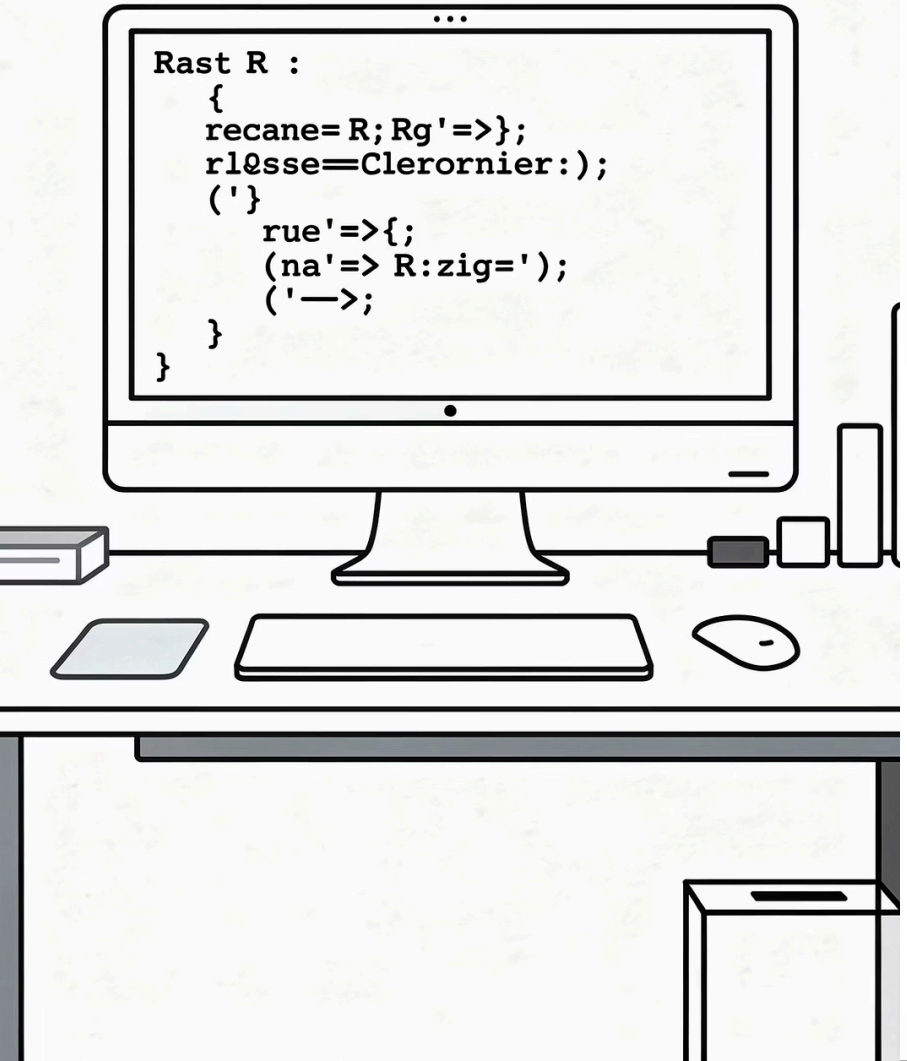


CEFET/RJ

Manipulação de Data Frames em R

Aprenda a criar e manipular baralhos de cartas usando data frames, uma das estruturas mais importantes da linguagem R

Eduardo Ogasawara

eduardo.ogasawara@cefet-rj.br

https://eic.cefet-rj.br/~eogasawara

CRIANDO ESTRUTURAS

Construindo um Baralho com expand.grid

A função expand.grid() é perfeita para criar todas as combinações possíveis entre faces e naipes. Com apenas três linhas de código, geramos um baralho completo de 52 cartas.

