R Zero - do simples ao complexo

Diogo J. A. Silva

2024-02-04

Contents

Chapter 1

Bem-vindos

R zero foi criado para introduzir e orientar novos usuários acadêmicos do programa R e RStudio do absoluto ZERO. A ideia é fornecer os conceitos essenciais para a utilização do R, guiando o leitor através da linguagem R, introduzindo pílulas de conceitos que irão se complementando ao longo dos capítulos.

Já existem livros muito bons sobre o assunto, mas então por que utilizar esse em específico? Eu acredito que o diferencial desse livro é como ele está organizado. Começando com conceitos básicos, e adicionando complexidade de uma forma fluída e didática. Além da introdução ao R, é mostrado como trabalhar de forma eficiente utilizando projetos reprodutíveis e fáceis de compartilhar. Muitos conceitos serão deixados de lado para focar no que realmente é essencial e assim criando um esqueleto conceitual que pode ser aprimorado com o tempo. Portanto, nesse livro, iniciantes terão uma boa base introdutória no R mostrando de forma prática como trabalhar com seus dados de forma mais simples.

A ideia é que esse seja o primeiro livro de uma coleção de quatro livros que se complementarão:

- R zero: uma introdução ao R
- R zero: manipulação e limpeza de dados
- R zero: análise estatística
- R zero: visualização de dados

Juntos esses materiais fornecerão, de forma simples, o arcabouço essencial da ciência de dados acadêmica.

Chapter 2

Pré-requisitos

2.1 Instalando R e RStudio

Para utilizar este livro de maneira otimizada, é necessário ter o **R** e o **RStudio** instalados. Não vou abordar a instalação aqui, pois acredito que você é capaz de encontrar tutoriais simples feitos por uma crianca de 8 anos no YouTube.

Brincadeiras à parte, a ideia é facilitar sua vida. Então saiba o seguinte. O R é o programa principal, o RStudio é uma "skin", uma "roupagem" do R. Alguns chamam essa roupagem de Ambiente de Desenvolvimento Integrado (IDE - Integrated Development Environment), mas você só precisa saber disso por um motivo: você precisa instalar o R primeiro, depois sua skin (o RStudio), caso contrário ocorrerão problemas.

O **RStudio** requer **R 3.3.0** ou superior. Clique no link abaixo, acesse o site para baixar o **R** e escolha uma versão compatível com o sistema operacional do seu computador.

1. Para baixar e instalar o **R**, acesse:

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

2. Em seguida, faça o download e instale o **RStudio** através do link:

https://posit.co/download/rstudio-desktop/#download

Chapter 3

R para iniciantes

Caro iniciante, meu objetivo aqui é mostrar que utilizar o R é muito divertido e recompensador. Você vai perceber que cada código rodado (e que funciona) vai te dar um pouquinho de dopamina e uma sensação de prazer. Claro que alguns erros vão te deixar maluco, mas você vai perceber que tudo é culpa sua. Mas não se preocupe, se é culpa sua, você pode consertá-los :)

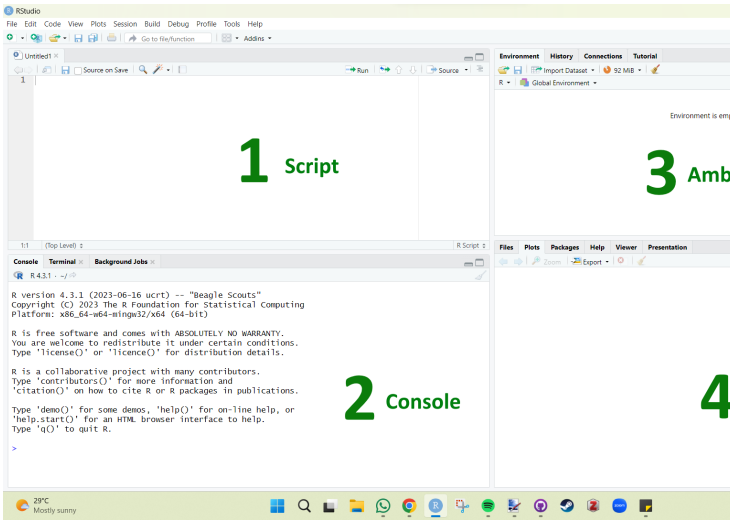
A tendência é que, quanto mais você utilizar o R, mais fácil sua vida se tornará, e, apesar da curva de aprendizagem ser um pouco desestimuladora, é 1000 vezes recompensadora. Como alguns gostam de dizer, você conseguirá fazer um gráfico até 30 vezes mais rápido! Não sei de onde veio esse cálculo, mas se está na internet, é verdade.

3.1 Entendendo o RStudio

Tudo que faremos será através do R Studio devido à organização que ele nos fornece. O RStudio é uma roupagem para o R, que nos oferece um ambiente de trabalho mais amigável e organizado. Se você seguiu as instruções de instalação corretamente, terá dois ícones, o do R base e o do RStudio. Vamos abrir o programa RStudio dando dois cliques sobre seu ícone (Figura 1).

Figure 3.1: Figura 1. Icone do RStudio

Ao abrir o R Studio, você verá o seguinte:



Não tem a aparência convencional dos programas estatísticos, mas isso acontece porque não se realizam tarefas clicando em abas ou botões. No R Studio, você executa comandos por meio de códigos! No futuro, é provável que exista um programa com o qual você poderá conversar em qualquer idioma, e ele realizará as tarefas que você pedir. Não, pera, isso já existe! Chama-se inteligência artificial! Entretanto, não se deixe enganar. Os conhecimentos íntimos de como pensar, organizar, analisar e mostrar seus dados são importantíssimos para todo um projeto científico. Mas se isso não te convence, as melhores inteligências artificiais do mercado são pagas e podem te retornar resultados enganosos. Então, tenha muito cuidado e utilize as inteligências artificiais disponíveis apenas como uma ferramenta auxiliar!

Vamos ao que importa: entender como o R funciona. O R possui uma linguagem,

e tudo que você vai fazer no R é utilizando essa linguagem, que são verdadeiros comandos. Importe essa planilha! Crie esse gráfico! Faça a média dessa variável! E por aí vai. Então, vamos dar o nosso primeiro comando para o R e falar diretamente com ele. Pediremos que ele some 1 + 1, e para fazer isso iremos digitar 1 + 1 no console (Figura 2 - janela 2) e apertar a tecla "Enter".

```
1+1
```

- *Dica: Utilize este livro realizando os comandos sugeridos diretamente no R Studio. Sinta-se livre para executar outros comandos similares.*

Ao fazer isso, ele vai te retornar o valor 2. Você pode utilizar o R como uma calculadora e realizar as operações básicas normalmente. Agora, vamos pedir para ele subtrair 10 - 5, digitando no console e pressionando Enter. Perceba que não faz diferença se você digitar 10-5, 10 - 5, 10-5, ou até mesmo 10 -5. Mas claro que ao escrever códigos, iremos utilizar a forma que mantém o código mais organizado. Minha sugestão é utilizar 10 - 5.

```
10 - 5
```

Agora vamos pedir para ele multiplicar:

```
5 * 5
```

E dividir:

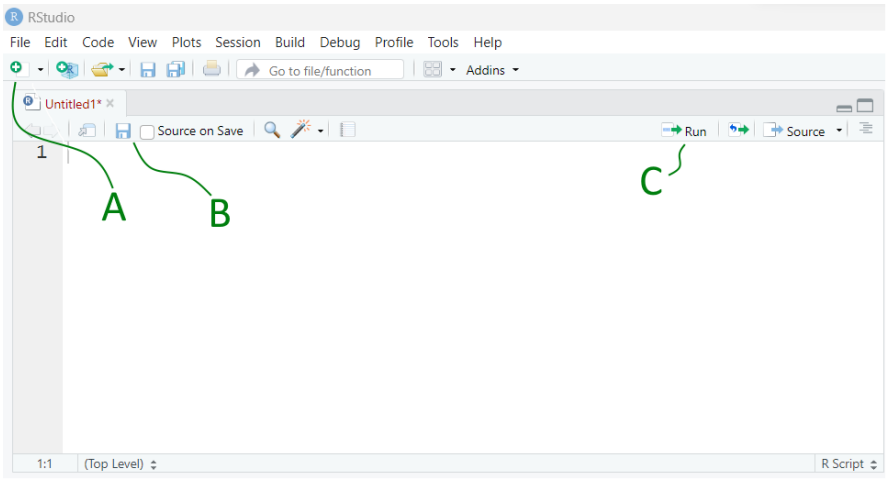
```
10 / 2
```

Você deve ter percebido que cada operação matemática possui seu próprio símbolo. A facilidade de encontrar informações na internet é uma das vantagens do R. Ser gratuito, de código aberto e contar com uma comunidade altamente ativa torna o R singular. Ao longo do livro, você encontrará alguns tópicos nos quais peço que realize pequenas tarefas para compreender o código e desenvolver autonomia.

- *Dica: Pesquise como realizar potenciacao no R e calcule 2 elevado a 2. voce pode utilizar o chatgpt!*

Massa! Mas você deve concordar comigo que essa forma de interagir com o R não é muito eficiente. Apesar de mostrar o histórico, se você quiser realizar a primeira operação que fizemos (1 + 1), precisará digitar novamente sempre que quiser fazer algo. Por esse motivo, a janela de script (Figura 2 - janela 1) se torna tão importante.

