Manipulação de dados: funções básicas

Daniel C. Mota e Kleber G. Abitante

Inserir colunas em matrix

```
ex_matriz <- matrix(c(1,2,3,4), nrow=2, byrow = T)
nova_coluna <- c(10,12,16)
ex_matriz <- cbind(ex_matriz, nova_coluna)
colnames(ex_matriz) <- NULL
```

Inserir linhas em matrix

```
nova_linha <- c(5,6)
ex_matriz <- rbind(ex_matriz, nova_linha)
rownames(ex_matriz) <- NULL
```

Apagar colunas em `matrix`

```
ex_matriz <- ex_matriz[,-3]
```

Apagar linhas em matrix

```
ex_matriz <- ex_matriz[-3,]
```

Exercício 1 (3 min.)

a. Crie um `matrix` de duas colunas, sendo que a primeira coluna deve ter os números 10 e 11 e a segunda coluna os números 35 e 42.

b. Crie um `vector` com os números 0 e 4 e adicione como uma nova linha na `matrix` criada.

c. Crie um `vector` com os números 100, 91 e 10 e adicione como uma nova coluna na `matrix` criada.

Inserir colunas em data.frame

a. Há duas formas para inserir colunas:

- `iris[,"Petal.Length2"] <- iris[,"Petal.Length"] * 2`
- `iris$Petal.Length2 <- iris$Petal.Length * 2`

Inserir linhas em `data.frame`

- Usar a função `rbind()`: `novaLinha <- iris[150,]`
  `iris <- rbind(iris, novaLinha)`

Apagar colunas em data.frame

a. Duas formas:

- `iris[,"Petal.Length2"] <- NULL`
- `iris <- iris[,c("Sepal.Length","Sepal.Width", "Petal.Length","Petal.Width","Species")]`

Apagar linhas em `data.frame`

a. Duas formas:

- `iris <- iris[-1,]`
- `iris <- iris[c(2:150),]`

Exercício 2 (3 min.)

a. Carregar o arquivo "res_prim.xlsx" para a memória do R;

b. Renomear as colunas para "data", "prim_gov_central" e "prim_gov_reg";

c. Crie a coluna "prim_set_pub" (resultado primário do setor público), formada pela soma das colunas "prim_gov_central" e "prim_gov_reg";

d. Crie um novo `data.frame` formado apenas pelas colunas "data" e "prim_set_pub";

e. Por fim, filtrar apenas as linhas que possuem "prim_set_pub" maior que zero no novo `data.frame` criado.

Selecionar elementos em um `list`

- Exemplo de como selecionar elementos em um `list`: `ex_lista <- list(ex_vector = c(10,5,1.3,9), ex_dataFrame = data.frame(ano=c(2021,2022), valor=c(1.1,2.7)), ex_listaDentro = list(ex_vector2=c(20,0,7),ex_vector3=c(1,0.3,4)))`

```
ex_lista[1]
ex_lista$ex_vector
ex_lista[[1]]
ex_lista[c(1,2)]
ex_lista[c("ex_vector","exDataFrame")]
ex_lista[[3]][[1]]
ex_lista[[3]]$ex_vector2
ex_lista$ex_listaDentro$ex_vector2
```

- Atenção: `class(ex_lista[1])` é diferente de `class(ex_lista[[1]])`.

Exercício 3 (2 min.)

a. Crie um `list` contendo os seguintes elementos:

- um `vector` com o nome `valores1` contendo os valores 1.1, 2 e 7;
- um `data.frame` com o nome `valores2` contendo uma coluna chamada "ano" 2020, 2021 e 2022 e "val" com os valores 3, 7 e 2.

b. Selecione o `vector` de dentro da lista e atribua para uma nova variável;

c. Selecione o `data.frame` de dentro da lista e atribua para uma nova variável.

