

Curso de R para Meteorologia IAG/USP

Sergio Ibarra-Espinosa, Amanda Rehbein, Daniel Schuch, Camila Lopes, Isabela Siqueira, e possivelmente outros (você está convidado para colaborar)

2018-06-06

Contents

1	Pré-Requisitos	5
1.1	Sistema Operacional	5
1.2	Pacotes usados neste curso	5
1.3	Dados usados neste curso	6
1.4	Colaborar	6
1.5	Compartilhar dados	6
2	Intro	7
2.1	IMPORTANTE	7
3	R!	11
3.1	Variáveis Básicas	11
3.2	Vetores	12
3.3	Classe	12
3.4	Converter objetos	12
3.5	Array	13
3.6	Matrizes e a função <code>matrix</code>	14
3.7	Listas	14
3.8	Data Frames	15
3.9	Tempo e Data	16
3.10	Fatores	17
4	Importando e Exportando Dados	19
4.1	Base	19
4.2	Tidyverse	29
4.3	Outros Tipos de Dados	35
5	Plotando	43
5.1	<code>plot</code> (base)	43
5.2	<code>image</code>	46
5.3	<code>ggplot</code> (<code>ggplot2</code>)	48
6	Estruturas de Controle	59
6.1	<code>if-else</code>	59
6.2	<code>for</code>	59
6.3	<code>while</code>	59
6.4	<code>repeat</code>	59
6.5	<code>lapply</code>	59
6.6	<code>sapply</code>	59
6.7	<code>split</code>	59
6.8	<code>tapply</code>	59
6.9	<code>apply</code>	59

6.10 mapply	59
7 De Scripts a Funções e de Funções a Pacotes	61
8 Geo Spatial: raster, sf e stars	63

Chapter 1

Pré-Requisitos

1.1 Sistema Operacional

Antes de instalar o R na sua plataforma de interesse, verifique se há recomendações abaixo:

Windows

A princípio não há pré-requisitos! Caso fique entusiasmado com o R e queira desenvolver os próprios pacotes, instale o Rtools <https://cran.r-project.org/bin/windows/Rtools/>

Instale NetCDF, GDAL, GEOS, udunits e PROJ

MacOS

```
brew unlink gdal
brew tap osgeo/osgeo4mac && brew tap --repair
brew install proj
brew install geos
brew install udunits
brew install gdal2 --with-armadillo --with-complete --with-libkml --with-unsupported
brew link --force gdal2
```

(Veja como instalar NetCDF no MacOS)

Linux (Ubuntu e derivados)

```
sudo add-apt-repository ppa:ubuntugis/ubuntugis-unstable --yes
sudo apt-get --yes --force-yes update -qq
# units/udunits2 dependency:
sudo apt-get install --yes libudunits2-dev
# sf dependencies:
sudo apt-get install --yes libproj-dev libgeos-dev libgdal-dev libnetcdf-dev netcdf-bin gdal-bin
```

1.2 Pacotes usados neste curso

Para fazer este curso instale os seguintes pacotes como indicado:

```
check.packages <- function(pkg){
  new.pkg <- pkg[!(pkg %in% installed.packages() [, "Package"])]
  if (length(new.pkg))
    install.packages(new.pkg, dependencies = TRUE)
```

```

  sapply(pkg, require, character.only = TRUE)
}

# Usage example
packages <- c("devtools", "tidyverse", "reshape2", "sf",
            "maptools", "mapview", "fields", "raster",
            "sp", "rgdal", "ncdf4", "data.table",
            "openair", "cptcity")
check.packages(packages)
devtools::install_github("atmoschem/veinreport")

```

Fonte: <https://gist.github.com/smithdanielle/9913897>

Descrição de alguns desses pacotes:

- devtools permite a instalação de versões de desenvolvimento de pacotes de diferentes repositórios
- tidyverse é o universo de pacotes do Hadley Wickham para tratamento e visualização de dados
 - Se você quiser plotar os objetos espaciais sf com o pacote ggplot2 (que faz parte do tidyverse), ele precisa ser instalado usando o devtools (`devtools::install_github("tidyverse/ggplot2")`), pois a função geom_sf ainda não está disponível na versão oficial
- sf, mapview, raster, sp, rgdal, maptools e fields tratam dados espaciais. Lembre-se que os objetos em Meteorologia são espaço-temporais
- ncdf4 é um pacote para manipular arquivos NetCDF
- openair é um pacote para trabalhar com dados de qualidade do ar e Meteorologia
- cptcity é um pacote que tem 7140 paletas de cores do arquivo web cpt-city

Preste atenção na instalação dos pacotes pois eles podem precisar de dependências do sistema.

1.3 Dados usados neste curso

Os exemplos mostrados neste curso usarão os dados que vocês podem baixar em: <https://github.com/iagdevs/cursoR/tree/master/dados>

1.4 Colaborar

A melhor forma de colaboração é com *pull requests* no repositório do curso. Aplique o Guia de Estilo de R do Google ou o formato formatR. Em poucas palavras, lembre que seu código vai ser lido por seres humanos. É possível editar qualquer página usando um dos botões acima.

1.5 Compartilhar dados

Se você conhece alguma fonte de dados para deixar este curso mais legal, edite este arquivo e faça um *pull request*.

1. NCEP: ftp://nomads.ncdc.noaa.gov/GFS/analysis_only/
2. ...
3. ...

Chapter 2

Intro

Este curso é voltado para os alunos de pós-graduação, dessa forma, veremos os conceitos rapidamente. Caso não haja tempo, o conteúdo ficará online no link: <https://github.com/iagdevs/cursoR>.

Sempre que tiver uma dúvida, tente utilizar: BASE.

Outros pacotes BASE: utils, stats, datasets, graphics, grDevices, grid, methods, tools, parallel, compiler, splines, tcltk , stats4.

Acesse aqui a lista de pacotes disponíveis.

Este curso foi baseado no livro R Programming for Data Science e possui exercícios a serem resolvidos, perguntas que ajudam a entender conceitos e desafios para aprofundar os conhecimentos adquiridos.

Neste curso iremos utilizar o software RStudio. A imagem abaixo resume um pouco das funcionalidades disponíveis.

Além de poder baixar o programa utilizado nesse curso, você também pode acessar muitos materiais como as Cheatsheets e Webnários que cobrem desde itens básicos como funções essenciais do RStudio até desenvolvimento de páginas para Web ou apps com Shiny. Explore o máximo que puder!

Dicas:

- Se não souber usar uma função, escreva: `?função`
- As funções tem argumentos, use **TAB** para vê-los numa função

2.1 IMPORTANTE

• TAB no RStudio

Isso te ajudará a evitar coisas como: grafia errada da função, verificar se a função existe, verificar argumentos, etc... Use sempre!

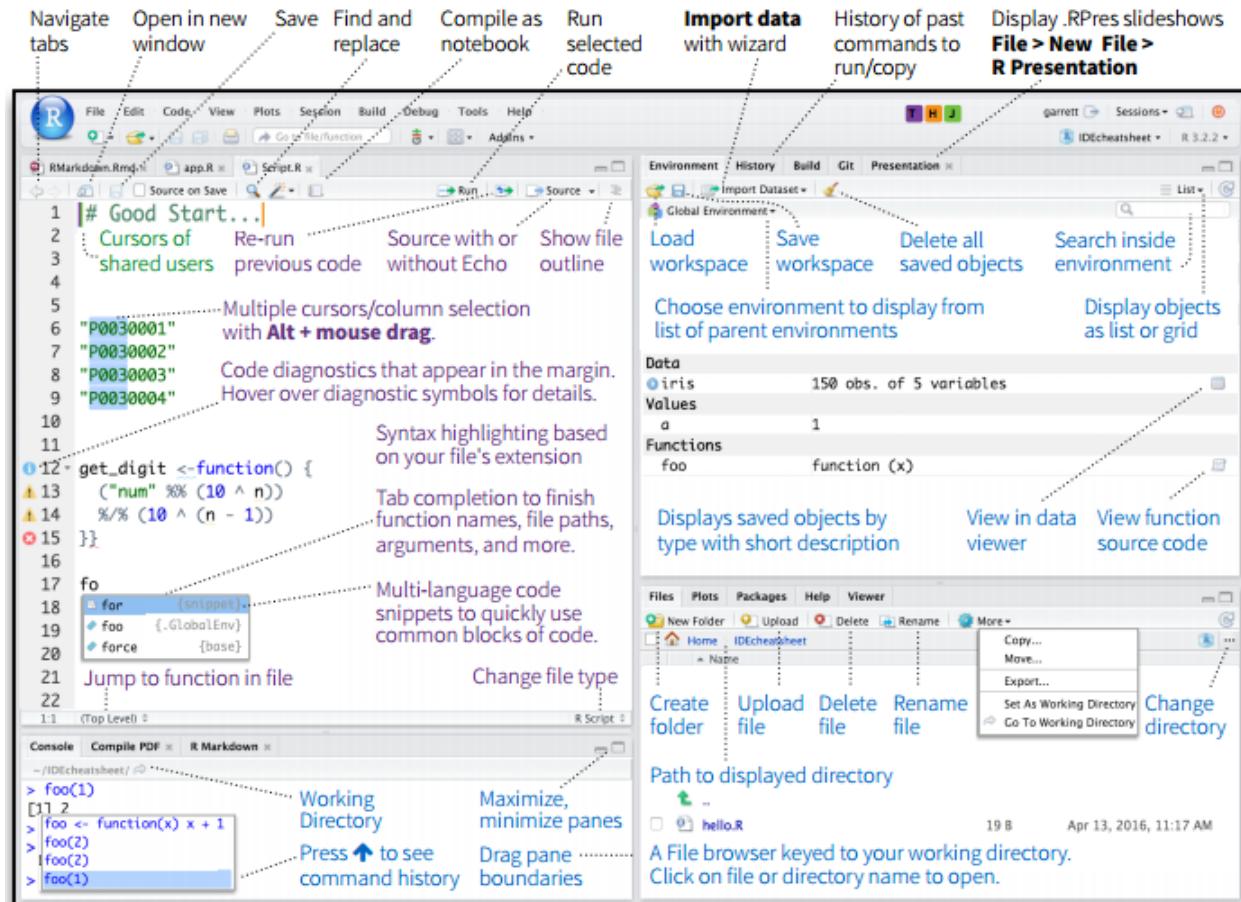


Figure 2.1: Fonte: RStudio IDE Cheatsheet



- **Stack Overflow** (Veja as últimas das mais de 240000 (!!!) perguntas sobre R)

Melhor forma de resolver problemas! Acredite, é praticamente impossível existir um problema que outra pessoa não tenha resolvido pelo Stack Overflow.

*Eu mesma só precisei fazer **uma** pergunta no Stack Overflow (e acabei respondendo eu mesma depois de resolver) usando R há uns 5 anos! - Camila.*

Vamos começar!

Chapter 3

R!

A linguagem R pode ser usada desde uma simples calculadora até uma poderosa ferramenta estatística, seja para análise de dados, seja para machine learning.

Nesse capítulo veremos o básico da linguagem. Como a ideia é cobrir o máximo de conteúdo possível, passaremos bem rápido pelos conceitos básicos, com alguns exercícios para melhor entendimento.

3.1 Variáveis Básicas

Podemos armazenar valores, strings ou operadores lógicos nas chamadas variáveis básicas. É com elas que podemos fazer operações básicas e transformar o RStudio em uma calculadora superpoderosa. Existem 5 tipos de variáveis básicas, sendo elas:

- Numeric (1): valores numéricos com ou sem casa decimal

```
5+2.3
```

```
## [1] 7.3
```

```
pi
```

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

- Character (a): com eles podemos armazenar strings, como o título de um gráfico

```
titulo <- "Isso é uma string"
```

```
titulo
```

```
## [1] "Isso é uma string"
```

- Integer (1): são valores inteiros

- Complex (0+1i): também é possível armazenar valores complexos nas variáveis básicas

```
3 + 4i
```

```
## [1] 3+4i
```

- Logical (TRUE): são os famosos operadores Booleanos, que permitem realizar comparações entre variáveis ou dados (**Você não pode criar uma variável com os nomes TRUE ou FALSE, esses nomes são reservados pelo R**)

```
1 == 2
```

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

3.2 Vetores

Vetores permitem que você armazene dados em uma sequência 1D de qualquer um dos tipos listados nas variáveis básicas, mais o formato “cru” (raw) que é o modo de armazenamento de bytes. Por exemplo:

- `c("A", "C", "D")`
- `1:5 = c(1, 2, 3, 4, 5)`
- `c(TRUE, FALSE)`
- `c(1i, -1i)`

Importante: ao contrário do C ou do Python, na linguagem R, a contagem das posições dos vetores começa do 1 e NÃO do zero!

3.3 Classe

Como foi possível notar, todas as variáveis pertencem a alguma classe, dessa forma, a função `class` permite descobrir qual a classe da variável que se está utilizando:

```
x <- c(1,2,3)
class(x)

## [1] "numeric"
```

Exercício: Qual a classe dos seguintes vetores?

```
c(1, "C", "D")
c(1, NA, "D")
c(1, NA, NaN)
```

3.4 Converter objetos

Às vezes quando trabalhamos com dados, podemos precisar “arredondar” valores ou converter vetores em listas, para isso existem algumas funções especiais.

3.4.1 as

Um modo de forçar um objeto a assumir outra classe é por meio da função `as`:

```
as.integer(c(1.5, 2.9, 1))

## [1] 1 2 1
```

Note que a função apenas converte os números de decimais para inteiros, sem arredondar para o número mais próximo.

Pergunta: O que acontece quando se tenta converter o seguinte vetor?

```
as.numeric(c(1, "C", "D"))
```

3.4.2 `merge` e `melt`

Nem sempre os conjuntos de dados que você encontrar pela vida estarão no formato desejado para plotar e/ou analisar estatisticamente, dessa forma, essas duas funções poderão ajudar na sua jornada:

- `merge`: permite a união entre dois data frames, seja por colunas em comum ou linhas em comum
- `melt`: do pacote reshape2, permite que você agrupe várias colunas em função de outra coluna em comum, de acordo com o nome especificado

3.5 Array

Ao contrário do vetor unidimensional, arrays permitem que você armazene dados em diversas dimensões, sendo todas com o mesmo comprimento.

Vamos dar uma olhada nos argumentos da função:

```
args(array)

## function (data = NA, dim = length(data), dimnames = NULL)
## NULL
```

Dessa forma, é preciso “informar” ao R qual o número de dimensões que você quer no seu array:

```
a <- array(data = 0, dim = c(1,1))
class(a)
```

```
## [1] "matrix"
```

No caso acima, como só foram designadas duas dimensões, o array é igual a uma matriz.

```
a <- array(data = 0, dim = c(3,3,1))
class(a)
```

```
## [1] "array"
```

Como dá pra ver acima, é possível armazenar diversos elementos em um array, como por exemplo as dimensões que utilizamos no dia-a-dia de modelos numéricos: espaço (x,y,z) e tempo (z). Dessa forma, podemos criar arrays a partir de vetores e armazená-los em diversas dimensões.

```
vetor1 <- c(1,2,3,4,5)
vetor2 <- c(10,12,14,16,18,20,22,24)

a <- array(data = c(vetor1,vetor2), dim = c(3,3,2))
class(a)

## [1] "array"
```

Se você quiser, também é possível nomear as colunas e linhas do seu array:

```
colunas <- c("col1", "col2", "col3")
linhas <- c("lin1", "lin2", "lin3")

array(data = c(vetor1, vetor2), dim = c(3, 3, 2), dimnames = list(linhas, colunas))
```

```
## , , 1
##
##      col1 col2 col3
## lin1    1    4   12
## lin2    2    5   14
```

```
## lin3    3   10   16
##
## , , 2
##
##      col1 col2 col3
## lin1    18   24    3
## lin2    20    1    4
## lin3    22    2    5
```

Além disso, sempre que precisar acessar elementos do seu array é só especificar as dimensões como para mostrar o elemento de um vetor.

```
a[1,2,2] #(linha, coluna, matriz)
## [1] 24
```

Exercício: Crie um array com 3 dimensões, contendo três linhas e 4 quatro colunas. Acesse o elemento da segunda linha e terceira coluna desse array. Não esqueça de verificar a classe desse objeto!

3.6 Matrizes e a função matrix

Uma matriz é um array com duas dimensões, sendo necessário informar o número de colunas e linhas, mas não o de dimensões.

Assim como em arrays, só são permitidos elementos **da mesma classe!**

Argumentos da função **matrix**:

```
args(matrix)
```

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

Colocando dados em uma matriz:

```
m1 <- matrix(data = 1:(4*4), nrow = 4, ncol = 4)
dim(m1)
```

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

Por padrão, a opção “byrow” é igual a **FALSE**. Quando passamos para **TRUE**, é possível organizar os dados por linha.

```
m2 <- matrix(data = 1:(4*4), nrow = 4, ncol = 4, byrow = TRUE)
```

Exercício: Construa uma matriz com três linhas que contenha os números de 1 a 9.

3.7 Listas

Já as listas permitem que você armazene qualquer tipo de variável básica, independente da classe. Dessa forma, podemos colocar numa lista: número, caracteres, argumentos lógicos, ou que você quiser:

```
list(list(list(1)))

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

Isso faz com que elas sejam bastante versáteis e sirvam para armazenar o que você precisar, mas elas só podem ter uma dimensão, como uma fila. Já os objetos armazenados dentro da lista não precisam ter a mesma dimensão.

