



26 April 2018

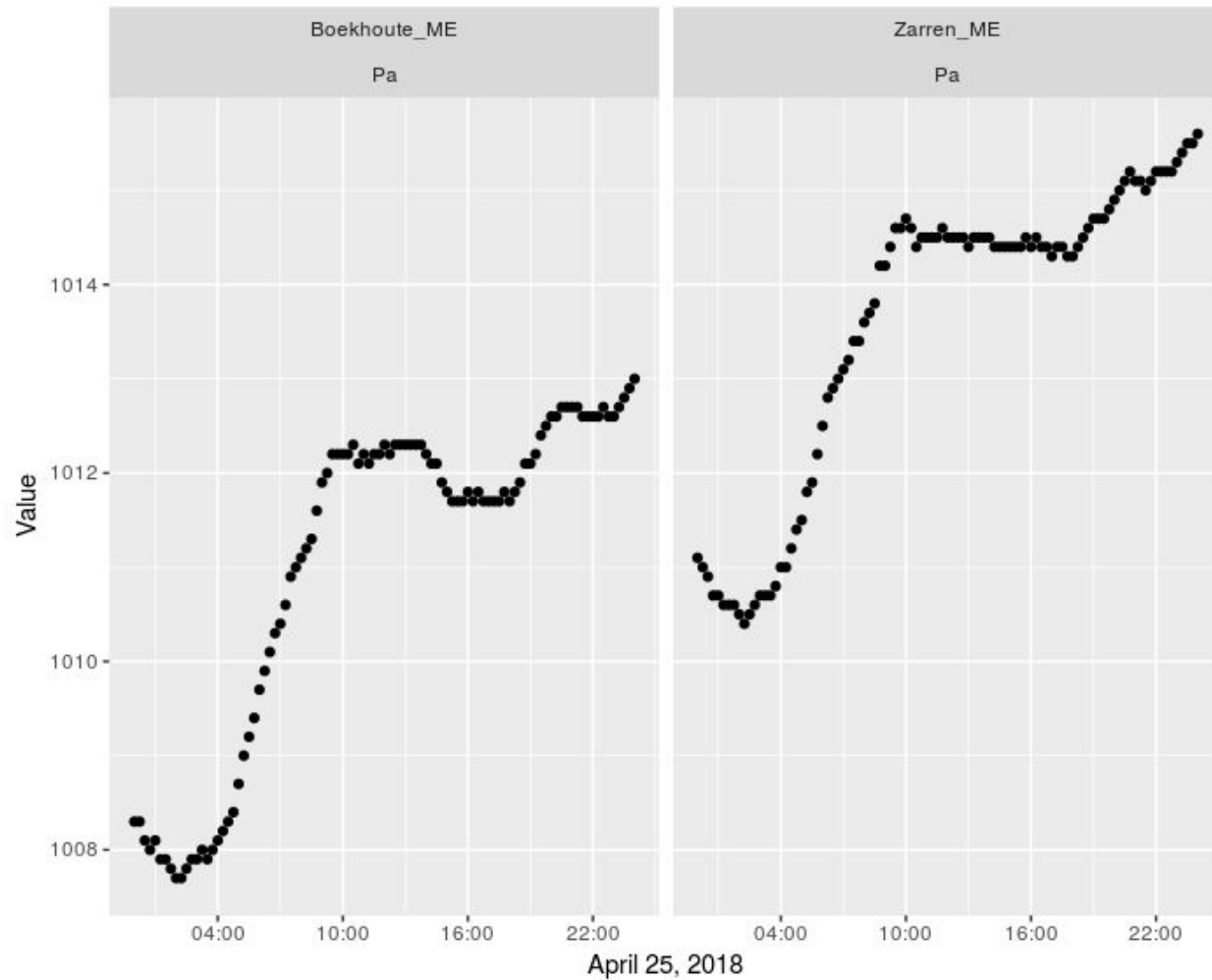
Herman Teirlinck,
01.21 - Jeanne Brabants