Este método garante que cada face apareça com cada naipe, criando a estrutura fundamental do nosso data frame.

```
faces <- c("ás", "dois", "três",
 "quatro","cinco", "seis", "sete",
 "oito","nove", "dez", "valete",
 "dama", "rei")

naipes = c("ouros", "copas",
 "paus", "espadas")

baralho <- expand.grid(
 face=faces, naipe=naipes)
```

ADICIONANDO DADOS

Criando e Atribuindo uma Nova Coluna

Sintaxe do Cifrão

Use `baralho$valor` para criar ou acessar colunas pelo nome de forma direta e intuitiva

Valores Repetidos

O vetor `1:13` repetido 4 vezes atribui valores para cada naipe de forma sequencial

Função head()

Visualize as primeiras linhas do data frame para confirmar que a coluna foi adicionada corretamente

```
baralho$valor <- c(1:13, 1:13,
  1:13, 1:13)

head(baralho)
```

```
## face naipe valor
## 1 ás ouros 1
## 2 dois ouros 2
## 3 três ouros 3
## 4 quatro ouros 4
## 5 cinco ouros 5
## 6 seis ouros 6
```

Formas de Acesso a Tabelas

Em R, data frames e matrizes usam a notação `tabela[linha, coluna]` para acessar dados. Existem diferentes métodos de indexação que oferecem flexibilidade na manipulação.

01

Inteiros Positivos

Selecionam elementos pela posição: `baralho[1, 2]`

02

Inteiros Negativos

Excluem elementos: `baralho[-1, ]`

03

Em Branco

Selecionam tudo: `baralho[, ]`

04

Valores Lógicos

Filtram por condições: `baralho[baralho$valor > 10, ]`

05

Nomes

Acessam por nome da coluna: `baralho[, "face"]`

06

Atributo $

Acesso direto à coluna: baralho$face

VETORES

Acessando Dados de uma Coluna

Quando acessamos uma única coluna de um data frame, o resultado é um **vetor**. Existem várias formas equivalentes de fazer isso.

Por Posição

```
baralho[1, 1]
## [1] ás
```

Acessa a primeira linha e primeira coluna

Por Nome

```
baralho[1, "face"]
## [1] ás
```

Usa o nome da coluna para maior clareza

Múltiplas Linhas

```
baralho[c(1,2), 1]
## [1] ás dois
```

Retorna vetor com múltiplos elementos

Coluna Inteira

```
baralho[ , 1]
```

Índice vazio seleciona todas as linhas

Múltiplas Formas de Acessar Colunas

O R oferece flexibilidade na sintaxe. Estas três formas produzem resultados idênticos ao acessar dados específicos de uma coluna.

Indexação Dupla

```
baralho[c(11,14), "face"]
## [1] valete ás
```

Especifica linhas e nome da coluna entre colchetes

Indexação por Posição

```
baralho[c(11,14), 1]
## [1] valete ás
```

Usa o número da coluna em vez do nome

Atributo $ + Vetor

```
baralho$face[c(11,14)]
## [1] valete ás
```

Primeiro acessa a coluna, depois filtra as linhas desejadas

Dica: A sintaxe com $ é mais legível quando você sabe exatamente qual coluna precisa acessar.

DATA FRAMES

Acessando Múltiplas Colunas

Quando selecionamos duas ou mais colunas, o resultado mantém a estrutura de **data frame** (tabela). Isso preserva as relações entre os dados.

Linha e Intervalo

```
baralho[11, 1:2]

##      face naipe
## 11 valete ouros
```

Usa `:` para selecionar colunas consecutivas

Linhas e Intervalo

```
baralho[c(11,14), 1:2]

##      face naipe
## 11 valete ouros
## 14     ás copas
```

Combina múltiplas linhas com intervalo de colunas

Por Nomes

```
baralho[c(11,14),
  c("face", "naipe")]

##      face naipe
## 11 valete ouros
## 14     ás copas
```

Usa vetor de nomes para maior clareza

Controle de Redução com drop=FALSE

Por padrão, quando selecionamos uma única coluna, o R reduz o resultado para um vetor. O parâmetro `drop=FALSE` mantém a estrutura de data frame.

Com Nome da Coluna

```
baralho[c(11,14), "face",
drop=FALSE]

##      face
## 11 valete
## 14     ás
```

Mantém estrutura de tabela mesmo com uma só coluna

Com Índice Numérico

```
baralho[c(11,14), 1,
drop=FALSE]

##      face
## 11 valete
## 14     ás
```

Funciona igualmente bem com índices numéricos

Importante: Use `drop=FALSE` quando precisar garantir que o resultado seja sempre um data frame, independente do número de colunas selecionadas.