```
x <- list(1, "a", TRUE, 1 + 4i)
```

Exercício: Crie uma lista contendo um vetor, uma matriz e um data frame e acesse o segundo elemento dela.

Para facilitar, já vamos te dar o data frame:

```
my_df <- mtcars[1:10,]
```

3.8 Data Frames

Os data frames são uma forma de armazenar seus dados em um formato parecido com uma planilha de excel. Você pode pensar em um data frame como uma matriz que armazena em cada coluna um dado diferente, ou como uma lista onde todos os elementos tem o mesmo comprimento.

```
df <- data.frame(a = 1:3)
names(df)
```

```
## [1] "a"
class(df)

## [1] "data.frame"
mode(df)

## [1] "list"
```

É normalmente em um data frame que você importará os seus dados e vale saber como visualizar algumas informações básicas sobre ele direto no seu console. Para isso, vamos pegar como exemplo o conjunto `mtcars` da base de dados do R:

```
df <- mtcars
head(df) #mostra as sete primeiras linhas do data frame
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
## Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
## Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
## Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
## Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
## Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

Para ver as últimas linhas do data frame basta usar a função `tail`. Já uma função muito útil é a `summary`, que apresenta um “resumo” dos seus dados, como média, mediana, mínimos e máximos para cada `coluna` do data frame.

```
summary(df)
```

```
##      mpg          cyl         disp        hp
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
##  1st Qu.:15.43  1st Qu.:4.000  1st Qu.:120.8  1st Qu.: 96.5
##  Median :19.20  Median :6.000  Median :196.3  Median :123.0
##  Mean   :20.09  Mean   :6.188  Mean   :230.7  Mean   :146.7
##  3rd Qu.:22.80  3rd Qu.:8.000  3rd Qu.:326.0  3rd Qu.:180.0
##  Max.   :33.90  Max.   :8.000  Max.   :472.0  Max.   :335.0
##      drat         wt         qsec        vs
##  Min.   :2.760   Min.   :1.513   Min.   :14.50  Min.   :0.0000
##  1st Qu.:3.080  1st Qu.:2.581  1st Qu.:16.89  1st Qu.:0.0000
##  Median :3.695  Median :3.325  Median :17.71  Median :0.0000
##  Mean   :3.597  Mean   :3.217  Mean   :17.85  Mean   :0.4375
##  3rd Qu.:3.920  3rd Qu.:3.610  3rd Qu.:18.90  3rd Qu.:1.0000
##  Max.   :4.930  Max.   :5.424  Max.   :22.90  Max.   :1.0000
##      am          gear        carb
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000
##  1st Qu.:0.0000  1st Qu.:3.000  1st Qu.:2.000
##  Median :0.0000  Median :4.000  Median :2.000
##  Mean   :0.4062  Mean   :3.688  Mean   :2.812
##  3rd Qu.:1.0000  3rd Qu.:4.000  3rd Qu.:4.000
##  Max.   :1.0000  Max.   :5.000  Max.   :8.000
```

Iremos trabalhar bastante com data frames daqui pra frente, eles se tornarão aliados muito poderosos.

3.9 Tempo e Data

O R trabalha com três classes de tempo: `POSIXct`, `POSIXlt` e `Date`, sendo que `POSIXct` se refere ao número de segundos desde o início de 1970 no modo UTC, enquanto que `POSIXlt` armazena as datas como uma lista, contendo segundos, minutos, horas, dias, meses, etc.

```
a <- ISOdate(year = 2018, month = 4, day = 5)
class(a)

## [1] "POSIXct" "POSIXt"

b <- ISOdate(year = 2018, month = 4, day = 5, tz = "Americas/Sao_Paulo")
```

Já a classe `Date`, armazena as datas como o número de dias contados a partir de 1970.

```
c <- as.Date(Sys.time())
class(c)

## [1] "Date"
```

Caso você precise, o pacote `nanotime` permite trabalhar com nano segundos.

Também é possível fazer sequências:

```
hoje <- Sys.time()
a <- seq.POSIXt(from = hoje, by = 3600, length.out = 24)
```

Funções úteis: `weekdays`, `month` e `julian`

```

weekdays(a)

## [1] "quarta" "quarta" "quarta" "quarta" "quarta" "quarta" "quinta"
## [8] "quinta" "quinta" "quinta" "quinta" "quinta" "quinta" "quinta"
## [15] "quinta" "quinta" "quinta" "quinta" "quinta" "quinta" "quinta"
## [22] "quinta" "quinta" "quinta"

months(a)

## [1] "junho" "junho" "junho" "junho" "junho" "junho" "junho" "junho"
## [9] "junho" "junho" "junho" "junho" "junho" "junho" "junho" "junho"
## [17] "junho" "junho" "junho" "junho" "junho" "junho" "junho" "junho"

julian(a) #dia Juliano

## Time differences in days
## [1] 17688.91 17688.95 17688.99 17689.03 17689.07 17689.11 17689.16
## [8] 17689.20 17689.24 17689.28 17689.32 17689.36 17689.41 17689.45
## [15] 17689.49 17689.53 17689.57 17689.61 17689.66 17689.70 17689.74
## [22] 17689.78 17689.82 17689.86
## attr(,"origin")
## [1] "1970-01-01 GMT"

```

O formato POSIXct é o mais comumente usado principalmente se os dados analisados serão plotados.

3.10 Fatores

Os **factors** podem ser um pouco infernais. Dê uma olhada em R INFERNO.

Usados em análise estatística, fatores são usados para armazenar variáveis categóricas, ou seja, é uma variável que pode pertencer a um número limitado de categorias, como por exemplo, dias da semana. Já uma variável contínua pode assumir um número infinito de valores.

```

a <- seq.POSIXt(from = hoje , by = 3600, length.out = 24*7)
aa <- weekdays(a)
class(aa)

## [1] "character"

factor(aa)

## [1] quarta quarta quarta quarta quarta quarta quarta quinta quinta
## [9] quinta quinta quinta quinta quinta quinta quinta quinta quinta
## [17] quinta quinta quinta quinta quinta quinta quinta sexta sexta
## [25] quinta quinta quinta quinta quinta quinta sexta sexta
## [33] sexta sexta sexta sexta sexta sexta sexta sexta sexta
## [41] sexta sexta sexta sexta sexta sexta sexta sexta sexta
## [49] sexta sexta sexta sexta sexta sexta sábado sábado
## [57] sábado sábado sábado sábado sábado sábado sábado sábado
## [65] sábado sábado sábado sábado sábado sábado sábado sábado
## [73] sábado sábado sábado sábado sábado domingo domingo
## [81] domingo domingo domingo domingo domingo domingo domingo domingo
## [89] domingo domingo domingo domingo domingo domingo domingo domingo
## [97] domingo domingo domingo domingo domingo segunda segunda
## [105] segunda segunda segunda segunda segunda segunda segunda segunda
## [113] segunda segunda segunda segunda segunda segunda segunda segunda
## [121] segunda segunda segunda segunda segunda terça terça

```

```
## [129] terça  terça  terça  terça  terça  terça  terça  terça  terça
## [137] terça  terça  terça  terça  terça  terça  terça  terça  terça
## [145] terça  terça  terça  terça  terça  terça  quarta  quarta
## [153] quarta quarta quarta quarta quarta quarta quarta quarta quarta
## [161] quarta quarta quarta quarta quarta quarta quarta quarta quarta
## Levels: domingo quarta quinta sábado segunda sexta terça
```

São muito úteis para regressões, gráficos e resumos estatísticos, uma vez que limita o número de possibilidades para a qual o dado pertença. Além disso, é possível estabelecer “níveis” que vão designar a categoria do seu dado.

```
ab <- factor(x = aa,
              levels = c("Monday", "Tuesday", "Wednesday", "Thursday",
                        "Friday", "Saturday", "Sunday"))
levels(ab)

## [1] "Monday"     "Tuesday"    "Wednesday"  "Thursday"   "Friday"     "Saturday"
## [7] "Sunday"
```

Exercício: Converta o vetor abaixo em um fator e mostre os seus níveis

```
genero <- c("Masculino", "Masculino", "Feminino", "Masculino", "Feminino", "Feminino")
```

Se tudo pareceu muito corrido, não se preocupe, todos esses conceitos serão praticados mais adiante!

```
##-- Opção para deixar possível visualizar erros
knitr::opts_chunk$set(error = TRUE)
```

Chapter 4

Importando e Exportando Dados

4.1 Base

4.1.1 Importando com `read.*`

Como vimos anteriormente, na maioria dos casos, iremos usar data frames para lidar com dados em R, sendo assim, podemos utilizar os seguintes modos de leitura:

- `read.csv`
- `read.csv2`
- `read.table`

Vamos ler alguns dados usando `read.table`. Para saber o que a função faz, use `?read.table`. Os argumentos da função são:

```
args(read.table)
```

```
## function (file, header = FALSE, sep = "", quote = "\"\"", dec = ".",
## numerals = c("allow.loss", "warn.loss", "no.loss"), row.names,
## col.names, as.is = !stringsAsFactors, na.strings = "NA",
## colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
## fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE,
## comment.char = "#", allowEscapes = FALSE, flush = FALSE,
## stringsAsFactors = default.stringsAsFactors(), fileEncoding = "",
## encoding = "unknown", text, skipNul = FALSE)
## NULL
```

O terceiro argumento é `sep`, definido como `" "` por padrão. Esse argumento indica para o R qual o tipo de separador utilizado entre as colunas dos dados.

```
df <- read.table("dados/NOXIPEN2014.txt")
```

Lembre-se que as funções `head` e `tail` permitem que você veja as primeiras ou últimas linhas do data frame.

```
head(df)
```

```
##   TipodeRede TipodeMonitoramento      Tipo     Data    Hora
## 2 Automático          CETESB Dados Primários 01/01/2014 01:00
## 3 Automático          CETESB Dados Primários 01/01/2014 02:00
## 4 Automático          CETESB Dados Primários 01/01/2014 03:00
```

```

## 5 Automático          CETESB Dados Primários 01/01/2014 04:00
## 6 Automático          CETESB Dados Primários 01/01/2014 05:00
## 7 Automático          CETESB Dados Primários 01/01/2014 06:00
##   CodigoEstação      NomeEstação           NomeParâmetro
## 2                  95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 3                  95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 4                  95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 5                  95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 6                  95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 7                  95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
##   UnidadedeMedida MediaHoraria MediaMovel Valido
## 2            ppb        9       -    Não
## 3            ppb        9       -    Sim
## 4            ppb        5       -    Sim
## 5            ppb        4       -    Sim
## 6            ppb        5       -    Sim
## 7            ppb        5       -    Sim
tail(df)

##      TipodeRede TipodeMonitoramento      Tipo      Data     Hora
## 8577 Automático          CETESB Dados Primários 01/01/2015 19:00
## 8578 Automático          CETESB Dados Primários 01/01/2015 20:00
## 8579 Automático          CETESB Dados Primários 01/01/2015 21:00
## 8580 Automático          CETESB Dados Primários 01/01/2015 22:00
## 8581 Automático          CETESB Dados Primários 01/01/2015 23:00
## 8582 Automático          CETESB Dados Primários 01/01/2015 24:00
##   CodigoEstação      NomeEstação           NomeParâmetro
## 8577          95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 8578          95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 8579          95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 8580          95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 8581          95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 8582          95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
##   UnidadedeMedida MediaHoraria MediaMovel Valido
## 8577        ppb        3       -    Sim
## 8578        ppb        8       -    Sim
## 8579        ppb       11      -    Sim
## 8580        ppb       11      -    Sim
## 8581        ppb       16      -    Sim
## 8582        ppb       NA      -    Sim

```

Vamos tentar ler outra versão dos mesmos dados utilizando a mesma função `read.table`:

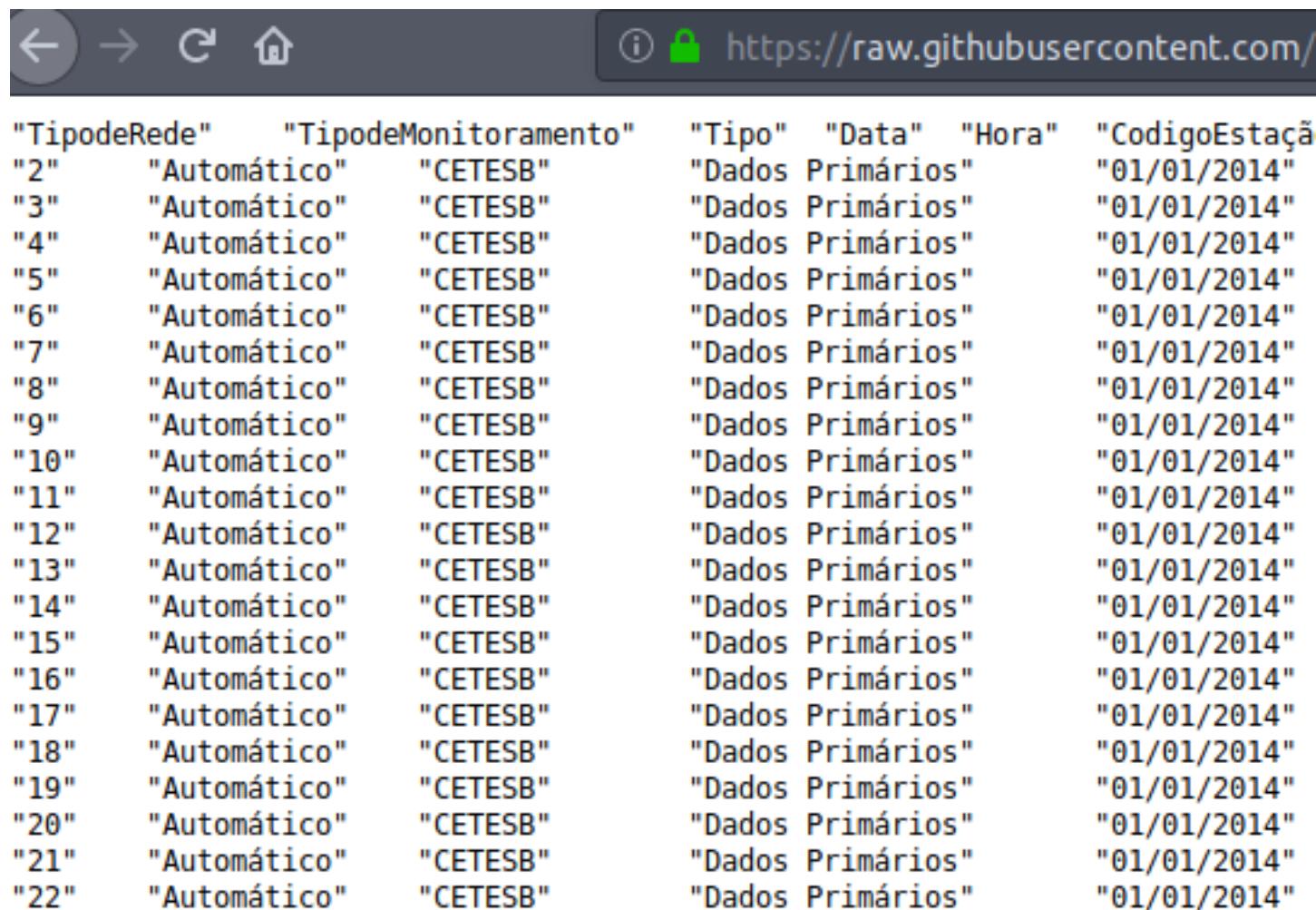
```
df2 <- read.table("dados/NOXIPEN2014v2.txt")
```

```
## Error in scan(file = file, what = what, sep = sep, quote = quote, dec = dec, : linha 1 não tinha 6 e
```

Apareceu uma mensagem de erro, você saberia dizer o porquê?

Caso você trabalhe com algum banco de dados em formato .txt e quiser abrir no R... Abra-os no bloco de notas primeiro!

Vamos dar uma olhada em alguns dados, o primeiro tem uma cara assim:



The screenshot shows a web browser window with the URL <https://raw.githubusercontent.com/>. The page displays a table of data with four columns: "TipodeRede", "TipodeMonitoramento", "Tipo", and "CodigoEstação". The data consists of 22 rows, each containing the value "Automático" for the first two columns and "CETESB" for the third column. The fourth column contains the values "Dados Primários" and the date "01/01/2014" repeated 22 times.

