Entre Florestas e Dados

Arthur Guilherme Schirmbeck Chaves

Última atualização: 2024-11-22

Sumário

Sobre

"Entre Florestas e Dados" é um livro digital dinâmico (sempre atualizado online) que compartilha soluções práticas para análises de dados florestais utilizando a linguagem R.

Por que este livro?

Este livro foi criado para preencher uma lacuna na literatura sobre análise de dados aplicada especificamente ao setor florestal, com o intuito de capacitar profissionais a tomar decisões fundamentadas em dados de inventários, planejamento e monitoramento ambiental.

Autor

Olá! Meu nome é Arthur Guilherme Schirmbeck Chaves. Sou professor de Engenharia Florestal no IFMT. Como engenheiro florestal e analista de sistemas com sólida experiência em planejamento florestal, inventário, silvicultura e análise de dados, estou desenvolvendo este livro para compartilhar soluções práticas e orientadas por dados para os engenheiros florestais empregarem em suas atividades profissionais e didáticas.

Capítulo 1

Introdução ao R

O R é provavelmente a mais importante ferramenta para a análise de dados florestais.

1.1 Instalação do R

A instalação padrão do R é feita a partir do CRAN, uma rede de servidores espalhada pelo mundo que armazena versões idênticas e atualizadas de códigos e documentações para o R.

1.1.1 Windows

Para instalar o R no Windows, siga os seguintes passos:

1. Acesse o CRAN: https://www.r-project.org/
2. No menu à esquerda, encontre a opção **Download** e clique em **CRAN**.
3. Escolha a opção de servidor (mirror) mais próxima de você.
4. Clique em **Download R for Windows**.
5. Clique na opção **base**.
6. Na nova página, clique em Download R x.x.x for Windows, sendo x.x.x o número da versão que será baixada. Se você teve algum problema com o download, tente escolher outro servidor no passo 3.
7. Feito o download, clique duas vezes no arquivo baixado e siga as instruções para instalação.

1.1.2 macOS

Abra o terminal: `Command + Espaço` "terminal" e `Enter`

```
brew install --cask r
```

1.1.3 Linux

Abra o terminal Linux: `ctrl + alt + t`

```
sudo apt update
sudo apt upgrade
sudo apt install -y r-base
```

1.2 Instalação do RStudio

Para instalar o RStudio, siga os passos atualizados abaixo:

Acesse a página de downloads do RStudio:

Visite https://posit.co/download/rstudio-desktop/#download. Verifique os requisitos do sistema:

Certifique-se de que seu sistema operacional é compatível: Windows: Windows 10 ou superior (64 bits). macOS: macOS 11 (Big Sur) ou superior. Linux: Distribuições compatíveis incluem Debian 10, Ubuntu 18.04 LTS, Ubuntu 20.04 LTS, Ubuntu 22.04 LTS, Debian 9, RHEL/CentOS 7, RHEL 8 e OpenSUSE/SLES 15. Baixe o instalador adequado:

Na seção "All Installers", selecione o instalador correspondente ao seu sistema operacional. Instale o RStudio:

1.2.1 Windows

Execute o arquivo .exe baixado e siga as instruções do assistente de instalação.

1.2.2 macOS

Abra o arquivo .dmg baixado, arraste o ícone do RStudio para a pasta "Aplicativos" e, em seguida, ejete a imagem de disco.

1.2.3 Linux

Baixe o pacote apropriado (.deb ou .rpm) e instale-o usando as ferramentas de gerenciamento de pacotes do seu sistema. Inicie o RStudio:

Após a instalação, abra o RStudio para começar a utilizá-lo. Lembre-se de que o RStudio requer o R instalado previamente. Certifique-se de ter o R instalado antes de prosseguir com a instalação do RStudio.

1.3 Sobre o R

1.3.1 O que são algoritmos?

Objetivo do Algoritmo: Plantar uma árvore.

Entradas do Algoritmo:

- Uma muda de árvore.
- Local de plantio escolhido.
- Ferramentas para cavar (como uma pá).
- Água para regar a muda.

Passos do Algoritmo:

1. Selecionar o Local: Escolher um local adequado para plantar a muda, considerando a exposição solar, o tipo de solo e o espaço necessário para o crescimento da árvore.
2. Cavar um Buraco: Usar a pá para cavar um buraco no local escolhido. O buraco deve ter profundidade e largura suficientes para acomodar as raízes da muda.
3. Preparar a Muda: Remover a muda do recipiente em que veio, com cuidado para não danificar as raízes.
4. Plantar a Muda: Colocar a muda no buraco, ajustando a profundidade para que a base do tronco fique no nível do solo. Preencher o buraco com terra, firmemente, mas sem compactar excessivamente.
5. Regar a Muda: Regar a muda generosamente para umedecer o solo e ajudar a estabelecer as raízes.
6. Cuidados Posteriores: Incluir instruções básicas de cuidados posteriores, como rega regular, aplicação de mulch para manter a umidade, e proteção contra ervas daninhas.

Saída do Algoritmo:

Uma árvore plantada e pronta para crescer no local escolhido.

1.3.2 O que é uma linguagem de programação?

Uma linguagem de programação é uma forma padronizada de comunicar instruções a um computador. É como uma língua que permite aos programadores escreverem códigos que são convertidos em ações executáveis pelo computador, permitindo a criação de software, aplicativos, websites, e muito mais. Cada linguagem de programação tem sua própria sintaxe e regras, assim como as línguas humanas têm gramática e vocabulário.

1.3.3 R

R é uma linguagem de programação voltada para análise estatística e visualização de dados.
Ela permite:
- Manipulação de grandes volumes de dados
- Criação de gráficos avançados
- Modelagem estatística complexa

Com uma vasta comunidade e milhares de pacotes disponíveis, **R** é ideal tanto para iniciantes quanto para especialistas em ciência de dados.

Alto Nível:
R é uma linguagem de programação de alto nível projetada para abstrair detalhes técnicos e permitir que os usuários se concentrem na análise de dados e estatísticas. Ela automatiza tarefas como alocação de memória e manipulação de estruturas complexas, permitindo maior foco nos problemas analíticos e menos em detalhes de implementação.

Foco em Análise de Dados:
Diferentemente de linguagens de propósito geral, R é especialmente projetada para análise estatística, ciência de dados e visualização. É amplamente usada em áreas como bioestatística, econometria, aprendizado de máquina e pesquisa acadêmica, fornecendo ferramentas especializadas para cada etapa do fluxo de trabalho analítico.

Visualizações Avançadas:
R é reconhecida por sua capacidade de criar gráficos e visualizações de dados de alta qualidade, tanto para análises exploratórias quanto para apresentações profissionais. Com pacotes como `ggplot2` e `plotly`, é possível produzir desde gráficos básicos até visualizações interativas complexas.

Tipagem Dinâmica e Flexibilidade:
R utiliza tipagem dinâmica, onde o tipo das variáveis é determinado em tempo de execução. Isso oferece flexibilidade para manipular diferentes tipos de dados, mas requer atenção para evitar inconsistências. Além disso, sua sintaxe foi projetada para facilitar operações vetoriais e manipulação de dados em larga escala, tornando-a eficiente em tarefas analíticas.

1.4 Fundamentos da linguagem

1.4.1 Boas práticas de codaR

1.4.1.1 O Básico

Acesse no menu superior `tools` > `Global Options...` > `code` > `Display` e defina os padrões de identação e estilização que deseja. E Salve.

Agora basta selecionar o trecho de código e usar o atalho `Ctrl + shift + A`(Windows).

1.4.1.2 Avançado

Para um estudo profundo verificar:

O manual de referências de boas práticas [Wickham and contributors, 2021]

1.4.2 ?Dúvidas

```
?mean
# ou
help(mean)
```

1.4.3 Tipos de dados

Conhecer os tipos de dados utilizados pela linguagem é essencial para sua manipulação e análise. A seguir, estão os principais tipos de dados:

Tipo	Descrição	Exemplo
`numeric`	Números, incluindo inteiros e reais	`10, 10.5`
`integer`	Números inteiros	`as.integer(10)`
`character`	Cadeias de caracteres (texto)	`"Olá"`
`logical`	Valores booleanos	`TRUE, FALSE`
`complex`	Números complexos	`1+2i`
`factor`	Variáveis categóricas	`factor(c("A", "B", "A"))`
`list`	Lista de elementos heterogêneos	`list(1, "a", TRUE)`
`data.frame`	Tabela de dados estruturados	`data.frame(x = 1:3, y = 4:6)`
`matrix`	Tabela de dados bidimensional	`matrix(1:6, nrow = 2)`
`array`	Dados com mais de duas dimensões	`array(1:8, dim = c(2, 2, 2))`
`NULL`	Representa a ausência de valor	`NULL`
`NA`	Valor ausente ou indefinido	`NA`

1.4.4 Operadores

Há inúmeras operações que podem ser realizadas com os dados dentro da lógica da programação. A seguir, estão os principais operadores:

Categoria	Operador	Descrição	Exemplo
Aritmético	`+`	Adição	`x + y`
	`-`	Subtração	`x - y`
	`*`	Multiplicação	`x * y`
	`/`	Divisão	`x / y`
	`^`	Exponenciação	`x ^ y`
	`%%`	Módulo (resto da divisão)	`x %% y`
	`%/%`	Divisão inteira	`x %/% y`
Comparação	`==`	Igual a	`x == y`
	`!=`	Diferente de	`x != y`
	`>`	Maior que	`x > y`
	`<`	Menor que	`x < y`
	`>=`	Maior ou igual a	`x >= y`
	`<=`	Menor ou igual a	`x <= y`
Lógico	`&`	AND elemento por elemento	`x & y`
	`|`	OR elemento por elemento	`x | y`
	`&&`	AND para o primeiro elemento	`x && y`
	`||`	OR para o primeiro elemento	`x || y`
	`!`	NOT lógico	`!x`
Atribuição	`<-`	Atribui um valor	`x <- y`
	`->`	Atribui um valor	`y -> x`
	`=`	Atribui um valor	`x = y`
Pertencimento	`%in%`	Pertencimento (contém)	`x %in% y`
Bit a bit	`bitwAnd()`	AND bit a bit	`bitwAnd(x, y)`
	`bitwOr()`	OR bit a bit	`bitwOr(x, y)`
	`bitwXor()`	XOR bit a bit	`bitwXor(x, y)`
	`bitwNot()`	NOT bit a bit	`bitwNot(x)`
	`bitwShiftL()`	Deslocamento à esquerda	`bitwShiftL(x, n)`
	`bitwShiftR()`	Deslocamento à direita	`bitwShiftR(x, n)`

1.4.5 Conversão de tipos

Por se tratar de uma linguagem de tipagem dinâmica, é possível converter os tipos de dados:

Função	Descrição	Exemplo	Resultado
`as.numeric()`	Converte para tipo numérico	`as.numeric("10.5")`	`10.5` (numeric)
`as.integer()`	Converte para inteiro	`as.integer(10.5)`	`10` (integer)
`as.character()`	Converte para texto (character)	`as.character(10)`	`"10"` (character)
`as.logical()`	Converte para lógico (booleano)	`as.logical(1)`	`TRUE` (logical)
		`as.logical(0)`	`FALSE` (logical)
`as.complex()`	Converte para número complexo	`as.complex(10)`	`10+0i` (complex)
`as.factor()`	Converte para fator (categorias)	`as.factor(c("A", "B"))`	`Factor` com níveis
`as.list()`	Converte para lista	`as.list(c(1, 2, 3))`	`list(1, 2, 3)`
`as.matrix()`	Converte para matriz	`as.matrix(c(1, 2, 3))`	`matrix` (2D)
`as.data.frame()`	Converte para data frame	`as.data.frame(matrix(1:4, ncol=2))`	`data.frame`

