Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD0601 - Bioestatística

Organização dos dados em R

Prof. Dr. Tetsu Sakamoto
Instituto Metrópole Digital - UFRN
Sala A224, ramal 182
Email: tetsu@imd.ufrn.br

Baixe a aula (e os arquivos)

- Para aqueles que não clonaram o repositório:

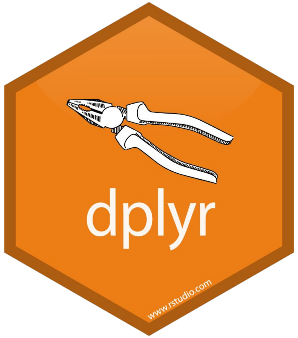
```
> git clone https://github.com/tetsufmbio/IMD0601.git
```

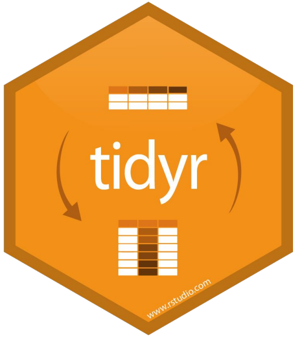
- Para aqueles que já tem o repositório local:

```
> cd /path/to/IMD0601

> git pull
```

Revisão

Entender e arrumar os dados

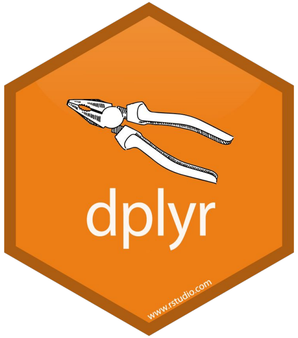




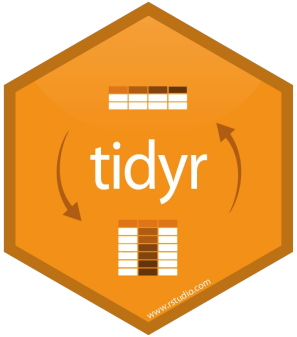
Conceito de uma tabela arrumada:

- Cada variável deve estar em uma coluna;
- Cada observação forma uma linha;
- Cada tipo de unidade observacional forma uma tabela.

Revisão



- **select**(<tbl_df>, <var1>, <var2>, ...)
- **filter**(<tbl_df>, <condition1>, <condition2>, ...)
- **arrange**(<tbl_df>, <var1>, desc(<var2>),...)
- **mutate**(<tbl_df>, <new_var1> = <value>, <new_var2> = <value>,...)
- **summarize**(<tbl_df>, <column> = <function>)



- **gather**(<tbl_df>, <col1>, <col2>, ...)
- **separate**(<tbl_df>, <col>, into = <vector>)
- **spread**(<tbl_df>, <col>, <col_value>)
- **unique**(<tbl_df>)
- **bind_rows**(<tbl_df1>,<tbl_df2>)

Revisão

Messy data:

- Valores representando colunas de uma tabela;
- Múltiplas variáveis em uma coluna;
- Variáveis armazenadas tanto em linhas quanto em colunas;
- Múltiplas unidades observacionais em uma única tabela. Dados redundantes na tabela.
- Duas tabelas separadas com a mesma unidade observacional.

https://makingnoiseandhearingthings.com/2018/04/19/datasets-for-data-cleaning-practice/

