# Aggregate

Wolfgang Peter

## stp25aggregate

### Daten Laden

GetData(): Ladet verschiedene Dateiformate von csv bis sav. Tabellen im Text-Format koennen direkt gelesen werden. Zurueckgegeben wird ein data.frame.

```
dat<-GetData("
sex treatment control
m  2 3
f  3 4
",
Tabel_Expand = TRUE, id.vars = 1)
#>
#>
#> File:
#>
#> sex treatment control
#> m  2 3
#> f  3 4
#>    character
#>
#> read.text2
#> Mon Jan 27 06:29:25 2020
head(dat)
#>   sex     value
#> 1   m treatment
#> 2   m treatment
#> 3   f treatment
#> 4   f treatment
#> 5   f treatment
#> 6   m   control
#xtabs(~sex +value, dat)
```

Andere Packete für SPSS und XLSX

```
library("readxl")

DF <- read_excel("Raw data/MIH Fragenkatalog.xlsx")
```

```
library("rio")
# Excel (.xls and .xlsx), using haven::read_excel.
# SPSS (.sav), using haven::read_sav
```

Data and Variable Transformation Functions

## Long/Wide

Umformen von einem Breit-Format nach einem Lang-Format. Melt2 und melt2 sind Erweiterungen der reshape2::melt Funktion. Intern wird *melt* und *dcast* verwendet.

```
df <- tibble::tibble(
   month=c(1,2,3,1,2,3),
   student= gl(2,3, labels =c("Amy", "Bob")),
   A=c(9,7,6,8,6,9),
   B=c(6,7,8,5,6,7)
)
df
#> # A tibble: 6 x 4
#>   month student     A     B
#>   <dbl> <fct>   <dbl> <dbl>
#> 1     1 Amy         9     6
#> 2     2 Amy         7     7
#> 3     3 Amy         6     8
#> 4     1 Bob         8     5
#> 5     2 Bob         6     6
#> 6     3 Bob         9     7
```

### Vergleich der Ergebnisse spread vs Wide

```
    tidyr::spread(df[-4], student, A)
#> # A tibble: 3 x 3
#>   month   Amy   Bob
#>   <dbl> <dbl> <dbl>
#> 1     1     9     8
#> 2     2     7     6
#> 3     3     6     9

# new style with pivot_wider
df[-4] %>%
  tidyr::pivot_wider(names_from = student,
                     values_from = A,
                     names_prefix = "Student_")
#> # A tibble: 3 x 3
#>   month Student_Amy Student_Bob
#>   <dbl>       <dbl>       <dbl>
#> 1     1           9           8
#> 2     2           7           6
#> 3     3           6           9
```

```
    (df_w1 <- Wide(df[-4], student, A))
#> # A tibble: 3 x 3
#>   month   Amy   Bob
#>   <dbl> <dbl> <dbl>
#> 1     1     9     8
#> 2     2     7     6
#> 3     3     6     9
```

### Vergleich der Ergebnisse gather vs Long

```
     tidyr::gather(df_w1,  key = "student", value = "A", Amy, Bob)
#> # A tibble: 6 x 3
#>   month student     A
#>   <dbl> <chr>   <dbl>
#> 1     1 Amy         9
#> 2     2 Amy         7
#> 3     3 Amy         6
#> 4     1 Bob         8
#> 5     2 Bob         6
#> 6     3 Bob         9
     Long(Amy + Bob ~ month, df_w1, key="student", value="A")
#> # A tibble: 6 x 3
#>   month student     A
#>   <dbl> <fct>   <dbl>
#> 1     1 Amy         9
#> 2     2 Amy         7
#> 3     3 Amy         6
#> 4     1 Bob         8
#> 5     2 Bob         6
#> 6     3 Bob         9
     Long(df_w1, id.vars=1, key = "student", value = "A")
#> # A tibble: 6 x 3
#>   month student     A
#>   <dbl> <fct>   <dbl>
#> 1     1 Amy         9
#> 2     2 Amy         7
#> 3     3 Amy         6
#> 4     1 Bob         8
#> 5     2 Bob         6
#> 6     3 Bob         9
```

```
df[-4] %>%
  tidyr::pivot_wider(names_from = student,
                     values_from = A,
                     names_prefix = "Student_") %>%
  knitr::kable()# %>%
```

| month | Student_Amy | Student_Bob |
|---:|---:|---:|
| 1 | 9 | 8 |
| 2 | 7 | 6 |
| 3 | 6 | 9 |

```
  #kableExtra::kable_styling()
```

