Introdução à análise de dados biomédicos utilizando o R

Diego R. Mazzotti, Ph.D.
Research Associate
Center for Sleep and Circadian Neurobiology
University of Pennsylvania
diegomaz@upenn.edu

Agradecimentos

- Profa. Dra. Isabel Céspedes, Departamento de Morfologia e Genética, Universidade Federal de São Paulo

- CAPES Print

- Research Data Management Support – Utrecht University, Nederlands

- Prof. Dr. Ivan Pisa, Departamento de Informática em Saúde - UNIFESP

Universiteit Utrecht

Recursos úteis

- **Materiais deste curso:**
- https://github.com/UtrechtUniversity/workshop-introduction-to-R-and-data

- **Outros materiais relevantes de introdução ao R**
- https://www.tidyverse.org/learn/
- https://github.com/hbctraining/Intro-to-R
- https://www.rstudio.com/resources/cheatsheets/

Antes de começarmos…

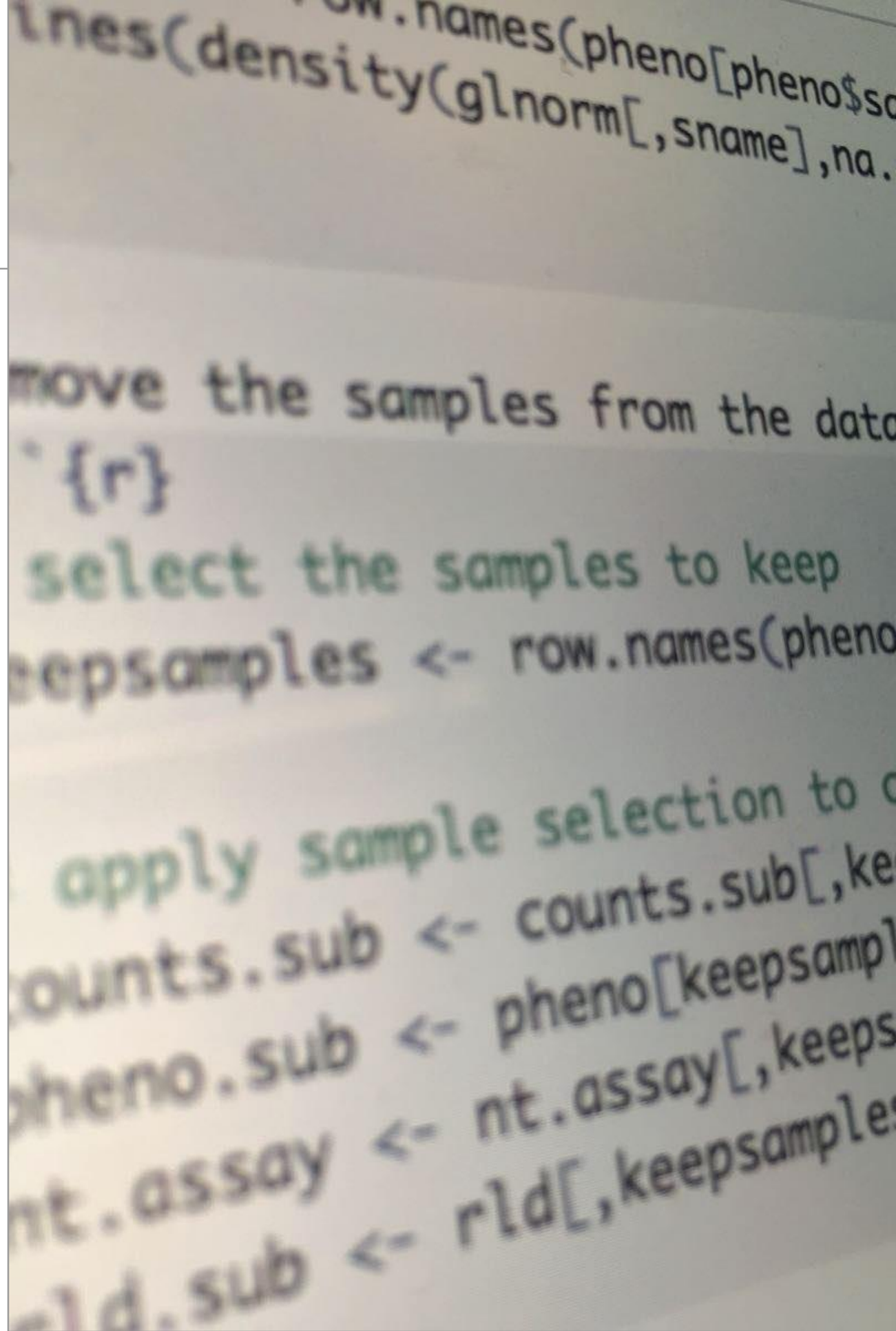
- Diversas formas de acessar o R:

  - Fazendo o download do R e do RStudio em seu computador
    - https://www.r-project.org/
    - https://www.rstudio.com/products/rstudio/download/

  - Acessando o RStudio server disponível pela UNIFESP (recomendado)
    - http://rserver.nuvia.unifesp.br/
    - Usar **usuário** e **senha** da INTRANET

Introdução ao R e análise de dados

- Parte 1: Fundamentos do R
- Parte 2: R e o universo **tidyverse**



Introdução ao R & análise de dados

- Parte 1: Fundamentos do R

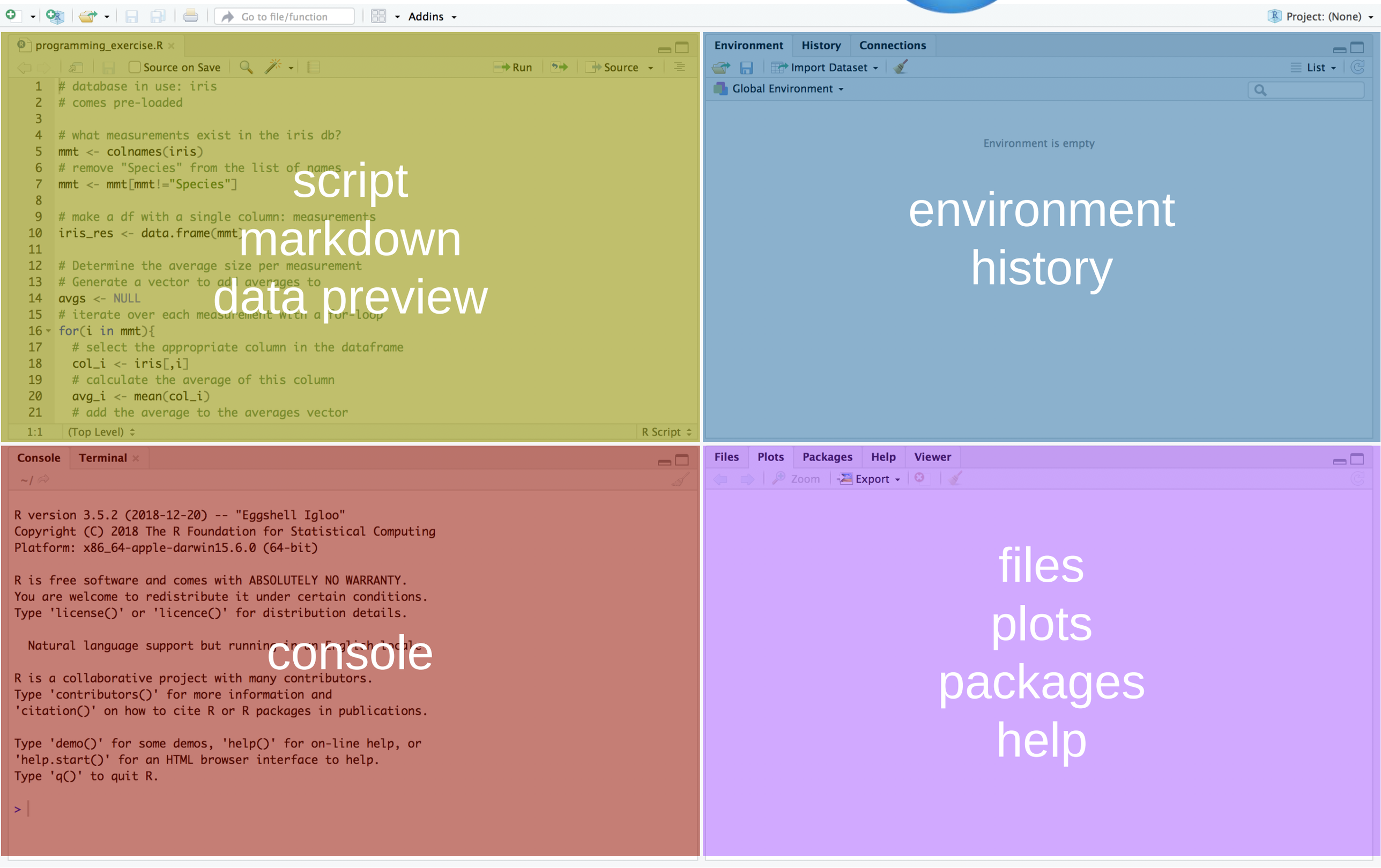
O que é o R?

- Uma linguagem de programação amplamente usada para análise de dados
- Baseada em uma linguagem de programação estatística chamada **S** (1976)
- Desenvolvida por **R**oss Ihaka & **R**obert Gentleman (1995)
- Apresenta uma comunidade de usuários extremamente ativa, com muitos pacotes específicos de diversas áreas do conhecimento

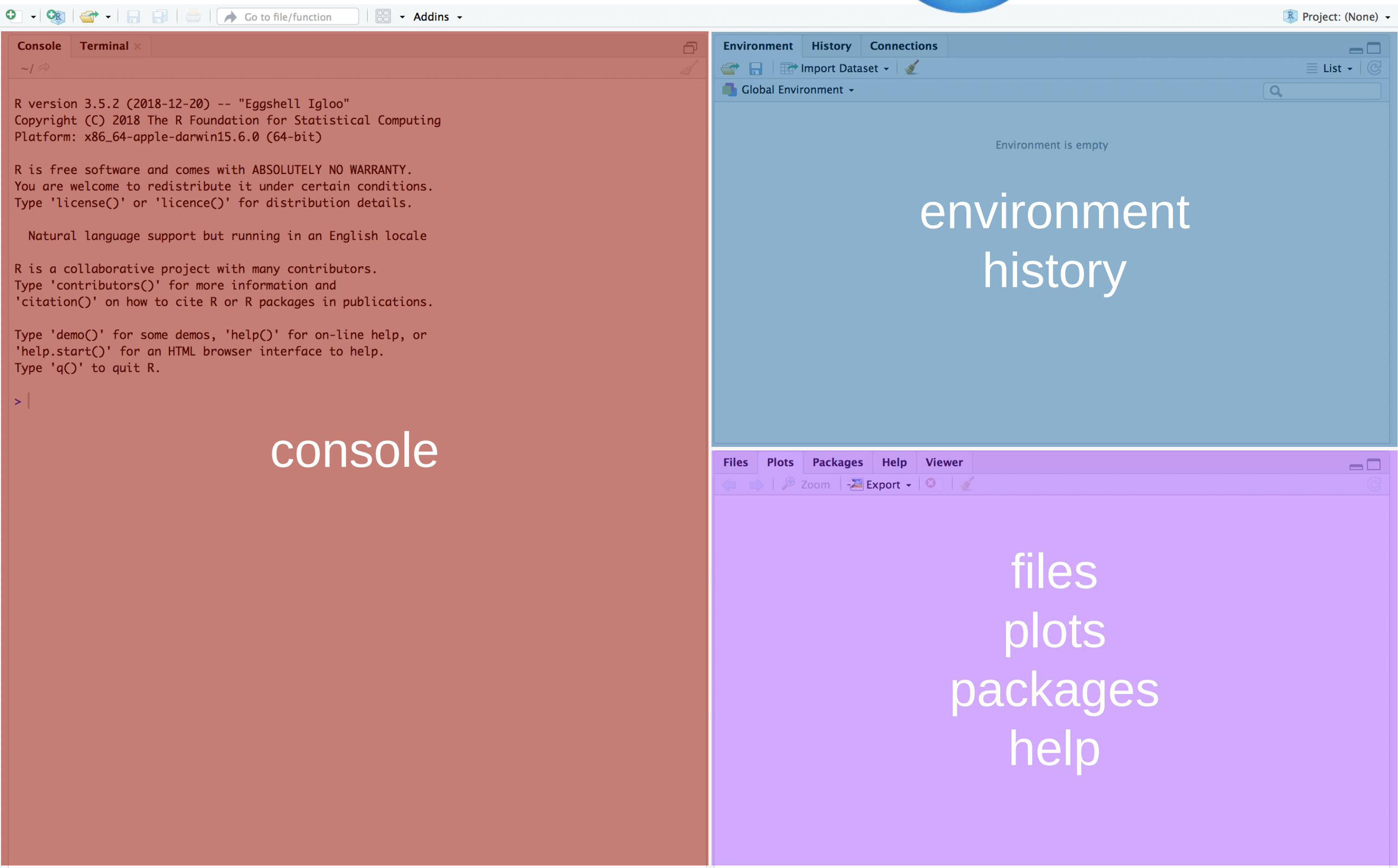
O que é o RStudio

- Ambiente de desenvolvimento integrado (IDE) para o R
- Fundado por JJ Allaire, e disponível desde 2010
- Extremamente útil!

The Rstudio interface



The Rstudio interface



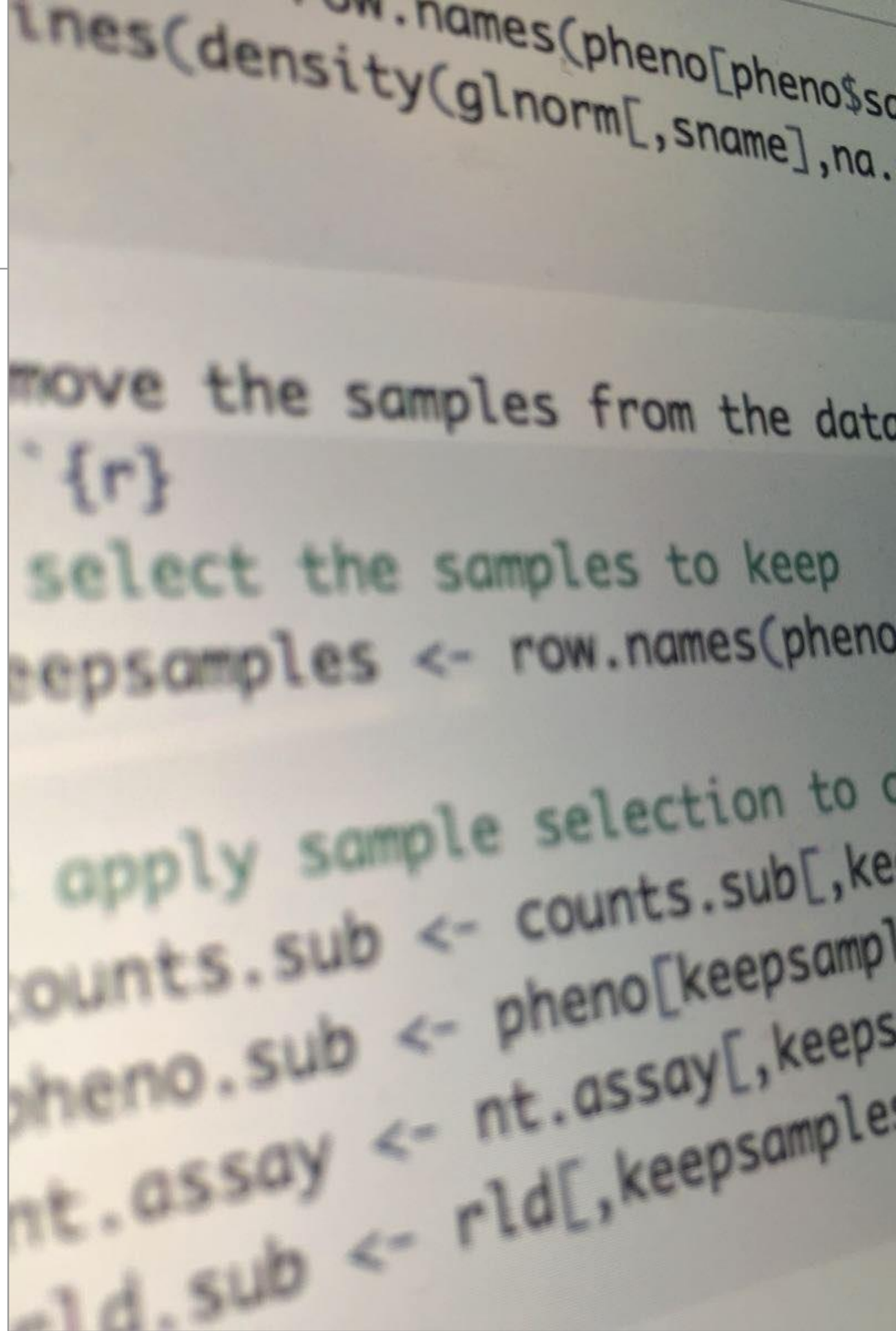
Download dos materiais necessários

- https://github.com/UtrechtUniversity/workshop-introduction-to-R-and-data

- Fazer o upload do arquivo no RStudio utilizando o botão “Upload”
- Ou descompate o arquivo .zip em um diretório onde irá trabalhar em seu computador local

Sintaxe do R e console

- Tipos de dados no R
- Funções
- Concatenando dados
- Indexando dados



Importante

- O console é apenas um local de execução dos comandos
- É possível (e recomendado!) escrever os comandos que vamos rodar em um script para facilitar o acompanhamento
- É possível (e recomendado!) escrever comentários explicando o que cada comando faz:

```
# comentários podem ser escritos dessa forma
# usando o  símbolo do hashtag (#)
# rodar uma linha comentada não resultará em nada
```

Atribuição de valores à variáveis

```
> x <- 6
```

Atribuição de valores à variáveis

```
> x <- 6

> x <- "hi!"

> 6 -> x

> x = 6

> 6 = x
Error in 6 = x : invalid (do_set)
```

Variable Assignment

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

Base R Cheat Sheet

Getting Help

Accessing the help files

?mean
Get help of a particular function.
help.search('weighted mean')
Search the help files for a word or phrase.
help(package = 'dplyr')
Find help for a package.

More about an object

str(iris)
Get a summary of an object's structure.
class(iris)
Find the class an object belongs to.

Using Packages

install.packages('dplyr')
Download and install a package from CRAN.

library(dplyr)
Load the package into the session, making all its functions available to use.

dplyr::select
Use a particular function from a package.

data(iris)
Load a built-in dataset into the environment.

Working Directory

getwd()
Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

c(2, 4, 6)	2 4 6	Join elements into a vector
2:6	2 3 4 5 6	An integer sequence
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x) Return x sorted.
rev(x) Return x reversed.
table(x) See counts of values.
unique(x) See unique values.

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.

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'.

