V08 – tidyr

10. Mai 2021

Contents

1 Tidy Data

Wir bezeichnen hier mit dem Begriff **Dataset** eine zusammengehörende Menge von Tabellen.

Bemerkung: Als Kandidat für eine Übersetzung des englischen Begriffs *data set* bietet sich *Datensatz* an. Dieser Begriff wird jedoch in der Informatik in Zusammenhang mit der Theorie von Datenbanken anders verwendet als in der Statistik, siehe https://de.wikipedia.org/wiki/Datensatz#Abweichende_Bedeutung_des_Begriffs.

Ein Dataset ist **tidy** falls folgende Bedingungen erfüllt sind:

- Jede experimentelle Variable / Messgröße bildet eine benannte Spalte.
- Jede Beobachtung / Messung bildet eine (unbenannte) Zeile.
- Jedes Experiment / jede Art Beobachtungstyp bildet eine Tabelle.

(aus Wickham, Hadley (2014). Tidy Data. *Journal of Statistical Software* **59**. https://vita.had.co.nz/papers/tidy-data.pdf)

Diese Beschreibung ist bewusst vage gehalten. Einen formalen Ansatz stellen wir kurz am Ende der Lektion vor.

Um dennoch diese Idee von **tidy data** zu verstehen, sehen wir uns einige Beispiele an.

Folgende Tabellen enthalten jeweils die Anzahl an Tuberkulose-Fällen in Afghanistan, Brasilien und China zwischen 1999 und 2000 (WHO).

```
library(tidyverse)
table1
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
table2
## # A tibble: 12 x 4
```

```
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
table3
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
table4a # cases
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
table4b # population
## # A tibble: 3 x 3
##   country         `1999`     `2000`
## * <chr>            <int>      <int>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
table5
## # A tibble: 6 x 4
##   country     century year  rate
## * <chr>       <chr>   <chr> <chr>
## 1 Afghanistan 19      99    745/19987071
## 2 Afghanistan 20      00    2666/20595360
## 3 Brazil      19      99    37737/172006362
## 4 Brazil      20      00    80488/174504898
## 5 China       19      99    212258/1272915272
## 6 China       20      00    213766/1280428583
```

Am ehesten entspricht `table1` den Voraussetzungen für tidy data.

Ein Beispiel für ein größeres tidy Dataset ist das Paket `nycflights13`.

Es enthält 5 Tibbles, die die Flüge von New York City aus im Jahr 2013 beschreiben:

- `airlines`: Airline names
- `airports`: Airport metadata
- `flights`: Flights data
- `planes`: Plane metadata
- `weather`: Hourly weather data

1.1 Pivoting

Der erste Schritt zum Erstellen eines tidy Datasets ist das Feststellen der Variablen und Beobachtungen.

Im zweiten Schritt wird meist eines der folgenden zwei typischen Problemen gelöst:

- Eine Variable ist über mehrere Spalten verteilt.
- Eine Beobachtung ist über mehrere Zeilen verteilt.

`pivot_longer(tb, cols, names_to="name", values_to="value")` nimmt die in `cols` beschriebenen Spalten des Tibbles `tb` und schreibt deren Werte in eine Spalte namens `"value"` und die zugehörigen Spaltennamen in eine Spalte namens `"name"`.

Mit `pivot_longer()` erhöht sich die Zeilenzahl (oder lässt sie gleich), die Tabelle wird also *longer*.

```
table4a # cases
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
table4b # population
## # A tibble: 3 x 3
##   country         `1999`     `2000`
## * <chr>            <int>      <int>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases") ->
  tidy4a
tidy4a
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Afghanistan 2000    2666
## 3 Brazil      1999   37737
## 4 Brazil      2000   80488
## 5 China       1999  212258
## 6 China       2000  213766
table4b %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "population") ->
  tidy4b
tidy4b
## # A tibble: 6 x 3
##   country     year  population
##   <chr>       <chr>      <int>
## 1 Afghanistan 1999    19987071
## 2 Afghanistan 2000    20595360
```

```
## 3 Brazil      1999   172006362
## 4 Brazil      2000   174504898
## 5 China       1999  1272915272
## 6 China       2000  1280428583
left_join(tidy4a, tidy4b)
## Joining, by = c("country", "year")
## # A tibble: 6 x 4
##   country     year   cases population
##   <chr>       <chr>  <int>      <int>
## 1 Afghanistan 1999     745   19987071
## 2 Afghanistan 2000    2666   20595360
## 3 Brazil      1999   37737  172006362
## 4 Brazil      2000   80488  174504898
## 5 China       1999  212258 1272915272
## 6 China       2000  213766 1280428583
```

