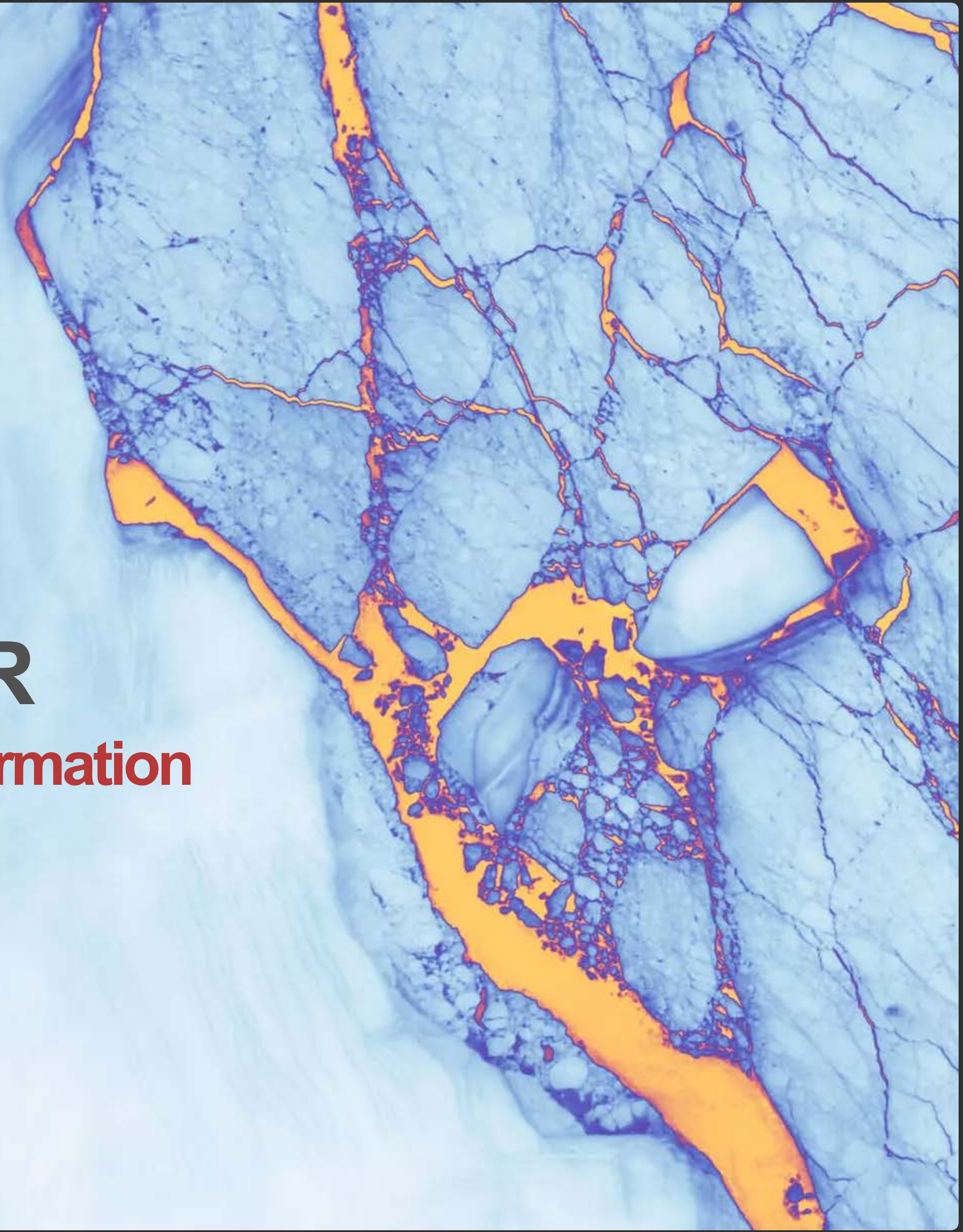
Marine Data Science

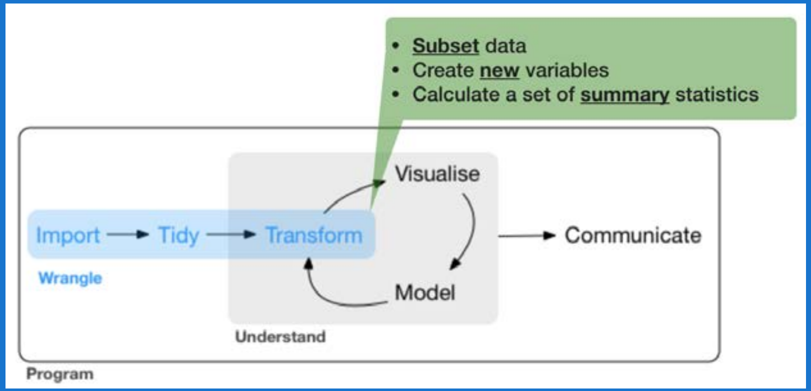
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG



Data Analysis with R

7 - Data wrangling - 3.Transformation

Saskia A. Otto
Postdoctoral Researcher



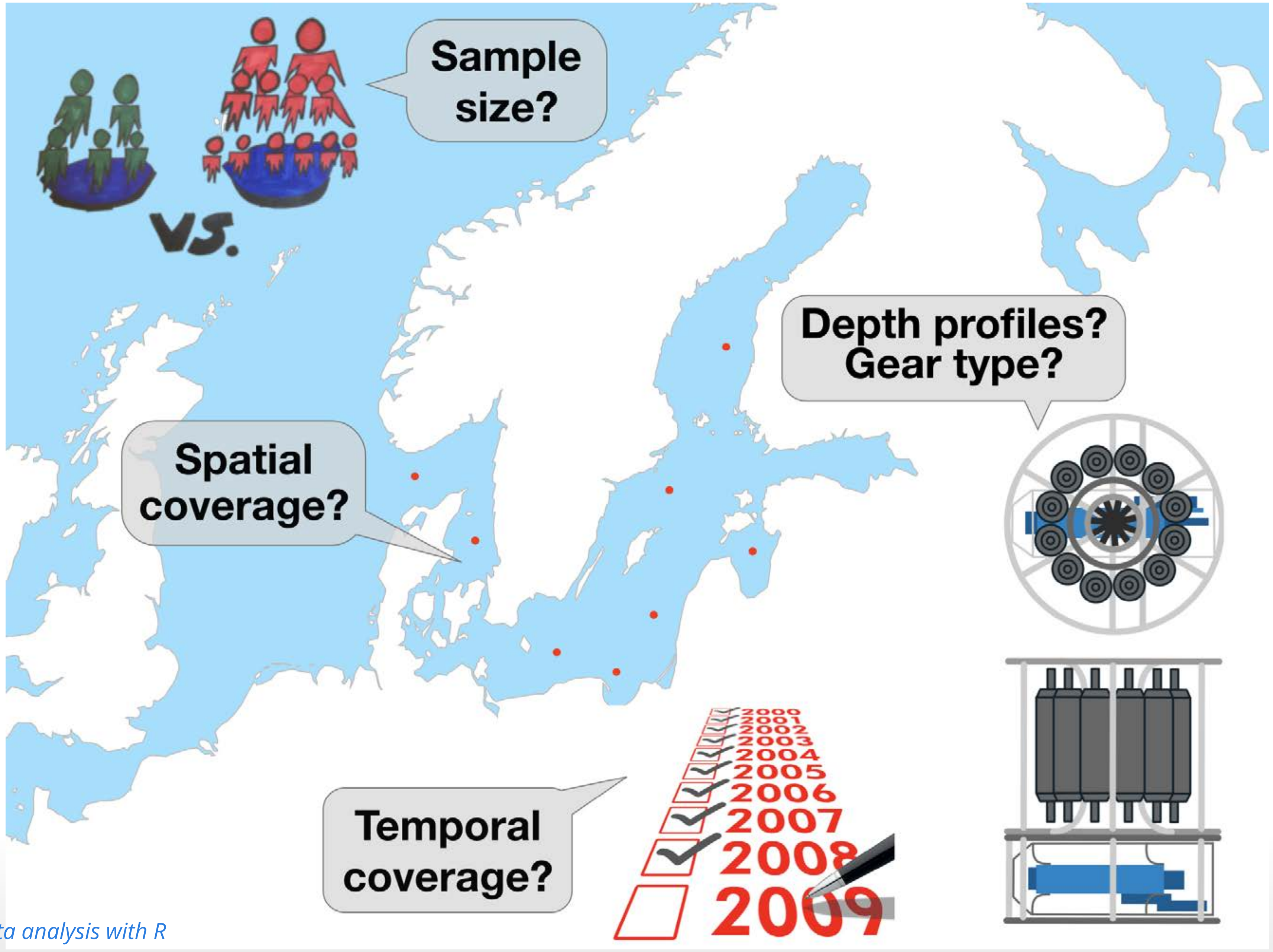
3. Data transformation

Before you start manipulating, ask yourself what do you want to explore?

What could be interesting questions for the oceanographic ICES data?

- Develop some questions yourself...

The **first thing** that should be checked before getting into the actual analysis of the oceanographic data is the **data quality**!


Sample size?
VS.
Spatial coverage?
Depth profiles? Gear type?
Temporal coverage?
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009

These questions require data aggregation

Aggregations can be done by

These questions require data aggregation

Aggregations can be done by

- **gear type**

These questions require data aggregation

Aggregations can be done by

- **gear type**
- **datasource**

These questions require data aggregation

Aggregations can be done by

- **gear type**
- **datasource**
- **spatially** (e.g., per ICES subdivision or rectangle in this example)

These questions require data aggregation

Aggregations can be done by

- **gear type**
- **datasource**
- **spatially** (e.g., per ICES subdivision or rectangle in this example)
- **temporally** (e.g., per year or month) → This requires some knowledge of handling dates and times!

Computer Programming

Handling dates and times

Measurement of time is highly idiosyncratic

Measurement of time is highly idiosyncratic

Surprisingly difficult for computers!

Dates and times in R

To see how R handles dates and times, have a look at `Sys.time()`:

```
Sys.time()
```

```
## [1] "2018-10-24 12:00:42 CEST"
```

You see, first comes the year-month-day, then the time (h:m:s), and then the time zone. If you type

```
unclass(Sys.time())
```

```
## [1] 1540375242
```

You get the number of seconds since 1 January 1970.

Dates and times in R

- Two basic classes of date/times:
    - **POSIXlt**
    - **POSIXct**
- In tidyverse 3 types of date/time data that refer to an instant in time:

Dates and times in R

- Two basic classes of date/times:
  - **POSIXlt**
  - **POSIXct**
- in tidyverse 3 types of date/time data that refer to an instant in time:

1. **date**: Tibbles print this as `<date>`.
2. **time** within a day: Tibbles print this as `<time>`.
3. **date-time** is a date plus a time: it uniquely identifies an instant in time (typically to the nearest second). Tibbles print this as `<dttm>`.

Dates and times in R (cont)

- Many ways of writing the date and time → **importing** the correct date format and **extracting parts** can be tricky!
- Always use the **simplest** possible data type that works for your needs.
- One tidy way to import the correct date and time is with the `parse_functions` in the **readr** package → but it requires some knowledge on the specification of the date format you want to import.

The 'lubridate' package

- Makes it **easier** to work with dates and times.
- Handles a wide **variaty of formats** automatically.
- Is **not** part of the **core** tidyverse so it needs to be installed once and loaded additionally every time:.

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

Create DATE objects from string

Depending on the order of the date components you have 3 functions to choose from:

```r
ymd("2017-11-17") # YEAR-MONTH-DAY
```

```
## [1] "2017-11-17"
```

```r
mdy("Nov 17th, 2017") # MONTH-DAY-YEAR
```

```
## [1] "2017-11-17"
```

```r
dmy("17-Nov-2017") # DAY-MONTH-YEAR
```

```
## [1] "2017-11-17"
```

- **Only the order matters!** The format is not important as lubridate will automatically recognize it.
- You can apply the function to an entire vector.

