Etwas R zum Nachmittag

Standort Köln – 18.01.2017

Norman Markgraf

Was ist eigentlich R?

Programmiersprache S:

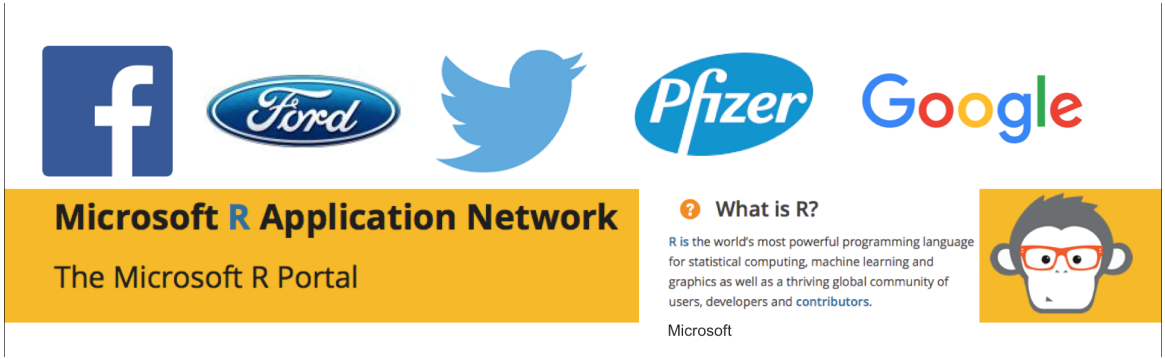
- Von Bell Labs für Statistik, Simulation, Grafik entwickelt (Becker and Chambers; 1984)
- kommerzielle Implementation: **S-PLUS**

Programmiersprache R:

- Implementation unter GPL (GNU General Public License), offener Quellcode
- **Vorteile:**
  - interpretierter Programmcode, objektorientiert
  - leicht erweiterbar durch eigene Routinen, Pakete, DLLs
  - viele Grafikmöglichkeiten
  - standardisiertes, einfach handhabbares Datenformat (data.frame)
  - gut durchdachtes Format zur Anpassung von (Regressions-)Modellen
  - aktive Entwicklergruppe, hilfreiche Mailingliste
  - modularer Aufbau mit mehr als 8000 Erweiterungspaketen
  - man kann ansprechende Diagramme und interaktive Apps entwickeln (z.B. plotly, shiny).
  - führende Plattform für statistische Analysen
- **Nachteile:**
  - bisher keine echte "Standard"-GUI (aber es gibt ja RStudio)
  - verfügbare Routinen/Pakete manchmal unübersichtlich

Wer nutzt R im echten Leben?

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



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

Falls Sie gerne **Werbevideos** ansehen, hier ein Link
https://www.youtube.com/watch?v=TR2bHSJ_eck

Einfach nur R oder darf es etwas mehr sein?

R ist ein eine *komandozeilenorientierte*-(Programmier-)Sprache!

```
1+1
```

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

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

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

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

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

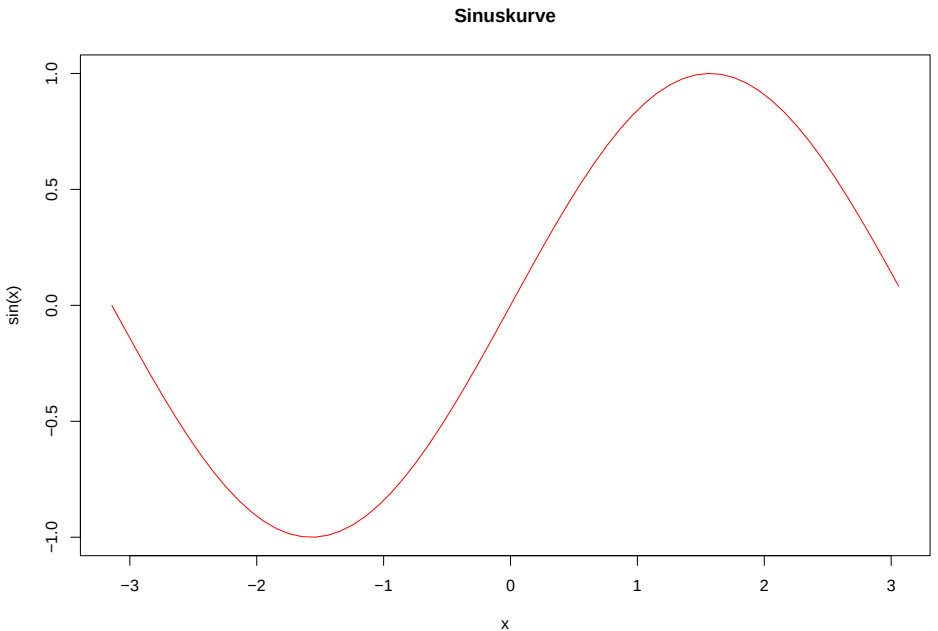
```
x = 1; y = 2; print(x+y)
```

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

Einfach nur R oder darf es etwas mehr sein?

Die mit unter recht schnell schöne Ergebnisse produzieren kann:

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



Einfach nur R oder darf es etwas mehr sein?

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 Entwicklungsumgebenung (*IDE*), wie z.B.

