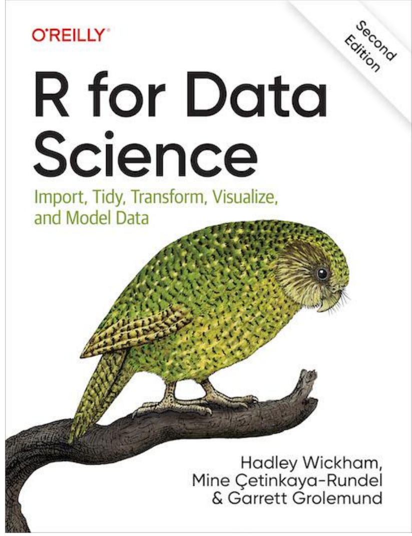
R I

Programmierkurs 2 Data Science WS24/25

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


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

Disclaimer

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

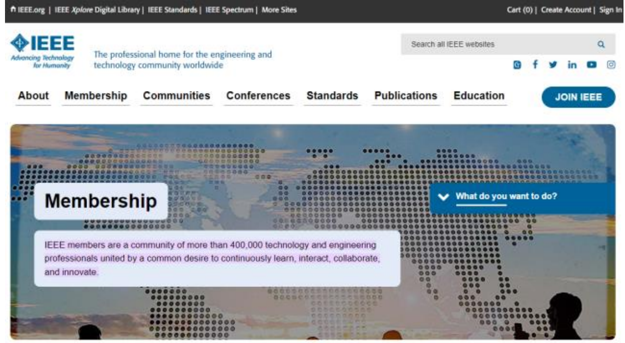
Data Transformation with dplyr:

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

Learning Goals R I

- **Explain** your personal preference of R and Python as a programming language given by giving a comparitive coding example.
- **List** the development stack and its components for programming in R.
- Install and Import libraries and **use** R as a calculator.
- **Create** variables, vectors, matricies, and simple scatter plots.
- Import tabular files as DataFrames and **apply** exploration, filtering, slicing, selection, mutation, and renaming opreations using the pipe syntax.

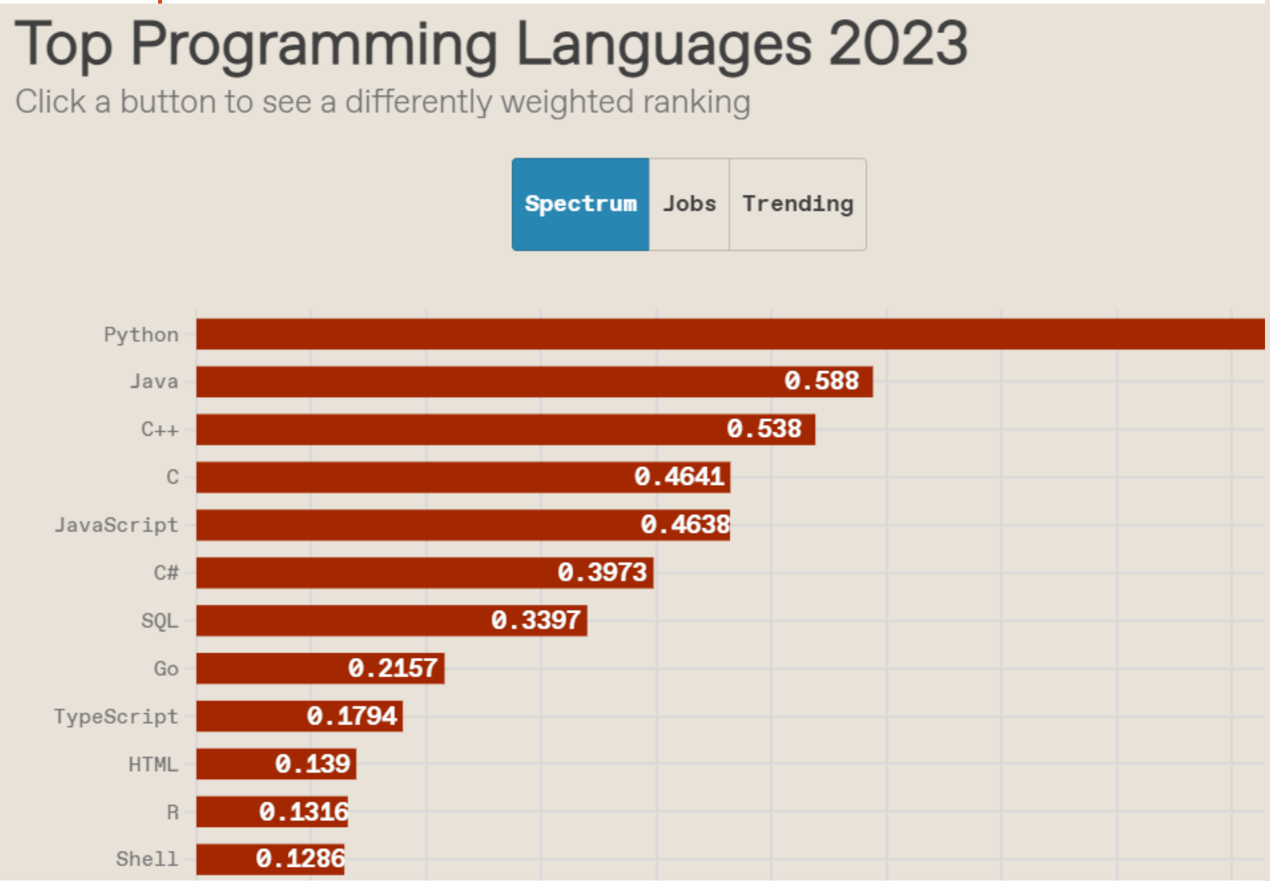
Top Programming Languages



IEEE.org | IEEE Xplore Digital Library | IEEE Standards | IEEE Spectrum | More Sites
Cart (0) | Create Account | Sign In
IEEE
Advancing Technology for Humanity
The professional home for the engineering and technology community worldwide
Search all IEEE websites
About
Membership
Communities
Conferences
Standards
Publications
Education
JOIN IEEE
What do you want to do?

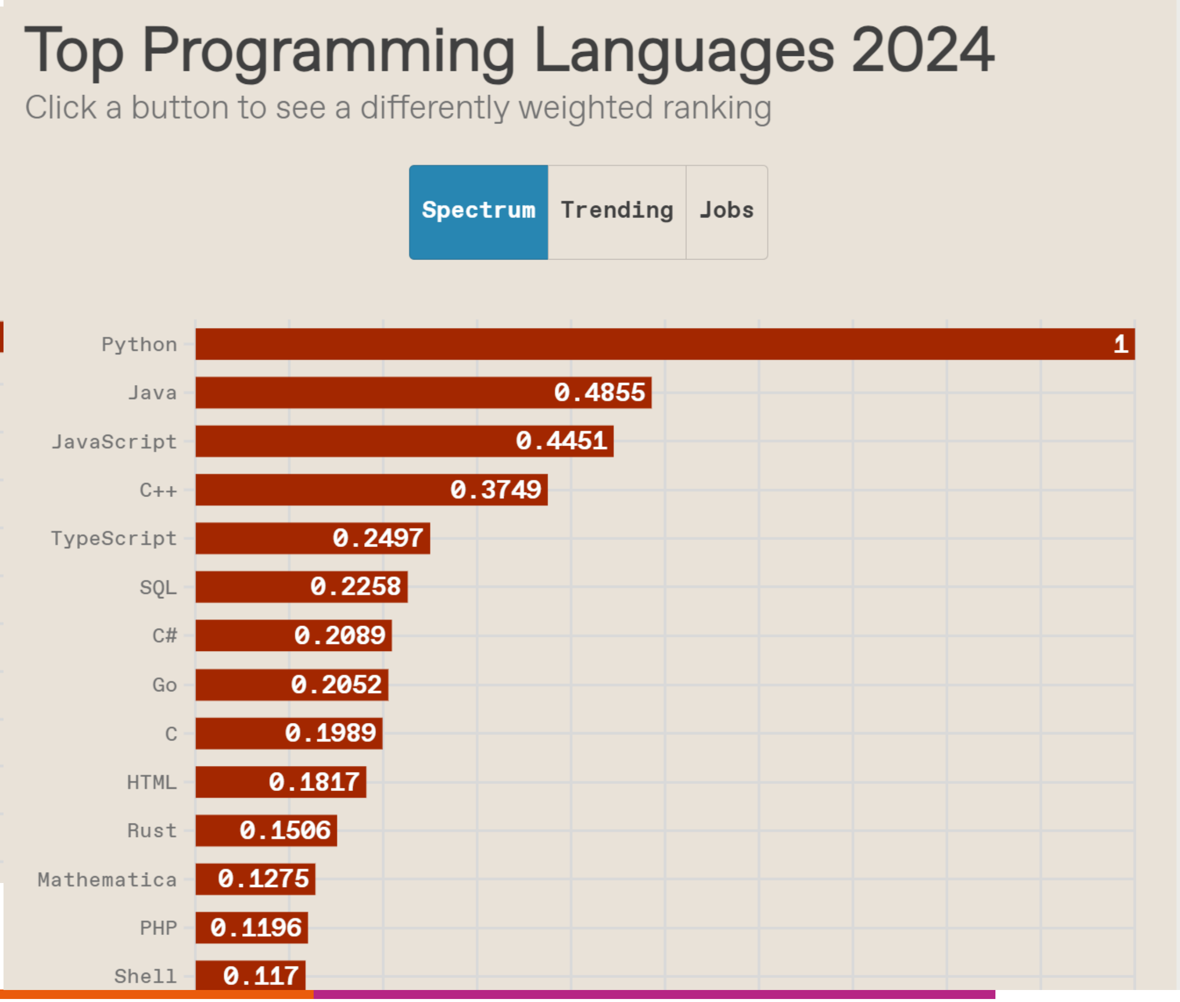
Membership

IEEE members are a community of more than 400,000 technology and engineering professionals united by a common desire to continuously learn, interact, collaborate, and innovate.



Top Programming Languages 2023

Click a button to see a differently weighted ranking

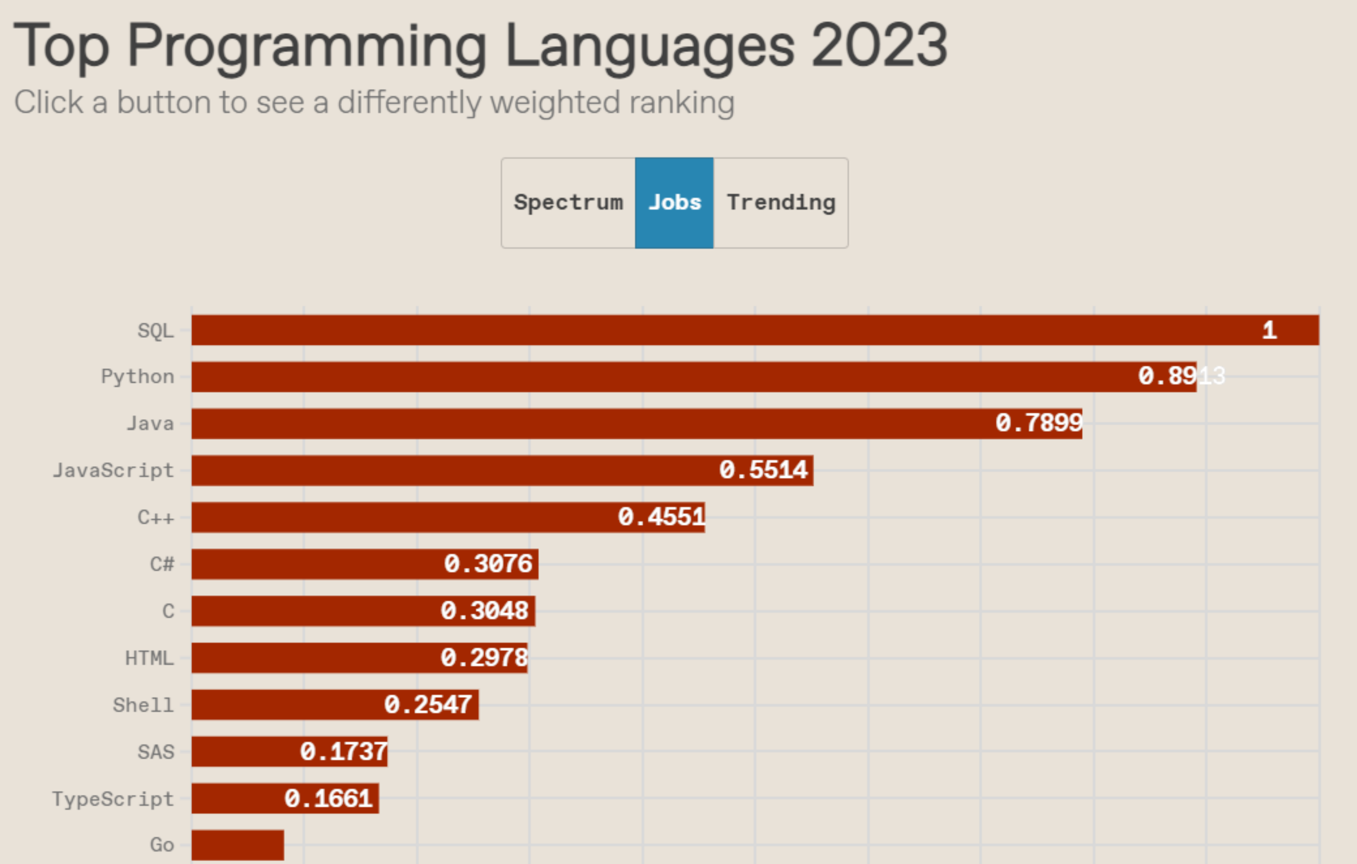


Top Programming Languages 2024

Click a button to see a differently weighted ranking

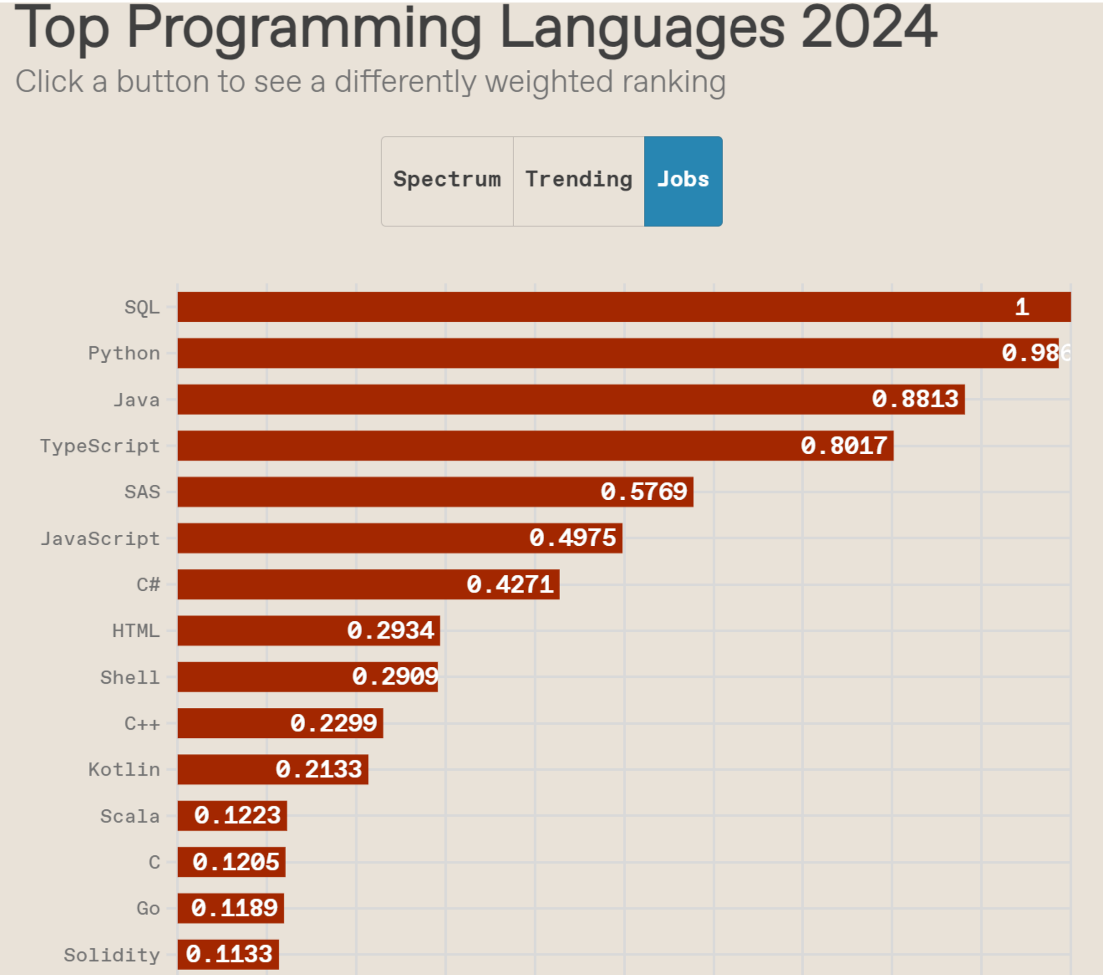
https://spectrum.ieee.org/top-programming-languages-2024

