

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)

2018-06-01

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	Objetos do R	11
3.2	Classe	11
3.3	Vetores	11
3.4	Converter objetos	12
3.5	Matrizes e a função <code>matrix</code>	12
3.6	Array	13
3.7	<code>list</code>	14
3.8	Tempo e Data	14
3.9	Fatores	16
3.10	<code>Paste!!</code>	17
3.11	Substituindo caracteres	17
3.12	Substrainndo caracteres	18
3.13	<code>Data.frames</code>	19
4	Importando e Exportando Dados	21
4.1	Data Frames	21
4.2	BASE	25
4.3	Tidyverse	31
4.4	Outros Tipos de Dados	31
5	Plotando	39
5.1	<code>plot</code> (base)	39
5.2	<code>ggplot</code> (ggplot2)	42
6	Estruturas de Controle	53
6.1	<code>if-else</code>	53
6.2	<code>for</code>	53
6.3	<code>while</code>	53
6.4	<code>repeat</code>	53
6.5	<code>lapply</code>	53
6.6	<code>sapply</code>	53

6.7 split	53
6.8 tapply	53
6.9 apply	53
6.10 mapply	53
7 De Scripts a Funções e de Funções a Pacotes	55
8 Geo Spatial: raster, sf e stars	57

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.

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!

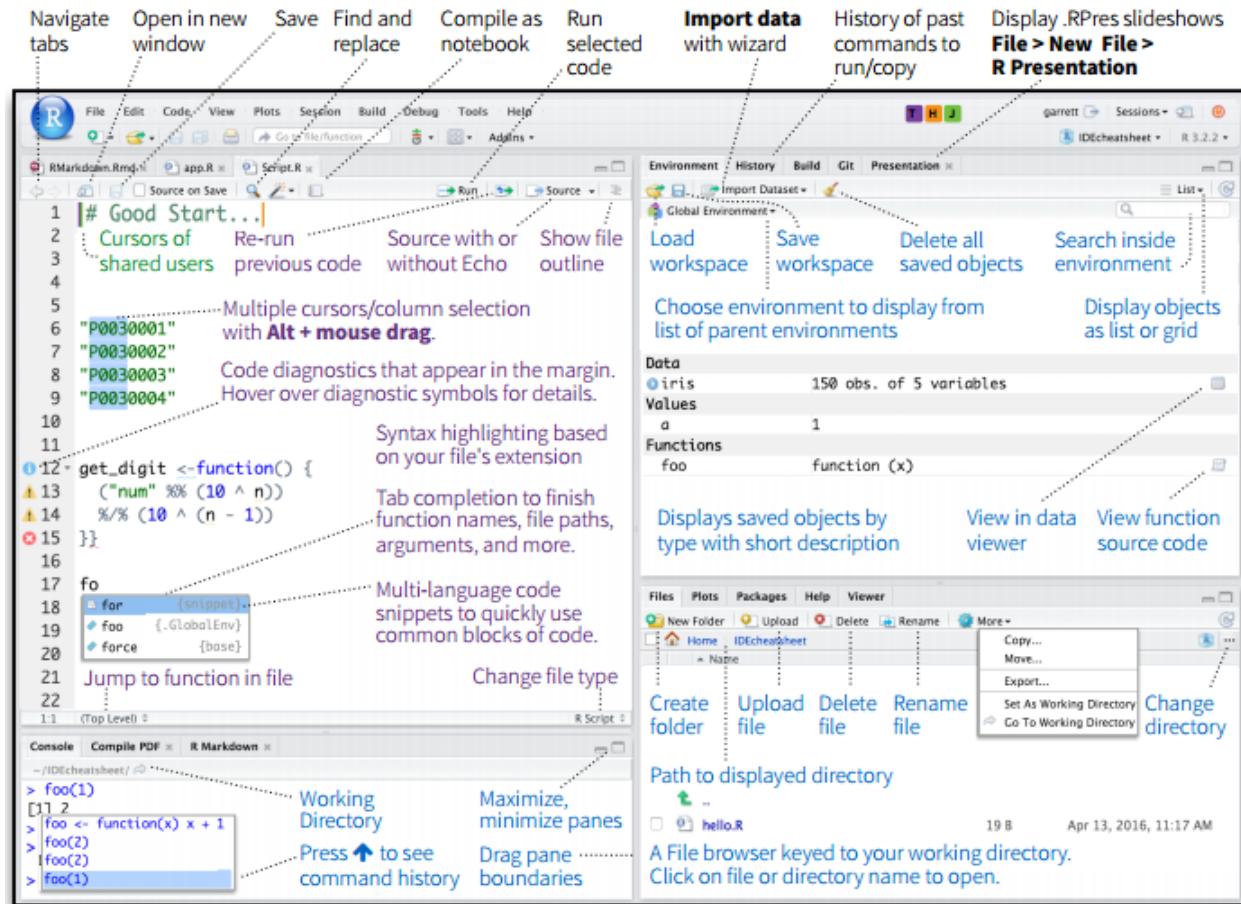


Figure 2.1: Fonte: RStudio IDE Cheatsheet



- **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!

- Iremos focar no Linux, mas R e RStudio estão disponíveis para Windows e Mac também.
- Documentação:
- Intro.
- I/O.
- Quer fazer um pacote? Veja, aqui e aqui.
- Stackoverflow te ajudará em horas de desespero.

3.1 Objetos do R

- Character a
- Numeric 1
- Integer 1
- Complex 0+1i
- Logical TRUE

3.2 Classe

`class` essa função permite ver a classe dos objetos

3.3 Vetores

- `c("A", "C", "D")`
- `1:5 = c(1, 2, 3, 4, 5)`
- `c(TRUE, FALSE)`
- `c(1i, -1i)`
- `c(1, "C", "D") qual é a classe???`
- `c(1, NA, "D") qual é a classe???`
- `c(1, NA, NaN) qual é a classe???`

3.4 Converter objetos

3.4.1 as

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

## Warning: NAs introduzidos por coerção
## [1] 1 NA NA
```

3.4.2 merge e melt

3.5 Matrizes e a função matrix

[linhas, colunas]

- 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
```

usando TAB

```
(m <- matrix(data = 0, nrow = 4, ncol = 4))

##      [,1] [,2] [,3] [,4]
## [1,]     0     0     0     0
## [2,]     0     0     0     0
## [3,]     0     0     0     0
## [4,]     0     0     0     0

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

##      [,1] [,2] [,3] [,4]
## [1,]     1     5     9    13
## [2,]     2     6    10    14
## [3,]     3     7    11    15
## [4,]     4     8    12    16

dim(m1)
```

```
## [1] 4 4

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

```
##      [,1] [,2] [,3] [,4]
## [1,]     1     2     3     4
## [2,]     5     6     7     8
## [3,]     9    10    11    12
## [4,]    13    14    15    16
```

3.6 Array

Permite armazenar diversos elementos, com diversas dimensões. Dessa forma, um array com duas dimensões é o mesmo que uma matriz, dessa forma, podemos armazenar diversas matrizes dentro de um array, mas suas dimensões são pré-estabelecidas.

```
args(array)
```

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

Não esqueça do TAB

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

```
##      [,1]
## [1,]    0
```

```
class(a)
```

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

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

```
## , , 1
```

```
##
```

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

```
## [1,]    0
```

```
class(a)
```

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

```
(a <- array(data = 0, dim = c(2,2,2)))
```

```
## , , 1
```

```
##
```

```
##      [,1] [,2]
```

```
## [1,]    0    0
```

```
## [2,]    0    0
```

```
##
```

```
## , , 2
```

```
##
```

```
##      [,1] [,2]
```

```
## [1,]    0    0
```

```
## [2,]    0    0
```

```
(a <- array(data = 0, dim = c(2,4,4)))
```

```
## , , 1
```

```
##
```

```
##      [,1] [,2] [,3] [,4]
```

```
## [1,]    0    0    0    0
```

```
## [2,]    0    0    0    0
```

```
##
```

```
## , , 2
```

```
##
```

```
##      [,1] [,2] [,3] [,4]
```

```
## [1,]    0    0    0    0
```

```

## [2,]    0    0    0    0
##
## , , 3
##
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    0    0
## [2,]    0    0    0    0
##
## , , 4
##
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    0    0
## [2,]    0    0    0    0
dim(a)

## [1] 2 4 4
(a <- array(data = 0, dim = c(2, 2, 2, 2)))

```

3.7 list

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

```
list(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.

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

```

## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i

```

3.8 Tempo e Data

R também tem classes de tempo e data:

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

## [1] "2018-04-05 12:00:00 GMT"

class(a)

## [1] "POSIXct" "POSIXt"

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

## [1] "2018-04-05 12:00:00 Americas"

Tempo

(d <- ISODatetime(year = 2018, month = 4, day = 5, hour = 0, min = 0, sec = 0,
tz = "Americas/Sao_Paulo"))

## [1] "2018-04-05 Americas"

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

É possível fazer sequências:

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

## [1] "2018-06-01 00:47:02 -03" "2018-06-01 01:47:02 -03"
## [3] "2018-06-01 02:47:02 -03" "2018-06-01 03:47:02 -03"
## [5] "2018-06-01 04:47:02 -03" "2018-06-01 05:47:02 -03"
## [7] "2018-06-01 06:47:02 -03" "2018-06-01 07:47:02 -03"
## [9] "2018-06-01 08:47:02 -03" "2018-06-01 09:47:02 -03"
## [11] "2018-06-01 10:47:02 -03" "2018-06-01 11:47:02 -03"
## [13] "2018-06-01 12:47:02 -03" "2018-06-01 13:47:02 -03"
## [15] "2018-06-01 14:47:02 -03" "2018-06-01 15:47:02 -03"
## [17] "2018-06-01 16:47:02 -03" "2018-06-01 17:47:02 -03"
## [19] "2018-06-01 18:47:02 -03" "2018-06-01 19:47:02 -03"
## [21] "2018-06-01 20:47:02 -03" "2018-06-01 21:47:02 -03"
## [23] "2018-06-01 22:47:02 -03" "2018-06-01 23:47:02 -03"

funções úteis: weekdays, month, julian

weekdays(a)

## [1] "sexta" "sexta" "sexta" "sexta" "sexta" "sexta" "sexta" "sexta"
## [9] "sexta" "sexta" "sexta" "sexta" "sexta" "sexta" "sexta" "sexta"
## [17] "sexta" "sexta" "sexta" "sexta" "sexta" "sexta" "sexta" "sexta"