⊖ EXCLUSÃO

Índices Negativos para Exclusão

Índices negativos são uma forma elegante de **excluir** elementos sem precisar especificar tudo que você quer manter. Muito útil para remover colunas ou linhas indesejadas.

Mantendo as Primeiras 13 Linhas

```
baralho[c(1:13), 1]
## [1] ás dois três quatro
cinco seis...
```

Seleção positiva: especifica o que incluir

Excluindo as Últimas 39 Linhas

```
baralho[-c(14:52), 1]
## [1] ás dois três quatro
cinco seis...
```

Seleção negativa: especifica o que remover (mesmo resultado!)

Excluindo a Primeira Coluna

```
baralho[1:3, -1]
##   naipe valor
## 1 ouros     1
## 2 ouros     2
## 3 ouros     3
```

Remove apenas a coluna de faces, mantendo naipe e valor

O Poder do Índice Vazio

Deixar um índice vazio significa "selecione tudo". É uma forma concisa e idiomática de trabalhar com dimensões completas.

Tabela Completa

```
baralho[ , ]

##    face naipe valor
## 1    ás ouros     1
## 2  dois ouros     2
## 3  três ouros     3
```

Uma Linha Inteira

```
baralho[1, ]

##  face naipe valor
## 1 ás ouros     1
```

Todas as colunas da primeira linha

Uma Coluna Inteira

```
baralho[ , 1]

## [1] ás dois três
## quatro cinco...
```

Todas as linhas da primeira coluna

- `baralho[ , ]` retorna todo o data frame
- `baralho[1, ]` retorna a primeira linha como data frame de uma linha
- `baralho[ , 1]` retorna a primeira coluna como vetor

Indexação Lógica para Filtrar Colunas

Vetores lógicos (`TRUE`/`FALSE`) permitem selecionar colunas de forma programática. Cada posição do vetor corresponde a uma coluna.

```
baralho[1, c(TRUE, TRUE, FALSE)]

##   face naipe
## 1   ás ouros
```

O vetor lógico `c(TRUE, TRUE, FALSE)` seleciona as duas primeiras colunas (*face* e *naipe*) e exclui a terceira (*valor*).

Esta técnica é especialmente útil quando você precisa selecionar colunas com base em condições computadas dinamicamente.

FILTRAGEM

Indexação Lógica para Filtrar Linhas

A indexação lógica brilha quando precisamos filtrar linhas baseadas em condições. Criamos um vetor de valores TRUE/FALSE que determina quais linhas incluir.

Criar o Filtro

```
filtro <- baralho$valor < 3
```

Gera vetor lógico: TRUE onde valor é menor que 3

Aplicar o Filtro

```
baralho[filtro, ]
```

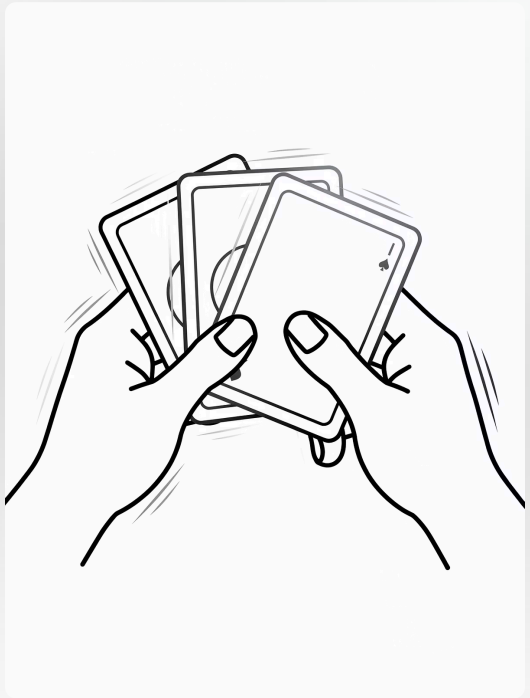
```
##    face naipe valor
## 1    ás ouros     1
## 2  dois ouros     2
## 14   ás copas     1
```

Retorna apenas linhas onde filtro é TRUE

Forma Direta

```
baralho[baralho$valor < 3, ]
```

Combina criação e aplicação em uma linha



Desafio: Embaralhando o Baralho

Para embaralhar, criamos uma ordem aleatória e usamos como índice. A função `sample()` gera uma permutação aleatória dos números de 1 até o total de linhas.

