R Zero - do simples ao complexo

Diogo J. A. Silva

2023-12-23

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 facilmente :)

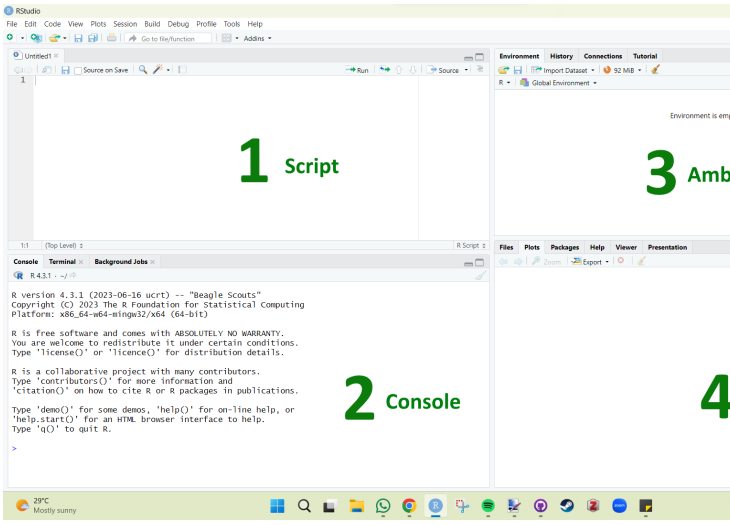
A tendência é que quanto mais você utiliza o R, mais fácil sua vida se torna, e apesar da curva de aprendizagem ser um pouco desestimuladora, é 1000 vezes recompensadora. Como alguns gostam de dizer, você conseguirá fazer um gráfico até 30 vezes mais rápido!!! Não sei de onde veio esse cálculo, mas se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

3.1 Entendendo o RStudio

Tudo que iremos fazer será através do R Studio devido à organização que ele nos fornece. O RStudio eh uma roupagem para o R, que nos fornece um ambiente de trabalho mais amigavel e organizado. Se voce seguiu as instrucoes de instalacao corretamente, voce tera dois icones, o do R base e o do RStudio. Vamos abrir o programa RStudio dando dois cliques sobre seu icone (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. Nao, pera, isso ja existe! Se chama inteligencia artificial! Entretanto, nao se deixe enganar. Os conhecimentos intimos de como pensar, organizar, analisar e mostrar seus dados sao importantissimos para todo um projeto cientifico. Mas se isso nao te convence, as melhores inteligencias artificiais do mercado sao pagas e podem te retornar resultados enganosos. Entao tenha muito cuidado e utilize as inteligencias artificiais disponiveis apenas como uma ferramenta auxiliar!

Vamos ao que importa e entender como o R funciona. O R possui uma lin-

guagem, e tudo que voce vai fazer no R eh utilizando essa linguagem que sao verdadeiros comandos. Importe essa planilha! Crie esse grafico! Faca a media dessa variavel! E por ai vai. Entao 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
```

Ao fazer isso, ele vai te retornar o valor 2. Voce pode utilizar o R como uma calculadora e e realizar as operacoes basicas normalmente. Agora vamos pedir para ele diminuir 10 - 5, digitando no console e dando enter. Perceba que nao faz diferenca se voce digitar 10-5, 10 - 5, 10-5, ou ate mesmo 10 -5. Mas claro que ao escrever codigos iremos utilizar a forma que mantem o codigo mais organizado. Minha sugestao eh utilizar 10 - 5.

```
10 - 5
```

Agora vamos pedir para ele multiplicar:

```
5 * 5
```

E dividir:

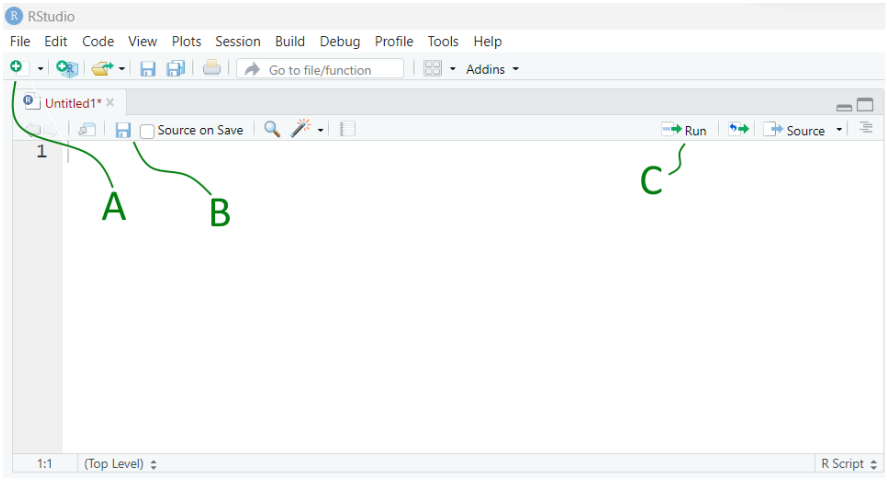
```
10 / 2
```

Voce pode encontrar como realizar potenciacao, raiz quadrada, entre outras coisas pesquisando facilmente na internet. Essa eh uma das vantagens do R. Ser gratuito, possuir codigo aberto, e uma comunidade altamente ativa, torna o R unico. Ao longo do livro voce encontrara alguns topicos onde eu peco para voce realizar pequenas tarefas para voce compreender o codigo e desenvolver autonomia.

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

Massa! Mas voce deve concordar comigo que essa forma de conversar com o R nao eh muito eficiente. Apesar de mostrar o historico, se voce quiser realizar a primeira operacao que fizemos (1 + 1), voce precisara digitar de novo, e de novo sempre que quiser fazer algo. Por esse motivo, a janela de script (Figura 2 - janela 1) se torna tao importante.

Vamos criar um script e utiliza-lo em uma situacao pratica. Mas antes precisamos aprender mais alguns botoes. Veja a figura abaixo:



Vamos criar um novo Script clicando no **botao "A" > New Script**. No Script, podemos digitar livremente sem que o codigo seja executado, inclusive podemos fazer comentarios utilizando o simbolo # antes do que foi digitado. Podemos dar enter sem enviar o codigo para o R. O codigo apenas sera executado se clicarmos na linha do codigo e apertarmos o botao "C" (Run) ou **Ctrl + Enter**. Copie e cole o codigo abaixo no seu script e execute apertando Ctrl + Enter linha por linha. Perceba que a # impede do nome Script ser executado. Se voce executar Script sem a # o R nao vai entender e vai dar erro.

```
#Script

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

#Fim
```

O comentario tambem pode ser feito na mesma linha depois do codigo, e assim, utilizar comentarios para explicar o que o codigo faz.

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

Por fim, voce pode salvar o Script para acessa-lo sempre que quiser clicando no **botao "B"** e selecionando um local no computador.

3.2 Objetos, vetores e funções

3.2.1 Objetos

Uma das funcoes mais importantes do R eh a capacidade de armazenar valores em objetos. Por exemplo, podemos dizer que o numero 1 sera atribuido a letra **"a"**, e para isso utilizamos o sinal de igual (=). O "**a**" agora eh considerado um objeto e ira aparecer na janela de Environment (Figura 2 - janela 3).

