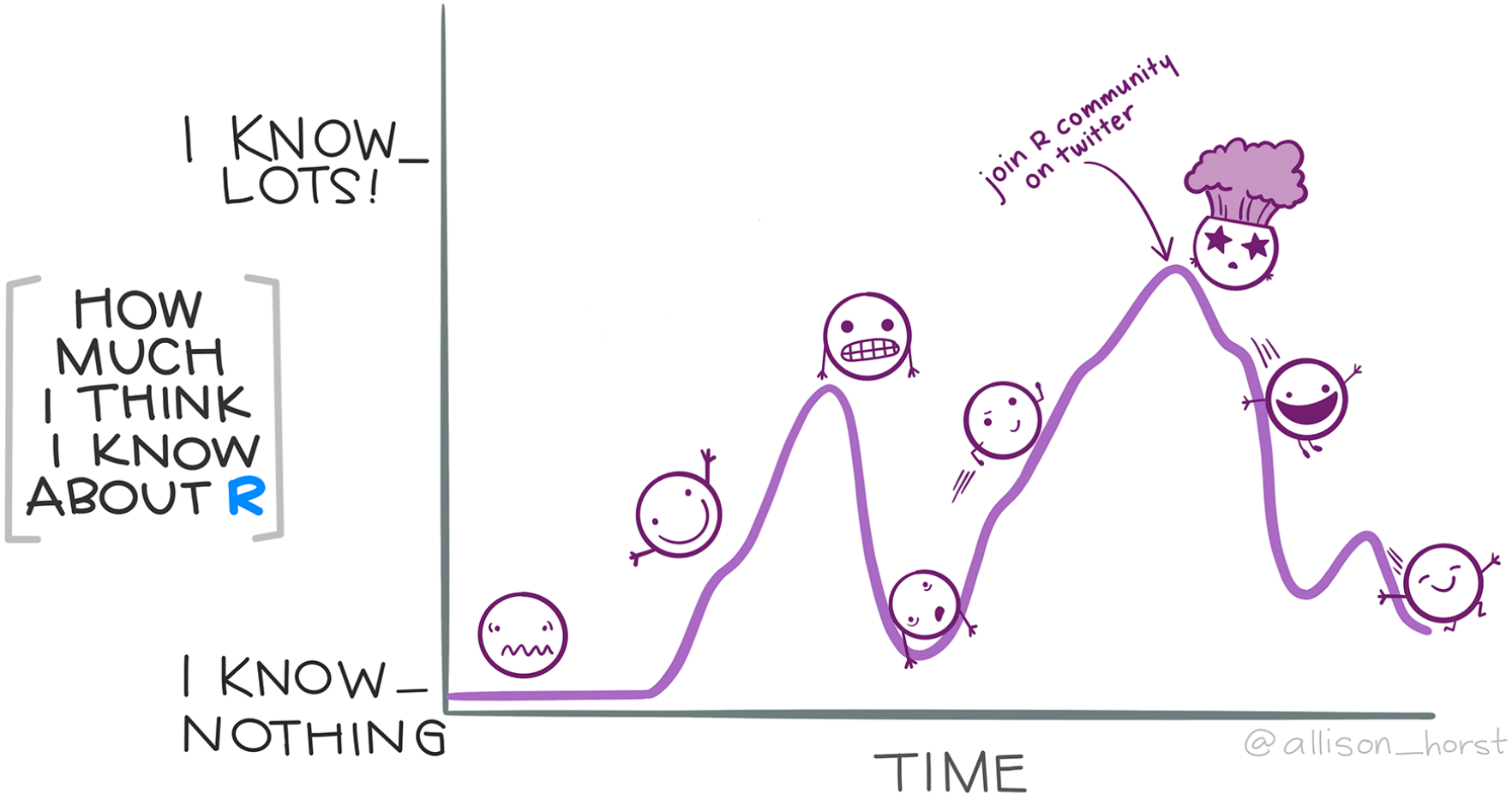
R basics and data summaries

Emile Latour, Nicky Wakim, Meike Niederhausen

January 12, 2026

Coding in R



Artwork by @allison_horst

We have options where to code in R

Console

- For quick experiments
- Temporary (not saved)