Programming

For Loop

```
for (variable in sequence){
  Do something
}
```

Example

```
for (i in 1:4){
  j <- i + 10
  print(j)
}
```

While Loop

```
while (condition){
  Do something
}
```

Example

```
while (i < 5){
  print(i)
  i <- i + 1
}
```

If Statements

```
if (condition){
  Do something
} else {
  Do something different
}
```

Example

```
if (i > 3){
  print('Yes')
} else {
  print('No')
}
```

Functions

```
function_name <- function(var){
  Do something
  return(new_variable)
}
```

Example

```
square <- function(x){
  squared <- x*x
  return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Ouput	Description
df <- read.table('file.txt')	write.table(df, 'file.txt')	Read and write a delimited text file.
df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a comma separated value file. This is a special case of read.table/ write.table.
load('file.RData')	save(df, file = 'file.Rdata')	Read and write an R data file, a file type special for R.

Conditions

a == b	Are equal	a > b	Greater than	a >= b	Greater than or equal to	is.na(a)	Is missing
a != b	Not equal	a < b	Less than	a <= b	Less than or equal to	is.null(a)	Is null

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

as.logical	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE).
as.numeric	1, 0, 1	Integers or floating point numbers.
as.character	'1', '0', '1'	Character strings. Generally preferred to factors.
as.factor	'1', '0', '1', levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

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.

Variable Assignment

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

The Environment

ls() List all variables in the environment.
rm(x) Remove x from the environment.
rm(list = ls()) Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrices

m <- matrix(x, nrow = 3, ncol = 3)
Create a matrix from x.

m[2,] - Select a row
m[, 1] - Select a column
m[2, 3] - Select an element

t(m) Transpose
m %*% n Matrix Multiplication
solve(m, n) Find x in: m * x = n

Lists

l <- list(x = 1:5, y = c('a', 'b'))
A list is a collection of elements which can be of different types.

l[[2]]	l[1]	l$x	l['y']
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Data Frames

Also see the **dplyr** package.

df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

df$x **df[[2]]**

Understanding a data frame

View(df) See the full data frame.
head(df) See the first 6 rows.

Matrix subsetting

df[, 2] **df[2,]** **df[2, 2]**

nrow(df) Number of rows.
ncol(df) Number of columns.
dim(df) Number of columns and rows.

cbind - Bind columns.
rbind - Bind rows.

Strings

Also see the **stringr** package.

paste(x, y, sep = ' ')	Join multiple vectors together.
paste(x, collapse = ' ')	Join elements of a vector together.
grep(pattern, x)	Find regular expression matches in x.
gsub(pattern, replace, x)	Replace matches in x with a string.
toupper(x)	Convert to uppercase.
tolower(x)	Convert to lowercase.
nchar(x)	Number of characters in a string.

Factors

factor(x) Turn a vector into a factor. Can set the levels of the factor and the order.
cut(x, breaks = 4) Turn a numeric vector into a factor by 'cutting' into sections.

Statistics

lm(y ~ x, data=df) Linear model.
glm(y ~ x, data=df) Generalised linear model.
summary Get more detailed information out a model.

t.test(x, y) Perform a t-test for difference between means.
pairwise.t.test Perform a t-test for paired data.

prop.test Test for a difference between proportions.
aov Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	rnorm	dnorm	pnorm	qnorm
Poisson	rpois	dpois	ppois	qpois
Binomial	rbinom	dbinom	pbinom	qbinom
Uniform	runif	dunif	punif	qunif

Plotting

Also see the **ggplot2** package.

plot(x) Values of x in order.
plot(x, y) Values of x against y.
hist(x) Histogram of x.

Dates

See the **lubridate** package.

Operações matemáticas

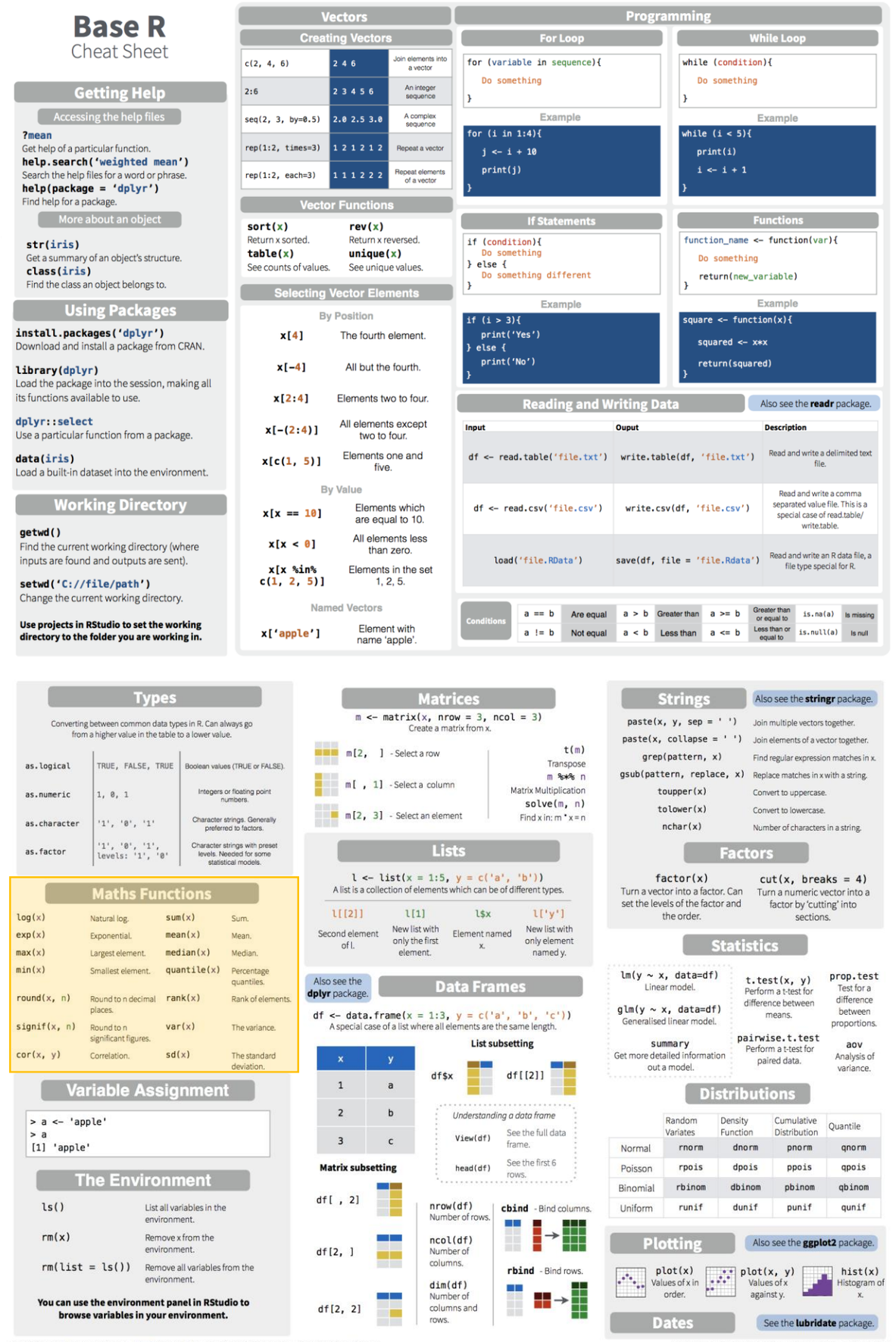
```
> x * 3
[1] 18

> 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.



Exercício

1. Faça a seguinte operação matemática no R:

$$\frac{1+5}{9}$$

2. Atribua o valor desta operação à uma variável

2. **Bônus:** faça o arredontamento do resultado para uma casa decimal.
   *Dica: Veja a seção **Maths Functions** da cheat sheet do Base R!*

Exercício

1. Faça a seguinte operação matemática no R:

$$\frac{1+5}{9}$$

```
> (1+5)/9
```

2. Atribua o valor desta operação à uma variável

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

2. **Bônus:** faça o arredontamento do resultado para uma casa decimal.
   *Dica: Veja a seção **Maths Functions** da cheat sheet do Base R!*

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

Outro tipo de dado: logical

T	TRUE
F	FALSE

```
> T
[1] TRUE

> F
[1] FALSE

> x==6
[1] TRUE

> 2>4
[1] FALSE
```

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

Concatenando dados: vetores

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

Vetor numérico

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

Vetor de caractere

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

Vetor lógico

Funções aplicadas a vetores

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

> p * q
[1] 5 8 9 8 5
```

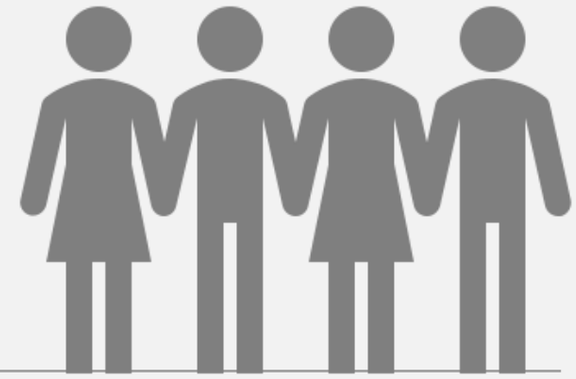
p * 2

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

Exercício: criação de vetores



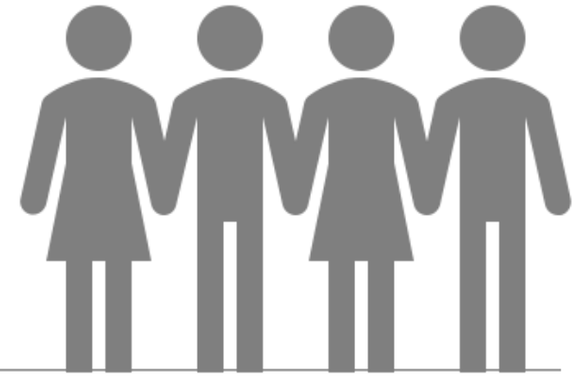
Crie um vetor com dados sobre 4 pessoas: Ana, Roberto, Beatriz, Daniel

1. Crie um vetor de caracteres com os nomes destas pessoas, usando a função `c()`. Salve o resultado em um vetor chamado `nome`.

2. Crie um vetor com a idade de cada pessoa, respectivamente. Salve em um vetor chamado `idade`.

Bônus: Use uma função do R para calcular a média de idade destas quarto pessoas.

Exercício: criação de vetores



Crie um vetor com dados sobre 4 pessoas: Ana, Roberto, Beatriz, Daniel

1. Crie um vetor de caracteres com os nomes destas pessoas, usando a função `c()`. Salve o resultado em um vetor chamado `nome`.

```
> nome <- c("Ana", "Roberto", "Beatriz", "Daniel")
```

2. Crie um vetor com a idade de cada pessoa, respectivamente. Salve em um vetor chamado `idade`.

```
> idade <- c(35,22,50,51)
```

Bônus: Use uma função do R para calcular a média de idade destas quarto pessoas.

```
> mean(idade)
[1] 39.5
```

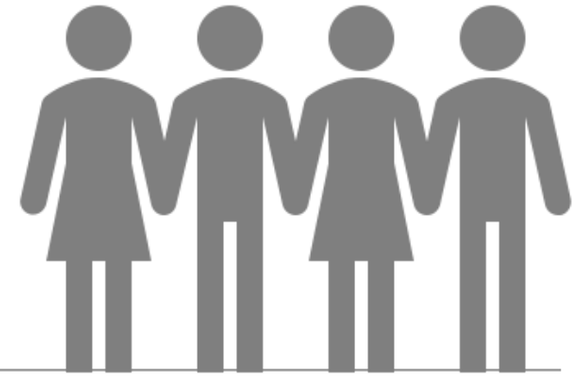
Vetores de fatores

```
> country <- c("UK","USA","USA","UK")
> country
[1] "UK"  "USA" "USA" "UK"

> as.factor(country)
[1] UK  USA USA UK
Levels: UK USA
```

Um **fator** é definido por **levels** (categorias)

Concatenando dados



```
> nome
[1] "Ana"    "Roberto" "Beatriz" "Daniel"
> idade
[1] 35 22 50 51

> c(nome,idade)
[1] "Ana" "Roberto" "Beatriz" "Daniel"   "35"    "22"    "50"    "51"
```

```
> list(nome,idade)
[[1]]
[1] "Ana"    "Roberto"    "Beatriz" "Daniel"

[[2]]
[1] 35 22 50 51

> data.frame(nome,idade)
    nome          idade
1    Ana          35
2    Roberto      22
3    Beatriz      50
4    Daniel       51
```

Vetores, listas e data frames

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

Exercício: concatenando dados

1. Crie um vetor chamado `country` contendo 4 países (utilize pelo menos uma duplicata).

2. Crie um data frame concatenando `nome`, `idade`, e `country`, salve como `df`.

Bônus: crie uma lista contendo `nome`, `idade`, e `country`.

Exercício: concatenando dados

1. Crie um vetor chamado `country` contend 4 países (utilize pelo menos uma duplicata).

```
> country <- c("UK","USA","USA","UK")
```

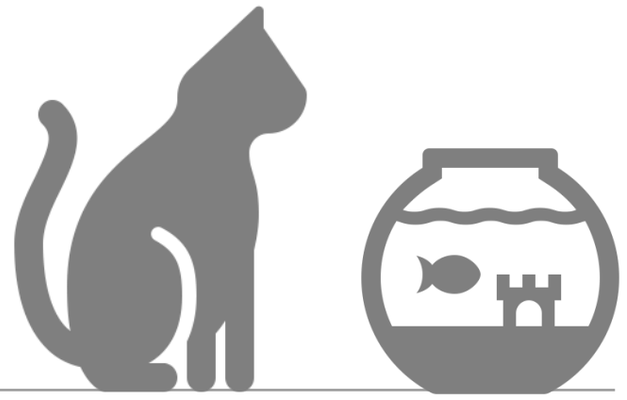
2. Crie um data frame concatenando `nome`, `idade`, e `country`, salve como `df`.

```
> df <- data.frame(nome,idade,country)
```

Bônus: crie uma lista contendo `nome`, `idade`, e `country`.

```
> list(nome,idade,country)
```

Dados “missing”: NA



```
> pet <- c("cat","none","fish",NA)
> pet
[1] "cat"  "none"  "fish" NA

> as.factor(pet)
[1] cat  none  fish  <NA>
Levels: cat fish none
```

NA = Not Available

nome	idade	country	pet
Ana	35	UK	cat
Roberto	22	USA	none
Beatriz	50	USA	fish
Daniel	51	UK	NA

Em outras palavras:

Sabemos que Roberto não tem **pets**.

Não sabemos se Daniel tem **pets**.

Exercício

Tente predizer o resultado das seguintes operações

```
> 5 == 5
```

```
> 5 == NA
```

```
> NA == NA
```

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

Exercício

Tente predizer o resultado das seguintes operações

```
> 5 == 5
[1] TRUE

> 5 == NA
[1] NA

> NA == NA
[1] NA

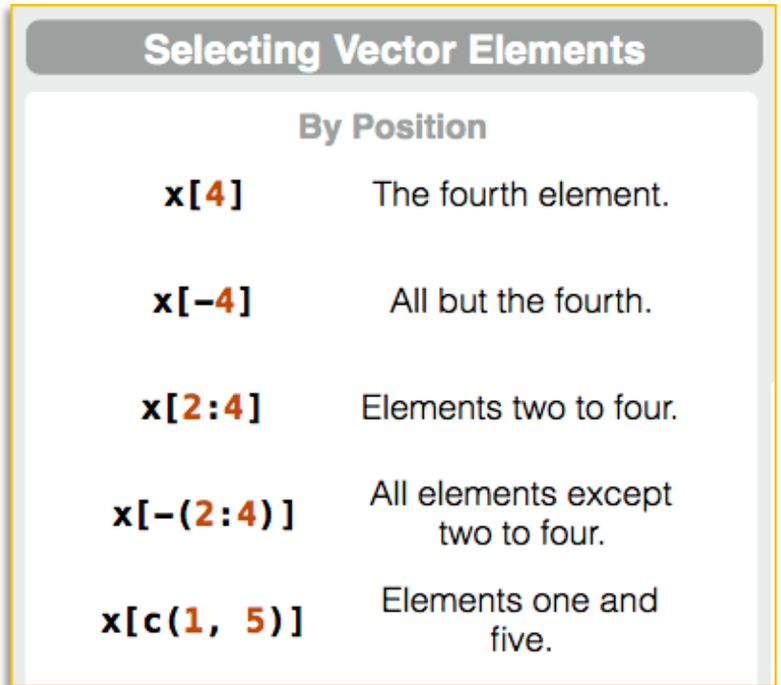
> is.na(NA)
[1] TRUE
```

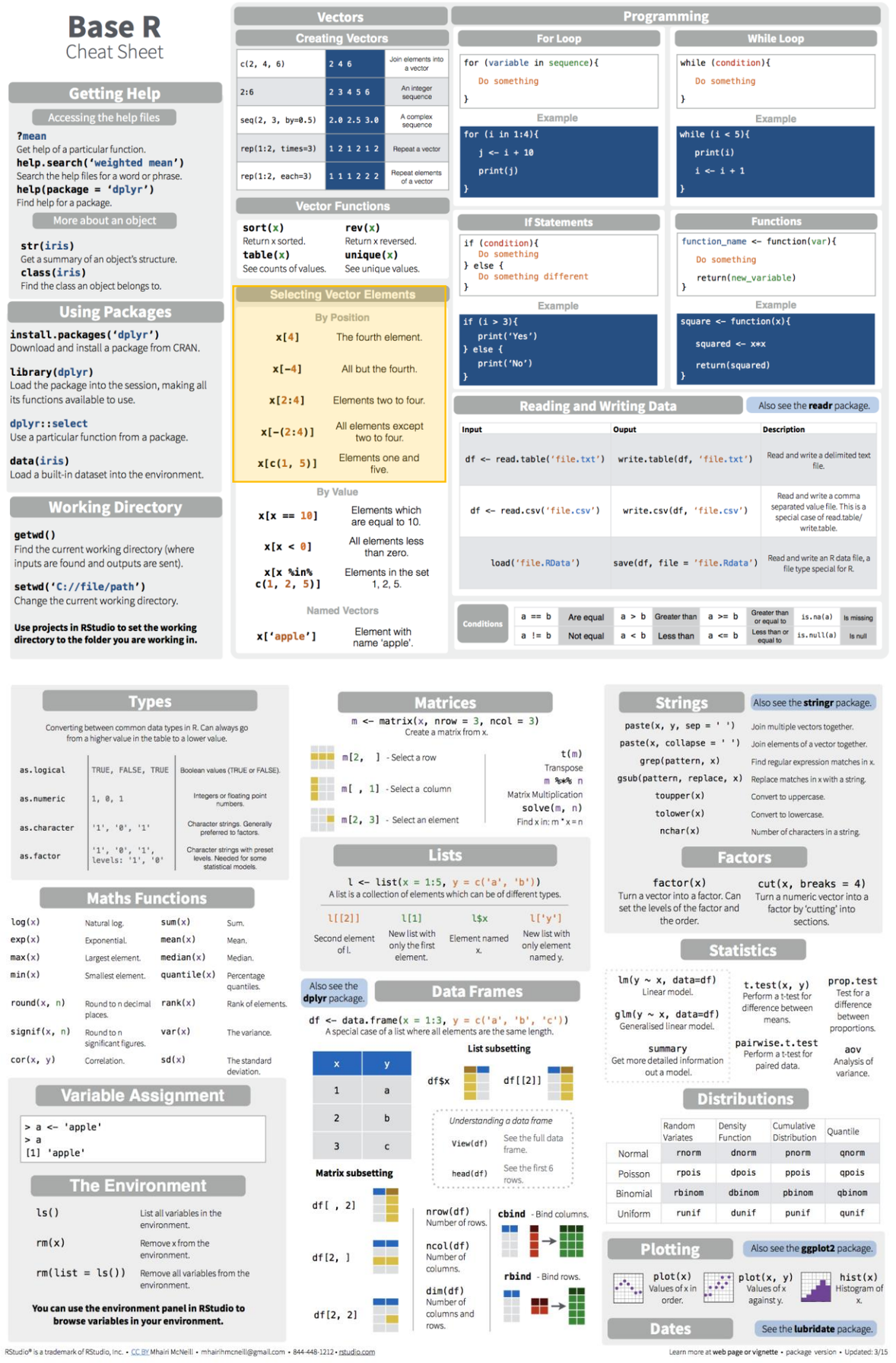
Selecionando elementos de um vetor pela posição

```
> nome
[1] "Ana"    "Roberto"
[3] "Beatriz" "Daniel"

> nome[2]
[1] "Roberto"

> nome[1:3]
[1] "Ann"    "Bob"    "Chloe"
```





Selecionado elementos de um vetor pelo valor

```
> idade
[1] 35 22 50 51

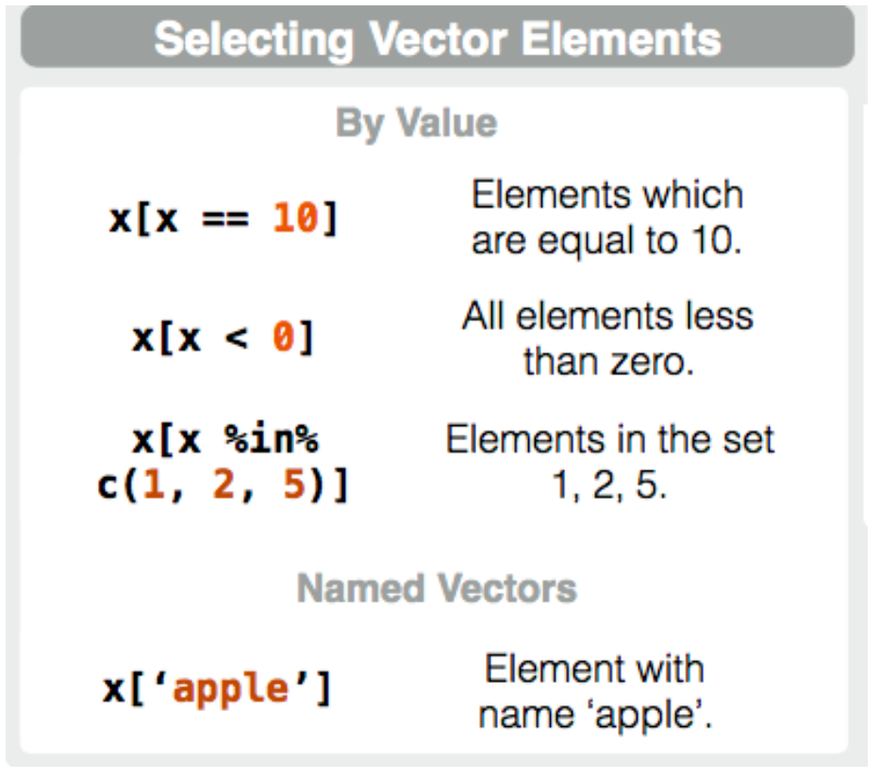
> idade[idade>40]
[1] 50 51

> age>40
[1] FALSE FALSE TRUE TRUE

> nome[nome %in% c("Beatriz","Ana")]
[1] "Ana"  "Beatriz"

> nome[NA]
[1] NA NA NA NA
```

age	age>40	result
35	FALSE	
22	FALSE	
50	TRUE	50
51	TRUE	51



Exercício

1. Selecione o primeiro elemento do vetor `idade`.

2. Selecione o segundo e quarto elemento do vetor `nome`.

3. Selecione apenas as idades menores que 30 do vetor `idade`.

Exercício

1. Selecione o primeiro elemento do vetor `idade`.

```
> idade[1]
[1] 35
```

2. Selecione o segundo e quarto elemento do vetor `nome`.