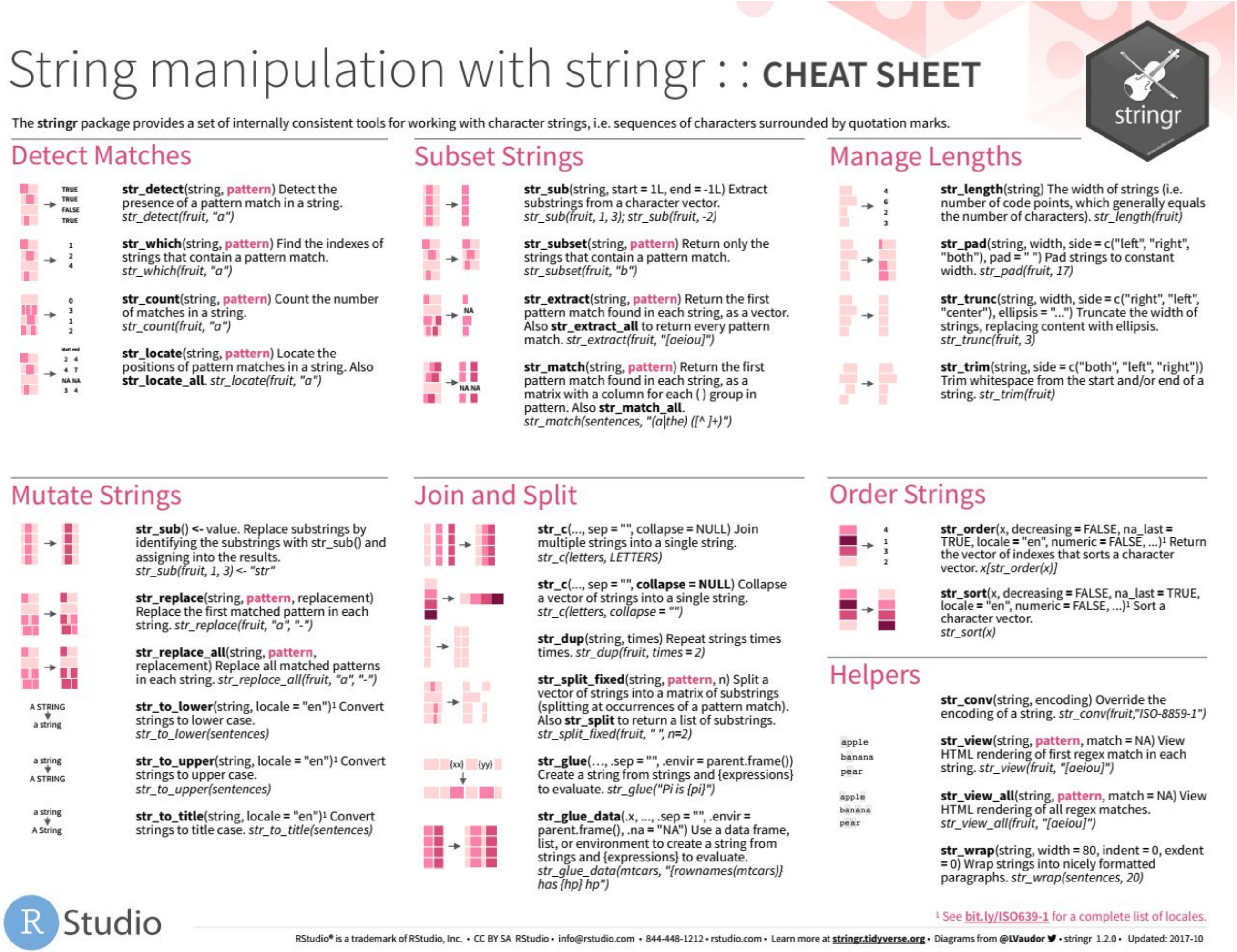
as.<class>()

```r
# logical
class(TRUE)

# character
class("8484.00")

# numeric
class(99)

# character
class("factor")

# character
class("FALSE")
```

```r
# Converter para "character"
class(as.character(TRUE))

# Converter para "numeric"
class(as.numeric("8484.00"))

# Converter para "integer"
class(as.integer(99))

# Converter para "factor"
class(as.factor("factor"))

# Converter para "logical"
class(as.logical("FALSE"))
```

stringr

```
# Pacote que lida com strings em R

str_detect(students3$dob, "1997")

str_length("TESTE") # 5

str_replace("F", "F", "Female") # Female

str_trim("  T  ") # "T"

str_c("T","E","S","T") # "TEST"

str_pad(x, width = 6, side = "left", pad = ".")
```



String manipulation with stringr : : CHEAT SHEET

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches

str_detect(string, pattern) Detect the presence of a pattern match in a string. *str_detect(fruit, "a")*

str_which(string, pattern) Find the indexes of strings that contain a pattern match. *str_which(fruit, "a")*

str_count(string, pattern) Count the number of matches in a string. *str_count(fruit, "a")*

str_locate(string, pattern) Locate the positions of pattern matches in a string. Also **str_locate_all**. *str_locate(fruit, "a")*