Create DATE-TIME objects from strings

Simply combine `ymd`, `mdy` or `dmy` with

- `_h` if you have only the hour
- `_hm` if you have hour and minute
- `_hms` for hour:min:sec

```
# Date with HOUR-MIN-SEC
ymd_hms("2017-11-17 12:11:59")
```

```
## [1] "2017-11-17 12:11:59 UTC"
```

```
# Date with HOUR-MIN
mdy_hm("11/17/2017 12:11")
```

```
## [1] "2017-11-17 12:11:00 UTC"
```

Create DATE-TIME objects from strings (cont)

If the time zone is not UTC (default) specify the `tz` argument

```
mdy_hm("11/17/2017 12:11", tz = "CET")
```

```
## [1] "2017-11-17 12:11:00 CET"
```

```
mdy_hm("11/17/2017 12:11", tz = "Europe/Helsinki")
```

```
## [1] "2017-11-17 12:11:00 EET"
```

CET = Central European Time, EET = Eastern European Time

Create DATE-TIME objects from individual components

If the date is split into different columns in your dataset you can combine them to a date object using `make_date()` or `make_datetime()`:

```
make_date(year = 2017, month = 11, day = 15:17)
```

```
## [1] "2017-11-15" "2017-11-16" "2017-11-17"
```

Year and month get recycled to the same length as days.

Switch between date-time and date

You can switch between both formats with `as_date()` and `as_datetime()` (but you might loose information):

```r
dt_utc <- mdy_hm("11/17/2017 12:11")
dt_utc
```

```
## [1] "2017-11-17 12:11:00 UTC"
```

```r
d_utc <- as_date(dt_utc)
d_utc
```

```
## [1] "2017-11-17"
```

```r
as_datetime(d_utc)
```

```
## [1] "2017-11-17 UTC"
```

Extract date components

For aggregation purposes its often useful to extract individual components. Lubridate has the following helper functions (all have simply the name of the component you want to extract):

- `year()`
- `month()`
- `mday()` - day of the month
- `yday()` - day of the year
- `wday()` - day of the week
- `hour()`, `minute()`, `second()`

```r
dt_utc <- mdy_hm("11/17/2017 12:11")
year(dt_utc)
```

```
## [1] 2017
```

```r
yday(dt_utc)
```

```
## [1] 321
```

Handling time periods, intervals

Lubridate offers many more functions that deal with dates and times such as

- `%--%` creates intervals
- and `as.duration()` calculates the duration of this interval

```
(day_int <- dmy("10/11/2017") %--% dmy("17/11/2017") )
```

```
## [1] 2017-11-10 UTC--2017-11-17 UTC
```

```
as.duration(day_int)
```

```
## [1] "604800s (~1 weeks)"
```

To learn more on functions offered by lubridate read the vignette or chapter 16 in R for Data Science.

Your turn...

Import the following dataset

```r
library(tidyverse)
date_ex <- read_csv("data/date_time_examples.csv")
print(date_ex, n = 5)
```

```
## # A tibble: 10 x 8
##   date1 date2 date3 sampling_start_CET  sampling_end_UTC     year month
##   <chr> <chr> <chr> <dttm>              <dttm>              <int> <int>
## 1 11-0… 8.11… 8 No… 2017-11-08 09:54:00 2017-11-08 10:40:00  2017    11
## 2 11-0… 9.11… 9 No… 2017-11-09 08:15:00 2017-11-09 09:07:00  2017    11
## 3 11-1… 10.1… 10 N… 2017-11-10 08:06:00 2017-11-10 09:09:00  2017    11
## 4 11-1… 11.1… 11 N… 2017-11-11 10:37:00 2017-11-11 11:59:00  2017    11
## 5 11-1… 12.1… 12 N… 2017-11-12 08:21:00 2017-11-12 09:02:00  2017    11
## # ... with 5 more rows, and 1 more variable: day <int>
```

Quiz 1: Handling dates

Which of the variables have been correctly parsed as dates?

- ☐ date1
- ☐ date2
- ☐ date3
- ☐ sampling_start_CET
- ☐ sampling_end_UTC
- ☐ year
- ☐ month
- ☐ day

Submit | Show Hint | Show Answer | Clear

Quiz 2: Handling dates

Convert variables `date1`, `date2`, and `date3` into the date format. Which are the correct functions for each date format? Do they look the same after conversion?

Quiz 3: Handling dates

Create a new date variable by combining the `year`, `month`, and `day` variables.

Quiz 4 - Challenge: Handling dates and times

A video plankton recorder (VPR) was towed along a transect in the Skagerrak (North of Denmark) from East to West on several subsequent days. The starting and ending time of the tow were recorded each time (col 4 and 5). Can you tell me for how long the VPR was towed at the following sampling dates (in min)?

1. 2017-11-08
2. 2017-11-14
3. 2017-11-15

Submit Show Hint Show Answer Clear

Exercise: Date-time in the ICES hydrographical data

As preparation for the following data manipulation, import the ICES data, change the variable names, and check the date-time variable. Was the format correctly parsed? (Don't forget to set the working directory beforehand!)

```r
hydro <- read_csv("data/1111473b.csv")
# Change names to e.g.
names(hydro) <- c("cruise", "station", "type", "date_time",
  "lat", "long", "depth", "pres", "temp", "psal", "doxy")
```

Create 3 new columns that contain the

- **year**
- **month**
- **day**

Data Exploration
& Visualization

Data transformation with 'dplyr'

The 'dplyr' package

Makes data manipulation easier and faster

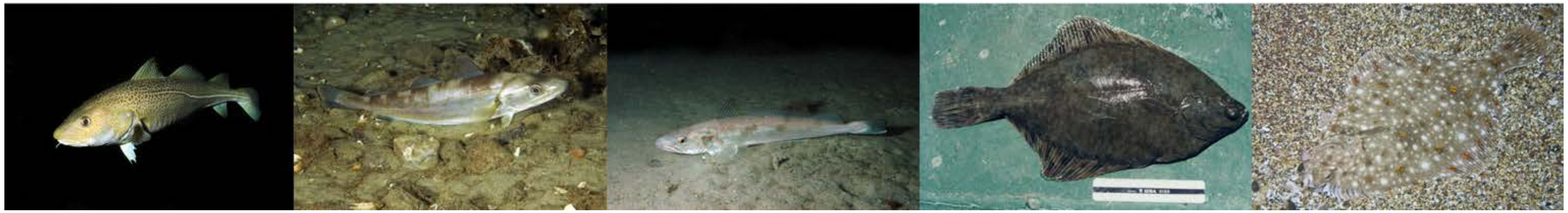
TYPICAL MANIPULATIONS	CORE FUNCTIONS IN DPLYR
Manipulate observations (rows)	`filter()`, `arrange()`
Manipulate variables (columns)	`select()`
Summarise observations	`summarise()`
Group observations	`group_by()`, `ungroup()`
Combine tables	`bind_` and `join_` functions

The 'dplyr' package (cont)

Function structure

- First argument is always a data frame or tibble
- Subsequent arguments say what to do with data frame
- Always return a data frame

A demonstration with growth information for 5 fish species

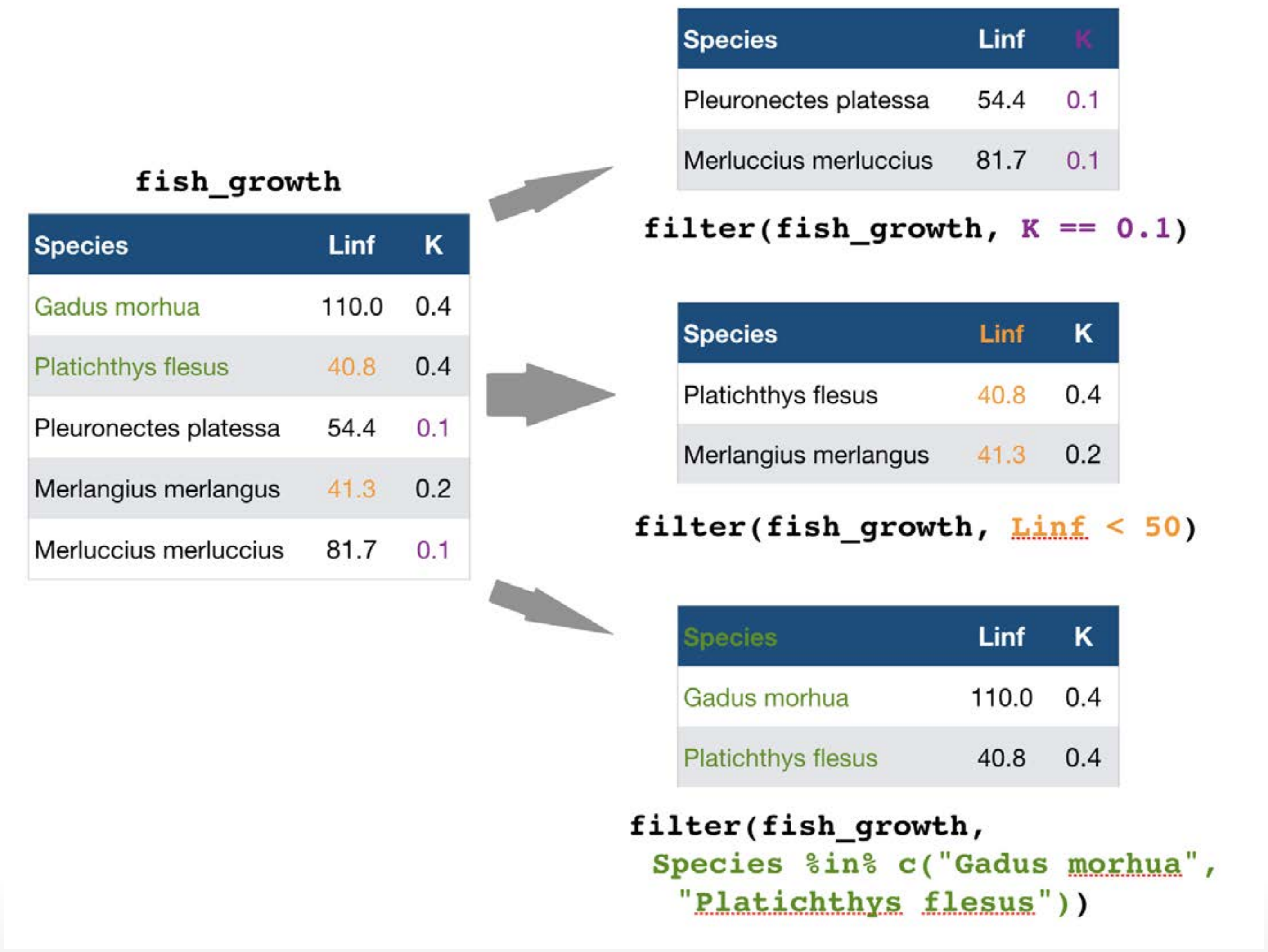


```r
fish_growth <- tibble(
  Species = c("Gadus morhua", "Platichthys flesus", "Pleuronectes platessa",
    "Merlangius merlangus", "Merluccius merluccius"),
  Linf = c(110, 40.8, 54.4, 41.3, 81.7),
  K = c(0.4, 0.4, 0.1, 0.2, 0.1)
)
```

(Linf = average maximum length, K = rate at which the fish approaches Linf)

Image courtesy of the photographers at fishbase.org (Konstantinos I. Stergiou, Jim Greenfield) and uwphoto.com (Rudolf Svensen).

