V07 – dplyr

3. Mai 2021

Contents

1 Verben für eine Tabelle

Das Paket `dplyr` gehört zum Tidyverse.

`dplyr` stellt Funktionen (genannt *Verben*) für die Manipulation von Daten in Tibbles bereit.

Zur Manipulation eines einzelnen Tibbles enthält `dplyr` folgende Verben:

- `filter()`: wähle bestimmte Zeilen aus
- `arrange()`: ordne Zeilen neu an
- `select()`: wähle bestimmte Spalten aus
- `mutate()`: erstelle neue Spalten
- `summarize()`: fasse Spalten zusammen, zB Mittelwert

Wir verwenden das Tibble `nycflights13::flights` in unseren Beispielen und setzen ein paar Optionen, um die Ausgabe kompakter zu gestalten.

```
library(tidyverse) # lädt auch dplyr
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
```

```
## x dplyr::lag()    masks stats::lag()
library(nycflights13)
options(
  tibble.print_min=4,
  tibble.print_max=4,
  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
## # ... with 336,772 more rows
```

1.1 filter()

`filter(tb, fltr_1, ..., fltr_n)` gibt ein Tibble all derjenigen Zeilen von `tb` zurück, die **alle** Bedingungen (`fltr_1`, ..., `fltr_n`) erfüllen.

`fltr_1`, ..., `fltr_n` müssen zu einem `logical`-Vektor der Länge `nrow(tb)` (oder 1) evaluieren.

`NA` wirkt wie `FALSE`.

Spaltennamen von `tb` werden wie Variablen behandelt (Zugriff ohne `tb$`).

```
sel <- flights$dep_delay == 0
nrow(flights) == length(sel)
## [1] TRUE
typeof(sel)
## [1] "logical"
filter(flights, sel)
## # A tibble: 16,514 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      559            559         0      702            706
## 2  2013     1     1      600            600         0      851            858
## 3  2013     1     1      600            600         0      837            825
## 4  2013     1     1      607            607         0      858            915
## # ... with 16,510 more rows

dep_delay # kein belegter Variablenname
## Error in eval(expr, envir, enclos): object 'dep_delay' not found
filter(flights, dep_delay < -30) # Spaltennamen fungieren als Variablen
## # A tibble: 3 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    11    10     1408           1440       -32     1549           1559
## 2  2013    12     7     2040           2123       -43       40           2352
## 3  2013     2     3     2022           2055       -33     2240           2338

filter(flights, dep_time < 5, day == 1) # fltr1, fltr2 wie fltr1 & fltr2
## # A tibble: 4 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     3     1        4           2159       125      318             56
## 2  2013     6     1        2           2359         3      341            350
## 3  2013     7     1        1           2029       212      236           2359
## 4  2013     7     1        2           2359         3      344            344
```

Weitere Beispiel:

```
filter(flights, month %in% c(11, 12), origin == "JFK" | day == 27)
## # A tibble: 19,199 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    11     1        5           2359         6      352            345
## 2  2013    11     1       35           2250       105      123           2356
## 3  2013    11     1      542            545        -3      831            855
## 4  2013    11     1      549            600       -11      912            923
## # ... with 19,195 more rows
filter(flights, air_time / distance > 0.5)
## # A tibble: 45 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    28     1332           1300        32     1551           1406
## 2  2013     1    28     1917           1825        52     2118           1935
## 3  2013     1    30     1037            955        42     1221           1100
## 4  2013    10    17     1535           1540        -5     1724           1651
## # ... with 41 more rows
filter(flights, is.na(dep_time))
## # A tibble: 8,255 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       NA           1630        NA       NA           1815
## 2  2013     1     1       NA           1935        NA       NA           2240
## 3  2013     1     1       NA           1500        NA       NA           1825
## 4  2013     1     1       NA            600        NA       NA            901
## # ... with 8,251 more rows
```

Bemerkung: Siehe `?slice`, um Zeilen nach Zeilennummer auszuwählen.

1.2 arrange()

`arrange(tb, val_1, ..., val_n)` gibt das Tibble `tb` nach `val_1` aufsteigend sortiert aus.

Bei gleichen Werten in `val_i` wird durch `val_{i+1}` entschieden.

`val_1`, ..., `val_n` müssen zu atomaren Vektoren der Länge `nrow(tb)` evaluieren. Spaltennamen werden wie Variablen behandelt.

```
sort_by <- nrow(flights):1
arrange(flights, sort_by)
## # 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     9    30       NA            840        NA       NA           1020
## 2  2013     9    30       NA           1159        NA       NA           1344
## 3  2013     9    30       NA           1210        NA       NA           1330
## 4  2013     9    30       NA           2200        NA       NA           2312
```

```
## # ... with 336,772 more rows
arrange(flights, dep_delay)
## # 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    12     7     2040           2123       -43       40           2352
## 2  2013     2     3     2022           2055       -33     2240           2338
## 3  2013    11    10     1408           1440       -32     1549           1559
## 4  2013     1    11     1900           1930       -30     2233           2243
## # ... with 336,772 more rows
arrange(flights, abs(dep_delay))
## # 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      559            559         0      702            706
## 2  2013     1     1      600            600         0      851            858
## 3  2013     1     1      600            600         0      837            825
## 4  2013     1     1      607            607         0      858            915
## # ... with 336,772 more rows
arrange(flights, year, month, day)
## # 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
## # ... with 336,772 more rows
```

`desc()` wandelt einen Vektor so um, dass die Sortierung (scheinbar) absteigend ist.

```
desc(1:10)
##  [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
desc(letters)
##  [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19
## [20] -20 -21 -22 -23 -24 -25 -26
arrange(flights, desc(dep_delay))
## # 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     9      641            900      1301     1242           1530
## 2  2013     6    15     1432           1935      1137     1607           2120
## 3  2013     1    10     1121           1635      1126     1239           1810
## 4  2013     9    20     1139           1845      1014     1457           2210
## # ... with 336,772 more rows
arrange(flights, year, desc(month), day)
## # 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    12     1       13           2359        14      446            445
## 2  2013    12     1       17           2359        18      443            437
## 3  2013    12     1      453            500        -7      636            651
## 4  2013    12     1      520            515         5      749            808
## # ... with 336,772 more rows
```

Bemerkung: Umsetzung in Base-R

```
arrange(flights, dep_delay) # in dplyr
flights[order(flights$dep_delay), ] # in Base-R
```

1.3 select()

`select(tb, vars_1, ..., vars_n)` gibt die in `vars_1`, ..., `vars_n` beschriebenen Spalten des Tibbles `tb` aus.

Zur Beschreibung der zu wählenden Spalten wird die **Tidy-Select**-Syntax genutzt, die in ihrer Auswertung nicht den Standard-Regeln von R genügt (**Non-Standard Evaluation**).

Die verschiedenen Möglichkeiten zur Spaltenwahl lassen sich in 2 Kategorien einteilen: *Position-Select* und *Condition-Select*.

