

# **Etwas R am Abend**

Standort Köln – 06.12.2017

**Norman Markgraf**

- 1 Die Programmiersprache R
- 2 Installation von R und RStudio
- 3 Die erste Schritte in R
- 4 Strukturen in R
- 5 Statistische Funktionen
- 6 Ein paar Schritte in R
- 7 Eine kurze Datenanalyse
- 8 R intern
- 9 Anhang

# 1 Die Programmiersprache R

# 1. Die Programmiersprache R

## Was ist eigentlich R?

### Programmierersprache S:

- ▶ Von Bell Labs für Statistik, Simulation, Grafik entwickelt (Becker and Chambers; 1984)
- ▶ kommerzielle Implementation: **S-PLUS**

### Programmierersprache R:

- ▶ Implementation unter GPL (GNU General Public License), offener Quellcode

### Vorteile:

- ▶ Frei und Offen; Kostenlos
- ▶ Numerische Stabilität / Genauigkeit
- ▶ Methoden- und Anwendungsvielfalt (Finance, Marketing, HR, Psychologie, ...)
- ▶ Leicht erweiterbar durch eigene Routinen, Pakete, DLLs
- ▶ Standardisiertes, einfach handhabbares Datenformat (*data.frame*)
- ▶ Gut durchdachtes Format zur Anpassung von (Regressions-)Modellen
- ▶ Einfache Entwicklung ansprechender Diagramme und interaktiver Apps (z.B. [shiny](#)).
- ▶ Große und aktive Entwickler\*innen Gemeinde mit langer Geschichte: seit 1993; R Konsortium, u. a. IBM, Microsoft, TIPCO, Google, ...
- ▶ Neue Methoden der Datenanalyse werden häufig in R entwickelt (auch Big Data, KI, etc.)
- ▶ Schnittstellen zu sehr vielen Datenquellen / -banken (auch SocialMedia etc.)

### Nachteile:

- ▶ Bisher keine echte "Standard"-GUI (aber es gibt ja RStudio)
- ▶ Verfügbare Routinen / Pakete manchmal unübersichtlich

# 1. Die Programmiersprache R

## Wer nutzt R im echten Leben?

Unternehmen, die „ernsthaft“ Daten analysieren, setzen häufig auf R.



**Microsoft R Application Network**

The Microsoft R Portal

### ? What is R?

R is the world's most powerful programming language for statistical computing, machine learning and graphics as well as a thriving global community of users, developers and contributors.

Microsoft



**Quelle:** <http://www.revolutionanalytics.com/companies-using-r>

Falls Sie gerne **Werbevideos** ansehen, hier ein Link  
[https://www.youtube.com/watch?v=TR2bHSJ\\_eck](https://www.youtube.com/watch?v=TR2bHSJ_eck)

*[...] she was also following a wider trend: for many academics [...] R is the data-analysis tool of choice.*<sup>1</sup>

Verbreitung z. B.: <http://r4stats.com/articles/popularity/>

**R ist eine weit verbreitete Eintrittskarte in das globale Datenzeitalter!**

---

<sup>1</sup>Tippmann, S.. Programming tools (2015): Adventures with R. A guide to the popular, free statistics and visualization software that gives scientists control of their own data analysis. Nature, 517, S. 109–110.  
<https://doi.org/10.1038%2F517109a>

### Politik:

*Ich glaube, dass die Fähigkeit zum Programmieren eine der Basisfähigkeiten von jungen Menschen wird, neben Lesen, Schreiben, Rechnen. Die werden nicht wegfallen. Aber Programmieren wird nochmal dazu kommen.<sup>a</sup>*

<sup>a</sup>Rede von Bundeskanzlerin Merkel zur Deutsch-Französischen Digitalkonferenz am 13. Dezember 2016

### Wirtschaft:

*Der Prozess, eine komplexe Aufgabe auf eine Reihe einfacher Anweisungen zu reduzieren - genau darum geht es beim Programmieren -, ist eine Fähigkeit, die in vielen Aspekten des modernen Lebens nützlich ist, nicht nur für professionelle Informatiker und Programmierer.<sup>a</sup>*

<sup>a</sup>Facebooks Forschungschef Yann LeCun

### Lehre:

*Don't fence off students from the computation pool, throw them in! Computing skills are essential to working with data in the 21st century. Given this fact, we feel that to shield students from computing is to ultimately do them a disservice.<sup>a</sup>*

<sup>a</sup>Ismay, C, Kim, A (2017): ModernDive

- ▶ Dokumentation des Vorgehens
- ▶ (Einfache) Nachvollziehbarkeit, Wiederholung
- ▶ Möglichkeit zur Automatisierung und Übertragung
- ▶ “Direkte” Kommunikation mit dem Programm / Computer
- ▶ Speziell R: Unzählige Literatur und Hilfe / Tutorials im Internet



Wir nutzen das Paket **mosaic**, da es i. d. R. einer einfachen Idee gehorcht:

```
analysiere( y ~      # ggfs. abhängige Variable
            x      # unabhängige Variable(n)
            | z,    # ggfs. bedingende (gruppierende) Variable(n)
            Optionen, # ggfs. weitere Optionen
            data = daten ) # Datensatz
```

`analysiere()`: Was soll R tun?

*Hinweis für Mac-User:* unter macOS: ~: alt+n, |: alt+7

### Zentrale Fragen

1. Was soll der Computer für mich tun?
2. Was muss der Computer dafür wissen?

## 2 Installation von R und RStudio

Natürlich können Sie **R** als Programmiersprache direkt von der Konsole aus füttern.

Besser ist es aber seine Skripte vorab mit Hilfe eines Texteditors zu schreiben und **R** dieses ausführen zu lassen.

Noch besser ist die Nutzung von Integrierten Entwicklungsumgebung (*IDE*), wie z. B. **RStudio**

### Meine Empfehlung:

- ▶ **R** (3.4.3)
  - ▶ **R** finden Sie hier: <https://cran.rstudio.com> oder <https://www.r-project.org>
  - ▶ Aktuell ist die Version 3.4.3
  - ▶ **Achtung MAC-Nutzer!!!**: Sie benötigen zusätzlich erst noch XQuartz.
    - ▶ XQuartz finden Sie hier: <https://www.xquartz.org>
- ▶ **RStudio Desktop** (akt. Version: 1.1.383)
  - ▶ Die aktuelle Version finden Sie hier: <https://www.rstudio.com/products/rstudio/download/>
  - ▶ Nur Linux-Nutzer **RStudio Server** (akt. Version: 1.1.383)

(Und *bitte*, vergessen Sie den **R-CmdR**! Wenn schon, dann schauen Sie sich einmal **jamovi** an!)

0. (Mac-User bitte zuerst [xquartz](#) installieren)!
1. R (<https://www.r-project.org/>)
2. RStudio Desktop (<https://www.rstudio.com/>)
3. Installation von Zusatzpaketen in RStudio:

### Tipps für den Installationsprozess:

- ▶ Abwarten und bestätigen ;-)
- ▶ Kaffee / Tee / Wasser / ... trinken und ruhig bleiben!

Die “Fehlermeldungen” gehören meisst zum Installationsprozess. Wenn etwas gar nicht läuft. Fragen!

```
install.packages("mosaic")
```

R ist eine *komandozeilenorientierte*-Sprache!

```
1+1
```

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

```
1+2*3^4
```

```
## [1] 163
```

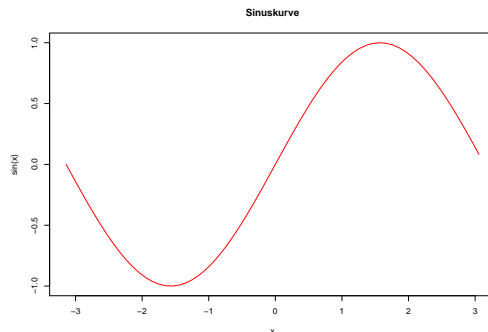
```
x <- 1; y <- 2
```

```
x+y
```

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

Die (recht schnell) schöne Ergebnisse produziert:

```
x <- seq(-pi,pi,by=0.1)
plot(x, sin(x), type="l", col="red",
     main="Sinuskurve")
```



Im Allgemeinen installiert man ein Paket durch den Befehl:

```
install.packages("<blubber>", dependencies = TRUE)
```

Für einen guten Start sollte man vor allem ein Paket installieren:

Das Paket **mosaic**!

Mehr Informationen zu **mosaic** finden Sie hier:

- ▶ [Project MOSAIC](#)
- ▶ [Less Volume, More Creativity – Getting Started with the mosaic Package](#)

## 2. Installation von R und RStudio

# Das Paket MOSAIC installieren

Ihr erster R Befehle sollten wie folgt lauten:

```
# Laden des mosaic Pakets:  
install.packages("mosaic", dependencies=TRUE)
```

Mit '#' leitet man einen Kommentar ein. Sie müssen aber die Beispiele nicht mit den Kommentaren eintippen, es reicht:

```
install.packages("mosaic", dependencies=TRUE)
```

Bitte bestätigen Sie alle Anfragen und haben Sie etwas Geduld. Es wird eine Menge nachgeladen. Aber nur einmal. Also keine Sorge!

Jede R Installation hat einen Vorratsspeicher, in dem die Pakete abgelegt werden. Mit dem Befehl `install.package()` laden Sie dieses Paket aus dem Internet in diesen Vorratsspeicher.

Ähnlich wie Sie ein Buch aus der Bücherrei / Buchhandlung kaufen und ins Regal stellen.

