functions

What functions have you encountered so far?

```
> c()
> rep()
> as.logical()
> data.frame()
> is.na()
> 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)

Help!

function & package names

table {base} R Documentation

Cross Tabulation and Table Creation

Description

`table` uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.

Usage

how would you use the function?

```
table(..., 
      exclude = if (useNA == "no") c(NA, NaN),
      useNA = c("no", "ifany", "always"),
      dnn = list.names(...), deparse.level = 1)
```

wait what? These are extra arguments. If you see `argument = "default"` then the setting is already specified, so you won't have to.

functions so related they share a help page

```
as.table(x, ...)
is.table(x)

## S3 method for class 'table'
as.data.frame(x, row.names = NULL, ...,
```

Help? Scroll down!

Examples

```
require(stats) # for rpois and xtabs
## Simple frequency distribution
table(rpois(100, 5))
## Check the design:
with(warpbreaks, table(wool, tension))
table(state.division, state.region)
```

```
# simple two-way contingency table
with(airquality, table(cut(Temp, quantile(Temp)), Month))

a <- letters[1:3]
table(a, sample(a))                         # dnn is c("a", "")
table(a, sample(a), deparse.level = 0) # dnn is c("", "")
table(a, sample(a), deparse.level = 2) # dnn is c("a", "sample(a)")

## xtabs() <-> as.data.frame.table() :
UCBAdmissions ## already a contingency table
DF <- as.data.frame(UCBAdmissions)
class(tab <- xtabs(Freq ~ ., DF)) # xtabs & table
## tab *is* "the same" as the original table:
all(tab == UCBAdmissions)
```

Take a break!

"Fresh Cup" by mikecpeck

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
```

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

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

> m[m %in% c(6,8,11)]
[1] 8 6

> 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	

Selecting Vector Elements

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Exercise

1. Generate a vector with numbers from 5 to 10.
   (If you can, try to use vector notation `p:q` *or the function* `seq()` *for this!)*

   ```
   > m <- ...
   ```

2. Return only the 2nd and 4th number in the vector.

   ```
   > m[...]
   ```

3. Return only numbers larger than 8.

   ```
   > m[...]
   ```

Exercise

1. Generate a vector with numbers from 5 to 10.
   (If you can, try to use vector notation `p:q` *or the function* `seq()` *for this!)*

```
> m <- 5:10
```

2. Return only the 2nd and 4th number in the vector.

```
> m[c(2,4)]
```

3. Return only numbers larger than 8.

```
> m[m>8]
```

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
> mylist[[1]][2]
[1] 5
> mylist[1][2]
[[1]]
NULL
```

select the first list element

select the **content** of the first list element

Indexing a data frame

Matrix subsetting

`df[ , 2]`

`df[2, ]`

row → `df[2, 2]` ← column

name	age	pet
Ann	33	Cat
Bob	25	None
Chloe	21	Iguana
Dan	45	NA

Indexing a data frame

```
> df[,2]
[1] 33 25 21 45
> df[,"age"]
[1] 33 25 21 45
> df$age
[1] 33 25 21 45

> df[2,]
  name age  pet
2  Bob  25 None
> df[df$name=="Bob",]
 name age  pet
2  Bob  25 None

> df[df$name=="Bob","age"]
[1] 25
```

```
df[,2]
df[,"age"]
df$age
```

name	age	pet
Ann	33	Cat
Bob	25	None
Chloe	21	Iguana
Dan	45	NA

```
df[2,]
df[df$name=="Bob",]
```

Exercise

1. Generate a simple data frame like this:

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

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

   (Hint: what information should you use for row indexing? What information should you use for column indexing?)

3. **Bonus:** can you use vector indexing on a column to achieve the same result?

Exercise

1. Generate a simple data frame like this:

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

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

   (Hint: what information should you use for row indexing? What information should you use for column indexing?)

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

3. **Bonus:** can you use vector indexing on a column to achieve the same result?

```
> df$name[df$age<30]
[1] Bob   Chloe
Levels: Ann Bob Chloe Dan
```

Which bracket does what?

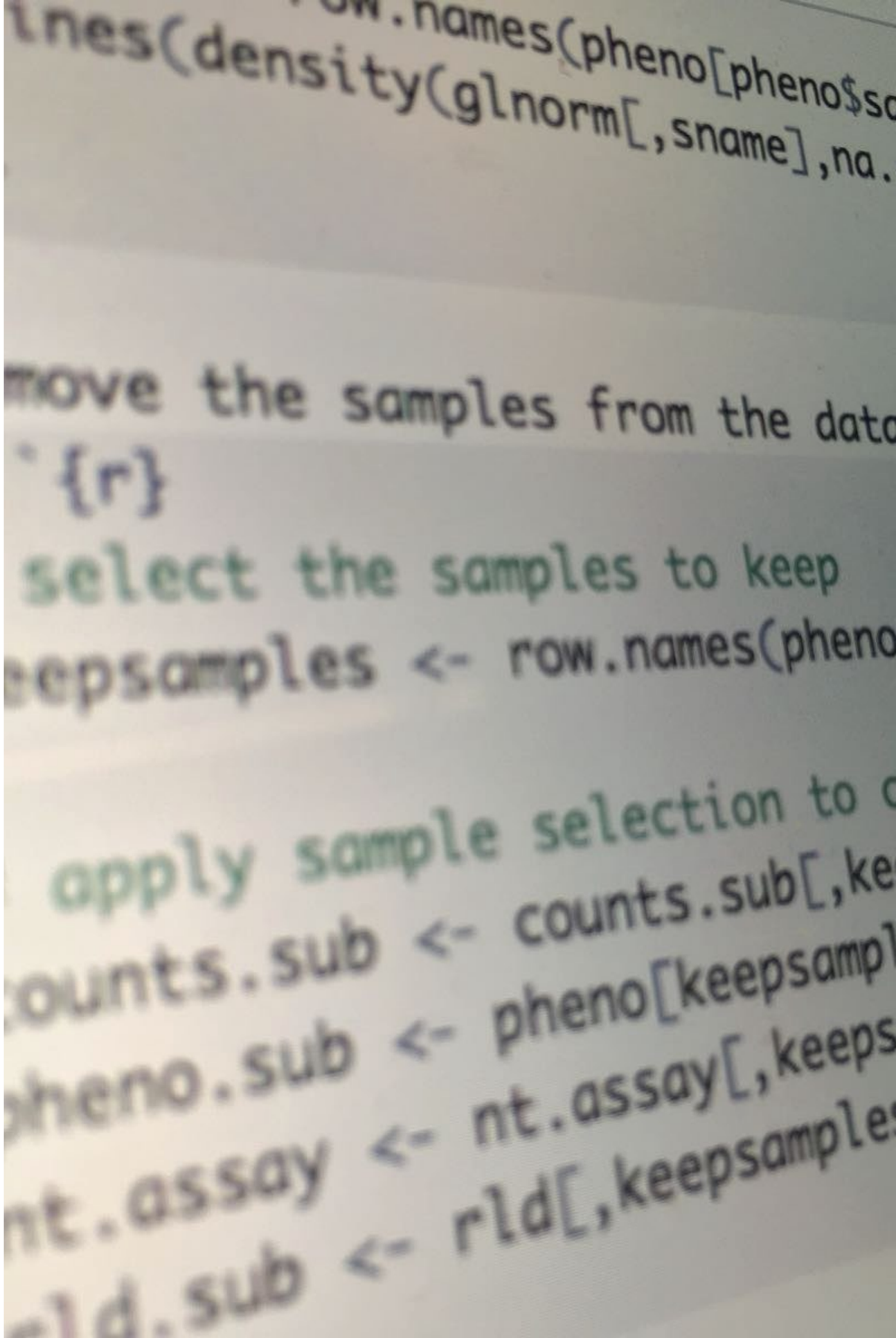
[ ] **Indexing** vectors, lists, dataframes…

( ) Passing **arguments** to functions

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

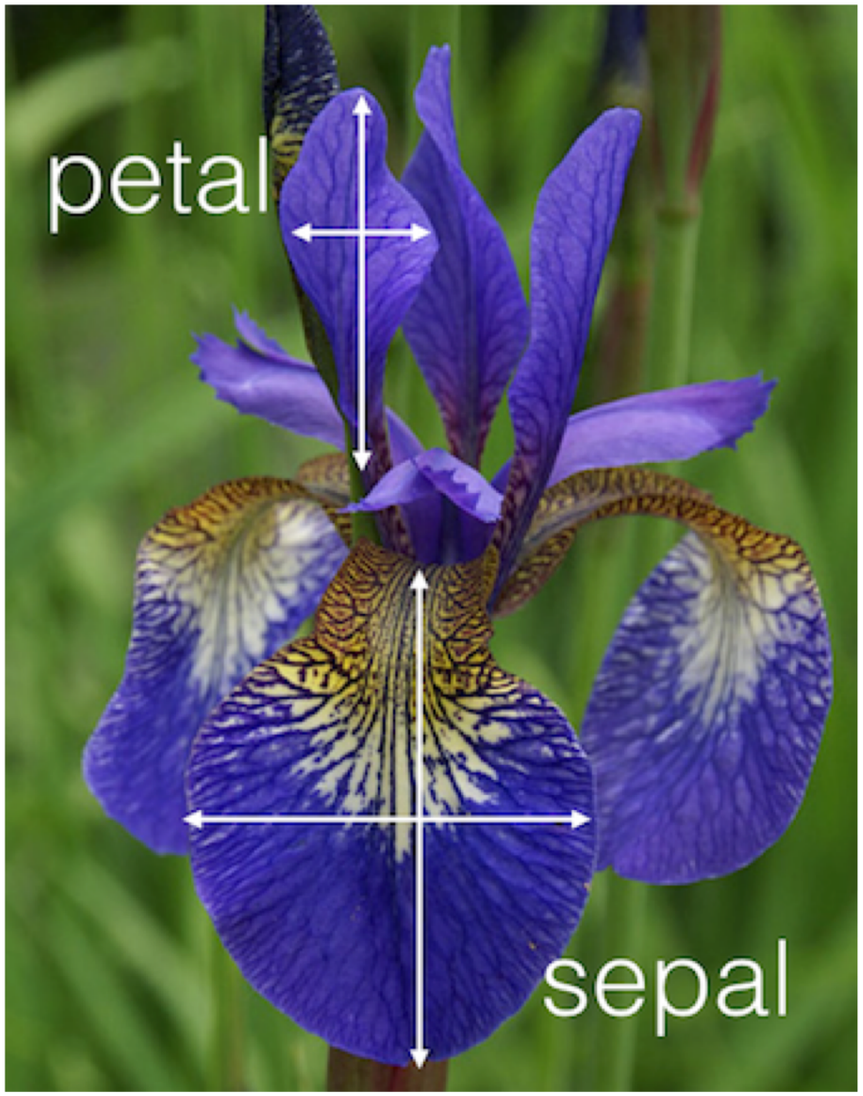
Programming

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



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
```

Exercise

Explore the `iris` dataframe, using some of the following functions:

- `head()`
- `tail()`
- `names()`
- `summary()`
- `dim()`
- `str()`

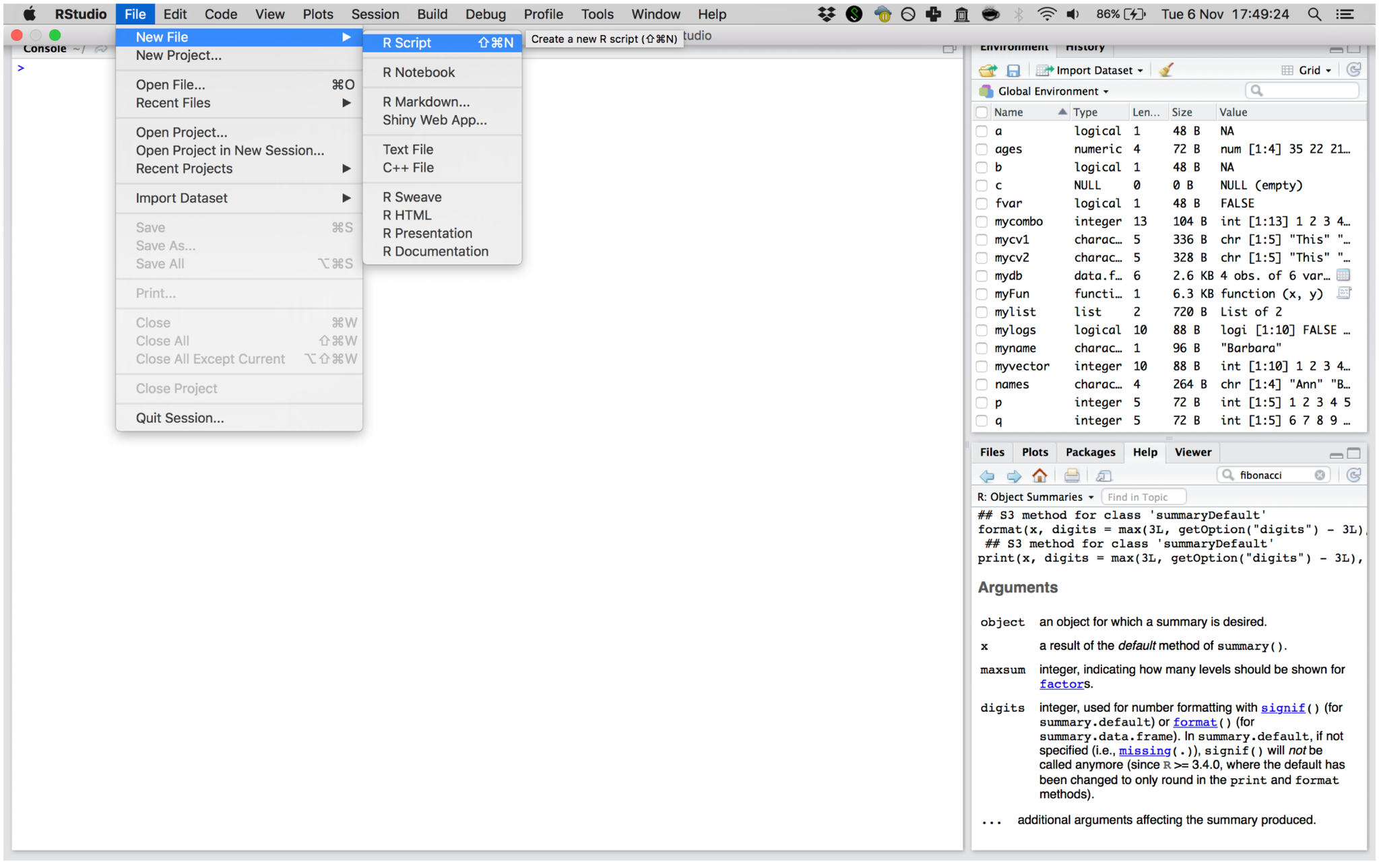
Can you figure out what these functions do? What do they teach you about the kind of data in `iris`?

