

# R Zero - do simples ao complexo

Diogo J. A. Silva

2023-12-19



# Contents

<b>1</b>	<b>Bem-vindos</b>	<b>5</b>
<b>2</b>	<b>Pre-requisitos</b>	<b>7</b>
2.1	Instalando R e RStudio . . . . .	7
<b>3</b>	<b>R para iniciantes</b>	<b>9</b>
3.1	Entendendo o RStudio . . . . .	9
3.2	Objetos, vetores e funções . . . . .	12
3.3	Tabela de dados . . . . .	15
3.4	Graficos . . . . .	18
3.5	Pacotes . . . . .	27
<b>4</b>	<b>Como trabalhar no R</b>	<b>31</b>
4.1	Criando um projeto de R . . . . .	32
4.2	Organizando o projeto de R . . . . .	33
4.3	Trabalhando em um projeto de R . . . . .	34
<b>5</b>	<b>Git e GitHub</b>	<b>43</b>



# Chapter 1

## Bem-vindos

Esse livro foi criado para organizar, de forma fácil, os códigos dos tipos de gráficos que podem ser criados utilizando o R, trazendo um compilado essencial da Visualização de dados. Claro que já existem livros muito bons sobre o assunto (<https://rkabacoff.github.io/datavis/> e <https://r-graphics.org/>), porém são livros em inglês e pouco amigáveis para iniciantes. Claro que existe um livro muito bom em português e serve para iniciantes (<https://analises-ecologicas.netlify.app/>), mas não é focado em visualização de dados. Portanto, nesse livro, iniciantes terão uma boa base introdutória no R e serão direcionados para a visualização de dados de forma mais simples e organizada.

Neste livro, você encontrará o essencial para produzir gráficos em R utilizando o pacote ggplot2. Para o melhor aproveitamento do conteúdo, você precisará entender como o R funciona e também compreender de forma clara os conceitos de variáveis estatísticas (ou seja, variável nominal, quantitativa, etc.). Com isso em mente, você poderá pular para qualquer parte do livro e aproveitar seu conteúdo, que se mostrará independente caso você tenha todos os pré-requisitos (ou seja, R + RStudio instalados).

Se você já possui os conhecimentos básicos de estatística e R, sinta-se livre para pular para qualquer parte do livro e aproveitar os códigos dos gráficos. :)



## Chapter 2

# Pre-requisitos

### 2.1 Instalando R e RStudio

Para você usar esse livro de forma otimizada, precisará ter o R e RStudio instalados. Nem vou falar sobre a instalação, pois sei que você é capaz de encontrar um tutorial feito por uma criança de 8 anos no YouTube.

Se voce eh um completo iniciante, a forma ideal de utilizar esse livro, eh seguir o que o livro pede executando os comandos no RStudio.





## Chapter 3

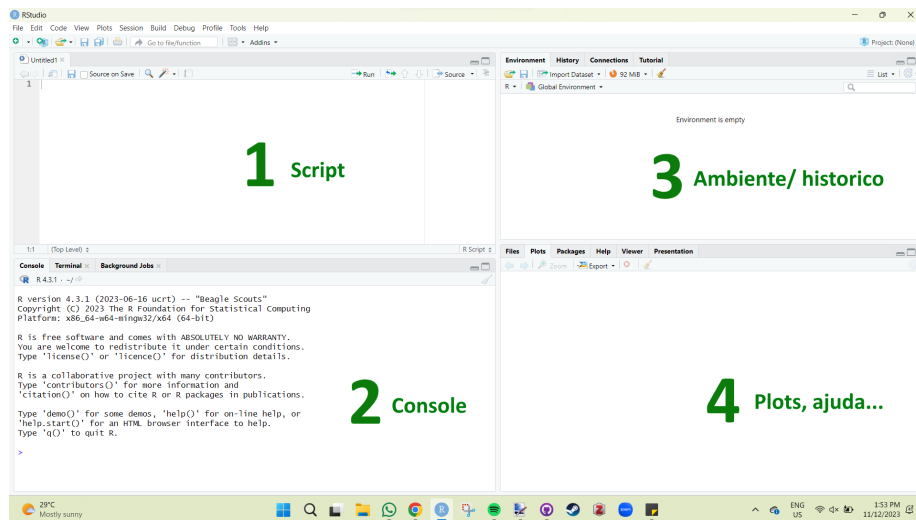
# R para iniciantes

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.

