Foundations of Data Science for Biologists

R: data import, data frames, and data types

BIO 724D

03-SEP-2024

Paul Magwene and Greg Wray

Data types and data structures

Data types

Computers work with 0s and 1s — but you want to work with numbers, names, dates, etc.

Data types instruct programs how to interpret and process different kinds of data

Common data types in R are numeric, integer, character, and logical

R has an extensive set of rules for each data type:

What **values** are allowed (e.g., an integer can be `42` but not `'42'` or `42.7`)

What **operations** are allowed (e.g., division for integers but not character or logical)

How to **display** data in human-readable form (e.g., `01010010` as R or 82)

Variables point to data

When you create a variable in R, two things happen

R stores two kinds of information in a single package:

Data: values, such as `-23.84` or `'Adelie'` or a sequence of values

Metadata: what kind of information is being stored, how many values, etc.

The package is called an **object**

R stores the variable name and the memory address of the object in a separate table

That table contains the names and address of all the variables currently in use

Now, when you type the variable name, R knows:

Where the data are stored and how many values there are

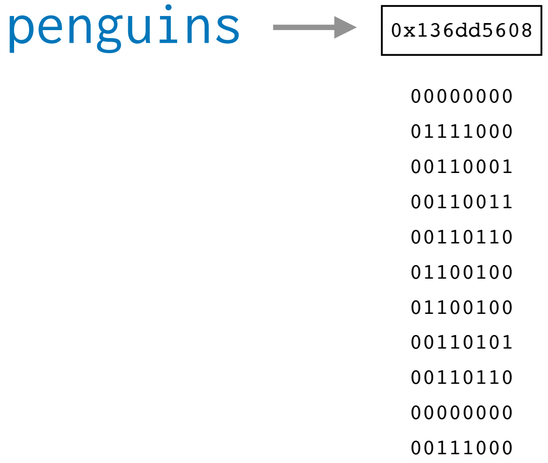
How to interpret the data: as numbers, letters, true/false, etc.

What a data frame looks like to you

`penguins` →

species	island	bill_length_mm	sex
Adelie	Torgersen	39.1	male
Adelie	Torgersen	39.5	female
Adelie	Biscoe	37.8	female
Adelie	Biscoe	37.7	male

What a data frame looks like to R and your computer



Data structures

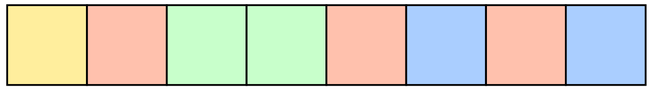
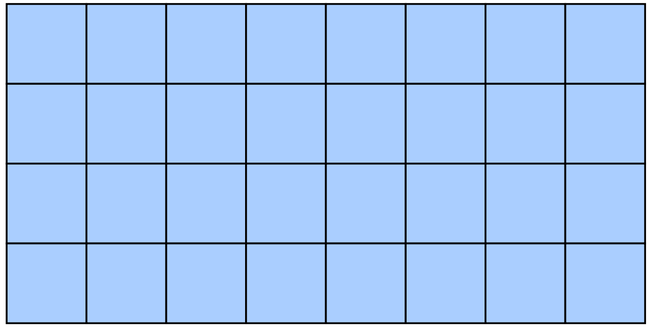
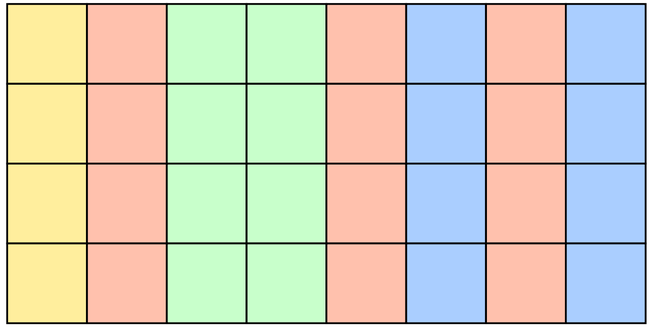
A data frame is an example of a **data structure**

- Data structures are built from more basic data types
- Defines what kinds of data can be stored (can be multiple types and dimensions)
- Defines what operations are permitted
- Usually optimized to work very efficiently with data in specific ways

For example, a data frame:

- Allows for arbitrary number of columns of mixed data types
- Requires all columns to be of the same length
- Allows for labels to be attached to columns (a type of metadata)

Taxonomy of basic data structures in R

	homogenous	mixed
1-dimensional	 vector	 list
2-dimensional	 matrix	 data frame

Vectors are the most basic data types in R

Vectors are also called **atomic** data types for this reason

Four atomic data types are very commonly used:

Numeric: real numbers; double-precision floating point by default

Integer: whole numbers

Logical: `TRUE`, `FALSE` (called **Boolean** in some languages)

Character: strings composed of letters, numerals, symbols, and whitespace

Two additional atomic data types are available but are rarely needed:

Complex: imaginary numbers with values like `2+3i`, where $i^2 = -1$

Raw: bytes with no implied meaning

What is a tibble?

