# Einführung in Quarto

**Dynamische und reproduzierbare Berichte mit R**

Daniela Palleschi

Mi. den 11.01.2023

## 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. (2023) (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

## Wiederholung

Letzte Woche haben wir…

## Set-up

Aufgabe 0.1: Ordner für Woche 3

**Beispiel 0.1.**

1. Fügen Sie einen Unterordner mit dem Namen `03-quarto` in `Notes` hinzu
2. Gehen Sie zu Moodle und speichern den Materialordner für '03 - Einführung in Quarto' in Ihrem `moodle` Ordner
3. Öffnen Sie das Dokument `_blatt.html` auf Ihren Computer
    - Sehen Sie das Dokument an; Sie können oben rechts auf verschiedene Schaltflächen klicken. Probieren Sie es.

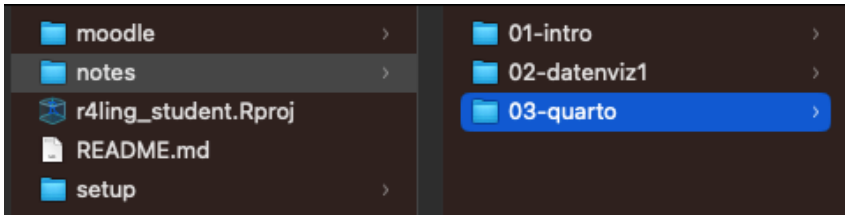


Abbildung 1: Notes folder structure

# 1 Quarto

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

## 1.1 Dynamische Berichte

- diejenigen, die Text, Code, Codeausgabe enthalten
- Quarto bietet ein "unified authoring framework" für Data Science, das Ihren Text, Ihren Code und eure Code-Ausgabe einschließt (Wickham et al., 2023, 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 Ihren 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.

## 1.2 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 1.1: RStudio version

**Beispiel 1.1.**

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

### 1.3 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)

### 1.4 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

## 2 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 2.1: erste Quarto

**Beispiel 2.1.**

1. Erstellen Sie in Ihrem R-Projekt-Ordner, in dem ihr eure Kursunterlagen/Notizen aufbewahren, einen neuen Ordner für Woche 3
2. `File > New Document > Quarto Document`
   - Geben Sie ihm einen Titel wie "Quarto - Woche 3"
   - Deaktivieren Sie die Option "open with Visual Editor".
3. Schauen das neue Skript an, um mehr über Quarto zu erfahren.
4. Klicken Sie auf die Schaltfläche "Render" am oberen Rand des Dokuments
   - Speichern Sie das Dokument in dem Ordner für Woche 3, den Sie gerade erstellt haben.
   - Was geschiehen? Vergleichen die Ausgabe mit dem Quellcode des Dokuments.
5. Gehen Sie zurück zu Ihrem neuen Ordner `02-quarto`
   - Was hat sich geändert?

## 2.1 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 ```` ``` ````

## 2.2 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 Ihrem 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 2.2: YAML

**Beispiel 2.2.**

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

```yaml
---
title: "Quarto - Woche 3"
author: "Vorname Nachname"
format:
  html:
    toc: true
---
```

5. Rendern nun das Dokument. Sehen Sie Ihre Änderungen?

## 2.3 Strukturierung eures Dokuments

- wir können unser Dokument strukturieren mit
  - `# Überschriften`
  - `## Zwischenüberschriften`
  - `### Unter-Zwischenüberschriften`, usw.

```yaml
---
title: "Quarto - Woche 3"
author: "Vorname Nachname"
format:
  html:
    toc: true
```

```
---

# 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:

- gehen zu Ihren Globalen Einstellungen (Werkzeuge > Globale Einstellungen)
- unter `Code` (linke Spalte) `> Display` (Tab), markieren das Kästchen `> Show whitespace character`

Aufgabe 2.3: Überschriften

**Beispiel 2.3.**

1. Kopiern den obigen Code (Überschriften und Unterüberschriften) und ersetzen den Text in der Quarto-Vorlage.
2. Ersetzen die erste Überschrift durch den Titel `Quarto`
    - Schreiben einen Text, der Quarto beschreibt, unter die Überschrift
3. Schreiben eine Unterüberschrift namens `YAML`
    - Schreiben einen Text, der die YAML-Struktur beschreibt, die wir besprochen haben
4. Erstellen eine Unterüberschrift mit dem Namen `Quarto-Struktur`.
    - Schreiben einige Notizen darüber, wie wir ein Quarto-Dokument strukturieren können (z.B. durch das Erstellen von Überschriften)

5. Finden Sie in RStudio die Schaltfläche `Outline` oben links im `.qmd` Text Editor Fenster
    - Was sehent Sie, wenn Sie darauf klicken?

## 2.4 Textformatierung

- zum Formatieren von Text müssen wir die Markdown-Syntax verwenden

| Format | Markdown | Ausgabe |
|---|---|---|
| Kursivschrift | `Dieser Text ist *kursiv*` | Dieser Text ist *kursiv* |
| Fett | `Dieser Text ist **fett**` | Dieser Text ist **fett** |
| Subskription | `Dieser Text ist ~tiefgestellt~` | Dieser Text ist$_{\text{tiefgestellt}}$ |
| Hochgestelt | `Dieser Text ist ^hochgestellt^` | Dieser Text ist $^{\text{hochgestellt}}$ |

## 2.5 Aufzählungen

- wir können 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
```