Localizar/substituir caracteres

- A função `grep()` retorna o número do elemento no vetor que contém o caracter procurado. `grep("a", c("vidro","casa"))`
- A função `grepl()` retorna um vetor de `TRUE` ou `FALSE` a respeito do caracter procurado. `grepl("a", c("vidro","casa"))`
- A função `sub()` substitui apenas a primeira ocorrência de um caracter. `sub("a", "w", c("vidro","casa"))`
- A função `gsub()` substitui todas as ocorrências de um caracter. `gsub("a", "w", c("vidro","casa"))`

Exercício 4 (2 min.)

a. Crie o `vector` "prec_ativ" contendo os valores `c("pib","ipca","IBC-Br","data")`.

b. Localize o número da posição em que se encontra a palavra "data" no vetor.

c. Procure a palavra "PIB" no vetor e retorne um vetor de valores lógicos (`TRUE` ou `FALSE`).

d. Substitua a palavra "pib" por "PIB" e a palavra "ipca" por "IPCA".

Regular expressions

- *Regular expressions* ou *regex* é um caracter ou uma sequência de caracteres que descrevem um certo padrão encontrado em um texto[1]
- Exemplo: `[a-z]`: retornar todos os caracteres minúsculos de a até z.

[1] DATACAMP. A guide to R regular expressions. 2022. Disponível em: https://www.datacamp.com/tutorial/regex-r-regular-expressions-guide. Acesso em: 07 mar. 2023

Regular expressions - posição

```
strings <- c("abcd", "cdab", "cabd", "c abd")
```

a. `^`: buscar no início do grupo de caracteres;
   - `^ab`= "abcd"

b. `$`: buscar no final do grupo de caracteres;
   - `ab$` = "cdab"

c. `\\b`: buscar no início ou após um espaço em branco. Deve-se acrescentar um `\` para realizar o *escape* do caracter;
   - `\\bab` = "abcd", "c abd"

d. `\\B`: buscar exceto no início ou após espaço em branco. Deve-se também acrescentar o `\`.
   - `\\Bab`: "cdab", "cabd"

Exercício 5 (3 min.)

a. Crie o vetor `dados` contendo os seguintes valores: `c("rstudio", "software r", "carro", "erroPerroB")`.

b. Use *regular expressions* quantas vezes forem necessárias para que o vetor fique da seguinte forma: `c("RStudio", "software R", "Carro", "PIB")`. Dica: para eliminar algum caracter, informe que você deseja substituir o caracter por `""` (abra e feche aspas).

Regular expressions - classes de caracteres

```
numLetra <- c("1ipca2", "PIB2030", "Receita998")
```

a. `[0-9]`: buscar todos os dígitos de 0 a 9;
   - `gsub("[0-9]", "", numLetra)` = c("ipca", "PIB", "Receita")

b. `[a-z]`: buscar todas as letras de "a" a "z" apenas em minúsculo;
   - `gsub("[a-z]", "", numLetra)` = c("12", "PIB2030", "R998")

c. `[A-Z]`: buscar todas as letras de "A" a "Z" apenas em maiúsculo;
   - `gsub("[A-Z]", "", numLetra)` = C("1ipca2", "2030", "eceita998")

d. `[A-z]`: buscar as letras de "A" a "z", tanto maiúsculas quanto minúsculas.
   - `gsub("[A-z]", "", numLetra)` = c("12", "2030", "998")

Exercício 6 (2 min.)

a. Crie o vetor `num_car` com os seguintes valores: 'c("10IPCA", "2023pib", "ativi99dade").

b. Remova todos os números do vetor e atribua o resultado para um novo vetor chamado `num_car2`.

Função `which()`

- Retorna os índices dos valores que são `TRUE` em uma comparação: `which(iris$Species=="setosa")`
- Pode ser usando para filtrar linhas: `iris[which(iris$Species=="setosa"),]`

Exercício 7 (2 min.)

a. Carregar o arquivo "res_prim.xlsx" para a memória do R (já foi carregado em exercício anterior);

b. Filtre a tabela usando a função `which()` para filtrar apenas as linhas com a coluna `res_prim_gov_central` menor que zero.