Top Programming Languages



Top Programming Languages 2023

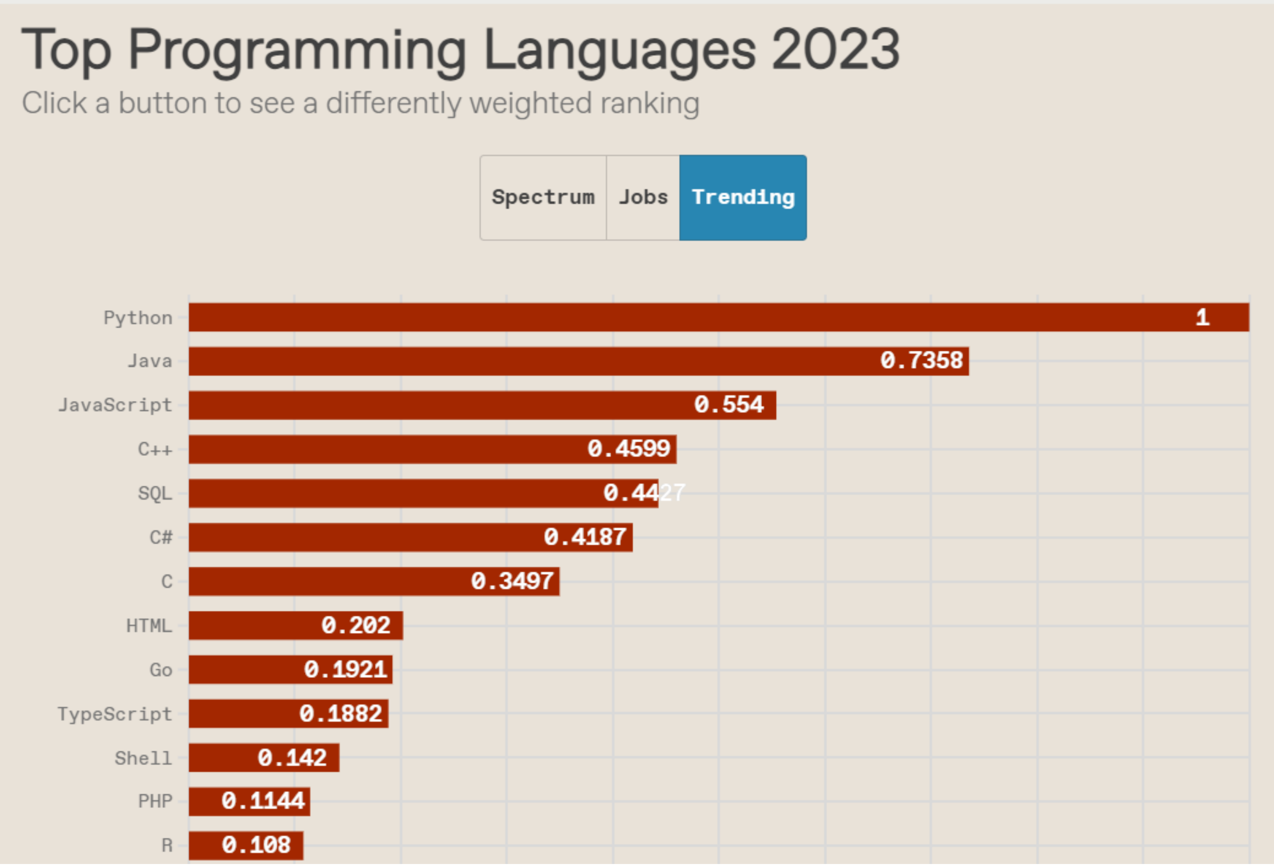
Click a button to see a differently weighted ranking



Top Programming Languages 2024

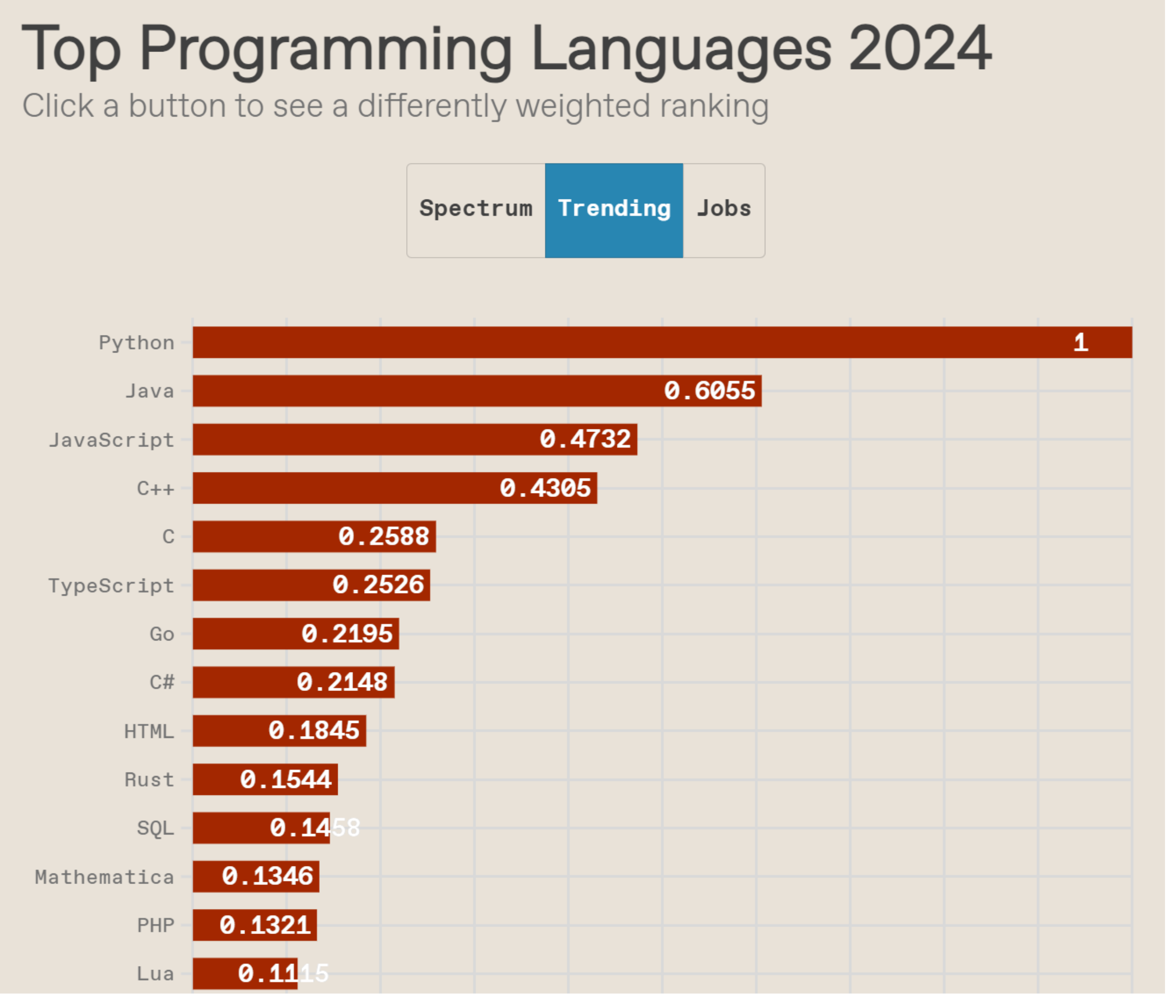
Click a button to see a differently weighted ranking

Top Programming Languages



Top Programming Languages 2023

Click a button to see a differently weighted ranking



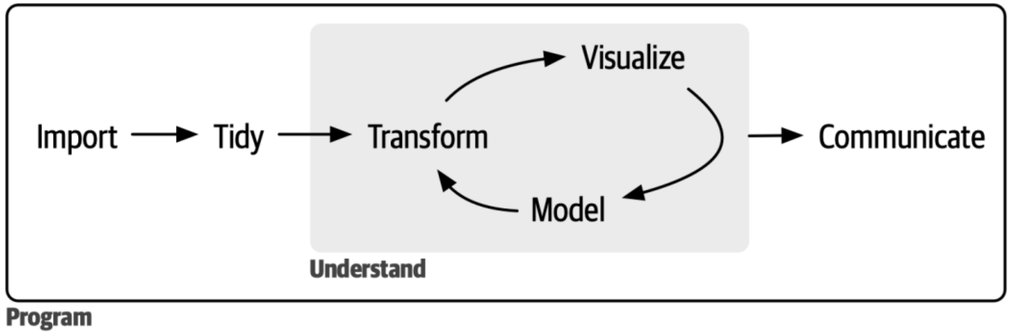
Top Programming Languages 2024

Click a button to see a differently weighted ranking

Python is considered to be more of a **general purpose programming language** and suited for **Machine Learning**.

Why R?

- Built to demonstrate the results of statistical analysis quickly.
- Suited for **statistical learning**.
- High level language; used by “non-techy” engineers and scientists.
- Open source, fast growing ecosystem with packages for almost everything in DS:



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

“Much like picking skis or snowboards, try them both and go with the one that feels right for the way you work.”

Python versus R

	Python	R
Do you have programming experience?	☺	
Do you care about visualization and graphics?		☺
Do you want to apply Statistical Models?		☺
Do you want to apply Machine Learning?	☺	
What do your colleagues, peers, advisors, industry-area use?	☺	☺

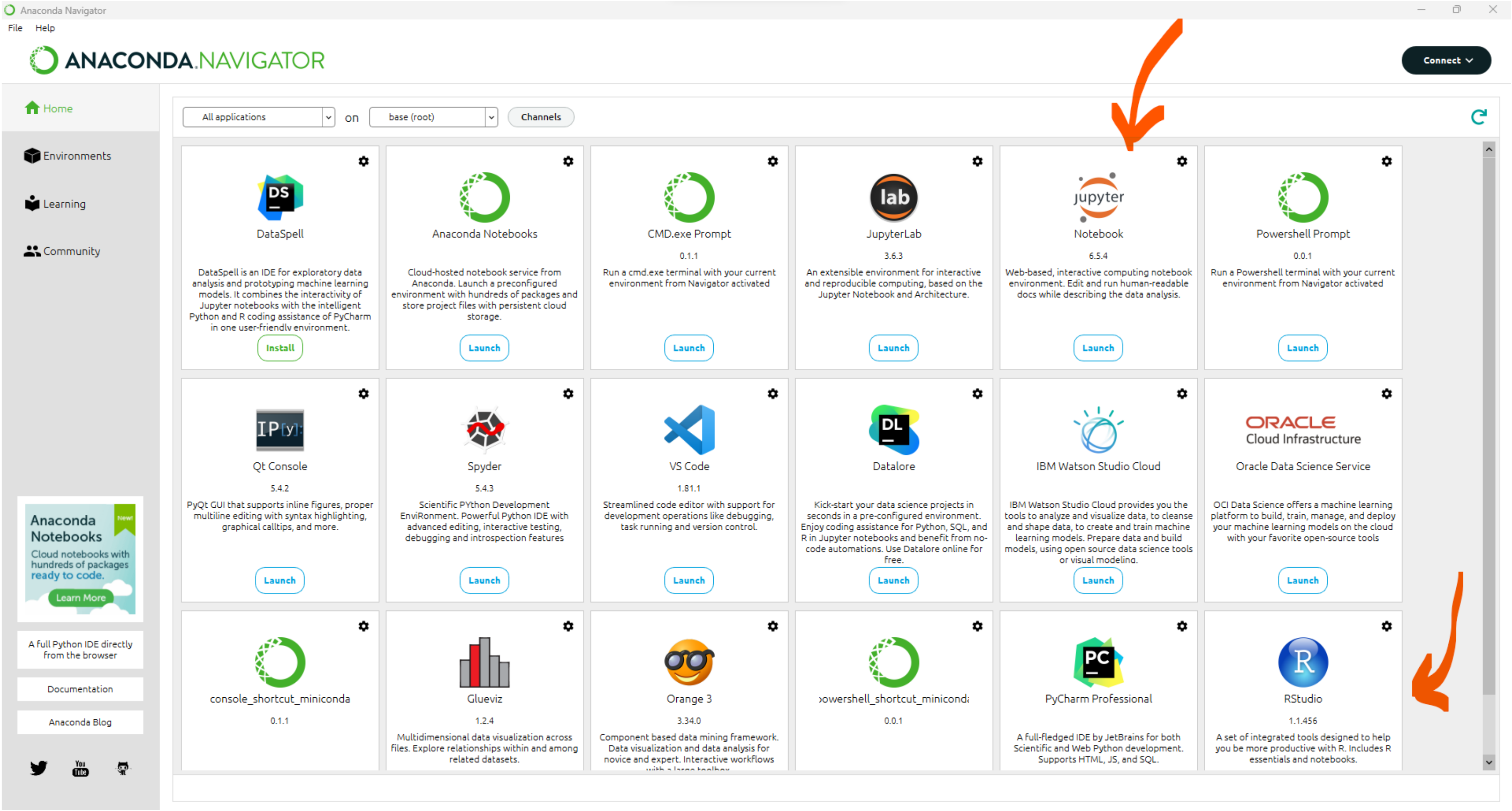
Various decision factors: *data collection, libraries, scale, integration, and many more...*

R set-up alternatives in this class

1. Anaconda
2. Local Installation of R (Software) and RStudio (IDE)
3. Notebook FH Dortmund laboratory - link does not work ☹

Anaconda

https://www.anaconda.com/



DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.

Cloud-hosted notebook service from Anaconda. Launch a preconfigured environment with hundreds of packages and store project files with persistent cloud storage.

Run a cmd.exe terminal with your current environment from Navigator activated

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Run a Powershell terminal with your current environment from Navigator activated

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Streamlined code editor with support for development operations like debugging, task running and version control.

Kick-start your data science projects in seconds in a pre-configured environment. Enjoy coding assistance for Python, SQL, and R in Jupyter notebooks and benefit from no-code automations. Use Datalore online for free.

IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

OCI Data Science offers a machine learning platform to build, train, manage, and deploy your machine learning models on the cloud with your favorite open-source tools

Cloud notebooks with hundreds of packages ready to code.

Multidimensional data visualization across files. Explore relationships within and among related datasets.

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

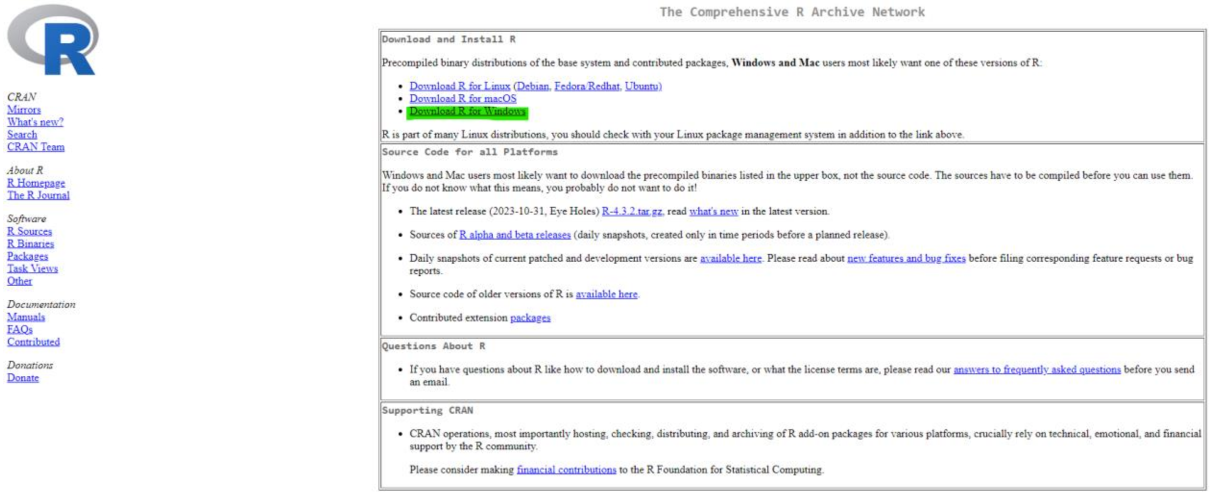
A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

R wird in Woche 11-12 behandelt

Local Install R

Statistik-Software R: https://ftp.fau.de/cran/



The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2023-10-31, Eye Holes) R-4.3.2.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

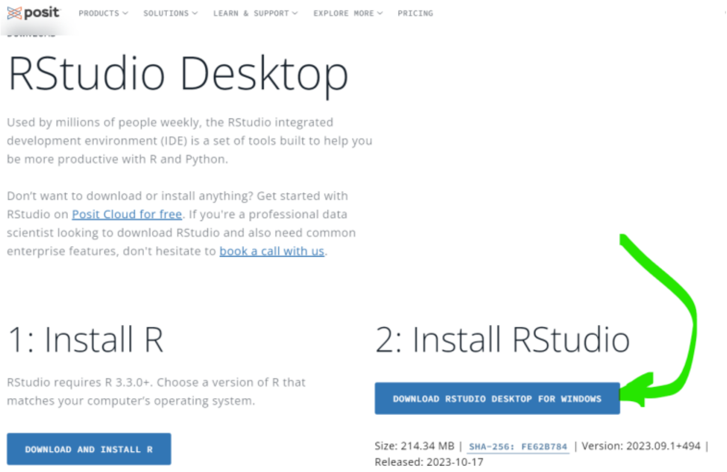
- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

Supporting CRAN

- CRAN operations, most importantly hosting, checking, distributing, and archiving of R add-on packages for various platforms, crucially rely on technical, emotional, and financial support by the R community.