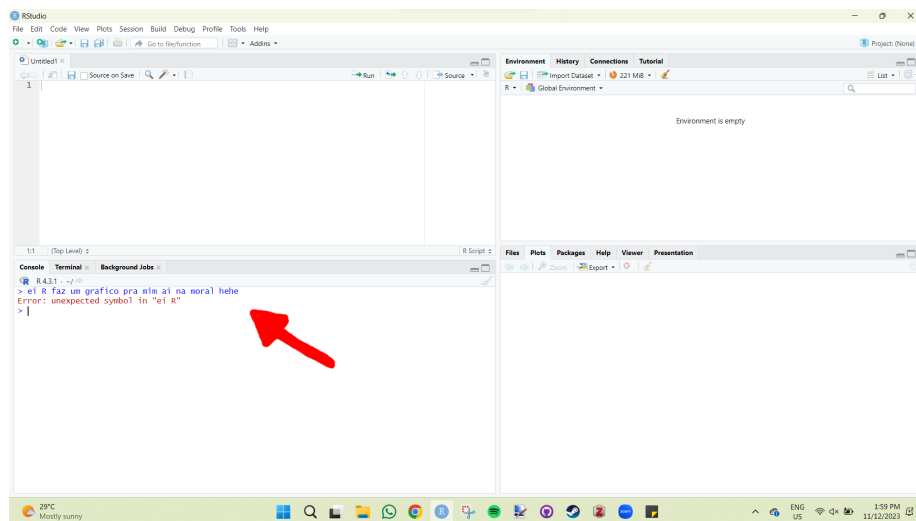
### 3.1 Entendendo o RStudio

Tudo que iremos fazer será através do R Studio devido à organização que ele nos fornece. 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. Até onde sei, ainda não existe nenhum programa com a precisão do R. No entanto, para utilizar o R, é necessário falar a linguagem dele.

Se voce for “falar” diretamente com o R no console e digitar e apertar o botao “Enter” do teclado: *“Ei R, faz um grafico pra mim ai na moral hehe”*. Ele vai te retornar um erro em vermelho.



Ele está basicamente indicando que não entendeu o que você disse. Vamos

tentar usar a linguagem dele. Digite: `1 + 1`

Você vai notar que ele retornará o resultado **2**. Você pode realizar qualquer operação matemática, desde que utilize uma linguagem que ele possa compreender.

**2+2 (soma)**

**2-2 (subtração)**

**2\*2 (multiplicação)**

**2/2 (divisão)**

**2^2 (potenciação)**

Agora digite uma equação mais complexa:

**2+2-2^2/2**

O resultado será **2**. Mas e se você quiser mudar? Trocar a raiz quadrada de 10 (`sqrt(10)`) para a raiz quadrada de 4 (`sqrt(4)`)? Você vai ter que digitar tudo de novo? De fato, sim, vai ter que digitar tudo de novo, porém você pode usar a tecla “seta para cima” do teclado, e ele vai te mostrando os últimos comandos dados.

Mas é por essa falta de praticidade de conversar com o R, que o R studio ganha sua importância na janela de Script.

Vamos criar um pequeno script e aprender algumas coisas. Copie o código abaixo e cole na janela de script.

```
# Mini Script
a <- 1
b <- 2
a + b
```

Veja que no Script, o documento é organizado por linhas (1, 2, 3, 4...). Nesse caso, cada linha representa um comando. Com exceção da linha 1 que é na verdade um comentário ou uma nota. Qualquer código pode ser transformado em um comentário, para isso, é só adicionar o `#` no início da linha. Comentários, não são comandos, e se você tentar executá-los, automaticamente irão pular e executar a próxima linha.

Para executar um comando no script não utilizamos apenas o “Enter” e sim o “Ctrl + Enter”. Ao executar a segunda linha `a <- 1`, você pediu para o R atribuir o número **1** à letra **a**. Ao executar a terceira linha, você atribuiu o número **2** à letra **b**. E por fim, ao executar a quarta linha, você realiza uma soma de **a + b**, ou seja **1 + 2**.

Você já deve ter percebido que o código `<-` significa **atribuir**.

*Nesse momento, nobre leitor, eu espero que você esteja realizando diferentes operações matemáticas, utilizando diferentes atribuições.*

Vamos continuar! Um objeto pode possuir mais de um valor ao mesmo tempo. Vamos fazer uma atribuicao chamada de **v** e outro chamado de **V**. Note que para o R, o **v** minuscuro e **V** maiusculo sao coisas diferentes. Muitos erros acontecem por causa desse pequeno detalhe. Vamos realizar essas atribuicoes com o codigo abaixo:

```
v <- c(2,4,10)
V <- c(4,8,20)
```

Note que para fazer uma atribuicao de muitos valores, eles precisam estar entre parenteses, separados por virgula antecidos do **c**.

