

Fundamentos da Linguagem R

Uma jornada completa pelos conceitos fundamentais da linguagem R, explorando desde a instalação e configuração até as estruturas de dados mais importantes para análise estatística e ciência de dados.

Eduardo Ogasawara

eduardo.ogasawara@cefet-rj.br
<https://eic.cefet-rj.br/~eogasawara>

Introdução ao R



Sobre a Linguagem

R é uma linguagem de programação e ambiente de software livre voltado para computação estatística, mantido pela R Foundation for Statistical Computing.

História e Evolução

Criado por Ross Ihaka e Robert Gentleman na Universidade de Auckland, Nova Zelândia, R foi derivado da linguagem S desenvolvida nos Bell Laboratories da AT&T. Atualmente na versão 4.4, R é amplamente utilizado por estatísticos, mineradores de dados e cientistas de dados em todo o mundo.

Comunidade Ativa: R possui uma das comunidades mais vibrantes no mundo da programação estatística, com milhares de pesquisadores, professores e programadores contribuindo constantemente.

<https://www.r-project.org/>

Console do R

O console do R oferece uma interface interativa onde você pode executar comandos diretamente e ver os resultados imediatamente. É a base para todo desenvolvimento em R, disponível gratuitamente para os principais sistemas operacionais.

Windows

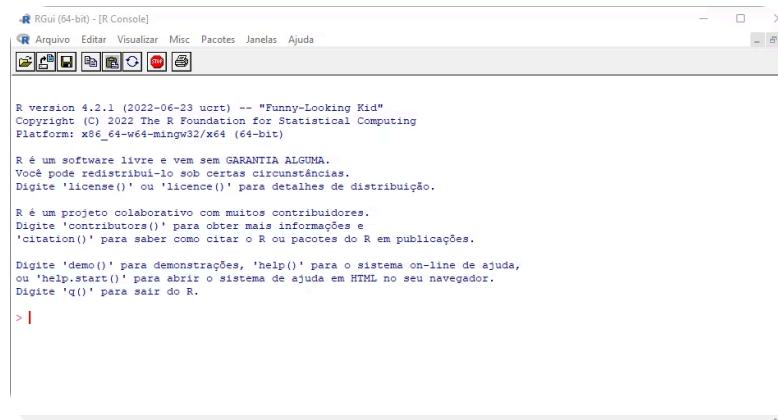
Instalação completa com interface gráfica nativa para o sistema operacional Windows.

Mac

Versão otimizada para macOS, aproveitando os recursos do sistema Apple.

Linux

Distribuições disponíveis para diversas versões do Linux, integradas aos gerenciadores de pacotes.



RStudio: Ambiente de Desenvolvimento Integrado

O RStudio é uma IDE (Integrated Development Environment) poderosa que transforma a experiência de programação em R. Com vantagens significativas sobre o console básico, o RStudio oferece visualização avançada de dados, ferramentas de depuração sofisticadas e recursos de criação de perfil de código.



Editor Avançado

Destaque de sintaxe, autocompletar e navegação inteligente pelo código.



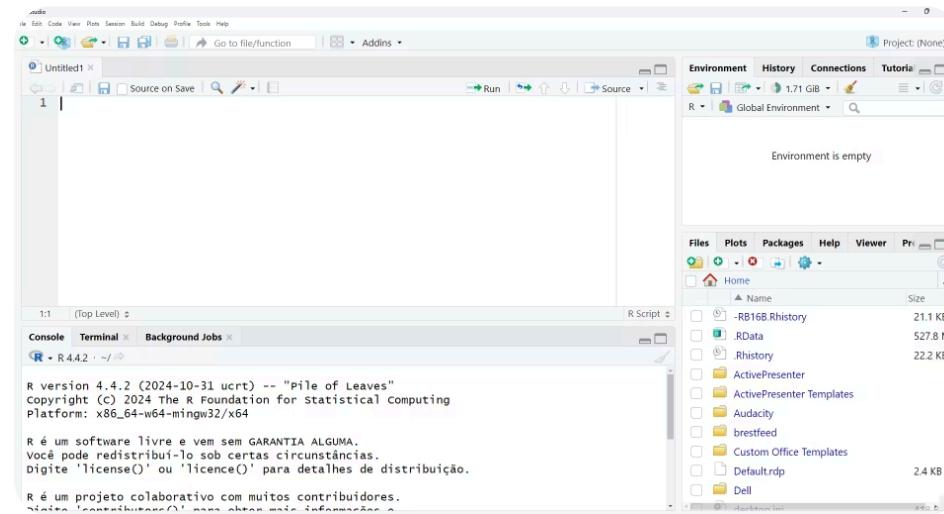
Visualização Integrada

Gráficos e visualizações aparecem diretamente na interface, facilitando a análise.



Gerenciamento de Ambiente

Visualize variáveis, histórico e pacotes instalados em painéis dedicados.