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] 17683.16 17683.20 17683.24 17683.28 17683.32 17683.37 17683.41
## [8] 17683.45 17683.49 17683.53 17683.57 17683.62 17683.66 17683.70
## [15] 17683.74 17683.78 17683.82 17683.87 17683.91 17683.95 17683.99
## [22] 17684.03 17684.07 17684.12
## attr(",origin")
```

```
## [1] "1970-01-01 GMT"
```

*Para mais informações: https://en.wikipedia.org/wiki/Julian_day:

3.9 Fatores

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

São variáveis que representam categorias, como por exemplo, dias da semana.

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

```
## [1] "character"
factor(aa)
```

```
## [1] sexta   sexta   sexta   sexta   sexta   sexta   sexta   sexta   sexta
## [9] sexta   sexta   sexta   sexta   sexta   sexta   sexta   sexta   sexta
## [17] sexta   sexta   sexta   sexta   sexta   sexta   sexta   sexta   sexta
## [25] sábado  sábado  sábado  sábado  sábado  sábado  sábado  sábado  sábado
## [33] sábado  sábado  sábado  sábado  sábado  sábado  sábado  sábado  sábado
## [41] sábado  sábado  sábado  sábado  sábado  sábado  sábado  sábado  sábado
## [49] domingo domingo domingo domingo domingo domingo domingo domingo
## [57] domingo domingo domingo domingo domingo domingo domingo domingo
## [65] domingo domingo domingo domingo domingo domingo domingo domingo
## [73] segunda segunda segunda segunda segunda segunda segunda segunda
## [81] segunda segunda segunda segunda segunda segunda segunda segunda
## [89] segunda segunda segunda segunda segunda segunda segunda segunda
## [97] terça   terça   terça   terça   terça   terça   terça   terça   terça
## [105] terça   terça   terça   terça   terça   terça   terça   terça   terça
## [113] terça   terça   terça   terça   terça   terça   terça   terça   terça
## [121] quarta  quarta  quarta  quarta  quarta  quarta  quarta  quarta  quarta
## [129] quarta  quarta  quarta  quarta  quarta  quarta  quarta  quarta  quarta
## [137] quarta  quarta  quarta  quarta  quarta  quarta  quarta  quarta  quarta
## [145] quinta  quinta  quinta  quinta  quinta  quinta  quinta  quinta  quinta
## [153] quinta  quinta  quinta  quinta  quinta  quinta  quinta  quinta  quinta
## [161] quinta  quinta  quinta  quinta  quinta  quinta  quinta  quinta  quinta
## Levels: domingo quarta quinta sábado segunda sexta terça
```

São muito úteis para regressões, plots e resumos estatísticos.

Olhe os **Levels**

Então:

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

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

3.10 Paste!!

Como concatenar, juntar, grudar arquivos, palavras, etc etc... é com `paste` e `paste0`. Estas funções são muito simples mas muito úteis!!!

```
args(paste0)
```

```
## function (... , collapse = NULL)
## NULL
args(paste)

## function (... , sep = " ", collapse = NULL)
## NULL
```

A diferença entre `paste0` e `paste`, é que `paste0` não tem uma separação de characteres por default, mas `paste` usa espaço.

```
path <- getwd()
nome <- "SeuNome"
paste(path, nome)

## [1] "/home/sergio/Documentos/cursoR SeuNome"
paste0(path, nome)

## [1] "/home/sergio/Documentos/cursoRSeuNome"
paste(path, "/", nome)

## [1] "/home/sergio/Documentos/cursoR / SeuNome"
paste(path, "-/_****;;;////", nome)

## [1] "/home/sergio/Documentos/cursoR /-_****;;;//// SeuNome"
class(paste(path, "-/_****;;;////", nome))

## [1] "character"
paste0(nome, "_", 1:10) #Função vetorizada

## [1] "SeuNome_1"  "SeuNome_2"  "SeuNome_3"  "SeuNome_4"  "SeuNome_5"
## [6] "SeuNome_6"  "SeuNome_7"  "SeuNome_8"  "SeuNome_9"  "SeuNome_10"
paste0(1:10) #Função vetorizada

## [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"
class(paste0(1:10))

## [1] "character"
```

3.11 Substituindo characteres

Vamos supor que tu vai ler um banco de dados, e uma coluna tem o character “BB”. Vamos supor que nessa coluna tem um numero, mas tu precisa `remover` o simbolo “BB” para logo convertir para numero. A função que vamos usar que podemos usar é `gsub`.

```

numeros <- 1:50
b <- round(runif(n = 10, min = 1, max = 50))
numeros[b] <- paste0(numeros, "BB")

## Warning in numeros[b] <- paste0(numeros, "BB"): número de itens para para
## substituir não é um múltiplo do comprimento do substituto
numeros

## [1] "1"    "2"    "5BB"   "4"    "5"    "6"    "7"    "8"    "9"    "10"
## [11] "1BB"  "10BB"  "13"    "14"   "15"   "16"   "17"   "18"   "19"   "20"
## [21] "21"   "22"   "23"    "24"   "25"   "26"   "27"   "8BB"  "29"   "30"
## [31] "31"   "7BB"  "3BB"   "34"   "35"   "36"   "37"   "38"   "39"   "40"
## [41] "41"   "9BB"  "43"    "2BB"  "45"   "46"   "4BB"  "6BB"  "49"   "50"

args(gsub)

## function (pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
##          fixed = FALSE, useBytes = FALSE)
## NULL

nn <- gsub(pattern = "BB", replacement = "", x = numeros)
class(nn)

## [1] "character"

as.numeric(nn)

## [1] 1 2 5 4 5 6 7 8 9 10 1 10 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 8 29 30 31 7 3 34 35 36 37 38 39 40 41 9 43 2 45 46
## [47] 4 6 49 50

```

3.12 Substraindo caracteres

Agora vamos supor que temos grudado ultimo character com BB, A função que vamos usar é `substr`:

```
args(substr)
```

```

## function (x, start, stop)
## NULL

```

Com `x` é o vetor, `start` primeiro elemento em ser reemplazado e `stop` ultimo elemento em ser reemplazado

Vamos super uma sequencia de 1 ate 10 e tem grudados as letras BB

```
(numeros<- paste0(1:9, "BB"))
```

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

Para remover os ultimos dois characters seria:

```
substr(numeros, 0, 1)
```

```

## [1] "1" "2" "3" "4" "5" "6" "7" "8" "9"
substr(numeros, 2, 3)

```

```
## [1] "BB" "BB" "BB" "BB" "BB" "BB" "BB" "BB" "BB"
```

3.13 Data.frames

lembre ?data.frame

Lembram uma planilha EXCEL... Mais ou menos

É uma classe bem especial, tem elementos de matriz mas o modo é lista

```
args(data.frame)
```

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

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

```
##   a
## 1 1
## 2 2
## 3 3
```

```
names(df)
```

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

```
class(df)
```

```
## [1] "data.frame"
```

```
mode(df)
```

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

Podemos utilizar para armazenar dados, sendo que um data.frame é sempre composto por vetores com comprimento IGUAL

```
nrow(df)
```

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

```
ncol(df)
```

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

```
dim(df)
```

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

Algumas operações que pode fazer com data.frames são:

- rbind: une verticalmente (por linhas... row) duas data.frames
- cbind: Une horizontalmente duas data.frames por colunas
- merge: Une horizontalmente data.frames por coincidencia de dados nas colunas

fazia?

```
?rbind
```

```
?cbind
```

```
?merge
```

Exemplo:

```
a <- data.frame(id = c("a", "b", "c"))
b <- data.frame(id = c("Z", "b", "D"))
merge(x = a, y = b, by = "id")
```

```
## id  
## 1 b
```

Chapter 4

Importando e Exportando Dados

4.1 Data Frames

Probabelmente um dos primeiros objetos que vamos usar quando começamos usar R. Pensa num data-frame como uma planilha de Libreoffice (o excel). Os data-frame pode ser criados como foi visto na seção anterior. O principal, é que temos varias funções para ler data-frames no R, entre elas

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

Agora vamos a ler dados do repositório usando `read.table`, mas primeiro vamos lembrar que se tu precisar ver a ajuda da função, tem que escrever no R `?read.table`. Então, agora vamos ver os argumentos da função:

```
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, nrow = -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
```

Aqui vêm-se os valores default dos argumentos da função `read.table`. O terceiro argumento é `sep`, com valores por default = “”.

```
df <- read.table("https://raw.githubusercontent.com/ibarraespinoza/cursoR/master/dados/NOXIPEN2014.txt")
```

Agora vamos usar a funções `head` and `tail` para ver as primeiras e as ultimas 6 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
```

```

##   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(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
##   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

