

Quantitative Textanalyse

Sitzung 2: Crashkurs in R

Mirko Wegemann

Universität Münster
Institut für Politikwissenschaft

16. Oktober 2024

Orga

- Fragen zum Syllabus? (Literatur, Themen, Erwartungen)
- Tool zum Einreichen von Seminarleistungen online (nach Sitzung 4 möglich)

Was ist R?



R ist eine Open-Source Programmiersprache, die es uns in den Sozialwissenschaften ermöglicht, statistische Analysen durchzuführen und zu visualisieren.

Es ist sowohl für Windows als auch Mac verfügbar (+ UNIX).

Was ist RStudio?



R ist eine Programmiersprache. Wir können sie in der OS-Konsole nutzen.

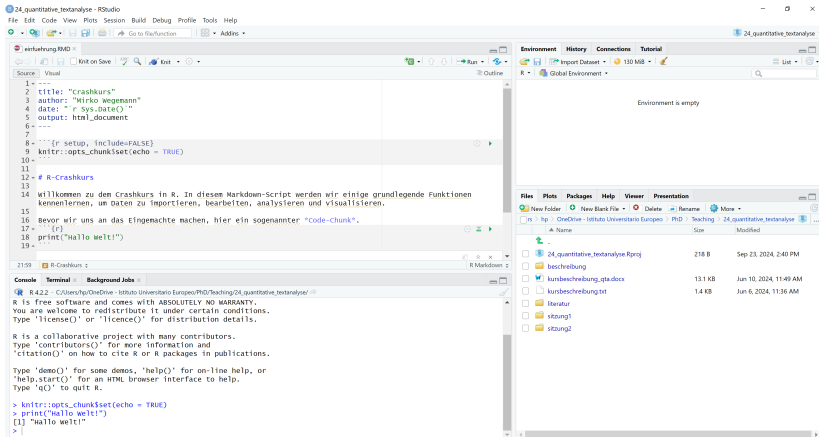
RStudio bietet darüber hinaus eine grafische Oberfläche. Es zeigt uns bspw. Grafiken und Umgebungsvariablen an.

Installation von RStudio

Unsere Softwareumgebung lässt sich in den folgenden Schritten herstellen:

1. Download und Installation von R
2. Download und Installation von RStudio
3. optional, aber später für einige Packages hilfreich: RTools

Aufbau von RStudio I



The screenshot shows the RStudio interface with the following components:

- Source Editor:** Contains an R Markdown file named "einführung.RMD". The code includes a title "Crashkurs", author "Mirko Wegemann", and a chunk of R code that prints "Hallo Welt!".
- Console:** Shows the output of the R code execution, including the R license notice and the printed message "Hallo Welt!".
- Environment:** Shows "Environment is empty".
- Files Panel:** Displays a file explorer view of the current project directory, listing files like "beschreibung", "kursbeschreibung_rta.docx", and "situng1".

Aufbau von RStudio II

R besteht aus vier **Workspace Panes**.

1. **Source** → hier schreiben wir unseren Code hinein
2. **Console** → hier sehen wir den Output unseres Codes
3. **Environment** → hier sehen wir alle Objekte, die wir durch unseren Code generiert haben
4. **Files/Plots/Packages/Help** → hier sehen wir u.a. die Dateien, welche sich in der gleichen Ordnerstruktur befinden oder Grafiken, welche wir generieren

Wofür brauchen wir R?

- Import von Dateien (Datensätze in unterschiedlichen Formaten, z.B. .xlsx, .csv, .dta, .spss, .R, .RDS, .txt)
- Datenaufbereitung (z.B., Variablenumbenennung und -generierung, Aggregation von Daten, Filtern von Datensätzen)
- Kommunikation mit Webseiten (vgl. Web-Scraping) → Funktionen bzw. Schleifen (*for-loops*)
- Textanalyse (z.B. über Dictionaries, Topic Models, Scaling Models, Neural Networks)
- Visualisierung
- evtl. einfache Regressionsanalysen

Tutorial – 1. Schritt

Was erinnert ihr noch in \mathbb{R} ? Findet es heraus und versucht euch in Kleingruppen an dem 1. Schritt des Tutorials.