Pacotes do R: Expandindo Funcionalidades

O Que São Pacotes?

Pacotes são coleções de funções disponibilizadas como bibliotecas, expandindo enormemente as capacidades do R base. Atualmente, existem mais de 22.000 pacotes disponíveis no repositório CRAN.

Repositório CRAN

- Controle rigoroso de qualidade
- Documentação completa e exemplos
- Garantia de compatibilidade
- Atualizações regulares

<https://cran.r-project.org/>

Repositórios GitHub

Versões em desenvolvimento podem ser instaladas diretamente do GitHub, permitindo acesso antecipado a novos recursos e correções.

Comunidade Ativa

Milhares de pesquisadores, professores, programadores e estatísticos contribuem constantemente com novos pacotes e melhorias.



Pacotes DAL

O laboratório DAL desenvolveu 7 pacotes especializados:

- [daltoolbox](#)
- [harbinger](#)
- [tspredit](#)

Instalação e Carregamento de Pacotes

01

Verificação e Instalação

Antes de instalar, verifique se o pacote já existe no sistema.

```
if(!require(daltoolbox)) {  
  install.packages("daltoolbox")  
}
```

02

Carregamento do Pacote

Após a instalação, carregue o pacote para uso na sessão atual.

```
library(daltoolbox)
```

03

Boas Práticas

Evite carregar pacotes desnecessários para manter o ambiente limpo e reduzir conflitos entre funções.

- **Importante:** A função `require()` retorna TRUE se o pacote foi carregado com sucesso e FALSE caso contrário, permitindo verificação condicional.



Definição de Variáveis e Tipos de Dados

Criando Variáveis

Em R, variáveis são criadas através de atribuição usando os operadores `<-` ou `=`. O operador `<-` é preferido por convenção na comunidade R.

```
weight <- 60  
height = 1.75  
subject <- "A"  
healthy <- TRUE
```

Avaliando Variáveis

Simplesmente digite o nome da variável para ver seu valor:

```
weight  
## [1] 60
```

Tipos de Dados Fundamentais

- **Numérico**

Números inteiros e decimais (60, 1.75)

- **Caractere (String)**

Sequências de caracteres entre aspas simples ou duplas

- **Lógico (Boolean)**

Valores TRUE ou FALSE para operações lógicas

- **Strings:** Uma string é uma sequência de caracteres representada internamente entre aspas duplas. Pode ser declarada com aspas simples ('Isto é uma string') ou aspas duplas ("Isto também é uma string").

Conversão Entre Tipos de Dados

R fornece uma família completa de funções para converter entre diferentes tipos de dados. Essas conversões são essenciais para garantir que os dados estejam no formato correto para operações específicas.

Conversão para Inteiro

```
weight <- as.integer(weight)  
is.integer(weight)  
## [1] TRUE
```

A função `as.integer()` converte valores numéricos para o tipo inteiro, removendo casas decimais.

Outras Conversões Úteis

- `as.numeric()` - converte para número decimal
- `as.character()` - converte para string
- `as.logical()` - converte para booleano
- `as.factor()` - converte para fator categórico

 **Dica Profissional:** Digite "as." no RStudio e pressione a tecla Tab para ver todas as funções de conversão disponíveis. As funções correspondentes `is.*` verificam o tipo de uma variável.

Vetores: Estrutura de Dados Fundamental

Definindo Vetores

Vetores são sequências ordenadas de elementos do mesmo tipo. A função `c()` (combine) é usada para criar vetores a partir de valores estáticos.

```
weight <- c(60, 72, 57, 90, 95, 72)
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
subject <- c("A", "B", "C", "D", "E", "F")
```

Avaliando Vetores

```
weight
## [1] 60 72 57 90 95 72

subject
## [1] "A" "B" "C" "D" "E" "F"
```

Vetores são a estrutura de dados mais básica e importante em R. Todas as operações em R são vetorizadas por padrão, permitindo operações eficientes em conjuntos de dados.

Outras Formas de Criar Vetores

→ Função `rep()`

Cria vetores repetindo valores um número específico de vezes

→ Sequências com :

Cria sequências numéricas, como `1:10`

→ Função `seq()`

Gera sequências com controle de início, fim e incremento

Iteração: Laços For

Laços for permitem repetir operações para cada elemento de um vetor ou sequência. São fundamentais para processar dados elemento por elemento quando operações vetorizadas não são suficientes.

Inicialização

Crie um vetor vazio para armazenar resultados

```
bmi <- 0
```

Processamento

Execute o cálculo para cada índice

```
bmi[i] <- weight[i] / height[i]^2  
}
```

Definição do Laço

Itere de 1 até o comprimento do vetor

```
for (i in 1:length(weight)) {
```

Resultado

Visualize o vetor resultante

```
bmi  
## [1] 19.59 22.22 20.94 24.93 31.38 19.74
```

 **Estrutura do For:** A variável definida (neste caso i) itera sobre todos os elementos do vetor especificado após in.

Iteração: Laços While

Conceito do While

Laços while executam enquanto uma condição for verdadeira. É esperado que as variáveis usadas na condição sejam modificadas dentro do laço para eventualmente tornar a condição falsa e encerrar a iteração.

