Café com estatística e R

Treinamento 2 - Importação e manipulação de dados no R e estatística descritiva

Marcelo Teixeira Paiva

2025-10-07

Abstract

Relatório do segundo treinamento onde foi apresentado como importar dados e manipulá-los no R, bem como as principais estatísticas descritivas univariadas e multivariadas.

Índice

Lista de Figuras

Lista de Tabelas

```
# Pacotes
library(tidyverse)
library(gridExtra)
library(plotly)
library(gt)
library(tidyverse)

# Tema personalizado para gráficos
tema_didatico <- theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 14),
    plot.subtitle = element_text(face = "italic", size = 11),
    axis.title = element_text(size = 12),
    legend.position = "top",
    panel.grid.minor = element_blank()
  )

cores <- c("#FF6B6B", "#4ECDC4", "#45B7D1", "#96CEB4", "#FFEAA7")
```

Chapter 1

Importação de dados no R

1.1 Pacotes necessários

Pacotes (**package**) são coleções de funções, dados e documentação que estendem as capacidades do R base (aquele que você recebe na instalação padrão). São como "caixas de ferramentas" especializadas que você adiciona ao R para realizar tarefas específicas, então você tem pacotes para elaboração de gráficos, para certos tipos de análises, para manipulação de dados, para leitura (importação) de dados. Em https://cran.r-project.org/web/views/ há uma "breve" lista de pacotes conforme a sua finalidade.

```
# funções no R base
length(ls("package:base"))
```

```
[1] 1270
```

```
# funções especializadas no pacote dplyr
length(ls("package:dplyr"))
```

```
[1] 297
```

```
# um pacote possui um conjunto de arquivos associados
system.file(package = "ggplot2") %>% list.files()
```

```
 [1] "CITATION"    "data"        "DESCRIPTION" "doc"         "help"
 [6] "html"        "INDEX"       "LICENSE"     "Meta"        "NAMESPACE"
[11] "NEWS.md"     "R"
```

Por padrão, ao iniciar uma sessão no R, serão carregados os pacotes e funções associados ao R base. Os demais devem ser instalados primeiramente, e depois carregados na seção para serem usados.

```
# pacotes carregados no seu ambiente
search()
```

```
 [1] ".GlobalEnv"        "package:gt"        "package:plotly"
 [4] "package:gridExtra" "package:lubridate" "package:forcats"
 [7] "package:stringr"   "package:dplyr"     "package:purrr"
[10] "package:readr"     "package:tidyr"     "package:tibble"
[13] "package:ggplot2"   "package:tidyverse" "package:stats"
[16] "package:graphics"  "package:grDevices" "package:utils"
[19] "package:datasets"  "package:methods"   "Autoloads"
```

```
[22] "package:base"
```

```
# pacotes instalados
instalados <- installed.packages()[, "Package"]
instalados[1:4]
```

```
      abind      ARTool     askpass   backports
    "abind"    "ARTool"   "askpass" "backports"
```

```
length(instalados)
```

```
[1] 330
```

```
# verificando se um pacote já está instalado
sum(installed.packages()[, "Package"] == 'dplyr')
```

```
[1] 1
```

```
any(installed.packages()[, "Package"] == 'dplyr')
```

```
[1] TRUE
```

```
"ggplot2" %in% rownames(installed.packages())
```

```
[1] TRUE
```

A instalação de pacotes no R é feita usando a função `install.packages` ou `devtools::install_github` para pacotes que estão no github e não em um repositório de pacotes.

```
# pelo repositório oficial (na web)
install.packages("ggplot2")
install.packages(c("dplyr", "tidyr", "readr")) # instalando vários pacotes de uma vez

# Instalar o pacote e todas dependências relacionadas a ele
install.packages("ggplot2", dependencies = TRUE)

# instalar de um arquivo local
install.packages("caminho/para/pacote.tar.gz", repos = NULL, type = "source")

# Instalar pacote mantido no GitHub
install.packages("devtools")
devtools::install_github("tidyverse/ggplot2")

# Usar outros repositórios para instalação
install.packages("ggplot2", repos = "https://cloud.r-project.org/")
```