Subset Strings

str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector. *str_sub(fruit, 1, 3); str_sub(fruit, -2)*

str_subset(string, pattern) Return only the strings that contain a pattern match. *str_subset(fruit, "b")*

str_extract(string, pattern) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match. *str_extract(fruit, "[aeiou]")*

str_match(string, pattern) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all**. *str_match(sentences, "(a|the) ([^]+)")*

Manage Lengths

str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). *str_length(fruit)*

str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. *str_pad(fruit, 17)*

str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. *str_trunc(fruit, 3)*

str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. *str_trim(fruit)*

Mutate Strings

str_sub() <- value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. *str_sub(fruit, 1, 3) <- "str"*

str_replace(string, pattern, replacement) Replace the first matched pattern in each string. *str_replace(fruit, "a", "-")*

str_replace_all(string, pattern, replacement) Replace all matched patterns in each string. *str_replace_all(fruit, "a", "-")*

str_to_lower(string, locale = "en")[1] Convert strings to lower case. *str_to_lower(sentences)*

str_to_upper(string, locale = "en")[1] Convert strings to upper case. *str_to_upper(sentences)*

str_to_title(string, locale = "en")[1] Convert strings to title case. *str_to_title(sentences)*

Join and Split

str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string. *str_c(letters, LETTERS)*

str_c(..., sep = "", **collapse = NULL**) Collapse a vector of strings into a single string. *str_c(letters, collapse = "")*

str_dup(string, times) Repeat strings times times. *str_dup(fruit, times = 2)*

str_split_fixed(string, pattern, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings. *str_split_fixed(fruit, " ", n=2)*

str_glue(…, .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. *str_glue("Pi is {pi}")*

str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. *str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")*

Order Strings

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. *x[str_order(x)]*

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector. *str_sort(x)*

Helpers

str_conv(string, encoding) Override the encoding of a string. *str_conv(fruit,"ISO-8859-1")*

str_view(string, pattern, match = NA) View HTML rendering of first regex match in each string. *str_view(fruit, "[aeiou]")*

str_view_all(string, pattern, match = NA) View HTML rendering of all regex matches. *str_view_all(fruit, "[aeiou]")*

