

# Introdução ao Ambiente R de Programação

Fabio Cop      [fabiocopf@gmail.com](mailto:fabiocopf@gmail.com)  
Instituto do Mar, Universidade Federal de São Paulo

2021-05-18



# Contents

|  |           |
|--|-----------|
| <b>Apresentação</b>  | <b>5</b>  |
| <b>I Fundamentos de programação em R</b>                           | <b>7</b>  |
| <b>1 Introdução ao programa R</b>                                  | <b>9</b>  |
| 1.1 Instalação do R . . . . .                                      | 9         |
| 1.2 Interface de desenvolvimento (IDE) . . . . .                   | 9         |
| 1.3 O R para cálculos aritméticos . . . . .                        | 10        |
| 1.4 Atribuição de valores . . . . .                                | 12        |
| 1.5 Estruturas de dados . . . . .                                  | 13        |
| 1.6 Operadores relacionais . . . . .                               | 26        |
| 1.7 Operadores lógicos . . . . .                                   | 28        |
| <b>2 (Básico da) Manipulação de data frames</b>                    | <b>31</b> |
| 2.1 Iniciando uma seção de trabalho . . . . .                      | 31        |
| 2.2 Importando arquivos .csv . . . . .                             | 32        |
| 2.3 Seleção de linhas e colunas em data frames . . . . .           | 33        |
| 2.4 Adicionando novas colunas a um data frame . . . . .            | 36        |
| 2.5 Aplicando uma função às linhas ou colunas de um data frame . . | 38        |
| 2.6 Exportando um data frame . . . . .                             | 41        |

|           |   |           |
|-----------|---|-----------|
| <b>3</b>  | <b>(Básico da) Visualização gráfica: pacote graphics</b>      | <b>43</b> |
| 3.1       | Doubs river dataset . . . . .                                 | 44        |
| 3.2       | Descrevendo os padrões de uma variável . . . . .              | 46        |
| 3.3       | Visualizando associações entre duas variáveis . . . . .       | 52        |
| 3.4       | Compreendendo o ambiente por meio de suas variáveis . . . . . | 59        |
| 3.5       | Mais um comentário sobre formatação gráfica no R . . . . .    | 65        |
| <b>II</b> | <b>Uma coleção integrada de pacotes para ciência de dados</b> | <b>69</b> |
| <b>4</b>  | <b>Os pacotes em tidyverse</b>                                | <b>71</b> |
| 4.1       | Carregando os pacotes . . . . .                               | 72        |
| <b>5</b>  | <b>Importando/Exportando dados</b>                            | <b>73</b> |
| 5.1       | Ajustando o diretório de trabalho . . . . .                   | 73        |
| 5.2       | Importando dados em arquivos de texto . . . . .               | 74        |
| <b>6</b>  | <b>Operador Pipe</b>  | <b>75</b> |
| <b>7</b>  | <b>Manipulação e formatação de dados</b>                      | <b>77</b> |
| 7.1       | Transformação de dados: os pacotes dplyr e tidyr . . . . .    | 77        |
| <b>8</b>  | <b>Um gráfico em camadas: o pacote ggplot2</b>                | <b>91</b> |
| 8.1       | Hubbard Brook stream flow . . . . .                           | 92        |
| 8.2       | Gráfico de dispersão . . . . .                                | 95        |
| 8.3       | Boxplot . . . . .   | 98        |
| 8.4       | Gráfico de linhas . . . . .                                   | 100       |
| 8.5       | Gráfico de barras . . . . .                                   | 102       |
| 8.6       | Temas no ggplot2 . . . . .                                    | 104       |
| 8.7       | Salvando uma figura gerada pelo ggplot2. . . . .              | 107       |

# Apresentação

Este é um material de introdução sobre o ambiente de programação R. O material está dividido em duas partes: I. Fundamentos de programação em R (capítulos 1 a 3) e II. Uma coleção integrada de pacotes para ciência de dados (capítulos 4 a 8).



## Part I

# Fundamentos de programação em R





# Chapter 1

## Introdução ao programa R

O R é um ambiente de análise de dados, cálculos matemáticos, análises estatísticas e apresentação gráfica. É um programa livre e de código aberto com aplicações nas mais diversas áreas das ciências. Para informações sobre o programa acesse a página oficial do R <https://www.r-project.org>, onde existe uma extensa variedade de informações, apostilas e área de download para as plataformas Linux, Windows e MacOS. Outras fontes de informação são o R-bloggers <https://www.r-bloggers.com>, o stackoverflow, towards data science ou ainda inúmeras páginas, tutoriais, videos e apostilas que podem ser encontrados em uma busca no Google. Neste capítulo faremos uma breve introdução ao R e, no capítulo seguinte, abordaremos tópicos sobre manipulação e visualização de dados.

### 1.1 Instalação do R

O R pode ser instalado a partir do site oficial CRAN. A instalação em ambiente Windows pode ser obtida no link **Download R for Windows --> base** onde você irá encontrar a versão mais atual disponível para seu sistema. Em ambiente Mac OS X a instalação pode ser feita a partir do link **Download R for (Mac) OS X** buscando pelo pacote **.pkg**. Para ambiente Linux, entre no link **Download R for Linux** e escolha sua plataforma (Ubuntu, Debian, Fedora, etc.). Nela você poderá obter as instruções de instalação.

### 1.2 Interface de desenvolvimento (IDE)

Uma IDE (*Integrated Development Environment*) é um ambiente de desenvolvimento integrado, que facilitam o uso de linguagens de programação. Existe

uma variedade de IDE's para programação em R. Você pode pesquisar por estas opções na internet e buscar por aquela que melhor atenda suas necessidades. Resalto duas opções utilizadas no contexto de análise de dados, estatística e ciência de dados: o RStudio e o Jupyter Notebook.

O RStudio é um ambiente integrado ao R, embora também seja utilizado para desenvolvimento em outras linguagens de programação. Por ter sido desenvolvido primariamente para desenvolvimento em R, oferece uma grande variedade de opções, desde a simples visualização de dados até a instalação e manutenção de pacotes. Esta será a interface que iremos utilizar para dar sequência a este material.

A instalação do RStudio pode ser feita em ambiente Windows, MAC OS X e Linux. A instalação em Windows e MAC OS X seguem o caminho tradicional de instalação de programas nestes ambientes. Em Linux você poderá encontrar uma variedade de tutoriais para a sua plataforma.

A vantagem do Jupyter Notebook é sua estrutura simples que facilita a curva de aprendizado. Atualmente (2021-05-18) é mais utilizado para desenvolvimento e análise de dados em Python, porém você poderá encontrar facilmente tutoriais para configuração do ambiente em R.

### 1.3 O R para cálculos aritméticos

Vamos iniciar nossa introdução ao R com seu uso mais simples, um ambiente para cálculos aritméticos. Como você verá, o R usa os operadores matemáticos de subtração ( $-$ ), adição ( $+$ ), multiplicação ( $*$ ), divisão ( $/$ ) e potenciação ( $^$ ) do modo análogo a outros softwares.

```
2 + 4
```

```
## [1] 6
```

```
2 * 4
```

```
## [1] 8
```

```
2 - 4
```

```
## [1] -2
```

```
2^4
```

```
## [1] 16
```

Além destes, temos operadores para extrairmos a parte inteira (%) e o resto (%%) de uma divisão.

```
13%/2
```

```
## [1] 6
```

```
13%%2
```

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

O uso de parênteses também permite o controle das operações matemáticas seguindo as prioridades conhecidas nestas operações. Por exemplo, a expressão:

```
5 * (9 + 2)
```

```
## [1] 55
```

é diferente de:

```
5 * 9 + 2
```

```
## [1] 47
```

Assim como a expressão:

```
(3 + 4)^2
```

```
## [1] 49
```

é diferente de:

```
3 + 4^2
```

```
## [1] 19
```

Existem também funções aritméticas comuns como  $\log(x)$ ,  $\sqrt{x}$ ,  $\sin(x)$ , o número  $\pi$ , etc.

```
log(100)
```

```
## [1] 4.60517
```

```
log10(100)
```

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

```
log(100, base = 2)
```

```
## [1] 6.643856
```

```
sqrt(36)
```

```
## [1] 6
```

```
pi
```

```
## [1] 3.141593
```

```
sin(0.5 * pi)
```

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

## 1.4 Atribuição de valores

O R se estrutura por meio de **objetos**. Ao atribuir um valor qualquer a uma variável, esta se torna um **objeto** que fica disponível na memória. Podemos, por exemplo, criar uma variável `x` e atribuir o valor 2 a esta variável.

```
x <- 2  
x
```

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

Após atribuir um valor a uma variável, esta fica disponível na memória da seção atual, de modo que podemos utilizá-la em cálculos subsequentes.

```
y <- x + 10  
y
```

```
## [1] 12
```

Ao atribuir outro valor à mesma variável, o valor inicial é substituído:

```
x <- 5  
y <- x + 10  
y
```

```
## [1] 15
```

O R diferencia caracteres minúsculos de MAIÚSCULOS. Portanto,

```
a <- sqrt(49)  
a
```

```
## [1] 7
```

```
A <- sqrt(81)  
A
```

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

## 1.5 Estruturas de dados

Os objetos em R podem ser de dos seguintes tipos: **vetores** (numéricos, alfanuméricos ou fatores), **matrizes** (numéricas ou alfanuméricas), **data frames** (estrutura bidimensional que pode combinar nas suas colunas vetores numéricos, alfanuméricos ou fatores) ou **listas** (que pode combinar em sua estrutura, todos os objetos descritos acima) e **funções** (comandos que realizam operações de transformação de objetos).

### 1.5.1 Vetores numéricos

Os objetos podem guardar mais de um único valor. A função concatenar `c()` pode ser utilizada para criar um vetor com múltiplos valores. Dizemos que cada valor individual é uma *entrada*.

```
x <- c(4, 3.0, 5, 9, 10)
x
```

```
## [1] 4 3 5 9 10
```

Podemos utilizar estes em nossas operações.

```
y <- x * 2
y
```

```
## [1] 8 6 10 18 20
```

Note que na operação acima, cada entrada foi multiplicada por 2.

Podemos ainda acessar e modificar entradas individuais. Por exemplo, o objeto `y` criado acima tem 5 elementos. O segundo elemento pode ser acessado com o comando:

```
y[2]
```

```
## [1] 6
```

E alterado com o comando:

```
y[2] <- 300
y
```

```
## [1] 8 300 10 18 20
```

Se quisermos excluir o quarto elemento de `y` e gravar o resultado em um novo objeto `z` fazemos:

```
z <- y[-4]
z
```

```
## [1] 8 300 10 20
```

*Obs:* Veja que o quarto elemento, 18, foi excluído.

Podemos obter a informação sobre o número de elementos do vetor. O vetor `y` tem tamanho igual a 5, enquanto o vetor `z` tem 4 elementos.

```
length(y)
```

```
## [1] 5
```

```
length(z)
```

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

#### 1.5.1.1 Sequências regulares e repetições

Podemos criar sequencias regulares.

```
2:10
```

```
## [1] 2 3 4 5 6 7 8 9 10
```

```
seq(2, 10, by = 2)
```

```
## [1] 2 4 6 8 10
```

```
seq(2, 10, length = 4)
```

```
## [1] 2.000000 4.666667 7.333333 10.000000
```

```
seq(2, 10, length = 10)
```

```
## [1] 2.000000 2.888889 3.777778 4.666667 5.555556 6.444444 7.333333  
## [8] 8.222222 9.111111 10.000000
```

E repetições de valores e vetores.

```
rep(4, times = 6)
```

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

```
rep(c(2, 5), times = 3)
```

```
## [1] 2 5 2 5 2 5
```

```
rep(c(2, 5), each = 3)
```

```
## [1] 2 2 2 5 5 5
```

Os resultados destas sequências podem ser guardadas em um objeto para utilização subsequente.

```
a <- seq(2, 10, by = 2)
a
```

```
## [1] 2 4 6 8 10
```

```
b <- seq(10, 2, by = -2)
b
```

```
## [1] 10 8 6 4 2
```

```
c <- a + b
c
```

```
## [1] 12 12 12 12 12
```

### 1.5.2 Vetores alfanuméricos

São vetores em que cada entrada é um caracter alfanumerico.

```
especie = c("Deuterodon iguape",
            "Characidium japyhybense",
            "Trichomycterus zonatus")
especie
```

```
## [1] "Deuterodon iguape"      "Characidium japyhybense"
## [3] "Trichomycterus zonatus"
```

Existe uma variedade de funções para lidarmos e manipularmos vetores alfanuméricos.

A função `sort()` por exemplo, se aplicada a um vetor *numérico* é utilizada para ordená-lo de forma crescente:



```
a = c(5,2,15,12)
a
```

```
## [1]  5  2 15 12
```

```
sort(a)
```

```
## [1]  2  5 12 15
```

ou decrescente:

```
sort(a, decreasing = T)
```

```
## [1] 15 12  5  2
```

Se aplicada a um vetor alfanumerico esta função ordena o vetor em ordem alfabética:

```
sort(especie, decreasing = FALSE)
```

```
## [1] "Characidium japuhybense" "Deuterodon iguape"
## [3] "Trichomycterus zonatus"
```

```
sort(especie, decreasing = TRUE)
```

```
## [1] "Trichomycterus zonatus" "Deuterodon iguape"
## [3] "Characidium japuhybense"
```

### 1.5.3 Unindo vetores: comando paste

Suponha que desejamos unir dois vetores alfanuméricos

```
x1 <- c("Experimento")
x2 <- c("A", "B", "C")
x3 <- paste(x1, x2, sep = "_")
```

O mesmo resultado pode ser obtido de forma mais concisa com o comando:

```
x4 <- paste("Experimento", LETTERS[1:3], sep = "_")
x4
```

```
## [1] "Experimento_A" "Experimento_B" "Experimento_C"
```

### 1.5.4 Fatores

Fatores são como vetores alfanuméricos, porém com um atributo adicional. Fatores são compostos por diferentes níveis. Por exemplo, podemos criar o objeto `dosagem` com o comando:

```
dosagem <- c("Alta", "Alta", "Alta",
             "Media", "Media", "Media",
             "Baixa", "Baixa", "Baixa")
dosagem
```

```
## [1] "Alta" "Alta" "Alta" "Media" "Media" "Media" "Baixa" "Baixa" "Baixa"
```

No exemplo acima, o R não reconhece as palavras `Alta`, `Media` e `Baixa` como diferentes níveis. Para isto devemos fazer:

```
dosagem <- factor(dosagem)
dosagem
```

```
## [1] Alta Alta Alta Media Media Media Baixa Baixa Baixa
## Levels: Alta Baixa Media
```

O objeto `dosagem` agora é um fator com 3 níveis.

```
levels(dosagem)
```

```
## [1] "Alta" "Baixa" "Media"
```

```
nlevels(dosagem)
```

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

```
levels(dosagem)[2]
```

```
## [1] "Baixa"
```

Note entretanto que os níveis foram reconhecidos em ordem alfabética. Se quisermos ordenar este níveis de outro modo fazemos:

```
dosagem <- factor(dosagem, ordered = T,
                  levels = c("Baixa", "Media", "Alta"))
dosagem

## [1] Alta  Alta  Alta  Media Media Media Baixa Baixa Baixa
## Levels: Baixa < Media < Alta
```

Como veremos a frente, esta operação pode facilitar a visualização gráfica de fatores ordenados.

### 1.5.5 Matrizes

Matrizes são objetos compostos por linhas e colunas. No R, uma matriz pode ser construída inicialmente criando um vetor numérico:

```
a <- c(21,26,5,18,17,28,20,15,13,14,27,22)
a
```

```
## [1] 21 26 5 18 17 28 20 15 13 14 27 22
```

e, em seguida, organizando este vetor em uma matriz com um número de linhas e colunas compatíveis com o tamanho do vetor. No exemplo acima temos um vetor de comprimento 12. Deste modo, este vetor pode ser organizado por exemplo, em uma matriz de 3 linhas e 4 colunas, utilizando o comando.

```
x <- matrix(a, nrow = 3, ncol = 4)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  21  18  20  14
## [2,]  26  17  15  27
## [3,]   5  28  13  22
```

Note que os elementos foram adicionados um por vez de **coluna em coluna**. Se quisermos preencher a matriz **por linhas** adicionamos ao comando, o argumento `byrow = TRUE`.

```
x <- matrix(a, nrow = 3, ncol = 4, byrow = TRUE)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   21   26    5   18
## [2,]   17   28   20   15
## [3,]   13   14   27   22
```

Os elementos de uma matriz podem ser acessados indicando sua posição na linha e na coluna. Por exemplo, o elemento da segunda linha e terceira coluna de `x` pode ser acessados pelo comando:

```
x[2, 3]
```

```
## [1] 20
```

De modo análogo, a segunda linha pode ser acessada por:

```
x[2, ]
```

```
## [1] 17 28 20 15
```

E a coluna 4 por:

```
x[, 4]
```

```
## [1] 18 15 22
```

Assim como fizemos com os vetores, podemos acessar e modificar valores individuais em matrizes. Por exemplo, se quisermos alterar o elemento segunda linha e terceira coluna de `x` por 1000 fazemos:

```
x[2, 3] <- 1000
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   21   26    5   18
## [2,]   17   28 1000   15
## [3,]   13   14   27   22
```

Também podemos excluir linhas e colunas de uma matriz.

```
x[-2,]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   21   26    5   18
## [2,]   13   14   27   22
```

```
x[, -3]
```

```
##      [,1] [,2] [,3]
## [1,]  21  26  18
## [2,]  17  28  15
## [3,]  13  14  22
```

Note que, acima, não salvamos os resultados da exclusão das linhas e colunas de `x` em nenhum objeto, de modo que `x` continua inalterado.

```
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  21  26   5  18
## [2,]  17  28 1000  15
## [3,]  13  14  27  22
```

Podemos criar matrizes unindo vetores de tamanho iguais em linhas ou colunas.

```
x <- 3:12
y <- 12:3
rbind(x, y)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## x      3   4   5   6   7   8   9  10  11  12
## y     12  11  10   9   8   7   6   5   4   3
```

```
cbind(x, y)
```

```
##      x y
## [1,] 3 12
## [2,] 4 11
## [3,] 5 10
## [4,] 6  9
## [5,] 7  8
## [6,] 8  7
## [7,] 9  6
## [8,] 10 5
## [9,] 11 4
## [10,] 12 3
```

Eventualmente, se desejarmos atribuir nomes às linhas e às colunas de uma matriz, podemos fazê-lo por meio das funções `rownames()` e `colnames()` respectivamente:

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

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
rownames(x_mat) <- paste("Linha", 1:3, sep = "")
x_mat
```

```
##      [,1] [,2] [,3] [,4]
## Linha1    1    4    7   10
## Linha2    2    5    8   11
## Linha3    3    6    9   12
```

```
colnames(x_mat) <- paste("Coluna", 1:4, sep = "")
x_mat
```

```
##      Coluna1 Coluna2 Coluna3 Coluna4
## Linha1      1      4      7      10
## Linha2      2      5      8      11
## Linha3      3      6      9      12
```

### 1.5.6 Data frames

Assim como Matrizes, **Data frames** são estruturas que permitem organizar dados em formato de linhas e colunas. No R entretanto, as Matrizes não podem guardar objetos de diferentes características. Por exemplo, uma matriz pode ser composta inteiramente numérica:

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

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

Ou alfanumérica:

```
matrix(letters[1:12], nrow = 4, ncol = 3)
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "e"  "i"
## [2,] "b"  "f"  "j"
## [3,] "c"  "g"  "k"
## [4,] "d"  "h"  "l"
```

Porém, se tentarmos unir um vetor numérico a um vetor alfanumérico, toda a matriz será convertida no formato alfanumérico.

```
z <- LETTERS[3:12]
z
```

```
## [1] "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"
```

```
cbind(x, z)
```

```
##      x      z
## [1,] "3"    "C"
## [2,] "4"    "D"
## [3,] "5"    "E"
## [4,] "6"    "F"
## [5,] "7"    "G"
## [6,] "8"    "H"
## [7,] "9"    "I"
## [8,] "10"   "J"
## [9,] "11"   "K"
## [10,] "12"  "L"
```

Para unir diferentes tipos de vetores devemos usar o comando `data.frame` que cria uma estrutura com colunas independentes, permitindo que estas colunas tenham diferentes formatos. Podemos unir os objetos `x` e `z` acima em um data frame como segue:

```
data.frame(x, z)
```

```
##      x z
## 1    3 C
## 2    4 D
## 3    5 E
## 4    6 F
```

```
## 5    7 G
## 6    8 H
## 7    9 I
## 8   10 J
## 9   11 K
## 10  12 L
```

Note que automaticamente, a função atribui nomes às colunas (*x* e *z*) e às linhas (1 a 10). Estes nomes podem ser alterados com as funções `rownames()` e `colnames()`.

Neste caso, a coluna *x* continua sendo numérica e a coluna *z* continua alfanumérica.

Podemos acessar os elementos de um data frame do mesmo modo que fizemos para matrizes. Além destas formas, podemos usar os seguintes comandos:

```
Dados <- data.frame(Regiao = factor(c("Santos", "Santos",
                                     "Bertioga", "Bertioga",
                                     "Peruibe", "Peruibe")),
                   Especie_A = c(12,43,80,91,75,115),
                   Especie_B = c(0, 59, 300, 350, 154, 200))
```

Dados

```
##      Regiao Especie_A Especie_B
## 1   Santos      12         0
## 2   Santos      43         59
## 3 Bertioga      80        300
## 4 Bertioga      91        350
## 5  Peruibe      75        154
## 6  Peruibe     115        200
```

Dados\$Regiao

```
## [1] Santos Santos Bertioga Bertioga Peruibe Peruibe
## Levels: Bertioga Peruibe Santos
```

Dados["Regiao"]

```
##      Regiao
## 1   Santos
## 2   Santos
## 3 Bertioga
## 4 Bertioga
## 5  Peruibe
## 6  Peruibe
```



```
Dados[, "Regiao"]
```

```
## [1] Santos Santos Bertioiga Bertioiga Peruibe Peruibe
## Levels: Bertioiga Peruibe Santos
```

```
Dados[, c("Especie_A", "Especie_B")]
```

```
##   Especie_A Especie_B
## 1      12         0
## 2      43         59
## 3      80        300
## 4      91        350
## 5      75        154
## 6     115        200
```

### 1.5.7 Listas

Combinam em um único objeto todas as estruturas anteriores. Veja o exemplo em que combinamos um vetor alfanumérico, um vetor nominal e um data frame dentro da mesma lista.

