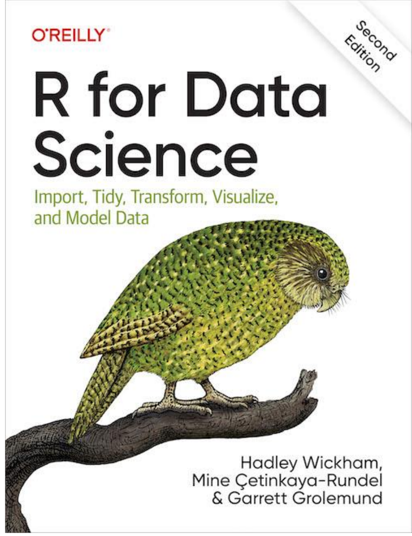
R II

Programmierkurs 2 Data Science WS24/25

Leonard Traeger
M. Sc. Information Systems
leonard.traeger@fh-dortmund.de

Disclaimer


O'REILLY
Second Edition
R for Data Science
Import, Tidy, Transform, Visualize, and Model Data
Hadley Wickham, Mine Çetinkaya-Rundel & Garrett Grolemund

Slides are mainly based on

- https://r4ds.hadley.nz/
- https://www.phonetik.uni-muenchen.de/~jmh/lehre/basic_r/_book/index.html

→ Find everything you need to know there!

Official R cheat sheet:

- https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Data Transformation with dplyr:

- https://raw.githubusercontent.com/rstudio/cheatsheets/master/data-transformation.pdf

Learning Goals R II

- **Deploy** functions in R with conditional transformations.
- **Explain** and **apply** logical summaries on logical vectors.
- **Describe** data types and **apply** string, numeric, and date vector transformations.
- **Construct** attributes with vector and data frame functions.
- **Explain** the concept of Embracing with sorting, aggregation, and grouping**.**

Make sure to enable them and resolve conflicts!

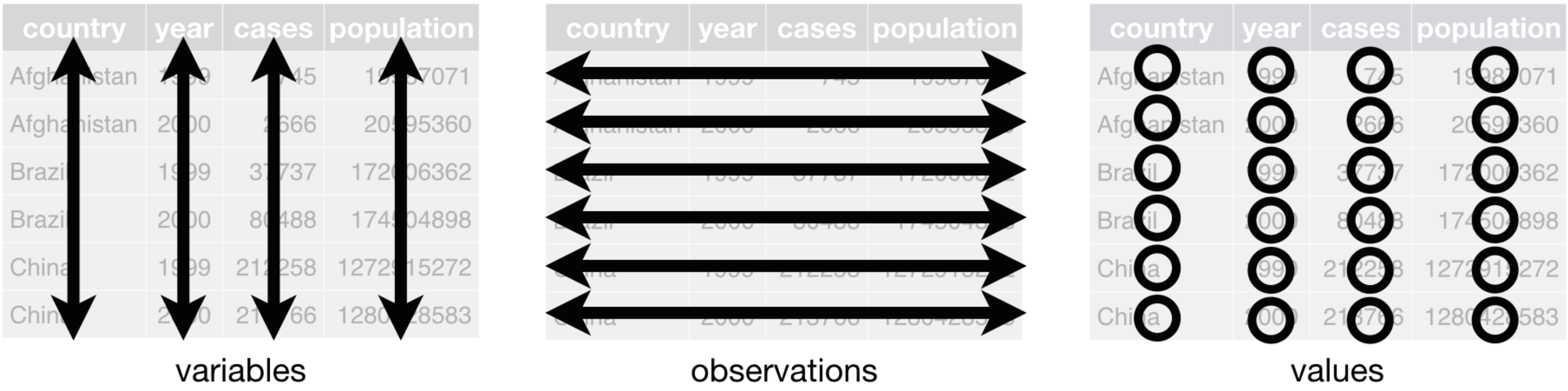
Libraries that we use

`readr`	Dataframe
`tidyverse`	Collection of packages (readr, dplyr, ggplot2, stringr,…)
`dplyr`	Data Manipulation
`stats`	Aggregation and many more
`magrittr`	Pipe Operator `%>%`

DataFrames (Recap)

An extremely important data structure in R (two-dimensional table).

- Rows also called observations.
- Columns also called variables (not to be confused with the variables from before!).

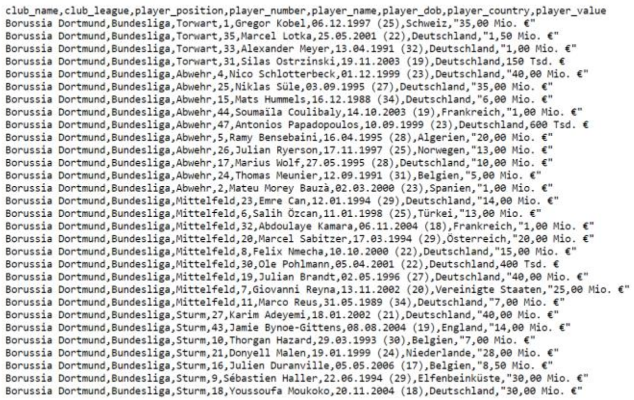


variables observations values

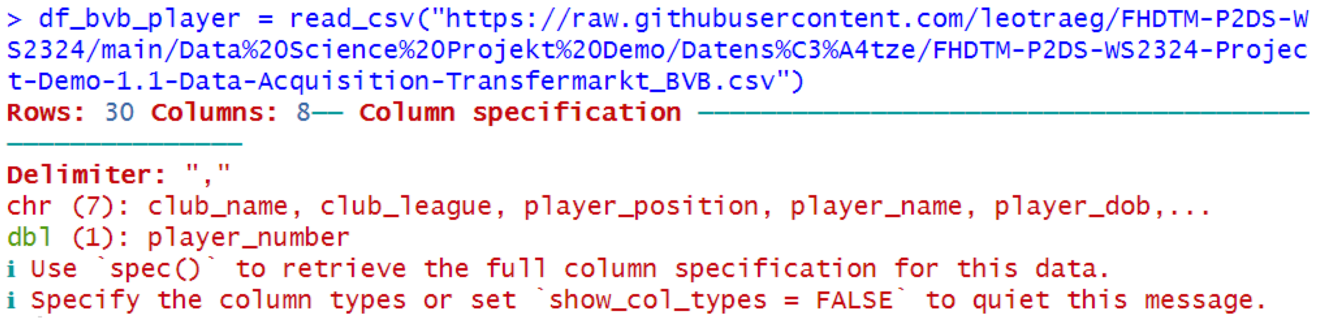
R for Data Science (e2) by Wickham, Çetinkaya-Rundel, and Grolemund

Reading data from .csv (Recap)

Read .csv files into R using `read_csv(path)` :



```
club_name,club_league,player_position,player_number,player_name,player_dob,player_country,player_value
Borussia Dortmund,Bundesliga,Torwart,1,Gregor Kobel,06.12.1997 (25),Schweiz,"35,00 Mio. €"
Borussia Dortmund,Bundesliga,Torwart,35,Marcel Lotka,25.05.2001 (22),Deutschland,"1,50 Mio. €"
Borussia Dortmund,Bundesliga,Torwart,33,Alexander Meyer,13.04.1991 (32),Deutschland,"1,00 Mio. €"
Borussia Dortmund,Bundesliga,Torwart,31,Silas Ostrzinski,19.11.2003 (19),Deutschland,150 Tsd. €
Borussia Dortmund,Bundesliga,Abwehr,4,Nico Schlotterbeck,01.12.1999 (23),Deutschland,"40,00 Mio. €"
Borussia Dortmund,Bundesliga,Abwehr,25,Niklas Süle,03.09.1995 (27),Deutschland,"35,00 Mio. €"
Borussia Dortmund,Bundesliga,Abwehr,15,Mats Hummels,16.12.1988 (34),Deutschland,"6,00 Mio. €"
Borussia Dortmund,Bundesliga,Abwehr,44,Soumaïla Coulibaly,14.10.2003 (19),Frankreich,"1,00 Mio. €"
Borussia Dortmund,Bundesliga,Abwehr,47,Antonios Papadopoulos,10.09.1999 (23),Deutschland,600 Tsd. €
Borussia Dortmund,Bundesliga,Abwehr,5,Ramy Bensebaini,16.04.1995 (28),Algerien,"20,00 Mio. €"
Borussia Dortmund,Bundesliga,Abwehr,26,Julian Ryerson,17.11.1997 (25),Norwegen,"13,00 Mio. €"
Borussia Dortmund,Bundesliga,Abwehr,17,Marius Wolf,27.05.1995 (28),Deutschland,"10,00 Mio. €"
Borussia Dortmund,Bundesliga,Abwehr,24,Thomas Meunier,12.09.1991 (31),Belgien,"5,00 Mio. €"
Borussia Dortmund,Bundesliga,Abwehr,2,Mateu Morey Bauzà,02.03.2000 (23),Spanien,"1,00 Mio. €"
Borussia Dortmund,Bundesliga,Mittelfeld,23,Emre Can,12.01.1994 (29),Deutschland,"14,00 Mio. €"
Borussia Dortmund,Bundesliga,Mittelfeld,6,Salih Özcan,11.01.1998 (25),Türkei,"13,00 Mio. €"
Borussia Dortmund,Bundesliga,Mittelfeld,32,Abdoulaye Kamara,06.11.2004 (18),Frankreich,"1,00 Mio. €"
Borussia Dortmund,Bundesliga,Mittelfeld,20,Marcel Sabitzer,17.03.1994 (29),Österreich,"20,00 Mio. €"
Borussia Dortmund,Bundesliga,Mittelfeld,8,Felix Nmecha,10.10.2000 (22),Deutschland,"15,00 Mio. €"
Borussia Dortmund,Bundesliga,Mittelfeld,30,Ole Pohlmann,05.04.2001 (22),Deutschland,400 Tsd. €
Borussia Dortmund,Bundesliga,Mittelfeld,19,Julian Brandt,02.05.1996 (27),Deutschland,"40,00 Mio. €"
Borussia Dortmund,Bundesliga,Mittelfeld,7,Giovanni Reyna,13.11.2002 (20),Vereinigte Staaten,"25,00 Mio. €"
Borussia Dortmund,Bundesliga,Mittelfeld,11,Marco Reus,31.05.1989 (34),Deutschland,"7,00 Mio. €"
Borussia Dortmund,Bundesliga,Sturm,27,Karim Adeyemi,18.01.2002 (21),Deutschland,"40,00 Mio. €"
Borussia Dortmund,Bundesliga,Sturm,43,Jamie Bynoe-Gittens,08.08.2004 (19),England,"14,00 Mio. €"
Borussia Dortmund,Bundesliga,Sturm,10,Thorgan Hazard,29.03.1993 (30),Belgien,"7,00 Mio. €"
Borussia Dortmund,Bundesliga,Sturm,21,Donyell Malen,19.01.1999 (24),Niederlande,"28,00 Mio. €"
Borussia Dortmund,Bundesliga,Sturm,16,Julien Duranville,05.05.2006 (17),Belgien,"8,50 Mio. €"
Borussia Dortmund,Bundesliga,Sturm,9,Sébastien Haller,22.06.1994 (29),Elfenbeinküste,"30,00 Mio. €"
Borussia Dortmund,Bundesliga,Sturm,18,Youssoufa Moukoko,20.11.2004 (18),Deutschland,"30,00 Mio. €"
```



```
> df_bvb_player = read_csv("https://raw.githubusercontent.com/leotraeg/FHDTM-P2DS-W
S2324/main/Data%20Science%20Projekt%20Demo/Datens%C3%A4tze/FHDTM-P2DS-WS2324-Projec
t-Demo-1.1-Data-Acquisition-Transfermarkt_BVB.csv")
Rows: 30 Columns: 8── Column specification ──────────────────────────────────────
──────────────
Delimiter: ","
chr (7): club_name, club_league, player_position, player_name, player_dob,...
dbl (1): player_number
ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

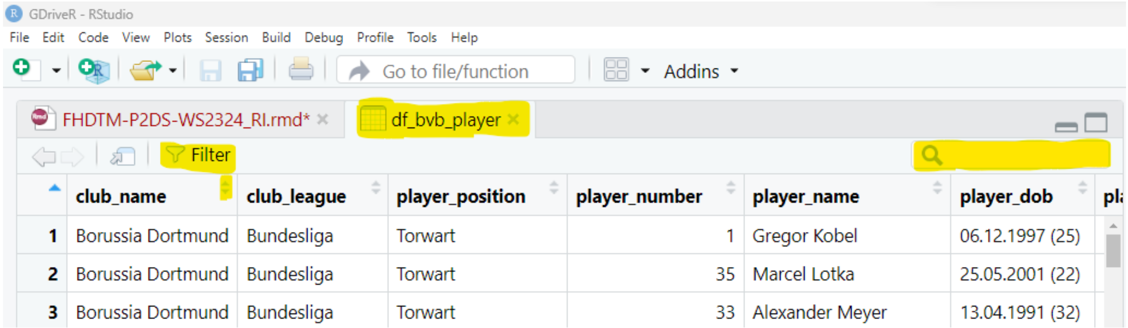
In successful .csv read; a log message tells you the

- Number of rows and columns.
- Delimiter in use.
- Column name and type specifications.

Viewing Meta Data

In R: Explore DataFrame (Recap)

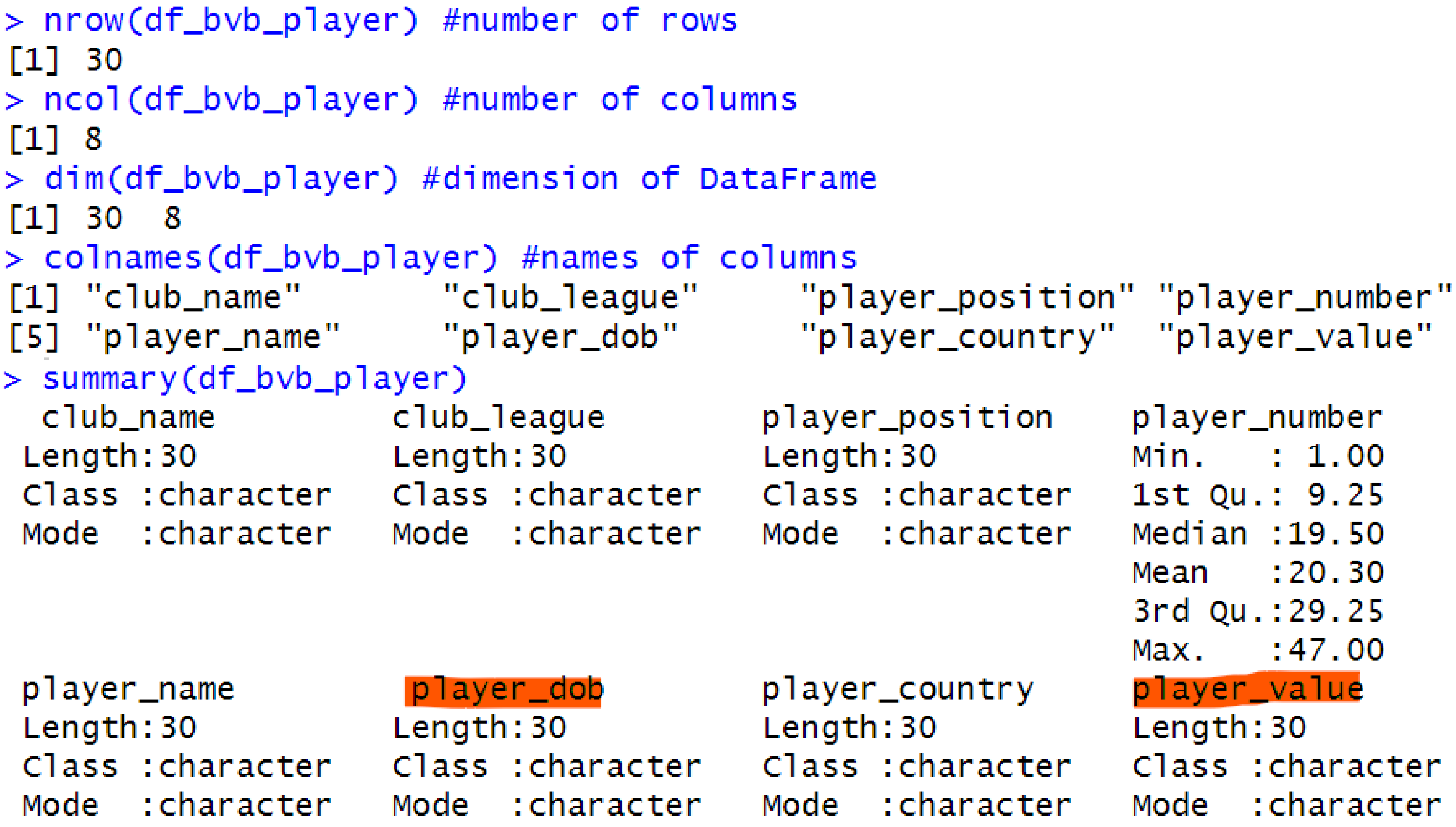
```
> head(df_bvb_player) #View the first rows of dataset
# A tibble: 6 × 8
  club_name   club_league player_position player_number player_name player_dob
  <chr>       <chr>       <chr>                   <dbl> <chr>       <chr>
1 Borussia D… Bundesliga  Torwart                     1 Gregor Kob… 06.12.199…
2 Borussia D… Bundesliga  Torwart                    35 Marcel Lot… 25.05.200…
3 Borussia D… Bundesliga  Torwart                    33 Alexander … 13.04.199…
4 Borussia D… Bundesliga  Torwart                    31 Silas Ostr… 19.11.200…
5 Borussia D… Bundesliga  Abwehr                      4 Nico Schlo… 01.12.199…
6 Borussia D… Bundesliga  Abwehr                     25 Niklas Süle 03.09.199…
# ℹ 2 more variables: player_country <chr>, player_value <chr>
> view(df_bvb_player) #View DataFrame in an additional sub-window
```


GDriveR - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
FHDTM-P2DS-WS2324_RI.rmd*
df_bvb_player
Filter

	club_name	club_league	player_position	player_number	player_name	player_dob	pl
1	Borussia Dortmund	Bundesliga	Torwart	1	Gregor Kobel	06.12.1997 (25)	
2	Borussia Dortmund	Bundesliga	Torwart	35	Marcel Lotka	25.05.2001 (22)	
3	Borussia Dortmund	Bundesliga	Torwart	33	Alexander Meyer	13.04.1991 (32)	

Viewing Meta Data

In R: Explore DataFrame (Recap)