Função `summary()`

- A função `summary()` serve para mostrar uma tabela com estatísticas descritivas de valores numéricos.
  `summary(iris$Sepal.Length)`

Exercício 8 (2 min.)

Use a função `summary()` para gerar um tabela de estatísticas descritivas do arquivo "res_prim.xlsx" carregado em exercícios anteriores. Lembre-se de filtrar as colunas para não incluir a coluna de ano na função `summary`.

Funções e operadores matemáticos

- `max()`: extrair o valor máximo;
- `min()`: extrair o valor mínimo;
- `exp()`: calcular a exponencial de um número;
- `log()`: calcular o log;
- `sqrt()`: calcular a raiz quadrada;
- `abs()`: calcular o valor absoluto;
- `x%%y`: retornar o resto da divisão de `x` por `y`;
- `mean()`: calcula a média de um vetor de números;
- `median()`: calcula a mediana de um vetor de números.

Exercício 9 (2 min.)

Utilize o mesmo `data.frame` do exercício anterior e calcule os valores máximo, mínimo e média das colunas `res_prim_gov_central` e `res_prim_gov_reg`.

Função `if()`

- A função `if` avalia se um valor é `TRUE` e, caso positivo, executa o código que está entre as chaves:

```
x <- 10
y <- 2
w <- 20
if(x>y){
print("x é maior que y")
}
if(x>y&w>x){
print("x é maior que y e w é maior de x")
}
```

Função else()

- A função `else` executa o código que está entre as chaves se o resultado do `if` for `FALSE`:

```
x <- 0
y <- 2
if(x>y){
print("x é maior que y")
} else {
print("x é menor que y")
}
```

Exercício 11 (2 min.)

a. Crie as seguintes variáveis:

- `ipca_expec` (expectativa do IPCA para 2023): com o valor de 0.059;
- `selic` (meta da taxa Selic): com o valor de 0.1375.

b. Utilize o `if` e o `else` para verificar se a `selic` é maior que o `ipca`:

- em caso positivo, crie a variável `selic_real`, a qual corresponde à Selic real. Fórmula da Selic real:

$$selic\_real_t = \frac{1 + selic_t}{1 + ipca\_expec_t} - 1 \tag{1}$$

- em caso negativo, imprima a mensagem: "taxa de juros real negativa".

Função ifelse()

- A função ifelse() é a versão vetorizada da função if():

```
library(jsonlite)
# url da serie 7326
urlPIB <-
"https://api.bcb.gov.br/dados/serie/bcdata.sgs.7326/dados/ultimos/24
?formato=json"
# baixar os dados
pib <- fromJSON(urlPIB)
# transformar a coluna do pib para numero
pib$valor <- as.numeric(pib$valor)
# ordenar por data
pib <- pib[order(pib$data),]
# criar uma coluna vazia de avaliacao do PIB
pib$aval <- NA
# preencher a coluna de avaliacao
pib$aval <- ifelse(pib$valor>0, "crescimento", pib$aval)
pib$aval <- ifelse(pib$valor=0, "estabilidade", pib$aval)
pib$aval <- ifelse(pib$valor<0, "redução", pib$aval)
```

Exercício 12 (2 min.)

Utilize o mesmo `data.frame` do exercício anterior e crie uma nova coluna nele chamada `avaliacao` e use o `ifelse` para preenchê-la com as palavras "deficit" se o valor da coluna "res_prim_gov_central" for menor que zero e "superavit" se for maior igual a zero.

Função for()

- A função `for()` é usado para gerar um *loop* de iteração.

```
# criar um vetor
pib <- c(1.1, 0.5, 2.5, 2)
# imprimir cada valor no console
for (x in pib) {
valorImp <- paste0("Valor: ", x)
print(valorImp)
}
ou: for (x in c(1:length(pib))) {
valorImp <- paste0("Valor: ", x)
print(valorImp)
}
```

Exercício 13 (3 min.)

