

Análise de Dados e Relatórios com R

David Souza Pinto

27 Maio 2022

VIII Encontro Fluminense de Engenharia de Produção (ENFEPro)

1. Escopo do minicurso
2. Introdução ao **R**
3. Boas práticas
4. Análise de Dados com **R**
5. Produção de relatórios

- Mestre em Engenharia de Produção (PUC-Rio)
- Engenheiro de Produção (UFF)
- Lean Six Sigma Black Belt
- Áreas de atuação: Gerência de Operações, Aprendizado Estatístico, Pesquisa Operacional, Inteligência de Negócios

Empregando R, como:

- Ler arquivos tabulares?
- Agregar e combinar dados, gerar medidas e gráficos?
- Manipular datas?
- Criar relatórios?
- Estruturar a análise para que seja reproduzível?

- `{openxlsx}`: leitura e escrita de arquivos XLSX [10].
- `{data.table}`: Processamento de dados tabulares [3].
- `{ggplot2}`: Geração de gráficos [12].
- `{lubridate}`: Processamento e ajuste de datas [4].

Introdução

O **R** é uma linguagem de programação e um ambiente para computação numérica, que conta com uma série de ferramentas numéricas e gráficas para manipular dados, realizar cálculos e criar gráficos [2].

O **RStudio** é um ambiente de desenvolvimento integrado (em inglês IDE, *integrated development environment*) para auxiliar o trabalho com a linguagem R, possuindo uma versão *open source* [8].

Por que usar o R?

O R é software livre *open source*, cujas funcionalidades podem ser ampliadas seja através de desenvolvimento local ou empregando pacotes disponibilizados no CRAN (*Comprehensive R Archive Network*) que cobrem diversos domínios numéricos. [2]

Por que programar?

- Desempenho.
- Escalabilidade.
- Reprodutibilidade.
- Evitar ou eliminar o retrabalho.

Introdução ao R

Para representar dados e informações a linguagem possui alguns tipos e estruturas já implementados [11, 15]:

- `numeric`: contém os tipos `integer` (93L), `double` (3.1415), `complex` (5+3i, -10i).
- `character`: representado usando aspas simples ou duplas (`'String'`, `"Outra string"`).
- `logical`: representa valores booleanos (verdadeiro ou falso).
- `factors`: emprego de inteiros para representar algumas categorias pré-determinadas.
- `Date`, `POSIXct`: representação de datas empregando tipos numéricos.
- `NA`: uma representação de dados que não existem (em inglês, *not applicable*).

Além dessas representações, o **R** também conta com as seguintes estruturas de dados:

Dimensões	Homogêneo	Heterogêneo
1	Vetor: <code>c()</code>	Lista: <code>list()</code>
2	Matriz: <code>matrix()</code>	Data Frame: <code>data.frame()</code>
n	Array: <code>array()</code>	—

Os dados podem ser acessados por índices (posição) através da seguinte notação:

1. Vetores: `ex_vetor[x]`
2. Matrizes e data frames: `ex_tabular[i, j]`
3. Arrays: `ex_array[i, j, k]`
4. Listas: `ex_lista[[x]]`

Acessando dados nas estruturas ii

Importante lembrar que todo os índices em **R** começam com **1**.

```
df_distribuicoes[2:4, ]
```

##	valor	exp_normal	exp_t	exp_uniforme
## 2	2	0.8637929	-0.9608806	0.4816912
## 3	3	0.3165841	1.1527830	0.6070512
## 4	4	0.1073252	-1.0701648	0.7178697

Acessando dados nas estruturas iii

Caso existam etiquetas de identificação nas estruturas, é possível fazer o acesso de duas formas:

1. Usando o acesso de índice, que permite o uso de múltiplos identificadores:

```
df_distribuicoes[18:21, c("exp_t", "exp_uniforme")]
```

##		exp_t	exp_uniforme
##	18	-1.7267311	0.5432894
##	19	1.1139583	0.2261194
##	20	-0.6152256	0.1967566
##	21	2.2479676	0.3397436

2. Usando o operador \$, que só permite o acesso a apenas uma coluna por vez:

```
df_distribuicoes$exp_normal[1:2]
```

```
## [1] -0.9352745  0.8637929
```


R é uma linguagem orientada a objetos e funcional [1, 11].