```
> nrow(df_bvb_player) #number of rows
[1] 30
> ncol(df_bvb_player) #number of columns
[1] 8
> dim(df_bvb_player) #dimension of DataFrame
[1] 30  8
> colnames(df_bvb_player) #names of columns
[1] "club_name"       "club_league"     "player_position" "player_number"
[5] "player_name"     "player_dob"      "player_country"  "player_value"
> summary(df_bvb_player)
  club_name          club_league        player_position    player_number
 Length:30          Length:30          Length:30          Min.   : 1.00
 Class :character   Class :character   Class :character   1st Qu.: 9.25
 Mode  :character   Mode  :character   Mode  :character   Median :19.50
                                                          Mean   :20.30
                                                          3rd Qu.:29.25
                                                          Max.   :47.00
 player_name         player_dob         player_country     player_value
 Length:30          Length:30          Length:30          Length:30
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character
```

Major Tasks in **Data Preprocessing**

Data Reduction

R DataFrame: Selecting

- Obtains reduced representation in volume but produces the same or similar analytical results.

Data Cleaning

R DataFrame: Slicing, Filtering, Mutating, Renaming

- **Fill in missing values**, **smooth noisy data**, identify or remove outliers, and resolve inconsistencies caused by data integration.

Data Integration

- Integration of multiple tables, databases, data cubes, or files.

Data Transformation

- Aggregation, generalization, normalization and attribute construction.

Missing values & attribute construction

Function `is.na(vector)` works with any type of vector.

- Returns TRUE for missing values – FALSE for anything else.

In R, with

- `mutate():` append or change columns to data frames and
- `if_else():` conditional transformation or
- `case_when():` conditional transformations

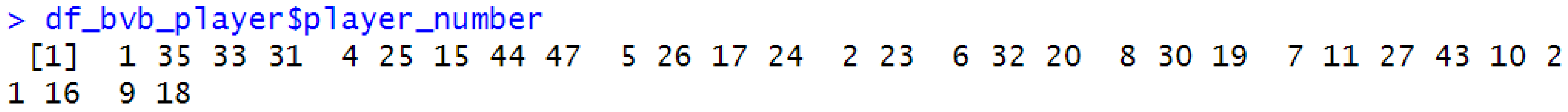
we have all functional tools for mutating vectors and data frames.

library(dplyr)

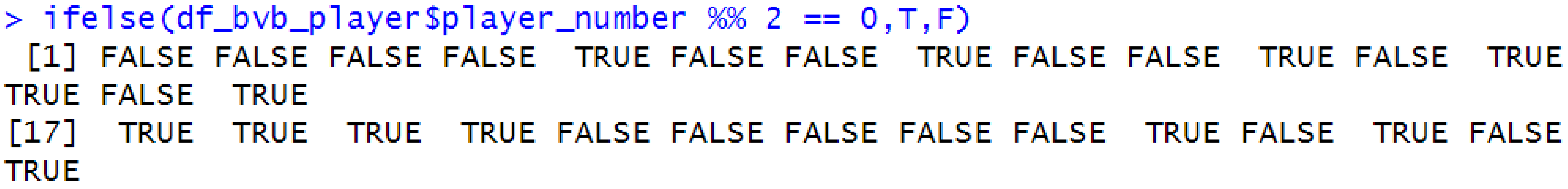
Conditional Transformations
if_else()

`if_else(condition, value_true, value_false, value_NA)`

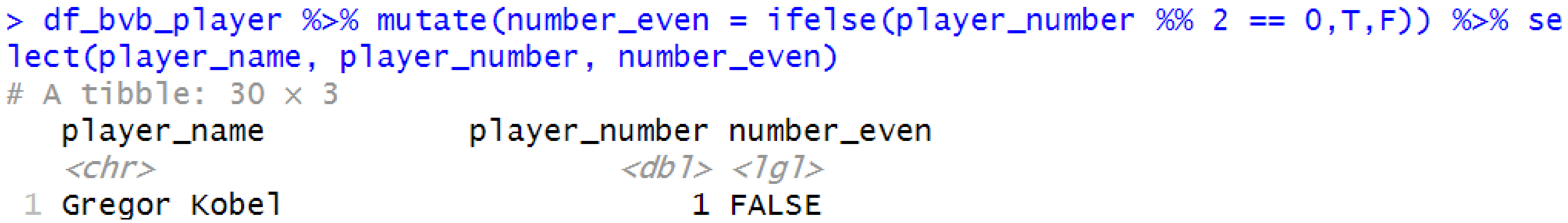
helps us to generate conditional transformations (e.g., **attribute construction**, missing values):



```
> df_bvb_player$player_number
 [1]  1 35 33 31  4 25 15 44 47  5 26 17 24  2 23  6 32 20  8 30 19  7 11 27 43 10 2
1 16  9 18
```



```
> ifelse(df_bvb_player$player_number %% 2 == 0,T,F)
 [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE
TRUE FALSE  TRUE
[17]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
TRUE
```



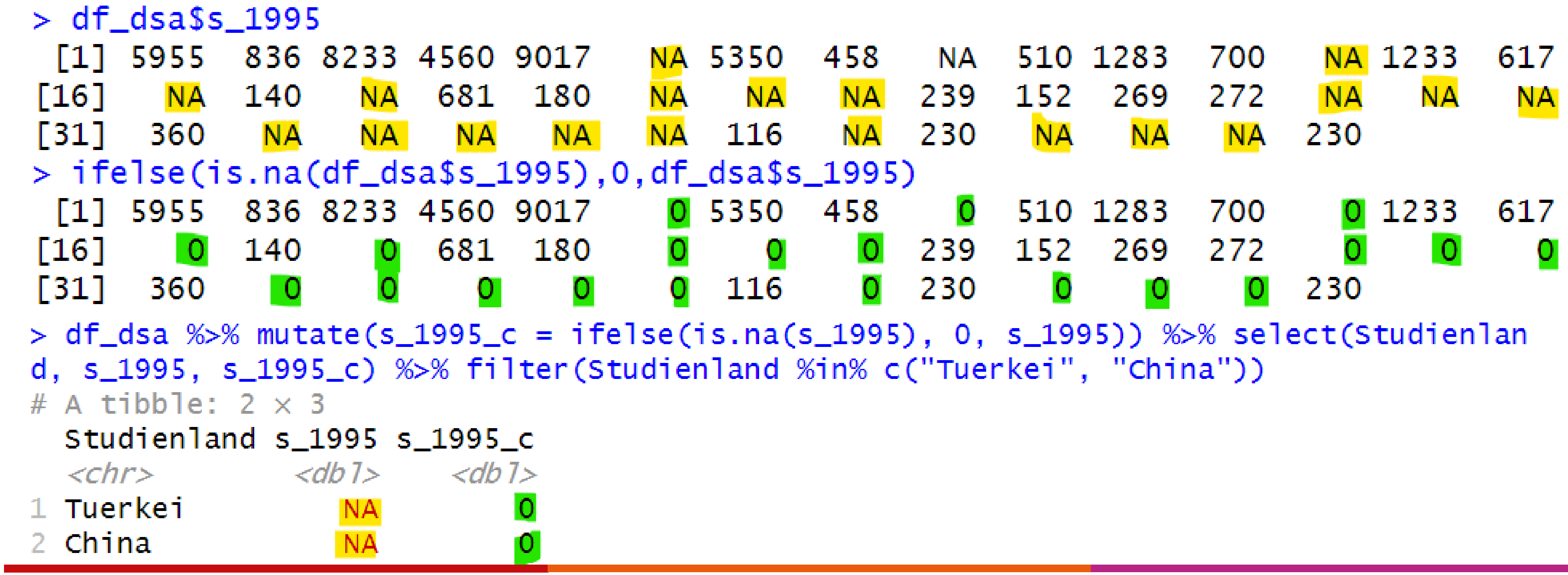
```
> df_bvb_player %>% mutate(number_even = ifelse(player_number %% 2 == 0,T,F)) %>% se
lect(player_name, player_number, number_even)
# A tibble: 30 × 3
   player_name          player_number number_even
   <chr>                        <dbl> <lgl>
 1 Gregor Kobel                     1 FALSE
```

library(dplyr)

Conditional Transformations if_else()

```
if_else(condition, value_true, value_false, value_NA)
```

helps us to generate conditional transformations (e.g., attribute construction, **missing values**):



```
> df_dsa$s_1995
 [1] 5955  836 8233 4560 9017   NA 5350  458   NA  510 1283  700   NA 1233  617
[16]   NA  140   NA  681  180   NA   NA   NA  239  152  269  272   NA   NA   NA
[31]  360   NA   NA   NA   NA   NA  116   NA  230   NA   NA   NA  230
> ifelse(is.na(df_dsa$s_1995),0,df_dsa$s_1995)
 [1] 5955  836 8233 4560 9017    0 5350  458    0  510 1283  700    0 1233  617
[16]    0  140    0  681  180    0    0    0  239  152  269  272    0    0    0
[31]  360    0    0    0    0    0  116    0  230    0    0    0  230
> df_dsa %>% mutate(s_1995_c = ifelse(is.na(s_1995), 0, s_1995)) %>% select(Studienlan
d, s_1995, s_1995_c) %>% filter(Studienland %in% c("Tuerkei", "China"))
# A tibble: 2 × 3
  Studienland s_1995 s_1995_c
  <chr>        <dbl>    <dbl>
1 Tuerkei         NA        0
2 China           NA        0
```

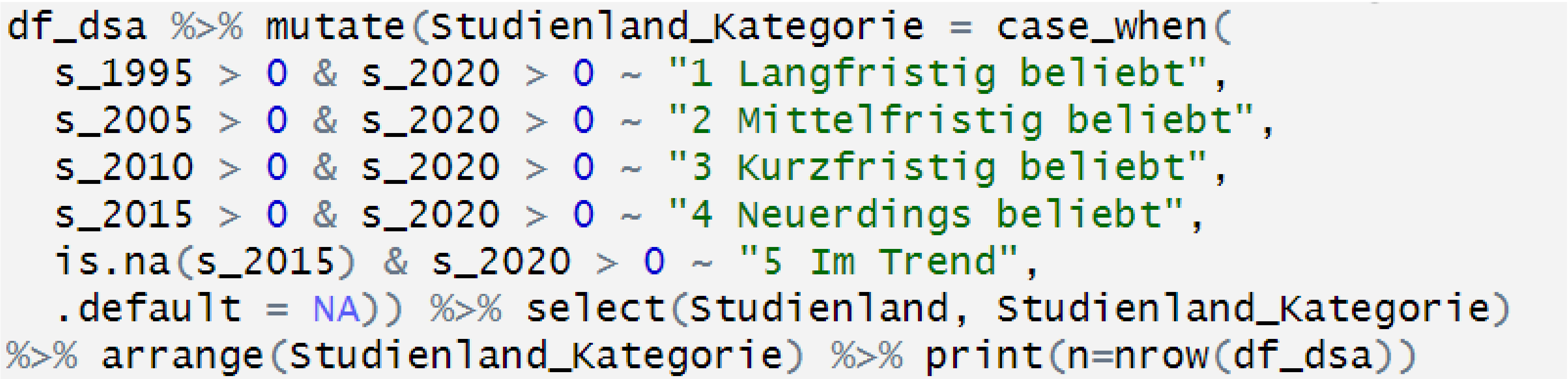
library(dplyr)

Conditional Transformations case_when()

Flexible way of performing different computations for different conditions:

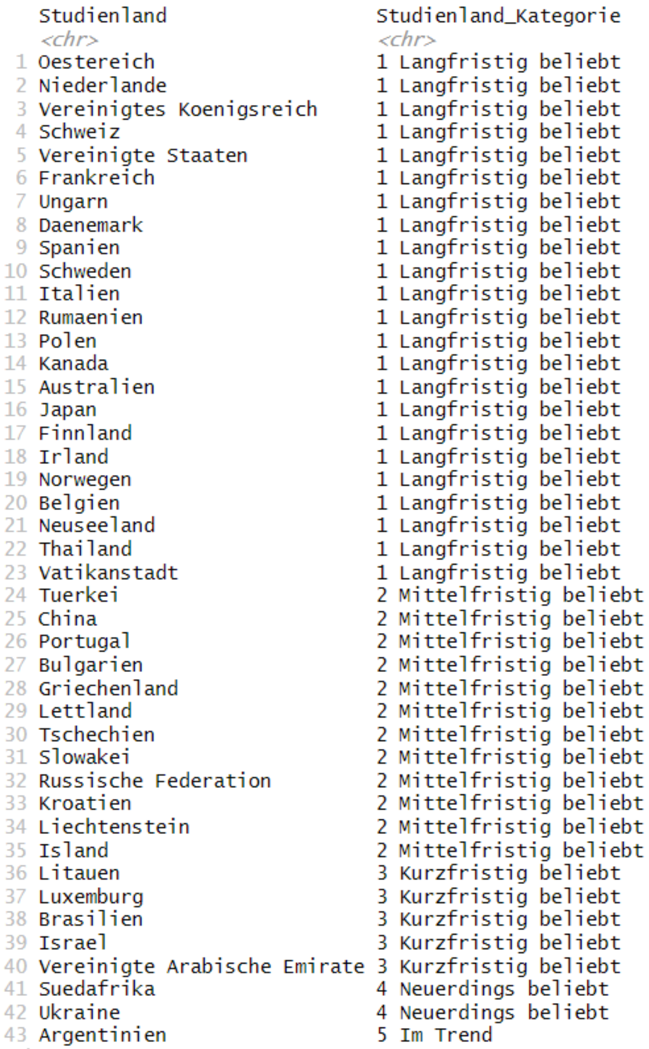
```
case_when( condition_1 ~ output,
           condition_N ~ output,
           .default = output )
```

- Inspired by SQL's CASE statement.