```
nossalista <- list(Ilha = c("Ilhabela", "Anchieta", "Cardoso"),
                  Areaskm2 = c(347.5, 8.3, 131),
                  Localizacao = data.frame(
                    Bioma = rep("Mata Atlantica", 3),
                    Lat = c(23, 25, 23),
                    Long = c(45, 47, 45)))
nossalista
```

```
## $Ilha
## [1] "Ilhabela" "Anchieta" "Cardoso"
##
## $Areaskm2
## [1] 347.5 8.3 131.0
##
## $Localizacao
##      Bioma Lat Long
## 1 Mata Atlantica 23 45
## 2 Mata Atlantica 25 47
## 3 Mata Atlantica 23 45
```

Podemos ainda inserir listas dentro de outras listas, criando estruturas altamente complexas.

Para acessar os elementos de uma lista podemos identificar seu nome após o operador `$` ou sua posição das formas que se seguem:

```
nossalista$Ilha
```

```
## [1] "Ilhabela" "Anchieta" "Cardoso"
```

```
nossalista[[1]]
```

```
## [1] "Ilhabela" "Anchieta" "Cardoso"
```

```
nossalista$Localizacao
```

```
##           Bioma Lat Long
## 1 Mata Atlantica  23   45
## 2 Mata Atlantica  25   47
## 3 Mata Atlantica  23   45
```

```
nossalista[[3]]
```

```
##           Bioma Lat Long
## 1 Mata Atlantica  23   45
## 2 Mata Atlantica  25   47
## 3 Mata Atlantica  23   45
```

## 1.6 Operadores relacionais

Operadores relacionais são aqueles que verificam as relações de **menor que** (`<`), **maior que** (`>`), **menor ou igual** (`<=`), **maior ou igual** (`>=`), **igual a** (`==`) ou **diferente de** (`!=`). O resultado de uma comparação retorna um objeto com o argumento **verdadeiro** (`TRUE`) ou **falso** (`FALSE`). Veja por exemplo:

```
3 > 5
```

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

```
3 > 3
```

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

```
3 >= 3
```

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

```
a <- 5  
b <- 7  
a == b
```

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

```
a != b
```

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

Se os objetos têm mais de um elemento, no caso de vetores, matrizes ou data frames, a comparação é feita elemento a elemento, comparando aqueles que estão na mesma posição, ou seja, os que têm o mesmo *índice de posição*.

```
a <- c(3,5,5,7,1)  
b <- c(3,6,1,9,-3)  
a < b
```

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

Os operadores TRUE e FALSE, quando utilizados em operações aritméticas se comportam respectivamente como valores 1 e 0.

```
a <- 5  
b <- c(3,6,1,9,-3)  
y <- a < b  
y
```

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

Somando os elementos de y temos:

```
sum(y)
```

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

E se tirarmos a média aritmética, teremos a *proporção* de 1's no vetor.

```
mean(y)
```

```
## [1] 0.4
```

- *Observação:* Lembre-se que ao compararmos vetores de tamanhos distintos, o R **não retorna um erro**, mas recicla os elementos do vetor menor para compensar elementos faltantes.

## 1.7 Operadores lógicos

Operadores lógicos são os de **NEGAÇÃO** (!), **E lógico**, versão vetorizada (&), **E lógico**, versão não-vetorizada (&&), **OU lógico** versão vetorizada (|), **OU lógico** versão não-vetorizada (||) e **OU exclusivo** (xor()). Exemplos destes operadores são:

```
x <- 3:5  
y <- 5:3
```

```
(x < 4)
```

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

```
!(x < 4)
```

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

```
(x < 4) & (y > 4)
```

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

```
(x < 4) && (y > 4)
```

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

```
(x < 4) | (y > 4)
```

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

```
(x < 4) || (y > 4)
```

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

```
xor(x,y)
```

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



## Chapter 2

# (Básico da) Manipulação de data frames

Ainda que seja possível criar um data frame entrando diretamente com os dados via linha de comando, é mais comum **importamos** tabelas a partir de arquivos `.csv`, `.txt` ou outros formatos. Ao importar estes tipos de arquivo, o R os lê como `data.frames`.

Inicialmente vamos introduzir métodos de preparação de uma nova seção em R e importação/exportação de tabelas.

### 2.1 Iniciando uma seção de trabalho

Seção no R, se refere ao ambiente em que ficam armazenados os objetos (vetores, matrizes, data frames, etc.) criados durante o processo de manipulação e análise de dados. Ao fechar uma seção do R (ex. ao sair do R Stúdio), esta pode ser salva guardando os objetos criados. O arquivo de uma seção é salvo com extensão `.RData`.

Ao abrir um novo *script* (com extensão `.r`) em um editor de texto é necessário definir o diretório em que iremos trabalhar. Este *diretório de trabalho* será o local de onde iremos importar dados, e para onde iremos salvar as figuras e tabelas criadas ao longo do trabalho. No R-Studio, um novo *script* pode ser aberto via menu **Arquivo --> Novo script**. Ao iniciar o R-Studio abre-se uma nova seção. O diretório desta seção pode ser verificado pelo comando:

```
getwd()
```

Crie uma pasta **IntroR** e direcione a seção de trabalho para esta pasta utilizando

a função `setwd("C:/seu_caminho/IntroR")` e verifique se houve mudança com a função `getwd()`

```
setwd("C:/seu_caminho/IntroR")
getwd()
```

```
## [1] "C:/seu_caminho/IntroR"
```

A partir deste momento o R irá ler e salvar arquivos sempre neste diretório.

## 2.2 Importando arquivos .csv

Importe o conjunto dados `dbenv.csv`. Abra o arquivo em algum editor de texto e veja as características deste arquivo. Você verá por exemplo que ele é composto por 11 variáveis (*colunas*) mensuradas em 30 pontos (*linhas*). Este conjunto de dados pode ser obtido Aqui em formato .csv. Após fazer o download, você pode importar o conjunto de dados por:

```
dbenv <- read.csv(file = "C:/seu_caminho/IntroR/dbenv.csv",
                  header = TRUE, dec = '.', sep = ',')
```

A função `read.csv` possui diferentes argumentos. A argumento `header` define se a primeira linha consiste do cabeçalho (`TRUE`) ou não (`FALSE`). O argumento `dec` define se o separador decimal consiste de *vírgula* ou *ponto* e o argumento `sep` informa sobre qual é o caracter separador de colunas utilizado no arquivo. No arquivo em questão as colunas são separadas por *vírgulas*. Outros tipos de separadores comuns são *ponto-e-vírgula* ou *tabulações*.

Veja os nomes das 11 variáveis (cabeçalho), a dimensão da tabela (número de linhas e colunas) e sua estrutura (um `data.frame` formado por 11 vetores numéricos).

```
dbenv
```

```
##      dfs alt    slo flo pH har pho nit amm oxy bdo
## 1      3 934 6.176  84 79 45   1 20   0 122 27
## 2     22 932 3.434 100 80 40   2 20  10 103 19
## 3    102 914 3.638 180 83 52   5 22   5 105 35
## 4    185 854 3.497 253 80 72  10 21   0 110 13
## 5    215 849 3.178 264 81 84  38 52  20  80 62
## 6    324 846 3.497 286 79 60  20 15   0 102 53
## 7    268 841 4.205 400 81 88   7 15   0 111 22
## 8    491 792 3.258 130 81 94  20 41  12  70 81
```



```
## 9 705 752 2.565 480 80 90 30 82 12 72 52
## 10 990 617 4.605 1000 77 82 6 75 1 100 43
## 11 1234 483 3.738 1990 81 96 30 160 0 115 27
## 12 1324 477 2.833 2000 79 86 4 50 0 122 30
## 13 1436 450 3.091 2110 81 98 6 52 0 124 24
## 14 1522 434 2.565 2120 83 98 27 123 0 123 38
## 15 1645 415 1.792 2300 86 86 40 100 0 117 21
## 16 1859 375 3.045 1610 80 88 20 200 5 103 27
## 17 1985 348 1.792 2430 80 92 20 250 20 102 46
## 18 2110 332 2.197 2500 80 90 50 220 20 103 28
## 19 2246 310 1.792 2590 81 84 60 220 15 106 33
## 20 2477 286 2.197 2680 80 86 30 300 30 103 28
## 21 2812 262 2.398 2720 79 85 20 220 10 90 41
## 22 2940 254 2.708 2790 81 88 20 162 7 91 48
## 23 3043 246 2.565 2880 81 97 260 350 115 63 164
## 24 3147 241 1.386 2976 80 99 140 250 60 52 123
## 25 3278 231 1.792 3870 79 100 422 620 180 41 167
## 26 3579 214 1.792 3910 79 94 143 300 30 62 89
## 27 3732 206 2.565 3960 81 90 58 300 26 72 63
## 28 3947 195 1.386 4320 83 100 74 400 30 81 45
## 29 4220 183 1.946 6770 78 110 45 162 10 90 42
## 30 4530 172 1.099 6900 82 109 65 160 10 82 44
```

```
colnames(dbenv)
```

```
## [1] "dfs" "alt" "slo" "flo" "pH" "har" "pho" "nit" "amm" "oxy" "bdo"
```

```
dim(dbenv)
```

```
## [1] 30 11
```

*Observação:* arquivos `.txt` podem ser lidos com a função `read.table`. Veremos a frente funções para leitura de outros formatos.

## 2.3 Seleção de linhas e colunas em data frames

No data frame que importamos as colunas têm nomes que podem ser acessados por:

```
colnames(dbenv)
```

```
## [1] "dfs" "alt" "slo" "flo" "pH" "har" "pho" "nit" "amm" "oxy" "bdo"
```

O nome das linhas podem ser acessados por:

```
rownames(dbenv)
```

```
## [1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12" "13" "14" "15"
## [16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29" "30"
```

Os números “entre aspas” significam que o R está lendo os nomes das linhas não como números, mas como caracteres.

Podemos utilizar colunas específicas deste por meio de seus nomes:

```
colunas <- c("dfs", "flo", "oxy")
dbenv[,colunas]
```

```
##      dfs flo oxy
## 1      3  84 122
## 2     22 100 103
## 3    102 180 105
## 4    185 253 110
## 5    215 264  80
## 6    324 286 102
## 7    268 400 111
## 8    491 130  70
## 9    705 480  72
## 10   990 1000 100
## 11  1234 1990 115
## 12  1324 2000 122
## 13  1436 2110 124
## 14  1522 2120 123
## 15  1645 2300 117
## 16  1859 1610 103
## 17  1985 2430 102
## 18  2110 2500 103
## 19  2246 2590 106
## 20  2477 2680 103
## 21  2812 2720  90
## 22  2940 2790  91
## 23  3043 2880  63
## 24  3147 2976  52
## 25  3278 3870  41
## 26  3579 3910  62
## 27  3732 3960  72
## 28  3947 4320  81
## 29  4220 6770  90
## 30  4530 6900  82
```

Também podemos acessá-las pela sua posição:

```
colunas_num <- c(1, 3, 4)
dbenv[,colunas_num]
```

```
##      dfs  slo flo
## 1      3 6.176  84
## 2     22 3.434 100
## 3    102 3.638 180
## 4    185 3.497 253
## 5    215 3.178 264
## 6    324 3.497 286
## 7    268 4.205 400
## 8    491 3.258 130
## 9    705 2.565 480
## 10   990 4.605 1000
## 11  1234 3.738 1990
## 12  1324 2.833 2000
## 13  1436 3.091 2110
## 14  1522 2.565 2120
## 15  1645 1.792 2300
## 16  1859 3.045 1610
## 17  1985 1.792 2430
## 18  2110 2.197 2500
## 19  2246 1.792 2590
## 20  2477 2.197 2680
## 21  2812 2.398 2720
## 22  2940 2.708 2790
## 23  3043 2.565 2880
## 24  3147 1.386 2976
## 25  3278 1.792 3870
## 26  3579 1.792 3910
## 27  3732 2.565 3960
## 28  3947 1.386 4320
## 29  4220 1.946 6770
## 30  4530 1.099 6900
```

O mesmo é válido para as linhas.

```
linhas <- c("3", "7", "9")
dbenv[linhas,]
```

```
##   dfs alt  slo flo pH har pho nit amm oxy bdo
## 3 102 914 3.638 180 83 52  5 22  5 105 35
```

```
## 7 268 841 4.205 400 81 88 7 15 0 111 22
## 9 705 752 2.565 480 80 90 30 82 12 72 52
```

Também podemos acessá-las pela sua posição:

```
linhas_num <- c(3, 7, 9)
dbenv[linhas_num,]
```

```
## dfs alt slo flo pH har pho nit amm oxy bdo
## 3 102 914 3.638 180 83 52 5 22 5 105 35
## 7 268 841 4.205 400 81 88 7 15 0 111 22
## 9 705 752 2.565 480 80 90 30 82 12 72 52
```

Finalmente, podemos combinar estes procedimentos para selecionar subconjunto de linhas e colunas.

```
dbenv[linhas,colunas]
```

```
## dfs flo oxy
## 3 102 180 105
## 7 268 400 111
## 9 705 480 72
```

## 2.4 Adicionando novas colunas a um data frame

Este conjunto de dados mostra medidas físicas e químicas obtidas em um riacho seguindo do trecho alto de cabeceira seguindo para os trechos baixos próximos à foz. O ponto mais alto (934 m de altitude) está a 3 km da cabeceira enquanto o ponto mais baixo está a 172 m de altitude e a 4530 km da cabeceira. Vamos criar uma nova variável categorizando os trechos do rio em “Alto”, “Medio” e “Baixo” assumindo a seguinte relação:

- 0 a 300 m: Baixo;
- 300 a 600 m: Médio;
- Acima de 600 m: Alto.

```
elv_cat <- cut(dbenv$alt, breaks = c(0, 300, 600, 1000),
              labels = c("Baixo", "Medio", "Alto"))
```

A inserção do novo objeto `elv_cat` no data frame pode ser feito simplesmente por:

```
dbenv$trecho <- elv_cat
```

Assim, inserimos assim, uma nova coluna denominada **trecho** contendo uma variável categórica com três níveis.

```
head(dbenv)
```

```
##   dfs alt   slo flo pH har pho nit amm oxy bdo trecho
## 1    3 934 6.176 84 79 45  1 20  0 122 27  Alto
## 2   22 932 3.434 100 80 40  2 20 10 103 19  Alto
## 3   102 914 3.638 180 83 52  5 22  5 105 35  Alto
## 4   185 854 3.497 253 80 72 10 21  0 110 13  Alto
## 5   215 849 3.178 264 81 84 38 52 20  80 62  Alto
## 6   324 846 3.497 286 79 60 20 15  0 102 53  Alto
```

Poderíamos ter realizado exatamente o mesmo utilizando a função **transform()**. Vamos utilizá-la a seguir como exemplo, criando uma nova variável categórica a partir do oxigênio dissolvido, considerando 3 níveis: pobre (0 a 5), médio (5 a 8), saturado (> 8).

```
dbenv <- transform(dbenv,
  saturacao = cut(dbenv$oxy, breaks = c(0, 40, 109, 124),
    labels = c("Pobre", "Medio", "Saturado")))
```

Veja agora o data frame

```
dbenv
```

```
##   dfs alt   slo flo pH har pho nit amm oxy bdo trecho saturacao
## 1    3 934 6.176 84 79 45  1 20  0 122 27  Alto  Saturado
## 2   22 932 3.434 100 80 40  2 20 10 103 19  Alto   Medio
## 3   102 914 3.638 180 83 52  5 22  5 105 35  Alto   Medio
## 4   185 854 3.497 253 80 72 10 21  0 110 13  Alto  Saturado
## 5   215 849 3.178 264 81 84 38 52 20  80 62  Alto   Medio
## 6   324 846 3.497 286 79 60 20 15  0 102 53  Alto   Medio
## 7   268 841 4.205 400 81 88  7 15  0 111 22  Alto  Saturado
## 8   491 792 3.258 130 81 94 20 41 12  70 81  Alto   Medio
## 9   705 752 2.565 480 80 90 30 82 12  72 52  Alto   Medio
## 10  990 617 4.605 1000 77 82  6 75  1 100 43  Alto   Medio
## 11 1234 483 3.738 1990 81 96 30 160 0 115 27  Medio  Saturado
## 12 1324 477 2.833 2000 79 86  4 50  0 122 30  Medio  Saturado
## 13 1436 450 3.091 2110 81 98  6 52  0 124 24  Medio  Saturado
## 14 1522 434 2.565 2120 83 98 27 123 0 123 38  Medio  Saturado
```

```
## 15 1645 415 1.792 2300 86 86 40 100 0 117 21 Medio Saturado
## 16 1859 375 3.045 1610 80 88 20 200 5 103 27 Medio Medio
## 17 1985 348 1.792 2430 80 92 20 250 20 102 46 Medio Medio
## 18 2110 332 2.197 2500 80 90 50 220 20 103 28 Medio Medio
## 19 2246 310 1.792 2590 81 84 60 220 15 106 33 Medio Medio
## 20 2477 286 2.197 2680 80 86 30 300 30 103 28 Baixo Medio
## 21 2812 262 2.398 2720 79 85 20 220 10 90 41 Baixo Medio
## 22 2940 254 2.708 2790 81 88 20 162 7 91 48 Baixo Medio
## 23 3043 246 2.565 2880 81 97 260 350 115 63 164 Baixo Medio
## 24 3147 241 1.386 2976 80 99 140 250 60 52 123 Baixo Medio
## 25 3278 231 1.792 3870 79 100 422 620 180 41 167 Baixo Medio
## 26 3579 214 1.792 3910 79 94 143 300 30 62 89 Baixo Medio
## 27 3732 206 2.565 3960 81 90 58 300 26 72 63 Baixo Medio
## 28 3947 195 1.386 4320 83 100 74 400 30 81 45 Baixo Medio
## 29 4220 183 1.946 6770 78 110 45 162 10 90 42 Baixo Medio
## 30 4530 172 1.099 6900 82 109 65 160 10 82 44 Baixo Medio
```

## 2.5 Aplicando uma função às linhas ou colunas de um data frame

### 2.5.1 Família de funções apply

Em muitas situações temos interesse aplicar um determinado cálculo a cada linha ou coluna de um data frame ou para grupos distintos.

*Observação:* o mesmo raciocínio serve-se aplica a objetos do tipo `matrix`.

Observe por exemplo se extraímos a média aritmética da coluna pH ( $\times 10$ ).

```
mean(dbenv$pH) # média aritmética
```

```
## [1] 80.5
```

#### Função `tapply`

Podemos estar interessados em extrair as médias para **cada categoria** de elevação. A função `tapply()` é útil nestas situações.

```
tapply(dbenv$pH, dbenv$trecho, mean)
```

```
##      Baixo      Medio      Alto
## 80.27273 81.22222 80.10000
```

## 2.5. APLICANDO UMA FUNÇÃO ÀS LINHAS OU COLUNAS DE UM DATA FRAME<sup>39</sup>

A função acima, pode ser *lida* do modo:

- Selecione a coluna `pH`;
- Agrupe os elementos em função dos níveis em `trecho` (Baixo, Medio, Alto);
- Calcule a média aritmética para cada sub-grupo.

Note que o resultado foi um vetor em que cada elemento corresponde à média de um sub-grupo. Funções que retornam mais de um valor resultam em um objeto no formato de `lista`. A função `range()` por exemplo, retorna dois valores (mínimo e máximo). Ao utilizá-la junto à função `tapply()` teremos como resultado uma lista composta por um vetor para cada subgrupo.

```
tapply(dbenv$pH, dbenv$trecho, range)
```

```
## $Baixo
## [1] 78 83
##
## $Medio
## [1] 79 86
##
## $Alto
## [1] 77 83
```

### Função `apply`

Podemos aplicar uma determinada função a todas as linhas ou colunas de um data frame (ou matriz).

