Eine Einführung in die statistische Datenanalyse mit R

Bernd Weiß, M. A.

Forschungsinstitut für Soziologie
Universität zu Köln
bernd.weiss@wiso.uni-koeln.de

Übersicht

Übersicht

Voraussetzungen

1. Kenntnisse von Statistik I („Deskriptive Statistik“).
2. Erfahrungen mit einem anderen Statistikprogramm (zumindest die Idee eines Statistikprogramms verstanden haben).
3. Wissen, dass Computer auch mit einer Tastatur bedient werden können.
4. Download der Daten und des Script-Files:
   <`http://www.metaanalyse.de/tmp/dataMannheim.zip`>
   bzw. dataAPIM-SpringS-Mannheim.por (.sav,.dta), danach Bindestriche durch Unterstriche ersetzen)
   <`http://www.metaanalyse.de/tmp/initMannheim.R`>

Ziele

1. „Philosophie“ von R verstehen: **Alles ist ein Objekt!** & **Viele Wege führen nach Rom!**
2. Unterschiede zwischen R und SPSS/Stata abschätzen können und
3. Antwort auf die Frage geben können, ob R für die eigene Arbeit sinnvoll sein kann.
4. Beispiele komplett verstanden haben.

Notation

Auszeichnung	Bedeutung
	R-Beispiel
	Umfangreiche Erläuterungen nötig, wird deshalb hier ausgespart.
`mean(xy)`	R-Code wird mit nicht-proportionaler Schrift dargestellt.
`R>`	Eingabeaufforderung (Prompt) des R-Interpreters. Steht immer *vor* dem R-Output.

Was ist R?

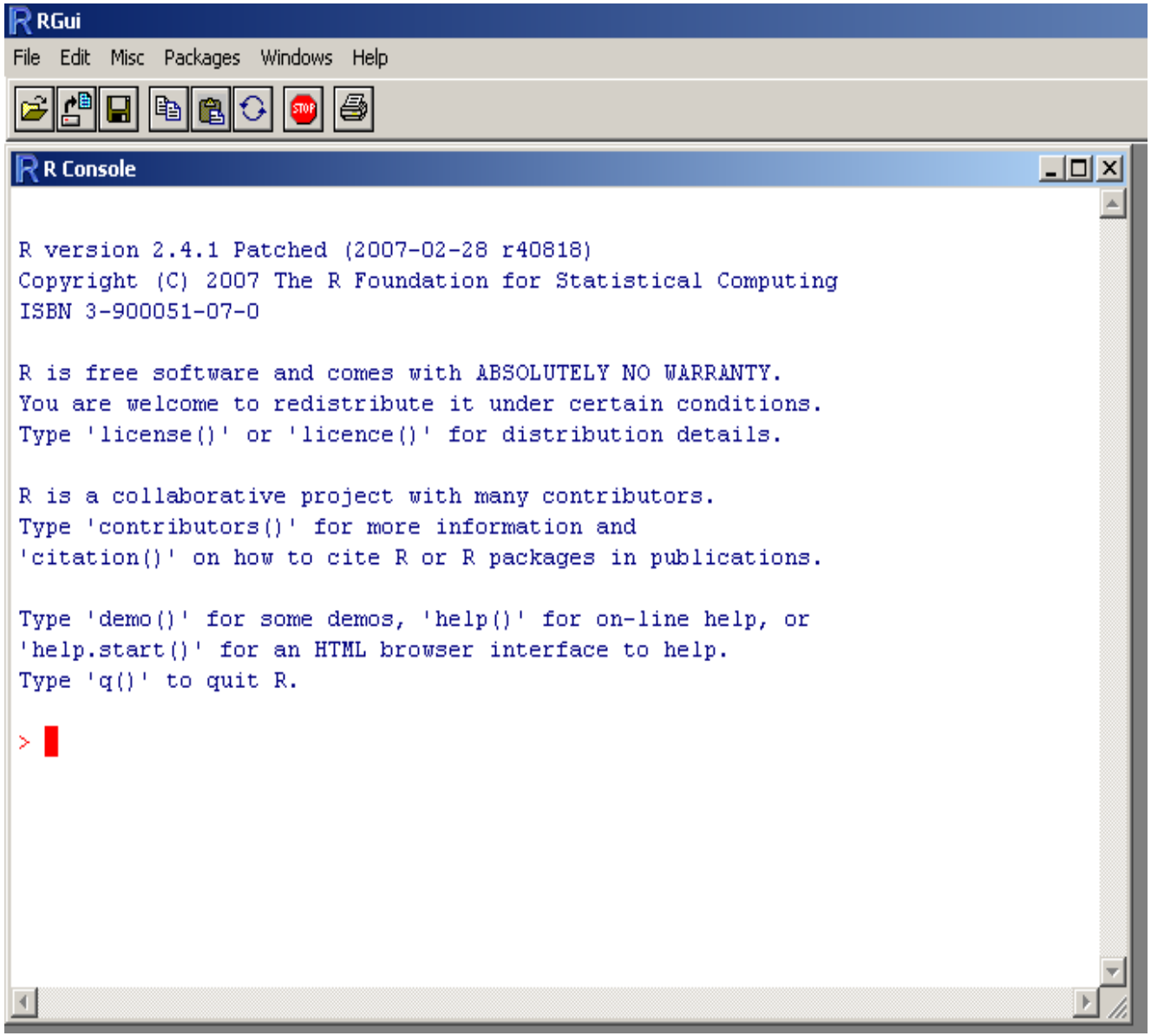
Die *offizielle* Definition für R lautet:

> *„R is a language and environment for statistical computing and graphics“ (Die vollständige Antwort findet sich unter <http://www.r-project.org/about.html>).*

Das heißt, R ist

1. der Name für ein Programm zur (statistischen) Datenanalyse und zur Visualisierung von Daten.
2. aber auch der Name für eine (Interpreter-)Programmiersprache (genauer: R ist ein „Dialekt“/Derivat von S).

Wie sieht R (unter unter MS-Windows) aus?



R vs SPSS vs Stata

Beim Wechsel von SPSS/Stata zu R sind zwei große Verständnishürden zu überwinden:

1. In R gibt es so gut wie keine grafische Benutzeroberfläche (*GUI*); sämtliche Befehle werden von Hand eingetippt (üblicherweise in einem Editor).

2. In R gibt es keine Beschränkung auf einen einzigen Datensatz.[1] In R gibt es Objekte und davon können – ausreichend Speicher vorausgesetzt – beliebig viele erzeugt werden.

[1] Seit Version 14 gibt es auch in SPSS die Möglichkeit, mehrere Datensätze gleichzeitig offen zu halten. Dennoch verbietet sich, wie noch zu zeigen sein wird, ein Vergleich mit R.

Objekte in R

„*Alles* ist ein Objekt, jegliche Art Daten und Funktionen“ (Ligges 2007: 13).

Was ist ein Objekt?

Objekte sind bspw. Ergebnisse von Berechnungen (etwa Mittelwerte, Tabellen, Regressionen, etc.), Datensätze, Graphiken.

Was nützt das? Wie geht man mit Objekten um?

- Beliebig viele Objekte können gleichzeitig im Speicher existieren – begrenzt nur durch die Größe des Speichers.
- Das „Speichern“ eines Ergebnisses in einem Objekt geschieht durch eine **Zuweisung** (symbolisiert durch <-, z.B.: `x <-1`).
- Gespeicherte Objekte können für weitere Berechnungen genutzt werden.
- Gegegeben eine bestimmte Objektklasse, „kennt“ R häufig den „korrekten“ Umgang damit (`plot(objektKlasse1)`, `plot(objektKlasse2)`).
- Objektnamen sind *case sensitive*, also `plot(objektKlasse2)` und `plot(ObjektKlasse2)` beziehtsich auf zwei verschiedene Objekte.

Vorteile von R

- Unter der GPL (GNU General Public License) lizensiert: (1) Quelltext ist verfügbar, (2) das Programm ist kostenlos.
- Sehr renommierte Entwicklergemeinde.
- Hohe Aktualität; neue statistische Verfahren werden inzwischen häufig zuerst in R implementiert.
- R ist für alle gängigen Betriebssysteme verfügbar.
- Als Programmiersprache ist R sehr flexibel (Datenaufbereitung, statistische Verfahren, Graphik).
- Einzigartige Möglichkeiten zur Erzeugung von Graphiken.
- Neue Funktionen können durch *packages* (in Stata *ado-files*) hinzugefügt werden (aktuell etwa 1000).
- Umfangreiches Dokumentationsmaterial verfügbar.
- Sehr flexibles Reporting-System (zum Beispiel *Sweave*) – diese Folien wurden mit Hilfe von Sweave erstellt.
- Hoher Grad an Automatisierung von Abläufen möglich.
- (Teilweise) Sehr ausführliche und (fast immer) mit lauffähigen Beispielen versehende Hilfeseiten.
- ...

Nachteile von R

- R ‚erschlägt' mit seinen Möglichkeiten; es fällt schwer, den Überblick zu finden.
- Wenn Benutzerfreundlichkeit *point-and-click* heißt, dann ist R wenig benutzerfreundlich.
- Umgang mit (Meta)Daten manchmal etwas spartanisch (keine ‚echten' *value/variable labels*; keine verschiedenen Typen von *missing values*).
- Kein globaler Umgang mit Gewichtungen möglich; insgesamt auch weniger Möglichkeiten als in Stata.
- Komplexe Verfahren auf große Datensätze angewandt (z.B. der kum. Allbus) kann zu Speicherproblemen führen.
- Wenngleich das *package*-System von R auf syntaktische Korrektheit achtet, können (in sehr seltenen Fällen) verschiedene *packages* „unglücklich" miteinander interagieren.
- ...

Die beiden Arbeitsmodi von R

Interaktiver Modus (Read-eval-print loop)

1. Eingabe / Aufruf einer R-Funktion,
2. R wertet diese aus,
3. das Ergebnis dieser Auswertung wird (meistens) am Bildschirm ausgegeben.

(→ Für die tägliche Arbeit wird der interaktive Modus selten genutzt.)

Batch-Modus

Mehrere R-Funktionen werden in eine Datei geschrieben, diese Datei wird in R mit `source(file="batchFilename")` (für Besonderheiten bei der Pfadangabe siehe Folie 3) aufgerufen und alle Befehle werden in einem ‚Rutsch' ausgeführt (*do-files* in Stata, *syntax-files* in SPSS).
(→ übliche Arbeitsweise)

Die Syntax einer R-Funktion

Formale Darstellung

```
funktionsname(Argument1 = Wert1, Argument2 = Wert2, ...)
```

Beispiele für Funktionsaufrufe

`plot(x = 1, y = 1)`	Plot eines Punktes
`table(myVariable)`	Haeufigkeiten der Variablen myVariable
`mean(x = myVariable)`	Mittelwert
`sd(x = myVariable)`	Standardabweichung
`summary(x = myVariable)`	Deskriptive Statistiken
`q()`	R beenden, Funktionsaufruf ohne Argument

Die Syntax einer R-Funktion (Fortsetzung)

- Solange eindeutig ist, welcher Wert, welchem Argument zugewiesen wird, ist auch der folgende Funktionsaufruf gültig: `funktionsname(Argument2 = Wert2, Argument1 = Wert1, ...)`.
- Solange die Reihenfolge der übergebenen Funktionswerte der Reihenfolge der Argumente entspricht, ist auch der folgende Funktionsaufruf gültig: `funktionsname(Wert1, Wert2, ...)`.
- Viele Grundfunktionen haben Voreinstellungen (*defaults*), z.B. die Berechnung des Mittelwertes `mean(x, trim = 0, na.rm = FALSE, ...)` (siehe `help(mean)`).
- R ist *case sensitive*, d.h. `plot(objekt)` ist (solange `objekt` existiert) ein gültiger Aufruf der `plot`-Funktion. Hingegen ist `PLOT(objekt)` ungültig.

Das Einfluss von Partnerschaftskonflikten auf die subjektive Partnerschaftstabilität

Inhaltliche Fragestellung

Einfluss der Partnerschaftszufriedenheit, der Konflikthäufigkeit und des Konfliktverhaltens auf die subjektive Partnerschaftsstabilität.

Arbeitsschritte

1. Daten einlesen
2. Daten aufbereiten (Missings definieren, neue Variablen kreieren etc.)
3. Deskriptive Statistik & Graphiken
4. Multivariate Modelle

Übersicht

R-interne Hilfen

In Abhängigkeit von der Fähigkeit, das Problem exakt beschreiben zu können, existieren unterschiedliche Hilfesysteme. Nachfolgend R-interne Hilfeseiten und Dokumentation:

- Funktionsname bekannt: `help(funktionsname)` oder die Kurzform `?funktionsname`
- Funktionsname bekannt beziehungsweise teilweise bekannt: `find("teilDesFunktionsnamens")`, `apropos("teilDesFunktionsnamens")`
- Kurzbeschreibung eines *package*: `packageDescription(packageName)`
- Inhalt (Funktionen und/oder Datensätze) eines *package* darstellen: `help(package = packageName)` oder `library(help = packageName)`
- Funktionsname unbekannt, Suche nach Stichwörtern: `help.search("suchausdruck")`, z.B.: `help.search("propensity␣score")`

R-externe Hilfen

Nachfolgend eine Liste R-externer Hilfeseiten und Dokumentation:

- Hilfesystem im Browser starten: `help.start()`
- In jeder R-Installation sind eine Reihe von Handbüchern enthalten. Diese sind nach Aufruf von `help.start()` zugänglich oder können direkt im R-Verzeichnis eingesehen werden (`/doc/manual`).
- Das E-Mail-Archiv von R-help, R-sig-geo, und R-sig-mixed-models durchsuchen: `RSiteSearch("suchstring")` oder <`http://www.r-project.org/`> → *search* auswählen.
- Auf CRAN sind nach Themen geordnete Listen von *packages* verfügbar: *task views* (Econometrics, SocialSciences, Graphics, etc.), etwa unter <`http://cran.r-project.org/`>.

