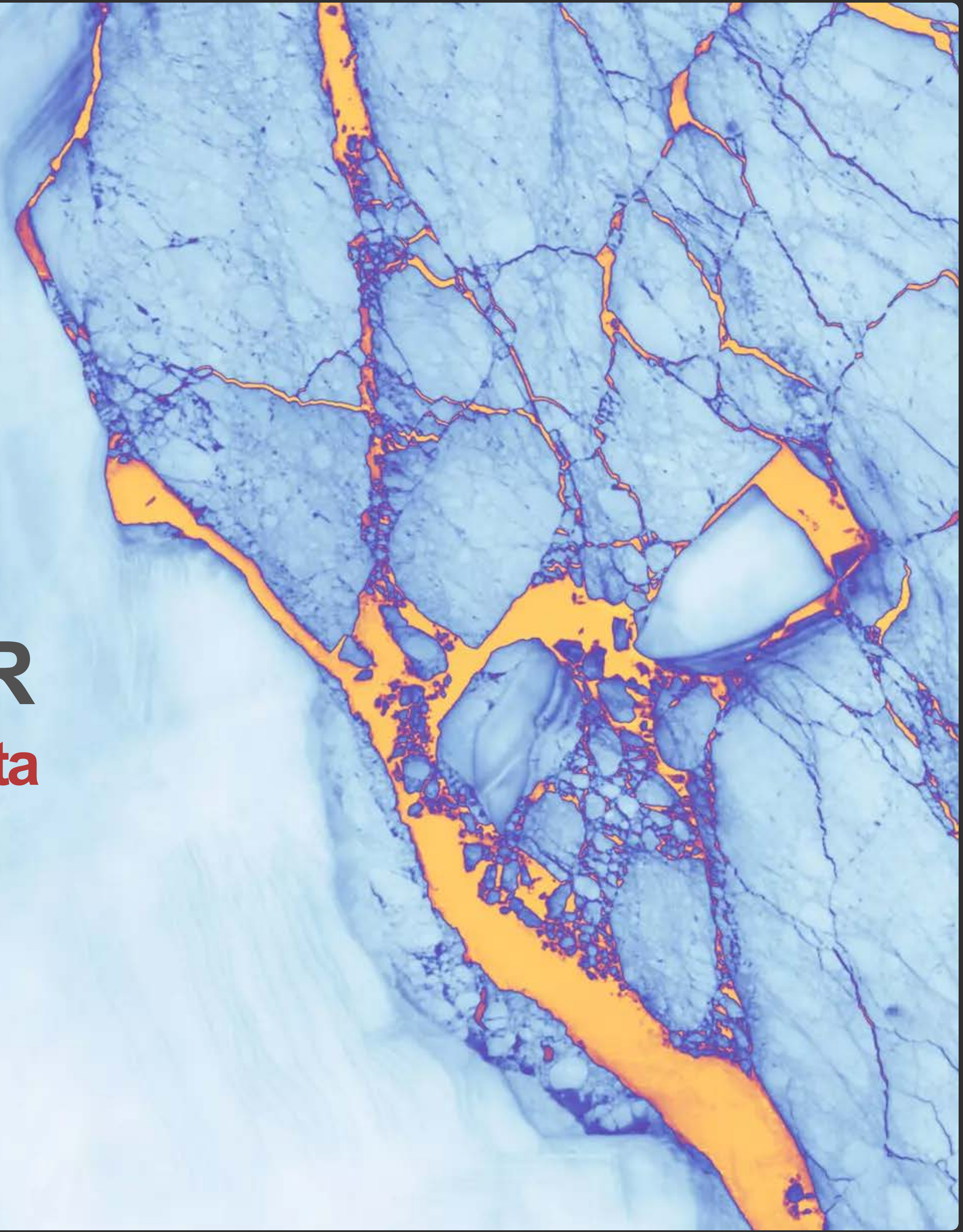
Marine Data Science

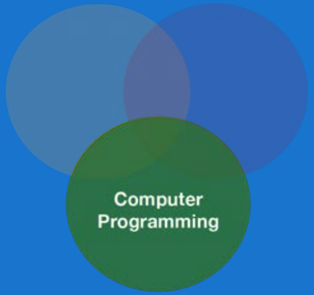
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

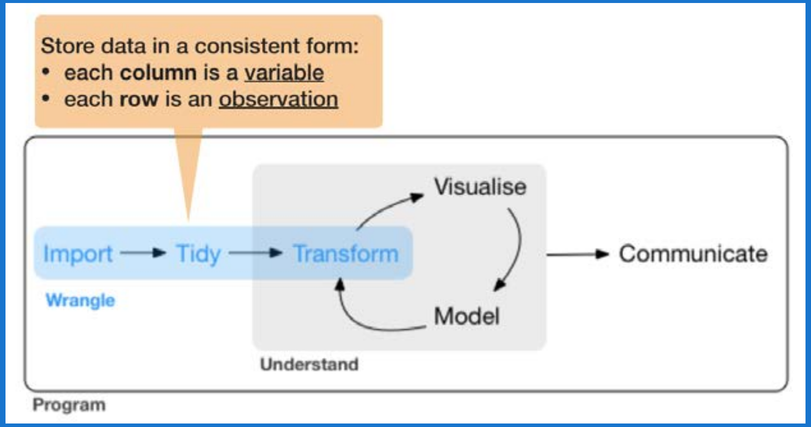


Data Analysis with R

6 - Data wrangling - 2.Tidy Data

Saskia A. Otto
Postdoctoral Researcher


Computer
Programming


Store data in a consistent form:
• each column is a variable
• each row is an observation
Import
Tidy
Transform
Wrangle
Visualise
Model
Understand
Communicate
Program

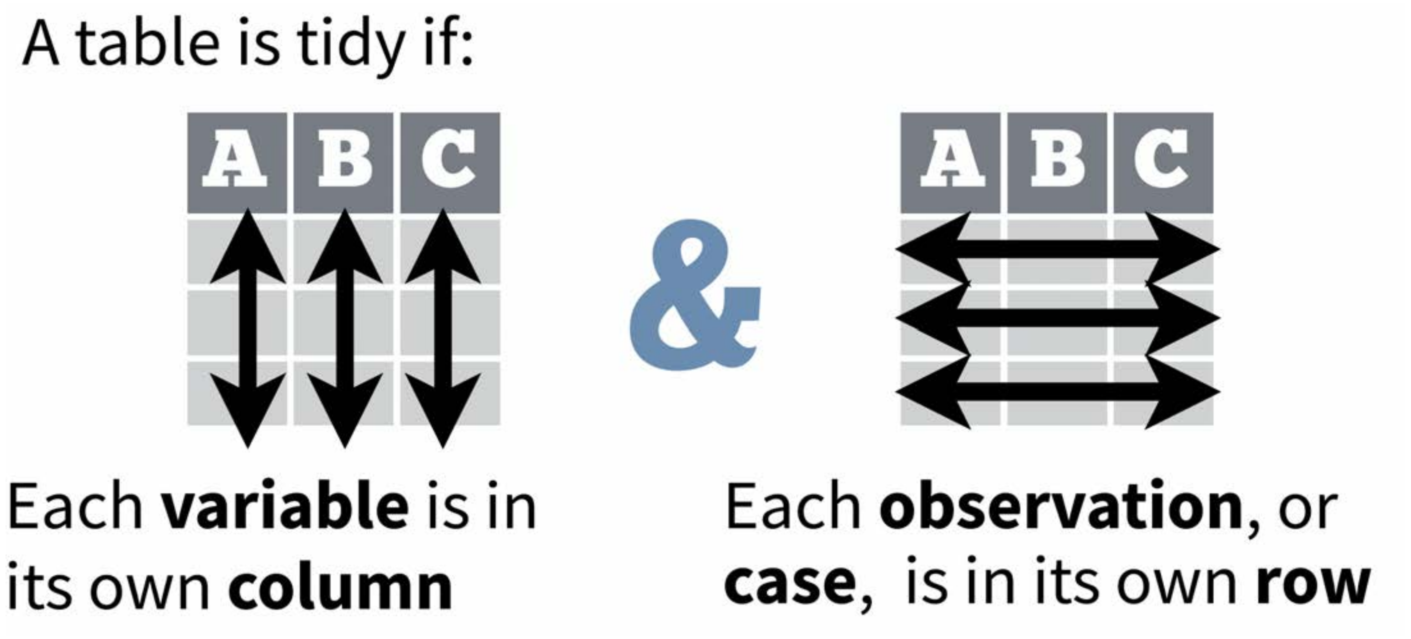
2. Tidy data

So what is "tidy" data?

