Übung: Daten transformieren

Am Beispiel der Tagesschau-Daten

Prof. Dr. Nicolas Meseth

Rohdaten sind selten analysebereit. Bevor ihr eine Visualisierung erstellt, müsst ihr auswählen, was ihr braucht, irrelevantes herausfiltern, neue Variablen ableiten, Gruppen zusammenfassen und verschiedene Tabellen verknüpfen. Genau das ist Datentransformation, und `dplyr` liefert dafür ein konsistentes, lesbares Vokabular aus wenigen, aber mächtigen Verben. In dieser Übung lernt ihr diese Verben am Tagesschau-Datensatz kennen und baut dabei Schritt für Schritt ein vollständiges Transformations-Repertoire auf.

Schritt 1: Projekt aufsetzen und Daten einlesen

Die Aufgaben 1 bis 4 sind für euch nur relevant, wenn ihr noch kein Projekt angelegt habt. Wenn ihr bereits ein Projekt für eine vorherige Übung erstellt habt, könnt ihr die Aufgaben 1 bis 3 sowie 5 auslassen und nur das neue R-Skript in Aufgabe 4 anlegen.

1. Erstellt ein neues Projekt für diese Übung und öffnet es in unserer Entwicklungsumgebung Positron. Stellt sicher, dass ihr eine *virtuelle R-Umgebung* für das Projekt erstellt und aktiviert habt.

2. Installiert die folgenden Pakete in eurer virtuellen Umgebung: `tidyverse`, `janitor`, `skimr`, `ggridges` und `tidylog`. Verwendet das Metapaket `pacman`, damit ihr die Pakete mit einem einzigen Befehl installieren und laden könnt.

3. Ladet den Datensatz `tagesschau.zip` auf euren Computer herunter und speichert ihn in einem Unterordner `data/` in eurem Projektverzeichnis.

4. Erzeugt einen neuen Ordner `scripts/` in eurem Projektverzeichnis und erstellt eine neue R-Skriptdatei `transform_news.R` in diesem Ordner.

5. Der Datensatz liegt als komprimierte ZIP-Datei vor. Schaut euch die Datei genauer an und prüft, wie das Format aussieht, zum Beispiel Trennzeichen und Zeichenkodierung. Lest die Daten dann mit einer geeigneten Funktion aus `readr` ein und speichert den Code in eurer Skriptdatei.

Schritt 2: Variablen auswählen und Zeilen filtern

`select()`, `filter()` und `arrange()` sind die drei grundlegenden Verben, um einen Datensatz einzugrenzen: Was brauche ich? Welche Zeilen sind relevant? In welcher Reihenfolge sollen sie erscheinen? Zusammen mit `tidylog` habt ihr außerdem ein Werkzeug, das euch bei jeder Transformation transparent zeigt, was tatsächlich passiert.

6. Das Paket `tidylog` überwacht dplyr-Operationen still im Hintergrund und gibt nach jeder Transformation eine Meldung aus, wie viele Zeilen entfernt wurden, wie viele neue Variablen entstanden sind und wie die Gruppen aussehen. Es muss *nach* dem Laden von `tidyverse` geladen werden, damit es die `dplyr`-Funktionen korrekt überschreibt.

Ladet `tidylog` mit `library(tidylog)` und führt dann probeweise folgende Befehle aus:

```
filter(news, ressort == "inland")
filter(news, word_count > 2000)
select(news, title, ressort, word_count)
```

Was steht in den Meldungen von `tidylog`? Welche Information bekommt ihr, die dplyr allein euch nicht geben würde? In welchen Situationen ist `tidylog` besonders hilfreich?

7. Der Datensatz enthält 21 Variablen, aber für viele Analysen braucht ihr davon nur einen kleinen Teil. Erstellt einen reduzierten Datensatz `news_slim`, der ausschließlich die Variablen `title`, `date_time`, `ressort`, `tag`, `author` und `word_count` enthält. Verwendet dafür `select()`. Beobachtet, was `tidylog` dabei meldet. Wie viele Zeilen hat `news_slim`?

8. `select()` kann Spalten auch umbenennen, indem ihr `neuer_name = alter_name` schreibt. Das ist praktisch, wenn ihr gleichzeitig auswählt und umbenennt. Wenn ihr jedoch *alle* Spalten behalten und nur eine oder wenige umbenennen wollt, ist `rename()` die bessere Wahl.