1.3.1 Position-Select

Bei *Position-Select* können Spalten durch ihren Spaltenindex ausgewählt werden. Alternativ können die Spaltennamen angegeben werden (auch ohne Anführungszeichen, falls Spaltenname gültiger Variablenname ist).

```
select(flights, 3, 5) # Spaltenindex
## # A tibble: 336,776 x 2
##     day sched_dep_time
##   <int>          <int>
## 1     1            515
## 2     1            529
## 3     1            540
## 4     1            545
## # ... with 336,772 more rows
select(flights, flight, "origin", dest) # Spaltenname
## # A tibble: 336,776 x 3
##   flight origin dest
##    <int> <chr>  <chr>
## 1   1545 EWR    IAH
## 2   1714 LGA    IAH
## 3   1141 JFK    MIA
## 4    725 JFK    BQN
## # ... with 336,772 more rows
select(flights, flight, 9) # gemischt
## # A tibble: 336,776 x 2
##   flight arr_delay
##    <int>     <dbl>
## 1   1545        11
## 2   1714        20
## 3   1141        33
## 4    725       -18
## # ... with 336,772 more rows
```

Spaltennamen werden in Verbindung mit `:`, unäres `-`, und `c()` wie ihre Index-Nummern behandelt.

```
select(flights, year:day, arr_delay)
## # A tibble: 336,776 x 4
##    year month   day arr_delay
##   <int> <int> <int>     <dbl>
## 1  2013     1     1        11
```

```
## 2  2013     1     1        20
## 3  2013     1     1        33
## 4  2013     1     1       -18
## # ... with 336,772 more rows
select(flights, -(year:day))
## # A tibble: 336,776 x 16
##   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
##      <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
## 1      517            515         2      830            819        11 UA
## 2      533            529         4      850            830        20 UA
## 3      542            540         2      923            850        33 AA
## 4      544            545        -1     1004           1022       -18 B6
## # ... with 336,772 more rows
select(flights, -c(2:dep_delay, sched_arr_time))
## # A tibble: 336,776 x 13
##    year arr_time arr_delay carrier flight tailnum origin dest  air_time distance
##   <int>    <int>     <dbl> <chr>    <int> <chr>   <chr>  <chr>    <dbl>    <dbl>
## 1  2013      830        11 UA        1545 N14228  EWR    IAH        227     1400
## 2  2013      850        20 UA        1714 N24211  LGA    IAH        227     1416
## 3  2013      923        33 AA        1141 N619AA  JFK    MIA        160     1089
## 4  2013     1004       -18 B6         725 N804JB  JFK    BQN        183     1576
## # ... with 336,772 more rows
```

Stehen Spaltennamen oder -nummern in Variablen, sollten sie mit `all_of()` übergeben werden, um `-` nutzen zu können und Zweideutigkeiten zu vermeiden.

```
vars <- c("day", "dep_delay", "sched_arr_time")
select(flights, all_of(vars))
## # A tibble: 336,776 x 3
##     day dep_delay sched_arr_time
##   <int>     <dbl>          <int>
## 1     1         2            819
## 2     1         4            830
## 3     1         2            850
## 4     1        -1           1022
## # ... with 336,772 more rows
select(flights, -all_of(vars))
## # A tibble: 336,776 x 16
##    year month dep_time sched_dep_time arr_time arr_delay carrier flight tailnum
##   <int> <int>    <int>          <int>    <int>     <dbl> <chr>    <int> <chr>
## 1  2013     1      517            515      830        11 UA        1545 N14228
## 2  2013     1      533            529      850        20 UA        1714 N24211
## 3  2013     1      542            540      923        33 AA        1141 N619AA
## 4  2013     1      544            545     1004       -18 B6         725 N804JB
## # ... with 336,772 more rows
vars <- 1:3
select(flights, all_of(vars), dep_delay)
## # A tibble: 336,776 x 4
##    year month   day dep_delay
##   <int> <int> <int>     <dbl>
## 1  2013     1     1         2
## 2  2013     1     1         4
## 3  2013     1     1         2
## 4  2013     1     1        -1
```

```
## # ... with 336,772 more rows
day <- 2
select(flights, day)
## # A tibble: 336,776 x 1
##      day
##    <int>
## 1      1
## 2      1
## 3      1
## 4      1
## # ... with 336,772 more rows
select(flights, all_of(day))
## # A tibble: 336,776 x 1
##    month
##    <int>
## 1      1
## 2      1
## 3      1
## 4      1
## # ... with 336,772 more rows
```

Mit `any_of()` statt `all_of()` entsteht kein Fehler, falls ein Wert des übergebenen Vektors nicht in der Tabelle als Spaltenname auftaucht.

```
vars <- c("day", "month", "Jahr")
# select(flights, all_of(vars)) # ERROR
select(flights, any_of(vars))
## # A tibble: 336,776 x 2
##      day month
##    <int> <int>
## 1      1     1
## 2      1     1
## 3      1     1
## 4      1     1
## # ... with 336,772 more rows
```

1.3.2 Condition-Select

Mit *Condition-Select* werden Spalten nach bestimmten Bedingungen ihrer Spaltennamen oder Werte ausgewählt.

Prädikate sind Funktionen, die angewendet auf eine ganze Spalte, ein einzelnes `TRUE` oder `FALSE` ergeben.

Spalten können mit Prädikaten ausgewählt werden (Bedingung an Werte der Spalte). Dabei muss das Prädikat innerhalb der Pseudo-Funktion `where()` stehen. `where()` ist keine Funktion mit Ein- und Ausgabe, sondern dient der Markierung ihres Arguments als Prädikat.

```
select(flights, where(is.character))
## # A tibble: 336,776 x 4
##    carrier tailnum origin dest
##    <chr>   <chr>   <chr>  <chr>
## 1 UA      N14228  EWR    IAH
## 2 UA      N24211  LGA    IAH
## 3 AA      N619AA  JFK    MIA
## 4 B6      N804JB  JFK    BQN
## # ... with 336,772 more rows
```

```
numeric_1000 <- function(x) is.numeric(x) && mean(x, na.rm=T) > 1000
select(flights, where(numeric_1000))
## # A tibble: 336,776 x 7
##    year dep_time sched_dep_time arr_time sched_arr_time flight distance
##   <int>    <int>          <int>    <int>          <int>  <int>    <dbl>
## 1  2013      517            515      830            819   1545     1400
## 2  2013      533            529      850            830   1714     1416
## 3  2013      542            540      923            850   1141     1089
## 4  2013      544            545     1004           1022    725     1576
## # ... with 336,772 more rows
```

Um eine Bedingung an den Namen einer Spalte zu stellen (zB enthält ein bestimmtes RegEx-Muster), können spezielle Funktionen, sogenannte **Select-Helpers** (siehe `?select_helpers`), genutzt werden.

```
select(flights, ends_with("_delay"))
## # A tibble: 336,776 x 2
##   dep_delay arr_delay
##       <dbl>     <dbl>
## 1         2        11
## 2         4        20
## 3         2        33
## 4        -1       -18
## # ... with 336,772 more rows
select(flights, starts_with("dep_"))
## # A tibble: 336,776 x 2
##   dep_time dep_delay
##      <int>     <dbl>
## 1      517         2
## 2      533         4
## 3      542         2
## 4      544        -1
## # ... with 336,772 more rows
select(flights, contains("dep"), day:year) # Condition- und Position-Select in einem Aufruf (aber versc
## # A tibble: 336,776 x 6
##   dep_time sched_dep_time dep_delay   day month  year
##      <int>          <int>     <dbl> <int> <int> <int>
## 1      517            515         2     1     1  2013
## 2      533            529         4     1     1  2013
## 3      542            540         2     1     1  2013
## 4      544            545        -1     1     1  2013
## # ... with 336,772 more rows
select(flights, matches("[aeiou]{2}"))
## # A tibble: 336,776 x 6
##    year carrier tailnum air_time  hour time_hour
##   <int> <chr>   <chr>      <dbl> <dbl> <dttm>
## 1  2013 UA      N14228       227     5 2013-01-01 05:00:00
## 2  2013 UA      N24211       227     5 2013-01-01 05:00:00
## 3  2013 AA      N619AA       160     5 2013-01-01 05:00:00
## 4  2013 B6      N804JB       183     5 2013-01-01 05:00:00
## # ... with 336,772 more rows
```

Zu den Select-Helpers zählt auch `everything()`, womit alle noch nicht selektierten Spalten gewählt werden. Dies kann zum Umordnen der Spalten dienen.

```
select(flights, time_hour, air_time, everything())
## # A tibble: 336,776 x 19
##   time_hour           air_time  year month   day dep_time sched_dep_time
##   <dttm>                 <dbl> <int> <int> <int>    <int>          <int>
## 1 2013-01-01 05:00:00      227  2013     1     1      517            515
## 2 2013-01-01 05:00:00      227  2013     1     1      533            529
## 3 2013-01-01 05:00:00      160  2013     1     1      542            540
## 4 2013-01-01 05:00:00      183  2013     1     1      544            545
## # ... with 336,772 more rows
```

Verschiedene Select-Methoden können mit den logischen Operatoren `&`, `|`, `!` verbunden werden.

```
select(flights, !(contains("dep") | where(is.character)) & 1:arr_delay | hour)
## # A tibble: 336,776 x 7
##    year month   day arr_time sched_arr_time arr_delay  hour
##   <int> <int> <int>    <int>          <int>     <dbl> <dbl>
## 1  2013     1     1      830            819        11     5
## 2  2013     1     1      850            830        20     5
## 3  2013     1     1      923            850        33     5
## 4  2013     1     1     1004           1022       -18     5
## # ... with 336,772 more rows
# Spalten, die weder 'dep' enthalten noch vom Typ character sind und
# aus vor einschließlich 'arr_delay' stehen oder die Spalte 'hour' sind.
```

1.3.3 Spalten umbenennen

Mit `select()` können Spalten umbenannt werden. Allerdings besteht die Ausgabe nur aus den genannten Spalten.

