Exercise 1

No Fear of Numbers: Introduction to Quantitative Data Analysis in R

Dr. Susanne de Vogel, Data Science Center, University of Bremen

2 December 2025

Research Question: Is there a Life beyond the PhD?

In this workshop, we will work with one coherent example throughout all four parts.We focus on two outcomes of interest:

- `bpsy01`: overall life satisfaction (10 point scale)
- `blcd06`: Children (yes/no)

We will examine how these outcomes are related to different aspects of doctoral researchers' lives and backgrounds:

PhD and Work conditions

- `adbi01`: Status of the doctorate
- `adbi15`: Discipline
- `bdcd17`: Emotional Support during PhD
- `bwdr12`: Perceived scientific pressure
- `bemp81`: Monthly gross income

Attitudes and Well-Being

- `bldc12`: Satisfaction with work-life balance
- `apar15b`: Relationship with parents
- `bpsy05`: Self-Efficacy

Demographics

- `adem01`: Gender
- `adem02`: Age in years
- `apar10`: Highest vocational degree of parents
- `adem03`: Country of birth
- `blcd01`: Relationship status

Before we start: Getting to know RStudio

When you work with R, you usually use ***RStudio*** — a user-friendly interface that makes it easier to write, run, and organize your R code. RStudio typically shows **four main panels** side by side:

Source (top left)

This is where you **write your code** — in scripts or R Markdown file. this is your main workspace.

Everything you write in here is treated by R as code. To structure your code with headlines or to comment your code, write a `#` in the beginning of the line to tell R that the following content is a comment. Hence, R will not read it as code.

To run your code, click the `Run` button at the top right of the `source` window and tell R to either `run selected lines` or `run all`.

Console (bottom left)

This is where **R executes your code**. You'll see results printed here.

You can also type commands directly into the Console, but they won't be saved. That's why it's best to write your code in a script or R Markdown file instead.

Environment / History (top right)

This shows which objects (datasets, variables, results, etc.) are **currently loaded in memory**:

- Environment: list your data and variables.
- History: keep track of all commands you've recently used.

Files / Plots / Packages / Help / Viewer (bottom right)

This panel has several **useful tabs**:

- Files: browse your working directory.
- Plots: view your generated plots.
- Packages: install or activate R packages.
- Help: access documentation (e.g., type ?mean in the Console).
- Viewer: display interactive outputs, such as HTML reports.

Exploring datasets

In this exercise, we will use R tidyverse to explore the dataset. The goal is to learn different ways to ***examine the dataset and its variables*** as a ***preparation for analysis***. You will learn how to

- install and load packages
- read SPSS datasets with tidyverse's `haven` package
- use different functions to inspect the dataset's structure, variable types and classes, labels, and missing values.

Exercise 1.1: Installing and loading packages

Okaaaaaaay, let's go!

1. Install `tidyverse` and `haven` if not already installed.

As a first step, we need to make sure all packages we will use for our data exploration are installed and up-to-date: **tidyverse**, but also **haven**.

When using the packages for the *first time*, you could simply install it with `install.packages()`, e.g. `install.packages("tidyverse")`.

Alternatively, you can use a *conditional check* like

```
if (!requireNamespace("tidyverse", quietly = TRUE)) install.packages("tidyverse")
```

`requireNamespace("tidyverse", quietly = TRUE)` returns TRUE if the package is available. The `!` means "not", so the condition becomes TRUE only when the package is missing. In that case, `install.packages("tidyverse")` is executed.

This checks automatically whether the package is already installed.

Solution:

```
# Install tidyverse
if (!requireNamespace("tidyverse", quietly = TRUE)) # quietly = TRUE means that R runs the function
 ↪  silently (it doesn't show messages or warnings in the console)
  install.packages("tidyverse")

# Install haven
if (!requireNamespace("haven", quietly = TRUE))
  install.packages("haven")
```

2. Use the `library()` function to load *tidyverse* and *haven* so that we can work with them.

Installing packages need to happen only once, but in order to work with them, you need to load them *every time* of the beginning of a new R session.

Hint: Use the same name of the package, but *without* quotation marks.