Benennt in `news_slim` die Spalte `word_count` mit `rename()` in `n_words` um und nennt das Ergebnis `news_renamed`. Wie würde dieselbe Umbenennung mit `select()` aussehen? Was müsstet ihr dabei zusätzlich angeben? Überschreibt `news_slim` *nicht*, da spätere

Aufgaben den ursprünglichen Namen `word_count` erwarten.

9. Filtert `news_slim` auf alle Artikel aus dem Ressort `"inland"` und speichert das Ergebnis als `news_inland`. Wie viele Inland-Artikel enthält der Datensatz? Was sagt `tidylog` dazu?

10. Häufig möchtet ihr nach mehreren erlaubten Werten gleichzeitig filtern. Der Operator `%in%` prüft, ob ein Wert in einem Vektor enthalten ist, und vermeidet mehrere `|`-Bedingungen.

Definiert zunächst einen Vektor `hauptressorts` mit den sechs häufigsten Ressorts: `"inland"`, `"ausland"`, `"wirtschaft"`, `"wissen"`, `"investigativ"` und `"faktenfinder"`. Filtert `news_slim` mit `%in%` auf diese Ressorts und speichert das Ergebnis als `news_haupt`. Wie viele Zeilen enthält es? Schreibt daneben, wie der äquivalente Filter mit `|` aussehen würde, und entscheidet, welche Version ihr lesbarer findet.

11. Wie viele Artikel aus dem Ressort `"ausland"` wurden seit dem 1. Januar 2020 veröffentlicht *und* haben mehr als 800 Wörter? Von welchem Datum stammt der älteste dieser Artikel?

12. Die Variable `author` enthält für viele Artikel keine Angabe. Ermittelt zunächst, welchen Anteil in Prozent die Artikel ohne Autorenangabe am Gesamtdatensatz ausmachen. Erstellt dann eine gefilterte Version `news_with_author`, die nur Zeilen *mit* Autorenangabe enthält.

Was könnte es inhaltlich bedeuten, wenn kein Autor angegeben ist? Beobachtet, wie `tidylog` den NA-Filter beschreibt.

13. Welche zehn Artikel haben die meisten Wörter? Verwendet `arrange()` mit `desc()` und `head()`, oder alternativ `slice_max()`, um die zehn längsten Artikel zu identifizieren. Gebt `title`, `ressort` und `word_count` aus. In welchen Ressorts erscheinen die besonders langen Artikel?

Schritt 3: Neue Variablen erzeugen

Mit `mutate()` könnt ihr bestehende Datensätze um neue, berechnete Variablen erweitern, ohne dabei andere Spalten oder die Zeilenanzahl zu verändern. In diesem Teil lernt ihr außerdem, wie ihr Datentypen anpasst, Datumsvariablen zerlegt, kategoriale Einteilungen vornehmt und mit `across()` dieselbe Transformation auf viele Spalten gleichzeitig anwendet.

14. Die Variable `date_time` enthält sowohl Datum als auch Uhrzeit. Für viele Analysen brauchen wir sie in Einzelteilen. Fügt dem Datensatz `news_slim` mit `mutate()` drei neue Variablen hinzu:

- `year`, das Erscheinungsjahr
- `month`, den Erscheinungsmonat als Zahl von 1 bis 12
- `weekday`, den Wochentag als geordneter Faktor, Funktion: `wday(..., label = TRUE)`

Speichert den erweiterten Datensatz wieder als `news_slim`. Welche Wochentage sind im Datensatz vertreten, und wie heißen sie in R?

15. Die Variable `ressort` wurde beim Einlesen als `character` gespeichert, ist inhaltlich aber eine nominale kategoriale Variable. Wandelt sie mit `mutate()` und `as_factor()` in einen Faktor um. Überprüft mit `levels()`, welche Ausprägungen vorhanden sind, und mit `nlevels()`, wie viele es insgesamt gibt.

Macht dasselbe mit der Variable `tag`. Wie viele verschiedene Tags enthält der Datensatz? Fällt euch dabei etwas Unerwartetes auf?

16. Auch aus Textvariablen können wir nützliche Kenngrößen ableiten. Erstellt mit `mutate()` und Funktionen aus `stringr` zwei neue Variablen:

- `title_length`, die Zeichenanzahl des Titels
- `title_word_count`, die geschätzte Wortanzahl des Titels, zählt dafür die Leerzeichen und addiert 1

Berechnet für `title_length` Minimum, Median und Maximum. Was ist der kürzeste und was der längste Titel im Datensatz? Druckt die Artikel mit dem kürzesten und dem längsten Titel auf der Konsole aus.