Implementação

```
bmi <- 0  
i <- 1  
while (i <= length(weight)) {  
  bmi[i] <- weight[i] / height[i]^2  
  i <- i + 1  
}  
  
bmi  
## [1] 19.59 22.22 20.94 24.93 31.38 19.74
```

Inicialização

Configure variáveis antes do laço

Condição

Defina quando o laço deve continuar

Incremento

Modifique variáveis para evitar loops infinitos

- ❑ **Cuidado:** Sempre garanta que a condição do while eventualmente se torne falsa, caso contrário você criará um loop infinito que travará seu programa.

Criando e Usando Funções

1

Definição

Funções encapsulam código reutilizável com parâmetros e retorno.

```
name <- function(parameters) {  
  body  
}
```

2

Implementação

Crie uma função para calcular o IMC (Índice de Massa Corporal).

```
compute_bmi <- function(weight, height) {  
  bmi <- weight / height^2  
  return(bmi)  
}
```

3

Uso com Escalar

Chame a função com valores únicos.

```
bmi <- compute_bmi(60, 1.75)  
bmi  
## [1] 19.59184
```

4

Uso com Vetores

A mesma função processa vetores automaticamente!

```
bmi <- compute_bmi(weight, height)  
bmi  
## [1] 19.59 22.22 20.94 24.93 31.38 19.74
```

Funções são definidas com parâmetros e código atribuído a um nome. Elas devem retornar um valor usando `return()`. A vetorização automática do R permite que a mesma função trabalhe tanto com valores únicos quanto com vetores inteiros.

Gráficos Básicos em R

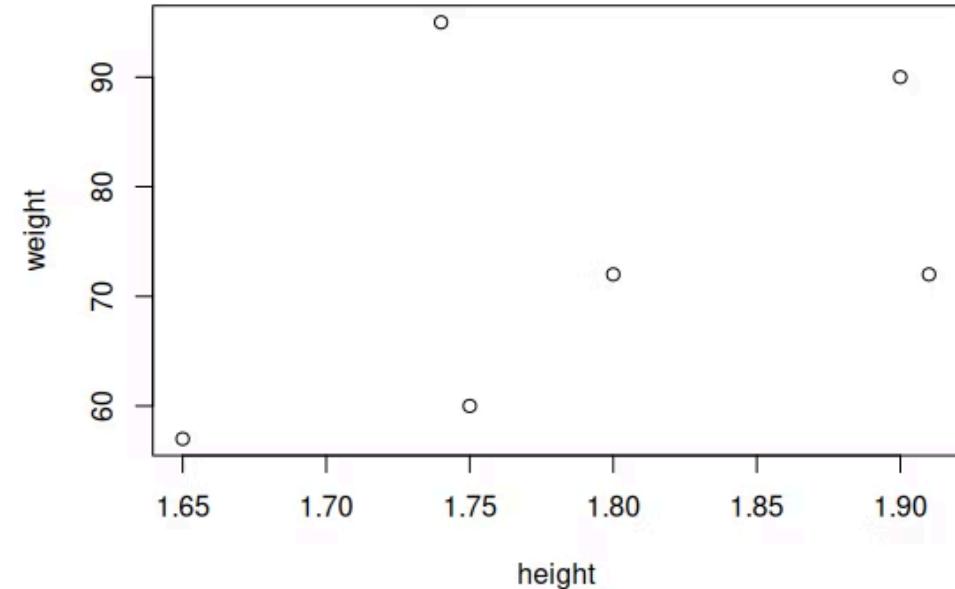
Preparação dos Dados

```
weight <- c(60, 72, 57, 90, 95, 72)  
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)  
bmi <- weight / height^2
```

Criando Gráficos de Dispersão

```
plot(height, weight)
```

A função `plot()` é a função básica de plotagem em R. O primeiro parâmetro representa as observações do eixo x, enquanto o segundo representa as observações do eixo y.



O gráfico de dispersão mostra a relação entre altura e peso, permitindo identificar padrões e correlações visuais nos dados.

Fatores: Dados Categóricos

Fatores são variáveis em R que representam dados categóricos. Eles são fundamentais para análise estatística, modelagem e visualização de dados qualitativos. Fatores são armazenados internamente como um vetor de valores inteiros com um conjunto correspondente de valores de caracteres para exibição.



Criação de Fatores

Tanto variáveis numéricas quanto de caracteres podem ser convertidas em fatores, mas os níveis de um fator são sempre valores de caracteres.



