Apostila do minicurso de Noções básicas de linguagem R aplicadas à biologia aquática

2023-11-16

O que é a Linguagem R

O R é um ambiente de software livre para computação estatística e Gráficos. É um ambiente de análise estatística amplamente utilizados em diversas áreas, incluindo as ciências biológicas. Criada por Ross Ihaka e Robert Gentleman na década de 1990 e tem ganhado destaque como uma das ferramentas mais populares para cientistas de dados, estatísticos e analistas de dados em todo o mundo. Ela se destaca por sua flexibilidade e extensibilidade, o que a torna uma escolha popular para a análise de dados complexos e a geração de resultados confiáveis. A ferramenta oferece uma rica coleção de pacotes e funções que permitem aos cientistas de dados e pesquisadores realizar uma ampla variedade de tarefas, desde análises estatísticas avançadas até a criação de gráficos personalizados. Sua comunidade ativa e colaborativa contribui para o constante desenvolvimento de recursos e pacotes. Como uma linguagem de programação, o R permite automatizar tarefas, escrever scripts personalizados e criar fluxos de trabalho de análise de dados eficientes. Ele também é capaz de lidar com grandes conjuntos de dados e é uma ferramenta poderosa para a exploração, visualização e interpretação de informações em ciências biológicas e outras áreas. Com uma comunidade ativa de desenvolvedores que criam pacotes e extensões para atender a diversas necessidades, o R oferece uma plataforma poderosa para explorar, modelar e comunicar informações a partir de dados, tornando-o uma escolha valiosa para profissionais que trabalham com análise de dados e estatísticas.

Vantagens de Aprender o R Aprender a Linguagem R oferece uma série de vantagens significativas, especialmente para profissionais e estudantes em ciências biológicas. Aqui estão algumas das principais vantagens:

- **Flexibilidade e Extensibilidade:** R é uma linguagem altamente flexível, que permite realizar uma ampla variedade de tarefas relacionadas à análise de dados. Além disso, sua extensibilidade significa que você pode criar suas próprias funções e pacotes, tornando-o adequado para abordar problemas específicos das ciências biológicas.
- **Comunidade Ativa:** R possui uma comunidade ativa de desenvolvedores e usuários, o que significa que há uma grande quantidade de recursos, pacotes e suporte disponíveis. Isso facilita a resolução de problemas e o aprendizado contínuo.
- **Visualização de Dados:** R oferece recursos avançados de visualização de dados, permitindo criar gráficos informativos e publicáveis. Isso é particularmente útil em ciências biológicas, onde a representação visual de resultados é crucial.
- **Análise Estatística:** R é uma poderosa ferramenta para análise estatística. Ele inclui uma ampla gama de funções estatísticas e técnicas avançadas que são essenciais para a análise de dados em ciências biológicas.
- **Integração com Outras Ferramentas:** R é altamente integrado com outras ferramentas e linguagens de programação. Isso permite a importação e exportação de dados de várias fontes, facilitando a colaboração e o uso de diferentes recursos.
- **Código Aberto e Gratuito:** R é uma linguagem de programação de código aberto e, portanto, é gratuito para uso. Isso o torna acessível para estudantes, pesquisadores e instituições acadêmicas.

Introdução ao RStudio

Atalhos Úteis no RStudio O RStudio é uma das ferramentas mais populares e poderosas para desenvolvimento e análise de dados com a linguagem de programação R. É uma IDE (Ambiente de Desenvolvimento Integrado) projetada especificamente para atender às necessidades dos usuários do R, desde cientistas de dados e estatísticos até analistas e pesquisadores. O RStudio oferece uma interface amigável e eficiente que simplifica o processo de escrever, depurar, testar e documentar código R, além de fornecer recursos poderosos para visualização de dados, criação de relatórios e colaboração em projetos.

Através de sua interface organizada em painéis, o RStudio permite que os usuários acessem o console do R, editem scripts, gerenciem projetos, visualizem gráficos, acessem ajuda e documentação, além de oferecer controle de versão integrado e suporte a pacotes. Ele é altamente personalizável e se adapta às necessidades individuais dos usuários.

O RStudio tem se destacado como uma ferramenta essencial no universo da ciência de dados, facilitando o trabalho com o R e contribuindo para a produtividade e eficácia na análise de dados, modelagem estatística e criação de visualizações de alta qualidade. Com uma comunidade ativa de usuários e desenvolvedores, o RStudio continua evoluindo e agregando recursos para atender às demandas crescentes no campo da análise de dados. Neste contexto, explorar o RStudio é fundamental para aqueles que buscam aproveitar ao máximo a linguagem R em suas atividades profissionais e projetos de pesquisa.

Atalhos Úteis no RStudio

- `Ctrl + Enter` - Executa o comando ativo no script ou console.
- `Ctrl + Shift + C` - Comenta ou descomenta as linhas selecionadas no script.
- `Ctrl + L` - Limpa o console.
- `Tab` - Completa automaticamente funções, nomes de variáveis e argumentos.
- `F1` - Abre a documentação de ajuda para a função ou pacote em foco.
- `Ctrl +Shift + A` - Arruma o script selecionado.
- `Ctrl + Shift + M`: Ativa o operador pipe.
- `Ctrl + Shift + R`: Cria uma seção.
- `Alt + -`: Ativa o operador de atribuição.

Estes atalhos ajudam a aumentar a eficiência no RStudio e facilitam a navegação e a execução de comandos. À medida que você se familiariza com esses atalhos, você economizará tempo e poderá se concentrar mais na análise de dados em vez de navegar pela interface.

Boas Práticas no Uso da Linguagem R

O R é uma linguagem de programação `"case sensitive"`, o que significa que ele faz distinção entre maiúsculas e minúsculas em nomes de objetos, variáveis, funções e comandos. Isso tem implicações importantes no modo como você escreve código e interage com a linguagem. Aqui estão alguns pontos-chave relacionados ao fato de o R ser case sensitive:

Diferenciação entre Nomes: No R, "pirarucu" e "Arapaima" são tratados como nomes diferentes. Portanto, `pirarucu` e `Arapaima` seriam objetos diferentes. Isso significa que você deve ser consistente ao usar letras maiúsculas e minúsculas ao nomear variáveis, funções e objetos.

Chamada de Funções: Ao chamar funções, é importante corresponder exatamente ao nome da função, incluindo maiúsculas e minúsculas. Por exemplo, `mean()` é uma função diferente de `Mean()`.

Acesso a Atributos de Objetos: Quando você trabalha com objetos complexos, como data frames ou listas, a distinção entre maiúsculas e minúsculas se estende aos nomes das colunas ou elementos. Por exemplo, `especie$arapaima_gigas` e `especie$Arapaima_gigas` se referem a colunas diferentes em um data frame.

Erros Comuns: Erros de digitação em nomes de variáveis, funções ou objetos são comuns e podem ser difíceis de identificar, especialmente em scripts mais longos. A verificação de consistência nas letras maiúsculas e minúsculas é importante para evitar tais erros.

Nomeação de Variáveis Embora o R seja case sensitive, ele permite uma ampla flexibilidade na nomenclatura. Você pode escolher um estilo de nomenclatura que funcione melhor para você, seja o estilo `"snake_case"` (todas as letras minúsculas com underscores), `"camelCase"` (inicial maiúscula, sem espaços ou underscores) ou outro estilo.

Ex:

- `eu_uso_snake_case`
- `algunsUsamCamelCase`
- `outros.usam.pontos`
- `E_Alguns.naoLigamParaa_Convencao`

Operadores básicos do R O R oferece uma variedade de operadores para realizar diversas operações em valores e variáveis. Abaixo, estão alguns dos operadores mais comuns no R:

Operadores Aritméticos:

- `+`: Adição.

```
1993 + 30
```

```
## [1] 2023
```

- `-`: Subtração.

```
2023 - 1993
```

```
## [1] 30
```

- `*`: Multiplicação.

```
88 * 200
```

```
## [1] 17600
```

- `/`: Divisão.

```
303 / 2
```

```
## [1] 151.5
```

- `^` ou `**`: Exponenciação.

```
2 ^ 8
```

```
## [1] 256
```

```
2 ** 8
```

```
## [1] 256
```

Operadores de Comparação:

- `==`: Igual a.

```
2 == 2
```

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

- `!=`: Diferente de.

```
2 != 2
```

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

- `>`: Maior que.

```r
5 > 2
```

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

- `<`: Menor que.

```r
5 < 2
```

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

- `>=`: Maior ou igual a.

```r
5 >=2
```

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

- `<=`: Menor ou igual a.

```r
2 <= 5
```

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

Operadores Lógicos:

- `&` ou `&&`: E lógico (elemento a elemento e escalar).

```r
5 > 2 & 2 < 5
```

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

- `|` ou `||`: OU lógico (elemento a elemento e escalar).

```r
5 > 2 | 2 > 5
```

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

- `!`: NÃO lógico (negação).

```r
!5 < 2
```

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

Estes são alguns dos operadores mais comuns no R. Lembre-se de que a combinação adequada de operadores é fundamental para realizar operações específicas em seus dados e cálculos. A prática e a familiaridade com esses operadores são essenciais para trabalhar eficazmente com o R.

Funções

O R é uma linguagem de programação estatística e, como tal, possui uma ampla gama de comandos e funções para realizar tarefas específicas. Aqui estão alguns dos comandos básicos e essenciais no R:

Estrutura de uma função Uma função no R tem a seguinte estrutura:

```
nome_da_funcao(argumento1, argumento2, argumento3..)
```

Funções Matemáticas:

- `sqrt()`: Calcula a raiz quadrada.

```r
sqrt(81)
```

```
## [1] 9
```

Ajuda:

- `?nome_da_funcao`: Fornece ajuda sobre uma função específica.
- ou `help(nome_da_funcao)`

Funções estatísticas

- `mean()`: Calcula a média de um vetor numérico.
- `median()`: Calcula a mediana de um vetor numérico.
- `sum()`: Calcula a soma dos elementos de um vetor numérico.
- `min()`: Encontra o valor mínimo em um vetor.
- `max()`: Encontra o valor máximo em um vetor.
- `sd()`: Calcula o desvio padrão de um vetor.
- `var()`: Calcula a variância de um vetor.

Sintaxe básica

O R é constituído por três componentes essenciais: Variáveis, Comentários e Palavras-chave. As Variáveis têm a função de armazenar dados, enquanto os Comentários desempenham o papel de melhorar a legibilidade do código. Já as Palavras-chave são termos reservados que possuem um significado específico para o compilador, desempenhando um papel crítico na estrutura e funcionalidade do programa.

Variáveis A semelhança de outras linguagens de programação, as variáveis no R desempenham o papel de rótulos atribuídos a locais de memória reservados, permitindo o armazenamento de uma ampla variedade de tipos de dados. No contexto do R, a atribuição de valores a variáveis pode ser realizada de três maneiras distintas, oferecendo flexibilidade e versatilidade no tratamento de informações.

- `=` (Atribuição Simples)
- `<-` (Atribuição para a esquerda)
- `->` (Atribuição à direita)

Por convenção usa-se o operador `<-` para criar os objetos e `=` para usar em funções

O funcionamento do operador de atribuição é o seguinte:

```
colocar aqui <- oque tiver aqui
```

Para atribuir vários dados usa-se o operador de concatenação `c()`

```
uma_espécie_coletada <- 1
uma_espécie_coletada
```

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

```
especies_coletadas <- c(5, 20, 12, 29)
especies_coletadas
```

```
## [1]  5 20 12 29
```

Comentários Comentários desempenham um papel essencial na aprimoração da clareza e compreensão do seu código. Eles são direcionados exclusivamente aos usuários e são ignorados pelo interpretador. No contexto de R, é possível criar comentários de uma única linha utilizando o símbolo `#` no início da instrução.

```
# Tudo que se escreve após o símbolo #, o interpretador não lê
especies_coletadas <- c(5, 20, 12, 29) # Espécies coletadas no rio
especies_coletadas <- c(2, 10, 5, 52) # Espécies coletadas no lago
```

Palavras-chave Palavras-chave são termos reservados por um programa devido ao seu significado especial. Como resultado, uma palavra-chave não pode ser empregada como nome de variável, função, ou em outros contextos semelhantes.

O R possui várias palavras reservadas que não podem ser usadas como nomes de variáveis, funções ou outros objetos em seu código. Algumas das palavras-chave mais comuns no R incluem:

if	else	repeat	while	for	in
next	break	function	NULL	NA	Inf
NaN	TRUE	FALSE	and	or	not
as	switch	class	library	require	

Lembre-se de que essas palavras-chave são sensíveis a maiúsculas e minúsculas, portanto, "if" é uma palavra-chave, mas "If" ou "IF" não são. Evite usar essas palavras reservadas como nomes para objetos em seu código para evitar conflitos e erros.

Tipos de Dados e estrutura de dados

Objetos No R, um objeto é uma instância de uma estrutura de dados que contém informações ou valores. Os objetos são usados para armazenar, manipular e organizar dados e resultados de cálculos. Cada objeto tem um nome que serve para identificá-lo e referenciá-lo em operações subsequentes.

Existem várias estruturas de dados no R, como vetores, matrizes, listas, data frames e muito mais, e cada uma delas pode ser usada para criar objetos. Essas estruturas de dados podem conter elementos de diferentes tipos (números, caracteres, lógicos, etc.) e podem ser nomeadas de forma personalizada.

A criação de objetos no R é uma parte fundamental da programação e análise de dados. Os objetos podem representar informações, como conjuntos de dados brutos, resultados de análises estatísticas, gráficos, funções e muito mais. Aqui estão alguns exemplos de como criar objetos em R:

Tipos de Dados

- Números racionais (numeric/double). `c(1,2,5, 1.2,6,8.10)`
- Texto (character). `c("bodó","pirarucu","sardinha","jaraqui")` Para escrever textos usa-se aspas `" "`
- Lógicos (logical). `c(TRUE, FALSE)`
- Inteiros (integer). `c(1L, 2L)`

Estrutura de Dados

1. **Vetores:**
    - Vetores são a estrutura de dados mais fundamental no R. Eles podem conter elementos do mesmo tipo (números inteiros, números reais, caracteres, lógicos) e são criados com a função `c()`.

```
riqueza_esp <- c(25, 30, 35, 40)
riqueza_esp
```

```
## [1] 25 30 35 40
```

```
nome_esp <- c("pirarucu","bodó","jaraqui","pacu")
nome_esp
```

```
## [1] "pirarucu" "bodó"     "jaraqui"  "pacu"
```

2. **Matriz:**

- Matrizes são semelhantes a vetores, mas têm dimensões, ou seja, podem ser bidimensionais. Você pode criar matrizes usando a função `matrix()`.

```
matriz_dados <- matrix(1:12, nrow = 3, ncol = 4)
matriz_dados
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
```

```
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

3. **Lista:**

- Listas podem conter elementos de diferentes tipos, incluindo outros vetores, matrizes, listas e até funções. Elas são criadas com a função `list()`.

```
lista_esp <- list(esp = "pirarucu", riqueza = 30, fase = "adulto")
lista_esp
```

```
## $esp
## [1] "pirarucu"
##
## $riqueza
## [1] 30
##
## $fase
## [1] "adulto"
```

4. **Data Frame:**

- Data frames são semelhantes a tabelas em um banco de dados. Eles são bidimensionais e podem conter colunas de diferentes tipos. Data frames são frequentemente usados para armazenar dados tabulares e são criados com a função `data.frame()`.

