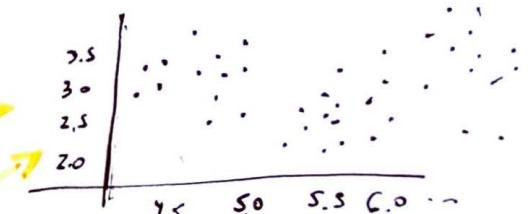


```

1 # Solução Lista de Exercícios Capítulo 2
2
3 # Obs: Caso tenha problemas com a acentuação, consulte este link:
4 # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6 # Configurando o diretório de trabalho
7 # Coloque entre aspas o diretório de trabalho que você está usando no seu
8 # computador
9 # Não use diretórios com espaço no nome
10 setwd("C:/FCD/BigDataRAzure/Cap03")
11 getwd()
12
13 # Exercício 1 - Crie um vetor com 12 números inteiros
14 vec <- c(1:12)
15 vec
16
17 # Exercício 2 - Crie uma matriz com 4 linhas e 4 colunas! preenchida com números
18 # inteiros
19 mat <- matrix(c(1:16), nrow = 4, ncol = 4)
20
21 # Exercício 3 - Crie uma lista unindo o vetor e matriz criados anteriormente
22 lst <- list(vec, mat)
23 lst
24
25 Uma lista com vetor e Matriz unidos
26
27 # Exercício 4 - Usando a função read.table() leia o arquivo do link abaixo para
28 # uma dataframe
29 # http://data.princeton.edu/wws509/datasets/effort.dat
30 df <-
31 data.frame(read.table("http://data.princeton.edu/wws509/datasets/effort.dat"))
32 class(df)
33 df
34
35 # Exercício 5 - Transforme o dataframe anterior, em um dataframe nomeado com os
36 # seguintes labels:
37 names(df) = c("config", "esfc", "chang")
38 names(df) = c("Col1", "Col2", "Col3")
39 df
40
41
42 # Exercício 6 - Imprima na tela o dataframe iris,
43 # verifique quantas dimensões existem no dataframe iris, imprima um resumo do
44 dataset
45 iris
46 class(iris)
47 dim(iris)
48 summary(iris)
49 str(iris)
50 View(iris)
51
52 # Exercício 7 - Crie um plot simples usando as duas primeiras colunas do
53 # dataframe iris
54 plot(iris$Sepal.Length, iris$Sepal.Width)
55
56 # Exercício 8 - Usando a função subset, crie um novo dataframe com o conjunto de
57 # dados do dataframe iris em que Sepal.Length > 7
58 # Dica: consulte o help para aprender como usar a função subset()
59 ?subset
60 iris1 <- subset(iris, Sepal.Length > 7)
61 View(iris1)

```

Nome das colunas



```
62
63 # Exercícios 9 (Desafio) - Crie um dataframe que seja cópia do dataframe iris e
64 # usando a função slice(),
65 # divida o dataframe em um subset de 15 linhas
66 # Dica 1: você vai ter que instalar e carregar o pacote dplyr
67 # Dica 2: Consulte o help para aprender como usar a função slice()
68 novo_iris <- iris
69 novo_iris
70 install.packages("dplyr")
71 library(dplyr)
72 ?slice
73 slice(novo_iris, 1:15)
74 class(slice(novo_iris, 1:15))
75
76 # Exercícios 10 - Use a função filter no seu novo dataframe criado no item
77 anterior
78 # e retorne apenas valores em que Sepal.Length > 6
79 # Dica: Use o RSiteSearch para aprender como usar a função filter
80 RSiteSearch('filter') → abre o Navegador Help
81 filter(novo_iris, Sepal.Length > 7)

82 → filtro informado que contenha Sepal.Length > 7
83
84
85
```

```

1  # Big Data na Prática 1 - Analisando a Temperatura Média nas Cidades Brasileiras
2
3  # Obs: Caso tenha problemas com a acentuação, consulte este link:
4  # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6  # Configurando o diretório de trabalho
7  # Coloque entre aspas o diretório de trabalho que você está usando no seu
8  # computador
9  # Não use diretórios com espaço no nome
10 setwd("C:/FCD/BigDataRAzure/Cap03")
11 getwd()
12
13 # Dataset:
14 # Berkeley Earth
15 # http://berkeleyearth.org/data
16 # TemperaturasGlobais.csv ~ 78 MB (zip) ~ 496 MB (unzip)
17 # Faça o download do arquivo zip no link abaixo e descompacte na mesma pasta onde
18 # está este script.
19 # https://drive.google.com/open?id=1nSwP3Y0V7gncbnG_DccNhrTRxmUNqMqa
20
21 # Instalando e Carregando Pacotes
22 # Obs: os pacotes precisam ser instalados apenas uma vez. Se já instalou em
23 # outros scripts, não é necessário instalar novamente!
24 install.packages("readr")
25 install.packages("data.table")
26 install.packages("dplyr")
27 install.packages("ggplot2")
28 library(readr)
29 library(dplyr)
30 library(ggplot2)
31 library(scales)
32 library(data.table)
33
34 # Carregando os dados (Usando um timer para comparar o tempo de carregamento com
35 # diferentes funções)
36 system.time(df_teste1 <- read.csv("TemperaturasGlobais/TemperaturasGlobais.csv"))
37
38 # Usando read.table()
39 system.time(df_teste2 <- read.table("TemperaturasGlobais/TemperaturasGlobais.csv"))
40
41 # Usando fread()
42 ?fread
43 system.time(df <- fread("TemperaturasGlobais/TemperaturasGlobais.csv"))
44
45
46 # Criando subsets dos dados carregados
47 cidadesBrasil <- subset(df, Country == 'Brazil')
48 cidadesBrasil <- na.omit(cidadesBrasil)
49 head(cidadesBrasil)
50 nrow(df)
51 nrow(cidadesBrasil)
52 dim(cidadesBrasil)
53
54
55 # Preparação e Organização
56
57 # Convertendo as Datas
58 cidadesBrasil$dt <- as.POSIXct(cidadesBrasil$dt, format='%Y-%m-%d')
59 cidadesBrasil$Month <- month(cidadesBrasil$dt)
60 cidadesBrasil$Year <- year(cidadesBrasil$dt)
61
62
63 # Carregando os subsets
64
65 # Palmas