Vamos criar um script e utilizá-lo em uma situação prática. Mas antes, precisamos aprender mais alguns botões. Veja a figura abaixo:



Vamos criar um novo Script clicando no **botão "A" > New Script**. No Script, podemos digitar livremente sem que o código seja executado; inclusive, podemos fazer comentários utilizando o símbolo # antes do que foi digitado. Podemos apertar **Enter** sem enviar o código para o R. O código apenas será executado se clicarmos na linha do código e apertarmos o botão **"C"** (Run) ou **Ctrl + Enter**. Copie e cole o código abaixo no seu script e execute, apertando Ctrl + Enter linha por linha. Perceba que o # impede que o nome Script seja executado. Se você executar Script sem a #, o R não vai entender e vai dar erro (voce pode tentar fazer isso para ver o que acontece).

```
#Script

1 + 1
2+ 2
10/2
5+5-2/2

#Fim
```

O comentário também pode ser feito na mesma linha após o código, permitindo utilizar comentários para explicar o que o código faz.

```
2*50 #Esse codigo realiza uma multiplicacao.
```

Por fim, voce pode salvar o Script para acessá-lo sempre que quiser clicando no **botao "B"** e selecionando um local no computador. Voce pode acessar esse script salvo sempre que quiser retornar e continuar um trabalho.

- *Dica: Crie um mini script com diferentes funções matematicas do jeito que voce quiser, incluindo comentarios e explicando como realizar isso no R. Use o chatgpt para criar equacoes mais complexas so para ver o poder do R como uma calculadora e como ferramenta auxiliar.*

- *Outra dica: sempre que você estiver no computador e quiser fazer algum cálculo, por mais simples que seja, não use uma calculadora (do PC ou celular), use o R. Praticar é a chave!*

3.2 Objetos, vetores e funções

3.2.1 Objetos

Uma das funções mais importantes do R é a capacidade de armazenar valores em objetos. Por exemplo, podemos dizer que o número 1 será atribuído à letra "**a**", e para isso utilizamos o sinal de igual (=). O "**a**" agora é considerado um objeto e aparecerá na janela de Environment (Figura 2 - janela 3).

```
# Criando objetos
a = 1
b = 2
```

Entretanto, o sinal de atribuição mais utilizado é a seta (**<-**), devido à sua direcionalidade. Então, vamos adotar esse símbolo de atribuição. Guarde isso na sua cabeça: o código **<-** significa **atribuir**.

```
# Criando Objetos
c <- 3 # c recebe 3
d <- 5 # atribuimos 5 ao d
```

Se voce esta copiando e colocando os codigos do livro no seu script, e apertando **Ctrl + Enter** em cada linha. Voce criou quatro objetos atraves das atribuicoes. Objetos a, b, c, d. Mas voce so recebera o seu respectivo valor se voce digitar o objeto e executa-lo com Ctrl + Enter.

```
a
b
c
d
```

Com esses objetos podemos realizar operacoes matematicas.

```
#Operacoes com objetos

a + b
c * d
b/2
2 * d

#Fim
```

Nesse momento, meu consagrado aprendiz, eu espero que você esteja realizando diferentes operações matemáticas, utilizando diferentes atribuições

Podemos atribuir valores em palavras inteiras. Como por exemplo, atribuir 10 ao objeto chamado "nota" (ou qualquer outra palavra).

```
nota <- 10
Nota <- 0
```

Perceba que o R detecta as diferenças entre letras maiúsculas (caixa alta) e minúsculas (caixa baixa). Portanto, "nota" é diferente de "Nota". Muito cuidado com isso. Conheço um rapaz que sempre aparece desesperado dizendo que o código não está funcionando, e 100% das vezes, é um erro de digitação. Para evitar erros, sugiro sempre utilizar letras minúsculas.

Alguns símbolos não podem ser utilizados para criar objetos, uma vez que são reservados para finalidades internas no R, como por exemplo o '**-**', ou **TRUE** / **FALSE**. Perceba que ao rodar os códigos abaixo, o R irá retornar um erro.

```
FALSE <- 2
TRUE <- 5
guarda-chuva <- 2
```

Também podemos atribuir uma equação inteira a um objeto. Na verdade, podemos atribuir quase tudo a um objeto: gráficos, resultados, planilhas, etc. Mas isso veremos mais adiante.

```
#Note que o underline (_) funciona bem para separar palavras, assim como ponto (.)
guarda_chuva <- 2*10/2
guarda_chuva

guarda.roupa <- 2*100/10
guarda.roupa
```

Alem disso tudo, podemos atribuir um texto a um objeto. Mas para isso o texto precisa estar entre aspas.

```
#Atribuindo a palavra "coxinha" ao objeto melhor_comida
melhor_comida <- "coxinha"
melhor_comida
```

Mas para que serve tudo isso? Por que aprender essas coisas de objetos, atribuições, e sei que lá? A resposta principal é a capacidade de automação e reprodutibilidade das suas análises. Você poderia usar outro programa estatístico? Poderia. Mas confie que o negócio aqui não possui limites.

Vamos criar um exemplo prático para fixar conceitos. Vamos imaginar que queremos calcular quanto gastamos de aluguel por mês e por ano, considerando que pagamos o aluguel semanalmente. Vamos supor que o aluguel por semana seja de 200 reais.

```
#Calculando o aluguel

aluguel_semana <- 200              #valor do aluguel por semana (em reais)
aluguel_mes <- 4 * aluguel_semana  #considerando que 1 mes possui 4 semanas.
aluguel_ano <- 12 * aluguel_mes    #Para o aluguel por ano basta apenas multiplicar o aluguem mes

#Aqui vemos o resultado de cada um deles (Ctrl + Enter neles)
aluguel_semana
aluguel_mes
aluguel_ano
```

Veja que massa! Criamos um script para calcular o valor do aluguel por mês e por ano. Com esse script, você pode calcular facilmente diferentes situações, simulando diferentes valores de aluguel. Para isso, é só alterar o valor 200 de **aluguel_semana** para o valor desejado e rodar linha por linha.

- *Dica: crie o objeto aluguel_dia que te dara a possibilidade de contabilizar qualquer quantidade de dias, bastando apenas multiplicar o objeto (aluguel_dia) pela quantidade de dias desejado.*

Se você pode fazer isso para calcular o aluguel, imagine o que esse sistema pode fazer pela sua análise de dados.

3.2.2 Vetores

Os objetos com múltiplos valores do mesmo tipo são chamados de vetores. Para criar um vetor, inserimos os valores dentro de parênteses separados por vírgulas. Vírgulas no R servem para separar o código. Então, se você quiser digitar números quebrados, nunca utilize **vírgula**, utilize **ponto**.

```
vetor.a <- c(2, 8, 15)
vetor.b <- c(2.5, 2.2, 2.1)
melhores_comidas <- c("coxinha","pastel","lasanha") #Lembre-se, caracteres precisam es

vetor.a
vetor.b
Melhores_comidas
```

Opa, deu um erro, né? Estou marcando o tempo de quanto tempo você demorou para perceber que o nome do objeto é "melhores_comidas" e não "Melhores_comidas" com M maiúsculo. É só pra ficar ligado.

3.2.3 Funções

No R, as funções servem para processar de alguma forma os dados. O R já possui algumas funções nativas, como, por exemplo, para calcular a raiz quadrada de um número. A função possui um nome, geralmente associado ao que ela faz, e argumentos que são o tipo de dado que você precisa inserir nela para ser calculado. Vamos dar uma olhada na função sqrt(). A sigla "sqrt" representa "square root" em inglês. Primeiro, vamos rodar a função sem argumentos, apenas o **sqrt()**, e depois inserimos o argumento, que nesse caso, é um número.

```
#funcao que calcula raiz quadrada

sqrt()

sqrt(25)
```

Note que quando não colocamos (ou erramos) o argumento, o R vai te retornar um erro, geralmente, explicando o problema.

Existem funções dentro do R para todo tipo de coisa! Vamos criar um vetor (conjunto de dados) e fazer alguns cálculos utilizando algumas funções.

```
#Notas do terceiro ano A
#Cada um desses numeros representa uma nota de um aluno
notas <- c(10, 8, 7, 5, 2, 4, 6.6, 7.9, 8.9, 9.5)

#Qual a media da turma?
mean(notas)

#Qual a mediana?
median(notas)
```

```
#Qual o desvio padrao das notas?
sd(notas) #sd vem de standard deviation

#Qual a menor e a maior nota?
min(notas)
max(notas)

#resumo das notas
summary(notas)
```

Cada uma dessas funções realiza algum cálculo específico. No caso de summary(), ela realiza o cálculo da média, mediana, máximo, mínimo e quartis, tudo ao mesmo tempo. Talvez em algum momento você não encontre a função que precisa para determinada tarefa; nesse caso, você pode criar sua própria função! Pode criar uma função utilizando outras funções para fazer algo específico para você. Por enquanto, vamos apenas ser usuários de funções e não criadores. Mas se você realmente quer saber como criar uma função, não irei te limitar. Abaixo, você encontra um dos exemplos mais simples de função. Utilize sua criatividade, seus conhecimentos de objetos, funções e vetores para criar alguma função. Mas sinta-se à vontade para pular essa parte, uma vez que ela não é essencial ao uso do R.