Para carregar um pacote em uma sessão usamos `library()` ou `require()`. A diferença entre os dois é que, na ausência do pacote que você pretende carregar, `library` gera um erro, enquanto o `require` retorna um valor `FALSE` invisível, o qual pode ser usado, por exemplo, para criar uma lógica em seu script para instalar o pacote caso o mesmo não possa ser carregado ou, então, para gerar uma mensagem no terminal indicando essa ausência do pacote.

```
library(ggplot2)

# Não exibir mensagens de carregamento do pacote
suppressPackageStartupMessages(library(ggplot2))
```

```r
# criando uma lógica simples com require para instalar pacotes que
# não possam ser carregados
if(!require(ggplot2)) {
  install.packages("ggplot2")
  require(ggplot2)
}

# usando uma função do pacote sem o carregar (namespace qualification)
head(dplyr::filter(mtcars, mpg > 20), 2)
```

```
              mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```r
# carregando vários pacotes de uma lista de nomes
pacotes <- c("ggplot2", "dplyr", "tidyr")
x <- lapply(pacotes, library, character.only = TRUE, quietly = TRUE)
```

A importação de dados é o primeiro passo crucial em qualquer análise. O R oferece múltiplos pacotes especializados para diferentes formatos, cada um com suas vantagens.

1.1.1 Pacotes Principais para Importação:

```r
# Instalar os pacotes necessários (execute uma vez)
install.packages(c("readxl", "writexl", "haven", "rio", "openxlsx"))

# Carregar pacotes
library(readxl)    # Excel
library(haven)     # SAS, SPSS, STATA
```

1.2 2. IMPORTANDO DADOS DO EXCEL

1.2.1 2.1 Usando `readxl` (Recomendado - Não requer Java)

```r
library(readxl)

# Importar primeira planilha
dados_excel <- read_excel("caminho/arquivo.xlsx")

# Especificar planilha por nome ou índice
dados_aba2 <- read_excel("arquivo.xlsx", sheet = "Vendas")
dados_aba3 <- read_excel("arquivo.xlsx", sheet = 2)

# Ver nomes das planilhas disponíveis
excel_sheets("arquivo.xlsx")

# Especificar intervalo de células
dados_range <- read_excel("arquivo.xlsx",
                          range = "A1:E100",
```

```
                          sheet = "Dados")

# Controlar tipos de colunas
dados_tipos <- read_excel("arquivo.xlsx",
                          col_types = c("text", "numeric", "date", "logical"))

# Pular linhas iniciais e definir nomes de colunas
dados_custom <- read_excel("arquivo.xlsx",
                           skip = 2,
                           col_names = c("ID", "Nome", "Valor", "Data"))

# Tratar valores missing
dados_na <- read_excel("arquivo.xlsx",
                       na = c("", "NA", "N/A", "-"))
```

1.2.2 2.2 Usando `openxlsx` (Para leitura e escrita avançada)

```
library(openxlsx)

# Leitura básica
dados_openxlsx <- read.xlsx("arquivo.xlsx", sheet = 1)

# Com opções avançadas
dados_adv <- read.xlsx("arquivo.xlsx",
                       sheet = "Dados",
                       startRow = 2,
                       colNames = TRUE,
                       detectDates = TRUE,
                       skipEmptyRows = TRUE)

# Ler múltiplas planilhas de uma vez
todas_planilhas <- lapply(getSheetNames("arquivo.xlsx"),
                          function(x) read.xlsx("arquivo.xlsx", sheet = x))
names(todas_planilhas) <- getSheetNames("arquivo.xlsx")
```