Orientada a objetos:

- Tudo no **R** é um objeto, inclusive as estruturas atômicas.
- Todo objeto possui alguma propriedade (e.g. tipo dos dados, dimensões).
- Objetos de uma mesma classe possuem as mesmas propriedades.
- Propriedades podem ser herdadas.
- Para interagir com objetos, são usados métodos (funções) apropriados para uma dada classe.

```
typeof(3L)
## [1] "integer"
dim(airquality)
## [1] 153    6
lista <- list(3L, 1:5, datasets::AirPassengers)
head(lista[[3]])
## [1] 112 118 132 129 121 135
```

Funcional:

1. Tudo que acontece no R é uma chamada de função.
2. O resultado de uma função deve depender apenas dos parâmetros de entrada (sem estado).
3. Uso de funções não devem alterar o estado (sem efeitos colaterais).

No entanto, para o ponto 3, algumas funções vão apresentar comportamentos impuros (leitura de arquivos, criação de documentos, print no console, alterar seus inputs).

```
du <- runif(100)
length(du)
## [1] 100
sum(du)
## [1] 47.62906
mean(du)
## [1] 0.4762906
head(du, 5)
## [1] 0.8297221 0.2675022 0.5417281 0.3495079 0.4542729
```

Vetorização (Broadcasting) i

Para certas estruturas, é possível realizar operações em todos os seus elementos sem usar estruturas de repetição [15].

```
# Criando um vetor de números aleatórios  
randn <- rnorm(4)  
randn  
## [1] -0.3459425 -0.3120913  1.2465189  1.0870727  
  
# Somando 5 a todos  
5 + randn  
## [1] 4.654057 4.687909 6.246519 6.087073
```

Vetorização (Broadcasting) ii

```
# Criando um vetor e um data frame de números aleatórios
vec_y <- 3 * rbeta(4, 0.5, 0.2)
df_randn <- data.frame(x = rnorm(4), y = rchisq(4, 10))

# Somando o vetor com o data frame
vec_y + df_randn
##           x           y
## 1 2.7059127 12.353276
## 2 0.2301125  9.073518
## 3 3.5097854 11.531722
## 4 1.9790811 19.225795
```

Podemos construir funções no **R** com a palavra-chave `function(args)`, colocando entre os parênteses os argumentos de entrada.

Declarando funções ii

```
fsoma <- function(x, y, z = 2) {  
  h <- 3*x + 2*y - z  
  return(h)  
}
```

```
fsoma(3, 5)  
## [1] 17  
fsoma(4, 15, 6)  
## [1] 36
```


Declarando funções iii

Em versões mais recentes do R, é possível usar a notação `\(args)`.

```
fsoma <- \(x, y, z = 2) {  
  h <- 3*x + 2*y - z  
  return(h)  
}
```

```
fsoma(3, 5)  
## [1] 17  
fsoma(4, 15, 6)  
## [1] 36
```

Boas práticas

- Código é qualquer coisa escrita na linguagem.
- Scripts são arquivos que terminam com `.R` e podem ser executados sozinhos, ou agregados para a criação de bibliotecas ou rotinas mais complexas.
- Notebook: forma de executar código através de células (Jupyter Notebook).
- RMarkdown: Um tipo de script, com extensão `.Rmd`, projetado para gerar documentos usando `pandoc`.

Sempre que possível, **evite notebooks**.

Ao organizar e escrever códigos para análise, considere os seguintes elementos [9, 13]:

- Nomes significantes para variáveis, funções, e arquivos.
- Espaçamento do código.
- Comentários.

Estes fatores vão auxiliar a leitura do código tanto por terceiros quanto por si próprio em um outro momento.