```
df_dsa %>% mutate(Studienland_Kategorie = case_when(
  s_1995 > 0 & s_2020 > 0 ~ "1 Langfristig beliebt",
  s_2005 > 0 & s_2020 > 0 ~ "2 Mittelfristig beliebt",
  s_2010 > 0 & s_2020 > 0 ~ "3 Kurzfristig beliebt",
  s_2015 > 0 & s_2020 > 0 ~ "4 Neuerdings beliebt",
  is.na(s_2015) & s_2020 > 0 ~ "5 Im Trend",
  .default = NA)) %>% select(Studienland, Studienland_Kategorie)
%>% arrange(Studienland_Kategorie) %>% print(n=nrow(df_dsa))
```

n = Inf

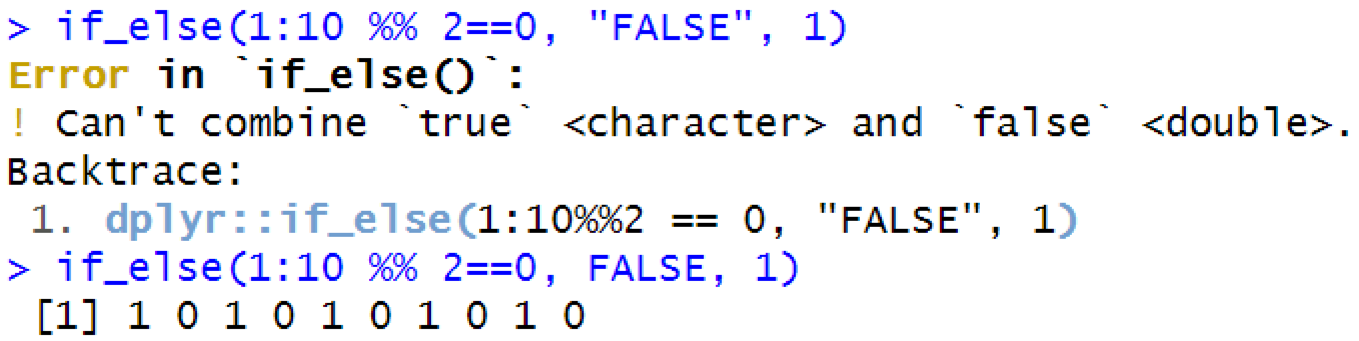


```
   Studienland                  Studienland_Kategorie
   <chr>                        <chr>
 1 Oestereich                   1 Langfristig beliebt
 2 Niederlande                  1 Langfristig beliebt
 3 Vereinigtes Koenigsreich     1 Langfristig beliebt
 4 Schweiz                      1 Langfristig beliebt
 5 Vereinigte Staaten           1 Langfristig beliebt
 6 Frankreich                   1 Langfristig beliebt
 7 Ungarn                       1 Langfristig beliebt
 8 Daenemark                    1 Langfristig beliebt
 9 Spanien                      1 Langfristig beliebt
10 Schweden                     1 Langfristig beliebt
11 Italien                      1 Langfristig beliebt
12 Rumaenien                    1 Langfristig beliebt
13 Polen                        1 Langfristig beliebt
14 Kanada                       1 Langfristig beliebt
15 Australien                   1 Langfristig beliebt
16 Japan                        1 Langfristig beliebt
17 Finnland                     1 Langfristig beliebt
18 Irland                       1 Langfristig beliebt
19 Norwegen                     1 Langfristig beliebt
20 Belgien                      1 Langfristig beliebt
21 Neuseeland                   1 Langfristig beliebt
22 Thailand                     1 Langfristig beliebt
23 Vatikanstadt                 1 Langfristig beliebt
24 Tuerkei                      2 Mittelfristig beliebt
25 China                        2 Mittelfristig beliebt
26 Portugal                     2 Mittelfristig beliebt
27 Bulgarien                    2 Mittelfristig beliebt
28 Griechenland                 2 Mittelfristig beliebt
29 Lettland                     2 Mittelfristig beliebt
30 Tschechien                   2 Mittelfristig beliebt
31 Slowakei                     2 Mittelfristig beliebt
32 Russische Federation         2 Mittelfristig beliebt
33 Kroatien                     2 Mittelfristig beliebt
34 Liechtenstein                2 Mittelfristig beliebt
35 Island                       2 Mittelfristig beliebt
36 Litauen                      3 Kurzfristig beliebt
37 Luxemburg                    3 Kurzfristig beliebt
38 Brasilien                    3 Kurzfristig beliebt
39 Israel                       3 Kurzfristig beliebt
40 Vereinigte Arabische Emirate 3 Kurzfristig beliebt
41 Suedafrika                   4 Neuerdings beliebt
42 Ukraine                      4 Neuerdings beliebt
43 Argentinien                  5 Im Trend
```

Conditional Transformations

Both `if_else()` and `case_when()` require **compatible types** in the **vector output**.

- *In simple words: each conditional row-wise output must have the same data type!*



```
> if_else(1:10 %% 2==0, "FALSE", 1)
Error in `if_else()`:
! Can't combine `true` <character> and `false` <double>.
Backtrace:
 1. dplyr::if_else(1:10%%2 == 0, "FALSE", 1)
> if_else(1:10 %% 2==0, FALSE, 1)
 [1] 1 0 1 0 1 0 1 0 1 0
```

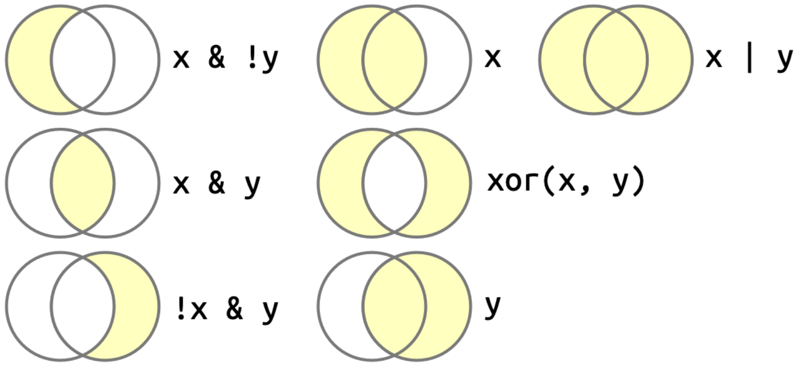
What rules apply for R vectors?

Exceptions for compatible cases in R vectors:

- Numeric and logical vectors
- Dates and date-times
- NA is compatible with everything (every vector can represent a missing value).

Logical Operators **and Summaries**

`a < b`	Less than
`a > b`	Greater than
`a <= b`	Less equal than
`a >= b`	Greater equal than
`a == b`	Equal
`a != b`	Not equal
`!a`	Not
`a | b`	a OR b
`a & b`	a AND b
`isTRUE(a)`	Check whether a is TRUE
`a %in% c`	Check whether a’s value is in a vector c



R for Data Science (e2) by Wickham, Çetinkaya-Rundel, and Grolemund

`any(x)`	True if **any** element in vector x is True
`all(x)`	True if **all** elements in vector x are True
`sum(x)`	Number of elements in vector x that are True
`mean(x)`	Portion of elements in vector x that are True
`length(x)`	Number of elements in vector x

Logical and numeric summaries of logical vectors

Transform Numeric Vectors

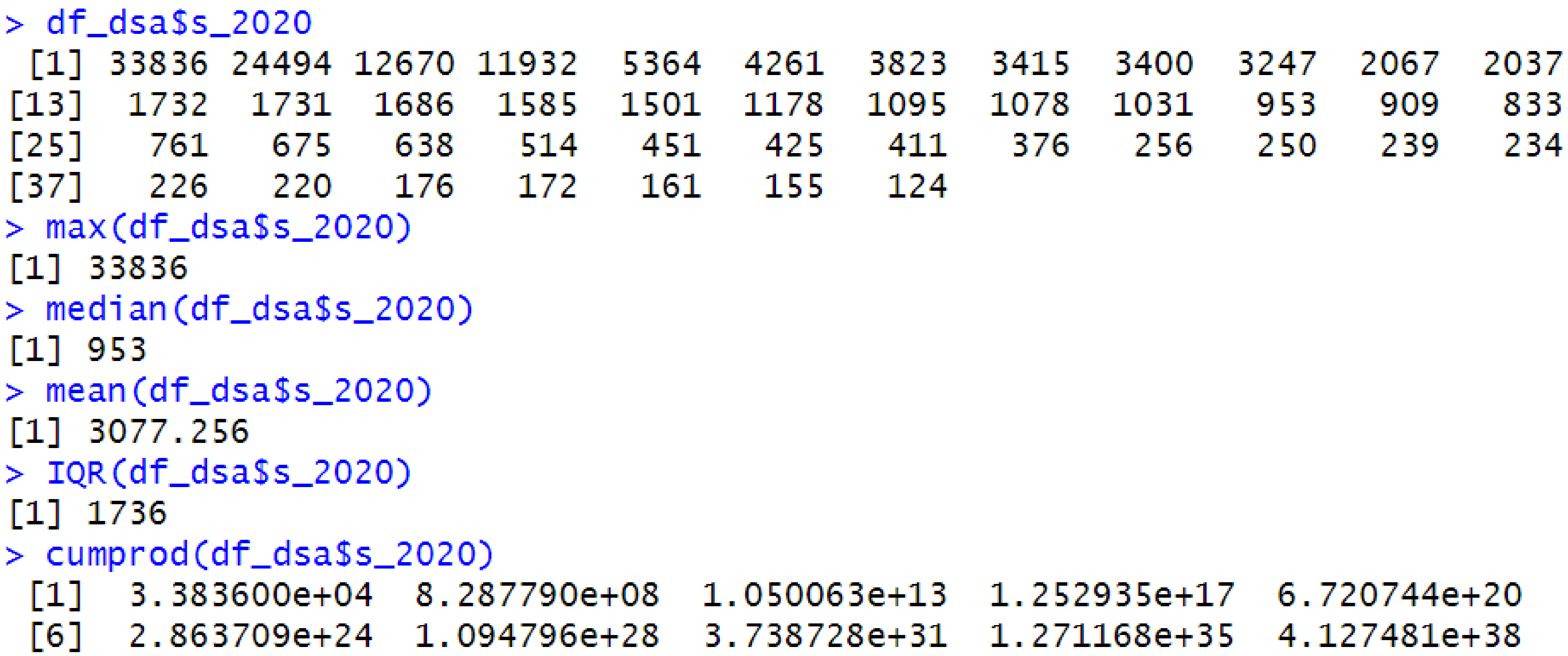
Backbone of data science - NumPy ☺

Operators *where x is a numerical value or vector*	**Numeric transformations**
`+, -, *, /`	
`log(), log2(), log10()`	
`exp(), ^2, ^10`	
`round(x, digits), floor(x), ceiling(x)`	
Aggregation *where x is a numeric vector*	**Numeric summaries**

`min(x), max(x), median(x),`
`quantile(x, %), sd(x) , IQR(x)`
	Single summarizing value

`cumsum(x), cumprod(x), cummean(x)`	Cummulative rolling aggregates

Transform Numeric Vectors (cont.)



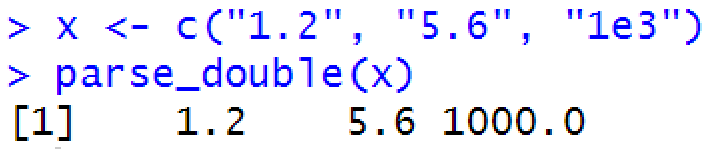
```
> df_dsa$s_2020
 [1] 33836 24494 12670 11932  5364  4261  3823  3415  3400  3247  2067  2037
[13]  1732  1731  1686  1585  1501  1178  1095  1078  1031   953   909   833
[25]   761   675   638   514   451   425   411   376   256   250   239   234
[37]   226   220   176   172   161   155   124
> max(df_dsa$s_2020)
[1] 33836
> median(df_dsa$s_2020)
[1] 953
> mean(df_dsa$s_2020)
[1] 3077.256
> IQR(df_dsa$s_2020)
[1] 1736
> cumprod(df_dsa$s_2020)
 [1]  3.383600e+04  8.287790e+08  1.050063e+13  1.252935e+17  6.720744e+20
 [6]  2.863709e+24  1.094796e+28  3.738728e+31  1.271168e+35  4.127481e+38
```

Transform Numeric Vectors (cont.)

https://readr.tidyverse.org/reference/parse_atomic.html
https://readr.tidyverse.org/reference/parse_number.html

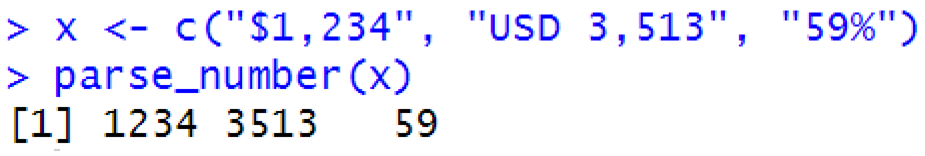
In some cases, you'll **encounter numeric vectors as strings** because something has gone wrong, e.g., in your data import process:

- `parse_double()` tranforms strings into numbers:



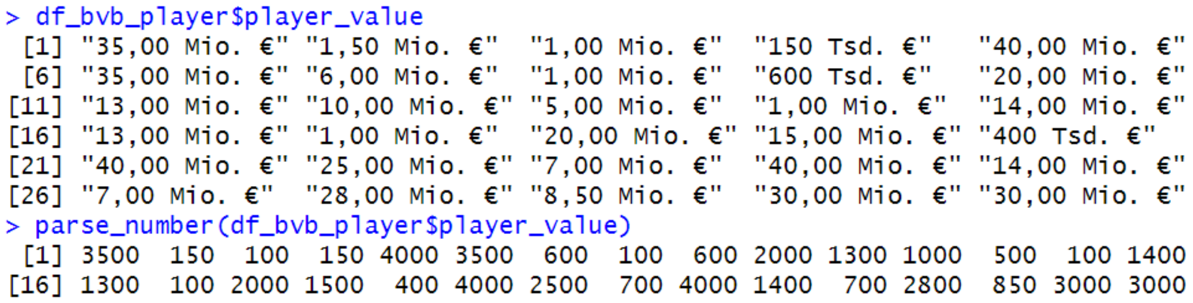
```
> x <- c("1.2", "5.6", "1e3")
> parse_double(x)
[1]    1.2    5.6 1000.0
```

- `parse_number()` tranforms strings into numbers ignoring non-numeric text:



```
> x <- c("$1,234", "USD 3,513", "59%")
> parse_number(x)
[1] 1234 3513   59
```

- `as.numeric()` and `as.double()` will test an object and coerce to numeric.



