Introduction to R

Marine Ecosystem Dynamics

Kinlan M.G. Jan

Stockholm University

Plan for today’s lecture

- The `R` syntax
- The `R studio` software
- Variables, functions and vectors
- Importing data

Why using R?

Pro

🤑Free
🔓Open source
🥼Reproducible science

```r
# You can keep track of all the data analy
2 + 2 + 3        # step 1
#> [1] 7
log(2 + 2 + 3)   # step 2
#> [1] 1.94591
```

Why using R?

Pro

🤑Free
🔓Open source
🥼Reproducible science

```r
# You can keep track of all the data analy
2 + 2 + 3        # step 1
#> [1] 7
log(2 + 2 + 3)   # step 2
#> [1] 1.94591
```

Cons

👻Scary
🧑‍💻Syntax

```r
x = 1:100 ; y = log(x)
library(ggplot2)
ggplot() +
  geom_line(mapping = aes(x = x,
                          y = y),
            col = "firebrick",
            linewidth = 2) +
  theme_classic()+
  theme(axis.ticks = element_blank(),
        axis.text = element_blank(),
        axis.title = element_text(size = 
  labs(x = "Time",
       y = "Skills")
```

Why using R?

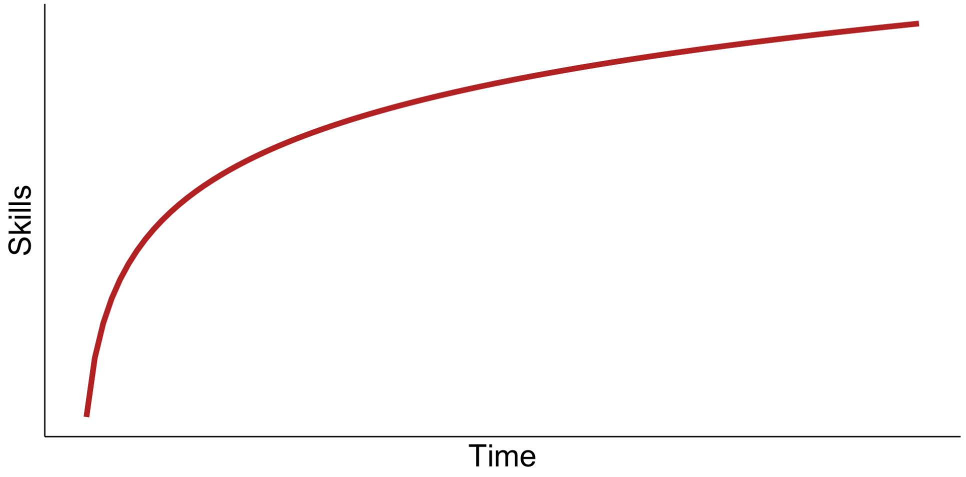
Pro

🤑Free
🔓Open source
🥼Reproducible science

```
# You can keep track of all the data analy
2 + 2 + 3         # step 1
#> [1] 7
log(2 + 2 + 3)   # step 2
#> [1] 1.94591
```

Cons

👻Scary
🧑‍💻Syntax



R studio is a great tool to use R

On one window it combines:

- Environment
- Console
- Script
- Plot, help, …

R is open and free

- People have worked on it and created tools and function that anyone case use!

- R base functions are already accessible when we open **R**
- More function from other packages 📦can be loaded

How to install and load packages

- A package need to be installed only once
- To use functions within a package call it using `library()`

```r
install.packages("PackageName")
library(PackageName)
```

- Once the package is installed we can look at the version of the package and how to cite it.

```r
packageVersion("PackageName")
citation("PackageName")
```

R syntax

- Like Excel, or a calculator **R** can help us resolve “basic” operations

```
2 + 2
#> [1] 4
```

R syntax

- Like Excel, or a calculator **R** can help us resolve “basic” operations

```r
2 + 2
#> [1] 4
4 * 4
#> [1] 16
```