```
> nome[c(2,4)]
[1] "Roberto" "Daniel"
```

3. Selecione apenas as idades menores que 30 do vetor `idade`.

```
> idade[idade<30]
   [1] 22
```

Selecionando elementos de listas

```
> mylist <- list(nome,idade)
> mylist
[[1]]
[1] "Ana"    "Roberto"    "Beatriz" "Daniel"
[[2]]
[1] 35 22 50 51
```

```
> mylist[1]
[[1]]
[1] "Ana"    "Roberto"    "Beatriz" "Daniel"
> mylist[[1]]
[1] "Ana"    "Roberto"    "Beatriz" "Daniel"
> mylist[[1]][2]
[1] "Roberto"
> mylist[1][2]
[[1]]
NULL
```

Seleciona o primeiro elemento da lista

Seleciona o **conteúdo** do primeiro elemento da lista

NA != NULL

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

Exercício: tente predizer o resultado das funções

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

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

```
> is.null(NA)

> is.na(NULL)
```

Exercício: tente predizer o resultado das funções

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

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

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

Selecionando elementos de um data frame

Matrix subsetting

`df[ , 2]`

`df[2, ]`

row

column

`df[2, 2]`

nome	idade	country
Ana	35	UK
Roberto	22	USA
Beatriz	50	USA
Daniel	51	UK

Selecionando elementos de um data frame

```
> df[,2]
[1] 35 22 50 51
> df[,"idade"]
[1] 35 22 50 51
> df$idade
[1] 35 22 50 51

> df[2,]
  nome    idade country
2  Roberto  22      USA
> df[df$nome=="Roberto",]
 nome     idade country
2 Roberto  22      USA

> df[df$nome=="Roberto","idade"]
[1] 22
```

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

nome	idade	country
Ana	35	UK
Roberto	22	USA
Beatriz	50	USA
Daniel	51	UK

```
df[2,]
df[df$nome=="Roberto",]
```

Remover variáveis

Antes do próximo exercício, vamos **remover** os vetores que criamos anteriormente:

```
> rm(nome,idade,country)
```

Exercício

1. No data frame `df`, selecione todos os dados de todos os indivíduos de um país a sua escolha.

1. Selecione **apenas os nomes** de todos no data frame `df` **com idades menores de 40 anos**.

1. **Bônus:** Use seleção de vetores para ter os mesmos resultados

Exercício

1. No data frame `df`, selecione todos os dados de todos os indivíduos de um país a sua escolha.

```
> df[df$country=="USA",]
   nome     idade   country
2  Roberto    22       USA
3  Beatriz    50       USA
```

1. Selecione **apenas os nomes** de todos no data frame `df` **com idades menores de 40 anos**.

```
> df[df$idade<40,"nome"]
   > df[df$idade<40,1]
   [1] Ana Roberto
   Levels: Ana Roberto Beatriz Daniel
```

índices do data frame `df`

1. **Bônus:** Use seleção de vetores para ter os mesmos resultados

```
> df$nome[df$idade<40]
```

Índices do vetor `df$nome`

Colchetes, parênteses e chaves

[ ] **Índices** de vetores, listas e data frames

( ) Determinar **argumentos** de funções

{ } **Definir expressões** (funções, loop, etc.)

Recapitulando

Tipos de dados

`logical`

`numeric`

`character`

Dados missing

`NA` (Não disponíveis)

`NULL` (Não existentes)

Tipos de combinação de dados

`vector` (uma dimensão)

`factor` (uma dimensão, com níveis)

`data frame` (duas dimensões)

`list` (várias dimensões)

Funções

Quais funções vimos até agora?

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

Caso não saiba o que estas funções signifiquem:

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

Help!

table {base} — function & package names — 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)

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

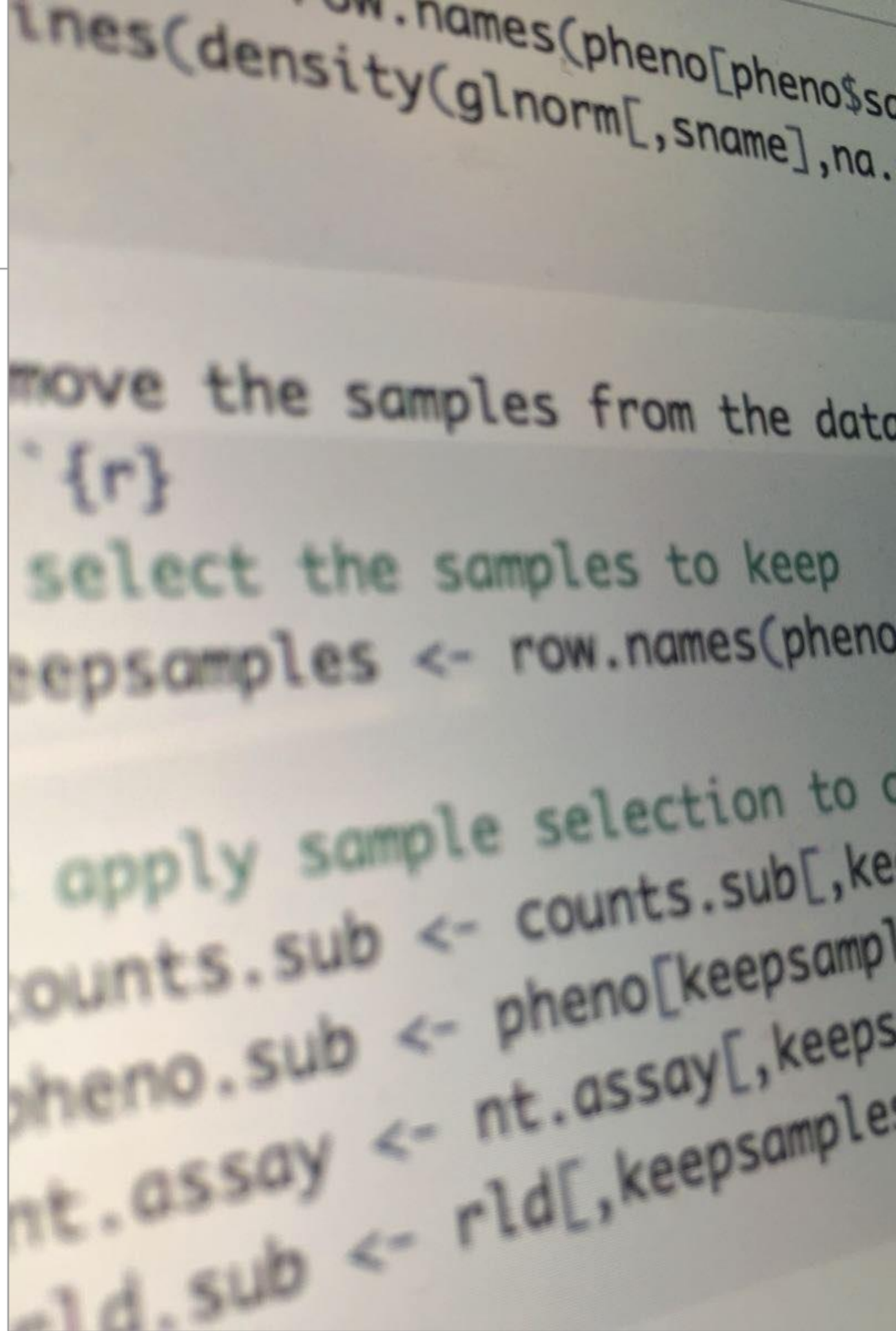
## S3 method for class 'table'
as.data.frame(x, row.names = NULL, ...,
```

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

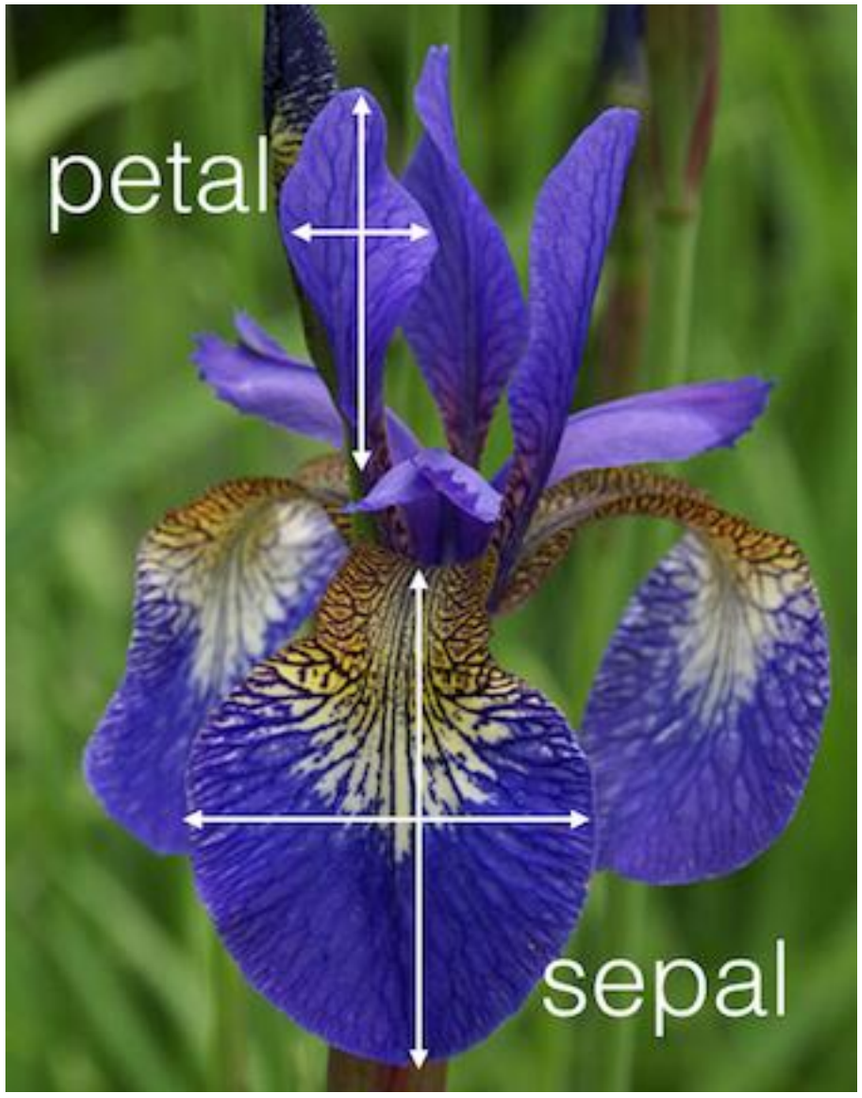
Programação em R

- Escrevendo o primeiro script
- Programando com loops e funções
- Manipulando e processando datasets



Um exemplo de banco de dados (dataset)

- Dataset: ‘iris’
- Dataset padrão sobre medidas de flores do gênero *Iris*



Introducing a sample dataset

- Dataset: ‘iris’

- Dataset padrão sobre medidas de flores do gênero *Iris*

```
> 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
```

Exercício

Explore o data frame `iris` usando as funções abaixo

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

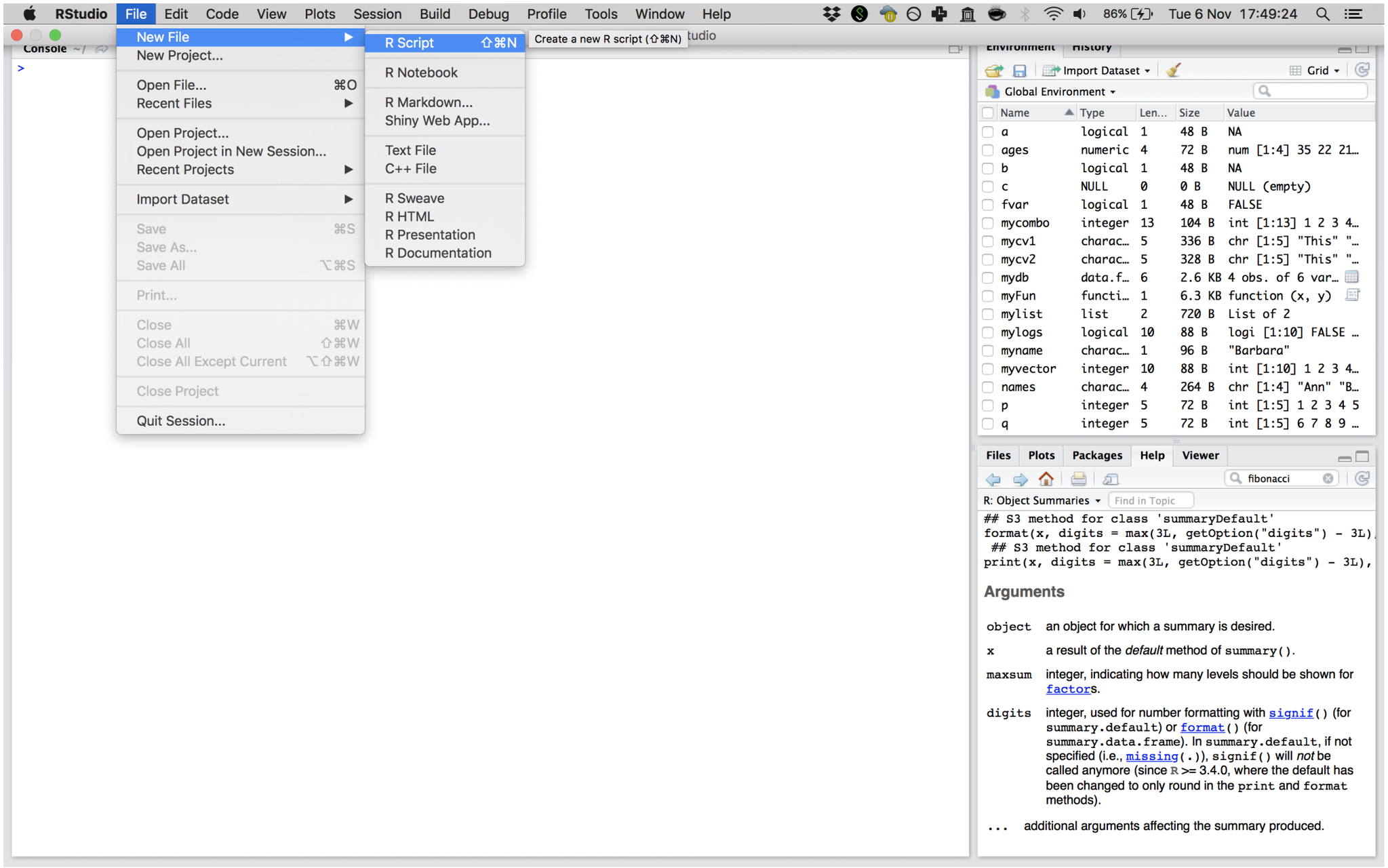
O que estas funções fazem? O que elas mostram sobre os tipos de dados no data frame `iris`?

Console e Scripts

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

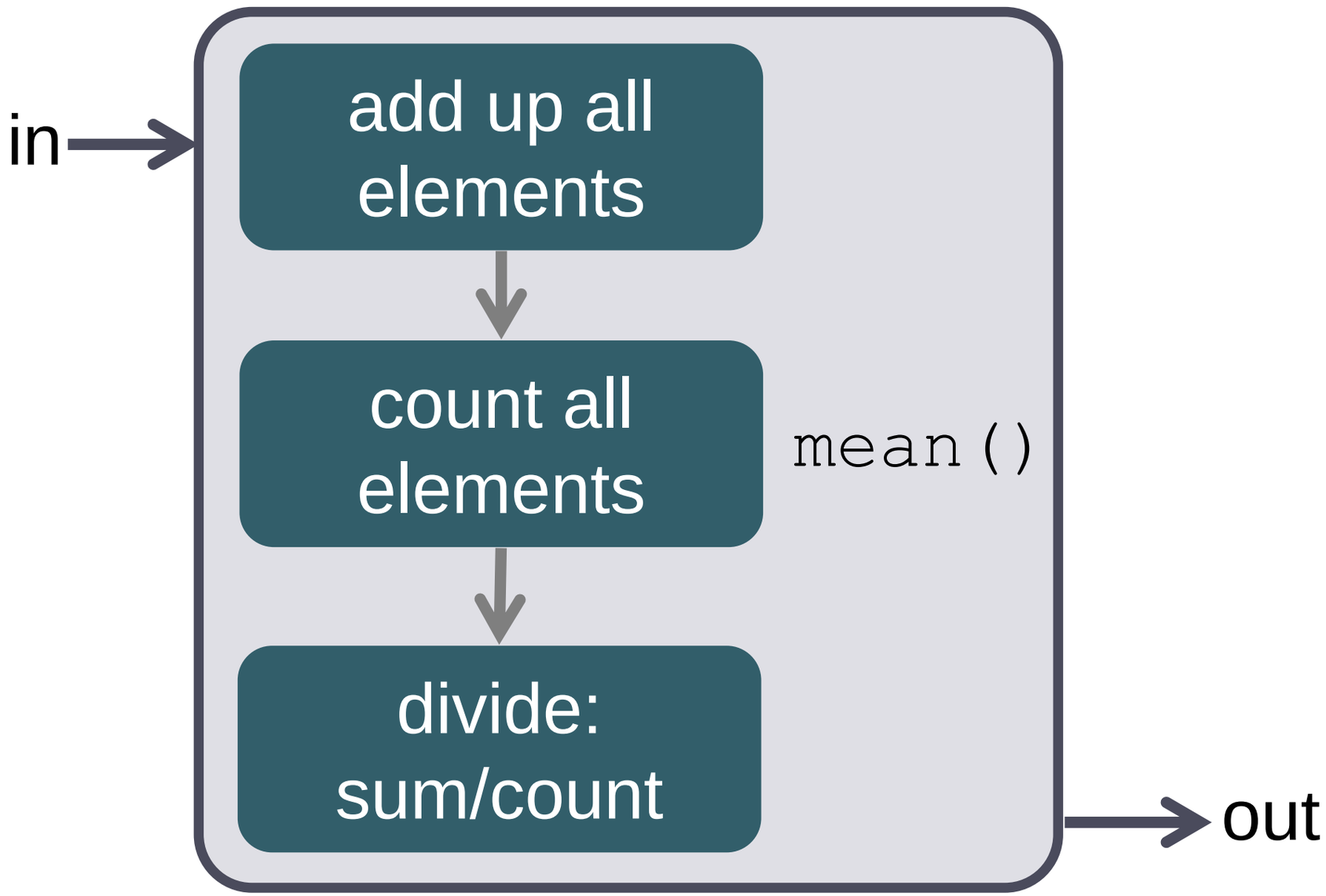
Demonstração de como escrever e executar um script

RStudio > File > New File > R Script



Programação: funções

- Diversas instruções organizadas
- Devem ser capazes de serem repetidas usando diferentes inputs



Programação: funções

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

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

Exercício

1. Escreva uma função que use como input um vetor, e retorna a media deste vetor (você pode utilizar a função `mean()` dentro de sua função).

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

1. Adicione outros elementos em sua função:
   a. Por exemplo, o desvio padrão (`sd()`), o mínimo (`min()`), e o máximo (`max()`) do vetor usado como input.
   b. Coloque o resultado destes três cálculos em um vetor, e use a função `c()` para retornar o resultado da função que está escrevendo.

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



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

Exercício

1. Escreva uma função que use como input um vetor, e retorna a media deste vetor (você pode utilizer a função `mean()` dentro de sua função).

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

1. Adicione outros elementos em sua função:
   a. Por exemplo, o desvio padrão (`sd()`), o mínimo (`min()`), e o máximo (`max()`) do vetor usado como input.
   b. Coloque o resultado destes três cálculos em um vetor, e use a função `c()` para retornar o resultado da função que está escrevendo.

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

Usando a função `apply_calc`

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

> apply_calc(iris$Sepal.Length)
[1] 5.8433333
```

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

> apply_calc(iris$Sepal.Length)
[1] 5.8433333 0.8280661 4.3000000 7.9000000
```

Usando loops para ações repetidas

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

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

Exercício

1. Faça um vetor com todos os nomes do data frame `iris`.
   Dica: use a função `colnames()`.

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

1. Faça um for-loop que itere sobre cada elemento do vetor criado, e mostra na tela cada um destes elementos

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

1. Dentro do for-loop, selecione a coluna correspondente em cada iteração no data frame `iris`, e mostre a media de cada coluna. *Dica: Você deve receber um Warning! Por quê?*

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


}
```

Exercício

1. Faça um vetor com todos os nomes do data frame `iris`.
   Dica: use a função `colnames()`.

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

1. Faça um for-loop que itere sobre cada elemento do vetor criado, e mostra na tela cada um destes elementos

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

1. Dentro do for-loop, selecione a coluna correspondente em cada iteração no data frame `iris`, e mostre a media de cada coluna. *Dica: Você deve receber um Warning! Por quê?*

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

Por que temos um Warning, e como corrigir?

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

Usando if-else

```
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!"
```

Programação: 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!"
```

Exercício

1. Faça um novo for-loop como o do exercício anterior. Dentro deste for-loop, inclua um if-statement de forma que a função `mean()` apenas seja realizada em vetores numéricos. *Dica: veja a função* `is.numeric()`.