Mit `rename()` werden alle Spalten beibehalten.

```
select(flights, jahr = year, monat = month, tag = day)
## # A tibble: 336,776 x 3
##    jahr monat   tag
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
## 4  2013     1     1
## # ... with 336,772 more rows
rename(flights, jahr = year, monat = month, tag = day)
## # A tibble: 336,776 x 19
##    jahr monat   tag 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
## # ... with 336,772 more rows
```

`select()` ändert die Reihenfolge der Spalten, `rename()` nicht.

```
select(flights, monat = month, tag = day, jahr = year, everything())
## # A tibble: 336,776 x 19
##   monat   tag  jahr dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
```

```
## 1     1     1  2013      517            515         2      830            819
## 2     1     1  2013      533            529         4      850            830
## 3     1     1  2013      542            540         2      923            850
## 4     1     1  2013      544            545        -1     1004           1022
## # ... with 336,772 more rows
rename(flights, monat = month, tag = day, jahr = year)
## # A tibble: 336,776 x 19
##    jahr monat   tag 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
## # ... with 336,772 more rows
```

1.4 mutate()

`mutate(tb, name_1 = val_1, ..., name_n = val_n)` fügt dem Tibble `tb` die Spalten `name_1`, ..., `name_n` mit den Einträgen aus `val_1`, ..., `val_n` hinzu.

`val_1`, ..., `val_n` müssen zu Vektoren der Länge 1 oder `nrow(tb)` evaluieren. Spaltennamen können wie Variablen benutzt werden.

```
# create smaller tibble
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time)

new_col <- nrow(flights_sml):1
mutate(flights_sml, num = new_col, one = 1)
## # A tibble: 336,776 x 9
##    year month   day dep_delay arr_delay distance air_time    num   one
##   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>  <int> <dbl>
## 1  2013     1     1         2        11     1400      227 336776     1
## 2  2013     1     1         4        20     1416      227 336775     1
## 3  2013     1     1         2        33     1089      160 336774     1
## 4  2013     1     1        -1       -18     1576      183 336773     1
## # ... with 336,772 more rows

mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60)
## # A tibble: 336,776 x 9
##    year month   day dep_delay arr_delay distance air_time  gain speed
##   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
## 1  2013     1     1         2        11     1400      227    -9  370.
## 2  2013     1     1         4        20     1416      227   -16  374.
## 3  2013     1     1         2        33     1089      160   -31  408.
## 4  2013     1     1        -1       -18     1576      183    17  517.
## # ... with 336,772 more rows

mutate(flights_sml, dep_delay_rank = rank(dep_delay))
```

```
## # A tibble: 336,776 x 8
##    year month   day dep_delay arr_delay distance air_time dep_delay_rank
##   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>          <dbl>
## 1  2013     1     1         2        11     1400      227         211256
## 2  2013     1     1         4        20     1416      227         222226
## 3  2013     1     1         2        33     1089      160         211256
## 4  2013     1     1        -1       -18     1576      183         174169
## # ... with 336,772 more rows
```

Neue Spalten überschreiben alte mit dem selben Namen.

Siehe in `?mutate` die Argumente `.before`, `.after` zur Bestimmung der Einfügeposition neuer Spalten.

Mit `transmute()` an Stelle von `mutate()` werden nur die angegebenen Spalten zurückgegeben. Siehe auch in `?mutate` das Argument `.keep`.

```
transmute(flights_sml,
  dep_delay,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60)
## # A tibble: 336,776 x 3
##   dep_delay  gain speed
##       <dbl> <dbl> <dbl>
## 1         2    -9  370.
## 2         4   -16  374.
## 3         2   -31  408.
## 4        -1    17  517.
## # ... with 336,772 more rows
```

1.5 summarize()

`summarize(tb, name_1 = val_1, ..., name_n = val_n)` erzeugt aus dem Tibble `tb` ein neues Tibble mit Zeile(n) der Werte `val_1`, ..., `val_n` in den Spalten `name_1`, ..., `name_n`.

`val_1`, ..., `val_n` müssen kompatible Länge haben, dh die selbe Länge oder 1. Spaltennamen können wie Variablen benutzt werden. Länge 1 Elemente werden ggf recyclt.

```
x <- 1:2
summarize(flights,
  x_col = x, # Länge 2
  y_col = "Ypsilon", # Länge 1
  delay_mean = mean(dep_delay, na.rm = TRUE), # Länge 2
  delay_range = range(dep_delay, na.rm = TRUE)) # Länge 1
## # A tibble: 2 x 4
##   x_col y_col   delay_mean delay_range
##   <int> <chr>        <dbl>       <dbl>
## 1     1 Ypsilon       12.6         -43
## 2     2 Ypsilon       12.6        1301
```

Typische Summary-Funktionen sind zB:

- `mean()`, `sd()`, `var()`
- `median()`, `quantile()`, `min()`, `max()`, `range()`
- `n()` (Anzahl der Zeilen), `n_distinct()` (Anzahl verschiedener Elemente)

```
summarize(flights,
  count_flights = n(), # == nrow(flights)
```

```
  count_origin = n_distinct(origin), # short for length(unique(origin))
  mean_pos_dep_delay = mean(dep_delay[dep_delay > 0], na.rm = T),
  missing_delays = sum(is.na(dep_delay) | is.na(arr_delay)))
## # A tibble: 1 x 4
##   count_flights count_origin mean_pos_dep_delay missing_delays
##           <int>        <int>              <dbl>          <int>
## 1        336776            3               39.4           9430
```

1.6 Pipe

Meist werden Verknüpfungen mehrerer Funktionen im Tidyverse mit dem Pipe-Operator `%>%` notiert (wird von `dplyr` automatisch aus dem Paket `magrittr` geladen).

`x %>% f()` ist äquivalent zu `f(x)` und `x %>% f(y)` entspricht `f(x, y)`.

Das linke Argument des Operators `%>%` wird zum ersten Argument des Funktionsaufrufs der rechten Seite.

```
x <- c(1:5, NA)
sqr <- function(x) x^2

# berechne Norm:
# 1. geschachtelt
sqrt(sum(sqr(x), na.rm=T))
## [1] 7.416198

# 2. Zwischenergebnisse
tmp <- sqr(x)
tmp <- sum(tmp, na.rm=T)
tmp <- sqrt(tmp)
tmp
## [1] 7.416198

# 3. Pipe
x %>%
  sqr() %>%
  sum(na.rm=T) %>%
  sqrt()
## [1] 7.416198
```

Um die Position des linken Wertes in der Argumentliste der rechten Funktion zu bestimmen, kann explizit der Variablenname `.` genutzt werden.

```
5 %>% # 5
  `-`(3, .) %>% # 3-5 = -2
  `^`(., 2) %>% # (-2)^2 = 4
  `+`(., 4) # 4 + 4 = 8
## [1] 8
```

Um die Lese- und Ausführungsreihenfolge auch bei Zuweisungen gleichzusetzen, wird der Zuweisungsoperator `->` genutzt.

```
x <- c(2, 3, 6, NA)
norm_x <-
  x %>%
  `[`(1:3)%>%
  `^`(2) %>%
```

```
  sum() %>%
  sqrt()
# oder:
x %>%
  `[`(1:3)%>%
  `^`(2) %>%
  sum() %>%
  sqrt() ->
  norm_x
norm_x
## [1] 7
```

Achtung: Geschachtelten Funktionsaufruf nicht mit `%>%` mischen!

```
x <- 1:4
x %>% mean(sqrt(.))
## Error in mean.default(., sqrt(.)): 'trim' must be numeric of length one
x %>% list(sqrt(.))
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] 1.000000 1.414214 1.732051 2.000000
# . ist 1. Argument von list(), sqrt(.) ist 2. Argument
```

Nutze `Ctrl+Shift+M` (windows) `Cmd+Shift+M` (mac) um den Pipe-Operator schnell einzufügen.