```
# Mini Script
a = 1
b = 2
```

Entretanto, o sinal de atribuicao mais utilizado eh a setinha (<-), devido a sua direcionalidade. Entao vamos adotar esse simbolo de atribuicao. Entao guarde isso na sua cabecinha: o codigo **<-** significa **atribuir**.

```
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, nobre leitor, eu espero que voce esteja realizando diferentes operacoes matematicas, utilizando diferentes atribuicoes.

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 diferencas entre letras maiusculas e minusculas. Portanto, “nota” eh diferente de “Nota”. Muito cuidado com isso. Para evitar erros, sugiro sempre utilizar letras minusculas.

Alguns simbolos nao podem ser utilizados para criar objetos, uma vez que eles sao reservados no R. Como por exemplo o “-”, ou TRUE / FALSE. Perceba que ao rodar os codigos abaixo, o R ira retornar um erro.

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

Tambem podemos atribuir uma equacao inteira a um objeto. Na verdade podemos atribuir quase tudo a um objeto, graficos, resultados, planilhas, etc. Mas isso veremos mais a frente.

```
#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 o que serve tudo isso? Para que aprender essas coisas? A resposta principal eh a capacidade de automatizacao e reprodutibilidade das suas analises.

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

```
#Calculando o aluguel

aluguel_semana <- 200              #valor do aluguel por semana
aluguel_mes <- 4 * aluguel_semana  #considerando que 1 mes possui 4 semanas.
```

```
aluguel_ano <- 12 * aluguel_mes    #o ano possui 12 meses eh serio

aluguel_semana
aluguel_mes
aluguel_ano
```

Veja que massa! Criamos um script para calcular o valor do aluguel por mes e ano. Com esse script, voce pode calcular facilmente diferentes situacoes, simulando diferentes valores de aluguel. Para isso, eh so alterar o valor 200 de aluguel_semana para o valor desejado e rodar linha por linha.

- *Substititua "aluguel_semana <- 200" por "aluguel_semana <- 247.53"*

Se voce pode fazer isso para calcular o aluguel, imagina o que esse sistema pode fazer com sua analise de dados.

3.2.2 Vetores

Os objetos com multiplos valores de um mesmo tipo sao chamados de vetores. Para criar um objeto vetor inserimos os valores dentro de parenteses separados por virgulas. Virgulas no R servem para separar o codigo. Entao se voce quiser digitar numeros quebrados nunca utilize **virgula**, utilize **ponto**.

```
vetor.A <- c(10.5, 20.6, 30.2)
vetor.B <- c(2, 2, 2)
melhores.comidas <- c("coxinha","pastel","lasanha") #Lembre-se, caracteres precisam estar dentro

vetor.A
vetor.B
melhores.comidas
```

3.2.3 Funções

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

```
#funcao que calcula raiz quadrada

sqrt()

sqrt(25)
```

Existem funcoes dentro do R para todo tipo de coisa! Vamos criar um conjunto de dados e fazer alguns calculos utilizando algumas funcoes.

```
#Notas do terceiro ano A
notas <- c(7, 8, 10, 5, 2, 4, 6.6, 7.9, 8.9, 0)

#Qual a media da turma?
mean(notas)

#Qual a mediana?
median(notas)

#Qual o desvio padrao das notas?
sd(notas)

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

#resumo das notas
summary(notas)
```

Cada uma dessas funcoes realiza algum calculo em especifico, no caso de *summary()*, ela realiza o calculo da media, mediana, maximo, minimo e quartis, tudo ao mesmo tempo. Talvez em algum momento voce nao encontre a funcao que voce precisa para determinada tarefa, nesse caso, voce pode criar sua propria funcao! Voce pode criar uma funcao utilizando outras funcoes, para fazer algo especifico para voce. Por enquanto, vamos apenas ser usuario de funcoes e nao criadores. Mas se voce realmente quer saber como criar uma funcao, nao irei te limitar. Abaixo voce encontra um dos exemplos mais simples de funcao. Utilize sua criatividade, seus conhecimentos de objetos, funcoes e vetores para criar alguma funcao. Mas sinta-se a vontade para pular essa parte, uma vez que ela nao eh essencial ao uso do R.

3.2.3.1 Criando uma funcao (opcional)

Vamos imaginar que voce precise criar uma funcao para somar dois numeros. Eu sei, eh uma funcao simples, mas vai nos ajudar a entender a estrutura de

uma funcao. Para criar uma funcao, precisamos utilizar a funcao chamada de *function()*.

```
#Estrutura de uma funcao
nome <- function(variables) {

    #dentro das chaves adicionamos o processo que queremos realizar.
    #Nesse caso, somar dois numeros.
}

#Criando a funcao soma

soma <- function(primeiro_numero, segundo_numero) {

add <- primeiro_numero + segundo_numero
print(add)

}
```

Essa eh uma funcao que possui dois argumentos (primeiro_numero, segundo_numero) e utilizando esses argumentos ela realiza um calculo de soma e atribui esse resultado a um objeto que chamamos de "add". Por fim, utilizamos a funcao *print()* para mostrar o valor embutido em "add".

Agora, vamos imaginar que talvez a funcao *summary()* nao seja o que voce precisa. Voce precisa mesmo eh de uma funcao que te forneca, media, mediana e desvio padrao dos seus dados.

Nesse caso, podemos criar a funcao *my_summary()* utilizando a funcao *function()*.

```
#Criando a funcao my_summary()

my_summary <- function(v) {

media   <- mean(v)
mediana <- median(v)
desvio  <- sd(v)

resultado <- c(media, mediana, desvio)
print(resultado)

}
```

Calma, tem muita coisa ai, mas eh tudo o que a gente ja sabe. Na funcao *my_summary()*, precisamos inserir apenas um argumento do tipo vetor, como

por exemplo, nosso objeto "notas". Esse argumento sera aplicado a tudo que possuir o caractere "v" dentro da funcao. O "v" a gente definiu em function(v). Mais uma vez, as chaves **{}** que abre e fecha contem tudo o que a gente quer que a funcao realize, linha por linha. Primeiro, queremos que a nossa funcao calcule a media e atribua o resultado a um objeto chamado "media". Segundo, calcular a mediana e atribuir o resultado a um objeto chamado "mediana", depois o desvio. Depois criamos um vetor chamado de "resultado" que armazena os objetos "media", "mediana" e "desvio". Em seguida, pedimos para o R nos mostrar os valores desse vetor "resultado" utilizando a funcao *print()*.

Vamos usar nossa funcao personalizada para calcular nosso objeto "notas".