```

→ carregamento muito mais rápido

↳ retira os valores missing NA

```

66 plm <- subset(cidadesBrasil, City == 'Palmas')
67 plm <- subset(plm, Year %in% c(1796,1846,1896,1946,1996,2012))
68
69 # Curitiba
70 crt <- subset(cidadesBrasil, City == 'Curitiba')
71 crt <- subset(crt, Year %in% c(1796,1846,1896,1946,1996,2012))
72
73 # Recife
74 recf <- subset(cidadesBrasil, City=='Recife')
75 recf <- subset(recf,Year %in% c(1796,1846,1896,1946,1996,2012))
76
77
78 # Construindo os Plots
79 p_plm <- ggplot(plm, aes(x = (Month), y = AverageTemperature, color =
80   as.factor(Year))) +
81   geom_smooth(se = FALSE, fill = NA, size = 2) +
82   theme_light(base_size = 20) +
83   xlab("Mês") +
84   ylab("Temperatura Média") +
85   scale_color_discrete("") +
86   ggtitle("Temperatura Média ao longo dos anos em Palmas") +
87   theme(plot.title = element_text(size = 18))
88
89 p_crt <- ggplot(crt, aes(x = (Month), y = AverageTemperature, color =
90   as.factor(Year))) +
91   geom_smooth(se = FALSE, fill = NA, size = 2) +
92   theme_light(base_size = 20) +
93   xlab("Mês") +
94   ylab("Temperatura") +
95   scale_color_discrete("") +
96   ggtitle("Temperatura Média ao longo dos anos em Curitiba") +
97   theme(plot.title = element_text(size = 18))
98
99 p_recf <- ggplot(recf, aes(x = (Month), y = AverageTemperature, color =
100  as.factor(Year))) +
101  geom_smooth(se = FALSE, fill = NA, size = 2) +
102  theme_light(base_size = 20) +
103  xlab("Mês") +
104  ylab("Temperatura Média") +
105  scale_color_discrete("") +
106  ggtitle("Temperatura Média ao longo dos anos em Recife") +
107  theme(plot.title = element_text(size = 18))
108
109 # Plotando!
110 p_plm
111 p_crt
112 p_recf
113

```

```

1 # Fatores
2
3 # Obs: Caso tenha problemas com a acentuação, consulte este link:
4 # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6 # Configurando o diretório de trabalho
7 # Coloque entre aspas o diretório de trabalho que você está usando no seu
8 computador
9 # Não use diretórios com espaço no nome
9 setwd("C:/FCD/BigDataRAzure/Cap03")
10 getwd()

11 vec1 <- c("Macho", "Femea", "Femea", "Macho", "Macho")
12 vec1
13 fac_vec1 <- factor(vec1)
14 fac_vec1
15 class(vec1)
16 class(fac_vec1)
17
18 # Variáveis categóricas nominais
19 # Não existe uma ordem implícita
20 animais <- c("Zebra", "Pantera", "Rinoceronte", "Macaco", "Tigre")
21 animais
22 animais
23 class(animais)
24 fac_animais <- factor(animais)
25 fac_animais
26 class(fac_animais)
27 levels(fac_animais)

28 # Variáveis categóricas ordinais
29 # Possuem uma ordem natural
30 grad <- c("Mestrado", "Doutorado", "Bacharelado", "Mestrado", "Mestrado")
31 grad
32
33 fac_grad <- factor(grad, order = TRUE, levels = c("Doutorado", "Mestrado",
34 "Bacharelado"))
35 fac_grad
36 levels(fac_grad)

37 # Sumarizar os dados fornece uma visão geral sobre o conteúdo das variáveis
38 summary(fac_grad) → 1 Doutorado, 3 mestrado, 1 Bacharelado
39 summary(grad)

40 vec2 <- c("M", "F", "F", "M", "M", "M", "F", "F", "M", "M", "M", "F", "F", "M",
41 "M")
42 vec2
43 fac_vec2 <- factor(vec2)
44 fac_vec2
45 levels(fac_vec2) <- c("Femea", "Macho"), Mudo o nome dos níveis
46 fac_vec2
47 summary(fac_vec2) → 6 Femias 9 Machos M F = Macho e Femeia
48 summary(vec2)
49
50
51 # Mais exemplos
52 data = c(1,2,2,3,1,2,3,3,1,2,3,3,1)
53 fdata = factor(data)
54 fdata
55
56 rdata = factor(data, labels = c("I", "II", "III"))
57 rdata
58
59 # Fatores Não-Ordenados
60 set1 <- c("AA", "B", "BA", "CC", "CA", "AA", "BA", "CC", "CC")
61 set1
62
63
64 # Transformando os dados.
65 # R apenas criou os níveis, o que não significa que exista uma hierarquia.
66 f.set1 <- factor(set1)

```

Macho e femea

Variáveis
 4
 idade | Peso | Sexo

Fator = transforma a variável
Para estimar o processamento

↳ sequência ↴ Nível

Mudo o nome dos níveis
6 Femias 9 Machos M F = Macho e Femeia

Podemos categorizar né também temos as categorias 1, 2, 3

Pod mudar o factor 1 = I

```

67 f.set1
68 class(f.set1)
69 is.ordered(f.set1) → Verifica se é ordenado
70
71 # Fatores Ordenados
72 o.set1 <- factor(set1,
73   levels = c("CA", "BA", "AA", "CC", "B"),
74   ordered = TRUE) → define os níveis
75
76 o.set1 → determina que seja ordenado
77 is.ordered(o.set1)
78
79 as.numeric(o.set1)
80 table(o.set1)
81
82
83 # Fatores e Dataframes
84 df <- read.csv2("etnias.csv", sep = ',') → Exibe data frame
85 View(df) → Recebe arquivo CSV Separado por ()
86
87
88 # Variáveis do tipo fator
89 str(df) → informações sobre o data frame
90
91 # Níveis dos fatores
92 # Internamente, o R armazena valores inteiros e faz um mapeamento para as strings
# (em ordem alfabética)
93 # e agrupa as estatísticas por níveis. Agora, se fizermos summarização de
# estatísticas, é possível visualizar
94 # a contabilização para cada categoria
95 levels(df$Etnia)
96 summary(df$Etnia)
97
98
99 # Plot
100 # Agora se fizermos um plot, temos um boxplot para estas variáveis categóricas
101 plot(df$Idade ~ df$Etnia, xlab = 'Etnia', ylab = 'Idade', main = 'Idade por Etnia')
102
103 # Regressão → Ligar model
104 summary(lm(Idade ~ Etnia, data = df))
105
106
107 # Convertendo uma coluna em variável categórica. Isso criará um fator não-ordenado
108 df
109 str(df)
110 df$Estado_Civil.cat <- factor(df$Estado_Civil, labels = c("Solteiro", "Casado",
# "Divorciado"))
111 df
112 str(df)
113
114
115
116