## 3.2 Objetos, vetores e funções

Dando nome aos bois! Entender que cada parte e cada coisa que voce faz possui uma nomenclatura, ajuda voce a se comunicar com outros, seja lendo um livro, seja assistindo um video ou conversando com um colega sobre R. Entao vamos entender alguns conceitos.

### 3.2.1 Objetos

Quando atribuimos o numero **1** a letra **a**, a gente criou um objeto de nome **a**. Abaixo mais alguns exemplos de objetos.

```
objeto_a <- 10
objeto_b <- 20
melhor_comida <- "Coxinha de frango!"
Brasil <- "Pentacampeao!"
Marilia <- "Troxa!"
```

Note que eh possivel atribuir letras, caracteres ou frases a um objeto. Mas eh necessario que o conteudo esteja entre aspas.

Toda vez que voce executa o objeto, o R ira retornar seu valor correspondente. Execute os objetos um por um e veja o resultado.

```
objeto_a
objeto_b
melhor_comida
Brasil
Marilia
```

### 3.2.2 Vetores

Os objetos com multiplos valores de um mesmo tipo sao chamados de vetores.

```
vetor.A <-c (10, 20, 30)
vetor.B <-c (2, 2, 2)
melhores.comidas <-c ("coxinha","pastel","lasanha")

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

### 3.2.3 Funções

No R, as funcoes servem para processar de alguma forma os dados. No codigo abaixo, vamos criar dois objetos e utilizar a funcao de adicao para soma-los.

```
#Objetos
a <- 2
b <- 8
vetor <- c(5,5,10)

#Funcao de soma
sum(a,b)
sum(vetor)
```

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.

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(vetor) {  
  
  media    <- mean(vetor)  
  mediana  <- median(vetor)  
  desvio   <- sd(vetor)  
  
  resultado <- c(media, mediana, desvio)  
  print(resultado)  
}
```

Calma, tem muita coisa aí, mas é tudo o que a gente já sabe. Na função `my_summary()`, precisamos inserir apenas um argumento do tipo vetor, como por exemplo, “notas”. Mais uma vez, as chaves que abre e fecha contêm tudo o que a gente quer que a função realize, linha por linha. Primeiro, queremos que a nossa função calcule a média 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 função `print()`.

Vamos usar nossa função personalizada para calcular nosso objeto “notas”.

```
#Usando a função my_function()  
  
my_function(notas)
```

Veja que a função retornou os valores do nosso vetor, mas apenas isso. Mais para frente, a gente vê como criar funções mais elaboradas.

## 3.3 Tabela de dados

### 3.3.1 Data frame

Data frame 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 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",
"Fiuzza", "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, utilizando pacotes como 'dplyr' e 'ggplot2', para manipulacao de dados e construaao de graficos, respectivamente.

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)
```

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



```
summary(tabela$notas)
```

### 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 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 de 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 reprodutíveis 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.

## 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)
```

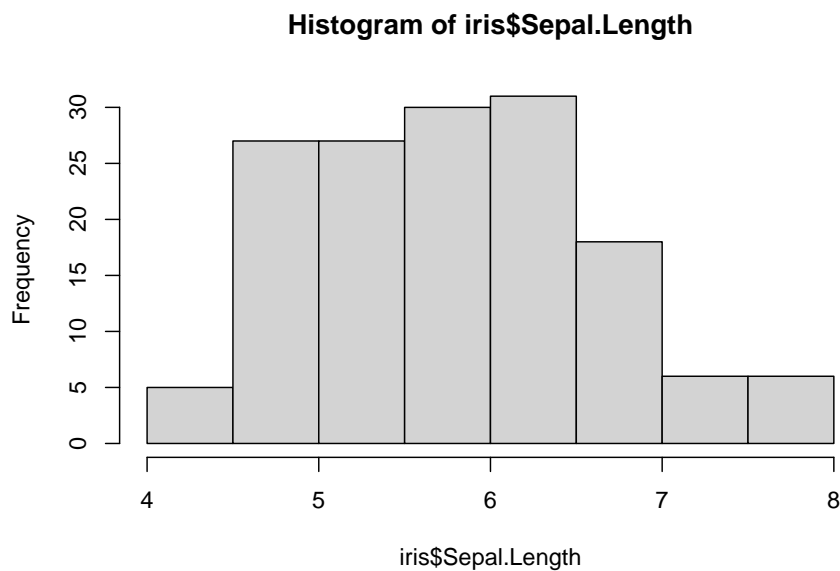


Figure 3.1: Gráficos com R base

Simple 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 Gráfico de dispersão

