

# Was uns die Daten sagen - erste Schritte der Daten Bearbeitung und Analyse

Jan-Philipp Kolb

06 Juni, 2019

# Inhalt dieses Abschnitts

- Wie bekommt man einen Überblick über die Daten
- Indizieren von Vektoren, Datensätzen und Listen
- Wie geht man mit fehlenden Werten um
- Schleifen und Funktionen
- Zusammenhänge zwischen Variablen

# data.frame's

- Beispieldaten importieren:

```
dat <- read.csv2("../data/wahldat_ffm.csv")
```

```
head(names(dat))
```

```
## [1] "X"                                "Stadtteilnummer"  
## [3] "Stadtteilname"                  "Wahlberechtigte.ob"  
## [5] "Wahlberechtigte.mit.Sperrvermerk" "Wahlberechtigte.in"
```

# Anzahl Zeilen und Spalten

- Anzahl der Zeilen/Spalten ermitteln

```
nrow(dat) # Zeilen
```

```
## [1] 45
```

```
ncol(dat) # Spalten
```

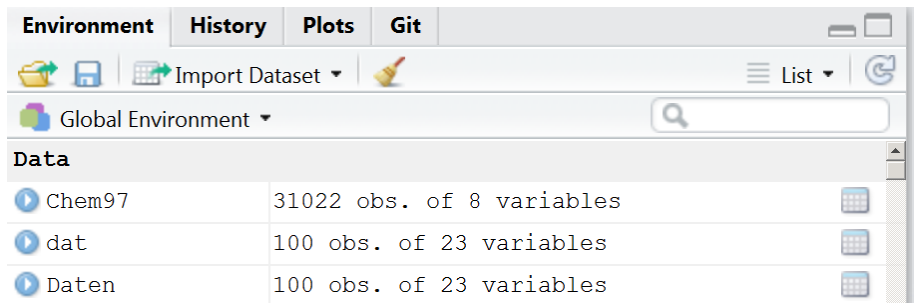
```
## [1] 58
```

# Die Daten ansehen

- Die ersten Zeilen sehen:

```
head(dat) # erste Zeilen  
tail(dat) # letzte Zeilen
```

- Einen Überblick mit Rstudio bekommen:



The screenshot shows the RStudio Environment pane. At the top, there are tabs for Environment, History, Plots, and Git. Below the tabs is a toolbar with icons for file operations and a search bar. The main area displays the Global Environment with a search bar. Under the 'Data' section, three objects are listed:

Object	Description	Icon
Chem97	31022 obs. of 8 variables	Calendar icon
dat	100 obs. of 23 variables	Calendar icon
Daten	100 obs. of 23 variables	Calendar icon

`View(dat)`

# Indizierung eines data.frame

```
dat[1,1] # das Element oben links bekommen
```

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

```
dat[2,] # nur die zweite Zeile sehen
```

```
##      X Stadtteilnummer Stadtteilname Wahlberechtigte.ohne.Sper  
## 2 2                2      Innenstadt
```

```
dat[,1] # sich nur die erste Spalte anzeigen lassen
```

```
## [1] 1 2 3 4 5 6
```

# Weitere Möglichkeiten zur Indizierung eines data.frame

```
dat[1:2,] # getting the first two rows
```

```
##      X Stadtteilnummer Stadtteilname Wahlberechtigte.ohne.Sper
## 1 1                      1      Altstadt
## 2 2                      2      Innenstadt
##      Wahlberechtigte.mit.Sperrvermerk Wahlberechtigte.insgesam
## 1                      495                      218
## 2                      653                      304
##      Wahlbeteiligung darunter.Wähler.mit.Wahlschein
## 1                      64.3                      473
## 2                      55.6                      607
##      Anteil.Wähler.mit.Wahlschein
## 1                      33.6
## 2                      35.8
```

# Indizierung

- Das Dollarzeichen kann auch zur Adressierung einzelner Spalten verwendet werden.

```
head(dat$Stadtteilname)
```

```
## [1] Altstadt      Innenstadt      Westend-Süd    Westend-Nord Nor  
## [6] Nordend-Ost  
## 45 Levels: Altstadt Bergen-Enkheim Berkersheim Bockenheim .
```

```
dat$Stadtteilname[1:10]
```

```
## [1] Altstadt      Innenstadt  
## [3] Westend-Süd    Westend-Nord  
## [5] Nordend-West    Nordend-Ost  
## [7] Ostend          Bornheim  
## [9] Gutleut-/Bahnhofsviertel Gallus
```