```

```

1 # Fatores e Dataframes - Compreendendo a Ordem dos Fatores
2
3 # Obs: Caso tenha problemas com a acentuação, consulte este link:
4 # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6 # Configurando o diretório de trabalho
7 # Coloque entre aspas o diretório de trabalho que você está usando no seu
8 # computador
9 # Não use diretórios com espaço no nome
10 setwd("C:/FCD/BigDataRAzure/Cap03")
11 getwd()
12
13 # Níveis dos fatores
14 # Internamente, o R armazena valores inteiros e faz um mapeamento para as strings
15 # (em ordem alfabética)
16 # e agrupa as estatísticas por níveis.
17
18 # Criando vetores
19 vec1 <- c(1001, 1002, 1003, 1004, 1005)
20 vec2 <- c(0, 1, 1, 0, 2)
21 vec3 <- c('Verde', 'Laranja', 'Azul', 'Laranja', 'Verde')
22
23 # Unindo os vetores em um dataframe
24 df <- data.frame(vec1, vec2, vec3)
25
26 # Verificando que o R categorizou a última coluna como fator
27 str(df)
28
29 # Verificando os níveis do fator. Perceba que os níveis estão categorizados em
30 # ordem alfabética
31 levels(df$vec3) → mostra os Níveis Azul Laranja Verde
32 # Criando uma outra coluna e atribuindo labels
33 df$cat1 <- factor(df$vec3, labels = c("cor2", "cor1", "cor3")) → determina o Label
34 # das strings
35 str(df) → coluna vec3 em df
36
37 # Internamente, os fatores são registrados como inteiros, mas a ordenação segue a
38 # ordem alfabética
39 # Sumário de informações
40 # Veja como foi feita a atribuição!
41 # Azul = cor2
42 # Laranja = cor1
43 # Verde = cor3
44 # Ou seja, os vetores com os labels, seguiram a ordem alfabética dos níveis
45 # classificados pelo R
46
47 # Criando uma outra coluna e atribuindo labels
48 # Ao aplicarmos a função factor() a coluna vec2, internamente o R classificou em
49 # ordem alfabética
50 # e quando atribuímos os labels, foi feita a associação.
51 df$cat2 <- factor(df$vec2, labels = c("Divorciado", "Casado", "Solteira"))
52 df
53 str(df)
54 levels(df$cat2)
55
56
57
58
59
60
61
62
63

```

Handwritten notes:

- # Unindo os vetores em um dataframe → criando dataframe df passando 3 vetores
- # Verificando que o R categorizou a última coluna como fator → Verificando que o R categorizou a última coluna como fator
- levels(df\$vec3) → mostra os Níveis Azul Laranja Verde → mostra os Níveis
- # Criando uma outra coluna e atribuindo labels → Criando uma outra coluna e atribuindo labels
- df\$cat1 <- factor(df\$vec3, labels = c("cor2", "cor1", "cor3")) → determina o Label → determina o Label de cada Nível
- # das strings → das strings
- str(df) → coluna vec3 em df → coluna vec3 em df
- # Internamente, os fatores são registrados como inteiros, mas a ordenação segue a ordem alfabética → Internamente, os fatores são registrados como inteiros, mas a ordenação segue a ordem alfabética
- # Sumário de informações → Sumário de informações
- # Veja como foi feita a atribuição! → Veja como foi feita a atribuição!
- # Azul = cor2 → Azul = cor2
- # Laranja = cor1 → Laranja = cor1
- # Verde = cor3 → Verde = cor3
- # Ou seja, os vetores com os labels, seguiram a ordem alfabética dos níveis classificados pelo R → Ou seja, os vetores com os labels, seguiram a ordem alfabética dos níveis classificados pelo R
- # Criando uma outra coluna e atribuindo labels → Criando uma outra coluna e atribuindo labels
- # Ao aplicarmos a função factor() a coluna vec2, internamente o R classificou em ordem alfabética → Aplicando a função factor() a coluna vec2, internamente o R classificou em ordem alfabética
- # e quando atribuímos os labels, foi feita a associação. → quando atribuímos os labels, foi feita a associação.

```

1 # Estruturas de Controle
2
3 # Obs: Caso tenha problemas com a acentuação, consulte este link:
4 # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6 # Configurando o diretório de trabalho
7 # Coloque entre aspas o diretório de trabalho que você está usando no seu
8 # computador
9 setwd("C:/FCD/BigDataRAzure/Cap03")
10 getwd()
11
12 # If-else
13 x = 25
14 if (x < 30)
15   ("Este número é menor que 30")
16
17
18 # Chaves não são obrigatórios, mas altamente recomendados
19 if (x < 30)
20   ("Este número é menor que 30") Funciona sem chaves
21
22 # Else
23 if (x < 7) {
24   "Este número é menor que 7"
25 } else {
26   "Este número não é menor que 7"
27 }
28
29
30 # Comandos podem ser aninhados
31 x = 7
32 if (x < 7) {
33   "Este número é menor que 7"
34 } else if(x == 7) {
35   "Este é o número 7"
36 } else{
37   "Este número não é menor que 7"
38 }
39
40
41 # Ifelse!
42 x = 5
43 ifelse(x < 6, "Correto!", NA) Só x < 6 entra "Correto"
44
45 x = 9
46 ifelse (x < 6, "Correto!", NA) Sendo NA
47
48
49
50 # Expressões ifelse aninhadas
51 x = c(7,5,4)
52 ifelse(x < 5, "Menor que 5",
53       ifelse(x == 5, "Igual a 5", "Maior que 5"))
54
55
56 # Estruturas if dentro de funções
57 func1 <- function(x,y){
58   ifelse(y < 7, x + y, "Não encontrado")
59 }
60
61 func1(4,2)
62 func1(40,7)
63
64
65 # Rep
66 rep(rnorm(10), 5) Cria uma repetição do rnorm(10)
67
68

