Text as Data

Session 2: Basics in R, Part 1

Mirko Wegemann

University of Münster
Institute for Political Science

22 April 2026

What is R?

R is an open-source programming language that enables us in the social sciences to perform and visualize statistical analyses.

It is available for Windows as well as Mac (+ UNIX).

What is RStudio?

`R` is a programming language. We can use it in the OS console.

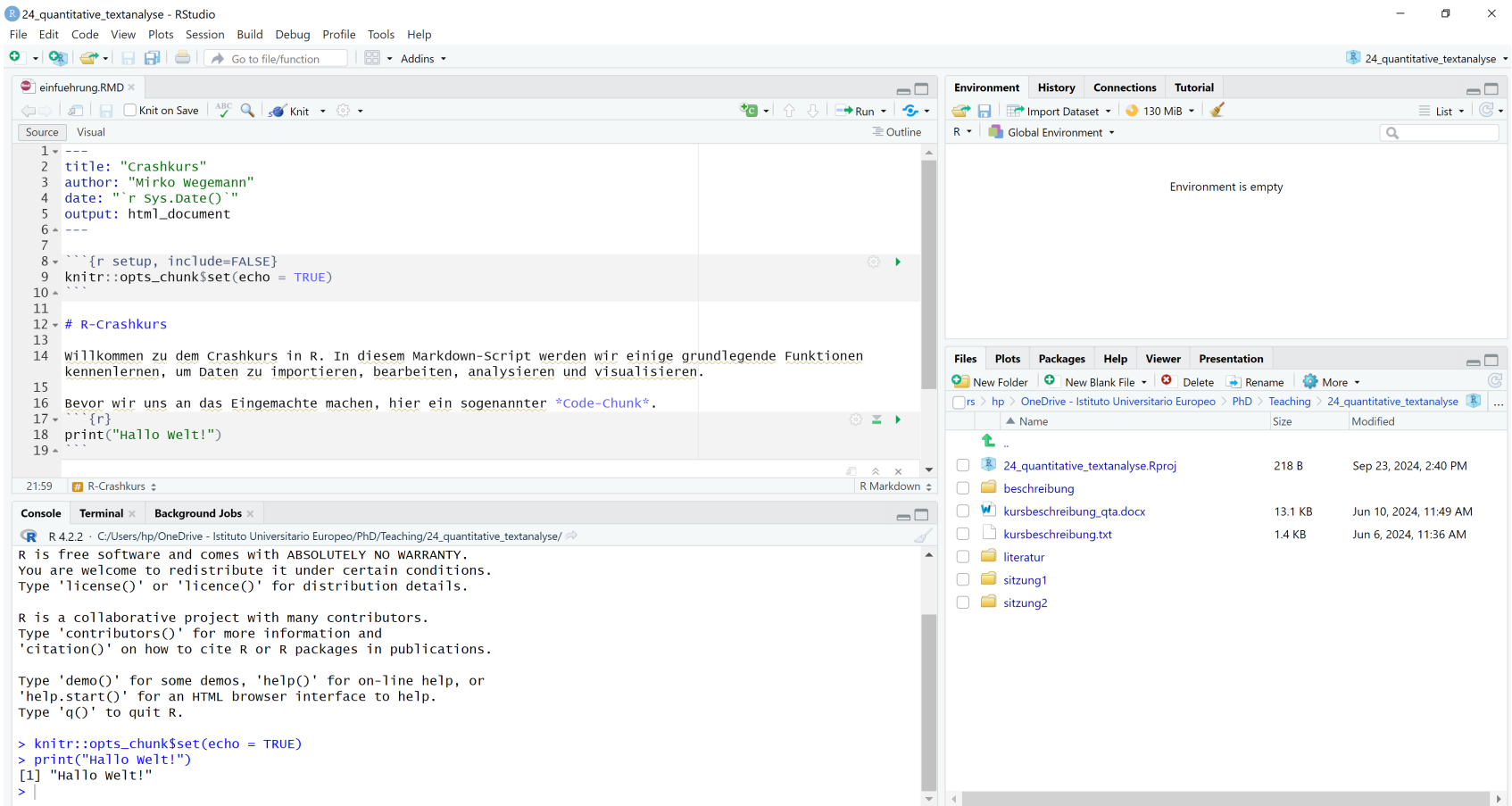
RStudio also provides a graphical interface. It shows us, for example, graphics and environment variables.

Installation of RStudio

Our software environment can be set up in the following steps:

1. Download and install R
2. Download and install RStudio
3. For now optional, but later useful for some packages: RTools

Structure of RStudio I



Structure of RStudio II

R consists of four **Workspace Panes**.

1. **Source** → this is where we write our code
2. **Console** → this is where we see the output of our code
3. **Environment** → this shows all objects that we have created with our code
4. **Files/Plots/Packages/Help** → here we see, among other things, the files that are in the same folder structure or the graphics we generate

Why do we need R?

