# Introdução ao R

Priscilla de Abreu Lopes

Novembro,2016

# Visão Geral

# O que é R?

- Muitos definem R como um software estatístico. No entanto, a definição dos desenvolvedores vai além:

  *R é um ambiente de programação com um conjunto integrado de ferramentas de software para manipulação de dados, simulação, cálculos e exibição de gráficos. (R Core Team 2016)*

- Baseado na linguagem S, R foi desenvolvido no final da década de 90
- R pode ser baixado e distribuído gratuitamente de acordo com a licença GNU

# O que é R?

- Interpretada:
  - Interpretador em linha de comando
  - Desenvolvimento baseado em scripts
- Tipagem dinâmica
- Orientada a Objetos
- Extensível:
  - 25 pacotes com funções básicas
  - 9500+ pacotes disponíveis no repostório oficial

## Instalar o R

Acesse o site do projeto para fazer o download da distribuição que mais lhe agrade (Linux, OS X, Windows) e execute o binário para instalação

# Interface Gráfica

- O R conta com uma interface gráfica simples, baseada em janelas, onde podem ser acessados:
  - o *R Console*
  - Um editor de scripts
  - Opções que facilitam:
    - Execução de scripts
    - Instalação de pacotes de extensão
- Existem variados editores de textos e IDEs com suporte para desenvolvimento R