"TipodeRede"	"TipodeMonitoramento"	"Tipo"	"CodigoEstação"
"2"	"Automático"	"CETESB"	"Dados Primários"
"3"	"Automático"	"CETESB"	"Dados Primários"
"4"	"Automático"	"CETESB"	"Dados Primários"
"5"	"Automático"	"CETESB"	"Dados Primários"
"6"	"Automático"	"CETESB"	"Dados Primários"
"7"	"Automático"	"CETESB"	"Dados Primários"
"8"	"Automático"	"CETESB"	"Dados Primários"
"9"	"Automático"	"CETESB"	"Dados Primários"
"10"	"Automático"	"CETESB"	"Dados Primários"
"11"	"Automático"	"CETESB"	"Dados Primários"
"12"	"Automático"	"CETESB"	"Dados Primários"
"13"	"Automático"	"CETESB"	"Dados Primários"
"14"	"Automático"	"CETESB"	"Dados Primários"
"15"	"Automático"	"CETESB"	"Dados Primários"
"16"	"Automático"	"CETESB"	"Dados Primários"
"17"	"Automático"	"CETESB"	"Dados Primários"
"18"	"Automático"	"CETESB"	"Dados Primários"
"19"	"Automático"	"CETESB"	"Dados Primários"
"20"	"Automático"	"CETESB"	"Dados Primários"
"21"	"Automático"	"CETESB"	"Dados Primários"
"22"	"Automático"	"CETESB"	"Dados Primários"

Já o segundo arquivo é assim:



The screenshot shows a web browser window with the URL <https://raw.githubusercontent.com/> in the address bar. The page content is a large block of text representing a dataset. The first few lines of the data are:

```
"TipodeRede"; "TipodeMonitoramento"; "Tipo"; "Data"; "Hora"; "CodigoEstação"; "Nome"
"2"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "01:00"; 95; "Cid.Univ
"3"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "02:00"; 95; "Cid.Univ
"4"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "03:00"; 95; "Cid.Univ
"5"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "04:00"; 95; "Cid.Univ
"6"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "05:00"; 95; "Cid.Univ
"7"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "06:00"; 95; "Cid.Univ
"8"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "07:00"; 95; "Cid.Univ
"9"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "08:00"; 95; "Cid.Univ
"10"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "09:00"; 95; "Cid.Univ
"11"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "10:00"; 95; "Cid.Univ
"12"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "11:00"; 95; "Cid.Univ
"13"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "12:00"; 95; "Cid.Univ
"14"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "13:00"; 95; "Cid.Univ
"15"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "14:00"; 95; "Cid.Univ
"16"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "15:00"; 95; "Cid.Univ
"17"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "16:00"; 95; "Cid.Univ
"18"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "17:00"; 95; "Cid.Univ
"19"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "18:00"; 95; "Cid.Univ
"20"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "19:00"; 95; "Cid.Univ
"21"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "20:00"; 95; "Cid.Univ
"22"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "21:00"; 95; "Cid.Univ
"23"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "22:00"; 95; "Cid.Univ
"24"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "23:00"; 95; "Cid.Univ
"25"; "Automático"; "CETESB"; "Dados Primários"; "01/01/2014"; "24:00"; 95; "Cid.Univ
"26"; "Automático"; "CETESB"; "Dados Primários"; "02/01/2014"; "01:00"; 95; "Cid.Univ
"27"; "Automático"; "CETESB"; "Dados Primários"; "02/01/2014"; "02:00"; 95; "Cid.Univ
"28"; "Automático"; "CETESB"; "Dados Primários"; "02/01/2014"; "03:00"; 95; "Cid.Univ
"29"; "Automático"; "CETESB"; "Dados Primários"; "02/01/2014"; "04:00"; 95; "Cid.Univ
"30"; "Automático"; "CETESB"; "Dados Primários"; "02/01/2014"; "05:00"; 95; "Cid.Univ
```

Você notou a diferença? Como vimos anteriormente, para importar os dados no R é super importante que você especifique o tipo de separador utilizado. Como o segundo arquivo é separado por “;”, precisamos especificar o argumento `sep` na hora de usar o comando `read.table`:

```
df2 <- read.table("dados/NOXIPEN2014v2.txt", sep = ";")
head(df2)
```

```
##   TipodeRede TipodeMonitoramento      Tipo      Data     Hora
## 2 Automático          CETESB Dados Primários 01/01/2014 01:00
## 3 Automático          CETESB Dados Primários 01/01/2014 02:00
## 4 Automático          CETESB Dados Primários 01/01/2014 03:00
## 5 Automático          CETESB Dados Primários 01/01/2014 04:00
## 6 Automático          CETESB Dados Primários 01/01/2014 05:00
```

```

## 7 Automático          CETESB Dados Primários 01/01/2014 06:00
##   CódigoEstação      NomeEstação      NomeParâmetro
## 2                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 3                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 4                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 5                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 6                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 7                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
##   UnidadedeMedida MediaHoraria MediaMovel Valido
## 2             ppb      9      -    Não
## 3             ppb      9      -    Sim
## 4             ppb      5      -    Sim
## 5             ppb      4      -    Sim
## 6             ppb      5      -    Sim
## 7             ppb      5      -    Sim

tail(df2)

##       TipodeRede TipodeMonitoramento      Tipo      Data     Hora
## 8577 Automático          CETESB Dados Primários 01/01/2015 19:00
## 8578 Automático          CETESB Dados Primários 01/01/2015 20:00
## 8579 Automático          CETESB Dados Primários 01/01/2015 21:00
## 8580 Automático          CETESB Dados Primários 01/01/2015 22:00
## 8581 Automático          CETESB Dados Primários 01/01/2015 23:00
## 8582 Automático          CETESB Dados Primários 01/01/2015 24:00
##   CódigoEstação      NomeEstação      NomeParâmetro
## 8577         95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 8578         95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 8579         95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 8580         95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 8581         95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 8582         95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
##   UnidadedeMedida MediaHoraria MediaMovel Valido
## 8577       ppb      3      -    Sim
## 8578       ppb      8      -    Sim
## 8579       ppb     11      -    Sim
## 8580       ppb     11      -    Sim
## 8581       ppb     16      -    Sim
## 8582       ppb      NA      -    Sim

```

Além desses comandos, vocês também pode utilizar a opção **Import Dataset** do RStudio, permitindo que você tenha um “preview” dos dados - como no Excel, mas melhor!

Para mais informações sobre importar dados no R, dê uma olha nesse webnário.

4.1.2 Exportando com `write.table`

Exportar é bem facil. Vamos dar uma olhada nos argumentos da função `write.table`:

```
args(write.table)
```

```

## function (x, file = "", append = FALSE, quote = TRUE, sep = " ",
##          eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE,
##          qmethod = c("escape", "double"), fileEncoding = ""),
##          NULL

```

Se temos um data frame com colunas de classe character, `quote = TRUE` quer dizer que o arquivo de texto resultante vai ter aspas nas colunas de caractere. Novamente, o argumento `sep` indica como podemos separar as colunas. Se você quiser abrir esses dados no excel, uma boa opção é utilizar os separadores “,”;/“;”/“/”⁶, sendo o último o separador que indica o espaçamento criado pela tecla TAB.

Já o argumento `eol` quer dizer *end of line*, e é uma forma de dizer ao R que a linha acaba ali. Por padrão, a opção `row.names` vem com a opção TRUE, mas sempre coloque a opção FALSE, caso contrário, será adicionada uma coluna com os índices das linhas. O argumento `col.names` indica se você quer nomear as suas colunas, o que é sempre uma boa ideia.

Exercício: Usando o conjunto de dados `mtcars` da base do R, exporte-o de uma forma que ele possa ser lido em algum Excel genérico. Não esqueça de usar o que foi ensinado acima.

4.1.3 Exportando objetos com `save`

Podemos salvar objetos do R com o comando `save`. Ele permite que você recarregue o objeto mais tarde.

```
args(save)
```

```
## function (... , list = character() , file = stop("'file' must be specified") ,
##      ascii = FALSE , version = NULL , envir = parent.frame() , compress = iSTRUE(!ascii) ,
##      compression_level , eval.promises = TRUE , precheck = TRUE)
## NULL
```

Essa função salva o objeto com a extensão .rda sendo que para carregá-lo de volta usamos a função `load`

```
args(load)
```

```
## function (file , envir = parent.frame() , verbose = FALSE)
## NULL
```

Muito cuidado ao utilizar esse comando, pois é bem possível cometer alguns deslizes, como trocar o nome do objeto. Veja esse exemplo abaixo:

```
#Primeiro vamos ler os dados do dia 01/06/2016 para uma estação automática do INMET de Paracatu:
paracatu <- read.csv(file = "dados/paracatu.csv" , sep = ",") #lendo como csv

# Vamos dizer que queremos salvar apenas a coluna correspondente à temperatura máxima horária:
paracatu_temp <- paracatu$temp_max

#Agora vamos salvar o objeto do tipo numeric com o nome especificado:
save(paracatu_temp , file = "Temp_max.rda")
```

Passado um tempo, queremos acessar de volta esse objeto, mas não lembramos se salvamos a temperatura máxima ou mínima, vamos confiar que foi a mínima:

```
load("Temp_min.rda")
```

E assim descobrimos que não era... O que nos ensina a sempre guardar na memória quais variáveis que salvamos no ambiente R. O bom dessa função é que ela permite salvar com tipos de compressão, por exemplo `compress = "xz"`.

4.1.4 Exportando objetos com `saveRDS`

Esta é uma das minhas funções favoritas no R, veja só o porquê:

```
saveRDS(paracatu_temp, "Temperatura.rds")
frenteQ <- readRDS("Temperatura.rds")
```

Você pode salvar seu objeto R de forma serializada e compactada com o argumento `compress` e na hora de chamar o objeto de volta é só usar `readRDS` e colocar o nome que você quiser.

4.1.5 Processando nossa data frame

Vamos revisar a classe de cada coluna do nosso data-frame com a função `sapply`, que será explicada em outro capítulo. Lembre-se, qualquer dúvida é só usar `?sapply`.

```
sapply(df, class)
```

	TipodeRede	TipodeMonitoramento	Tipo
##	"factor"	"factor"	"factor"
##	Data	Hora	CodigoEstação
##	"factor"	"factor"	"integer"
##	NomeEstação	NomeParâmetro	UnidadedeMedida
##	"factor"	"factor"	"factor"
##	MediaHoraria	MediaMovel	Valido
##	"integer"	"factor"	"factor"

Quando trabalhamos com séries temporais, é importante ter a variável tempo reconhecida como “tempo”, especificamente como classe “`POSIXct`”. Porém, a classe do tipo Data é “`factor`” assim como a Hora, o que pode ser ruim. Então, vamos criar uma variável de tempo mais padronizada com o formato `2018-06-06 18:45:30`.

Para isso temos que juntar as variáveis Data e Hora. Faremos isso numa nova variável chamada “`tempo_char`”, adicionando-a diretamente no data frame com o cifrão (“`$`”). Podemos fazer isso com as funções `paste` ou `paste0`.

```
df$tempo_char <- paste(df$data, df$hora)
head(df$tempo_char)
```

```
## [1] "01/01/2014 01:00" "01/01/2014 02:00" "01/01/2014 03:00"
## [4] "01/01/2014 04:00" "01/01/2014 05:00" "01/01/2014 06:00"
class(df$tempo_char)
```

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

Melhorou, mas ainda tem classe character.

Para converter a nossa classe para `POSIXct` podemos usar a função `as.POSIXct` (olhe `?as.POSIXct`). Seus argumentos são:

```
args(as.POSIXct)
```

```
## function (x, tz = "", ...)
## NULL
```

Então, vamos criar outra variável tempo com o formato `POSIXct`:

```
df$tempo <- as.POSIXct(x = df$tempo_char, tz = "America/Sao_Paulo",
                        format = "%d/%m/%Y %H:%M")
head(df$tempo)

## [1] "2014-01-01 01:00:00 -02" "2014-01-01 02:00:00 -02"
## [3] "2014-01-01 03:00:00 -02" "2014-01-01 04:00:00 -02"
## [5] "2014-01-01 05:00:00 -02" "2014-01-01 06:00:00 -02"
class(df$tempo)

## [1] "POSIXct" "POSIXt"
```

Agora, vamos extrair os dias da semana do tempo, mes e dia juliano:

```
df$weekdays <- format(df$tempo, "%A")
head(df$weekdays)

## [1] "quarta" "quarta" "quarta" "quarta" "quarta" "quarta"

df$mes <- format(df$tempo, "%B")
head(df$mes)

## [1] "janeiro" "janeiro" "janeiro" "janeiro" "janeiro" "janeiro"
df$diajuliano <- julian(df$tempo)
head(df$diajuliano)

## Time differences in days
## [1] 16071.12 16071.17 16071.21 16071.25 16071.29 16071.33
df$ano <- format(df$tempo, "%Y")
```

Pronto! Agora temos o tempo no formato que desejamos.

4.1.5.1 aggregate

Vamos dar uma olhada no nosso data frame:

```
head(df)

##   TipodeRede TipodeMonitoramento      Tipo      Data    Hora
## 2 Automático          CETESB Dados Primários 01/01/2014 01:00
## 3 Automático          CETESB Dados Primários 01/01/2014 02:00
## 4 Automático          CETESB Dados Primários 01/01/2014 03:00
## 5 Automático          CETESB Dados Primários 01/01/2014 04:00
## 6 Automático          CETESB Dados Primários 01/01/2014 05:00
## 7 Automático          CETESB Dados Primários 01/01/2014 06:00
##   CódigoEstação        NomeEstação        NomeParâmetro
## 2                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 3                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 4                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 5                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 6                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 7                 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
##   UnidadedeMedida MediaHoraria MediaMovel Valido      tempo_char
## 2             ppb         9           - Não 01/01/2014 01:00
## 3             ppb         9           - Sim 01/01/2014 02:00
## 4             ppb         5           - Sim 01/01/2014 03:00
```

```

## 5          ppb        4      -    Sim 01/01/2014 04:00
## 6          ppb        5      -    Sim 01/01/2014 05:00
## 7          ppb        5      -    Sim 01/01/2014 06:00
##           tempo weekdays     mes   diajuliano ano
## 2 2014-01-01 01:00:00    quarta janeiro 16071.12 days 2014
## 3 2014-01-01 02:00:00    quarta janeiro 16071.17 days 2014
## 4 2014-01-01 03:00:00    quarta janeiro 16071.21 days 2014
## 5 2014-01-01 04:00:00    quarta janeiro 16071.25 days 2014
## 6 2014-01-01 05:00:00    quarta janeiro 16071.29 days 2014
## 7 2014-01-01 06:00:00    quarta janeiro 16071.33 days 2014

```

Com a função `aggregate`, podemos agregar os dados de diversas formas. Por exemplo, a média horaria por dia da semana é:

```

dff <- aggregate(df$MediaHoraria, by = list(df$weekdays), mean, na.rm = T)
names(dff) <- c("dias", "MediaHoraria")
dff

```

```

##      dias MediaHoraria
## 1 domingo    17.30973
## 2 quarta     34.07973
## 3 quinta     33.65931
## 4 sábado      27.31250
## 5 segunda     28.93543
## 6 sexta       35.40599
## 7 terça       32.04057

```

E o desvio-padrão é:

```

dff$sd <- aggregate(df$MediaHoraria, by = list(df$weekdays), sd, na.rm = T)$x
dff

##      dias MediaHoraria      sd
## 1 domingo    17.30973 22.11229
## 2 quarta     34.07973 41.26901
## 3 quinta     33.65931 42.14773
## 4 sábado      27.31250 35.06755
## 5 segunda     28.93543 32.15303
## 6 sexta       35.40599 44.44488
## 7 terça       32.04057 33.87323

```

Exercício: Em média, como o NOx varia ao longo do dia de acordo com esses dados? Qual a amplitude diária?

4.1.5.2 subset

Como podemos escolher só o mês de janeiro??

```

#[ LINHAS , COLUNAS ]
head(df[df$mes == "janeiro", ]) #TODAS AS COLUNAS

##      TipodeRede TipodeMonitoramento      Tipo      Data Hora
## 2 Automático          CETESB Dados Primários 01/01/2014 01:00
## 3 Automático          CETESB Dados Primários 01/01/2014 02:00

```

```

## 4 Automático CETESB Dados Primários 01/01/2014 03:00
## 5 Automático CETESB Dados Primários 01/01/2014 04:00
## 6 Automático CETESB Dados Primários 01/01/2014 05:00
## 7 Automático CETESB Dados Primários 01/01/2014 06:00
## CódigoEstação NomeEstação NomeParâmetro
## 2 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 3 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 4 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 5 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 6 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## 7 95 Cid.Universitária-USP-Ipen NOx (Óxidos de Nitrogênio)
## UnidadedeMedida MediaHoraria MediaMovel Valido tempo_char
## 2 ppb 9 - Não 01/01/2014 01:00
## 3 ppb 9 - Sim 01/01/2014 02:00
## 4 ppb 5 - Sim 01/01/2014 03:00
## 5 ppb 4 - Sim 01/01/2014 04:00
## 6 ppb 5 - Sim 01/01/2014 05:00
## 7 ppb 5 - Sim 01/01/2014 06:00
## tempo weekdays mes diajuliano ano
## 2 2014-01-01 01:00:00 quarta janeiro 16071.12 days 2014
## 3 2014-01-01 02:00:00 quarta janeiro 16071.17 days 2014
## 4 2014-01-01 03:00:00 quarta janeiro 16071.21 days 2014
## 5 2014-01-01 04:00:00 quarta janeiro 16071.25 days 2014
## 6 2014-01-01 05:00:00 quarta janeiro 16071.29 days 2014
## 7 2014-01-01 06:00:00 quarta janeiro 16071.33 days 2014

```

Só que agora temos só a média horária para esse mês, que retorna um vetor numérico:

```
names(df)
```

```

## [1] "TipodeRede"           "TipodeMonitoramento" "Tipo"
## [4] "Data"                 "Hora"                  "CódigoEstação"
## [7] "NomeEstação"          "NomeParâmetro"       "UnidadedeMedida"
## [10] "MediaHoraria"         "MediaMovel"          "Valido"
## [13] "tempo_char"           "tempo"                "weekdays"
## [16] "mes"                  "diajuliano"          "ano"

```

```
head(df[df$mes == "janeiro", 10])
```

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

```
head(df[df$mes == "janeiro", "MediaHoraria"])
```

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

```
class(df[df$mes == "janeiro", "MediaHoraria"])
```

```
## [1] "integer"
```

Exercício: Em média, como o NOx varia ao longo do dia em Julho? E em Dezembro?