- RStudio
- Rcmdr
- StatET for R eine auf Eclipse basierende IDE für R
- ESS ein add-on package für GNU Emacs und XEmacs

Empfehlung:

- **R** (Version 3.3.2) ,
  - **R** finden Sie hier: `https://cran.rstudio.com` oder `https://www.r-project.org`
  - Aktuell ist die Version 3.3.2
  - **Achtung MAC-Nuzter!!!**: Sie benötigen zusätlich erst noch XQuartz.
    - XQuartz finden Sie hier: `https://www.xquartz.org`
- **RStudio** (Desktop-Version 1.0.136) (vergessen Sie lieber den **R-Cmdr**)
  - Die aktuelle Version finden Sie hier: `https://www.rstudio.com/products/rstudio/download/`
  - Oder die tagesaktuelle Entwicklerversion von hier: `https://dailies.rstudio.com`

Die Installation

- Die wichtigsten Schritte bei der Installation:
  - Abwarten und bestätigen ;-)

Die wichtigsten Pakete und wie man diese installiert

Im Allgemeinen instaliert man ein Paket durch den Befehl:

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

Für einen guten Start sollte man folgende Pakete installieren:

- **tidyverse**
  tidyverse ist eine Sammlung von Paketen, die einem den Umgang mit **R** und *Grafik* erleichtern. Das sind u.a. die Pakete:
  - ggplot2 # DAS Grafikpaket von *R*
  - dplyr # Das Paket zur Daten manipulation
  - readr # Das Paket zum Einlesen von Daten
- **mosaic**
  Mehr Informationen zu **mosaic** finden Sie hier:
  - Project MOSAIC
  - Less Volume, More Creativity – Getting Started with the mosaic Package
    ## Spielmaterial für den Anfang
- *Daten* zum Experimentieren und Spielen und ("**R**"-)Forschen:
  Prof. Dr. Sebastian Sauer hat auf *GitHub* eine kleine Sammlung von Daten zusammengestellt. Diese werden u.a. im Fach "Datenerhebung und Statistik" genutzt und sind ein guter Ausgangspunkt um ein paar Schritt in **R** selber zu gehen. Sie finden die Daten unter:
  https://github.com/sebastiansauer/Daten_Unterricht
  Entzippen Sie den herunter geladenen Ordner. Darin finden Sie die hier verwendeten Datensätze.

Die wichtigsten Pakete und wie man diese installiert

Die ersten Befehle sollten wie folgt lauten:

```
# Laden von tidyverse Paketen:
install.packages("tidyverse", dependencies=TRUE)
# Laden des mosaic Pakets:
install.packages("mosaic", dependencies=TRUE)
# Laden des corrgram Pakets:
install.packages("corrgram", dependencies = T)
```

Für die Eingabe in der Konsole reicht es, die Zeilen OHNE ein '#' am Anfang einzutippen. Mit '#' leitet man einen Kommentar ein. In der Konsole reicht daher:

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

Bitte bestätigen Sie alle Anfragen und haben Sie etwas Geduld. Es wird eine Menge nachgeladen.

Aber nur einmal. Also keine Sorge!

Bemerkung	Umsetzung in **R**
Grundrechenarten	+ - * /
Potenzieren	^
logische Operatoren	== != < > <= >=
Funktionen	cos() sin() tan() exp() sqrt()
Konstante	pi
Kommentare	#
Dezimalzeichen	.

```r
# Punkt- vor Strichrechnung
2 * 3 + 2 - 25/5 + 2^3
```

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

```r
# Trigometrische Funktionen
cos(pi/2)^2 + sin(pi/2)^2
```

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

```r
# Logarithmen & Exponetialfunktion
log(exp(3))
```

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

```r
# Unendlich
1/0
```

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

```r
# Not a Number (keine Zahl)
0/0
```

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

```r
# Not Available; ein fehleder Wert
NA
```

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

```r
# Vektoren (combine)
c(1, 4:8)
```

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

```r
# Vektor/Liste ohne Inhalt
c()
```

```
## NULL
```

Variablen

- 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-sensetiv**, 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
```

Datentypen

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"
```

Vektoren

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

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

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

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

```
vektorMitBeliebigemNamen * 2
```

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

```
vektorMitBeliebigemNamen + 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 = 10)
```

```
##  [1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5
```

```
vektor <- 1:4 # 'n:m' entspricht seq(from=n, to=m, by=1)
```

Werte können mit **rep()** wiederholt werden:

```
rep("X", times = 5) # wiederholt 'X' 5-mal
```

```
## [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
```

Logische Abfragen

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

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

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

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

```
vektorMitBeliebigemNamen != 0
```

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

```
logischerVektor <- vektorMitBeliebigemNamen <= 3
logischerVektor
```

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

Eigenschaften von Vektoren

names(a) gibt die Namen der Einträge des Vaktors a zurück:

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

```
## NULL
```

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

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

- **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)
mean(aVec)
```

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

```
sd(aVec)
```

```
## [1] 3.559026
```

```
var(aVec)
```

```
## [1] 12.66667
```

```
min(aVec)
```

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

```
max(aVec)
```

```
## [1] 9
```

```
length(aVec)
```

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

```
sum(aVec)
```

```
## [1] 16
```

Rechnen mit Vektoren

```
median(aVec)
```

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

```
length(aVec)
```

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

```
sum(aVec)
```

```
## [1] 16
```

```
median(aVec)
```

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

```
IQR(aVec)
```

```
## [1] 3.5
```

```
summary(aVec)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    1.75    3.00    4.00    5.25    9.00
```

```
library(mosaic)
favstats(aVec)
```

```
##  min   Q1 median   Q3 max mean       sd n missing
##    1 1.75      3 5.25   9    4 3.559026 4       0
```

R berechnent die Varianz von Daten mit Hilfe der Formel

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

wie man leicht nachrechnen kann:

```
var(aVec)
```

```
## [1] 12.66667
```

```
# Ist das selbe wie
1/(length(aVec)-1) * sum( (aVec-mean(aVec))^2 )
```

```
## [1] 12.66667
```

```
# Dagegen ist
1/length(aVec) * sum( (aVec-mean(aVec))^2 )
```

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

Standardabweichung

Die Standardabweichung ist die Quadratwurzel der Varianz:

```
sd(aVec)
```

```
## [1] 3.559026
```

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

```
## [1] 3.559026
```

Varianz

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:

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

```
## [1] 12.66667
```

```
# Korrigierter Wert
factor*var(aVec)
```

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

```
# Zur Probe:
1/length(aVec) * sum( (aVec-mean(aVec))^2 )
```

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

Standardabweichung

```
factorSD <-sqrt((length(aVec)-1)/(length(aVec)))
# Wert von R:
sd(aVec)
```

```
## [1] 3.559026
```

```
# Korrigierter Wert
factor*sd(aVec)
```

```
## [1] 2.66927
```

```
# Zur Probe
sqrt(1/length(aVec) * sum( (aVec-mean(aVec))^2 ))
```

```
## [1] 3.082207
```

Rechnen mit Vektoren

```
aVec2 <- rep(2, 4)
aVec
```

```
## [1] 1 2 4 9
```

```
aVec2
```

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

```
aVec %*% aVec2
```

```
##      [,1]
## [1,]   32
```

```
aVec * aVec2
```

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

```
aVec3 <- aVec
aVec3[3]
```

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

```
aVec3[3] <- NA
aVec3
```

```
## [1]  1  2 NA  9
```

```
mean(aVec3)
```

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

```
mean(aVec3, na.rm = TRUE)
```

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

Der Arbeitsbereich in R

Der **workspace** (*Arbeitsbereich*) 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()
```

Pakete

- **R** ist in **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 *C*omprehensive *R* *A*rchive *N*etwork kurz **CRAN**.