```
dados <- data.frame(riqueza_esp = c(25, 30, 35, 40),
                    nome_esp = c("pirarucu","bodó","jaraqui","pacu"))
dados
```

```
##   riqueza_esp nome_esp
## 1          25 pirarucu
## 2          30     bodó
## 3          35  jaraqui
## 4          40     pacu
```

Depois de criar objetos, você pode realizar diversas operações com eles, como realizar cálculos, criar gráficos, realizar análises estatísticas e muito mais. Os objetos no R facilitam o armazenamento e a manipulação de dados, tornando-o uma linguagem poderosa para análise de dados e estatísticas.

Manipulação de Dados

A manipulação de dados é uma etapa fundamental em qualquer análise de dados ou projeto de ciência de dados. Refere-se ao processo de preparação e transformação de conjuntos de dados para que eles estejam em um formato adequado e contenham as informações necessárias para realizar análises estatísticas ou gerar insights valiosos. No contexto da linguagem de programação R, a manipulação de dados envolve uma série de operações que permitem a você limpar, organizar e preparar seus dados para análise.

Objetivos da Manipulação de Dados A manipulação de dados no R visa atingir vários objetivos essenciais:

1. **Limpeza de Dados**: Isso envolve lidar com dados ausentes, duplicados e inconsistentes. A remoção de observações com dados ausentes ou a imputação de valores faltantes são etapas comuns na limpeza de dados.
2. **Transformação de Dados**: Isso inclui a conversão de tipos de dados, como datas e strings, em formatos que sejam compatíveis com as operações de análise. Também envolve a criação de novas variáveis a partir das existentes, aplicação de funções matemáticas e estatísticas, entre outras transformações.

3. **Reestruturação de Dados**: Às vezes, os dados precisam ser remodelados para se adequar a um formato específico. Isso pode envolver a transposição de tabelas, agregação de dados, ou até mesmo a separação de colunas em várias colunas ou vice-versa.
4. **Seleção de Variáveis**: Em muitos casos, os conjuntos de dados contêm uma grande quantidade de variáveis. A manipulação de dados permite que você selecione apenas as variáveis relevantes para a análise, tornando o conjunto de dados mais focado.
5. **Agregação e Sumarização**: Em análises mais avançadas, é comum agrupar dados por categorias e calcular estatísticas resumidas, como médias, medianas, desvios-padrão, entre outras. Isso é especialmente útil para explorar padrões nos dados.

Manipulando dados com o `R base`

Indexação de vetor e dataframe A linguagem de programação R oferece poderosas ferramentas para manipulação de dados, e a **indexação de vetores** e dataframes desempenha um papel fundamental nesse processo. Essa capacidade permite acessar, modificar e explorar dados de maneira eficiente.

Indexação de Vetores: Vetores em R são estruturas unidimensionais que podem conter elementos de diversos tipos de dados. A indexação de vetores é a técnica pela qual você acessa elementos específicos de um vetor. Em R, os índices começam em 1, diferentemente de algumas outras linguagens que começam em 0. Por exemplo:

Além disso, é possível criar subconjuntos de vetores com base em condições específicas, facilitando a extração de informações relevantes.

Para acessar um objeto usa-se `[ ]`

Os vetores possuem só uma dimensão

- Dados numéricos

```
dados_vetor <- c(1:20)
dados_vetor
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
dados_vetor[2] <- 3 # Substituir o segundo valor por 3
dados_vetor
```

```
##  [1]  1  3  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

- Dados de texto

```
dados_vetor2 <- c("Rio Amazonas", "Rio Negro")
dados_vetor2
```

```
## [1] "Rio Amazonas" "Rio Negro"
```

```
dados_vetor2[1] <- "Rio Solimões"
dados_vetor2
```

```
## [1] "Rio Solimões" "Rio Negro"
```

Indexação de Dataframes: Dataframes são estruturas bidimensionais que podem ser vistas como tabelas, onde as colunas representam variáveis e as linhas representam observações. A indexação de dataframes em R é uma extensão da indexação de vetores, permitindo o acesso a elementos específicos em linhas e colunas.

É possível criar subconjuntos de dataframes com base em condições em uma ou mais colunas, proporcionando uma flexibilidade valiosa na manipulação de conjuntos de dados complexos.

- Dataframe possui duas dimensões `[linha ,coluna]`

```r
dados <- data.frame(
  Espécies = paste0("esp", 1:12),
  Periodo = as.factor(rep(c("Cheia", "Seca"), each = 6)),
  Abundancia = c(24,25,22,30,55,45,38,75,58,62,64,48)
)
dados
```

```
##    Espécies Periodo Abundancia
## 1      esp1   Cheia         24
## 2      esp2   Cheia         25
## 3      esp3   Cheia         22
## 4      esp4   Cheia         30
## 5      esp5   Cheia         55
## 6      esp6   Cheia         45
## 7      esp7    Seca         38
## 8      esp8    Seca         75
## 9      esp9    Seca         58
## 10    esp10    Seca         62
## 11    esp11    Seca         64
## 12    esp12    Seca         48
```

- Exemplos de acessar dados de um dataframe

```r
# Acessar a primeira coluna
dados[1]
```

```
##    Espécies
## 1      esp1
## 2      esp2
## 3      esp3
## 4      esp4
## 5      esp5
## 6      esp6
## 7      esp7
## 8      esp8
## 9      esp9
## 10    esp10
## 11    esp11
## 12    esp12
```

```r
# Acessar as colunas menos a 3
dados[-3]
```

```
##    Espécies Periodo
## 1      esp1   Cheia
## 2      esp2   Cheia
## 3      esp3   Cheia
## 4      esp4   Cheia
## 5      esp5   Cheia
## 6      esp6   Cheia
## 7      esp7    Seca
## 8      esp8    Seca
## 9      esp9    Seca
## 10    esp10    Seca
## 11    esp11    Seca
## 12    esp12    Seca
```

```r
# Acessar a primeira linha
dados[1,]
```

```
##   Espécies Periodo Abundancia
## 1     esp1   Cheia         24
```

```r
# Acessar a primeira linha e a primeira coluna
dados[1,1]
```

```
## [1] "esp1"
```

```r
# Acessar as 5 primeiras linhas
dados[1:5,]
```

```
##   Espécies Periodo Abundancia
## 1     esp1   Cheia         24
## 2     esp2   Cheia         25
## 3     esp3   Cheia         22
## 4     esp4   Cheia         30
## 5     esp5   Cheia         55
```

Outras opções para observar um dataframe como encontrar o somatórios dos dados

```r
# Cheia
sum(dados[1:6,3])
```

```
## [1] 201
```

```r
# usando um operador lógico
sum(dados[dados$Periodo == "Cheia", 3])
```

```
## [1] 201
```

```r
# 201

# Seca
sum(dados[7:12,3])
```

```
## [1] 345
```

```r
# usando um operador lógico
sum(dados[dados$Periodo == "Seca", 3])
```

```
## [1] 345
```

```r
# 345
```

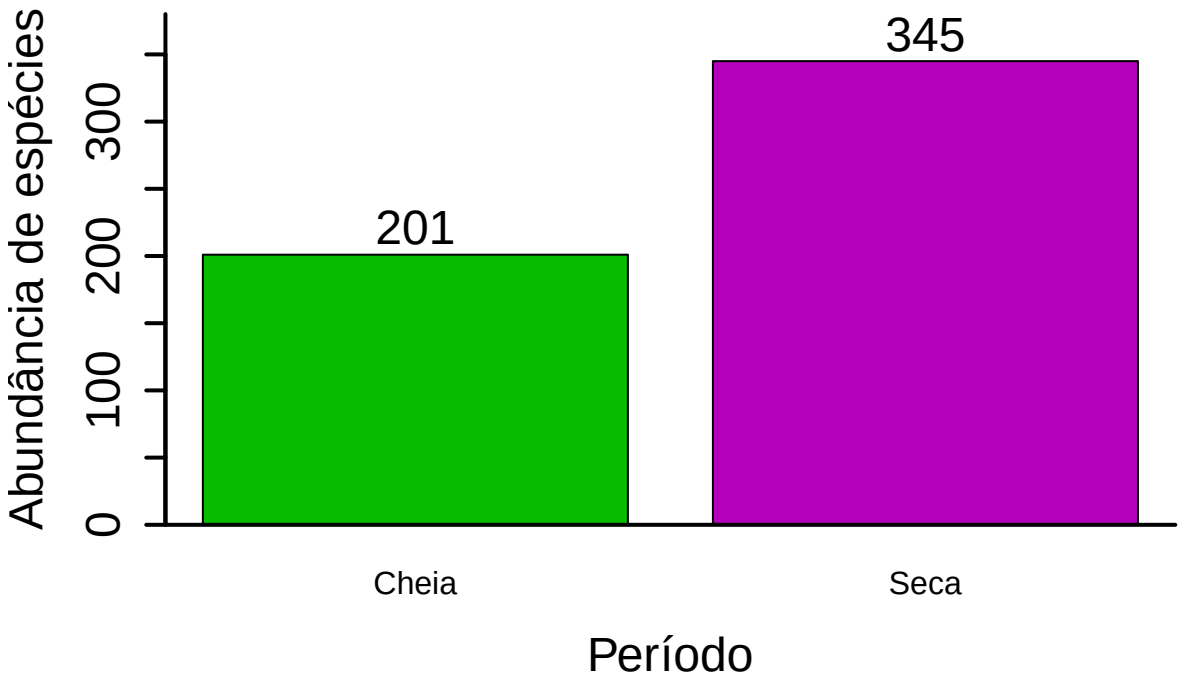
Gráficos simples É possível criar gráficos simples no R base para explorar os dados `(recursos avançados de gráficos serão tratados mais tarde)`

```r
# Vetor com os valores das somas dos períodos
soma_periodo <- c(Cheia = 201, Seca = 345)
```

```r
grafico_barra <- barplot(soma_periodo,
        # Cores dos gráficos
        col = c("#08bb00",  "#b300bb"),
        # nome dos eixos
        ylab = "Abundância de espécies",
```

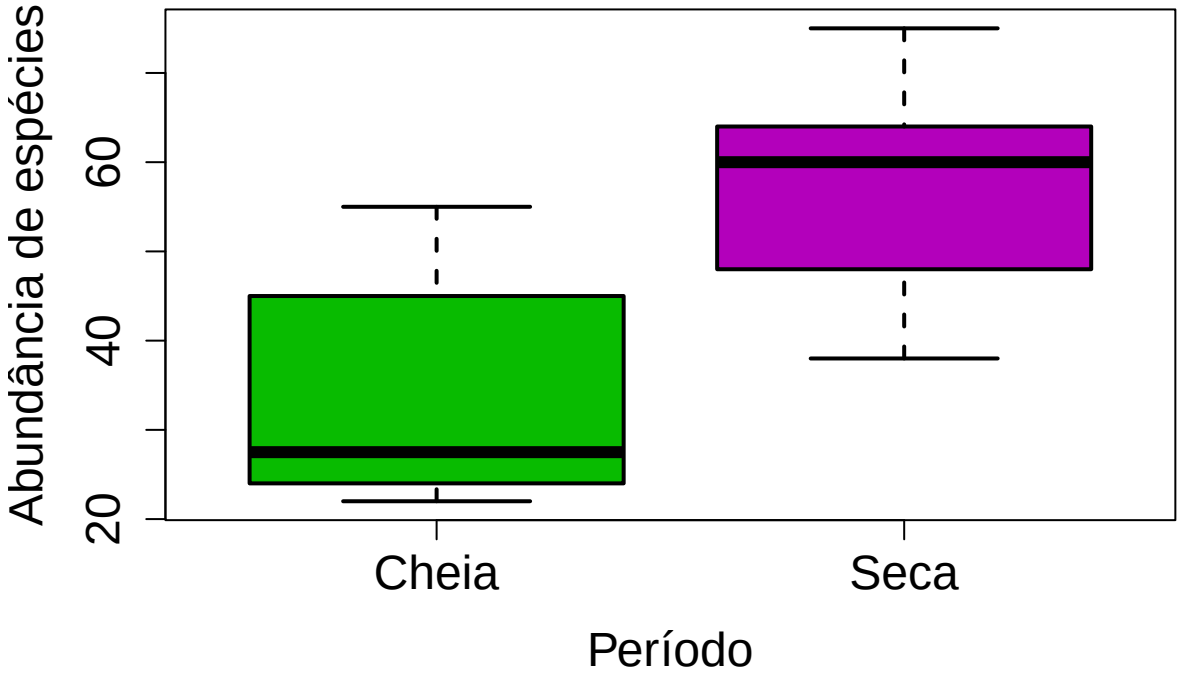
```
        xlab = "Período",
        # Largura das bordas
        lwd = 2,
        # Limite do eixo Y
        ylim = c(0,380),
        # Tamanho da fonte dos eixos
        cex.lab = 1.5, cex.axis = 1.5)
# Adicionando uma linha em formato L
box(bty = "L", lwd = 2)
# Adicionando os valores nas barras
text(x= grafico_barra, y = soma_periodo+20,
     labels = soma_periodo, cex = 1.5)
```

Barplot



Boxplot Também pode ser criado um boxplot

```
boxplot(Abundancia ~ Periodo, data = dados,
        # Cores das caixas
        col = c("#08bb00",  "#b300bb"),
        # Nomes dos eixos
        ylab = "Abundância de espécies",
        xlab = "Período", lwd = 2, cex.lab = 1.5,
        cex = 1.5, cex.axis = 1.5)
```



Importação de dados

Importando dados no R: Pacotes e Métodos O R é uma linguagem de programação amplamente utilizada para análise de dados e estatísticas, e uma das primeiras etapas em qualquer análise de dados é a importação de dados. O R oferece diversas maneiras de importar dados de várias fontes, como arquivos CSV, Excel, bancos de dados, APIs da web e muito mais. A capacidade de importar e manipular dados é fundamental para qualquer análise estatística ou projeto de ciência de dados, e o R oferece uma ampla gama de ferramentas e pacotes que tornam essa tarefa mais eficaz e eficiente.

Pacotes para Importação de Dados A importação de dados no R é uma parte fundamental do processo de análise de dados. O R oferece uma variedade de pacotes que simplificam esse processo, tornando-o mais acessível e eficaz. Alguns dos pacotes mais amplamente usados para importação de dados incluem:

1. **base**: O R possui um conjunto de funções base que permite a importação de dados a partir de arquivos CSV, TXT e outros formatos simples. A função `read.csv()` é comumente usada para ler arquivos CSV, enquanto `read.table()` é usada para ler dados tabulares de texto. No entanto, essas funções têm algumas limitações em relação a formatos de dados mais complexos.
2. **readr**: O pacote "readr" é parte do ecossistema do tidyverse e fornece funções aprimoradas para importação de dados. Ele é especialmente útil para ler arquivos CSV, TSV e outros formatos de dados delimitados. O `read_csv()` e `read_tsv()` são funções populares deste pacote, que também automatizam a detecção do tipo de dados e codificação.

3. **readxl**: Se você precisa importar dados de planilhas do Excel, o pacote "readxl" é uma escolha sólida. Ele fornece funções como `read_excel()` para importar dados diretamente de arquivos Excel (.xls e .xlsx) e é uma alternativa eficaz às funções base do R.

```
# Pacotes------------------------------------------------
library(tidyverse)
library(readxl)
library(ggtext)
library(ggthemes)
library(broom)
library(patchwork)
library(rstatix)
library(car)
library(devtools)
```

Pacotes usados nesse tópico

```
dados_ictio <- read.delim2(file = "clipboard", header = TRUE)
```

Lendo dados pelo Ctrl C + Ctrl V

```
# Lendo dados de arquivos .csv
dados_ictio <- read.csv2("dados/dados_esp.csv")
dados_ictio
```

Lendo dados com a função `read.csv2`

```
# Lendo dados de arquivos do excel .xls e .xlsx
dados_ictio <- read_excel("dados/dados_esp.xlsx", sheet = 1, na = "NA")
dados_ictio
```

Lendo dados com a função `read_excel`

```
## # A tibble: 22,398 x 14
##    order  family genus species river_system locality date                habitat
##    <chr>  <chr>  <chr> <chr>   <chr>        <chr>    <dttm>              <chr>
##  1 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  2 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  3 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  4 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  5 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  6 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  7 Chara~ Acest~ Aces~ Acestr~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  8 Chara~ Acest~ Aces~ Acestr~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  9 Silur~ Dorad~ Ossa~ Ossanc~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
## 10 Silur~ Dorad~ Ossa~ Ossanc~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
## # i 22,388 more rows
## # i 6 more variables: temp <dbl>, ph <dbl>, do <dbl>, cd <dbl>, tb <dbl>,
## #   tp <dbl>
```

Antes de começar a manipular os dados é bom verificar se os dados foram carregados corretamente. Existem várias formas, serão abordadas algumas.