1.3 3. IMPORTANDO DADOS DO SAS

1.3.1 3.1 Usando `haven`

```
library(haven)

# Importar arquivo SAS (.sas7bdat)
dados_sas <- read_sas("arquivo.sas7bdat")

# Com opções adicionais
dados_sas_custom <- read_sas("arquivo.sas7bdat",
                             encoding = "UTF-8",
                             col_select = c("var1", "var2", "var3"))
```

```r
# Importar arquivo de transporte SAS (.xpt)
dados_xpt <- read_xpt("arquivo.xpt")

# Preservar labels e atributos
dados_labels <- read_sas("arquivo.sas7bdat")
# Ver labels das variáveis
sapply(dados_labels, function(x) attr(x, "label"))

# Converter labels em nomes de colunas
library(labelled)
dados_clean <- remove_labels(dados_labels)
```

1.4 4. IMPORTANDO DADOS DO SPSS

1.4.1 4.1 Usando haven

```r
library(haven)

# Importar arquivo SPSS (.sav)
dados_spss <- read_sav("arquivo.sav")

# Com seleção de variáveis
dados_spss_sel <- read_sav("arquivo.sav",
                           col_select = c("id", "idade", "sexo"))

# Converter variáveis categóricas com labels
dados_spss_factor <- read_sav("arquivo.sav") %>%
  as_factor()  # Converte labeled para factor

# Preservar valores missing do SPSS
dados_missing <- read_sav("arquivo.sav",
                          user_na = TRUE)  # Mantém missing values definidos

# Verificar labels e atributos
View(dados_spss)  # No RStudio mostra labels
print(dados_spss, n = 5)  # Mostra primeiras linhas com info de labels
```

1.5 5. IMPORTANDO DADOS DO STATA

1.5.1 5.1 Usando haven

```r
library(haven)

# Importar arquivo Stata (.dta)
dados_stata <- read_dta("arquivo.dta")

# Especificar versão do Stata (se necessário)
dados_stata_v14 <- read_dta("arquivo.dta",
```

```
                            encoding = "UTF-8")

# Preservar value labels
dados_stata_labels <- read_dta("arquivo.dta") %>%
  as_factor()

# Selecionar variáveis específicas
dados_stata_sel <- read_dta("arquivo.dta",
                            col_select = starts_with("var"))

# Ver notas e características do dataset
# Notas do Stata
attr(dados_stata, "notes")
# Labels das variáveis
sapply(dados_stata, function(x) attr(x, "label"))
```

1.6 6. USANDO O PACOTE `rio` (SOLUÇÃO UNIVERSAL)

O pacote `rio` simplifica a importação detectando automaticamente o formato:

```
library(rio)

# Import detecta automaticamente o formato pela extensão
dados_excel_rio <- import("arquivo.xlsx", sheet = "Dados")
dados_sas_rio <- import("arquivo.sas7bdat")
dados_spss_rio <- import("arquivo.sav")
dados_stata_rio <- import("arquivo.dta")

# Importar múltiplos arquivos de uma vez
arquivos <- c("dados1.xlsx", "dados2.csv", "dados3.dta")
lista_dados <- import_list(arquivos)

# Converter entre formatos facilmente
convert("arquivo.sav", "arquivo.csv")  # SPSS para CSV
convert("arquivo.dta", "arquivo.xlsx") # Stata para Excel
```

1.7 7. VERIFICAÇÃO E DIAGNÓSTICO PÓS-IMPORTAÇÃO

```
# Workflow de verificação após importação
verificar_dados <- function(dados) {
  cat("Dimensões:", dim(dados), "\n")
  cat("Tipos de variáveis:\n")
  print(sapply(dados, class))
  cat("\nPrimeiras linhas:\n")
  print(head(dados, 3))
  cat("\nResumo estatístico:\n")
  print(summary(dados))
  cat("\nValores missing por coluna:\n")
```

```r
  print(colSums(is.na(dados)))
  cat("\nStructura dos dados:\n")
  str(dados)
}

# Aplicar a qualquer dataset importado
verificar_dados(dados_excel)
```