Wenn Sie nun so ein Buch lesen wollen, dann müssen Sie es in die Hand nehmen, aufschlagen und lesen.

In R wird ein Buch aus dem Vorratsspeicher geholt und R angewiesen damit zu arbeiten in dem man den Befehl `library()` nutzt.

**Denken Sie daran:** Sie kaufen ein Buch nur einmal! Lesen tun Sie es aber öfter! Dementsträgend müssen Sie ein Paket nur *einmal* in den Vorratsspeicher laden, können es aber dann mit `library()` beliebig oft benutzen!



# 3 Die erste Schritte in R

*# Punkt- vor Strichrechnung*

```
2 * 3 + 2 - 25/5 + 2^3
```

```
## [1] 11
```

*# Trigometrische Funktionen*

```
cos(pi/2)^2 + sin(pi/2)^2
```

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

*# Logarithmen & Exponentialfunktion*

```
log(exp(3))
```

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

*# Unendlich*

```
1/0
```

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

*# Not a Number (keine Zahl)*

```
0/0
```

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

*# Not Available; ein fehlender Wert*

```
NA
```

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

*# Vektoren (combine)*

```
c(1, 4:8)
```

```
## [1] 1 4 5 6 7 8
```

*# Vektor/Liste ohne Inhalt*

```
c()
```

```
## NULL
```

## 4 Strukturen in R

- ▶ Variablen in **R** können Skalare, Vektoren, Matrizen oder Objekte beliebiger anderer Klassen sein.
- ▶ Man **erzeugt** eine Variable in dem man ihr mit Hilfe von "<-" oder "=" etwas **zuweist**.
- ▶ **Variablennamen** können Kombinationen aus Buchstaben, Ziffern, Punkt und Unterstrich sein. Aber **keine Ziffern vorne!**
- ▶ **R** ist **case-sensitiv**, es unterscheidet zwischen Groß- und Kleinschreibung!

```
a <- c("FOM", "und", "R", "sind", "SUPER")
```

```
A <- 42
```

```
a
```

```
## [1] "FOM"    "und"    "R"      "sind"   "SUPER"
```

```
A
```

```
## [1] 42
```

In **R** gibt es die Datentypen

- ▶ **numeric** - ganzzahlige (*integer*) oder reelle (*double*) Zahlen
- ▶ **character** - Zeichenketten
- ▶ **logic** - die logischen Operatoren **TRUE** und **FALSE**
- ▶ **list** - Liste von Objekten jeder Art (die wiederum Listen beinhalten können!)

Befehle zum überprüfen der Datentypen:

```
mode(a)
```

```
## [1] "character"
```

```
str(a)
```

```
## chr [1:5] "FOM" "und" "R" "sind" "SUPER"
```

```
typeof(a)
```

```
## [1] "character"
```

Ein Vektor wird mit dem Befehl `c()` (für *combine*) erzeugt:

```
a <- 5
vektorMitBeliebigenNamen <- c(log(1), a, sqrt(16), 3^2)
vektorMitBeliebigenNamen
```

```
## [1] 0 5 4 9
```

R kann (Rechen-)Operationen auf ganzen Vektoren (elementweise) durchführen:

```
vektorMitBeliebigenNamen * 2
```

```
## [1] 0 10 8 18
```

```
vektorMitBeliebigenNamen + 1
```

```
## [1] 1 6 5 10
```

Zahlensequenzen werden mit dem Befehl **seq()** erzeugt. Dem Befehl können verschiedene Argumente übergeben werden:

```
seq(from = 2, to = 9)
```

```
## [1] 2 3 4 5 6 7 8 9
```

```
seq(from = 2, to = 8, by=3)
```

```
## [1] 2 5 8
```

```
seq(from = 2, by = 0.5, length.out = 8)
```

```
## [1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

```
# 'n:m' entspricht
```

```
# seq(from=n, to=m, by=1)
```

```
vektor <- 1:4
```

```
vektor
```

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

```
# Werte mit rep() wiederholen::
```

```
rep("X", times = 5)
```

```
## [1] "X" "X" "X" "X" "X"
```

```
zahlen1 <- c(2, 4)
```

```
zahlen1
```

```
## [1] 2 4
```

```
zahlen2 <- rep(zahlen1, times = 2)
```

```
zahlen2
```

```
## [1] 2 4 2 4
```

```
rep(zahlen1, each = 2)
```

```
## [1] 2 2 4 4
```

```
people <- c("Klaus", "Max", "Jens", "Dieter")  
people
```

```
## [1] "Klaus" "Max" "Jens" "Dieter"
```

```
people == "Max"
```

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

```
vektorMitBeliebigenNamen != 0
```

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

```
logischerVektor <- vektorMitBeliebigenNamen <= 3  
logischerVektor
```

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



`names(a)` gibt die Namen der Einträge des Vektors *a* zurück:

```
weight <- c(67, 80, 72, 90)
names(weight)
```

```
## NULL
```

```
weight
```

```
## [1] 67 80 72 90
```

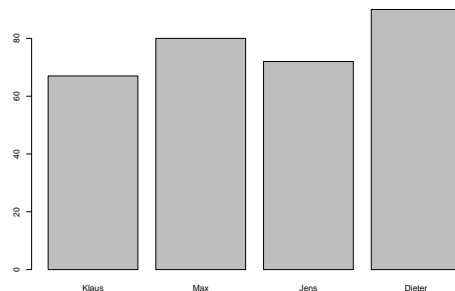
Jetzt geben wir den Werten einen Namen:

```
names(weight) <- people
weight
```

```
## Klaus   Max   Jens Dieter
##      67    80    72    90
```

Diese Namen werden von **R** sehr oft mit ausgewertet und verwendet:

```
barplot(weight)
```



In Vektoren speichern wir Datenreihen.

**Wichtige** Befehle für Vektoren sind - *mean()*, *sd()*, *var()*, *min()*, *max()*, *length()*, *sum()*, *median()*, *IQR()*, *summary()* - **Zugriff** auf das *i*-te Element eines Vektors *a* mit *a[i]*.

```
aVec <- c(1, 2, 4, 9, 16, 25)
```

```
mean(aVec)
```

```
## [1] 9.5
```

```
sd(aVec)
```

```
## [1] 9.396808
```

```
var(aVec)
```

```
## [1] 88.3
```

```
min(aVec)
```

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

```
max(aVec)
```

```
## [1] 25
```

```
length(aVec)
```

```
## [1] 6
```

```
sum(aVec)
```

```
## [1] 57
```

```
median(aVec)
```

```
## [1] 6.5
```

```
IQR(aVec)
```

```
## [1] 11.75
```

```
summary(aVec)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      1.00   2.50   6.50   9.50  14.25  25.00
```

```
library(mosaic)
```

```
favstats(aVec)
```

```
##   min  Q1 median    Q3 max mean      sd n missing
##    1 2.5   6.5 14.25  25  9.5 9.396808 6         0
```

```
quantile(aVec)
```

```
##      0%   25%   50%   75%  100%
##      1.00  2.50  6.50 14.25 25.00
```

---

## 5 Statistische Funktionen

**R** berechnet die Varianz von Daten mit Hilfe der (Schätzer-)Formel

$$\frac{1}{n-1} \cdot \sum (x - \bar{x})^2,$$

wie man leicht nachrechnen kann:

```
n <- length(aVec)
var(aVec)
```

```
## [1] 88.3
```

```
# ist das selbe wie
```

```
1/(n - 1) * sum((aVec - mean(aVec))^2)
```

```
## [1] 88.3
```

```
# Dagegen ist
```

```
1/n * sum((aVec - mean(aVec))^2)
```

```
## [1] 73.58333
```

Die Standardabweichung ist die Quadratwurzel der Varianz:

```
sd(aVec)
```

```
## [1] 9.396808
```

```
sqrt(var(aVec))
```

```
## [1] 9.396808
```

Will man die Varianz und die Standardabweichung mit Hilfe der Formel

$$\frac{1}{n} \cdot \sum (x - \bar{x})^2,$$

berechnen, so muss man in **R** etwas tun:

```
n <- length(aVec)
factor <- (n - 1)/(n)
# Wert
var(aVec)
```

```
## [1] 88.3
```

```
# Korrigierter Wert
```

```
factor * var(aVec)
```

```
## [1] 73.58333
```

```
# Zur Probe:
```

```
1/n * sum((aVec - mean(aVec))^2)
```

```
## [1] 73.58333
```

```
factorSD <- sqrt((n-1)/n)
```

```
# Wert von R:
```

```
sd(aVec)
```

```
## [1] 9.396808
```

```
# Korrigierter Wert
```

```
factor*sd(aVec)
```

```
## [1] 7.830673
```

```
# Zur Probe
```

```
sqrt(1/n * sum( (aVec-mean(aVec))^2 ))
```

```
## [1] 8.578073
```

```
aVec2 <- rep(2, 6)
aVec
## [1] 1 2 4 9 16 25
```

```
aVec2
## [1] 2 2 2 2 2 2
```

```
# Skalarprodukt
aVec %*% aVec2
```

```
##      [,1]
## [1,] 114
```

```
# Komponentenweises *
aVec * aVec2
```

```
## [1] 2 4 8 18 32 50
```

```
aVec3 <- aVec[2:5]
aVec3
## [1] 2 4 9 16
```

```
aVec3[3]
## [1] 9
```

```
aVec3[3] <- NA
aVec3
## [1] 2 4 NA 16
```

```
mean(aVec3)
```

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

```
# NA ignorieren:
mean(aVec3, na.rm = TRUE)
## [1] 7.333333
```

Merkmale eines Merkmalsträgers fasst man gelegentlich zu Tabellen zusammen. In **R** kann man dazu *data frames* nutzen:

*# Die Datenreihen*

