Einführung in Quarto

Dynamische und reproduzierbare Berichte mit R

Daniela Palleschi

2023-04-25

Inhaltsverzeichnis

Heutige Ziele

- lernen, was dynamische Berichte sind
- unser eigenes Quarto-Dokument erstellen
- lernen, wie man ein Quarto-Dokument bearbeitet
- lernen, wie man Code in ein Quarto-Dokument einfügt
- ein Quarto-Dokument in verschiedenen Formaten wiedergeben

Lust auf mehr Praxis?

- wir können mehr lernen/mehr Übung bekommen, indem wir
  - Kap. 29 und 30 in Wickham et al. (o. J.) (speziell zu Quarto), oder
  - Ch. 2 in Nordmann & DeBruine (2022) (reproduzierbarer Arbeitsablauf im Allgemeinen)
    - Nordmann & DeBruine (2022) verwendet Rmarkdown-Skripte, während wir die nächste Generation verwenden werden: Quarto
    - wir sollten in Quarto immer noch in der Lage sein, genau die gleichen Dinge zu tun, wie sie in Rmarkdown vorgeschlagen werden

1 Wiederholung

Letzte Woche haben wir…

- den Unterschied zwischen R und RStudio kennengelernt
- ein RProjekt für den Kurs erstellt
  - der Projektordner enthält:
    - eine `.RProj`-Datei
    - einen Ordner `daten`, der derzeit leer ist
    - einen Ordner mit dem Namen `notes`, der eure Notizen enthält
- unser erstes R-Skript erstellt
- gelernt, wie man:
  - Pakete installiert mit `install.packages("")`
  - Pakete mit `library()` ladt
  - eine Paketfunktion aufruft, z.B. `beep()`
  - Funktionsargumente verwendet, z.B. `beep(sound = "wilhelm")`

  - einfaches Rechnen in R, z. B. `4+4`
  - Funktionen aufruft, um einfache Berechnungen in R durchzuführen, z.B. `sum(4,5)`, oder `mean(c(4,5))`

1.1 Weitere Einrichtung

- Die Aufrechterhaltung eines reproduzierbaren Arbeitsablaufs erfordert einige Änderungen, wenn euer Projekt wächst
- also lasst uns erstellen:
  - Unterordner in `notes`, einen für jede Woche
    * z.B. einen Ordner `01-rstudio`, in dem wir unser R-Skript von letzter Woche ablegen werden
    * Dies wird euch helfen, eure Ordner zu organisieren und zu strukturieren, wenn ihr im Laufe des Semesters weitere Skripte erstellt
- einen Ordner mit dem Namen `moodle`, in dem ihr jede Woche die Materialien aus Moodle speichert
- Jetzt könnt die Materialordner der einzelnen Moodle-Abschnitte in den Ordner `moodle` herunterladen

Aufgabe 1.1: Ordner für Woche 2

Beispiel 1.1.

1. Fügt einen Unterordner mit dem Namen `02-quarto` in `Notes` hinzu
2. Geht zu Moodle und speichert den Materialordner für '02 - Einführung in Quarto' in eurem `moodle` Ordner
3. Öffnet das Dokument `_blatt.html` auf euren Computer
    - Seht euch das Dokument an; ihr könnt oben rechts auf verschiedene Schaltflächen klicken. Probiert es

2 Quarto

- Quarto ist ein Dateityp, der dynamische Berichte erstellt
- Quarto-Dokumente sehen genauso aus wie ihr Vorgänger, Rmarkdown

2.1 Dynamische Berichte

- diejenigen, die Text, Code, Codeausgabe enthalten

- Quarto bietet ein "unified authoring framework" für Data Science, das euren Text, euren Code und eure Code-Ausgabe einschließt (Wickham et al., o. J., Kap 29.1)
- Quarto wurde entwickelt, um auf drei Arten verwendet zu werden:

1. Für die Kommunikation mit Entscheidungsträgern, die sich auf die Schlussfolgerungen und nicht auf den Code hinter der Analyse konzentrieren wollen.
2. Für die Zusammenarbeit mit anderen Datenwissenschaftlern (einschließlich euch in der Zukunft!), die sowohl an euren Schlussfolgerungen als auch daran interessiert sind, wie t diese erreicht habt (d. h. am Code)
3. Als eine Umgebung, in der Datenwissenschaft betrieben werden kann, als ein modernes Labornotizbuch, in dem t nicht nur festhalten könnt, was t getan habt, sondern auch, was t gedacht habt.