  Please consider making financial contributions to the R Foundation for Statistical Computing.

RStudio Desktop: https://posit.co/download/rstudio-desktop/#download



RStudio Desktop

Used by millions of people weekly, the RStudio integrated development environment (IDE) is a set of tools built to help you be more productive with R and Python.

Don't want to download or install anything? Get started with RStudio on Posit Cloud for free. If you're a professional data scientist looking to download RStudio and also need common enterprise features, don't hesitate to book a call with us.

1: Install R

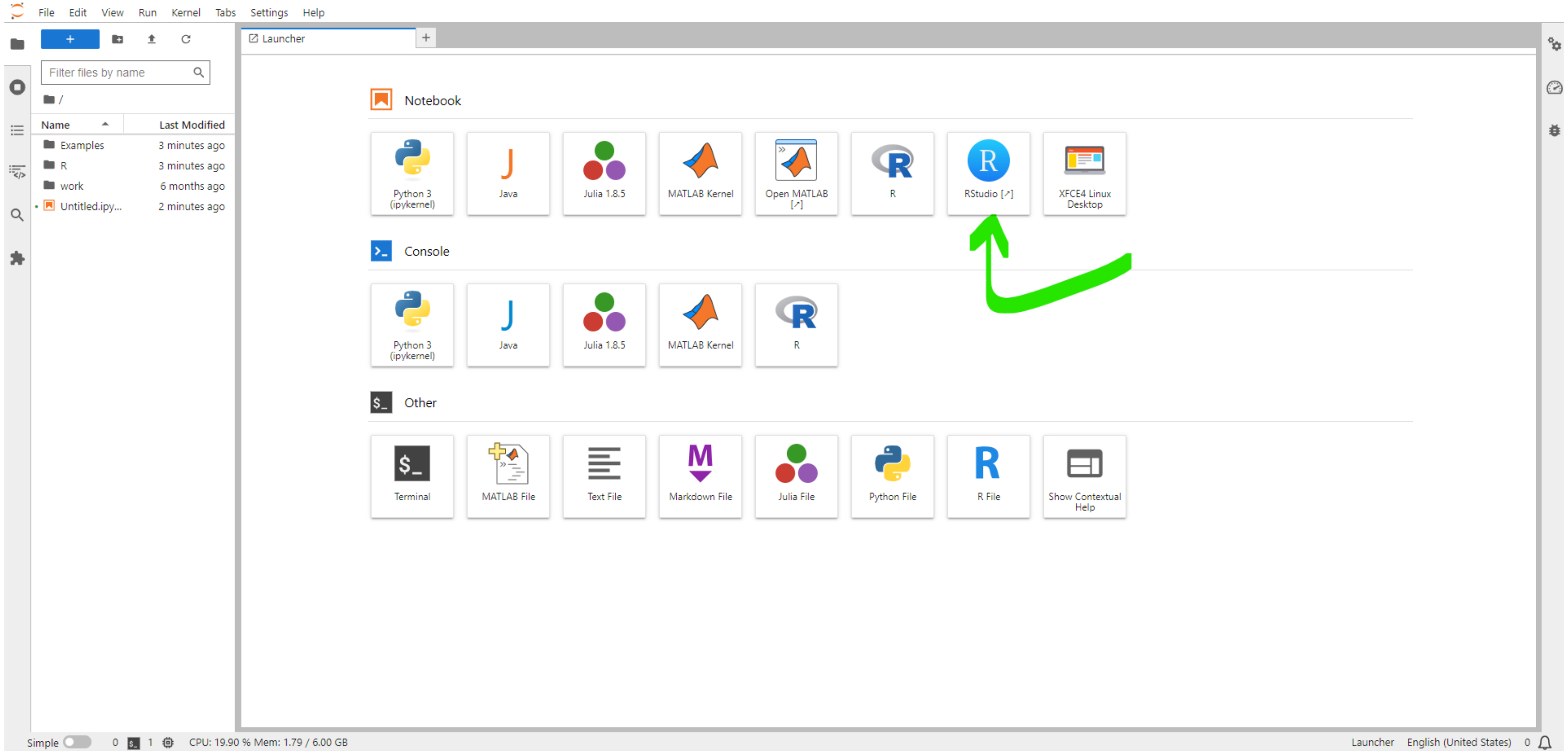
RStudio requires R 3.3.0+. Choose a version of R that matches your computer's operating system.

2: Install RStudio

Size: 214.34 MB | SHA-256: FE62B784 | Version: 2023.09.1+494 | Released: 2023-10-17

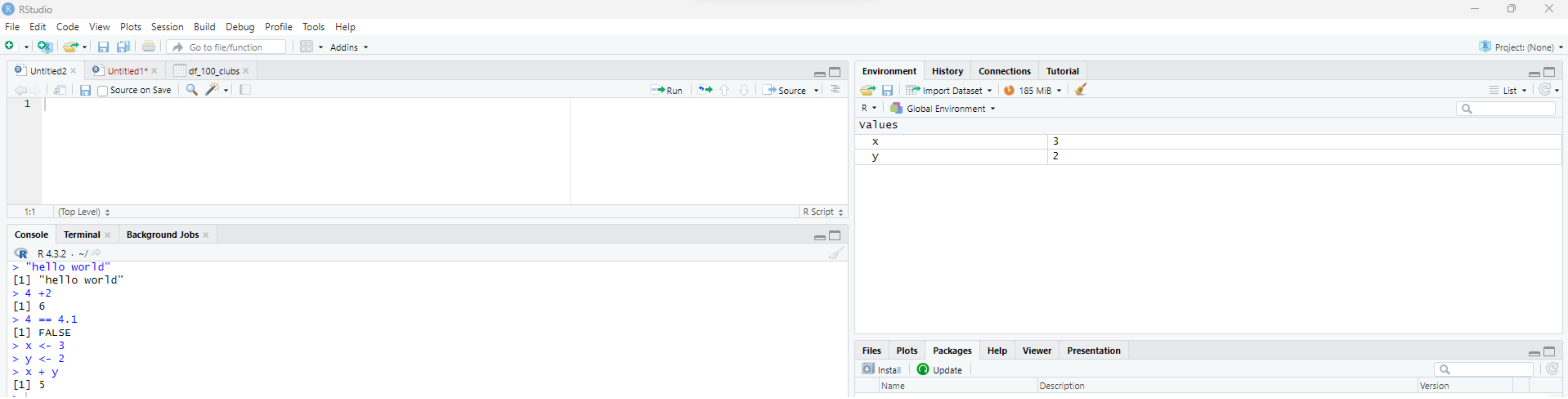
Technology Arts Sciences
TH Köln

https:// jup.labs.inf.fh-dortmund.de/



Name	Last Modified
Examples	3 minutes ago
R	3 minutes ago
work	6 months ago
Untitled.ipy...	2 minutes ago

Check RStudio after installation



```
> "hello world"
[1] "hello world"
> 4 +2
[1] 6
> 4 == 4.1
[1] FALSE
> x <- 3
> y <- 2
> x + y
[1] 5
```

```
values
  x    3
  y    2
```

We will learn more about the syntax of R next week ☺

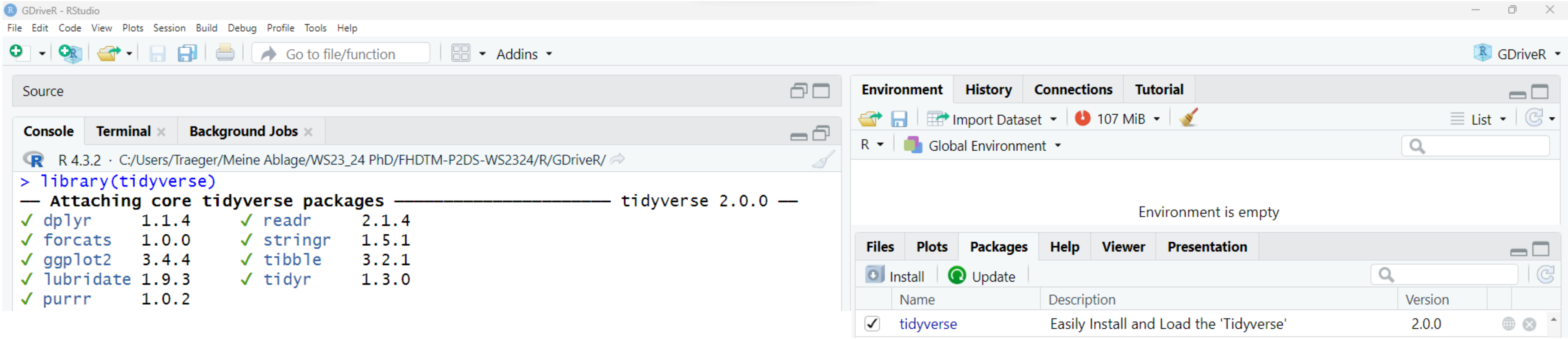
What you will need

R

RStudio: IDE for R programming

Tidyverse: a collection of R packages

- `install.packages("tidyverse")`
- `library("tidyverse")`



R 4.3.2 · C:/Users/Traeger/Meine Ablage/WS23_24 PhD/FHDTM-P2DS-WS2324/R/GDriveR/

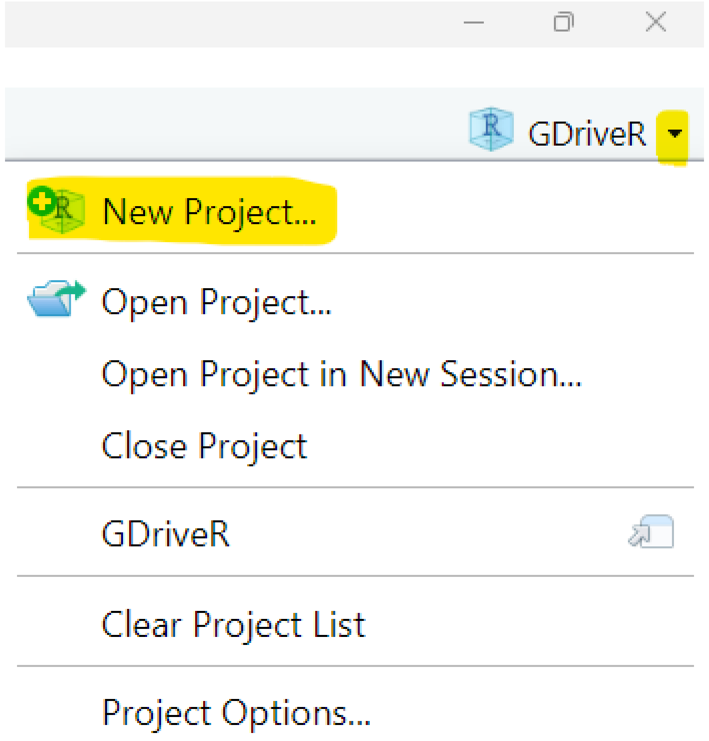
```
> library(tidyverse)
── Attaching core tidyverse packages ─────────────────── tidyverse 2.0.0 ──
✓ dplyr     1.1.4     ✓ readr     2.1.4
✓ forcats   1.0.0     ✓ stringr   1.5.1
✓ ggplot2   3.4.4     ✓ tibble    3.2.1
✓ lubridate 1.9.3     ✓ tidyr     1.3.0
✓ purrr     1.0.2
```

Environment is empty

	Name	Description	Version
☑	tidyverse	Easily Install and Load the 'Tidyverse'	2.0.0

Project in RStudio

File > New Project > Choose Directory



Advantages

- Restores the state of work where you left off.
- Files that you save during the course can be easily opened via the panel.

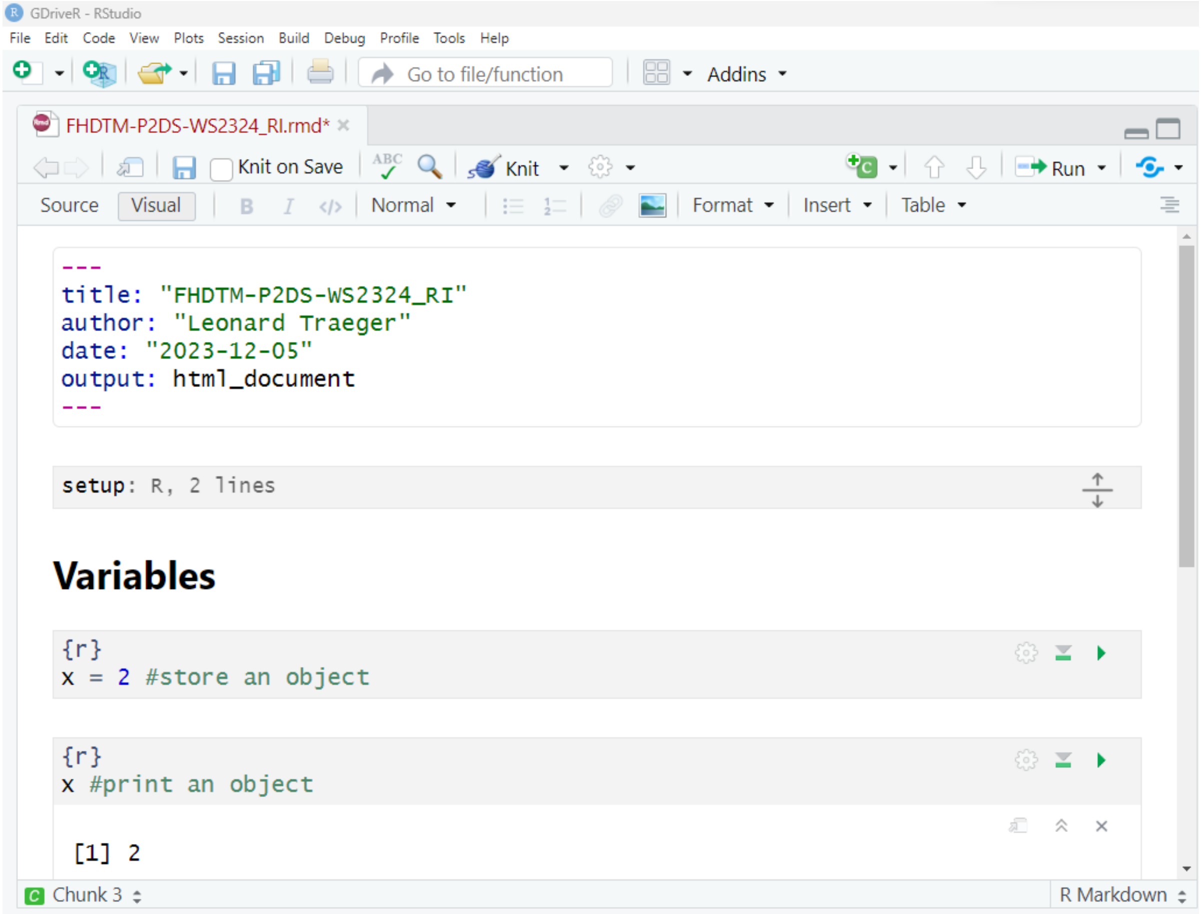
R Markdown

File > New File > R Markdown > …

- Choose HTML type
- Store file as .Rmd

Text annotations

- # headline
- **bold**
- *italics*
- `code`



```
---
title: "FHDTM-P2DS-WS2324_RI"
author: "Leonard Traeger"
date: "2023-12-05"
output: html_document
---
```

```
setup: R, 2 lines
```

Variables

```
{r}
x = 2 #store an object
```

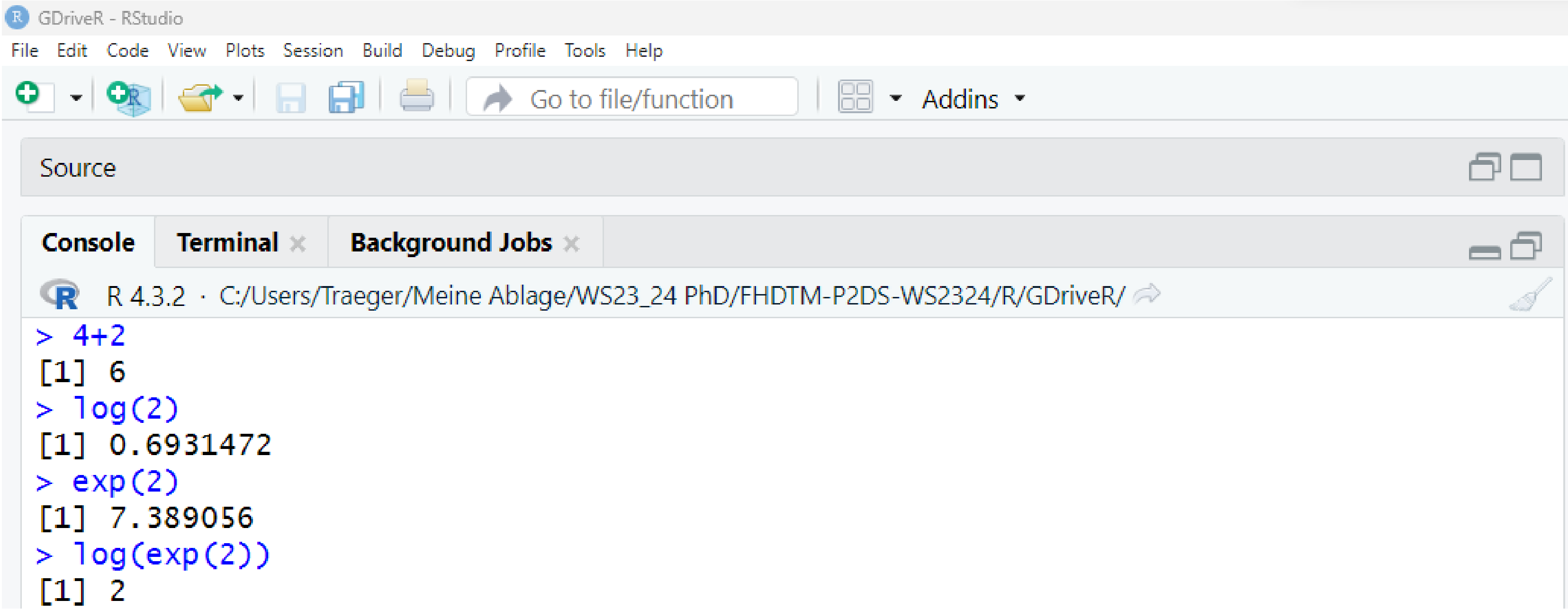
```
{r}
x #print an object
```

```
[1] 2
```

1. R Markdown documents has become established for the creation of report material (similar to our work in Python scripts).