- A way to **organize tabular** data

So what is "tidy" data?

- A way to **organize tabular** data


A table is tidy if:
A
B
C
&
A
B
C
Each **variable** is in its own **column**
Each **observation**, or **case**, is in its own **row**

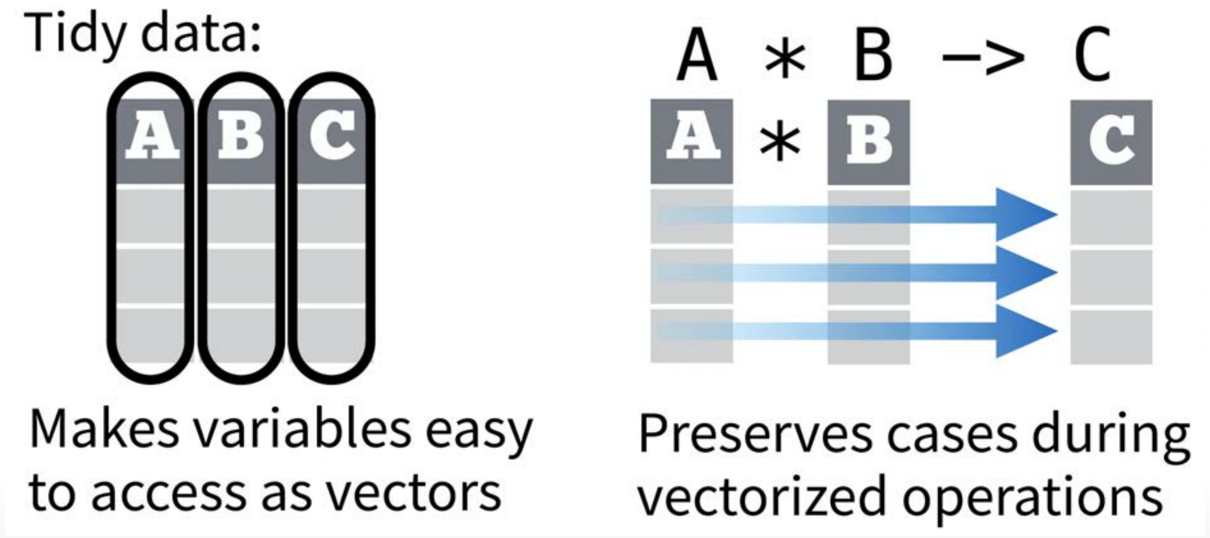
source: Data Import cheat sheet (licensed under CC-BY-SA)

So what is "tidy" data?

- A way to **organize tabular** data
- Provides a **consistent** data structure across packages.
- Is **easy** to aggregate, visualise and model (i.e. works well with dplyr, ggplot, and lm)
- Complements R's **vectorized operations** --> R will automatically preserve observations as you manipulate variables.

So what is "tidy" data?

- A way to **organize tabular** data
- Provides a **consistent** data structure across packages.
- Is **easy** to aggregate, visualise and model (i.e. works well with dplyr, ggplot, and lm)
- Complements R's **vectorized operations** --> R will automatically preserve observations as you manipulate variables.


Tidy data:
A B C
Makes variables easy to access as vectors
A * B -> C
A * B C
Preserves cases during vectorized operations

source: Data Import cheat sheet (licensed under CC-BY-SA)

So what is "tidy" data?

- A way to **organize tabular** data
- Provides a **consistent** data structure across packages.
- Is **easy** to aggregate, visualise and model (i.e. works well with dplyr, ggplot, and lm)
- Complements R’s **vectorized operations** --> R will automatically preserve observations as you manipulate variables.
- Something than many people have intuitively applied
- But Hadley Wickham developed a **whole framework** around this concept and with the tidverse packages, and particularly **'tidyr'**, it is much easier to adopt

Imagine the following crosstabulation

This table shows the number of times the 3 fish species were caught in 5 trawls:

species	present	absent
cod	1	4
herring	3	2
sprat	5	0

How many variables do you see?

Imagine the following crosstabulation

This table shows the number of times the 3 fish species were caught in 5 trawls:

species	present	absent
cod	1	4
herring	3	2
sprat	5	0

How many variables do you see?

- You should see **3**:
  - the **species**
  - the **occurence** category with 2 levels
  - the actual **values** of these 2 levels

Accessing elements in crosstabulations

species	present	absent
cod	1	4
herring	3	2
sprat	5	0

How would you access elements, e.g., to

- get the occurence categories?
- get the species list?
- get all values in one vector?

Accessing elements in crosstabulations

species	present	absent
cod	1	4
herring	3	2
sprat	5	0

How would you access elements, e.g., to

- get the occurence categories?
- get the species list?
- get all values in one vector?

```
names(df)[2:3]
df[1, 2:4]
c( df[1, ], df[2, ], df[3, ])
```

SIMPLE?

Alternatively: Restructure the table

species	present	absent
cod	1	4
herring	3	2
sprat	5	0

WIDE format

species	occurrence	n
cod	present	1
herring	present	3
sprat	present	5
cod	absent	4
herring	absent	2
sprat	absent	0

LONG format

```
names(df)[2:3]
df[1, 2:4]
c( df[1, ], df[2, ], df[3, ])
```

```
df$occurrences
df$species
df$n
```

Which one do you prefer?

Another example with the hydrographical data from ICES

What can be potential issues with such a table?

This is a table from one cruise containing the temperature values.

pressure	station_1	station_2
1	14.1	15.3
2	14.0	15.4
3	14.0	15.2
4	13.9	15.0
5	13.8	14.9