2.2 R v. Rmarkdown v. Quarto

- `.R` -Dateien enthalten nur (R-)Quellcode
- `.Rmd` *dynamische Berichte* mit
  - R-Code (und R-Pakete)
- `.qmd` *dynamische Berichte* (RStudio v2022.07 oder später) mit
  - R-Code (und R-Pakete)
  - Native Unterstützung für Python (und Jupyter-Notebooks)
  - Native Unterstützung für Julia

Aufgabe 2.1: RStudio version

Beispiel 2.1.

1. Führen den folgenden Code in der `Konsole` aus: `RStudio.Version()$version`
   - wenn die ausgegebene Version `2022.07` oder höher ist, könnt euch Quarto benutzen
   - wenn nicht:
2. Aktualisiert euch RStudio: `Help > Check for updates`

2.3 Markdown

- `.md`-Dateien
- ein Klartext-Editor-Format, das

  - Formatierungselemente hinzufügt, die unabhängig von Gerät und Ausgabeformat sind (PDF, Word-Dokument, html…)
  - leicht zu lesen ist

- Markdown-Dokumente sind das Bindeglied zwischen unserem Quelldokument (`.qmd`) und unserer Ausgabe (z.B. PDF)

2.4 Ausgabeformate

- wenn wir das Dokument rendern:

  1. Quarto sendet die `.qmd`-Datei an `knitr` (ein R-Paket für dynamische Berichte mit R)
  2. `knitr` führt die Code-Chunke aus und erstellt ein neues `.md` Dokument mit Code und Ausgabe
  3. die `.md`-Datei wird von `pandoc` verarbeitet, das `.md`-Dateien in die fertige Datei konvertieren kann, mit vielen Ausgabeformaten

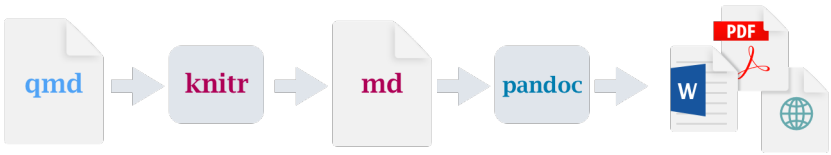

Abbildung 1: Diagramm des Quarto-Workflows von qmd, zu knitr, zu md, zu pandoc, zur Ausgabe im PDF-, MS Word- oder HTML-Format. (Quelle: Wickham et al. (o. J.))

> **Andere Verwendungen**
>
> Quarto kann für eine Vielzahl von Zwecken verwendet werden, wie z. B.:
>
> - Websites/Blogs
> - Notizen machen
> - Dokumentieren von allem, was mit Code zu tun hat, um die Reproduzierbarkeit zu verbessern
>   - Tipps zum Arbeitsablauf
>   - Bearbeitung von csv-Dateien (z. B. Stimuluslisten)

2.5 Folder structure

- jede `.qmd` sollte (normalerweise) in einem eigenen Ordner sein

  - d.h. es sollten nicht mehrere `.qmd` Dateien im selben Ordner sein
- dies ist nur mein Vorschlag, um die Ordner ordentlich und organisiert zu halten
  - d.h., es gibt keinen technischen Grund dafür (die Dokumente laufen auch dann, wenn sie sich alle im selben Ordner befinden)
- werfen wir einen Blick auf einige meiner früheren und aktuellen Projektordner

3 Euer erstes Quarto-Dokument

- letzte Woche haben wir ein R-Skript erstellt, das t über Moodle eingereicht habt
- ihr werdet nun eure erste `.qmd`-Datei erstellen
- von nun an wird dies die Datei sein, die ihr in Moodle einreicht (kein R-Skript)

Aufgabe 3.1: erste Quarto

Beispiel 3.1.