Stockholm
University

R syntax

- Like Excel, or a calculator **R** can help us resolve “basic” operations

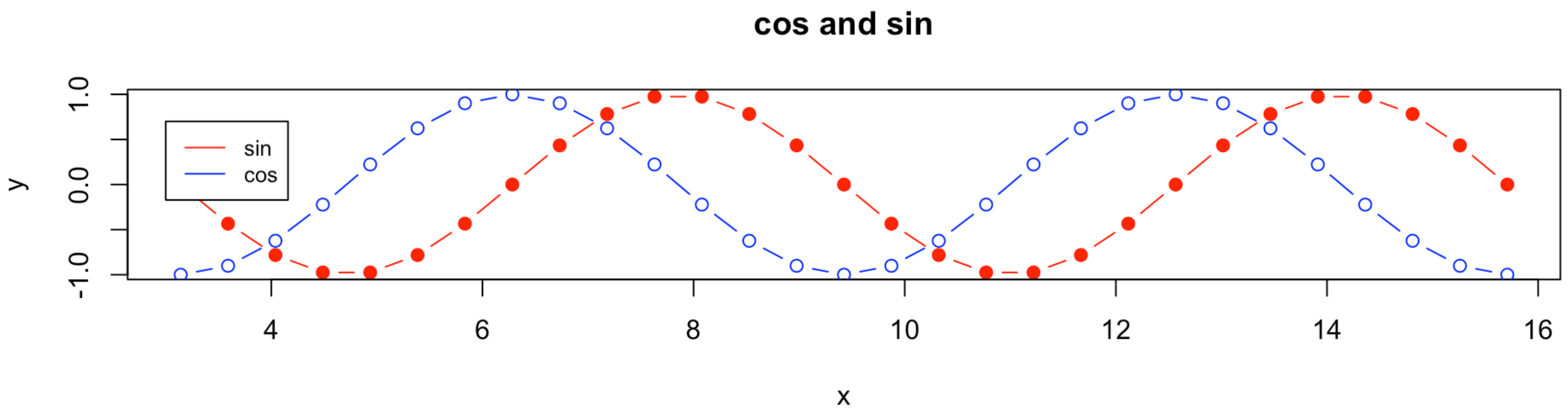
```r
2 + 2
#> [1] 4
4 * 4
#> [1] 16
(5 + 4) / (1 - 4 ^ 2)
#> [1] -0.6
```

R syntax

- Like Excel, or a calculator **R** can help us resolve “basic” operations

```r
2 + 2
#> [1] 4
4 * 4
#> [1] 16
(5 + 4) / (1 - 4 ^ 2)
#> [1] -0.6
```

- But also more complex operations



Values

In **R** values can be of several categories:

- Logical: `TRUE` or `FALSE`
- Numeric: `3` or `3.2`
- Character: `"t"`, `"blue"` or `"this is a character"`

We can ask **R** the category of our values

```
class(TRUE)
#> [1] "logical"
```

Values

In **R** values can be of several categories:

- Logical: `TRUE` or `FALSE`
- Numeric: `3` or `3.2`
- Character: `"t"`, `"blue"` or `"this is a character"`

We can ask **R** the category of our values

```
class(TRUE)
#> [1] "logical"
class(3)
#> [1] "numeric"
```

Values

In **R** values can be of several categories:

- Logical: `TRUE` or `FALSE`
- Numeric: `3` or `3.2`
- Character: `"t"`, `"blue"` or `"this is a character"`

We can ask **R** the category of our values

```
class(TRUE)
#> [1] "logical"
class(3)
#> [1] "numeric"
class("t")
#> [1] "character"
```

Values

In **R** values can be of several categories:

- Logical: `TRUE` or `FALSE`
- Numeric: `3` or `3.2`
- Character: `"t"`, `"blue"` or `"this is a character"`

We can ask **R** the category of our values

```
class(TRUE)
#> [1] "logical"
class(3)
#> [1] "numeric"
class("t")
#> [1] "character"
class(pi)
#> [1] "numeric"
```

Assigning variables

We can create variables that contain our values.

To do so, use `<-` or `=`

```
variable <- value
```

If we want to create a variable `x` that is equal to the value `3` and `y` that is equal to the value `"blue"`

```
x <- 3
y <- "blue"
```

Warning

Do not mix with `==` that test if the values are equals.

Tip

The opposite of `==` is `!=`

The variables are then stored in our “environment” and we can reuse them

```
x * 2 + x^x
#> [1] 33
```

Functions

R uses functions that all have the same structure:

`function_name(argument, ...)`

```r
log(argument1)
plot(argument1, argument2, ...)
```

It is **impossible** to know everything by heart and what arguments are needed.