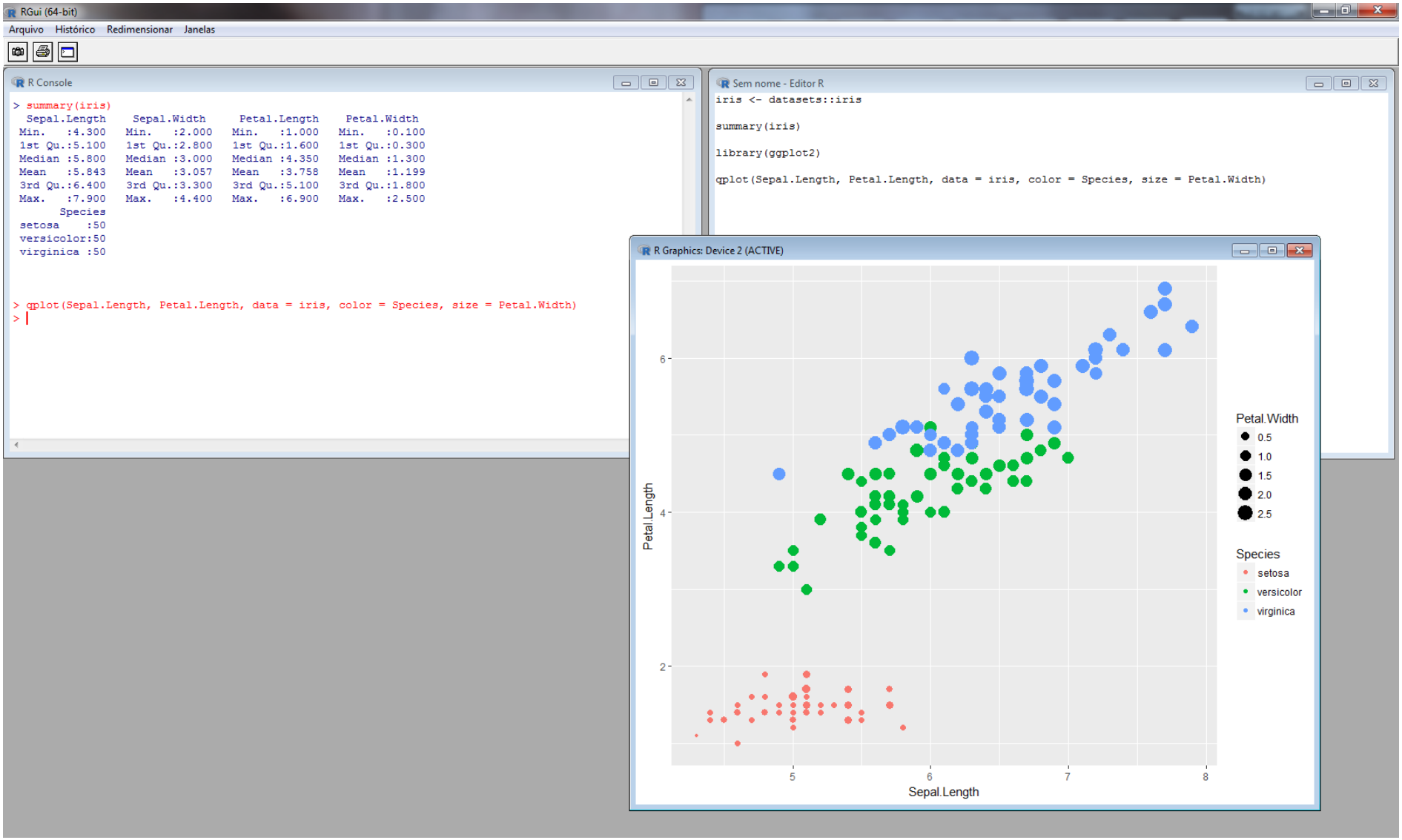



Figura 1: Screenshot da GUI do R

# O que é RStudio?

- O RStudio é uma IDE livre e de código aberto, que pode ser executada
  - Localmente (Widnows, OS X e Linux)
  - Servidor (Linux)
- O RStudio tem como objetivo facilitar o desenvolvimento em R, contando com:
  - Painéis para console, editor de scripts, gráficos, gerenciador de arquivos, entre outros
  - Debugger visual
  - Ferramentas para controle de versão (Paulson 2016)
  - Ferramentas de apoio à reprodutibilidade (RStudio Inc. 2016a)
  - Framework para aplicações web (RStudio Inc. 2016b)

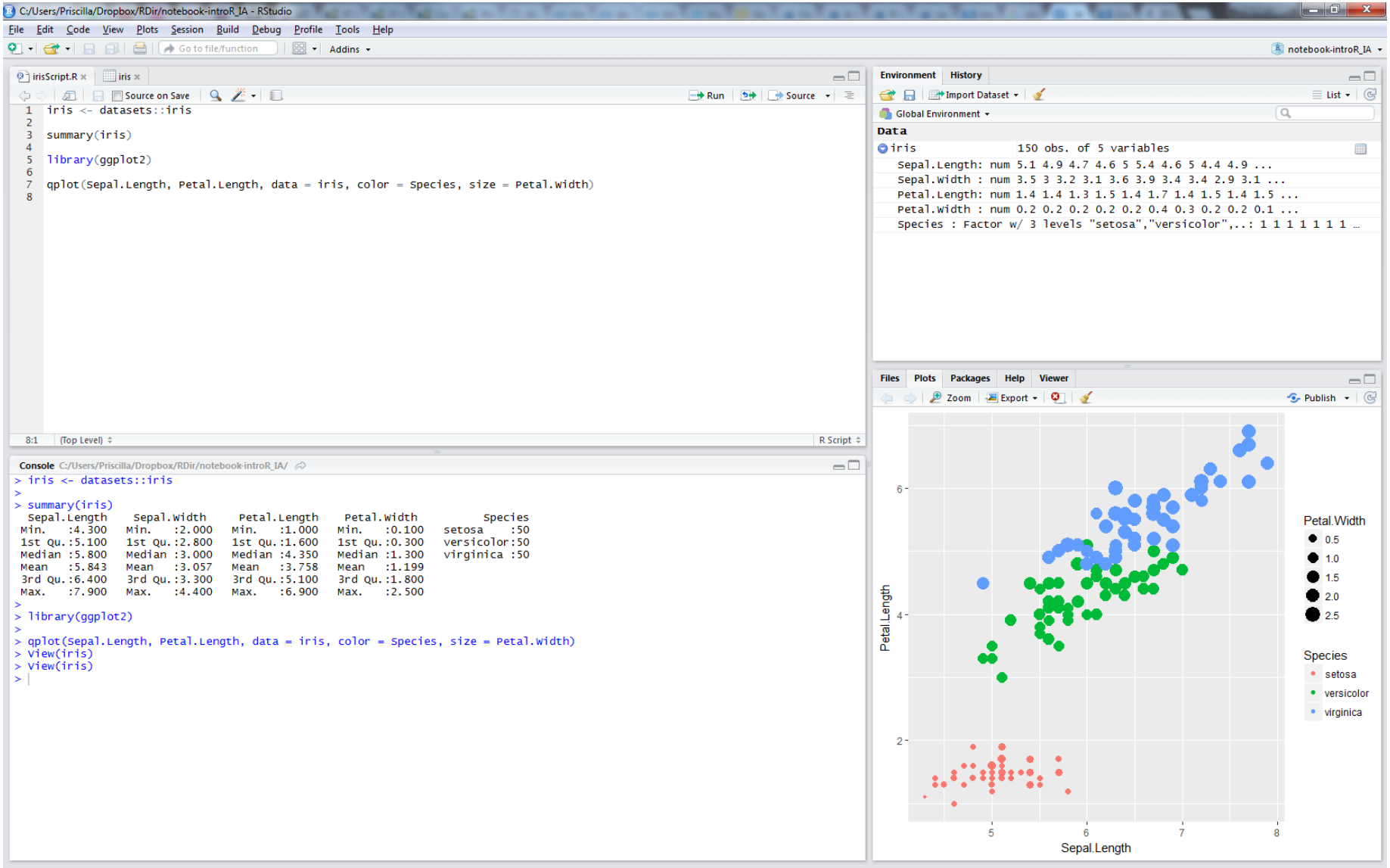



Figura 2: Screenshot do `RStudio`

## Instalar o RStudio

Acesse o site do RStudio para fazer o download da IDE (Linux, OS X, Windows) e execute o binário para instalação.

