Crash Course de R

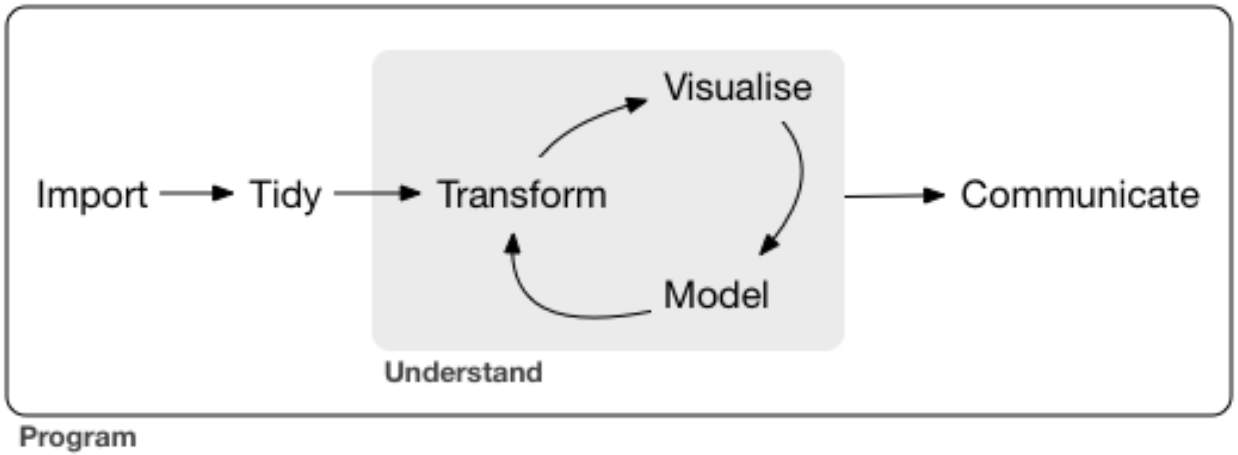
Introdução ao R e comandos básicos

Prof. Carlos Trucíos
ctruciosm.github.io
carlos.trucios@facc.ufrj.br

Faculdade de Administração e Ciências Contábeis,
Universidade Federal de Rio de Janeiro

Introdução

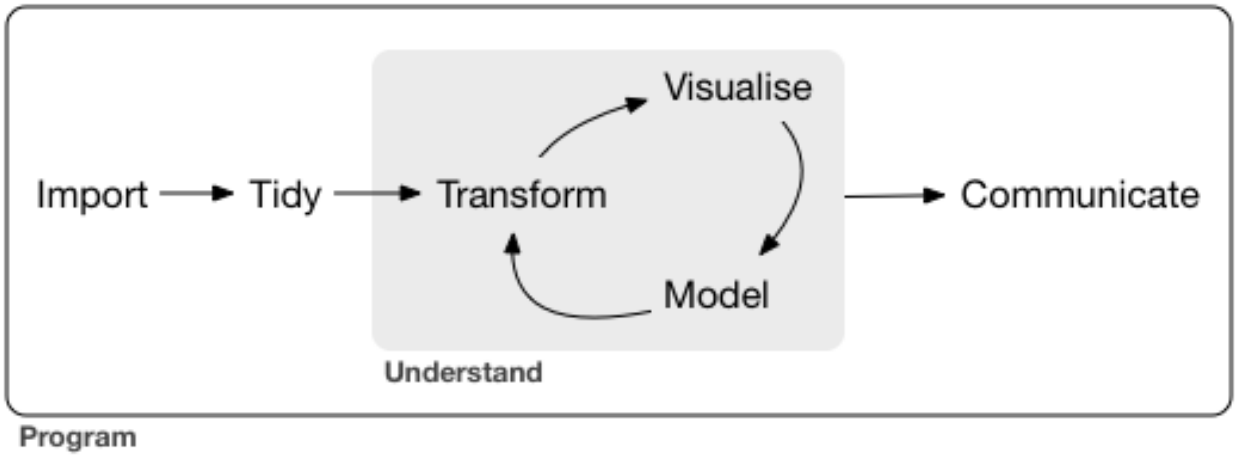
Introdução



Fonte: LIvro R for Data Science

- **Importar** Trazer os dados contidos num arquivo, banco de dados, API, etc, para o R
- **Organizar (Tidy)** Armazenar os dados de uma forma que seja mais facil de se trabalhar. Cada coluna é uma variável e cada linha é uma observação
- **Transformar** selecionar o grupo de interesse, crear novas varáveis que são função das já existentes, agregar dados, etc

