Daten verarbeiten und vorbereiten

Dirk Seidensticker/Clemens Schmid

7. Juli 2017

Daten verknüpfen (%in%, merge)

Kategorisierung und Rangfolgen (factor & levels)

Duplikate entfernen (unique & duplicated)

Sortieren (sort & order)

Gruppieren und Gruppenoperationen (aggregate & group_by & summarise)

Pivotieren (*apply & dcast)

Datenanordnung - wide to long und long to wide (melt)

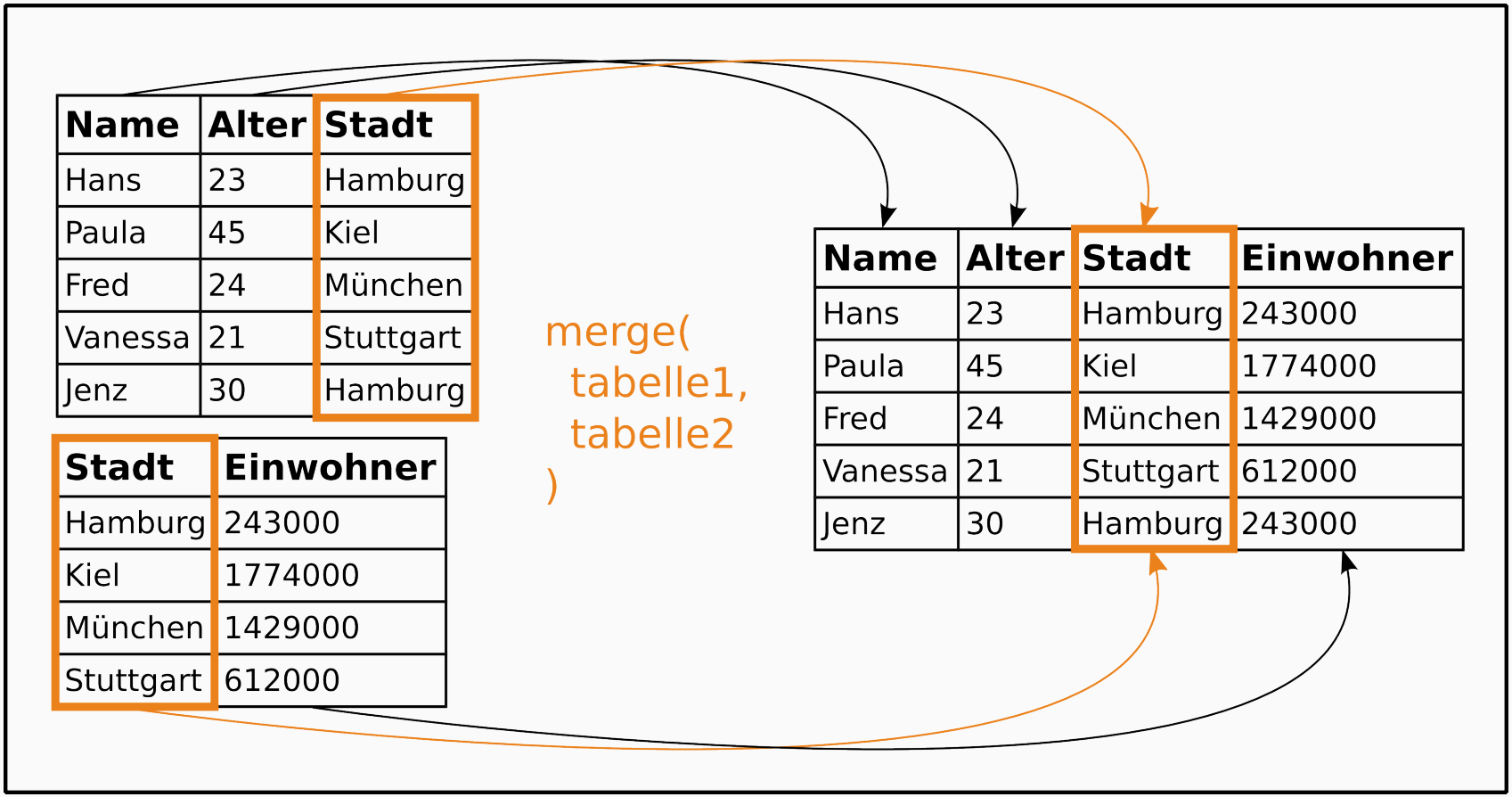
Filtern (subset & filter)

SQL-artige Bearbeitungen und Abfragen

Daten verknüpfen (%in%, merge)

Daten verknüpfen - Idee

Zusammenführen von Informationen aus verschiedenen Tabellen.