Spalten können wie `dplyr::select()` angegeben werden, also auch mit den *Select-Helpers*.

```
table4a %>%
  pivot_longer(matches("[0-9]{4}"), names_to = "year", values_to = "cases")
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Afghanistan 2000    2666
## 3 Brazil      1999   37737
## 4 Brazil      2000   80488
## 5 China       1999  212258
## 6 China       2000  213766
```

Das Gegenstück zu `pivot_longer()` ist `pivot_wider()`.

`pivot_wider(tb, names_from=name, values_from=value)` erzeugt für jeden in der Spalte `name` von `tb` auftretenden Wert eine Spalte mit diesem Wert als Namen und den zugehörigen Werten aus der Spalte `value`.

Mit `pivot_wider()` erhöht sich die Spaltenzahl (oder bleibt gleich), die Tabelle wird also *wider*.

```
table2
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
table2 %>%
    pivot_wider(names_from = type, values_from = count)
## # A tibble: 6 x 4
```

```
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Beide Funktionen haben noch weitere Argumente, die ihr Verhalten in speziellen Situationen beeinflussen. Hierzu sei auf die Dokumentation `?pivot_wider` und `?pivot_longer` verwiesen.

1.2 separate and unite

`separate(tb, col, into, sep=",")` macht aus einer `character`-Spalte `col` mehrere (mit den Namen `into`), indem an bestimmten Symbolen (oder Positionen) getrennt wird.

Das Argument `sep` gibt an, an welchem Zeichen getrennt werden soll. Der Wert von `sep` wird als regulärer Ausdruck interpretiert.

```
table3
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
table3 %>%
  separate(rate, into = c("cases", "population"), sep = "/")
## # A tibble: 6 x 4
##   country      year cases  population
##   <chr>       <int> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
## 4 Brazil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Der Default-Wert von `sep` ist ein regulärer Ausdruck für alle nicht-alphanumerischen Symbole (`"[^[:alnum:]]+"`).

```
table3 %>%
  separate(rate, into = c("cases", "population"))
## # A tibble: 6 x 4
##   country      year cases  population
##   <chr>       <int> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
## 4 Brazil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Die neuen Spalten sind — wie die alte — vom Typ `character`. Mit der Option `convert = TRUE` aktivieren wir automatische Typumwandlung.

```
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Alternativ zu `separate()` kann `extract()` genutzt werden. Anstatt `sep` ist hier das vierte Argument `regex` — ein regulärer Ausdruck mit einer RegEx-Gruppe pro Element von `into`.

```
table3 %>%
  extract(rate, into = c("cases", "population"), regex="([0-9]*)/([0-9]*)", convert = TRUE)
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Das Gegenstück zu `separate()` ist `unite()`.

`unite(tb, "new_col", col_1, ..., col_n, sep=",")` Fügt die Spalten `col_1`, ..., `col_n` zu einer neuen `character`-Spalte `new_col` zusammen. Die Werte der einzelnen Spalten werden dabei durch `sep` getrennt.