Console and Scripts

Console	Code execution	-
Script	Code	Extension: .R (example_script.R)
Rmarkdown	Code + Narrative	Extension: .Rmd (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
```

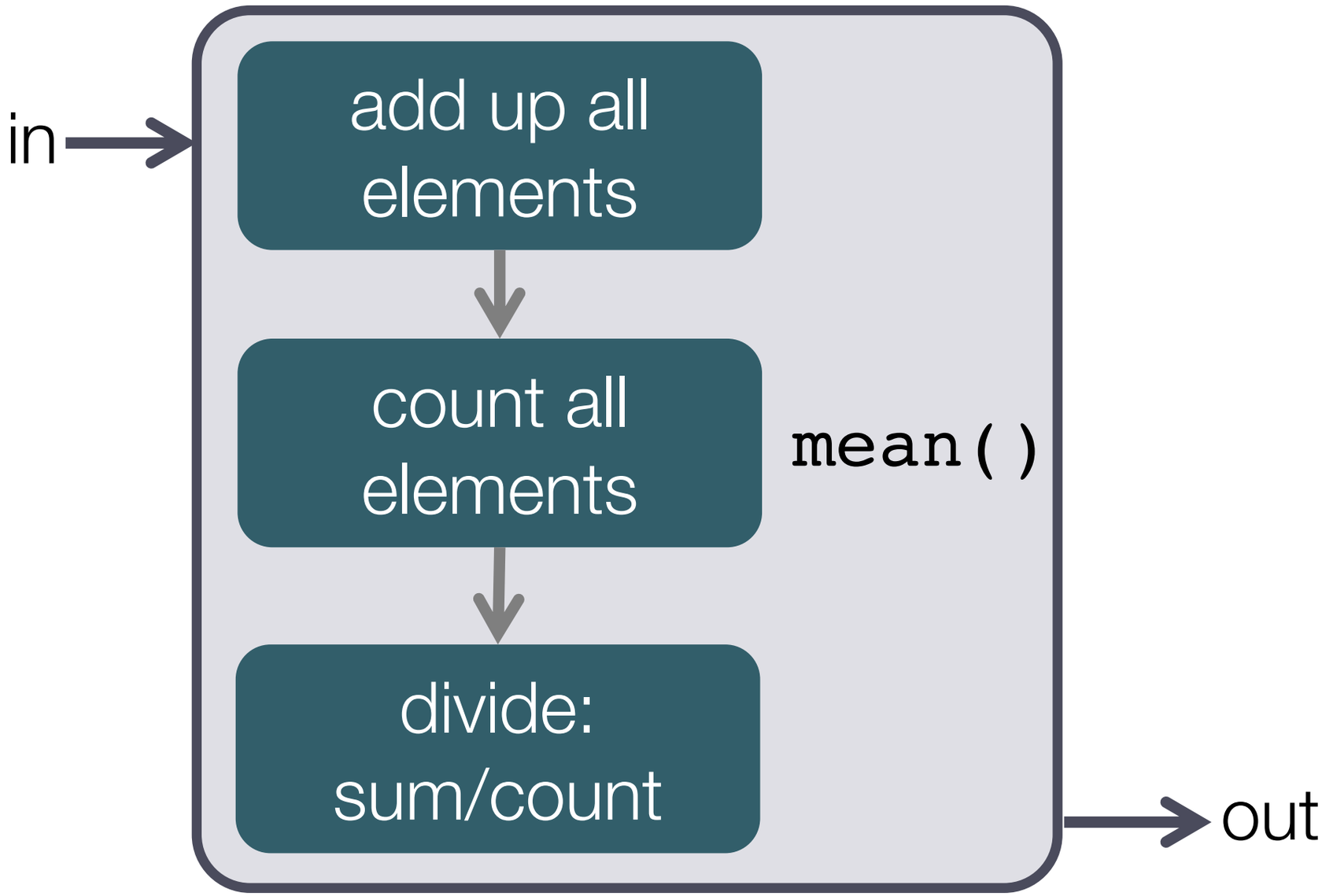
console output

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

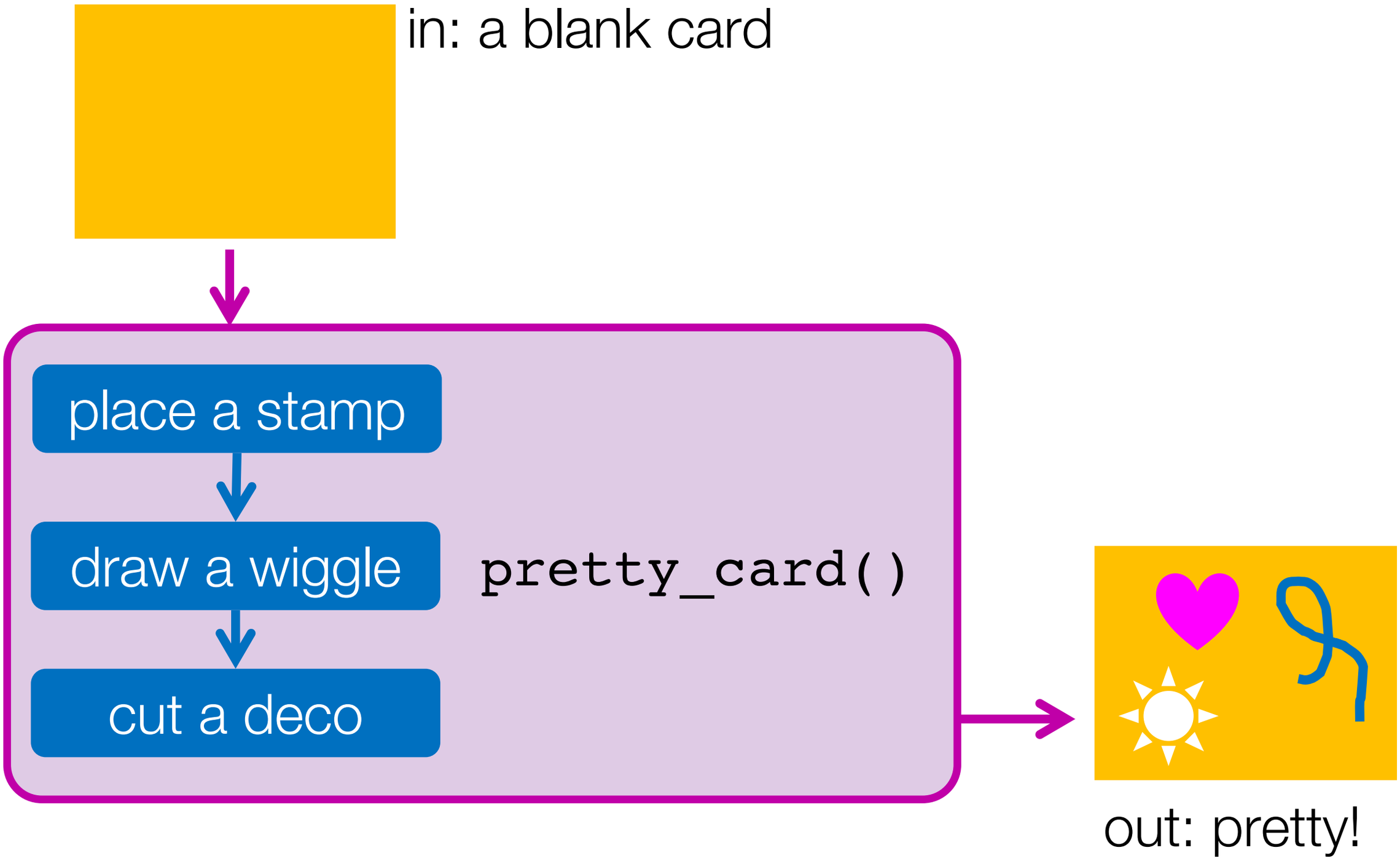
```
# Assigning the value 1 to x
x <- 1
```

Programming: functions

- Multiple instructions that form a cohesive unit
- Should be able to be repeated on different inputs

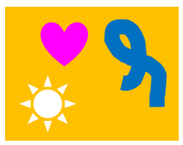


Live exercise: the assembly line of a function



How would this look in R?

```
pretty_card <- function(x){
  x <- place_stamp(x)
  x <- draw_wiggle(x)
  x <- cut_deco(x)
  return(x)
}
```

```
> pretty_card(  )
[1] 
```

Programming: functions

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

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

Exercise

1. Write a function that takes a vector as input, and returns the mean of this vector (you can use the existing function **`mean()`** inside your function).

```
apply_calc <- function(...){
    ...
    return(...)
}
```

2. Add further options to your function:
   a. for example, the standard deviation (**`sd()`**), the minimum (**`min()`**), and the maximum (**`max()`**) of your input vector.
   b. Put all of these calculations in a vector using the function **`c()`**, and return this result vector.

```
apply_calc <- function(...){
    ...



    allres <- c(...)
    return(...)
}
```

Exercise

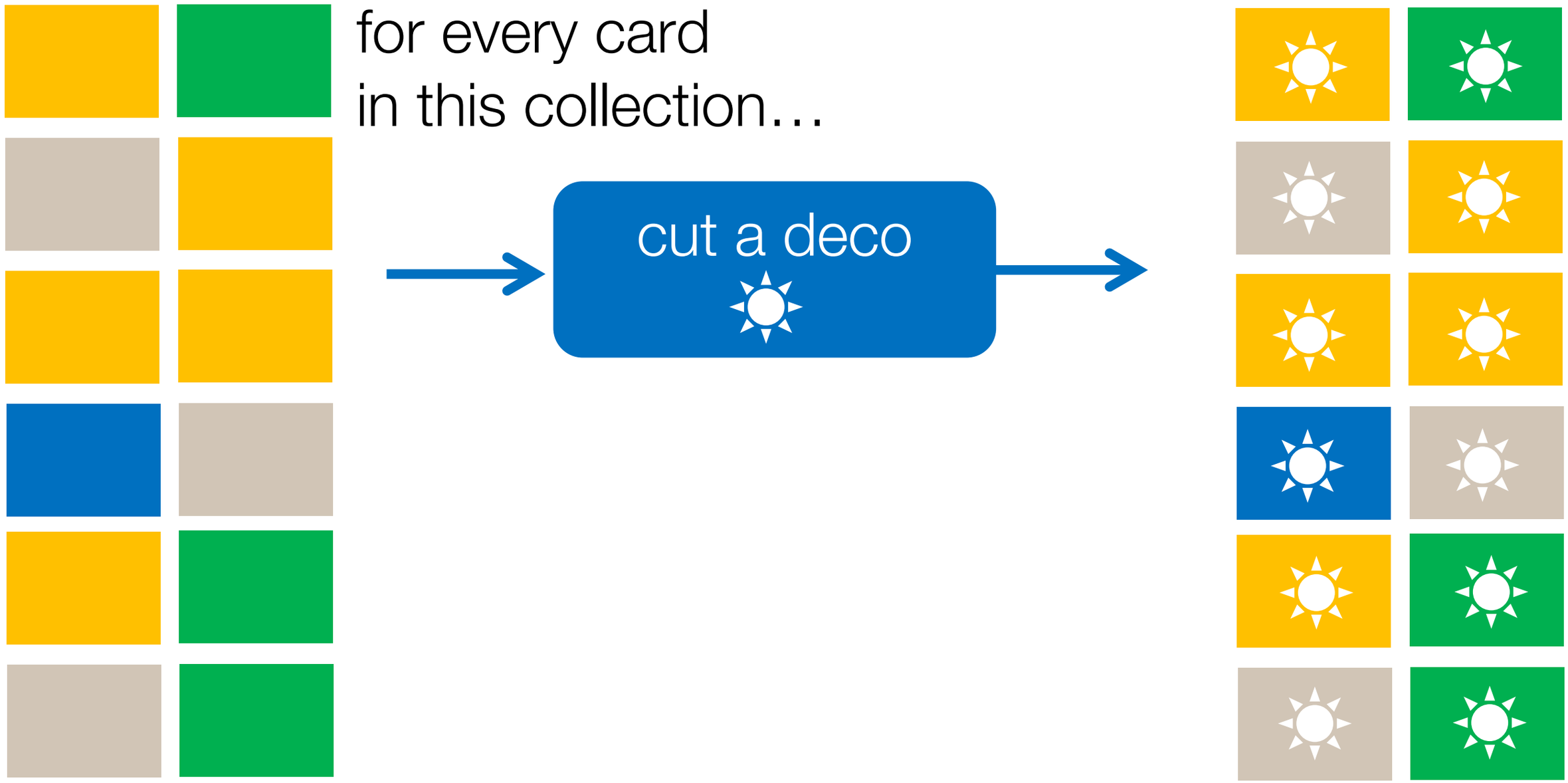
1. Write a function that takes a vector as input, and returns the mean of this vector (you can use the existing function **`mean()`** inside your function).

```
apply_calc <- function(x){
    m <- mean(x)
    return(m)
}
```

2. Add further options to your function:
   a. for example, the standard deviation (**`sd()`**), the minimum (**`min()`**), and the maximum (**`max()`**) of your input vector.
   b. Put all of these calculations in a vector using the function **`c()`**, and return this result vector.

```
apply_calc <- function(x){
    m <- mean(x)
    s <- sd(x)
    mi <- min(x)
    ma <- max(x)
    allres <- c(m,s,mi,ma)
    return(allres)
}
```

Live exercise: the repeated action in a for-loop



How would this look in R?

```
for( ■ in ■■■■■■ ){
  cut_deco(■)
}
```

```
[1] ☀
[1] ☀
[1] ☀
[1] ☀
[1] ☀
[1] ☀
```

```
for(i in 1:6){
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
```

Exercise

1. Make a vector with all the column names in **`iris`**.
   Hint: use the function **`colnames()`**.

   ```
   iriscols <- ...
   ```

2. Make a for-loop that iterates over all the column names in **`iris`**, and prints these column names.

   ```
   for(...){
        ...
   }
   ```

3. Elaborate on this for-loop: select the corresponding column in **`iris`**, and print the mean. *Hint: yes,* ***you should get a warning****! Do you understand why?*

   ```
   for(...){
        ...


   }
   ```

Exercise

1. Make a vector with all the column names in **`iris`**.
   Hint: use the function **`colnames()`**.

```
iriscols <- colnames(iris)
```

2. Make a for-loop that iterates over all the column names in **`iris`**, and prints these column names.

```
for(i in iriscols){
    print(i)
}
```

3. Elaborate on this for-loop: select the corresponding column in **`iris`**, and print the mean. *Hint: yes,* ***you should get a warning****! Do you understand why?*

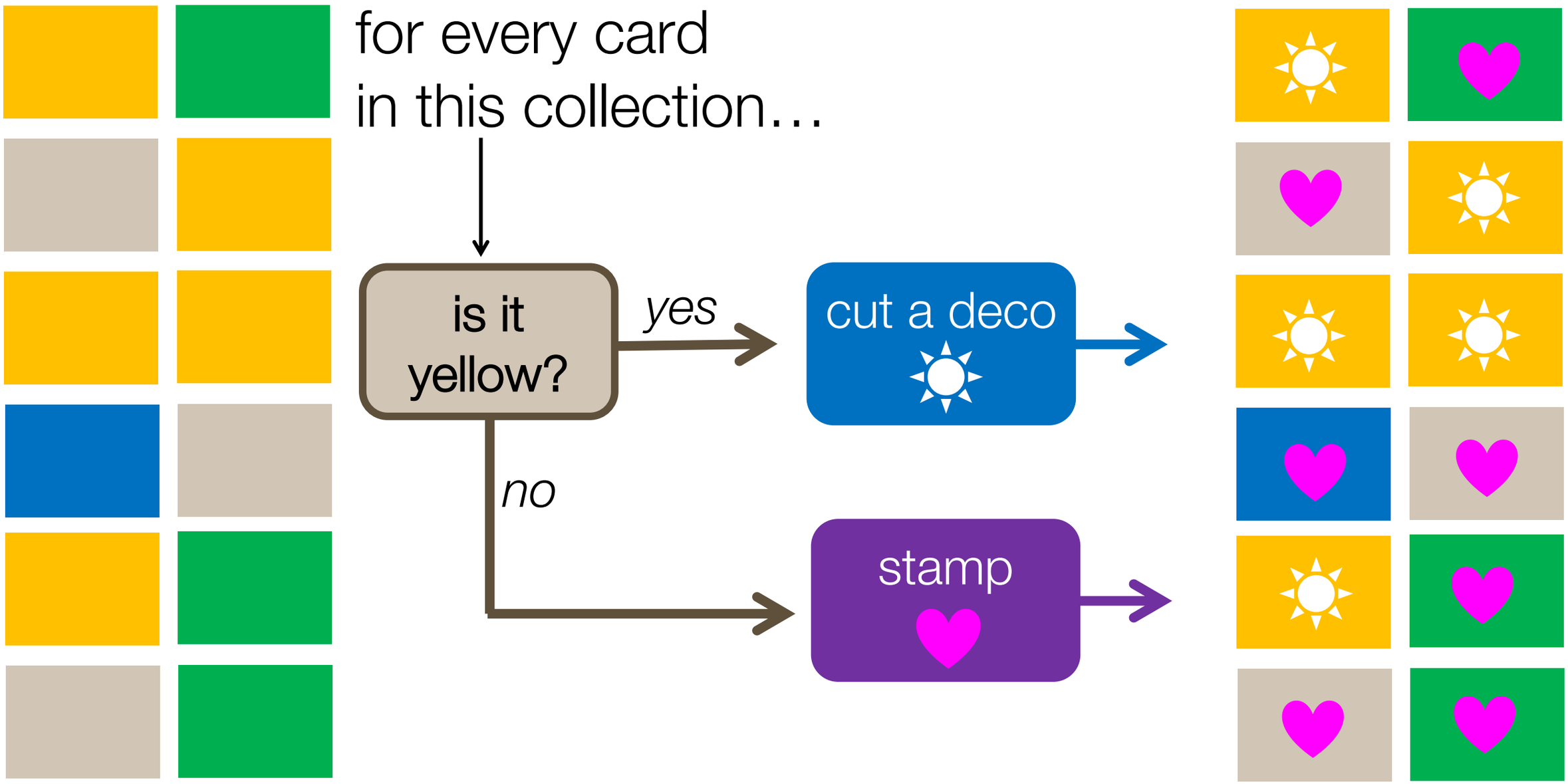
```
for(i in iriscols){
    c <- iris[,i]
    stat <- mean(c)
    print(stat)
}
```

What's wrong? And how do we fix it?

