Daten verarbeiten und vorbereiten

Dirk Seidensticker/Clemens Schmid

6. Februar 2016

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)

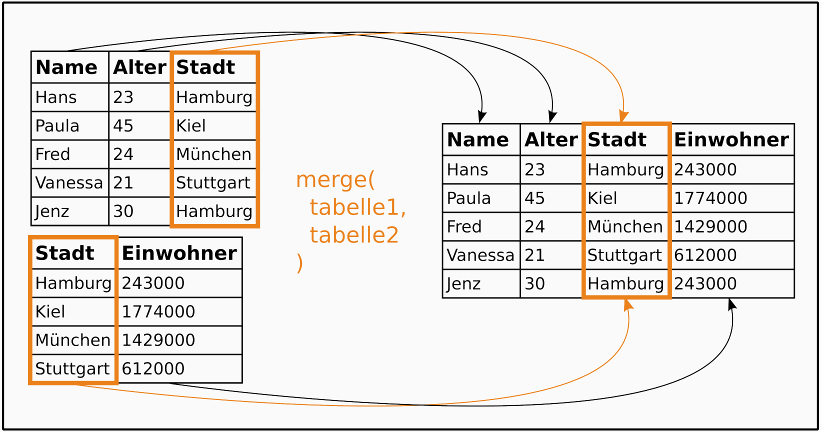
Datenanordnung - wide to long und long to wide (melt)

Filtern (subset & filter)

Daten verknüpfen (%in%, merge)

Daten verknüpfen - Idee

Zusammenführen von Informationen aus verschiedenen Tabellen.



Name	**Alter**	**Stadt**
Hans	23	Hamburg
Paula	45	Kiel
Fred	24	München
Vanessa	21	Stuttgart
Jenz	30	Hamburg

Stadt	**Einwohner**
Hamburg	243000
Kiel	1774000
München	1429000
Stuttgart	612000

```
merge(
  tabelle1,
  tabelle2
)
```

Name	**Alter**	**Stadt**	**Einwohner**
Hans	23	Hamburg	243000
Paula	45	Kiel	1774000
Fred	24	München	1429000
Vanessa	21	Stuttgart	612000
Jenz	30	Hamburg	243000

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
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

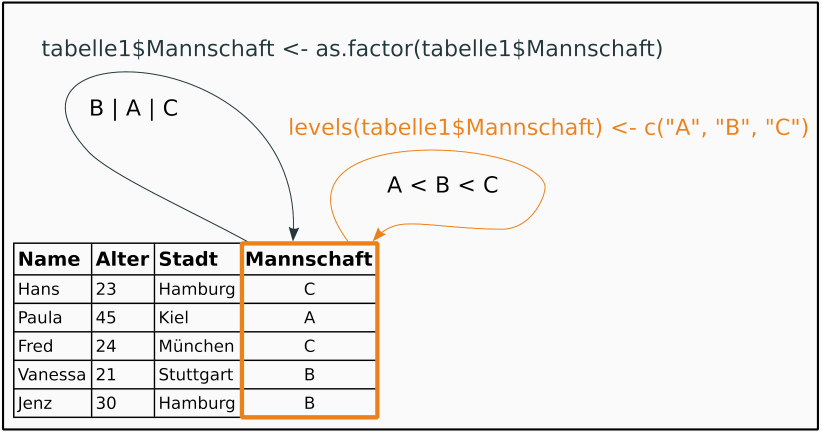
```r
R <- data.frame(V1 = c(1,2,3), V2 = c(4,5,6), V3 = c("A","B","C"))
S <- data.frame(V4 = c(7,8,9), V5 = c(10,11,12), V6 = c("A","C","D"))

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