`filter()` → extract rows that meet logical criteria



fish_growth

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4
Pleuronectes platessa	54.4	0.1
Merlangius merlangus	41.3	0.2
Merluccius merluccius	81.7	0.1

Species	Linf	K
Pleuronectes platessa	54.4	0.1
Merluccius merluccius	81.7	0.1

```
filter(fish_growth, K == 0.1)
```

Species	Linf	K
Platichthys flesus	40.8	0.4
Merlangius merlangus	41.3	0.2

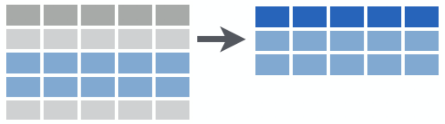
```
filter(fish_growth, Linf < 50)
```

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4

```
filter(fish_growth,
 Species %in% c("Gadus morhua",
  "Platichthys flesus"))
```

Some other helpful functions to ...

Subset Observations (Rows)



dplyr::**filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**

Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**

Randomly select n rows.

dplyr::**slice(iris, 10:15)**

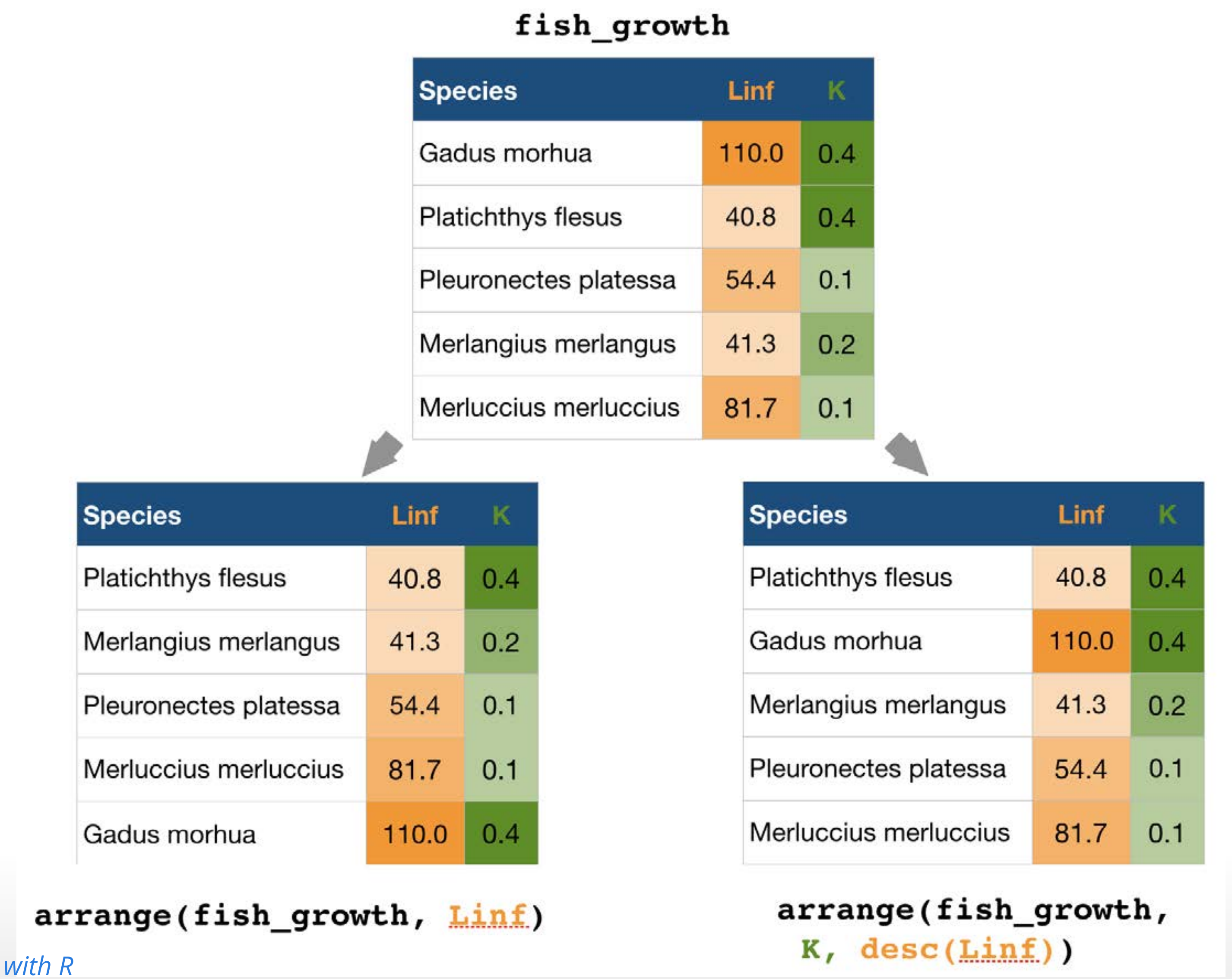
Select rows by position.

dplyr::**top_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

source: older version of Data Transformation with dplyr cheat sheet (licensed under CC-BY-SA)

`arrange()` → sort observations by specific variables



`fish_growth`

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4
Pleuronectes platessa	54.4	0.1
Merlangius merlangus	41.3	0.2
Merluccius merluccius	81.7	0.1

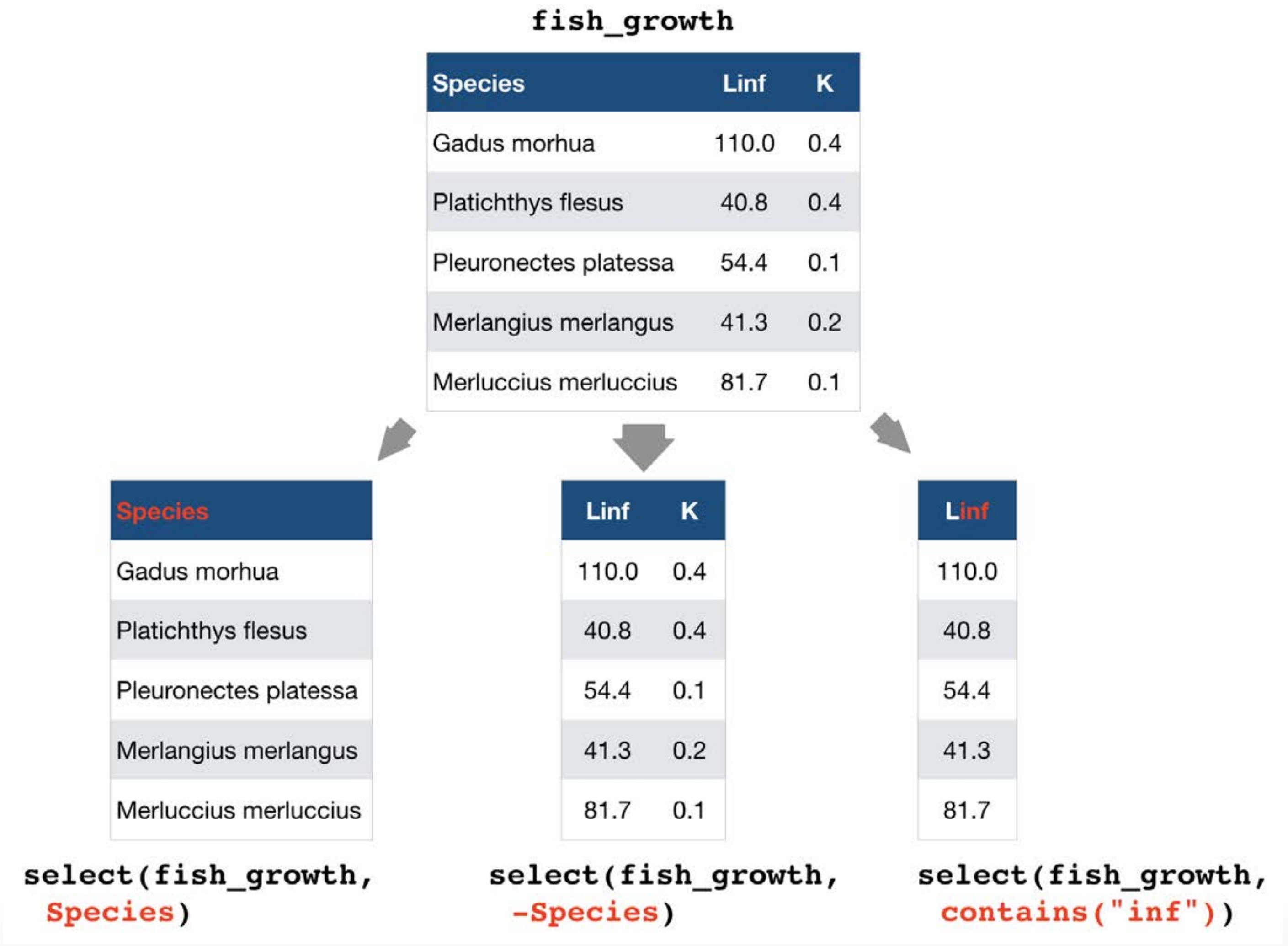
Species	Linf	K
Platichthys flesus	40.8	0.4
Merlangius merlangus	41.3	0.2
Pleuronectes platessa	54.4	0.1
Merluccius merluccius	81.7	0.1
Gadus morhua	110.0	0.4

```
arrange(fish_growth, Linf)
```

Species	Linf	K
Platichthys flesus	40.8	0.4
Gadus morhua	110.0	0.4
Merlangius merlangus	41.3	0.2
Pleuronectes platessa	54.4	0.1
Merluccius merluccius	81.7	0.1

```
arrange(fish_growth,
 K, desc(Linf))
```

`select()` → extract columns by name or helper function


fish_growth

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4
Pleuronectes platessa	54.4	0.1
Merlangius merlangus	41.3	0.2
Merluccius merluccius	81.7	0.1

Species
Gadus morhua
Platichthys flesus
Pleuronectes platessa
Merlangius merlangus
Merluccius merluccius

```
select(fish_growth,
 Species)
```

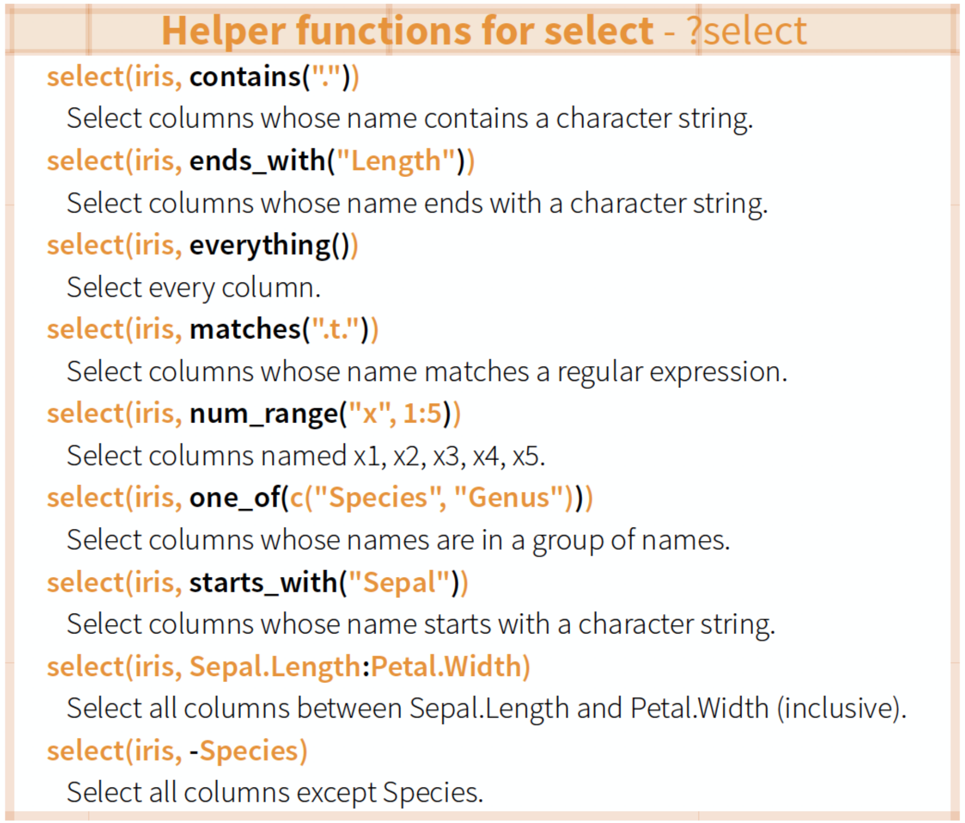
Linf	K
110.0	0.4
40.8	0.4
54.4	0.1
41.3	0.2
81.7	0.1

```
select(fish_growth,
 -Species)
```

Linf
110.0
40.8
54.4
41.3
81.7

```
select(fish_growth,
 contains("inf"))
```

Overview of helper functions



Helper functions for select - ?select

```
select(iris, contains("."))
```
Select columns whose name contains a character string.

```
select(iris, ends_with("Length"))
```
Select columns whose name ends with a character string.

```
select(iris, everything())
```
Select every column.

```
select(iris, matches(".t."))
```
Select columns whose name matches a regular expression.

```
select(iris, num_range("x", 1:5))
```
Select columns named x1, x2, x3, x4, x5.

```
select(iris, one_of(c("Species", "Genus")))
```
Select columns whose names are in a group of names.