Fatores Ordenados

Fatores ordinais representam categorias com uma ordem natural (pequeno, médio, grande).



Fatores Não Ordenados

Fatores categóricos representam categorias sem ordem inerente (cores, países, tipos).

- ❑ Fatores são especialmente úteis em modelos estatísticos e gráficos, onde R pode tratá-los de forma apropriada para análises categóricas, aplicando contrastes e agrupamentos automaticamente.

Exemplo Prático: Fator Ordenado de Dor

Criando o Fator

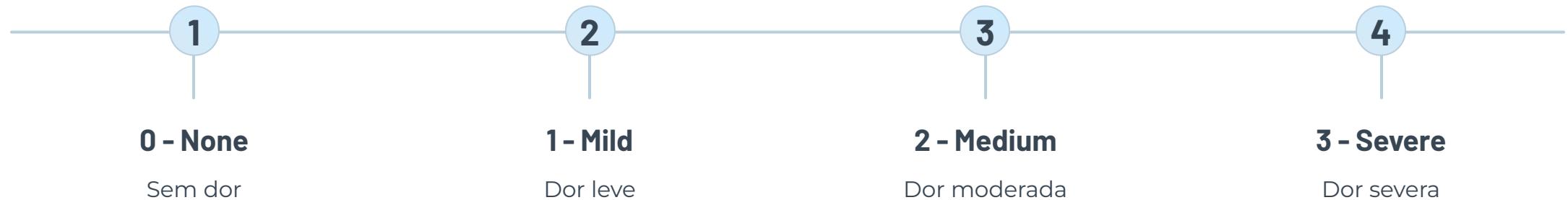
Vamos criar um fator ordenado indicando o nível de dor dos pacientes, onde 0 representa ausência de dor e 3 representa dor severa.

```
pain <- c(0, 3, 2, 2, 1)
fpain <- factor(pain, levels=0:3, ordered=TRUE)
fpain
## [1] 0 3 2 2 1
## Levels: 0 < 1 < 2 < 3
```

Atribuindo Rótulos

Os níveis fornecem correspondência entre valores numéricos e rótulos categóricos descritivos.

```
levels(fpain) <- c("none", "mild", "medium", "severe")
fpain
## [1] none  severe medium medium mild
## Levels: none < mild < medium < severe
```



Função cut(): Categorizando Dados Contínuos

A função `cut()` é poderosa para converter variáveis numéricas contínuas em fatores categóricos, dividindo o intervalo de valores em "bins" ou faixas.

Problema: Categorizar Alturas

Considere a variável altura. Queremos classificar pessoas em três categorias:

- Pessoas com menos de 1,5m são **baixas**
- Pessoas com mais de 1,9m são **altas**
- Pessoas entre 1,5m e 1,9m são de altura **média**

Implementação

```
lev <- cut(height,  
           breaks=c(0, 1.5, 1.9, .Machine$double.xmax),  
           ordered=TRUE)  
  
lev  
## [1] (1.5,1.9] (1.5,1.9] (1.5,1.9]  
## [4] (1.5,1.9] (1.5,1.9] (1.9,1.8e+308]  
## Levels: (0,1.5] < (1.5,1.9] < (1.9,1.8e+308]
```

Atribuindo Rótulos Descritivos

```
levels(lev) <- c("short", "medium", "tall")  
lev  
## [1] medium medium medium medium medium tall  
## Levels: short < medium < tall
```

 **Nota:** `.Machine$double.xmax` representa o maior número decimal que R pode representar, garantindo que todos os valores acima de 1,9m sejam capturados.

Matrizes: Estruturas Bidimensionais

O Que São Matrizes?

Uma matriz é uma estrutura de dados bidimensional em R. Todos os elementos devem ser do mesmo tipo (numérico, caractere, etc.).

Criando a partir de um Vetor

```
x <- 1:9  
x  
## [1] 1 2 3 4 5 6 7 8 9
```

Transformando em Matriz

As dimensões de um vetor podem ser alteradas para transformá-lo em uma matriz.

```
dim(x) <- c(3, 3)  
x  
## [,1] [,2] [,3]  
## [1,] 1 4 7  
## [2,] 2 5 8  
## [3,] 3 6 9
```

Matrizes são essenciais para operações de álgebra linear, processamento de imagens, e diversas aplicações em ciência de dados e estatística. R fornece operações eficientes para manipulação matricial.

