

# **Angewandte Datenverarbeitung und Visualisierung (WiSe23/24)**

**WiSe23/24**

Daniela Palleschi

2023-10-16

# Inhaltsverzeichnis

<b>Kursübersicht</b>	<b>3</b>
Kursbeschreibung auf AGNES . . . . .	3
Ziele des Kurses . . . . .	3
Ressourcen . . . . .	4
<b>I. Kursübersicht</b>	<b>5</b>
<b>Syllabus</b>	<b>6</b>
<b>Erforderliche Software</b>	<b>7</b>
<b>R und RStudio</b>	<b>8</b>
Pakete . . . . .	8
RStudio Globale Optionen (optional) . . . . .	9
<b>tinyTex (optional)</b>	<b>10</b>
<b>II. Grundlagen</b>	<b>11</b>
<b>1. Einführung in R und RStudio</b>	<b>12</b>
Heutige Ziele . . . . .	12
Weitere Lektüre . . . . .	12
1.1. Vorbereitung . . . . .	12
1.2. RProjekt . . . . .	13
1.3. R in RStudio . . . . .	14
1.4. Reproduzierbarkeit . . . . .	17
1.5. Rechnen in R . . . . .	21
1.6. Vektoren . . . . .	24
1.7. Endergebnis . . . . .	25
1.8. Session Info . . . . .	26
1.9. Nächste Woche . . . . .	26
<b>2. Datenvisualisierung 1</b>	<b>29</b>
Heutige Ziele . . . . .	29

2.1. Datenrahmen . . . . .	31
2.2. Lexical Decision Task (LDT) . . . . .	32
2.3. <code>lexdec</code> Datensatz . . . . .	34
2.4. Erstellen von Plots mit <code>ggplot2</code> . . . . .	36
2.5. Entscheidung für ein Geom . . . . .	49
2.6. Exercises . . . . .	49
. . . . .	49
Session Info . . . . .	50
<b>3. Dynamic reports with Quarto</b>	<b>53</b>
Lernziele . . . . .	53
Lesungen . . . . .	53
Wiederholung . . . . .	53
Set-up . . . . .	55
3.1. Quarto . . . . .	56
3.2. Unsere erstes Quarto-Dokument . . . . .	57
3.3. Codierung in Quarto . . . . .	62
3.4. Plots in Quarto . . . . .	64
3.5. Ausgabeformate . . . . .	67
3.6. Extra: Reproduzierbarkeit in Quarto . . . . .	69
<b>4. Data Wrangling 1: Transformation</b>	<b>72</b>
Wiederholung . . . . .	72
Heutige Ziele . . . . .	72
4.1. Voraussetzungen . . . . .	73
4.2. Data Wrangling . . . . .	74
4.3. Zeilen . . . . .	76
4.4. Spalten . . . . .	81
4.5. <code>dplyr</code> und <code>ggplot2</code> . . . . .	87
Aufgaben . . . . .	88
Session Info . . . . .	89
<b>5. Datenvizualisierung 2</b>	<b>92</b>
Set-up . . . . .	93
5.1. Datenvizualisierung . . . . .	95
5.2. Visualisierung von Beziehungen . . . . .	96
5.3. Bearbeitete Daten . . . . .	103
5.4. Quarto Code Chunk Einstellungen . . . . .	105
5.5. Plots speichern . . . . .	106
5.6. Übungen . . . . .	108
<b>6. Bericht 1</b>	<b>111</b>
6.1. Einrichtung . . . . .	111

6.2. Data wrangling . . . . .	112
6.3. Datenvizualisierung . . . . .	113
6.4. Interpretation . . . . .	114
<b>III. Nächste Stufe</b>	<b>115</b>
<b>7. Einlesen von Daten</b>	<b>116</b>
7.1. Einrichtung . . . . .	117
7.2. CSV: Komma getrennter Wert . . . . .	119
7.3. Tabelle zu csv . . . . .	120
7.4. Das <code>readr</code> -Paket . . . . .	123
7.5. Das <code>here</code> -Paket . . . . .	123
7.6. Arbeit mit Daten . . . . .	124
7.7. Andere Dateitypen und Begrenzungszeichen . . . . .	129
7.8. Übungen . . . . .	129
Session Info . . . . .	131
<b>8. Deskriptive Statistik</b>	<b>133</b>
Lernziele . . . . .	133
Lesungen . . . . .	133
8.1. Einrichten . . . . .	133
8.2. Deskriptive Statistik . . . . .	134
8.3. Zusammenfassende Statistiken mit R . . . . .	145
8.4. Gruppierung von Variablen . . . . .	148
8.5. Anscombes Quartett . . . . .	150
8.6. Hausaufgaben . . . . .	153
Session Info . . . . .	154
<b>9. Data Wrangling 2</b>	<b>157</b>
Lernziele . . . . .	157
Lesungen . . . . .	157
9.1. Wiederholung . . . . .	157
9.2. Einrichtung . . . . .	158
9.3. ‘Tidy’ Arbeitsablauf . . . . .	158
9.4. ‘Tidy’ Daten . . . . .	159
9.5. Datenbereinigung . . . . .	163
9.6. Verlängern von Daten: <code>df_eng</code> . . . . .	164
9.7. Verbreiterung der Daten: <code>df_eng</code> . . . . .	167
9.8. Hausaufgaben . . . . .	169
Session Info . . . . .	170

<b>10. Datenvisualisierung 3</b>	<b>173</b>
Lernziele . . . . .	173
Ressourcen . . . . .	173
Einrichten . . . . .	173
10.1. Rückblick: Visualisierung von Verteilungen . . . . .	174
10.2. Darstellung von zusammenfassenden Statistiken . . . . .	176
10.3. Visualisierung des Mittelwerts . . . . .	182
Hausaufgabe . . . . .	195
Session Info . . . . .	196
<b>Bericht 2</b>	<b>199</b>
<b>IV. Fortgeschrittene Themen</b>	<b>200</b>
<b>Literaturverzeichnis</b>	<b>201</b>

# Kursübersicht

Dies ist die Webseite der Lehrveranstaltung “Angewandte Datenverarbeitung und Visualisierung” an der Humboldt-Universität zu Berlin, Institut der deutschen Sprache und Linguistik für das Wintersemester 2023/24. Wenn Sie für den Kurs eingeschrieben sind, finden Sie alle relevanten Materialien auf dem Kurs Moodle [hier](#) (Moodle-Schlüssel wird in der Vorlesung bereitgestellt).

Jedes Kapitel entspricht einer Vorlesung von einer Woche. Vorerst werden die Materialien auf dieser Website im Bullet-Point-Format erscheinen und genau denselben Inhalt wie die Kursfolien enthalten. Ich plane, die Aufzählungspunkte später in Prosa umzuwandeln und die einzelnen Themen zu vertiefen.

## Kursbeschreibung auf AGNES

Dies ist ein Einführungskurs in das Denken, Arbeiten und Kommunizieren mit / über sprachliche Daten. Der Kurs fokussiert sich auf praktische Anwendungen und die Vermittlung übertragbarer Fähigkeiten. In RStudio machen sich die Teilnehmenden sich mit der Programmiersprache R vertraut und entwickeln Fähigkeiten zur Erstellung und Vermittlung zusammenfassender Statistiken für den akademischen und beruflichen Kontext. Die Teilnehmenden lernen, Rohdaten zu laden und zu manipulieren, Tabellen mit deskriptiven Statistiken zu erstellen und die Daten angemessen visuell darzustellen. Am Ende des Kurses werden die Teilnehmenden ein besseres Verständnis dafür haben, wie man mit Daten umgeht und die Fähigkeiten besitzen, Ergebnisse klar zu kommunizieren. Studierende, die keinen eigenen Laptop zum Unterricht mitbringen können, setzen sich bitte so früh wie möglich mit der Dozentin in Verbindung, damit ein alternativer Laptop organisiert werden kann. Der Kurs wird auf Deutsch gehalten.

## Ziele des Kurses

Das Hauptziel dieses Kurses ist es, die Kenntnisse und Fähigkeiten zu entwickeln, die für die Durchführung einer “Explorativen Datenanalyse (EDA)” erforderlich sind. EDA ist kein formaler Prozess mit spezifischen Regeln, sondern vielmehr “a state of mind” (Wickham et al., 2023, Kapitel 11). Das Wissen, das für die Durchführung einer EDA erforderlich ist, besteht

einfach darin, die Daten zu verstehen und ihre Struktur zu erforschen, um ein Verständnis für ihre Verteilung und Muster zu bekommen. Die für die Durchführung einer EDA erforderlichen Fähigkeiten sind spezifisch für die zur Durchführung der EDA verwendete Sprache, in unserem Fall R.

## Ressourcen

Die meisten unserer Materialien basieren auf dem Buch “R for Data Science” von Hadley Wickham (2. Auflage), das Sie [hier](#) vollständig online einsehen können. Wo es möglich war, habe ich die in diesem Buch verwendeten Daten durch linguistische Datensätze ersetzt, damit Sie sich ein Bild davon machen können, wie Linguisten R verwenden könnten.

Einige andere Ressourcen, die wir von Zeit zu Zeit verwenden werden oder die Sie vielleicht selbst erkunden möchten, sind das E-book *Data visualisation using R, for researchers who don't use R* (Nordmann et al., 2022) und das Lehrbuch *Statistics for Linguists: An Introduction Using R* by Bodo Winter [Winter (2019); PDF erhältlich über das Grimm Zentrum].

# **Teil I.**

# **Kursübersicht**

# Syllabus

Die vorgeschlagene Lektüre erleichtert die Arbeit mit dem Material für jede Woche. Die Lektüre umfasst Kapitel oder Abschnitte aus Nordmann et al. (2022) (web tutorial), Wickham et al. (2023) (E-book), and Winter (2019) (PDF verfügbar über die Grimm-Bibliothek).

Woche	Datum	Thema	Vorbereitung
1	18.10.2023	Einführung in R und RStudio	R4DS - Ch 1 (Introduction)
2	25.10.2023	Data Viz 1: Verteilungen	R4DS - Ch 2 (Data visualisation)
3	01.11.2023	Dynamische Berichte mit Quarto	R4DS - Ch 29 (Quarto)
4	08.11.2023	Wrangling 1: Umwandlung von Daten	R4DS - Ch 4 (Data transformation)
5	15.11.2023	Data Viz 2: Visualisierung von Beziehungen	R4DS - Ch 5 (Workflow)
6	22.11.2023	Bericht 1	
7	29.11.2023	Daten einlesen	R4DS - Ch 8 (Data import)
8	06.12.2023	Wrangling 2: Tidying data	R4DS - Ch 6 (Data tidy)
9	13.12.2023	Deskriptive Statistik	Winter (2019) - Ch 3 (Descriptive statistics)
10	20.12.2023	Data Viz 3: Visualisierung von Zusammenfassungen	R4DS - Ch 2 (Data visualisation)
Vorlesungsfrei	27.12.2023	NA	
Vorlesungsfrei	03.01.2024	NA	
11	10.01.2024	Bericht 2	
12	17.01.2024	Einführung in Base R	R4DS - Ch 28 (A field guide to R)
13	24.01.2024	Regular expressions	R4DS - Ch 16 (Regular expressions)
14	31.01.2024	Data Viz 4: Kommunikation	R4DS - Ch 12 (Communication)
15	07.02.2024	Bericht 3	
16	14.02.2024	Offene Sitzung: Q&A	

# Erforderliche Software

Dieses Dokument beschreibt die Schritte, die erforderlich sind, um unseren reproduzierbaren Arbeitsablauf für den Kurs ‘Angewandte Datenanalyse und -visualisierung’ einzurichten. `?@sec-R` gibt einen Überblick über die Installation von R, RStudio und der erforderlichen Pakete. Diese Schritte sind erforderlich. `?@sec-tinytex` beschreibt die Installation von TinyTex, das benötigt wird, um Dokumente im LaTeX-Stil (z.B. PDFs) in R darzustellen.

# R und RStudio

Um an diesem Kurs teilnehmen zu können, müssen Sie R und RStudio installieren.

R ist eine statistische Programmiersprache, die für statistische Berechnungen und grafische Darstellungen verwendet wird. Am häufigsten wird sie zur Analyse und Visualisierung von Daten verwendet, beides werden wir in diesem Semester tun. RStudio ist eine IDE (integrierte Entwicklungsumgebung) für R und andere Sprachen. RStudio macht die Analyse und Visualisierung von Daten in R viel einfacher (glauben Sie mir, als ich mit R anfing, gab es kein RStudio!).

Sie müssen R herunterladen, bevor Sie RStudio herunterladen können.

1. [R herunterladen](#)
2. [RStudio herunterladen](#)

## Pakete

R-Pakete, die im Comprehensive R Archive Network, allgemein bekannt als CRAN-Repository, verfügbar sind, können einfach mit dem Befehl `install.packages("packageName")` installiert werden. Einige Pakete, die wir brauchen werden, sind:

- `here` Paket (Müller, 2020)
- `tidyverse`-Paketfamilie (Wickham et al., 2019)
  - enthält automatisch Pakete, die wir brauchen, wie `dplyr` und `ggplot2`
- `languageR`-Paket (Baayen & Shafaei-Bajestan, 2019)

Um mehrere Pakete auf einmal herunterzuladen, verwenden Sie die ‘concatenate’-Funktion in `r (c())` innerhalb von `install.packages()`:

```
install.packages(c("here",
                    "tidyverse",
                    "pacman"))
```

## RStudio Globale Optionen (optional)

Hier sind meine bevorzugten globalen Optionen (RStudio > Werkzeuge > Globale Optionen). Ich empfehle dringend, die Einstellungen für “Arbeitsbereich” und “R-Sitzungen” zu befolgen, um die Reproduzierbarkeit zu gewährleisten. Mit den anderen Einstellungen können Sie herumspielen, um herauszufinden, was Ihnen gefällt.

- Allgemein > Grundeinstellungen
  - **Arbeitsbereich** (für reproduzierbare Arbeitsabläufe!!!)
    - \* Deaktivieren Sie das Kontrollkästchen “RData beim Starten in Arbeitsbereich wiederherstellen”.
    - \* Arbeitsbereich beim Beenden in .RData speichern: **Niemals**
  - **R-Sitzungen**
    - \* Deaktivieren Sie das Kontrollkästchen “Zuvor geöffnete Quelldokumente beim Start wiederherstellen”.
- Code > Anzeige
  - Allgemein
    - \* Leerzeichen anzeigen
    - \* Scrollen über das Ende des Dokuments hinaus zulassen
    - \* Ausgewählte Zeile hervorheben
- Erscheinungsbild
  - Editor-Thema: Kobalt

## **tinyTex (optional)**

Im weiteren Verlauf des Kurses werden wir lernen, wie man verschiedene Ausgabeformate, einschließlich PDF, erzeugt. Um PDF-Dokumente mit LaTeX unter der Haube darstellen zu können, müssen wir [tinytex](#) installieren. Es gibt verschiedene Möglichkeiten, dies zu tun:

- Führen Sie folgendes im *Terminal* aus: `quarto install tinytex`
- oder in der Konsole: `tinytex::install_tinytex()`

Sie können diesen Schritt vorerst überspringen, falls Sie Probleme haben.

## **Grundlagen**

# 1. Einführung in R und RStudio

## Heutige Ziele

- R und RStudio installieren
- in der Lage sein, Zusatzpakete zu installieren
- in der Lage sein, Hilfe für Pakete und Funktionen zu erhalten
- in der Lage sein, Objekte in der Konsole zu erstellen

## Weitere Lektüre

- Dieser Vortrag basiert lose auf Kapitel 1 - *Introduction* und Kapitel 3 - *Workflow Basics* von Wickham et al. (2023)
- dieser Kurs folgt mehr oder weniger diesem Buch
- wo möglich, ersetze ich die Datensätze im Buch durch linguistische Datenbeispiele

### 1.1. Vorbereitung

- hoffentlich haben Sie R und RStudio bereits installiert/aktualisiert
  - falls nicht: Versuchen Sie es mit [Posit Cloud](#) für heute [posit.cloud](#)
- Gehen Sie zum [Kurs GitHub](#) und laden Sie eine ZIP-Datei des Repositorys herunter
  - große grüne Schaltfläche ‘<> Code’ > ZIP herunterladen

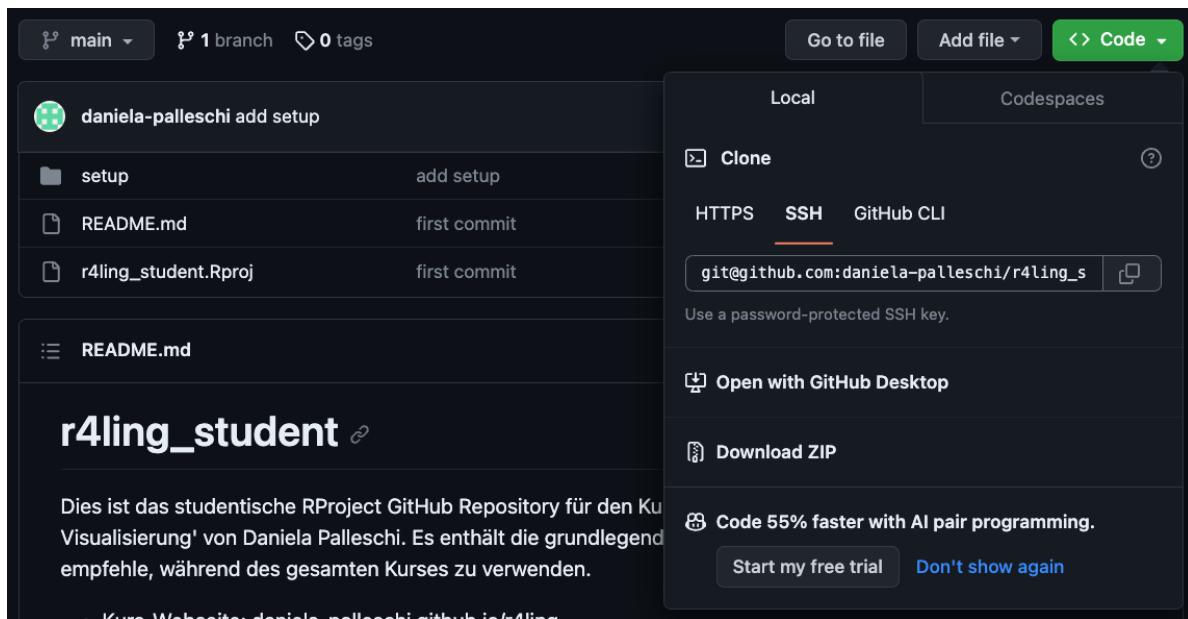


Abbildung 1.1.: Download GitHub repository

## 1.2. RProjekt

- Suchen Sie die ZIP-Datei, die Sie soeben heruntergeladen haben, auf Ihrem Computer und dekomprimieren Sie sie.
- Öffnen Sie den Ordner und navigieren Sie zu r4ling\_student.Rproj, doppelklicken Sie darauf
- Sie sollten nun RStudio sehen, wie in Abbildung 1.2
- Jetzt können wir an unserem ersten Skript arbeiten

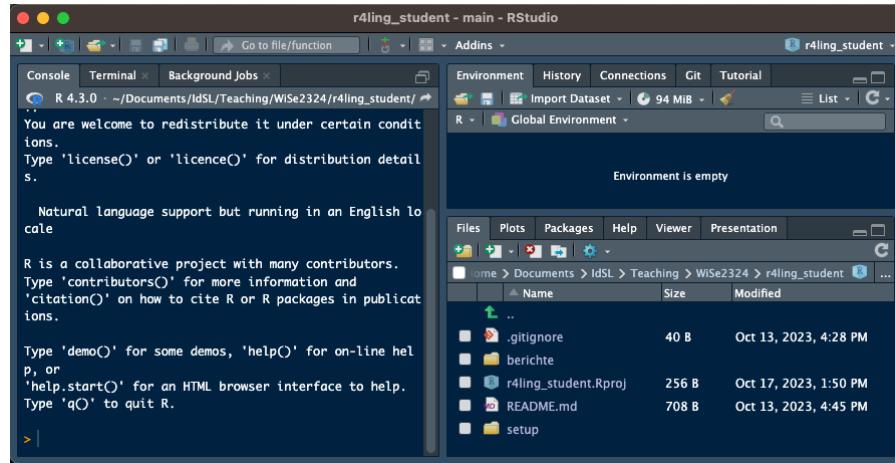


Abbildung 1.2.: Student RProject

### **⚠ Warnung**

#### **Wichtig!!**

Verschieben oder benennen Sie den Ordner `data/` nicht um! Sie müssen denselben Dateipfad zu den Datensätzen haben, um meinen Code in den nächsten Wochen nahtlos verwenden zu können.

## 1.3. R in RStudio

1. Öffnen Sie RStudio *immer* durch einen Doppelklick auf `r4ling_student.Rproj` (für diesen Kurs)
2. klicken Sie auf `File > New File > R Script`
  - sehen Sie nun vier Quadrate (statt 3 in Abbildung 1.2):
    - i. Texteditor - oben Links - wo wir unseren Code schreiben werden
    - ii. R-Konsole (EN: Console) - unten links - wo wir die Ausgabe unseres Codes und Warn-/Fehlermeldungen sehen werden
    - iii. Arbeitsumgebung (EN: Environment) - oben rechts - wo unsere Daten und Objekte nach dem Laden gespeichert werden
    - iv. Dateien und Grafikausgabe - unten links - wo wir unsere Dateien und die von uns erstellten Grafiken sehen oder Hilfe bekommen können

### 1.3.1. Erweiterungspakete

- R hat eine Reihe von nativen Funktionen und Datensätzen, auf die wir zugreifen können
  - ähnlich wie die Standard-Apps, die auf Ihrem Handy vorinstalliert sind
- Jeder kann Zusatzpakete für R erstellen, z.B.,
  - für Datenvisualisierung
  - Datenverarbeitung
- Dies ist ähnlich wie bei Handy-Apps, die von jedem erstellt und auf Ihr Gerät heruntergeladen werden können
  - aber Pakete sind *immer kostenlos*
- Es gibt 2 Schritte, um ein Paket zu verwenden:
  1. Installieren des Pakets (einmalig) mit `install.packages("Paket")`
  2. Laden Sie das Paket (zu Beginn jeder Sitzung) `library(Paket)`

#### 1.3.1.1. Paket-Installation

- erfolgt mit der Funktion `install.packages()`
  - Sie machen dies nur einmal (wie das Herunterladen einer App)
- das Paket `tidyverse` ist sehr hilfreich für Datenverarbeitung und Visualisierung
  - Installieren wir es jetzt

#### Paket-Installation

- installieren Sie die Pakete `tidyverse` und `beesr`

```
install.packages("tidyverse")
install.packages("beesr")
```

##### ! Pakete in der Konsole installieren

Installieren Sie Pakete immer über die Konsole, nicht über ein Skript!  
Sie können auch die Registerkarte “Pakete” in der unteren rechten Box verwenden (Pakete > Installieren)

### 1.3.1.2. `tinytex`

- wir brauchen auch LaTeX und `tinytex` (Xie, 2023), um PDF-Dokumente zu erstellen
- führen Sie diesen Code aus, um `tinytex` zu installieren

```
## run this in the console
install.packages("tinytex")
tinytex::install_tinytex()
```

- Sie müssen auch LaTeX installieren, wenn Sie es noch nicht haben: <https://www.latex-project.org/get/>

### 1.3.2. Laden eines Pakets

- die Funktion `library()` lädt ein Paket in Ihre Umgebung
- dies muss zu Beginn jeder Sitzung geschehen, um auf das entsprechende Paket zugreifen zu können

```
library(beep)
```

#### 1.3.2.1. Verwendung einer Funktion

- Sobald Sie ein Paket geladen haben, können Sie auf dessen Funktionen zugreifen
- Zum Beispiel hat das Paket `beep` eine Funktion `beep()`, probieren wir sie aus

---

#### Listing 1.1 in der Konsole laufen

---

```
beep()
```

---

#### 1.3.2.2. Funktionsargumente

- Argumente enthalten optionale Informationen, die an eine Funktion übergeben werden
  - Die Funktion `beep()` hat das Argument `sound`, das einen numerischen Wert von 1:11 annimmt.
  - Versuchen Sie, den folgenden Code mit anderen Zahlen auszuführen, was passiert?

---

### **Listing 1.2** in der Konsole laufen

---

```
beep(sound = 5)
```

---

### **Funktionsargumente**

**i** ?help

Sie können mehr über eine Funktion (einschließlich ihrer verfügbaren Argumente) herausfinden, indem Sie ihren Namen nach einem Fragezeichen in die Konsole schreiben (z.B. ?beep). Versuchen Sie, ?beep auszuführen. Kannst du auf der Hilfeseite herausfinden, was du anstelle von sound = 5 schreiben kannst, um denselben Ton zu erzeugen?

### **1.3.3. Aufgabe: Paket-Installation**

**💡** Aufgabe

Wir brauchen auch das here-Paket. Installieren Sie dieses.  
Nachdem Sie das Paket installiert haben, führen Sie den Befehl here() aus. Was geschieht?

## **1.4. Reproduzierbarkeit**

- in diesem Kurs werden wir lernen, wie man *reproduzierbare Berichte* erstellt
  - Das bedeutet, dass unser Code später noch einmal ausgeführt werden kann und immer noch die gleichen Ergebnisse liefert
- wenn Ihre Arbeit reproduzierbar ist, können andere Leute (und Sie selbst) Ihre Arbeit verstehen und überprüfen
  - Für Kursaufgaben werden Sie Berichte sowie den Quellcode einreichen, die ich auf meinem Rechner ausführen können sollte

### **1.4.1. RStudio-Einstellungen**

- wir wollen immer mit einem freien Arbeitsbereich in RStudio beginnen, um die Reproduzierbarkeit zu gewährleisten
  - Wir wollen auch niemals unseren Arbeitsbereich für später speichern
  - wir wollen nur unseren Code (und die Ausgabeberichte) speichern
- Gehen Sie zu `Tools > Global Options`
  - Deaktivieren Sie das Kontrollkästchen `Restore .RData into workspace at startup`
  - Setzen Sie `Save workspace to .RData on exit:` to `Never`

### **RStudio-Einstellungen**

RStudio: Tools > Global Options:

- `Restore .RData into workspace at startup`
  - nein
- `Save workspace to .RData on exit:`
  - `Never`

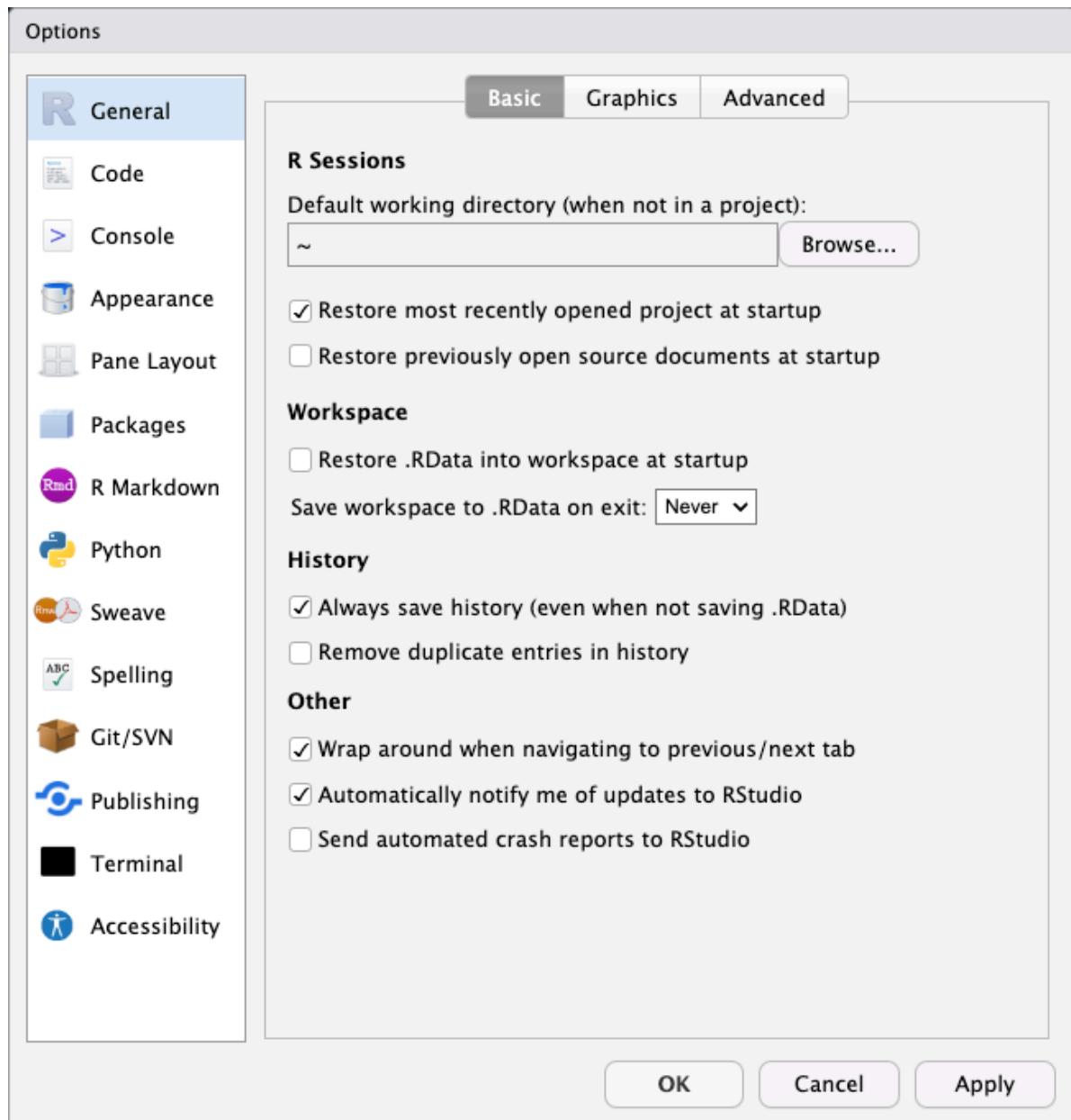


Abbildung 1.3.: Ihre ‘Global Options’ sollten wie folgt aussehen

## RStudio-Einstellungen

- Klicken Sie auf Appearance (linke Spalte)

- Öffnen Sie die Optionen “Editor Theme” und wählen Sie ein Farbschema, das Ihnen gefällt
- Sie können auch die Schriftart/Schriftgröße ändern, wenn Sie dies wünschen

### 1.4.2. Aufgabe: neues R-Skript

 Aufgabe

- in RStudio: File > New File > R Script
  - wenn sich oben links ein neues Fenster öffnet: “Datei > Speichern unter...”.
    - \* speichern Sie es in Ihrem ‘notizen’ Ordner
  - schreiben Sie oben in das Skript: `## Angewandte Datenverarbeitung und Visualisierung - Woche 1 (17.04.2023)`

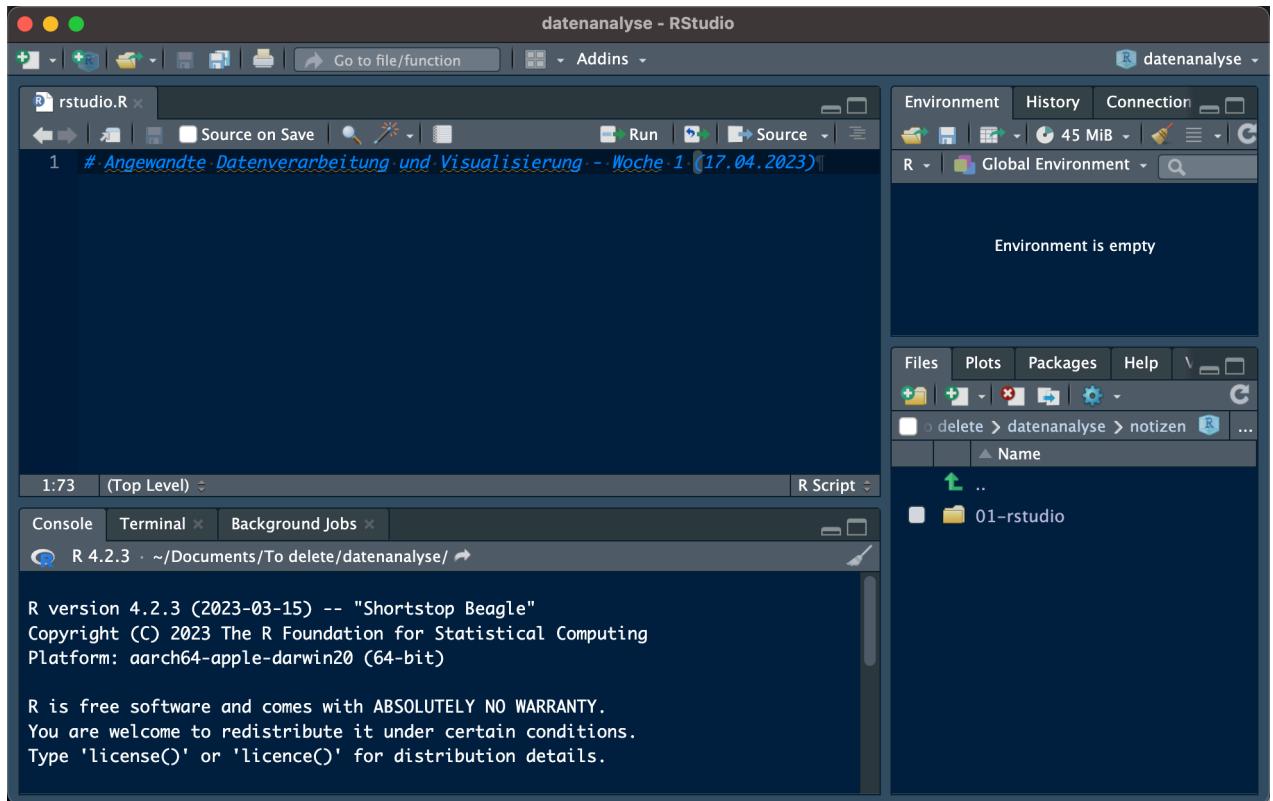


Abbildung 1.4.: Ihre Skript (oben links) sollten so aussehen

## 1.5. Rechnen in R

- können wir Berechnungen in R durchführen
- wir können addieren (+), subtrahieren (-), multiplizieren (\*) und dividieren (/)

### 1.5.1. Aufgabe: Berechnungen

#### 💡 Aufgabe

1. Versuchen Sie, die folgenden Berechnungen in der Konsole auszuführen:

```
# Addition  
16+32
```

```
[1] 48
```

```
# Multiplikation  
16*32
```

```
[1] 512
```

```
# Subtraktion  
16-32
```

```
[1] -16
```

```
# Division  
16/32
```

```
[1] 0.5
```

2. schreiben Sie diese Berechnungen in Ihr Skript, und drücken Sie **Cmd/Strg-Enter**, um sie auszuführen

- Was passiert?

### 1.5.2. Kommentare

- Sie haben vielleicht bemerkt, dass in meinen Code-Blöcken z. B. `# Subtraktion` über dem Code stand
- R ignoriert jeden Text nach `#` (plus ein Leerzeichen)
- also können wir Kommentare nach `#` schreiben

```
# Kommentar zum folgenden Code  
16-32
```

[1] -16

- Wir können auch eine Abschnittsüberschrift erstellen, um unsere R-Skripte zu strukturieren, indem wir vier `#` nach einem Titel hinzufügen
- Die Struktur des Skripts kann dann durch Klicken auf die Schaltfläche “Gliederung” oberhalb des Skriptfensters angezeigt werden

```
# Rechnen mit R #####  
  
# Subtraction  
16-32
```

[1] -16

### 1.5.3. Objekte

- wir können auch Werte als Objekte/Variablen speichern, die in der Arbeitsumgebung gespeichert sind

```
x <- 16  
y <- 32
```

#### i Assignment operator

Das Symbol `<-` ist ein sogenannter *assignment operator*. Es erstellt ein neues Objekt in Ihrer Arbeitsumgebung oder überschreibt ein vorhandenes Objekt mit demselben Namen. Es ist wie ein Pfeil, der sagt: “Nimm das, was rechts steht, und speichere es als den Objektnamen auf der linken Seite”.

#### 1.5.4. Rechnen mit Funktionen

- es gibt auch eingebaute Funktionen für komplexere Berechnungen
- z.B., `mean()` (DE: Durchschnitt), `sum()` (DE: Summe)
- was passiert, wenn wir folgendes ausführen?

```
sum(6,10)
```

```
[1] 16
```

```
6+10
```

```
[1] 16
```

```
mean(6,10)
```

```
[1] 6
```

```
(6+10)/2
```

```
[1] 8
```

#### Rechnen mit Funktionen

- die Funktion `mean()` nimmt nur ein Argument an; alles andere wird ignoriert
  - das Komma in `6,10` listet 2 Argumente auf, also wird alles nach dem Komma ignoriert
- wenn wir mehr als ein Objekt in ein Argument einschließen wollen, müssen wir die “concatenate”-Funktion `c()` verwenden
  - “concatenate” bedeutet zusammenfügen oder kombinieren

```
mean(c(6,10))
```

```
[1] 8
```

## Rechnen mit Funktionen

- Sie können auch benannte Objekte (d.h. die in Ihrer Arbeitsumgebung) verwenden, die einen numerischen Wert haben

💡 Aufgabe: Rechnen mit Funktionen

1. Versuchen Sie, die Funktion `mean()` mit Ihren gespeicherten Variablen (`x` und `y`) als “verkettete” Argumente auszuführen
2. Machen Sie dasselbe mit der Funktion `sum()`. Was passiert, wenn Sie `c()` nicht verwenden?

## 1.6. Vektoren

- Vektoren sind eine Liste von Elementen desselben Typs (z. B. numerisch, Zeichenkette)
- wir können einen Vektor mit der Verkettungsfunktion `c()` erstellen
- Der folgende Code speichert in einem Objekt namens ‘vec’ einen Vektor aus mehreren Zahlen

```
# einen Vektor erstellen  
vec <- c(171, 164, 186, 191)
```

- der folgende Code ruft das Objekt auf, das wir als ‘vec’ gespeichert haben, und gibt seinen Inhalt aus

```
# print vec  
vec
```

```
[1] 171 164 186 191
```

### 1.6.1. Arithmetic mit Vektoren

- Grundlegende Arithmetik auf Vektoren wird auf jedes Element angewendet

```
# add 5 to vec  
vec + 5
```

```
[1] 176 169 191 196
```

- können wir auch Funktionen auf Vektoren anwenden

```
# Summe von vec  
sum(vec)
```

```
[1] 712
```

```
# Mittelwert von vec  
mean(vec)
```

```
[1] 178
```

```
# Quadratwurzel aus vec  
sqrt(vec)
```

```
[1] 13.07670 12.80625 13.63818 13.82027
```

### 1.6.2. Ausgabe: Vektoren

#### 💡 Ausgabe

1. Erstelle einen Vektor namens `vec1`, der die Werte 12, 183, 56, 25 und 18 enthält
2. Erstellen Sie einen Vektor namens `vec2`, der die Werte 8, 5, 1, 6 und 8 enthält
3. Create a vector called `vec3` that contains the values 28, 54, 10, 13, 2, and 81
4. Finde die Summe von `vec1`.
5. Finde die Summe von `vec1` plus `vec2`. Wie unterscheidet sich das Ergebnis von dem, das Sie für `vec1` allein erhalten haben?
6. Was passiert, wenn du versuchst, die Summe von `vec1` und `vec3` zu finden?

## 1.7. Endergebnis

- Speichern Sie Ihr R-Skript (`File > Save`, oder `Cmd/Strg-S`)
- Sie sollten nun einen *RProject-Ordner* für diesen Kurs, der Folgendes enthält:
  - `r4ling_student.RProj`
  - einen Ordner namens `Daten`
  - einen Ordner namens `notes`, der Folgendes enthält + eine `.R`-Datei mit der heutigen Arbeit

- Sie wissen jetzt, wie man
  - einfache Berechnungen in R durchführen
  - Objekte in Ihrer Arbeitsumgebung zu speichern
  - einfache mathematische Berechnungen mit Ihren gespeicherten Objekten durchführen

## 1.8. Session Info

- Um die Reproduzierbarkeit zu verbessern, ist es nützlich, die Version von R, RStudio und die verwendeten Pakete zu verfolgen
  - Zu diesem Zweck können Sie die folgenden Befehle ausführen:

```
## R version
R.version.string
```

[1] "R version 4.3.0 (2023-04-21)"

```
## R version name
R.version$nickname
```

[1] "Already Tomorrow"

```
## RStudio version
RStudio.Version()$version
## RStudio version name
RStudio.Version()$release_name
```

```
## alle Paketeversionen
sessionInfo()
```

## 1.9. Nächste Woche

vor nächster Woche, stellen Sie bitte sicher, dass Sie:

- R und RStudio installiert/aktualisiert haben
- die Pakete tidyverse und here installiert haben

- bitte stellen Sie sicher, dass Sie die Übungen des heutigen Kurses in Ihrem R-Skript durcharbeiten
- (optional) speichern Sie das Skript, und laden Sie es auf Moodle hoch, wenn Sie es auf Ihre 6 Skripte für die Teilnahme-LP anrechnen lassen möchten

## 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; 

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] beepr_1.3    magick_2.7.4

loaded via a namespace (and not attached):
 [1] digest_0.6.33  fastmap_1.1.1   xfun_0.39       magrittr_2.0.3
 [5] glue_1.6.2     stringr_1.5.0   audio_0.1-10    knitr_1.44
 [9] htmltools_0.5.5 png_0.1-8     rmarkdown_2.22  lifecycle_1.0.3
[13] cli_3.6.1      compiler_4.3.0  rprojroot_2.0.3 here_1.0.1
[17] rstudioapi_0.14 tools_4.3.0    evaluate_0.21   Rcpp_1.0.11
[21] yaml_2.3.7     rlang_1.1.1    jsonlite_1.8.7  stringi_1.7.12
```

## Literaturverzeichnis

- Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*.
- Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>
- Davies, R., Locke, S., & D'Agostino McGowan, L. (2022). *datasauRus: Datasets from the Datasaurus Dozen*. <https://CRAN.R-project.org/package=datasauRus>
- Müller, K. (2020). *here: A Simpler Way to Find Your Files*. <https://CRAN.R-project.org/package=here>
- Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills*. Zenodo. <https://doi.org/10.5281/zenodo.6365078>
- Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2022). Data Visualization Using R for Researchers Who Do Not Use R. *Advances in Methods and Practices in Psychological Science*, 5(2), 251524592210746. <https://doi.org/10.1177/25152459221074654>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. <https://doi.org/10.21105/joss.01686>
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2. Aufl.).
- Winter, B. (2019). Statistics for Linguists: An Introduction Using R. In *Statistics for Linguists: An Introduction Using R*. Routledge. <https://doi.org/10.4324/9781315165547>
- Xie, Y. (2023). *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>

## 2. Datenvisualisierung 1

Visualisierung von Verteilungen

### Wiederholung

Letzte Woche haben wir...

- R und RStudio installiert
- unser erstes R-Skript erstellt
- einfache Arithmetik mit Objekten und Vektoren durchgeführt

### Wiederholung

```
x <- c(1,2,3)
y <- sum(1,2,3)
```

- Was enthalten die Vektoren x und y?
- Das Objekt x enthält 1, 2, 3
- Das Objekt y enthält ‘ 6 “

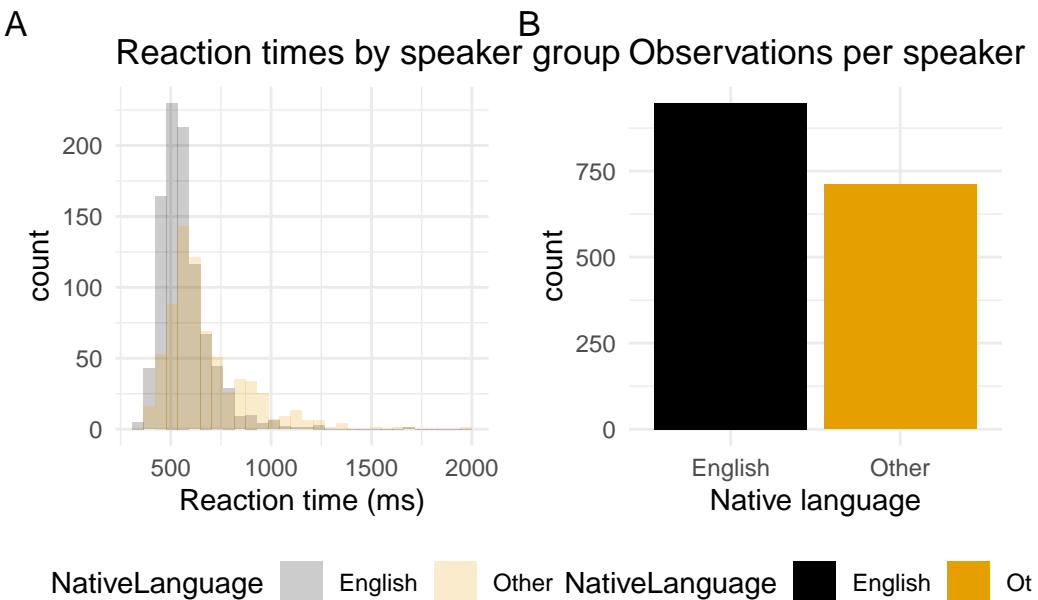
### Heutige Ziele

Heute werden wir lernen...

- was Datenframes sind
- den Unterschied zwischen kategorialen und kontinuierlichen Daten
- wie man Diagramme mit `ggplot` erstellt
- die richtige Darstellung für unsere Daten auszuwählen

## Endg ltiges Ziel

- Unser heutiges Ziel ist es, die Daten wie folgt zu visualisieren
    - Das Diagramm zeigt die Verteilung (**Anzahl**) der Reaktionszeiten und der Muttersprache der Teilnehmer



# Lust auf mehr?

- Kapitel 2 (Datenvisualisierung) in Wickham et al. (2023), bis zum Abschnitt 2.4
  - Kapitel 3 (Datenvisualisierung) in Nordmann & DeBruine (2022)

## Vorbereitung

## In Ihrem RProject-Ordner...

- erstellen Sie einen neuen Ordner mit dem Namen **moodle**
    - Laden Sie die Moodle-Materialien von heute herunter und speichern Sie sie dort
  - Erstellen Sie einen neuen Ordner in **notes** mit dem Namen **02-datenviz1**
  - öffne ein neues .R Skript
    - speichere es in dem neuen Ordner

### 2.0.0.1. Pakete

- Pakete laden (und installieren)

- tidyverse
- languageR
- ggthemes
- patchwork

```
## in the CONSOLE: install packages if needed
install.packages("tidyverse")
install.packages("languageR")
install.packages("ggthemes") ## for customising our plots
install.packages("patchwork") ## plot layouts

