Advanced object types

06.04.2022, Data Science (SpSe 2022): T5

Prof. Dr. Claudius Gräbner-Radkowitsch
Europa-University Flensburg, Department of Pluralist Economics
www.claudius-graebner.com | @ClaudiusGraebner | claudius@claudius-graebner.com

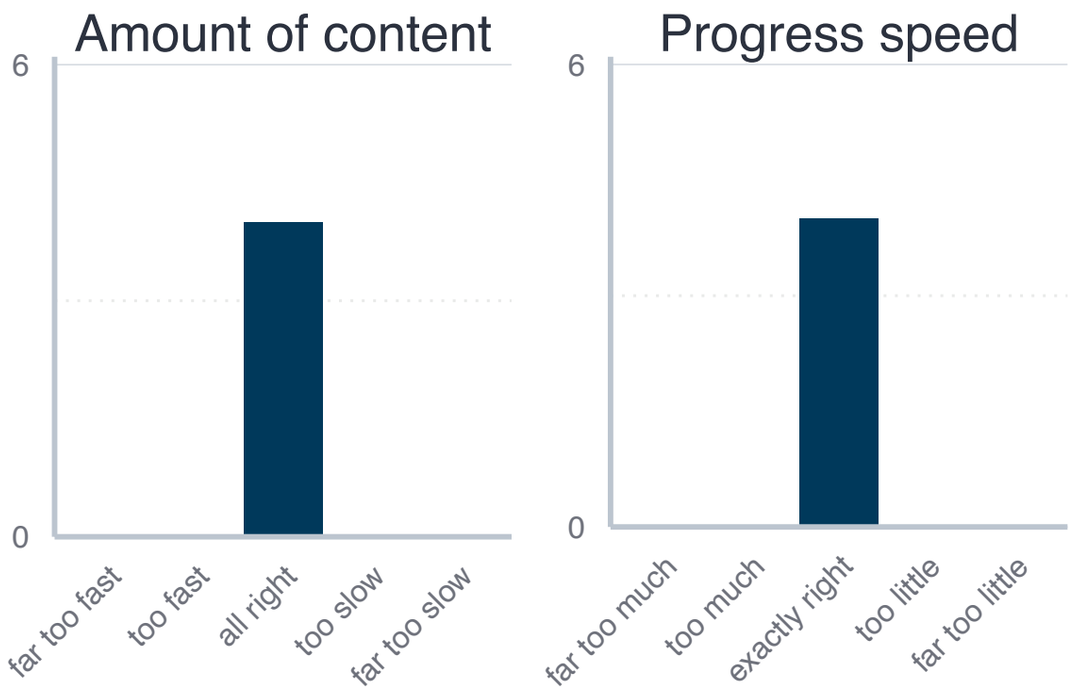
Europa-Universität Flensburg

Europa-Universität Flensburg
International Institute of Management and Economic Education
Department of Pluralist Economics

Prologue:

Prologue
Feedback and exercises

- Five of you filled out the feedback survey. Main results:

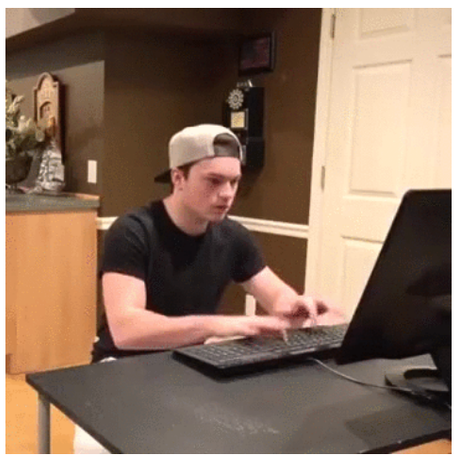


Highlights

- Hierarchy of object types
- How to identify and coerce the different object types

Lowlights

- Lists: hard to understand about the area of application



- Make sure you post questions on **exercises** in the Moodle forum!

Applications for lists

- Lists are important mainly for two reasons:
  - First, they allow us to create sets of objects of different types
  - Second, they are the base for more complex data structures, esp. `data.frame`s

- We will deal with the second case later, here an example for the first case:

```
experiment_result <- list(
        "name_participant"=c("Ellie", "Abby", "Dina"),
        "height"=c(175L, 179L, 169L),
        "vaccinated"=c(TRUE, FALSE, FALSE))
```

- Here we want to store data formalized as different types in one single object – no alternative to a `list`!