RStudio Console

- The console is the direct connection between R and the computer that performs the calculations.
- You can use R as a simple calculator:

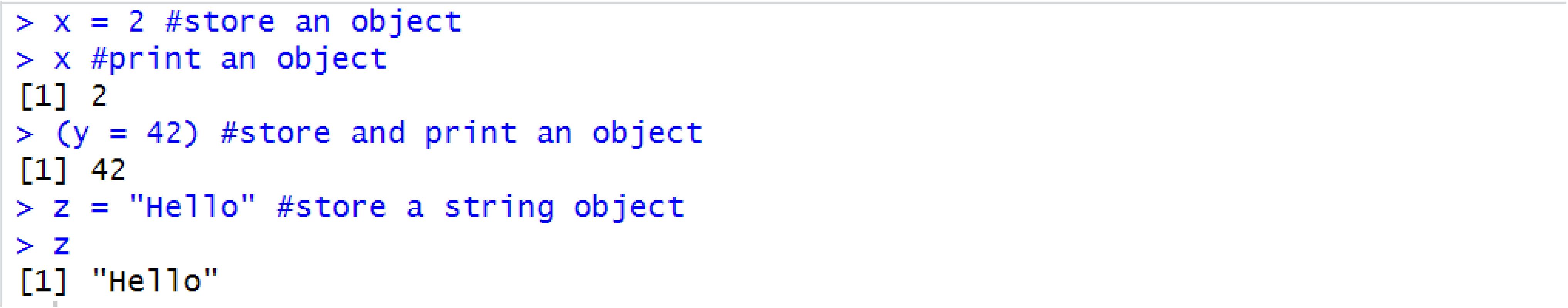


GDriveR - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Source
Console
Terminal
Background Jobs
R 4.3.2 · C:/Users/Traeger/Meine Ablage/WS23_24 PhD/FHDTM-P2DS-WS2324/R/GDriveR/

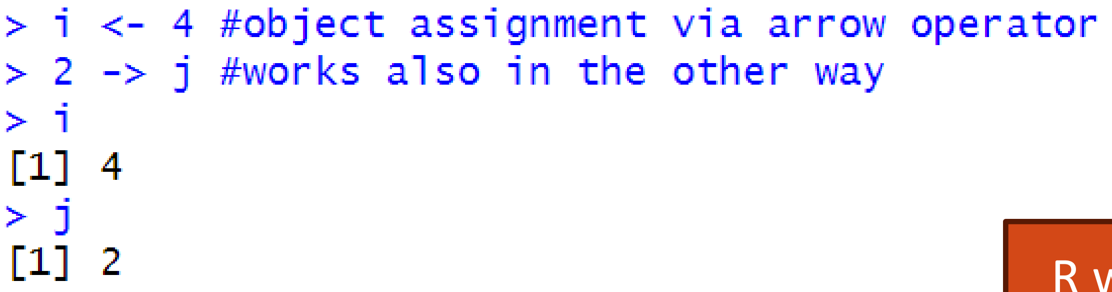
```
> 4+2
[1] 6
> log(2)
[1] 0.6931472
> exp(2)
[1] 7.389056
> log(exp(2))
[1] 2
```

Numeric and string objects

R is a dynamically typed language.



```
> x = 2 #store an object
> x #print an object
[1] 2
> (y = 42) #store and print an object
[1] 42
> z = "Hello" #store a string object
> z
[1] "Hello"
```

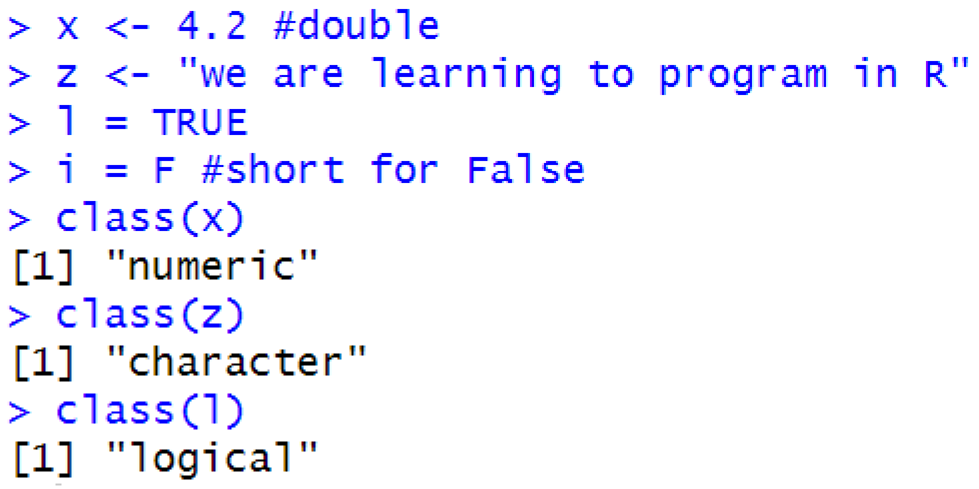


```
> i <- 4 #object assignment via arrow operator
> 2 -> j #works also in the other way
> i
[1] 4
> j
[1] 2
```

R will ignore any text after # for that line.

Objects and Class

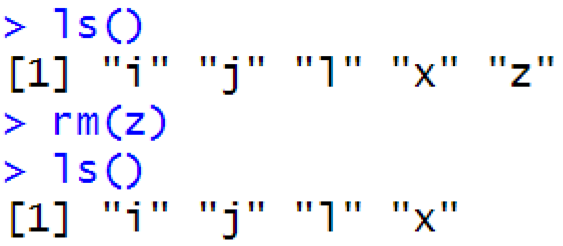
To find out which object class a variable has, use the `class(variable)` function.



```
> x <- 4.2 #double
> z <- "we are learning to program in R"
> l = TRUE
> i = F #short for False
> class(x)
[1] "numeric"
> class(z)
[1] "character"
> class(l)
[1] "logical"
```

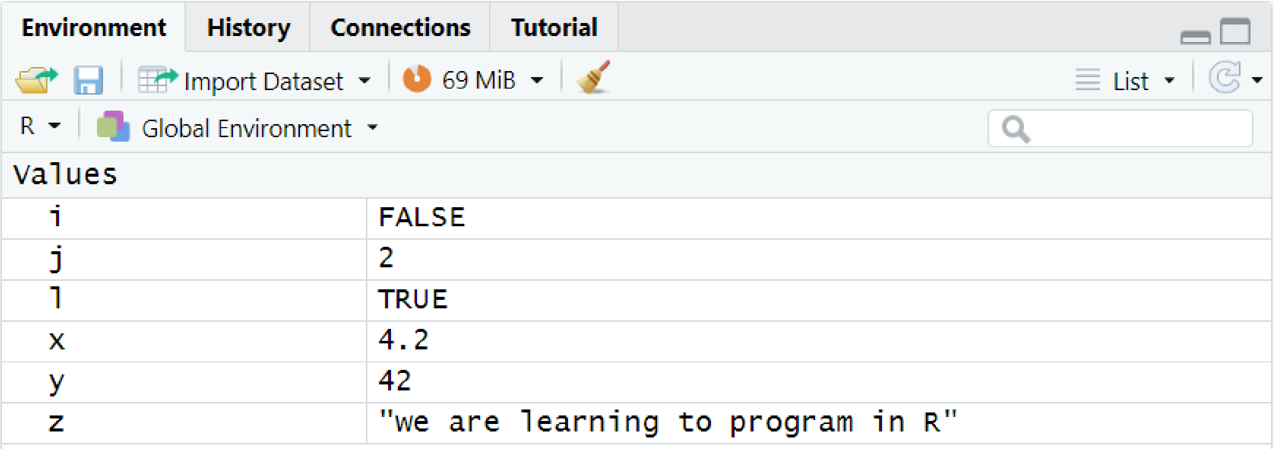
Environment Variables

- List all environment variables with the `ls()` function.
- Remove via `rm(variable)` function.



```
> ls()
[1] "i" "j" "l" "x" "z"
> rm(z)
> ls()
[1] "i" "j" "l" "x"
```

You can also see your environment variables in the top right window.



Values

i	FALSE
j	2
l	TRUE
x	4.2
y	42
z	"we are learning to program in R"

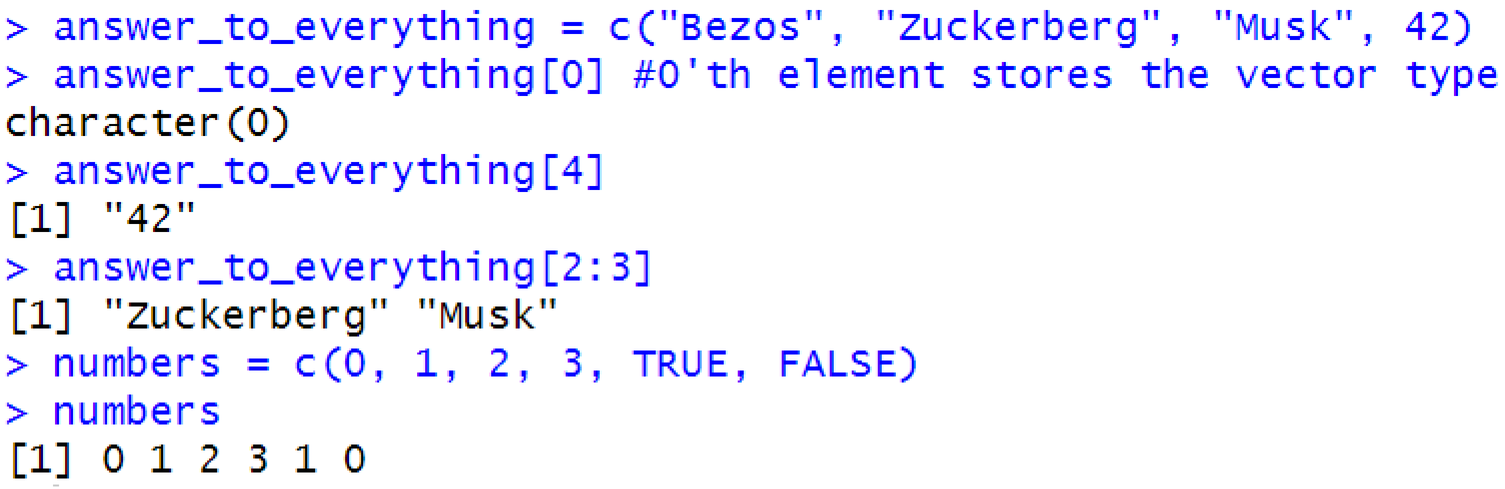
Logical Operators

`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

Vectors

Function `c()` (**concatenate**) creates a vector, a data structure with several elements.

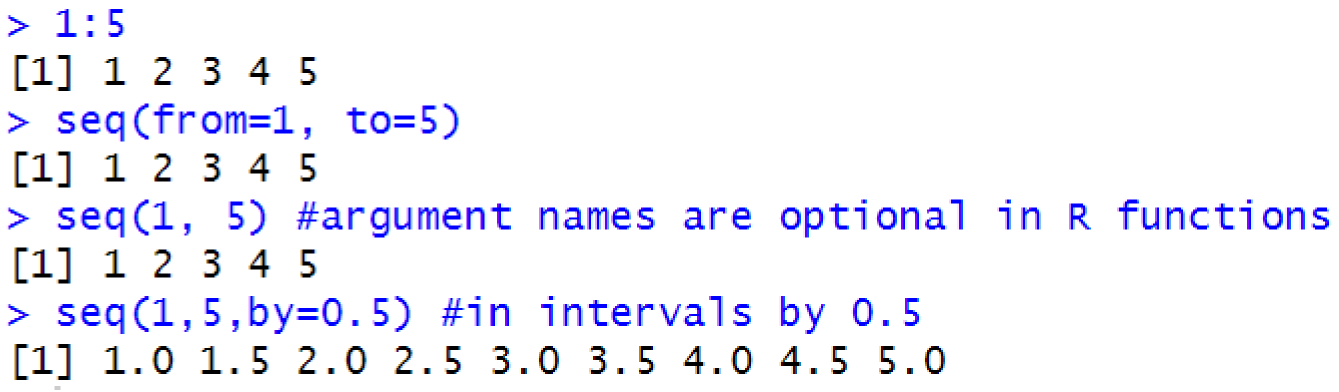
If the elements belong to **different classes** (strings, booleans, numerics), the elements are **converted** to the **same type silently**, i.e. without a warning message.



```
> answer_to_everything = c("Bezos", "Zuckerberg", "Musk", 42)
> answer_to_everything[0] #0'th element stores the vector type
character(0)
> answer_to_everything[4]
[1] "42"
> answer_to_everything[2:3]
[1] "Zuckerberg" "Musk"
> numbers = c(0, 1, 2, 3, TRUE, FALSE)
> numbers
[1] 0 1 2 3 1 0
```

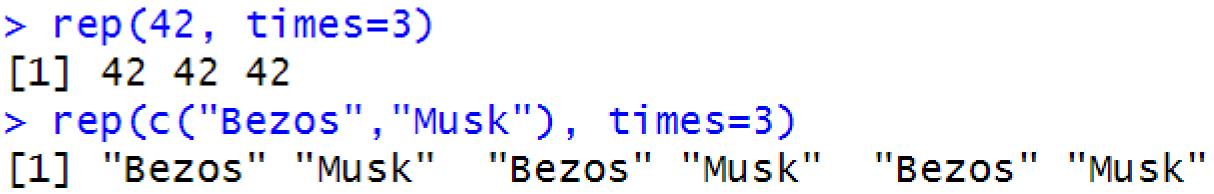
Vectors (cont.)