```
apply(dbenv[,1:5], MARGIN = 2, mean)
```

```
##          dfs          alt          slo          flo          pH
## 1879.033333  481.500000    2.757733  2220.100000    80.500000
```

O argumento `MARGIN = 2` diz que desejamos aplicar a função às colunas da matriz. Com `MARGIN = 1` aplicamos a função às linhas da matriz.

### Função `lapply`

Se o objeto é do formato `lista`, o comando `lapply()` aplica uma função a cada elemento da lista. Considere a lista:

```
nossalista <- list(Ilha = c("Ilhabela", "Anchieta", "Cardoso"),
                  Areaskm2 = c(347.5, 8.3, 131),
                  Bioma = rep("Mata Atlantica",3),
                  Lat = c(23, 25, 23),
                  Long = c(45, 47, 45))
```

Veja os resultados dos comandos abaixo:

```
lapply(nossalista, sort)
```

```
## $Ilha
## [1] "Anchieta" "Cardoso"  "Ilhabela"
##
## $Areaskm2
## [1] 8.3 131.0 347.5
##
## $Bioma
## [1] "Mata Atlantica" "Mata Atlantica" "Mata Atlantica"
##
## $Lat
## [1] 23 23 25
##
## $Long
## [1] 45 45 47
```

```
lapply(nossalista, sort)
```

```
## $Ilha
## [1] "Anchieta" "Cardoso"  "Ilhabela"
##
## $Areaskm2
## [1] 8.3 131.0 347.5
##
## $Bioma
## [1] "Mata Atlantica" "Mata Atlantica" "Mata Atlantica"
##
## $Lat
## [1] 23 23 25
##
## $Long
## [1] 45 45 47
```

Obs.: Existem outras funções neste grupo, Veja o `help()` destas funções pois são extremamente úteis na manipulação de data frames e listas.



```
?tapply
?apply
?lapply
?mapply
?replicate
```

### 2.5.2 Função aggregate

A função `tapply()` aplica uma função a subgrupos de uma **única** coluna. A função `aggregate()` faz o mesmo, porém para **múltiplas** colunas agrupadas de acordo com uma ou mais categorias. O comando abaixo calcula os valores médios das variáveis para os trechos alto, médio e baixo combinados com níveis acima e abaixo de  $\text{pH} = 80$ .

```
media.trecho <- aggregate(dbenv[, 1:11],
                          by = list(TRECHO = dbenv$trecho,
                                     ALCALINO = dbenv$pH >= 80),
                          FUN = mean)
media.trecho
```

```
##   TRECHO ALCALINO      dfs      alt      slo      flo      pH      har
## 1  Baixo    FALSE 3472.250 222.5000 1.982000 4317.5000 78.75000 97.25000
## 2  Medio    FALSE 1324.000 477.0000 2.833000 2000.0000 79.00000 86.00000
## 3  Alto     FALSE  439.000 799.0000 4.759333  456.6667 78.33333 62.33333
## 4  Baixo     TRUE 3402.286 228.5714 1.986571 3786.5714 81.14286 95.57143
## 5  Medio     TRUE 1754.625 393.3750 2.501500 2206.2500 81.50000 91.50000
## 6  Alto      TRUE  284.000 847.7143 3.396429  258.1429 80.85714 74.28571
##           pho       nit      amm      oxy      bdo
## 1 157.50000 325.50000 57.5000000  70.75000 84.75000
## 2   4.00000  50.00000  0.0000000 122.00000 30.00000
## 3   9.00000  36.66667  0.3333333 108.00000 41.00000
## 4 92.42857 274.57143 39.7142857  77.71429 73.57143
## 5 31.62500 165.62500 7.5000000 111.62500 30.50000
## 6 16.00000  36.14286  8.4285714  93.00000 40.57143
```

## 2.6 Exportando um data frame

Finalmente, podemos exportar um resultado para arquivos texto. Vamos exportar o data frame `media.trecho` obtido acima para um arquivo `.csv`.

```
write.table(media.trecho, file = "Mediaportecho.csv",
            sep = ",", dec = '.', row.names = FALSE,
            col.names = TRUE)
```

- Veja o `help()` sobre a função `write.table` para mais informações.
- O arquivo será salvo em seu diretório de trabalho, aquele que você definiu no início desta seção com o comando `setwd()`.

## Chapter 3

# (Básico da) Visualização gráfica: pacote `graphics`

A visualização gráfica consiste em mostrar visualmente ou padrões de *distribuição* de uma variável ou padrões de *associação* entre duas ou mais variáveis. Os tipos de gráficos a serem utilizados depende basicamente do tipo de variável (ex. categórica ou numérica) e do número de variáveis envolvidas. Temos os gráficos **univariados** quando a visualização envolve uma única variável, gráficos **bivariados** quando a visualização envolve duas variáveis e gráficos **multivariados** que buscam expressar o padrão de associação entre mais de duas variáveis. Deixaremos os gráficos multivariados para outro momento e veremos aqui os tipos básicos para descrever padrões **uni** e **bivariados**.

As funções gráficas apresentadas neste tópico estão todas disponíveis no pacote **graphics**, que se inclui na lista dos pacotes **previamente** instalados no R. Não é necessário portanto preocupar-se com nenhuma instalação adicional. Estas funções possibilitam elevado controle sobre cada um dos elementos gráficos (fontes, tamanhos, cores), porém a custo de maior complexidade se temos a intenção de gerar figuras complexas. Vale ressaltar ainda que mesmo utilizando muitas nomenclaturas compatíveis para o controle de eixos, títulos, tamanho de fonte, estas funções nem sempre usam argumentos coesos entre os tipos gráficos, o que pode tornar a curva de aprendizado mais demorada. Por outro lado, estas funções fornecem **conhecimentos básicos** sobre a estrutura gráfica no R, permitindo resolver de forma rápida e simples muitas situações que encontramos no dia-a-dia da análise exploratória. Na próxima seção iremos tratar de outro pacote gráfico (`ggplot2`) que possui elevada capacidade para gerar estruturas complexas de imagem, utilizando uma estrutura mais coesa de manipulação gráfica. Também veremos novamente vários tipos gráficos no capítulo 7 quando formos falar de Estatística Descritiva.

### 3.1 Doubs river dataset

Para demonstrar algumas ferramentas gráficas, vamos utilizar um conjunto de dados já disponível na base de dados do R chamados *Doubs River data*. Na realidade, já analisamos parte deste conjunto de dados no capítulo 2, quando importamos o arquivo `doubs_environment.csv`. Deste ponto em diante, vamos utilizar o conjunto dados completo que está disponível no pacote `ade4` (Dray et al., 2015).

Este conjunto de dados resulta da tese de doutorado de Verneaux (Verneaux, 1973). O texto a seguir foi traduzido do site do excelente site de **David Zelený** sobre análise de dados multivariados em Ecologia de Comunidades ([www.davidzeleny.net](http://www.davidzeleny.net)).

Verneaux (Verneaux, 1973) propôs o uso de espécies de peixes para caracterizar zonas ecológicas ao longo de rios e riachos europeus. O autor mostrou que as comunidades de peixes eram bons indicadores biológicos desses corpos d'água. Partindo da foz até a cabeceira, Verneaux propôs uma tipologia em quatro zonas, nomeadas a partir da predominância de uma dada espécie de peixe: **trout zone**, **grayling zone**, **barbel zone** e **breame zone**. As condições ecológicas correspondentes nestas zonas variam desde águas relativamente bem oxigenadas e oligotróficas até águas eutróficas e desprovidas de oxigênio.

Os dados foram coletados em 30 localidades ao longo do rio Doubs, próximo à fronteira da França e Suíça. O conjunto de dados que você irá importar consiste em uma lista de 4 elementos:

1. `$env`: um data frame com 30 linhas por 11 colunas, em que as linhas representam os locais de amostragem da cabeceira a foz do riacho e as colunas representam variáveis ambientais relacionadas à hidrologia, geomorfologia e química do rio.
2. `$fish`: um data frame com 30 linhas por 27 colunas, em que as linhas representam os mesmo locais de amostragem e a cada coluna as abundâncias das 27 espécies de peixes capturadas.
3. `$xy`: um data frame com 30 linhas por 2 colunas. As linhas novamente são os locais de amostragem e as colunas representam as coordenadas geográficas de cada ponto de amostragem.
4. `$species`: um data frame com 27 linhas por 4 colunas. As linhas representam cada uma das 27 espécies capturadas e as colunas representam seus nomes científicos, nomes populares em francês, em inglês e um código abreviado.

#### 3.1.1 Instalando o pacote `ade4` e carregando os dados

Caso ainda não tenha feito, instale o pacote `ade4` através do comando:

```
install.packages("ade4")
```

Feito isto, carregue o pacote:

```
library(ade4)
```

O conjunto de dados `doubs`

```
data(doubs)
```

Veja o formato em lista destes dados:

```
class(doubs)
str(doubs)
```

E leia a descrição do conjunto de dados para entender melhor nossas discussões a frente.

```
?doubs
```

Diversos autores em ecologia de comunidades tem utilizado os *Doubs dataset* para exemplificar métodos e modelos de análise de dados, sobretudo multivariados. Inicialmente, vamos extrair os dados ambientais para um novo `data.frame` que iremos utilizar nas análises gráficas:

```
ambiente <- doubs$env
```

Baseados em Borcard et al. (Borcard et al., 2018), vamos adicionar a este data frame uma nova variável categórica denominada `secao` com quatro níveis.

```
ambiente$secao <- c(rep("Seção 1", 16), rep("Seção 4", 14))
ambiente$secao[c(5,9,17)] <- "Seção 2"
ambiente$secao[23:25] <- "Seção 3"
ambiente$secao <- factor(ambiente$secao)
```

Outra variável categórica, indicando três níveis de concentração de oxigênio em cada ponto.

```
ambiente$trofia <- cut(ambiente$oxy, breaks = c(0, 80, 109, 124),
                      labels = c("Pobre", "Médio", "Saturado"))
head(ambiente, 10)
```

```
##      dfs alt    slo  flo pH har pho nit amm oxy bdo  secao  trofia
## 1      3 934 6.176   84 79 45   1 20   0 122 27 Seção 1 Saturado
## 2     22 932 3.434  100 80 40   2 20  10 103 19 Seção 1 Médio
## 3    102 914 3.638  180 83 52   5 22   5 105 35 Seção 1 Médio
## 4    185 854 3.497  253 80 72  10 21   0 110 13 Seção 1 Saturado
## 5    215 849 3.178  264 81 84  38 52  20 80 62 Seção 2 Pobre
## 6    324 846 3.497  286 79 60  20 15   0 102 53 Seção 1 Médio
## 7    268 841 4.205  400 81 88   7 15   0 111 22 Seção 1 Saturado
## 8    491 792 3.258  130 81 94  20 41  12 70 81 Seção 1 Pobre
## 9    705 752 2.565  480 80 90  30 82  12 72 52 Seção 2 Pobre
## 10  990 617 4.605 1000 77 82   6 75   1 100 43 Seção 1 Médio
```

## 3.2 Descrevendo os padrões de uma variável

### 3.2.1 Gráfico de barras

Um gráfico de barras é utilizado para verificar a contagem de cada nível de uma variável. Portanto, necessariamente deve ser aplicado a uma *variável categórica*. Vamos fazer um gráfico de barras para a variável `trofia`

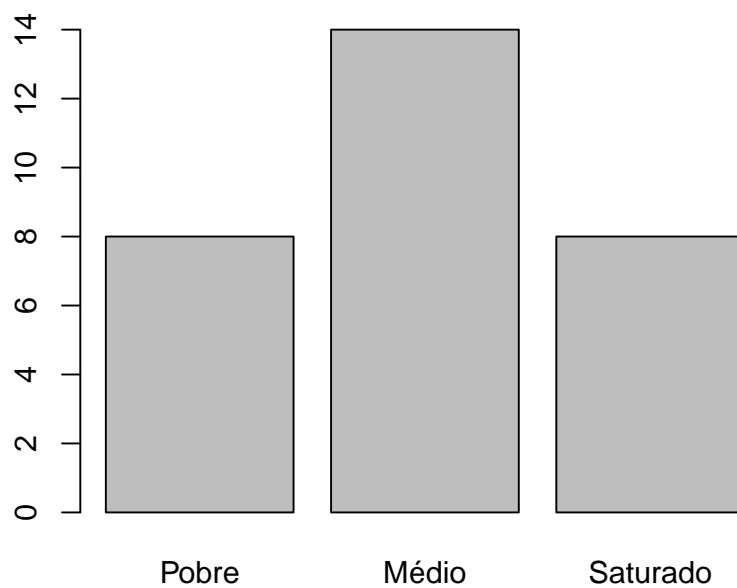
Montamos uma tabela de frequencia:

```
tab1 <- table(ambiente$trofia)
tab1
```

```
##
##      Pobre      Médio Saturado
##          8         14          8
```

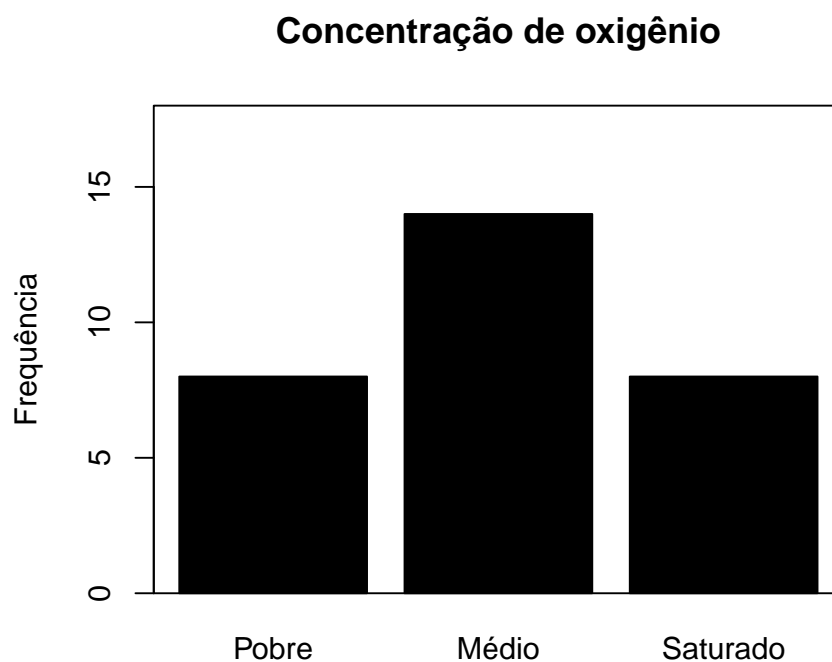
E em seguida mostramos esta tabela em um gráfico de barras:

```
barplot(tab1)
```



Melhorando a formatação gráfica:

```
barplot(tab1,  
        main = "Concentração de oxigênio",  
        ylab = "Frequência",  
        ylim = c(0, 18), col = "black")  
box()
```



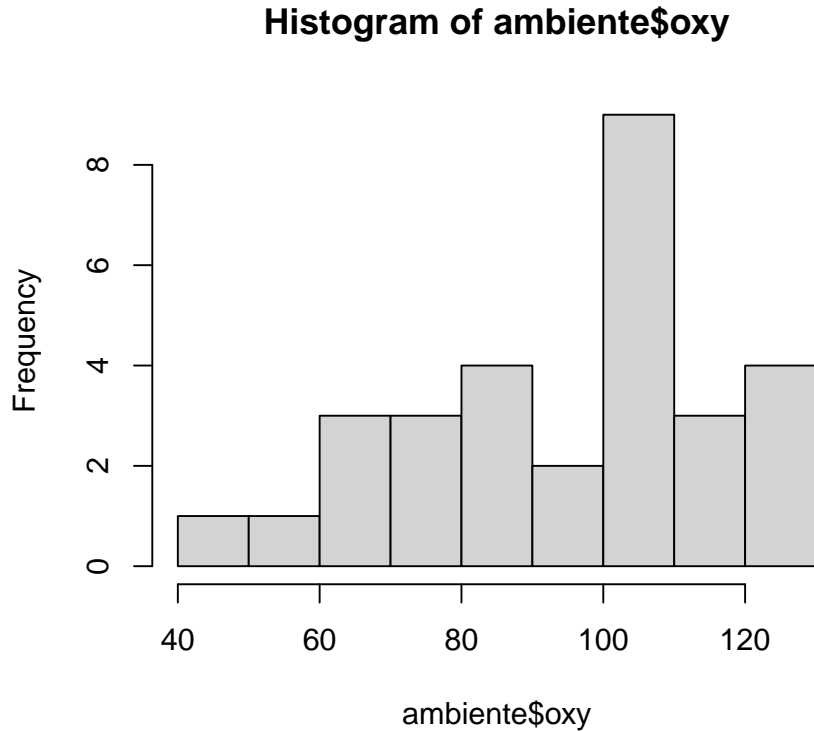
### 3.2.2 Histograma

Um histograma é a forma mais direta de avaliarmos o padrão de distribuição de uma variável quantitativa. Um histograma é construído a partir da divisão de uma variável em *intervalos de classe* e contando o número de classes dentro deste intervalo.

Veja o histograma abaixo para a oxy que expressa a concentração de oxigênio em  $\text{mg/l} \times 10$ .

```
hist(ambiente$oxy)
```





A figura mostra por exemplo que existe 1 seção com concentração entre 40 e 50  $\text{mg/l} \times 10$ , e 2 seções com concentração entre 90 e 100  $\text{mg/l} \times 10$ . Verifique quais são estes veículos com o comando abaixo:

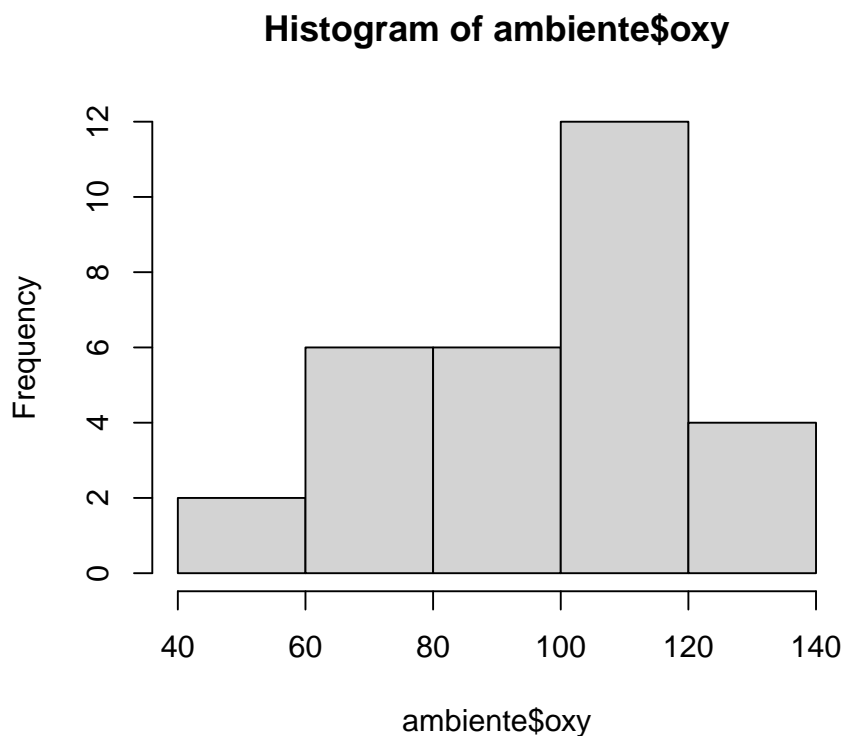
```
ambiente[order(ambiente$oxy),]
```

```
##      dfs alt  slo flo pH har pho nit amm oxy bdo  secoa  trofia
## 25 3278 231 1.792 3870 79 100 422 620 180 41 167 Seção 3  Pobre
## 24 3147 241 1.386 2976 80 99 140 250 60 52 123 Seção 3  Pobre
## 26 3579 214 1.792 3910 79 94 143 300 30 62 89 Seção 4  Pobre
## 23 3043 246 2.565 2880 81 97 260 350 115 63 164 Seção 3  Pobre
## 8 491 792 3.258 130 81 94 20 41 12 70 81 Seção 1  Pobre
## 9 705 752 2.565 480 80 90 30 82 12 72 52 Seção 2  Pobre
## 27 3732 206 2.565 3960 81 90 58 300 26 72 63 Seção 4  Pobre
## 5 215 849 3.178 264 81 84 38 52 20 80 62 Seção 2  Pobre
## 28 3947 195 1.386 4320 83 100 74 400 30 81 45 Seção 4  Médio
## 30 4530 172 1.099 6900 82 109 65 160 10 82 44 Seção 4  Médio
## 21 2812 262 2.398 2720 79 85 20 220 10 90 41 Seção 4  Médio
```

```
## 29 4220 183 1.946 6770 78 110 45 162 10 90 42 Seção 4 Médio
## 22 2940 254 2.708 2790 81 88 20 162 7 91 48 Seção 4 Médio
## 10 990 617 4.605 1000 77 82 6 75 1 100 43 Seção 1 Médio
## 6 324 846 3.497 286 79 60 20 15 0 102 53 Seção 1 Médio
## 17 1985 348 1.792 2430 80 92 20 250 20 102 46 Seção 2 Médio
## 2 22 932 3.434 100 80 40 2 20 10 103 19 Seção 1 Médio
## 16 1859 375 3.045 1610 80 88 20 200 5 103 27 Seção 1 Médio
## 18 2110 332 2.197 2500 80 90 50 220 20 103 28 Seção 4 Médio
## 20 2477 286 2.197 2680 80 86 30 300 30 103 28 Seção 4 Médio
## 3 102 914 3.638 180 83 52 5 22 5 105 35 Seção 1 Médio
## 19 2246 310 1.792 2590 81 84 60 220 15 106 33 Seção 4 Médio
## 4 185 854 3.497 253 80 72 10 21 0 110 13 Seção 1 Saturado
## 7 268 841 4.205 400 81 88 7 15 0 111 22 Seção 1 Saturado
## 11 1234 483 3.738 1990 81 96 30 160 0 115 27 Seção 1 Saturado
## 15 1645 415 1.792 2300 86 86 40 100 0 117 21 Seção 1 Saturado
## 1 3 934 6.176 84 79 45 1 20 0 122 27 Seção 1 Saturado
## 12 1324 477 2.833 2000 79 86 4 50 0 122 30 Seção 1 Saturado
## 14 1522 434 2.565 2120 83 98 27 123 0 123 38 Seção 1 Saturado
## 13 1436 450 3.091 2110 81 98 6 52 0 124 24 Seção 1 Saturado
```

Em um histograma, a escolha do intervalo de classes determina o formato exato do gráfico. No exemplo acima, a escolha foi feita automaticamente. No entanto, podemos definir explicitamente o intervalo desejado utilizando o argumento `breaks` conforme abaixo:

```
classes <- seq(40, 140, by = 20)
hist(ambiente$oxy, breaks = classes)
```

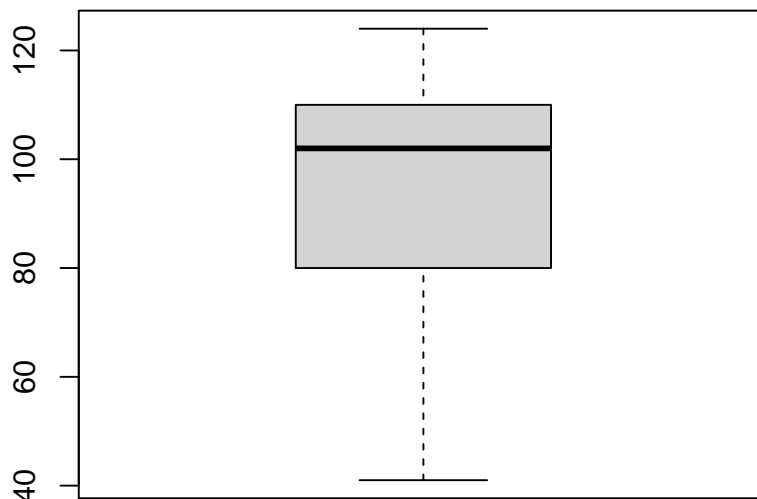


Aqui fizemos a divisão em intervalos de tamanho 20, iniciando em 40 e terminando em 140. A escolha do tamanho das classes é de certa forma arbitrária e definida para que o figura evidencie da melhor forma possível o padrão de distribuição dos dados.

### 3.2.3 Boxplot

Boxplots oferecem um *resumo gráfico* da distribuição de uma variável quantitativa. Abaixo veja um boxplot da variável oxy.