Dokumentation

Auf der R-Homepage unter <http://www.r-project.org/> gibt es den Bereich *Documentation*. Hier finden sich folgende Einträge:

- Manuals (vom R Development Core Team; mehr oder weniger für Anfänger geeignet „An Introduction to R“)
- FAQ (Frequently Asked Questions)
- Newsletter
- Wiki
- Books (auf R bezogene Bücher)
- Other (weitere Dokumente, teilweise auch auf Deutsch)

Empfehlenswerte Dokumente

- „Auswertung ökologischer Daten“ <http://www.hydrology.uni-kiel.de/~schorsch/statistik/statistik_datenauswertung.pdf>
- „Einführung in die Statistik mit R“ <http://www.wiwi.uni-bielefeld.de/~frohn/Mitarbeiter/Handl/stagrund.html>
- „Eine kurzer Streifzug durch R“ <http://hhbio.wasser.tu-dresden.de/projects/modlim/doc/First_Rsession.pdf>
- **„R FOR SAS AND SPSS USERS“** <http://oit.utk.edu/scc/RforSAS&SPSSusers.pdf>
- „ Using R for psychological research: A simple guide to an elegant package“ <http://personality-project.org/r/r.guide.html>
- „Statistik mit R“ <http://www.staff.uni-marburg.de/~gruener/lehre/r.w05/welcome.html>

Mailing-Liste *r-help*

Bevor man sich an die Liste mit einer Frage wendet, bitte die unten aufgeführten Hinweise verinnerlicht haben!

R-help@stat.math.ethz.ch mailing list
https://stat.ethz.ch/mailman/listinfo/r-help
PLEASE do read the posting guide
http://www.R-project.org/posting-guide.html
and provide commented, minimal, self-contained, reproducible code.

Übersicht

Übersicht

Nutzbare Datenformate

ASCII-Daten (im `base`-package enthalten; **zum Umgang mit packages → Folie 114ff.**)

Es lassen sich beliebige ASCII-Daten (Text-Daten) ein- und ausgeben.

Binärdaten (zusätzliche *packages* erforderlich)

- SPSS (`.por`-Dateien), Stata, SAS (laden und abspeichern, siehe packages `foreign` und `Hmisc`)
- R (via `save()` und `load()`)
- MS-Excel
- etc.

DBMS (zusätzliche *packages* erforderlich)

Diverse Datenbankmanagementsysteme wie Postgres, MySQL, MS-Access (via ODBC), etc.

Hinweise zum Speichern von Datenobjekten

- Ausführliche Informationen zum Umgang mit Daten bietet: R Development Core Team, 2007: R Data Import/Export (in der Grundinstallation von R enthalten).
- Wenn Daten eingelesen werden, dann **müssen** sie einem Objekt zugeordnet werden; etwa (unter MS-Windows)
  `myData <-read.table(file="c:/myData.csv")` oder
  `myData <-read.table(file="c:\\myData.csv")`.
- Unter **MS-Windows dürfen keine backslash (\) verwendet werden**. Dateipfade werden entweder mit einem *slash* (/) oder einem doppelten *backslash* (\\) spezifiziert.
- Bedingt durch die Objektorientierung von R muss man sich Gedanken darüber machen, was abgespeichert werden soll. Abgespeichert werden können einzelne Objekte, Objektgruppen oder alle verfügbaren Objekte (*workspace*).

ASCII-Daten laden und speichern

Laden

- `read.table(file="", header = TRUE, sep = "", ...)` (daneben gibt es noch: `read.csv()`, `read.csv2()`, `read.delim()`, `read.delim2()`)
- `read.fwf()` (Dateien mit vorgegebener Spaltenbreite; *fixed-width-format files*).

Speichern

- `write.table(x, file = "", sep = "")`
- `write.csv(...)`
- `write.csv2(...)`

R-Objekte laden und speichern

Laden

- Ein (oder mehrere) via `save()` gespeichertes Objekt laden:
  `load(file = "",...)`
- Beliebigen R-Code (z.B. via `dump()` erzeugt) laden:
  `source(file = "",...)`
- `data(nameOfDataset, package = packageName)` oder zunächst mit `library(packageName)` das *package* laden und anschließend `data(nameOfDataset)` eingeben.

Speichern

- R-Objekte als Binärdaten speichern:
  `save(object1, object2, ..., file="")`
- R-Objekte als ASCII-Text speichern:
  `dump(c("object1","object2",...),file="")`

SPSS-Datensätzen laden

Sowohl das Laden als auch das Speichern von SPSS- bzw. Stata-Daten setzen das `foreign`-package voraus. Das Einlesen von SPSS-Daten geschieht mit der Funktion `read.spss`:

```
read.spss(file, use.value.labels = TRUE, to.data.
frame = FALSE, max.value.labels = Inf, trim.factor.
names = FALSE)
```

und diese kennt u.a. folgende Argumente:

- `use.value.labels`: Falls *value labels* vorhanden sind, werden diese statt der numerischen Werte genutzt und die Variable ist vom Typ `factor`. Mit `max.value.labels` lässt sich etwas gegensteuern.
- `to.data.frame`: Sollte immer auf `TRUE` gesetzt werden, da ansonsten die Daten als Liste repräsentiert werden.

SPSS-Daten laden

```
R> library(foreign)
R> meineDatei <- "E:/projects/paarkonf/data/pairfam/welle1.2005/data
R> nonsensObject <- read.spss(file = meineDatei,
+     to.data.frame = TRUE)
```

Und jetzt? Z.B. die ersten zehn (`[1:10]`) Variablennamen auflisten:

```
R> names(nonsensObject)[1:10]
```

```
 [1] "PAARID"   "PART"     "AZ001"    "ALTER"
 [5] "BILDKAT"  "AZ119_R"  "AZ121_R"  "AZ123_R"
 [9] "AZ124_R"  "AZ127_R"
```

Und jetzt das mittlere Alter in Jahren (für die $-Notation siehe Folien 32ff.):

```
R> mean(nonsensObject$ALTER)
```

```
[1] 30.00233
```

```
R> range(nonsensObject$ALTER)
```

```
[1] 14 99
```

Offene Fragen

1. Was bedeutet der Pfeil (<-) noch einmal?
2. Was kann man mit einem Datenobjekt machen?
3. Wie werden einzelne Variablen angesprochen?
4. Was bedeutet `[1:10]`?

Übersicht

Der Zuweisungsoperator

Das „Speichern“ von Daten, Berechnungen, etc. in einem Objekt geschieht durch eine **Zuweisung** (*assignment*), davon gibt es (unter anderem historisch bedingt) verschiedene Varianten:

- `x <-3` (empfohlene Variante!),
- `3 -> x` (besser von rechts nach links zuweisen!),
- `x = 3` (nicht empfohlen!).
- `assign("x",3)` (bietet noch weitergehende Möglichkeiten).

Objekten Werte zuweisen

```
R> mean(dataDyadic$ALTER)

[1] 30.00233

R> mean.age <- mean(dataDyadic$ALTER)
R> mean.age

[1] 30.00233

R> mean.age2 <- mean.age
R> mean.age2

[1] 30.00233
```

Datentypen: Welche Art von Daten?

Atomare Datentypen

- Leere Menge (*NULL*) (`NULL`)
- Logische Werte (*logical*) (`TRUE, FALSE`)
- Ganze und reelle Zahlen (*numeric*) (`2.789, 10.006`)
- Komplexe Zahlen (*complex*) (`1.56+1i`)
- Buchstaben und Zeichenfolgen (*character*) (`"a","b"`)

Faktoren

- Faktoren (*factor*) (`"verheiratet","geschieden","verwitwet"`)
- Geordnete Faktoren[a] (*ordered*) (`"selten","haeufig"`)

[a] Sollte nicht mehr benutzt werden, Faktoren können ein Argument `ordered = TRUE | FALSE` enthalten.

Datenstrukturen: Wie werden Daten abgelegt?

Welche Datenstrukturen gibt es?

- Vektoren (*vector*) (`c(1,2,3)`)
- Matrizen (*matrix*) (`matrix(c(1,2,3,4),ncol=2)`)
- Arrays (*array*)
- Listen (*list*)
- Datensätze (*data frames*) (`data.frame(c(1,2,3,4),c(1,2,3,4))`)

Datenstrukturen testen und umwandeln

- `is.vector(object)`, `is.matrix(object)`, `is.array(object)`, ...
- `as.vector(object)`, `as.matrix(object)`, `as.array(object)`, ...

Eine deutliche Warnung beim Umgang mit Daten!

Sich immer klar machen, welcher Datentyp und welche Datenstruktur benutzt wird! Mag in einigen Programmen die Mittelwertsberechnung der Konfessionszugehörigkeit möglich (wenn auch nicht sinnvoll) sein

```
COMPUTE mymean = MEAN(V1).
EXECUTE.
```

so führt das in R (berechtigterweise) zu einem `NA` („Not Available“):

```
R> V1 <- as.factor(c("katholisch", "evangelisch",
+     "sonstiges"))
R> mymean <- mean(V1)
R> mymean
[1] NA
R> class(V1)
[1] "factor"
```

Die zentrale Datenstruktur: `data.frame`

- Die meisten Datenobjekte werden in Form eines `data.frame` vorliegen.
- Ein `data.frame` setzt sich aus 1 bis c (c = *column*) Vektoren (Variablen) zusammen.
- Ein `data.frame` ist eine $r \times c$-Matrix (r = *row*).
- Ein `data.frame` kann Spalten mit unterschiedlichen Datentypen enthalten (innerhalb der Spalte *nur ein* Datentyp möglich).
- Eine Liste der enthaltenen Variablen erhält man mit der Funktion `names(myDataFrame)`.
- Die c Spalten bzw. Variablen eines `data.frame` lassen sich wie folgt auswählen:
  - `myDataFrame$VariableName` (Kurzfassung)
  - `myDataFrame[,"VariableName"]` (Langfassung; man beachte das Komma!)
  - `myDataFrame[,ColumnNumber]` (Langfassung; man beachte das Komma!)

Beispiel eines `data.frame`

```
R> dataDyadic[1:10, 1:5]
```

	PAARID	PART	AZ001	ALTER	BILDKAT
1	1031	1	1	15	2
2	1126	1	1	17	2
3	1151	1	1	25	1
4	1154	1	1	24	3
5	1171	1	1	25	2
6	1189	1	1	26	2
7	1201	1	1	25	3
8	1203	1	1	25	1
9	1210	1	1	25	3
10	1215	1	1	25	3

Auswählen einzelner Teile aus einer komplexen Datenstruktur (Indizieren)

- Da im Arbeitsspeicher mehrere (Daten)Objekte existieren können, muss eindeutig sein, auf welches Objekt sich eine Funktion (etwa eine Mittelwertsberechnung) beziehen soll, d.h. (1) wähle Datensatz (`data.frame`) (allg. Objekt), 2. wähle Variable (= Vektor) des `data.frame` ($\rightarrow$ Alternative: `attach(data.frame)`).
- Sofern sich ein Objekt aus mehreren Einzelobjekten zusammensetzt, ist es möglich mit dem [-Operator (genauer: `[...]`) auf einzelne davon zuzugreifen.
- Bei mehrdimensionale Datenstrukturen (`data.frame`, `matrix`, `list` etc.) muss immer deren Dimensionalität berücksichtigt werden:
  - `myDataFrame[1,1]` = 1. Zeile, 1. Spalte = 1 Skalar
  - `myDataFrame[,1]` = Alle Zeilen, 1. Spalte = 1 Vektor
  - `myDataFrame[,"VariableName"]` = Alle Zeile, benannte Spalten = 1 Vektor
  - `myDataFrame[1,]` = 1 Zeile, alle Spalten = 1 Vektor

Auswählen einzelner Teile aus einer komplexen Datenstruktur (Indizieren) (Forts.)

- Mehrere einzelne Indizes können zu einem Vektor zusammengefasst werden und dieser Vektor dann auf ein Objekt angewandt werden:
  - `myDataFrame[,c(1,2,3)]` = Alle Zeilen, 1.–3. Spalte.
  - `myDataFrame[,c("VariableName1","VariableName2")]` = Alle Zeile, benannte zwei Spalten = $r \times 2$-Matrix.
  - `myDataFrame[c(1,2,3),]` = 1.–3. Zeile, alle Spalten.
  - `myDataFrame[c(1,2,3),c(1,2,3)]` = 1.–3. Zeile, 1.–3. Spalte = 3×3 Matrix.
- Ein vorangestelltes Minuszeichen führt zum Ausschluss eines oder mehrerer Elemente:
  - `myDataFrame[,-c(1,2,3)]` = Alle Zeilen, Ausschluss der 1.–3. Spalte.
  - `myDataFrame[-1,]` = Ausschluss 1. Zeilen, alle Spalten.