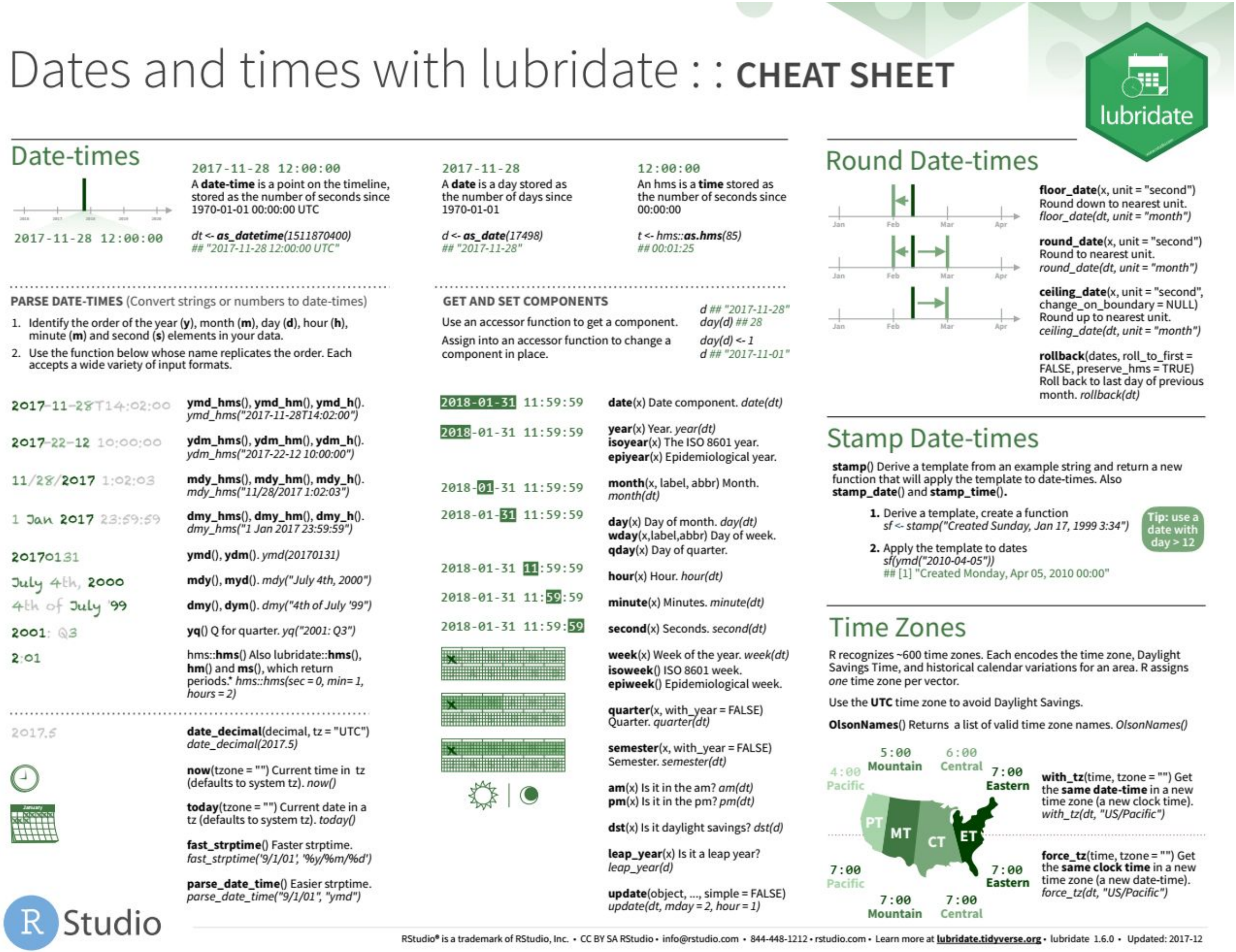
str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. *str_wrap(sentences, 20)*

[1] See bit.ly/ISO639-1 for a complete list of locales.

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **stringr.tidyverse.org** • Diagrams from **@LVaudor** • stringr 1.2.0 • Updated: 2017-10

lubridate

```
# Pacote que lida com dados temporais em R

library(lubridate)

ymd("2012, Aug 13") # "2012-08-13"

ymd_hm("2012-08-13 12:43") # "2012-08-13 12:43:00 UTC"

hms("12:43:54") # "12H 43M 54S"
```



Dates and times with lubridate : : CHEAT SHEET

Date-times

2017-11-28 12:00:00

2017-11-28 12:00:00
A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

*dt <- **as_datetime**(1511870400)*
## "2017-11-28 12:00:00 UTC"

2017-11-28
A **date** is a day stored as the number of days since 1970-01-01

*d <- **as_date**(17498)*
## "2017-11-28"

12:00:00
An hms is a **time** stored as the number of seconds since 00:00:00

*t <- hms::**as.hms**(85)*
## 00:01:25

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

2017-11-28T14:02:00	**ymd_hms**(), **ymd_hm**(), **ymd_h**(). *ymd_hms("2017-11-28T14:02:00")*
2017-22-12 10:00:00	**ydm_hms**(), **ydm_hm**(), **ydm_h**(). *ydm_hms("2017-22-12 10:00:00")*
11/28/2017 1:02:03	**mdy_hms**(), **mdy_hm**(), **mdy_h**(). *mdy_hms("11/28/2017 1:02:03")*
1 Jan 2017 23:59:59	**dmy_hms**(), **dmy_hm**(), **dmy_h**(). *dmy_hms("1 Jan 2017 23:59:59")*
20170131	**ymd**(), **ydm**(). *ymd(20170131)*
July 4th, 2000	**mdy**(), **myd**(). *mdy("July 4th, 2000")*
4th of July '99	**dmy**(), **dym**(). *dmy("4th of July '99")*
2001: Q3	**yq**() Q for quarter. *yq("2001: Q3")*
2:01	hms::**hms**() Also lubridate::**hms**(), **hm**() and **ms**(), which return periods.* *hms::hms(sec = 0, min= 1, hours = 2)*