1.8 8. TRATAMENTO DE PROBLEMAS COMUNS

1.8.1 8.1 Encodings e Caracteres Especiais

```r
# Para problemas com acentos/caracteres especiais
dados_utf8 <- read_excel("arquivo.xlsx", locale = readr::locale(encoding = "UTF-8"))
dados_latin1 <- read_sas("arquivo.sas7bdat", encoding = "LATIN1")

# Verificar e corrigir encoding
Encoding(dados$nome_coluna) <- "UTF-8"
```

1.8.2 8.2 Datas e Horários

```r
library(lubridate)

# Converter datas após importação
dados$data <- as.Date(dados$data, format = "%d/%m/%Y")
# ou
dados$data <- dmy(dados$data)  # usando lubridate

# Para SPSS/SAS/Stata com datas numéricas
dados$data_corrigida <- as.Date(dados$data_numerica, origin = "1960-01-01")
```

1.8.3 8.3 Labels e Fatores

```r
library(labelled)

# Remover todos os labels (converte para R padrão)
dados_limpo <- zap_labels(dados_spss)
dados_limpo <- zap_formats(dados_limpo)

# Converter labels em fatores seletivamente
dados_mixed <- dados_spss %>%
  mutate(sexo = as_factor(sexo),
         idade = zap_labels(idade))  # mantém numérico
```

1.9 9. EXEMPLO COMPLETO: PIPELINE DE IMPORTAÇÃO

```r
# Pipeline reprodutível de importação e preparação
library(tidyverse)
library(haven)
library(here)

# Definir caminho relativo ao projeto
caminho_dados <- here("data", "raw")

# Função genérica de importação com tratamento de erros
importar_seguro <- function(arquivo, ...) {
  tryCatch({
    # Detectar formato pela extensão
    ext <- tools::file_ext(arquivo)

    dados <- switch(ext,
      "xlsx" = read_excel(arquivo, ...),
      "xls" = read_excel(arquivo, ...),
      "sav" = read_sav(arquivo, ...) %>% as_factor(),
      "dta" = read_dta(arquivo, ...) %>% as_factor(),
      "sas7bdat" = read_sas(arquivo, ...),
      stop("Formato não suportado: ", ext)
    )

    # Log de sucesso
    cat("  Importado com sucesso:", basename(arquivo), "\n")
    cat("  Dimensões:", nrow(dados), "x", ncol(dados), "\n")

    return(dados)

  }, error = function(e) {
    cat("  Erro ao importar", basename(arquivo), ":\n")
    cat("  ", conditionMessage(e), "\n")
    return(NULL)
  })
}

# Usar a função
dados <- importar_seguro(file.path(caminho_dados, "pesquisa.sav"))
```

1.10 10. PERFORMANCE E BOAS PRÁTICAS

```r
# Para arquivos grandes, considere:

# 1. Ler apenas colunas necessárias
dados_sel <- read_sav("arquivo_grande.sav",
                      col_select = c("id", "var1", "var2"))
```

```r
# 2. Para Excel grandes, use readxl com range
dados_parte <- read_excel("arquivo_grande.xlsx",
                          range = cell_limits(c(1, 1), c(10000, 20)))

# 3. Considere converter para formatos mais eficientes
library(arrow)
write_parquet(dados, "dados.parquet")  # Muito mais rápido e compacto
dados_fast <- read_parquet("dados.parquet")

# 4. Para múltiplos arquivos similares
library(purrr)
arquivos <- list.files("data/", pattern = "\\.sav$", full.names = TRUE)
todos_dados <- map_df(arquivos, read_sav)  # Combina todos em um dataframe
```

1.11 RESUMO DAS RECOMENDAÇÕES

Formato	Pacote Recomendado	Função Principal	Observações
Excel	`readxl`	`read_excel()`	Não requer Java, rápido
SAS	`haven`	`read_sas()`	Preserva labels
SPSS	`haven`	`read_sav()`	Use `as_factor()` para labels
Stata	`haven`	`read_dta()`	Suporta Stata 13+
Múltiplos	`rio`	`import()`	Detecta automaticamente

Dica Final: Sempre verifique a integridade dos dados após importação usando `str()`, `summary()` e `head()`. Documente o processo de importação em um script para garantir reprodutibilidade!