Bio & Business Data Science

Nicolas Meseth

Inhalt

Vorwort

Part I

Fall: Klima

Der Datensatz

Der Datensatz stammt von der offiziellen Seite des Deutschen Wetterdienstes, der dort Klimadaten zum direkten Download bereitstellt. Die Daten werden als Textdateien bereitgestellt, die wir im Kapitel "Textdateien einlesen" genauer betrachten.

Daten

1 Textdateien einlesen

1.1 Die `read_csv2` Funktion

```
library(dplyr)
library(readr)
library(janitor)

stations <- read_csv2("./data/stations_list_CLIMAT_data.txt")

# Rows: 4510 Columns: 6
# -- Column specification --------------------------------------------------
# Delimiter: ";"
# chr (6): WMO-Station ID, StationName, Latitude, Longitude, Height, Co...
```

1.1.1 Das `janitor` Paket

Oben haben wir mit `janitor` ein nützliches Paket eingeführt. Wir verwenden daraus die Funktion `clean_names`, die wir unmittelbar an das Laden der Daten mit `read_delim` aufrufen. Was macht diese Funktion?

Um den Effekt der `clean_names` Funktion zu beleuchten, schauen wir uns das Ergebnis einmal mit und ohne diese Funktion an:

```
stations %>%
  colnames()

# [1] "WMO-Station ID" "StationName"    "Latitude"       "Longitude"
# [5] "Height"         "Country"

stations %>%
  clean_names() %>%
  colnames()
```

```
# [1] "wmo_station_id" "station_name"   "latitude"       "longitude"
# [5] "height"         "country"
```

1.2 Spalten konvertieren

Die Datentypen der Spalten wurden offensichtlich nicht richtig erkannt.

2 Das Tidyverse

2.1 Ein Metapaket

2.2 Tibbles

In Kapitel **X** haben wir den Dataframe als Datenstruktur in R kennengelernt. Mit einem Dataframe können wir tabellarische Daten, wie wir sie aus Spreadsheet-Programmen wie Microsoft Excel oder Google Sheets kennen, in R abbilden. Das Tidyverse bringt eine verbesserte Version des klassischen Dataframes mit, das *Tibble*.

Der Name Tibble leitet sich vermutlich aus dem Begriff ***Ti**dy Ta**ble*** ab. Ein Tibble bringt die folgenden Verbesserungen im Vergleich zu einem klassischen Dataframe mit sich:

- Kürzere und prägnantere Ausgabe auf der Konsole.
- Keine Veränderung von Spaltennamen im Vergleich zur Quelle.
- Keine Umwandlung von Zeichenketten in Faktoren als Standard.
- Ein Tibble erzeugt keine Namen für Zeilen (`rownames`)

2.2.1 Ausgabe auf der Konsole

Eine nützliche Verbesserung ist die sinnvollere Ausgabe eines Tibbles auf der Konsole. Das Beispiel unten gibt das Tibble `tweets`, das wir im nächsten Kapitel laden werden, auf der Konsole aus:

```
tweets
```

```
# A tibble: 46,787 x 22
   id        scree~1 text  retwe~2 favor~3 is_qu~4 is_re~5 lang  in_re~6 in_re~7
   <chr>     <chr>   <chr>   <dbl>   <dbl> <lgl>   <lgl>   <chr> <chr>   <chr>
 1 15188914~ MarcoB~ "beg~       3      12 FALSE   FALSE   de    MarcoB~ 151889~
 2 15188914~ MarcoB~ "Die~       3      15 FALSE   FALSE   de    MarcoB~ 151889~
 3 15189075~ starkw~ "@Ko~       1      10 FALSE   FALSE   de    Konsta~ 151888~
 4 15189075~ starkw~ "@Ko~       1      10 FALSE   FALSE   de    Konsta~ 151888~
 5 15189169~ NancyF~ "Uns~      53     490 FALSE   FALSE   de    <NA>    <NA>
```

```
 6 15189169~ NancyF~ "Uns~      53     490 FALSE   FALSE   de    <NA>    <NA>
 7 15189479~ c_lind~ "So ~       6      53 FALSE   FALSE   de    c_lind~ 151894~
 8 15189604~ MarcoB~ "@ar~       1       4 FALSE   FALSE   de    arlete~ 151895~
 9 15189574~ MarcoB~ "Bei~      20      99 FALSE   FALSE   de    <NA>    <NA>
10 15189220~ MarcoB~ "@sc~       1       0 FALSE   FALSE   de    schles~ 151889~
# ... with 46,777 more rows, 12 more variables: hashtags <list>, urls <list>,
#   user_mentions <list>, photos <list>, source <chr>, insert_timestamp <chr>,
#   created_at <dttm>, quote_count <dbl>, reply_count <dbl>,
#   retweeted_status_id <chr>, retweeted_user <chr>, quoted_status_id <chr>,
#   and abbreviated variable names 1: screen_name, 2: retweet_count,
#   3: favorite_count, 4: is_quote_status, 5: is_retweet,
#   6: in_reply_to_screen_name, 7: in_reply_to_status_id
```