- 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 2.4: Aufzählungen

**Beispiel 2.4.**

1. Fügen Ihrem `.qmd` Dokumententext eine Textformatierung hinzu.
2. Fügen eine Aufzählungsliste hinzu
3. Fügen eine nummerierte Liste hinzu
4. Rendern Sie das Dokument. Hat es geklappt?

# 3 Codierung in Quarto

- Der große Vorteil von dynamischen Berichten ist die Integration von Text und Code
- Vorletzte 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)

## 3.1 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` verwendt, 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 3.1: Code-Chunks

**Beispiel 3.1.**

1. Füge einen Code Chunk zu deiner `.qmd` Datei hinzu
    - Füge einige mathematische Operationen ein (Addition, Subtraktion, etc)
    - Fügt informative Anmerkungen zu Ihrem Code hinzu (z.B. `# Addition`)
2. Füge einen Text unter deinem Code-Chunk hinzu, der beschreibt, was der obige Code erreicht hat.
3. Rendern Sie das Dokument. Hat es geklappt?

Erinnerung! Überschriften und Code-Anmerkungen

Denken Sie 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: Klicken Sie auf die Schaltfläche “Outline” oben rechts im Texteditor-Fenster. Was zeigt sie an?

## 3.2 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 3.2: `c()`

**Beispiel 3.2.**

1. Erinnern Sie 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: Erstellen sie ein Objekt, das eine Liste von Zahlen enthält (z.B. `Objektname <- c(...)`)
3. Berechnen Sie den Mittelwert dieser Zahlen, indem Sie nur den Objektnamen verwendt.
4. Speichern Sie den Mittelwert dieser Zahlen als ein Objekt
5. Rendern Sie das Dokument und seht sich den Abschnitt mit Ihrem Code-Chunk an.
    - Ändern Sie nun im Quellcode die Chunk-Einstellungen auf `echo: false` und rendern das Dokument. Was ändert sich?
    - Setzen nun `echo: true`, aber `eval: false`. Rendern das Dokument. Was ändert sich?

# 4 Plots in Quarto

- Ein großer Vorteil der gerenderten Quarto-Dokumente besteht darin, dass wir unsere Abbildungen zusammen mit den Textbeschreibungen anzeigen können
- Lassen Sie uns versuchen, eine Handlung von letzter Woche in unserem neuen Quarto-Dokument zu reproduzieren

## 4.1 Set-up

- unsere Pakete in einen Codechunk laden: `tidyverse` und `languageR`

```
```{r}
# Pakete laden
```
```

```
library(tidyverse)
library(languageR)
library(ggthemes)
```

- unsere Daten in einen separaten Codechunk laden (am besten ist es, einen einzigen Codechunk für einen einzigen Zweck zu verwenden)

```
```{r}
# Daten laden
df_lexdec <- lexdec
```
```

## 4.2 Plots in Quarto

- Erstellen Sie jetzt einfach einen neuen Codechunk, der einen Code von letzter Woche enthält
- wir speichern es als Objekt mit dem Namen `fig_lexdec_hist`:

```
## histogram of reaction times by native language
  ggplot(data = df_lexdec) +
  aes(x = exp(RT), fill = NativeLanguage) + ## set aesthetics
  geom_histogram(position = "identity", alpha = 0.3) +
  scale_fill_colorblind() + ## make fill colorblind friendly
  theme_minimal() ## set plot theme
```