Die Verben in `dplyr` sind auf Nutzung mit dem Pipe-Operator ausgelegt (Tibble immer an erster Stelle).

```
flights %>%
  select(ends_with("delay"), distance, air_time) %>%
  filter(!is.na(dep_delay), !is.na(arr_delay)) %>%
  mutate(
    gain = dep_delay - arr_delay,
    speed = distance / air_time * 60) %>%
  summarize(
    gain_mean = mean(gain),
    speed_mean = var(speed))
## # A tibble: 1 x 2
##   gain_mean speed_mean
##       <dbl>      <dbl>
## 1      5.66      3676.
```

1.7 group by

Mit `group_by()` lässt sich `summarize()` auf Gruppen von Zeilen anwenden.

```
flights %>%
  summarize(
    total_delay_mean = mean(dep_delay, na.rm = TRUE),
    total_delay_var = var(dep_delay, na.rm = TRUE))
## # A tibble: 1 x 2
##   total_delay_mean total_delay_var
##              <dbl>           <dbl>
## 1             12.6           1617.
```

```
flights %>%
  group_by(year, month, day) %>%
  summarize(
    daily_delay_mean = mean(dep_delay, na.rm = TRUE),
    daily_delay_var = var(dep_delay, na.rm = TRUE)) ->
  daily
## `summarise()` has grouped output by 'year', 'month'. You can override using the `.groups` argument.
daily
## # A tibble: 365 x 5
## # Groups:   year, month [12]
##    year month   day daily_delay_mean daily_delay_var
##   <int> <int> <int>            <dbl>           <dbl>
## 1  2013     1     1            11.5            2049.
## 2  2013     1     2            13.9            1384.
## 3  2013     1     3            11.0             990.
## 4  2013     1     4             8.95            769.
## # ... with 361 more rows
```

`group_by(tb, var_1, ..., var_n)` gibt das Tibble `tb` als **gruppiertes Tibble** mit einem zusätzlichen Attribut `groups` zurück.

Für jede in `tb` vorhandene Kombination aus Werten der Spalten `var_1`, ..., `var_n` wird eine Gruppe angelegt. Das Attribut `groups` enthält für jede Gruppe den Index-Vektor aller Zeilen mit den entsprechenden Werten bei `var_1`, ..., `var_n`.

```
class(daily)
## [1] "grouped_df" "tbl_df"     "tbl"        "data.frame"

daily %>%
  group_by(month) %>% # nochmal gruppieren
  attr("groups") ->
  groups
groups # Wert des Attributs "groups"
## # A tibble: 12 x 2
##   month       .rows
##   <int> <list<int>>
## 1     1        [31]
## 2     2        [28]
## 3     3        [31]
## 4     4        [30]
## # ... with 8 more rows
groups[[1,2]] # Indizes der Gruppenelemente Januar
## <list_of<integer>[1]>
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31
groups[[2,2]] # Indizes der Gruppenelemente Februar
## <list_of<integer>[1]>
## [[1]]
##  [1] 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
## [26] 57 58 59
```

Die Gruppierung wird mit `ungroup()` aufgehoben (Attribut entfernt).

```
by_day <- group_by(flights, year, month, day)
attr(ungroup(by_day), "groups")
## NULL
```

`group_cols()` ist ein Selection-Helper um Gruppierungsvariablen mit `select()` auszuwählen.

```
by_day %>% select(group_cols())
## # A tibble: 336,776 x 3
## # Groups:   year, month, day [365]
##    year month   day
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
## 4  2013     1     1
## # ... with 336,772 more rows
```

1.7.1 grouped summarize

Wird `summarize(gtb, name_1 = val_1,... , name_n = val_n)` auf ein gruppiertes Tibble `gtb` angewendet, werden `val_1`, ..., `val_n` für jede Gruppe einzeln ausgewertet und angewendet. Das Rückgabe-Tibble enthält für jede Gruppe eine Zeile.

Siehe oben.

```
flights %>%
  group_by(year, month, day, origin) %>%
  summarize(n = n()) # count flights per day
## `summarise()` has grouped output by 'year', 'month', 'day'. You can override using the `.groups` arg
## # A tibble: 1,095 x 5
## # Groups:   year, month, day [365]
##    year month   day origin     n
##   <int> <int> <int> <chr>  <int>
## 1  2013     1     1 EWR      305
## 2  2013     1     1 JFK      297
## 3  2013     1     1 LGA      240
## 4  2013     1     2 EWR      350
## # ... with 1,091 more rows
# short notation:
flights %>%
  count(year, month, day, origin)
## # A tibble: 1,095 x 5
##    year month   day origin     n
##   <int> <int> <int> <chr>  <int>
## 1  2013     1     1 EWR      305
## 2  2013     1     1 JFK      297
## 3  2013     1     1 LGA      240
## 4  2013     1     2 EWR      350
## # ... with 1,091 more rows
flights %>%
  count(year, month, day, origin, sort=TRUE) # count and sort descending
## # A tibble: 1,095 x 5
##    year month   day origin     n
##   <int> <int> <int> <chr>  <int>
## 1  2013     4    15 EWR      377
```

```
## 2  2013     4    11 EWR      376
## 3  2013     4    18 EWR      376
## 4  2013     4    10 EWR      375
## # ... with 1,091 more rows
```

1.7.2 grouped filter

Wird `filter(gtb, fltr_1, ..., fltr_n)` auf ein gruppiertes Tibble `gtb` angewendet, werden `fltr_1`, ..., `fltr_n` für jede Gruppe einzeln ausgewertet und angewendet.

```
# flights with departure delay larger than average of all flights
flights %>%
  select(origin, dep_delay, everything()) %>% # reorder
  filter(dep_delay > mean(dep_delay, na.rm=TRUE))
## # A tibble: 77,584 x 19
##   origin dep_delay  year month   day dep_time sched_dep_time arr_time
##   <chr>      <dbl> <int> <int> <int>    <int>          <int>    <int>
## 1 LGA           13  2013     1     1      623            610      920
## 2 EWR           24  2013     1     1      632            608      740
## 3 EWR           47  2013     1     1      732            645     1011
## 4 JFK           13  2013     1     1      743            730     1107
## # ... with 77,580 more rows

# flights with departure delay larger than average of flights from same origin
flights %>%
  select(origin, dep_delay, everything()) %>% # reorder
  group_by(origin) %>%
  filter(dep_delay > mean(dep_delay, na.rm=TRUE))
## # A tibble: 76,198 x 19
## # Groups:   origin [3]
##   origin dep_delay  year month   day dep_time sched_dep_time arr_time
##   <chr>      <dbl> <int> <int> <int>    <int>          <int>    <int>
## 1 LGA           13  2013     1     1      623            610      920
## 2 EWR           24  2013     1     1      632            608      740
## 3 EWR           47  2013     1     1      732            645     1011
## 4 JFK           13  2013     1     1      743            730     1107
## # ... with 76,194 more rows

# top 3 most arrival delay in dataset
flights %>%
  select(year:day, arr_delay, everything()) %>% # reorder
  filter(rank(desc(arr_delay)) <= 3)
## # A tibble: 3 x 19
##    year month   day arr_delay dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>     <dbl>    <int>          <int>     <dbl>    <int>
## 1  2013     1     9      1272      641            900      1301     1242
## 2  2013     1    10      1109     1121           1635      1126     1239
## 3  2013     6    15      1127     1432           1935      1137     1607

# top 3 most arrival delay per day
flights %>%
  select(year:day, arr_delay, everything()) %>% # reorder
  group_by(year, month, day) %>%
  filter(rank(desc(arr_delay)) <= 3)
```

```
## # A tibble: 1,085 x 19
## # Groups:   year, month, day [365]
##    year month   day arr_delay dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>     <dbl>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1       851      848           1835       853     1001
## 2  2013     1     1       338     1815           1325       290     2120
## 3  2013     1     1       456     2343           1724       379      314
## 4  2013     1     2       323     1412            838       334     1710
## # ... with 1,081 more rows

# if more than 5e7 miles in dataset, then all flights in dataset, else none
# (not meaningful...)
flights %>%
  select(carrier, distance, everything()) %>% # reorder
  filter(sum(distance) > 5e7)
## # A tibble: 336,776 x 19
##   carrier distance  year month   day dep_time sched_dep_time dep_delay arr_time
##   <chr>      <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1 UA          1400  2013     1     1      517            515         2      830
## 2 UA          1416  2013     1     1      533            529         4      850
## 3 AA          1089  2013     1     1      542            540         2      923
## 4 B6          1576  2013     1     1      544            545        -1     1004
## # ... with 336,772 more rows

# all flights form carriers which fly more than 5e7 miles total
flights %>%
  select(carrier, distance, everything()) %>% # reorder
  group_by(carrier) %>%
  filter(sum(distance) > 5e7)
## # A tibble: 161,410 x 19
## # Groups:   carrier [3]
##   carrier distance  year month   day dep_time sched_dep_time dep_delay arr_time
##   <chr>      <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1 UA          1400  2013     1     1      517            515         2      830
## 2 UA          1416  2013     1     1      533            529         4      850
## 3 B6          1576  2013     1     1      544            545        -1     1004
## 4 DL           762  2013     1     1      554            600        -6      812
## # ... with 161,406 more rows
```