Pakete installieren und aktualisieren

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

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

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

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

Mit der Funktion

```
update.packages()
```

werden installierte Pakete mit denen von CRAN hinterlegten verglichen und ggf. akualisiert.

Es gibt aber noch andere Wege. (Z.B. direkt in RStudio)

Pakete benutzen

- 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
```

Wir laden ein paar Daten

- Via **RStudio**:
  Gehen Sie auf das recht obere Fenster und klicken Sie **Import Dataset**, danach **From CSV...** und geben Sie als URL bitte https://raw.githubusercontent.com/NMarkgraf/Etwas-R-zum-Nachmittag/master/Datasets/miete03.asc ein.
  Drücken Sie **Update**.
  Stellen Sie *Delimiter* auf *Tab*.
  Drücken Sie dann **Import**
- Via **R** direkt: Man kann auch direkt aus **R** mittels ein paar Zeilen die Daten laden!
  Die selben Daten können Sie u.a. durch die Zeilen

```
miete03 <- read.table(
   file = paste0("https://raw.githubusercontent.com/NMarkgraf/",
                 "Etwas-R-zum-Nachmittag/master/Datasets/miete03.asc"),
   header = TRUE)
```

aus dem Netz laden.

Die ersten Zeilen der Tabelle ansehen

Mit dem Befehl **head()** schaut man sich die ersten Zeilen (im Bsp. die ersten 4 Zeilen) eines *Dataframes* an:

```
head(miete03, 4)
```

```
##       nm  nmqm wfl rooms   bj bez wohngut wohnbest ww0 zh0 badkach0
## 1 741.39 10.90  68     2 1918   2       1        0   0   0        0
## 2 715.82 11.01  65     2 1995   2       1        0   0   0        0
## 3 528.25  8.38  63     3 1918   2       1        0   0   0        0
## 4 553.99  8.52  65     3 1983  16       0        0   0   0        0
##   badextra kueche
## 1        0      0
## 2        0      0
## 3        0      0
## 4        1      0
```

Die letzten Zeilen der Tabelle ansehen

Mit dem Befehl **tail()** schaut man sich die ersten Zeilen (im Bsp. die letzen 3 Zeilen) eines *Dataframes* an:

```
tail(miete03, 3)
```

```
##           nm nmqm wfl rooms   bj bez wohngut wohnbest ww0 zh0 badkach0
## 2051 567.54 8.11  70     3 1973  16       0        0   0   0        0
## 2052 323.42 9.24  35     1 1970  21       0        0   0   0        0
## 2053 506.19 7.79  65     3 1966   7       0        0   0   0        1
##      badextra kueche
## 2051        0      0
## 2052        0      0
## 2053        0      0
```

Mit dem Befehl **table** können wir eine *(absolute) Häufigkeitstabelle* erstellen:

```
table(miete03$rooms)
```

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

Eine *relative Häufigkeitstabelle* erhält man durch:

```
prop.table(table(miete03$rooms))
```

```
##
##           1           2           3           4           5           6
## 0.124208475 0.348270823 0.369702874 0.128105212 0.022893327 0.006819289
```

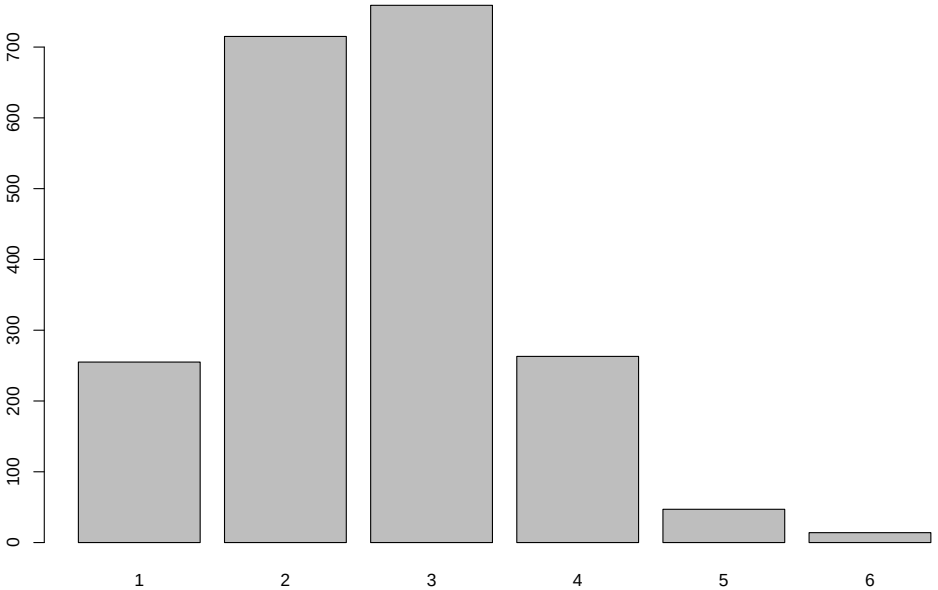
Ein anderer Weg ... mit dem Paket *dplyr*