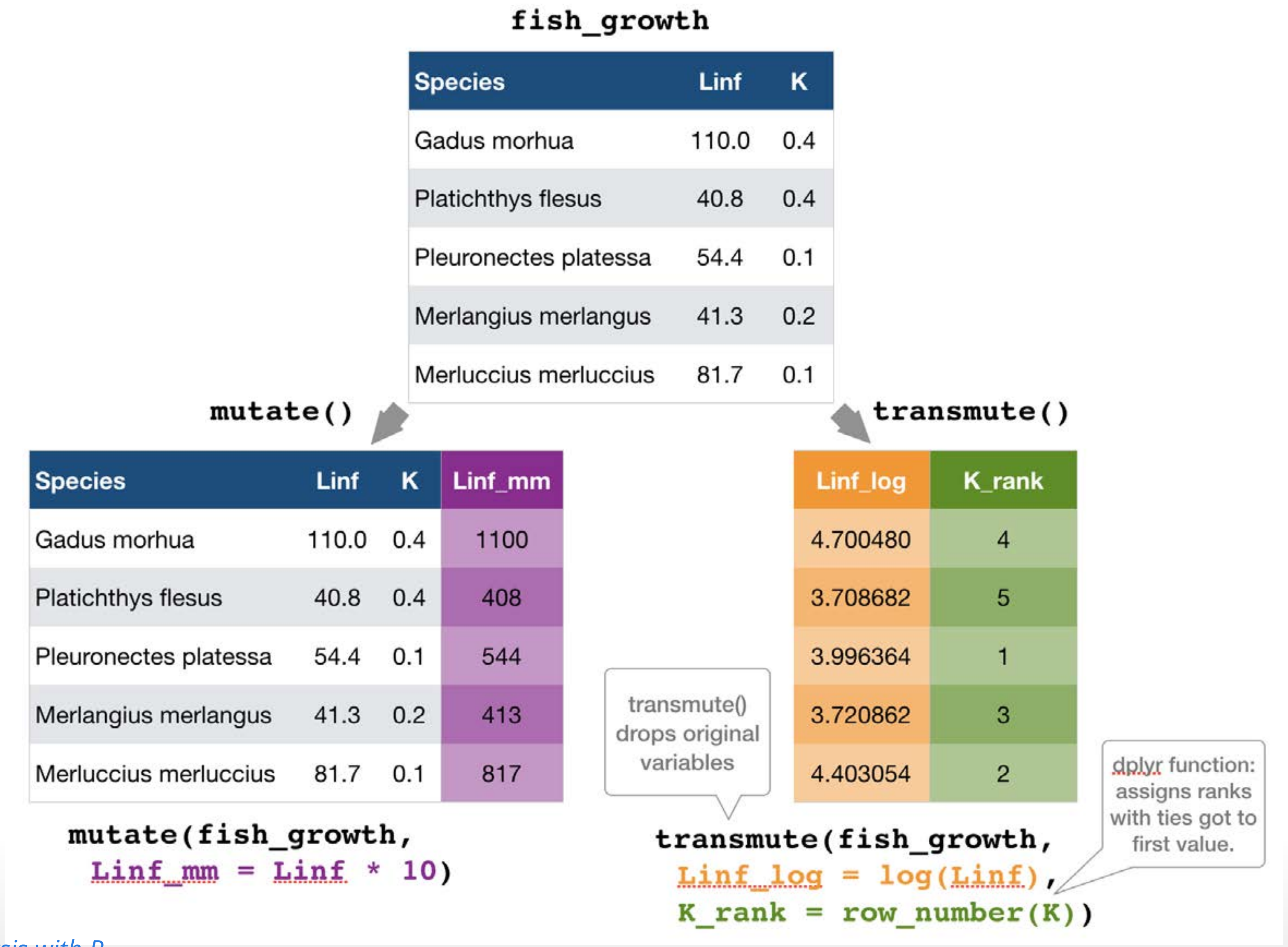
```
select(iris, starts_with("Sepal"))
```
Select columns whose name starts with a character string.

```
select(iris, Sepal.Length:Petal.Width)
```
Select all columns between Sepal.Length and Petal.Width (inclusive).

```
select(iris, -Species)
```
Select all columns except Species.

source: older version of Data Transformation with dplyr cheat sheet (licensed under CC-BY-SA)

`mutate()` and `transmute()` → create new variables


fish_growth
mutate()
transmute()
transmute() drops original variables
dplyr function: assigns ranks with ties got to first value.

fish_growth

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4
Pleuronectes platessa	54.4	0.1
Merlangius merlangus	41.3	0.2
Merluccius merluccius	81.7	0.1

mutate()

Species	Linf	K	Linf_mm
Gadus morhua	110.0	0.4	1100
Platichthys flesus	40.8	0.4	408
Pleuronectes platessa	54.4	0.1	544
Merlangius merlangus	41.3	0.2	413
Merluccius merluccius	81.7	0.1	817

```
mutate(fish_growth,
  Linf_mm = Linf * 10)
```

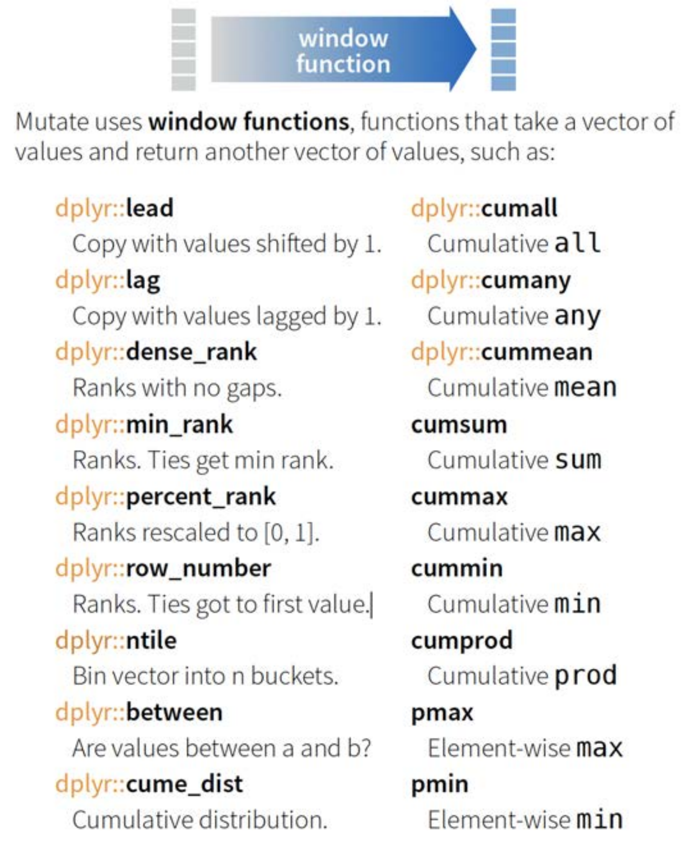
transmute()

Linf_log	K_rank
4.700480	4
3.708682	5
3.996364	1
3.720862	3
4.403054	2

```
transmute(fish_growth,
  Linf_log = log(Linf),
  K_rank = row_number(K))
```

`mutate()` and `transmute()`

You can do any calculation with a variable or apply a so-called **window function**:

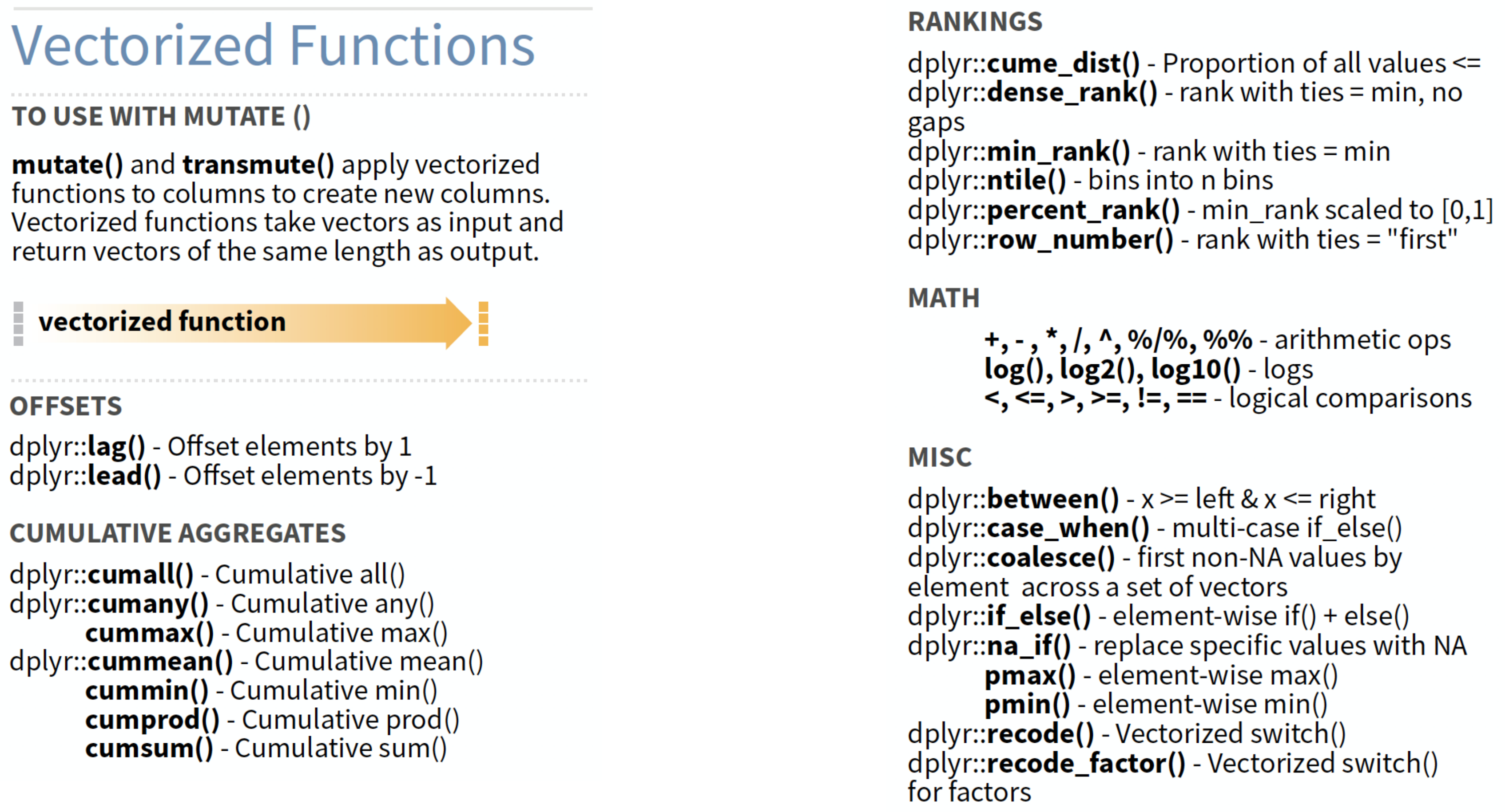

window function

Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

dplyr::**lead**	Copy with values shifted by 1.
dplyr::**lag**	Copy with values lagged by 1.
dplyr::**dense_rank**	Ranks with no gaps.
dplyr::**min_rank**	Ranks. Ties get min rank.
dplyr::**percent_rank**	Ranks rescaled to [0, 1].
dplyr::**row_number**	Ranks. Ties got to first value.
dplyr::**ntile**	Bin vector into n buckets.
dplyr::**between**	Are values between a and b?
dplyr::**cume_dist**	Cumulative distribution.
dplyr::**cumall**	Cumulative `all`
dplyr::**cumany**	Cumulative `any`
dplyr::**cummean**	Cumulative `mean`
cumsum	Cumulative `sum`
cummax	Cumulative `max`
cummin	Cumulative `min`
cumprod	Cumulative `prod`
pmax	Element-wise `max`
pmin	Element-wise `min`

source: older version of Data Transformation with dplyr cheat sheet (licensed under CC-BY-SA)