Função	Descrição	Exemplo	Resultado
`as.Date()`	Converte para data (classe Date)	`as.Date("2024-11-18")`	`2024-11-18` (Date)
`as.POSIXct()`	Converte para data/hora	`as.POSIXct("2024-11-18 10:00:00")`	Data/hora (POSIXct)
`as.vector()`	Converte para vetor	`as.vector(matrix(1:4, nrow=2))`	`c(1, 2, 3, 4)`
`as.array()`	Converte para array (multi-dimensional)	`as.array(1:8)`	`array`
`is.numeric()`	Testa se é numérico	`is.numeric(10.5)`	`TRUE`
`is.character()`	Testa se é texto	`is.character("text")`	`TRUE`
`is.logical()`	Testa se é lógico	`is.logical(TRUE)`	`TRUE`
`is.factor()`	Testa se é fator	`is.factor(as.factor("A"))`	`TRUE`

1.4.6 Estruturas de Controle

A lógica da programação e a construção de algoritmos dependem das estruturas de controle:

Estrutura	Descrição	Exemplo
`if`	Executa um bloco de código se a condição for verdadeira	`if (x > 0) { print("Positivo") }`
`if-else`	Executa um bloco se a condição for verdadeira e outro se for falsa	`if (x > 0) { print("Positivo") } else { print("Negativo") }`
`ifelse`	Avaliação vetorizada para condicional	`result <- ifelse(x > 0, "Positivo", "Negativo")`
`for`	Itera sobre elementos de um vetor ou lista	`for (i in 1:5) { print(i) }`
`while`	Executa um bloco enquanto a condição for verdadeira	`while (x < 5) { print(x); x <- x + 1 }`
`repeat`	Executa um bloco até encontrar um `break`	`repeat { if (x > 5) break; print(x); x <- x + 1 }`
`break`	Interrompe a execução de um laço	`for (i in 1:5) { if (i == 3) break; print(i) }`
`next`	Pula para a próxima iteração de um laço	`for (i in 1:5) { if (i == 3) next; print(i) }`
`switch`	Seleciona uma opção com base em uma expressão	`switch(x, "a" = "Opção A", "b" = "Opção B")`
Funções anônimas	Executa expressões diretamente em chamadas	`sapply(1:5, function(x) x^2)`
`try`	Captura erros em blocos de código	`try({ log("a") })`
`tryCatch`	Captura e lida com erros e mensagens	`tryCatch(log("a"), error = function(e) print("Erro"))`

1.5 Arrays e Dataframes

1.5.1 Vetor

Um vetor é a estrutura mais básica no R. Ele representa um array unidimensional que armazena elementos do mesmo tipo (numérico, lógico, caractere, etc.).

Podemos criar vetores usando a função c().

```r
# Criando vetores de diferentes tipos
vetor_numerico <- c(1, 2, 3, 4, 5)
vetor_caractere <- c("Maçã", "Banana", "Laranja")
vetor_logico <- c(TRUE, FALSE, TRUE)

# Visualizando os vetores
print(vetor_numerico)
```

```
## [1] 1 2 3 4 5
```

```r
print(vetor_caractere)
```

```
## [1] "Maçã"    "Banana"  "Laranja"
```

```r
print(vetor_logico)
```

```
## [1]  TRUE FALSE  TRUE
```

1.5.2 Arrays

Um array em R é uma estrutura de dados multidimensional que pode armazenar elementos de um único tipo (númerico, caractere, etc.). Ele é como uma extensão de vetores, podendo ter duas ou mais dimensões.

1.5.2.1 Criando Arrays

Para criar um array, usamos a função array().

```r
# Criando um array 3x3x2
meu_array <- array(data = 1:18, dim = c(3, 3, 2))

# Visualizando o array
print(meu_array)
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   10   13   16
## [2,]   11   14   17
## [3,]   12   15   18
```

O exemplo acima cria um array com 3 linhas, 3 colunas e 2 “camadas”.

1.5.2.2 Acessando elementos

Os elementos em arrays podem ser acessados usando índices.

```
# Acessar o elemento na posição [1, 2, 1]
meu_array[1, 2, 1]
```

```
## [1] 4
```

```
# Acessar a segunda camada inteira
meu_array[, , 2]
```

```
##      [,1] [,2] [,3]
## [1,]   10   13   16
## [2,]   11   14   17
## [3,]   12   15   18
```

1.5.3 Dataframes

Os dataframes são a estrutura de dados mais comum no R para manipular tabelas de dados. Eles permitem armazenar diferentes tipos de variáveis (numéricas, categóricas, etc.) em colunas.

1.5.3.1 Criando Dataframes

```
# Criando um dataframe
meu_dataframe <- data.frame(
  Nome = c("João", "Ana", "Carlos"),
  Idade = c(25, 30, 22),
  Altura = c(1.75, 1.68, 1.80)
)

# Visualizando o dataframe
print(meu_dataframe)
```

```
##     Nome Idade Altura
## 1   João    25   1.75
## 2    Ana    30   1.68
## 3 Carlos    22   1.80
```

1.5.3.2 Acessando elementos

Você pode acessar linhas, colunas ou células específicas de um dataframe.

```
# Acessar a coluna "Idade"
meu_dataframe$Idade
```

```
## [1] 25 30 22
```

```
# Acessar a primeira linha
meu_dataframe[1, ]
```

```
##   Nome Idade Altura
## 1 João    25   1.75
```

```
# Acessar um elemento específico (linha 2, coluna 3)
meu_dataframe[2, 3]
```

```
## [1] 1.68
```

1.5.3.3 Criando ou Removendo uma coluna

```
# Adicionar uma nova coluna
meu_dataframe$Peso <- c(70, 60, 80)

print(meu_dataframe)
```

```
##     Nome Idade Altura Peso
## 1   João    25   1.75   70
## 2    Ana    30   1.68   60
## 3 Carlos    22   1.80   80
```

```
# Remover a coluna "Peso"
meu_dataframe$Peso <- NULL

print(meu_dataframe)
```

```
##     Nome Idade Altura
## 1   João    25   1.75
## 2    Ana    30   1.68
## 3 Carlos    22   1.80
```

1.6 Listas

As listas são estruturas mais flexíveis do que vetores ou dataframes. Uma lista pode armazenar diferentes tipos de objetos, incluindo vetores, dataframes, e até outras listas.

1.6.1 Criando listas

```
# Criando uma lista
minha_lista <- list(
  numeros = c(1, 2, 3),
  nomes = c("Alice", "Bob", "Carol"),
  dataframe = meu_dataframe
)

# Visualizando a lista
print(minha_lista)
```

```
## $numeros
## [1] 1 2 3
##
## $nomes
## [1] "Alice" "Bob"   "Carol"
##
## $dataframe
##     Nome Idade Altura
## 1   João    25   1.75
## 2    Ana    30   1.68
## 3 Carlos    22   1.80
```

1.6.2 Acessando Elementos de Listas

Os elementos de uma lista podem ser acessados por índices ou nomes.

```
# Acessar o primeiro elemento
minha_lista[[1]]
```

```
## [1] 1 2 3
```

```
# Acessar pelo nome
minha_lista$numeros
```

```
## [1] 1 2 3
```

```
# Acessar elementos dentro de um dataframe armazenado
minha_lista$dataframe$Idade
```

```
## [1] 25 30 22
```

1.6.3 Dataframe vs. Lista

Embora um dataframe seja tecnicamente uma lista, ele é estruturado de maneira que todas as colunas tenham o mesmo número de elementos e seja mais simples de trabalhar com dados tabulares. Já as listas são muito mais flexíveis.

Característica	Dataframe	Lista
Estrutura	Tabela (linhas e colunas)	Estrutura hierárquica
Tipos de dados	Colunas podem ter tipos diferentes	Elementos podem ser de qualquer tipo, incluindo listas ou dataframes
Indexação	Por linhas e colunas	Por índices ou nomes
Uso típico	Dados tabulares	Agrupamento de objetos heterogêneos

1.7 Gerenciamento de Pacotes

O R é uma linguagem poderosa, ampliada pelo uso de pacotes. Com pacotes, é possível adicionar novas funcionalidades e acessar ferramentas específicas para diversas tarefas, como visualização de dados, manipulação e modelagem estatística.

1.7.1 O que são pacotes?

Pacotes no R são coleções de funções, conjuntos de dados e documentações. Eles estendem as funcionalidades do R base e são fundamentais para tarefas mais avançadas.

- Os pacotes podem ser obtidos do **CRAN** (repositório oficial), **GitHub** ou outras fontes.
- Pacotes populares incluem `ggplot2` (gráficos), `dplyr` (manipulação de dados) e `shiny` (aplicações interativas).

1.7.2 Instalando pacotes

Para instalar pacotes no R, utilize a função `install.packages()`.

1.7.2.1 Exemplo básico:

```
# Instalando um pacote do CRAN
install.packages("ggplot2")
```

1.7.2.2 Instalando múltiplos pacotes:

```
install.packages(c("dplyr", "tidyr", "stringr"))
```

1.7.2.3 Instalando pacotes do GitHub:

Pacotes disponíveis no GitHub requerem o pacote `remotes` ou `devtools` para instalação:

```
# Instalando o pacote remotes
install.packages("remotes")

# Instalando um pacote do GitHub
remotes::install_github("tidyverse/ggplot2")
```

1.7.3 Carregando pacotes

Depois de instalados, os pacotes precisam ser carregados para uso. Isso é feito com a função `library()`.

1.7.3.1 Exemplo:

```
# Carregando o pacote ggplot2
library(ggplot2)
```

Se você não quiser carregar o pacote inteiro, pode usar o operador `::` para chamar uma função específica:

```
# Usando a função ggplot() sem carregar o pacote
ggplot2::ggplot(data = mtcars, aes(x = mpg, y = hp))
```

1.7.4 Atualizando pacotes

Pacotes instalados podem ser atualizados para suas versões mais recentes.

1.7.4.1 Atualizando todos os pacotes:

```
update.packages()
```

1.7.4.2 Atualizando pacotes específicos:

Reinstale o pacote desejado:

```
install.packages("dplyr")
```

1.7.5 Removendo pacotes

Se um pacote não for mais necessário, você pode removê-lo com a função `remove.packages()`.

1.7.5.1 Exemplo:

```
remove.packages("stringr")
```

1.7.6 Verificando pacotes instalados

Você pode listar todos os pacotes instalados ou verificar se um pacote específico está presente.

1.7.6.1 Listando todos os pacotes:

```
installed.packages()
```

1.7.6.2 Verificando se um pacote está instalado:

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

1.7.7 Repositórios

O R utiliza o **CRAN** como repositório padrão para instalação de pacotes. No entanto, você pode configurar o repositório manualmente, por exemplo, escolhendo um mirror mais rápido:

```
# Configurando um repositório brasileiro
options(repos = c(CRAN = "https://cran.rstudio.com/"))
```

1.7.8 Ferramentas avançadas para gerenciamento

Para projetos complexos, você pode usar ferramentas para gerenciar versões de pacotes e ambientes:

- **renv**: Cria ambientes isolados para projetos, garantindo que as dependências permaneçam consistentes:

```
install.packages("renv")
renv::init()
```

- **packrat**: Alternativa mais antiga ao **renv**, também para isolamento de dependências.
- **conda**: Se você utiliza Python junto com R, o **conda** permite gerenciar pacotes R em ambientes híbridos.

