

ENG 792 - Análise e visualização de dados com R (RStudio)

Jackson Rodrigues

2021-08-16

Contents

Chapter 1

Apresentação

1.1 Hello World! - Sejam Bem Vindos!

Assista este conteúdo em **Cap 1 - Hello World! - Sejam Bem Vindos!** no PVANet.



Este é o material para os alunos matriculados na disciplina **ENG 792 - Análise e Visualização de dados com R (RStudio)** oferecida pelo programa de pós-graduação em Meteorologia Aplicada da Universidade Federal de Viçosa (UFV).

Este curso foi criado e é ministrado por Jackson Rodrigues, professor da Universidade Federal Fluminense (UFF), mas também professor no programa de pós-graduação em Meteorologia Aplicada do Departamento de Engenharia Agrícola (DEA) da Universidade Federal de Viçosa (UFV).

Este curso foi construído pensando em minha saga para aprender algo utilizando o R em meu doutorado. Foi bastante dolorido, principalmente no começo quando eu nunca havia tido contato com a linguagem, na verdade, sabia quase nada em programação.

Desta forma, o público alvo é aquele que tem conhecimento **zero** sobre o assunto. Mas se você já sabe algo em qualquer nível, seja bem vindo também! Podemos aprender juntos, você ensina o que você sabe e eu ensino o que eu sei.

A ideia principal deste curso é que o(a) aluno(a) que nada sabe de **R** seja exposto(a) a grande quantidade de conceitos e métodos para que o “gelo” seja quebrado e possa se virar sozinho ao término do curso. Desta forma, não é um curso de estatística, não é um curso de produção gráfica nem um curso produção de conteúdo (eg. modelagem), mas um curso de exposição, quase um “*how to*”. Vamos levantar demandas do dia a dia e tentar resolver os problemas comuns enfrentados pelo estudante na condução de suas análises.

Eventualmente algum especialista será convidado para dar uma aula, bater um papo, fazer uma apresentação e etc sobre um assunto específico.

O R é uma ferramenta espetacular para análise e produção de dados, bem como produção gráfica! Vamos começar devagar e ao poucos vocês vão se tornando autossuficientes. Muitas dúvidas surgem em fóruns, grupos e tanto outros canais de comunicação da comunidade de usuários de R.

Muitas vezes surgem perguntas como: ”**Tem como fazer isso no R?**” A resposta imediada, devido à sua versatilidade, é: **A pergunta não é “se tem jeito”, mas “como fazer”.**

Vamos dar uma passeio por muitas coisas legais que o **R** pode fazer e que vão ajudar a tornar sua vida bem melhor. Com tempo ganho na otimização das suas tarefas por utilização do **R** sobrará tempo para você escrever um paper a mais na pós-graduação, dormir no final de semana sem culpa, tomar uma cerveja na sexta com a turma e ir ao culto domingo sem pedidos.

Vamos fazer tudo isso nos divertindo. É legal! Vocês verão. Por isso não se preocupem comigo (professor), será muito pior para mim por ter que corrigir trabalhos enquanto vocês produzem mais um paper, dormem, tomam um cerveja com a turma e/ou rezam como foi na véspera de natal de 2020 quando estava corrigindo trabalhos.

Let's ride to Metal Land!

Não se sintam pressionados ou intimidados, façam as perguntas que quiserem. Eu também não sei tudo, o R é uma ferramenta que está em constante desenvolvimento tornando impossível acompanhar cada novidade. Caso eu não saiba uma resposta não tenho problemas em dizer que não sei, mas vou me esforçar para buscar a resposta.

Por isso:

- Vamos aprender e tentar nos divertir;
- Pensei nesse curso como uma forma de lutar contra a “dor” que senti quando comecei a trabalhar no R sozinho;
- Então sou um usuário e não um programador;
- A verdade é que o R tem uma curva de aprendizado muito íngreme que uma vez vencido o primeiro obstáculo as coisas deslancham;

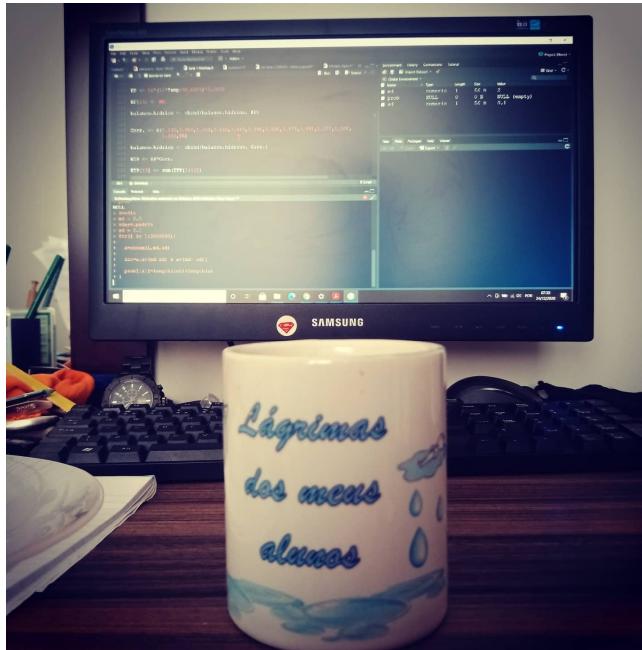


Figure 1.1: Véspera de Natal 2020

- Vamos trabalhar de maneira que esse curso atenda superficialmente suas demandas e que te habilitem a se virarem sozinhos;
- Então vamos seguir passo-a-passo para um aprendizado gradual com vários exemplos;
- Tudo que você aprender em determinado momento não será descartado, você deverá guardar aquele conhecimento para utilização em próximo trabalho ou tarefa;
- Ou ainda, este conhecimento inicial será utilizado para sedimentar o caminho para um próximo passo;
- Aplicar o máximo possível nosso conhecimento a problemas reais, do mundo real.

Para finalizar gostaria de deixar algumas coisas claras.

- Este curso conta com vasto material encontrado em artigos, livros, blogs especializados, grupos de discussões e etc. Desta forma, caso encontre por aí algo que ofertei em aula não precisa me chamar de picareta. Provavelmente foi tirado de lá mesmo.
- O curso funciona como um **How to**. Não teremos tempo de nos aprofundar nas teorias dos assuntos aqui apresentados (nem é a intenção), por isso,

vou mostrar o “que é”, “como usar” e “como fazer”.

- Ao término de cada aula será mostrada uma bibliografia básica sobre o conteúdo. Como são conteúdos diversificados, acho melhor separar as bibliografias por aula.

1.1.1 Sobre

Este é um material de apoio compilado e criado para os alunos da disciplina **ENG 792 - Análise e visualização de dados com R (RStudio)**.

Importante mencionar que o conteúdo aqui apresentado é um compilado de vários anos de materiais estudados disponíveis *online* ou em livros e artigos especializados. Desta forma, caso identifique algum conteúdo apresentado aqui que não esteja devidamente referenciado fique à vontade para solicitar os devidos créditos aos autores originais. Não tenho a intenção de ter crédito que não é meu.

1.1.2 Utilização

Cada capítulo é referente ao conteúdo de mais ou menos uma semana de curso. Cada semana trata de um assunto diferente e complementar ao conteúdo da semana anterior. Desta forma, fique livre para ir e vir no conteúdo caso algo não esteja claro o suficiente.

- Teremos aulas gravadas e aulas síncronas através de alguma plataforma.
- Apenas os alunos matriculados na disciplina terão acesso ao conteúdo gravado.
- A distribuição e/ou compartilhamento do conteúdo gravado por qualquer meio é proibido.

1.1.3 Códigos e Dados

Todo o material necessário para acompanhar a disciplina será oferecido via github. Os códigos com alguma explicação neste material online (explicação completa nas aulas) e os dados onde forem possíveis de serem armazenados.

1.1.4 Cronograma de aulas

As aulas serão ofertadas por material gravado e presencial às quintas e sextas entre 14:00 e 15:30.

Semana	Conteúdo
1.1	Apresentação do Conteúdo e Instalação do R
1.2	Funcionamento do R, tipo e estrutura dos objetos

Semana	Conteúdo
2.1	Manipulação de dados 1
2.2	Manipulação de dados 2, testes lógicos e simbolos, condicionais e interações
3.1	Pacotes e Funções
3.2	Entrando/Importando dados. definição de diretórios
4.1	Produção gráfica com Rbase
4.2	Produção gráfica com ggplot 2
5.1	Elementos de estatística básica 1 (Est. Descritiva, Med. Tend. Central, Medi. de variabilidade)
5.2	Elementos de estatística básica 2 (Dife. Entre médias, T-Student, Teste F, Testes de normalidade)
6.1	Elementos de estatística básica 3 (ANOVA, delineamento, comparação múltiplas, Regressão, Resíduos, Homocedasticidade, Normalidade dos resíduos, regressão múltipla, superfície de resposta)
7.1	Análise Multivariada (Análise de agrupamento, medidas de (dis)similaridade, métodos de conexão, número de clusters, produção de dendogramas, árvores de decisão)
7.2	Métodos de Ordenação (Ana. Componentes Principais, Análise Canônica, Análise de fatores, Análise de Mahalanobis)
8 - 15	Será preenchido em breve

1.1.5 Métodos de avaliação

- Faremos pelo menos 3 listas de exercícios distribuídas pelo semestre.
- O trabalho final será a confecção de um atrabalho autoral com o conteúdo do curso que poderá ser feito em grupo (isso será definido ainda em conversa com vocês). Este trabalho será apresentado na forma de seminário ao término do curso.

Chapter 2

Vamos ao que interessa

2.1 Conhecendo o R

2.1.1 O que é o R?

É uma linguagem de programação voltada para resolução de problemas estatísticos, tratamento e visualização de dados.

Para ? essa resposta é simples, “*R é um dialeto do S*”.

De acordo com ? *O código base do R foi inicialmente criado no laboratório da Bell/ AT& T por John Chambers e seus colegas, com base na linguagem S. Esse código foi reaproveitado por dois acadêmicos, Ross Ihaka e Robert Gentleman, resultando na plataforma de programação que temos hoje. Para os curiosos, o nome R foi escolhido devido ao compartilhamento da primeira letra do nome de seus criadores.*



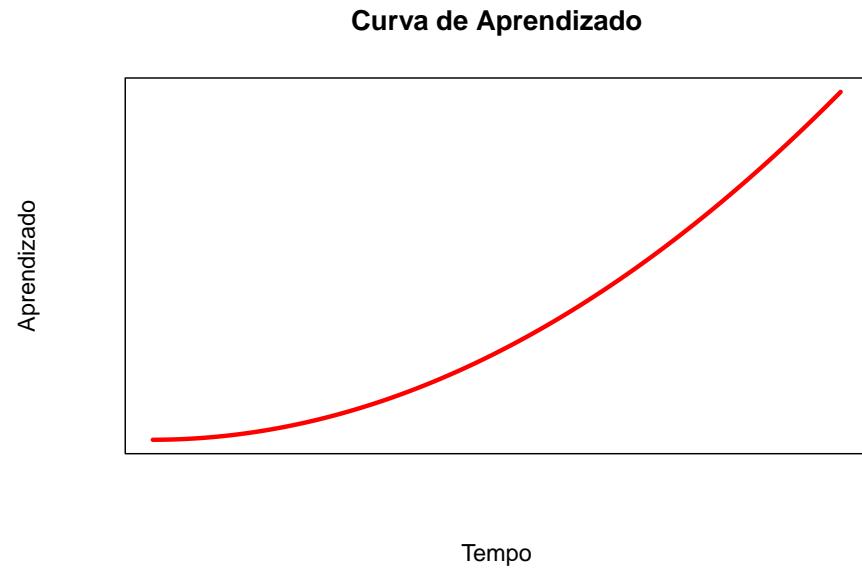
O R está em constante desenvolvimento por um grupo chamado R Team Core e conta com colaboração gratuita de centenas de milhares de usuários e desenvolvedores ao redor do mundo. Por isso, atualmente o R é utilizado por diversas áreas do conhecimento variando das ciências humanas até exatas, naquelas ciências que poderíamos imaginar pouco ou nada relacionadas. Por isso não se limite a procurar informações apenas no seu nicho, abra sua mente e busque aprender

de outras ciências também. Eu, particularmente, busco muita coisa na econometria. Embora presente em todo tipo de livro sobre R, esta citação acima (?) é de um livro de econometria. Veremos mais conteúdos desse material em breve.