1.7.3 grouped mutate

Wird `mutate(gtb, name_1 = val_1, ..., name_n = val_n)` auf ein gruppiertes Tibble `gtb` angewendet, werden `val_1`, ..., `val_n` für jede Gruppe einzeln ausgewertet.

```
# flight's proportion of total distance
flights %>%
  mutate(prop_distance = distance / sum(distance)) %>%
  select(carrier, distance, prop_distance)
## # A tibble: 336,776 x 3
##   carrier distance prop_distance
##   <chr>      <dbl>         <dbl>
## 1 UA          1400    0.00000400
## 2 UA          1416    0.00000404
## 3 AA          1089    0.00000311
```

```
## 4 B6          1576    0.00000450
## # ... with 336,772 more rows

# flight's proportion of total distance for each carrier
flights %>%
  group_by(carrier) %>%
  mutate(prop_distance = distance / sum(distance)) %>%
  select(carrier, distance, prop_distance)
## # A tibble: 336,776 x 3
## # Groups:   carrier [16]
##   carrier distance prop_distance
##   <chr>      <dbl>         <dbl>
## 1 UA          1400     0.0000156
## 2 UA          1416     0.0000158
## 3 AA          1089     0.0000248
## 4 B6          1576     0.0000270
## # ... with 336,772 more rows
```

1.8 Column-wise operations

Mit `across()` wenden wir Operationen, die durch Verben wie `summarize()`, `mutate()`, `filter()`, ..., bereitgestellt werden, auf eine Menge von Spalten an.

Dabei wird diese Menge an Spalten mit der Tidy-Select-Syntax von `select()` ausgewählt.

```
tb <- tibble(let=letters[1:5], unif=runif(5), norm=rnorm(5), exp=rexp(5))
tb %>%
  summarize(across(where(is.numeric), mean))
## # A tibble: 1 x 3
##    unif   norm   exp
##   <dbl>  <dbl> <dbl>
## 1 0.336 -0.259  1.36

rescale01 <- function(x) {
  rng <- range(x, na.rm = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}
tb %>%
  mutate(across(unif:exp, rescale01))
## # A tibble: 5 x 4
##   let     unif  norm   exp
##   <chr>  <dbl> <dbl> <dbl>
## 1 a     0.0887 0.506 0.923
## 2 b     1      0.226 0.583
## 3 c     0.718  1     1
## 4 d     0.181  0     0.552
## # ... with 1 more row

tb %>%
  filter(across(-let, function(x) x>0))
## # A tibble: 1 x 4
##   let    unif  norm   exp
##   <chr> <dbl> <dbl> <dbl>
## 1 c     0.601 0.747  2.10
```

Das erste Argument von `across(cols, fun)` beschreibt in der Tidy-Select-Syntax die Gruppe der Spalten. Das zweite Argument ist eine einzelne Funktion oder eine Liste von Funktionen.

Für `summarize()` müssen die Funktionen auf allen ausgewählten Spalten einen Vektor der selben Länge oder 1 ergeben; `mutate()` Länge `nrow(tb)` oder 1.

Für `filter()` müssen die Funktionen angewendet auf Spalten `logical`-Vektoren der Länge `nrow(tb)` oder 1 ergeben.

Der angewendeten Funktion können weitere konstante Argumente übergeben werden.

```
tb[1,2] <- NA
tb %>%
  summarize(across(where(is.numeric), mean, na.rm=TRUE))
## # A tibble: 1 x 3
##    unif   norm   exp
##   <dbl>  <dbl> <dbl>
## 1 0.400 -0.259  1.36
```

Mit dem Argument `.names` werden für `mutate()` und `summarize()` die Namen der neuen Spalten bestimmt. Dabei wird `glue`-Syntax verwendet, wobei Variablen `fn` und `col` für Listenname der Funktionen bzw ursprünglicher Spaltenname stehen.

```
tb %>%
  summarise(across(where(is.numeric), list(mini=min, maxi=max), na.rm=T, .names = "{fn}.{col}"))
## # A tibble: 1 x 6
##   mini.unif maxi.unif mini.norm maxi.norm mini.exp maxi.exp
##       <dbl>     <dbl>     <dbl>     <dbl>    <dbl>    <dbl>
## 1   0.00740     0.834    -0.935     0.747    0.201     2.10
```

1.9 Row-wise operations

Die meisten Funktionen in R sind auf irgendeine Weise vektorisiert. Möchten wir in jeder Zeile eines Tibbles die Summe von der Einträge von zwei Spalten berechnen, machen wir dies nicht mit einer Schleife, sondern addieren die beiden Spalten als Vektoren mit dem vektorisierten `+` Operator.

Solche Spalten-Operationen sind immer Zeilen-Operationen vorzuziehen.

Dennoch kann es in seltenen Fällen passieren, dass zeilenweises Ausführen von Funktionen auf einer Tabelle nötig ist.

Dann kann `rowwise(tb)` eingesetzt werden. Im Wesentlichen entspricht dies `group_by(tb, 1:nrow(tb))`. Dh jede Zeile ist in einer eigenen Gruppe.

```
tb <- tibble(x = 1:2, y = 3:4, z = 5:6)
tb %>% rowwise() -> rtb
rtb
## # A tibble: 2 x 3
## # Rowwise:
##       x     y     z
##   <int> <int> <int>
## 1     1     3     5
## 2     2     4     6
tb %>% mutate(m = mean(c(x, y, z)))
## # A tibble: 2 x 4
##       x     y     z     m
##   <int> <int> <int> <dbl>
## 1     1     3     5   3.5
```

```
## 2     2     4     6   3.5
rtb %>% mutate(m = mean(c(x, y, z)))
## # A tibble: 2 x 4
## # Rowwise:
##       x     y     z     m
##   <int> <int> <int> <dbl>
## 1     1     3     5     3
## 2     2     4     6     4
```