1.8 Importação e Exportação de Dados

O R oferece diversas ferramentas para **importar** e **exportar dados** em diferentes formatos, permitindo que você trabalhe com arquivos como CSV, Excel, bancos de dados, arquivos de texto e muito mais.

1.8.1 Importando dados

1.8.1.1 1. Arquivos CSV

O formato CSV é amplamente utilizado e facilmente manipulável no R com a função `read.csv()` ou a função mais moderna `readr::read_csv()`.

```
# Usando a função base
dados <- read.csv("caminho/para/seu/arquivo.csv")

# Usando o pacote readr (do Tidyverse)
dados <- readr::read_csv("caminho/para/seu/arquivo.csv")
```

1.8.1.2 2. Arquivos Excel

Para importar dados de planilhas Excel, você pode usar pacotes como **readxl** ou **openxlsx**.

```
# Instale o pacote readxl, se necessário
install.packages("readxl")

# Lendo uma planilha Excel
library(readxl)
dados <- read_excel("caminho/para/seu/arquivo.xlsx", sheet = 1)
```

1.8.1.3 3. Arquivos de texto

Para arquivos de texto delimitados (como TSV), use `read.delim()` ou `readr::read_tsv()`.

```r
# Arquivo delimitado por tabulação
dados <- read.delim("caminho/para/seu/arquivo.txt")

# Com readr
dados <- readr::read_tsv("caminho/para/seu/arquivo.txt")
```

1.8.1.4 4. Bancos de Dados

Para se conectar a bancos de dados, use pacotes como `DBI` e `RSQLite`.

```r
# Exemplo com SQLite
install.packages("DBI")
install.packages("RSQLite")

library(DBI)
con <- dbConnect(RSQLite::SQLite(), "caminho/para/seu/banco.sqlite")
dados <- dbReadTable(con, "nome_da_tabela")
dbDisconnect(con)
```

1.8.1.5 5. Outros formatos populares

- JSON: Use o pacote `jsonlite`.
- XML: Use o pacote `xml2`.
- Arquivos SPSS, Stata e SAS: Use o pacote `haven`.

```r
# Exemplo de JSON
install.packages("jsonlite")
dados <- jsonlite::fromJSON("caminho/para/seu/arquivo.json")
```

1.8.2 Exportando dados

1.8.2.1 1. Exportando para CSV

A função `write.csv()` é usada para salvar dados em formato CSV.

```r
# Salvando um data frame como CSV
write.csv(dados, "caminho/para/saida.csv", row.names = FALSE)
```

1.8.2.2 2. Exportando para Excel

Para salvar dados em Excel, use o pacote `openxlsx`.

```r
# Instale o pacote openxlsx, se necessário
install.packages("openxlsx")

# Escrevendo dados em uma planilha Excel
library(openxlsx)
write.xlsx(dados, "caminho/para/saida.xlsx")
```

1.8.2.3 3. Exportando para outros formatos

O R permite salvar dados em diversos formatos com as funções correspondentes: - Texto (`write.table()`) - JSON (`jsonlite::toJSON()`) - Bancos de dados (`dbWriteTable()`).

Exemplo com JSON:

```
jsonlite::toJSON(dados, pretty = TRUE, file = "caminho/para/saida.json")
```

1.8.3 Dicas e boas práticas

- Sempre verifique os tipos de dados importados com `str()` ou `glimpse()`.
- Para grandes volumes de dados, considere pacotes otimizados como `data.table` ou `vroom`.
- Configure o diretório de trabalho corretamente para evitar erros de caminho:

```
setwd("caminho/para/seu/diretorio")
```

Capítulo 2

R Intermediário

2.1 Funções da família `apply()` no R

As funções da família `apply()` são usadas para realizar operações repetitivas em vetores, listas, matrizes e data frames de forma eficiente e vetorizada, substituindo muitos loops explícitos (`for`).

2.1.1 1. `apply()`

Aplica uma função ao longo de uma **margem** de uma matriz ou array (linhas ou colunas).

- **Uso:** Para matrizes e arrays.
- **Sintaxe:** `apply(X, MARGIN, FUN, ...)`
  - `X`: Matriz ou array.
  - `MARGIN`: Margem a operar:
    - `1` para linhas.
    - `2` para colunas.
  - `FUN`: Função a ser aplicada.

Exemplo:

```r
# Soma das colunas de uma matriz
mat <- matrix(1:9, nrow = 3)
apply(mat, 2, sum)
# Resultado: [1] 12 15 18

# Média das linhas
apply(mat, 1, mean)
# Resultado: [1] 2 5 8
```

2.1.2 2. `lapply()`

Aplica uma função a cada elemento de uma **lista** ou **vetor** e retorna uma **lista** como resultado.

- **Uso:** Para listas e vetores.

- **Sintaxe:** `lapply(X, FUN, ...)`

Exemplo:

```r
# Quadrado de cada elemento de um vetor
vec <- 1:5
lapply(vec, function(x) x^2)
# Resultado: [[1]] 1, [[2]] 4, [[3]] 9, [[4]] 16, [[5]] 25

# Aplicação em uma lista
lst <- list(a = 1:3, b = 4:6)
lapply(lst, sum)
# Resultado: $a 6, $b 15
```

2.1.3 3. `sapply()`

Uma versão simplificada de `lapply()` que retorna um **vetor** ou **matriz** (se possível) em vez de uma lista.

- **Uso:** Para listas e vetores.
- **Sintaxe:** `sapply(X, FUN, ...)`

Exemplo:

```r
# Quadrado de cada elemento
vec <- 1:5
sapply(vec, function(x) x^2)
# Resultado: [1] 1 4 9 16 25

# Soma dos elementos em uma lista
lst <- list(a = 1:3, b = 4:6)
sapply(lst, sum)
# Resultado: [1] 6 15
```

2.1.4 4. `vapply()`

Semelhante a `sapply()`, mas requer que você **especifique o tipo de retorno esperado** (mais seguro).

- **Uso:** Para listas e vetores.
- **Sintaxe:** `vapply(X, FUN, FUN.VALUE, ...)`
  - `FUN.VALUE`: Define o tipo e formato esperado do retorno.

Exemplo:

```r
# Soma dos elementos em uma lista com tipo esperado (numeric)
lst <- list(a = 1:3, b = 4:6)
vapply(lst, sum, numeric(1))
# Resultado: [1]  6 15
```

2.1.5 5. mapply()

Aplica uma função a múltiplos argumentos/vetores **simultaneamente** (como um `map` em Python).

- **Uso:** Para múltiplos vetores ou listas.
- **Sintaxe:** `mapply(FUN, ..., MoreArgs = NULL)`

Exemplo:

```r
# Soma de dois vetores
vec1 <- 1:5
vec2 <- 6:10
mapply(sum, vec1, vec2)
# Resultado: [1]  7  9 11 13 15

# Repetição customizada
mapply(rep, 1:3, 3:1)
# Resultado: [[1]] 1, [[2]] 2 2, [[3]] 3 3 3
```

2.1.6 6. tapply()

Aplica uma função a subconjuntos de um vetor, definidos por um **fator** ou grupos.

- **Uso:** Para agrupamento.
- **Sintaxe:** `tapply(X, INDEX, FUN, ...)`
  - `X`: Vetor numérico.
  - `INDEX`: Fator ou lista de fatores para agrupar.

Exemplo:

```r
# Média por grupo
vec <- c(1, 2, 3, 4, 5, 6)
grp <- factor(c("A", "A", "B", "B", "C", "C"))
tapply(vec, grp, mean)
# Resultado: $A 1.5, $B 3.5, $C 5.5
```

2.1.7 7. by()

Semelhante a `tapply()`, mas retorna resultados organizados por **subgrupos** e funciona com data frames.

- **Uso:** Para data frames.
- **Sintaxe:** `by(data, INDICES, FUN, ...)`

Exemplo:

```r
# Soma dos valores por grupo em um data frame
df <- data.frame(value = 1:6, group = c("A", "A", "B", "B", "C", "C"))
by(df$value, df$group, sum)
# Resultado:
# A: 3
# B: 7
# C: 11
```

	mpg	cyl	hp
Mazda RX4	21.0	6	110
Mazda RX4 Wag	21.0	6	110
Datsun 710	22.8	4	93
Hornet 4 Drive	21.4	6	110
Merc 240D	24.4	4	62
Merc 230	22.8	4	95

Conclusão

As funções da família `apply()` tornam o R extremamente poderoso e eficiente para manipulação de dados. Use: - **`apply()`** para matrizes/arrays. - **`lapply()`** e **`sapply()`** para listas/vetores. - **`mapply()`** para múltiplos vetores. - **`tapply()`** e **`by()`** para agrupamentos.

2.2 dplyr

2.2.1 O que é o Pipe %>%?

O operador **%>%** é chamado de pipe. Ele permite encadear funções de maneira a facilitar a leitura e compreensão do código. Ao invés de ficar atribuindo valores intermediários ou aninhando várias funções, você simplesmente passa o resultado de uma função como entrada para a próxima.

2.2.1.1 Exemplos de Uso

Vamos usar o conjunto de dados mtcars para entender como o pipe facilita a manipulação de dados:

Sem **%>%** (Sem Pipe) Imagine que queremos fazer as seguintes operações no dataset mtcars:

Filtrar apenas os carros que têm mais de 20 milhas por galão (mpg > 20). Selecionar apenas as colunas mpg, cyl, e hp. Sem o operador **%>%**, o código ficaria assim:

```
library(dplyr)

# Sem pipe
filtered_data <- filter(mtcars, mpg > 20)
selected_data <- select(filtered_data, mpg, cyl, hp)
```

Com **%>%** (Com Pipe) Podemos simplificar isso usando o pipe:

```
library(dplyr)

# Usando pipe
selected_data <- mtcars %>%
  filter(mpg > 20) %>%
  select(mpg, cyl, hp)
```

Note como o código com **%>%** é mais legível e direto. Ele segue uma lógica sequencial que permite facilmente entender o que está sendo feito em cada etapa.