R scripts (.R files)

- Often used for data cleaning, functions, or prep
- Common in real analysis workflows
- Code only

Code chunks in a .qmd file

- Used for analysis and reporting
- Combine **code and written explanation**
- Saved and reproducible

Running code

- Run the current line or selection with `Cmd/Ctrl + Enter`

Coding along (optional)

If you want to follow along during class:

- Open a **new Quarto document** (`.qmd`)
- This is how homework and exams will be completed

Insert a code chunk: - Mac: `Cmd + Option + I` - Windows: `Ctrl + Alt + I`

You can run code with `Cmd/Ctrl + Enter`.

R as a calculator

- Rules for order of operations are followed
- Spaces between numbers and characters are ignored

```
1 10^2
```

```
[1] 100
```

```
1 3 ^ 7
```

```
[1] 2187
```

```
1 6/9
```

```
[1] 0.6666667
```

```
1 9-43
```

```
[1] -34
```

```
1 4^3-2* 7+9 /2
```

```
[1] 54.5
```

The equation above is computed as

$$4^3 - (2 \cdot 7) + \frac{9}{2}$$

Variables (objects) in R

Variables let us **store results** so we can reuse them later (e.g., data, summaries, plots, model output).

Assigning a single value

- Use <– to assign a value to a name
- Read as “gets”, “becomes”, or “assigns”

```
x <- 5
x
```

```
[1] 5
```

You’ll mostly see <– in examples and documentation, so that’s what we’ll use in this course. But sometimes people use =.

```
x = 5
x
```

```
[1] 5
```

Assigning multiple values (vectors)

Sequences

```
a <- 3:10
a
```

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

Combining values with c( )

```
b <- c(5, 12, 2, 100, 8)
b
```

```
[1]   5  12   2 100   8
```

Overwriting objects

In R, you can overwrite an object by assigning a new value to the same name.

```
1 x <- 5
2 x
```

```
[1] 5
```

```
1 x <- 10
2 x
```

```
[1] 10
```

- R does not warn you when this happens.
- The old value is replaced.
- This is normal and common in R.

Comments in R

Comments are notes to yourself (and others).

R ignores anything after #.

```
# This is a comment
x <- 5   # assign 5 to x
x
```

```
[1] 5
```

- Comments can explain *what* your code is doing.
- Even better, comments can explain *why* you chose to do something.

```
# Example: WHAT the code is doing
# Adding two numbers and storing the result
sum_result <- 2 + 3

# Example: WHY
# Creating a constant that will be reused later in the analysis
total_score <- 2 + 3
```

- For this class, comments can be helpful in homework and exams
- They let us see your thought process or where you might have been unsure

Quick practice (30-60 seconds)

In your `.qmd`, create:

- `y <- 8`
- `c_vec <- 15:20`
- `d_vec <- c(16:19, 22)`

(Do not overthink it - just try it.)

Quick practice: solution

```
y <- 8
y
```

```
[1] 8
```

```
c_vec <- 15:20
c_vec
```

```
[1] 15 16 17 18 19 20
```

```
d_vec <- c(16:19, 22)
d_vec
```

```
[1] 16 17 18 19 22
```

Using functions

- A **function** performs a task (e.g., calculate a mean)
- Functions use parentheses: `function_name()`
- Inputs to a function are called **arguments**
- Use `?function_name` to see the help file

Arguments specified by name

```
1 mean(x = 1:4)
```

```
[1] 2.5
```

```
1 seq(from = 1, to = 12, by = 3)
```

```
[1]  1  4  7 10
```

```
1 seq(by = 3, to = 12, from = 1)
```

```
[1]  1  4  7 10
```

Arguments specified by position

```
1 mean(1:4)
```

```
[1] 2.5
```

```
1 seq(1, 12, 3)
```

```
[1]  1  4  7 10
```

Using functions for summary statistics

Now let's apply functions to the vectors you just created.