```
#Notas do terceiro ano A
notas <- c(7, 8, 10, 5, 2, 4, 6.6, 7.9, 8.9, 0)

#Utilizando a funcao my_summary()

my_summary(notas)
```

Veja que a funcao retornou os valores do nosso vetor, mas apenas isso. Mais para frente, a gente ve como criar funcoes mais elaboradas, como por exemplo para simular gastos de alugueis como anteriormente.

3.3 Tabela de dados

3.3.1 Data frame

Data frame nada mais eh do que uma tabela de dados. Vamos direto ao que interessa e usar a funcao *data.frame()* para criar uma tabela. Iremos criar as notas de 10 alunos e seus respectivos nomes. Se voce ainda estiver na mesma sessao de R, podemos ver que o objeto "notas" ainda esta salvo na janela 3 (Environment e History) e podemos aproveita-lo. De qualquer forma vou recriar "notas", caso por algum motivo voce SAIU do R, um absurdo, mas acontece.

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

#Primeiro criamos as notas
notas <- c(7, 8, 10, 5, 2, 4, 6.6, 7.9, 8.9, 0)

#Segundo criamos os nomes
nomes <- c("Marilia", "Jonatas", "Guilherme", "Camila",
"Fiuza", "Brunno", "Diego", "Mathews", "Biny", "Rebecca")

#Terceiro utilizamos a funcao data.frame() para criar a tabela
```

```
tabela <- data.frame(nomes, notas)
tabela
```

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

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

-Variaveis sao colunas; -Observacoes sao linhas; -Cada valor na sua celula.

Com essa formatacao de planilha, voce consegue fazer quase tudo no R, para manipulacao de dados, construcao de graficos e analises estatisticas.

Logo, para voce acessar uma variavel de uma tabela, voce pode usar o **$** da seguinte forma:

```
#Acessando uma variavel

tabela$nomes
tabela$notas
```

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

```
fix(tabela)
```

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

Agora sabendo como acessar as variaveis de uma tabela, voce pode utilizar funcoes para calcular uma variavel especifica do data frame.

```
mean(tabela$notas)
```

Utilize as funcoes abaixo para calcular mediana, maximo, minimo:

- 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 o quais podemos utilizar para treinar nossas habilidades. Para isso, precisamos dizer para o R qual banco de dados queremos invocar utilizando a funcao *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).

As vezes a planilha eh importada com as variaveis mal formatadas e isso pode gerar problemas de reconhecimento por parte do R. Nesse caso, eh sempre interessante realizar a verificacao dos dados utilizando a funcao *str()* ou outras funcoes similares. Caso o R nao esteja identificando as variaveis corretamente, duas funcoes muito uteis podem ser utilizadas: *as.numeric()* e *as.factor()*.

```
data(iris)

#Convertendo variavel para numerica
iris$Sepal.Length <- as.numeric(iris$Sepal.Length)

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

Basicamente voce ta transformando em numerica a variavel Sepal length da planilha iris e atribuindo a ela mesma. Dessa forma, voce faz a alteracao de forma permanente. A mesma coisa ocorre com as.factor.

Vamos brincar um pouco com a planilha iris. Primeiro veja que ela esta construida de forma organizada (variaveis nas colunas, observacoes nas linhas, cada celula um valor). Podemos entao realizar o summary() para saber a media, mediana, maximo e minimo da largura da petala (Petal.Width).

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

Porem note que existe 3 especies diferentes. A funcao *summary()* nao leva isso em consideracao. O ideal seria calcular o *summary()* para cada especie. Para isso temos uma funcao muito legal chamada *tapply()*, onde: o primeiro argumento eh a variavel numerica, o segundo a variavel com os fatores, e o terceiro argumento eh a funcao que voce quer aplicar.

```
tapply(iris$Petal.Width, iris$Species, summary)
```

3.3.3 Importando sua planilha

No RStudio podemos importar uma planilha do formato excel (xlsx) utilizando o botao na janela Environment (superior direita):

Environment > Import Dataset > From excel

Acho pouco eficiente ensinar a importar dados de um diretorio. Acho pouco pratico, uma vez que temos o botao de importar. Ao inves disso, irei treina-lo a utilizar projetos de R reprodutiveis e organizados. Ao trabalhar no R, voce seguira um protocolo basico de como tratar os dados a serem analisados. Ao longo do tempo, voce sera capaz de otimizar esse protocolo e aplicar para quase todo tipo de dado que voce possuir. De qualquer forma, saiba que o R trabalha com uma pasta especifica chamada de diretorio, e para saber qual eh o seu diretorio padrao, utilize a funcao *getwd()*:

```
getwd()
```

Ao executar esse funcao, o R te retornara o caminho da pasta do diretorio padrao atual que provavelmente eh a pasta “documentos” do seu computador.

Para baixar uma planilha de dados ficticios clique no link abaixo e tente importa-la atraves do botao **Import from excel**.

Clique aqui para baixar a planilha ficticia

3.4 Graficos

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

3.4.1 Grafico de histograma

Esse grafico nos mostra a frequencia dos valores da variavel de interesse, como por exemplo comprimento da petala 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 nao eh? Mas podemos modificar ainda mais nosso grafico. Alterar cores, nomes dos eixos, titulo, etc. Vamos ver mais argumentos com o grafico de dispersao de pontos.

3.4.2 Grafico de dispersao

Esse grafico nos mostra a relacao entre duas variaveis continuas. Vamos utilizar o banco de dados "iris" para construir e visualizar alguns graficos. Para construir um grafico de dispersao utilizamos a funcao *plot()*. Note que a funcao plot, possui varios argumentos que podemos ir adicionando para personalizar

o grafico. Alem disso, para deixar o codigo mais organizado, sempre apos a virgula 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)
```

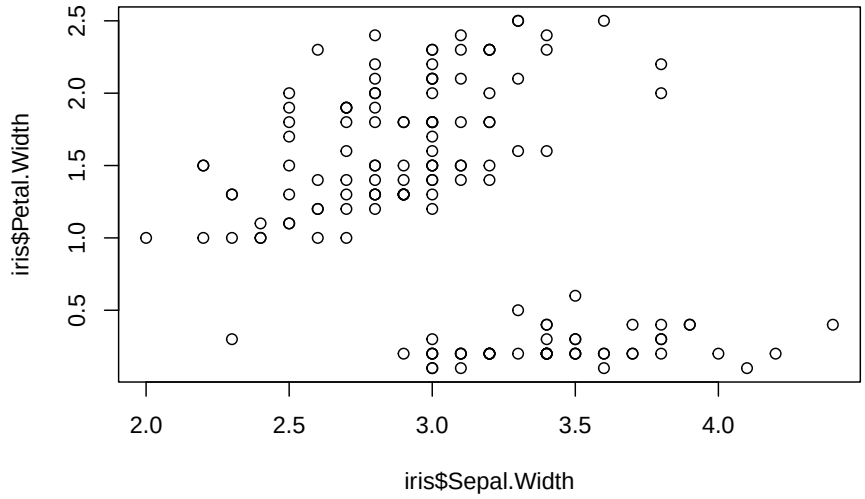


Figure 3.3: Gráficos com R base

```
#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",
     xlab = "Sepal Width"
     )
```