Nützliche Funktionen zum Umgang mit Daten

- `names(x)`: (Variablen)Namen eines Objektes `x` ausgeben.
- `dim(x)`: Dimensionen (Zeilen- und Spaltenzahl) eines Objektes `x` ausgeben.
- `str(x)`: Strukturellen Aufbau eines Objektes `x` darstellen.

Der Umgang mit einem data.frame

```
R> names(dataDyadic)[c(1:6)]

[1] "PAARID"  "PART"    "AZ001"   "ALTER"
[5] "BILDKAT" "AZ119.R"

R> dim(dataDyadic)

[1] 430  51

R> dataDyadic[1:10, "PAARID"]

 [1] 1031 1126 1151 1154 1171 1189 1201 1203 1210
[10] 1215

R> mean(dataDyadic[, "ALTER"])

[1] 30.00233

R> sd(dataDyadic[, "ALTER"])

[1] 8.86824
```

Der Umgang mit einem data.frame

```
R> names(dataDyadic)[c(1:6)]

[1] "PAARID"  "PART"    "AZ001"   "ALTER"
[5] "BILDKAT" "AZ119.R"

R> attach(dataDyadic)
R> PAARID[1:10]

 [1] 1031 1126 1151 1154 1171 1189 1201 1203 1210
[10] 1215

R> mean(ALTER)

[1] 30.00233

R> sd(ALTER)

[1] 8.86824

R> detach(dataDyadic)
```

Nach inhaltlichen Kriterien Teilmengen erzeugen

- Häufig sollen statistische Operationen nur auf Teilmengen eines Datensatzes angewandt werden.
- In R gibt es mehrere Möglichkeiten:
  1. Mit
     `myDataFrameNeu <-subset(myDataFrame,LogicalExpressions)`
     eine reduzierte Kopie des Ursprungsdatensatz erstellen (= `SELECT IF` in SPSS oder `if` in Stata).
  2. Mit Hilfe von logischen Objekten nur ausgewählte Teile eines Datenobjektes bearbeiten. Zweistufige Vorgehensweise:
     1. Erzeuge durch die Verwendung von logischen Operatoren einen Vektor mit logischen Werten (`c(TRUE, FALSE, FALSE, ...)`).
     2. Wende diesen Vektor auf einen Datensatz an, wobei alle `TRUE`-Elemente im Datensatz verbleiben, während `FALSE` Fälle ausschließt.

Logische Operatoren, Funktionen und Verknüpfungen

`==, !=`	gleich, ungleich
`>, >=`	größer als, größer gleich
`<, <=`	kleiner als, kleiner gleich
`!`	nicht
`&, &&`	und (vektorwertig), und (nicht vektorwertig)
`|, ||`	oder (vektorwertig), oder (nicht vektorwertig)
`xor()`,	entweder oder (aussschließend)
`TRUE, FALSE`	wahr, falsch

Logische Ausdrücke

Altersmittelwert der 15-17 jährigen Personen:

```
R> c(dataDyadic$ALTER >= 14 & dataDyadic$ALTER <=
+     18)[1:10]

 [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
 [8] FALSE FALSE FALSE

R> mean(dataDyadic[dataDyadic$ALTER >= 14 &
+     dataDyadic$ALTER <= 18, "ALTER"])

[1] 16.36735

R> dataDyadicNeu <- subset(dataDyadic, ALTER >=
+     14 & ALTER <= 18)
R> mean(dataDyadicNeu[, "ALTER"])

[1] 16.36735
```

Übersicht

Fehlende Werte

- `NA` steht für *Not Available* und beschreibt unter R fehlende Werte.
- `NaN` steht für *Not a Number* und beschreibt das Ergebnis einer nicht definierten Operation (etwa `log(-1)`).
- `NULL` steht für eine leere Menge.

Ratschläge zum Umgang mit fehlenden Werten

1. In R muss fehlenden Werten große Aufmerksamkeit geschenkt werden.
2. Werden einer Funktion fehlende Werte übergeben, dann ist das Ergebnis dieser Funktion immer `NA`.
3. Einige Funktionen schließen in der Grundeinstellung (*default*) **keine** fehlenden Werte aus! Beispielsweise die Funktion `mean()`:

```
R> mean(c(1, 2, 3, NA))

[1] NA

R> mean(c(1, 2, 3, NA), na.rm = TRUE)

[1] 2

R> mean(c(1, 2, 3, NaN))

[1] NaN

R> mean(c(1, 2, 3, NaN), na.rm = TRUE)

[1] 2
```

Der Umgang mit fehlenden Werten

- Mit Hilfe des Arguments `na.rm` („NA remove“) erlauben *einige* Funktionen den Auschluss von fehlenden Werten, z.B. `mean(c(1, 2, 3, NA),na.rm = TRUE)`.
- Einen *listwise*n Ausschluss von fehlenden Werten erlaubt die Funktion `na.omit()`:

```
R> x <- c(1, 2, 3, NA)
R> na.omit(x)
[1] 1 2 3
attr(,"na.action")
[1] 4
attr(,"class")
[1] "omit"
```

- Die Existenz fehlender Werte lässt sich mit `is.na(x)`, `is.nan(x)` oder `is.null(x)` testen. Liegen fehlende Werte vor, dann geben die Funktionen `TRUE` zurück, ansonsten `FALSE`.

Fehlende Werte

```
R> table(is.na(dataDyadic$BILDKAT))

FALSE  TRUE
  414    16

R> mean(dataDyadic$BILDKAT)

[1] NA

R> mean(dataDyadic$BILDKAT, na.rm = TRUE)

[1] 2.492754
```

Eine ausführliche Einführung der `table`-Funktion findet sich ab Folie 3.

Übersicht

Rekodieren von Variablen

Gegeben ein Datenvektor, die Werte 7, 8, 9 sind im Codebuch als Missings definiert und sollen nun mit NA ersetzt werden.

- „Klassische“ Methode:

```
R> (x <- c(1:10))

 [1]  1  2  3  4  5  6  7  8  9 10

R> x[x %in% c(7, 8, 9)] <- NA
R> x

 [1]  1  2  3  4  5  6 NA NA NA 10
```

- „Neuere“ Methoden unter Verwendung von *packages* wie `car`, `epicalc` etc.:

```
R> library(car)
R> (x <- c(1:10))

 [1]  1  2  3  4  5  6  7  8  9 10

R> x <- recode(x, "c(7, 8, 9) = NA")
R> detach(package:car)
```

Datensätze zusammenspielen (*merge*), I

```
R> id <- c(1, 2, 3)
R> x <- c(1, 4, 4)
R> a <- data.frame(id, x)
R> id <- c(1, 2, 3, 1, 2, 2)
R> y <- c(1, 4, 4, 5, 5, 1)
R> b <- data.frame(id, y)
R> a

  id x
1  1 1
2  2 4
3  3 4

R> b

  id y
1  1 1
2  2 4
3  3 4
4  1 5
```

Datensätze zusammenspielen (*merge*), II

```
R> merge(a, b)
```

```
  id x y
1  1 1 1
2  1 1 5
3  2 4 4
4  2 4 5
5  2 4 1
6  3 4 4
```

Übersicht

SPSS-/Stata-Datensätze speichern

Sowohl das Laden als auch das Speichern von SPSS- bzw. Stata-Daten setzen das `foreign`-package voraus.

Daten im `.sav`-Format abspeichern

Die Funktion `write.foreign` erzeugt ein Daten- und ein Syntax-File. In SPSS muss dann das Syntax-File („codefile“) ausgeführt werden.

```
write.foreign(df, datafile, codefile, package = c("SPSS", "
Stata", "SAS"), ...)
```

Daten im `.dta`-Format abspeichern

```
write.dta(dataframe, file, version = 6)
```

Übersicht

Übersicht

Einfache deskriptive Statistiken mit summary

```
R> summary(dataDyadic[, c("ZUFRIED1", "AP38SUM1",
+     "STABIL1")])
```

```
    ZUFRIED1         AP38SUM1
 Min.   : 2.000   Min.   : 2.000
 1st Qu.: 8.000   1st Qu.: 8.000
 Median : 9.000   Median : 9.000
 Mean   : 8.443   Mean   : 8.443
 3rd Qu.:10.000   3rd Qu.:10.000
 Max.   :10.000   Max.   :10.000
 NA's   :10.000   NA's   :10.000
    STABIL1
 Min.   : 2.000
 1st Qu.: 5.000
 Median : 6.000
 Mean   : 5.166
 3rd Qu.: 6.000
 Max.   : 6.000
 NA's   :32.000
```

Die `table(...)`-Funktion

- Vorteile:
  - 1 bis *n*-dimensionale Tabellen mit absoluten Häufigkeiten (für relative Häufigkeiten siehe `prop.table(...)`)
  - In der Grundinstallation vorhanden (`stats`-package)
- Nachteile:
  - Keine Variablenbezeichnungen
  - Kein `data=...`-Argument möglich, d.h. mehr Schreibarbeit

```
R> table(dataDyadic$SEX)

  0   1
217 213

R> table(dataDyadic$BILDKAT, dataDyadic$SEX)

     0   1
  1  11  19
  2  81  69
  3 119 115
```

Die `ftable(...)`-Funktion

- Vorteile:
  - 2 bis *n*-dimensionale Tabellen mit absoluten Häufigkeiten (für relative Häufigkeiten siehe `prop.table(...)`)
  - In der Grundinstallation vorhanden (`stats`-package)
  - Formelnotation
  - `data=...`-Argument möglich
- Nachteile:
  - Unintuitiv (siehe Bsp., wenn Spalten = unab. Var.)

```
R> ftable(SEX ~ BILDKAT, data = dataDyadic)
```

```
        SEX   0   1
BILDKAT
1            11  19
2            81  69
3           119 115
```

Die prop.table(...)-Funktion

Berechnet Gesamt-, Zeilen- oder Spaltenprozente

```
R> prop.table(ftable(SEX ~ AZ001, data = dataDyadic))

      SEX            0           1
AZ001
0         0.018604651 0.486046512
1         0.486046512 0.009302326

R> prop.table(ftable(SEX ~ AZ001, data = dataDyadic),
+     1)

      SEX           0          1
AZ001
0         0.03686636 0.96313364
1         0.98122066 0.01877934

R> prop.table(ftable(SEX ~ AZ001, data = dataDyadic),
+     2)

      SEX           0          1
AZ001
0         0.03686636 0.98122066
1         0.96313364 0.01877934
```

Die CrossTable(...)-Funktion (gmodels-package)

```
R> library(gmodels)
R> CrossTable(dataDyadic$AZ001, dataDyadic$SEX,
+     format = c("SPSS"))
```

```
   Cell Contents
|-------------------------|
|                   Count |
| Chi-square contribution |
|             Row Percent |
|          Column Percent |
|           Total Percent |
|-------------------------|

Total Observations in Table:  430

                 | dataDyadic$SEX
dataDyadic$AZ001 |         0  |         1  | Row Total  |
-----------------|------------|------------|------------|
               0 |         8  |       209  |       217  |
                 |    94.094  |    95.861  |            |
                 |     3.687% |    96.313% |    50.465% |
                 |     3.687% |    98.122% |            |
                 |     1.860% |    48.605% |            |
-----------------|------------|------------|------------|
               1 |       209  |         4  |       213  |
                 |    95.861  |    97.661  |            |
                 |    98.122% |     1.878% |    49.535% |
                 |    96.313% |     1.878% |            |
                 |    48.605% |     0.930% |            |
-----------------|------------|------------|------------|
    Column Total |       217  |       213  |       430  |
                 |    50.465% |    49.535% |            |
                 |------------|------------|------------|
```

Übersicht

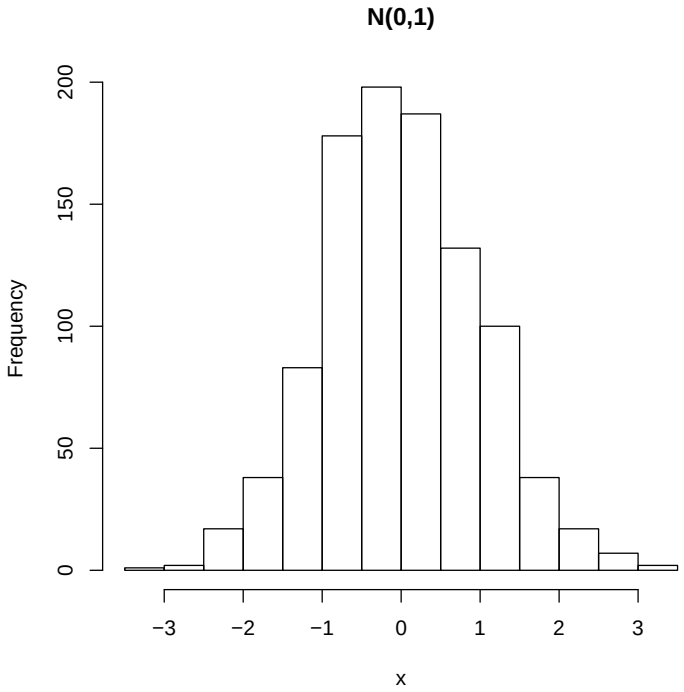
Verteilungsfunktionen

Für die Berechnung von Dichte- und Verteilungsfunktion sowie von Pseudo-Zufallszahlen und Quantilen der gängigen Verteilungen sind in R bereits Funktionen vordefiniert:

- `d...`: (density) Dichte-/Wahrscheinlichkeitsfunktion (Standardnormalverteilung: `dnorm(...)`; Binomialverteilung `dbinom(...)`).
- `p...`: (probability) kummulierte Verteilungsfunktion (Standardnormalverteilung: `pnorm(...)`; Binomialverteilung `pbinom(...)`).
- `q...`: (quantiles) Quantile der Funktion (Standardnormalverteilung: `qnorm(...)`; Binomialverteilung `qbinom(...)`).
- `r...`: (random) Pseudo-Zufallszahlen (Standardnormalverteilung: `rnorm(...)`; Binomialverteilung `rbinom(...)`).