R é um software livre de análise de dados (não só estatística) que funciona em diversos sistemas operacionais: GNU Linux, Microsoft Windows, Mac OS X e outros.

O aprendizado do R é difícil no início devido à necessidade de se adaptar à sua lógica de funcionamento, se acostumar com a estrutura dos seus documentos de ajuda e memorizar alguns comandos básicos.

```
eq = function(x){x*x}
plot(eq(1:1000), type="l", lwd=3, col="red", xaxt="n", yaxt="n", xlab="Tempo", ylab="Aprendizado")
```



É preciso bastante perseverança e motivação para aprender os comandos básicos, e disposição para ler as páginas de ajuda e os manuais. Entretanto, depois de um certo tempo, ele possibilita que se trabalhe com grande produtividade e, o que é mais importante, eficácia e independência.

Leia também sobre o mito da curva de aprendizado do R.

2.1.2 Instalação do R

O **R** é um software gratuito para análises estatísticas e além. Pode ser baixado de The R Project for Statistical Computing.

Clique em **download R**.



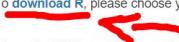
[Home]

Download[CRAN](#)**R Project**[About R](#)[Logo](#)[Contributors](#)[What's New?](#)[Reporting Bugs](#)[Conferences](#)[Search](#)[Get Involved: Mailing Lists](#)[Developer Pages](#)[R Blog](#)

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOs. To [download R](#), please choose your preferred CRAN mirror.



If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News

- [R version 4.1.1 \(Kick Things\) prerelease versions](#) will appear starting Saturday 2021-07-31. Final release is scheduled for Tuesday 2021-08-10.
- [R version 4.1.0 \(Camp Pontanezen\)](#) has been released on 2021-05-18.
- [R version 4.0.5 \(Shake and Throw\)](#) was released on 2021-03-31.
- Thanks to the organisers of useR! 2020 for a successful online conference. Recorded tutorials and talks from the conference are available on the [R Consortium YouTube channel](#).
- You can support the R Foundation with a renewable subscription as a supporting member

Escolha o “espelho”. Escolha o mais próximo de você.

CRAN Mirrors

The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here: [main page](#), [windows release](#), [windows old release](#).

If you want to host a new mirror at your institution, please have a look at the [CRAN Mirror HOWTO](#).

0-Cloud	https://cloud.r-project.org/	Automatic redirection to servers worldwide, currently sponsored by Rimado
Algeria	https://cran.utlib.dz/	University of Science and Technology Houari Boumediene
Argentina	https://mirror.fcylq.unlp.edu.ar/CRAN/	Universidad Nacional de La Plata
Australia	https://cran.csiro.au/ https://mirror.csiro.au/pub/CRAN/ https://cran.ms.unimelb.edu.au/ https://cran.curta.edu.au/	CSIRO AARNet School of Mathematics and Statistics, University of Melbourne Curtin University
Austria	https://cran.wu.ac.at/	Wirtschaftsuniversität Wien
Belgium	https://www.freestatistics.org/cran/ https://fb.belnet.be/mirror/CRAN/	Patrick Wessa Belnet, the Belgian research and education network
Brazil	https://ibc.ribmes.br/mirror/cran/ https://cran.cict.ufsc.br/ https://cran.fioem.br/ https://vs.fmv.unesp.br/CRAN/ https://tricper.eads.upr.br/CRAN/	Computational Biology Center at Universidade Estadual da Santa Cruz Universidade Federal do Piauí Oswaldo Cruz Foundation, Rio de Janeiro University of São Paulo, São Paulo University of São Paulo, Piracicaba
Bulgaria	https://fis.uni-sofia.bg/CRAN/	Sofia University
Canada	https://mirrcv.sfu.ca/mirror/CRAN/ https://tanrc.ca/mirror/cran/ https://mirrcv.tld.ca/cran/	Simon Fraser University, Burnaby Manitoba Unix User Group Dalhousie University, Halifax

Escolha o seu sistema operacional. Caso você seja usuário de windows clique em **Download R for Windows** em seguida em **install R for the first time** e finalmente em **Download R 4.1.0 for Windows**. Veja que no momento que este tutorial foi feito a versão mais recente é a 4.1.0. No vídeo abaixo a versão é uma anterior, mas a lógica é a mesma.

The Comprehensive R Archive Network

[Download and Install R](#)

Precompiled binary distributions of the base system and contributed packages. **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux \(Debian, Fedora, Redhat, Ubuntu\)](#)
- [Download R for macOS](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

[Source Code for all Platforms](#)

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2021-05-18, Camp Pontanezen) [R-4.1.0.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

[Questions About R](#)

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

Assista este conteúdo em **Cap 1 - Instalação do R no Windows** no PVANet.

Eu não tenho um sistema operacional de cada para mostrar a instalação, por isso deixo este vídeo para instalação no linux e este para instalação no mac. Caso você não consiga instalar me procure.

2.1.3 Primeiro contato

Assista este conteúdo em **Cap 1 - Primeiro contato** no PVANet.

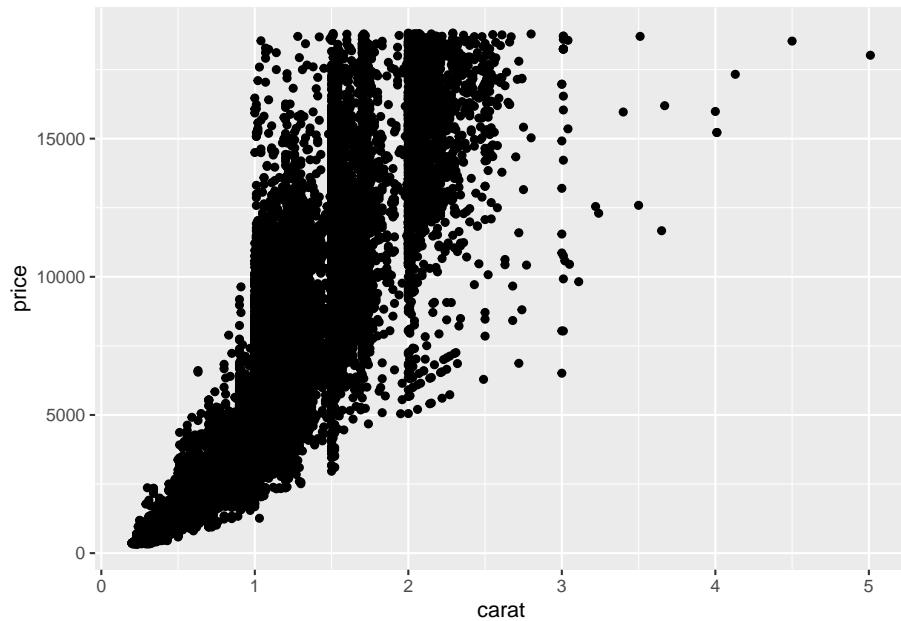


Temos 1 arquivo **Cap_1_P2-Mispriced-Diamonds.csv** PVANet no PVANet com mais de 50.000 linhas referentes a transações de venda de diamantes dividida em 3 colunas *clarity*, *carat* e *price*. Quanto mais claro mais caro, certo? Ou há sub ou super valorização? Vamos investigar se essa relação é verdadeira como sugerido em ecapitaladvisors.

Abaixo vamos apenas dar uma olhada no potencial de análise e produção gráfica do R. Não precisa se preocupar se não conseguir fazer tudo funcionar. Vamos aos poucos aprender cada comando apresentado.

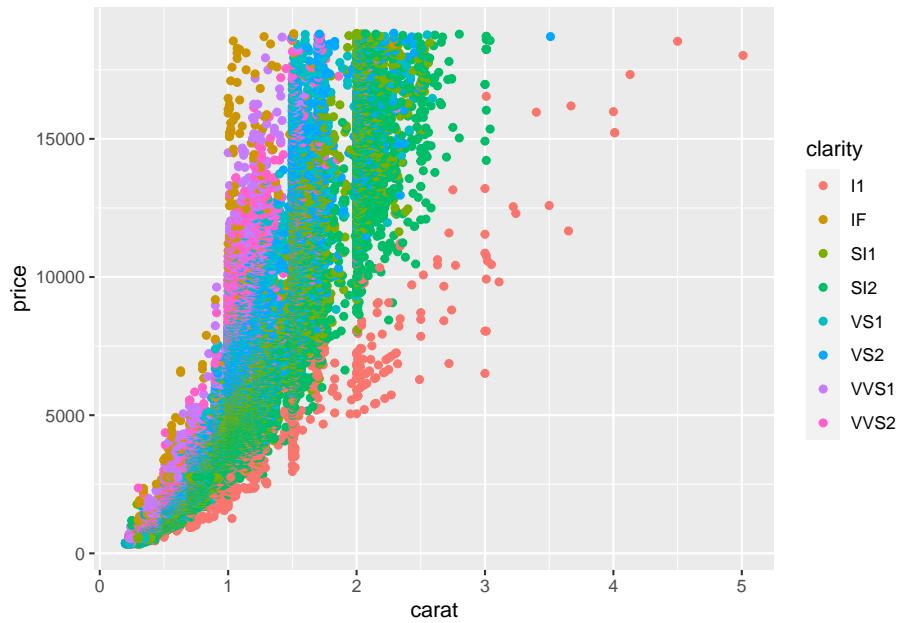
```
mydata<-read.csv("J:/ENG 792/ENG_792-AVDR/ENG.792-AVDR/Cap_1_P2-Mispriced-Diamonds.csv")
library("ggplot2")
```

```
ggplot(data=mydata, aes(x=carat, y=price))+  
  geom_point()
```



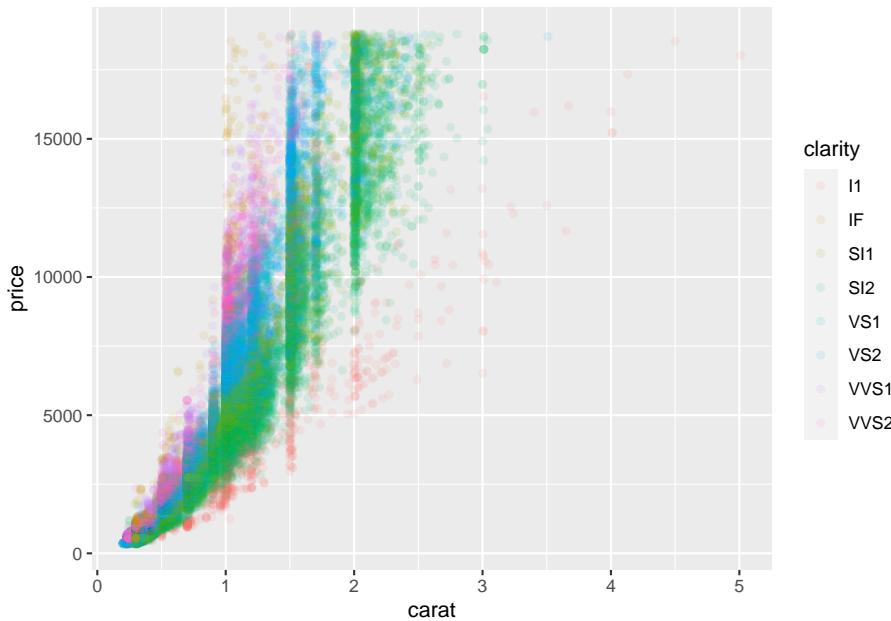
Faz algum sentido. Mas está difícil de visualizar então vamos fazer um tratamento, vamos atribuir cores de acordo com a classificação do atributo *clarity*.

```
ggplot(data=mydata, aes(x=carat, y=price, colour=clarity))+  
  geom_point()
```



Os pontos estão sobrepostos impossibilitando a visualização. Vamos mexer na transparência.

```
ggplot(data=mydata,  
       aes(x=carat, y=price, colour=clarity))+  
  geom_point(alpha=0.1)
```



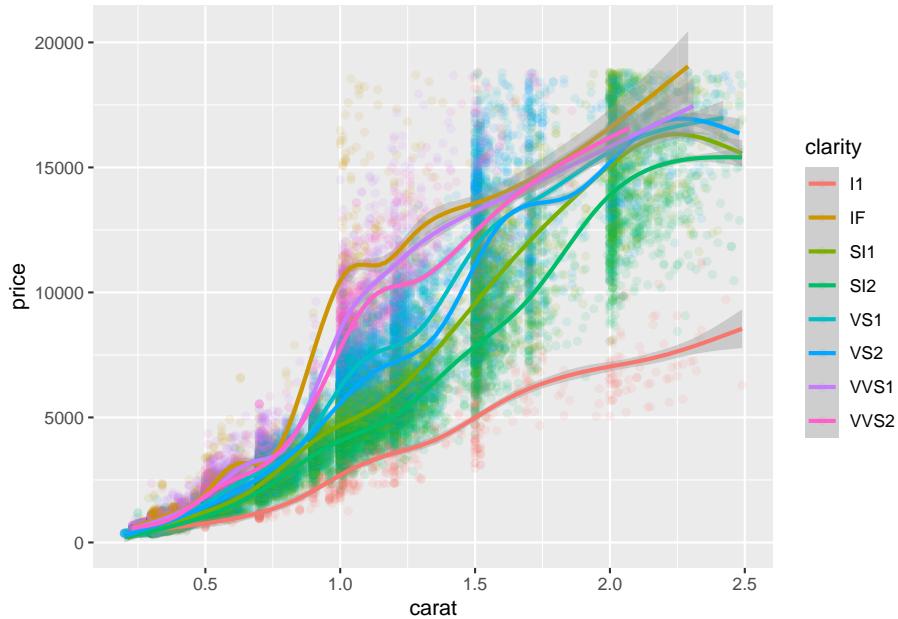
Temos pontos que não são estatisticamente significantes à direita. Vamos nos livrar dos pontos não significativos, aqueles que são *carat* menores que 2.5.