Vergleiche **Join-Operationen** in SQL (NATURAL JOIN, INNER JOIN, LEFT JOIN, RIGHT JOIN).

Am Anfang steht oft die Frage, ob gemeinsame Merkmalsausprägungen vorhanden sind: **%in%**

base::merge()

*merge(**x**, **y**, **by** = intersect(names(x), names(y)), **by.x** = by, **by.y** = by, **all** = FALSE, **all.x** = all, **all.y** = all, sort = TRUE, suffixes = c(".x",".y"), incomparables = NULL, . . .)*

Beschreibung

Merge two data frames by common columns or row names, or do other versions of database join operations.

Beispiel

```
R <- data.frame(V1 = c(1,2,3), V2 = c(4,5,6), V3 = c("A","B","C"))
S <- data.frame(V4 = c(7,8,9), V5 = c(10,11,12), V6 = c("A","C","D"))

merge(x = R, y = S, by.x = "V3", by.y = "V6", all.x = TRUE)
```

```
##   V3 V1 V2 V4 V5
## 1  A  1  4  7 10
## 2  B  2  5 NA NA
## 3  C  3  6  8 11
```

base::match()

*match(**x**, **table**, **nomatch** = NA_integer_, incomparables = NULL)*

*x **%in%** table*

Beschreibung

match returns a vector of the positions of (first) matches of its first argument in its second.

%in% is a more intuitive interface as a binary operator, which returns a logical vector indicating if there is a match or not for its left operand.

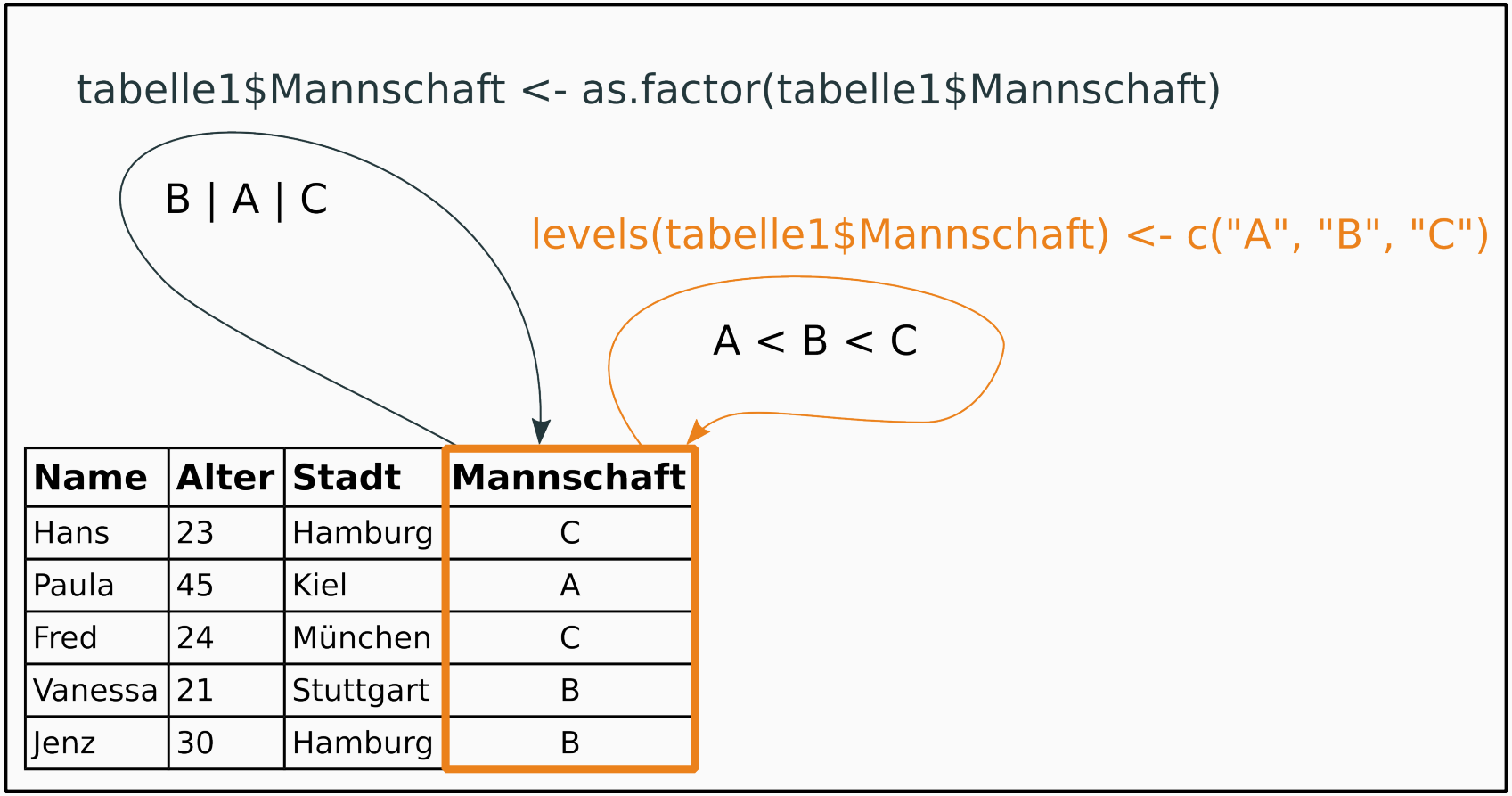
Beispiel

```
R <- data.frame(V1 = c(1,2,3), V2 = c(4,5,6), V3 = c("A","B","C"))
S <- data.frame(V4 = c(7,8,9), V5 = c(10,11,12), V6 = c("A","C","D"))

R$V3 %in% S$V6

## [1]  TRUE FALSE  TRUE
```