You can do any calculation with a variable as long as it is vectorized. Useful functions are:



Vectorized Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons

MISC

dplyr::**between()** - x >= left & x <= right
dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

source: older version of Data Transformation with dplyr cheat sheet (licensed under CC-BY-SA)

`summarise()` → reduce variables to *values*

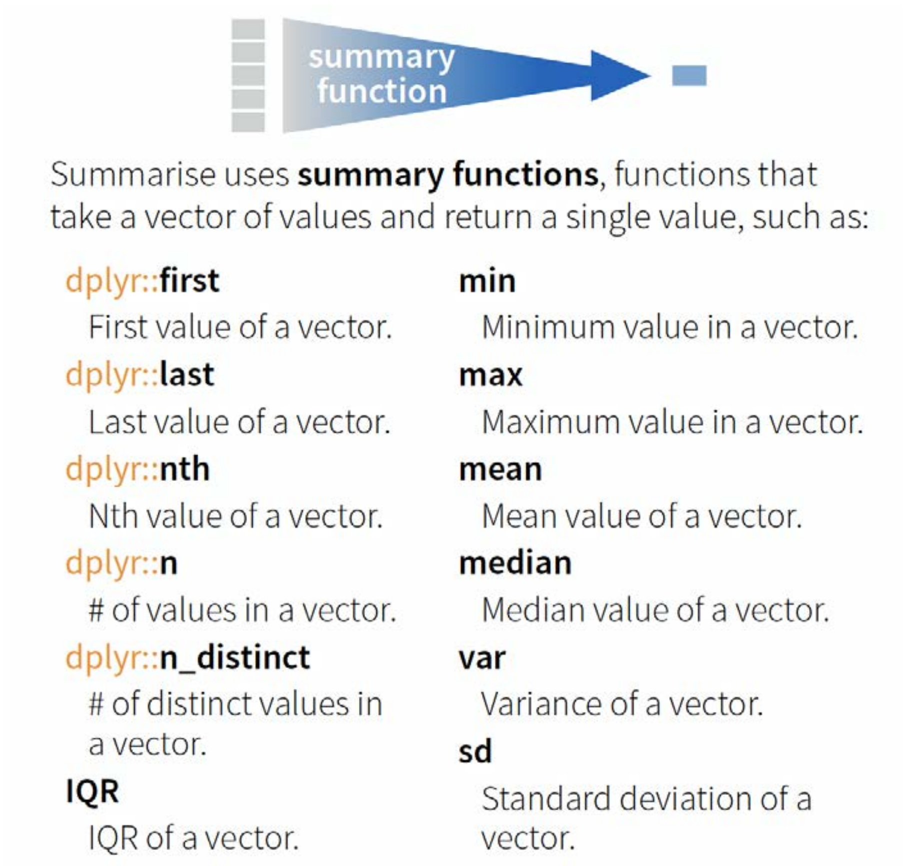
`fish_growth`

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4
Pleuronectes platessa	54.4	0.1
Merlangius merlangus	41.3	0.2
Merluccius merluccius	81.7	0.1

Linf_mean	Linf_min	Linf_max	K_mean
65.64	40.8	110	0.24

```
summarise(fish_growth,
  Linf_mean = mean(Linf),
  Linf_min = min(Linf),
  Linf_max = max(Linf),
  K_mean = mean(K)
)
```

Useful summary functions


summary function

Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

dplyr::**first** First value of a vector.	**min** Minimum value in a vector.
dplyr::**last** Last value of a vector.	**max** Maximum value in a vector.
dplyr::**nth** Nth value of a vector.	**mean** Mean value of a vector.
dplyr::**n** # of values in a vector.	**median** Median value of a vector.
dplyr::**n_distinct** # of distinct values in a vector.	**var** Variance of a vector.
IQR IQR of a vector.	**sd** Standard deviation of a vector.

source: older version of Data Transformation with dplyr cheat sheet (licensed under CC-BY-SA)

Your turn...

Import the oceanographic ICES dataset

If you haven't done it before in the date handling section do it now (don't forget to set the working directory beforehand!):

```
hydro <- read_csv("data/1111473b.csv")
names(hydro) <- c("cruise", "station", "type",
  "date_time", "lat", "long", "depth",
  "pres", "temp", "psal", "doxy")
```

Extract from the **date_time** variable the **year**, **month**, and **day** and save them in separate variables.

Quiz 5: Data manipulation

- Create a subset by **filtering** month *7* and pres *1*.
- **Select** from this subset only the *cruise*, *station*, and *day* variables.
- **Arrange** this subset now by *day*, then by *station*, and then by *cruise*.

Questions (solution code will be at the end of the presentation):

1. How many stations were sampled on day 2?

2. And how many cruises sampled these stations?

Submit Show Hint Show Answer Clear

Quiz 6: Data manipulation

Lets try a different approach to a similar question (code is at the end of slides):

- Create a subset by **filtering** month *2*, day *4*, and pres *1*.
- **Select** from this subset only the *cruise* and *station* variables (this step could also be skipped).
- **Summarise** the *cruise* and *station* variables by calculating the number of unique values: `n_distinct()`.

1. How many stations were sampled on day 4?

2. And how many cruises sampled these stations?

Submit Show Hint Show Answer Clear

Well done! You managed to calculate the number of sampled stations and cruises for a **single day**! But what about all the other days?

Devise a strategy for all days or months!

Try to get something like this

```
## # A tibble: 223 x 4
##   month   day cruise_count station_count
##   <dbl> <int>        <int>         <int>
## 1     1    12            2             4
## 2     1    13            1             2
## 3     1    14            1             2
## 4     1    15            1             1
## 5     1    19            3             9
## 6     1    20            3             7
## 7     1    21            3             5
## # ... with 216 more rows
```

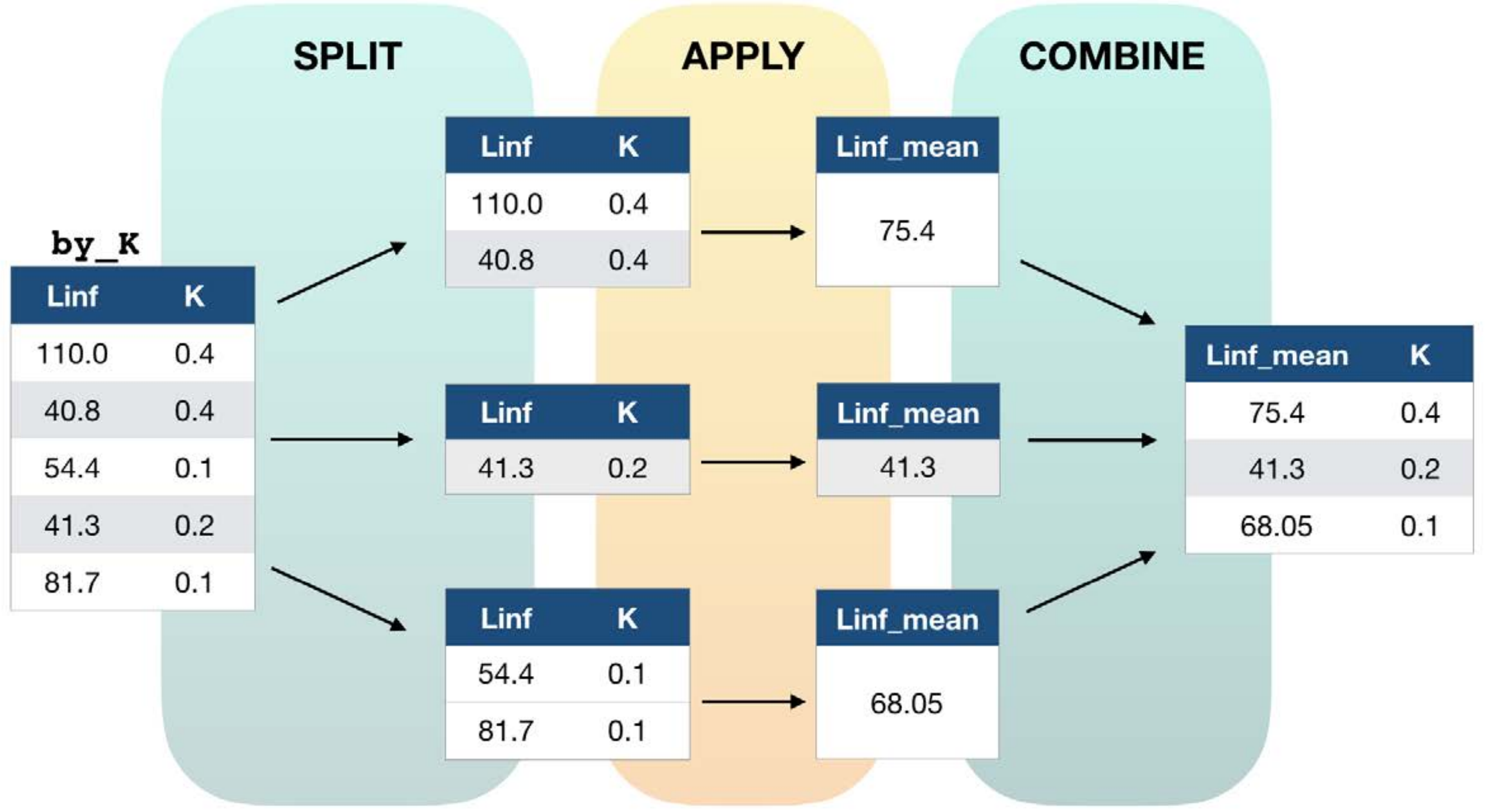
You get 2 minutes to think of a strategy

...