```
Name <- c("Anna", "Beria", "Carlo", "Edda")
Geschlecht <- c("weiblich", "weiblich", "männlich", "weiblich")
Groesse <- c(1.60, 1.68, 1.81, 1.71)
Gewicht <- c(52, 60, 80, 70)
```

Diese Datenreihen kann man wie folgt zu einem Objekt zusammenfassen:

*# Als Dataframe:*

```
df <- data.frame(
  Name = Name,
  Geschlecht = Geschlecht,
  Groesse = Groesse,
  Gewicht = Gewicht)
#
```



```
df
```

```
##      Name Geschlecht Groesse Gewicht
## 1  Anna    weiblich    1.60      52
## 2 Beria    weiblich    1.68      60
## 3 Carlo    männlich    1.81      80
## 4  Edda    weiblich    1.71      70
```

In *RMarkdown*-Skripten möchte man gerne eine schönere Ausgabe haben. Dies geht z. B. mit dem Befehl `kable` aus dem Paket `knitr`:

```
library(knitr)
kable(df)
```

Name	Geschlecht	Groesse	Gewicht
Anna	weiblich	1.60	52
Beria	weiblich	1.68	60
Carlo	männlich	1.81	80
Edda	weiblich	1.71	70

## 5. Statistische Funktionen

# Arbeiten mit Dataframes

```
# Ausgabe der Durchschnittsgröße:
```

```
mean(df$Groesse)
```

```
## [1] 1.7
```

```
# Ausgabe des Durchschnittsgewichts:
```

```
mean(df$Gewicht)
```

```
## [1] 65.5
```

```
library(dplyr)
```

```
# Nur die Damen
```

```
mean(filter(df, Geschlecht=="weiblich")$Groesse)
```

```
## [1] 1.663333
```

```
# Nur die Herren
```

```
mean(filter(df, Geschlecht=="männlich")$Gewicht)
```

```
## [1] 80
```

*# BMI der Tabelle hinzufügen:*

```
df <- (mutate(df, BMI = round(Gewicht/Groesse^2,
```

*# Nur das Geschlecht und den*

*# BMI auswählen:*

```
df2 <- select(df,  
  c(Geschlecht, BMI))
```

*# Nur das Geschlecht und*

*# den BMI ausgeben*

```
kable(df2)
```

Geschlecht	BMI
weiblich	20.31
weiblich	21.26
männlich	24.42
weiblich	23.94

*# Der Größe nach sortieren (absteigend)*

```
df3 <- arrange(df2, desc(Groesse))
```

```
kable(df3)
```

Geschlecht	BMI
männlich	24.42
weiblich	23.94
weiblich	21.26
weiblich	20.31

## 6 Ein paar Schritte in R

## 6. Ein paar Schritte in R

### Vorbereitungen

Wir arbeiten mit **mosaic**. Daher laden wir als erstes (immer) das Paket in den Speicher von R. Als zweites setzen wir hier mit `set.seed(2009)` den Zufallszahlengenerator von R auf einen von uns gewählten Startwert; dies dient der Reproduzierbarkeit!

```
library(mosaic)
set.seed(2009)
```

Wir wollen nun den Datensatz “miete03” aus dem Netz laden

- ▶ Via **RStudio**: Gehen Sie auf das recht obere Fenster, klicken Sie auf *Environment*, Dann klicken Sie **Import Dataset**. Danach **From Text (readr)** und geben Sie als URL bitte <https://raw.githubusercontent.com/NMarkgraf/Etwas-R-zum-Nachmittag/master/Datasets/miete03.asc> oder einfacher <https://tinyurl.com/yblxykf3> ein drücken Sie auf **Update**. Stellen Sie dann den **Delimiter** auf **Tab** (und warten oder drücken nochmal auf **Update**). Mit **Import** wird dann der Datensatz geladen.
- ▶ Via **R** direkt: Man kann auch direkt aus **R** mittels ein paar Zeilen die Daten laden:

```
miete03 <- read.table(file = "https://tinyurl.com/yblxykf3", header = TRUE)
```

Falls Sie die Daten via RStudio-Oberfläche geladen haben (oder mittels des im *Code preview*) angegebenen Codes, werden sogenannte `tibbles` statt `data.frames` geladen.

**Tibbles** sind eine Weiterentwicklung der klassischen `data.frames`. Sie erkennen den Unterschied z. B. in dem Sie sich die ersten Zeilen ansehen.

Bei einem **Tibble** sieht die ersten Zeilen wie folgt aus:

```
## Parsed with column specification:
## cols(
##   nm = col_double(),
##   nmqm = col_double(),
##   wfl = col_integer(),
##   rooms = col_integer(),
##   bj = col_double(),
##   bez = col_integer(),
##   wohngut = col_integer(),
##   wohnbest = col_integer(),
##   ww0 = col_integer(),
##   zh0 = col_integer(),
##   badkach0 = col_integer(),
##   badextra = col_integer(),
##   kueche = col_integer()
```

Bei einem Datenrahmen (data.frame) dagegen:

```
miete03 <- read.table(file = "https://tinyurl.com/yblxykf3", header = TRUE)
```

```
head(miete03, 2)
```

```
## # A tibble: 2 x 13
##       nm nmqm  wfl rooms  bj  bez wohngut wohnbest  ww0  zh0
##   <dbl> <dbl> <int> <int> <dbl> <int>   <int>   <int> <int> <int>
## 1 741.39 10.90   68     2 1918     2     1     0     0     0
## 2 715.82 11.01   65     2 1995     2     1     0     0     0
## # ... with 3 more variables: badkach0 <int>, badextra <int>,
## #   kueche <int>
```

Mit dem Befehl:

```
miete03 <- as.data.frame(miete03)
```

wandeln wir, falls gewünscht, **tibbles** in klassische data.frames um.

Zahlreiche deutsche Städte erstellen sogenannte Mietspiegel, um Mietern, Vermietern, Mietberatungsstellen und Sachverständigen eine möglichst objektive Entscheidungshilfe in Mietfragen zur Verfügung zu stellen. Die Mietspiegel werden dabei insbesondere zur Ermittlung der "ortsüblichen Vergleichsmiete" (Nettomiete in Abhängigkeit von Wohnungsgröße, -ausstattung, -alter, etc.) herangezogen. Bei der Erstellung von Mietspiegeln wird aus der Gesamtheit aller in Frage kommenden Wohnungen eine repräsentative Zufallsstichprobe gezogen (im Fall der Stadt München durch Infratest), und die interessierenden Daten werden von Interviewern anhand von Fragebögen ermittelt.

Der vorliegende Datensatz stellt einen Ausschnitt aus dem Mietspiegel München des Jahres 2003 dar und enthält die Daten von 2053 Wohnungen.

Schauen wir uns die Spaltenüberschriften einmal kurz an:

```
names(miete03)
```

```
## [1] "nm"          "nmqm"        "wfl"         "rooms"       "bj"          "bez"
## [7] "wohngut"     "wohnbest"    "ww0"         "zh0"         "badkach0"    "badextra"
## [13] "kueche"
```



## 6. Ein paar Schritte in R

### Der Befehl `inspect()`

```
inspect(miete03)
```

```
##
## quantitative variables:
##      name  class   min      Q1 median      Q3      max      mean
## 1      nm numeric  77.31 389.95 534.30 700.48 1789.55 5.700930e+02
## 2     nmqm numeric   1.47   6.80   8.47  10.09  20.09 8.393902e+00
## 3      wfl integer  17.00  53.00  67.00  83.00 185.00 6.959523e+01
## 4    rooms integer   1.00   2.00   3.00   3.00   6.00 2.597662e+00
## 5       bj numeric 1918.00 1948.00 1960.00 1973.00 2001.00 1.957983e+03
## 6      bez integer   1.00   5.00  10.00  17.00  25.00 1.126790e+01
## 7  wohngut integer   0.00   0.00   0.00   1.00   1.00 3.911349e-01
## 8  wohnbest integer   0.00   0.00   0.00   0.00   1.00 2.191914e-02
## 9      ww0 integer   0.00   0.00   0.00   0.00   1.00 3.507063e-02
## 10     zh0 integer   0.00   0.00   0.00   0.00   1.00 8.524111e-02
## 11 badkach0 integer   0.00   0.00   0.00   0.00   1.00 1.850950e-01
## 12 badextra integer   0.00   0.00   0.00   0.00   1.00 9.303458e-02
## 13  kueche integer   0.00   0.00   0.00   0.00   1.00 7.306381e-02
##
##      sd      n missing
## 1 245.4345066 2053      0
## 2  2.4667425 2053      0
## 3 25.1625576 2053      0
```

## Quiz: Sind die Daten plausibel?

---

Schauen Sie sich die Daten und die Spaltenbezeichner an, sind die von R gewählten Datentypen (numeric und integer) wirklich immer passend?

Falls nein, wo würden Sie gerne warum was ändern?