```
for(i in iriscols){
  # select the appropriate column
  c <- iris[,i]
  # calculate the mean
  stats <- mean(c)
  # print the mean
  print(stats)
}
```

```
[1] 5.843333
[1] 3.057333
[1] 3.758
[1] 1.199333
[1] NA
Warning message:
In mean.default(c) : argument is not numeric or logical:
returning NA
```

Live exercise: the selectivity of an if-statement



PS: note how this is still a for-loop? That's not a requirement!

How would this look in R?

```
for( ■ in ■■■■■■ ){
  if( ■ == ■ ){
    cut_deco( ■ )
  } else{
    stamp( ■ )
  }
}
```

```
[1] ♥
[1] ☀
[1] ☀
[1] ♥
[1] ☀
[1] ♥
```

```
for(i in 1:6){
  if(i > 3){
    print("Large!")
  } else{
    print("small...")
  }
}
```

```
[1] "small..."
[1] "small..."
[1] "small..."
[1] "Large!"
[1] "Large!"
[1] "Large!"
```

Programming: if-statement

```
d <- 5

if(is.na(d)){
  print("My data is missing!")
} else if(is.null(d)){
  print("My data does not even exist...")
} else{
  print("I have data!")
}
```

```
[1] "I have data!"
```

Exercise: choose one!

1. Copy-paste the for-loop you made in the previous exercise. Inside this for-loop, add an if-statement, so that **`mean()`** is only performed on numeric vectors.
   Hint: check the function **`is.numeric()`**.

```
for(i in iriscols){
    c <- iris[,i]
    ...
}
```

2. Are you feeling comfortable with the material?
   a. Open the file ‘programming_exercise.R’
   b. Read through the code, see if you understand it *(it is mostly the previous two exercises, but with some small tweaks!)*
   c. Scroll down to line 32 for the exercise.
   d. Read the bonus material — you can check your work with the code there.

Exercise: choose one!

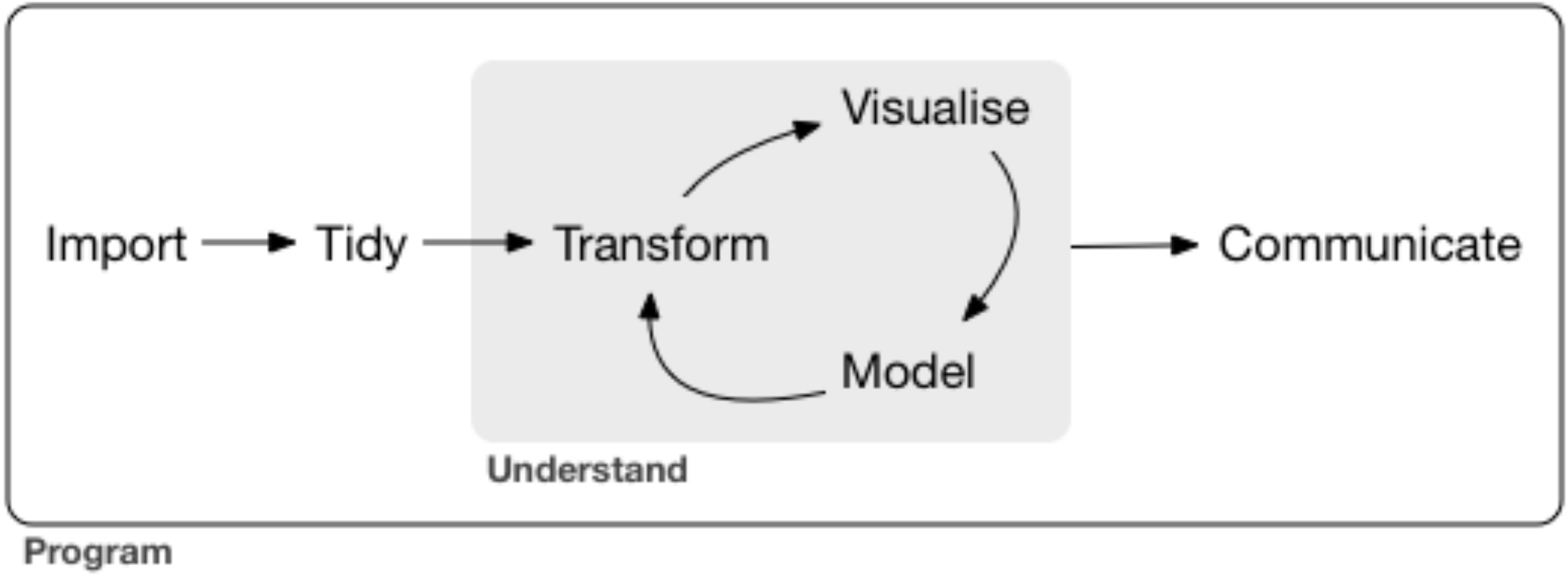
1. Copy-paste the for-loop you made in the previous exercise. Inside this for-loop, add an if-statement, so that **`mean()`** is only performed on numeric vectors.
   Hint: check the function **`is.numeric()`**.

```
for(i in iriscols){
    c <- iris[,i]
    if(is.numeric(c)){
        stat <- mean(c)
        print(stat)
    }
}
```

2. Are you feeling comfortable with the material?
   a. Open the file 'programming_exercise.R'
   b. Read through the code, see if you understand it *(it is mostly the previous two exercises, but with some small tweaks!)*
   c. Scroll down to line 32 for the exercise.
   d. Read the bonus material — you can check your work with the code there.

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)
}
```

To check before lunch: please load tidyverse

```
> library(tidyverse)
── Attaching packages ─────────────────────────────────── tidyverse 1.2.1 ──
   ggplot2 3.1.0       purrr   0.3.0
   tibble  2.0.1       dplyr   0.7.8
   tidyr   0.8.2       stringr 1.4.0
   readr   1.3.1       forcats 0.3.0
── Conflicts ────────────────────────────────────── tidyverse_conflicts() ──
dplyr::filter() masks stats::filter()
dplyr::lag()    masks stats::lag()
```

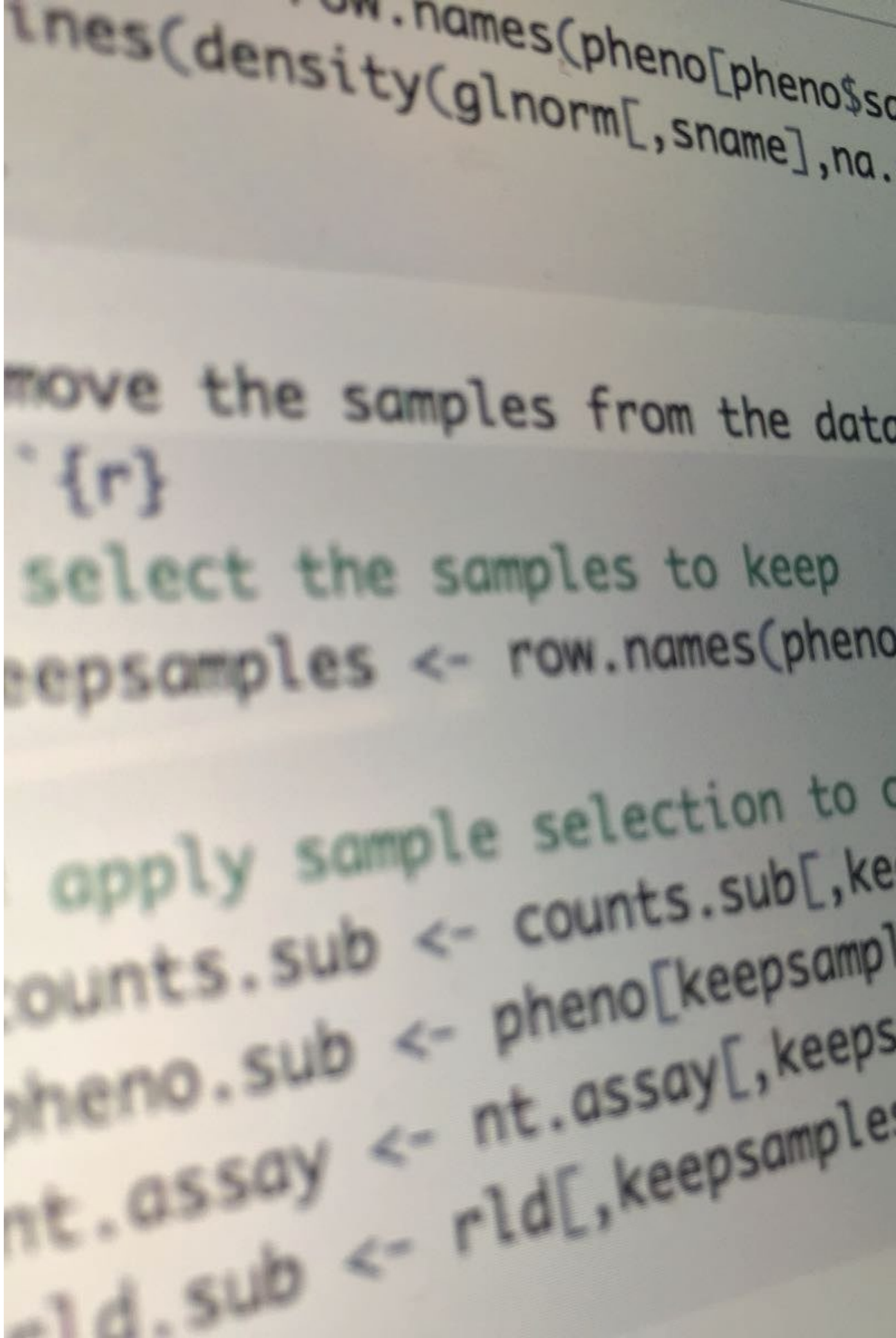
Enjoy your lunch!

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**



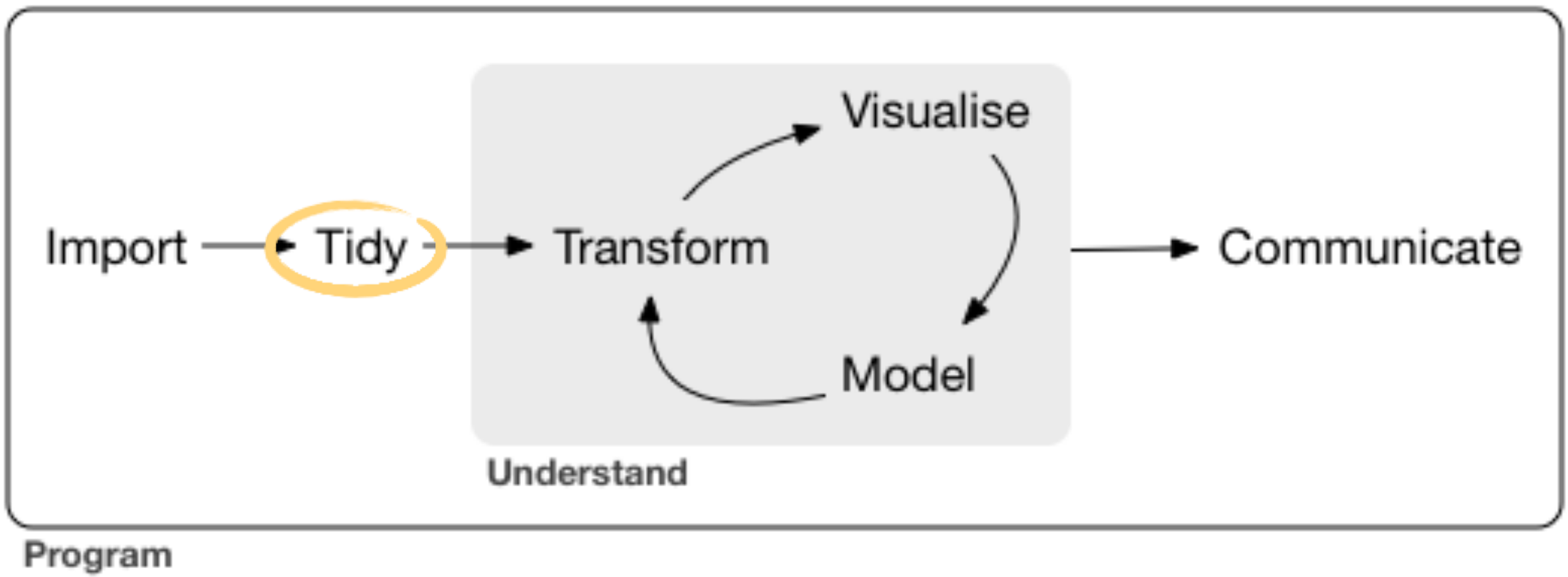
Tidyverse

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)

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

multiple observations per row

values in column names

Patient	Temp	Med_A	Temp_A	Med_B	Temp_B
122030	37.0	300	37.1	NA	NA
122021	38.2	200	38.1	85	36.5
124500	38.1	300	38.0	NA	NA
126098	39.1	NA	NA	100	36.8