Vamos salvar nosso data frame para depois fazer gráficos:

```
saveRDS(df, "dados/df.rds")
```

4.1.6 Data.Table e mais

O data.table é um pacote que apresenta a classe `data.table`, que é como uma versão melhorada da classe `data.frame`. O termo específico é que `data.table` tem um “parentesco” (`inherits`) com a classe `data.frame`.

Vamos ver como funciona `data.table` lendo os dois arquivos e comparar quanto tempo demora cada um. A função de leitura do `data.table` é `fread`.

```
df1 <- print(system.time(read.table("dados/NOXIPEN2014.txt", h = T)))

##    user  system elapsed
##  0.076   0.000   0.075

library(data.table)
df2 <- print(system.time(fread("dados/NOXIPEN2014.txt", h = T)))

##    user  system elapsed
##  0.008   0.000   0.008
```

4.2 Tidyverse

Um método mais recente (e muito interessante!) de tratar data frames é usando os pacotes dentro do Tidyverse.

Usando diversas funções dos pacotes `readr`, `tidyverse` e `dplyr`, por exemplo, é possível ler e processar dados de uma maneira mais *user-friendly* devido à sintaxe das funções e de como elas podem ser usadas em conjunto. Assim como o `data.table` tem uma classe própria, as funções dentro do tidyverse costumam trabalhar com sua própria classe, que é o `tibble` (ou `tbl_df`) - *que segundo os desenvolvedores são data.frames preguiçosos e mal-humorados. Isso pode parecer ruim, mas na verdade é bom pois exige funções “mais espertas” para lidar com eles.* Ainda assim é possível trabalhar com data frames sem dificuldades.

Note que muitas das funções usadas abaixo podem ser encontradas em Base ou em outros pacotes associados.

4.2.1 Importando dados

Todas as funções de leitura possuem a mesma estrutura:

```
read_*(arquivo,
      col_names = TRUE,
      col_types = NULL,
      locale = default_locale(),
      na = c("", "NA"),
      quoted_na = TRUE,
      comment = "",
      trim_ws = TRUE,
      skip = 0,
      n_max = Inf,
      guess_max = min(1000, n_max), progress = interactive())
```

Dica/Lembrete: Quando a descrição de uma função mostra os argumentos já definidos (`col_names = TRUE`), isso significa que esses são os valores padrão e que você não precisa escrever os argumentos se não quiser mudá-los

Assim, todas as funções de leitura só precisam do nome do arquivo!

Então como ler diferentes arquivos? Usando diferentes funções.

- `read_csv` lê arquivos .csv **separados por ,**
- `read_csv2` lê arquivos .csv **separados por ;**
- `read_delim` lê arquivos **com outros separadores** (definidos com o argumento `delim`)
- `read_fwf` lê arquivos **com delimitação fixa** (definidos com o argumento `col_positions`)
- `read_xl` lê arquivos **Excel** (.xls e .xlsx)

Assim, para ler os dados de Paracatu do INMET:

```
library(tidyverse)
paracatu <- read_csv("dados/paracatu.csv")
summary(paracatu)
```

```
##   codigo_estacao      data         hora       temp_inst
##   Length:24          Length:24     Length:24    Min.   :14.50
##   Class  :character  Class  :character  Class  :character  1st Qu.:16.25
##   Mode   :character  Mode   :character  Mode   :character  Median  :19.40
##                                         Mean   :20.04
##                                         3rd Qu.:24.32
##                                         Max.   :26.40
##   temp_max      temp_min      umid_inst      umid_max
##   Min.   :15.50  Min.   :14.20  Length:24      Length:24
##   1st Qu.:17.15 1st Qu.:15.38  Class  :character  Class  :character
##   Median  :19.75 Median  :18.15  Mode   :character  Mode   :character
##   Mean    :20.78  Mean    :19.26
##   3rd Qu.:24.98 3rd Qu.:23.38
##   Max.    :27.20  Max.    :26.20
##   umid_min      pto_orvalho_inst  pto_orvalho_max
##   Length:24      Length:24      Length:24
##   Class  :character  Class  :character  Class  :character
##   Mode   :character  Mode   :character  Mode   :character
##                                         Mean   :934.0
##                                         1st Qu.:934.7
##                                         Median :935.1
##                                         Mean   :935.3
##                                         3rd Qu.:935.6
##                                         Max.   :937.6
##   pto_orvalho_min      pressao      pressao_max      pressao_min
##   Length:24          Min.   :934.0  Min.   :934.1  Min.   :933.9
##   Class  :character  1st Qu.:934.7  1st Qu.:934.8  1st Qu.:934.4
##   Mode   :character  Median :935.1  Median :935.4  Median :934.9
##                                         Mean   :935.3
##                                         3rd Qu.:935.6
##                                         Max.   :937.7
##   vento_direcao      vento_vel      vento_rajada      radiacao
##   Min.   :0.300      Min.   :14.00     Min.   :0.80    Min.   :-3.60
##   1st Qu.:0.600      1st Qu.:77.75     1st Qu.:1.85   1st Qu.: -3.60
##   Median :1.350      Median :209.00    Median :2.80    Median : 21.03
##   Mean   :1.567      Mean   :177.96    Mean   :3.30    Mean   : 713.77
##   3rd Qu.:2.450      3rd Qu.:256.00    3rd Qu.:4.60   3rd Qu.:1483.50
##   Max.   :4.100      Max.   :355.00    Max.   :6.70    Max.   :2642.00
##   precipitacao
##   Min.   :0
##   1st Qu.:0
##   Median :0
```

```
## Mean    :0
## 3rd Qu.:0
## Max.   :0
```

4.2.2 Leitura %>% Processamento

Existem funções de leitura e modificação de data frames. Muitas vezes, você precisa lidar com dados “brutos” e que precisam de um certo processamento antes de serem utilizados em cálculos e gráficos. Como vimos anteriormente, isso exige no mínimo duas funções em duas linhas de código (uma para ler e outra para modificar), mas em geral esse processo precisa de bem mais do que isso.

O operador `%>%` (chamada de *pipe*) está dentro do pacote magnittr (dentro do Tidyverse) e é muito útil nesse processo! (Leia um pouco sobre ele aqui)

Como ele funciona?

variável `%>%` função1(., faz a modificação 1) `%>%` função2(., faz a modificação 2) `%>%` ... `%>%` função_n(., faz a modificação n)

O ponto (.) acima indica que a função será aplicada na versão da variável que chega nela.

Em notação matemática, podemos dizer que $x \xrightarrow{\% > \%} f(y) = f(x, y)$.

Por exemplo, observe o código abaixo:

```
library("tidyverse")

seq(1, 10) %>% order(., decreasing = T) %>% paste(., "n")

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

O vetor (que não precisa necessariamente ser definido como uma variável) é primeiro ordenado de forma decrescente e, a partir dessa modificação, é transformado em um vetor de caracteres ao colar a string “n” a ele.

Uma forma de fazer isso sem usar `%>%` seria:

```
paste(order(seq(1, 10), decreasing = T), "vezes")

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

Pergunta: Na sua opinião, qual é o código mais fácil e rápido de ser entendido?

Como isso pode ser aplicado a data frames?

Voltando aos dados de Paracatu, vamos supor que você ainda não tenha lido esses dados, mas já saiba que o dia e a hora estão em colunas separadas e que você precisa juntá-los para calcular médias em períodos específicos. Além disso, você quer comparar a temperatura instantânea “temp_inst” com a média entre temperatura máxima “temp_max” e mínima “temp_min”. Então, precisamos:

1. Ler os dados
2. Juntas as colunas “data” e “hora”
3. Transformar a coluna resultante em `POSIXct`
4. Calcular a média entre “temp_max” e “temp_min”
5. Comparar o que foi calculado com “temp_inst”

Sem `%>%`:

```

library(tidyverse)
paracatu <- read_csv("dados/paracatu.csv") #-- 1.
paracatu <- mutate(paracatu, data_completa = paste(data, hora, sep = "-")) #-- 2.
paracatu <- mutate(paracatu, data_completa = as.POSIXct(data_completa, format = "%d/%m/%Y-%H")) #-- 3.
paracatu <- mutate(paracatu, temp_med = (temp_max + temp_min)/2) #-- 4.
paracatu <- mutate(paracatu, temp_residuo = temp_inst - temp_med) #-- 5.
summary(paracatu)

##   codigo_estacao      data        hora      temp_inst
##   Length:24      Length:24      Length:24      Min.   :14.50
##   Class :character  Class :character  Class :character  1st Qu.:16.25
##   Mode  :character  Mode  :character  Mode  :character  Median  :19.40
##                               Mean   :20.04
##                               3rd Qu.:24.32
##                               Max.   :26.40
##   temp_max      temp_min      umid_inst      umid_max
##   Min.   :15.50  Min.   :14.20  Length:24      Length:24
##   1st Qu.:17.15 1st Qu.:15.38  Class :character  Class :character
##   Median  :19.75 Median  :18.15  Mode   :character  Mode   :character
##   Mean    :20.78  Mean    :19.26
##   3rd Qu.:24.98 3rd Qu.:23.38
##   Max.   :27.20  Max.   :26.20
##   umid_min      pto_orvalho_inst  pto_orvalho_max
##   Length:24      Length:24      Length:24
##   Class :character  Class :character  Class :character
##   Mode  :character  Mode  :character  Mode  :character
##
##   pto_orvalho_min      pressao      pressao_max      pressao_min
##   Length:24      Min.   :934.0  Min.   :934.1  Min.   :933.9
##   Class :character  1st Qu.:934.7  1st Qu.:934.8  1st Qu.:934.4
##   Mode  :character  Median  :935.1  Median  :935.4  Median  :934.9
##                               Mean   :935.3  Mean   :935.6  Mean   :935.1
##                               3rd Qu.:935.6 3rd Qu.:936.0  3rd Qu.:935.4
##                               Max.   :937.6  Max.   :937.7  Max.   :937.4
##   vento_direcao      vento_vel      vento_rajada      radiacao
##   Min.   :0.300  Min.   :14.00  Min.   :0.80  Min.   :-3.60
##   1st Qu.:0.600  1st Qu.:77.75  1st Qu.:1.85  1st Qu.:-3.60
##   Median  :1.350  Median  :209.00  Median  :2.80  Median  :21.03
##   Mean    :1.567  Mean    :177.96  Mean    :3.30  Mean    :713.77
##   3rd Qu.:2.450  3rd Qu.:256.00  3rd Qu.:4.60  3rd Qu.:1483.50
##   Max.   :4.100  Max.   :355.00  Max.   :6.70  Max.   :2642.00
##   precipitacao data_completa      temp_med
##   Min.   :0      Min.   :2018-06-01 00:00:00  Min.   :14.85
##   1st Qu.:0      1st Qu.:2018-06-01 05:45:00  1st Qu.:16.21
##   Median  :0      Median :2018-06-01 11:30:00  Median  :18.90
##   Mean    :0      Mean   :2018-06-01 11:30:00  Mean   :20.02
##   3rd Qu.:0      3rd Qu.:2018-06-01 17:15:00  3rd Qu.:24.18
##   Max.   :0      Max.   :2018-06-01 23:00:00  Max.   :26.70
##   temp_residuo
##   Min.   :-1.60000
##   1st Qu.:-0.35000
##   Median  : 0.02500

```

```
##  Mean   : 0.02292
##  3rd Qu.: 0.55000
##  Max.   : 1.65000
```

Note que sempre se mostra necessário salvar cada passo em uma nova variável ou, nesse caso, reciclar a mesma variável. Foi preciso uma linha para cada operação.

Já com %>%:

```
library(tidyverse)
paracatu <- read_csv("dados/paracatu.csv") %>% #-- 1.
  mutate(., data_completa = paste(data, hora, sep = "-")) %>% #-- 2.
  mutate(., data_completa = as.POSIXct(data_completa, format = "%d/%m/%Y-%H")) %>% #-- 3.
  mutate(., temp_med = (temp_max + temp_min)/2) %>% #-- 4.
  mutate(., temp_residuo = temp_inst - temp_med) #-- 5.
summary(paracatu)

##   codigo_estacao      data          hora      temp_inst
##   Length:24      Length:24      Length:24      Min.   :14.50
##   Class :character  Class :character  Class :character  1st Qu.:16.25
##   Mode  :character  Mode  :character  Mode  :character  Median  :19.40
##                               Mean   :20.04
##                               3rd Qu.:24.32
##                               Max.   :26.40
##   temp_max      temp_min      umid_inst      umid_max
##   Min.   :15.50  Min.   :14.20  Length:24      Length:24
##   1st Qu.:17.15 1st Qu.:15.38  Class :character  Class :character
##   Median  :19.75 Median  :18.15  Mode   :character  Mode   :character
##   Mean    :20.78  Mean    :19.26
##   3rd Qu.:24.98 3rd Qu.:23.38
##   Max.    :27.20  Max.    :26.20
##   umid_min      pto_orvalho_inst      pto_orvalho_max
##   Length:24      Length:24      Length:24
##   Class :character  Class :character  Class :character
##   Mode  :character  Mode  :character  Mode  :character
##                               Length:24
##   pto_orvalho_min      pressao      pressao_max      pressao_min
##   Length:24      Min.   :934.0  Min.   :934.1  Min.   :933.9
##   Class :character  1st Qu.:934.7  1st Qu.:934.8  1st Qu.:934.4
##   Mode  :character  Median :935.1  Median :935.4  Median :934.9
##                               Mean   :935.3  Mean   :935.6  Mean   :935.1
##                               3rd Qu.:935.6 3rd Qu.:936.0  3rd Qu.:935.4
##                               Max.   :937.6  Max.   :937.7  Max.   :937.4
##   vento_direcao      vento_vel      vento_rajada      radiacao
##   Min.   :0.300  Min.   :14.00  Min.   :0.80  Min.   :-3.60
##   1st Qu.:0.600  1st Qu.:77.75  1st Qu.:1.85  1st Qu.: -3.60
##   Median :1.350  Median :209.00  Median :2.80  Median : 21.03
##   Mean   :1.567  Mean   :177.96  Mean   :3.30  Mean   : 713.77
##   3rd Qu.:2.450  3rd Qu.:256.00  3rd Qu.:4.60  3rd Qu.:1483.50
##   Max.   :4.100  Max.   :355.00  Max.   :6.70  Max.   :2642.00
##   precipitacao      data_completa      temp_med
##   Min.   :0      Min.   :2018-06-01 00:00:00  Min.   :14.85
##   1st Qu.:0      1st Qu.:2018-06-01 05:45:00  1st Qu.:16.21
##   Median :0      Median :2018-06-01 11:30:00  Median :18.90
```

```

##  Mean    :0      Mean    :2018-06-01 11:30:00  Mean    :20.02
##  3rd Qu.:0      3rd Qu.:2018-06-01 17:15:00  3rd Qu.:24.18
##  Max.    :0      Max.    :2018-06-01 23:00:00  Max.    :26.70
##  temp_residuo
##  Min.    :-1.60000
##  1st Qu.:-0.35000
##  Median  : 0.02500
##  Mean    : 0.02292
##  3rd Qu.: 0.55000
##  Max.    : 1.65000

```

O código pode estar separado por linhas, mas perceba que, ao rodar qualquer linha desse conjunto, todo ele é rodado de uma vez!

Assim, adicionar um passo fica fácil. Por exemplo, as colunas “data” e “hora” não são mais necessárias:

```

library(tidyverse)
paracatu <- read_csv("dados/paracatu.csv") %>% #-- 1.
  mutate(., data_completa = paste(data, hora, sep = "-")) %>% #-- 2.
  mutate(., data_completa = as.POSIXct(data_completa, format = "%d/%m/%Y-%H")) %>% #-- 3.
  mutate(., temp_med = (temp_max + temp_min)/2) %>% #-- 4.
  mutate(., temp_residuo = temp_inst - temp_med) %>% #-- 5.
  select(., -c(data, hora)) #-- 6.