Há outros elementos que podem (e devem ser considerados), mas estes podem ser consultados em guias de estilo (*style guides*):

- Guia de estilo, Bioconductor: <https://contributions.bioconductor.org>
- Guia de estilo, Tidyverse: <https://style.tidyverse.org>
- Guia de estilo, JefWorks: <http://jef.works/R-style-guide>

Boas práticas: Exemplos i

Adequado

```
# DataFrame ilustrando 3 distribuições  
df_distribuicoes <- data.frame(valor = 1:50,  
                                exp_normal = rnorm(50),  
                                exp_t = rt(50, 10),  
                                exp_uniforme = runif(50))
```

Inadequado

```
dists <- data.frame(1:50, rnorm(50), rt(50, 10), runif(50))
```

Boas práticas: Exemplos ii

Adequado

Cálculo de indicador a partir de parâmetros x, y, z, k.

```
calcularIndicador <- function(x, y, z, k = 3) {  
  if (x > y) {  
    return(x * z - y)  
  } else {  
    return(k * z * y - x)  
  }  
}
```

Inadequado

```
computoNumerico <- function(x,y,z,k=3) {  
  if (x > y) x * z - y else k * z * y - x  
}
```

Boas práticas: Estrutura de pastas

Na hora de realizar uma análise, é uma boa prática criar uma estrutura de projeto para organizar os arquivos necessário. A organização auxilia na reprodutibilidade, e permite que o código seja distribuído como um pacote.

Nome do Projeto

— R/	Código escrito em R
— data/	Dados a serem usados (leitura apenas)
— docs/	Documentação/artigo gerado sobre as análises
— img/	Figuras geradas pelo código
— output/	Saídas geradas pelo código
— src/	Códigos em C/C++
— main.R	Arquivo que executa a análise

Considerações sobre dependências

- Pensar em dependências como partes móveis.
- Quanto mais dependências, maior o risco de colisões.
- Sempre ponderar a relação facilidade–fragilidade.
- Site sobre o assunto: The Tinyverse: <https://www.tinyverse.org>

Considerações sobre dependências

```
tools::package_dependencies(package="data.table",  
                             recursive=TRUE)$data.table
```

```
## [1] "methods"
```

```
tools::package_dependencies(package="dplyr",  
                             recursive=TRUE)$dplyr
```

```
## [1] "generics" "glue" "lifecycle" "magrittr" "methods"  
## [6] "R6" "rlang" "tibble" "tidyselect" "utils"  
## [11] "vctrs" "pillar" "cli" "crayon" "ellipsis"  
## [16] "fansi" "utf8" "pkgconfig" "purrr" "grDevices"
```

Análise de dados: Conceitos

Antes de iniciar qualquer trabalho de fato, é importante se perguntar qual o objetivo de uma análise.

Em um contexto de negócios, alguém em algum momento irá precisar da informação para a tomada de decisão, o que pode acontecer uma única vez, ou de maneira recorrente.

Se não há familiaridade com os dados, é preciso primeiro entendê-los num processo chamado EDA (*Exploratory Data Analysis*, análise exploratória de dados).

Esse processo normalmente não é rígido, e também é iterativo: a ideia é fazer perguntas sobre o conjunto de dados, realizar transformações e visualizações, e refinar o entendimento sobre o mesmo (gerando mais perguntas e repetindo o ciclo) [15].

Algumas atividades de rotina em EDAs incluem a geração de estatísticas descritivas, criação de gráficos, transformações e agrupamentos [17].

Há outros aspectos para a realização de análises de dados, como o processo da coleta de dados em si, e a forma de entrega dos resultados (caso necessário).

É possível se valer de algumas metodologias para estruturar o processo, especialmente se em algum momento, as informações serão consumidas para tomadas de decisão.