What have I done?!?

```
stations <- get_stations("air_pressure") %>%
  filter(stringr::str_detect(station_no, "03"))

air_pressure <- stations %>%
  group_by(ts_id) %>%
  do(get_timeseries_tsid(.$ts_id, period = "P1D",
                        to = lubridate::today())) %>%
  ungroup() %>%
  left_join(stations, by = "ts_id")

air_pressure %>%
  ggplot(aes(x = Timestamp, y = Value)) +
  geom_point() + xlab(format(lubridate::today() - 1, format="%B %d %Y")) +
  facet_wrap(c("station_name", "stationparameter_name")) +
  scale_x_datetime(date_labels = "%H:%M",
                   date_breaks = "6 hours")
```



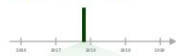


DATES
&
STRINGS

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"
```

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

- Identify the order of the year (y), month (m), day (d), hour (h), minute (m) and second (s) elements in your data.
- 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()
```

```
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")
```



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"
```

GET AND SET COMPONENTS

Use an accessor function to get a component. Assign into an accessor function to change a component in place.

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
```

```
d## "2017-11-28"
day(d) ## 28
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)
```

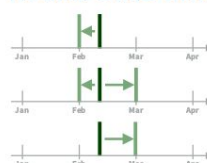
```
am(x) Is it in the am? am(dt)
pm(x) Is it in the pm? pm(dt)
```

```
dst(x) Is it daylight savings? dst(dt)
```

```
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()**.

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

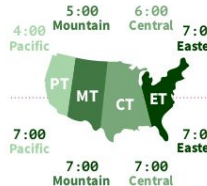
Tip: use a date with day > 12

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")*


Work with strings 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

 → `TRUE`
`TRUE`
`FALSE`
`TRUE`

 → 1
2
4

 → 0
3
1
2

 → 2 4
4 7
NA NA
3 4


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")`


 → NA


 → NA NA


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 (1) group in pattern. Also **str_match_all**. `str_match(sentences, "(a)the ([^]+)*")`

Manage Lengths


 → 4
6
2
3


 → `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", ".")`

 → A STRING
▼
a string

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


 → a string
▼
A STRING


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


 → a string
▼
A string


`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)`


 → (xx) (yy)
↓
`glue`:
`glue`(..., sep = "", envir = parent.frame(), open = "{", close = "}") Create a string from strings and (expressions) to evaluate. `glue:"glue"("Pis ipi")`

 → `glue`:
`glue_data`(x, ..., sep = "", envir = parent.frame(), open = "{", close = "}") Use a data frame, list, or environment to create a string from strings and (expressions) to evaluate. `glue:glue_data(mtcars, "(rownames(mtcars)) has {hp} hp")`


Order Strings


 → 4
1
3
2

 → `str_order`(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...) Sort a character vector. `str_order(x)`


 → `str_sort`(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...) Sort a character vector. `str_sort(x)`

Helpers


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

 → apple
banana
pear

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

 → apple
banana
pear

`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, expand = 0) Wrap strings into nicely formatted paragraphs. `str_wrap(sentences, 20)`

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

Install the package suite:

```
install.packages("tidyverse")  
install.packages("lubridate")
```

Load the package suite:

```
library(tidyverse)  
library(lubridate)
```

Share your snippets during the coding session!

Go to <https://hackmd.io/aPEFORMXSI0eEycsDsSTqw> and post your code in between backticks:

For example:

```
```\n\nlibrary(lubridate)\n\nmy_data <- ... \n\n```
```



# recap/showcase

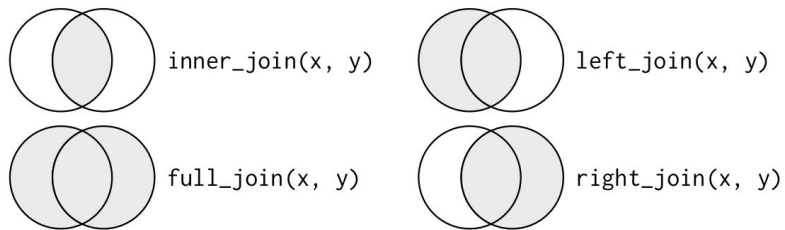
Read in the [20180222\\_surveys.csv](#) and the [20180222\\_species.csv](#) data.