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

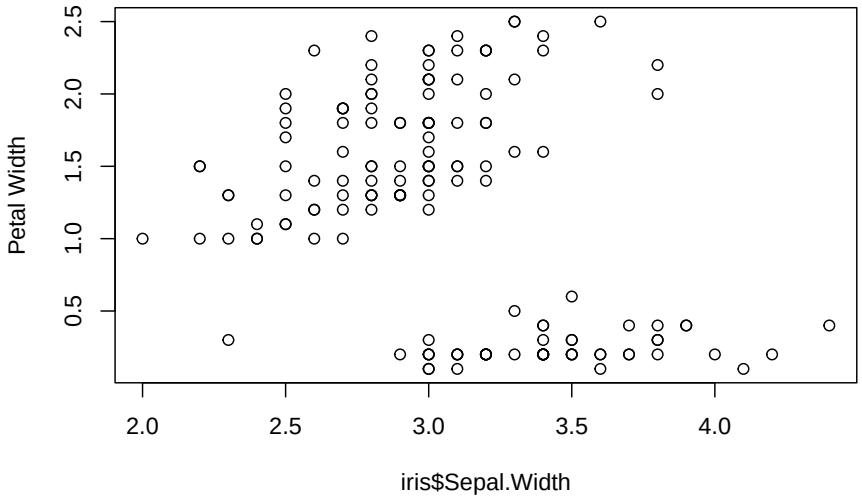


Figure 3.4: Gráficos com R base

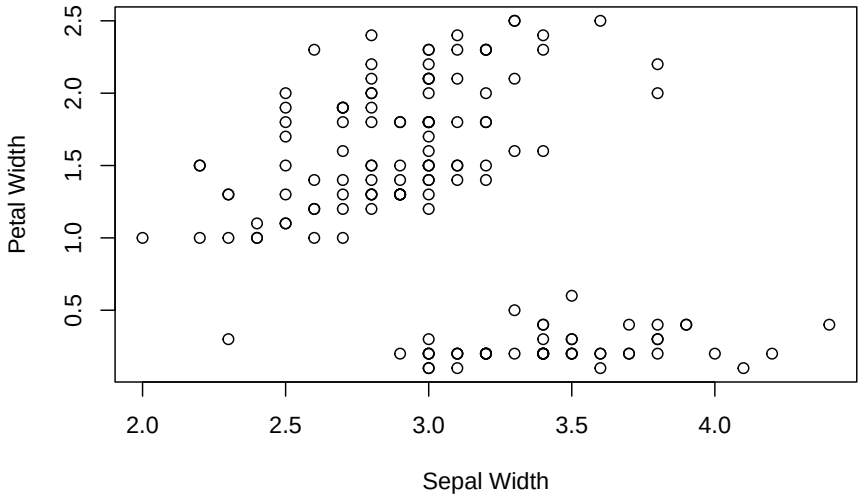


Figure 3.5: Gráficos com R base

```
     xlab = "Sepal Width",
     col = "blue"
     )
```

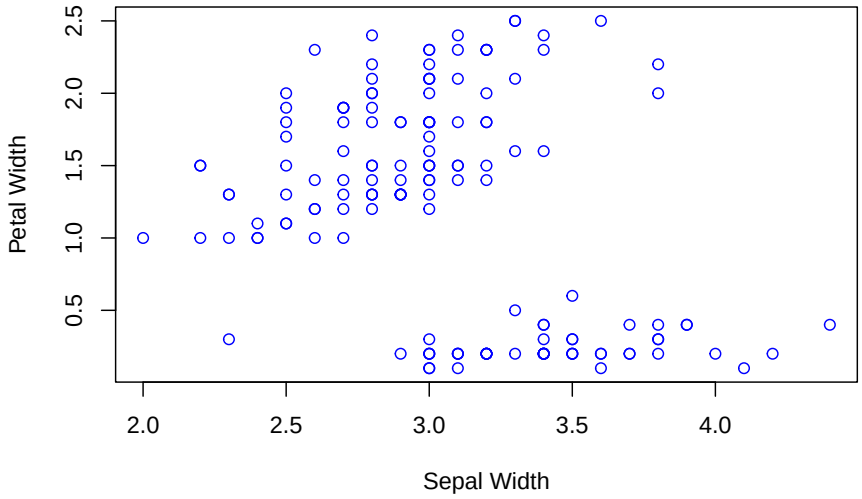


Figure 3.6: Gráficos com R base

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

```
#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"
     )
```

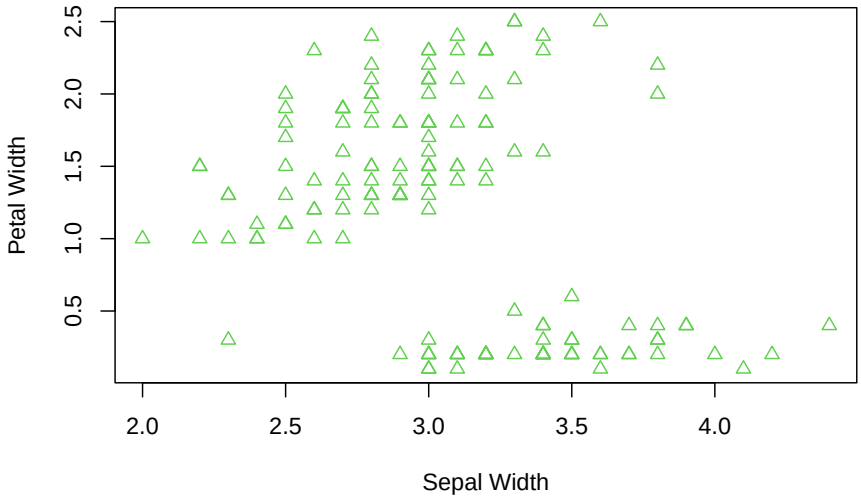


Figure 3.7: Gráficos com R base

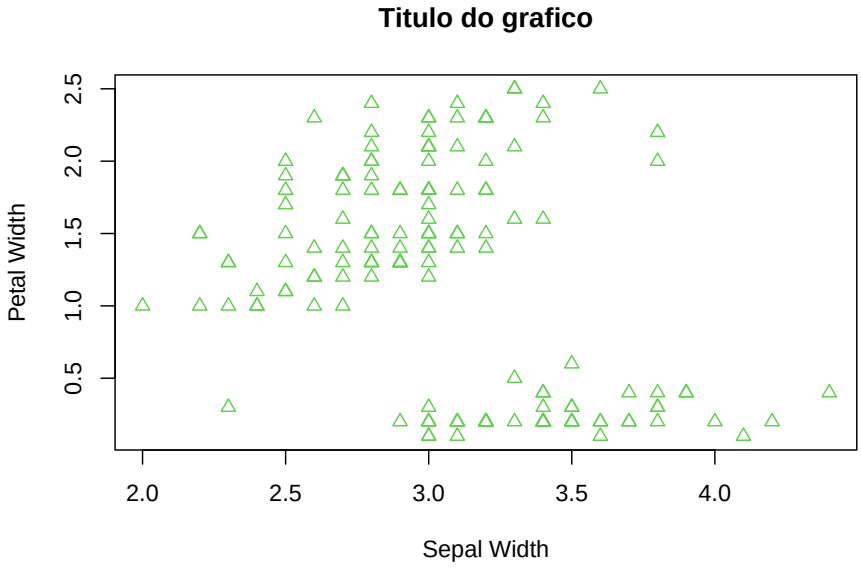


Figure 3.8: Gráficos com R base

3.4.3 Grafico boxplot

Para criar o grafico de boxplot, iremos utilizar a variavel numerica Sepal.Width do banco de dados iris em relacao a variavel Species. Note que na funcao plot, as duas variaveis (argumentos) eram separados por virgula, aqui utilizamos o ~ para relacionar a variavel numerica com a categorica.