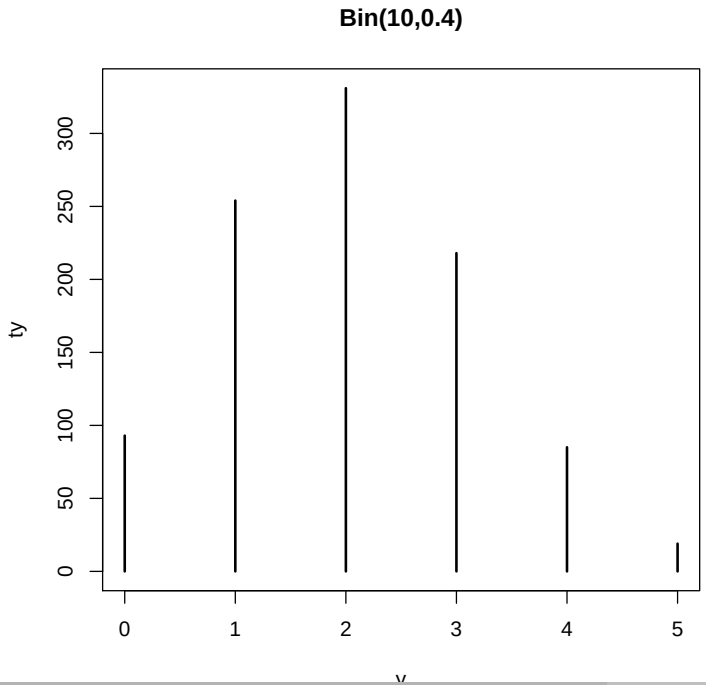
Eine normalverteilte (stetig) Zufallsvariable erzeugen

```
R> x <- rnorm(1000)
R> hist(x, main = "N(0,1)")
```



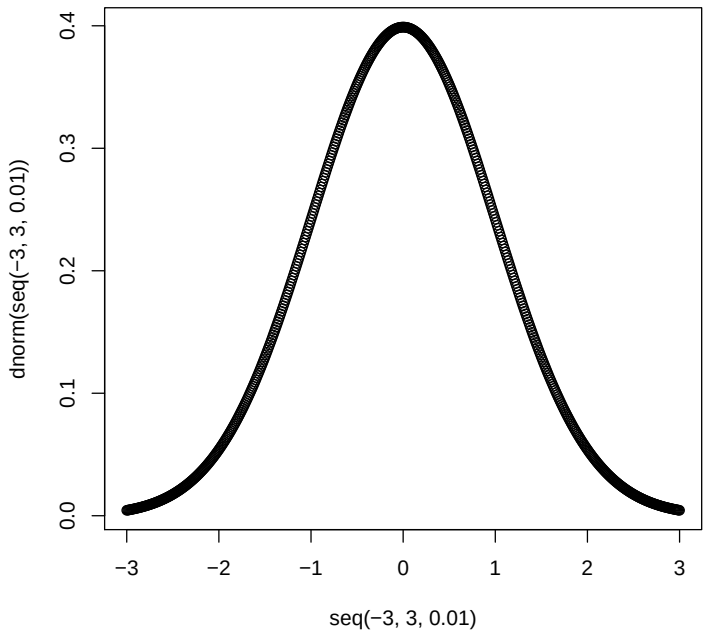
Eine binomialverteilte (diskret) Zufallsvariable erzeugen

```
R> y <- rbinom(1000, 5, 0.4)
R> print(ty <- table(y))

y
  0   1   2   3   4   5
 86 257 361 209  78   9
R> plot(ty, main = "Bin(10,0.4)")
```



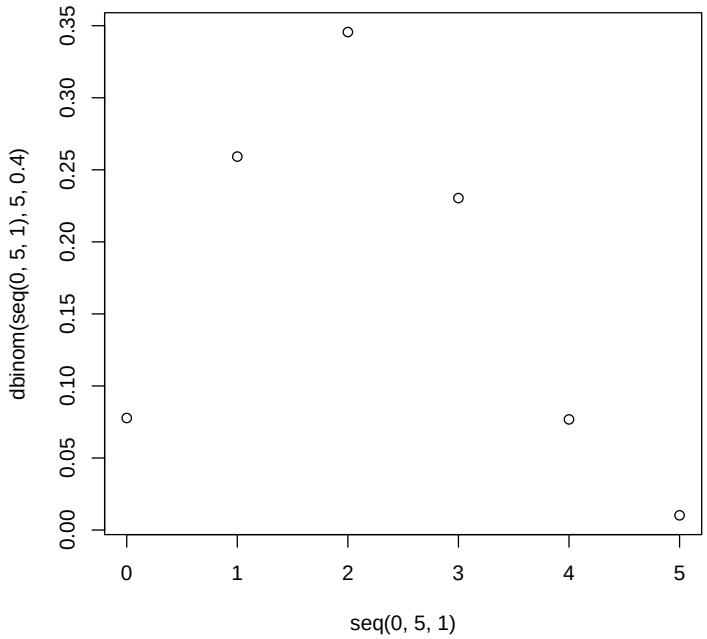
Dichtefunktion (Standardmormalverteilung)

```
R> plot(seq(-3, 3, 0.01), dnorm(seq(-3, 3,
+     0.01)))
```



Wahrscheinlichkeitsfunktion (Binomialverteilung)

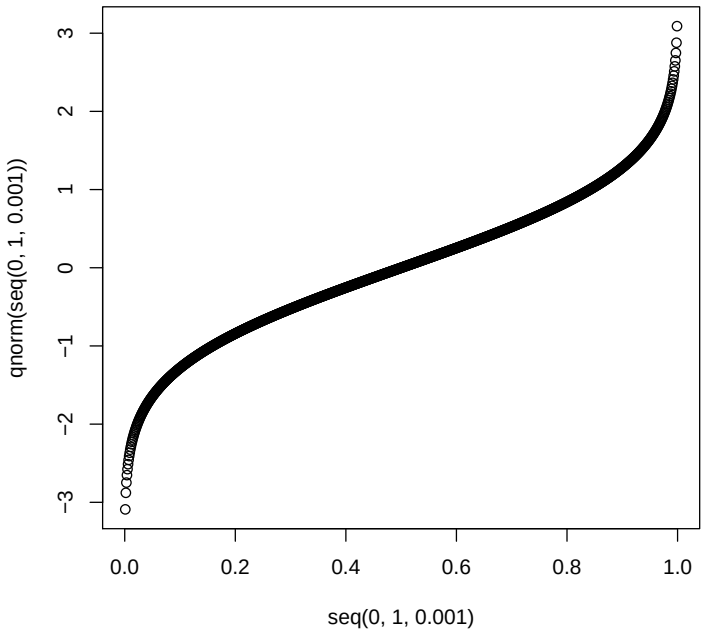
```
R> plot(seq(0, 5, 1), dbinom(seq(0, 5, 1),
+     5, 0.4))
```



Quantilsfunktion (Standardmormalverteilung)

Gegegeben eine Wahrscheinlichkeit, welcher z-Wert ist damit verbunden.

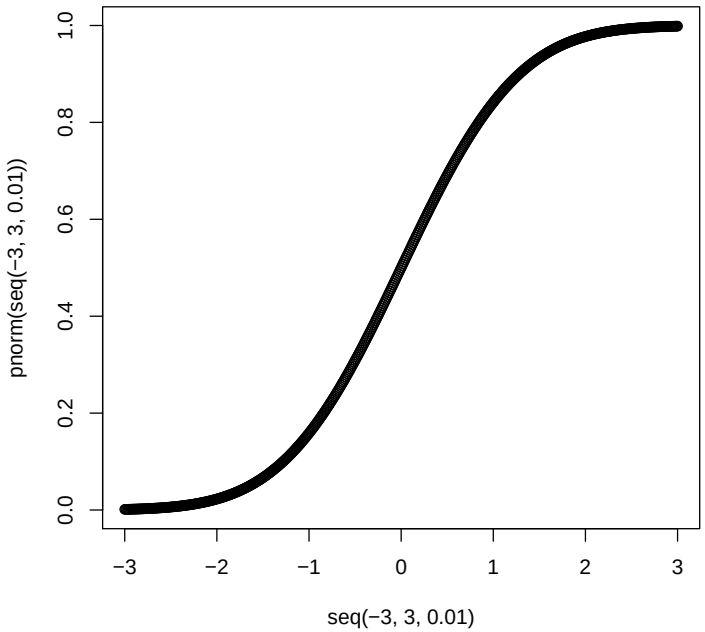
```
R> plot(seq(0, 1, 0.001), qnorm(seq(0, 1,
+     0.001)))
```



Kum. Verteilungsfunktion (Standardmormalverteilung)

Gegeben ein z-Wert, welche Wahrscheinlichkeit ist damit verbunden.

```
R> plot(seq(-3, 3, 0.01), pnorm(seq(-3, 3,
+     0.01)))
```



Übersicht

Eine Auswahl bivariater Zusammenhangsmaße

Metrisch, dichotom

t-Test: `t.test(...)` (base-package), Punkt-biseriale Korrelation[a]: `biserial.cor(...)` (ltm-package)

[a] Bin unsicher, ob diese Variante korrekt ist, siehe auch `http://tolstoy.newcastle.edu.au/R/help/03a/3354.html` für `cor.biserial(...)`.

Metrisch, metrisch

Korrelation nach Pearson: `cor(...)` (base-package), Korrelation nach Pearson/Spearman: `rcorr(...)` (Hmisc-package)

Nominal, nominal

χ^2-Test: `assocstats()` (vcd-package), `chisq.test()` (base-package)

Subjektive Partnerschaftsstabilität nach Geschlecht, I

```
R> dataHetero <- subset(dataDyadic, HETERO ==
+     1)
R> attach(dataHetero)
R> print("Sind die Varianzen in beiden Gruppen gleich?")

[1] "Sind die Varianzen in beiden Gruppen gleich?"

R> (x <- by(STABIL1, list(SEX), var, na.rm = T))

: 0
[1] 1.710276
------------------------------------
: 1
[1] 1.414802
```

Subjektive Partnerschaftsstabilität nach Geschlecht, II

```
R> var.test(STABIL1[SEX == 0], STABIL1[SEX ==
+     1])

F test to compare two variances

data:  STABIL1[SEX == 0] and STABIL1[SEX == 1]
F = 1.2088, num df = 192, denom df = 193,
p-value = 0.1891
alternative hypothesis: true ratio of variances is not equal t
95 percent confidence interval:
 0.9106017 1.6049405
sample estimates:
ratio of variances
          1.208845
```

Subjektive Partnerschaftsstabilität nach Geschlecht, III

```
R> test <- t.test(STABIL1 ~ SEX, data = dataHetero,
+     var.equal = TRUE)
R> detach(dataHetero)
```

Übersicht

Überblick über die Möglichkeiten zur Graphikerzeugung

Es gibt im WWW eine Reihe von Seiten, die Beispiele von R-Grafiken vorstellen. Besonders empfehlenswert ist die R Graph Gallery <http://addictedtor.free.fr/graphiques/> von Romain François.