- Calculate the **mean** of `c_vec`
- Calculate the **standard deviation** of `c_vec`
- Use `?sd` to see what arguments `sd()` takes

```
1 mean(c_vec)
```

```
[1] 17.5
```

```
1 sd(c_vec)
```

```
[1] 1.870829
```

If you have time, also try:

```
1 median(c_vec)
```

```
[1] 17.5
```

```
1 IQR(c_vec)
```

```
[1] 2.5
```

Quick check

- Do these functions return **one number or many**?
- Does that match what you expect a “summary” to do?

Working with a real dataset

- R comes with several built-in datasets that we can use to practice.
- We'll start with the `iris` dataset.
  - This dataset is useful because it includes both numeric and categorical variables.

Fisher's (or Anderson's) Iris data set

Data description:

- n = 150
- 3 species of Iris flowers (Setosa, Virginica, and Versicolour)
  - 50 measurements of each type of Iris
- **Variables**:
  - sepal length, sepal width, petal length, petal width, and species

Gareth Duffy

Load and view the data

The `iris` dataset is already available in base R.

```
1 data(iris)
```

Viewing in the console

You can display the dataset by typing its name:

```
1 iris
```

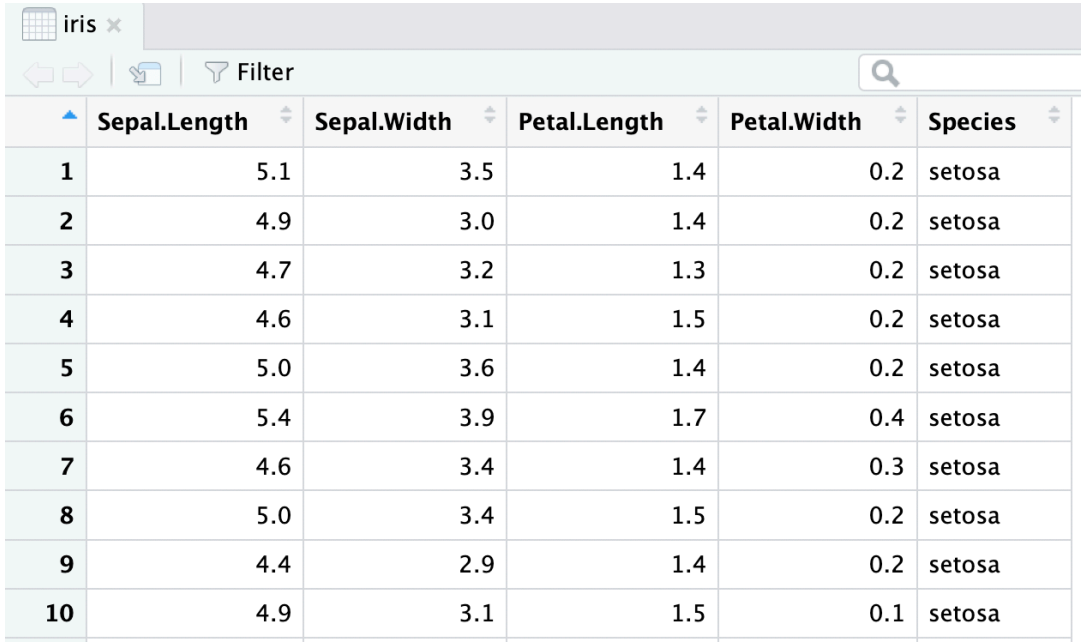
```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
4            4.6         3.1          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
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
```

Interactive view

You can open an interactive view of the dataset like this:

```
1 View(iris)
```

(Note: `View()` works interactively in RStudio, but does not display in slides.)



	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

Inspecting the dataset

Before summarizing or plotting, it helps to understand what's in the dataset.

```r
dim(iris)
names(iris)
str(iris)
head(iris)
tail(iris)
```

These functions help answer questions like:

- How many rows and columns are there?
- What are the variable names?
- Which variables are numeric vs categorical?