Vamos também adicionar algumas linhas através dos dados para avaliar o comportamento das variáveis em conjunto.

brown é a melhor claridade, vejam que temos mispricing onde as linhas se cruzam.

```
ggplot(data=mydata[mydata$carat<2.5],
       aes(x=carat, y=price, colour=clarity))+  
  geom_point(alpha=0.1) +  
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Vamos agora reproduzir um ourto exemplo muito legal do pacote `rayshader`.

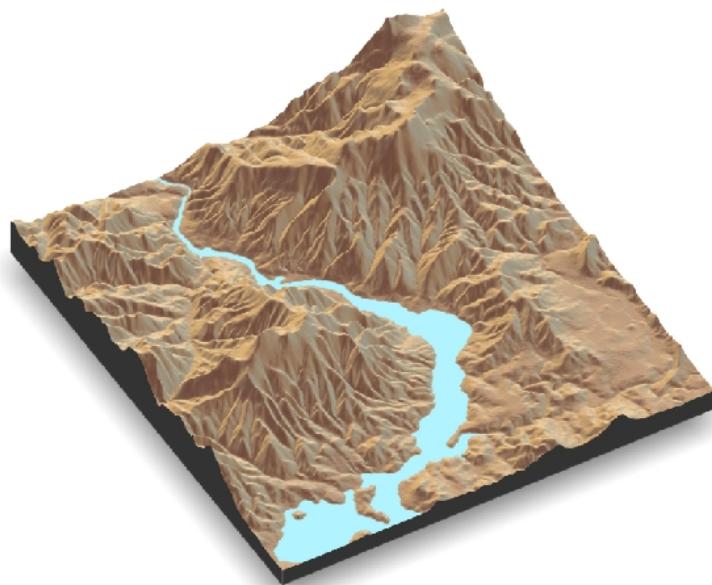
Executando o código abaixo você produzirá uma Modelo Digital do Terreno em 3D em uma janela **pop up**.

```
library(rayrender)
library(rayshader)
library(magick)

#Vamos carregar o mapa com o pacote raster.
loadzip = tempfile()
download.file("https://tylermw.com/data/dem_01.tif.zip", loadzip)
localtif = raster::raster(unzip(loadzip, "dem_01.tif"))
unlink(loadzip)

# convertê-lo para matriz:
elmat = raster_to_matrix(localtif)

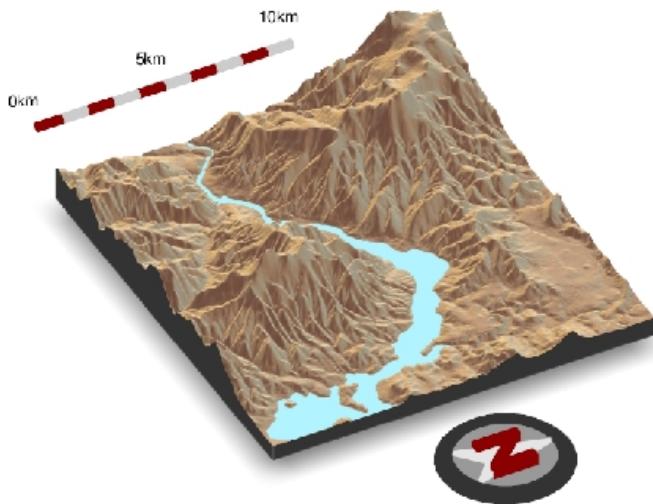
#Usar uma outra textura do rayshader
elmat %>%
  sphere_shade(texture = "desert") %>%
  add_water(detect_water(elmat), color = "desert") %>%
  plot_3d(elmat, zscale = 10, fov = 0, theta = 60, zoom = 0.75, phi = 45, windowsize =
```



Agora vamos adicionar mais algumas informações como escala e indicador de norte.

```
render_scalebar(limits=c(0, 5, 10),label_unit = "km",position = "W", y=50,scale_length = c(0.33,10))

render_compass(position = "E")
Sys.sleep(0.2)
render_highquality(samples=200, scale_text_size = 24,clear=TRUE)
```



2.2 Como R funciona

Assista este conteúdo em [Cap 1 - Como R Funciona](#) no PVANet.

Diferentemente de outras linguagens, todos os comandos escritos são diretamente executados, desta forma o R não precisa de um compilador para executar os comandos como Fortran. Por isso, torna-se uma linguagem muito mais amigável e acessível para não programadores.

A linguagem é muito intuitiva (quase uma sintaxe lógica). Por exemplo uma regressão linear pode ser executada como `lm(x~y)` (`lm` vem de linear model). Como no exemplo do modelo linear acima, sempre que formos executar um comando temos que seguir da seguinte forma função(dados e demais ajustes ou parâmetros), ou seja chame a função e coloque o resto dentro de parênteses.

Tudo que é executado pelo R fica armazenado na memória ativa (RAM) do computador na forma de objetos que possuem um nome. Os objetos, que variam em tipos e estruturas, podem ser funções criadas pelo próprio usuário, dados criados ou importados de uma memória, expressões e etc. Antes de entrarmos em detalhes sobre funções ou expressões, vamos nos ater aos objetos enquanto tipo e, na sequência, suas estruturas.

Dica de livro de cabeceira sobre R ?.

Antes de avançarmos para os objetos é necessário fazer algumas recomendações e ressalvas.

Para criar um objeto qualquer podemos utilizar `<-` ou `=`. As boas práticas recomendam o uso de `<-` para evitar confusão já que o sinal `=` tem outras funções.

Um objeto é criado da seguinte forma `nome_do_objeto <- atribuição`.

```
nome_do_objeto <- "atribuição"
nome_do_objeto
```

```
## [1] "atribuição"
```

Contudo, evitem nomes longos, que comecem com numerais ou caracteres especiais ou letras maiúsculas (R é case-sensitive). Caso sua linha de comando esteja ficando longa demais opte por quebrar a linha veremos isso mais adiante).

2.2.1 Tipos de objetos (*mode* ou *type*)

Os objetos no R podem ser do tipo *lógico*, *inteiro*, *simples*, *dupla*, *complexo*, *função* ou *caractere*.

mode()	Armazenamento	Exemplo
logical	lógico	TRUE or FALSE
numeric	inteiro, simples ou dupla	Números 1, 3.14, 2e-308 etc
complex	complexo	3+2i
function	função	Soma<-function(...)
name	caractere	média

Tabela 1: Tipos de modos para objetos no R.

```
#logical
q1 <-T
mode(q1);typeof(q1)

## [1] "logical"

## [1] "logical"
q2 <- FALSE #pode ser a palavra toda mas em maiúsculas
mode(q2);typeof(q2)

## [1] "logical"

## [1] "logical"
#integer
x<-2L #L garante que 2 será integer
mode(x);typeof(x)

## [1] "numeric"

## [1] "integer"
```

```

#double
y<-2.5
mode(y);typeof(y)

## [1] "numeric"

## [1] "double"

#Complex
z<-3+2i
mode(z);typeof(z)

## [1] "complex"

## [1] "complex"

#function
Soma<-function(x,y){
  x+y
}
mode(Soma);typeof(Soma)

## [1] "function"

## [1] "closure"

#Character
a <-"h" #Para colocar uma letra em uma variável é preciso colocar entre ""
mode(a);typeof(a)

## [1] "character"

## [1] "character"

média<-"média"
mode(a);typeof(a)

## [1] "character"

## [1] "character"

```

Saber as diferenças entre os diversos objetos é importante para uma exploração mais adequada dos dados, utilização eficiente de funções ou operações lógicas, aritméticas, estatísticas e etc.

Veja que no caso acima em *integer* (*x <- 2L*) optamos por adicionar “L” após o número 2, pois o R por padrão decide onde e como aloca/aloja/armazena um operador. A informação será preferencialmente salva como *double* e isso faz sentido caso você queira mais adiante realizar operações com números decimais ou realizar operações que resultem em números decimais.

No entanto, caso queira saber que tipo de dado está manipulando você pode

“perguntar” utilizando `is.` seguido da designação do tipo de dados quer testar (`integer`, `numeric`, `double` e etc) e teremos uma resposta lógica.

```
is.double(x)
```

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

Caso você deseje que sua variável seja de um tipo específico, você pode transformá-la utilizando `as.` seguido da designação desejada (`integer`, `numeric`, `double` e etc).

```
x<-as.double(x)
is.double(x)
```

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

Cada tipo de dado é associado com um teste e uma função de conversão conforme a tabela 2.

Tipo	Teste	Função de conversão
character	<code>is.character</code>	<code>as.character</code>
complex	<code>is.complex</code>	<code>as.complex</code>
double	<code>is.double</code>	<code>as.double</code>
expression	<code>is.expression</code>	<code>as.expression</code>
integer	<code>is.integer</code>	<code>as.integer</code>
list	<code>is.list</code>	<code>as.list</code>
logical	<code>is.logical</code>	<code>as.logical</code>
numeric	<code>is.numeric</code>	<code>as.numeric</code>
single	<code>is.single</code>	<code>as.single</code>
raw	<code>is.raw</code>	<code>as.raw</code>
Date	<code>is.Date</code>	<code>as.Date</code>

Tabela 2: Tipos de dados, teste e modos de conversão.

2.2.2 Estrutura do objetos (*class*)

As informações armazenadas em objetos no R podem ser organizadas em diferentes estruturas ou classes.

- No R existe uma grande variedade de classes de objetos, e sempre surgindo mais. No entanto, há classes de objetos que são mais comuns em situações em que estamos manipulando bases de dados, seja ela já existente, ou quando estamos criando com coleta usando técnicas de webscraping, por exemplo. São elas: `interger`, `numeric`, `character`, `factor`, `matrix`, `data.frame` e `list`. Uma coisa importante, no R não precisamos declarar qual classe vai ser o objeto unidimensional como em outras linguagem, embora possamos fazer isso, pois, o R aloca o objeto automaticamente em uma classe a partir do seu conteúdo. ?

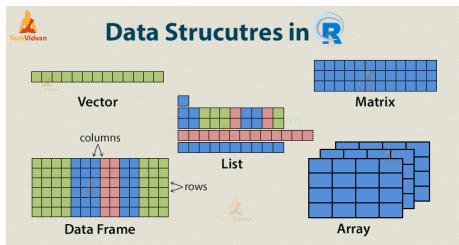


Figura 1: Estrutura de dados no R. Fonte: techvidvan

Objeto	modes	descrição
vector	numeric, character, complex ou logical	Com um ou mais elementos
factor	numeric ou character	Vetor que representa dados categóricos
matriz	numeric, character, complex ou logical	Um array de duas dimensões
array	numeric, character, complex ou logical	Pode conter um, duas ou mais arrays
data frame	numeric, character, complex ou logical	Um array de duas dimensões com estrutura de dados
list	numeric, character, complex, logical, function, expression, ...	Objeto que permite combinar diferentes tipos de objetos

Tabela 3: Características dos tipos de objetos.

2.2.2.1 Vetores (*Vectors*)

Vetores são os tipos de objetos mais comuns no R. Um vetor é composto de uma informação ou uma série de informações (*arrays*) unidimensionais que podem conter informações numéricas, caracteres ou dados lógicos.

Mesmo quando digitamos apenas um único elemento ele se torna um vetor de comprimento um (1).

Vetores com Apenas 1 elemento

```
esquerdo<-("direito") # O objeto "esquerdo" recebe a palavra "direito"
esquerdo #execute o arquivo e veja seu conteúdo
```

```
## [1] "direito"
direito=c("esquerdo") #Outra maneira de criar objeto
print(direito) #Outra forma de executar o conteúdo
```

```
## [1] "esquerdo"
b=(10) # O objeto "b" recebe o número 10
b
```

```
## [1] 10
(15.23)->c # O objeto "c" recebe o número 15.23
c
```

```
## [1] 15.23
```

Vetores com múltiplos elementos.