Dimensões

Linhas e colunas formam a estrutura

Tipo Único

Todos elementos devem ser do mesmo tipo

Manipulação de Matrizes

Criação por Linha

A conversão de um vetor em matriz pode ser definida com preenchimento baseado em linhas usando o parâmetro `byrow=TRUE`.

```
x <- matrix(1:9, nrow=3, byrow=TRUE)  
x  
## [,1] [,2] [,3]  
## [1,] 1 2 3  
## [2,] 4 5 6  
## [3,] 7 8 9
```

Transposição

Uma matriz pode ser transposta usando a função `t()`, que troca linhas por colunas.

```
x <- t(x)  
x  
## [,1] [,2] [,3]  
## [1,] 1 4 7  
## [2,] 2 5 8  
## [3,] 3 6 9
```



Acesso por Linha

`x[1,]` retorna a primeira linha



Acesso por Coluna

`x[, 2]` retorna a segunda coluna



Elemento Específico

`x[2, 3]` retorna o elemento na linha 2, coluna 3

Listas: Estruturas Heterogêneas

Listas são objetos R que podem conter elementos de diferentes tipos, incluindo números, strings, vetores, matrizes, data frames e até outras listas. São extremamente flexíveis e fundamentais para programação avançada em R.

Números

Valores numéricos únicos ou vetores de números

Strings

Caracteres e texto de qualquer comprimento

Vetores

Sequências ordenadas de elementos do mesmo tipo

Matrizes

Estruturas bidimensionais de dados

Funções

Blocos de código reutilizáveis

Outras Listas

Listas podem conter outras listas

- Manipulação de Listas: Use `[]` para fatiar uma lista (retorna outra lista) e `[[]]` para acessar um elemento específico dentro da lista.

Trabalhando com Listas: Exemplos Práticos

Criando e Manipulando Listas

Criação Básica

```
a <- c(5260, 5470, 5640, 6180, 6390, 6515,  
      6805, 7515, 7515, 8230, 8770)  
b <- c(3910, 4220, 3885, 5160, 5645, 4680,  
      5265, 5975, 6790, 6900, 7335)  
mybag <- list(a, b, 0, "a")
```

Adicionando Elementos

```
n <- length(mybag)  
mybag[[n+1]] <- "b"
```

{

Fatiamento []

slice <- mybag[1] retorna uma lista



Acesso [[]]

h <- mybag[[1]] retorna o elemento

\$

Atributo \$

mybag\$x acessa elemento nomeado

Listas podem conter qualquer coisa, incluindo outras listas! Para adicionar um novo item, use uma posição vazia. Para remover, atribua NULL ao elemento. Listas com atributos nomeados facilitam o acesso e tornam o código mais legível.

Listas com Atributos Nomeados

```
mybag <- list(x=a, y=b, const=0, lit="a")  
mybag$c <- mybag$x - mybag$y
```

Removendo Elementos

```
mybag$const <- NULL  
mybag$lit <- NULL
```



Manipulação e Análise de Dados

Explore técnicas essenciais para trabalhar com dados estruturados em R, desde a criação de data frames até análises estatísticas avançadas e integração com Python.

Data Frames: Estruturas de Dados Tabulares

O que são Data Frames?

Um data frame é uma tabela onde cada coluna corresponde a atributos e cada linha corresponde a uma tupla (objeto). É a estrutura fundamental para análise de dados em R.

Data frames fornecem suporte robusto para dados estruturados, permitindo armazenar diferentes tipos de dados em cada coluna enquanto mantém a integridade relacional entre linhas.

Criando um Data Frame

```
a <- c(5260,5470,5640,6180,6390,  
      6515,6805,7515,7515,8230,8770)  
b <- c(3910,4220,3885,5160,5645,  
      4680,5265,5975,6790,6900,7335)  
  
data <- data.frame(A=a, B=b)  
head(data)  
  
## A B  
## 1 5260 3910  
## 2 5470 4220  
## 3 5640 3885  
## 4 6180 5160  
## 5 6390 5645  
## 6 6515 4680
```

Adicionando Colunas a um Data Frame

01

Utilize o Operador \$

O operador \$ é usado para acessar colunas específicas de um data frame pelo nome.

02

Crie Novas Colunas

Atribua valores a uma nova coluna para adicioná-la automaticamente ao data frame existente.

03

Realize Operações

Combine colunas existentes usando operações matemáticas ou lógicas para criar novos atributos.

```
data$c <- data$A + data$B  
head(data)
```

```
## A B c  
## 1 5260 3910 9170  
## 2 5470 4220 9690  
## 3 5640 3885 9525  
## 4 6180 5160 11340  
## 5 6390 5645 12035  
## 6 6515 4680 11195
```

Removendo Colunas de um Data Frame

Sintaxe Simples e Eficiente

Cada coluna possui um nome único que permite acesso direto. Para remover uma coluna, simplesmente atribua NULL ao nome da coluna usando o operador \$.