Tidy data

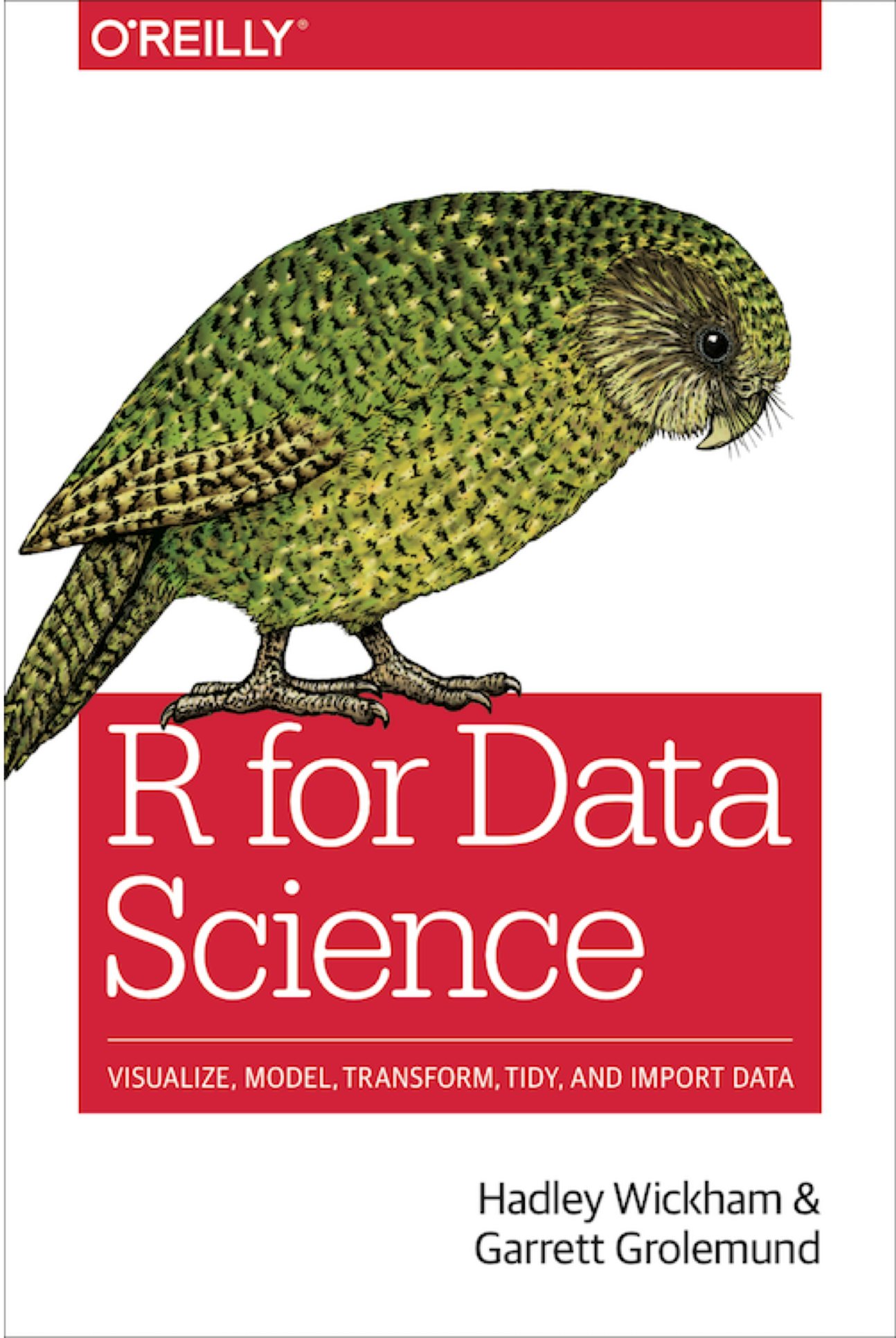
ID no longer unique per row
(but can still be used to collect all info on single patient)

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

Patient	Temp	Treatment	Dose
122030	37.0	None	NA
122030	37.1	A	300
122021	38.2	None	NA
122021	38.1	A	200
122021	36.5	B	85
124500	38.1	None	NA
124500	38.0	A	300
126098	39.1	None	NA
126098	36.8	B	100

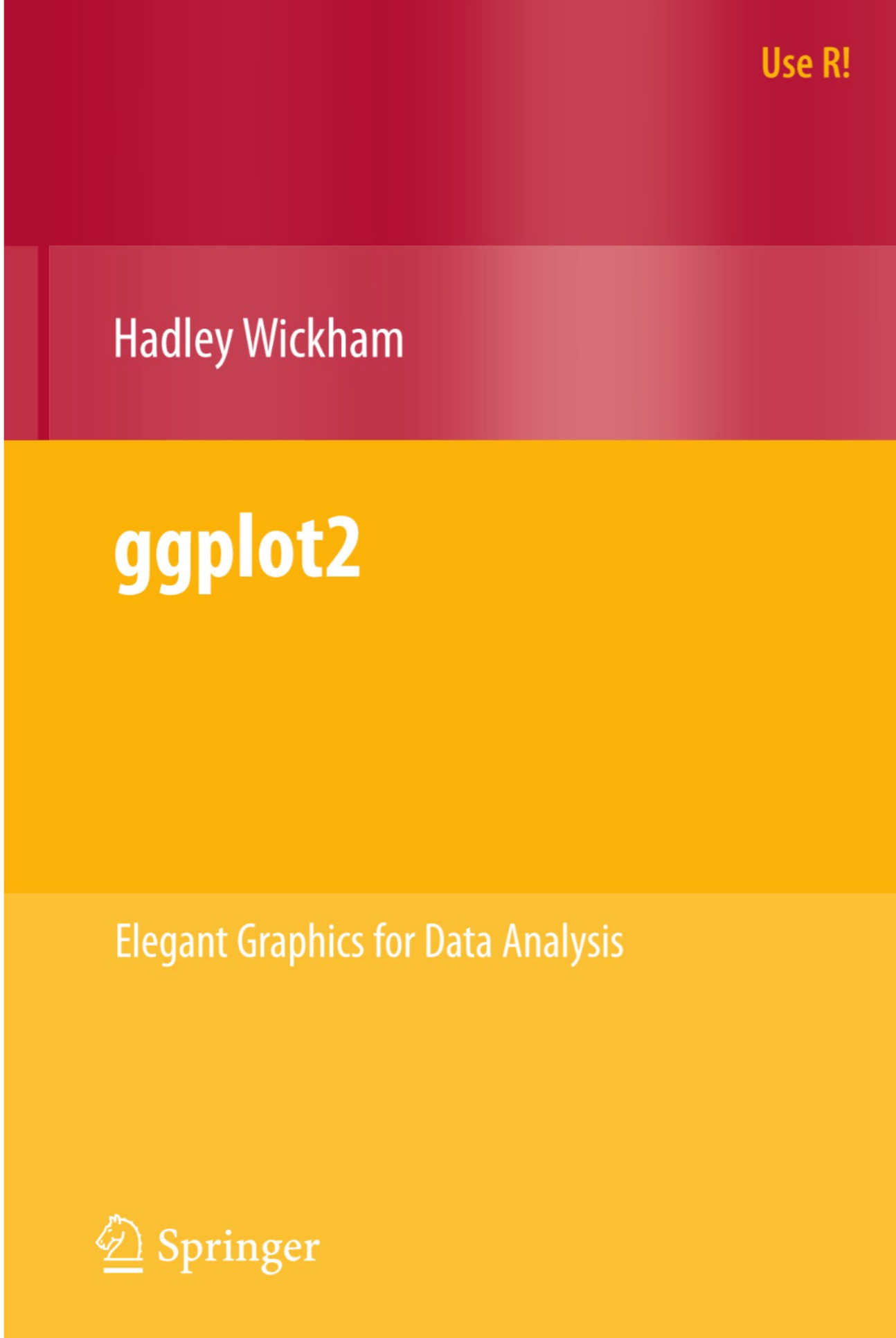
Learn tidyverse

- 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/

Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes
x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, ...) Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE) Count number of rows in each group defined by the variables in ... Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE) Returns copy of table grouped by ...
g_iris <- group_by(iris, Species)

ungroup(x, ...) Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, ...) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values. *distinct(iris, Species)*

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows. *sample_frac(iris, 0.5, replace = TRUE)*

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, ...) Select rows by position. *slice(iris, 10:15)*

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	| xor()
>	>=	!is.na()	!	&

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, ...) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, ...) Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)	**num_range**(prefix, range)	:, e.g. mpg:cyl
ends_with(match)	**one_of**(...)	-, e.g, -Species
matches(match)	**starts_with**(match)	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, ...) Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, ...) Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, ...) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, ...) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, ...) Rename columns.
rename(iris, Length = Sepal.Length)

RStudio

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , * , /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column() Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames() Move col in row names.
column_to_rownames(a, var = "C")

Combine Tables

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...) Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...) Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...) Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...) Join data. Retain all values, all rows.

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL) Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, ...) Rows that appear in both x and y.

setdiff(x, y, ...) Rows that appear in x but not y.

union(x, y, ...) Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, ...) Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...) Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

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
- **Combine code with narrative (R with markdown)**
- Turn your analyses into high quality documents, reports, presentations and dashboards.
- … or your note book.

Console, Scripts and Rmarkdown

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

Markdown

```
# title

Some text about your project

## Section

More text about your project

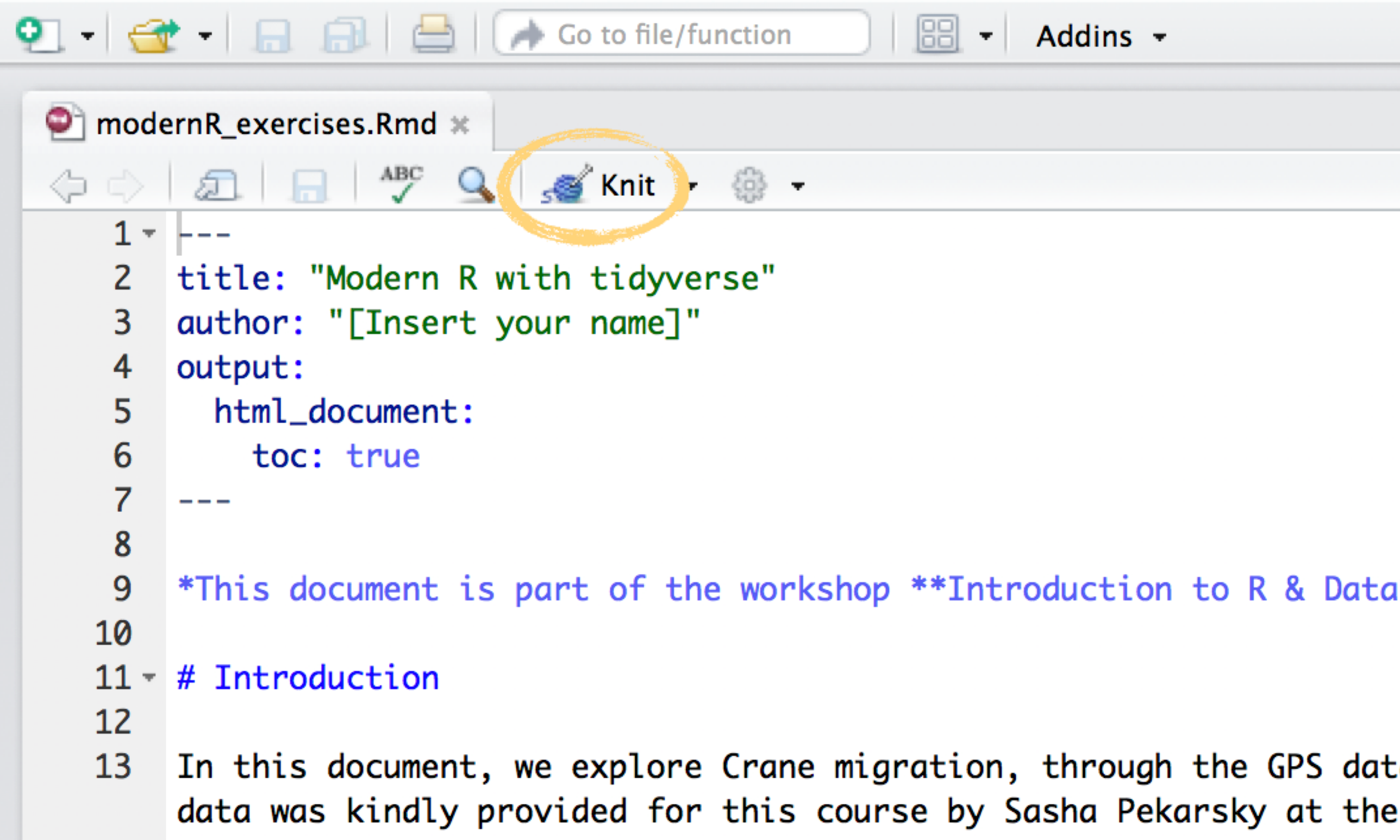
```{r}
max(iris$Petal.Length)
min(iris$Petal.Length)
```
```

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!

Run the first code chunk!

```
## Technical requirements

This project depends on the `tidyverse` package. Load tidyverse into your work
environment.

```{r}
library(tidyverse)
```
```

```
> library(tidyverse)
── Attaching packages ─────────────────────────────────── tidyverse 1.2.1 ──
   ggplot2 3.1.0       purrr   0.3.0
   tibble  2.0.1       dplyr   0.7.8
   tidyr   0.8.2       stringr 1.4.0
   readr   1.3.1       forcats 0.3.0
── Conflicts ────────────────────────────────────── tidyverse_conflicts() ──
   dplyr::filter() masks stats::filter()
   dplyr::lag()    masks stats::lag()
```

All done? Time to knit!



package: Tibble

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

- 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
7            4.6         3.4          1.4         0.3     setosa
8            5.0         3.4          1.5         0.2     setosa
9            4.4         2.9          1.4         0.2     setosa
10           4.9         3.1          1.5         0.1     setosa
11           5.4         3.7          1.5         0.2     setosa
12           4.8         3.4          1.6         0.2     setosa
13           4.8         3.0          1.4         0.1     setosa
14           4.3         3.0          1.1         0.1     setosa
15           5.8         4.0          1.2         0.2     setosa
16           5.7         4.4          1.5         0.4     setosa
```

Load and save data

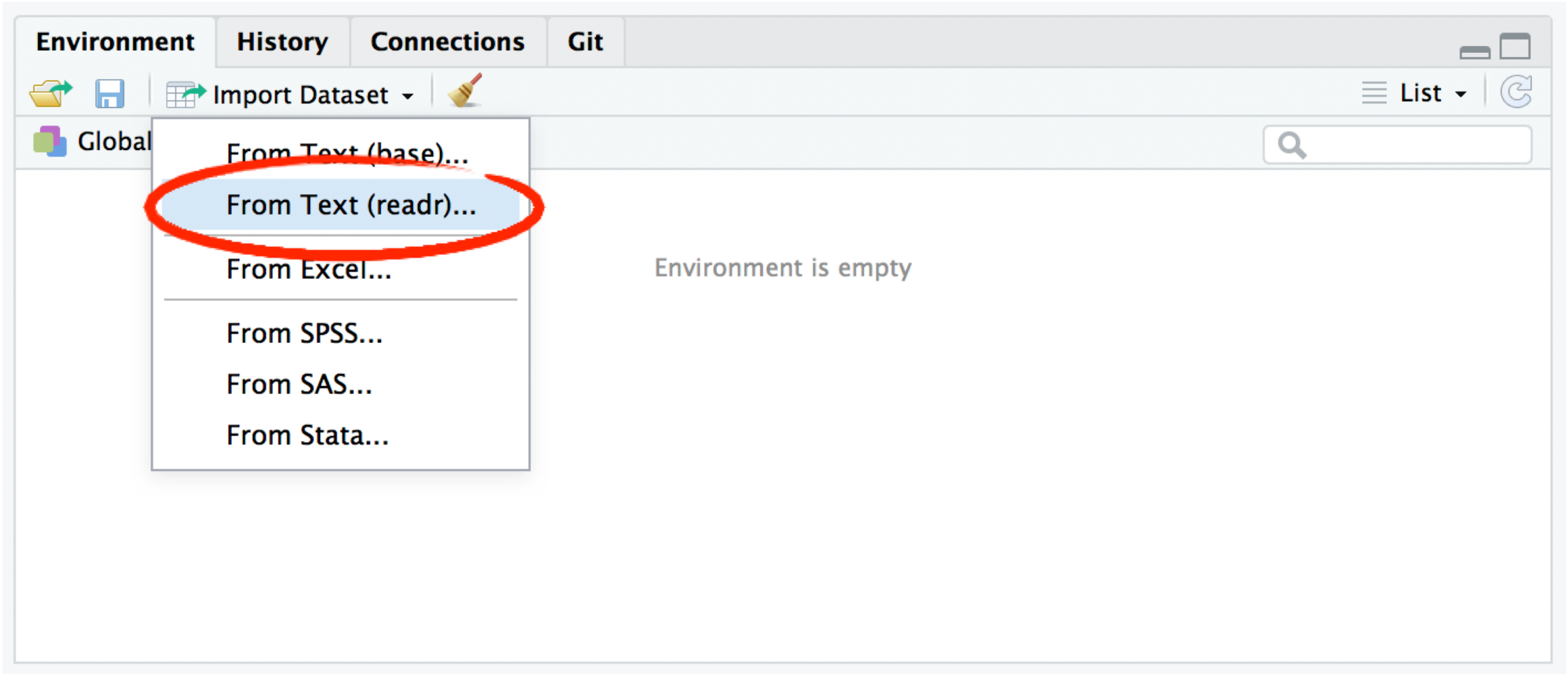
The basics of readr

Flat data files

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

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

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:

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()
)
```

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.