```
> df_bvb_player$player_value
 [1] "35,00 Mio. €" "1,50 Mio. €"  "1,00 Mio. €"  "150 Tsd. €"   "40,00 Mio. €"
 [6] "35,00 Mio. €" "6,00 Mio. €"  "1,00 Mio. €"  "600 Tsd. €"   "20,00 Mio. €"
[11] "13,00 Mio. €" "10,00 Mio. €" "5,00 Mio. €"  "1,00 Mio. €"  "14,00 Mio. €"
[16] "13,00 Mio. €" "1,00 Mio. €"  "20,00 Mio. €" "15,00 Mio. €" "400 Tsd. €"
[21] "40,00 Mio. €" "25,00 Mio. €" "7,00 Mio. €"  "40,00 Mio. €" "14,00 Mio. €"
[26] "7,00 Mio. €"  "28,00 Mio. €" "8,50 Mio. €"  "30,00 Mio. €" "30,00 Mio. €"
> parse_number(df_bvb_player$player_value)
 [1] 3500  150  100  150 4000 3500  600  100  600 2000 1300 1000  500  100 1400
[16] 1300  100 2000 1500  400 4000 2500  700 4000 1400  700 2800  850 3000 3000
```

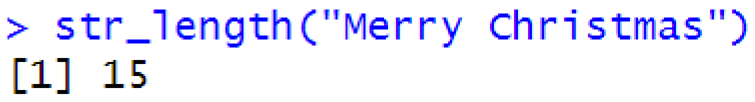
Transform String Vectors

https://stringr.tidyverse.org/

All `stringr` functions start with `str_`

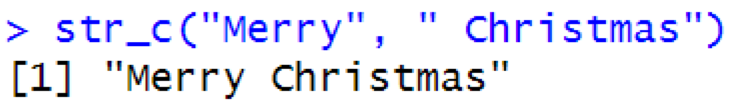
- Advantage in RStudio: typing `str_` will trigger autocomplete

- `str_length()` returns the number of letters in the string:



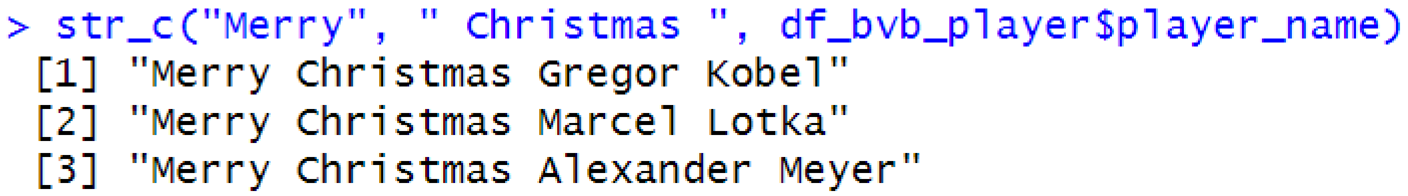
```
> str_length("Merry Christmas")
[1] 15
```

- `str_c()` takes any number of strings and returns a character vector:



```
> str_c("Merry", " Christmas")
[1] "Merry Christmas"
```

- `str_c()` can also process strings and string vectors input:



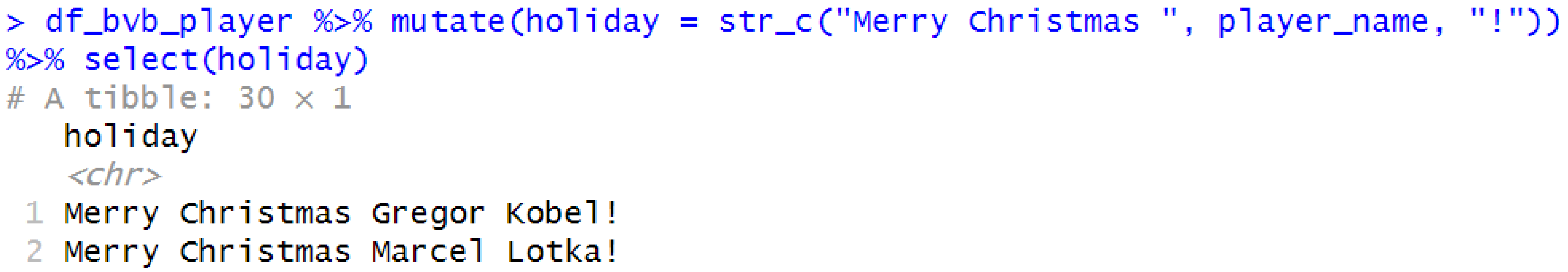
```
> str_c("Merry", " Christmas ", df_bvb_player$player_name)
 [1] "Merry Christmas Gregor Kobel"
 [2] "Merry Christmas Marcel Lotka"
 [3] "Merry Christmas Alexander Meyer"
```

Transform String Vectors (cont.)

https://stringr.tidyverse.org/reference/str_c.html

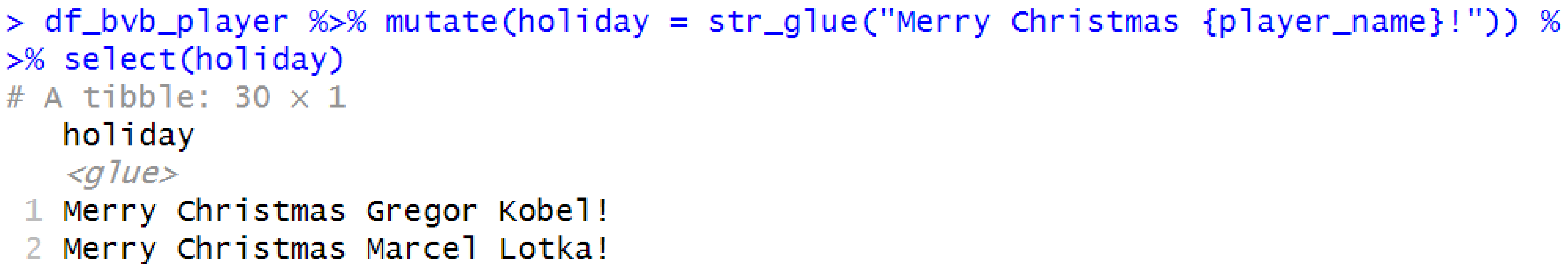
Both these methods can also be applied to DataFrames via `mutate()`:

- `str_c()`:



```
> df_bvb_player %>% mutate(holiday = str_c("Merry Christmas ", player_name, "!"))
%>% select(holiday)
# A tibble: 30 × 1
   holiday
   <chr>
 1 Merry Christmas Gregor Kobel!
 2 Merry Christmas Marcel Lotka!
```

- `str_glue()` glues with anything inside `{column}`:

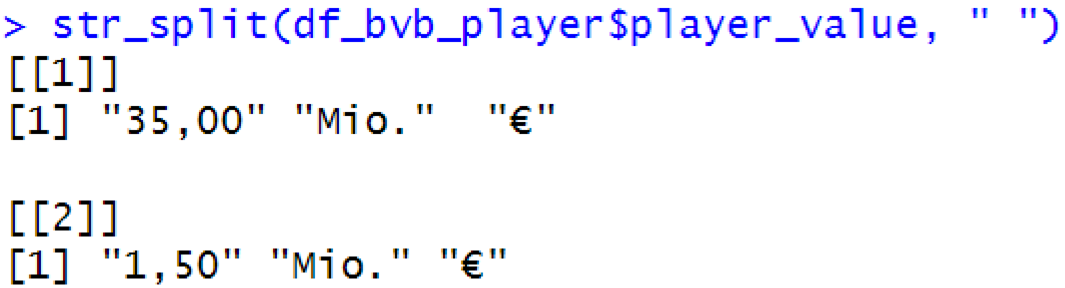


```
> df_bvb_player %>% mutate(holiday = str_glue("Merry Christmas {player_name}!")) %
>% select(holiday)
# A tibble: 30 × 1
   holiday
   <glue>
 1 Merry Christmas Gregor Kobel!
 2 Merry Christmas Marcel Lotka!
```

Transform String Vectors (cont.)

https://stringr.tidyverse.org/reference/str_split.html

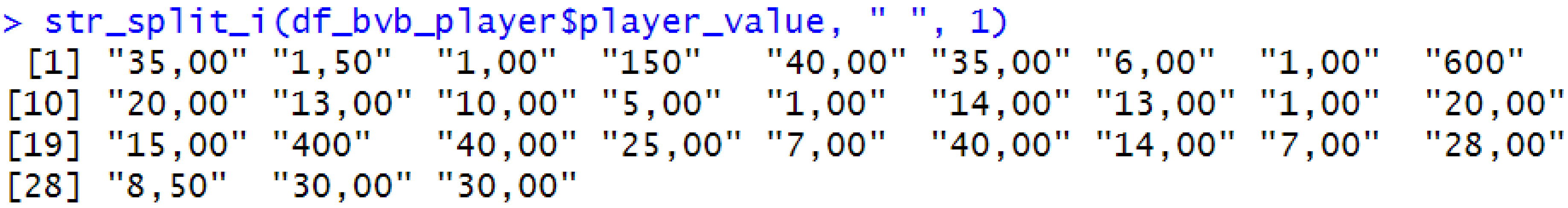
- `str_split()` splits each string in a character vector into a varying number of pieces:



```
> str_split(df_bvb_player$player_value, " ")
[[1]]
[1] "35,00" "Mio."  "€"

[[2]]
[1] "1,50" "Mio." "€"
```

- `str_split_i()` does `str_split()` and extracts the ith value:

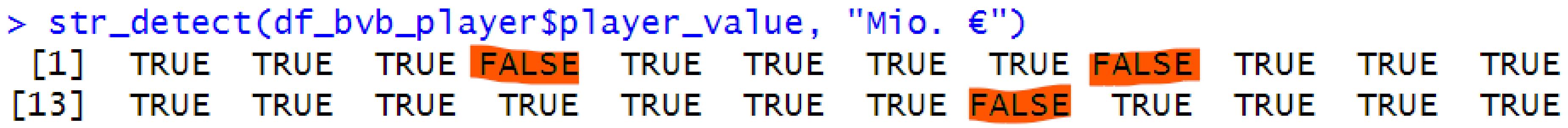


```
> str_split_i(df_bvb_player$player_value, " ", 1)
 [1] "35,00" "1,50"  "1,00"  "150"   "40,00" "35,00" "6,00"  "1,00"  "600"  
[10] "20,00" "13,00" "10,00" "5,00"  "1,00"  "14,00" "13,00" "1,00"  "20,00"
[19] "15,00" "400"   "40,00" "25,00" "7,00"  "40,00" "14,00" "7,00"  "28,00"
[28] "8,50"  "30,00" "30,00"
```

Transform String Vectors (cont.)

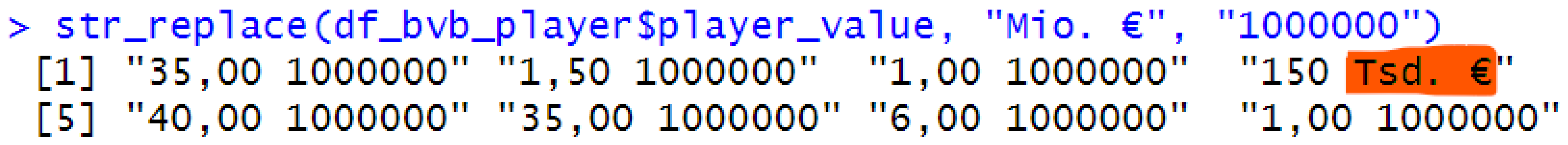
https://stringr.tidyverse.org/reference/str_split.html

- `str_detect(vector, pattern)` returns a logical vector with pattern matches:



```
> str_detect(df_bvb_player$player_value, "Mio. €")
 [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

- `str_replace(vector, pattern, substitute)` replaces the first match and `str_replace_all()` replaces all matches.

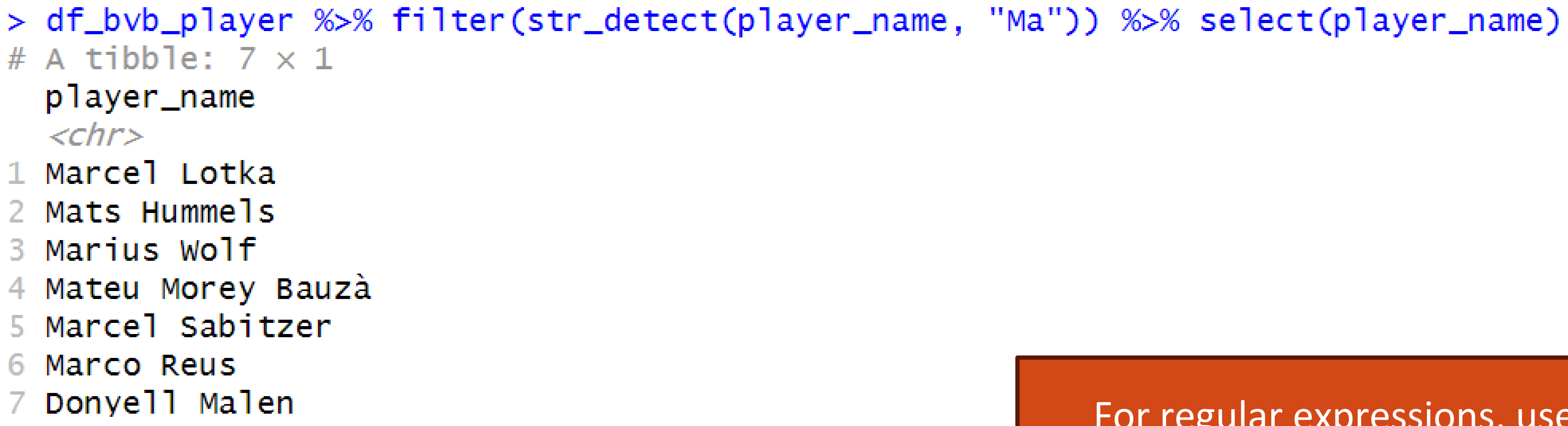


```
> str_replace(df_bvb_player$player_value, "Mio. €", "1000000")
 [1] "35,00 1000000" "1,50 1000000"  "1,00 1000000"  "150 Tsd. €"
 [5] "40,00 1000000" "35,00 1000000" "6,00 1000000"  "1,00 1000000"
```

- `str_remove()` and `str_remove_all()` handy shortcuts for `str_replace(x, pattern, "")`

Transform String Vectors (cont.)

String operations pair well with `filter()` and `str_detect()` :

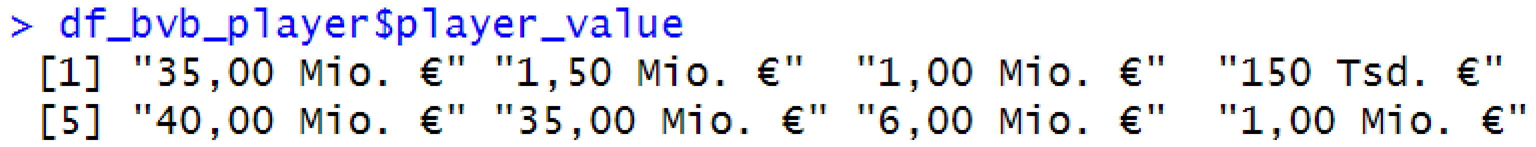


```
> df_bvb_player %>% filter(str_detect(player_name, "Ma")) %>% select(player_name)
# A tibble: 7 × 1
  player_name
  <chr>
1 Marcel Lotka
2 Mats Hummels
3 Marius Wolf
4 Mateu Morey Bauzà
5 Marcel Sabitzer
6 Marco Reus
7 Donyell Malen
```

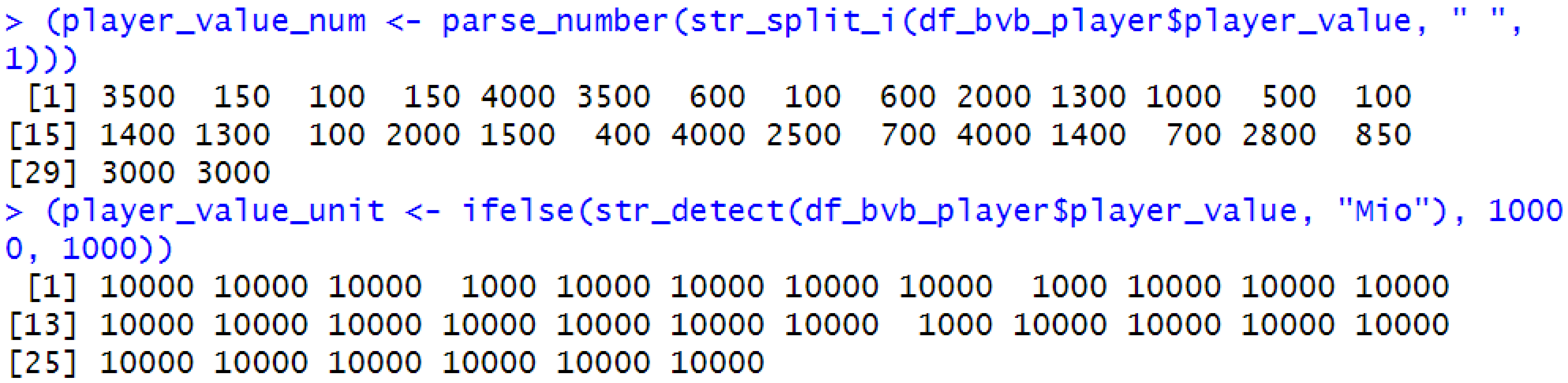
For regular expressions, use `separate_wider_regex()`

Transform String Vectors (cont.)