3.2.3.1 Criando uma funcao - parte I (opcional)

Vamos imaginar que voce precise criar uma funcao para somar dois numeros. Uma funcao simples mas que vai nos ajudar a entender a estrutura de uma funcao. Para criar uma funcao utilizamos uma funcao chamada "**function()**". Essa eh uma funcao que possui dois argumentos numericos que chamamos de x e y. A funcao vai pegar esses dois valores e realizar um calculo, que no caso, eh uma soma simples. O resultado da soma sera atribuido a um objeto chamado "resultado", e ele sera mostrado atraves da funcao "print". Agora que criamos a funcao, vamos utiliza-la para somar dois numeros qualquer.

```
#Criando a funcao soma

soma <- function(x, y) {

resultado <- x + y
print(resultado)

}
```

Veja que: o nome "soma", a gente que criou, poderia ter sido "add" ou "adição". Os argumentos da função ficam dentro do parênteses; se existir mais de um, são

separados por vírgulas. Dentro das chaves, vem todo o processamento que a gente quis que a função realizasse, no caso, apenas uma soma, e que esse resultado seja mostrado para a gente.

Agora vamos usar essa função. Uma vez criada, ela fica armazenada no R e podemos utilizá-la. Para usá-la, só precisamos fornecer dois números para que esses sejam somados.

```
#Utilizando a funcao soma

soma(2,2) #pode colocar qualquer numero
```

Agora, vamos criar uma função para calcular a velocidade média. Para isso, a gente precisa dividir a distância pelo tempo.

```
#Criando funcao
vm <- function(distancia, tempo) {
  resultado <- distancia / tempo
  print(resultado)
}

#Utilizando funcao vm (velocidade media)
#a distancia percorrida foi 100km
#o tempo gasto foi 2 horas

vm(100, 2) #resultado deve ser interpretado em km por hora.
```

Você deve estar dizendo "vish que lezeira, já daria pra ter feito direto haha". Sim, daria, mas a ideia é aprender a criar uma função. Mais para frente voce vai ver o potencial disso.

3.3 Tabela de dados

3.3.1 Data frame

DataFrame nada mais é do que uma tabela de dados. Vamos direto ao que interessa e usar a função *data.frame()* para criar uma tabela. Iremos criar as notas de 10 alunos e seus respectivos nomes. Se você ainda estiver na mesma sessão de R, podemos ver que o objeto "notas" ainda está salvo na janela 3 (Environment e History) e podemos aproveitá-lo. De qualquer forma, vou recriar "notas", caso por algum motivo você TENHA SAÍDO do R, um absurdo, mas acontece.

```r
#Criando um data frame contendo as notas e os nomes de cada aluno

#Primeiro criamos as notas (cada numero representa a nota de um aluno)
notas <- c(10, 8, 7, 5, 2, 4, 6.6, 7.9, 8.9, 9.5)

#Segundo criamos os nomes (na ordem das notas)
#Ex: o primeiro valor do objeto nota (10) representara a nota de Marilia que eh o primeiro nome.

nomes <- c("Marilia", "Jonatas", "Guilherme", "Thiago",
"Fifi", "Brunnin", "Diego", "Mazana", "Bruna", "Vitoria")

#Terceiro utilizamos a funcao data.frame() para criar a tabela
tabela <- data.frame(nomes, notas)
tabela
```

Esse data.frame (tabela) é uma forma muito simplificada das tabelas que iremos importar de nossos dados reais. No entanto, ela nos serve muito bem para entender como funciona uma planilha no R.

A partir de agora, quero que você perceba uma coisa muito importante e leve isso para a vida. Uma planilha organizada deve ser construída da seguinte forma:

-Variáveis são colunas; -Observações são linhas; -Cada valor na sua célula.

Com essa formatação de planilha, você consegue fazer quase tudo no R, para manipulação de dados, construção de gráficos e análises estatísticas.

Logo, para você acessar uma variável de uma tabela, usamos o **$** da seguinte forma:

```r
#Acessando uma variavel

tabela$nomes
tabela$notas
```

Perceba que quando você digita **tabela$**, pode apertar a tecla TAB para mostrar e selecionar as variáveis de sua tabela. Caso alguma nota esteja errada e você queira corrigi-la, pode utilizar a função *fix()*. Essa função abre uma janela onde você pode clicar e modificar o valor desejado.

```r
fix(tabela)
```

Quase nunca utilizamos a função *fix()* para corrigir os dados, mas é interessante saber que o R pode abrir novas janelas e criar coisas interativas.

Agora, sabendo como acessar as variáveis de uma tabela, você pode utilizar funções para calcular uma variável específica do data frame.

```
#Calculando a media da variavel notas da tabela
mean(tabela$notas)
```

Utilize as funções abaixo para calcular mediana, máximo, mínimo:

- median()
- max()
- min()
- summary()

3.3.2 Planilha de dados nativa do R

O R possui um banco de dados que nos fornece algumas tabelas de estudo reais, as quais podemos utilizar para treinar nossas habilidades. Para isso, precisamos dizer para o R qual banco de dados queremos invocar utilizando a função *data()*.

```
#Importando o banco de dados "iris" do R
data(iris)

#Funcoes exploratorias
head(iris) #Mostra a parte de cima da planilha
tail(iris) #Mostra a parte de baixo
str(iris)  #Mostra os tipos das variaveis
```

Veja que ao usarmos a funcao *str()* o R retorna a natureza das variaveis da planilha.

- *num* significa numerica.
- *Factor* significa que eh uma variavel qualitativa com fatores (setosa, versicolor, virginica).

Às vezes, a planilha é importada com as variáveis mal formatadas, e isso pode gerar problemas de reconhecimento por parte do R. Nesse caso, é sempre interessante realizar a verificação dos dados utilizando a função *str()* ou outras funções similares. Caso o R não esteja identificando as variáveis corretamente, duas funções muito úteis podem ser utilizadas: *as.numeric()* e *as.factor()*.

```
data(iris)

#Convertendo variavel para numerica
#a variavel sepal lenght da tabela iris recebe ela mesma transformado em numera
iris$Sepal.Length <- as.numeric(iris$Sepal.Length)
```

```
#Convertendo variavel para fator
iris$Species <- as.factor(iris$Species)
```

Basicamente, você está transformando em numérica a variável Sepal length da planilha iris e atribuindo a ela mesma. Dessa forma, você faz a alteração de forma permanente. A mesma coisa ocorre com *as.factor()*.

Vamos brincar um pouco com a planilha iris. Primeiro, veja que ela está construída de forma organizada (variáveis nas colunas, observações nas linhas, cada célula um valor). Podemos então realizar o *summary()* para saber a média, mediana, máximo e mínimo da largura da pétala (Petal.Width).

```
summary(iris$Petal.Width)
```

Porém, note que existem 3 espécies diferentes. A função *summary()* não leva isso em consideração. O ideal seria calcular o *summary()* para cada espécie. Para isso, temos uma função muito legal chamada *tapply()*, onde: o primeiro argumento é a variável numérica, o segundo é a variável com os fatores, e o terceiro argumento é a função que você quer aplicar.

```
#Calculando por especie
tapply(iris$Petal.Width, iris$Species, summary)
```

3.3.3 Importando sua planilha

No RStudio, podemos importar uma planilha no formato Excel (xlsx) utilizando o botão na janela Environment (canto superior direito):

Environment > *Import Dataset* > *From Excel*

Acho pouco eficiente ensinar a importar dados de um diretório. Acho pouco prático, uma vez que temos o botão de importar. Ao invés disso, irei treiná-lo a utilizar projetos de R reprodutíveis e organizados. Ao trabalhar no R, você seguirá um protocolo básico de como tratar os dados a serem analisados. Ao longo do tempo, você será capaz de otimizar esse protocolo e aplicá-lo para quase todo tipo de dado que você possuir. De qualquer forma, saiba que o R trabalha com uma pasta específica chamada de diretório, e para saber qual é o seu diretório padrão, utilize a função *getwd()*:

```
getwd()
```

Ao executar essa função, o R te retornará o caminho da pasta do diretório padrão atual, que provavelmente é a pasta "Documentos" do seu computador.

Para baixar uma planilha de dados fictícios, clique no link abaixo e tente importá-la através do botão **Import from Excel**.

Clique aqui para baixar a planilha ficticia

3.4 Graficos

Agora começa a parte divertida! A visualização de dados no R tem um potencial quase infinito devido à sua grande capacidade de personalizar cada pedacinho da figura.

