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

Características

- 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

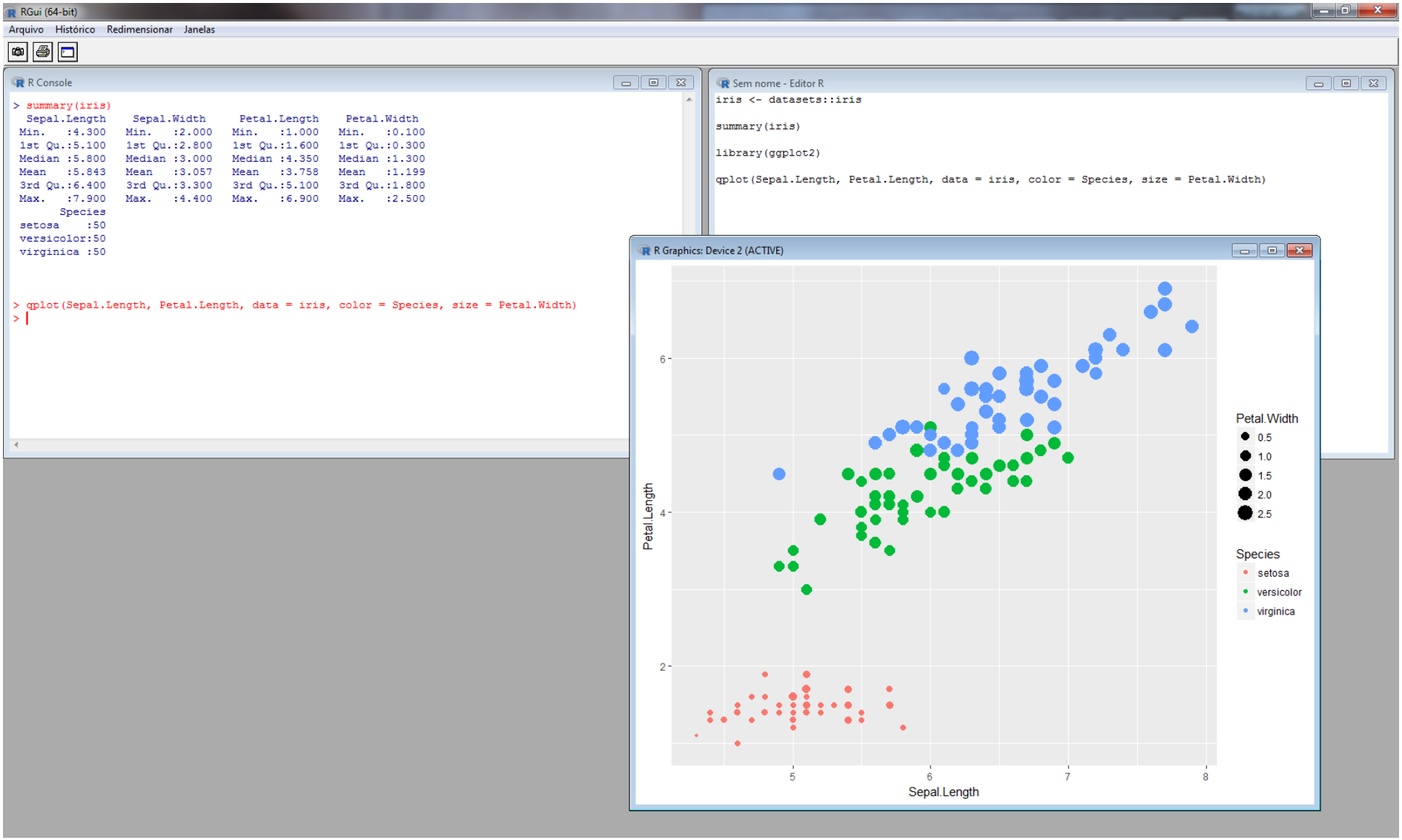


Figura 1: Screenshot da GUI do R

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)