Inspecting the dataset: key questions

How many rows and columns are there?

```r
dim(iris)
```

```
[1] 150   5
```

What are the variable names?

```r
names(iris)
```

```
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

Which variables are numeric vs categorical?

```r
str(iris)
```

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Viewing rows of a dataset

When a dataset has many rows, it helps to look at just a few.

```
1 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
```

```
1 tail(iris)
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
145          6.7         3.3          5.7         2.5 virginica
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

You can also specify how many rows to view:

```
1 head(iris, 3)
```

```
  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
```

```
1 tail(iris, 2)
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

Accessing a single variable with $

To work with one column at a time, use the `$` operator.

```
1 iris$Petal.Length
```

```
  [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2 1.5 1.3 1.4
 [19] 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4 1.5 1.2
 [37] 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4 4.7 4.5 4.9 4.0
 [55] 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.0 4.7 3.6 4.4 4.5 4.1 4.5 3.9 4.8 4.0
 [73] 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.1 4.5 4.5 4.7 4.4 4.1 4.0
 [91] 4.4 4.6 4.0 3.3 4.2 4.2 4.2 4.3 3.0 4.1 6.0 5.1 5.9 5.6 5.8 6.6 4.5 6.3
[109] 5.8 6.1 5.1 5.3 5.5 5.0 5.1 5.3 5.5 6.7 6.9 5.0 5.7 4.9 6.7 4.9 5.7 6.0
[127] 4.8 4.9 5.6 5.8 6.1 6.4 5.6 5.1 5.6 6.1 5.6 5.5 4.8 5.4 5.6 5.1 5.1 5.9
[145] 5.7 5.2 5.0 5.2 5.4 5.1
```

`dataset$variable` pulls out a single column from a data frame.

Summarizing one numeric variable

Once you've selected a single numeric variable, you can apply summary functions to it.

```r
mean(iris$Petal.Length)
```

```
[1] 3.758
```

```r
sd(iris$Petal.Length)
```

```
[1] 1.765298
```

```r
median(iris$Petal.Length)
```

```
[1] 4.35
```

```r
IQR(iris$Petal.Length)
```

```
[1] 3.5
```

- Mean / SD describe center and spread
- Median / IQR are more robust to outliers

Quick summaries for all variables

We can also get a fast overview of the entire dataset at once.

```
1 summary(iris)
```

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
Median :5.800   Median :3.000   Median :4.350   Median :1.300
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
       Species
setosa    :50
versicolor:50
virginica :50
```

`summary()` provides basic summaries for every column, based on its type.

Missing values (NA) in R

Sometimes datasets contain missing values, represented as NA.

By default, summary functions return NA if any values are missing.

```
x <- c(2, 5, NA, 7)
mean(x)
```

```
[1] NA
```

To ignore missing values, use `na.rm = TRUE`.

```
mean(x, na.rm = TRUE)
```

```
[1] 4.666667
```

```
sd(x, na.rm = TRUE)
```

```
[1] 2.516611
```

```
median(x, na.rm = TRUE)
```

```
[1] 5
```

```
IQR(x, na.rm = TRUE)
```

```
[1] 2.5
```

- `na.rm = TRUE` tells R to remove missing values before calculating

Summarizing a categorical variable (counts)

For categorical variables, we often start by counting how many observations fall in each category.

```r
table(iris$Species)
```

```
setosa versicolor  virginica
    50         50         50
```

Categorical proportions

We can convert counts into proportions using `prop.table()`.

```
prop.table(table(iris$Species))
```

```
    setosa versicolor  virginica
 0.3333333  0.3333333  0.3333333
```

To express proportions as percentages:

```
100 * prop.table(table(iris$Species))
```

```
    setosa versicolor  virginica
  33.33333   33.33333   33.33333
```