```
d<-0:10 # Criando uma sequência de 0 até 10
d

## [1] 0 1 2 3 4 5 6 7 8 9 10
e<-10.5:20.5 # criando uma sequência de 10.5 até 20.5
e

## [1] 10.5 11.5 12.5 13.5 14.5 15.5 16.5 17.5 18.5 19.5 20.5
f<-(10.6:20.3) # O último elemento é descartado por não encaixar na sequência
f

## [1] 10.6 11.6 12.6 13.6 14.6 15.6 16.6 17.6 18.6 19.6
```

Podemos utilizar também a função *seq* para gerar uma sequência de dados.

```
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)), length.out = NULL,
along.with = NULL, ...)
```

```
g<-seq(0,10,0.5) # O objeto "g" recebe a sequência de 0 até 10 a cada 0.5
g

## [1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0
## [16] 7.5 8.0 8.5 9.0 9.5 10.0
h<-seq(from=10,to=20,length.out=50) # O objeto "h" recebe a sequência de 0 até 10 que é do comprimento 50
h

## [1] 10.00000 10.20408 10.40816 10.61224 10.81633 11.02041 11.22449 11.42857
## [9] 11.63265 11.83673 12.04082 12.24490 12.44898 12.65306 12.85714 13.06122
## [17] 13.26531 13.46939 13.67347 13.87755 14.08163 14.28571 14.48980 14.69388
## [25] 14.89796 15.10204 15.30612 15.51020 15.71429 15.91837 16.12245 16.32653
## [33] 16.53061 16.73469 16.93878 17.14286 17.34694 17.55102 17.75510 17.95918
## [41] 18.16327 18.36735 18.57143 18.77551 18.97959 19.18367 19.38776 19.59184
## [49] 19.79592 20.00000
```

Experimente também a função *rep()*.

```
rep(x, times = 1, length.out = NA, each = 1)
i<-rep(0,10) # O objeto "i" recebe 10 números 1
i

## [1] 0 0 0 0 0 0 0 0 0 0
j<-rep(c(1:3),10) # O objeto "j" recebe 10 vezes a sequência 1, 2 e 3
j

## [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

Um Vetor só pode conter informações de um único tipo!

```

k<-c(0,1,2,3,4, "A") # O objeto "k" é do tipo character por causa de "A"
typeof(k);mode(k)

## [1] "character"
## [1] "character"

l<-c(0,1,2,3,4) # O objeto "l" é do tipo numérico
typeof(l);mode(l)

## [1] "double"
## [1] "numeric"

```

2.2.2.2 Fatores (*Factors*)

Os fatores são vetores em que os elementos pertencem a uma ou mais categorias temáticas. As variáveis aleatórias podem ser divididas em contínuas e categóricas.

- As contínuas podem ser medidas nas escalas: relacional e intervalar.
- As categóricas nas escalas: nominal e ordinal.

No R, as variáveis categóricas medidas nas escalas nominal e ordinal são chamados fatores. A função *factor()* armazena os valores categóricos como um vetor de inteiros [1..k] e um vetor interno de *strings* referentes ao nomes. *Em outras palavras, um factor é um vetor usado para especificar uma classificação discreta (agrupamento) dos componentes de outros vetores de mesmo tamanho.*

```
factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA)
```

ou

```

gl(n, k, length = n X k, labels = seq_len(n), ordered = FALSE)
m<-factor(c("H", "H", "H", "M", "M")) # O objeto "k" recebe 3 H's e 2 M's
m

```

```

## [1] H H H M M
## Levels: H M
as.integer(m)

## [1] 1 1 1 2 2
n<-gl(n=2,k=3,labels=c("M", "F"))
n

## [1] M M M F F F
## Levels: M F

```

Podemos verificar os níveis de um fator usando o comando *levels()*.

```
levels(m)
## [1] "H" "M"
levels(n)
## [1] "M" "F"
```

2.2.2.3 Matriz (*Matrix*)

É o tipo de dado mais comum que encontramos do dia a dia. A maioria dos dados que analisamos estão organizados em matrizes que são dados combinados em 2 dimensões (linhas e colunas). Existem várias maneiras de criar uma matriz como utilizando o comando *matrix()*.

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
```

Assim como os vetores, as matrizes só aceitam dados do mesmo tipo.

```
o<-1:10 # cria um vetor de 1 a 10
o_matriz1<-matrix(o,ncol=5) # Organiza o vetor "o" e 5 colunas
o_matriz1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]     1     3     5     7     9
## [2,]     2     4     6     8    10
o_matriz2<-matrix(o,nrow=5) # Organiza o vetor "o" e 5 linhas
o_matriz2
```

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

Podemos utilizar também o argumento *byrow*=, que, diferente do exemplo acima, preenche a tabela por linha.

```
p<-1:10 # cria um vetor de 1 a 10
p_matriz1<-matrix(o,nrow=5,byrow=T) # Organiza o vetor "o" e 5 linhas
p_matriz1; o_matriz2 # compare os 2 modos
```

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

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

As dimensões de uma matriz podem ser acessadas através do comando `dim()`:

```
dim(o_matriz1);dim(o_matriz2) # Leia o o resultado como linha e coluna

## [1] 2 5

## [1] 5 2
```

Também é interessante usar o comando `summary()`.

```
summary(o_matriz1);summary(o_matriz2) # Mostra informações por coluna
```

```
##          V1           V2           V3           V4           V5
##  Min.   :1.00   Min.   :3.00   Min.   :5.00   Min.   :7.00   Min.   : 9.00
##  1st Qu.:1.25   1st Qu.:3.25   1st Qu.:5.25   1st Qu.:7.25   1st Qu.: 9.25
##  Median :1.50   Median :3.50   Median :5.50   Median :7.50   Median : 9.50
##  Mean    :1.50   Mean    :3.50   Mean    :5.50   Mean    :7.50   Mean    : 9.50
##  3rd Qu.:1.75   3rd Qu.:3.75   3rd Qu.:5.75   3rd Qu.:7.75   3rd Qu.: 9.75
##  Max.    :2.00   Max.    :4.00   Max.    :6.00   Max.    :8.00   Max.    :10.00

##          V1           V2
##  Min.   :1   Min.   : 6
##  1st Qu.:2   1st Qu.: 7
##  Median :3   Median : 8
##  Mean    :3   Mean    : 8
##  3rd Qu.:4   3rd Qu.: 9
##  Max.    :5   Max.    :10
```

Outras formas de construir matrizes é juntando objetos existentes através dos comandos `cbind()` e `rbind()` que concatenam objetos por colunas e linhas, respectivamente.

Vamos juntar as matrizes já criadas.

```
o_matriz3<-rbind(o_matriz1,o_matriz1) # concatena por linhas (row)

o_matriz3

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

```

o_matriz4<-cbind(o_matriz2,o_matriz2) # concatena por colunas (column)
o_matriz4

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

```

2.2.2.4 Array

O array é um conjunto de matrizes ou vetores que podem ter qualquer número de dimensões. Estas dimensões podem receber nomes. Podemos criar *arrays* atribuindo dimensões a um vetor com o comando *dim()* ou usando *array()*.

```

array(data = NA, dim = length(data), dimnames = NULL)

q_array<-1:12 # cria um vetor
dim(q_array)<-c(2,3,2) # atribuindo 3 dimensões a q_array
q_array

```

```

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

r_array<-array(1:12,c(2,3,2)) # cria um array de 3 dimensões
r_array

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

```

Equipes	Jogos	Vitórias	Empates	Derrotas	Gols Pró	Gols Contra	
Flamengo	38	28	6	4	86	37	0
Santos	38	22	8	8	60	33	0
Palmeiras	38	21	11	6	61	32	0
Grêmio	38	19	8	11	64	39	0
Athletico Paranaense	38	18	10	10	51	32	0
Sao Paulo	38	17	12	9	39	30	0
Internacional	38	16	9	13	44	39	0
Corinthians	38	14	14	10	42	34	0
Fortaleza	38	15	8	15	50	49	0
Goiás	38	15	7	16	46	64	0
Bahia	38	12	13	13	44	43	0
Vasco da Gama	38	12	13	13	39	45	0
Atlético Mineiro	38	13	9	16	45	49	0
Fluminense	38	12	10	16	38	46	0
Botafogo	38	13	4	21	31	45	0
Ceará	38	10	9	19	36	41	0
Cruzeiro	38	7	15	16	27	46	0
CSA	38	8	8	22	24	58	0
Chapecoense	38	7	11	20	31	52	0
Avaí	38	3	11	24	18	62	0

2.2.2.5 Data Frame

Tão comum quanto a matriz, o data frame também um modo bidimensional de organização dos dados (linhas e colunas) que, diferentemente da matriz, permite objetos de tipos diferentes (character, numeric, logical e etc) sejam armazenados. Normalmente, nos data frames temos nas linhas as observações e nas colunas temos as variáveis. No entanto, é importante que cada coluna tenha o mesmo tamanho.

```
data.frame(..., row.names = NULL, check.rows = FALSE,check.names = TRUE,
fix.empty.names = TRUE,stringsAsFactors = default.stringsAsFactors())
```

Tabela 3: Data frame do resultados final do Campeonato Brasileiro de 2019.

```
Nome<-c("A", "B", "C")
Idade<-c(25, 32, 28)
Sexo<-c("M", "M", "F")
Nome;Idade;Sexo
```

```
## [1] "A" "B" "C"
## [1] 25 32 28
```

```

## [1] "M" "M" "F"
Ficha<-data.frame(Nome,Idade,Sexo)

attributes(Ficha) # exibe os atributos do data frame

## $names
## [1] "Nome"  "Idade" "Sexo"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3



| Nome | Idade | Sexo |
|------|-------|------|
| A    | 25    | M    |
| B    | 32    | M    |
| C    | 28    | F    |


```

Tabela 4: Tabela com dados fictícios.

2.2.2.6 Lista (*list*)

List permite combinar diferentes estruturas de dados em um mesmo objeto, ou seja, *vetores*, *matrizes*, *arrays*, *data.frames* e até outras listas. As listas são construídas utilizando o comando *list()*.

Os componentes da lista são criados da mesma maneira como para *data.frame*.
list(...)

```

Nome<-c("A")
Idade<-c(25)
Sexo<-c("M")
Notas<-c(55,42,50,35)
Nome;Idade;Sexo;Notas

## [1] "A"
## [1] 25
## [1] "M"
## [1] 55 42 50 35
Desempenho<-list(nome="A", idade=25, Sexo="M", Notas=c(55,42,50,35))
Desempenho

## $nome
## [1] "A"
## 
```

```
## $idade
## [1] 25
##
## $Sexo
## [1] "M"
##
## $Notas
## [1] 55 42 50 35
is.list(Desempenho)

## [1] TRUE
```

Existem também alguns comandos/funções que mostram resultados como listas.

```
s<-c(1:30)
t<-c(30:59)

teste.t<-t.test(s,t, var.equal = T) # calcula o test t para 2 amostras de variâncias iguais
teste.t

## 
## Two Sample t-test
##
## data: s and t
## t = -12.758, df = 58, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -33.54996 -24.45004
## sample estimates:
## mean of x mean of y
##      15.5       44.5
is.list(teste.t);mode(teste.t)

## [1] TRUE
## [1] "list"
```

Para finalizar por hoje preste atenção no seu *Global Environment*. Veja que há uma grande quantidade de objetos lá. Agora execute o comando abaixo:

```
rm(Desempenho)

rm(list=ls())
```

Voilà! Sumiu tudo.