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

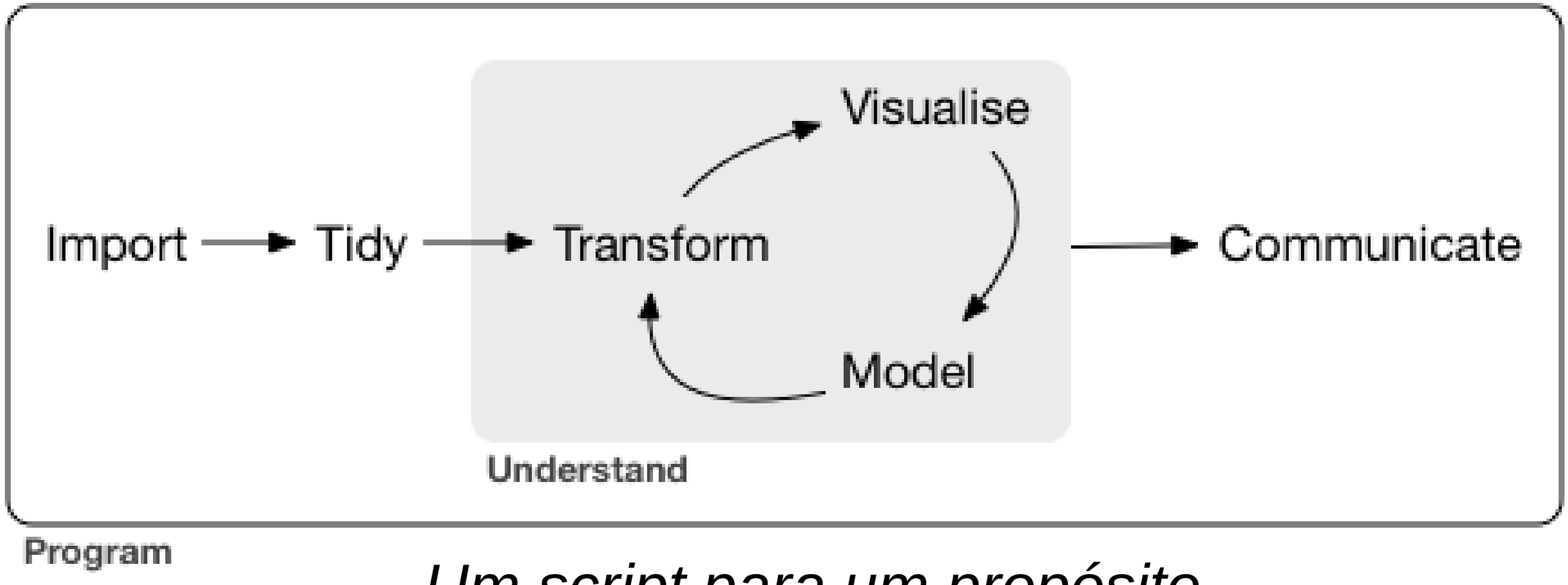
Exercício

1. Faça um novo for-loop como o do exercício anterior. Dentro deste for-loop, inclua um if-statement de forma que a função `mean()` apenas seja realizada em vetores numéricos. *Dica: veja a função* `is.numeric()`.

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

Workflow de ciência dos dados: scripting é fundamental

- Scripting combina diversos comandos em um conjunto de instruções definidas.
- Um script é um código que pode ser **salvo, re-utilizado**, **compartilhado** e **publiacado**
- É uma etapa essencial para uma análise de dados reproduzível



Um script para um propósito

Escrevendo um script: cabeçalho

```
## Date: 7 November 2018
## Author: S. Tudent
## This script was written as part of the R course
## "Introduction to R & Data"
```

Escrevendo um script: pacotes

```
## Date: 7 November 2018
## Author: S. Tudent
## This script was written as part of the R course
## "Introduction to R & Data"

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

Escrevendo um script: funções customizadas

```
## Date: 7 November 2018
## Author: S. Tudent
## This script was written as part of the R course
## "Introduction to R & Data"

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

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

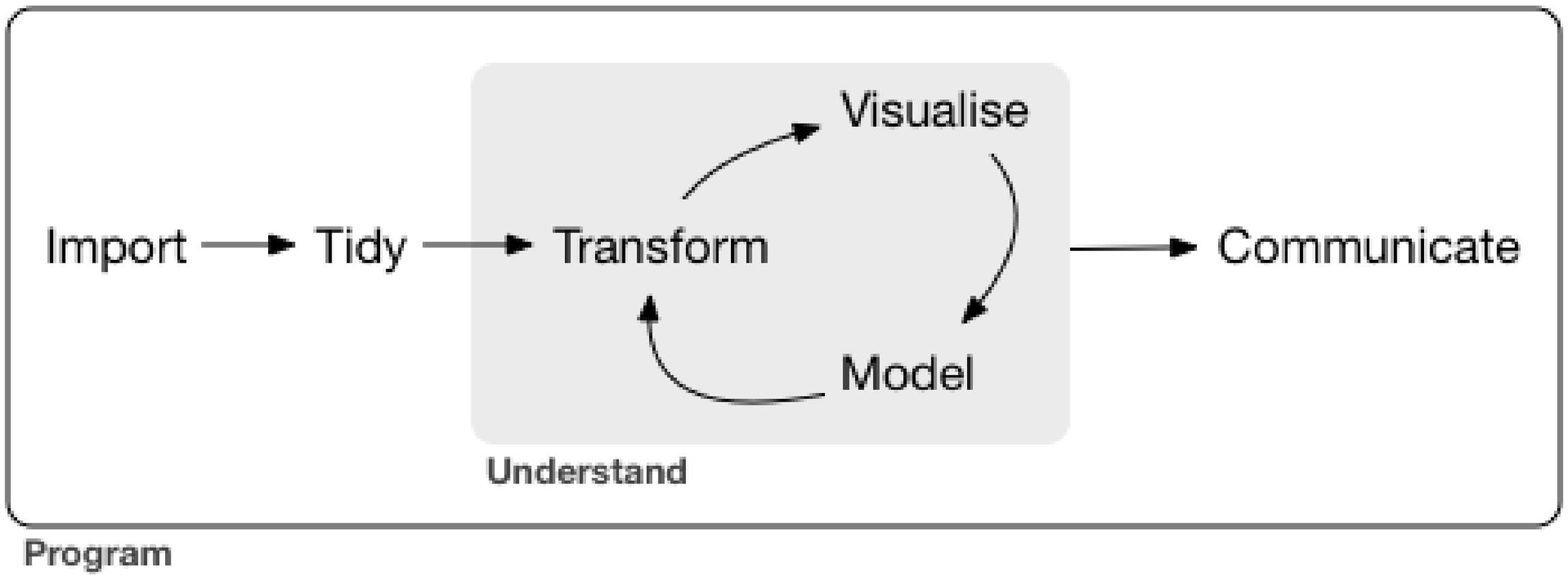
Escrevendo um script

```
## Date: 7 November 2018
## Author: S. Tudent
## This script was written as part of the R course
## "Introduction to R & Data"
```

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

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

Workflow de uma análise de dados



"Tidy data" (dados limpos, arrumados) garante que o processamento seja eficiente e reproduzível.

Tidy data é fácil de manipular, modelar e visualizar

Tidy data

- Cada **variável** é uma coluna e contém **valores**
- Cada **observação** é uma linha
- Cada **unidade observacional** é uma tabela

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

- Cada **variável** é uma coluna e contém **valores**
- Cada **observação** é uma linha
- Cada **unidade observacional** é uma tabela

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

- Cada **variável** é uma coluna e contém **valores**
- Cada **observação** é uma linha
- Cada **unidade observacional** é uma tabela

multiple observations per row

values in column names

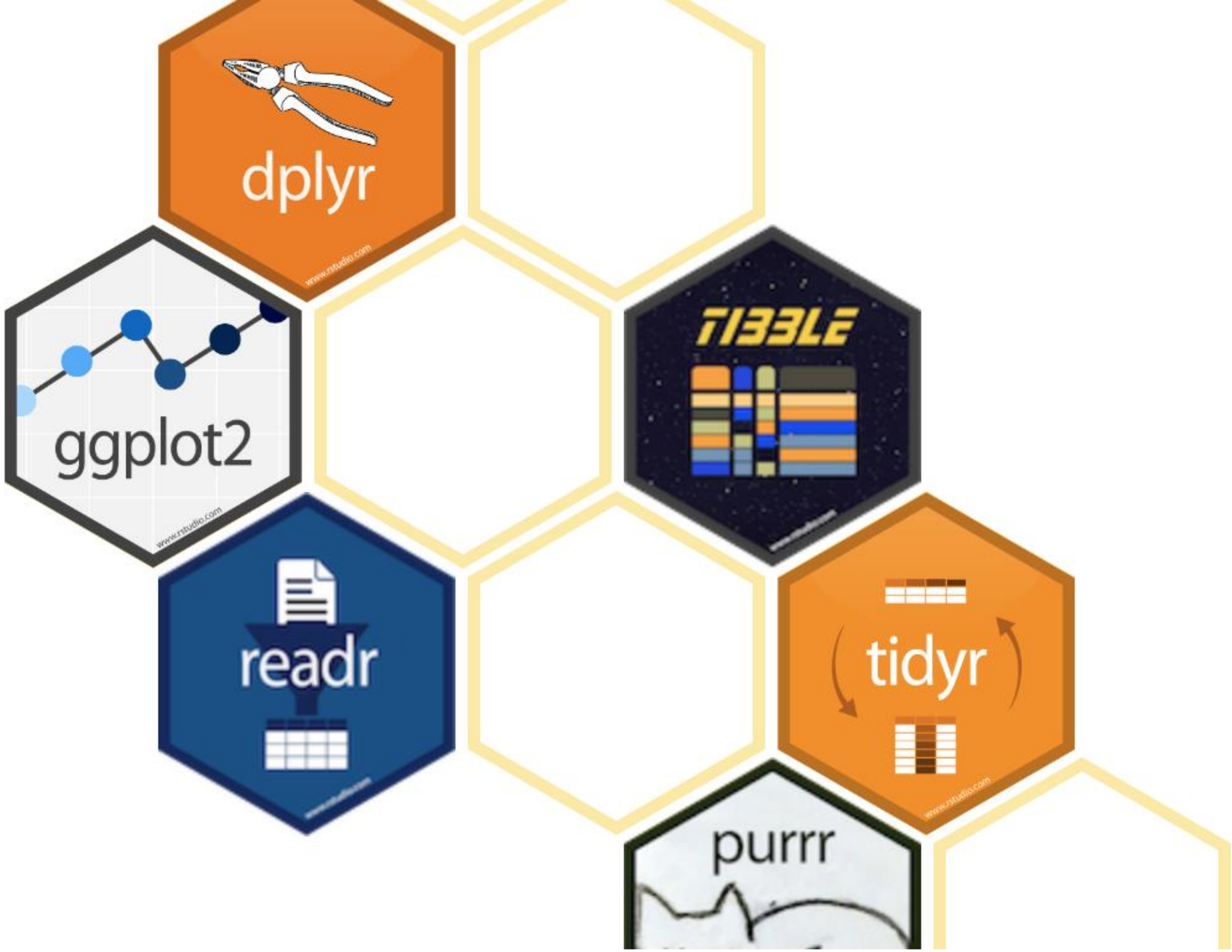
Patient	Temp	Med_A	Temp_A	Med_B	Temp_B
122030	37.0	300	37.1	NA	NA
122021	38.2	200	38.1	85	36.5
124500	38.1	300	38.0	NA	NA
126098	39.1	NA	NA	100	36.8

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

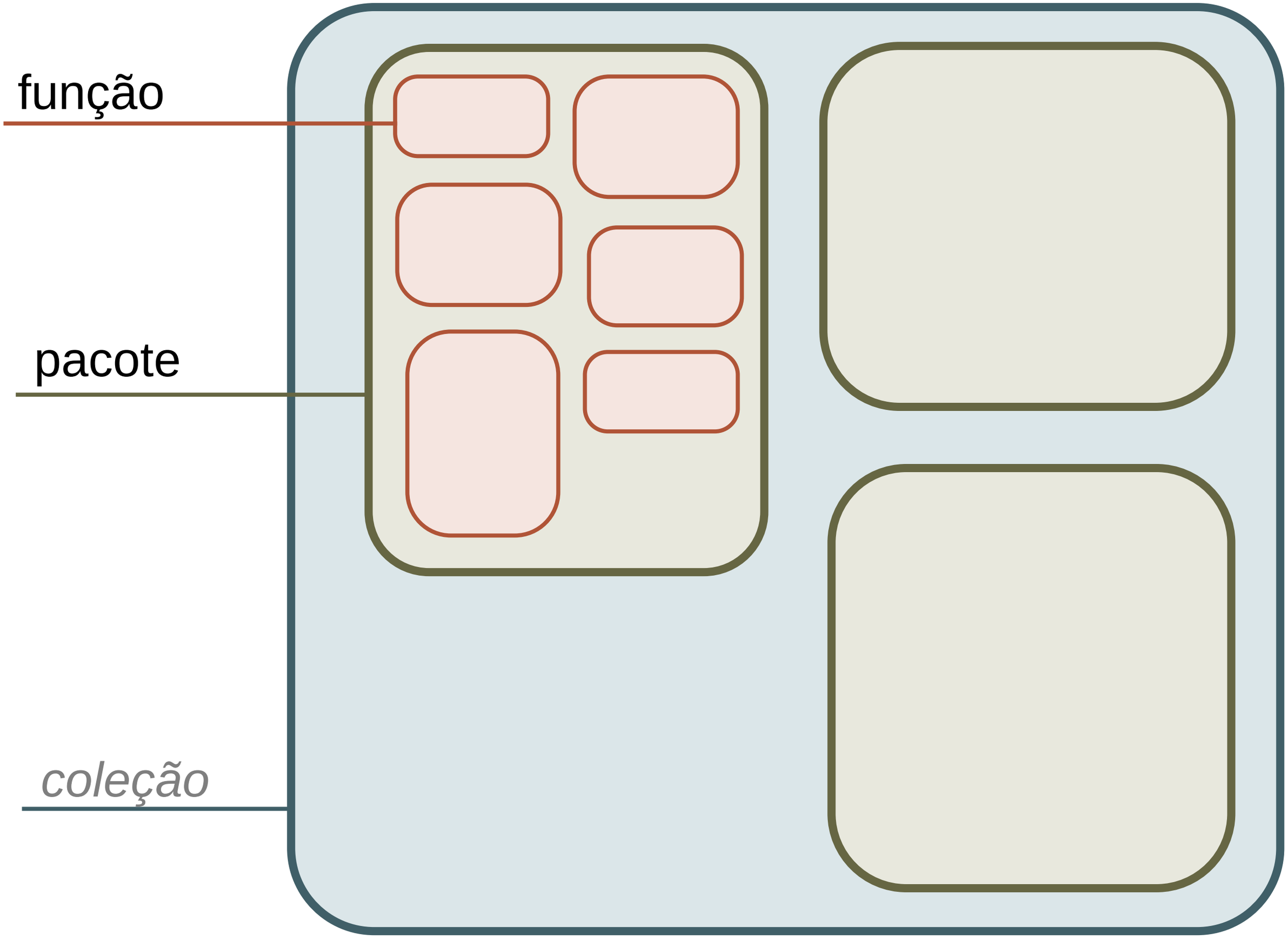


Tidyverse

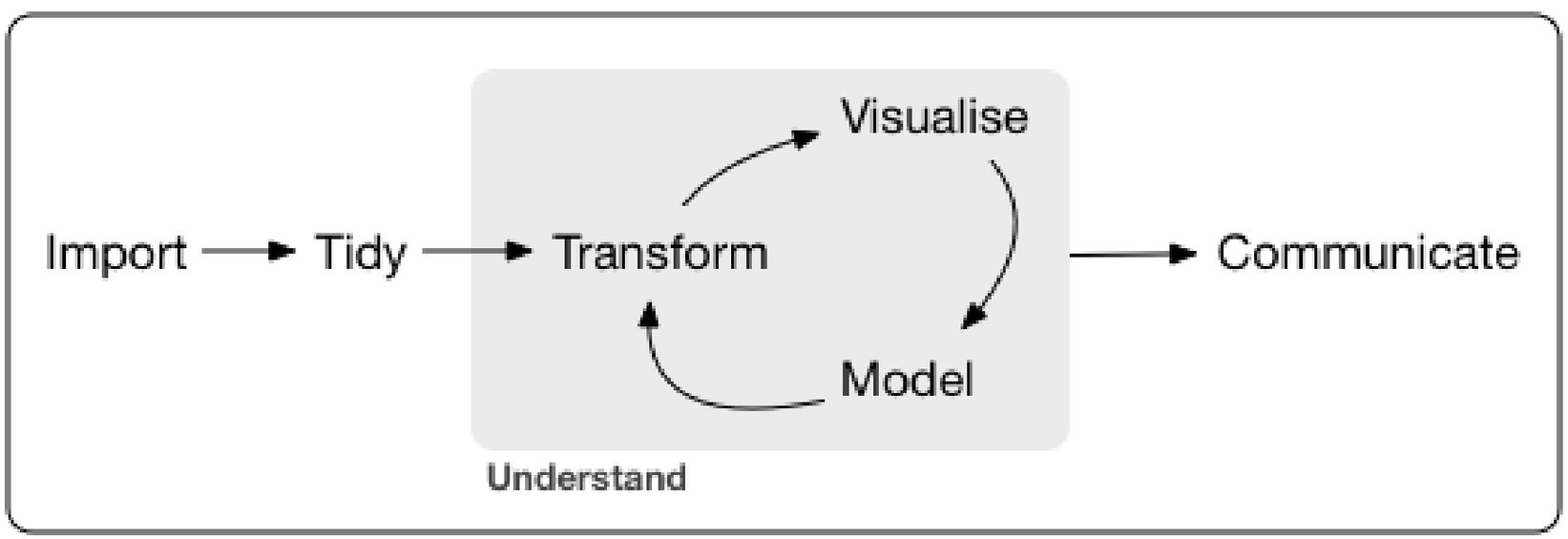
Modern R for data science

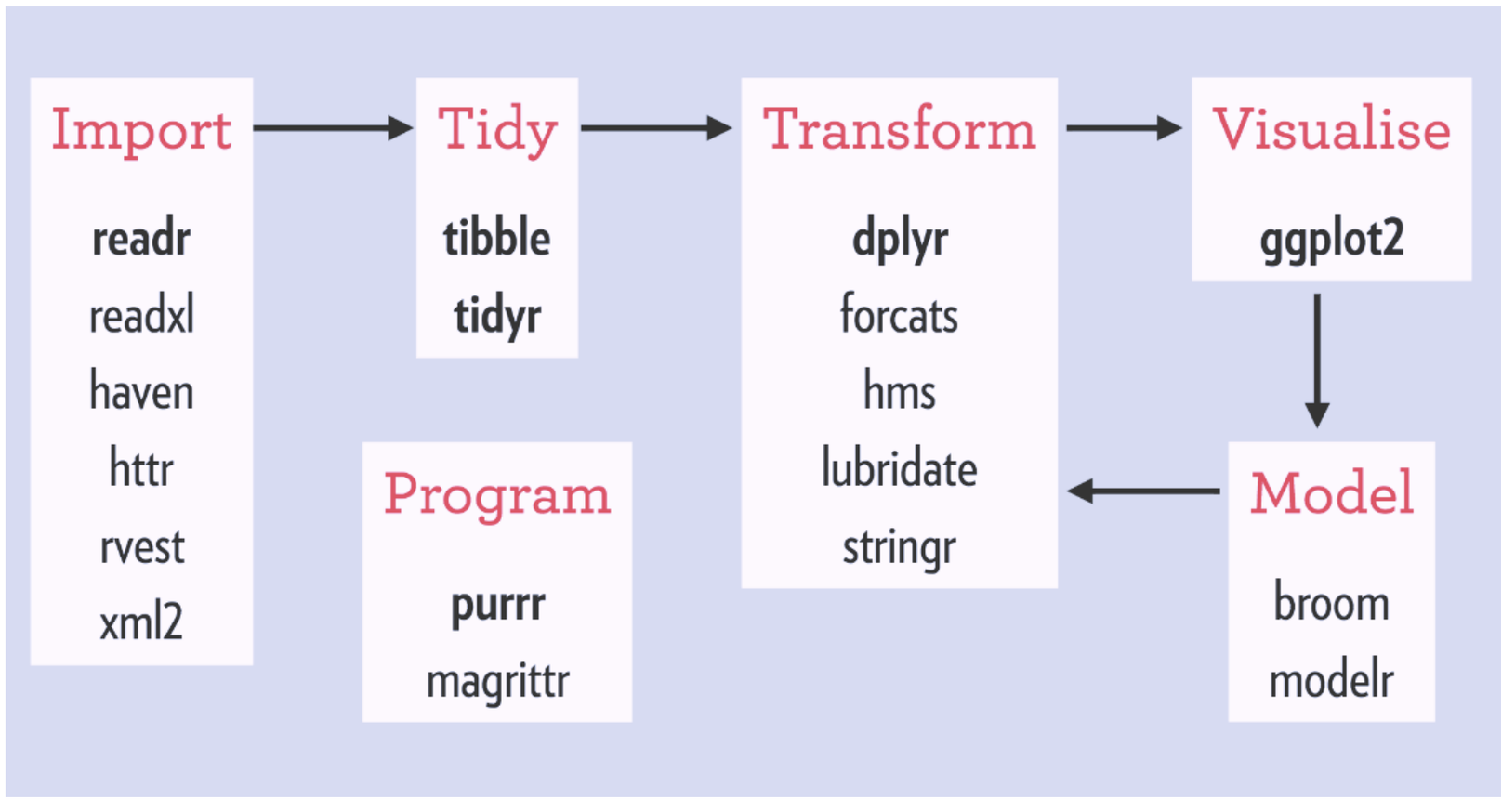
"The tidyverse is an opinionated collection of R packages **designed for data science**. All packages **share an underlying design** philosophy, grammar, and data structures."