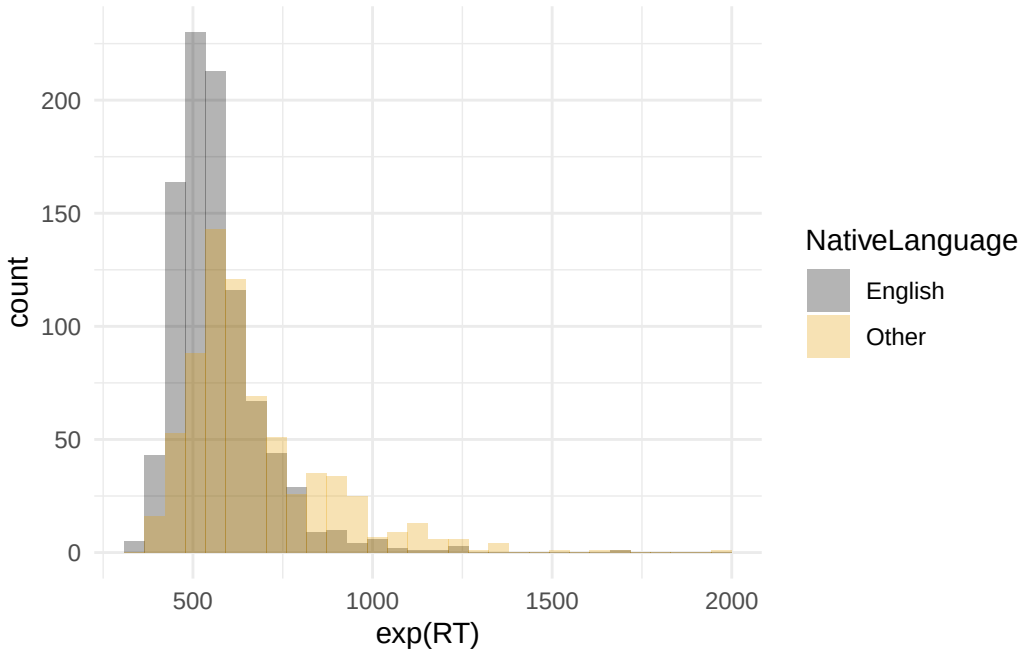


Abbildung 2: Histogram of reactiontimes per native language from lexdec

### 4.3 Plots drucken

- Erinnern Sie sich an die letzte Woche: Wenn Sie einen Plot benennen, wird er nur gedruckt, wenn Sie den Namen des Objekts eingeben
- wenn Sie den Plot nicht als Objekt speichern, wird er gedruckt, wenn Sie den Code ausführen, der den Plot erzeugt
- Wenn Sie den Plot als Objekt speichern, wird er nicht gedruckt, wenn Sie den Code ausführen.
  - In diesem Fall müssen Sie den Objektnamen ausführen, um zu sehen, was unter diesem Namen gespeichert ist
  - Dies gilt für alle Arten von Objekten, nicht nur für Diagramme!

Aufgabe 4.1: Plots in Quarto

**Beispiel 4.1.**

1. Einen neuen Codeabschnitt erstellen und das Balkendiagramm von letzter Woche erzeugen, aber als Objekt speichern
2. In einem separaten Codechunk nur den Objektnamen dieses Diagramms angeben
3. Rendern Sie das Dokument, um zu sehen, wo die Abbildung gedruckt wurde.

```r
fig_lexdec_l1 <-
  ggplot(data = df_lexdec) +
  aes(x = NativeLanguage, fill = NativeLanguage) +
  # add the geom:
  geom_bar() +
  scale_fill_colorblind() + # add colourblind colours
  theme_minimal()
```

```r
fig_lexdec_l1
```