Solution for group-wise operations:

- `group_by()` takes an existing tbl and converts it into a grouped tbl where operations are performed "by group"
- `ungroup()` removes grouping

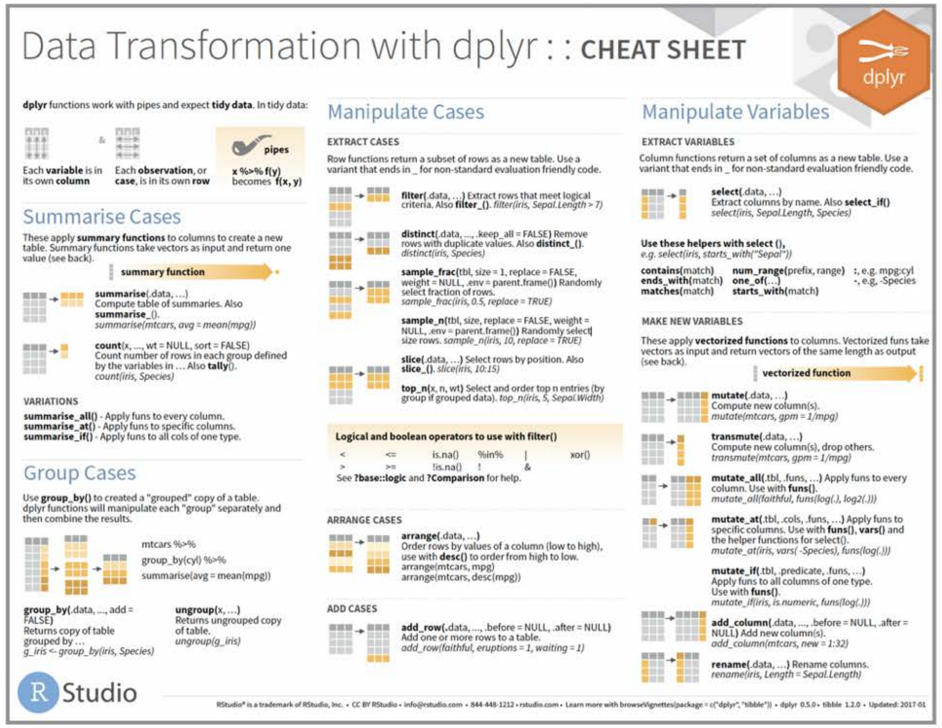
Principle of group-wise operations



```
group_by(by_K, K)
 |
 └──→ summarise(by_K, Linf_mean = mean(Linf))
```

dplyr offers many more functions!

From now on you should constantly look into the cheat sheet:



Data Transformation with dplyr : : **CHEAT SHEET**

dplyr

dplyr functions work with pipes and expect **tidy data**. In tidy data:

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

pipes **x %>% f(y)** becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries. Also **summarise_()**.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS
summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to created a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

filter(.data, …) Extract rows that meet logical criteria. Also **filter_()**. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values. Also **distinct_()**. *distinct(iris, Species)*

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows. *sample_frac(iris, 0.5, replace = TRUE)*

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position. Also **slice_()**. *slice(iris, 10:15)*

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …)
Order rows by values of a column (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

select(.data, …)
Extract columns by name. Also **select_if()**
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match) **num_range**(prefix, range) :, e.g. mpg:cyl
ends_with(match) **one_of**(…) -, e.g, -Species
matches(match) **starts_with**(match)

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**.
mutate_all(faithful, funs(log(.), log2(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

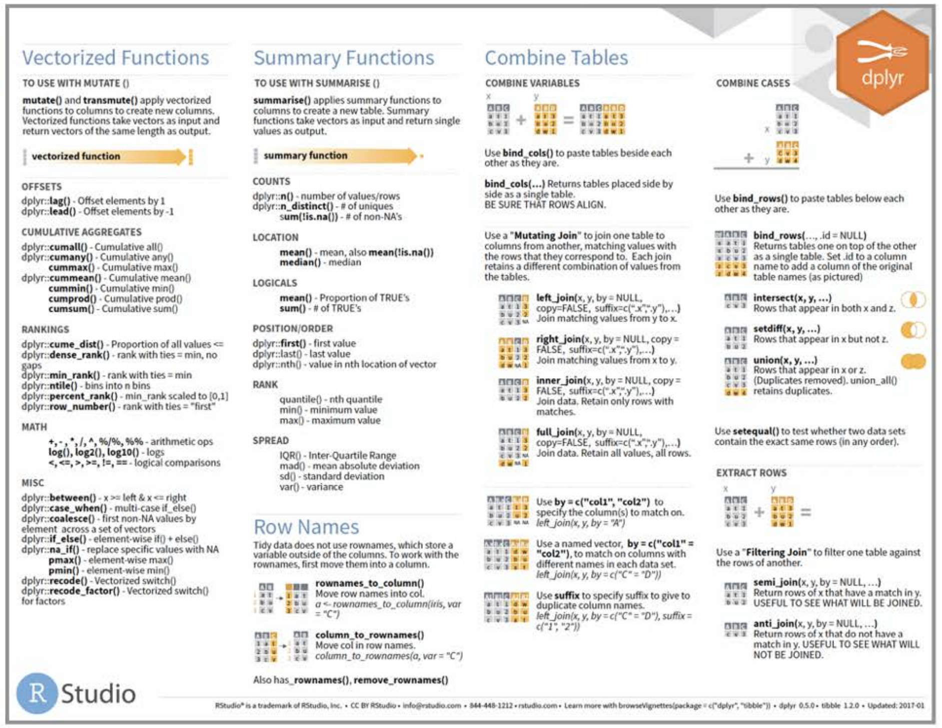
mutate_if(.tbl, .predicate, .funs, …)
Apply funs to all columns of one type. Use with **funs()**.
mutate_if(iris, is.numeric, funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s).
add_column(mtcars, new = 1:32)

rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.5.0 • tibble 1.2.0 • Updated: 2017-01



Vectorized Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS
dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES
dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS
dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH
+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons

MISC
dplyr::**between()** - x >= left & x <= right
dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS
dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION
mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS
mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER
dplyr::**first()** - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK
quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD
IQR() - Inter-Quartile Range
mad() - mean absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also has_**rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a **"Mutating Join"** to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…) Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…) Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…) Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…) Join data. Retain all values, all rows.

Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(…, .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, …)
Rows that appear in both x and z.

setdiff(x, y, …)
Rows that appear in x but not z.

union(x, y, …)
Rows that appear in x or z. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use a **"Filtering Join"** to filter one table against the rows of another.

semi_join(x, y, by = NULL, …)
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, …)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.5.0 • tibble 1.2.0 • Updated: 2017-01

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

Before you can practice your data manipulation skills you will get to know **one very usefool tool** for more complex operations!!!

The pipe operator

Basic piping with %>%

- The so-called pipe-operator is provided by the **magritr** package.
- Is part of the **core** tidyverse so you only need to install 'tidyverse' or any of the tidyverse core packages.
- **Simplifies operations**!
- Imagine taking the square root of the sums of squares of a data subset in **one step**:

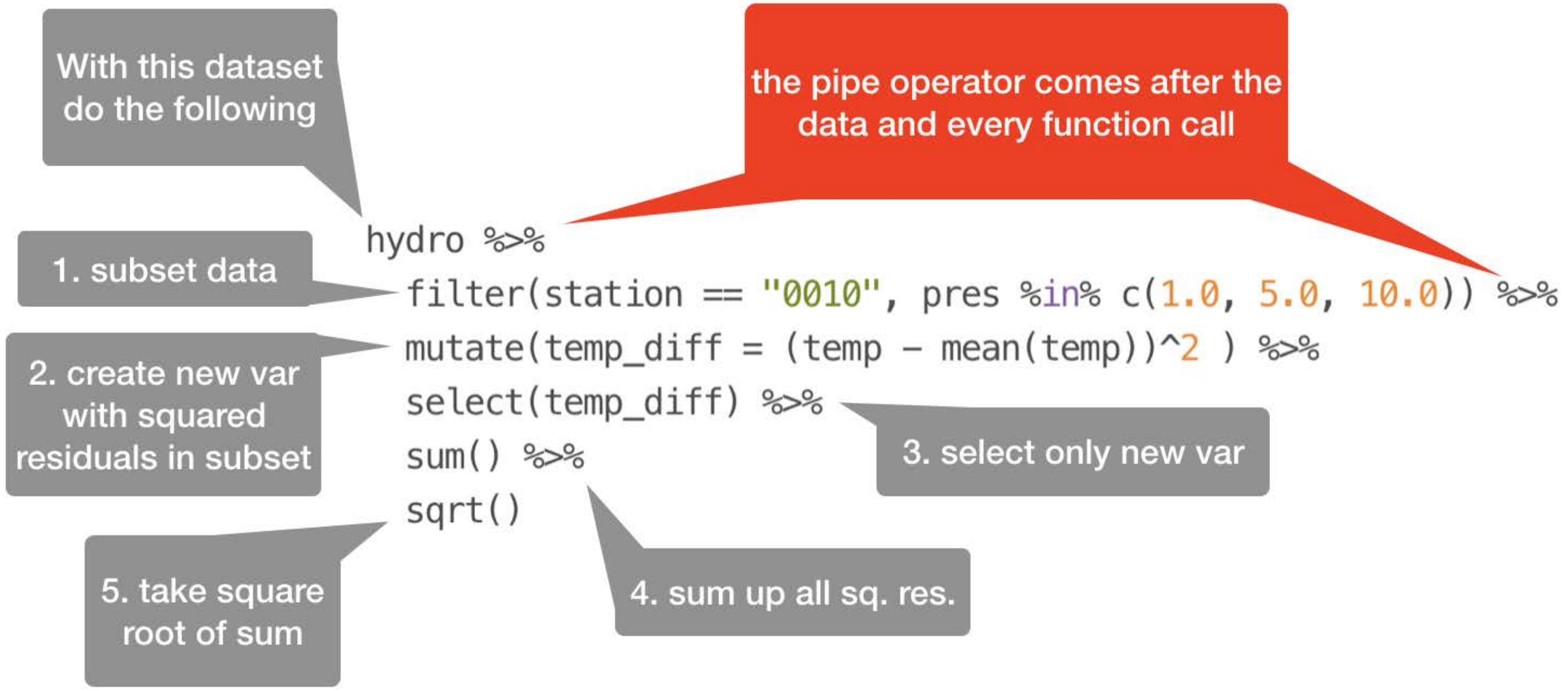
Basic piping with %>%

- The so-called pipe-operator is provided by the **magritr** package.
- Is part of the **core** tidyverse so you only need to install 'tidyverse' or any of the tidyverse core packages.
- **Simplifies operations**!
- Imagine taking the square root of the sums of squares of a data subset in **one step**:

```r
sqrt(sum( (hydro$temp[hydro$station == "0010" &  hydro$pres %in% c(1,5,10)] -
    mean(hydro$temp[hydro$station == "0010" &  hydro$pres %in% c(1,5,10)]))^2))
```

Does that look simple and readable?