17. Die Funktion `transmute()` verhält sich wie `mutate()`, behält aber im Ergebnis nur die neu erzeugten Spalten. Erstellt mit `transmute()` einen neuen kompakten Datensatz `news_timeline`, der ausschließlich folgende Variablen enthält: `year`, `month`, `weekday`, `ressort`, `word_count` sowie `title_length` aus Aufgabe 16. Speichert diesen Datensatz für spätere Analysen.

In welchen Situationen ist `transmute()` praktischer als `mutate()`? Nennt ein konkretes Beispiel.

18. Metrische Variablen lassen sich sinnvoll in inhaltliche Kategorien einteilen. Erstellt mit `mutate()` und `case_when()` eine neue Variable `article_type`:

- `"Kurzmeldung"`, weniger als 200 Wörter
- `"Standardartikel"`, 200 bis unter 700 Wörter
- `"Langer Artikel"`, 700 Wörter und mehr

Wandelt `article_type` anschließend in einen Faktor mit inhaltlich sinnvoller Reihenfolge um. Berechnet, wie viele Artikel auf jede Kategorie entfallen, und ermittelt den Anteil in Prozent.

19. `across()` erlaubt es, dieselbe Transformation auf mehrere Spalten gleichzeitig anzuwenden, ohne den Code zu wiederholen. Es funktioniert sowohl in `mutate()` als auch in `summarize()`.

Löst die folgenden beiden Aufgaben mit `across()`:

- Berechnet in einem einzigen `summarize()`-Aufruf Mittelwert und Median sowohl für `word_count` als auch für `title_length`. Nutzt dafür `across(c(word_count, title_length), list(mean = ~mean(.x, na.rm = TRUE), median = ~median(.x, na.rm = TRUE)))`.
- Wendet in einem einzigen `mutate()`-Aufruf `str_trim()` auf alle Zeichenketten-Spalten in `news_slim` an. Nutzt dafür `across(where(is.character), str_trim)`.

Wie viele Zeilen Code hätte jede Lösung ohne `across()` gebraucht?

Schritt 4: Daten gruppieren und zusammenfassen

`group_by()` und `summarize()` sind das mächtigste Werkzeugpaar in `dplyr`. Mit ihnen berechnet ihr beliebige Kenngrößen für beliebige Gruppen. Wichtig ist dabei, den Unterschied zwischen `summarize()` und `mutate()` nach `group_by()` zu verstehen, sowie die Konsequenzen, die es hat, `ungroup()` zu vergessen.

20. Berechnet die Anzahl der Artikel pro Ressort und sortiert das Ergebnis absteigend nach Häufigkeit. Welches ist das häufigste Ressort? Welche Ressorts enthalten weniger als 100 Artikel? Notiert euch die sechs häufigsten Ressorts als `hauptressorts`-Vektor für spätere Analysen.

21. Berechnet für die sechs Hauptressorts folgende Kenngrößen in einem einzigen `summarize()`-Aufruf:

- Anzahl der Artikel, `n`
- Median der Wortanzahl, `median_word_count`
- Mittlere Wortanzahl, gerundet auf ganze Zahlen, `mean_word_count`
- Anteil der Artikel mit namentlichem Autor in Prozent. Ein Autor gilt als namentlich, wenn `author` weder `NA` noch `"tagesschau.de"` ist.

Sortiert das Ergebnis absteigend nach der mittleren Wortanzahl. Welches Ressort produziert im Schnitt die längsten Artikel?

22. Wie hat sich die Artikelanzahl über die Jahre entwickelt? Berechnet für jedes Jahr die Gesamtanzahl aller Artikel sowie separat die Anzahl für die drei Ressorts `"inland"`, `"ausland"` und `"wirtschaft"`.

Fällt euch etwas an den Jahren 2006 und 2026 auf? Warum könnten diese Jahrgänge unvollständig sein, und wie solltet ihr das bei Interpretationen berücksichtigen?

23. `group_by()` funktioniert nicht nur mit `summarize()`, sondern auch mit `mutate()`. Während `summarize()` den Datensatz auf eine Zeile pro Gruppe verdichtet, behält `mutate()` alle Originalzeilen und fügt nur eine neue Spalte hinzu.