Neste curso, vamos seguir o modelo proposto por Wickham e Grolemund (2017), com foco nas etapas de Transformação e Visualização (em azul na imagem):

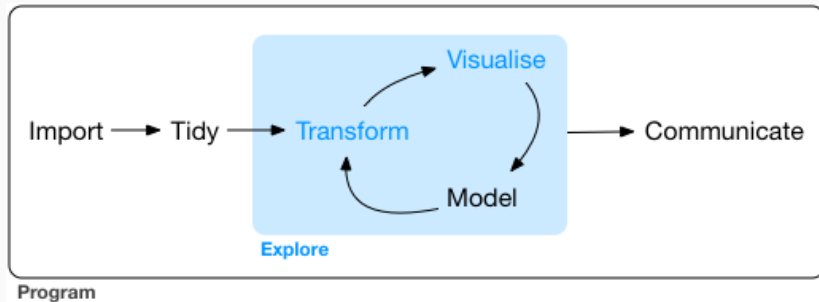


Figura 1: Fluxo para análise de dados (Fonte: Wickham e Grolemund, 2017)

Enquanto dados podem ser representados de diversas formas, há uma forma de organizar os dados que facilita o trabalho de análises, chamado de *tidy data* (em português: dados organizados) [14, 15].

Para manter os dados organizados, temos que ter:

- Cada variável é representada por uma coluna.
- Cada observação é representada por uma linha.
- Cada valor fica em uma única célula.

Esse formato de dados é baseado na 3a forma normal de Codd [14].

Estrutura dos dados

Em situações de negócio, os dados dificilmente estarão em uma mesma tabela.

Uma forma de modelagem muito comum para armazenamento de dados é o **Esquema em Estrela** (*Star Scheme* no inglês), que emprega dois tipos de tabela, Fato e Dimensão [5].

- **Tabelas de dimensões** representam os itens a serem modelados (entidades de negócio). Costumam ter um número menor de linhas.
- **Tabelas de fatos** registram observações (temperatura em um dado horário) ou outros eventos (quando um cliente realiza uma compra). Podem apresentar uma grande quantidade de linhas, e podem continuar a crescer enquanto estiverem em uso.

Ambas as tabelas se valem de identificadores únicos (**chaves**) para poderem ser relacionadas.

Vamos empregar uma versão de dados de vendas do site Bandcamp, disponibilizada na plataforma Observable [7].

O arquivo original foi convertido de uma base em formato `.parquet` para um arquivo `.csv`.

Análise de dados: Manipulação de Tabelas

Extensão do `data.frame` do **R**, projetado para ser rápido e lidar com grandes quantidades de arquivos em memória. Trabalha com alterações por referência. Apresenta uma sintaxe flexível para facilitar o desenvolvimento [3].

`{data.table}`: Sintaxe

A sintaxe para empregar os recursos do `data.table` tem o seguinte formato:

$$DT[i, j, by]$$

Onde:

- `i`: representa um índice de linhas ou algum filtro.
- `j`: representa um índice para colunas, ou nomes, ou funções.
- `by`: indica como devemos agrupar os dados.

Toda operação nesse formato retorna uma `data.table`, permitindo que seja possível realizar várias em sequência: `DT[...] [...] [...]`

`{data.table}`: Filtrando linhas

Para gerar subconjuntos de dados, usamos o primeiro campo `i`:

```
DT[col_1 == valor_1, ]
```

Também é possível usar os operadores booleanos do **R**:

```
DT[col_1 == valor_1 & (col_2 >= valor_2 | col_3 != valor_3), ]
```

{data.table}: Selecionando colunas

Há algumas peculiaridades para selecionar colunas na biblioteca.

Usar o nome da coluna, sem aspas, gera um vetor

```
DT[, col_name]
```

Usando `list()` ou `.(())`, gera uma `data.table`

```
DT[, .(col_name)]
```

Para selecionar mais de uma coluna, é possível usar `list()` ou `.(())`

```
DT[, .(col_1, col_2)]
```

```
DT[, list(col_1, col_2)]
```

*# Para selecionar e **renomear** as colunas*

```
DT[, .(nome_1 = col_1, nome_2 = col_2)]
```

{data.table}: Selecionando colunas

Há uma outra forma de selecionar colunas, usando texto.

```
# Colunas de interesse
```

```
colunas = c('col_1', 'col_2')
```

```
# Para evitar a identificação automática
```

```
DT[, ..colunas]
```

```
DT[, colunas, with = FALSE]
```

```
# Selecionando colunas **exceto** as colunas declaradas no vetor
```

```
DT[, !..colunas]
```

```
DT[, !colunas, with = FALSE]
```


A biblioteca conta com algumas funções especiais para facilitar algumas tarefas numéricas, estes são:

- `.N`: Conta o número de elementos em um subconjunto de dados.
- `.SD`: Representa um subconjunto de dados.
- `.SDcols`: Lista contendo as colunas que vão gerar um subconjunto de dados. Precisa ser declarado como texto, usando `c()`.