2.2.2 Tibbles

Tibbles são data frames com ajustes que os deixam mais amigáveis aos cientistas de dados. Elas são parte do pacote {tibble}. Assim, para começar a usá-las, instale e carregue o pacote.

```
install.packages("tibble")
library(tibble)
```

Criando um exemplo:

```
# Criando um exemplo de dados florestais
library(dplyr)

dados_florestais <- tibble::tibble(
  parcela = rep(1:5, each = 10),
  especie = sample(c("Cedro", "Ipê", "Jatobá", "Angelim", "Castanheira"), 50, replace
   ↪  = TRUE),
  dap_cm = round(runif(50, 10, 80), 1),  # Diâmetro em cm
  altura_m = round(runif(50, 5, 30), 1),  # Altura em metros
  volume_m3 = dap_cm * altura_m * 0.00007854 # Fórmula fictícia para volume
)

dados_florestais
```

```
## # A tibble: 50 x 5
##    parcela especie     dap_cm altura_m volume_m3
##      <int> <chr>        <dbl>    <dbl>     <dbl>
##  1       1 Ipê           32.5     22.3    0.0569
##  2       1 Ipê           43.5     12.1    0.0413
##  3       1 Castanheira   12       25.3    0.0238
##  4       1 Cedro         48.3      7.3    0.0277
##  5       1 Cedro         55.1     25.6    0.111
##  6       1 Castanheira   51.7     15.7    0.0638
##  7       1 Castanheira   32.5     23.9    0.0610
##  8       1 Angelim       72.4     21.6    0.123
##  9       1 Castanheira   53.8     16.1    0.0680
## 10       1 Castanheira   31.2     20.7    0.0507
## # i 40 more rows
```

2.2.3 filter()

Podemos filtrar árvores de uma espécie específica ou com características particulares. Exemplo: selecionar árvores com DAP maior que 50 cm.

```
# Filtrando árvores com DAP > 50 cm
arvores_grandes <- dados_florestais %>%
  filter(dap_cm > 50)

head(arvores_grandes)
```

```
## # A tibble: 6 x 5
##   parcela especie     dap_cm altura_m volume_m3
##     <int> <chr>        <dbl>    <dbl>     <dbl>
```

```
## 1       1 Cedro         55.1     25.6    0.111
## 2       1 Castanheira   51.7     15.7    0.0638
## 3       1 Angelim       72.4     21.6    0.123
## 4       1 Castanheira   53.8     16.1    0.0680
## 5       2 Castanheira   70.4     22.6    0.125
## 6       2 Cedro         76.7     10.4    0.0626
```

2.2.4 select()

Para trabalhar com um subconjunto de variáveis, podemos usar select(). Exemplo: selecionar apenas as colunas especie, dap_cm e altura_m.

```
# Selecionando colunas específicas
dados_selecionados <- dados_florestais %>%
  select(especie, dap_cm, altura_m)

head(dados_selecionados)
```

```
## # A tibble: 6 x 3
##   especie     dap_cm altura_m
##   <chr>        <dbl>    <dbl>
## 1 Ipê           32.5     22.3
## 2 Ipê           43.5     12.1
## 3 Castanheira   12       25.3
## 4 Cedro         48.3      7.3
## 5 Cedro         55.1     25.6
## 6 Castanheira   51.7     15.7
```

2.2.5 mutate()

Podemos criar novas variáveis com base em cálculos. Exemplo: calcular o índice de esbeltez (altura/DAP).

```
# Adicionando o índice de esbeltez
dados_florestais <- dados_florestais %>%
  mutate(indice_esbeltez = altura_m / dap_cm)

head(dados_florestais)
```

```
## # A tibble: 6 x 6
##   parcela especie     dap_cm altura_m volume_m3
##     <int> <chr>        <dbl>    <dbl>     <dbl>
## 1       1 Ipê           32.5     22.3    0.0569
## 2       1 Ipê           43.5     12.1    0.0413
## 3       1 Castanheira   12       25.3    0.0238
## 4       1 Cedro         48.3      7.3    0.0277
## 5       1 Cedro         55.1     25.6    0.111
## 6       1 Castanheira   51.7     15.7    0.0638
## # i 1 more variable: indice_esbeltez <dbl>
```

2.2.6 arrange()

Para ordenar as árvores pelo DAP em ordem decrescente:

```
# Ordenando por DAP
dados_ordenados <- dados_florestais %>%
  arrange(desc(dap_cm))

head(dados_ordenados)
```

```
## # A tibble: 6 x 6
##   parcela especie dap_cm altura_m volume_m3
##     <int> <chr>    <dbl>    <dbl>     <dbl>
## 1       2 Cedro     79.8     22.2    0.139
## 2       5 Cedro     78.2     16.6    0.102
## 3       2 Cedro     76.7     10.4    0.0626
## 4       1 Angelim   72.4     21.6    0.123
## 5       4 Jatobá    72.3      9.5    0.0539
## 6       3 Ipê       70.7      7.2    0.0400
## # i 1 more variable: indice_esbeltez <dbl>
```

2.2.7 summarise() e group_by()

Agrupar e resumir dados é uma tarefa comum em análises florestais. Exemplo: calcular o DAP médio e o volume total por espécie.

```
# Resumo por espécie
resumo_especies <- dados_florestais %>%
  group_by(especie) %>%
  summarise(
    dap_medio = mean(dap_cm),
    volume_total = sum(volume_m3)
  )

resumo_especies
```

```
## # A tibble: 5 x 3
##   especie     dap_medio volume_total
##   <chr>           <dbl>        <dbl>
## 1 Angelim          45.0        0.426
## 2 Castanheira      38.4        0.721
## 3 Cedro            51.6        0.699
## 4 Ipê              36.9        0.588
## 5 Jatobá           44.5        0.399
```

2.2.8 ungroup()

Após usar group_by() em uma análise, o objeto resultante permanece "agrupado". Isso pode afetar operações subsequentes. Para remover o agrupamento, usamos ungroup().

Exemplo: calcular o DAP médio por espécie e, em seguida, adicionar uma coluna ao dataset original com o DAP médio total (sem agrupamento).

```
# Resumo por espécie com agrupamento
resumo_especies <- dados_florestais %>%
  group_by(especie) %>%
  summarise(dap_medio = mean(dap_cm))

# Adicionando o DAP médio geral
```

```
dap_geral <- resumo_especies %>%
  ungroup() %>%
  summarise(dap_medio_geral = mean(dap_medio))

dap_geral
```

```
## # A tibble: 1 x 1
##   dap_medio_geral
##             <dbl>
## 1            43.3
```

2.3 data.table

2.3.1 O que é o :=?

O operador := é chamado de operador de atribuição por referência. Ele pertence ao pacote data.table e é usado para adicionar ou modificar colunas em um data.table. O := é mais eficiente do que usar <- em um data.table, pois faz as modificações "por referência", o que significa que não cria cópias desnecessárias dos dados, sendo muito mais rápido e eficiente em termos de memória.

2.3.2 Exemplos de Uso

Vamos usar o pacote data.table para entender como o := funciona.

Primeiro, vamos criar um data.table com alguns dados fictícios:

```
library(data.table)

# Criação de um data.table
DT <- data.table(
  id = 1:5,
  valor = c(10, 15, 20, 25, 30)
)
```

Adicionando uma Nova Coluna com :=

Podemos adicionar uma nova coluna chamada valor_2 que seja o dobro do valor existente na coluna valor:

```
DT[, valor_2 := valor * 2]

print(DT)
```

```
##    id valor valor_2
## 1:  1    10      20
## 2:  2    15      30
## 3:  3    20      40
## 4:  4    25      50
## 5:  5    30      60
```

Modificando uma Coluna Existente com :=

Podemos também modificar uma coluna existente. Vamos modificar a coluna valor_2 para ser o triplo do valor da coluna valor:

```
DT[, valor_2 := valor * 3]

print(DT)
```

```
##    id valor valor_2
## 1:  1    10      30
## 2:  2    15      45
## 3:  3    20      60
## 4:  4    25      75
## 5:  5    30      90
```

2.4 Manipulando dados com dplyr

O pacote dplyr é uma das ferramentas mais populares do ecossistema R para manipulação de dados. Ele fornece uma sintaxe simples e eficiente para tarefas como filtragem, ordenação, agrupamento e transformação de dados. Neste capítulo, exploraremos as principais funções do dplyr aplicadas a um conjunto de dados florestais.

Capítulo 3

Dendrometria

3.1 Diâmetros

3.1.1 Histogramas de distribuição diamétrica

Este tipo de abordagem é útil para se verificar o grau de aproximação da Normalidade dos dados pela "formato" das classes através da frequência de indivíduos por classe diamétrica.

Os engenheiros florestais geralmente estabelecem classes diamétricas em intervalos fixos de 1,5 ou 2,0 cm para poderem comparar entre si as inúmeras parcelas; bem como para acompanhar a mudança de classe das árvores ao longo do tempo.

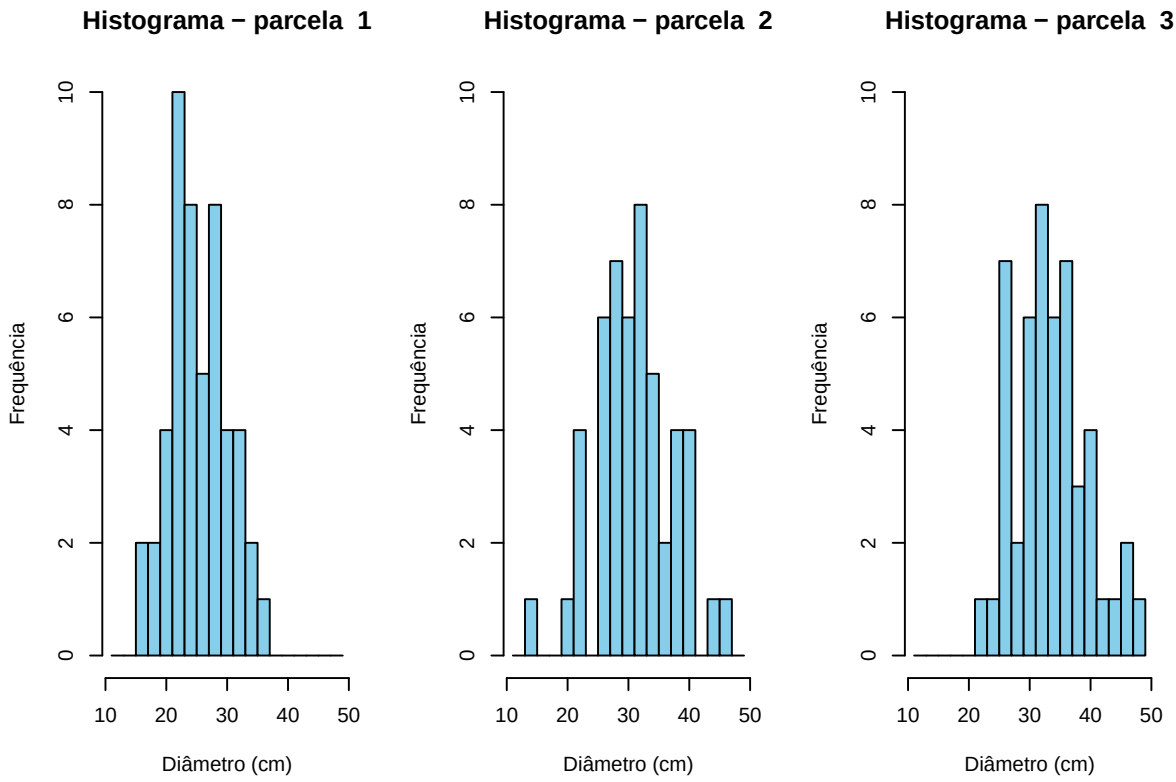
3.1.1.1 Criando dados para demonstração

```
# Definir número de árvores por parcela
n_arvores <- 50

# Gerar dados de 3 parcelas com diâmetros distribuídos normalmente
set.seed(123)  # Para garantir reprodutibilidade

parcela1 <- rnorm(n_arvores, mean = 25, sd = 5)  # Parcela 1 com média 25 cm e desvio
 ↪  padrão 5 cm
parcela2 <- rnorm(n_arvores, mean = 30, sd = 7)  # Parcela 2 com média 30 cm e desvio
 ↪  padrão 7 cm
parcela3 <- rnorm(n_arvores, mean = 35, sd = 6)  # Parcela 3 com média 35 cm e desvio
 ↪  padrão 6 cm

# Criar um data frame com os dados das parcelas
dados_inventario <- data.frame(
  Parcela = rep(c("1", "2", "3"), each = n_arvores),
  Diametro = c(parcela1, parcela2, parcela3)
)
```