# Primeiros Comandos

# Console

- O sinal de maior (>) indica o prompt e significa que o R está pronto para receber comandos
- Se o comando inserido estiver incompleto, o sinal de adição (+) aparece, para que o comando seja completado.

```
> summary(iris     ## OMG! faltou um )
+ )                ## sem problemas!
```

# Calculadora R

- O pacote de operadores aritméticos e outras funções básicas de matemática estão disponíveis por padrão:

```
> ## Isso é um comentário
> 2        ## imprime um número
>
> ## cálculos simples
> 4 + 3
> 5 - 9
> 7 * 8
> 5 / 2    ## divisão real
> 10^3
```

# Calculadora R

```
> 5 %/% 2        ## divisão inteira
> 5 %% 2         ## resto da divisão inteira
> sqrt(9)        ## raiz quadrada
> abs(9-20)      ## valor absoluto
> factorial(4)   ## fatorial
```

# Variáveis

- Atribuição de valores a variáveis é feita pelo operador <-:

```
> x <- 2  ## Atribui
> x       ## Imprime
```

- Lembrando que a tipagem em R é dinâmica:

```
> (x <- "Hello World!")  ## Atribui e imprime
```

# Working Directory

- O *Working Directory* (WD) é o ambiente de trabalho atual

```
> getwd()                 ## retorna o WD atual
> setwd(meuDiretorio) ## altera o WD atual
>
> ls()                    ## lista de objetos no WD
```

# Working Directory

- Durante uma sessão R, uma imagem do ambiente (objetos e valores) e o histórico de comandos podem ser salvos

```
> savehistory(file = ".Rhistory") ## histórico de comandos
> save.image(file = ".RData")     ## imagem do ambiente
```

- Se não especificado um diretório, os arquivos são salvos no WD atual

# Instalando Pacotes

Uma das vantagens de `R` é a facilidade de extensão. Para instalar um novo pacote, disponível no CRAN, basta utilizar a função `install.packages()`.

```
> install.packages("ggplot2")
```

Pacotes que não fazem parte da distribuição padrão devem ser carregados antes da utilização:

```
> library(ggplot2)
```

# Tipos de Dados

# Vetores

# Vetores

- Uma lista de elementos de um **mesmo tipo**
- A função c cria vetores a partir de valores:

```
> (vInt <- c(1,2,3))                ## um vetor de inteiros
```

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

```
> (vChar <- c('a', 'b', 'c'))  ## um vetor de caracteres
```

```
## [1] "a" "b" "c"
```

# Vetores

- Para verificar o tipo de um vetor:

```
> typeof(vInt)
```

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

```
> typeof(vChar)
```

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

# Vetores

- A função c também pode ser usada para concatenar vetores:

```
> v <- c(vInt, vChar)
```

- Qual o tipo de v?

```
> typeof(v)
```

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

```
> v
```

```
## [1] "1" "2" "3" "a" "b" "c"
```

# Vetores

- Os elementos de um vetor são numerados a partir do valor **1** e podem ser acessados por seus índices:

```
> vInt[3]       ## vInt = c(1, 2, 3)
```

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

```
> vChar[2]      ## vChar = c('a', 'b', 'c')
```

```
## [1] "b"
```

# Vetores

- Acessando partes de vetores:

```
> v[3:5]      ## Elementos do índice 3 ao índice 5
```

```
## [1] "3" "a" "b"
```

```
> v[-2]       ## Todos com exceção do elemento de índice 2
```

```
## [1] "1" "3" "a" "b" "c"
```

# Vetores

- Operações aritméticas também podem ser calculadas para vetores numéricos

```
> ## vInt = c(1, 2, 3)
> vInt + c(3, 2, 1)
```

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

```
> vInt + c(5, 4)        ## vetores de tamanhos diferentes
```

```
## Warning in vInt + c(5, 4): comprimento do objeto maior não
## comprimento do objeto menor
```

```
## [1] 6 6 8
```

# Vetores

- Operações aritméticas também podem ser calculadas para vetores numéricos

```
> ## vInt = c(1, 2, 3)
> vInt * c(3, 2, 1)
```

```
## [1] 3 4 3
```

```
> vInt * 3
```

```
## [1] 3 6 9
```

# Vetores

- Mais formas de criar vetores

```
> (v11 <- logical(10))
> (v12 <- numeric(20))
> (v13 <- character(5))
> (v2 <- 1:10)
> (v3 <- seq(1, 10, by = 3))
> (v4 <- seq(1, 5, length.out = 8))
> (v5 <- rep(.5, times = 10))
```

# Vetores

- Funções padrão aplicadas a vetores

```
> length(v11)
```

```
## [1] 10
```

```
> sum(v3)
```

```
## [1] 22
```

```
> var(v4)
```

```
## [1] 1.959184
```