```

```

69 # Repeat
70 x = 1
71 repeat {
72   x = x + 3
73   if (x > 99)
74     break
75   print(x)
76
77 # Loop For
78 for (i in 1:20) {print(i)} → 1 2 3 4 5 6 7 8 9 10 11 12 ...
79 for (q in rnorm(10)) {print(q)}
80
81
82 # Ignora alguns elementos dentro do loop
83 for(i in 1:22){
84   if(i == 13 | i == 15) → Se i for = a 13 ou 15
85   next
86   print (i)} → Pula
87
88
89
90 # Interromper o loop
91 for(i in 1:22){
92   if(i == 13)
93     break / interrompe o loop
94   print (i)}
95
96
97 # Loop While
98 x = 1
99 while(x < 5){
100   x = x + 1
101   print(x)
102 }
103
104 # O loop while não será executado
105 y = 6
106 while(y < 5){ → Não entra no loop
107   y = y+10
108   print(y)
109 }
110
111
112
113

```

Repete o loop ate satisfazer a condicao e dar o break

Loop For

for (i in 1:20) {print(i)} → 1 2 3 4 5 6 7 8 9 10 11 12 ...

for (q in rnorm(10)) {print(q)} → imprime rnorm(10)

Ignora alguns elementos dentro do loop

for(i in 1:22){

if(i == 13 | i == 15) → Se i for = a 13 ou 15

next

print (i)} → Pula

Interromper o loop

for(i in 1:22){

if(i == 13)

break / interrompe o loop

print (i)}

Loop While

x = 1

while(x < 5){

x = x + 1

print(x)

}

O loop while não será executado

y = 6

while(y < 5){ → Não entra no loop

y = y+10

print(y)

}

```

1 # Funções
2
3 # Obs: Caso tenha problemas com a acentuação, consulte este link:
4 # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6 # Configurando o diretório de trabalho
7 # Coloque entre aspas o diretório de trabalho que você está usando no seu
8 # computador
9 # Não use diretórios com espaço no nome
10 setwd("C:/FCD/BigDataRAzure/Cap03")
11 getwd()
12
13 # Help
14 ?sample
15 args(sample) → argumentos da função sample
16 args(mean)
17 args(sd)
18 → standard deviation
19 # Funções Built-in
20 abs(-43) → absoluto
21 sum(c(1:5))
22 mean(c(1:5))
23 round(c(1.1:5.8))
24 rev(c(1:5)) → reverso 1 2 3 4 5 → 5 4 3 2 1
25 seq(1:5)
26 sort(rev(c(1:5)))
27 append(c(1:5), 6) → adiciona um item
28
29 vec1 <- c(1.5, 2.5, 8.4, 3.7, 6.3)
30 vec2 <- rev(vec1)
31 vec2
32
33 # Funções dentro de funções
34 plot(rnorm(10)) → Cria um plot do resultado da função rnorm(10)
35 mean(c(abs(vec1), abs(vec2)))
36
37
38 # Criando funções
39 myfunc <- function(x) { x + x }
40 myfunc(10)
41 class(myfunc)
42
43 myfunc2 <- function(a, b) {
44   valor = a ^ b
45   print(valor)
46 }
47 myfunc2(3, 2) → Não recebe parâmetros
48
49 jogando_dados <- function() {
50   num <- sample(1:6, size = 1) # Local
51   num
52 }
53 → Variável local
54 jogando_dados() → pega uma amostra dos 6 elementos
55
56
57 # Escopo
58 print(num)
59 num <- c(1:6)
60 num # Global
61
62
63
64 # Funções sem número definido de argumentos
65 vec1 <- (10:13)
66 vec2 <- c("a", "b", "c", "d")
67 vec3 <- c(6.5, 9.2, 11.9, 5.1)
68

```

```

69 myfunc3 <- function(...){
70   df = data.frame(cbind(...))
71   print(df)
72 }
73
74 myfunc3(vec1)
75
76 myfunc3(vec2, vec3)
77
78 myfunc3(vec1, vec2, vec3)
79
80
81 # Funções Built-in - Não tente recriar a roda
82 # Comparação de eficiência entre função vetorizada e função "vetorizada no R"
83
84 x <- 1:10000000 10 milhões de registros
85
86 # Função que calcula a raiz quadrada de cada elemento de um vetor de números
87 meu_sqrt <- function(numeros) {
88   resp <- numeric(length(numeros))
89   for(i in seq_along(numeros)) {
90     resp[i] <- sqrt(numeros[i])
91   }
92   return(resp)
93 }
94
95 system.time(x2a <- meu_sqrt(x)) pego o tempo
96
97 system.time(x2b <- sqrt(x)) recebe o resultado
98
99 # Sua máquina pode apresentar resultado diferente por conta da precisão de
100 # cálculo do processador.
101 identical(x2a, x2b)
102
103
104
105