`View()` Visualiza os dados em uma planilha

```
View(dados_ictio)
```

`glimpse()` Verifica o tipos dos dados

```
glimpse(dados_ictio)
```

```
## Rows: 22,398
## Columns: 14
## $ order        <chr> "Siluriformes", "Siluriformes", "Siluriformes", "Silurifo~
## $ family       <chr> "Doradidae", "Doradidae", "Doradidae", "Doradidae", "Dora~
## $ genus        <chr> "Trachydoras", "Trachydoras", "Trachydoras", "Trachydoras~
## $ species      <chr> "Trachydoras nattereri", "Trachydoras nattereri", "Trachy~
## $ river_system <chr> "Amazonas", "Amazonas", "Amazonas", "Amazonas", "Amazonas~
## $ locality     <chr> "Floodplain lake at Pajau; on the south margin of the mai~
## $ date         <dttm> 2014-11-26, 2014-11-26, 2014-11-26, 2014-11-26, 2014-11-~
## $ habitat      <chr> "Floodplain lake", "Floodplain lake", "Floodplain lake", ~
## $ temp         <dbl> 30.1, 30.1, 30.1, 30.1, 30.1, 30.1, 30.1, 30.1, 30.1, 30.~
## $ ph           <dbl> 6.94, 6.94, 6.94, 6.94, 6.94, 6.94, 6.94, 6.94, 6.94, 6.9~
## $ do           <dbl> 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.~
## $ cd           <dbl> 78.1, 78.1, 78.1, 78.1, 78.1, 78.1, 78.1, 78.1, 78.1, 78.~
## $ tb           <dbl> 169, 169, 169, 169, 169, 169, 169, 169, 169, 169, 169, 16~
## $ tp           <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.~
```

`class()` Verifica a classe do objeto

```
class(dados_ictio)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

`dim()` Verifica as dimensões do objeto

```
dim(dados_ictio)
```

```
## [1] 22398    14
```

`head()`Carrega os primeiros dados

```
head(dados_ictio)
```

```
## # A tibble: 6 x 14
##   order   family genus species river_system locality date                habitat
##   <chr>   <chr>  <chr> <chr>   <chr>        <chr>    <dttm>              <chr>
## 1 Siluri~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
## 2 Siluri~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
## 3 Siluri~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
## 4 Siluri~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
## 5 Siluri~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
## 6 Siluri~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
## # i 6 more variables: temp <dbl>, ph <dbl>, do <dbl>, cd <dbl>, tb <dbl>,
## #   tp <dbl>
```

`tail()`Carrega os últimos dados

```
tail(dados_ictio)
```

```
## # A tibble: 6 x 14
##   order   family genus species river_system locality date                habitat
##   <chr>   <chr>  <chr> <chr>   <chr>        <chr>    <dttm>              <chr>
## 1 Charac~ Hemio~ Bivi~ Bivibr~ Tapajos      rio Tap~ 2015-02-15 00:00:00 River ~
```

```
## 2 Percif~ Sciae~ Pach~ Pachyp~ Tapajos      rio Tap~ 2014-10-16 00:00:00 River ~
## 3 Percif~ Sciae~ Pach~ Pachyp~ Tapajos      rio Tap~ 2014-10-16 00:00:00 River ~
## 4 Myliob~ Potam~ Pota~ Potamo~ Tapajos      rio Tap~ 2014-10-16 00:00:00 River ~
## 5 Charac~ Iguan~ Bryc~ Brycon~ Tapajos      rio Tap~ 2015-02-15 00:00:00 River ~
## 6 Charac~ Iguan~ Bryc~ Brycon~ Tapajos      rio Tap~ 2015-02-15 00:00:00 River ~
## # i 6 more variables: temp <dbl>, ph <dbl>, do <dbl>, cd <dbl>, tb <dbl>,
## #   tp <dbl>
```

Operador Pipe

O operador Pipe (`%>%`) é uma das funcionalidades mais poderosas e úteis na linguagem de programação R. Introduzido pelo pacote `magrittr` e amplamente adotado em muitos outros pacotes, o operador Pipe torna o código mais legível, eficiente e expressivo. Ele é especialmente útil quando se trabalha com manipulação de dados, análise estatística e visualização de dados.

O operador Pipe permite encadear várias operações de maneira lógica, da esquerda para a direita, em um pipeline de processamento de dados. Isso significa que o resultado de uma operação é passado diretamente para a próxima operação, evitando a necessidade de criar variáveis intermediárias ou atribuições temporárias. Isso torna o código mais conciso e fácil de entender.

O operador Pipe pode ser lido: **E ENTÃO**

Vamos ver um exemplo simples de como o operador Pipe funciona:

```r
# Sem o operador Pipe
resultado <- mean(sqrt(log(1:10)))
# Com o operador Pipe
resultado <-
  1:10 %>%
  log() %>%
  sqrt() %>%
  mean()
```

No exemplo acima, o operador Pipe permite que você leia o código de maneira mais natural, seguindo o fluxo de processamento dos dados da esquerda para a direita. Além disso, ele evita a necessidade de criar variáveis intermediárias para armazenar os resultados parciais de cada operação.

O operador Pipe é particularmente útil ao trabalhar com pacotes como `dplyr` e `ggplot2`, que são amplamente utilizados na análise de dados e visualização. Por exemplo, ao usar o `dplyr` para manipular um conjunto de dados, você pode encadear operações como filtrar, agrupar e resumir os dados de forma limpa e eficiente usando o operador Pipe.

Operador Pipe nativo O operador Pipe nativo da linguagem R foi introduzido na versão 4.1.0, que foi lançada após o meu último treinamento em janeiro de 2022. O operador Pipe nativo, representado por `|>`, foi adicionado para melhorar a legibilidade e a eficiência do código, permitindo a execução de operações em sequência de forma mais direta.

O operador Pipe nativo funciona de maneira semelhante ao operador Pipe do pacote `magrittr`, mas ele é integrado diretamente na linguagem R. Ele encadeia as operações de esquerda para a direita, passando o resultado de uma operação como o primeiro argumento da próxima. Isso facilita a leitura do código e a compreensão do fluxo de dados. Aqui está um exemplo de como o operador Pipe nativo funciona:

```r
# Sem o operador Pipe
resultado <- mean(sqrt(log(1:10)))
# Com o operador Pipe nativo
resultado <-
  1:10 |>
  log() |>
```

```
  sqrt() |>
  mean()
```

Neste exemplo, o operador Pipe nativo é usado para encadear as operações de log, raiz quadrada e cálculo da média de uma maneira mais clara e concisa.

O operador Pipe nativo é uma adição bem-vinda à linguagem R, pois simplifica o código, tornando-o mais legível e reduzindo a necessidade de criar variáveis intermediárias. Ele é particularmente útil em tarefas de manipulação de dados, análise estatística e visualização, tornando o processo de análise de dados mais eficiente e expressivo. Certifique-se de estar usando uma versão do R que inclui o operador Pipe nativo, caso contrário, você precisará continuar a usar a abordagem com o operador Pipe do pacote `magrittr`.

Manipulando dados com o `dplyr`

O pacote `dplyr` é uma ferramenta poderosa projetada para tornar essa tarefa mais intuitiva e eficaz. Desenvolvido por Hadley Wickham, o `dplyr` oferece uma série de funções simples e consistentes para realizar operações comuns de manipulação de dados, como filtragem, seleção, agrupamento, ordenação e resumo.

Ao adotar uma sintaxe clara e coerente, o `dplyr` permite que os usuários expressem suas intenções de maneira mais legível e concisa, tornando o código mais eficiente e fácil de entender. Uma característica marcante do `dplyr` é o uso do operador Pipe (`%>%`), que permite encadear operações de maneira lógica, promovendo um fluxo de código mais natural da esquerda para a direita.

Seja para explorar dados, realizar análises descritivas ou preparar dados para modelagem estatística, o `dplyr` simplifica muitas tarefas comuns, proporcionando uma experiência mais agradável e produtiva para os usuários de R. Nesta introdução, exploraremos algumas das principais funcionalidades do `dplyr` e como elas podem ser aplicadas para otimizar a manipulação de conjuntos de dados em R.

`filter()` A função `filter()` do pacote `dplyr` desempenha um papel central na manipulação de dados em R, permitindo que os usuários filtrem linhas de um conjunto de dados com base em condições específicas..

```
dados_ictio |>
  filter(river_system == "Amazonas")
```

```
## # A tibble: 9,876 x 14
##    order  family genus species river_system locality date                habitat
##    <chr>  <chr>  <chr> <chr>   <chr>        <chr>    <dttm>              <chr>
##  1 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  2 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  3 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  4 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  5 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  6 Silur~ Dorad~ Trac~ Trachy~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  7 Chara~ Acest~ Aces~ Acestr~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  8 Chara~ Acest~ Aces~ Acestr~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
##  9 Silur~ Dorad~ Ossa~ Ossanc~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
## 10 Silur~ Dorad~ Ossa~ Ossanc~ Amazonas     Floodpl~ 2014-11-26 00:00:00 Floodp~
## # i 9,866 more rows
## # i 6 more variables: temp <dbl>, ph <dbl>, do <dbl>, cd <dbl>, tb <dbl>,
## #   tp <dbl>
```

`group_by()` e `summarise()` A função **`group_by()`** é utilizada para agrupar dados com base em uma ou mais variáveis. Essa função é frequentemente usada em conjunto com outras funções do **`dplyr`**, como **`summarise()`**, permitindo análises agrupadas.

A função **summarise()** é usada para resumir os dados dentro de grupos.

```
dados_ictio |>
  group_by(river_system) |>
  # n_distinct conta valores únicos
  summarise(n = n_distinct(species))
```

```
## # A tibble: 3 x 2
##   river_system     n
##   <chr>        <int>
## 1 Amazonas       224
## 2 Arapiuns       128
## 3 Tapajos        149
```

select() e count() A função **select()** é empregada para escolher colunas específicas de um conjunto de dados, tornando mais fácil trabalhar com um subconjunto relevante de variáveis.

A função **count()** simplifica a contagem de observações em cada grupo. É especialmente útil quando você deseja contar a frequência de valores em uma variável categórica.

```
dados_ictio |>
  group_by(river_system) |>
  select(species) |>
  count()
```

```
## # A tibble: 3 x 2
## # Groups:   river_system [3]
##   river_system     n
##   <chr>        <int>
## 1 Amazonas      9876
## 2 Arapiuns      6617
## 3 Tapajos       5905
```

---

arrange() A função **arrange()** ordena as linhas de um conjunto de dados com base em uma ou mais colunas. É útil para organizar os dados de acordo com critérios específicos.

```
# Ordem crescente - Padrão
 dados_ictio |>
  group_by(river_system) |>
  count() |>
  arrange(n)
```

```
## # A tibble: 3 x 2
## # Groups:   river_system [3]
##   river_system     n
##   <chr>        <int>
## 1 Tapajos       5905
## 2 Arapiuns      6617
## 3 Amazonas      9876
```

```
# Ordem decrescente
 dados_ictio |>
   group_by(river_system) |>
   count() |>
   arrange(desc(n))
```

```
## # A tibble: 3 x 2
## # Groups:   river_system [3]
##   river_system     n
##   <chr>        <int>
## 1 Amazonas      9876
## 2 Arapiuns      6617
## 3 Tapajos       5905
```

mutate() A função **mutate()** é usada para criar novas variáveis ou transformar as existentes. Ela permite adicionar colunas com base em operações em colunas existentes.

```
dados_ictio |>
  group_by(river_system) |>
  count() |>
  # desagrupa
  ungroup() |>
  mutate(n_prop = (n/sum(n))*100)|>
  arrange(desc(n))
```

```
## # A tibble: 3 x 3
##   river_system     n n_prop
##   <chr>        <int>  <dbl>
## 1 Amazonas      9876   44.1
## 2 Arapiuns      6617   29.5
## 3 Tapajos       5905   26.4
```

Exploração de dados

A exploração de dados pode ser feita por:

análise numérica: computar estatísticas descritivas

análise gráfica: explorar o comportamento e a relação entre as variáveis através de gráficos.

Visualização de dados com o ggplot2

O pacote ggplot2, desenvolvido pelo renomado estatístico Hadley Wickham, é uma ferramenta essencial no ecossistema R para a criação de gráficos complexos e esteticamente agradáveis. Lançado pela primeira vez em 2005, o ggplot2 se destaca por sua abordagem baseada na gramática de gráficos, oferecendo uma maneira poderosa e intuitiva de visualizar dados.

A filosofia subjacente ao ggplot2 é a de que a criação de gráficos deve ser uma tarefa simples e coerente, independentemente da complexidade do gráfico desejado. O pacote adota uma abordagem declarativa, onde os usuários especificam as características desejadas do gráfico, como variáveis a serem mapeadas, geometrias a serem utilizadas e ajustes de estética. O ggplot2, então, traduz essas instruções em um gráfico completo.

Uma característica distintiva do ggplot2 é a ênfase na criação de gráficos como uma forma de comunicação clara e eficaz. A gramática de gráficos proporcionada pelo ggplot2 permite que os usuários expressem visualizações de dados complexas de maneira consistente, facilitando a interpretação e análise por parte do público-alvo. A modularidade do pacote também incentiva a exploração iterativa, permitindo que os usuários adicionem e ajustem camadas para refinar seus gráficos.

Ao longo dos anos, o ggplot2 tornou-se um padrão de facto para visualização de dados em R, sendo amplamente adotado em ambientes acadêmicos, industriais e de pesquisa. Sua comunidade ativa contribui para uma ampla gama de tutoriais, pacotes complementares e exemplos práticos, tornando o ggplot2 uma ferramenta acessível e poderosa para analistas e cientistas de dados. Em resumo, o ggplot2 não apenas simplifica a criação de gráficos em R, mas eleva a arte da visualização de dados a um nível mais intuitivo e expressivo.

A gramática A gramática de gráficos (Grammar of Graphics) é um conceito fundamental por trás da criação de gráficos com o pacote ggplot2 em R. Desenvolvida por Leland Wilkinson, a ideia é fornecer uma estrutura consistente para expressar visualizações de dados, permitindo que os usuários construam gráficos complexos de maneira modular e compreensível. O ggplot2 implementa essa gramática por meio de uma abordagem declarativa, na qual você especifica o que deseja fazer em vez de como fazer.

A gramática de gráficos do ggplot2 é composta por alguns elementos-chave:

1. **Dados (Data)**: Representa o conjunto de dados que você deseja visualizar. Geralmente, você fornece um dataframe a partir do qual as variáveis serão extraídas para construir o gráfico.
2. **Mapeamento Estético (Aesthetics Mapping)**: Refere-se à associação das variáveis do conjunto de dados às propriedades visuais do gráfico, como eixos x e y, cor, forma, tamanho, etc. Isso é feito usando a função `aes()`.

```
ggplot(dados, aes(x = variavel1, y = variavel2, color = variavel3))
```

3. **Geometria (Geometry)**: Representa o tipo de gráfico que você deseja criar. As funções `geom_*()` (como `geom_point()`, `geom_line()`, etc.) são usadas para especificar a geometria do gráfico.

```
ggplot(dados, aes(x = variavel1, y = variavel2)) +
  geom_point()
```

4. **Facetas (Facets)**: Permite dividir os dados em subconjuntos com base em uma ou mais variáveis, criando múltiplos painéis. Isso é feito usando funções como `facet_grid()` ou `facet_wrap()`.

```
ggplot(dados, aes(x = variavel1, y = variavel2)) +
  geom_point() +
  facet_grid(variavel3 ~ variavel4)
```

5. **Escala (Scale)**: Controla a escala das variáveis visuais, como os eixos x e y. Pode ser usado para personalizar as escalas e transformações.

```
ggplot(dados, aes(x = variavel1, y = variavel2)) +
  geom_point() +
  scale_x_log10()
```

6. **Tema (Theme)**: Define a aparência visual geral do gráfico, incluindo cores, fontes, tamanhos de texto, etc.

```
ggplot(dados, aes(x = variavel1, y = variavel2)) +
  geom_point() +
  theme_minimal()
```

Geometrias Ao combinar esses elementos, você constrói visualizações complexas de dados de maneira modular e compreensível. A gramática de gráficos do ggplot2 torna a criação de gráficos mais intuitiva e facilita a exploração visual dos dados. Cada componente é ajustável de forma independente, permitindo grande flexibilidade na criação de gráficos personalizados e informativos.

Tipos comuns de gráficos e as respectivas geometrias (geoms) frequentemente usadas no ggplot2:

Tipo de Gráfico	Geom no ggplot2
Dispersão	`geom_point()`
Linha	`geom_line()`
Barras	`geom_bar()`
Histograma	`geom_histogram()`
Boxplot	`geom_boxplot()`
Área	`geom_area()`

Tipo de Gráfico	Geom no ggplot2
Superfície	`geom_contour()`
Violino	`geom_violin()`
Densidade	`geom_density()`
Mapa de Calor	`geom_tile()`
Bolhas	`geom_jitter()`
Linhas de Referência	`geom_hline()`, `geom_vline()`

É bom fazer uma pergunta para filtrar os dados

São quantas ordem, familias e espécies?

