

# **Etwas R zum Nachmittag**

Standort Köln – 18.01.2017

**Norman Markgraf**

## 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

## Programmiersprache R:

### ► Vorteile (Forts.):

- 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.



**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)

# Einfach nur R oder darf es etwas mehr sein?

R ist eine *komandozeilenorientierte*-Sprache!

```
1+1
```

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

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

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

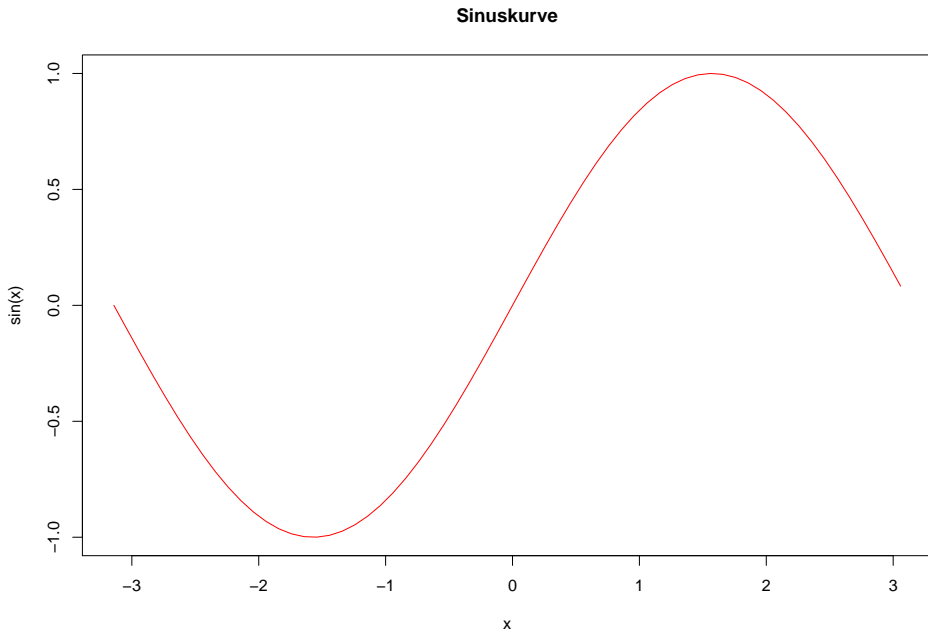
```
x <- 1; y <- 2  
x+y
```

```
## [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 Entwicklungsumgebungen (*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

# Was bekommt man wo und wie?

## Empfehlung:

- ▶ **R** (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-Nutzer!!!**: Sie benötigen zusätzlich 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, für mutige, die tagesaktuelle Entwicklerversion von hier: <https://dailies.rstudio.com>



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

# Die wichtigsten Pakete und wie man diese installiert

Im Allgemeinen installiert 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](#)

- ▶ *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](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)
```

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)
```

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	.

# Die ersten Schritte

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

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

```
# Trigonometrische 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
```

- ▶ 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
```

# 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"
```



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 = 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)
```

# Sequenzen

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
```

```
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
```

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

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

# Rechnen mit Vektoren

- ▶ **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
```

```
median(aVec)
```

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

```
median(aVec)
```

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

```
length(aVec)
```

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

```
IQR(aVec)
```

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

```
sum(aVec)
```

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

```
summary(aVec)
```

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

R berechnet 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
```



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
```

```
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 **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()
```

- ▶ **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 [Comprehensive R Archive Network](#) kurz **CRAN**.

# Pakete installieren und aktualisieren

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)
```

Mit der Funktion

```
update.packages()
```

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

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

- ▶ 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 letzten 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
```