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)
```

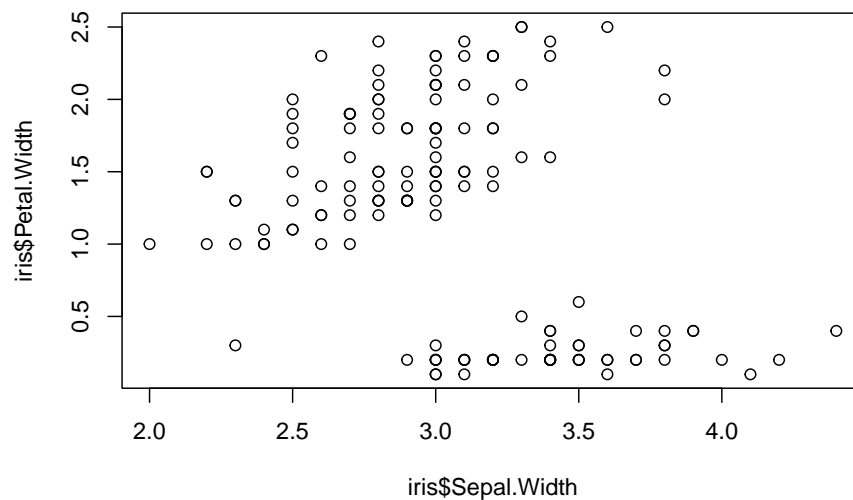


Figure 3.2: Gráficos com R base

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

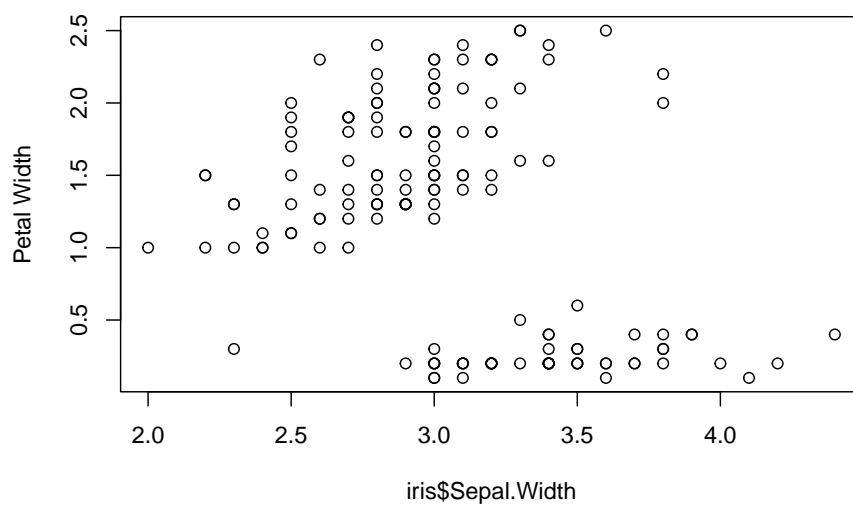


Figure 3.3: Gráficos com R base

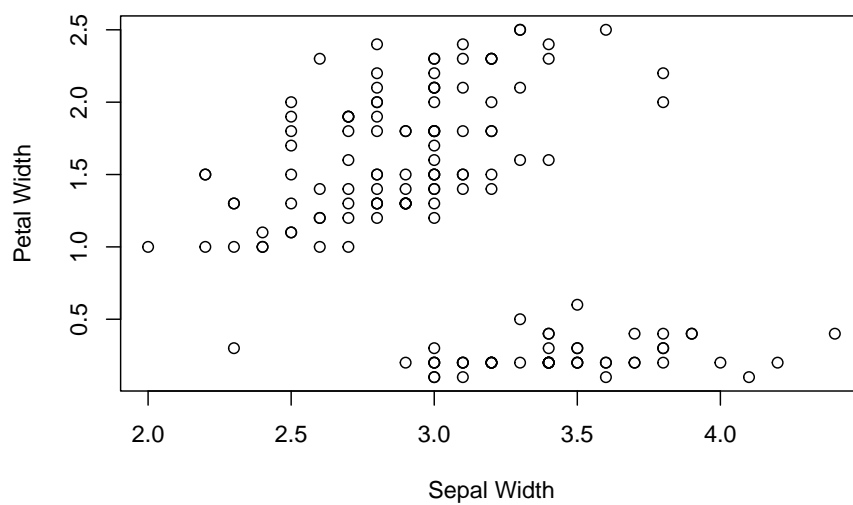


Figure 3.4: Gráficos com R base

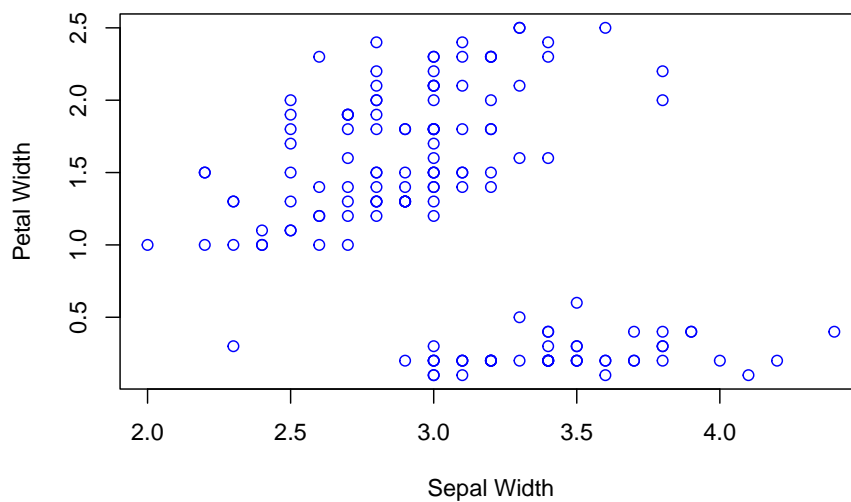


Figure 3.5: Gráficos com R base