Fügt dem auf die Hauptressorts gefilterten Datensatz mit `group_by()` und `mutate()` zwei neue Variablen hinzu:

- `ressort_median_wc`, der Median der Wortanzahl *des jeweiligen Ressorts*
- `above_median`, ein logischer Wert `TRUE`/`FALSE`, der angibt, ob der Artikel über dem Ressort-Median liegt

Wie viele Artikel im Ressort `"inland"` liegen über dem eigenen Ressort-Median? Ist das überraschend?

24. Nach `group_by() |> mutate()` bleibt der Datensatz in R *weiterhin gruppiert*. Das kann zu unerwarteten Ergebnissen führen, wenn ihr den Datensatz danach ohne explizites `ungroup()` weiterverwendet.

Zeigt diesen Effekt anhand eines konkreten Beispiels: Erstellt einen nach Ressort gruppierten Datensatz mit einer neuen Spalte `n_ressort = n()`. Ruft darauf `slice_max(word_count, n = 1)` auf, *ohne* vorher `ungroup()` zu verwenden. Was erhaltet ihr? Ruft dann `ungroup()` auf und wiederholt `slice_max()`. Was ändert sich? Wann muss man `ungroup()` explizit aufrufen?

Schritt 5: Datensätze verknüpfen mit Joins

In der Praxis liegen Informationen selten in einer einzigen Tabelle vor. *Joins* ermöglichen es, Tabellen über gemeinsame Schlüsselspalten zu verknüpfen. In diesem Teil erstellt ihr eine Metadaten-Tabelle und verknüpft sie auf verschiedene Weisen mit dem Hauptdatensatz.

25. Erstellt in R eine kleine Metadaten-Tabelle `ressort_meta` mit folgenden Spalten:

- `ressort`, der Ressort-Schlüssel, wie er im Datensatz vorkommt, zum Beispiel `"inland"`
- `ressort_label`, ein leserfreundlicher Anzeigename, zum Beispiel `"Inland"`
- `themenbereich`, eine übergeordnete thematische Kategorie, zum Beispiel `"Politik"`, `"Wirtschaft"`, `"Gesellschaft"` oder `"Faktencheck"`

Befüllt die Tabelle mit Einträgen für mindestens die acht häufigsten Ressorts. Lasst dabei bewusst einige seltene Ressorts *weg*, das wird in den nächsten Aufgaben relevant.

26. Verknüpft den auf die Hauptvariablen reduzierten Datensatz mit `ressort_meta` über einen `left_join()`, verbunden über die gemeinsame Spalte `ressort`. Was bedeutet ein `left_join()` in diesem Kontext, welche Zeilen bleiben garantiert erhalten?

Prüft das Ergebnis. Für wie viele Artikel wurde kein passender Eintrag in `ressort_meta` gefunden, erkennbar an `NA` in `ressort_label`? Welche Ressorts sind das?

Erstellt anschließend ein Balkendiagramm, das die Anzahl der Artikel pro `themenbereich` zeigt, sortiert nach Häufigkeit. Behandelt Artikel ohne Match als eigene Kategorie `"Nicht klassifiziert"`, Tipp: `replace_na()`, und hebt diese Kategorie farblich ab.

27. Führt denselben Join als `inner_join()` durch. Was ist der Unterschied zum `left_join()` aus Aufgabe 26? Wie viele Zeilen hat das Ergebnis im Vergleich?

Erstellt mit dem Ergebnis des `inner_join()` einen horizontalen Boxplot der Wortanzahl, gruppiert nach `themenbereich`, gefiltert auf `word_count < 3000`, sortiert nach Median. Welcher Themenbereich produziert die längsten Artikel?

28. Verwendet `anti_join()`, um herauszufinden, welche Artikel im Hauptdatensatz einen Ressort-Wert haben, der *nicht* in `ressort_meta` vorkommt. Gebt die fehlenden Ressorts mit ihrer Häufigkeit aus, sortiert nach Häufigkeit.

Was ist der praktische Nutzen von `anti_join()` beim Arbeiten mit Lookup-Tabellen? Wie hängt der `anti_join()` mit dem `left_join()` zusammen? Was ist der Bezug zwischen den Zeilen, die beim `left_join()` den Wert `NA` erhalten, und dem Ergebnis des

`anti_join()`?

Erstellt ein Balkendiagramm der nicht gematchten Ressorts, nur solche mit mindestens 5 Artikeln, ohne `NA`. Hebt mit einer abweichenden Farbe die Ressorts hervor, die ihr für eine vollständigere Lookup-Tabelle als prioritär erachtet, zum Beispiel weil sie mehr als 100 Artikel enthalten.