# Häufigkeitstabelle und Balkendiagramme

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
```

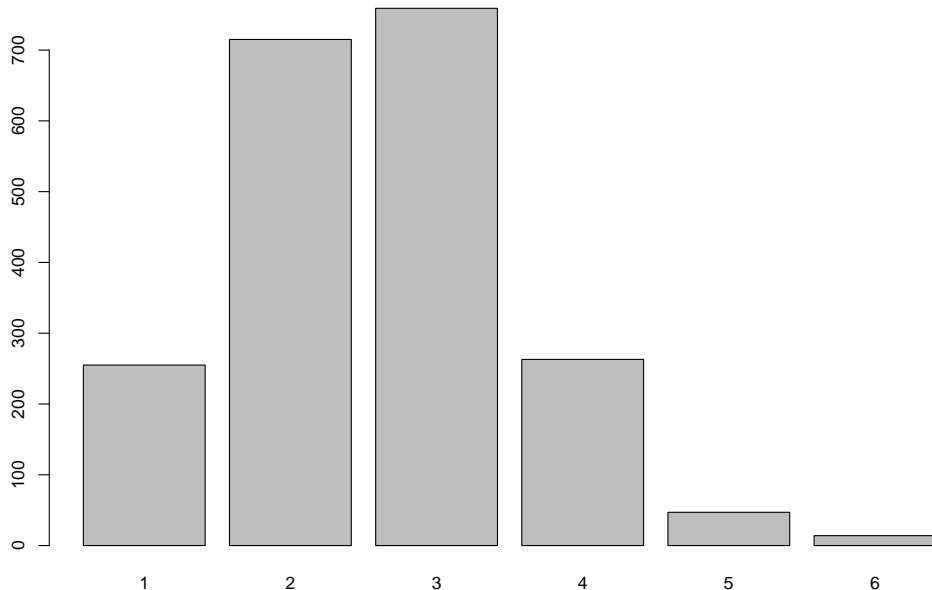
```
library(dplyr)
miete03 %>% group_by(rooms) %>% tally(sort=TRUE)
```

```
## # 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
```

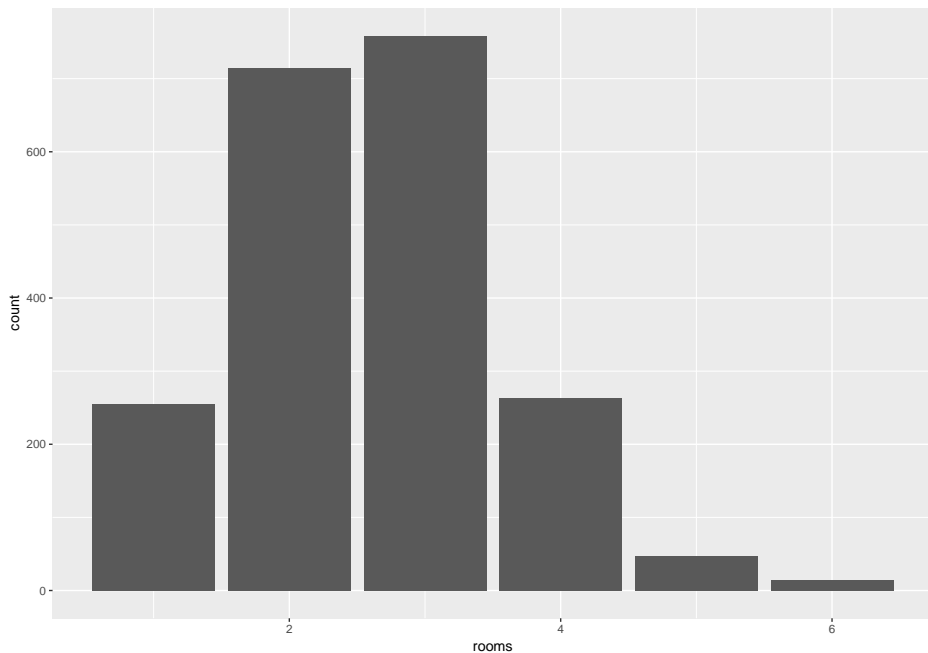
# Häufigkeitstabelle und Balkendiagramme

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

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



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



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>



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=9XBj0mA7sNs>
- ▶ Textdatei öffnen <https://www.youtube.com/watch?v=QnM9HBe23Y8>
- ▶ Öffnen der Datei Polizeistudie [https://www.youtube.com/watch?v=SD0oKuj5\\_7o](https://www.youtube.com/watch?v=SD0oKuj5_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\\_0st00](https://www.youtube.com/watch?v=brE72_0st00)
- ▶ 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.