Funktionen in R

Mit der Installation von R können wir bereits auf eine Vielzahl an vorinstallierten Funktionen zugreifen (sogenanntes *baseR*).

- Einige Analysen (wie bspw. unsere Textanalysen) bedürfen aber weiterer, benutzergeschriebener Funktionen
- kostenlos verfügbar über *CRAN*, welches in der Regel auch über eine ausführliche Dokumentierung der Pakete verfügt
- in R installieren wir diese Pakete über den Befehl `install.packages(PACKAGENAME)` und laden sie anschließend durch `library(PACKAGENAME)` in die Arbeitsumgebung

Ein Beispiel:

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

Import von Datensätzen

R kann mit einer Vielzahl an Dateiformaten umgehen.

.txt-Files

```
1 > txt <-  
  readLines("./sitzung2/SnowballStopwordsGerman.txt")  
2 > head(txt)  
3 [1] "aber          | but" ""  
  "alle          | all" "allem"  
4 [5] "allen"                "aller"
```

Import von Datensätzen II

.RDS-Files

```
1 > df <- readRDS("./sitzung2/speeches_german.RDS")
2 > summary(df)
3 date                agenda                speechnumber
   speaker                party
4 Min.      :1991-03-12   Length:2000           Length:2000
   Length:2000           Length:2000
5 1st Qu.:1991-11-26   Class :character   Class
   :character   Class :character   Class :character
6 Median :1992-05-20   Mode  :character   Mode
   :character   Mode  :character   Mode  :character
7 Mean    :1992-06-19
8 3rd Qu.:1993-01-22
9 Max.    :1993-09-09
```

Import von Datensätzen III

Je nach Format benötigen wir allerdings separate Pakete.

- *openxlsx* für Excel-Files
- *readstata13* für Stata-Files
- *haven* für Stata oder SPSS-Files
- *pdftools* und *tesseract* für pdf-Files

Import von Datensätzen IV

.xlsx-Files

```
1 > df3 <- read.xlsx("./unemployment_1222.xlsx")
```

Import von Datensätzen V

.dta-Files

```
1 > df4 <- read.dta13("./allb18.dta")
```

Import von Datensätzen VI

.pdf-Files

```
1 > syllabus <- pdf_ocr_text("qta_syllabus.pdf")
2 First use of Tesseract: copying language data...
3
4 Converting page 1 to qta_syllabus_1.png... done!
5 Converting page 2 to qta_syllabus_2.png... done!
6 Converting page 3 to qta_syllabus_3.png... done!
```


Verschiedene Methoden I

In R stehen uns verschiedene Methoden zur Modifikation von Daten zur Verfügung

- *base R*: die vorprogrammierte Ursprungsmethode, um Vektoren zu transformieren
- *dplyr* (aus dem *tidyverse*): eine angepasste, vereinfachte Methode, welche sich vor allem auf die Lesbarkeit des Codes fokussiert
- *data.table*: ähnelt in ihrer Schreibweise eher *baseR*, ist aber auf Effizienz spezialisiert

Verschiedene Methoden II

Alle Methoden haben ihre Vor- und Nachteile. Wir werden in diesem Seminar aufgrund der relativen Einfachheit größtenteils mit *dplyr* arbeiten. Je nachdem, wie groß die Datenstrukturen werden, zeigen wir aber auch Anwendungen von *data.table*.

Die Pipestruktur

dplyr ist vor allem für seine Pipestruktur bekannt. Der Clue: wir können verschiedene Optionen sequentiell an einem Objekt (bspw. einem `data.frame`) vornehmen.