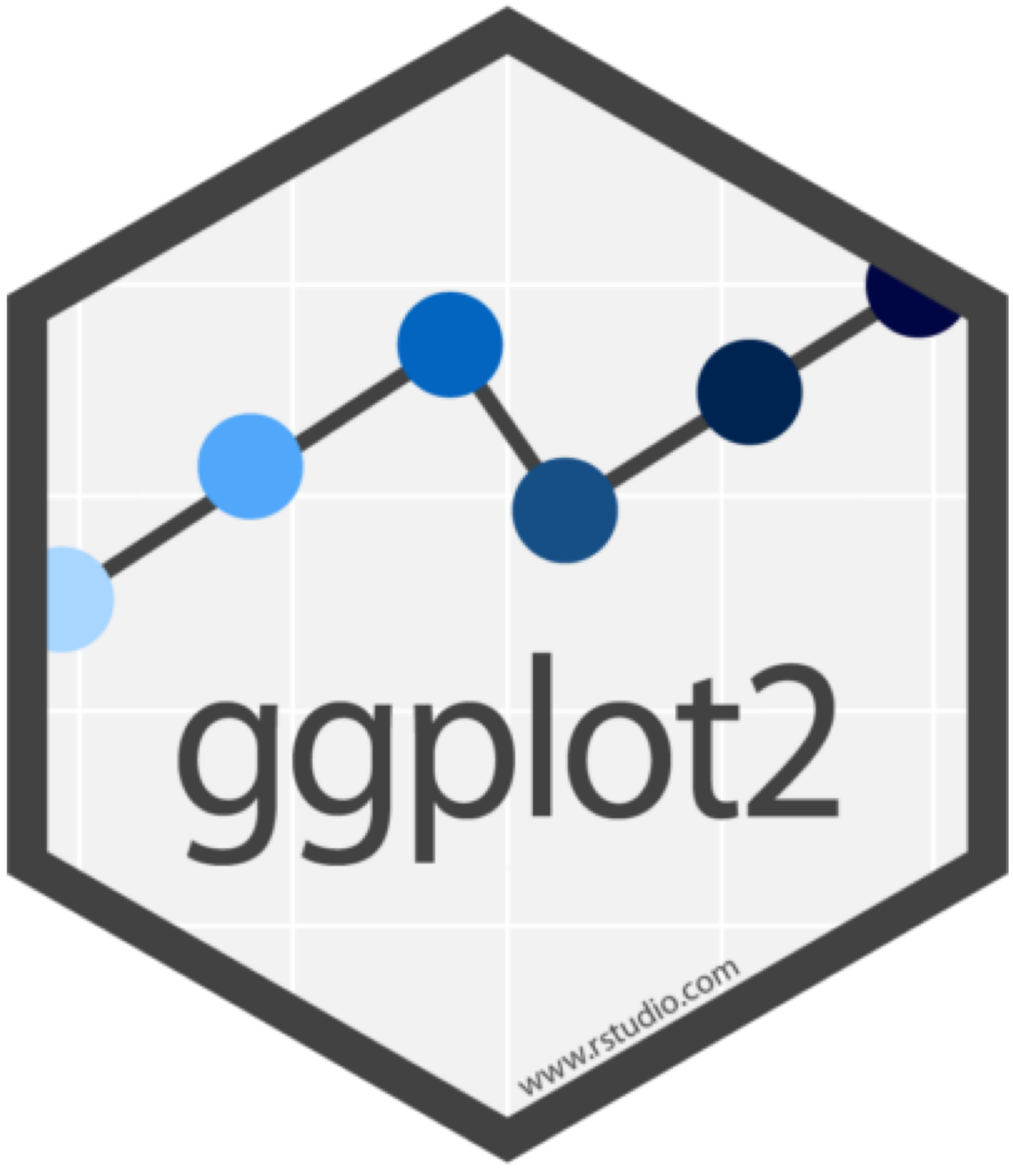
Exercises

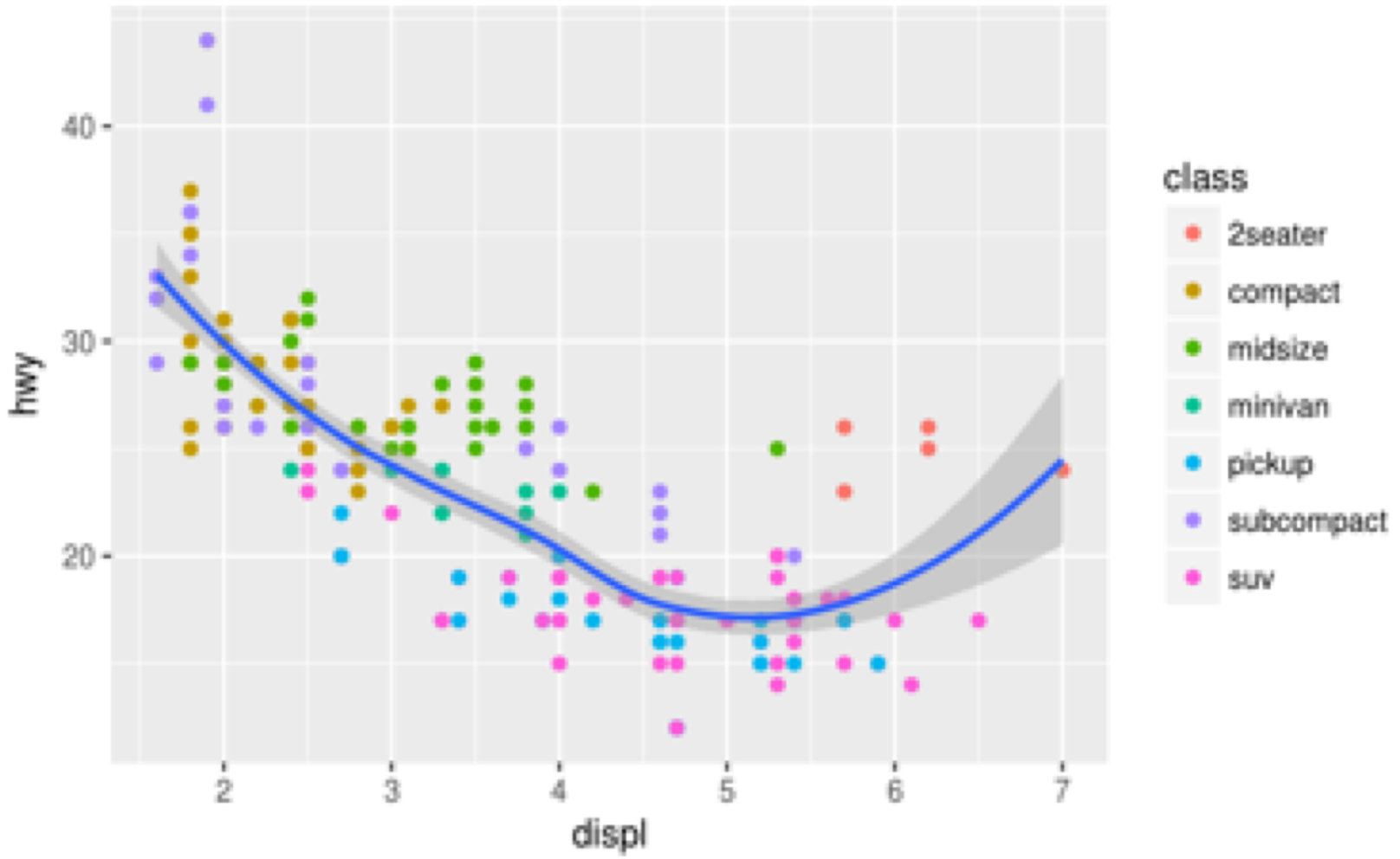
Please do basic exercises 1-I and 1-II.

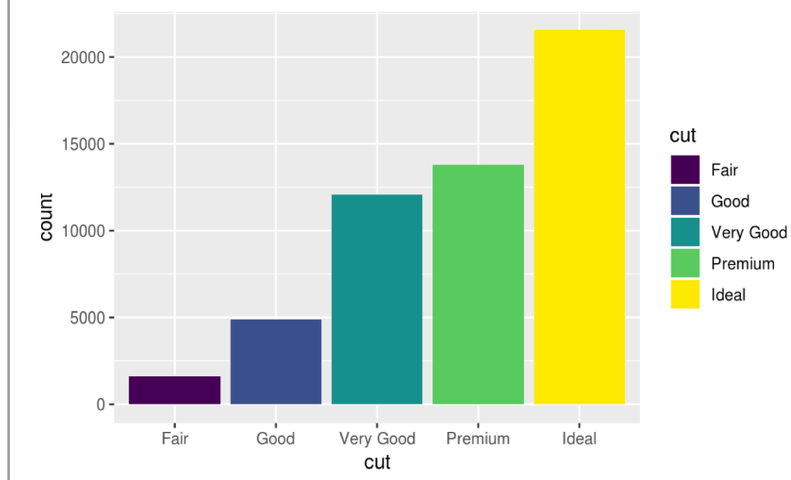
Time left? Opt for a reading exercise or optional exercise!

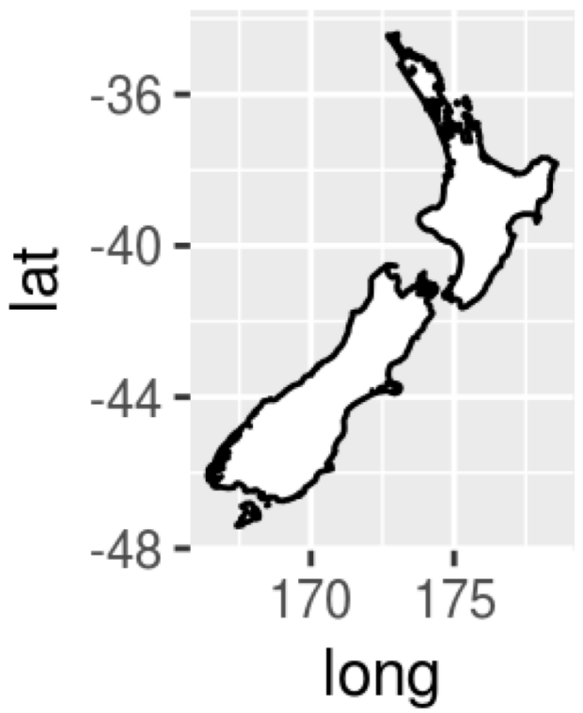
Data visualisation

Create eye candy with ggplot2







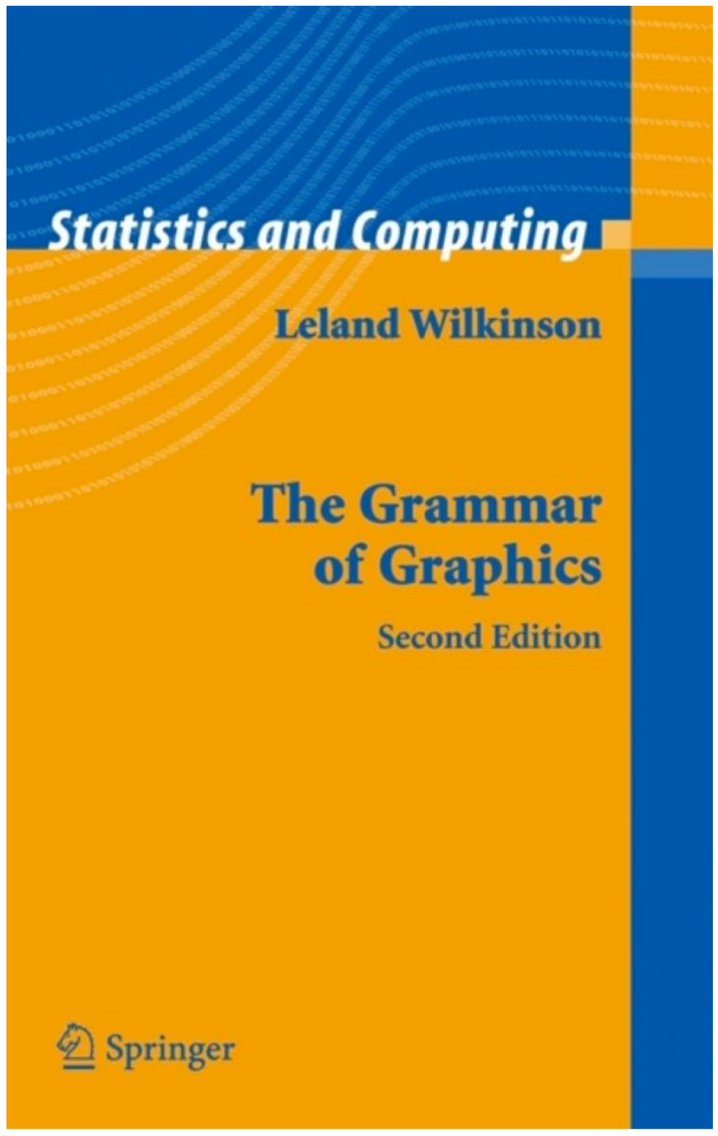


Examples of plots with R (and ggplot2)

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



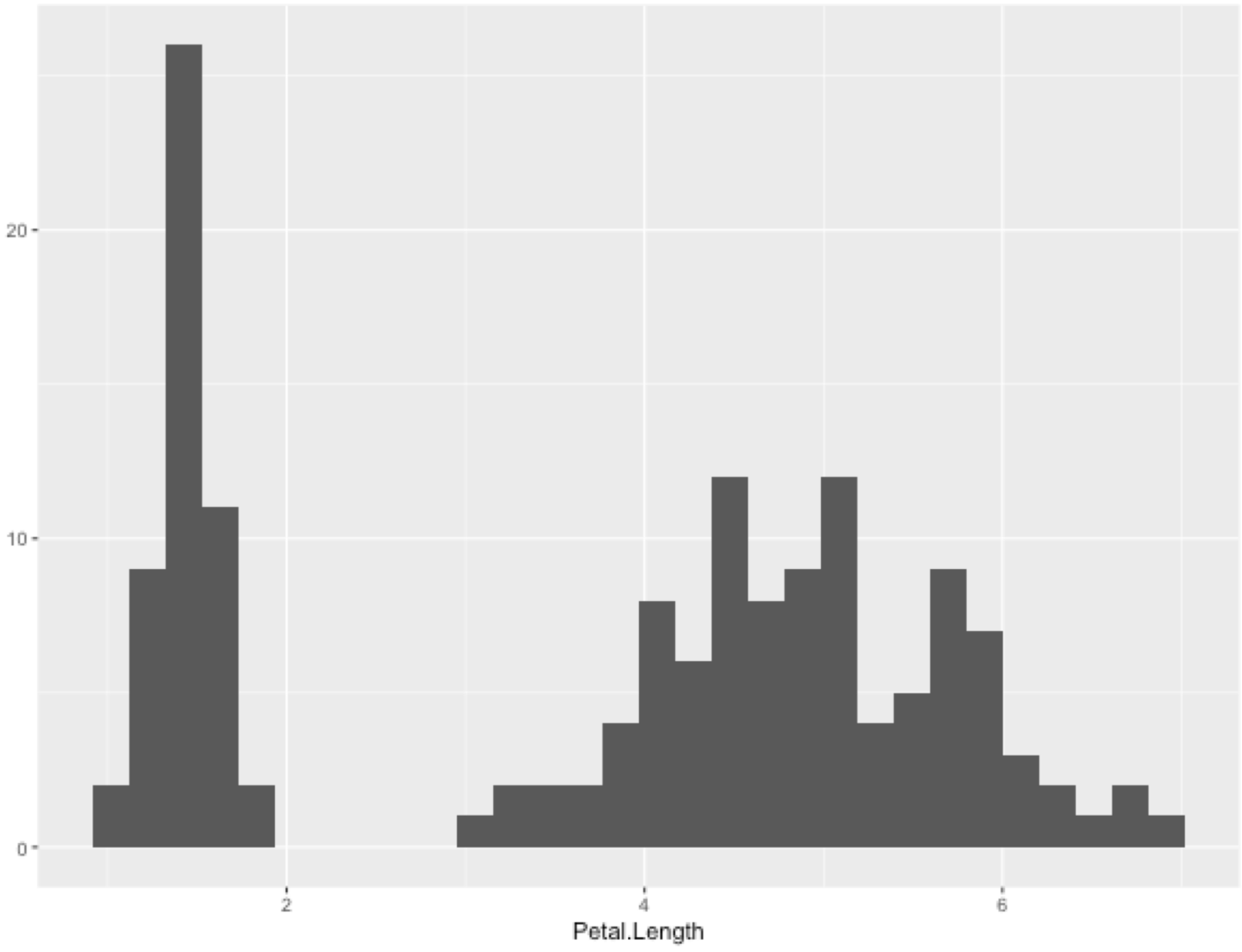
Data	The variables in a tibble or data.frame
Aesthetics	x- and y-axis, colour, size, alpha, shape
Geometries	point, line, bar, histogram