1. Erstellt euch in eurem R-Projekt-Ordner, in dem ihr eure Kursunterlagen/Notizen aufbewahren, einen neuen Ordner für Woche 2
2. `File > New Document > Quarto Document`
   - Gebt euch ihm einen Titel wie "Quarto - Woche 2"
   - Deaktiviert euch die Option "open with Visual Editor".
3. Schau dir das neue Skript an, um mehr über Quarto zu erfahren.
4. Klickt euch auf die Schaltfläche "Render" am oberen Rand des Dokuments
   - Speichert ihr das Dokument in dem Ordner für Woche 2, den ihr gerade erstellt habt.
   - Was geschieht? Vergleicht die Ausgabe mit dem Quellcode des Dokuments.
5. Geht euch zurück zu eurem neuen Ordner `02-quarto`
   - Was hat sich geändert?

3.1 Quarto-Grundlagen

- Quarto-Dokumente (wie Rmarkdown) enthalten drei wichtige Arten von Inhalten:
  1. den **YAML-Header**, der von `---` umgeben ist
  2. Text mit einer einfachen Formatierung oder Strukturierung wie `# Überschrift` oder `*Kursivschrift*`
  3. R-Code-Chunk, umgeben von ```

3.2 YAML

- stand ursprünglich für *Yet Another Markup Language*
  - wurde aber in *YAML Ain't Markup Language* umbenannt, um den Zweck der Sprache als datenorientiert und nicht als Dokumentauszeichnung zu betonen (laut Wikipedia)
- enthält alle Metainformationen zu eurem Dokument
  - z.B. Titel, Autorenname
- auch Formatierungsinformationen
  - z.B. Typ der Ausgabedatei
- es gibt viele Möglichkeiten der Dokumentformatierung und -anpassung, die wir in diesem Kurs nicht behandeln werden
  - aber ich habe zum Beispiel viele YAML-Formatierungsoptionen im Quellcode meiner Folien

Aufgabe 3.2: YAML

Beispiel 3.2.

1. Ändert den Titel, wenn ihr das möchtet
2. Ratet, wie ihr einen "Untertitel" hinzufügen könntet (Hinweis: es ist ähnlich wie beim Hinzufügen eines "Titels")
3. Fügt einen Autor hinzu, `author: "Vorname Nachname"` (siehe Beispiel unten)
4. Fügt ein Inhaltsverzeichnis hinzu, indem ihr `format` so ändert, dass es wie folgt aussieht:

```yaml
---
title: "Quarto - Woche 2"
author: "Vorname Nachname"
format:
  html:
    toc: true
---
```

5. Rendert nun das Dokument. Seht eure Änderungen?

3.3 Strukturierung eures Dokuments

- ihr könnt euer Dokument strukturieren mit

- `# Überschriften`
- `## Zwischenüberschriften`
- `### Unter-Zwischenüberschriften`, usw.

```
# Überschrift 1

Hier ist ein Text über das Thema, das mit dieser Überschrift verbunden ist.

# Überschrift 2

Hier ist ein weiterer Text zu einem anderen Thema.

## Unterüberschrift 2.1

Dies ist ein Text über das Unterthema.
```

Die Bedeutung der Formatierung

Zwischenüberschriften benötigen ein Leerzeichen nach dem letzten Hashtag (`##Zwischenüberschrift` anstelle von `##Zwischenüberschrift`), um als Überschrift gelesen zu werden. YAML erfordert außerdem einen sehr präzisen Schriftsatz. Da die Abstände in der YAML (und anderswo) so wichtig sind, möchte ich die Leerzeichen sehen und zählen können. Um dies zu tun, geht in RStudio:

- geht zu euren Globalen Einstellungen (Werkzeuge > Globale Einstellungen)
- unter `Code` (linke Spalte) `> Display` (Tab), markiert das Kästchen `> Show whitespace character`

Aufgabe 3.3: Überschriften

Beispiel 3.3.

1. Kopiert den obigen Code (Überschriften und Unterüberschriften) und ersetzt den Text in der Quarto-Vorlage.
2. Ersetzt die erste Überschrift durch den Titel `Quarto`
   - Schreibe einen Text, der Quarto beschreibt, unter die Überschrift
3. Schreibe eine Unterüberschrift namens `YAML`
   - Schreibt einen Text, der die YAML-Struktur beschreibt, die wir besprochen haben
4. Erstellt eine Unterüberschrift mit dem Namen `Quarto-Struktur`.