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

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

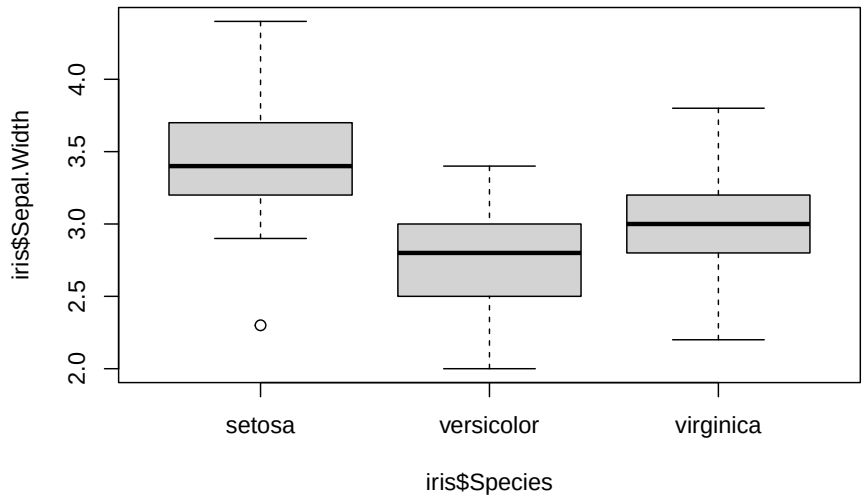


Figure 3.9: (#fig:3_basic_boxplot-1)Boxplot com R base utilizando o banco de dados iris

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

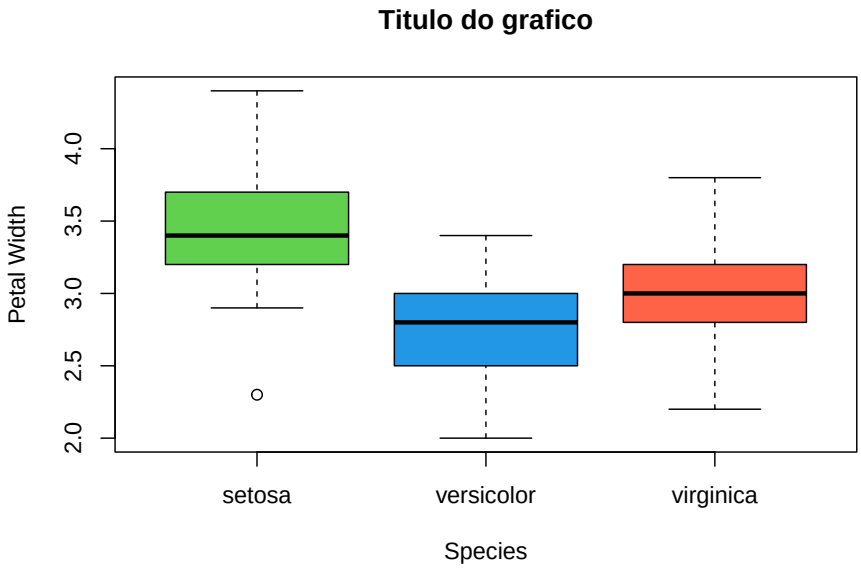


Figure 3.10: (#fig:3_basic_boxplot-2)Boxplot com R base utilizando o banco de dados iris

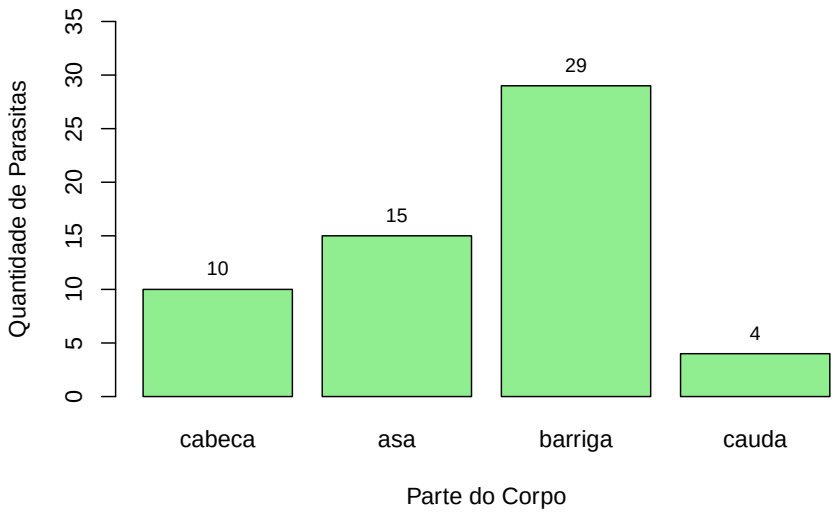
Perceba que o argumento col pode ser um vetor, e pode receber mais de um valor. Nesse caso, demos uma cor para cada fator de Species. Tente substituir os numeros por outros, ou colocar o nome das principais cores como: red, green, blue, yellow... So nao esqueca de colocar entre aspas.

3.4.4 Grafico de barras

Para criar um grafico de barras de contagem, vamos criar nosso proprio data frame. Imagine que queremos tabular a contagem total de ectoparasitas (piolho) de ave de cada parte corporal. Como eh algo simples, podemos criar direto 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)
```

```r
# 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")

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



Faca as seguintes modificacoes no codigo e veja o que acontece:

- Veja o que acontece se voce 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 funcao text(), troque pos = 3, por pos = 1.
  (rode o grafico antes para nao sobrepor os numeros)

Eh possivel fazer muitos outros tipos de graficos, utilizando diferentes funcoes. Mas a ideia desse capitulo eh ensinar a utilizar o R, e nao a criar graficos. Entao vamos com calma, pois quando voce pensar que nao, ja foi.

3.5 Pacotes

Agora ja podemos comecar a expandir nosso universo do R. Tudo que fizemos ate agora foi utilizando o proprio R base. A partir de agora, iremos incluir um conjunto de novas funcoes atraves dos pacotes de R. Antes de tudo, precisamos instalar o pacote que queremos utilizar, e para isso existe duas formas:

- Utilizando o botao *Tools* do RStudio.
- Utilizando a funcao *install.packages("nome_do_pacote")*

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

Utilizando o botao *Tools* do RStudio click em: **Tools > Install package > digite o nome "lattice".** Ao comecar a digitar o R ira sugerir opcoes de pacotes com as iniciais que voce digitou. Depois clica em "instalar".

Utilizando a funcao *install.packages()*

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

O processo de instalacao mostra no console varios processos e enquanto isso acontece, um icone de "stop" vermelho aparece na parte superior direita da janela do console. O processo so termina quando esse icone 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 eh so dizer que queremos utilizar o pacote. O processo de instalacao em um determinado computador so precisa ser feito uma vez. Porem, toda vez, voce precisa dizer para o R, qual pacote voce estar utilizando. Para isso utilizamos a funcao *library()*.