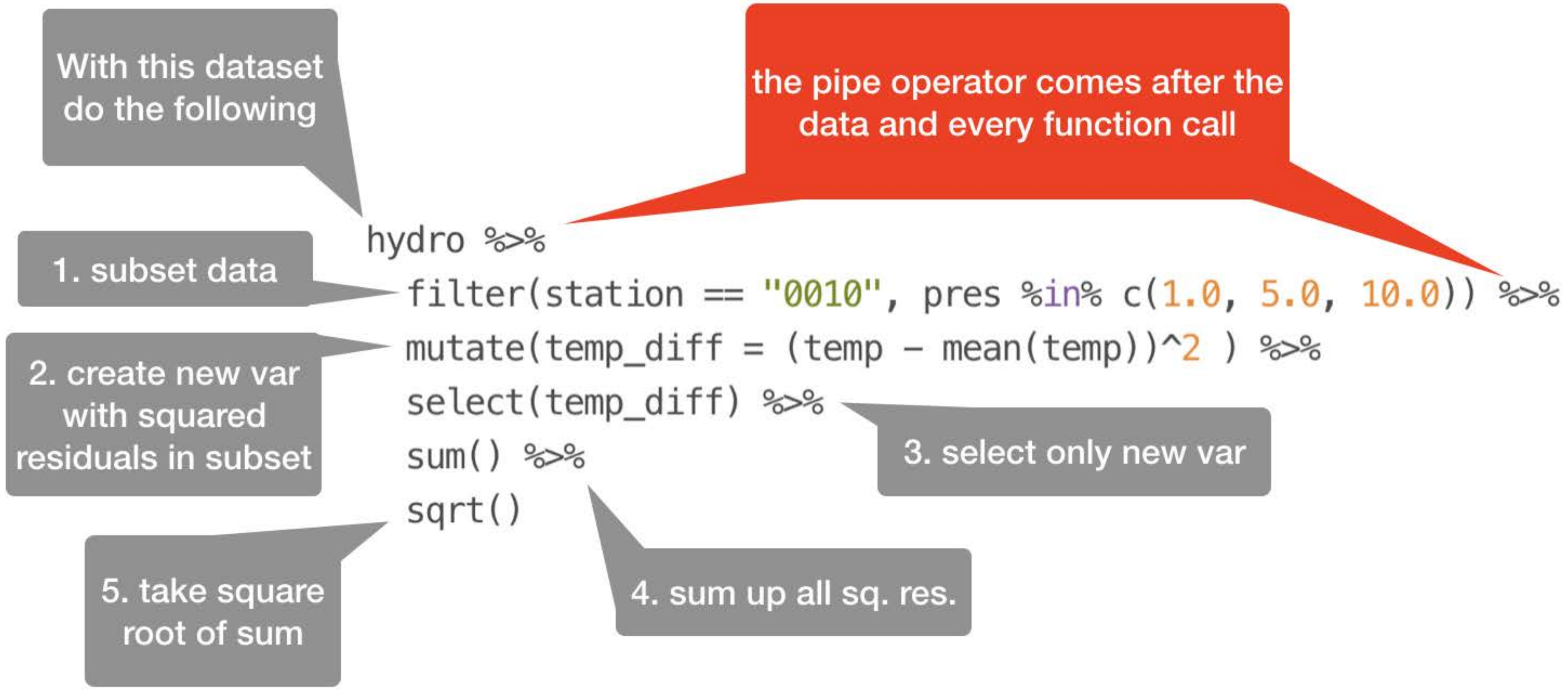
With `%>%` you can couple several function calls sequentially without creating many intermediate objects:



```
hydro %>%
  filter(station == "0010", pres %in% c(1.0, 5.0, 10.0)) %>%
  mutate(temp_diff = (temp - mean(temp))^2 ) %>%
  select(temp_diff) %>%
  sum() %>%
  sqrt()
```

With this dataset do the following
the pipe operator comes after the data and every function call
1. subset data
2. create new var with squared residuals in subset
3. select only new var
4. sum up all sq. res.
5. take square root of sum

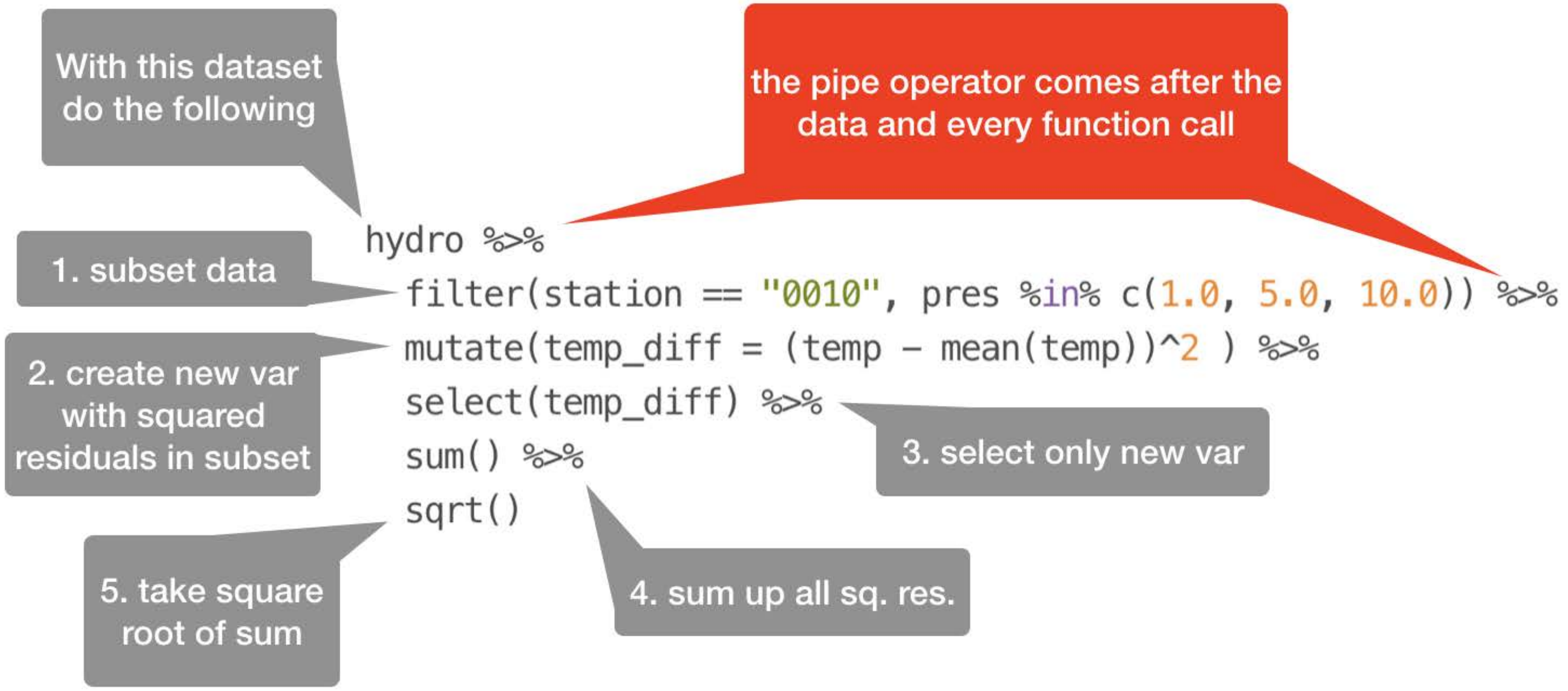
With `%>%` you can couple several function calls sequentially without creating many intermediate objects:


With this dataset do the following
the pipe operator comes after the data and every function call
1. subset data
2. create new var with squared residuals in subset
3. select only new var
4. sum up all sq. res.
5. take square root of sum

```r
hydro %>%
  filter(station == "0010", pres %in% c(1.0, 5.0, 10.0)) %>%
  mutate(temp_diff = (temp - mean(temp))^2 ) %>%
  select(temp_diff) %>%
  sum() %>%
  sqrt()
```

- `%>%` pipes left-hand side values forward into expressions that appear on the right-hand side.

With `%>%` you can couple several function calls sequentially without creating many intermediate objects:


With this dataset do the following
the pipe operator comes after the data and every function call
1. subset data
2. create new var with squared residuals in subset
3. select only new var
4. sum up all sq. res.
5. take square root of sum

```r
hydro %>%
  filter(station == "0010", pres %in% c(1.0, 5.0, 10.0)) %>%
  mutate(temp_diff = (temp - mean(temp))^2 ) %>%
  select(temp_diff) %>%
  sum() %>%
  sqrt()
```

- `%>%` pipes left-hand side values forward into expressions that appear on the right-hand side.
- Additional steps can be easily added anywhere in the sequence of operations.

Your turn...

Tell me ...

1. Which dplyr function can you use to remove duplicated row values?
2. And which dplyr function(s) can you use to count the number of rows in each variable group?

→ These functions can be helpful in the next data manipulation exercises!

More complex data manipulations

With the `group_by()` function and the pipe operator you will be able to answer the following questions (choose **at least 3** questions):

```
1. On average, how many stations were sampled per month during 2015?
2. Which stations were sampled more than 3 times per month?
3. How many days took the sampling place in each month?
4. Do you see any temporal gap during the year where no sampling took place?
5. Which depths are most frequently sampled?
6. What are the most common depth profiles taken? (Every 1 metre, every 5 metres?)
7. Are the NAs in the dataset related to specific months or cruises?

What else could be relevant in terms of data quality?
```

(the solution code is at the end of the presentation) base: `Sys.time()`, `unclass(Sys.time())`

lubridate: `ymd`, `mdy`, `dmy`, `ymd_hms`, `mdy_hm` `make_date`, `as_date()`, `as_datetime()` `year()`, `month()`, `mday()`, `yday()`, `wday()`, `hour()`, `minute()`, `second()`, `%--%`, `as.duration()`

dplyr: `filter()`, `arrange()`, `select()`, `mutate()` and `transmute()`, `summarise()`, `group_by()`, `ungroup()`

magrittr: `%>%`

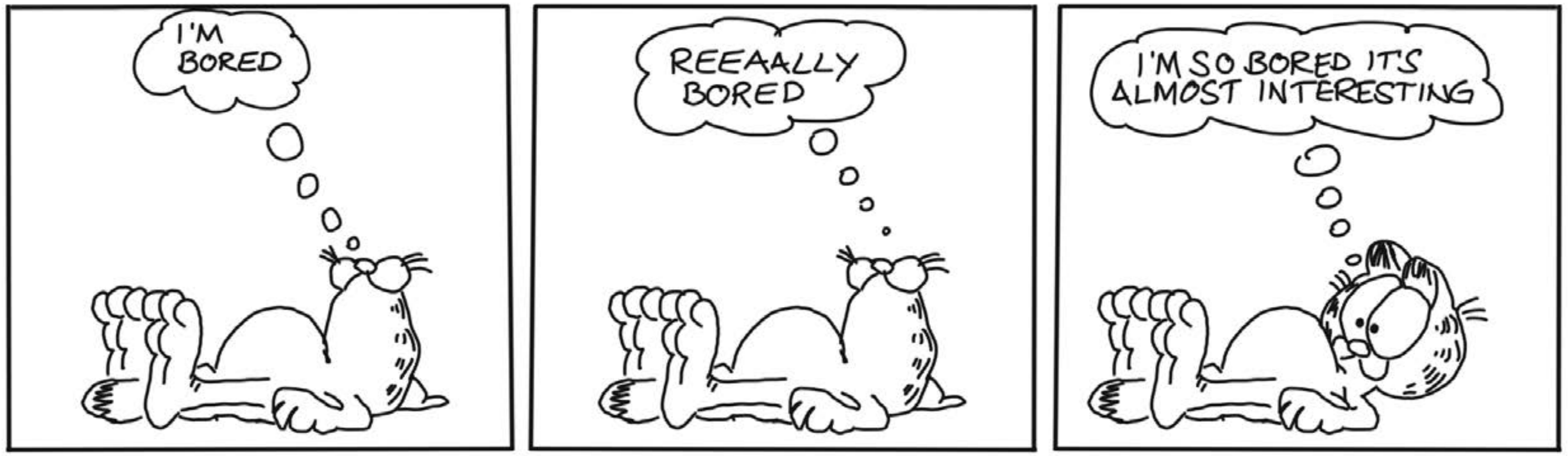
Overview of functions you learned today

How do you feel now.....?

Totally confused?

Tr out ALL the exercises and compare your code and results with the solution code! Read chapter 5 on data transformation, chapter 16 on dates and times, and chapter 18 on the pipe operator 'in R for Data Science'.