```

```

## Error in (function (classes, fdef, mtable) : unable to find an inherited method for function 'select'
summary(paracatu)

```

```

##  codigo_estacao      data          hora          temp_inst
##  Length:24           Length:24       Length:24       Min.   :14.50
##  Class :character    Class :character  Class :character  1st Qu.:16.25
##  Mode   :character    Mode   :character  Mode   :character  Median  :19.40
##                                         Mean   :20.04
##                                         3rd Qu.:24.32
##                                         Max.   :26.40
##  temp_max            temp_min        umid_inst      umid_max
##  Min.   :15.50        Min.   :14.20       Length:24       Length:24
##  1st Qu.:17.15        1st Qu.:15.38      Class :character  Class :character
##  Median  :19.75        Median :18.15      Mode   :character  Mode   :character
##  Mean    :20.78        Mean    :19.26
##  3rd Qu.:24.98        3rd Qu.:23.38
##  Max.   :27.20        Max.   :26.20
##  umid_min            pto_orvalho_inst  pto_orvalho_max
##  Length:24           Length:24       Length:24
##  Class :character    Class :character  Class :character
##  Mode   :character    Mode   :character  Mode   :character
##  pto_orvalho_min     pressao        pressao_max    pressao_min
##  Length:24           Min.   :934.0     Min.   :934.1     Min.   :933.9
##  Class :character    1st Qu.:934.7    1st Qu.:934.8    1st Qu.:934.4
##  Mode   :character    Median :935.1     Median :935.4     Median :934.9
##                                         Mean   :935.3     Mean   :935.6     Mean   :935.1
##                                         3rd Qu.:935.6    3rd Qu.:936.0    3rd Qu.:935.4
##                                         Max.   :937.6     Max.   :937.7     Max.   :937.4

```

```

##  vento_direcao      vento_vel      vento_rajada      radiacao
##  Min.   :0.300      Min.   :14.00      Min.   :0.80      Min.   :-3.60
##  1st Qu.:0.600      1st Qu.:77.75      1st Qu.:1.85      1st Qu.:-3.60
##  Median :1.350      Median :209.00      Median :2.80      Median : 21.03
##  Mean    :1.567      Mean   :177.96      Mean   :3.30      Mean   : 713.77
##  3rd Qu.:2.450      3rd Qu.:256.00      3rd Qu.:4.60      3rd Qu.:1483.50
##  Max.    :4.100      Max.   :355.00      Max.   :6.70      Max.   :2642.00
##  precipitacao data_completa          temp_med
##  Min.   :0           Min.   :2018-06-01 00:00:00  Min.   :14.85
##  1st Qu.:0           1st Qu.:2018-06-01 05:45:00  1st Qu.:16.21
##  Median :0           Median :2018-06-01 11:30:00  Median :18.90
##  Mean    :0           Mean   :2018-06-01 11:30:00  Mean   :20.02
##  3rd Qu.:0           3rd Qu.:2018-06-01 17:15:00  3rd Qu.:24.18
##  Max.    :0           Max.   :2018-06-01 23:00:00  Max.   :26.70
##  temp_residuo
##  Min.   :-1.60000
##  1st Qu.:-0.35000
##  Median : 0.02500
##  Mean   : 0.02292
##  3rd Qu.: 0.55000
##  Max.   : 1.65000

```

Pergunta: E agora? Na sua opinião, qual é o código mais fácil e rápido de ser entendido?

Exercício: Crie um data frame dos dados de Paracatu com a data no formato `POSIXct` e os resíduos de temperatura, umidade e pressão. Em geral, qual grandeza tem maior resíduo?

4.3 Outros Tipos de Dados

4.3.1 NetCDF

O NetCDF (Network Common Data Form) é um conjunto de bibliotecas de software e formatos de dados independentes de máquina e autodescritivos com suporte para criação, acesso e compartilhamento de dados científicos orientados a matrizes. Arquivos NetCDF (criado por essa biblioteca ou por programas que utilizam essa biblioteca) são arquivos compostos por dados, atributos e metadados.

O pacote `ncdf4` é um exemplo de interface do R com a biblioteca NetCDF 4, os comandos abaixo instalam e carregam esse pacote:

```

#install.packages("ncdf4") # instala o pacote
library("ncdf4")           # carrega o pacote
nc_version()                # versão da biblioteca

```

```
## [1] "ncdf4_1.16_20170401"
```

Um exemplo de NetCDF:

```
edgar <- nc_open(filename = "dados/v431_v2_REFERENCE_CO_2010_10_AGR.0.1x0.1.nc")
```

O objeto `edgar` contém algumas informações sobre o conteúdo do arquivo, com um `print(edgar)` ou simplesmente `edgar` visualizamos o conteúdo do arquivo:

```
class(edgar)
edgar
```

Isso mostra o nome do arquivo (e versão da biblioteca usada para criar), número de variáveis (1 variável neste exemplo), uma descrição de cada variável (incluindo atributos) as dimensões (2 para esse arquivo) e atributos globais.

Agora vamos abrir alguma variável:

```
names(edgar$var)
co <- ncdf4::ncvar_get(edgar, "emi_co")
class(co)
```

Como o NetCDF é organizado para guardar matrizes (arrays), só sabemos que a variável `emi_co` é um array

```
ncatt_get(edgar, "emi_co") # ou ncatt_get(edgar, "emi_co", verbose = T)
```

repete a informação do print anterior, mas sem os atributos globais:

```
# $standard_name
# [1] "tendency_of_atmosphere_mass_content_of_carbon_monoxide_due_to_emission"
# $long_name
# [1] "Emissions of CO - "
# $units
# [1] "kg m-2 s-1"
# $cell_method
# [1] "time: mean (interval: 1 month, 31 days)"
# $total_emi_co
# [1] " 6.32889e+009 kg/month"
# $comment
# [1] " (see http://edgar.jrc.ec.europa.eu/methodology.php#12sou for the definitions of the single sourc...
```

A latitude de cada ponto de grade, assim como longitude níveis (tempo em outros casos) o podem ser extraídas:

```
lat <- ncvar_get(wrf, "lat")
lon <- ncvar_get(wrf, "lon")
```

O metadado de Longitude também pode conter informações úteis:

```
ncatt_get(edgar, "lon")
# $standard_name
# [1] "longitude"
# $long_name
# [1] "longitude"
# $units
# [1] "degrees_east"
# $comment
# [1] "center_of_cell"
```

e Latitude:

```
ncatt_get(edgar, "lat")
# $standard_name
# [1] "latitude"
# $long_name
# [1] "latitude"
# $units
# [1] "degrees_north"
```

```
# $comment
# [1] "center_of_cell"
```

e também os atributos globais podem ser acessados separadamente:

```
ncatt_get(edgar, varid = 0)
# $Conventions
# [1] "CF-1.0"
# $title
# [1] "Monthly Mean (October) Emissions of CO - "
# $institution
# [1] "European Commission, Joint Research Centre"
# $source
# [1] "http://edgar.jrc.ec.europa.eu/"
# $history
# [1] "Created from original data (0.1x0.1 degrees) using IDL program (edgar_ascii_to_ncdf.pro) on Wed ...
# $references
# [1] "European Commission, Joint Research Centre (JRC)/Netherlands Environmental Assessment Agency (PBA...
# $copyright_notice
# [1] "Reproduction of the data is authorized, except for commercial purposes, provided the source is f...
# $contact
# [1] "edgar-info@jrc.ec.europa.eu"
```

Da mesma forma com que podemos acessar variáveis e atributos com `ncvar_get` e `ncatt_get`, podemos modificar estes valores com `ncvar_put` e `ncatt_put`. Outras operações como renomear (`ncvar_rename`) e trocar o valor de missval (`ncvar_change_missval`) também estão disponíveis.

DICA: `ncatt_get` e `ncatt_put` acessam e alteram os atributos de variáveis e também atributos globais do NetCDF usando o argumento `varid=0` como no código mostrado anteriormente.

Para salvar as alterações e/ou liberar o acesso ao arquivo use a função `nc_close` (ou a função `nc_sync` que sincroniza o NetCDF mas não fecha a conexão com o arquivo).

```
nc_close(edgar) # ou nc_sync(edgar)
```

O pacote possui ainda funções mais específicas para a criação de arquivos em NetCDF como `nc_create`, funções que definem dimensões como `ncdim_def` e funções para colocar e tirar o arquivo de modo de definição `nc_redef` e `nc_enddef`.

DICA: o NetCDF no R funciona de forma parecida com uma lista ou data frame, podemos “ver” ou selecionar suas sub-partes (sub-sub-partes...) com “\$” e TAB.

4.3.2 Dados Binários

O R lê e escreve dados binários usando as funções `readBin` e `writeBin`, respectivamente.

Ler dados binários no R

Em meteorologia, frequentemente os dados estão em formato binário. A maior “dificuldade” em ler estes dados está em conhecer como eles foram gerados. Neste curso, o arquivo binário que vamos abrir como exemplo contém dados de temperatura de brilho obtidas com o satélite GOES-13 (informações em: https://disc.gsfc.nasa.gov/datasets/GPM_MERGIR_1/summary).

Lembrem-se de baixar o dado em: <https://github.com/iagdevs/cursoR/tree/master/dados>

Repare que a função `readBin` requer alguns argumentos para ler estes dados da forma correta:

```
args(readBin)

## function (con, what, n = 1L, size = NA_integer_, signed = TRUE,
##         endian = .Platform$endian)
## NULL
# Lembre-se de usar ?readBin
```

O que são estes argumentos?

- `con` é um objeto de conexão ou um string de caracteres nomeando um arquivo.
- `what` é um string de caracteres que indica o modo dos dados a serem lidos. Pode ser um “numeric”, “double”, “integer”, “int”, “logical”, “complex”, or “character”.
- `n` um inteiro especificando o número máximo de valores a serem lidos.
- `size` um inteiro especificando o número de bytes por valor lido. O padrão de `NA_integer_` usa um tamanho padrão para um valor do modo (`what`) especificado (4 para “integer”, 8 para “double” e 16 para “complex”). Este argumento é ignorado para dados “character”. Neste caso, os dados de caracteres são lidos como sequências de caracteres de byte único terminadas em nulo.
- `signed` é um valor lógico. Usado apenas para números “integer” de tamanhos 1 e 2. Se `TRUE` (por default), “integers” são considerados como números inteiros assinados.
- `endian` é um string de caracteres. Se `endian = "big"` ou se `endian = "little"`, especifica-se a extremidade (endian-ness) dos valores lidos ou escritos. Se `endian="swap"` a extremidade é trocada. Hardware Intel é “little” endian e hardware Sun é “big” endian.

```
# Ler o arquivo binário: readBin
11 <- readBin("dados/gs.140422.1900g.ch4",
              what="int",
              n = 1349*1613,
              size = 2,
              signed = T,
              endian = "little")
class(11)
```

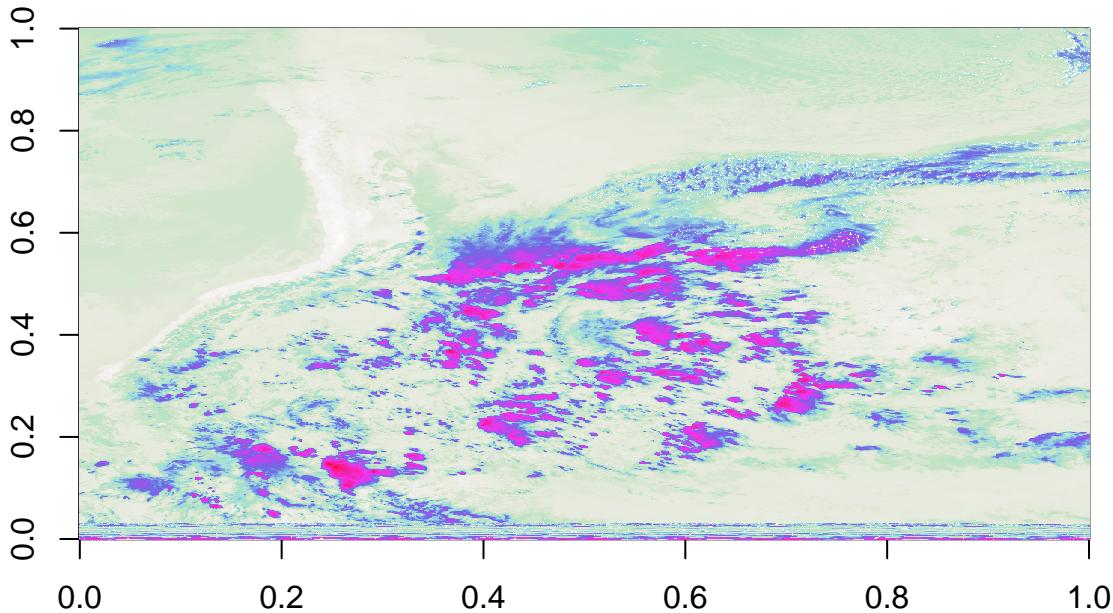
```
## [1] "integer"
```

Uma forma rápida para verificarmos os nossos dados é gráfica. Logo, que tal um plot?

```
12 <- matrix(11, ncol=1613, nrow=1349)
class(12)
```

```
## [1] "matrix"
# Vamos chamar o pacote cptcity para selecionar facilmente uma paleta de cores legal.
library(cptcity)
image(12,
      col = cpt(find_cpt("sat")[8]),
      main = "Temperatura de brilho")
```

Temperatura de brilho



Tem algo estranho com esta imagem. O que é? (valendo um sticker).

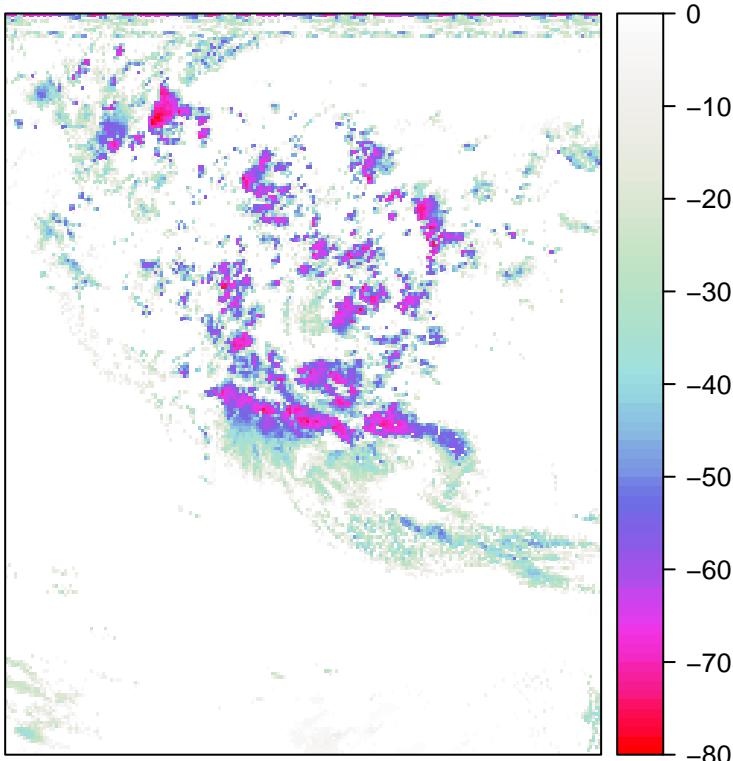
```
library(raster, quietly = TRUE)
l3 <- raster(t(12[1:1349,1:1613]),
             xmn=-82.00,
             ymn=-44.96,
             xmx=-82.0 + (0.03593245*1349),
             ymx=-44.96 + (0.03593245*1613),
             crs = CRS("+init=epsg:4326"))
class(l3)
```

```
## [1] "RasterLayer"
## attr(,"package")
## [1] "raster"
```

O capítulo geoespacial será visto no final deste curso. Porém, nesta etapa vamos usar o pacote `raster` somente para analisar se os dados binários foram lidos corretamente.

```
sp::spplot(((l3 + 75)/100)-273, # Estas correções são necessárias. Veja: http://www.cpc.ncep.noaa.gov/p
           col.regions = cpt(find_cpt("sat")[8]),
           at = seq(-80,0,1),
           main = "Temperatura de brilho (°C)")
```

Temperatura de brilho (ºC)



Escrever dados binários no R

```
# Escrever um arquivo binário: writeBin
args(writeBin)

## function (object, con, size = NA_integer_, endian = .Platform$endian,
##           useBytes = FALSE)
## NULL
# Lembre-se de usar ?writeBin
```

`useBytes` um valor lógico. No R, se FALSE (o default), especifica que `writeBin` converteria strings com codificações para uma string imprimível antes de escrever os bytes. Se TRUE, especifica que `writeBin` escreveria os bytes do string não convertidos.