Solution:

```
# Load the packages (every time you start R)
library(tidyverse)
library(haven)
```

Note: What happens when you load tidyverse?

- You might see a **Warning** like: *"Package was built under R version 4.x.y"*. This is **not an error**. It only means the package was created with a newer R version. In almost all cases, everything will work fine.
- R shows which **core tidyverse packages** are attached automatically: `ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, `tibble`, `stringr`, `forcats` etc.
- Under **Conflicts**, R warns you if functions from different packages have the same name (e.g. `filter()` exists in both `dplyr` and `stats`). By default, the tidyverse version is used. If you want the other one, call it explicitly, e.g. `stats::filter()`.

Exercise 1.2: Opening a dataset from an SPSS file

To work with data in R, we then need to import a dataset as a data frame. The function `read_sav()` from the tidyverse `haven` package helps us read **SPSS files**.

1. For this exercise, we will use the file *01_qa_exploring_data.sav* in the *exercises* folder. Define the *exercise* folder as your working directory with the function `setwd("path")` (Note that R uses / instead of a backslash).

2. Choose `mydata1` as a name for your data frame and import it with the following structure: `chosen_name <- read_sav("filename")`.

Solution:

```
# define working directory
setwd(
 ↪  "C:/Users/Susanne/Nextcloud/share_DSC/003_Trainings/001_Workshops/2025/2025-12-02_SdV_Data_Analysis/Materials/E
 ↪  )
# Load the Nacaps dataset and name this data frame "mydata1"
mydata1 <- read_sav("01_qa_exploring_data.sav")
```

3. On the right side, in the Environment pane, the data frame `mydata1` should now appear under *Data*. What kind of information about the data frame do you find here?

Solution:

Firstly, there is the **number of observations**, e.g. the number of participants in your data frame. This corresponds to the number of rows in your data frame. Secondly, the **number of variables**, e.g. the attrributes in your data frame. This corresponds to the number of columns in your data frame.

Exercise 1.3: Exploring the structure of your data frame

To check whether the data are in a tidy structure, **tidyverse** provides a range of functions that allow us to explore the data frame and its variables.

1. Use the function `glimpse(mydata1)` to take a first look at your data frame. What kind of information do you get from the output?

Solution:

```
# Exploring the structure of the data frame
glimpse(mydata1)
```

```
## Rows: 2,354
## Columns: 17
## $ pid       <dbl> 2, 15, 39, 55, 75, 81, 90, 104, 116, 117, 137, 165, 166, 167~
```

```
## $ adbi15    <dbl+lbl>  7,  4,  1,  3,  5,  1,  4,  4,  3,  4,  1,  4, NA,  7, ~
## $ adem01    <dbl+lbl> 2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 3, 2, 2,~
## $ adem02    <dbl> 27, 25, 41, 30, 26, 25, 27, 28, 34, 29, 34, 27, 42, 29, 31, ~
## $ adem03    <dbl+lbl> 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,~
## $ apar10    <dbl+lbl> 3, 1, 2, 3, 3, 1, 3, 2, 3, 2, 2, 2, 1, 2, 1, 3, 1, 1, 2,~
## $ apar15b   <dbl+lbl>  3,  4,  2,  3,  3,  3,  4,  3,  3,  3,  4,  3,  3,  4, ~
## $ bdbi01    <dbl+lbl> 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,~
## $ bdcd17    <dbl> 3.000000, 4.333333, 4.000000, 4.666667, 1.000000, 3.333333, ~
## $ bemp81    <dbl> 3076, 0, 0, 0, 985, 1762, 4266, 4381, NA, NA, 2592, 660, 428~
## $ bemp81_g3 <dbl+lbl>  3, NA, NA, NA,  1,  2,  3,  3, NA, NA,  3,  1,  3,  3, ~
## $ bdwr12    <dbl> 2.50, 2.50, 2.00, 2.50, 1.00, 3.50, 2.50, 2.50, NA, NA, 4.25~
## $ bpsy01    <dbl+lbl>  5,  7,  7,  8,  5,  5,  7,  7, NA, NA,  8,  5,  8,  9, ~
## $ blcd01    <dbl+lbl>  2,  2,  1,  1,  1,  2,  2,  2, NA, NA,  1,  2,  1,  1, ~
## $ blcd06    <dbl+lbl>  2,  2,  2,  2,  2,  2,  2,  2, NA, NA,  1,  2,  1,  2, ~
## $ blcd12    <dbl+lbl>  8,  3,  8,  8,  5, 10,  3,  5, NA, NA,  3,  7,  9,  9, ~
## $ bpsy05    <dbl> 5.000000, 4.000000, 4.333333, 3.666667, 4.666667, 5.000000, ~
```