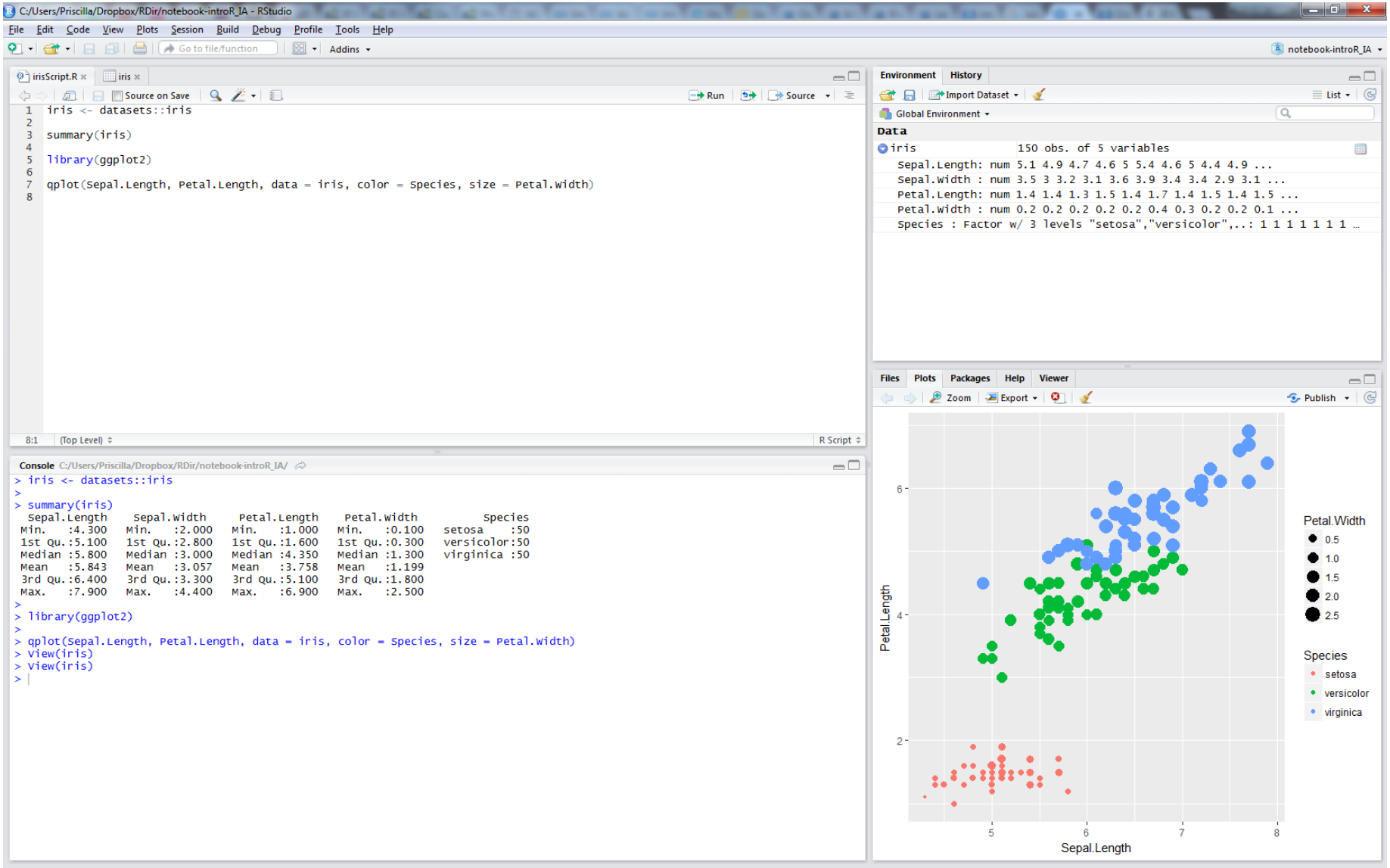


Figura 2: Screenshot do `RStudio`

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") ## salva o histórico de coma
> save.image(file = ".RData")     ## salva imagem do ambiente
```

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

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)
```

```
## 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 <- 1:5
Altura <- c(168, 177, 177, 177, 165)
Peso <- c(88, 72, 85, 52, 71)
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"
```