```
tabelle1$Mannschaft <- as.factor(tabelle1$Mannschaft)
```

```
levels(tabelle1$Mannschaft) <- c("A", "B", "C")
```

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
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

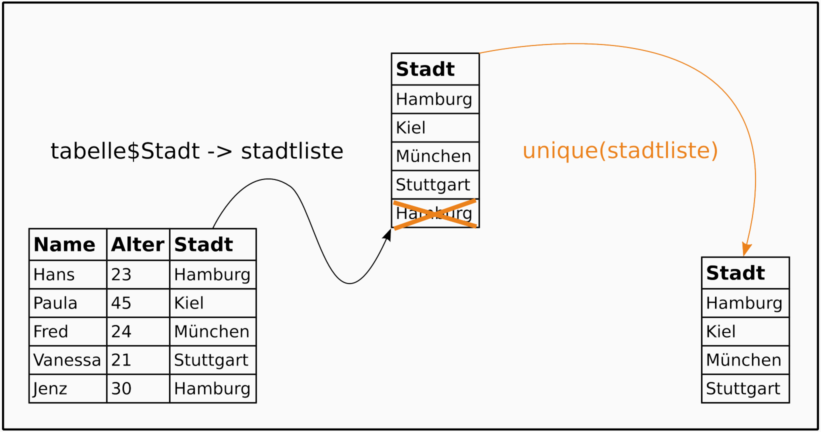
```
R <- data.frame(V1 = c(1.1,1.2,1.3), V2 = c(1,2,3))
levels(x = R$V1) <- c(1.3,1.2,1.1)
R$V1
```

```
## [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



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

tabelle$Stadt -> stadtliste

Stadt
Hamburg
Kiel
München
Stuttgart
~~Hamburg~~

unique(stadtliste)

Stadt
Hamburg
Kiel
München
Stuttgart

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
```

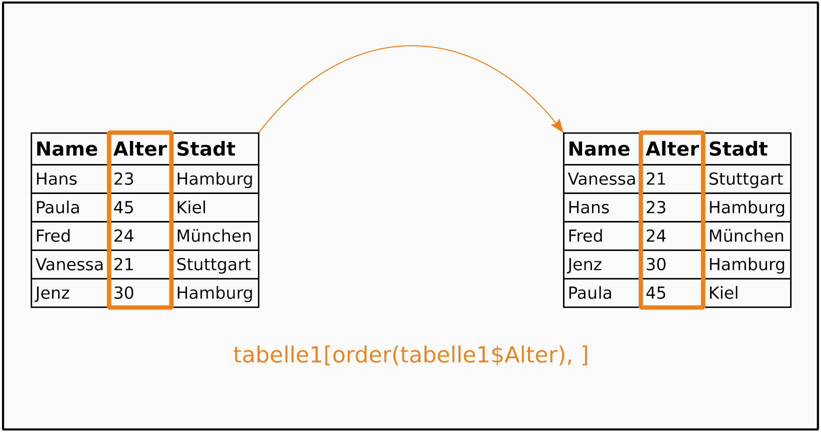
```
any(duplicated(R))
```

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

Sortieren (sort & order)

Sortieren - Idee

Werte und Wertekombinationen in eine Reihenfolge bringen



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

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

```
tabelle1[order(tabelle1$Alter), ]
```

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

```r
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
```

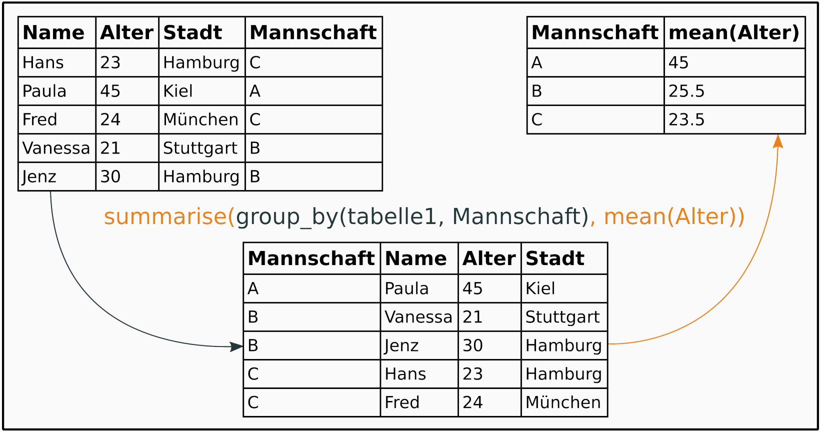
```
R[order(R$V3, R$V2), ]
```