```
library(dplyr)
miete03 %>% group_by(rooms) %>% tally(sort=TRUE)
```

```
## # A tibble: 6 × 2
##   rooms     n
##   <int> <int>
## 1     3   759
## 2     2   715
## 3     4   263
## 4     1   255
## 5     5    47
## 6     6    14
```

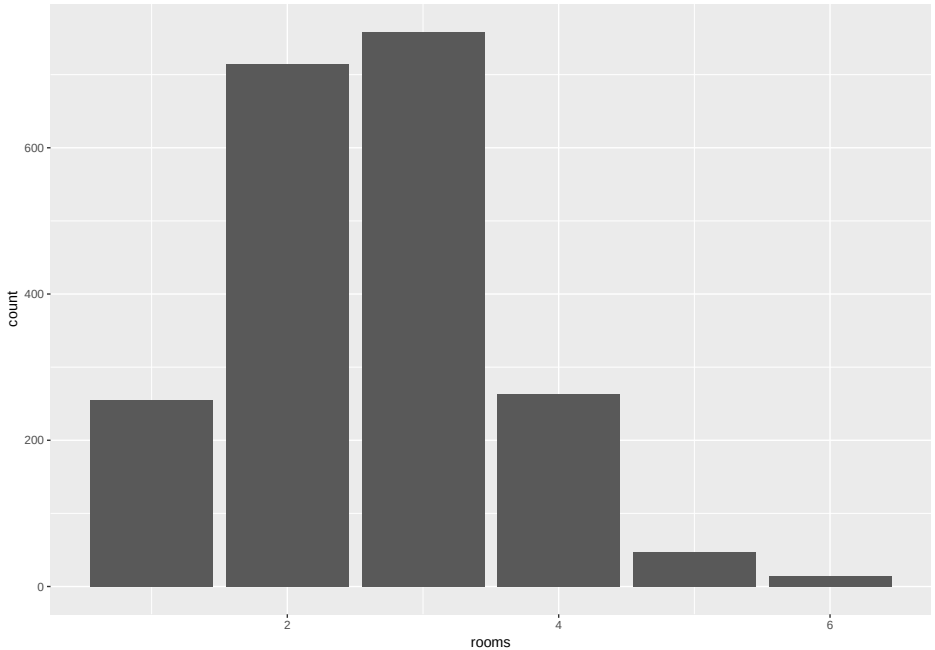
Und mit dem Befehl **barplot()** erstellen wir ein Balkendiagramm daraus:

```
barplot( table(miete03$rooms) )
```



Schönere Diagramme mit **ggplot2**

```
library(ggplot2) # Bibliothek laden!
ggplot(miete03, aes(x = rooms)) + geom_bar()
```



Mehr Informationen zu **ggplot2**

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

Mehr Informationen zu **R**

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
- SPSS Datei importieren 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_0st00
- Korrelationsmatrix https://www.youtube.com/watch?v=pl92q_S-r8E
- Datenmatrix erstellen https://youtu.be/-EaeBL9J4IE

Die Videos wurden von Frau Prof. Ferreira erstellt.

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

Dazu nehmen wir die Daten *tips* aus dem Unterrichtsmaterial von Prof. Dr. S. Sauer.

```
library(readr)
tips <- read_csv(file= paste0("https://raw.githubusercontent.com/sebastiansauer/",
                              "Daten_Unterricht/master/tips.csv"))
tips$X1 <- NULL
```

Darüberhinaus nutzen wir einige Befehle aus dem Paket **Mosaic**, welches wir dazu laden

```
library(mosaic)
```

Daten ansehen

Schauen wir uns die Daten etwas an:

```
glimpse(tips)
```

```
## Observations: 244
## Variables: 7
## $ total_bill <dbl> 16.99, 10.34, 21.01, 23.68, 24.59, 25.29, 8.77, 26....
## $ tip        <dbl> 1.01, 1.66, 3.50, 3.31, 3.61, 4.71, 2.00, 3.12, 1.9...
## $ sex        <chr> "Female", "Male", "Male", "Male", "Female", "Male",...
## $ smoker     <chr> "No", "No", "No", "No", "No", "No", "No", "No", "No...
## $ day        <chr> "Sun", "Sun", "Sun", "Sun", "Sun", "Sun", "Sun", "S...
## $ time       <chr> "Dinner", "Dinner", "Dinner", "Dinner", "Dinner", "...
## $ size       <int> 2, 3, 3, 2, 4, 4, 2, 4, 2, 2, 2, 4, 2, 4, 2, 2, 3, ...
```

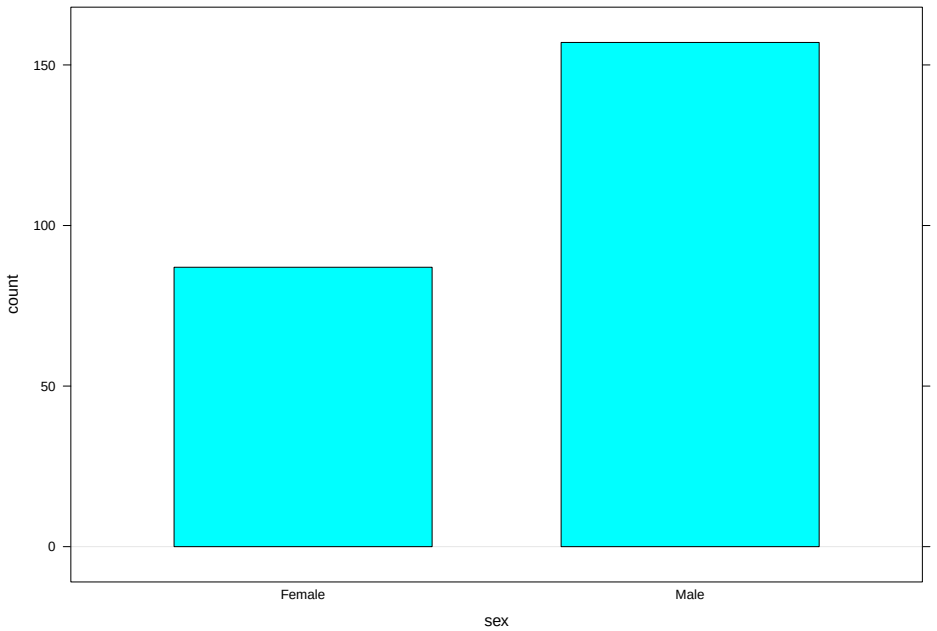
Dimension des Datensatzes:

```
dim(tips)
```

```
## [1] 244   7
```

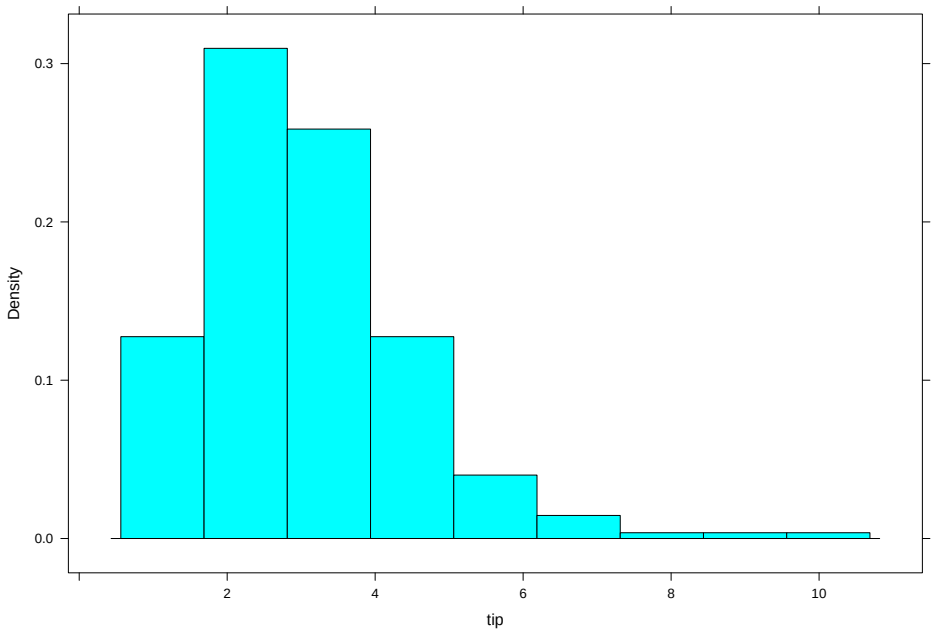
Balkendiagramm bei kategorialen Daten

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



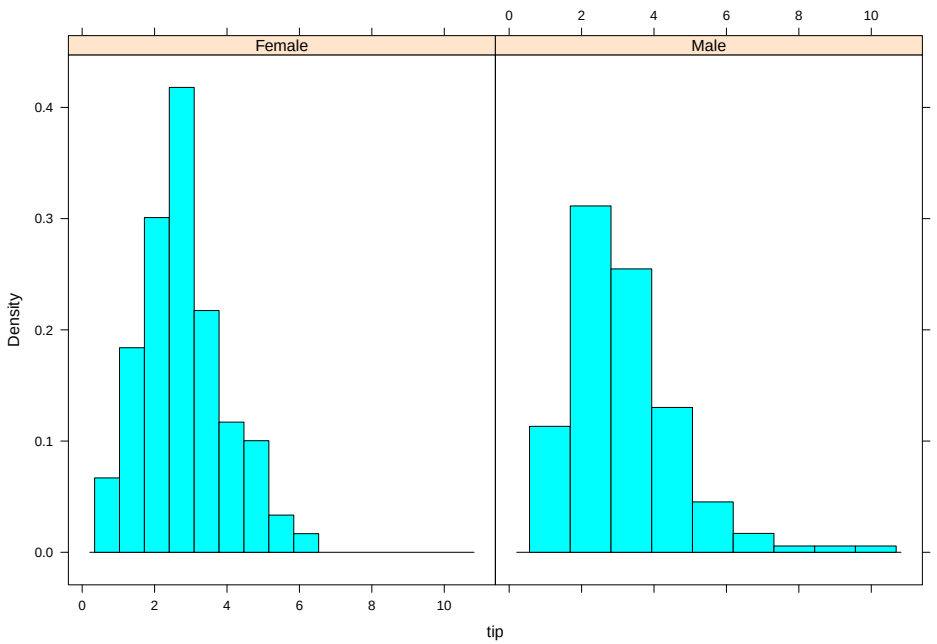
Histogramm bei metrischen Daten

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



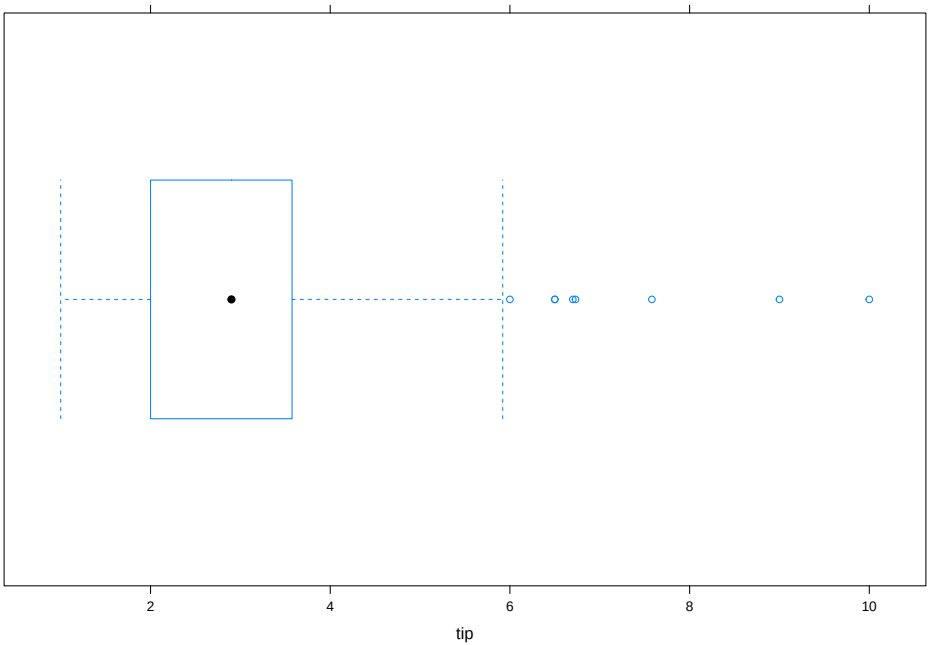
Histogramm bei metrischen Daten, facettiert

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



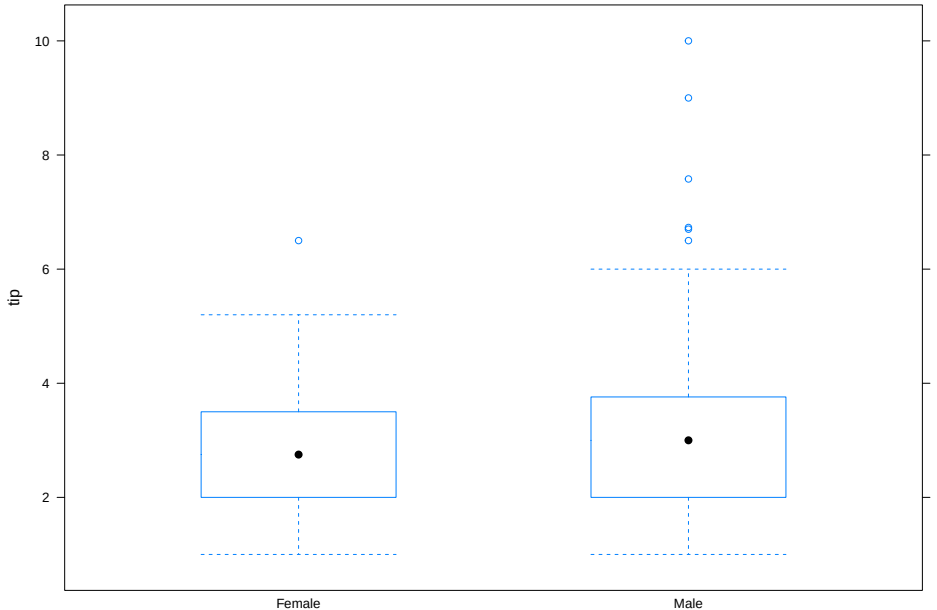
Boxplots bei merischen Daten