{data.table}: Passando por referência

Uma grande vantagem da biblioteca é poder realizar operações dentro da própria tabela usando o operador :=.

```
# Colunas := Valores
```

```
DT[ , c('col_1', 'col_2') := list(valor_1, valor_2)]
```

```
DT[ , c('col_1', 'col_2') := .(valor_1, valor_2)]
```

```
# Declaração funcional
```

```
DT[, `:=`(col_1 = valor_1, col_2 = valor_2)]
```

É preciso ter cuidado ao usar essa formulação, uma vez que ela altera por referência a tabela.

A biblioteca contém duas funções para realizar agregações em conjuntos, de maneira similar ao que é oferecida no SQL:

- `rollup()`: Cria uma tabela com os totais para combinações de argumentos, mas não gera todas as combinações.
- `cube()`: Cria uma tabela com os totais para **todas** as combinações de argumentos.

`{data.table}`: Pivoteamento de Tabelas

Uma funcionalidade presente no Excel e também disponível na biblioteca `{data.table}` é o pivoteamento de tabelas com a função `dcast()`

```
dcast(dt, linhas ~ colunas, value.var = 'coluna')
```

Também é possível declarar uma função para agregar os dados, usando o argumento `fun.aggregate`.

Para reverter o processo, é possível usar a função `melt()`.

Análise de dados: Visualização

Enquanto o **R** possui três sistemas gráficos [6], o foco será na biblioteca `{ggplot2}`, desenvolvida por Wickham [12].

O funcionamento da biblioteca é orientado pela Gramática de Gráficos (em inglês, *Grammar of Graphics*): um gráfico é construído por um conjunto de dados e um conjunto de mapeamentos, camadas, escalas, coordenadas, facetas, e temas.

Enquanto a gramática serve para orientar como construir bons gráficos, ela não informa qual a melhor escolha de gráfico para um conjunto de dados [12].

{ggplot2}: Funcionamento básico

```
ggplot(df, aes(x, y, ... [, colour, shape, size])) +  
  geom_* +      # funções que definem o tipo de gráfico  
  scale_* +     # funções que definem a escala para cada eixo  
  facet_* +     # funções para ajustar facetas  
  theme_*       # funções para customização dos temas
```

{ggplot2}: Tipos de Gráficos

No {ggplot2}, todo tipo de gráfico começa com o texto `geom_*` (de *geometry*), seguido do identificador.

Gráfico	Função
Linha	<code>geom_line()</code>
Pontos (scatter)	<code>geom_point()</code>
Barra	<code>geom_bar()</code> ou <code>geom_col()</code>
Histograma	<code>geom_histogram()</code>
Boxplot	<code>geom_boxplot()</code>
Densidade	<code>geom_density()</code>

{ggplot2}: Título, Rótulos, Cor das legendas

Há funções para ajustar o texto do título, subtítulo, e os rótulos dos eixos e das legendas.

Elemento	Função
Título	<code>labs(title = 'título'), ggtitle('título')</code>
Sub-título	<code>labs(subtitle = 'subtítulo')</code>
Título (alt.)	<code>ggtitle('título')</code>
Sub-título (alt.)	<code>ggtitle('título', subtitle = 'subtítulo')</code>
Legenda	<code>labs(caption='legenda')</code>
Eixo X	<code>labs(x = 'eixo-x'), xlab('eixo-x')</code>
Eixo Y	<code>labs(y = 'eixo-y'), ylab('eixo-y')</code>

Para colorir de maneira adequada tanto os elementos do gráfico, quanto a legenda, podemos usar as famílias de funções `scale_fill_*` e `scale_colour_*`.