## Pakete laden
library(tidyverse)
library(languageR)
library(ggthemes)
library(patchwork)
```

## 2.1. Datenrahmen

- Datenrahmen sind eine Sammlung von Variablen, wobei
  - jede Variable eine Spalte ist
  - jede Zeile eine einzelne Beobachtung/ein einzelner Datenpunkt ist
  - jede Zelle in einer Zeile verknüpft ist
- Datenrahmen sind genau wie Tabellenkalkulationen, aber rechteckig
- Verschiedene Wörter für Datenrahmen:
  - Datenrahmen
  - Datensatz
  - Tibble (im tidyverse)

### 2.1.1. Sprechen über Datensätze

- eine **Variable**: eine Menge, Qualität oder Eigenschaft, die man messen kann
- ein **Wert**: der Zustand einer Variablen, wenn man sie misst

- eine **Beobachtung**: eine Reihe von Messungen, die unter ähnlichen Bedingungen durchgeführt werden
  - enthält mehrere Werte, die jeweils mit einer Variablen verbunden sind
  - eine Beobachtung für eine einzelne Variable wird manchmal als *Datenpunkt* bezeichnet
- **Tabellendaten** sind eine Reihe von Werten, die jeweils mit einer Variablen und einer Beobachtung verbunden sind
  - Tabellarische Daten sind “tidy”, wenn jeder Wert in einer eigenen *Zelle*, jede Variable in einer eigenen Spalte und jede Beobachtung in einer eigenen Zeile steht

### 2.1.2. Kategoriale und kontinuierliche Variablen

- Wie wir die Verteilung einer Variablen darstellen, hängt davon ab, welche Art von Daten sie repräsentiert: *kategorisch* oder *numerisch*
- Eine Variable ist *kategorisch*, wenn sie eine kleine Menge von Werten annehmen kann, die sich in Gruppen zusammenfassen lassen
  - z. B. alt/jung, klein/groß, grammatisch/ungrammatikalisch, L1/L2-Sprecher
- eine Variable ist *numerisch* (d. h. quantitativ), wenn sie eine große Bandbreite an numerischen Werten annehmen kann
  - und es sinnvoll wäre, zu addieren, zu subtrahieren, den Mittelwert zu berechnen usw.
  - kann *kontinuierlich* sein (Dezimalpunkte sind sinnvoll, z. B. 1,5 cm)
  - oder *diskret* (Dezimalpunkte sind *nicht* sinnvoll, z. B. 1,5 Kinder sind nicht sinnvoll)
- wir erstellen verschiedene Diagramme, je nachdem, welche Art von Variablen wir visualisieren wollen

## 2.2. Lexical Decision Task (LDT)

- unser erster Datensatz enthält Daten aus einer lexikalischen Entscheidungsaufgabe
- Bei der LDT drücken die Teilnehmer eine Taste, um anzugeben, ob ein Wort ein echtes Wort oder ein Pseudowort ist.

## Lexical-Decision Task

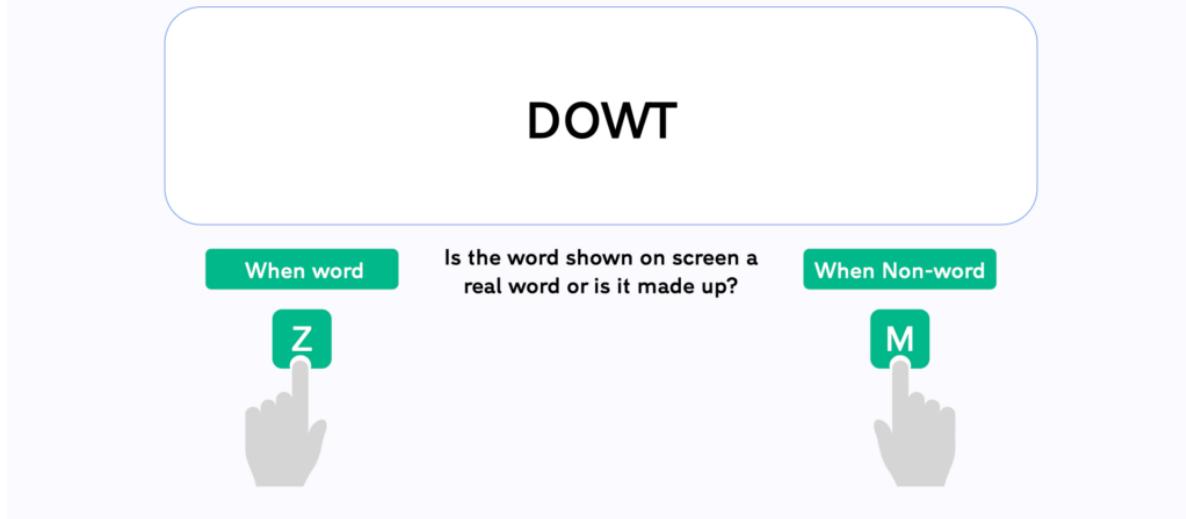


Abbildung 2.1.: Source: [https://www.testable.org/wp-content/uploads/2022/11/Lexical\\_decision\\_task-1024x576.png](https://www.testable.org/wp-content/uploads/2022/11/Lexical_decision_task-1024x576.png)

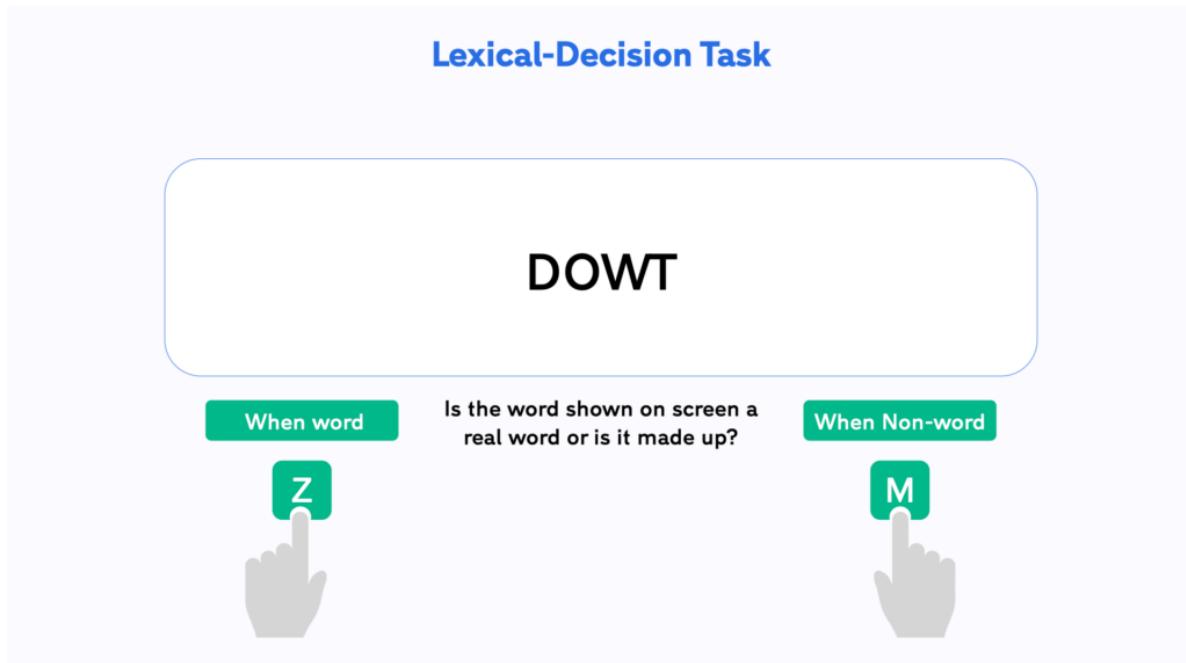


Tabelle 2.1.: Datenwörterbuch für `df_lexdec`: Lexikalische Entscheidungslatenzen, die von 21 Probanden für 79 konkrete englische Substantive erhoben wurden, mit Variablen, die mit dem Subjekt oder dem Wort verknüpft sind.

Variable	Beschreibung
Subject	ein Faktor für die Probanden
RT	ein numerischer Vektor für die Reaktionszeit in Millisekunden
Trial	ein numerischer Vektor für den Rang des Versuchs in der Versuchsliste
Sex	ein Faktor mit den Ausprägungen F (weiblich) und M (männlich)
NativeLanguage	ein Faktor mit den Niveaus English und Other, der zwischen englischen Muttersprachlern v.

### 2.2.1. LDT-Variablen

- Die üblichen Variablen, die in einem Experiment zur lexikalischen Entscheidungsaufgabe erhoben werden, sind:
  - Reaktionszeit
  - Genauigkeit (richtig/falsch)
  - Wortkategorie (z. B. real/pseudo, Nomen/Verb)
  - Worthäufigkeit
- Zusätzliche Variablen, die erhoben werden könnten, sind:
  - demografische Daten der Teilnehmer (z. B. Alter, L1/L2, Geschlecht)

## 2.3. lexdec Datensatz

- `languageR` ist ein Begleitpaket für das Lehrbuch Baayen (2008)
  - enthält linguistische Datensätze, z.B. `lexdec`.
- der `lexdec`-Datensatz enthält Daten für eine lexikalische Entscheidungsaufgabe im Englischen
  - wir werden mit Variablen wie Reaktionszeiten und Genauigkeit arbeiten

### 2.3.1. lexdec-Variablen

- eine Liste einiger der Variablen ist in Tabelle 2.1 enthalten

### 2.3.2. LDT-Forschungsfragen

- bevor wir ein Experiment durchführen, haben wir Forschungsfragen, die wir mit den Daten beantworten wollen
  - Wir werden uns heute mit der folgenden Frage beschäftigen:
    - \* Unterscheiden sich die Reaktionszeiten zwischen Muttersprachlern und Nicht-Muttersprachlern?

### 2.3.3. Laden der Daten

- unsere Daten sind in dem Paket `languagR` verfügbar, das wir bereits geladen haben
  - um die Daten zu drucken, geben Sie einfach den Namen des Datensatzes ein und führen Sie ihn aus
- Unten sehen wir nur ein paar Variablen, aber Sie sollten mehr in Ihrer Konsole sehen

```
lexdec
```

	Subject	RT	Trial	Sex	NativeLanguage	Correct	PrevType	PrevCorrect
1	A1	6.340359	23	F	English	correct	word	correct
2	A1	6.308098	27	F	English	correct	nonword	correct
3	A1	6.349139	29	F	English	correct	nonword	correct
4	A1	6.186209	30	F	English	correct	word	correct
5	A1	6.025866	32	F	English	correct	nonword	correct
6	A1	6.180017	33	F	English	correct	word	correct

- Wie viele Variablen haben wir? Beobachtungen?

#### 2.3.3.1. Daten als Objekt speichern

- Um die Daten in unserer Umgebung zu speichern, müssen wir ihnen einen Namen zuweisen
  - Nennen wir es `df_lexdec`, was soviel bedeutet wie “Datenrahmen lexikalische Entscheidung”.

```
df_lexdec <- lexdec
```

- jetzt sehen wir es in unserem Environment
  - Doppelklicken Sie darauf, um es im Editorfenster zu sehen.

### 2.3.4. Relevante Variablen

- Zu den Variablen, die wir haben, gehören:
  1. **Subjekt**: Teilnehmer-ID
  2. **RT**: protokollierte Reaktionszeiten
  3. **NativeLanguage**: die Muttersprache des Teilnehmers
  4. **Word**: welches Wort präsentierte wurde
  5. **Class**: ob das Wort ein Tier oder eine Pflanze war

 Aufgabe 9.2: `?lexdec`

#### Beispiel 2.1.

Um herauszufinden, wofür die anderen Variablen stehen, führen Sie `?lexdec` in der Konsole aus.

## 2.4. Erstellen von Plots mit ggplot2

- das **tidyverse** ist eine Sammlung von Paketen, die das Aufräumen und die Visualisierung von Daten erleichtern
  - wenn wir **tidyverse** laden, wird diese Sammlung von Paketen automatisch geladen
- das **ggplot2**-Paket ist ein **tidyverse**-Paket, das Plots in Schichten aufbaut

### ggplot2 Schichten

#### 2.4.1. Ebene 1: leere Leinwand

- die erste Ebene mit der Funktion `ggplot()` ist wie eine leere Leinwand

```
ggplot(data = df_lexdec)
```

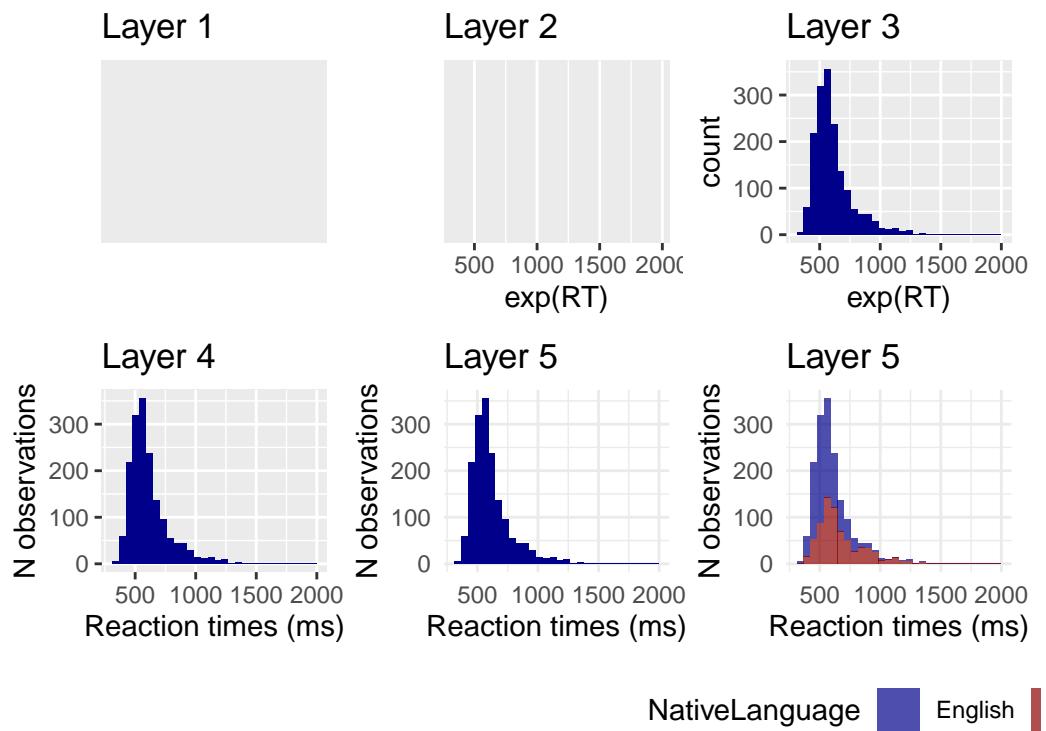
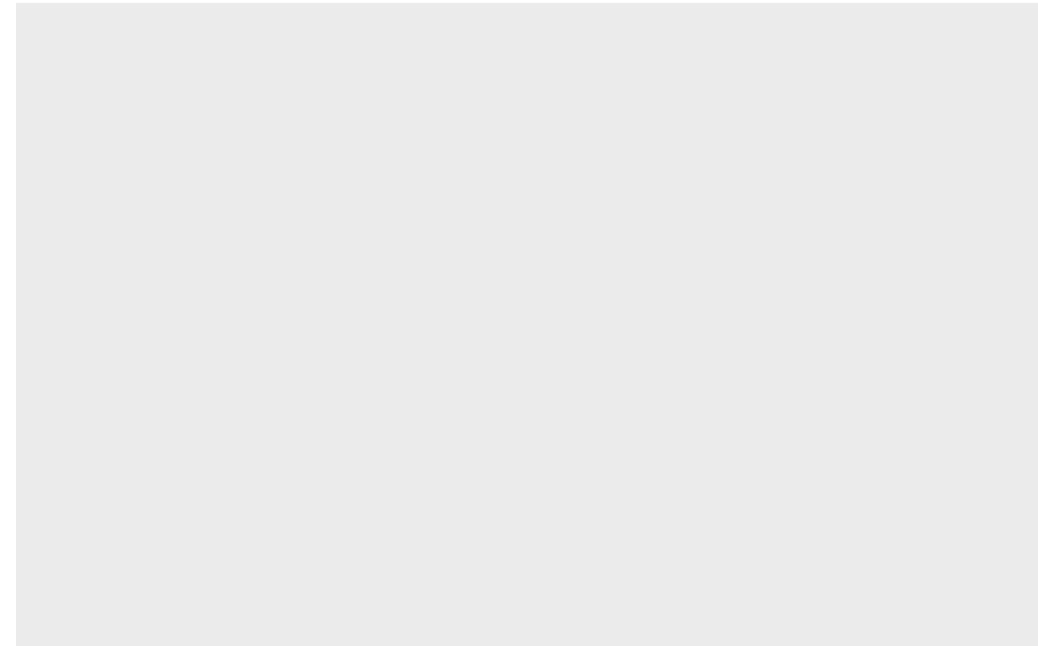


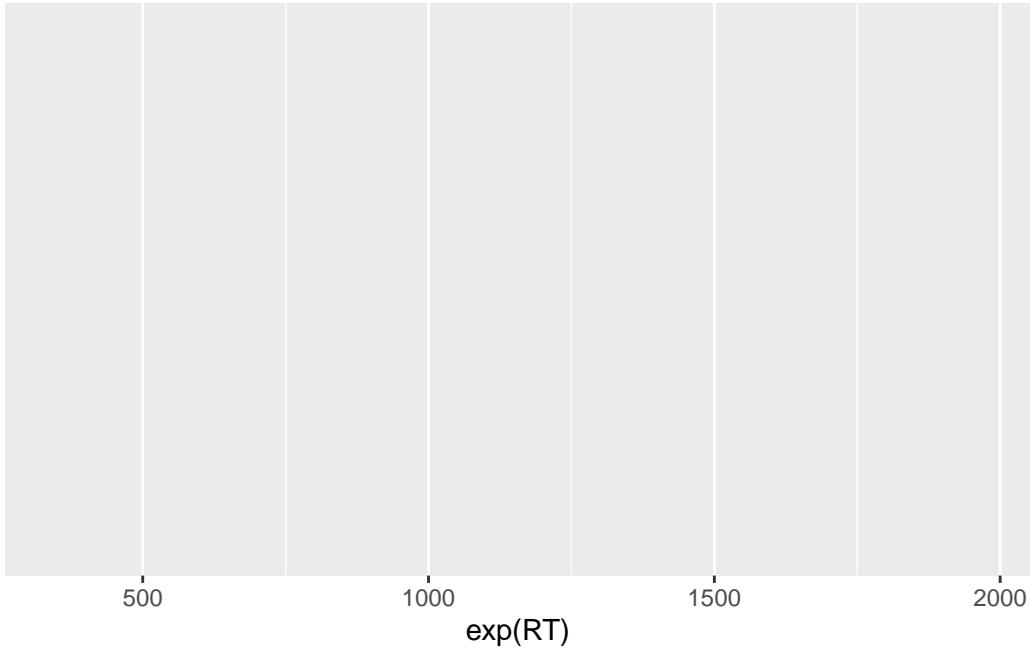
Abbildung 2.2.: Example of layers in a ggplot figure



#### 2.4.2. Ebene 2: Ästhetik der Darstellung

- als nächstes teilen wir `ggplot()` mit, wie unsere Variablen visuell dargestellt werden sollen
  - Wir fügen das “+” am Ende unserer Codezeile ein und verwenden in einer neuen Codezeile die Funktion “`aes()`”, um unsere *Ästhetik* zu definieren.
- Unsere erste Ästhetik bildet die Reaktionszeiten (RT) auf der x-Achse ab (der untere Teil der Grafik)
  - wir wickeln die protokolierte RT in die Funktion `exp()` ein, um RTs in Millisekunden zu erhalten (aus Gründen, die wir nicht diskutieren werden)

```
ggplot(data = df_lexdec) +  
  aes(x = exp(RT))
```



Aufgabe 2.2: Ästhetische Kartierung

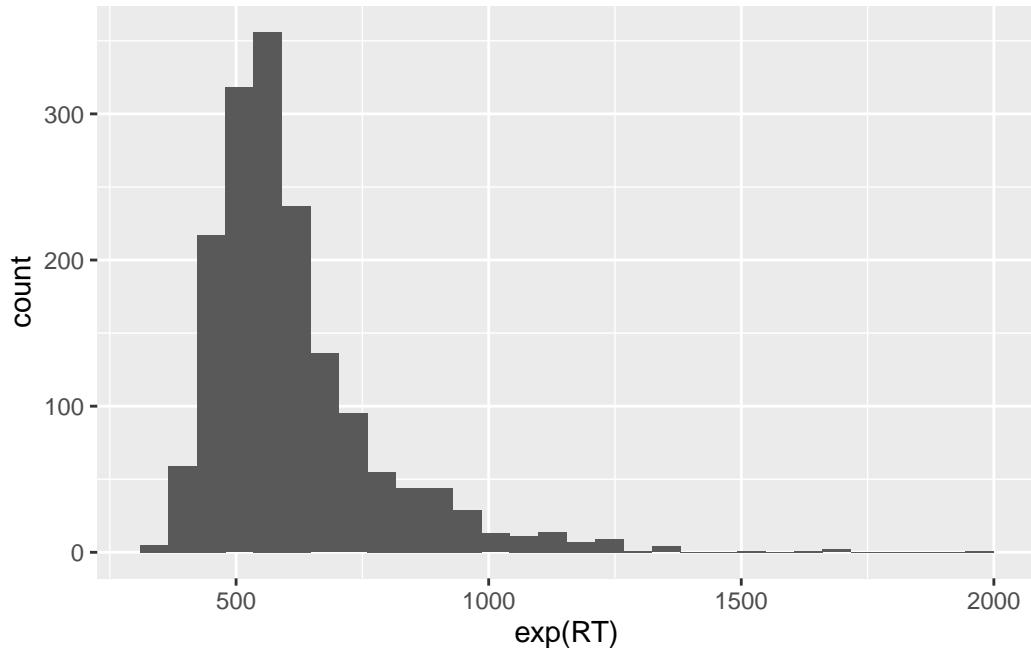
**Beispiel 2.2.**

Add the x-axis aesthetic.

#### 2.4.3. Schicht 3: Hinzufügen von Beobachtungen

- wir sehen keine Beobachtungen (d.h. die Balken) in der Grafik, warum nicht?
  - wir haben `ggplot()` nicht gesagt, wie sie dargestellt werden sollen
- wir müssen ein **Geom** definieren: das *geometrische Objekt*, das ein Diagramm verwendet, um Daten darzustellen
  - in `ggplot2` beginnen die Geom-Funktionen mit `geom_`
  - wir beschreiben Diagramme oft in Bezug auf die Arten von Geomen, die sie verwenden, z.B. verwenden Balkendiagramme Balkengeome (`geom_bar()`), Liniendiagramme Liniengeome (`geom_line()`), Punktdiagramme ein Punktgeom (`geom_point()`), usw.
- Erzeugen wir unser Histogramm mit dem Geom `geom_histogram()`

```
ggplot(data = df_lexdec) +  
  aes(x = exp(RT)) +  
  geom_histogram()
```



### i Hinweis

Wir erhielten die folgende Meldung, als wir `geom_point()` einschlossen:

`stat_bin()` mit `bins = 30`. Wählen Sie einen besseren Wert mit `binwidth`.

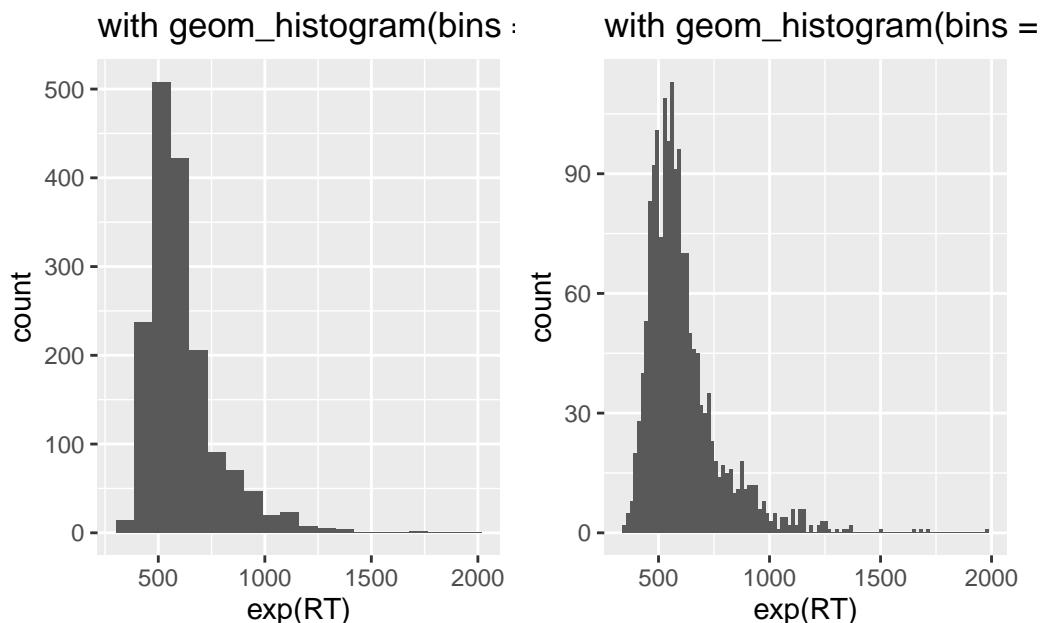
Dies sagt uns nur etwas über die Breite unserer Balken: jeder Balken repräsentiert einen Bereich möglicher Reaktionszeitwerte + `bins = 30` bedeutet einfach, dass es 30 Balken gibt, wir können dies ändern und mehr oder weniger Balken haben, indem wir z.B. `bins = 20` oder `bins = 100` in `geom_histogram()` einfügen

```
ggplot(  
  data = df_lexdec,  
  mapping = aes(x = exp(RT))  
) +  
  labs(title = "with geom_histogram(bins = 20)") +  
  geom_histogram(bins = 20) +
```

```

ggplot(
  data = df_lexdec,
  mapping = aes(x = exp(RT))
) +
  labs(title = "with geom_histogram(bins = 100)") +
  geom_histogram(bins = 100)

```



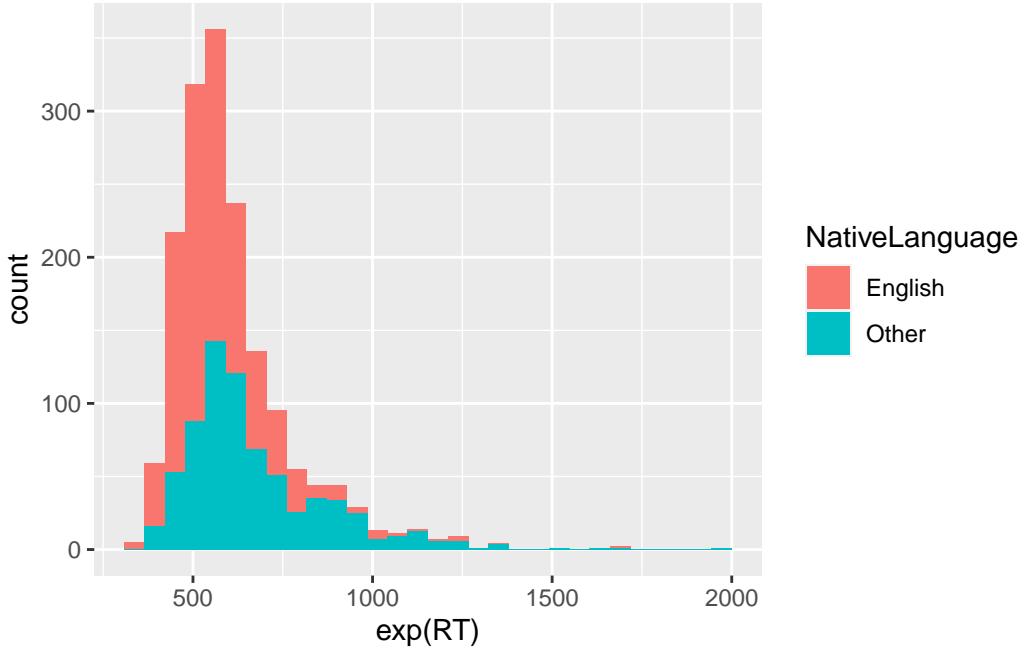
#### 2.4.4. Hinzufügen von Ästhetik

- Es ist nützlich, die Verteilung der Reaktionszeiten im Allgemeinen zu sehen.
  - aber wir wollen normalerweise Gruppen vergleichen
  - z. B. Unterschiede zwischen Muttersprachlern und Nicht-Muttersprachlern oder zwischen verschiedenen Wortarten
- Wir haben auch die Muttersprache als Variable, wie könnten wir diese in unserem Diagramm visualisieren?

```

ggplot(
  data = df_lexdec,
  aes(x = exp(RT), fill = NativeLanguage)
) +
  geom_histogram()

```



- wir sehen die roten und die blauen Balken, aber ist das blaue Histogramm über das rote geschichtet?
  - oder sind die roten Balken über den blauen Balken gestapelt?
- Es ist letzteres
  - stellen wir es so ein, dass das blaue Histogramm über dem roten liegt

```
ggplot(
  data = df_lexdec,
  aes(x = exp(RT))
) +
  labs(title = "No grouping") +
  geom_histogram() +
  theme_minimal()

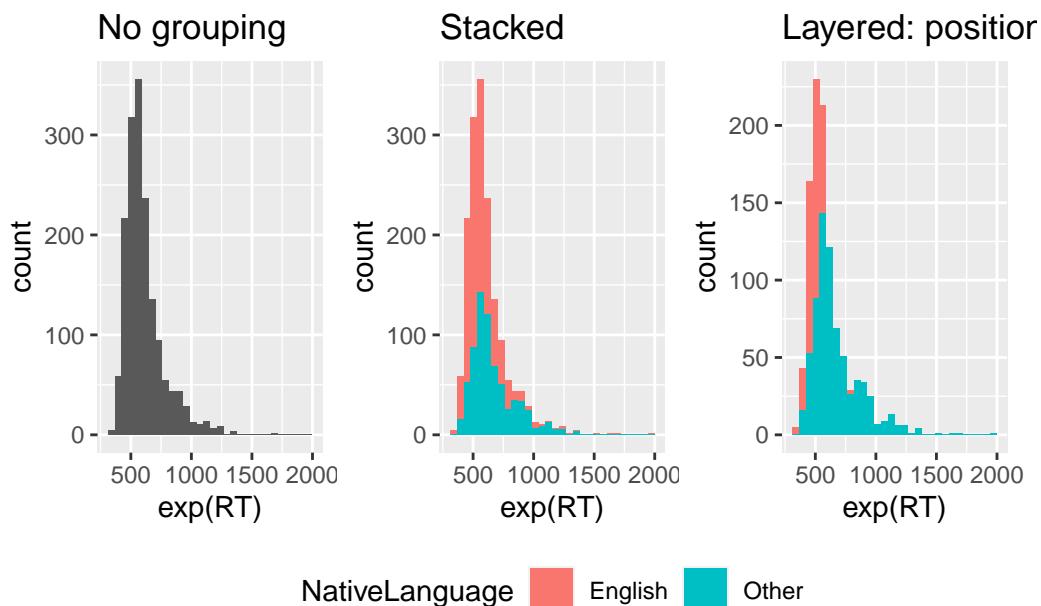
ggplot(
  data = df_lexdec,
  aes(x = exp(RT), fill = NativeLanguage)
) +
  labs(title = "Stacked") +
  geom_histogram() +
  theme_minimal()

ggplot(
  data = df_lexdec,
```

```

aes(x = exp(RT), fill = NativeLanguage)
) +
labs(title = "Layered: position = \"identity\"") +
geom_histogram(position = "identity") +
plot_layout(guides = "collect") & theme(legend.position = 'bottom')

```



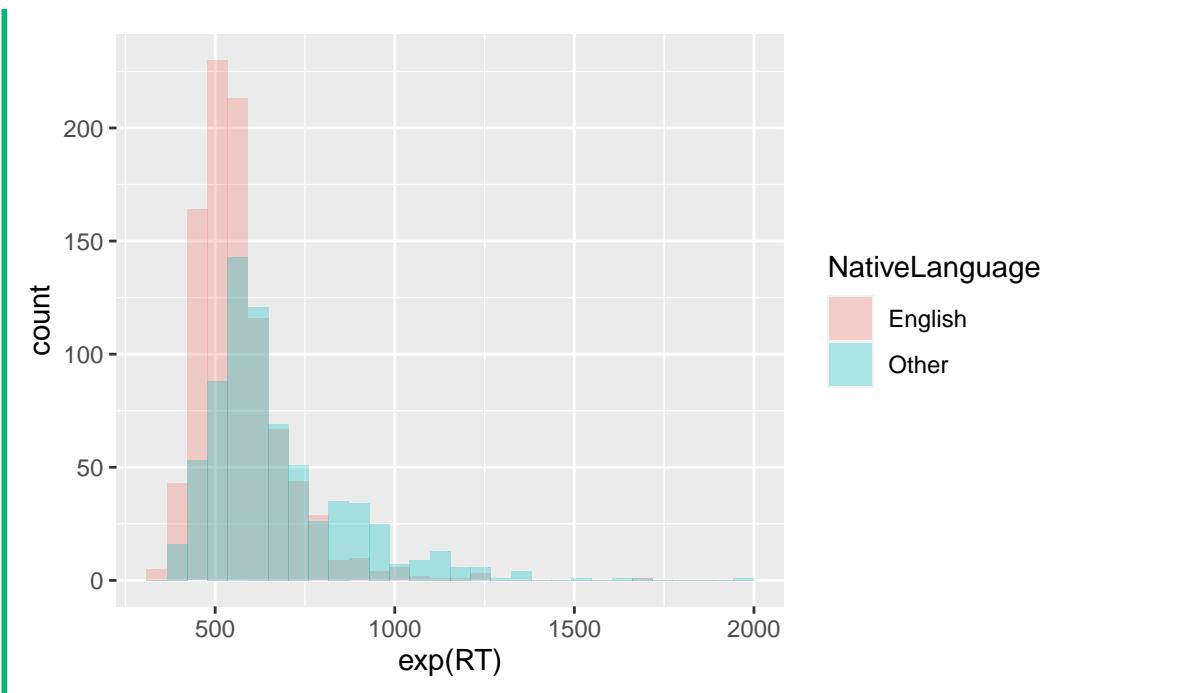
#### 2.4.5. Globale und lokale Ästhetik

- in unserer endgültigen Darstellung ist die Farbe der Histogramme leicht transparent
  - Wir können dies steuern, indem wir das Argument `alpha = 0.3` zu `geom_histogram()` hinzufügen.
  - `alpha` kann jeden anderen Wert zwischen 0 und 1 annehmen.

Aufgabe 2.3: Transparenz

##### Beispiel 2.3.

Spielen Sie mit der Transparenz des Histogramms geom. Wählen Sie den von Ihnen bevorzugten Alpha-Wert. Die Ausgabe sollte in etwa so aussehen:



#### 2.4.6. Anpassen unseres Plots

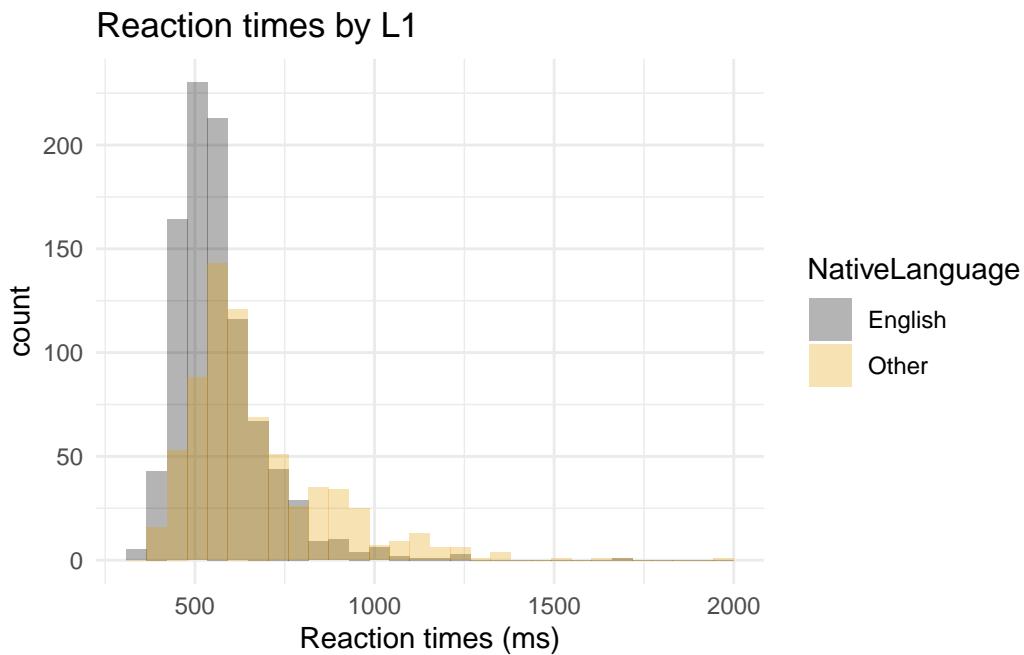
- wir können unsere Achsen- und Legendenbeschriftungen verbessern und auch Titel hinzufügen, indem wir die Funktion `labs()` verwenden
- Wir können auch die Funktion `scale_fill_colorblind()` aus dem Paket `ggthemes` verwenden.
  - dies erzeugt farbenblind-sichere Farben
- Wir werden auch die Funktion `theme_minimal()` aus dem Paket `ggplot2` verwenden; was bewirkt diese Funktion?
- Versuchen Sie, Ihrem Diagramm Folgendes hinzuzufügen
  - Ändern Sie die Beschriftungen entsprechend
  - und fügen Sie dem Code sinnvolle Kommentare mit `#` hinzu

```
labs(title = "Plot title",
     x = "x-axis label",
     y = "y-axis label") +
  scale_fill_colourblind() +
  theme_minimal()
```

## 2.4.7. Kommentar

- Der Code und die Darstellung sollten in etwa so aussehen:

```
## histogram of reaction times by native language
ggplot(data = df_lexdec) +
  aes(x = exp(RT), fill = NativeLanguage) + ## set aesthetics
  labs(title = "Reaction times by L1",
       x = "Reaction times (ms)") +
  geom_histogram(position = "identity", alpha = 0.3) +
  scale_fill_colorblind() + ## make fill colorblind friendly
  theme_minimal() ## set plot theme
```



## 2.4.8. Speichern von Plots

- Wir können Diagramme in unserer Umgebung speichern, genau wie wir Zahlen und Daten als Objekte speichern können.
  - Sie können Objekte beliebig benennen
  - aber es ist ratsam, den Namen sinnvoll zu gestalten (z.B. *nicht fig1* oder *xyz*)
- Nennen wir diese Grafik `fig_lexdec_rt`, für “figure lexical decision task reaction times”.

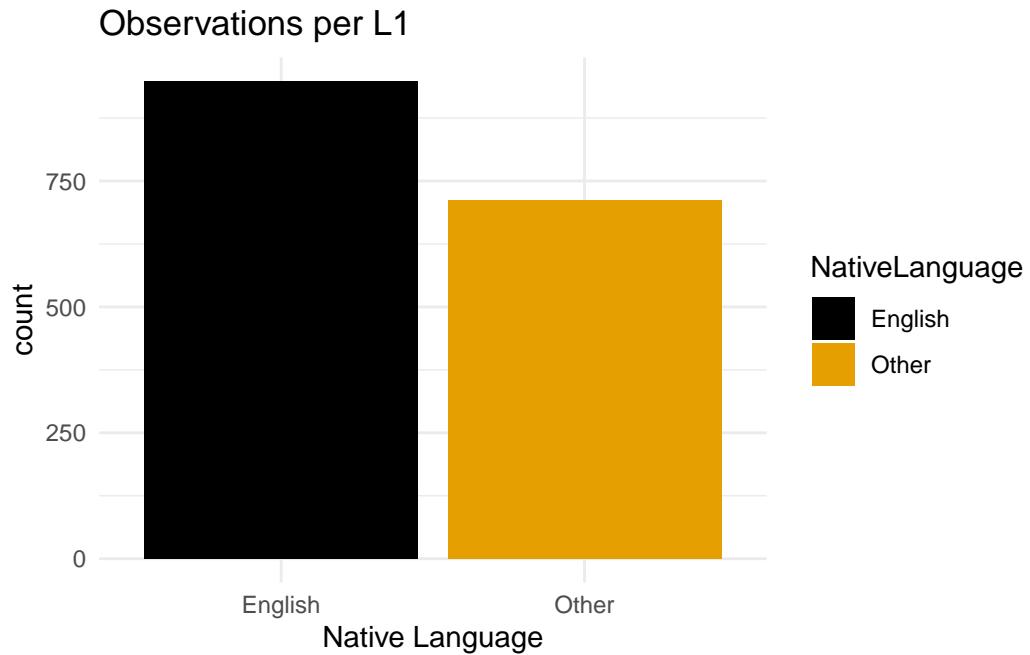
 Aufgabe 2.4: Figur als Objekt speichern

#### Beispiel 2.4.

1. Speichern Sie unsere endgültige Darstellung als Objekt mit dem Namen `fig_lexdec_rt`.

#### 2.4.9. Balkendiagramme

1. Kopieren Sie den Code für Ihr Histogramm
2. Nehmen Sie die folgenden Änderungen vor, um unser Balkendiagramm darzustellen
  - Entfernen Sie die Namenszuweisung (`fig_lexdec_rt`)
  - auf der x-Achse wollen wir `NativeLanguage`
  - Ersetzen Sie `geom_histogram()` durch `geom_bar()`
    - Entfernen Sie die Argumente für das Histogramm (kein `position` oder `alpha`)
  - ändern Sie die Beschriftungen entsprechend
3. Speichern Sie das Diagramm als Objekt mit einem aussagekräftigen Namen (z.B. `fig_lexdec_11`)
  - sollte das Diagramm in etwa so aussehen:

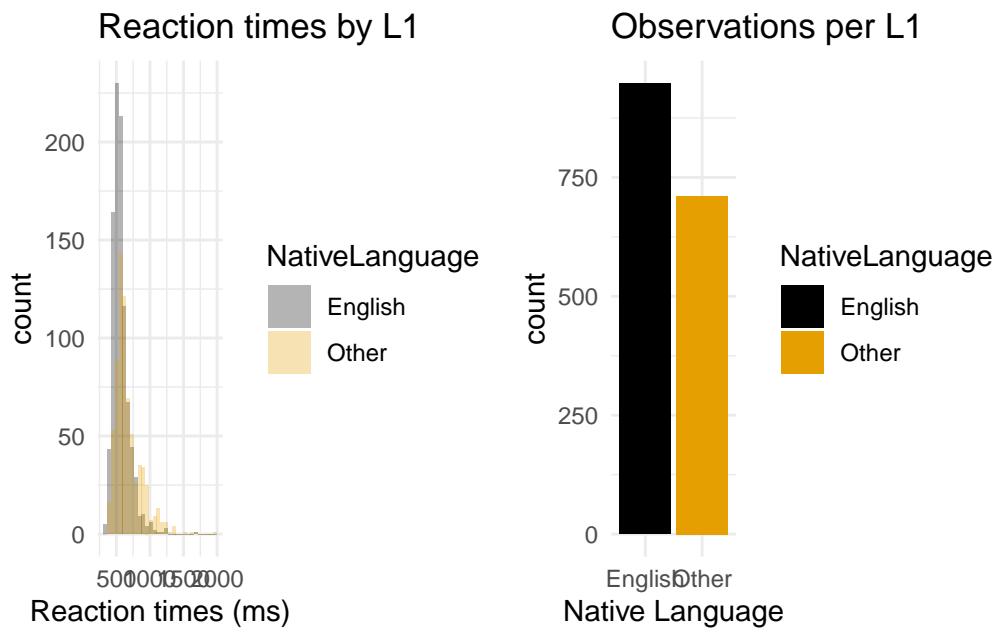


#### 2.4.10. Kombinieren von Plots

- Ein Grund, Ihre Darstellung als Objekt zu speichern, ist, dass wir sie später aufrufen können
  - d.h. Sie können den Plot an einer Stelle in Ihrem Dokument erstellen, sich aber entscheiden, ihn erst im gerenderten Bericht weiter unten zu drucken
- ein weiterer Grund ist, dass wir mehrere Diagramme kombinieren können
  - Dies kann mit einer Vielzahl von Paketen geschehen
  - Versuchen wir es mit dem Paket **patchwork**
    - \* Benutze + um zwei Plots nebeneinander zu verbinden
    - \* oder /, um sie übereinander darzustellen

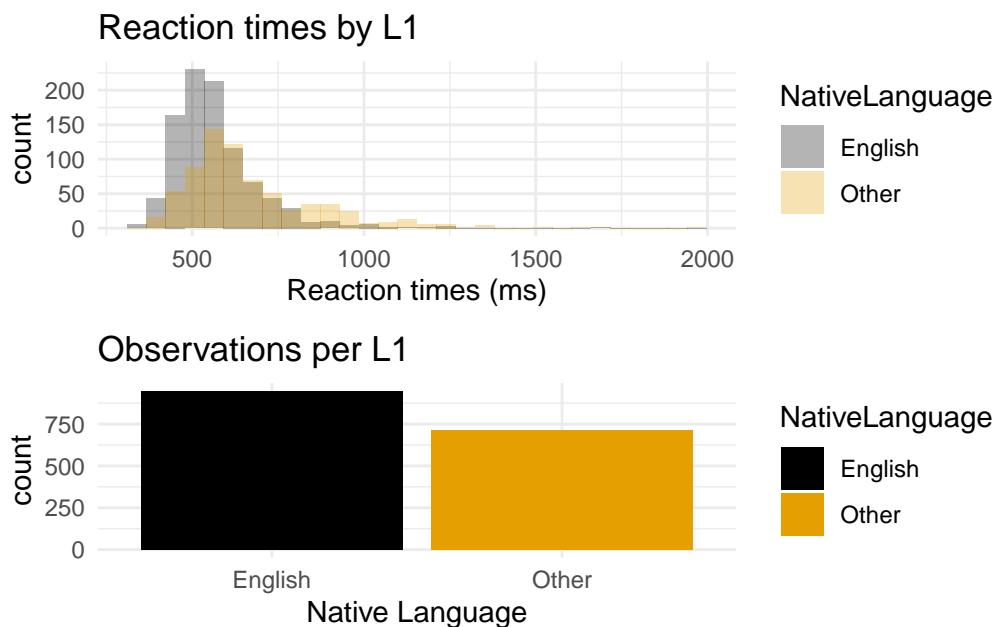
##### 2.4.10.1. Kombinieren von Plots mit +

```
fig_lexdec_rt + fig_lexdec_11
```



#### 2.4.10.2. Kombinieren von Plots mit /

```
fig_lexdec_rt / fig_lexdec_l1
```



## 2.5. Entscheidung für ein Geom

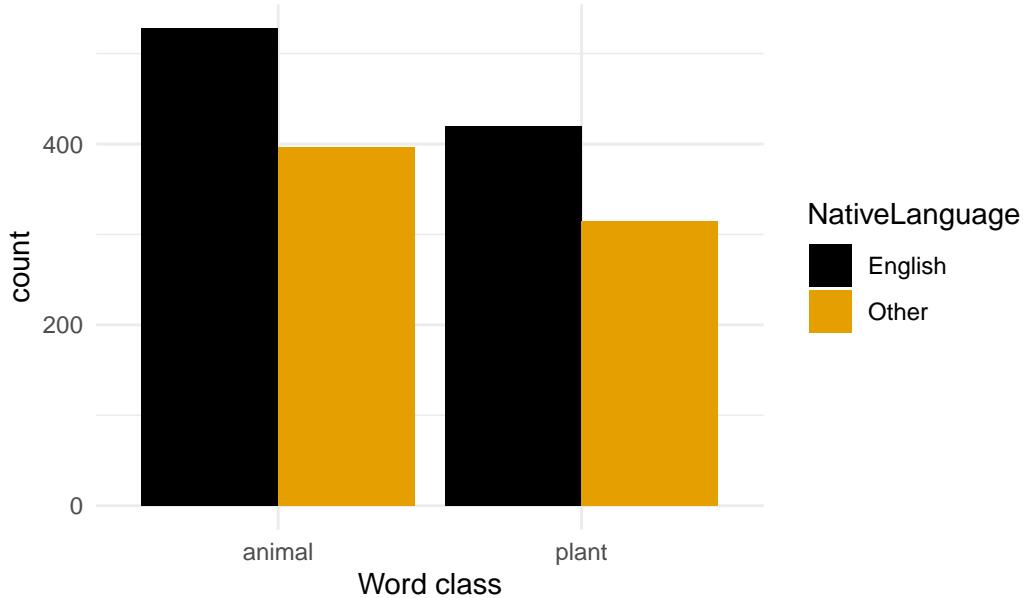
- Warum verwenden wir ein Histogramm für die Reaktionszeit und ein Balkendiagramm für die Muttersprache?
- Um welche Arten von Variablen handelt es sich?
  - Reaktionszeit ist kontinuierlich
  - Muttersprache ist eine kategoriale Variable
- Wir verwenden Histogramme, um die Verteilungen von *kontinuierlichen* Variablen zu visualisieren.
- Wir verwenden Balkendiagramme, um Verteilungen von *kategorischen* Variablen zu visualisieren.
- Wenn wir wissen, was wir visualisieren wollen (z. B. Verteilungen) und welche Art von Variable wir haben (d. h. kontinuierlich, kategorial), können wir entscheiden, welche Art von Diagramm wir erstellen wollen.
- Oft ist es eine gute Idee, die Darstellung auf Papier zu zeichnen, bevor man in R beginnt (ich mache das auch oft).

## 2.6. Exercises

Diese Übungen sollten auch in Ihrem Skript enthalten sein, wenn Sie es auf Moodle hochladen. Das Durcharbeiten des Unterrichtsmaterials wird Sie auf diese Aufgaben vorbereiten.

1. Reproduzieren Sie unser Histogramm als *Dichte-Diagramm*, indem Sie `geom_histogram()` durch `geom_density()` ersetzen.
  - Was zeigt diese Art der Darstellung?
2. Erstellen Sie ein Balkendiagramm, das die Anzahl der Beobachtungen pro Wortklasse zeigt (Hinweis: Sie benötigen die Variable `Class` aus unserem Datensatz).
3. Drucken Sie Ihren Dichteplot und Ihren Klassen-Balkenplot übereinander mit Hilfe des `patchwork` Pakets
4. Reproduzieren Sie die folgenden Diagramme so genau wie möglich (Hinweis: Sie benötigen das Argument `position = "dodge"`):

Observations per word class/native language



## Heutige Ziele

Heute haben wir gelernt...

- was Datenrahmen sind
- den Unterschied zwischen kategorialen und kontinuierlichen Daten
- wie man Diagramme mit `ggplot` erstellt
- die richtige Darstellung für unsere Daten auszuwählen

## 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; 1

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] magick_2.7.4     kableExtra_1.3.4 knitr_1.44      patchwork_1.1.3
[5] ggthemes_4.2.4   languageR_1.5.0  lubridate_1.9.2 forcats_1.0.0
[9] stringr_1.5.0   dplyr_1.1.3    purrr_1.0.2    readr_2.1.4
[13] tidyverse_2.0.0  tibble_3.2.1    ggplot2_3.4.3  tidyverse_2.0.0

loaded via a namespace (and not attached):
[1] utf8_1.2.3       generics_0.1.3   xml2_1.3.4     stringi_1.7.12
[5] hms_1.1.3        digest_0.6.33   magrittr_2.0.3  evaluate_0.21
[9] grid_4.3.0       timechange_0.2.0 fastmap_1.1.1   rprojroot_2.0.3
[13] jsonlite_1.8.7   httr_1.4.6     rvest_1.0.3    fansi_1.0.4
[17] viridisLite_0.4.2 scales_1.2.1    cli_3.6.1     rlang_1.1.1
[21] munsell_0.5.0    withr_2.5.0    yaml_2.3.7    tools_4.3.0
[25] tzdb_0.4.0       colorspace_2.1-0 webshot_0.5.4  here_1.0.1
[29] pacman_0.5.1    vctrs_0.6.3    R6_2.5.1     lifecycle_1.0.3
[33] pkgconfig_2.0.3  pillar_1.9.0    gtable_0.3.4  Rcpp_1.0.11
[37] glue_1.6.2       systemfonts_1.0.4 xfun_0.39    tidyselect_1.2.0
[41] rstudioapi_0.14  farver_2.1.1    htmltools_0.5.5 svglite_2.1.1
[45] rmarkdown_2.22    labeling_0.4.3   compiler_4.3.0

```

## Literaturverzeichnis

- Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*.
- Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>
- Davies, R., Locke, S., & D'Agostino McGowan, L. (2022). *datasauRus: Datasets from the Datasaurus Dozen*. <https://CRAN.R-project.org/package=datasauRus>
- Müller, K. (2020). *here: A Simpler Way to Find Your Files*. <https://CRAN.R-project.org/package=here>

- Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills*. Zenodo. <https://doi.org/10.5281/zenodo.6365078>
- Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2022). Data Visualization Using R for Researchers Who Do Not Use R. *Advances in Methods and Practices in Psychological Science*, 5(2), 251524592210746. <https://doi.org/10.1177/25152459221074654>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. <https://doi.org/10.21105/joss.01686>
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2. Aufl.).
- Winter, B. (2019). Statistics for Linguists: An Introduction Using R. In *Statistics for Linguists: An Introduction Using R*. Routledge. <https://doi.org/10.4324/9781315165547>
- Xie, Y. (2023). *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>

# 3. Dynamic reports with Quarto

## Lernziele

- 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

## Lesungen

Die **Pflichtlektüre** zur Vorbereitung auf dieses Thema ist [Kap. 29 \(Quarto\)](#) und [Kap. 30 \(Quarto formats\)](#) in Wickham et al. (2023).