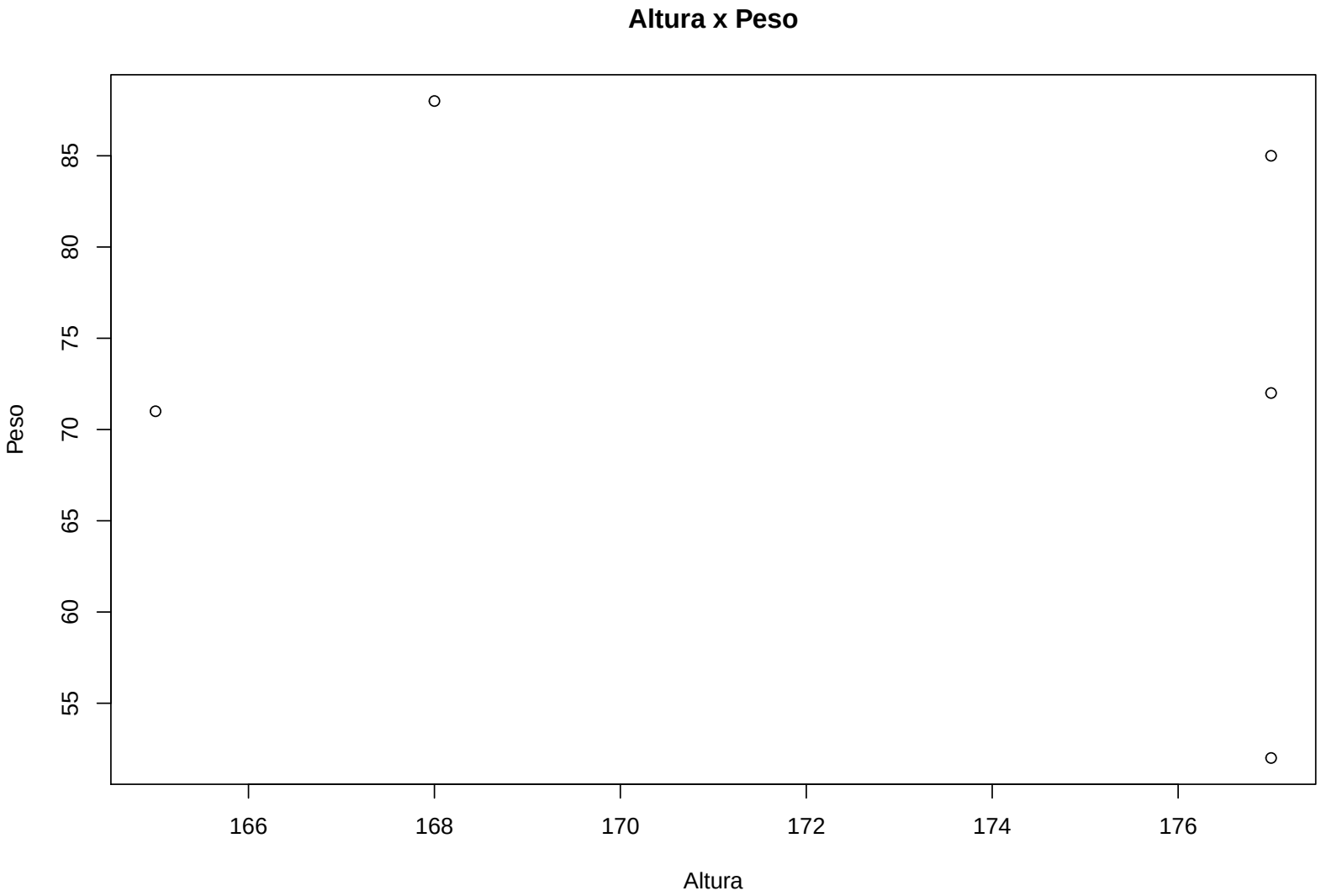
```
> dim(M)
```

```
## [1] 5 4
```

Matrizes

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

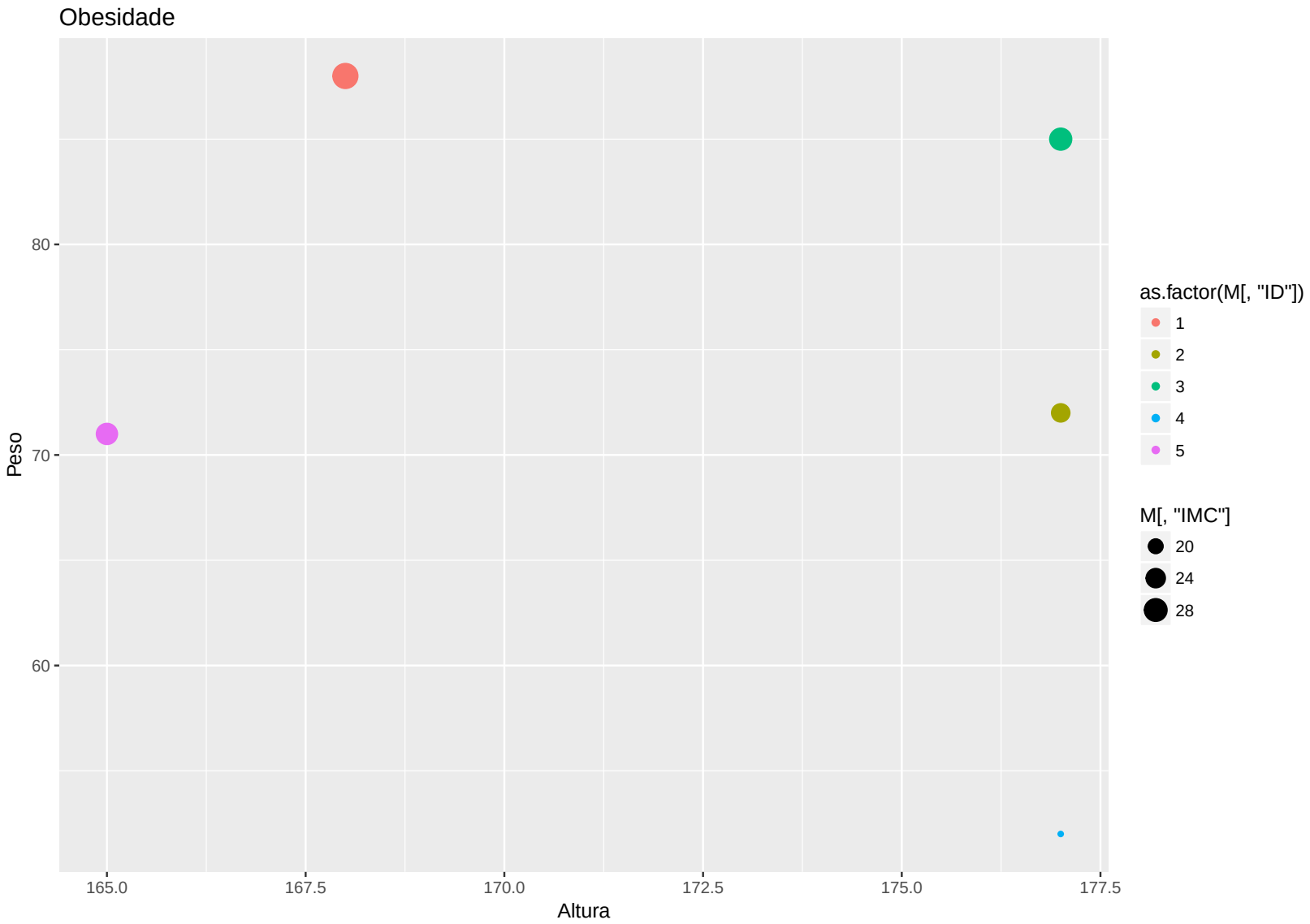
```
> plot(M[,"Altura"], M[,"Peso"],
+      ylab="Peso", xlab="Altura",
+      main="Altura x Peso")
```


Altura x Peso
Peso
55
60
65
70
75
80
85
166
168
170
172
174
176
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"]))
```


Obesidade
Peso
80
70
60
165.0
167.5
170.0
172.5
175.0
177.5
Altura
as.factor(M[, "ID"])
1
2
3
4
5
M[, "IMC"]
20
24
28

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!!