Another example with the hydrographical data from ICES

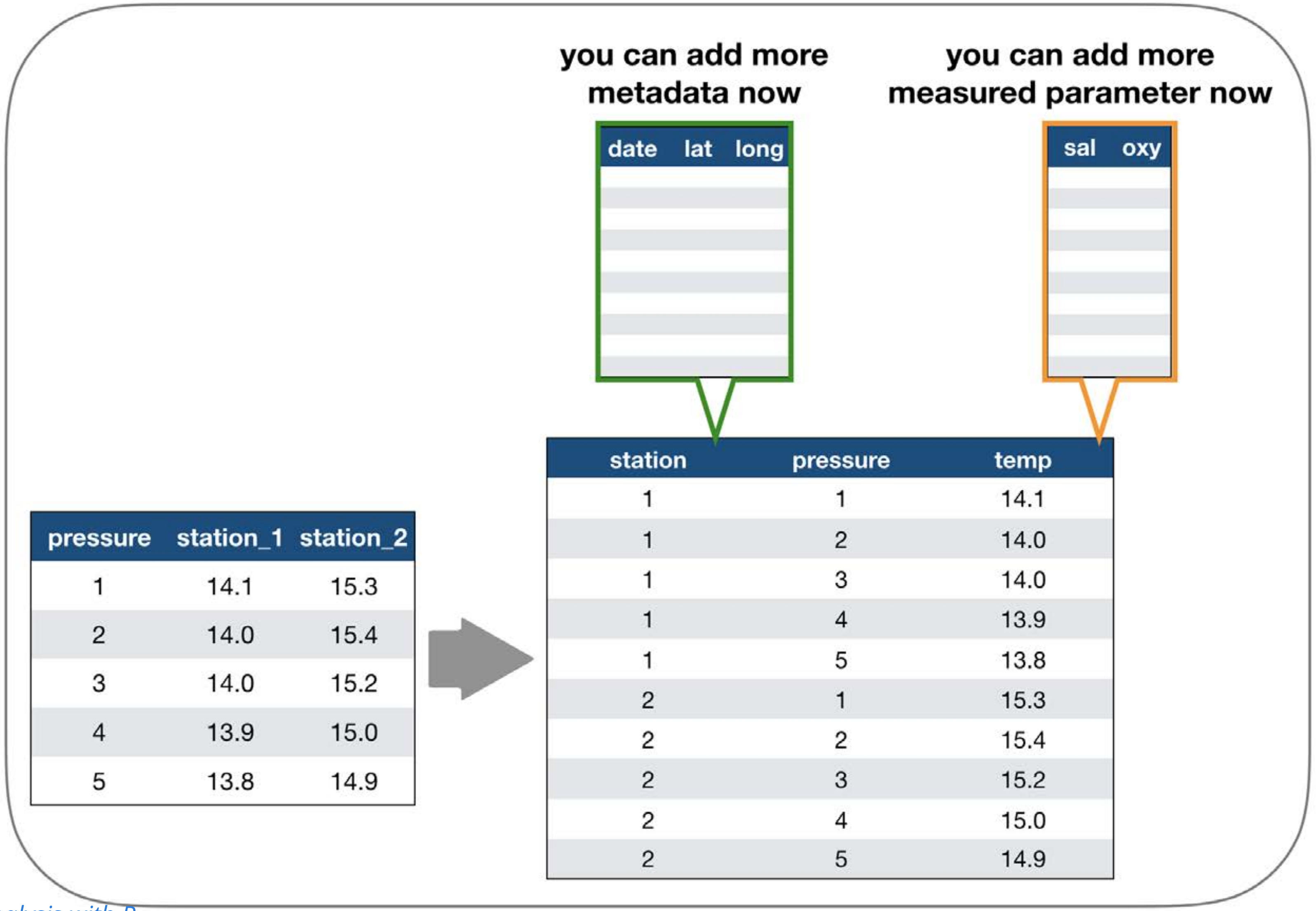
What can be potential issues with such a table?

This is a table from one cruise containing the temperature values.

pressure	station_1	station_2
1	14.1	15.3
2	14.0	15.4
3	14.0	15.2
4	13.9	15.0
5	13.8	14.9

- Where should the coordinates per station be added? In an extra table?
- Where should the date be added? On top of the station names?
- What to do with the other parameters, salinity and oxygen? Different files?
- Not clear that the values represent temperature!

Solution: Restructure the table again



you can add more metadata now
date lat long
you can add more measured parameter now
sal oxy

pressure	station_1	station_2
1	14.1	15.3
2	14.0	15.4
3	14.0	15.2
4	13.9	15.0
5	13.8	14.9

station	pressure	temp
1	1	14.1
1	2	14.0
1	3	14.0
1	4	13.9
1	5	13.8
2	1	15.3
2	2	15.4
2	3	15.2
2	4	15.0
2	5	14.9

Which one do you consider tidy?

Which one do you consider tidy?

- The **'long'** data format is considered more tidy as
    - each observation (here temperature measurement) is in its own row
    - each variable has its own column → station and temperature are not mixed anymore!
- **BUT**: some functions require the data to be in a **wide** format --> you need to adjust your data table in that case

Computer Programming

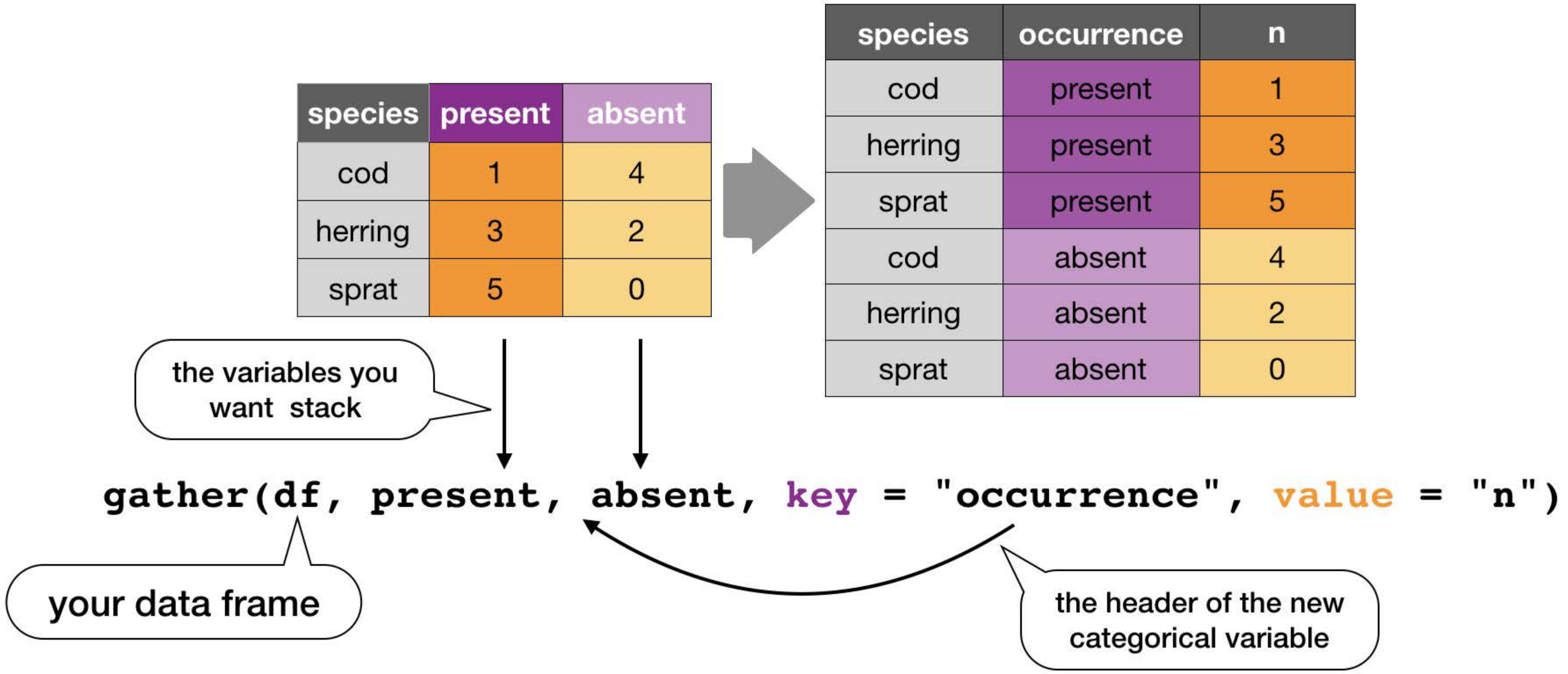
2. Tidy data - How to change between data formats?