Plotting a numeric variable

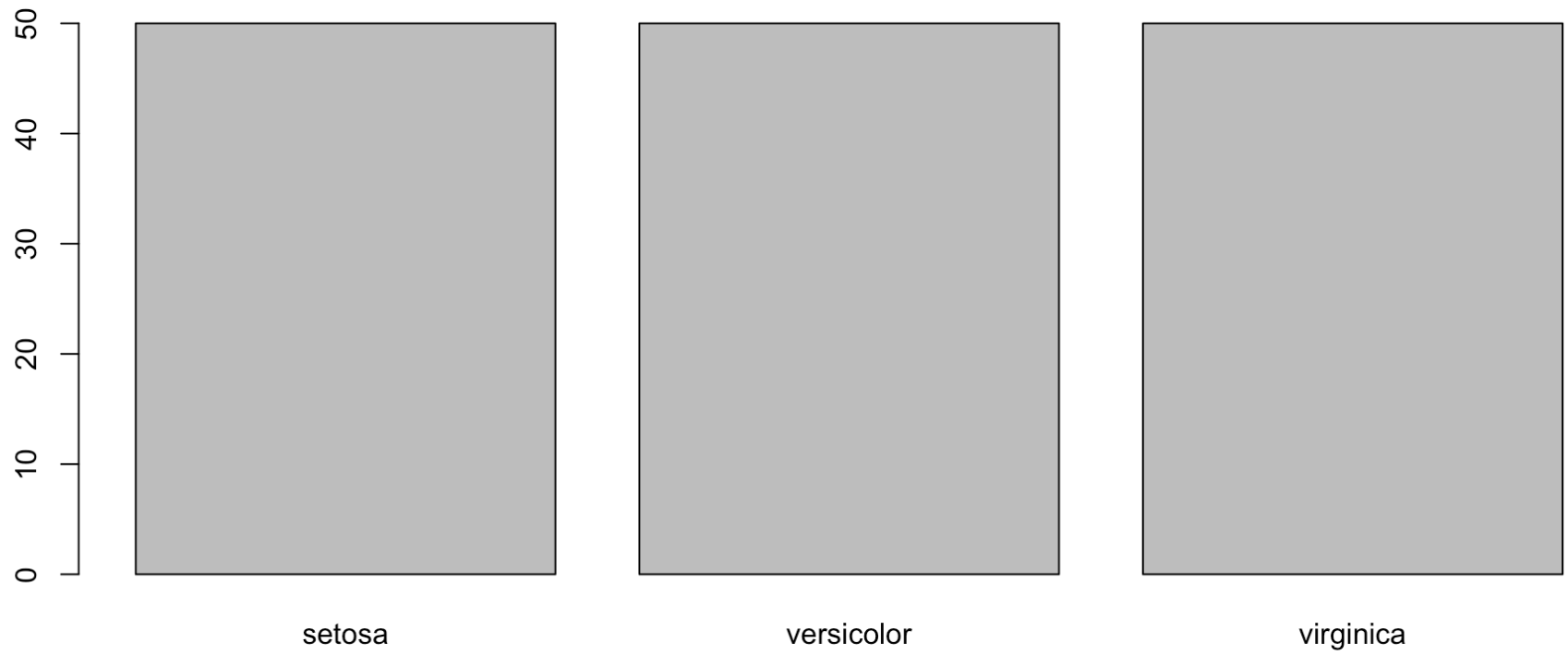
A histogram shows the distribution of a single numeric variable.

```
1 hist(iris$Petal.Length)
```

Plotting a categorical variable

A bar chart shows counts for each category.

```
1 barplot(table(iris$Species))
```



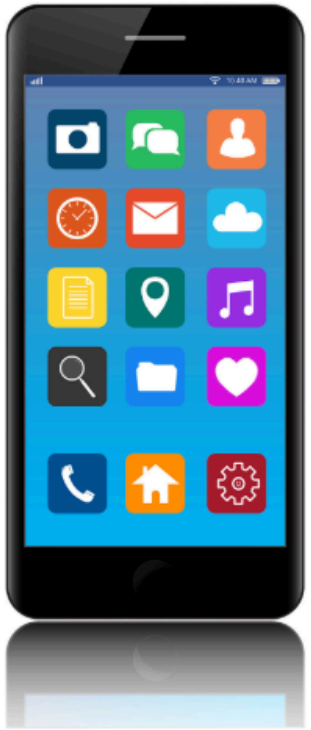
`barplot(table())` is a quick way to visualize categorical data.