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

Se voce carregou o pacote e o R nao te retornou nenhum erro, ta tudo certo. As vezes ele te retorna um Warning message, tambem esta tudo certo. Vamos adiante.

O pacote lattice que acabamos de instalar nos fornece varias funcoes para criacao de graficos. Tenha em mente que existem milhares de pacotes cada um trazendo um conjunto de funcoes e banco de dados de diferentes contextos, tanto para criar graficos, quanto para realizar modelos estatisticos avancados.

Vamos criar alguns graficos utilizando as funcoes que o pacote lattice nos forneceu.

```r
#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
```

```r
# 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")
```

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

Acho que ja deu pra entender como o R funciona ne? Nos proximos capitulos, voce vai aprender a como trabalhar de forma eficiente no R, seguindo um fluxo de trabalho eficiente, e o melhor de tudo, reproduzivel.

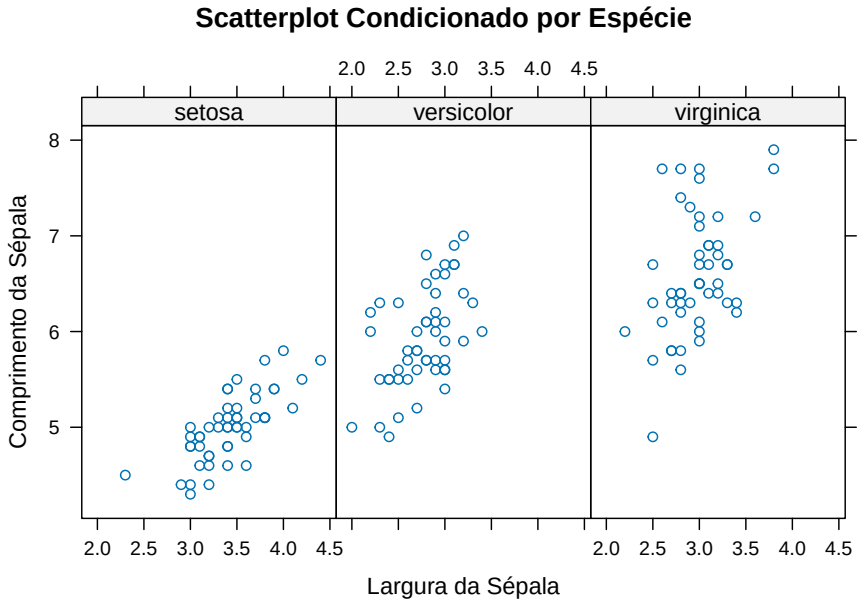


Figure 3.11: Gráficos com Lattice

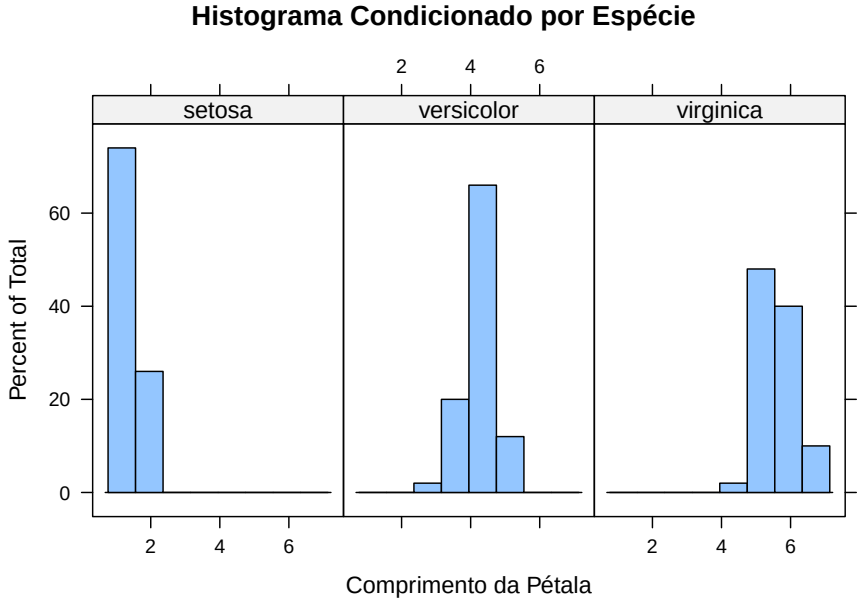


Figure 3.12: Gráficos com Lattice

Chapter 4

Como trabalhar no R

Se voce ja tem uma nocao de R, mas ainda sente dificuldade de organizar seu fluxo de trabalho, esse capitulo eh pra voce! Utilizaremos tecnicas de ciencia de dados e repositorios como o github para tornar o trabalho no R prazeroso e eficiente!

Se voce possuir um banco de dados para analizar a quantidade de parasitas de aves da mata atlantica, ou o comportamento sexual de caranguejos chama-mares (ou seja la qual for o seu estudo), aproveite para utiliza-lo aqui nesse capitulo. Caso voce ainda nao possua seu banco de dados, trabalharemos com algum banco de dados nativo do R. Optei por utilizar o banco de dados "iris", mas sinta-se a vontade para utilizar qualquer outro.

Primeiro vamos exportar o banco de dados iris para que possamos simular todo o precesso desde a sua importacao. Para isso, iremos carregar o banco de dados iris utilizando a funcao *data()*, e utilizaremos a funcao *write.csv()* para exportar, no formato csv, a planilha "iris" para o computador. Deixe essa planilha salva em algum lugar no computador e finja que ela eh sua, mais para frente iremos utiliza-la.

```
#Carregando pacote de dados iris
data(iris)

#Salvando (exportando) iris no computador
write.csv(x = iris,            #nome do banco de dados do R
          file = "iris.csv",   #nome do banco de dados salvo no computador
          row.names = FALSE)   #Mostrar o nome das linhas na planilha salva utilizando TRUE ou na
```

Para entender a funcao *write.csv()* modifique alguns argumentos:

- Substitua: file = "iris.csv" por file = "planilha_do_R.csv".

- Substitua: row.names = FALSE, por row.names = TRUE.

Por padrao, iremos utilizar sempre row.names = FALSE.

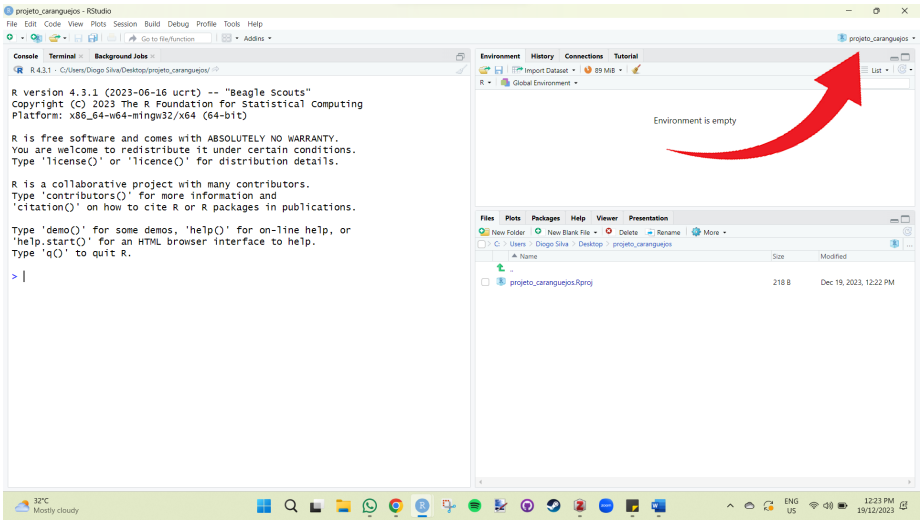
4.1 Criando um projeto de R

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, ira aparecer uma janela para escolher o nome do projeto e o local no computador que seu projeto ira ficar. Coloque o nome desejado (sugestao: "projeto_caranguejo"), selecione qualquer pasta no computador (sugestao: desktop) e clique em "criar projeto".