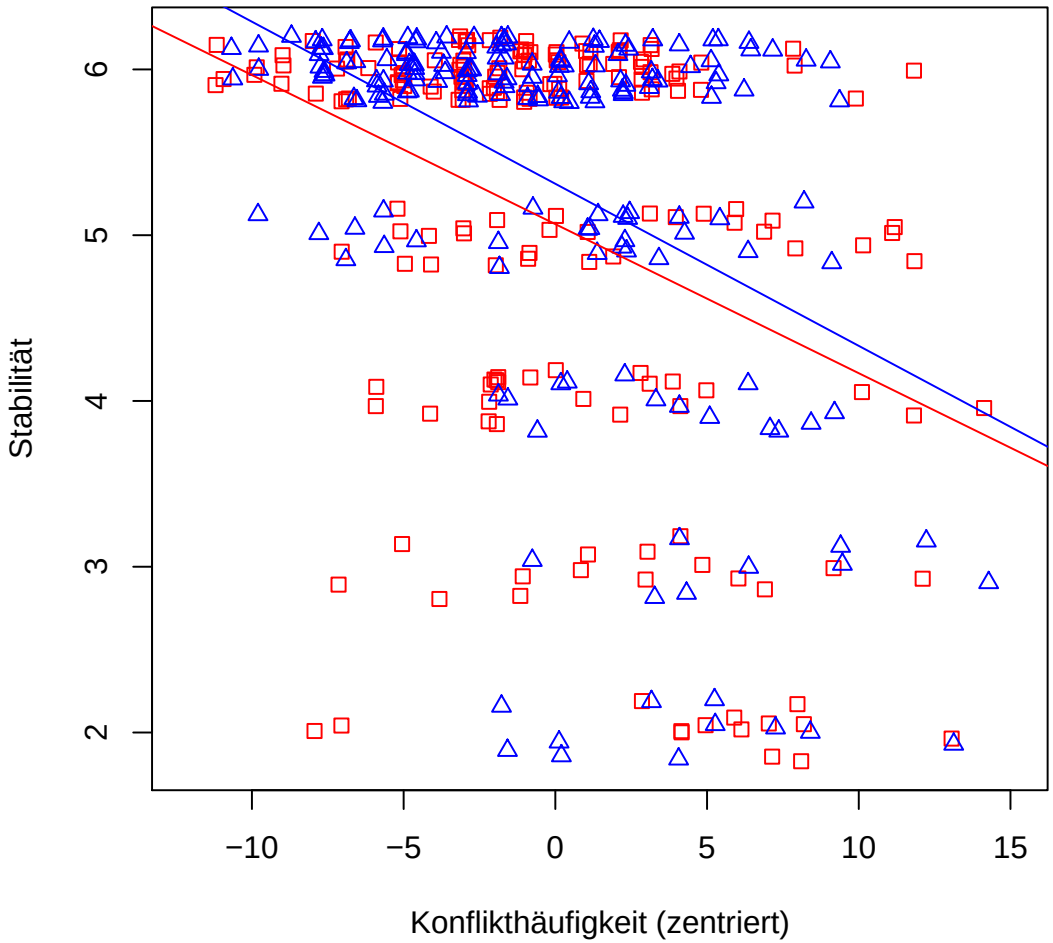
Weitere ansehnliche Seiten sind:

- http://zoonek2.free.fr/UNIX/48_R/all.html
- http://www.stat.ucl.ac.be/ISpersonnel/lecoutre/stats/fichiers/_gallery.pdf
- http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html

Grafiken in R

- *Base graphics*
- `Lattice`: basiert auf dem `grid-package` (siehe auch <`http://cm.bell-labs.com/cm/ms/departments/sia/project/trellis/`>)
- `ggplot`: grammar of graphics (basiert auf dem `grid-package`, siehe auch <`http://had.co.nz/ggplot/`>)

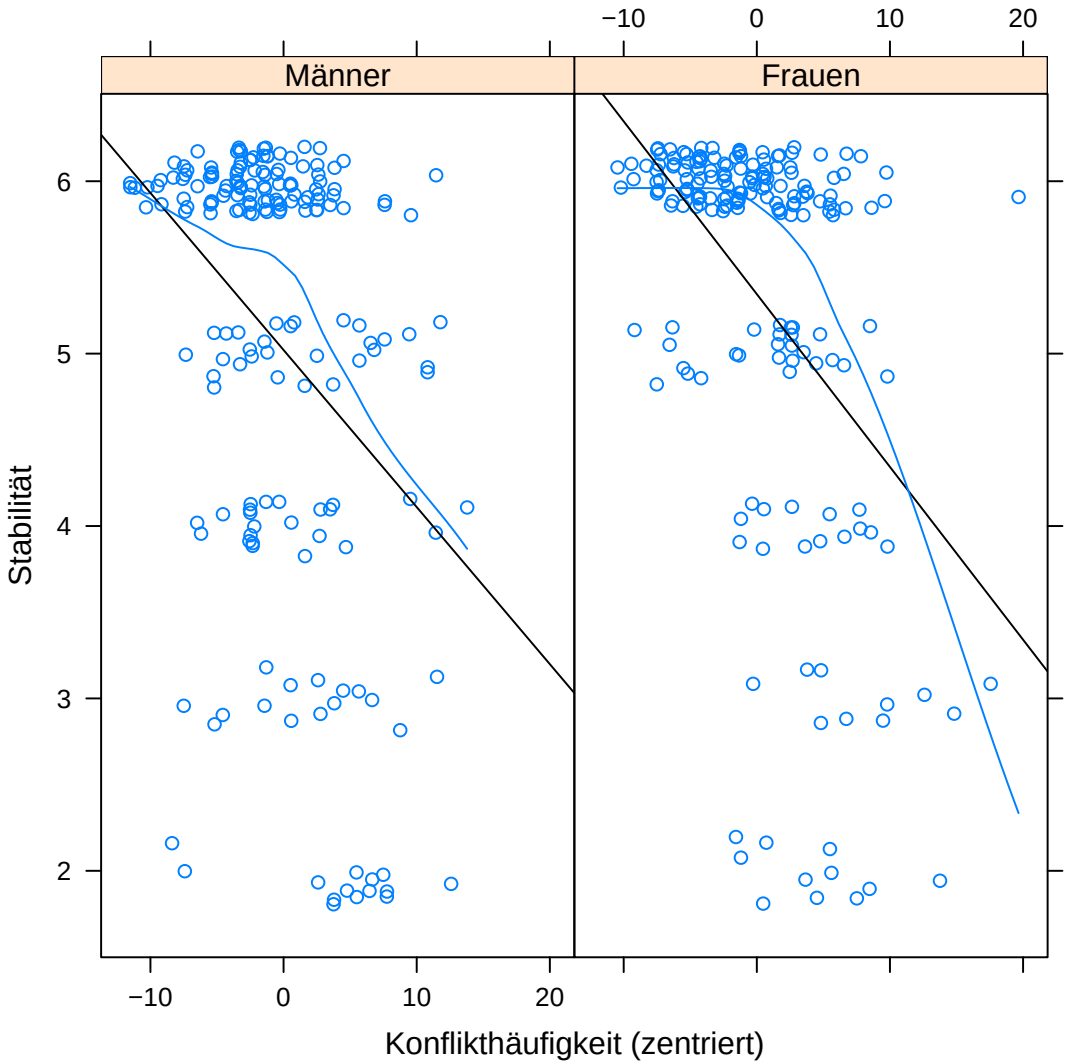
Partnerschaftsstabilität & Konflikthäufigkeit



Partnerschaftsstabilität & Konflikthäufigkeit

```
R> x.m <- jitter(scale(dataHetero$KONFSUM1[dataHetero$SEX ==
+     0], scale = F))
R> y.m <- jitter(dataHetero$STABIL1[dataHetero$SEX ==
+     0])
R> x.f <- jitter(scale(dataHetero$KONFSUM1[dataHetero$SEX ==
+     1], scale = F))
R> y.f <- jitter(dataHetero$STABIL1[dataHetero$SEX ==
+     1])
R> plot(x.m, y.m, ylab = "Stabilität", xlab = "Konflikthäufigk
+     pch = 0, col = "red", xlim = c(min(x.m,
+         na.rm = T) - 1, max(x.m, na.rm = T) +
+         1))
R> points(x.f, y.f, pch = 2, col = "blue")
R> abline(lm(y.m ~ x.m), col = "red")
R> abline(lm(y.f ~ x.f), col = "blue")
```

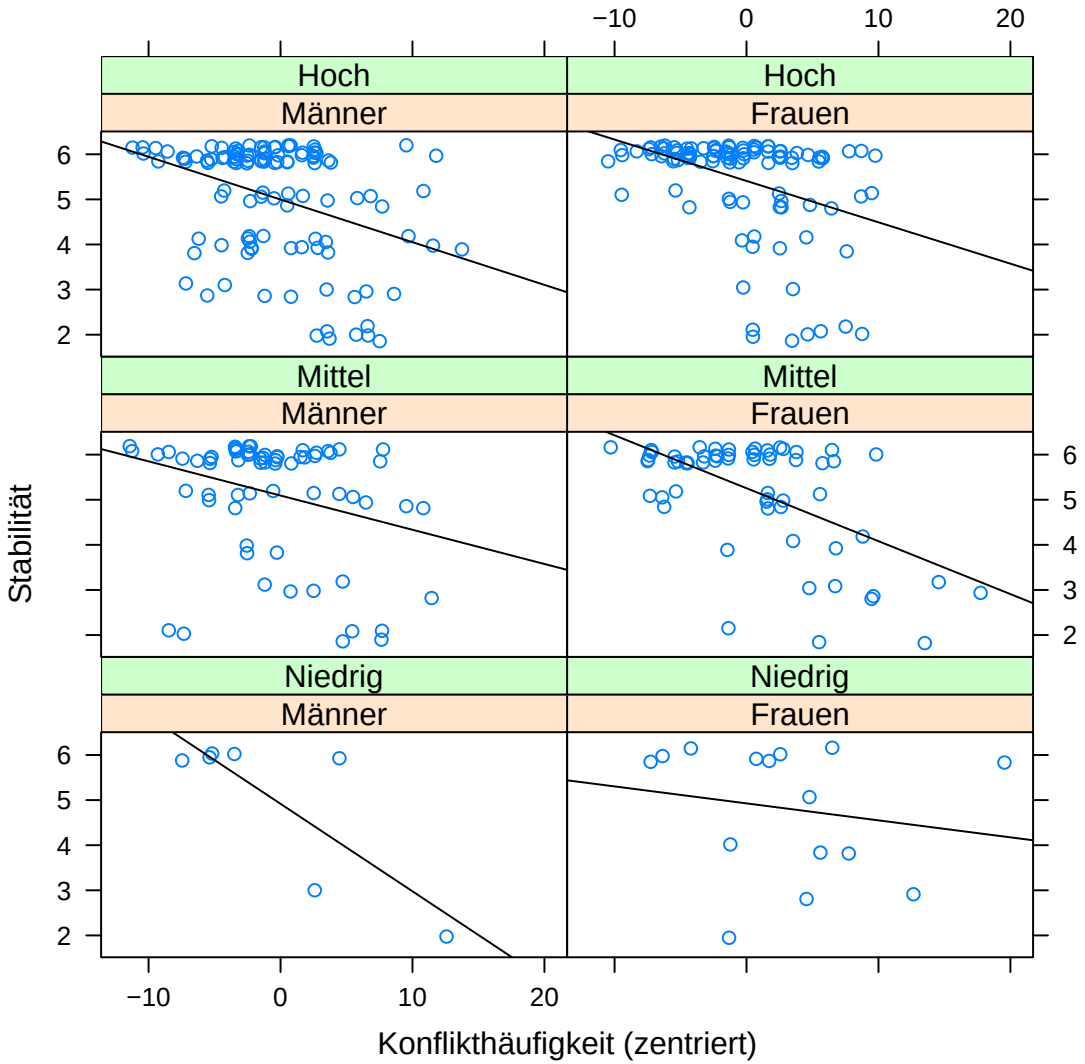
Partnerschaftsstabilität & Konflikthäufigkeit



Partnerschaftsstabilität & Konflikthäufigkeit

```
R> library(lattice)
R> print(xyplot(jitter(STABIL1) ~ jitter(scale(KONFSUM1,
+     scale = FALSE)) | funcLevelsSex(SEX),
+     data = dataHetero, jitter = TRUE,
+     ylab = "Stabilität", xlab = "Konflikthäufigkeit (zentrie
+     panel = function(x, y, ...) {
+         panel.xyplot(x, y)
+         panel.lmline(x, y)
+         panel.loess(x, y)
+     }))
```

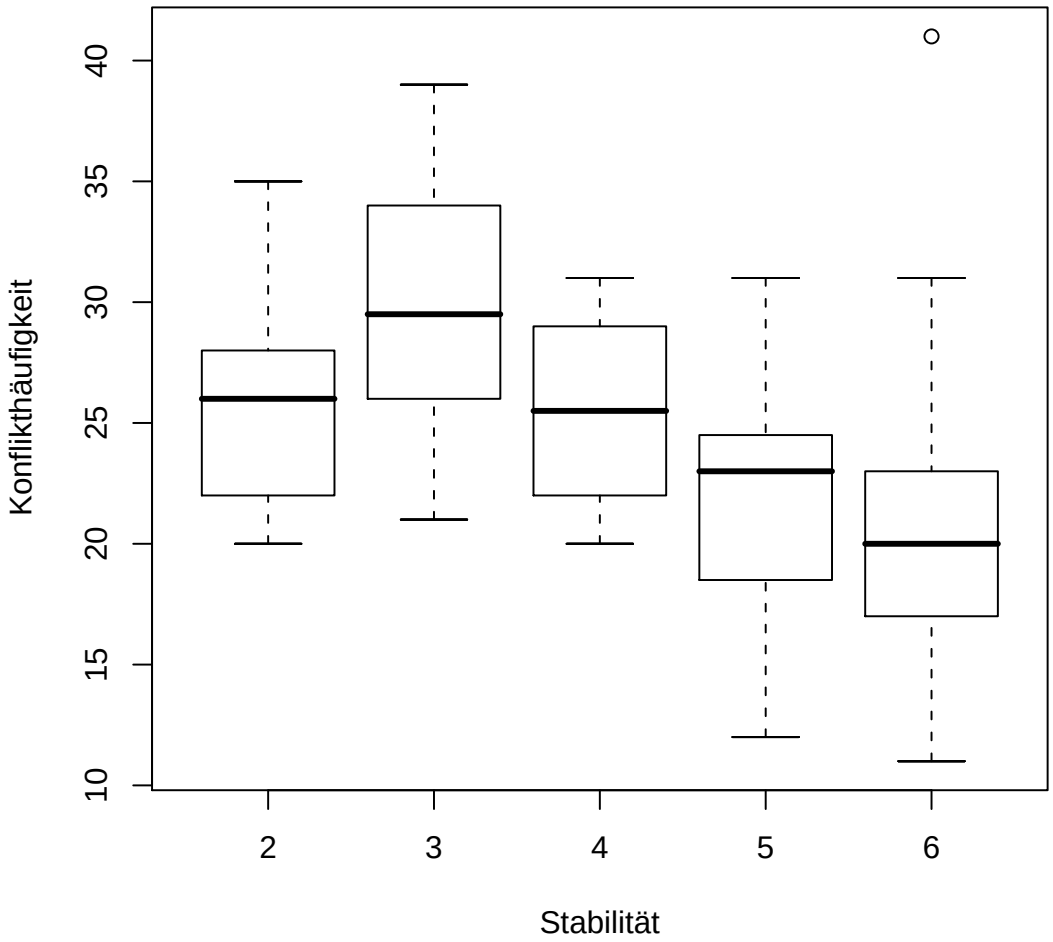
Partnerschaftsstabilität & Konflikthäufigkeit