- Importing files (datasets in various formats, e.g., .xlsx, .csv, .dta, .spss, .R, .RDS, .txt) → this week
- Data preparation (e.g., renaming and creating variables, aggregating data, filtering datasets) → this week
- Visualization → this week
- Regression analyses → this week
- Communicating with websites (e.g., web-scraping) → functions or loops (*for-loops*) → later
- Text analysis (e.g., via dictionaries, topic models, scaling models, neural networks) → later

Tutorial

Open the tutorial. The first part of the tutorial is to understand object types in R. The second part covers data wrangling. For the second part, you can switch between the slides and the tutorial. The slides explain basic functions of dplyr().

For any further assistance, refer to Wickham et al. (2023) or the R help file (?command).

Functions in R

With the installation of R we already have access to a large number of pre-installed functions (the so-called *base R*).

- Some analyses (such as our text analyses) however require additional user-written functions
- freely available through *CRAN*, which usually also provides extensive documentation for the packages
- in R we install these packages with the command *install.packages*(*PACKAGENAME*) and then load them into the workspace with *library*(*PACKAGENAME*)

An example:

```
1
2   > install.packages("tidyverse")
3   > library("tidyverse")
```

Importing Data Sets

R can handle a wide variety of file formats. For this week, we only focus on read.csv().

```
1
2     data <- read.csv("./session2/ess11.csv")
3     View(data)
```

Various Methods I

In R we have several methods available for modifying data

- *base R*: the built-in original method for transforming vectors
- *dplyr* (from the *tidyverse*): a streamlined approach that emphasises code readability
- *data.table*: syntactically closer to *base R* but optimised for efficiency

Various Methods II

All methods have their pros and cons. In this seminar we will mainly use *dplyr* because of its relative simplicity. Depending on the size of the data structures, we will also demonstrate applications of *data.table*.

The Pipe Structure

dplyr is best known for its pipe syntax. The idea: we can apply a series of operations sequentially to a single object (e.g., a data.frame).

- Pipe chains always start with the name of the object we want to modify
- At the end of each command a pipe %>% follows

Creating a New Variable

Creating a new variable is done with the *mutate* function

```
1   > df <- df %>%
2   > mutate(new_var = "value")
```

Generating/Modifying Variables with Conditions

Often we create new variables that are transformations of existing ones. For example, we might categorise an income variable that was originally recorded as a continuous amount. The *case_when()* function is especially handy for this.

```
> df <- df %>%
> mutate(inc_cat = case_when(
> inc < 1000                          "<1000 Euro",
> inc >= 1000 & inc < 2000            "1000-2000 Euro",
> inc >= 2000 & inc < 3000            "2000-3000 Euro",
> inc >= 3000                         ">3000 Euro"))
```

Filtering the Data Set

A raw data set sometimes contains cases that are irrelevant for our analysis. We can remove them with the *filter()* function. In the example below we restrict the data set to cases from the former East German states.

```
1   > df <- df %>%
2   > filter(eastwest == "NEUE BUNDESLAENDER")
```

Aggregating Data

One of the intuitive innovations of the *dplyr* package is that we can first group data with *group_by()* and then aggregate it with *summarise()*, all in a clear, readable way.

```
> df_aggr <- df %>%
> group_by(eastwest) %>%
> summarise(average_income = mean(inc, na.rm =
    TRUE))
```

Common operations include the mean (*mean()*), median (*median()*), minimum (*min()*), maximum (*max()*) or simply counting observations in a category (*n()*).

Selecting Specific Variables

We do not have to use *dplyr* to assign our modified object to a new or existing data structure (via *new_obj* $<-$). Instead, we can display selected variables directly. The *select()* function is useful for this; it shows only the chosen variable(s) rather than the whole data set. In the example below we first sort the data with *arrange()* and then select the variable of interest.

```
> df4 %>%
> arrange(desc(inc)) %>%
> select(inc)
```

Visualization in R

In R there are several ways to visualize distributions and relationships. We mainly use the *ggplot2* package from the *tidyverse*.

A Histogram

The structure of ggplot is simple. We can use the pipe to first specify the data set that contains the target variable (here *inc*). Then, in the main call we define the *aesthetics*, i.e., the variables we want to plot. By adding a plus sign we can choose one or more geometries, in this case a histogram.

```
1   > df4 %>%
2   >   ggplot(aes(inc)) +
3   >   geom_histogram()
```

Bivariate Relationships

Visualizations are especially helpful when we want to display bivariate relationships (associations between two variables). Here we plot a scatterplot **and** a regression line (in this case based on local effect sizes).

```
1   > df4 %>%
2   >   ggplot(aes(hhinc, inc)) +
3   >   geom_point() + geom_smooth()
```

Regression Analyses

In R multivariate regression analyses can be performed easily. There are many packages, depending on the model that underlies our data.

- *lm(y x1 + x2, data)* estimates a multivariate linear regression
- *glm(y x1 + x2, data, family="binomial")* estimates a logistic regression

A Linear Regression Model

In the following example a linear regression model is estimated for the human-capital theory. Income is modeled as a function of education and work experience.

```
1  > m1 <- lm(inc ~ educ + experience, data)
2  > summary(m1)
```

Literature I

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science*. O'Reilly Media, Inc.