Kategorisierung und Rangfolgen (factor & levels)

Kategorisierung und Rangfolgen - Idee

Werte zu Klassen zusammen fassen und Klassen in eine Rangfolgen bringen



Name	Alter	Stadt	Mannschaft
Hans	23	Hamburg	C
Paula	45	Kiel	A
Fred	24	München	C
Vanessa	21	Stuttgart	B
Jenz	30	Hamburg	B

Klassifizierung und die Qualität von Klassen stecken nicht direkt in Daten - sie müssen klar zugewiesen werden.
Insbesondere für Plots von Bedeutung.

base::factor()

*factor(**x** = character(), **levels**, **labels** = levels, exclude = NA, **ordered** = is.ordered(x), nmax = NA)*

Beschreibung

The function factor is used to encode a vector as a factor (the terms 'category' and 'enumerated type' are also used for factors). If argument ordered is TRUE, the factor levels are assumed to be ordered.

Beispiel

```
R <- data.frame(V1 = c(1.1,1.2,1.3), V2 = c(1,2,3))
R$V1 <- as.factor(x = R$V1)
R$V1
```

```
## [1] 1.1 1.2 1.3
## Levels: 1.1 1.2 1.3
```

base::levels()

*levels(**x**)*

*levels(**x**) <- value*

Beschreibung

levels provides access to the levels attribute of a variable. The first form returns the value of the levels of its argument and the second sets the attribute.