```
# Ordens
dados_ictio |>
   select(order) |>
   distinct()
```

Ordens

```
## # A tibble: 15 x 1
##    order
##    <chr>
##  1 Siluriformes
##  2 Characiformes
##  3 Cichliformes
##  4 Gymnotiformes
##  5 Clupeiformes
##  6 Perciformes
##  7 Osteoglossiformes
##  8 Beloniformes
##  9 Cyprinodontiformes
## 10 Myliobatiformes
## 11 Pleuronectiformes
## 12 Tetraodontiformes
## 13 Gobiiformes
## 14 Ceratodontiformes
## 15 Synbranchiformes
```

```
# Contribuição de cada ordem

 dados_ictio |>
   select(order) |>
   distinct()
```

```
## # A tibble: 15 x 1
##    order
##    <chr>
##  1 Siluriformes
##  2 Characiformes
##  3 Cichliformes
##  4 Gymnotiformes
##  5 Clupeiformes
##  6 Perciformes
##  7 Osteoglossiformes
```

```
##  8 Beloniformes
##  9 Cyprinodontiformes
## 10 Myliobatiformes
## 11 Pleuronectiformes
## 12 Tetraodontiformes
## 13 Gobiiformes
## 14 Ceratodontiformes
## 15 Synbranchiformes
```

```
# Familias
 dados_ictio |>
   select(family) |>
   distinct()
```

Familias

```
## # A tibble: 45 x 1
##    family
##    <chr>
##  1 Doradidae
##  2 Acestrorhynchidae
##  3 Characidae
##  4 Cichlidae
##  5 Sternopygidae
##  6 Pristigasteridae
##  7 Auchenipteridae
##  8 Triportheidae
##  9 Curimatidae
## 10 Cynodontidae
## # i 35 more rows
```

```
# Contribuição de cada familia

 dados_ictio |>
   select(family) |>
   count(family) |>
   arrange(desc(n))
```

```
## # A tibble: 45 x 2
##    family               n
##    <chr>            <int>
##  1 Characidae        7528
##  2 Iguanodectidae    2463
##  3 Cichlidae         1888
##  4 Engraulidae       1768
##  5 Hemiodontidae     1553
##  6 Curimatidae       1405
##  7 Sternopygidae      873
##  8 Heptapteridae      633
##  9 Auchenipteridae    631
## 10 Acestrorhynchidae  499
## # i 35 more rows
```

```r
# Espécies

 dados_ictio |>
  select(species) |>
  distinct()
```

Especies

```
## # A tibble: 310 x 1
##    species
##    <chr>
##  1 Trachydoras nattereri
##  2 Acestrorhynchus abbreviatus
##  3 Ossancora asterophysa
##  4 Anadoras grypus
##  5 Roeboides myersii
##  6 Cichla monoculus
##  7 Eigenmannia limbata
##  8 Pellona flavipinnis
##  9 Trachelyopterus galeatus
## 10 Triportheus albus
## # i 300 more rows
```

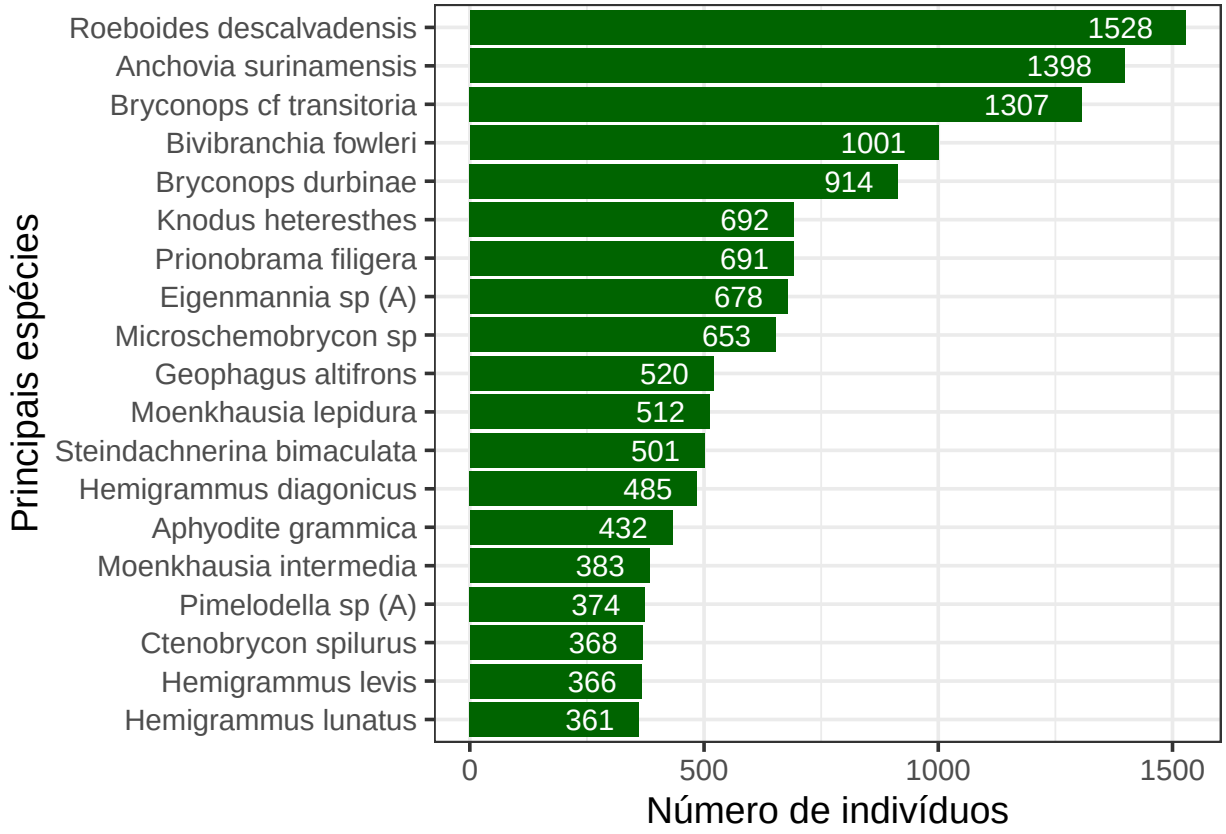
```r
 # Contribuição de especie

 # Espécies por sistema
 dados_ictio |>
   group_by(river_system) |>
   select(species) |>
   distinct() |>
   count()
```

```
## # A tibble: 3 x 2
## # Groups:   river_system [3]
##   river_system     n
##   <chr>        <int>
## 1 Amazonas       224
## 2 Arapiuns       128
## 3 Tapajos        149
```

Quais as espécies mais abundantes?

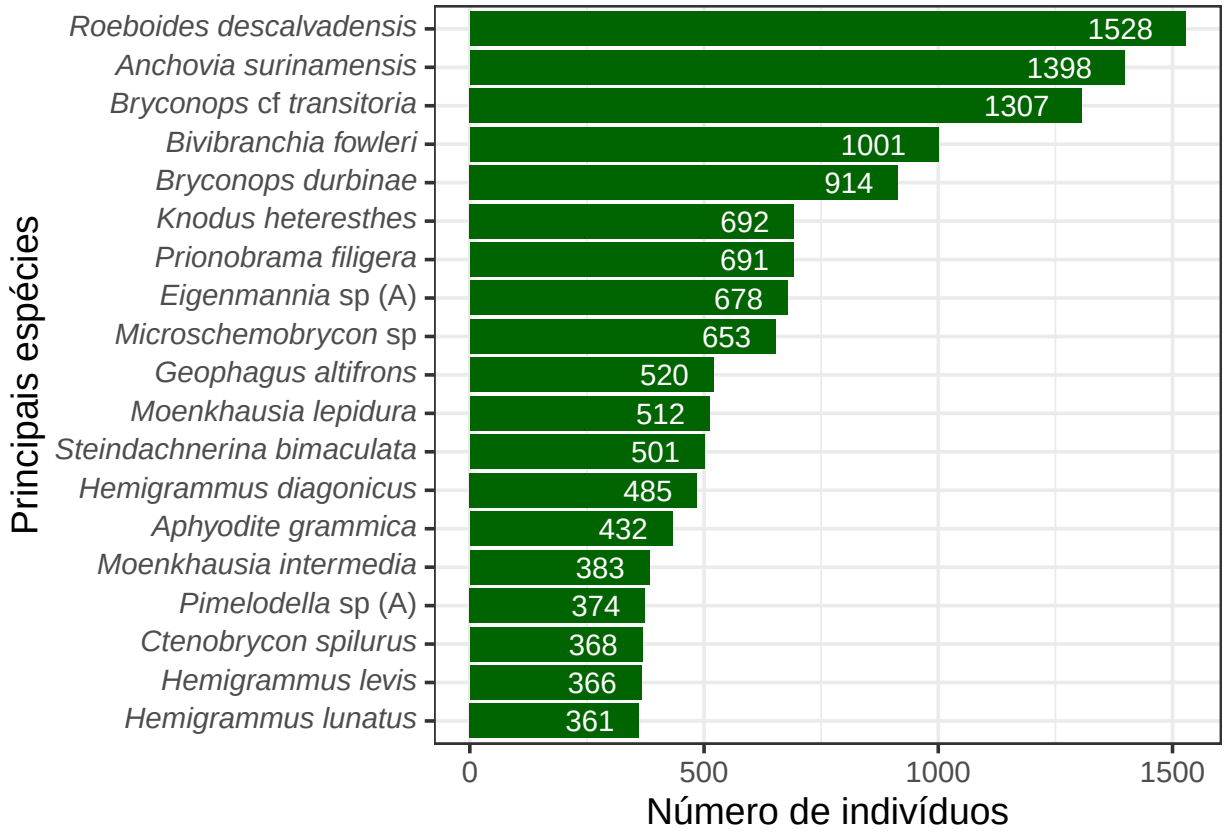
```r
# Gráfico

# Espécies mais frequentes
 esp <- dados_ictio |>
   select(species) |>
   count(species) |>
   filter(n > 357) |>
   arrange(desc(n))

ggplot(esp, aes(x = reorder(species, n), y = n)) +
  # gráfico de barra
 geom_bar(stat = "identity", fill = "darkgreen") +
  # Inverte os eixos
```

```
coord_flip() +
 # adciona rótulos nas barras
geom_text(
  aes(label = n),
  color = "white",
  vjust = 0.5,
  hjust = 1.5
) +
 # Renomeia os eixos
labs(x = "Principais espécies", y = "Número de indivíduos") +
 # tema do gráfico
theme_bw(base_size = 14)
```




```
#------- Para deixar itálico------------#
 # Primeiro adiciona <i> para todas
 esp$esp2 <- paste0("<i>", esp$species, "</i>")

 # Separa sp
 esp$esp2<- sub(" sp</i>", "</i> sp",  esp$esp2)

 # Separação manual
 esp$esp2[3] <-"<i>Bryconops</i> cf <i>transitoria</i>"
 esp$esp2[8] <- "<i>Eigenmannia</i> sp (A)"
 esp$esp2[16] <- "<i>Pimelodella</i> sp (A)"

 # Gráfico
```

```r
ggplot(esp, aes(x = reorder(esp2, n), y = n)) +
  geom_bar(stat = "identity", fill = "darkgreen") +
  coord_flip() +
  geom_text(
    aes(label = n),
    color = "white",
    vjust = 0.5,
    hjust = 1.5
  ) +
  labs(x = "Principais espécies", y = "Número de indivíduos") +
  theme_bw(base_size = 14) +
  theme(axis.text.y = element_markdown())
```



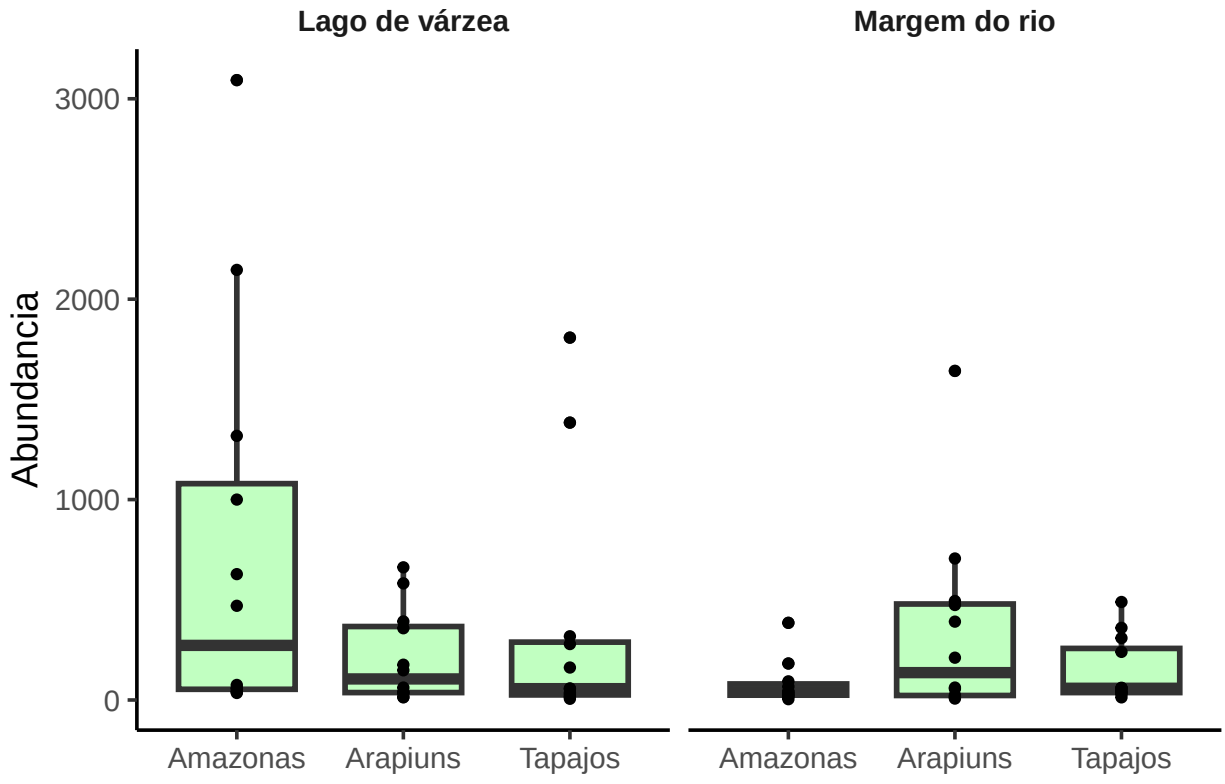
Abundância

```r
# Abundancia-------------------------------
abundancia <-  dados_ictio |>
  group_by(river_system,habitat ,date) |>
  count()
```

```r
# Boxplot

# Traduzir os fatores
abundancia$habitat <- as.factor(abundancia$habitat )
```

```r
levels(abundancia$habitat) <- c("Lago de várzea", "Margem do rio")

plot_abundancia <- ggplot(abundancia, aes(x= river_system, y = n))+
  # Geometria boxplot
  geom_boxplot(width = .7, fill = "darkseagreen1",
    show.legend = FALSE, size = 1)+
  # geometria de ponto
  geom_point(show.legend = FALSE)+
  # tema do gráfico
  theme_classic(base_size = 14)+
  # Divisão do dataset
  facet_wrap( ~ habitat)+
  # tema do gráfico
  theme(strip.text = element_text(face = "bold"))+
  labs(x = "", y = "Abundancia")+
  theme(strip.background = element_blank())

plot_abundancia
```

Gráfico de abundancia



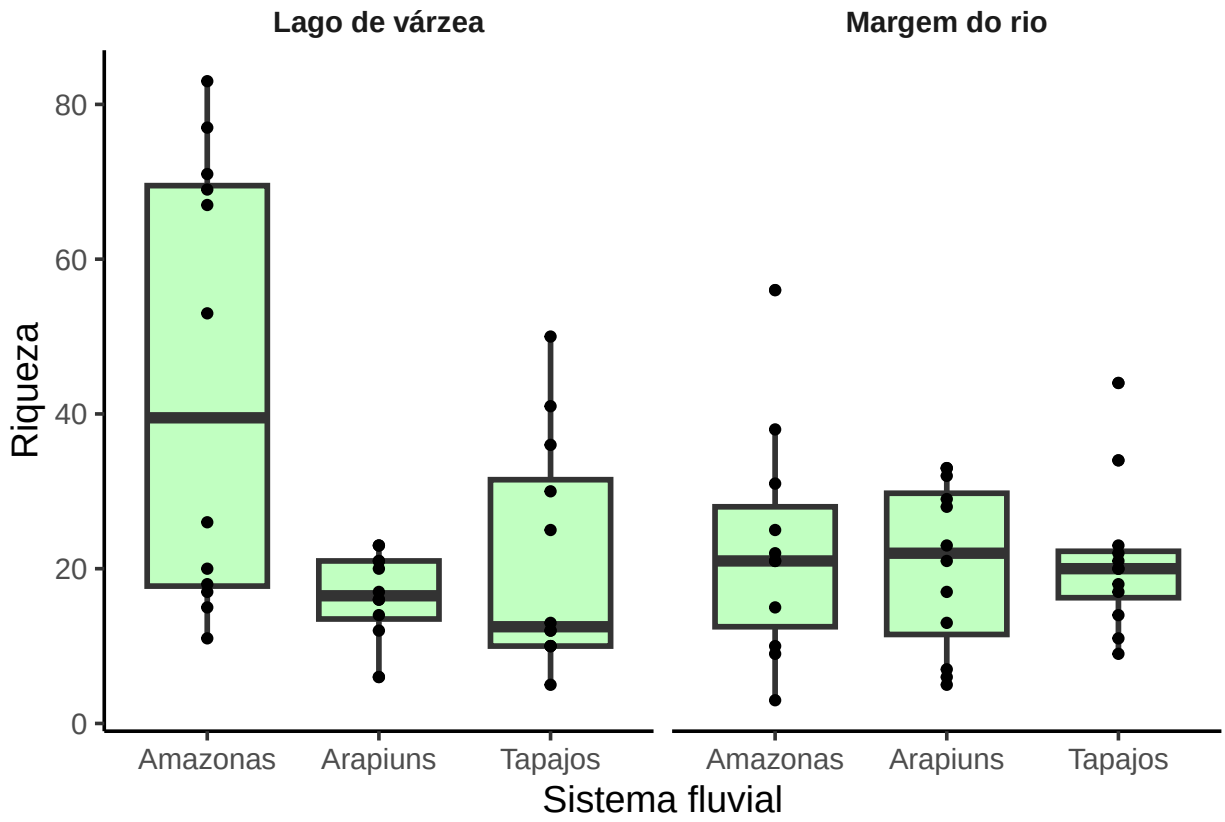