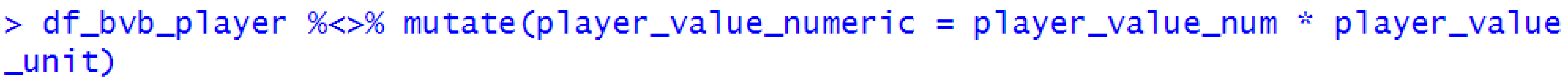
String operations pair well with `mutate()` and `str_detect()`:



```
> df_bvb_player$player_value
 [1] "35,00 Mio. €" "1,50 Mio. €"  "1,00 Mio. €"  "150 Tsd. €"
 [5] "40,00 Mio. €" "35,00 Mio. €" "6,00 Mio. €"  "1,00 Mio. €"
```



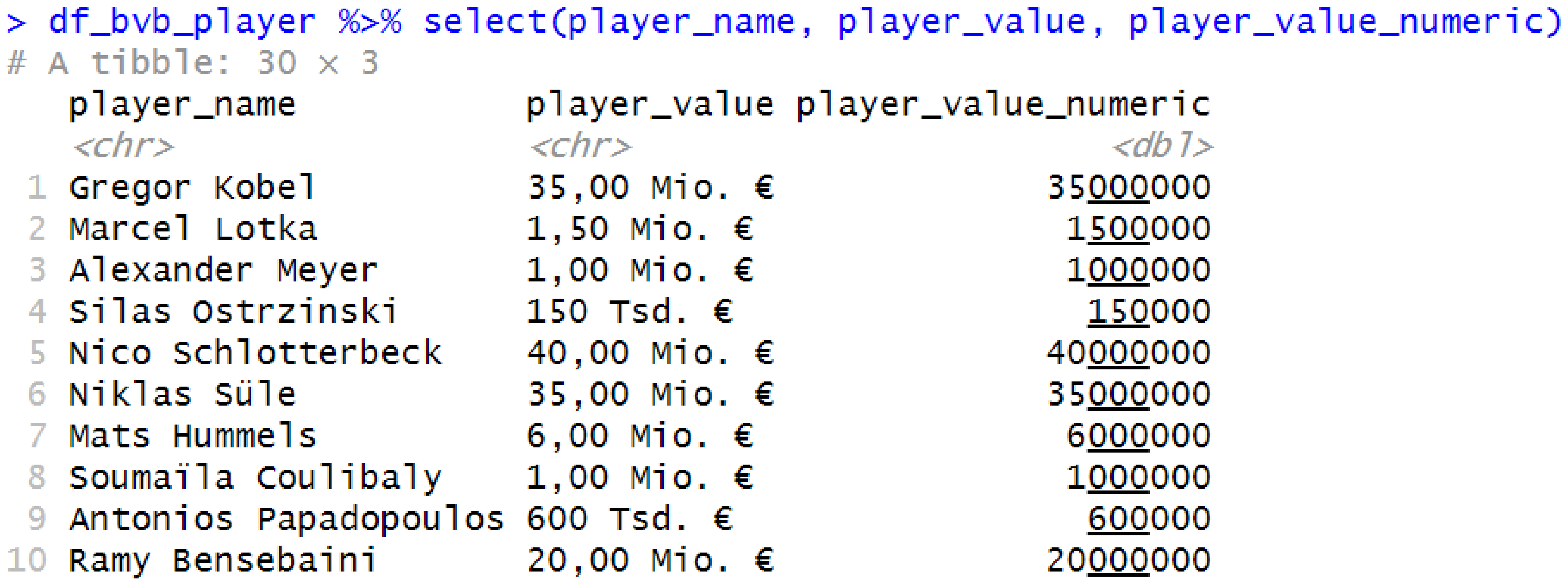
```
> (player_value_num <- parse_number(str_split_i(df_bvb_player$player_value, " ",
1)))
 [1] 3500  150  100  150 4000 3500  600  100  600 2000 1300 1000  500  100
[15] 1400 1300  100 2000 1500  400 4000 2500  700 4000 1400  700 2800  850
[29] 3000 3000
> (player_value_unit <- ifelse(str_detect(df_bvb_player$player_value, "Mio"), 1000
0, 1000))
 [1] 10000 10000 10000  1000 10000 10000 10000 10000  1000 10000 10000 10000
[13] 10000 10000 10000 10000 10000 10000 10000  1000 10000 10000 10000 10000
[25] 10000 10000 10000 10000 10000 10000
```



```
> df_bvb_player %<>% mutate(player_value_numeric = player_value_num * player_value
_unit)
```

Transform String Vectors (cont.)

String operations pair well with `mutate()` and `str_detect()` :



```
> df_bvb_player %>% select(player_name, player_value, player_value_numeric)
# A tibble: 30 × 3
   player_name          player_value player_value_numeric
   <chr>                <chr>                       <dbl>
 1 Gregor Kobel         35,00 Mio. €             35000000
 2 Marcel Lotka         1,50 Mio. €               1500000
 3 Alexander Meyer      1,00 Mio. €               1000000
 4 Silas Ostrzinski     150 Tsd. €                 150000
 5 Nico Schlotterbeck   40,00 Mio. €             40000000
 6 Niklas Süle          35,00 Mio. €             35000000
 7 Mats Hummels         6,00 Mio. €               6000000
 8 Soumaïla Coulibaly   1,00 Mio. €               1000000
 9 Antonios Papadopoulos 600 Tsd. €                600000
10 Ramy Bensebaini      20,00 Mio. €             20000000
```

What We Watched: A Netflix Engagement Report
https://about.netflix.com/en/news/what-we-watched-a-netflix-engagement-report

Training #1

1. Import the following .csv dataset: https://github.com/leotraeg/FHDTM-P2DS-WS2425/raw/refs/heads/main/Praktikum/Netflix.csv as a DataFrame called **df_netflix** in R using the readr library.

2. Transform the attribute Hours_Viewed to a numeric data type using `mutate()` and `parse_number()` or `parse_double()`. Did the type conversion work?

3. `mutate()` the Hours_Viewed attribute to delete the empty spaces " " beforehand.
Hint: *`str_replace()`/ `str_replace_all()`/`str_remove()`/`str_remove_all()`.*
What function makes most sense to use for "812 100 000"?

4. What title had the **most hours** watched that was a "Season 6" and globally available?

5. Create an additional attribute **category_love** for titles that contain "love" or "Love".
How many titles have this category? *Hint: sum(df_netflix$category_love)*
What portion? *Hint: mean(df_netflix$category_love)*

Break

Dates and times

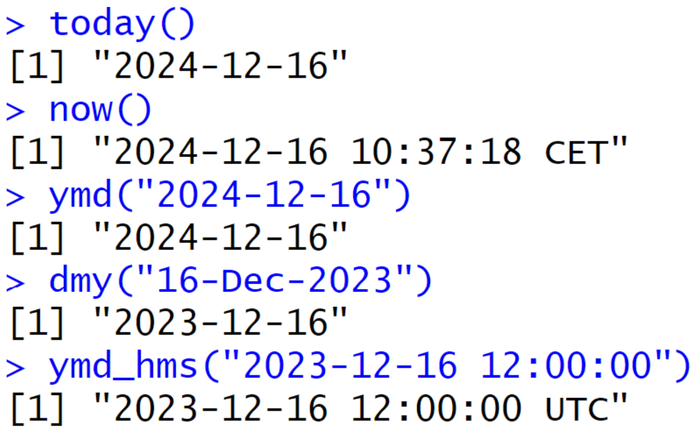
Consolidate year, month, day, hour, and minute columns into a single date via `make_date()` or `make_datetime()`

In R, there are **three types** of date/time data:

- `<date>`
- `<time>` within a day.
- `<dttm>` date plus a time (caution: need to handle time zones).

Hint: you should prefer the simplest date type for your needs.

- Current date: `today()`.
- Current date-time: `now()`.
- Create date/time via `ymd("2023-12-18")`, `dmy("18-Dec-2023")`,`ymd_hms("2023-12-18 15:00:00")`.
- Switch types via `as.Date(date, format="%d.%m.%Y")` or in the import wizzard of `read_csv` (readr).



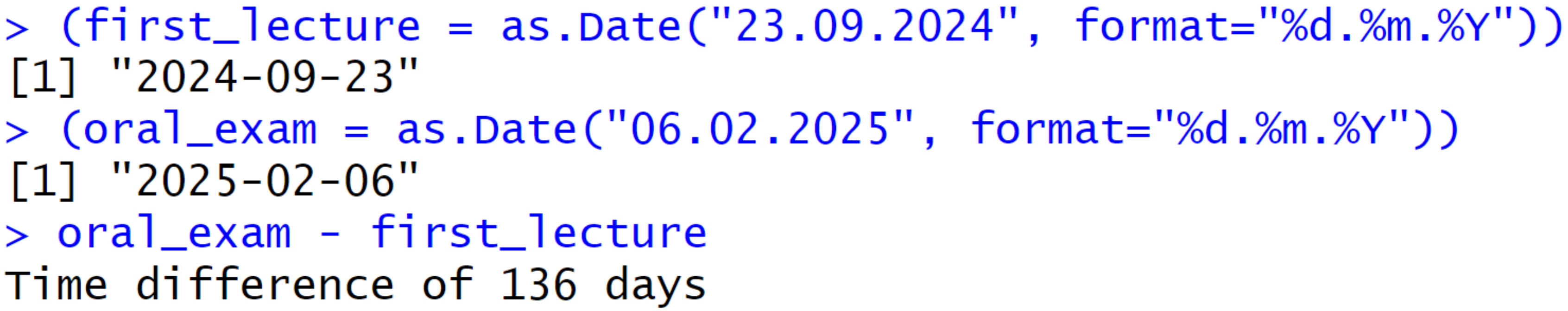
```
> today()
[1] "2024-12-16"
> now()
[1] "2024-12-16 10:37:18 CET"
> ymd("2024-12-16")
[1] "2024-12-16"
> dmy("16-Dec-2023")
[1] "2023-12-16"
> ymd_hms("2023-12-16 12:00:00")
[1] "2023-12-16 12:00:00 UTC"
```

Durations

https://r4ds.hadley.nz/datetimes#time-spans

In R, when you subtract two dates, you get a `difftime` object.

- Records a time span of seconds, minutes, hours, days, or weeks.



```
> (first_lecture = as.Date("23.09.2024", format="%d.%m.%Y"))
[1] "2024-09-23"
> (oral_exam = as.Date("06.02.2025", format="%d.%m.%Y"))
[1] "2025-02-06"
> oral_exam - first_lecture
Time difference of 136 days
```

You can also apply `as.Date()` on vectors

In case difftime becomes ambiguous, you can use granular time spans and arithmetics:

- Durations: exact number of seconds `as.duration(difftime_object)`
- Periods: units like weeks and months.
- Intervals: a starting and ending point.

What We Watched: A Netflix Engagement Report
https://about.netflix.com/en/news/what-we-watched-a-netflix-engagement-report

Training #2

1. Use your existing **df_netflix** or import the following .csv dataset: https://github.com/leotraeg/FHDTM-P2DS-WS2425/raw/refs/heads/main/Praktikum/Netflix.csv as a DataFrame called **df_netflix** in R using the readr library.

2. Create the variable **netflix_date_released** with the date *01.07.2023*.

3. Change the data type of the Release_Date to a date format. You can use `mutate(Release_Date = as.Date(Release_Date, format="%d.%m.%Y"))` on **df_netflix**.

4. Create a new attribute via mutate() called **duration** storing the number of days between **Release_Date** and **netflix_date_released**.

5. What title was released for the longest duration?
   You can sort the **df_netflix** based on the duration by using
   `df_netflix %>% arrange(desc(duration))`

Program in R

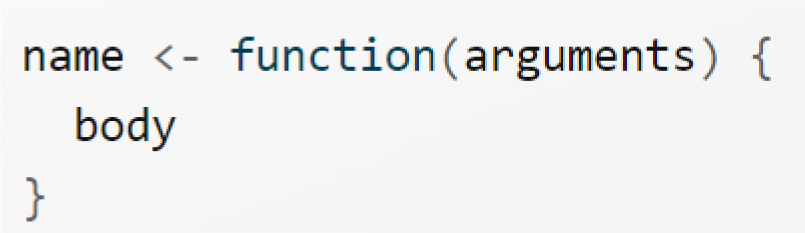
- Deploy functions to **automate common tasks** instead copy-and-pasting.
- Makes your code **easier to understand**.
- Update code in one place, instead of many (and reduce errors).
- Reuse work from project-to-project.

Analyze your repeated code to figure what parts are constant and what parts vary.

Vector Functions

Are unary, binary, … functions: take one or more vectors and return a vector result.

You need three elements to write a function using the template:



```
name <- function(arguments) {
  body
}
```

- Name: descriptive name of the function (do not be reserved).
- Arguments: the numerical or character typed vector input(s).
- Body: the code that is repeated across all the calls.

Vector Functions (cont.)

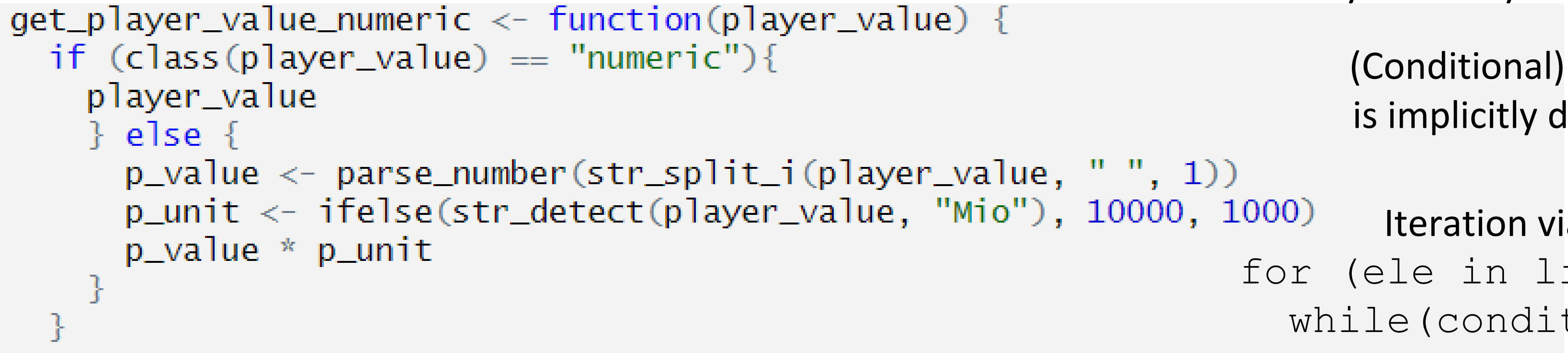
Conditions in paranthesis and conditional execution in curly paranthesis.

Internal variables can be dynamically declared.

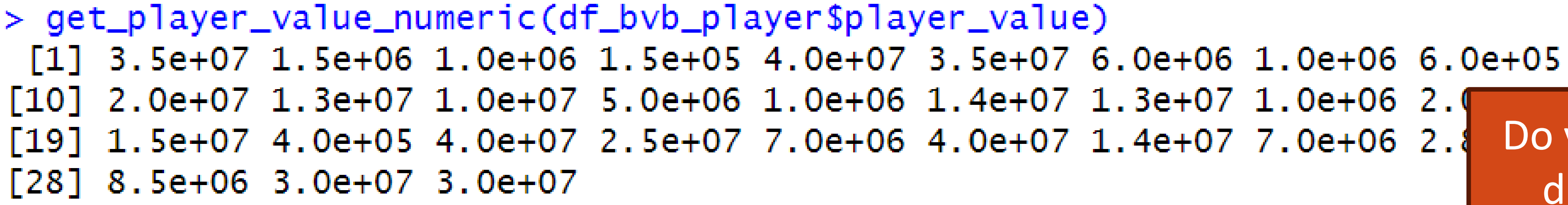
(Conditional) output is implicitly defined.

Iteration via

```
for (ele in list(c))
while(condition)
```



```
get_player_value_numeric <- function(player_value) {
  if (class(player_value) == "numeric"){
    player_value
    } else {
      p_value <- parse_number(str_split_i(player_value, " ", 1))
      p_unit <- ifelse(str_detect(player_value, "Mio"), 10000, 1000)
      p_value * p_unit
    }
  }
```

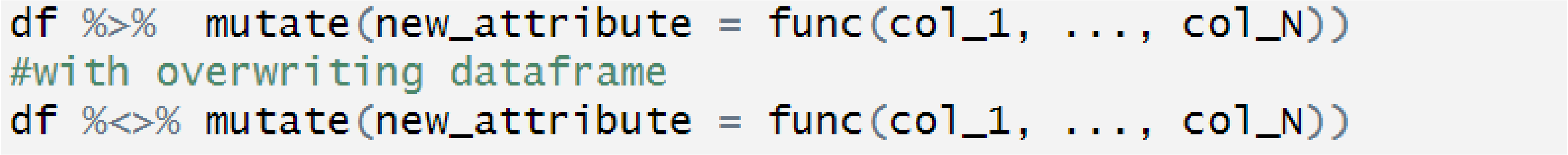


```
> get_player_value_numeric(df_bvb_player$player_value)
 [1] 3.5e+07 1.5e+06 1.0e+06 1.5e+05 4.0e+07 3.5e+07 6.0e+06 1.0e+06 6.0e+05
[10] 2.0e+07 1.3e+07 1.0e+07 5.0e+06 1.0e+06 1.4e+07 1.3e+07 1.0e+06 2.0
[19] 1.5e+07 4.0e+05 4.0e+07 2.5e+07 7.0e+06 4.0e+07 1.4e+07 7.0e+06 2.8
[28] 8.5e+06 3.0e+07 3.0e+07
```

Do you notice any differences in functional syntax?