Se voce fez tudo direitinho, o RStudio estara do seguinte forma. Note que ao inves de project (none), estara o nome do projeto que voce criou.



Sugiro meu nobre consagrado leitor, que voce va no local do computador que voce criou o projeto e veja a pasta, veja o arquivo de R que foi criado. Voce tambem pode fazer isso utilizando a janela de Files, onde mostrara todas as pastas do seu diretorio que, a partir de agora, sera a pasta que o seu arquivo de projeto de R esta situado. Todos os scripts, arquivos e planilhas que voce ira utilizar nas suas analises, ficara dentro da pasta do projeto. Isso significa que vai ficar tudo solto, baguncado? Claro que nao. Iremos criar pastas organizadas onde hospedara cada coisa que iremos trabalhar como por exemplo: dados brutos, dados processados, scripts, outputs, etc. Voce pode criar manualmente, mas porque fariamos isso se temos o R para fazer por nos?

4.2 Organizando o projeto de R

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

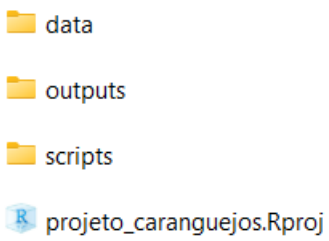
```
install.packages("here")
library(here)
```

Apos a instalacao do pacote "here" iremos criar uma funcao que criara as pastas automaticamente no nosso diretorio. Nao se assuste com o script da funcao, ela eh mais simples do que parece e voce nao precisa entende-lo por completo. Apenas rode o codigo para criar a funcao e depois rode o codigo que utiliza a funcao para criar as pastas.

Antes de rodar o codigo, certifique-se de que voce esta no projeto de R que voce criou.

```
# Criando funcao para criacao das pastas do projeto
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 voce encontra as pastas “raw” e “processed”. Em outputs voce encontra “figures”, “temp” e “tables”. Em script, voce encontrara nada (por enquanto). O arquivo com o simbolo do R “projeto_caranguejos.Rproj” eh o seu projeto de R. Voce pode abri-lo (dando duplo clique) toda vez que voce for trabalhar no projeto. Isso abrira o Rstudio ja 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 voce tem tudo pronto para comecar a trabalhar com um fluxo de trabalho eficiente e reprodutivel!

4.3 Trabalhando em um projeto de R

Antes de tudo baixe a planilha de dados ficticios de uma especie de passarinho (tambem ficticia), que contem informacoes ficticias sobre peso (g) e quantidade de parasitas de tres diferentes localidades (A, B, C):

Clique aqui para baixar a planilha ficticia

Na pasta **data > raw** voce adiciona sua planilha de dados brutos ficticias (fake_data.xlsx). O fluxo de trabalho sera o mais simples possivel, mas envolvera etapas essenciais da analise de dados.

Limpar dados brutos > Realizar analise > Mostrar graficos

Para limpar a planilha de dados brutos, iremos criar um script para isso. Utilizaremos funcoes do pacote dplyr para modificar nome das variaveis, nome 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
```

Apos isso, utilizaremos os comentarios, para organizar o script em topicos fazendo manualmente ou utilizando o atalho: **ctrl + shift + R**. A primeira parte do codigo eh a instalacao dos pacotes que iremos utilizar no processo.

```
# Packages ------------------------------------------------------------
if(!require(dplyr)) install.packages("dplyr") #se o pacote dplyr nao estiver sido instalado, ele
library(dplyr) #carregar pacote
```

Segundo, importamos o banco de dados a ser processado (limpado) utilizando a funcao *read.csv()* e atribuindo (<-) a um objeto que iremos chamar de "dados". Para importar voce so precisa especificar o caminho do diretorio em que o arquivo ta inserido. Como estamos utilizando o projeto de R, o diretorio eh onde o projeto esta. Dessa forma, nao ha necessidade de dizer para o R qual o seu diretorio de trabalho. Logo fica facil importar os dados brutos pois sabemos que ele esta na pasta dados > raw > iris.csv. Dizemos isso para o R utilizando os nomes das pastas separado por barras e por fim, o nome do arquivo (iris) e suas extensao (.csv).

```
# Load data ------------------------------------------------------------
dados <- read.csv("data/raw/iris.csv")
```

Agora que o banco de dados foi importado, podemos fazer um check up basico utilizando algumas funcoes.

```
# 1. Basic checks -----------------------------------------------------------
nrow(dados)              # How many rows
str(dados)               # Variables classes
attributes(dados)        # Attributres
head(dados)              # First rows
any(duplicated(dados))   # There is any duplicated rows?
any(is.na(dados))        # There are NAs in the data?
```

As vezes os nomes das variaveis nao sao faceis de trabalhar, possuindo espaco, acentos, letras maiusculas e minusculas, e nomes confusos. Alem de complicar o

codigo, o nome de algumas variaveis podem bugar o script (caso das acentuacoes de palavras em portugues). Entao vamos mudar o nome das variaveis utilizando a funcao *rename()* do pacote **dplyr**. Iremos deixar os nomes mais faceis de trabalhar, deixando todas as letras minusculas.

```
# 3. Rename columns names-------------------------------------------------------
head(dados)
dados2 <- rename(iris,                          #sua planilha bruta
                 sepal_length = Sepal.Length,  #nomes das variaveis (nome_novo = nome
                 sepal_width  = Sepal.Width,
                 petal_length = Petal.Length,
                 petal_width  = Petal.Width,
                 species      = Species)
head(dados2)
```

Agora vamos dizer para o R os tipos de nossas variaveis. Um problema muito comum, eh o R nao reconhecer que a variavel categorica, nao possui fatores, mesmo possuindo. Por exemplo, em "iris", temos a variavel categorica 'Species' com seus fatores (setosa, versicolor, virginica). Eh possivel que no seu banco de dados, o R nao reconheca isso, entao vamos utilizar a funcao *levels()* para verificar se o R entendeu quais sao os fatores. Se o R retornar NULL, voce precisa dizer para o R utilizando a funcao *as.factor()*.

```
# 4. Fix factor variables------------------------------------------------------
# check factor variable
levels(dados2$species)

# making factor variables
dados2$species <- as.factor(dados2$species) #atribuindo a variavel specie transformada

# check factor variable
levels(dados2$species)
```