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

Quick plotting

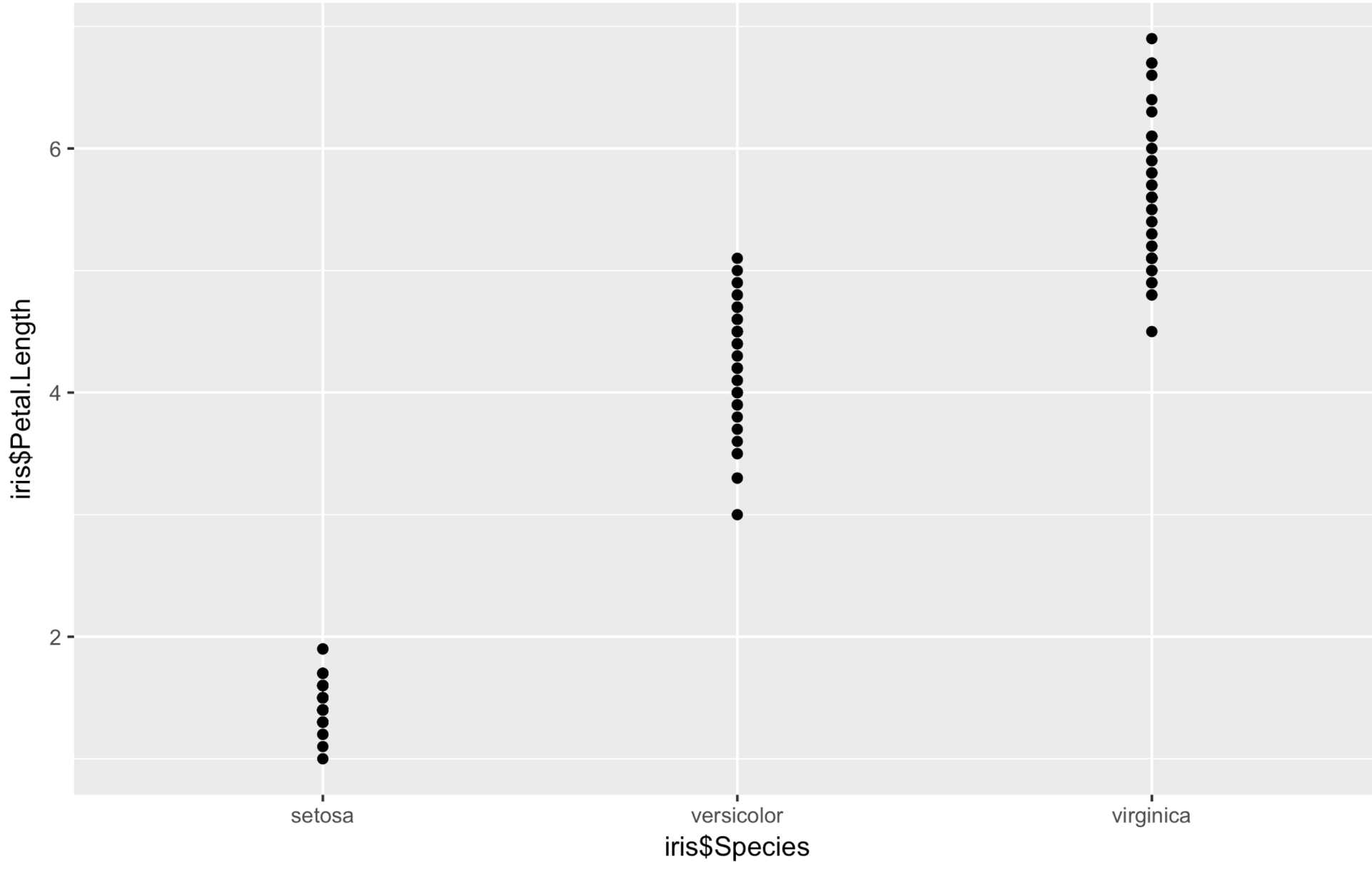
Quick plots

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



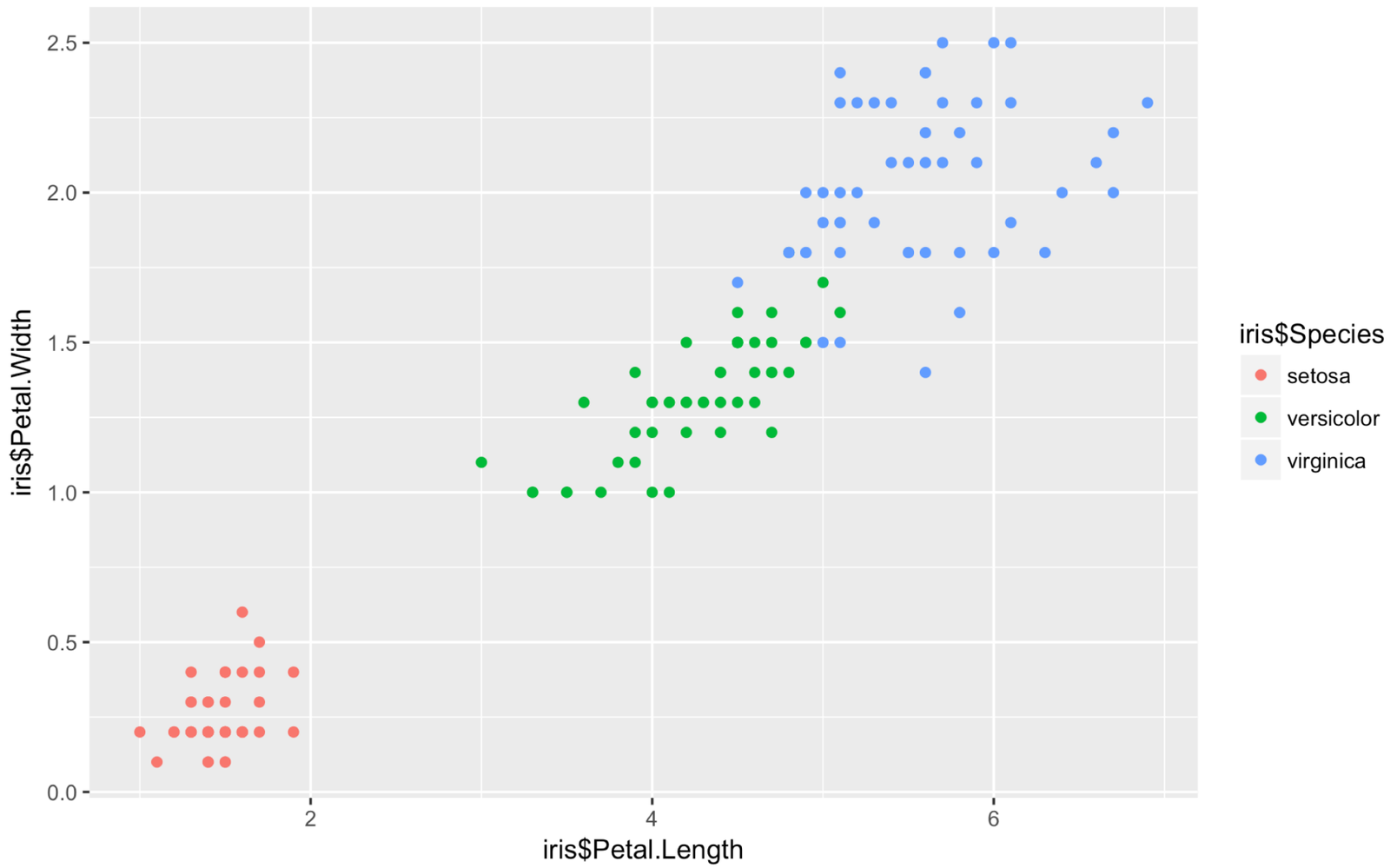
Quick plots

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



Quick plots

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



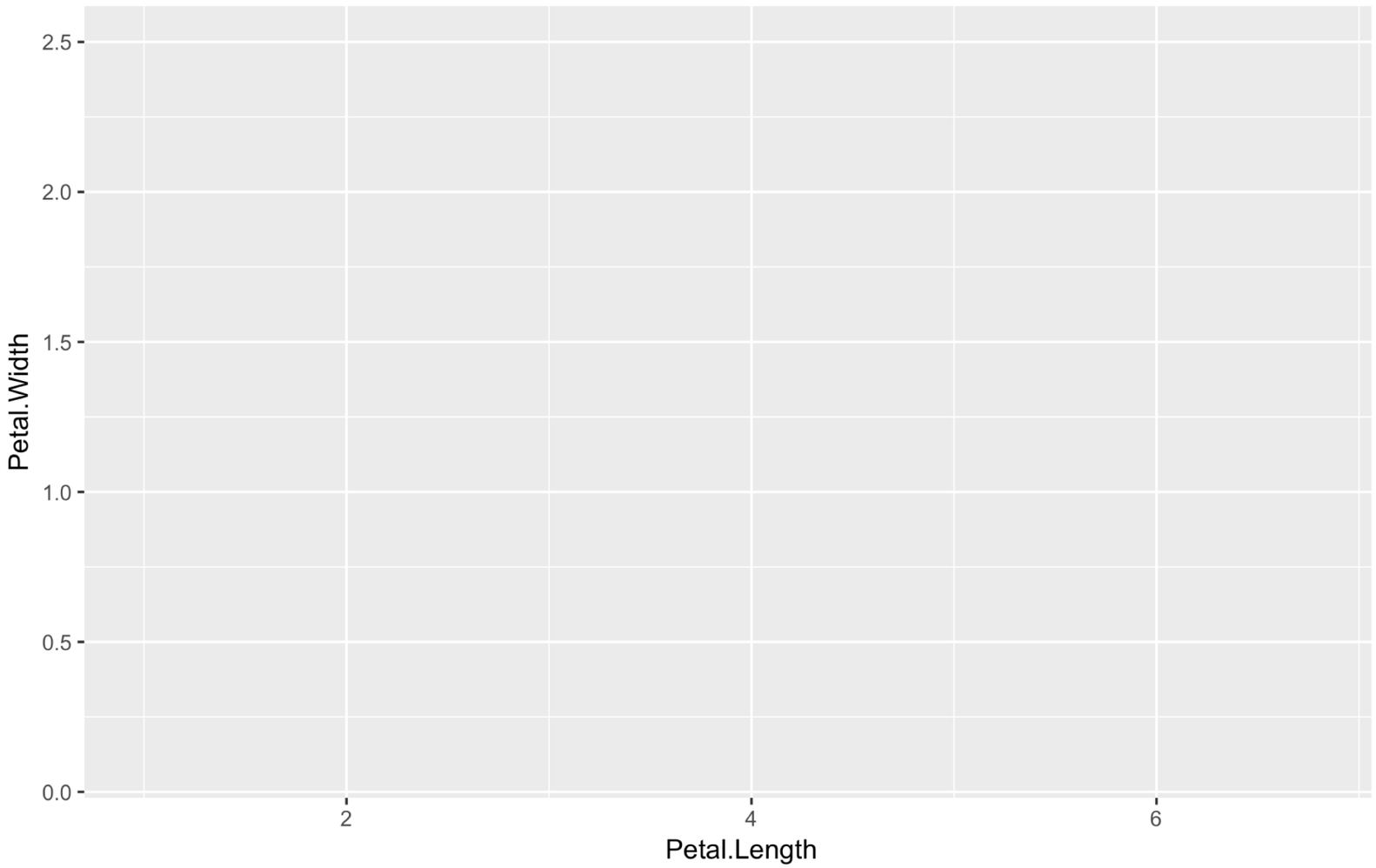
ggplot (grammar of graphics)