```

Agora vamos ler os mesmos dados com outro formato e testar se read.table funciona do mesmo jeito

```

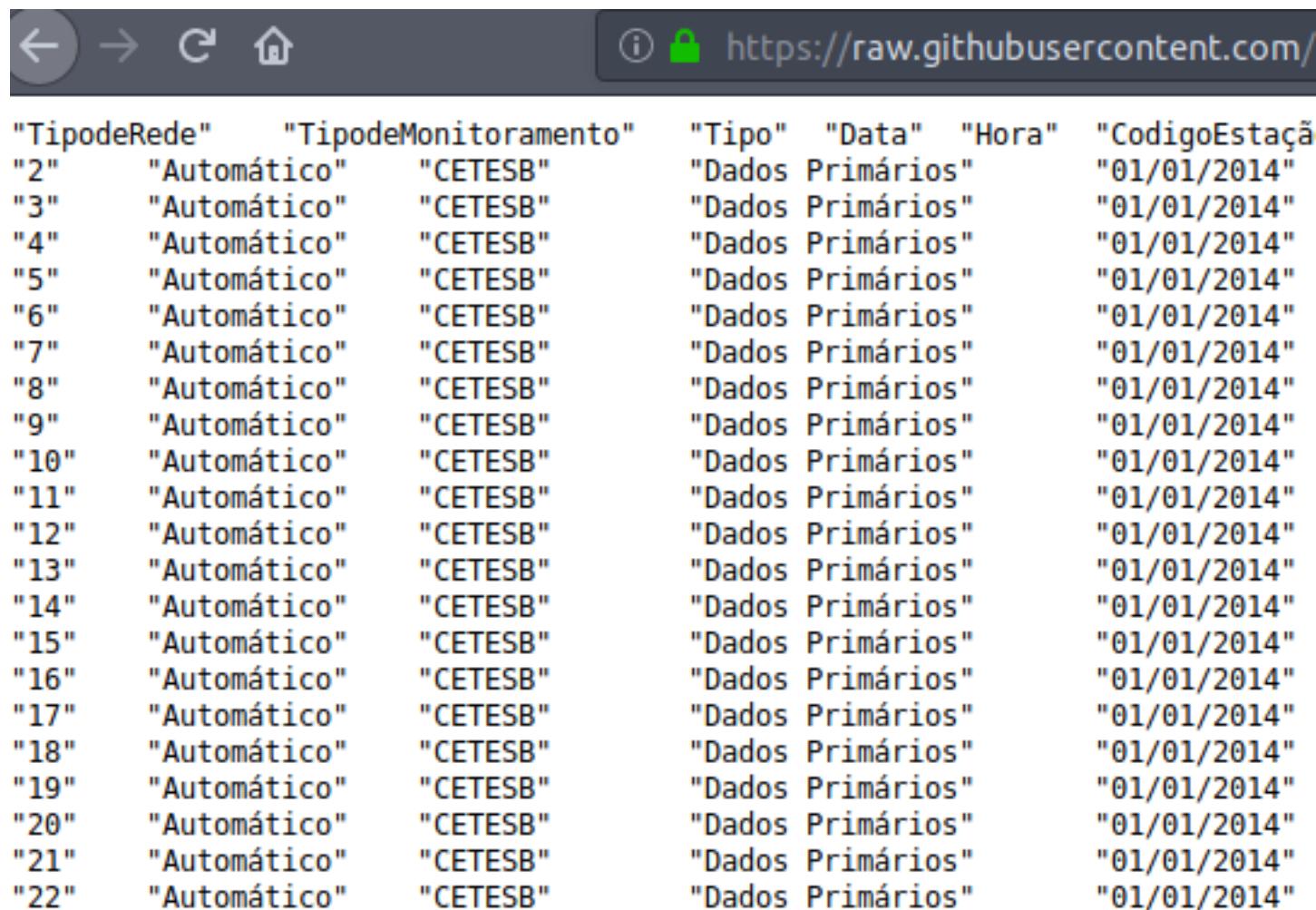
df2 <- read.table("https://raw.githubusercontent.com/ibarraespinoza/cursoR/master/dados/NOXIPEN2014v2.txt")
# Error in scan(file = file, what = what, sep = sep, quote = quote, dec = dec, :
# linha 1 não tinha 6 elementos

```

Vemos a mensagem de erro, mas o que quer dizer.

Se tu receber um banco de dados tipo .txt e quer abrir no R... ABRE ELE COM BLOCO DE NOTAS PRIMEIRO!!!

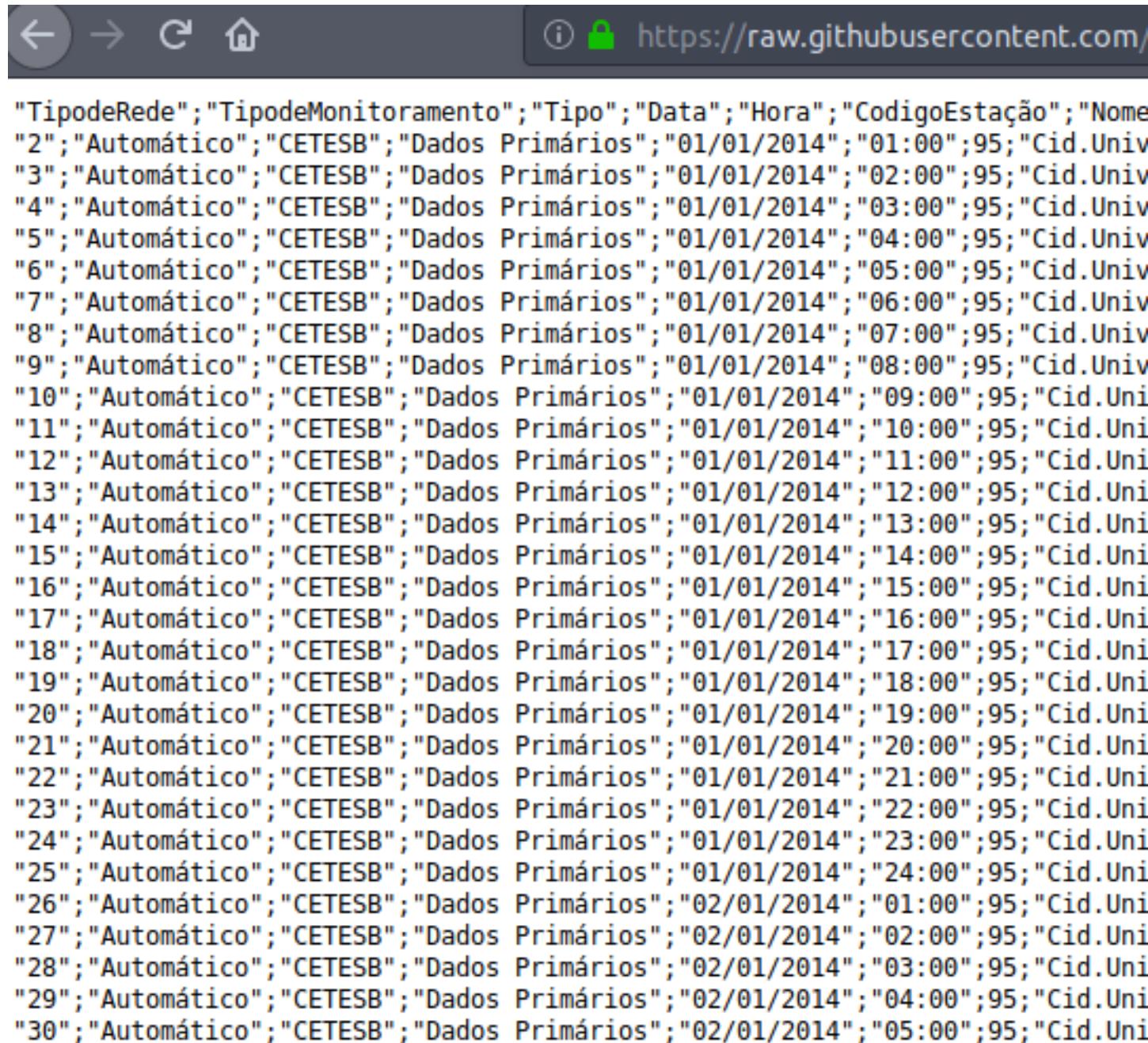
O primeiro arquivo:



The screenshot shows a web browser window with the URL <https://raw.githubusercontent.com/>. The page displays a table of data frames. The columns are labeled "TipodeRede", "TipodeMonitoramento", "Tipo", "Data", "Hora", and "CodigoEstação". The rows show 22 entries, all with the value "Automático" in the first two columns and "CETESB" in the third column. The fourth column contains "Dados Primários", the fifth column contains "01/01/2014", and the sixth column contains "01/01/2014".

TipodeRede	TipodeMonitoramento	Tipo	Data	Hora	CodigoEstação
"2"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"3"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"4"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"5"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"6"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"7"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"8"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"9"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"10"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"11"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"12"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"13"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"14"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"15"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"16"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"17"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"18"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"19"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"20"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"21"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"
"22"	"Automático"	"CETESB"	"Dados Primários"	"01/01/2014"	"01/01/2014"

O segundo arquivo é:



The screenshot shows a web browser displaying a large amount of data. The URL in the address bar is <https://raw.githubusercontent.com/ibarraespinoza/cursoR/master/dados/NOXIPEN2014v2.txt>. The data consists of approximately 30 rows of text, each containing several fields separated by semicolons. The fields include 'TipodeRede', 'TipodeMonitoramento', 'Tipo', 'Data', 'Hora', 'CodigoEstação', 'Nome', and numerical values ranging from 2 to 95. The browser interface includes standard navigation buttons (back, forward, search) and a status bar at the bottom.

```

"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

```

qual é a diferença?

Como vemos o segundo arquivo tem separação de ";", entao, temos que ler o arquivo assim:

```

df2 <- read.table("https://raw.githubusercontent.com/ibarraespinoza/cursoR/master/dados/NOXIPEN2014v2.txt",
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

```

Quais dificuldades tu já enfrentou importando dados?

4.2 BASE

4.2.1 Exportando texto com `write.table`

Exportar é bem fácil, mas se sabemos os argumentos das funções, vai ser mais eficiente ainda. Vamos `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 caracter.

sep é como vão ser separadas as colunas. Se tu quer abrir o arquivo com Excel, poderia separar com “;”, “,”, “ ”, “ ”... Depende como tu quer.

eol quer dizer *end of line*, e é para ver a forma de colocar o “end of line”

row.names.. esta TRUE mas SEMPRE SEMPRE SEMPRE COLOCA:

row.names = FALSE. Se não, R vai adicionar uma coluna com os indices das linhas....

col.names se tu quer o nome nas colunas...

PRATICA!

4.2.2 Exportando objetos com **save**

```
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
```

save salva o objeto com a extensão .rda. Para carregar de volta o objeto, tem que ser feito com a função load

```
args(load)
```

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

O que pode ser ruim, porque as vezes tu esqueceu o nome do objeto no ambiente de R. Por exemplo, tu salvou o arquivo

```
save(frenteFria, file = "FrenteQuente.rda")
```

logo tu carrega

```
load("FrenteQuente.rda")
```

acreditando que vai ter tua frente quente, mas o nome do objeto no ambiente de R é frenteDria... então, tem que ficar de olho, e como somos imperfeito, vai dar merda....

O melhor da função é que permite salvar com tipos de compressão, por exemplo compress = “xz”.

4.2.3 Exportando objetos com **saveRDS**

Esta é uma das minhas funções favoritas no R

```
args(saveRDS)
```

```
## function (object, file = "", ascii = FALSE, version = NULL, compress = TRUE,
##          refhook = NULL)
## NULL
```

e

```
args(readRDS)

## function (file, refhook = NULL)
## NULL
```

Tu consegue salvar o objeto R de forma serializada e compactada com o argumento **compress** mas o melhor é quando vai chamar o objeto de volta ao R. Agora tu usa o **readRDS** e coloca o nome que tu quiser.

```
saveRDS(FrenteQuente, "FrenteQuente.rds")

frenteQ <- readRDS("FrenteQuente.rds")
```

4.2.4 Processando nossa data-frame

Tem numeroas formas e pacotes para ordenar, arranjar (Arrange), mutar e cambiar as data-frames. As mais conhecidas são provavelmente do universe *tidyverse* com o famoso pacote *dplyr*. Mas, nesta curso vamos focar em **base**.