Beispiel

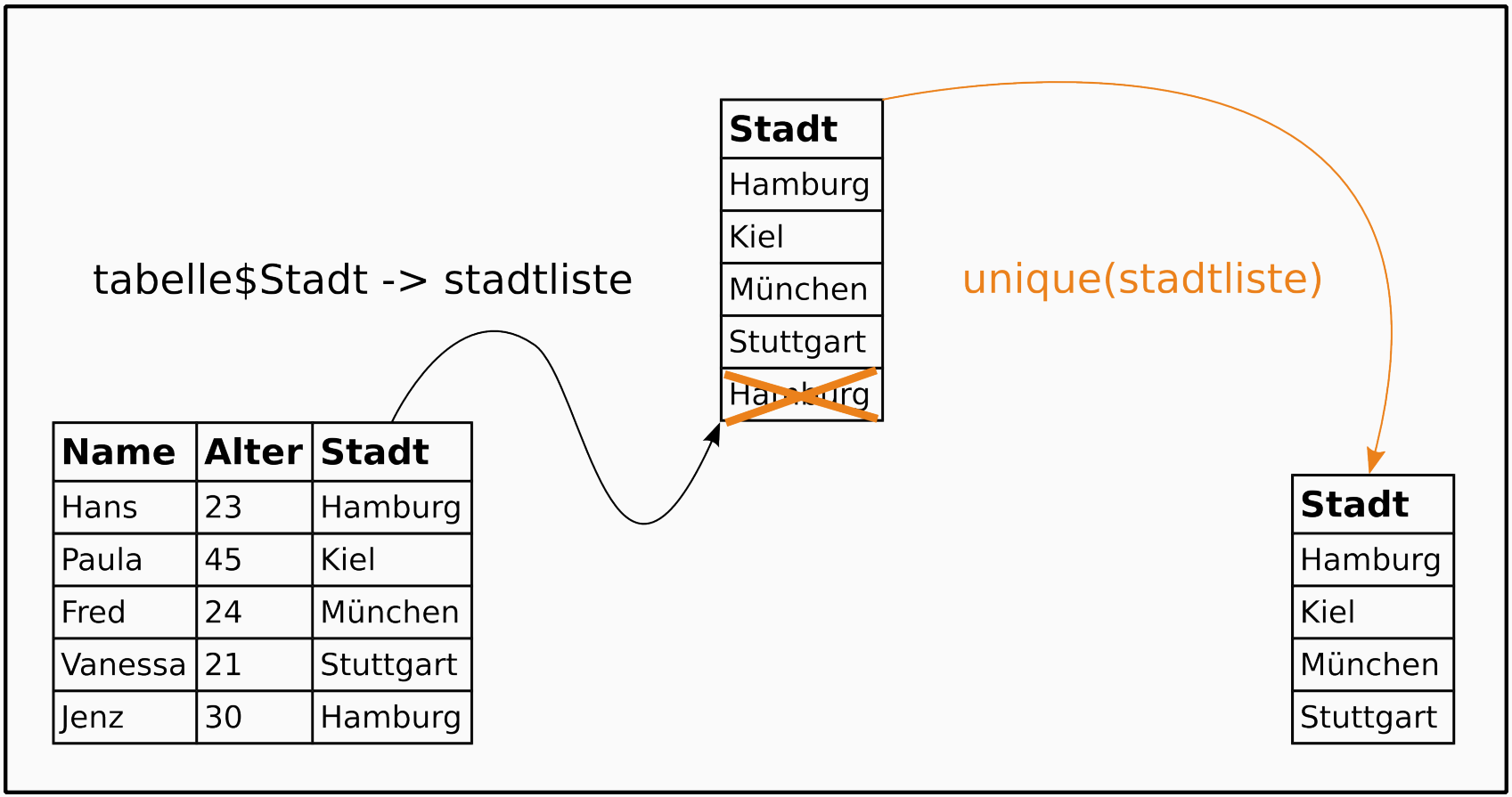
```
R <- data.frame(V1 = c(1.1,1.2,1.3), V2 = c(1,2,3))
levels(x = R$V1) <- c(1.3,1.2,1.1)
R$V1
```

```
## [1] 1.1 1.2 1.3
## attr(,"levels")
## [1] 1.3 1.2 1.1
```

Duplikate entfernen (unique & duplicated)

Duplikate entfernen - Idee

Finden und Entfernen doppelter Werte und Wertekombinationen



Duplikate entfernen und Bandbreite einer Merkmalsausprägung erfassen: “Was gibt es überhaupt?”

base::unique()

*unique(**x**, incomparables = FALSE, **fromLast** = FALSE, nmax = NA, ...)*

Beschreibung

unique returns a vector, data frame or array like x but with duplicate elements/rows removed.

Beispiel

```
R <- data.frame(V1 = c(1,1,1), V2 = c(2,2,2), V3 = c("A","A","B"))
unique(x = R)
```

```
##   V1 V2 V3
## 1  1  2  A
## 3  1  2  B
```

base::duplicated()

*duplicated(**x**, incomparables = FALSE, **fromLast** = FALSE, nmax = NA, . . .)*

Beschreibung

duplicated() determines which elements of a vector or data frame are duplicates of elements with smaller subscripts, and returns a logical vector indicating which elements (rows) are duplicates.