Wird `rowwise(tb, id_1, ..., id_n)` noch eine oder mehrere `id`-Spalten übergeben, zählen diese als Gruppierungsvariablen. Damit bleiben sie beim Aufruf mit `summarize()` erhalten.

```
tb <- tibble(id = letters[1:6], w = 10:15, x = 20:25, y = 30:35, z = 40:45)
tb
## # A tibble: 6 x 5
##   id        w     x     y     z
##   <chr> <int> <int> <int> <int>
## 1 a        10    20    30    40
## 2 b        11    21    31    41
## 3 c        12    22    32    42
## 4 d        13    23    33    43
## # ... with 2 more rows
rtb <- tb %>% rowwise(id)
rtb %>% mutate(total = sum(c(w, x, y, z)))
## # A tibble: 6 x 6
## # Rowwise:  id
##   id        w     x     y     z total
##   <chr> <int> <int> <int> <int> <int>
## 1 a        10    20    30    40   100
## 2 b        11    21    31    41   104
## 3 c        12    22    32    42   108
## 4 d        13    23    33    43   112
## # ... with 2 more rows
rtb %>% summarize(total = sum(c(w, x, y, z)))
## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.
## # A tibble: 6 x 2
## # Groups:   id [6]
##   id    total
##   <chr> <int>
## 1 a       100
## 2 b       104
## 3 c       108
## 4 d       112
## # ... with 2 more rows
```

Um Spalten mit Tidy-Select-Syntax auszuwählen, nutze `c_across()`.

```
rtb %>% mutate(total = sum(c_across(w:z)))
## # A tibble: 6 x 6
## # Rowwise:  id
##   id        w     x     y     z total
##   <chr> <int> <int> <int> <int> <int>
## 1 a        10    20    30    40   100
## 2 b        11    21    31    41   104
## 3 c        12    22    32    42   108
```

```
## 4 d        13    23    33    43   112
## # ... with 2 more rows
rtb %>% summarize(total = sum(c_across(where(is.numeric))))
## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.
## # A tibble: 6 x 2
## # Groups:   id [6]
##   id    total
##   <chr> <int>
## 1 a       100
## 2 b       104
## 3 c       108
## 4 d       112
## # ... with 2 more rows
```

Wie bei `group_by()` wird die zeilenweise Gruppierung mit `ungroup()` aufgehoben.

```
# compute the proportion of total for each column:
rtb %>%
  mutate(total = sum(c_across(w:z))) %>%
  ungroup() %>%
  mutate(across(w:z, function(x) x/total))
## # A tibble: 6 x 6
##   id        w     x     y     z total
##   <chr> <dbl> <dbl> <dbl> <dbl> <int>
## 1 a     0.1   0.2   0.3   0.4     100
## 2 b     0.106 0.202 0.298 0.394   104
## 3 c     0.111 0.204 0.296 0.389   108
## 4 d     0.116 0.205 0.295 0.384   112
## # ... with 2 more rows
```

1.10 Non-Standard Evaluation

Viele `dplyr`-Funktionen nutzen **Non-Standard Evaluation** ihrer Argumente. Dh sie folgen nicht den üblichen Regeln, wie call-by-value. Stattdessen wird der gesamte als Argument eingegebene Ausdruck von den Funktionen verwertet.

So ist es möglich bestimmte Anweisungen zu vereinfachen.

```
tb <- tibble(x=1:3, y=3:1)
filter(tb, x < 3, y > 1) # x, y nicht als Variablen vorhanden
## # A tibble: 2 x 2
##       x     y
##   <int> <int>
## 1     1     3
## 2     2     2
tb[tb$x < 3 & tb$y > 1, ] # ohne Non-Standard Evaluation
## # A tibble: 2 x 2
##       x     y
##   <int> <int>
## 1     1     3
## 2     2     2
```

Dies vereinfacht bestimmte Funktionsaufrufe, bringt aber auch Nachteile mit sich.

Abhängig davon, welche Spaltennamen in `tb` vorhanden sind, hat `filter(tb, x==y)` verschiedene Bedeutungen. Zuerst wird in `tb` nach entsprechenden Spaltennamen gesucht. Nur falls ein Name dort nicht gefunden

wird, wird außerhalb gesucht.

```
x <- 1
y <- 2
tb <- tibble(x=1:3, y=3:1)
filter(tb, x==y) # ist gleich:
## # A tibble: 1 x 2
##       x     y
##   <int> <int>
## 1     2     2
tb[tb$x == tb$y, ]
## # A tibble: 1 x 2
##       x     y
##   <int> <int>
## 1     2     2
```

```
x <- 1
y <- 2
tb <- tibble(u=1:3, y=3:1)
filter(tb, x==y) # ist gleich:
## # A tibble: 1 x 2
##       u     y
##   <int> <int>
## 1     3     1
tb[x == tb$y, ]
## # A tibble: 1 x 2
##       u     y
##   <int> <int>
## 1     3     1
```

```
x <- 1
y <- 2
tb <- tibble(x=1:3, v=3:1)
filter(tb, x==y) # ist gleich:
## # A tibble: 1 x 2
##       x     v
##   <int> <int>
## 1     2     2
tb[tb$x == y, ]
## # A tibble: 1 x 2
##       x     v
##   <int> <int>
## 1     2     2
```

```
x <- 1
y <- 2
tb <- tibble(u=1:3, v=3:1)
filter(tb, x==y) # ist gleich:
## # A tibble: 0 x 2
tb[x == y, ]
## # A tibble: 0 x 2
```

Außerdem sind solche Argumente nicht *referentially transparent*, dh Werte können nicht immer direkt durch eine äquivalente Variable ersetzt werden.

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

```
## # A tibble: 1 x 2
##       x     y
##   <int> <int>
## 1     1     3
my_var <- "x"
filter(tb, my_var == 1)
## # A tibble: 0 x 2
```

Eine Lösung für letzteres Problem ist, explizit das Daten-Tibble anzugeben. Dies kann auch mit `.data` geschehen, was immer das erste Argument referenziert (nützlich für `%>%`).

```
filter(tb, tb[[my_var]] == 1)
## # A tibble: 1 x 2
##       x     y
##   <int> <int>
## 1     1     3
filter(tb, .data[[my_var]] == 1)
## # A tibble: 1 x 2
##       x     y
##   <int> <int>
## 1     1     3
srt <- "dep_delay"
fltr <- "dep_time"
flights %>%
  arrange(.data[[srt]]) %>%
  filter(.data[[fltr]] < 500)
## # A tibble: 1,484 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     5     8      445            500       -15      620            640
## 2  2013     5     5      446            500       -14      636            640
## 3  2013     9     4      446            500       -14      618            648
## 4  2013    10     1      447            500       -13      614            648
## # ... with 1,480 more rows
```

Mehr zum Umgang mit non-standard Evaluation unter https://cran.r-project.org/web/packages/dplyr/vignettes/programming.html und in einem späteren Kapitel des Kurses (*Meta-Programmierung*).

2 Verben für zwei Tabellen

Wie im Kapitel Tibbles beschrieben, fügen `bind_rows()` und `bind_cols()` Tibbles mit kompatibler Struktur zusammen, unabhängig von konkreten Werten.

In diesem Kapitel zeigen wir, wie wir Tabellen abhängig vom Vorkommen von Werten einer Tabelle in der anderen zusammenfügen.

2.1 Mutating Joins

Das Dataset `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.

```
library(tidyverse)
library(nycflights13)
```

```
flights2 <- flights %>%
  select(year:day, hour, tailnum, carrier)

flights2
## # A tibble: 336,776 x 6
##    year month   day  hour tailnum carrier
##   <int> <int> <int> <dbl> <chr>   <chr>
## 1  2013     1     1     5 N14228  UA
## 2  2013     1     1     5 N24211  UA
## 3  2013     1     1     5 N619AA  AA
## 4  2013     1     1     5 N804JB  B6
## # ... with 336,772 more rows
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
## # ... with 12 more rows
```

Soll nun zu einem Flug der Langname der Airline angezeigt werden, müssen beide Tabellen genutzt werden.

`left_join(tb1, tb2, by)` erzeugt ein Tibble mit den Spalten aus `tb1` und `tb2`. `by` benennt ein Paar aus Spalten, je eine aus `tb1` und `tb2`. Sie werden als **Key** bezeichnet. Zeilen der Ausgabe setzen sich aus Zeilen aus `tb1` und `tb2` zusammen, deren Key-Werte übereinstimmen.

```
flights2 %>%
  left_join(airlines, by = "carrier")
## # A tibble: 336,776 x 7
##    year month   day  hour tailnum carrier name
##   <int> <int> <int> <dbl> <chr>   <chr>   <chr>
## 1  2013     1     1     5 N14228  UA      United Air Lines Inc.
## 2  2013     1     1     5 N24211  UA      United Air Lines Inc.
## 3  2013     1     1     5 N619AA  AA      American Airlines Inc.
## 4  2013     1     1     5 N804JB  B6      JetBlue Airways
## # ... with 336,772 more rows
```

Das Argument `by` gibt an, welche Variablen zum Vereinigen der Tabellen genutzt werden.