2017.5	**date_decimal**(decimal, tz = "UTC") *date_decimal(2017.5)*
	now(tzone = "") Current time in tz (defaults to system tz). *now()*
January	**today**(tzone = "") Current date in a tz (defaults to system tz). *today()*
	fast_strptime() Faster strptime. *fast_strptime('9/1/01', '%y/%m/%d')*
	parse_date_time() Easier strptime. *parse_date_time("9/1/01", "ymd")*

GET AND SET COMPONENTS

Use an accessor function to get a component. *d ## "2017-11-28"*
day(d) ## 28

Assign into an accessor function to change a component in place. *day(d) <- 1*
d ## "2017-11-01"

2018-01-31 11:59:59	**date**(x) Date component. *date(dt)*
2018-01-31 11:59:59	**year**(x) Year. *year(dt)* **isoyear**(x) The ISO 8601 year. **epiyear**(x) Epidemiological year.
2018-01-31 11:59:59	**month**(x, label, abbr) Month. *month(dt)*
2018-01-31 11:59:59	**day**(x) Day of month. *day(dt)* **wday**(x,label,abbr) Day of week. **qday**(x) Day of quarter.
2018-01-31 11:59:59	**hour**(x) Hour. *hour(dt)*
2018-01-31 11:59:59	**minute**(x) Minutes. *minute(dt)*
2018-01-31 11:59:59	**second**(x) Seconds. *second(dt)*
	week(x) Week of the year. *week(dt)* **isoweek**() ISO 8601 week. **epiweek**() Epidemiological week.
	quarter(x, with_year = FALSE) Quarter. *quarter(dt)*
	semester(x, with_year = FALSE) Semester. *semester(dt)*
☼ | ◉	**am**(x) Is it in the am? *am(dt)* **pm**(x) Is it in the pm? *pm(dt)*
	dst(x) Is it daylight savings? *dst(d)*
	leap_year(x) Is it a leap year? *leap_year(d)*
	update(object, ..., simple = FALSE) *update(dt, mday = 2, hour = 1)*

Round Date-times

floor_date(x, unit = "second")
Round down to nearest unit.
floor_date(dt, unit = "month")

round_date(x, unit = "second")
Round to nearest unit.
round_date(dt, unit = "month")

ceiling_date(x, unit = "second", change_on_boundary = NULL)
Round up to nearest unit.
ceiling_date(dt, unit = "month")

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE)
Roll back to last day of previous month. *rollback(dt)*

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date**() and **stamp_time**().

1. Derive a template, create a function
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
2. Apply the template to dates
   sf(ymd("2010-04-05"))
   ## [1] "Created Monday, Apr 05, 2010 00:00"

Tip: use a date with day > 12

Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns *one* time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. *OlsonNames()*

with_tz(time, tzone = "") Get the **same date-time** in a new time zone (a new clock time). *with_tz(dt, "US/Pacific")*

force_tz(time, tzone = "") Get the **same clock time** in a new time zone (a new date-time). *force_tz(dt, "US/Pacific")*

Exercício

Carregue o arquivo data.csv no R(studio). Este arquivo contém dados climatológicos de algumas localidades no Brasil.

Ao explorar os dados, você verificará que os dados estão bagunçados. Utilizando os recursos visto nesta e na aula passada, tente organizar estes dados.

Descrição sobre as colunas pode ser acessada em:
https://www.kaggle.com/PROPPG-PPG/hourly-weather-surface-brazil-southeast-region

Exercício

1. Explore os dados;
2. Divida a tabela de forma que cada tabela tenha dados de uma unidade observacional;
3. Retire colunas redundantes na tabela que contém os dados climatológicos;
4. Formate a coluna que contém a data e a hora utilizando o pacote lubridate;
5. Explore a coluna prcp. O que representa o valor 0 e o NA nesta coluna?
6. Explore a coluna stp. O que representa o valor 0 e o NA nesta coluna?
7. Corrija os valores das colunas prcp, stp, smax e smin em relação aos dados faltantes.

Referência

Esta aula foi baseada no curso “**Cleaning Data in R**” de Nick Carchedi (https://www.datacamp.com/courses/cleaning-data-in-r)