3.1.1.2 Construindo os histogramas

```
# Definir as classes diamétricas com intervalo de 2 cm
intervalo <- 2
min_diametro <- floor(min(dados_inventario$Diametro))  # Valor mínimo de diâmetro
 ↪  arredondado para baixo
```

```r
max_diametro <- ceiling(max(dados_inventario$Diametro))  # Valor máximo de diâmetro
 ↪  arredondado para cima
intervalos <- seq(min_diametro - intervalo, max_diametro + intervalo, by = intervalo)
 ↪  # Definir as classes com intervalo de 2 cm

# Criar layout para os gráficos
par(mfrow = c(1, 3))  # Define 3 gráficos em uma linha

# Definir lista de parcelas
parcelas <- unique(dados_inventario$Parcela)

# Loop para plotar histogramas de cada parcela
for (parcela in parcelas) {
  # Selecionar os dados da parcela atual
  dados_parcela <- dados_inventario[dados_inventario$Parcela == parcela, "Diametro"]
  # Plotar o histograma
  hist(dados_parcela,
       breaks = intervalos,
       main = paste("Histograma - parcela ", parcela),
       xlab = "Diâmetro (cm)",
       ylab = "Frequência",
       ylim = c(0, 10),
       col = "skyblue",
       border = "black")
}
```



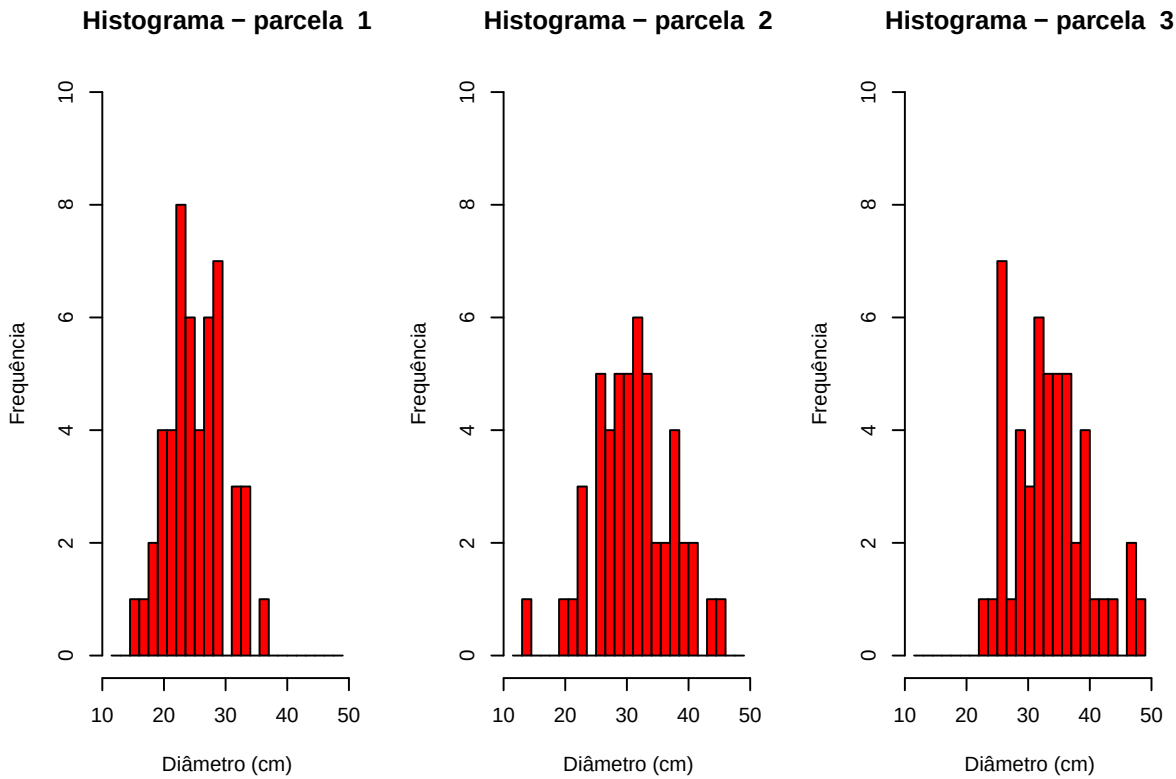
3.1.1.3 Exercício

Faça o mesmo gráfico com intervalos de 1,5 cm e mude a cor das barras para vermelho.

Ver a Solução

Solução do problema

```
par(mfrow = c(1, 3))  # Define 3 gráficos em uma linha

# Definir as classes diamétricas com intervalo de 2 cm
intervalo <- 1.5
intervalos <- seq(min_diametro - intervalo, max_diametro + intervalo, by =
↪  intervalo)

# Loop para plotar histogramas de cada parcela
for (parcela in parcelas) {
  # Selecionar os dados da parcela atual
  dados_parcela <- dados_inventario[dados_inventario$Parcela == parcela, "Diametro"]
  # Plotar o histograma
  hist(dados_parcela,
       breaks = intervalos,
       main = paste("Histograma - parcela ", parcela),
       xlab = "Diâmetro (cm)",
       ylab = "Frequência",
       ylim = c(0, 10),
       col = "red",
       border = "black")
}
```



Capítulo 4

Inventário florestal

4.1 Fitossociologia (Teoria)

Os engenheiros florestais realizam estudos de fitossociologia para compreender a composição, a estrutura e as interações das comunidades vegetais em um determinado ecossistema. Esses estudos permitem identificar espécies dominantes, associadas e raras, além de avaliar a biodiversidade e as dinâmicas ecológicas do ambiente. Essas informações são essenciais para planejar o manejo sustentável, recuperar áreas degradadas e conservar ecossistemas naturais.

Os engenheiros florestais realizam estudos de fitossociologia para determinar tipologias vegetais e fitofisionomias, aspectos fundamentais em processos como a Autorização para Exploração Vegetal (AUTEX) e os Programas de Recuperação Ambiental (PRA).

4.1.1 Estrutura horizontal

4.1.1.1 Densidade absoluta

Refere-se ao número de indivíduos de uma espécie i (n_i) por unidade de área (A), expressando a abundância total da espécie na comunidade.

$$DA_i = \frac{n_i}{A}$$

4.1.1.2 Densidade relativa

É a proporção da densidade absoluta de uma espécie (DA_i) em relação à soma das densidades absolutas de todas as espécies, expressa em porcentagem. Indica a importância relativa da espécie em termos de abundância.

$$DR_i = \frac{DA_i}{\sum_{i=1}^{s}(DA_i)}$$

4.1.1.3 Dominância absoluta

Calcula a área basal total (G_i) de todos os indivíduos de uma espécie por unidade de área (A). Mede o espaço físico ocupado por uma espécie no ambiente.

$$DoA_i = \frac{\sum_{j=1}^{n_i} g_j}{A} = \frac{G_i}{A}$$

4.1.1.4 Dominância relativa

Representa a proporção da dominância absoluta de uma espécie (DoA_i) em relação à dominância total (G_T) de todas as espécies, expressa em porcentagem. Indica a importância relativa em termos de área ocupada.

$$DoR_i = \frac{DoA_i}{\sum_{i=1}^{S}(DoA_i)} * 100 = \frac{G_i}{G_T} * 100$$

4.1.1.5 Frequência Absoluta

É a relação entre o número de unidades amostrais onde a espécie está presente (U_i) e o total de unidades amostrais (U_T), expressa em porcentagem. Mede a distribuição da espécie no espaço amostrado.

$$FA_i = \frac{U_i}{U_T} * 100$$

4.1.1.6 Frequência Relativa

Corresponde à frequência absoluta de uma espécie (FA_i) em relação à soma das frequências absolutas de todas as espécies, expressa em porcentagem. Avalia a importância espacial relativa da espécie.

$$FR_i = \frac{FA_i}{\sum_{i=1}^{S}(FA_i)} * 100$$

4.1.1.7 Valor de Cobertura

É a soma da densidade relativa (DR_i), da dominância relativa (DoR_i) e, em alguns casos, da frequência relativa (FR_i), dependendo da metodologia. Reflete a contribuição total de uma espécie na comunidade.

$$VC_i = DR_i + DoR_i + FR_i$$

4.1.1.8 Porcentagem de Cobertura (Horizontal)

É a média ponderada de densidade relativa, dominância relativa e, opcionalmente, frequência relativa. A interpretação pode variar entre o cálculo bidimensional ($DR+DoR$) ou tridimensional ($DR+DoR+FR$), dependendo do objetivo.

$$PC_i = \frac{DR_i + DoR_i + FR_i}{3}$$

4.1.2 Diversidade

4.1.2.1 Índice de Shannom

Quantifica a diversidade da comunidade, considerando tanto a abundância (p_i) quanto a equitabilidade das espécies. Valores maiores indicam maior diversidade e distribuição equitativa entre as espécies.

$$H' = -\sum_{i=1}^{S} p_i \ln(p_i)$$

4.1.3 Agregação

4.1.3.1 Índice de Morisita

Mede o padrão de distribuição espacial das espécies. Valores $I_\delta > 1$ indicam agregação, $I_\delta = 1$ distribuição ao acaso, e $I_\delta < 1$ distribuição uniforme.

$$I_\delta = \frac{n \sum_{i=1}^{S} n_i(n_i - 1)}{N(N-1)}$$

4.1.4 Estrutura vertical

4.1.4.1 Posição Sociológica Absoluta

Representa a distribuição vertical de uma espécie na comunidade, considerando a contribuição proporcional das classes de altura.

$$PSA_i = \sum_{j=1}^{J} \left(\frac{N_j}{N} \cdot N_{ij} \right)$$

4.1.4.2 Posição Sociológica Relativa

É a proporção da posição sociológica absoluta de uma espécie (PSA_i) em relação à soma das posições sociológicas absolutas de todas as espécies, expressa em porcentagem. Indica a importância relativa da espécie na estrutura vertical.

$$PSR_i = \frac{PSA_i}{\sum_{i=1}^{S} PSA_i} \cdot 100$$

4.1.4.3 Valor de Importância Absoluta (Horizontal + Vertical)

Combina os índices horizontais (densidade relativa, dominância relativa e frequência relativa) com a posição sociológica absoluta para refletir a contribuição global de uma espécie na comunidade.

$$VI_a = DR_i + DoR_i + FR_i + PSA_i$$

4.1.4.4 Valor de Importância Relativa (Horizontal + Vertical)

É a média ponderada dos índices horizontais e verticais. Indica a importância relativa de uma espécie considerando a estrutura horizontal e vertical.

$$VI_r = \frac{DR_i + DoR_i + FR_i + PSA_i}{4}$$

4.1.5 Regeneração

Pode-se usar os mesmos índices anteriores de estrutura horizontal, vertical, etc. O resultado final será o Indicador de Regeneração Natural que poderá ser usado para calcular o **Valor de Importância Ampliado**:

$$VIA_r = \frac{DR_i + DoR_i + FR_i + PSA_i + RN_i}{5}$$

4.2 Fitossociologia (Aplicação)

De acordo com o estudo realizado por Verly et al. (2020), a caracterização florística defina para a Reserva Legal do IFMT foi de um fragmento Cerradão em estado de conservação adequada. De acordo com [Otávio Miranda Verly, 2020], a caracterização florística...