```
##   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
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))
```

```
## Source: local data frame [2 x 3]
##
##       V3 Mittwelwert Summe
##   (fctr)       (dbl) (dbl)
## 1      A           3    13
## 2      B           4     3
```

Pivotieren (*apply & dcast)

Pivotieren

```r
atlant <- read.csv(
  "../data/AtlantData1.csv",
  sep = "\t",
  header = TRUE
  )
```

Kreuztabelle mit Auszählen

```r
table(atlant$site, atlant$size)
```

```
##
##      30 70 120 200 500
##   A  0  0   1   2   5
##   B  0  0   0   2   2
##   C  0  1   2   0   1
##   D  0  8   7   1   0
##   E  0  0   0   2   2
##   F  1 12  20  23   9
##   G  0  0   1   2   2
##   H  0 15   8   5   0
```

Kreuztabelle mit Auszählen

dito mit tapply

```
a <- tapply(atlant$qty, list(atlant$site, atlant$size), length)
a[is.na(a)] <- 0
a
```

```
##   30 70 120 200 500
## A  0  0   1   2   5
## B  0  0   0   2   2
## C  0  1   2   0   1
## D  0  8   7   1   0
## E  0  0   0   2   2
## F  1 12  20  23   9
## G  0  0   1   2   2
## H  0 15   8   5   0
```

Berechnungen in Abhängigkeit zu einer Variable

```r
tapply(atlant$wt, list(atlant$site), sum)
```

```
##     A     B     C     D     E     F     G     H
##  8805  6621  1641  1760  4825 22849  3525  4530
```

Berechnungen in Abhängigkeit zu einer Variable

```r
data.frame(tapply(atlant$wt, list(atlant$site), sum))
```

```
##   tapply.atlant.wt..list.atlant.site...sum.
## A                                      8805
## B                                      6621
## C                                      1641
## D                                      1760
## E                                      4825
## F                                     22849
## G                                      3525
## H                                      4530
```

Berechnungen in Abhängigkeit zu einer Variable

```r
data.frame(tapply(atlant$wt, list(atlant$site),
                  function(x){sum(x)/1000}))
```

```
##   tapply.atlant.wt..list.atlant.site...function.x...
## A                                              8.805
## B                                              6.621
## C                                              1.641
## D                                              1.760
## E                                              4.825
## F                                             22.849
## G                                              3.525
## H                                              4.530
```

Berechnungen in Abhängigkeit zu zwei Variablen

```
b <- data.frame(tapply(atlant$wt,
                list(atlant$site, atlant$size), sum),
                check.names = FALSE)
b[is.na(b)] <- 0
b
```

```
##   30  70  120  200   500
## A  0   0   67 1536  7202
## B  0   0    0 1003  5618
## C  0  45  148    0  1448
## D  0 272 1150  338     0
## E  0   0    0 1481  3344
## F  6 453 2015 9812 10563
## G  0   0   84 1002  2439
## H  0 512 1007 2921     0
```

Auszählen in Abhängigkeit zu drei Variablen

```r
library(reshape2)

pivot <- dcast(atlant, site + feature ~ size)
```

```
## Using vesselShape as value column: use value.var to override.

## Aggregation function missing: defaulting to length
```

```r
head(pivot)
```

```
##   site feature 30 70 120 200 500 NA
## 1    A       1  0  0   1   1   4  0
## 2    A surface  0  0   0   1   1  0
## 3    B surface  0  0   0   2   2  0
## 4    C surface  0  1   2   0   1  0
## 5    D       1  0  6   4   1   0  0
## 6    D       2  0  1   1   0   0  0
```

Berechnungen in Abhängigkeit zu drei Variablen