```
boxplot(ambiente$oxy)
```



No boxplot, a linha do meio representa a **mediana** dos dados, os limites das caixas representam o 1º e 3º percentis e as linhas os pontos mínimo e máximo. Podemos ver quais são estes valores com o comando:

```
quantile(ambiente$oxy, probs = c(0, 0.25, 0.5, 0.75, 1))
```

```
##      0%    25%    50%    75%   100%
##  41.00  80.25 102.00 109.00 124.00
```

### 3.3 Visualizando associações entre duas variáveis

#### 3.3.1 Gráfico de barras

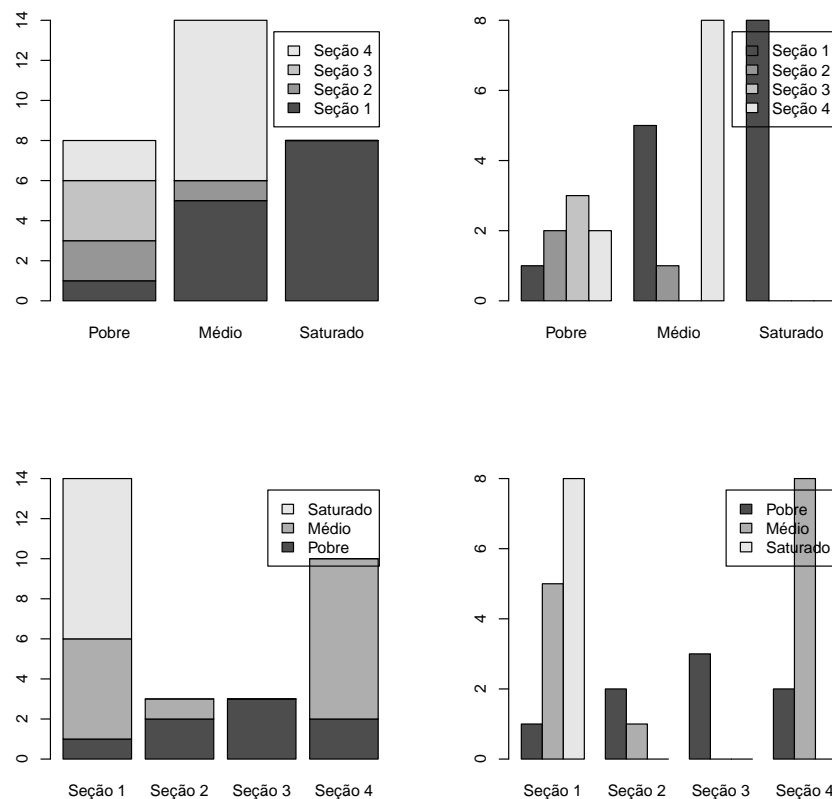
Passando aos gráficos bi-variados, vamos criar um gráfico de barras combinando as variáveis categóricas **secao** e **trofia**. Como fizemos anteriormente, montamos uma tabela de frequência, porpém neste caso combinando as duas variáveis.

```
tab2 <- table(ambiente[,c("secao", "trofia")])
tab2
```

```
##          trofia
## secao      Pobre Médio Saturado
## Seção 1      1      5         8
## Seção 2      2      1         0
## Seção 3      3      0         0
## Seção 4      2      8         0
```

Neste caso, podemos fazer gráficos de barras de quatro formas distintas:

```
layout(mat = matrix(1:4, nrow = 2, ncol = 2, byrow = TRUE))
barplot(tab2, legend = TRUE)
barplot(tab2, legend = TRUE, beside = TRUE)
barplot(t(tab2), legend = TRUE)
barplot(t(tab2), legend = TRUE, beside = TRUE)
```



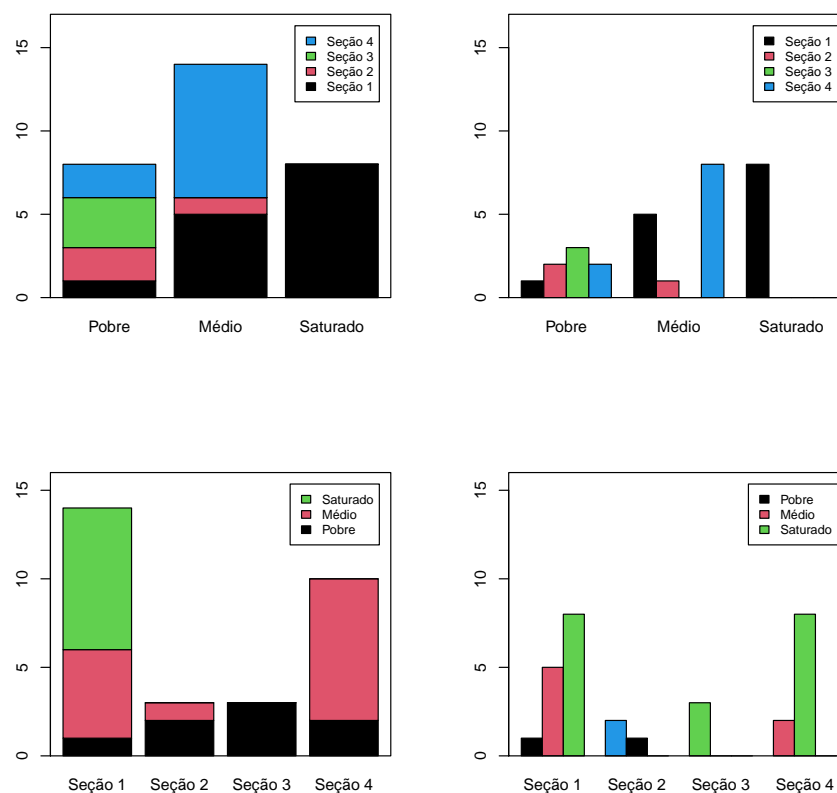
O comando acima necessita de algumas explicações.

1. A função `layout(mat = matrix(1:4, nrow = 2, ncol = 2, byrow = TRUE))` organiza o espaço gráfico em um formato matricial com 2 linhas por 2 colunas, permitindo a inserção de 4 figuras. O argumento `byrow = TRUE` define que as figuras serão adicionadas linha-a-linha;
2. A expressão `t(tab2)` tem como resultado *transpor* a tabela, o que consequentemente altera a referência da figura. No primeiro caso, a referência é a concentração de oxigênio e no segundo caso, as seções;
3. O argumento `beside = TRUE` faz com que todas as barras apareçam lado-a-lado. Caso contrário, cada barra representa uma coluna da matriz `tab2` ou da sua transposta `t(tab2)`;
4. Em todos os gráficos foi adicionada uma legenda para permitir a interpretação dos gráficos;

Aqui vale melhorarmos a formatação:

```
cores <- 1:4
limy1 <- c(0, 17)
limy2 <- c(0, 16)
legenda <- list(cex = 0.8)

layout(mat = matrix(1:4, nrow = 2, ncol = 2, byrow = TRUE))
barplot(tab2, legend = TRUE, col = cores, ylim = limy1,
        args.legend = legenda)
box()
barplot(tab2, legend = TRUE, beside = TRUE, col = cores,
        ylim = limy1, args.legend = legenda)
box()
barplot(t(tab2), legend = TRUE, col = cores, ylim = limy2,
        args.legend = legenda)
box()
barplot(t(tab2), legend = TRUE, beside = TRUE, col = cores,
        ylim = limy2, args.legend = legenda)
box()
```

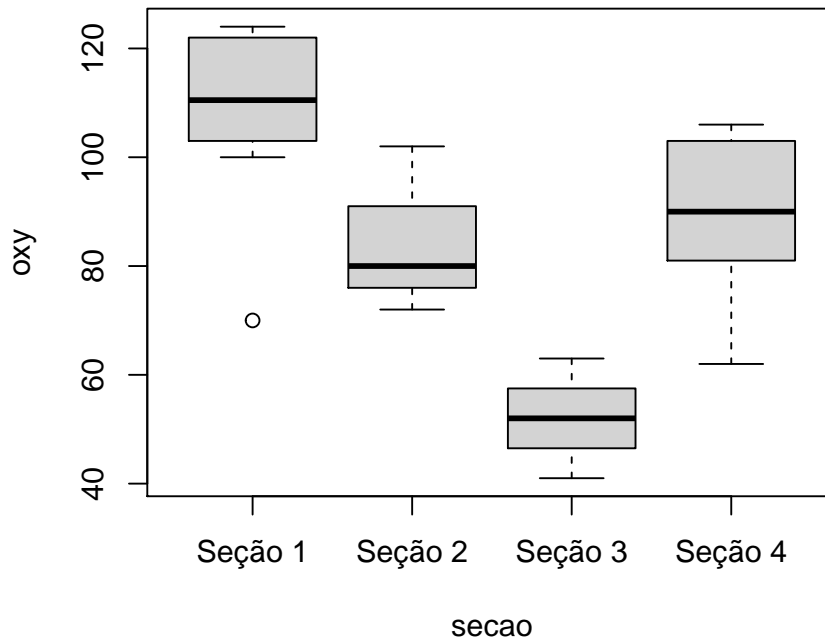


### 3.3.2 Boxplot

O boxplot é mais utilizado na situação a seguir em que queremos sumarizar uma variável quantitativa para diferentes níveis de uma variável categórica. Para isto, vamos associar a variável oxy à variável secas.

```
boxplot(oxy ~ secas, data = ambiente)
```





Vemos que *aparentemente* os pontos associados ao **Seção 1** têm maiores concentrações de oxigênio (mediana = NA) e que os pontos associados à **Seção 3** os menores valores (mediana = 52). É neste tipo de comparação que geralmente estamos interessados ao fazer um boxplot deste tipo.

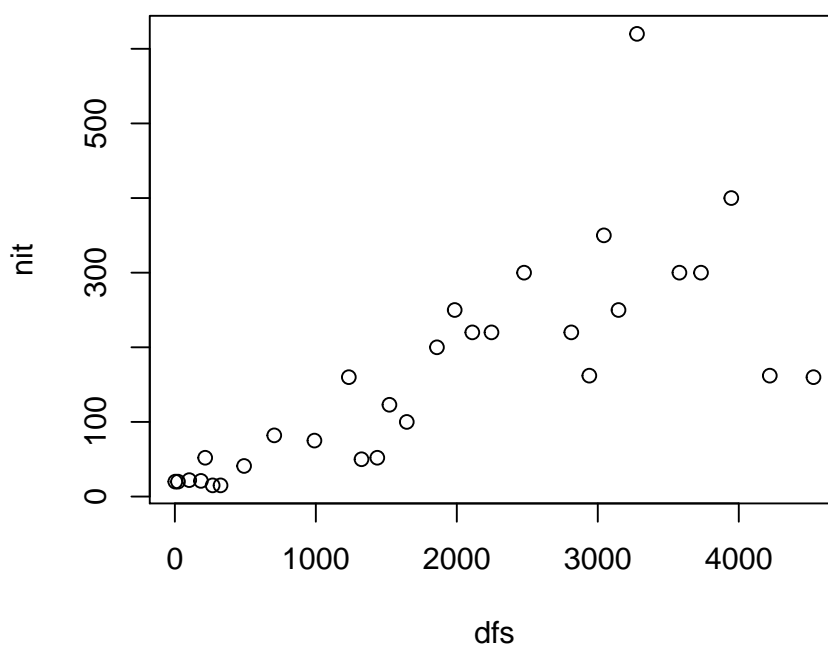
Aqui utilizamos uma notação diferente.

1. Ao invés de dizermos explicitamente qual variável está no eixo y e qual está no eixo x, utilizamos o símbolo  $\sim$  para expressar que y *depende de* x. Esta notação é amplamente utilizada em modelos estatísticos como Regressão e Análise de Variância e está associada aos conceitos de variável *dependente* (ou *reposta*, y) e de variável *independente* (ou *preditora*, x). Neste caso, então a concentração de oxigênio *depende* da seção do rio.
2. Ao invés de chamarmos a variável por `ambiente$oxy`, utilizamos somente o nome da coluna (`oxy`) e adicionamos o argumento `data = ambiente` para indicar em qual data frame a função irá buscar as variáveis. Deste ponto em diante iremos utilizar esta notação sempre que possível, para que você se familiarize com sua utilização na prática de ajuste de modelos estatísticos no R.

### 3.3.3 Gráfico de dispersão

Um gráfico de dispersão mostra a associação entre duas variáveis quantitativas. Vamos verificar a associação entre concentração de nitrato ( $\text{mg/l} \times 100$ ) e a distância da foz ( $\text{km} \times 10$ ). Neste caso, é fundamental definirmos quem serão as variáveis dependentes e independentes. Aqui, faz sentido pensar que a concentração de nitrato varia **em função** da distância da foz e não o contrário.

```
plot(nit ~ dfs, data = ambiente)
```

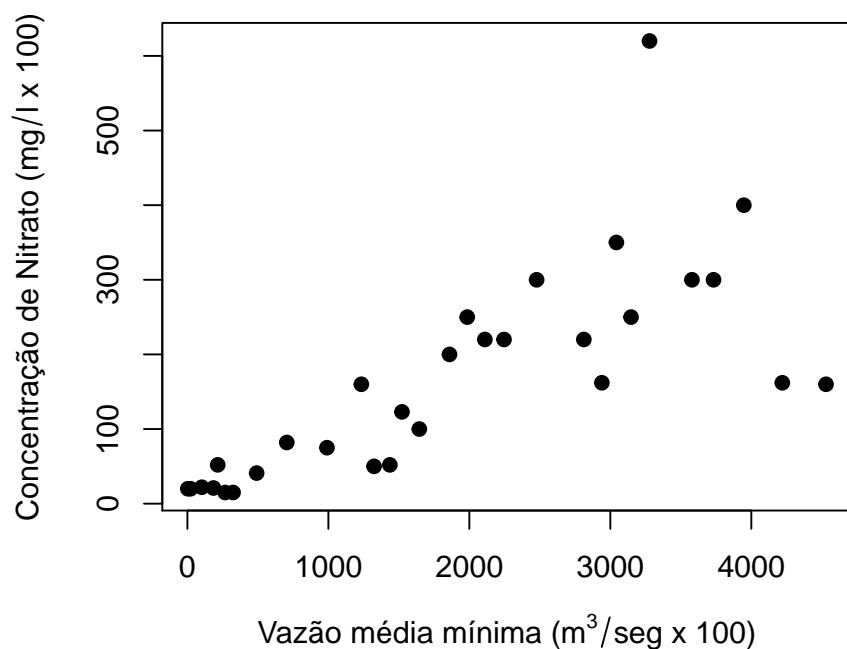


Os resultados aqui parecem expressar uma relação esperada em que a concentração de nutrientes aumenta à medida que nos aproximamos da foz de rio e riachos.

Novamente, vamos aproveitar para falar um pouco sobre formatação gráfica alterando os nomes dos eixos (argumentos `xlab` e `ylab`), tipo de ponto (argumento `pch`).

### 3.4. COMPREENDENDO O AMBIENTE POR MEIO DE SUAS VARIÁVEIS59

```
plot(nit ~ dfs, data = ambiente,  
     xlab = bquote("Vazão média mínima (m" ^3/"seg x 100)"),  
     ylab = bquote("Concentração de Nitrato (mg"/"l x 100)"),  
     pch = 19  
)
```



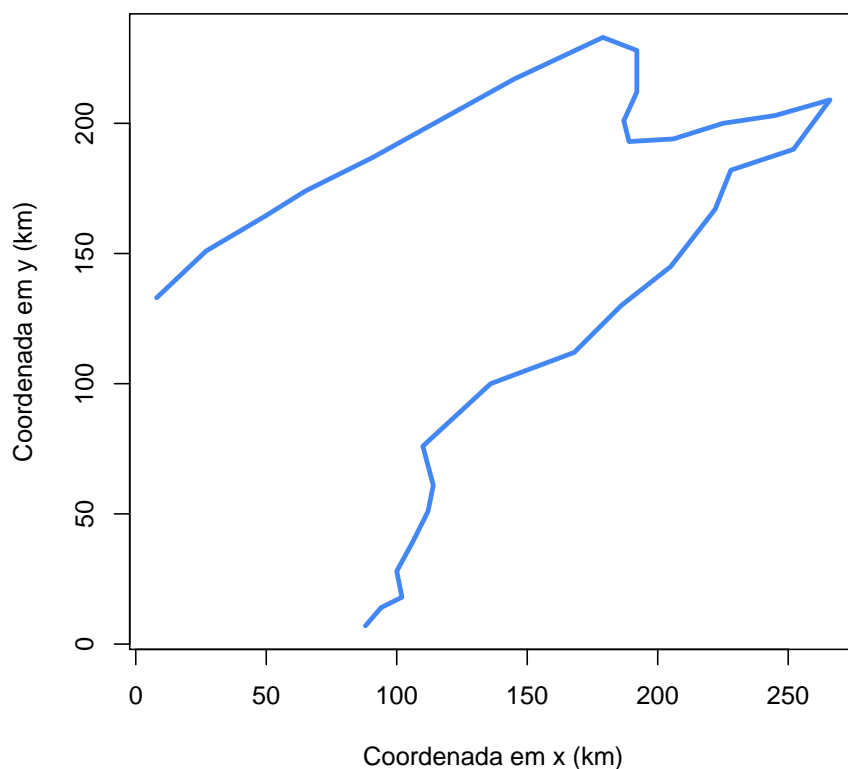
### 3.4 Compreendendo o ambiente por meio de suas variáveis

Em um estudo como o de Verneaux (Verneaux, 1973) o objetivo é entender os sistemas de riachos por meio das variáveis que escolhemos quantificar e como o modo como escolhemos visualizá-las. Os gráficos vistos acima não são certamente a única forma de incorporar variáveis em uma figura. As possibilidades de manipulação de cores, símbolos e textos no ambiente gráfico fornecem formas adicionais de incluir uma determinada informação. Nesta seção vamos explorar

um pouco melhor estas questões.

Dissemos que os pontos de amostragem foram obtidos ao longo do gradiente cabeceira-foz de um rio na França. Parte das informações que temos se referem às coordenadas geográficas destes pontos (no data frame `$yx`). Verifique também que a sequência dos pontos segue uma ordem crescente da distância da foz. Inicialmente, vamos plotar as coordenadas geográficas de todos os pontos utilizando um gráfico de linhas:

```
plot(x = doubs$xy$x, y = doubs$xy$y, type = "l",  
     xlab = "Coordenada em x (km)",  
     ylab = "Coordenada em y (km)",  
     col = "#4287f5", lwd = 3)
```



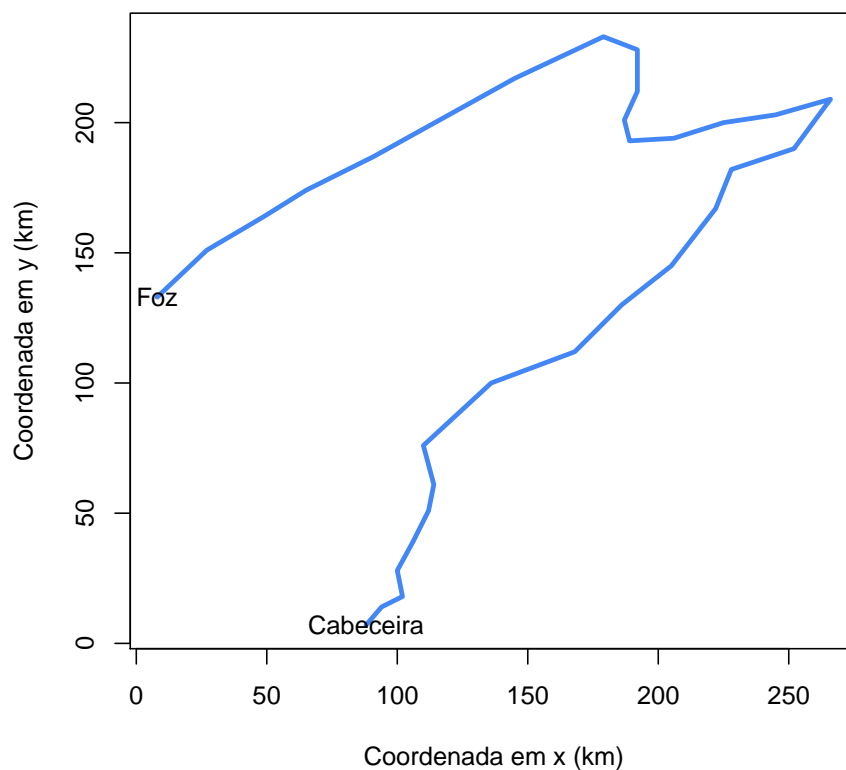
Compare, a figura com o desenho do rio Doubs.

### 3.4. COMPREENDENDO O AMBIENTE POR MEIO DE SUAS VARIÁVEIS61

- obs: utilizamos aqui a definição de cores em **HEXADECIMAL**. Você pode fazer o mesmo, escolhendo a cor desejada aqui: [hex color picker](#).

Vamos indicar os pontos de cabeceira e foz.

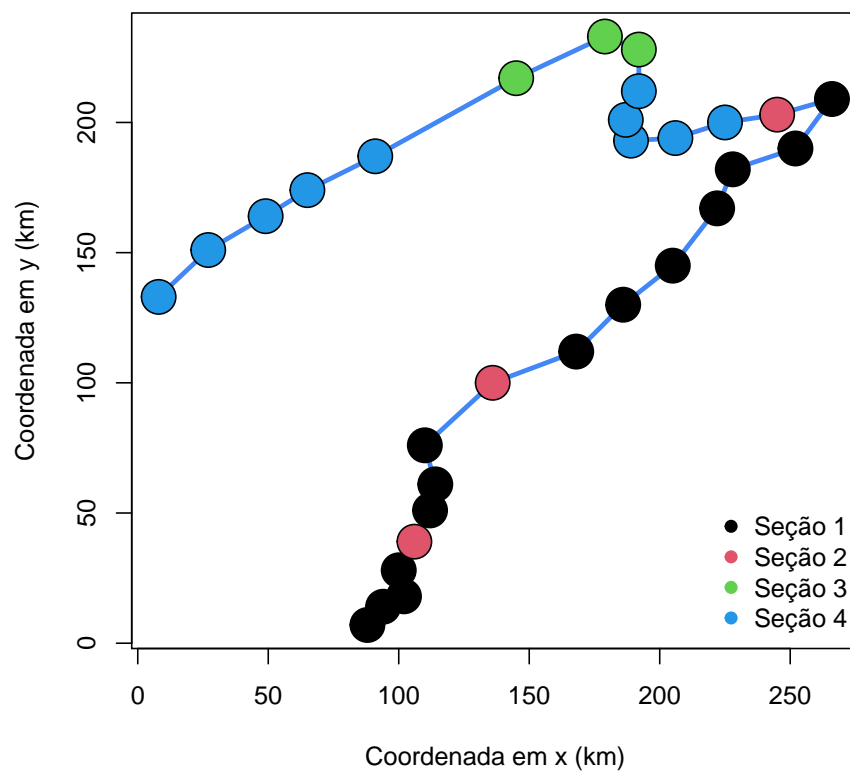
```
pontos_extremos <- doubs$xy[which(doubs$env$dfs == min(doubs$env$dfs) |  
                                doubs$env$dfs == max(doubs$env$dfs)),]  
plot(x = doubs$xy$x, y = doubs$xy$y, type = "l",  
     xlab = "Coordenada em x (km)",  
     ylab = "Coordenada em y (km)",  
     col = "#4287f5", lwd = 3)  
text(x = pontos_extremos$x,  
     y = pontos_extremos$y,  
     labels = c("Cabeceira", "Foz"))
```



Lembre-se que definimos acima 4 trechos. Vamos ver onde estas seções se localizam plotando-os com cores distintas.

```
secao_cor <- as.numeric(ambiente$secao)

plot(x = douds$xy$x, y = douds$xy$y, type = "l",
      xlab = "Coordenada em x (km)",
      ylab = "Coordenada em y (km)",
      col = "#4287f5", lwd = 3)
points(x = douds$xy$x, y = douds$xy$y, pch = 21,
       bg = secao_cor, cex = 3)
legend(x = "bottomright", col = 1:4,
       legend = levels(ambiente$secao), bty = "n", pch = 19)
```

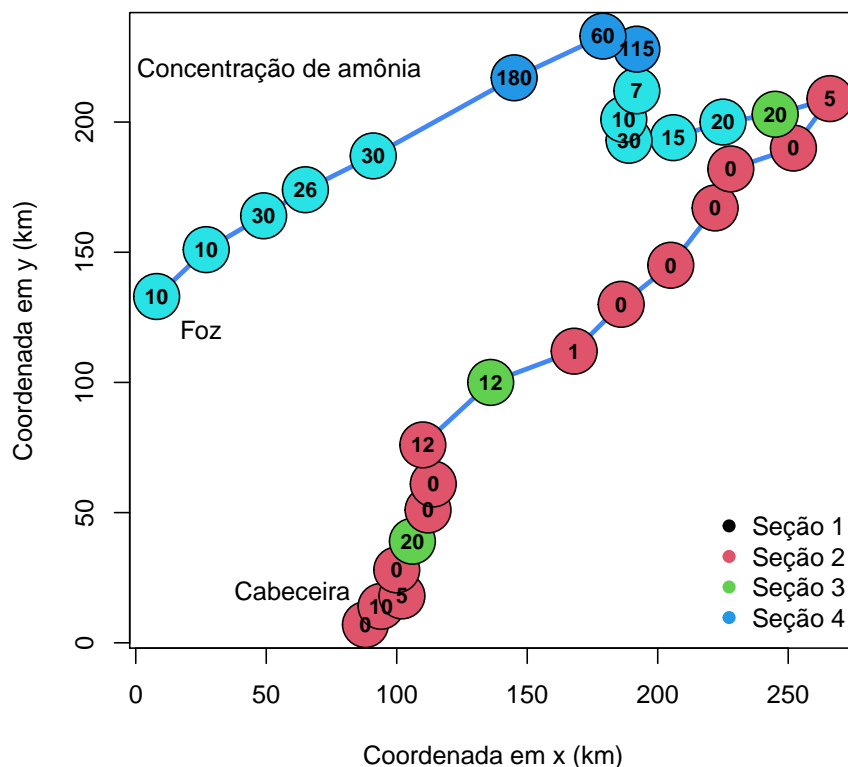


### 3.4. COMPREENDENDO O AMBIENTE POR MEIO DE SUAS VARIÁVEIS63

na metade superior do rio e as seções 3 e 4 na metade inferior. Para continuar explorando os dados, vamos inserir a informação sobre a concentração de amônia (amm).

```
secao_cor <- as.numeric(ambiente$secao) + 1

plot(x = doubs$xy$x, y = doubs$xy$y, type = "l",
     xlab = "Coordenada em x (km)",
     ylab = "Coordenada em y (km)",
     col = "#4287f5", lwd = 3)
points(x = doubs$xy$x, y = doubs$xy$y, pch = 21,
       bg = secao_cor, cex = 4)
legend(x = "bottomright", col = 1:4,
       legend = levels(ambiente$secao), bty = "n", pch = 19)
text(x = doubs$xy$x, y = doubs$xy$y, labels = doubs$env$amm,
     cex = 0.8, font = 2)
text(x = 55, y = 220, labels = "Concentração de amônia")
text(x = 25, y = 120, label = "Foz")
text(x = 60, y = 20, label = "Cabeceira")
```



Veja, que a concentração de amônia nos pontos em azul (Seção 4) é muito superior à dos pontos ao redor. Algo similar ocorre nos pontos em verde.

Provavelmente, a concentração de amônia não é a única variável pela formação destes grupos. Você pode explorar as demais variáveis químicas para verificar se outras também apresentam padrões similares.

O ponto importante e que merece ser ressaltado, é que o gráfico acima, não se enquadra em nenhuma das categorias anteriores (uni-variados, bi-variados, gráficos de barras, boxplots, etc.). No entanto, a figura nos informa sobre três variáveis: as coordenadas geográficas, a variável categórica `secao` e a concentração de amônia.

- Obs: utilizamos uma série de funções novas: `text`, `points`, `legend`. Para entender como elas funcionam, rode os comandos acima **linha por linha** e veja como cada função adiciona uma informação adicional à figura.



## 3.5 Mais um comentário sobre formatação gráfica no R

### 3.5.1 Outros argumentos

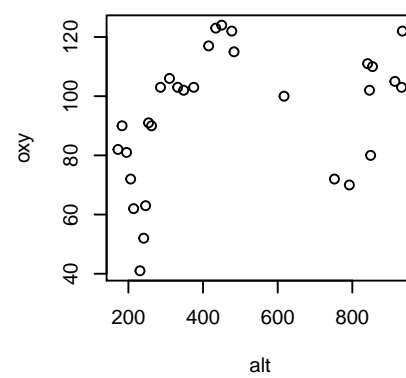
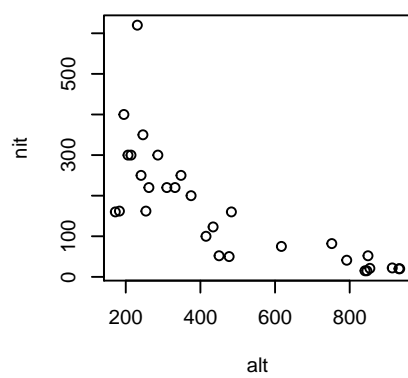
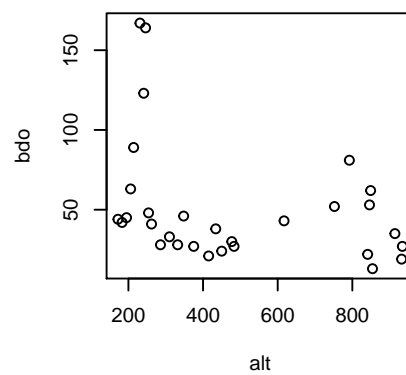
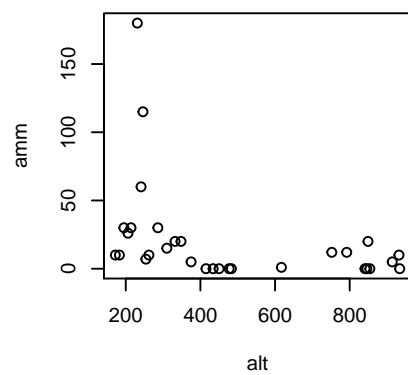
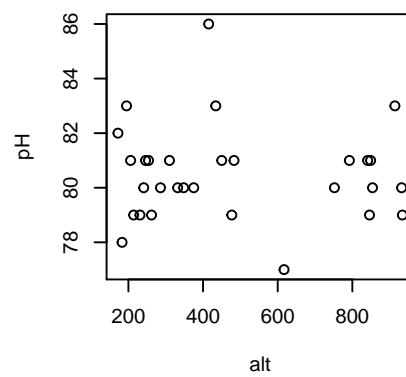
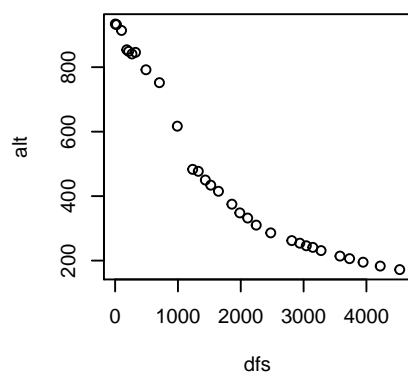
A capacidade de formatação gráfica no R é extensa. Qualquer tentativa de resumir todas elas seria incompleta. Portanto, apresento aqui somente alguns argumentos mais comuns. Rode abaixo cada uma das linhas e veja as figuras resultantes:

```
plot(nit ~ dfs, data = ambiente)
plot(nit ~ dfs, data = ambiente, pch = 2)
plot(nit ~ dfs, data = ambiente, pch = 19)
plot(nit ~ dfs, data = ambiente, pch = 19, type = "b")
plot(nit ~ dfs, data = ambiente, pch = 19, type = "b",
      xlab = "Nitrato", ylab = "Vazão")
plot(nit ~ dfs, data = ambiente, pch = 19, type = "b",
      xlab = "Nitrato", ylab = "Vazão", font.lab = 3)
plot(nit ~ dfs, data = ambiente, pch = 19, type = "l",
      lty = 2)
plot(nit ~ dfs, data = ambiente, pch = 19, type = "l",
      lty = 2, lwd = 3)
plot(nit ~ dfs, data = ambiente, pch = 19, type = "l",
      lty = 2, lwd = 3, col = 2)
```

### 3.5.2 Figuras compostas

Podemos formar figuras compostas, inserindo múltiplos gráficos. Uma das formas mais simples para isto é utilizando a função `layout`. Abaixo, vamos inserir 6 gráficos em um mesmo espaço.

```
layout(mat = matrix(1:6, nrow = 3, ncol = 2))
plot(alt ~ dfs, data = ambiente)
plot(amm ~ alt, data = ambiente)
plot(nit ~ alt, data = ambiente)
plot(pH ~ alt, data = ambiente)
plot(bdo ~ alt, data = ambiente)
plot(oxy ~ alt, data = ambiente)
```



### 3.5.3 Exportando figuras com as funções `png`, `tiff`, `jpeg` e `bmp`

Temos melhor controle sobre a qualidade gráfica no R exportando figuras em uma variedade de formatos e resoluções. Exemplificamos esta funcionalidade abaixo com a função `png`. No entanto, uma breve busca nos menus de ajuda mostrará que existem funções similares para outras extensões de imagem que possuem funcionamentos similares.

```
png(filename = "Exemplo_figura.png",  
     width = 15, height = 15, units = "cm",  
     pointsize = 10, bg = "white", res = 800)  
  
plot(alt ~ dfs, data = ambiente, pch = 19, type = "b",  
     xlab = "Vazão", ylab = "Elevação")  
  
dev.off()
```

Lembre-se que a figura foi salva do diretório atual de sua seção de trabalho. Você pode conferir este diretório com:

```
getwd()
```

Experimente alterar os argumentos `width`, `height`, `pointsize`, `units` (com `"px"`, `"in"`, `"cm"` ou `"mm"`) e `res`.

As capacidades gráficas no R incluem ainda muitos outros argumentos. Alguns deles são: cores (`col`), tipos da fonte (`font`), tamanhos de símbolos (`cex`), dos labels (`cex.lab`), dos rótulos dos eixos (`cex.axis`), título (`main`), etc. Pode-se ainda inserir legendas (função `legend`) e textos (função `text`). Veja o **help** de cada uma destas funções e a lista de argumentos possíveis para o ambiente gráfico do R em `?par`. Veja também uma demonstração com `demo(graphics)`, `demo(image)`, `demo(persp)` e `demo(plotmath)`.

Existem diversos outros pacotes gráficos além do `graphics`:

- `ggplot2`
- `ggvis`
- `Lattice`
- `highcharter`
- `Leaflet`
- `RColorBrewer`
- `Plotly`
- `sunburstR`
- `RGL`

- `dygraphs`

Veremos somente uma introdução ao pacote `ggplot2` no capítulo 4. Você pode buscar informações nos manuais oficiais destes pacotes, mas sem dúvida a fonte mais extensa de informação são todos os usuários que dispõem de seus exemplos na rede. Assim, para aprender sobre estes pacotes ou outras técnicas gráficas no R não existe em tentar um **Google** do tipo quero fazer meus gráficos no R.

**Boa sorte!!**

## Part II

# Uma coleção integrada de pacotes para ciência de dados



## Chapter 4

# Os pacotes em tidyverse

A Ciência de Dados, como chamada atualmente, passa por uma coleção de ações relacionadas à importação de dados, formatação, padronização de informações, visualização, modelagem e comunicação dos resultados. Ainda que estes processos sejam conhecidos e aplicados por profissionais de diferentes áreas na academia ou no mercado, o uso cada vez mais recorrente do termo **Ciência de Dados** impulsionou a construção de ferramentas em diferentes linguagens de programação para integrar estes processos de forma coesa. No R, o pacote que trás esta filosofia em sua estrutura é o **tidyverse**. O **tidyverse** na realidade, agrega um *conjunto de pacotes* que funcionam de maneira integrada. Atualmente (versão 1.3.1 em 2021-05-18) estão incorporados no **tidyverse** os pacotes:

- **readr**: importação de dados
- **dplyr**: manipulação de dados
- **tidyr**: organização de dados
- **purrr**: programação funcional
- **tibble**: visualização de data frames
- **stringr**: manipulação de texto
- **forcats**: manipulação de fatores
- **ggplot2**: visualização gráfica

Além destes existem ainda outros que se integram bem à filosofia do **tidyverse** como o **lubridate** (manipulação de datas), o **readxl** (leitura de arquivos **.xls** e **.xlsx**), **rvest** (manipulação na web), o **rmarkdown** (formatação de relatórios dinâmicos, apresentações e outros documentos). Não iremos ver exemplos de todos os pacotes, somente algumas das funções mais úteis. Você pode buscar mais informações sobre cada um deles no site do tidyverse. Para uma visão geral de cada pacote, você pode verificar as Cheatsheets que oferecem um resumo sobre as funções de diversos pacotes.

### 4.0.1 Instalando os pacotes tidyverse

Um **pacote** no R, trás um conjunto de funções organizadas ao redor de um problema comum. Você pode instalar um pacote no R de três formas:

- Utilizando o comando `install.packages("nome_do_pacote")` que irá buscá-lo no CRAN, Comprehensive R Archive Network;
- a partir do Github: “`devtools::install_github("nome_do_repositorio/nome_do_pacote")`”
- A partir de um arquivo compactado `.zip` ou `tar.gz` com o comando `install.packages("C:/seu_diretorio/nome_do_pacote", repos = NULL)`

Lembre-se ao utilizar a segunda opção que o `devtools` também é um pacote e portanto, deve ser instalado no R. Alguns pacotes não disponíveis no site oficial do CRAN, Comprehensive R Archive Network podem ser instalados somente por esta opção.

Cada um dos pacotes incorporados no **tidyverse** pode ser instalado individualmente. Por exemplo:

```
install.packages("dplyr")
install.packages("ggplot2")
```

Entretando, ao instalar o **tidyverse**, todos são instalados de uma única vez:

```
install.packages("tidyverse")
```

## 4.1 Carregando os pacotes

Ao iniciar uma seção, você deve sempre carregar os pacotes que irá utilizar. No caso do **tidyverse**, você pode carregar cada pacote individualmente:

```
library(dplyr)
library(ggplot2)
```

Ou todos de uma única vez:

```
library(tidyverse)
```



## Chapter 5

# Importando/Exportando dados

### 5.1 Ajustando o diretório de trabalho

Antes de apresentarmos as funções de importação e exportação de dados devemos re-lembrar as funções que nos permitem verificar e alterar o *diretório de trabalho*. Um diretório de trabalho é que o local onde o R irá buscar e salvar arquivos em seu computador. Se não for definido, o R irá utilizar o diretório base. No R-Studio você pode verificar qual é seu diretório base no menu: **Tools** --> **Global Options** --> **R General**.

Se você estiver utilizando um **projeto** o R irá utilizar o diretório onde está o arquivo com extensão **.Rproj**. Uma das vantagens em utilizar *projetos* no R é a facilidade em manipular o diretório de trabalho e sub-diretórios dentro deles. Sub-diretórios dentro da pasta de um projeto podem auxiliar na organização das base de dados para importação, do material gerado como figuras, slides, arquivos **.pdf**, etc. No R-Studio você pode criar um projeto via menu: **File** --> **New Project**....

Você pode verificar seu diretório de trabalho com a função `getwd()`. Esta função não requer argumentos. A função `setwd()` é utilizada para alterar o diretório de trabalho e recebe como argumento o diretório de destino. O comando:

```
getwd("C:/seu_caminho/IntroR")
```

Irá alterar o caminho de busca para esta pasta.

## 5.2 Importando dados em arquivos de texto

O pacote responsável pela importação de dados no `tidyverse` é o `readr`. Este pacote permite importar arquivos de texto no formato `.csv` ou `.txt`.

Existem diversas funções no pacote `readr`. Veja seu manual [Aqui](#). A função `read_csv()` importa arquivos texto em que as colunas são separadas por *vírgulas*. A função `read_tsv()` importa arquivos texto em que as colunas são separadas por *tabulações*. Iremos ver a função `read_delim()` que é mais geral e permite que você especifique o tipo de separador (*delimitador*) de acordo com o padrão em seu arquivo. Os delimitadores mais comuns são *vírgulas*, *tabulações* ou *ponto-e-vírgula*.

Vamos voltar ao arquivo `doubs_environment.csv` visto no capítulo 2. Vamos importá-lo agora utilizando a função `read_delim()`.

```
library(readr)
```

```
dbenv = read_delim(file = "C:/seu_caminho/IntroR/dbenv.csv", delim = ",")
```

verifique o objeto importado.