Partnerschaftsstabilität & Konflikthäufigkeit

```
R> library(lattice)
R> print(xyplot(jitter(STABIL1) ~ jitter(scale(KONFSUM1,
+     scale = FALSE)) | funcLevelsSex(SEX) *
+     funcLevelsBildkat(BILDKAT), data = dataHetero,
+     jitter = TRUE, ylab = "Stabilität",
+     xlab = "Konflikthäufigkeit (zentriert)",
+     panel = function(x, y, ...) {
+         panel.xyplot(x, y)
+         panel.lmline(x, y)
+         panel.loess(x, y)
+     }))
```

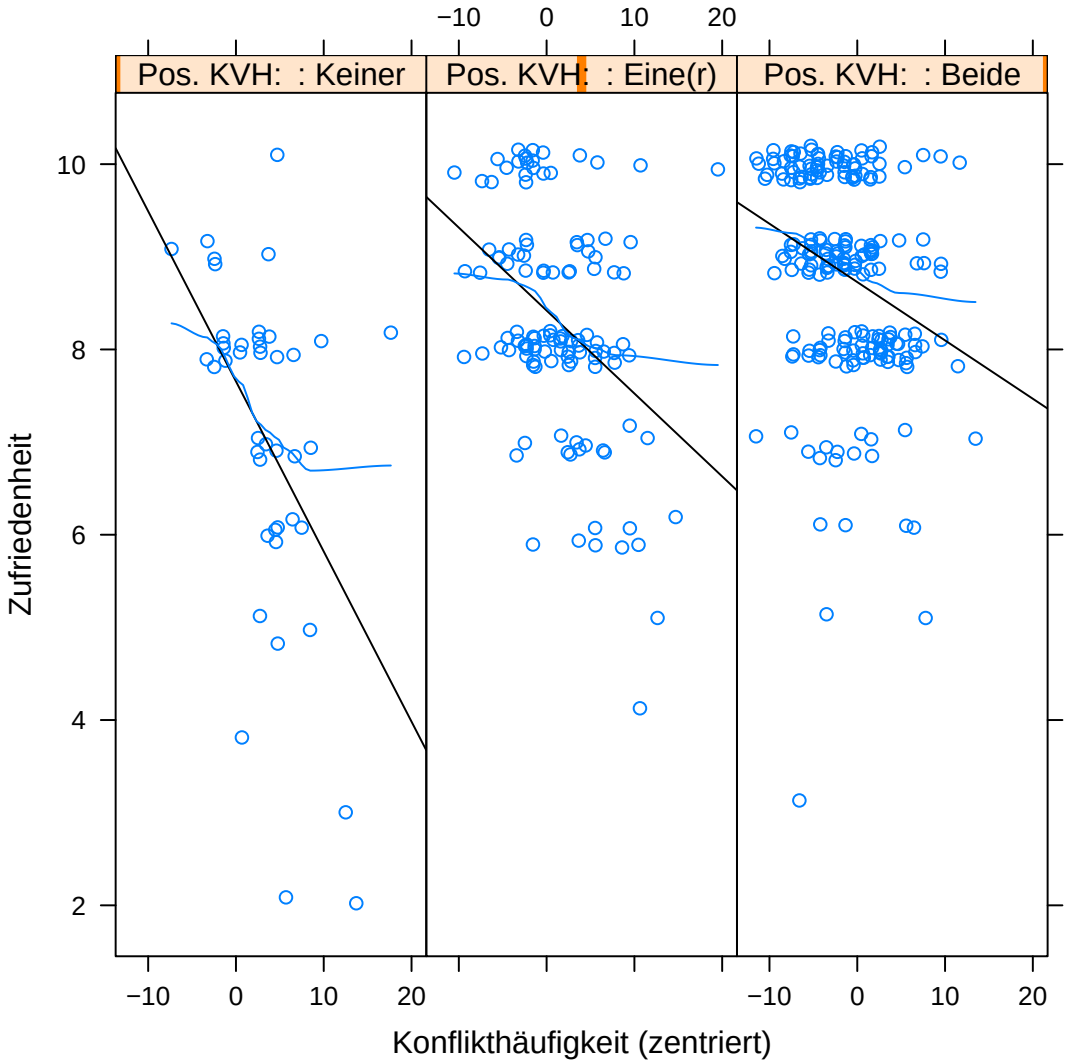
Partnerschaftsstabilität & Konflikthäufigkeit



Partnerschaftsstabilität & Konflikthäufigkeit

```
R> boxplot(dataHetero$KONFSUM1[dataHetero$SEX ==
+     1] ~ dataHetero$STABIL1[dataHetero$SEX ==
+     1], xlab = "Stabilität", ylab = "Konflikthäufigkeit")
```

Partnerschaftszufriedenheit & Konflikthäufigkeit



Übersicht

Die Formelnotation

- Die sogennante Formelnotation ist eine einfache Möglichkeit, ein statistisches Modell zu beschreiben. Dabei lautet die allgemeine Form: `y ~model`.
- `y` beschreibt die abhängige Variable, während im `model`-Term die unabhängigen Variablen spezifiziert werden.
- Die Variablen im `model`-Teil der Formel können unterschiedlich verknüpft werden:
  - $+$ Hinzunahme einer Variablen: `y ~x1 + x2`.
  - $-$ Ausschluss einer Variablen (-1 für Achsenabschnitt): `y ~-1 + x1`.
  - $*$ Interaktionen zwischen zwei Variablen; Haupteffekte werden automatisch einbezogen: `y ~x1 *x2` ist gleich `y ~x1 + x2 + x1 *x2`.
  - $:$ Nur die Interaktionsterme, ohne die Haupteffekte: `y ~x1:x2`.

OLS-Regression: Partnerschaftsstabilität & Konflikthäufigkeit (Frauen)

```
R> myData <- subset(dataHetero, !is.na(SEX) & !is.na(KONFSUM1) &
+     !is.na(STABIL1))
R> myData$breakvar <- 1
R> tmp <- aggregate(myData$breakvar, by = list(myData$PAARID), sum,
+     na.rm = T)
R> myData <- subset(myData, PAARID %in% tmp$Group.1[tmp$x == 2])
R> summary(modelF <- lm(STABIL1 ~ KONFSUM1, data = subset(myData,
+     SEX == 0)))

Call:
lm(formula = STABIL1 ~ KONFSUM1, data = subset(myData, SEX ==
    0))

Residuals:
    Min      1Q  Median      3Q     Max
-3.7769 -0.5544  0.4808  0.8244  1.9412

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  6.89362    0.38247  18.024  < 2e-16 ***
KONFSUM1    -0.08590    0.01783  -4.817 3.21e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.2 on 170 degrees of freedom
Multiple R-Squared: 0.1201, Adjusted R-squared: 0.1149
F-statistic: 23.21 on 1 and 170 DF,  p-value: 3.206e-06
```

OLS-Regression: Partnerschaftsstabilität & Konflikthäufigkeit (Männer)

```
R> summary(modelM <- lm(STABIL1 ~ KONFSUM1, data = subset(myData,
+     SEX == 1)))

Call:
lm(formula = STABIL1 ~ KONFSUM1, data = subset(myData, SEX ==
    1))

Residuals:
    Min      1Q  Median      3Q     Max
-3.4984 -0.2185  0.2218  0.6182  2.4606

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  7.36410    0.34155  21.561  < 2e-16 ***
KONFSUM1    -0.09329    0.01542  -6.052  8.9e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.049 on 170 degrees of freedom
Multiple R-Squared: 0.1772, Adjusted R-squared: 0.1724
F-statistic: 36.62 on 1 and 170 DF,  p-value: 8.893e-09
```

OLS-Regression: Partnerschaftsstabilität & Konflikthäufigkeit (Interaktion Geschlecht)

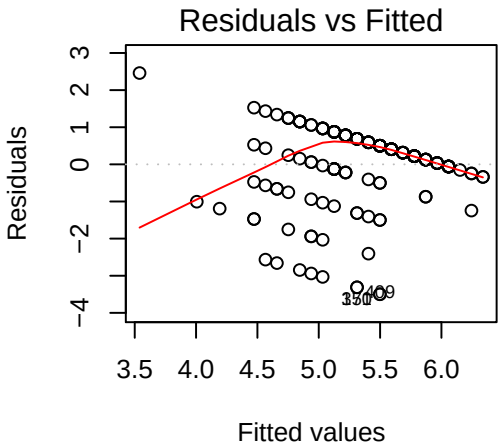
```
R> summary(modelI <- lm(STABIL1 ~ KONFSUM1 * funcLevelsSex(SEX),
+     data = dataHetero))

Call:
lm(formula = STABIL1 ~ KONFSUM1 * funcLevelsSex(SEX), data = dataHetero)

Residuals:
    Min      1Q  Median      3Q     Max
-3.7921 -0.4725  0.2985  0.7517  2.5523

Coefficients:
                                  Estimate Std. Error t value Pr(>|t|)
(Intercept)                       6.970457   0.340083  20.496  < 2e-16 ***
KONFSUM1                         -0.090641   0.015768  -5.749 1.87e-08 ***
funcLevelsSex(SEX)Frauen          0.430490   0.479840   0.897    0.370
KONFSUM1:funcLevelsSex(SEX)Frauen -0.005779   0.021896  -0.264    0.792
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.135 on 375 degrees of freedom
  (39 observations deleted due to missingness)
Multiple R-Squared: 0.1716, Adjusted R-squared: 0.165
F-statistic:  25.9 on 3 and 375 DF,  p-value: 3.021e-15
```

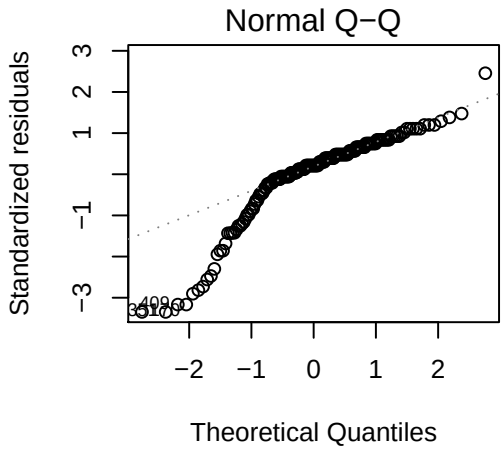
OLS-Regression: Partnerschaftsstabilität & Konflikthäufigkeit (Modellvergleich)

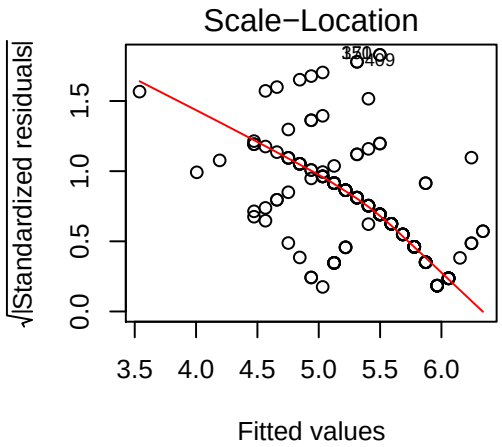
```
R> model0 <- lm(STABIL1 ~ 1, data = myData)
R> model1 <- lm(STABIL1 ~ KONFSUM1, data = myData)
R> model2 <- lm(STABIL1 ~ KONFSUM1 * funcLevelsSex(SEX), data = myData)
R> anova(model0, model1)

Analysis of Variance Table

Model 1: STABIL1 ~ 1
Model 2: STABIL1 ~ KONFSUM1
  Res.Df    RSS  Df Sum of Sq      F    Pr(>F)
1    343 510.86
2    342 440.35   1     70.51 54.761 1.065e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R> anova(model1, model2)

Analysis of Variance Table

Model 1: STABIL1 ~ KONFSUM1
Model 2: STABIL1 ~ KONFSUM1 * funcLevelsSex(SEX)
  Res.Df    RSS  Df Sum of Sq      F  Pr(>F)
1    342 440.35
2    340 431.78   2      8.57 3.3737 0.03541 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

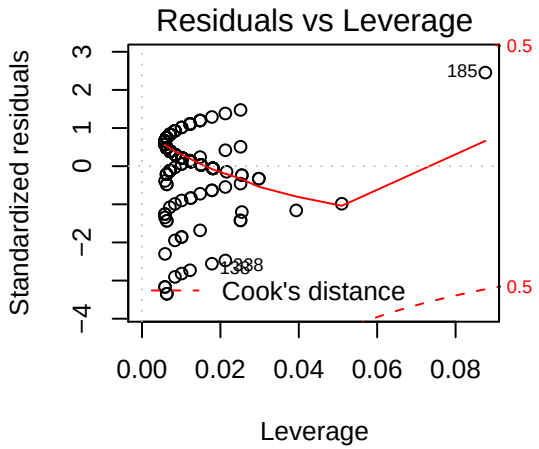
Generische Funktion zum plot von `lm`-Objekten