Eine **ergänzende Lektüre** ist [Ch. 2 \(Reproducible Workflows\)](#) in Nordmann & DeBruine (2022). 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 gelernt...

- was Datenrahmen sind
- den Unterschied zwischen kategorialen und kontinuierlichen Daten
- wie man Diagramme mit `ggplot` erstellt
- die richtige Darstellung für unsere Daten auszuwählen

## Wiederholung: `ggplot()`

Sehen Sie sich diesen Code an. Was würde passieren, wenn wir ihn ausführen würden?

```

library(languageR)
library(tidyverse)
df_lexdec <- lexdec

fig_lexdec <-
  df_lexdec |>
  ggplot() +
  aes(x = RT, colour = Class) +
  geom_histogram(position = "identity", alpha = .5) +
  theme_bw()

```

Welche Darstellung in Abbildung 3.1 wird durch den folgenden Code erzeugt?

```

library(languageR)
library(tidyverse)
df_lexdec <- lexdec

fig_lexdec1 <-
  df_lexdec |>
  ggplot() +
  aes(x = RT, colour = Class) +
  geom_density(alpha = .5) +
  theme_bw()

```

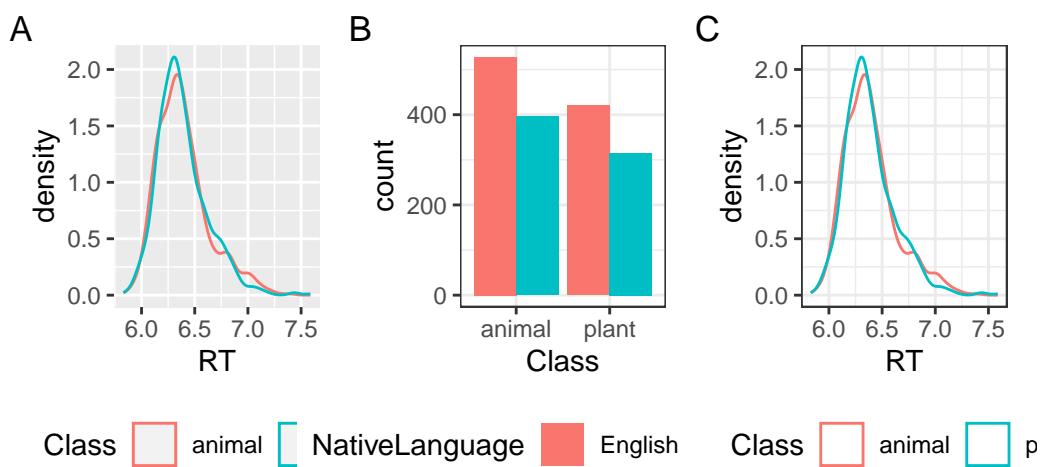


Abbildung 3.1.: Drei aus dem lexdec-Datensatz generierte Diagramme

## Set-up

- wir müssen eine LaTeX-Distribution verwenden, um PDF-Dokumente mit Quarto zu erstellen
  - LaTeX ist ein Schriftsatzsystem
  - TinyTex ist eine eigene LaTeX-Distribution, mit der wir PDFs erstellen können.
  - Das Paket `tinytex` kann uns helfen, TinyTex zu installieren

### Installation von LaTeX über `tinytex`

- Führen Sie den folgenden Code *in der Konsole* aus
- oder, wenn Sie ihn in einem Skript ausführen wollen, um zu dokumentieren, was Sie getan haben, kommentieren Sie ihn nach der Ausführung aus (d.h. fügen Sie ein `#` davor)

```
# run this in the console
install.packages("tinytex")
tinytex::install_tinytex()
```

### Ordner für Woche 3

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.

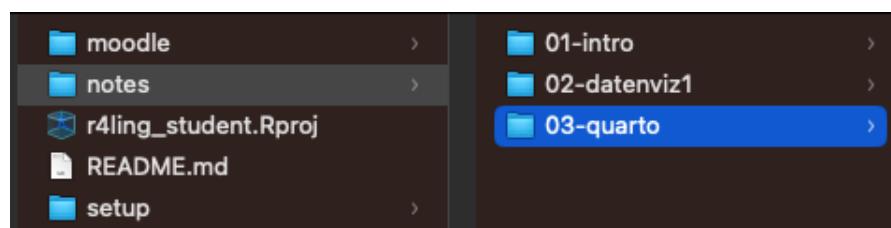


Abbildung 3.2.: Notes folder structure

## 3.1. Quarto

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

### 3.1.1. Dynamische Berichte

- diejenigen, die Text, Code, Codeausgabe enthalten
- Quarto bietet ein “unified authoring framework” für Data Science, das Ihren Text, Ihren Code und Ihre 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 Ihnen in der Zukunft!), die sich sowohl für Ihre Schlussfolgerungen als auch für die Art und Weise interessieren, wie Sie zu ihnen gekommen sind (d. h. für den Code).
  3. als eine Umgebung, in der Datenwissenschaft betrieben wird, als ein modernes Labornotizbuch, in dem wir nicht nur aufzeichnen können, was wir getan haben, sondern auch unsere Gedankengänge.

### 3.1.2. 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 3.1: RStudio version

#### Beispiel 3.1.

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

### 3.1.3. 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)

### 3.1.4. 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

## 3.2. Unsere erstes Quarto-Dokument

- letzte Woche haben wir ein R-Skript erstellt, das wir über Moodle eingereicht haben
- wir werden nun unsere erste .qmd-Datei erstellen
- von nun an wird dies die Datei sein, die wir in Moodle einreichen (kein R-Skript)

💡 Aufgabe 3.2: erste Quarto

#### Beispiel 3.2.

1. Erstellen Sie in Ihrem R-Projekt-Ordner, in dem ihr Ihre 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 03-quarto
  - Was hat sich geändert?

### 3.2.1. Quarto-Grundlagen

- Quarto-Dokumente (wie Rmarkdowm) 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 `r`

```
```{r}
#| code-line-numbers: false
## Dies ist ein Code Chunk
1 + 1
```
```

[1] 2

### 3.2.2. 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, Autorennname
- 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 3.3: YAML

#### Beispiel 3.3.

1. Ändern Sie den Titel, wenn Sie das tun möchten.
2. Raten Sie, wie man einen “Untertitel” (EN: subtitle) hinzufügen könnte (Hinweis: es ist ähnlich wie beim Hinzufügen eines title)
3. Fügen Sie einen Autor hinzu, Autor: "vorname nachname" (siehe Beispiel unten)
4. Füge ein Inhaltsverzeichnis hinzu (EN: Table of Contents, toc), indem du format so änderst, dass es wie folgt aussieht:

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

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

#### 3.2.3. Strukturierung Ihres Dokuments

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

```
---
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.

```

### **i** 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 3.4:** Überschriften

#### **Beispiel 3.4.**

1. Kopieren 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 seht Sie, wenn Sie darauf klicken?

### 3.2.4. Textformatierung

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

| Format        | Markdown                                | Ausgabe                                 |
|---------------|---|---|
| Kursivschrift | Dieser Text ist <i>kursiv</i>           | Dieser Text ist <i>kursiv</i>           |
| Fett          | Dieser Text ist <b>**fett**</b>         | Dieser Text ist <b>fett</b>             |
| Subskription  | Dieser Text ist <u>tiefgestellt</u>     | Dieser Text ist <u>tiefgestellt</u>     |
| Hochgestellt  | Dieser Text ist <sup>hochgestellt</sup> | Dieser Text ist <sup>hochgestellt</sup> |

### 3.2.5. 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 (beachtet 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 (beachtet den doppelten Einzug)  
 2. dies ist der zweite nummerierte Punkt

### Aufgabe 3.5: Aufzählungen

#### Beispiel 3.5.

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?

## 3.3. 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)

### 3.3.1. 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 Ihrer 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 3.6: Code-Chunks

#### Beispiel 3.6.

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?

#### 3.3.2. 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

- Wichtig ist, dass die Codechunk-Optionen:
  - mit `#|` beginnen, mit einem Leerzeichen dahinter und keinem Leerzeichen davor
  - direkt unter ````{r}` platziert werden

#### Aufgabe 3.7: `c()`

##### Beispiel 3.7.

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. `Objektnname <- c(...)`)
3. Berechnen Sie den Mittelwert dieser Zahlen, indem Sie nur den Objektnamen verwendet.
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?

## 3.4. 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

### 3.4.1. Set-up

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

```
```{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
```
```

### 3.4.2. 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
```

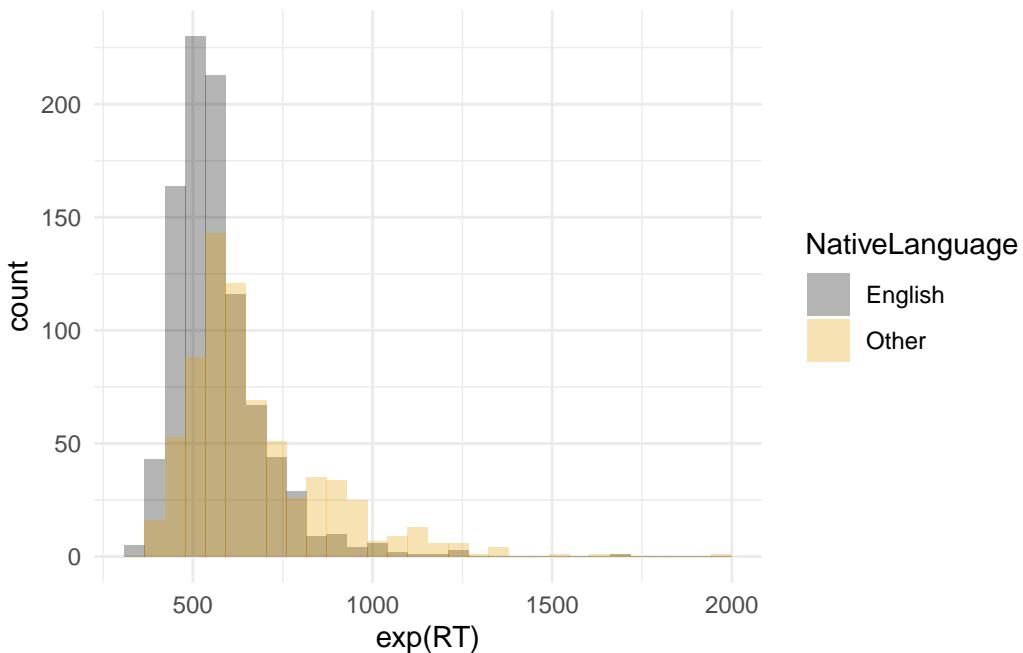


Abbildung 3.3.: Histogram of reactiontimes per native language from lexdec

### 3.4.3. 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 3.8: Plots in Quarto

#### Beispiel 3.8.

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.

```

fig_lexdec_11 <-
  ggplot(data = df_lexdec) +
  aes(x = NativeLanguage, fill = NativeLanguage) +
  ## add the geom:
  geom_bar() +
  scale_fill_colorblind() + ## add colourblind colours
  theme_minimal()

fig_lexdec_11

```

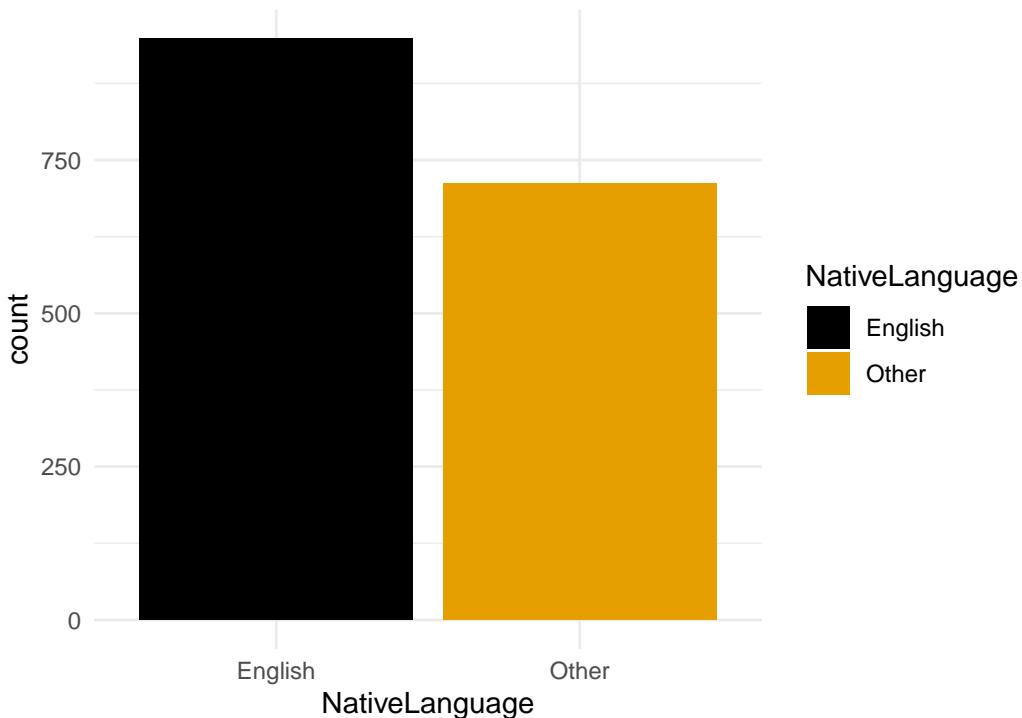


Abbildung 3.4.: Barplot of observations per native language

### 3.5. Ausgabeformate

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

### 3.5.1. 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 3.5.: 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 3.9: Ausgabeformate

##### Beispiel 3.9.

1. Ersetzt `html` in der YAML durch `revealjs`. Rendert das Dokument.
  - Schauen Sie den Ordner für die Notizen dieser Woche an. Welche Dateien sieht?
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 sieht?
  - Ist die Ausgabe von `revealjs` dort?

## Lernziele

Wir haben...

- 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

## 3.6. 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 3.10: Session Info

**Beispiel 3.10.**

- 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

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  magick_2.7.4    patchwork_1.1.3 lubridate_1.9.2
[5] forcats_1.0.0   stringr_1.5.0   dplyr_1.1.3     purrr_1.0.2
[9] readr_2.1.4     tidyverse_2.0.0  tibble_3.2.1    ggplot2_3.4.3
[13] tidyverse_2.0.0 languageR_1.5.0

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       commonmark_1.9.0 munsell_0.5.0
[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   rmarkdown_2.22  labeling_0.4.3  compiler_4.3.0
[45] markdown_1.7

```

## Literaturverzeichnis

Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R.*

- Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>
- Davies, R., Locke, S., & D'Agostino McGowan, L. (2022). *datasauRus: Datasets from the Datasaurus Dozen*. <https://CRAN.R-project.org/package=datasauRus>
- Müller, K. (2020). *here: A Simpler Way to Find Your Files*. <https://CRAN.R-project.org/package=here>
- Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills*. Zenodo. <https://doi.org/10.5281/zenodo.6365078>
- Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2022). Data Visualization Using R for Researchers Who Do Not Use R. *Advances in Methods and Practices in Psychological Science*, 5(2), 251524592210746. <https://doi.org/10.1177/25152459221074654>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. <https://doi.org/10.21105/joss.01686>
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2. Aufl.).
- Winter, B. (2019). Statistics for Linguists: An Introduction Using R. In *Statistics for Linguists: An Introduction Using R*. Routledge. <https://doi.org/10.4324/9781315165547>
- Xie, Y. (2023). *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>

# 4. Data Wrangling 1: Transformation

Umwandlung von Daten

## Lesungen

Die **Pflichtlektüre** zur Vorbereitung auf dieses Thema ist [Kap. 4 \(Data Transformation\)](#) in Wickham et al. (2023).

Eine **ergänzende Lektüre** ist [Ch. 9 \(Data Wrangling\)](#) in Nordmann & DeBruine (2022).

## Wiederholung

Letzte Woche 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

## Heutige Ziele

Heute werden wir...

- lernen, wie man Daten mit dem Paket `dplyr` aus dem `tidyverse` verarbeitet
- lernen, wie man die `pipe` (`|>`) verwendet, um das Ergebnis einer Funktion in eine andere Funktion einzuspeisen
- Funktionen kennenlernen, die auf Zeilen operieren
- Funktionen kennenlernen, die mit Spalten arbeiten
- lernen, wie man `dplyr`-Funktionen mit Plots von `ggplot2` kombiniert

## Lust auf mehr?

- Kapitel 4 (Data transformation) in ([wickham\\_r\\_nodate?](#))
- Kapitel 9 (Data wrangling) in Nordmann & DeBruine (2022)

## 4.1. Voraussetzungen

### 1. Frisches Quarto-Dokument

- Erstellen Sie ein neues Quarto-Dokument für den heutigen Unterricht
  - Datei > Neues Dokument > Quarto Dokument, mit dem Namen 04-wrangling
- YAML einrichten: Titel, Ihr Name, ein toc hinzufügen

```
title: "Data wrangling"
subtitle: "Transforming data"
author: "Your name here"
lang: de
date: "11/08/2023"
format:
  html:
    toc: true
```

### 2. Pakete

- Die heutigen Pakete sind:
  - tidyverse: zum Verarbeiten (dplyr) und Plotten (ggplot2)
  - languageR: für linguistische Datensätze

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

### 3. Daten

- wir arbeiten wieder mit dem lexdec-Datensatz aus dem languageR-Paket (Baayen & Shafaei-Bajestan, 2019)
- wir speichern ihn als Objekt mit dem Namen df\_lexdec
- wir wandeln auch die Variable RT um, so dass sie in Millisekunden angegeben wird (vorher war sie in log Millisekunden angegeben, aber machen Sie sich keine Gedanken darüber, was das bedeutet)
- und wir wählen 10 Variablen aus, die für uns heute relevant sind

```

df_lexdec <- lexdec |>
  mutate(RT = exp(RT)) |>
  select(Subject, RT, Trial, Sex, NativeLanguage, Correct, Word, Frequency, Class, Length)

```

## 4.2. Data Wrangling

- Im Englischen bezieht sich “wrangling” auf einen langen, schwierigen Prozess
  - z. B. treiben Cowboys ihre Rinder oder Herden zusammen (sammeln, sammeln ihre Tiere)
- Es gibt zwei Hauptbestandteile des Wrangling
  - Transformieren: Sortieren oder Erstellen neuer Variablen (was wir heute tun werden)
  - Aufräumen: Umformung oder Strukturierung Ihrer Daten (dies werden wir in einigen Wochen tun)
- Sowohl das Aufräumen als auch das Transformieren von Daten erfordern das Paket `dplyr` aus dem `tidyverse`.
  - `dplyr` Funktionen werden oft als Verben bezeichnet, weil sie etwas *tun*

### Der Name `dplyr`

- Der Name `dplyr` kommt von einem früheren Paket, `plyr`, das dazu verwendet wird, Daten zu zerlegen, Funktionen darauf anzuwenden und zu kombinieren
  - Im Englischen klingt `plyr` wie das Wort für Zangen (“pliers”), die benutzt werden, um Dinge auseinander zu nehmen, wie das, was `plyr` mit Daten macht
  - das “d” in “`dplyr`” wurde hinzugefügt, weil das Paket speziell für die Arbeit mit Datenrahmen gedacht ist

### 4.2.1. `lexdec`

- der `lexdec`-Datensatz enthält Daten für eine lexikalische Entscheidungsaufgabe im Englischen
  - Schauen wir uns den Datensatz mit der Funktion `head()` an, die nur die ersten 6 Zeilen ausgibt
    - \* hier geben wir die ersten 10 Zeilen aus

- In meinen Materialien verwende ich oft die Funktion „head()“, um zu vermeiden, dass der gesamte Datensatz in der Ausgabe gedruckt wird, aber Sie würden im Allgemeinen nicht „head()“ verwenden wollen, wenn Sie Ihre Daten betrachten, sondern Ihren gesamten Datensatz betrachten wollen

 Aufgabe 4.1: df\_lexdec

**Beispiel 4.1.**

1. Betrachten Sie den Datensatz
  - wie viele Beobachtungen gibt es?
  - Wie viele Variablen gibt es?
2. Geben Sie den Datensatz in die Funktion `glimpse()` ein.
  - Was zeigt Ihnen das?
  - Wie sieht es im Vergleich zu dem aus, was Sie sehen, wenn Sie `summary()` verwenden?

#### 4.2.2. dplyr-Grundlagen

- heute lernen wir einige der wichtigsten `dplyr`-Verben (Funktionen) kennen, mit denen wir die meisten unserer Datenmanipulationsprobleme lösen können
  - Ich verwende diese Verben mehrfach in wahrscheinlich jedem Analyseskript
- Die `dplyr`-Verben haben einige Dinge gemeinsam:
  1. das erste Argument ist immer ein Datenrahmen
  2. die folgenden Argumente beschreiben in der Regel die zu bearbeitenden Spalten, wobei der Variablenname (ohne Anführungszeichen) verwendet wird
  3. die Ausgabe ist immer ein neuer Datenrahmen
- Die Verben sind alle für eine Sache gut geeignet, so dass wir oft mehrere Verben auf einmal verwenden wollen.
  - Wir verwenden dazu die Pipe (`|>` oder `|>`)
  - Wir haben diese Pipe bereits gesehen, als wir einen Datenrahmen in `ggplot()` einspeisten.
  - wir können die Pipe als `und dann` lesen
- In dem folgenden Code identifizieren
  - den Datenrahmen
  - `dplyr`-Verben

- Variablennamen
- Kannst du versuchen, herauszulesen (zu erraten), was der folgende Code macht?

```
df_lexdec |>
  filter(Subject == "A1") |>
  select(Subject, Trial, RT, NativeLanguage, Word) |>
  relocate(NativeLanguage, .after = Trial)
```

### Korrekte Syntax

.Beachten Sie, dass A1 mit Anführungszeichen geschrieben wird, aber keiner der anderen Codes. Wenn wir ein Objekt (z.B. df\_lexdec) oder seine Variablen (z.B. Subject) aufrufen, setzen wir sie nicht in Anführungszeichen. Wenn wir einen bestimmten Wert einer Variablen aufrufen, der nicht numerisch ist, müssen wir diesen Wert in Anführungszeichen setzen, weil die Subject ID A1 ein Wert der Variablen Subject ist, müssen wir sie in Anführungszeichen setzen.

Versuchen Sie, die Anführungszeichen zu entfernen. Welche Fehlermeldung erhalten Sie?  
Versuchen Sie, einen Variablenamen in Anführungszeichen zu setzen, welche Fehlermeldung erhalten Sie?

Dies ist eine wichtige Übung, denn Sie werden oft feststellen, dass Ihr Code nicht läuft, aber die Lösung ist oft etwas so Einfaches wie fehlende oder zusätzliche Anführungszeichen oder Interpunktionsfehler.

## 4.3. Zeilen

- In aufgeräumten Daten stellen die Zeilen Beobachtungen dar.
- die wichtigsten Verben für Zeilen sind:
  - `filter()`: ändert, welche Zeilen vorhanden sind
  - `arrange()`: ändert die Reihenfolge der Zeilen
- Wir besprechen auch
  - `distinct()`: findet Zeilen mit unterschiedlichen Werten basierend auf einer Variablen (Spalte)

### 4.3.1. `filter()`

- ändert, welche Zeilen vorhanden sind, ohne ihre Reihenfolge zu ändern

- nimmt den Datenrahmen als erstes Argument
  - Die folgenden Argumente sind Bedingungen, die TRUE sein müssen, damit die Zeile erhalten bleibt
- findet alle Reaktionszeiten, die länger als 450 Millisekunden waren:

```
df_lexdec |>
  filter(RT > 450) |>
  head()
```

|   | Subject | RT       | Trial | Sex | NativeLanguage | Correct | Word       | Frequency | Class  |
|---|---------|----------|-------|-----|----------------|---------|------------|-----------|--------|
| 1 | A1      | 566.9998 | 23    | F   | English        | correct | owl        | 4.859812  | animal |
| 2 | A1      | 548.9998 | 27    | F   | English        | correct | mole       | 4.605170  | animal |
| 3 | A1      | 572.0000 | 29    | F   | English        | correct | cherry     | 4.997212  | plant  |
| 4 | A1      | 486.0002 | 30    | F   | English        | correct | pear       | 4.727388  | plant  |
| 6 | A1      | 483.0002 | 33    | F   | English        | correct | blackberry | 4.060443  | plant  |
| 8 | A1      | 524.9999 | 38    | F   | English        | correct | squirrel   | 4.709530  | animal |
|   | Length  |          |       |     |                |         |            |           |        |
| 1 |         | 3        |       |     |                |         |            |           |        |
| 2 |         | 4        |       |     |                |         |            |           |        |
| 3 |         | 6        |       |     |                |         |            |           |        |
| 4 |         | 4        |       |     |                |         |            |           |        |
| 6 |         | 10       |       |     |                |         |            |           |        |
| 8 |         | 8        |       |     |                |         |            |           |        |

- Beachten Sie, dass wir den Wert der Reaktionszeit nicht in Anführungszeichen setzen, da er *numerisch* ist
- wenn Sie die gefilterten Daten speichern wollen, ist es in der Regel ratsam, sie unter einem *neuen* Objektnamen zu speichern
  - wenn Sie die vorgefilterte Version nicht überschreiben wollen, ist ein neuer Name erforderlich

```
df_lexdec_450 <-
  df_lexdec |>
  filter(RT > 450)
```

### i Logische Operatoren

- Symbole, die zur Beschreibung einer logischen Bedingung verwendet werden
  - == ist *identisch* ( $1 == 1$ )

- `!=` ist nicht identisch (`1 != 2`)
- `>` ist größer als (`2 > 1`)
- `<` ist kleiner als (`1 < 2`)
- um Bedingungen zu kombinieren
  - `&` oder `, und auch` (für mehrere Bedingungen)
  - `| oder` (für mehrere Bedingungen)
- es gibt eine nette Abkürzung für die Kombination von `==` und `|: %in%`
  - behält Zeilen, in denen die Variable gleich einem der Werte auf der rechten Seite ist

#### 4.3.1.1. == und |

```
df_lexdec |>
  filter(Trial == 30 | Trial == 23) |>
  head()
```

|     | Subject | RT       | Trial | Sex | NativeLanguage | Correct | Word    | Frequency | Class  |
|-----|---------|----------|-------|-----|----------------|---------|---------|-----------|--------|
| 1   | A1      | 566.9998 | 23    | F   | English        | correct | owl     | 4.859812  | animal |
| 4   | A1      | 486.0002 | 30    | F   | English        | correct | pear    | 4.727388  | plant  |
| 475 | A2      | 561.0001 | 23    | M   | English        | correct | dog     | 7.667626  | animal |
| 949 | C       | 688.0001 | 23    | F   | English        | correct | vulture | 4.248495  | animal |
| 83  | D       | 553.0000 | 30    | M   | Other          | correct | walnut  | 4.499810  | plant  |
| 317 | J       | 824.0004 | 23    | F   | Other          | correct | beaver  | 3.951244  | animal |
|     | Length  |          |       |     |                |         |         |           |        |
| 1   |         | 3        |       |     |                |         |         |           |        |
| 4   |         | 4        |       |     |                |         |         |           |        |
| 475 |         | 3        |       |     |                |         |         |           |        |
| 949 |         | 7        |       |     |                |         |         |           |        |
| 83  |         | 6        |       |     |                |         |         |           |        |
| 317 |         | 6        |       |     |                |         |         |           |        |

#### 4.3.1.2. %in%

```
df_lexdec |>
  filter(Trial %in% c(30, 23)) |>
  head()
```

|   | Subject | RT       | Trial | Sex | NativeLanguage | Correct | Word | Frequency | Class  |
|---|---------|----------|-------|-----|----------------|---------|------|-----------|--------|
| 1 | A1      | 566.9998 | 23    | F   | English        | correct | owl  | 4.859812  | animal |

|        |    |          |    |   |         |         |         |          |        |
|--------|----|----------|----|---|---------|---------|---------|----------|--------|
| 4      | A1 | 486.0002 | 30 | F | English | correct | pear    | 4.727388 | plant  |
| 475    | A2 | 561.0001 | 23 | M | English | correct | dog     | 7.667626 | animal |
| 949    | C  | 688.0001 | 23 | F | English | correct | vulture | 4.248495 | animal |
| 83     | D  | 553.0000 | 30 | M | Other   | correct | walnut  | 4.499810 | plant  |
| 317    | J  | 824.0004 | 23 | F | Other   | correct | beaver  | 3.951244 | animal |
| Length |    |          |    |   |         |         |         |          |        |
| 1      |    | 3        |    |   |         |         |         |          |        |
| 4      |    | 4        |    |   |         |         |         |          |        |
| 475    |    | 3        |    |   |         |         |         |          |        |
| 949    |    | 7        |    |   |         |         |         |          |        |
| 83     |    | 6        |    |   |         |         |         |          |        |
| 317    |    | 6        |    |   |         |         |         |          |        |

#### 💡 Aufgabe 4.2: filter()

#### Beispiel 4.2.

1. Filtern Sie die Daten, um Zeilen aus Versuch 25 und Nicht-Muttersprachler (**andere**) einzuschließen.
2. Wie viele Zeilen gibt es?

#### 4.3.2. arrange()

- ändert die Reihenfolge der Zeilen auf der Grundlage eines Wertes in einer oder mehreren Spalten

```
df_lexdec |>
  arrange(RT) |>
  head()
```

|              | Subject | RT       | Trial | Sex | NativeLanguage | Correct   | Word    | Frequency |
|--------------|---------|----------|-------|-----|----------------|-----------|---------|-----------|
| 542          | A2      | 340.0001 | 159   | M   | English        | incorrect | pig     | 6.660575  |
| 815          | K       | 347.9998 | 83    | F   | English        | incorrect | lemon   | 5.631212  |
| 822          | K       | 363.0001 | 99    | F   | English        | incorrect | potato  | 6.461468  |
| 73           | A1      | 364.9999 | 174   | F   | English        | correct   | chicken | 6.599870  |
| 524          | A2      | 365.9999 | 117   | M   | English        | correct   | goose   | 5.267858  |
| 1516         | I       | 367.0001 | 51    | F   | Other          | correct   | carrot  | 4.976734  |
| Class Length |         |          |       |     |                |           |         |           |
| 542          | animal  |          |       | 3   |                |           |         |           |
| 815          | plant   |          |       | 5   |                |           |         |           |

```

822 plant      6
73  animal     7
524 animal     5
1516 plant     6

```

- wenn Sie mehr als einen Spaltennamen verwenden, wird jede zusätzliche Spalte verwendet, um die Verbindung zwischen den Werten der vorangegangenen Spalten zu lösen

```

df_lexdec |>
  arrange(Length, Sex) |>
  head(10)

```

|     | Subject | RT       | Trial | Sex | NativeLanguage | Correct   | Word | Frequency | Class  |
|-----|---------|----------|-------|-----|----------------|-----------|------|-----------|--------|
| 1   | A1      | 566.9998 | 23    | F   | English        | correct   | owl  | 4.859812  | animal |
| 5   | A1      | 414.0000 | 32    | F   | English        | correct   | dog  | 7.667626  | animal |
| 15  | A1      | 556.9999 | 53    | F   | English        | correct   | bee  | 5.700444  | animal |
| 20  | A1      | 456.9998 | 61    | F   | English        | incorrect | bat  | 5.918894  | animal |
| 31  | A1      | 581.9997 | 88    | F   | English        | correct   | fox  | 5.652489  | animal |
| 44  | A1      | 494.0002 | 113   | F   | English        | correct   | pig  | 6.660575  | animal |
| 62  | A1      | 467.9999 | 152   | F   | English        | correct   | cat  | 7.086738  | animal |
| 64  | A1      | 875.9999 | 157   | F   | English        | correct   | ant  | 5.347108  | animal |
| 719 | A3      | 607.0001 | 41    | F   | Other          | correct   | ant  | 5.347108  | animal |
| 720 | A3      | 562.0001 | 44    | F   | Other          | correct   | pig  | 6.660575  | animal |
|     | Length  |          |       |     |                |           |      |           |        |
| 1   |         | 3        |       |     |                |           |      |           |        |
| 5   |         | 3        |       |     |                |           |      |           |        |
| 15  |         | 3        |       |     |                |           |      |           |        |
| 20  |         | 3        |       |     |                |           |      |           |        |
| 31  |         | 3        |       |     |                |           |      |           |        |
| 44  |         | 3        |       |     |                |           |      |           |        |
| 62  |         | 3        |       |     |                |           |      |           |        |
| 64  |         | 3        |       |     |                |           |      |           |        |
| 719 |         | 3        |       |     |                |           |      |           |        |
| 720 |         | 3        |       |     |                |           |      |           |        |

- wir können `desc()` innerhalb von `arrange()` hinzufügen, um eine absteigende Reihenfolge (groß-klein) anstelle der standardmäßigen aufsteigenden Reihenfolge zu verwenden

```

df_lexdec |>
  arrange(desc(Length)) |>

```

```
head()
```

|     | Subject | RT       | Trial | Sex | NativeLanguage | Correct | Word       | Frequency |
|-----|---------|----------|-------|-----|----------------|---------|------------|-----------|
| 6   | A1      | 483.0002 | 33    | F   | English        | correct | blackberry | 4.060443  |
| 7   | A1      | 417.9998 | 34    | F   | English        | correct | strawberry | 4.753590  |
| 69  | A1      | 540.9998 | 168   | F   | English        | correct | woodpecker | 2.890372  |
| 505 | A2      | 503.9999 | 87    | M   | English        | correct | woodpecker | 2.890372  |
| 516 | A2      | 400.9998 | 105   | M   | English        | correct | strawberry | 4.753590  |
| 518 | A2      | 517.0001 | 108   | M   | English        | correct | blackberry | 4.060443  |
|     | Class   | Length   |       |     |                |         |            |           |
| 6   | plant   | 10       |       |     |                |         |            |           |
| 7   | plant   | 10       |       |     |                |         |            |           |
| 69  | animal  | 10       |       |     |                |         |            |           |
| 505 | animal  | 10       |       |     |                |         |            |           |
| 516 | plant   | 10       |       |     |                |         |            |           |
| 518 | plant   | 10       |       |     |                |         |            |           |

#### 💡 Aufgabe 4.3: `arrange()`

##### Beispiel 4.3.

1. Filtere die Daten so, dass sie nur Beobachtungen der “Probanden” M1 und W2 enthalten, *und dann*
2. Ordnen Sie die Daten nach absteigender Reaktionszeit

## 4.4. Spalten

- In Tidy Data stellen die Spalten Variablen dar.
- die wichtigsten Verben für Spalten sind:
  - `rename()`: ändert die Namen der Spalten
  - `mutate()`: erzeugt neue Spalten, die von den vorhandenen Spalten abgeleitet werden
  - `select()`: ändert, welche Spalten vorhanden sind
  - `relocate()`: ändert die Position der Spalten

### 4.4.1. `rename()`

- Mit `rename()` können wir den Namen von Spalten ändern

- die Reihenfolge der Argumente ist `neuer_name = alter_name`
- Versuchen wir, einige der Variablennamen auf Deutsch zu ändern
  - Ich neige dazu, Variablennamen in Kleinbuchstaben zu schreiben, als Kodierungskonvention

```
## single variable
df_lexent <-
  df_lexdec |>
  rename(teilnehmer = Subject)

## or multiple variables at once
df_lexent <-
  df_lexdec |>
  rename(teilnehmer = Subject,
         rz_ms = RT,
         geschlect = Sex,
         laenge = Length)
```

#### 4.4.2. `mutate()`

- Mit `mutate()` werden neue Spalten aus vorhandenen Spalten erzeugt.
  - So können wir z.B. einfache Algebra mit den Werten in jeder Spalte durchführen

```
df_lexent |>
  mutate(
    rz_laenge = rz_ms / laenge,
  ) |>
  head()
```

|   | teilnehmer | rz_ms    | Trial  | geschlect | NativeLanguage | Correct | Word       |
|---|------------|----------|--------|-----------|----------------|---------|------------|
| 1 | A1         | 566.9998 | 23     | F         | English        | correct | owl        |
| 2 | A1         | 548.9998 | 27     | F         | English        | correct | mole       |
| 3 | A1         | 572.0000 | 29     | F         | English        | correct | cherry     |
| 4 | A1         | 486.0002 | 30     | F         | English        | correct | pear       |
| 5 | A1         | 414.0000 | 32     | F         | English        | correct | dog        |
| 6 | A1         | 483.0002 | 33     | F         | English        | correct | blackberry |
|   | Frequency  | Class    | laenge | rz_laenge |                |         |            |
| 1 | 4.859812   | animal   | 3      | 188.99994 |                |         |            |
| 2 | 4.605170   | animal   | 4      | 137.24994 |                |         |            |
| 3 | 4.997212   | plant    | 6      | 95.33333  |                |         |            |

```

4 4.727388 plant      4 121.50005
5 7.667626 animal     3 138.00000
6 4.060443 plant     10 48.30002

```

- Mit `mutate()` werden diese neuen Spalten auf der rechten Seite des Datensatzes hinzugefügt.
  - Das macht es schwierig zu sehen, was passiert.
- um zu kontrollieren, wo die neue Spalte hinzugefügt wird, können wir `.before` oder `.after` verwenden

```

df_lexent |>
  mutate(
    rz_laenge = rz_ms / laenge,
    .after = rz_ms
  ) |>
  head()

```

|   | teilnehmer | rz_ms     | rz_laenge | Trial  | geschlect | NativeLanguage | Correct |
|---|------------|-----------|-----------|--------|-----------|----------------|---------|
| 1 | A1         | 566.9998  | 188.99994 | 23     | F         | English        | correct |
| 2 | A1         | 548.9998  | 137.24994 | 27     | F         | English        | correct |
| 3 | A1         | 572.0000  | 95.33333  | 29     | F         | English        | correct |
| 4 | A1         | 486.0002  | 121.50005 | 30     | F         | English        | correct |
| 5 | A1         | 414.0000  | 138.00000 | 32     | F         | English        | correct |
| 6 | A1         | 483.0002  | 48.30002  | 33     | F         | English        | correct |
|   | Word       | Frequency | Class     | laenge |           |                |         |
| 1 | owl        | 4.859812  | animal    | 3      |           |                |         |
| 2 | mole       | 4.605170  | animal    | 4      |           |                |         |
| 3 | cherry     | 4.997212  | plant     | 6      |           |                |         |
| 4 | pear       | 4.727388  | plant     | 4      |           |                |         |
| 5 | dog        | 7.667626  | animal    | 3      |           |                |         |
| 6 | blackberry | 4.060443  | plant     | 10     |           |                |         |

### ! Rendernpause!

Nehmen Sie sich einen Moment Zeit, um Ihr Dokument zu rendern. Wird es gerendert? Können Sie das Dokument besser strukturieren? Z. B. durch Hinzufügen von mehr Überschriften, Text?