Beispiel

```
R <- data.frame(V1 = c(1,1,1), V2 = c(2,2,2), V3 = c("A","A","B"))
duplicated(x = R, fromLast = TRUE)
```

```
## [1]  TRUE FALSE FALSE
```

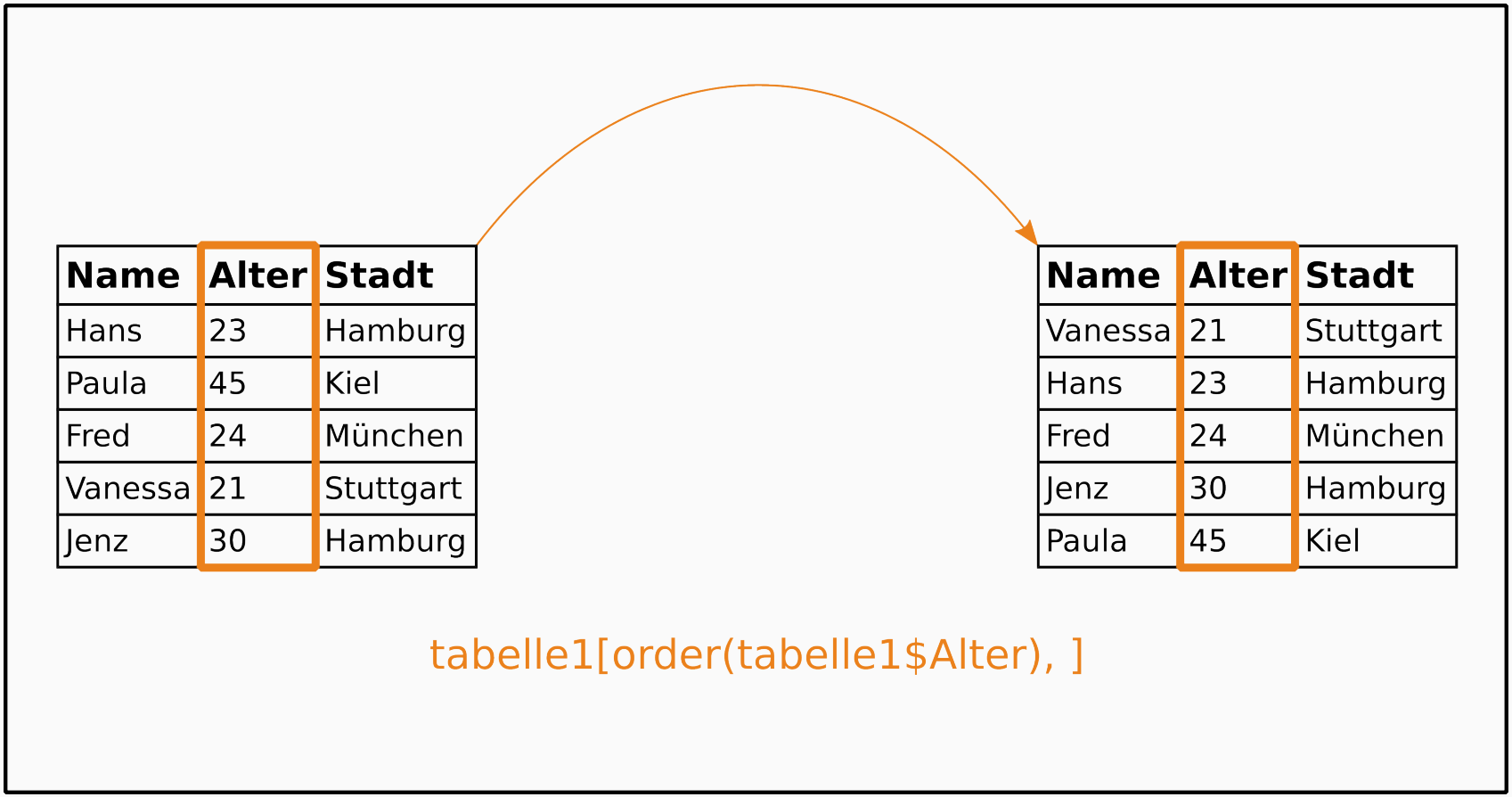
```
any(duplicated(R))
```

```
## [1] TRUE
```

Sortieren (sort & order)

Sortieren - Idee

Werte und Wertekombinationen in eine Reihenfolge bringen



Sortieren läuft für verschiedene Datentypen unterschiedlich ab: Vgl. Zahlwerte, Worte, BLOBs, etc.

base::sort()

*sort(**x**, **decreasing** = FALSE, na.last = NA, . . .)*

Beschreibung

Sort a vector or factor into ascending or descending order.