Use the `seq(from, to, by)` function to create regular sequences of numbers:



```
> 1:5
[1] 1 2 3 4 5
> seq(from=1, to=5)
[1] 1 2 3 4 5
> seq(1, 5) #argument names are optional in R functions
[1] 1 2 3 4 5
> seq(1,5,by=0.5) #in intervals by 0.5
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Use the `rep()` function to create sequences with repeating values:



```
> rep(42, times=3)
[1] 42 42 42
> rep(c("Bezos","Musk"), times=3)
[1] "Bezos" "Musk"  "Bezos" "Musk"  "Bezos" "Musk"
```

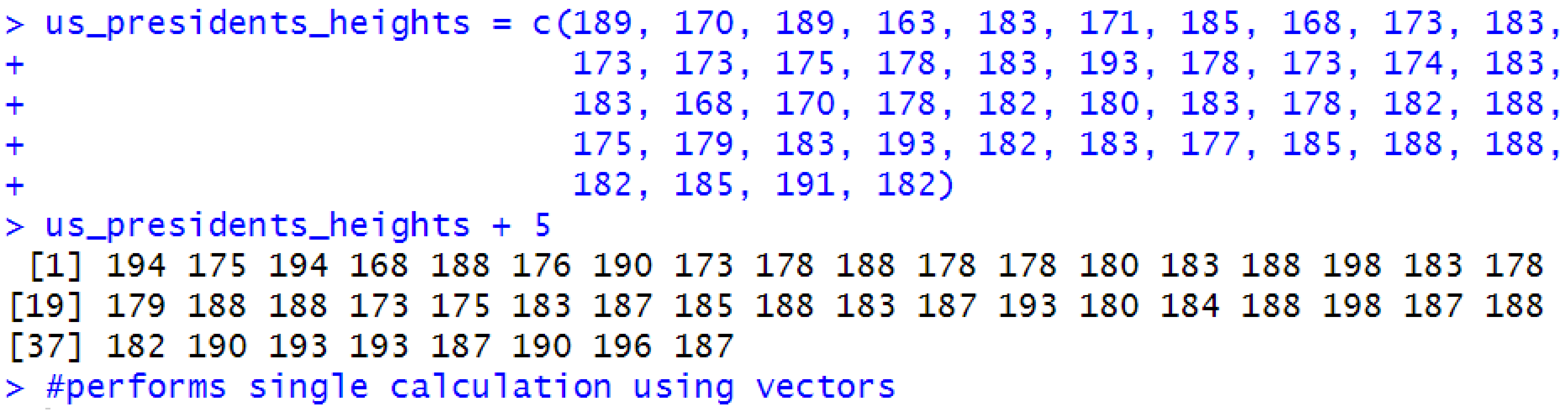
Training #1

Set-up R and Rstudio in your environment and

1. Create a new project in Rstudio for this class.
2. Create two variables x = “Hello“ and y = “World“ and use them to print your first “Hello World“ in R.
   Hint: use the `cat(a, b, ...)` function to print multiple variables on one line.
3. Return a vector with interchangble “R“ and “Python“ elements with the length 100.

Vectors (cont.)

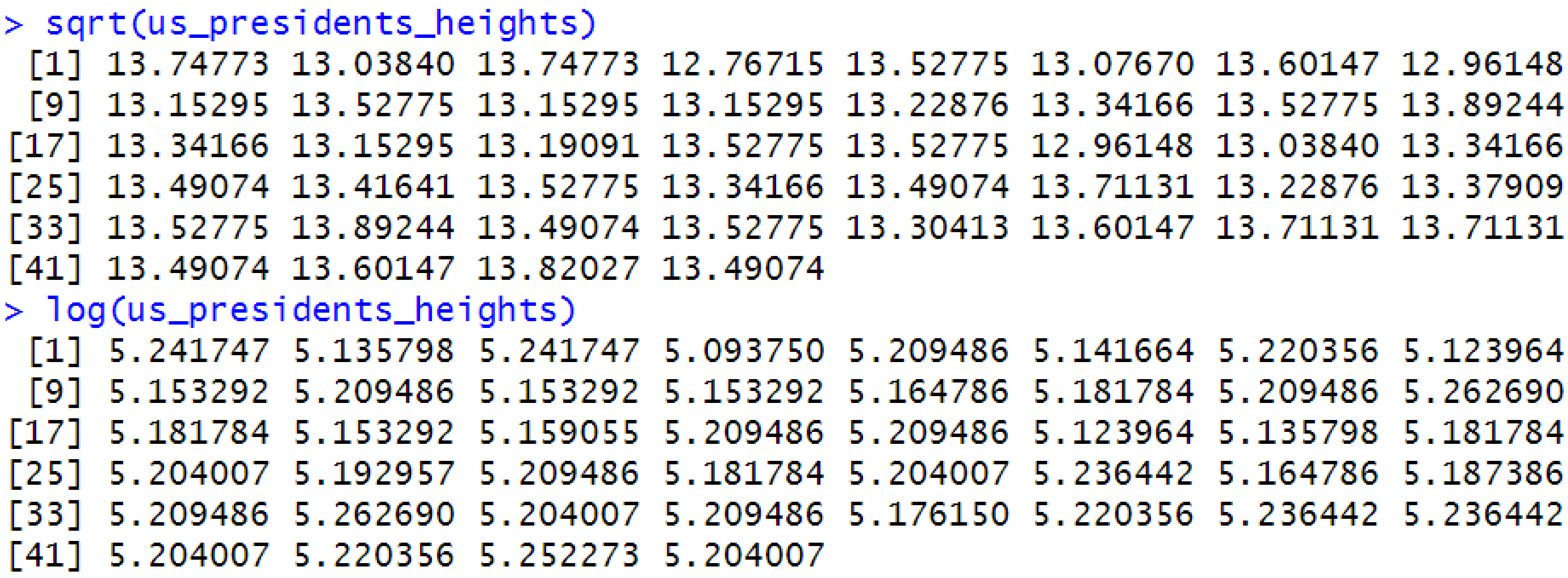
You can apply basic **arithmetic operations** and arithmetic **functions** to numeric vectors:



```
> us_presidents_heights = c(189, 170, 189, 163, 183, 171, 185, 168, 173, 183,
+                           173, 173, 175, 178, 183, 193, 178, 173, 174, 183,
+                           183, 168, 170, 178, 182, 180, 183, 178, 182, 188,
+                           175, 179, 183, 193, 182, 183, 177, 185, 188, 188,
+                           182, 185, 191, 182)
> us_presidents_heights + 5
 [1] 194 175 194 168 188 176 190 173 178 188 178 178 180 183 188 198 183 178
[19] 179 188 188 173 175 183 187 185 188 183 187 193 180 184 188 198 187 188
[37] 182 190 193 193 187 190 196 187
> #performs single calculation using vectors
```

Vectors (cont.)

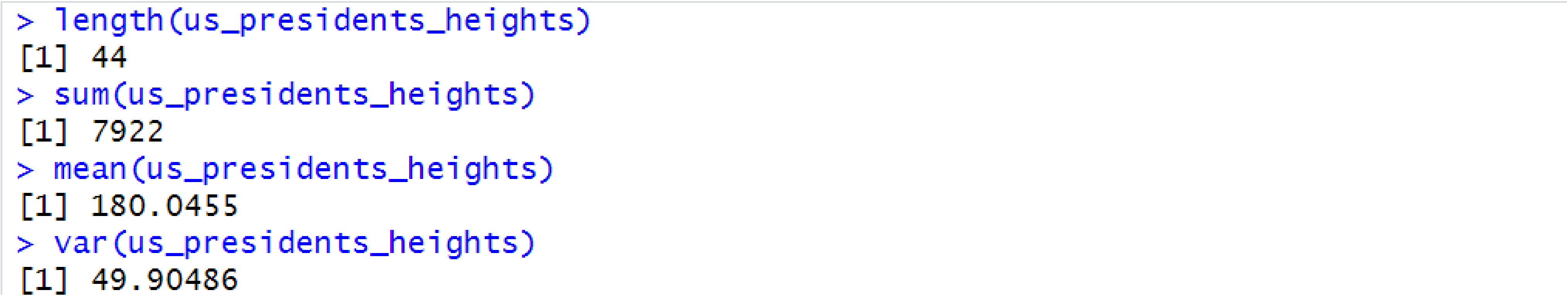
You can apply basic **arithmetic operations** and arithmetic **functions** to numeric vectors:



```
> sqrt(us_presidents_heights)
 [1] 13.74773 13.03840 13.74773 12.76715 13.52775 13.07670 13.60147 12.96148
 [9] 13.15295 13.52775 13.15295 13.15295 13.22876 13.34166 13.52775 13.89244
[17] 13.34166 13.15295 13.19091 13.52775 13.52775 12.96148 13.03840 13.34166
[25] 13.49074 13.41641 13.52775 13.34166 13.49074 13.71131 13.22876 13.37909
[33] 13.52775 13.89244 13.49074 13.52775 13.30413 13.60147 13.71131 13.71131
[41] 13.49074 13.60147 13.82027 13.49074
> log(us_presidents_heights)
 [1] 5.241747 5.135798 5.241747 5.093750 5.209486 5.141664 5.220356 5.123964
 [9] 5.153292 5.209486 5.153292 5.153292 5.164786 5.181784 5.209486 5.262690
[17] 5.181784 5.153292 5.159055 5.209486 5.209486 5.123964 5.135798 5.181784
[25] 5.204007 5.192957 5.209486 5.181784 5.204007 5.236442 5.164786 5.187386
[33] 5.209486 5.262690 5.204007 5.209486 5.176150 5.220356 5.236442 5.236442
[41] 5.204007 5.220356 5.252273 5.204007
```

Vector Aggregation

You can also describe numerical vectors with **aggregate** functions:

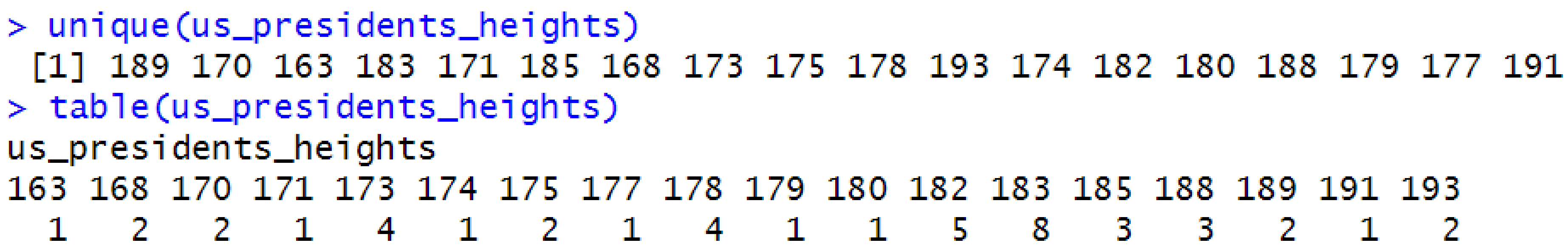


```
> length(us_presidents_heights)
[1] 44
> sum(us_presidents_heights)
[1] 7922
> mean(us_presidents_heights)
[1] 180.0455
> var(us_presidents_heights)
[1] 49.90486
```

Vector Aggregation (cont.)

Use `unique(vector)` to display the unique items of a vector.

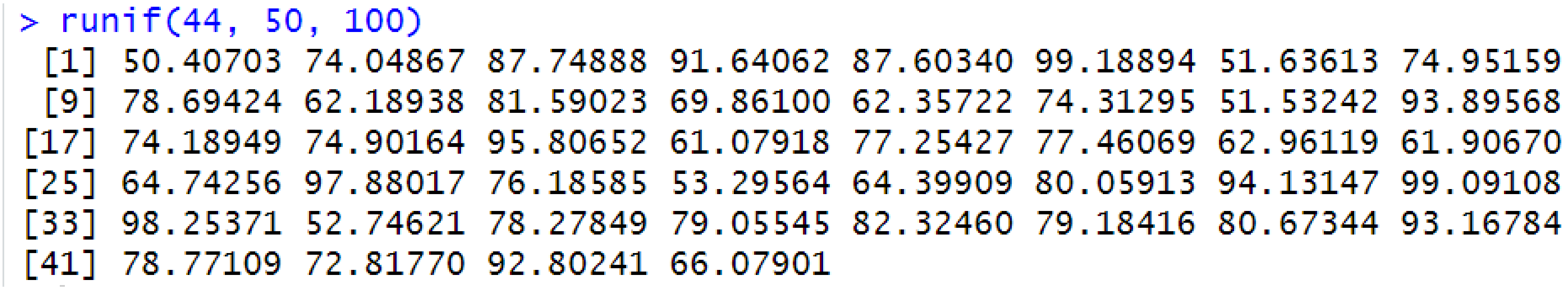
Use `table(vector)` to display a list of unique items and frequency of occurrence.



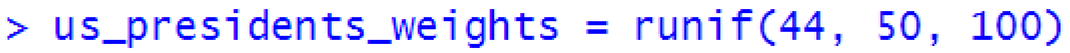
```
> unique(us_presidents_heights)
 [1] 189 170 163 183 171 185 168 173 175 178 193 174 182 180 188 179 177 191
> table(us_presidents_heights)
us_presidents_heights
163 168 170 171 173 174 175 177 178 179 180 182 183 185 188 189 191 193 
  1   2   2   1   4   1   2   1   4   1   1   5   8   3   3   2   1   2 
```

Random Numbers

You can use the `runif(#, min, max)` function to generate random variables:



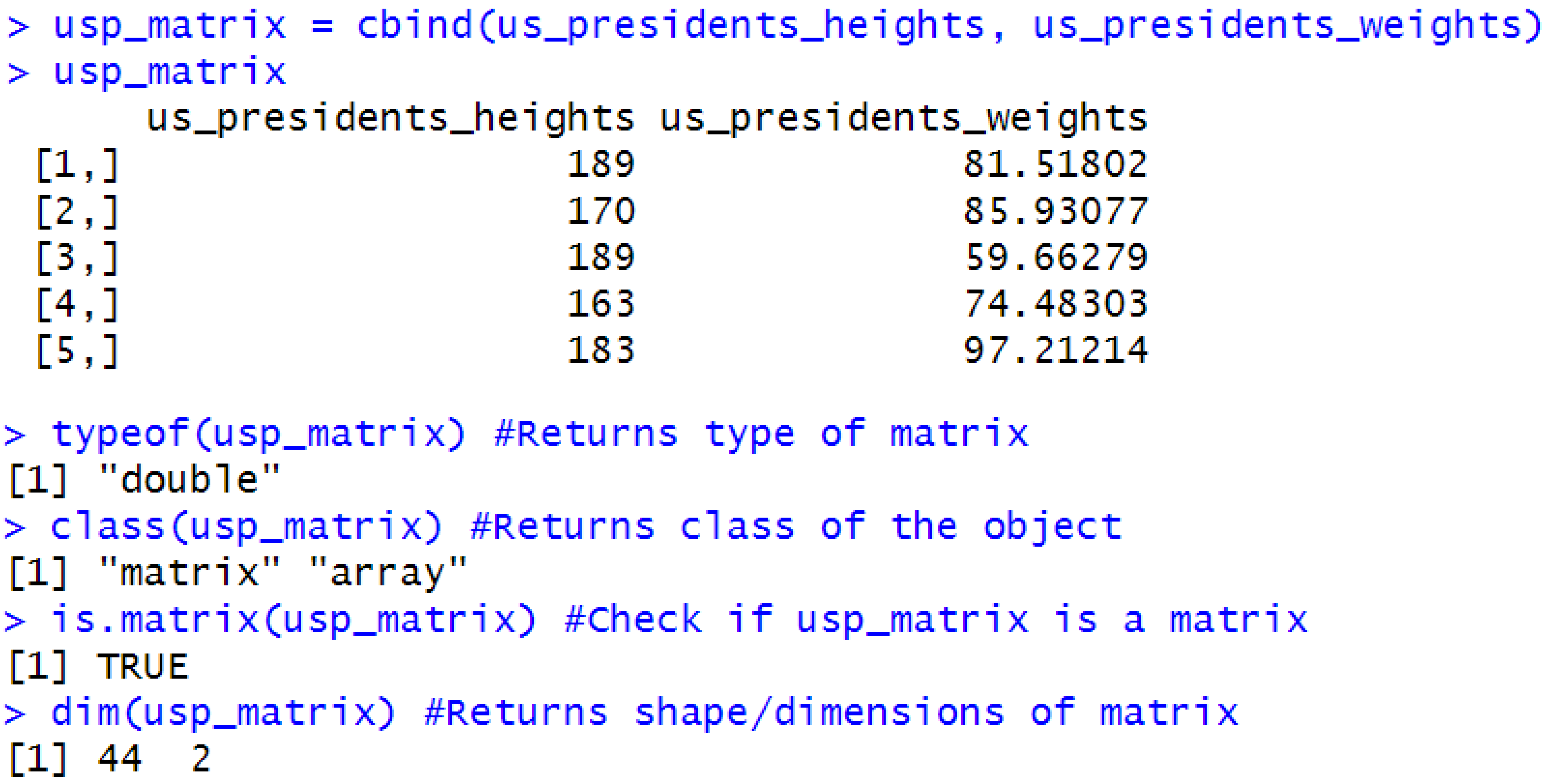
```
> runif(44, 50, 100)
 [1] 50.40703 74.04867 87.74888 91.64062 87.60340 99.18894 51.63613 74.95159
 [9] 78.69424 62.18938 81.59023 69.86100 62.35722 74.31295 51.53242 93.89568
[17] 74.18949 74.90164 95.80652 61.07918 77.25427 77.46069 62.96119 61.90670
[25] 64.74256 97.88017 76.18585 53.29564 64.39909 80.05913 94.13147 99.09108
[33] 98.25371 52.74621 78.27849 79.05545 82.32460 79.18416 80.67344 93.16784
[41] 78.77109 72.81770 92.80241 66.07901
```



```
> us_presidents_weights = runif(44, 50, 100)
```

Matricies

Create matricies via the `cbind(vector**)` function:



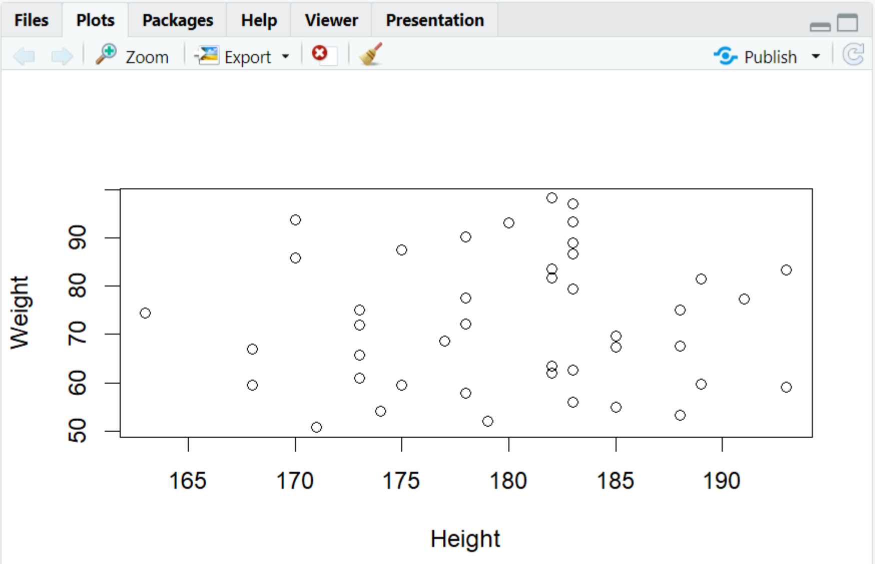
```
> usp_matrix = cbind(us_presidents_heights, us_presidents_weights)
> usp_matrix
     us_presidents_heights us_presidents_weights
[1,]                   189              81.51802
[2,]                   170              85.93077
[3,]                   189              59.66279
[4,]                   163              74.48303
[5,]                   183              97.21214