```

pego o tempo

recebe o resultado

Verifico se os objetos são identicos

```

1 # Família Apply - Uma Forma Elegante de Fazer Loops
2
3 # Obs: Caso tenha problemas com a acentuação, consulte este link:
4 # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6 # Configurando o diretório de trabalho
7 # Coloque entre aspas o diretório de trabalho que você está usando no seu
8 # computador
9 # Não use diretórios com espaço no nome
10 setwd("C:/FCD/BigDataRAzure/Cap03")
11 getwd()
12
13 # apply() - arrays e matrizes
14 # tapply() - os vetores podem ser divididos em diferentes subsets
15 # lapply() - vetores e listas
16 # sapply() - versão amigável da lapply
17 # vapply() - similar a sapply, com valor de retorno modificado
18 # rapply() - similar a lapply()
19 # eapply() - gera uma lista
20 # mapply() - similar a sapply, multivariada
21 # by
22
23 # Se você estiver trabalhando com os objetos:
24
25 # list, numeric, character (list/vector) => sapply ou lapply
26 # matrix, data.frame (agregação por coluna) => by / tapply
27 # Operações por linha ou operações específicas => apply
28
29 # Usando um Loop
30 listal <- list(a = (1:20), b = (35:67)) Cria duas listas e joga em
31 # Calculando o total de cada elemento da lista com loop for lista 1
32 valor_a = 0
33 valor_b = 0
34
35 for (i in listal$a){
36   valor_a = valor_a + i Soma todos os itens
37 }
38
39 for (j in listal$b){
40   valor_b = valor_b + j
41 }
42
43 print(valor_a)
44 print(valor_b) lista
45
46 # Calculando cada elemento da lista com sapply
47 ?sapply
48 sapply(listal, sum) função Soma !
49
50 # Aplicando funções com sapply
51 sapply(listal, mean)
52
53
54 # apply()
55 ?apply
56
57 x <- matrix(c(20, 13, 65, 32, 45, 12, 76, 49, 82), nr = 3, byrow = T)
58 x
59
60 apply(x, mean) indica linha, vai calcular a média por cada
61 apply(x, 1, mean) linha do matriz
62 apply(x, 2, mean) indica coluna
63 apply(x, 1, plot)
64
65 resultapply <- apply(x, 1, mean)
66 resultapply
67
68 # Aplicando apply() a um Dataframe

```

Funções

vou um objeto
em R

←,

Cria duas listas e joga em lista 1

listal <- list(a = (1:20), b = (35:67))

Calculando o total de cada elemento da lista com loop for

valor_a = 0

valor_b = 0

for (i in listal\$a){

valor_a = valor_a + i

Soma todos os itens

}

for (j in listal\$b){

valor_b = valor_b + j

}

print(valor_a)

print(valor_b)

lista

Calculando cada elemento da lista com sapply

?sapply

sapply(listal, sum)

função Soma !

Aplicando funções com sapply

sapply(listal, mean)

apply()

?apply

x <- matrix(c(20, 13, 65, 32, 45, 12, 76, 49, 82), nr = 3, byrow = T)

x

apply(x, mean)

indica linha, vai calcular a média por cada

apply(x, 1, mean)

linha do matriz

apply(x, 2, mean)

indica coluna

apply(x, 1, plot)

resultapply <- apply(x, 1, mean)

resultapply

Aplicando apply() a um Dataframe

```

69 escola <- data.frame(Aluno = c('Bob', 'Tereza', 'Marta', 'Felipe', 'Zacarias',
70   'Elton'),
71   Fisica = c(91, 82, 75, 97, 62, 74),
72   Matematica = c(99, 100, 86, 92, 91, 87),
73   Quimica = c(56, 72, 49, 60, 59, 77))
74
75 escola
76
77 # Calculando a média por aluno
78 escola$Media = NA
79 escola → cria a coluna Média e coloca NA
80
81 escola$Media = apply(escola[7c(2, 3, 4)], 1, mean)
82 escola → por linha
83 escola$Media = round(escola$Media) → se não preenche antes da
84 escola → singulo, indica todos as linhas
85 arredondar →
86
87 # tapply()
88 ?ql
89 tabela_basquete <-
90   data.frame(equipe = gl(5, 5, labels = paste("Equipe", LETTERS[1:5])),
91     jogador = sample(letters, 25), → a b c d e f g ...
92     num_cestas = floor(runif(25, min=0, max=50)))
93
94 View(tabela_basquete)
95 summary(tabela_basquete)
96
97 # Como calcular o total de cestas por Equipe?
98
99 # tapply() vs sqldf → SQL
100 install.packages('sqldf') → instalando pacote SQL
101 library(sqldf) → chamando biblioteca
102
103 sqldf("select equipe, sum(num_cestas) from tabela_basquete group by equipe")
104
105 ?tapply → Seleciona Num-cestas → Por equipe
106 tapply(tabela_basquete$num_cestas, tabela_basquete$equipe, sum)
107 tapply(tabela_basquete$num_cestas, tabela_basquete$equipe, mean)
108
109 # by → Vetor de 1,2,3 → mais performance
110 ?by
111
112 dat <- data.frame(species=c(rep(c(1,2,3), each=5)),
113   petal.length=c(rnorm(5, 4.5, 1),
114     rnorm(5, 4.5, 1),
115     rnorm(5, 5.5, 1)),
116   petal.width=c(rnorm(5, 2.5, 1),
117     rnorm(5, 2.5, 1),
118     rnorm(5, 4, 1)))
119
120 dat$species <- factor(dat$species)
121 View(dat)
122
123 by(dat, dat$species, function(x) → coluna 5
124   # calcular o comprimento médio da pétala para cada espécie
125   mean.pl <- mean(x$petal.length)
126 })
127
128
129 # lapply()
130 ?lapply
131
132 listal <- list(a = (1:10), b = (45:77))
133 listal
134 lapply(listal, sum)
135 sapply(listal, sum) → Aplica uma função a uma lista
136