```
R> par(mfrow = c(2, 2))
R> plot(modelM)
```









MLM: Partnerschaftsstabilität & Konflikthäufigkeit (Interaktion Geschlecht)

```
R> dataDyadic$fSEX <- funcLevelsSex(dataDyadic$SEX)
R> summary(modelI <- lmer(STABIL1 ~ KONFSUM1 * fSEX + (1 | PAARID),
+     data = dataDyadic))

Linear mixed-effects model fit by REML
Formula: STABIL1 ~ KONFSUM1 * fSEX + (1 | PAARID)
   Data: dataDyadic
  AIC  BIC logLik MLdeviance REMLdeviance
 1188 1207 -588.8       1158         1178
Random effects:
 Groups   Name        Variance Std.Dev.
 PAARID   (Intercept) 0.63318  0.79572
 Residual             0.67536  0.82180
number of obs: 389, groups: PAARID, 213

Fixed effects:
                    Estimate Std. Error t value
(Intercept)          6.517289   0.305506  21.333
KONFSUM1            -0.069849   0.014070  -4.964
fSEXFrauen           0.490930   0.397762   1.234
KONFSUM1:fSEXFrauen -0.009386   0.018290  -0.513

Correlation of Fixed Effects:
            (Intr) KONFSUM1 fSEXFr
KONFSUM1    -0.964
fSEXFrauen  -0.641  0.636
KONFSUM1:SE  0.640 -0.665   -0.976
```

Übersicht multivariate Verfahren

- Allgemeines lineares Modell (OLS-Regressionen, ANOVAs in allen Varianten)
- Verallgemeinertes lineares Modell (logistische Regressionen, Zählvariablen-Modelle, etc.)
- Gemische Modelle (aka Mehrebenenmodelle)
- Bayesianische Regressionsmodelle
- Propensity Score Matching
- Event History Analysis
- ...

Übersicht

Eigene Funktionen

Mit Hilfe des Befehls `function()` lassen sich eigene Funktionen definieren. Die formale Befehlssyntax lautet:

```
meineFunktion <- function(argumentenListe){
  ## Folge von R Funktionen
  ## getrennt durch Semikolon oder Zeilenumbruch
}
```

```
R> t2r <- function(t, df) {
+     r <- sqrt(t^2/(t^2 + df))
+     return(r)
+ }
R> t2r(-2.33, 1628)
```