- Pipestrukturen fangen immer mit der Benennung des Objektes, welches wir modifizieren wollen, an
- Am Ende jeden Befehls folgt eine Pipe `% > %`

Generieren einer neuen Variablen

Das Erstellen einer neuen Variablen erfolgt durch die Funktion *mutate*

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

Verändern/generieren neuer Variablen mit Bedingungen

Oftmals erstellen wir neue Variablen, die eine Veränderung/Transformation von bereits generierten Variablen sind. Beispielsweise könnten wir eine Einkommensvariable kategorisieren, die zuvor offen abgefragt worden ist. Hierfür ist die Funktion `case_when()` äußerst hilfreich.

```
1 df <- df %>%  
2   mutate(inc_cat = case_when(inc<1000 ~ "<1000  
   Euro",  
3   inc>=1000 & inc<2000 ~ "1000-2000 Euro",  
4   inc>=2000 & inc<3000 ~ "2000-3000 Euro",  
5   inc>=3000 ~ ">3000 Euro"))
```

Filtern des Datensatzes

Ein Datensatz im Rohformat beinhaltet manchmal Fälle, welche wir für unsere Analyse nicht benötigen. Wir können sie über die Funktion *filter()* entfernen. In dem unteren Beispiel beschränken wir unseren Datensatz bspw. nur auf Fälle aus ostdeutschen Bundesländern.

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

Aggregation von Daten

Eine der intuitiven Innovationen des *dplyr*-Pakets besteht darin, dass wir Daten einfach und lesbar zunächst mit *group_by()* gruppieren und sie daraufhin mit *summarise()* aggregieren können.

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

Gängige Operationen sind der Mittelwert (*mean()*), Median (*median()*), Minimum (*min()*), Maximum (*max()*) oder einfach nur das Zählen einer Kategorie (*n()*).

Auswählen von einer bestimmten Variablen

Wir müssen *dplyr* nicht unbedingt verwenden, um unser modifiziertes Objekt einer neuen/existierenden Datenstruktur zuzuweisen (über *new_obj < -*). Stattdessen können wir uns auch Variablen anzeigen lassen. Hierfür ist die *select()*-Funktion hilfreich, welche uns nicht den gesamten Datensatz, sondern nur eine/mehrere Zielvariablen anzeigt. In diesem Fall haben wir die Variable zuvor mit *arrange()* sortiert.

```
1 df4 %>%  
2   arrange(desc(inc)) %>%  
3   select(inc)
```


Regular Expressions I

Eine elementare Operation, wenn wir mit Text arbeiten, sind Regular Expressions.

- Text als Sammlung unterschiedlicher Tokens
- Tokens können klassifiziert werden (beispielsweise wissen wir, dass "6" eine Zahl ist, "W" ein Buchstabe und " " ein Leerzeichen)
- Hierdurch können wir, ohne den genauen Inhalt eines Textes zu kennen, wichtige Information finden, extrahieren, ersetzen oder löschen

Regular Expressions II

In R gibt es unterschiedliche Pakete, welche uns ermöglichen Regular Expressions zu nutzen. Wir greifen auf *stringr* aus dem *tidyverse* zurück.

Es kommt mit unterschiedlichen Funktionen, einige der wichtigsten:

- *str_detect(string, pattern)* erkennt ein gewisses Muster im Text und gibt einen boolean Vektor zurück (WAHR/FALSCH)
- *str_extract(string, pattern)* erkennt ein Muster und extrahiert dieses
- *str_remove(string, pattern)* erkennt ein Muster und löscht es aus dem Text

Regular Expressions III

- `str_replace(string, pattern, replacement)` erkennt ein Muster und ersetzt es durch ein anderes
- `str_count(string, pattern)` erkennt ein Muster und zählt die Anzahl der Vorkommnisse in einem Text
- `str_split(string, pattern)` teilt eine Variable nach einem gewissen Muster auf

Regular Expressions IV

Niemand kann sich alle Regular Expressions merken, eine Übersicht gibt es u.a. hier.