Chapter 3

Acessando e manipulação de dados

3.1 Organização

Como mencionado anteriormente, `vector` é uma das principais estruturas do **R**. Um vetor é uma sequência de elementos (1...n) do mesmo tipo organizados tal qual livros em uma prateleira.

No caso abaixo temos um vetor numérico que pode ser `integer` ou `double`.

```
a<-c(25,17,55,3,12,315,10,2,3,11)
a
## [1] 25 17 55  3 12 315 10  2  3 11
      Posição 1 2 3 4 5 6 7 8 9 10
      Elementos 25 17 55  3 12 315 10  2  3 11
```

Um vetor também pode armazenar caracteres. Neste caso eles devem estar entre aspas duplas ("").

```
b<-c("Z","f", "7", "2a", "Yes", "A", "Ab")
b;typeof(b)
## [1] "Z"    "f"    "7"    "2a"   "Yes"  "A"    "Ab"
## [1] "character"
```

Embora tenhamos o número 7 entre os nossos elementos do vetor acima, por estar entre aspas o número 7 não é `numéric`, mas caractere.

Lembrem-se que um `vector` é um banco de dados da mesmo tipo Logo, se você

colocar um número 7 dentro dele, o **R** vai automaticamente convertê-lo em caractere independente de estar entre "" ou não.

	Posição	1	2	3	4	5	6	7
Elementos		"Z"	"f"	"7"	"2a"	"Yes"	"A"	"Ab"

Mesmo que você tenha apenas um único numeral ele será armazenado como vector, vector de **comprimento = 1**. Então um único número ou um único caractere será um vector.

```
c<-(1)
c;typeof(c);is.vector(c)
```

```
## [1] 1
## [1] "double"
## [1] TRUE
```

	Posição	1
Elemento		"1"

Entender a posição de cada elemento dentro de um **vector** é crucial! Saber a posição correta onde determinado(s) elemento(s) está(estão) facilita em muito nossa limpeza, nosso tratamento e demais passos para uma eficiente manipulação dos dados.

```
seq(100,50,-1)
```

```
## [1] 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82
## [20] 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63
## [39] 62 61 60 59 58 57 56 55 54 53 52 51 50
```

Atenção para os números dentro dos colchetes [1] [17] [33] [49]. Como mencionado em aulas passadas, eles, os colchetes, indicam a posição de determinados elementos dentro de um banco de dados.

Especificamente o [1] informa que o número 100 é o primeiro elemento, [17] informa que o número 84 é o vigésimo elemento, [33] informa que o número 68 é o trigésimo nono elemento, [49] informa que o número 52 é o trigésimo nono elemento.

3.1.1 Usando [] - Subscribing ou indexing

3.1.1.1 Vetores

Podemos extrapolar esta ideia de utilização de [] para acessar informações dentro dos objetos que criamos ou importamos.

```
d<-c("a", "b", "c", "d", "e")
d
```

```
## [1] "a" "b" "c" "d" "e"
```

	Posição	1	2	3	4	5
Elementos		"a"	"b"	"c"	"d"	"e"

No `vector d` temos cada um dos elementos em sua respectiva posição. Para acessar um elemento específico podemos inserir o número referente à sua posição dentro de [].

Por exemplo:

```
d[1]
```

```
## [1] "a"
```

```
d[2]
```

```
## [1] "b"
```

```
d[3]
```

```
## [1] "c"
```

```
d[4]
```

```
## [1] "d"
```

```
d[5]
```

```
## [1] "e"
```

Importante saber que [] é muito versátil.

```
d[-1] # Acessa todos elementos exceto o primeiro
```

```
## [1] "b" "c" "d" "e"
```

```
d1<-d[3] # Atribuir o(s) elemento(s) de um vetor a outro vetor, ou criar um novo objeto de elemen
d1
```

```
## [1] "c"
```

```
d[1:3] # Acessar um intervalo de elementos
```

```
## [1] "a" "b" "c"
d[3:5]

## [1] "c" "d" "e"
d[c(1,3:5)] # Acessar utilizando combinações

## [1] "a" "c" "d" "e"
d[c(-2,-4)] # Exceto o segundo e quanto elementos

## [1] "a" "c" "e"
d[-3:-5] # Exceto o intervalo entre 3 e 5

## [1] "a" "b"
```

Vamos dar mais uma olhada na importância do [].

Considere os seguintes vetores.

Posição	1	2	3	4	5	6	7	8	9	10
Elemento	50	34	11	7	24	631	20	4	7	21

Posição	1	2	3	4	5	6	7	8	9	10
Elemento	100	2	56	12	0	65	93	10	244	1

Vamos somar os dois vetores e observe que são adicionados o primeiro elemento com primeiro elemento, segundo elemento com segundo elemento e assim por diante.

Em algumas outras linguagens se você quiser fazer esse procedimento, você vai provavelmente usar um `loop`. Mas em **R** você pode simplesmente adicioná-los. Esse é o motivo do **R** ser linguagem vetorizada. Você pode testar qualquer método matemático (`soma`, `divisão`, `booleana`, `lógica`).

```
e<-c(50,34,11,7,24,631,20,4,7,21)

f<-c(100,2,56,12,0,65,93,10,244,1)

e;f

## [1] 50 34 11 7 24 631 20 4 7 21
## [1] 100 2 56 12 0 65 93 10 244 1
e+f

## [1] 150 36 67 19 24 696 113 14 251 22
```

Neste caso funcionou tudo perfeitamente, pois nossos vetores são do mesmo tamanho.

No entanto, caso tenhamos vetores de tamanho diferentes o **R** vai fazer um procedimento chamado recycling of vector. O **R** vai fazer com que os dois vetores se encaixarem no mesmo tamanho caso sejam múltiplos. Basicamente, o **R** vai copiar os elementos do começo do vetor menor e adicionar ao seu fim até que fique do mesmo tamanho do outro vetor para, então, fazer a operação.

Posição	1	2	3	4	5
Elemento	50	34	11	7	24

Posição	1	2	3	4	5	6	7	8	9	10
Elemento	100	2	56	12	0	65	93	10	244	1

```

h<-c( 50,34,11, 7,24)

i<-c(100, 2,56,12, 0,65,93,10,244,1)

h+i

## [1] 150 36 67 19 24 115 127 21 251 25
h;i

## [1] 50 34 11 7 24
## [1] 100 2 56 12 0 65 93 10 244 1
h+i

## [1] 150 36 67 19 24 115 127 21 251 25

```

E se não forem múltiplos?

Nesse caso vamos ter uma aviso, em que o **R** ao fazer o preenchimento dos valores faltantes vai querer saber se está certo o que estamos fazendo e emitindo um **warning**.

```

j<-c(50,34,11,7,24,631)
k<-c(100,2,56,12,0,65,93,10,244,1)
j+k

## Warning in j + k: comprimento do objeto maior não é múltiplo do comprimento do
## objeto menor

## [1] 150 36 67 19 24 696 143 44 255 8

```

3.1.1.2 Matrizes

Da mesma forma que utilizamos [] para acessar elementos específicos em um vetor, estes podem ser utilizados para acessar elementos de matrizes. Lembrando que as matrizes possuem duas dimensões e que as dimensões são organizadas na sequência de linhas e colunas.

```
1<-cbind(matrix(14:1, ncol=2),matrix(1:14, ncol=2))
1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    14    7    1    8
## [2,]    13    6    2    9
## [3,]    12    5    3   10
## [4,]    11    4    4   11
## [5,]    10    3    5   12
## [6,]     9    2    6   13
## [7,]     8    1    7   14
```

As linhas e colunas [linhas, colunas] são numeradas sequencialmente [x...n,y] e [x,y...n].

Podemos desta forma selecionar uma ou várias linhas e colunas de uma só vez. Para isto basta apenas numerar a linhas ou coluna requerida.

```
1[1,] # Apenas a primeira linhas
```

```
## [1] 14 7 1 8
1[,1] # Apenas a primeira coluna
```

```
## [1] 14 13 12 11 10 9 8
1[1:3,] # O intervalo de linhas entre 1 a 3
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    14    7    1    8
## [2,]    13    6    2    9
## [3,]    12    5    3   10
1[,1:3] # O intervalo de colunas entre 1 e 3
```

```
##      [,1] [,2] [,3]
## [1,]    14    7    1
## [2,]    13    6    2
## [3,]    12    5    3
## [4,]    11    4    4
## [5,]    10    3    5
## [6,]     9    2    6
## [7,]     8    1    7
```

```

l[-3,] # Exceto a terceira linha

##      [,1] [,2] [,3] [,4]
## [1,]    14    7    1    8
## [2,]    13    6    2    9
## [3,]    11    4    4   11
## [4,]    10    3    5   12
## [5,]     9    2    6   13
## [6,]     8    1    7   14

l[,-3] # Exceto a terceira coluna

##      [,1] [,2] [,3]
## [1,]    14    7    8
## [2,]    13    6    9
## [3,]    12    5   10
## [4,]    11    4   11
## [5,]    10    3   12
## [6,]     9    2   13
## [7,]     8    1   14

l[-1:-3] # Exceto o intervalo de linhas entre 1 a 3

##      [,1] [,2] [,3] [,4]
## [1,]    11    4    4   11
## [2,]    10    3    5   12
## [3,]     9    2    6   13
## [4,]     8    1    7   14

l[,-1:-3] # Exceto o intervalo de colunas entre 1 e 3

## [1] 8 9 10 11 12 13 14

```

Podemos também acessar elementos específicos de uma matriz definindo o “endereço” do elemento nas linhas e colunas.

```

l[2,3] # Acessa apenas o elemento do cruzamento da linha 2 e coluna 3

## [1] 2

```

Para facilitar a manipulação das matrizes podemos nomear as linhas e colunas. Para isso podemos utilizar as funções `colnames()` e `rownames()`.

```

nrow(l) # Retorna o número de linhas

## [1] 7

ncol(l) # Retorna o número de colunas

## [1] 4

```

```

dim(l) # Retorna as dimensões (nº de linhas e colunas)

## [1] 7 4

l

##      [,1] [,2] [,3] [,4]
## [1,]    14    7    1    8
## [2,]    13    6    2    9
## [3,]    12    5    3   10
## [4,]    11    4    4   11
## [5,]    10    3    5   12
## [6,]     9    2    6   13
## [7,]     8    1    7   14

rownames(l)<-letters[1:7] # Atribui 7 letras minúsculas sequenciais de "a" até "g" com
l

##      [,1] [,2] [,3] [,4]
## a    14    7    1    8
## b    13    6    2    9
## c    12    5    3   10
## d    11    4    4   11
## e    10    3    5   12
## f     9    2    6   13
## g     8    1    7   14

colnames(l)<-LETTERS[1:ncol(l)] # Atribui letras maiúsculas sequenciais obedecendo o n
l

##      A B C D
## a 14 7 1 8
## b 13 6 2 9
## c 12 5 3 10
## d 11 4 4 11
## e 10 3 5 12
## f  9 2 6 13
## g  8 1 7 14

```

Agora podemos acessar via nomes das linhas e colunas também.

```

l["a",] # Apenas a primeira linhas

##  A  B  C  D
## 14 7 1 8

l[, "A"] # Apenas a primeira coluna

##  a  b  c  d  e  f  g
## 14 13 12 11 10  9  8

```

```

l[letters[1:3],] # O intervalo de linhas entre "a" a "c"

##      A B C D
## a 14 7 1 8
## b 13 6 2 9
## c 12 5 3 10

l[,LETTERS[1:3]] # O intervalo de colunas entre "A" e "C"

##      A B C
## a 14 7 1
## b 13 6 2
## c 12 5 3
## d 11 4 4
## e 10 3 5
## f  9 2 6
## g  8 1 7

l[c("a","c","d"),] # Seleciona as linhas "a", "c" e "d"

##      A B C D
## a 14 7 1 8
## c 12 5 3 10
## d 11 4 4 11

l[,c("A","C","D")] # Seleciona as colunas "a", "c" e "d"

##      A C D
## a 14 1 8
## b 13 2 9
## c 12 3 10
## d 11 4 11
## e 10 5 12
## f  9 6 13
## g  8 7 14

```

3.1.1.3 Arrays

Sendo muito similares aos vetores e matrizes, os arrays podem ter n dimensões.

```

m<-1:24
dim(m)<-c(4,3,2)
m

## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10

```

```

## [3,]    3    7   11
## [4,]    4    8   12
##
## , , 2
##
## [,1] [,2] [,3]
## [1,]  13   17   21
## [2,]  14   18   22
## [3,]  15   19   23
## [4,]  16   20   24

# ou

m<-array(1:24,c(4,3,2))
m

## , , 1
##
## [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
##
## , , 2
##
## [,1] [,2] [,3]
## [1,]  13   17   21
## [2,]  14   18   22
## [3,]  15   19   23
## [4,]  16   20   24

m[1,,] # Retorna as primeiras linhas do array

## [,1] [,2]
## [1,]    1   13
## [2,]    5   17
## [3,]    9   21

m[,2,] # Retorna as segundas colunas do array

## [,1] [,2]
## [1,]    5   17
## [2,]    6   18
## [3,]    7   19
## [4,]    8   20

```

```
m[1,2,] # Retorna as intersecções entre as primeiras linhas com as segundas colunas
```

```
## [1] 5 17
```

```
m[1,2,2] # Retorna as intersecções entre a primeira linha com segunda coluna da sugunda camada
```

```
## [1] 17
```

3.1.1.4 Data Frame

Sendo semelhantes às **matrizes**, os **data frames** podem ser acessados pelo número das linhas ou colunas.