```
table5
## # A tibble: 6 x 4
##   country     century year  rate
## * <chr>       <chr>   <chr> <chr>
## 1 Afghanistan 19      99    745/19987071
## 2 Afghanistan 20      00    2666/20595360
## 3 Brazil      19      99    37737/172006362
## 4 Brazil      20      00    80488/174504898
## 5 China       19      99    212258/1272915272
## 6 China       20      00    213766/1280428583
table5 %>%
  unite("year", century, year, sep="")
## # A tibble: 6 x 3
##   country     year  rate
##   <chr>       <chr> <chr>
## 1 Afghanistan 1999  745/19987071
## 2 Afghanistan 2000  2666/20595360
## 3 Brazil      1999  37737/172006362
## 4 Brazil      2000  80488/174504898
## 5 China       1999  212258/1272915272
```

```
## 6 China       2000  213766/1280428583
table5 %>%  # convert year to integer
  unite("year", century, year, sep="") %>%
  mutate_at("year", as.integer)
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

Auch hier verweisen wir wieder auf `?separate`, `?extract`, `?unite` für ausführlichere Informationen.

1.3 Fehlende Werte

```
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66)
)
```

Fehlende Werte können explizit sein (`NA` oder anderer Wert, der Fehlen markiert) oder implizit (taucht nicht in der Tabelle auf).

In `stocks` fehlt Quartal 2015.4 explizit. Quartal 2016.1 fehlt implizit.

```
# manche Operationen machen implizit fehlende Werte expilizt
stocks %>%
  pivot_wider(names_from=year, values_from=return)
## # A tibble: 4 x 3
##     qtr `2015` `2016`
##   <dbl>  <dbl>  <dbl>
## 1     1   1.88  NA
## 2     2   0.59   0.92
## 3     3   0.35   0.17
## 4     4  NA      2.66
```

Die Funktion `complete()` vervollständigt die Beobachtungen, sodass es für alle Kombinationen der Werte der übergebenen Spalten eine Zeile gibt. Implizit fehlende Werte werden dadurch explizit.

```
stocks %>%
  complete(year, qtr) ->
  stocks_cmpl
stocks_cmpl
## # A tibble: 8 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2015     3   0.35
## 4  2015     4  NA
## 5  2016     1  NA
```

```
## 6  2016     2   0.92
## 7  2016     3   0.17
## 8  2016     4   2.66
```

Um `NA`-Werte durch andere Werte zu ersetzen nutze `replace_na()`.

```
stocks_cmpl %>%
  replace_na(list(return = 99)) ->
  stocks_99
stocks_99
## # A tibble: 8 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2015     3   0.35
## 4  2015     4  99
## 5  2016     1  99
## 6  2016     2   0.92
## 7  2016     3   0.17
## 8  2016     4   2.66
```

Um Zeilen mit `NA`-Werten zu entfernen, nutze `drop_na()`.

```
stocks_cmpl %>%
  drop_na()
## # A tibble: 6 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2015     3   0.35
## 4  2016     2   0.92
## 5  2016     3   0.17
## 6  2016     4   2.66
```

Oft werden fehlende Werte im ursprünglichen Dataset nicht mit `NA` sondern einem anderen Wert markiert. `na_if()` ersetzt bestimmte Werte in einem Vektor mit `NA`. Damit wir `na_if()` auf Tabellen anwenden können, nutzen wir `mutate()`.

```
stocks_99 %>%
  mutate(return = na_if(return, 99))
## # A tibble: 8 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2015     3   0.35
## 4  2015     4  NA
## 5  2016     1  NA
## 6  2016     2   0.92
## 7  2016     3   0.17
## 8  2016     4   2.66
```

2 Normalisierung Relationaler Datenbanken

Wir wenden uns wieder kurz der Theorie **Relationalen Datenbanken** zu und deuten auf den Zusammenhang des Begriffes *tidy data* und der Normalformen von Datenbanken hin.

Ziel der Normalisierung eines Datasets ist es Redundanzen zu verringern.

Redundanzen können bei Änderungen von Werten Widersprüche (sogenannte *Anomalien*) erzeugen.

Die verschiedenen **Normalformen** von Datenbanken sind ein formaler Zugang zu einer Idee, die dem Konzept von tidy data ähnelt.

Wir geben hier die Normalformen in leicht vereinfachter Form wider. Zunächst benötigen wir jedoch noch weitere Definitionen.