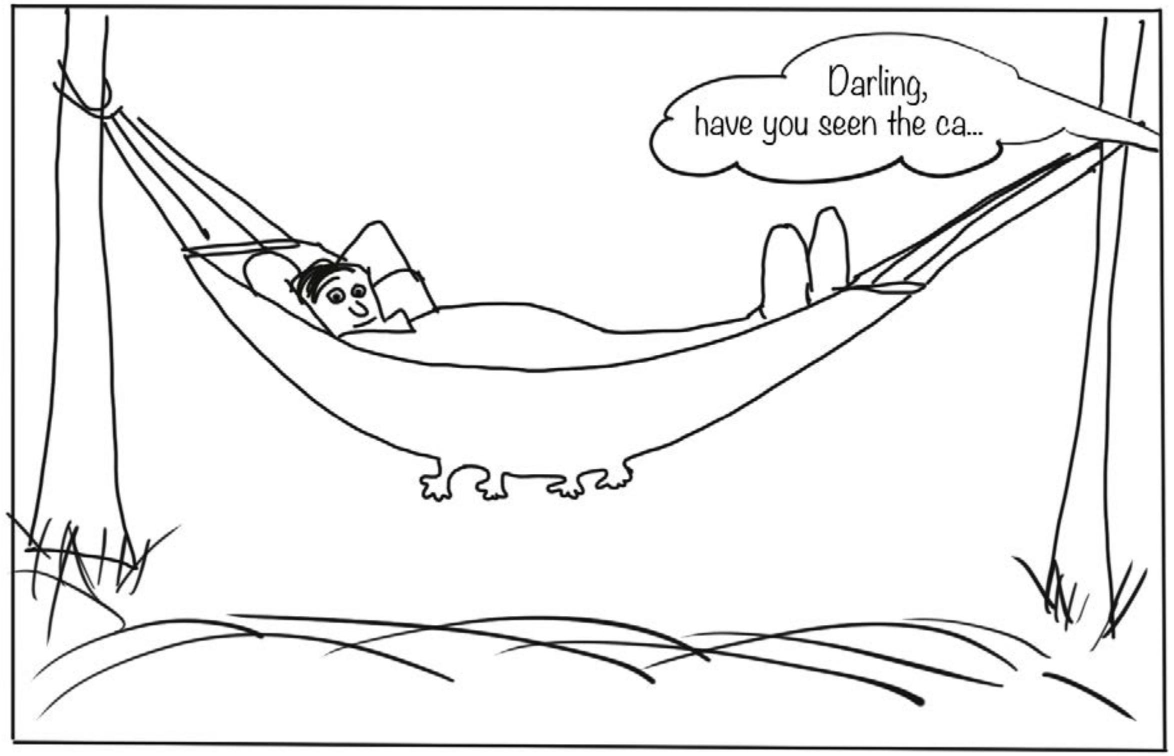
Totally bored?



Then play around with the full hydro dataset "1111473b.csv" and explore already the hydrographical variables.

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 5: Data manipulation

```r
h_filt <- filter(hydro, month == 7, pres == 1)
h_sel <- select(h_filt, cruise,station, day)
h_arr <- arrange(h_sel, day, station, cruise)
# View(h_arr) or filter by day
out <- filter(h_sel, day == 2)
```

```r
out
```

```
## # A tibble: 6 x 3
##   cruise station   day
##   <chr>  <chr>   <int>
## 1 3490   0093        2
## 2 3490   0257        2
## 3 3490   0229        2
## 4 ESLV   012c        2
## 5 ESLV   0038        2
## 6 ESLV   00N8        2
```

Using the **pipe operator**:

```r
hydro %>%
  filter(month == 7, pres == 1) %>%
  select(cruise, station, day) %>%
  arrange(day, station, cruise) %>%
  filter(day == 2)
```

Quiz 6: Data manipulation

```
h_filt <- filter(hydro, month == 2, day == 4, pres == 1)
summarise(h_filt, cruise_nr = n_distinct(cruise),
  station_nr = n_distinct(station) )
```

```
## # A tibble: 1 x 2
##   cruise_nr station_nr
##       <int>      <int>
## 1         2          8
```

Using the **pipe operator**:

```
hydro %>%
  filter(month == 2, day == 4, pres == 1) %>%
  summarise(cruise_count = n_distinct(cruise),
    station_count = n_distinct(station) )
```

Complex data manipulations - Question 1

On average, how many stations were sampled per month during 2015?

You want the number of sampled stations per month before you can calculate the mean. This could be done by counting the number of rows with different station values per month. Problem: The data consists of double entries (duplicated station values) due to the different sampling depths (and maybe the station was sampled more than once at the same day during the cruise). So first remove double entries by using the `distinct()` function!

Complex data manipulations - Question 1 (cont)

On average, how many stations were sampled per month during 2015?

```r
hydro %>%
  select(station, month) %>%
  # to remove duplicates
  distinct() %>%
  group_by(month) %>%
  count() %>%
  ungroup() %>%
  summarise(stat_per_month = mean(n))
```

```
## # A tibble: 1 x 1
##   stat_per_month
##            <dbl>
## 1           210.
```

As we are only interested in stations per month and not in double entries, etc.,

- both variables are first selected
- and then duplicated entries removed.
- The dataset is grouped by month,
- number of rows per month (= the stations) calculated and
- the mean across months computed.

Complex data manipulations - Question 2

Which stations were sampled more than 3 times per month?

```
hydro %>%
  select(station,date_time,month) %>%
  distinct() %>%
  group_by(month, station) %>%
  count() %>%
  filter(n > 3)
```

```
## # A tibble: 9 x 3
## # Groups:   month, station [9]
##   month station     n
##   <dbl> <chr>   <int>
## 1     3 0001        4
## 2     3 0021        4
## 3     3 0023        4
## 4     3 0025        4
## 5     3 0027        4
## 6     3 0029        4
## 7     4 0002        4
## 8     4 0003        5
## 9     6 0038        4
```

- date_time is kept in to indicate the nr of samplings at this station per month
- instead of *count()* you can also use *summarise(n = length(station))* or *summarise(n = n())*

Complex data manipulations - Question 3

How many days took the sampling place in each month?

What are you interested in here? In *day* and *month*, so select only those 2 variables, remove duplicated rows, group by month so that you can count the number of rows with different day values:

```
hydro %>%
  select(month, day) %>%
  distinct() %>%
  group_by(month) %>%
  summarise(n = n()) # or count()
```

```
## # A tibble: 12 x 2
##    month     n
##    <dbl> <int>
##  1     1    19
##  2     2    26
##  3     3    31
##  4     4    17
##  5     5    24
##  6     6    26
##  7     7    25
##  8     8    25
##  9     9    28
## 10    10    29
## 11    11    27
## 12    12    13
```

Complex data manipulations - Question 4

Do you see any temporal gap during the year where no sampling took place?

This is a question were you can play around with various other functions. No approach will be the correct one. Here is one solution where the julian days are computed with the lubridate function `yday()` and the difference between successive julian days then calculated:

```
hydro %>%
  mutate(julian_day = lubridate::yday(date_time)) %>%
  select(julian_day, month) %>%
  distinct() %>%
  arrange(julian_day) %>%
  mutate( timegap = c(NA, diff(julian_day)) ) %>%
  group_by(month) %>%
  filter(timegap > 3)
```

Complex data manipulations - Question 4 (cont)

Do you see any temporal gap during the year where no sampling took place?

```
## # A tibble: 8 x 3
## # Groups:   month [6]
##   julian_day month timegap
##        <dbl> <dbl>   <dbl>
## 1         19     1       4
## 2         97     4       6
## 3        103     4       4
## 4        110     4       4
## 5        124     5       6
## 6        180     6       4
## 7        306    11       4
## 8        341    12       4
```

So mainly April shows the greatest gaps (with a gap of 6 days, and twice of 4 days). Why could that be?

Complex data manipulations - Question 5 and 6

Which depths are most frequently sampled? What are the most common depth profiles taken? (Every 1 metre, every 5 metres?)

```r
hydro %>%
 select(pres) %>%
 group_by(pres) %>%
 count() %>%
 arrange(desc(n)) %>% print(n=3)
```

```
## # A tibble: 1,193 x 2
## # Groups:   pres [1,193]
##     pres     n
##    <dbl> <int>
## 1      5  2319
## 2     10  2215
## 3     20  1792
## # ... with 1,190 more rows
```

If you got the same result you probably noted, that the depth (or pres) values are not integers and the number of unique values is there fore very high (1,193). To reduce the numer of depth levels we could round them first. Instead of using the function `round()` I suggest using `ceiling()`, which rounds to the next higher integer (so that 0.4m is considered 1m):

Complex data manipulations - Q5 and 6 (cont)

Which depths are most frequently sampled? What are the most common depth profiles taken? (Every 1 metre, every 5 metres?)

```
hydro %>%
transmute(pres2 = ceiling(pres)) %>%
 group_by(pres2) %>%
 count() %>%
 arrange(desc(n)) %>% print(n=3)
```

```
## # A tibble: 216 x 2
## # Groups:   pres2 [216]
##   pres2     n
##   <dbl> <int>
## 1     5  2370
## 2    10  2232
## 3     1  1880
## # ... with 213 more rows
```

- Q5: The depths most often sampled are 5m, 10m, and 1m.
- Q6: From 0 to 30m depth samples were mostly taken in 5m intervals (1, 5, 10, 15, 20, 25, 30m) depth and afterwards mostly in 10m intervals.

Complex data manipulations - Question 7

Are the NAs in the dataset related to specific months or cruises?

Check if related to months

```r
hydro %>%
 select(month, temp, psal, doxy) %>%
 group_by(month) %>%
 summarise(
  t_na = sum(is.na(temp)),
  s_na = sum(is.na(psal)),
  o_na = sum(is.na(doxy))
 ) %>%
 mutate(sum_na = t_na+s_na+o_na) %>%
 arrange(desc(sum_na))
```

```
## # A tibble: 12 x 5
##    month  t_na  s_na  o_na sum_na
##    <dbl> <int> <int> <int>  <int>
##  1    10   184   234  1115   1533
##  2     8   310   413   630   1353
##  3     3   115   108   854   1077
##  4     6   155   267   649   1071
##  5     9   123   232   619    974
##  6     7   239   357   372    968
##  7     2    47    82   798    927
##  8    11    46    83   764    893
##  9     5   235   311   330    876
## 10     4   177   200   375    752
## 11     1    73    85   549    707
## 12    12    10    10   249    269
```

Complex data manipulations - Question 7 (cont)

Are the NAs in the dataset related to specific months or cruises?

Check if related to cruises

```
hydro %>%
 select(cruise, temp, psal, doxy) %>%
 group_by(cruise) %>%
 summarise(
  t_na = sum(is.na(temp)),
  s_na = sum(is.na(psal)),
  o_na = sum(is.na(doxy))
 ) %>%
 mutate(sum_na = t_na+s_na+o_na) %>%
 arrange(desc(sum_na))
```

```
## # A tibble: 36 x 5
##    cruise  t_na  s_na  o_na sum_na
##    <chr>  <int> <int> <int>  <int>
##  1 67BC     127   147  2300   2574
##  2 ESSA     729   729   243   1701
##  3 3490     194   706   488   1388
##  4 26DA       0     2  1234   1236
##  5 34AR      17    17  1160   1194
##  6 77FY      39     0   490    529
##  7 67LL      92   272    92    456
##  8 ESLV     145   145   151    441
##  9 77K9       4     3   285    292
## 10 ESQT      59    59    89    207
## # ... with 26 more rows
```

Complex data manipulations - Question 7 (cont)

Are the NAs in the dataset related to specific months or cruises?

- NAs are most common in October and August but there is no clear seasonal pattern in the occurrence of NAs.
- Certain cruises provided data to ICES with many more missing values.
    - The NAs are mainly related to specific cruises, with the highest number of NAs found for oxygen.
    - It might be smart to go into the original data and check for those cruises if NAs occur only for specific depths.

```
hydro %>%
  filter(month==2,cruise=="67BC") %>%
  View()
```

→ at this cruise doxy was only taken in 10m depth intervals not in 5m as for temp and psal