Dados exemplo: Baixar dados

Parc	Subparc	Id	Nome Científico	Família	H T	HC	CAP	DAP
1	1	1	Myracrodruon urundeuva Allemão	Anacardiaceae R.Br.	11.5	6	84.5	26.897185382530314
1	1	2	SP	-	-	-	-	-
1	1	3	Tabebuia roseoalba (Ridl.) Sandwith	Bignoniaceae Juss.	10.5	5	54	17.188733853924695
1	1	4	Platypodium elegans Vogel	Fabaceae Lindl.	11.5	8.5	57.6	18.334649444186343
1	1	5	Callisthene fasciculata Mart.	Vochysiaceae A.St.-Hil.	8.5	4.5	37.700000000000003	12.000282709128909
1	1	6	Magonia pubescens A.St.-Hil.	Sapindaceae Juss.	11.5	8	94.7	30.143946221604978
1	1	7	Morta	-	-	-	-	-
1	1	8	Pseudobombax tomentosum (Mart. & Zucc.) A.Robyns	Malvaceae Juss.	11	6	108.5	34.53662265094129
1	1	9	Platypodium elegans Vogel	Fabaceae Lindl.	10.5	6.5	56	17.82535362629228
1	1	10	Aspidosperma cylindrocarpon Müll.Arg.	Apocynaceae Juss.	8	6	36.4	11.58647985708998
1	1	11	Callisthene fasciculata Mart.	Vochysiaceae A.St.-Hil.	8.5	6	37.799999999999997	12.032113697747286
1	1	12	Tabebuia roseoalba (Ridl.) Sandwith	Bignoniaceae Juss.	8	6	41.5	13.209860276627314
1	1	13	Callisthene fasciculata Mart.	Vochysiaceae A.St.-Hil.	10	7	44	14.00563499208679
1	1	14	Aspidosperma cylindrocarpon Müll.Arg.	Apocynaceae Juss.	7.5	4	31.6	10.058592403407786
1	1	15	Aspidosperma subincanum Mart.	Apocynaceae Juss.	12	4	79	25.146481008519466
1	1	16	Aspidosperma cylindrocarpon Müll.Arg.	Apocynaceae Juss.	12	5.5	55.6	17.698029671818762
1	1	17	Platypodium elegans Vogel	Fabaceae Lindl.	11	8.5	32.5	10.345071300973197
1	1	18	Anadenanthera colubrina (Vell.) Brenan	Fabaceae Lindl.	13	9.5	58.8	18.716621307606893
1	1	19	Aspidosperma cylindrocarpon Müll.Arg.	Apocynaceae Juss.	15	9	106	33.74084793548181
1	1	20	Platypodium elegans Vogel	Fabaceae Lindl.	7.5	2.5	32	10.185916357881302
1	1	21	Astronium fraxinifolium Schott	Anacardiaceae R.Br.	9.5	7.5	37.299999999999997	11.872958754655391
1	1	22	Simarouba Aubl.	Simaroubaceae DC.	9.5	7	43.5	13.846480048994895
1	1	23	Morta	-	-	-	-	-
1	1	24	Astronium fraxinifolium Schott	Anacardiaceae R.Br.	10	9	42.6	13.560001151429484
1	1	25	Platypodium elegans Vogel	Fabaceae Lindl.	11.5	5.5	50	15.915494309189533
1	1	26	Callisthene fasciculata Mart.	Vochysiaceae A.St.-Hil.	11	5.5	88.7	28.234086904502234
1	1	27	Handroanthus impetiginosus (Mart. ex DC.) Mattos	Bignoniaceae Juss.	11	6	70.400000000000006	22.409015987338865
1	1	28	Platypodium elegans Vogel	Fabaceae Lindl.	11.5	8	52.5	16.71126902464901
1	1	29	Machaerium acutifolium Vogel	Fabaceae Lindl.	10.5	6.5	38.4	12.223099629457561
1	1	30	Platypodium elegans Vogel	Fabaceae Lindl.	10	7	31.5	10.026761414789407
1	1	31	Handroanthus impetiginosus (Mart. ex DC.) Mattos	Bignoniaceae Juss.	11	8	77.5	24.669016179243776
1	1	32	Senegalia polyphylla (DC.) Britton & Rose	Fabaceae Lindl.	11.5	9	33.5	10.663381187156988
1	1	33	Aspidosperma cylindrocarpon Müll.Arg.	Apocynaceae Juss.	9	5.5	38.700000000000003	12.318592595312701
1	1	34	Platypodium elegans Vogel	Fabaceae Lindl.	9	7.5	34.5	10.981691073340778
1	1	35	Callisthene fasciculata Mart.	Vochysiaceae A.St.-Hil.	9.5	6	45.4	14.451268832744097
1	1	36	Qualea Aubl.	Vochysiaceae A.St.-Hil.	10.5	6	66.3	21.103945453985322
1	1	37	Qualea Aubl.	Vochysiaceae A.St.-Hil.	10	6	55.4	17.634367694582004
1	1	38	Callisthene fasciculata Mart.	Vochysiaceae A.St.-Hil.	10.5	4.5	55.5	17.666198683200381
1	1	39	Callisthene fasciculata Mart.	Vochysiaceae A.St.-Hil.	12	7	79	25.146481008519466
1	1	40	Myracrodruon urundeuva Allemão	Anacardiaceae R.Br.	13	6.5	106.7	33.963664855810464
1	1	41	Dipteryx alata Vogel	Fabaceae Lindl.	13.5	3.5	96.6	30.748735005354177
1	1	42	Platypodium elegans Vogel	Fabaceae Lindl.	12	8.5	92.5	29.443664472000638
1	1	43	Callisthene fasciculata Mart.	Vochysiaceae A.St.-Hil.	11	7	62	19.735212943395023
1	1	44	Callisthene fasciculata Mart.	Vochysiaceae A.St.-Hil.	10	7	42.5	13.528170162811104
1	1	45	Aspidosperma subincanum Mart.	Apocynaceae Juss.	13.5	8.5	70.099999999999994	22.313523021483725
1	1	46	Callisthene fasciculata Mart.	Vochysiaceae A.St.-Hil.	11.5	7	65.5	20.84929754503829
1	1	47	Anadenanthera colubrina (Vell.) Brenan	Fabaceae Lindl.	13.5	10	62.6	19.926198875105296
1	2	91	Qualea Aubl.	Vochysiaceae A.St.-Hil.	10.5	7.5	61.1	19.44873404582961
1	2	90	Platypodium elegans Vogel	Fabaceae Lindl.	11	7.5	37.5	11.93662073189215
1	2	89	Dipteryx alata Vogel	Fabaceae Lindl.	12.5	8	73.400000000000006	23.363945645890237
1	2	88	Platypodium elegans Vogel	Fabaceae Lindl.	9	7	32.799999999999997	10.440564266828334
1	2	87	Qualea Aubl.	Vochysiaceae A.St.-Hil.	10	7.5	40	12.732395447351628
1	2	86	Dipteryx alata Vogel	Fabaceae Lindl.	13	6	118.1	37.593297558905677

4.2.1 Consistência dos dados

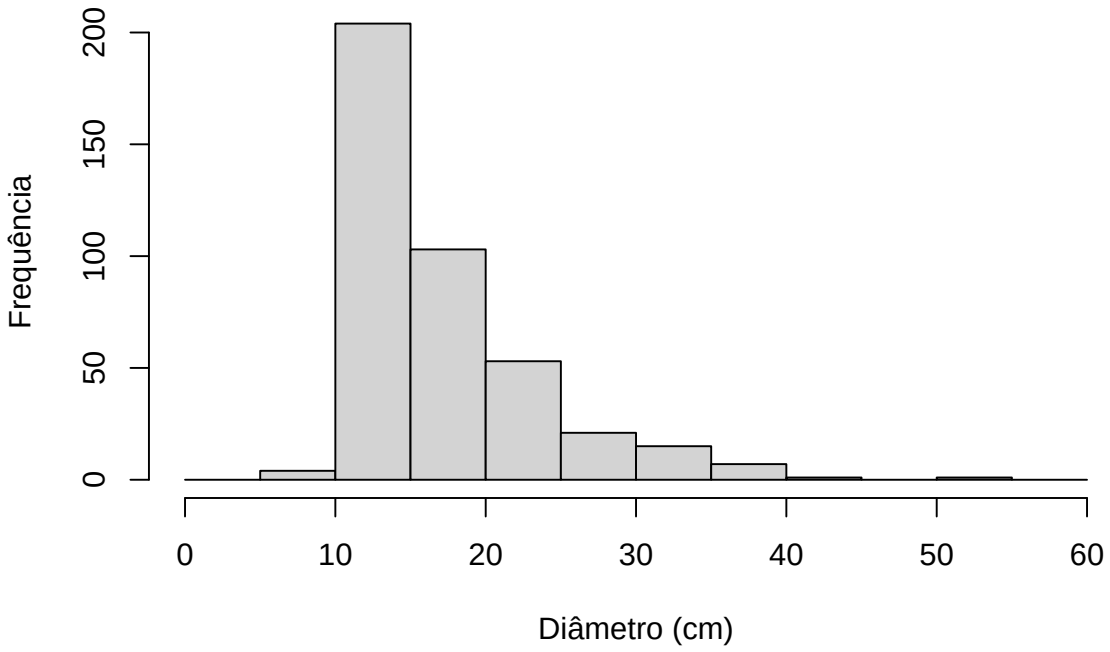
Antes de qualquer análise, é fundamental verificar a consistência dos dados coletados. Isso envolve identificar e corrigir possíveis erros ou inconsistências, garantindo a qualidade e a confiabilidade das informações para as análises subsequentes.

```
items_excluir <- c("SP", "Morta", "PP", "PS", "Repetida", "Repitida", "Morfoespécie
 ↪  7", "Morfoespécie 8")
BD_ <- BD[!BD$`Nome Científico` %in% items_excluir, ]

BD_$DAP <- as.numeric(BD_$DAP)
BD_$`H T` <- as.numeric(BD_$`H T`)
```

4.2.2 Distribuição diamétrica geral

A análise da distribuição dos diâmetros das árvores fornece insights sobre a estrutura etária e o desenvolvimento da floresta. Uma distribuição equilibrada indica uma regeneração contínua, enquanto a predominância de árvores em determinadas classes diamétricas pode sugerir distúrbios ou práticas de manejo específicas.

```
intervalo <- 5
max_d <- ceiling(max(BD_$DAP, na.rm = TRUE))
intervalos <- seq(0, max_d + intervalo, by = intervalo)
hist(BD_$DAP, breaks = intervalos, main = NULL, xlab = "Diâmetro (cm)", ylab =
 ↪  "Frequência")
```