– tidyverse.org (2018)


função
pacote
coleção

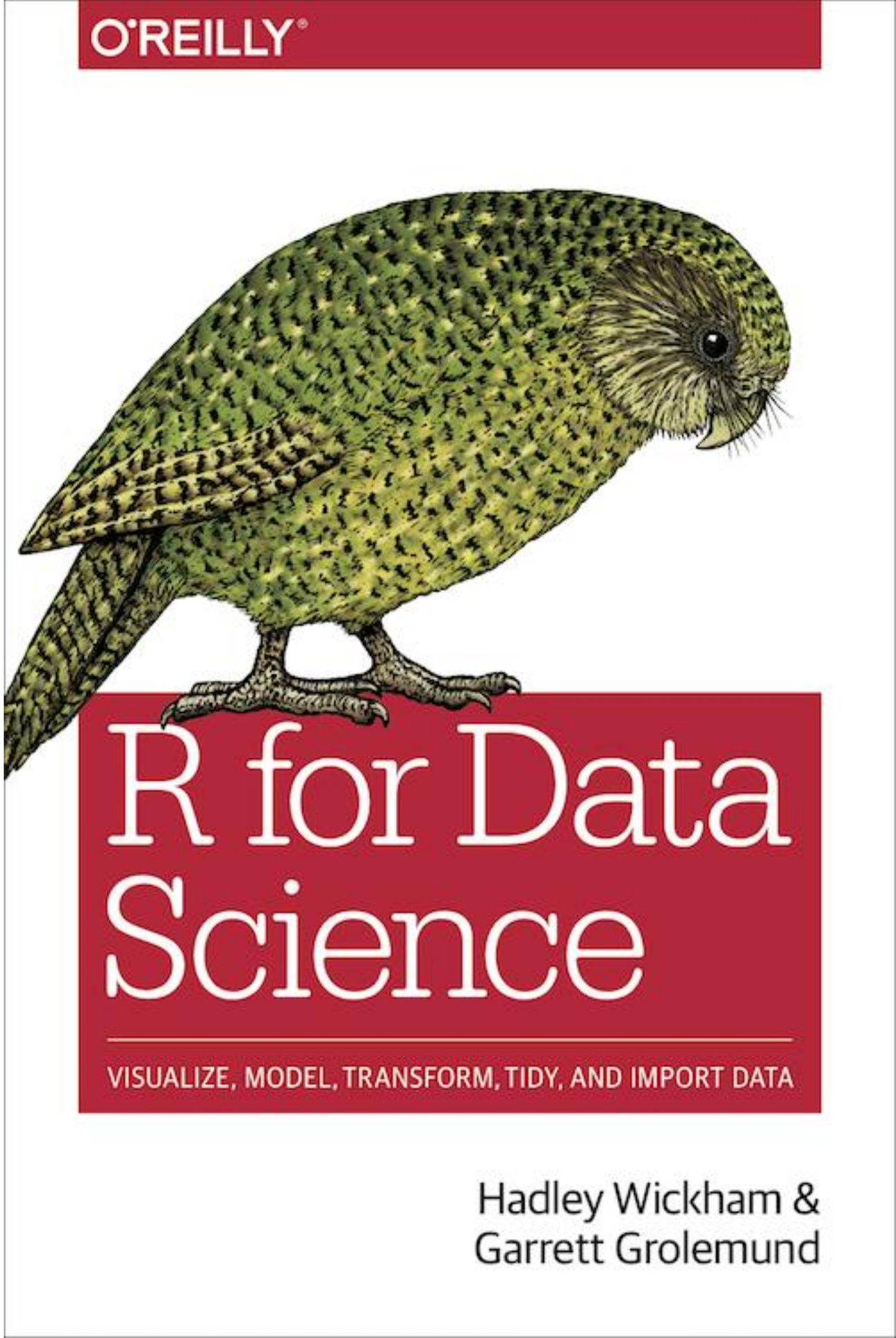
Tidyverse: conectando pacotes para todas as etapas da análise de dados





Aprendendo o 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/

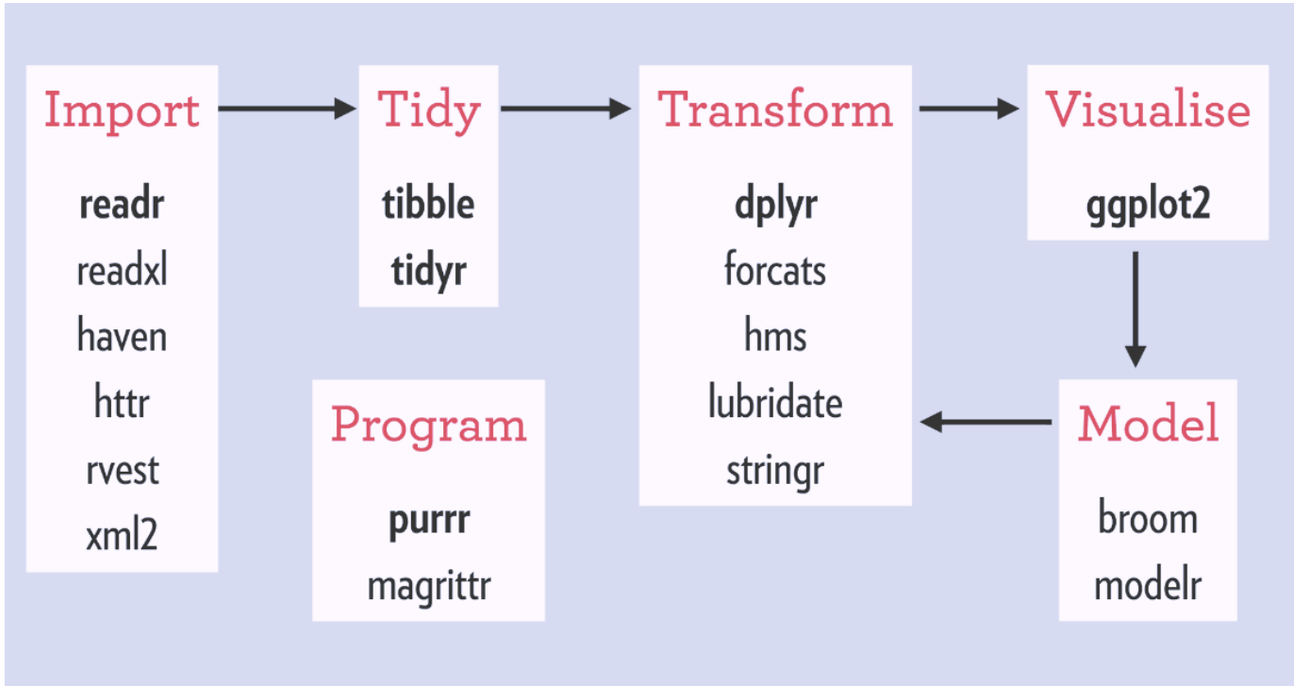


Próximos passos

- Explicação teórica
- Exercícios no RStudio
  - 2-3 exercícios básicos
  - Exercícios opcionais para praticar
- Outros datasets
- Exercícios apresentados no formato RMarkdown

Próximos passos

- Introdução ao *tidyverse*, R*markdown*, e *tibble*.
- Carregar e salvar dados com o *readr*
- Visualização de dados com o ggplot2
- Manipulação de dados com *tidyr* and *dplyr*



Console, Scripts e Rmarkdown

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

Do código para o documento

- **Combinando código com uma narrativa (R + markdown)**

- Análises podem ser automaticamente salvas como um relatório, um produto pronto para visualizar, um blog, uma página na Web, e até um aplicativo, ou apenas servir como um “caderno de laboratório”

Rmarkdown: exemplo

Untitled.Rmd

Knit

```
1  ---
2  title: "Demo"
3  author: "S. Tudent"
4  date: "6/12/2019"
5  output: html_document
6  ---
7
8  ```{r setup, include=FALSE}
9  knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 # Title
13
14 Some text about your project
15
16 ## Section
17
18 More text about your project
19
20 ```{r}
21 max(iris$Petal.Length)
22 plot(iris$Sepal.Length,iris$Sepal.Width)
23 ```
24
```

24:1 # Section R Markdown

Demo

S. Tudent

6/12/2019

Title

Some text about your project

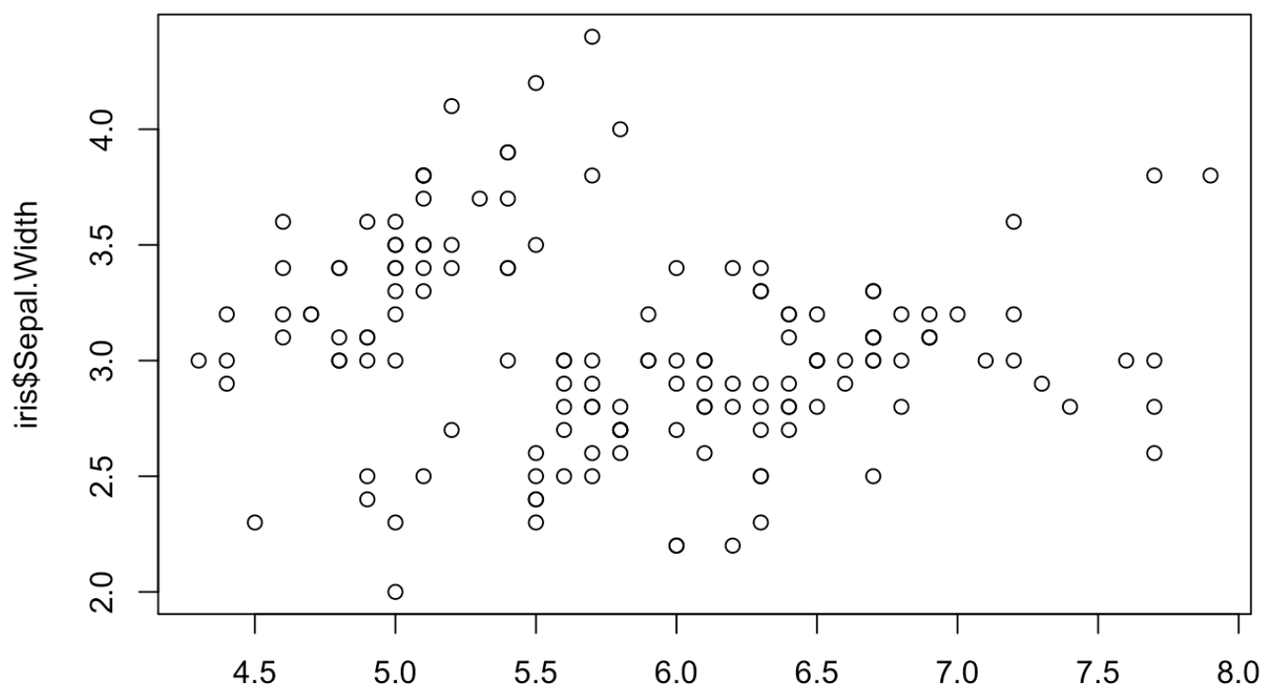
Section

More text about your project

```
max(iris$Petal.Length)
```

```
## [1] 6.9
```

```
plot(iris$Sepal.Length,iris$Sepal.Width)
```



Exercício: abra o exercício em Rmarkdown

1. RStudio > File > Open Project > introduction-to-R-and-data-github.Rproj

2. Selecione o arquivo modernR_exercises.Rmd no canto direito

3. Atualize o cabeçalho com o seu nome

Rode o primeiro “code chunk”

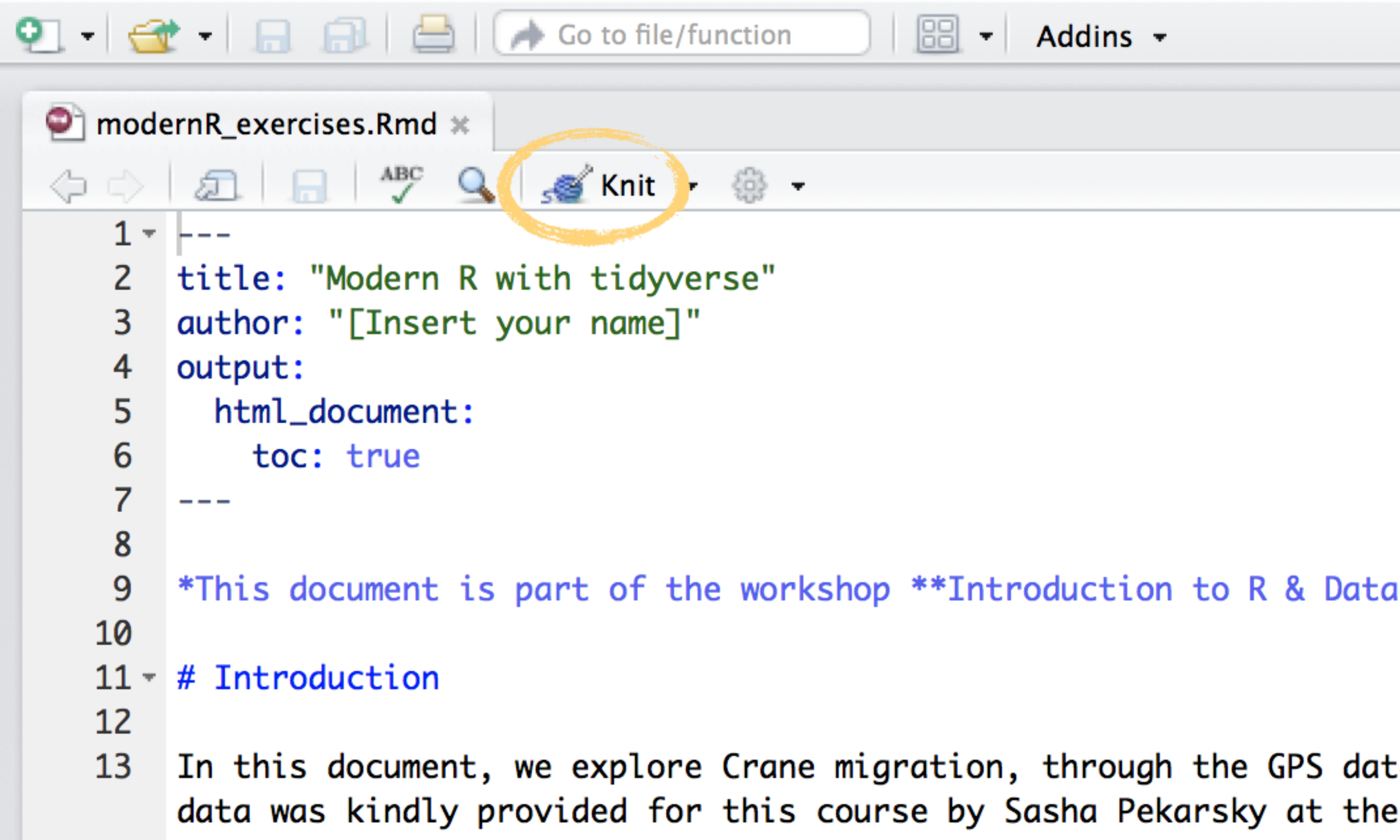
```
## Technical requirements

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()
```

Vá no botão “Knit” (tricotar)



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: convertendo data.frame para 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: convertendo tibble para 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
```

Carregando e salvando dados

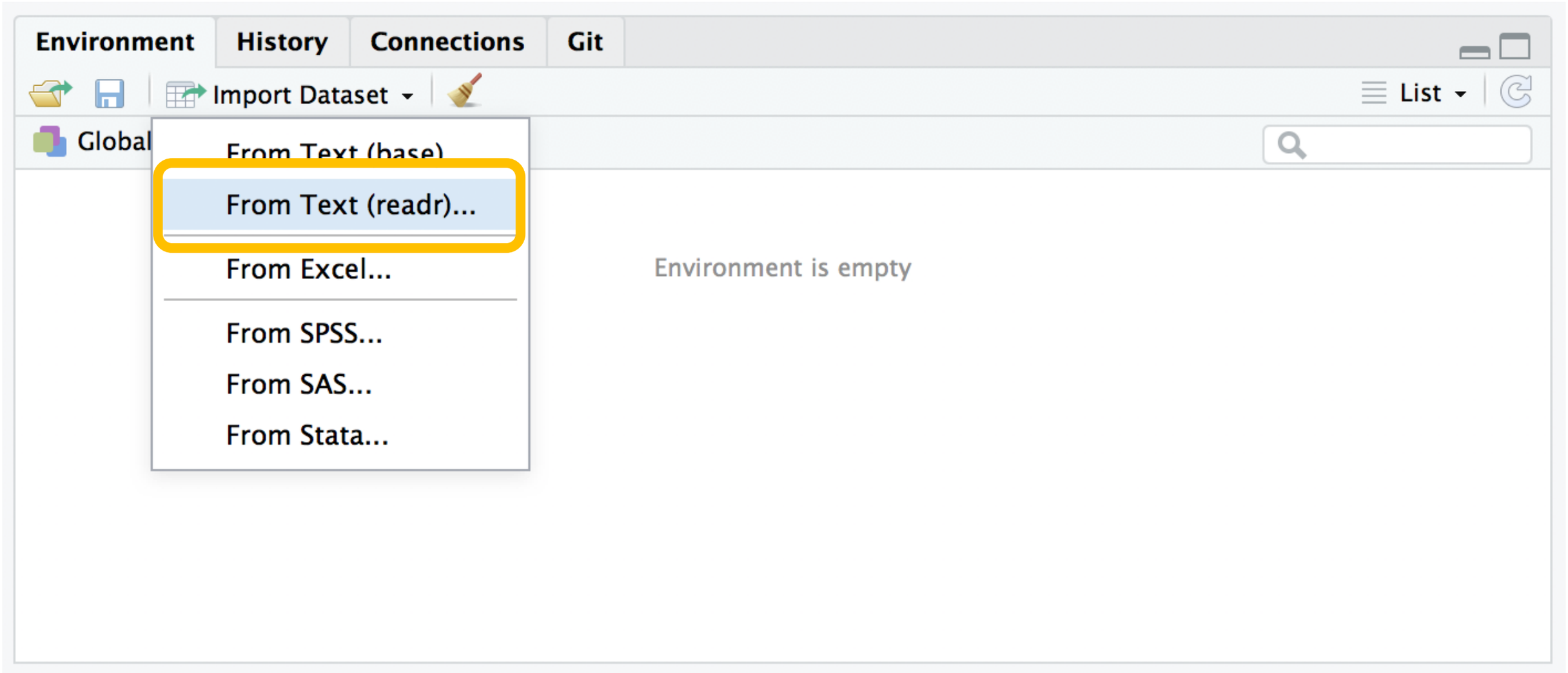
Fundamentos do readr

Dados em arquivos de texto (“flat” files)

- Formato mais comum: Comma-separated values (CSV)
- Delimiter-separated values (also stored as CSV)
- Delimitadores mais comuns:
  - 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
```

Carregando dados pelo RStudio



Carregando dados pelo RStudio

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: Default

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

Lendo arquivos de texto

```
> library(readr)
> data_iris <- read_delim("data/iris.csv", delim=',')
```

Lendo arquivos de texto

```
> 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()
)
```

Lendo arquivos de texto

```
> 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()
)
```

Dataset do exercício

- Dados de GPS de 39 garças durante a migração de outono de 2017
- Dados: longitude, latitude, velocidade, status to GPS…
- Cortesia de Sasha Pekarsky da Hebrew University of Jerusalem, Israel
- Disponível no diretório “data”