3.4.1 Grafico de histograma

Esse gráfico nos mostra a frequência dos valores da variável de interesse, como por exemplo, o comprimento da pétala das flores do banco de dados iris.

```
#Importando banco de dados "iris"
data(iris)

#Criando histograma
hist(iris$Sepal.Length)
```

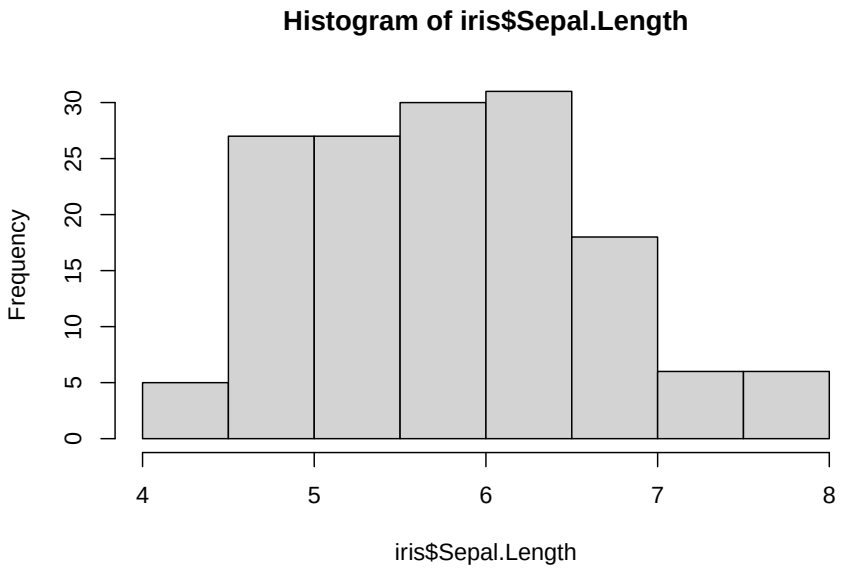


Figure 3.2: Gráficos com R base

Simples, não é? Mas podemos modificar ainda mais nosso gráfico. Alterar cores, nomes dos eixos, título, etc. Vamos ver mais argumentos com o gráfico de dispersão de pontos.

3.4.2 Grafico de dispersao

Esse gráfico nos mostra a relação entre duas variáveis contínuas. Vamos utilizar o banco de dados "iris" para construir e visualizar alguns gráficos. Para construir um gráfico de dispersão, utilizamos a função *plot()*. Note que a função plot possui vários argumentos que podemos ir adicionando para personalizar o gráfico. Além disso, para deixar o código mais organizado, sempre após a vírgula de um argumento, pulamos a linha, de forma que cada argumento fique numa linha.

```
#Cria o grafico basico
plot(iris$Sepal.Width, iris$Petal.Width)
```

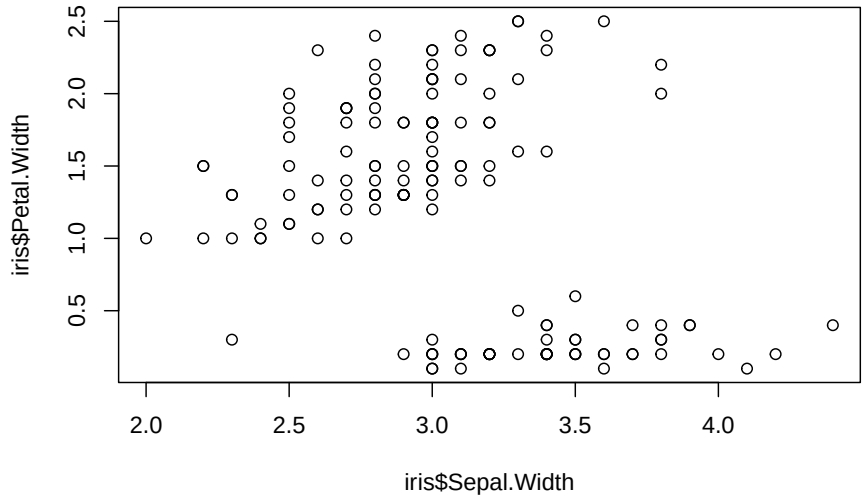


Figure 3.3: Gráficos com R base

```
#adiciona o argumento para mudar o nome do eixo y
plot(iris$Sepal.Width, iris$Petal.Width,
     ylab = "Petal Width")
```

```
#adiciona o argumento para mudar o nome do eixo x
plot(iris$Sepal.Width, iris$Petal.Width,
     ylab = "Petal Width",
```

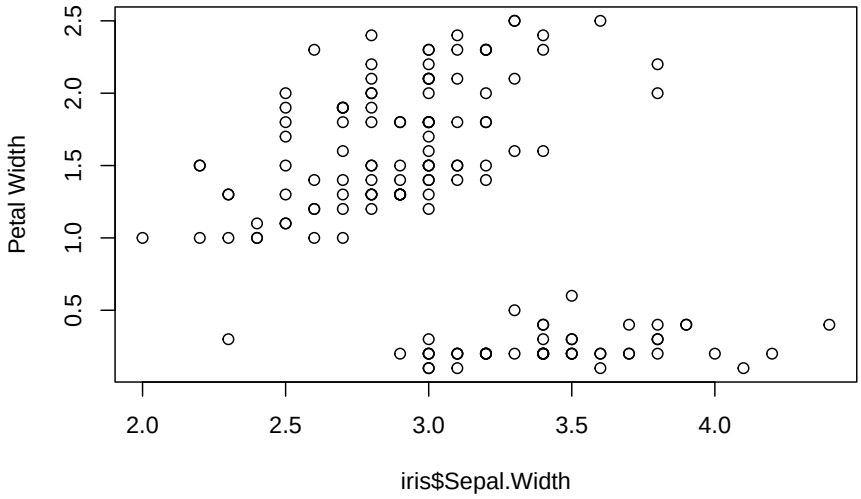


Figure 3.4: Gráficos com R base

```
     xlab = "Sepal Width"
     )
```

```
#argumento para mudar a cor dos pontos
plot(iris$Sepal.Width, iris$Petal.Width,
     ylab = "Petal Width",
     xlab = "Sepal Width",
     col = "blue"
     )
```

```
#argumento para mudar o formato dos pontos
plot(iris$Sepal.Width, iris$Petal.Width,
     ylab = "Petal Width",
     xlab = "Sepal Width",
     col = 3,
     pch = 2
     )
```

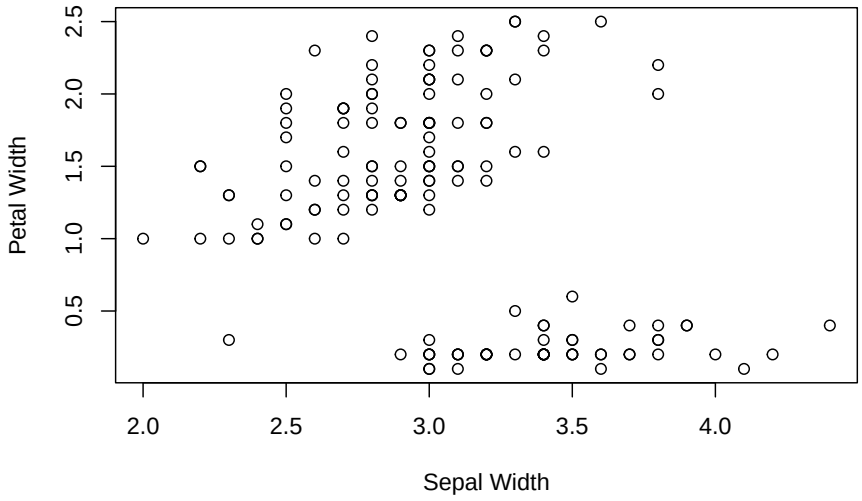


Figure 3.5: Gráficos com R base

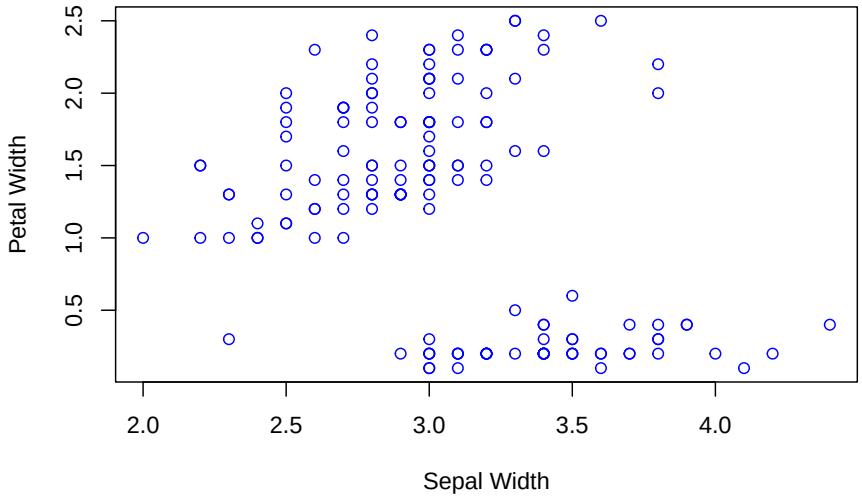


Figure 3.6: Gráficos com R base

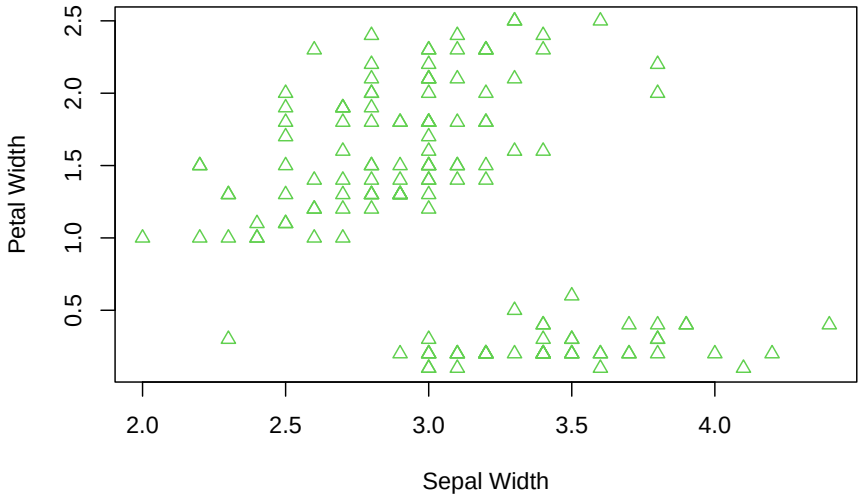


Figure 3.7: Gráficos com R base

```
#argumento para adicionar um titulo no grafico
plot(iris$Sepal.Width, iris$Petal.Width,
     ylab = "Petal Width",
     xlab = "Sepal Width",
     col = 3,
     pch = 2,
     main = "Titulo do grafico"
     )
```