ggplot: Data, aesthetics, geometrics

```
> ggplot(iris)
```

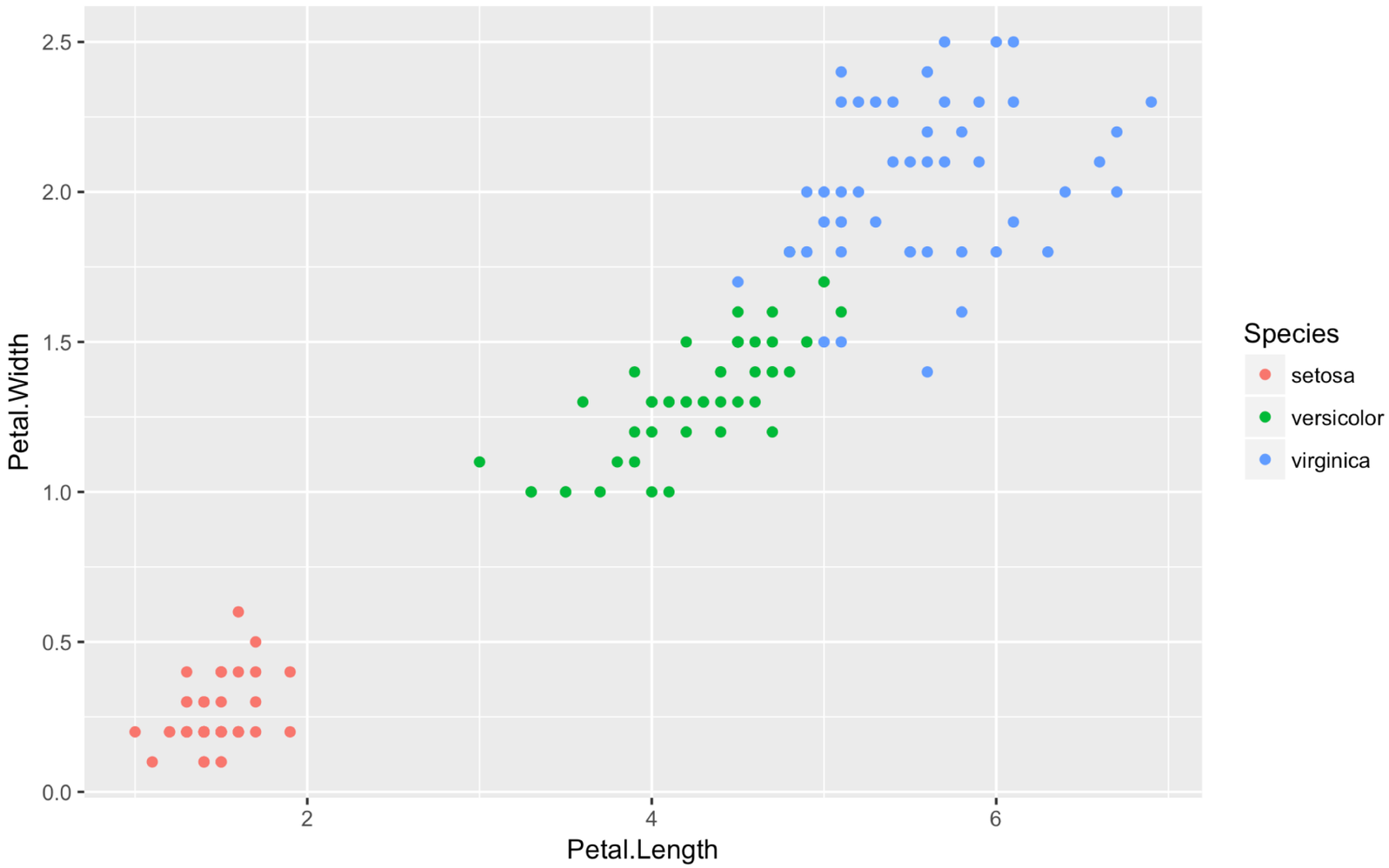
ggplot: Data, aesthetics, geometrics

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



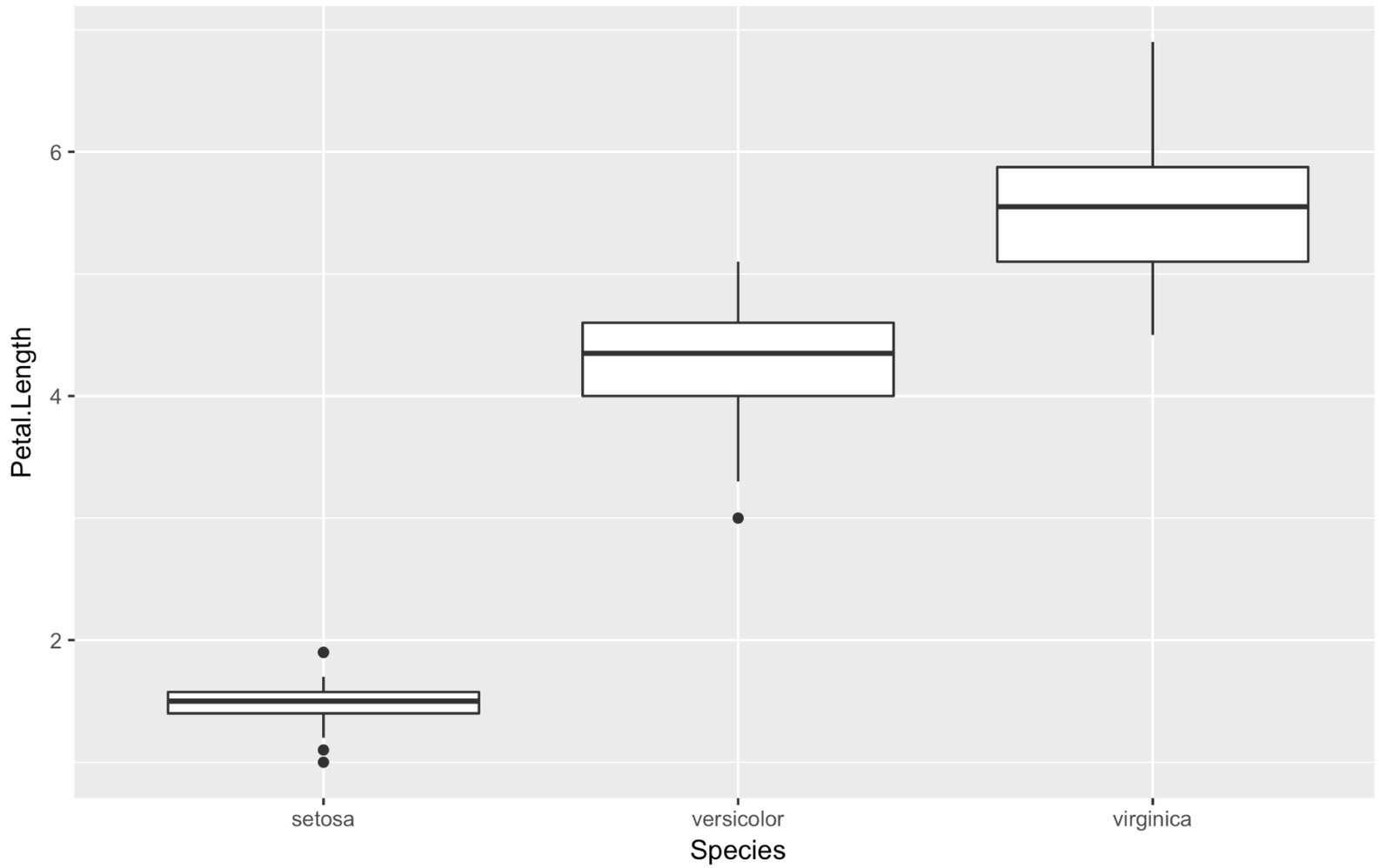
ggplot: Data, aesthetics, geometrics

```
> 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()
```



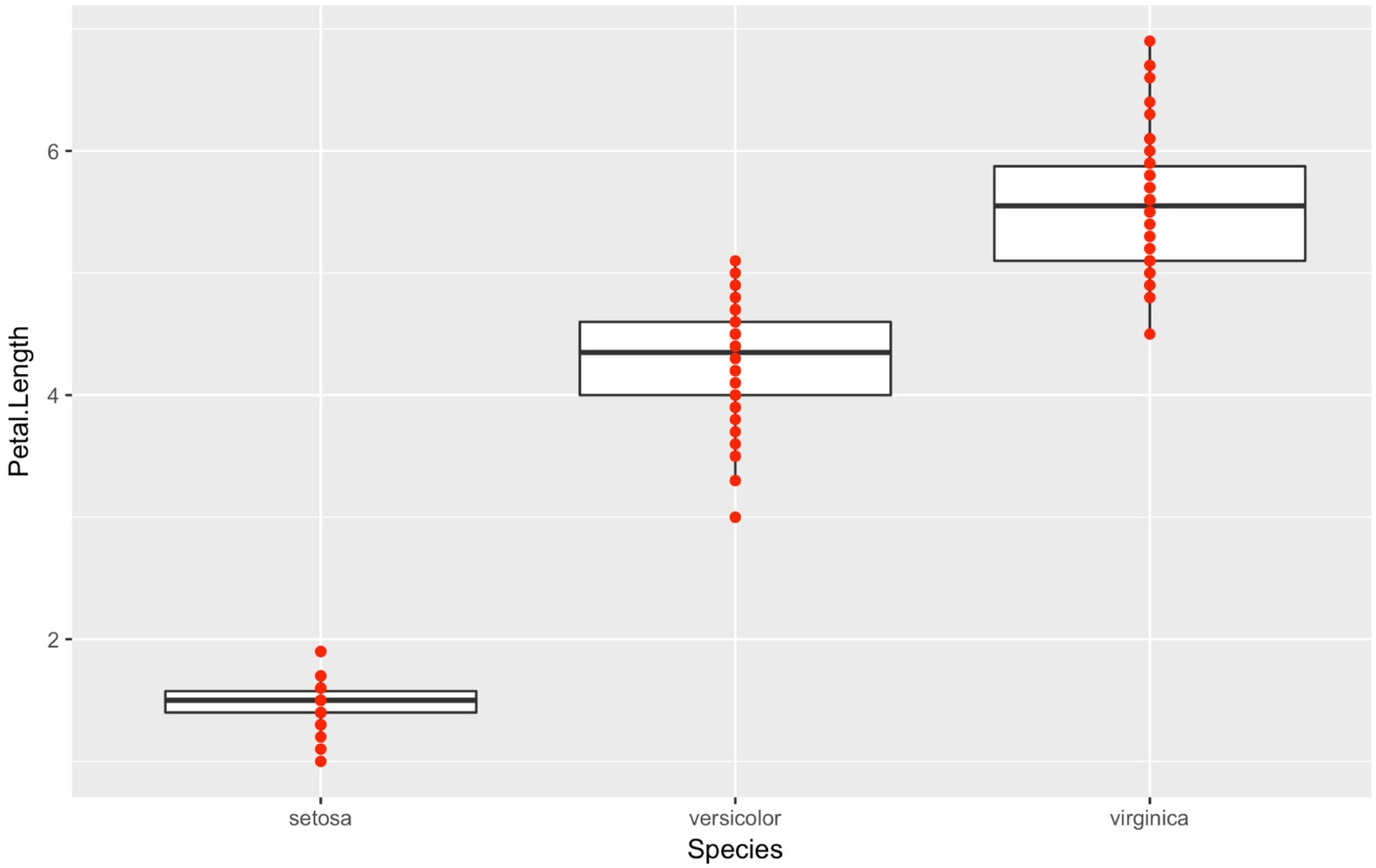
ggplot: Multiple geometric layers

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



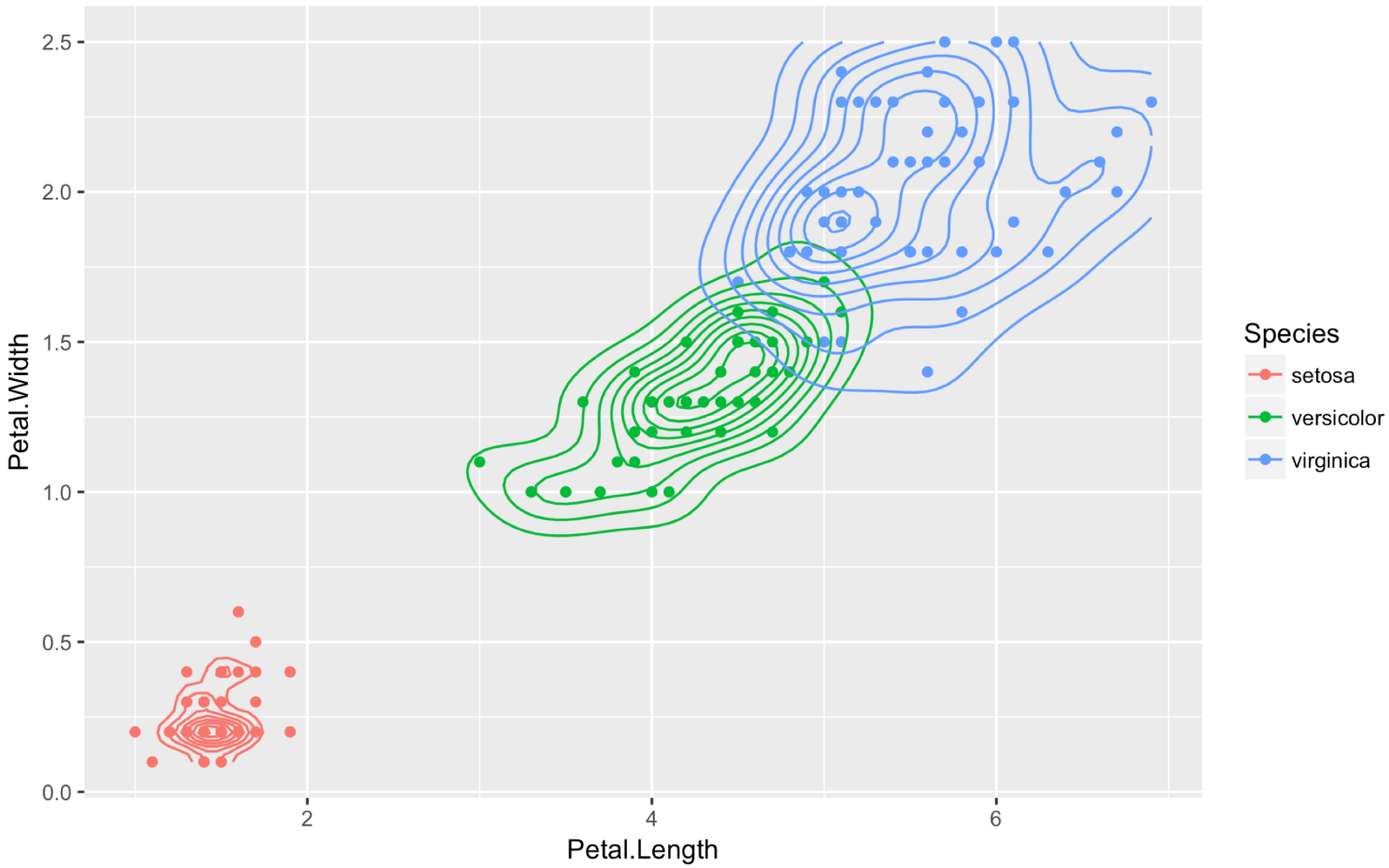
ggplot: Multiple geometric layers

```
> 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()
```



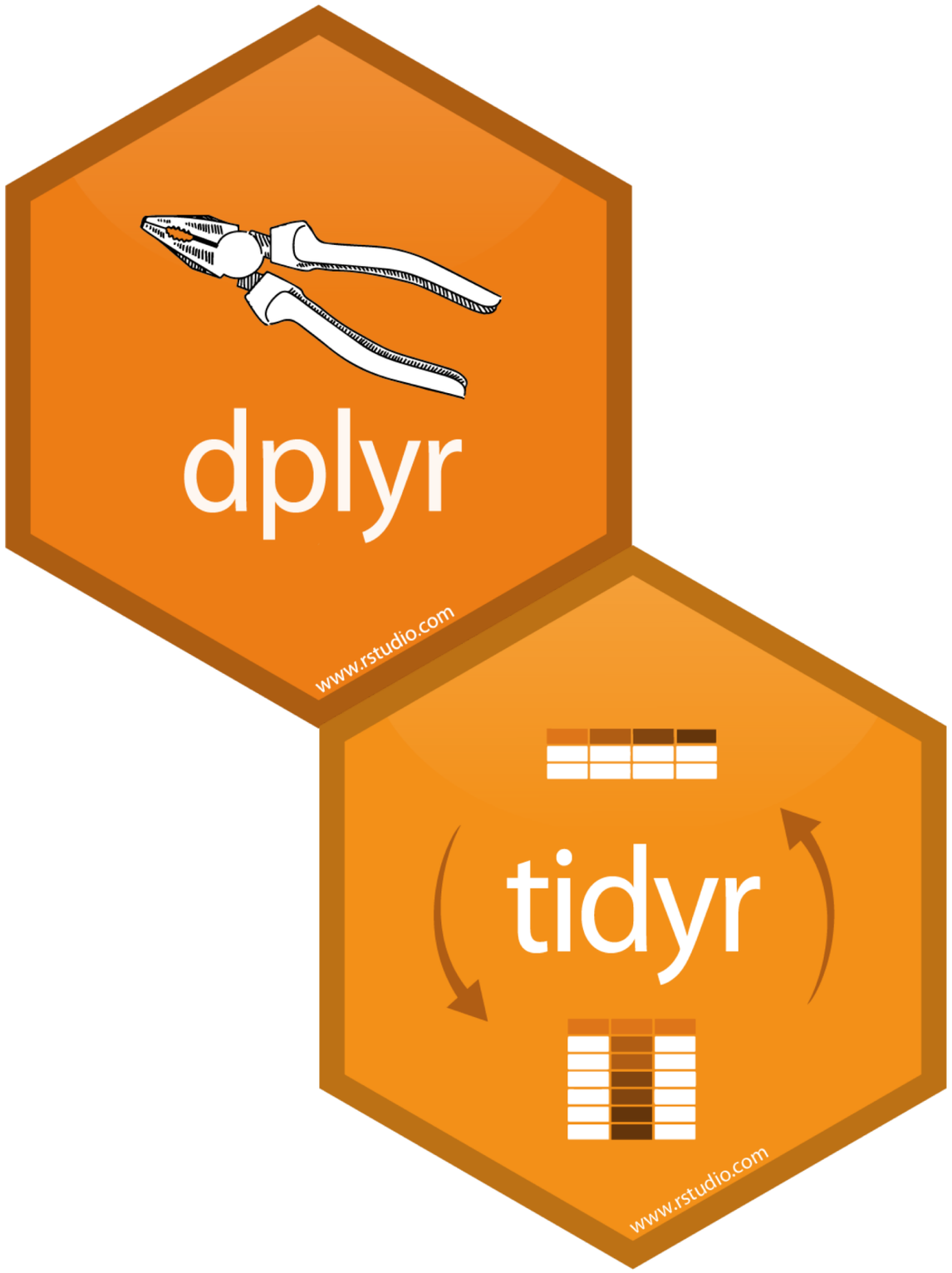
Exercises

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 Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

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

&

Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

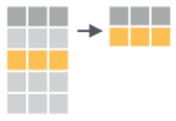
group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

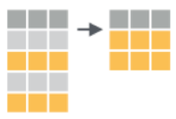
EXTRACT CASES

Row functions return a subset of rows as a new table.