Vamos então revisar a classe de cada columna do nosso data-frame com a função **sapply**, apresentada em outro capitulo, mas se quiser, da uma olhada em **?sapply**.

```
sapply(df, class)

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

Quando nos trabalhamos com series de tempo, é importante ter a variabel de tempo reconhecida como “tempo”, especificamente como classe “POSIXct”. Mas, a classe de Data é “factor” e de Hora tambem “factor”, o que é ruim. Então, vamos criar uma variabel de tempo mais standard com formato 2018-06-01 00:47:05.

Para isso temos que grudar as variabel Data e Hora. Faremos isso numa nova varaiel chamada **tempo_char**, adicionando ela diretamente no df com o cífrão DOLLAR \$. O grude pode ser feito 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"
```

Esta melhorando mas ainda tem classe character.

Para convertir a nossa classe POSIXct podemos usar a função **as.POSIXct** (olha **as.POSIXct**). Seus argumentos são:

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

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

```
## NULL
```

Então, vamos criar outra variável tempo o formato POSIXct

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

## [1] "2014-01-01 01:00:00 Americas" "2014-01-01 02:00:00 Americas"
## [3] "2014-01-01 03:00:00 Americas" "2014-01-01 04:00:00 Americas"
## [5] "2014-01-01 05:00:00 Americas" "2014-01-01 06:00:00 Americas"

class(df$tempo)

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

Agora, vamos a extraer 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] 16071.04 16071.08 16071.12 16071.17 16071.21 16071.25

df$ano <- format(df$tempo, "%Y")
```

4.2.5 aggregate

Vamos a carregar a nossa data.frame. Primeiro uma olhada

```
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	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 16071.04 days 2014
## 3 2014-01-01 02:00:00 quarta janeiro 16071.08 days 2014
## 4 2014-01-01 03:00:00 quarta janeiro 16071.12 days 2014
## 5 2014-01-01 04:00:00 quarta janeiro 16071.17 days 2014
## 6 2014-01-01 05:00:00 quarta janeiro 16071.21 days 2014
## 7 2014-01-01 06:00:00 quarta janeiro 16071.25 days 2014

```

Poderíamos calcular a media horaria por dia da semana. Então:

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

```

##  Group.1      x
## 1 domingo 20327
## 2 quarta 40180
## 3 quinta 41199
## 4 sábado 32298
## 5 segunda 34057
## 6 sexta 42558
## 7 terça 37904

names(dff) <- c("dias", "MediaHoraria")

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

##      dias MediaHoraria      sd
## 1 domingo      20327 20327
## 2 quarta       40180 40180
## 3 quinta       41199 41199
## 4 sábado       32298 32298
## 5 segunda      34057 34057
## 6 sexta        42558 42558
## 7 terça        37904 37904

```

4.2.6 subset

Como poderíamos escolher só o mes 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 16071.04 days 2014
## 3 2014-01-01 02:00:00 quarta janeiro 16071.08 days 2014
## 4 2014-01-01 03:00:00 quarta janeiro 16071.12 days 2014
## 5 2014-01-01 04:00:00 quarta janeiro 16071.17 days 2014
## 6 2014-01-01 05:00:00 quarta janeiro 16071.21 days 2014
## 7 2014-01-01 06:00:00 quarta janeiro 16071.25 days 2014

```

Mes janeiro pero solo o valor mediahoraria, que retorna um vetor numerico

```
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"
```

Mas vamos salvar o nosso “df”

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

4.2.7 data.table, read_xl e mais

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 herencia (inherits) da classe `data.frame`.

Vamos ver como funciona `data.table` lendo o dois arquivos e comparar quanto tempo demoram cada um.

```
df1 <- print(system.time(read.table("https://raw.githubusercontent.com/ibarraespinoza/cursoR/master/dados/df1.csv")))
```

```
##   user  system elapsed
```

```
##   0.107   0.020   1.211
library(data.table)
df2 <- print(system.time(fread("https://raw.githubusercontent.com/ibarraespinoza/cursoR/master/dados/NO2_2010_2011.nc")))

##    user  system elapsed
##   0.036   0.000   0.102
```

olha que estamos usando a função `fread`.

`read_xl` é mais uma função do universo tidyverse que permite importar excel no R, diretamente e inteligentemente.

4.3 Tidyverse

4.3.1 Leitura %>% Processamento

4.4 Outros Tipos de Dados

4.4.1 NetCDF (Linux o MacOS)

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` pode ser usado para acessar a essa biblioteca, os comandos abaixo instalam e carregam esse pacote:

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

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

Um exemplo de NetCDF:

```
wrfinput <- paste0(system.file("extdata", package = "eixport"), "/wrfinput_d01")
```

Abre o arquivo .nc

```
wrf <- ncdf4::nc_open(wrfinput)
print(wrf)
```

```
## File /home/sergio/R/x86_64-pc-linux-gnu-library/3.4/eixport/extdata/wrfinput_d01 (NC_FORMAT_CLASSIC)
##
##      3 variables (excluding dimension variables):
##        char Times[DateStrLen,Time]
##        float XLAT[west_east,south_north]
##          MemoryOrder: XY
##          description: LATITUDE, SOUTH IS NEGATIVE
##          units: degree north
##          stagger:
##          FieldType: 104
##        float XLONG[west_east,south_north]
##          MemoryOrder: XY
```

```
##           description: LONGITUDE, WEST IS NEGATIVE
##           units: degree east
##           stagger:
##           FieldType: 104
##
## 4 dimensions:
##           DateStrLen  Size:19
##           Time   Size:1 *** is unlimited ***
##           west_east  Size:149
##           south_north  Size:99
##
## 79 global attributes:
##           TITLE: OUTPUT FROM REAL_EM V3.9.1.1 PREPROCESSOR
##           START_DATE: 2011-08-01_00:00:00
##           SIMULATION_START_DATE: 2011-08-01_00:00:00
##           WEST-EAST_GRID_DIMENSION: 150
##           SOUTH-NORTH_GRID_DIMENSION: 100
##           BOTTOM-TOP_GRID_DIMENSION: 35
##           DX: 9000
##           DY: 9000
##           GRIDTYPE: C
##           DIFF_OPT: 1
##           KM_OPT: 4
##           DAMP_OPT: 3
##           DAMPCOEF: 0.200000002980232
##           KHDIF: 0
##           KVDIR: 0
##           MP_PHYSICS: 10
##           RA_LW_PHYSICS: 4
##           RA_SW_PHYSICS: 4
##           SF_SFCLAY_PHYSICS: 1
##           SF_SURFACE_PHYSICS: 2
##           BL_PBL_PHYSICS: 1
##           CU_PHYSICS: 11
##           SF_LAKE_PHYSICS: 0
##           SURFACE_INPUT_SOURCE: 1
##           SST_UPDATE: 0
##           GRID_FDDA: 0
##           GFDDA_INTERVAL_M: 0
##           GFDDA_END_H: 0
##           GRID_SFDDA: 0
##           SGFDDA_INTERVAL_M: 0
##           SGFDDA_END_H: 0
##           HYPBOMETRIC_OPT: 2
##           USE_THETA_M: 0
##           USE_MAXW_LEVEL: 0
##           USE_TROP_LEVEL: 0
##           GWD_OPT: 0
##           SF_URBAN_PHYSICS: 1
##           SF_OCEAN_PHYSICS: 0
##           SIMULATION_INITIALIZATION_TYPE: REAL-DATA CASE
##           WEST-EAST_PATCH_START_UNSTAG: 1
##           WEST-EAST_PATCH_END_UNSTAG: 149
##           WEST-EAST_PATCH_START_STAG: 1
```

```

##      WEST-EAST_PATCH_END_STAG: 150
##      SOUTH-NORTH_PATCH_START_UNSTAG: 1
##      SOUTH-NORTH_PATCH_END_UNSTAG: 99
##      SOUTH-NORTH_PATCH_START_STAG: 1
##      SOUTH-NORTH_PATCH_END_STAG: 100
##      BOTTOM-TOP_PATCH_START_UNSTAG: 1
##      BOTTOM-TOP_PATCH_END_UNSTAG: 34
##      BOTTOM-TOP_PATCH_START_STAG: 1
##      BOTTOM-TOP_PATCH_END_STAG: 35
##      GRID_ID: 1
##      PARENT_ID: 1
##      I_PARENT_START: 1
##      J_PARENT_START: 1
##      PARENT_GRID_RATIO: 1
##      DT: 45
##      CEN_LAT: -23.5499954223633
##      CEN_LON: -45
##      TRUELAT1: -23
##      TRUELAT2: -24
##      MOAD_CEN_LAT: -23.5499954223633
##      STAND_LON: -45
##      POLE_LAT: 90
##      POLE_LON: 0
##      GMT: 0
##      JULYR: 2011
##      JULDAY: 213
##      MAP_PROJ: 1
##      MAP_PROJ_CHAR: Lambert Conformal
##      MMINLU: MODIFIED_IGBP_MODIS_NOAH
##      NUM_LAND_CAT: 21
##      ISWATER: 17
##      ISLAKE: 21
##      ISICE: 15
##      ISURBAN: 13
##      ISOILWATER: 14
##      HYBRID_OPT: -1
##      ETAC: 0

```

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

```

class(wrf)

## [1] "ncdf4"
# fazer print(wrf)

```

que mostra o nome do arquivo (e versão da biblioteca usada para criar), número de variáveis (92 no arquivo de exemplo), uma descrição de cada variável (incluindo atributos) as dimensões (13 para esse arquivo) e os atributos globais.

Agora vamos abrir alguma variável:

```

names(wrf$var)          # print no nome de cada variavel

## [1] "Times" "XLAT"   "XLONG"

```

```
#Times <- ncdf4::ncvar_get(wrf, varid = "Times") # escolho você picachu
#class(Times)
```

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

```
ncatt_get(wrf, "Times", verbose = T) # ou ncatt_get(wrf, "TT", verbose = T)
```

```
## [1] "ncatt_get: entering"
## [1] "ncatt_get: is NOT a global att"
## [1] "ncatt_get: getting object id"
## [1] "vobjtovarid4: entering"
## [1] "Variable named Times found in file with varid= 65536 2"
## [1] "ncatt_get: calling ncatt_get_inner for a non-global att"
## [1] "ncatt_get_inner: entering with ncid= 65536 varid= 2 attname= NA"
## [1] "ncatt_get_inner: no attname specified, returning a list with name/value pairs *****"
## [1] "ncatt_get_inner: number of atts for this var [or file, if global]: 0"
## [1] "ncatt_get_inner: no attributes for this var/file, returning empty list"

## list()
```

praticamente a mesma informação do outro caso:

```
float TT[west_east,south_north,num_metgrid_levels,Time]
FieldType: 104
MemoryOrder: XYZ
units: K
description: Temperature
stagger: M
sr_x: 1
sr_y: 1
```

como temos apenas 1 tempo essa dimensão é desconsiderada para simplificar.