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



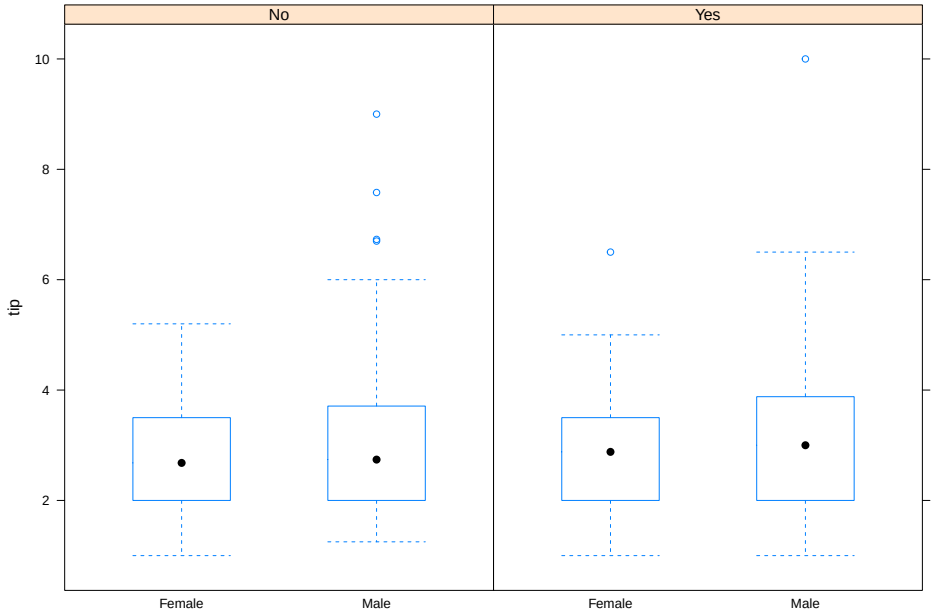
Boxplot mit metrischen Daten für Gruppem

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



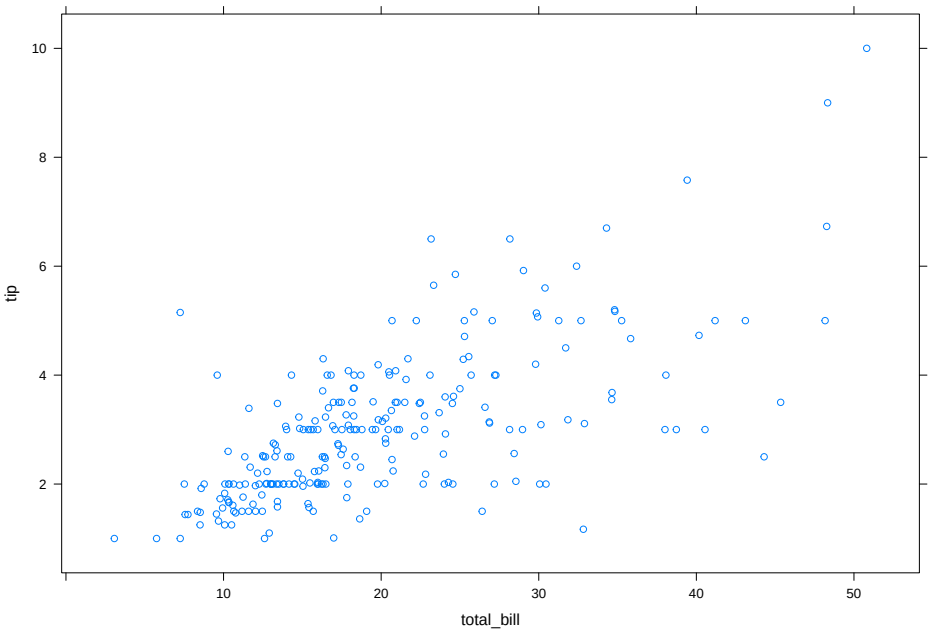
Boxplot mit metrischen Daten für Gruppem, facettiert

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



Streudiagramm mit zwei metrischen Variablen

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



Häufigkeitstabellen zwei kategorialen Variablen

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
```