 Aufgabe 4.4: `mutate()`

**Beispiel 4.4.**

1. Create a new variable called `rz_s` in `df_lexent`:
  - equals `rz_ms` divided by 1000 (i.e., converts milliseconds to seconds)
  - appears after `rz_ms`
2. Render your document

#### 4.4.3. `select()`

- `select()` fasst die Daten so zusammen, dass sie nur die gewünschten Spalten enthalten
- Spalten nach Namen auswählen

```
df_lexent |>  
  select(teilnehmer, rz_ms, Word) |>  
  head()
```

```
teilnehmer      rz_ms        Word  
1          A1 566.9998      owl  
2          A1 548.9998     mole  
3          A1 572.0000   cherry  
4          A1 486.0002      pear  
5          A1 414.0000      dog  
6          A1 483.0002 blackberry
```

- select alle Spalten zwischen `rz_ms` und `geschlecht`

```
df_lexent |>  
  select(rz_ms:geschlect) |>  
  head()
```

```
rz_ms      rz_s Trial geschlecht  
1 566.9998 0.5669998    23      F  
2 548.9998 0.5489998    27      F  
3 572.0000 0.5720000    29      F  
4 486.0002 0.4860002    30      F  
5 414.0000 0.4140000    32      F  
6 483.0002 0.4830002    33      F
```

- alle Spalten außer `rz_s` auswählen (! wird als “nicht” gelesen)

```
df_lexent |>
  select(!rz_s) |>
  head()
```

|   | teilnehmer | rz_ms    | Trial  | geschlect | NativeLanguage | Correct | Word       |
|---|------------|----------|--------|-----------|----------------|---------|------------|
| 1 | A1         | 566.9998 | 23     | F         | English        | correct | owl        |
| 2 | A1         | 548.9998 | 27     | F         | English        | correct | mole       |
| 3 | A1         | 572.0000 | 29     | F         | English        | correct | cherry     |
| 4 | A1         | 486.0002 | 30     | F         | English        | correct | pear       |
| 5 | A1         | 414.0000 | 32     | F         | English        | correct | dog        |
| 6 | A1         | 483.0002 | 33     | F         | English        | correct | blackberry |
|   | Frequency  | Class    | laenge |           |                |         |            |
| 1 | 4.859812   | animal   | 3      |           |                |         |            |
| 2 | 4.605170   | animal   | 4      |           |                |         |            |
| 3 | 4.997212   | plant    | 6      |           |                |         |            |
| 4 | 4.727388   | plant    | 4      |           |                |         |            |
| 5 | 7.667626   | animal   | 3      |           |                |         |            |
| 6 | 4.060443   | plant    | 10     |           |                |         |            |

#### 4.4.3.1. `select()`-Hilfsfunktionen

- einige Hilfsfunktionen, die das Leben bei der Arbeit mit `select()` erleichtern:
  - `starts_with("abc")`: wählt Spalten aus, die mit einer bestimmten Zeichenkette beginnen
  - `ends_with("xyz")`: wählt Spalten aus, die mit einer bestimmten Zeichenkette enden
  - `contains("ijk")`: wählt Spalten aus, die eine bestimmte Zeichenkette enthalten
  - `where(is.character)`: wählt Spalten aus, die einem logischen Kriterium entsprechen
    - \* z.B. gibt die Funktion `is.character()` den Wert TRUE zurück, wenn eine Variable Zeichenketten enthält, nicht numerische Werte oder Kategorien

```
df_lexent |>
  select(starts_with("w")) |>
  head()
```

|   | Word |
|---|------|
| 1 | owl  |

```
2      mole
3      cherry
4      pear
5      dog
6 blackberry
```

```
df_lexent |>
  select(ends_with("er")) |>
  head()
```

```
teilnehmer
1      A1
2      A1
3      A1
4      A1
5      A1
6      A1
```

#### 💡 Aufgabe 4.5: `select()`

##### Beispiel 4.5.

1. Drucke die Spalten in `df_lexent`, die mit "t" beginnen
2. Drucke die Spalten in `df_lexent`, die "ge" enthalten
3. Drucke die Spalten in `df_lexent`, die
  - mit mit "r" beginnen, und
  - mit "s" enden

#### 4.4.4. `relocate()`

- `relocate()` verschiebt Variablen
  - standardmäßig werden sie nach vorne verschoben

```
df_lexent |> relocate(Trial) |>
  head()
```

|   | Trial | teilnehmer | rz_ms    | rz_s      | geschlect | NativeLanguage | Correct |
|---|-------|------------|----------|-----------|-----------|----------------|---------|
| 1 | 23    | A1         | 566.9998 | 0.5669998 | F         | English        | correct |

```

2    27      A1 548.9998 0.5489998      F      English correct
3    29      A1 572.0000 0.5720000      F      English correct
4    30      A1 486.0002 0.4860002      F      English correct
5    32      A1 414.0000 0.4140000      F      English correct
6    33      A1 483.0002 0.4830002      F      English correct

      Word Frequency Class laenge
1      owl 4.859812 animal      3
2      mole 4.605170 animal      4
3     cherry 4.997212 plant      6
4      pear 4.727388 plant      4
5      dog 7.667626 animal      3
6 blackberry 4.060443 plant     10

```

- aber wir können auch `.before` oder `.after` verwenden, um eine Variable zu platzieren

```

df_lexent |>
  relocate(Trial, .after = teilnehmer) |>
  head()

```

```

teilnehmer Trial    rz_ms      rz_s geschlect NativeLanguage Correct
1          A1    23 566.9998 0.5669998      F      English correct
2          A1    27 548.9998 0.5489998      F      English correct
3          A1    29 572.0000 0.5720000      F      English correct
4          A1    30 486.0002 0.4860002      F      English correct
5          A1    32 414.0000 0.4140000      F      English correct
6          A1    33 483.0002 0.4830002      F      English correct

      Word Frequency Class laenge
1      owl 4.859812 animal      3
2      mole 4.605170 animal      4
3     cherry 4.997212 plant      6
4      pear 4.727388 plant      4
5      dog 7.667626 animal      3
6 blackberry 4.060443 plant     10

```

## 4.5. dplyr und ggplot2

- wir können einen Datensatz mit den `dplyr`-Verben ändern und diese Änderungen dann in `ggplot2` einspeisen
- Was wird der folgende Code ergeben?

```

df_lexent |>
  ## filter the data
  filter(rz_ms > 120,
         rz_ms > 500) |>
  ## plot the filtered data
  ggplot(aes(x = fct_infreq(Correct))) +
  geom_bar() +
  theme_minimal()

```

#### 4.5.1. Pipe versus plus (|> vs. +)

- wichtig: wir können Pipes (|>) verwenden, um zusätzliche Verben/Funktionen mit dem Ergebnis einer vorherigen Codezeile auszuführen
  - Die Funktion `ggplot()` verwendet jedoch `+`, um neue *Ebenen* zur Darstellung hinzuzufügen

**!** **Rendernpause!**

Nehmen Sie sich einen Moment Zeit, um Ihr Dokument zu rendern. Wird es gerendert? Können Sie das Dokument besser strukturieren? Z. B. durch Hinzufügen von mehr Überschriften, Text?

## Aufgaben

1. Drucken Sie in einer einzigen Pipeline `df_lexent`, wobei Sie nur die Spalten Reaktionszeiten (in Millisekunden), NativeLanguage und Word für Zeilen auswählen, die jede der folgenden Bedingungen erfüllen, sie in der Reihenfolge der Reaktionszeiten anordnen und so filtern, dass nur diese Zeilen berücksichtigt werden:
  - die Reaktionszeiten waren größer als 500ms *und* kleiner als 550ms
  - aus den Wörtern “pear”, “elephant” oder “tortoise” stammen
2. Sortiere (`arrange()`) `df_lexent` in absteigender Reihenfolge, um die Versuche mit den längsten Reaktionszeiten zu finden.
3. Speichern Sie in einer einzigen Pipeline ein neues Objekt namens `df_rz`, das `df_lexent` enthält, *und dann*:
  - Selektieren (`select()`) Sie die Variablen `Teilnehmer`, `NativeLanguage`, `Word`, `rz_s`, `laenge`, und `Frequency`

- Erstelle eine neue Variable `rz_s_laenge` (`mutate()`), die `rz_s` geteilt durch `laenge` ist
  - und wird vor `Laenge` gesetzt
- Benennen (`rename()^``) Sie diese Variablen in Englisch um, so dass sie in Deutsch (und mit Kleinbuchstaben) sind.

## Heutige Ziele

Heute haben wir gelernt...

- wie man Daten mit dem Paket `dplyr` aus dem `tidyverse` verarbeitet
- wie man die `pipe` (`|>`) verwendet, um das Ergebnis einer Funktion in eine andere Funktion einzuspeisen
- über Funktionen, die auf Zeilen operieren
- über Funktionen, die auf Spalten operieren
- wie man `dplyr`-Funktionen mit Plots von `ggplot2` kombiniert

## Session Info

Hergestellt mit R version 4.3.0 (2023-04-21) (Already Tomorrow) und RStudioversion 2023.9.0.463 (Desert Sunflower).

```
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

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] languageR_1.5.0 lubridate_1.9.2forcats_1.0.0  stringr_1.5.0
[5] dplyr_1.1.3     purrr_1.0.2     readr_2.1.4     tidyverse_2.0.0
[9] tibble_3.2.1    ggplot2_3.4.3  tidyverse_2.0.0

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

```

## Literaturverzeichnis

- Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*.
- Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>
- Davies, R., Locke, S., & D'Agostino McGowan, L. (2022). *datasauRus: Datasets from the Datasaurus Dozen*. <https://CRAN.R-project.org/package=datasauRus>
- Müller, K. (2020). *here: A Simpler Way to Find Your Files*. <https://CRAN.R-project.org/package=here>
- Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills*. Zenodo. <https://doi.org/10.5281/zenodo.6365078>
- Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2022). Data Visualization Using R for Researchers Who Do Not Use R. *Advances in Methods and Practices in Psychological Science*, 5(2), 251524592210746. <https://doi.org/10.1177/25152459221074654>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. <https://doi.org/10.21105/joss.01686>
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2. Aufl.).
- Winter, B. (2019). Statistics for Linguists: An Introduction Using R. In *Statistics for Linguists: An Introduction Using R*. Routledge. <https://doi.org/10.4324/9781315165547>

Xie, Y. (2023). *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>

# 5. Datenvisualisierung 2

Visualisierung von Beziehungen

## Lesungen

Die **Pflichtlektüre** zur Vorbereitung auf dieses Thema ist [Kap. 2 \(Datenvisualisierung\)](#) aus [Abschnitt 2.5](#) in Wickham et al. (2023).

Eine **ergänzende Lektüre** ist [Ch. 3 \(Data visualtion\)](#) in Nordmann & DeBruine (2022).

## Wiederholung

Letzte Woche haben wir gelernt...

- wie man Daten mit dem Paket `dplyr` aus dem `tidyverse` verarbeitet
- gelernt, wie man die `pipe` (`|>`) verwendet, um das Ergebnis einer Funktion in eine andere Funktion einzuspeisen
- über Funktionen, die auf Zeilen operieren
  - `filter()`, `arrange()`
- über Funktionen, die auf Spalten operieren
  - `rename()`, `mutate()`, `select()`, `relocate()`
- wie man `dplyr`-Funktionen mit Plots von `ggplot2` kombiniert

## Lernziele

Heute werden wir lernen...

- wie man zwei oder mehr Variablen darstellt
  - mit Ästhetik und mit Facettenrastern
- wie man Codechunk-Optionen verwendet
- wie man Plots als Dateien speichert

## Lesungen

Die **Pflichtlektüre** zur Vorbereitung auf dieses Thema ist [Kap. 2 \(Datenvisualisierung\)](#) aus [Abschnitt 2.5](#) in Wickham et al. (2023).

Eine **ergänzende Lektüre** ist [Ch. 3 \(Data visualtion\)](#) in Nordmann & DeBruine (2022).

## Set-up

### Packages

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

- **tidyverse** Familie von Paketen
  - **ggplot2** für Diagramme
  - **dplyr** für die Datenverarbeitung
- **ggthemes** für farbenblindfreundliche Farbpaletten
- **patchwork** für Plot-Layouts
- **languageR** für linguistische Datensätze

### ggplot theme

Ich habe mein bevorzugtes **ggplot**-Thema global festgelegt. Das bedeutet, dass nach dem Ausführen dieses Codes alle Diagramme dieses Thema verwenden werden.

Tabelle 5.1.: english dataset variables of interest

| variable         | description   |
|------------------|---|
| RTlexdec         | Reaktionszeiten für eine visuelle lexikalische Entscheidung (Millisekunden)   |
| RTnaming         | Reaktionszeiten für den Beginn einer verbalen Wortbenennungsaufgabe (Millisekunden)                                       |
| WrittenFrequency | numerischer Vektor mit der logarithmischen Häufigkeit in der lexikalischen Datenbank von Baayen & Shafaei-Bajestan (2019) |
| Wort             | ein Faktor mit 2284 Wörtern   |
| AgeSubject       | ein Faktor mit der Altersgruppe des Probanden als Level: jung versus alt  |
| WordCategory     | ein Faktor mit den Wortkategorien N (Substantiv) und V (Verb) als Ebenen  |
| CV               | Faktor, der angibt, ob das Anfangsphonem des Wortes ein Konsonant (C) oder ein Vokal (V) ist                              |
| CorrectLexdec    | numerischer Vektor mit dem Anteil der Probanden, die das Item bei der lexikalischen Entscheidung korrekt bearbeitet haben |

```
theme_set(theme_bw())
```

## Data

Wir verwenden den `english`-Datensatz aus dem Baayen & Shafaei-Bajestan (2019).

- enthält Daten aus einer lexikalischen Entscheidungsaufgabe in Englisch
- Die logarithmisch transformierten Reaktionszeiten werden zurücktransformiert, so dass sie in Millisekunden angegeben werden.
  - Wir verwenden dazu die Funktion `exp()`.

```
df_english <-  
  english |>  
  mutate(RTlexdec = exp(RTlexdec),  
         RTnaming = exp(RTnaming))
```

## english dataset

Unsere Variablen von Interesse sind:

## Hypotheses

- Welche Arten von Hypothesen könnten Sie für solche Daten aufstellen?
  - Unsere Reaktionszeitdaten sind unsere *Messvariablen*.
    - \* d.h. das, was wir messen
  - Alle anderen Variablen sind mögliche *Vorhersagevariablen*.

- \* d.h. wir könnten vorhersagen, dass ihr Wert unsere Messvariablen beeinflussen würde
- Welche Auswirkung (wenn überhaupt) könnte zum Beispiel die Worthäufigkeit auf die Reaktionszeiten bei lexikalischen Entscheidungsaufgaben haben? auf die Benennungszeiten?
  - Wie sieht es mit Unterschieden in den Reaktionszeiten zwischen jüngeren und älteren Teilnehmern aus?
- Welchen Effekt (wenn überhaupt) könnte die Wortkategorie auf die Reaktionszeiten haben?

## 5.1. Datenvisualisierung

- Die Visualisierung unserer Daten hilft uns, die Beziehung zwischen den Variablen zu veranschaulichen, um eine Geschichte zu erzählen.
- In der Regel visualisieren wir Variablen, für die wir eine bestimmte Hypothese haben: Prädiktor- und Messvariable(n)

### 5.1.1. Visualisierung von Verteilungen

- Histogramme, Dichtediagramme und Balkendiagramme für Zählwerte visualisieren die *Verteilung* von Beobachtungen
  - Sie geben Aufschluss darüber, wie oft wir bestimmte Werte einer Variablen beobachtet haben.
  - In der Regel tun wir dies, um ein Gefühl dafür zu bekommen, wie unsere Daten aussehen
- \* Was ist der Bereich unserer Daten, der Modus, die Gesamtverteilung der Werte?

 Aufgabe: Beziehungen visualisieren

1. Erstellen Sie ein Diagramm, das die Verteilung der Häufigkeit der geschriebenen Wörter visualisiert.
2. Erstellen Sie ein Diagramm, das die Verteilung von Substantiven und Verben visualisiert.

## 5.2. Visualisierung von Beziehungen

- Um Beziehungen zwischen Variablen zu visualisieren, müssen wir mindestens zwei Variablen auf die Ästhetik eines Diagramms abbilden
- Wir haben dies bereits getan, indem wir Farbe oder Füllung einer kategorischen Variable zugeordnet haben, während wir eine
  - eine kontinuierliche Variable auf die x-Achse für Histogramme/Dichte-Diagramme, oder
  - eine kategoriale Variable auf die y-Achse für ein Balkendiagramm

### Aufgabe: Visualisierung von Beziehungen in Verteilungen

1. Fügen Sie den soeben erstellten Diagrammen eine weitere Ästhetik hinzu, um sie darzustellen:

- die Verteilung der WrittenFrequency-Werte für Wörter mit Anfangskonsonanten und Vokalen
- die Verteilung der Substantive und Verben für Wörter mit Anfangskonsonanten und Vokalen

### 5.2.1. Gruppierte kontinuierliche Variable

- Unsere Histogramme, Dichtediagramme und Balkendiagramme zeigen die Verteilung der Werte einer *kontinuierlichen* Variable nach verschiedenen Stufen einer *kategorischen* Variable

#### 5.2.1.1. Gestapelt

- Beachten Sie, dass diese Kategorien standardmäßig übereinander gestapelt sind.

#### 5.2.1.2. Dodged (Ausgewicke)

- aber dass wir sie nebeneinander haben können, indem wir `identity` auf `dodge` setzen
  - Ich finde, dass dies für Balkenplots nützlicher ist

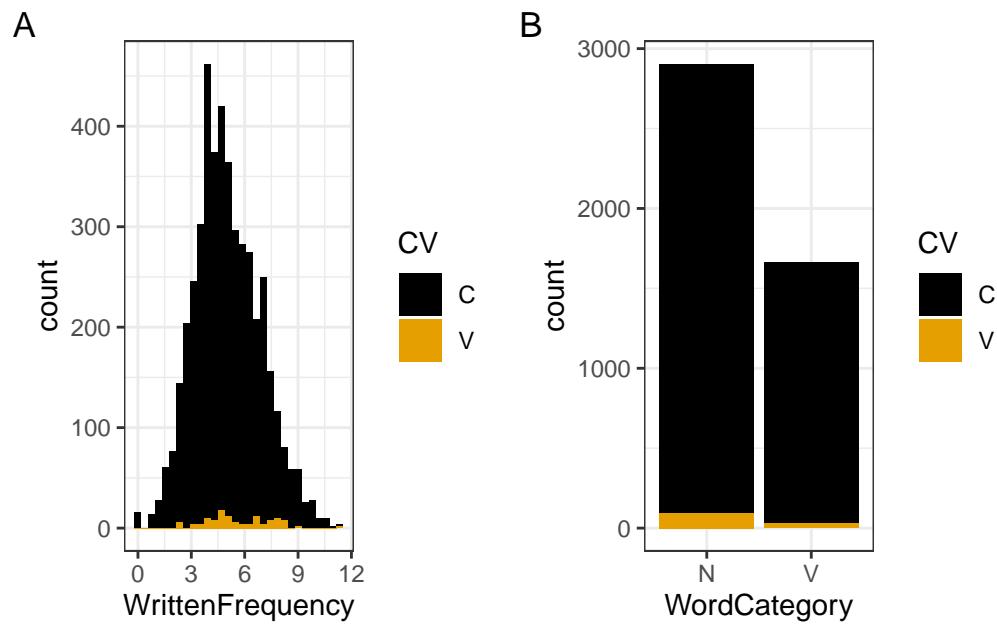


Abbildung 5.1.: Visualising relationships in distributions

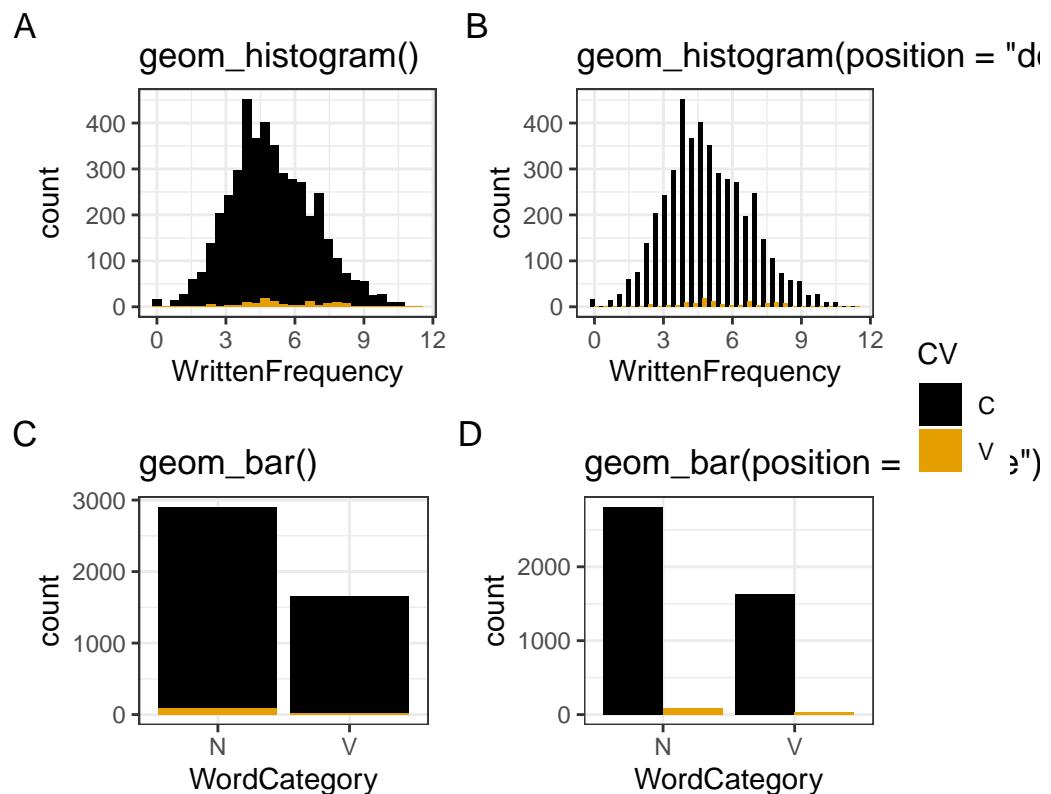
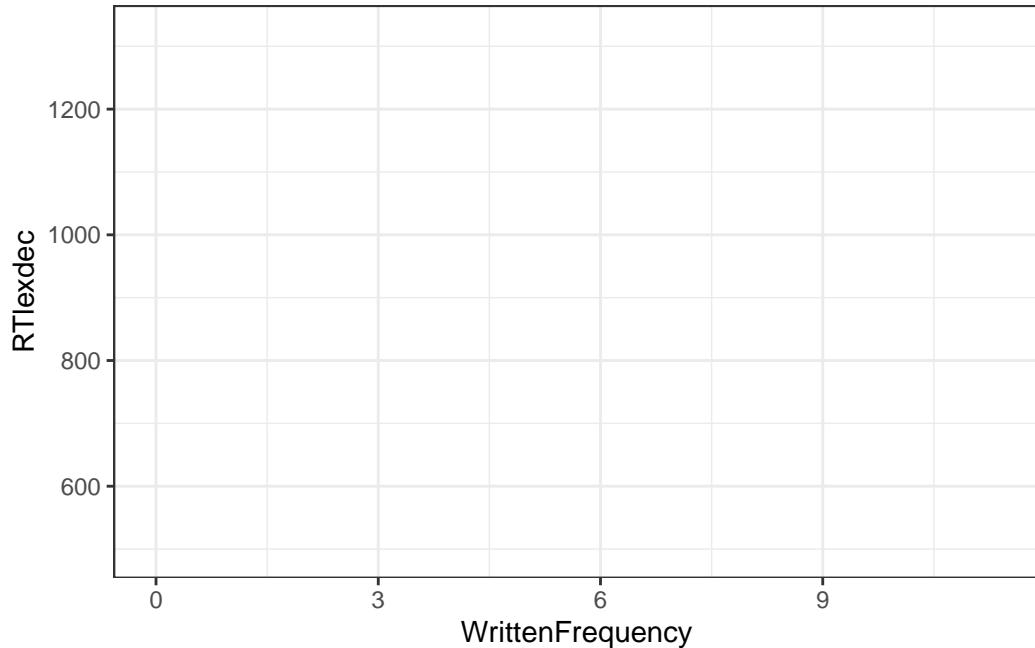


Abbildung 5.2.: Visualising relationships in distributions

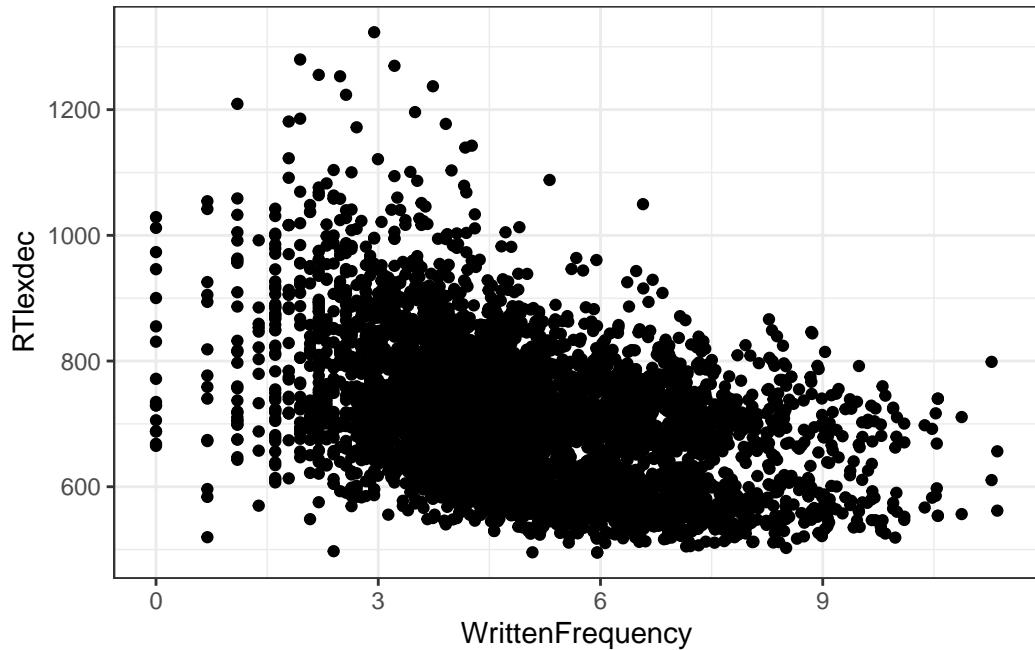
### 5.2.2. Zwei kontinuierliche Variablen

- Wir wollen oft die Auswirkungen einer kontinuierlichen Variable auf eine andere sehen.
- In unserem Datensatz `english` haben wir zum Beispiel die Variablen `WrittenFrequency` und `RTlexdec`
  - Welche Art von Beziehung werden diese beiden Variablen Ihrer Meinung nach haben?
  - Denken Sie z.B., dass Wörter mit einer niedrigeren WrittenFrequency in einer lexikalischen Entscheidungsaufgabe tendenziell längere oder kürzere Reaktionszeiten haben werden?
  - Wie könnte man sich eine solche Beziehung vorstellen?

```
## + geom_?
df_english |>
  ggplot() +
  aes(x = WrittenFrequency, y = RTlexdec)
```



```
df_english |>
  ggplot() +
  aes(x = WrittenFrequency, y = RTlexdec) +
  geom_point()
```

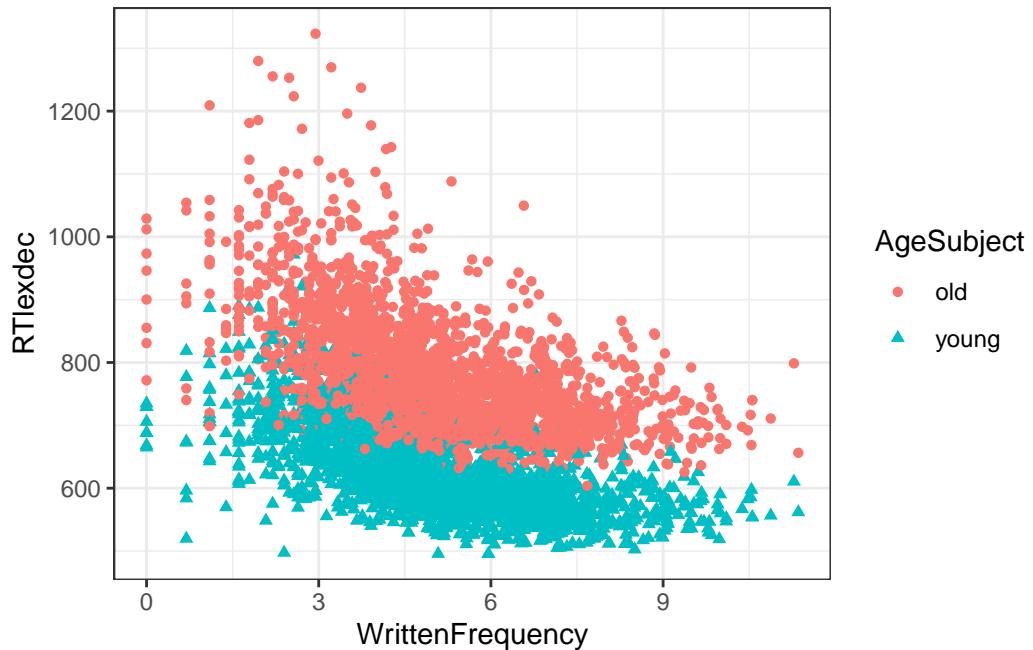


- Nehmen Sie sich einen Moment Zeit, um diese Grafik zu betrachten und eine Interpretation zu finden
  - Welchen Einfluss hat die Schrifthäufigkeit eines Wortes auf die Reaktionszeit bei einer lexikalischen Entscheidungsaufgabe?
  - Vervollständigen Sie den Satz: “Wörter mit einer höheren Worthäufigkeit lösten \_\_\_\_\_ Reaktionszeiten aus”
- Wo gab es mehr Variation in den Reaktionszeiten? Wo gab es weniger Variation?

### 5.2.3. Hinzufügen weiterer Variablen

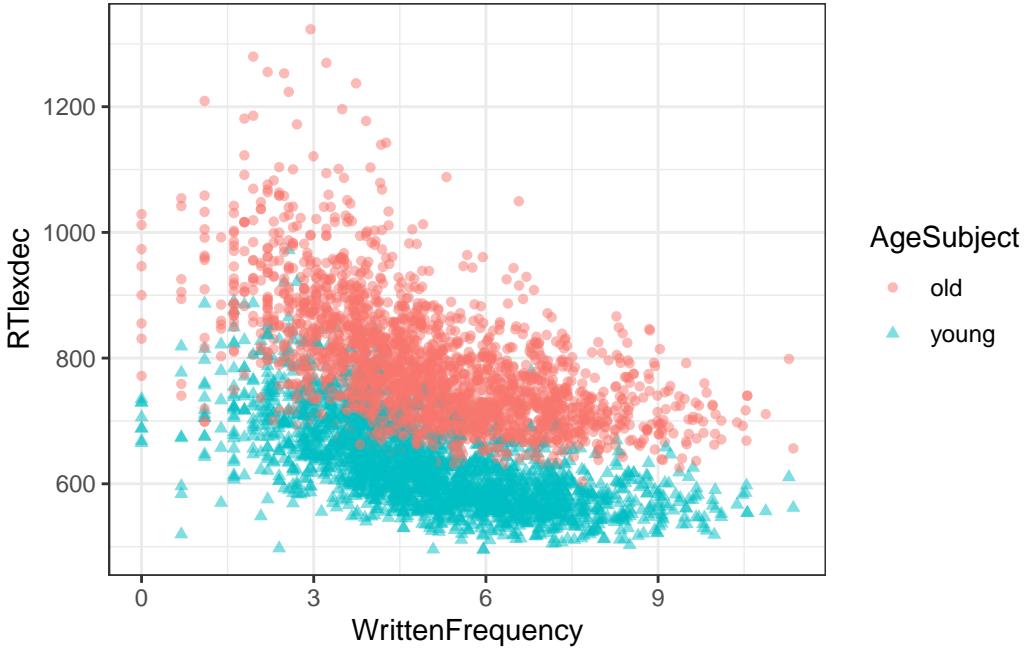
- Erinnern Sie sich daran, dass wir andere Ästhetiken wie `fill` oder `colour` verwenden können
  - für `geom_point()` ist es auch hilfreich, `shape` zu verwenden

```
df_english |>
  ggplot() +
  aes(x = WrittenFrequency, y = RTlexdec,
      colour = AgeSubject,
      shape = AgeSubject) +
  geom_point()
```



- In der Mitte des Diagramms gibt es viele Überschneidungen.
  - Wie können wir die Deckkraft der Punkte ändern?

```
df_english |>
  ggplot() +
  aes(x = WrittenFrequency, y = RTlexdec,
      colour = AgeSubject,
      shape = AgeSubject) +
  geom_point(alpha = .5)
```



- den Zusammenhang zwischen Altersgruppe und Reaktionszeit beschreiben

Aufgabe 5.1: Adding another variable

#### Beispiel 5.1.

Wie könnten Sie eine vierte Variable in die obige Darstellung einfügen? Versuchen Sie, CV hinzuzufügen. Ergibt die Darstellung immer noch eine klare Geschichte?

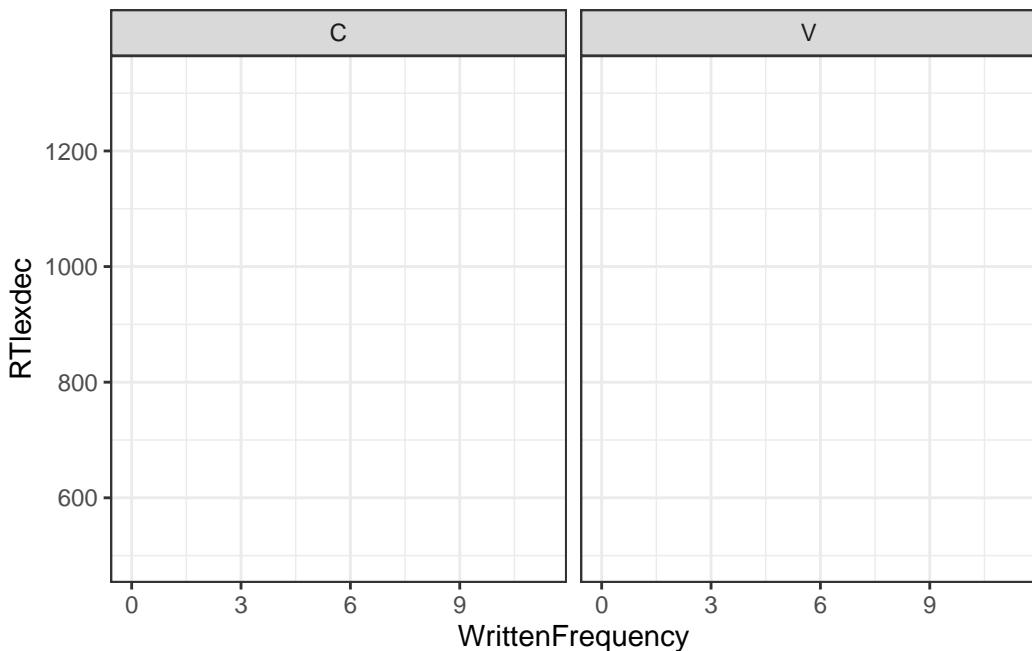
#### 5.2.4. Facet grids

- Wenn Sie mehr als drei Variablen darstellen wollen, ist es im Allgemeinen eine gute Idee, kategorische Variablen in *Facetten* aufzuteilen.
  - Facetten sind Teilplots, die Teilmengen der Daten anzeigen
- wir können `facet_wrap()` verwenden, das eine Formel als Argument annimmt
  - Diese Formel enthält `~` und den Namen einer kategorialen Variable, z. B. `~CV`

```

1 ## + geom_?
2 df_english |>
3   ggplot() +
4   aes(x = WrittenFrequency, y = RTlexdec,
5       colour = AgeSubject,
6       shape = AgeSubject) +
7   facet_wrap(~CV)

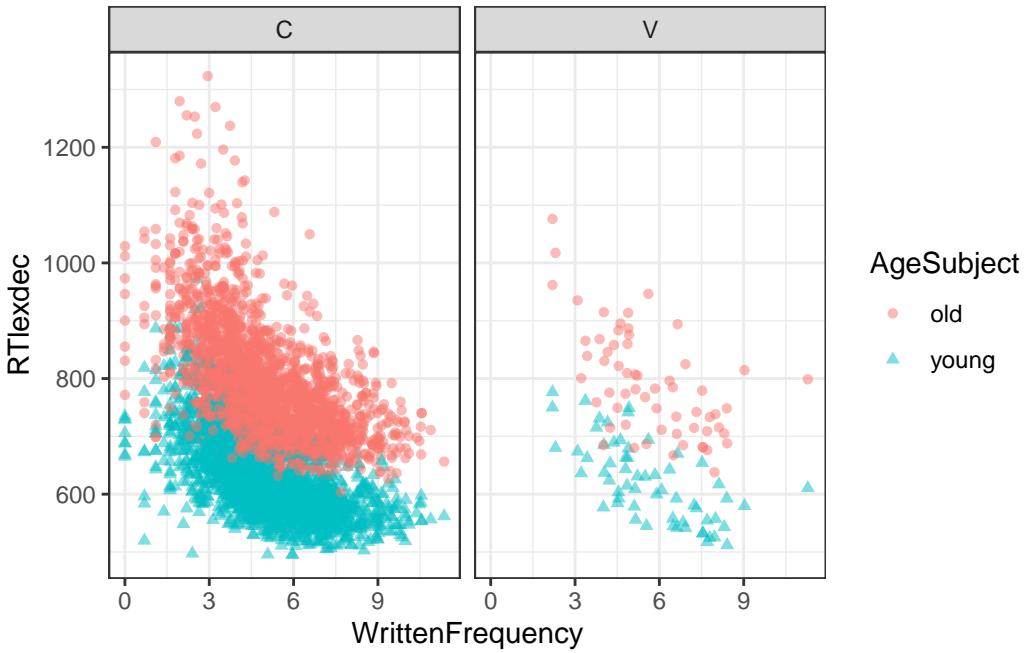
```



```

1 df_english |>
2   ggplot() +
3   aes(x = WrittenFrequency, y = RTlexdec,
4       colour = AgeSubject,
5       shape = AgeSubject) +
6   facet_wrap(~CV) +
7   geom_point(alpha = .5)

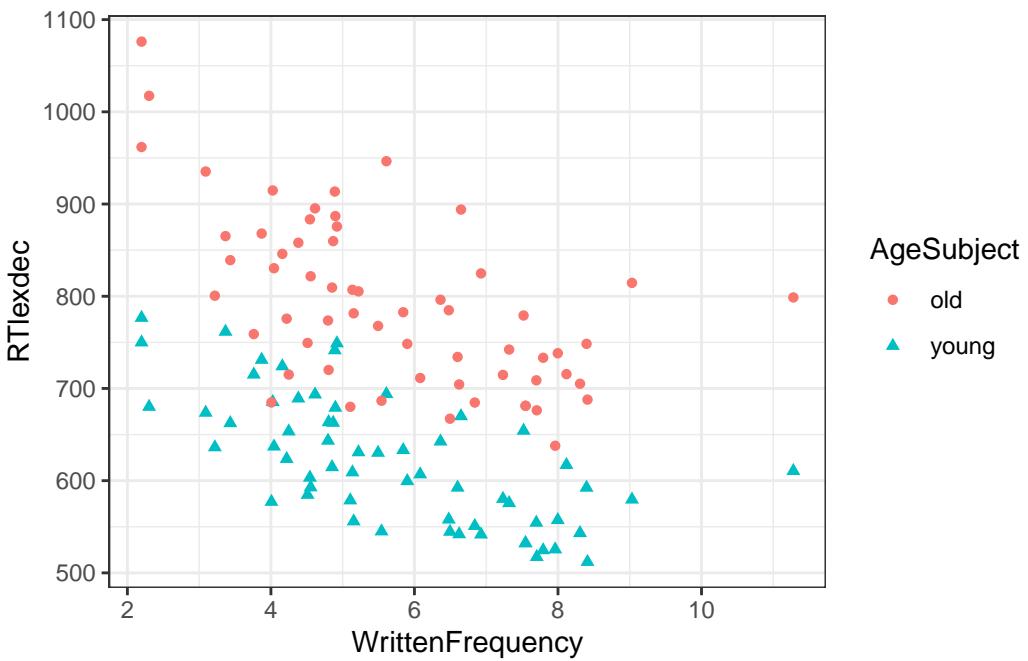
```



### 5.3. Bearbeitete Daten

- Wir können unsere Daten auch bearbeiten, bevor wir sie in `ggplot()` eingeben.
  - Dies ist nützlich, wenn wir keine permanenten Änderungen an den Daten vornehmen wollen, sondern nur eine Teilmenge der Daten darstellen wollen
- Vielleicht wollen wir nur die Wörter betrachten, die mit einem Vokal beginnen. Wie könnten wir das mit einem `dplyr`-Verb machen?

```
df_english |>
  filter(CV == "V") |>
  ggplot() +
  aes(x = WrittenFrequency, y = RTlexdec,
      colour = AgeSubject,
      shape = AgeSubject) +
  geom_point()
```



### 💡 Aufgabe 5.2: Plot-Anmerkung

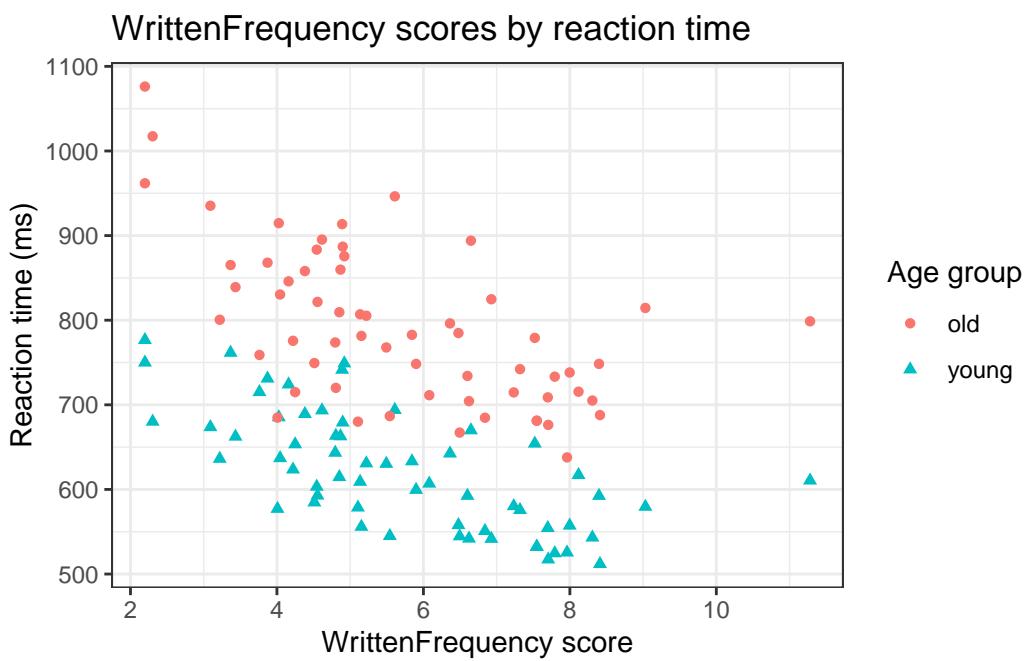
#### Beispiel 5.2.

- Vergessen Sie nicht, Ihre Diagramme mit nützlichen Beschriftungen zu versehen, um dem Leser die Interpretation des Diagramms zu erleichtern
- Fügen wir einen Titel und Beschriftungen für die x- und y-Achse hinzu

```
df_english |>
  filter(CV == "V") |>
  ggplot() +
  aes(x = WrittenFrequency, y = RTlexdec,
      colour = AgeSubject,
      shape = AgeSubject) +
  labs(title = "WrittenFrequency scores by reaction time",
       x = "WrittenFrequency score",
       y = "Reaction time (ms)",
       colour = "Age group",
       shape = "Age group") +
  geom_point()
```

Tabelle 5.2.: Most common chunk options

| option   | values     | function  |
|----------|------------|---|
| #  echo: | true/false | should this code chunk be printed when rendering? |
| #  eval: | true/false | should this code chunk be run when rendering?     |



## 5.4. Quarto Code Chunk Einstellungen

- lange Codeabschnitte können zu sehr unübersichtlichen Ausgabedokumenten führen
- normalerweise ist nur die Darstellung für den Leser wichtig, nicht der Code, der sie erzeugt hat
- wir können die Darstellung und Auswertung von Code Chunks durch Code Chunk Optionen steuern
  - diese beginnen mit #|
  - und befinden sich direkt unter `~~{r}~~`
- wichtige Code-Chunk-Optionen:

### 5.4.1. Verwendung von Code-Bausteinen

- warum sehen wir das Ergebnis dieser Darstellung nicht?

```
```{r}
#| eval: false
df_english |>
  ggplot() +
  aes(x = RTlexdec, y = RTnaming,
      colour = AgeSubject,
      shape = AgeSubject) +
  geom_point()
```
```

## 5.5. Plots speichern

- oft wollen wir unsere Plots in einem Dokument verwenden, das nicht in RStudio erstellt wurde
  - zum Beispiel in einer Dissertation oder einem in LaTeX geschriebenen Papier
- um dies zu tun, müssen wir unsere Zahlen als einen akzeptierten Dateityp laden, wie `jpeg` oder `png`
- Das können wir mit der Funktion `ggsave()` machen.
- Können Sie erraten, welche Arten von Argumenten `ggsave()` benötigt, um unsere Diagramme zu speichern? Einige sind erforderlich, einige sind optional.

### 5.5.1. `ggsave()`

Als Minimum benötigt `ggsave()` Argumente:

1. den Namen des Plots in Ihrer Umgebung, den Sie speichern möchten
2. den Dateinamen, unter dem Sie Ihre Darstellung speichern möchten
  - Es ist eine gute Idee, einen Ordner zu erstellen, in dem Sie Ihre Plots speichern, und den Dateipfad in den Namen aufzunehmen

### 5.5.1.1. ggsave() optionale Argumente

- einige optionale Argumente sind:
  - `width` = wie breit soll der Plot in cm, mm, Zoll oder Pixel sein?
  - `height` = wie hoch soll der gespeichert Plot in cm, mm, Zoll oder Pixel sein?
  - `dpi` = gewünschte Auflösung (numerisch, oder eine Reihe von Strings: “retina” = 320, “print” = 300 oder “screen” = 72)

#### ⚠ Warnung

Setzen Sie Code-Chunks, die Dateien auf Ihrem Rechner speichern, *immer* auf `eval: false!!!` Andernfalls wird jedes Mal, wenn Sie Ihr Skript ausführen, die Datei lokal neu geschrieben.

#### 💡 Aufgabe 5.3: ggsave()

#### Beispiel 5.3.

1. Kopieren Sie den unten stehenden Code in einen Codechunk und führen Sie ihn aus. Schauen Sie sich Ihre “Files”-Tab an, was hat sich geändert?

```
```{r}
#| eval: false
ggsave(
  ## required:
  "figures/04-dataviz2/fig_lexdec_rt.png",
  plot = fig_lexdec_rt,
  ## optional:
  width = 2000,
  height = 1000,
  units = "px",
  scale = 1,
  dpi = "print")
````
```

2. Versuchen Sie, mit dem Maßstab und den dpi zu spielen. Was ändert sich?
3. Versuchen Sie, die Werte für Einheiten, Breite und Höhe zu ändern. Was ändert sich?

## 5.6. Übungen

1. a. Zeichnen Sie abweichende Balkenplots von `AgeSubject` (x-Achse) nach `CV` (Facetten).  
b. Ändern Sie Ihre Code-Chunk-Optionen für den letzten Plot so, dass der Code, aber nicht der Plot, in der Ausgabe gedruckt wird.
2. a. Filtern Sie die Daten, um nur ältere Teilnehmer einzuschließen, und stellen Sie `RTlexdec` (x-Achse) durch `RTnaming` (y-Achse) dar. Übertragen Sie `CV` auf Farbe und Form. Fügen Sie geeignete Beschriftungen hinzu.  
b. Ändern Sie die Code-Chunk-Optionen für den letzten Plot so, dass der Plot, aber nicht der Code, in der Ausgabe gedruckt wird.
3. Speichern Sie den letzten Plot lokal und stellen Sie den Code Chunk so ein, dass er beim Rendern *nicht* ausgeführt wird.