```r
# Riqueza de espécies ----------------------------------

riqueza <-  dados_ictio |>
  group_by(river_system,habitat ,date) |>
```

```
  summarise(riq = n_distinct(species))
```

Riqueza de espécies

```
# Boxplot

riqueza$habitat <- as.factor(riqueza$habitat)
levels(riqueza$habitat) <- c("Lago de várzea", "Margem do rio")

plot_riqueza <- ggplot(riqueza, aes(x= river_system, y = riq))+
  geom_boxplot(width = .7, fill = "darkseagreen1",
    show.legend = FALSE, size = 1)+
  geom_point(show.legend = FALSE)+
  theme_classic(base_size = 14)+
  facet_wrap( ~ habitat)+
  theme(strip.text = element_text(face = "bold"))+
  labs(x = "Sistema fluvial", y = "Riqueza")+
  theme(strip.background = element_blank())

plot_riqueza
```

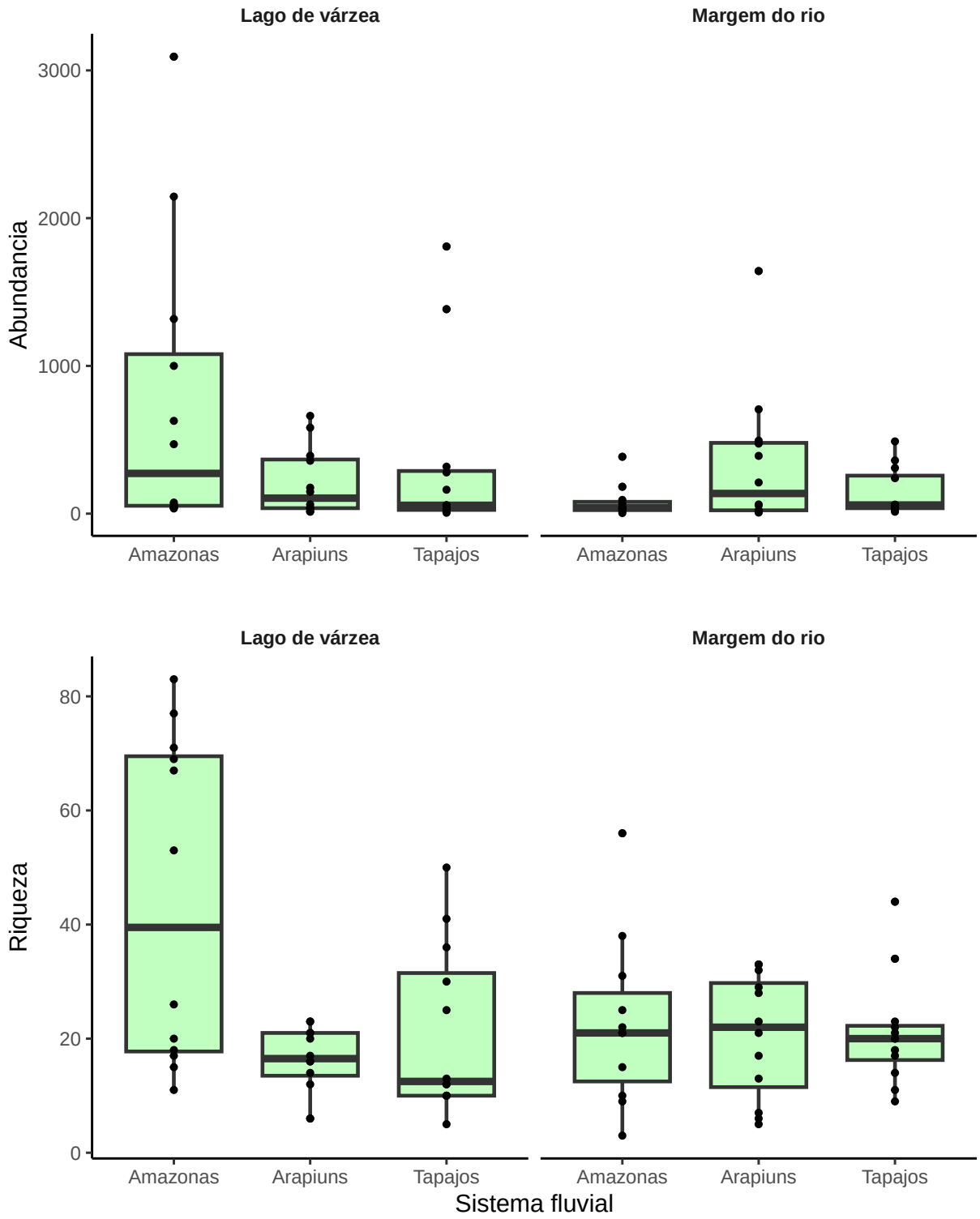
Gráfico de riquza



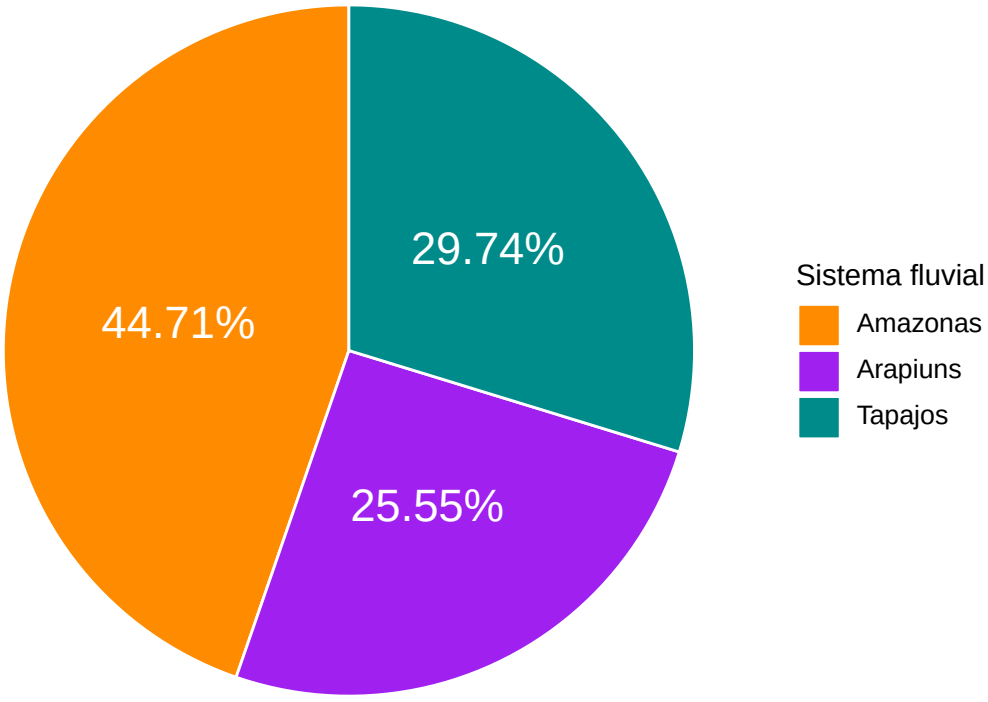

```
# juntando os gráficos
```

```
plot_abundancia / plot_riqueza
```

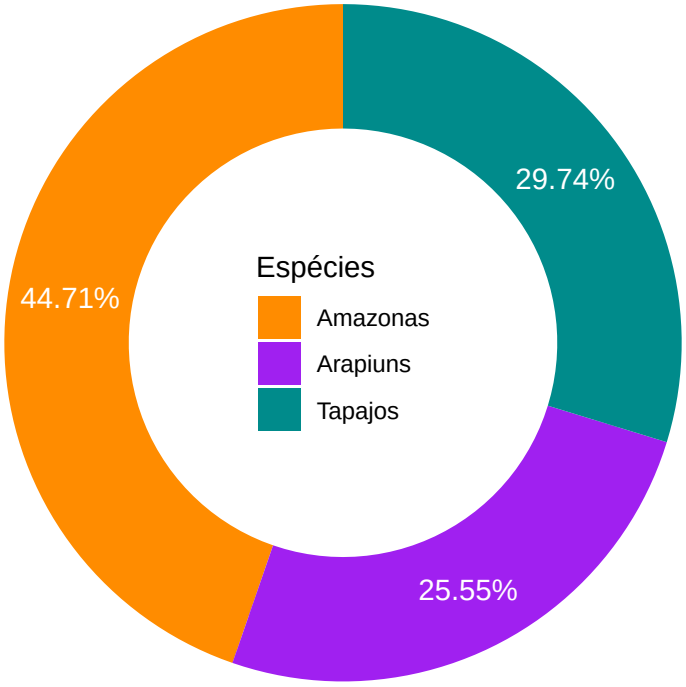
juntando os gráficos



```r
# Riqueza de espécies por sistema

riqueza <- dados_ictio |>
  group_by(river_system) |>
  summarise(riq = n_distinct(species)) |>
  ungroup()

# Porcentagem
riqueza_prop <- riqueza %>%
  dplyr::mutate(prop = round(riq/sum(riq), 4)*100)


# Gráfico de pizza
ggplot(data = riqueza_prop, aes(x = "", y = prop,
  fill = river_system)) +
  geom_bar(stat = "identity", color = "white") +
  geom_text(aes(label = paste0(prop, "%")), color = "white",
    position = position_stack(vjust = .5), size = 6) +
  scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
  coord_polar(theta = "y", start = 0) +
  theme_void() +
  labs(fill = "Sistema fluvial")+
  theme(legend.text = element_markdown(color = "black",
    size = 10))
```



```r
# Gráfico de donut
ggplot(data = riqueza_prop, aes(x = 2, y = prop, fill = river_system)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(prop, "%")), color = "white",
    position = position_stack(vjust = .5), size = 4) +
  scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
  coord_polar(theta = "y", start = 0) +
  xlim(0, 2.5) +
  theme_void() +
  theme(legend.position = c(.5, .5)) +
  labs( fill = "Espécies")
```



Variáveis limnológicas

```r
glimpse(dados_ictio[,c(5, 7:13)])
```

```
## Rows: 22,398
## Columns: 8
## $ river_system <chr> "Amazonas", "Amazonas", "Amazonas", "Amazonas", "Amazonas~
## $ date         <dttm> 2014-11-26, 2014-11-26, 2014-11-26, 2014-11-26, 2014-11-~
## $ habitat      <chr> "Floodplain lake", "Floodplain lake", "Floodplain lake", ~
## $ temp         <dbl> 30.1, 30.1, 30.1, 30.1, 30.1, 30.1, 30.1, 30.1, 30.1, 30.~
## $ ph           <dbl> 6.94, 6.94, 6.94, 6.94, 6.94, 6.94, 6.94, 6.94, 6.94, 6.9~
## $ do           <dbl> 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.7, 4.~
## $ cd           <dbl> 78.1, 78.1, 78.1, 78.1, 78.1, 78.1, 78.1, 78.1, 78.1, 78.~
## $ tb           <dbl> 169, 169, 169, 169, 169, 169, 169, 169, 169, 169, 169, 16~
```

```r
dados_limno <-  dados_ictio[,c(4,5, 7:13)]
```

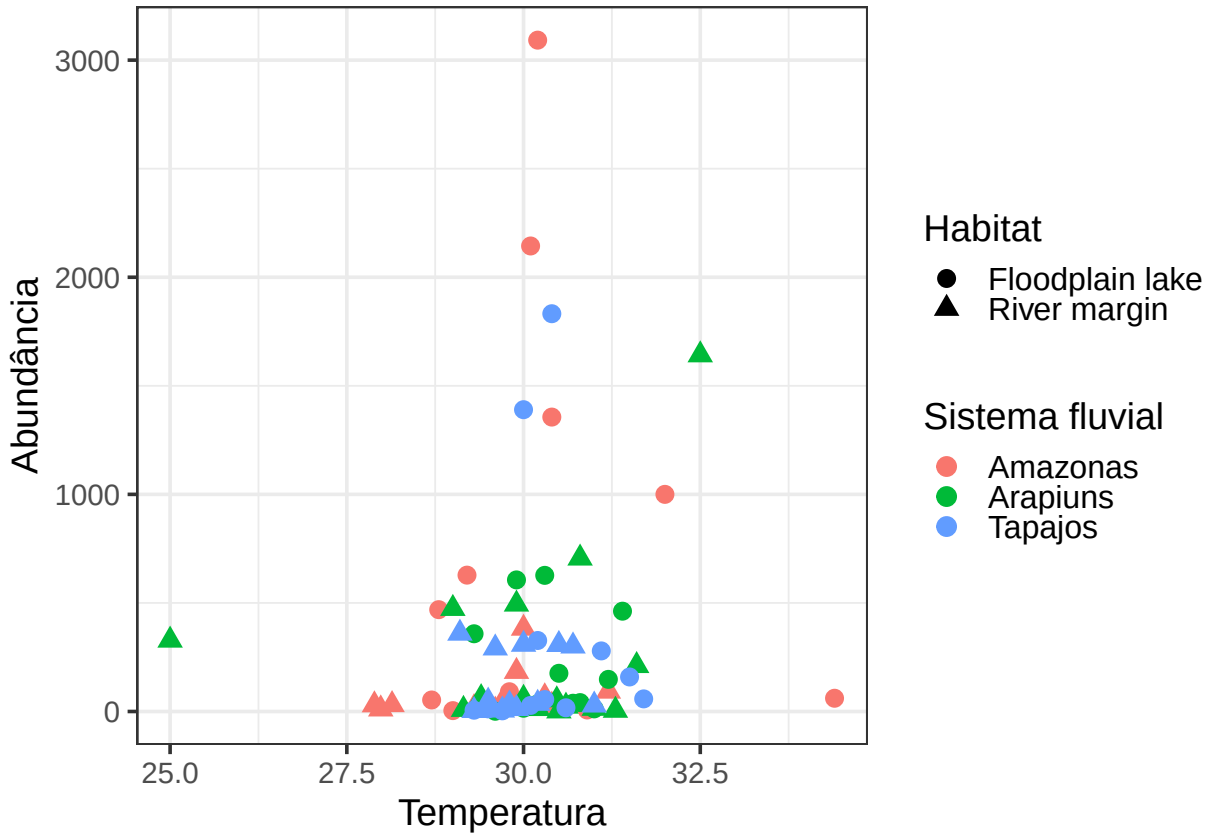
Temperatura Verificando a relação com dados das espécies

```r
# Temperatura
dados_temp <- dados_limno |>
  group_by(river_system, habitat, temp) |>
  select(species) |>
  count()
dados_temp
```