```
#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"  
)
```

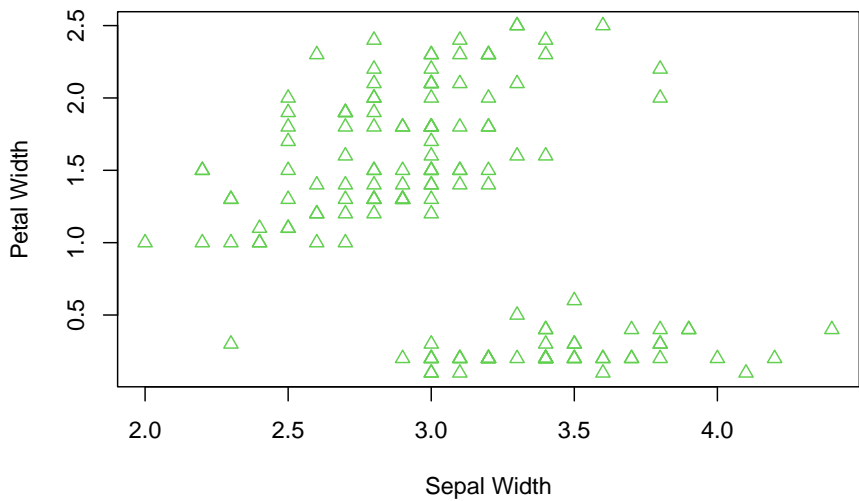


Figure 3.6: Gráficos com R base

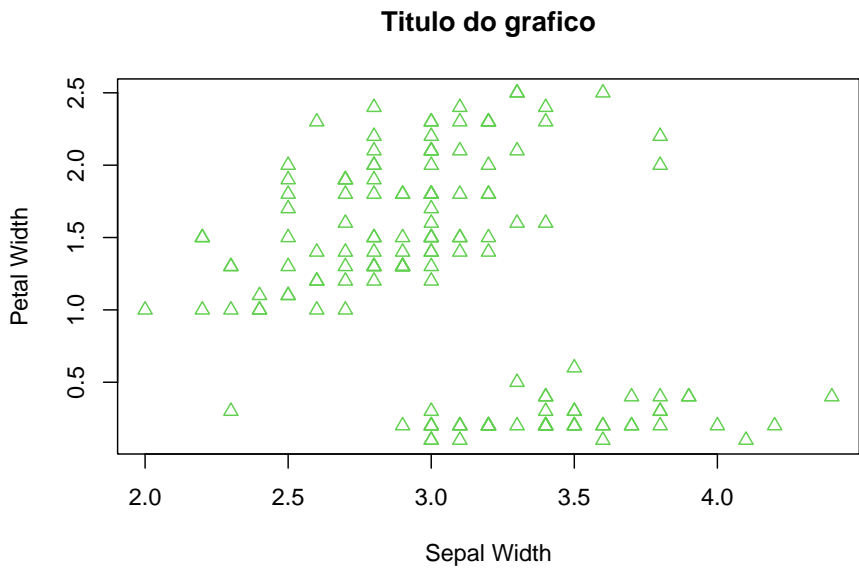


Figure 3.7: Gráficos com R base

### 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)
```