Einige der wichtigsten Regex sind:

- w: matched alle Buchstaben
- d: matched alle Zahlen
- s: matched alle Leerzeichen

Regular Expressions V

stringr hat darüber hinaus noch einige weitere nützliche Regex definiert, u.a.

- `::lower::` : erkennt kleingeschriebene Buchstaben
- `::upper::` : erkennt großgeschriebene Buchstaben
- `::punct::` : erkennt Satzzeichen
- `::alnum::` : erkennt Zahlen und Buchstaben

Regular Expressions VI

Wichtig ist außerdem, wie oft ein Muster auftreten soll, bevor es gematched wird.

- `?` erkennt ein Muster, was 0 oder 1 Mal auftritt
- `+` erkennt ein Muster, was mindestens 1 Mal auftritt
- `*` erkennt ein Muster, was auftritt oder nicht
- `n` erkennt ein Muster, was genau n-Mal auftritt
- `n,` erkennt ein Muster, was n-Mal oder öfters auftritt

Beispiele von Regular Expressions

Ein gängiges Beispiel ist es, Jahreszahlen aus einem Text zu extrahieren. Im folgenden Code werden alle Zahlenfolgen, die aus exakt vier Zahlen bestehen extrahiert.

Beachtet die zwei Backslashes, welche benötigt werden, um den Backslash zu escapen (zu kennzeichnen, dass es sich um Syntax handelt)

```
1 digits <- str_extract_all(syllabus, "\\d{4}")
```

Zählen

Wir können auch die Häufigkeit eines Musters zählen. Hier zählen wir, wie oft das Wort "ECTS" genutzt wird.

```
1 str_count(syllabus, "ECTS")
```


Entfernen

Wir können auch ein Muster entfernen. Ein Standardbeispiel dafür sind Zeilenumbrüche, die meist durch ein "`\n`" gekennzeichnet sind.

```
1 syllabus2 <- str_remove_all(syllabus, "\n")
```

Ersetzen

Wenn wir ein Muster entfernen, dann entfernen wir evtl. auch eine natürliche Trennung zwischen dem Wort oder der Zahlenfolge zuvor und danach. Um das zu verhindern, ersetzen wir es oftmals durch ein Leerzeichen.

```
1 syllabus2 <- str_replace_all(syllabus, "\n", " ")
```

Tutorial – 2. Schritt

Nun wieder zur Anwendung. Im zweiten Schritt des Tutorials nutzen wir eine Textdatei.

Loops I

Die Idee von Loops ist, dass wir eine Operationen an verschiedenen Objekten vornehmen wollen, ohne diese jedes Mal niederzuschreiben. Beispiele sind...

- das Öffnen verschiedener Webseiten (für das Scraping wichtig)
- das Importieren mehrerer Dateien
- das Bearbeiten mehrerer Variablen nach demselben Muster

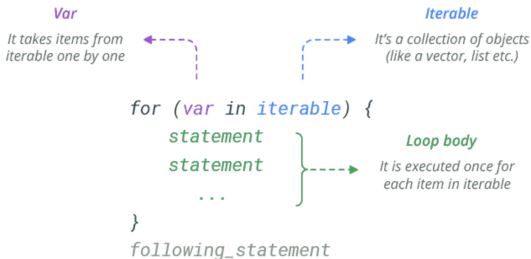
Loops II

Es gibt in R verschiedene Arten von Loops

- *for*-Loops gehen durch jedes Element einer definierten Liste oder Sequenz durch und führen eine Operation durch
- *while*-Loops gehen nur so lange durch eine Liste durch, bis ein bestimmtes Kriterium erfüllt ist

Wir greifen größtenteils auf *for*-Loops zurück.