```
miete03$badextra <- as.factor(miete03$badextra) # Extra Bad
miete03$badkach0 <- as.factor(miete03$badkach0) # Bad gekachelt
miete03$bez <- as.factor(miete03$bez) #
miete03$kueche <- as.factor(miete03$kueche) # Küche
miete03$wohnbest <- as.factor(miete03$wohnbest) #
miete03$wohngut <- as.factor(miete03$wohngut) #
miete03$ww0 <- as.factor(miete03$ww0) # Warmwasser
miete03$zh0 <- as.factor(miete03$zh0) # Zentralheizung
levels(miete03$zh0) <- c("ja", "nein")
miete03$rooms <- as.factor(miete03$rooms) # Anzahl der Zimmer
```

Mit dem Befehl **tally** können wir *Häufigkeitstabellen* erstellen:

Absolute Häufigkeitstabelle

```
tally( ~ rooms, data=miete03,
      format="count")
```

```
## rooms
##    1    2    3    4    5    6
## 255 715 759 263  47   14
```

Relative Häufigkeitstabelle in Prozent

```
options(digits=2)
tally( ~ rooms, data=miete03,
      format="percent")
```

```
## rooms
##    1    2    3    4    5    6
## 12.42 34.83 36.97 12.81  2.29  0.68
```

Relative Häufigkeitstabelle

```
options(digits=1)
tally( ~ rooms, data=miete03,
      format="proportion")
```

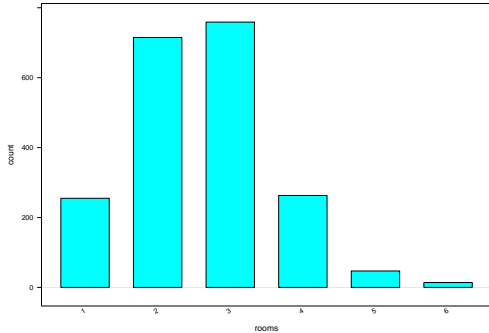
```
## rooms
##    1    2    3    4    5    6
## 0.124 0.348 0.370 0.128 0.023 0.007
```

```
tally( zh0 ~ rooms, data=miete03,
      format="count")
```

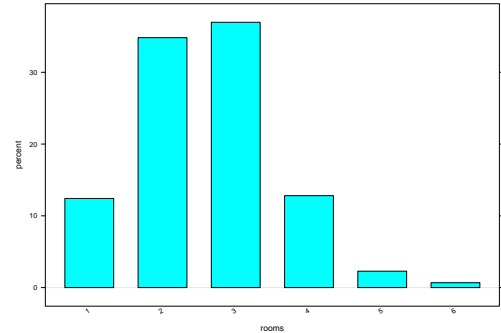
```
##          rooms
## zh0      1    2    3    4    5    6
##   ja    246 644 690 244  42   12
##   nein    9  71  69  19   5    2
```

Und mit dem Befehl **bargraph()** erstellen wir ein Balkendiagramm daraus:

```
bargraph(~rooms, data = miete03)
```



```
bargraph(~rooms, data = miete03, type = "percent")
```

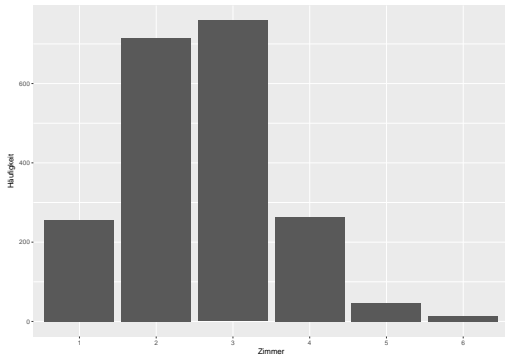


#

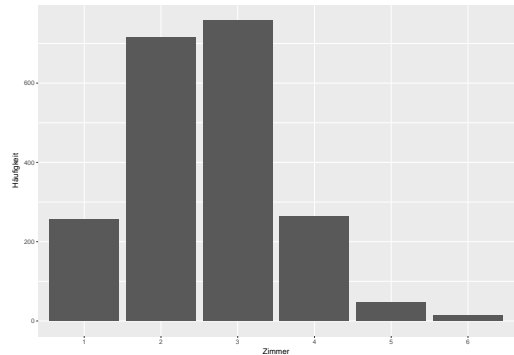
### Kurzer Exkurs:

Mit dem Paket ggplot2 und der Funktion qplot() würden sich auch schöne Graphiken erstellen lassen!

```
library(ggplot2)
qplot(miete03$rooms, bins=12,
      xlab="Zimmer", ylab="Häufigkeit")
```



```
library(ggplot2)
ggplot(miete03, aes(x = rooms)) +
  geom_bar() + xlab("Zimmer") +
  ylab("Häufigkeit")
```



Wie man mit **ggplot2** noch mehr und noch schönere Grafiken erstellt, können Sie finden bei:

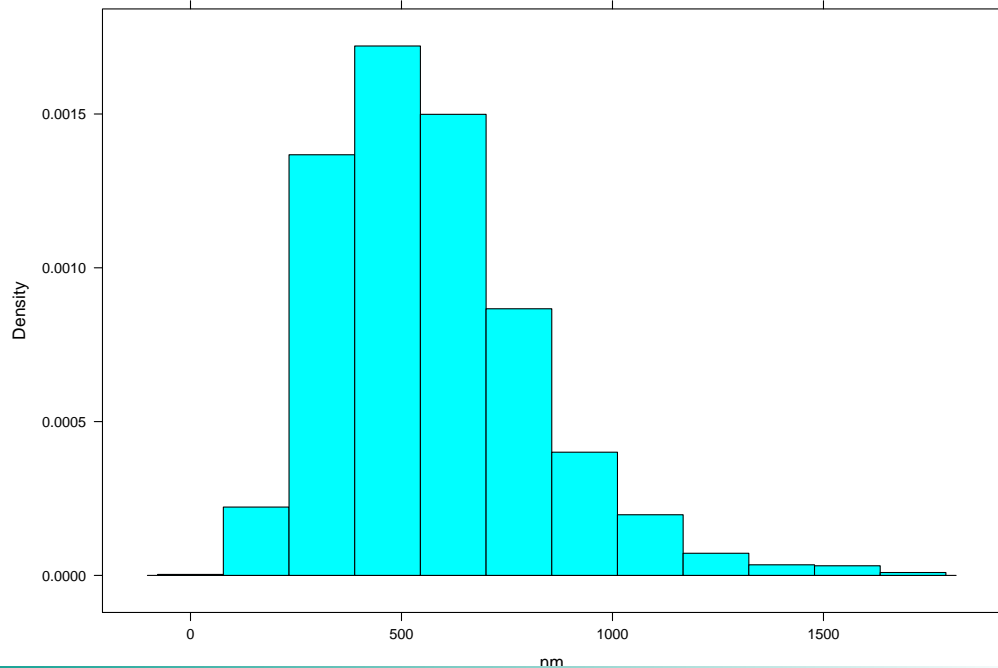
- ▶ <http://ggplot2.org>
- ▶ <http://docs.ggplot2.org/current/index.html>
- ▶ <http://www.cookbook-r.com/Graphs/>
- ▶ <https://www.datacamp.com/courses/data-visualization-with-ggplot2-1>
- ▶ <http://r4ds.had.co.nz>

## 6. Ein paar Schritte in R

# Die Nettokaltmiete

Automatische Klassenanzahl:

```
histogram( ~ nm, data=miete03)
```



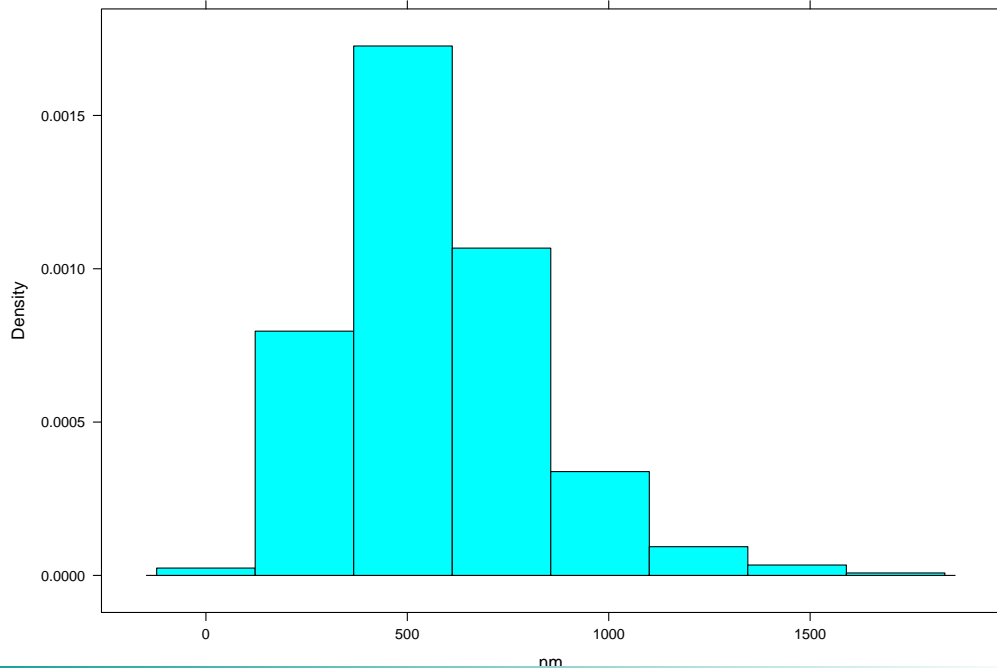


## 6. Ein paar Schritte in R

# Die Nettokaltmiete

Vorgegebene Klassenanzahl von 8 (wird ungefähr eingehalten):

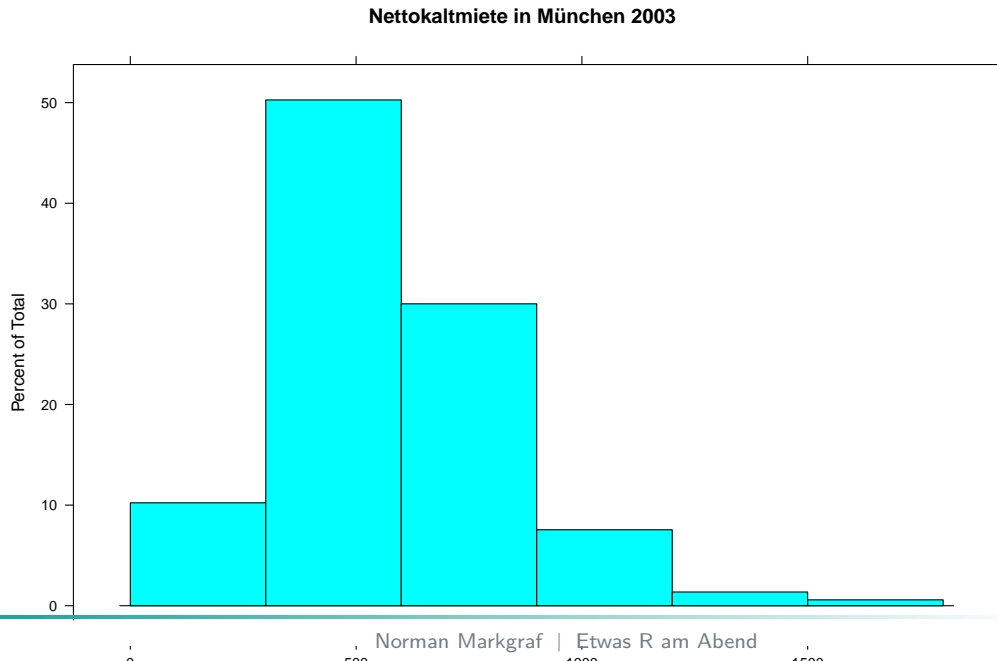
```
histogram( ~ nm, data=miete03, nint=8)
```



## 6. Ein paar Schritte in R

# Die Nettokaltmiete

```
histogram( ~ nm, data=miete03, type="percent",  
           breaks=c(0, 300, 600, 900, 1200, 1500, 1800),  
           main="Nettokaltmiete in München 2003",  
           sub="Stichprobe vom Umfang n=2053",  
           xlab="Nettokaltmiete in Euro")
```



# 7 Eine kurze Datenanalyse

Einlesen der Daten:

Im Folgenden kann man ein paar der Möglichkeiten von **R** finden, wie man sie in Vorlesungen braucht.

Dazu nehmen wir die “tipping” Daten<sup>2</sup> in den Datenrahmen *tips*.

```
# Herunterladen
download.file("https://goo.gl/whKjnl", destfile = "tips.csv")
# Einlesen in R
tips <- read.csv2("tips.csv")

# Alternativ - heruntergeladene Datei einlesen:
# tips <- read.csv2(file.choose())
```

Darüberhinaus nutzen wir einige Befehle aus der Paketsammlung **Mosaic**, welches wir dazu laden

```
library(mosaic) # Hauptsächlich "lattice"
```

---

<sup>2</sup>Bryant, P. G. and Smith, M (1995) Practical Data Analysis: Case Studies in Business Statistics. Homewood, IL: Richard D. Irwin Publishing

Schauen wir uns die Daten etwas an:

```
inspect(tips)
```

```
## categorical variables:
##      name  class levels    n missing                distribution
## 1    sex factor      2 244          0 Male (64.3%), Female (35.7%)
## 2 smoker factor      2 244          0 No (61.9%), Yes (38.1%)
## 3    day factor      4 244          0 Sat (35.7%), Sun (31.1%), Thur (25.4%) ...
## 4    time factor      2 244          0 Dinner (72.1%), Lunch (27.9%)
##
## quantitative variables:
##      name  class  min      Q1 median      Q3    max      mean      sd    n missing
## 1 total_bill numeric 3.07 13.3475 17.795 24.1275 50.81 19.78594 8.90241 244        0
## 2      tip numeric 1.00  2.0000  2.900  3.5625 10.00  2.99828 1.38364 244        0
## 3      size integer 1.00  2.0000  2.000  3.0000  6.00  2.56967 0.95110 244        0
```

Umwandeln der Tischgröße in eine kategoriale Variable:

```
levels(tips$size)
```

```
## NULL
```

```
tips$size <- as.factor(tips$size)  
levels(tips$size)
```

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

Schauen wir uns die Daten erneut an:

```
inspect(tips)
```

```
## categorical variables:
```

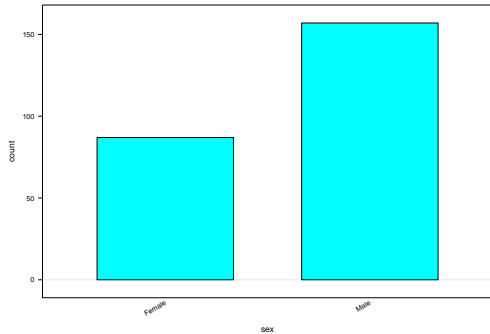
##	name	class	levels	n	missing	distribution
## 1	sex	factor	2	244	0	Male (64.3%), Female (35.7%)
## 2	smoker	factor	2	244	0	No (61.9%), Yes (38.1%)
## 3	day	factor	4	244	0	Sat (35.7%), Sun (31.1%), Thur (25.4%) ...
## 4	time	factor	2	244	0	Dinner (72.1%), Lunch (27.9%)
## 5	size	factor	6	244	0	2 (63.9%), 3 (15.6%), 4 (15.2%) ...

```
##
```

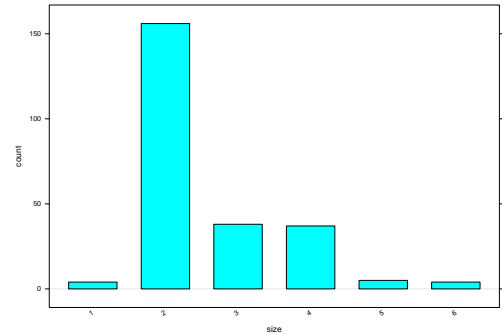
```
## quantitative variables:
```

##	name	class	min	Q1	median	Q3	max	mean	sd	n	missing
## 1	total_bill	numeric	3.07	13.3475	17.795	24.1275	50.81	19.78594	8.90241	244	0
## 2	tip	numeric	1.00	2.0000	2.900	3.5625	10.00	2.99828	1.38364	244	0

```
bargraph(~sex, data = tips)
```



```
bargraph(~size, data = tips)
```

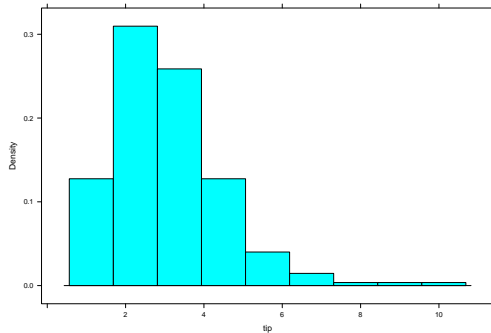




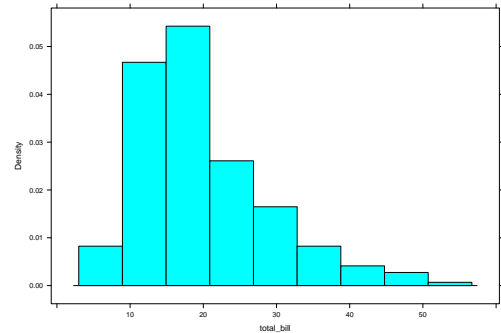
## 7. Eine kurze Datenanalyse

# Histogramm bei metrischen Daten

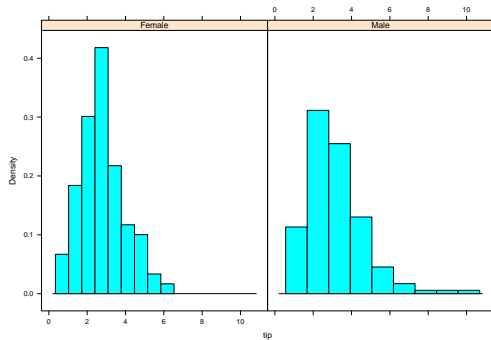
```
histogram(~tip, data = tips)
```



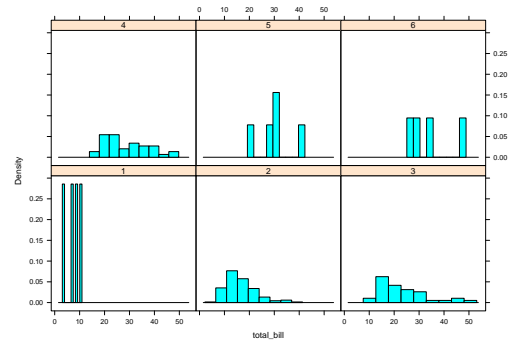
```
histogram(~total_bill, data = tips)
```



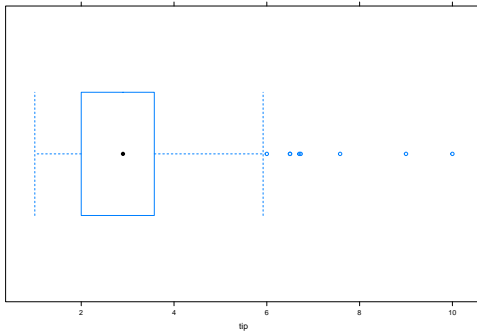
```
histogram( ~ tip | sex,  
           data=tips)
```



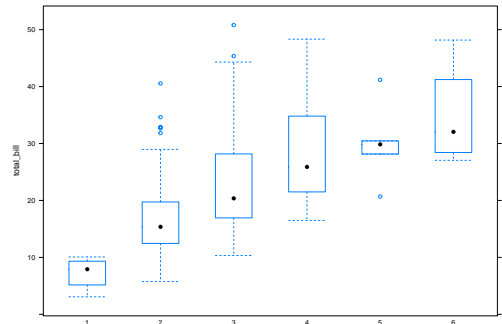
```
histogram( ~ total_bill | size,  
           data=tips)
```



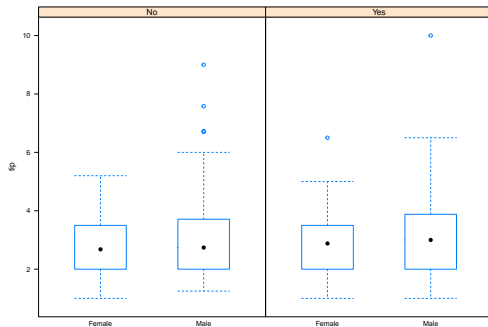
```
bwplot( ~ tip, data=tips)
```



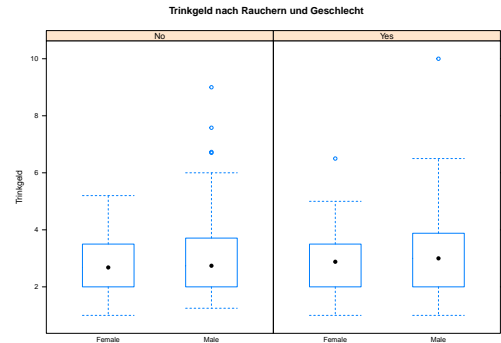
```
bwplot(total_bill ~ size, data=tips)
```