# Zugriff auf Spalten

- Wie bereits beschrieben, können Sie über Zahlen auf die Spalten zugreifen.

```
head(dat[,3])
```

```
## [1] Altstadt      Innenstadt      Westend-Süd    Westend-Nord Nor  
## [6] Nordend-Ost  
## 45 Levels: Altstadt Bergen-Enkheim Berkersheim Bockenheim .
```

```
head(dat[, "Stadtteilname"]) # dasselbe Ergebnis
```

```
## [1] Altstadt      Innenstadt      Westend-Süd    Westend-Nord Nor  
## [6] Nordend-Ost  
## 45 Levels: Altstadt Bergen-Enkheim Berkersheim Bockenheim .
```

# Logische Operatoren

```
(a <- 1:7) # Beispieldaten - numerisch
```

```
## [1] 1 2 3 4 5 6 7
```

```
a>4
```

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

```
a>=4
```

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

```
a<3
```

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

# Logische Operatoren II

```
(b <- letters[1:7]) # Beispieldaten - Strings
```

```
## [1] "a" "b" "c" "d" "e" "f" "g"
```

```
b=="e"
```

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

```
b %in% c("e", "f")
```

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

# GESIS Panel Variable - Estimated duration (bazq020a)

Wie lange haben Sie den Fragebogen ausgefüllt?

```
wahlberechtigte <- as.numeric(dat$Wahlberechtigte.insgesamt)
```

```
summary(wahlberechtigte)
```

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	2189	5723	9092	18609	11796	418703

# Missing values

- Fehlende Werte sind in R als NA definiert

```
wahlberechtigte[5] <- NA
```

- Bei mathematische Funktionen gibt es in der Regel eine Möglichkeit, fehlende Werte auszuschließen.
- Bei `mean()`, `median()`, `colSums()`, `var()`, `sd()`, `min()` und `max()` gibt es das Argument `na.rm`.

```
mean(wahlberechtigte)
```

```
## [1] NA
```

```
mean(wahlberechtigte, na.rm=T)
```

```
## [1] 18588.25
```

# Die fehlenden Werte finden

```
is.na(head(wahlberechtigte))
```

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

```
which(is.na(wahlberechtigte))
```

```
## [1] 5
```

```
table(is.na(wahlberechtigte))
```

```
##  
## FALSE TRUE  
##    44    1
```

# Der Befehl `complete.cases()`

```
# Beispiel Datensatz
```

```
mydata <- data.frame(A=c(1,NA,9,6),B=c("A","B",1,NA))
```

- Der Befehl `complete.cases()` gibt einen logischen Vektor zurück, der angibt, welche Fälle vollständig sind.

```
# Datenzeilen mit fehlenden Werten auflisten
```

```
mydata[complete.cases(mydata),]
```

```
##      A B
```

```
## 1 1 A
```

```
## 3 9 1
```

# Verschiedene Arten von fehlenden Werten (NAs) spezifizieren

- Spezifiziere verschiedene Arten von Fehlern mit dem Paket `memisc`.
- Benutze dazu den Befehl `include.missings()`

```
library(memisc)
```

```
?include.missings
```

- Es ist auch möglich, Codebuch-Einträge mit `memisc` zu erstellen.

```
codebook(dat$Wähler)
```



# Kategoriale Variable einführen

```
dat$wb_kat <- cut(dat$Wahlbeteiligung,3)
head(dat$wb_kat)
```

```
## [1] (58.8,67.5] (50,58.8] (67.5,76.3] (67.5,76.3] (67.5,76.3]
## Levels: (50,58.8] (58.8,67.5] (67.5,76.3]
```

```
levels(dat$wb_kat) <- c("niedrig", "mittel", "hoch")
head(dat$wb_kat)
```

```
## [1] mittel niedrig hoch hoch hoch hoch
## Levels: niedrig mittel hoch
```

# Datensatz indizieren

```
table(dat$wb_kat)
```

```
##  
## niedrig  mittel    hoch  
##      14      16      15
```

```
dat[dat$wb_kat=="mittel", "Stadtteilname"]  
dat[dat$wb_kat!="mittel", "Stadtteilname"]
```

## Weitere wichtige Optionen

- Speichern des Ergebnisses in einem Objekt

```
subDat <- dat[dat$Wahlbeteiligung>65,]
```

- mehrere Bedingungen können mit & verknüpft werden

```
dat[dat$Anteil.DIE.LINKE>12 & dat$Anteil.GRÜNE>25, "Stadtteilname"]
```

- das oder das Argument - eine der beiden Bedingungen muss erfüllt sein

```
dat[dat$Anteil.CDU>28 | dat$Anteil.AfD>28, "Stadtteilname"]
```