Introdução



Fonte: LIvro R for Data Science

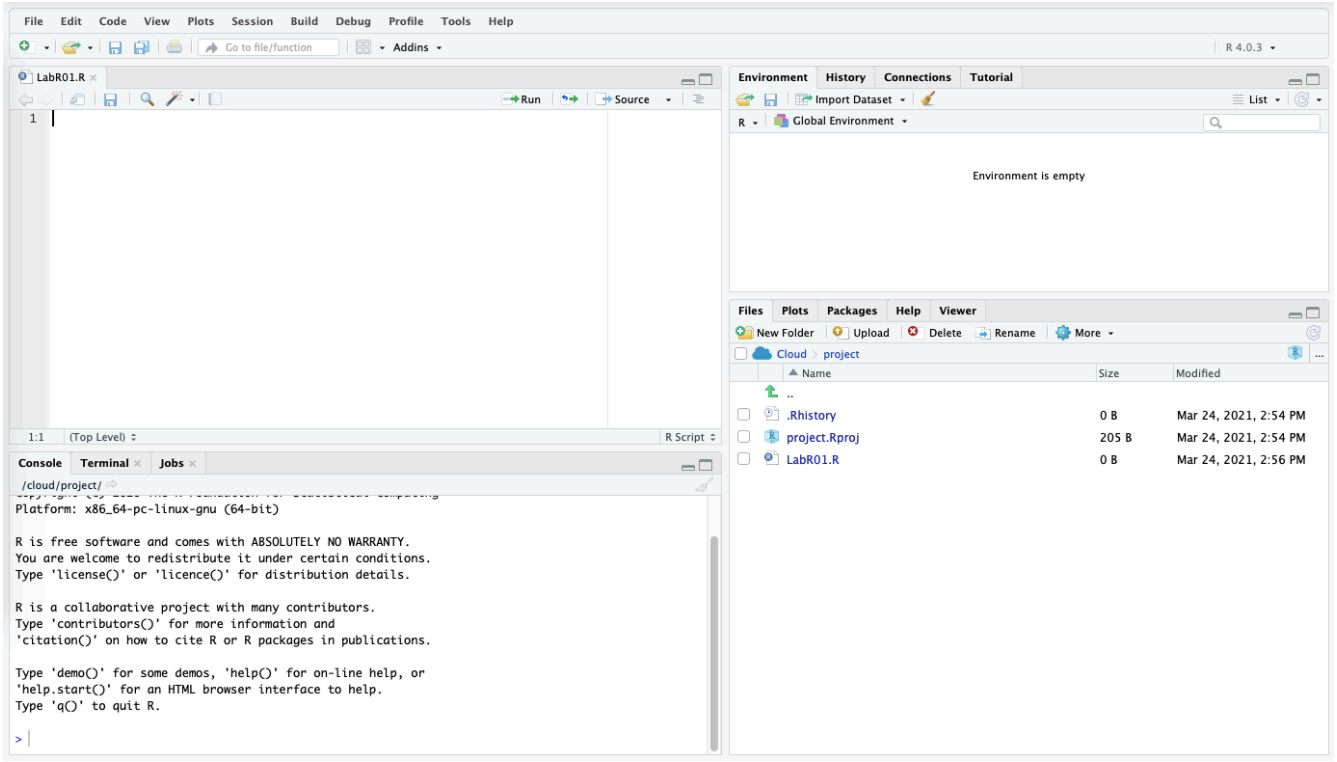
- **Visualisar** Fazer gráficos que nos permitam obter *insights* dos dados. Eventualmente podem trazer novas hipóteses ou mesmo mostrar que precisamos de mais/diferentes dados
- **Modelar** Utilizar um modelo matemático/estatístico/econométrico para responder às perguntas de interesse.
- **Comunicar** A parte final de todo projeto, onde os resultados e *insights* são apresentados.

Introdução

R is not just a programming language, but it is also an interactive environment for doing data science. Handley W.