## Session Info

Hergestellt mit R version 4.3.0 (2023-04-21) (Already Tomorrow) und RStudioversion 2023.9.0.463 (Desert Sunflower).

```
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

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] kableExtra_1.3.4 knitr_1.44      languageR_1.5.0 ggthemes_4.2.4
[5] patchwork_1.1.3 lubridate_1.9.2 forcats_1.0.0  stringr_1.5.0
[9] dplyr_1.1.3     purrr_1.0.2    readr_2.1.4   tidyverse_2.0.0
[13] tibble_3.2.1   ggplot2_3.4.3  tidyverse_2.0.0

loaded via a namespace (and not attached):
[1] utf8_1.2.3          generics_0.1.3    xml2_1.3.4       stringi_1.7.12
[5] hms_1.1.3           digest_0.6.33     magrittr_2.0.3   evaluate_0.21
[9] grid_4.3.0          timechange_0.2.0  fastmap_1.1.1   jsonlite_1.8.7
[13] httr_1.4.6          rvest_1.0.3      fansi_1.0.4     viridisLite_0.4.2
[17] scales_1.2.1        cli_3.6.1       rlang_1.1.1     munsell_0.5.0
[21] withr_2.5.0         yaml_2.3.7      tools_4.3.0     tzdb_0.4.0
[25] colorspace_2.1-0   webshot_0.5.4   pacman_0.5.1    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       systemfonts_1.0.4 xfun_0.39
[37] tidyselect_1.2.0    rstudioapi_0.14  farver_2.1.1    htmltools_0.5.5
[41] labeling_0.4.3      rmarkdown_2.22   svglite_2.1.1   compiler_4.3.0

```

## Literaturverzeichnis

- Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*.
- Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>
- Davies, R., Locke, S., & D'Agostino McGowan, L. (2022). *datasauRus: Datasets from the Datasaurus Dozen*. <https://CRAN.R-project.org/package=datasauRus>
- Müller, K. (2020). *here: A Simpler Way to Find Your Files*. <https://CRAN.R-project.org/package=here>
- Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills*. Zenodo. <https://doi.org/10.5281/zenodo.6365078>
- Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2022). Data Visualization Using R for Researchers Who Do Not Use R. *Advances in Methods and Practices in Psychological Science*, 5(2), 251524592210746. <https://doi.org/10.1177/25152459221074654>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. <https://doi.org/10.21105/joss.01686>
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2. Aufl.).
- Winter, B. (2019). Statistics for Linguists: An Introduction Using R. In *Statistics for Linguists: An Introduction Using R*. Routledge. <https://doi.org/10.4324/9781315165547>

Xie, Y. (2023). *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>

# 6. Bericht 1

Konsolidierung der neuen Kenntnisse

Dieser Bericht dient dazu, das bisher Gelernte zu wiederholen und zu festigen. Ihre Aufgaben umfassen das Laden von Paketen und Daten sowie eine leichte Datenverarbeitung (Kapitel 6.2). Außerdem werden Sie 4 Diagramme erstellen (Kapitel 6.3) und eine kurze Interpretation zu einem der Diagramme schreiben (Kapitel 6.4).

Ein Tipp: Ich empfehle Ihnen, Ihr Dokument häufig zu rendern, um eventuelle Fehler frühzeitig zu erkennen.

Sie müssen nur das Quarto-Skript einreichen, das auf meinem Rechner gerendert werden sollte (wenn es auf Ihrem gerendert wird, sollte es auch auf meinem gerendert werden).

## 6.1. Einrichtung

### 6.1.1. Quarto

Öffnen Sie ein neues Quarto-Skript und speichern Sie es als `nachname_vorname_bericht1.qmd`. Ändern Sie das YAML so, dass es einen:

- einen aussagekräftigen Titel
- Ihren Namen als `Autor`
- ein Inhaltsverzeichnis

Achten Sie darauf, Code Chunks, Prosa und Überschriften zu verwenden, um Ihre Aufgaben angemessen zu dokumentieren. Eine gute Faustregel ist, für jede (Unter-)Überschrift in diesem Dokument eine Überschrift hinzuzufügen.

### 6.1.2. Pakete

Laden Sie die Pakete `tidyverse` und `languageR` ein.

Tabelle 6.1.: ?(caption)

|                     | (a)  |
|---------------------|--|
| Variable            | Beschreibung   |
| Word                | ein Faktor mit den Wörtern als Ebenen  |
| Frequency           | ein numerischer Vektor mit der absoluten Häufigkeit des Wortes im Spoken Dutch Corpus          |
| Speaker             | ein numerischer Vektor mit der absoluten Häufigkeit des Wortes im Spoken Dutch Corpus          |
| Sex                 | ein Faktor mit den Lautsprechern als Ebenen  |
| YearOfBirth         | ein numerischer Vektor mit Geburtsjahren   |
| DurationOfPrefix    | ein numerischer Vektor mit der Dauer des Präfixes -ge in Sekunden.                             |
| SpeechRate          | ein numerischer Vektor, der die Sprechgeschwindigkeit in Anzahl der Silben pro Sekunde kodiert |
| NumberSegmentsOnset | ein numerischer Vektor, der die Sprechgeschwindigkeit in Anzahl der Silben pro Sekunde kodiert |

### 6.1.3. Daten

Der Datensatz `durationsGe` aus dem languageR-Paket (Baayen & Shafaei-Bajestan, 2019) enthält Dauermessungen zur niederländischen Vorsilbe *ge*. Eine Beschreibung aller Variablen des Datensatzes findet sich in Tabelle 6.1. Ihre Aufgabe ist es:

1. Speichern Sie den Datensatz als Objekt `df_ge` in Ihrer Umgebung (dies kann auf die gleiche Weise geschehen wie bei allen Datensätzen, die wir bisher verwendet haben)
2. Drucken Sie die ersten 10 Zeilen des Datensatzes mit der Funktion “`head()`” aus.

## 6.2. Data wrangling

Hier werden Sie die `dplyr`-Verben aus Woche 4 verwenden. Denken Sie daran, dass Sie den Zuweisungsoperator (`<-`) nur verwenden müssen, wenn Sie die Änderungen, die Sie vornehmen, als Objekt in der Umgebung speichern wollen. Wenn Sie diese Änderungen nur ausdrucken wollen, brauchen Sie den Zuweisungsoperator nicht.

### 6.2.1. Subsetting

Drucken (aber nicht in Ihrer Umgebung speichern) Sie die Zeilen von `df_ge`, in denen `SpeechRate` über 9 liegt, nur mit den Spalten `word`, `speaker` und `SpeechRate`. Es sollten 5 Zeilen sein.

### 6.2.2. `mutate()`

Fügen Sie eine neue Variable hinzu, `duration_ms`, die `DauerVonPräfix` multipliziert mit 1000 (`DurationOfPrefix*1000`) entspricht. Dies entspricht der Dauer von `ge` in Millisekunden, statt in Sekunden. Stellen Sie sicher, dass Sie diese neue Variable in Ihrem Datenrahmen speichern (Hinweis: Sie müssen den Zuweisungsoperator `<-` und das `dplyr`-Verb `mutate()` verwenden).

### 6.2.3. Fehlersuche

Warum läuft dieser Code nicht? Es gibt zwei Probleme mit dem Code, identifizieren und beheben Sie sie.

```
## Troubleshooting
df_ge |>
  filter(YearOfBirth == 1978) +
  select(Frequency, word)
```

## 6.3. Datenvisualisierung

Verwenden Sie für alle Diagramme `labs(title = "...")`, um entsprechende Diagrammtitel hinzuzufügen.

*Optional:* Ändern Sie die `x` und `y` Achsenbeschriftungen, wenn Sie wollen, mit `labs(x = "...", y = "...")`. Vielleicht möchten Sie auch ein Thema hinzufügen (z.B. `theme_minimal()`).

### 6.3.1. Scatterplot

Erstellen Sie ein Streudiagramm mit `SpeechRate` (x-Achse) und `DurationOfPrefix` (y-Achse), mit `YearOfBirth` als Farbe (`colour`). Ändern Sie die Einstellungen für den Codechunk so, dass das Diagramm beim Rendern des Skripts *nicht* gedruckt wird, der Code aber schon. Tipp: Sie müssen `#| eval:` verwenden.

### 6.3.2. Facetten

Fügen Sie Facetten für `Sex` hinzu (denken Sie daran, die Tilde ~ einzufügen). Ändern Sie die Code-Chunk-Einstellungen so, dass die Darstellung gedruckt wird, wenn das Skript gerendert wird, aber der Code nicht (Sie benötigen `echo` anstelle von `eval`).

### 6.3.3. Reproduzieren eines Plots

Reproduzieren Sie die Abbildung 6.1 (es muss keine exakte Kopie sein, aber kommen Sie ihr so nahe wie möglich). Stellen Sie sicher, dass sowohl der Code als auch die Darstellung beim Rendern gedruckt werden. Hinweis: Sie müssen `filter()` sowohl für `Frequency` als auch für `Sex` verwenden. Ich würde mich darauf konzentrieren, zuerst das Diagramm zu erstellen und dann zu versuchen, die Daten zu filtern.

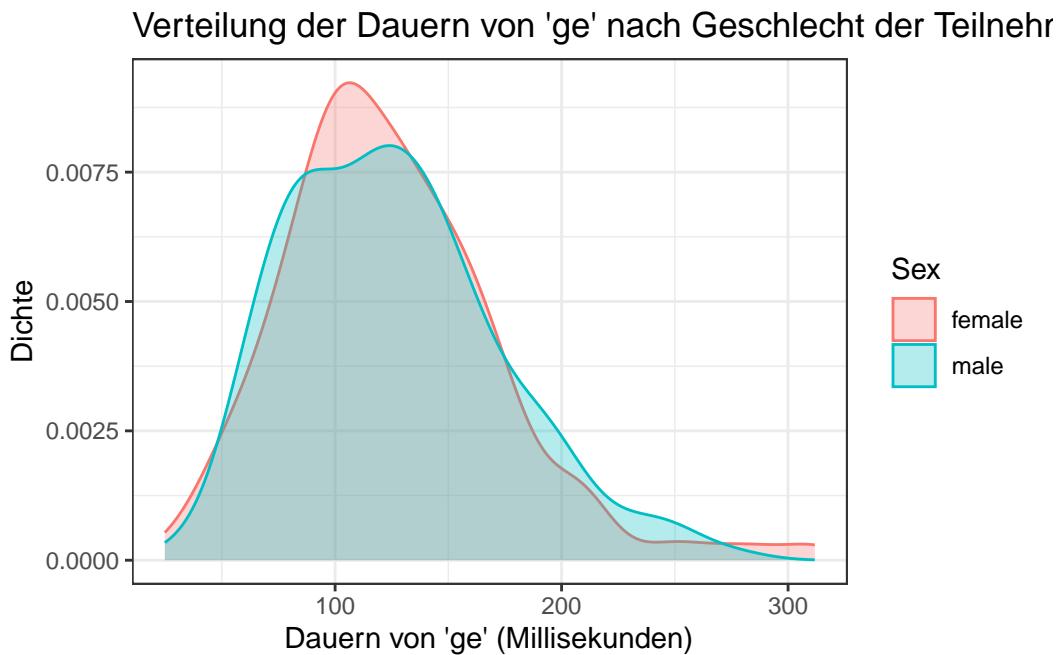


Abbildung 6.1.: Eine zu reproduzierende Figur

## 6.4. Interpretation

Beschreiben Sie die Beziehung zwischen den beiden Variablen, die Sie in Abbildung 6.1 sehen.

## **Teil III.**

# **Nächste Stufe**

# 7. Einlesen von Daten

Importieren von Datendateien

## Lernziele

In diesem Kapitel werden wir lernen, wie man:

- lokale Datendateien mit dem Paket `readr` zu importieren
- mit fehlenden Werten umzugehen
- Variablen in Faktoren umwandeln

## Lesungen

Die **Pflichtlektüre** zur Vorbereitung auf dieses Thema ist [Kap. 8 \(Data Import\)](#) in Wickham et al. (2023).

Eine **ergänzende Lektüre** ist [Ch. 4 \(Data Import\)](#) in Nordmann & DeBruine (2022).

## Wiederholung

Im letzten Kapitel haben wir das Gelernte in die Praxis umgesetzt. Wir haben einen Datensatz aus dem `languageR`-Paket (Baayen & Shafaei-Bajestan, 2019) eingelesen, ihn mit dem `dplyr`-Paket aus dem `tidyverse` verarbeitet und mehrere Diagramme mit dem `ggplot2`-Paket aus dem `tidyverse` erstellt. All dies wurde mit einem Quarto-Skript durchgeführt.

## 7.1. Einrichtung

### 7.1.1. Pakete mit pacman

Zu Beginn werden wir mit dem Paket `pacman` beginnen. Die Funktion `p_load()` nimmt Paketnamen als Argumente und prüft dann, ob Sie das Paket installiert haben. Wenn ja, dann lädt sie das Paket (genau wie `library()`). Wenn Sie das Paket nicht installiert haben, wird das Paket installiert und geladen (wie mit `install.packages()`, `library()`). Das erspart uns, neue Pakete einzeln zu installieren, und bedeutet auch, dass, wenn wir unser Skript mit anderen teilen, sie einfach `pacman::p_load()` ausführen können.

```
## install new packages IN THE CONSOLE!
install.packages("pacman")

## load packages
pacman::p_load(tidyverse, ## wrangling
               janitor, ## wrangling
               here ## relative file paths
)
```

Wir haben nun `tidyverse` geladen, und die neuen Pakete `janitor` und `here` installiert und geladen. Um mehr über diese Pakete herauszufinden, versuchen Sie `?janitor` und `?here` in der Konsole einzugeben.

### 7.1.2. RProjects

Bevor wir mit unserer ersten Datei beginnen, müssen wir sicherstellen, dass wir innerhalb unseres RProjekts arbeiten. Zu Beginn des Kurses haben Sie eine ZIP-Datei von [GitHub](#) ([https://github.com/daniela-palleschi/r4ling\\_student](https://github.com/daniela-palleschi/r4ling_student)) heruntergeladen, die einige Ordner und eine `.RProj`-Datei enthielt. Hoffentlich haben Sie bis jetzt *innerhalb* dieses RProjekts gearbeitet und Ihre Skripte im Ordner `notizen` gespeichert. Von nun an wird es notwendig sein, innerhalb dieses RProjekts zu arbeiten, damit wir immer auf unsere relevanten Datendateien zugreifen können, die in einem Ordner namens `daten` gespeichert werden sollten.

Um ein RProjekt zu öffnen, navigieren Sie einfach zu dem Ordner auf Ihrem Rechner und doppelklicken Sie auf die `.RProj`-Datei. Wenn Sie sich bereits in RStudio befinden, können Sie auch überprüfen, ob Sie im richtigen RProjekt arbeiten, indem Sie oben im Fenster nachsehen.

## Aufgabe

1. Überprüfen Sie, ob Sie tatsächlich in Ihrem RProjekt arbeiten. Wenn dies der Fall ist, sehen Sie `r4ling_student-main` oben in Ihrem RStudio ([?@fig-project](#)).
  - Wenn dies nicht der Fall ist ([?@fig-no-project](#)), können Sie zum RProjekt wechseln, indem Sie auf die Schaltfläche “Projekt” oben rechts im RStudio klicken ([?@fig-project-open](#); Hinweis: Die Screenshots stammen von einem Mac, auf einem Windows-Rechner sieht es etwas anders).
2. Fügen Sie Ihrem RProject-Ordner einen Ordner namens `daten` hinzu.

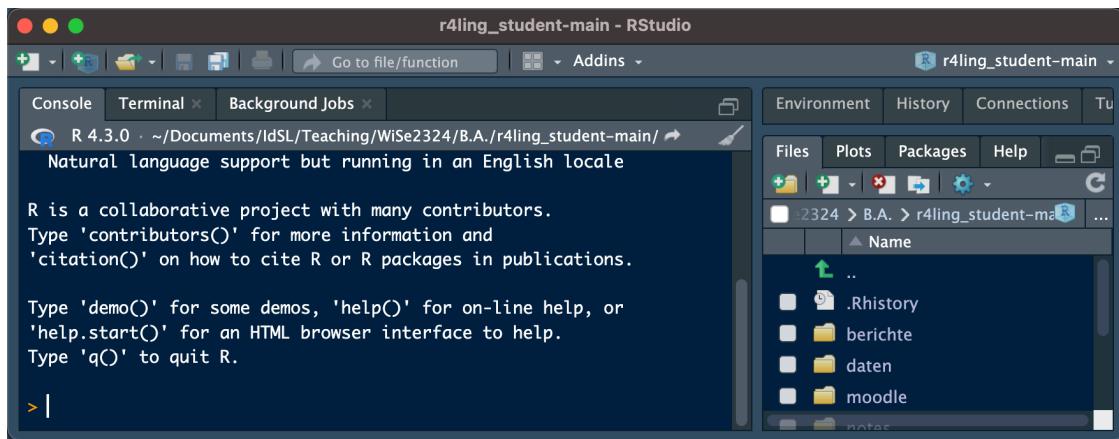


Abbildung 7.1.: RStudio will display the RProject name if you are working in an RProject.

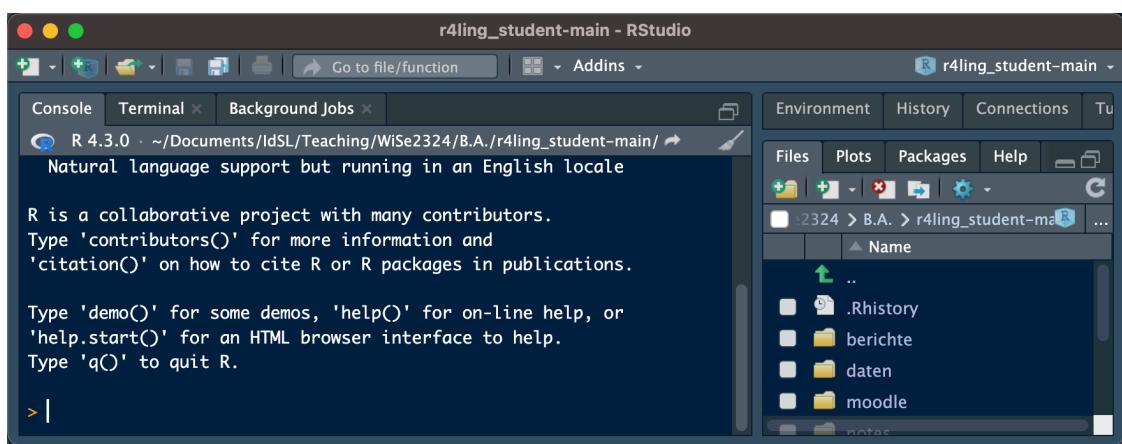


Abbildung 7.2.: RStudio will display the RProject name if you are working in an RProject.

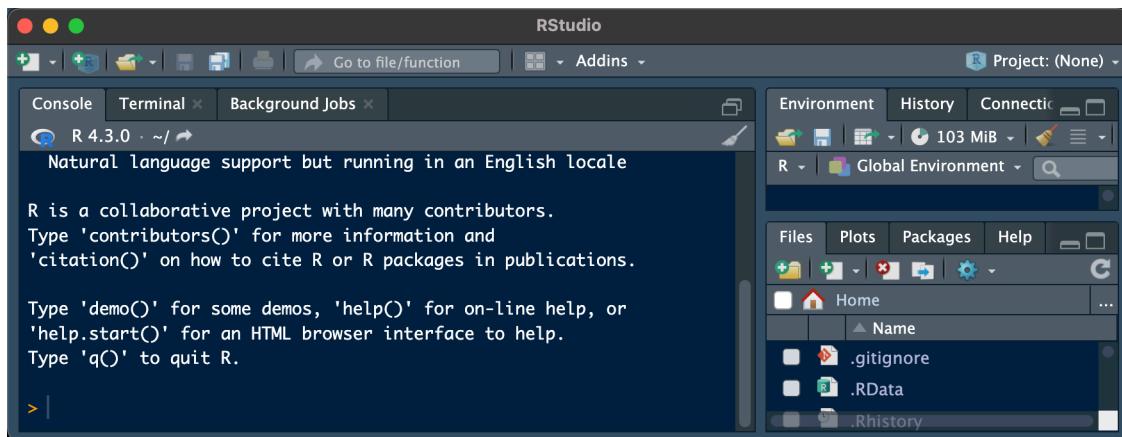


Abbildung 7.3.: RStudio will say 'RStudio' and 'Project (none)' if you are not working in an RProject.

## 7.2. CSV: Komma getrennter Wert

Bisher haben wir mit Daten aus dem R-Paket `languageR` gearbeitet. Daten aus Paketen sind eine großartige Möglichkeit, die Werkzeuge der Datenwissenschaft zu erlernen, aber normalerweise wollen wir mit unseren *eigenen* Daten arbeiten, nicht mit eingebauten Spielzeugdaten. Wir werden uns nur auf *rechteckige* Daten (d. h. aufgeräumte Daten) konzentrieren, obwohl Ihre Daten zu Beginn oft nicht aufgeräumt sind. Es gibt viele

verschiedene Dateitypen, die Daten annehmen können, z. B. .xlsx, .txt, .csv, .tsv. Der Dateityp "csv" ist der häufigste Dateityp und steht für: Comma Separated Values.

So sieht eine einfache CSV-Datei aus:

| Student ID | Full Name        | favourite.food     | mealPlan            | AGE  |
|------------|------------------|--------------------|---------------------|------|
| 1          | Sunil Huffmann   | Strawberry yoghurt | Lunch only          | 4    |
| 2          | Barclay Lynn     | French fries       | Lunch only          | 5    |
| 3          | Jayendra Lyne    | N/A                | Breakfast and lunch | 7    |
| 4          | Leon Rossini     | Anchovies          | Lunch only          |      |
| 5          | Chidiegwu Dunkel | Pizza              | Breakfast and lunch | five |
| 6          | Güvenç Attila    | Ice cream          | Lunch only          | 6    |

Die erste Zeile (die “Kopfzeile”) enthält die Spaltennamen. Die folgenden Zeilen enthalten die Daten. Wie viele Variablen gibt es? Wie viele Beobachtungen?

Wir lernen jetzt etwas über aufgeräumte Daten und sehen uns ein Beispiel an. Anschließend werden wir eine CSV-Datei in R laden.

! Microsoft Excel

Versuchen Sie, .xlsx-Dateien im Allgemeinen zu vermeiden, vor allem aber, wenn Sie Ihre Daten in R laden wollen. Der Grund dafür ist, dass Excel viele Formatierungsprobleme hat, die für R problematisch sind. Wenn Sie einen Excel-Datensatz haben, versuchen Sie, ihn als .csv zu speichern, bevor Sie ihn in R einlesen (**Datei > Speichern unter > Dateiformat > Komma-getrennter Wert**).

### 7.2.1. Tidydaten

Unabhängig davon, in welchem Format Ihre Daten vorliegen, sollten sie *aufgeräumt* sein. Das bedeutet erstens, dass die Daten rechteckig sein sollten und dass jede Spalte eine Variable, jede Zeile eine Beobachtung und jede Zelle einen Datenpunkt darstellt. ([?@fig-tidy-data](#)).

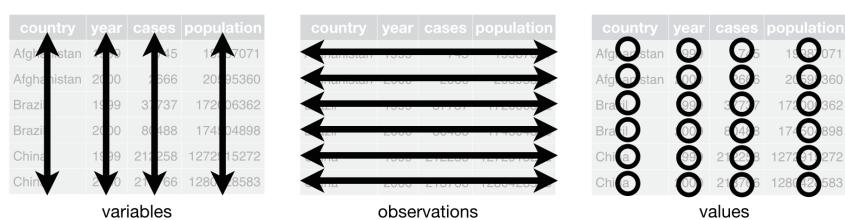


Abbildung 7.4.: Source: Wickham et al. (2023) (all rights reserved)

## 7.3. Tabelle zu csv

Lassen Sie uns einige Spielzeugdaten in einem Arbeitsblatt sammeln, das wir dann als CSV-Datei speichern und in R laden werden. [Klicken Sie hier](#), um zu einem bearbeitbaren Arbeitsblatt zu gelangen. Geben Sie die relevanten Informationen über sich selbst ein, oder erfinden Sie einige Daten: den Namen eines Haustiers, das Sie haben/hatten, Größe, Geburtsmonat und -tag sowie Ihre erste Sprache. Wenn Sie kein Haustier haben, lassen Sie die Zelle leer.

|   | A        | B     | C              | D   | E          |
|---|----------|-------|----------------|-----|------------|
| 1 | Haustier | Größe | Monat der Gebu | Tag | L1         |
| 2 | Lola     | 171   | 5              |     | 7 Englisch |
| 3 |          |       |                |     |            |
| 4 |          |       |                |     |            |

Abbildung 7.5.: Our spreadsheet

### 7.3.1. CSV speichern

Jetzt müssen wir unser Arbeitsblatt als CSV-Datei auf unserem Computer speichern. Solange wir in unserem RProjekt arbeiten, wird R immer nach Dateien aus dem Ordner suchen, der unser RProjekt enthält. Stellen wir also zunächst sicher, dass unser Ordner einen Unterordner namens **daten** enthält. Darin werden wir alle unsere Daten speichern.

#### Aufgabe 7.1: Speichern einer CSV

##### Beispiel 7.1.

1. Erstellen Sie einen neuen Ordner mit dem Namen **daten** in Ihrem Projektordner (falls Sie das nicht schon getan haben).
2. Laden Sie das Google Sheet herunter und speichern Sie es in Ihrem **daten**-Ordner als **groesse\_geburtstag.csv**.
3. Gehen Sie zu Ihrem **daten**-Ordner und überprüfen Sie, ob die CSV-Datei dort ist.

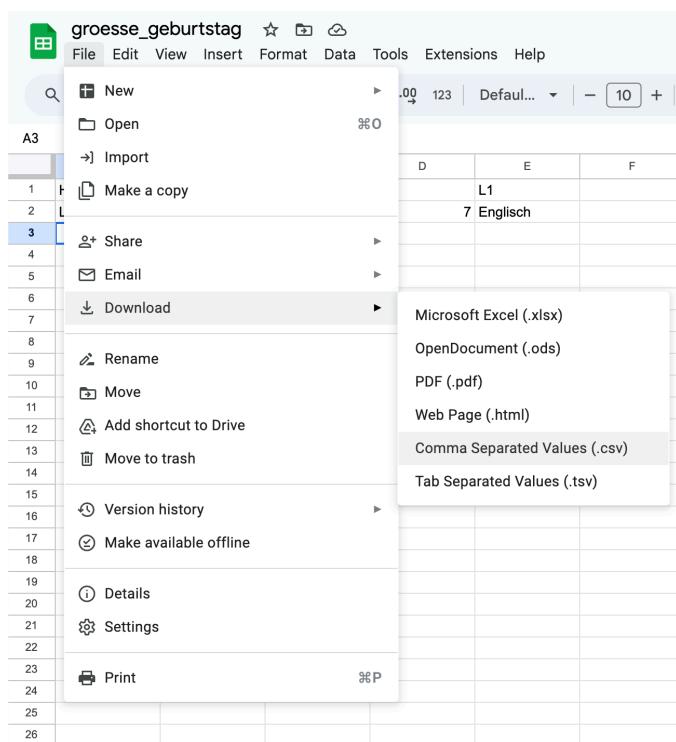


Abbildung 7.6.: Download a Google Sheet as a CSV.

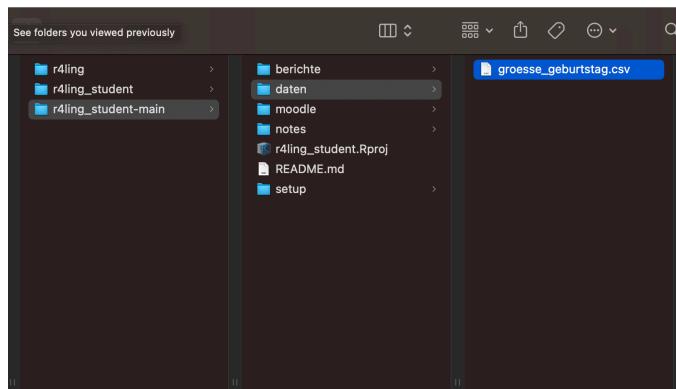


Abbildung 7.7.: Download a Google Sheet as a CSV.

## 7.4. Das `readr`-Paket

Unsere Daten können als Tabelle angezeigt werden, genau wie unsere eingebauten Datensätze aus dem `languageR`-Paket (Baayen & Shafaei-Bajestan, 2019). Genau wie bei den eingebauten Datensätzen müssen wir zuerst die Daten *einlesen*, aber anstatt nur den Namen des eingebauten Datensatzes anzugeben, müssen wir eine Funktion verwenden, die CSV-Daten liest. Wir müssen auch angeben, *wo* sich die Daten in unserem RProject-Ordner befinden.

Das Paket `readr` (Teil von `tidyverse`) kann die meisten Datentypen einlesen und hat mehrere Funktionen für verschiedene Datentypen.

```
read_csv(here::here("daten", "groesse_geburtstag.csv"))
```

Tabelle 7.1.: Data from the students.csv file as a table.

| Haustier | Größe | Monat der Geburt | Tag | L1         |
|----------|-------|------------------|-----|------------|
| Lola     | 171   |                  | 5   | 7 Englisch |
| NA       | 168   |                  | 11  | 26 Deutsch |
| N/A      | 182   |                  | 4   | 15 Deutsch |

### Aufgabe 7.2: df\_groesse

#### Beispiel 7.2.

1. Importieren Sie den Datensatz “groesse\_geburtstag.csv” und speichern Sie ihn als Objekt mit dem Namen “df\_groesse”.
  - `df_` ist die Abkürzung für DataFrame; es ist eine gute Idee, ein Präfix vor Objektnamen zu verwenden, damit wir wissen, was jedes Objekt enthält.
2. Wenn Daten mit `read_csv` importiert werden, werden einige Informationen in der Konsole ausgegeben. Was wird gedruckt?
3. Untersuche den Datensatz mit Funktionen wie `summary()` oder `head()`
4. Sehen Sie etwas Ungewöhnliches?

## 7.5. Das `here`-Paket

Aber woher weiß R genau, wo der Ordner `daten` zu finden ist? Unser *Arbeitsverzeichnis* ist auf den Ort unseres RProjekts auf unserem Computer festgelegt. Wann immer wir auf Daten in unserem RProjekt zugreifen wollen, sollten wir `here()` verwenden (vorausgesetzt, wir haben

das `here`-Paket bereits geladen). Um zu sehen, von wo aus `here()` startet, führen Sie `here()` aus. Wie sieht die Ausgabe im Vergleich zu der von `getwd()` (für ‘get working directory’)?

```
here()
```

```
[1] "/Users/danielapalleschi/Documents/IdSL/Teaching/WiSe2324/B.A./r4ling"
```

```
getwd()
```

```
[1] "/Users/danielapalleschi/Documents/IdSL/Teaching/WiSe2324/B.A./r4ling/mats"
```

Die Ausgabe wird auf allen unseren Rechnern anders aussehen, aber was gleich sein sollte, ist unsere Ordnerstruktur innerhalb unserer Projekte (z. B. `data/groesse_geburtstag.csv`).

#### `here`-Paket

Vor dem `here`-Paket mussten wir R explizit mitteilen, wo sich eine Datei auf unserem Computer befindet (z.B., `/Users/danielapalleschi/Documents/IdSL/Teaching/SoSe23/BA/ba_daten/daten/students.csv`), oder die Funktion `setwd()` (set Working Directory) benutzen, um R mitzuteilen, wo alle Dateien zu finden sind (z.B. `setwd("/Users/danielapalleschi/Documents/IdSL/Teaching/SoSe23/BA/ba_daten")`). Glücklicherweise brauchen Sie diese absoluten Dateipfade oder `setwd()` nie zu benutzen!

Aus der [hier-Paketdokumentation](#):

The goal of the `here` package is to enable easy file referencing in project-oriented workflows. In contrast to using `setwd()`, which is fragile and dependent on the way you organize your files, `here` uses the top-level directory of a project to easily build paths to files.

Das bedeutet, dass wir nun den *großen* Vorteil haben, dass wir unseren Projektordner überall hin verschieben können und unser Dateipfad immer noch relativ zu dem Ort ist, an den wir unseren Projektordner verschoben haben. Das bedeutet, dass das Projekt unabhängig davon läuft, wo es sich auf Ihrem Computer befindet. Sie können auch jemandem den Projektordner schicken, und alles sollte auf dessen Rechner laufen!

## 7.6. Arbeit mit Daten

Daten sind chaotisch, aber mit Erfahrung können wir lernen, sie zu bändigen. Im Folgenden finden Sie einige Tipps, die das Gelernte über die Datenverarbeitung erweitern.



Abbildung 7.8.: Image source: Allison Horst (all rights reserved)

### 7.6.1. Fehlende Werte

Bei der Datentransformation geht es darum, unsere Daten zu “reparieren”, wenn sie nicht “in Ordnung” sind. In unserem `df_groesse` Datenrahmen haben Sie vielleicht einige `NA` oder `N/A` Werte bemerkt. `N/A` wurde in einer unserer Beobachtungen als Text geschrieben und wird von R als solcher gelesen. `N/A` in R bezieht sich auf fehlende Daten (“Nicht verfügbar”). Echte fehlende Werte sind komplett leer, so dass `N/A` in unseren `df_groesse`-Daten nicht wirklich als fehlender Wert gelesen wird, obwohl wir möchten, dass R weiß, dass dies als fehlende Daten zählt und nicht das Haustier von jemandem namens “NA” (Menschen tun seltsame Dinge!). Um dies zu beheben, können wir das Argument `na =` für die Funktion `read_csv()` verwenden, das der Funktion `read_csv()` mitteilt, welche Werte sie mit fehlenden Werten gleichsetzen soll.

```
# force "N/A" to missing values
df_groesse <- read_csv(here::here("daten", "groesse_geburtstag.csv"),
                      na = "N/A")
# print the head of the data set
head(df_groesse)
```

```
# A tibble: 3 x 5
  Haustier GröÙe `Monat der Geburt` Tag L1
  <chr>    <dbl>           <dbl> <dbl> <chr>
1 "Lola"      171            5     7 Englisch
2 ""          168            11    26 Deutsch
3 <NA>        182            4     15 Deutsch
```

Jetzt wird der Wert, der vorher `""` war, als `NA` gelesen. Aber was ist mit der leeren Zelle? Wir haben jetzt überschrieben, dass `read_csv()` leere Zellen als `NA` liest. Jetzt wollen wir `read_csv()` anweisen, *mehr als eine* Art von Eingabe als `NA` zu lesen, d.h. wir wollen es anweisen, `""` und `"N/A"` als `NA` zu lesen. Dafür verwenden wir unsere immer nützliche Verkettungsfunktion: `c()`.

```
# force "N/A" and empty cells to missing values
df_groesse <- read_csv(here::here("daten", "groesse_geburtstag.csv"),
                      na = c("N/A", ""))
# print the head of the data set
head(df_groesse)
```

```
# A tibble: 3 x 5
  Haustier GröÙe `Monat der Geburt` Tag L1
  <chr>    <dbl>           <dbl> <dbl> <chr>
```

|   |      |     |    |    |          |
|---|------|-----|----|----|----------|
| 1 | Lola | 171 | 5  | 7  | Englisch |
| 2 | <NA> | 168 | 11 | 26 | Deutsch  |
| 3 | <NA> | 182 | 4  | 15 | Deutsch  |

## 7.6.2. Spaltennamen

Wenn wir `df_groesse` in der Konsole ausdrucken, werden wir sehen, dass ein Spaltenname von Backticks umgeben ist (z.B. `Monat der Geburt`). Das liegt daran, dass er ein Leerzeichen enthält, das syntaktisch nicht gültig ist (Variablennamen müssen mit einem Buchstaben beginnen und dürfen keine Leerzeichen oder Sonderzeichen enthalten). Eine schnelle Lösung ist die Funktion `clean_names()` aus dem Paket `janitor`, das wir bereits geladen haben.

```
clean_names(df_groesse)

# A tibble: 3 x 5
  haustier grosse monat_der_geburt   tag 11
  <chr>     <dbl>           <dbl> <dbl> <chr>
1 Lola        171             5     7 Englisch
2 <NA>       168            11    26 Deutsch
3 <NA>       182             4    15 Deutsch
```

Das sieht besser aus! Aber wenn Sie jetzt `head(df_groesse)` ausführen, sehen Sie dann die bereinigten Spaltennamen?

Das sollten Sie nicht, denn wenn wir ein Objekt durch eine Funktion übergeben, wird das Objekt nicht ‘aktualisiert’, so dass wir das Objekt erneut mit dem Zuweisungsoperator `<-` zuweisen müssen.

```
df_groesse <- janitor::clean_names(df_groesse)
```

Aber wir wissen oft, dass wir mehrere Funktionen (`read_csv()`, `clean_names()`) auf denselben Objekt ausführen wollen, denken Sie daran, dass wir das mit Pipes tun können.

## 7.6.3. Pipes

Pipes werden am Ende eines Funktionsaufrufs eingefügt, wenn das Ergebnis dieser Funktion durch eine nachfolgende Funktion weitergegeben werden soll. Pipes können als “und dann...” gelesen werden.

```
read_csv(here::here("daten", "groesse_geburtstag.csv")) |>
  head()
```

```
# A tibble: 3 x 5
  Haustier GröÙe `Monat der Geburt` Tag L1
  <chr>    <dbl>           <dbl> <dbl> <chr>
1 Lola      171             5     7 Englisch
2 <NA>       168             11    26 Deutsch
3 N/A        182             4     15 Deutsch
```

Derzeit gibt es 2 Pipes, die in R verwendet werden können.

1. die `magrittr` Paket-Pipe: `%>%`
2. die neue native R-Pipe: `|>`

Es gibt keine großen Unterschiede, die für unsere aktuellen Anwendungen wichtig sind, also benutzen wir die neue `|>`. Sie können die Tastenkombination Cmd/Ctrl + Shift/Strg + M verwenden, um eine Pipe zu erzeugen. Dies könnte die “Magrittr”-Paket-Pipe erzeugen, was in Ordnung ist, aber wenn Sie das ändern möchten, können Sie das unter **Werkzeuge > Globale Optionen > Code > Native Pipe-Operator verwenden** tun.

### Aufgabe 7.3: pipes

#### Beispiel 7.3.

1. Laden Sie den Datensatz `groesse_geburtstag.csv` erneut mit festen NAs *und dann*
  - Benutze eine Pipe, um `clean_names()` auf dem Datensatz aufzurufen, *und dann*
  - rufen Sie die Funktion `“head()”` auf
  - Überprüfen Sie die Anzahl der Beobachtungen und Variablen, gibt es ein Problem?
2. Laden Sie den Datensatz `groesse_geburtstag.csv` erneut mit festen NAs, speichern Sie ihn als Objekt `df_groesse`, *und dann*
  - Verwenden Sie eine Pipe, um `clean_names()` auf den Datensatz anzuwenden.
3. Warum sollte man nicht eine Pipe und die Funktion `head()` verwenden, wenn man den Datensatz als Objekt speichert?

#### 7.6.4. Variablentypen

Das Paket `readr` errät den Typ der Daten, die jede Spalte enthält. Die wichtigsten Spaltentypen, die man kennen muss, sind **numerisch** und **Faktor** (kategorisch). Faktoren enthalten *Kategorien* oder *Gruppen* von Daten, können aber manchmal *aussehen* wie **numerische** Daten. Zum Beispiel enthält unsere Spalte “Monat” Zahlen, aber sie könnte auch

den Namen jedes Monats enthalten. Ein guter Weg, um zu wissen, was was ist: Es ist sinnvoll, den Mittelwert einer “numerischen” Variablen zu berechnen, aber nicht den eines “Faktors”. Zum Beispiel ist es sinnvoll, den Mittelwert der Körpergröße zu berechnen, aber nicht den Mittelwert des Geburtsmonats.

Um sicherzustellen, dass eine Variable als Faktor gespeichert wird, können wir die Funktion `as_factor()` verwenden. Wir können entweder die R-Basisyntax verwenden, um dies zu tun, indem wir ein `$` verwenden, um eine Spalte in einem Datenrahmen zu indizieren:

```
df_groesse$monat_der_geburt <- as_factor(df_groesse$monat_der_geburt)
```

Or we can use `tidyverse` syntax and the `mutate()` function.

```
df_groesse <-  
  df_groesse |>  
  mutate(monat_der_geburt = as_factor(monat_der_geburt))
```

## 7.7. Andere Dateitypen und Begrenzungszeichen

Sobald Sie mit `read_csv()` vertraut sind, sind die anderen Funktionen von `readr` einfach zu benutzen, Sie müssen nur wissen, wann Sie welche benutzen.

Die Funktion `read_csv2()` liest Semikolon-separierte Dateien. Diese verwenden Semikolons (`;`) anstelle von Kommas (`,`), um Felder zu trennen und sind in Ländern üblich, die `,` als Dezimaltrennzeichen verwenden (wie Deutschland).

Die Funktion `read_tsv()` liest Tabulator-getrennte Dateien. Die Funktion `read_delim()` liest Dateien mit beliebigen Trennzeichen ein und versucht, das Trennzeichen zu erraten, es sei denn, Sie geben es mit dem Argument `delim` = an (z.B. `read_delim(students.csv, delim = ",")`).

`readr` hat mehrere andere Funktionen, die ich persönlich noch nicht gebraucht habe, wie zum Beispiel:

- `read_fwf()` liest Dateien mit fester Breite
- `read_table()` liest eine gängige Variante von Dateien mit fester Breite, bei der die Spalten durch Leerzeichen getrennt sind
- `read_log()` liest Log-Dateien im Apache-Stil

## 7.8. Übungen

Nun wollen wir üben, das Paket `readr` zu benutzen und mit unseren Daten zu arbeiten.

## **readr Funktionen**

1. Welche Funktion würdest du benutzen, um eine Datei zu lesen, in der die Felder mit “|” getrennt sind?
2. Welche Argumente haben `read_csv()` und `read_tsv()` gemeinsam?
3. Welche Funktion(en) könnten Sie verwenden, um einen Datensatz mit einem Semikolon (;) als Trennzeichen einzulesen?

## **Data wrangling**

Laden Sie die Datei `groesse_geburtstag.csv` erneut. Benutzen Sie Pipes, um auch die Funktion `clean_names` zu benutzen, und um die folgenden Änderungen im Objekt `df_groesse` vorzunehmen.

1. Umwandlung der Variablen `11` in einen Faktor.
2. Umbenennen von
  - `grosses` in `groesse`
  - `monat_der_geburt` in `geburtsmonat`

## **Plots**

Erstelle ein Streudiagramm mit unserem `df_groesse`-Datensatz, das die Beziehung zwischen unserem Geburtsdatum und unseren Geburtstagen visualisiert (es macht keinen Sinn, dies zu vergleichen, aber es ist nur eine Übung). Suchen Sie Ihren Geburtstag in der Grafik. Stellen Sie die Farbe und die Form so ein, dass sie “L1” entsprechen. Fügen Sie einen Titel für die Grafik hinzu.

## **Lernziele**

Heute haben wir gelernt, wie man...

- lokale Datendateien mit dem Paket `readr` importiert
- fehlende Werte behandeln
- Variablen in Faktoren umwandeln

Lassen Sie uns nun dieses neue Wissen anwenden.

## Session Info

Hergestellt mit R version 4.3.0 (2023-04-21) (Already Tomorrow) und RStudioversion 2023.9.0.463 (Desert Sunflower).

```
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; 

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] magick_2.7.4     here_1.0.1       janitor_2.2.0    lubridate_1.9.2
[5]forcats_1.0.0    stringr_1.5.0    dplyr_1.1.3      purrr_1.0.2
[9]readr_2.1.4      tidyverse_2.0.0  tibble_3.2.1     ggplot2_3.4.3
[13]tidyverse_2.0.0

loaded via a namespace (and not attached):
[1] utf8_1.2.3      generics_0.1.3   stringi_1.7.12   hms_1.1.3
[5] digest_0.6.33   magrittr_2.0.3   evaluate_0.21   grid_4.3.0
[9] timechange_0.2.0 fastmap_1.1.1   rprojroot_2.0.3 jsonlite_1.8.7
[13] fansi_1.0.4     scales_1.2.1     cli_3.6.1       crayon_1.5.2
[17] rlang_1.1.1     bit64_4.0.5     munspell_0.5.0  withr_2.5.0
[21] yaml_2.3.7      parallel_4.3.0   tools_4.3.0     tzdb_0.4.0
[25] colorspace_2.1-0 pacman_0.5.1    vctrs_0.6.3     R6_2.5.1
[29] lifecycle_1.0.3  snakecase_0.11.0  bit_4.0.5      vroom_1.6.3
[33] pkgconfig_2.0.3  pillar_1.9.0    gtable_0.3.4    glue_1.6.2
[37] Rcpp_1.0.11     xfun_0.39      tidyselect_1.2.0 rstudioapi_0.14
[41] knitr_1.44      htmltools_0.5.5   rmarkdown_2.22   compiler_4.3.0
```

## Literaturverzeichnis

- Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*.
- Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>
- Davies, R., Locke, S., & D'Agostino McGowan, L. (2022). *datasauRus: Datasets from the Datasaurus Dozen*. <https://CRAN.R-project.org/package=datasauRus>
- Müller, K. (2020). *here: A Simpler Way to Find Your Files*. <https://CRAN.R-project.org/package=here>
- Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills*. Zenodo. <https://doi.org/10.5281/zenodo.6365078>
- Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2022). Data Visualization Using R for Researchers Who Do Not Use R. *Advances in Methods and Practices in Psychological Science*, 5(2), 251524592210746. <https://doi.org/10.1177/25152459221074654>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. <https://doi.org/10.21105/joss.01686>
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2. Aufl.).
- Winter, B. (2019). Statistics for Linguists: An Introduction Using R. In *Statistics for Linguists: An Introduction Using R*. Routledge. <https://doi.org/10.4324/9781315165547>
- Xie, Y. (2023). *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>

# 8. Deskriptive Statistik

Maße der zentralen Tendenz und Streuung

## Lernziele

In diesem Kapitel werden wir lernen...

- über Maße der zentralen Tendenz (Mittelwert, Median, Modus)
- über Streuungsmaße (Bereich, Standardabweichung)
- wie man die Funktion `summarise()` von `dplyr` benutzt
- wie man Zusammenfassungen `.by` Gruppe erstellt

## Lesungen

Die **Pflichtlektüre** zur Vorbereitungen auf dieses Thema sind

1. Kap. 3, Abschnitt 3.4-3.9 (Descriptive statistics, models, and distributions) in Winter (2019) (online verfügbar über das [HU Grimm Zentrum](#)).
2. [Bereich 4.5 \(Groups\)](#) in Kapital 4 (Data Transformation) in Wickham et al. (2023).

## 8.1. Einrichten

### 8.1.1. Umgebung löschen

Ein wichtiger Schritt, über den wir noch nicht viel gesprochen haben, ist sicherzustellen, dass Sie ein neues Skript *immer* mit einer leeren R-Umgebung starten. Das bedeutet, dass wir keine Objekte in der Umgebung gespeichert haben sollten, aber auch keine Pakete geladen haben sollten. Wir wollen nämlich sicherstellen, dass alles, was wir tun, ausschließlich in diesem Skript ausgeführt wird und nicht von einem Paket oder Daten abhängt, die wir aus

einem anderen Skript geladen haben. Um dies zu erreichen, können Sie auf **Sitzung > R neu starten** klicken, um mit einer neuen Umgebung zu beginnen, oder die Tastenkombination **Cmd/Strg+Strg+0** verwenden.