A latitude de cada ponto de grade, assim como longitude níveis e tempo podem ser extraídas:

```
lat <- ncvar_get(wrf, "XLAT")
lon <- ncvar_get(wrf, "XLONG")
time <- ncvar_get(wrf, "Times")
```

O metadado de Longitude:

```
float XLONG[west_east,south_north,Time]
FieldType: 104
MemoryOrder: XY
units: degrees longitude
description: Longitude on mass grid
stagger: M
sr_x: 1
sr_y: 1
```

Latitude:

```
float XLAT[west_east,south_north,Time]
FieldType: 104
MemoryOrder: XY
units: degrees latitude
description: Latitude on mass grid
stagger: M
sr_x: 1
```

```
sr_y: 1
```

e a altura:

```
float GHT[west_east,south_north,num_metgrid_levels,Time]
FieldType: 104
MemoryOrder: XYZ
units: m
description: Height
stagger: M
sr_x: 1
sr_y: 1
```

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`.

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(wrf) # ou nc_sync(wrf)
```

Novas dimensões e também novas variáveis podem ser criadas com `ncvar_def` e `ncvar_add` em um arquivo aberto com permissão de leitura, como por exemplo:

```
wrf      <- nc_open("dados/met_em.d03.2016-01-10.nc", write = TRUE)
extrema <- ncvar_def(name = "Tex",
                      units = "K",
                      dim = list(wrf$dim$west_east,
                                 wrf$dim$south_north,
                                 wrf$dim$Time),
                      missval = -999,
                      longname = "temperatura extrema")
ncvar_add(wrf, extrema)
names(wrf$var)
nc_close(wrf)
```

Se esse arquivo for aberto novamente vai conter 93 variáveis junto com a variável `Tex` da forma que definimos, caso queria os mesmos atributos que as demais é só usar a função `ncatt_get` na variável.

```
wrf      <- ncdf4::nc_open("dados/met_em.d03.2016-01-10.nc", write=T)
print(wrf)[1:10]
```

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.

4.4.2 Dados Binários

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.

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

```
args(readBin)
```

```
## function (con, what, n = 1L, size = NA_integer_, signed = TRUE,
##          endian = .Platform$endian)
## NULL
```

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>

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

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

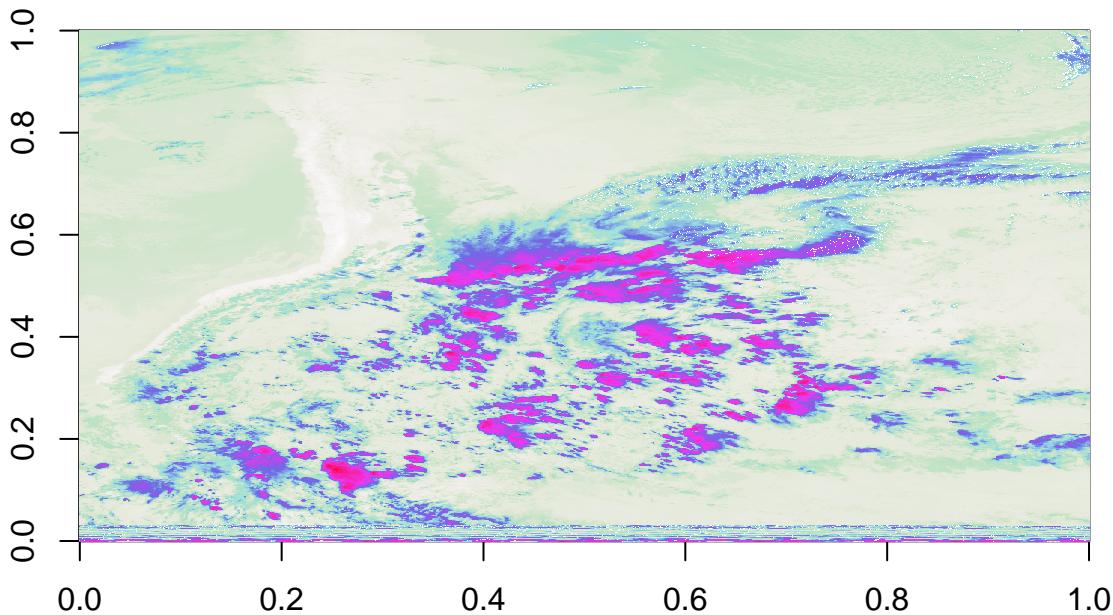
Note que o argumento `endian` por default é `.Platform$endian`. Se rodarmos `.Platform$endian` no R obteremos a ordenação dos bytes (“big” ou “little”) utilizada pela plataforma que estamos usando.

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).

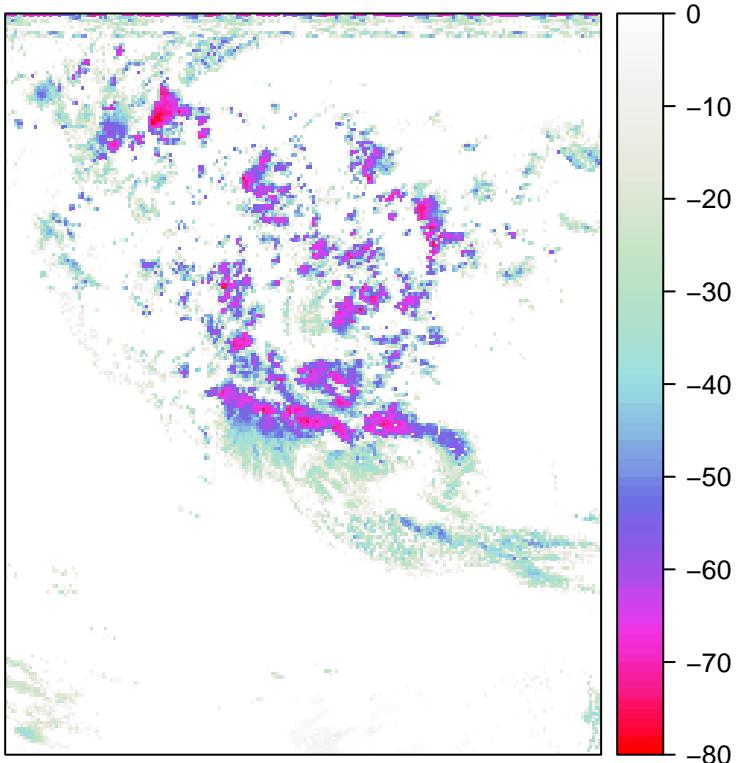
```
library(raster, quietly = TRUE)
l3 <- raster(t(12[1:1349,1:1613]),
             xmn=-82.00,
             ymn=-44.96,
             xmx=-82.0 + (0.03593245*1349),
             ymx=-44.96 + (0.03593245*1613),
             crs = CRS("+init=epsg:4326"))
class(l3)
```

```
## [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 ($^{\circ}\text{C}$)



Escrever dados binários no R

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 20:22:55
## 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)
```

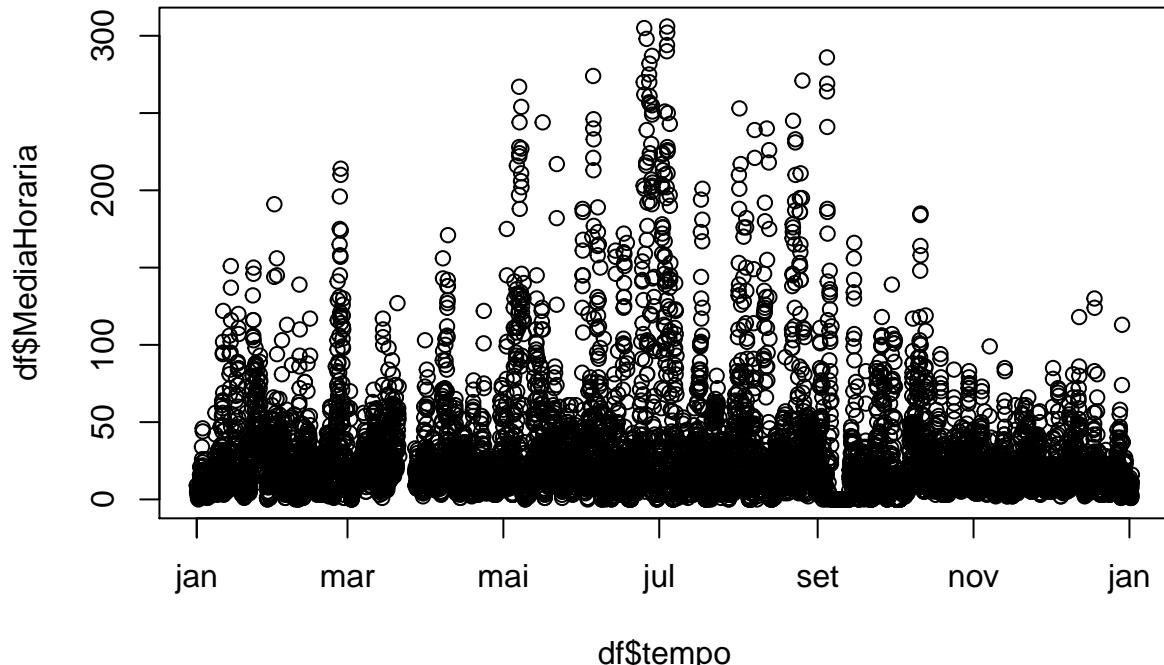
```