```
## # A tibble: 77 x 4
## # Groups:   river_system, habitat, temp [77]
##    river_system habitat          temp     n
##    <chr>        <chr>           <dbl> <int>
##  1 Amazonas     Floodplain lake  28.7    53
##  2 Amazonas     Floodplain lake  28.8   469
##  3 Amazonas     Floodplain lake  29       4
##  4 Amazonas     Floodplain lake  29.2   628
##  5 Amazonas     Floodplain lake  29.4    19
##  6 Amazonas     Floodplain lake  29.8    91
##  7 Amazonas     Floodplain lake  30      19
##  8 Amazonas     Floodplain lake  30.1  2144
##  9 Amazonas     Floodplain lake  30.2  3092
## 10 Amazonas     Floodplain lake  30.4  1356
## # i 67 more rows
```

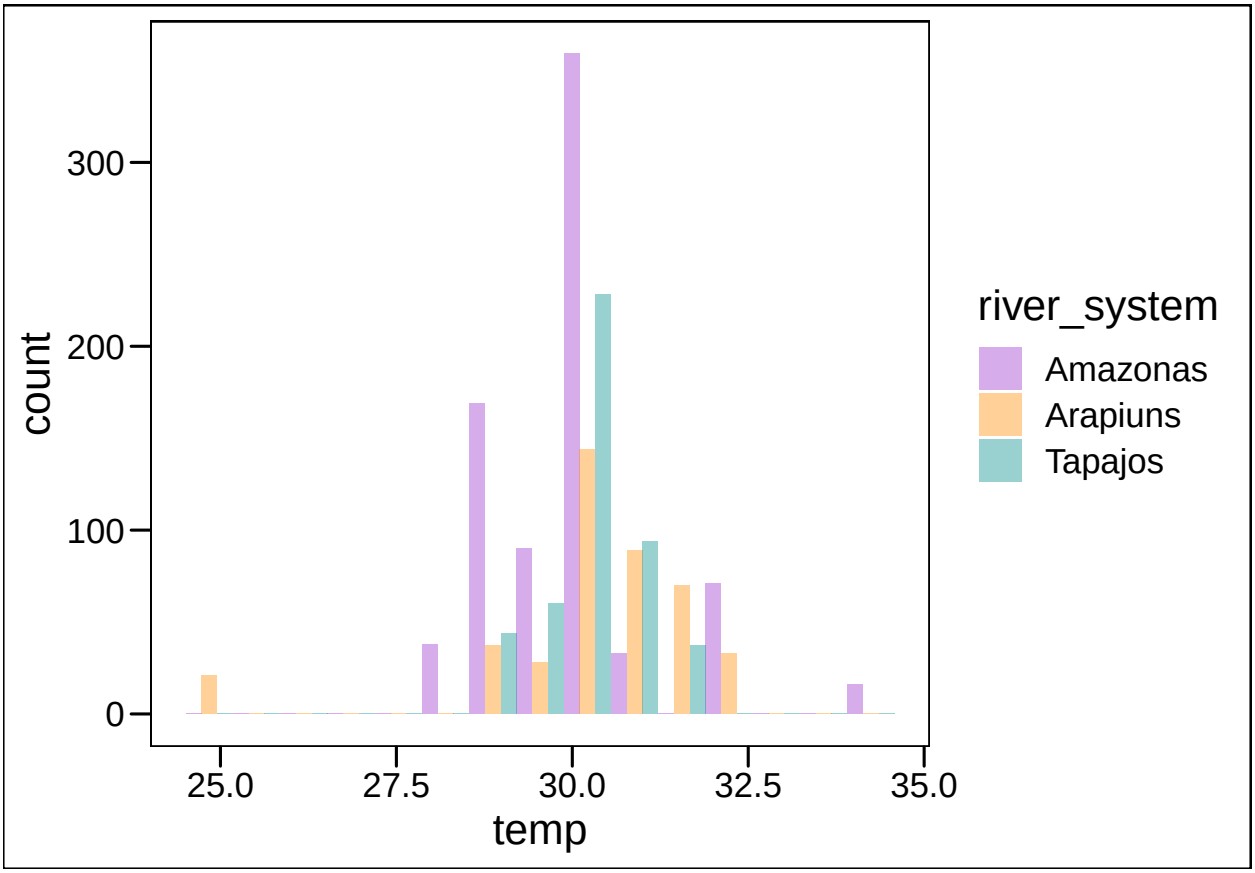
```r
ggplot(dados_temp, aes(x = temp, y = n))+
  geom_point(aes(col = river_system, shape = habitat), size = 3)+
  theme_bw(base_size = 14)+
  labs(x = "Temperatura", y = "Abundância", col = "Sistema fluvial ", shape = "Habitat")+
  # "right", "left", "bottom", "top"
  theme(legend.position = "right",
    legend.key.height = unit(.3, 'cm'),
    legend.text = element_text(size=12),
    legend.background  = element_blank())
```

Gráfico



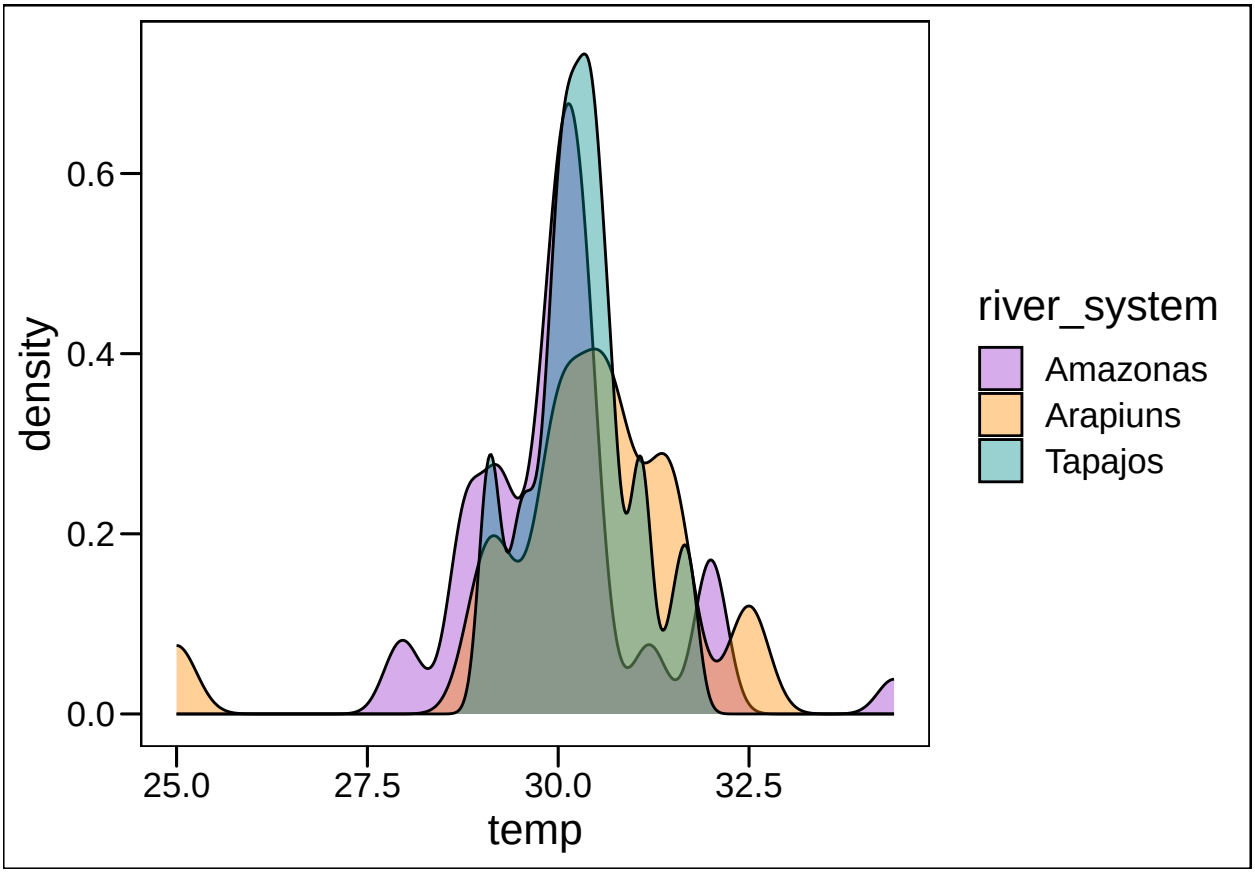
Da para ver o comportamento dos dados por um histograma

```
dados_ictio |>
  group_by(river_system, species, temp) |>
  count(species) |>
  ggplot(aes(x = temp,  fill = river_system))+
  geom_histogram(alpha = .4, position = "dodge",bins = 15)+
  scale_fill_manual(values = c("darkorchid","darkorange", "cyan4"))+
  theme_base()
```



Ou por um gráfico de densidade

```
dados_ictio |>
  group_by(river_system, species, temp) |>
  count(species) |>
  ggplot(aes(x = temp,  fill = river_system))+
  geom_density(alpha = .4)+
  scale_fill_manual(values = c("darkorchid","darkorange", "cyan4"))+
  theme_base()
```




```
sumario_temp <-  dados_limno |>
  group_by(river_system, habitat) |>
  select(temp) |>
  get_summary_stats(type = "common")

sumario_temp
```

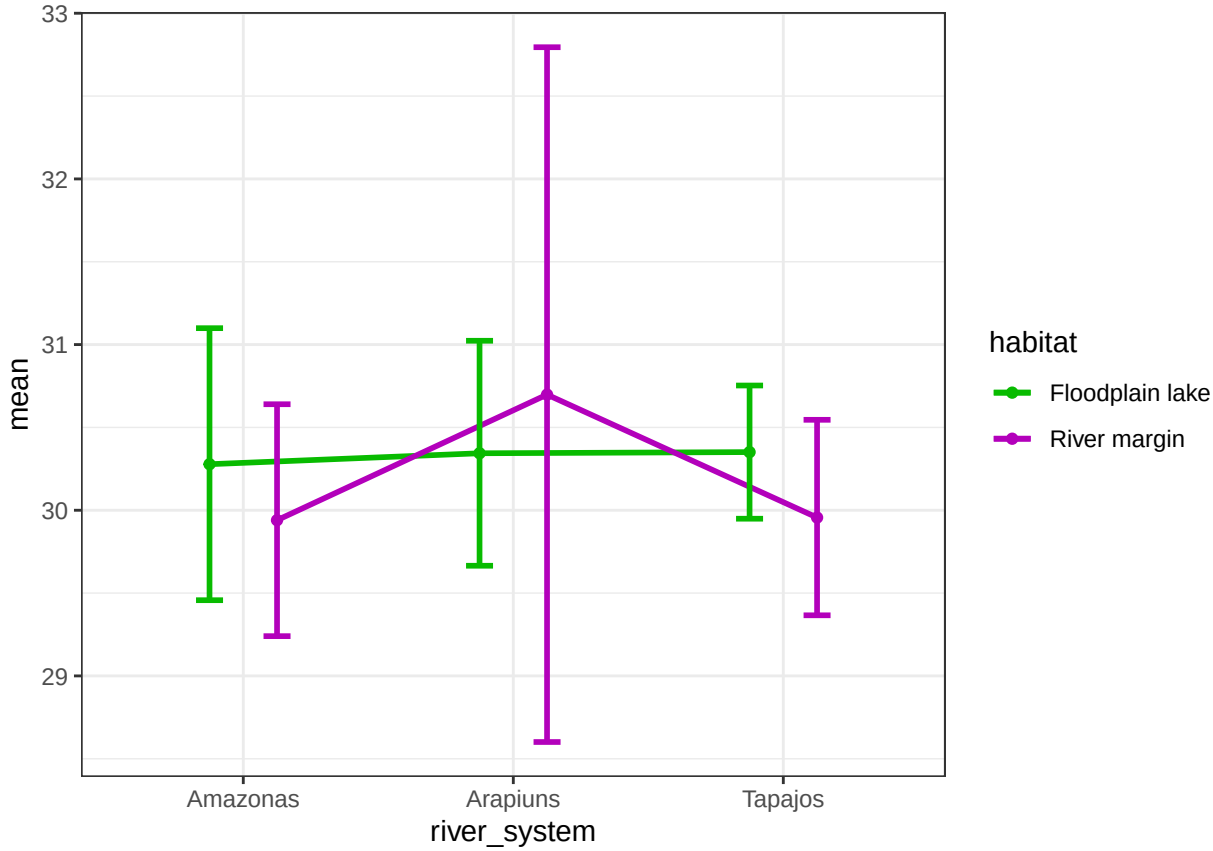
Sumário estatístico

```
## # A tibble: 6 x 12
##   river_system habitat variable     n   min   max median   iqr  mean    sd    se
##   <chr>        <chr>   <fct>    <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Amazonas     Floodp~ temp      8965  28.7  34.4   30.2   0.3  30.3 0.821 0.009
## 2 Amazonas     River ~ temp       911  27.9  31.2   30     0.1  29.9 0.7   0.023
## 3 Arapiuns     Floodp~ temp      2519  29.3  31.4   30.3   0.9  30.3 0.679 0.014
## 4 Arapiuns     River ~ temp      4098  25    32.5   30.8   2.6  30.7 2.10  0.033
## 5 Tapajos      Floodp~ temp      4166  29.3  31.7   30.4   0.4  30.4 0.402 0.006
## 6 Tapajos      River ~ temp      1739  29.1  31     30     0.9  30.0 0.59  0.014
## # i 1 more variable: ci <dbl>
```

```
plot_temp <-  ggplot(sumario_temp,
  aes(x = river_system,y = mean,col = habitat,group = habitat)) +
  geom_line(linewidth = 1, position = position_dodge(.5)) +
  geom_point(position = position_dodge(0.5)) +
```

```
  geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd),
    linewidth = 1,width = .2, position = position_dodge(0.5))+
  theme_bw()+
  scale_color_manual(values = c("#08bb00",  "#b300bb"))
plot_temp
```

Visualizando o resultado



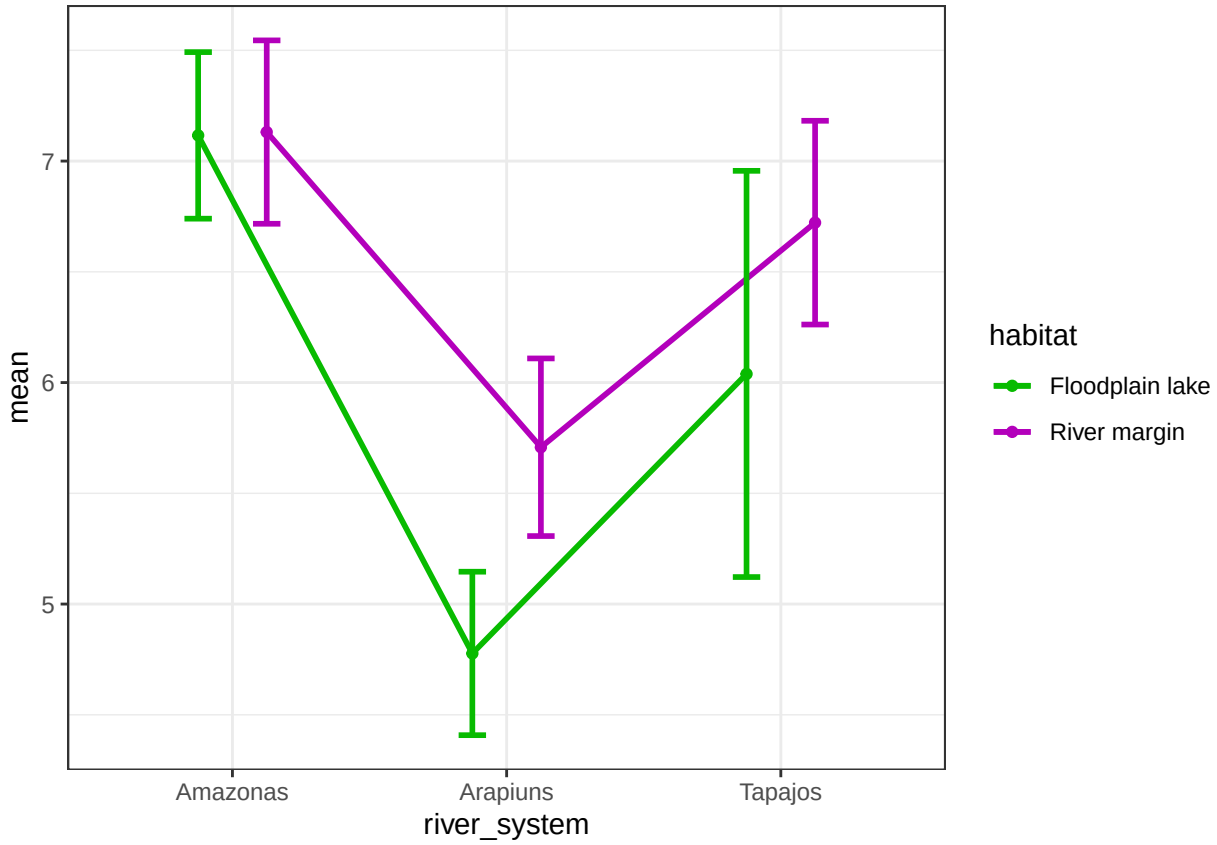

```
# Sumário estatístico de pH
sumario_ph <-  dados_limno |>
  group_by(river_system, habitat) |>
  select(ph) |>
  get_summary_stats(type = "common")
```

pH

```
# Visualizando
plot_ph <-  ggplot(sumario_ph,
  aes(
    x = river_system,
    y = mean,
    col = habitat,
    group = habitat
  )) +
```

```
  geom_line(linewidth = 1, position = position_dodge(.5)) +
  geom_point(position = position_dodge(0.5)) +
  geom_errorbar(
    aes(ymin = mean - sd, ymax = mean + sd),
    linewidth = 1,
    width = .2,
    position = position_dodge(0.5)
  ) +
  theme_bw() +
  scale_color_manual(values = c("#08bb00",  "#b300bb"))

plot_ph
```

Visualizando o reultado

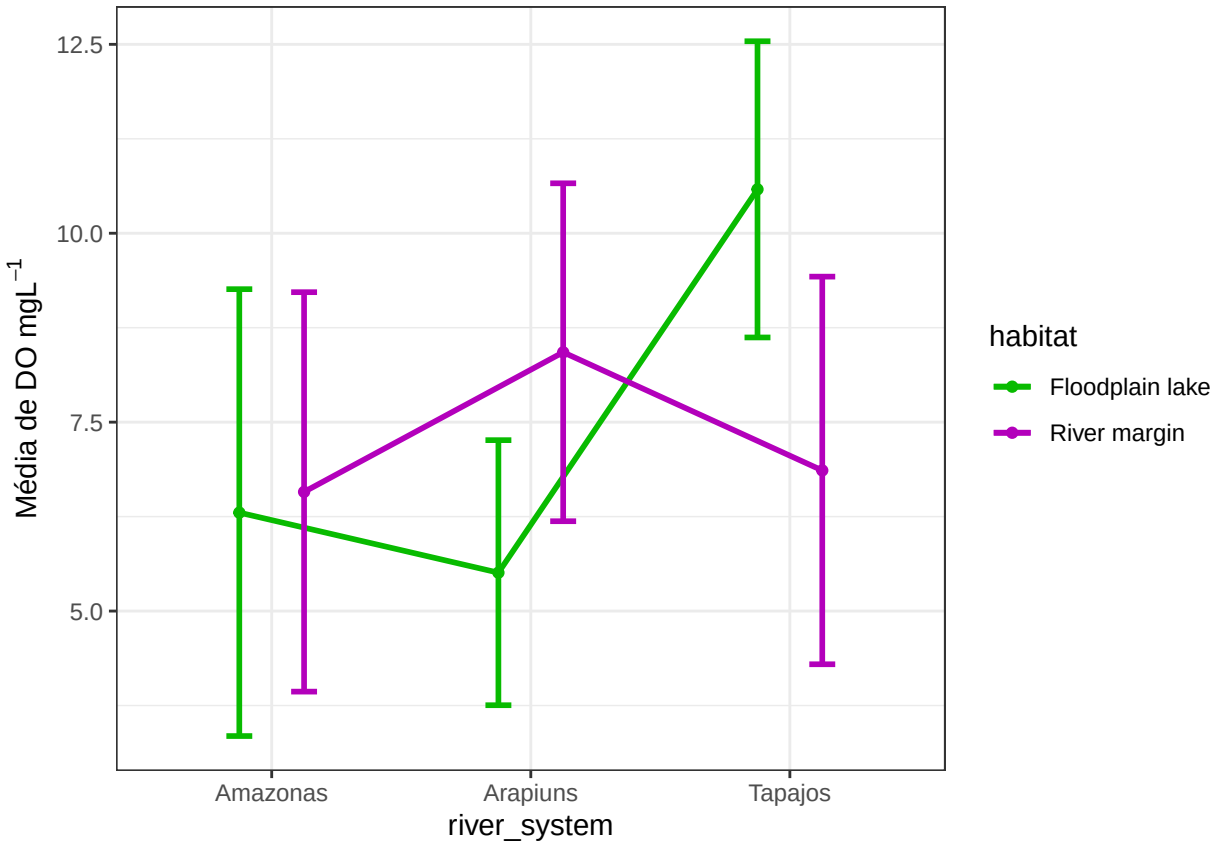



```
# Sumário estatístico de Do
sumario_do <-  dados_limno |>
  group_by(river_system, habitat) |>
  select(do) |>
  get_summary_stats(type = "common")
```

Do