- *R* é um *software* gratuito, facil de usar e possui uma comunidade grande desenvolvendo pacotes novos todo dia.
- Sempre podemos utilizar o R *nativo*, mas uma IDE poderosa e muito útil é o R studio.
- Uma alternativa é utilizar o R studio Cloud, porém a versão gratuita tem certas limitações em tempo de processamento e capacidade de armazenamento.

Introdução



Rstudio Cloud

Instalando pacotes

Instalando pacotes

- Quando trabalhamos com *R* precisaremos constantemente instalar pacotes e carregalos no *R*.
- Instalar pacotes é uma tarefa simples e pode ser feita com apenas uma linha de código:

```
install.packages("nome_do_pacote")
```

Uma vez o pacote esteja instalado, precisamos carregá-lo no *R*. Isto é feito com a seguinte linha de código:

```
library(nome_do_pacote)
```

Exemplo

```
install.packages("tidyverse")
library(tidyverse)
```

Instalando pacotes

Hands-on:

1. Instalar o pacote `tidyverse`
2. Carregar o pacote `tidyverse`
3. Instalar os pacotes `nycflights13`, `gapminder`, `Lahman`
4. Carregar os pacotes `nycflights13`, `gapminder`, `Lahman`

Gabarito:

```
pacotes = c("nycflights13", "gapminder", "Lahman")
install.packages(pacotes)
library(nycflights13)
library(gapminder)
library(Lahman)
```

Comandos básicos

Comandos básicos

Rodar código no R não é fácil e intuitivo.

- Por exemplo podemos utilizá-lo como calculadora

```
2+4
3-1
4*5
20/2
2^3
floor(2.99)
sin(pi)
cos(pi)
tan(tan)
sin(pi/2)^2 + cos(pi/2)^2
```

Comandos básicos

- Também podemos fazer cálculos mais interessantes

```
peso <-  c(66, 75.3, 82.1, 51.5, 72, 71.7, 68.2, 70)
sum(peso)       # soma
```

```
## [1] 556.8
```

```
mean(peso)      # média
```

```
## [1] 69.6
```

```
median(peso)    # mediana
```

```
## [1] 70.85
```

```
sd(peso)        # desvio padrão
```

```
## [1] 8.796103
```

Comandos básicos

Hands-on:

1. Escreva os seguintes comandos:
   - `10 %% 2`,
   - `10 %% 3`,
   - `11 %% 3`
   - Descubriu o que o operador `%%` faz?
2. O que o operado `%/%` faz?
3. O seguinte vetor contém as notas da P_1 de alunos de uma determinada matéria na UFRJ

```
notas <- c(9.4, 8.5, 8.4, 5.2, 7.5, 7.4, 7.9, 7.4, 7.5, 7.5, 5.5, 9.1, 7, 7.4, 6, 9, 9.4)
```

Calcule: `min()`, `max()`, `sum()`, `length()`, `sum()/length()`, `mean()`, `median()`, `sd()`, `var()`

```
c(min(notas), max(notas), sum(notas), length(notas), sum(notas)/length(notas), mean(notas), median(notas), sd(nc
```

```
## [1]   5.200000   9.400000 130.100000  17.000000   7.652941   7.652941   7.500000
## [8]   1.260515   1.588897
```

Vetores e matrizes no R

- **vetores:** um vetor é conjunto de objetos do mesmo tipo, vetores no *R* são definidos utilizando `c()`.

```
x <- c(1, 2, 3, 4, 5)
x
```

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

vetores podem ser de vários tipos:

- `logical`
- `character`
- `integer`
- `double`

Vetores e matrizes no R

```
# Lógicos
x_logical <- c(TRUE, FALSE, FALSE, TRUE)
x_logical
```

```
## [1]  TRUE FALSE FALSE  TRUE
```

```
typeof(x_logical)
```

```
## [1] "logical"
```

```
# Caracter (texto)
x_character <- c("Elisa", "Lucas", "Brenda", "Caio")
x_character
```

```
## [1] "Elisa"  "Lucas"  "Brenda" "Caio"
```

```
typeof(x_character)
```

```
## [1] "character"
```

Vetores e matrizes no R

```
# numéricos (inteiros)
x_int <- c(1, 2, 3, 4, 5) # teste também com c(1L, 2L, 3L, 4L, 5L)
x_int
```

