V06 – tibble

3. Mai 2021

Contents

1 Tabellen

Ein **Data-Frame** ist eine tabellenförmige Datenstruktur.

In Base-R wird sie in S3-Objekten der Klasse `data.frame` gespeichert.

```
df <- data.frame(x=1:3, y=letters[1:3])
df
##   x y
## 1 1 a
## 2 2 b
## 3 3 c
class(df)
## [1] "data.frame"
```

Zeilen nennen wir auch **Beobachtungen**, Spalten nennen wir auch **Variablen**.

Einträge in verschiedenen Spalten können unterschiedlichen Typ haben.

Meist ist eine Spalte ein atomarer Vektor. Listen-, Matrix- und Data-Frame-Spalten sind prinzipiell möglich.

Im Zusammenhang mit Funktionen der Paketsammlung *Tidyverse* ist es üblich Data-Frames in Form von **Tibbles** zu nutzen.

Ein Tibble ist ein R-Objekt der Klasse `c("tbl_df", "tbl", "data.frame")`. As baut also auf einem `data.frame`-Objekt auf.

Bemerkung:

- Tabelle ist ein konzeptioneller Begriff, keine spezielle Datenstruktur in R.
- Jedes Data-Frame (und damit auch jedes Tibble) repräsentiert eine Tabelle.
- Es gibt weitere Datenstrukturen in R, die Tabellen repräsentieren, zB `data.table` (was nicht unbedingt zur besseren Unterscheidung der Begriffe beiträgt).

Um Tibbles zu nutzen, muss das Paket `tibble` geladen werden. Es gehört zum Tidyverse.

```
library(tibble) # oder library(tidyverse)
tb <- tibble(x=1:3, y=letters[1:3])
tb
## # A tibble: 3 x 2
```

```
##       x y
##   <int> <chr>
## 1     1 a
## 2     2 b
## 3     3 c
class(tb)
## [1] "tbl_df"     "tbl"        "data.frame"
```

Bei der Ausgabe eines Tibbles auf der Konsole wird unter dem Spaltennamen der Typ der Spalte in Kurzform angezeigt.

```
tibble(logical=F, integer=0L, double=0, character="", list=list(1))
## # A tibble: 1 x 5
##   logical integer double character list
##   <lgl>     <int>  <dbl> <chr>     <list>
## 1 FALSE         0      0 ""        <dbl [1]>
```

Mit `as_tibble()` werden passende Listen, Matrizen und Data-Frames in Tibbles konvertiert.

```
lst <- list(x=1:2, y=letters[1:2])
str(lst)
## List of 2
##  $ x: int [1:2] 1 2
##  $ y: chr [1:2] "a" "b"
as_tibble(lst)
## # A tibble: 2 x 2
##       x y
##   <int> <chr>
## 1     1 a
## 2     2 b

mat <- matrix(1:4, ncol=2)
colnames(mat) <- c("A", "B")
mat
##      A B
## [1,] 1 3
## [2,] 2 4
as_tibble(mat)
## # A tibble: 2 x 2
##       A     B
##   <int> <int>
## 1     1     3
## 2     2     4

df
##   x y
## 1 1 a
## 2 2 b
## 3 3 c
as_tibble(df)
## # A tibble: 3 x 2
##       x y
##   <int> <chr>
## 1     1 a
## 2     2 b
```

```
## 3     3 c
```