The function `glimpse()` provides a compact overview of the data frame.It shows

- how many rows and columns the data frame has
- the name of each variable
- the order of the variables
- the data type (e.g., character, double, integer, factor or dbl+lbl)
- a preview of the first few values

2. Execute the functions `head(mydata1)` and `tail(mydata1)`. What is the difference to `glimpse()`? Which information do you get?

Solution:

```
# display the first six rows of the dataset
head(mydata1)
```

```
## # A tibble: 6 x 17
##     pid adbi15      adem01  adem02 adem03  apar10  apar15b bdbi01  bdcd17 bemp81
##   <dbl> <dbl+lbl>   <dbl+l>  <dbl> <dbl+l> <dbl+l> <dbl+l> <dbl+l>  <dbl>  <dbl>
## 1     2 7 [enginee~ 2 [mal~     27 1 [in ~ 3 [Oth~ 3 [wel~ 1 [I a~   3      3076
## 2    15 4 [mathema~ 2 [mal~     25 1 [in ~ 1 [PhD~ 4 [ver~ 1 [I a~   4.33      0
## 3    39 1 [humanit~ 1 [fem~     41 2 [in ~ 2 [Bac~ 2 [not~ 1 [I a~   4         0
## 4    55 3 [law, ec~ 1 [fem~     30 1 [in ~ 3 [Oth~ 3 [wel~ 1 [I a~   4.67      0
## 5    75 5 [human m~ 1 [fem~     26 1 [in ~ 3 [Oth~ 3 [wel~ 3 [I h~   1       985
## 6    81 1 [humanit~ 2 [mal~     25 1 [in ~ 1 [PhD~ 3 [wel~ 1 [I a~   3.33   1762
## # i 7 more variables: bemp81_g3 <dbl+lbl>, bdwr12 <dbl>, bpsy01 <dbl+lbl>,
## #   blcd01 <dbl+lbl>, blcd06 <dbl+lbl>, blcd12 <dbl+lbl>, bpsy05 <dbl>
```

```
# display the last six rows of the dataset
tail (mydata1)
```

```
## # A tibble: 6 x 17
##     pid adbi15      adem01  adem02 adem03  apar10  apar15b bdbi01  bdcd17 bemp81
##   <dbl> <dbl+lbl>   <dbl+l>  <dbl> <dbl+l> <dbl+l> <dbl+l> <dbl+l>  <dbl>  <dbl>
## 1 28277 3 [law, ec~ 1 [fem~     29 1 [in ~ 2 [Bac~ 2 [not~ 1 [I a~   4      3074
```

```
## 2 28301 5 [human m~ 2 [mal~     30 2 [in ~ 1 [PhD~ 4 [ver~ 1 [I a~   5      2015
## 3 28335 4 [mathema~ 2 [mal~     39 1 [in ~ 1 [PhD~ 3 [wel~ 2 [I h~   2.67   5897
## 4 28337 3 [law, ec~ 2 [mal~     31 1 [in ~ 2 [Bac~ 4 [ver~ 1 [I a~   4.33     50
## 5 28341 1 [humanit~ 1 [fem~     29 1 [in ~ 1 [PhD~ 4 [ver~ 1 [I a~   4.67   2182
## 6 28342 7 [enginee~ 1 [fem~     31 1 [in ~ 1 [PhD~ 4 [ver~ 1 [I a~   5        NA
## # i 7 more variables: bemp81_g3 <dbl+lbl>, bdwr12 <dbl>, bpsy01 <dbl+lbl>,
## #   blcd01 <dbl+lbl>, blcd06 <dbl+lbl>, blcd12 <dbl+lbl>, bpsy05 <dbl>
```

`head(mydata1)` shows the first few rows (by default 6). `tail(mydata1)` shows the last few rows (by default 6).