# Matrizes

# Matrizes

- Vetores bidimensionais (elementos de um **mesmo tipo**)
- Sejam os vetores

```
> ID <- 1:20
> Altura <- sample(150:190, size = 20, replace = TRUE)
> Peso <- sample(50:100, size = 20, replace = TRUE)
> IMC <- Peso/((Altura/100)^2)
```

# Matrizes

- Uma matriz pode ser criada pela combinação de vetores

```
> M <- cbind(ID, Altura, Peso, IMC)
> typeof(M)
```

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

```
> class(M)
```

```
## [1] "matrix"
```

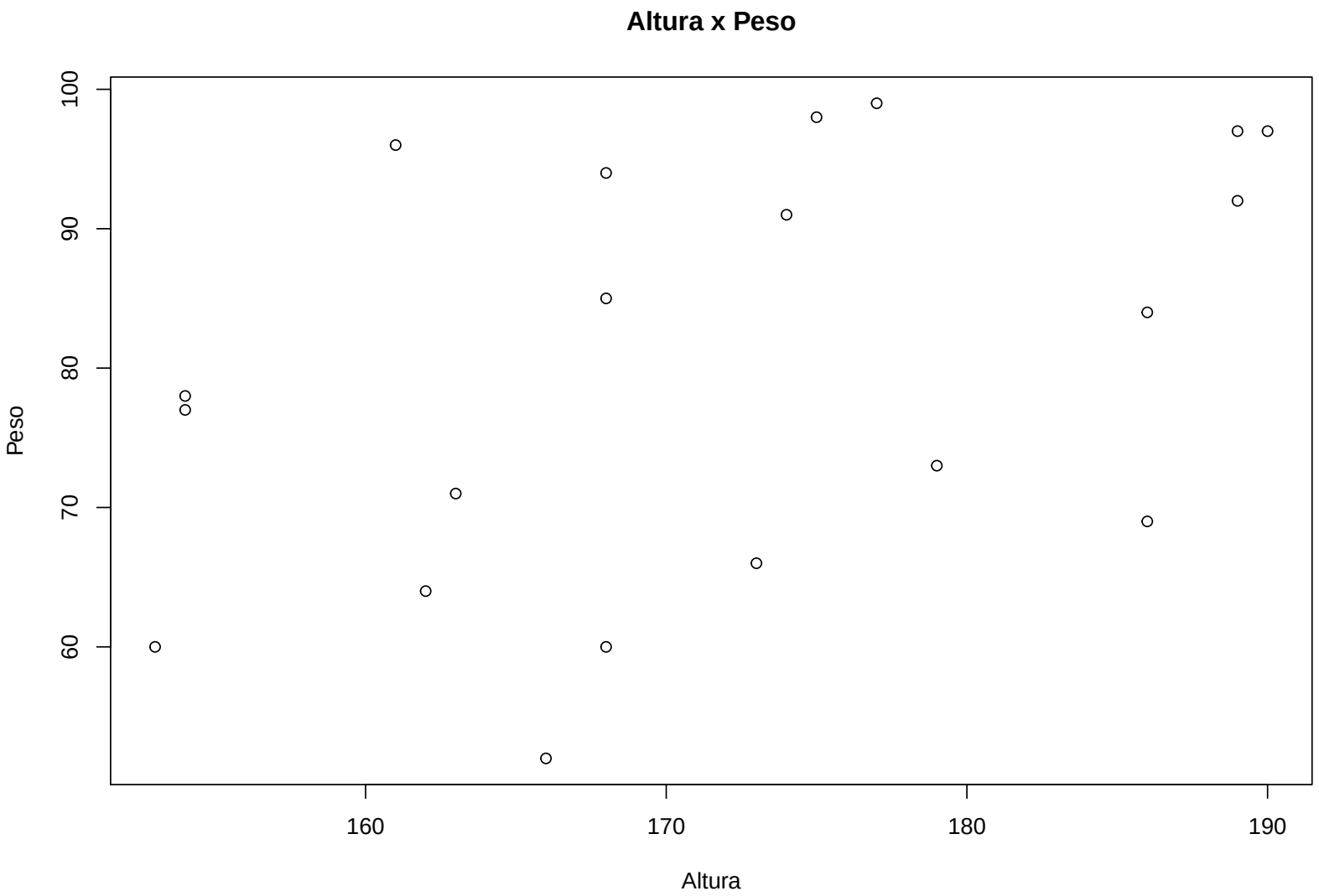
```
> dim(M)
```

```
## [1] 20  4
```

# Matrizes

- Uma matriz pode ser exibida graficamente usando a função padrão `plot`

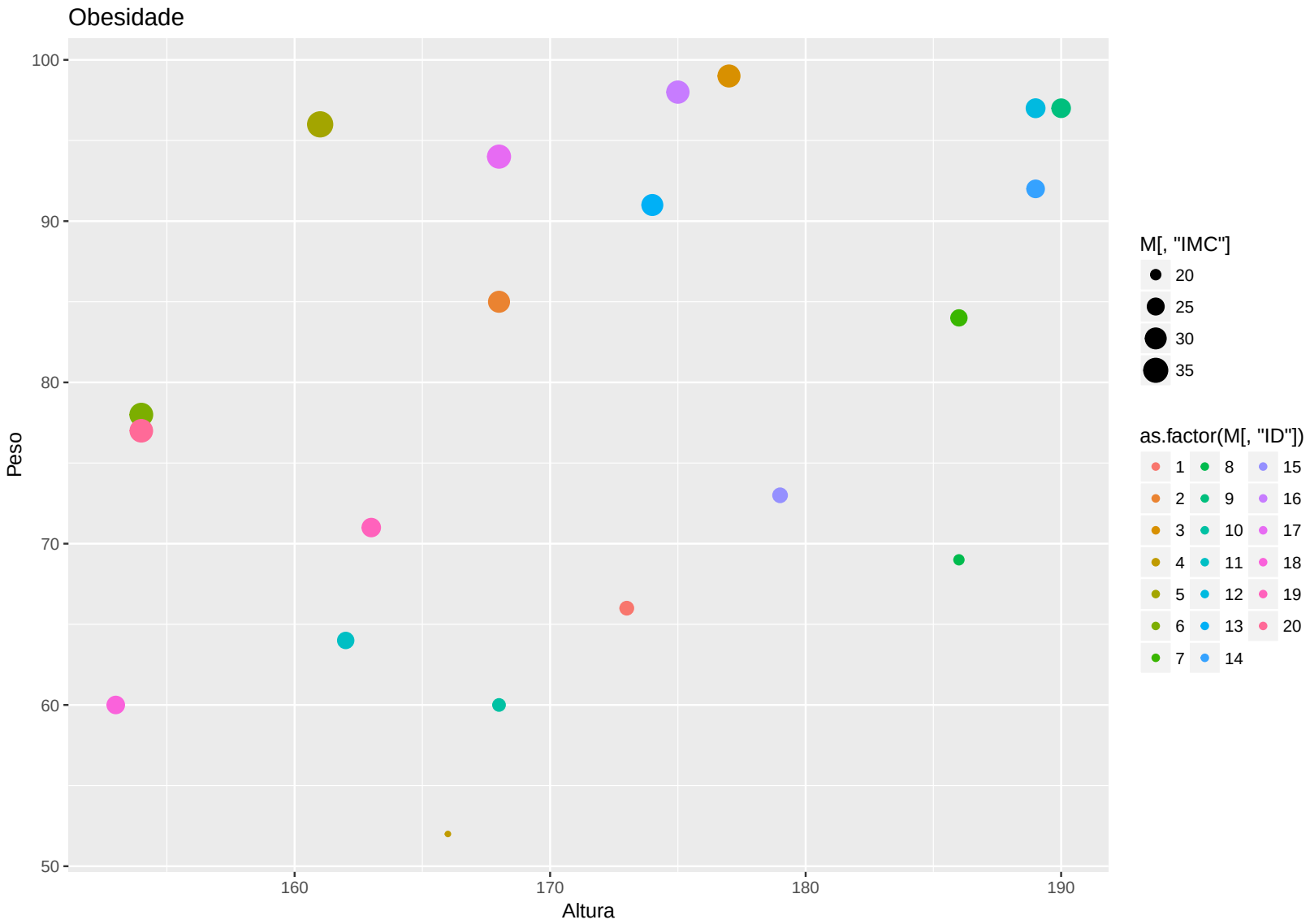
```
> plot(M[,"Altura"], M[,"Peso"],
+      ylab="Peso", xlab="Altura",
+      main="Altura x Peso")
```


Altura x Peso
Peso
100
90
80
70
60
160
170
180
190
Altura

# Matrizes

- Ou usando a função quickplot do pacote ggplot2

```
> install.packages("ggplot2")
>
> ggplot2::qplot(M[,"Altura"], M[,"Peso"],
+                ylab="Peso", xlab="Altura",
+                main="Obesidade",
+                size = M[,"IMC"],
+                color = as.factor(M[,"ID"])) +
+ ggplot2::guides(col = guide_legend(ncol = 3))
```


Obesidade
Peso
Altura
100
90
80
70
60
50
160
170
180
190
M[, "IMC"]
20
25
30
35
as.factor(M[, "ID"])
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

# Listas

# Listas

- Uma coleção de objetos R que podem ser de tipos e tamanhos **diferentes**
- Sejam os objetos armazenados nas variáveis x, y e z:

```
> x <- c(1:4)
> y <- FALSE
> z <- matrix(c(10, 20, 30, 40),nrow=2,ncol=2)
```