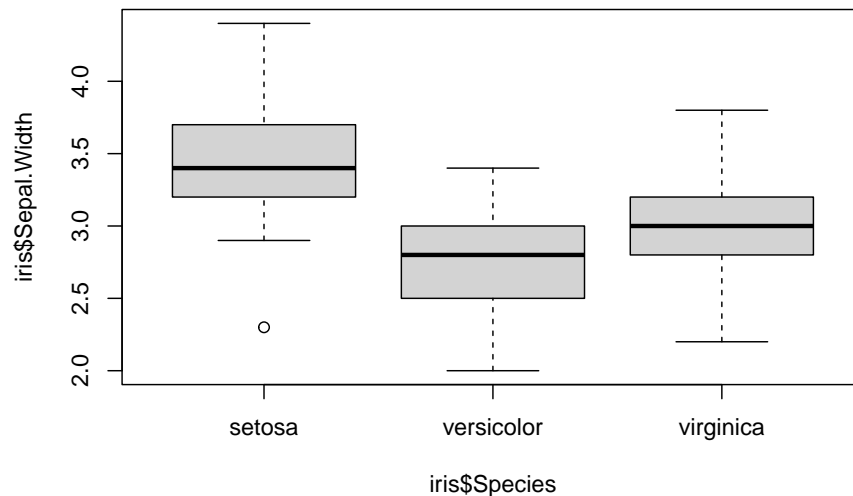


Figure 3.8: (#fig:3\_basic\_boxplot-1)Boxplot com R base utilizando o banco de dados iris

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



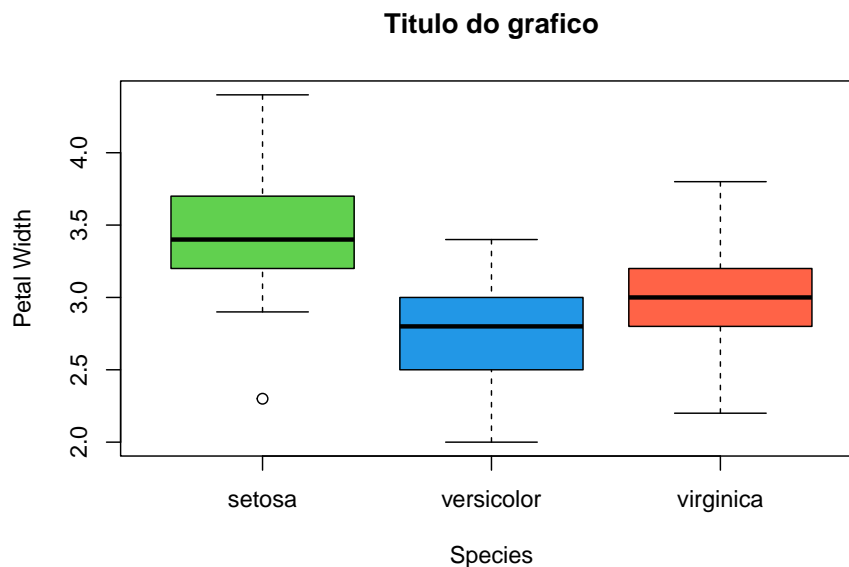


Figure 3.9: (#fig:3\_basic\_boxplot-2)Boxplot com R base utilizando o banco de dados iris

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.

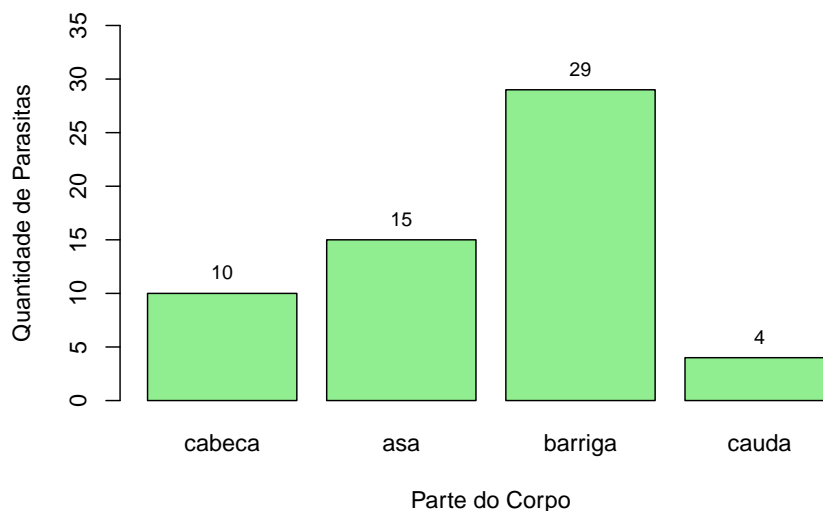
```
#Criando quantidade total de parasitas
qt_parasita <- c(10, 15, 29, 4)

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

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

```
# Criando 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 começar 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 começar 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()`.

```
#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.