Mit `add_column()` und `add_row()` werden Spalten bzw Zeilen zu einem Tibble hinzugefügt.

```
tb <- tibble(x = 1:3, y = 3:1)

add_column(tb, z = -1:1, w = 0)
## # A tibble: 3 x 4
##       x     y     z     w
##   <int> <int> <int> <dbl>
## 1     1     3    -1     0
## 2     2     2     0     0
## 3     3     1     1     0
add_column(tb, z = -1:1, .before = "y")
## # A tibble: 3 x 3
##       x     z     y
##   <int> <int> <int>
## 1     1    -1     3
## 2     2     0     2
## 3     3     1     1

add_row(tb, x = 4:5, y = 0:-1)
## # A tibble: 5 x 2
##       x     y
##   <int> <int>
## 1     1     3
## 2     2     2
## 3     3     1
## 4     4     0
## 5     5    -1
add_row(tb, x = 4, y = 0, .before = 2)
## # A tibble: 4 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     3
## 2     4     0
## 3     2     2
## 4     3     1
add_row(tb, x = 4) # set NA if needed
## # A tibble: 4 x 2
##       x     y
##   <dbl> <int>
## 1     1     3
## 2     2     2
## 3     3     1
## 4     4    NA
```

Die Subsetting-Operatoren (ggf mit Zuweisung) `[`, `[[`, `$`, `[<-`, `[[<-`, `$<-` können ebenso benutzt werden.

```
tb <- tibble(x = 1:3)
tb$y <- 3:1
tb[3, 2] <- 12
tb[c(1,3), ]
## # A tibble: 2 x 2
##       x     y
```

```
##   <int> <int>
## 1     1     3
## 2     3    12
tb[tb$y > 2, ]
## # A tibble: 2 x 2
##       x     y
##   <int> <int>
## 1     1     3
## 2     3    12
tb[[1]] <- letters[1:3]
tb[["x"]]
## [1] "a" "b" "c"
```

Spaltennamen sollten den Regeln für Variablennamen folgen, damit ein einfacher Zugriff möglich ist.

```
# Negativ-Beispiel: Spaltenname "0"
tb["0"] <- 1:3
# tb$0 # ERROR
tb$"0"
## [1] 1 2 3
tb$`0` # alternativ
## [1] 1 2 3
```

Die Funktionen `head()` und `tail()` sind alternative Notationen für Subsetting der ersten oder letzten Zeilen.

```
tb <- tibble(pos = 1:26, lower = letters, upper = LETTERS)
head(tb, 4)
## # A tibble: 4 x 3
##     pos lower upper
##   <int> <chr> <chr>
## 1     1 a     A
## 2     2 b     B
## 3     3 c     C
## 4     4 d     D
tail(tb, 4)
## # A tibble: 4 x 3
##     pos lower upper
##   <int> <chr> <chr>
## 1    23 w     W
## 2    24 x     X
## 3    25 y     Y
## 4    26 z     Z
```

Mit `bind_rows()`, `bind_cols()` fügen wir mehrere Tibbles gleicher Struktur zusammen. Im Gegensatz zu den vorigen sind diese Funktionen im Paket `dplyr` (auch Tidyverse).

```
t1 <- tibble(x=1:2, y=letters[1:2])
t2 <- tibble(x=11:12, y=letters[11:12])

dplyr::bind_rows(t1, t2)
## # A tibble: 4 x 2
##       x y
##   <int> <chr>
## 1     1 a
## 2     2 b
## 3    11 k
```

```
## 4    12 l

t3 <- tibble(x=11:12, z=0:-1)
dplyr::bind_rows(t1, t3) # set NA if needed
## # A tibble: 4 x 3
##       x y         z
##   <int> <chr> <int>
## 1     1 a        NA
## 2     2 b        NA
## 3    11 <NA>      0
## 4    12 <NA>     -1

t4 <- tibble(u = c(T,F), v = c(pi, exp(1)))
dplyr::bind_cols(t1, t4)
## # A tibble: 2 x 4
##       x y     u         v
##   <int> <chr> <lgl> <dbl>
## 1     1 a     TRUE   3.14
## 2     2 b     FALSE  2.72
```

1.1 nycflights13

Neben Funktionen stellen manche Pakete auch Datasets bereit.