```
[1] 0.05765086
```

Kontrollstrukuren

- `if(cond){cons.expr}else{alt.expr}`
- `for(var in seq){expr}`
- `while(cond){expr}`
- `repeat{expr}`

Übersicht

Übersicht

Der Installationsprozess im Überblick

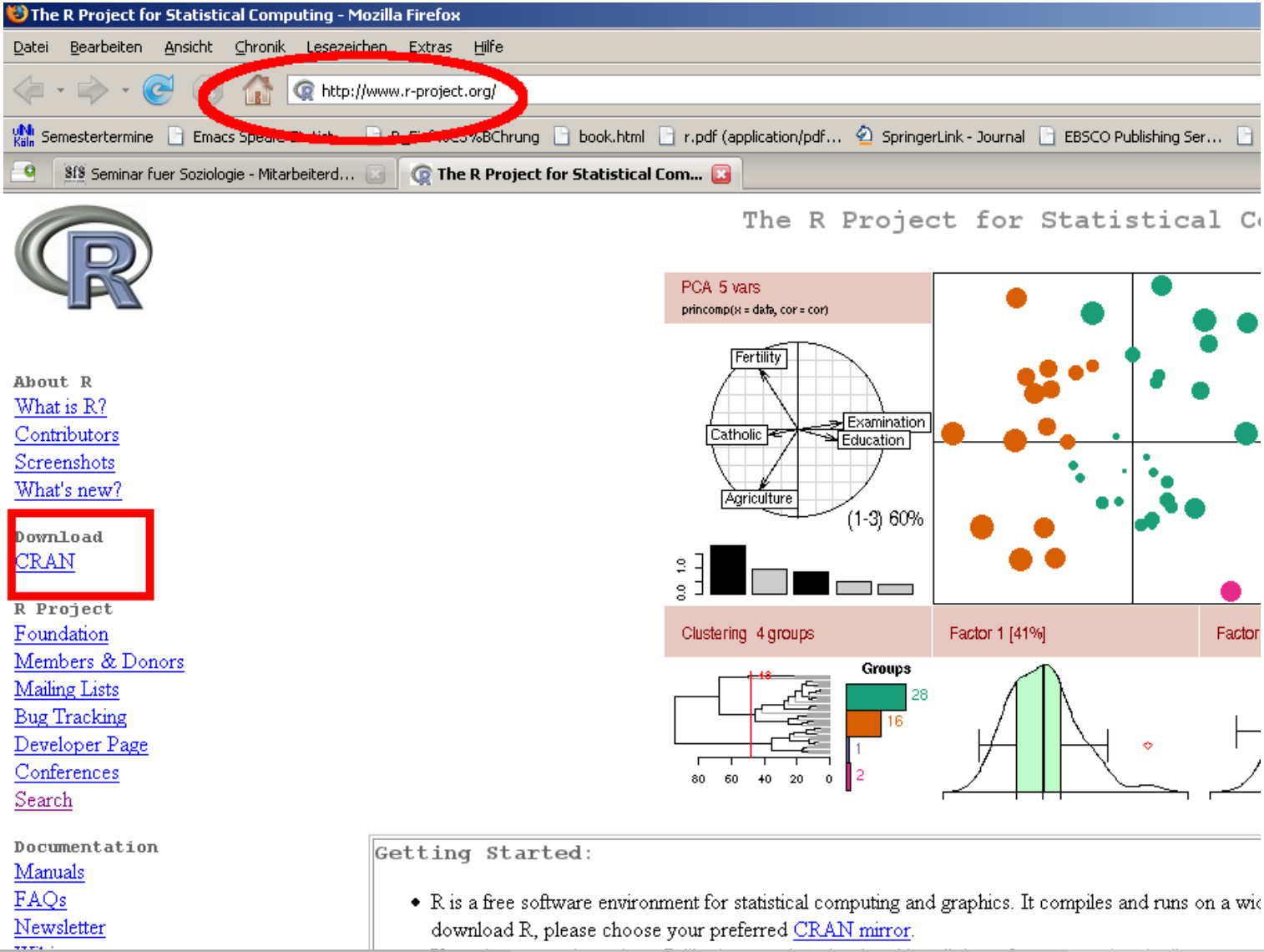
Um R auf dem heimischen Rechner zu installieren, sind zwei Schritte durchzuführen:

1. R aus dem WWW <http://www.r-project.org> herunterladen und
2. anschließend auf dem eigenen Rechner installieren.

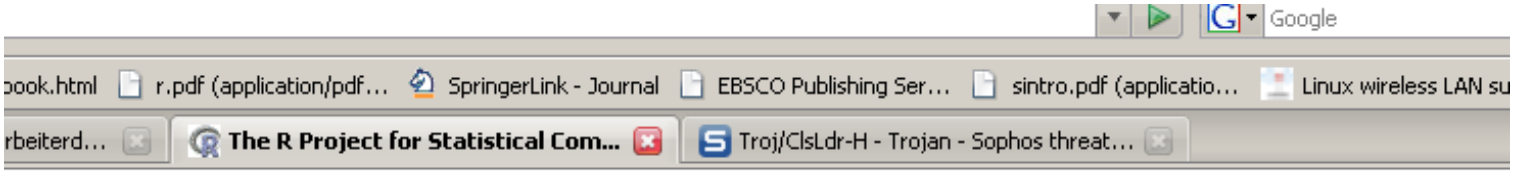
Benötigt wird auch der Editor TinnR <http://www.sciviews.org/Tinn-R/>.
Im WWW finden sich weitere Installationsanleitungen:

- Die offizielle „R for Windows FAQ“: <http://cran.r-project.org/bin/windows/base/rw-FAQ.html>.
- Reich bebildert ist die folgende Anleitung: <http://evfh-berlin.de/evfh-berlin/html/oe/mitarbeiter/R_installieren.pdf>.
- Eine Anleitung zur Installation von TinnR: <www.zhwin.ch/~lor/R/InstallationUndErsteSchritteMitTinnR.pdf>.

Schritt 1: Aufruf der R-Homepage <http://www.r-project.org>



Schritt 2: Auswahl des CRAN Mirrors

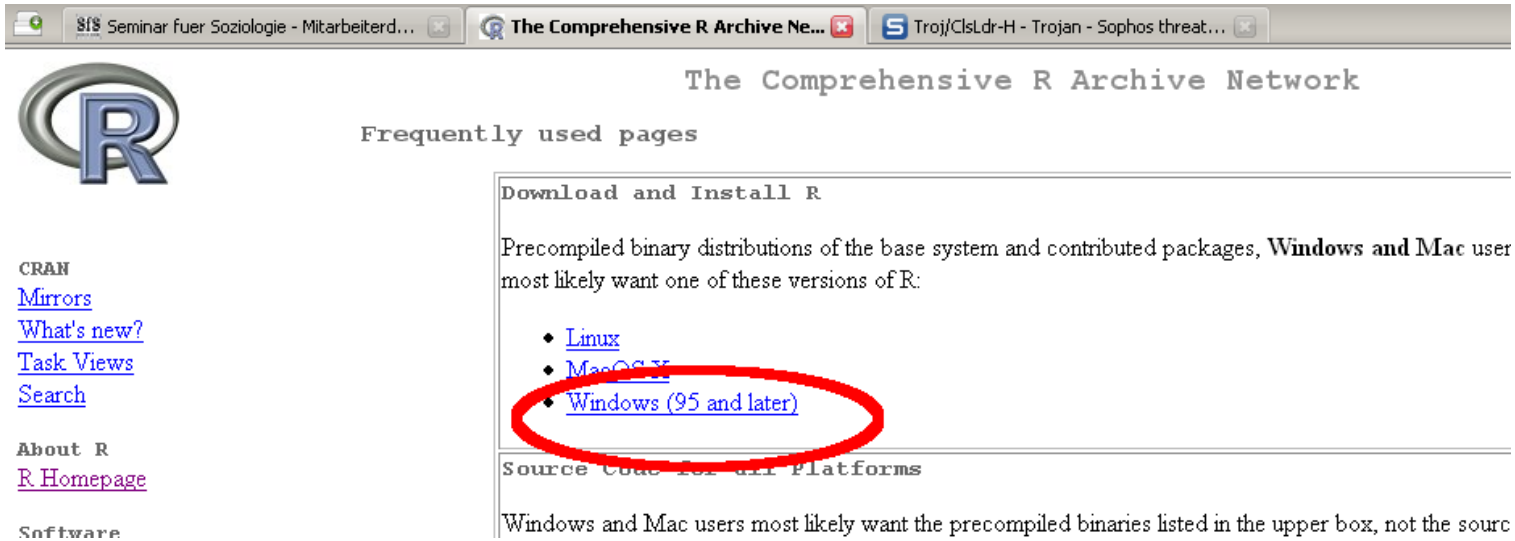


CRAN Mirrors

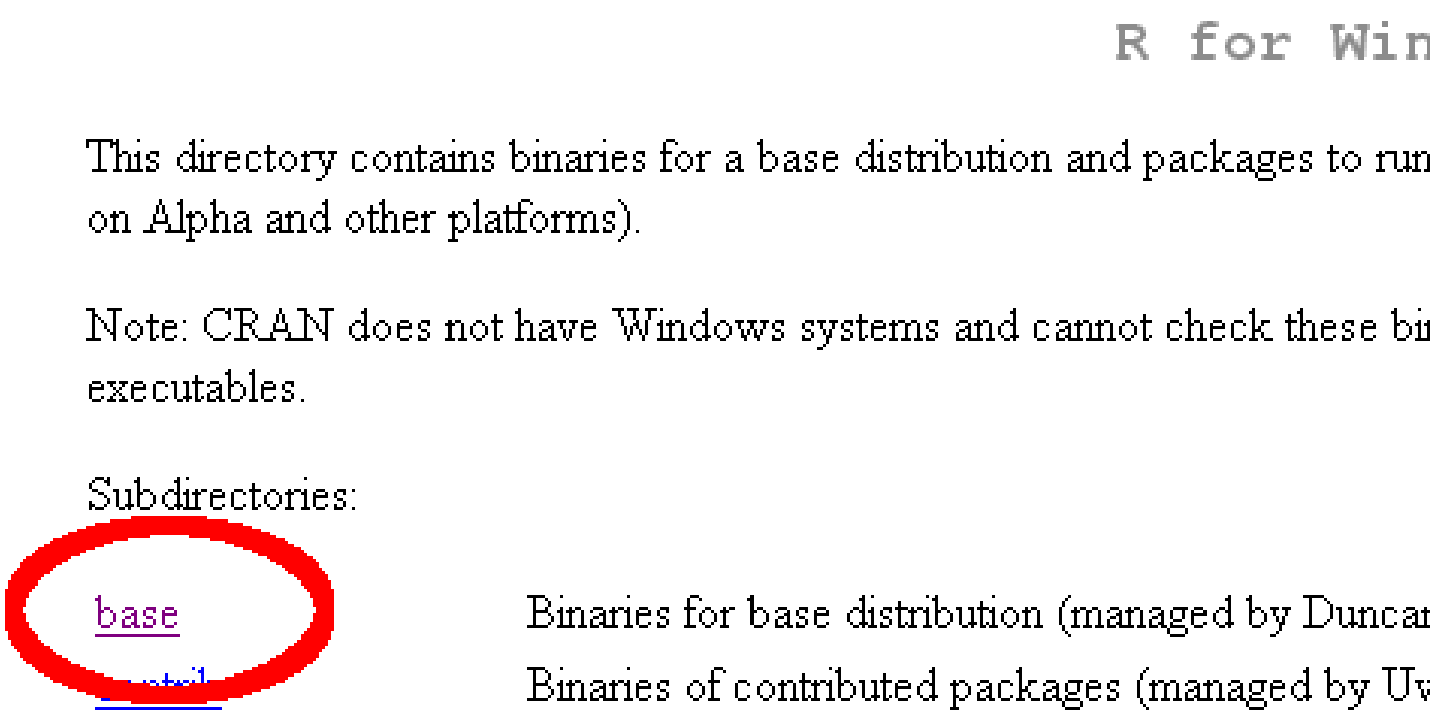
The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you:

Australia	
http://cran.au.r-project.org/	PlanetMirror, Brisbane
http://cran.ms.unimelb.edu.au/	University of Melbourne
Austri	
http://cran.at.r-project.org/	Wirtschaftsuniversitaet Wien
Brazi	
http://cran.br.r-project.org/	Universidade Federal do Parana
http://www.insecta.ufv.br/CRAN/	Federal University of Vicosa
http://cran.fiocruz.br/	Oswaldo Cruz Foundation, Rio de Janeiro
http://lmq.esalq.usp.br/CRAN/	University of Sao Paulo, Piracicaba
http://www.vps.fmvz.usp.br/CRAN/	University of Sao Paulo, Sao Paulo
Canada	
http://cran.stat.sfu.ca/	Simon Fraser University, Burnaby
http://probability.ca/cran/	University of Toronto
Chile	

Schritt 3: Auswahl des Betriebssystems



Schritt 4: Auswahl der Grundinstallation



R for Win

This directory contains binaries for a base distribution and packages to run
on Alpha and other platforms).

Note: CRAN does not have Windows systems and cannot check these bi
executables.

Subdirectories:

base — Binaries for base distribution (managed by Duncar

contrib — Binaries of contributed packages (managed by Uv

Please do not submit binaries to CRAN. Package developers might want t
questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Last modified: April 4, 2004, by Friedrich Leisch

Schritt 5: Auswahl der R-Version

R-2.4.1 for Windows

This directory contains a binary distribution of R-2.4.1 to run on Windows 95, 98, ME, NT4.0, 2000 and XP on Intel/clone chips.

A build of the test version (which will become R 2.5.0 around April 24) is available here. **Please contribute by testing this version and reporting any problems!**

Patches to R 2.4.1 are incorporated in the r-patched snapshot build.

A build of the development version (which will eventually become R 2.6.0 in the fall) is available in the R-devel snapshot build.

In this directory:

README.R-2.4.1	Installation and other instructions.
CHANGES	New features of this Windows version.
NEWS	New features of all versions.
R-2.4.1-win32.exe	Setup program (about 28 megabytes). Please download this from a mirror near you. This corresponds to the file named **SetupR.exe** or **rwXXXX.exe** in pre-2.2.0 releases.
old	The previous release.
md5sum.txt	md5sum output for the setup program. A Windows GUI version of md5sum is available at http://www.md5summer.org/, a Windows command line version is available at http://www.etree.org/md5com.html.

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information, including upgrade advice.

Note to webmasters: A stable link which will redirect to the current Windows binary release is <CRAN MIRROR>/bin/windows/base/release.htm.

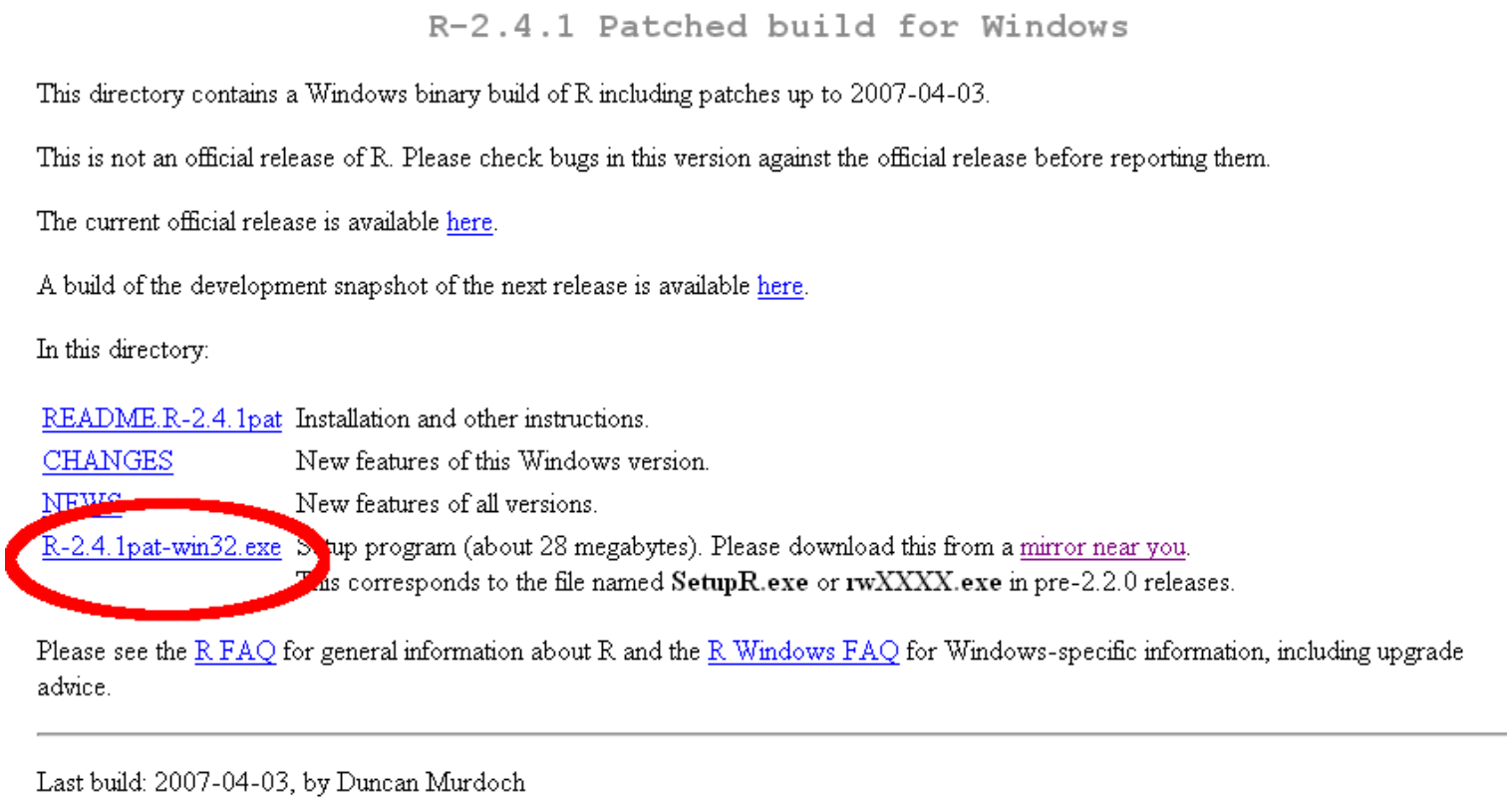
Last change: 2006-12-18, by Duncan Murdoch

Warum gibt es verschiedene R-Versionen?

R gibt es in verschiedenen Versionen:

- Neben der jeweils aktuellen Version (zur Zeit ist das „R-2.4.1 for Windows“) findet sich
- auch eine „patched“, das heißt, um Fehler bereinigte Version („R-2.4.1 Patched build for Windows“) auf CRAN.
- Schließlich gibt es noch eine in Entwicklung befindliche, mehr oder weniger stabile Version („R-2.5.0 alpha build for Windows“). Diese sollte nicht benutzt werden!

Schritt 6: Auswahl der Installationsdatei



R-2.4.1 Patched build for Windows

This directory contains a Windows binary build of R including patches up to 2007-04-03.

This is not an official release of R. Please check bugs in this version against the official release before reporting them.

The current official release is available here.

A build of the development snapshot of the next release is available here.

In this directory:

README.R-2.4.1pat	Installation and other instructions.
CHANGES	New features of this Windows version.
NEWS	New features of all versions.
R-2.4.1pat-win32.exe	Setup program (about 28 megabytes). Please download this from a mirror near you. This corresponds to the file named **SetupR.exe** or **rwXXXX.exe** in pre-2.2.0 releases.

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information, including upgrade advice.

Last build: 2007-04-03, by Duncan Murdoch

R lokal installieren

1. Installationsdatei (`R-2.4.1pat-win32.exe`) aufrufen.
2. Sprache der Installationsroutine auswählen („Weiter“ klicken).
3. Es erscheint die Nachricht „Willkommen zum R for Windows Setup-Assistenten“ („Weiter“ klicken).
4. Es erscheint die Lizenz (GNU GENERAL PUBLIC LICENSE) unter der R vertrieben wird („Weiter“ klicken).
5. Den „Ziel-Ordner wählen“, z.B. `C:/Programme/R` (die Versionsangabe `R-2.4.1pat`) kann wegfallen (benötigter Speicher etwa 1,4 GB).
6. Alle Komponenten auswählen; in jedem Kästchen ein Haken!
7. Startoptionen anpassen? „Nein“ auswählen.
8. Der „Startmenü-Ordner“ sollte nicht geändert werden.
9. In „Zusätzliche Aufgaben auswählen“ bitte das Kästchen „Verknüpfe R mit .RData Dateien“ aktivieren. Nach dem Klick auf „Weiter“ beginnt die Installation.

Übersicht

R mit Paketen erweitern

- R ist modular aufgebaut und neue Funktionen können durch sogenannte *packages* hinzugefügt werden.
- In der Grundinstallation von R sind bereits 12 Basis-Pakete enthalten (*base*, *graphics*, *datasets* etc.). Hinzu kommen 13 empfohlene Pakete (*boot*, *foreign*, *survival* etc.).
- Viele Pakete liegen auf einem der Server des CRAN (Comprehensive R Archive Network).
- Zur Zeit (20. März 2007) sind etwa 1000 Pakete verfügbar[2], diese belegen etwa $1,4\,GB$ an Festplattenspeicher.

[2] Information erhältlich durch `dim(available.packages())`.

Pakete verwalten und installieren

Pakete installieren

- Neue Pakete installieren: `install.packages("packageName")` (Anführungsstriche!).
- Alle verfügbaren Pakete installieren: `install.packages(new.packages())`, danach einen möglichst nahen CRAN-Server auswählen.

Pakete laden und aktualisieren

- Bevor Funktionen aus Paketen genutzt werden können, müssen sie „geladen“ werden: `library(packageName)` (keine Anführungsstriche)[a].
- Vorhandene Pakete mit `update.packages()` aktualisieren.

[a] „Entladen“, d.h aus dem Suchpfad entfernen: `detach("package:packageName")`

Übersicht

Editoren

Eine allgemeine Übersicht findet sich unter:
<http://www.sciviews.org/_rgui/projects/Editors.html>.

- TinnR <http://www.sciviews.org/Tinn-R/>
- WinEdt <http://www.winedt.com/> und
- Emacs <http://www.gnu.org/software/auctex/download-for-windows.html> + ESS (Emacs speaks statistis) <http://ess.r-project.org/>

Übersicht

R im Zusammenspiel mit anderer Software

- Einbinden von C++ oder Fortran-Code
- Reporting mit LaTeX, MS-Word, OpenOffice etc.
- MS-Excel + R
- R auf einem Web-Server
- ...

Quellen für diesen Vortrag / Literatur

- Grüner, Erwin, 2007: „Statistik mit R“ <http://www.staff.uni-marburg.de/~gruener/lehre/r.w05/>.
- Ligges, U. (2007). Programmieren mit R. Berlin/Heidelberg, Springer.
- ...