The image shows a collage of cheat sheets for dplyr functions. The main sections are:

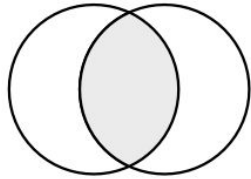
- Summarise Data:** Functions like summarise(), summarise\_each(), count(), and summarise\_if().
- Make New Variables:** Functions like mutate(), mutate\_each(), window(), lead(), lag(), and dense\_rank().
- Group Data:** Functions like group\_by(), ungroup(), and summarise().
- Joining Data:** Functions like inner\_join(), left\_join(), right\_join(), full\_join(), and anti\_join().

Join the species information columns (genus, species, taxa) to the survey data set, using the common identifier.

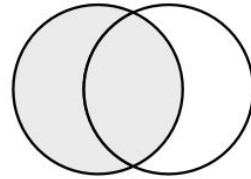
Compare the result when applying the different commands to join the data...



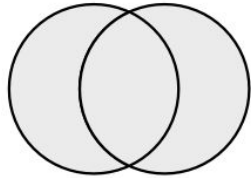
# recap/showcase



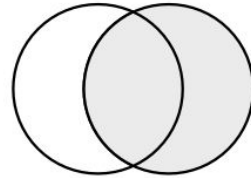
`inner_join(x, y)`



`left_join(x, y)`



`full_join(x, y)`




`right_join(x, y)`

`semi_join`

`anti_join`

# The concept

We defined a number of challenges. If you were able to achieve a challenge, add a  to your laptop screen.

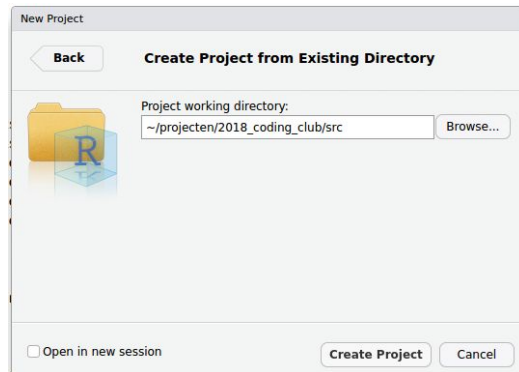
The objective is that **everyone** achieves !

- Someone has more  than you? **Ask for help!**
- Someone has less  than you? **Provide help!**

- Download coding club material and work locally, not in sync with the Google drive



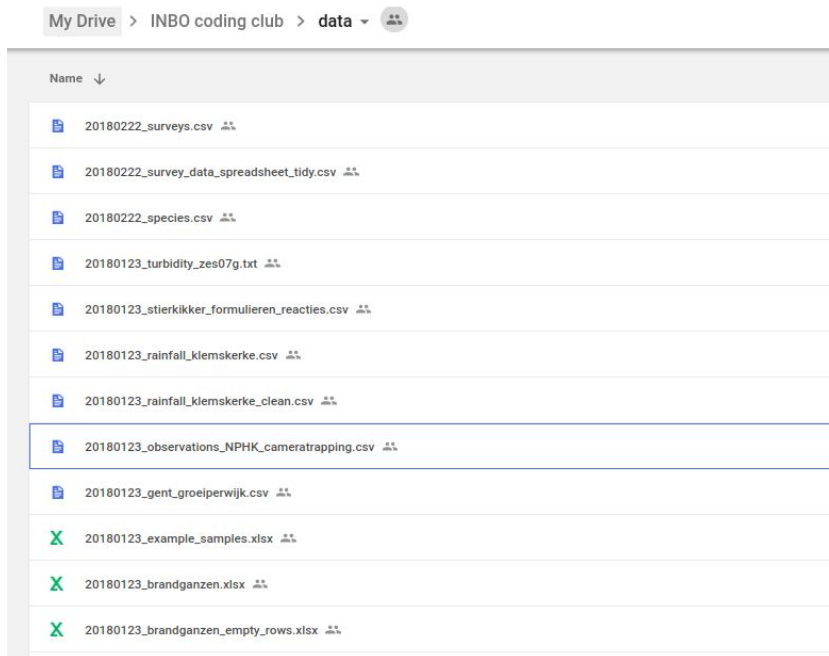
- Create new Rstudio project in the **/src** folder