Utilize a tabela do exercício anterior e a função `for()` para criar uma nova coluna na tabela, chamada "res_prim_gov_reg_diff", a qual é formada pela diferença entre `t` e `t-1` da coluna "res_prim_gov_reg". Dica: o `for` deverá percorrer de 2 até o número de linhas da tabela.

Função `while()`

- A função `while()` é utilizada para executar um código enquanto uma condição for `TRUE`. Atenção: sempre assegure que a condição é atualizada a cada *loop* do `while()`.

```
x <- 0
while(x <= 10){
print(x)
x <- x + 1
}
```

Exercício 14 (2 min.)

Crie uma variável chamada `ipca1`, com valor de 5, uma variável chamada `ipca2` com valor de 4 e uma variável chamada `meta` que é a diferença entre `ipca1` e `ipca2`. Use o `while()` para ir reduzindo a variável `ipca1` em 0.1 a cada *loop*, até que `meta = 0`.

Escrevendo funções

- Quando há necessidade de se repetir um trecho de código ao longo de um aplicativo, o ideal é transformar esse trecho de código em um função.
- Por exemplo: deflacionar e dessazonalizar séries temporais.
- A função `function()` serve para escrevermos as nossas funções personalizadas.

Escrevendo funções - sem argumento padrão

```
soma <- function(numero1, numero2){
resultado <- numero1 + numero2
return(resultado)
}
w <- soma(2, 3)
```

- A função `return` retorna alguma variável ou objeto de dentro da função e, em seguida, termina a execução da função. Se não for usado o `return`, a função não irá retornar nada.

Escrevendo funções - sem argumento padrão

```
# criar um data.frame
novo_df <- data.frame(valor1 = c(5,9,10), valor2 = c(1,3,10))
# criar uma funcao que multiplica duas colunas quaisquer de um data.frame
somar_cols <- function(df, coluna1, coluna2){
df[,"multiplicacao"] <- df[,coluna1] * df[,coluna2]
return(df)
}
# utilizar a funcao

somar_cols(novo_df, "valor1", "valor2")
```

Escrevendo funções - com argumento padrão

```
soma <- function(numero1, numero2 = 0){
resultado <- numero1 + numero2
return(resultado)
}

w <- soma(2, 3)
```

Escrevendo funções - *ellipses*

- Os três pontinhos (`...`) no argumento de uma função são chamados de `ellipsis` e significam que é possível que a função receba um ou mais argumentos, que não estão explícitos, no momento da sua execução.
  Exemplo:
  `median(x, na.rm = FALSE, ...)`

Escrevendo funções - *ellipses*

```
ellipis_exemp <- function(...) {
cat("Eu possuo os seguintes argumentos:\n\n")
print(list(...))
}

ellipis_exemp(x = 3, y = "ipca", z = FALSE)
```

Escrevendo funções - *ellipses*

```
soma <- function(numero1 , numero2, ...){
valor <- list(...)
valor <- unlist(valor)
valor <- sum(valor)
resultado <- numero1 + numero2 + valor
return(resultado)
}
soma(2, 3, 10, 7, 1)
```

Exercício 15 (3 min.)

a. Crie uma função que tenha 3 argumentos: um `data.frame`, um `character` representando o nome de uma coluna e outro `character` representando o nome de outra coluna. Esta função deve somar as duas colunas informadas pelo usuário, atribuir o resultado em uma nova coluna do `data.frame` informado pelo usuário chamada "resultado" e retornar o `data.frame` acrescido da nova coluna;

b. Teste a função com o `data.frame` gerado pelo carregamento do arquivo "res_prim.xlsx" utilizado nos exercícios anteriores. Informe as colunas "res_prim_gov_central" e "res_prim_gov_reg" para serem somadas. Dica: referencie a coluna da seguinte forma: `df[,coluna]`.

Verificando o interior da função

- Quando ocorre um problema na função, é necessário examinar o que está acontecendo em seu interior.
- Isto pode ser feito inserir a função `browser()` no início da função ou antes do local onde acredita-se que ocorra o erro
- A função `browser()` faz com que a função pare a sua execução no ponto em que o `browser()` se encontra e o programador pode executar as linhas restantes uma de cada vez pela tecla `F10`.