# Listas

- É possível criar uma lista usando a função `list()`

```
> l1 <- list(x, y, z)
> l2 <- list(x = x, y = y, z = z)
> l3 <- list()
```

# Listas

- Listas podem ser referenciadas por índices ou nomes

```
> l1[[1]]
```

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

```
> l2[[2]]
```

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

```
> l2$z
```

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

# Listas

- Elementos podem ser inseridos a listas usando índice ou nomes

```
> l3
```

```
## list()
```

```
> l3[[1]] <- "Elemento 1 da l3 não é mais vazia"
> l3$Subscribe <- TRUE
> l3
```

```
## [[1]]
## [1] "Elemento 1 da l3 não é mais vazia"
##
## $Subscribe
## [1] TRUE
```

# Listas

- Atenção!!

```
> l1[[1]]      ## valor do elemento 1 de l1
```

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

```
> l3$Subscribe ## valor do elemento 'Subscribe' de l3
```

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

# Listas

- Atenção!!

```
> l3[2]          ## retorna lista com o elemento 2
```

```
## $Subscribe
## [1] TRUE
```

```
> l2[2:3]        ## retorna lista com os elementos 2 a 3
```

```
## $y
## [1] FALSE
##
## $z
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

# data.frame

# data.frame

- Uma lista de variáveis de **mesmo tamanho**
- Sejam os vetores `X1` e `X2`

```
> X1 <- 1:5
> X2 <- rep(c(T, F), length.out = 5)
```

# data.frame

- Criação a partir de vetores e matrizes

```
> (df <- data.frame(X1, X2))
```

```
##   X1    X2
## 1  1  TRUE
## 2  2 FALSE
## 3  3  TRUE
## 4  4 FALSE
## 5  5  TRUE
```

```
> dfM <- data.frame(M)
```

# data.frame

- Acessando colunas

```
> df$X2
```

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

```
> dfM[,2:3] ## ou c('Altura', 'Peso')
```

```
##    Altura Peso
## 1     173   66
## 2     168   85
## 3     177   99
## 4     166   52
## 5     161   96
## 6     154   78
## 7     186   84
## 8     186   69
```

# data.frame

- Acessando linhas

```
> df[1,]
```

```
##   X1   X2
## 1  1 TRUE
```

```
> dfM[5:10,-1]
```

```
##    Altura Peso      IMC
## 5     161   96 37.03561
## 6     154   78 32.88919
## 7     186   84 24.28026
## 8     186   69 19.94450
## 9     190   97 26.86981
## 10    168   60 21.25850
```

# data.frame

- Função summary

```
> summary(dfM[,2:4])
```

```
##      Altura          Peso            IMC
##  Min.   :153.0   Min.   :52.00   Min.   :18.87
##  1st Qu.:162.8   1st Qu.:68.25   1st Qu.:23.91
##  Median :170.5   Median :81.00   Median :26.80
##  Mean   :171.8   Mean   :80.15   Mean   :27.26
##  3rd Qu.:180.8   3rd Qu.:94.50   3rd Qu.:31.70
##  Max.   :190.0   Max.   :99.00   Max.   :37.04
```

# Outros Tipos de Dados

# Fatores

- Vetores com níveis
- Podem representar variáveis categóricas

```
> factor(
+   c("sim","nao","sim","talvez","talvez",
+     "nao","talvez","nao","nao")
+ )
```

```
## [1] sim    nao    sim    talvez talvez nao    talvez nao
## Levels: nao sim talvez
```

# Arrays

- Objeto *n*-dimensional com elementos de um **mesmo tipo**

```
> array(1:8,dim=c(2,2,2))
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2
##
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
```

# Estruturas de Controle

# Condicional

- Considera uma condição unidimensional que pode ser `TRUE` ou `FALSE`, `T` ou `F`, `1` ou `0`, ou o valor de avaliação de alguma expressão lógica

```
> if (TRUE){
+   print("Verdadeiro")
+ } else {
+   print("Falso")
+ }
```

# Laços

- Três tipos de laços explícitos: `for`, `repeat` e `while`

```
> for(k in 1:5){
+   print(k)
+ }
```

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

- Informações sobre condicionais e laços em `R` podem ser obtidas pelo comando `?Control`

# Família `apply`

- Funções que implementam laços implícitos
  - `apply()`: aplica uma função aos elementos de uma matriz, `data.frame` ou array.
  - `lapply()`: aplica uma função a cada coluna de um `data.frame` e retorna uma lista
  - `sapply()`: aplica uma função a cada coluna de um `data.frame` e retorna uma estrutura simplificada (vetor ou matriz)
  - `tapply()`: aplica uma função considerando cada nível de um fator
  - `...`

# Família apply

- Considerando o `data.frame` `dfM` construído a partir da matriz `M`, com colunas `ID`, `Altura`, `Peso` e `IMC`

```
> ## aplica função para cada linha (MARGIN = 1)
> apply(dfM, MARGIN = 1, mean)
```