## 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?

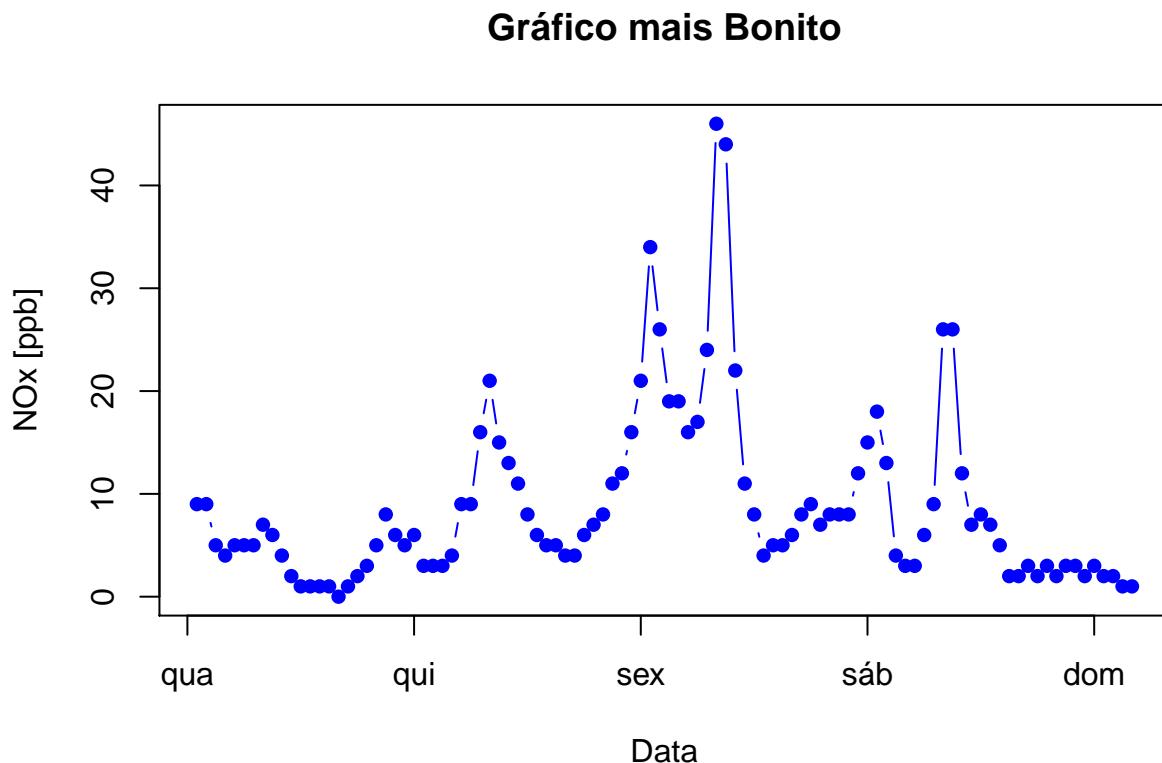
Tentando deixar mais bonito...

```

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)

```

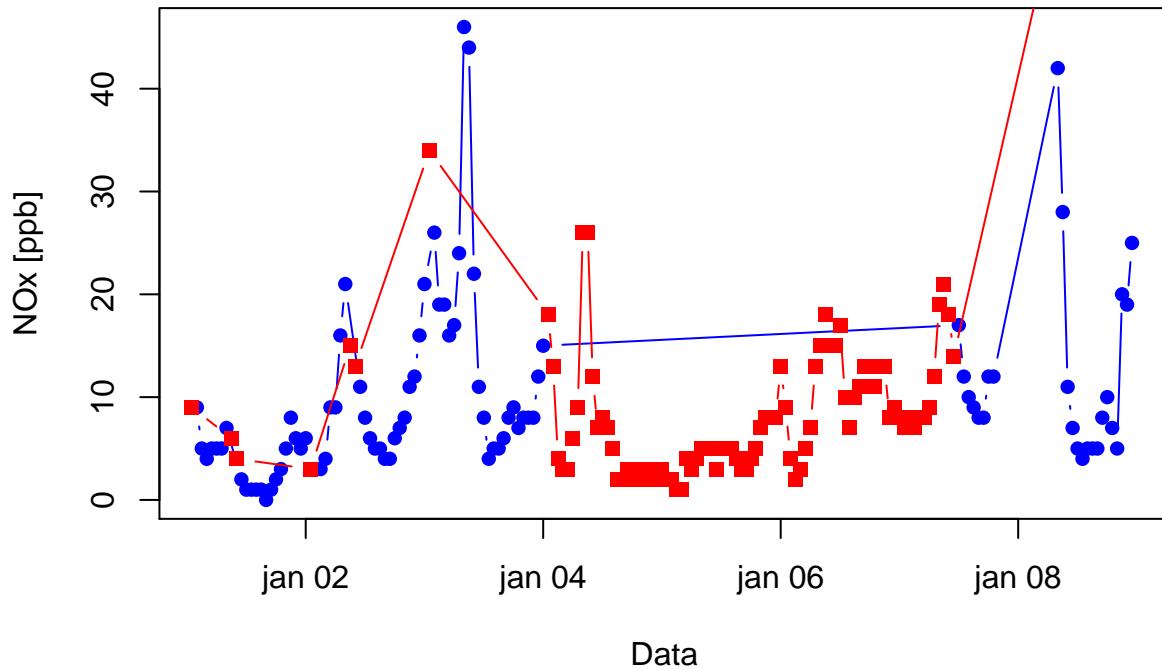
```
col = "blue", #-- Cor do elemento (definido pelo type)
xlab = "Data", ylab = "NOx [ppb]", #-- Nome dos eixos x e y
main = "Gráfico mais Bonito") #-- Título do gráfico
```



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 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).
```

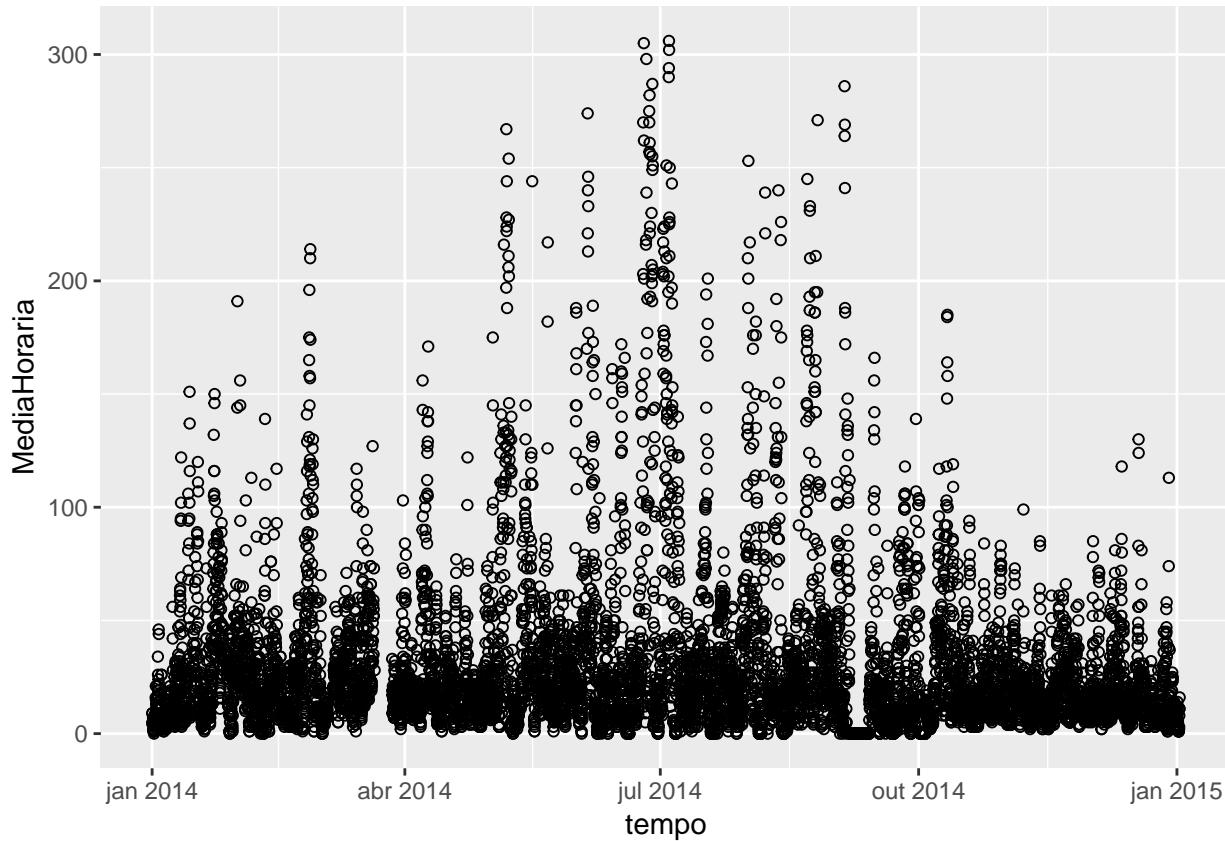
Complete the template below to build a graph.

```
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

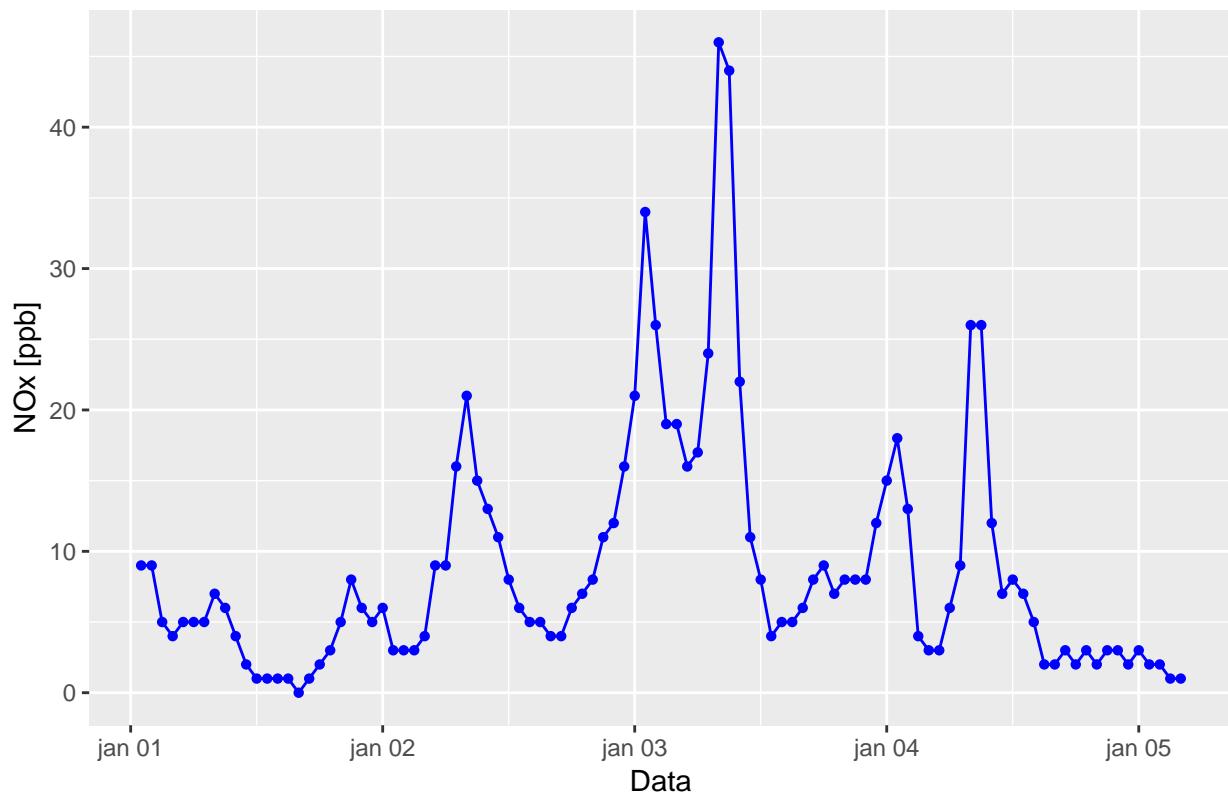
ggplot(data = mpg, aes(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