'tidyr' provides two functions for that:

- `gather()` and
- `spread()`

`gather()`

Moves column names into a key column, gathering the column values into a single value column.



species	present	absent
cod	1	4
herring	3	2
sprat	5	0

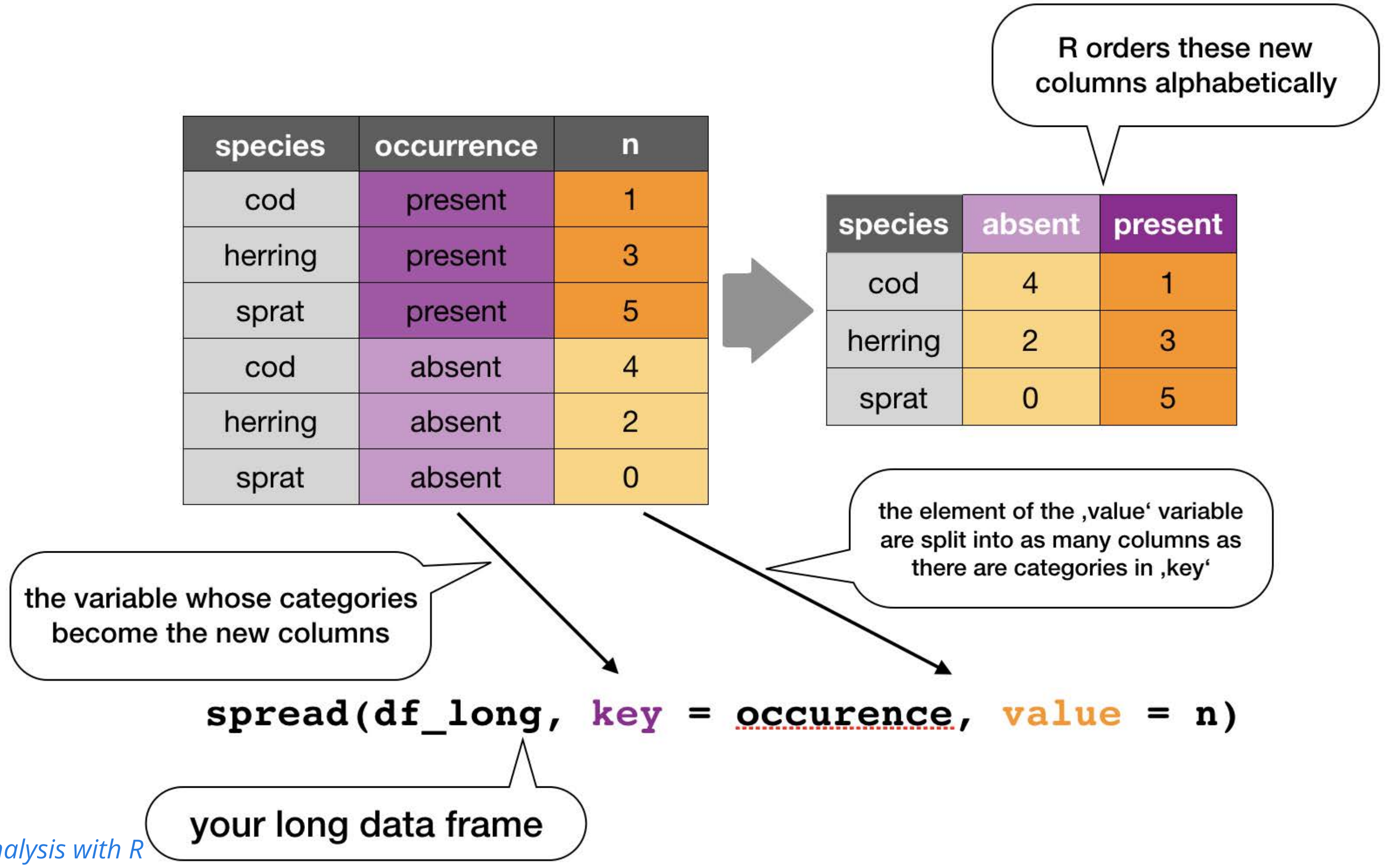
species	occurrence	n
cod	present	1
herring	present	3
sprat	present	5
cod	absent	4
herring	absent	2
sprat	absent	0

```
gather(df, present, absent, key = "occurrence", value = "n")
```

the variables you want stack
your data frame
the header of the new categorical variable

`spread()`

Moves the unique values of a key column into the column names, spreading the values of a value column across the new columns.



species	occurrence	n
cod	present	1
herring	present	3
sprat	present	5
cod	absent	4
herring	absent	2
sprat	absent	0

species	absent	present
cod	4	1
herring	2	3
sprat	0	5

```
spread(df_long, key = occurence, value = n)
```

R orders these new columns alphabetically
the element of the ‚value‘ variable are split into as many columns as there are categories in ‚key‘
the variable whose categories become the new columns
your long data frame

In the R console you would write for the hydrographical example

```r
cruise <- tibble(pressure = 1:5, station_1 = c(14.1,14.0,14.0,13.9,13.8),
  station_2 = c(15.3,15.4,15.2,15.0,14.9))
cruise
# Make tibble long
cruise_long <- gather(cruise, station_1, station_2, key = "station", value = "temp")
cruise_long
# Reshape tibble back into wide format
cruise_wide <- spread(cruise_long, station, temp)
cruise_wide # same as cruise!
```

TRY IT YOURSELF!

Your turn...

Quiz 1: Restructuring data formats

Why are `gather()` and `spread()` not perfectly symmetrical? Carefully consider the following example:

```
(stocks <- tibble(
  year = c(rep(2014,3), rep(2015, 4), rep(2016,2)),
  quarter = c(2:4, 1:4, 1:2), return = round(rnorm(9, mean = 1, sd = 0.3), 2) ) )
```

```
## # A tibble: 9 x 3
##    year quarter return
##   <dbl>   <int>  <dbl>
## 1  2014       2   0.68
## 2  2014       3   0.93
## 3  2014       4   1.12
## 4  2015       1   1.27
## 5  2015       2   1.26
## 6  2015       3   0.8
## 7  2015       4   0.93
## 8  2016       1   1.26
## 9  2016       2   1.27
```

Quiz 1: Restructuring data formats

Why are `gather()` and `spread()` not perfectly symmetrical? Carefully consider the following example:

```r
(stocks <- tibble(
  year = c(rep(2014,3), rep(2015, 4), rep(2016,2)),
  quarter = c(2:4, 1:4, 1:2), return = round(rnorm(9, mean = 1, sd = 0.3), 2) ) )
```

Now lets make the data wide and then long again:

```r
stocks_wide <- spread(stocks, quarter, return)
stocks_long <- gather(stocks_wide, `1`:`4`, key = "quarter", value = "return")
```

What is the difference between stocks and stocks_long? (Hint: look also at the variable types and think about column names.)

Quiz 2: Restructuring data formats

What is the argument 'factor_key' in `gather()` for?

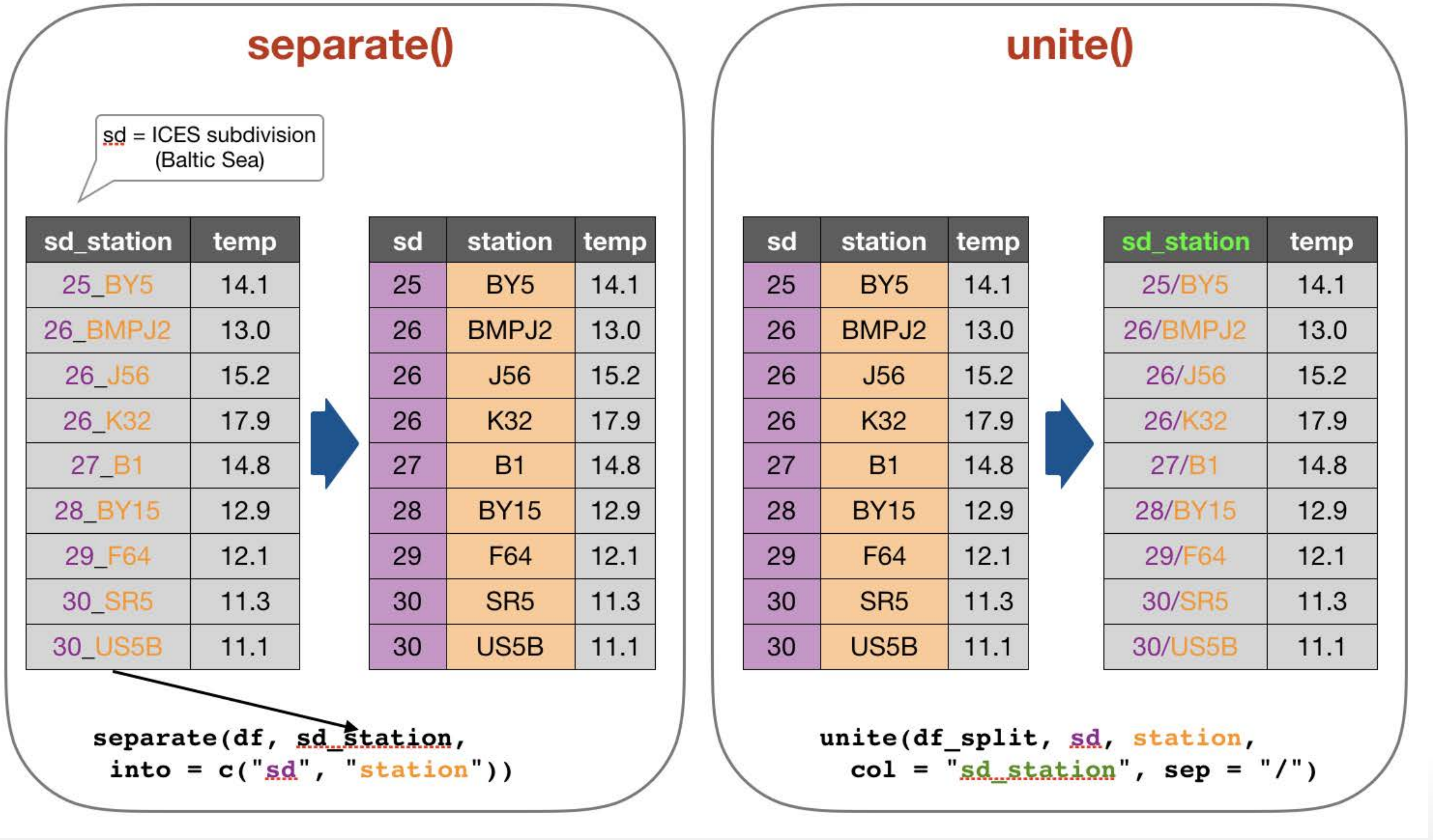
- ○ determines whether key values will be stored as factors
- ○ determines whether factors will be also stacked
- ○ determines whether all factors in the dataset should be coerced to character values