Fortunately, manuals for each function exists using `?` before the function name.

```r
?log()
```

Vectors

R stores values in vectors or arrays that can be created in different ways:

```
vector1 <- c(1, 2, 3) ; print(vector1)
#> [1] 1 2 3
```

Vectors

R stores values in vectors or arrays that can be created in different ways:

```r
vector1 <- c(1, 2, 3) ; print(vector1)
#> [1] 1 2 3
vector2 <- seq(from = 3, to = 4, by = 0.34) ; print(vector2)
#> [1] 3.00 3.34 3.68
```

Vectors

R stores values in vectors or arrays that can be created in different ways:

```r
vector1 <- c(1, 2, 3) ; print(vector1)
#> [1] 1 2 3
vector2 <- seq(from = 3, to = 4, by = 0.34) ; print(vector2)
#> [1] 3.00 3.34 3.68
vector3 <- rep("blue", 2) ; print(vector3)
#> [1] "blue" "blue"
```

Vectors

R stores values in vectors or arrays that can be created in different ways:

```r
vector1 <- c(1, 2, 3) ; print(vector1)
#> [1] 1 2 3
vector2 <- seq(from = 3, to = 4, by = 0.34) ; print(vector2)
#> [1] 3.00 3.34 3.68
vector3 <- rep("blue", 2) ; print(vector3)
#> [1] "blue" "blue"
vector4 <- c(vector1, vector2) ; print(vector4)
#> [1] 1.00 2.00 3.00 3.00 3.34 3.68
```

Vectors

R stores values in vectors or arrays that can be created in different ways:

```r
vector1 <- c(1, 2, 3) ; print(vector1)
#> [1] 1 2 3
vector2 <- seq(from = 3, to = 4, by = 0.34) ; print(vector2)
#> [1] 3.00 3.34 3.68
vector3 <- rep("blue", 2) ; print(vector3)
#> [1] "blue" "blue"
vector4 <- c(vector1, vector2) ; print(vector4)
#> [1] 1.00 2.00 3.00 3.00 3.34 3.68
```

We use these vectors to do our calculations:

```r
vector1 * vector2
#> [1]  3.00  6.68 11.04
mean(vector2)
#> [1] 3.34
sd(vector4)
#> [1] 0.9924314
max(vector1)
#> [1] 3
```

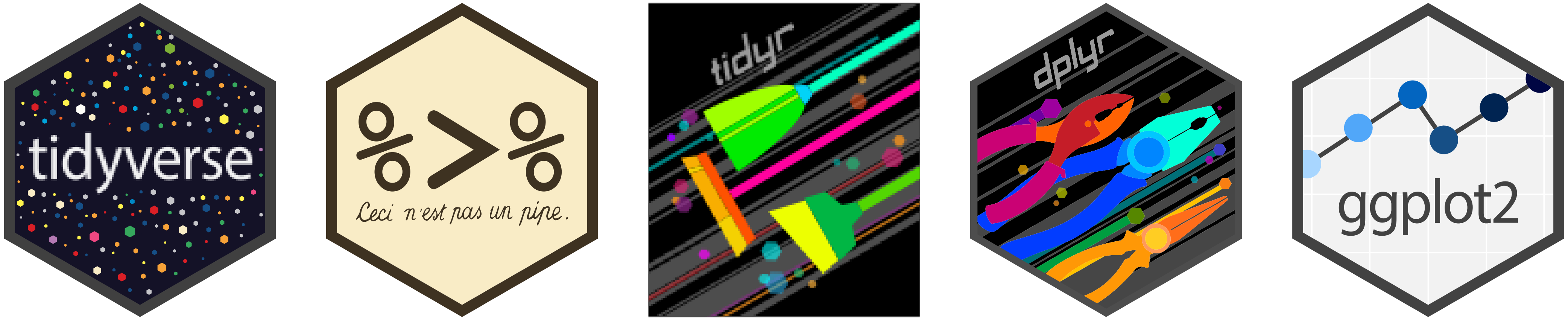
Importing data

The best way to do so is to use:

```
read.*(file = "/your/file.*", sep = "/t", dec = ",")
```

- Where `*` is:
  - `csv` - comma-separated values
  - `csv2` - semicolon-separated values, with comma as the decimal mark
  - `delim` - any delimited files
- `file` corresponds to the path of the file
- `sep` specifies the separator mark
- `dec` specifies the decimal mark

Plan for the next session



- Introduction to `tidyverse`
- Pipe the data using `magrittr`
- Clean the data using `tidyr`
- Arrange the data using `dplyr`
- Plot using `ggplot2`

Do not hesitate to use google to get help !

If you have an issue with something, you are probably not the first and someone asked a solution on a forum !

Google

stackoverflow

DATA NOVIA