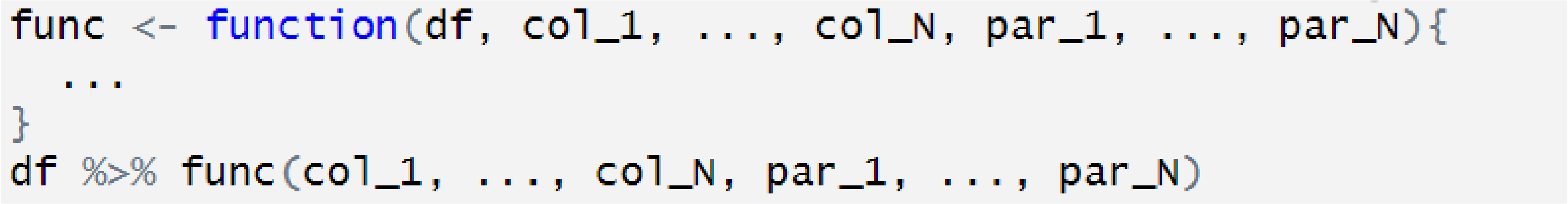
DataFrame Functions

We keep the `mutate()` function to apply functions on one or multiple attributes:



```
df %>%  mutate(new_attribute = func(col_1, ..., col_N))
#with overwriting dataframe
df %<>% mutate(new_attribute = func(col_1, ..., col_N))
```

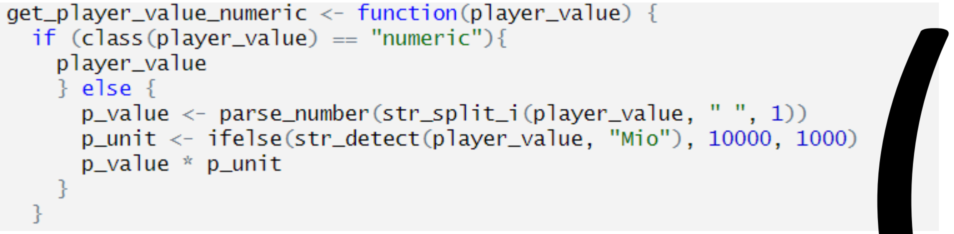
Or apply functions immediately on data frames if the first argument is a data frame:



```
func <- function(df, col_1, ..., col_N, par_1, ..., par_N){
  ...
}
df %>% func(col_1, ..., col_N, par_1, ..., par_N)
```

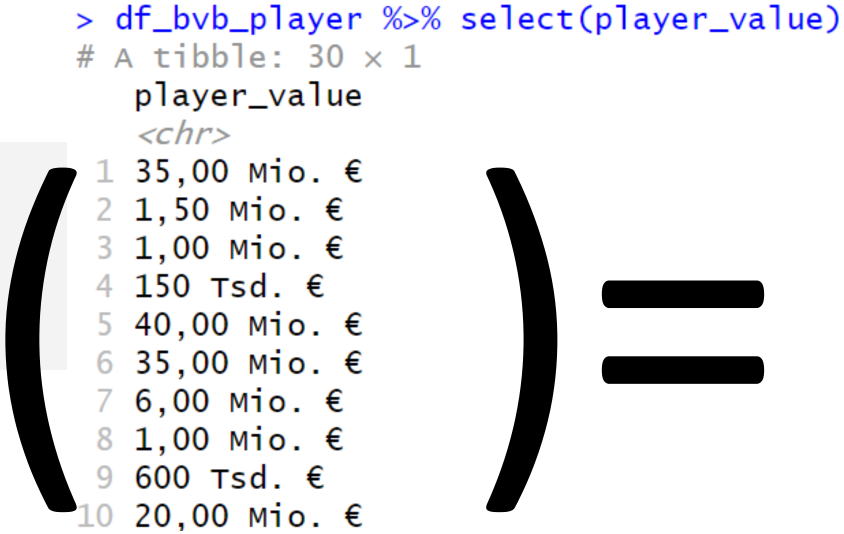
Data Cleaning: **smooth noisy data**

Definition: `function(A)` → `A`, where `A` is a set



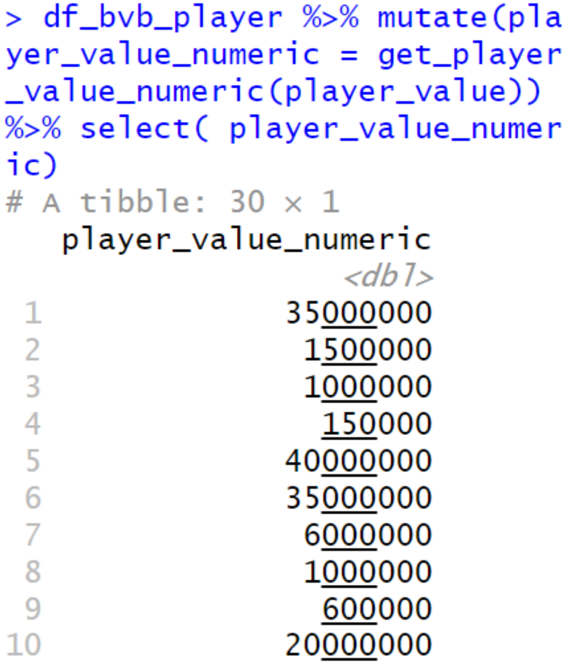
```r
get_player_value_numeric <- function(player_value) {
  if (class(player_value) == "numeric"){
    player_value
    } else {
      p_value <- parse_number(str_split_i(player_value, " ", 1))
      p_unit <- ifelse(str_detect(player_value, "Mio"), 10000, 1000)
      p_value * p_unit
    }
  }
```

(



```
> df_bvb_player %>% select(player_value)
# A tibble: 30 × 1
   player_value
   <chr>
 1 35,00 Mio. €
 2 1,50 Mio. €
 3 1,00 Mio. €
 4 150 Tsd. €
 5 40,00 Mio. €
 6 35,00 Mio. €
 7 6,00 Mio. €
 8 1,00 Mio. €
 9 600 Tsd. €
10 20,00 Mio. €
```

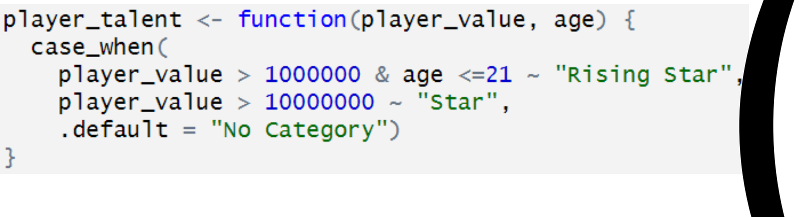
) =



```
> df_bvb_player %>% mutate(pla
yer_value_numeric = get_player
_value_numeric(player_value))
%>% select( player_value_numer
ic)
# A tibble: 30 × 1
   player_value_numeric
                  <dbl>
 1             35000000
 2              1500000
 3              1000000
 4               150000
 5             40000000
 6             35000000
 7              6000000
 8              1000000
 9               600000
10             20000000
```

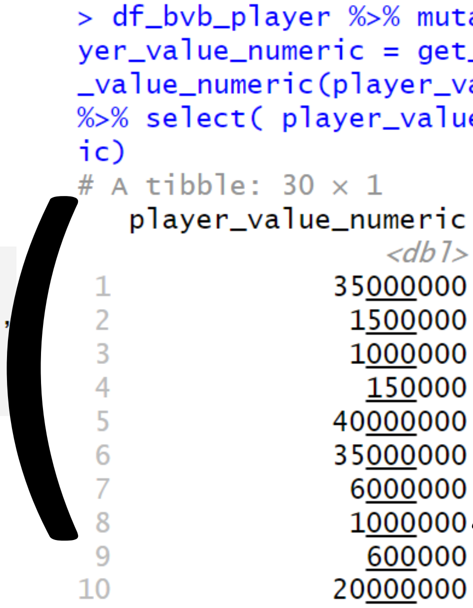
Data Transformation: **attribute construction**

Definition binary function: `f(A, B)` → `A*B`



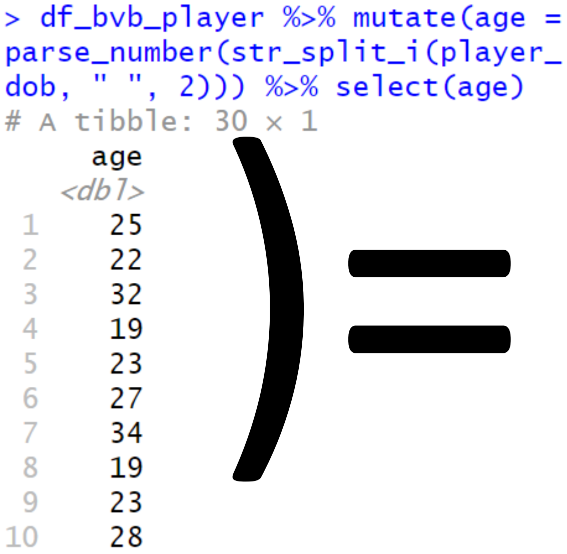
```
player_talent <- function(player_value, age) {
  case_when(
    player_value > 1000000 & age <=21 ~ "Rising Star",
    player_value > 10000000 ~ "Star",
    .default = "No Category")
}
```

(



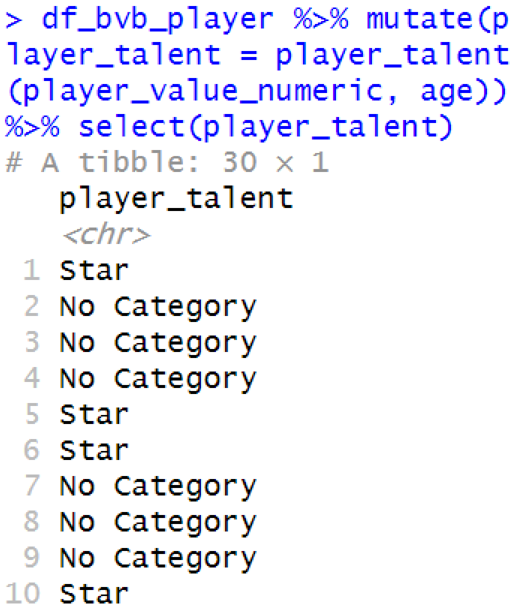
```
> df_bvb_player %>% mutate(pla
yer_value_numeric = get_player
_value_numeric(player_value))
%>% select( player_value_numer
ic)
# A tibble: 30 × 1
   player_value_numeric
                  <dbl>
 1             35000000
 2              1500000
 3              1000000
 4               150000
 5             40000000
 6             35000000
 7              6000000
 8              1000000
 9               600000
10             20000000
```

,



```
> df_bvb_player %>% mutate(age =
parse_number(str_split_i(player_
dob, " ", 2))) %>% select(age)
# A tibble: 30 × 1
     age
   <dbl>
 1    25
 2    22
 3    32
 4    19
 5    23
 6    27
 7    34
 8    19
 9    23
10    28
```

) =



```
> df_bvb_player %>% mutate(p
layer_talent = player_talent
(player_value_numeric, age))
%>% select(player_talent)
# A tibble: 30 × 1
   player_talent
   <chr>
 1 Star
 2 No Category
 3 No Category
 4 No Category
 5 Star
 6 Star
 7 No Category
 8 No Category
 9 No Category
10 Star
```

DataFrame Functions **Embracing**

*“**Embracing** a variable **tells dplyr** to **use** the value stored inside the **argument**, **not** the argument as the **literal variable name**.”*

If you are using **tidyverse verbs** in **functions,** use **embracing** to wrap variables **{{ var }}**.

Common case for:

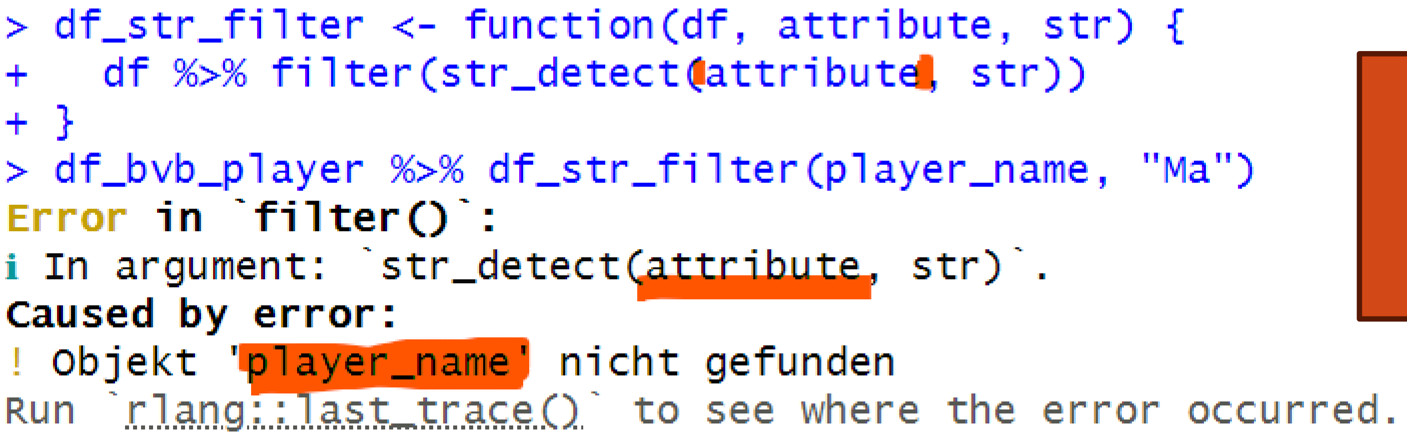
- **Data-masking:** arrange(), filter(), summarize()
- **Tidy-selection**: select(), rename()

Problem arises because **dplyr** uses **tidy evaluation** to allow you to **refer to the names** of **variables inside** your **data frame** without any special treatment.

DataFrame Functions Embracing (cont.)

Example without Embracing

A generic function that filters a data frame based on a string occurence within an attribute:



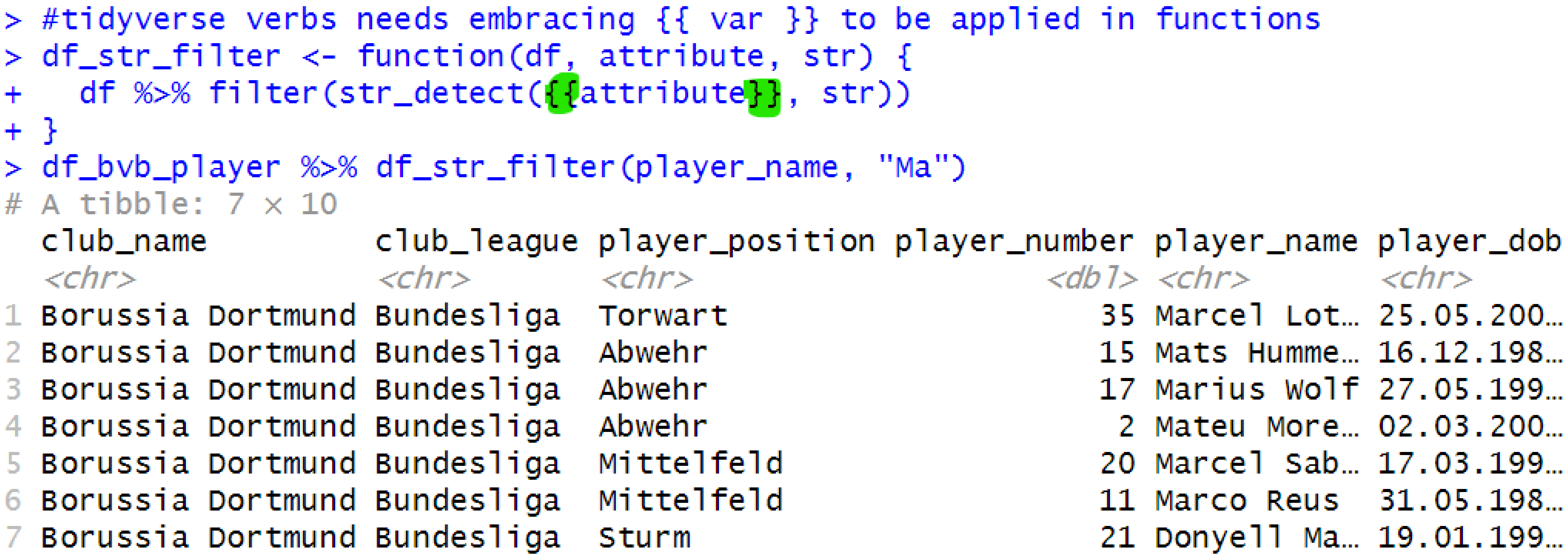
```
> df_str_filter <- function(df, attribute, str) {
+   df %>% filter(str_detect(attribute, str))
+ }
> df_bvb_player %>% df_str_filter(player_name, "Ma")
Error in `filter()`:
i In argument: `str_detect(attribute, str)`.
Caused by error:
! Objekt 'player_name' nicht gefunden
Run `rlang::last_trace()` to see where the error occurred.
```

What happened?
The data frame filters df$attribute
instead df$player_name

DataFrame Functions Embracing (cont.)