```
#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")
```

```
# 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, reproduzível.

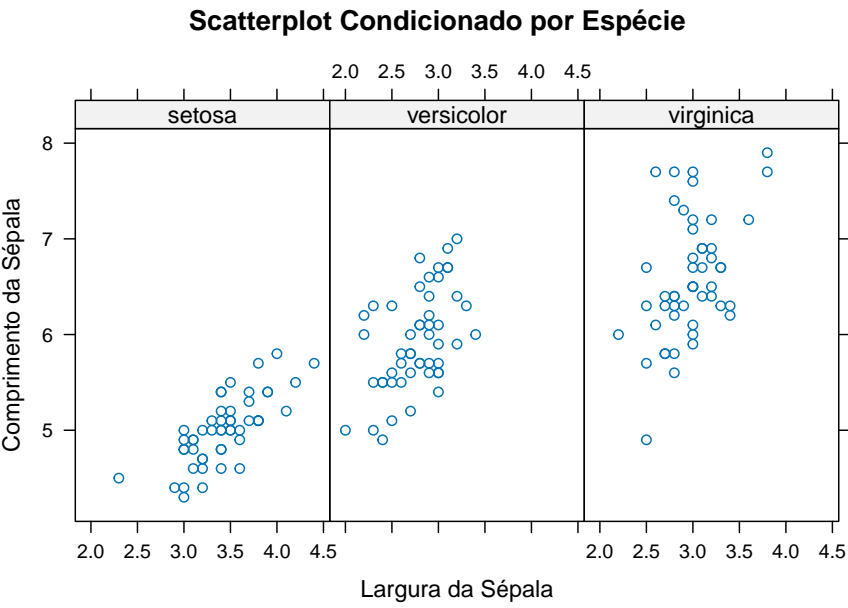


Figure 3.10: Gráficos com Lattice

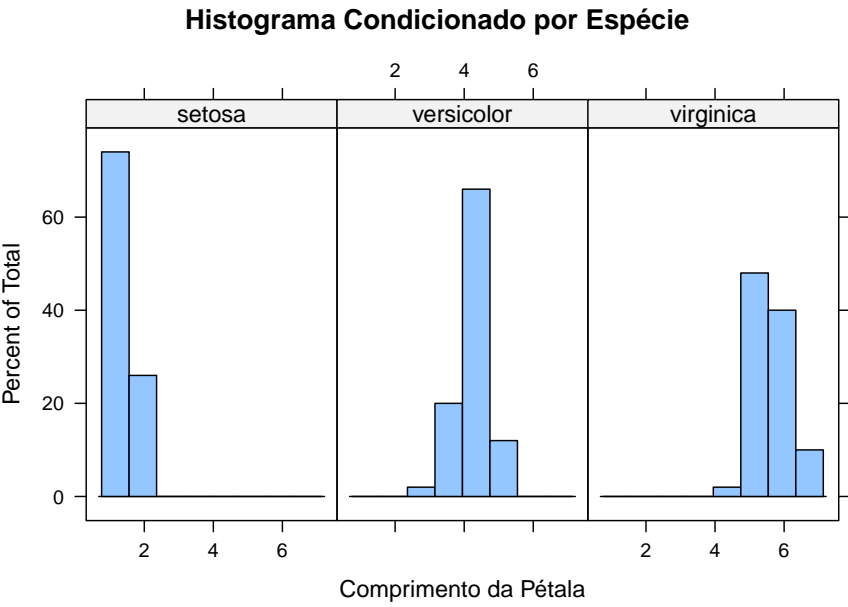


Figure 3.11: Gráficos com Lattice



## 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 analisar a quantidade de parasitas de aves da mata atlantica, ou o comportamento sexual de caranguejos chamameres (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 n
```

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`.

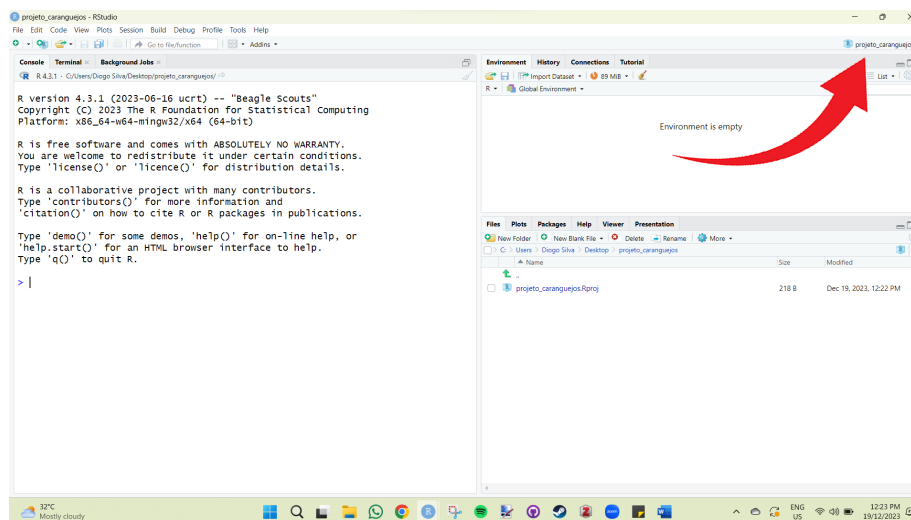
## 4.1 Criando um projeto de R

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”.

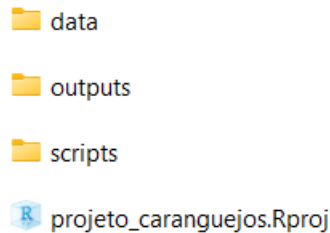
```
install.packages("here")  
library(here)
```

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 estão assim:



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

Na pasta **data > raw** você adiciona sua planilha de dados brutos. Para aqueles que não possuem nenhum dados, utilize a planilha “iris”. 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 nome das variáveis, 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 códigos que irei fornecendo a seguir copiando e colando no seu RStudio. Assim, a primeira coisa que iremos fazer do script é seu cabeçalho. Utilizaremos os comentários para criar um cabeçalho com as informações de título, 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 facéis 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 facéis de trabalhar, deixando todas as letras minusculas.