- Download coding club material and work locally, not in sync with the Google drive
- Create new Rstudio project in the **src** folder...
- Use relative paths to data files:

```
> library(readr)
```

```
> read_csv2("../data/20180123_gent_groeiperwijk.csv")
```



part 1

```
my_date <- "August 2nd, 2018 14:00"
```

Which day of the week (monday, tuesday,...)  
is "August 2nd, 2018 14:00"?

## Dates and times with lubridate : : CHEAT SHEET



### Date-times

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(211870400)`  
#> "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(7490)`  
#> "2017-11-28"

12:00:00

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

`h = hms(as_hms(5))`  
#> "00:00:05"

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

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

2017-11-28 14:02:00 `ymd_hms()`, `ymd_hms()`, `ymd_h_m_s()`  
2017-22-12-10:00:00 `ymd_hms()`, `ymd_hms()`, `ymd_h_m_s()`  
11/28/2017 10:02:03 `mdy_hms()`, `mdy_hms()`, `mdy_h_m_s()`  
1 Jan 2017 23:59:59 `dmy_hms()`, `dmy_hms()`, `dmy_h_m_s()`  
20170131 `ymd()`, `ymd()`, `ymd(20170131)`  
July 4th, 2000 `mdy()`, `mdy()`, `mdy("July 4th, 2000")`  
4th of July 99 `dmy()`, `dmy()`, `dmy("4th of July '99")`  
2001\_Q3 `yq()`, `Q3` for quarter, `yq("2001: Q3")`  
2 01 `hm()`, `hm()` Also `lubridate::hm()`, `hm()` and `mm()`, which return periods: `dtm::hm(spec = "H", min = 1, hour = 2)`

**GET AND SET COMPONENTS**

Use an accessor function to get a component. Assign into an accessor function to change a component in place.

2018-01-31 11:59:59 `date()` Date component, `date(dt)`  
2018-01-31 11:59:59 `year()` Year, `year(dt)`  
`isoyear()` The ISO 8601 year, `isoyear(dt)` Epidemiological year.  
2018-01-31 11:59:59 `month()`, `label_abbr()` Month, `month(dt)`  
2018-01-31 11:59:59 `day()` Day of month, `day(dt)`  
`weekday_label_abbrev()` Day of week, `qday()` Day of quarter.  
2018-01-31 11:59:59 `hour()` Hour, `hour(dt)`  
2018-01-31 11:59:59 `minute()` Minutes, `minute(dt)`  
2018-01-31 11:59:59 `second()` Seconds, `second(dt)`  
`week()` Week of the year, `week(dt)`  
`isoweek()` ISO 8601 week, `isoweek(dt)` Epidemiological week.  
`quarter()`, `with_year = FALSE`  
`semester()`, `with_year = FALSE`  
`semester()`, `with_year = FALSE`  
`am()` Is it in the am? `am(dt)`  
`pm()` Is it in the pm? `pm(dt)`  
`dst()` Is it daylight savings? `dst(dt)`  
`leap_year()` Is it a leap year?  
`leap_year(dt)`  
`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(x, unit = "minute")  
Round up to nearest unit.  
round\_date(x, unit = "month")  
Round to nearest unit.  
ceiling\_date(x, unit = "second")  
change\_on\_boundary = NULL  
Round up to nearest unit.  
ceiling\_date(x, unit = "month")  
rollback\_date(x, to = time = FALSE, preserve\_hms = TRUE)  
Roll back to last day of previous month, rollback().