```

Cria data frame

→ cria a coluna Média e coloca NA

→ por linha

→ se não preenche antes da singulo, indica todos as linhas

→ A B C D E

→ N = randomico de cestos

→ SQL

→ instalando pacote SQL

→ chamando biblioteca

→ Seleciona Num-cestas → Por equipe

→ mais performance

→ 1ª coluna

→ Repete 5 vezes o 1,2 e 3

→ 2ª coluna com 3 vetores de 5 números

→ 3ª coluna com 3 vetores de 5 números

→ Coloco a média

→ Aplica uma função a uma lista

```
137  
138 # vapply()  
139 ?vapply  
140  
141 # A função fivenum() retorna 5 estatísticas do conjunto de dados: (minimum,  
lower-hinge, median, upper-hinge, maximum)  
142 # https://stat.ethz.ch/R-manual/R-patched/library/stats/html/fivenum.html  
143 vapply(lista1,  
144     fivenum,  
145     c(Min. = 0,  
146         "1stQu." = 0,  
147         Median = 0,  
148         "3rd Qu." = 0,  
149         Max = 0))  
150  
151  
152 # replicate  
153 ?replicate  
154 replicate(7, runif(10))  
155  
156  
157 # mapply()  
158 ?mapply  
159 mapply(rep, 1:4, 4:1)  
160  
161  
162 # rapply()  
163 ?rapply  
164  
165 lista2 <- list(a = c(1:5), b = c(6:10))  
166 lista2  
167  
168 rapply(lista2, sum)  
169 rapply(lista2, sum, how = "list")  
170  
171  
172  
173  
174  
175  
176  
177
```

```

1 # Funções Especiais
2
3 # Obs: Caso tenha problemas com a acentuação, consulte este link:
4 # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6 # Configurando o diretório de trabalho
7 # Coloque entre aspas o diretório de trabalho que você está usando no seu
8 # computador
9 # Não use diretórios com espaço no nome
10 setwd("C:/FCD/BigDataRAzure/Cap03")
11 getwd()
12
13 # unlist()
14 # Produz um vetor com os elementos da lista
15 ?unlist
16
17 lst1 <- list(6, "b", 15)
18 lst1
19 class(lst1)
20
21 unlist(lst1)
22 vec1 <- unlist(lst1)
23 class(vec1)
24 vec1
25
26 lst2 <- list(v1 = 6, v2 = list(381, 2190), v3 = c(30, 217))
27 lst2
28 unlist(lst2)          V2 Posição!
29
30 mean(unlist(lst2))
31 round(mean(unlist(lst2)))    v1      v2
32                                     6        381
33
34 # do.call()
35 # Executa uma função em um objeto
36 # *** ATENÇÃO ***
37 # As funções da família apply aplicam uma função a cada elemento de um objeto
# (substitui um loop)
38 # A função do.call aplica uma função ao objeto inteiro e não a cada elemento
# individualmente
39 ?do.call
40
41 data <- list() → cria lista data
42 N <- 100 → N = 100 → a, b, d, J, ...
43
44 for (n in 1:N) { → de 1 a 100 → 1, 2, 3, 4, ...
45   data[[n]] = data.frame(index = n, char = sample(letters, 1), z = rnorm(1))
46 } → data no posição[n] → 1, 2, 3, 4, ...
47 head(data) recebe a informação. → rnorm
48
49 do.call(rbind, data) → Aplica rbind todo obj. data → Números
50 class(do.call(rbind, data))
51
52 # lapply() x do.call()
53 y <- list(1:3, 4:6, 7:9) → cria três listas
54 y
55
56 lapply(y, sum) → 6 15 24 → soma cada lista (vetor)
57 do.call(sum, y) → 45 → Soma todos os itens, todo o objeto
58
59
60 # O resultado da função lapply() pode ser obtido de outras formas
61 # Pacote plyr
62
63 install.packages('plyr')
64 library(plyr)
65
66

```

V2 Posição!

converte uma lista com diferentes elementos em um vetor

cria lista data

N = 100

de 1 a 100

1, 2, 3, 4, ...

recebe a informação.

Aplica rbind todo obj. data

cria três listas

soma cada lista (vetor)

Soma todos os itens, todo o objeto

O resultado da função lapply() pode ser obtido de outras formas

Pacote plyr

install.packages('plyr')

library(plyr)

```

67 y <- list(1:3, 4:6, 7:9)
68 y
69
70 dplyr(y, sum)
71
72
73 # strsplit()
74 # Divide uma string ou vetor de caracteres
75
76 texto <- "Data Science Academy"
77 strsplit(texto, " ")
78
79 texto <- "Data Science Academy"
80 strsplit(texto, "")           "doto" "Science" "Academy"
81
82
83 dates <- c("1998-05-23", "2008-12-30", "2009-11-29")
84 temp <- strsplit(dates, "-")  Separa por traço e Vou umas colunas
85 temp
86 class(temp)                 A no mês dia
87
88 # Transforma a lista em matriz, fazendo antes um unlist()
89 matrix(unlist(temp), ncol = 3, byrow = TRUE)
90
91 Names <- c("Brin, Sergey", "Page, Larry",
92           "Dorsey, Jack", "Glass, Noah",
93           "Williams, Evan", "Stone, Biz")
94
95 temp <- strsplit(Names, ", ")
96 temp
97
98
99 frase <- "Muitas vezes temos que repetir algo diversas vezes e essas diversas
100 vezes parecem algo estranho"
101 palavras <- strsplit(frase, " ")[[1]] / palavras
102 unique(tolower(palavras))
103
104
105 # strsplit() com dataframes
106 antes = data.frame(attr = c(1,30,4,6), tipo = c('pao_e_agua','pao_e_agua_2'))
107 antes
108 strsplit(as.character(antes$tipo), '_e_')
109
110
111 library(stringr)
112 str_split_fixed(antes$tipo, "_e_", 2)
113
114
115 # Usando do.call()
116 antes = data.frame(attr = c(1,30,4,6), tipo = c('pao_e_agua','pao_e_agua_2'))
117 antes
118 depois <- strsplit(as.character(antes$tipo), '_e_')
119 depois
120 do.call(rbind, depois)      rbind = liga linhas      liga linhas
121
122 # Usando dplyr e tidyr
123 install.packages("dplyr")
124 install.packages("tidyr")
125 library(dplyr)
126 library(tidyr)
127
128 antes <- data.frame(
129   attr = c(1, 30 ,4 ,6 ),
130   tipo = c('pao_e_agua','pao_e_agua_2'))           conectar de concatenação para tidyr
131
132
133 antes %>% separate(tipo, c("pao", "agua"), "_e_")  Porque
134