### 8.1.2. Pakete

Wir müssen die Pakete `tidyverse`, `here` und `janitor` laden. Die letzten beiden brauchen wir, weil wir lokale CSV-Datensätze laden werden.

```
pacman::p_load(tidyverse,
                 here,
                 janitor)
```

### 8.1.3. Daten laden

Wir werden heute zwei Datensätze verwenden: eine leicht veränderte Version des `groesse_geburtstag`-Datensatzes aus dem letzten Abschnitt (`groesse_geburtstag_ws2324.csv`) und `languageR_english.csv`, das eine kürzere Version des `english`-Datensatzes aus dem `languageR`-Paket ist. Wenn Sie diese Daten noch nicht haben, laden Sie sie direkt in Ihren Daten-Ordner vom GitHub-Kurs herunter (klicken Sie auf “Download raw file” neben dem “Raw”-Button):

- [languageR\\_english.csv](#)
- [groesse\\_geburtstag\\_ws2324.csv](#)

```
df_groesse <- read_csv(here("daten", "groesse_geburtstag_ws2324.csv"))
```

```
df_eng <- read_csv(here("daten", "languageR_english.csv")) |>
  clean_names() |>
  # fix some wonky variable names:
  rename(rt_lexdec = r_tlexdec,
         rt_naming = r_tnaming)
```

## 8.2. Deskriptive Statistik

Deskriptive Statistiken beschreiben quantitativ die zentrale Tendenz, die Variabilität und die Verteilung von Daten. Sie werden manchmal auch als zusammenfassende Statistiken bezeichnet, weil wir die beobachteten Daten *zusammenfassen*. Zu den gängigen zusammenfassenden Statistiken gehören der Wertebereich (Minimum, Maximum), der Mittelwert und die

Standardabweichung. Deskriptive Statistiken helfen uns, unsere Daten in vollem Umfang zu verstehen, und sind ein wichtiger Schritt bei der Untersuchung unseres Datensatzes, bevor wir fortgeschrittenere *Inferenzstatistiken* durchführen (die wir in diesem Kurs nicht behandeln werden).

### 8.2.1. Anzahl der Beobachtungen ( $n$ )

Die Anzahl der Beobachtungen in einem Datensatz ist keine statistische Größe, sondern eine wichtige Information bei der Zusammenfassung oder Beschreibung von Daten. Wenn wir mehr Daten haben (höher  $n$ ), können wir den Schlussfolgerungen, die wir aus unseren Daten ziehen, mehr Vertrauen schenken, da wir mehr Beweise haben. Umgekehrt kann es sein, dass bei weniger Daten (niedriger  $n$ ) unsere zusammenfassende Statistik nicht auf die Grundgesamtheit verallgemeinerbar ist. Wir können die Anzahl der Beobachtungen in einem Datensatz mit der R-eigenen Funktion `nrow()` überprüfen:

```
nrow(df_groesse)
```

```
[1] 9
```

#### i length() gegenüber nrow()

Die Funktion `length()` sagt uns, wie viele (horizontale) Werte in einem Objekt enthalten sind. Wenn das Objekt ein Datenrahmen (statt eines Vektors) ist, sagt sie uns, wie viele *Spalten* wir haben.

```
length(df_groesse)
```

```
[1] 5
```

Wenn es sich bei dem Objekt jedoch um einen Vektor handelt, dann gibt uns `length()` die Anzahl der Beobachtungen an.

```
vector <- c(1,5,2,6,8,4,7,8,3)
length(vector)
```

```
[1] 9
```

## 8.2.2. Maße der zentralen Tendenz (Lagemaße)

Maße der zentralen Tendenz beschreiben quantitativ die Mitte unserer Daten. Wahrscheinlich haben Sie schon einmal drei Maße der zentralen Tendenz kennengelernt: den Mittelwert, den Median und den Modus.

### 8.2.2.1. Mittelwert ( $\mu$ oder $\bar{x}$ )

Der Mittelwert oder Durchschnitt ist die Summe aller Werte geteilt durch die Anzahl der Werte (wie in Gleichung 8.1). In der mathematischen Notation wird *sum* mit dem großen griechischen Sigma ( $\sum$ ) geschrieben, wie in der Gleichung 8.2.

$$\mu = \frac{\text{Summe der Werte}}{n} \quad (8.1)$$

$$\bar{x} = \frac{\sum x}{n} \quad (8.2)$$

### 8.2.2.2. Populationsmittelwert ( $\mu$ ) versus Stichprobenmittelwert ( $\bar{x}$ )

Beide Gleichungen bedeuten dasselbe, verwenden aber unterschiedliche Schreibweisen, um dieselbe Gleichung darzustellen. Während  $\mu$  den *Populationsmittelwert* darstellt, repräsentiert  $\bar{x}$  den *Stichprobenmittelwert*. Der Populationsmittelwert ist der wahre Mittelwert einer Messung in einer gesamten Population (z. B. die Körpergröße aller Studenten an der Humboldt-Universität zu Berlin). Ein *Stichprobenmittel* ist der Mittelwert einer *Stichprobenpopulation*, aus der wir unsere Daten erhoben haben. Wir haben zum Beispiel 9 Beobachtungen in `df_groesse`. Diese Daten stellen eine Stichprobe von Daten aus einer größeren Grundgesamtheit dar.

Wir können den Mittelwert leicht von Hand berechnen, wenn wir nur ein paar Werte haben. Erinnern Sie sich an unseren Datensatz von letzter Woche, in dem wir unsere Höhen in Zentimetern gesammelt haben (171, 168, 182, 190, 170, 163, 164, 167, 189). Es gibt 9 Werte, also müssen wir diese Höhen addieren und die Summe durch 9 teilen.

```
171+ 168+ 182+ 190+ 170+ 163+ 164+ 167+ 189 / 9
```

```
[1] 1396
```

Daraus ergibt sich eine durchschnittliche Körpergröße von 1396 cm. Das kann nicht richtig sein, was ist also schief gelaufen? Wir können die obige Gleichung korrigieren, indem wir die Höhen in Klammern setzen (), bevor wir durch  $n$  dividieren.

```
(171+ 168+ 182+ 190+ 170+ 163+ 164+ 167+ 189) / 9
```

```
[1] 173.7778
```

Dieses Problem wurde durch die *Reihenfolge der Operationen* verursacht, die im Folgenden näher beschrieben wird. Das Wichtigste ist, dass Sie sicher sein können, dass das *Ergebnis* einer bestimmten Operation vor allen anderen Operationen ausgeführt wird, wenn Sie es in Parenthesen einschließen.

Wir können auch die Ergebnisse einer Gleichung als Objekt oder mehrere Werte als Vektor (eine Liste von Werten der gleichen Klasse) speichern. Wir könnten dann die Funktionen `sum()` und `length()` verwenden, um den Mittelwert zu berechnen, oder einfach die Funktion `mean()` benutzen.

```
# save groesse as a vector
groesse <- c(171, 168, 182, 190, 170, 163, 164, 167, 189)
# divide the sum of groesse by the n of groesse
sum(groesse)/length(groesse)
```

```
[1] 173.7778
```

```
# or use the mean() function
mean(groesse)
```

```
[1] 173.7778
```

Unsere Daten sind oft nicht in einem einzelnen Vektor gespeichert, sondern in einem Datensatz. Wir können die Funktion `mean()` auf eine Variable in einem Datenrahmen anwenden, indem wir den Operator `$` verwenden, um anzugeben, dass wir eine Spalte aus einem Datenrahmen auswählen wollen (`datenrahmen$variable`).

```
mean(df_groesse$groesse)
```

```
[1] 173.6667
```

Der `$`-Operator ist Teil der nativen R-Syntax und ähnelt dem Operator `pdf_groesse |>select(groesse)` in der dplyr-Syntax.

### 8.2.2.3. Median

Ein weiteres Maß für die zentrale Tendenz ist der **Median**, d. h. der Wert in der Mitte des Datensatzes. Wenn Sie alle Ihre Werte in aufsteigender (oder absteigender) Reihenfolge anordnen, ist der mittlere Wert der Median. Wenn Sie zum Beispiel 5 Werte haben, ist der 3. Bei 6 Werten ist der Mittelwert des 3. und 4. Wertes der Median. Die Hälfte der Daten liegt unter dem Median, die andere Hälfte über dem Median.

Um unsere Daten in aufsteigender Reihenfolge zu sortieren, können wir die Funktion `sort()` verwenden. Wir können dann einfach zählen, welches der mittlere Wert ist:

```
sort(df_groesse$groesse)
```

```
[1] 163 164 167 167 170 171 182 189 190
```

Das ist einfach, wenn wir nur ein paar Beobachtungen haben. Wir könnten alternativ einfach die Funktion `Median()` verwenden.

```
median(df_groesse$groesse)
```

```
[1] 170
```

Ein wichtiges Merkmal des Medians ist, dass er nicht von Ausreißern oder Extremwerten beeinflusst wird. Schauen wir uns an, was passiert, wenn wir unsere größte Körpergröße (190 cm) so ändern, dass sie der Größe der derzeit größten Person der Welt entspricht: 251 cm.

```
df_groesste <- df_groesse |> mutate(groesse = ifelse(groesse == 190, 251, groesse))
```

```
sort(df_groesste$groesse)
```

```
[1] 163 164 167 167 170 171 182 189 251
```

```
median(df_groesste$groesse)
```

```
[1] 170
```

```
mean(df_groesste$groesse)
```

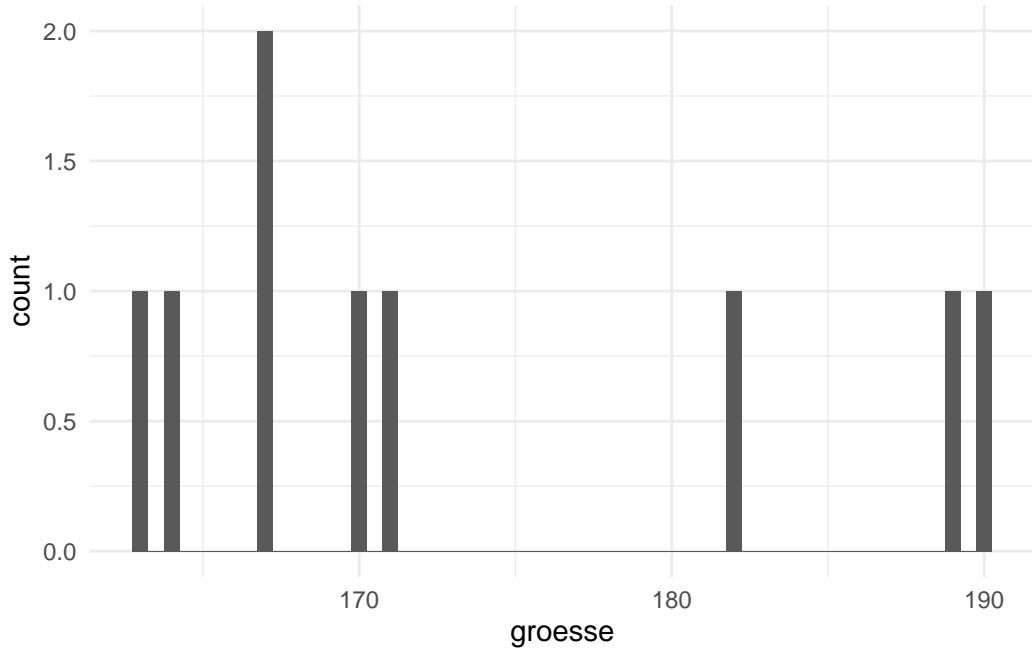
```
[1] 180.4444
```

Wir sehen, dass sich der Mittelwert von ungefähr 174cm auf 180cm geändert hat. Der Median blieb jedoch gleich (170 cm), weil der Mittelwert unabhängig von den anderen Werten in einem Datensatz ist. Aus diesem Grund wird der Median häufig anstelle des Mittelwerts angegeben, wenn die Daten stark zu extremeren Werten neigen, wie z. B. bei der Angabe der Einkommen in einer Bevölkerung. Durchschnittseinkommen können aufgrund einer kleinen Gruppe von extrem gut Verdienenden stark verzerrt sein und sind in der Regel nicht repräsentativ für das Einkommen der Mehrheit der Bürger.

#### 8.2.2.4. Modus

Der Modus ist der Wert, der in einem Datensatz am häufigsten vorkommt, und ist ein weiteres Maß für die zentrale Tendenz. Es gibt keine R-Funktion, um den Modus zu bestimmen, aber wir haben bereits einige gängige Möglichkeiten gesehen, ihn zu visualisieren: mit einem Histogramm oder einem Dichteplot.

```
df_groesse |>  
  ggplot(aes(x = groesse)) +  
  geom_histogram(binwidth = .5) +  
  theme_minimal()
```



### 8.2.3. Streuungsmaße

Maße der zentralen Tendenz beschreiben (normalerweise) die Mitte der Daten. Streuungsmaße beschreiben die Streuung der Datenpunkte und sagen etwas darüber aus, wie die Daten insgesamt verteilt sind.

#### 8.2.3.1. Bereich

Der “Bereich” von Werten kann sich auf den höchsten (maximalen) und den niedrigsten (minimalen) Wert oder auf die Differenz zwischen höchstem und niedrigstem Wert beziehen. Die R-Basisfunktionen `max()` und `min()` geben die höchsten und niedrigsten Werte aus.

```
max(groesse)
```

```
[1] 190
```

```
min(groesse)
```

```
[1] 163
```

Oder wir können einfach die Funktion `range()` verwenden, die diese beiden Zahlen nebeneinander ausgibt.

```
range(groesse)
```

```
[1] 163 190
```

Wir können die Differenz zwischen diesen Werten ermitteln, indem wir den Minimalwert vom Maximalwert subtrahieren.

```
max(groesse) - min(groesse)
```

```
[1] 27
```

In einem Histogramm oder Dichteplot werden diese Werte durch den niedrigsten und den höchsten Wert auf der x-Achse dargestellt.



### 8.2.3.2. Standardabweichung (sd oder $\sigma$ )

Die Standardabweichung ist ein Maß dafür, wie weit die Daten *im Verhältnis zum Mittelwert* gestreut sind. Eine niedrige Standardabweichung bedeutet, dass die Daten um den Mittelwert herum gruppiert sind (d. h. es gibt weniger Streuung), während eine hohe Standardabweichung bedeutet, dass die Daten stärker gestreut sind. Ob eine Standardabweichung hoch oder niedrig ist, hängt von der Skala und der Maßeinheit ab, in der die Daten vorliegen. Die Standardabweichung wird sehr oft angegeben, wenn der Mittelwert berichtet wird.

Die Standardabweichung (`sd`) ist gleich der Quadratwurzel ( $\sqrt{\cdot}$  oder `sqrt()` in R) der Summe der quadrierten Wertabweichungen vom Mittelwert  $((x - \mu)^2)$  geteilt durch die Anzahl der Beobachtungen minus 1 ( $n - 1$ ), angegeben in Gleichung 8.3.

$$\sigma = \sqrt{\frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \dots + (x_N - \mu)^2}{N - 1}} \quad (8.3)$$

Das sieht einschüchternd aus, aber wir können die Standardabweichung in R mit der Funktion `sd()` berechnen.

```
sd(groesse)
```

```
[1] 10.46157
```

Wenn man jedoch weiß, wie man die Standardabweichung von Hand berechnen kann, versteht man, was die Zahl bedeutet. Lassen Sie uns die Berechnung der Standardabweichung für eine kleine Gruppe von Werten üben. Unter Berücksichtigung der Gleichung für die Standardabweichung in 8.3 können wir die Standardabweichung von Hand berechnen, wenn wir den Wert jeder Beobachtung, den Mittelwert dieser Werte und die Anzahl dieser Werte kennen. In einem Vektor mit 3 Beobachtungen (3, 5, 9) sind unsere Werte ( $x$ ) zum Beispiel folgende:

```
values <- c(3,5,16)  
values
```

```
[1] 3 5 16
```

Setzt man diese in die Gleichung für die Standardabweichung ein, erhält man Gleichung 8.4.

$$\sigma = \sqrt{\frac{(3 - \mu)^2 + (5 - \mu)^2 + (16 - \mu)^2}{N - 1}} \quad (8.4)$$

Unser Mittelwert ( $\mu$ ) ist:

```
mean(values)
```

```
[1] 8
```

Wenn wir dies zu Gleichung 8.4 hinzufügen, erhalten wir Gleichung 8.5.

$$\sigma = \sqrt{\frac{(3 - 8)^2 + (5 - 8)^2 + (16 - 8)^2}{N - 1}} \quad (8.5)$$

Die Anzahl der Werte ( $n$ ) ist:

```
length(values)
```

```
[1] 3
```

Wenn wir dies zu Gleichung 8.5 hinzufügen, erhalten wir Gleichung 8.6.

$$\sigma = \sqrt{\frac{(3-8)^2 + (5-8)^2 + (16-8)^2}{3-1}} \quad (8.6)$$

If we carry out all of the operations following PEDMAS, then we get Equations 8.8 through 8.3:

$$\sigma = \sqrt{\frac{(-5)^2 + (-3)^2 + (8)^2}{3-1}} \quad (8.7)$$

(8.8)

$$= \sqrt{\frac{25 + 9 + 64}{3-1}} \quad (8.9)$$

(8.10)

$$= \sqrt{\frac{98}{2}} \quad (8.11)$$

(8.12)

$$= \sqrt{49}$$

$$= 7$$

Um unsere Arbeit zu überprüfen, berechnen wir die Standardabweichung ( $\sigma$ ) in “R”:

```
sd(values)
```

```
[1] 7
```

### 8.2.3.3. Warum Standardabweichung?

Die Standardabweichung ist ein Maß dafür, wie “eng” die beobachteten Werte am Mittelwert liegen. Wenn die meisten Beobachtungen sehr nahe am Mittelwert liegen, ist die Standardabweichung im Verhältnis zum Mittelwert eine kleine Zahl. Wenn es viele Beobachtungen mit großen Abweichungen vom Mittelwert gibt, wird die Standardabweichung tendenziell eine große Zahl sein (relativ zum Mittelwert).

Verschiedene Datensätze können denselben Mittelwert, aber sehr unterschiedliche Standardabweichungen aufweisen. Ein Beispiel:

```
values2 <- c(55, 55, 55, 55, 55, 57, 57, 57, 57, 57)
values3 <- c(1, 1, 1, 1, 100, 100, 100, 100, 100)
```

```
mean(values2)
```

```
[1] 56
```

```
mean(values3)
```

```
[1] 56
```

Wir sehen, dass `values2` und `values3` den gleichen Mittelwert haben. Daraus könnte man schließen, dass die Daten ähnlich sind. Aber ihre Standardabweichungen werden sich unterscheiden, weil ihre jeweiligen beobachteten Werte alle unterschiedlich weit vom Mittelwert abweichen. Welcher Vektor wird Ihrer Meinung nach die *geringste* Standardabweichung haben? Und warum?

```
sd(values2)
```

```
[1] 1.054093
```

```
sd(values3)
```

```
[1] 52.17758
```

Die größere Standardabweichung für `Werte3` spiegelt die Tatsache wider, dass die Werte tendenziell sehr weit vom Mittelwert entfernt sind. Die kleinere Standardabweichung für `Werte2` spiegelt die Tatsache wider, dass der Wert für diese Variable tendenziell recht nahe am Mittelwert liegt.

### 💡 Quadrat und Quadratwurzel

Warum quadrieren wir die Abweichung jeder Beobachtung vom Mittelwert, um dann später die Quadratwurzel aus ihrer Summe geteilt durch  $N - 1$  zu berechnen? Da die Hälfte unserer Beobachtungen unterhalb und die Hälfte oberhalb des Mittelwerts liegen wird, sind die resultierenden Differenzen, wenn wir den Mittelwert von den Werten abziehen, zur Hälfte negativ und zur Hälfte positiv. Wenn wir positive und negative Werte zusammenzählen, heben sie sich gegenseitig auf. Wenn wir also alle diese Abweichungen vom Mittelwert quadrieren, werden alle Werte positiv sein (eine positive Zahl multipliziert mit einer positiven Zahl ist eine positive Zahl, während eine negative Zahl multipliziert

mit sich selbst ebenfalls eine positive Zahl ergibt). Wenn wir dann die Quadratwurzel dieser Werte berechnen, erhalten wir die ursprüngliche Größe der Abweichung, aber immer als positiven Wert.

### 💡 Eigenschaften der Grundgesamtheit

Sowohl der Mittelwert als auch die Standardabweichung sagen etwas über die Grundgesamtheit aus, aus der unsere Datenstichprobe stammt. Je mehr Beobachtungen wir sammeln, desto genauer werden diese Maße im Durchschnitt sein.

## 8.3. Zusammenfassende Statistiken mit R

Wir haben bereits einige nützliche Funktionen zur Berechnung von zusammenfassenden Statistiken gesehen (z.B. `mean()`, `median()`, `sd()`). In der Regel möchten wir jedoch mehrere zusammenfassende Statistiken auf einmal erstellen und zusammenfassende Statistiken zwischen Gruppen vergleichen. Um dies zu erreichen, bietet das Paket `dplyr` aus dem `tidyverse` einige hilfreiche Funktionen. Lassen Sie uns nun den `df_eng`-Datensatz verwenden, um diese `dplyr`-Verben kennenzulernen.

### 8.3.1. `dplyr::summarise`

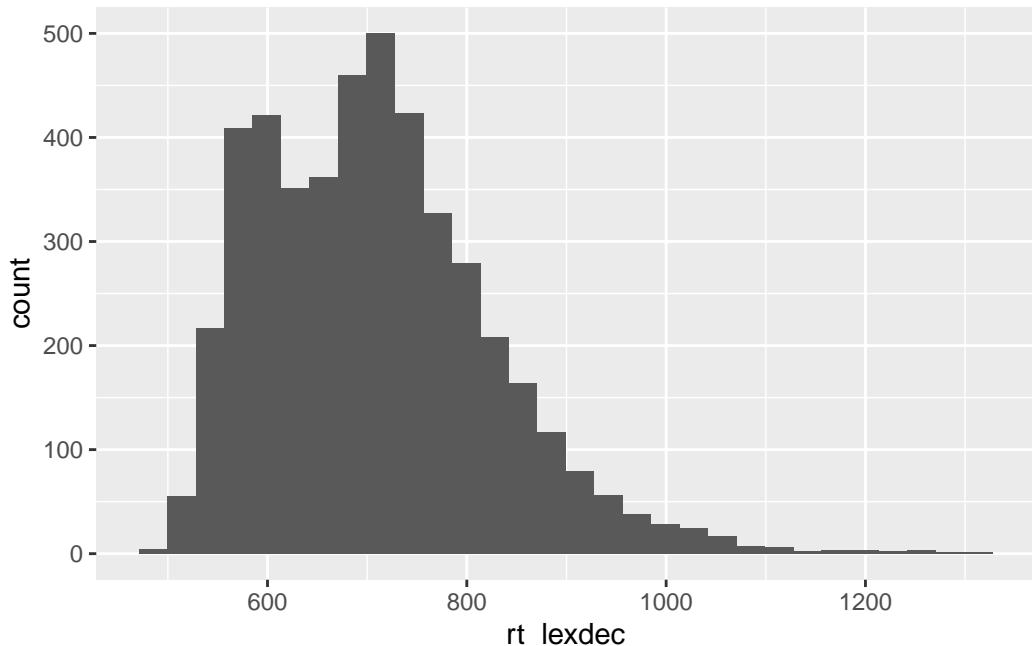
Die Funktion `summarise()` von `dplyr` berechnet Zusammenfassungen von Daten, aber wir müssen ihr sagen, *was* sie berechnen soll, und für welche Variable(n). Die Funktion `n()` liefert zum Beispiel die Anzahl der Beobachtungen (nur wenn sie innerhalb von `summarise()` oder `mutate()` verwendet wird). Lassen Sie uns zunächst prüfen, wie viele Beobachtungen wir im Datensatz `df_eng` haben:

```
df_eng |>  
  summarise(N = n())
```

```
# A tibble: 1 x 1  
      N  
  <int>  
1  4568
```

Werfen wir nun einen Blick auf das Histogramm von `rt_lexdec`, der Variable, die die lexikalische Entscheidungsantwortzeit in Millisekunden enthält:

```
df_eng |>
  ggplot() +
  aes(x = rt_lexdec) +
  geom_histogram()
```



Wir sehen, dass die Antwortzeit zwischen 500 ms und 1320 ms liegt, wobei die meisten Antworten zwischen 550 ms und 900 ms liegen. Wir sehen auch eine *bimodale* Verteilung, d. h. es gibt zwei Modi (zwei Spitzen). Der allgemeine Modus liegt bei 700 ms (500 Beobachtungen), mit einer zweiten Spitzte bei 600 ms (~420 Beobachtungen).

Wir können auch mehrere Berechnungen auf einmal durchführen. Lassen Sie uns auch den Mittelwert und die Standardabweichung der lexikalischen Entscheidungsaufgabe (`rt_lexdec`, in Millisekunden) berechnen.

```
df_eng |>
  summarise(mean_lexdec = mean(rt_lexdec, na.rm=T),
            sd_lexdec = sd(rt_lexdec, na.rm = T),
            N = n())

# A tibble: 1 x 3
  mean_lexdec    sd_lexdec      N
  <dbl>        <dbl>     <int>
```

```
1           708.       115.  4568
```

Jetzt sehen wir, dass die durchschnittliche lexikalische Entscheidungsantwortzeit 708.1 ms betrug, mit einer Standardabweichung von 114.9.

Und wir können Berechnungen mit typischen mathematischen Operatoren (z.B. +, -, /, \*, ^ ...) und/oder Funktionen angeben. Was war der Unterschied zwischen der längsten und der kürzesten lexikalischen Entscheidungsantwortzeit?

```
df_eng |>
  summarise(range_lexdec = max(rt_lexdec) - min(rt_lexdec))

# A tibble: 1 x 1
  range_lexdec
  <dbl>
1     828.
```

### 💡 Fehlende Werte

Einige Berechnungen sind nicht möglich, wenn Werte fehlen. Die Variable `rt_naming` hat einen fehlenden Wert. Dies ist in der Ausgabe der Funktion `summary()` zu sehen, die vor der Berechnung der zusammenfassenden Statistik alle NA-Werte löscht.

```
df_eng |>
  select(rt_lexdec, rt_naming) |>
  summary()
```

	rt_lexdec	rt_naming
Min.	: 495.4	Min. :412.3
1st Qu.	: 617.4	1st Qu.:468.1
Median	: 699.6	Median :570.6
Mean	: 708.1	Mean :565.9
3rd Qu.	: 775.3	3rd Qu.:658.4
Max.	:1323.2	Max. :808.9
NA's		:1

Die Funktion `mean()` entfernt jedoch *nicht* die NA-Werte.

```
df_eng |>
  summarise(mean_naming = mean(rt_naming))

# A tibble: 1 x 1
```

```
mean_naming  
      <dbl>  
1        NA
```

Was tun wir mit fehlenden Werten? Bei der Arbeit mit realen Daten ist es nicht trivial, wie wir mit fehlenden Werten umgehen. Wir könnten z. B. alle `NA`-Werte in 0 umwandeln, wenn wir wollen, dass sie zur Berechnung des Mittelwerts beitragen. Meistens wollen wir sie jedoch einfach entfernen.

Wir können dies leicht mit dem `dplyr`-Verb `drop_na()` tun:

```
df_eng |>  
  drop_na() |>  
  summarise(mean_naming = mean(rt_naming))  
  
# A tibble: 1 x 1  
  mean_naming  
      <dbl>  
1        566.
```

## 8.4. Gruppierung von Variablen

Wir wollen jedoch nicht immer nur die zusammenfassenden Statistiken für einen gesamten Datensatz kennen. Normalerweise wollen wir bestimmte Gruppen *vergleichen* (z. B. Vergleich der Reaktionszeiten bei lexikalischen Entscheidungen zwischen Altersgruppen)

### 8.4.1. `.by =`

Das semi-neue (und experimentelle) Argument `.by =` in `summarise()` berechnet unsere Berechnungen auf gruppierten Teilmengen der Daten. Es nimmt eine **Variable** (d.h. einen Spaltennamen) und gruppiert nach den Stufen dieser Variable.

```
df_eng |>  
  drop_na() |>  
  summarise(mean_lexdec = mean(rt_lexdec),  
            sd_lexdec = sd(rt_lexdec),  
            N = n(),  
            .by = age_subject) |>  
  arrange(mean_lexdec)
```

```
# A tibble: 2 x 4
  age_subject mean_lexdec sd_lexdec     N
  <chr>          <dbl>      <dbl> <int>
1 young           630.       69.1   2283
2 old             787.       96.2   2284
```

#### 8.4.2. Gruppieren nach mehreren Variablen

- Wir können auch nach mehreren Variablen gruppieren
  - dafür brauchen wir `concatenate (c())`

```
1 df_eng |>
2   drop_na() |>
3   summarise(mean_lexdec = mean(rt_lexdec),
4             sd_lexdec = sd(rt_lexdec),
5             N = n(),
6             .by = c(age_subject, word_category)) |>
7   arrange(age_subject)
```

```
# A tibble: 4 x 5
  age_subject word_category mean_lexdec sd_lexdec     N
  <chr>        <chr>          <dbl>      <dbl> <int>
1 old          N              790.       101.   1452
2 old          V              780.       86.5    832
3 young        N              633.       70.8   1451
4 young        V              623.       65.7    832
```

##### i `group_by()`

- Anstelle des neuen `.by`-Arguments können wir das `dplyr`-Verb `group_by()` und `ungroup()` verwenden
  - Ich bevorzuge das neue `.by`, weil es die Gruppierung lokal hält (keine Notwendigkeit für `ungroup()`)
  - Behalten Sie dies im Hinterkopf, Sie könnten `group_by()` in freier Wildbahn sehen

Tabelle 8.1.: Summary stats of Anscombe's quartet datasets

grp	mean_x	mean_y	min_x	min_y	max_x	max_y	
Group 1	9	7.500909	4	4.26	14	10.84	0
Group 2	9	7.500909	4	3.10	14	9.26	0
Group 3	9	7.500000	4	5.39	14	12.74	0
Group 4	9	7.500909	8	5.25	19	12.50	0

```

1 df_eng |>
2   group_by(age_subject, word_category) |>
3   summarise(mean_lexdec = mean(rt_lexdec),
4             sd_lexdec = sd(rt_lexdec),
5             N = n()) |>
6   ungroup() |>
7   arrange(age_subject)

# A tibble: 4 x 5
#>   age_subject word_category mean_lexdec sd_lexdec     N
#>   <chr>        <chr>           <dbl>      <dbl> <int>
#> 1 old          N              790.       101.    1452
#> 2 old          V              780.       86.5     832
#> 3 young        N              633.       70.8    1452
#> 4 young        V              623.       65.7     832

```

## 8.5. Anscombes Quartett

Francis Anscombe erstellte 1973 vier Datensätze, um zu veranschaulichen, wie wichtig es ist, Daten zu visualisieren, bevor man sie analysiert und ein Modell erstellt. Diese vier Diagramme stellen 4 Datensätze dar, die alle einen nahezu identischen Mittelwert und eine Standardabweichung, aber sehr unterschiedliche Verteilungen aufweisen.

### 8.5.1. DatasaurRus

Das Paket `datasaurRus` (Davies et al., 2022) enthält einige weitere Datensätze, die ähnliche Mittelwerte und `sd`, aber unterschiedliche Verteilungen aufweisen, die in Tabelle 8.2

## Anscombe's Quartet

$y = 0.5x + 3$  ( $r \approx 0.82$ ) for all groups

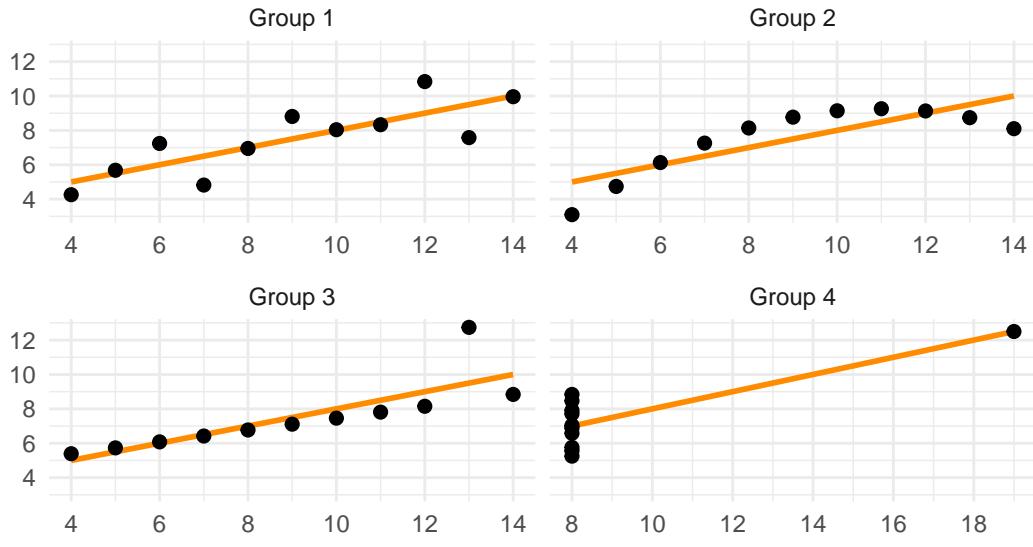


Abbildung 8.1.: Plots of Anscombe's quartet distributions

angegeben sind.

```
pacman::p_load("datasauRus")

datasaurus_dozen |>
  group_by(dataset) |>
  summarize(
    mean_x      = mean(x),
    mean_y      = mean(y),
    std_dev_x   = sd(x),
    std_dev_y   = sd(y),
    corr_x_y   = cor(x, y)
  ) |>
  knitr::kable() |>
  kableExtra::kable_styling(font_size = 20)
```

Wenn wir die Datensätze grafisch darstellen, sehen sie alle sehr unterschiedlich aus (Abbildung 8.2)!

Der Punkt hier ist: ***Stellen Sie Ihre Daten immer grafisch dar*** und betrachten Sie nicht nur die beschreibenden Statistiken!!! Beides ist sehr wichtig für das Verständnis Ihrer Daten. Wir haben bereits gesehen, wie wir unsere Rohdaten mithilfe von Histogrammen, Dichteplots,

Tabelle 8.2.: Summary stats of datasauRus datasets

dataset	mean_x	mean_y	std_dev_x	std_dev_y	corr_
away	54.26610	47.83472	16.76983	26.93974	-0.064
bullseye	54.26873	47.83082	16.76924	26.93573	-0.068
circle	54.26732	47.83772	16.76001	26.93004	-0.068
dino	54.26327	47.83225	16.76514	26.93540	-0.064
dots	54.26030	47.83983	16.76774	26.93019	-0.060
h_lines	54.26144	47.83025	16.76590	26.93988	-0.061
high_lines	54.26881	47.83545	16.76670	26.94000	-0.068
slant_down	54.26785	47.83590	16.76676	26.93610	-0.068
slant_up	54.26588	47.83150	16.76885	26.93861	-0.068
star	54.26734	47.83955	16.76896	26.93027	-0.062
v_lines	54.26993	47.83699	16.76996	26.93768	-0.069
wide_lines	54.26692	47.83160	16.77000	26.93790	-0.066
x_shape	54.26015	47.83972	16.76996	26.93000	-0.065

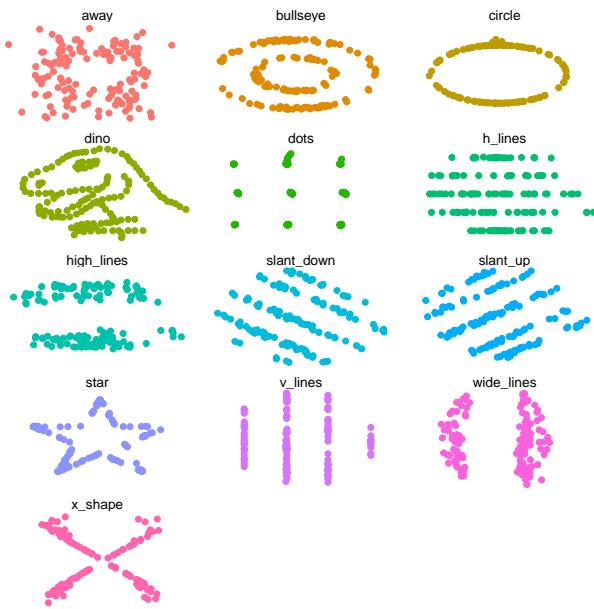


Abbildung 8.2.: Plots of datasauRus dataset distributions

Balkendiagrammen und Streudiagrammen darstellen können. Nächste Woche werden wir uns ansehen, wie wir unsere zusammenfassenden Statistiken darstellen und wie wir die Rohdaten in die Darstellung mit mehrteiligen Diagrammen einbeziehen können.

## Lernziele

Today we learned...

- über Maße der zentralen Tendenz
- über Streuungsmaße
- wie man die Funktion `summarise()` von `dplyr` verwendet
- wie man Zusammenfassungen nach Gruppen erstellt

## 8.6. Hausaufgaben

::: nonincremental

1. Berechnen Sie die Standardabweichung der Werte 152, 19, 1398, 67, 2111, ohne die Funktion `sd()` zu benutzen.

- zeige deine Arbeit. Die folgende R-Syntax könnte nützlich sein (je nachdem, wie Sie es machen wollen):
    - `c()`
    - `mean()`
    - `x^2` berechnet das Quadrat eines Wertes (hier, `x`)
    - `sqrt()` berechnet die Quadratwurzel
    - `length()` liefert die Anzahl der Beobachtungen in einem Vektor
2. Benutze die Funktion `sd()`, um die Standardabweichung der obigen Werte zu drucken. Haben Sie es richtig gemacht?
  3. Benutze `summarise`, um den Mittelwert, die Standardabweichung und die Anzahl der Beobachtungen für `rt_naming` im `df_lexdec` Datenrahmen zu drucken.
    - Hinweis: Müssen Sie fehlende Werte (`NA`) entfernen?
  4. Machen Sie dasselbe, aber fügen Sie das Argument `.by()` hinzu, um die mittlere Reaktionszeit der Benennungsaufgabe (`rt_naming`) pro Monat zu ermitteln
    - Ordnen Sie die Ausgabe nach der mittleren Antwortzeit für die Namensgebung an.

## Session Info

Erstellt mit R version 4.3.0 (2023-04-21) (Already Tomorrow) und RStudioversion 2023.9.0.463 (Desert Sunflower).

```
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; 

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] datasauRus_0.1.6 patchwork_1.1.3  janitor_2.2.0    here_1.0.1
[5] lubridate_1.9.2 forcats_1.0.0    stringr_1.5.0    dplyr_1.1.3
[9] purrr_1.0.2     readr_2.1.4     tidyverse_2.0.0
[13] ggplot2_3.4.3

loaded via a namespace (and not attached):
[1] gtable_0.3.4      xfun_0.39       lattice_0.21-8   tzdb_0.4.0
[5] vctrs_0.6.3       tools_4.3.0      generics_0.1.3    parallel_4.3.0
[9] fansi_1.0.4       pacman_0.5.1    pkgconfig_2.0.3   Matrix_1.5-4
[13] webshot_0.5.4    lifecycle_1.0.3 compiler_4.3.0    farver_2.1.1
[17] munsell_0.5.0    snakecase_0.11.0 htmltools_0.5.5  yaml_2.3.7
[21] pillar_1.9.0     crayon_1.5.2    nlme_3.1-162     tidyselect_1.2.0
[25] rvest_1.0.3      digest_0.6.33   stringi_1.7.12   labeling_0.4.3
[29] splines_4.3.0    rprojroot_2.0.3 fastmap_1.1.1    grid_4.3.0
[33] colorspace_2.1-0 cli_3.6.1      magrittr_2.0.3   utf8_1.2.3
[37] withr_2.5.0      scales_1.2.1    bit64_4.0.5     timechange_0.2.0
[41] rmarkdown_2.22    httr_1.4.6     bit_4.0.5       hms_1.1.3
[45] kableExtra_1.3.4 evaluate_0.21   knitr_1.44     viridisLite_0.4.2
[49] mgcv_1.8-42      rlang_1.1.1     glue_1.6.2     xml2_1.3.4
[53] svglite_2.1.1    rstudioapi_0.14 vroom_1.6.3    jsonlite_1.8.7
[57] R6_2.5.1         systemfonts_1.0.4

```

## Literaturverzeichnis

- Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*.
- Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>
- Davies, R., Locke, S., & D'Agostino McGowan, L. (2022). *datasauRus: Datasets from the Datasaurus Dozen*. <https://CRAN.R-project.org/package=datasauRus>
- Müller, K. (2020). *here: A Simpler Way to Find Your Files*. <https://CRAN.R-project.org/package=here>
- Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills*. Zenodo. <https://doi.org/10.5281/zenodo.6365078>
- Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2022). Data Visualization Using R for Researchers Who Do Not Use R. *Advances in Methods and Practices in Psychological Science*, 5(2), 251524592210746. <https://doi.org/10.1177/25152459221074654>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund,

- G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. <https://doi.org/10.21105/joss.01686>
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2. Aufl.).
- Winter, B. (2019). Statistics for Linguists: An Introduction Using R. In *Statistics for Linguists: An Introduction Using R*. Routledge. <https://doi.org/10.4324/9781315165547>
- Xie, Y. (2023). *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>

# 9. Data Wrangling 2

Datenbereinigung (Data tidying)

## Lernziele

In diesem Kapitel werden wir lernen...

- über breite versus lange Daten
- wie man breite Daten länger machen kann
- wie man lange Daten breiter macht

## Lesungen

Die **Pflichtlektüre** zur Vorbereitung auf dieses Thema ist [Kapitel 6 \(Data tidying\)](#) in Wickham et al. (2023).

Eine **ergänzende Lektüre** ist [Kapitel 8 \(Data tidying\)](#) in Nordmann & DeBruine (2022).

### 9.1. Wiederholung

Im letzten Kapitel haben wir etwas über deskriptive Statistik gelernt, insbesondere über Maße der zentralen Tendenz (Mittelwert, Median, Modus) und der Streuung (Bereich, Standardabweichung). Wir haben auch gesehen, wie man diese Werte mit Base R (z. B. `mean()`, `sd()`) und der `tidyverse` (z. B. `summarise()`) und nach Gruppen (`summarise(.by = )`) berechnet.

In diesem Kapitel werden wir das Konzept der aufgeräumten Daten besprechen und sehen, wie wir unsere Daten organisieren und neu anordnen können, damit sie aufgeräumt sind.

## 9.2. Einrichtung

Wir brauchen die Pakete `tidyverse`, `here`, und `janitor`.

```
pacman::p_load(tidyverse,  
                here,  
                janitor)
```

We'll use the `languageR_english.csv` dataset (in `daten` folder).

```
df_eng <- read_csv(here("daten", "languageR_english.csv")) |>  
  clean_names() |>  
  arrange(word) |>  
  rename(  
    rt_lexdec = r_tlexdec,  
    rt_naming = r_tnaming  
)
```

## 9.3. ‘Tidy’ Arbeitsablauf

Abbildung 9.1 zeigt einen Überblick über den typischen Data-Science-Prozess, bei dem wir unsere Daten importieren, sie bereinigen und dann einen Zyklus von Umwandlung, Visualisierung und Modellierung durchlaufen, bevor wir schließlich unsere Ergebnisse mitteilen.

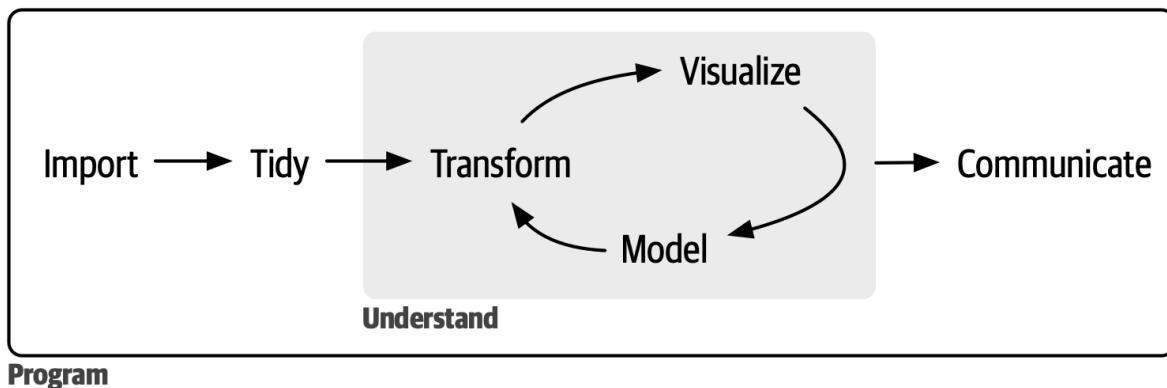


Abbildung 9.1.: [Image source](#): Wickham et al. (2023) (all rights reserved)

Wir haben bereits gesehen, wie wir unsere Daten importieren (`readr::read_csv`), transformieren (Paket `dplyr`) und visualisieren (Paket `ggplot`) können. Aber wir haben

Tabelle 9.1.: Tabelle 1

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

bisher nur aufgeräumte Daten gesehen, so dass wir den Schritt “aufräumen” nicht durchführen mussten.

## 9.4. ‘Tidy’ Daten

Dieselben Daten können auf verschiedene Weise dargestellt werden. Die folgenden Datensätze zeigen alle dieselben Werte für vier Variablen: Land, Jahr, Bevölkerungszahl und Anzahl der Tuberkulosefälle. Jeder Datensatz ordnet die Werte anders an. Nehmen Sie sich einen Moment Zeit, um die verschiedenen Optionen zu betrachten. Welche ist am einfachsten zu lesen?

Wahrscheinlich ist Tabelle 9.1 für Sie am einfachsten zu lesen. Das liegt daran, dass sie den drei Regeln für aufgeräumte Daten folgt (visualisiert in Abbildung 9.2):

1. Jede Variable ist eine Spalte, jede Spalte ist eine Variable
2. Jede Beobachtung ist eine Zeile, jede Zeile ist eine Beobachtung
3. Jeder Wert ist eine Zelle, jede Zelle ist ein Einzelwert

In Tabelle 9.1 steht jede Spalte für eine Variable: `country`, `year`, `population` und `case`. Jede Zeile steht für eine einzelne Beobachtung: ein Land in einem bestimmten Jahr. Und schließlich enthält jede Zelle einen einzigen Wert.

### 9.4.1. Warum ‘tidy’ Daten?

“Glückliche Familien sind alle gleich; jede unglückliche Familie ist auf ihre eigene Art unglücklich.” — Leo Tolstoy

Tabelle 9.2.: Tabelle 2

country	year	type	count
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

Tabelle 9.3.: Tabelle 3

country	year	rate
Afghanistan	1999	745/19987071
Afghanistan	2000	2666/20595360
Brazil	1999	37737/172006362
Brazil	2000	80488/174504898
China	1999	212258/1272915272
China	2000	213766/1280428583

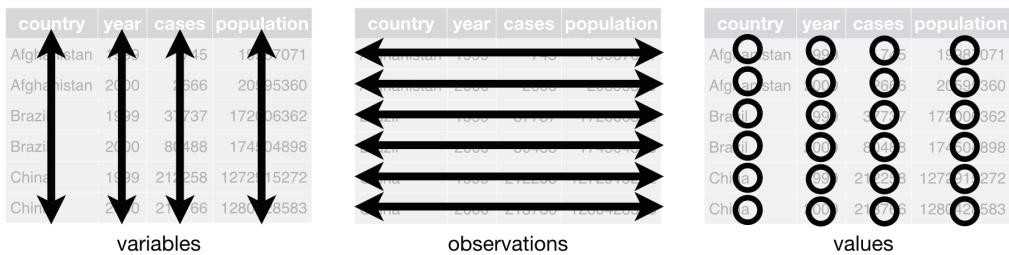


Abbildung 9.2.: [Image source](#): Wickham et al. (2023) (all rights reserved)

“**Saubere Datensätze** sind alle gleich, aber jeder **unordentliche Datensatz** ist auf seine eigene Weise unordentlich.” — Hadley Wickham

Wenn Sie erst einmal ‘tidy’ Daten haben, verbringen Sie weniger Zeit mit dem Versuch, Ihre Daten in die richtige Form zu bringen, um das zu tun, was Sie wollen. Das Aufräumen von Daten erfordert im Vorfeld etwas Arbeit, ist aber langfristig gesehen hilfreich.