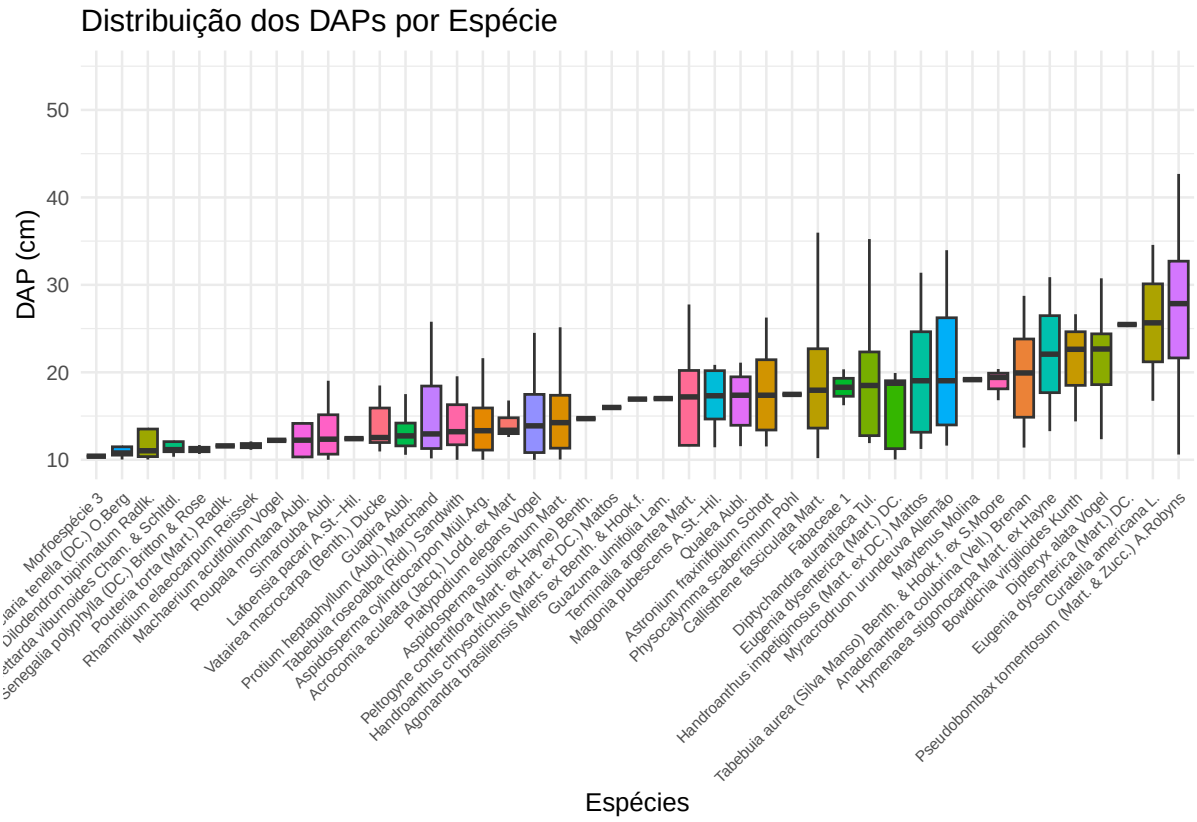


4.2.3 Distribuição por espécies (boxplots)

Os boxplots permitem visualizar a variação dos diâmetros por espécie, destacando tendências e identificando outliers. Essa análise auxilia na compreensão do crescimento e do desenvolvimento de cada espécie, bem como na identificação de espécies que possam necessitar de atenção especial no manejo.

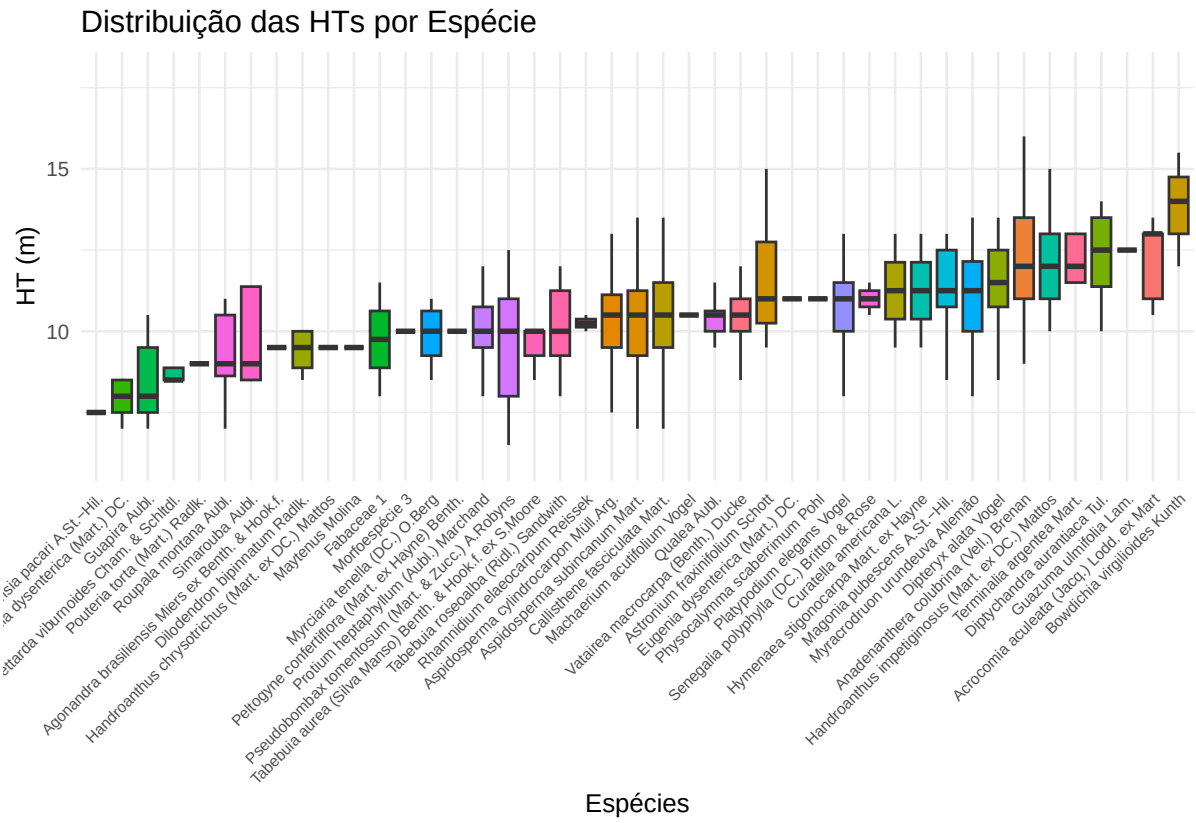
4.2.3.1 Boxplot dos DAPs

```
require(ggplot2)
especies <- unique(BD_$`Nome Científico`)

p <- ggplot(BD_, aes(x = reorder(`Nome Científico`, DAP, median), y = DAP, fill =
 ↪  `Nome Científico`)) +
  geom_boxplot(outlier.shape = NA) +  # Boxplot sem os outliers
  theme_minimal(base_size = 10) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 6),
    legend.position = "none"  # Remover legenda, se preferir
  ) +
  labs(title = "Distribuição dos DAPs por Espécie",
       x = "Espécies", y = "DAP (cm)")

print(p)
```



4.2.3.2 boxplot das alturas totais

```
p <- ggplot(BD_, aes(x = reorder(`Nome Científico`, `H T`, median), y = `H T`, fill =
 ↪  `Nome Científico`)) +
  geom_boxplot(outlier.shape = NA) +  # Boxplot sem os outliers
  theme_minimal(base_size = 10) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 6),
    legend.position = "none"  # Remover legenda, se preferir
  ) +
  labs(title = "Distribuição das HTs por Espécie",
       x = "Espécies", y = "HT (m)")

print(p)
```



4.2.4 Parâmetros Fitossociológicos

A aplicação prática dos parâmetros fitossociológicos, como densidade, dominância e frequência, fornece uma visão detalhada da composição e da estrutura da comunidade vegetal. Esses indicadores são essenciais para o planejamento de ações de manejo, conservação e recuperação de áreas florestais.

4.2.4.1 Estrutura horizontal

```
EH <- function(species, sample, d, A) {
    DT <- data.table(species = species, sample = sample, d = d)
    DT <- DT[, `:=`(gi = pi * d^2 / 40000)]
    Ui <- unique(DT, by = c("species", "sample"))[, .(Ui = .N), by = "species"]
    ni <- DT[, .(ni = .N, Gi = sum(gi)), by = "species"]
    ni <- ni[Ui, on = "species"]
```

```
    ni[, DAi := ni / A][, DRi := (DAi / sum(DAi)) * 100]
    ni[, DoAi := Gi / A][, DoRi := (DoAi / sum(DoAi)) * 100]
    ni[, FAi := (Ui / length(unique(DT$sample))) * 100][, FRi := (FAi / sum(FAi)) *
↪   100]

    num_cols <- names(ni)[sapply(ni, is.numeric)]
    ni[, (num_cols) := lapply(.SD, round, 4), .SDcols = num_cols]

    setnames(ni, old = "species", new = "Espécies")

    return(ni)
}

EH_result <- EH(BD_$`Nome Científico`, BD_$Parc, BD_$DAP, A = 0.8)
```

Resultados:

Espécies	ni	Gi	Ui	DAi	DRi	DoAi	DoRi	FAi	FRi
Myracrodruon urundeuva Allemão	18	0.6704	2	22.50	4.4010	0.8380	6.3106	100	2.9412
Tabebuia roseoalba (Ridl.) Sandwith	23	0.3590	2	28.75	5.6235	0.4487	3.3793	100	2.9412
Platypodium elegans Vogel	35	0.6419	2	43.75	8.5575	0.8024	6.0427	100	2.9412
Callisthene fasciculata Mart.	41	1.2693	2	51.25	10.0244	1.5866	11.9479	100	2.9412
Magonia pubescens A.St.-Hil.	12	0.3156	2	15.00	2.9340	0.3945	2.9711	100	2.9412
Pseudobombax tomentosum (Mart. & Zucc.) A.Robyns	15	0.9521	2	18.75	3.6675	1.1901	8.9621	100	2.9412
Aspidosperma cylindrocarpon Müll.Arg.	60	1.3732	2	75.00	14.6699	1.7165	12.9260	100	2.9412
Aspidosperma subincanum Mart.	16	0.3735	2	20.00	3.9120	0.4669	3.5162	100	2.9412
Anadenanthera colubrina (Vell.) Brenan	17	0.5402	2	21.25	4.1565	0.6753	5.0851	100	2.9412
Astronium fraxinifolium Schott	15	0.4479	2	18.75	3.6675	0.5598	4.2159	100	2.9412
Simarouba Aubl.	4	0.0606	1	5.00	0.9780	0.0758	0.5706	50	1.4706
Handroanthus impetiginosus (Mart. ex DC.) Mattos	11	0.3690	2	13.75	2.6895	0.4612	3.4733	100	2.9412
Machaerium acutifolium Vogel	1	0.0117	1	1.25	0.2445	0.0147	0.1105	50	1.4706
Senegalia polyphylla (DC.) Britton & Rose	2	0.0196	1	2.50	0.4890	0.0246	0.1850	50	1.4706
Qualea Aubl.	12	0.2719	2	15.00	2.9340	0.3399	2.5594	100	2.9412
Dipteryx alata Vogel	11	0.4873	2	13.75	2.6895	0.6092	4.5874	100	2.9412
Guapira Aubl.	9	0.1251	2	11.25	2.2005	0.1563	1.1772	100	2.9412
Peltogyne confertiflora (Mart. ex Hayne) Benth.	1	0.0170	1	1.25	0.2445	0.0212	0.1599	50	1.4706

4.2.4.2 Índice de Shannon (Diversidade)

Na aplicação, o cálculo do índice de Shannon avalia a diversidade alfa da comunidade vegetal. Um valor alto reflete maior diversidade e distribuição equitativa entre as espécies no local estudado.

```
shannon <- function(species_counts) {
  pi <- species_counts / sum(species_counts)
  -sum(pi * log(pi), na.rm = TRUE)
}

H <- shannon(EH_result$ni)
cat("Índice de Shannon:", H, "\n")
```

```
## Índice de Shannon: 3.16804
```

4.2.4.3 Índice de Morisita (Agregação)

Mede o padrão de distribuição espacial das espécies. Valores $I_\delta > 1$ indicam agregação.

```
morisita <- function(species_counts, parcelas) {
  # Agregar os indivíduos por parcela
  counts <- tapply(species_counts, parcelas, sum)

  N <- sum(counts)       # Total de indivíduos em todas as parcelas
  n <- length(counts)    # Número de parcelas
  numerator <- n * sum(counts * (counts - 1))  # Soma dos xi(xi - 1) por parcela
  denominator <- N * (N - 1)                   # Total de combinações possíveis
  I_M <- numerator / denominator
  return(I_M)
}

# Exemplo de aplicação:
# Substituir species_counts e parcelas pelos dados reais
species_counts <- BD_$DAP[!is.na(BD_$DAP)]  # DAP como proxy de presença/indivíduo
parcelas <- BD_$Parc                        # Número da parcela correspondente
I_delta <- morisita(species_counts, parcelas)
cat("Índice de Morisita corrigido:", I_delta, "\n")
```

```
## Índice de Morisita corrigido: 1.022617
```