3.4.3 Grafico boxplot

Para criar o gráfico de boxplot, iremos utilizar a variável numérica Sepal.Width do banco de dados iris em relação à variável Species. Note que na função plot, as duas variáveis (argumentos) eram separadas por vírgula; aqui utilizamos o ~ para relacionar a variável numérica com a categórica.

```
#Importa o banco de dados iris
data(iris)
```

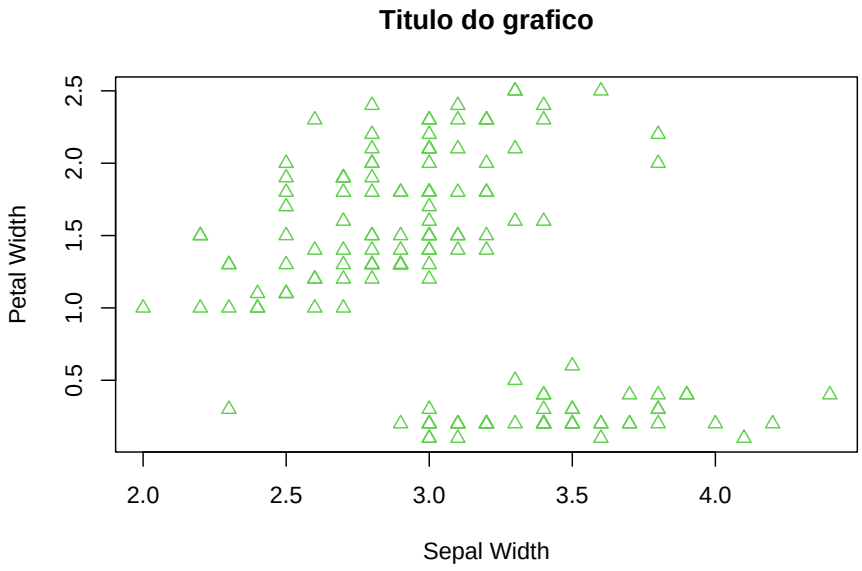


Figure 3.8: Gráficos com R base

```
#Cria o grafico de boxplot basico
boxplot(iris$Sepal.Width ~ iris$Species)
```

```
#Boxplot personalizado
boxplot(iris$Sepal.Width ~ iris$Species,
      ylab = "Petal Width",
      xlab = "Species",
      col = c(3,4,"tomato"),
      main = "Titulo do grafico"
      )
```

Perceba que o argumento **col** pode ser um vetor e pode receber mais de um valor. Nesse caso, demos uma cor para cada fator de Species. Tente substituir os números por outros ou colocar o nome das principais cores como: red, green, blue, yellow... Só não esqueça de colocar entre aspas.

3.4.4 Grafico de barras

Para criar um gráfico de barras de contagem, vamos criar nosso próprio data frame. Imagine que queremos tabular a contagem total de ectoparasitas (piolho)

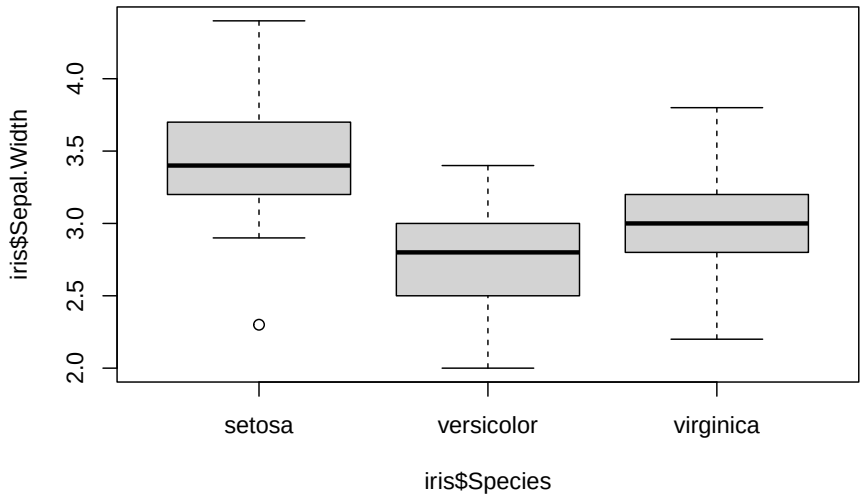


Figure 3.9: (#fig:3_basic_boxplot-1)Boxplot com R base utilizando o banco de dados iris

de aves em cada parte corporal. Como é algo simples, podemos criar diretamente no R.

```
#Criando quantidade total de parasitas
qt_parasita <- c(10, 15, 29, 4)

#Criando parte corporal da ave
parte_corporal <- c("cabeca", "asa", "barriga", "cauda")

#Criando tabela
dados_parasita <- data.frame(qt_parasita, parte_corporal)

# Cria ndo gráfico de barras
grafico <- barplot(dados_parasita$qt_parasita,
                   names.arg = dados_parasita$parte_corporal,
                   xlab = "Parte do Corpo",
                   ylab = "Quantidade de Parasitas",
                   ylim = c(0,35), #define o limite do eixo y (de 0 a 35)
                   col = "lightgreen",
                   border = "black")
```

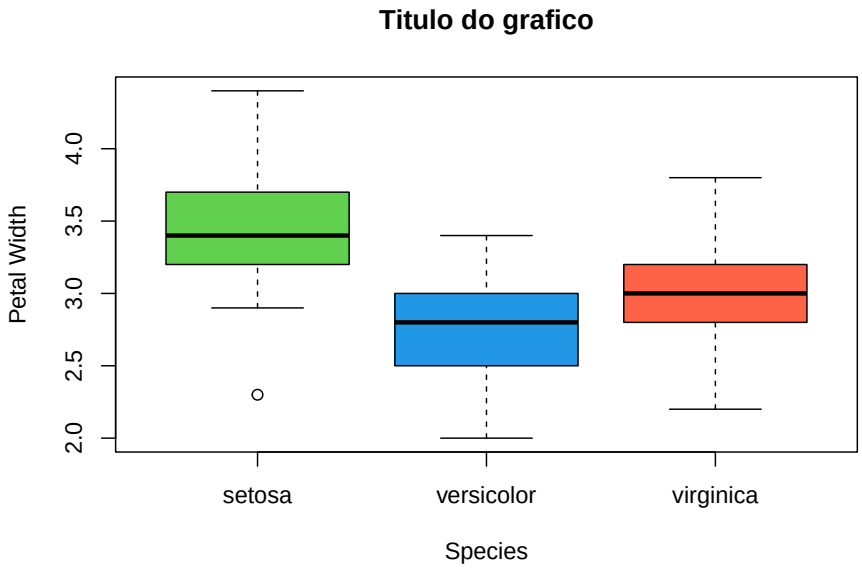
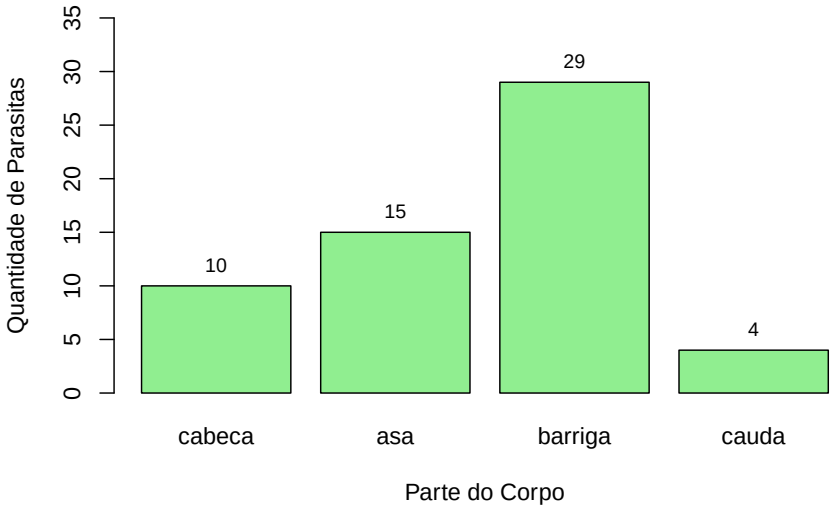


Figure 3.10: (#fig:3_basic_boxplot-2)Boxplot com R base utilizando o banco de dados iris

```
# Adicionando os números acima das barras
text(grafico, dados_parasita$qt_parasita,
labels = dados_parasita$qt_parasita, pos = 3, cex = 0.8)
```



Faça as seguintes modificações no código e veja o que acontece:

- Veja o que acontece se você deletar a linha completa do argumento names.arg.
- Troque o valor 35 de ylim por 50.
- No argumento border, troque "black" por "red" e depois "white".
- Na função text(), troque pos = 3 por pos = 1. (rode o gráfico antes para não sobrepor os números)

É possível fazer muitos outros tipos de gráficos, utilizando diferentes funções. Mas a ideia desse capítulo é ensinar a utilizar o R, e não a criar gráficos. Então vamos com calma, pois quando você pensar que não, já foi.