Gerando Ordem Aleatória

```
ordem <- sample(1:nrow(baralho))
ordem

## [1] 2 40 11 46 35 48 41...
```

Cria vetor com números de 1 a 52 em ordem aleatória

Reordenando o Baralho

```
cartas <- baralho[ordem, ]
cartas

##      face   naipe valor
## 2    dois   ouros     2
## 40    ás espadas     1
## 11 valete   ouros    11
```

Usa o vetor ordem para reorganizar as linhas

Conceito-chave: A função `sample()` é perfeita para randomização. `nrow()` retorna o número de linhas, tornando o código flexível para qualquer tamanho de data frame.

REUTILIZAÇÃO

Encapsulando como Função

Transformar código útil em funções facilita a reutilização e manutenção. Nossa função `embaralhar()` aceita qualquer data frame e retorna uma versão embaralhada.

```
embaralhar <- function(baralho) {
  ordem <- sample(1:nrow(baralho))
  return(baralho[ordem, ])
}

cartas <- embaralhar(baralho)
cartas
```

```
##      face   naipe valor
## 16   três   copas     3
## 49    dez espadas    10
## 37 valete    paus    11
## ...
```

Agora podemos embaralhar com uma única chamada de função!

Reutilizável

Use quantas vezes quiser sem reescrever código

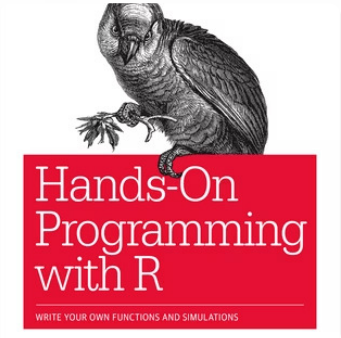
Código Limpo

Esconde detalhes de implementação, foca na intenção

Flexível

Funciona com qualquer data frame, não só baralhos

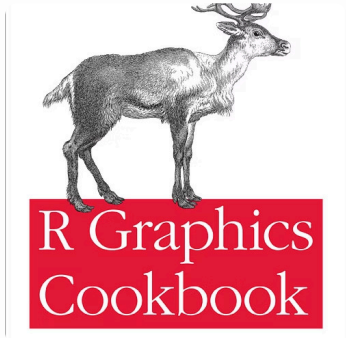
Referências



Hands-on Programming

Aprenda R criando suas próprias funções e simulações

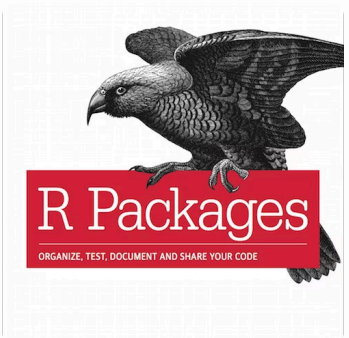
https://rstudio-education.github.io/hopr/index.html



R Graphics Cookbook

Domine visualizações de dados em R

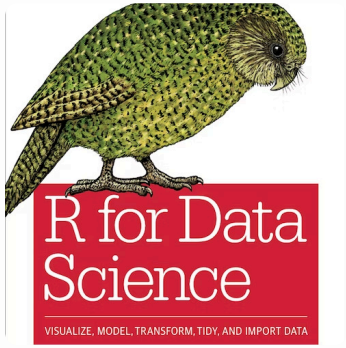
https://r-graphics.org



R Packages

Desenvolva seus próprios pacotes R

https://r-pkgs.org/index.html



R for Data Science

Guia completo para ciência de dados

https://r4ds.had.co.nz

G. Grolemund, 2014, *Hands-On Programming with R: Write Your Own Functions and Simulations*. O'Reilly Media, Inc.