```
##  [1] 65.51305 71.27905 77.65003 60.21767 74.75890 67.72230
##  [8] 70.73613 80.71745 64.81463 65.34663 81.28872 77.01420
## [15] 72.44583 80.25000 78.07625 64.15779 69.93072 70.86688
```

```
> ## aplica função para cada coluna (MARGIN = 2)
> apply(dfM, MARGIN = 2, mean)
```

```
##         ID    Altura      Peso       IMC
##   10.50000 171.75000  80.15000  27.25902
```

# Família apply

```
> ## retorna uma lista
> lapply(dfM, mean)
```

```
## $ID
## [1] 10.5
##
## $Altura
## [1] 171.75
##
## $Peso
## [1] 80.15
##
## $IMC
## [1] 27.25902
```

# Família apply

```
> ## retorna um vetor nomeado
> sapply(dfM, mean)
```

```
##         ID     Altura       Peso        IMC
##   10.50000 171.75000   80.15000   27.25902
```

# Família apply

- Incluímos uma coluna a mais com a informação sobre a idade

```
> dfM <- cbind(dfM,
+                Idade = sample(
+                  c("Jovem", "Meia-Idade", "Terceira Idade"),
+                  size = 5, replace = TRUE))
```

```
> ## vetor com a média de Altura para cada Idade
> tapply(dfM$Altura, dfM$Idade, mean)
```

```
##       Jovem Meia-Idade
##    171.6667   171.8750
```

# Programando em R

# Scripts

- Uma série de comandos salvos em um arquivo .R

```
> ## carrega e executa um arquivo .R para a workspace
> source("meuScript.R")
```

**meuScript.R**

```
x <- 5
print(x^2)
```

# Funções

- Criação de objetos do tipo `function`
  - Função anônima
  - Função atribuída a uma variável

```
> v2
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
> ## aplicação de uma função anônima
> sapply(v2, function(x) x^2)
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

# Funções

- Função atribuída a uma variável

```
> quadrado <- function(x) {
+   print(x^2)
+ }
>
> quadrado(2)
```

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

# Funções

```
> ## aplicação da função quadrado
> sapply(v2, quadrado)
```

```
## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
## [1] 36
## [1] 49
## [1] 64
## [1] 81
## [1] 100
```

```
##   [1]   1   4   9  16  25  36  49  64  81 100
```

# Funções

- Podemos alterar a função para que retorne o quadrado do parâmetro

```
> quadrado <- function(x) {
+   x^2  ## ou return(x^2)
+ }
>
> quadrado(2)
```

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

```
> sapply(v2, quadrado)
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

# Funções

- Definindo uma função com valor padrão para parâmetro

```
> potencia <- function(x, y = 2) {
+   x^y  ## ou return(x^y)
+ }
>
> c(potencia(3),
+   potencia(3,4),
+   potencia(y = 3, x = 4))
```

```
## [1]  9 81 64
```

# Classes

- Classes podem ser definidas como funções que retornam objetos
- São 3 os sistemas para OO:
  - S3: o primeiro e mais simples
  - S4: similar a S3, mas com mais formalismo
  - RC: Classes de Referência (Reference Classes), é o sistema mais parecido com a ideia de objetos de outras linguages OO, como Python, Ruby, Java, ...
- Para grande parte dos casos, o sistema S3 é suficiente

# Sistema S3

- Seja o objeto `meuVetor`

```
> (meuVetor <- c(1, 2, 3))
```

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

```
> class(meuVetor)
```

```
## [1] "numeric"
```

# Sistema S3

- Atualizando a classe de `meuVetor`

```
> class(meuVetor) <- append(class(meuVetor), "Sequencia")
> class(meuVetor)
```

```
## [1] "numeric"   "Sequencia"
```

# Sistema S3

- Implementandos métodos por funções genéricas

```
> quinzinho <- list(nome = "Joaquim",
+                   apelido = "Quinzinho",
+                   idade = 10)
>
> class(quinzinho) <- append(class(quinzinho),"Personagem")
>
> quinzinho
```

```
## $nome
## [1] "Joaquim"
##
## $apelido
## [1] "Quinzinho"
##
## $idade
```

# Sistema S3

- Implementando métodos por funções genéricas

```
## Uma nova função genérica
maioridade <- function(obj) {
  UseMethod("maioridade",obj)
}

## Uma implementação específica para a classe Personagem
maioridade.Personagem <- function(obj) {
  return(obj$idade >= 18)
}
```

# Sistema S3

- Implementando métodos por funções genéricas já definidas pelo R

```
## Implementação de print para a classe Personagem
print.Personagem <- function(obj) {
  cat("Nome:", obj$nome, "\t")
  cat("Idade:", obj$idade, "\n")
  cat("Apelido:", obj$apelido, "\n")
}
```

# Sistema S3

- Executando funções

```
> maioridade(quinzinho)
```

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