```
# Visualizando
plot_do <-  ggplot(sumario_do, aes( x = river_system,
                                    y = mean, col = habitat,
                                    group = habitat)) +
  geom_line(linewidth = 1, position = position_dodge(.5)) +
  geom_point(position = position_dodge(0.5)) +
  geom_errorbar(
    aes(ymin = mean - sd, ymax = mean + sd),
    linewidth = 1,
    width = .2,
    position = position_dodge(0.5)
  ) +
  theme_bw() +
  labs(y = bquote("Média de DO mgL"^-1)) +
  scale_color_manual(values = c("#08bb00",  "#b300bb"))

plot_do
```

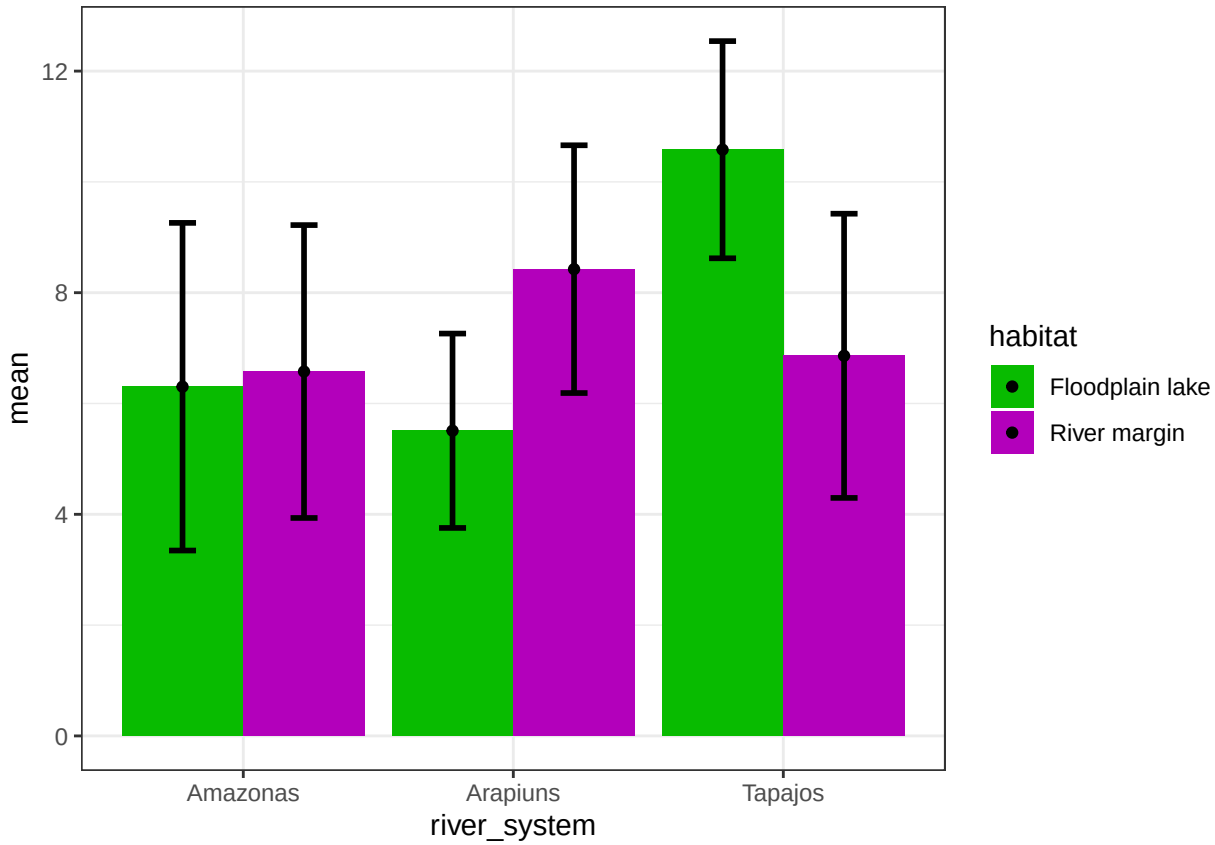
Visualizando o resultado



Os resultados podem ser apresentados em um gráfico de barras com erros

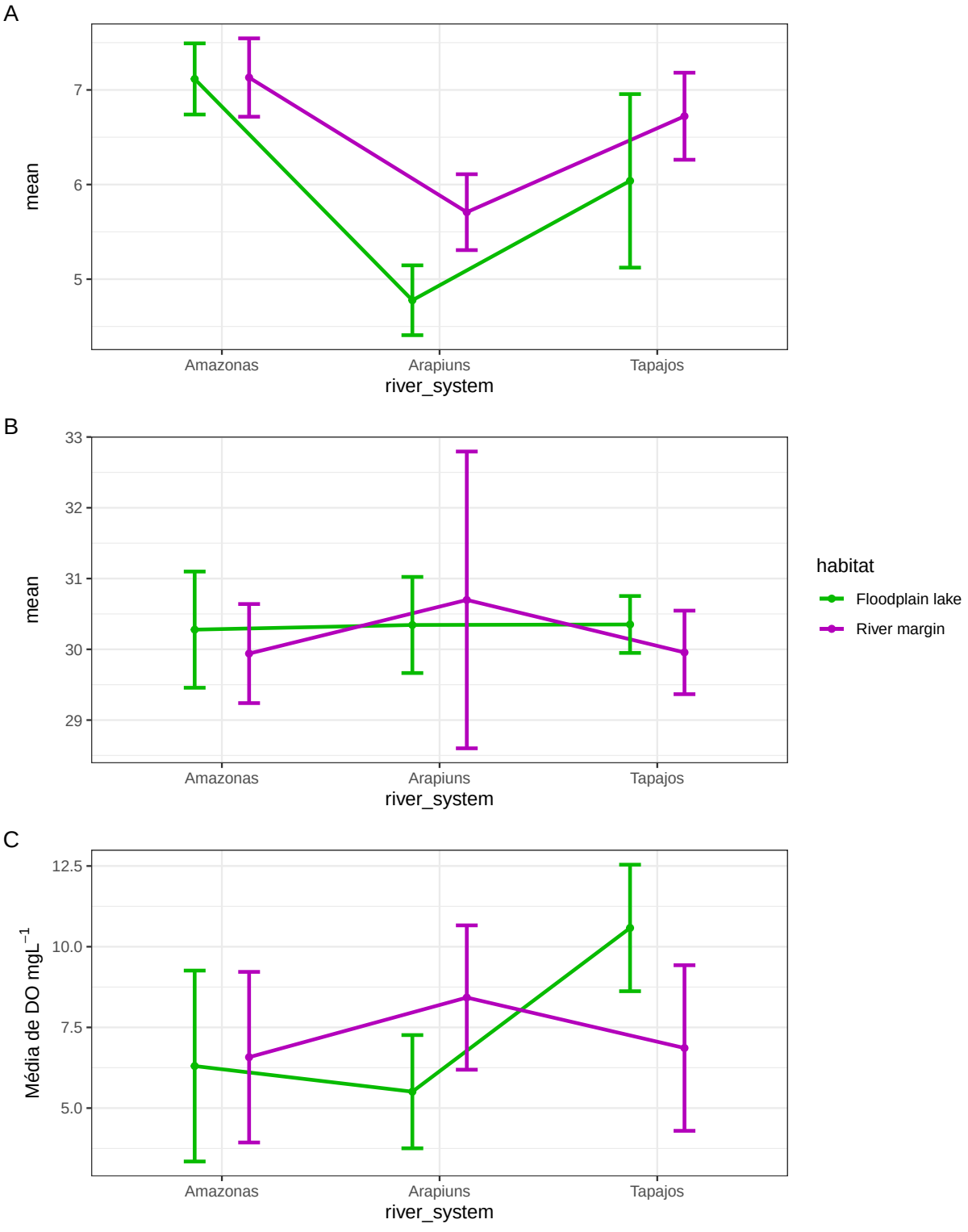
```
# Gráfico de barras com erros
ggplot(sumario_do, aes( x = river_system,
                        y = mean, fill = habitat,
                        group = habitat)) +
#  geom_line(linewidth = 1, position = position_dodge(.5)) +
```

```
  geom_bar(stat="identity", position=position_dodge())+
  geom_errorbar(
    aes(ymin = mean - sd, ymax = mean + sd),
    linewidth = 1,
    width = .2,
    position = position_dodge(0.9)
  )+
  geom_point(position = position_dodge(0.9)) +
  theme_bw() +
  scale_fill_manual(values = c("#08bb00",   "#b300bb"))
```




```
plot_ph/ plot_temp / plot_do + plot_layout(guides = "collect")+
  plot_annotation(tag_levels = "A")
```

Juntando gráficos




```
# Salvando o gráfico
ggsave('resultados/plot.jpeg', width = 15, height = 10, units = "cm", dpi = 300)
```

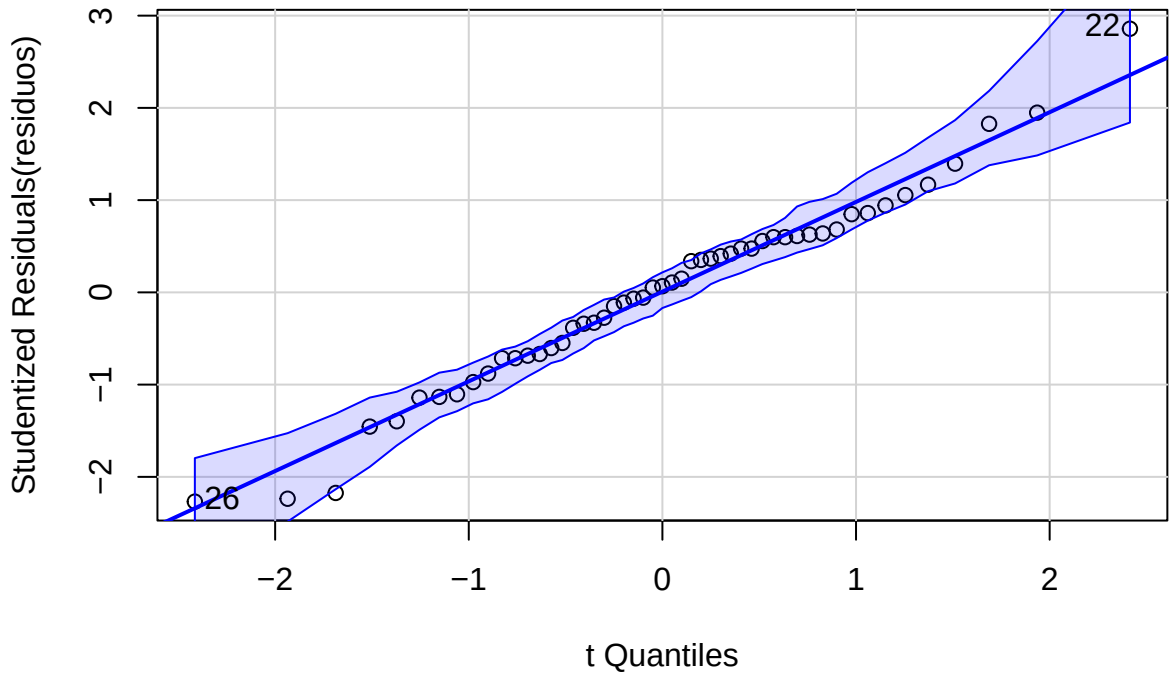
Testes estatisticos

Teste t **Pergunta:** Será se o comprimento padrão da espécie é diferente por período?

```
# Teste t ----------------------------------------------------------

dados_peixes <- read_csv2('dados/teste_t.csv')
```

```
# Normalidade

# Teste de normalidade
residuos <- lm(cp ~ as.factor(periodo), data = dados_peixes)

# visualização dos resíduos
qqPlot(residuos)
```

Checando a normalidade




```
## [1] 22 26
```

```
## Teste de Shapiro-Wilk
residuos_modelo <- residuals(residuos)
shapiro.test(residuos_modelo)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuos_modelo
```

```
## W = 0.98307, p-value = 0.6746
```

```
# Homocedasticidade

## Teste de homogeneidade de variância
leveneTest(cp ~ as.factor(periodo), data = dados_peixes)
```

Homocedasticidade

```
## Levene's Test for Homogeneity of Variance (center = median)
##       Df F value Pr(>F)
## group  1  1.1677 0.2852
##       49
```

```
# Teste T
t.test(cp ~ periodo, data = dados_peixes, var.equal = TRUE)
```

Resultado do teste T

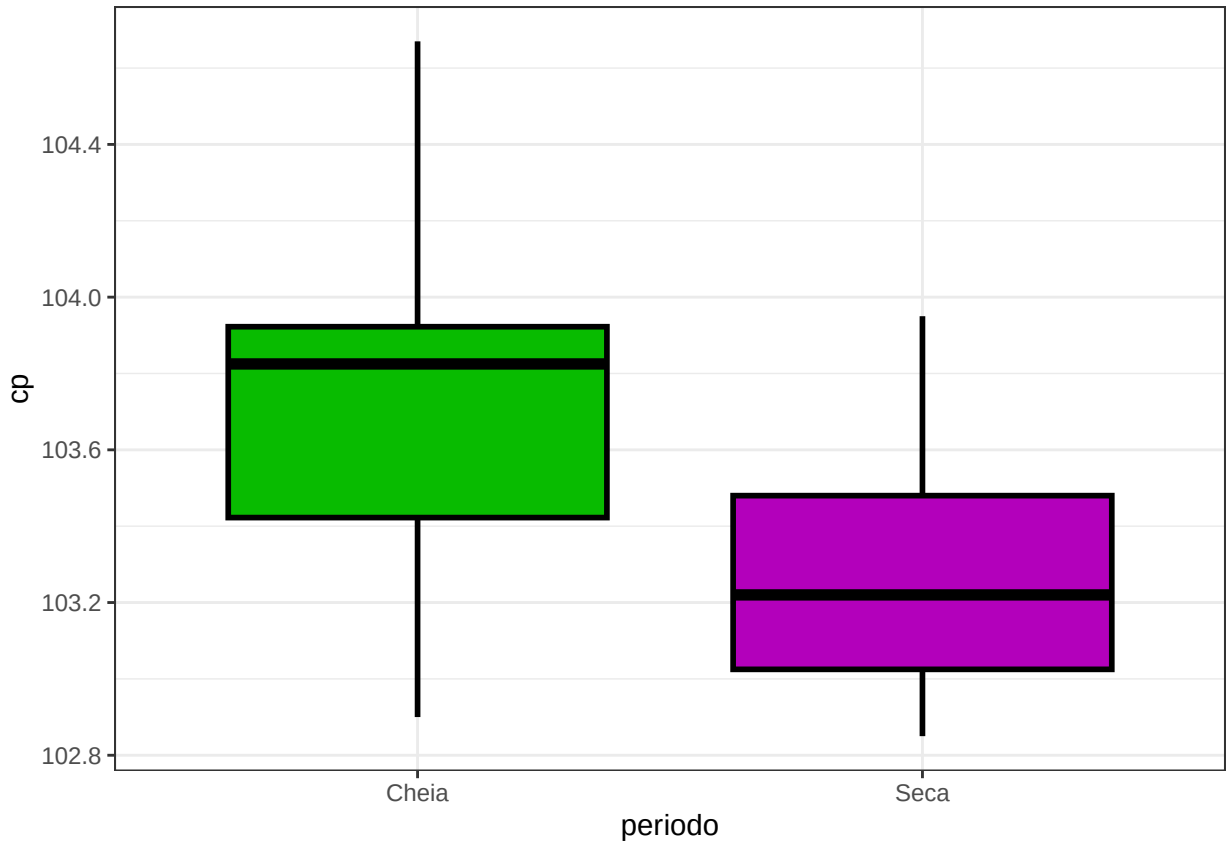
```
##
##  Two Sample t-test
##
## data:  cp by periodo
## t = 4.1524, df = 49, p-value = 0.000131
## alternative hypothesis: true difference in means between group Cheia and group Seca is not equal to 0
## 95 percent confidence interval:
##  0.2242132 0.6447619
## sample estimates:
## mean in group Cheia  mean in group Seca
##            103.6954            103.2609
```