- `by = NULL` (default): nutze die Variable, die in beiden Tabellen gleich heißt
- `by = "name"`: nutze Variable `name`, die in beiden Tabellen vorkommt.
- `by = c("name1" = "name2")`: verbinde Variable `name1` aus `tb1` mit `name2` aus `tb2`.

```
flights2 <- flights %>%
  select(month:day, hour, origin, dest, tailnum)
airports2 <- airports %>% select(1:2)

flights2
## # A tibble: 336,776 x 6
##   month   day  hour origin dest  tailnum
##   <int> <int> <dbl> <chr>  <chr> <chr>
## 1     1     1     5 EWR    IAH   N14228
## 2     1     1     5 LGA    IAH   N24211
## 3     1     1     5 JFK    MIA   N619AA
```

```
## 4     1     1     5 JFK    BQN   N804JB
## # ... with 336,772 more rows
airports2
## # A tibble: 1,458 x 2
##   faa   name
##   <chr> <chr>
## 1 04G   Lansdowne Airport
## 2 06A   Moton Field Municipal Airport
## 3 06C   Schaumburg Regional
## 4 06N   Randall Airport
## # ... with 1,454 more rows

flights2 %>%
  left_join(airports2, c("dest" = "faa"))
## # A tibble: 336,776 x 7
##   month   day  hour origin dest  tailnum name
##   <int> <int> <dbl> <chr>  <chr> <chr>   <chr>
## 1     1     1     5 EWR    IAH   N14228  George Bush Intercontinental
## 2     1     1     5 LGA    IAH   N24211  George Bush Intercontinental
## 3     1     1     5 JFK    MIA   N619AA  Miami Intl
## 4     1     1     5 JFK    BQN   N804JB  <NA>
## # ... with 336,772 more rows
flights2 %>%
  left_join(airports2, c("origin" = "faa"))
## # A tibble: 336,776 x 7
##   month   day  hour origin dest  tailnum name
##   <int> <int> <dbl> <chr>  <chr> <chr>   <chr>
## 1     1     1     5 EWR    IAH   N14228  Newark Liberty Intl
## 2     1     1     5 LGA    IAH   N24211  La Guardia
## 3     1     1     5 JFK    MIA   N619AA  John F Kennedy Intl
## 4     1     1     5 JFK    BQN   N804JB  John F Kennedy Intl
## # ... with 336,772 more rows
flights2 %>%
  left_join(airports2, c("origin" = "faa")) %>%
  left_join(airports2, c("dest" = "faa"))
## # A tibble: 336,776 x 8
##   month   day  hour origin dest  tailnum name.x           name.y
##   <int> <int> <dbl> <chr>  <chr> <chr>   <chr>            <chr>
## 1     1     1     5 EWR    IAH   N14228  Newark Liberty ~ George Bush Intercont~
## 2     1     1     5 LGA    IAH   N24211  La Guardia       George Bush Intercont~
## 3     1     1     5 JFK    MIA   N619AA  John F Kennedy ~ Miami Intl
## 4     1     1     5 JFK    BQN   N804JB  John F Kennedy ~ <NA>
## # ... with 336,772 more rows
flights2 %>%
  left_join(airports2, c("origin" = "faa")) %>%
  left_join(
    airports2,
    c("dest" = "faa"),
    suffix=c("_origin", "_dest"))
## # A tibble: 336,776 x 8
##   month   day  hour origin dest  tailnum name_origin       name_dest
##   <int> <int> <dbl> <chr>  <chr> <chr>   <chr>              <chr>
## 1     1     1     5 EWR    IAH   N14228  Newark Liberty I~ George Bush Intercon~
```

```
## 2     1     1     5 LGA    IAH   N24211  La Guardia        George Bush Intercon~
## 3     1     1     5 JFK    MIA   N619AA  John F Kennedy I~ Miami Intl
## 4     1     1     5 JFK    BQN   N804JB  John F Kennedy I~ <NA>
## # ... with 336,772 more rows
```

Existiert ein Key-Wert mehrere Male in einer der beiden Spalten, werden alle passenden Kombinationen an Zeilen hinzugefügt.

```
x <- tibble(
  key = c(1, 2, 2, 3, 3),
  val_x = c("x1", "x2a", "x2b", "x3a", "x3b"))
y <- tibble(
  key = c(1, 1, 2, 3, 3),
  val_y = c("y1a", "y1b", "y2", "y3a", "y3b"))
left_join(x, y)
## Joining, by = "key"
## # A tibble: 8 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1a
## 2     1 x1    y1b
## 3     2 x2a   y2
## 4     2 x2b   y2
## # ... with 4 more rows
```

Werden mehrere Keys angegeben, müssen die Werte aller Keys übereinstimmen, um eine vollständige Ausgabezeile zu erhalten.

```
x <- tibble(
  key1 = c(1, 1, 2, 2, 3),
  key2 = c(1, 2, 1, 2, 3),
  val = c("x1", "x2", "x3", "x4", "x5"))
y <- tibble(
  key1 = c(1, 1, 2, 2, 3),
  key2 = c(1, 2, 1, 2, 4),
  val = c("y1", "y2", "y3", "y4", "y5"))
left_join(x, y, by = c("key1", "key2"))
## # A tibble: 5 x 4
##    key1  key2 val.x val.y
##   <dbl> <dbl> <chr> <chr>
## 1     1     1 x1    y1
## 2     1     2 x2    y2
## 3     2     1 x3    y3
## 4     2     2 x4    y4
## # ... with 1 more row
```

`left_join()`, `right_join()`, `full_join()`, `inner_join()` unterscheiden sich darin, was passiert, wenn ein Key-Wert nur in einer von beiden Tabellen vorkommt.

```
x <- tibble(
  key = 1:3,
  val_x = c("x1", "x2", "x3"))
y <- tibble(
  key = c(1, 2, 4),
  val_y = c("y1", "y2", "y3"))
x %>%
```

```
  left_join(y, by = "key")
## # A tibble: 3 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>
x %>%
  right_join(y, by = "key")
## # A tibble: 3 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     4 <NA>  y3
x %>%
  full_join(y, by = "key")
## # A tibble: 4 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>
## 4     4 <NA>  y3
x %>%
  inner_join(y, by = "key")
## # A tibble: 2 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
```

2.2 Filtering joins

Filtering joins wählen eine Teilmenge von Zeilen einer Tabelle aus, basierend auf dem Vorkommen von Key-Werten in einer zweiten Tabelle.

- `semi_join(tb1, tb2, by)`: wählt alle Beobachtungen in `tb1`, deren Key in `tb2` vorkommt.
- `anti_join(tb1, tb2, by)`: wählt alle Beobachtungen in `tb1`, deren Key nicht in `tb2` vorkommt.

```
# create tibble of top10 destinations
top_dest <- flights %>%
  count(dest, sort = TRUE) %>%
  head(10)
top_dest
## # A tibble: 10 x 2
##   dest      n
##   <chr> <int>
## 1 ORD   17283
## 2 ATL   17215
## 3 LAX   16174
## 4 BOS   15508
## # ... with 6 more rows
```

```
flights %>%
  semi_join(top_dest)
## Joining, by = "dest"
## # A tibble: 141,145 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      542            540         2      923            850
## 2  2013     1     1      554            600        -6      812            837
## 3  2013     1     1      554            558        -4      740            728
## 4  2013     1     1      555            600        -5      913            854
## # ... with 141,141 more rows

# dies entspricht
flights %>%
  filter(dest %in% top_dest$dest)
## # A tibble: 141,145 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      542            540         2      923            850
## 2  2013     1     1      554            600        -6      812            837
## 3  2013     1     1      554            558        -4      740            728
## 4  2013     1     1      555            600        -5      913            854
## # ... with 141,141 more rows
```

```
# finde flights mit tailnum, die nicht einem Flugzeug aus planes zugeordnet werden kann
flights %>%
  anti_join(planes, by = "tailnum") %>%
  count(tailnum, sort = TRUE)
## # A tibble: 722 x 2
##    tailnum     n
##    <chr>   <int>
## 1 <NA>     2512
## 2 N725MQ    575
## 3 N722MQ    513
## 4 N723MQ    507
## # ... with 718 more rows
```

3 Relationale Datenbanken

Wir wenden uns wieder kurz der Theorie **Relationalen Datenbanken** und der **relationalen Algebra** zu und zeigen, wie einige der oben vorgestellten Funktionen im Kontext von Relationen formalisiert werden können.