Eine Variante mit relativen Häufigkeiten erhält man mit:

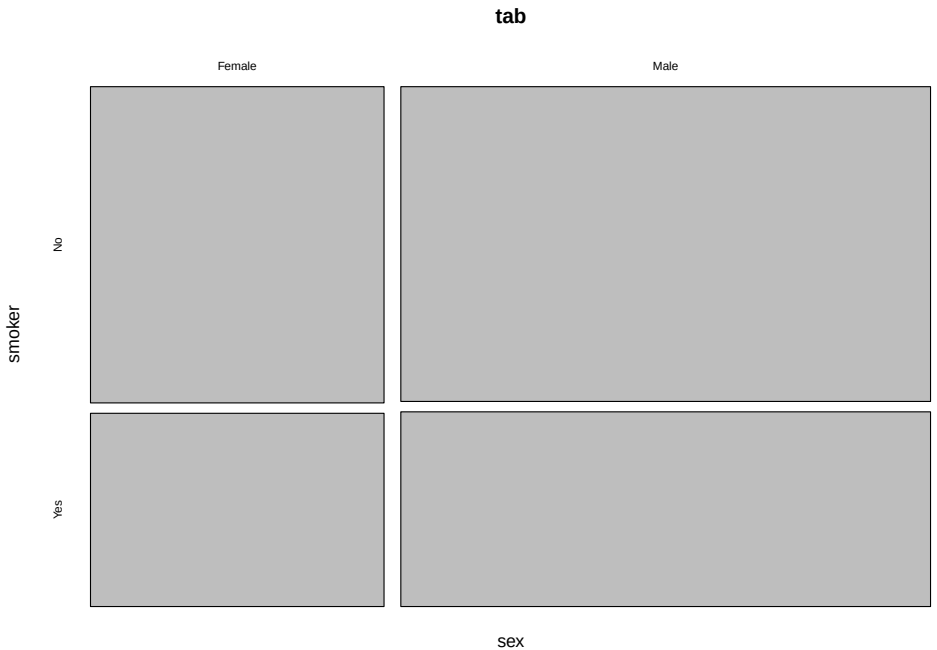
```
tally(sex ~ smoker, format="proportion", data=tips)
```

```
##         smoker
## sex             No       Yes
##   Female 0.3576159 0.3548387
##   Male   0.6423841 0.6451613
```

mosaicplot mit zwei kategorialen Variablen

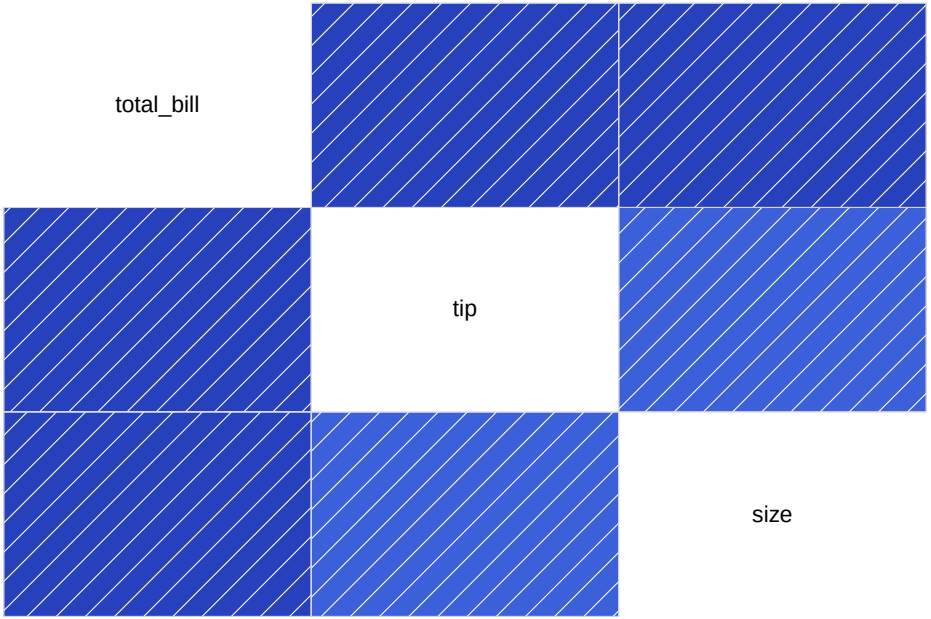
Mit der Tabelle *tab* kann nun ein mosaic plot generiert werden:

```
mosaicplot(tab)
```



Korrelationsplot mit metrischen Variablen

```
# ggf:  install.packages("corrgram", dependencies=T)
library(corrgram)
corrgram(tips)
```



Kennzahlen

Mittelwert

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

```
##   Female     Male
## 2.833448 3.089618
```

Anstatt *mean* können alle Lageparameter und Streumaße erechnet 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 2.833448 1.159495  87       0
## 2   Male   1  2   3.00 3.76 10.0 3.089618 1.489102 157       0
```

Korrelation als Zusammenhangsmaß mit metrischen Variablen

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

```
## [1] 0.6757341
```

χ^2-Test

Test der Unabhängigkeit geht nur mit zwei nominalen Variablen. In *tab* haben wir solche schon generiert.