```
> quinzinho
```

```
## Nome: Joaquim    Idade: 10
## Apelido: Quinzinho
```

# Classe S3 - ProgramaTV

```
> ProgramaTV <- function(titulo, genero = "Reality", notas = N
+
+   this <- list(
+     titulo = titulo,
+     genero = genero,
+     notas = notas
+   )
+
+   ## Define nome da classe
+   class(this) <- append(class(this), "ProgramaTV")
+
+   return(this)
+ }
```

# Classe S3 - Instância de ProgramaTV

```
> got <- ProgramaTV(
+             "Game of Thrones",
+             c("Aventura", "Drama", "Épico", "Fantasia"),
+             c(IMDB = 9.5, Rotten = 94))
```

# Classe S3 - Instância de ProgramaTV

```
> got
```

```
## $titulo
## [1] "Game of Thrones"
##
## $genero
## [1] "Aventura" "Drama"    "Épico"    "Fantasia"
##
## $notas
##   IMDB Rotten
##    9.5   94.0
##
## attr(,"class")
## [1] "list"       "ProgramaTV"
```

# Classe S3 - Métodos para ProgramaTV

```
> ## Novo método genérico
> getTitulo <- function(objeto) {
+   UseMethod("getTitulo", objeto)
+ }
>
> ## Implementação de getTitulo para a classe ProgramaTV
> getTitulo.ProgramaTV <- function(objeto) {
+   return(objeto$titulo)
+ }
```

# Classe S3 - Métodos para ProgramaTV

```
> ## Implementação de print para a classe ProgramaTV
> print.ProgramaTV <- function(objeto){
+   cat("Título:", objeto$titulo, "\n")
+   cat("Gênero:", objeto$genero, "\n")
+   cat("Notas:\n")
+   notas <- objeto$notas
+     for(i in 1:length(notas)){
+       cat("\t",names(notas)[i],":", notas[i],"\n")
+     }
+ }
```

# Classe S3 - Métodos para ProgramaTV

```
> getTitulo(got)
```

```
## [1] "Game of Thrones"
```

```
> got
```

```
## Título: Game of Thrones
## Gênero: Aventura Drama Épico Fantasia
## Notas:
##   IMDB : 9.5
##   Rotten : 94
```

# HELP!!11!!!!1!!

# Acessando Páginas de Ajuda

Para ter mais informações sobre uma determinada função ou pacote:

```
> help(ggplot2) ## ajuda sobre o pacote ggplot2
> ?Control      ## ajuda sobre estruturas de controle
```

Para pesquisar um termo nos arquivos de ajuda:

```
> help.search("plot") ## procura pelo termo "plot"
> ??solve             ## procura pelo termo "solve"
```

# Prática

# Cursos Online *free*

- swirl: http://swirlstats.com/
- DataCamp: https://www.datacamp.com/#r-courses
- Code School: https://www.codeschool.com/courses/try-r
- Dataquest: https://www.dataquest.io/course/r-programming-beginner
- Coursera: https://www.coursera.org/learn/r-programming
- edX: https://www.edx.org/course/introduction-r-data-science-microsoft-dat204x-2#!
- Udacity: https://br.udacity.com/course/data-analysis-with-r--ud651/
- Udemy: https://www.udemy.com/r-basics/

# Pacotes Interessantes

# Gerais

- `ggplot2`: visualização elegante de dados
- `dplyr`: manipulação de `data.frame`
- `lubridate`: para trabalhar com date/time
- `shiny`: aplicações web
- `rmarkdown`: Notebooks `R`, relatórios com código `R` (PDF e Word), apresentações
- `parallel`: processamento paralelo
- `testthat`: escrita e avaliação de testes unitários
- `foreach`: outra maneira de escrever laços `for`
- `Rcpp`: integração `R` e `C++`
- `plotKML`: Visualização de dados espaço-temporais integrado ao Google Earth

# Mineração de dados:

- `caret`: classificação e regressão
- `e1071`: ferramentas para validação e algoritmos clássicos de Aprendizado de Máquina
- `rpart`, `tree`: árvores de classificação e regressão
- `igraph`: análise e manipulação de grafos
- `rWeka`: interface R e WEKA
- `nnet`: redes neurais
- `arules`: extração de regras de associação
- `stream`: framework para Aprendizado em Fluxo de Dados
- `twitteR`: ferramentas para mineração de texto do twitter
- Mais pacotes? Clique aqui (Zhao 2016)

# Material de Apoio

Paulson, Josh. 2016. "Version Control with Git and Svn." https://support.rstudio.com/hc/en-us/articles/200532077-Version-Control-with-Git-and-SVN.

R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

RStudio Inc. 2016a. "R Notebooks." http://rmarkdown.rstudio.com/r_notebooks.html.

———. 2016b. "Shiny." http://shiny.rstudio.com/.

Zhao, Yanchang. 2016. "RDataMining.com: R and Data Mining." http://www.rdatamining.com/.