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

```
typeof(x_int)
```

```
## [1] "double"
```

```
# numéricos (reais)
x_double <- c(1.5, 2.42, 3.76423, 0.434, 5.2)
x_double
```

```
## [1] 1.50000 2.42000 3.76423 0.43400 5.20000
```

```
typeof(x_double)
```

```
## [1] "double"
```

Vetores e matrizes no R

```
x_numeric <- c(1, 0.444, 3, 4.5, pi)
typeof(x_numeric)
```

```
## [1] "double"
```

```
x_mix <- c(TRUE, pi, 3, "Carlos")
typeof(x_mix)
```

```
## [1] "character"
```

```
x_mix
```

```
## [1] "TRUE"             "3.14159265358979" "3"                "Carlos"
```

para descobrir o número de elementos no vetor, utilizamos `length()`.

```
length(x_mix)
```

```
## [1] 4
```

Vetores e matrizes no R

- Se quisermos algum elemento do vetor, baste selecionar com `[i]` o elementos desejado.

```
x_double
```

```
## [1] 1.50000 2.42000 3.76423 0.43400 5.20000
```

```
x_double[2]
```

```
## [1] 2.42
```

```
x_double[1:4]
```

```
## [1] 1.50000 2.42000 3.76423 0.43400
```

```
x_double[c(1,3,5)]
```

```
## [1] 1.50000 3.76423 5.20000
```

Vetores e matrizes no R

- Se quisermos alterar um elemento do vetor:

```
x_double
```

```
## [1] 1.50000 2.42000 3.76423 0.43400 5.20000
```

```
x_double[2] <- 2.5
x_double
```

```
## [1] 1.50000 2.50000 3.76423 0.43400 5.20000
```

- Se quisermos alterar o tipo de vetor:

```
as.character(x_double)
```

```
## [1] "1.5"     "2.5"     "3.76423" "0.434"   "5.2"
```

outras opções são: `as.numeric()`, `as.factor()`, `as.logical()`

Vetores e matrizes no R

- **matrizes:** uma matriz é uma coleção de elementos organizados em duas dimensões, no *R* para criar matrizes utilizamos `matrix()`.

```
A <- matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
A
```

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

```
A <- matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3, byrow = TRUE)
A
```

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

Vetores e matrizes no R

Se quisermos saber as dimensões da matriz, utilizamos `dim()` e se quisermos especificamente o número de linhas ou de colunas, utilizamos `nrow()` e `ncol()`.

```
dim(A)
```

```
## [1] 2 3
```

```
ncol(A)
```

```
## [1] 3
```

```
nrow(A)
```

```
## [1] 2
```

Vetores e matrizes no R

Se quisermos selecionar alguns elementos da matriz

```
A[1,1]
```

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

```
A[1:2,c(1,3)]
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    4    6
```

Se quisermos alterar alguns elementos da matriz:

```
A[1,2] <- 9
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    9    3
## [2,]    4    5    6
```

Operações com matrizes

```
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    9    3
## [2,]    4    5    6
```

```
# Transposta
t(A)
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    9    5
## [3,]    3    6
```

```
# Inversa
solve(A[1:2,1:2])
```

```
##            [,1]        [,2]
## [1,] -0.1612903  0.29032258
## [2,]  0.1290323 -0.03225806
```

Operações com matrizes

```
A[1:2,1:2] %*% solve(A[1:2,1:2])
```

```
##      [,1]         [,2]
## [1,]    1 5.551115e-17
## [2,]    0 1.000000e+00
```

```
# Multiplicação de matrizes
B <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 3)
A%*%B
```

```
##      [,1] [,2] [,3]
## [1,]   28   67   28
## [2,]   32   77   32
```

```
A%*%A # funciona?
```

Vetores e matrizes no R

Listas no R

Listas: São objetos que contém outros objetos (que podem ser de tipos diferentes). Para criar uma lista utilizamos o comando `list()`.

```
list_data <- list("Vermelho", c(1,5,7), TRUE, 51.23, A)
print(list_data)
```

```
## [[1]]
## [1] "Vermelho"
##
## [[2]]
## [1] 1 5 7
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 51.23
##
## [[5]]
##      [,1] [,2] [,3]
## [1,]    1    9    3
## [2,]    4    5    6
```

Outras estruturas no R

> Existem outras estruturas no *R* como arrays (`array()`), fatores (`factor()`), séries temporais (`ts()`), data frames (`data.frame()`). Iremos conhecendo elas a medida que formos precisando.