`object` um objeto escrito no `con`.

```
# Na linha abaixo vamos criar um arquivo temporário com o auxílio da função "tempfile"
# para depois escrever os dados dentro deste arquivo.
```

```
tf <- tempfile()
# Vamos atribuir a "tb" os 20 primeiros valores de l1 (temperatura de brilho).
tb <- c(21000, 21000, 20800, 21000, 20800, 20700, 20600, 20600, 20900, 20900,
      20900, 20900, 20700, 20500, 20500, 20400, 20500, 20700, 20400, 20300)
class(tb)
```

```
## [1] "numeric"
```

```
x <- as.integer(tb)
writeBin(x, con = tf)
```

```
# Agora vamos ler o arquivo binário escrito acima. Você lembra quais são os argumentos para ler binário
```

```
readBin(tf,
       what = "integer",
       n = 20)

## [1] 21000 21000 20800 21000 20800 20700 20600 20600 20900 20900 20900
## [12] 20900 20700 20500 20500 20400 20500 20700 20400 20300
```


Chapter 5

Plotando

5.1 plot (base)

Exemplo: Dados de qualidade do ar

```
df <- readRDS("dados/df.rds")
summary(df)

##      TipodeRede      TipodeMonitoramento          Tipo
##  Automático:8581    CETESB:8581      Dados Primários:8581
##
##      Data            Hora        CódigoEstação
##  01/01/2014: 24 16:00 : 363   Min.   :95
##  01/01/2015: 24 12:00 : 361   1st Qu.:95
##  01/02/2014: 24 14:00 : 361   Median :95
##  01/04/2014: 24 18:00 : 361   Mean    :95
##  01/05/2014: 24 13:00 : 360   3rd Qu.:95
##  01/06/2014: 24 17:00 : 360   Max.    :95
##  (Other)   :8437 (Other) :6415
##                  NomeEstação           NomeParâmetro
##  Cid.Universitária-USP-Ipen:8581 NOx (Óxidos de Nitrogênio):8581
##
##      UnidadedeMedida  MediaHoraria  MediaMovel Valido      tempo_char
##  ppb:8581           Min.   : 0.00  -:8581    Não: 907  Length:8581
##                      1st Qu.: 9.00                    Sim:7674  Class :character
##                      Median :18.00                   Mode  :character
##                      Mean   :29.87
##                      3rd Qu.:34.00
##                      Max.   :306.00
```

```

##          NA's :260
## tempo            weekdays            mes
## Min.   :2014-01-01 01:00:00  Length:8581      Length:8581
## 1st Qu.:2014-04-05 14:00:00  Class :character  Class :character
## Median :2014-07-05 23:00:00  Mode  :character  Mode  :character
## Mean   :2014-07-04 20:03:16
## 3rd Qu.:2014-10-03 10:00:00
## Max.   :2015-01-02 00:00:00
##
## diajuliano        ano
## Length:8581      Length:8581
## Class :difftime   Class :character
## Mode  :numeric    Mode  :character
##
## 
## 
## 
## 
```

A função `plot` precisa dos seguintes argumentos:

```
args(plot)
```

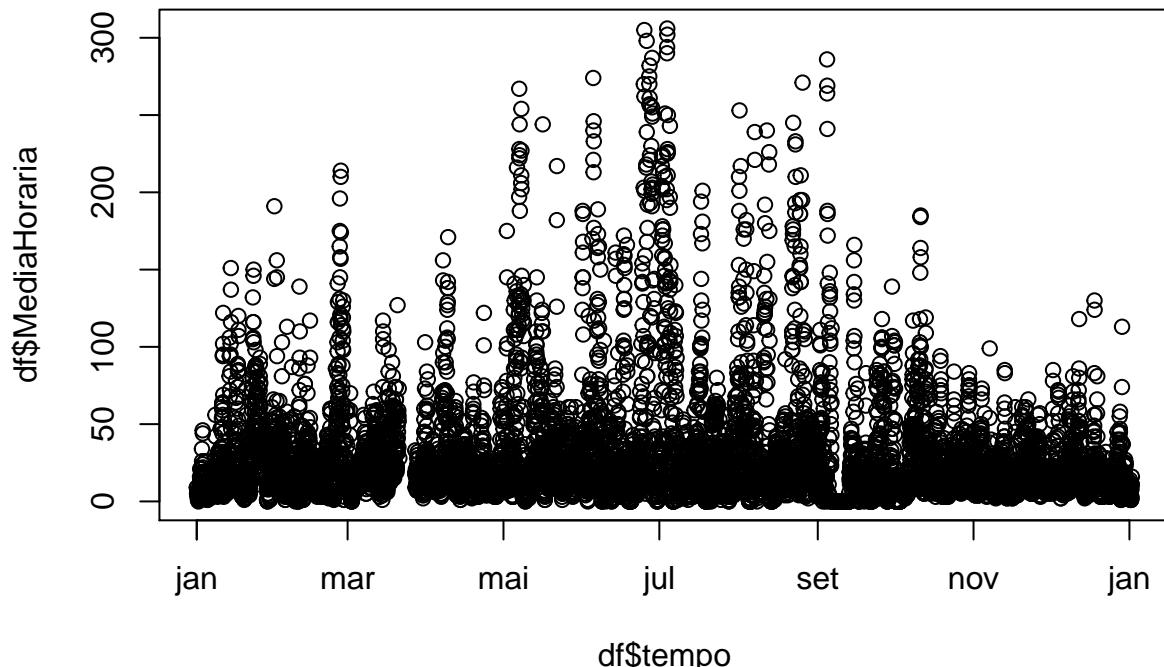
```

## function (x, y, ...)
## NULL

```

Então, a forma mais fácil de plotar uma variável em função do tempo é:

```
plot(x = df$tempo, y = df$MediaHoraria)
```



Feio, né?

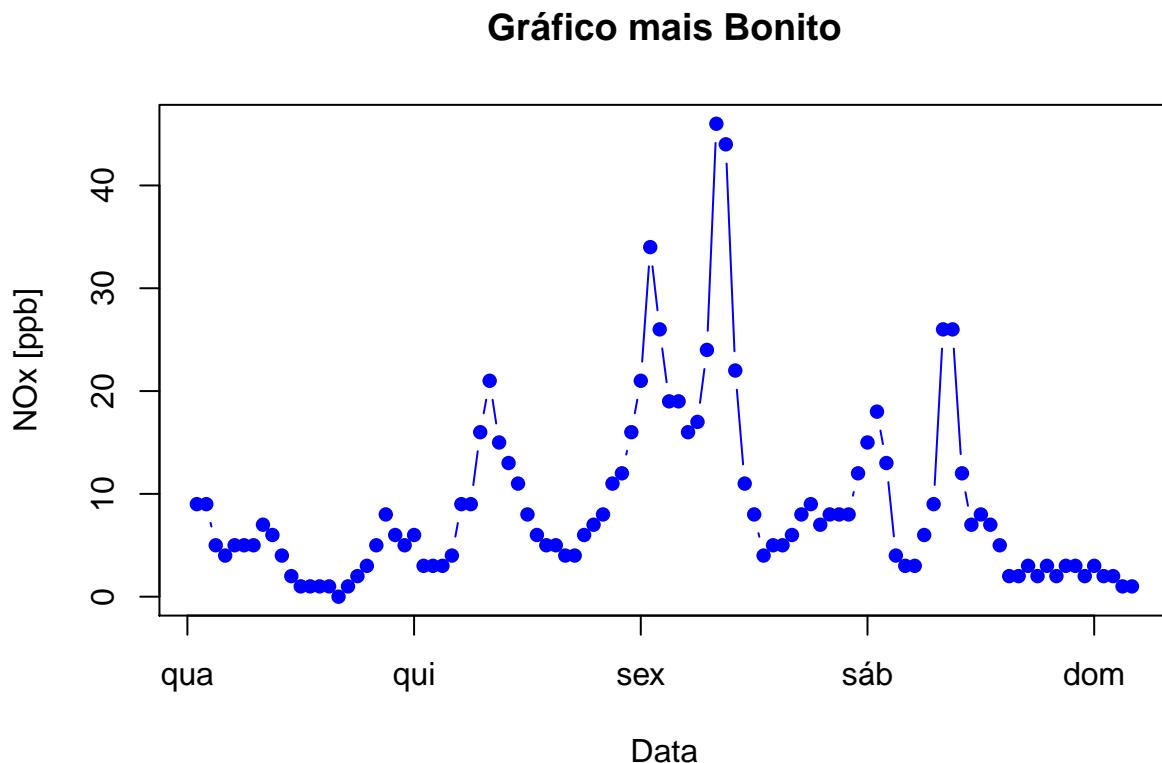
Tentando deixar mais bonito...

```

plot(x = df$tempo[1:100], y = df$MediaHoraria[1:100], #-- Selecionando uma parte do df!
      pch = 16, #-- Forma do ponto (círculo preenchido)
      type = "b", #-- Tipo de gráfico ("b" = both, ponto e linha)

```

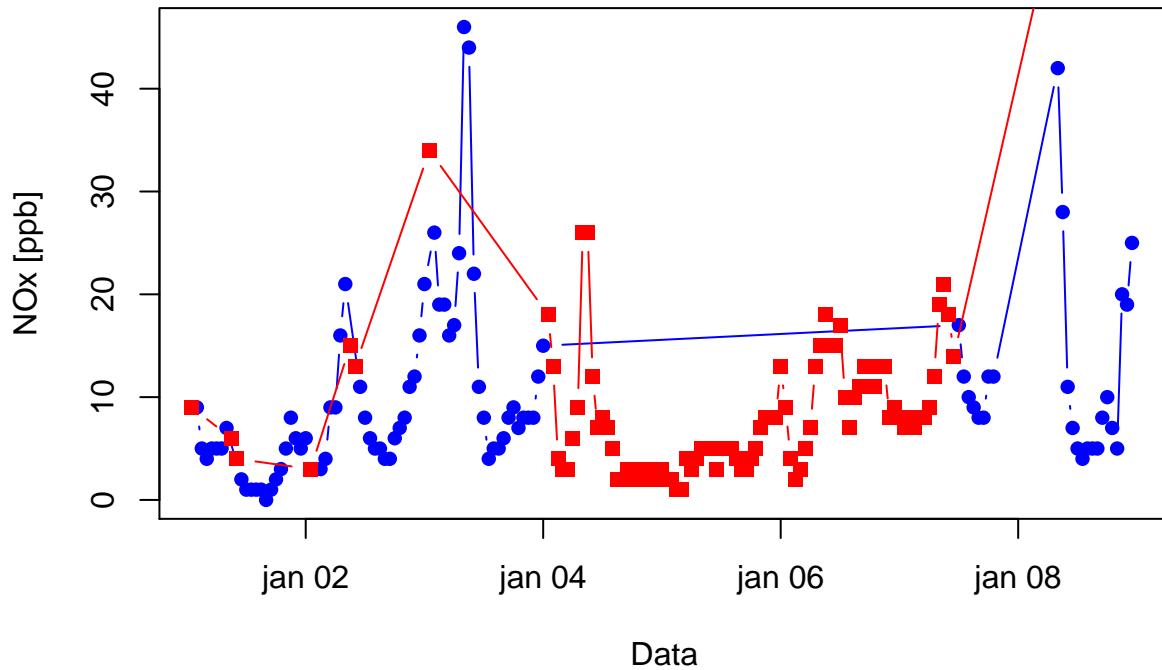
```
col = "blue", #-- Cor do elemento (definido pelo type)
xlab = "Data", ylab = "NOx [ppb]", #-- Nome dos eixos x e y
main = "Gráfico mais Bonito") #-- Título do gráfico
```



Colocando **DOIS** elementos no mesmo gráfico:

```
df_parcial <- df[1:180,] #-- Selecionando uma parte do df!
plot(x = df_parcial$tempo[df_parcial$Valido == "Sim"],
      y = df_parcial$MediaHoraria[df_parcial$Valido == "Sim"],
      pch = 16, type = "b", col = "blue",
      xlab = "Data", ylab = "NOx [ppb]",
      main = "Dados Válidos e Inválidos")
lines(x = df_parcial$tempo[df$Valido == "Não"],
      y = df_parcial$MediaHoraria[df$Valido == "Não"],
      pch = 15, type = "b", col = "red")
```

Dados Válidos e Inválidos



Desafio: Coloque uma legenda na figura especificando que os dados válidos estão em azul e os inválidos em vermelho

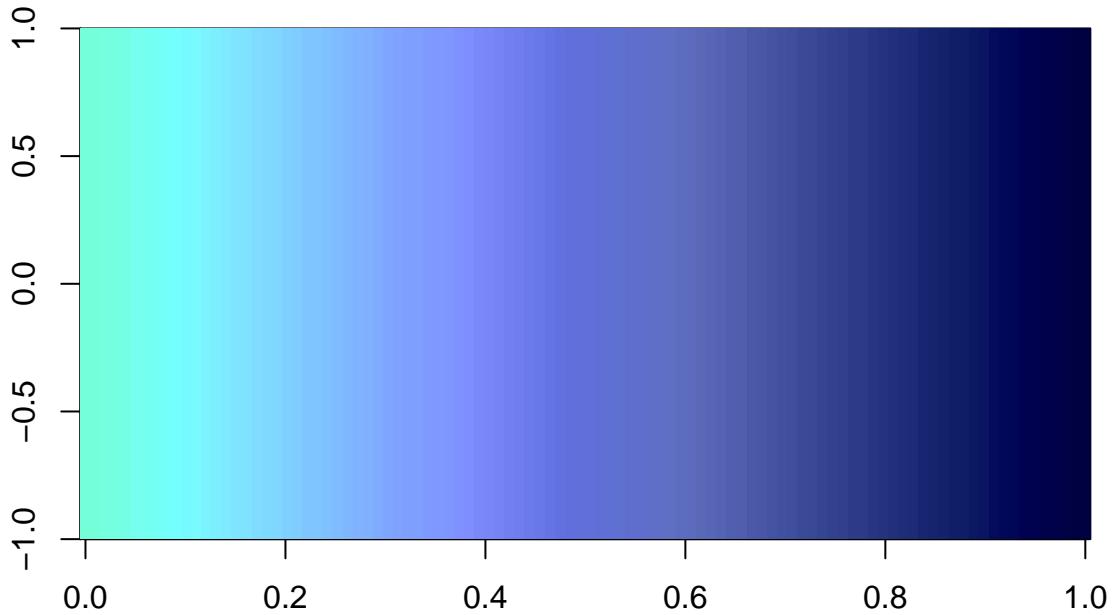
A função `plot` cumpre bem o papel de gerar um gráfico simples, e até permite algumas customizações, mas ela exige cada vez mais linhas de código e argumentos dentro das funções para deixar o gráfico “mais bonito” - ao cumprir o desafio, você irá perceber como uma coisa “simples” como colocar uma legenda pode exigir muito mais do que parece!

5.2 image

A função `image` permite pôrtear matriizes diretamente. Vamos testar esta função com a librería `cptcity`. Argumentos similares de `plot` podem ser usados em `image`.

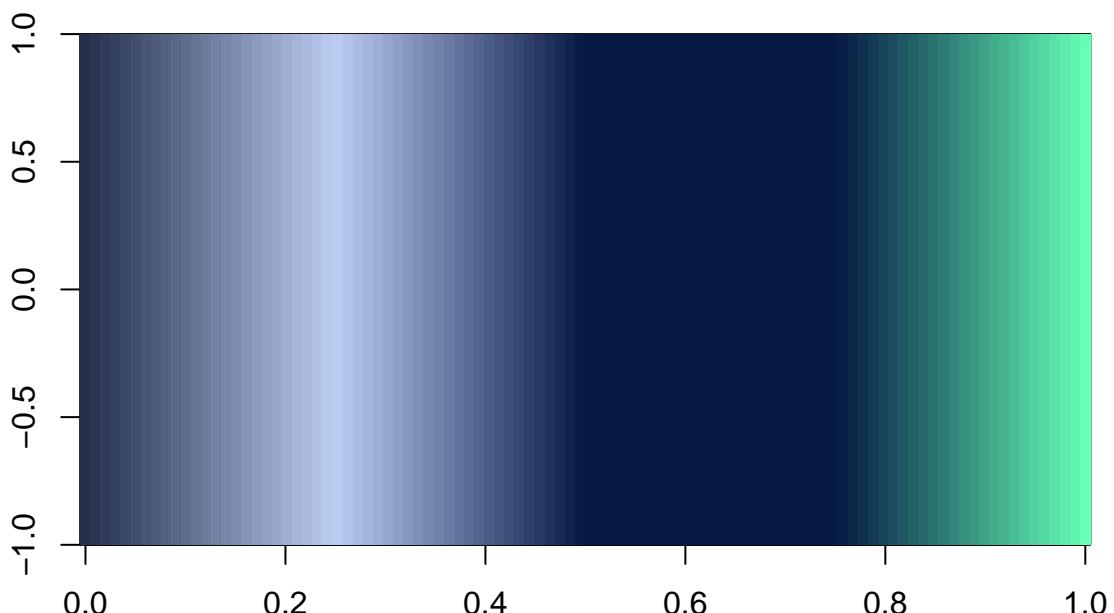
```
library(cptcity)
image(matrix(1:100), col = lucky())
```

```
## Colour gradient: jm_ad_ad_d, number: 4507
```



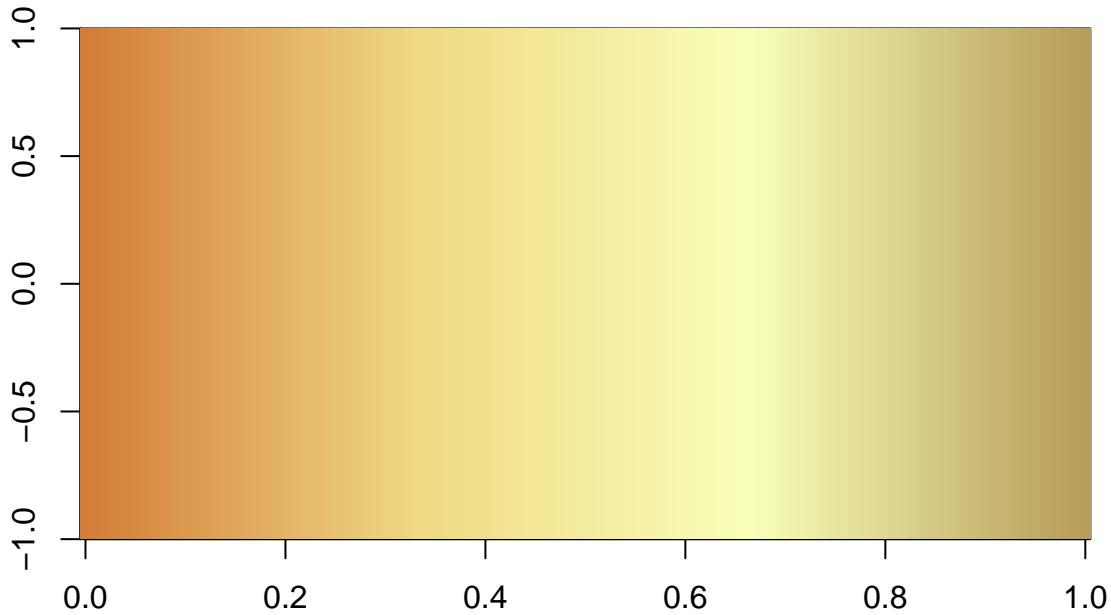
```
image(matrix(1:100), col = lucky())
```

Colour gradient: gacruxa_kon_kon_21, number: 2751



```
image(matrix(1:100), col = lucky())
```

Colour gradient: ma_spice_spice15, number: 4957



5.3 ggplot (ggplot2)

A função `ggplot` funciona de um jeito um pouco diferente. Veja a figura abaixo:

Em vez de uma única função, o gráfico é formado por camadas, sendo que cada camada é um elemento (`geom_...` ou `stat_...`) ou configuração (`scale_..._..., coord_..., theme` ou `theme_..., guides`, `labs`, etc). Consulte a maioria das opções disponíveis em Data Visualization Cheatsheet.