Die Arbeit mit aufgeräumten Daten hat zwei wesentliche Vorteile:

1. Die Arbeit mit einer einheitlichen Datenstruktur ermöglicht es uns, Konventionen zu übernehmen.
  - Da aufgeräumte Daten die allgemein vereinbarte Datenstruktur sind, basieren die Konventionen auf der Annahme dieser Struktur.
  - so haben die Werkzeuge eine zugrunde liegende Einheitlichkeit
2. Die vektorisierte Natur von R kann glänzen
  - die meisten eingebauten R-Funktionen arbeiten mit *Vektorwerten* (und Spalten sind im Wesentlichen Vektoren)
  - Alle Pakete im `tidyverse` sind darauf ausgelegt, mit aufgeräumten Daten zu arbeiten (z.B. `ggplot2` und `dplyr`)

### 💡 Rückblick: Vektoren

Vektoren sind der grundlegendste Datenobjekttyp in R. Ein Vektor enthält Daten desselben Typs und ist im Wesentlichen eine Liste. Sie können einen Vektor z.B. mit der Funktion `c()` erzeugen.

```
vector1 <- c(2, 3, 4, 6, 7)
vector2 <- c(2, 3, 4, 6, "x")
```

`vector1` enthält numerische Werte, da alle Elemente Zahlen sind. `vector2` wird alle

Zeichenwerte (d.h. Text) enthalten, da es ein einziges eindeutiges Zeichenelement ("x") gibt. R liest also alle Elemente als Zeichentyp. Wir können einen Datenrahmen aus Vektoren gleicher Länge erstellen, indem wir z. B. die Funktion `tibble()` verwenden.

```
tibble(vector1,vector2)

# A tibble: 5 x 2
  vector1 vector2
  <dbl> <chr>
1     2 2
2     3 3
3     4 4
4     6 6
5     7 x
```

Die meisten Daten "in freier Wildbahn" sind unordentlich. Die Daten werden oft zunächst für ein anderes Ziel als die Analyse organisiert. Dieses Ziel ist in der Regel die Erleichterung der Dateneingabe: Wir wollen unsere Beobachtungen zunächst einfach dokumentieren können. Die meisten Menschen sind mit den Grundsätzen ordentlicher Daten nicht vertraut, und erst wenn sie viel Zeit mit Daten verbringen, wird klar, warum ordentliche Daten notwendig sind. Das bedeutet, dass die meisten *echten* Analysen zumindest ein gewisses Maß an Aufräumen erfordern.

#### Aufgabe 9.1: Tidy data

##### Beispiel 9.1.

1. Gehen Sie zurück zu den Tabellen 1-3. Beschreiben Sie für jede Tabelle, was jede Beobachtung und jede Spalte darstellt.
2. Skizzieren Sie das Verfahren, mit dem Sie die Rate für `table1`. berechnen würden. Sie brauchen nur ein Verb, das:
  - eine neue Variable erzeugt (nennen Sie sie `rate`), die Folgendes enthält:
    - die Anzahl der TB-Fälle (`cases`) pro Land und Jahr, geteilt durch
    - die entsprechende Bevölkerung (`population`) pro Land und Jahr,
    - multipliziert mit 10000
  - Hinweis: Welches `dplyr`-Verb erzeugt neue Variablen? (Sehen Sie sich das Kapitel 4 an.)
3. Schauen Sie die Tabelle 9.2 und Tabelle 9.3 an. Wäre es so einfach gewesen, die `rate` mit diesen Datenstrukturen zu berechnen?

## 9.5. Datenbereinigung

Die Datenbereinigung besteht im Wesentlichen aus der Umwandlung breiter Daten in lange Daten und langer Daten in breite Daten (neben anderen Schritten). Das Ergebnis sind aufgeräumte Daten, bei denen jede Spalte eine Variable und jede Zeile eine Beobachtung darstellt. Wie genau wir eine Beobachtung definieren, hängt davon ab, was genau wir erreichen wollen, und kann sich von einem Analyseschritt zum anderen ändern.

### 9.5.1. Datenaufräumung mit dem tidyverse

Das Paket `tidyr` aus `tidyverse` hat zwei nützliche Funktionen zum Transponieren unserer Daten:

- `pivot_longer()`: macht breite Daten länger
- `pivot_wider()`: lange Daten breiter machen

Oft müssen wir zwischen diesen Formaten konvertieren, um verschiedene Arten von Zusammenfassungen oder Visualisierungen zu erstellen. Aber was genau sind breite und lange Daten?



Abbildung 9.3.: die berühmteste Verwendung des Wortes Pivot (zumindest für Millenials)

### 9.5.2. Breite versus lange Daten

Bei breiten Daten befinden sich alle Beobachtungen zu einer Sache in derselben Zeile. Breite Daten sind *normalerweise* nicht aufgeräumt. Bei langen Daten befindet sich jede Beobachtung in einer eigenen Zeile. Lange Daten sind *normalerweise* aufgeräumt. Beginnen wir mit dem typischsten Fall: der Umwandlung breiter Daten in lange Daten.

Tabelle 9.4.: df\_eng

age_subject	word	length_in_letters	written_frequency	word_category	rt_lexdec	rt_naming
young	ace	3	4.219508	N	623.61	456.3
old	ace	3	4.219508	N	775.67	607.8
young	act	3	8.118207	V	617.10	445.8
old	act	3	8.118207	V	715.52	639.7
young	add	3	7.319203	V	575.70	467.8
old	add	3	7.319203	V	742.19	605.4

## 9.6. Verlängern von Daten: df\_eng

- im Datensatz languageR\_english.csv
  - jede Zeile ist eine Beobachtung
  - die erste Spalte beschreibt die Altersgruppe des Teilnehmers
  - die Spalten `word`, `length_in_letters`, `written_frequency` und `word_category` beschreiben Eigenschaften des Stimulus für eine bestimmte Beobachtung (d. h. das Wort)
  - wir haben 4568 Beobachtungen

```
df_eng %>%
  head() %>%
  knitr::kable() %>%
  kableExtra::kable_styling()
```

- Sind diese Daten in Tabelle 9.4 aufgeräumt?
- Sind diese Daten zu breit oder zu lang?
- Wie können wir diese Daten länger machen?

Ob wir diese Daten verlängern wollen oder nicht, hängt von der jeweiligen Aufgabe ab. Wenn wir die Antwortzeiten für die lexikalische Entscheidungsaufgabe (`rt_lexdec`) zusammen mit der Antwortzeit für die Benennungsaufgabe (`rt_naming`) aufzeichnen wollen, könnten wir die beiden in `facet_wrap()` einschließen. Allerdings nimmt `facet_wrap()` eine *kategorische* Variable als Argument und erzeugt Diagramme für jede Kategorie. Wir bräuchten eine neue Variable, zum Beispiel `response`, die die Stufen `lexdec` und `naming` enthält, und eine weitere, zum Beispiel `time`, die die Antwortzeit enthält. Versuchen wir das mal.

### 9.6.1. pivot\_longer()

Die Funktion `tidyR pivot_longer()` konvertiert eine breite Datentabelle in ein längeres Format, indem sie die Überschriften der angegebenen Spalten in die Werte neuer Spalten

umwandelt und die Werte dieser Spalten zu einer neuen, zusammengefassten Spalte kombiniert.

```
df_eng_long <-  
  df_eng %>%  
  pivot_longer(  
    cols = starts_with("rt_"),  
    names_to = "response",  
    values_to = "time"  
)
```

Die Ausgabe der ersten 12 Zeilen (nach einigen zusätzlichen Formatierungen, um eine hübsche Tabelle zu erstellen) sollte wie Tabelle 9.5 aussehen.

```
df_eng_long %>%  
  head(n = 12) %>%  
  knitr::kable() %>%  
  kableExtra::kable_styling(font_size = 20)
```

Nehmen wir uns einen Moment Zeit, um die Werte in Tabelle 9.5 mit denen der ersten 6 Zeilen in df\_eng zu vergleichen, die in Tabelle 9.4 angegeben sind. Vergleichen Sie die Werte in der df\_eng-Variablen rt\_lexdec (Tabelle 9.4) mit den time-Werten, wenn response rt\_lexdec (Tabelle 9.5) ist: Sie sind identisch. Was ist nun mit rt\_naming sowohl in Tabelle 9.4 als auch in Tabelle 9.5? Sie sind ebenfalls identisch. Dies ist eine wichtige Erkenntnis: Wir haben keine Daten oder Beobachtungswerte geändert, sondern lediglich die Organisation der Datenpunkte neu strukturiert.

Wie hat pivot\_longer() das gemacht? Hier ist eine Aufschlüsselung der Argumente, die pivot\_longer() benötigt (die Sie auch durch Ausführen von ?pivot\_longer in der Konsole untersuchen können):

- **col** = gibt an, welche Spalten gepivotet werden müssen (sollte eine kategorische Variable sein)
  - nimmt die gleiche Syntax wie select(), also könnten wir z.B. starts\_with("") verwenden
- **names\_to** = benennt die Variable, die in den aktuellen Spaltennamen gespeichert ist, hier ist es **response**
- **values\_to** = benennt die in den Zellwerten gespeicherte Variable, die wir **time** nennen
- N.B., wir mussten **response** und **time** in Anführungszeichen setzen, weil sie noch keine Variablennamen sind.

Tabelle 9.5.: A pivoted version of `df_billboard` (first 10 rows)

age_subject	word	length_in_letters	written_frequency	word_
young	ace	3	4.219508	N
young	ace	3	4.219508	N
old	ace	3	4.219508	N
old	ace	3	4.219508	N
young	act	3	8.118207	V
young	act	3	8.118207	V
old	act	3	8.118207	V
old	act	3	8.118207	V
young	add	3	7.319203	V
young	add	3	7.319203	V
old	add	3	7.319203	V
old	add	3	7.319203	V

### 9.6.1.1. Plotten unserer ‘tidy’ Daten

Da wir nun die `response`-Daten in einer Variable und die `time`-Daten in einer anderen Variable haben, wollen wir versuchen, ein Diagramm zu erstellen, in dem wir `age_subject` auf der x-Achse, `time` auf der y-Achse und `response` auf der y-Achse haben.

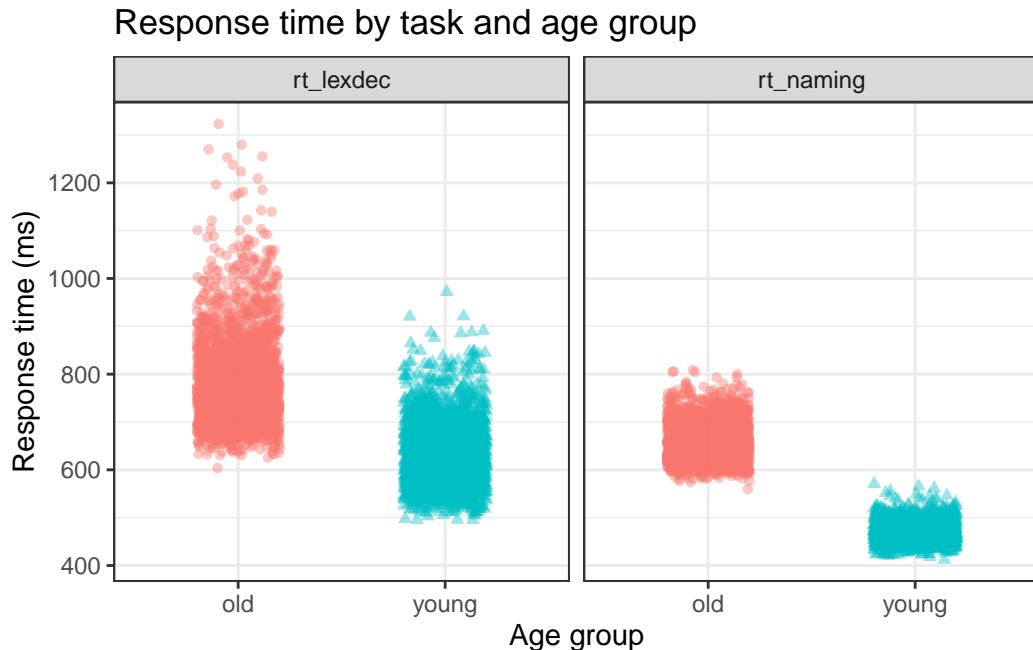


Abbildung 9.4.: Response times per age group for the lexical decision task vs. naming task

Aufgabe 9.1: Tidy data

**Beispiel 9.2.**

Abbildung 9.4 neu erstellen.

## 9.7. Verbreiterung der Daten: df\_eng

Die `tidyverse`-Funktion `pivot_wider()` macht Datensätze *breiter*, indem sie Spalten vergrößert und Zeilen verkleinert. Dies ist hilfreich, wenn eine Beobachtung über mehrere Zeilen verteilt ist. Obwohl diese Art von Daten in der freien Wildbahn nicht sehr häufig vorkommt, ist sie zum Beispiel bei Regierungsdaten ziemlich verbreitet.

Wir können wieder mit `df_eng` beginnen, um die Daten zu erweitern. Zum Beispiel könnten wir eine einzelne Zeile pro *Wort* haben, die eine einzelne Variable für die Antwort des “jungen” Probanden und die Antwort des “alten” Probanden enthält.

Tabelle 9.6.: Wider `df_eng` data

word	length_in_letters	written_frequency	word_category	lex
ace	3	4.219508	N	
act	3	8.118207	V	
add	3	7.319203	V	
age	3	8.397959	N	
aid	3	6.927558	V	
aide	4	4.615120	N	

### 9.7.1. `pivot_wider()`

Pivot wider nimmt ähnliche Argumente wie `pivot_longer()`, mit einigen leichten Unterschieden (z.B. `?pivot_wider`):

- `id_cols`: identifizierende Spalten (welche Spalten identifizieren jede Beobachtung eindeutig?)
- `names_from`: wie soll die neue Spalte heißen, die die vorherigen Spaltennamen enthält (muss eine kategorische Variable sein)?
- `names_prefix`: Präfix für die neuen Spaltennamen (optional)
- `Werte_von`: neue Spaltenwerte

Erstellen wir zwei neue Variablen, die ihre Namen von `age_subject` und ihre Werte von `rt_lexdec` übernehmen. Das Ergebnis sollte wie `?@tbl-eng_wider` aussehen.

```
df_eng_wide <-
  df_eng %>%
  select(-rt_naming) |>
  pivot_wider(
    names_from = age_subject,
    values_from = rt_lexdec,
    names_prefix = "lexdec_"
  )
```

Tabelle 9.7 zeigt wieder die ersten 6 Zeilen des Originaldatensatzes. Wie werden die Daten aus Tabelle 9.6 in Tabelle 9.7 dargestellt?

Tabelle 9.7.: head(df\_eng, n = 6)

age_subject	word	length_in_letters	written_frequency	word_category	rt_lexdec	rt_naming
young	ace	3	4.219508	N	623.61	456.3
old	ace	3	4.219508	N	775.67	607.8
young	act	3	8.118207	V	617.10	445.8
old	act	3	8.118207	V	715.52	639.7
young	add	3	7.319203	V	575.70	467.8
old	add	3	7.319203	V	742.19	605.4

## Lernziele

In diesem Kapitel haben wir gelernt...

- über breite versus lange Daten
- wie man breite Daten länger macht
- wie man lange Daten breiter macht

## 9.8. Hausaufgaben

Für diese Aufgaben werden wir mit dem Datensatz `df_eng` arbeiten.

1. Verwenden Sie `pivot_wider`, um mit `rt_naming` neue Variablen zu erstellen: `naming_old` und `naming_young`, die die Reaktionszeiten beim Benennen für alte bzw. junge Teilnehmer enthalten. Hinweis: Sie müssen `rt_lexdec` entfernen. Der resultierende Datenrahmen sollte 2284 Beobachtungen und 6 Variablen enthalten.
2. Erstellen Sie Abbildung 9.5 neu. Hinweis: Sie benötigen `pivot_wider()`.
3. Warum brauchen wir unseren `df_eng_wide`-Datensatz, um Abbildung 9.5 zu erstellen? Mit anderen Worten, warum ist `df_eng_wide` die geeignete Struktur, aber nicht `df_eng_long` für ein solches Streudiagramm?
4. Benutze `df_eng_long` und die Funktion `summarise()`, die wir im letzten Abschnitt gesehen haben, und reproduziere die folgende Zusammenfassung:

```
# A tibble: 2 x 3
  response   mean    sd
  <chr>     <dbl> <dbl>
1 rt_lexdec 708.  115.
2 rt_naming  566.  101.
```

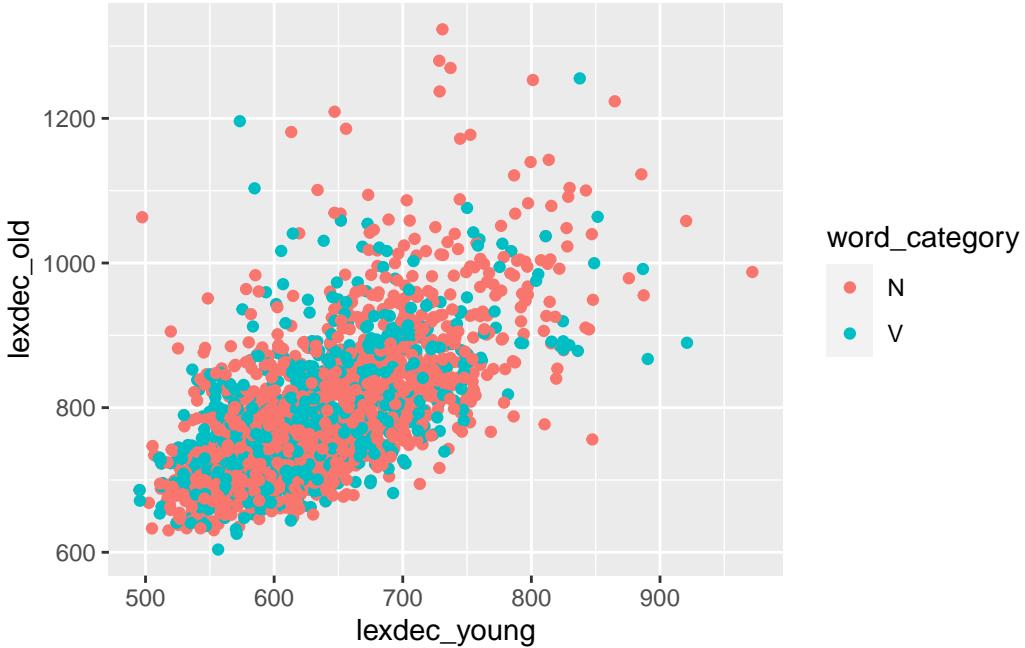


Abbildung 9.5.: Scatterplot of lexical decision task response times per word for old versus young participants

Hinweis: Müssen Sie NA entfernen (wir haben im letzten Kapitel gesehen, wie man das macht)?

## Session Info

Hergestellt mit R version 4.3.0 (2023-04-21) (Already Tomorrow) und RStudioversion 2023.9.0.463 (Desert Sunflower).

```
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;
```

```
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] janitor_2.2.0   here_1.0.1       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] tidyverse_2.0.0  tibble_3.2.1     ggplot2_3.4.3   tidyverse_2.0.0

loaded via a namespace (and not attached):
[1] utf8_1.2.3        generics_0.1.3    xml2_1.3.4      stringi_1.7.12
[5] hms_1.1.3         digest_0.6.33     magrittr_2.0.3   evaluate_0.21
[9] grid_4.3.0        timechange_0.2.0 fastmap_1.1.1    rprojroot_2.0.3
[13] jsonlite_1.8.7    httr_1.4.6       rvest_1.0.3     fansi_1.0.4
[17] viridisLite_0.4.2 scales_1.2.1     cli_3.6.1       rlang_1.1.1
[21] crayon_1.5.2     bit64_4.0.5     munsell_0.5.0   withr_2.5.0
[25] yaml_2.3.7        tools_4.3.0      parallel_4.3.0  tzdb_0.4.0
[29] colorspace_2.1-0  webshot_0.5.4   pacman_0.5.1    kableExtra_1.3.4
[33] vctrs_0.6.3       R6_2.5.1       magick_2.7.4    lifecycle_1.0.3
[37] snakecase_0.11.0  bit_4.0.5      vroom_1.6.3     pkgconfig_2.0.3
[41] pillar_1.9.0      gtable_0.3.4   Rcpp_1.0.11     glue_1.6.2
[45] systemfonts_1.0.4 xfun_0.39     tidyselect_1.2.0 rstudioapi_0.14
[49] knitr_1.44        farver_2.1.1    htmltools_0.5.5 labeling_0.4.3
[53] svglite_2.1.1    rmarkdown_2.22  compiler_4.3.0

```

## Literaturverzeichnis

- Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*.
- Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>
- Davies, R., Locke, S., & D'Agostino McGowan, L. (2022). *datasauRus: Datasets from the Datasaurus Dozen*. <https://CRAN.R-project.org/package=datasauRus>
- Müller, K. (2020). *here: A Simpler Way to Find Your Files*. <https://CRAN.R-project.org/package=here>
- Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills*. Zenodo. <https://doi.org/10.5281/zenodo.6365078>
- Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2022).

- Data Visualization Using R for Researchers Who Do Not Use R. *Advances in Methods and Practices in Psychological Science*, 5(2), 251524592210746. <https://doi.org/10.1177/25152459221074654>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. <https://doi.org/10.21105/joss.01686>
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2. Aufl.).
- Winter, B. (2019). Statistics for Linguists: An Introduction Using R. In *Statistics for Linguists: An Introduction Using R*. Routledge. <https://doi.org/10.4324/9781315165547>
- Xie, Y. (2023). *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>

# 10. Datenvisualisierung 3

Darstellung der zusammenfassenden Statistik

## Lernziele

In diesem Kapitel lernen wir wie man...

- Boxplots zu erstellen und zu interpretieren
- Mittelwerte und Standardabweichungen zu visualisieren

## Ressourcen

Für weitere Lektüre und Übungen zu diesem Thema empfehle ich die Lektüre von [Abschnitt 2.5 \(Visualisierung von Relationen\)](#) in Wickham et al. (2023), [Kapitel 4 \(Darstellung von zusammenfassenden Statistiken\)](#) in Nordmann et al. (2022) und die Abschnitte 3.5-3.9 in Winter (2019).

## Einrichten

### Pakete

Wie üblich laden wir die `tidyverse` Familie von Paketen. Um uns beim Laden unserer Daten zu helfen, laden wir auch das `here`-Paket und das `janitor`-Paket, das nützlich ist, um unsere Daten aufzuräumen (z.B. die `clean_names()`-Funktion). Um unsere Diagramme anzupassen, verwenden wir auch die Pakete `ggthemes` und `patchwork`. Ersteres hilft uns bei der Erstellung von farbenblindfreundlichen Diagrammen, während letzteres uns erlaubt, mehrere Diagramme zusammen zu drucken.

```
pacman::p_load(tidyverse,
  here,
  janitor,
  ggthemes,
  patchwork)
```

## Daten

Wir arbeiten wieder mit unserer leicht veränderten Version des `english`-Datensatzes aus dem `languageR`-Paket. Sie sollten `languageR_english.csv` in Ihrem Daten Ordner haben. Der folgende Code lädt den Datensatz, bereinigt die Namen und korrigiert einige fehlerhafte Namen.

```
df_eng <- read_csv(
  here(
    "daten",
    "languageR_english.csv"
  )
) |>
  clean_names() |>
  rename(
    rt_lexdec = r_tlexdec,
    rt_naming = r_tnaming
  )
```

### 10.1. Rückblick: Visualisierung von Verteilungen

Wir haben bereits mehrere Arten von Diagrammen gesehen, die zur Visualisierung der Verteilung und der Beziehungen zwischen Variablen verwendet werden:

- Histogramme (1 numerische Variable)
- Dichteplots (1 numerische Variable)
- Streudiagramme (2 numerische Variablen)
- Balkendiagramme (kategorische Variablen)

Schauen Sie sich jede Abbildung in Abbildung 10.1 an. Wie viele Variablen werden jeweils dargestellt, und um welche *Typen* von Variablen handelt es sich? Welche zusammenfassende(n) Statistik(en) wird/werden in jedem Diagramm dargestellt?

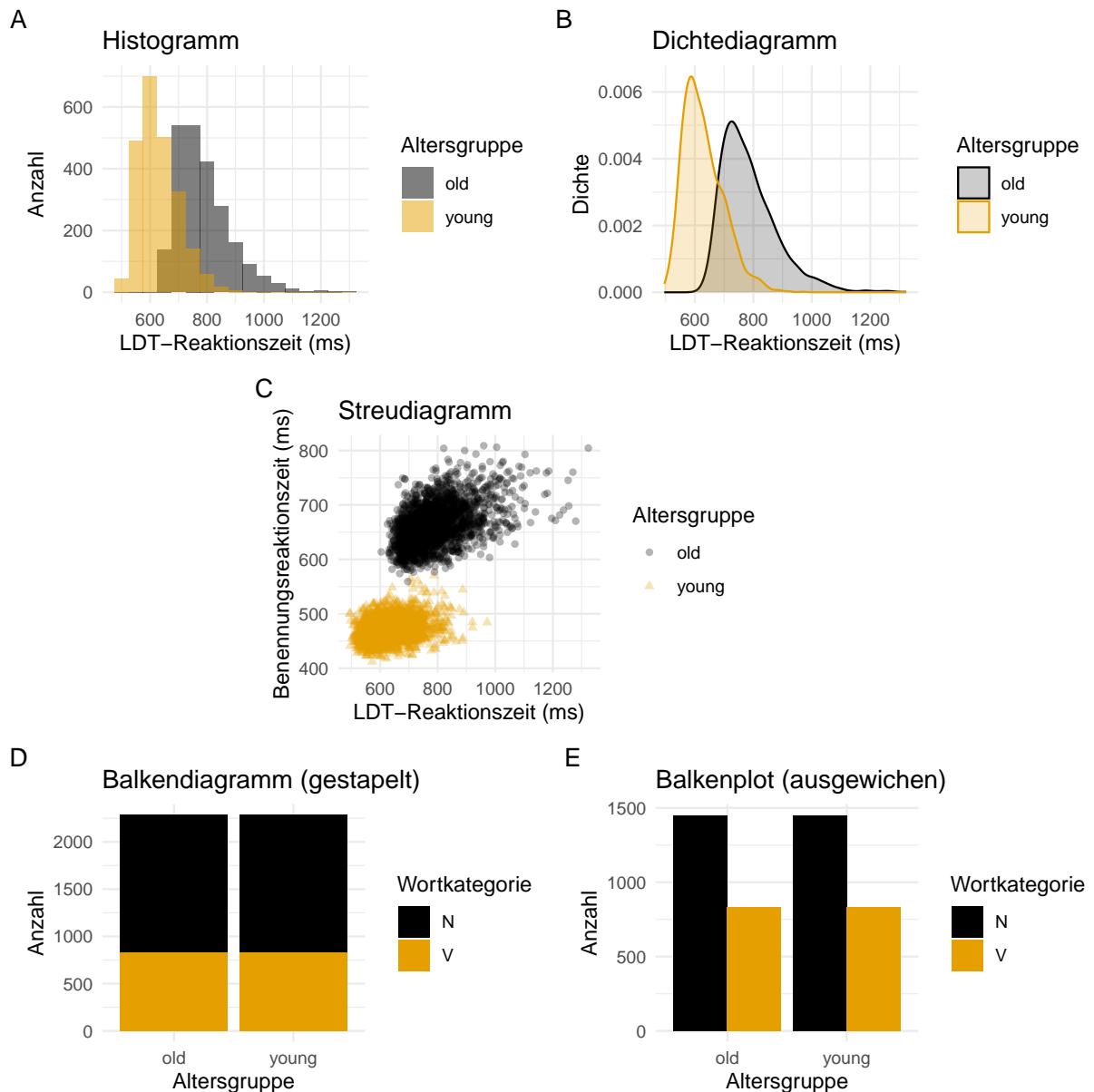


Abbildung 10.1.: Verschiedene Diagrammtypen zur Visualisierung der Verteilung von Rohdaten: Histogramm (A), Dichte-Diagramm (B), Streudiagramm (C), gestapeltes Balkendiagramm (D) und unscharfes Balkendiagramm (E)

## 10.2. Darstellung von zusammenfassenden Statistiken

In Kapitel 8 haben wir etwas über zusammenfassende Statistiken gelernt. Wir behandelten Maße der zentralen Tendenz, nämlich Modus, Median und Mittelwert, sowie Maße der Streuung, wie Bereich und Standardabweichung. Wie können wir zusammenfassende Statistiken visualisieren?

Wir haben bereits gesehen, dass Histogramme, Dichtediagramme und nun auch Geigenplots den Modus (höchster Wert) und den Bereich (niedrigster und höchster Wert) visualisieren. Jetzt lernen wir zwei weitere Arten von Diagrammen kennen, eines zur Darstellung der Verteilung der beobachteten Werte und eines zur Darstellung von Mittelwert und Standardabweichung.

### 10.2.1. Boxplot

Boxplots (manchmal auch Box-and-Whisker-Plots genannt, z. B. Abbildung 10.2) bestehen aus einer Box mit einer Linie in der Mitte (die “Box”) und Linien, die an beiden Enden der Box herausragen (die “Whisker”), sowie manchmal einigen Punkten. Schauen Sie sich Abbildung 10.2 an und identifizieren Sie jeden dieser 4 Aspekte der Darstellung. Kannst du erraten, was jeder dieser Aspekte darstellen könnte und wie du die Darstellung interpretieren solltest?

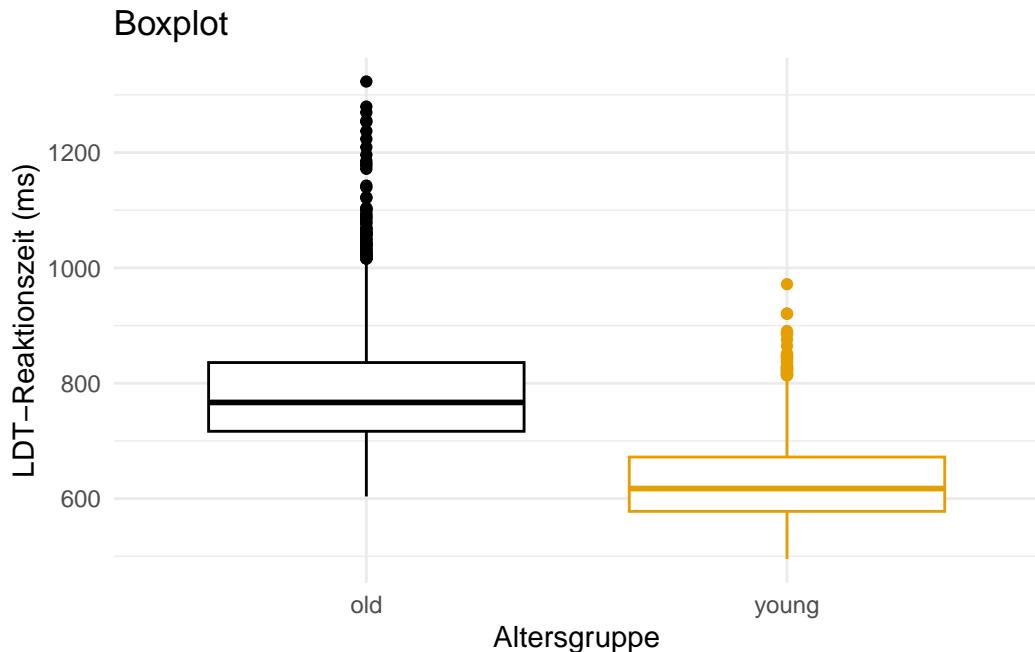


Abbildung 10.2.: Boxplot von df\_eng (Körpermasse nach Alter\_Proband)

Die Box und die Whiskers stellen eine Vielzahl von Informationen in einer einzigen Visualisierung dar. Die Linie in der Mitte des Boxplots stellt den *Median* dar, auch Q2 genannt (2. Quartil; der mittlere Wert, über/unter dem 50% der Daten liegen). Die Box selbst stellt den *Interquartilsbereich* (IQR; der Wertebereich, der zwischen den mittleren 50% der Daten liegt) dar. Die Grenzen der Box stellen Q1 (1. Quartil, unter dem 25% der Daten liegen) und Q3 (3. Quartil, über dem 75% der Daten liegen) dar. Die Whisker stellen  $1,5 \times \text{IQR}$  von Q1 (unterer Whisker) oder Q3 (oberer Whisker) dar. Alle Punkte, die außerhalb der Whisker liegen, stellen Ausreißer dar (d. h. Extremwerte, die außerhalb des IQR liegen).

Abbildung 10.3 zeigt die Beziehung zwischen der Verteilung einer Variablen, wie sie in einem Histogramm dargestellt wird, und einem Boxplot. Während das Histogramm die Balkenhöhe verwendet, um die Anzahl der Beobachtungen innerhalb eines bestimmten Bereichs anzuzeigen, verwendet der Boxplot die Box und die Whiskers, um die Schwellenwerte anzugeben, in denen bestimmte Anteile der Daten enthalten sind (d. h. der Interquartilsbereich).

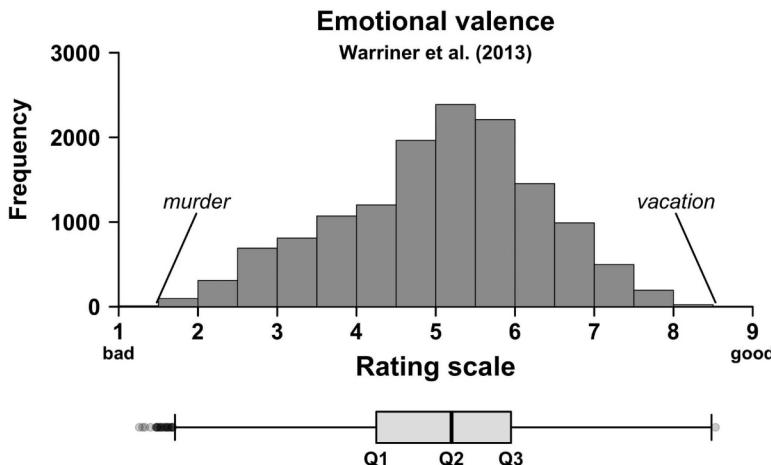


Figure 3.4. A histogram of the emotional valence rating data

Abbildung 10.3.: Image source: Winter (2019) (all rights reserved)

Abbildung 10.4 bietet einen ähnlichen Vergleich, wobei die einzelnen Beobachtungen im Streudiagramm auf der linken Seite hinzugefügt wurden.

Ich hoffe, Sie haben jetzt ein wenig verstanden, wie man Boxplots interpretiert. Man braucht etwas Übung, aber das Wichtigste ist, sich daran zu erinnern, dass die mittleren 50% der Daten in der Box enthalten sind, während die "Schwänze" der Daten durch die "Whisker" dargestellt werden.

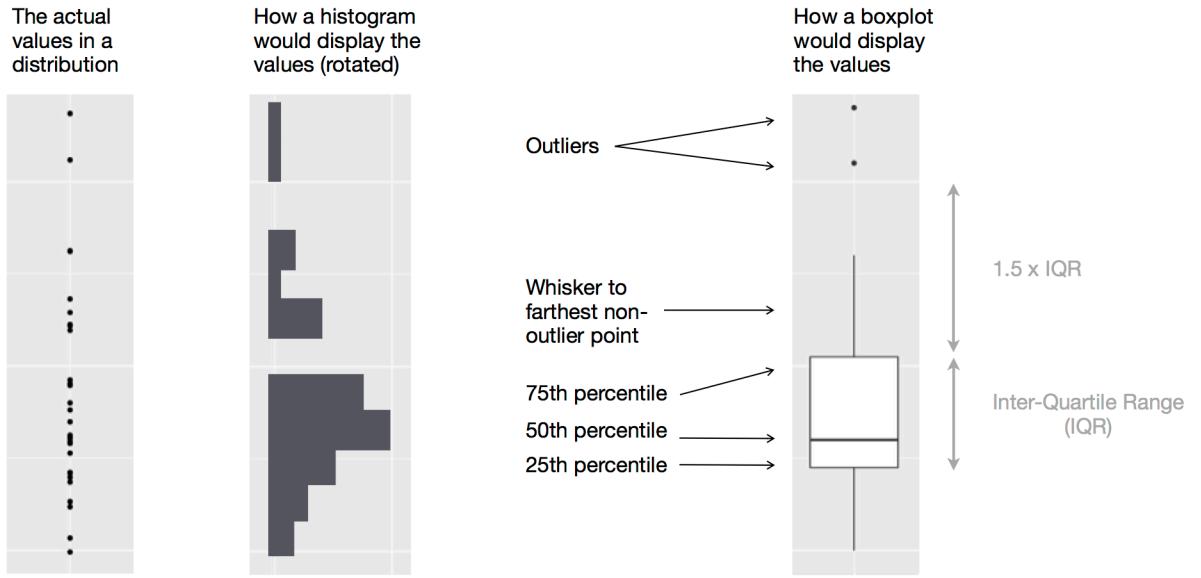


Abbildung 10.4.: Image source: Wickham et al. (2023) (all rights reserved)

### 10.2.1.1. geom\_boxplot()

Wir können Boxplots mit der Funktion `geom_boxplot()` von `ggplot2` erstellen. Zum mindesten müssen wir eine numerische Variable als x oder y Achse angeben (Abbildung 10.5). Wenn wir Boxplots für verschiedene Gruppen erstellen wollen, können wir den Namen einer kategorischen Variable auf der anderen Achse angeben (Abbildung 10.6).

```
1 df_eng |>
2   ggplot(aes(y = rt_lexdec)) +
3     geom_boxplot() +
4     theme_bw() +
5     theme(axis.text.x = element_blank(),
6           axis.ticks.x = element_blank())
```

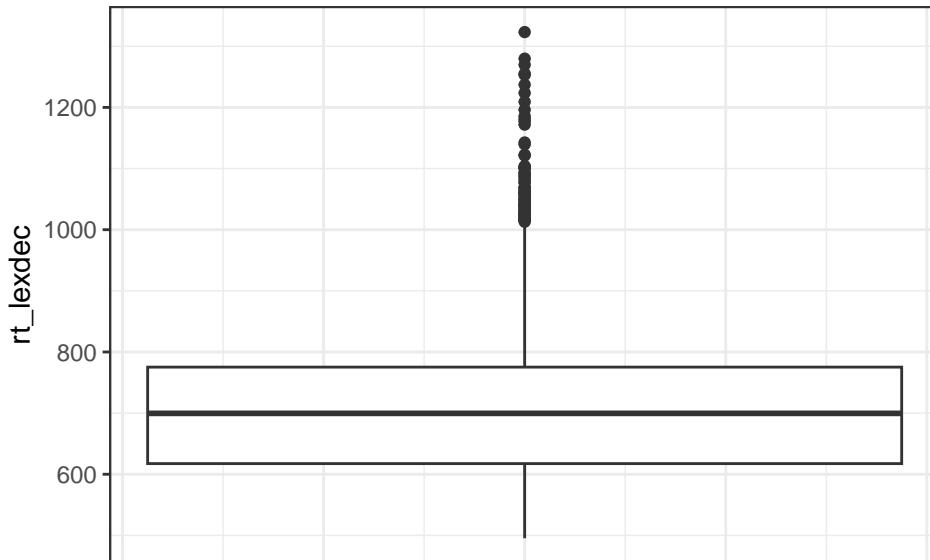


Abbildung 10.5.: Ein Boxplot für alle Beobachtungen einer kontinuierlichen Variablen

```
1 df_eng |>
2   ggplot(aes(x = age_subject, y = rt_lexdec)) +
3     geom_boxplot() +
4     theme_bw()
```

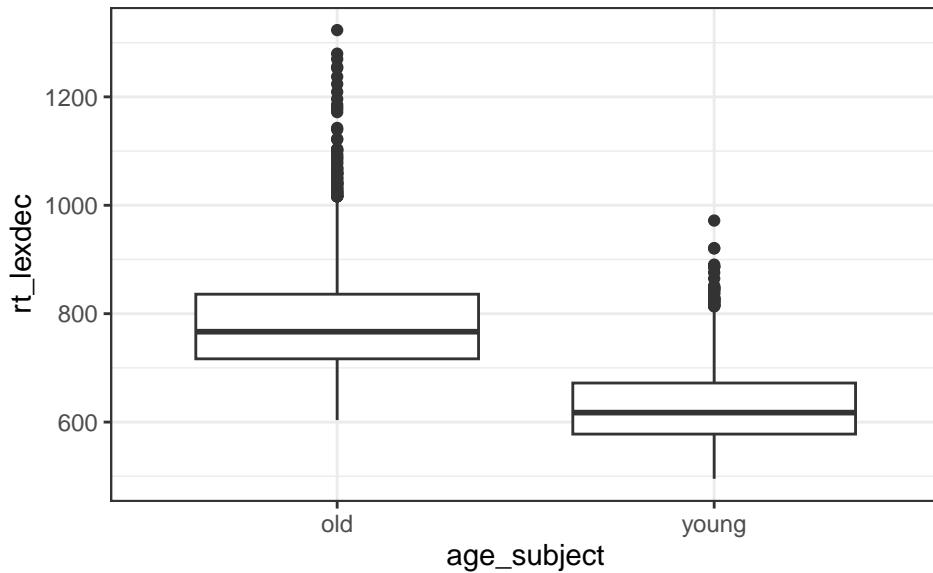


Abbildung 10.6.: Ein Boxplot für zwei Gruppen

### 10.2.1.2. Gruppiertter Boxplot

Genau wie ein Bargraph können wir gruppierte Boxplots erstellen, um mehr Variablen zu visualisieren. Ordnen Sie einfach eine neue Variable mit `colour` oder `fill` ästhetisch zu.

```
df_eng |>
  ggplot(aes(x = age_subject, y = rt_lexdec, colour = word_category)) +
  geom_boxplot() +
  labs(
    x = "Altersgruppe",
    y = "LDT-Reaktionszeit (ms)",
    color = "Wortart"
  ) +
  scale_colour_colorblind() +
  theme_bw()
```

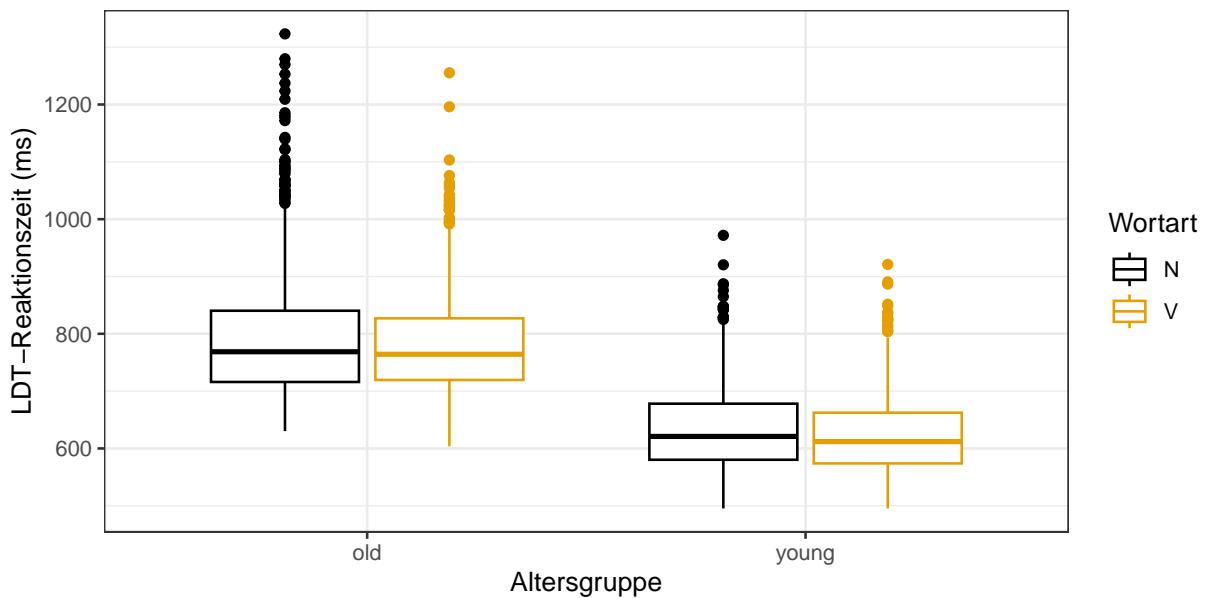


Abbildung 10.7.: Ein gruppiertes Boxplot

### 💡 Boxplots in R

Mit der Funktion `boxplot()`, die einen kontinuierlichen (d.h. numerischen) Vektor als Argument akzeptiert, kann ein Boxplot in der Base-R-Syntax erstellt werden. Da Datenrahmen eine Ansammlung von Vektoren (d.h. die Variablen/Spalten) gleicher Länge sind, können wir auch eine kontinuierliche Variable in einem Datenrahmen als Argument verwenden. Dazu verwenden wir den Subsetting-Operator `$`, der einen Datenrahmen in eine einzelne Variable, in unserem Fall `rt_lexdec`, unterteilt.

```
boxplot(df_eng$rt_lexdec)
```

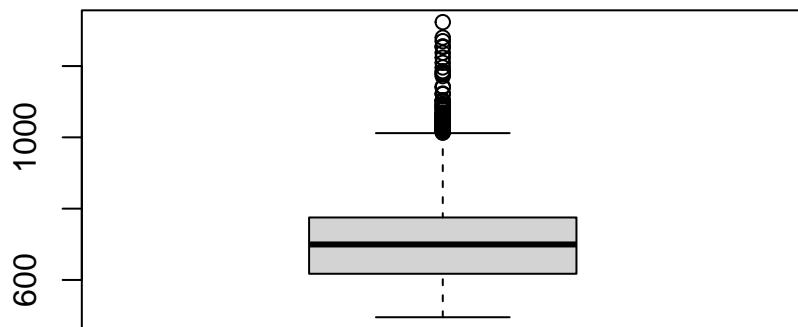


Abbildung 10.8.: Boxplot erstellt mit Basis R

Wir können auch eine kategoriale Variable als “Prädiktor” verwenden, mit der Syntax `kontinuierlich ~ kategorisch`, wobei `~` als “vorhergesagt von” gelesen werden kann.

```
boxplot(df_eng$rt_lexdec ~ df_eng$age_subject)
```

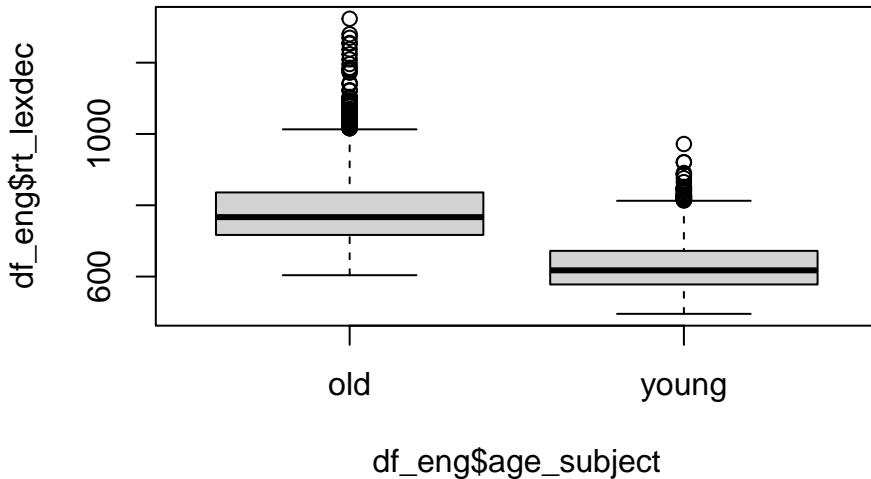


Abbildung 10.9.: Boxplot erstellt mit Basis R

## 10.3. Visualisierung des Mittelwerts

Boxplots zeigen ein Maß für die zentrale Tendenz (Median) und mehrere Maße für die Streuung. In der Regel wird auch der Mittelwert mit der Standardabweichung dargestellt.<sup>1</sup>. Wie könnte man dies tun?