Salvando documentos

Usando a biblioteca `{openxlsx}`, podemos salvar várias tabelas em um único arquivo usando listas nomeadas.

A sintaxe para guardar os arquivos é: `write.xlsx(lista_tabelas, arquivo_saida)`

Para salvar imagens, há duas opções:

- Usar a função `ggsave()`;
- Usar uma combinação de funções `png()/jpeg()/pdf()` e `dev.off()` [Mais trabalhosa].

Em ambos os casos, não é possível salvar em massa como na função `openxlsx::write.xlsx()`, mas é possível usar uma estrutura de repetição com funções.

Criação de Relatórios

Para organizar um relatório de negócios, podemos usar a seguinte estrutura [16].

- Resumo Executivo: resumo do assunto, método de análise, descobertas/vereditos, recomendações.
- Sumário
- Introdução
- Corpo
- Conclusão
- Lista de referências
- Apêndice(s)

Além do Word, podemos usar outras ferramentas:

- \LaTeX : sistema de composição tipográfica criado por Donald Knuth.
- Overleaf: site gratuito que possibilita o uso de \LaTeX .
- RMarkdown: permite combinar código em **R**/Python com Markdown para a geração de documentos em diversos formatos.
- Sweave: funcionalidade do **R** que emprega \LaTeX para a geração de relatórios.

- [1] John M Chambers. *Extending R*. CRC Press, 2016, p. 357.
- [2] CRAN. *What is R?* 2022. URL: <https://www.r-project.org/about.html>.
- [3] Matt Dowle e Arun Srinivasan. *data.table: Extension of 'data.frame'*. R package version 1.14.2. 2021. URL: <https://CRAN.R-project.org/package=data.table>.
- [4] Garrett Grolemond e Hadley Wickham. “Dates and Times Made Easy with lubridate”. Em: *Journal of Statistical Software* 40.3 (2011), pp. 1–25. URL: <https://www.jstatsoft.org/v40/i03/>.

Referências ii

- [5] Microsoft. *Understand star schema and the importance for Power BI*. Publicado em 06 abr 2022. 2022. URL: <https://docs.microsoft.com/en-us/power-bi/guidance/star-schema>.
- [6] Paul Murrell. *R Graphics*. 3^a ed. CRC Press, 2019, p. 423.
- [7] Observable. *Bandcamp Sales Data*. Publicado em 14 abr 2022 por Ian Johnson. 2022. URL: <https://observablehq.com/@observablehq/bandcamp-sales-data>.
- [8] RStudio. *RStudio*. 2022. URL: <https://www.rstudio.com/products/rstudio/>.
- [9] Kevin Rue-Albrecht et al. *Bioconductor Packages: Development, Maintenance, and Peer Review*. Última compilação em 12 mai 2022. 2022. URL: <https://contributions.bioconductor.org>.

- [10] Philipp Schaubberger e Alexander Walker. *openxlsx: Read, Write and Edit xlsx Files*. R package version 4.2.5. 2021. URL:
<https://CRAN.R-project.org/package=openxlsx>.
- [11] Hadley Wickham. *Advanced R*. 2ª ed. CRC Press, 2019, p. 587.
- [12] Hadley Wickham. *ggplot2. Elegant Graphics for Data Analysis*. 2ª ed. Springer, 2016, p. 260. DOI: 10.1007/978-3-319-24277-4.
- [13] Hadley Wickham. *The tidyverse style guide*. 2021. URL:
<https://style.tidyverse.org>.
- [14] Hadley Wickham. “Tidy Data”. Em: *Journal of Statistical Software* 59.10 (2014), pp. 1–23.

- [15] Hadley Wickham e Garrett Grolemond. *R for Data Science*. O'Reilly, 2017, p. 492.
- [16] University of Wollongong. *Report Writing: Business*. 2022, p. 4. URL: <https://www.uow.edu.au/student/support-services/learning-development/resources/>.
- [17] Donghui Yan e Gary E Davis. “A first course in data science”. Em: *Journal of Statistics Education* 27.2 (2019), pp. 99–109.