- Schreibt einige Notizen darüber, wie wir ein Quarto-Dokument strukturieren können (z.B. durch das Erstellen von Überschriften)

5. Finde in RStudio die Schaltfläche `Outline` oben links im `.qmd` Text Editor Fenster
   - Was siehst du, wenn du darauf klickst?

3.4 Text formatting

- ihr könnt Wörter betonen:
  - *Kursivschrift*: `_Text_` oder `*Text*`
  - **fett**: `**text**`
  - Unterstreichen: `[text]{.underline}`
  - ~~durchgestrichen~~: `~~text~~`
  - $_{\text{Subskription}}$: `~text~`
  - $^{\text{Hochgestellt}}$: `^text^`
  - SMALLCAPS: `[text]{.smallcaps}`

3.5 Aufzählungen

- ihr könnt Aufzählungslisten mit Bindestrichen erstellen.
  - Unteraufzählungen müssen eingerückt werden (drückt die Tabulatortaste)
- nummerierte Listen können durch einfaches Schreiben einer nummerierten Liste erstellt werden
  - Unteraufzählungen müssen in nummerierten Listen *doppelt* eingerückt werden

```
- dies ist ein Aufzählungszeichen
  + dies ist ein Unterpunkt

1. Dies ist ein nummerierter Punkt
    a. dies ist ein unternummerierter Punkt (beachtt den doppelten Einzug)
2. dies ist der zweite nummerierte Punkt
```

Aufgabe 3.4: Aufzählungen

Beispiel 3.4.

1. Fügt eurem `.qmd` Dokumententext eine Textformatierung hinzu.
2. Füge eine Aufzählungsliste hinzu

3. Füge eine nummerierte Liste hinzu
4. Rendert ihr das Dokument. Hat es geklappt?

4 Codierung in Quarto

- Der große Vorteil von dynamischen Berichten ist die Integration von Text und Code
- Letzte Woche haben wir gelernt, wie man einfache mathematische Berechnungen in R durchführt.
- wie würden wir R-Befehle in ein `.qmd`-Dokument einfügen?
  - Inline-Code (Code, der innerhalb einer Textzeile ausgeführt wird)
  - Code-Chunke (ein Code-Chunk, der nicht in Text enthalten ist)

4.1 Code-Chunks

- Code Chunks sind zwischen ```` ```{r} ```` und ```` ``` ```` eingebettet.
- eine schöne Tastenkombination:`Cmd-Option-I` (Mac) oder `Strg-Alt-I` (PC)

````
```{r}
#| eval: false

# Addition
4+6
```
````

- ihr könnt den Code in eurer RStudio-Sitzung ausführen, indem ihr:
  - auf das kleine grüne Dreieck oben rechts im Chunk klickt
  - die Tastenkombination `Cmd/Strg-Enter` verwendet, um eine einzelne Code-Zeile auszuführen (je nachdem, worauf der Cursor steht)
  - der Tastenkombination `Cmd/Strg-Shift-Enter` benutzt, um den gesamten Code-Chunk auszuführen (falls es mehrere Befehle innerhalb eines einzelnen Abschnitts gibt)

Aufgabe 4.1: Code-Chunks

Beispiel 4.1.

1. Füge einen Code Chunk zu deiner `.qmd` Datei hinzu
   - Füge einige mathematische Operationen ein (Addition, Subtraktion, etc)

- Fügt informative Anmerkungen zu eurem Code hinzu (z.B. `# Addition`)

2. Füge einen Text unter deinem Code-Chunk hinzu, der beschreibt, was der obige Code erreicht hat.
3. Füge einen weiteren Code-Chunk unter diesem Text hinzu.
    - Erstellt einige Objekte, `wert1` und `wert2`, die die Werte `15` und `46` haben (Hinweis: Denkt daran, den Zuweisungsoperator `objekt_name <- objekt_wert` zu verwenden)
4. Fügt einen weiteren Code-Chunk unter diesem Text ein
    - Benutze die Funktionen `sum()` oder `mean()`, um die Summe oder den Mittelwert einiger Zahlen zu berechnen