```
bwplot(tip ~ sex | smoker, data = tips)
```



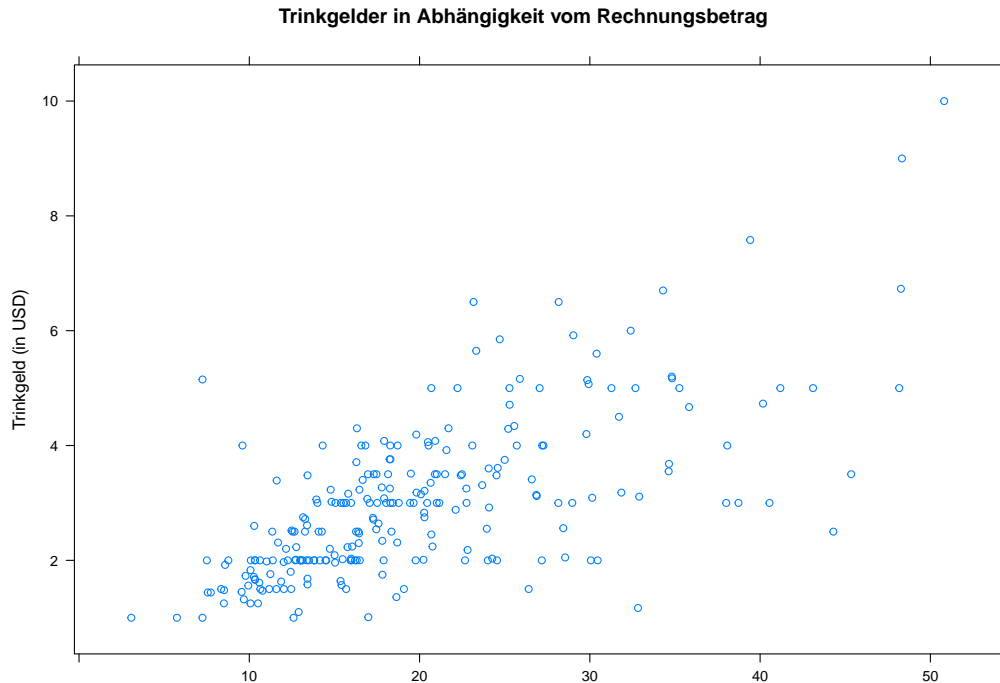
```
bwplot(tip ~ sex | smoker, data = tips,
       ylab = "Trinkgeld")
```



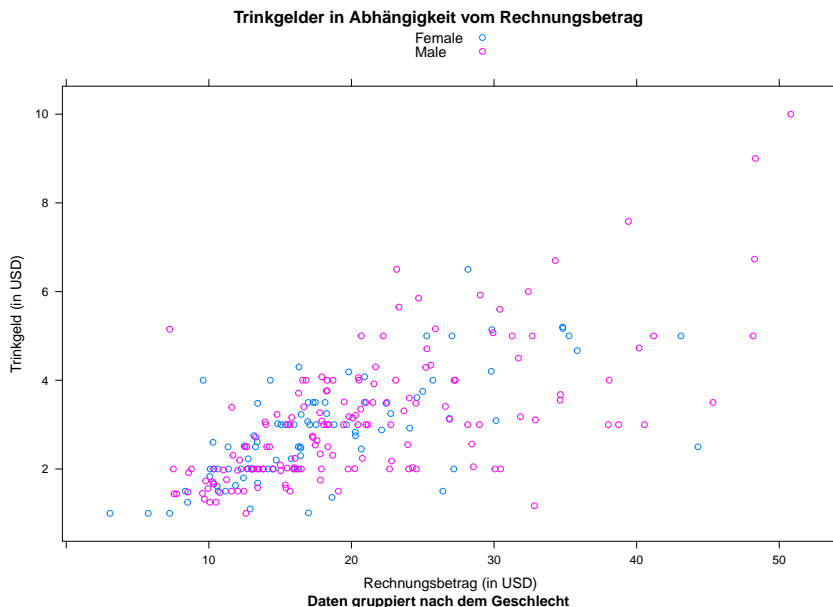
#

# Streudiagramm mit zwei metrischen Variablen

```
xyplot(tip~total_bill, data=tips,  
       ylab="Trinkgeld (in USD)", xlab="Rechnungsbetrag (in USD)",  
       main="Trinkgelder in Abhängigkeit vom Rechnungsbetrag")
```



```
xyplot(tip~total_bill, data=tips,  
  ylab="Trinkgeld (in USD)", xlab="Rechnungsbetrag (in USD)",  
  main="Trinkgelder in Abhängigkeit vom Rechnungsbetrag",  
  sub="Daten gruppiert nach dem Geschlecht",  
  groups=sex, auto.key = T)
```



Dazu generieren wir die Häufigkeitstabelle mit dem Befehl *tally* und speichern sie in *tab*

```
tab <- tally(sex ~ smoker, data = tips)
tab
```

```
##           smoker
## sex      No Yes
## Female  54  33
## Male    97  60
```

```
library(knitr)
```

```
kable(tab,
```

```
  col.names=c("N.-Raucher", "Raucher"))
```

	N.-Raucher	Raucher
Female	54	33
Male	97	60

Eine Variante mit relativen Häufigkeiten erhält man mit:

```
tally(sex ~ smoker, data=tips,
      format="proportion")
```

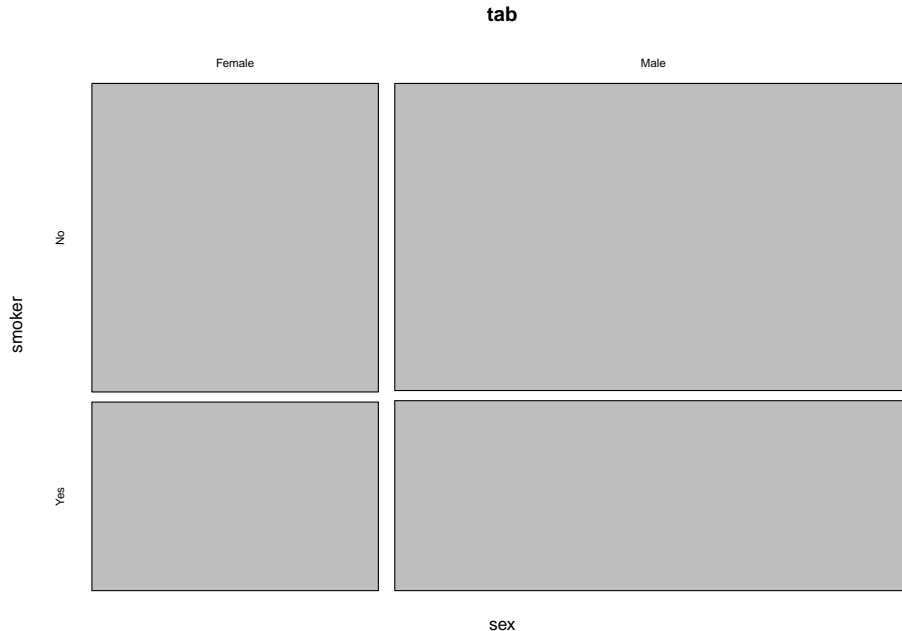
```
##           smoker
## sex      No    Yes
## Female 0.357616 0.354839
## Male   0.642384 0.645161
```