```
dbenv
```

```
## # A tibble: 30 x 11
##   dfs alt slo flo pH har pho nit amm oxy bdo
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     3  934  6.18   84  79  45   1   20   0  122   27
## 2    22  932  3.43  100  80  40   2   20  10  103   19
## 3   102  914  3.64  180  83  52   5   22   5  105   35
## 4   185  854  3.50  253  80  72  10  21   0  110   13
## 5   215  849  3.18  264  81  84  38  52  20   80   62
## 6   324  846  3.50  286  79  60  20  15   0  102   53
## 7   268  841  4.20  400  81  88   7  15   0  111   22
## 8   491  792  3.26  130  81  94  20  41  12   70   81
## 9   705  752  2.56  480  80  90  30  82  12   72   52
## 10  990  617  4.61 1000  77  82   6  75   1  100   43
## # ... with 20 more rows
```

O objeto é do tipo `tibble` com 30 linhas por 11 colunas. Note que dependendo do tamanho de sua janela, nem todas as linhas e colunas irão aparecer, porém será indicado as linhas e colunas ocultas.

## Chapter 6

# Operador Pipe

O **operador pipe** (`%>%`) é um facilitador para a manipulação e formatação de dados na filosofia *tidyverse*. O operador pipe não precisa ser utilizado somente com as funções dos pacotes do tidyverse, pode ser utilizado com qualquer função em R. Sua função é facilitar a leitura de códigos, sobretudo códigos longos de formatação de data frames.

Para utilizá-lo instale e carregue o pacote `magrittr`:

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

Este operador funciona fazendo com que um objeto criado ao lado esquerdo de uma expressão, seja utilizado como *entrada* de uma função escrita ao lado direito da expressão conforme:

$y \%>\% f(y)$

Veja um exemplo:

```
variavel <- 49  
variavel %>% sqrt()
```

```
## [1] 7
```

O objeto `variavel` à esquerda de `%>%` é portanto utilizado como entrada para a função `sqrt()`, à direita de `%>%`. A expressão é similar a:

```
variavel <- 49  
sqrt(variavel)
```

```
## [1] 7
```

Neste exemplo, o uso do operador pipe não trás qualquer vantagem. Por outro lado veja a sequência abaixo escrita da forma convencional:

```
objeto <- c(2,6,3,8,10)
sum(diff(round(sqrt(objeto),digits = 2)))
```

```
## [1] 1.75
```

E utilizando o operador pipe:

```
objeto <- c(2,6,3,8,10)

objeto %>% sqrt() %>% round(digits = 2) %>% diff() %>% sum()
```

```
## [1] 1.75
```

Na forma convencional, a leitura se dá de **dentro para fora** da expressão. À medida que o número de operações aumenta, a leitura e consequentemente a correção desta sequência se torna complexa.

Utilizando o operador pipe a leitura se dá:

1. Pegue o objeto *variável*;
2. Tire a raiz quadrada;
3. Arredonde a raiz quadrada para duas casa decimais;
4. Calcule a diferença de um objeto com o anterior; e
5. Some as entradas.

Por hora, busque entender esta sequência. As vantagens do operador pipe se tornarão mais claras quando formos falar de formatação de data frames nos capítulos a frente.

## Chapter 7

# Manipulação e formatação de dados

Após importar uma base de dados para o R, os principais pacotes para manipulação e formatação de data frames são o `dplyr` e o `tidyr`. As funções destes pacotes oferecem uma etapa inicial na análise, modelagem e comunicação de dados. Lembramos aqui que uma base de dados é organizada em um formato de tabela em que as linhas são as *observações* e as colunas são as *variáveis* que descrevem estas observações. Estas variáveis podem ser quantitativas (contínuas ou discretas) ou qualitativas (ordenadas ou não).

### 7.1 Transformação de dados: os pacotes `dplyr` e `tidyr`

Aqui veremos funções principais para manipular as observações nas linhas (*Manipulate Cases*), manipular descritores nas colunas (*Manipulate Variables*) e combinar tabelas (*Combine Tables*). Na Cheatsheets do `dplyr` você verá também a variedade de funções relacionadas ao cálculo sobre vetores (*Vector Functions*), funções resumo (*Summary Functions*, e *Summarise Cases*) e manipulação de nomes das linhas.

#### 7.1.1 Ordenando as linhas: funções `arrange()` e `desc()`

Estas funções permitem que você ordene a base de dados seguindo os valores de alguma de suas colunas. Vamos utilizar como exemplo a base de dados `iris` comumente utilizadas em uma variedade de tutorias sobre ciência de dados. Carregue e verifique a base de dados composta por 150 linhas e 5 colunas.

```
data("iris")
head(iris, 10)
```

Vamos ordenar a tabela pela coluna `Sepal.Length`.

```
iris %>%
  arrange(Sepal.Length)
```

| ##    | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|-------|--------------|-------------|--------------|-------------|---------|
| ## 1  | 4.3          | 3.0         | 1.1          | 0.1         | setosa  |
| ## 2  | 4.4          | 2.9         | 1.4          | 0.2         | setosa  |
| ## 3  | 4.4          | 3.0         | 1.3          | 0.2         | setosa  |
| ## 4  | 4.4          | 3.2         | 1.3          | 0.2         | setosa  |
| ## 5  | 4.5          | 2.3         | 1.3          | 0.3         | setosa  |
| ## 6  | 4.6          | 3.1         | 1.5          | 0.2         | setosa  |
| ## 7  | 4.6          | 3.4         | 1.4          | 0.3         | setosa  |
| ## 8  | 4.6          | 3.6         | 1.0          | 0.2         | setosa  |
| ## 9  | 4.6          | 3.2         | 1.4          | 0.2         | setosa  |
| ## 10 | 4.7          | 3.2         | 1.3          | 0.2         | setosa  |

As linhas aparecem agora de acordo com os valores (em ordem crescente) de `Sepal.Length`, iniciando em 4.3.

Em seguida faça o mesmo em ordem *decrecente*.

```
iris %>%
  arrange(desc(Sepal.Length))
```

Podemos combinar duas colunas, ordenando a tabela em função da coluna `Species` (em ordem alfabética decrescente) e em função de `Sepal.Length` (em ordem crescente).

```
iris %>%
  arrange(desc(Species), Sepal.Length)
```

Nos exemplos acima, somente visualizamos a tabela em diferentes ordens. Se quisermos criar um novo objeto com a tabela em alguma destas sequências fazemos:

```
iris_ordenado <- iris %>%
  arrange(Sepal.Length)

iris_ordenado
```

|      | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|------|--------------|-------------|--------------|-------------|---------|
| ## 1 | 4.3          | 3.0         | 1.1          | 0.1         | setosa  |
| ## 2 | 4.4          | 2.9         | 1.4          | 0.2         | setosa  |
| ## 3 | 4.4          | 3.0         | 1.3          | 0.2         | setosa  |
| ## 4 | 4.4          | 3.2         | 1.3          | 0.2         | setosa  |
| ## 5 | 4.5          | 2.3         | 1.3          | 0.3         | setosa  |
| ## 6 | 4.6          | 3.1         | 1.5          | 0.2         | setosa  |

### 7.1.2 Filtrando linhas: função `filter()`

Esta função permite extrair somente as linhas de uma tabela que satisfaçam uma condição lógica. Vamos extrair as linhas referentes à espécie `virginica`.

```
iris %>%
  filter(Species == "virginica")
```

Ou às espécies *diferentes* de `virginica`.

```
iris %>%
  filter(Species != "virginica")
```

Agora, vamos filtrar as linhas em que o comprimento das pétalas seja menor que 1.3.

```
iris %>%
  filter(Petal.Length < 1.3)
```

E para o comprimento das pétalas seja menor que 1.3 **E** o comprimento das sépalas maior ou igual a 5.

```
iris %>%
  filter(Petal.Length < 1.3 & Sepal.Length >= 5)
```

### 7.1.3 Selecionando colunas: função `select()`

A função `select` permite extrair ou reorganizar um subconjunto de colunas de um data frame. Rode os exemplos a seguir:

```
iris %>%
  select(Petal.Length)
```

```
iris %>%
  select(Petal.Length, Species)
```

```
iris %>%
  select(Petal.Length:Species)
```

```
iris %>%
  select(Species:Petal.Length)
```

```
iris %>%
  select(!c(Petal.Length, Species))
```

```
iris %>%
  select(starts_with("Sepal"))
```

Finalmente, combine as funções `filter()` e `select()` para extrair um subconjunto do data frame:

```
iris %>%
  select(starts_with("Sepal")) %>%
  filter(Sepal.Length <= 4.5)
```

#### 7.1.4 Agrupando tabelas: funções do grupo join

Se você tem alguma experiência em linguagem SQL para banco de dados, irá compreender facilmente o uso do grupo de funções **join**. Veremos aqui as funções `left_join()`, `right_join()`, `inner_join()` e `anti_join()`.

Considere as duas tabelas abaixo que podem ser acessadas em `Regiao.csv` e `Habitat.csv`

```
## # A tibble: 10 x 4
##   Riacho Bacia      Município      Área
##   <chr>  <chr>      <chr>      <dbl>
## 1 R1     Boicucanga São Sebastião 30.3
## 2 R4     Boicucanga São Sebastião 30.3
## 3 R8     Boicucanga São Sebastião 30.3
## 4 R2     Cubatão    Cubatão      189
## 5 R5     Cubatão    Cubatão      189
## 6 R10    Cubatão    Cubatão      189
## 7 R13    Cubatão    Cubatão      189
## 8 R6     Quilombo   Santos        86
## 9 R9     Quilombo   Santos        86
## 10 R7    Quilombo   Santos        86
```



```
## # A tibble: 8 x 4
##   Riacho Altitude Largura Profundidade
##   <chr>      <dbl>   <dbl>      <dbl>
## 1 R1         74     7.8        20.2
## 2 R4         14    10.9        17.7
## 3 R8        245     8.3        19.5
## 4 R11        241     2.2        20.3
## 5 R2         29     1.6        11.8
## 6 R6         86    15.2        35.3
## 7 R9         77     4.1        18.9
## 8 R7         63    14.2        42.1
```

```
regiao <- read_csv("C:/seu_caminho/IntroR/Regiao.csv")
regiao
```

```
habitat <- read_csv("C:/seu_caminho/IntroR/Habitat.csv")
habitat
```

A tabela **regiao**, contém informações sobre alguns o bacia hidrográfica, á área da bacia e o município de alguns riachos da região litorânea do estado de São Paulo. A tabela **trecho**, contém informações sobre a largura e profundidades destes riachos. Alguns riachos são comuns às duas tabelas, enquanto outros ocorrem somente em uma ou em outra tabela. Sabendo que a coluna **Riacho** (R1, R2, R3,...R13) se referem ao mesmo ponto de amostragem, podemos utilizá-la para **combinar** as informações das duas tabelas.

#### 7.1.4.1 Função `left_join()`

Retorna todas as linhas da tabela à **esquerda** as as colunas das duas tabelas. Linhas sem correspondência na tabela da direita terão valores de **NA**. Se houver várias correspondências, todas as combinações serão retornadas.

```
regiao %>% left_join(y = habitat)
```

```
## # A tibble: 10 x 7
##   Riacho Bacia      Município      Área Altitude Largura Profundidade
##   <chr> <chr>      <chr>      <dbl>   <dbl>   <dbl>      <dbl>
## 1 R1    Boicucanga São Sebastião 30.3     74     7.8        20.2
## 2 R4    Boicucanga São Sebastião 30.3     14    10.9        17.7
## 3 R8    Boicucanga São Sebastião 30.3    245     8.3        19.5
## 4 R2    Cubatão    Cubatão    189     29     1.6        11.8
## 5 R5    Cubatão    Cubatão    189    NA     NA         NA
## 6 R10   Cubatão    Cubatão    189    NA     NA         NA
## 7 R13   Cubatão    Cubatão    189    NA     NA         NA
```

```
## 8 R6      Quilombo Santos      86      86      15.2      35.3
## 9 R9      Quilombo Santos      86      77       4.1      18.9
## 10 R7     Quilombo Santos      86      63      14.2      42.1
```

Veja que **todas** as linhas da tabela `regiao` estão presentes. Foram adicionadas a elas as informações de `Altitude`, `Largura` e `Profundidade`, somente para os riachos que também estavam presentes na tabela `habitat`.

#### 7.1.4.2 Função `right_join()`

Retorna todas as linhas da tabela à **direita** as as colunas das duas tabelas. Linhas sem correspondência na tabela da esquerda terão valores de `NA`. Se houver várias correspondências, todas as combinações serão retornadas.

```
regiao %>% right_join(y = habitat)
```

```
## # A tibble: 8 x 7
##   Riacho Bacia Município Área Altitude Largura Profundidade
##   <chr> <chr>    <chr>   <dbl>   <dbl>   <dbl>     <dbl>
## 1 R1    Boicucanga São Sebastião 30.3     74     7.8     20.2
## 2 R4    Boicucanga São Sebastião 30.3     14    10.9     17.7
## 3 R8    Boicucanga São Sebastião 30.3    245     8.3     19.5
## 4 R2    Cubatão    Cubatão    189     29     1.6     11.8
## 5 R6    Quilombo   Santos     86     86    15.2     35.3
## 6 R9    Quilombo   Santos     86     77     4.1     18.9
## 7 R7    Quilombo   Santos     86     63    14.2     42.1
## 8 R11   <NA>      <NA>      NA      241     2.2     20.3
```

Note que a coluna de `R11` contém `NA` nas colunas `Bacia` e `Município`, pois este riacho não está presente na tabela `regiao`.

#### 7.1.4.3 Função `inner_join()`

Resgata apenas as linhas coincidentes a **ambas** as tabelas.

```
regiao %>% inner_join(y = habitat)
```

#### 7.1.4.4 Função `anti_join()`

Retorna todas as linhas da tabela à esquerda em que existem correspondências com a de direita. Retorna todas as colunas de ambas. Se houver várias correspondências, todas as combinações serão retornadas. Testes os comandos abaixo.

```
regiao %>% anti_join(y = habitat)
```

```
habitat %>% anti_join(y = regiao)
```

#### 7.1.4.5 Função full\_join()

Retorna todas as linhas e colunas das duas tabelas. Nas células em que não houver valores correspondentes, retorna NA na posição faltante.

```
regiao %>% full_join(y = habitat)
```

```
habitat %>% semi_join(y = regiao)
```

### 7.1.5 Criando e modificando colunas: função mutate()

Voltemos à base de dados Doubs river.

```
library(ade4)
data(doubs)
ambiente <- doubs$env
head(ambiente)
```

```
##   dfs alt   slo flo pH har pho nit amm oxy bdo
## 1    3 934 6.176 84 79 45   1  20   0 122  27
## 2   22 932 3.434 100 80 40   2  20  10 103  19
## 3   102 914 3.638 180 83 52   5  22   5 105  35
## 4   185 854 3.497 253 80 72  10  21   0 110  13
## 5   215 849 3.178 264 81 84  38  52  20  80  62
## 6   324 846 3.497 286 79 60  20  15   0 102  53
```

Veja que a coluna pH está dada em  $\text{ph} \times 10$ . Vamos retornar à escala unitária.

```
ambiente %>%
  mutate(pH = pH/10)
```

```
##   dfs alt   slo flo pH har pho nit amm oxy bdo
## 1    3 934 6.176 84 7.9 45   1  20   0 122  27
## 2   22 932 3.434 100 8.0 40   2  20  10 103  19
## 3   102 914 3.638 180 8.3 52   5  22   5 105  35
## 4   185 854 3.497 253 8.0 72  10  21   0 110  13
## 5   215 849 3.178 264 8.1 84  38  52  20  80  62
```

```
## 6 324 846 3.497 286 7.9 60 20 15 0 102 53
## 7 268 841 4.205 400 8.1 88 7 15 0 111 22
## 8 491 792 3.258 130 8.1 94 20 41 12 70 81
## 9 705 752 2.565 480 8.0 90 30 82 12 72 52
## 10 990 617 4.605 1000 7.7 82 6 75 1 100 43
## 11 1234 483 3.738 1990 8.1 96 30 160 0 115 27
## 12 1324 477 2.833 2000 7.9 86 4 50 0 122 30
## 13 1436 450 3.091 2110 8.1 98 6 52 0 124 24
## 14 1522 434 2.565 2120 8.3 98 27 123 0 123 38
## 15 1645 415 1.792 2300 8.6 86 40 100 0 117 21
## 16 1859 375 3.045 1610 8.0 88 20 200 5 103 27
## 17 1985 348 1.792 2430 8.0 92 20 250 20 102 46
## 18 2110 332 2.197 2500 8.0 90 50 220 20 103 28
## 19 2246 310 1.792 2590 8.1 84 60 220 15 106 33
## 20 2477 286 2.197 2680 8.0 86 30 300 30 103 28
## 21 2812 262 2.398 2720 7.9 85 20 220 10 90 41
## 22 2940 254 2.708 2790 8.1 88 20 162 7 91 48
## 23 3043 246 2.565 2880 8.1 97 260 350 115 63 164
## 24 3147 241 1.386 2976 8.0 99 140 250 60 52 123
## 25 3278 231 1.792 3870 7.9 100 422 620 180 41 167
## 26 3579 214 1.792 3910 7.9 94 143 300 30 62 89
## 27 3732 206 2.565 3960 8.1 90 58 300 26 72 63
## 28 3947 195 1.386 4320 8.3 100 74 400 30 81 45
## 29 4220 183 1.946 6770 7.8 110 45 162 10 90 42
## 30 4530 172 1.099 6900 8.2 109 65 160 10 82 44
```

Vamos agora criar uma variável categórica `pH_cat` com os níveis `Elevado` se maior ou igual a 8 e `Neutro` caso contrário.

```
ambiente %>%
  mutate(pH = pH/10) %>%
  mutate(pH_cat = ifelse(pH < 8, yes = "Neutro", no = "Elevado"))
```

A nova variável está no final da tabela. Vamos colocá-la logo após a coluna `pH`

```
ambiente %>%
  mutate(pH = pH/10) %>%
  mutate(pH_cat = ifelse(pH < 8, yes = "Neutro", no = "Elevado"), .after = pH)
```

### 7.1.6 Unindo colunas: função `unite()`

Todas as funções vistas acima neste capítulo são do pacote `dplyr`. A função `unite()` é do pacote `tidyr` e permite unir duas colunas. Para isto, vamos retornar à tabela `Iris` e selecionar algumas linhas para exemplificar a união de colunas.

```
iris2 <- iris %>% filter(Sepal.Length > 5.5 & Sepal.Length < 6.1)
iris2
```

| ##    | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species    |
|-------|--------------|-------------|--------------|-------------|------------|
| ## 1  | 5.8          | 4.0         | 1.2          | 0.2         | setosa     |
| ## 2  | 5.7          | 4.4         | 1.5          | 0.4         | setosa     |
| ## 3  | 5.7          | 3.8         | 1.7          | 0.3         | setosa     |
| ## 4  | 5.7          | 2.8         | 4.5          | 1.3         | versicolor |
| ## 5  | 5.9          | 3.0         | 4.2          | 1.5         | versicolor |
| ## 6  | 6.0          | 2.2         | 4.0          | 1.0         | versicolor |
| ## 7  | 5.6          | 2.9         | 3.6          | 1.3         | versicolor |
| ## 8  | 5.6          | 3.0         | 4.5          | 1.5         | versicolor |
| ## 9  | 5.8          | 2.7         | 4.1          | 1.0         | versicolor |
| ## 10 | 5.6          | 2.5         | 3.9          | 1.1         | versicolor |
| ## 11 | 5.9          | 3.2         | 4.8          | 1.8         | versicolor |
| ## 12 | 6.0          | 2.9         | 4.5          | 1.5         | versicolor |
| ## 13 | 5.7          | 2.6         | 3.5          | 1.0         | versicolor |
| ## 14 | 5.8          | 2.7         | 3.9          | 1.2         | versicolor |
| ## 15 | 6.0          | 2.7         | 5.1          | 1.6         | versicolor |
| ## 16 | 6.0          | 3.4         | 4.5          | 1.6         | versicolor |
| ## 17 | 5.6          | 3.0         | 4.1          | 1.3         | versicolor |
| ## 18 | 5.8          | 2.6         | 4.0          | 1.2         | versicolor |
| ## 19 | 5.6          | 2.7         | 4.2          | 1.3         | versicolor |
| ## 20 | 5.7          | 3.0         | 4.2          | 1.2         | versicolor |
| ## 21 | 5.7          | 2.9         | 4.2          | 1.3         | versicolor |
| ## 22 | 5.7          | 2.8         | 4.1          | 1.3         | versicolor |
| ## 23 | 5.8          | 2.7         | 5.1          | 1.9         | virginica  |
| ## 24 | 5.7          | 2.5         | 5.0          | 2.0         | virginica  |
| ## 25 | 5.8          | 2.8         | 5.1          | 2.4         | virginica  |
| ## 26 | 6.0          | 2.2         | 5.0          | 1.5         | virginica  |
| ## 27 | 5.6          | 2.8         | 4.9          | 2.0         | virginica  |
| ## 28 | 6.0          | 3.0         | 4.8          | 1.8         | virginica  |
| ## 29 | 5.8          | 2.7         | 5.1          | 1.9         | virginica  |
| ## 30 | 5.9          | 3.0         | 5.1          | 1.8         | virginica  |

Vamos criar uma nova coluna **Genus** e uní-la às coluna **Species**.