3.5 Pacotes

Agora já podemos começar a expandir nosso universo do R. Tudo que fizemos até agora foi utilizando o próprio R base. A partir de agora, iremos incluir um conjunto de novas funções através dos pacotes de R. Antes de tudo, precisamos instalar o pacote que queremos utilizar, e para isso existem duas formas:

- Utilizando o botão *Tools* do RStudio.
- Utilizando a função *install.packages("nome_do_pacote")*

Vamos instalar o pacote "lattice" para criarmos uns gráficos diferentes do que fizemos anteriormente.

Utilizando o botão *Tools* do RStudio clique em: **Tools > Install package > digite o nome "lattice".** Ao começar a digitar, o R irá sugerir opções de pacotes com as iniciais que você digitou. Depois, clique em "instalar".

Utilizando a função *install.packages()*

```
install.packages("lattice")
```

O processo de instalação mostra no console vários processos, e enquanto isso acontece, um ícone de "stop" vermelho aparece na parte superior direita da janela do console. O processo só termina quando esse ícone some e um aviso aparece:

```
> install.packages("lattice")
Installing package into 'C:/Users/Diogo Silva/AppData/Local/R/win-library/4.3'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.3/lattice_0.22-5.zip'
Content type 'application/zip' length 1380135 bytes (1.3 MB)
downloaded 1.3 MB

package 'lattice' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Diogo Silva\AppData\Local\Temp\RtmpcdHtSO\downloaded_packages
>
```

Pacote instalado, agora é só dizer que queremos utilizar o pacote. O processo de instalação em um determinado computador só precisa ser feito uma vez. Porém, toda vez, você precisa dizer para o R qual pacote você está utilizando. Para isso, utilizamos a função *library()*.

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

Se você carregou o pacote e o R não te retornou nenhum erro, está tudo certo. Às vezes, ele te retorna um Warning message, também está tudo certo. Vamos adiante.

O pacote lattice que acabamos de instalar nos fornece várias funções para criação de gráficos. Tenha em mente que existem milhares de pacotes, cada um trazendo um conjunto de funções e bancos de dados de diferentes contextos, tanto para criar gráficos quanto para realizar modelos estatísticos avançados.

Vamos criar alguns gráficos utilizando as funções que o pacote lattice nos forneceu.

```
#Se o pacote já estiver instalado, você só precisa carregar o pacote
library(lattice)
```

```
## Warning: package 'lattice' was built under R version 4.3.2
```

```
# Gráfico de dispersão condicionado por uma variável
xyplot(Sepal.Length ~ Sepal.Width | Species, data = iris,
       main = "Scatterplot Condicionado por Espécie",
       xlab = "Largura da Sépala",
       ylab = "Comprimento da Sépala")
```

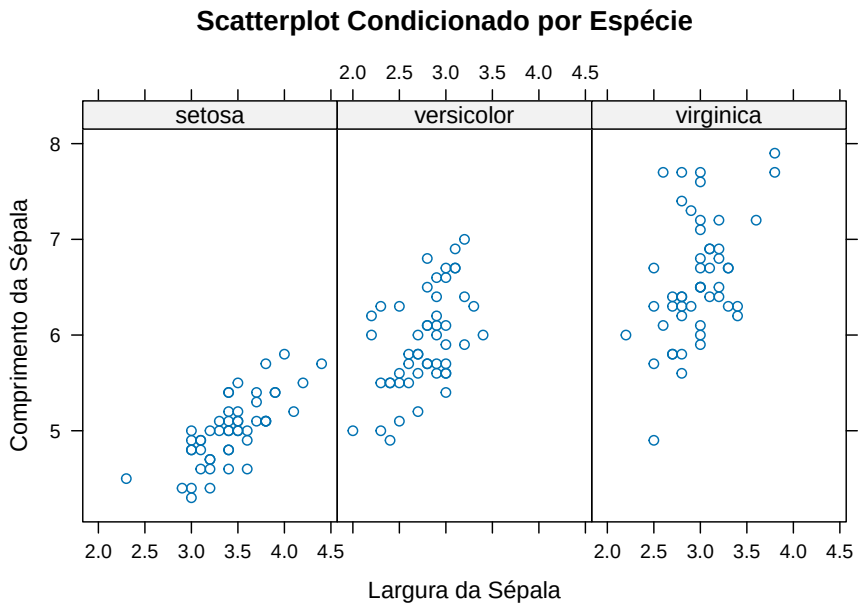


Figure 3.11: Gráficos com Lattice

```
# Histograma condicionado por uma variável
histogram(~ Petal.Length | Species, data = iris,
          main = "Histograma Condicionado por Espécie",
          xlab = "Comprimento da Pétala")
```

Acho que já deu para entender como o R funciona, né? Nos próximos capítulos, você vai aprender como trabalhar de forma eficiente no R, seguindo um fluxo de trabalho eficiente e, o melhor de tudo, reproduzível.

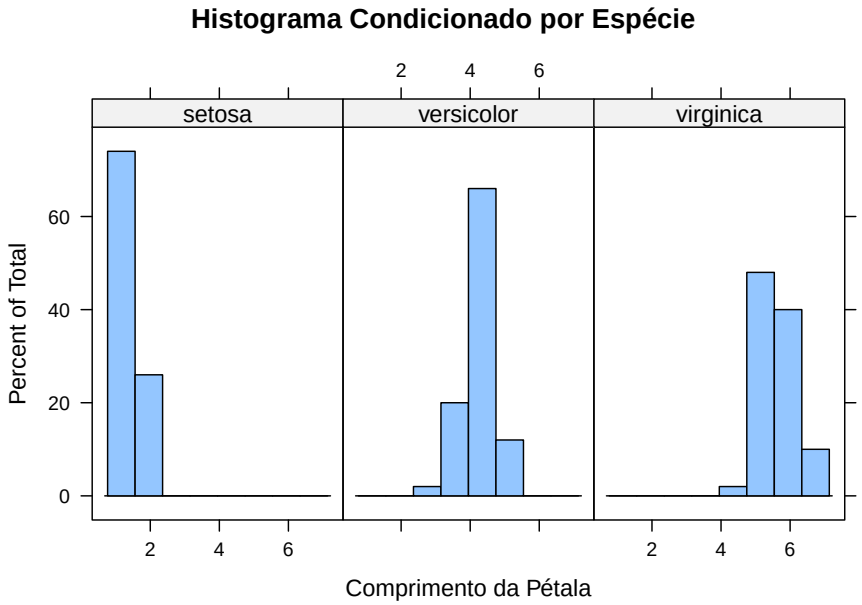


Figure 3.12: Gráficos com Lattice

Chapter 4

Como trabalhar no R

Se você já tem uma noção de R, mas ainda sente dificuldade em organizar seu fluxo de trabalho, este capítulo é para você! Utilizaremos técnicas de ciência de dados e repositórios como o GitHub para tornar o trabalho no R prazeroso e eficiente!

Iremos utilizar o banco de dados ficticios que possui erros comuns que podemos utilizar para aprender a lidar com eles sem precisar de utilizar o Excel.

Baixe a planilha fake e veja que ela possui quatro variaveis (ID, peso, numero de parasitas e local) coletadas do passarinho bem-te-vi.

Clique aqui para baixar a planilha ficticia

Vamos importar a planilha fictícia mais para frente. Relaxe aí e finja que ela é sua.

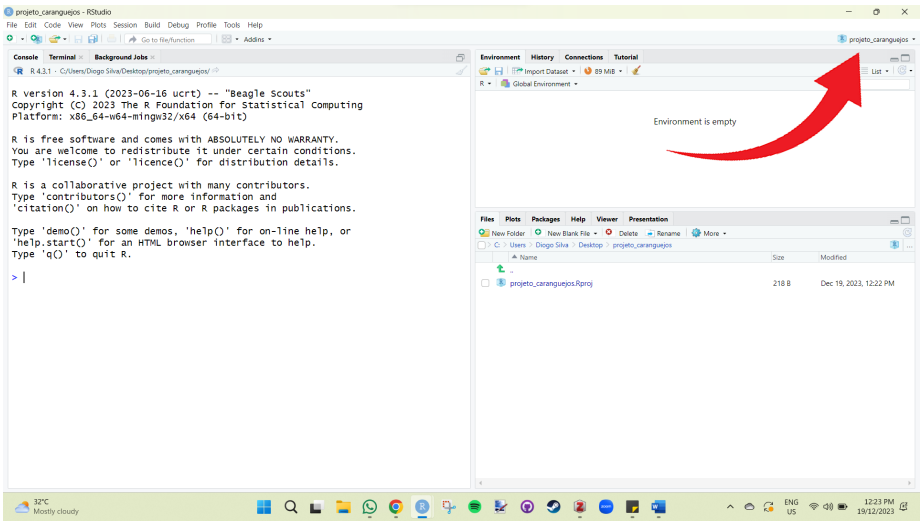
4.1 Criando um projeto de R

Imaginemos que precisamos analisar um banco de dados de algum projeto. O primeiro passo eh criar um projeto de R, para isso voce vai no canto superior direito do RStudio (proximo a janela de environment e history) e clica:

project (none)" > new project > new directory > new project

Ao clicar em "New Project", aparecerá uma janela para escolher o nome do projeto e o local no computador onde seu projeto irá ficar. Coloque o nome desejado (sugestão: "projeto_caranguejo"), selecione qualquer pasta no computador (sugestão: desktop) e clique em "Criar Projeto".

