

Datentransformation

Arbeiten mit Zeilen und Spalten

Daniela Palleschi

Mi. den 08.11.2023

Inhaltsverzeichnis

Wiederholung	2
Heutige Ziele	2
1 Voraussetzungen	2
2 Data Wrangling	3
2.1 lexdec	4
2.2 dplyr-Grundlagen	4
3 Zeilen	6
3.1 filter()	6
3.1.1 == und 	7
3.1.2 %in%	8
3.2 arrange()	8
4 Spalten	11
4.1 rename()	11
4.2 mutate()	11
4.3 select()	13
4.4 select()-Hilfsfunktionen	14
4.5 relocate()	16
5 dplyr und ggplot2	17
5.1 Pipe versus plus (> vs. +)	17
5.2 Aufgaben	17
Session Info	18

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 wiedergibt

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?

- [Kapital 4 \(Data transformation\)](#) in ([wickham_r_nodate?](#))
- [Kapital 9 \(Data wrangling\)](#) in Nordmann & DeBruine (2022)

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)
```

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

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 2.1: `df_lexdec`

Beispiel 2.1.

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?

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 Variablennamen 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 Interpunktion.

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)

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

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

```

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 3.1: filter()

Beispiel 3.1.

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

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 3.2: `arrange()`

Beispiel 3.2.

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 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

4.1 `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,
         geschlecht = Sex,
         laenge = Length)
```

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 geschlecht 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 geschlecht 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 4.1: mutate()

Beispiel 4.1.

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

4.3 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	geschlect
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 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 4.2: `select()`

Beispiel 4.2.

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

4.5 relocate()

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

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

	Trial	teilnehmer		rz_ms	rz_s	geschlecht	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	geschlecht	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
```

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()
```

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?

5.2 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 `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 Sie die Variablen `Teilnehmer`, `NativeLanguage`, `Word`, `rz_s`, `laenge`, und `Frequency`
 - Erstelle eine neue Variable `rz_s_laenge`, die `rz_s` geteilt durch `laenge` ist
 - und wird vor `Laenge` gesetzt
 - Benennen 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_1.9.2 forcats_1.0.0  stringr_1.5.0
[5] dplyr_1.1.3     purrr_1.0.2   readr_2.1.4   tidyr_1.3.0
[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., & Shafaei-Bajestan, E. (2019). *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>

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