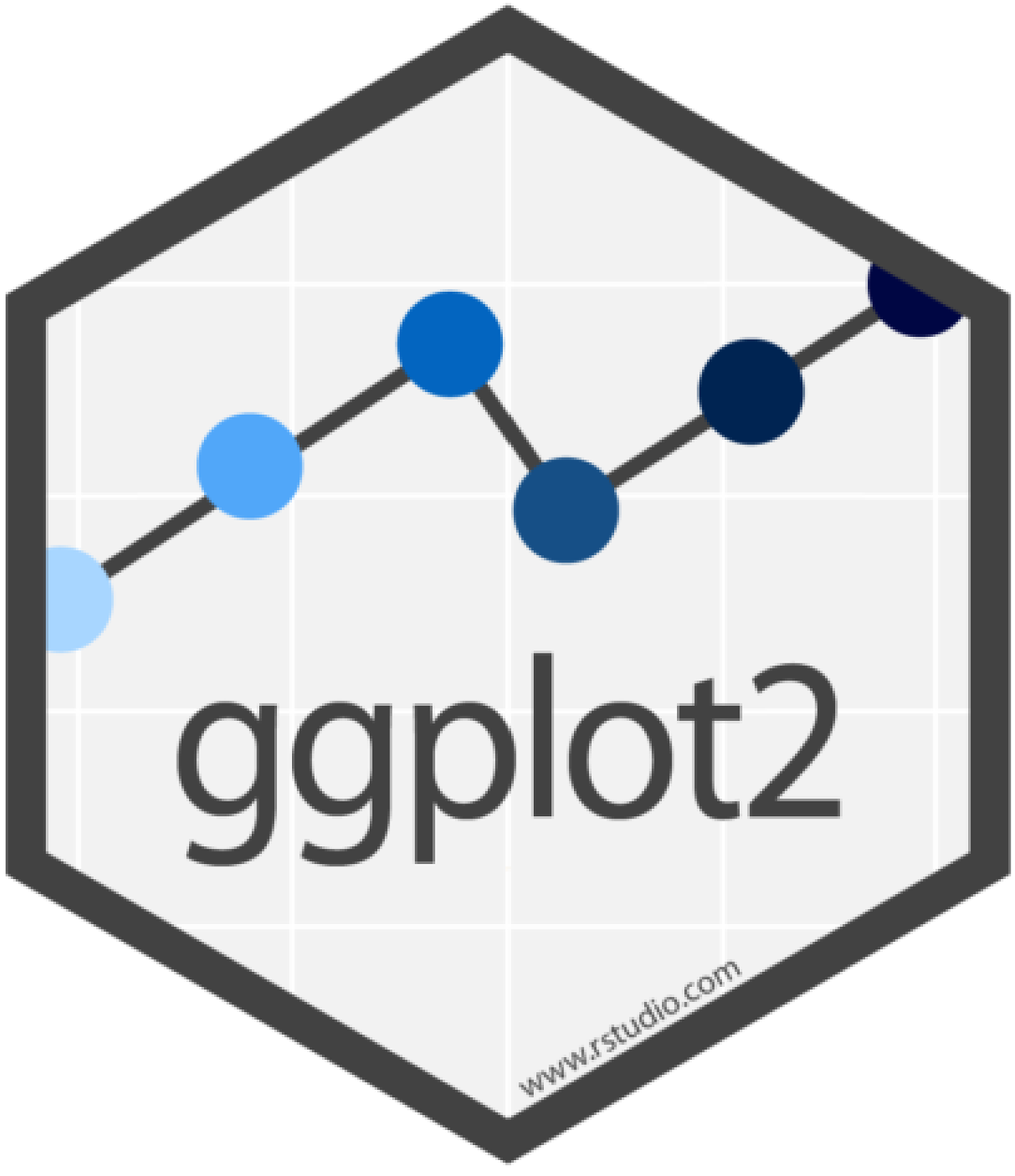
Exercício prático

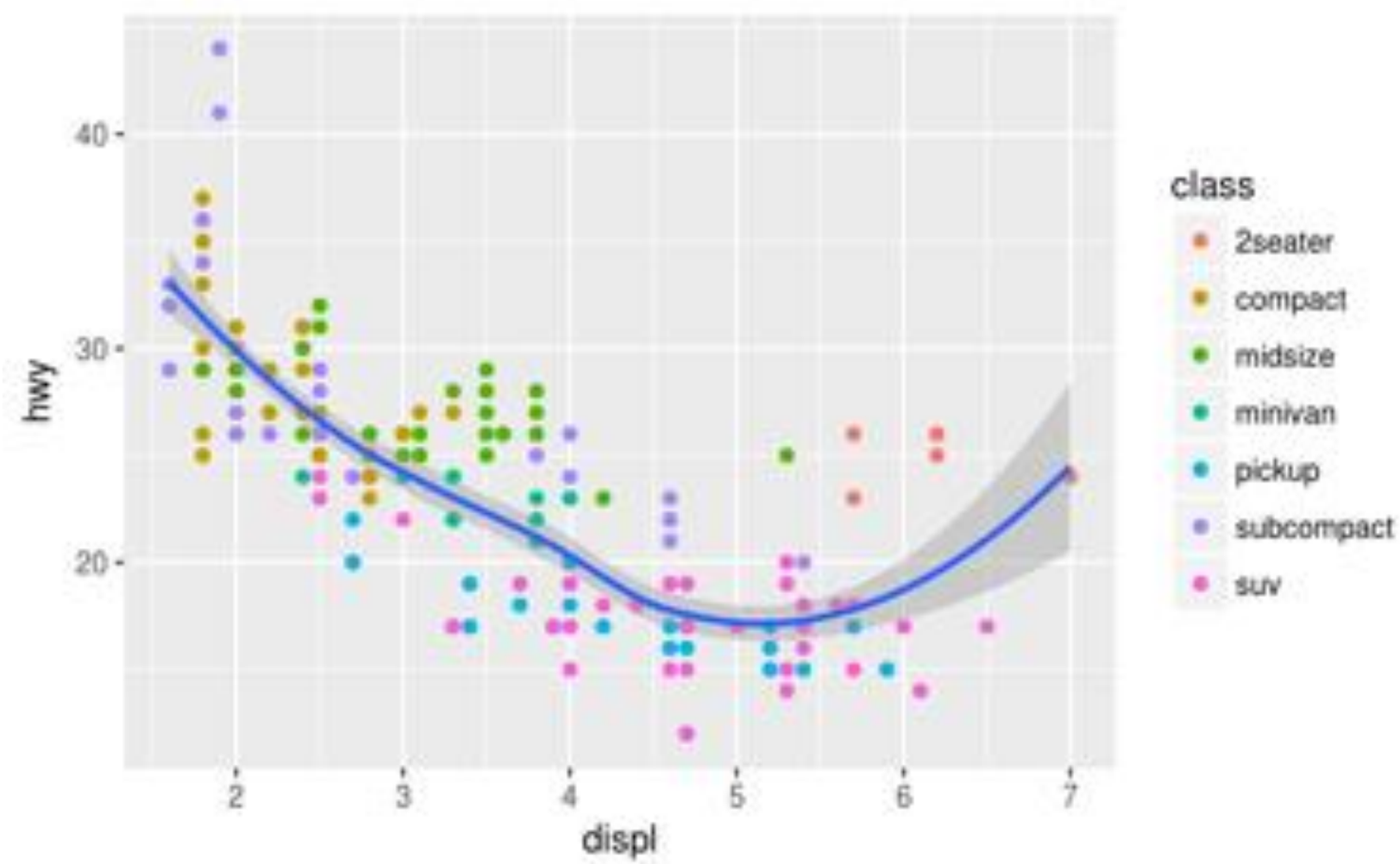
Fazer os exercícios 1-I e 1-II.

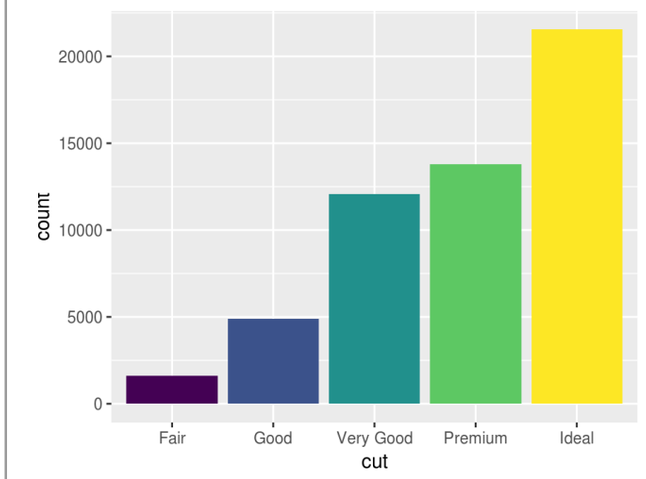
Caso tenha tempo, tente fazer um exercício opcional, ou explore a leitura recomendada

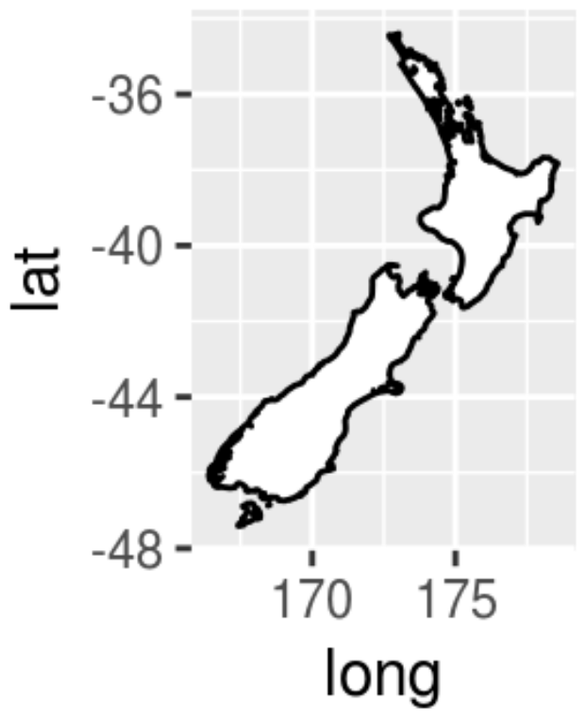
Visualização de dados

Fundamentos do ggplot2







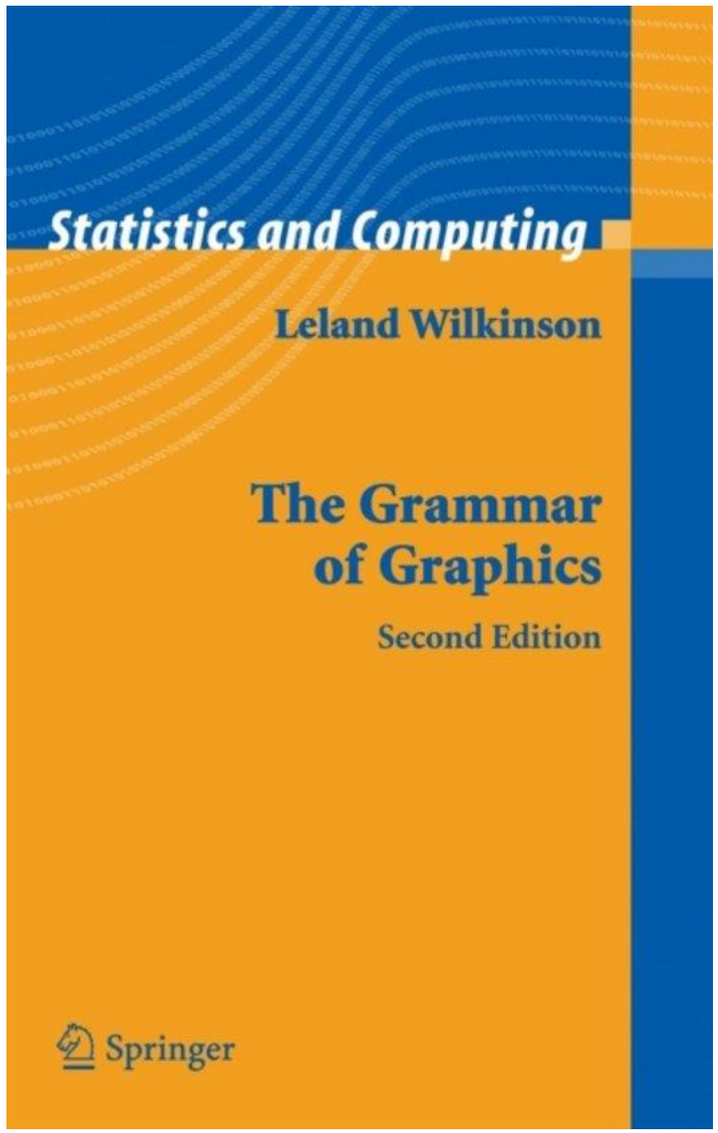


Exemplos de visualização gráfica usando o ggplot2

Visualização de dados com o ggplot2

- **ggplot2** é um pacote de visualização muito útil disponível no R
- Sintaxe simples, fácil de aprender e gráficos que podem ser usados em publicações
- Desenvolvido e mantido por Hadley Wickham
- Baseado no livro: The Grammar of Graphics (Wilkinson, 2005)

“Gramática dos gráficos”

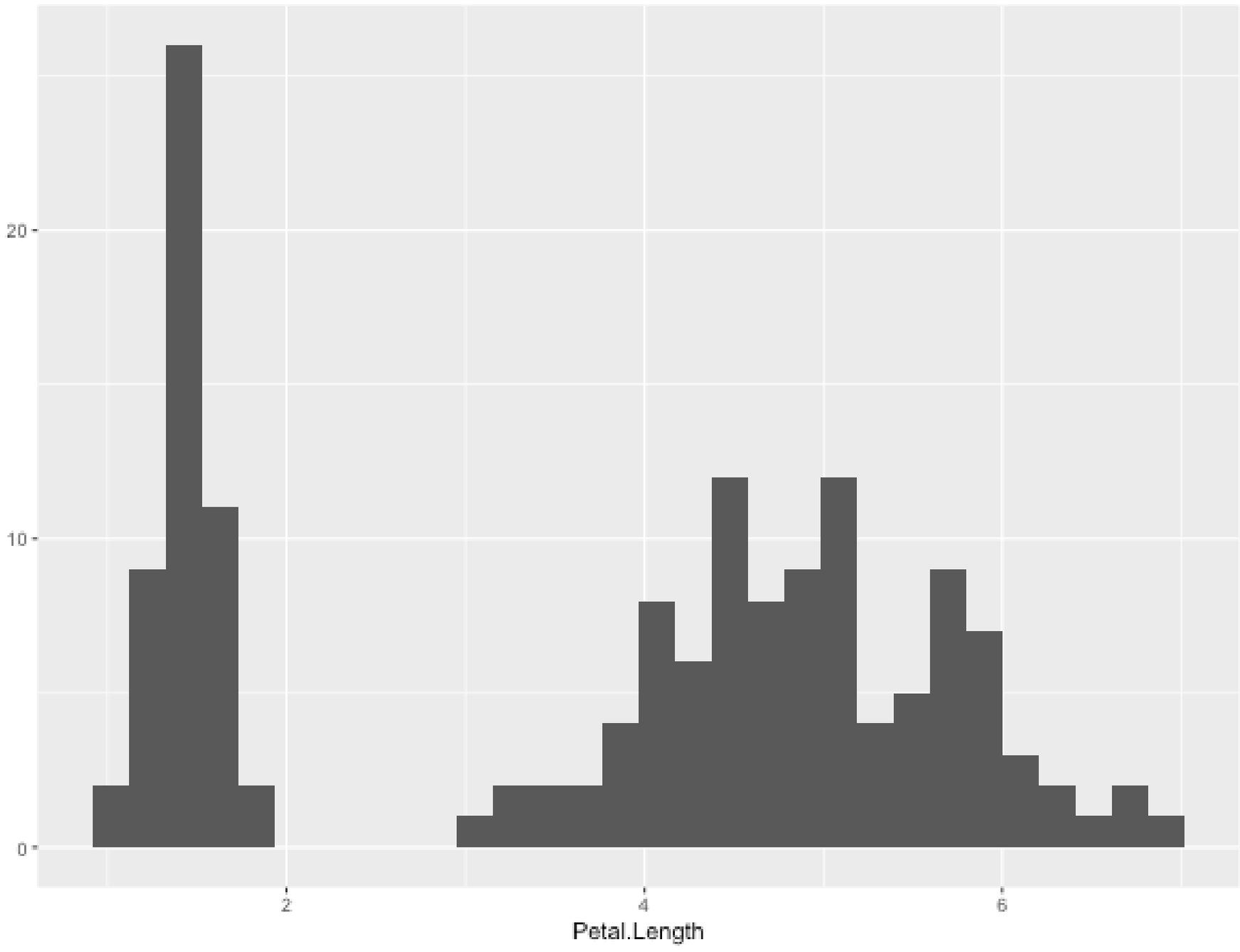


Data	Variáveis em um tibble ou data.frame
Aesthetics	Eixos x e y, cores, tamanho dos pontos, transparência, formas
Geometries	Tipo de gráfico: ponto, linha, barra, histograma