Hands-on:

A função `rnorm()` gera números aleatórios de uma distribuição $N(0, 1)$.

1. Crie um vetor `x_normal` formado por 4 elementos de uma distribuição normal padrão.
2. Crie uma matrix `M` com 3 colunas, em que os elementos são os elementos contidos em `x_normal`
3. Calcule `M * M` e `M %*% M`, o que acontece? qual é diferença entre essas operações?.

Vetores e matrizes no R

Gabarito:

```
x_normal <- rnorm(4)
M <- matrix(x_normal, ncol = 2)
M
```

```
##             [,1]       [,2]
## [1,] -0.3359412 -0.3839961
## [2,]  2.4111777 -0.1801516
```

```
M*M
```

```
##           [,1]       [,2]
## [1,] 0.1128565 0.14745299
## [2,] 5.8137777 0.03245461
```

```
M%*%M
```

```
##             [,1]       [,2]
## [1,] -0.8130263  0.1981776
## [2,] -1.2443914 -0.8934282
```

Comentários finais

- A gente consegue conhecer como funciona qualquer função utilizando o comando

```
help(nome_da_funcao)
```

Para praticar em casa

1. Crie um vetor chamado `x_uniforme` que contenha 100 números aleatorios de uma disribuição uniforme $U_{[5,10]}$ (dica: veja o `help(runif)`).
2. O que acontece se fizermos `log(x_uniforme)`?
3. Crie um vetor chamado `x_chi` que contenha 100 números aleatorios de uma disribuição chi-quadrado com 1 grau de liberdade χ^2_1 (dica: veja o `help(rchisq)`).
4. Some `x_uniforme` com `x_chi`, chame o vetor resultante como `x_soma`
5. Aplique a função `floor()` no vetor x_soma
6. Um número é par se ele é divisivel por 2, quantos números pares temos em `x_soma`? (dica: o operador `%%` e a função `sum()` podem ajudar)
7. Para que serve a função `set.seed()`?

Algumas funções úteis

As seguintes Tabelas foram extraidos do livro *R for Beginners* do *Emmanuel Paradis*

`sum(x)`	sum of the elements of `x`
`prod(x)`	product of the elements of `x`
`max(x)`	maximum of the elements of `x`
`min(x)`	minimum of the elements of `x`
`which.max(x)`	returns the index of the greatest element of `x`
`which.min(x)`	returns the index of the smallest element of `x`
`range(x)`	id. than `c(min(x), max(x))`
`length(x)`	number of elements in `x`
`mean(x)`	mean of the elements of `x`
`median(x)`	median of the elements of `x`
`var(x)` or `cov(x)`	variance of the elements of `x` (calculated on $n - 1$); if `x` is a matrix or a data frame, the variance-covariance matrix is calculated
`cor(x)`	correlation matrix of `x` if it is a matrix or a data frame (1 if `x` is a vector)
`var(x, y)` or `cov(x, y)`	covariance between `x` and `y`, or between the columns of `x` and those of `y` if they are matrices or data frames
`cor(x, y)`	linear correlation between `x` and `y`, or correlation matrix if they are matrices or data frames

`round(x, n)`	rounds the elements of `x` to `n` decimals
`rev(x)`	reverses the elements of `x`
`sort(x)`	sorts the elements of `x` in increasing order; to sort in decreasing order: `rev(sort(x))`
`rank(x)`	ranks of the elements of `x`