> typeof(usp_matrix) #Returns type of matrix
[1] "double"
> class(usp_matrix) #Returns class of the object
[1] "matrix" "array"
> is.matrix(usp_matrix) #Check if usp_matrix is a matrix
[1] TRUE
> dim(usp_matrix) #Returns shape/dimensions of matrix
[1] 44  2
```

Simple Plotting

Use the `plot(x,y, ylab, xlab)` function for a Scatter Plot:

```
> plot(us_presidents_heights,us_presidents_weights,ylab="Weight",xlab="Height")
```

The plot will appear in the right bottom grid under the tab „Plots“:



For more advanced and attractive data visualizations, use ggplot.

Training #2

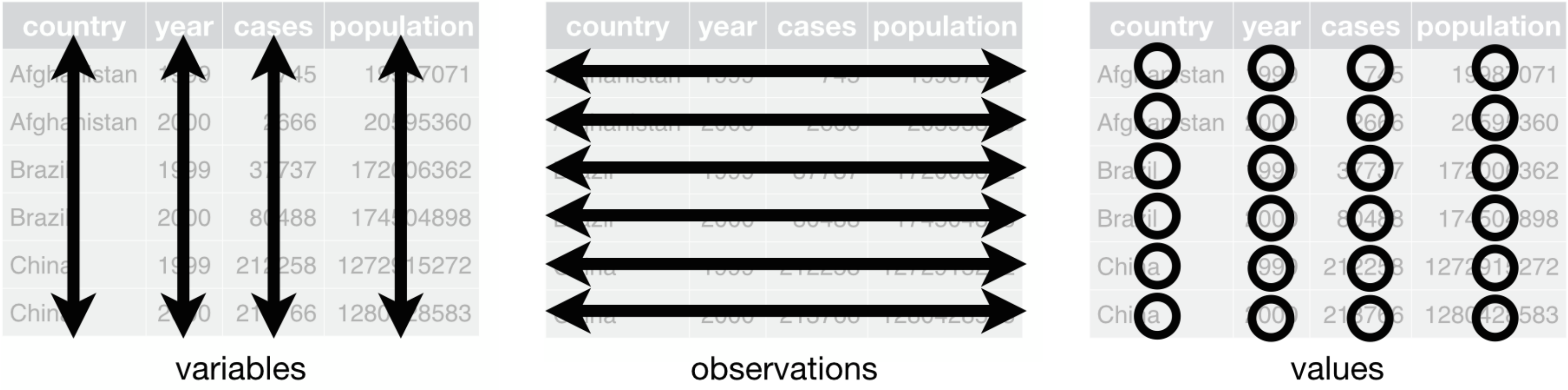
1. Create a vector called **grades** with 100 (random) grades ranging from one to five.
2. Use the `round(vector, digits=0)` method to round the grades to integers.
3. View the frequency of grades using the `table()` function.
4. Compute the mean grade using the `length()` and `sum()` function. Do you receive the same mean grade when using the `mean()` function?

Break

DataFrames

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

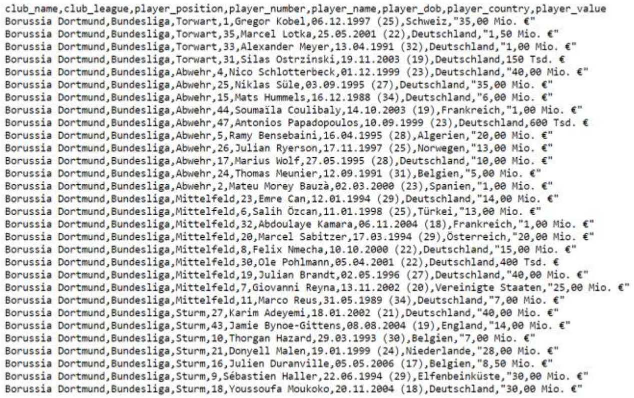
- Rows also called observations.
- Columns also called variables (not to be confused with the variables from before!).



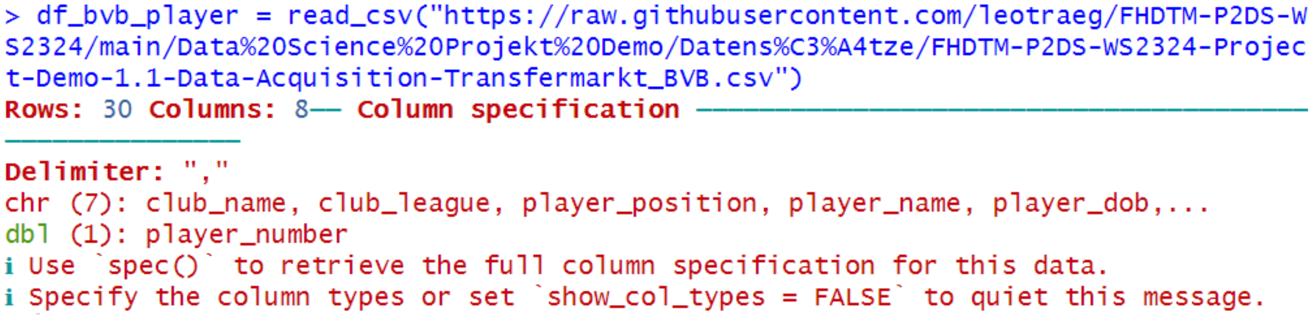
R for Data Science (e2) by Wickham, Çetinkaya-Rundel, and Grolemund `library(readr)`

Reading data from .csv

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.
```

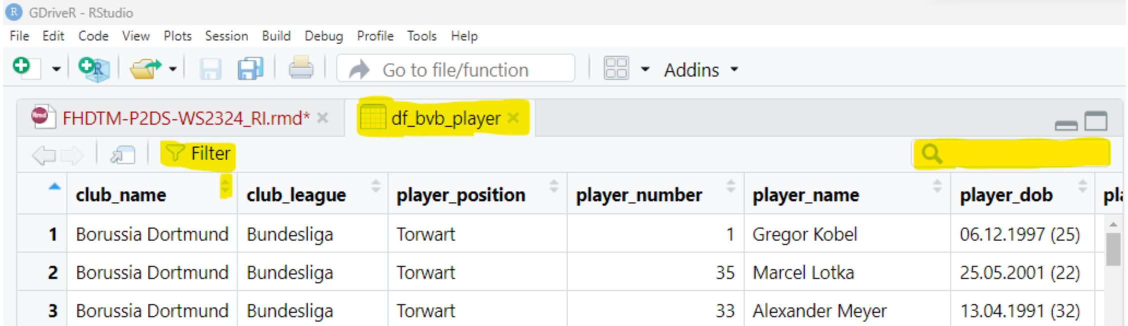
In successful .csv read; a log message tells you the

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

Explore DataFrame

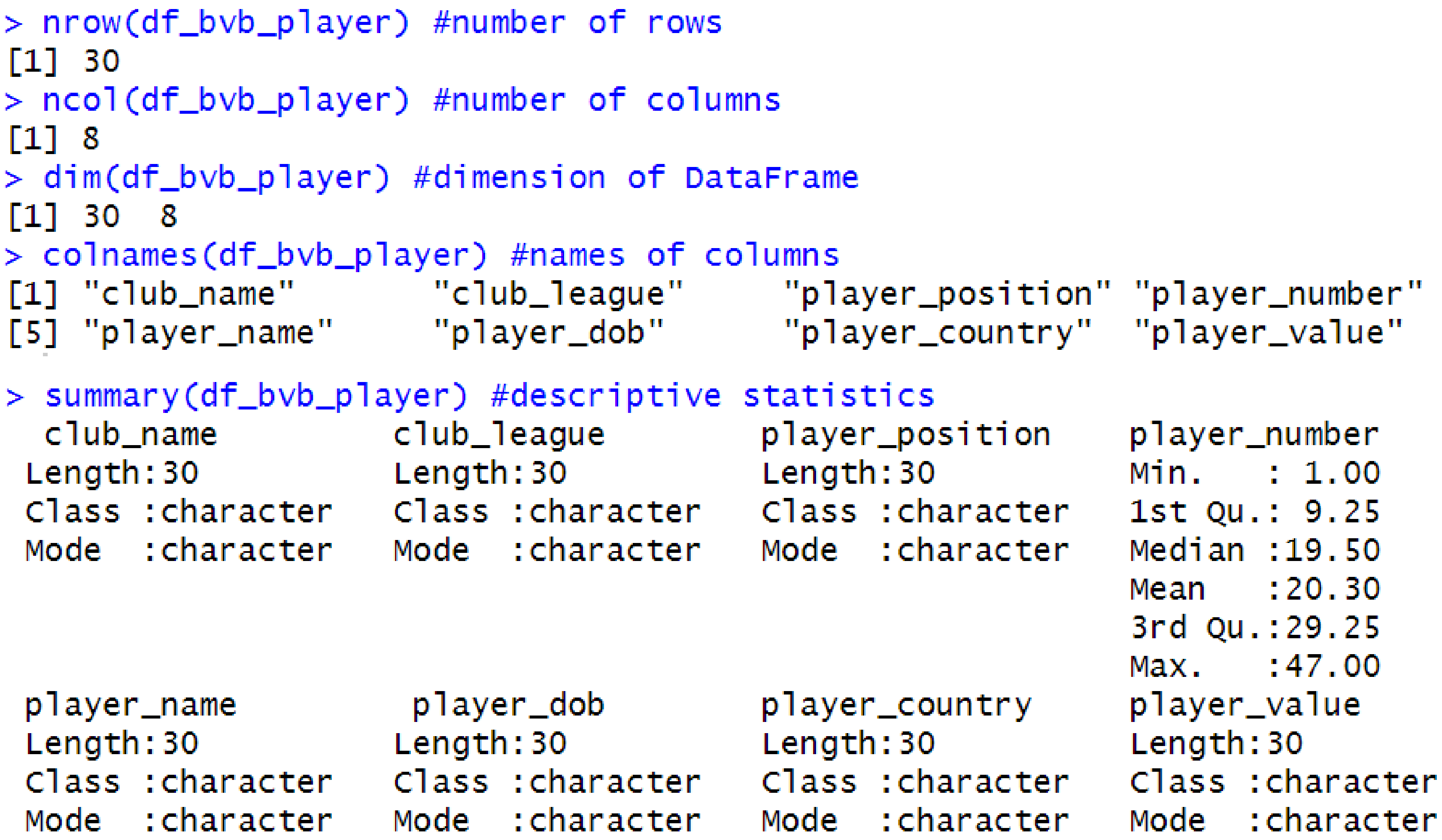
`tail(dataframe)`
returns the last few rows

```
> 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)	

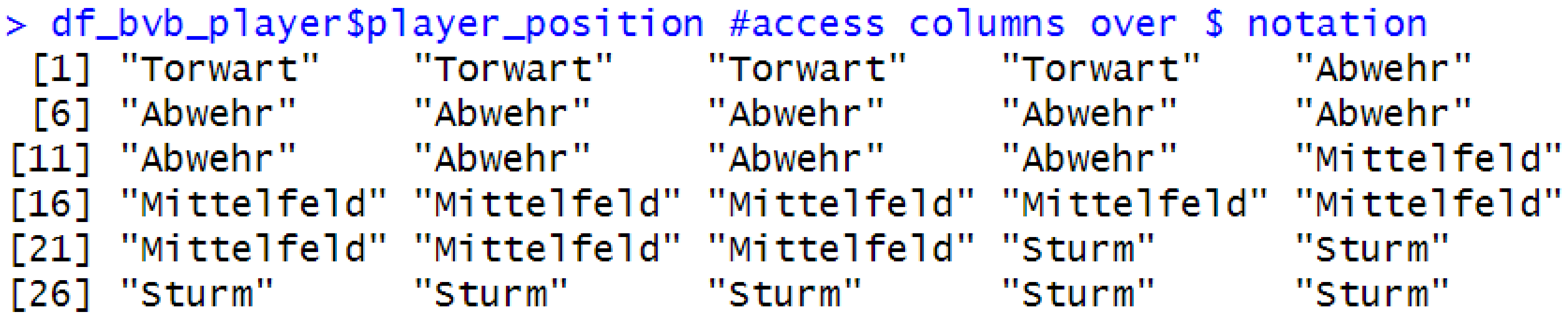
Explore DataFrame (cont.)



```
> 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) #descriptive statistics
  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
```

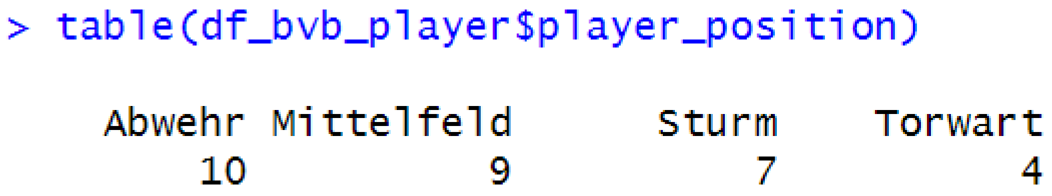
Explore DataFrame (cont.)

You can access columns in a DataFrame via $ notation:



```
> df_bvb_player$player_position #access columns over $ notation
 [1] "Torwart"    "Torwart"    "Torwart"    "Torwart"    "Abwehr"    
 [6] "Abwehr"     "Abwehr"     "Abwehr"     "Abwehr"     "Abwehr"    
[11] "Abwehr"     "Abwehr"     "Abwehr"     "Abwehr"     "Mittelfeld"
[16] "Mittelfeld" "Mittelfeld" "Mittelfeld" "Mittelfeld" "Mittelfeld"
[21] "Mittelfeld" "Mittelfeld" "Mittelfeld" "Sturm"      "Sturm"     
[26] "Sturm"      "Sturm"      "Sturm"      "Sturm"      "Sturm"     
```

In the end, columns of DataFrames or nothing else than a vector (c) – and you can apply vector functions on them such as `table()`:



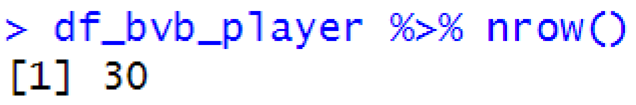
```
> table(df_bvb_player$player_position)

    Abwehr Mittelfeld      Sturm    Torwart 
        10          9          7          4 
```

Pipe %>%

Important tidyverse **syntax** important for Data Wrangling.

From now on, we start our coding with the DataFrame, the pipe %>%, and the function.

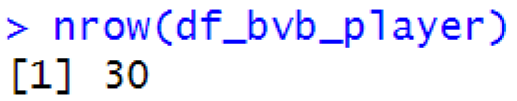
The pipe always takes what is to the left and passes it on to the function to the right:



```
> df_bvb_player %>% nrow()
[1] 30
```

In the code above, the `nrow()` function is applied to the `df_bvb_player`.

This is the same as the following line of code:



```
> nrow(df_bvb_player)
[1] 30
```

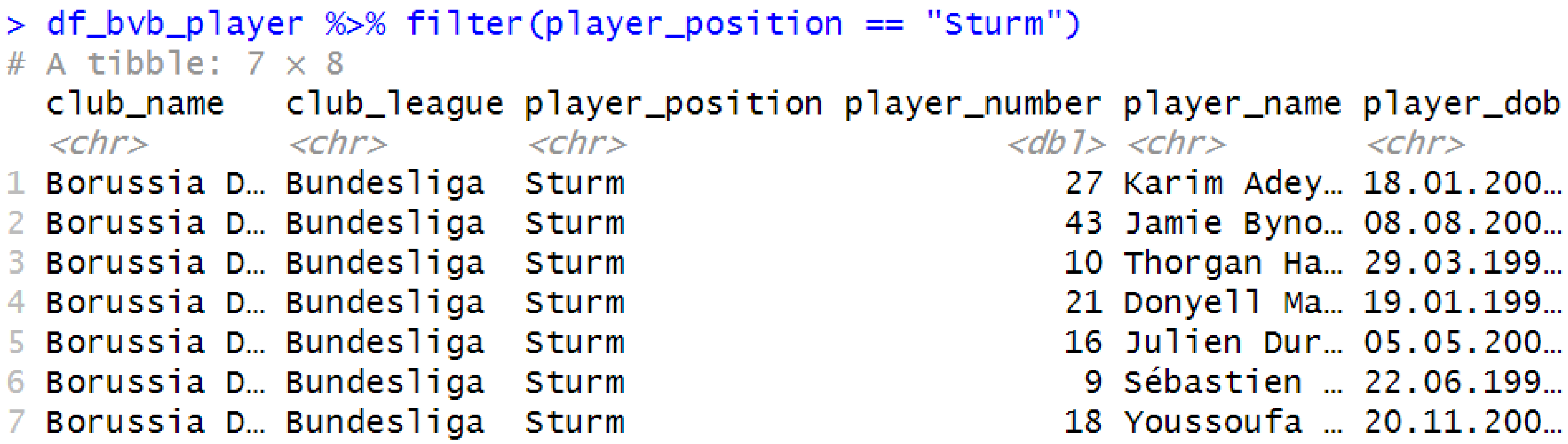
The **advantage** is that you can **connect as many functions** to the pipe **as you want**.

library(dplyr)

DataFrame Filtering

Rows are selected using the `filter(bool_statement)` function.

The function receives one or more logical expressions as argument(s).

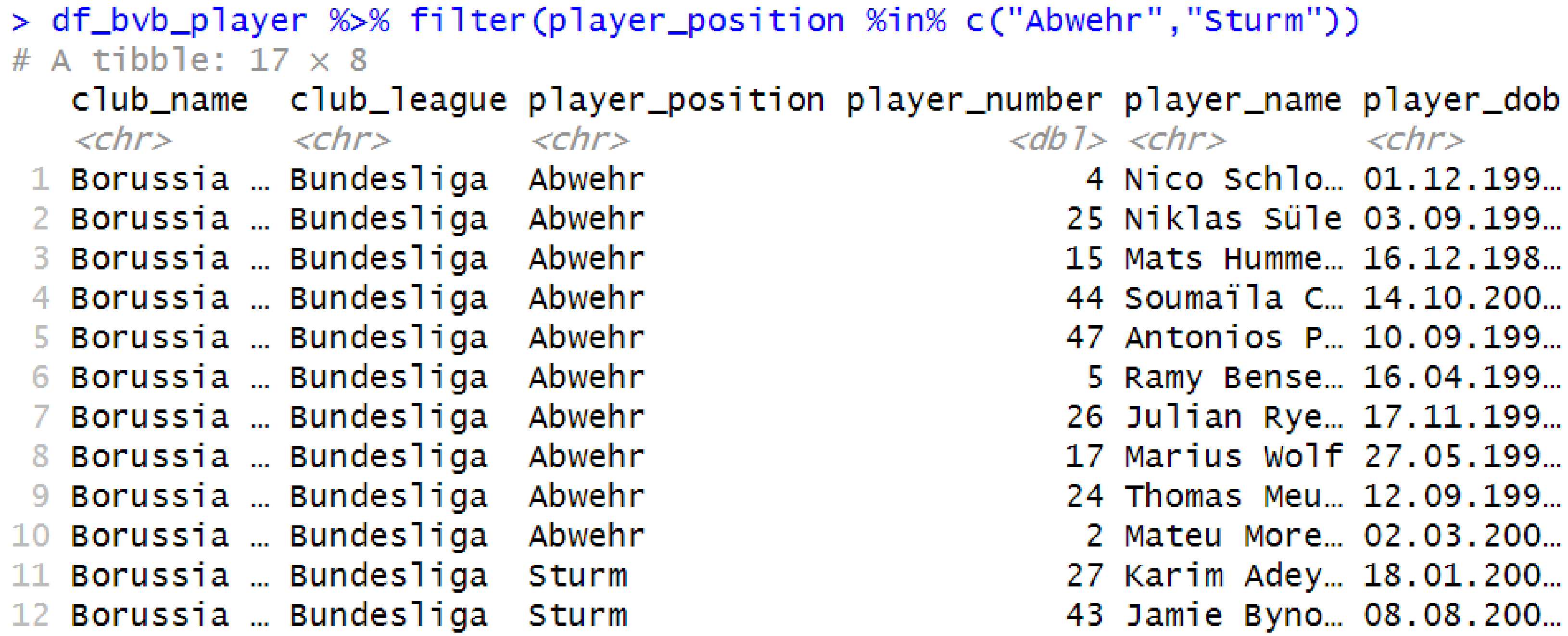


```
> df_bvb_player %>% filter(player_position == "Sturm")
# A tibble: 7 × 8
  club_name   club_league player_position player_number player_name player_dob
  <chr>       <chr>       <chr>                   <dbl> <chr>       <chr>     
1 Borussia D… Bundesliga  Sturm                      27 Karim Adey… 18.01.200…
2 Borussia D… Bundesliga  Sturm                      43 Jamie Byno… 08.08.200…
3 Borussia D… Bundesliga  Sturm                      10 Thorgan Ha… 29.03.199…
4 Borussia D… Bundesliga  Sturm                      21 Donyell Ma… 19.01.199…
5 Borussia D… Bundesliga  Sturm                      16 Julien Dur… 05.05.200…
6 Borussia D… Bundesliga  Sturm                       9 Sébastien … 22.06.199…
7 Borussia D… Bundesliga  Sturm                      18 Youssoufa … 20.11.200…
```

DataFrame Filtering (cont.)

Rows are selected using the `filter(bool_statement)` function.

The function receives one or more logical expressions as argument(s).

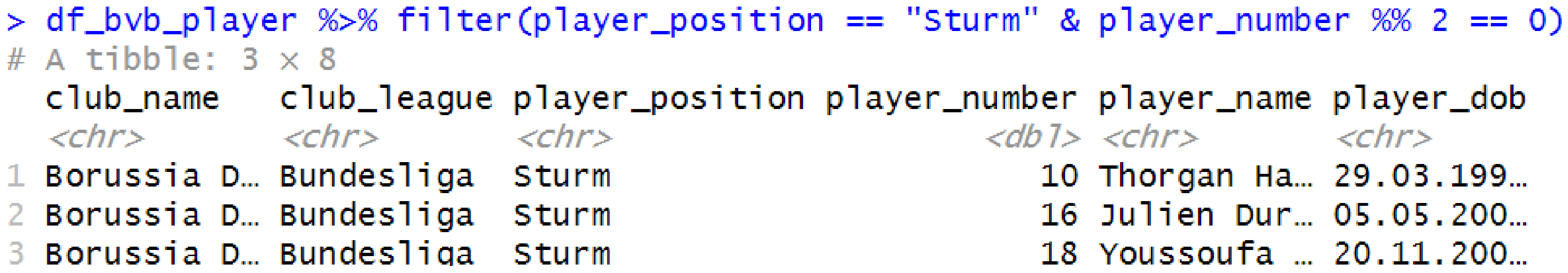


```
> df_bvb_player %>% filter(player_position %in% c("Abwehr","Sturm"))
# A tibble: 17 × 8
   club_name   club_league player_position player_number player_name player_dob
   <chr>       <chr>       <chr>                   <dbl> <chr>       <chr>
 1 Borussia …  Bundesliga  Abwehr                      4 Nico Schlo… 01.12.199…
 2 Borussia …  Bundesliga  Abwehr                     25 Niklas Süle 03.09.199…
 3 Borussia …  Bundesliga  Abwehr                     15 Mats Humme… 16.12.198…
 4 Borussia …  Bundesliga  Abwehr                     44 Soumaïla C… 14.10.200…
 5 Borussia …  Bundesliga  Abwehr                     47 Antonios P… 10.09.199…
 6 Borussia …  Bundesliga  Abwehr                      5 Ramy Bense… 16.04.199…
 7 Borussia …  Bundesliga  Abwehr                     26 Julian Rye… 17.11.199…
 8 Borussia …  Bundesliga  Abwehr                     17 Marius Wolf 27.05.199…
 9 Borussia …  Bundesliga  Abwehr                     24 Thomas Meu… 12.09.199…
10 Borussia …  Bundesliga  Abwehr                      2 Mateu More… 02.03.200…
11 Borussia …  Bundesliga  Sturm                      27 Karim Adey… 18.01.200…
12 Borussia …  Bundesliga  Sturm                      43 Jamie Byno… 08.08.200…
```

DataFrame Filtering (cont.)

Rows are selected using the `filter(bool_statement)` function.

The function receives one or more logical expressions as argument(s).

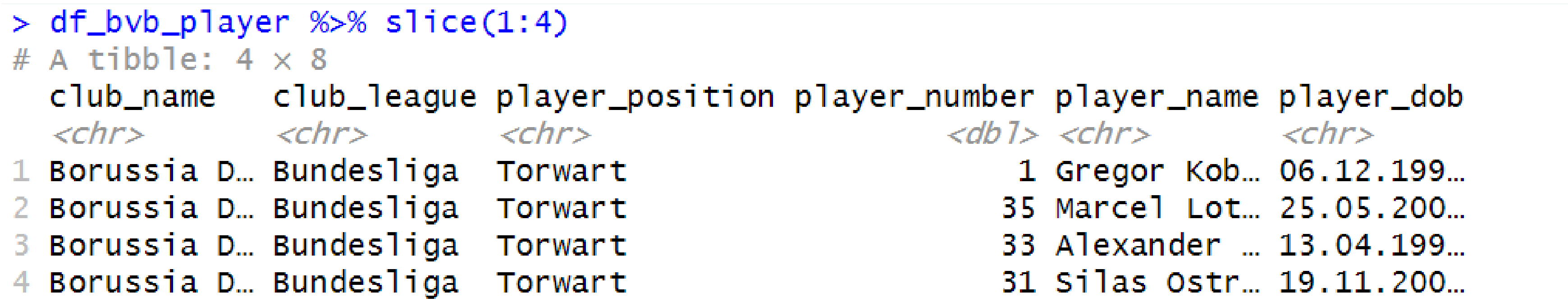


```
> df_bvb_player %>% filter(player_position == "Sturm" & player_number %% 2 == 0)
# A tibble: 3 x 8
  club_name    club_league player_position player_number player_name player_dob
  <chr>        <chr>       <chr>                   <dbl> <chr>       <chr>
1 Borussia D… Bundesliga  Sturm                      10 Thorgan Ha… 29.03.199…
2 Borussia D… Bundesliga  Sturm                      16 Julien Dur… 05.05.200…
3 Borussia D… Bundesliga  Sturm                      18 Youssoufa … 20.11.200…
```

DataFrame Slicing

The rows in a data frame are numbered consecutively, i.e., the rows have an **index**.

Use `slice(index or sequence)` to select rows with the internal index.

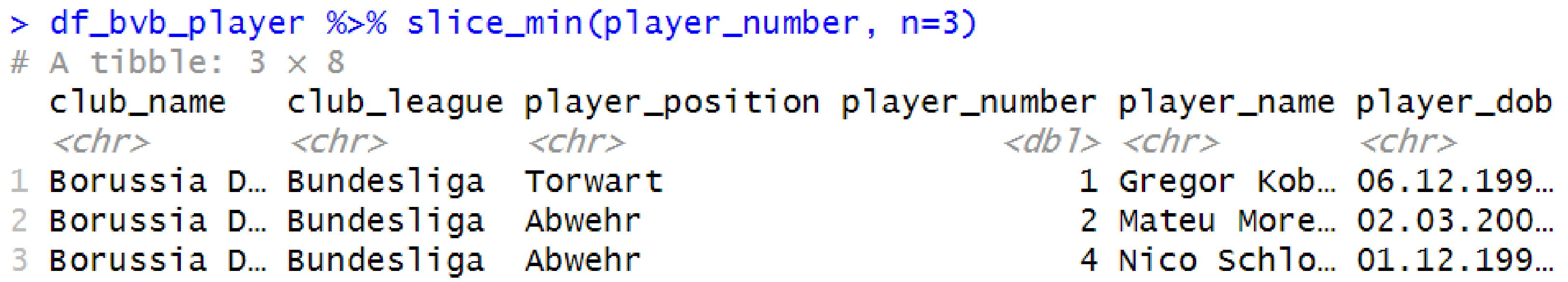


```
> df_bvb_player %>% slice(1:4)
# A tibble: 4 × 8
  club_name   club_league player_position player_number player_name player_dob
  <chr>       <chr>       <chr>                   <dbl> <chr>       <chr>
1 Borussia D… Bundesliga  Torwart                     1 Gregor Kob… 06.12.199…
2 Borussia D… Bundesliga  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…
```

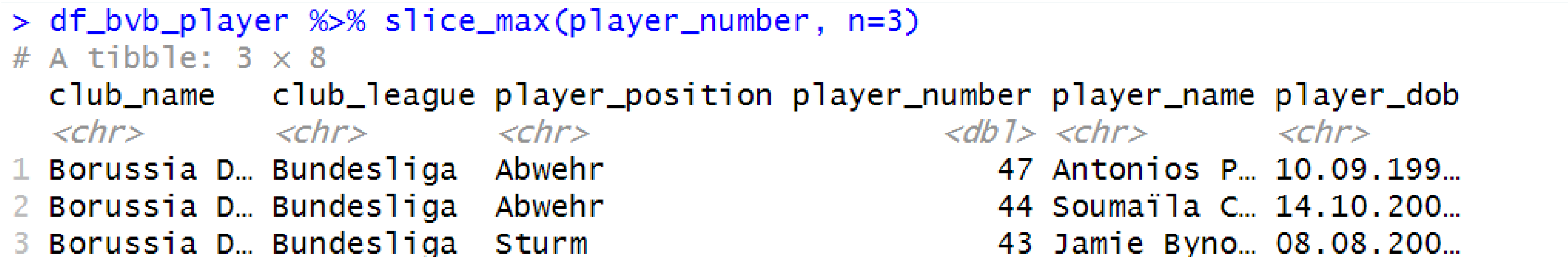
DataFrame Slicing (cont.)

Also works on categorical attributes with alphabetical order

The functions `slice_min(column , n=1)` and `slice_max(column, n=1)` return the n rows that have the **lowest** or **highest** values in a column.



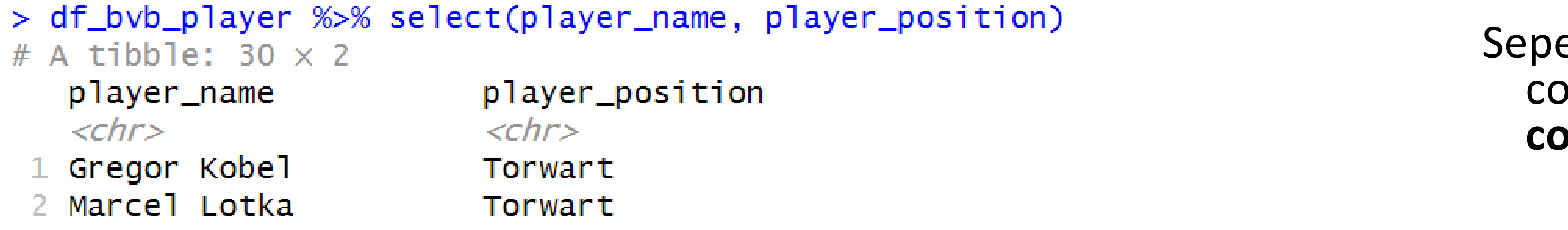
```
> df_bvb_player %>% slice_min(player_number, n=3)
# A tibble: 3 × 8
  club_name    club_league player_position player_number player_name player_dob
  <chr>        <chr>       <chr>                   <dbl> <chr>       <chr>
1 Borussia D… Bundesliga  Torwart                     1 Gregor Kob… 06.12.199…
2 Borussia D… Bundesliga  Abwehr                      2 Mateu More… 02.03.200…
3 Borussia D… Bundesliga  Abwehr                      4 Nico Schlo… 01.12.199…
```



```
> df_bvb_player %>% slice_max(player_number, n=3)
# A tibble: 3 × 8
  club_name    club_league player_position player_number player_name player_dob
  <chr>        <chr>       <chr>                   <dbl> <chr>       <chr>
1 Borussia D… Bundesliga  Abwehr                     47 Antonios P… 10.09.199…
2 Borussia D… Bundesliga  Abwehr                     44 Soumaïla C… 14.10.200…
3 Borussia D… Bundesliga  Sturm                      43 Jamie Byno… 08.08.200…
```

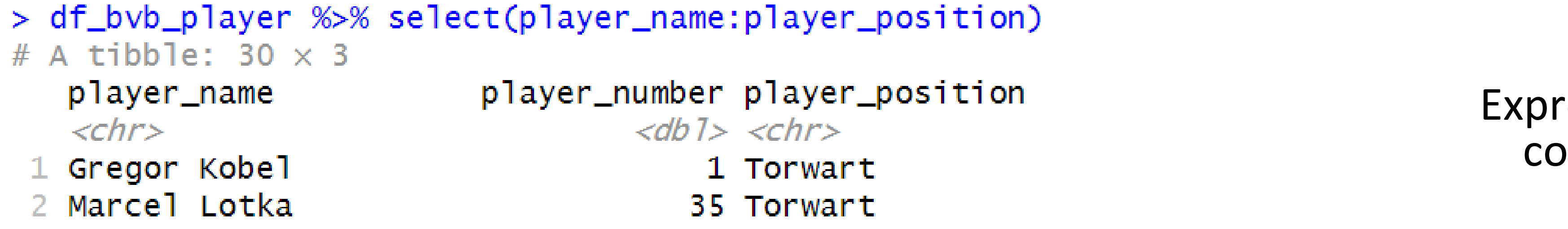
DataFrame Selecting

To select attributes/columns/variables, you can use the function `select()`.



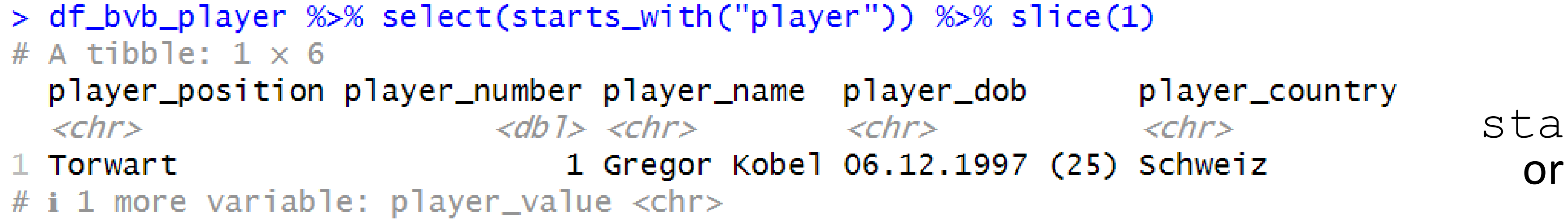
```
> df_bvb_player %>% select(player_name, player_position)
# A tibble: 30 × 2
   player_name           player_position
   <chr>                 <chr>
 1 Gregor Kobel          Torwart
 2 Marcel Lotka          Torwart
```

Seperate multiple columns with a **comma**