### Long mit meheren Parametern

```
    (df_w2 <- Wide(df, student, c(A, B)))
#> # A tibble: 3 x 5
#>   month Amy_A Amy_B Bob_A Bob_B
#>   <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1     1     9     6     8     5
#> 2     2     7     7     6     6
#> 3     3     6     8     9     7
# # new style with pivot_longer
     Long(list(A=c("Amy_A", "Bob_A" ), B=c("Amy_B", "Bob_B")), df_w2,
                 by =  ~ month,
                 key = "student",
                 key.levels= c("Amy", "Bob"))
#> # A tibble: 6 x 4
#>   month student     A     B
#>   <dbl> <fct>   <dbl> <dbl>
#> 1     1 Amy         9     6
#> 2     2 Amy         7     7
#> 3     3 Amy         6     8
#> 4     1 Bob         8     5
#> 5     2 Bob         6     6
#> 6     3 Bob         9     7
```

### Berechnen

```
mean3<- function(x)round(mean(x, na.rm=TRUE), 1)

    Summarise(A + B ~ student, df, mean3, key = "group", value = "cbc")
#>   student group cbc
#> 1     Amy     A 7.3
#> 2     Amy     B 7.0
#> 3     Bob     A 7.7
#> 4     Bob     B 6.0
    Summarise(A + B ~ student, df, mean3,  margins = TRUE)
#>    student variable value
#> 1      Amy        A   7.3
#> 2      Amy        B   7.0
#> 3      Bob        A   7.7
#> 4      Bob        B   6.0
#> 11  gesamt        A   7.5
#> 21  gesamt        B   6.5
    Summarise(A + B ~ student,
                      df,
                      mean3,
                      formula = variable ~ student,
                      margins = TRUE)
#>   variable Amy Bob (all)
#> 1        A 7.3 7.7   7.5
#> 2        B 7.0 6.0   6.5
#> 3    (all) 7.2 6.8   7.0
```

### Aufdröseln vom Mehrfachantworten

Die Funktion separate_multiple_choice() transformiert einen String mit Trennzeichen zu einem Multi-Set mit 0 und 1. (Separate multiple choice) der param x ist entweder ein Character oder eine zahl

```
    dat <-  data.frame(
      Q1 = c(134, NA, 35, 134, 5, 24),
      Q2 = c(
        "Alm Dudler, Essig, Cola",  NA,
        "Cola, Holer", "Alm Dudler, Cola,
        Essig","Holer", "Bier, Essig"
      )
    )
    dat
#>    Q1                               Q2
#> 1 134          Alm Dudler, Essig, Cola
#> 2  NA                             <NA>
#> 3  35                      Cola, Holer
#> 4 134 Alm Dudler, Cola, \n        Essig
#> 5   5                            Holer
#> 6  24                      Bier, Essig
```

```
    cbind(dat[-1],
          separate_multiple_choice(dat$Q2))
#> Warning: Expected 7 pieces. Missing pieces filled with `NA` in 6 rows [1, 2, 3,
#> 4, 5, 6].
#>                                  Q2 Q2_1 Q2_2 Q2_3 Q2_4 Q2_5 Q2_6
#> 1          Alm Dudler, Essig, Cola nein   ja nein   ja   ja nein
#> 2                              <NA> <NA> <NA> <NA> <NA> <NA> <NA>
#> 3                       Cola, Holer nein nein nein   ja nein   ja
#> 4 Alm Dudler, Cola, \n        Essig   ja   ja nein   ja nein nein
#> 5                             Holer nein nein nein nein nein   ja
#> 6                       Bier, Essig nein nein   ja nein   ja nein
```

```
    dat <- cbind(dat[-2],
                 separate_multiple_choice(dat$Q1,
                                          label = c(
                                            "Alm Dudler", "Bier", "Cola", "Essig", "Holer"
                                          )))
#> Warning: Expected 6 pieces. Missing pieces filled with `NA` in 6 rows [1, 2, 3,
#> 4, 5, 6].

dat
#>    Q1 Q1_1 Q1_2 Q1_3 Q1_4 Q1_5
#> 1 134   ja nein   ja   ja nein
#> 2  NA <NA> <NA> <NA> <NA> <NA>
#> 3  35 nein nein   ja nein   ja
#> 4 134   ja nein   ja   ja nein
#> 5   5 nein nein nein nein   ja
#> 6  24 nein   ja nein   ja nein
```