```

Separar por espaço

Separar todos os caracteres do texto

Separa por traço e Vou umas colunas

A no mês dia

Muitas vezes temos que repetir algo diversas vezes e essas diversas vezes parecem algo estranho

palavras

unique(tolower(palavras))

str_split_fixed(antes\$tipo, "_e_", 2)

rbind = liga linhas

liga linhas

conectar de concatenação para tidyr

Porque

separador

```

1 # Pacotes e Instalação de Pacotes
2
3 # Obs: Caso tenha problemas com a acentuação, consulte este link:
4 # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6 # Configurando o diretório de trabalho
7 # Coloque entre aspas o diretório de trabalho que você está usando no seu
8 # computador
9 # Não use diretórios com espaço no nome
10 setwd("C:/FCD/BigDataRAzure/Cap03")
11 getwd()
12
13 # De onde vem as funções? Pacotes (conjuntos de funções)
14 # Quando você inicia o RStudio, alguns pacotes são
15 # carregados por padrão
16
17 # Busca os pacotes carregados
18 search()
19
20 # Instala e carrega os pacotes
21 install.packages(c("ggvis", "tm", "dplyr"))
22 library(ggvis)
23 library(tm)
24 require(dplyr)
25
26 search()
27 ?require
28 detach(package:dplyr) → Tira o pacote da memória
29
30 # Lista o conteúdo dos pacotes
31 ?ls
32 ls(pos = "package:tm")
33 ls(getNamespace("tm"), all.names = TRUE)
34
35 # Lista as funções de um pacote
36 lsf.str("package:tm")
37 lsf.str("package:ggplot2")
38 library(ggplot2)
39 lsf.str("package:ggplot2")
40
41 # R possui um conjunto de datasets preinstalados.
42
43 library(MASS)
44 data()
45
46 ?lynx
47 head(lynx)
48 head(iris)
49 tail(lynx)
50 summary(lynx)
51
52 plot(lynx)
53 hist(lynx)
54 head(iris)
55 iris$Sepal.Length
56 sum(Sepal.Length)
57
58 ?attach
59 attach(iris)
60 sum(Sepal.Length)
61

```

Você pode instalar
 vários pacotes ao mesmo
 tempo

→ Tira o pacote da memória

Para poder chamar uma coluna sem
 informar o dataset

```

1 # Expressões Regulares
2
3 # Obs: Caso tenha problemas com a acentuação, consulte este link:
4 # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6 # Configurando o diretório de trabalho
7 # Coloque entre aspas o diretório de trabalho que você está usando no seu
8 # computador
9 # Não use diretórios com espaço no nome
10 setwd("C:/FCD/BigDataRAzure/Cap03")
11 getwd()
12
13 # grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE, fixed =
14 # FALSE, useBytes = FALSE, invert = FALSE)
15 # grepl(pattern, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes =
16 # FALSE)
17 # sub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE,
18 # useBytes = FALSE)
19 # gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE,
20 # useBytes = FALSE)
21 # regexpr(pattern, text, ignore.case = FALSE, perl = FALSE, fixed = FALSE,
22 # useBytes = FALSE)
23 # gregexpr(pattern, text, ignore.case = FALSE, perl = FALSE, fixed = FALSE,
24 # useBytes = FALSE)
25
26 str <- c("Expressões", "regulares", "em linguagem R",
27         "permitem a busca de padrões", "e explorarção de textos",
28         "podemos buscar padrões em dígitos",
29         "como por exemplo",
30         "10992451280")
31
32 length(str)
33
34 # grep()
35 ?grep
36 grep("ex", str, value = F) → Procura por ex
37 grep("ex", str, value = T) → Retorna apenas a posição
38 grep("\\d", str, value = F) → Retorna o linha que contém ex
39 grep("\\d", str, value = T) → Encontrou um Padrão
40
41 # grepl()
42 ?grepl
43 grepl("\\d+", str) → Verifico se tem dígitos
44 grepl("\\D", str) → retorna True ou False para cada linha
45
46 # gsub
47 ?gsub
48 gsub("em", "***", str) → retorna qual linha tem
49 gsub("ex", "EX", str, ignore.case = T) → qualquer dígito + algo
50
51 # sub()
52 ?sub
53 sub("em", "EM", str) → Busco 'em' e Substitue por ***
54
55 # regexpr()
56 frase <- "Isso é uma string."
57 regexpr(pattern = "u", frase) → transforma ex em Ex
58
59 # gregexpr()
60 gregexpr(pattern = "u", frase) → ignora o case sensitive
61
62 str2 <- c("2678 é maior que 45 - @!!!$%",
```

Handwritten annotations:

- Annotations for grep():
 - Line 36: "Procura por ex" (Searches for ex)
 - Line 37: "Retorna apenas a posição" (Returns only the position)
 - Line 38: "Retorna o linha que contém ex" (Returns the line that contains ex)
 - Line 39: "Encontrou um Padrão" (Found a pattern)
 - Line 41: "Verifico se tem dígitos" (Check if it has digits)
 - Line 44: "retorna True ou False para cada linha indicando qual linha tem" (Returns True or False for each line indicating which line has it)
 - Line 48: "tudo que não for dígito + algo" (anything that is not a digit + something)
- Annotations for gsub():
 - Line 48: "Busco 'em' e Substitue por ***" (Searches for 'em' and substitutes it with ***)
 - Line 49: "transforma ex em Ex" (Transforms ex into Ex)
 - Line 50: "ignora o case sensitive" (ignores case sensitivity)
- Annotation for sub():
 - Line 53: "Busco 'em' e Substitue por ***" (Searches for 'em' and substitutes it with ***)
- Annotation for regexpr():
 - Line 57: "= 8 encontra a letra u na 8ª posição." (= 8 finds the letter u at the 8th position.)

63 "Vamos escrever 14 scripts R")

64
65 str2

66
67 # gsub()
68 gsub("\\d", "", str2) → Remove todos os dígitos
69 gsub("\\D", "", str2) → Remove os não dígitos
70 gsub("\\s", "", str2)
71 gsub("[!\\d]", "Q", str2)
72 gsub("[[:punct:]]", "", str2)