Die Struktur eines einfachen for-Loops



Tutorial zu for-Loops Grafik und Anleitung zu for-Loops

Ein Beispiel

Im Folgenden sehen wir ein Beispiel eines einfachen Loops, in dem wir einfach durch eine Zahlenreihe von 0 bis 10 gehen und alle Elemente dieser Zahlenreihe anzeigen lassen.

```
1 for(i in 0:10){  
2   print(i)  
3 }
```

Visualisierung in R

In R gibt es mehrere Möglichkeiten, Verteilungen und Zusammenhänge zu visualisieren. Wir nutzen hierzu vor allem das Paket *ggplot2* aus dem *tidyverse*.

Ein Histogramm

Die Struktur von ggplot ist simpel. Wir können die Pipe nutzen, um zuerst den Datensatz festzulegen, aus dem die Zielvariable (in diesem Fall *inc*) stammt. Daraufhin definieren wir in dem Hauptbefehl die *aesthetics*, d.h., die Variablen, welche wir plotten wollen. Über ein Plus-Symbol können wir aus einer oder mehrerer Visualisierungsformen wählen, in diesem Fall ein Histogramm.

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

Bivariate Zusammenhänge

Visualisierungen sind besonders hilfreich, wenn wir bivariate Zusammenhänge (Assoziationen zwischen zwei Variablen) darstellen möchten. Hier plotten wir einen Scatterplot **und** eine Regressionslinie (in diesem Fall basierend auf lokalen Effektgrößen).

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

Regressionsanalysen

In R lassen sich multivariate Regressionsanalysen einfach durchführen. Es gibt viele Pakete, je nachdem, was für ein Modell unseren Daten zugrundeliegt.

- $lm(y \sim x1 + x2, data)$ schätzt eine multivariate lineare Regression
- $glm(y \sim x1 + x2, data, family="binomial")$ schätzt eine logistische Regression

Ein lineares Regressionsmodell

Im folgenden Beispiel wird ein lineares Regressionsmodell zur Humankapitaltheorie geschätzt. Einkommen ist hiernach eine Funktion zwischen Bildung und Berufserfahrung.

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

Ein logistisches Regressionsmodell

Im folgenden Beispiel wird ein logistisches Regressionsmodell zur Wahrscheinlichkeit der Beteiligung an der Bundestagswahl geschätzt.

```
1 m1_log <- glm(vote ~ sex + age + S01 + eastwest +  
    di01a, data=df4, family="binomial")  
2 summary(m1_log)
```

Tutorial – 3. Schritt

Im letzten Teil des Tutorials widmen wir uns der Darstellung und Analyse bi- und multivariater Zusammenhänge.

Ein kurzer Ausblick I

In der nächsten Woche:

- ...beschäftigen wir uns mit einer Begriffsklärung der quantitativen Textanalyse als Form der Inhaltsanalyse
- Hierzu bitte folgende Literatur lesen:
 1. Krippendorff, K. (2018). *Content Analysis: An Introduction to its Methodology* (Fourth Edition). SAGE (Kapitel 2)
 2. Grimmer, J., & Stewart, B. M. (2013). Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts. *Political Analysis*, 21(3), 267–297. <https://doi.org/10.1093/pan/mps028>

Optional: Benoit, K. (2020). Text as data: An overview.. In L. Curini & R. Franzese (Eds.), *The SAGE Handbook of Research Methods in Political Science and International Relations*. SAGE Publications Ltd. <https://doi.org/10.4135/9781526486387>

Literatur I

Benoit, K. (2020). Text as data: An overview.. In L. Curini & R. Franzese (Eds.), *The SAGE Handbook of Research Methods in Political Science and International Relations*. SAGE Publications Ltd.

<https://doi.org/10.4135/9781526486387>

Grimmer, J., & Stewart, B. M. (2013). Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts. *Political Analysis*, 21(3), 267–297.

<https://doi.org/10.1093/pan/mps028>

Krippendorff, K. (2018). *Content Analysis: An Introduction to its Methodology* (Fourth Edition). SAGE.