Ein **Schlüsselkandidat** ist eine minimale Menge an Spalten, die die Zeilen einer Tabelle eindeutig identifiziert.

Eine Spalte $S \in \{1, \ldots, m\}$ ist **funktional abhängig** von einer Menge an Spalten $\beta \subset \{1, \ldots, m\}$, wenn sich (die Werte in) S als Funktion (der Werte) der Spalten β schreiben lässt.

Ist $S \notin \beta$, dann heißt β **Determinante** für S.

Jede Nicht-Schlüssel-Spalte (Spalte die nicht Teil eines Schlüsselkandidaten ist) ist funktional abhängig von einem Schlüsselkandidaten.

```
tb <- tibble(
  Name = c("Alice", "Bob", "Claire", "Dave"),
  Ort = c("Ahausen", "C-Stadt", "C-Stadt", "C-Stadt"),
  PLZ = c(1111 , 2222, 2223, 2223))
# Name ist Schlüsselkandidat
# PLZ ist Determinante, da Ort funktional abhängig ist von PLZ
tb
## # A tibble: 4 x 3
##   Name   Ort       PLZ
##   <chr>  <chr>   <dbl>
## 1 Alice  Ahausen  1111
## 2 Bob    C-Stadt  2222
## 3 Claire C-Stadt  2223
## 4 Dave   C-Stadt  2223


tb <- tibble(
  weekday = c("Friday", "Monday", "Tuesday"),
  day = c(1, 1, 15),
  month = c(5, 6, 6),
  year = c(2020, 2020, 2021))
# Für diese konkrete Tabelle ist weekday Schlüsselkandidat
# Jedoch wäre dies nicht mehr der Fall, wenn noch einige weitere Einträge in der Tablle hinzukommen
# Dann wäre evtl {day, month, year} ein nüztlicherer Schlüsselkandidat
tb
## # A tibble: 3 x 4
##   weekday   day month  year
##   <chr>   <dbl> <dbl> <dbl>
## 1 Friday      1     5  2020
## 2 Monday      1     6  2020
## 3 Tuesday    15     6  2021
```

Die Normalformen werden meist mit 1NF, 2NF, ... für die k-te Normalform bezeichnet oder BCNF für die Boyce-Codd-Normalform.

1NF: Jede Spalte ist *atomisch*.

Ein Spalte ist *atomisch* wenn sie nicht sinnvoll in weitere Spalten aufgeteilt werden kann.

```
lectures <- tibble(
  lecture_id = c(12, 34, 56),
  lecture = c("Analysis (Gauß)", "Lineare Algebra (Euclid)", "Geometrie (Euclid)"),
  year = c(1815, -310, -309))
lectures %>%
  extract(lecture, into = c("lecture", "teacher"), regex="^([^\\(]*) \\(([^\\)]*)\\)")
## # A tibble: 3 x 4
##   lecture_id lecture         teacher  year
##        <dbl> <chr>           <chr>   <dbl>
## 1         12 Analysis        Gauß     1815
## 2         34 Lineare Algebra Euclid   -310
## 3         56 Geometrie       Euclid   -309
```