Das Paket `nycflights13` enthält mehrere Tibbles. Das Tibble `flights` enthält Informationen zu allen Flügen von New York City im Jahr 2013.

```
#install.packges("nycflights13")
library(nycflights13)
flights
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Für nähere Informationen zu den Spalten (zB Einheiten) siehe `?flights`.

Bei großen Tibbles werden nur die ersten Zeilen und Spalten ausgegeben.

Wir setzen ein paar Optionen, um die Ausgabe hier kompakter zu gestalten.

```
options(
  tibble.print_min=6,
  tibble.print_max=6,
```

```
  tibble.max_extra_cols=0)
flights
## # A tibble: 336,776 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 336,770 more rows
```

Oft sind Daten eines Datasets über mehrere Tabellen (Tibbles) unterschiedlicher Struktur verteilt.

Das Paket `nycflights13` enthält neben `flights` noch die Tabellen `airlines`, `airports`, `planes`, `weather`.

Die Tabelle `flights` enthält eine Variable `carrier` – die 2-Buchstaben-Abkürzung der Airline. In der Tabelle `airlines` ist der Langname der Airline angegeben.

```
flights$carrier[1:10]
##  [1] "UA" "UA" "AA" "B6" "DL" "UA" "B6" "EV" "B6" "AA"
airlines
## # A tibble: 16 x 2
##   carrier name
##   <chr>   <chr>
## 1 9E      Endeavor Air Inc.
## 2 AA      American Airlines Inc.
## 3 AS      Alaska Airlines Inc.
## 4 B6      JetBlue Airways
## 5 DL      Delta Air Lines Inc.
## 6 EV      ExpressJet Airlines Inc.
## # ... with 10 more rows
```

Dieses Dataset wird uns in den folgenden Kapiteln als Beispiel dienen.

2 Relationale Datenbanken

Die Theorie der **Relationalen Datenbanken** mit der **relationalen Algebra**, stellt Beziehungen von Tabellen in Datasets und Operationen auf Datasets auf formale Beine.

In diesem Abschnitt schneiden wir nur sehr knapp diese Themen an und vereinfachen an vielen Stellen. Mit dem vollen Umfang des Themenfelds können mehrere eigene Kurse gefüllt werden.

Die Nomenklatur in der Datenbank-Theorie unterscheidet sich etwas von der traditionell im Zusammenhang mit R verwendeten.

R / allgemein	Dataset	Tabelle	Spalte	Zeile
R (alternativ)		Tibble, Data-Frame	Variable	Beobachtung
DB-Theorie	Datenbank	Relation	Attribut	Tupel

2.1 Definitionen der Grundbegriffe

Eine endliche Folge von Mengen $(W_1, \ldots, W_m)$ heißt **Relationenschema**.

Eine endliche Teilmenge $R \subseteq W_1 \times \cdots \times W_m$ heißt **Relation** des Relationenschemas $(W_1, \ldots, W_m)$. In diesem

Zusammenhang heißen die W_j auch **Wertebereiche**.

Ein Element $t \in R$ einer Relation R heißt **Tupel**.

Bemerkungen:

1. Relationen entsprechen in etwa Tabellen und Tupel den Zeilen. Allerdings gibt es Unterschiede:
   - Tabellen haben eine Ordnung ihrer Zeilen, Relationen sind ungeordnete Mengen.
   - In einer Tabelle kann eine Zeile mehrere Male vorkommen, in einer Relation nicht (Menge).
2. In der Datenbank-Theorie gehört in ein Relationenschema noch jeweils ein Name (Spaltennamen) zu jedem Wertebereich D_j. Zur Vereinfachung lassen wir jedoch hier Namen weg.

Wir identifizieren **Spalten** (genannt Attribut, hat nichts mit Attributen von R-Objekten zu tun) durch ihren Spaltenindex $j \in \{1, \ldots, m\}$.

Die "Zeilenzahl" n einer Relation R ist also die Anzahl der ihrer Elemente $n = |R|$.

Die "Spaltenzahl" m einer Relation R ist die Anzahl der Wertebereiche im zugehörigen Relationenschema.

Beachte: Diese formale Definition entspricht nicht der Implementation eines Tibbles in R. Dort ist `length(tb)` die Spaltenzahl.

2.2 Operationen

2.2.1 Mengenoperationen

Auch wenn Tibbles nicht Relationen direkt repräsentieren (siehe Bemerkung oben), können wir Operationen auf Relationen damit nachvollziehen.

Wir können Tibbles Relationen ähnlich machen, indem wir mehrfach auftretende Zeilen entfernen.