### 10.3.1. Fehlerbalken-Diagramme

Fehlerbalkendiagramme werden üblicherweise verwendet, um den Mittelwert und die Standardabweichung mit Hilfe von Fehlerbalken zu visualisieren. Auch hier werden in der Regel Standardfehler oder Konfidenzintervalle (oder glaubwürdige Intervalle) durch Fehlerbalken dargestellt, die wir in diesem Kurs jedoch nicht behandeln werden. Diese Diagramme bestehen aus zwei Teilen: dem Mittelwert, der mit `geom_point()` dargestellt wird, und der Standardabweichung, die mit `geom_errorbar()` dargestellt wird. Die Fehlerbalken

<sup>1</sup>Anstelle der Standardabweichung werden in der Regel Standardfehler oder Konfidenzintervalle (oder glaubwürdige Intervalle) dargestellt, die jedoch in diesem Kurs nicht behandelt werden. Aus diesem Grund sollten wir im Titel der Grafik vermerken, was die Fehlerbalken darstellen

stellen den Bereich von 1 Standardabweichung über und unter dem Mittelwert dar (Mittelwert  $\pm$  1SD).

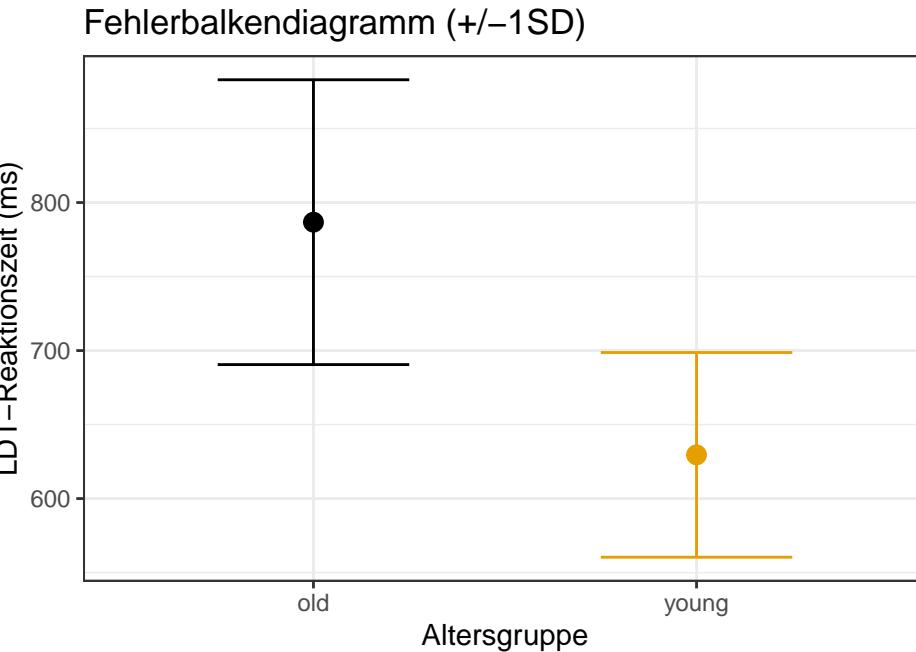


Abbildung 10.10.: Fehlerbalken-Diagramm von df\_eng (Körpermasse nach Alter\_Proband)

Es gibt einige Möglichkeiten, Fehlerbalken-Diagramme zu erstellen, aber wir werden uns auf die Verwendung von `ggplot2` und die Erstellung von zusammenfassenden Statistiken konzentrieren, wie wir sie in Kapitel 8 mit der Funktion `summarise()` von `dplyr` gesehen haben.

#### 10.3.1.1. Berechnung der zusammenfassenden Statistik

Zunächst müssen wir den Mittelwert und die Standardabweichung berechnen, gruppiert nach den Variablen, die wir visualisieren wollen. Bleiben wir bei `rt_lexdec` nach `age_subject`. Wie können wir den Mittelwert und die Standardabweichung von `rt_lexdec` nach `age_subject` berechnen?

```
df_eng |>
  summarise(mean = mean(rt_lexdec),
            sd = sd(rt_lexdec),
            N = n(),
            .by = age_subject) |>
  arrange(age_subject)
```

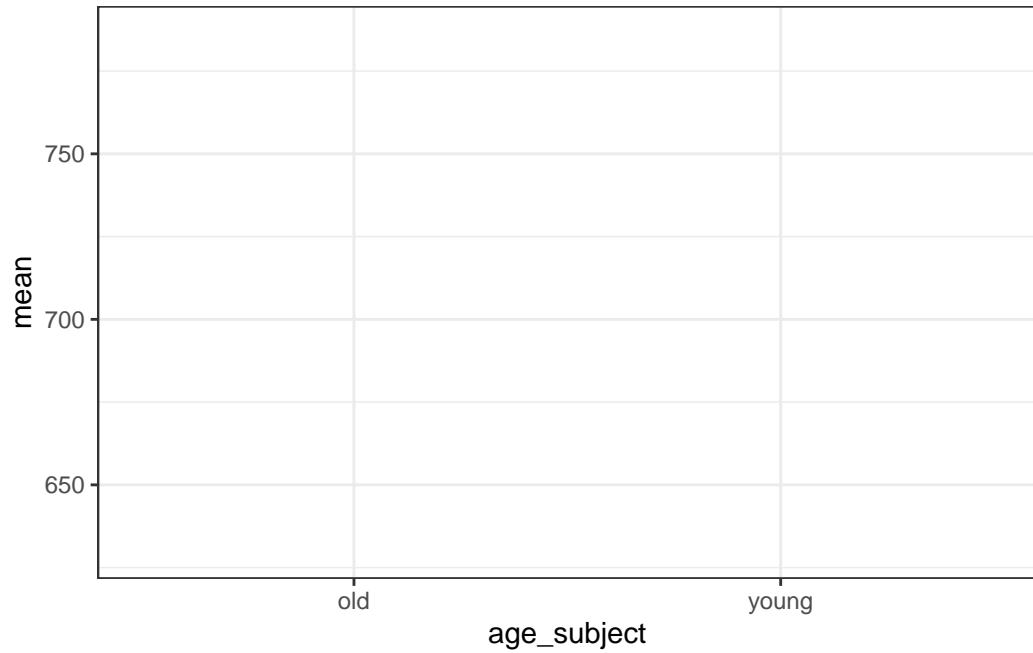
```
# A tibble: 2 x 4
  age_subject  mean     sd     N
  <chr>        <dbl>   <dbl> <int>
1 old           787.   96.2  2284
2 young         630.   69.1  2284
```

Um zusammenfassende Statistiken zu erstellen, können wir entweder den obigen Code direkt in ein `ggplot`-Objekt einfügen, indem wir eine Pipe verwenden, oder wir können die Zusammenfassung als ein Objekt speichern, das wir dann in `ggplot` einfügen. Beide Optionen erzeugen das gleiche Diagramm, wie wir unten sehen.

### 10.3.1.2. Neues Objekt

```
## Neues Objekt mit Zusammenfassungen erstellen
sum_eng <- df_eng |>
  summarise(mean = mean(rt_lexdec),
            sd = sd(rt_lexdec),
            N = n(),
            .by = age_subject) |>
  arrange(age_subject, age_subject)

## Neues Objekt in ggplot einfügen
sum_eng |>
  ggplot(aes(x = age_subject, y = mean, colour = age_subject))
```

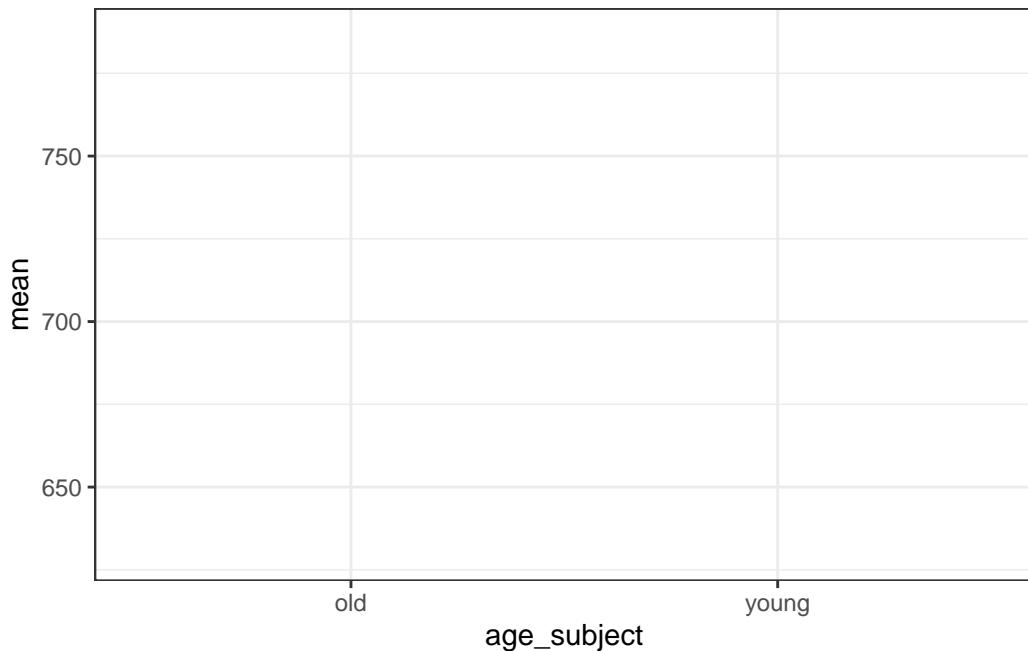


#### 10.3.1.3. With a pipe

```
df_eng |>
  summarise(mean = mean(rt_lexdec),
            sd = sd(rt_lexdec),
            N = n(),
            .by = age_subject) |>
  arrange(age_subject, age_subject) |>
  arrange(age_subject, age_subject) |>
  ggplot() +
  aes(x = age_subject, y = mean, colour = age_subject)
```

Tabelle 10.1.: Formatierte Tabelle von `sum_eng`

Altersgruppe	Mittlere LDT (ms)	SD	N
old	786.7	96.2	2284
young	629.5	69.1	2284

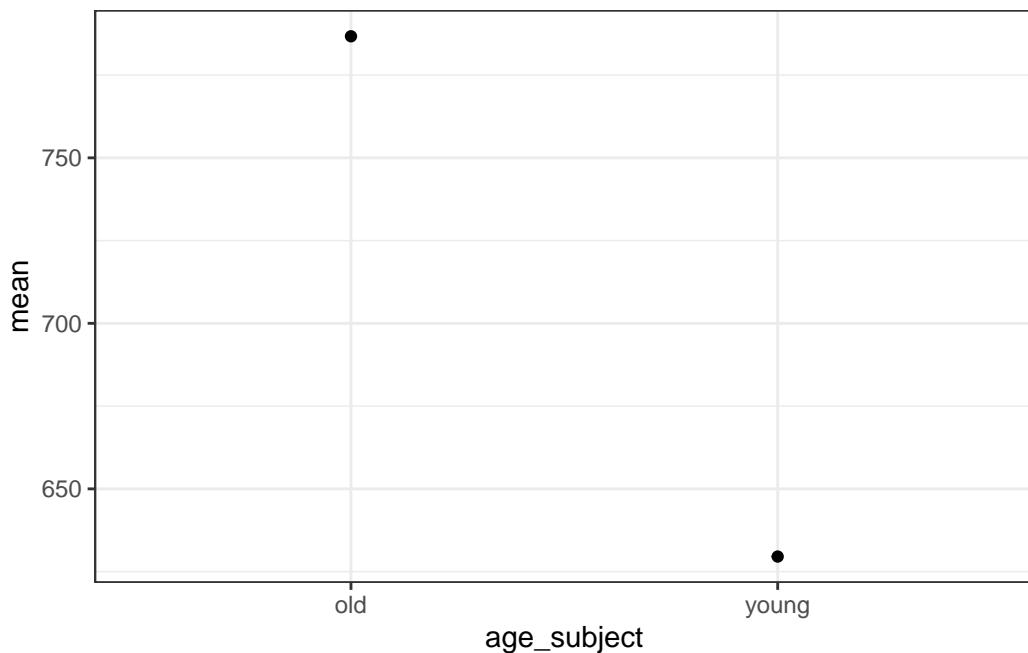


Ich neige dazu, eine Mischung aus diesen beiden Optionen zu verwenden. Manchmal erstelle ich ein neues Objekt und manchmal nicht, je nachdem, was für meinen Arbeitsablauf am sinnvollsten ist. In den Fällen, in denen ich die zusammenfassenden Statistiken auch drucken oder im Auge behalten möchte, würde ich ein Objekt erstellen, das die Zusammenfassung enthält. Dies hat den zusätzlichen Vorteil, dass es mit zusätzlichen Formatierungen gedruckt werden kann, um eine schöne Tabelle zu erstellen (wie Tabelle 10.1).

#### 10.3.1.4. Plotten von Mittelwerten

Aber alles, was wir bis jetzt haben, ist eine leere Leinwand, wir müssen unsere `geoms` hinzufügen. Zuerst fügen wir die Mittelwerte mit `geom_point()` ein.

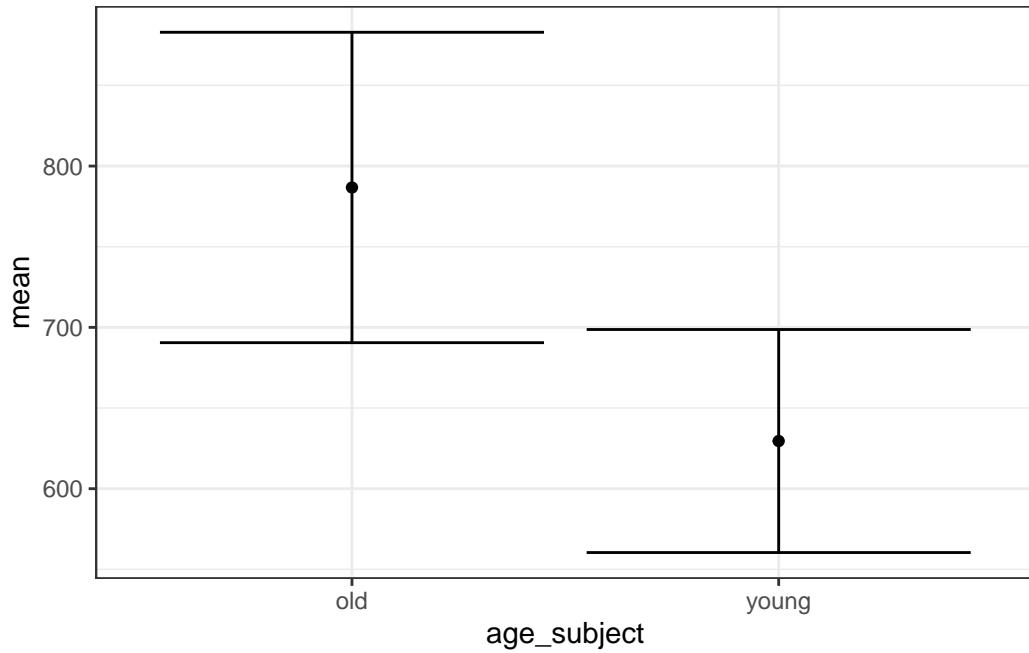
```
sum_eng |>
  ggplot() +
  aes(x = age_subject, y = mean) +
  geom_point()
```



#### 10.3.1.5. Hinzufügen von Fehlerbalken

Fügen wir nun unsere Fehlerbalken hinzu, die eine Standardabweichung über und unter dem Mittelwert darstellen. Wir tun dies mit `geom_errorbar()`, das `ymin` und `ymax` als Argumente benötigt. Diese sind jeweils gleich `mean-/+sd`. Wir haben sie der Übersichtlichkeit halber in einen weiteren `aes()`-Aufruf innerhalb von `geom_errorbar()` eingefügt, aber sie könnten auch im ersten `aes()`-Aufruf erscheinen.

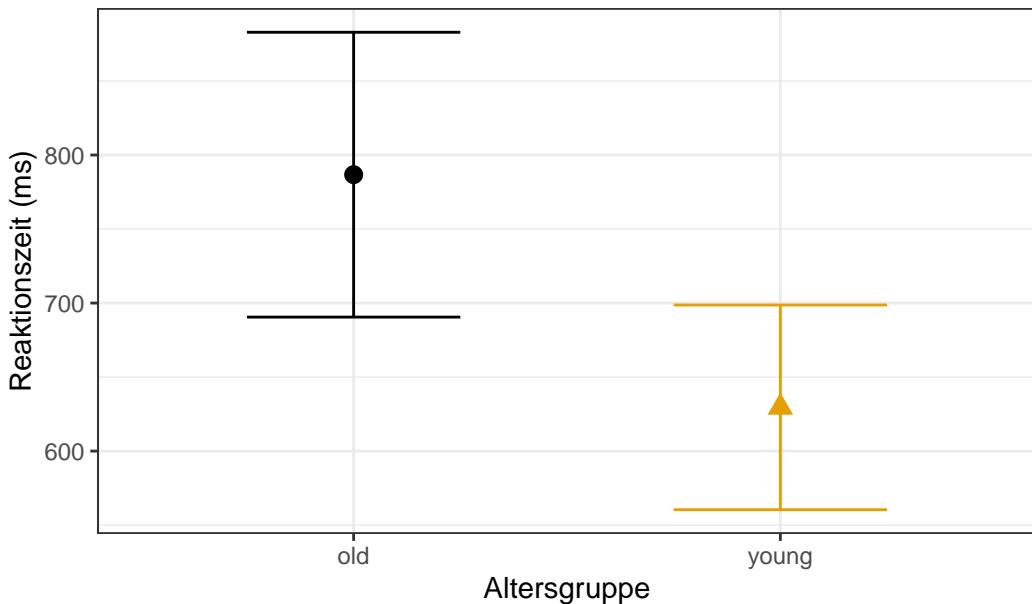
```
sum_eng |>
  ggplot() +
  aes(x = age_subject, y = mean) +
  geom_point() +
  geom_errorbar(aes(ymin = mean-sd,
                     ymax = mean+sd)) +
  theme_bw()
```



Hier sehen wir also den Mittelwert mit +/-1SD für die älteren und jüngeren Teilnehmergruppen. Und wenn wir einige weitere Anpassungen hinzufügen, erhalten wir ?@fig-errorbar-custom.

```
sum_eng |>
  ggplot() +
  aes(x = age_subject, y = mean, colour = age_subject, shape = age_subject) +
  labs(title = "Mittlere LDT-Zeiten (+/-1SD)",
       x = "Altersgruppe",
       y = "Reaktionszeit (ms)",
       color = "Altersgruppe")
  ) +
  geom_point(size = 3) +
  geom_errorbar(width = .5, aes(ymin=mean-sd, ymax=mean+sd)) +
  scale_color_colorblind() +
  theme_bw() +
  theme(
    legend.position = "none"
  )
```

## Mittlere LDT-Zeiten (+/-1SD)



### 💡 stat\_summary()

In `ggplot2` gibt es eine weitere nützliche Funktion, mit der zusammenfassende Statistiken visualisiert werden können, ohne dass zuvor mit `dplyr::summarise()` Zusammenfassungen erstellt werden müssen. Die Funktion `stat_summary()` erlaubt es uns, zusammenfassende Statistiken direkt in unserem `ggplot()`-Objekt zu erstellen, was bedeutet, dass wir mehrere zusammenfassende Statistiken im selben Plot darstellen können (was wir noch nicht tun werden...).

Die Funktion `stat_summary()` benötigt mindestens zwei Argumente: `stat` =, das ist der Typ der Statistik, die man darstellen möchte, und `geom` =, das ist der Typ des `geom`, mit dem man es visualisieren möchte. Wir können Mittelwerte leicht mit Punkten (Abbildung 10.11) oder einem Balkenplot (Abbildung 10.12) darstellen, obwohl ich dringend empfehle, Balkenplots zu vermeiden, wenn ein Punktwert wie ein Mittelwert dargestellt werden soll.

```
df_eng |>
  ggplot() +
  aes(x = age_subject, y = rt_lexdec, colour = age_subject) +
  stat_summary(fun = "mean", geom = "point") +
  labs(title = 'geom = "point"')
```

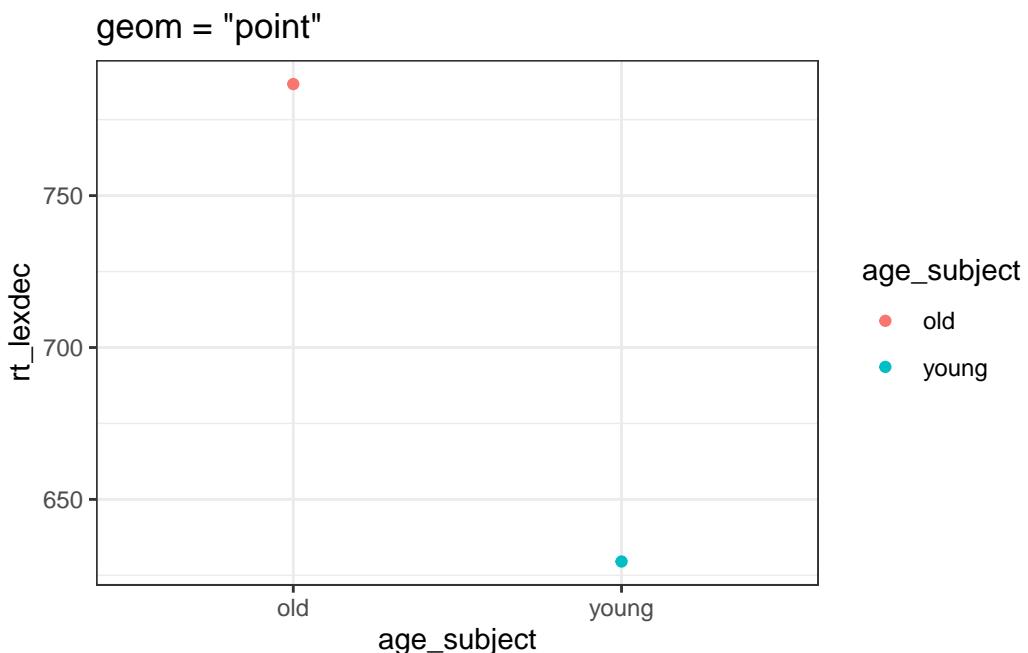


Abbildung 10.11.: `stat_summary(stat_summary(fun = "mean", geom = "point"))`

```
df_eng |>  
  ggplot() +  
  aes(x = age_subject, y = rt_lexdec, fill = age_subject) +  
  stat_summary(fun = "mean", geom = "bar") +  
  labs(title = 'geom = "bar"')
```

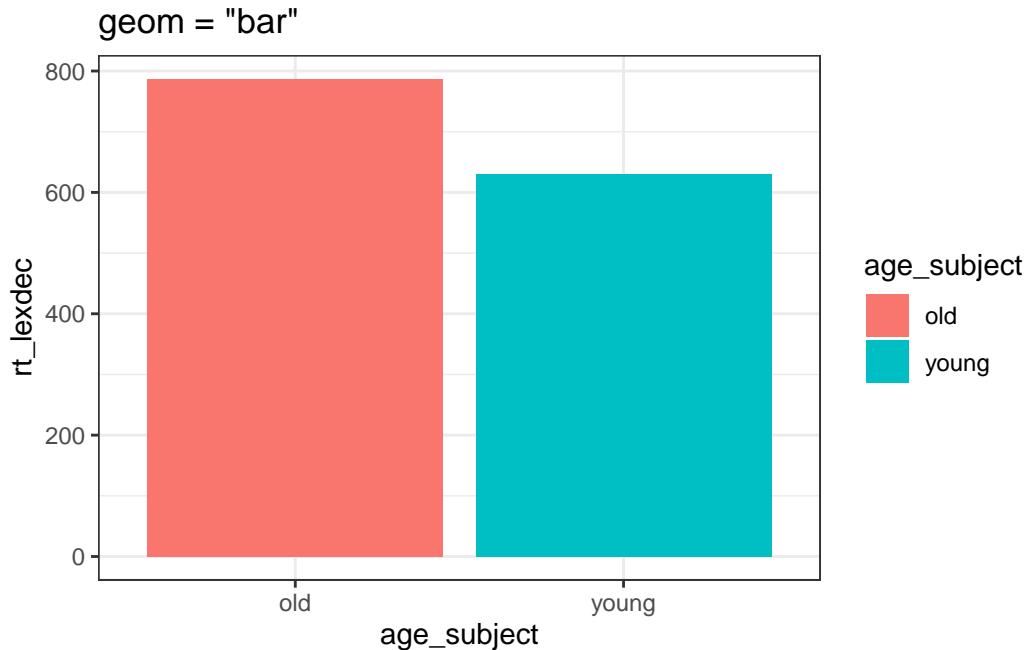


Abbildung 10.12.: `stat_summary(stat_summary(fun = "mean", geom = "bar"))`

Wir können auch Fehlerbalken mit `stat_summary()` einfügen, wie in Abbildung 10.13. Dies erzeugt doppelte Standardabweichungen, so dass wir die `fun.args` -Werte einfügen müssen, um anzugeben, dass wir einfache Standardabweichungen visualisieren wollen.

```
df_eng |>
  ggplot() +
  aes(x = age_subject, y = rt_lexdec, colour = age_subject) +
  stat_summary(fun = "mean", geom = "point") +
  stat_summary(fun.data = "mean_sdl",
              geom = "errorbar",
              fun.args = list(mult = 1)) +
  labs(title = 'Fehlerbalken-Darstellung mit `stat_summary()`') +
  theme_bw()
```

### Fehlerbalken–Darstellung mit ‘stat\_summary()’

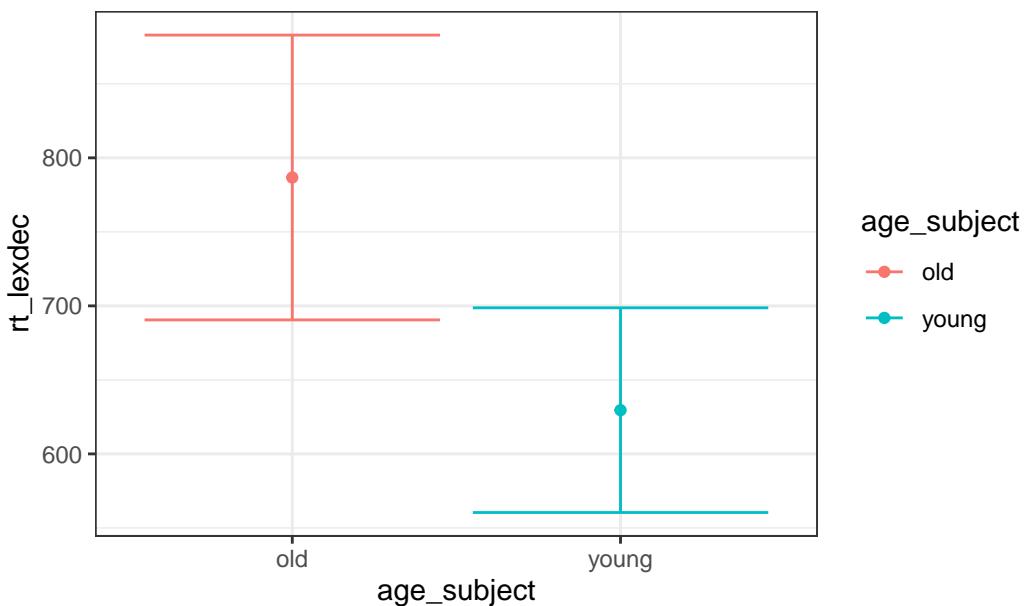


Abbildung 10.13.: `stat_summary(fun.data = "mean_sdl", geom = "errorbar", ...)`

Wie Sie sehen können, ist das Hinzufügen von Fehlerbalken mit `stat_summary()` etwas weniger einfach, weshalb wir uns für den Weg `summarise() |> ggplot() + ...` entschieden haben. Ein zusätzlicher Vorteil der Verwendung von `summarise()` ist, dass Sie Ihre Zusammenfassung als Tibble (d.h. als Tabelle oder Datenrahmen) speichern können, die zusätzlich zum Plot gedruckt werden kann (wie wir mit Tabelle 9.4 gesehen haben). Ich habe erst vor ein oder zwei Jahren begonnen, `dplyr::summarise()` anstelle von `ggplot2::stat_summary()` zu verwenden, und bevorzuge Ersteres, weil ich dann die berechneten Werte vor dem Plotten überprüfen kann. Dies ist eine Frage der persönlichen Vorliebe, wenn Sie also neugierig sind, schlage ich vor, dass Sie `stat_summary()` ausprobieren, um zu sehen, ob Sie eine Vorliebe haben. Wenn Sie mehr über `stat_summary()` erfahren wollen, können Sie `?stat_summary` in der Console eingeben oder nach Tutorials oder YouTube-Videos googeln, es gibt viele davon.

### 10.3.2. Balkendiagramm der Mittelwerte: Finger weg!

Ich flehe Sie an, *nicht* Mittelwerte mit Fehlerbalken darzustellen! Sie werden sehr oft Balkendiagramme von Mittelwerten sehen, und andere unterrichten dies vielleicht sogar in anderen Kursen, aber es gibt viele Gründe, warum dies eine schlechte Idee ist!!!

Erstens können sie sehr irreführend sein. Sie beginnen bei 0 und vermitteln den Eindruck, dass die Daten beim Mittelwert enden, obwohl etwa die Hälfte der Daten (normalerweise) über dem

Mittelwert liegt.

Außerdem hat der Balkenplot ein schlechtes Daten-Tinten-Verhältnis, d. h. die Menge der Datentinte geteilt durch die Gesamttinte, die zur Erstellung der Grafik benötigt wird, oder die Menge der Tinte, die entfernt werden kann, ohne dass Informationen verloren gehen. Beispielsweise beginnen Balkenplots normalerweise bei Null und enden beim Mittelwert. Was aber, wenn es nur sehr wenige oder gar keine Beobachtungen in der Nähe von Null gibt? Wir verbrauchen eine Menge Tinte, wo es keine Beobachtungen gibt! Ein ebenso abscheuliches Verbrechen ist, dass der Balken nur den Bereich abdeckt, in dem die untere *Hälfte* der Beobachtungen liegt; ebenso viele Beobachtungen liegen über dem Mittelwert!

Meiner Meinung nach sollten Balkendiagramme nur für Zählungen oder Häufigkeiten verwendet werden. Abgesehen davon sind Fehlerbalken allein nicht die Lösung. Die Darstellung *nur* des Mittelwerts und der Standardabweichung (oder des Standardfehlers/des Konfidenzintervalls/der glaubwürdigen Intervalle) verbirgt eine Menge Informationen über die tatsächliche Streuung und Verteilung der Daten. Erinnern Sie sich an das Paket `datasauRus`, das Datensätze mit ähnlichen Mittelwerten, Standardabweichungen und Anzahl der Beobachtungen, aber *sehr* unterschiedlichen Verteilungen enthält. Abbildung 10.14 zeigt die Verteilung von 5 dieser Datensätze (A), einen Balkenplot des Mittelwerts, der Standardabweichung und der Anzahl der Beobachtungen für die Variablen “x” und “y” (B) sowie einen Fehlerbalkenplot (C).

Sie werden sehen, dass die Verteilungen sehr unterschiedlich aussehen (in Abbildung 10.14 A), aber Abbildung 10.14 B und C vermitteln das nicht. Aus diesem Grund ist es ein guter Grund, die Rohdatenpunkte *immer* zu visualisieren, unabhängig davon, welche zusammenfassende Darstellung Sie erstellen (z. B. verbergen Errorbar-Plots auch viele Daten). Eine gute Möglichkeit, alle Grundlagen abzudecken, besteht darin, die Verteilung der Daten zusammen mit einer Visualisierung der zusammenfassenden Statistiken darzustellen. Sie werden dies in der Hausaufgabe üben, und in ?@sec-dataviz4 werden wir sehen, wie man diese zusammen in einem Diagramm visualisiert.

## Lernziele

In diesem Kapital haben wir gelernt, wie man...

- Boxplots zu erstellen und zu interpretieren
- Mittelwerte und Standardabweichungen zu visualisieren

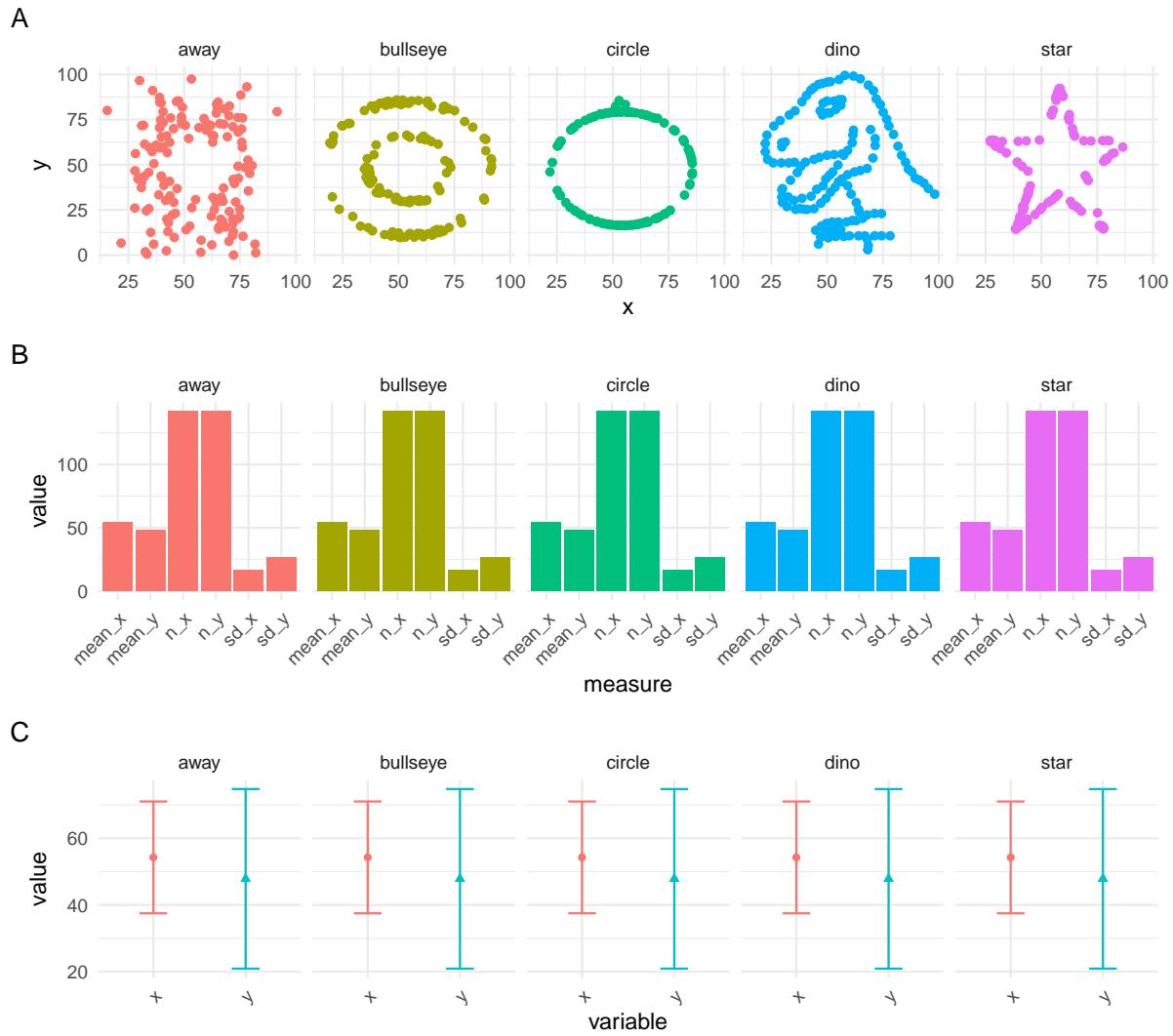


Abbildung 10.14.: Datasets with the same means, sds, and Ns, but very different distributions

## Hausaufgabe

### Boxplot mit Facette

1. Erzeugen Sie einen Plot namens `fig_boxplot`, der ein Boxplot der `df_eng` Daten ist, mit:

- `age_subject` auf der x-Achse
- `rt_naming` auf der y-Achse
- `age_subject` als `colour oder fill` (wähle eine, es gibt keine falsche Wahl)
- `Wort_Kategorie` in zwei Facetten mit `facet_wrap()` aufgetragen
- die von Ihnen gewählte `theme_-Einstellung` (z.B. `theme_bw()`; für weitere Optionen siehe [hier](#))

### Errorbar plot

2. Versuchen Sie, Abbildung [10.15](#) zu reproduzieren. Hinweis: Sie werden die Variable `rt_naming` aus `df_eng` verwenden.

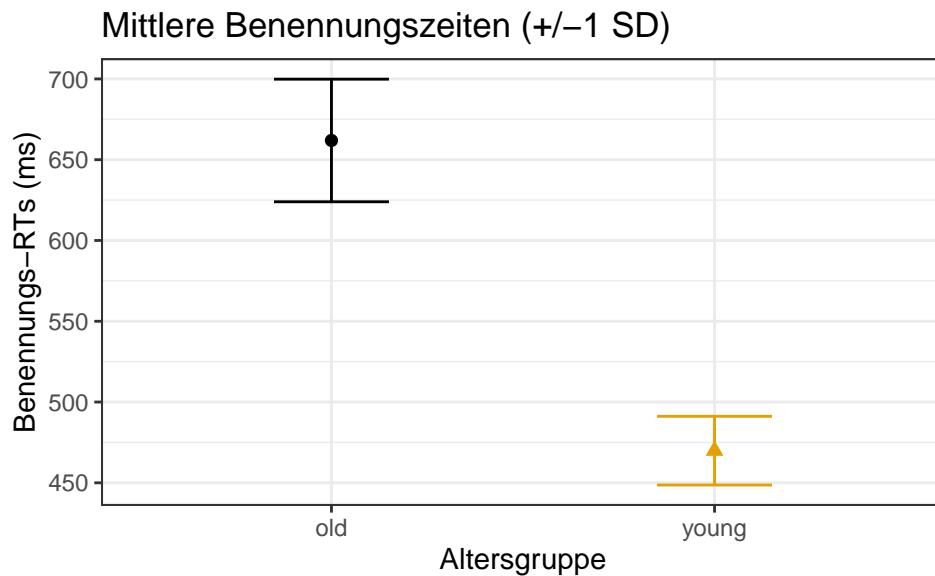


Abbildung 10.15.: Plot to be reproduced

### Patchwork

3. Verwenden Sie das Paket `patchwork`, um Ihren Boxplot und Ihre Fehlerbalkenplots nebeneinander darzustellen. Es sollte ungefähr so aussehen wie Abbildung [10.16](#). Hinweis:

Wenn Sie die "tag-level" ("A" und "B") zu den Plots hinzufügen möchten, müssen Sie +  
`plot_annotation(tag_level = "A")` aus `patchwork` hinzufügen.

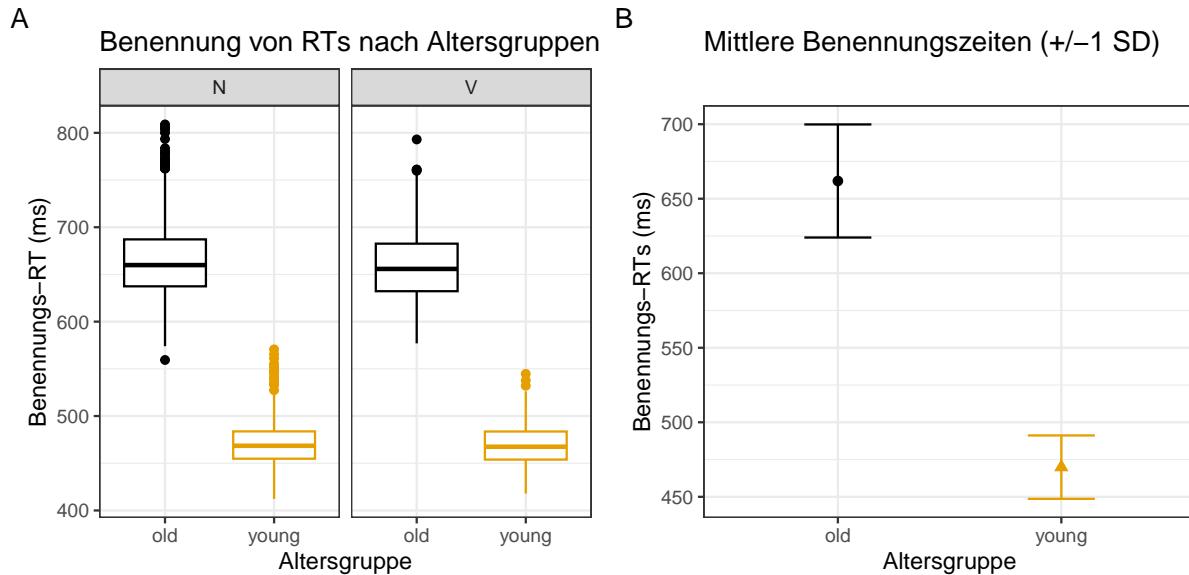


Abbildung 10.16.: Combined plots with `patchwork`

## Session Info

Hergestellt mit R version 4.3.0 (2023-04-21) (Already Tomorrow) und RStudioversion 2023.9.0.463 (Desert Sunflower).

```
print(sessionInfo(), locale = F)
```

```
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

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

other attached packages:
```

```

[1] magick_2.7.4      patchwork_1.1.3 ggthemes_4.2.4   janitor_2.2.0
[5] here_1.0.1       lubridate_1.9.2 forcats_1.0.0   stringr_1.5.0
[9] dplyr_1.1.3      purrr_1.0.2    readr_2.1.4    tidyverse_2.0.0
[13] tibble_3.2.1    ggplot2_3.4.3  tidyverse_2.0.0

loaded via a namespace (and not attached):
[1] gtable_0.3.4      xfun_0.39        htmlwidgets_1.6.2 tzdb_0.4.0
[5] vctrs_0.6.3       tools_4.3.0      generics_0.1.3   parallel_4.3.0
[9] fansi_1.0.4       cluster_2.1.4   pacman_0.5.1    pkgconfig_2.0.3
[13] data.table_1.14.8 checkmate_2.2.0 webshot_0.5.4   lifecycle_1.0.3
[17] compiler_4.3.0   farver_2.1.1   munsell_0.5.0   datasauRus_0.1.6
[21] snakecase_0.11.0 htmltools_0.5.5 yaml_2.3.7     htmlTable_2.4.1
[25] Formula_1.2-5   pillar_1.9.0   crayon_1.5.2   Hmisc_5.1-0
[29] rpart_4.1.19     tidyselect_1.2.0 rvest_1.0.3    digest_0.6.33
[33] stringi_1.7.12   labeling_0.4.3  rprojroot_2.0.3 fastmap_1.1.1
[37] grid_4.3.0       colorspace_2.1-0 cli_3.6.1     magrittr_2.0.3
[41] base64enc_0.1-3  utf8_1.2.3    foreign_0.8-84 withr_2.5.0
[45] backports_1.4.1  scales_1.2.1   bit64_4.0.5    timechange_0.2.0
[49] rmarkdown_2.22    httr_1.4.6    nnet_7.3-18   bit_4.0.5
[53] gridExtra_2.3    png_0.1-8    hms_1.1.3     kableExtra_1.3.4
[57] evaluate_0.21    knitr_1.44   viridisLite_0.4.2 rlang_1.1.1
[61] Rcpp_1.0.11      glue_1.6.2    xml2_1.3.4    svglite_2.1.1
[65] rstudioapi_0.14  vroom_1.6.3   jsonlite_1.8.7 R6_2.5.1
[69] systemfonts_1.0.4

```

## Literaturverzeichnis

- Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*.
- Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>
- Davies, R., Locke, S., & D'Agostino McGowan, L. (2022). *datasauRus: Datasets from the Datasaurus Dozen*. <https://CRAN.R-project.org/package=datasauRus>
- Müller, K. (2020). *here: A Simpler Way to Find Your Files*. <https://CRAN.R-project.org/package=here>
- Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills*. Zenodo. <https://doi.org/10.5281/zenodo.6365078>
- Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2022). Data Visualization Using R for Researchers Who Do Not Use R. *Advances in Methods and Practices in Psychological Science*, 5(2), 251524592210746. <https://doi.org/10.1177/25152459221074654>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund,

- G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. <https://doi.org/10.21105/joss.01686>
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2. Aufl.).
- Winter, B. (2019). Statistics for Linguists: An Introduction Using R. In *Statistics for Linguists: An Introduction Using R*. Routledge. <https://doi.org/10.4324/9781315165547>
- Xie, Y. (2023). *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>

# **Bericht 2**

## **Teil IV.**

# **Fortgeschrittene Themen**

# Literaturverzeichnis

- Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*.
- Baayen, R. H., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>
- Davies, R., Locke, S., & D'Agostino McGowan, L. (2022). *datasauRus: Datasets from the Datasaurus Dozen*. <https://CRAN.R-project.org/package=datasauRus>
- Müller, K. (2020). *here: A Simpler Way to Find Your Files*. <https://CRAN.R-project.org/package=here>
- Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills*. Zenodo. <https://doi.org/10.5281/zenodo.6365078>
- Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2022). Data Visualization Using R for Researchers Who Do Not Use R. *Advances in Methods and Practices in Psychological Science*, 5(2), 251524592210746. <https://doi.org/10.1177/25152459221074654>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. <https://doi.org/10.21105/joss.01686>
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2. Aufl.).
- Winter, B. (2019). Statistics for Linguists: An Introduction Using R. In *Statistics for Linguists: An Introduction Using R*. Routledge. <https://doi.org/10.4324/9781315165547>
- Xie, Y. (2023). *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>