```
# ou
dados_peixes |> t_test(cp ~ periodo,  var.equal = TRUE)
```

```
## # A tibble: 1 x 8
##   .y.   group1 group2    n1    n2 statistic    df        p
## * <chr> <chr>  <chr>  <int> <int>     <dbl> <dbl>    <dbl>
## 1 cp    Cheia  Seca      28    23      4.15    49 0.000131
```

```
ggplot(dados_peixes, aes(x = periodo, y = cp))+
  geom_boxplot(fill = c("#08bb00",  "#b300bb"), color = "black",
               linewidth = 1) +
  scale_color_manual(values = c("black", "black")) +
  theme_bw() +
  theme(legend.position = "none")
```

Visualizando o resultado



Regressão **Pergunta:** A temperatura afeta no tamanho da espécie?

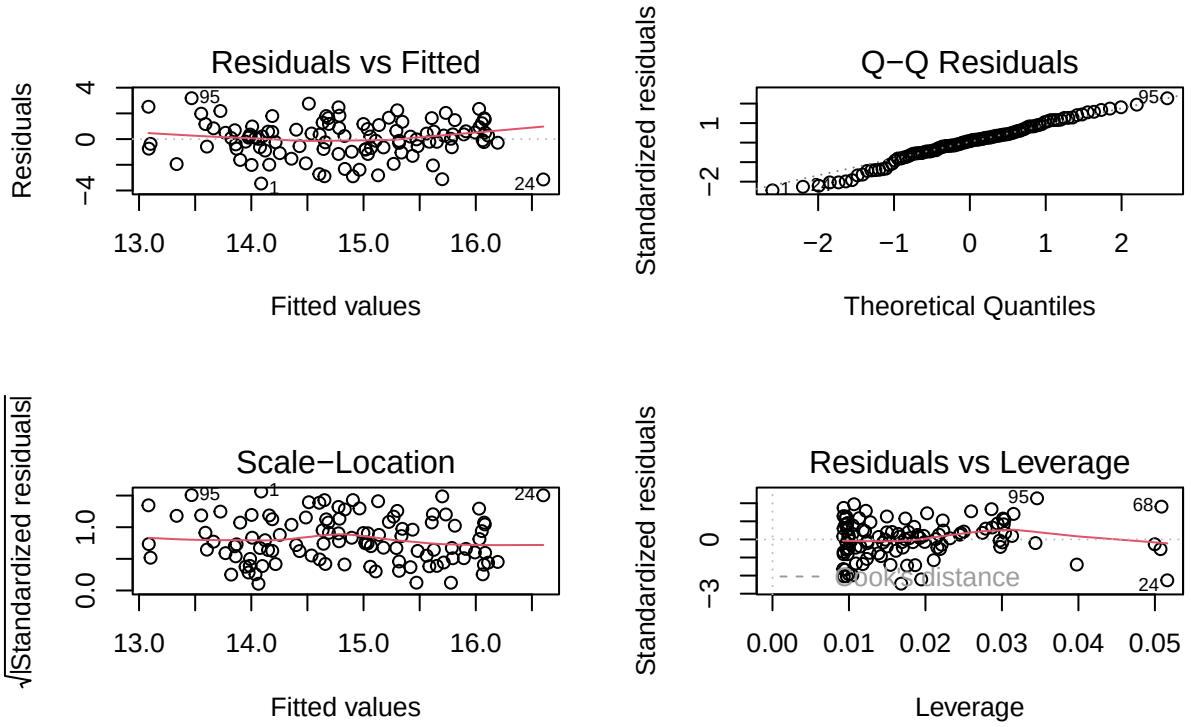
```
# Regressão ----------------------------------------------------

regressao <- read_csv2('dados/regressao.csv')
```

```
# modelo
```

Modelo

```
modelo_regressao <- lm(cp ~ temp, data = regressao)
## Verificar as premissas do teste
par(mfrow = c(2, 2), oma = c(0, 0, 2, 0))
plot(modelo_regressao)
```

Verificar as premissas




```
## Teste de Shapiro-Wilk
residuos_modelo <- residuals(modelo_regressao)
shapiro.test(residuos_modelo)
```

Normalidade

```
##
##  Shapiro-Wilk normality test
##
## data:  residuos_modelo
## W = 0.98367, p-value = 0.2092
```

```
## Resultados usando a função anova
anova(modelo_regressao)
```

Resultados

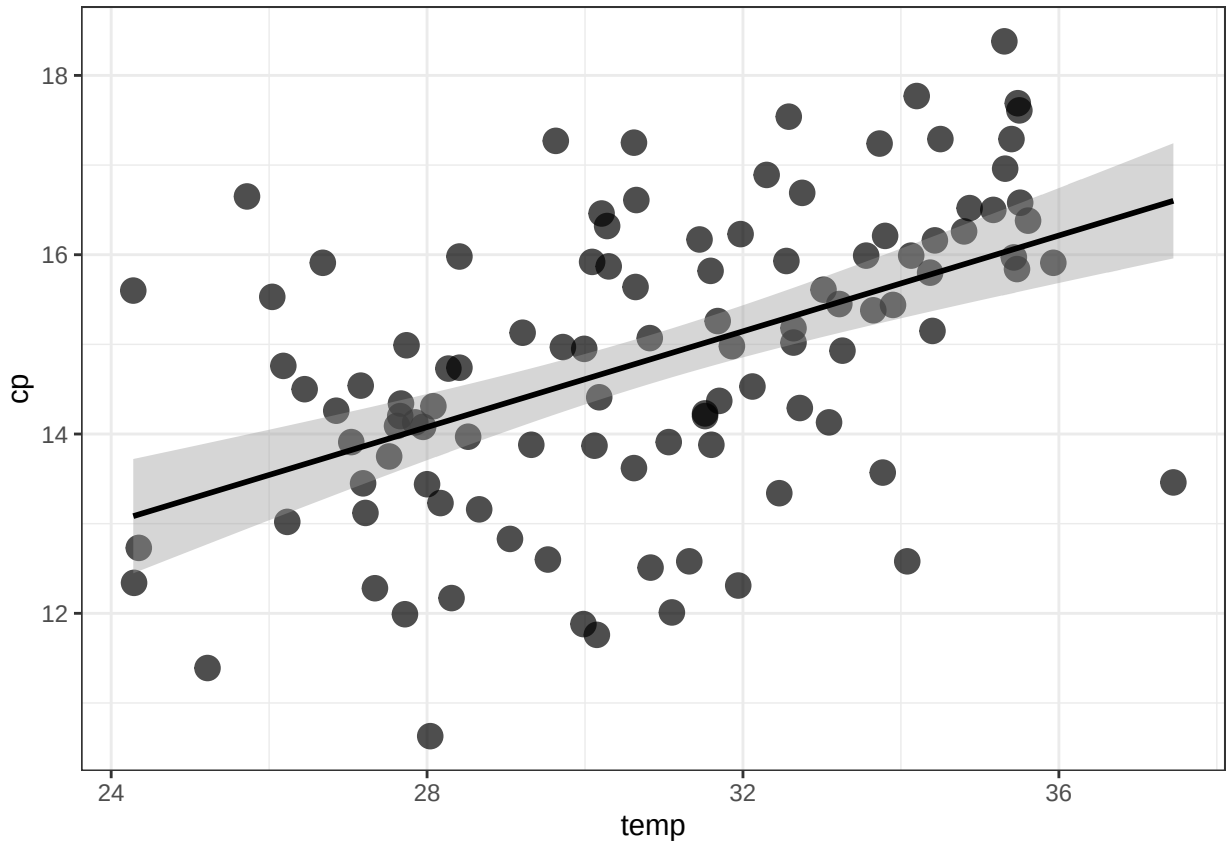
```
## Analysis of Variance Table
##
## Response: cp
##            Df  Sum Sq Mean Sq F value    Pr(>F)
## temp        1  73.677  73.677   36.25 2.533e-08 ***
## Residuals 106 215.443   2.032
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
## Resultados usando a função summary
summary(modelo_regressao)
```

```
##
## Call:
## lm(formula = cp ~ temp, data = regressao)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4581 -0.7552  0.1034  0.8751  3.1815
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.59948    1.37465   4.801 5.21e-06 ***
## temp         0.26707    0.04436   6.021 2.53e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.426 on 106 degrees of freedom
## Multiple R-squared:  0.2548, Adjusted R-squared:  0.2478
## F-statistic: 36.25 on 1 and 106 DF,  p-value: 2.533e-08
```

```r
ggplot(regressao, aes(x = temp, y = cp))+
  geom_point(size = 4, shape = 19, alpha = 0.7)+
  geom_smooth(method = "lm", se = TRUE, color = "black")+
  theme_bw()
```

Visualizando o resultado



Anova **Pergunta:** A riqueza de espécie é diferente entre os períodos

```
# Anova ------------------------------------------------------
dados_anova <- read.csv2("dados/dados_anova.csv")
```

```
modelo <- aov(riqueza ~ periodo, data = dados_anova)
```

Modelo

```
## Normalidade
shapiro.test(residuals(modelo))
```

Normalidade

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(modelo)
## W = 0.97289, p-value = 0.2016
```

```
## Homogeneidade da variância
bartlett.test(riqueza ~ periodo, data = dados_anova)
```

Homogeneidade da variância

```
##
##  Bartlett test of homogeneity of variances
##
## data:  riqueza by periodo
## Bartlett's K-squared = 15.657, df = 3, p-value = 0.001333
```

```
# ou
leveneTest(riqueza ~ as.factor(periodo), data = dados_anova)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##       Df F value   Pr(>F)
## group  3  4.3277 0.008183 **
##       56
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Quando a Homogeneidade é violada é feita a Anova de Welch

```
# Anova de Welch
oneway.test(riqueza ~ periodo, data = dados_anova)
```

```
##
##  One-way analysis of means (not assuming equal variances)
##
## data:  riqueza and periodo
## F = 72.755, num df = 3.000, denom df = 29.307, p-value = 1.099e-13
```

```
# Resultado bonitinho
welch_anova_test(dados_anova, riqueza ~ periodo)
```

```
## # A tibble: 1 x 7
##   .y.         n statistic   DFn   DFd        p method
## * <chr>   <int>     <dbl> <dbl> <dbl>    <dbl> <chr>
## 1 riqueza    60      72.8     3  29.3 1.10e-13 Welch ANOVA
```

Diferenças entre os períodos

```
## Diferenças entre os períodos
TukeyHSD(modelo)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = riqueza ~ periodo, data = dados_anova)
##
## $periodo
##                         diff        lwr       upr     p adj
## Enchente-Cheia    -5.933333 -24.235434  12.36877 0.8260846
## Seca-Cheia        81.733333  63.431233 100.03543 0.0000000
## Vazante-Cheia     26.466667   8.164566  44.76877 0.0018098
## Seca-Enchente     87.666667  69.364566 105.96877 0.0000000
## Vazante-Enchente  32.400000  14.097900  50.70210 0.0001047
## Vazante-Seca     -55.266667 -73.568767 -36.96457 0.0000000
```

```
# Resultado bonitinho
tukey_hsd(modelo)
```

```
## # A tibble: 6 x 9
##   term    group1   group2   null.value estimate conf.low conf.high    p.adj
```

```
## * <chr>   <chr>    <chr>         <dbl>    <dbl>    <dbl>     <dbl>    <dbl>
## 1 periodo Cheia    Enchente          0    -5.93   -24.2       12.4 8.26e- 1
## 2 periodo Cheia    Seca              0    81.7     63.4      100.  7.32e-12
## 3 periodo Cheia    Vazante           0    26.5      8.16      44.8 1.81e- 3
## 4 periodo Enchente Seca              0    87.7     69.4      106.  7.31e-12
## 5 periodo Enchente Vazante           0    32.4     14.1       50.7 1.05e- 4
## 6 periodo Seca     Vazante           0   -55.3    -73.6      -37.0 4.77e-10
## # i 1 more variable: p.adj.signif <chr>
```

Abordagem não paramétrica

```
# Não paramétrico
kruskal_test(dados_anova, riqueza ~ periodo)
```

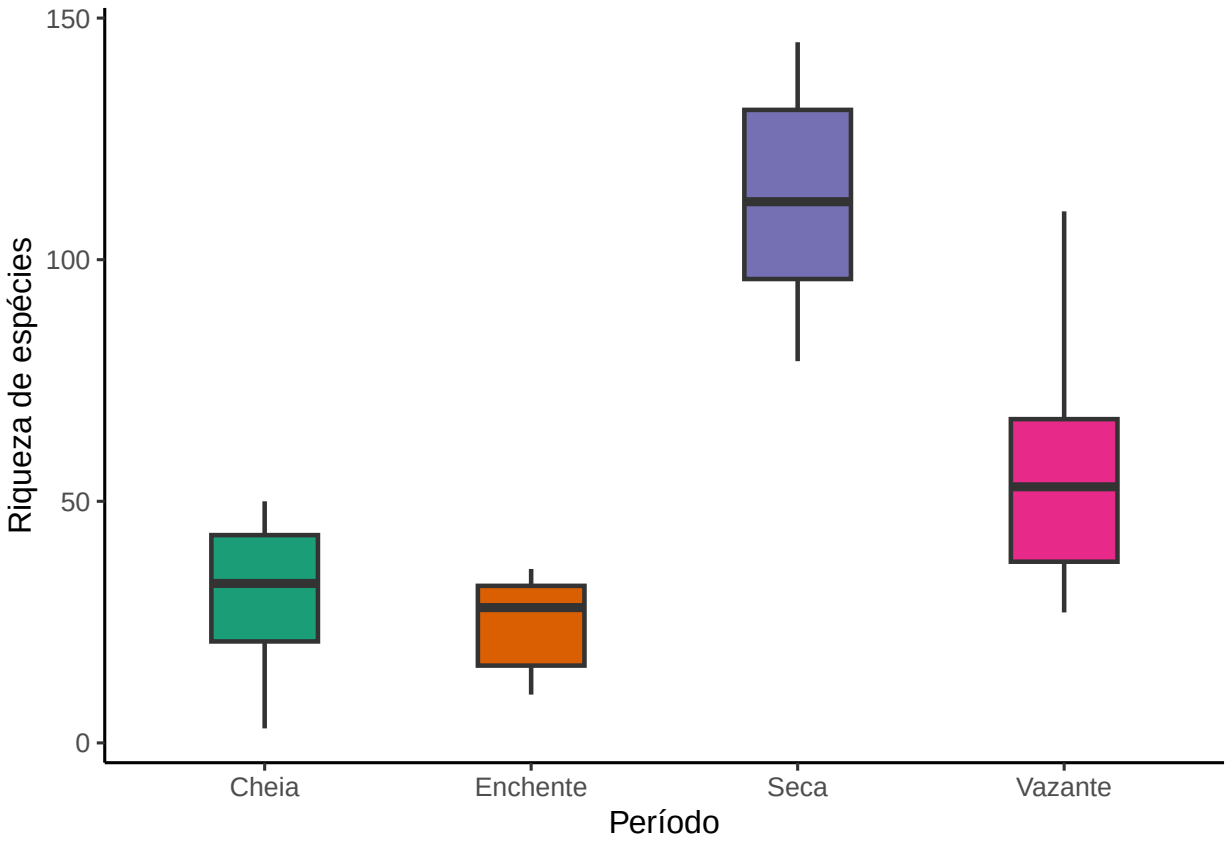
```
## # A tibble: 1 x 6
##   .y.         n statistic    df             p method
## * <chr>   <int>     <dbl> <int>         <dbl> <chr>
## 1 riqueza    60      40.4     3 0.00000000878 Kruskal-Wallis
```

```
dunn_test(dados_anova, riqueza ~ periodo)
```

```
## # A tibble: 6 x 9
##   .y.     group1   group2      n1    n2 statistic         p   p.adj p.adj.signif
## * <chr>   <chr>    <chr>    <int> <int>     <dbl>     <dbl>   <dbl> <chr>
## 1 riqueza Cheia    Enchente    15    15    -0.957   3.39e-1 3.39e-1 ns
## 2 riqueza Cheia    Seca        15    15     4.89    1.01e-6 5.07e-6 ****
## 3 riqueza Cheia    Vazante     15    15     2.15    3.12e-2 6.24e-2 ns
## 4 riqueza Enchente Seca        15    15     5.85    5.04e-9 3.02e-8 ****
## 5 riqueza Enchente Vazante     15    15     3.11    1.86e-3 7.45e-3 **
## 6 riqueza Seca     Vazante     15    15    -2.73    6.24e-3 1.87e-2 *
```

```
ggplot(dados_anova, aes(x = periodo, y = riqueza, fill = periodo))+
  geom_boxplot(show.legend = FALSE, width = .4, linewidth = .8)+
  theme_classic(base_size = 12)+
  labs(x = "Período", y = "Riqueza de espécies")+
  scale_fill_brewer(palette = "Dark2")
```

Visualizando o resultado


150
100
50
0
Riqueza de espécies
Cheia
Enchente
Seca
Vazante
Período