Esta operação modifica o data frame permanentemente, removendo todos os dados associados à coluna especificada. É importante ter cuidado ao executar esta operação, pois não há desfazer.

```
data$A <- NULL  
head(data)  
  
##   B   c  
## 1 3910 9170  
## 2 4220 9690  
## 3 3885 9525  
## 4 5160 11340  
## 5 5645 12035  
## 6 4680 11195
```

Dica Importante

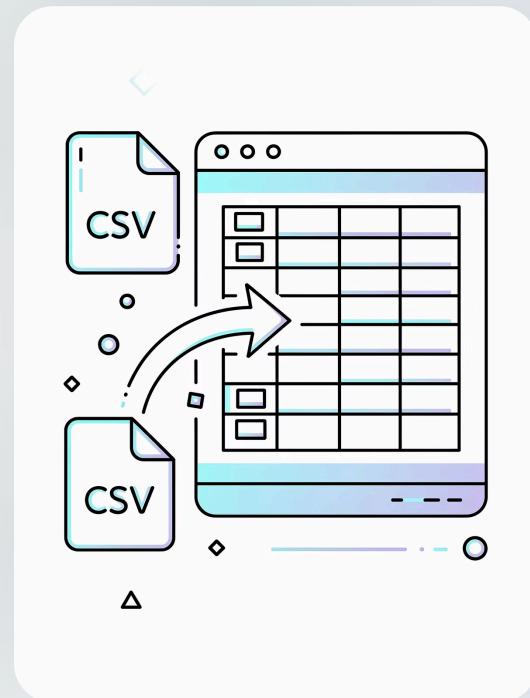
Sempre faça backup dos seus dados antes de remover colunas. Uma vez removida, a coluna não pode ser recuperada sem recarregar os dados originais.

Lendo Dados de Arquivos CSV

R oferece diversas funções para importar dados de diferentes formatos, incluindo CSV, Excel e RData. A função **read.table()** é versátil e permite personalizar como os dados são interpretados.

```
wine = read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data",
                  header = TRUE, sep = ",")
colnames(wine) <- c('Type', 'Alcohol', 'Malic', 'Ash','Alcalinity', 'Magnesium',
                    'Phenols','Flavanoids', 'Nonflavanoids','Proanthocyanins',
                    'Color', 'Hue','Dilution', 'Proline')
head(wine)

##  Type Alcohol Malic  Ash Alcalinity Magnesium Phenols Flavanoids Nonflavanoids
## 1  1   13.20  1.78 2.14    11.2     100  2.65    2.76     0.26
## 2  1   13.16  2.36 2.67    18.6     101  2.80    3.24     0.30
## 3  1   14.37  1.95 2.50    16.8     113  3.85    3.49     0.24
```



Experimente visualizar o arquivo original em: <http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data>

Salvando e Carregando Arquivos RData

1

Salvar

Use **save()** para armazenar data frames em formato binário compactado.

2

Remover

Utilize **rm()** para liberar memória removendo variáveis temporariamente.

3

Carregar

Empregue **load()** para restaurar dados salvos em sessões futuras.

Vantagens do Formato RData

- Compressão eficiente reduz espaço de armazenamento
- Preserva tipos de dados e estruturas complexas
- Carregamento rápido comparado a formatos de texto
- Ideal para salvar múltiplos objetos simultaneamente

```
## Salvando data frame  
save(wine, file="wine.RData")
```

```
## Removendo da memória  
rm(wine)
```

```
## Carregando novamente  
load("wine.RData")  
head(wine, 3)
```

```
## Type Alcohol Malic Acid  
## 1 113.20 1.78 2.14  
## 2 113.16 2.36 2.67  
## 3 114.37 1.95 2.50
```

Exportando Data Frames para CSV

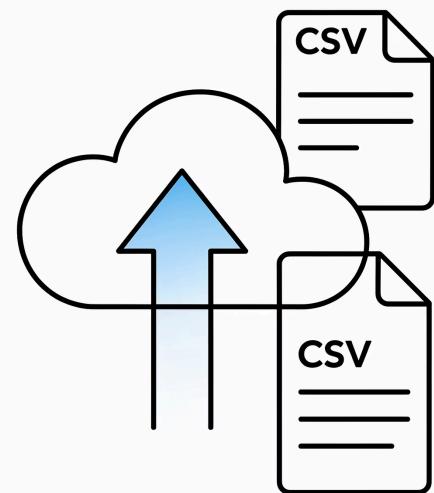
Compartilhamento Universal

A função **write.table()** permite exportar data frames para formato CSV, garantindo compatibilidade com Excel, Google Sheets e outras ferramentas de análise.

Configure os parâmetros cuidadosamente para controlar como os dados são formatados no arquivo de saída.

```
write.table(wine,  
           file="wine.csv",  
           row.names=FALSE,  
           quote = FALSE,  
           sep = ",")
```

- ❑ **Experimente:** Abra o arquivo wine.csv em um editor de texto ou planilha para visualizar o resultado da exportação.



Pipelines: Transformação de Dados Elegante

Pipelines em R permitem encadear operações de forma legível e eficiente. O operador `|>` (pipe nativo do R) passa o resultado de uma função como primeiro argumento da próxima, criando fluxos de transformação de dados intuitivos.



