

# Eine Einführung in R: Grundlagen II

**Bernd Klaus, Verena Zuber**

Institut für Medizinische Informatik, Statistik und Epidemiologie (IMISE),  
Universität Leipzig

<http://www.uni-leipzig.de/zuber/teaching/ws11/r-kurs/>

20. Oktober 2011

Empfehlenswerte frei verfügbare Literatur:  
(Links auf der Homepage!)

- Wim Krijnen, 2009: *Applied Statistics for Bioinformatics Using R*
- Andrew Robinson, 2008: *IcebreakR*
- W. N. Venables, D. M. Smith and the R Development Core Team, 2009: *An Introduction to R*
- Tom Short, 2004: *R reference card*
- Dormann, C.F., I. Kühn. 2008: *Statistische Analyse biologischer Daten*

Editor:

- tinn-R: *<http://www.sciviews.org/Tinn-R/>*

- ① I. Grundlegendes zu R: Datenim- und Export
- ② II. Grundlegendes zu R: Wichtige Funktionen zum Datenumgang
  - Ordnen und Sortieren
  - Die Apply-Funktionen
  - Die Abfrage mittels which

# Das .csv Format

- Wir verwenden hier das sogenannte *.csv* – Format, ein einfaches und weitverbreitetes Textformat
- Bsp.: *Patienten.csv* vom ersten Übungsblatt (Der Eintrag NA wurde durch 60.0 ersetzt.)

	Groesse,	Gewicht
P1,	1.65,	80.0
P2,	1.30,	60.0
P3,	1.20,	50.0

- Einträge einer Zeile werden durch `,` getrennt, Leerzeichen kennzeichnen fehlende Werte

- Mit `getwd()` bzw. `setwd()` lässt sich das aktuelle Arbeitsverzeichnis abfragen bzw. setzen
- `list.files("/data")` listet alle Dateien auf, die im Unterverzeichnis "data" des Arbeitsverzeichnisses liegen
- Mit `read.csv()` lässt sich eine Datei im `.csv` – Format einlesen
- Bsp.: `pat <- read.csv(file = "C://path/to/filename/Patienten.csv")`  
(absoluter Pfad) bzw.
- `pat <- read.csv(file = "/data/Patienten.csv")`  
(relativer Pfad)
- Exportieren entsprechend mit `write.csv()`

Allgemeiner ist der `read.table` bzw. `write.table` Befehl:

```
read.table(file, header = FALSE, sep = "", dec =  
".", row.names, col.names, na.strings = "NA", ... )
```

- `file`: der Datensatz, bzw. Pfad zu dem Datensatz
- `header`: enthält der Datensatz Variablennamen in der ersten Zeile?
- `sep`: Trennungszeichen
- `dec`: Dezimalzeichen
- `row/col.names`: die gewünschten Namen der Zeilen bzw. Spalten
- `na.strings`: Bezeichnung der fehlenden Werte

## Weitere Befehle zum Dateneinlesen

R bietet eine Vielzahl von Möglichkeiten Daten ein- und auszulesen:

- `read.table`: der allgemeine Befehl
- `read.csv`: Einlesen von `.csv` Datenformaten
- `read.delim`: Einlesen von Datenformaten, die *tabs* verwenden
- `read.fwf`: Einlesen von besonders formatierten Daten
- `scan`: Einlesen von Daten in Vektoren oder Listen

# Ordnen und Sortieren

- `sort`: Sortiert eine Vektor in aufsteigender Ordnung
- `order`: Ordnet die Indices so, dass die Elemente des Vektors sortiert werden, d.h. es gilt `sort(V) = V[order(V)]`
- `rank`: Verteilt Ränge, wobei es verschiedene Optionen gibt, gleiche Werte (*ties*) zu behandeln

Beispiel: Der Anteil an Kinder in einem armen Elternhaus (`Armut`)

- `sort(Armut)`: 2.7 4.0 4.2 4.6 ... 22.2 24.6
- `sort(Armut, decreasing=TRUE)`: 24.6 22.2 ... 4.0 2.7
- `Armut[order(Armut)]`: 2.7 4.0 4.2 4.6 ... 22.2 24.6
- `rank(Armut)[order(Armut)]`:  
1.0 2.0 3.0 ... 22.0 23.5 23.5 25.0 ... 30.0



# Die Apply-Funktion

R ermöglicht es mit der Funktion `apply()` eine Funktion auf jede Zeile, bzw Spalte einer Matrix anzuwenden.

```
apply(X, MARGIN, FUN, ...)
```

- **X**: die entsprechende Matrix
- **MARGIN** = 1 (-> Zeilenweise)
- **MARGIN** = 2 (-> Spaltenweise)
- **FUN**: die gewünschte Funktion, z.B. `mean`, `var`, ...

# Apply-Funktionen

- `apply( )` eine Funktion spalten- oder zeilenweise auf eine Matrix anwenden
- `sapply( )` eine Funktion spaltenweise auf einen Datensatz anwenden
- `mapply( )` multivariate Variante von `sapply( )`, mehrere Datensätze können übergeben werden
- `tapply( )` eine Funktion auf Daten anwenden, die zu bestimmten Faktorgruppen gehören

Fehlende Werte ( `NA` ) können bei den Apply-Funktionen mittels der Option `na.rm = TRUE` herausgenommen werden (sonst gibt R in der Regel eine Fehlermeldung zurück).

# Beispiele

Betrachten wir unsere Patientendaten:

	Groesse,	Gewicht
P1,	1.65,	80.0
P2,	1.30,	60.0
P3,	1.20,	50.0

Berechnung des Mittelwert für jede Spalte in diesem Datensatz:

```
sapply(X = pat[,1:2], FUN = mean)
  Groesse Gewicht
 1.383333 63.333333
```

Betrachten wir eine erweiterte Tabelle unserer Patientendaten:

	Groesse,	Gewicht,	Ge
P1,	1.65,	80.0,	"w"
P2,	1.30,	60.0,	"m"
P3,	1.20,	50.0,	"w"

- Zu den bekannten Daten ist ein Faktor **Ge** hinzugekommen.
- Berechnung der durchschnittlichen Größe getrennt nach dem Faktor Geschlecht:

```
tapply(X = pat$Groesse, FUN = mean,
       INDEX = pat$Ge)
```

Die Ausgabe ist:

```
      m w
1.3 1.425
```

# Die Abfrage mittels which

Will man auf Beobachtungen mit einer bestimmten Eigenschaft zugreifen, so bietet sich die Abfrage mittels `which()` an:

`which(Bedingung)`

- **Bedingung**: logischer Vektor, Bsp. `Variable == Wert`
- Ergebnis: ein Vektor mit den Stellen, an denen die Bedingung "wahr" ist
- Um direkt auf die Elemente zuzugreifen:  
`Variable[which(Bedingung)]`

# Einschub: Logische Variablen

- `Variable == Wert`: Gleichheit
- `Variable != Wert`: Ungleichheit
- `Variable < Wert`: kleiner
- `Variable > Wert`: größer
- `&`: *und*
- `|`: *oder*
- `!`: Negation
- `%in%`: Überprüfung auf Elementeigenschaft

## Beispiele

Betrachten wir wieder unsere Patientendaten:

	Groesse,	Gewicht
P1,	1.65,	80.0,
P2,	1.30,	60.0,
P3,	1.20,	50.0,

- In welchen Zeilen sind Beobachtungen, die kleiner 1.5 sind?

```
which(Groesse<1.5)
2 3
```

- Wie klein sind die Patienten, die kleiner als 1.5 sind?

```
Groesse[which(Groesse<1.5)]
1.3 1.2
```