# Umbenennen der Spaltennamen

- Mit dem Befehl `colnames` erhält man die Spaltennamen

```
colnames(dat)
```

- Wir können die Spaltennamen umbenennen:

```
colnames(dat)[1] <- "Nummer"
```

- Das gleiche gilt für die Zeilenamen

```
rownames(dat)
```

# Exkurs - Wie man Labels verwendet

## Werkzeuge für das Arbeiten mit kategorischen Variablen (Faktoren)

```
library("forcats")
```

- `fct_collapse` - um Faktorstufen zu verdichten
- `fct_count` - um die Einträge in einem Faktor zu zählen
- `fct_drop` - Entferne unbenutzte Levels

# Der Befehl `fct_count`

## Freizeit Häufigkeit: Bücher lesen (a11c026a)

```
fct_count(f = dat$wbkat5)
```

```
## # A tibble: 5 x 2
##   f           n
##   <fct> <int>
## 1 1           5
## 2 2          12
## 3 3          11
## 4 4          10
## 5 5           7
```

# Der Befehl `fct_collapse`

```
wbkat <- fct_collapse(.f = dat$wbkat5,  
  hoch=c("4","5"))
```

```
fct_count(wbkate)
```

```
## # A tibble: 4 x 2  
##   f           n  
##   <fct> <int>  
## 1 1           5  
## 2 2          12  
## 3 3          11  
## 4 hoch       17
```

# Schleifen in R

- Der Befehl `for()` kennzeichnet den Start einer Schleife
- in Klammern, haben wir einen Index und die Anzahl der Läufe (in diesem Fall läuft die Schleife von 1 bis 4).
- in den geschweiften Klammern `{}` ist angegeben, was bei einer Iteration passiert.

```
for (i in 1:4){  
  cat(i, "\n")  
}
```

```
## 1  
## 2  
## 3  
## 4
```



# Schleifen - die Ergebnisse behalten

- Wir können die Ergebnisse in einem Objekt speichern
- dieses kann bspw. ein Vektor oder eine Liste sein.

```
erg <- list()

for (i in 1:ncol(dat)){
  erg[[i]] <- summary(dat)
  cat(i, "\n")
}
```

# Die apply Familie

```
(ApplyDat <- cbind(1:4,runif(4),rnorm(4))) #Example
```

```
##           [,1]      [,2]      [,3]
## [1,]      1 0.7953328  0.85442846
## [2,]      2 0.5409116 -1.31083037
## [3,]      3 0.5886844  0.01193112
## [4,]      4 0.4892190  0.44217802
```

```
apply(ApplyDat,1,mean)
```

```
## [1] 0.8832538 0.4100271 1.2002052 1.6437990
```

```
apply(ApplyDat,2,mean)
```

```
## [1] 2.500000000 0.603536959 -0.000573196
```

## Der Befehl `apply()`

```
apply(ApplyDat, 1, var)
```

```
## [1] 0.01109533 2.75324751 2.51260718 4.16431554
```

```
apply(ApplyDat, 1, sd)
```

```
## [1] 0.1053344 1.6592913 1.5851206 2.0406655
```

```
apply(X = ApplyDat, MARGIN = 1, FUN = range)
```

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.7953328 -1.31083 0.01193112 0.442178
## [2,] 1.0000000  2.00000 3.00000000 4.000000
```

# Die Argumente des Befehls `apply()`

- Wenn `MARGIN=1` wird die Funktion `mean` auf die Reihen angewendet,
- Wenn `MARGIN=2` wird die Funktion `mean` auf die Spalten angewendet,
- Anstatt `mean` kann man auch `var`, `sd` oder `length` verwenden.

# Der Befehl `tapply()`

```
ApplyDat <- data.frame(Income=rnorm(5,1400,200),  
                        Sex=sample(c(1,2),5,replace=T))
```

## Beispiel Befehl `tapply()`

```
tapply(ApplyDat$Income,  
       ApplyDat$Sex,function(x)x)
```

```
## $`1`  
## [1] 1376.446 1196.731  
##  
## $`2`  
## [1] 1280.205 1076.546 1471.397
```

- Andere Befehle können auch verwendet werden. .... auch selbst geschriebene

## Übung - `tapply()` Befehl verwenden

- Erstellen Sie eine Variable `wbkat`, in der sie die Wahlbeteiligung in den Stadtteilen in fünf Kategorien einteilen.
- Berechnen Sie mit Hilfe des `tapply` Befehls den durchschnittlichen Anteil der AFD pro Wahlbeteiligungskategorie.

# Edgar Anderson's Iris Datensatz