Filtrar

Selecione linhas baseadas em condições específicas



Selecionar

Escolha apenas as colunas relevantes para análise



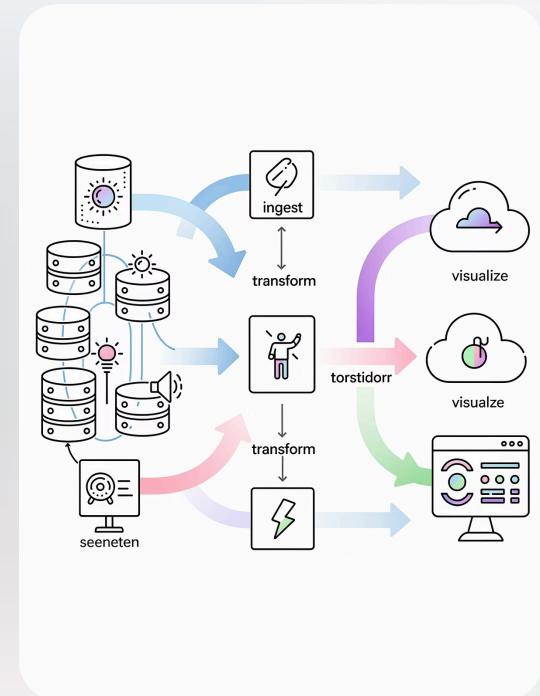
Agrupar

Organize dados por categorias para agregações



Resumir

Calcule estatísticas agregadas dos grupos



Visualizando os Dados de Voos

Conjunto de Dados

O data frame **flight_data** contém informações sobre voos por trimestre, incluindo o número total de voos e atrasos registrados.

Utilizar **head()** é fundamental para inspecionar rapidamente a estrutura e primeiras linhas de qualquer data frame antes de iniciar análises mais complexas.

```
head(flight_data)  
## Year Quarter Flights Delays  
## 1 2016 1 11 6  
## 2 2016 2 12 5  
## 3 2016 3 13 3  
## 4 2016 4 12 5  
## 5 2017 1 10 4  
## 6 2017 2 9 3
```

Consultas Básicas com Pipelines

Carregue a Biblioteca

Importe o pacote **dplyr** que fornece funções poderosas para manipulação de dados.

```
library(dplyr)

result <- flight_data |>
  filter(Delays > 5) |>
  select(Year, Quarter, Flights)
head(result)

## #> #> #> Year Quarter Flights
## #> #> 1 2016     1      11
## #> #> 2 2017     4      25
```

Filtre os Dados

Use **filter()** para selecionar apenas voos com mais de 5 atrasos, aplicando condições lógicas.

Selecione Colunas

Com **select()**, escolha apenas as colunas Year, Quarter e Flights para o resultado final.

Este pipeline demonstra como operações podem ser encadeadas de forma clara e legível, tornando o código mais fácil de entender e manter.

Consultas Agregadas: Estatísticas por Grupo

Análise Temporal

Consultas agregadas permitem calcular estatísticas resumidas para grupos de dados. Neste exemplo, agrupamos por ano e calculamos a média e desvio padrão do número de voos.

A função **group_by()** organiza os dados em grupos, enquanto **summarize()** aplica funções estatísticas como **mean()** e **sd()** a cada grupo.

```
result <- flight_data |>
  group_by(Year) |>
  summarize(
    mean = mean(Flights),
    sd = sd(Flights)
  )
head(result)
```

```
## # A tibble: 3 × 3
##   Year  mean   sd
##   <dbl> <dbl> <dbl>
## 1 2016  12.0  0.816
## 2 2017  13.8  7.54 
## 3 2018  13.5  1.29
```

- **Interpretação:** O ano de 2017 apresenta maior variabilidade ($sd = 7.54$), indicando flutuações significativas no número de voos entre trimestres.

Preparando Dados para Junção de Tabelas

Tabela de Lojas

Contém informações sobre valores de vendas por cidade.

```
stores <- data.frame(  
  city = c("Rio de Janeiro",  
          "Sao Paulo",  
          "Paris",  
          "New York",  
          "Tokyo"),  
  value = c(10, 12, 20, 25, 18)  
)  
head(stores)  
  
##      city value  
## 1 Rio de Janeiro 10  
## 2 Sao Paulo 12  
## 3 Paris 20  
## 4 New York 25  
## 5 Tokyo 18
```

Tabela de Divisões

Mapeia cada cidade ao seu respectivo país.

```
divisions <- data.frame(  
  city = c("Rio de Janeiro",  
          "Sao Paulo",  
          "Paris",  
          "New York",  
          "Tokyo"),  
  country = c("Brazil",  
             "Brazil",  
             "France",  
             "US",  
             "Japan")  
)  
head(divisions)  
  
##      city country  
## 1 Rio de Janeiro Brazil  
## 2 Sao Paulo Brazil  
## 3 Paris France  
## 4 New York US  
## 5 Tokyo Japan
```

Estas duas tabelas relacionadas serão combinadas através da coluna comum **city**, permitindo análises que integram informações geográficas e financeiras.