```
tally(sex ~ smoker, data=tips,
      format="percent")
```

```
##           smoker
## sex      No    Yes
## Female 35.7616 35.4839
## Male   64.2384 64.5161
```

Mit der Tabelle *tab* kann nun ein mosaic plot generiert werden:

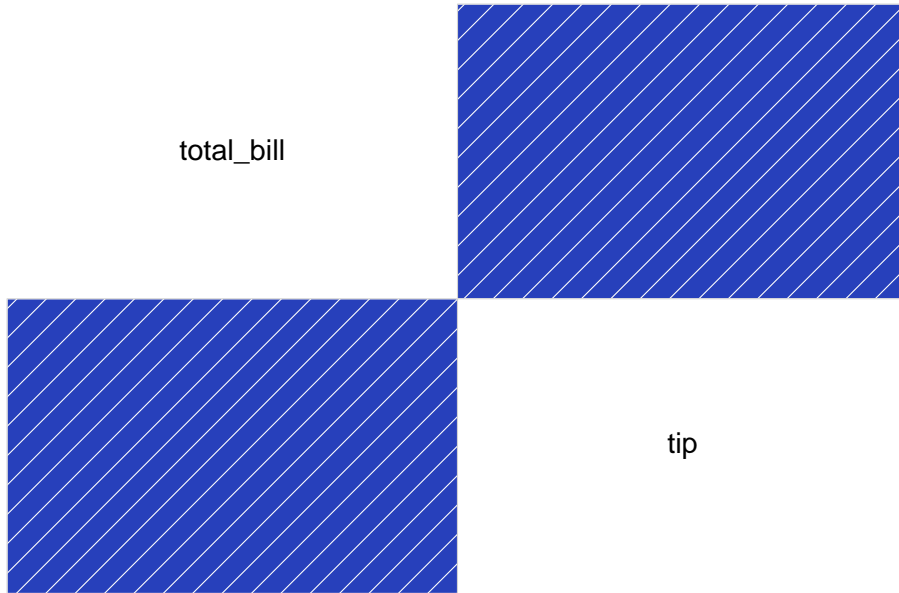
```
mosaicplot(tab)
```





# Korrelationsplot mit metrischen Variablen

```
# ggf: install.packages('corrgram', dependencies=T)
library(corrgram)
corrgram(tips)
```



Mittelwert

```
mean(tip ~ sex, data = tips)
```

```
## Female    Male  
## 2.83345 3.08962
```

Anstatt *mean* können alle Lageparameter und Streumaße errechnet werden (min, max, median, sd, var):

```
favstats(tip ~ sex, data = tips)
```

```
##      sex min Q1 median   Q3  max   mean      sd    n missing  
## 1 Female   1  2   2.75 3.50   6.5 2.83345 1.15949  87         0  
## 2 Male     1  2   3.00 3.76  10.0 3.08962 1.48910 157         0
```

Korrelation als Zusammenhangsmaß mit metrischen Variablen

```
cor(tip ~ total_bill, data = tips)
```

```
## [1] 0.675734
```

Test der Unabhängigkeit zweier nominalen Variablen mit Hilfe des  $\chi^2$ -Test:

```
xchisq.test(sex ~ smoker, data=tips)
```

```
## Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tally(x, data = data)
## X-squared = 0, df = 1, p-value = 1
##
##      54      33
## (53.84) (33.16)
## [0.00047] [0.00077]
## < 0.022> <-0.028>
##
##      97      60
## (97.16) (59.84)
## [0.00026] [0.00043]
## <-0.016> < 0.021>
##
## key:
## observed
## (expected)
## [continuity correction]
## <Pearson residual>
```

## 7. Eine kurze Datenanalyse

# Nullhypotestentests

---

Variablen müssen beide metrische sein und zwischen beiden Variablen wird eine Differenz gebildet.

Die Forschungsfrage lautet meist:

- ▶ V1 unterscheidet sich von V2 (ungerichtet)
- ▶  $V1 > V2$  (gerichtet)
- ▶  $V2 > V1$  (gerichtet)

```
t.test(~(tip - total_bill), data = tips)
```

```
## ~(tip - total_bill)
##
## One Sample t-test
##
## data:  tip
## t = -32.65, df = 243, p-value <2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -17.8006 -15.7748
## sample estimates:
## mean of x
##  -16.7877
```

Wenn die Forschungshypothese (Alternativhypothese) gerichtet ist, und  $V1-V2 < 0$  ist, dann wird das Argument `alternative="less"` hinzugefügt, wenn  $V1-V2 > 0$ , dann `"greater"`.

```
t.test ~(tip - total_bill), alternative = "less", data = tips)
```

```
## ~(tip - total_bill)
##
## One Sample t-test
##
## data: tip
## t = -32.65, df = 243, p-value <2e-16
## alternative hypothesis: true mean is less than 0
## 95 percent confidence interval:
##      -Inf -15.9386
## sample estimates:
## mean of x
## -16.7877
```

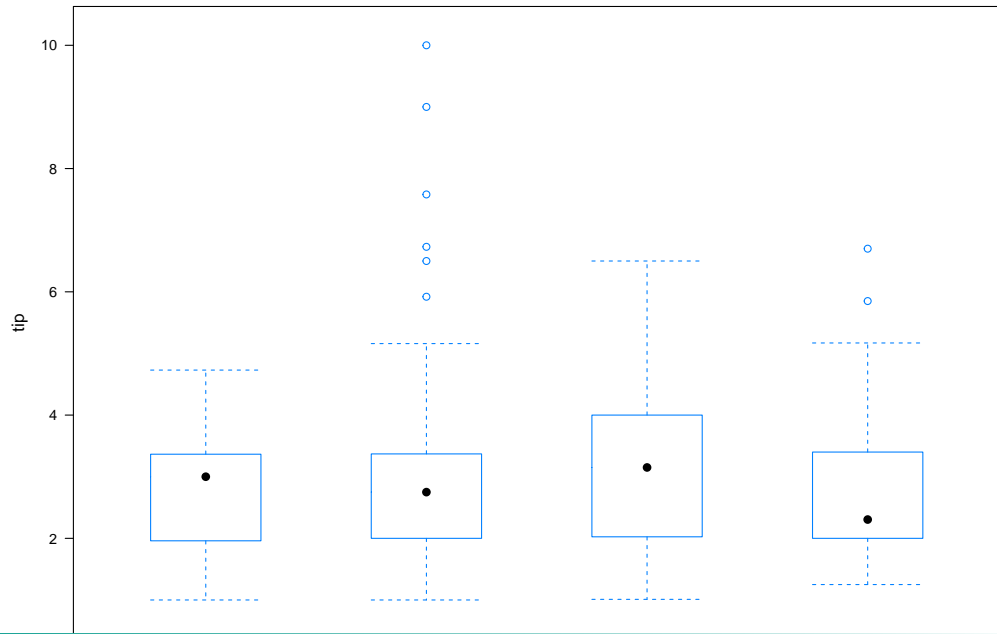
**Achtung:** Bei der Dokumentation von t-Tests ist es wichtig, einseitiges Testen von zweiseitigem Testen zu unterscheiden (einseitig/zweiseitig).

## 7. Eine kurze Datenanalyse

# ANOVA (Varianzanalyse)

Bezüglich einer Gruppe (nominale Variable) mit mehr als zwei Levels wird eine metrische Variable getestet.

```
bwplot(tip ~ day, data = tips)
```



## 7. Eine kurze Datenanalyse

# ANOVA (Varianzanalyse)

```
favstats(tip ~ day, data=tips)
```

##	day	min	Q1	median	Q3	max	mean	sd	n	missing
## 1	Fri	1.00	1.9600	3.000	3.3650	4.73	2.73474	1.01958	19	0
## 2	Sat	1.00	2.0000	2.750	3.3700	10.00	2.99310	1.63101	87	0
## 3	Sun	1.01	2.0375	3.150	4.0000	6.50	3.25513	1.23488	76	0
## 4	Thur	1.25	2.0000	2.305	3.3625	6.70	2.77145	1.24022	62	0

*Forschungshypothese:* Es gibt einen Unterschied beim Trinkgeld bei/zwischen den Tagen.

```
summary(aov(tip ~ day, data=tips))
```

##		Df	Sum Sq	Mean Sq	F value	Pr(>F)
## day		3	10	3.18	1.67	0.17
## Residuals		240	456	1.90		



Modellierung einer angängigen Variable (AV) durch eine unabhängige Variable (UV).

```
Mod1 <- lm( tip ~ total_bill, data=tips)
summary(Mod1)
```

```
##
## Call:
## lm(formula = tip ~ total_bill, data = tips)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.198 -0.565 -0.097  0.486  3.743
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.92027    0.15973     5.76 2.5e-08 ***
## total_bill   0.10502    0.00736    14.26 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.02 on 242 degrees of freedom
## Multiple R-squared:  0.457, Adjusted R-squared:  0.454
## F-statistic: 203 on 1 and 242 DF,  p-value: <2e-16
```

**Achtung:** Das nicht ausgegebene Level in der Ausgabe ist das Referenzlevel.