Se você fez tudo direitinho, o RStudio estará da seguinte forma. Note que ao invés de "Project (None)", estará o nome do projeto que você criou.



Sugiro, meu nobre consagrado Rzeiro, que você vá no local do computador onde você criou o projeto e veja a pasta, veja o arquivo de R que foi criado. Você também pode fazer isso utilizando a janela de "Files", onde mostrará todas as pastas do seu diretório que, a partir de agora, será a pasta que o seu arquivo de projeto de R está situado. Todos os scripts, arquivos e planilhas que você irá utilizar nas suas análises, ficarão dentro da pasta do projeto. Isso significa que vai ficar tudo solto, bagunçado? Claro que não. Iremos criar pastas organizadas onde hospedarão cada coisa que iremos trabalhar, como por exemplo: dados brutos, dados processados, scripts, outputs, etc. Você pode criar manualmente, mas por que faríamos isso se temos o R para fazer por nós?

4.2 Organizando o projeto de R

Para criar as pastas de forma organizada, voce pode fazer manualmente ou utilizando o pacote "here".

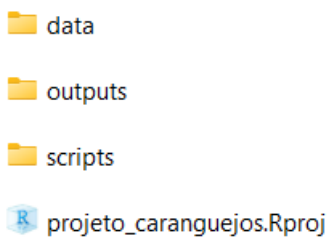
```
install.packages("here")
library(here)
```

Após a instalação do pacote "here", iremos criar uma função que criará as pastas automaticamente no nosso diretório. Não se assuste com o script da função, ela é mais simples do que parece e você não precisa entendê-lo por completo. Apenas rode o código para criar a função e depois rode o código que utiliza a função para criar as pastas.

Antes de rodar o código, certifique-se de que você está no projeto de R que você criou.

```r
# Criando funcao para criacao das pastas do projeto
# Codigo disponibilizado pelo Gustavo Paterno (https://github.com/paternogbc)

build_project <- function(type = "analysis",
                          temp = TRUE) {

  if(type == "analysis"){
    # Data
    dir.create(path = here::here("data"))
    dir.create(path = here::here("data", "raw"))
    dir.create(path = here::here("data", "processed"))

    # outputs
    dir.create(path = here::here("outputs"))
    dir.create(path = here::here("outputs", "figures"))
    dir.create(path = here::here("outputs", "tables"))
    if(isTRUE(temp)){
      dir.create(path = here::here("outputs", "temp"))
    }

    # scripts
    dir.create(path = here::here("scripts"))

    # docs
    #dir.create(path = here::here("docs")) #para criar a pasta docs, so tirar o comentario dessa
  }
}

#Utilizando a funcao criada para gerar as pastas
build_project(type = "analysis",
              temp = TRUE) #se FALSE, nao cria a pasta temp.
```

Se tudo ocorreu bem, as pastas estao assim:



Dentro da pasta “data”, você encontra as subpastas “raw” e “processed”. Em “outputs”, você encontra as subpastas “figures”, “temp” e “tables”. Em “script”,

você não encontrará nada (por enquanto). O arquivo com o símbolo do R "projeto_caranguejos.Rproj" é o seu projeto de R. Você pode abri-lo (dando duplo clique) toda vez que você for trabalhar no projeto. Isso abrirá o RStudio já com o seu projeto aberto e pronto para trabalhar.

- Feche o RStudio e abra-o novamente dando clique duplo no seu projeto de R.

Agora você tem tudo pronto para começar a trabalhar com um fluxo de trabalho eficiente e reprodutível!

4.3 Trabalhando em um projeto de R

Antes de tudo baixe a planilha de dados ficticios (caso nao tenha baixado ainda) de uma especie de passarinho (tambem ficticia), que contem informacoes ficticias sobre peso (g) e quantidade de parasitas de tres diferentes localidades do bem-te-vi:

Clique aqui para baixar a planilha ficticia

Na pasta **data > raw**, você adiciona sua planilha de dados brutos fictícia (fake_data.xlsx). O fluxo de trabalho será o mais simples possível, mas envolverá etapas essenciais da análise de dados.

Limpar dados brutos > Realizar análise > Mostrar gráficos

Para limpar a planilha de dados brutos, iremos criar um script para isso. Utilizaremos funções do pacote *dplyr* para modificar nomes das variáveis, nomes dos fatores, analisar dados faltantes, entre outras coisas.

4.3.1 Etapa 1: limpar dados

Vamos criar um script para limpar nossos dados brutos. Monte o script completo utilizando os codigos que irei fornecendo a seguir copiando e colando no seu RStudio. Assim, a primeira coisa que iremos fazer do script eh seu cabecalho. Utilizaremos os comentarios para criar um cabecalho com as informacoes de titulo, autores e data.

```
# Script to load and clean raw data flores iris
# Author: Marilia T. Ericcson
# Data: 2019-11-06
```

Após isso, utilizaremos os comentários para organizar o script em tópicos, fazendo isso manualmente ou utilizando o atalho: **Ctrl + Shift + R**. A

primeira parte do código é a instalação dos pacotes que iremos utilizar no processo.

```
# Packages ------------------------------------------------------------
# Caso nao tenha os pacotes instalados, instale!

library(dplyr)  #Pacote com funcoes para manipular dados
library(readxl) #Pacote com funcao para importar planilha em formato de excel (xlsx)
```

Segundo, importamos o banco de dados a ser processado (limpado) utilizando a função *read.csv()* e atribuindo (<-) a um objeto que iremos chamar de "dados". Para importar, você só precisa especificar o caminho do diretório em que o arquivo está inserido. Como estamos utilizando o projeto de R, o diretório é onde o projeto está. Dessa forma, não há necessidade de dizer para o R qual é o seu diretório de trabalho. Logo, fica fácil importar os dados brutos pois sabemos que ele está na pasta "dados > raw > iris.csv". Dizemos isso para o R utilizando os nomes das pastas separados por barras e, por fim, o nome do arquivo (iris) e sua extensão (.csv).

```
# Load data -----------------------------------------------------------
dados <- read_excel("data/raw/fake_data.xlsx")
```

Agora que o banco de dados foi importado, podemos fazer um check up basico utilizando algumas funcoes. Como assim check up basico? Ver o numero de observacoes, se tem dado faltando ou observacao duplicada, essas coisas.

```
# 1. Check up basico --------------------------------------------------------
nrow(dados)              # Quantas linhas (observacoes) tem no banco de dados?
str(dados)               # quais as classes das variaveis?
attributes(dados)        # quais atributos?
head(dados)              # Primeiras linhas do banco de dados.
any(duplicated(dados))   # Checar se tem linhas duplicadas.
any(is.na(dados))        # Checar se tem dado faltando (NA).
```

Às vezes, os nomes das variáveis não são fáceis de trabalhar, possuindo espaços, acentos, letras maiúsculas e minúsculas, e nomes confusos. Além de complicar o código, o nome de algumas variáveis pode bugá-lo (caso das acentuações de palavras em português). Então, vamos mudar o nome das variáveis utilizando a função *rename()* do pacote **dplyr**. Iremos deixar os nomes mais fáceis de trabalhar, deixando todas as letras minúsculas.

```
# 3. Renomeando o nome das variaveis ----------------------------------------------
#ver nome das colunas (variaveis) ----
colnames(dados)

#Renomeando variaveis ----
#Vamos renomear de forma a tirar espacos e manter apenas letras minusculas.
dados_renamed <- rename(dados,
                        id = ID,  #nomes das variaveis (nome novo = nome antigo)
                        peso_g  = `Peso (g)`,
                        total_parasitas = `Total de parasitas`,
                        local  = Local
                        )
colnames(dados_renamed)
```

Note que quando existe espaço, o nome da variável precisa estar entre crases. Agora vamos dizer para o R os tipos de nossas variáveis. Um problema muito comum é o R não reconhecer que a variável categórica não possui fatores, mesmo possuindo. Por exemplo, em "iris", temos a variável categórica 'Species' com seus fatores (setosa, versicolor, virginica). É possível que no seu banco de dados, o R não reconheça isso, então vamos utilizar a função *levels()* para verificar se o R entendeu quais são os fatores. Se o R retornar NULL, você precisa dizer para o R utilizando a função *as.factor()*.

```
# 4. Consertar fatores  -----------------------------------------------------------
# ver quais fatores estao na variavel local
levels(dados_renamed$local) #Se retornar NULL eh porque nao ta reconhecendo como fator

# Transformando em fator ----
dados_renamed$local <- as.factor(dados_renamed$local)

# checar fatores
levels(dados_renamed$local)
```

Agora podemos renomear o nome dos fatores de uma variável categórica utilizando a função *recode_factor()*. Nesse processo, podemos identificar erros de digitação que ocorreram durante o planilhamento dos dados. Por exemplo, você pode ter digitado "Macho", "macho" e "male" na variável sexo durante o planilhamento. O R vai identificar os três como sendo fatores diferentes, mas na verdade são o mesmo fator. Portanto, é necessário que seja padronizado.

```
# 4.1 Renomear fatores

#Checar fatores
levels(dados_renamed$local)
```