Equipes	Jogos	Vitórias	Empates	Derrotas	Gols Pró	Gols Contra	Pontos
Flamengo	38	28	6	4	86	37	90
Santos	38	22	8	8	60	33	74
Palmeiras	38	21	11	6	61	32	74
Grêmio	38	19	8	11	64	39	65
Athletico Paranaense	38	18	10	10	51	32	64
Sao Paulo	38	17	12	9	39	30	63
Internacional	38	16	9	13	44	39	57
Corinthians	38	14	14	10	42	34	56
Fortaleza	38	15	8	15	50	49	53
Goiás	38	15	7	16	46	64	52
Bahia	38	12	13	13	44	43	49
Vasco da Gama	38	12	13	13	39	45	49
Atlético Mineiro	38	13	9	16	45	49	48
Fluminense	38	12	10	16	38	46	46
Botafogo	38	13	4	21	31	45	43
Ceará	38	10	9	19	36	41	39
Cruzeiro	38	7	15	16	27	46	36
CSA	38	8	8	22	24	58	32
Chapecoense	38	7	11	20	31	52	32
Avaí	38	3	11	24	18	62	20

```
Campeonato.Brasileiro.2019[1,] # Retorna a primeira linha do data frame
```

```
##   Equipes Jogos Vitórias Empates Derrotas Gols.Pró Gols.Contra Pontos
## 1 Flamengo  38     28      6       4      86          37        90
## Saldo.de.Gols Aprov. Destino
## 1             49     79 Libertadores
```

```
Campeonato.Brasileiro.2019[,1] # Retorna a primeira coluna do data frame
```

```
## [1] "Flamengo"           "Santos"                "Palmeiras"
## [4] "Grêmio"              "Athletico Paranaense" "Sao Paulo"
## [7] "Internacional"       "Corinthians"           "Fortaleza"
## [10] "Goiás"               "Bahia"                 "Vasco da Gama"
## [13] "Atlético Mineiro"   "Fluminense"            "Botafogo"
## [16] "Ceará"                "Cruzeiro"              "CSA"
## [19] "Chapecoense"          "Avaí"
```

Todas as outras combinações utilizadas nas `matrizes` podem ser utilizadas aqui nos `data frames`.

No entanto, os `data frames` possuem uma outra “vantagem” sobre as `matrizes`. Além de armazenar elementos de diferentes tipos (`numeric`, `character`, `logical` e etc), eles também podem ser acessados utilizando `$`.

```
Campeonato.Brasileiro.2019$Equipes # Retorna os elementos da coluna "Equipes"
```

```
## [1] "Flamengo"           "Santos"                "Palmeiras"
```

```

## [4] "Grêmio"           "Athletico Paranaense" "Sao Paulo"
## [7] "Internacional"    "Corinthians"        "Fortaleza"
## [10] "Goiás"            "Bahia"              "Vasco da Gama"
## [13] "Atlético Mineiro" "Fluminense"        "Botafogo"
## [16] "Ceará"             "Cruzeiro"          "CSA"
## [19] "Chapecoense"       "Avaí"

Campeonato.Brasileiro.2019$Vitórias # Retorna os elementos da coluna "Vitórias"

## [1] "28"  "22"  "21"  "19"  "18"  "17"  "16"  "14"  "15"  "15"  "12"  "12"  "13"  "12"  "13"
## [16] "10"  "7"   "8"   "7"   "3"

Campeonato.Brasileiro.2019$Gols.Pró # Retorna os elementos da coluna "Gols.Pró"

## [1] "86"  "60"  "61"  "64"  "51"  "39"  "44"  "42"  "50"  "46"  "44"  "39"  "45"  "38"  "31"
## [16] "36"  "27"  "24"  "31"  "18"

Campeonato.Brasileiro.2019$Saldo.de.Gols # Retorna os elementos da coluna "Saldo.de.Gols"

## [1] "49"  "27"  "29"  "25"  "19"  "9"   "5"   "8"   "1"   "-18" "1"   "-6"
## [13] "-4"  "-8"  "-14" "-5"  "-19" "-34" "-21" "-44"

Campeonato.Brasileiro.2019$Destino # Retorna os elementos da coluna "Destino"

## [1] "Libertadores" "Libertadores" "Libertadores" "Libertadores" "Libertadores"
## [6] "Libertadores" "Libertadores" "Libertadores" "Sulamericana" "Sulamericana"
## [11] "Sulamericana" "Sulamericana" "Sulamericana" "Sulamericana" "NA"
## [16] "NA"          "Rebaixado"   "Rebaixado"   "Rebaixado"   "Rebaixado"

Campeonato.Brasileiro.2019$Equipes[2] # Retorna o segundo elemento da coluna "Nome"

## [1] "Santos"

Campeonato.Brasileiro.2019$Vitórias[5] # Retorna o quinto elemento da coluna "Vitórias"

## [1] "18"

Campeonato.Brasileiro.2019$Gols.Pró[3] # Retorna os elementos da coluna "Gols.Pró"

## [1] "61"

Campeonato.Brasileiro.2019$Saldo.de.Gols[1] # Retorna os elementos da coluna "Nivel.deR"

## [1] "49"

Campeonato.Brasileiro.2019$Destino[10] # Retorna os elementos da décima coluna "Destino"

## [1] "Sulamericana"

Outra maneira de acessar informações pode através [].

Campeonato.Brasileiro.2019[Campeonato.Brasileiro.2019$Equipes=="Flamengo",] # Retorna apenas as

```

```

##      Equipes Jogos Vitórias Empates Derrotas Gols.Pró Gols.Contra Pontos
## 1 Flamengo    38     28      6      4     86        37     90
##   Saldo.de.Gols Aprov.      Destino
## 1             49     79 Libertadores
split(Campeonato.Brasileiro.2019,Destino) # Retorna apenas as informações da coluna "Destino"

## $Libertadores
##      Equipes Jogos Vitórias Empates Derrotas Gols.Pró Gols.Contra
## 1     Flamengo    38     28      6      4     86        37
## 2       Santos    38     22      8      8     60        33
## 3     Palmeiras    38     21     11      6     61        32
## 4      Grêmio     38     19      8     11     64        39
## 5 Athletico Paranaense    38     18     10     10     51        32
## 6      Sao Paulo    38     17     12      9     39        30
## 7     Internacional    38     16      9     13     44        39
## 8     Corinthians    38     14     14     10     42        34
##   Pontos Saldo.de.Gols Aprov.      Destino
## 1     90          49     79 Libertadores
## 2     74          27     65 Libertadores
## 3     74          29     65 Libertadores
## 4     65          25     57 Libertadores
## 5     64          19     56 Libertadores
## 6     63           9     55 Libertadores
## 7     57           5     55 Libertadores
## 8     56           8     49 Libertadores
##
## $`NA`
##      Equipes Jogos Vitórias Empates Derrotas Gols.Pró Gols.Contra Pontos
## 15 Botafogo    38     13      4     21     31        45     43
## 16     Ceará     38     10      9     19     36        41     39
##   Saldo.de.Gols Aprov. Destino
## 15          -14     38     NA
## 16          -5      34     NA
##
## $Rebaixado
##      Equipes Jogos Vitórias Empates Derrotas Gols.Pró Gols.Contra Pontos
## 17     Cruzeiro    38      7     15     16     27        46     36
## 18       CSA     38      8      8     22     24        58     32
## 19 Chapecoense    38      7     11     20     31        52     32
## 20      Avaí     38      3     11     24     18        62     20
##   Saldo.de.Gols Aprov.      Destino
## 17          -19     32 Rebaixado
## 18          -34     28 Rebaixado
## 19          -21     28 Rebaixado
## 20          -44     18 Rebaixado

```

```

## 
## $Sulamericana
##           Equipes Jogos Vitórias Empates Derrotas Gols.Pró Gols.Contra Pontos
## 9         Fortaleza   38      15       8      15      50        49      53
## 10        Goiás     38      15       7      16      46        64      52
## 11        Bahia     38      12      13      13      44        43      49
## 12 Vasco da Gama   38      12      13      13      39        45      49
## 13 Atlético Mineiro 38      13       9      16      45        49      48
## 14 Fluminense    38      12      10      16      38        46      46
##   Saldo.de.Gols Aprov. Destino
## 9            1    46 Sulamericana
## 10           -18   46 Sulamericana
## 11            1    43 Sulamericana
## 12            -6   43 Sulamericana
## 13            -4   42 Sulamericana
## 14            -8    4 Sulamericana

```

Alterando elementos dentro do `data frame`.

```
Campeonato.Brasileiro.2019$Destino # Retorna apenas as informações da coluna "Destino"
```

```

## [1] "Libertadores" "Libertadores" "Libertadores" "Libertadores" "Libertadores"
## [6] "Libertadores" "Libertadores" "Libertadores" "Sulamericana" "Sulamericana"
## [11] "Sulamericana" "Sulamericana" "Sulamericana" "Sulamericana" "NA"
## [16] "NA"          "Rebaixado"    "Rebaixado"    "Rebaixado"    "Rebaixado"
Campeonato.Brasileiro.2019$Destino[Campeonato.Brasileiro.2019$Destino=="NA"]<- "Lugar Algum"

# Substitui "Libertadores" na coluna "Destino" por "Liberta".

```

Veja como ficou com a informação alterada.

Equipes	Jogos	Vitórias	Empates	Derrotas	Gols Pró	Gols Contra	Pontos
Flamengo	38	28	6	4	86	37	90
Santos	38	22	8	8	60	33	74
Palmeiras	38	21	11	6	61	32	74
Grêmio	38	19	8	11	64	39	65
Athletico Paranaense	38	18	10	10	51	32	64
Sao Paulo	38	17	12	9	39	30	63
Internacional	38	16	9	13	44	39	57
Corinthians	38	14	14	10	42	34	56
Fortaleza	38	15	8	15	50	49	53
Goiás	38	15	7	16	46	64	52
Bahia	38	12	13	13	44	43	49
Vasco da Gama	38	12	13	13	39	45	49
Atlético Mineiro	38	13	9	16	45	49	48
Fluminense	38	12	10	16	38	46	46
Botafogo	38	13	4	21	31	45	43
Ceará	38	10	9	19	36	41	39
Cruzeiro	38	7	15	16	27	46	36
CSA	38	8	8	22	24	58	32
Chapecoense	38	7	11	20	31	52	32
Avaí	38	3	11	24	18	62	20

Vamos expurgar remover o *Vasco da Gama* da nossa lista.

```
Campeonato.Brasileiro.2019$Equipes # Retorna apenas as informações da coluna "Equipes"
```

```
## [1] "Flamengo"          "Santos"           "Palmeiras"
## [4] "Grêmio"            "Athletico Paranaense" "Sao Paulo"
## [7] "Internacional"     "Corinthians"        "Fortaleza"
## [10] "Goiás"             "Bahia"              "Vasco da Gama"
## [13] "Atlético Mineiro"  "Fluminense"         "Botafogo"
## [16] "Ceará"              "Cruzeiro"          "CSA"
## [19] "Chapecoense"        "Avaí"
```

```
Campeonato.Brasileiro.2019[Campeonato.Brasileiro.2019$Equipes!="Vasco da Gama",] #Sele
```