Beispiel

```
V <- c("A","F","G","Käsebrot","B","X","Q")
sort(x = V, decreasing = TRUE)
```

```
## [1] "X"         "Q"         "Käsebrot" "G"         "F"         "B"
## [7] "A"
```

base::order()

order(..., *na.last* = *TRUE*, ***decreasing*** = *FALSE)*

Beschreibung

order returns a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments.

Beispiel

```
R <- data.frame(V1 = c(1,1,1), V2 = c(132,78,5), V3 = c("B","A","A"))
order(R$V3, R$V2)
```

```
## [1] 3 2 1
```

```
R[order(R$V3, R$V2), ]
```

```
##   V1  V2 V3
## 3  1   5  A
## 2  1  78  A
## 1  1 132  B
```

Gruppieren und Gruppenoperationen (aggregate & group_by & summarise)

Gruppieren und Gruppenoperationen - Idee

Gruppen bilden und gruppenbezogene Fragen stellen

Name	Alter	Stadt	Mannschaft
Hans	23	Hamburg	C
Paula	45	Kiel	A
Fred	24	München	C
Vanessa	21	Stuttgart	B
Jenz	30	Hamburg	B

Mannschaft	mean(Alter)
A	45
B	25.5
C	23.5

summarise(group_by(tabelle1, Mannschaft), mean(Alter))

Mannschaft	Name	Alter	Stadt
A	Paula	45	Kiel
B	Vanessa	21	Stuttgart
B	Jenz	30	Hamburg
C	Hans	23	Hamburg
C	Fred	24	München

Gruppen werden immer in Abhängigkeit von Merkmalsausprägungen definiert.

Gruppenbezogene Fragen lassen nur gruppenbezogene Antworten zu - individuelle Werte gehen verloren.

stats::aggregate()

*aggregate(**x**, **by**, **FUN**, . . . , simplify = TRUE)*

*aggregate(**formula**, **data**, **FUN**, . . . , subset, na.action = na.omit)*

Beschreibung

Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

Beispiel

```
R <- data.frame(V1 = c(3,5,1), V2 = c(3,8,5), V3 = c("B","A","A"))
aggregate(x = R, by = list(R$V3), FUN = "mean") -> a1
aggregate(V2~V3, data = R, FUN = "sum")
```

```
##   V3 V2
## 1  A 13
## 2  B  3
```

dplyr::group_by()

*group_by(**.data**, **...**, add = FALSE)*

Beschreibung

The group_by function takes an existing tbl and converts it into a grouped tbl where operations are performed "by group".

Beispiel

```
R <- data.frame(V1 = c(3,5,1), V2 = c(3,8,5), V3 = c("B","A","A"))
group_by(R, V3)
```

```
## Source: local data frame [3 x 3]
## Groups: V3 [2]
##
##      V1    V2     V3
##   <dbl> <dbl> <fctr>
## 1     3     3      B
## 2     5     8      A
## 3     1     5      A
```

dplyr::summarise()

*summarise(**.data**, **...**)*

Beschreibung

Summarise multiple values to a single value.

Beispiel

```
R <- data.frame(V1 = c(4,5,1), V2 = c(3,8,5), V3 = c("B","A","A"))
R.g <- group_by(R, V3)
summarise(R.g, Mittwelwert = mean(V1), Summe = sum(V2))
```

```
## # A tibble: 2 × 3
##       V3 Mittwelwert Summe
##   <fctr>       <dbl> <dbl>
## 1      A           3    13
## 2      B           4     3
```

Pivotieren (*apply & dcast)

Kreuztabelle mit Auszählen

```r
df <- read.csv("../data/2-3_data.csv", encoding = 'UTF-8')

table(df$Stadt, df$Mannschaft)
```

```
##
##               A B C
##   Hamburg     0 2 1
##   Kiel        3 0 0
##   München     0 0 2
##   Stuttgart 0 2 0
```

Kreuztabelle mit Auszählen

dito mit tapply

```
a <- tapply(df$Name,
            list(df$Stadt,
                 df$Mannschaft),
            length)
a[is.na(a)] <- 0
a
```

```
##           A B C
## Hamburg   0 2 1
## Kiel      3 0 0
## München   0 0 2
## Stuttgart 0 2 0
```

Berechnungen in Abhängigkeit zu einer Variable

```
tapply(df$Alter, list(df$Mannschaft), mean)
```

```
##        A        B        C
## 43.66667 24.75000 30.66667
```