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
```

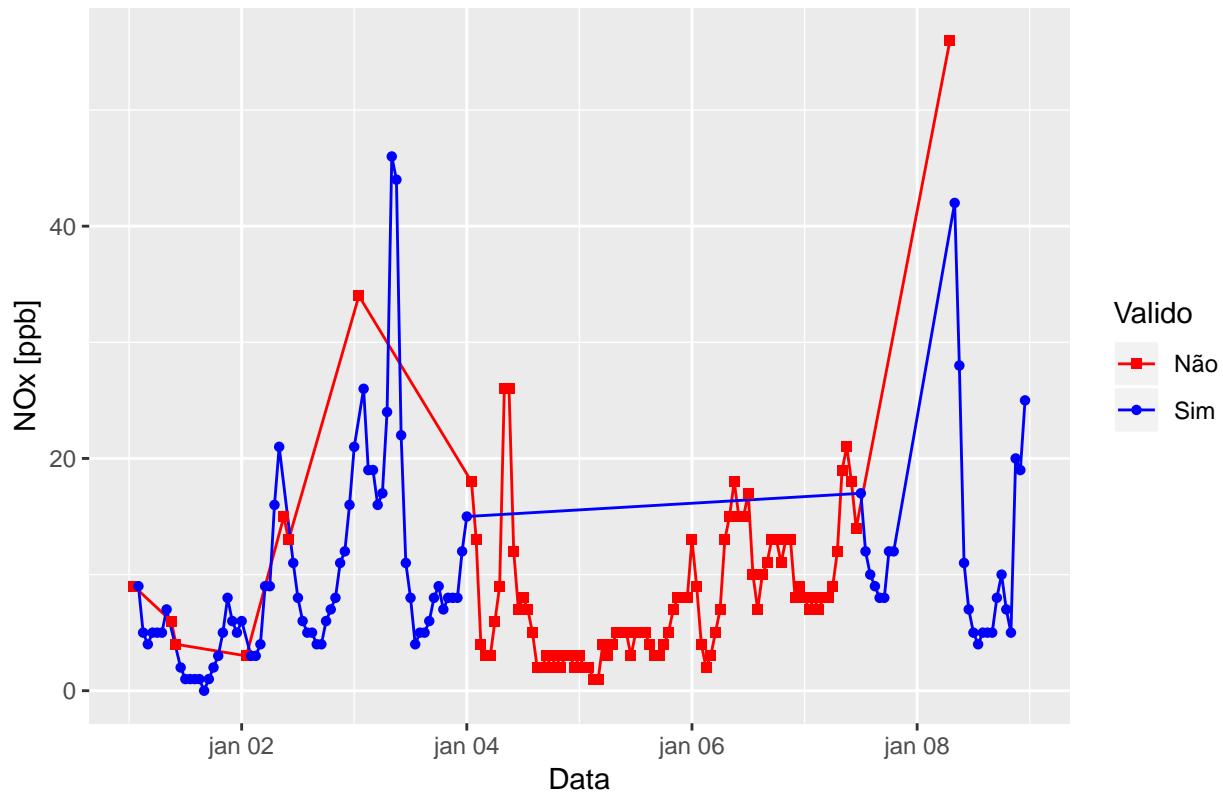
Gráfico mais Bonito



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 funções dentro do pacote `tidyverse`, que funcionam bem com o *pipe* (%>%):

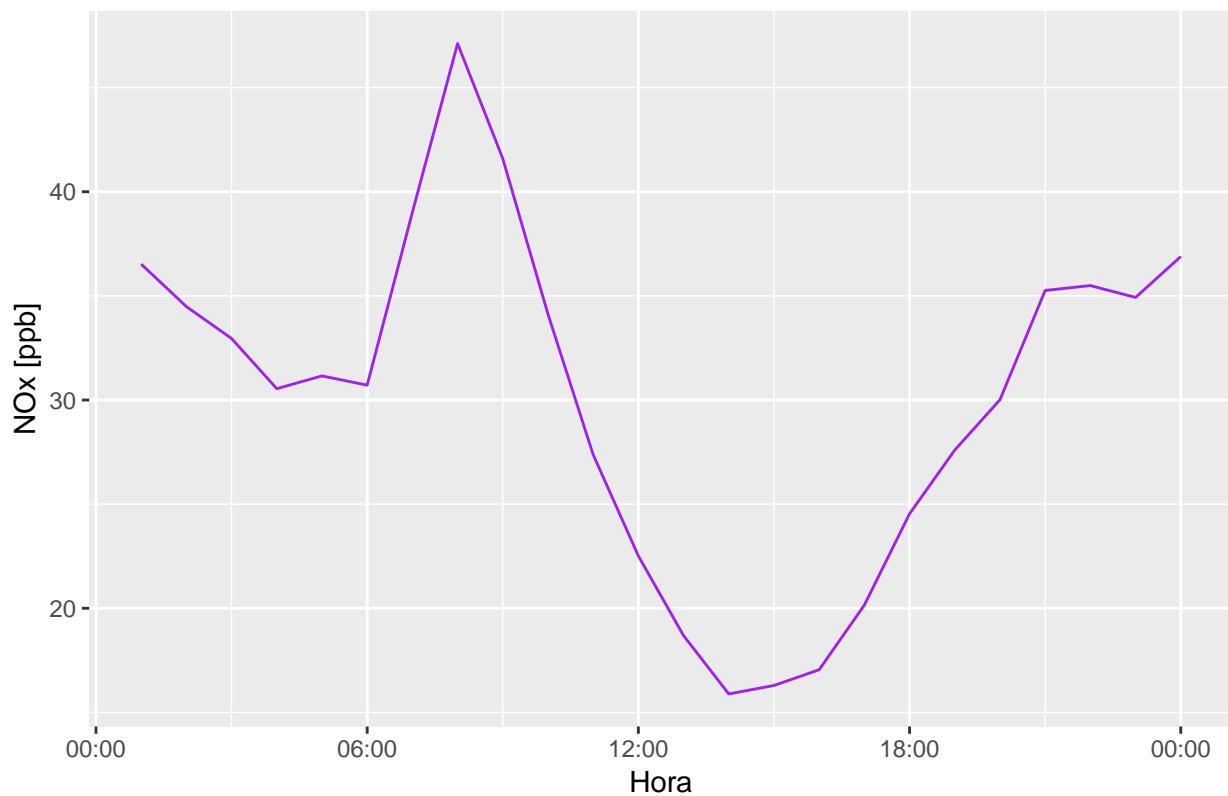
```
library(tidyverse)

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]")
```

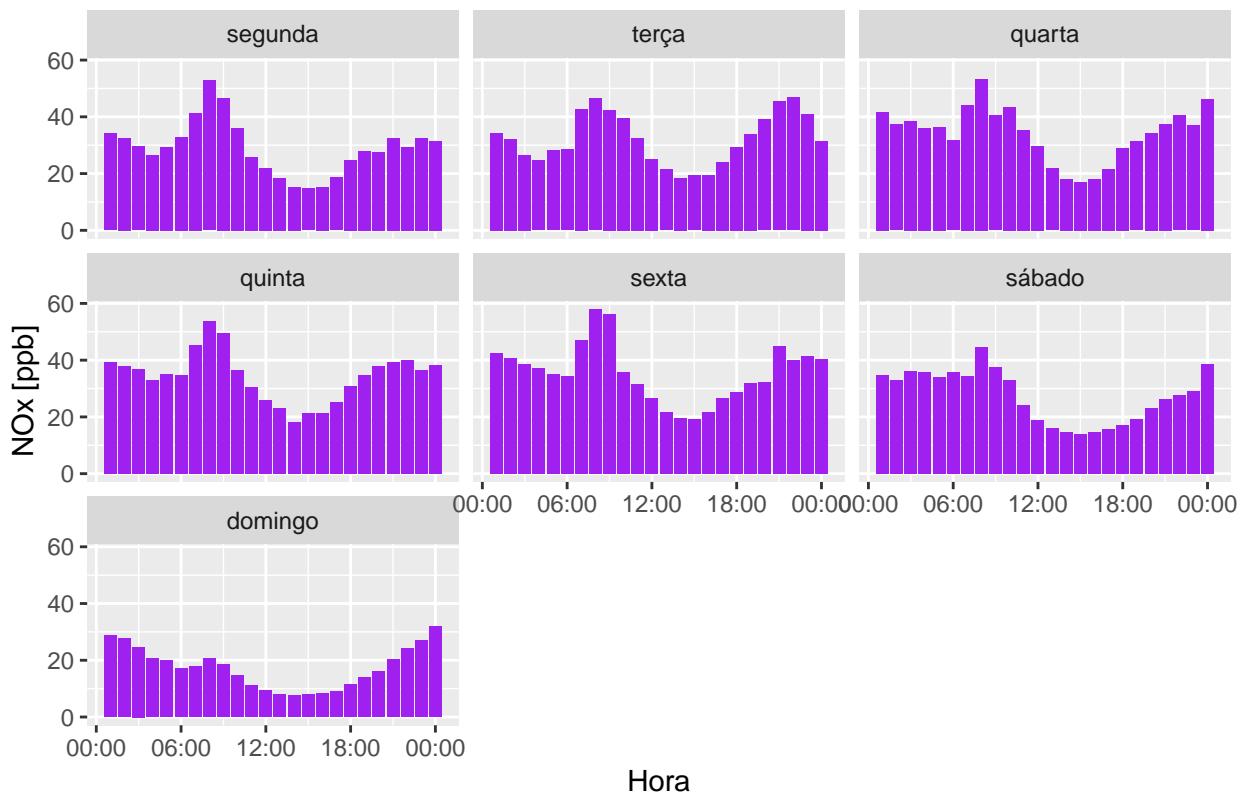
Média Horária Anual