Submit Show Hint Show Answer Clear

2. Tidy data - Separating and uniting cells

Use these functions to split or combine cells into individual, isolated values: `separate()`, `separate_row()`, `unite()`

Use these functions to split or combine cells into individual, isolated values: `separate()`, `separate_row()`, `unite()`



sd_station	temp
25_BY5	14.1
26_BMPJ2	13.0
26_J56	15.2
26_K32	17.9
27_B1	14.8
28_BY15	12.9
29_F64	12.1
30_SR5	11.3
30_US5B	11.1

sd	station	temp
25	BY5	14.1
26	BMPJ2	13.0
26	J56	15.2
26	K32	17.9
27	B1	14.8
28	BY15	12.9
29	F64	12.1
30	SR5	11.3
30	US5B	11.1

```
separate(df, sd_station,
 into = c("sd", "station"))
```

sd	station	temp
25	BY5	14.1
26	BMPJ2	13.0
26	J56	15.2
26	K32	17.9
27	B1	14.8
28	BY15	12.9
29	F64	12.1
30	SR5	11.3
30	US5B	11.1

sd_station	temp
25/BY5	14.1
26/BMPJ2	13.0
26/J56	15.2
26/K32	17.9
27/B1	14.8
28/BY15	12.9
29/F64	12.1
30/SR5	11.3
30/US5B	11.1

```
unite(df_split, sd, station,
  col = "sd_station", sep = "/")
```

Your turn...

Try it out yourself

```r
df <- tibble(
  sd_station = c("25_BY5", "26_BMPJ2", "26_J56", "26_K32",
    "27_B1", "28_BY15", "29_F64", "30_SR5", "30_US5B"),
  temp = c(14.1, 13.0, 15.2, 17.9, 14.8, 12.9, 12.1, 11.3, 11.1)
)

df_split <- separate(df, sd_station, into = c("sd", "station"))

df_join <- unite(df_split, sd, station, col = "sd_station", sep = "/")
```

Quiz 3: Separating and uniting cells

What do the extra and fill arguments do in `separate()`? Experiment with the various options for the following two toy datasets.

```
df1 <- tibble(x = c("a,b,c", "d,e,f,g", "h,i,j"))
separate(df1, x, c("one", "two", "three"), remove =F)

df2 <- tibble(x = c("a,b,c", "d,e", "f,g,i"))
separate(df2, x, c("one", "two", "three"))
```

(The solution is at the end of the presentation.)

Quiz 4: Separating and uniting cells

Both `unite()` and `separate()` have a remove argument. What does it do?

- ○ It removes extra values.
- ○ It removes the column names.
- ○ It removes the original colum.
- ○ It removes special characters in the separating column.

Why would you set it to `FALSE`?

Submit Show Hint Show Answer Clear

Computer Programming

2. Tidy data - Handling missing values and other replacements

Handling missing values (NAs)

- **Leave them** in and **accomodate algorithm** to missing data, e.g. missing values are skipped during calculations like "pairwise deletion"; can cause problems
- **Delete** rows/columns
- **Interpolate** missing values
    - replace NAs by **mean/median** (advantage median: distribution can be skewed)
    - replace NAs by **regression** (linear interpolation)

Checking for NAs

You can apply the function `is.na()` to single vectors and **single variables in a data frame** as you learned in lecture 2. Here an example with the ICES dataset

```r
is.na(hydro$temp)
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [34] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [56]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
##  [89] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [133]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [155] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [166] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [177] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [188] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [199] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [210] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

..... Puh!

Remember, the `is.na()` function returns a logical vector of the **same length then the original vector** (which has, in our case, **30012** values!).

BETTER: If you wrap the function by the `sum()`function, you can calculate the sum of all TRUEs in this vector:

```
sum(is.na(hydro$temp))
```

```
## [1] 1714
```

```
sum(is.na(hydro$psal))
```

```
## [1] 2382
```

You can do the same with the entire data frame

```
sum(is.na(hydro))
```

```
## [1] 13809
```

A shortcut to check NAs in all variables

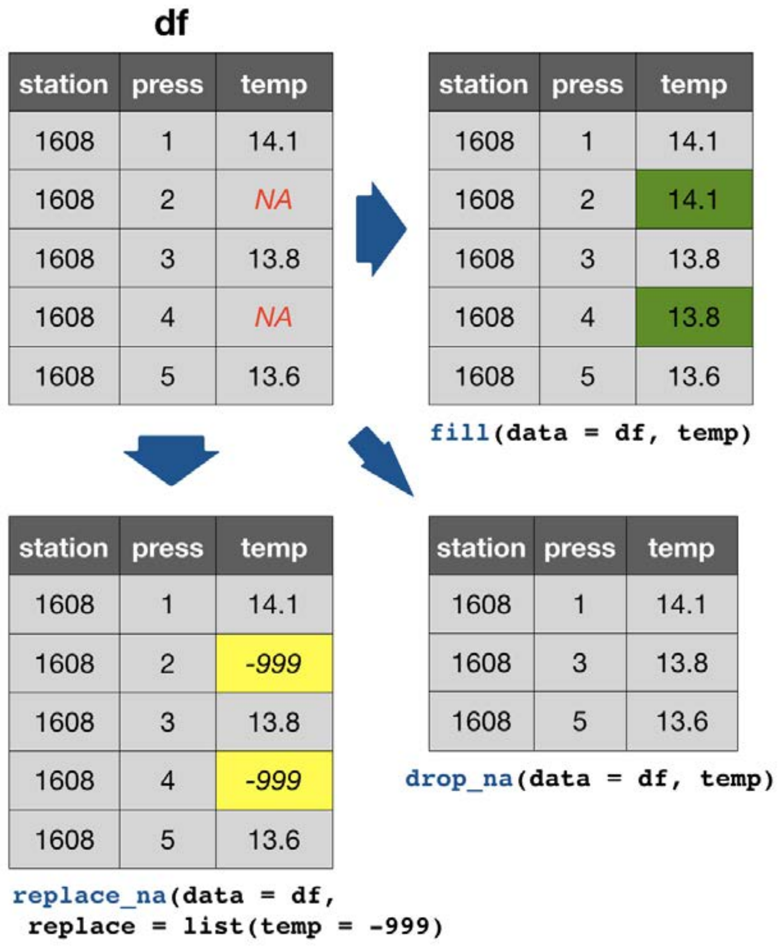
A very fast way to see whether and how many NAs you have in a dataframe is to use the `summary()` function, which displays not only some **descriptive statistics** but also the **number of NAs**:

```r
summary(hydro[ ,9:11]) # for display purposes I selected not all columns
```

```
##       temp              psal              doxy
##  Min.   :-0.700   Min.   : 0.065   Min.   : 0.000
##  1st Qu.: 4.200   1st Qu.: 6.548   1st Qu.: 5.870
##  Median : 6.067   Median : 7.430   Median : 6.960
##  Mean   : 7.668   Mean   : 8.082   Mean   : 6.493
##  3rd Qu.:10.500   3rd Qu.: 8.436   3rd Qu.: 8.200
##  Max.   :24.400   Max.   :34.016   Max.   :11.760
##  NA's   :1714     NA's   :2382     NA's   :7304
```

tidyr provides 3 useful functions for handling NAs

1. `drop_na(data, ...)`: Drops entire rows containing missing values.
2. `fill(data, ..., .direction = c("down", "up"))`: Fills missing values in using the previous (direction = "down") or following (direction = "up") entry. Useful if only values that change are recorded.
3. `replace_na(data, replace = list(), ...)`: Replaces missing values with a value specified for each column.



df

station	press	temp
1608	1	14.1
1608	2	*NA*
1608	3	13.8
1608	4	*NA*
1608	5	13.6

station	press	temp
1608	1	14.1
1608	2	14.1
1608	3	13.8
1608	4	13.8
1608	5	13.6

```
fill(data = df, temp)
```

station	press	temp
1608	1	14.1
1608	2	*-999*
1608	3	13.8
1608	4	*-999*
1608	5	13.6

```
replace_na(data = df,
 replace = list(temp = -999)
```

station	press	temp
1608	1	14.1
1608	3	13.8
1608	5	13.6

```
drop_na(data = df, temp)
```

Replacing (single) values other than NAs

Use the indexing rules and operators for subsetting you've learned so far:

```r
cruise <- tibble(pressure = 1:5, station_1 = c(4.1,4.0,4.0,3.9,33.8),
  station_2 = c(-5.3,-5.4,5.2,5.0,4.9))
```

Replacing (single) values other than NAs

Use the indexing rules and operators for subsetting you've learned so far:

```r
cruise <- tibble(pressure = 1:5, station_1 = c(4.1,4.0,4.0,3.9,33.8),
  station_2 = c(-5.3,-5.4,5.2,5.0,4.9))
```

1. Replace a **single** value, e.g. set an extreme value to NA so it gets excluded in the analysis:

```r
# if you know the row of the outlier
cruise$station_1[5] <- NA
```

1. Replace **all** negative temperature values with NAs (might be typing or measurement errors)

```r
cruise$station_2[
  cruise$station_2 < 0] <- NA
# works only if you have no NAs,
# otherwise try
cruise$station_2[
  !is.na(cruise$station_2) &
  cruise$station_2< 0] <- NA
```

Replacing (single) values other than NAs (cont)

A useful function to identify the position of elements in a vector for which the specified conditions holds is `which()`. Lets identify the temperature value(s) that are equal to the maximum observed temperature value and set this/these to NA:

```r
id <- which(cruise$station_1 == max(cruise$station_1))
id
```

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

```r
cruise$station_1[id] <- NA
```

Your turn...

Task: Replace NAs

We have the following dataset:

```r
df <- tibble(
  station = paste0("station_", 1:20), # concatenates "station" with the number 1 to
  temp = rnorm(20, mean = 14, sd = 2), # generates 20 values from a normal distribut
  sal = rnorm(20, mean = 8, sd = 0.5)
)
random_nas <- sample(1:20, 7) # samples 7 random positions
df$temp[random_nas] <- NA # fill these positions in variable 'temp' with NAs
df$sal[random_nas] <- NA
```

Quiz 5: Replace NAs in temp

Try to deal with the NAs in the variable 'temp' in the following way:

1. Drop all rows with NAs in 'temp'
2. Replace NAs with the previous value
3. Replace NAs with the mean value (of the entire variable)
4. Replace NAs with -999

What are the consequences if you do this? What is the best option (out of the four and having the original NAs in) for which situation?

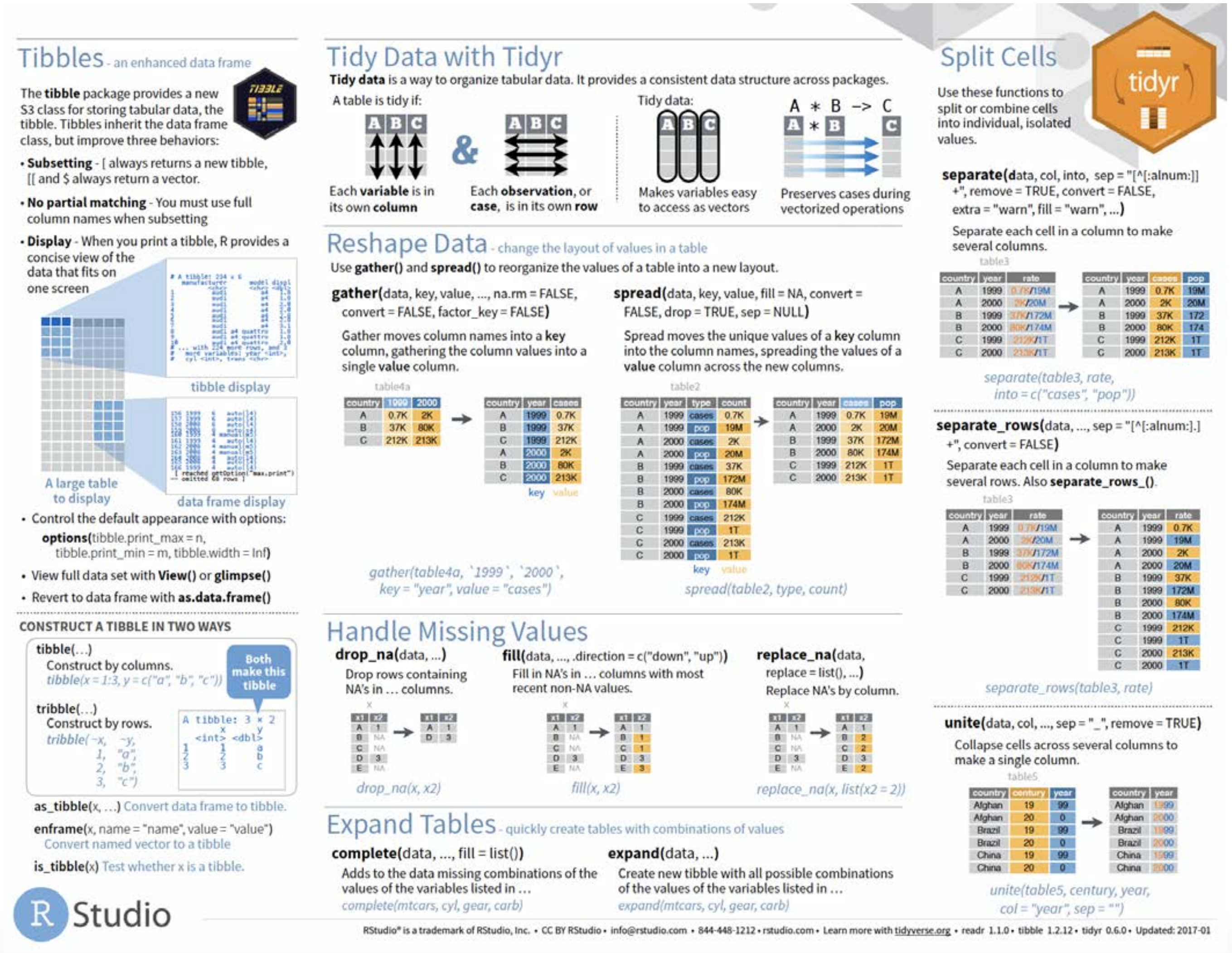
As a help: calculate for each option the mean temperature and compare the results (the solution is at the end of the presentation).

Quiz 6: Replace NAs in temp AND sal

Restore the original df (including the NAs in temp) and figure out how to change NAs (with the previous value) in both variables 'temp' and 'sal' **IN ONE STEP**.

(The solution is at the end of the presentation.)

Overview of more functions:



Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:

- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen

- Control the default appearance with options:
  options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)**
- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

CONSTRUCT A TIBBLE IN TWO WAYS

tibble(…)
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…)
Construct by rows.
tribble(~x, ~y,
1, "a",
2, "b",
3, "c")

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <dbl>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …**)** Convert data frame to tibble.

enframe(x, name = "name", value = "value"**)**
Convert named vector to a tibble

is_tibble(x**)** Test whether x is a tibble.

RStudio

Tidy Data with Tidyr

Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

Tidy data:

Makes variables easy to access as vectors

A * B -> C

Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE**)**

Gather moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL**)**

Spread moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

key value

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

spread(table2, type, count)

Handle Missing Values

drop_na(data, ...**)**
Drop rows containing NA's in … columns.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(x, x2)

fill(data, ..., .direction = c("down", "up")**)**
Fill in NA's in … columns with most recent non-NA values.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(x, x2)

replace_na(data, replace = list(), ...**)**
Replace NA's by column.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(x, list(x2 = 2))

Expand Tables - quickly create tables with combinations of values

complete(data, ..., fill = list()**)**
Adds to the data missing combinations of the values of the variables listed in …
complete(mtcars, cyl, gear, carb)

expand(data, ...**)**
Create new tibble with all possible combinations of the values of the variables listed in …
expand(mtcars, cyl, gear, carb)

Split Cells

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...**)**

Separate each cell in a column to make several columns.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174
C	1999	212K	1T
C	2000	213K	1T

separate(table3, rate, into = c("cases", "pop"))

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE**)**

Separate each cell in a column to make several rows. Also **separate_rows_()**.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M
C	1999	212K
C	1999	1T
C	2000	213K
C	2000	1T

separate_rows(table3, rate)

unite(data, col, ..., sep = "_", remove = TRUE**)**

Collapse cells across several columns to make a single column.

table5

country	century	year
Afghan	19	99
Afghan	20	0
Brazil	19	99
Brazil	20	0
China	19	99
China	20	0

→

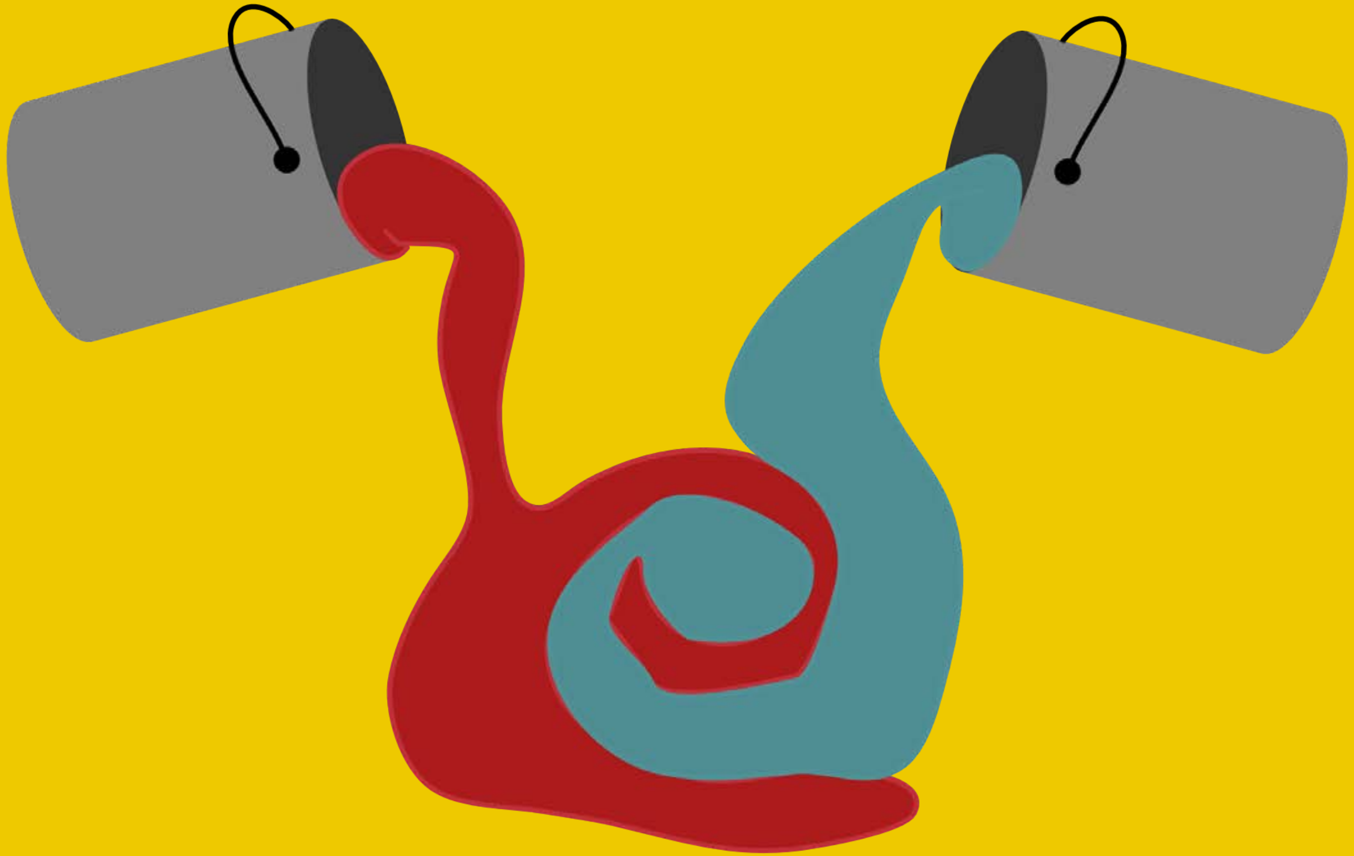
country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

unite(table5, century, year, col = "year", sep = "")

Cheat sheet is freely available at https://www.rstudio.com/resources/cheatsheets/

Your turn again...

Let’s put together everything you’ve learned to tackle a realistic data tidying problem.



Tasks

1. Import the hydrographical ICES dummy dataset: "dummy_hydro.csv"

2. Make the data tidy

WHAT ARE THE THINGS TO CONSIDER????

Tasks

1. Import the hydrographical ICES dummy dataset: "dummy_hydro.csv"
2. Make the data tidy
   - Is any restructuring needed?
   - Is any separation or union needed?
   - Do you agree with the column names?
   - Are the data types correct?
   - Do you need to handle NAs?
   - Are there any awkward values in the data (potential typing errors)?

base package: `is.na()`, `which()`, `summary()`

tidyr package: `gather()`, `spread()`, `separate()`, `separate_row()`, `unite()`, `drop_na()`, `fill()`, `replace_na()`

Overview of functions you learned today

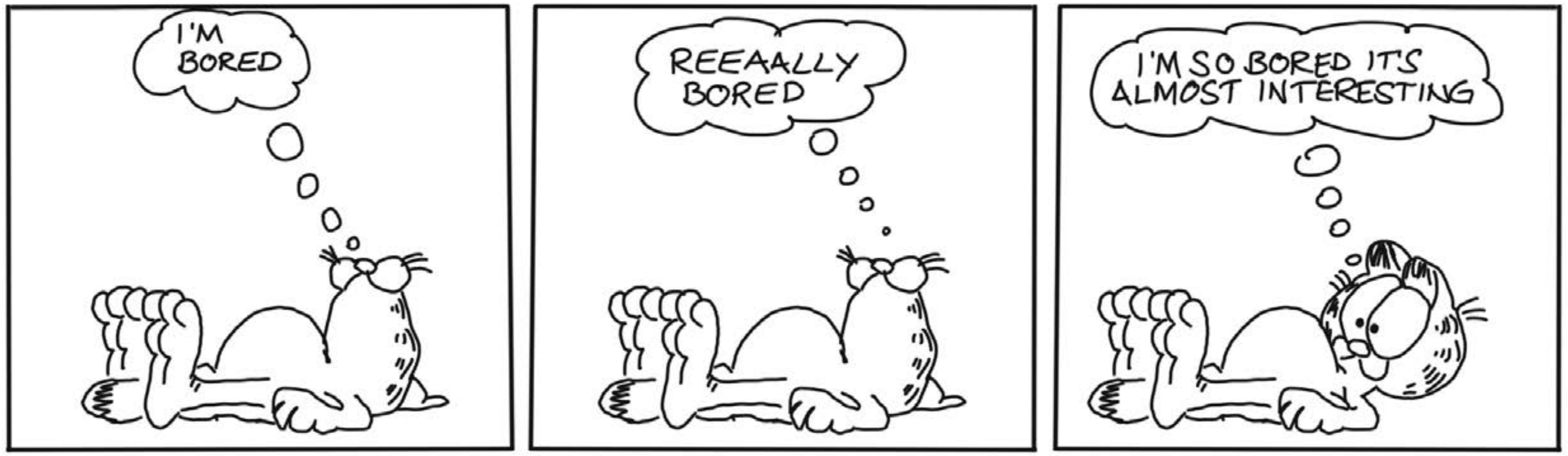
How do you feel now.....?

Totally confused?

23uzr8?! §€@@´´`§/;...€ hbfj,§%46;)9=.,§4`´ 7:@,9 RRR4RRR8RRR RRRROORRR§()

Go thorougly through the tasks and quizzes. Read the chapter 12 Tidy Data in 'R for Data Science'.

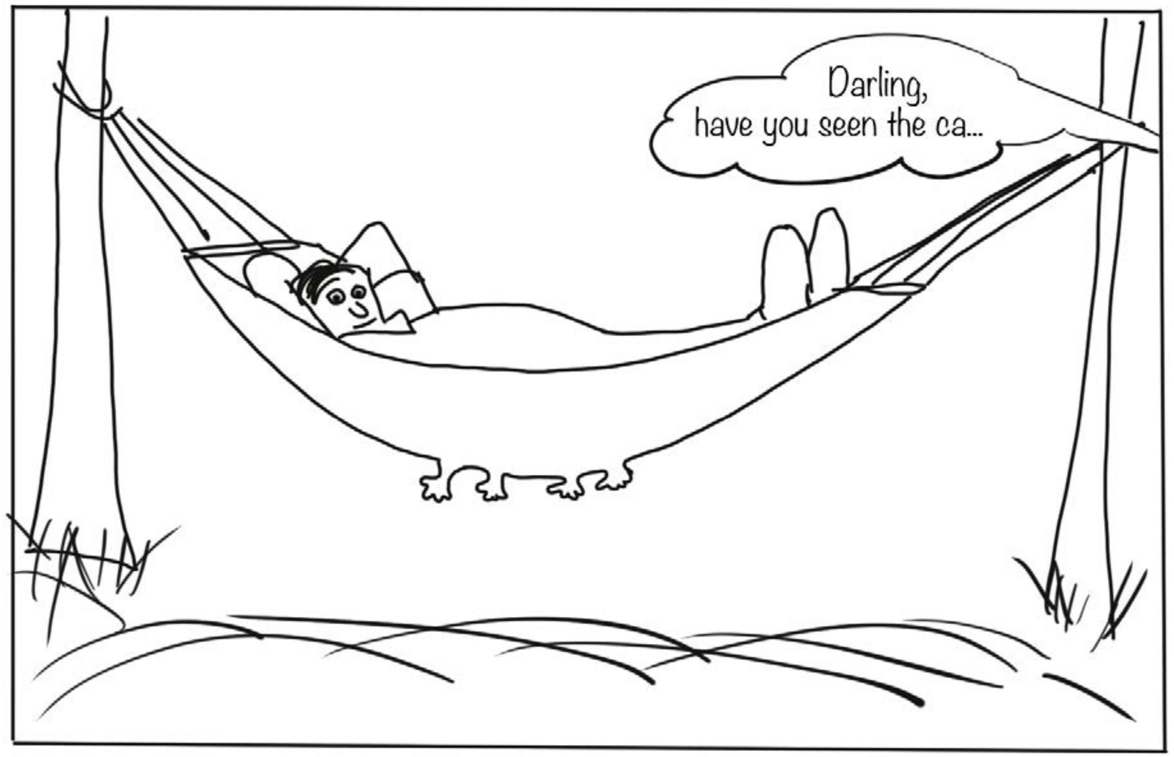
Totally bored?



Then play around already with the full hydro dataset "1111473b.csv" and make it tidy.

Totally content?

Then go grab a coffee, lean back and enjoy the rest of the day...!



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Thank You

For more information contact me: saskia.otto@uni-hamburg.de

http://www.researchgate.net/profile/Saskia_Otto
http://www.github.com/saskiaotto

Solutions

Quiz 3: Separating and uniting cells

- df1: row 2 has one element MORE when splitting the string into individual columns —> R will return an error message and drop the extra value as default. If you want to get no warning or keep the extra value (united with the previous value) change the 'extra' argument to extra = „drop“ or „merge“:

```
separate(df1, x, c("one", "two", "three"), extra = "merge")
```

- df2: row 2 has one element LESS when splitting the string into individual columns —> with the default settings (fill = „warn“) R will fill the empty element with NA from the right and return a message. If you want to fill from the left without any warning, you should change to fill = „left“:

```
separate(df2, x, c("one", "two", "three"), fill = "left")
```

Quiz 5: Replace NAs in temp

```r
# This would be your code:
# 1.
drop_na(df, temp)
# 2.
fill(df, temp) # default is previous value
# 3.
temp_mean <- mean(df$temp, na.rm= TRUE)
replace_na(df, list(temp = temp_mean))
# 4.
replace_na(df, list(temp = -999))
```

Some function cannot handle NAs so one way to solve the problem is to drop the entire rows that contain NAs somewhere or replace the NAs with some values such as the mean, median, or previous values:

Quiz 5: Replace NAs in temp (cont)

- **dropping rows**: a simple, clean solution that doesn't bring any artificial information into the data. But you might loose so many rows that your sample size becomes too small. In that case, filling NAs with values is a better option.

- **previous values**: can be useful in time series or spatial data (including depth profiles) where values next to each other (e.g. temperatures the next day or the adjacent water depth) are likely to be similar. Problematic here is if you have large gaps, e.g. the temperature in 1m depth and then in 100m depth again. For all intermediate depths (2-99m) the values would resemble the one from 1m depth, which is highly unrealistic. In such a case, removing the entire sample might be the best choice.

Quiz 5: Replace NAs in temp (cont)

- **mean**: replacing NAs with means (or medians) is a common technique, particularly if samples are not closely related (e.g. temperature measurements of stations far away from each other). But, think carefully which mean to take. The overall mean (across all samples, stations, etc)? The mean of that sample? The mean of the specific depth? The mean of the specific depth in the specific month?

- Some datasets come with the value -999 for missing values. To be consistent you might want to do the same for other datasets. But as soon as you do any calculations, this value will be treated as a real number messing up all your results (imagine the mean of 15, 16 and -999). Always change such missing values to NAs before you do any calculation, visualisation, or modelling.

Best solution: there isn't a best solution but a good practice is to keep your missing values as NAs and only for particular methods you deal with NAs in a way that is best suited for the type of data.

Quiz 6: Replace NAs in temp AND sal

If you don’t specify any variable, NAs will be replace in the entire dataset:

```
fill(df)
```

If you want to make sure only in temp and sal NAs are replaced specify that:

```
fill(df, temp, sal)
```