```
tb <- tibble(
  x = c(1, 1, 2, 2, 1),
  y = c("a", "b", "b", "b", "a"))
dplyr::distinct(tb) # dplyr ist auch Teil des Tidyverse
## # A tibble: 3 x 2
##       x y
##   <dbl> <chr>
## 1     1 a
## 2     1 b
## 3     2 b
```

Für zwei Relationen des gleichen Relationenschemas $R, S \subseteq W_1 \times \cdots \times W_m$ können Standard-Mengenoperationen durchgeführt werden.

```
# Relationenschema (double, character)
# wobei double für die Menge der möglichen double-Werte in R steht
R <- tibble(
  x = c(1, 1),
  y = c("a", "b"))
S <- tibble(
  x = c(1, 2),
  y = c("a", "a"))
R
## # A tibble: 2 x 2
##       x y
##   <dbl> <chr>
## 1     1 a
## 2     1 b
S
```

```
## # A tibble: 2 x 2
##       x y
##   <dbl> <chr>
## 1     1 a
## 2     2 a
```

Im Folgenden steht $\wedge$ für "und" und $\vee$ für "oder".

- Vereinigung $R \cup S := \{t | t \in R \vee t \in S\}$

```
library(dplyr)
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
union(R, S)
## # A tibble: 3 x 2
##       x y
##   <dbl> <chr>
## 1     1 a
## 2     1 b
## 3     2 a
```

- Schnitt $R \cap S := \{t | t \in R \wedge t \in S\}$

```
intersect(R, S)
## # A tibble: 1 x 2
##       x y
##   <dbl> <chr>
## 1     1 a
```

- Differenz $R \setminus S := \{t | t \in R \wedge t \notin S\}$

```
setdiff(R, S)
## # A tibble: 1 x 2
##       x y
##   <dbl> <chr>
## 1     1 b
```

- Symmetrische Differenz $R \triangle S := (R \cup S) \setminus (R \cap S)$

```
# Keine eigene Funktion vorhanden
setdiff(union(R, S), intersect(R, S))
## # A tibble: 2 x 2
##       x y
##   <dbl> <chr>
## 1     1 b
## 2     2 a
```

2.2.2 Kartesisches Produkt

Seien $R \subseteq W_1 \times \cdots \times W_\ell$ und $S \subseteq W_{\ell+1} \times \cdots \times W_m$ Relationen.

- Kartesisches Produkt $R \times S := \{(x_1, \ldots, x_m) | (x_1, \ldots, x_\ell) \in R \wedge (x_{\ell+1}, \ldots, x_m) \in S\}$.

```r
# Schema (integer, character)
R <- tibble(
  x = 1:3,
  y = letters[1:3])
# Schema (character, logical)
S <- tibble(
  u = LETTERS[1:2],
  v = c(T, F))
R
## # A tibble: 3 x 2
##       x y
##   <int> <chr>
## 1     1 a
## 2     2 b
## 3     3 c
S
## # A tibble: 2 x 2
##   u     v
##   <chr> <lgl>
## 1 A     TRUE
## 2 B     FALSE
tidyr::crossing(R, S) # tidyr ist auch Teil des Tidyverse
## # A tibble: 6 x 4
##       x y     u     v
##   <int> <chr> <chr> <lgl>
## 1     1 a     A     TRUE
## 2     1 a     B     FALSE
## 3     2 b     A     TRUE
## 4     2 b     B     FALSE
## 5     3 c     A     TRUE
## 6     3 c     B     FALSE
```