**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_datetime()`.

- Derive a template, create a function  
`f = stamp("Created Sunday, Jan 7, 1995 8:34")` (Use `stamp_date` with `date()`)
- Apply the template to dates  
`f(mdy("2008-04-05"))`  
#> [1] "Created Monday, Apr 05, 2008 00:00"

**Time Zones**

tz\_names() tz names. Each records the time zone, Daylight Savings Time, and historical calendar variations for an area. It assigns one time zone per vector.  
Use the UTC time zone to avoid Daylight Savings.  
olson\_names() Returns a list of valid time zone names. `OlsonNames()`

5:00 Mountain 6:00 Central 7:00 Eastern  
7:00 Mountain 7:00 Central 7:00 Eastern  
7:00 Mountain 7:00 Central 7:00 Eastern

force\_timezone(zone = "TZ") Get the same date-time in a new time zone (a new clock time), with zone, "TZ" (Pawlik)  
force\_timezone(zone = "TZ") Get the same date-time in a new time zone (a new date-time), force\_timezone("TZ", Pacific)





## PUBLIC SERVICE ANNOUNCEMENT:

OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT.

THIS IS *THE* CORRECT WAY TO WRITE NUMERIC DATES:

2013-02-27


THE FOLLOWING FORMATS ARE THEREFORE DISCOURAGED:

02/27/2013 02/27/13 27/02/2013 27/02/13

20130227 2013.02.27 27.02.13 27-02-13

27.2.13 2013. II. 27.  $27\frac{1}{2}$ -13 2013.158904109

MMXIII-II-XXVII MMXIII  $\frac{\text{LVII}}{\text{CCCLXV}}$  1330300800

$((3+3) \times (111+1) - 1) \times 3 / 3 - 1 / 3^3$  ~~2013~~ 

10/11011/1101 02/27/20/13  $\begin{matrix} 2 & 3 & 1 & 4 \\ 0 & 1 & 2 & 3 & 7 \\ 5 & 6 & 7 & 8 & \end{matrix}$

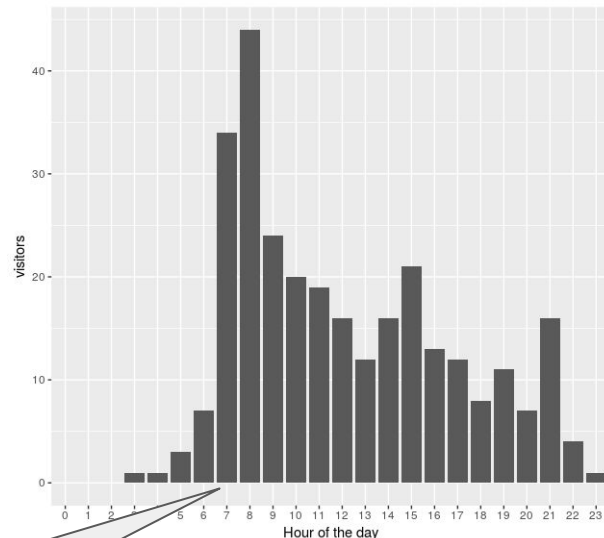






Read in the data set [20180316\\_grofwild\\_logs.csv](#), create a table that provides the number of visits (`n``) for **each(!)** hour of the day.

```
A tibble: 24 x 2
 hour n
 <int> <dbl>
1 0 0
2 1 0
3 2 0
4 3 1.00
5 4 1.00
6 5 3.00
7 6 7.00
8 7 34.0
...
```



Bonus points for a bar plot ;-)



[vragenlijst coding club!](#)



Zaal: Herman Teirlinck - 01.05 - Isala Van Diest

Datum: 22/05/2018, van 10:00 tot 12:00

*(registratie aangekondigd via [DG\\_useR@inbo.be](mailto:DG_useR@inbo.be))*