Wäre `tweets` ein normaler Dateframe bekämen wir eine sehr lange Ausgabe aller 46787 Zeilen und 22 Spalten. Ein Tibble zeigt uns nur die ersten 10 Zeilen als Beispiele, mit jeweils so vielen Spalten wie sinnvoll in die Ausgabe passen. Die Information, dass es noch weitere Zeilen und Spalten gibt (und wieviele), folgt darunter.

2.2.2 Tibbles sind auch Dataframes

Jedes Tibble ist gleichzeitig ein klassischer Dataframe. Es besitzt aber zusätzlich die Klassen `tbl` und `tbl_df`, die für die zusätzlichen Funktionen und das veränderte Verhalten sorgen.

Zur Illustration erstellen wir manuell einen Dataframe und einen Tibble und lassen uns danach mit `class` die R-Klassen ausgeben, die beide Objekte besitzen:

```
df <- data.frame(id = c(1, 2, 3), name = c("Mark", "John", "Eve"))

class(df)
```

```
[1] "data.frame"
```

Und nun das Tibble:

```
tbl <- tibble(id = c (1, 2, 3), name = c("Mark", "John", "Eve"))

class(tbl)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

Dadurch, dass ein Tibble auch gleichzeitig ein klassischer Dataframe ist, können wir alle Funktionen darauf anwenden, die auch mit Dataframes funktionieren. Schließlich *ist* ein Tibble ja ein Dataframe.

```
colnames(tbl)
```

```
[1] "id"   "name"
```

Wir können jederzeit aus einem Dataframe ein Tibble erzeugen:

```
df <- as_tibble(df)

df
```

```
# A tibble: 3 x 2
     id name
  <dbl> <chr>
1     1 Mark
2     2 John
3     3 Eve
```

Das manuelle Erzeugen eines Tibble geht auf zwei unterschiedliche Wege. Zum einen mit der `tibble` Funktion, die Paare aus Spaltenname und zugehörigen Werten als Vektor erwartet (s. oben):

```
tbl <- tibble(id = c (1, 2, 3), name = c("Mark", "John", "Eve"))
```

Eine andere Möglichkeit ist die zeilenweise Erstellung mit `tribble`:

```
tbl <- tribble(
  ~id, ~name,
    1, "Mark",
    2, "John",
    3, "Eve"
)
```

Mehr lesen

- Link zur offziellen Webseite des Tidyverse
- Link zur Dokumentation des Tibble-Pakets

3 Spalten auswählen

Part II

Fall: Covid

Lernziele

- Wie verdeutlichen wir Trends und Entwicklungen über die Zeit?
- Sind Daten vertrauenswürdig? Was sind Indizien für oder dagegen?

Datensatz

4 Das CSV-Format

Im ersten Schritt jeder Datenanalyse müssen wir unserem Computer den Datensatz zur Verfügung stellen. Wir sprechen dabei auch vom *Laden* des Datesatzes. Dabei sagen wir dem Computer, wo die Daten zu finden sind und dass er sie für den schnelleren Zugriff in seinen Arbeitsspeicher holen soll.

Daten liegen in den meisten Fällen in Form von Dateien vor. In manchen Fällen sind sie auch in einer Datenbank gespeichert. Im Fall einer Datei kann ein Datensatz in unterschiedlichen *Formaten* darin gespeichert werden. Ein gängiges Format ist das CSV-Format.

4.1 Das CSV-Format

4.1.1 Strukturierte Daten

CSV steht für *Comma Separated Values* und beschreibt ein Format, um *strukturierte Daten* in einer Textdatei abzuspeichern. Ihr erkennt eine Textdatei im CSV-Format an der Endung `.csv`.

Das CSV-Format ist das wohl verbreitetste Format für die Speicherung und den Austausch von strukturierten Daten. Fast jede Software, die Daten verwaltet oder analysiert, bietet Schnittstellen für CSV-Dateien an. Dafür gibt es gute Gründe:

- Die Verwendung von einfachen Textdateien erlaubt die Speicherung und Verarbeitung auf unterschiedlichen Umgebungen wie Windows, macOS oder Linux.
- Das Format ist einfach zu verstehen und auch für Menschen lesbar.
- CSV ist ein offenes Format, d. h. es gibt keine Organisation, die daran die Rechte besitzt und es kann daher von jeder Software verwendet werden. Es gab lange nicht einmal eine offizielle Spezifikation des Formats. Mittlerweile gibt es eine Spezifikation als offizieller MIME Type.

4.1.2 Kopfzeile und Trennzeichen

Das CSV-Format speichert strukturierte Daten in einer tabellarischen Form, ähnlich wie in Spreadsheets. Jede Zeile stellt eine Beobachtung (Englisch: *observation* oder *case*) oder Datensatz (Englisch: *record*) dar, und jeder Datensatz hat verschiedene Attribute (oder Spalten),

deren Werte durch ein Komma voneinander getrennt sind. Das Komma als Trennzeichen ist keineswegs verbindlich. Generell kann jedes Symbol verwendet werden. Häufige Alternativen sind das Semikolon, Leerzeichen oder ein Tabstop. Letzteres wird oft mit der eigenen Endung `.tsv` für *Tab Separated Values* gespeichert.

Unten ihr einen Ausschnitt aus dem Covid19-Datensatz von Our World in Data, wie man ihn in einem einfachen Texteditor wie Notepad++ anzeigen lassen kann. Man erkennt schnell, dass sich die erste Zeile von den anderen unterscheidet: Sie beinhaltet die Spaltennamen, die hintereinander durch das Trennzeichen (hier: Komma) getrennt aufgelistet werden. Man nennt die erste Zeile auch *Kopfzeile* (Englisch: *Header*). Die Kopfzeile ist nicht verpflichtend. Es gibt auch CSV-Dateien ohne Kopfzeile. In diesem Fall muss die Benennung der Spalten später manuell erfolgen.

Nach der Kopfzeile stellt jede weitere Zeile die Spaltenwerte eines Datensatzes (oder Beobachtung) dar. Diese werden stets in der gleichen Reihenfolge wie in der Kopfzeile durch das Trennzeichen (hier: Komma) voneinander getrennt aufgelistet. Diese Regel ist wichtig, da sonst die Zuordnung von Werten zu Spaltennamen unmöglich wäre.

Es muss nicht jeder Spaltenwert existieren. Sollte ein Wert für eine Beobachtung nicht vorhanden sein, so wird einfach nach dem Komma nichts eingetragen und es folgen zwei Kommata nacheinander.