Veja que aqui deveríamos ter apenas três fatores: floresta, urbano e parque. Mas, devido a erros de digitação, o R entende que são fatores diferentes. Isso é muito comum em planilhas de dados grandes, e você teria que fazer isso manualmente, identificando cada erro. Imagina o trabalhão.

```
#Renomear fatores
dados_renamed$local <- recode_factor(dados_renamed$local,   #variavel categorica
                                     Floresta = "floresta",         #mudando nome Floresta para f
                                     park = "parque",
                                     parque. = "parque",
                                     urban = "urbano",
                                     Urbano = "urbano")

#check factors
levels(dados_renamed$local)
```

Agora sim! Temos os três fatores bonitinhos. Agora que limpamos nossos dados brutos, iremos exportar nossa planilha limpa. Perceba que o processo do script é colocar a planilha bruta de um lado e sair limpa e bonitinha do outro. É com os dados processados que iremos trabalhar de fato. Vamos salvar sempre no formato CSV, por ser um formato mais simples, leve e estável.

```
# 8. Salvar dado processado ------------------------------------------------
write.csv(x = dados_renamed,    #nome da planilha que voce quer exportar
          file = "data/processed/processed_fake_data.csv", #local e nome da planilha exportada
          row.names = FALSE)  #sempre utilizaremos row.names = FALSE
```

O processo de manipulação de dados no R é bastante completo, e existem diferentes formas de limpar seus dados brutos. O pacote **dplyr** possui funções capazes de selecionar variáveis, selecionar linhas, criar variáveis, criar subconjuntos, entre outras. Para aprimorar seu script de limpeza de dados, você precisará aprender sobre manipulação de dados e incluir no processo os códigos no seu script. Por sorte, existe muito material disponível na internet sobre o assunto. Talvez algum dia eu crie um material sobre manipulação de dados com dplyr, mas aqui esse não é meu objetivo. Meu objetivo é simplesmente fornecer o esqueleto teórico, dando base para crescer de forma mais direcionada.

Por fim, não esqueça de salvar o script na pasta script com o nome '01_clean_raw_data'.

4.3.2 Etapa 2: fazer analise

Com a planilha bruta processada, podemos realizar testes estatísticos em um novo script para responder à nossa hipótese. Vamos supor que nossa pergunta

seja saber se existe diferença na infestação de parasitas em cada local. Organizaremos o script de análise de forma semelhante ao script de limpeza, ou seja, escreveremos um cabeçalho, carregaremos pacotes, importaremos e analisaremos dados de forma organizada.

Vamos fazer um cabeçalho para nosso script de análise.

```
# Script para analisar dados
# Author: seu nome
# Data: 2019-11-07
```

Vamos criar o código para carregar os pacotes que iremos utilizar.

```
#Carregar pacotes
library(broom)
```

Dessa vez, iremos importar a planilha limpa e processada. Note que o caminho de importação é similar ao caminho que exportamos no script de limpeza.

```
# Load data ------------------------------------------------------------------
dados <- read.csv("data/processed/processed_iris.csv")
```

Vamos fazer um check basico.

```
# 1. Check up basico ---------------------------------------------------------
head(dados)
str(dados)
leves(dados$local)
```

Vamos realizar uma análise para ver se existe diferença na quantidade de parasitas encontrada nos bem-te-vis de cada local. Para isso, iremos utilizar o teste não paramétrico de Kruskal-Wallis através da função kruskal.test(). Nessa função, você precisa dizer qual sua variável numérica de interesse em relação a (~) sua variável categórica e seus grupos (fatores). Veja que o p-value é menor que 0.05, então dizemos que existe diferença da largura entre as três espécies.

```
#realizando o teste
kw <- kruskal.test(dados$total_parasitas ~ dados$local)
kw
```

Vamos salvar o resultado em uma tabela organizada (tidy) utilizando a função *tidy()* do pacote **broom**.

```
#Convertendo tabela para formato tidy
table_kw <- tidy(kw_test)
table_kw
```

Agora salvaremos essa tabela na nossa pasta outputs em tables.

```
#Exportando resultado
write.csv(x = table_kw,
          file = "outputs/tables/resultado_kw.csv",
          row.names = FALSE)
```

Maravilha! Já temos nosso segundo script pronto. Agora é só salvá-lo na pasta script com o nome '02_kruskal_test'.

4.3.3 Etapa 3: criar grafico

Agora vamos criar o gráfico para representar nosso resultado de que existe diferença na largura das pétalas entre as espécies de iris.

```
# Script para criar graficos
# Author: seu nome
# Data: 2019-11-07

#Carregar pacotes
library(ggplot2)

# Load data ---------------------------------------------------------------
dados <- read.csv("data/processed/processed_fake_data.csv")

# 1. Basic checks ---------------------------------------------------------
head(dados)
str(dados)
leves(dados$species)

# 2. Criar grafico boxplot ------------------------------------------------

grafico <- ggplot(dados, aes(x = species, y = petal_width))+
  geom_boxplot()

grafico

# 3. Salvando grafico -----------------------------------------------------
```

```
ggsave(plot = grafico,    #nome do grafico
       filename = "output/figures/Grafico_boxplot.png", #nome do grafico exportado
       width = 10,                                      #largura do grafico
       height = 10,                                     #Altura do grafico
       dpi = 300)                                       #resolucao do grafico
```

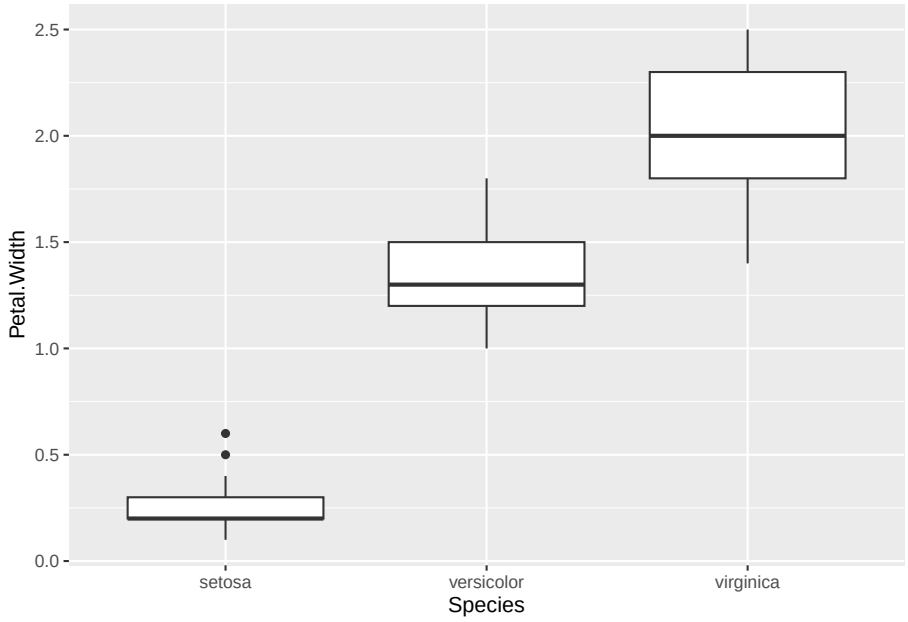


Figure 4.1: ggplot2

Você acabou de criar um gráfico utilizando o pacote ggplot2, que é o pacote mais utilizado para produção de gráficos no R. O código pode parecer um pouco confuso no início, mas não precisa estudar muito para perceber que ele é muito simples e intuitivo. Para aprender mais sobre gráficos, sugiro os livros: Data Visualization: A practical introduction e R Graphics Cookbook.

Não esqueça de salvar o script com o nome '03_grafico'.

4.3.4 Etapa 4: Replicabilidade

O seu projeto está organizado de forma que se torna fácil a replicabilidade em qualquer computador, gerando todos seus outputs (tabelas e figuras) automaticamente! Para isso, você precisa manter e **JAMAIS deletar** ou modificar seus dados brutos (pasta raw). Todos os demais arquivos podem ser produzidos com um clique a partir dos seus dados brutos, portanto, podem ser deletados. Isso

se torna útil ao compartilhar seus dados com alguém, em que você pode zipar todo seu projeto, enviar para um contribuidor, e ele poderá ter acesso a tudo rodando apenas um script. É esse script que iremos fazer agora.

```
# Script to run all project code
# Importante: reinicie o R antes de rodar esse script session > restart R

# Start----
# 1. Run all scripts again on a fresh R section--------------------------------
# 1.1 Load and clean data----
source(file = "script/01_clean_raw_data.R")

# 1.2 Run analysis ------------------------------------------------------------
source(file = "script/02_kruskal_test.R")

# 1.3 Plot figures------------------------------------------------------------
source(file = "script/03_grafico.R")
```

Ao rodar este código, todos os seus scripts são executados, e todos os seus outputs e planilhas processadas são gerados. Mas lembre-se, seu dado bruto contido na pasta data/raw não pode ser deletado ou modificado.

Salve este script com o nome “run_project”.

Chapter 5

Git e GitHub

Chapter 6

Criando funcoes avancadas

Chapter 7

Criando pacotes