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

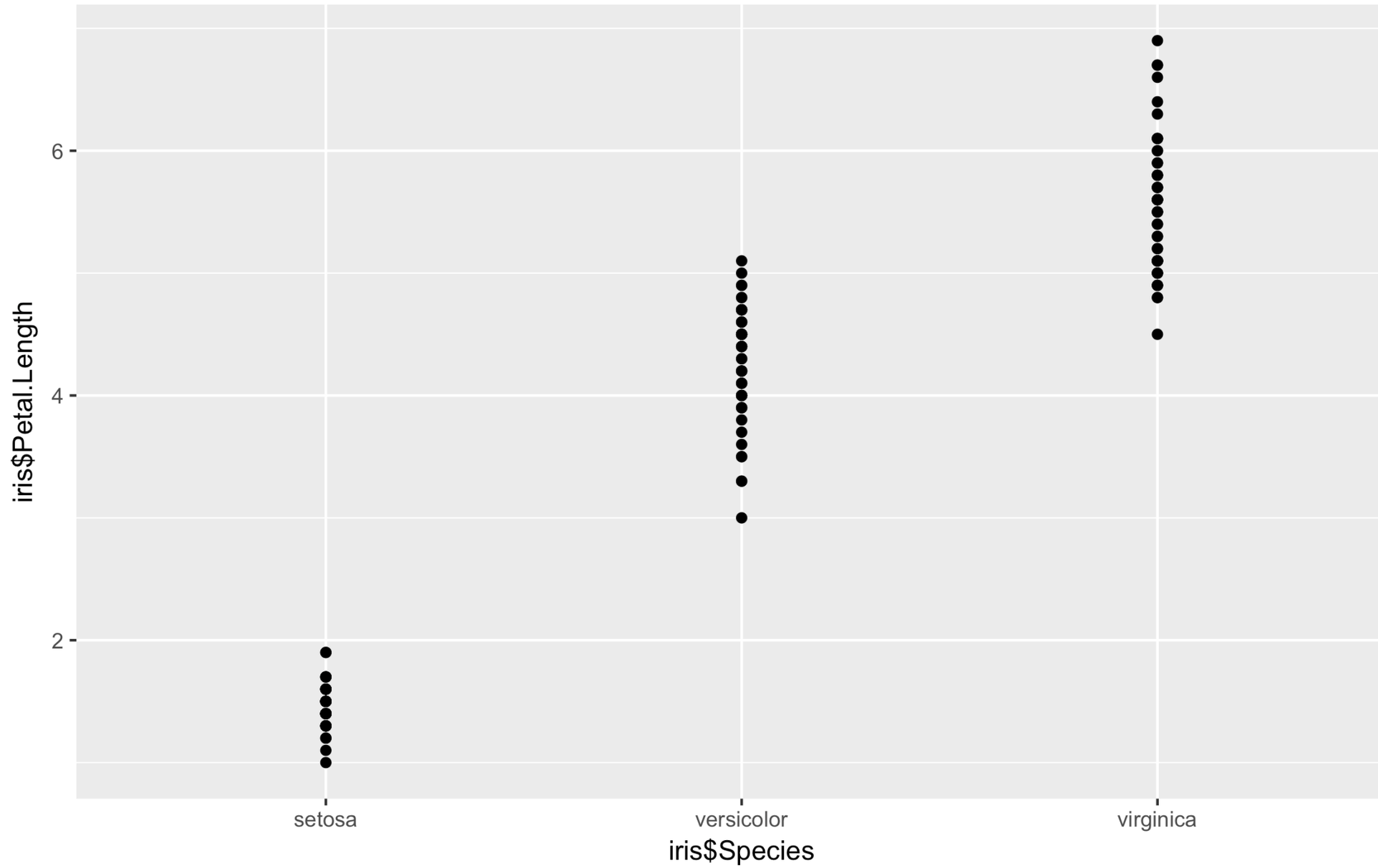
Alguns gráficos rápidos

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



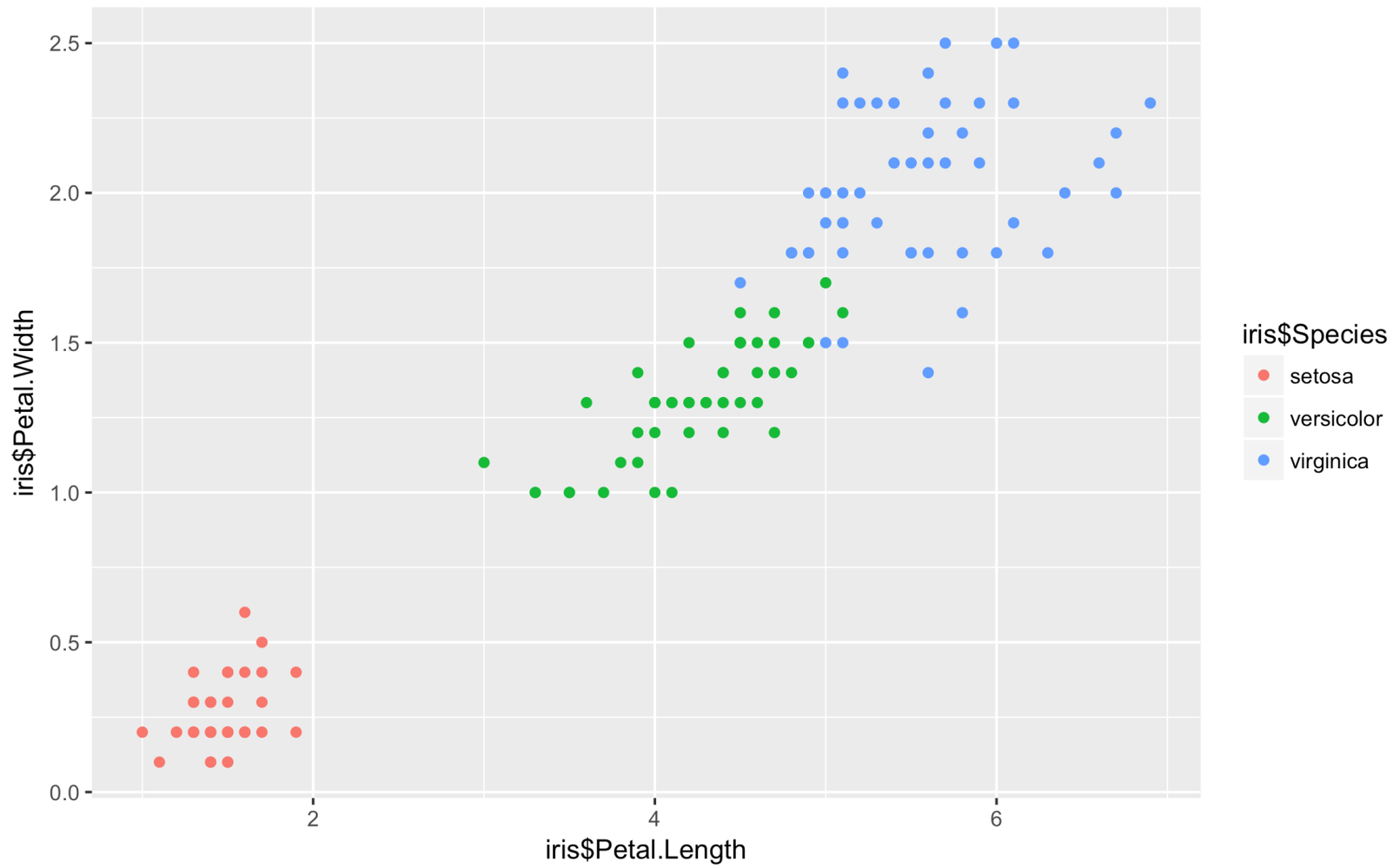
Alguns gráficos rápidos

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



Alguns gráficos rápidos

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



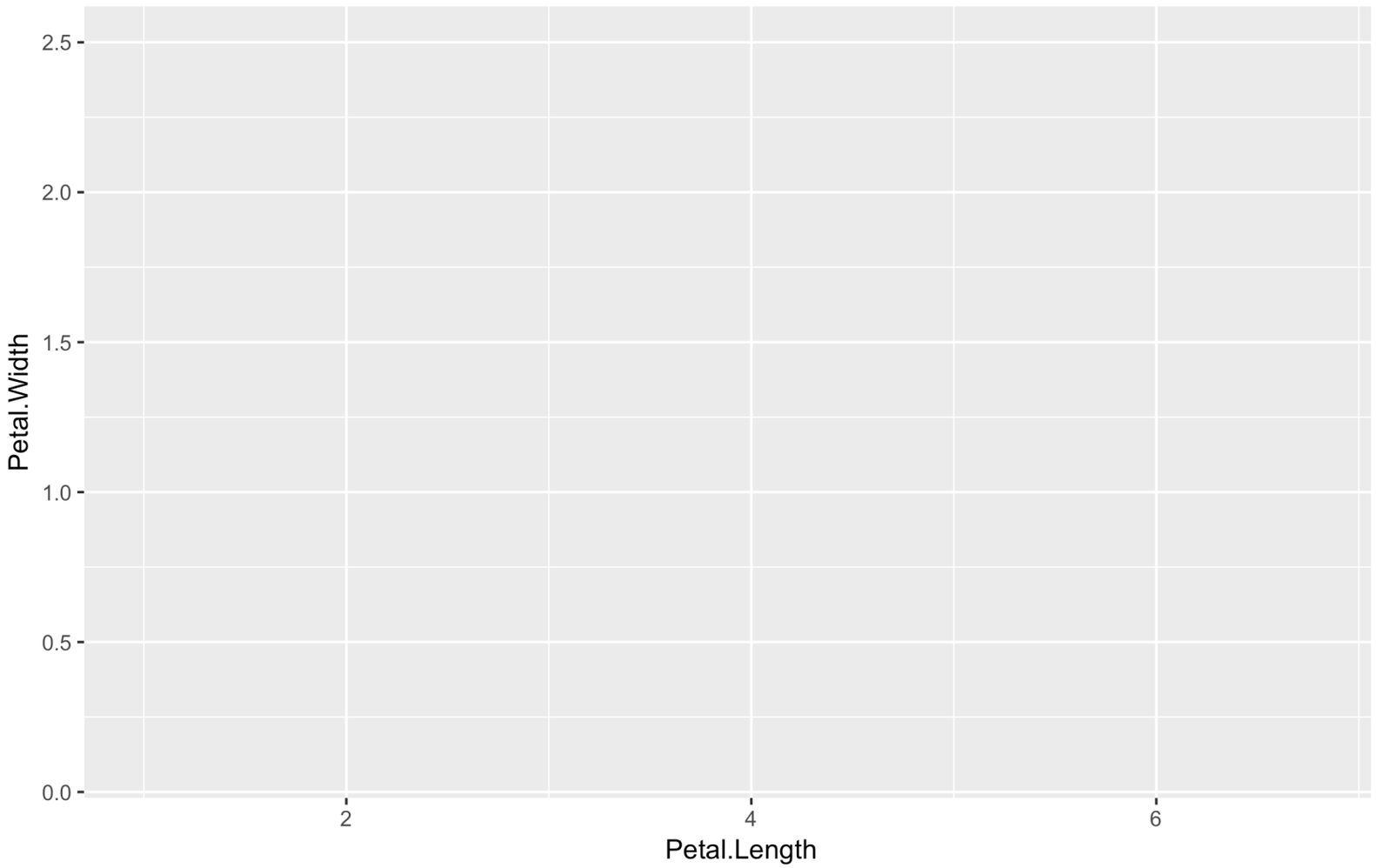
ggplot (grammar of graphics)

ggplot: Data, aesthetics, geometrics

```
> ggplot(iris)
```

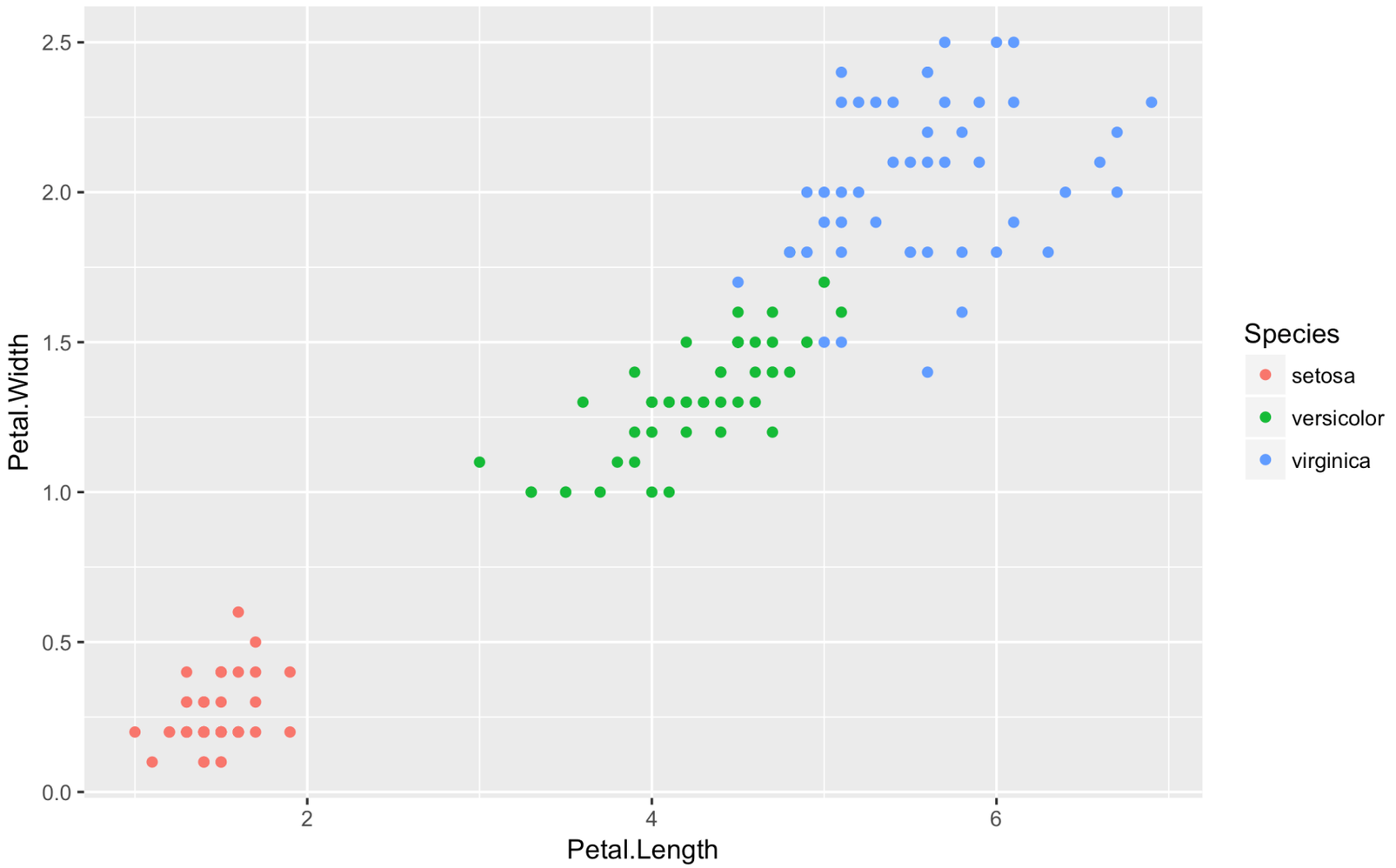
ggplot: Data, aesthetics, geometrics

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



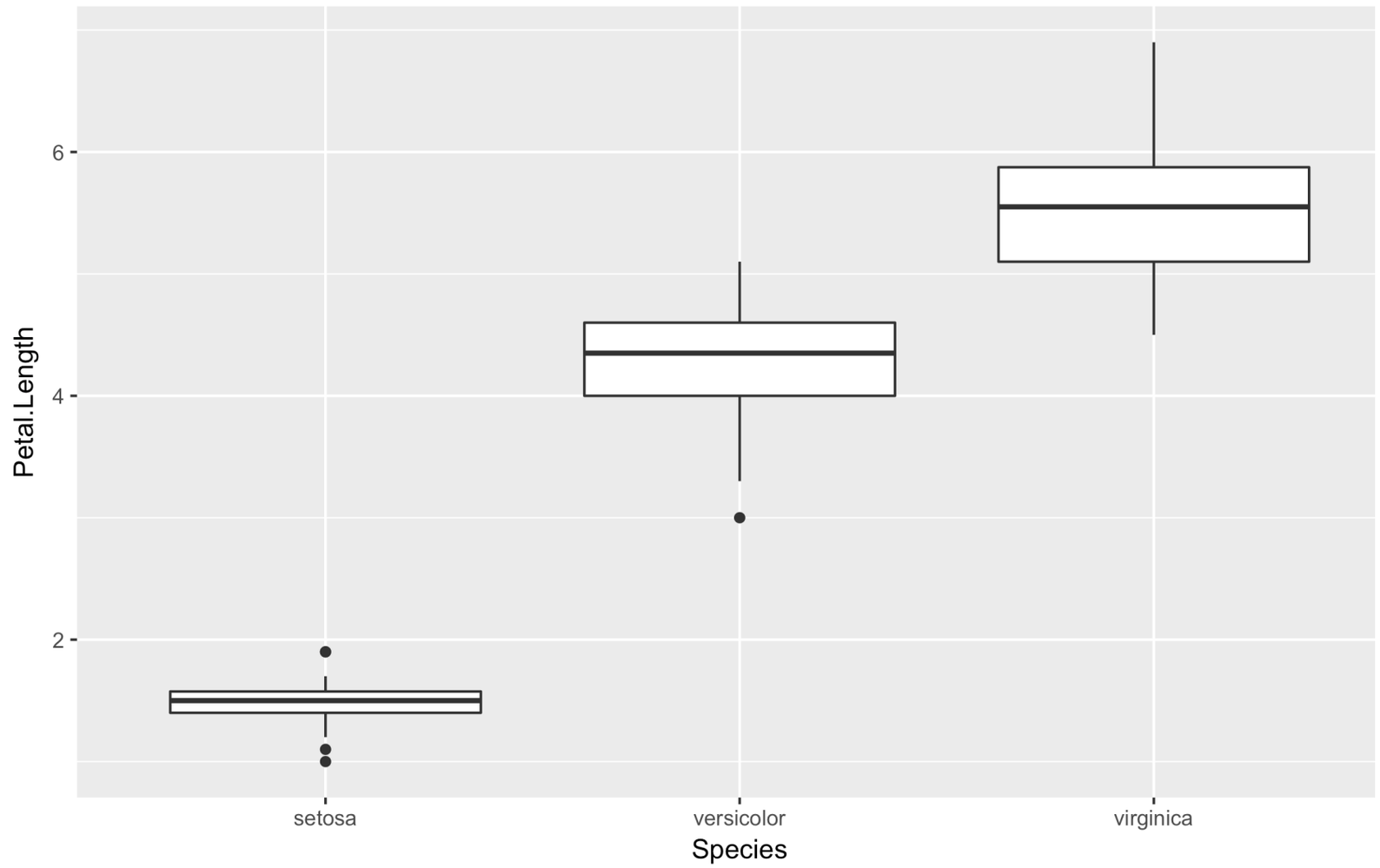
ggplot: Data, aesthetics, geometrics

```
> 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()
```



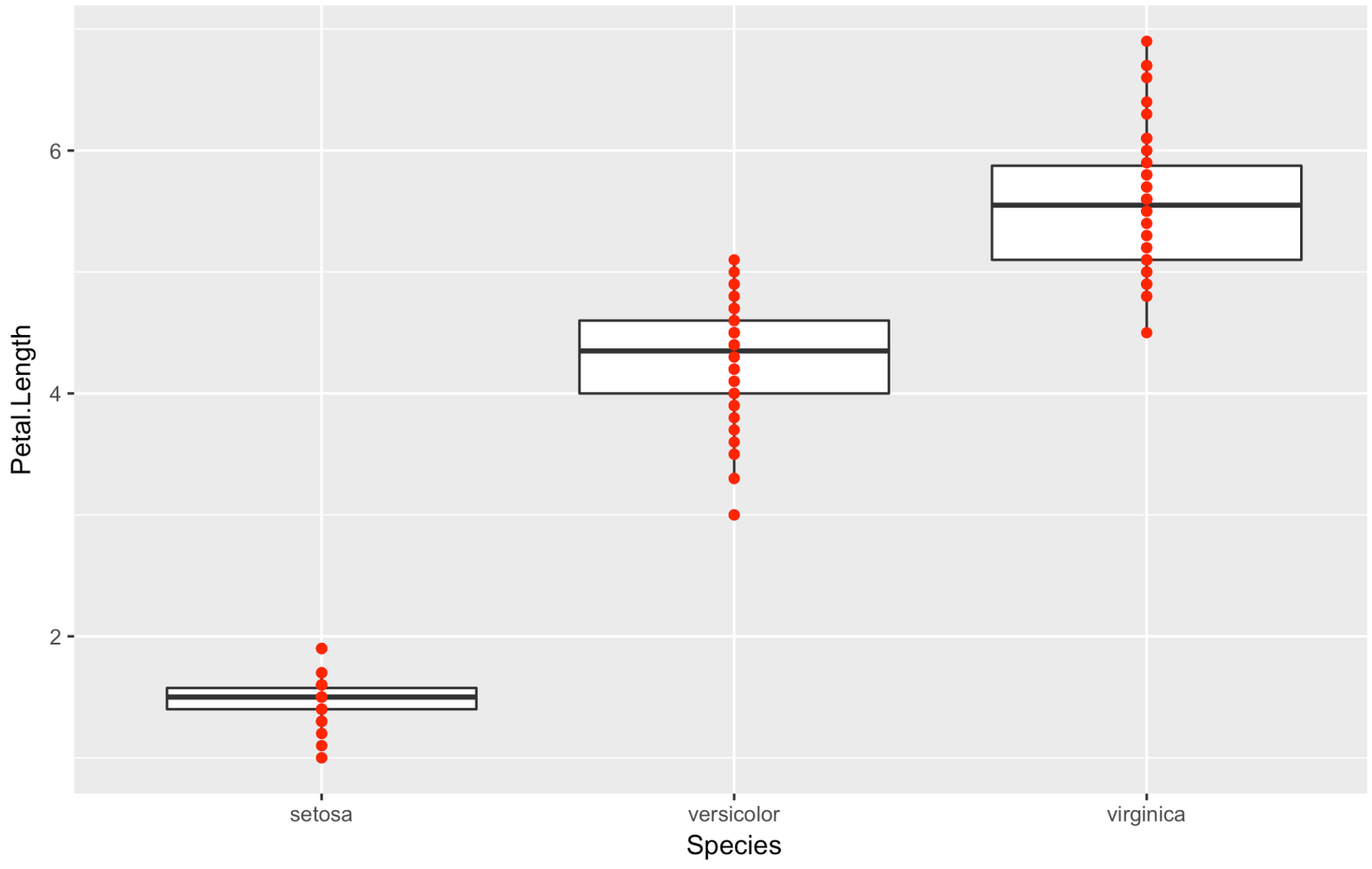
ggplot: Várias camadas

```
> 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: Camadas estatísticas

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

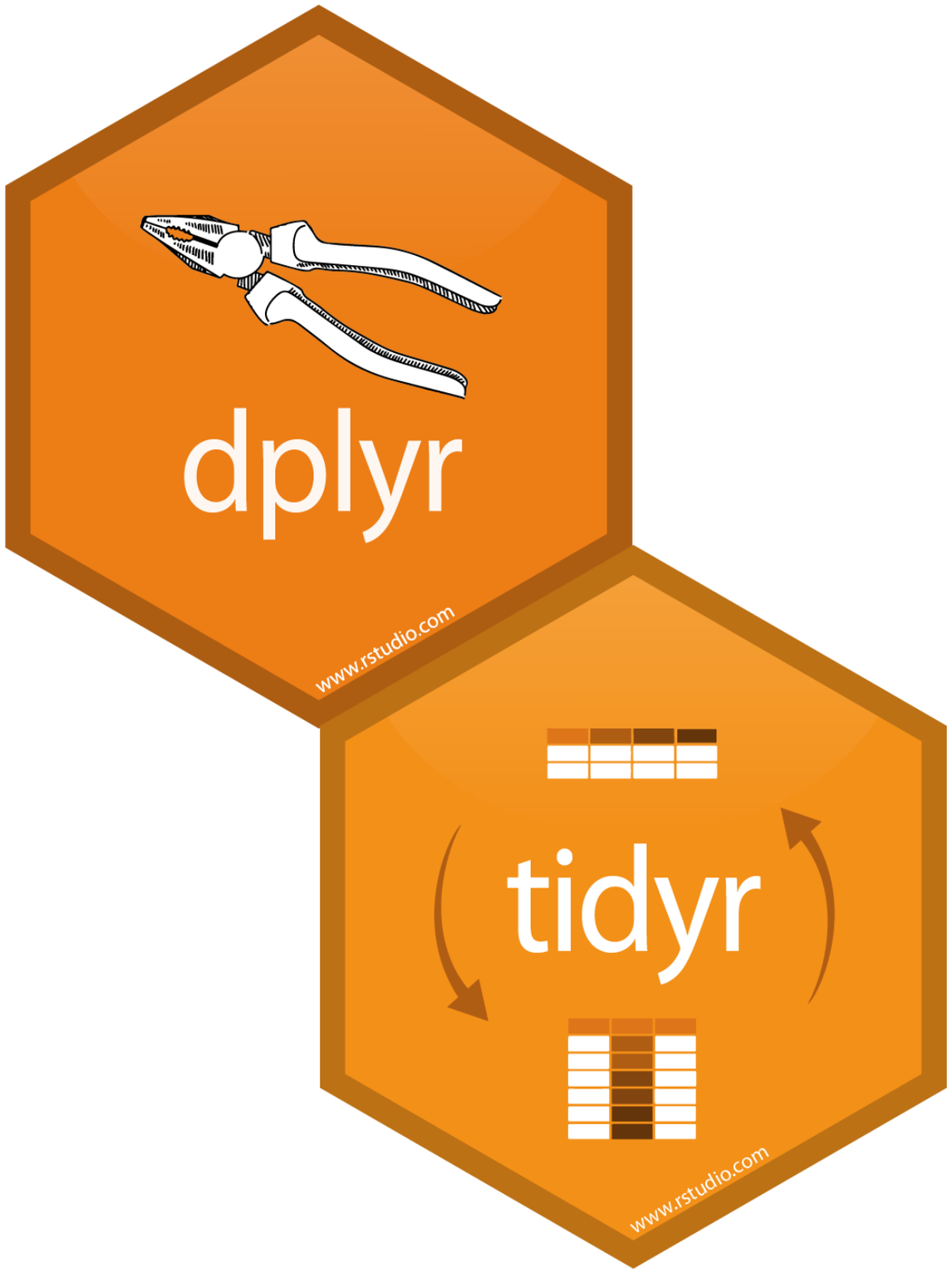
Exercício prático

Fazer os exercícios 2-I e 2-II.

Caso tenha tempo, tente fazer um exercício opcional, ou explore a leitura recomendada

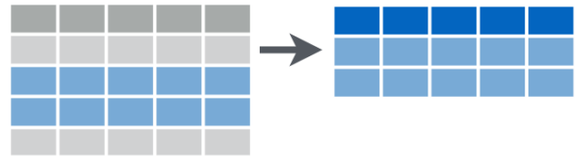
Manipulação de dados

Fundamentos do dplyr e tidyr



Subset Observations (Rows)

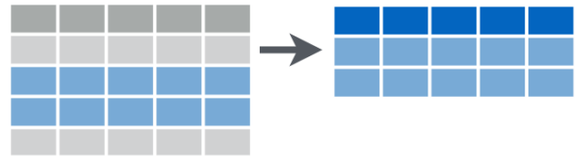
Função: filter()