```
iris2 %>%
  mutate(Genus = rep("Iris", times = nrow(iris2)), .before = Species)
```

| ##   | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Genus | Species |
|------|--------------|-------------|--------------|-------------|-------|---------|
| ## 1 | 5.8          | 4.0         | 1.2          | 0.2         | Iris  | setosa  |
| ## 2 | 5.7          | 4.4         | 1.5          | 0.4         | Iris  | setosa  |
| ## 3 | 5.7          | 3.8         | 1.7          | 0.3         | Iris  | setosa  |

```
## 4      5.7      2.8      4.5      1.3 Iris versicolor
## 5      5.9      3.0      4.2      1.5 Iris versicolor
## 6      6.0      2.2      4.0      1.0 Iris versicolor
## 7      5.6      2.9      3.6      1.3 Iris versicolor
## 8      5.6      3.0      4.5      1.5 Iris versicolor
## 9      5.8      2.7      4.1      1.0 Iris versicolor
## 10     5.6      2.5      3.9      1.1 Iris versicolor
## 11     5.9      3.2      4.8      1.8 Iris versicolor
## 12     6.0      2.9      4.5      1.5 Iris versicolor
## 13     5.7      2.6      3.5      1.0 Iris versicolor
## 14     5.8      2.7      3.9      1.2 Iris versicolor
## 15     6.0      2.7      5.1      1.6 Iris versicolor
## 16     6.0      3.4      4.5      1.6 Iris versicolor
## 17     5.6      3.0      4.1      1.3 Iris versicolor
## 18     5.8      2.6      4.0      1.2 Iris versicolor
## 19     5.6      2.7      4.2      1.3 Iris versicolor
## 20     5.7      3.0      4.2      1.2 Iris versicolor
## 21     5.7      2.9      4.2      1.3 Iris versicolor
## 22     5.7      2.8      4.1      1.3 Iris versicolor
## 23     5.8      2.7      5.1      1.9 Iris virginica
## 24     5.7      2.5      5.0      2.0 Iris virginica
## 25     5.8      2.8      5.1      2.4 Iris virginica
## 26     6.0      2.2      5.0      1.5 Iris virginica
## 27     5.6      2.8      4.9      2.0 Iris virginica
## 28     6.0      3.0      4.8      1.8 Iris virginica
## 29     5.8      2.7      5.1      1.9 Iris virginica
## 30     5.9      3.0      5.1      1.8 Iris virginica
```

```
iris3 <- iris2 %>%
  mutate(Genus = rep("Iris", times = nrow(iris2)), .before = Species) %>%
  unite(col = scientific_name, Genus, Species, sep = " ")
iris3
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width  scientific_name
## 1      5.8      4.0      1.2      0.2      Iris setosa
## 2      5.7      4.4      1.5      0.4      Iris setosa
## 3      5.7      3.8      1.7      0.3      Iris setosa
## 4      5.7      2.8      4.5      1.3 Iris versicolor
## 5      5.9      3.0      4.2      1.5 Iris versicolor
## 6      6.0      2.2      4.0      1.0 Iris versicolor
## 7      5.6      2.9      3.6      1.3 Iris versicolor
## 8      5.6      3.0      4.5      1.5 Iris versicolor
## 9      5.8      2.7      4.1      1.0 Iris versicolor
## 10     5.6      2.5      3.9      1.1 Iris versicolor
## 11     5.9      3.2      4.8      1.8 Iris versicolor
```

```
## 12      6.0      2.9      4.5      1.5 Iris versicolor
## 13      5.7      2.6      3.5      1.0 Iris versicolor
## 14      5.8      2.7      3.9      1.2 Iris versicolor
## 15      6.0      2.7      5.1      1.6 Iris versicolor
## 16      6.0      3.4      4.5      1.6 Iris versicolor
## 17      5.6      3.0      4.1      1.3 Iris versicolor
## 18      5.8      2.6      4.0      1.2 Iris versicolor
## 19      5.6      2.7      4.2      1.3 Iris versicolor
## 20      5.7      3.0      4.2      1.2 Iris versicolor
## 21      5.7      2.9      4.2      1.3 Iris versicolor
## 22      5.7      2.8      4.1      1.3 Iris versicolor
## 23      5.8      2.7      5.1      1.9 Iris virginica
## 24      5.7      2.5      5.0      2.0 Iris virginica
## 25      5.8      2.8      5.1      2.4 Iris virginica
## 26      6.0      2.2      5.0      1.5 Iris virginica
## 27      5.6      2.8      4.9      2.0 Iris virginica
## 28      6.0      3.0      4.8      1.8 Iris virginica
## 29      5.8      2.7      5.1      1.9 Iris virginica
## 30      5.9      3.0      5.1      1.8 Iris virginica
```

*Observação:* a função `unite()` **\*exclui** as colunas que foram unificadas da tabela.

### 7.1.7 Reformatando tabelas: funções `spread()` e `gather()`

Importe a tabela `HubbardBrook.csv` com dados anuais de vazão e precipitação em dois riachos de áreas desmatadas e referência. Os dados foram retirados de [tiee.esa.org](http://tiee.esa.org) e podem ser obtidos em `HubbardBrook.csv`.

```
## # A tibble: 62 x 4
##   Year Treatment Flow Precipitation
##   <dbl> <chr>      <dbl>      <dbl>
## 1 1958 Deforested 645.      1168.
## 2 1959 Deforested 1012.     1483.
## 3 1960 Deforested 825.      1321.
## 4 1961 Deforested 470.       980.
## 5 1962 Deforested 777.     1232.
## 6 1963 Deforested 774.     1139.
## 7 1964 Deforested 712.     1175.
## 8 1965 Deforested 599.     1115.
## 9 1966 Deforested 1189.     1222.
## 10 1967 Deforested 1132.     1315.
## # ... with 52 more rows
```

```
stream <- read_csv("C:/seu_caminho/IntroR/HubbardBrook.csv")
stream
```

A função `spread()` reorganiza dados do formato longo para o formato largo. Vamos fazer isto abaixo somente para a variável `Flow` e excluindo `Precipitation`.

```
stream_largo <- stream %>%
  select(-Precipitation) %>%
  spread(key = Treatment, value = Flow)
stream_largo
```

```
## # A tibble: 31 x 3
##   Year Deforested Reference
##   <dbl>      <dbl>      <dbl>
## 1 1958         645.        567.
## 2 1959        1012.        918.
## 3 1960         825.        752.
## 4 1961         470.        436.
## 5 1962         777.        699.
## 6 1963         774.        663.
## 7 1964         712.        630.
## 8 1965         599.        547.
## 9 1966        1189.        727.
## 10 1967        1132.        781.
## # ... with 21 more rows
```

Enquanto a função `gather()` faz o caminho reverso.

```
stream_longo <- stream_largo %>%
  gather(key = Desmatamento, value = Flow, -Year)
stream_longo
```

```
## # A tibble: 62 x 3
##   Year Desmatamento Flow
##   <dbl> <chr>      <dbl>
## 1 1958 Deforested    645.
## 2 1959 Deforested   1012.
## 3 1960 Deforested    825.
## 4 1961 Deforested    470.
## 5 1962 Deforested    777.
## 6 1963 Deforested    774.
## 7 1964 Deforested    712.
## 8 1965 Deforested    599.
## 9 1966 Deforested   1189.
```



```
## 10 1967 Deforested 1132.  
## # ... with 52 more rows
```



## Chapter 8

# Um gráfico em camadas: o pacote ggplot2

No capítulo 3 vimos os tipos básicos de gráficos em R. Vamos agora rever estes gráficos utilizando o pacote `ggplot2`, mais um pacote do grupo `tidyverse`. O `ggplot2` fornece uma gramática coesa para os elementos de um gráfico, o que torna o aprendizado mais simples, rápido e os códigos mais reutilizáveis. Uma segunda vantagem é a elevada capacidade de formatação dos elementos gráficos com relativa simplicidade.

Aqui veremos somente uma introdução para que você acompanhe o restante do material. Você pode buscar por `ggplot2` tutorial para encontrar uma imensa variedade de excelentes tutoriais na internet.

Uma das primeiras características que devemos ter em mente é que o `ggplot2` gera gráficos a partir da coluna de um data frame.

Se ainda não o fez, instale e carregue o `ggplot2`.

```
install.packages("ggplot2")  
library(ggplot2)
```

Iremos utilizar também os pacotes `readr` e o `dplyr` que podem ser instalados e carregados por:

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

## 8.1 Hubbard Brook stream flow

Vamos fazer um histograma dos dados de vazão da tabela HubbardBrook.csv (dados retirados de tiee.esa.org). Veja novamente a tabela:

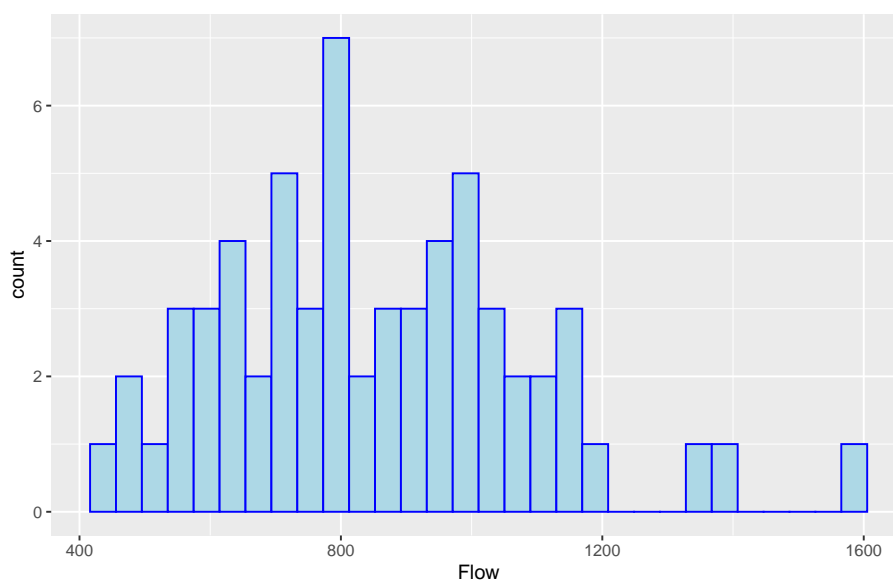
```
hub = read_csv("C:/seu_caminho/IntroR/HubbardBrook.csv")
hub
```

```
## # A tibble: 6 x 4
##   Year Treatment   Flow Precipitation
##   <dbl> <chr>      <dbl>      <dbl>
## 1 1958 Deforested  645.      1168.
## 2 1959 Deforested 1012.      1483.
## 3 1960 Deforested  825.      1321.
## 4 1961 Deforested  470.       980.
## 5 1962 Deforested  777.      1232.
## 6 1963 Deforested  774.      1139.
```

### 8.1.1 Entendendo a gramática do ggplot

Considere o histograma da variável Flow.

```
ggplot(data = hub, mapping = aes(x = Flow)) +
  geom_histogram(color = "blue", fill = "lightblue")
```



No comando existem dois tipos de informação, separadas pelo símbolo `+`. Este símbolo marca o fim de uma camada e início da outra. No `ggplot()` cada camada adiciona um elemento novo ou formata um elemento existente no gráfico. A ordem em que as camadas são inseridas raramente importa, ainda que seja interessante inseri-las de modo que facilite a leitura do código. No exemplo acima temos as camadas gedaras por:

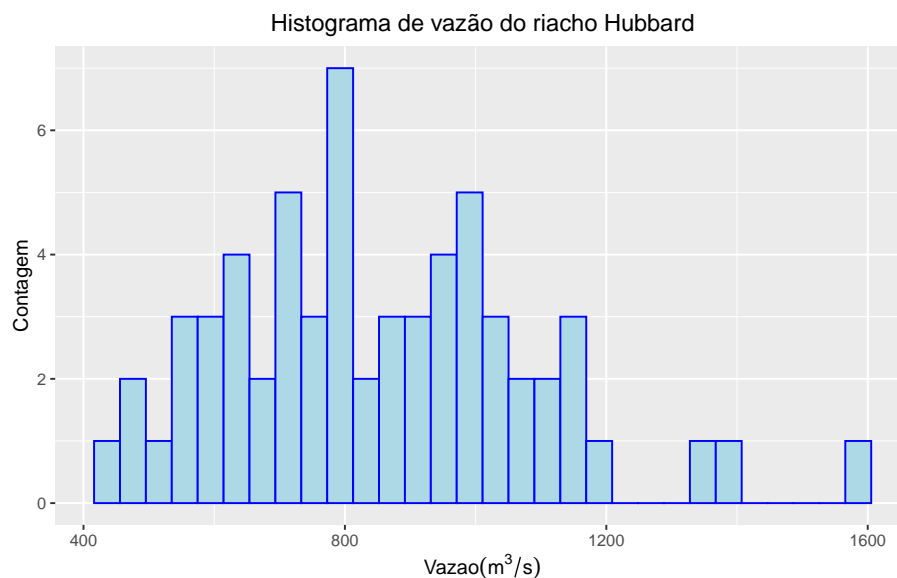
- `ggplot()`: esta função mostra um argumento `data =`, onde informamos o nome do `data.frame` de onde serão lidos os dados (*No `ggplot`, os dados sempre devem estar dispostos dentro de um data frame*). No segundo argumento `mapping =`, informamos sobre a **estética do gráfico**. Por hora, esta estética foi limitada a `aes(x = Flow)`, dizendo que nosso gráfico irá conter no eixo `x` a variável `Flow` do data frame.

Até este momento não definimos qual será o tipo gráfico.

- Na segunda camada estabelecemos a **geometria** do gráfico, neste caso composta por um histograma (`geom_histogram`), onde a cor da borda foi definida por `color = "blue"` e o preenchimento por `fill = "lightblue"`.

Vamos adicionar algumas formatações adicionais:

```
ggplot(data = hub, mapping = aes(x = Flow)) +  
  geom_histogram(color = "blue", fill = "lightblue") +  
  labs(title = "Histograma de vazão do riacho Hubbard",  
        x = bquote(Vazao (m3/s)),  
        y = "Contagem") +  
  theme(plot.title = element_text(hjust = 0.5))
```

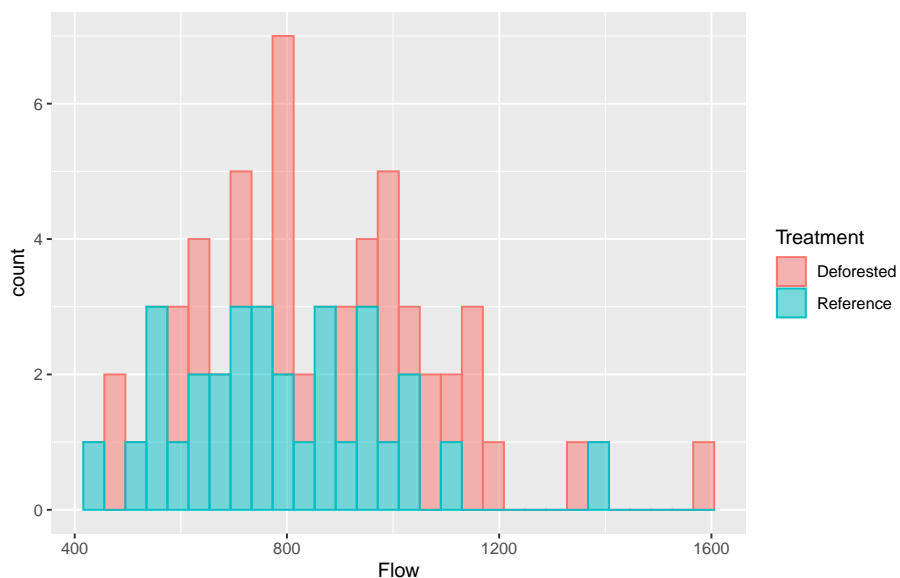


Formatamos agora o **título**, os nomes dos eixos x e y e centralizamos o título (`theme(plot.title = element_text(hjust = 0.5))`).

Embora possa parecer muita informação de uma única vez (e de fato é), o ponto é que você não precisará decorar nenhuma delas (mas se utilizar, acabará decorando!!). Uma simples busca por `centralize title ggplot2` te levará a estes comandos.

No histograma anterior, a vazão foi medida anualmente em trechos Deforested e em trechos Deforested. Vamos então adicionar esta variável à *estética* (`aes()`) do gráfico, diferenciando os grupos em função da cor.

```
ggplot(data = hub, mapping = aes(x = Flow, color = Treatment, fill = Treatment)) +
  geom_histogram(alpha = 0.5)
```



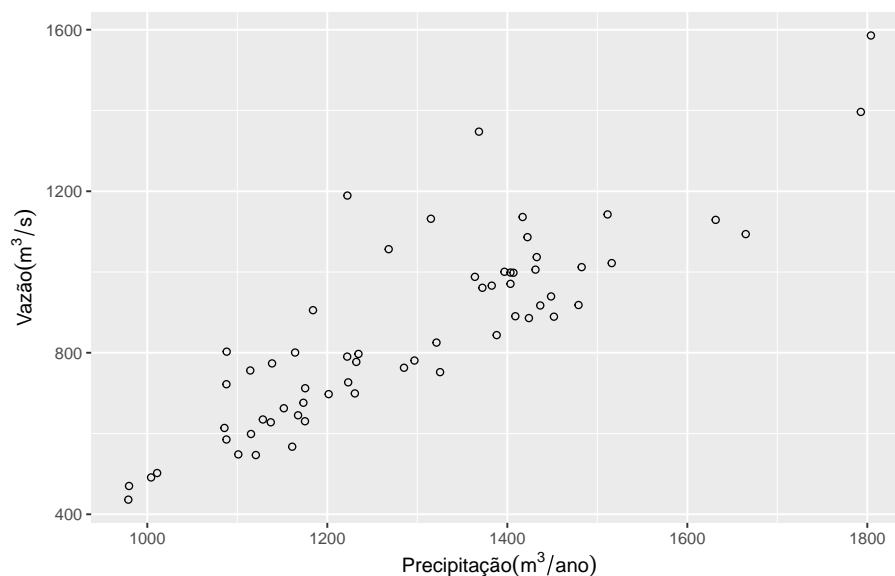
### 8.1.2 Outras *Geometrias* gráficas

Além dos histogramas temos muitas outras geometrias gráficas do tipo `geom_NOME()`. Algumas muito utilizadas são:

- `geom_abline()`
- `geom_bar()`
- `geom_boxplot()`
- `geom_line()`
- `geom_point()`
- `geom_smooth()`
- `geom_text()`

## 8.2 Gáfico de dispersão

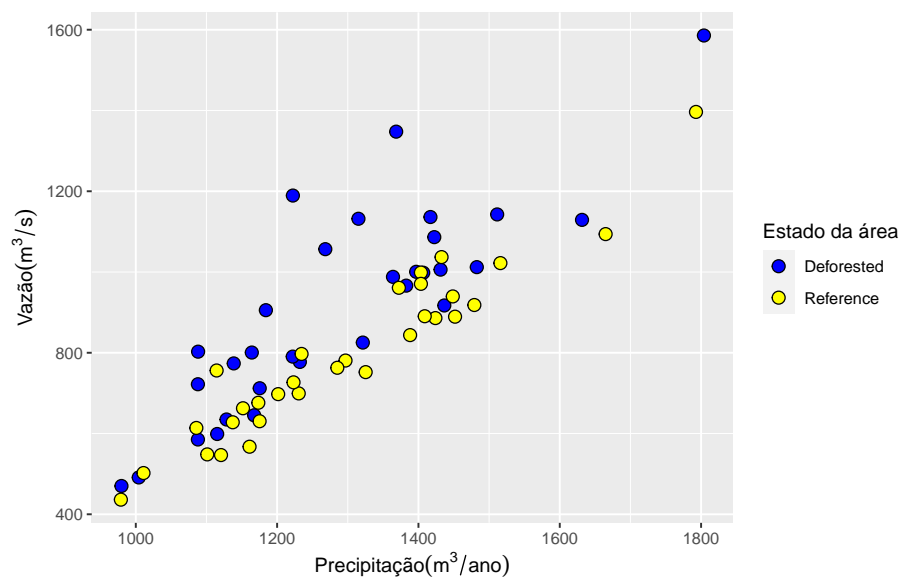
```
ggplot(data = hub, mapping = aes(x = Precipitation, y = Flow)) +
  geom_point(shape = 21) +
  labs(y = bquote(Vazão (m3/s)),
       x = bquote(Precipitação (m3/ano)))
```



Adicionando o `Treatment` à cor do gráfico e formatando a legenda.

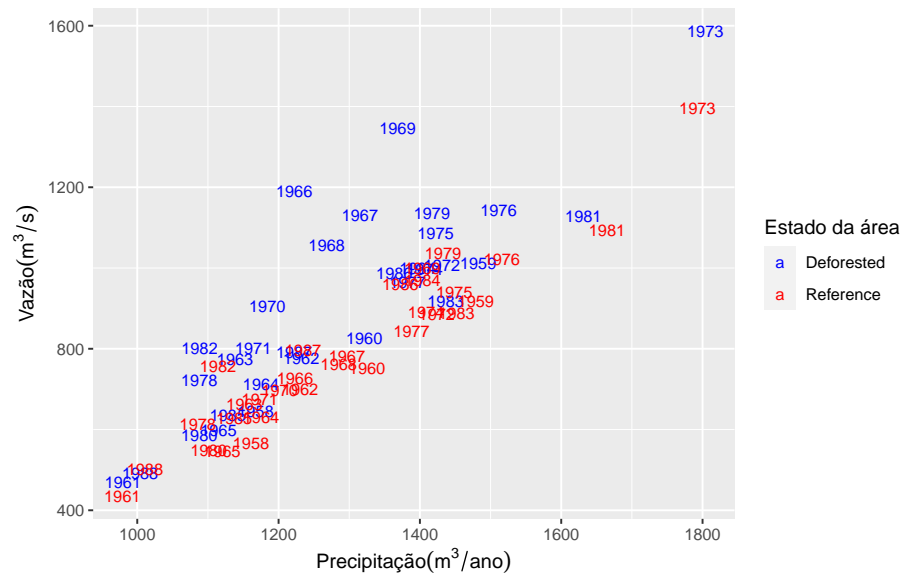
```
ggplot(data = hub, mapping = aes(x = Precipitation, y = Flow, fill = Treatment)) +
  geom_point(shape = 21, size = 3) +
  labs(y = bquote(Vazão (m3/s)),
       x = bquote(Precipitação (m3/ano))) +
  guides(fill=guide_legend(title="Estado da área")) +
  scale_fill_manual(values = c("blue", "yellow"))
```





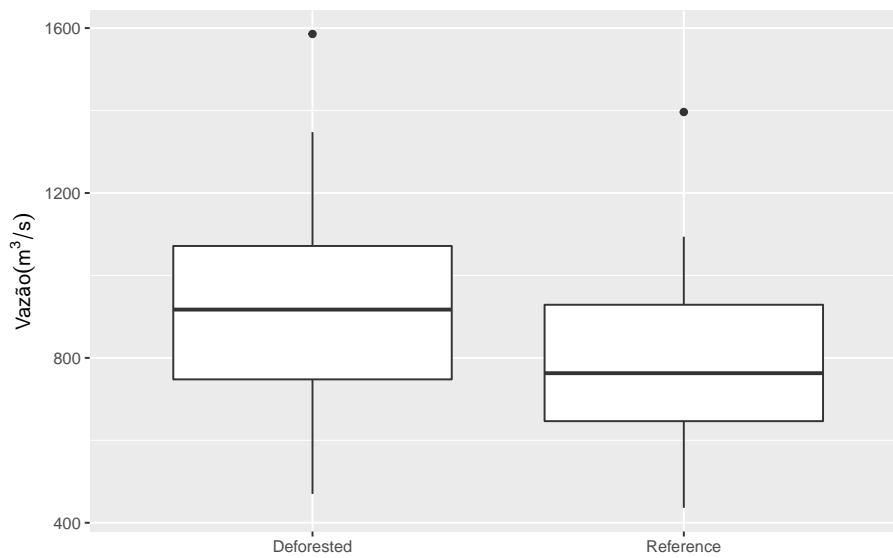
Adicionando um texto com informação do Ano de mensuração.