Verificando o interior da função

```
soma <- function(numero1 , numero2, ...){
browser()
valor <- list(...)
valor <- unlist(valor)
valor <- sum(valor)
resultado <- numero1 + numero2 + valor
return(resultado)
}
soma(2, 3, 10, 7, 1)
```

Exercício 16 (2 min.)

a. Utilize a mesma função criada no exercício anterior e insira o função `browser()` no começo dela;

b. Execute a função utilizando o `data.frame` gerado pelo carregamento do arquivo "res_prim.xlsx" e, após ela parar no `browser()`, execute-a linha-a-linha com a tecla `F10`.

Apêndice

Apêndice - Criar um array a partir de um data.frame

```
# series do SGS
series_sgs <- c(20545, # Saldo - Pessoas jurídicas - Desconto de cheques
20552, # Saldo - Pessoas jurídicas - Cheque especial
20553, # Saldo - Pessoas jurídicas - Aquisição de veículos
20591, # Saldo - Pessoas físicas - Desconto de cheques
20573, # Saldo - Pessoas físicas - Cheque especial
20581, # Saldo - Pessoas físicas - Aquisição de veículos
20637, # Concessões - Pessoas jurídicas - Desconto de cheques
20644, # Concessões - Pessoas jurídicas - Cheque especial
20645, # Concessões - Pessoas jurídicas - Aquisição de veículos
20683, # Concessões - Pessoas físicas - Desconto de cheques
20665, # Concessões - Pessoas físicas - Cheque especial
20673  # Concessões - Pessoas físicas - Aquisição de veículos
)
# vetor de tipo de pessoa
pessoa <- c("PJ","PJ","PJ","PF","PF","PF","PJ","PJ","PJ","PF","PF","PF")
operacao <- c("Desconto de cheques","Cheque especial","Aquisição de veículos",
"Desconto de cheques","Cheque especial","Aquisição de veículos",
"Desconto de cheques","Cheque especial","Aquisição de veículos",
"Desconto de cheques","Cheque especial","Aquisição de veículos")
# vetor de tipo de operacao
tipo_op <- c("saldo","saldo","saldo","saldo","saldo","saldo","conc","conc","conc","conc","conc","conc")
# criar tabela conjunta

dados_cred <- data.frame(series_sgs = series_sgs, pessoa = pessoa, operacao = operacao, tipo_op = tipo_op)
```

Apêndice - Criar um array a partir de um data.frame

```
library(tidyr)
library(stringr)
library(reshape2)
library(readxl)
# carregar os dados
saldo_conc <- read_excel("saldo_conc.xlsx")
# transformar para o formato long
saldo_conc_2 <- saldo_conc %>% gather("variable", "value", -data)
# extrair o numero das series
saldo_conc_2$codigo1 <- as.numeric(str_sub(saldo_conc_2$variable, 7, 11))
# unir a tabela de valores com a tabela de descricao das series
saldo_conc_2 <- merge(saldo_conc_2, dados_cred, by.x="codigo1", by.y="series_sgs", all.x=T)
# manter apenas as colunas de interesse
saldo_conc_2 <- saldo_conc_2[,c("data","pessoa","operacao","tipo_op","value")]
# renomear as colunas
colnames(saldo_conc_2) <- c("data","pessoa","tipo_oper","tipo_valor","valor")
# gerar o array
array_teste <- acast(saldo_conc_2, data~pessoa~tipo_oper~tipo_valor, value.var="valor")
# renomear as dimensoes
dimnames(array_teste) <- list(data = dimnames(array_teste)[[1]],pessoa=dimnames(array_teste)[[2]],
tipo_oper = dimnames(array_teste)[[3]], valor = dimnames(array_teste)[[4]])
# exemplo de filtro do array

array_teste[,"PF",,"saldo"]
```