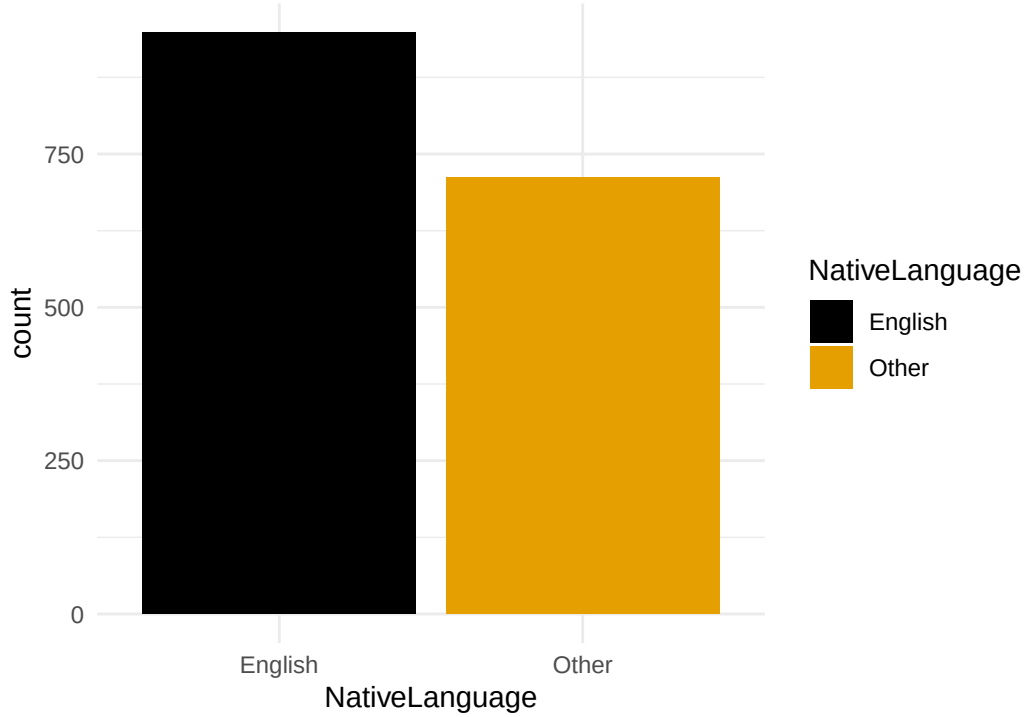


Abbildung 3: Barplot of observations per native language

## 5 Ausgabeformate

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

## 5.1 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 4: 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. (2023))

**i** 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)

Aufgabe 5.1: Ausgabeformate

**Beispiel 5.1.**

1. Ersetzt `html` in der YAML durch `revealjs`. Rendert das Dokument.
    - Schauen Sie 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 Ihrem 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()
```

Aufgabe 5.1: Session Info

**Beispiel 5.1.**

- fügt eine “Session Info” Abschnitt am Ende des Dokuments hin

## Session Info

Hergestellt mit R version 4.3.0 (2023-04-21) (Already Tomorrow) und RStudioversion 2023.3.0.386 (Cherry Blossom).

```
sessionInfo()
```

```
R version 4.3.0 (2023-04-21)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.2.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  L

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

time zone: Europe/Berlin
tzcode source: internal

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

other attached packages:
 [1] ggthemes_4.2.4  languageR_1.5.0 lubridate_1.9.2 forcats_1.0.0
 [5] stringr_1.5.0   dplyr_1.1.3     purrr_1.0.2     readr_2.1.4
 [9] tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.3   tidyverse_2.0.0
[13] magick_2.7.4

loaded via a namespace (and not attached):
 [1] gt_0.9.0         utf8_1.2.3       generics_0.1.3   xml2_1.3.4
 [5] stringi_1.7.12   hms_1.1.3        digest_0.6.33    magrittr_2.0.3
 [9] evaluate_0.21    grid_4.3.0       timechange_0.2.0 fastmap_1.1.1
[13] rprojroot_2.0.3  jsonlite_1.8.7   fansi_1.0.4      scales_1.2.1
[17] cli_3.6.1        rlang_1.1.1      munsell_0.5.0    commonmark_1.9.0
[21] withr_2.5.0      yaml_2.3.7       tools_4.3.0      tzdb_0.4.0
[25] colorspace_2.1-0 here_1.0.1       png_0.1-8        vctrs_0.6.3
[29] R6_2.5.1         lifecycle_1.0.3  pkgconfig_2.0.3  pillar_1.9.0
[33] gtable_0.3.4     glue_1.6.2       Rcpp_1.0.11      xfun_0.39
[37] tidyselect_1.2.0 rstudioapi_0.14  knitr_1.44       farver_2.1.1
[41] htmltools_0.5.5  labeling_0.4.3   rmarkdown_2.22   compiler_4.3.0
[45] markdown_1.7
```

## Literaturverzeichnis


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

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2. Aufl.).