Example with Embracing

A generic function that filters a data frame based on a string occurence within an attribute:



```
> #tidyverse verbs needs embracing {{ var }} to be applied in functions
> df_str_filter <- function(df, attribute, str) {
+   df %>% filter(str_detect({{attribute}}, str))
+ }
> df_bvb_player %>% df_str_filter(player_name, "Ma")
# A tibble: 7 × 10
  club_name         club_league player_position player_number player_name player_dob
  <chr>             <chr>       <chr>                   <dbl> <chr>       <chr>     
1 Borussia Dortmund Bundesliga  Torwart                    35 Marcel Lot… 25.05.200…
2 Borussia Dortmund Bundesliga  Abwehr                     15 Mats Humme… 16.12.198…
3 Borussia Dortmund Bundesliga  Abwehr                     17 Marius Wolf 27.05.199…
4 Borussia Dortmund Bundesliga  Abwehr                      2 Mateu More… 02.03.200…
5 Borussia Dortmund Bundesliga  Mittelfeld                 20 Marcel Sab… 17.03.199…
6 Borussia Dortmund Bundesliga  Mittelfeld                 11 Marco Reus  31.05.198…
7 Borussia Dortmund Bundesliga  Sturm                      21 Donyell Ma… 19.01.199…
```

Major Tasks in **Data Preprocessing**

R DataFrame: Selecting

Data Reduction

- Obtains reduced representation in volume but produces the same or similar analytical results.

R DataFrame: Slicing, Filtering, Mutating, Renaming

Data Cleaning

- **Fill in missing values**, **smooth noisy data**, identify or remove outliers, and resolve inconsistencies caused by data integration.

R: Conditional Transformations, Functions

Data Integration

- Integration of multiple tables, databases, data cubes, or files.

Data Transformation

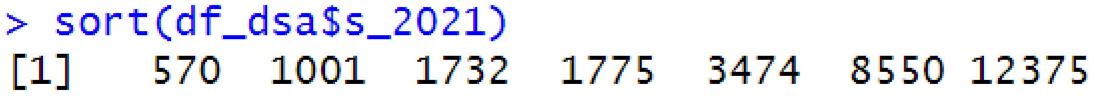
- Aggregation, generalization, normalization and attribute construction.

Data Transformation: **sorting**

https://dplyr.tidyverse.org/reference/arrange.html

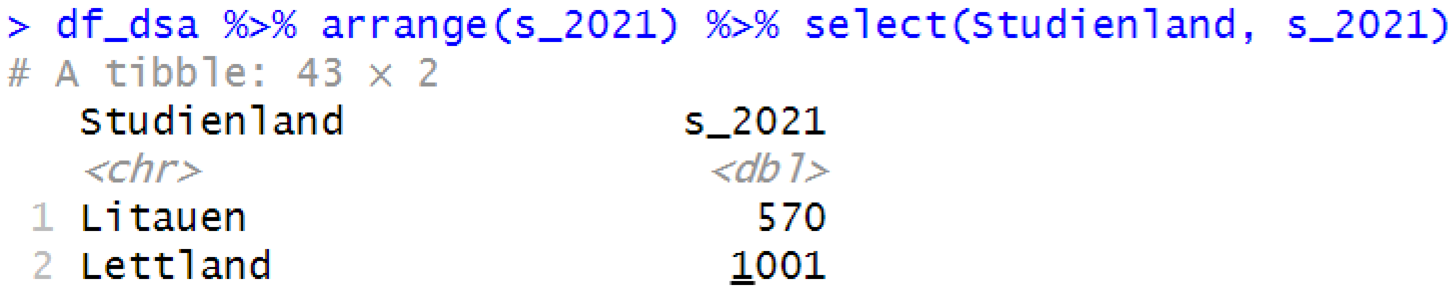
Where did our NAs go?

In R, base sorting of vectors is done via `sort()`:

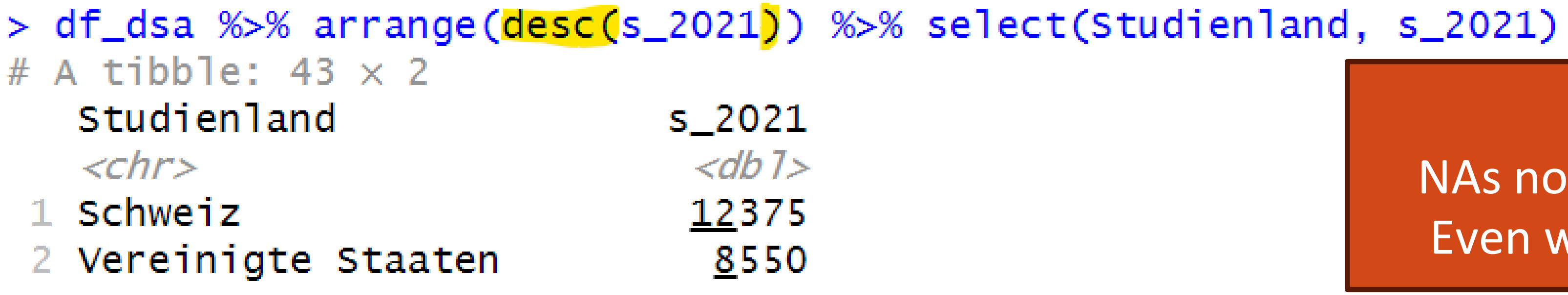


```
> sort(df_dsa$s_2021)
[1]   570  1001  1732  1775  3474  8550 12375
```

For data frames, `arrange()` returns the calling data frame in the order of the selected column(s).



```
> df_dsa %>% arrange(s_2021) %>% select(Studienland, s_2021)
# A tibble: 43 × 2
   Studienland                s_2021
   <chr>                       <dbl>
 1 Litauen                       570
 2 Lettland                     1001
```



```
> df_dsa %>% arrange(desc(s_2021)) %>% select(Studienland, s_2021)
# A tibble: 43 × 2
   Studienland                s_2021
   <chr>                       <dbl>
 1 Schweiz                     12375
 2 Vereinigte Staaten           8550
```

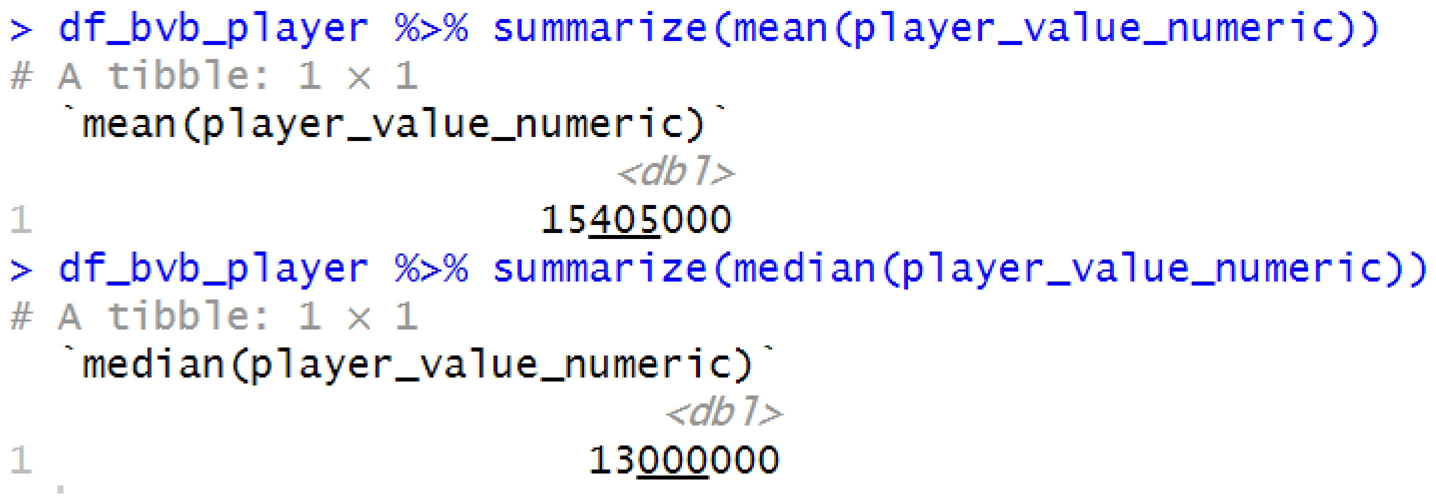
In `arrange()`,
NAs now dropped and always sorted!
Even when wrapped with `desc()`.

Data Transformation: **aggregation**

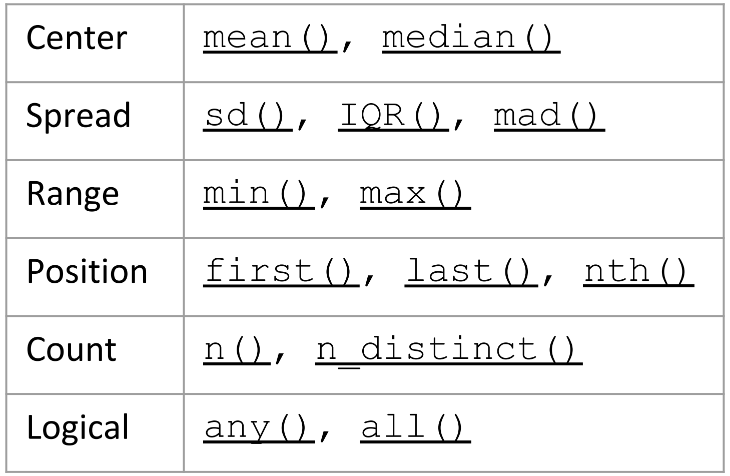
https://dplyr.tidyverse.org/reference/summarise.html

Essential piece of analysis of large data is **efficient summarization.**

For data frames, `summarise()` or `summarize()` creates a **fundamentally** new one based on a numeric summarizing value.

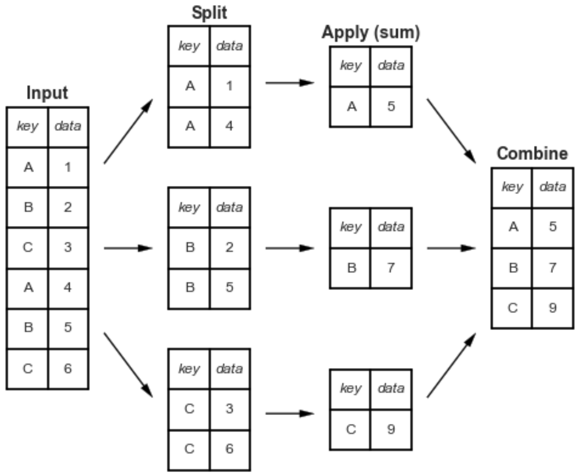


```
> df_bvb_player %>% summarize(mean(player_value_numeric))
# A tibble: 1 × 1
  `mean(player_value_numeric)`
                         <dbl>
1                     15405000
> df_bvb_player %>% summarize(median(player_value_numeric))
# A tibble: 1 × 1
  `median(player_value_numeric)`
                           <dbl>
1                       13000000
```



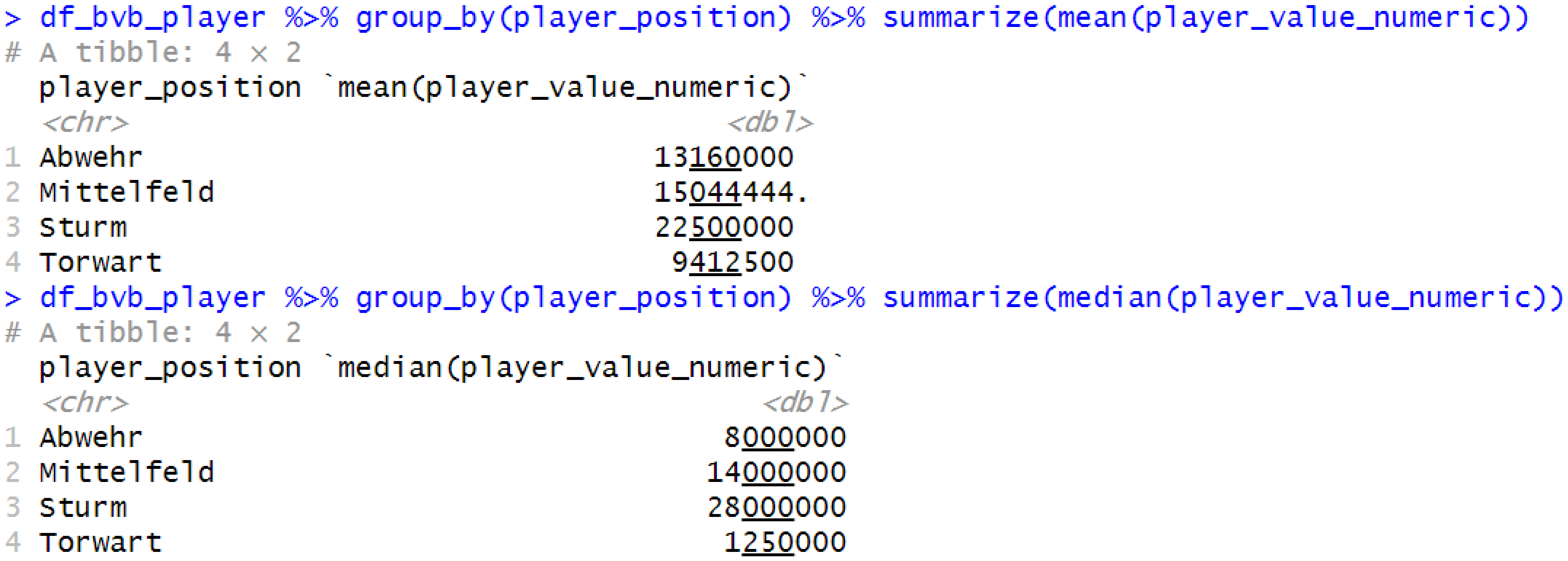
Center	`mean()`, `median()`
Spread	`sd()`, `IQR()`, `mad()`
Range	`min()`, `max()`
Position	`first()`, `last()`, `nth()`
Count	`n()`, `n_distinct()`
Logical	`any()`, `all()`

Data Transformation: **grouping and aggregation**



Hadley Wickham of Rstats fame*: split, apply, combine.*

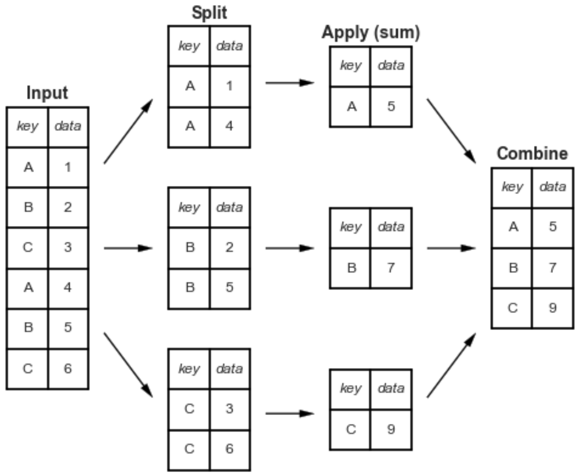
A preceeding `group_by()` internally orders the groups (attribute) in ascending order.