2NF: 1NF und keine Nicht-Schlüssel-Spalte ist funktional abhängig von einer echten Teilmenge eines Schlüsselkandidaten.

```
tb <- tibble(
  weekend = c(F, F, F, T, F),
  weekday = c("Friday", "Monday", "Monday", "Saturday", "Friday"),
  first_monday = c(F, T, F, F, F),
  month_name = month.name[c(5,6,6,5,5)],
  day = c(1,1,14,15,15),
  month = c(5,6,6,5,5),
  year = c(2020, 2020, 2021,2021,2020))
tb
## # A tibble: 5 x 7
##   weekend weekday  first_monday month_name   day month  year
##   <lgl>   <chr>    <lgl>        <chr>      <dbl> <dbl> <dbl>
## 1 FALSE   Friday   FALSE        May            1     5  2020
## 2 FALSE   Monday   TRUE         June           1     6  2020
## 3 FALSE   Monday   FALSE        June          14     6  2021
## 4 TRUE    Saturday FALSE        May           15     5  2021
## 5 FALSE   Friday   FALSE        May           15     5  2020
# Schlüsselkandidat: (day, monyh, year)
# month_name fuktional abhängig von month
tb %>% select(-month_name) -> tb2
tb %>% select(month_name, month) %>% distinct() -> months
tb2
## # A tibble: 5 x 6
##   weekend weekday  first_monday   day month  year
##   <lgl>   <chr>    <lgl>        <dbl> <dbl> <dbl>
## 1 FALSE   Friday   FALSE            1     5  2020
## 2 FALSE   Monday   TRUE             1     6  2020
## 3 FALSE   Monday   FALSE           14     6  2021
## 4 TRUE    Saturday FALSE           15     5  2021
## 5 FALSE   Friday   FALSE           15     5  2020
months
## # A tibble: 2 x 2
##   month_name month
##   <chr>      <dbl>
## 1 May            5
## 2 June           6
```

3NF: 2NF und keine Nicht-Schlüssel-Spalte ist funktional abhängig von andereren Nicht-Schlüssel-Spalten. Äquivalent: 2NF und keine Menge an Nicht-Schlüssel-Spalten bilden eine Determinante.

```
tb2
## # A tibble: 5 x 6
##   weekend weekday  first_monday   day month  year
##   <lgl>   <chr>    <lgl>        <dbl> <dbl> <dbl>
## 1 FALSE   Friday   FALSE            1     5  2020
## 2 FALSE   Monday   TRUE             1     6  2020
## 3 FALSE   Monday   FALSE           14     6  2021
## 4 TRUE    Saturday FALSE           15     5  2021
## 5 FALSE   Friday   FALSE           15     5  2020
# weekday ist Determinante für weekend
tb2 %>% select(-weekend) -> tb3
tb2 %>% select(weekend, weekday) %>% distinct() -> weekend
tb3
## # A tibble: 5 x 5
##   weekday  first_monday   day month  year
##   <chr>    <lgl>        <dbl> <dbl> <dbl>
## 1 Friday   FALSE            1     5  2020
## 2 Monday   TRUE             1     6  2020
## 3 Monday   FALSE           14     6  2021
## 4 Saturday FALSE           15     5  2021
## 5 Friday   FALSE           15     5  2020
weekend
## # A tibble: 3 x 2
##   weekend weekday
##   <lgl>   <chr>
## 1 FALSE   Friday
## 2 FALSE   Monday
## 3 TRUE    Saturday
```

BCNF: 3NF und jede Determinante ist ein Schlüsselkandidat.

```
tb3 %>% select(day, weekday, first_monday) %>% distinct() -> first_monday
tb3 %>% select(-first_monday) -> tb4
tb4
## # A tibble: 5 x 4
##   weekday    day month  year
##   <chr>    <dbl> <dbl> <dbl>
## 1 Friday       1     5  2020
## 2 Monday       1     6  2020
## 3 Monday      14     6  2021
## 4 Saturday    15     5  2021
## 5 Friday      15     5  2020
first_monday
## # A tibble: 5 x 3
##     day weekday  first_monday
##   <dbl> <chr>    <lgl>
## 1     1 Friday   FALSE
## 2     1 Monday   TRUE
## 3    14 Monday   FALSE
## 4    15 Saturday FALSE
## 5    15 Friday   FALSE
```

Für eine ausführlichere Beschreibung dieser und weiterer Normalformen, siehe https://en.wikipedia.org/wiki/Database_normalization

Das moderne und schwammige, aber in der Anwendung nützliche Konzept von *tidy data* ist maßgeblich von den älteren (1970er) und formalen Definition der Normalformen beeinflusst.

Oft ist es sinnvoll, eine neues Dataset zunächst zu normalisieren und in einer normalisierten Form abzuspeichern. Während der Analyse wird das Dataset jedoch oft aus praktischen Gründen nicht in einer Normalform sein.