tidyverse encourages using tibbles in place of data frames as the tabular data structure

A **tibble** is a data frame with slightly different behavior

Be aware, as you may encounter data as tibbles

To convert a tibble into a standard R data frame:

```
my_dataframe <- as.dataframe(data_in_tibble)
```

Check the tibble documentation for details: https://tibble.tidyverse.org/

Working with data structures in R

Naming data objects

R has some simple rules for naming data objects:

- Must start with a letter *or* . (dot) immediately followed by a letter
- May include: letters, numbers, underscore, dot, standard keyboard symbols
- May not include spaces (there is a work-around, but spaces are usually a bad idea)
- Case-sensitive
- Can be arbitrarily long
- Cannot be a **reserved word**; type `help(reserved)` or `?reserved`

Best practices

- When writing programs, favor descriptive, long names over simple, short ones
- Avoid relying on case and using symbols (other than underscore and dot)
- Avoid naming variables with the names of functions (although this is allowed!)

Use assignment method to create any kind of data object

The basic form of assignment is:

`my_obj <- 7`	read as: “my_object gets 7”
`my_obj <- c(7, 14, 21)`	creates an integer vector of length 3

Other valid forms of assignment:

`7 -> my_obj`	sometimes more readable
`my_obj <- 7 -> other_obj`	assigns value 7 to two different variables
`A <- B <- C <- 7`	assigns value 7 to three different variables
`my_obj = 7`	alternative assignment (not recommended)
`assign(my_obj, 7)`	using a function (but awkward!)

Use a **shortcut** for the assignment operator: alt+minus (Win) / opt+minus (Mac)

Data objects

The process of assignment creates a package of information called a **data object**

- The identifier and associated value(s) are stored together in memory
- Metadata are also stored: always data type and length; often additional information

You can learn about a data object in several ways, including:

`my_var`	returns current value(s)
`typeof(my_var)`	returns the object's specific data type
`class(my_var)`	returns the object's more general data type or structure
`length(my_var)`	returns the number of items in the data object
`str(my_var)`	returns a description of the structure of a data object
`attributes(my_var)`	returns the non-standard metadata of a data object
`View(my_var)`	displays all the data in a scrollable window (RStudio only)

Converting between data types

It is often possible and useful to convert between data types (called **coercion** in R)

- Must be a homogenous data type (vector, matrix, or column in a data frame)
- Must make logical sense (e.g., “2” can be coerced to integer but “kangaroo” cannot)

To coerce, use `as.integer()`, `as.logical()`, `as.character()`, etc.

Coercion rules to be aware of:

Numeric to integer	truncates any decimal values (does not round!)
Numeric to logical	`0` becomes `FALSE`; non-zero values become `TRUE`
Logical to numeric	`TRUE` becomes `1`, `FALSE` becomes `0`
Numeric to character	numerals and symbols become characters
Character to numeric	must be a formatted number (`-`, `+` and `.` allowed)

And many more; check documentation to avoid unexpected results!

Missing values

R provides three special values that represent missing, invalid, or undefined information

`NA`	a missing value; acronym = not available
`NaN`	an invalid mathematical result (e.g., `0/0`); acronym = not a number
`NULL`	a value that is undefined (e.g. vector of length `0`)

Points to remember:

- Do not use quotes: `'NA'` is interpreted a character value
- Do not use in mathematical operations: `my_var + NA` substitutes every item with `NA`
- Do not use in logical tests: `my_var == NA` returns `NA`

To identify missing values:

`is.na(my_vec)`	returns a logical vector with NAs FALSE, all others TRUE
`which(is.na(my_vec))`	returns the position(s) of any NAs in the vector

Assignment has many uses

Store the result of an operation:

`my_var <- 1 + 2`	evaluates RHS and assigns result to LHS
`my_vec <- old_vec * 3`	multiples each element by 3 during assignment

Create a new data object:

`my_vec <- c(1:10)`	creates a numeric vector containing values 1-10
`my_list <- list(1, “a”)`	creates a list containing values 1 and “a”

Update 1 or more values in an existing data object:

`my_vec[10] <- 42`	changes the value of item 10 to 42
`mvec[1:length(mvec)] <- 42`	replaces every value with 42, preserving length
`my_vec <- 42`	re-binds my_vec to a single value, length = 1
`my_vec[is.na(my_vec)] <- 0`	replaces NAs with 0; other values unchanged

Assignment has many uses, *continued*

Add items to an existing data object:

`my_vec[11] <- 300`	adds 1 item to a vector containing 10 items
`my_vec[12:15] <- c(1,3,4)`	adds 3 items to a vector containing 11 items

Delete items from an existing data object:

`my_vec <- my_vec[c(1,3,6)]`	removes all items except 1, 3, and 6
`my_vec <- my_vec[1:3]`	removes all items except the first 3

Copies an existing vector:

`new_vec <- old_vec`	copies values from one vector to another

Create a logical vector to use for subsetting or counting:

`logic_vec <- age_vec < 3`	assigns TRUE and FALSE values accordingly
`logic_vec <- is.na(my_vec)`	assigns TRUE and FALSE values accordingly