```
> l3[[2]]      ## retorna valor do elemento 2
```

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

```
> l3$Subscribe ## retorna valor do elemento 2
```

```
## [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**

```
> (df <- data.frame(v1, v2))
```

```
##   v1    v2
## 1  1  TRUE
## 2  2 FALSE
## 3  3  TRUE
## 4  4 FALSE
## 5  5  TRUE
```

Outros Tipos de Dados

- **data.frame**: uma lista de variáveis de mesmo tamanho

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

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

```
> df$v2
```

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

Outros Tipos de Dados

- **data.frame**: uma lista de variáveis de mesmo tamanho

```
> summary(df)
```

```
##        v1          v2
##  Min.   :1   Mode :logical
##  1st Qu.:2   FALSE:2
##  Median :3   TRUE :3
##  Mean   :3   NA's :0
##  3rd Qu.:4
##  Max.   :5
```

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 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 a cada nível de um fator

Família apply

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

```
> apply(df, 1, mean) ## aplica função para cada linha
```

```
## [1] 72.04478 68.49548 73.03285 62.39951 66.76974
```

```
> apply(df, 2, mean) ## aplica função para cada coluna
```

```
##         ID    Altura      Peso       IMC
##    3.00000 172.80000  73.60000  24.79389
```

Família apply

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

```
## $ID
## [1] 3
##
## $Altura
## [1] 172.8
##
## $Peso
## [1] 73.6
##
## $IMC
## [1] 24.79389
```

Família apply

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

```
##         ID     Altura       Peso        IMC
##    3.00000 172.80000  73.60000  24.79389
```

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

```
> ## descobrir a média de Altura para cada Idade
> tapply(df$Altura, df$Idade, mean) ## retorna um vetor
```

```
##  18  20  22  26  41  45  46  47  48  49
## 177 168 177 177 168 165 165 177 177 177
```

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

```
> ## sapply com aplicação de uma função anônima
> sapply(v1, function(x) x^2)
```

```
## [1]  1  4  9 16 25
```

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

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

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(v1, quadrado)
```

```
## [1]  1  4  9 16 25
```

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 definem outras funções
- Uma classe pode ser definida da maneira que a imaginação mandar
- 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, por isso é mais comumente utilizado

Sistema S3

- Ideia geral: como alterar a classe de um objeto

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

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

```
> class(x)
```

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

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

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

Sistema S3

- Implementandos métodos por funções genéricas

```
> x <- list(nome = "Joaquim", apelido = "Quinzinho")
>
> class(x) <- append(class(x),"Personagem")
>
> x
```

```
## $nome
## [1] "Joaquim"
##
## $apelido
## [1] "Quinzinho"
##
## attr(,"class")
## [1] "list"       "Personagem"
```

Sistema S3

- Implementando métodos por funções genéricas

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

## Uma implementação de pegaNome para a classe Personagem
pegaNome.Personagem <- function(obj) {
  obj$nome
}

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

Sistema S3

- Executando funções

```
> pegaNome(x)
```

```
## [1] "Joaquim"
```

```
> x
```

```
## Nome: Joaquim    Apelido: Quinzinho
```

Classe S3 - ProgramaTV

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

Objeto got

```
> got <- ProgramaTV("Game of Thrones", c("Aventura", "Drama",
>
> got$titulo
```

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

```
> got$genero
```

```
## [1] "Aventura" "Drama"    "Épico"    "Fantasia"
```

Classe S3 - Métodos Genéricos

```
> getTitulo <- function(objeto) {
+   UseMethod("getTitulo", objeto)
+ }
>
> getTitulo.ProgramaTV <- function(objeto) {
+   return(objeto$titulo)
+ }
>
> print.ProgramaTV <- function(objeto){
+   cat("Título:", objeto$titulo, "\n")
+   cat("Gênero:", objeto$genero, "\n")
+ }
```

Objeto got

```
> getTitulo(got)
```

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

```
> got
```

```
## Título: Game of Thrones
## Gênero: Aventura Drama Épico Fantasia
```

HELP!!11!!!!1!!

Acessando Páginas de Ajuda

Para ter mais informações sobre uma determinada função ou pacote, utilizamos os comandos `help()` ou `?`.

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

Para pesquisar um termo nos arquivos de ajuda, utilizamos os comandos `help.search()` ou `??`

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

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)
```

Atividade

- Salve seu histórico de comandos, uma imagem do ambiente e, se for o caso, quaisquer os scripts gerados durante esta aula

```
> savehistory()
> save.image()
```

- Envie os arquivos .Rhistory, .RData e scripts .R (se houver) para `alopes.priscilla@gmail.com`:
  - Assunto: IA - 18/11/2016
  - Corpo: nome e RA dos que compartilharam a máquina

Prática

- 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

- `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/.