```
> df_bvb_player %>% select(player_name:player_position)
# A tibble: 30 × 3
   player_name           player_number player_position
   <chr>                         <dbl> <chr>
 1 Gregor Kobel                      1 Torwart
 2 Marcel Lotka                     35 Torwart
```

Express range of columns with **colon**



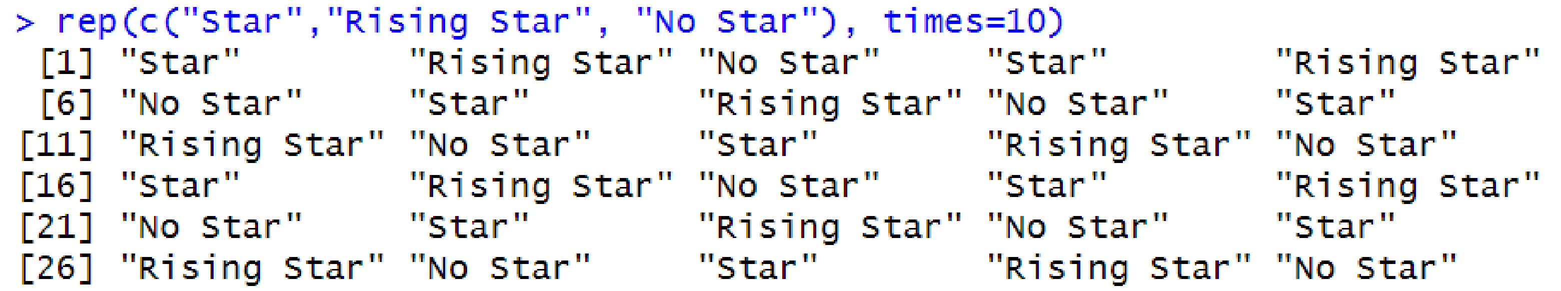
```
> df_bvb_player %>% select(starts_with("player")) %>% slice(1)
# A tibble: 1 × 6
  player_position player_number player_name  player_dob      player_country
  <chr>                   <dbl> <chr>        <chr>           <chr>
1 Torwart                     1 Gregor Kobel 06.12.1997 (25) Schweiz
# ℹ 1 more variable: player_value <chr>
```

`starts_with()` or `end_with()`

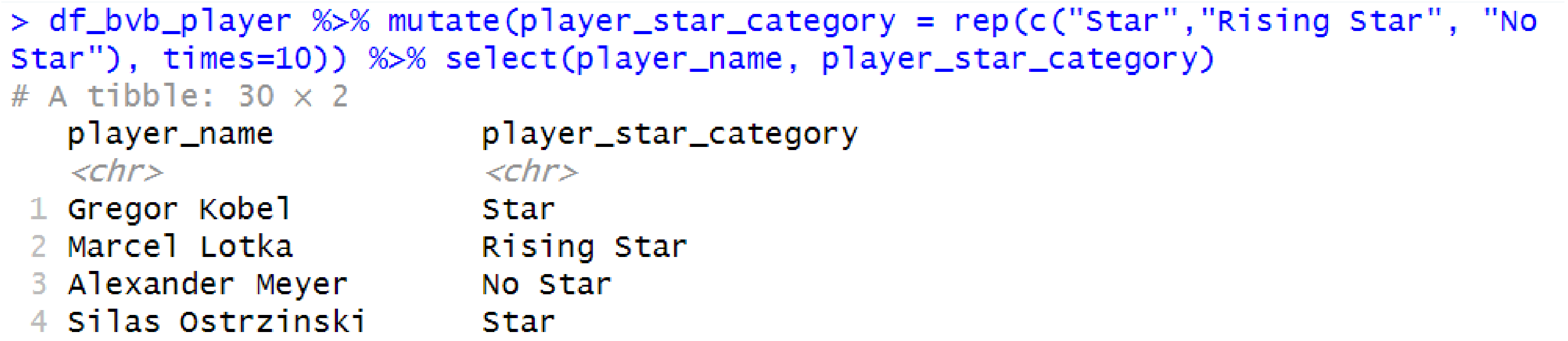
DataFrame Mutating

We can append or change columns to data frames with `mutate()`.

It receives as arguments a new column name with the values as a vector.



```
> rep(c("Star","Rising Star", "No Star"), times=10)
 [1] "Star"        "Rising Star" "No Star"     "Star"        "Rising Star"
 [6] "No Star"     "Star"        "Rising Star" "No Star"     "Star"
[11] "Rising Star" "No Star"     "Star"        "Rising Star" "No Star"
[16] "Star"        "Rising Star" "No Star"     "Star"        "Rising Star"
[21] "No Star"     "Star"        "Rising Star" "No Star"     "Star"
[26] "Rising Star" "No Star"     "Star"        "Rising Star" "No Star"
```

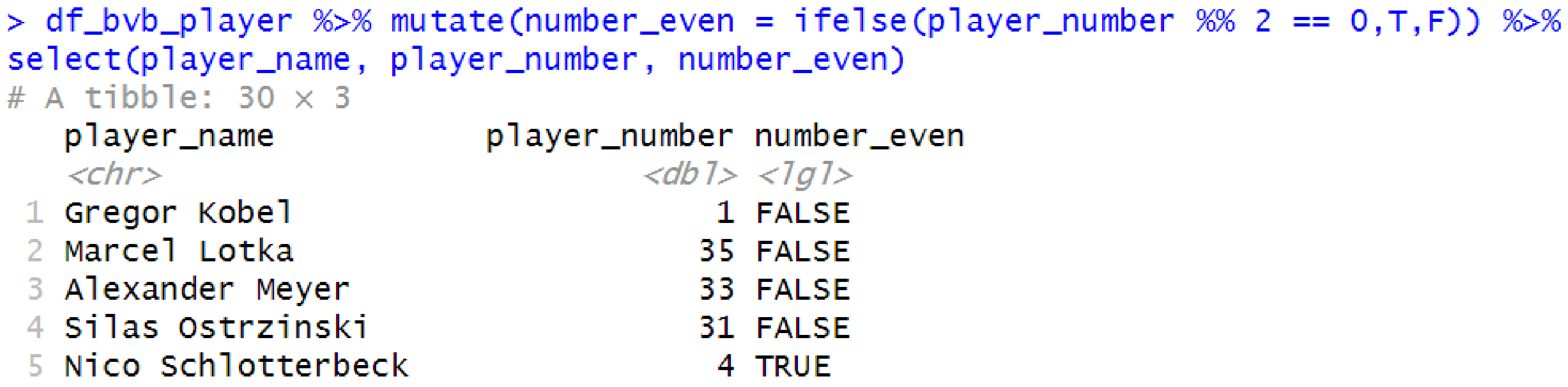


```
> df_bvb_player %>% mutate(player_star_category = rep(c("Star","Rising Star", "No
Star"), times=10)) %>% select(player_name, player_star_category)
# A tibble: 30 × 2
   player_name            player_star_category
   <chr>                  <chr>
 1 Gregor Kobel           Star
 2 Marcel Lotka           Rising Star
 3 Alexander Meyer        No Star
 4 Silas Ostrzinski       Star
```

DataFrame Mutating (cont.)

We can append or change columns to data frames with `mutate()`.

It receives as arguments a new column name with the values as a vector.

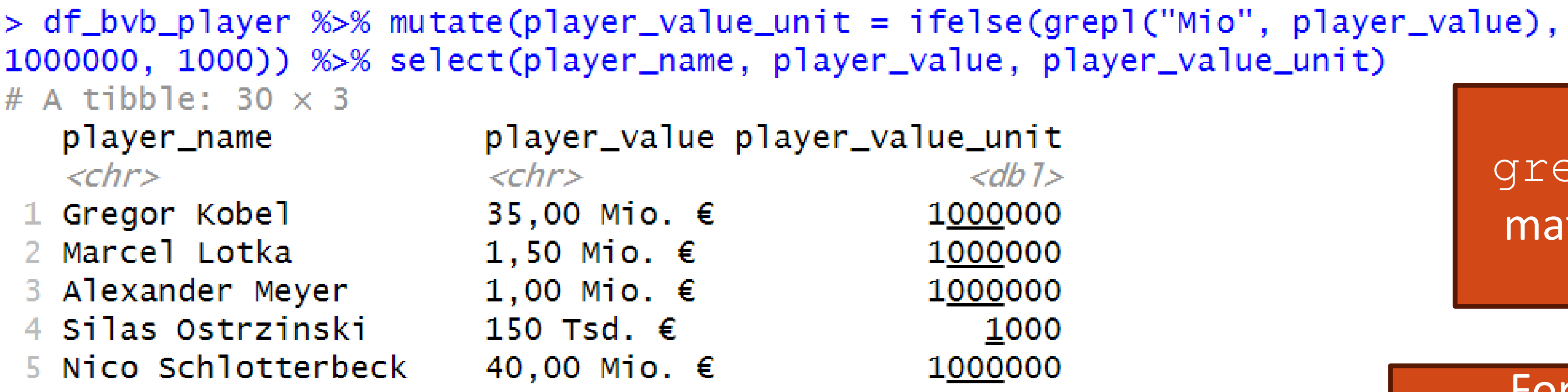


```
> df_bvb_player %>% mutate(number_even = ifelse(player_number %% 2 == 0,T,F)) %>%
select(player_name, player_number, number_even)
# A tibble: 30 × 3
   player_name           player_number number_even
   <chr>                         <dbl> <lgl>
 1 Gregor Kobel                      1 FALSE
 2 Marcel Lotka                     35 FALSE
 3 Alexander Meyer                  33 FALSE
 4 Silas Ostrzinski                 31 FALSE
 5 Nico Schlotterbeck                4 TRUE
```

DataFrame Mutating (cont.)

We can append or change columns to data frames with `mutate()`.

It receives as arguments a new column name with the values as a vector.



```
> df_bvb_player %>% mutate(player_value_unit = ifelse(grepl("Mio", player_value),
1000000, 1000)) %>% select(player_name, player_value, player_value_unit)
# A tibble: 30 × 3
   player_name          player_value player_value_unit
   <chr>                <chr>                    <dbl>
 1 Gregor Kobel         35,00 Mio. €           1000000
 2 Marcel Lotka         1,50 Mio. €            1000000
 3 Alexander Meyer      1,00 Mio. €            1000000
 4 Silas Ostrzinski     150 Tsd. €                1000
 5 Nico Schlotterbeck   40,00 Mio. €           1000000
```

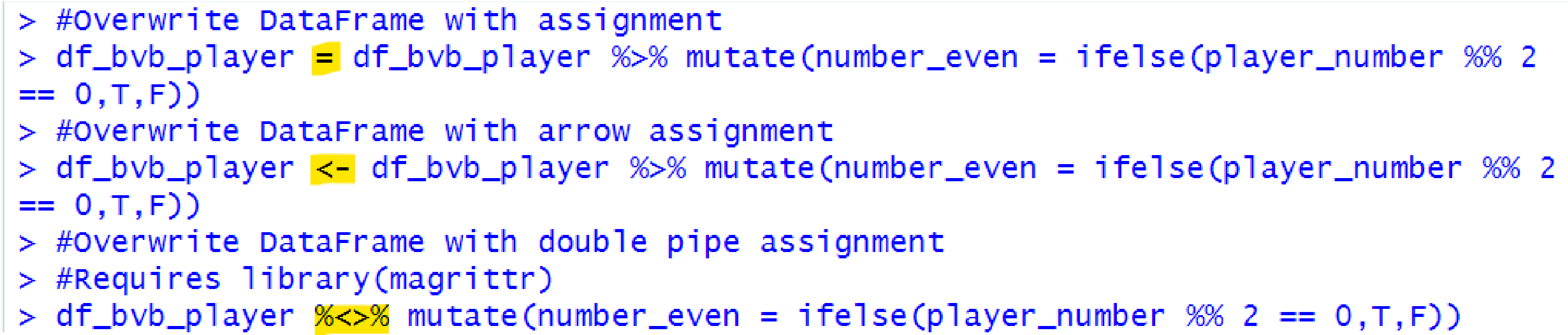
`grepl()` searches for matches in characters

For non-binary decisions, R has the `case_when()` function.

DataFrame Mutating

`mutate()` **does not change** the original **DataFrame**.

If you want, you can **overwrite** it using:



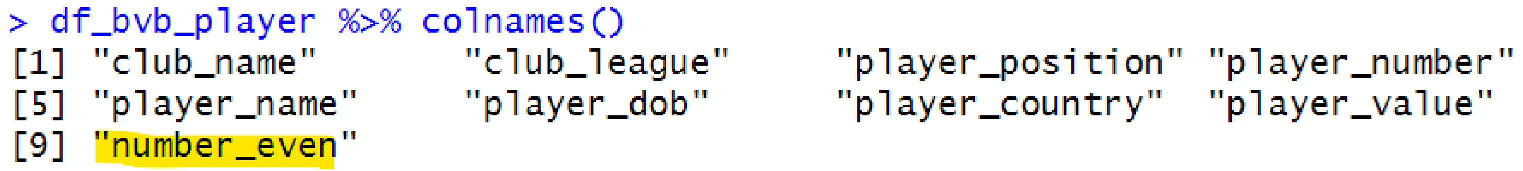
```
> #Overwrite DataFrame with assignment
> df_bvb_player = df_bvb_player %>% mutate(number_even = ifelse(player_number %% 2
== 0,T,F))
> #Overwrite DataFrame with arrow assignment
> df_bvb_player <- df_bvb_player %>% mutate(number_even = ifelse(player_number %% 2
== 0,T,F))
> #Overwrite DataFrame with double pipe assignment
> #Requires library(magrittr)
> df_bvb_player %<>% mutate(number_even = ifelse(player_number %% 2 == 0,T,F))
```

`ifelse(cond, value for True, value for False)`

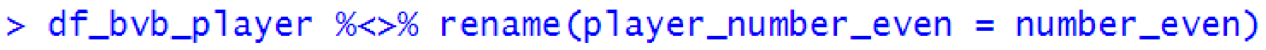
DataFrame Renaming

It often makes sense to **rename** columns and give them **reasonable names**.

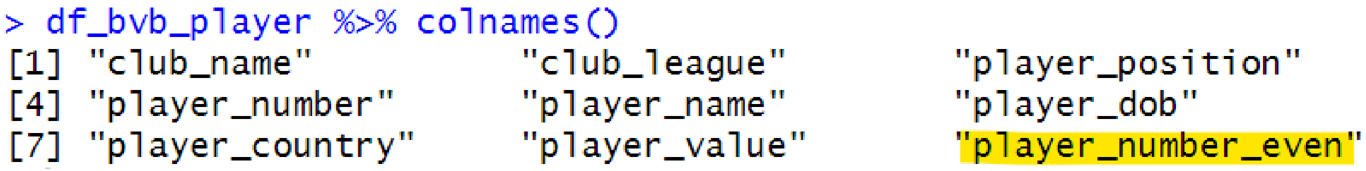
We use the `rename(col_new = col_old)` function and overwrite using the double pipe:



```
> df_bvb_player %>% colnames()
[1] "club_name"       "club_league"     "player_position" "player_number"
[5] "player_name"     "player_dob"      "player_country"  "player_value"
[9] "number_even"
```



```
> df_bvb_player %<>% rename(player_number_even = number_even)
```



```
> df_bvb_player %>% colnames()
[1] "club_name"          "club_league"        "player_position"
[4] "player_number"      "player_name"        "player_dob"
[7] "player_country"     "player_value"       "player_number_even"
```

library(readr)

Training #3

1. Import the following .csv dataset: https://github.com/leotraeg/FHDTM-P2DS-WS2425/raw/refs/heads/main/Praktikum/FHDTM-P2DS-WS2425_PraktikumII_uft8.csv as a DataFrame called **df_dsa** in R using the readr library.
2. View the column names, dimensions, and generate a summary of df_dsa.
3. Return the frequencies of continents of the countries using table().
4. Compare whether more German students went abroad in 2015 or 2010.
    - Rename the attribute names 2015 to s_2015 and 2010 to s_2010.
    - Fill the NA values of s_2015 and s_2010 using mutate( ) and the ifelse(is.na(vector), 0, vector)) statement.
    - You should be able to use the sum( ) method to compare both years.

Takeaways

- Know both Python and R and decide your toolkit based on your personal and other decision factors.
- R, RStudio, and tidyverse offer a broad range for data analysis.
- Similar to Python’s containers, R has vectors.
- Similar to Python’s DataFrame, R also has a DataFrame with filtering and slicing rows and selecting, mutating, and renaming attributes.

Outlook

Next week we will see how to

- Deploy Functions
- Data Preprocessing
- Data Transformation

with R.

See you again next week.

Questions?