Tipp: If you want to display more or less rows than the first/last six rows, you can specify it using e.g. `head(mydata1, 10)` for the first ten rows.

With these functions, you see ***actual records from your dataset***, row by row, across all variables.

3. Inspect the data frame in the RStudio Data Viewer using the function `View(mydata1)` (be careful: R is case sensitive, so use V with a capital letter). What can you see here?

Solution:

```
View(mydata1)
```

The function `View()` opens the dataset in a spreadsheet-like window inside RStudio. You can

- investigate the investigate the variables (columns), cases (rows) and values (cells) to check for a tidy data structure.
- check variable names, labels and variable orders.
- scroll through rows and columns, sort variables, and visually inspect values. This is helpful for spotting obvious data entry errors, strange characters, or missing values.

Note: `View()` is only for looking at the data, **not for editing**. Any changes you make inside the Data Viewer will not be saved to your dataset. If you want to clean or transform the data, you need to use functions like `mutate()`, `filter()`, or `select()` in your code.

Exercise 1.4: Inspecting variables

To understand what kind of variables we are dealing with, we now inspect a few example variables in more detail.

Your task is to find out which type and class of variable we have, which labels are available, whether there are any missings (`NA`s).

1. Inspect a categorical variable: `adem01` (gender).

To call not a whole data frame, but a *specific variable* in that data frame, write `mydata1$adem01`. You can read this as: "the variable `adem01` from the dataset mydata1":

- `mydata1` = name of the dataset (the table)
- `$` = "take this specific column from the table"
- `adem01` = name of the variable / column

1a. Use `class()` or `glimpse()` to inspect the variable `adem01`. What type of variable is it in R?

Solution:

```
# check variable type
class(mydata1$adem01)
```

```
## [1] "haven_labelled" "vctrs_vctr"     "double"
```

```
# use glimpse() again
glimpse(mydata1$adem01)
```

```
##  dbl+lbl [1:2354] 2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 3, 2, 2, ...
##  @ label      : chr "Gender"
##  @ format.spss: chr "F8.2"
##  @ labels     : Named num [1:3] 1 2 3
##   ..- attr(*, "names")= chr [1:3] "female" "male" "other"
```

`adem01` is stored as a labelled numeric variable (`<dbl+lbl>`). This means: internally it is numeric (stored as double), but it comes with value labels from SPSS.

1b. Use `attr()` with the options `"label"` and `"labels"` to check whether a variable label and value labels are stored for `adem01`.

Solution:

```
attr(mydata1$adem01, "label")      # variable label
```

```
## [1] "Gender"
```

```
attr(mydata1$adem01, "labels")     # value labels
```

```
## female   male  other
##      1      2      3
```

The variable label is *Gender*, the values 1 to 3 are labelled 1 = *female*, 2 = *male* , 3 = *other*

1c. Create a simple frequency table for `adem01` using `count()`.

Hint: Use the pipe operator `%>%` to first pass the whole data frame into `count()`, e.g. `mydata1 %>% count(adem01)`, because `count()` needs a data frame as input and cannot be applied directly to a single variable like `mydata1$adem01`.

Solution:

```
# Take the dataset mydata1 and then count the occurrences of each value of adem01.
mydata1 %>%
  count(adem01)
```

```
## # A tibble: 4 x 2
##   adem01          n
##   <dbl+lbl>   <int>
## 1  1 [female]  1153
## 2  2 [male]    1068
## 3  3 [other]    132
## 4 NA              1
```

1d. Check whether there are any missing values using the `sum(is.na())` function.

Solution:

```
# 3) Check for missing values
sum(is.na(mydata1$adem01))
```

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

There is one missing value.

2. Inspecting a numeric variable: `adem02` (age in years)

2a. Use `class()` and `glimpse()` to check the type of the variable `adem02`.