Berechnungen in Abhängigkeit zu einer Variable

```r
b <- data.frame(tapply(df$Alter, list(df$Mannschaft), mean))
colnames(b) <- c("Mittleres Alter")
b
```

```
##   Mittleres Alter
## A        43.66667
## B        24.75000
## C        30.66667
```

Berechnungen in Abhängigkeit zu einer Variable

```
b <- data.frame(tapply(df$Alter, list(df$Mannschaft),
                       function(x){sum(x)/length(df$Name)}))
colnames(b) <- c("berechneter Wert")
b
```

```
##   berechneter Wert
## A             13.1
## B              9.9
## C              9.2
```

Berechnungen in Abhängigkeit zu zwei Variablen

```
b <- data.frame(tapply(df$Alter,
                list(df$Stadt, df$Mannschaft), mean),
                check.names = FALSE)
b[is.na(b)] <- 0
b
```

```
##                   A    B    C
## Hamburg     0.00000 26.5 23.0
## Kiel       43.66667  0.0  0.0
## München     0.00000  0.0 34.5
## Stuttgart   0.00000 23.0  0.0
```

Auszählen in Abhängigkeit zu drei Variablen

```
library(reshape2)

c <- dcast(df, Mannschaft + Stadt ~ Geschlecht)


## Using Mannschaft as value column: use value.var to override.

## Aggregation function missing: defaulting to length

head(c)
```

```
##   Mannschaft     Stadt f m
## 1          A      Kiel 2 1
## 2          B   Hamburg 0 2
## 3          B Stuttgart 1 1
## 4          C   Hamburg 0 1
## 5          C   München 1 1
```

Berechnungen in Abhängigkeit zu drei Variablen

```
c <- dcast(df, Mannschaft + Stadt ~ Geschlecht,
      value.var = "Alter",
      fun.aggregate = mean)

head(c)
```

```
##   Mannschaft     Stadt    f    m
## 1          A      Kiel 54.5 22.0
## 2          B   Hamburg  NaN 26.5
## 3          B Stuttgart 21.0 25.0
## 4          C   Hamburg  NaN 23.0
## 5          C   München 45.0 24.0
```

Datenanordnung - wide to long und long to wide (melt)

Melt

Eingabeformat für ggplot2()!!!

Wir wollen diese vorhin erstellte Tabelle plotten:

```
a
```

```
##           A B C
## Hamburg   0 2 1
## Kiel      3 0 0
## München   0 0 2
## Stuttgart 0 2 0
```

Melt

```
library(reshape2)
a.long <- melt(a)

a.long
```

```
##          Var1 Var2 value
## 1     Hamburg    A     0
## 2        Kiel    A     3
## 3     München    A     0
## 4  Stuttgart    A     0
## 5     Hamburg    B     2
## 6        Kiel    B     0
## 7     München    B     0
## 8  Stuttgart    B     2
## 9     Hamburg    C     1
## 10       Kiel    C     0
## 11    München    C     2
## 12 Stuttgart    C     0
```

Melt

```r
colnames(a.long) <- c('Stadt', 'Mannschaft', 'Anzahl')

head(a.long)
```

```
##       Stadt Mannschaft Anzahl
## 1   Hamburg          A      0
## 2      Kiel          A      3
## 3   München          A      0
## 4 Stuttgart          A      0
## 5   Hamburg          B      2
## 6      Kiel          B      0
```

Filtern (subset & filter)

Filter und Subseting nach Mannschaft 'A'

```r
df.sub <- subset(df, Mannschaft == 'A')
head(df.sub)
```

```
##         Name Alter Geschlecht Stadt Mannschaft
## 2      Paula    45          f  Kiel          A
## 6     Robert    22          m  Kiel          A
## 8 Christina    64          f  Kiel          A
```

Filter mit mehrere Bedingungen

```
library(dplyr)
```

```
df.filt <- filter(df, Mannschaft == 'A' & Alter < 60)
head(df.filt)
```

```
##     Name Alter Geschlecht Stadt Mannschaft
## 1  Paula    45          f  Kiel          A
## 2 Robert    22          m  Kiel          A
```

SQL-artige Bearbeitungen und Abfragen

sqldf()

```
sqldf('SELECT Stadt,
              Mannschaft
      FROM df
      WHERE Geschlecht IS "m"
      GROUP BY Stadt,
               Mannschaft')
```

```
##       Stadt Mannschaft
## 1   Hamburg          B
## 2   Hamburg          C
## 3      Kiel          A
## 4   München          C
## 5 Stuttgart          B
```