Erinnerung! Überschriften und Code-Anmerkungen

Denkt beim Schreiben von Notizen/bei der Bearbeitung von Übungen im Unterricht daran, informative Überschriften/Unterüberschriften zu erstellen! Auf diese Weise wird das Dokument strukturiert und übersichtlich, wenn ihr-in-der-Zukunft (oder ich) darauf zurückblickt.
Überschriften/Zwischenüberschriften strukturieren das gesamte Dokument. Code-Anmerkungen beschreiben, was bestimmte Teile des Codes bewirken (und warum). Beide beginnen mit einem Hashtag + Leerzeichen (`# `), aber Überschriften stehen außerhalb eines Codeabschnitts, während Codeanmerkungen innerhalb eines Codeabschnitts erscheinen.
Tipp: Klickt auf die Schaltfläche "Outline" oben rechts im Texteditor-Fenster. Was zeigt sie an?

4.2 Code-Chunk-Optionen

- wir können die Ausführung von Code-Chunken steuern
- wir wollen nicht immer unseren Code in einem Bericht wiederholen
    - wir können dies in jedem Code-Chunk mit `#| echo: true` oder `false` steuern
- wir wollen nicht immer unseren Code in einem Bericht ausführen lassen
    - wir können dies in jedem Code-Chunk mit `#| eval: true` oder `false` steuern
- Dies würde wie folgt aussehen:

````
```{r}
#| eval: true
```
````

```
# Addition
4+6
```

```
[1] 10
```

Aufgabe 4.2: `c()`

Beispiel 4.2.

1. Erinnert ihr sich, dass wir letzte Woche die Funktion `c()` (EN: concatenate) gesehen haben, die mehrere Werte kombiniert (z.B. `mean(c(3,4,25))` ergibt den Mittelwert von 3,4 und 25)
2. In einem Code-Stück: Erstellt ein Objekt, das eine Liste von Zahlen enthält (z.B. `Objektname <- c(...)`)
3. Berechnt den Mittelwert dieser Zahlen, indem ihr nur den Objektnamen verwendt.
4. Speichert den Mittelwert dieser Zahlen als ein Objekt
5. Rendert das Dokument und seht sich den Abschnitt mit eurem Code-Chunk an.
    - Ändert nun im Quellcode die Chunk-Einstellungen auf `echo: false` und rendert das Dokument. Was ändert sich?
    - Setzt nun `echo: true`, aber `eval: false`. Rendert das Dokument. Was ändert sich?

4.3 Inline-Code

- ihr könnt Code auch in-line, d.h. in eurem Text, ausführen, indem ihr ihn mit `` `r ` `` umgebt.
    - Wenn ihr euer Dokument rendert, wird der berechnete Wert ausgegeben (nicht der Quellcode).
- es ist klug, dies nur mit kurzen Code-Chunken zu tun (d.h. `mean(c(3,5,7,2,4,556,7,8,3,2,35,4,56,57,` in-line zu schreiben ist unschön, obwohl dies nur in eurem Quellcode erscheinen wird)
    - deshalb ist es eine gute Idee, z.B. lange Listen von Zahlen (d.h. Daten) als Objekte zu speichern
    - und alle Berechnungen usw. in Code-Blöcken auszuführen und alle wichtigen Werte als Objekte zu speichern

````
```{r}
#| results: hide

# Bevölkerung von Berlin
berlin <- 3645000
# Bevölkerung von Deutschland
de <- 8320000

# welcher Anteil der deutschen Bevölkerung lebt in Berlin?
berlin/de

# dies als Objekt speichern
ber_anteil <- berlin/de
```
````

- Nachdem ich diesen Code-Chunk ausgeführt habe, kann ich das Ergebnis auf verschiedene Arten ausgeben:
  - `3645000/8320000`, was 0.438101 ausgeben würde
  - `berlin/de`, was 0.438101 ausgeben würde
  - `ber_anteil`, was 0.438101 ausgeben würde
- aber wir können auch Funktionen verwenden und etwas mehr Mathematik betreiben, um diese Zahl zu einem *Prozentsatz* zu machen
  - Multiplizieren der Zahl mit 100
    - `ber_anteil*100`
  - in die Funktion `round()` einfügen, um 1 Dezimalpunkt anzuzeigen
    - das sollte so aussehen: `43.8`
  - fügt dies in einen Satz ein
    - der Quelletext "Ungefähr `` `r round(ber_anteil*100,1)` ``% der deutschen Bevölkerung lebt in Berlin, der Hauptstadt." wird wie folgt rendert:
    - Ungefähr 43.8% der deutschen Bevölkerung lebt in Berlin, der Hauptstadt.