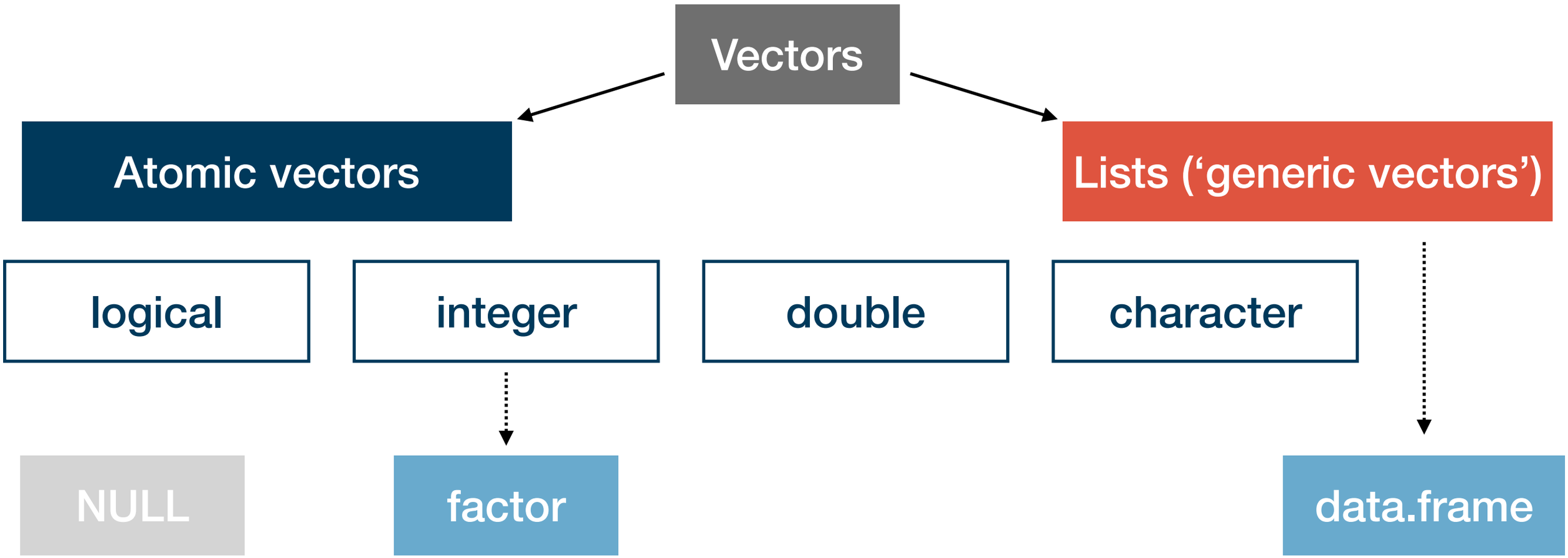
Goals for today

I. Learn about three advanced object types: `factors`, `data.frames`, and `tibbles`

II. Understand and their relation to the basic types discussed previously

Advanced object types in R

Summary and outlook



On more advanced object types

- While there are many object types in R, understanding the basics is key
  - These are by far the most common ones
  - All other object types are somehow ‘built upon’ the basic types by adding **attributes**
- Among the special types, two stand out in their prevalence:

Categorical data: factor

- Can also take a pre-specified number of values: levels
- Classical example: Male, Female, Diverse
- Created using the function `factor()`

Data frames: data.frame & tibble

- A kind of ‘table’ in which different variables are stored as vectors
- A table-like form of
- Tibbles as a new v
that “do less and c
- Created using `data`
`tibble::tibble(`

	gender	height
1	male	189
2	male	175
3	male	180
4	female	166
5	female	150

- Others that we will not cover here are, e.g., matrices, durations, or dates

Digression: some remarks on attributes

- To turn our basic object types into something more fancy we can give them **attributes**, one of which is called **class**
  - This changes their behaviour when functions are applied to them
  - Technically, adding a class attribute changes the class but not the type:

```
ff <- factor(c("F", "M", "M"), levels = c("F", "M", "D"))
typeof(ff)
class(ff)
```

- The class `factor` is an integer with two attributes:

```
attributes(ff)
```

- Not too important for us right now, but good to keep in mind!

Factors

- Factors are used to represent ordinal or categorial data
  - Elements of factors can take one out of several pre-specified values: levels
  - Factors are integers with the attributes `levels` and `class`
- We create factors using the function `factor()`, which takes a vector and an optional argument levels:

```
f_1 <- factor(c(rep("F", 4), rep("D", 5), rep("M", 3)),
              levels = c("D", "F", "M"))
```

Your turn

- What happens if we do not specify `levels` explicitly?
- What happens if the vector contains elements not pre-specified as levels?

Factors

- Usually levels are not ordered, but for ordinal data you can use the argument ordered:

```
f_2 <- factor(c("high", "high", "low"),
              levels = c("low", "mid", "high"),
              ordered=TRUE)
```

- There are some useful factor-specific functions such as `table()`.
  - What does it do? Try it on `f_1` and `f_2`!
- In general, its usually better to store categorial data as character, and only transform them to factors if necessary

Data frames

- Data frames are special lists of vectors where the length of each vector is equal!