Que tal refazermos os gráficos da seção anterior?

```
## Não esqueça de carregar o pacote!
library(ggplot2)

ggplot(df, aes(x = tempo, y = MediaHoraria)) +
  geom_point(pch = 1)
```

```
## Warning: Removed 260 rows containing missing values (geom_point).
```

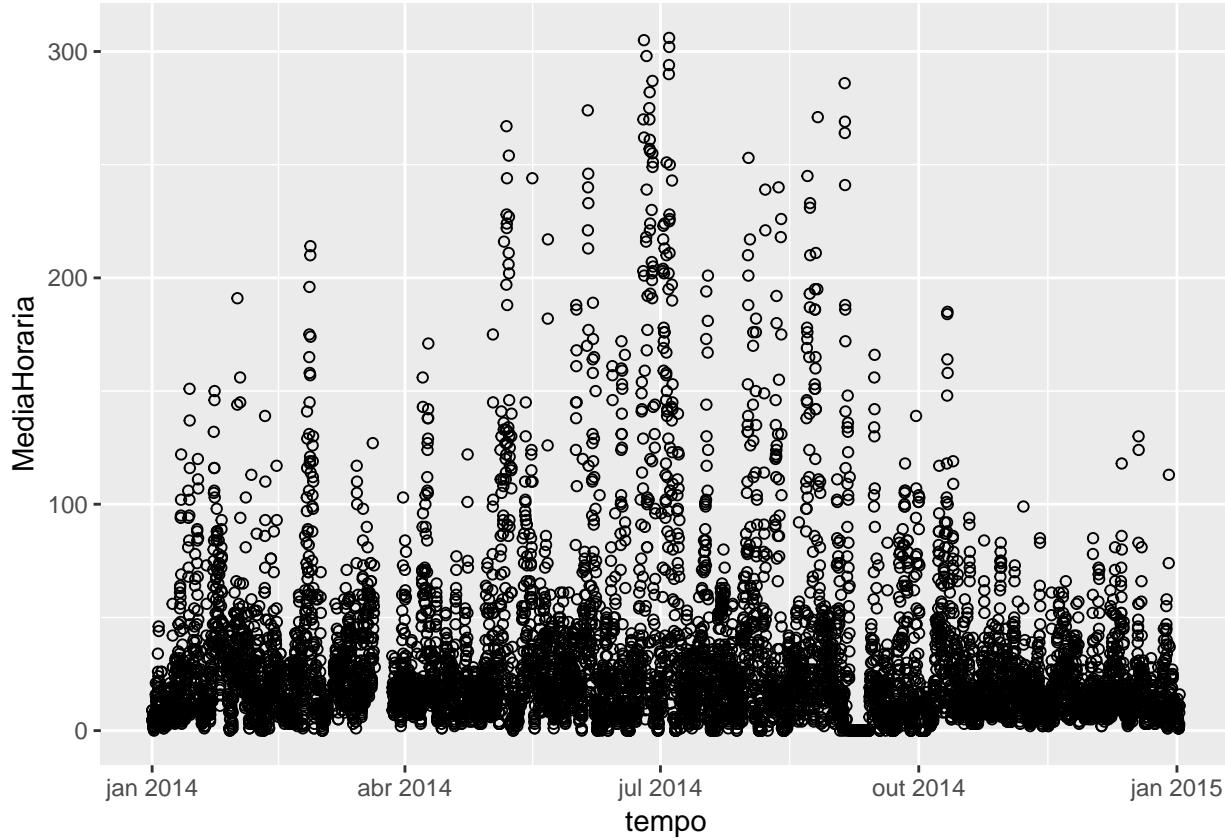
Complete the template below to build a graph.

```
ggplot (data = <DATA>) +  
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),  
    stat = <STAT>, position = <POSITION>) +  
    <COORDINATE_FUNCTION> +  
    <FACET_FUNCTION> +  
    <SCALE_FUNCTION> +  
    <THEME_FUNCTION>
```

] required
] Not required, sensible defaults supplied

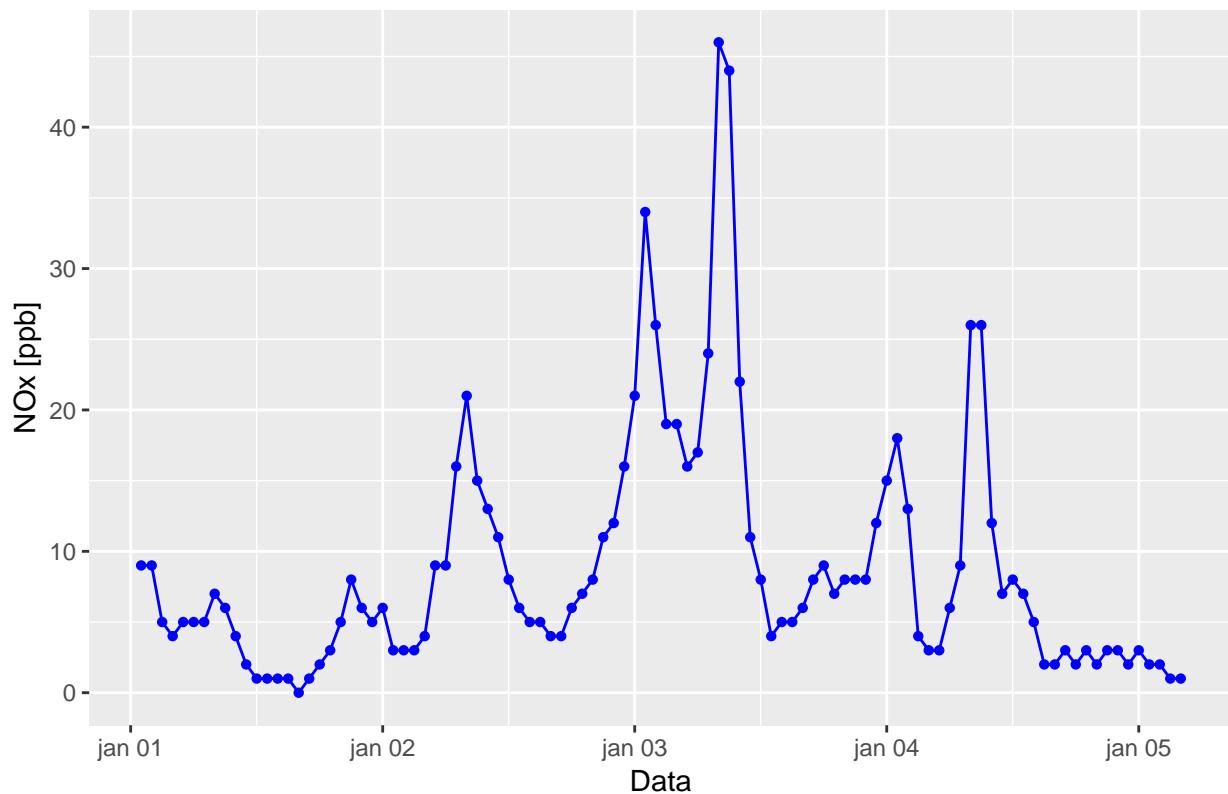
ggplot(data = mpg, aes(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

Figure 5.1: Fonte: <https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf>



```
ggplot(df[1:100,], aes(x = tempo, y = MediaHoraria)) +
  geom_line(color = "blue") + #-- Linhas...
  geom_point(color = "blue", pch = 16) + #-- ... com pontos
  labs(title = "Gráfico mais Bonito", x = "Data", y = "NOx [ppb]") + #-- Títulos
  theme(plot.title = element_text(hjust = 0.5)) #-- Centralizando o título
```

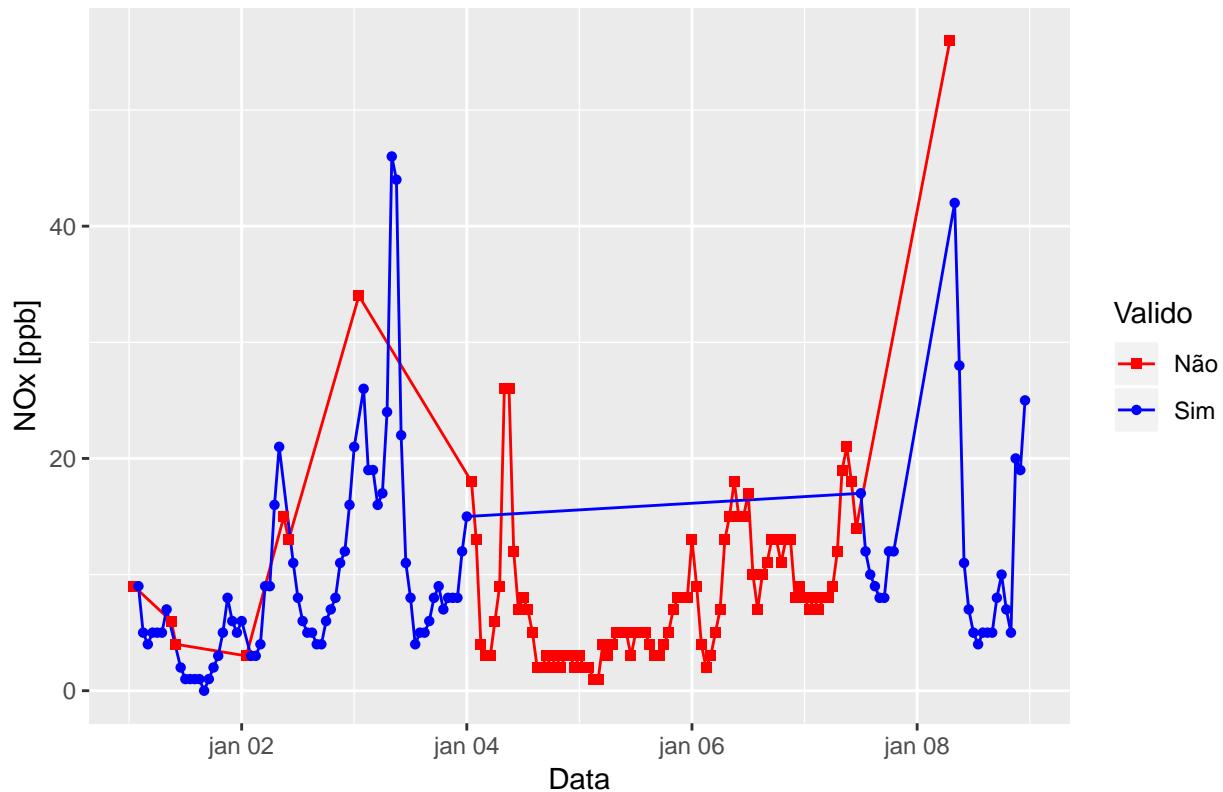
Gráfico mais Bonito



Agora o mais interessante:

```
ggplot(df[1:180,], aes(x = tempo, y = MediaHoraria)) +
  geom_line(aes(color = Valido)) +
  geom_point(aes(color = Valido, shape = Valido)) +
  labs(title = "Dados Válidos e Inválidos", x = "Data", y = "NOx [ppb]") +
  scale_color_manual(values = c("red", "blue")) + #-- Definindo as cores manualmente
  scale_shape_manual(values = c(15, 16)) + #-- Definindo as formas manualmente
  theme(plot.title = element_text(hjust = 0.5))
```

Dados Válidos e Inválidos



Pergunta: Qual a principal diferença entre o código acima e o código usando `plot`?

A função `ggplot` plota apenas data frames, pois ela mapeia as variáveis por nomes de colunas. Assim, é preciso converter matrizes ou arrays em data frames.

Uma vantagem de trabalharmos com data frames, como já vimos antes, é poder manipular esses dados de muitas formas possíveis antes de plotá-los.

Continuação do Exemplo: Extrair algumas informações sobre os dados

Vamos analisar o ano de 2014:

- Em média, como o NOx varia ao longo do dia?
 - E para cada dia da semana?
 - E para cada mês?

Usando algumas outras funções dentro do **Tidyverse**:

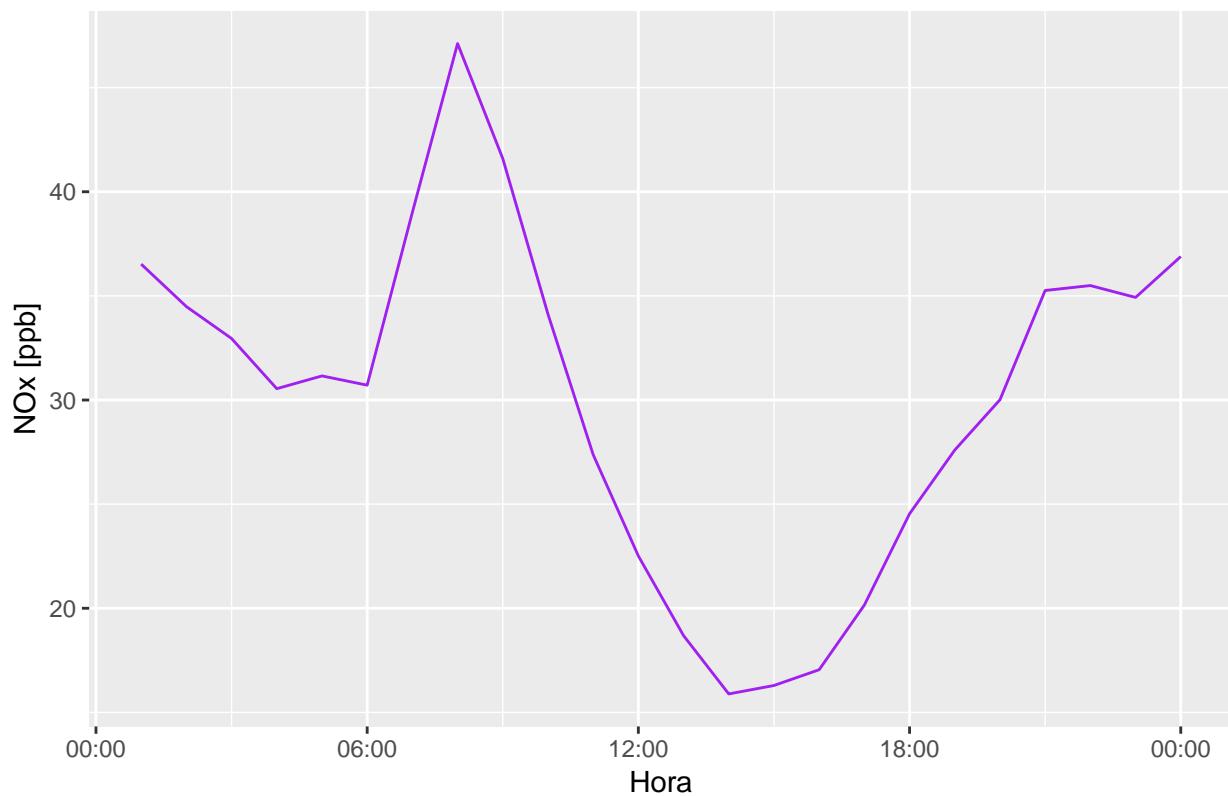
```
library(tidyverse)

df_2014 <- filter(df, ano == "2014")
df_2014_hour <- df_2014 %>% #-- A partir do data frame df_2014
  group_by(Hora) %>% #-- Agrupe os dados pela coluna hora
  summarise(Media = mean(MediaHoraria, na.rm = T)) %>% #-- E calcule as médias,
  #-- salvando em uma coluna nova
  mutate(Hora = as.POSIXct(strptime(Hora, "%H:%M"))) %>% #-- Transformando em data
  ungroup() #-- Desagrupando

ggplot(df_2014_hour) +
  scale_x_datetime(date_labels = "%H:%M") + #-- Formato de data que aparecerá no eixo x
```

```
geom_line(aes(x = Hora, y = Media, group = 1), color = "purple") +
  labs(title = "Média Horária Anual", y = "NOx [ppb]")
```