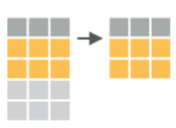
filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*



slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

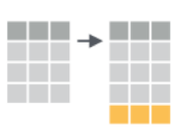
<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES



add_row(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

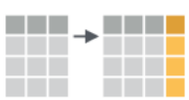
Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)
ends_with(match)
matches(match)
num_range(prefix, range)
one_of(…)
starts_with(match)
:, e.g. mpg:cyl
-, e.g, -Species

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

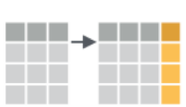


mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

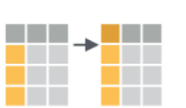
transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))



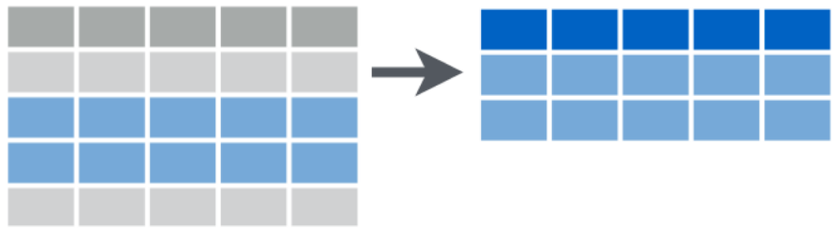
add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*



rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

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 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()

Make New Variables

Function: mutate()

```
> mutate(iris,
         petal_area = pi * Petal.Length/2 * Petal.Width/2)
```

Function: mutate()

```
> mutate(iris,
         petal_area = pi * Petal.Length/2 * Petal.Width/2)
# A tibble: 150 x 6
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species petal_area
          <dbl>       <dbl>        <dbl>       <dbl> <fct>        <dbl>
 1          5.1         3.5          1.4         0.2 setosa       0.220
 2          4.9         3            1.4         0.2 setosa       0.220
 3          4.7         3.2          1.3         0.2 setosa       0.204
 4          4.6         3.1          1.5         0.2 setosa       0.236
 5          5           3.6          1.4         0.2 setosa       0.220
 6          5.4         3.9          1.7         0.4 setosa       0.534
 7          4.6         3.4          1.4         0.3 setosa       0.330
 8          5           3.4          1.5         0.2 setosa       0.236
 9          4.4         2.9          1.4         0.2 setosa       0.220
10          4.9         3.1          1.5         0.1 setosa       0.118
# ... with 140 more rows
```

Function: mutate()

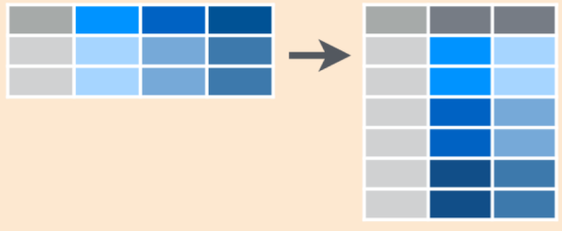
```
> mutate(iris,
         # compute the area of a single petal
         petal_area = pi * Petal.Length/2 * Petal.Width/2,
         # abbreviate the name of the species
         Species_abbr = substring(Species, 1, 3)
)
```

Function: mutate()

```
> mutate(iris,
         # compute the area of a single petal
         petal_area = pi * Petal.Length/2 * Petal.Width/2,
         # abbreviate the name of the species
         Species_abbr = substring(Species, 1, 3)
)
```

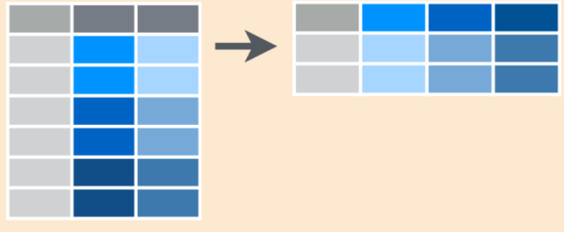
```
# A tibble: 150 x 7
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species petal_area Species_abbr
          <dbl>       <dbl>        <dbl>       <dbl> <fct>        <dbl> <chr>
 1          5.1         3.5          1.4         0.2 setosa       0.220 set
 2          4.9         3            1.4         0.2 setosa       0.220 set
 3          4.7         3.2          1.3         0.2 setosa       0.204 set
 4          4.6         3.1          1.5         0.2 setosa       0.236 set
 5          5           3.6          1.4         0.2 setosa       0.220 set
 6          5.4         3.9          1.7         0.4 setosa       0.534 set
 7          4.6         3.4          1.4         0.3 setosa       0.330 set
 8          5           3.4          1.5         0.2 setosa       0.236 set
 9          4.4         2.9          1.4         0.2 setosa       0.220 set
10          4.9         3.1          1.5         0.1 setosa       0.118 set
# ... with 140 more rows
```

gather() and spread()



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

Gather columns into rows.



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

Spread rows into columns.

Tidy data is long data

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

Patient	Temp	Med_A	Temp_A	Med_B	Temp_B
122030	37.0	300	37.1	NA	NA
122021	38.2	200	38.1	85	36.5
124500	38.1	300	38.0	NA	NA
126098	39.1	NA	NA	100	36.8

Wide and long data

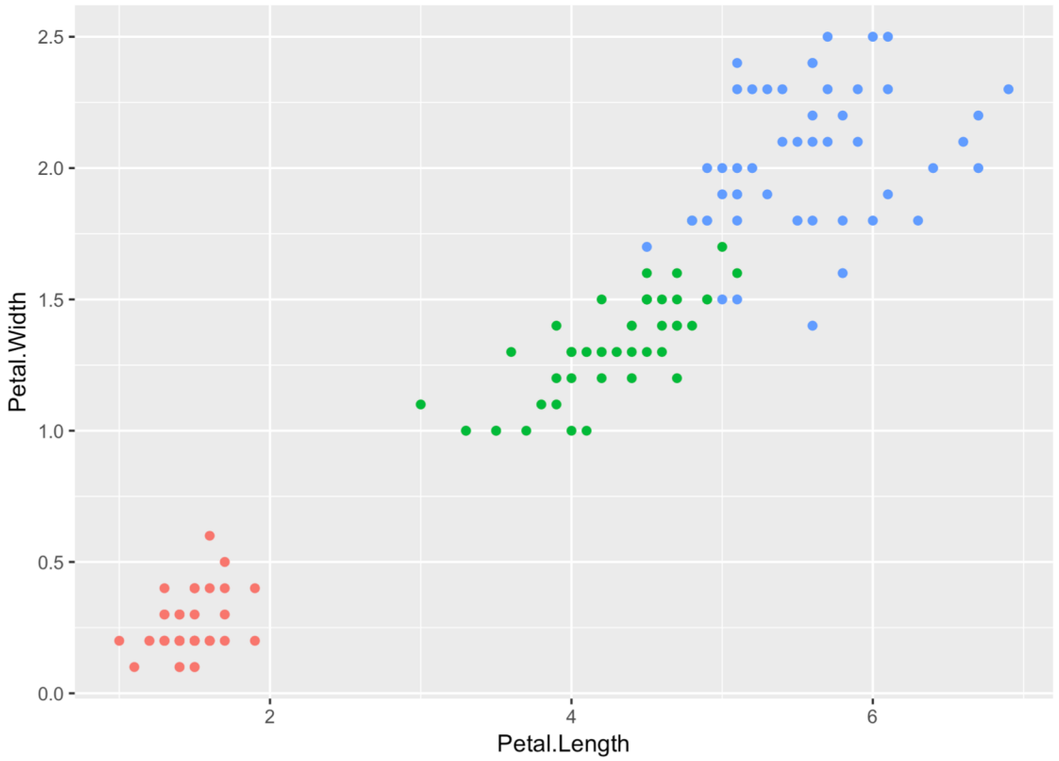
```
# A tibble: 150 x 6
   Species observation Petal.Length Petal.Width Sepal.Length Sepal.Width
   <chr>         <int>        <dbl>       <dbl>        <dbl>       <dbl>
 1 setosa            1         1.40       0.200         5.10        3.50
 2 setosa            2         1.40       0.200         4.90        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
```

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

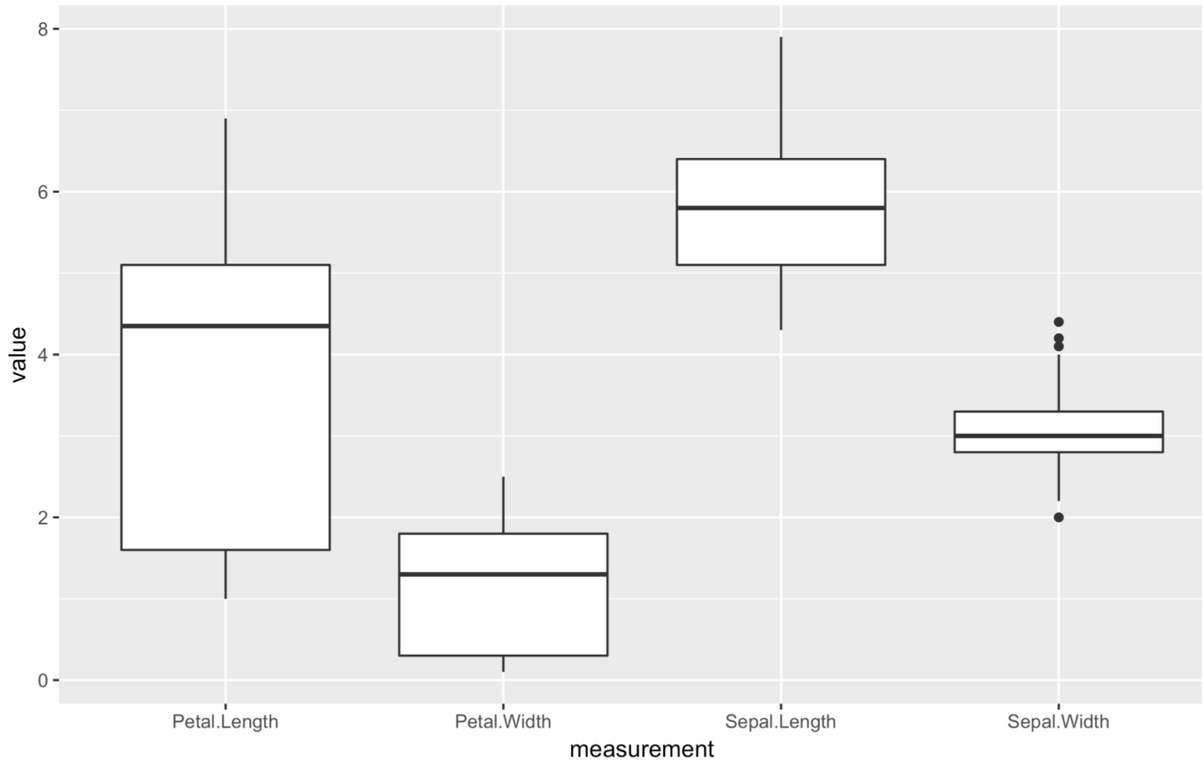
```
# A tibble: 600 x 4
   Species observation measurement  value
   <chr>         <int> <chr>        <dbl>
 1 setosa            1 Sepal.Length 5.10
 2 setosa            1 Sepal.Width  3.50
 3 setosa            1 Petal.Length 1.40
 4 setosa            1 Petal.Width  0.200
 5 setosa            2 Sepal.Length 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 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
# A tibble: 150 x 6
```

headers

	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
```

content

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
10 setosa            3 Sepal.Width   3.20
# ... with 590 more rows
```

headers

content

Function: spread()

column to headers

column to be split as **content**

```
> 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
```

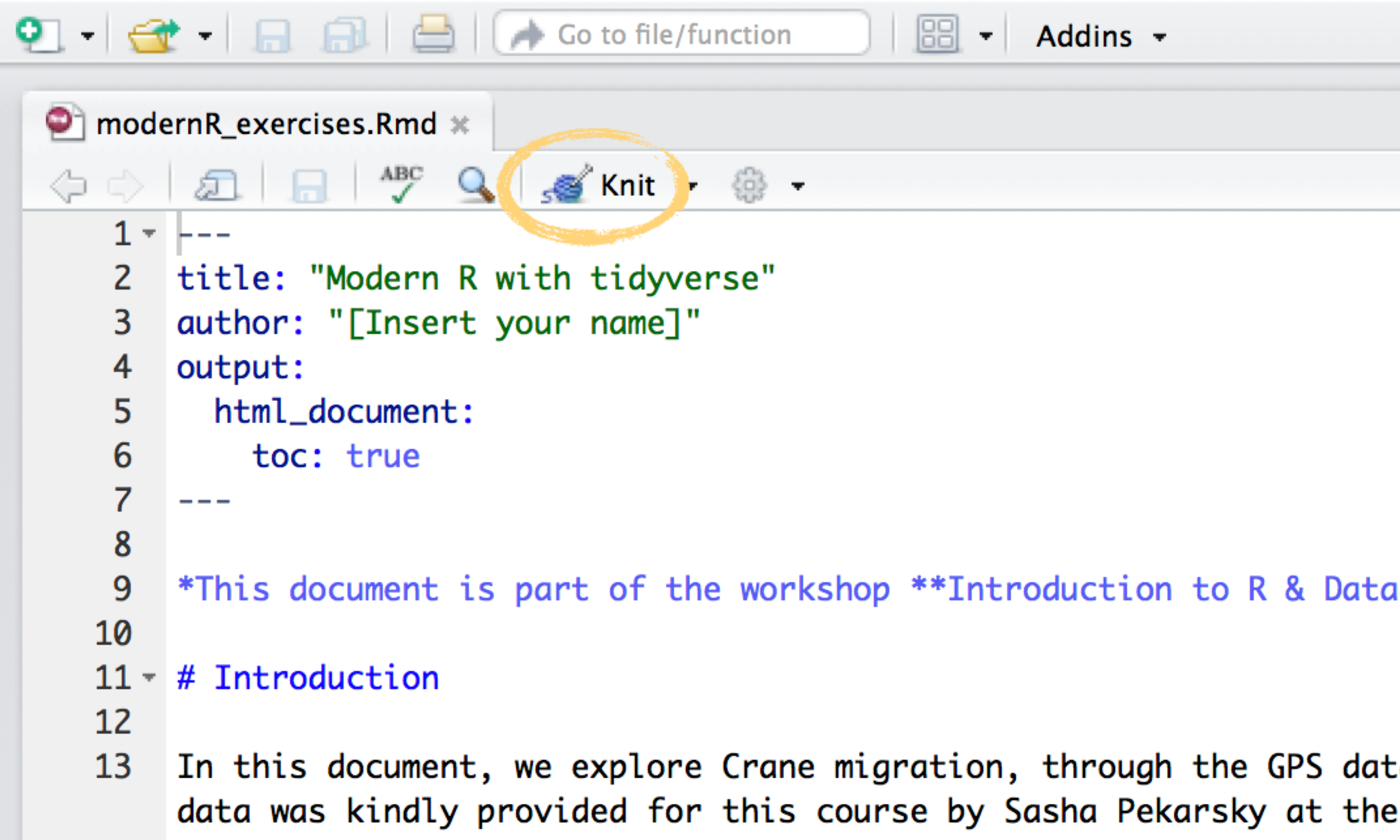
necessary for grouping!

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!



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)
- Quickstart to Research Data Management

R Cafe

Monthly event for R programmers at UU/UMCU. Be part of the R community! Work on your own project, and ask (or answer!) questions.

More info on:
github.com/UtrechtUniversity/R-data-cafe

(Or subscribe to the newsletter, check the RDM support website…)

Mondays, 15-17, Bucheliuszaal (6.18) UB

Next edition: **Monday 15/4, 15:00-17:00**

We love feedback! It's how we learn…

Please send your feedback, remarks, questions to Barbara: b.m.i.vreede@uu.nl

Or generate an issue on github!

tinyurl.com/introRData

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