Aufgabe 4.3: Code-Chunks

Beispiel 4.3.

1. Beginne ein neues Code-Chunk
2. googeln und als R-Objekt speichern die Bevölkerung von:
   - Toronto, Kanada
   - Kanada
   - der Provinz Ontario

- der Provinz Prince Edward Island

3. Ermittelt den Anteil der kanadischen Bevölkerung, der in Toronto lebt
4. Ermittelt den Anteil der Bevölkerung von Ontario, der in Toronto lebt
5. Ermittelt den Anteil der kanadischen Bevölkerung, der in Ontario lebt
6. Ermittelt den Anteil der kanadischen Bevölkerung, der in Prince Edward Island lebt
7. Erstellt ein Objekt, das die Bevölkerungen von Ontario und Prince Edward Island kombiniert (d.h. "verkettet") (nennt es z.B. "on_pei"). Verwendet dieses Objekt, ermittelt den Anteil der kanadischen Bevölkerung, der in Prince Edward Island oder Ontario lebt
8. Gebt diese Ergebnisse in einem Text an (als Prozentsatz, gerundet auf eine Dezimalstelle)

5 Ausgabeformate

- es gibt mehrere Ausgabeformate, die wahrscheinlich nützlichsten sind:
  - `html` (default)
  - `pdf`
  - `revealjs` (Folien)
  - `docx`

Aufgabe 5.1: Ausgabeformate

Beispiel 5.1.

1. Ersetzt `html` in der YAML durch `revealjs`. Rendert das Dokument.
   - Schaut euch den Ordner für die Notizen dieser Woche an. Welche Dateien seht?
2. Setzt nun `format` auf `pdf`. Rendert das Dokument.
   - Läuft es?
   - Versuche, `pdf` durch den Buchstaben `l` zu ersetzen. R schlägt eine Vervollständigung vor, welche ist es? Wähle sie aus und rendere das Dokument.
3. Setzt das Format wieder auf `html`. Rendert das Dokument.
4. Geht zurück zu eurem Ordner mit den Notizen dieser Woche. Welche Dateien seht?
   - Ist die Ausgabe von `revealjs` dort?

Heutige Ziele

Heute haben wir...

- gelernt, was dynamische Berichte sind
- unser eigenes Quarto-Dokument erstellt
- gelernt, wie man ein Quarto-Dokument bearbeitet
- gelernt, wie man Code in ein Quarto-Dokument einfügt
- ein Quarto-Dokument in verschiedenen Formaten wiedergebt

5.1 Extra: Reproduzierbarkeit in Quarto

- die Paketversionen mit `sessionInfo()` ausgeben
  - wenn ich ein neues Dokument beginne, ist eines der ersten Dinge, die ich tue, eine Kopfzeile `# Session Info` am unteren Ende hinzuzufügen, mit dem folgenden:

```
sessionInfo()
```

::: callout-tip ### **?@exm-sessionInfo**: Session Info ::: {#exm-sessionInfo .custom}

- fügt eine "Session Info" Abschnitt am Ende des Dokuments hin :::

Session Info

Hergestellt mit R version 4.2.3 (2023-03-15) (Shortstop Beagle) und RStudioversion 2023.3.0.386 (Cherry Blossom).

```
sessionInfo()
```

```
R version 4.2.3 (2023-03-15)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.2.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
```

```
attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
 [1] compiler_4.2.3  here_1.0.1      fastmap_1.1.1   rprojroot_2.0.3
 [5] cli_3.6.1       tools_4.2.3     htmltools_0.5.5 rstudioapi_0.14
 [9] yaml_2.3.7      rmarkdown_2.21  knitr_1.42      jsonlite_1.8.4
[13] xfun_0.38       digest_0.6.31   rlang_1.1.0     png_0.1-8
[17] evaluate_0.20
```

Literaturverzeichnis

Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills* (Version 2.0). Zenodo. https://doi.org/10.5281/zenodo.6365078

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (o. J.). *R for Data Science* (2. Aufl.). https://r4ds.hadley.nz/