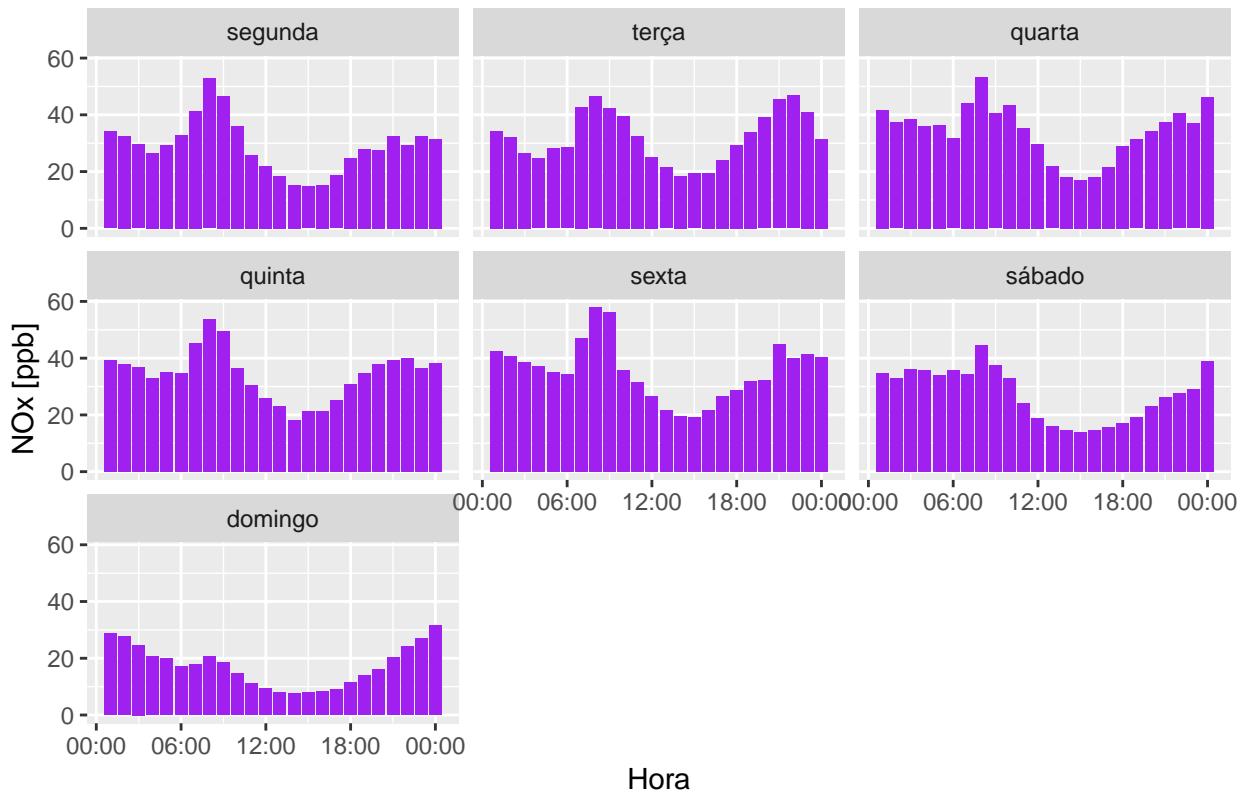
Média Horária Anual



```
df_2014_weekly <- df_2014 %>%
  group_by(Hora, weekdays) %>% #-- Agrupando os dados pelas colunas Hora e weekdays
  summarise(Media = mean(MediaHoraria, na.rm = T)) %>%
  ungroup() %>%
  mutate(Hora = as.POSIXct(strptime(Hora, "%H:%M"))) %>%
  mutate(weekdays = factor(weekdays, levels = c("segunda", "terça", "quarta",
                                                "quinta", "sexta", "sábado",
                                                "domingo))) #-- Ordenando os dias da semana

ggplot(df_2014_weekly) +
  scale_x_datetime(date_labels = "%H:%M") +
  geom_col(aes(x = Hora, y = Media), fill = "purple") +
  labs(title = "Média Horária Anual por Dia da Semana", y = "NOx [ppb]") +
  facet_wrap(~ weekdays) #-- Criando painéis em função do dia da semana
```

Média Horária Anual por Dia da Semana



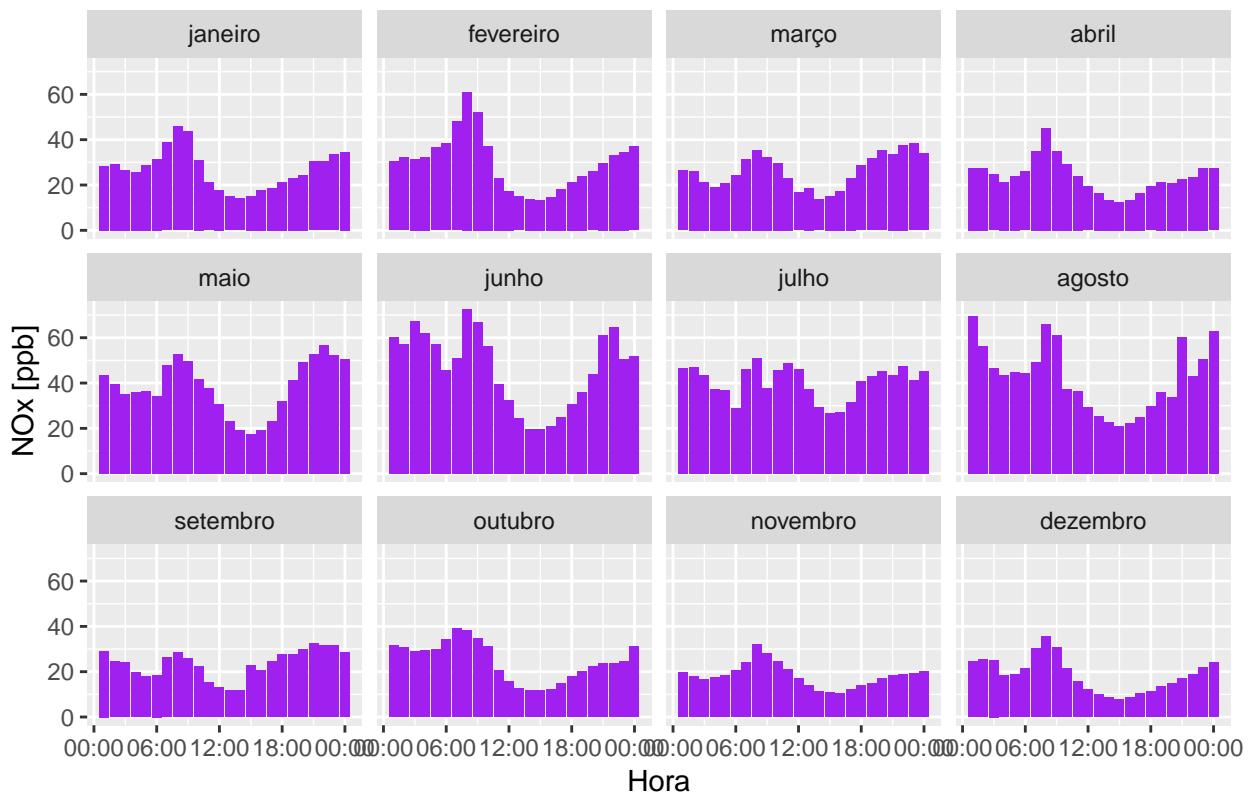
```

df_2014_monthly <- df_2014 %>%
  group_by(Hora, mes) %>% #-- Agrupando os dados pelas colunas Hora e mes
  summarise(Media = mean(MediaHoraria, na.rm = T)) %>%
  ungroup() %>%
  mutate(Hora = as.POSIXct(strptime(Hora, "%H:%M"))) %>%
  mutate(mes = factor(mes, levels = c("janeiro", "fevereiro", "março",
                                      "abril", "maio", "junho", "julho",
                                      "agosto", "setembro", "outubro",
                                      "novembro", "dezembro"))) #-- Ordenando os meses

ggplot(df_2014_monthly) +
  scale_x_datetime(date_labels = "%H:%M") +
  geom_col(aes(x = Hora, y = Media), fill = "purple") +
  labs(title = "Média Horária Anual por Mes", y = "NOx [ppb]") +
  facet_wrap(~ mes) #-- Criando painéis em função do mês

```

Média Horária Anual por Mes



Exercício: Em média, como os dados válidos de NOx variam mensalmente ao longo do ano de 2014? Faça um gráfico.

Desafio: Ainda é possível melhorar os gráficos acima! Pesquise como:

- * Diminuir a quantidade de horários no eixo x
- * Separar por dias da semana e meses a partir da coluna “tempo”, não precisando usar as colunas de caracteres e consequentemente ordená-las manualmente

5.3.1 Explorando outras escalas de cores e temas

Pacotes **veinreport** e **cptcity**

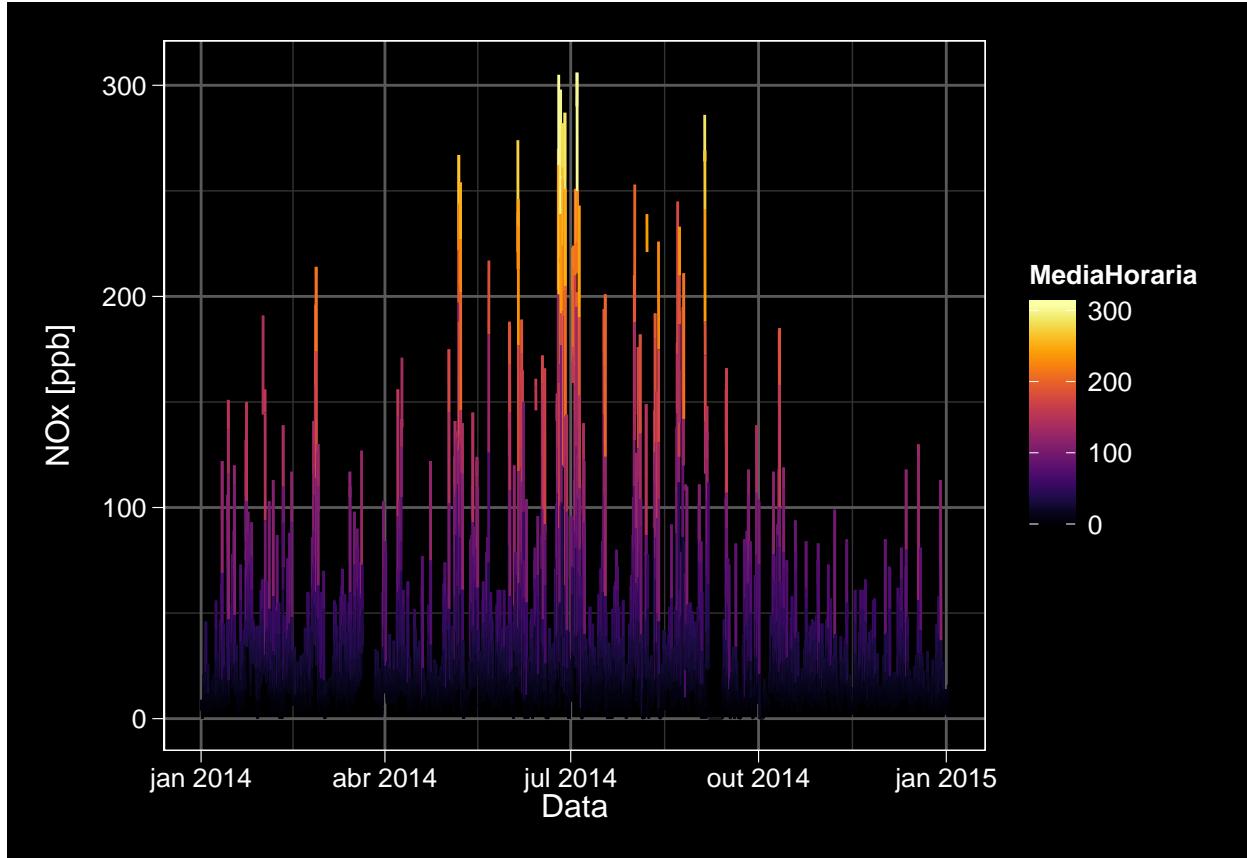
```
devtools::install_github("atmoschem/veinreport")
```

```
library(veinreport)
library(cptcity)
```

Refazendo alguns gráficos:

```
ggplot(df, aes(x = tempo, y = MediaHoraria)) +
  geom_line(aes(color = MediaHoraria)) +
  labs(x = "Data", y = "NOx [ppb]") +
  scale_color_gradientn(colours = cpt()) + #-- Definindo as cores com uma escala gradiente
  theme_black()
```

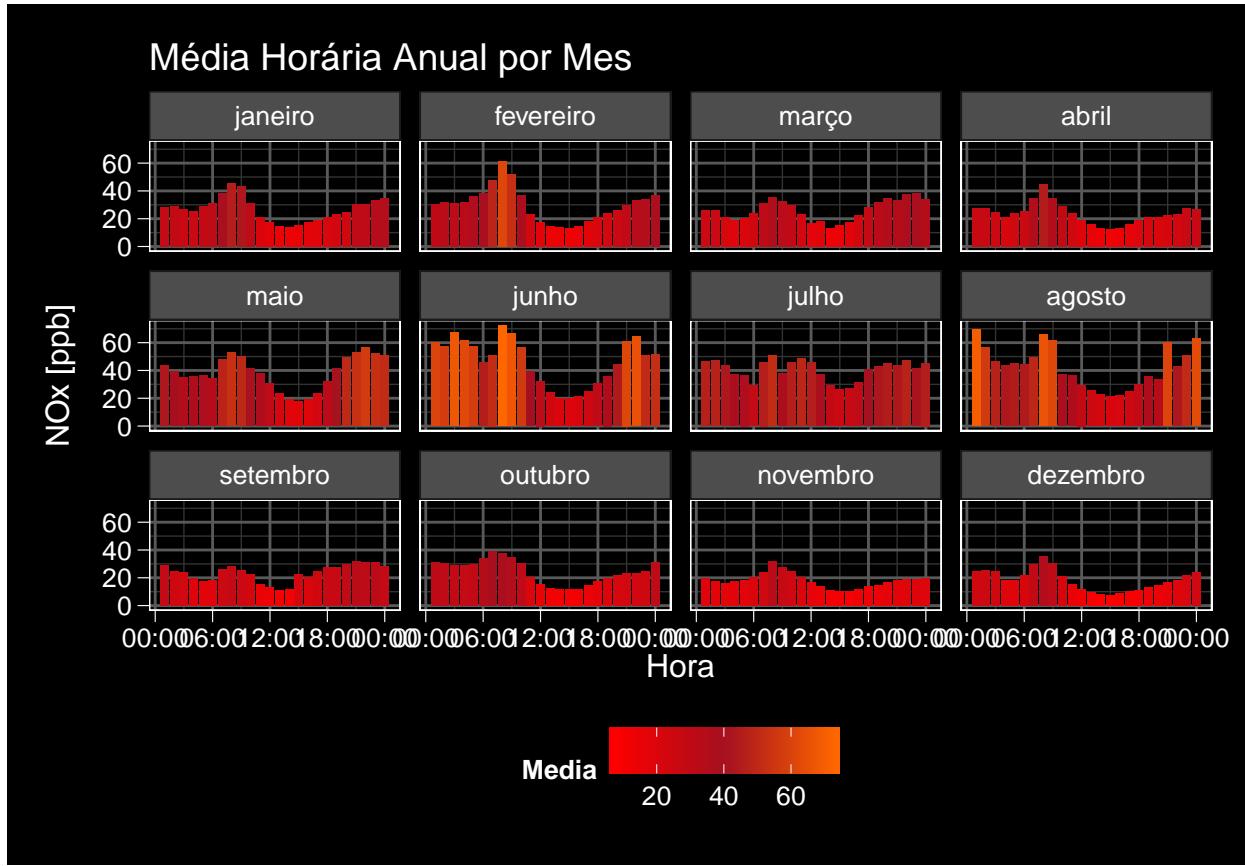
```
## Warning: Removed 1 rows containing missing values (geom_path).
```



Experimentando escalas de cores com a função `lucky`:

```
ggplot(df_2014_monthly) +
  scale_x_datetime(date_labels = "%H:%M") +
  geom_col(aes(x = Hora, y = Media, fill = Media)) +
  labs(title = "Média Horária Anual por Mes", y = "NOx [ppb]") +
  scale_fill_gradientn(colors = lucky()) + #-- Definindo as cores com uma escala gradiente aleatória
  theme_black() +
  theme(legend.position = "bottom", legend.direction = "horizontal") + #-- Colocando a legenda na parte
  facet_wrap(~ mes) #-- Criando painéis em função do mês
```

```
## Colour gradient: nd_terra_Analogous_08a, number: 5770
```



Este é só o começo! Veja aqui um pouco mais das muitas aplicações do ggplot.

Chapter 6

Estruturas de Controle

Em breve.

6.1 if-else

6.2 for

6.3 while

6.4 repeat

6.5 lapply

6.6 sapply

6.7 split

6.8 tapply

6.9 apply

6.10 mapply

Chapter 7

De Scripts a Funções e de Funções a Pacotes

Em breve.

Chapter 8

Geo Spatial: raster, sf e stars

Em breve.

- R Programming for Data Science (Leanpub)
- R Graphics Cookbook (Cookbook for R)
- An Introduction to R (CRAN)
- R Data Import/Export (CRAN)
- RStudio Cheat Sheets (RStudio)
- Webinars and Videos On Demand (RStudio)
- Online Learning (RStudio)
- Learn the tidyverse (Tidyverse)
- Writing R Extensions (CRAN)
- R Internals (CRAN)
- R Language Definition (CRAN)

Quer fazer um documento com o mesmo estilo do nosso? Leia bookdown: Authoring Books and Technical Documents with R Markdown (bookdown.org)