```r
pivot2 <- dcast(atlant, site + feature ~ size,
      value.var = "wt",
      fun.aggregate = sum)

head(pivot2)
```

```
##   site feature 30  70 120  200  500 NA
## 1    A       1  0   0  67 1012 5630  0
## 2    A surface  0   0   0  524 1572  0
## 3    B surface  0   0   0 1003 5618  0
## 4    C surface  0  45 148    0 1448  0
## 5    D       1  0 159 749  338    0  0
## 6    D       2  0  74 167    0    0  0
```

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

Melt

Eingabeformat für ggplot2()!!!

Wir wollen diese vorhin erstelle Tabelle plotten:

```
a
```

```
##    30 70 120 200 500
## A   0  0   1   2   5
## B   0  0   0   2   2
## C   0  1   2   0   1
## D   0  8   7   1   0
## E   0  0   0   2   2
## F   1 12  20  23   9
## G   0  0   1   2   2
## H   0 15   8   5   0
```

Melt

```r
library(reshape2)
atlant_sites.size <- melt(a)

head(atlant_sites.size)
```

```
##   Var1 Var2 value
## 1    A   30     0
## 2    B   30     0
## 3    C   30     0
## 4    D   30     0
## 5    E   30     0
## 6    F   30     1
```

Melt

```r
colnames(atlant_sites.size) <- c('site', 'size', 'qty')

head(atlant_sites.size)
```

```
##   site size qty
## 1    A   30   0
## 2    B   30   0
## 3    C   30   0
## 4    D   30   0
## 5    E   30   0
## 6    F   30   1
```

Filtern (subset & filter)

Filter und Subseting nach site ‘F’

```r
atlant_sub <- subset(atlant, site == 'F')
head(atlant_sub)
```

```
##    site      X       Y    archont feature                 object cla
## 37    F 458694 3951533 Autochthon       2                   -8:1
## 38    F 458694 3951533 Autochthon       1                   -3:1
## 39    F 458694 3951533 Autochthon       1                   -6:2
## 40    F 458694 3951533 Autochthon       2     -2:2 -4:3,10-11,13
## 41    F 458694 3951533 Autochthon       2                   -4:2
## 42    F 458694 3951533 Autochthon       2 -4:5-6,8,12,14-21 -5:3
##    sherd qty  wt size wall muendungsD muendungsH minD minD_H maxD ma
## 37     G   1 829  500    7       20.5       12.5 17.5    8.5 19.0
## 38     G   1 155  120    4       11.5        6.5 12.0    5.0  5.0
## 39     G   1 504  200    4        5.0       18.5  6.5   14.0 16.5
## 40     G   1 419  200   10       19.0       14.0 17.0   10.0 17.5
## 41     G   1 691  500   10       20.0       12.0 17.5    9.0 18.0
## 42     G   1 623  500   10       18.5       13.0 17.0   11.0 18.0
##    bodenD temperSize vesselShape
## 37    0.0         VF         S1b
```

Filter mit mehrere Bedingungen

```r
library(dplyr)
```

```r
atlant_filter <- filter(atlant, site == 'F' & wall > 5)
head(atlant_filter)
```

```
##   site      X       Y    archont feature                    object clas
## 1    F 458694 3951533 Autochthon       2                      -8:1
## 2    F 458694 3951533 Autochthon       2     -2:2 -4:3,10-11,13
## 3    F 458694 3951533 Autochthon       2                      -4:2
## 4    F 458694 3951533 Autochthon       2 -4:5-6,8,12,14-21 -5:3
## 5    F 458694 3951533 Autochthon       2    -4:4,7,9 -5:4-6,8-9
## 6    F 458694 3951533 Autochthon       2                      -5:2
##   sherd qty   wt size wall muendungsD muendungsH minD minD_H maxD ma
## 1     G   1  829  500    7       20.5       12.5 17.5    8.5 19.0
## 2     G   1  419  200   10       19.0       14.0 17.0   10.0 17.5
## 3     G   1  691  500   10       20.0       12.0 17.5    9.0 18.0
## 4     G   1  623  500   10       18.5       13.0 17.0   11.0 18.0
## 5     G   1  344  200   10         NA         NA 18.0    8.5 17.5
## 6     G   1 1671  500   12       24.5       15.0 21.5   10.5 22.5
```