Agora podemos renomear o nome dos fatores de uma variavel categorica utilizando a funcao *recode_factor()*. Nesse processo podemos identificar erros de digitacao que ocorreram durante o planilhamento dos dados. Por exemplo, voce pode ter digitado "Macho", "macho" e "male" na variavel sexo durante o planilhamento. O R vai identificar os tres como sendo fatores diferentes mas na verdade sao o mesmo fator. Portanto eh necessario que seja padronizado.

```
# 4.1 fix species factors

#check factors
levels(dados2$species)
```

```
#rename factors
dados2$species <- recode_factor(dados2$species,                 #variavel categorica
                                setosa = "iris_setosa",         #mudando nome setosa para iris_se
                                versicolor = "iris_versicolor",
                                virginica = "iris_virginica")

#check factors
levels(dados2$species)
```

Agora que limpamos nossos dados brutos, iremos exportar nossa planilha limpa. Perceba que o processo do script eh colocar a planilha bruta de um lado e sair limpa e bonitinha do outro. Eh com os dados processados que iremos trabalhar de fato. Vamos salvar sempre no formato csv por ser um formato mais simples, leve e estavel.

```
# 8. Save processed data ------------------------------------------------------
write.csv(x = dados2,                                  #nome da planilha que voce quer exportar
          file = "data/processed/processed_iris.csv", #local e nome da planilha exportada
          row.names = FALSE)                           #sempre utilizaremos row.names = FALSE
```

O processo de manipulacao de dados no R eh bastante completo e existem diferentes formas de limpar seus dados brutos. O pacote **dplyr** possui funcoes capazes de selecionar variaveis, selecionar linhas, criar variaveis, criar subsets, entre outras. Para aprimorar seu script de limpeza de dados, voce precisara aprender sobre manipulacao de dados e incluir no processo, os codigos no seu script. Por sorte, existe muito material disponivel na internet sobre o assunto. Talvez algum dia eu crie um material sobre manipulacao de dados com dplyr, mas aqui esse nao eh meu objetivo. Meu objetivo eh simplesmente fornecer o esqueleto teorico, te dando base para crescer de forma mais direcionada.

Por fim, nao esqueca 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 estatisticos para responder nossa hipotese. Vamos supor que nossa pergunta seja saber se existe diferenca entre o tamanho das petalas entre as tres especies. Organizaremos o script de analise de forma similar ao script de limpeza, ou seja, escreveremos um cabecalho, carregaremos pacotes, importaremos e analisaremos dados de forma organizada.

Vamos fazer um cabecalho para nosso script de analise.

```
# Script to load and analyse processed data iris
# Author: Marilia T. Ericcson
# Data: 2019-11-07
```

Vamos criar o codigo 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 importacao eh 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. Basic checks ---------------------------------------------------------------
head(dados)
str(dados)
leves(dados$species)
```

Vamos realizar uma analise para comparar a largura das petalas entre as especies. Para isso, iremos utilizar o teste nao parametrico de kruskal-wallis atraves da funcao kruskal.test(). Nessa funcao, voce precisa dizer qual sua variavel numerica de interesse em relacao a (~) sua variavel categorica e seus grupos (fatores). Veja que o p-value eh menor que 0.05, entao dizemos que existe diferenca da largura entre as tres especies.

```
#realizando o teste
kw <- kruskal.test(iris$Sepal.Width ~ iris$Species)
kw
```

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

```
#Ajeitando 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! Ja temos nosso segundo script pronto. Agora eh so salva-lo na pasta script com o nome: '02_kruskal_test'.

4.3.3 Etapa 3: criar grafico

Agora vamos criar o grafico para representar nosso resultado de que existe diferenca da largura das petadas entre as especies de iris.

```
# Script to load and analyse processed data iris
# Author: Marilia T. Ericcson
# Data: 2019-11-07

#Carregar pacotes
library(ggplot2)

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

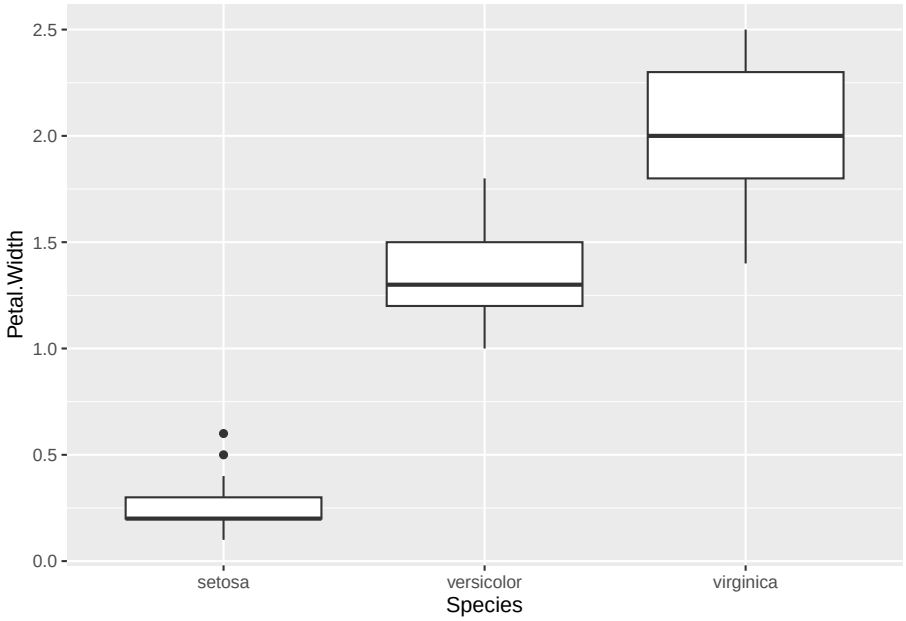


Figure 4.1: ggplot2

Voce acabou de criar um grafico utilizando o pacote ggplot2 que eh o pacote mais utilizado para producao de graficos no R. O codigo pode parecer um pouco confuso no inicio, mas nao precisa estudar muito para perceber que ele eh muito simples e intuitivo. Para aprender mais sobre graficos, sugiro os livros: https://rkabacoff.github.io/datavis/ e https://r-graphics.org/

Nao esqueca de salvar o script com o nome '03_grafico' #Etapa 4: Replicabilidade

O seu projeto esta organizado de forma que se torna facil a replicabilidade em qualquer computador gerando todos seus outputs (tabelas e figuras) automaticamente! Para isso, voce precisa manter e JAMAIS deletar ou modificar seus dados brutos (pasta raw). Todos os demais arquivos podem ser produzidos com um click a partir dos seus dados brutos, portanto, podem ser deletados. Isso se torna util ao compartilhar seus dados com alguem, em que voce pode zipar todo seu projeto, enviar para um contribuidor e ele podera ter acesso a tudo rodando apenas um script. Eh 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--------------------------------
```

```r
# 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 esse codigo, todos os seus scripts sao rodados e todos os seus outputs e planilhas processadas sao geradas. Mas lembre, seu dado bruto contido na pasta data/raw, nao pode ser deletada ou modificada.

Salve esse script com o nome “run_project”.

Chapter 5

Git e GitHub