4.2.4.4 Estrutura vertical e classificação em estratos

A inclusão da dimensão vertical nos parâmetros fitossociológicos agrega uma visão tridimensional da comunidade florestal, indo além da análise convencional baseada apenas na densidade e dominância (horizontal).

```
BD_$`H T` <- as.numeric(BD_$`H T`)
meanH <- mean(BD_$`H T`, na.rm = TRUE)
sdH <- sd(BD_$`H T`, na.rm = TRUE)

# Calcular estrato
BD_$estrato <- case_when(
  BD_$`H T` < (meanH - sdH) ~ "Inferior",
  BD_$`H T` >= (meanH - sdH) & BD_$`H T` <= (meanH + sdH) ~ "Médio",
  BD_$`H T` > (meanH + sdH) ~ "Superior"
```

```
)

# Calcular PSAi e PSRi
resultados <- BD_ %>%
  group_by(estrato) %>%
  mutate(
    Nj = n(),                      # Total de indivíduos no estrato
    N = nrow(BD_),                 # Total de indivíduos na floresta
    Pj = Nj / N                    # Peso do estrato no total
  ) %>%
  group_by(`Nome Científico`, estrato) %>%
  summarise(
    Nji = n(),                     # Número de indivíduos da espécie no estrato
    Nj = first(Nj),                # Total de indivíduos no estrato
    Pj = first(Pj),                # Peso do estrato
    PSAi_partial = (Nji / Nj) * Pj, # Contribuição parcial para PSAi
    .groups = "drop"
  ) %>%
  group_by(`Nome Científico`) %>%
  summarise(
    PSAi = sum(PSAi_partial, na.rm = TRUE)        # Soma das contribuições parciais
     ↪  para PSAi
  ) %>%
  ungroup() %>%
  mutate(
    PSRi = (PSAi / sum(PSAi, na.rm = TRUE)) * 100 # PSRi como porcentagem global
  )

# Fusão das tabelas com left_join
EH_result <- EH_result %>%
  left_join(resultados, by = c("Espécies" = "Nome Científico"))

# Adicionando o Valor de Importância Ampliado (VIA)
required_cols <- c("DAi", "DoAi", "FAi", "PSAi", "DRi", "DoRi", "FRi", "PSRi")
missing_cols <- required_cols[!required_cols %in% names(EH_result)]

if (length(missing_cols) > 0) {
  stop(paste("Colunas ausentes em `EH_result`: ", paste(missing_cols, collapse = ",
   ↪  ")))
}

EH_result <- EH_result %>%
  mutate(
    VIAa = (DAi + DoAi + FAi + PSAi) / 4,  # Cálculo do VIA absoluto
  )

# Visualizar os resultados finais com arredondamento
resultado_final <- EH_result %>%
  arrange(desc(VIAa)) %>%
  mutate(across(where(is.numeric), round, 4))
```

Espécies	ni	Gi	Ui	DAi	DRi	DoAi	DoRi	FAi	FRi	PSAi	PSRi	VIAa
Aspidosperma cylindrocarpon Müll.Arg.	60	1.3732	2	75.00	14.6699	1.7165	12.9260	100	2.9412	0.1467	14.6699	44.2158
Callisthene fasciculata Mart.	41	1.2693	2	51.25	10.0244	1.5866	11.9479	100	2.9412	0.1002	10.0244	38.2342
Platypodium elegans Vogel	35	0.6419	2	43.75	8.5575	0.8024	6.0427	100	2.9412	0.0856	8.5575	36.1595
Protium heptaphyllum (Aubl.) Marchand	23	0.4445	2	28.75	5.6235	0.5556	4.1837	100	2.9412	0.0562	5.6235	32.3405
Tabebuia roseoalba (Ridl.) Sandwith	23	0.3590	2	28.75	5.6235	0.4487	3.3793	100	2.9412	0.0562	5.6235	32.3137
Myracrodruon urundeuva Allemão	18	0.6704	2	22.50	4.4010	0.8380	6.3106	100	2.9412	0.0440	4.4010	30.8455
Anadenanthera colubrina (Vell.) Brenan	17	0.5402	2	21.25	4.1565	0.6753	5.0851	100	2.9412	0.0416	4.1565	30.4917
Aspidosperma subincanum Mart.	16	0.3735	2	20.00	3.9120	0.4669	3.5162	100	2.9412	0.0391	3.9120	30.1265
Pseudobombax tomentosum (Mart. & Zucc.) A.Robyns	15	0.9521	2	18.75	3.6675	1.1901	8.9621	100	2.9412	0.0367	3.6675	29.9942
Astronium fraxinifolium Schott	15	0.4479	2	18.75	3.6675	0.5598	4.2159	100	2.9412	0.0367	3.6675	29.8366
Vatairea macrocarpa (Benth.) Ducke	13	0.2176	2	16.25	3.1785	0.2720	2.0485	100	2.9412	0.0318	3.1785	29.1384
Diptychandra aurantiaca Tul.	12	0.3926	2	15.00	2.9340	0.4908	3.6957	100	2.9412	0.0293	2.9340	28.8800
Magonia pubescens A.St.-Hil.	12	0.3156	2	15.00	2.9340	0.3945	2.9711	100	2.9412	0.0293	2.9340	28.8560
Qualea Aubl.	12	0.2719	2	15.00	2.9340	0.3399	2.5594	100	2.9412	0.0293	2.9340	28.8423
Dipteryx alata Vogel	11	0.4873	2	13.75	2.6895	0.6092	4.5874	100	2.9412	0.0269	2.6895	28.5965
Handroanthus impetiginosus (Mart. ex DC.) Mattos	11	0.3690	2	13.75	2.6895	0.4612	3.4733	100	2.9412	0.0269	2.6895	28.5595
Guapira Aubl.	9	0.1251	2	11.25	2.2005	0.1563	1.1772	100	2.9412	0.0220	2.2005	27.8571
Acrocomia	7	0.1052	2	8.75	1.7115	0.1314	0.9899	100	2.9412	0.0171	1.7115	27.2246

Na aplicação prática, a utilização do VIA, que combina parâmetros horizontais e verticais com indicadores de regeneração, fornece uma visão holística da importância das espécies. Isso é fundamental para identificar não apenas as espécies dominantes atuais, mas também aquelas com maior potencial de contribuir para a sustentabilidade futura do ecossistema.

Capítulo 5

Manejo de Florestas Plantadas

5.1 Altura Dominante

A altura dominante, que representa a média das alturas das árvores mais altas e/ou mais grossas de um determinado número de árvores por hectare, é menos influenciada por variações de densidade e competições locais entre árvores menores, tornando-a uma medida confiável da qualidade do sítio. Com ela, é possível estimar o potencial de crescimento da floresta, modelar curvas de crescimento e projetar a produção futura, auxiliando no planejamento sustentável e na tomada de decisões estratégicas no manejo florestal.

Dados exemplo: Baixar dados

5.1.1 Calculando altura dominante por parcela (Assman)

```
BD <- read.csv2("data/DesafioCalcHdom.csv")
colnames(BD)[1] = c("areaParc")
BD <- BD[, 1:5]
BD <- BD[order(BD$Parc, BD$DAP, decreasing = TRUE),]
row.names(BD) <- NULL

Parc <- unique(BD$Parc)
Hdom_assman <- unique(BD$Parc)

j <- 1
for (i in Parc) {
  a <- ceiling(mean(BD[which(BD$Parc==i),1])*100/10000)
  H <- BD[which(BD$Parc==i),5]
  Hdom_assman[j] <- round(mean(H[1:a]), 2)
  j <- j+1
}
```

A.Parc.	Parc	n	DAP	Ht	Hdom
720	711	1	19.74	14.65	NA
720	711	2	NA	NA	NA
720	711	3	19.19	14.44	NA
720	711	4	17.83	13.89	NA
720	711	5	18.46	14.15	NA
720	711	6	20.05	14.78	NA

Parc	Hdom_assman
711	14.56
623	13.99
622	13.49
621	14.60
534	14.98
533	14.59
532	14.74
531	14.93
453	14.49
452	14.50
451	14.50
344	15.04
343	14.44
342	14.76
341	14.92
262	13.25
261	14.30
173	14.10
172	14.22
171	14.80

```
BD2 <- as.data.frame(cbind(Parc, Hdom_assman))
BD <- merge(BD, BD2)
```

Resultado:

5.1.1.1 Exercício

Faça o cálculo da altura dominante agora seguindo o conceito de Hart: 100 árvores mais altas por hectare. Tente usar o pipe para resolver o problema.

Ver a Solução

Solução do problema sem o pipe

```
BD <- read.csv2("data/DesafioCalcHdom.csv")
colnames(BD)[1] = c("areaParc")
BD <- BD[, 1:5]
BD <- BD[order(BD$Parc, BD$Ht, decreasing = TRUE),]
row.names(BD) <- NULL

Parc <- unique(BD$Parc)
Hdom_hart <- unique(BD$Parc)

j <- 1
for (i in Parc) {
  a <- ceiling(mean(BD[which(BD$Parc==i),1])*100/10000)
  H <- BD[which(BD$Parc==i),5]
  Hdom_hart[j] <- round(mean(H[1:a]), 2)
  j <- j+1
}

BD2 <- as.data.frame(cbind(Parc, Hdom_hart))
BD <- merge(BD, BD2)
```

Parc	Hdom_hart
711	14.56
623	13.99
622	13.49
621	14.60
534	14.98
533	14.59
532	14.74
531	14.93
453	14.49
452	14.50
451	14.50
344	15.04
343	14.44
342	14.76
341	14.92
262	13.25
261	14.30
173	14.10
172	14.22
171	14.80

Resultado:

Solução com pipe

```
library(dplyr)

BD <- BD %>%
  arrange(Parc, desc(Ht))  # Garantir a ordem correta

BD_Hart <- BD %>%
  group_by(Parc) %>%
  mutate(
    num_top = ceiling(mean(areaParc) * 100 / 10000)
  ) %>%
  slice(1:first(num_top)) %>%
  summarise(
    Hdom_Hart2 = round(mean(Ht, na.rm = TRUE), 2)
  ) %>%
  ungroup()

BD_HDOM <- merge(BD2, BD_Hart, by = "Parc")
```

Resultado:

Parc	Hdom_hart	Hdom_Hart2
171	14.80	14.80
172	14.22	14.22
173	14.10	14.10
261	14.30	14.30
262	13.25	13.25
341	14.92	14.92
342	14.76	14.76
343	14.44	14.44
344	15.04	15.04
451	14.50	14.50
452	14.50	14.50
453	14.49	14.49
531	14.93	14.93
532	14.74	14.74
533	14.59	14.59
534	14.98	14.98
621	14.60	14.60
622	13.49	13.49
623	13.99	13.99
711	14.56	14.56

Referências

Referências Bibliográficas

et al. Otávio Miranda Verly. Caracterização florística e fitossociologia de um fragmento de cerradão em cáceres, mato grosso. *Pesquisa Florestal Brasileira*, 40:e201801742, 2020.

Hadley Wickham and contributors. *Tidyverse Style Guide*. RStudio, PBC, 2021. URL https://style.tidyverse.org/. Acesso em: 18 nov. 2024.