Função Merge: Combinando Tabelas

A função **merge()** é essencial para integrar dados de múltiplas fontes. Especifique a coluna de junção usando o parâmetro **by**, e escolha o tipo de junção com **all.x**, **all.y**, ou **all** para controlar quais linhas são preservadas.

Inner Join

Retorna apenas linhas com correspondência em ambas as tabelas, ideal para análises que exigem dados completos.

Left Join

Preserva todas as linhas da tabela esquerda, preenchendo com NA valores ausentes da direita.

Right Join

Mantém todas as linhas da tabela direita, complementando com NA onde necessário.

Full Join

Combina todas as linhas de ambas as tabelas, preenchendo com NA valores ausentes em qualquer lado.

Exemplo prático de uso da função merge():

```
stdiv <- merge(stores, divisions, by.x="city", by.y="city")
head(stdiv)
```

```
##      city value country
## 1  New York   25    US
## 2    Paris    20  France
## 3 Rio de Janeiro   10  Brazil
## 4   Sao Paulo   12  Brazil
## 5     Tokyo    18  Japan
```

Agregando Dados Mesclados

Pipeline Completo

Após mesclar as tabelas **stores** e **divisions**, podemos agrupar por país e calcular estatísticas agregadas.

Este exemplo conta o número de lojas por país usando **n()** e soma os valores totais com **sum()**, demonstrando o poder de pipelines para análises multi-etapa.

```
result <- stdiv |>
  group_by(country) |>
  summarize(
    count = n(),
    amount = sum(value)
  )
head(result)

## # A tibble: 4 × 3
##   country count amount
##   <chr>     <dbl>   <dbl>
## 1 Brazil      2     22
## 2 France      1     20
## 3 Japan       1     18
## 4 US          1     25
```

Análise Estatística: Teste t de Student

R oferece ampla variedade de testes estatísticos. O teste t verifica se a média de observações difere significativamente de um valor teórico esperado.

Exemplo: Teste de IMC

```
weight <- c(60, 72, 57, 90, 95, 72)
height <- c(1.75, 1.80, 1.65,
          1.90, 1.74, 1.91)
bmi <- weight/height^2

t.test(bmi, mu=22.5)

## One Sample t-test
## t = 0.34488, df = 5
## p-value = 0.7442
## 95% confidence interval:
## 18.41734 27.84791
## mean of x: 23.13262
```

Interpretando Resultados

- **Hipótese Nula:** A média do IMC é igual a 22.5 (p-valor > 0.05)
- **Hipótese Alternativa:** A média do IMC difere de 22.5
- **p-valor = 0.7442:** Não rejeitamos a hipótese nula, indicando que não há evidência estatística de diferença significativa
- **Intervalo de Confiança:** Com 95% de confiança, a média verdadeira está entre 18.42 e 27.85

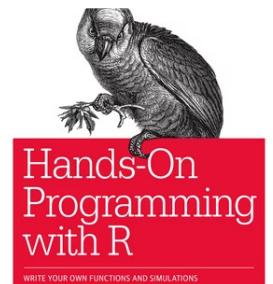
❑ Para mais detalhes sobre testes paramétricos e não-paramétricos: Health Knowledge - Statistical Methods

Integração R + Python



A biblioteca **reticulate** permite integração perfeita entre R e Python, aproveitando as forças de ambas as linguagens em um único fluxo de trabalho.

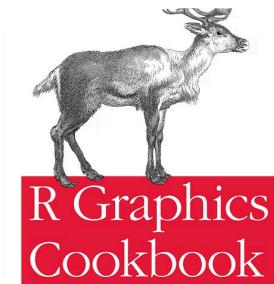
Referências



Hands-on Programming with R

Aprenda a escrever suas próprias funções e simulações

[rstudio-
education.github.io/hopr](https://rstudio-education.github.io/hopr)



R Graphics Cookbook

Guia completo para criar visualizações impressionantes

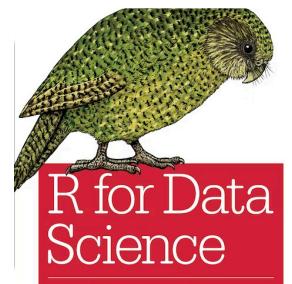
r-graphics.org



R Packages

Desenvolva e compartilhe seus próprios pacotes R

r-pkgs.org



R for Data Science

Referência definitiva para análise de dados moderna

r4ds.had.co.nz

Leitura adicional sobre fundamentos: <https://rstudio-education.github.io/hopr/basics.html>