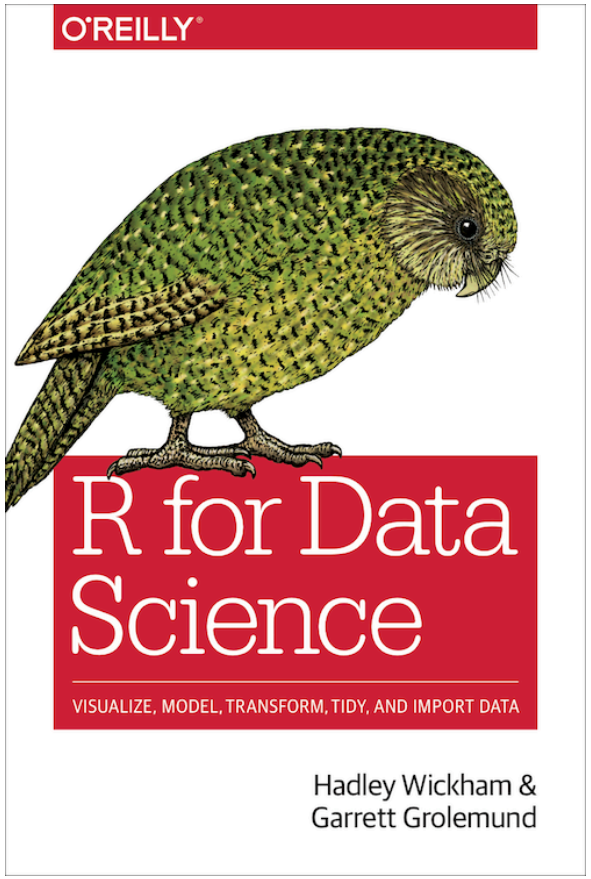
Algumas funções úteis

`log(x, base)`	computes the logarithm of `x` with base `base`
`scale(x)`	if `x` is a matrix, centers and reduces the data; to center only use the option `center=FALSE`, to reduce only `scale=FALSE` (by default `center=TRUE, scale=TRUE`)
`pmin(x,y,...)`	a vector which ith element is the minimum of `x[i]`, `y[i]`, ...
`pmax(x,y,...)`	id. for the maximum
`cumsum(x)`	a vector which ith element is the sum from `x[1]` to `x[i]`
`cumprod(x)`	id. for the product
`cummin(x)`	id. for the minimum
`cummax(x)`	id. for the maximum
`match(x, y)`	returns a vector of the same length than `x` with the elements of `x` which are in `y` (`NA` otherwise)
`which(x == a)`	returns a vector of the indices of `x` if the comparison operation is true (`TRUE`), in this example the values of `i` for which `x[i] == a` (the argument of this function must be a variable of mode logical)
`choose(n, k)`	computes the combinations of k events among n repetitions $= n!/[(n-k)!k!]$
`na.omit(x)`	suppresses the observations with missing data (`NA`) (suppresses the corresponding line if `x` is a matrix or a data frame)
`na.fail(x)`	returns an error message if `x` contains at least one `NA`
`unique(x)`	if `x` is a vector or a data frame, returns a similar object but with the duplicate elements suppressed
`table(x)`	returns a table with the numbers of the differents values of `x` (typically for integers or factors)
`table(x, y)`	contingency table of `x` and `y`
`subset(x, ...)`	returns a selection of `x` with respect to criteria (..., typically comparisons: `x$V1 < 10`); if `x` is a data frame, the option `select` gives the variables to be kept (or dropped using a minus sign)
`sample(x, size)`	resample randomly and without replacement `size` elements in the vector `x`, the option `replace = TRUE` allows to resample with replacement

Algumas funções úteis

law	function
Gaussian (normal)	`rnorm(n, mean=0, sd=1)`
exponential	`rexp(n, rate=1)`
gamma	`rgamma(n, shape, scale=1)`
Poisson	`rpois(n, lambda)`
Weibull	`rweibull(n, shape, scale=1)`
Cauchy	`rcauchy(n, location=0, scale=1)`
beta	`rbeta(n, shape1, shape2)`
'Student' (t)	`rt(n, df)`
Fisher–Snedecor (F)	`rf(n, df1, df2)`
Pearson (χ^2)	`rchisq(n, df)`
binomial	`rbinom(n, size, prob)`
multinomial	`rmultinom(n, size, prob)`
geometric	`rgeom(n, prob)`
hypergeometric	`rhyper(nn, m, n, k)`
logistic	`rlogis(n, location=0, scale=1)`
lognormal	`rlnorm(n, meanlog=0, sdlog=1)`
negative binomial	`rnbinom(n, size, prob)`
uniform	`runif(n, min=0, max=1)`
Wilcoxon's statistics	`rwilcox(nn, m, n)`, `rsignrank(nn, n)`

Quer saber mais do R?



Livro: R for Data Science

Um breve comentário do livro no meu blog