```
iso_code,continent,location,date,total_cases,new_cases, ...
DEU,Europe,Germany,2020-01-27,1.0,1.0, ...
DEU,Europe,Germany,2020-01-28,4.0,3.0, ...
DEU,Europe,Germany,2020-01-29,4.0,0.0, ...
DEU,Europe,Germany,2020-01-30,4.0,0.0, ...
...
```

4.2 Die Funktion `read_csv`

Die Funktion `read_csv` aus dem `readr` Paket erlaubt und das effiziente Laden von Daten aus CSV-Dateien. Der Code unten lädt die tagesaktuelle Version des Covid-19 Datensatzes von Our World in Data:

```
covid <- read_csv("https://covid.ourworldindata.org/data/owid-covid-data.csv")
```

```
Rows: 219868 Columns: 67
-- Column specification --------------------------------------------------------
Delimiter: ","
chr   (4): iso_code, continent, location, tests_units
dbl  (62): total_cases, new_cases, new_cases_smoothed, total_deaths, new_dea...
```

```
date  (1): date

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

5 Trends

Kennzahlen einer globalen Pandemie, wie die täglichen Neuinfektionen, eignen sich gut für die Einführung von Visualisierungsformen für Trends und Entwicklungen. Spätestens sein den Jahren der Pandemie, beginnend im Frühjahr 2020, kennt jeder entsprechende Diagramme aus den Nachrichten. In diesem Kapitel lernen wir, diese und andere Diagramme mit `ggplot2` selbst zu erstellen.

> **i** In diesem Kapitel geht es um die Möglichkeiten, Trends und Entwicklungen visuell darzustellen. Die Feinheiten einer Visualisierung, wie die Beschriftung der Achsen, Verwendung von Farben oder Formatierung der Achsenbeschriftungen, werden hier nicht betrachtet und für möglichst einfache Codebeispiele weggelassen.

5.1 Liniendiagramme

5.1.1 Eine Serie

Liniendiagramme verwenden Verbindungen zwischen benachbarten Punkten, um über die Steigung der Linien Entwicklungen sichtbar werden zu lassen. Für die Umsetzung dieser Linien in `ggplot2` verwenden wir die Funktion `geom_line`. Die Punkte selbst werden über die x- und y-Achsen bestimmt. Typischerweise tragen wir auf der x-Achse eine zeitliche Dimension ab, während wir die y-Achse für die betrachtete Größe verwenden.

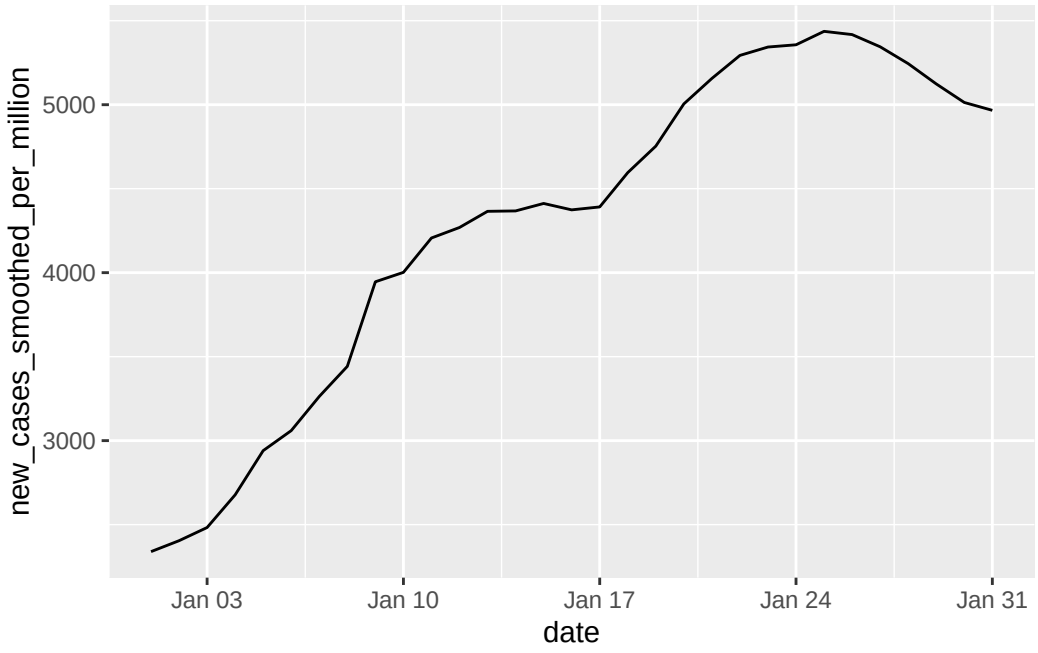
Für die folgenden Visualisierungen filtern wir die Daten auf Frankreich und den Januar 2022:

```
cov_france <-
  covid %>%
  filter(location == "France", date >= "2022-01-01", date <= "2022-01-31") %>%
  select(date, new_cases_smoothed_per_million)
```