### Mittelwerte Addieren

```
#' Mittelwerte Addieren
#'
#'
#' @param n
#' @param m
#' @param sd
#'
calc.mean <- function(n, m, sd) {
  j <- length(n) #-- Anzahl an Elementen
  if (length(n) != length(m) |
      length(n) != length(sd))
    stop("Ungleiche Anzahl an n,m oder sd!")

  #print(paste( m ,"-", trunc(m),"=", m - trunc(m)))
  #-- Interne Function berechnet zwei Werte
  calc.mean2 <- function(n, m, sd) {
    var <- sd ^ 2
    n.total <- sum(n)
    n.minus1 <- n - 1
    m.total <- sum(m * n) / n.total
    var.total <-
      1 / (n.total - 1) * (sum(n.minus1 * var)  +   prod(n) / n.total * diff(m) ^
                             2)
    #var3 =  1/(n1+n2-1) *( (n1-1)*var1  +  (n2-1)*var2 + n1*n2/(n1+n2)*((m1-m2)^2)  )
    data.frame(
      n = c(n, n.total),
      m = c(m, m.total),
      sd = c(sd, sqrt(var.total)),
      var = c(var, var.total)
    )
  }

  if (j == 2) {
    zw <- calc.mean2(n, m, sd)
  }
  else{
    n1 <- n[1:2]
    m1 <- m[1:2]
    sd1 <- sd[1:2]
    zw <- calc.mean2(n = n1, m = m1, sd = sd1)
    zw.j <- zw[3, ]
    for (i in 3:j) {
      zw.i <- calc.mean2(
        n  = c(zw.j$n, n[i])
        ,
    m  = c(zw.j$m, m[i])
    ,
    sd = c(zw.j$sd , sd[i])
   )
   zw[i, ] <- zw.i[2, ]
   zw <- rbind(zw, zw.i[3, ])
  }
  zw
 }
 cbind(value = c(1:j, "total") , zw)
}



x1 <- rnorm( 4, 3.0,  2.1)
x2 <- rnorm(10, 4.0,  2.0)
x3 <- rnorm(11, 3.5,  3.0)

calc.mean(
  n = c(length(x1), length(x2)),
  m = c(mean(x1), mean(x2)),
  sd = c(sd(x1), sd(x2))
)
#>   value  n        m       sd      var
#> 1     1  4 3.564975 1.404418 1.972391
#> 2     2 10 3.334813 1.610394 2.593369
#> 3 total 14 3.400573 1.504068 2.262219
mean(c(x1, x2))
#> [1] 3.400573
sd(c(x1, x2))
#> [1] 1.504068
var(c(x1, x2))
#> [1] 2.262219


calc.mean(
  n = c(length(x1), length(x2), length(x3)),
  m = c(mean(x1), mean(x2), mean(x3)),
  sd = c(sd(x1), sd(x2) , sd(x3))
)
#>    value  n        m       sd       var
#> 1      1  4 3.564975 1.404418  1.972391
#> 2      2 10 3.334813 1.610394  2.593369
#> 3      3 11 4.878103 3.286384 10.800317
#> 31 total 25 4.050686 2.507155  6.285829
mean(c(x1, x2, x3))
#> [1] 4.050686
sd(c(x1, x2, x3))
#> [1] 2.507155
var(c(x1, x2, x3))
#> [1] 6.285829
```

```r
# Total Asf
calc.mean(
  n = c(22, 26, 22, 26),
  m = c(407, 613, 434, 565) / 100,
  sd = c(258, 236, 345, 253) / 100
)
#>    value  n        m       sd      var
#> 1      1 22 4.070000 2.580000  6.65640
#> 2      2 26 6.130000 2.360000  5.56960
#> 3      3 22 4.340000 3.450000 11.90250
#> 31     4 26 5.650000 2.530000  6.40090
#> 32 total 74 5.348919 2.599394  6.75685
```

## Funktionen aus anderen Packeten

### dplyr::case_when

```r
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

set.seed(0815)
n<-15
df<- data.frame(x=rnorm(n), y=rnorm(n), z=rnorm(n))
df <- df %>%
  mutate(group = case_when(x<0 & y<0 & z<0 ~ "A",
                           x<0 | y<0 | z<0 ~ "B",
                           TRUE ~ "C"
                           ))

df
#>             x           y           z group
#> 1  -1.4252474  1.33147168  0.65778658     B
#> 2  -1.2231010 -0.73598141  0.30443639     B
#> 3  -0.1202374  0.96772489 -1.24252785     B
#> 4   1.4440589  0.16358929  0.05297387     C
#> 5  -0.9666228 -1.24964628 -0.35817203     A
#> 6  -1.1378613 -0.22007043  0.04906483     B
#> 7   1.1018533 -0.03454882  0.79278829     B
#> 8   1.0038359  0.51085258  0.65341253     C
#> 9   0.7926149 -1.35826538  0.33830078     B
#> 10 -1.4113182 -0.39961979 -0.12137287     A
#> 11  0.3699992 -0.33334412  0.70076765     B
#> 12 -0.1885835 -2.80254097 -1.39342434     A
#> 13 -0.4797290  1.45521302  0.65376034     B
#> 14 -0.8474767  0.29402771  1.76151260     B
#> 15  0.3923676 -2.12929920  0.49804261     B
```

## Berechnungen mit tydy