```
Mod2 <- lm(tip ~ day, data = tips)
summary(Mod2)
```

```
##
## Call:
## lm(formula = tip ~ day, data = tips)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.245 -0.993 -0.235  0.538  7.007
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.7347     0.3161   8.65 7.5e-16 ***
## daySat        0.2584     0.3489    0.74  0.46
## daySun        0.5204     0.3534    1.47  0.14
## dayThur       0.0367     0.3613    0.10  0.92
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.38 on 240 degrees of freedom
## Multiple R-Squared:  0.0051, Adjusted R-Squared:  0.0000
## F-statistic: 1.67 on 3 and 240 DF, p-value: 0.174
```

## 7. Eine kurze Datenanalyse

# Multiple Regression

```
Mod3 <- lm(tip ~ total_bill + sex + smoker + day + time + size, data = tips)
summary(Mod3)
```

```
## Call:
## lm(formula = tip ~ total_bill + sex + smoker + day +
## time + size,
## data = tips)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.849 -0.571 -0.103  0.470  4.134
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.86318    0.61168   1.41    0.16
## total_bill   0.09387    0.00975   9.63 <2e-16 ***
## sexMale     -0.03250    0.14350  -0.23    0.82
## smokerYes   -0.07710    0.14830  -0.52    0.60
## daySat      -0.11261    0.31411  -0.36    0.72
## daySun      -0.01248    0.32507  -0.04    0.97
## dayThur     -0.17351    0.40090  -0.43    0.67
## timeLunch    0.08158    0.45290   0.18    0.86
## size2       0.29568    0.53806   0.55    0.58
## size3       0.45547    0.57150   0.80    0.43
## size4       0.69172    0.59090   1.17    0.24
## size5       0.44743    0.73569   0.61    0.54
## size6       1.18049    0.79189   1.49    0.14
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'
## 0.1 ' ' 1
##
## Residual standard error: 1.03 on 231 degrees of freedom
## Multiple R-squared:  0.472, Adjusted R-squared:  0.445
## F-statistic: 17.2 on 12 and 231 DF, p-value: <2e-16
##
```

## 7. Eine kurze Datenanalyse

### Der Befehl *step*

Mit dem Befehl *step* führt man eine stufenweise Regressionsanalyse durch, bei der die Variablen nach der Reihenfolge ihrer Wichtigkeit entfernt werden.

*step*(Mod3)

```
## Start: AIC=27.56
## tip ~ total_bill + sex + smoker + day + time + size
##
##           Df Sum of Sq  RSS    AIC
## - day      3      0.65 246.2  22.21
## - size     5      5.02 250.6  22.49
## - time     1      0.03 245.6  25.59
## - sex      1      0.05 245.6  25.61
## - smoker   1      0.29 245.9  25.85
## <none>          245.6  27.56
## - total_bill 1     98.64 344.2 107.95
##
## Step: AIC=22.21
## tip ~ total_bill + sex + smoker + time + size
##
##           Df Sum of Sq  RSS    AIC
## - size     5      5.26 251.5  17.36
## - time     1      0.01 246.2  20.21
## - sex      1      0.04 246.3  20.24
## - smoker   1      0.30 246.5  20.51
## <none>          246.2  22.21
## - total_bill 1     98.74 345.0 102.48
##
## Step: AIC=17.36
## tip ~ total_bill + sex + smoker + time
##
##           Df Sum of Sq  RSS    AIC
## - smoker   1      1.27 252.7  16.60
## <none>          251.5  17.36
## - total_bill 1     204.87 456.3 160.76
##
## Step: AIC=15.37
## tip ~ total_bill + sex + smoker
##
##           Df Sum of Sq  RSS    AIC
## - sex      1      0.04 251.5  13.41
## - smoker   1      1.27 252.7  14.60
## <none>          251.5  15.37
## - total_bill 1     210.05 461.5 161.52
##
## Step: AIC=13.41
## tip ~ total_bill + smoker
##
##           Df Sum of Sq  RSS    AIC
## - smoker   1      1.27 252.8  12.63
## <none>          251.5  13.41
## - total_bill 1     213.67 465.2 161.45
##
## Step: AIC=12.63
## tip ~ total_bill
##
##           Df Sum of Sq  RSS    AIC
## <none>          252.8  12.63
```

## 8 R intern

Der *Arbeitsbereich* (**workspace**) in **R** ist eine Sammlung von Objekten, die aktuell im Speicher vorhanden ist.

Sie können diese Objekte anzeigen mit dem Befehl:

```
ls()
```

Löschen können sie die Objekte mit

```
rm()
```

- ▶ **R** ist ein **Paketen** organisiert.
- ▶ Ein *Paket* ist eine kompakte Zusammenfassung von Code, Hilfeseiten, Daten, Beispielen usw. zu einem bestimmten Themengebiet.
- ▶ **R** wächst durch ständig neue Pakete!
- ▶ Mit der Installation von **R** haben Sie eine Grundausstattung an wichtigen Paketen, welche Sie nach belieben erweitern können und sollten!
- ▶ Die wichtigsten Pakete gibt es beim [Comprehensive R Archive Network](#) kurz **CRAN**.

Um ein Paket, zum Beispiel **ggplot2**, zu installieren benutzt man den Befehl:

```
install.packages("ggplot2")
```

Um gleichzeitig weitere, notwendige Pakete zu installieren nutzt man die Option “dependencies=TRUE”:

```
install.packages("ggplot2", dependencies=TRUE)
```

Es gibt aber noch andere Wege! (Direkt in RStudio z.B.!)

**Aktualisieren von (allen) Paketen:**

```
update.packages()
```



- ▶ Mit dem Befehl **library()** (ohne Argument) werden alle bereits installierten Pakete aufgelistet. Nicht alle davon sind automatisch verfügbar, sondern müssen erst geladen werden

```
library(ggplot2)
```

- ▶ Liegt das Paket nicht im Standard-library-Verzeichnis, benutzt man die Option **lib.loc=**:

```
library(ggplot2, lib.loc= <Verzeichnis> )
```

- ▶ Alternativ können Pakete auch mit **require()** geladen werden. Diese Funktion liefert als Rückgabe die Information ob das Paket verfügbar ist oder nicht.

```
require(ggplot2)
```

```
[1] TRUE
```



## 9 Anhang

Wenn Sie sich mehr für **R** interessieren. Ein erster Anlaufpunkt wäre z. B. das Skript “Praxis der Datenanalyse” von

[https://sebastiansauer.github.io/Praxis\\_der\\_Datenanalyse/](https://sebastiansauer.github.io/Praxis_der_Datenanalyse/)

Sauer, S., Gansser, O. (2017). Praxis der Datenanalyse. Skript zum Modul im MSc.-Studiengang “Wirtschaftspsychologie & Consulting” an der FOM. FOM Nürnberg. DOI: 10.5281/zenodo.580649.

Ein gutes Werk in englischer Sprache finden Sie hier:

<http://r4ds.had.co.nz/>

Garrett Golemund und Hadley Wickham. R for Data Science. Published by O'Reilly January 2017 First Edition.

Hier finden Sie Videos, die einige Schritte der Datenaufbereitung und deskriptiver / explorativer Datenanalyse erläutern (zumeist mit R-Commander):

- ▶ boxplots erstellen <https://www.youtube.com/watch?v=9XBjOmA7sNs>
- ▶ Textdatei öffnen <https://www.youtube.com/watch?v=QnM9HBe23Y8>
- ▶ Öffnen der Datei Polizeistudie [https://www.youtube.com/watch?v=SDOoKuj5\\_7o](https://www.youtube.com/watch?v=SDOoKuj5_7o)
- ▶ SPSS Datei importieren [https://www.youtube.com/watch?v=HS8H\\_n7Vrm0](https://www.youtube.com/watch?v=HS8H_n7Vrm0)
- ▶ Deskriptive Statistik erstellen <https://www.youtube.com/watch?v=qrMpgk-7Wus>
- ▶ Variablen in Faktoren umwandeln und Balkendiagramm  
<https://www.youtube.com/watch?v=PRR-3kblt8k>
- ▶ Streudiagramm [https://www.youtube.com/watch?v=brE72\\_0stO0](https://www.youtube.com/watch?v=brE72_0stO0)
- ▶ Korrelationsmatrix [https://www.youtube.com/watch?v=pl92q\\_S-r8E](https://www.youtube.com/watch?v=pl92q_S-r8E)
- ▶ Datenmatrix erstellen <https://youtu.be/-EaeBL9J4IE>

Die Videos wurden von Frau Prof. Ferreira erstellt.

(C)opyright in 2016/2017 by Norman Markgraf

Diese Folien wurde von Norman Markgraf entwickelt und stehen unter der Lizenz CC-BY-SA-NC 3.0 de: <https://creativecommons.org/licenses/by-nc-sa/3.0/de/>.

- ▶ Datum erstellt: 2017-12-07
- ▶ R Version: 3.4.2
- ▶ mosaic Version: 1.1.0

Diese Präsentation wurde mit **RMarkdown** und ein paar kleinen Helferlein (**NPBT**) erstellt.

Sie finden die (tages-)aktuelle Version, inklusive der Quellen, unter:

<https://github.com/NMarkgraf/Etwas-R-am-Abend>

Bitte melden Sie Fehler und Verbesserungsvorschläge an: [nmarkgraf@hotmail.com](mailto:nmarkgraf@hotmail.com)

**Meinen Dank** an Oliver Gansser, Matthias Gehrke, Karsten Lübke, Chistian Schwarz und Sebastian Sauer für Ideen, Tipps und Unterstützung bei der Skripterstellung!