```
xchisq.test(tab)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  x
## 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)
##  [contribution to X-squared]
##  <Pearson residual>
```

t-Test für abhängige Stichproben (Differenzentest)

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)
```

```
##
##  One Sample t-test
##
## data:  tips$(tip - total_bill)
## t = -32.647, df = 243, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -17.80057 -15.77476
## sample estimates:
## mean of x
## -16.78766
```

t-Test für abhängige Stichproben (Differenzentest)

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)
```

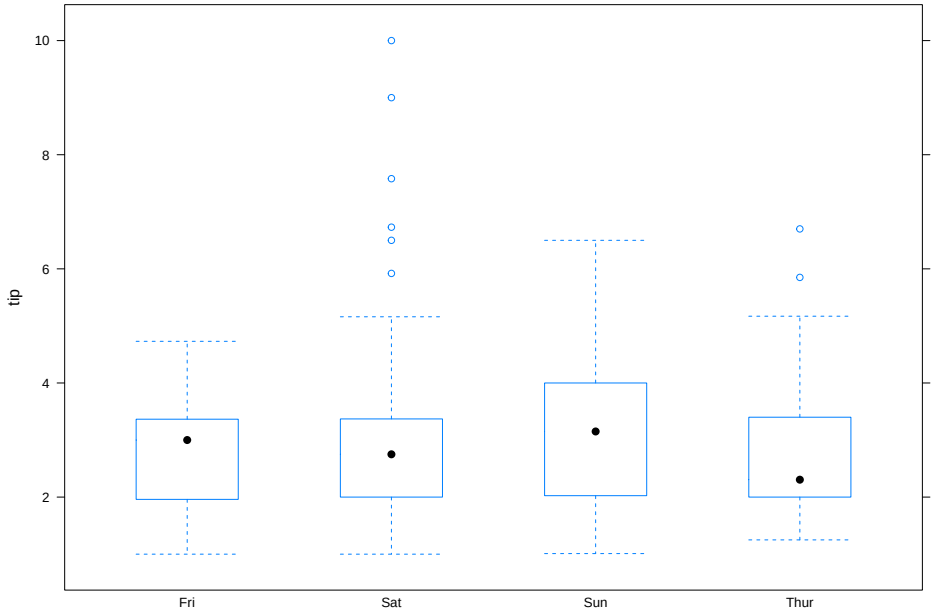
```
##
##  One Sample t-test
##
## data:  tips$(tip - total_bill)
## t = -32.647, df = 243, p-value < 2.2e-16
## alternative hypothesis: true mean is less than 0
## 95 percent confidence interval:
##      -Inf -15.9386
## sample estimates:
## mean of x
## -16.78766
```

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

ANOVA (Varianzanalyse)

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

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



```
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 2.734737 1.019577 19       0
## 2  Sat 1.00 2.0000  2.750 3.3700 10.00 2.993103 1.631014 87       0
## 3  Sun 1.01 2.0375  3.150 4.0000  6.50 3.255132 1.234880 76       0
## 4 Thur 1.25 2.0000  2.305 3.3625  6.70 2.771452 1.240223 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    9.5   3.175   1.672  0.174
## Residuals   240  455.7   1.899
```

Lineare Einfachregression mit metrischer Variable.

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
## -3.1982 -0.5652 -0.0974  0.4863  3.7434
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.920270   0.159735   5.761 2.53e-08 ***
## total_bill  0.105025   0.007365  14.260  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.022 on 242 degrees of freedom
## Multiple R-squared:  0.4566, Adjusted R-squared:  0.4544
## F-statistic: 203.4 on 1 and 242 DF,  p-value: < 2.2e-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
## -2.2451 -0.9931 -0.2347  0.5382  7.0069
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.73474    0.31612   8.651 7.46e-16 ***
## daySat       0.25837    0.34893   0.740    0.460
## daySun       0.52039    0.35343   1.472    0.142
## dayThur      0.03671    0.36132   0.102    0.919
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.378 on 240 degrees of freedom
## Multiple R-squared:  0.02048,    Adjusted R-squared:  0.008232
## F-statistic: 1.672 on 3 and 240 DF,  p-value: 0.1736
```

```
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
## -2.8475 -0.5729 -0.1026  0.4756  4.1076
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.803817   0.352702   2.279   0.0236 *
## total_bill   0.094487   0.009601   9.841   <2e-16 ***
## sexMale     -0.032441   0.141612  -0.229   0.8190
## smokerYes   -0.086408   0.146587  -0.589   0.5561
## daySat      -0.121458   0.309742  -0.392   0.6953
## daySun      -0.025481   0.321298  -0.079   0.9369
## dayThur     -0.162259   0.393405  -0.412   0.6804
## timeLunch    0.068129   0.444617   0.153   0.8783
## size         0.175992   0.089528   1.966   0.0505 .
## ---
```

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=20.51
## tip ~ total_bill + sex + smoker + day + time + size
##
##              Df Sum of Sq    RSS     AIC
## - day         3     0.609 247.14  15.116
## - time        1     0.025 246.55  18.538
## - sex         1     0.055 246.58  18.568
## - smoker      1     0.365 246.89  18.874
## <none>                    246.53  20.513
## - size        1     4.054 250.58  22.493
## - total_bill  1   101.595 348.12 102.713
##
## Step:  AIC=15.12
## tip ~ total_bill + sex + smoker + time + size
##
##              Df Sum of Sq    RSS    AIC
## - time        1     0.001 247.14 13.117
## - sex         1     0.042 247.18 13.157
## - smoker      1     0.380 247.52 13.490
## <none>                    247.14 15.116
## - size        1     4.341 251.48 17.365
```

Wie geht es weiter?

Dieses kleine Kick-Off für **R** ist natürlich nicht vollsätndig. - Wo es lohnt aus meiner persönlichen Sicht lohnt weiter zu schauen sind die folgenden Themen:

- **Grammar of Graphic** und **ggplot2** für schönere und aussagekräftige Grafiken in **R**.
- *tidyr*, *dplyr* und Co, die Welt des Datenmanagement in **R**
- *rmarkdown* und der Weg zu eigenen Dokumenten
- *Shiny* und die interaktive Webdarstellung von Statistiken
- *mosaic*

Quellen

Ein großer Teil der Beispiele stammt von **Prof. Dr. Oliver Gansser** aus seinem Handout *"Wichtige Befehle in R - Datenerhebung und Statistik"* vom 15. Dezember 2016, welches mir von **Prod. Dr. Joachim Schwarz** freundlicherweise überlassen wurde.

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

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

https://github.com/NMarkgraf/Etwas-R-zum-Nachmittag

Erstellt wurde dieses Dokument mit **R** (Version 3.3.2) und RStudio (1.0.136) am 19. Jan 2017