```

                                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 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 básico.

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

Vamos realizar uma análise para comparar a largura das pétalas entre as espécies. 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(iris$Sepal.Width ~ iris$Species)  
kw
```

Vamos salvar o resultado em uma tabela organizada (tidy) utilizando a função `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! 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 gráfico

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

```
# Script to load and analyse processed data iris
# Author: Marilia T. Ericsson
# 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)
leaves(dados$species)

# 2. Criar gráfico boxplot -----

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

grafico

# 3. Salvando gráfico -----

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

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: <https://rkabacoff.github.io/datavis/> e <https://r-graphics.org/>

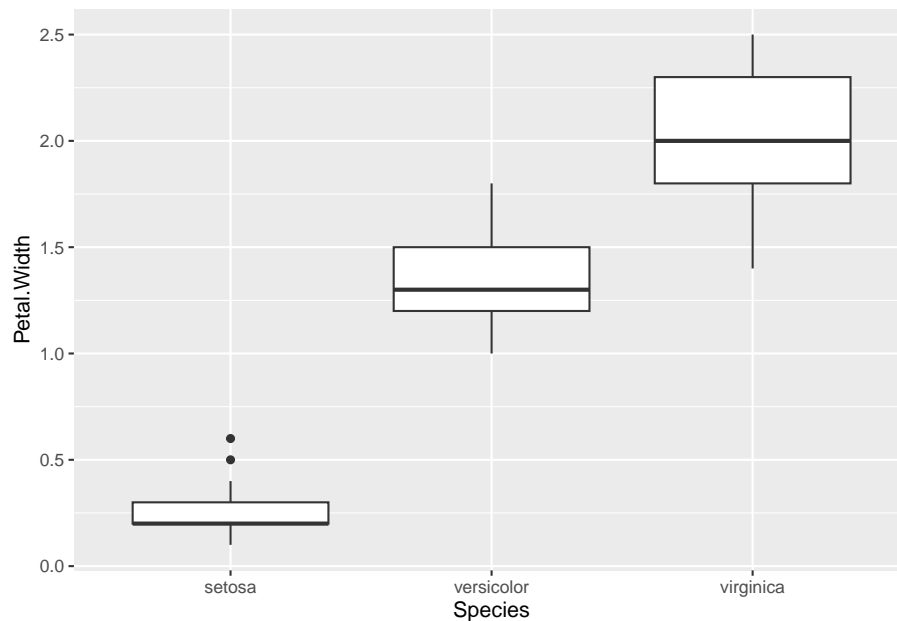


Figure 4.1: ggplot2

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-----
# 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 código, todos os seus scripts são rodados e todos os seus outputs e planilhas processadas são geradas. Mas lembre, seu dado bruto contido na pasta data/raw, não pode ser deletada ou modificada.

Salve esse script com o nome “run\_project”.



## Chapter 5

# Git e GitHub