```
> filter(iris, Species=="virginica")
```

Subset Observations (Rows)

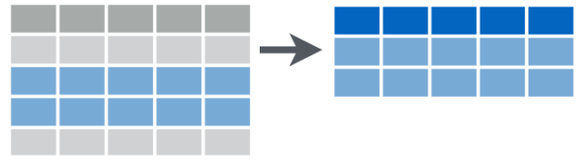
Função: filter()



```
> 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
```

Subset Observations (Rows)

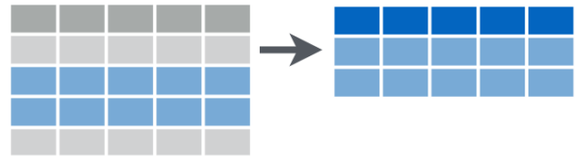
Função: filter()



```
> filter(iris, Species=="virginica", Sepal.Length>= 7.5)
```

Subset Observations (Rows)

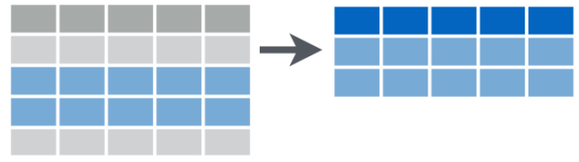
Função: filter()



```
> 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
```

Subset Observations (Rows)

Função: 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)
```

Subset Variables (Columns)

Função: select()

```
> select(iris, Sepal.Length, Sepal.Width, Species)
```

Subset Variables (Columns)

Função: 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
```

Subset Variables (Columns)

Função: select()

```
> select(iris, Sepal.Length, Sepal.Width, Species)
> select(iris, starts_with("Sepal"), Species)
```

Subset Variables (Columns)

Função: 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
```

Subset Variables (Columns)

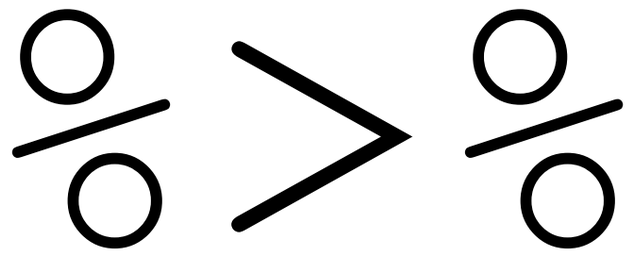
Função: select()

```
> select(iris, Sepal.Length, Sepal.Width, Species)
> select(iris, starts_with("Sepal"), Species)
> select(iris, -starts_with("Petal"))
```

Função: 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
```

Combinando funções: *piping*



```
iris %>% filter(Species=="setosa") %>%
   select(starts_with("Sepal"), Species) %>%
   head()
```

```
  Sepal.Length Sepal.Width Species
1          5.1         3.5  setosa
2          4.9         3.0  setosa
3          4.7         3.2  setosa
4          4.6         3.1  setosa
5          5.0         3.6  setosa
6          5.4         3.9  setosa
```

Make New Variables

Função: mutate()

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

Make New Variables

Função: 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
```

Make New Variables

Função: 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)
)
```

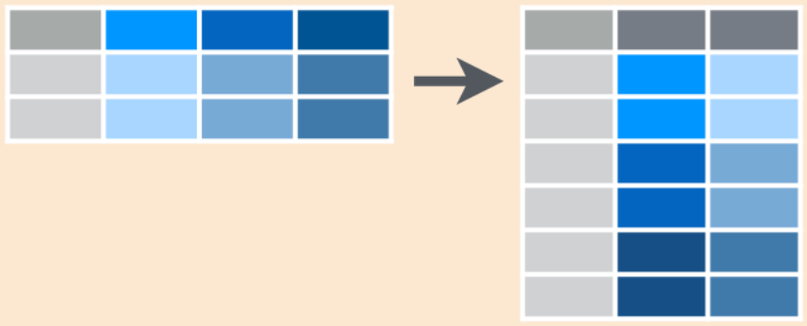
Make New Variables

Função: 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
```

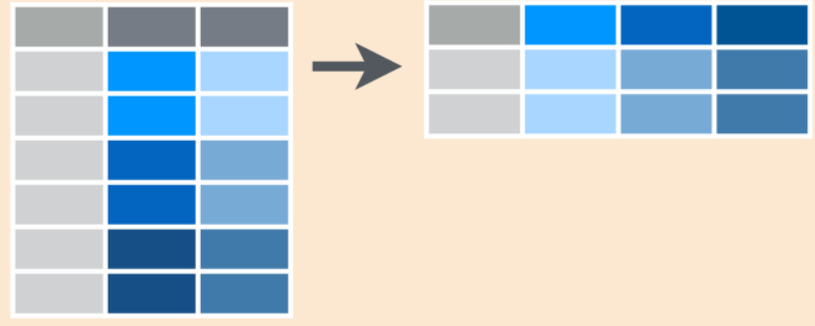
gather() e spread()



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

Gather columns into rows.

- Wide to long: **gather**



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

Spread rows into columns.

- Long to wide: **spread**

Wide and long data

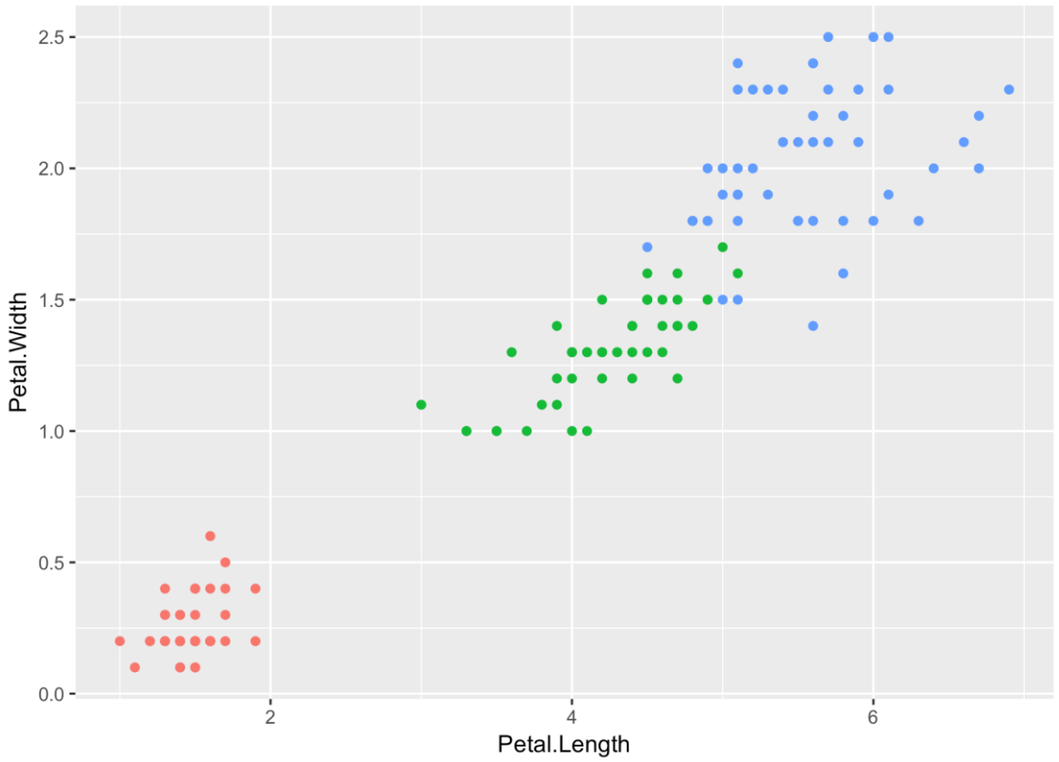
```
# A tibble: 150 x 6
   Species observation Petal.Length Petal.Width Sepal.Length Sepal.Width
   <chr>         <int>        <dbl>       <dbl>        <dbl>       <dbl>
 1 setosa            1         1.40       0.200         5.10        3.50
 2 setosa            2         1.40       0.200         4.90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

- Mais informações por observação
- Medidas de cada indivíduo
- Formato necessário no caso de plotar medidas relacionadas

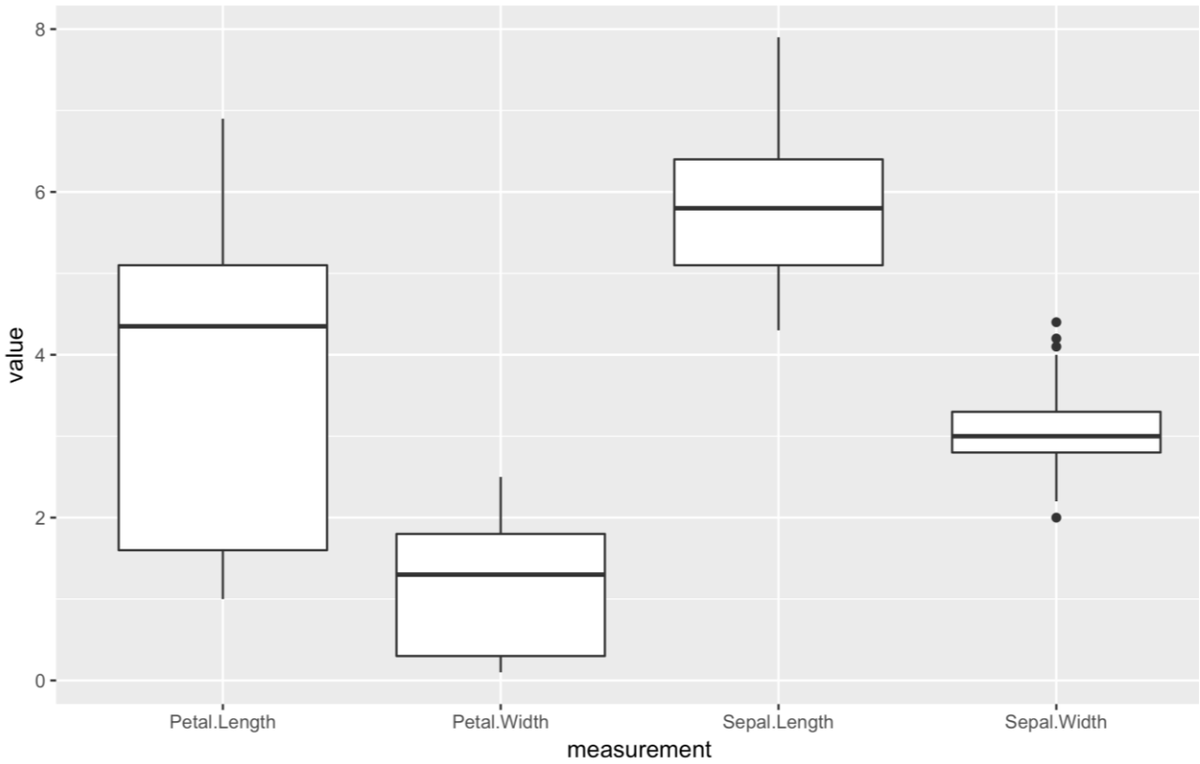
```
# A tibble: 600 x 4
   Species observation measurement  value
   <chr>         <int> <chr>        <dbl>
 1 setosa            1 Sepal.Length 5.10
 2 setosa            1 Sepal.Width  3.50
 3 setosa            1 Petal.Length 1.40
 4 setosa            1 Petal.Width  0.200
 5 setosa            2 Sepal.Length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

- Mais informacões por colunas
- Sem valores nas colunas (tidy)
- Uma única observação por linha (tidy)
- Necessário para plotar muitos dados em um único gráfico

Wide vs. Long

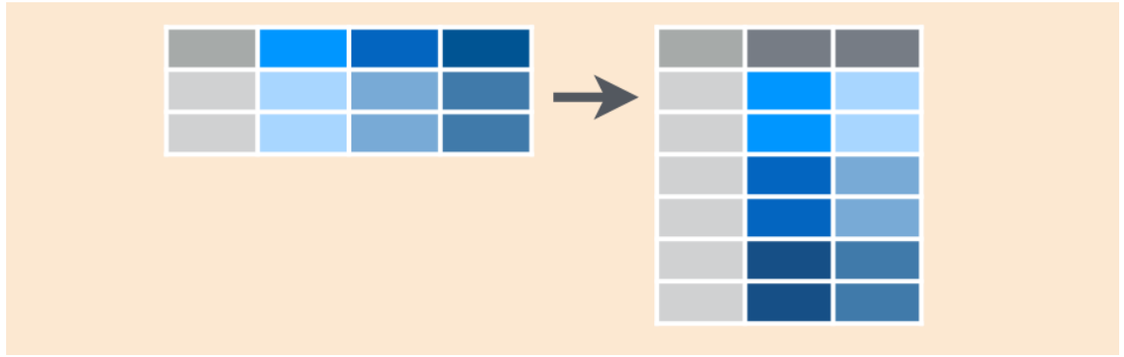


- Mais informações por observação
- Medidas de cada indivíduo
- Formato necessário no caso de plotar medidas relacionadas



- Mais informacões por colunas
- Sem valores nas colunas (tidy)
- Uma única observação por linha (tidy)
- Necessário para plotar muitos dados em um único gráfico

Função: gather()



```
> iris_obs <- mutate(iris, observation = 1:n())
> iris_obs
# A tibble: 150 x 6
```

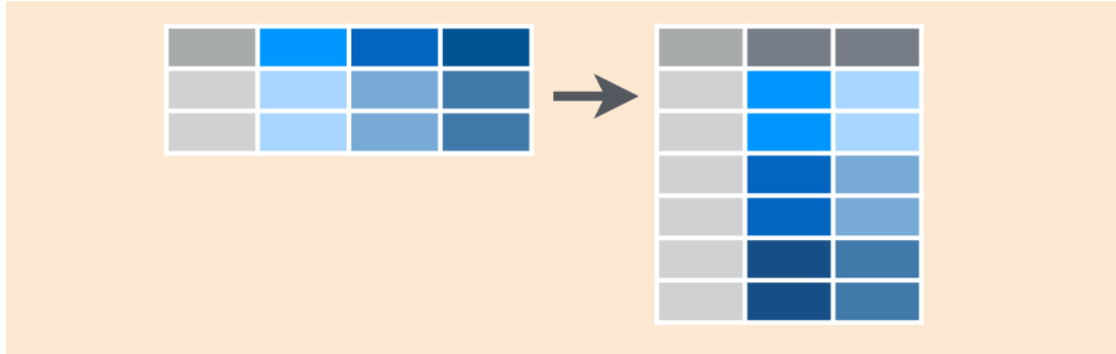
headers

	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

Função: 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
```

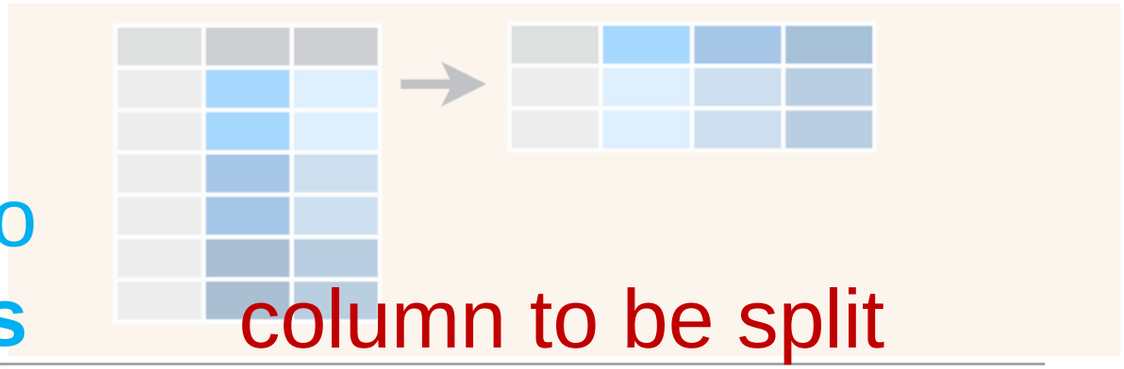
headers

content

Função: spread()

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

Função: 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>
setosa	1	1.40	0.200	5.10	3.50
setosa	2	1.40	0.200	4.90	3.00
setosa	3	1.30	0.200	4.70	3.20
setosa	4	1.50	0.200	4.60	3.10
setosa	5	1.40	0.200	5.00	3.60
setosa	6	1.70	0.400	5.40	3.90
setosa	7	1.40	0.300	4.60	3.40
setosa	8	1.50	0.200	5.00	3.40
setosa	9	1.40	0.200	4.40	2.90
setosa	10	1.50	0.100	4.90	3.10

```
# ... with 140 more rows
```

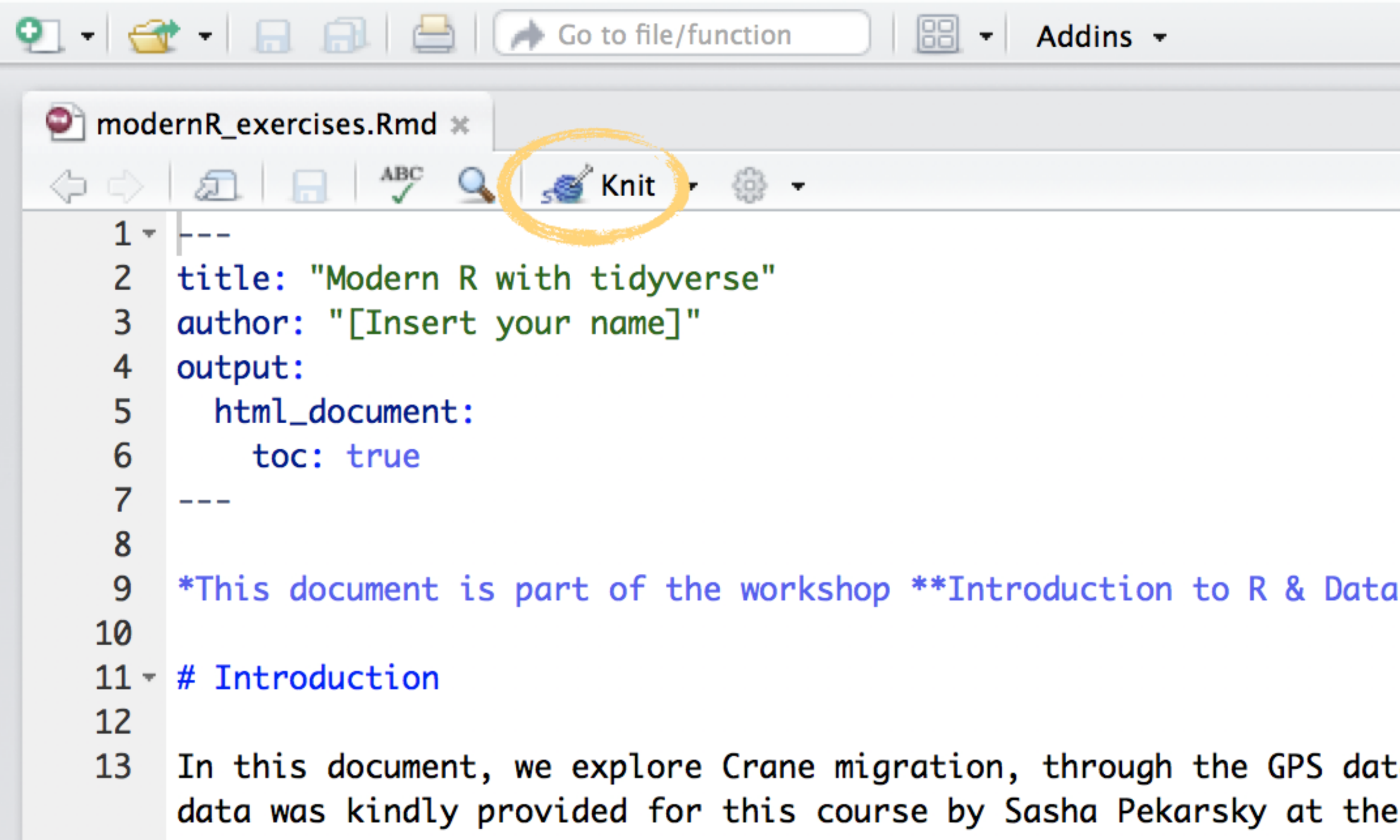
necessary for grouping!

Exercício prático

Fazer os exercícios 3-I e 3-II.

Caso tenha tempo, tente fazer um exercício opcional, ou explore a leitura recomendada

Quando finalizar, use o botão “Knit” novamente



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