```
data(iris)
```

```
head(iris)
```

##	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
## 1	5.1	3.5	1.4	0.2
## 2	4.9	3.0	1.4	0.2
## 3	4.7	3.2	1.3	0.2
## 4	4.6	3.1	1.5	0.2
## 5	5.0	3.6	1.4	0.2
## 6	5.4	3.9	1.7	0.4

- petal length and width - Länge und Breite der Blütenblätter
- sepal length and width - Kelchlänge und -breite
- **Wikipedia Artikel zum IRIS Datensatz**

# Zusammenhang zwischen kontinuierliche Variablen

```
# Pearson correlation coefficient  
cor(iris$Sepal.Length,iris$Petal.Length)
```

```
## [1] 0.8717538
```

- Zusammenhang zwischen Blütenblattlänge und Blütenblattlänge ist 0,87
- Der Pearson-Korrelationskoeffizient ist die Standardmethode in `cor()`.



# Verschiedene Korrelationskoeffizienten

```
# Pearson correlation coefficient  
cor(iris[,1:4])
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width  
## Sepal.Length      1.0000000 -0.1175698    0.8717538    0.8179411  
## Sepal.Width       -0.1175698    1.0000000   -0.4284401   -0.3661259  
## Petal.Length       0.8717538   -0.4284401    1.0000000    0.9628654  
## Petal.Width        0.8179411   -0.3661259    0.9628654    1.0000000
```

```
# Kendall's tau (rank correlation)  
cor(iris[,1:4], method = "kendall")
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width  
## Sepal.Length      1.0000000 -0.07699679    0.7185159    0.6553618  
## Sepal.Width       -0.07699679    1.0000000   -0.1859944   -0.1571651  
## Petal.Length       0.71851593  -0.18599442    1.0000000    0.8068592
```

# Eine zweidimensionale Kreuztabelle erstellen

## Variablen

```
dat <- read.csv2("../data/bauenwohnen_teil.csv")
```

- Spiel100K - Wohnumfeld öffentlicher Raum - Spielplätze je 100 Kinder 2012
- baugenehm12 - Baugenehmigungen - Neue Ein/Zweifamilienhäuser 2012

## Tabelle erstellen

```
tab <- table(dat$Spiel100K, dat$baugenehm12)
```

# Kreuztabelle anschauen

## Tabelle

##		sehr wenig viele wenig		
##				
##	(0.198,1.03]	12	1	1
##	(1.03,1.87]	0	0	0
##	(1.87,2.7]	0	0	0

# Beziehung zwischen kategorialen Variablen

- `chisq.test()` prüft, ob zwei kategoriale Merkmale stochastisch unabhängig sind.
- Der Test wird gegen die Nullhypothese der Gleichverteilung durchgeführt.

```
chisq.test(tab)
```

```
##  
##  Pearson's Chi-squared test  
##  
## data:  tab  
## X-squared = NaN, df = 4, p-value = NA
```

# Übung - eine interaktive Tabelle

- Laden Sie den Datensatz `bauenwohnen_teil.RData` vom **Github Verzeichnis** herunter.
- Importieren Sie den Datensatz in R
- Erstellen Sie eine interaktive Tabelle mit den folgenden Befehlen:

```
library(DT)
DT::datatable(dat)
```

- Probieren Sie weitere Argumente der Funktion `datatable` aus.

# Shiny App für eine schnelle explorative Datenanalyse

<https://pharmacometrics.shinyapps.io/ggplotwithyourdata/>

Welcome to ggquicked!

Inputs

Graph Options

How To

Choose csv file to upload or use sample data

Browse...

ZA5666\_v1-0-0.csv

Upload complete

y variable(s):

Age

x variable:

Weight

ID

Time

Amt

Conc

Age

Weight

Gender

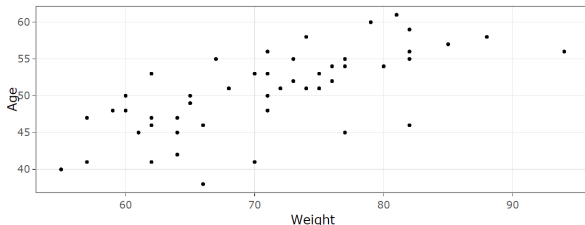
Race

N Digits

0

X/Y Plot   Export Plots   Experimental Plotly   Descriptive Stats   Data   Plot Code

Note: This is experimental and does not work all the time due to plotly:ggplotly limitations.



# Weitere Links

- **Tidy data** - das Paket `tidyr`
- Homepage für: **the tidyverse collection**
- **Data wrangling mit R und RStudio**
- Hadley Wickham - **Tidy Data**
- Hadley Wickham - **Advanced R**
- Colin Gillespie and Robin Lovelace **Efficient R programming**