	Equipes	Jogos	Vitórias	Empates	Derrotas	Gols.Pró	Gols.Contra
## 1	Flamengo	38	28	6	4	86	37
## 2	Santos	38	22	8	8	60	33
## 3	Palmeiras	38	21	11	6	61	32
## 4	Grêmio	38	19	8	11	64	39
## 5	Athletico Paranaense	38	18	10	10	51	32
## 6	Sao Paulo	38	17	12	9	39	30
## 7	Internacional	38	16	9	13	44	39
## 8	Corinthians	38	14	14	10	42	34
## 9	Fortaleza	38	15	8	15	50	49
## 10	Goiás	38	15	7	16	46	64

## 11	Bahia	38	12	13	13	44	43
## 13	Atlético Mineiro	38	13	9	16	45	49
## 14	Fluminense	38	12	10	16	38	46
## 15	Botafogo	38	13	4	21	31	45
## 16	Ceará	38	10	9	19	36	41
## 17	Cruzeiro	38	7	15	16	27	46
## 18	CSA	38	8	8	22	24	58
## 19	Chapecoense	38	7	11	20	31	52
## 20	Avaí	38	3	11	24	18	62
## Pontos Saldo.de.Gols Aprov. Destino							
## 1	90	49	79	Libertadores			
## 2	74	27	65	Libertadores			
## 3	74	29	65	Libertadores			
## 4	65	25	57	Libertadores			
## 5	64	19	56	Libertadores			
## 6	63	9	55	Libertadores			
## 7	57	5	5	Libertadores			
## 8	56	8	49	Libertadores			
## 9	53	1	46	Sulamericana			
## 10	52	-18	46	Sulamericana			
## 11	49	1	43	Sulamericana			
## 13	48	-4	42	Sulamericana			
## 14	46	-8	4	Sulamericana			
## 15	43	-14	38	Lugar Algum			
## 16	39	-5	34	Lugar Algum			
## 17	36	-19	32	Rebaixado			
## 18	32	-34	28	Rebaixado			
## 19	32	-21	28	Rebaixado			
## 20	20	-44	18	Rebaixado			

Equipes	Jogos	Vitórias	Empates	Derrotas	Gols Pró	Gols Contra	Pontos
Flamengo	38	28	6	4	86	37	90
Santos	38	22	8	8	60	33	74
Palmeiras	38	21	11	6	61	32	74
Grêmio	38	19	8	11	64	39	65
Athletico Paranaense	38	18	10	10	51	32	64
Sao Paulo	38	17	12	9	39	30	63
Internacional	38	16	9	13	44	39	57
Corinthians	38	14	14	10	42	34	56
Fortaleza	38	15	8	15	50	49	53
Goiás	38	15	7	16	46	64	52
Bahia	38	12	13	13	44	43	49
Vasco da Gama	38	12	13	13	39	45	49
Atlético Mineiro	38	13	9	16	45	49	48
Fluminense	38	12	10	16	38	46	46
Botafogo	38	13	4	21	31	45	43
Ceará	38	10	9	19	36	41	39
Cruzeiro	38	7	15	16	27	46	36
CSA	38	8	8	22	24	58	32
Chapecoense	38	7	11	20	31	52	32
Avaí	38	3	11	24	18	62	20

Atendendo a pedidos, vamos continuar brevemente com manipulação de dados em `data.frame`.

As funções `grep`, `grepl`, `regexpr`, `gregexpr` e `regexec` busca por elementos de um objeto que coincidem com um *argumento padrão*.

3.1.2 grep()

```
grep("Lugar Algum", Campeonato.Brasileiro.2019$Destino) # retorna as posições de tudo que contém "Lugar Algum"

## [1] 15 16

grep("Libertadores|Lugar Algum", Campeonato.Brasileiro.2019$Destino) # retorna as posições de "Libertadores" ou "Lugar Algum"

## [1] 15 16

grep("Lugar Algum", Campeonato.Brasileiro.2019$Destino) # retorna as posições de tudo que contém "Lugar Algum" exceto "Libertadores"

## [1] 15 16

Campeonato.Brasileiro.2019$Destino[15]<- "lugar algum" #Substitui a 15ª posição por "lugar algum"
```

Execute o comando `grep` anterior novamente e veja as diferenças.

Equipes	Jogos	Vitórias	Empates	Derrotas	Gols.Pró	Gols.Contra	Pontos	Saldo.de.Gol
Flamengo	38	28	6	4	86	37	90	49
Santos	38	22	8	8	60	33	74	27
Palmeiras	38	21	11	6	61	32	74	29
Grêmio	38	19	8	11	64	39	65	25
Athletico Paranaense	38	18	10	10	51	32	64	19
Sao Paulo	38	17	12	9	39	30	63	9
Internacional	38	16	9	13	44	39	57	5
Corinthians	38	14	14	10	42	34	56	8
Fortaleza	38	15	8	15	50	49	53	1
Goiás	38	15	7	16	46	64	52	-18
Bahia	38	12	13	13	44	43	49	1
Vasco da Gama	38	12	13	13	39	45	49	-6
Atlético Mineiro	38	13	9	16	45	49	48	-4
Fluminense	38	12	10	16	38	46	46	-8
Botafogo	38	13	4	21	31	45	43	-14
Ceará	38	10	9	19	36	41	39	-5
Cruzeiro	38	7	15	16	27	46	36	-19
CSA	38	8	8	22	24	58	32	-34
Chapecoense	38	7	11	20	31	52	32	-21
Avaí	38	3	11	24	18	62	20	-44

```

grep("[L l]ugar [A a]lgum", Campeonato.Brasileiro.2019$Destino) # retorna as posições de tudo no vetor
## [1] 15 16

la <- grep("lugar algum", Campeonato.Brasileiro.2019$Destino)
LA <- grep("Lugar Algum", Campeonato.Brasileiro.2019$Destino)
str(la);str(LA) # mostra a estrutura de um objeto

## int 15
## int 16
setdiff(la, LA) #Identifica a diferença entre X e Y
## [1] 15
setdiff(LA, la)

## [1] 16
Campeonato.Brasileiro.2019$Equipes[1]

## [1] "Flamengo"
grep("Flamengo", Campeonato.Brasileiro.2019$Equipes, value = TRUE)
## [1] "Flamengo"

```

3.1.3 grep()

```
grepl("Flamengo", Campeonato.Brasileiro.2019$Equipes) # Retorna valores lógicos de acordo com a condição
```



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

3.1.4 sub() e gsub()

```

sub("Fla", "fla", Campeonato.Brasileiro.2019$Equipes) #Substitui Fla por fla naquilo q

## [1] "flamengo"           "Santos"                 "Palmeiras"
## [4] "Grêmio"              "Athletico Paranaense" "Sao Paulo"
## [7] "Internacional"       "Corinthians"            "Fortaleza"
## [10] "Goiás"                "Bahia"                  "Vasco da Gama"
## [13] "Atlético Mineiro"    "Fluminense"             "Botafogo"
## [16] "Ceará"                 "Cruzeiro"               "CSA"
## [19] "Chapecoense"          "Avaí"

gsub("Atlé|Athle", "Atlé", Campeonato.Brasileiro.2019$Equipes) #Substitui Atlé ou Athl

## [1] "Flamengo"           "Santos"                 "Palmeiras"
## [4] "Grêmio"              "Atlético Paranaense" "Sao Paulo"
## [7] "Internacional"       "Corinthians"            "Fortaleza"
## [10] "Goiás"                "Bahia"                  "Vasco da Gama"
## [13] "Atlético Mineiro"    "Fluminense"             "Botafogo"
## [16] "Ceará"                 "Cruzeiro"               "CSA"
## [19] "Chapecoense"          "Avaí"

```

Alguns elementos especiais podem ser encontrados no R como: **Infinito positivo ou negativo (-Inf e Inf)**, elementos “não número” (NaN - Not a Number) e valores ausentes (NA, Not Available).

3/0

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

-3/0

```
## [1] -Inf
```

```
x<-Inf;x
```

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

```
y<-(-Inf);y
```

```
## [1] -Inf
```

0/0

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

```
Inf-Inf
```

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

```
Campeonato.Brasileiro.2019[Campeonato.Brasileiro.2019<=10] <-NA
```

Equipes	Jogos	Vitórias	Empates	Derrotas	Gols.Pró	Gols.Contra	Pontos	Saldo.de.Gol
Flamengo	38	28	6	4	86	37	90	49
Santos	38	22	8	8	60	33	74	27
Palmeiras	38	21	11	6	61	32	74	29
Grêmio	38	19	8	11	64	39	65	25
Athletico Paranaense	38	18	NA	NA	51	32	64	19
Sao Paulo	38	17	12	9	39	30	63	9
Internacional	38	16	9	13	44	39	57	5
Corinthians	38	14	14	NA	42	34	56	8
Fortaleza	38	15	8	15	50	49	53	NA
Goiás	38	15	7	16	46	64	52	NA
Bahia	38	12	13	13	44	43	49	NA
Vasco da Gama	38	12	13	13	39	45	49	NA
Atlético Mineiro	38	13	9	16	45	49	48	NA
Fluminense	38	12	NA	16	38	46	46	NA
Botafogo	38	13	4	21	31	45	43	NA
Ceará	38	NA	9	19	36	41	39	NA
Cruzeiro	38	7	15	16	27	46	36	NA
CSA	38	8	8	22	24	58	32	NA
Chapecoense	38	7	11	20	31	52	32	NA
Avaí	38	3	11	24	18	62	20	NA

```
any(is.na(Campeonato.Brasileiro.2019))
```

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

Identificando onde estão os valores que atendem critérios desejados.

```
which(is.na(Campeonato.Brasileiro.2019), arr.ind=TRUE)
```

```
##      row col
## [1,] 16   3
## [2,]  5   4
## [3,] 14   4
## [4,]  5   5
## [5,]  8   5
## [6,]  9   9
## [7,] 10   9
## [8,] 11   9
## [9,] 12   9
## [10,] 13   9
```

```

## [11,] 14 9
## [12,] 15 9
## [13,] 16 9
## [14,] 17 9
## [15,] 18 9
## [16,] 19 9
## [17,] 20 9
which((Campeonato.Brasileiro.2019<1), arr.ind=TRUE)

##      row col

```

3.2 Operadores

O R tem vários operadores, tais quais outras linguagens, que nos permitem realizar procedimentos *aritméticos, lógicos ou relacionais*.

Os operadores aritméticos atuam em 2 elementos (x e y) e retorna respostas lógicas. Estes elementos, por sua vez, podem ser de modo (`mode()`) numéricos ou complexos e também variáveis lógicas.

Símbolo	Operação
+	adição
-	subtração
*	multiplicação
/	divisão
\wedge	potência
%%	resto da divisão
%/%	divisão inteira

```
x<-2;y<-3
```

```
x+y
```

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

```
x-y
```

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

```
x*y
```

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

```
y/x
```

```
## [1] 1.5
```

```
y%/%x
```

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

```
y%/%x
```

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

```
y^x
```

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

Os operadores de comparações atuam em cada elemento de 2 objetos sob comparação (fazendo o *recycling* se necessário) retornando um objeto do mesmo tamanho.

Símbolo	Relação
<	menor que
>	maior que
<=	menor ou igual
>=	maior ou igual
==	igual
!=	diferente

```
x<-1:3;y<-1:3
```

```
x>y
```

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

```
x<y
```

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

```
x*y
```

```
## [1] 1 4 9
```

```
y>=x
```

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

```
y<=x
```

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

```
y==x
```

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

```
y != x
```

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

Caso queira comparar bancos de dados completos, por inteiro de uma vez só, utilize `identical(x,y)` e `all.equal(x,y)`.

`identical`faz uma comparação estrita da representação dos objetos e retorna como TRUE ou FALSE.

`all.equal` compara a “*igualdade próxima*” e retorna TRUE ou mostra as diferenças.

```
x <- 0.9; y <- 0.9
identical(x,y)
```

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

```
identical(0.9, 1.1 - 0.2)
```

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

```
all.equal(x,y)
```

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

```
all.equal(0.9, 1.1 - 0.2, tolerance = 1e-16)
```

```
## [1] "Mean relative difference: 1.233581e-16"
```

Operadores lógicos são utilizados para para testes lógicos entre 1 ou 2 objetos e retorna valores lógicos também.

Símbolo	Relação
!	Logical NOT
&	Logical AND
or	Exclusive OR

Lembrando que temos outro operadores \$, @, [, [[, :, ?, <-, <<-, =, :: que já começamos a manipular.

Com o passar do tempo vamos “brincando” com cada operador e, assim, assimilando o que cada uma faz.