3.1 Erinnerung 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**.

3.2 Operationen

Aus dem vorigen Kapitel sind bereits folgende Operationen auf Relationen bekannt:

- Vereinigung `union()`
- Schnitt `intersect()`
- Differenz: `setdiff()`
- Symmetrische Differenz
- Kartesisches Produkt: `tidyr::crossing()`

3.2.1 Projektion

Seien $R \subseteq W_1 \times \cdots \times W_m$ eine Relation und $\beta = (j_1, \ldots, j_\ell)$ mit $j_1, \ldots, j_\ell \in \{1, \ldots, m\}$.

Die Projektion von R auf β ist

- $\pi_\beta(R) = \{t_\beta | t \in R\}$, wobei $t_\beta = (t_{j_1}, \ldots, t_{j_\ell})$ für $t = (t_1, \ldots, t_m)$.

```
# Schema (integer, character, logical)
R <- tibble(
  x = c(1L, 1L, 1L, 2L),
  y = letters[1:4],
  z = c(T, F, F, T))
R
## # A tibble: 4 x 3
##       x y     z
##   <int> <chr> <lgl>
## 1     1 a     TRUE
## 2     1 b     FALSE
## 3     1 c     FALSE
## 4     2 d     TRUE
beta <- c(1,3)
unique(R[beta])
## # A tibble: 3 x 2
##       x z
##   <int> <lgl>
## 1     1 TRUE
## 2     1 FALSE
## 3     2 TRUE
# oder
R %>%
  select(all_of(beta)) %>%
  distinct()
## # A tibble: 3 x 2
##       x z
##   <int> <lgl>
## 1     1 TRUE
## 2     1 FALSE
## 3     2 TRUE
```

3.2.2 Selektion

Seien $R \subseteq W_1 \times \cdots \times W_m$ und p ein Prädikat, dh eine (nicht notwendigerweise endliche) Teilmenge $p \subseteq W_1 \times \cdots \times W_m$.

In der Regel wird p durch einen logischen Ausdruck A beschrieben, etwa $t_2 > 0$. Dies steht für die Menge $\{t \in W_1 \times \cdots \times W_m | t_2 > 0\}$.

- Selektion $\sigma_p(R) := \{t \in R | t \in p\}$, $\sigma_A(R) := \{t \in R | t \text{ erfüllt } A\}$.

```r
R <- tibble(
  x = c(1:4),
  y = letters[1:4])
R[R$x>2,]
## # A tibble: 2 x 2
##       x y
##   <int> <chr>
## 1     3 c
## 2     4 d
# oder
R %>% filter(x > 2)
## # A tibble: 2 x 2
##       x y
##   <int> <chr>
## 1     3 c
## 2     4 d
```

Achtung: Eine Selektion im Sinne relationaler Datenbanken entspricht der `dplyr`-Funktion `filter()`, eine Projektion entspricht `select()`.

3.2.3 Verbund (Join)

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

- Verbund $R \bowtie_p S := \sigma_p(R \times S)$.

Ein natürlicher Verbund (natural join) ist ein Verbund mit Gleichheitsbedingung zusammen mit einer Projektion zur Entfernung der duplizierten Spalten.

Seien $R \subseteq U_1 \times \cdots \times U_\ell \times W_1 \times \cdots \times W_m$ und $S \subseteq W_1 \times \cdots \times W_m \times V_1 \times \cdots \times V_k$.

- natürlicher Verbund $R \bowtie S := \{(u_1, \ldots, u_\ell, w_1, \ldots, w_m, v_1, \ldots, v_k) | (u_1, \ldots, u_\ell, w_1, \ldots, w_m) \in R \wedge (w_1, \ldots, w_m, v_1, \ldots, v_k) \in S\}$.

```r
R <- tibble(
  x = c("x1", "x2", "x3"),
  key = 1:3)
S <- tibble(
  key = c(1, 2, 4),
  y = c("y1", "y2", "y3"))
inner_join(R, S, by="key") #natürlicher Verbund
## # A tibble: 2 x 3
##   x       key y
##   <chr> <dbl> <chr>
## 1 x1        1 y1
## 2 x2        2 y2
```

- Semi Join $R \ltimes S := \{(u_1, \ldots, u_\ell, w_1, \ldots, w_m) \in R | \exists v_j \in V_j \colon (w_1, \ldots, w_m, v_1, \ldots, v_k) \in S\}$

```r
R <- tibble(
  x = c("x1", "x2", "x3"),
  key = 1:3)
S <- tibble(
  key = c(1, 2, 4),
  y = c("y1", "y2", "y3"))
semi_join(R, S, by="key")
## # A tibble: 2 x 2
```

```
##   x       key
##   <chr> <int>
## 1 x1        1
## 2 x2        2
```

3.2.4 Division

Seien $R \subseteq W_1 \times \cdots \times W_\ell \times W_{\ell+1} \times \cdots \times W_m$ und $S \subseteq W_{\ell+1} \times \cdots \times W_m$ Relationen. Definiere $\beta = (1, \ldots, \ell)$.

- Division: $R \div S := \pi_\beta(R) \setminus \pi_\beta((\pi_\beta(R) \times S) \setminus R)$.

Die Division $R \div S$ enthält alle Tupel $u \in W_1 \times \cdots \times W_\ell$ mit $\{u\} \times S \subseteq R$.

3.2.5 Rechenregeln

Mit einer mathematischen Definition ist es einfacher Rechenregeln der Relations- oder Tabellen-Operationen zu finden.

Diese Rechenregeln können genutzt werden, um eine Operationsfolge durch eine effizientere Operationsfolge mit dem gleichen Ergebnis zu ersetzen.

Hier sind ein paar Beispiele für Rechenregeln:

Seien R, S, T Relationen und A, B, C Ausdrücke, die eine Bedingung für eine Selektion σ beschreiben.

- $(R \times S) \cup (R \times T) = R \times (S \cup T)$
- $(R \times S) \div S = R$ für kompatible R, S im Sinne der Definition von Division
- $\sigma_A(R) = \sigma_A(\sigma_A(R))$
- $\sigma_A(\sigma_B(R)) = \sigma_B(\sigma_A(R))$
- $\sigma_{A \vee B}(R) = \sigma_A(R) \cup \sigma_A(R)$
- $\sigma_{A \wedge B}(R) = \sigma_A(\sigma_B(R))$
- Teile $D = A \wedge B \wedge C$ so auf, dass A nur Bedingungen ab R und B nur Bedingungen an S stellt. Dann gilt $\sigma_{A \wedge B \wedge C}(R \times S) = \sigma_C(\sigma_A(R) \times \sigma_B(S))$.
- $\sigma_A(R \setminus S) = \sigma_A(R) \setminus \sigma_A(S) = \sigma_A(R) \setminus S$
- $\sigma_A(R \cup S) = \sigma_A(R) \cup \sigma_A(S)$
- $\sigma_A(R \cap S) = \sigma_A(R) \cap \sigma_A(S) = \sigma_A(R) \cap S$
- $\pi_\beta(\sigma_A(R)) = \sigma_A(\pi_\beta(R))$ für eine Menge an Spalten β, sodass A nur Bedingungen an die Spalten in β stellt
- $\pi_\alpha(\pi_\beta(R)) = \pi_\alpha(R)$ für $\alpha \subseteq \beta$
- $\pi_\beta(R \cup S) = \pi_\beta(R) \cup \pi_\beta(S)$