```
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
```

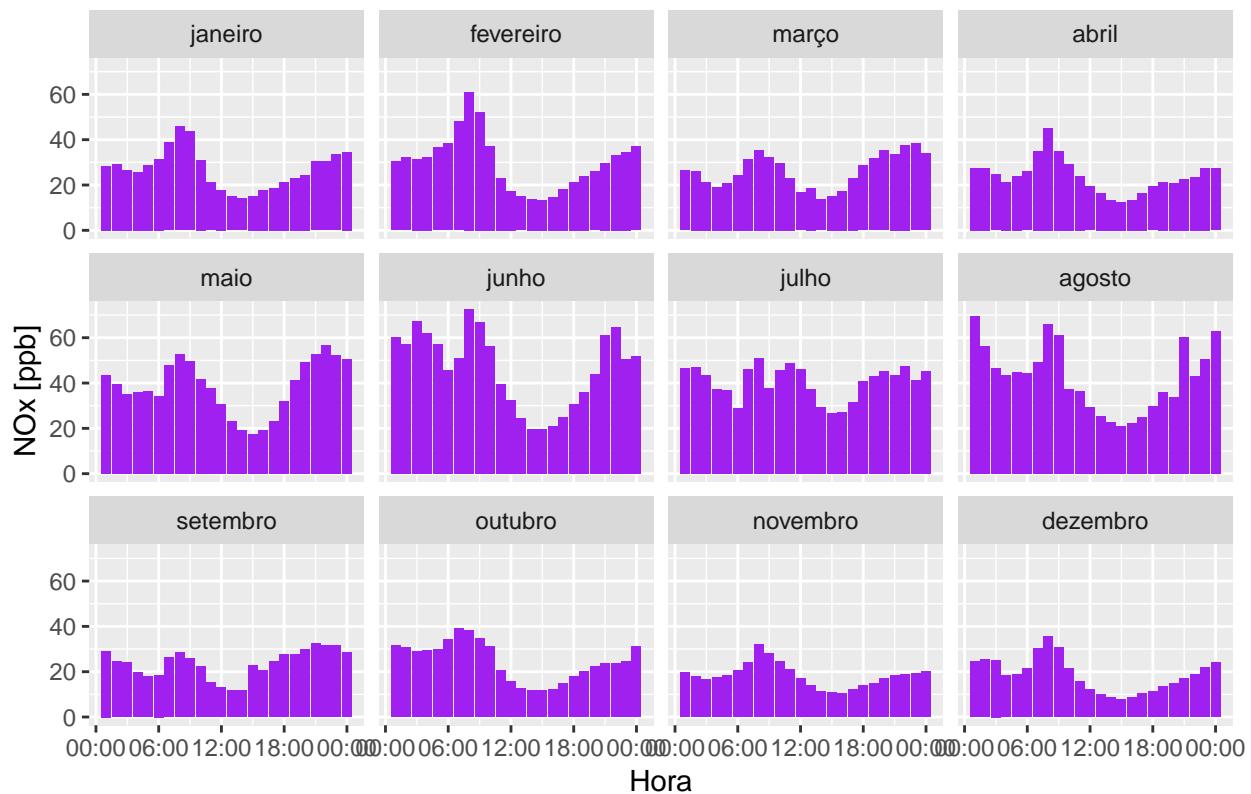
Média Horária Anual por Dia da Semana



```
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

5.2.1 Explorando outras escalas de cores e temas

Pacotes `veinreport` e `cptcity`

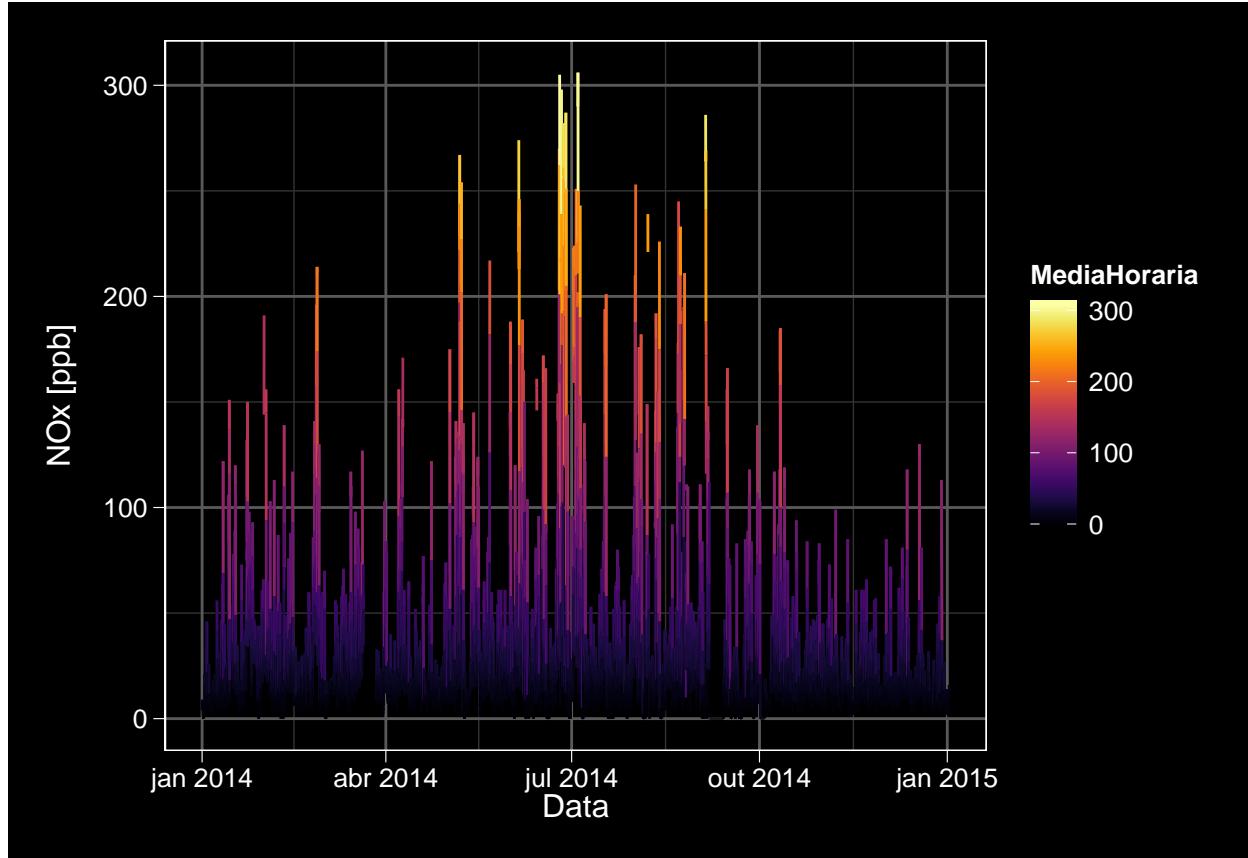
```
devtools::install_github("atmoschem/veinreport")
```

```
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()
```

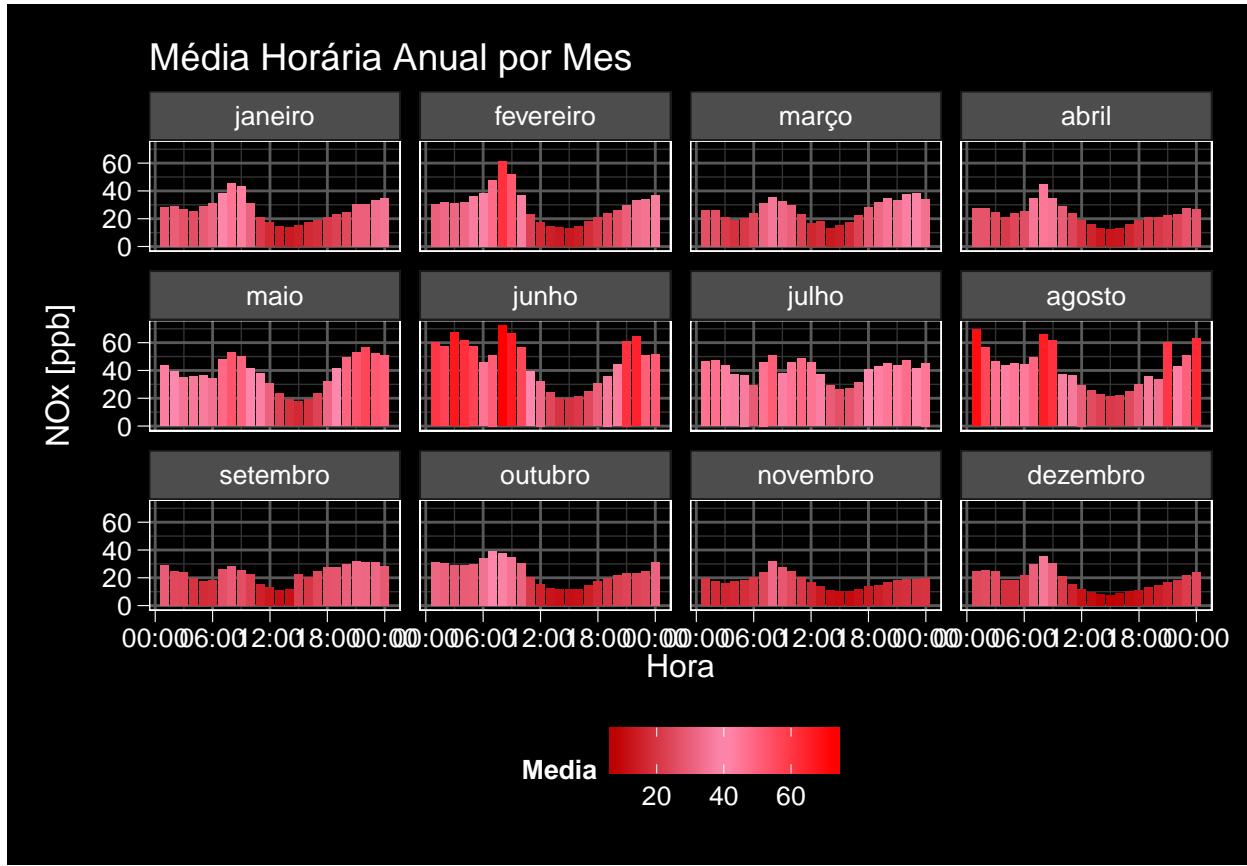
```
## Warning: Removed 1 rows containing missing values (geom_path).
```



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_red_Mono_8, number: 5597
```



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

Chapter 6

Estruturas de Controle

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)