Solution:

```
# check variable type
class(mydata1$adem02)
```

```
## [1] "numeric"
```

```
# use glimpse() again
glimpse(mydata1$adem02)
```

```
##  num [1:2354] 27 25 41 30 26 25 27 28 34 29 ...
##  - attr(*, "label")= chr "Age in years 2019"
##  - attr(*, "format.spss")= chr "F8.2"
```

2b. Use `attr()` to check the variable and value labels. Anything noticeable?

Solution:

```
attr(mydata1$adem02, "label")      # variable label
```

```
## [1] "Age in years 2019"
```

```
attr(mydata1$adem02, "labels")     # value labels
```

```
## NULL
```

We see that there are no value labels for numeric variables, as their values are self-explanatory.

2c. Use `summary()` and `count()` to get a quick overview of the range of ages.

Solution:

```
# quick numeric summary
summary(mydata1$adem02)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   22.00   28.00   30.00   31.45   33.00   68.00      15
```

```r
# Take the dataset mydata1 and then count the occurrences of each value of adem02.
mydata1 %>%
  count(adem02)
```

```
## # A tibble: 47 x 2
##    adem02     n
##     <dbl> <int>
##  1     22    25
##  2     23    15
##  3     24    19
##  4     25   113
##  5     26   113
##  6     27   236
##  7     28   215
##  8     29   275
##  9     30   288
## 10     31   195
## # i 37 more rows
```

There are 15 missing values.

3. Inspecting a key outcome variable: `bpsy01` (overall life satisfaction)

`bpsy01` will later be an important outcome variable in our analyses. Before we use it, we want to understand its type, its label and its range of values.

3a. Inspect the class, variable label and value labels of `bpsy01`.

Solution:

```r
# check class of variable
class(mydata1$bpsy01)
```

```
## [1] "haven_labelled" "vctrs_vctr"     "double"
```

```r
glimpse(mydata1$bpsy01)
```

```
##  dbl+lbl [1:2354]  5,  7,  7,  8,  5,  5,  7,  7, NA, NA,  8,  5,  8,  9,  ...
##  @ label      : chr "Overall life satisfaction"
##  @ format.spss: chr "F8.0"
##  @ labels     : Named num [1:2] 0 10
##   ..- attr(*, "names")= chr [1:2] "0 not at all satisfied" "10 fully satisfied"
```

```r
# check labels
attr(mydata1$bpsy01, "label")
```

```
## [1] "Overall life satisfaction"
```

```r
attr(mydata1$bpsy01, "labels")
```

```
## 0 not at all satisfied     10 fully satisfied
##                      0                     10
```

Overall life satisfaction is measured on a 10-point scale is stored as a labelled numeric variable (`<dbl+lbl>`). This means: internally it is numeric (stored as double), but it comes with value labels from SPSS.

3b. Use `summary()` to look at the distribution and `count` to see, how often each value occurs.

Solution:

```
# quick numeric summary
summary(mydata1$bpsy01)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   0.000   6.000   7.000   7.036   8.000  10.000     157
```

```
# Take the dataset mydata1 and then count the occurrences of each value of adem02.
mydata1 %>%
  count(bpsy01)
```

```
## # A tibble: 12 x 2
##    bpsy01                          n
##    <dbl+lbl>                   <int>
##  1  0 [0 not at all satisfied]    10
##  2  1                             13
##  3  2                             43
##  4  3                             92
##  5  4                             87
##  6  5                            137
##  7  6                            243
##  8  7                            531
##  9  8                            596
## 10  9                            342
## 11 10 [10 fully satisfied]       103
## 12 NA                            157
```

Take Home Checklist: Exploring a new dataset in R

Step	What to do when you open a new dataset in R	Useful functions / questions
1	Check the overall structure	`glimpse()`, `head()`, `tail()`, `View()`
2	Understand key variables	`class()`, `attr(..., "label")`, `attr(..., "labels")`
3	Look at typical value ranges	`summary()`, `count()`
4	Identify missing and special codes	`is.na()`, unusual values like `-996`, `-998`, `-989`
5	Take notes	Which variables look promising? Which ones clearly need cleaning?