```
ggplot(data = hub, mapping = aes(x = Precipitation, y = Flow, label = Year,
                                color = Treatment)) +
  geom_text(size = 3) +
  labs(y = bquote(Vazão (m3/s)),
       x = bquote(Precipitação (m3/ano))) +
  guides(color=guide_legend(title="Estado da área")) +
  scale_color_manual(values = c("blue", "red"))
```



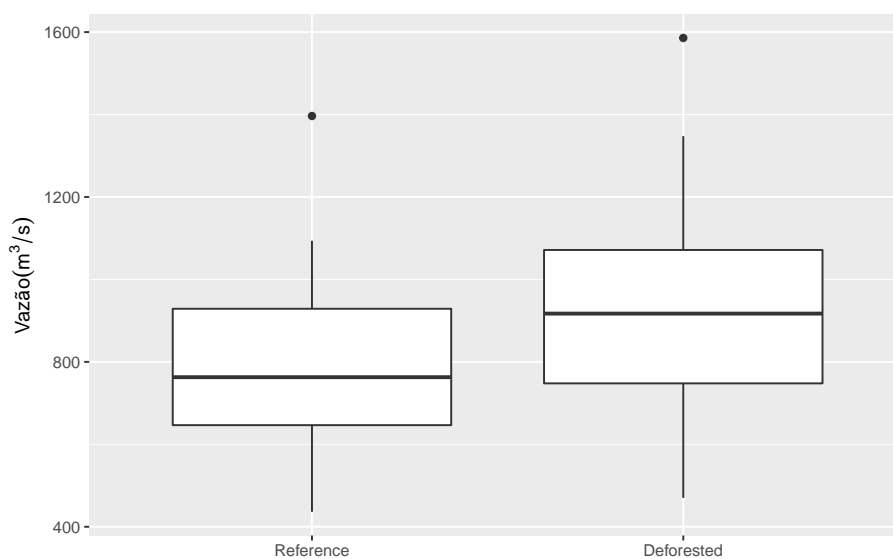
### 8.3 Boxplot

```
ggplot(data = hub, mapping = aes(x = Treatment, y = Flow)) +
  geom_boxplot() +
  labs(y = bquote(Vazão (m3/s)),
       x = "")
```



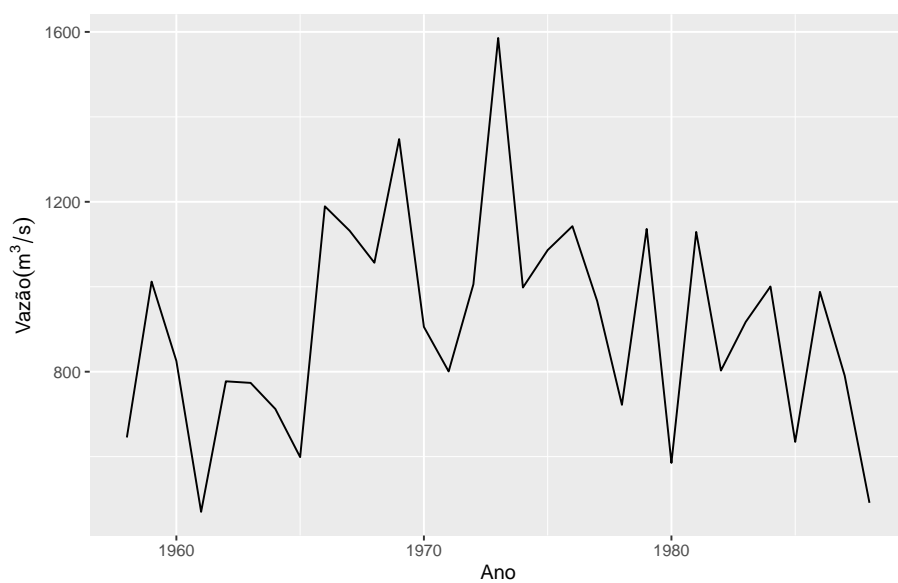
Vamos inverter a ordem dos boxplots.

```
ggplot(data = hub, mapping = aes(x = Treatment, y = Flow)) +  
  geom_boxplot() +  
  labs(y = bquote(Vazão (m3/s)),  
       x = "") +  
  scale_x_discrete(  
    limits = c("Reference", "Deforested"))
```



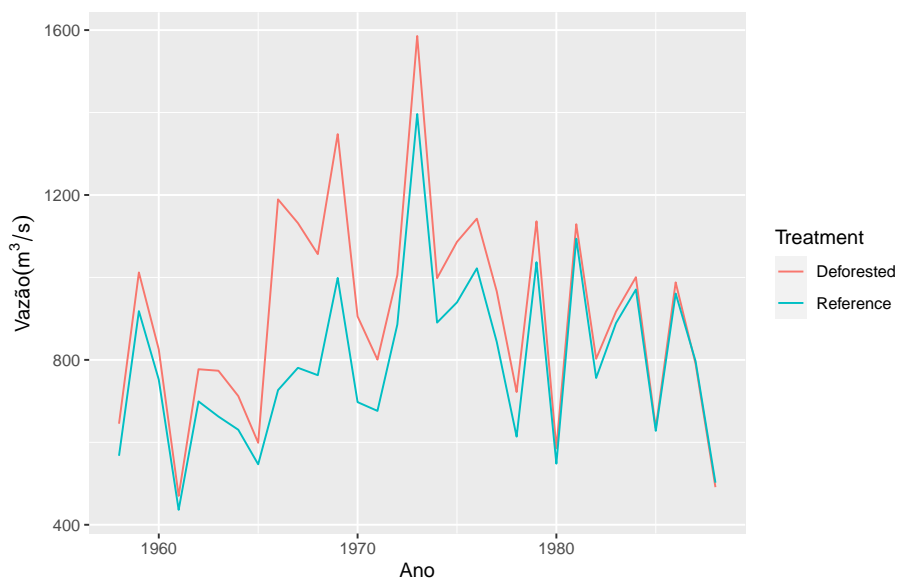
## 8.4 Gráfico de linhas

```
hub %>%
  filter(Treatment == "Deforested") %>%
  ggplot(mapping = aes(x = Year, y = Flow)) +
    geom_line() +
    labs(y = bquote(Vazão (m3/s)),
         x = "Ano")
```



Na figura, utilizamos o operador **pipe**, e a função **filter()** para extrair somente as linhas do riacho desmatado. Vamos adicionar os dois riachos.

```
ggplot(hub, mapping = aes(x = Year, y = Flow, color = Treatment)) +  
  geom_line() +  
  labs(y = bquote(Vazão (m3/s)),  
       x = "Ano")
```

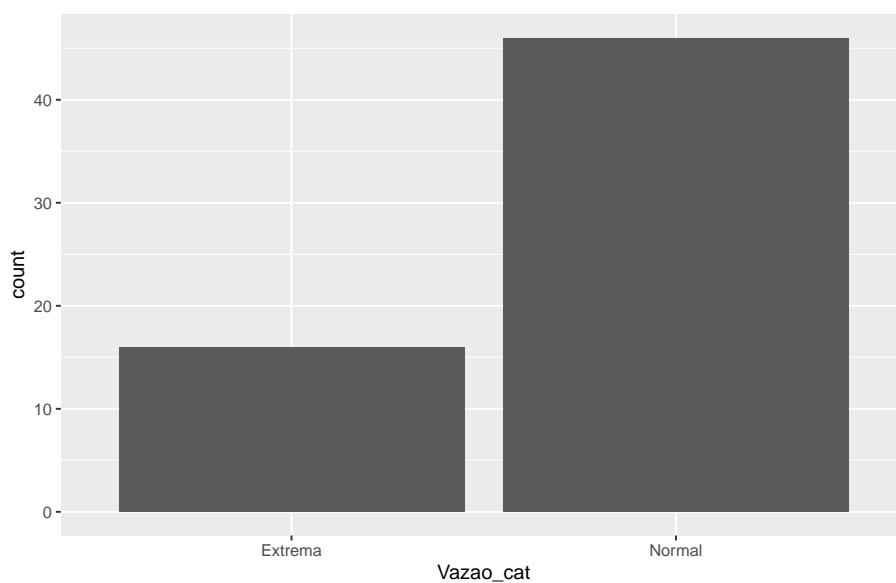


## 8.5 Gráfico de barras

Vamos criar uma variável categórica `Vazao_cat` contendo os níveis **Extrema** (se `Flow`  $\geq 1000 \text{ m}^3/\text{s}$ ) e **Normal** caso contrário. Em seguida vamos contar o número de observações com vazão extrema.

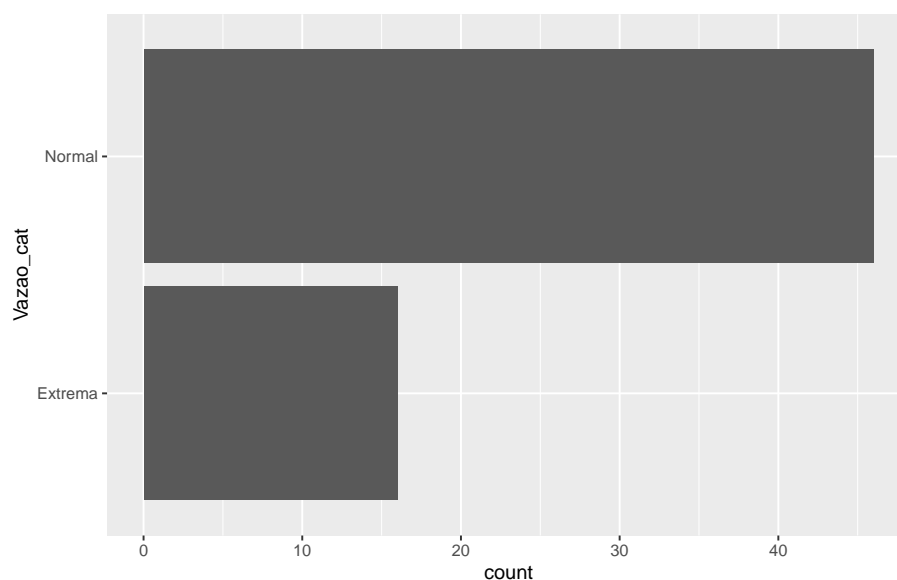
```
extremo <- 1000

hub %>%
  mutate(Vazao_cat = ifelse(Flow >= extremo,
                             yes = "Extrema",
                             no = "Normal")) %>%
  ggplot(mapping = aes(x = Vazao_cat)) +
  geom_bar()
```



Se dissermos que a variável está em *y* (`aes(y = Vazao_cat)`) o gráfico fica na horizontal.

```
hub %>%  
  mutate(Vazao_cat = ifelse(Flow >= extremo, yes = "Extrema", no = "Normal")) %>%  
  ggplot(mapping = aes(y = Vazao_cat)) +  
  geom_bar()
```

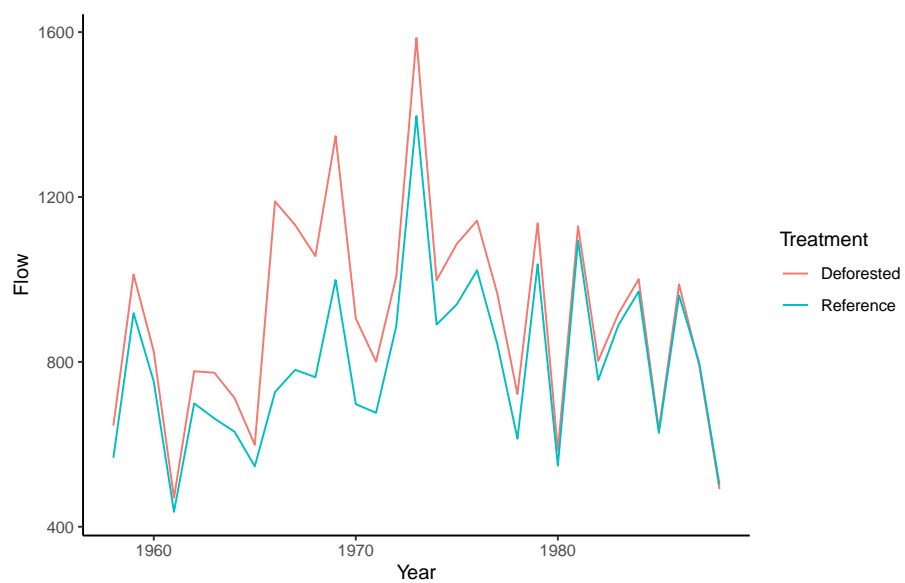


## 8.6 Temas no ggplot2

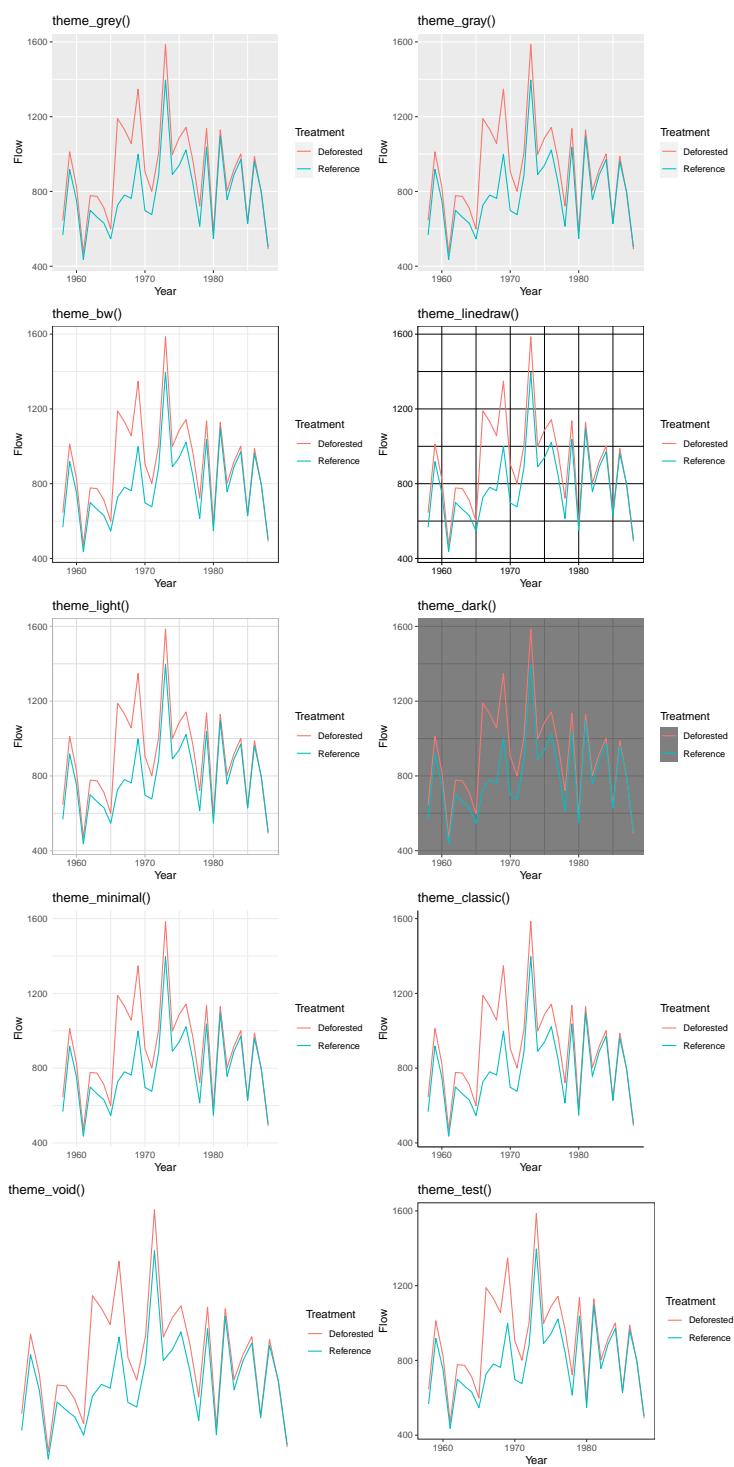
Embora possamos formatar todos os elementos gráficos, o `ggplot2` vem com temas pré-formatados que facilita este processo. Vejamos alguns. Para escolher um tema adicionamos uma camada com seu nome (`theme_NOME()`). Veja o exemplo:

```
ggplot(hub, mapping = aes(x = Year, y = Flow, color = Treatment)) +  
  geom_line() +  
  theme_classic()
```





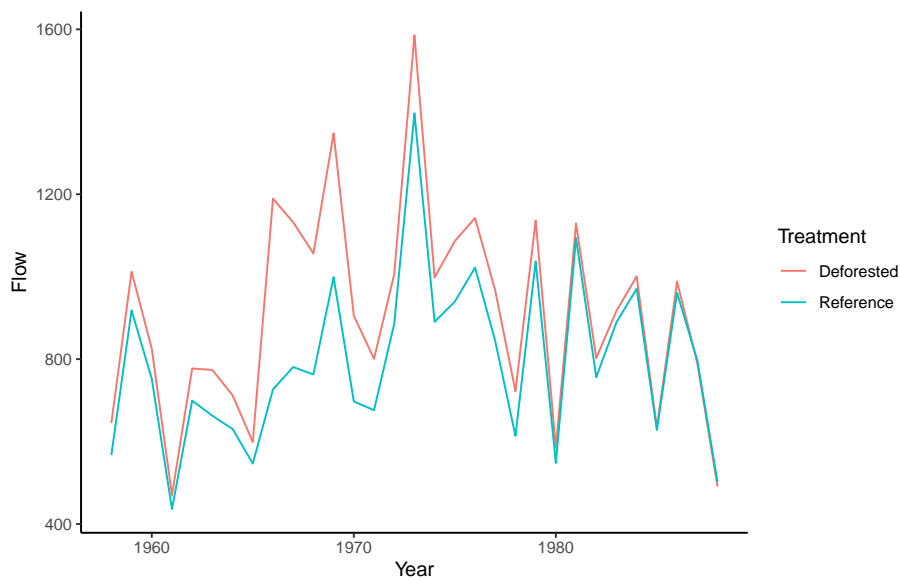
Os temas básicos estão exemplificados abaixo. Além destes, o pacote **ggthemes** oferece uma extensa variedade de outras formatações.



## 8.7 Salvando uma figura gerada pelo ggplot2.

Uma forma simples de salvar um gráfico gerados pelo ggplot2 é utilizando a função `ggsave()`.

```
ggplot(hub, mapping = aes(x = Year, y = Flow, color = Treatment)) +  
  geom_line() +  
  theme_classic()  
  
ggsave(filename = "Exemplo_ggsave.png",  
        device = "png",  
        width = 20,  
        height = 20,  
        units = "cm",  
        dpi = 480)
```



Por padrão a função `ggsave()` salva o ultimo gráfico criado. Porém se existe um objeto salvo, você pode especificá-lo com o argumento `plot = objeto_grafico`.



# Bibliography

- Borcard, D., Gillet, F., and Legendre, P. (2018). *Numerical ecology with R*. Springer.
- Dray, S., Dufour, A., and Thioulouse, J. (2015). ade 4: analysis of ecological data: exploratory and euclidean methods in environmental sciences. r package version 1.7-2.
- Verneaux, J. (1973). *Cours d'eau de Franche-Comté (Massif du Jura). Recherches écologiques sur le réseau hydrographique du Doubs. Essai de biotypologie*. PhD thesis, Thèse d'état, Besançon.