R Packages


stringr
forcats
tidyverse
www.rstudio.com
dplyr
tidyr
readr
purrr
Horst

R Packages

A good analogy for R packages is that they are like apps you can download onto a mobile phone:

R: A new phone	R Packages: Apps you can download
	

ModernDive Figure 1.4

- Packages contain additional functions and data

Installing vs loading packages

There are two separate steps:

Install (only once per computer, example):

```
1 install.packages("dplyr")
```

Load (every time you start R):

```
1 library(dplyr)
```

Installing puts the package on your computer.

Loading makes it available in your R session.

- In practice, `install.packages()` and `library()` calls should go near the **top of a script** or in an **early code chunk** of a `.qmd` file

Using `::` to access package functions

There are two ways to use functions from a package.

Option 1: Load the package

```
1 library(dplyr)
2 arrange(iris, Petal.Length)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	4.6	3.6	1.0	0.2	setosa
2	4.3	3.0	1.1	0.1	setosa
3	5.8	4.0	1.2	0.2	setosa
4	5.0	3.2	1.2	0.2	setosa
5	4.7	3.2	1.3	0.2	setosa
6	5.4	3.9	1.3	0.4	setosa
7	5.5	3.5	1.3	0.2	setosa
8	4.4	3.0	1.3	0.2	setosa
9	5.0	3.5	1.3	0.3	setosa
10	4.5	2.3	1.3	0.3	setosa
11	4.4	3.2	1.3	0.2	setosa
12	5.1	3.5	1.4	0.2	setosa

Option 2: Use `::` without loading the package

```
1 dplyr::arrange(iris, Petal.Length)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	4.6	3.6	1.0	0.2	setosa
2	4.3	3.0	1.1	0.1	setosa
3	5.8	4.0	1.2	0.2	setosa
4	5.0	3.2	1.2	0.2	setosa
5	4.7	3.2	1.3	0.2	setosa
6	5.4	3.9	1.3	0.4	setosa
7	5.5	3.5	1.3	0.2	setosa
8	4.4	3.0	1.3	0.2	setosa
9	5.0	3.5	1.3	0.3	setosa
10	4.5	2.3	1.3	0.3	setosa
11	4.4	3.2	1.3	0.2	setosa
12	5.1	3.5	1.4	0.2	setosa

Both are valid.

- `library()` loads all functions from a package
- `package::function()` uses just one function
- Either approach is fine.
- I recommend including `library()` calls at the top of a file as a cue to yourself (and others) which packages are being used.

Learn more (optional)

If you want a short, clear walkthrough on R packages:

> **Installing and loading R packages** Danielle Navarro (YouTube) **https://www.youtube.com/watch?v=kpHZVyDvEhQ**

Textbook datasets (important)

The textbook datasets live in an R package called `oibiostat`.

It is **not on CRAN**, so we install it from GitHub, per the book authors' instructions.

```
install.packages("devtools")
devtools::install_github("OI-Biostat/oi_biostat_data", force = TRUE)
```

(This may take a few minutes the first time.)

After installing, load it (this part is done **every time** you restart R):

```
library(oibiostat)
```

Check that it worked

If installation worked, you can load the package and access a dataset:

```
library(oibiostat)
data(dds.discr)
head(dds.discr)
```

If this runs without errors, you’re set.

Wrap-up

Today you learned how to:

- Run basic R code
- Inspect a dataset
- Summarize numeric and categorical variables
- Make simple plots

We'll build on this foundation to:

- Work more efficiently with data
- Start using additional functions and packages