73

74

75

76

77

78

79

```

1 # Datas e Hora
2
3 # Obs: Caso tenha problemas com a acentuação, consulte este link:
4 # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6 # Configurando o diretório de trabalho
7 # Coloque entre aspas o diretório de trabalho que você está usando no seu
8 # computador
9 # Não use diretórios com espaço no nome
10 setwd("C:/FCD/BigDataRAzure/Cap03")
11 getwd()
12
13 # Hora e Data do sistema
14 hoje <- Sys.Date() data atual no sistema
15 hoje
16 class(hoje)
17 Sys.time()
18 Sys.timezone()
19
20 # Data - representada por Date
21 # Armazenados como número de dias desde 1 de Janeiro de 1970
22
23 # Time - representado por POSIXct
24 # Armazenados como número de segundos desde 1 de Janeiro de 1970
25
26 # Formatando Datas
27 # %d: dia do mês em 2 dígitos (13)
28 # %m: mês em 2 dígitos (01)
29 # %y: ano em 2 dígitos (82)
30 # %Y: ano em 4 dígitos (1982)
31 # %A: dia da semana (Friday)
32 # %a: dia da semana abreviado (Fri)
33 # %B: mês (July)
34 # %b: mês abreviado (Jul)
35
36
37 # Formatando hora
38 # %H: hora (00-23)
39 # %M: minuto
40 # %S: segundo
41 # %T: formado reduzido para %H:%M:%S
42 ?strptime
43
44
45 # Formatando a saída - as.Date()
46 as.Date("2018-06-28") converte String pr data
47 as.Date("Jun-28-18", format = "%b-%d-%y")
48 as.Date("28 June, 2018", format = "%d %B, %Y")
49
50
51 # Função format()
52 Sys.Date()
53 ?format
54 format(Sys.Date(), format = "%d %B, %Y") Hoje é Friday!
55 format(Sys.Date(), format = "Hoje é %A!")
56
57
58 # Convertendo Datas - as.POSIXct
59 datel <- "Jun 13, '96 hours:23 minutes:01 seconds:45"
60 datel_convert <- as.POSIXct(datel, format = "%B %d, '%y hours:%H minutes:%M
seconds:%S")
61 datel_convert
62
63
64 # Operações com Datas
65 data_de_hoje <- as.Date("2016-06-25", format = "%Y-%m-%d")
66 data_de_hoje
67 data_de_hoje + 1 Soma 1 dia

```

```

68 my_time <- as.POSIXct("2016-05-14 11:24:134")
69 my_time
70 my_time + 1 → Soma 1 Segundo
71
72 data_de_hoje = as.Date(my_time)
73 data_de_hoje = my_time
74
75
76
77 # Convertendo Data em formato específico
78 # O vetor de números pode representar o número de dias, horas ou minutos (de
79 # acordo com o que você quer converter)
80 # A Linguagem R considera o ponto de início a data de 01 de Janeiro de 1970 e
81 # contabiliza o total
82 # de horas, minutos ou segundos, aquilo que o vetor numérico representar
83 dts =
84 c(1127056501, 1104295502, 1129233601, 1113547501, 1119826801, 1132519502, 1125298801, 1113
85 289201)
86 mydates = dts
87
88 # POSIXct, armazena os segundos desde uma data específica,
89 # convertendo os valores numéricos (que podem representar horas, minutos ou
90 # segundos) desde 01 de Janeiro de 1970
91 # POSIXt é a classe principal e POSIXct e POSIXlt são subclasses.
92 # Poderíamos usar aqui apenas POSIXct, que é a subclasse (mas não podemos usar
93 # apenas a classe principal)
94 class(mydates) = c('POSIXt', 'POSIXct')
95 mydates
96 class(mydates)
97
98 mydates = structure(dts, class = c('POSIXt', 'POSIXct'))
99 mydates
100
101
102
103 # Função ISODate
104 b1 = ISODate(2011, 3, 23)
105 b1
106 b2 = ISODate(2012, 9, 19)
107 b2
108 b2 - b1
109
110 difftime(b2, b1, units = 'weeks')
111
112
113
114
115 chegada <- ymd_hms("2016-06-04 12:00:00", tz = "Pacific/Auckland")
116 partida <- ymd_hms("2011-08-10 14:00:00", tz = "Pacific/Auckland")
117
118 chegada
119 partida
120
121 second(chegada)
122 second(chegada) <- 23
123 chegada
124 wday(chegada) → week day
125 wday(chegada, label = TRUE)
126 class(chegada)
127
128 # Cria um objeto que especifica a data de inicio e data de fim
129 interval(chegada, partida)
130

```

*# Pacote lubridate /
?lubridate
install.packages("lubridate") → instalo pacote
require(lubridate) → comega pacote
ymd("20180604") → ano mês e dia
mdy("06-04-2018")
dmy("04/06/2018")*

Time zone

```

131
132 tm1.lub <- ymd_hms("2020-05-24 23:55:26")
133 tm1.lub
134
135 tm2.lub <- mdy_hm("05/25/20 08:32")
136 tm2.lub
137
138 year(tm1.lub)
139 week(tm1.lub)
140
141 tm1.dechr <- hour(tm1.lub) + minute(tm1.lub)/60 + second(tm1.lub)/3600
142 tm1.dechr
143 force_tz(tm1.lub, "Pacific/Auckland")
144
145
146 # Gerando um dataframe de datas
147 sono <- data.frame(bed.time = ymd_hms("2013-09-01 23:05:24", "2013-09-02
22:51:09",
148
149
150
151
152 sono
153 sono$eficiencia <- round(sono$sleep.time/(sono$rise.time - sono$bed.time) * 100, 1)
154 sono
155
156
157 # Gerando um plot a partir de datas
158 par(mar = c(5, 4, 4, 4))
159 plot(round_date(sono$rise.time, "day"), sono$eficiencia, type = "o", col =
"blue", xlab = "Manhã", ylab = NA)
160 par(new = TRUE)
161 plot(round_date(sono$rise.time, "day"), sono$sleep.time/3600, type = "o", col =
"red", axes = FALSE, ylab = NA, xlab = NA)
162 axis(side = 4)
163 mtext(side = 4, line = 2.5, col = "red", "Duracão do Sono")
164 mtext(side = 2, line = 2.5, col = "blue", "Eficiênciac do Sono")
165
166
167

```

```
1 # Operadores de Atribuição
2
3 # Obs: Caso tenha problemas com a acentuação, consulte este link:
4 # https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
5
6 # Configurando o diretório de trabalho
7 # Coloque entre aspas o diretório de trabalho que você está usando no seu
8 # computador
9 # Não use diretórios com espaço no nome
10 setwd("C:/FCD/BigDataRAzure/Cap03")
11 getwd()
12
13 vec1 = 1:4
14 vec2 <- 1:4
15 class(vec1)
16 class(vec2)
17
18 typeof(vec1)
19 typeof(vec2)
20
21 mean(x = 1:10)
22 x
23
24
25 mean(x <- 1:10)
26 x
```

*dentro de uma função = atribui o valor
e depois descarta x*

<- mantém o valor de x