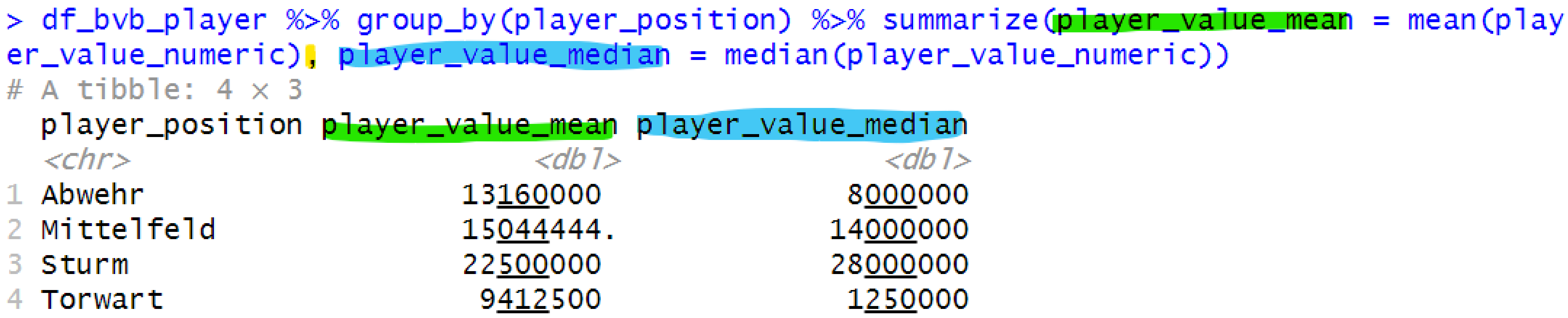
```
> df_bvb_player %>% group_by(player_position) %>% summarize(mean(player_value_numeric))
# A tibble: 4 × 2
  player_position `mean(player_value_numeric)`
  <chr>                                  <dbl>
1 Abwehr                              13160000
2 Mittelfeld                          15044444.
3 Sturm                               22500000
4 Torwart                              9412500
> df_bvb_player %>% group_by(player_position) %>% summarize(median(player_value_numeric))
# A tibble: 4 × 2
  player_position `median(player_value_numeric)`
  <chr>                                    <dbl>
1 Abwehr                                 8000000
2 Mittelfeld                            14000000
3 Sturm                                 28000000
4 Torwart                                1250000
```

Data Transformation: **grouping and aggregation**

A preceeding `group_by()` internally orders the groups (attribute) in ascending order.



Hadley Wickham of Rstats fame*: split, apply, combine.*



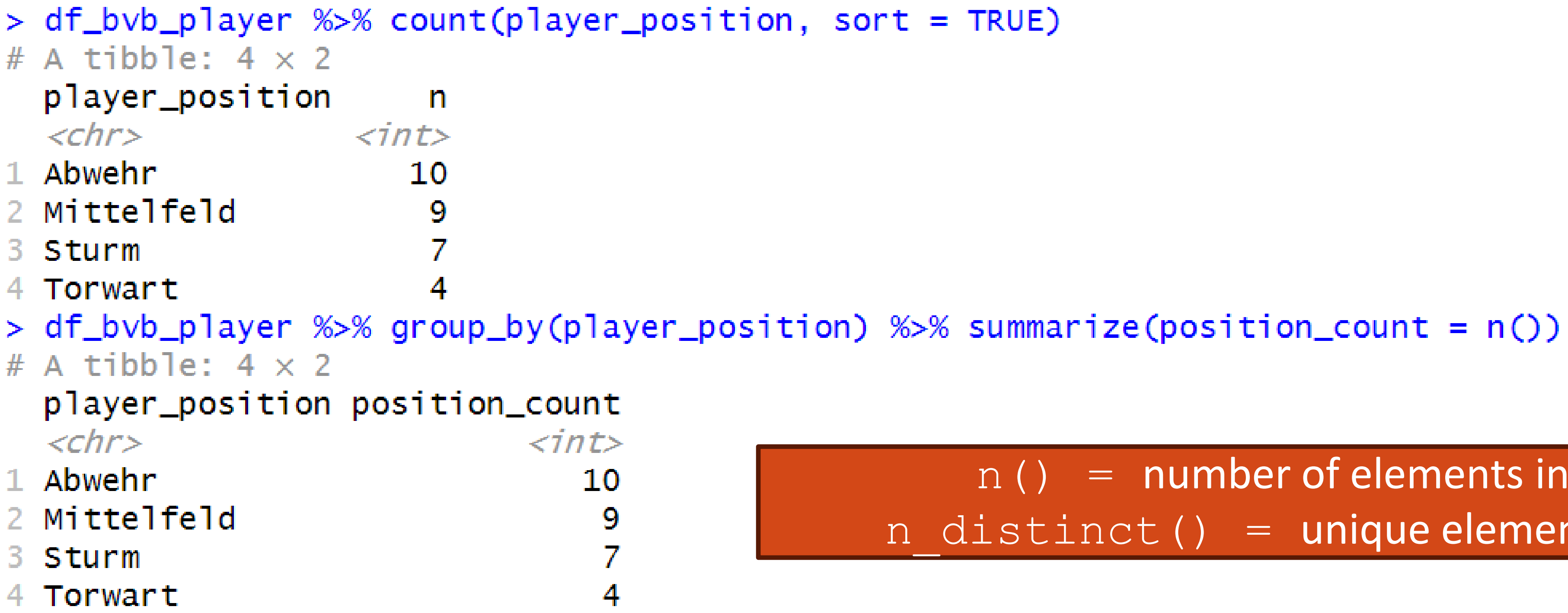
```
> df_bvb_player %>% group_by(player_position) %>% summarize(player_value_mean = mean(play
er_value_numeric), player_value_median = median(player_value_numeric))
# A tibble: 4 × 3
  player_position player_value_mean player_value_median
  <chr>                       <dbl>               <dbl>
1 Abwehr                   13160000             8000000
2 Mittelfeld               15044444.           14000000
3 Sturm                    22500000            28000000
4 Torwart                   9412500             1250000
```

Data Transformation: **count**

https://dplyr.tidyverse.org/reference/summarise.html

Essential piece of analysis of large data is **efficient summarization**.

For data frames, `count()` returns a new data frame with the occurrences.



```
> df_bvb_player %>% count(player_position, sort = TRUE)
# A tibble: 4 × 2
  player_position     n
  <chr>           <int>
1 Abwehr             10
2 Mittelfeld          9
3 Sturm               7
4 Torwart             4
> df_bvb_player %>% group_by(player_position) %>% summarize(position_count = n())
# A tibble: 4 × 2
  player_position position_count
  <chr>                    <int>
1 Abwehr                      10
2 Mittelfeld                   9
3 Sturm                        7
4 Torwart                      4
```

`n()` = number of elements in attribute
`n_distinct()` = unique elements in attribute

Major Tasks in **Data Preprocessing**

Data Reduction

R DataFrame: Selecting

- Obtains reduced representation in volume but produces the same or similar analytical results.

Data Cleaning

R DataFrame: Slicing, Filtering, Mutating, Renaming

- **Fill in missing values**, **smooth noisy data**, identify or remove outliers, and resolve inconsistencies caused by data integration.

R: Conditional Transformations, Functions

Data Integration

- Integration of multiple tables, databases, data cubes, or files.

Data Transformation

R DataFrame: Arrange, Summarize, Grouping

- **Aggregation**, **generalization**, **normalization** and **attribute construction**.

What We Watched: A Netflix Engagement Report
https://about.netflix.com/en/news/what-we-watched-a-netflix-engagement-report

Training #3

1. Use your existing **df_netflix** or import the following .csv dataset: https://github.com/leotraeg/FHDTM-P2DS-WS2425/raw/refs/heads/main/Praktikum/Netflix.csv as a DataFrame called **df_netflix** in R using the readr library.

2. Define a function that computes a **ratio** between **duration** and **Hours_Viewed** with the goal to detect the best performing titles over time.
   - `Mutate()` a new attribute called **duration_numeric** by using `as.numeric(duration, "days")`
   - Write a function **netflix_ratio1** that takes as an argument hours_viewed and duration_numeric and outputs the ratio between those.
   - Apply this function to the df_netflix and create a new attribute called **ratio1**.
   - What title has the highest ratio1 value? *Hint: use arrange.*

Summary

The `%>%` **pipe** operator is essential to **cascadingly manipulate** data frames in R.

R offers a broad range **of predefined functions** and **functional programming** to handle data reduction, cleaning, integration, and transformation.

The **techicallities** of preprocessing and analysing any data (e.g., sport, students, streaming) in Python or R are **secondary**, but **theoretical concepts** of noisy data, missing data, data types, formatting and transformation **remain** the same.

A few features of R that we did not cover:

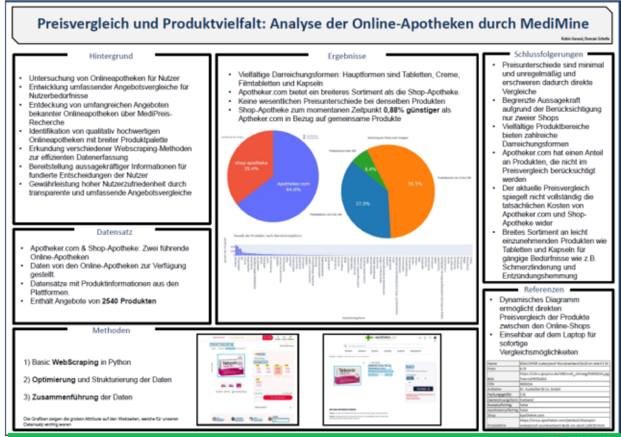
- Factors
- Joins
- Web Scraping

Week 15: Project Day

First hour: students present their project (Milestone III.1) to each other.

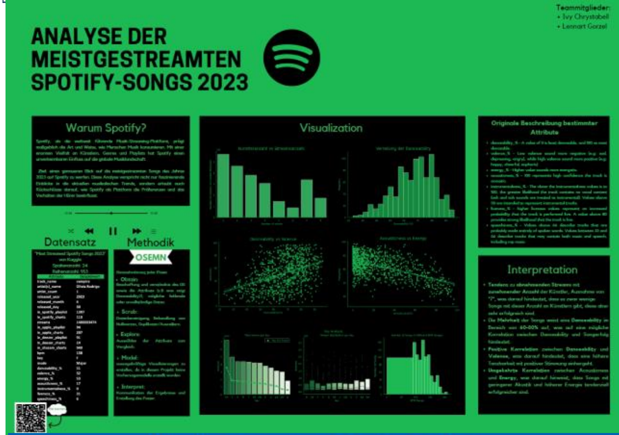
Second hour: graded presentation à 10 minutes for each project.

After lunch: feedback session.



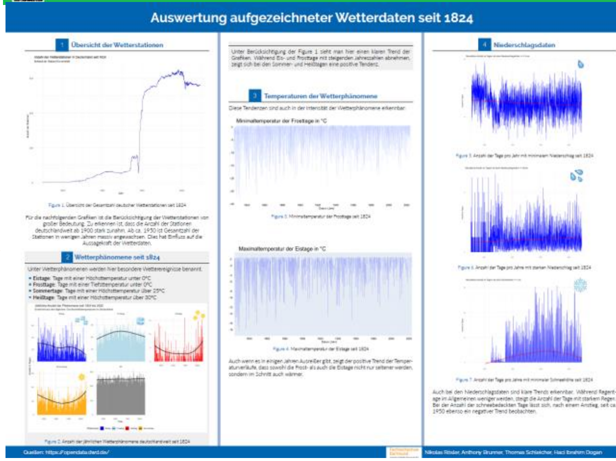
Preisvergleich und Produktvielfalt: Analyse der Online-Apotheken durch MediMine

Hintergrund · Ergebnisse · Schlussfolgerungen · Datensatz · Methoden · Referenzen



ANALYSE DER MEISTGESTREAMTEN SPOTIFY-SONGS 2023

Warum Spotify? · Visualization · Datensatz · Methodik · Interpretation



Auswertung aufgezeichneter Wetterdaten seit 1824

Übersicht der Wetterstationen · Wetterphänomene seit 1824 · Temperaturen der Wetterphänomene · Niederschlagsdaten

project posters from previous years

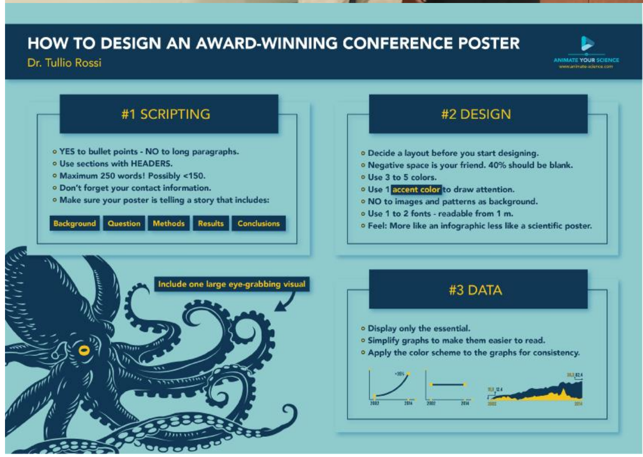
Guidance:

- guides.nyu.edu/posters or
- colinpurrington.com/tips/poster-design/

Project Day (cont.)

Over the past weeks, you became an expert on a topic, dataset, achieved project milestones and got through technical and team challenges.

- Try to be clear, direct, and **authentic** about the storyline ☺
- **Do not be reserved** when presenting your project ☺



HOW TO DESIGN AN AWARD-WINNING CONFERENCE POSTER

Dr. Tullio Rossi

ANIMATE YOUR SCIENCE
www.animate-science.com

#1 SCRIPTING

- YES to bullet points - NO to long paragraphs.
- Use sections with HEADERS.
- Maximum 250 words! Possibly <150.
- Don't forget your contact information.
- Make sure your poster is telling a story that includes:

Background | Question | Methods | Results | Conclusions

#2 DESIGN

- Decide a layout before you start designing.
- Negative space is your friend. 40% should be blank.
- Use 3 to 5 colors.
- Use 1 accent color to draw attention.
- NO to images and patterns as background.
- Use 1 to 2 fonts - readable from 1 m.
- Feel: More like an infographic less like a scientific poster.

Include one large eye-grabbing visual

#3 DATA

- Display only the essential.
- Simplify graphs to make them easier to read.
- Apply the color scheme to the graphs for consistency.

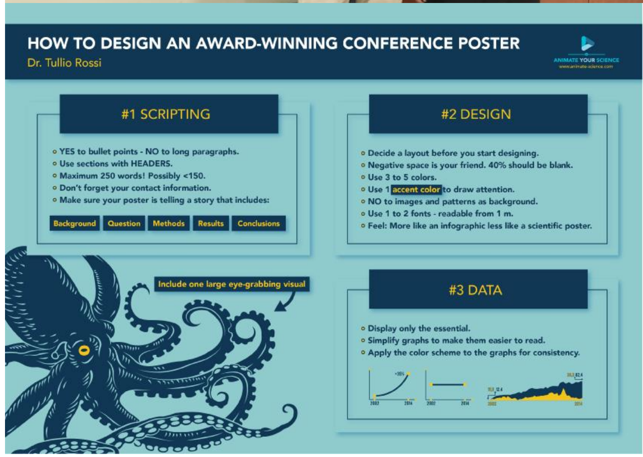
Project Day (cont.)

In person event 06.01.2024.

You will have the opportunity to revise your poster based on (my) feedback ahead of the 11:59pm deadline. I will grade the final digital version.

Posters will be evaluated with equal weight on:

- Informativeness
- Attractiveness
- Understandability
- Delivery of presentation
- Technical merit



HOW TO DESIGN AN AWARD-WINNING CONFERENCE POSTER

Dr. Tullio Rossi

ANIMATE YOUR SCIENCE
www.animate-science.com

#1 SCRIPTING

- YES to bullet points - NO to long paragraphs.
- Use sections with HEADERS.
- Maximum 250 words! Possibly <150.
- Don't forget your contact information.
- Make sure your poster is telling a story that includes:

Background | Question | Methods | Results | Conclusions

#2 DESIGN

- Decide a layout before you start designing.
- Negative space is your friend. 40% should be blank.
- Use 3 to 5 colors.
- Use 1 accent color to draw attention.
- NO to images and patterns as background.
- Use 1 to 2 fonts - readable from 1 m.
- Feel: More like an infographic less like a scientific poster.

Include one large eye-grabbing visual

#3 DATA

- Display only the essential.
- Simplify graphs to make them easier to read.
- Apply the color scheme to the graphs for consistency.

See you again at the poster session!

Feel free to email me for questions or schedule a meeting.

Enjoy holidays ☺