3.2.1 Iterações

3.2.1.1 `while()`

Essa forma de *loop* é menos comumente usada e potencialmente perigosa, uma vez que ela pode resultar em uma execução descontrolada. Por isso, em muitos

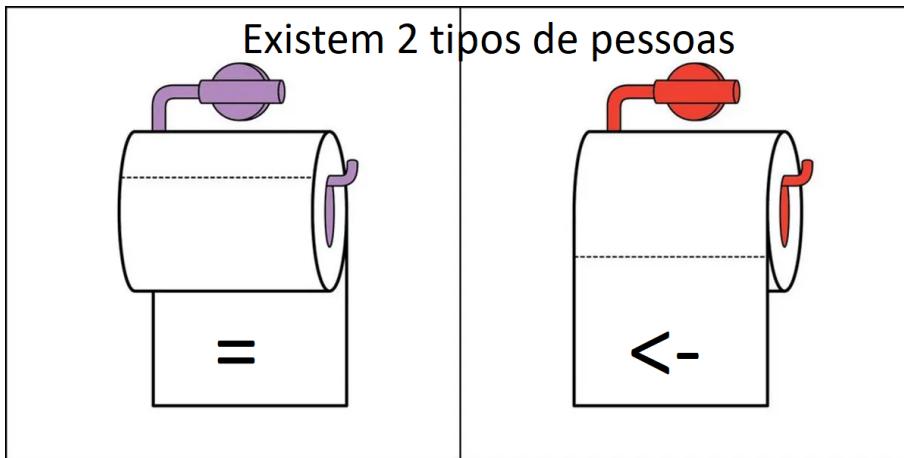
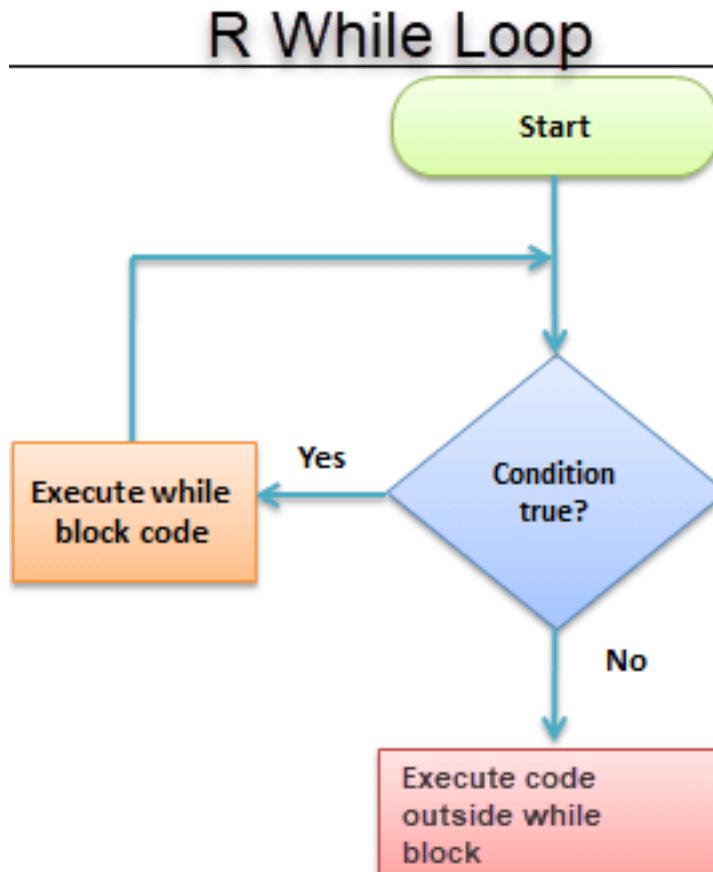


Figure 3.1: Existem dois tipos de pessoas: = & <->

casos iterações restritivas são mais adequadas. A função `while()` requer uma condição testável que continuará a resultar em declarações subsequentes a serem executadas até o resultado do teste dar falso (`FALSE`).

Estrutura: Começa com a palavra *while* seguida por *parênteses* e *colchetes* `while(){}` A segunda parte vai dentro de parênteses e diz respeito a uma *expressão lógica* e a terceira parte, entre colchetes, o *corpo do loop*. `while(abc){xyz} #Se abc for TRUE ele vai executar o resto {xyz}`. Depois volta para checar se (abc) continua *TRUE* ou *FALSE* para decidir se continua no loop ou parar.

```
while(condição){
  "Código a ser executado"
}
```



Fonte: Guru99

Exemplo 1: Teste Ter aula na segunda-feira é bom ou não?

```

while(FALSE){
  print("Aula segunda-feira é ótimo!")}

while(TRUE){
  print("Aula segunda-feira é ótimo!")}
```

Exemplo 2: Teste Num concurso de beleza Jackson desafiou qualquer um de seus alunos. Cada um recebeu uma nota.

```

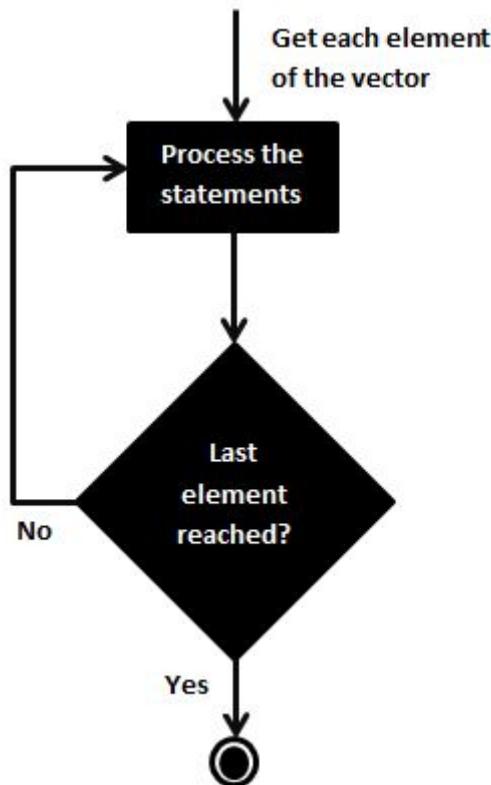
Jackson<-9.5
Aluno<-6
while(Aluno<Jackson){
  print(Jackson)}
```

3.2.1.2 for()

É similar ao `while()`, mas é mais conveniente na maioria dos casos. Iterações restritas (loops) destinam-se a execução de um número fixo de vezes dos comandos usando a função `for()`. O loop é executado tantas vezes quanto variam os valores de *i*. Começa com a palavra `for` seguido de por *parênteses e colchetes* – `for(){}.`

`()` não especifica funções lógicas como no `while`, mas especifica as interações do loop. `{}` são usados para preencher com os comandos do loop

```
for(i in conjunto_de_valores){
  comandos que serão repetidos
}
```



Fonte:tutorialspoint

```
for(i in 1:5){
  print("Aula segunda-feira é ótimo!")
}
```

```
## [1] "Aula segunda-feira é ótimo!"
## [1] "Aula segunda-feira é ótimo!"
```

```
## [1] "Aula segunda-feira é ótimo!"
## [1] "Aula segunda-feira é ótimo!"
## [1] "Aula segunda-feira é ótimo!"
```

Significa que *Aula segunda-feira é ótimo!* Será mostrada 5 vezes sempre retornando como o objeto criado *i*.

```
for(i in 1:5){
  print((i+1)*3)
}
```

```
## [1] 6
## [1] 9
## [1] 12
## [1] 15
## [1] 18
```

Não importa onde você começa ou o tipo de vetor que é (*Caractere*, *número* e etc). Basicamente toda vez que mudar ele vai executar o que estiver dentro do corpo do loop.

3.2.2 Condicionais

3.2.2.1 if, else

A função `if()` é usado para conduzir uma ou mais declarações contidas dentro de `[]`, fornecendo a condição dentro de `()`.

Voltando para nosso concurso de beleza, temos um controle um pouco mais inteligente do resultado mostrado.

```
Jackson<-9.5
Aluno<-6
if(Jackson>Aluno){
  print("Jackson é mais bonito que Aluno!")
}
```

```
## [1] "Jackson é mais bonito que Aluno!"
if(Aluno*2>=Jackson){
  print("Aluno é mais bonito!")
}
```

```
## [1] "Aluno é mais bonito!"
```

Adaptando a o código acima para fazer a declaração condicional **FALSE** simplesmente retorna nenhum comando no prompt uma vez que o código não disse como **R** deveria responder nesse caso.

```
if(Aluno*2>=Jackson^2){
  print("Aluno é mais bonito!")
```

```
}
```

Existem 2 métodos de informar ao R como responder em casos onde a declaração de condição é retornada como **FALSE**: usando o modificador **else** ou a função **ifelse()**. Ao contrário de uma função qualquer, **else** não pode ser usado independentemente, mas apenas como um modificador (adição) ao **if()**. If no exemplo acima, o resultado é solicitado quando o valor retornado é **FALSE**, **else** pode ser adicionado como segue:

```
if(Aluno*2>=Jackson^2){  
  print("A declaração é verdadeira!")  
} else{  
  print("A declaração é falsa!")  
}  
  
## [1] "A declaração é falsa!"
```

Note que **else** foi escrito na mesma linha que conclui o **if()** para informar ao R que o teste condicional continua.

Podemos retirar a segunda parte do teste anterior e adicionar uma nova comparação.

```
Aluno2<-7  
if(Aluno2>Jackson){  
  print("Aluno1 é mais bonito!")  
} else if(Aluno2>Aluno){  
  print("Aluno2 é mais bonito que Aluno e menos bonito que Jackson.")  
}  
  
## [1] "Aluno2 é mais bonito que Aluno e menos bonito que Jackson."
```

3.2.2.2 ifelse()

A função **ifelse()** é separada de **if()** e é formada de uma sintaxe mais simples e concisa. No entanto, ela pode ser usada para gerar o mesmo efeito que nas demonstrações anteriores usando **if()** com o modificador **else**.

```
ifelse(Aluno*2>=Jackson^2, "Aluno é mais bonito!", "Jackson é mais bonito!")  
  
## [1] "Jackson é mais bonito!"
```

Como mostrado acima, o uso de **ifelse()** usa a forma: **ifelse(teste condicional, resultado se verdadeiro, resultado se falso)**

Declarações condicionais também podem ser aninhadas dentro uma outra. Adaptando o exemplo com o argumento **else** um outro teste é executado se o teste inicial é **FALSE**, onde o secundário é **TRUE**.

```
ifelse(Aluno*2>=Jackson^2, "Aluno é mais bonito!",  
      ifelse(Aluno*2>=Jackson^0.5 , "Agora o Aluno é mais bonito!", "Aluno continua feio!"))
```

```
## [1] "Agora o Aluno é mais bonito!"
```

Se o teste secundário é ajustado de maneira que os resultados em um teste lógico **FALSE**, o seguinte resultado é retornado.

```
ifelse(Aluno*2>=Jackson^2, "Aluno é mais bonito!",
      ifelse(Aluno*2>=Jackson^3, "Agora o Aluno é mais bonito!", "Aluno continua feio!")

## [1] "Aluno continua feio!"
```

3.3 Dica de leitura

Spatial Data Science - Indexing

Indexing into a data structure
datamentor - R if...else Statement

3.4 Exercícios

1. Crie o vetor p que contenham uma sequência de 200 iniciando em 0 aumentando 0.5.
2. Eleve o vetor p ao cubo criando o vetor o e multiplique por p . Organize p por linhas em uma matriz com 10 colunas .
3. Crie os elementos $x = 10$ e $y = 25$.
4. Mostre na tela **A declaração é verdadeira** sob a condição de x ser menor que y .
5. Mostre 2 maneiras condicionais de executar a declaração na qual mostraria **A declaração é correta** se x é menor ou igual a y , ou caso contrário mostre **A declaração é incorreta** se não for o caso.
6. Atribua $z = 50$ e construa uma função `ifelse()` que testa se x é maior que y , mostrando **true** se a condição é encontrada. Aninhe dentro disso uma segunda condição que mostre **uma verdadeira, uma falsa ou ambas falsas** caso x seja maior que z .
7. Use uma iteração restrita para gerar uma sequência de números de 1 até 20.
8. Use iteração restrita para conduzir o cálculo $(i^2)-i$, onde i vai de 5 até 10.
9. Use interação não restrita para gerar uma sequência de inteiros de -10 até -20.