  → Most list operations also work for data.frames

- We usually represent data frames as tables:

	gender	height
1	male	189
2	male	175
3	male	180
4	female	166
5	female	150

Names of the vectors

vector 1 & vector 2

- To create a data frame from scratch use `data.frame()`:

```
df_1 <- data.frame(
  "gender" = c(rep("male", 3), rep("female", 2)),
  "height" = c(189, 175, 180, 166, 150)
)
```

- To create a data frame from a list use `as.data.frame()`

- If you read in data into R, it almost always starts off as a data.frame
- How to transform them is the main subject of the sessions on **data wrangling**

Data frames and tibbles

- A modern version of the data.frame is the `tibble` (from the package *tibble*)
  - We will mostly use `tibbles` in this course, but make sure you familiarise yourself with the differences to the `data.frame`, which continues to be widespread (see the tutorial reading)
- To transform a `data.frame` (or a `list`) into a `tibble`, use `tibble::as_tibble()`:

```
tb_1 <- tibble::as_tibble(df_1)
```

- To extract single columns use the `[` or `[[` operators
  - What's the difference between the two?
  - How do you think you cam test for the type of a column vector?

Data frames and tibbles

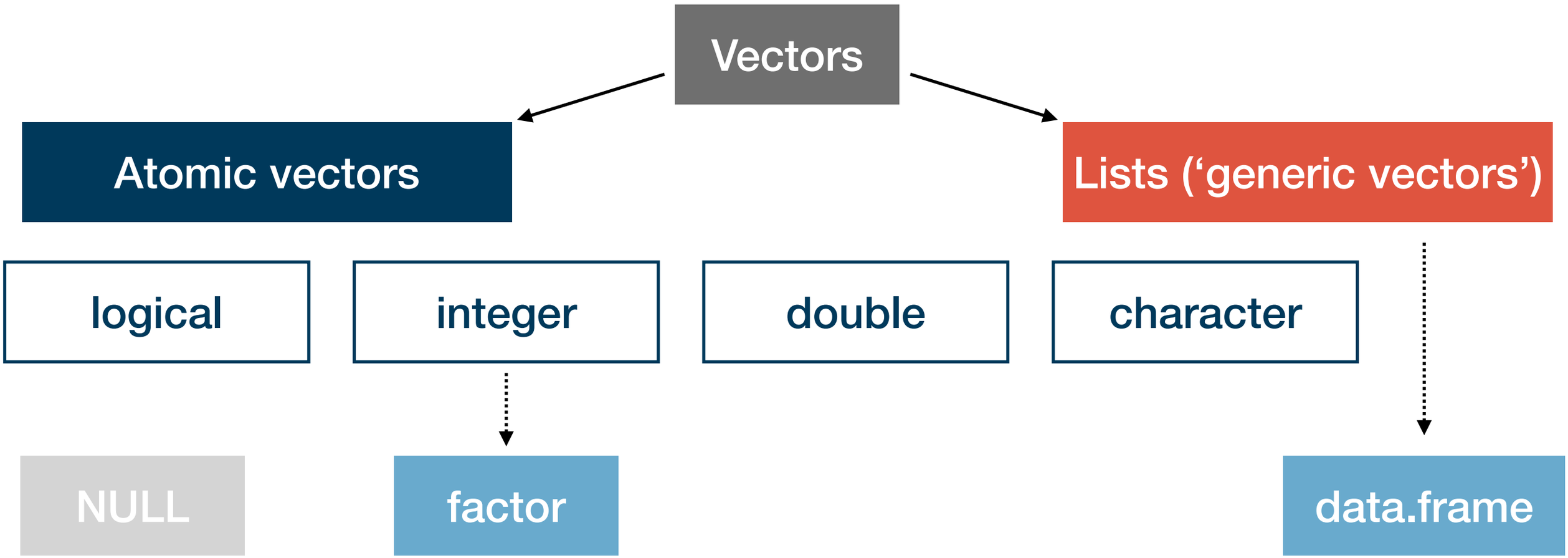
- To get a quick overview about the content, use `dplyr::glimpse()` or `head()`
- A complete overview can be obtained via `View()`
- Data frames are among the most widely used data types
  - There different approaches of how to handle and transform them, each associated with an R **dialect**
  - We mainly rely on the **tidyverse** dialect, which is the easiest to learn and comprehend → built upon **tibbles**
  - Alternatives are the **base** (classical) and **data.table** (fastest) dialect, which mainly use **data.frames** and **data.tables**
- This is useful to keep in mind when searching help in the internet

Quick exercises

- Create a factor with the levels “still”, “medium” and “sparkling”, and arbitrary instances of the three levels
- Transform this factor into a character vector
- Get the relative frequencies for “medium” of this factor
- Create a data frame with two columns, one called “nb” containing the numbers `1` to `5` as double, the other called “char” containing the numbers `6` to `10` as character
- Transform this data frame into a tibble!
- Extract the second column of this tibble such that you have a vector

Summary and outlook

- This was the last session on the fundamentals of R
- We learned about the most important object types in R
- Functions do different things when applied to different objects → understanding object types is absolutely fundamental



Summary and outlook

- Next week we will learn how to visualise data that is stored in data frames
- This will be the first big intro session into data science fundamentals, succeeded by sessions on data wrangling and project management

Tasks until next week:

1. Fill in the **quick feedback survey** on Moodle
2. Read the **tutorials** posted on the course page
3. Do the **exercises** provided on the course page and **discuss problems** and difficulties via the Moodle forum