Auf der Grundlage erstellen wir ein Liniendiagramm:

```
cov_france %>%
  ggplot() +
  aes(x = date, y = new_cases_smoothed_per_million) +
```

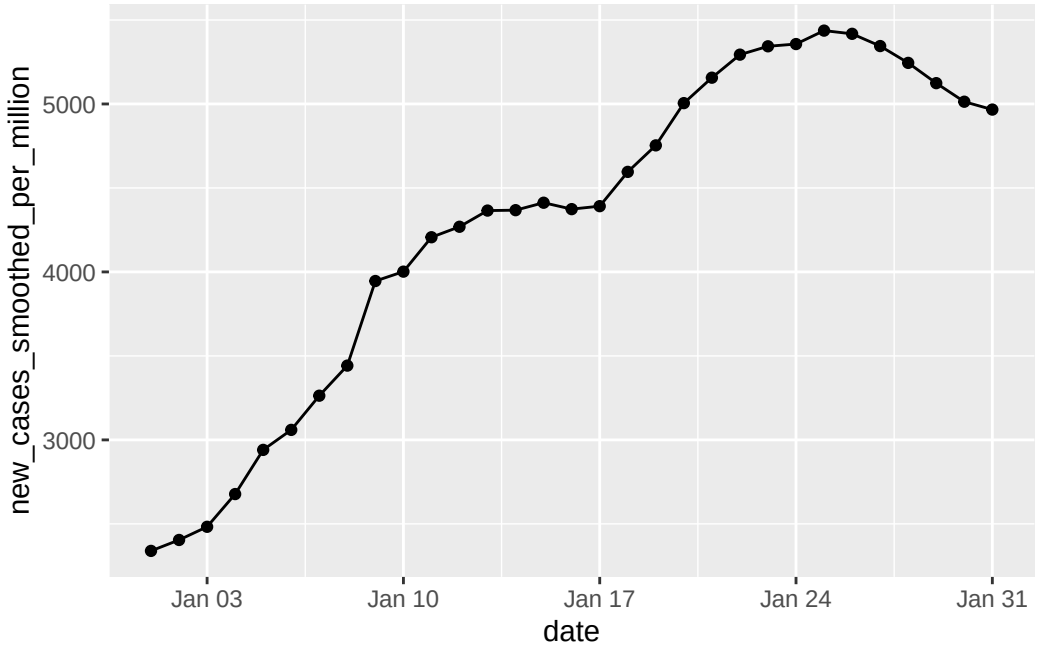
```
  geom_line()
```



5.1.2 Linien und Punkte zusammen

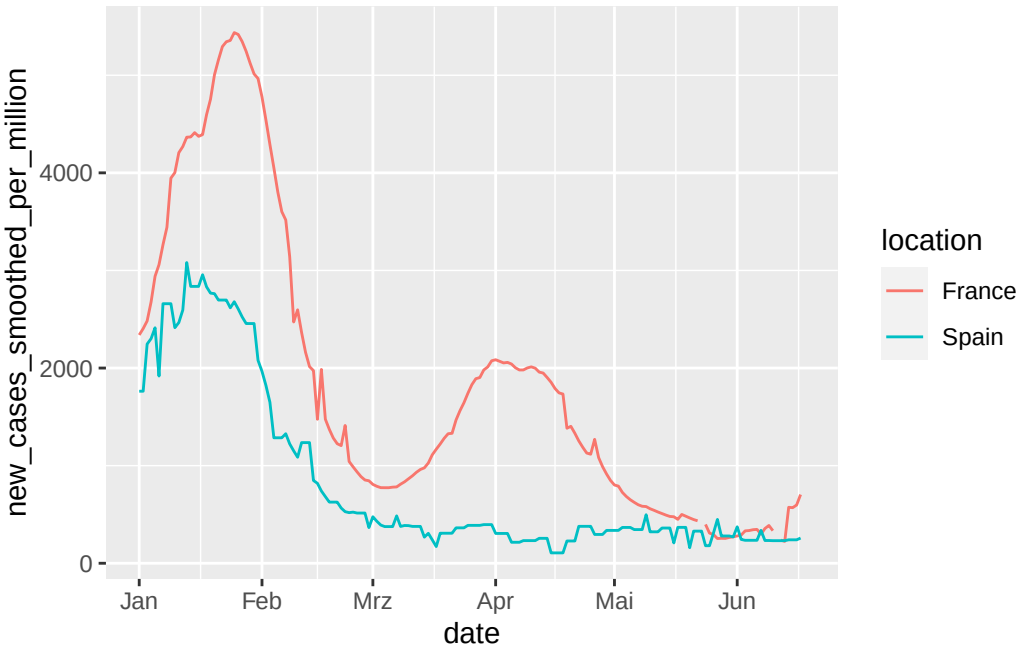
Um die Verbindungen zwischen zwei Punkten deutlicher sichtbar werden zu lassen, können wir zusätzlich zu den Linien auch die Punkte in das Diagramm einzeichnen:

```
cov_france %>%
  ggplot() +
  aes(x = date, y = new_cases_smoothed_per_million) +
  geom_line() +
  geom_point()
```



5.1.3 Mehrere Serien

```
covid %>%
  filter(location %in% c("France", "Spain"), date >= "2022-01-01") %>%
  select(date, new_cases_smoothed_per_million, location) %>%
  ggplot() +
  aes(x = date, y = new_cases_smoothed_per_million, color = location) +
  geom_line()
```


new_cases_smoothed_per_million
4000
2000
0
Jan
Feb
Mrz
Apr
Mai
Jun
date
location
France
Spain

6 Datenherkunft

Part III

Fall: Tweets

Lernziele

- Was ist **SQL** und wofür können wir es verwenden?
- Wie können wir mit SQL Daten abfragen?

Datensatz

7 JSON

7.1 Objekte

7.2 Listen oder *Arrays*

8 Zusammenhänge

Part IV

Fall: REWE

Lernziele

- Wie stellen wir Größenverhältnisse (Proportionen) effektiv dar?

Daten

Datensatz

Der Datensatz besteht aus einem Snapshot aus dem März 2018 aller im REWE Online-Shop angebotenen Produkte.

> **i** Was ist ein Snapshot?
>
> Ein **Snapshot** ist der Stand eines Datensatzes zu einem bestimmten Zeitpunkt in der Vergangenheit. In diesem Fall wurden die Daten an einem besitmmten Tag aus dem REWE Online-Shop abgegriffen und gespeichert. Die Daten auf der Webseite können sich seitdem verändert haben. Der Datensatz reflektiert diese Änderungen nicht, er bleibt auf dem Stand der Erstellung.

9 Proportionen

Part V

Fall: Limonade

10 Lange und breite Daten

Wie bereits in Abschnitt Fall 2: Covid gesehen, können Daten in unterschiedlicher Form vorliegen.

Part VI

Fall: Open Food Facts

11 Ausreisser

12 Texte durchsuchen

13 Stichprobenverzerrung

Quellen